La 3D maillée avec Blender et POV-Ray

Jean-Luc Ancey

15 août 2007

Chapitre 1

Introduction

Quand j'ai commencé à m'intéresser à la maquette 3D informatisée, c'était avec la conviction naïve que ce serait forcément plus simple que la maquette en carton. Je venais tout juste de découvrir à quel point il est plus facile de rédiger avec un traitement de texte qu'avec une plume, et j'imaginais qu'il était nécessairement plus aisé d'élaborer une maquette en 3D avec un ordinateur plutôt qu'avec des ciseaux et de la colle.

Il me semblait que la possibilité de travailler dans un univers virtuel et immatériel ne pouvait manquer de simplifier radicalement l'exercice. Plus besoin de réserver de la place sur une grande table pendant des mois (ni d'interdire au chat d'y faire la sieste). Plus besoin de casser ma tirelire pour acheter à prix d'or des bouts de plastique à messieurs Jouef, Airfix et Heller. Plus de risque de pester contre la disproportion entre ma gare de chemin de fer (Jouef, échelle HO) et les personnages collés sur le quai (Airfix, échelle HO également, ce qui ne les empêchait pas d'être trop grands d'un bon tiers). Plus besoin de tirer des rallonges pour que toutes les lampes de bureau et de chevet de la maison éclairent médiocrement des éléments minuscules (colorés avec goût en marron sombre par monsieur Heller). Plus besoin de fouiller le sac à poussière de l'aspirateur quand un de ces petits fourbis manque à l'appel. Plus besoin de me disputer avec le chat quand l'antenne radar du porte-avions a visiblement été mordillée avec application. Plus besoin de m'user les yeux sur des schémas de montage couverts de chiffres allant de 1 à 85 et seulement expliqués par quelques lignes en corps 6. Plus besoin de supplier un prof d'allemand de me traduire ces lignes (il est vrai que j'aurais pu me rabattre sur la version originale en japonais). Plus besoin de dissimuler à ma mère les dégâts causés au tapis et à la table du salon par les pots de peinture renversés et les tubes de colle écrasés (et pendant que j'essayais de nettoyer le désastre, ces produits aussi coûteux que malodorants devenaient durs comme du bois en moins de temps qu'il n'en faut pour l'écrire). Plus besoin d'avoir recours à la méditation zen pour cesser de trembloter de rage en voyant cette maudite colle baver toujours

exactement à côté du trou où je voulais la faire pénétrer...

Ah non, vraiment, avec l'informatique, ça ne pouvait pas manquer d'être plus facile (et accessoirement, moins cher). Je m'enthousiasmais d'avance devant la possibilité de travailler sur des scènes de plusieurs kilomètres de large, la probable facilité pour démultiplier les poteaux électriques et les réverbères (hop ! copier-coller !) ou pour les disposer en ligne avec une parfaite régularité (il faut tellement moins d'habileté pour faire des additions que pour se servir proprement d'un tube de colle), ou encore pour intégrer des corrections de détail sans devoir détacher des éléments minuscules et fragiles qu'on avait eu tant de mal à faire tenir en place...

Je reste persuadé comme au premier jour que l'apprentissage de la 3D informatisée ne devrait pas prendre plus de quelques heures – par exemple celles que vous avez décidé de consacrer à la lecture de ce livre. Il n'empêche que ma propre initiation m'a demandé une quinzaine d'années. A ma profonde stupéfaction, je me rends compte que les recherches de dilettante que j'ai menées en la matière m'ont pratiquement transformé en informaticien. J'ai appris à manipuler et parfois même à comprendre tout un vocabulaire ésotérique (pointeurs, librairies, instances, transtypage...) qui m'est très pratique pour impressionner les profanes – mais qui à vrai dire n'a presque aucune utilité pour l'ambition toute simple que je poursuivais : réaliser sur l'écran d'un ordinateur ce que je pensais savoir faire avec du carton, des ciseaux, de la colle et de la peinture.

Pourquoi ai-je fait l'effort d'apprendre autant de choses inutiles ? Essentiellement, je crois, parce que je n'ai jamais pu trouver nulle part un abécédaire de la 3D informatisée du type de celui que vous avez entre les mains. Il existait certes des manuels d'initiation à tel ou tel logiciel de CAO (généralement beaucoup trop coûteux pour que je me les offre). Mais moi, je ne m'intéressais pas du tout à la Konception Hassistée par Hordinateur, je n'avais aucune ambition de devenir un infographiste professionnel... Je voulais juste réaliser rapidement de petites maquettes de rien du tout : des tables, des chaises, des escaliers – et dans mes rêves les plus fous, des petites maisons, voire des gares de chemin de fer. Je pensais – et j'avais raison – qu'il devait être possible d'y arriver sans investir trois mois de salaire dans un logiciel coûteux ni surtout six mois d'études pour parvenir à y comprendre quelque chose. Il me fallait juste apprendre quelques bases – le nécessaire pour informatiser des bouts de carton, en somme, et ça ne devait pas être bien sorcier.

J'avais raison, ça ne l'était pas, et ce livre le démontre. Mais les manuels d'informatique auxquels j'ai eu recours, eux, se contrefichaient de la maquette en carton. Les quelques rares informations qui m'étaient réellement utiles pour transposer à l'écran de l'ordinateur les techniques de la maquette classique étaient certes présentes dans ces ouvrages, mais toujours sous des intitulés ésotériques apparemment dépourvus de rapport avec la question, et perdues au milieu de di-

zaines d'autres – certainement utiles aux informaticiens, mais pas à moi. Dans ma quête désespérée du moyen d'informatiser des bouts de carton, j'ai donc dû me familiariser avec quantité de techniques pour la gestion des bases de données, les connexions sécurisées, les sauvegardes incrémentales, les ports d'entrée-sortie... toutes choses absolument sensationnelles sans doute, mais dont je n'avais pas, et dont je n'ai toujours pas, le moindre usage.

Avec le recul, je suis stupéfait que cela m'ait amené à acquérir un niveau de compétence informatique somme toute assez honnête alors que rien ne m'y prédisposait – et surtout pas mes goûts. J'ai un peu le sentiment d'avoir gâché ma belle jeunesse à lire des grimoires techniques. J'enrage surtout parce que les quelques connaissances réellement utiles pour ce que je me proposais de faire (que j'ai mis des années à identifier au milieu de tout ce fatras) auraient pu être résumées en quelques dizaines de pages ; celles que vous tenez entre vos mains.

Ce livre n'est pas un mode d'emploi des logiciels auquel il a recours (Blender, POV-Ray, Python), en tout cas pas un mode d'emploi complet. Je ne dirai pratiquement pas un mot de leurs fonctions les plus exotiques (marbrures, transparences, particules...). Je n'essaierai pas de vous faire connaître ces paquetages dans leurs moindres recoins, je me contenterai de vous expliquer comment en tirer le meilleur usage après un apprentissage de quelques semaines, voire de quelques heures seulement. Je chercherai encore bien moins à vous convaincre que ces outils sont les meilleurs du monde – d'autant qu'à mes yeux, ils sont bien souvent inutilement complexes. J'apprécie surtout en eux le fait qu'on peut en utiliser en toute légalité des copies réalisées sans bourse délier. Peu m'importe d'ailleurs qu'ils fassent mieux ou moins bien que leurs concurrents commerciaux : il me suffit qu'ils soient largement assez puissants pour gérer sur ordinateur des maquettes précises et détaillées. Au reste, ce que je serai amené à dire à leur sujet peut sans difficulté être transposé à leurs concurrents commerciaux. L'ambition de ce livre est seulement de vous permettre de comprendre qu'on peut réaliser avec eux (comme d'ailleurs avec leurs concurrents), sans grande difficulté, des maquettes aussi jolies, détaillées et précises que celles que tout un chacun saurait réaliser avec du carton, des ciseaux, de la colle et des pots de peinture, et au prix d'un effort intellectuel tout aussi sommaire. Pour moi, la maquette en 3D reste tout à fait similaire à la maquette en carton – à ceci près qu'elle est plus facile à mettre en œuvre, plus souple, plus exacte, moins chère, et vraiment beaucoup plus simple à transformer en films d'animation !

Ceci est le livre d'un amateur qui a conscience d'avoir perdu quinze ans à étudier la technique de la 3D informatisée dans ses moindres recoins, alors qu'il cherchait seulement un équivalent informatique aux ciseaux, à la colle et aux pots de peinture. Il promet à ses lecteurs qu'ils arriveront à ce résultat en quelques semaines, voire en quelques heures, pourvu qu'ils fassent l'effort de se coltiner quelques notions techniques à la portée d'un enfant de douze ans. Après quoi vous

pourrez concevoir vos propres projets en 3D sans rester bloqué devant ces tristes écrans informatiques. Au contraire, vous travaillerez pour l'essentiel comme il convient, c'est-à-dire tout simplement dans votre tête, dans le noir paradis de l'espace euclidien, ou à l'extrême rigueur avec un crayon, sur un petit carnet ou une feuille de papier à dessin.

Rambouillet, le 24 mai 2005.

Première partie

Les bases

Chapitre 2

Configurer sa machine pour la 3D

2.1 Philosophie d'utilisation

Les manipulations décrites dans ce livre sont principalement effectuées avec les logiciels Blender et POV-Ray – et certaines opérations particulières seront en outre facilitées par un usage élémentaire du langage Python. Ces trois logiciels sont largement diffusés sur le web et disponibles gratuitement. Cela plaide assurément en leur faveur, mais ce n'est pas la raison principale de leur choix.

L'auteur a décidé de les employer avec l'ambition de définir une méthode de travail aussi universelle et pérenne que possible, qui ne se limite pas à une configuration informatique précise et ne coure pas le risque d'être rapidement rendue obsolète par les évolutions techniques inéluctables. Les projets 3D sont en effet très susceptibles de faire travailler des équipes entières pendant des mois et même des années, et le risque d'une remise en cause des méthodes de travail doit être réduit au minimum, sinon même éliminé.

L'emploi de Blender, POV-Ray et Python est d'emblée assez rassurant dans cet esprit, car aucun d'entre eux n'est prisonnier d'une configuration informatique précise. En témoigne le fait qu'ils fonctionnent tous les trois de façon très similaire sur des ordinateurs aussi différents que des PC sous Windows (98, ME, 2000 ou XP, indifféremment), des Macintosh (sous Mac OS/X) ou même des machines tournant sous le système d'exploitation libre Linux. Ils sont tous les trois régulièrement adaptés à l'évolution de ces plates-formes, depuis de longues années. On peut donc très raisonnablement faire le pari que cette nécessaire adaptation continuera d'avoir lieu dans l'avenir, même si ces différents systèmes d'exploitation devaient être profondément modifiés. Cela garantit au lecteur que vous êtes qu'il pourra continuer à exploiter son travail même s'il change totalement de matériel ou de système d'exploitation dans l'avenir. Par ailleurs, cela ouvre la possibilité de réunir dans des clubs de 3D informatique des amateurs travaillant avec des

machines même très hétéroclites.

Vous pouvez donc faire confiance aux logiciels choisis pour cet ouvrage : ils vous rendront service pendant de longues années, sans que vous ayez besoin de prévoir un budget pour suivre leur évolution. Ce n'est pas là une mince qualité pour un amateur de 3D : les utilisateurs de logiciels de CAO prestigieux (et coûteux !) savent bien à quel point ils sont véritablement rançonnés au fil des ans à coups de mises à jour ; quant aux amateurs désargentés (comme l'auteur), ils se souviennent avec tristesse de petits logiciels sympathiques et peu coûteux, vite tombés dans l'oubli après la faillite de leur éditeur – ou devenus brusquement très chers après s'être constitué un public captif.

Blender, POV-Ray et Python, eux, ne disparaîtront pas, et ils ne vous obligeront pas non plus à vous ruiner en mises à jour. Leur nature de logiciels libres (ou quasi, dans le cas de POV-Ray) les protège très largement de la logique commerciale. En effet, un logiciel libre constitue avant tout un projet informatique dont personne n'est le maître ni le propriétaire (ce qui ne l'empêche pas d'avoir très souvent un coordinateur) : il est le résultat de la collaboration de dizaines de programmeurs passionnés, souvent même bénévoles. Chacun d'entre eux a la faculté d'enrichir le projet (dans son coin éventuellement, mais le plus souvent en se conformant à la ligne définie par un coordinateur). En revanche, aucun n'a la possibilité de s'approprier le logiciel pour lui imposer un prix ou une politique de marketing : des dispositions légales (la licence du logiciel) l'interdisent. La conséquence en est que les logiciels libres sont presque toujours gratuits (c'est en particulier le cas pour les trois que nous emploierons), mais surtout qu'ils ne peuvent mourir (ni même changer de philosophie générale) qu'avec le dernier programmeur qui s'y intéresse – autant dire, dans le cas de logiciels faisant l'objet d'autant d'enthousiasme, pas de sitôt. Quand l'un de ces programmeurs décède, atteint les limites de son talent (pour le dire moins poliment : travaille comme un cochon) ou tout simplement se désintéresse du projet, ça n'empêche pas le logiciel de continuer à s'adapter et même à s'enrichir grâce aux efforts d'autres informaticiens. Même en se limitant aux contributions de bénévoles, on est assuré que le logiciel continuera d'être maintenu.

En informatique, la recherche d'une solution universelle et éternelle reste quand même une utopie (même dans le monde des logiciels libres, où l'évolution de la mode informatique fait parfois des ravages). Il est en revanche parfaitement possible et hautement souhaitable de réduire au strict minimum les difficultés d'adaptation en s'appuyant sur des outils standards – dont les logiciels libres. Mais la sécurité de l'utilisateur est encore bien plus grande s'il peut ajouter à cela une certaine compétence informatique. C'est elle qui lui permettra éventuellement d'adapter à ses propres besoins les solutions générales mises au point par des programmeurs professionnels.

On aurait grand tort de penser qu'un tel objectif est inaccessible à l'amateur.

Par exemple, quantité d'écrivains pas informaticiens pour deux sous complètent la puissance de leur traitement de texte par... leur talent littéraire. L'ambition de ce livre est de permettre à l'amateur de 3D que vous êtes de compléter la compétence des programmeurs de Blender et POV-Ray par votre talent graphique, bien évidemment, mais aussi par votre propre ingéniosité, que vous pourrez exprimer dans le langage Python même en vous limitant à un apprentissage de quelques heures.

2.2 Les logiciels choisis par cet ouvrage

2.2.1 Blender

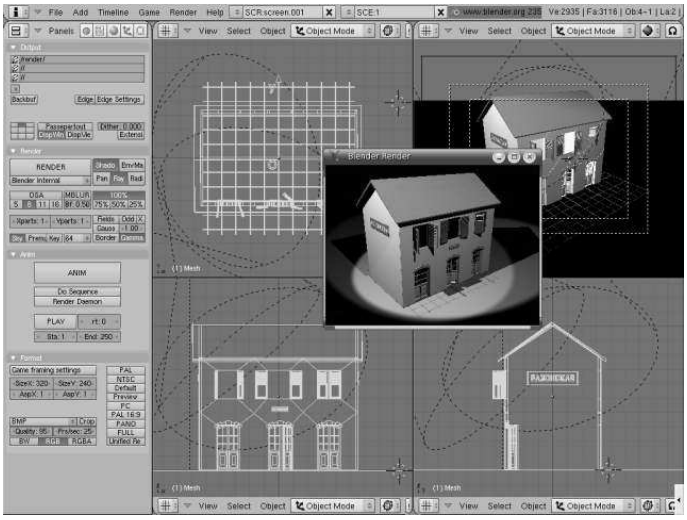


FIG. 2.1 – *Blender, le logiciel phare de la 3D libre, a débuté sa carrière comme un produit commercial.*

Blender se présente comme un logiciel 3D à tout faire : modelage, images de synthèse, animations et même jeux vidéo. En contrepartie de ce cahier des charges très ambitieux (et même probablement trop), c'est un logiciel surchargé de fonctions, et donc difficile à maîtriser en totalité. Cet ouvrage n'est pas assez épais pour permettre de présenter tous ses atouts... et lourdeurs. Nous avons choisi de l'utiliser essentiellement comme modeleur, tâche pour laquelle il est tout à fait efficace et adapté. Il aurait quand même été dommage de ne pas du tout parler de ses aptitudes à produire des images de synthèse, lesquelles peuvent rendre bien des services. Nous en verrons donc les grandes lignes. Mais en la matière, l'auteur estime que POV-Ray lui est encore assez supérieur.

Blender était à l'origine un logiciel commercial, produit par une société néerlandaise qui a cherché – en vain – à atteindre la rentabilité en le commercialisant. Alors que cette société s'approchait du dépôt de bilan, une souscription sur le web a permis son rachat par la communauté du logiciel libre, qui s'est ainsi dotée du jour au lendemain d'un outil très puissant. Mais si Blender est désormais totalement libre d'un point de vue juridique (il est diffusé sous licence GPL), il ne s'est pas encore totalement débarrassé des lourdeurs liées à la politique commerciale de son éditeur (volonté de réaliser toutes les opérations à la souris dans une interface graphique spécifique, pléthore de fonctions d'intérêt anecdotique, choix d'un format d'enregistrement totalement opaque, faiblesses en matière d'échanges de données...). Il continue de s'enrichir grâce à l'effort de développeurs bénévoles, mais en contrepartie, il est susceptible d'évoluer dans l'avenir dans un sens qui ne conviendra pas forcément à tous ses utilisateurs (ceux qui l'apprécient pour ses aptitudes au jeu vidéo interactif n'ont pas du tout les mêmes besoins que ceux qui s'intéressent à la production d'animations de qualité cinématographique). Bref, en dépit de ses grandes qualités, il vaut mieux ne pas tout miser sur lui, et en particulier ne pas se laisser enfermer dans sa logique d'utilisation, qui est parfois inutilement lourde en dépit de l'usage systématique de la souris.

2.2.2 POV-Ray

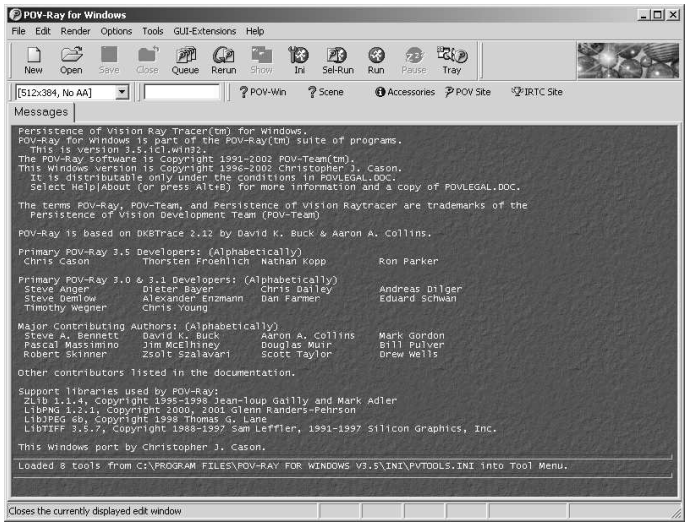


FIG. 2.2 – *POV-Ray dans sa version pour Windows, enrichi d'icônes et de menus déroulants dont la version pour Linux... se passe totalement !*

POV-Ray (également connu sous les noms de POV et *Persistence of vision*) est

ce que l'on appelle un moteur de rendu : sa fonction est de transformer des descriptifs géométriques (de longues listes d'instructions informatiques, qu'il ne produit pas lui-même) en images de synthèse. C'est un projet presque aussi ancien que la micro-informatique grand public (il était déjà connu internationalement du temps où il tournait sous MS-DOS), et qui dès le départ a refusé de poursuivre un objectif lucratif : il a toujours été programmé par des bénévoles et pour une diffusion gratuite. A cause de cette longue histoire, POV-Ray est aujourd'hui d'une richesse, d'une fiabilité et d'une puissance enthousiasmantes... mais malheureusement, ce n'est pas à strictement parler un logiciel libre. En effet, à l'époque de sa création, la réflexion autour des aspects juridiques du logiciel libre était encore balbutiante, et ses créateurs ont commis l'erreur d'interdire qu'il soit jamais diffusé autrement que de façon gratuite : on peut donner POV-Ray sans aucune restriction (il est d'ailleurs diffusé sur le web), mais on n'a pas le droit de le vendre. Et c'est une grave erreur : de ce fait, POV-Ray n'est jamais inclus dans un CD-ROM de logiciels libres ou de sharewares vendu avec un magazine ou un livre, et c'est la raison pour laquelle il reste assez peu diffusé en dépit de ses immenses qualités.

Sur le plan juridique, POV-Ray n'est donc pas totalement un logiciel libre. Mais même les partisans les plus intégristes de l'informatique libre auraient tort de voir en lui un ennemi. Bien au contraire, il faut le vénérer pour ce qu'il est : un précurseur des logiciels libres parmi les plus altruistes et les plus idéalistes.

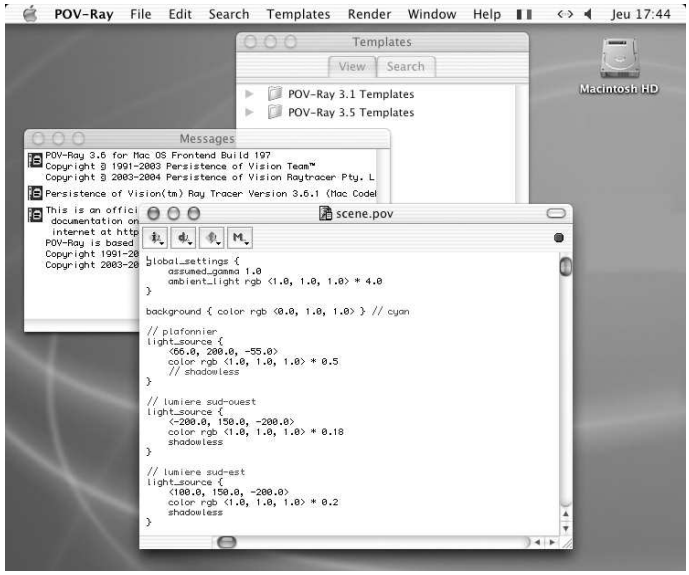


FIG. 2.3 – *POV-Ray dans sa version Macintosh. Même si les menus sont organisés très différemment de la version Windows, les images produites sont rigoureusement semblables.*

2.2.3 Une panoplie de convertisseurs

Blender et POV-Ray n'ont pas du tout été conçus pour travailler ensemble. Leurs développeurs ne se sont pas concertés, on ne peut même pas dire qu'il y ait eu entre eux une saine émulation : la vérité est que les deux projets s'ignorent presque totalement. Pourtant, leurs fonctions sont largement complémentaires : Blender est un très bon modeleur mais un moteur de rendu très perfectible, POV-Ray est un moteur de rendu époustouflant et ne possède pas en lui-même la moindre aptitude au modelage.

Voici donc un cas d'école pour montrer les vertus de l'informatique libre : le mariage entre ces deux logiciels complémentaires va être rendu possible non pas par les développeurs talentueux de POV-Ray, non pas par les informaticiens géniaux de Blender... mais par de minables petits programmes de conversion écrits sans talent aucun par un informaticien amateur : l'auteur de ce livre. Sur le CD-ROM joint à cet ouvrage se trouve une panoplie de convertisseurs (en version Windows, Mac OS/X et Linux) qui permettent de faire transiter les données entre nos deux logiciels phares, et rendent aussi sans fioritures quantité de petits services bien utiles à l'amateur de 3D.

Ces convertisseurs sont tous diffusés avec leur code informatique sous la licence GPL qui en fait des logiciels libres. N'importe quel informaticien peut donc les reprendre et les améliorer... sauf que si vraiment le besoin s'en faisait sentir, il serait sans doute préférable de réécrire intégralement et avec plus de talent ces petites bidouilles d'amateur ! Mais si perfectibles soient-elles, elles ont l'immense qualité d'exister, et vous verrez qu'elles vous rendront bien des services.

2.2.4 Python

Nous avons parlé de Python comme d'un logiciel. C'est un peu abusif, il s'agit en fait d'un langage de programmation – que l'on emploie pour écrire de petits logiciels. Ce langage n'est pas particulièrement destiné à la 3D, mais il a été choisi par les développeurs de Blender pour enrichir les fonctions de ce dernier. Comme cet ouvrage avait besoin d'un outil simple pour permettre à l'amateur de 3D d'automatiser toutes sortes d'opérations, il n'a pas paru utile de se singulariser en cherchant autre chose : c'est donc Python que nous emploierons pour expliquer comment on peut automatiser toutes sortes d'opérations élémentaires.

D'ailleurs, Python est effectivement un langage riche, clair, bien conçu et très pédagogique. Il a en outre la qualité d'être disponible en standard sous Mac OS/X comme avec la plupart des distributions Linux (sous Windows, il faudra l'installer, mais cela ne présente aucune difficulté ; il est diffusé gratuitement). Pour l'amateur de 3D, il présente surtout l'intérêt de permettre même à des débutants complets de réaliser automatiquement et avec une régularité parfaite des opéra-

tions qui demanderaient trop de minutie si elles étaient réalisées manuellement. Il faut insister sur le fait que l'amateur peut aboutir à de tels résultats en n'ayant de ce langage qu'une connaissance même pas élémentaire. Ce que nous dirons de Python sera strictement utilitaire, et si le lecteur souhaite une véritable initiation à ce langage, il lui faudra étudier par ses propres moyens.

Python est certes un langage simple et pédagogique, mais c'en est un parmi d'autres. Ce que nous lui demanderons de faire est tellement élémentaire que cela pourrait sans difficulté être réalisé avec pratiquement n'importe quel autre langage. Si donc le lecteur a déjà des notions de C, de C++, de Java, de Pascal, de Tcl/Tk, de Perl (excellent choix pour les partisans du moindre effort) ou de n'importe quoi d'autre, il peut allègrement se désintéresser de Python et réaliser les opérations d'automatisation dont nous parlerons avec les outils qu'il connaît ; comme il s'agit de manipulations élémentaires, cela ne lui demandera presque aucun effort d'adaptation.

2.2.5 Un logiciel de retouche d'images

Bien évidemment, l'amateur d'images de synthèse est d'abord un amateur d'images tout court, et il a besoin d'un logiciel susceptible de l'aider dans cette perspective : pour éventuellement recadrer les images produites, les contraster, les compresser, effectuer des montages... et même, tout simplement, pour les regarder ! Mais par ailleurs – nous aurons l'occasion d'y revenir –, une grande partie du travail créatif en 3D est toujours ébauchée par des esquisses ou des approximations dessinées à main levée, et il est extrêmement utile de pouvoir prendre des repères sur de tels croquis en travaillant au pixel près.

Donc, procurez-vous un logiciel de retouche d'images. Cet ouvrage n'en recommande aucun en particulier, et si vous avez déjà l'habitude de travailler avec un de ces outils, utilisez-le pour la 3D. Certes, tous les logiciels de retouche ne se valent pas – évitez de vous contenter du Paint rudimentaire de Windows ; on peut vraiment faire mieux ! – mais tous couvrent les besoins les plus courants, et vous n'avez pas forcément l'obligation de faire les frais d'une licence d'utilisation du célébrissime Photoshop d'Adobe. En particulier, le logiciel libre Gimp existe désormais pour Windows, Mac et Linux, et si vous l'utilisez comme assistant au modelage en 3D, il n'aura aucune peine à vous donner toute satisfaction.

2.2.6 Le partage des tâches entre les logiciels

L'esprit général pour l'emploi des différents logiciels dont nous venons de parler est le suivant : a priori, on modèlera les objets en 3D avec Blender ; on les fera passer du format de Blender à celui de POV-Ray grâce aux convertisseurs ; enfin, on réalisera les images de synthèse avec POV-Ray.

Mais cette démarche générale peut très facilement être adaptée, enrichie (notamment avec le logiciel de retouche) et surtout automatisée : ce sera le rôle de certains convertisseurs, de petits scripts Python, voire de modifications directes du code informatique 3D entre Blender et POV-Ray.

2.3 Prérequis pour l'installation

2.3.1 Exigences matérielles

Même si l'image de synthèse est toujours capable de pousser les ordinateurs les plus puissants jusqu'à leurs limites, l'amateur de 3D peut déjà faire énormément de choses sur une machine ordinaire, même assez ancienne. Disons pour schématiser que vous ne devriez avoir aucun problème avec un ordinateur fabriqué à partir de 2001 (sous Linux, vous risquez d'avoir quand même besoin d'un noyau plus récent) et doté de 128 Mo de mémoire vive, que ce soit un PC ou un Macintosh (dans ce dernier cas, il faut quand même qu'il tourne sous Mac OS/X).

Travailler sur des configurations plus vieilles ou moins puissantes est envisageable, mais à la condition de vous procurer des versions un peu désuètes de nos logiciels phares. Par exemple, l'auteur s'amuse parfois à faire tourner Blender 2.0 et POV-Ray 3.5 sur son vieux PC taiwanais de 1998 doté de 48 Mo de mémoire vive. Non seulement ça marche encore avec des temps de réponse très acceptables, mais les vieilles versions de Blender et POV-Ray ne plantent pas quand on leur fait avaler des documents préparés avec leurs avatars postérieurs – pas plus que quand on demande à une version Mac de relire un document préparé sur PC, ou le contraire. Exploit technique ? Non, honnêteté élémentaire : c'est à cela qu'on reconnaît avoir affaire à des logiciels libres, qui ne se financent pas en obligeant leurs utilisateurs à acheter des mises à jour...

2.3.2 Se procurer les versions récentes des logiciels

Le CD-ROM joint contient une version récente de Blender (la 2.37 est la dernière en date au moment où nous rédigeons ces lignes) et une panoplie de convertisseurs, le tout en triples versions : pour Windows, Mac OS/X et Linux. POV-Ray a lui aussi été adapté à ces trois environnements et peut être dupliqué librement, mais sa licence particulière nous interdit de le diffuser avec cet ouvrage ; il vous faudra donc le télécharger à partir de `http ://www.povray.org` (cliquez sur « Download ») ; à l'heure actuelle, sa version la plus récente est numérotée 3.6. Python est déjà installé en standard sous Mac OS/X et la plupart des distributions Linux, généralement en version 2.3 même si la 2.4.1 est disponible... et si la 2.2 est déjà largement suffisante pour nos besoins. Windows étant la seule plate-

forme qui n'installe pas systématiquement Python sur une configuration standard, nous avons placé la version 2.4.1 de Python sur le CD-ROM joint (ainsi que la version 2.2.1, moins récente mais déjà très suffisante, et par ailleurs plus facile à installer sur les versions de Windows un peu anciennes).

2.4 Installation proprement dite

2.4.1 Des configurations similaires mais pas identiques

Bâti autour de logiciels libres dont les auteurs détestent se rendre esclaves d'une configuration quelconque, cet ouvrage s'est fixé pour objectif de présenter des méthodes de travail utilisables sur n'importe quelle machine récente : PC sous un Windows quelconque (2000, XP, ou même 98 et ME) ou sous Linux, ou encore Macintosh (sous Mac OS/X). A un certain degré d'abstraction, tous ces ordinateurs se ressemblent et sont capables d'effectuer les mêmes calculs géométriques. Dans le détail, hélas, ils polluent toujours la logique informatique universelle par quantité de spécificités et de gadgets – dont on est bien forcé de tenir un peu compte, ne serait-ce que pour les contourner. Et à cet égard, Blender et POV-Ray n'ont pas choisi la même philosophie...

Blender est avant tout un outil manipulable à la souris au sein d'une interface graphique. Or, l'aspect des menus, des ascenseurs, des fenêtres de dialogue, des polices de caractères, etc., diffère sur d'innombrables détails d'une plate-forme à l'autre, obligeant les programmeurs à un énorme travail d'adaptation lorsqu'ils veulent conférer un aspect unifié à un logiciel tournant aussi bien sur Mac que sur PC. Blender a éliminé ce problème de façon radicale en n'utilisant pas du tout les routines d'affichage déjà prévues par Windows, Mac OS/X et X Window (l'interface graphique de la plupart des Unix, dont Linux). Blender ne ressemble donc pas à un logiciel pour Windows, ni pour Mac. En revanche, il a rigoureusement le même aspect – spécifique – sur toutes les plates-formes où il existe, et donc les pages que nous lui consacrerons seront abondamment illustrées de copies d'écran auxquelles tous nos lecteurs pourront se référer, quelle que soit leur machine.

Pour POV-Ray, c'est tout le contraire : POV-Ray pour Windows a l'aspect classique d'un logiciel Windows (fig. 2.2), POV-Ray pour Mac a l'aspect d'un logiciel Mac (fig. 2.3)... mais les deux interfaces n'ont rigoureusement rien à voir : même les intitulés des menus sont différents ! Ces interfaces ont en effet été réalisées par des programmeurs bénévoles, passionnés les uns par Windows, les autres par le Mac – et qui ne se sont pas du tout mis d'accord (les logiciels libres, c'est aussi cela : un moyen de laisser les bonnes volontés être productives sans les contraindre forcément à consacrer du temps à des réunions de concertation). Cela n'empêche pas ces deux avatars de POV-Ray de produire au pixel près rigoureuse-

ment les mêmes images. Car dans les deux cas, l'interface graphique n'est rien de plus qu'un emballage, plaqué de façon assez artificielle sur un logiciel qui, en fait, travaille exclusivement dans la mémoire de l'ordinateur – et pas du tout à l'écran. C'est tellement vrai que la version pour Linux de POV-Ray... n'a carrément pas d'interface graphique. Par conséquent, les chapitres que nous consacrerons à POV-Ray ne comporteront presque pas de copies d'écran, mais seulement des vues en 3D des volumes manipulés.

Comment les linuxiens parviennent-ils à utiliser POV-Ray en l'absence d'interface à menus déroulants ? C'est fort simple : il leur suffit de taper une ligne de commande et d'y indiquer pour POV-Ray le nom d'un fichier contenant une longue suite d'instructions informatiques à convertir en image ; on n'a pratiquement besoin de rien d'autre. Il est même possible de demander à POV-Ray de ne pas afficher l'image qu'il fabrique, mais seulement de l'enregistrer une fois qu'il a terminé son travail. Pendant ce temps-là, les linuxiens peuvent patienter en utilisant leurs écrans pour travailler à autre chose ; généralement, ils ne s'en privent pas.

A un niveau beaucoup moins ambitieux, les convertisseurs enregistrés sur le CD-ROM joint à cet ouvrage fonctionnent sur le même principe : une ligne de commande inscrite dans la fenêtre noire d'un terminal suffit pour leur intimer un ordre, qu'ils exécutent aussitôt, pratiquement sans rien afficher à l'écran, mais en enregistrant sur le disque dur le résultat de leur travail. Leur interface se réduit donc à cette simple ligne de commande, pratiquement identique d'un ordinateur à l'autre, que ce soit avec un austère Windows 2000 ou les fenêtres translucides de Mac OS/X. Les minuscules programmes Python que vous pourrez être amenés à rédiger vous-même fonctionneront eux aussi à base de ligne de commande et d'écran noir. Ce n'est peut-être pas très joli... mais qu'importe si cela mène à produire de jolies images ! En guise d'illustrations, les pages que nous consacrerons à l'usage des convertisseurs n'auront pas recours aux copies d'écran, mais seulement au fac-similé de quelques lignes de code informatique, et aux représentations 3D auxquelles elles permettent d'aboutir.

Python a été conçu par ses auteurs pour fonctionner de façon aussi identique que possible d'une plate-forme à l'autre, et il est par ailleurs diffusé gratuitement, mais cela étant, ce ne sont malheureusement pas ses concepteurs qui pouvaient décider seuls d'intégrer ou non ce remarquable langage à une configuration de base. Mac OS/X a pris la décision de rendre Python disponible sur toute machine équipée de ce système d'exploitation, et la plupart des distributions Linux récentes ont opéré le même choix. Microsoft, lui, par hostilité générale aux principes du logiciel libre, s'est refusé à prendre en compte ce langage, et Windows est donc la seule plate-forme sur laquelle Python devra être installé par vos soins. Comme on touche là à des questions techniques relativement pointues, vous trouverez peut-être son installation sur Windows nettement plus laborieuse que celle d'un logiciel

ordinaire... mais justement, un langage de programmation n'est pas un logiciel ordinaire. Vous devriez vous en sortir sans trop de peine en suivant nos indications.

Il faut savoir que les différences visuelles d'une plate-forme à l'autre ne sont que la partie émergée de l'iceberg des incompatibilités. Dans les profondeurs de la machine, au niveau du processeur, les mêmes traitements logiques sont réalisés de façon extrêmement différente sur Windows, Linux ou Mac OS/X. Pour le programmeur, ce serait une calamité que de devoir élaborer trois programmes différents pour résoudre trois fois le même problème (soit dit en passant, c'est la raison pour laquelle bien des programmes écrits pour Windows n'ont jamais été adaptés au Mac, et inversement). Heureusement, avec les bons vieux programmes austères en ligne de commande, il est assez facile d'obtenir une version Windows, une version Mac et une version Linux à partir d'un seul listing d'instructions. Sur le CD-ROM joint à cet ouvrage, les convertisseurs mis au point par l'auteur (en langage C) ont pu ainsi être compilés et livrés en triple exemplaire : Windows, Mac OS/X, Linux ; sans supplément de prix pour vous – et surtout sans surcroît d'effort pour l'auteur. Cette facilité à passer d'une plate-forme à l'autre est inappréciable si l'on cherche à faire travailler en équipe des amateurs utilisant des plates-formes différentes. En particulier, toutes les bidouilles mises au point en Python sur un Mac peuvent servir sous Windows (ou d'ailleurs Linux), et vice-versa. Cette bonne vieille ligne de commande a décidément bien des vertus...

2.4.2 Installation des logiciels sous Windows

Blender

Dans le répertoire `windows` de notre CD-ROM, vous trouverez un sous-répertoire `blender` contenant la version 2.37 de ce logiciel sous deux formes : un programme d'installation et une archive .zip. C'est également sous ces deux formes que le logiciel est diffusé sur `http ://www.blender3d.com`. Le programme d'installation, qu'on lance évidemment par un double-clic, fonctionne de la façon la plus classique qui soit (fig. 2.4 : acceptation de la licence, proposition d'un répertoire pour installer le logiciel, etc.) et donne toute satisfaction. L'archive .zip n'est donc pas utile a priori, mais bien des partisans du logiciel libre détestent les programmes d'installation dont le fonctionnement leur paraît opaque, notamment en ce qui concerne les modifications de la base de registres de Windows. Si vous appartenez à cette catégorie d'utilisateurs méfiants et seulement dans ce cas, vous pourrez vous passer du programme d'installation en recopiant manuellement sur la partie du disque dur qui vous sied le contenu de ce fichier .zip (qu'il vous faudra bien sûr décompresser avec Winzip ou un équivalent) ; il est seulement à craindre dans ce cas que les fichiers de Blender n'aient pas leur icône standard et ne soient accessibles que par les menus de Blender et non par

double-clic. Après tout, on n'en meurt pas... à vous de voir !

Il est arrivé par le passé que des versions de Blender pour Windows se révèlent incompatibles avec certaines cartes graphiques, au point de planter de façon radicale dès le lancement. Ce n'est normalement plus le cas, mais on peut encore constater certaines anomalies sur des ordinateurs sans carte accélératrice 3D (ce qui est le cas de certains portables conçus pour la seule bureautique). En particulier, il arrive que les icônes soient illisibles, ce qui ne rend pas le logiciel inutilisable pour quelqu'un qui est très habitué à Blender, mais peut quand même être très gênant. Si votre machine est capable de faire tourner des CD de jeux, vous n'aurez normalement pas de problème. Dans le cas contraire, il faudra peut-être songer à faire les frais d'une carte graphique plus adaptée.

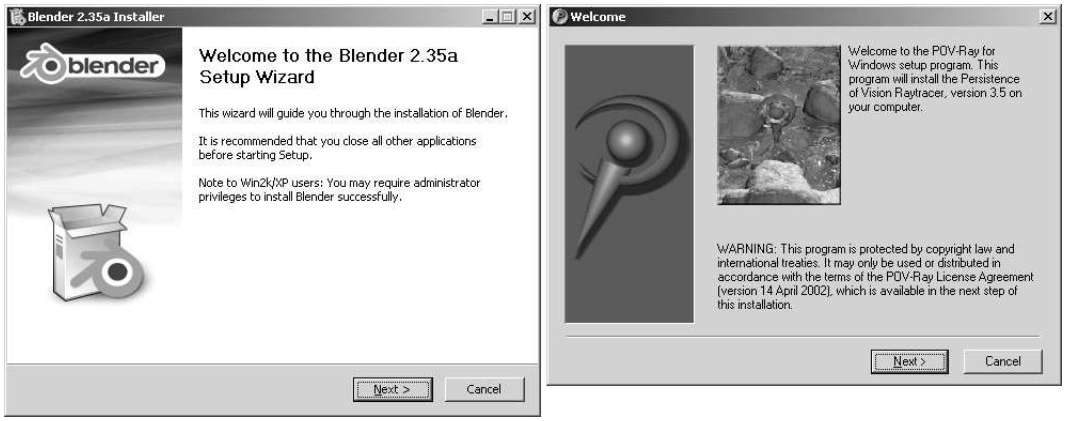


FIG. 2.4 – *Sous Windows, Blender et POV-Ray sont l'un et l'autre dotés d'un programme d'installation très simple.*

POV-Ray

La licence du logiciel gratuit POV-Ray ne permettant pas sa diffusion avec un ouvrage vendu, nous n'avons pu l'inclure dans le CD-ROM joint. Vous devrez donc télécharger vous-même depuis `http ://www.povray.org` le programme d'installation, un .exe très classique, qui pèse quelque 10 Mo. Si c'est plus que votre connexion téléphonique ne peut le supporter, il n'y a guère d'autre solution que de faire réaliser le téléchargement sur la machine d'un ami, et de graver l'utilitaire d'installation sur un CD-ROM.

Ce programme se lance par un double-clic et n'appelle pas de commentaire particulier (fig. 2.4). Après l'installation, POV-Ray pour Windows lance automatiquement le calcul d'une petite image de synthèse qui vous permettra de vous assurer du bon fonctionnement du logiciel.

Python

Le programme d'installation de la version 2.4.1 de Python pour Windows se trouve dans le répertoire `windows` du CD-ROM joint, sous-répertoire `python`. Vous pouvez également le télécharger sur `http ://www.python.org`. Il s'agit d'un fichier .msi (de quelque 10 Mo), c'est-à-dire qu'il ne fonctionne qu'avec le programme Microsoft Installer. Ce dernier est inclus en standard dans Windows XP, mais ne l'était pas dans toutes les versions précédentes de Windows. Il peut être téléchargé gratuitement sur le site de Microsoft, mais au prix de formalités d'enregistrement assez déplaisantes. Si vous y êtes allergique, vous pouvez vous rabattre sur la version 2.2.1 de Python (très suffisante pour ce que nous souhaitons faire), dont vous trouverez le programme d'installation dans le répertoire `versions_anciennes` de notre CD-ROM. Cette fois, il s'agit d'un fichier .exe on ne peut plus classique.

Dans les deux cas, prenez bien garde à l'emplacement de l'arborescence où Python sera installé. C'est normalement `c :\Python24` si l'on installe avec le fichier .msi, et `c :\Python22` avec le .exe. Notez-le bien, cela va nous resservir.

L'installation de Python proprement dite s'opère de façon relativement classique, mais pour que vous puissiez utiliser le programme en ligne de commande (dans une fenêtre noire, en tapant des instructions informatiques ; mais si, vous en êtes capable, vous le verrez !), vous allez devoir vous livrer manuellement à une opération assez technique, qu'on appelle (aïe aïe aïe) la modification d'une variable d'environnement. Si vous avez un informaticien sous la main, il se fera une joie de vous expliquer à quoi cela correspond, et même de le faire à votre place (surtout ne l'en empêchez pas !). Dans le cas contraire, il va vous falloir accomplir l'opération suivante – qui nous permettra, comme souvent avec Windows, de nous extasier sur les vertus d'indéniable simplicité de cette interface censée être si conviviale. La manipulation que nous décrivons (fig. 2.5) correspond à la version 2000 de Windows, mais il est probable qu'il vous faudra l'adapter un peu si vous travaillez avec un autre avatar de ce soi-disant standard.

Cliquez sur le bouton Démarrer, en bas à gauche de la barre des tâches de Windows. Cliquez sur l'option Paramètres, puis la sous-option Panneau de configuration. Dans la fenêtre qui apparaît (c'est la première de la série), double-cliquez sur l'icone Système. Dans la fenêtre qui apparaît (et de deux), cliquez sur l'onglet Avancé, puis sur le bouton Variables d'environnement. Dans la fenêtre qui apparaît (et de trois), fouillez au moyen de l'ascenseur dans la deuxième... sous-fenêtre, intitulée Variables système, jusqu'à trouver la variable Path, et cliquez dessus, puis sur le bouton « Modifier... ». Cela fait encore apparaître une nouvelle fenêtre de dialogue (la quatrième, sauf erreur ; au cas où vous l'ignoreriez, Windows signifie « Fenêtres », et ce mot... est vraiment au pluriel !), avec une jolie case de saisie intitulée « Valeur de la variable ». Nous y voici : c'est maintenant

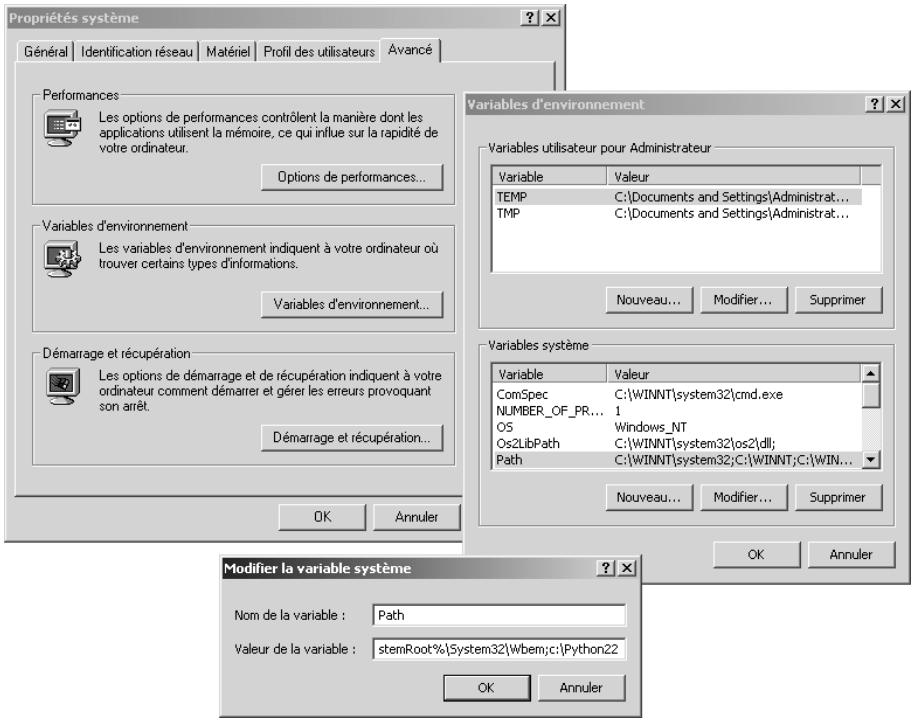


FIG. 2.5 – *Il est souhaitable de compléter l'installation de Python sous Windows par la modification d'une variable d'environnement, afin que ce langage soit toujours accessible en ligne de commande.*

que ça se corse !

La terrible manipulation à laquelle vous allez devoir vous livrer consiste en fait... à taper une douzaine de lettres au clavier à la fin de cette case de saisie (oui, c'est pour en arriver là que nous avons ouvert quatre fenêtres ; c'est ça, Windows !). Pour que ce soit encore plus amusant, cette case est généralement trop étroite pour que l'on puisse lire tout son contenu ; il vous faut donc cliquer dedans, puis déplacer le curseur à la toute fin de ce qui est déjà écrit en tapant autant de fois que nécessaire sur la touche de déplacement du curseur vers la droite (ah oui, désolé, on ne fait pas ça à la souris). Quand vous êtes arrivé au bout du bout, et surtout pas avant, vous tapez très exactement

```
;c :\Python24
```

à la fin de cette case de saisie... à moins qu'il ne faille plutôt taper `;c :\Python22` si vous avez choisi d'installer la vieille version de Python (rappelez-vous : il y a quelques paragraphes, nous vous avons demandé de le noter). Dans les deux cas, faites bien attention : la mention commence par `;c :` et non par `c :` ; n'oubliez surtout pas le point-virgule.

Ouf, ça y est presque. Cliquez sur tous les boutons OK (il y en a trois) que Windows vous propose dans les fenêtres déjà ouvertes – ce qui aura l'heureux effet de les refermer une par une dans l'ordre inverse de leur ouverture. Vous pouvez aussi refermer la fenêtre du Panneau de configuration.

En entreprise, la plupart des informaticiens Windows sont payés grassement à accomplir deux ou trois fois par jour (par semaine ?) des manipulations du type de celle que vous venez d'accomplir. Aimeriez-vous faire leur métier ?

Les convertisseurs

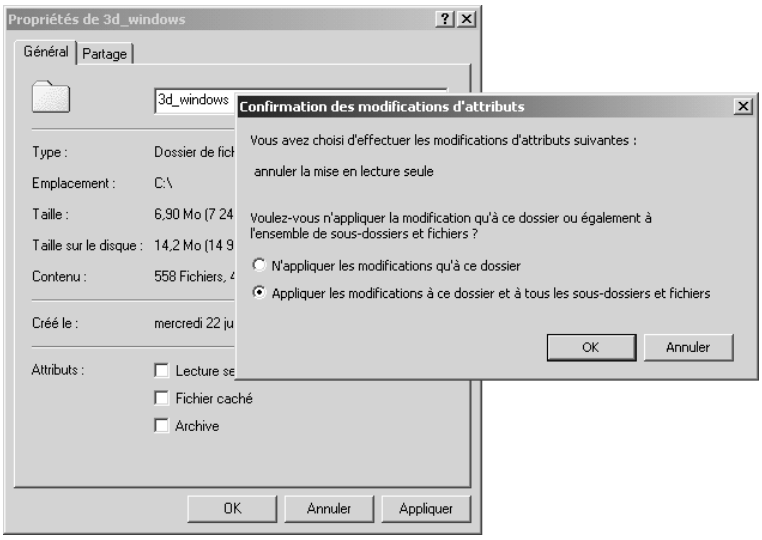


FIG. 2.6 – *Ce qui est recopié d'un CD-ROM est au départ en lecture seule. Il faut donc modifier les propriétés du dossier recopié (sous Windows 2000 : clic droit sur son icône, Propriétés, décocher « Lecture seule », cliquer OK ; il faut aussi valider « Appliquer les modifications aux sous-dossiers » dans la fenêtre qui apparaît ensuite).*

L'installation des convertisseurs sur votre disque dur est fort heureusement nettement moins complexe que celle de Python. Il vous suffit d'ouvrir le dossier `windows` du CD-ROM, et de recopier le sous-dossier `3d_win` sur votre disque dur. En théorie, vous pouvez placer cette copie n'importe où, mais il vous sera beaucoup plus facile d'y accéder en ligne de commande si vous choisissez de la mettre tout en haut de l'arborescence, à la racine du disque dur `c :`. Pour cela, le moyen le plus simple est de faire apparaître l'icône du disque dur `c :` en double-cliquant sur l'icône du Poste de travail, puis de glisser sur elle l'icône du sous-dossier `3d_win` du CD-ROM.

Ce n'est pas tout à fait suffisant. Les dossiers et documents sauvegardés sur un CD-ROM sont pourvus de l'attribut « Lecture seule » (il est en effet impossible de les modifier directement sur la galette), et par conséquent, leur copie sur le disque dur est elle aussi protégée en écriture. Or, vous aurez besoin d'y inscrire de nouveaux paramètres et d'y enregistrer des fichiers. Il vous faut donc modifier ce réglage, ce qui n'est pas très compliqué. Double-cliquez sur l'icône du Poste de travail, puis sur celle du disque dur `c :` (si c'est bien là que vous avez recopié le dossier) jusqu'à faire apparaître l'icône de `3d_win`. Faites un clic droit sur elle, puis validez Propriétés. La case « Lecture seule » est cochée, décochez-la et cliquez OK. Windows vous offrira la possibilité d'appliquer la même modification au contenu du dossier (fig. 2.6 ; attention, ce n'est pas l'option validée par défaut), et c'est bien ce qu'il convient de faire.

2.4.3 Installation des logiciels sous Mac OS/X

Blender

Vous trouverez sans surprise les versions de Blender pour Macintosh dans le répertoire `mac` de notre CD-ROM, sous-répertoire `blender`. Il en existe deux variantes, la première pour Mac OS/X 10.2, la seconde pour les versions ultérieures du système d'exploitation d'Apple. Dans les deux cas, il s'agit de fichiers compressés .zip (Mac OS/X les ouvre grâce à un double-clic banal), dans lesquels on trouve un dossier à recopier dans le répertoire Applications de votre Mac. Vous pouvez aussi (et c'est d'ailleurs préférable pour éviter des problèmes en cas de mise à jour du système) créer un sous-répertoire Applications dans votre dossier personnel et recopier là le dossier de Blender.

POV-Ray

Pas plus que la version Windows, la version pour Mac de POV-Ray ne peut être diffusée avec un ouvrage vendu, et vous devrez donc la télécharger par vos propres moyens (depuis `http ://www.povray.org`). Elle prend la forme d'un fichier .sit (d'environ 9 Mo), c'est-à-dire compressé avec le logiciel Stuffit. Les gourous du logiciel libre ne manqueront pas de s'indigner de l'usage de cet utilitaire propriétaire pour diffuser un programme quasi libre comme POV-Ray, et c'est effectivement une grave faute de goût (et d'autant plus que l'utilitaire libre gzip, qui rend des services équivalents, est toujours installé avec Mac OS/X). Mais pour des raisons historiques, Stuffit est effectivement l'utilitaire de compression le plus utilisé par les fans d'Apple, et il est presque toujours installé en standard sur les Mac (dans le cas contraire, on peut se le procurer sans peine sur quantité de sites consacrés aux produits d'Apple).

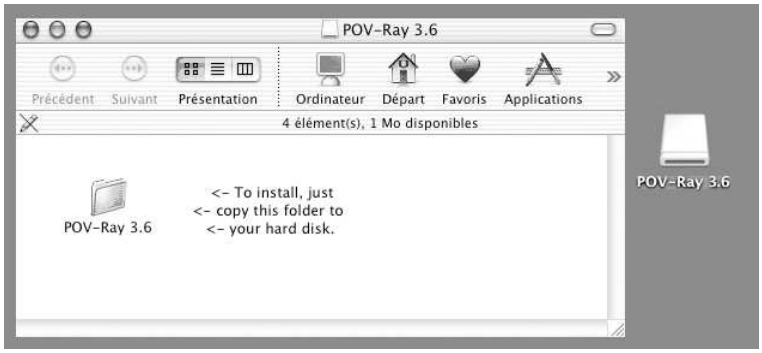


FIG. 2.7 – *Sous Mac OS/X, POV-Ray s'installe en recopiant sur le disque dur un dossier accessible grâce à un fichier .dmg (disk image), qui apparaît sous le Finder avec l'aspect d'un disque dur externe.*

Si Stuffit est bien installé sur votre Mac, il vous suffira de double-cliquer sur le fichier téléchargé pour faire apparaître sur le Finder l'image d'un disque dur virtuel (fig. 2.7), dans lequel vous trouverez un dossier à recopier où vous le souhaitez sur votre disque dur (le répertoire Applications est évidemment un emplacement de choix pour effectuer cette copie, mais l'observation que nous avons faite pour Blender reste valable : il est préférable de créer un sous-répertoire Applications dans votre dossier personnel).

Si vous sauvegardez le fichier .sit sur un CD-ROM avec l'idée de réinstaller POV-Ray ultérieurement, votre prudence est digne d'éloges... mais il ne faudra pas oublier de recopier ce fichier sur votre disque dur avant d'opérer la décompression : Stuffit a besoin d'inscrire des fichiers provisoires pour fonctionner, et il le fait par défaut à l'endroit où se trouve le fichier .sit ; donc si ce fichier est placé sur un CD-ROM, non inscriptible par nature, un plantage (sans conséquence) est assuré.

Les convertisseurs

Pour installer nos convertisseurs, il vous suffira de recopier sur votre disque dur le sous-dossier `3d_mac` (lui-même inclus dans le dossier `mac` de notre CD-ROM). Placez de préférence ce dossier au premier niveau de votre répertoire personnel, en faisant glisser son icône sur celle de la maison blanche, qui symbolise votre répertoire. Mac OS/X se charge tout seul d'enlever l'attribut « Lecture seule » des fichiers recopiés, et vous êtes donc dispensé d'une manipulation équivalente à celle que nous avons décrite pour Windows.

2.4.4 Installation des logiciels sous Linux

Blender

Blender existe en deux versions pour Linux : la normale, qui nécessite une carte accélératrice 3D et Open GL (ou un équivalent libre comme Mesa), et la statique, qui se passe de ces additifs au prix d'une légère augmentation des temps de réponse. Vous les trouverez toutes deux dans le sous-répertoire `blender` du répertoire `linux` de notre CD-ROM. Dans les deux cas, il s'agit de fichiers .tar traditionnels, compressés avec l'utilitaire bzip2, lequel est désormais livré avec toutes les distributions courantes de Linux même s'il n'est pas encore très usité (les versions plus anciennes de Blender étaient d'ailleurs compressées plus traditionnellement avec gzip). Fondamentalement, l'installation consiste à recopier ces fichiers sur votre disque dur, puis à les décompresser où bon vous semble : le répertoire décompressé contiendra un exécutable nommé blender (ou blenderstatic pour la version statique), qu'il suffira de lancer depuis un quelconque environnement graphique de type XWindow. On peut bien sûr faire tout cela à la souris avec les outils standard d'un environnement du genre de Gnome ou KDE. Mais un linuxien qui se respecte (comme l'auteur) préférera ouvrir un terminal sous XWindow et taper quelque chose de beaucoup plus simple et clair, du genre :

```
cd /mnt/cdrom
cp linux/blender/blender-2.37a-linux-glibc2.2.5-i386.tar.bz2

cd
bzip2 -d blender-2.37a-linux-glibc2.2.5-i386.tar.bz2
tar -xvf blender-2.37a-linux-glibc2.2.5-i386.tar
rm -f blender-2.37a-linux-glibc2.2.5-i386.tar
mv blender-2.37a-linux-glibc2.2.5-i386 blender
cd blender
./blender &
```

Si vous faites observer à un linuxien pur et dur que ce qui précède est quand même un tantinet illisible, il ne manquera pas de vous répondre avec une entière conviction : « Oui, mais au moins on comprend ce qu'on fait. » Et le pire, c'est qu'il le pense vraiment.

POV-Ray

Le programme d'installation de POV-Ray pour Linux, tout comme les versions pour Windows et Mac OS/X, devra être téléchargé depuis `http ://www.povray.org`. Il est livré sous la forme d'un fichier .tgz comprenant un programme d'installation, qui devra être lancé avec les privilèges du super-utilisateur root, et qu'il est

donc plus simple de recopier directement en `/root`. Cela fait, la manipulation prendra à peu près cette forme :

```
su
[mot de passe de root]
cd
tar -zxvf povlinux-3.6.tgz
rm -f povlinux-3.6.tgz
cd povray-3.6
./install
```

Le programme d'installation s'interrompra brièvement pour vous demander de taper R ou S pour lancer ou non un test de rendu. Vous pouvez taper R si vous comprenez l'anglais, car le résultat du test sera affiché dans cette langue. Sinon, tapez S.

L'exécutable de POV-Ray est placé en `/usr/local/bin` (ajoutez éventuellement le nom de ce répertoire à votre variable d'environnement `PATH`), et le reste des fichiers utiles au fonctionnement en `/usr/local/share/povray-3.6`. Le sous-répertoire `povray-3.6` qui reste dans votre répertoire `/root` après l'installation contient des exemples ou des explications, et vous avez tout intérêt à le déplacer à un endroit moins sensible de l'arborescence, par exemple dans une branche de `/home`. Actuellement, seul le super-utilisateur dispose des droits d'accès à ce sous-répertoire, mais cela se corrige facilement à coups de `chown, chgrp, chmod`.

Il faut signaler une bizarrerie imposée depuis la version 3.6 de POV-Ray pour Linux : le super-utilisateur va devoir donner explicitement l'autorisation d'enregistrer les images de synthèse sur le disque dur en modifiant deux lignes d'un fichier de configuration `/usr/local/etc/povray/3.6/povray.conf` : là où il était inscrit (aux environs de la ligne 45) :

```
[File I/O Security]
;none
;read-only
restricted
```

il faut déplacer un point-virgule pour aboutir à :

```
[File I/O Security]
none
;read-only
;restricted
```

Pourquoi diable la configuration de base de POV-Ray 3.6 interdit-elle d'enregistrer les images produites ? L'auteur avoue qu'il se perd en conjectures... mais le programme fonctionne de nouveau très bien après la modification que nous venons de décrire !

Les convertisseurs

Les convertisseurs se trouvent dans le sous-répertoire `3d_linux` du répertoire `linux` de notre CD-ROM. Vous pouvez recopier ce sous-répertoire où bon vous semble, mais son emplacement le plus naturel est en haut de l'arborescence de votre répertoire personnel. N'oubliez pas de rétablir les droits d'écriture, systématiquement éliminés d'une sauvegarde sur CD-ROM. Dans une console, cela se fait ainsi :

```
chmod a+rw 3d_linux -R
```

Chapitre 3

Ce que l'amateur doit savoir de la 3D

3.1 La 3D maillée

Il existe une petite poignée de techniques pour représenter des volumes sur un écran d'ordinateur, mais nous nous focaliserons sur la plus simple et la plus universelle – dans laquelle nous aurons d'ailleurs le plaisir de rencontrer l'équivalent exact de la maquette en carton. Cette technique merveilleusement élémentaire est connue sous le nom de 3D maillée (en anglais : *mesh*). Elle consiste à assimiler n'importe quel volume à l'assemblage des surfaces qui constituent son enveloppe.

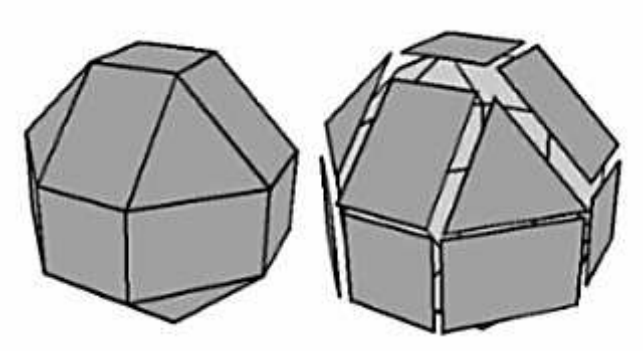


FIG. 3.1 – *En 3D maillée, exactement comme pour la maquette en carton, un volume est géré comme l'assemblage des surfaces constituant son enveloppe.*

Notez bien que ces surfaces sont des polygones rigoureusement plats. Cette caractéristique rend les traitements informatiques particulièrement simples et rapides, car une grande partie d'entre eux pourront être opérés dans un espace eucli-

dien à deux dimensions. Soit dit en passant, il faut tordre le cou à une idée reçue : la 3D informatisée n'est pas du tout une technique d'avant-garde, et de simples amateurs peuvent tout à fait apprendre à la dominer en détail. Pour l'essentiel, il ne s'agit que de l'application à l'informatique de principes géométriques assez peu complexes (on les enseigne au collège) et parfaitement connus depuis des millénaires. L'informatique a seulement apporté à la 3D la possibilité d'effectuer à très grande vitesse des dizaines de milliers de calculs fort simples, tous réalisables avec une calculette à bon marché.

Les polygones représentés par la 3D maillée sont presque toujours des triangles. Là encore, c'est particulièrement simple, puisqu'un triangle est plat par définition, et qu'il suffit de connaître l'emplacement de ses trois sommets pour que toutes ses caractéristiques – longueur des côtés, valeur des angles... – puissent être calculées (rassurez-vous, pas par vous : l'ordinateur s'en chargera !). L'usage systématique des triangles est aussi extrêmement pratique pour produire une image de synthèse : la représentation en perspective d'un triangle est elle-même un triangle (évidemment plat), et ce en dépit de tous les mouvements, déformations et rotations que l'on peut décider de lui faire subir. Rien de plus facile que de constituer une image en perspective d'une scène en plaçant les uns à côté des autres les représentations en perspective (toutes plates) de tous les triangles qui la constituent. Un logiciel de 3D fait tout simplement cela sur l'écran – plat – de votre ordinateur.

Bien entendu, si le polygone que l'on veut représenter a plus de trois côtés, il suffit de le découper en triangles pour se retrouver dans les conditions simplissimes du problème précédent. C'est très exactement ce que fait la 3D maillée : pour afficher un rectangle, elle juxtapose deux triangles ; pour représenter un pentagone, elle en assemble trois, et ainsi de suite.

FIG. 3.2 – *Pour la 3D maillée, tout est triangles. Pour représenter un polygone, il suffit de le découper.*

Tout cela serait parfait si les triangles n'avaient pas le défaut d'être... anguleux, pointus. Cela ne signifie pas que la 3D maillée est incapable de représenter des formes lisses et courbes. Il est toujours possible de disposer avec régularité une multitude de petits triangles adjacents pour donner l'illusion d'une forme

courbe. Cela dit, il n'y a pas de miracle : plus ces triangles seront nombreux, plus l'ordinateur devra travailler pour les représenter. On a parfaitement le droit de vouloir intégrer des formes courbes à une scène, mais il faut bien comprendre qu'à nombre de triangles égal, un objet anguleux est toujours beaucoup plus riche en détails visuels qu'un objet courbe : à la limite, la représentation d'une simple bille parfaitement lissée pourrait demander autant de triangles – et donc autant de puissance – que celle d'un immeuble de cinq étages. C'est la raison pour laquelle la 3D maillée a une prédilection très nette pour la représentation des motifs architecturaux, aux angles bien nets.

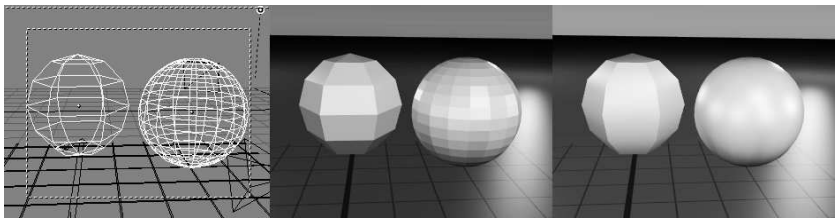


FIG. 3.3 – *Deux approximations de sphères, comportant respectivement 32 et 290 sommets. En 3D maillée classique (au centre), les variations d'éclairage permettent facilement de distinguer les facettes, mais certains algorithmes de Blender ou de POV peuvent atténuer ce défaut (à droite).*

Résumons-nous : en 3D maillée, tout est triangles. Autrement dit, comprenez ce qu'est un triangle et la 3D maillée n'aura plus aucun secret pour vous !

Note
Pour être tout à fait honnête, ce que nous disons là est une simplification un tout petit peu abusive : pour une opération bien particulière (la division fractale), certains logiciels de 3D, dont Blender, raisonnent sur de vrais quadrilatères (plats de préférence), sans les découper en deux triangles. Ne tremblez pas devant cette terrible audace conceptuelle, l'ordinateur la gérera pour vous.

3.2 Les coordonnées cartésiennes

Si vous savez ce qu'est un triangle, vous avez tout compris à la 3D maillée. Enfin presque... Il vous reste à apprendre comment les triangles se présentent dans la mémoire de l'ordinateur.

Rien de bien sorcier. Pour les logiciels de 3D, chaque triangle est défini par ses trois sommets, et chacun de ces points se voit affecter un numéro arbitraire. En très gros, le rôle des logiciels de 3D est de définir de grandes quantités de points en

3D, de leur affecter des numéros, puis d'afficher à la demande la représentation en perspective de n'importe quel triangle – lui-même défini par les numéros de chacun de ses trois sommets. C'est beaucoup de travail pour ce malheureux logiciel (il y a couramment des milliers de points même dans des scènes élémentaires), mais ça n'a rien de très compliqué.

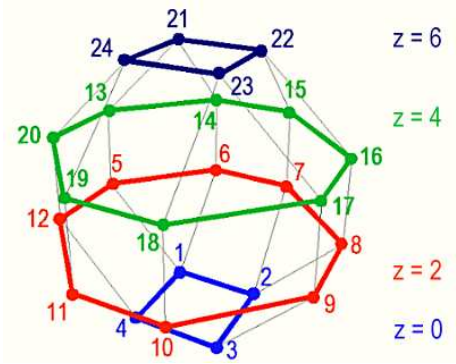


FIG. 3.4 – *Numérotation arbitraire des sommets d'un volume (ici, de bas en haut).*

Bonne nouvelle : tous les logiciels de 3D, passés, présents et à venir, sont peu ou prou capables de travailler de cette façon – et surtout d'enregistrer leurs données selon cette logique. C'est cette circonstance qui fait de la 3D maillée la technique la plus standard et la plus portable, et permet de faire transiter sans trop de peine des données d'un logiciel à l'autre. En particulier, cela nous permettra de faire travailler ensemble Blender et POV-Ray, qui n'ont pas du tout été conçus pour cela. La 3D maillée est un peu aux logiciels de 3D ce que le format « texte seul » est au traitement de texte : le moyen le plus simple de faire transiter des données d'un logiciel à un autre.

Quant aux points eux-mêmes, ils sont définis sans surprise par des coordonnées cartésiennes X, Y et Z. On ne s'attendrait pas à ce qu'une méthode aussi classique puisse donner lieu à des incompatibilités, mais ce serait compter sans les spécificités culturelles anglo-saxonnes : pour un anglo-saxon, le plan XY est généralement vertical (et c'est ainsi que fonctionne POV-Ray), tandis qu'il est horizontal pour la quasi-totalité du continent européen (et donc, en particulier, pour Blender). De minuscules différences de philosophies mathématiques ou informatiques peuvent donc contraindre à écrire des convertisseurs du type de ceux que nous avons placés sur le CD-ROM joint à ce livre.

3.3 Le stockage et l'échange de données

Voilà qui nous amène à la question du stockage de tout le travail que vous effectuerez en 3D. Le plus simple paraît de se contenter d'enregistrer ses données tout bêtement au format du logiciel qu'on utilise (en l'occurrence Blender, ou POV-Ray). Eh bien, c'est une très mauvaise idée. On ne saurait vraiment trop recommander à l'amateur qui lit ces lignes de bien prendre conscience du problème s'il ne veut pas être amené dans quelques années à jeter purement et simplement le résultat de semaines entières de travail minutieux.

Pour un amateur de modélisme, l'élaboration d'une maquette détaillée peut tout à fait être étalée sur une période assez longue (des mois, parfois des années) – durant laquelle la planète informatique ne s'arrête pas de tourner. Pendant ce temps, les logiciels de 3D sont revus de fond en comble (ou tombent dans l'oubli), et il en va de même pour les cartes vidéo, les interfaces graphiques, les systèmes d'exploitation... Par ailleurs, la machine sur laquelle vous travaillez peut tomber définitivement en panne à tout moment, et tôt ou tard vous aurez envie d'en acheter une meilleure – qui sera forcément très différente.

Personne n'y peut rien : la technique évolue sans cesse. Même si par un improbable miracle des raisons techniques de remettre en cause les standards cessaient d'apparaître à tout moment, on peut faire toute confiance à l'économie informatique pour prolonger la remise en cause perpétuelle des méthodes : il faut bien faire marcher le commerce ! Bref, on peut tenir pour certain que de nombreuses techniques de 3D informatisée couramment utilisées aujourd'hui seront modifiées ou au moins marginalisées à très brève échéance.

A cet égard, il faut faire mention du défunt format VRML (pour *Virtual Reality Modeling Language*). Ce format de stockage et d'échange de données en 3D avait fait l'objet – excusez du peu – de conventions internationales et d'une standardisation ISO à l'échelle planétaire ; la presse informatique en a fait ses choux gras pendant plusieurs années ; c'était devenu un standard si incontestable que les navigateurs Internet de modèle banal (Netscape et Internet Explorer) s'étaient crus obligés de se doter dans leur version standard de la capacité d'afficher des données 3D enregistrées en VRML... Ça n'empêche pas que ce format on ne peut plus officiel est aujourd'hui aussi oublié que les techniques militaires de la bataille d'Azincourt et d'usage aussi courant que le rouet et la lampe à huile. Tous ceux qui avaient accordé une confiance aveugle à ce format – et sur l'ensemble de la planète, ils sont probablement des dizaines de milliers – possèdent maintenant sur leurs disquettes de sauvegarde des montagnes de documents informatiques très largement inexploitables, même en ayant recours à des convertisseurs (par parenthèse, les fonctions d'import-export VRML dont Blender est censé être doté sont aussi indigentes qu'obsolètes).

Le format de stockage 3D complet, standard et pérenne, c'est comme la fourmi

de dix-huit mètres avec un chapeau sur la tête : ça n'existe pas, ça n'existe pas. Un jour ou l'autre, que vous le vouliez ou non, vous aurez besoin d'adapter vos données 3D à l'évolution de la technique et de la mode informatiques. Disons les choses clairement : un jour ou l'autre, vous aurez besoin d'abandonner Blender et POV-Ray pour passer à autre chose.

Je sais bien que s'ils lisaient cela, les développeurs de ces remarquables logiciels seraient légitimement exaspérés par l'arrogance et le mépris que je parais manifester à leur égard. Mais ce livre est écrit pour des débutants complets en 3D informatisée, auxquels je crois ne pas avoir le droit de masquer une terrible réalité : quand les groupies de la 3D vous assurent avec la plus entière conviction que la technique qu'ils préconisent constitue désormais un standard incontestable, voué à se maintenir indéfiniment... c'est une vaste blague. VRML avait été l'objet d'une véritable standardisation on ne peut plus officielle, il jouissait d'une popularité bien supérieure à celle dont Blender ou POV-Ray bénéficient actuellement, et ça n'empêche pas que VRML est mort et enterré. J'aimerais me tromper, mais je suis à peu près certain que Blender et POV-Ray mourront eux aussi – comme d'ailleurs les produits commerciaux coûteux auxquels ils se mesurent et que l'amateur aurait donc grand tort d'adopter !

Est-ce à dire que tous les efforts qu'on peut consacrer à la 3D sont voués à partir en fumée à très brève échéance ? Absolument pas, et même bien au contraire. Ce qui est merveilleux avec la 3D, c'est qu'elle utilise des principes mathématiques qui n'ont pas changé d'un iota depuis Euclide et Pythagore. Si l'on choisit comme l'auteur de ce livre de se limiter aux effets 3D qu'il est facile de définir selon les termes de la géométrie classique (autrement dit, à la 3D maillée), la conversion des données d'un logiciel à un autre est un exercice relativement simple. Internet est plein de petits logiciels gratuits (voire libres) qui effectuent cette conversion sans trop de peine – tant qu'on se limite à la 3D maillée classique, dont parle ce livre. Dès qu'on en sort, en revanche... c'est l'hécatombe.

L'écriture de convertisseurs de 3D maillée est d'ailleurs tout à fait à la portée d'un programmeur amateur, comme l'auteur. Si le lecteur que vous êtes se croit incapable de s'y attaquer lui-même, il se sous-estime totalement, mais cela n'a aucune importance : Blender et POV-Ray sont largement assez répandus sur la planète informatique pour qu'on puisse facilement trouver en fouillant un peu sur Google une foule de convertisseurs efficaces vers tous les logiciels de 3D passés, présents et à venir... Mais autant on peut compter sur de tels convertisseurs tant qu'il s'agit de 3D maillée, autant il serait déraisonnable d'espérer récupérer avec eux les effets optiques spectaculaires dont les fanatiques de la 3D sont, bien à tort, si friands.

L'auteur de ce livre, lui, ne s'est même pas donné la peine de chercher des convertisseurs de 3D maillée sur Internet : il les a écrits lui-même – parce que c'est vraiment très facile. C'est ce qui lui a permis de faire transiter ses données

3D sans jamais perdre un octet d'AutoCAD LT à Ray Dream Designer, de Ray Dream Designer à trueSpace, de trueSpace à divers utilitaires VRML, de VRML à POV-Ray et de POV-Ray à Blender – et ce n'est sûrement pas fini. A force de pratiquer l'exercice, et en s'inspirant de quelques grands principes informatiques (ceux des langages SGML), l'auteur a fini par définir son propre format d'échange, dont la principale vertu est d'être très facile à traduire (accessoirement, il permet aussi de simplifier de nombreuses manipulations). C'est en particulier ce format que nous utiliserons pour passer de Blender à POV-Ray, et aussi pour faire avaler à ces deux logiciels des formes dessinées à main levée.

Nous ne saurions trop recommander au lecteur de toujours faire l'effort d'enregistrer ses données à ce format d'échange, dont nous découvrirons les spécifications au prochain chapitre. Rien ne lui interdit d'effectuer en parallèle des copies de sauvegarde dans un format plus complexe, s'il l'estime fiable, mais notre format d'échange a une énorme vertu : il est extrêmement basique, totalement incapable d'aller au-delà de la 3D maillée classique... donc tout ce que vous parviendrez à enregistrer avec lui sera facile à récupérer avec les logiciels de 3D que vous serez amené à utiliser dans le futur. Comprenez-le bien : hors de la 3D maillée classique sur laquelle ce livre a choisi de se focaliser, vos données sont vulnérables, mortelles – et pour tout dire, condamnées. Cette vérité n'est pas agréable à entendre, mais il serait irresponsable de ne pas vous le signaler.

3.4 Image de synthèse et temps de réponse

S'il est relativement facile d'afficher des triangles en perspective sur la surface plane de l'écran de l'ordinateur, cela n'est pas suffisant pour qu'une image ainsi produite soit intelligible, pour deux raisons.

En premier lieu, les objets représentés peuvent être placés exactement sur la même direction par rapport à l'observateur, mais à des distances différentes. Les logiciels de 3D doivent donc définir des logiques pour que dans ce cas les objets les plus proches masquent les plus lointains. Ils y parviennent sans grande difficulté, il faut juste remarquer que cette représentation en « faces cachées » n'est généralement pas la plus pratique pour mettre en place une scène, et qu'en cours d'élaboration, on lui préfère généralement le mode « fil de fer » où tous les triangles d'une scène sont représentés par leurs seuls traits de contour, en se chevauchant au besoin. L'utilisateur a ainsi la faculté de voir au travers des murs jusqu'à ce qu'il soit parfaitement satisfait du positionnement des objets (accessoirement, le mode « fil de fer » demande relativement peu de calculs informatiques et permet donc des affichages quasi-instantanés même sur des scènes très riches en détail).

En second lieu, si tous les triangles constituant la représentation de l'objet sont

représentés à l'écran rigoureusement avec la même nuance de couleur, l'ensemble de ces triangles apparaîtra comme une simple silhouette de couleur plate. Il est indispensable pour la lisibilité que les objets représentés soient éclairés de façon non uniforme, les côtés de l'objet placés dans la pénombre étant par conséquent représentés avec des nuances plus sombres que ceux situés en pleine lumière.

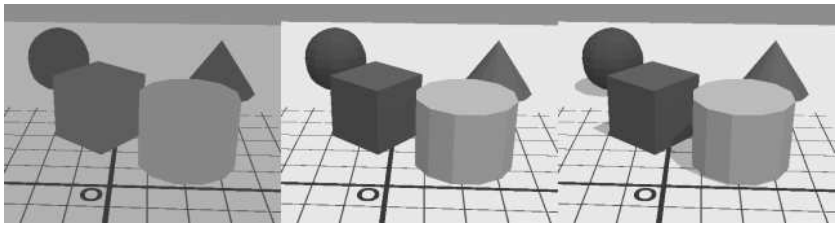


FIG. 3.5 – *A gauche, avec un éclairage totalement uniforme, les objets apparaissent comme des silhouettes plates. Au centre, les variations d'éclairage ajoutent déjà du relief. A droite, les ombres portées achèvent de rendre l'image compréhensible.*

Cette nécessité de définir un éclairage irrégulier et adapté à la scène a une conséquence dont les débutants sont rarement conscients : l'éclairage par défaut défini par le logiciel est généralement inadapté et souvent même inexistant. Par conséquent, vos premiers essais d'images de synthèse risquent de ne produire que des images illisibles, voire entièrement noires.

Par ailleurs, il faut bien comprendre que la lumière fictive employée par les logiciels de 3D est une pure abstraction mathématique, et qu'elle diffère de la lumière réelle par bien des points. En tout premier lieu, il faut savoir que les rayons lumineux fictifs sur lesquels raisonnent les logiciels de 3D sont très comparables à des balles de fusil : ils se propagent en ligne droite et transpercent les objets qu'ils rencontrent (fictivement, bien sûr : ils n'y percent pas de trous !), de sorte qu'une lampe informatique située à l'intérieur d'une maison aux volets clos peut tout à fait éclairer le jardin ! Variante : il peuvent interrompre définitivement leur course en s'écrasant sur ces objets, auquel cas l'image de synthèse comportera des ombres portées particulièrement nettes dans les zones épargnées par le mitraillage lumineux. On parle de technique de lancer de rayons, et ce n'est vraiment pas un hasard : les modèles mathématiques employés s'inspirent bien plus de la balistique que de l'optique.

La lumière informatique-balle de fusil n'est pas réellement capable de se comporter comme la véritable lumière : elle ne sait pas éclater lorsqu'elle rencontre un obstacle pour se répartir, atténuée, sur les objets environnants. Bref, les éclairages informatiques sont par nature violents et non tamisés. Dans la réalité, il suffit de

craquer une simple allumette dans une pièce aux murs clairs pour que tous les objets qui s'y trouvent soient décelables. Rien de tel en informatique, où les scènes les plus élémentaires ont couramment besoin d'une demi-douzaine de sources lumineuses (jamais moins de deux, pratiquement jamais moins de trois).

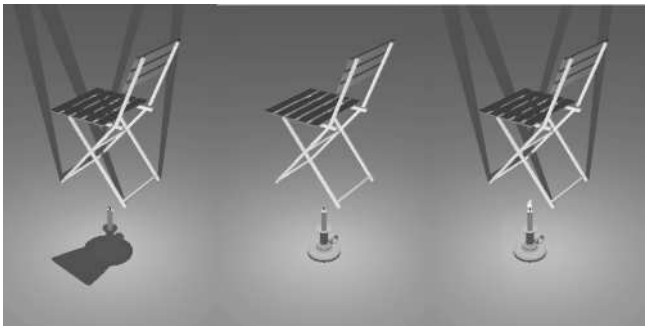


FIG. 3.6 – *A gauche, représentation avec ombres portées ; notez que la source de lumière elle-même (censée se trouver à la pointe de la mèche de la bougie) ne produit aucune lumière à son emplacement, mais projette cependant l'ombre très disgracieuse de la bougie. Au milieu, représentation avec des rayons « transperçants », sans ombre portée. A droite, un montage photographique des deux autres images atténue les défauts ; la flamme de la bougie a été ajoutée manuellement par retouche photographique.*

Beaucoup d'amateurs de logiciels de 3D détestent tellement ces défauts de la lumière informatique qu'ils consacrent toute leur énergie à les contourner, les atténuer, les dissimuler, les travestir, en employant des techniques informatiques toutes plus complexes et exigeantes en puissance les unes que les autres. Disons pour schématiser que ces tricheries consistent principalement à multiplier à l'infini les rayons lumineux-balles de fusil en déviant légèrement leur trajectoire normalement rectiligne. POV-Ray a d'ailleurs bâti l'essentiel de sa réputation sur sa capacité théorique à représenter les effets d'optique les plus complexes (diffraction, flou, brouillard, reflets, transparences, irisations, moirage...). Cela permet certes de créer l'illusion d'une lumière naturelle... mais au prix d'une multiplication absolument exponentielle des temps de calcul des images de synthèse. Lorsque l'on s'intéresse à ces effets complexes, il n'est pas du tout rare que le calcul d'une petite image comportant deux malheureuses sphères demande déjà des dizaines de minutes !

On a certes le droit de se passionner pour cet exercice – et c'est en effet ce que font la plupart des amateurs de POV-Ray. Mais il est pratiquement impossible d'employer ces techniques sur l'ordinateur de M. Tout-le-monde sans se limiter à des formes mathématiques simples : sphères, cylindres, à l'extrême rigueur an-

neaux et tores. Dès lors que l'on se propose, comme ce livre, de représenter des objets inspirés du monde réel et donc riches en petits détails, ces techniques dites photoréalistes deviennent très vite gravement handicapantes. Et si l'on souhaite, là encore comme ce livre, s'autoriser des incursions dans le domaine de l'animation (qui exige le calcul de 25 images par seconde de vidéo diffusée), le photoréalisme est à bannir absolument, en tout cas en l'état actuel de la technique – à moins de disposer comme les studios Pixar de fermes de calculs composées de centaines d'ordinateurs ! Dans un ouvrage d'initiation comme celui-ci, nous nous limiterons à produire des images parfaitement lisibles et à atténuer les laideurs insupportables. Libre à chacun de nos lecteurs de consacrer ensuite l'énergie qu'il souhaitera à la poursuite de l'utopie photoréaliste.

3.5 Le coût humain

Nous venons de parler des moyens techniques employés par l'industrie du cinéma pour produire des images de synthèse. Ce n'est pas seulement grâce à eux que les films d'animation parviennent à atteindre une qualité d'image assez inaccessible pour l'amateur. Cela tient aussi et même surtout au fait que ces compagnies sont de véritables armées, où des équipes de dizaines, voire de centaines de personnes travaillent pendant des mois sur un projet unique. Ne vous bercez pas de l'illusion que votre talent et votre persévérance vous permettront d'atteindre la même qualité : faute de pouvoir vous aussi investir des années-homme sur un projet, il vous faudra bien placer la barre notablement moins haut.

Cela ne signifie nullement que l'amateur doit renoncer d'avance à produire des images époustouflantes. Il est de fait que beaucoup de spécialistes autoproclamés de la 3D ne vont jamais très au-delà du simple empilement de cylindres et de cônes – et s'ils s'en contentent, ma foi, tant mieux pour eux. Ce livre, lui, vise à vous faire comprendre que certains types d'images de synthèse très complexes et riches – et en particulier, la maquette d'architecture – sont réellement à la portée d'un amateur isolé ne consacrant à la discipline que quelques heures par semaine. Feuilletez ce livre : tout ce que vous y verrez a été produit par un amateur travaillant seul sur un ordinateur de modèle courant, et à titre de simple passe-temps. Non seulement vous pourrez en faire autant quand vous aurez achevé l'étude de ce livre, mais votre travail sera considérablement accéléré par les petits utilitaires inclus sur le CD-ROM joint. Ces programmes sont élémentaires et sans prétention, ils n'en sont pas moins parfaitement adaptés aux besoins d'un amateur. Et vous, vous pourrez les utiliser directement, sans avoir comme l'auteur à consacrer de longs mois à les mettre au point...

Chapitre 4

Les formats d'échange

4.1 Formats propriétaires et ouverts

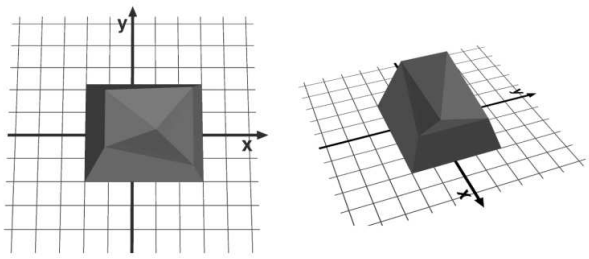


FIG. 4.1 – *Un objet de 3D maillée très simple, vu sous deux angles. Un bon format 3D permet de décrire avec une extrême précision n'importe quel volume de ce type, sous la forme d'un code informatique aussi bref et intelligible que possible.*

Chaque logiciel de 3D enregistre les données qu'il élabore à un format qui lui est propre. En l'absence de standardisation, cela n'a a priori rien que de très naturel : ce format spécifique est évidemment en parfaite adéquation avec les méthodes spécifiques employées par le logiciel – et qui font l'intérêt de ce dernier. Mais si la cacophonie des formats résulte en partie de la nature des problèmes informatiques, elle est très souvent aggravée par une politique commerciale détestable. En effet, si les données mises au point par un logiciel ne peuvent être manipulées que par lui, cela donne à son éditeur l'assurance que ses utilisateurs y réfléchiront à deux fois avant de l'abandonner au profit d'un autre, y compris si cet autre est bien meilleur... car cela signifierait qu'ils cesseraient de pouvoir prolonger tout le travail entamé précédemment : changer de logiciel signifie bien

souvent, au moins pour le profane informatique, devoir renoncer à exploiter ses propres données. C'est pour cette raison qu'un format informatique qui a été rendu sciemment cryptique et intraduisible par ses concepteurs est appelé propriétaire.

Au contraire, un logiciel aux pratiques commerciales honnêtes laisse à ses utilisateurs la possibilité de l'abandonner, en enregistrant ses données à un format compréhensible, sinon par le premier venu (en tout état de cause, l'exploitation des données informatiques demande toujours une certaine compétence), du moins par le premier informaticien venu – même pas très qualifié, même amateur. Un format d'enregistrement permettant sans trop de peine de faire passer des données d'un logiciel à un autre est appelé un format ouvert.

La 3D maillée étant fondée sur des principes simplissimes (géométrie euclidienne et coordonnées cartésiennes), elle peut a priori tout à fait être enregistrée dans des formats ouverts. De fait, il en existe de très nombreux, aux intitulés ésotériques : AutoCAD DXF, VideoScape OBJ... Aucun ne peut être considéré comme un standard véritable, mais tous se ressemblent et peuvent sans trop de difficulté être employés pour le stockage des données de 3D maillée. Pour faciliter le travail des informaticiens, tous ces formats sont lisibles avec un éditeur de lignes banal.

Les éditeurs de lignes
Le Bloc-Notes de Windows est l'éditeur de lignes le plus répandu... et incontestablement l'un des plus calamiteux ; l'amateur de 3D sera bien inspiré de lui préférer le bon vieil Edit hérité de MS-DOS (qui fonctionne dans la fenêtre de l'invite de commandes), éventuellement le shareware UltraEdit ou, bien mieux même s'il est plus complexe, le logiciel libre Emacs ; il en existe bien d'autres.
Nous préférons parler d'éditeurs de lignes plutôt que d'éditeurs de texte, pour bien marquer la différence avec les logiciels de traitement de texte (comme par exemple Word). En effet, les automatismes de ces logiciels, pensés pour la bureautique, sont une vraie calamité pour la manipulation du code informatique.

Dans un format ouvert de 3D maillée, les points sont toujours exprimés par trois nombres à virgule (plus exactement à point décimal, selon la tradition anglo-saxonne) correspondant aux coordonnées cartésiennes X, Y et Z. Ces formats numérotent les points 3D (explicitement, ou implicitement en fonction du numéro de la ligne correspondant à chacun), et définissent ensuite les volumes comme des assemblages de polygones, chacun de ces derniers étant lui-même défini par l'inventaire des numéros des points qui constituent son périmètre (voir le chapitre précédent ; on peut aussi brûler les étapes et jeter un œil sur la fig. 4.3).

Là s'arrête l'uniformité. Dans le détail, les différences de syntaxe sont aussi nombreuses que minimes : selon les cas, les deux premières coordonnées exprimées définissent un plan horizontal ou vertical ; ces coordonnées sont séparées par des espaces, des sauts de ligne, des virgules, des points-virgules ou d'autres

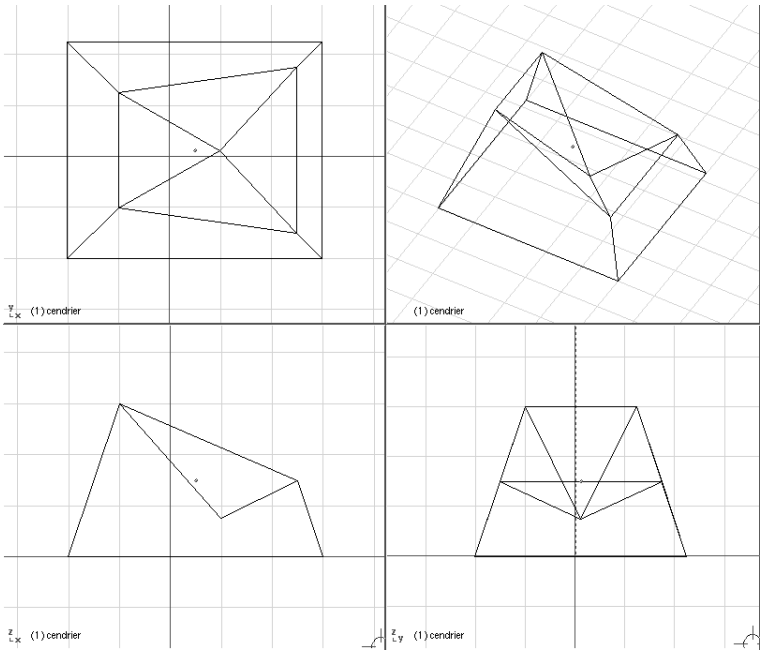


FIG. 4.2 – *Le même objet dans la représentation classique des logiciels de 3D : une perspective axonométrique (en haut à droite) et trois projections disposées en L (en haut à gauche, vue de dessus ; en bas à gauche, vue de face ; en bas à droite, vue de profil).*

signes cabalistiques ; l'unité de mesure est le mètre, le pouce, ou autre chose ; la numérotation des points démarre à 0 ou à 1, elle peut atteindre des numéros d'ordre très élevés (une scène 3D comporte couramment des dizaines de milliers de points) ou être structurée pour repartir souvent à zéro. Plus complexe, les manipulations d'objets (déplacements, rotations, mises à l'échelle) peuvent être supprimées, simplifiées, ou énumérées dans un ordre plus ou moins logique, avec des valeurs angulaires exprimées en degrés ou en radians, voire par des calculs matriciels.

Enfin, une certaine fantaisie règne dans les procédés de définition des couleurs, même si les informaticiens ont une tendresse marquée pour la logique de la synthèse additive rouge-vert-bleu des écrans de télévision – celle-là même à laquelle les êtres humains normaux n'ont jamais rien compris sans absorber des doses massives d'aspirine (d'aucuns prétendent même que c'est quand on a cherché à la lui expliquer que Van Gogh, de rage, s'est tranché l'oreille...).

L'accumulation de ces difficultés peut remplir le profane d'effroi, mais on aurait tort de s'en faire une montagne : pris individuellement, chacun de ces problèmes (sauf peut-être celui de la couleur) serait jugé totalement élémentaire par

n'importe quel étudiant en première année d'informatique. Il faut seulement retenir ceci : les données de 3D maillée peuvent en théorie passer assez facilement d'un logiciel à un autre... mais en pratique, chacun de ces passages est irréalisable à moins d'être effectué à l'aide d'un programme de traduction qu'on appelle un convertisseur (on dit aussi, moins pompeusement, une moulinette). Par ailleurs, il reste à se demander à quel saint ou à quel format d'échange se vouer pour réduire au minimum les problèmes de traduction de données.

Le format d'enregistrement .pov de POV-Ray est relativement compréhensible, d'autant que ses spécifications sont disponibles sur le Web dans une documentation aussi claire que complète. Il peut donc être regardé comme un format ouvert, mais son ambition est de produire des images et non de stocker des données géométriques. En particulier, il ne stocke pas à proprement parler des maillages 3D, mais seulement des triangles indépendants. Il est tout à fait possible et même facile de traduire au format de POV-Ray des données de 3D maillée élaborées par ailleurs – c'est la façon normale d'utiliser POV-Ray, et c'est effectivement ce que nous ferons –, mais ce format est inadapté pour l'archivage de la 3D.

Le format .blend de Blender n'est assurément pas ouvert, c'est même à vrai dire un modèle d'illisibilité (ce qui n'est sans doute pas sans rapport avec le fait qu'historiquement, ce logiciel a été conçu comme un produit commercial). Même si Blender est aujourd'hui devenu un logiciel libre, il n'a encore que partiellement corrigé ce défaut. Le moyen le plus commode d'échanger des données avec lui reste d'employer un format assez peu répandu, VideoScape .obj. Ce dernier est facilement compréhensible et convient assez bien pour la 3D maillée (en revanche, il n'est guère utilisable pour prendre en compte les fonctions avancées de Blender, et cette impossibilité d'archiver proprement son travail fait que nous préférons généralement ne pas y toucher).

Bref, il n'existe malheureusement pas de standard véritable pour l'enregistrement des données de 3D maillée. L'auteur de cet ouvrage, qui a dû au moins à quatre reprises changer de logiciel de 3D au cours des quinze dernières années, a donc dû apprendre à contourner l'obstacle en écrivant ses propres convertisseurs – ce qui, nous l'avons déjà dit, n'est nullement un exploit : c'est tout à fait à la portée de programmeurs débutants. Ce livre profite de cette expérience, et est accompagné d'un CD-ROM contenant une série de programmes de conversion certes élémentaires, mais parfaitement adaptés aux tâches que nous nous proposons d'effectuer.

Ces convertisseurs ne font pas transiter directement les données de Blender à POV-Ray. Ils passent par un format d'échange intermédiaire, évidemment ouvert, et dont nous allons maintenant indiquer les spécifications. En dépit des apparences, le fait d'avoir ajouté cette étape va considérablement simplifier notre travail et non l'alourdir, et se révéler utile pour bien d'autres usages que le simple

transfert de données 3D d'un logiciel à un autre. Car ce format d'échange d'une simplicité biblique a une vertu que ni AutoCAD DXF, ni Videoscape OBJ, ni VRML, ni aucun autre format bizarroïde ne possèdent : il permet de comprendre parfaitement ce que nous faisons.

4.2 Le format SGML employé par cet ouvrage

4.2.1 Remarque : une facilité, non une obligation

Ce que nous allons exposer dans les quelques pages qui suivent risque d'être jugé fastidieux par certains, qui ne manqueront pas de se demander s'ils ne pourraient pas purement et simplement se contenter de sauter ce passage. Il serait sans doute plus pédagogique d'affirmer le contraire, mais c'est pourtant un fait : la plupart des amateurs de 3D (surtout les mauvais, d'ailleurs) ne se sont jamais préoccupés de ces questions, et parviennent à élaborer des images de synthèse rien qu'avec un logiciel de 3D comme Blender, sans jamais poser les yeux sur une ligne de code informatique. C'est vrai, on peut faire pas mal de choses intéressantes en 3D rien qu'à la souris (même si dans ce cas, il faudra se contenter de Blender et renoncer à le marier avec POV-Ray).

Une autre attitude concevable serait d'utiliser le format défini ici pour le stockage et l'échange de données, mais sans prendre le risque de modifier ne fût-ce qu'une virgule dans les lignes de code produites automatiquement par les convertisseurs inclus sur le CD-ROM joint. Ce serait déjà moins bête... mais il est vraiment souhaitable d'aller un peu plus loin.

Nous considérons que c'est une erreur grossière que de refuser d'accomplir le moindre effort pour comprendre un peu de technique élémentaire d'informatisation de la 3D, alors que de minuscules modifications d'une instruction judicieusement choisie sont souvent suffisantes pour accomplir des tâches effroyablement minutieuses et gourmandes en temps avec d'autres méthodes. Il faut bien comprendre qu'il ne s'agit en aucune façon de dessiner intégralement des objets avec un éditeur de texte, mais seulement de modifier quelques paramètres de ci de là pour obtenir des effets précis. Par exemple, dans n'importe quelle scène mise au point par l'auteur pour les illustrations de ce livre, la part des lignes de code qu'il a rédigées manuellement ne doit jamais dépasser 0,5 % : les modifications manuelles dans le code informatique concernent généralement une douzaine de lignes au niveau le plus élevé d'une hiérarchie de fichiers – qui peut, elle, comporter couramment des dizaines de milliers de lignes de code résultant toutes de conversions automatiques, et traduisant des opérations réalisées avec des outils beaucoup moins rustiques qu'un éditeur de texte (en particulier, des manipulations effectuées à la souris avec Blender).

C'est vraiment un tort que d'opposer les manipulations manuelles et l'élaboration du code informatique : les deux techniques se conjuguent très utilement, et de façon particulièrement aisée dans le cas de la 3D. Il n'est vraiment pas exagéré de dire que l'amateur de 3D capable de consacrer ne serait-ce que 5 % de son temps, de sa compétence et de son énergie à de minuscules interventions dans le code informatique 3D atteint tout de suite une rentabilité deux ou trois fois supérieure à celle de ceux qui ont la sottise de s'y refuser totalement. Il est de bonne politique de n'avoir recours aux manipulations de code informatique que de façon très marginale, mais il est vraiment dommage et inintelligent de s'interdire totalement et définitivement d'y avoir recours.

Donc, ne baissez pas les bras. L'auteur de ces pages n'est qu'un amateur pas particulièrement compétent ni motivé, et s'il a pris la peine d'étudier à fond ce qui suit, c'est parce que l'expérience lui a appris que cela lui permettait de s'épargner beaucoup de travail inutile. Ne sous-estimez pas vos capacités intellectuelles, accrochez-vous ne serait-ce que quelques minutes. Dans le pire des cas, vous allez seulement vous ennuyer un peu en lisant ce qui suit, et personne n'en est jamais mort. Peut-être ne comprendrez-vous pas immédiatement l'intérêt de tout cela, mais il est quand même bon de savoir que certaines possibilités existent. Si vous avez lu ce chapitre, vous penserez certainement de vous-même à y revenir quand votre apprentissage de Blender vous aura fait comprendre à quel point certaines opérations apparemment triviales sont imprécises et lourdes à mettre en œuvre avec une souris, alors qu'elles peuvent si facilement être réalisées de façon tout à fait impeccable grâce à quelques manipulations du code informatique. Par ailleurs, les fichiers 3D d'exemples enregistrés sur le CD-ROM joint font abondamment appel au format que nous définissons plus loin, et vous comprendrez tout l'intérêt de ce type de stockage quand vous vous amuserez à les modifier.

4.2.2 Cahier des charges

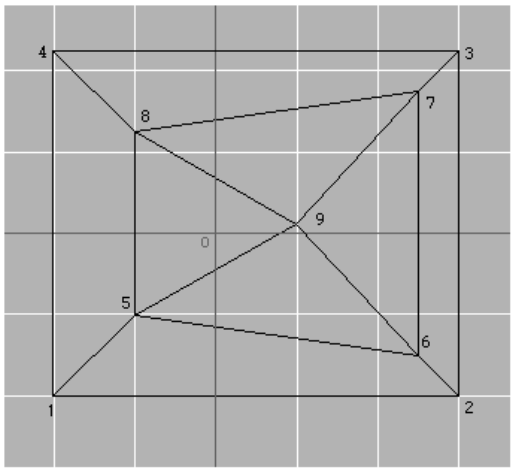
Esprit général

Le format d'échange que nous emploierons correspond évidemment aux exigences traditionnelles de la 3D maillée, en termes informatiques et géométriques. Mais surtout, il cherche à rester intelligible pour un adulte moyennement doué et motivé. Voici son cahier des charges :

– être un format ouvert (voir plus haut) ;

– être lisible (et intelligible) à l'aide de n'importe quel éditeur de lignes ;

– être assez simple pour ne poser aucun problème de compréhension à une personne dotée de capacités intellectuelles normales, sans




```
<points3D>
1, -2.0, -2.0,   0.0
2,  3.0, -2.0,   0.0
3,  3.0,  2.25,  0.0
4, -2.0,  2.25,  0.0
5, -1.0, -1.0,   3.0
6,  2.5, -1.5,   1.5
7,  2.5,  1.75,  1.5
8, -1.0,  1.25,  3.0
9,  1.0,  0.125, 0.75
</points3D>
<faces>
1,4,3,2
1,2,6,5
2,3,7,6
3,4,8,7
4,1,5,8
5,6,9
6,7,9
7,8,9
8,5,9
</faces>
```

FIG. 4.3 – *Toujours le même objet, représenté en vue de dessus. Cette fois, les sommets des polygones constituant sa surface extérieure ont été numérotés dans un ordre arbitraire. Le code informatique à droite indique d'abord les coordonnées X, Y et Z de chacun des points numérotés, puis décrit tous les polygones de la surface extérieure (y compris le rectangle constituant la base de l'objet) par un inventaire des numéros des points de leurs périmètres.*

connaissances particulières en matière de géométrie ou d'informatique ;

– en particulier, faire en sorte que tout objet 3D perçu comme un tout cohérent par le cerveau humain (un clou en 3D, une brouette en 3D, une maison de cinq étages en 3D) puisse être manipulé comme un fichier informatique unique (en d'autres termes : que chacun puisse archiver, sauvegarder et dupliquer l'objet 3D exactement comme il le fait quotidiennement quand il manipule un document de son traitement de texte ou une photo de son appareil numérique : déplacement d'icônes sur le bureau, suppression par déplacement dans la corbeille, etc.) ;

– cette dernière condition étant remplie, permettre que n'importe quel objet puisse être inclus dans un autre simplement en inscrivant dans le code informatique de l'objet englobant le nom de l'objet inclus (par exemple, le code informatique décrivant une maison entière n'aura besoin que d'une ligne de texte comportant le nom du fichier informatique décrivant une porte pour que ce dernier objet figure dans toutes les représentations 3D de la maison)...

– mais aussi pouvoir à la demande recopier intégralement le code informatique de l'objet inclus dans celui de l'objet englobant, pour faciliter les manipulations informatiques et notamment les traductions ;

– dans la mesure du possible, faire en sorte que chaque ligne du document informatique décrivant l'objet 3D corresponde à une unité logique complète (un point, un polygone, une couleur, un déplacement, ou encore, nous venons de le voir, un objet entier) ;

– être facile à traduire automatiquement à n'importe quel autre format de 3D maillée ;

– permettre la mise en couleur de façon simple et intelligible ;

– faciliter des opérations géométriques simples comme le déplacement des objets, leur rotation et même leur changement d'échelle...

– mais aussi pouvoir recalculer toutes les coordonnées après ces modifications, afin de simplifier les traitements ultérieurs (et en particulier les traductions) ;

– enfin, pour des usages particuliers, définir à côté du format 3D un format 2D à la logique très similaire.

Ouf. On pourrait résumer tout cet inventaire en une formule simple : résoudre toutes les galères informatiques rencontrées par l'auteur pendant les quinze années où il n'a cessé de faire transiter ses données d'un logiciel de 3D à un autre...

Nous allons voir dans le détail comment tout cela se traduit en syntaxe informatique. Rassurez-vous : c'est tout à fait à la portée d'un élève de collège, même très moyen – et vous n'avez pas encore idée d'à quel point cela vous facilitera le travail par la suite, au moins pour les opérations nécessitant de la précision.

Une ligne par information complète

Le format informatique que nous utiliserons pour les échanges peut être lu et modifié avec un éditeur de texte banal, et chacune de ses lignes constitue une unité logique : on emploie une ligne unique pour définir un point, un polygone, une couleur, un mouvement, une modification de coordonnées, voire l'inclusion d'un objet 3D entier (évidemment défini par ailleurs). D'autre part, et contrairement à des usages informatiques courants, notre format n'admet pas qu'une ligne commence ou se termine par des espaces ou des tabulations.

D'autres formats de 3D maillée ont opté pour un comportement différent. Par exemple, DXF emploie deux lignes pour chaque coordonnée X, Y ou Z, et il lui en faut donc six pour définir un simple point ; c'est acceptable pour une machine, mais à notre avis, cela rend le code inintelligible pour un être humain normal.

D'autres doctrines informatiques, notamment celles qui valorisent le langage XML (très en vogue aujourd'hui), préconisent de ne pas accorder de significa-

tion particulière aux retours à la ligne ni d'ailleurs aux répétitions d'espaces et de tabulations, de façon à laisser chacun décider de l'organisation visuelle de son code selon ses propres habitudes. Les partisans du langage XML font valoir qu'il existe aujourd'hui de nombreuses librairies informatiques parfaitement capables de prendre en compte cette relative anarchie. A notre sens, le cerveau humain, lui, ne peut pas s'orienter dans un code informatique complexe avec la même virtuosité, et nous tenons absolument à ce que notre code informatique reste intelligible pour des amateurs. Donc, au risque de nous faire excommunier par l'école XML, nous avons pris le parti d'organiser notre format d'échange ligne par ligne. Signalons tout de même aux excommunicateurs qu'il serait extrêmement facile de traduire notre format vers du XML standard... si un jour futur un XML standard pour la 3D est jamais défini.

Balisage de type SGML

Il est bien sûr nécessaire de pouvoir distinguer les lignes décrivant des points de celles décrivant des polygones, et de regrouper en unités logiques des sous-ensembles de lignes (tous les points servant à décrire un même objet, par exemple). Pour cela, nous ferons montre d'un minimum d'orthodoxie en utilisant les balises ouvrantes et fermantes chères aux différents langages SGML (pour *Standard Generalized Markup Language* ; le HTML des pages Web en est une variante) : la balise ouvrante est entourée des signes inférieur et supérieur (par exemple : `<balise>`), la balise fermante respecte la même syntaxe mais intercale une barre de fraction avant le nom de la balise (par exemple : `</balise>`). Attention : dans les deux cas, notre format exige que chaque balise soit disposée sur une ligne indépendante.

Il serait sans doute très prétentieux de présenter le format que nous employons comme un standard international. Cependant, SGML est une référence très connue, ancienne et respectée dans le monde informatique. L'auteur de cet ouvrage peut donc prétendre avoir fait en s'en rapprochant bien plus d'efforts de standardisation que la plupart des formats 3D plus répandus, tous enfermés dans leur glorieuse tour d'ivoire. Nous pousserons cette fierté jusqu'à parler dans cet ouvrage de « SGML 3D » pour désigner le format que nous employons. Certes, c'est abusif ou tout au moins très exagéré (les spécifications SGML comportent bien des aspects que nous avons négligés)... mais l'appellation a l'énorme avantage d'être brève.

Une même logique pour 2D et 3D

Les amateurs de 3D n'ont généralement que faire des techniques de dessin traditionnel en deux dimensions (et en particulier, sur le papier) ; à l'extrême rigueur,

ils consentiront à utiliser des programmes spécialisés dans le dessin 2D (logiciels de retouche d'images ou de dessin vectoriel), sans leur demander quoi que ce soit de spécifique. Nous pensons que c'est là une erreur très lourde de conséquences, car les outils 2D sont aussi utiles à la 3D... que totalement inadaptés à elle dans leurs versions courantes.

Le concepteur 3D a tout à fait intérêt à effectuer rapidement des brouillons de ce qu'il a en tête, et on n'a jamais rien inventé pour cela d'aussi pratique que le papier et le crayon. Il est donc fort utile de pouvoir transformer en données 3D des ébauches dessinées à main levée sur un bout de papier – ou à la rigueur à la souris sur un écran d'ordinateur.

Mais précisément, la plupart des logiciels de 2D raisonnent en référence à la feuille de papier à dessin, d'une vingtaine de centimètres de large, et sur laquelle une précision supérieure au quart de millimètre est inutile : les plus forts taux d'agrandissement gérés par les logiciels de dessin sont généralement de 1600 %... soit une multiplication des coordonnées par 16 seulement (eh oui ! l'usage du signe % est bien pratique pour donner une fausse impression de puissance). Or, grâce à la perspective, une même image en 3D peut très couramment mélanger de façon lisible des détails de l'ordre du millimètre au premier plan et de la centaine de mètres à l'arrière-plan : au lieu du minable facteur 16 des logiciels de 2D courants, il faudrait un facteur 100 000 ! Pour la 3D, les logiciels de 2D classiques manquent donc dramatiquement de précision.

Par ailleurs, nous l'avons vu, la 3D conçoit tout ce qu'elle affiche sous forme de triangles juxtaposés... alors que les logiciels de 2D utilisent soit une technique de grille de petits points de couleur (c'est ainsi que fonctionnent les logiciels de retouche d'images), soit une logique de remplissage de périmètres. En termes informatiques, tout cela est extrêmement différent de la logique de la 3D... et donc presque impossible à exploiter sans un énorme gaspillage de puissance.

Nous tournerons donc certaines difficultés en ayant recours à un format de 2D maillée, évidemment calqué sur la logique de la 3D : définition de points par leurs coordonnées, puis définition de polygones par les points de leur périmètre. Attention : bien que ces deux formats obéissent à des logiques et à des syntaxes extrêmement voisines, ils correspondent bien à des usages différents et ne sauraient donc être mélangés ; si l'on veut absolument faire figurer des motifs 2D dans une scène 3D, il faudra au préalable leur faire subir une conversion.

Bien entendu, notre format 2D poursuit des objectifs strictement utilitaires et serait très inadapté pour le dessin d'art. Il supporte sans difficulté des agrandissements d'un facteur de 1 000 000... mais en compensation, il est d'une effroyable pauvreté (on pourrait aussi dire d'une merveilleuse simplicité, c'est une question de point de vue) en matière de gestion de la couleur et des formes courbes.

Une logique séquentielle

Même si la 3D maillée est a priori tout à fait statique – tant qu'on ne s'aventure pas dans le domaine de l'animation –, le texte au format 3D qui la décrit a quant à lui un ordre précis : un début, une suite, une fin. Il obéit à une logique séquentielle, et l'interversion des instructions qu'il contient n'est pas forcément sans conséquence. Par exemple, une suite logique du type « couleur bleue, cube, couleur jaune, cône, cylindre, couleur rouge, sphère » engendre un cube bleu, un cône et un cylindre jaunes, et enfin une sphère rouge. Mais si on modifie très légèrement cet ordre en déplaçant le cylindre à la toute fin de l'inventaire, cela aura pour conséquence d'en changer la couleur : alors qu'il était jaune comme le cône qui le précédait, il deviendra rouge comme la sphère après laquelle on l'aura placé. En termes informatiques, cela n'a rien pour surprendre (chaque définition de couleur modifie le réglage « par défaut »)... mais cela peut être un peu déconcertant pour un débutant.

En revanche, l'ordre dans lequel on appelle les objets 3D constituant une scène n'a aucune influence sur leur façon de se masquer les uns les autres sur l'image de synthèse finale : ce sont évidemment les objets placés au premier plan par leurs coordonnées qui masquent les autres, et non les derniers appelés qui masquent les premiers. S'il vous prend fantaisie de définir le toit d'une maison avant son rez-de-chaussée, cela ne l'empêchera pas du tout de tenir debout !

Cela peut vous paraître évident... Ça ne l'est pourtant pas totalement, car en 2D c'est bien le contraire qui se passe : l'objet défini en dernier masque tous ceux définis avant lui, comme s'il était posé par-dessus (c'est d'ailleurs la raison pour laquelle l'interface des logiciels de dessin 2D emploie souvent des expressions comme « passage au premier plan » ou « décalage vers l'arrière », qui pour le mathématicien n'ont aucun sens quand on raisonne en deux dimensions ; en réalité, ces expressions se rapportent au chamboulement d'une liste séquentielle d'instructions informatiques).

4.2.3 Détail de la syntaxe

Les points (3D et 2D)

Les lignes permettant la définition géométrique de points en trois ou deux dimensions sont encadrées par les balises `<points3D>` et `</points3D>` ou `<points2D>` et `</points2D>`, respectivement. Leur syntaxe est la suivante : numéro du point, virgule, coordonnée X, virgule, coordonnée Y, virgule, coordonnée Z (évidemment superflue pour un point 2D). Les nombres dits « à virgule » sont en fait notés avec un point décimal, selon l'usage informatique : on note 3.1416 et non 3,1416 (attention, cela peut causer des erreurs et même des plan-

tages).

Par exemple, le point 3D numéroté 4 et de coordonnées x = 3.2, y = 6, z = -0.376 serait décrit ainsi :

```
4,3.2,6.0,-0.376
```

Soit dit en passant, nous respecterons dans cet ouvrage l'usage informatique, qui est d'ajouter un point et un zéro à la fin d'une valeur entière dans le cas où l'on attend normalement un nombre réel plutôt qu'un entier : on note 6.0 et non 6 tout court. Nous vous recommandons de faire de même car certains langages de programmation en tiennent compte.

L'unité employée pour exprimer les coordonnées est normalement le mètre.

Le numéro du premier point : 0 ou 1 ?
Pour des raisons historiques (il s'agit de l'histoire informatique de l'auteur !), l'usage est de commencer la numérotation des points à 1 quand il s'agit de 3D et à 0 quand il s'agit de 2D, mais il n'y a aucun inconvénient à démarrer systématiquement à 1 ; nous le signalons simplement pour expliquer le comportement de certains de nos convertisseurs, qui incrémentent la numérotation des points en passant de la 2D à la 3D et la décrémentent dans le cas inverse.

Les numéros des points ne peuvent en aucun cas être négatifs (ce serait le plantage assuré). Il est préférable de faire en sorte qu'ils ne soient jamais trop élevés, quitte à fractionner les objets trop complexes en morceaux (on peut cependant atteindre l'indice 1000 sans problème). Il n'y a en théorie aucun inconvénient à laisser des « trous » dans les séries de numéros, ni à définir les points du plus grand numéro d'ordre au plus petit, voire dans un désordre total ; cela dit, cet usage est généralement l'indice d'une confusion d'esprit, et il est donc fortement déconseillé.

Les polygones (ou faces)

Les lignes définissant des polygones sont encadrées par les balises `<faces>` et `</faces>`. Leur syntaxe est la suivante : numéro du premier point, virgule, numéro du deuxième point, virgule... jusqu'au numéro du dernier point, lui-même précédé d'une virgule.

Par exemple, le rectangle à la base de l'objet représenté sur la fig. 4.3 est décrit par la ligne suivante :

```
1,4,3,2
```

Dans quel ordre appeler les points d'un périmètre ?
Pourquoi 1,4,3,2 plutôt que 1,2,3,4 ? Ce n'est pas tout à fait indifférent. Certains logiciels de 3D accélèrent les traitements informatiques en ne prenant pas en compte les faces cachées d'un objet. De fait, en vue de dessus, le rectangle à la base de l'objet de la fig. 4.3 est de toute façon masqué, et il est donc inutile de s'en préoccuper lorsque l'objet est vu sous cet angle. Eh bien, le simple fait de décrire cette face sous la forme 1,4,3,2, c'est-à-dire (reportez-vous à la figure) en énumérant les points dans le sens des aiguilles d'une montre, signifie que cette face n'est normalement visible que de l'autre côté. Vous remarquerez que toutes les autres faces de la fig. 4.3 sont au contraire visibles, et donc leurs points ont été définis dans le sens géométrique (c'est-à-dire, en français courant, dans le sens inverse de celui des aiguilles d'une montre).
Deuxième question : pourrait-on écrire 4,3,2,1, voire 3,2,1,4 ou 2,1,4,3 pour décrire la même face ? En l'occurrence, cela mènerait rigoureusement au même résultat, mais là encore, ce n'est quand même pas tout à fait indifférent. Certains logiciels de 3D ne connaissent rigoureusement rien d'autre que les triangles, et vont donc diviser le polygone 4,3,2,1 en deux triangles : 4,3,2 et 4,2,1 – et le premier point cité dans l'inventaire décrivant le périmètre servira de pointe commune à tous les triangles résultant du découpage du polygone. Dans la plupart des cas, cela n'a aucune conséquence visible, mais il peut être utile d'y penser lorsque l'on constate certaines aberrations visuelles (fig. 4.4).

Soit dit en passant, il ne faut pas oublier que les polygones décrits doivent être rigoureusement plats. Si ce n'est pas le cas et que l'un des points du périmètre s'écarte du plan défini par les autres, les logiciels de 3D essaieront malgré tout de s'en sortir en découpant le polygone « tordu » en triangles plats. Il faudra y penser, par exemple, lorsque l'on utilisera les fonctions de modelage de Blender pour déformer un objet dont la surface comporte des quadrilatères (un cube, par exemple) : les quadrilatères initialement plats risquent de se découper en triangles d'une façon pas toujours très intelligible.

Les traits

Les lignes de code informatique définissant des traits sont encadrées par les balises `<traits>` et `</traits>`. Les traits ne sont a priori gérés qu'en 2D (en 3D, leur absence d'épaisseur les rendrait invisibles). Les traits sont en fait des assemblages de segments de droite, et peuvent tout à fait se tortiller pour décrire des périmètres complexes.

Par exemple, le périmètre d'un quadrilatère pourra être décrit par une ligne de ce type :

```
1,2,3,4,1
```

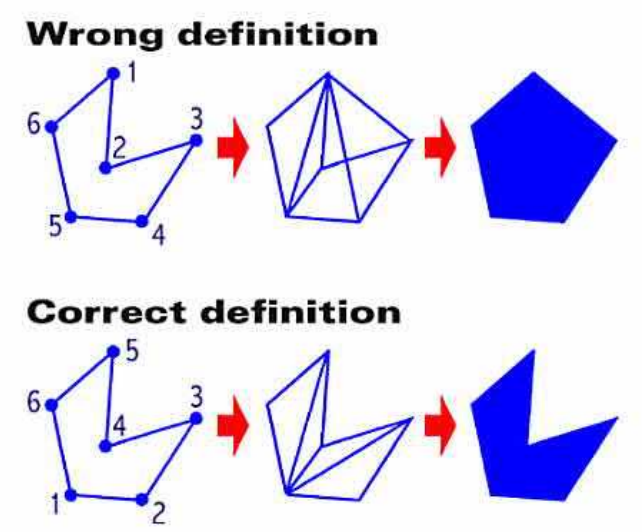


FIG. 4.4 – *Le premier point cité dans l'inventaire décrivant le périmètre d'un polygone n'est pas totalement indifférent : s'il est mal choisi, le découpage éventuel du polygone en triangles risque de mener à des aberrations visuelles.*

Notez que le point de départ est répété, faute de quoi nous n'aurions défini que trois segments (1-2, 2-3 et 3-4).

Les couleurs

Les lignes définissant des couleurs sont encadrées par les balises `<couleur>` et `</couleur>`. En fait, ces balises n'encadrent jamais qu'une ligne unique, décrivant la couleur qu'on appliquera à tous les objets décrits par la suite – à moins de rencontrer la définition d'une nouvelle teinte (c'est ce qu'on appelle un paramétrage « par défaut »).

Le format 2D que nous utilisons n'ayant aucune ambition artistique, on n'a le choix qu'entre seize couleurs ainsi nommées et orthographiées : rouge, jaune, vert, cyan, bleu, magenta ; rouge sombre, jaune sombre, vert sombre, cyan sombre, bleu sombre, magenta sombre ; noir, blanc, gris, gris sombre.

Le format 3D, lui, a vocation à exprimer toutes les nuances du cercle chromatique, des plus claires aux plus sombres et des plus agressives aux plus ternes. L'usage dans les logiciels de 3D est d'employer un modèle colorimétrique « rouge, vert, bleu » calqué sur celui qu'emploient les écrans de télévision... et incompréhensible pour le profane. Pour notre part, nous avons choisi d'exprimer les couleurs selon un modèle également abstrait, mais beaucoup plus intelligible : le modèle « teinte, saturation, niveau de gris » (en anglais : *hue, saturation, grayscale*).

Une couleur se trouve donc définie par une ligne de ce type :

```
240.0,0.67,0.75
```

Il s'agit d'un bleu pastel clair, même si cela ne vous paraît pas encore très évident ! La valeur 240.0 indique une position sur le cercle chromatique (et pas n'importe laquelle : c'est celle d'un bleu pur, ne tendant ni vers le vert, ni vers le violet). Cette valeur exprime en degrés d'angle une direction sur le cercle chromatique traditionnel ; elle est donc comprise entre 0.0 et 360.0 (ou d'ailleurs entre 0.0 et -360.0 : 240.0 et -120.0 sont deux façons également valides de désigner le bleu). La fig. 4.5 indique les valeurs pertinentes pour les couleurs primaires... et on peut bien sûr utiliser des valeurs intermédiaires pour des couleurs moins pures ; par exemple, on obtient de l'orange avec des valeurs comprises entre 45.0 et 55.0 (à peu près).

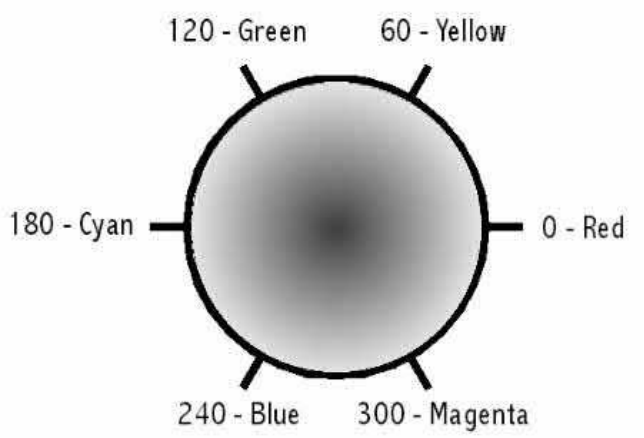


FIG. 4.5 – *Le cercle chromatique... en noir et blanc, avec les valeurs exprimant en degrés la position des couleurs primaires. Notez qu'avec le modèle « teinte, saturation, niveau de gris », aucune couleur n'est claire ou sombre par nature : le jaune n'est pas censé a priori être plus clair que le rouge ou le bleu.*

La deuxième valeur indiquée sur la ligne de code informatique que nous venons de donner en exemple, 0.67, exprime la saturation, c'est-à-dire le caractère plus ou moins agressif de la couleur. Cette valeur est comprise entre 0.0 (aucune agressivité : en d'autres termes, on obtient du gris) et 1.0 (agressivité maximale : la couleur est aussi pure que possible). Avec une valeur de 0.67, c'est-à-dire « deux tiers d'agressivité, un tiers de gris », on obtient une couleur un peu passée, pastel.

La troisième valeur de notre ligne de code, 0.75, exprime un niveau de gris, c'est-à-dire (attention, c'est un peu abstrait) le gris correspondant à la couleur si elle était photographiée en noir et blanc : à 0.0, ce serait du noir absolu, à 1.0 du blanc pur, à 0.5 un gris moyen. En l'occurrence, nous avons une valeur de 0.75,

donc un gris assez clair... ou plus exactement une couleur qui se traduirait par un gris assez clair si on la photographiait en noir et blanc.

Quelques exemples ne seront pas de trop pour y voir un peu plus clair. `60.0, 1.0, 0.85` exprime un jaune citron, vif et pur ; `0.0, 1.0, 0.8` exprime un rose fluo, aussi « agressif » qu'un rose peut l'être compte tenu de sa clarté ; `60.0, 0.0, 0.5` et `312.0, 0.0, 0.5` se traduiront tous deux par le même gris moyen, puisque dans les deux cas, une saturation à 0.0 indique que la teinte est convertie en gris ; enfin `0.0, 1.0, 0.0` ne donnera pas du rouge mais bel et bien du noir puisque la troisième valeur (celle qui exprime le niveau de gris) est nulle, et `240.0, 1.0, 1.0` ne donnera pas du bleu mais bel et bien du blanc, puisque la troisième valeur vaut 1.0 et exprime donc le niveau de gris le plus clair qui se puisse concevoir... le blanc.

Les inclusions de sous-fichiers

Les lignes indiquant le nom d'un sous-fichier à inclure (par exemple, pour inclure un fichier décrivant une chaise dans le fichier décrivant une salle à manger) sont encadrées par les balises `<inclusion>` et `</inclusion>`. Cela donnera par exemple les trois lignes suivantes :

```
<inclusion>
table.3d
</inclusion>
```

Quelques remarques. Contrairement à ce à quoi l'on pourrait s'attendre, il est préférable de ne jamais faire figurer plus d'un nom de fichier entre les balises `<inclusion>` et `</inclusion>`. Si l'on souhaite inclure en plus de la table une chaise et un lampadaire, il faudra écrire

```
<inclusion>
table.3d
</inclusion>
<inclusion>
chaise.3d
</inclusion>
<inclusion>
lampadaire.3d
</inclusion>
```

et non

```
<inclusion>
table.3d
chaise.3d
lampadaire.3d
</inclusion>
```

comme on aurait pu s'y attendre. L'honnêteté force l'auteur à admettre que c'est probablement là l'indice d'une erreur de conception de sa part, et qu'il lui faudra un jour corriger cela. Mais cela n'a presque aucune importance en pratique, car il est rarissime que le même positionnement puisse être retenu pour trois objets différents... donc il faudra inclure dans le code informatique (entre la table et la chaise, puis entre la chaise et le lampadaire) quelques lignes de code permettant de déplacer un peu ces objets par rapport au positionnement théorique qu'on leur avait imposé lors de leur définition.

Il est nettement préférable de placer tous les fichiers SGML 3D décrivant les objets d'une même scène dans le même dossier (on dit aussi répertoire) du disque dur, de façon à pouvoir les appeler par leur seul nom (`table.3d`, par exemple), sans avoir à spécifier un chemin d'accès (du type `c :\objets3d\table.3d` sous Windows, ou `/home/user/table.3d` sous Linux ou Mac OS/X). Quant à l'adressage relatif (du type `../autre_repertoire/chaise.3d`), mieux vaut considérer qu'il est interdit pour le moment (en fait, il est utilisable et même très utile, mais cette question étant un peu complexe, nous ne l'aborderons pas avant la troisième partie de ce livre).

4.2.4 Les modifications de coordonnées

Généralités

Voilà qui nous amène à la question très importante des modifications de coordonnées. La facilité et la précision avec lesquelles on peut les définir sont très certainement le principal avantage d'une codification SGML de la 3D. Nous allons donc prendre le temps d'expliquer la démarche en détail.

Les lignes définissant les modifications de coordonnées sont encadrées par les balises `<modifcoord>` et `</modifcoord>`. Plusieurs modifications de coordonnées peuvent être définies les unes à la suite des autres, chacune sur une ligne indépendante.

Il existe trois types de modifications de coordonnées : déplacement, rotation et changement d'échelle (qu'il s'agisse d'un changement de taille global ou d'une déformation en longueur, largeur ou hauteur). Le type de modification choisi est indiqué par un mot-clé de quatre lettres : « Posi » pour les déplacements (changements de position), « Rota » pour les rotations et « Tail » pour les mises à l'échelle (changements de taille).

En fait, ce que nous venons de dire est vrai pour la 2D. En 3D, il faut distinguer trois types de rotation (autour de l'axe des X, des Y ou des Z), et donc le mot-clé « Rota » est remplacé selon les cas par « Rotx », « Roty » ou « Rotz ».

Déplacement

Une instruction de déplacement aura une forme de ce type :

```
Posi 3.0,-5.0,0.2
```

Les trois paramètres qui suivent le mot-clé Posi (en 2D, il n'y en aurait bien sûr que deux) concernent, on s'en doute, une modification des coordonnées X, Y et Z, respectivement. Cette ligne d'exemple permet donc de déplacer les objets définis après elle de 3 m vers la droite, 5 m vers l'arrière, et de les soulever de 20 cm.

Comprenez bien qu'il s'agit d'un déplacement relatif et non d'un positionnement absolu : si les coordonnées de l'objet affecté par cette modification n'étaient pas 0.0, 0.0, 0.0 au départ (en d'autres termes, si l'objet n'était pas placé exactement au point O, intersection des axes), mais par exemple en -1.0, -2.0, 0.0, sa position finale sera le résultat de l'addition de ses coordonnées initiales et du déplacement ; il se retrouvera donc en 2.0, -7.0, 0.2.

Rotation

Nous prendrons pour exemple la rotation autour de l'axe des Z, qui est de loin la plus courante (pour quantité d'objets du monde réel, et en particulier les meubles, c'est la seule qui ait quelque vraisemblance). Une instruction de rotation autour de l'axe des Z aura cette forme :

```
Rotz 30.0
```

La valeur de la rotation est indiquée en degrés (ce qui est bien le signe que notre format est d'abord fait pour être lu : presque tous les formats de 3D préfèrent employer les radians, ce qui les rend rigoureusement incompréhensibles). Pour faciliter la compréhension, la valeur d'une rotation est de préférence choisie entre -180.0 et 180.0, mais il n'y aucun inconvénient à dépasser ces seuils pour atteindre -360.0 et 360.0... voire pour aller au-delà.

Le sens de rotation est le sens géométrique (l'inverse de celui des aiguilles d'une montre). Quand il s'agit d'une rotation autour de l'axe des Z, tout le monde raisonne en vue de dessus, ce qui convient parfaitement. S'agissant en revanche des rotations autour de l'axe des X ou des Y, cela peut occasionner des difficultés de compréhension selon le point de vue que l'on adopte mentalement. En particulier, sur la vue dite « de face » de Blender, une rotation positive autour de l'axe des Y se traduit à l'écran par un mouvement dans le sens des aiguilles d'une montre. C'est dû au fait que ce que l'on appelle traditionnellement la vue de face devrait plutôt être considéré comme une vue de dos (voir fig. 4.6).

On peut bien sûr enchaîner plusieurs rotations pour aboutir à des positionnements complexes. Par exemple, une voiture garée à flanc de colline peut subir un dévers de 5 ° à gauche, une inclinaison de 15 ° vers le haut, et être orientée vers le

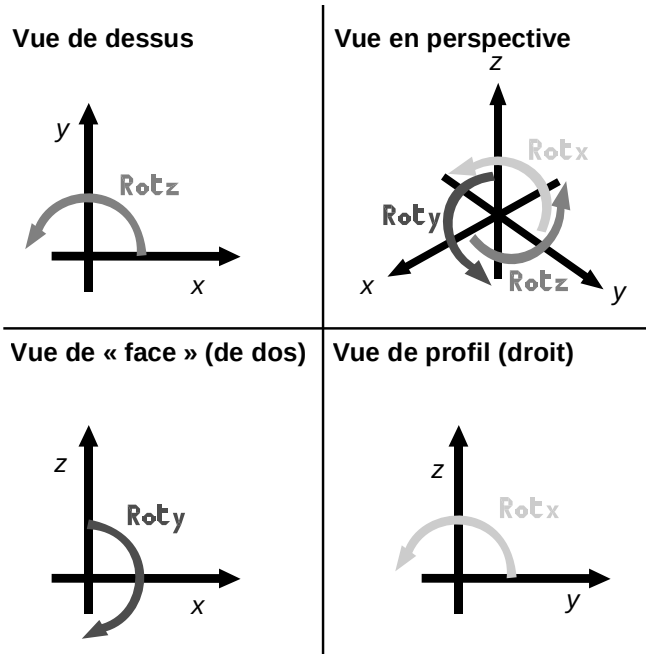


FIG. 4.6 – *Les rotations s'opèrent toujours en sens inverse de celui des aiguilles d'une montre, comme on le voit sur la vue en perspective... mais la vue en projection dite « de face » donne une impression fausse à cet égard. En fait, il serait préférable de parler de vue « de dos ».*

nord-ouest (soit 45 ° plus à gauche que le nord). Si, conformément à une habitude que nous recommandons d'adopter, la voiture a été définie à l'horizontale, avec l'avant orienté vers le nord (c'est-à-dire les valeurs les plus élevées de l'axe des Y), le plus compréhensible pour lui donner ce positionnement « à flanc de colline » sera d'indiquer d'abord le dévers (Roty -5.0 ; si cela vous étonne que cette valeur soit négative, reportez vous à la fig. 4.6), puis l'inclinaison (ou pente : Rotx 15.0), puis l'orientation (Rotz 45.0).

Cet ordre « Roty, Rotx, Rotz » est le plus naturel pour l'esprit humain... mais bien entendu les logiciels de 3D ont plutôt tendance à employer l'ordre « Rotx, Roty, Rotz » (c'est le cas de Blender... et il existe aussi des logiciels qui choisissent l'ordre inverse !). Cela peut donner le même résultat, bien sûr... mais cela oblige à indiquer des valeurs angulaires différentes, et plus difficiles à estimer mentalement car elles ne correspondent plus à des concepts relativement simples comme le dévers et la pente.

Rien n'interdit d'aller au-delà de trois rotations si on le juge plus compréhensible (en particulier dans le cas de structures articulées comme les bras d'une

marionnette).

Mise à l'échelle et déformation

Une instruction de mise à l'échelle ou de déformation aura une forme de ce type :

```
Tail 1.0,1.0,1.5
```

Les trois paramètres qui suivent le mot-clé Tail (en 2D, il n'y en aurait bien sûr que deux) provoquent une multiplication des coordonnées X, Y et Z, respectivement. Cette ligne d'exemple laisse donc les coordonnées X et Y inchangées, mais augmente de moitié (multiplication par 1.5) la hauteur de l'objet.

Pour un agrandissement ou un rapetissement homogènes de l'objet, les trois paramètres doivent être égaux (attention : ils doivent bien être tous énoncés explicitement, quitte à répéter trois fois la même valeur). Ils peuvent différer comme dans la ligne donnée en exemple, lorsque l'on souhaite allonger, aplatir ou étroitiser l'objet. Un ou plusieurs des paramètres peuvent aussi être négatifs afin d'engendrer des symétries : `Tail -1.0,1.0,1.0` entraîne par exemple un effet de miroir gauche-droite.

Remarque
Bien que très pratique, cette façon d'engendrer des symétries en multipliant les coordonnées par -1.0 ou une autre valeur négative n'est quand même pas totalement orthodoxe, et pourrait poser des problèmes de portabilité vers d'autres logiciels que Blender et POV-Ray. En effet, comme nous l'avons déjà expliqué lorsque nous parlions du mode de définition des polygones, certains logiciels considèrent qu'aucune face d'un objet n'a besoin d'être prise en compte lorsque l'énumération des points de son périmètre dessine un tracé tournant dans le sens des aiguilles d'une montre. Or, bien évidemment, ce sens peut être inversé si l'on demande une symétrie.

Importance du positionnement par rapport au point O

Il faut bien comprendre que pour les formats SGML 3D et 2D, les opérations de modifications de coordonnées (qu'il s'agisse de déplacement, de rotation ou de mise à l'échelle) ne sont rien d'autre que des calculs simples sur les coordonnées des points 3D. Une modification de position consiste à ajouter des valeurs aux coordonnées existantes, non à les remplacer, et cela entraîne donc un déplacement relatif de l'objet et non un positionnement absolu. Une mise à l'échelle consiste à multiplier ces coordonnées (ce qui ne fait pas seulement grossir ou diminuer l'objet, mais aussi le rapproche ou l'éloigne des axes selon son positionnement initial). Une rotation consiste à faire tourner l'objet non pas sur lui-même, mais

autour des axes, et donc ce n'est pas seulement l'orientation propre de l'objet qui est modifiée, mais aussi sa position par rapport aux axes.

Lorsque l'on définit un objet, il ne faut donc pas seulement se préoccuper de sa forme, mais aussi de son positionnement par rapport au point O (l'intersection des axes). Ce n'est jamais une bonne idée que de l'en éloigner sans raison. Généralement (mais il y a de nombreuses exceptions), le positionnement le plus intelligible consiste à faire en sorte que le point O soit placé à la base de l'objet et à la moyenne de ses coordonnées X et Y (la plupart des logiciels de 3D ont l'habitude de placer le point O à mi-hauteur de l'objet et non à sa base, ce qui est généralement une mauvaise idée).

Si en outre l'objet est par nature structuré avec un avant, un arrière, une partie gauche et une partie droite (s'il s'agit d'un véhicule, ou d'un personnage, voire d'une chaise), il est souhaitable que l'avant de l'objet soit orienté vers les valeurs les plus élevées de l'axe des Y, que sa partie droite ait des coordonnées X positives et sa partie gauche des coordonnées X négatives. Dans ce cas, comme nous l'avons expliqué plus haut, ce que l'on a coutume d'appeler une vue « de face » sera en fait une vue de dos... ce qui est plus pratique, les mots « droite » et « gauche » ayant alors la même signification que l'on se place du point de vue de l'observateur ou de l'objet lui-même.

Annulation des modifications de coordonnées

Une scène en 3D peut comporter des dizaines d'objets (parfois réunis en ensembles logiques) et chacun d'entre eux devra bénéficier d'un positionnement précis. Il est donc important de permettre aux modifications de coordonnées de limiter leurs effets à une partie seulement d'un fichier SGML abondant.

En SGML « canonique », et surtout en XML, on atteindrait cet objectif grâce à l'imbrication des balises ouvrantes et fermantes. L'auteur, lui, s'est autorisé sur ce point une légère hérésie par rapport aux règles SGML. Elle n'entraîne pas de difficulté de compréhension, bien au contraire... et de surcroît elle simplifie très considérablement les traitements informatiques. On procèdera donc de la façon suivante :

```
<modifcoord>
```

(ici la liste des modifications de coordonnées à effectuer)

```
</modifcoord>
```

(ici toutes les instructions concernant les objets dont les coordonnées seront modifiées : définition de couleurs, de points, de polygones, etc.)

```
<modifcoord>
Fin
</modifcoord>
```

Et voilà, pas plus compliqué que cela : cette ligne « Fin » entre deux balises permet de repartir à zéro dans les modifications de coordonnées.

Positionnement des objets enregistrés indépendamment

L'intérêt de cette légère inélégance va vite être flagrant lorsque l'on va positionner dans une scène des objets définis dans des fichiers SGML 3D extérieurs. L'extrême concision du code SGML décrivant la scène va alors considérablement faciliter les manipulations minutieuses. Par exemple, pour placer une chaise rouge pivotée de 30 ˚ vers la gauche en x = -3.0 et y = 2.0, ainsi qu'une chaise verte pivotée de 20 ˚ vers la droite en x = 5.0 et y = -0.5 (ces deux objets étant définis dans des fichiers indépendants), on écrira simplement :

```
<modifcoord>
Rotz 30.0
Posi -3.0,2.0,0.0
</modifcoord>
<inclusion>
chaiserouge.3d
</inclusion>
<modifcoord>
Fin
</modifcoord>

<modifcoord>
Rotz -20.0
Posi 5.0,-0.5,0.0
</modifcoord>
<inclusion>
chaiseverte.3d
</inclusion>
<modifcoord>
Fin
</modifcoord>
```

ce qui n'est même pas bavard, mais surtout autorise une précision absolue (qu'on aurait toutes les peines du monde à obtenir à la souris) tout en restant parfaitement intelligible. Dès lors – et c'est là le gros avantage –, tous les éléments de cette scène peuvent être modifiés (valeur d'une rotation, variation de placement... on peut même remplacer toute une hiérarchie d'objets par une autre) en ouvrant ce fichier avec un éditeur de texte et en modifiant l'une de ses... 20 lignes, pas une de plus (rappelons qu'une scène 3D comporte couramment des dizaines de milliers de points et de polygones !).

Une scène peut ainsi inclure plusieurs objets, qui peuvent eux-mêmes appeler plusieurs sous-objets, qui eux-mêmes... Ce n'est pas tout : notez bien que les fichiers appelés peuvent eux-mêmes contenir des modifications de coordonnées, et appeler d'autres sous-fichiers qui en contiennent d'autres, etc. Les représentations 3D les plus minutieuses, fussent-elles composées de dizaines d'objets, peuvent ainsi être découpées en fichiers SGML indépendants, d'autant plus courts et lisibles qu'ils jouent un rôle plus important dans la hiérarchie de la scène. Toute cette complexité devient limpide parce qu'elle est toujours structurée logiquement. Il s'en faut de beaucoup que l'interface d'un logiciel 3D autorise la même rigueur.

Vous pensez qu'il est plus facile d'effectuer ces manipulations à la souris avec un logiciel comme Blender ? Eh bien, vous avez complètement tort : quand une scène est composée de dizaines de petits éléments de toutes tailles, l'enchevêtrement des traits à l'écran rend les manipulations à la souris excessivement malcommodes. Comme si cela ne suffisait pas, sur une scène complexe, les temps de réponse à la moindre manipulation sont très vite insupportables. Les logiciels de 3D essaient de contourner la difficulté en employant des diagrammes structurés (très vite si volumineux qu'ils créent plus de difficultés qu'ils n'en résolvent), ou en découpant une même scène en calques (ce qui n'est probablement ni moins lourd ni moins abstrait... mais est en revanche totalement intraduisible d'un logiciel à l'autre). Rien n'y fait : très vite, tous les amateurs de 3D qui s'attaquent à des scènes complexes se mettent à pleurer sur la petite taille de leur mémoire vive et le peu de puissance de leur ordinateur. Au lieu de se lamenter, ils feraient bien mieux d'apprendre à travailler avec une hiérarchie de fichiers SGML... car cette technique aussi rapide que précise ne demande rigoureusement aucune puissance.

Et quand bien même on supposerait (ce qui, à notre sens, est faux) que le positionnement des objets à la souris est acceptable pour produire une image fixe, sans l'ombre d'un doute on ne peut pas envisager une telle méthode lorsque l'on souhaite s'attaquer à l'animation. Cette dernière, en effet, exige vingt-cinq modifications de coordonnées, précises, régulières, pour chaque seconde de vidéo produite ! Ce n'est vraiment pas envisageable manuellement. Il existe certes (notamment dans Blender) des interfaces graphiques pour tenter de définir à la souris de tels processus... Nous n'hésitons pas à dire qu'elles sont très inutilement compliquées (inutilisables serait d'ailleurs plus exact), alors que ces actions peuvent être réalisées avec une simplicité déconcertante par la modification de quelques lignes bien choisies dans une hiérarchie de fichiers SGML.

4.2.5 Récapitulation des balises employées

Pour décrire la 3D maillée, le format SGML défini dans cet ouvrage emploie en tout et pour tout cinq balises : *points3D, faces, couleur, inclusion, modifcoord*

(avec les mots-clés *Posi, Tail, Rotx, Roty, Rotz, Fin*).

Pour décrire la 2D maillée, le format SGML défini dans cet ouvrage emploie en tout et pour tout six balises : *points2D, faces, traits, couleur, inclusion, modifcoord* (avec les mots-clés *Posi, Tail, Rota, Fin*).

Toute la puissance et la simplicité d'emploi auxquelles donnent accès les convertisseurs du CD-ROM joint sont donc à votre portée si vous faites l'effort d'apprendre... une quinzaine de mots. Vous auriez vraiment tort de vous en dispenser.

4.3 Principaux usages des convertisseurs

Les convertisseurs du CD-ROM joint ont été conçus pour couvrir la plupart des besoins courants de l'amateur de 3D. On peut définir cinq hypothèses d'utilisation principales.

4.3.1 Synthèse

En premier lieu, certains convertisseurs peuvent transformer une hiérarchie de fichiers 3D ou 2D, même extrêmement détaillée et répartie en un nombre indéfini de niveaux hiérarchiques (jusqu'à une centaine, ce qui va très au-delà du vraisemblable) en un fichier unique, plus facile à manipuler et en particulier à dupliquer. En contrepartie, bien entendu, ce fichier de synthèse ne doit normalement plus être modifié, car il est la plupart du temps trop abondant pour qu'on puisse s'orienter en son sein.

La synthèse pourrait consister en une simple recopie de toutes les instructions (laissant en particulier subsister des imbrications de commandes de modifications de coordonnées). Mais dans la plupart des cas, on procède à la fabrication d'un fichier unique dans l'idée de faciliter un traitement, en particulier une traduction vers un autre format. Aussi, la synthèse est généralement accompagnée de tous les calculs géométriques (déplacements, rotations, mises à l'échelle) permettant l'expression directe des coordonnées de tous les points servant à décrire l'objet ; les lignes qui exprimaient des modifications de coordonnées au sein de la structure hiérarchique peuvent ainsi être éliminées sans que l'objet change d'aspect.

Le fichier unique résultant d'une telle synthèse comporte donc exclusivement des balises (ouvrantes et fermantes, bien entendu) `<points3D>` ou `<points2D>`, `<faces>` (ainsi qu'éventuellement `<traits>` dans le cas d'un fichier 2D) et `<couleur>`, mais en aucun cas les balises `<inclusion>` ni `<modifcoord>`. En d'autres termes, il n'est pas possible de décrire la 3D ou la 2D maillée de façon plus simple. De tels fichiers simplifiés sont idéaux pour toutes les formes d'automatisation, et en particulier pour les traductions.

4.3.2 Traduction, stockage en vue de traduction ultérieure

La deuxième fonction des convertisseurs est la plus évidente : certains d'entre eux peuvent traduire un fichier SGML 3D ou 2D à un format reconnu par un programme puissant (en particulier Blender ou POV-Ray), ou inversement, traduire en SGML des données exportées par l'un de ces programmes. Le format SGML peut alors servir de simple intermédiaire entre programmes incompatibles.

C'est en particulier ce que nous faisons pour intégrer des formes définies avec Blender dans une image de synthèse produite par POV-Ray. Mais il faut insister sur le fait que le stockage de vos données en SGML est d'une prudence élémentaire si vous n'exluez pas a priori (et comment le pourriez-vous ?) la possibilité de changer de modeleur (pour remplacer Blender) ou de moteur de rendu (pour remplacer POV-Ray) au cours des prochaines années : à ce moment, vous n'aurez guère de difficulté à automatiser la traduction du format SGML simplissime, alors que vous risquez fort de laisser vos données devenir obsolètes si vous vous contentez de les sauvegarder à un format complexe qui sera un jour revu en profondeur, voire deviendra obsolète, comme c'est inéluctable. Ne mettez pas tous vos œufs dans le même panier (en l'occurrence, Blender) !

4.3.3 Passages de la 2D à la 3D

Le passage de la 2D à la 3D est utile dans trois sortes de cas. Il peut s'agir de la simple traduction d'un motif décoratif défini en 2D, et que l'on veut plaquer à plat dans une scène 3D (par exemple pour décorer un mur). En particulier, c'est ainsi que l'on peut plaquer une photographie ou une image informatique sur une forme 3D même si elle n'est ni plate ni rectangulaire.

On peut aussi vouloir conférer une épaisseur à un périmètre défini en 2D, afin de définir rapidement des formes plates d'épaisseurs diverses, notamment des parois. C'est ce que l'on appelle une extrusion.

Enfin, la technique peut être employée pour transformer un profil défini en 2D en un objet rond (ayant l'aspect d'une poterie ou de n'importe quelle forme modelée au tour : pied de meuble, colonne de réverbère, etc.).

4.3.4 Passages de la 3D à la 2D

Certains convertisseurs peuvent aider à représenter des données tridimensionnelles sur l'espace plat d'un ordinateur ou d'une feuille de papier (certaines illustrations de cet ouvrage ont d'ailleurs été fabriquées avec cette technique). C'est possible pour toutes les représentations traditionnelles : projections (vues de dessus, de face et de profil), perspectives classiques ou axonométriques (c'est-à-dire

dépourvues de lignes de fuite : les objets les plus lointains n'apparaissent alors pas plus petits que les plus proches).

Certains convertisseurs joints à cet ouvrage sont en effet capables de représenter des données 3D en mode « fil de fer » d'une façon très élémentaire, mais souvent beaucoup plus compréhensible qu'une vue d'artiste. Cela permet en particulier de procéder à des agrandissements astronomiques sur les plus petits détails d'une scène gigantesque.

Cependant, nous aurons très peu recours à ces outils, car Blender s'acquitte de ces missions d'une façon beaucoup plus souple et rapide. Il est quand même bon de connaître cette possibilité, d'autant qu'elle est peu gourmande en puissance et ne pose pas de problèmes de compatibilité. Elle peut donc dans une certaine mesure permettre de se passer de Blender sur une machine trop ancienne pour faire tourner ce programme... ou au contraire trop récente pour que Blender lui ait déjà été adapté (l'apparition de nouvelles normes de cartes graphiques peut en effet poser des problèmes d'adaptation aux développeurs de Blender – et n'oubliez pas qu'il s'agit de bénévoles travaillant forcément à leur rythme).

4.3.5 Exploitation de dessins faits à main levée

Last but not least, les convertisseurs peuvent traduire au format SGML 2D (pour une exploitation ultérieure en 3D) des données repérées sur n'importe quelle image : dessin à main levée, scan d'un petit croquis, copie d'écran... et même photographie. C'est souvent cette technique qui est la plus utile à l'amateur pour transformer rapidement en scènes 3D des formes qu'il sait dessiner avec un outil informatique quelconque (logiciel de dessin vectoriel ou de retouche), ou tout simplement avec un crayon (le croquis le plus élémentaire peut être exploité dès lors qu'on l'a fait passer par un scanner).

4.4 Les autres formats gérés

Le format de type SGML que nous avons défini dans ce chapitre étant avant tout conçu pour permettre des échanges, il faut faire mention des autres types de fichiers informatiques qui peuvent être convertis à sa syntaxe ou créés à partir de lui. Les convertisseurs livrés sur notre CD-ROM se limitent aux logiciels autour desquels cet ouvrage est conçu, et se bornent donc à permettre l'import-export de données au format Videoscape OBJ (que Blender utilise pour les échanges), et l'exportation vers POV-Ray. Mais par le passé ou en marge de cet ouvrage, il est arrivé à l'auteur d'utiliser le même format SGML pour récupérer (importer) des données 3D enregistrées en AutoCAD DXF, en Caligari TrueSpace et en VRML 1.0 et 2.0. L'exportation vers ces mêmes formats est également possible, ainsi que

la conversion en fichiers 2D d'Adobe Illustrator (nous avons d'ailleurs utilisé cette faculté pour certaines illustrations de ce livre). Tout cela n'a rien pour surprendre un informaticien : la structure des fichiers SGML est idéale pour réaliser tous types de conversion, y compris – et c'est de loin le plus important – pour permettre une compatibilité avec les formats qui ne pourront pas manquer d'apparaître dans l'avenir.

Deuxième partie

La 3D modelée (Blender)

Chapitre 5

L’interface de Blender

5.1 Les différents types de fenêtres et leurs limites

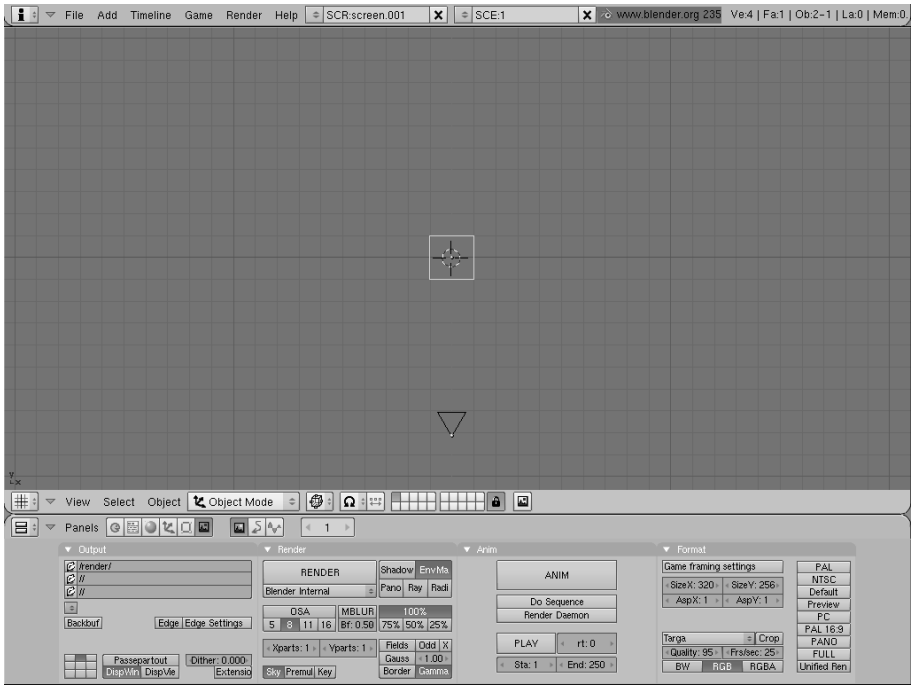


FIG. 5.1 – *L’écran d’accueil de Blender.*

L’interface de Blender obéit à une logique particulière et très déconcertante au départ. Pour faire court, disons que la plupart des opérations y sont effectuées avec les deux mains : la main droite manipule la souris (d’une façon d’ailleurs assez particulière : la position du pointeur est importante même lorsque l’on ne clique

pas, en revanche les doubles et triples clics sont généralement inopérants), et la main gauche tape de temps à autre une touche de clavier pour modifier, préciser ou compléter un mode de fonctionnement. Faute de savoir cela, la plupart de ceux qui ont cherché à comprendre Blender intuitivement, en cliquant dans tous les coins, se sont vite découragés avec le sentiment d'avoir affaire à une épouvantable usine à gaz.

Cette impression est totalement fausse. Une fois qu'on a bien assimilé les principes généraux de l'interface de Blender – ce qui est l'affaire de quelques dizaines de minutes –, on se rend compte que pour toutes les opérations courantes elle est très facile à utiliser en dépit de ses bizarreries. Plus tard, avec la pratique acquise au fil des semaines, on comprendra que ce sont justement ces spécificités qui permettent d'aboutir très rapidement à des résultats élaborés.

Cela étant, il est indéniable que cette interface ne se conforme absolument pas aux habitudes que la plupart des utilisateurs ont acquises en manipulant des logiciels de bureautique. L'écran de Blender comporte bien des menus déroulants et de petites icônes d'apparence classique, mais ne vous y fiez pas : tout cela ne joue qu'un rôle assez marginal. Laissez donc absolument de côté tout ce que vous êtes habitué à faire avec des ascenseurs, des cases à cocher, des doubles clics et ainsi de suite : le mode de fonctionnement de Blender n'a rien à voir avec tout cela.

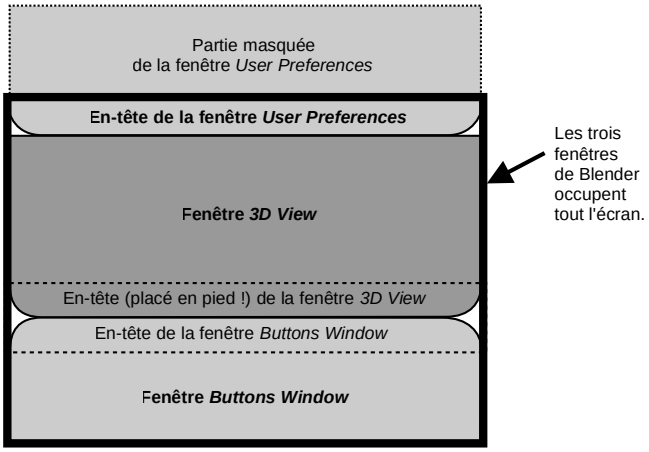


FIG. 5.2 – *Disposition schématique des trois fenêtres de l'écran d'accueil. Notez l'orientation des arrondis à gauche et à droite des en-têtes.*

La première chose à comprendre dans l'interface de Blender, c'est que ce que l'on a sous les yeux ne doit pas être considéré comme une fenêtre unique, mais bien comme un écran composé de plusieurs fenêtres – ou, plus exactement, entièrement rempli par des fenêtres juxtaposées. Ces dernières sont toujours les unes à

côté des autres, un peu comme des tableaux sur un mur, et absolument jamais les unes par-dessus les autres comme des feuilles qu'on fait glisser sur un bureau. Oubliez donc vos réflexes acquis avec les fenêtres de Windows ou de Mac OS, qu'on ne cesse de déplacer du premier plan à l'arrière-plan : les fenêtres de Blender, elles, ne se superposent jamais.

Partons de l'écran tel qu'il se présente au lancement du logiciel (fig. 5.1). Même si cela n'est pas évident pour le néophyte, cet écran comporte bel et bien trois fenêtres disposées comme l'indique la fig. 5.2. Nous allons apprendre comment les identifier.

FIG. 5.3 – *Les lignes d'en-tête des trois fenêtres de l'écran d'accueil. De haut en bas : User Preferences, 3D View, Buttons Window. Notez l'orientation des arrondis à gauche et à droite.*

Chacune de ces fenêtres comporte un en-tête (header), c'est-à-dire une barre horizontale qui comprend des mentions textuelles (correspondant à des menus déroulants que nous étudierons plus tard) et de petites icônes. La fig. 5.3 regroupe les trois en-têtes présents sur l'écran d'accueil. Sur votre écran, ces en-têtes ne sont bien sûr pas collés ainsi en un groupe compact : ils peuvent être éloignés les uns des autres et séparés par de vastes parties de l'écran ; nous ne les avons regroupés que pour la lisibilité.

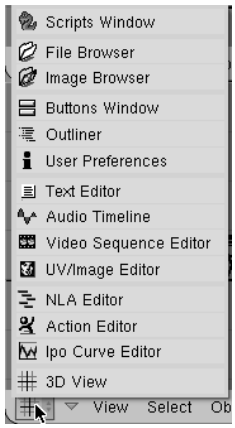


FIG. 5.4 – *Le menu des types de fenêtres.*

Repérez bien l'emplacement de ces lignes d'en-tête sur l'écran, et regardez attentivement à gauche et à droite de chacune d'entre elles : vous y noterez la présence de légers arrondis (que nous avons délibérément exagérés sur la représentation schématique de la fig. 5.2). Ces derniers sont placés tantôt en bas de l'en-tête comme dans les deux premiers cas, tantôt en haut comme dans le troisième. Cette indication à peine visible est pourtant essentielle, car elle vous permet de comprendre si l'en-tête en question concerne la zone d'écran située au-dessous ou au-dessus de lui (et dans ce dernier cas, le terme « header », qui signifie en-tête, est un peu impropre, mais c'est ainsi que les concepteurs de Blender ont choisi de l'appeler !). En l'occurrence, seul le troisième en-tête, celui du bas, est bien placé en haut de la fenêtre à laquelle il correspond. Au contraire, le premier en-tête, celui qui est disposé tout en haut de l'écran, correspond à une fenêtre fictive placée au-dessus de lui, hors de l'écran, si bien qu'on n'en voit pas le contenu.

Tout cela est assez déconcertant, aussi allons-nous nous livrer à un petit exercice pour le rendre plus compréhensible. Notez que chacun de ces en-têtes comporte à sa gauche une petite icône (qui symbolise le type de fenêtre auquel on a affaire). Lorsque l'on clique sur cette icône (avec le bouton gauche de la souris), on fait apparaître le menu du type des fenêtres (fig. 5.4).

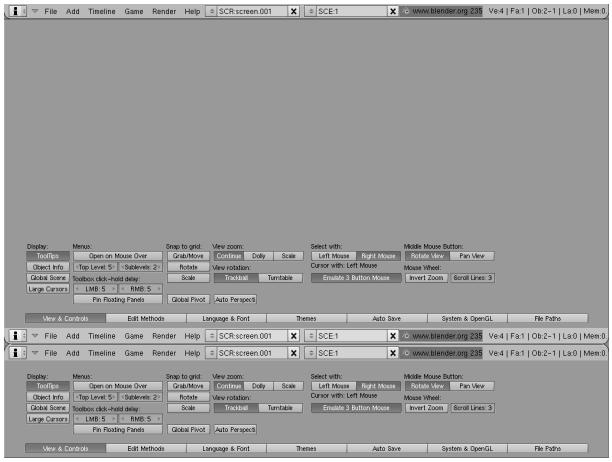


FIG. 5.5 – *L'écran dans sa configuration de départ, mais modifié pour afficher trois fenêtres « User Preferences ». Seule la ligne d'en-tête de la fenêtre du haut est visible.*

Allez-y : en cliquant sur ces trois icônes l'une après l'autre, choisissez par trois fois « User Preferences » pour chacun des trois en-têtes. Vous devriez obtenir un écran semblable à celui de la fig. 5.5.

Notre écran comporte désormais trois vues similaires sur une fenêtre « User Preferences ». Les deux fenêtres du bas seraient totalement identiques si leurs en-

têtes n'étaient disposés différemment : l'un en bas, l'autre en haut de sa fenêtre. Mais pour la toute première fenêtre, celle du haut de l'écran, seule la ligne d'en-tête est visible (comme c'était déjà le cas sur la fig. 5.2). Il faudrait descendre légèrement cette fenêtre pour en voir tout le contenu.

Comme les fenêtres de Blender occupent toujours tout l'espace disponible sur l'écran, il est impossible de ne redimensionner qu'une seule d'entre elles : on ne peut jamais augmenter l'espace que l'on accorde à l'une sans diminuer simultanément celui dévolu à une autre. Nous allons donc à la fois agrandir l'espace occupé par la fenêtre du haut et diminuer celui de la fenêtre centrale.

Pour cela, nous devons placer le pointeur de la souris à la limite entre les deux fenêtres, c'est-à-dire juste sur le bord inférieur de la barre d'en-tête du haut de l'écran. Le pointeur prend alors la forme d'une double flèche <=> [1], ce qui signale que l'on peut déplacer la limite entre les deux fenêtres en maintenant la pression sur le bouton gauche de la souris. Abaissez donc cette limite pour aboutir à un écran du type de celui de la fig. 5.6.

Quand vous serez familier de ces manipulations, elles vous permettront de configurer l'interface de Blender à votre convenance. Mais dans l'immédiat, il est vraisemblable que vous souhaiteriez plutôt remettre Blender dans l'état où vous l'aviez trouvé... Rien de plus facile : déroulez simplement le menu « File » que vous trouverez dans l'un quelconque des trois en-têtes, et cliquez sur l'option « New ». Il apparaîtra aussitôt, juste sous le pointeur, le bloc de message de la fig. 5.7, et il vous suffira de cliquer sur la mention « Erase All » pour retrouver l'interface de Blender dans son état initial, celui de la fig. 5.1.

Utiliser Blender sans souris à trois boutons

Toute l'interface de Blender a été pensée pour être manipulée avec une souris à trois boutons, mais vous pouvez très bien vous contenter d'une souris à molette (la molette, que l'on peut cliquer, étant assimilée au bouton central). Avec une souris à deux boutons seulement, il vous suffira d'appuyer sur les deux boutons à la fois pour simuler le bouton central. Même la souris sans bouton du Macintosh est utilisable, en association avec les touches Ctrl, Option et Pomme (dans la partie inférieure gauche du clavier) : Ctrl-clic simule un clic sur le bouton gauche, Option-clic sur le bouton central, Pomme-clic sur le bouton droit.

Signalons à ce sujet que la documentation officielle de Blender emploie les abréviations LMB, MMB et RMB *(left, middle* et *right button)* pour désigner respectivement les boutons gauche, central et droit de la souris.

Blender emploie très peu la technique du double et du triple clics : a priori, toutes les opérations réalisées à la souris peuvent se contenter d'un clic unique sur le bouton approprié.

[1] Ce changement de pointeur n'a malheureusement pas lieu avec la version pour Mac de Blender.

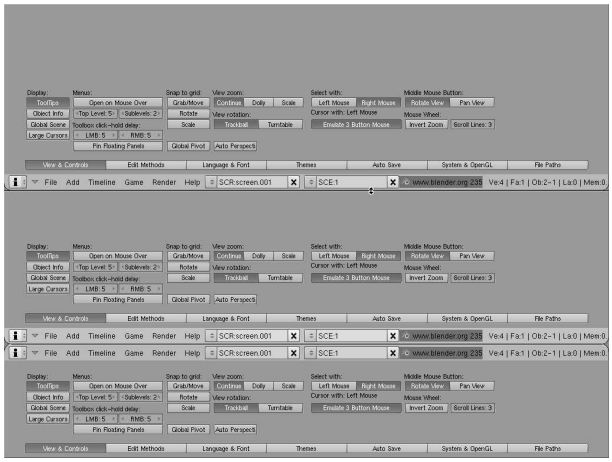


FIG. 5.6 – *Les trois fenêtres « User Preferences », toutes visibles après l'abaissement de la limite entre les deux fenêtres supérieures. Remarquez le pointeur de la souris en forme de double flèche à la limite entre les deux premières fenêtres.*

FIG. 5.7 – *Le message « Erase All » apparaît sous le pointeur au moment où l'on abandonne un document non enregistré. Chaque fois qu'un message de ce type apparaît, on valide une option en cliquant sur son nom, ou on abandonne l'opération en tapant la touche Echap.*

5.2 La fenêtre active

Nous venons de voir que l'écran de Blender comporte plusieurs fenêtres. Il faut savoir qu'à un moment précis, une seule de ces fenêtres est réputée active, c'est-à-dire vouée à réagir aux sollicitations de l'utilisateur. Il est donc important de savoir que la fenêtre active est toujours celle sur laquelle se trouve le pointeur de la souris au début de la manipulation.

Cela paraît trivial. Pourtant, il est indispensable d'insister lourdement sur ce point. Car ce mode de fonctionnement est en fait très différent de celui en vigueur avec les interfaces graphiques de type Windows ou Macintosh (pour lesquelles la fenêtre active est généralement la dernière sur laquelle on a cliqué). Avec Blender, le fait de cliquer ou non ne change rien à l'affaire : c'est la position du pointeur et elle seule qui désigne la fenêtre active. Cela a une conséquence souvent inat-

tendue : on peut changer de fenêtre active sans même s'en rendre compte si l'on déplace la souris par mégarde – en lui donnant un coup de coude pour attraper le téléphone qui sonne, par exemple.

Pour mieux comprendre cela, faites la navette avec le pointeur de la souris entre les en-têtes des deux fenêtres du bas, qui sont placés l'un juste au-dessus de l'autre (Fig. 5.8). En même temps que vous opérez ce déplacement, faites très attention à la couleur d'arrière-plan de ces lignes d'en-tête. Vous verrez qu'à chaque fois que le pointeur franchit la limite entre les deux fenêtres, l'en-tête de la fenêtre anciennement active (celle que le pointeur vient de quitter) s'assombrit, en même temps que celui de la fenêtre nouvellement active (celle où le pointeur vient d'entrer) s'éclaircit. La différence est subtile et même presque indécelable, mais elle n'en témoigne pas moins du changement de fenêtre active... et cela a une importance essentielle, car Blender ne réagira pas du tout de la même façon selon que le pointeur se trouvera en-deçà ou au-delà de la limite entre les deux fenêtres.

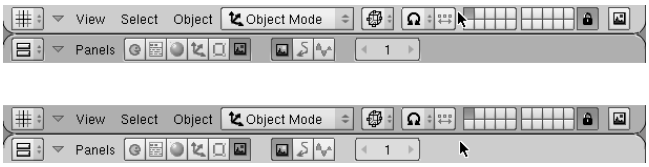


FIG. 5.8 – *La position du pointeur de la souris détermine la fenêtre active. Cette dernière est aussi signalée, quoique de façon très discrète, par un en-tête légèrement plus clair.*

En voici un exemple. Placez votre pointeur au milieu de la grande fenêtre grise et quadrillée, et tapez alternativement les touches + et - du pavé numérique de votre clavier. Vous verrez le quadrillage s'élargir à chaque pression sur la touche +, se resserrer à chaque pression sur la touche -. Passez à présent le pointeur de la souris au bas de l'écran, dans la zone des boutons, et procédez à ce qui semble pourtant être rigoureusement la même manipulation : tapez à nouveau en alternance sur les touches + et -. Cette fois, le quadrillage ne sera plus modifié le moins du monde... ce sont les boutons de la fenêtre du bas qui grossiront ou rétréciront, jusqu'à devenir illisibles ou être éjectés hors des limites de l'écran (voir fig. 5.9) ! Vous imaginez donc sans peine à quel point des erreurs de positionnement du pointeur de la souris peuvent être déconcertantes pour un débutant. Faites y très attention et apprenez à ne pas laisser traîner ce pointeur n'importe où.

Au cours de votre apprentissage, et à cause des réflexes que vous avez acquis avec d'autres logiciels, vous aurez très certainement à bien des reprises l'impression que Blender réagit de façon absurde à vos sollicitations – ou qu'il ne réagit pas du tout. Vous penserez avoir affaire à un bug, ou à un plantage. Eh non ! Dans

l'immense majorité des cas, cette impression est simplement due au fait que vous avez changé l'emplacement du pointeur de la souris sans vous en rendre compte.

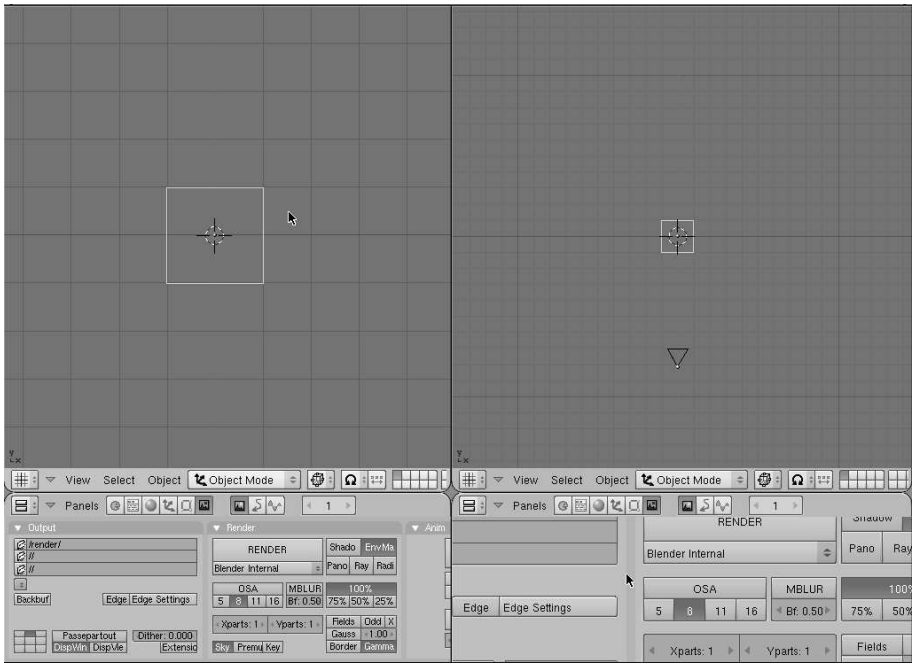


FIG. 5.9 – *Ce qui paraît être la même manipulation peut avoir des conséquences très différentes selon la position du pointeur de la souris. A gauche, le pointeur de la souris se trouvant dans la zone centrale « 3D View », quelques pressions sur la touche + du pavé numérique ont changé l'échelle du quadrillage. A droite, le pointeur étant dans la fenêtre du bas « Buttons Window », la même opération a démesurément grossi les boutons. Pour l'utilisateur novice, cela a toutes les apparences d'un bug. Il s'agit pourtant bel et bien d'une erreur de manipulation.*

5.3 Division, suppression et agrandissement des fenêtres

N'importe laquelle des fenêtres de Blender peut être divisée en deux, horizontalement ou verticalement. Dans la pratique, cette faculté n'est guère employée que pour démultiplier les vues en 3D, lorsque l'on souhaite examiner le même objet simultanément sous plusieurs angles (de dessus, de face, de profil, etc.). C'est ce que nous allons faire à présent. Commençons donc par placer le pointeur de la souris dans la zone grisée et quadrillée pour en faire la fenêtre active. Faisons-le

ensuite glisser vers la droite, jusqu'à toucher le bord de l'écran. Comme toujours lorsque le pointeur atteint la limite de la fenêtre, il se transforme en double flèche <=> (sauf sur un Mac, malheureusement, mais la manipulation n'en reste pas moins réalisable). Cliquez alors sur le bouton droit (ou faites Pomme-clic avec une souris de Macintosh), et choisissez « Split Areas » dans le petit menu qui apparaît alors sous le pointeur (Fig. 5.10). Cela fait apparaître une ligne horizontale grise qui suit verticalement les déplacements du pointeur de la souris. Il n'y a plus qu'à effectuer un clic gauche (ou un simple clic avec une souris de Mac) pour terminer l'opération de division de la fenêtre en deux.

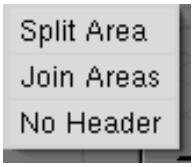


FIG. 5.10 – *Le menu de division des fenêtres qui apparaît sous le pointeur lorsqu'on effectue un clic droit sur l'un quelconque des côtés de la fenêtre.*

Maintenant que nous savons le faire, apprenons à le défaire... Déplacez le pointeur de la souris sur la limite de fenêtre que nous venons de faire apparaître (la transformation du pointeur en double flèche signale – sauf sur Macintosh – que l'emplacement correct est atteint), et effectuez à nouveau un clic droit. Le menu de la fig. 5.10 apparaît de nouveau, mais il faut maintenant valider l'option « Join Areas » : immédiatement, la fenêtre nouvellement créée disparaît.

Il nous faut signaler une difficulté : l'option « Join Areas » (qui supprime une fenêtre) apparaît toujours dans le menu de la fig. 5.10, mais en fait elle n'est pas systématiquement disponible : elle ne peut fonctionner que s'il y a bien eu auparavant une opération « Split Area » correspondante. On s'attendrait à ce que dans le cas contraire, la mention « Join Areas » apparaisse en grisé ; ce n'est malheureusement pas le cas.

Entraînez-vous. Pour obtenir une division verticale, la démarche est la même, mais il faut placer le pointeur sur la limite supérieure ou inférieure de la fenêtre, et non plus sur le côté. Notez que rien ne vous interdit de changer le type d'une fenêtre nouvellement créée grâce au menu des types de fenêtres (Fig. 5.4). Essayez d'aboutir seul à une configuration comportant quatre fenêtres « 3D View » (comme sur la fig. 5.12), puis de revenir à un écran de Blender plus classique.

C'est une bonne chose que de pouvoir multiplier les petites fenêtres, mais de temps à autre, par exemple lorsque l'on veut travailler avec précision, on peut avoir besoin de donner à l'une d'entre elles une extension maximale. Fort heureusement, il n'est pas nécessaire de reconfigurer pour cela toute la mosaïque des fenêtres. Il

suffit de taper « Ctrl-flèche haut » pour que la fenêtre où se trouve le pointeur de la souris occupe tout l'écran – et de recommencer cette opération pour retrouver l'écran dans son état précédent.

5.4 Lecture des fichiers

Il est temps d'apprendre à manipuler Blender avec autre chose que des fenêtres vides. Nous allons donc ouvrir le document `05_mobilier.blend`, inclus dans le dossier `blender_fichiers` du CD-ROM joint à cet ouvrage.

Au départ, l'opération n'a rien que de très classique : on déroule le menu « File » et on clique sur l'option « Open... ». Blender va alors transformer l'une quelconque des fenêtres déjà ouvertes en lui donnant le type « File Browser » (fig. 5.11). Si, comme il est fréquent, cette fenêtre est de trop petite taille pour qu'on y lise toutes les indications nécessaires, il suffit comme nous venons de le voir d'y placer le pointeur de la souris, puis de taper « Ctrl-flèche haut ». Et si le répertoire contient tellement de fichiers qu'il n'est pas possible d'afficher tous leurs noms simultanément – c'est rare –, vous pouvez faire défiler la liste en faisant glisser (avec le bouton gauche de la souris) la grande barre verticale à gauche de la fenêtre : c'est un ascenseur relativement classique, une des rares concessions de Blender à l'ergonomie des logiciels de bureautique.

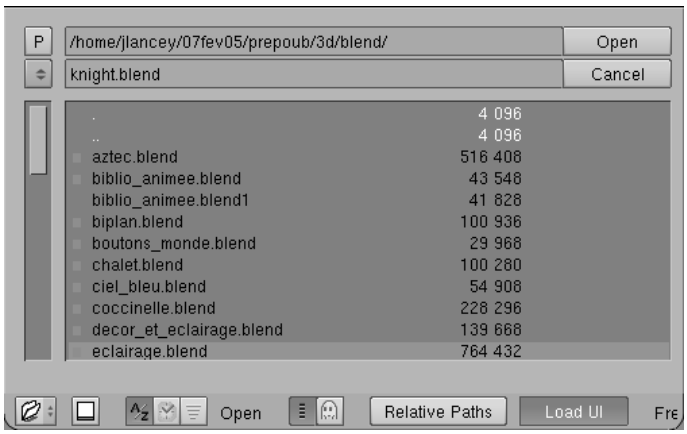


FIG. 5.11 – *Une fenêtre « File Browser » de petite taille. Notez l'icône en bas à gauche : en cliquant sur elle, vous pourrez revenir à l'état initial de la fenêtre si vous souhaitez abandonner l'opération de lecture ou écriture d'un fichier.*

La première ligne de cette fenêtre indique le nom d'un répertoire. Si ce n'est pas celui qui convient, on peut remonter dans l'arborescence en cliquant sur la

lettre P en haut à gauche de la fenêtre, ou au contraire entrer dans les sous-répertoires dont on lit le nom dans la vaste zone centrale, en plaçant le pointeur sur leur nom, puis en cliquant avec le bouton du milieu (pression simultanée sur les deux boutons si la souris n'a pas de bouton central, Option-clic sur un Macintosh). Sous Windows, il peut aussi être nécessaire de passer d'une unité de stockage à l'autre en modifiant le début de cette ligne (en général, `c :` désigne le disque dur, et la suite des lettres de l'alphabet, `d :, e :, f :, g :` donne accès aux lecteurs de CD-ROM ou aux autres dispositifs de stockage).

Lorsque le fichier que l'on cherche apparaît dans la liste, c'est aussi avec un clic central qu'on va le charger en mémoire. Il est fort possible que cette opération bouleverse totalement la disposition des fenêtres (Fig. 5.12). Blender, en effet, enregistre la configuration de l'interface en même temps que les objets en 3D qu'il manipule.

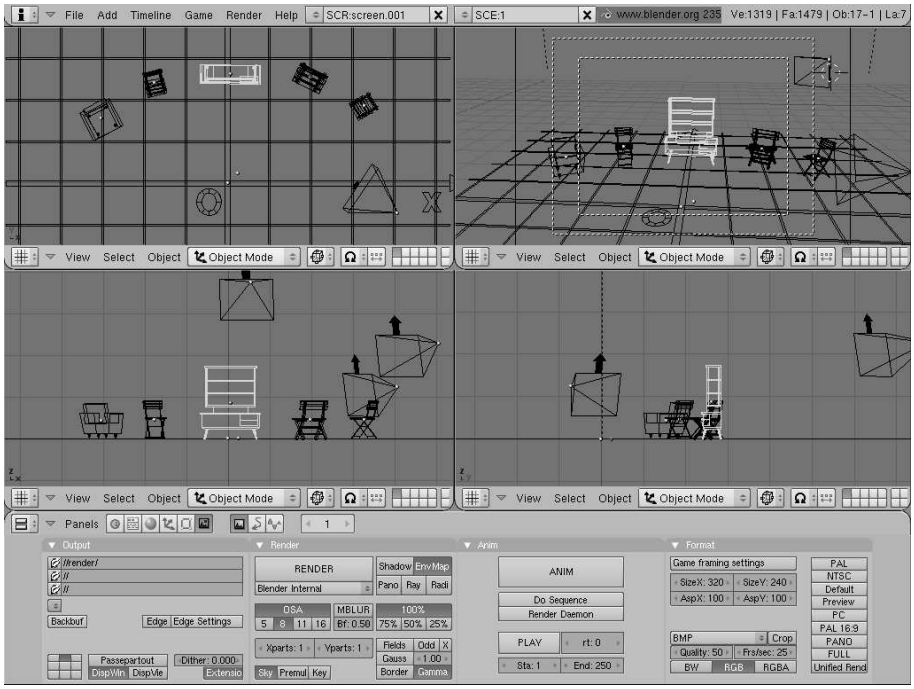


FIG. 5.12 – *Aspect de l'écran après chargement du fichier 05_mobilier.blend. Il s'agit d'une configuration très classique, avec quatre fenêtres « 3D View » permettant de voir la scène simultanément de dessus, de face, de profil et en perspective.*

5.5 Les variations d'échelle

La scène que vous venez de charger dans Blender est censée occuper une dizaine de mètres de large, et les fenêtres 3D ont été configurées pour qu'elle soit intégralement visible dans chacune d'entre elles. Bien sûr, il est fréquent que l'on ait besoin de voir les choses de plus près lorsque l'on souhaite travailler sur des détails. Rien de plus simple. Placez le pointeur sur la fenêtre dont vous voulez modifier l'échelle, et tapez alternativement les touches + et - du pavé numérique. Vous vous rendrez compte que l'échelle des représentations s'en trouve immédiatement modifiée, et si vous poussez assez loin le grossissement, vous verrez aussi le quadrillage changer d'unité : dans la vue de départ, chaque carreau a un mètre de côté. Si vous appuyez assez de fois sur la touche +, l'unité deviendra le décimètre, puis le centimètre – ou, avec la touche -, le décamètre, l'hectomètre et même le kilomètre.

5.6 Le cadrage des vues en projection

Bien sûr, à force de grossir la représentation, vous risquez de vous placer sur une zone où aucun objet n'est visible. Si vous êtes totalement perdu, vous avez toujours le recours d'ouvrir à nouveau le fichier `05_mobilier.blend`. Mais il se peut aussi que vous souhaitiez seulement modifier un peu le cadrage de la zone observée, en allant par exemple un peu plus haut ou un peu plus à gauche. Ne cherchez pas les ascenseurs au bord de la fenêtre : Blender ne les emploie pas, et vous allez voir qu'il s'en passe très facilement, quoique au prix d'un peu de gymnastique...

Placez le pointeur de la souris sur la vue de dessus (c'est un exemple, mais il est préférable de ne pas choisir la vue en perspective, le positionnement des caméras obéissant à une autre logique). Avec un doigt de la main gauche, enfoncez l'une des touches Majuscule de votre clavier (avec un Macintosh, c'est un peu plus périlleux : il vous faut deux doigts pour enfoncer simultanément Majuscule et Option) et maintenez la pression ; enfoncez aussi le bouton central de votre souris (ou sa molette, ou ses deux boutons si vous n'avez pas de troisième bouton), et bougez la souris en maintenant la pression : le cadrage sera modifié à chaque mouvement de la souris (oui, tout cela est un peu compliqué, mais il faut quand même regarder l'écran et pas vos mains !). Et quand vous serez satisfait du cadrage, lâchez tout (ouf !). Tout cela demande un peu d'entraînement, mais on s'y habitue très vite.

Note
A partir de maintenant, pour alléger la rédaction, nous cesserons d'expliquer comment simuler le bouton droit ou le bouton central de la souris avec les dispositifs de pointage qui en sont dépourvus, comme par exemple la souris du Macintosh. Reportez-vous à l'encadré « Utiliser Blender sans souris à trois boutons ».

5.7 Choix des vues en projection ou en perspective

Dans le fichier `05_mobilier.blend`, la vue de dessus est en haut à gauche, la vue de face en bas à gauche, la vue de profil en bas à droite (la vue en perspective occupe la fenêtre restante en haut à droite). C'est là une configuration classique en L, mais vous êtes tout à fait libre de la modifier en fonction de vos besoins. Le type de représentation adopté par chacune des fenêtres peut être changé très simplement. Il suffit de placer le pointeur de la souris sur la fenêtre dont on veut modifier la représentation, et de taper l'une des touches 7, 1, 3 (ou 0) du pavé numérique. Ces valeurs ont été choisies pour des raisons mnémotechniques, même si cela ne vous paraît pas encore évident ! La touche 7 est en effet en haut à gauche du pavé numérique, de même que la vue de dessus est normalement en haut à gauche de l'écran ; la touche 1 est en bas à gauche du pavé numérique, de même que la vue de face est normalement en bas à gauche de l'écran, etc. Quant à la touche 0, vous pouvez la retenir en vous disant que la forme du chiffre 0 évoque celle de l'objectif d'une caméra.

Quand, à force de manipulations ou à la suite d'une fausse manœuvre, vous ne savez même plus ce qu'une fenêtre 3D est supposée représenter, vous pouvez toujours y placer le pointeur et taper 7, 1, 3 (ou 0, mais le positionnement des caméras est une opération un peu plus complexe) pour retrouver la vue classique correspondante... au taux d'échelle en vigueur sur cette fenêtre (éventuellement, vous devrez encore le corriger avec les touches + et -). Vous pouvez aussi noter, en bas à gauche de chaque fenêtre représentant une vue en projection, la petite équerre avec le nom des deux coordonnées représentées : X et Y pour une vue de dessus, X et Z pour une vue de face, Y et Z pour une vue de profil.

5.8 Manipulation des objets

5.8.1 Sélection

La scène décrite par le fichier `05_mobilier.blend` comporte différents objets que nous allons apprendre à agencer à notre convenance, essentiellement en les déplaçant et en les faisant pivoter comme nous pourrions le faire avec des objets réels. Mais tout d'abord, il nous faut apprendre comment indiquer à Blender

quels objets nous voulons manipuler. C'est fort simple : on sélectionne un objet en effectuant un clic droit sur l'un des traits qui en décrivent l'aspect. Si l'opération réussit, l'objet sélectionné est représenté avec des traits de couleur magenta (ce que dans le langage courant on appellerait du mauve) ; si l'on s'est trompé, il suffit de recommencer en cliquant sur un autre trait.

Pour sélectionner plusieurs objets, il faut effectuer des clics droits sur les traits appropriés, en gardant la touche Majuscule enfoncée chaque fois qu'on veut ajouter un objet à la sélection. On peut d'ailleurs employer la même méthode pour désélectionner l'un d'entre eux.

La seule difficulté consiste à trouver parmi les quatre représentations une zone où l'enchevêtrement des traits ne soit pas tel qu'il risque de prêter à confusion. C'est en général avec la vue de dessus qu'on rencontre le moins de problèmes, mais le fichier `05_mobilier.blend` comporte un quadrillage horizontal qui est lui-même un objet 3D, et qu'on peut donc sélectionner par mégarde (ce qui n'est nullement grave : il suffit de le désélectionner par Majuscule-clic droit).

5.8.2 Déplacement

Nous allons à présent déplacer l'espèce de vaisselier placé au centre de la scène. Commençons par le sélectionner en effectuant un clic droit sur lui. N'importe quelle fenêtre convient pour effectuer la sélection. En revanche, pour effectuer un déplacement à l'horizontale, il faut placer le pointeur de la souris sur la vue de dessus, et de préférence au-dessus ou à proximité immédiate du vaisselier lui-même. Cela fait, on tape G au clavier, et instantanément le vaisselier va suivre tous les déplacements de la souris (et ce déplacement est visible sur les quatre fenêtres 3D, et pas seulement sur celle où le pointeur de la souris se déplace). Pour valider le nouvel emplacement, il suffit d'effectuer un clic gauche (mais on peut aussi abandonner l'opération en tapant la touche Echap).

G est l'abréviation de « grab », que l'on peut traduire par « saisir ».

Notez que l'opération est tout à fait réalisable sans que le pointeur de la souris soit à proximité de l'objet déplacé : il faut seulement qu'il soit placé dans la bonne fenêtre au début du déplacement. Là encore, si vous avez le sentiment que Blender se comporte de façon absurde, c'est vraisemblablement que vous n'avez pas assez fait attention à l'endroit où votre pointeur était placé...

Il faut aussi signaler que l'opération, très simple par elle-même, est sujette à confusion : on sélectionne l'objet avec un clic droit, et on valide sa position avec un clic gauche. Invariablement, les débutants confondent les deux aspects de l'opération, et terminent un positionnement très minutieux et laborieux par un grand clic droit de soulagement. Raté ! Il fallait un clic gauche, et tout est à recommencer. C'est très énervant, mais inévitable tant qu'on n'a pas pris l'habitude.

5.8.3 Rotation

Le vaisselier étant toujours sélectionné, modifions à présent son orientation. Pour cela, nous allons continuer à placer le pointeur sur la vue de dessus (qui est appropriée pour faire pivoter l'objet autour d'un axe vertical). Cette fois, il est souhaitable que le pointeur ne se trouve pas au centre de l'objet, mais un peu à côté.

Remarquez que l'objet sélectionné comporte à peu près en son centre (en fait, très exactement à la moyenne de ses coordonnées extrêmes) un gros point magenta. Pour effectuer une rotation, vous allez devoir imaginer une ligne entre ce gros point et le pointeur de la souris : il est donc important que ce pointeur ne se trouve ni trop près, ni trop loin de ce point magenta.

Tapez R et faites tourner le pointeur de la souris autour du point magenta (ou plutôt, faites tourner la ligne imaginaire dont nous venons de parler) : l'objet pivote sur lui-même en même temps. En quelque sorte, il est fixé au sol par un axe central passant par le point magenta, et il pivote autour de cet axe en même temps que le pointeur de la souris tourne autour du point magenta.

Lorsque l'orientation vous convient, un clic gauche permet de la valider – mais vous pouvez aussi abandonner l'opération en tapant la touche Echap.

R est bien évidemment l'abréviation de « rotate » (rotation).

Rien ne vous empêche (sauf le souci de la vraisemblance !) d'effectuer la même opération sur les vues de face ou de profil, pour coucher le vaisselier par terre ou le mettre les quatre pieds en l'air.

5.8.4 Mise à l'échelle

Pour grossir ou diminer un objet sélectionné, il faut d'une façon très semblable placer le pointeur de la souris ni trop près, ni trop loin du point magenta qui désigne le centre de l'objet. Après avoir tapé la touche S (pour abréger « scale », qui signifie « échelle »), il suffit de rapprocher le pointeur de la souris du point magenta pour diminuer la taille de l'objet, et d'éloigner ce pointeur pour grossir l'objet. Comme précédemment, on valide l'opération par un clic gauche, ou on l'abandonne en tapant Echap.

Notez que l'objet dont on diminue la taille est par là-même décollé du sol (toutes ses coordonnées se sont rapprochées de son centre qui, lui, n'a pas changé de place), et que celui que l'on grossit semble au contraire s'y enfoncer. Cela n'a aucune gravité, il faut seulement penser à replacer l'objet après l'avoir mis à l'échelle, pour éviter un positionnement absurde.

5.8.5 Duplication

Avant de nous attaquer à la déformation des objets (qui, comme son nom l'indique, risque de les endommager), apprenons à en faire des copies. Rien de bien sorcier : il faut sélectionner l'objet à dupliquer par un clic droit, et taper Majuscule-D. Cela suffit à créer une copie, mais cette dernière occupe alors exactement le même emplacement que son original, avec lequel on risque donc de la confondre. Il faut donc la déplacer quelque peu, raison pour laquelle Blender se place aussitôt en mode déplacement (sans qu'il soit nécessaire de taper G) : bougez donc quelque peu la copie en même temps que le pointeur de la souris, puis validez par un clic gauche une fois que vous avez placé la copie à l'emplacement qui vous convient. Vous pourrez de toute façon le modifier ultérieurement avec les manipulations que nous venons d'étudier.

5.8.6 Suppression

Supprimer un objet sélectionné est encore plus simple : il suffit de taper X (ou encore la touche Suppr), puis de confirmer la destruction en cliquant la mention « Erase Selected » dans le message qui apparaît aussitôt sous le pointeur de la souris (vous pouvez aussi annuler l'opération en tapant Echap).

5.8.7 Déformation

La scène sur laquelle nous travaillons représente des meubles dont les tailles sont réalistes et cohérentes, et cela n'a guère de sens de leur donner la taille de jouets pour maison de poupée. En revanche, on peut être tenté d'allonger le vaisselier pour le transformer en bibliothèque, ou d'élargir le fauteuil pour en faire un canapé.

Si l'on veut accomplir cette opération avec la souris, le mieux est de commencer par faire pivoter l'objet sur lui-même pour le replacer bien parallèlement aux axes. Il peut aussi être utile de le faire pivoter autour d'un axe horizontal de façon que les dimensions dont on veut modifier la proportion soit toutes deux visibles en vue de dessus (par exemple, si l'on veut allonger un cylindre en lui gardant une section circulaire, il faut le disposer à l'horizontale... quitte à le redresser ensuite). Cela étant fait, la déformation va s'opérer en deux temps : d'abord on changera les proportions de l'objet, et ensuite, éventuellement, on procédera à une mise à l'échelle de l'objet déformé. Retenez bien cela : on travaille d'abord sur les proportions, et on ne se soucie de la taille de l'objet qu'une fois que l'on est satisfait de ses proportions.

Placez le pointeur sur une fenêtre 3D, de préférence la vue de dessus. Sélectionnez l'objet, placez le pointeur un peu à droite ou à gauche du point magenta,

tapez S (pour « scale ») et cliquez sur le bouton central de la souris (cliquez simplement, ne maintenez pas la pression). A ce détail près, l'opération ressemble beaucoup à une banale mise à l'échelle : à partir de cet instant, tout déplacement vers la droite ou la gauche allongera ou étroitisera l'objet, selon que le pointeur s'éloignera ou se rapprochera du point magenta. Comme toujours, vous pouvez valider la nouvelle forme de l'objet avec un clic gauche, ou abandonner l'opération en tapant Echap.

Rappelons que vous venez de travailler sur les proportions de l'objet vu en projection, et non sur sa taille globale : par voie de conséquence, il est possible qu'après la déformation l'objet vous paraisse désormais trop petit ou trop grand. Il vous suffit alors de procéder à une mise à l'échelle banale (voir ci-dessus) pour corriger cet inconvénient.

5.9 L'enregistrement des fichiers

A coup de duplications, de déformations et de modifications de coordonnées, vous voici déjà capable de modifier la scène assez profondément pour avoir envie de conserver une trace de votre travail. L'enregistrement des fichiers avec Blender n'a plus grand-chose pour vous surprendre si vous avez bien compris les principes de la fenêtre « File Browser » que nous avons vue plus haut (fig. 5.11). Bien entendu, si l'on souhaite simplement enregistrer la scène en cours en écrasant la version précédente, il suffit de choisir la commande « Save » dans le menu « File » (Blender demande quand même confirmation en faisant apparaître un message sous le pointeur de la souris, mais cet aspect de l'interface vous est déjà bien connu).

Toutefois, il peut être nécessaire d'enregistrer la scène sous un autre nom ou à un autre endroit de l'arborescence (en particulier, vous ne pouvez évidemment pas enregistrer votre travail sur le CD-ROM où vous avez lu le fichier `05_mobilier.blend`). Cela ne vous surprendra pas, il faut donc choisir la commande « Save As... » dans le menu « File », ce qui transforme aussitôt l'une des fenêtres de l'écran en la fenêtre « File Browser » qui nous est déjà connue (Fig. 5.11).

Tout ce que nous avons déjà dit à son sujet reste valable, notamment en ce qui concerne la possibilité d'agrandir la fenêtre, mais il va vous falloir en outre indiquer sur la seconde ligne de la fenêtre le nom sous lequel vous voulez enregistrer le fichier. Cliquez donc sur cette ligne (clic gauche, banal), effacez ou modifiez éventuellement la mention que Blender a pu y inscrire, et tapez à la place le nom qui vous convient (les fichiers de Blender portent normalement l'extension « .blend », mais Blender la rajoutera de lui-même si vous l'oubliez).

Il ne vous reste qu'à valider en tapant Entrée... deux fois. Oui, deux fois : ce n'est qu'à la deuxième fois que l'enregistrement sera effectif (et que la fenêtre

« File Browser » reprendra son aspect antérieur). Ce serait dommage de ne pas le savoir, car alors vos modifications seraient bel et bien perdues !

Pourquoi cette bizarrerie – qui a suffi à convaincre bien des utilisateurs que la fonction d'enregistrement de Blender était gravement boguée ? Parce que la 3D est une discipline très minutieuse, où les fausses manœuvres sont fréquentes, et que pour cette raison il est prudent de s'habituer à ne pas effacer immédiatement les versions antérieures d'une scène sur laquelle on travaille. Blender considère donc qu'il est normal d'enregistrer une scène avec un numéro de version : scene1.blend, scene2.blend, scene27.blend, etc.

Quel rapport ? Vous allez voir. Recommencez l'opération : menu « File », commande « Save As... », choisissez le bon répertoire, cliquez sur la deuxième ligne de la fenêtre « File Browser », tapez un nom de fichier (par exemple, « scene ») – mais là, attention, variante ! Au lieu de taper Entrée deux fois comme nous venons de le voir, interrompez-vous après la première ; puis, tapez la touche + du pavé numérique, et relisez votre nom de fichier.

Vous voyez l'astuce ? Blender a ajouté un chiffre à ce nom, et « scene » est devenu « scene1 ». Si vous continuez à taper sur la touche +, la numérotation défilera : « scene2 », « scene3 », etc. ; et si vous tapez la touche - du pavé numérique, la valse des chiffres repart, mais dans l'autre sens.

Et quand le nom numéroté de votre fichier vous convient, il ne vous reste qu'à taper Entrée... une seconde fois. Voilà l'explication du mystère de la double validation, et voilà pourquoi les trois quarts des personnes qui ont téléchargé Blender sans se soucier de son mode d'emploi n'ont jamais réussi à enregistrer leur travail !

5.10 Le paramétrage des caméras

5.10.1 Le choix de la caméra en vigueur

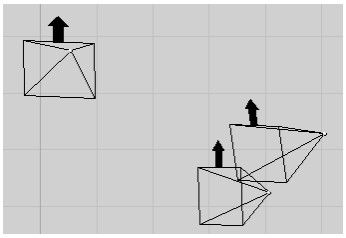


FIG. 5.13 – *Trois caméras de Blender. La flèche noire que chacune d'entre elles porte indique la direction du haut de l'image de son point de vue.*

Comme vous l'aurez remarqué par vous-même, la scène du fichier `05_mobilier.blend`

comporte, outre les meubles eux-mêmes, des objets en forme de pyramide, et comportant en outre une flèche dirigée vers le haut (Fig. 5.13). Peut-être l'aviez-vous déjà compris : ces objets symbolisent des caméras (c'est sous ce nom qu'on a coutume de les désigner même en français, mais « camera » signifie plutôt « appareil photo »). Les vues en perspective que Blender est capable d'afficher sont toujours connectées à l'une d'entre elles.

Une scène de Blender peut comporter un nombre indéfini de caméras, mais une seule est prise en compte à un moment donné pour les vues en perspective et le calcul des images de synthèse.

Pour choisir la caméra en vigueur, il faut commencer par la sélectionner par un clic droit (ce qui suppose bien sûr qu'elle soit visible sur l'une des vues en 3D). Cela fait, on place le pointeur de la souris sur la vue en perspective (si toutefois il y en a une), on enfonce la touche Ctrl et on tape la touche 0 du pavé numérique (en maintenant la pression sur la touche Ctrl ; autrement dit, on tape Ctrl-0 – et c'est bien cette formulation que nous emploierons désormais) : immédiatement, la fenêtre 3D où se trouve le pointeur représente la scène en perspective du point de vue de la caméra qu'on vient de sélectionner.

5.10.2 Le positionnement en projection

Une caméra est un objet presque comme un autre, et elle peut être déplacée et pivotée sur les vues en projection, exactement comme nous l'avons fait pour les autres objets : la vue en perspective varie en conséquence. En revanche, une mise à l'échelle d'une caméra ne modifie pas le taux d'agrandissement de la vue en perspective qui lui est associée, et ne présente donc pas d'intérêt.

5.10.3 Réglage de la caméra du point de vue du caméraman

Bien que ce ne soit pas toujours plus pratique, on peut préférer mettre en place la caméra en se mettant dans la peau d'un caméraman virtuel, qui observe la scène l'œil rivé au viseur. Dans cette hypothèse, il faut commencer par sélectionner la caméra et créer une vue en perspective lui correspondant (en tapant Ctrl-0 avec le pointeur sur l'une des fenêtres en 3D). Bien entendu, on place le pointeur sur cette fenêtre (à peu près au centre, de préférence).

Cela ne vous surprendra pas : pour déplacer la caméra, il faut taper G ; et pour en changer l'orientation, il faut taper R. Ensuite, fort classiquement, on choisit de nouveaux paramètres en déplaçant le pointeur de la souris, et on valide avec un clic gauche (ou bien on abandonne l'opération et on retrouve les paramètres de départ en tapant Echap).

Soyons plus précis : l'appui de la touche G suivi de déplacements latéraux du pointeur permet au cameraman de se déplacer « en crabe », tandis que des

déplacements du pointeur vers le haut ou vers le bas de l'écran permettent au caméraman de monter ou descendre comme dans un ascenseur.

Pour des raisons assez obscures, ces déplacements s'effectuent en sens opposé de ce à quoi on s'attendrait (tout se passe comme si l'on déplaçait la scène par rapport à la caméra et non le contraire) : le cameraman descend quand le pointeur monte, il se déplace vers la gauche quand le pointeur va vers la droite. De toute façon, le résultat est immédiatement visible, on n'a donc guère de peine à repérer et corriger les fausses manœuvres.

Un peu plus difficile : pour avancer ou reculer, il faut taper G, cliquer sur le bouton du milieu (on peut ensuite relâcher la pression), et pousser la souris vers l'avant ou l'arrière (ce qui revient à déplacer le pointeur vers le haut ou le bas de l'écran). Le déplacement s'effectue alors à une majestueuse lenteur, jusqu'à ce que l'on valide la nouvelle position par un clic gauche (ou qu'on annule l'opération en tapant Echap).

Pour ce qui est des rotations, les mouvements du pointeur après l'appui de la touche R permettent de changer l'orientation de la caméra en dévers... ce qui, en fait, est rarement utile à moins que l'on ne soit un fanatique des cadrages « psychologiques » avec l'horizon de travers ! Cela peut quand même servir à l'occasion.

Mais si l'on souhaite tout bêtement orienter la caméra un peu plus à gauche, un peu plus haut... il faut taper R et cliquer sur le bouton du milieu ; dès lors, les mouvements du pointeur permettront rapidement d'aboutir à un cadrage correct, que l'on validera bien sûr par un clic gauche.

5.10.4 Le choix de la focale

Par défaut, les caméras de Blender ont une focale moyenne de 35 mm, mais on peut choisir des valeurs plus élevées pour de pseudo-vues au téléobjectif, ou plus basses pour des vues grand-angulaires. Cela va nous fournir l'occasion d'une courte incursion dans les recoins cachés de la fenêtre « Buttons Window » qui occupe le bas de l'écran.

Avant tout, sélectionnez la caméra dont vous voulez modifier la focale. Si ce n'est déjà fait, tapez Ctrl-0 avec le pointeur sur une fenêtre 3D, pour obtenir une vue en perspective lui correspondant.

FIG. 5.14 – *Le bloc des icônes figurant sur la ligne d'en-tête d'une fenêtre Buttons Window. L'icône de droite, grisée, est validée.*

Repérez maintenant la petite barre d'icônes de l'en-tête de la fenêtre du bas de

l'écran (Fig. 5.14). Placez le pointeur sur l'icône la plus à droite, qui est censée représenter un paysage (et si vous attendez quelques instants, vous devriez voir apparaître une infobulle qui dit « Scene (F10) »). Cette icône est la seule de son lot qui soit actuellement grisée. Les innombrables boutons que vous voyez au bas de la fenêtre (les plus gros sont orangés et intitulés Render, Anim, Play) sont visibles si et seulement si c'est cette icône qui est grisée – autrement dit, validée.

Cliquez à présent sur l'icône juste à gauche de celle dont nous venons de parler ; elle représente un carré noir avec quatre coins jaunes (et si vous laissez le pointeur de la souris quelques instants dessus, cela devrait faire apparaître une infobulle qui dit « Editing (F9) »). Immédiatement, tous les boutons de la fenêtre du bas sont remplacés par un lot entièrement différent (Fig. 5.15)... qui comporte notamment le seul qui nous intéresse pour le moment (Fig. 5.16).

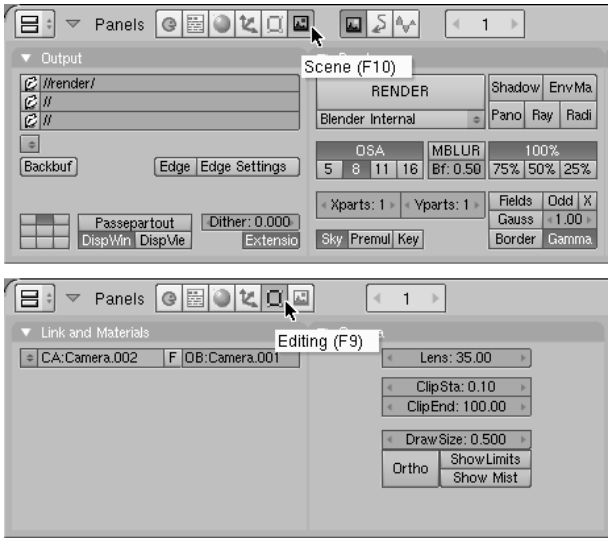


FIG. 5.15 – *La panoplie des boutons de la fenêtre Buttons Window est intégralement remplacée par un simple clic sur l'icône appropriée de l'en-tête.*

FIG. 5.16 – *Le bouton Lens, qui permet de modifier la focale d'une caméra. Remarquez les petites pointes de flèche à gauche et à droite.*

Voici l'occasion de découvrir l'une des spécificités les plus étonnantes de l'interface de Blender : cette minuscule zone grisée comporte bel et bien... un ascenseur (symbolisé par les minuscules pointes de flèche de part et d'autre). Un clic

sur sa partie droite, et la valeur qu'il affiche est majorée ; un clic sur sa partie gauche, elle est diminuée (et on peut bien sûr cliquer une dizaine de fois pour modifier la valeur de dix unités). Encore plus fort, si l'on clique sur cette zone, puis qu'on pousse le pointeur vers la droite en maintenant la pression sur le bouton de la souris, on voit les chiffres augmenter à toute vitesse dans la zone grisée... et dès qu'on relâche la pression, la vue en 3D est mise à jour avec la focale correspondante ! Enfin (et c'est quand même nettement le plus simple), on peut transformer cette zone grisée en petite fenêtre de saisie, en cliquant sur elle tout en maintenant enfoncée une touche Majuscule : on pourra alors effacer la valeur qui s'y trouve et en taper une autre, que l'on validera en tapant Entrée. Tout ça dans une zone de 15 pixels de haut !

Quand la focale vous convient, vous avez tout intérêt à revenir à la fenêtre « Buttons Window » classique en cliquant sur la petite icône représentant un paysage. Vous pouvez aussi taper la touche F10, conformément aux indications de l'infobulle.

5.11 Les autres modes de visualisation

5.11.1 Projection, axonométrie, perspective

Il nous reste à toucher un mot de quelques possibilités plus spectaculaires qu'utiles à ce stade de notre apprentissage... mais qu'il faut quand même connaître car il n'est pas rare qu'elles soient validées par erreur ; il faut au moins apprendre à en sortir !

Reprenez la scène `05_mobilier.blend` dans son état initial (Fig. 5.12). Vous pouvez constater que sur les trois vues en projection (dessus, face, profil), le quadrillage est fait de lignes parallèles et perpendiculaires. Il s'agit de vues techniques en projection, fort utiles pour le positionnement précis des objets, mais sans aucun effet de perspective. Eh bien, il suffit de placer le pointeur de la souris sur l'une quelconque de ces fenêtres, et de taper la touche 5 du pavé numérique, pour passer de cette vue en projection à une perspective classique – et inversement (Fig. 5.17).

Comme leur nom l'indique, les vues de dessus, de face et de profil ne permettent d'observer la scène que sous un angle bien précis, qui n'est pas toujours idéal. Il suffit de mettre en place une caméra à un endroit approprié pour pallier cet inconvénient, mais Blender offre aussi une autre possibilité un peu rustique, mais très rapide à manipuler (Fig. 5.18). On peut en effet faire pivoter n'importe laquelle de ces vues autour d'un axe vertical ou horizontal (et c'est possible aussi bien avec les vues auxquelles on a conféré un effet de perspective qu'avec les autres ; en revanche, cette possibilité n'existe pas avec une vue de caméra). Il suf-

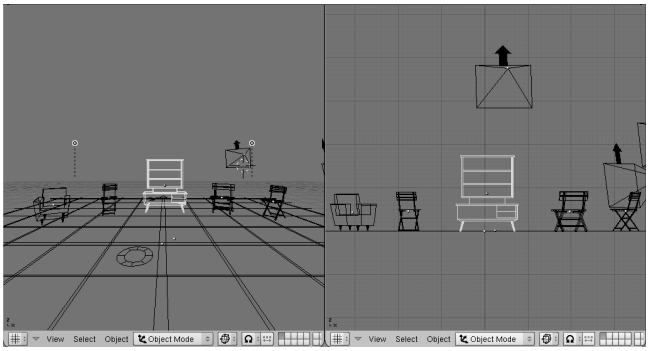


FIG. 5.17 – *La vue en projection de face, avec ou sans effet de perspective. Notez la disparition du quadrillage à gauche, et remarquez qu'il ne s'agit pas d'une perspective photographique : les lignes verticales restent rigoureusement parallèles.*

fit de placer le pointeur de la souris sur la fenêtre à laquelle on s'intéresse, et de taper les touches 2, 4, 6 ou 8 du pavé numérique. C'est bien sûr leur position sur le pavé numérique et non la valeur qu'elles portent qui compte : 4 pivote vers la gauche, 6 vers la droite, 8 vers le haut, 2 vers le bas (sur la plupart des claviers, ces touches portent d'ailleurs une flèche qui indique la direction pertinente).

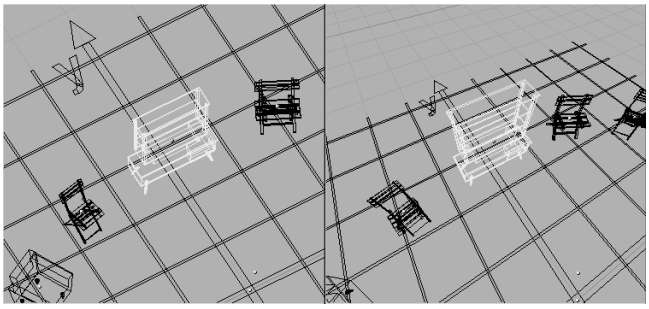


FIG. 5.18 – *Deux représentations de la scène sous un angle arbitraire, obtenues par tâtonnements avec les touches 2, 4, 6 et 8 du pavé numérique. La vue de gauche est une axonométrie : les lignes verticales et horizontales restent parallèles, et les proportions des objets ne sont pas affectées par leur éloignement.*

Il n'est pas du tout rare que la manipulation de ces touches mène à des représentations absurdes, par exemple avec la tête en bas. Il suffira de taper les touches 7, 1 et 3 pour retrouver les classiques vues de dessus, de face et de profil. Pensez aussi à taper sur la touche 5 si vous ne voyez pas réapparaître le quadrillage, qui est absent des vues en perspective sous la plupart des angles.

5.11.2 Bounding Box, Wireframe, Solid, Shaded, Textured

Jusqu'ici, nous avons toujours observé la scène en mode « fil de fer » (*wireframe* en anglais), c'est-à-dire que tous les objets étaient représentés comme un assemblage de traits droits, et qu'ils paraissaient transparents. Le menu de la fig. 5.19 (auquel on accède par un clic gauche sur l'icône appropriée dans l'en-tête d'une fenêtre « 3D View ») permet d'aboutir à d'autres modes de représentation, généralement moins lisibles mais quand même utiles dans certaines circonstances.

– **Bounding Box** (on peut traduire mot à mot par « boîte limitante », mais « limites extrêmes » est sans doute de meilleur français). Dans ce mode, les objets sont symbolisés par un simple parallélépipède ayant pour côtés leurs dimensions maximales en longueur, largeur et hauteur. Cela peut faciliter le travail sur les alignements et les superpositions, éviter un trop grand embrouillamini de traits avec des scènes très riches en détails, et aussi épargner des gaspillages de temps de réponse lorsque l'on manipule des objets aux innombrables facettes (des approximations de sphères, par exemple).

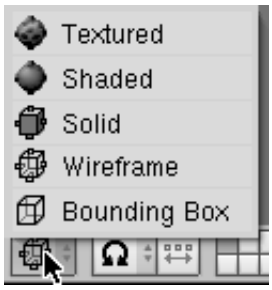


FIG. 5.19 – *Le menu des modes de visualisation.*

– **Wireframe.** C'est le mode par défaut et d'ailleurs le plus lisible (Fig. 5.20). Lorsqu'on veut y revenir après en avoir choisi un autre, on peut éviter le passage par le menu de la fig. 5.19 en tapant simplement la touche Z.

– **Solid et Shaded.** Ces deux modes représentent deux variantes d'une représentation 3D en « faces cachées ». Le premier prend en compte un éclairage fictif situé un peu au-dessus et à gauche de l'observateur, et qui se déplace en même temps que lui ; la lisibilité du résultat est médiocre, mais constante quel que soit le point de vue. Le second garde le même éclairage sous tous les angles, et donne donc des résultats plus ou moins lisibles selon l'endroit où l'observateur se place. Pour les amateurs de cuisine informatique : Solid fait appel aux techniques d'Open GL, et Shaded se conforme à l'algorithme de Gouraud.

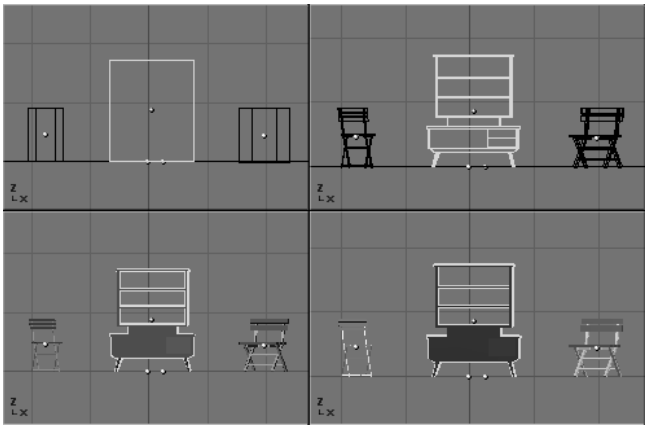


FIG. 5.20 – *La même scène représentée selon les modes de visualisation Bounding Box, Wireframe, Solid et Shaded (de gauche à droite et de haut en bas). La représentation en mode fil de fer est presque toujours la plus lisible, et il est en fait très rare qu'on l'abandonne au profit d'une autre.*

– **Textured.** Cette représentation n'a d'intérêt que pour les objets texturés – autrement dit, ceux sur lesquels on a plaqué un motif, un peu comme on appose du papier peint sur un mur. C'est la seule représentation qui prenne en compte ce motif, mais cela se fait au prix d'une dégradation notable des temps de réponse de Blender.

5.12 Les pièges de l'interface

Ce panorama de l'interface de Blender touche à sa fin, et nous avons déjà vu tout ce qu'il est nécessaire de savoir en utilisation courante du logiciel (tant qu'on ne fait que manipuler des objets en 3D déjà définis, sans s'attaquer à la création de nouvelles formes). Comme vous avez pu le constater, ce n'était pas si terrible : l'interface de Blender est assez bien pensée et même relativement simple. Son seul défaut, réel, est qu'elle n'a vraiment rien d'intuitif.

Bien évidemment, ce panorama n'est pas exhaustif. En particulier, la fenêtre « Buttons Window », celle que l'on place généralement au bas de l'écran, comporte d'innombrables boutons de toutes tailles correspondant en général à des utilisations très particulières. Au moins au début de votre apprentissage, nous vous recommandons d'adopter vis-à-vis de ces boutons la même politique que celle qu'on conseille aux boys-scouts à l'égard des champignons : si vous ne les connaissez pas... n'y touchez pas.

Chapitre 6

Vos premières images de synthèse

6.1 Production et enregistrement des images

6.1.1 Une interface modifiée

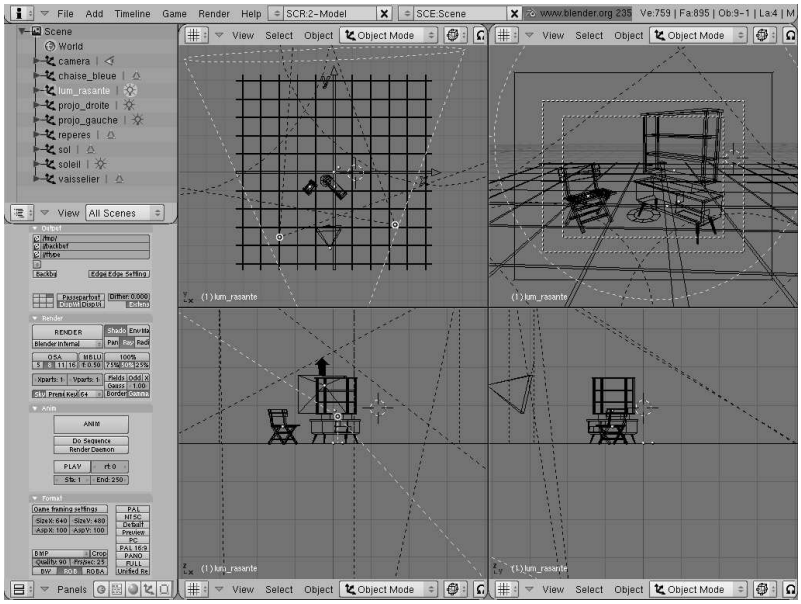


FIG. 6.1 – *Cette scène de Blender va nous servir à travailler sur les éclairages. Elle comporte déjà quatre sources de lumière, ce qui n'a rien d'exagéré.*

Pour ce chapitre, comme pour le précédent, nous allons travailler à partir d'une scène de Blender enregistrée dans le dossier `blender_fichiers` du CD-ROM joint à cet ouvrage. Il s'agit du fichier `06_eclairage.blend`, qui devrait avoir

l'aspect décrit sur la fig. 6.1. Nous y retrouvons les quatre fenêtres traditionnelles de la 3D : une vue en perspective représentant le cadrage d'une caméra (en haut à droite) et trois fenêtres disposées en L pour des projections en vue de dessus, de face et de profil. La fenêtre des boutons contient les mêmes éléments qu'au chapitre précédent, mais un peu réduits et disposés cette fois à la verticale, en bas à gauche de l'écran. Cela dégage un peu de place en haut à gauche pour une petite fenêtre de type « Outliner », où l'on voit figurer les noms de la dizaine d'objets qui constituent cette scène, y compris la caméra et les sources de lumière. Si vous sélectionnez un objet par un clic droit sur une fenêtre 3D comme nous l'avons vu au chapitre précédent, son nom va apparaître en blanc dans cette fenêtre, et symétriquement, si vous cliquez sur le nom d'un objet, ce dernier sera sélectionné dans les fenêtres 3D ; il sera donc représenté en magenta (mauve) à l'écran.

6.1.2 Les temps de réponse pour chaque image

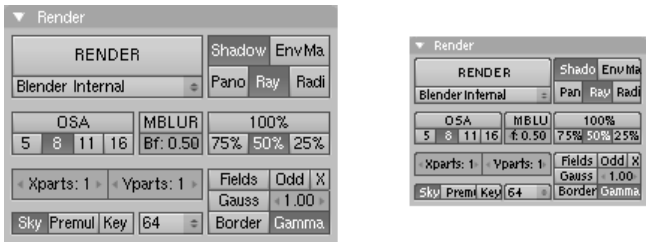


FIG. 6.2 – *L'onglet « Render » de la fenêtre des boutons, à gauche dans sa taille normale, à droite tel qu'il apparaît sur l'écran employé comme référence dans ce chapitre. Un clic sur le grand bouton orangé provoque la production d'une image de synthèse.*

Pour obtenir une image de synthèse, il vous suffit de cliquer sur le grand bouton orangé « Render » figurant sur l'onglet du même nom dans la fenêtre des boutons (fig. 6.2). Cela provoque aussitôt la création d'une petite fenêtre noire (indépendante de l'écran de Blender), que vous allez voir se remplir graduellement par le bas jusqu'à constituer une image de synthèse relativement élaborée (en particulier, elle comporte des ombres portées) (fig. 6.3).

Bien que la scène en question soit très simple d'un point de vue géométrique et que l'image soit de petite taille, vous avez probablement dû attendre quelques secondes pour qu'elle soit entièrement affichée. C'est ainsi : la technique du lancer de rayon est très gourmande en temps de calcul, même sur des machines puissantes. Gardez-vous bien du perfectionnisme qui consisterait à toujours exiger la

meilleure qualité d'image lorsque vous travaillez au brouillon, vous y perdriez inévitablement des heures. Prenez le temps de regarder un peu l'image dont vous venez de provoquer la fabrication, puis refermez-la comme une fenêtre ordinaire.

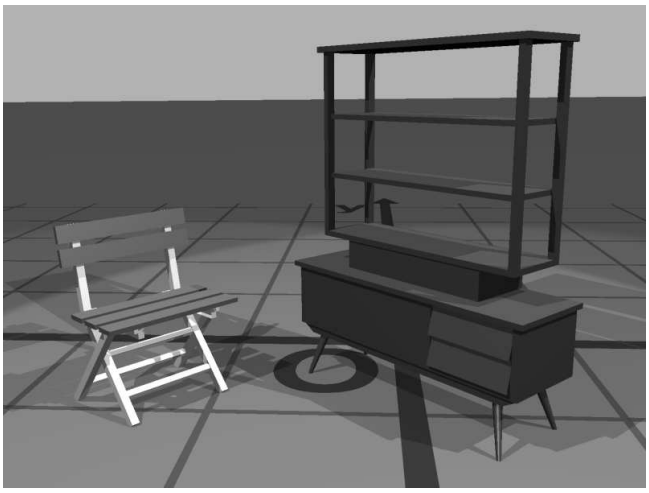


FIG. 6.3 – *Quatre sources de lumière, dont deux seulement à ombres portées, constituent l'éclairage de cette scène.*

Nous allons tout de suite apprendre à accélérer le processus de calcul de l'image de synthèse en réduisant un peu sa qualité (ce qui ne nous interdira évidemment pas de rétablir les meilleurs paramétrages par la suite). Blender peut considérablement réduire ses temps de réponse en employant des algorithmes qui ne relèvent pas véritablement de la technique du lancer de rayon, mais donnent un résultat très voisin. Repérez, à droite du grand bouton « Render » un groupe de cinq boutons plus petits de couleur bleu pâle (fig. 6.4). Deux d'entre eux sont actuellement validés. Cliquez simplement dessus pour qu'ils prennent la même teinte que les trois autres, puis cliquez à nouveau sur « Render ». Vous obtiendrez beaucoup plus rapidement et même presque instantanément une nouvelle image de synthèse... mais en compensation, vous constaterez que les ombres portées auront disparu.

Puisque nous tenons maintenant une technique qui produit rapidement des images, profitons-en et attaquons-nous à une image deux fois plus grosse. Inévitablement, cela demandera quatre fois plus de travail et donc de temps à votre ordinateur. Pourquoi quatre fois et non pas deux ? Parce que la géométrie est impitoyable : quand on multiplie par deux la largeur et la hauteur d'une image, sa surface et donc le nombre de points qui la constituent sont multipliés par... deux fois deux.

Juste sous le groupe de cinq boutons dont nous venons de parler, il y en a un

FIG. 6.4 – *Le calcul des images avec lancer de rayon (« Ray ») et ombres portées (« Shadow ») étant très gourmand en puissance, il est utile de savoir le neutraliser pour obtenir rapidement des brouillons des images de synthèse.*

autre de taille similaire, qui comporte un grand bouton et trois petits (fig. 6.5), intitulés 100 %, 75 %, 50 % et 25 %. C'est actuellement « 50 % » qui est validé. Cliquez sur « 100 % » et recommencez la production de l'image de synthèse pour obtenir cette fois une grande image. Vous n'aurez probablement pas besoin de sortir un chronomètre pour vous rendre compte de l'accroissement des temps de calcul : ils ont été multipliés par quatre.

Inutile de vous l'expliquer : au moins tant qu'on est au brouillon, on préfère se limiter à 25 % et 50 % pour travailler avec des ombres portées, mais on n'a aucune raison de se refuser au moins un taux honnête de 75 % si l'on n'a pas validé la technique du lancer de rayon.

FIG. 6.5 – *Deux réglages de rendu entre lesquels on jongle fréquemment tant qu'on travaille au brouillon : à gauche, une grande image calculée rapidement, mais sans ombres portées ; à droite, une toute petite image avec ombres.*

6.1.3 Taille de l'image et formats d'enregistrement

Le groupe de quatre boutons dont nous venons de parler est bien pratique pour passer rapidement d'une grosse image à une toute petite, mais Blender peut aussi réaliser des réglages plus fins et déterminer très exactement les dimensions en pixels de l'image de synthèse produite. Cela s'opère grâce aux boutons « SizeX » (largeur en pixels) et « SizeY » (hauteur en pixels) de l'onglet « Format » (fig. 6.6). Il s'agit de boutons-zones de saisie du même type que le bouton « Lens » que nous avons vu au chapitre précédent à propos du choix de la focale de la caméra. Un Majuscule-clic sur chacun de ces boutons vous permettra de régler les dimensions au pixel près. Les dimensions courantes pour des images photographiques sont 640 x 480, 800 x 600 et 1024 x 768... mais vous faites bien comme vous voulez !

C'est fort bien de parvenir à faire apparaître une image sur l'écran au bout de quelques secondes, mais il paraît encore plus souhaitable de l'enregistrer sur le disque dur, pour pouvoir la retrouver sans avoir à recommencer l'opération chaque fois qu'on veut la voir, et aussi pour pouvoir la retravailler avec un logiciel de retouche d'images. L'enregistrement s'opère par la commande « Save Image... » du menu « File », d'une façon très similaire à ce que nous avons vu au chapitre précédent pour l'enregistrement des fichiers de Blender – sauf qu'en fait d'enregistrement, il s'agit d'une recopie pure et simple de la dernière image de synthèse produite (même si elle a été refermée longtemps avant, même si vous avez changé les paramètres entre-temps et même... si vous avez chargé une autre scène !).

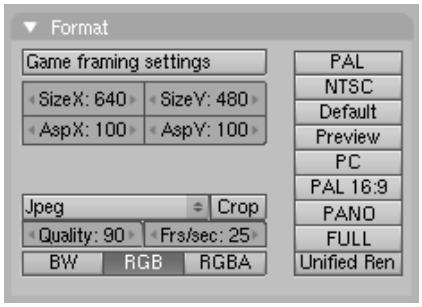


FIG. 6.6 – *L'onglet Format, avec les réglages par défaut de Blender. Notez en bas à gauche le choix (discutable) du format compressé Jpeg.*

Gare au bug à l'enregistrement !
Il faut signaler ici qu'on voit parfois se manifester un bug de Blender aux conséquences potentiellement désastreuses : le logiciel suggère parfois comme nom d'enregistrement celui du dernier document manipulé... et s'il s'agit de la scène elle-même (avec l'extension .blend) il ne faut surtout pas l'accepter ; au contraire, il faut impérativement employer l'extension .bmp, et surtout ne pas écraser la scène .blend de même nom. Méfiez-vous de ce bug, et méfiez-vous en aussi lors de l'enregistrement ultérieur de la scène elle-même (il peut arriver que le nom suggéré, ou même adopté automatiquement par la commande « Save », soit celui de la dernière image enregistrée et non celui déjà donné à la scène). Ce bug est très facile à contourner si l'on y prend garde (lisez bien les extensions !), mais une distraction en la matière peut faire perdre des heures de travail.

C'est donc juste avant de produire l'image de synthèse qu'il faut vous interroger sur son format d'enregistrement. Ce choix s'opère dans un sous-menu que l'on fait apparaître en cliquant sur le nom du format actuellement en vigueur, en bas à gauche de l'onglet Format (fig. 6.7). Il est préférable de choisir un format sans compression – quitte à en changer ensuite en reprenant l'image dans un logiciel de

retouche, qui permettra aussi d'effectuer un recadrage, un rééchantillonnage, un choix du taux de compression, etc. Sur les fichiers de démonstration du CD-ROM joint à cet ouvrage, nous avons généralement validé le format BMP (Windows bitmap), qui n'est pas plus mauvais qu'un autre et est reconnu par tous les logiciels de retouche actuels. Si cela vous convient, n'oubliez pas de terminer par .bmp le nom sous lequel vous enregistrerez votre image (vous devez indiquer explicitement cette extension, Blender ne la rajoutera pas pour vous).

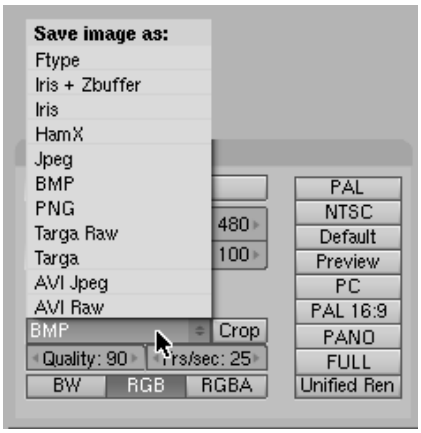


FIG. 6.7 – *Le choix des formats pour la production, et donc l'enregistrement, de l'image de synthèse. Il est préférable de choisir un format sans compression, par exemple BMP.*

Avant de poursuivre, nous vous recommandons de vérifier que vous parvenez bien à retrouver sur votre disque dur les images de synthèse que vous produisez. Essayez ensuite de les relire avec un logiciel de retouche d'image, comme par exemple Photoshop. Si vous n'en disposez pas, vous pouvez tout à fait employer à la place le logiciel libre Gimp, qui a été conçu pour les systèmes d'exploitation de la famille des Unix (notamment Linux et Mac OS/X) mais a depuis été adapté à Windows.

6.1.4 Utiliser les calques

Nous savons désormais produire et enregistrer une image de synthèse, il ne nous reste donc plus qu'à apprendre à nous amuser en réglant les éclairages ! Toutefois, avant cela, il nous faut encore découvrir une particularité de l'interface de Blender, qui est sa façon de répartir en calques (en anglais : *layers*) les différents éléments d'une même scène, de façon à pouvoir les masquer temporairement lorsqu'ils sont si nombreux et mélangés à l'écran qu'il devient difficile de les dis-

tinguer. La scène sur laquelle nous travaillons est assez dépouillée avec seulement une dizaine d'objets à prendre en compte, mais la confusion guette déjà. Nous avons donc choisi de réserver un calque aux sources de lumière.

FIG. 6.8 – *Les vingt petits boutons carrés commandant l'affichage des calques dans le menu d'une fenêtre 3D de Blender. Seul problème : la fenêtre est parfois trop étroite pour que ces boutons soient visibles...*

Pour ce qui concerne l'affichage des calques, l'interface de Blender est d'une sobriété qui confine au jansénisme : tout dépend de vingt minuscules boutons carrés, dépourvus de toute mention, à la droite du menu d'une fenêtre 3D (fig. 6.8) : si le bouton correspondant à un calque est grisé, les objets inclus dans ce calque sont visibles dans toutes les vues 3D, et seront pris en compte en cas de production d'une image de synthèse ; dans le cas contraire, tout se passera comme s'ils n'existaient pas... sauf qu'ils seront quand même enregistrés avec la scène.

Première difficulté : ces boutons ne sont pas forcément visibles, si toutes les fenêtres 3D sont trop étroites pour afficher leur menu dans son intégralité (comme dans la scène qui nous occupe). Ce n'est pas un bien gros problème, il suffit d'agrandir l'une quelconque de ces fenêtres en y plaçant le curseur et en tapant « Ctrl-flèche haut », comme nous l'avons vu au chapitre précédent ; une fois qu'on aura terminé de manipuler les calques, on tapera de nouveau « Ctrl-flèche haut » pour retrouver la configuration initiale de l'écran.

Si l'on veut qu'un seul calque soit visible, il suffit de cliquer sur le petit bouton carré qui le symbolise : ce calque sera validé, et tous les autres invalidés. Si l'on veut au contraire ajouter un calque aux éléments déjà visibles sans faire disparaître ces derniers, c'est à peine plus compliqué : il faut enfoncer la touche Majuscules, et cliquer sur le bouton du calque considéré en maintenant la pression sur cette touche. La même manipulation peut servir à extraire un calque de l'ensemble des éléments visibles.

Dans la configuration initiale du fichier `06_eclairage.blend`, deux calques sont validés : ceux qui correspondent aux boutons les plus à gauche. Le bouton du haut correspond au calque qui contient les objets 3D proprement dits (le « sol », le quadrillage servant de repères, et les deux meubles) ainsi que la caméra, et le bouton du bas à un calque que nous avons arbitrairement choisi de réserver aux sources de lumière. Regardez dans les fenêtres 3D quels éléments apparaissent et disparaissent selon que ces calques sont validés ou non.

Pour faire passer un objet d'un calque à un autre, il faut bien sûr commencer par le sélectionner, puis appeler la commande « Move to Layer... » du menu « Object » d'une fenêtre 3D. Le reste tombe sous le sens : sur la petite fenêtre qui apparaît, il faut cliquer sur le carré symbolisant le calque sur lequel on veut placer l'objet, puis cliquer « OK ».

6.2 L'éclairage des scènes

6.2.1 Type et orientation des sources de lumière

L'éclairage d'une scène 3D n'est pas une science exacte. Un éclairage doit toujours être adapté à la scène qu'il concerne... et même au point de vue d'où on la regarde : dans l'idéal, il faudrait retravailler les éclairages chaque fois qu'on déplace la caméra ! Pour vous en convaincre, vous pouvez vous amuser à placer une caméra derrière nos deux meubles et non plus devant eux, et à produire une nouvelle image de synthèse ; nul doute qu'elle sera totalement illisible.

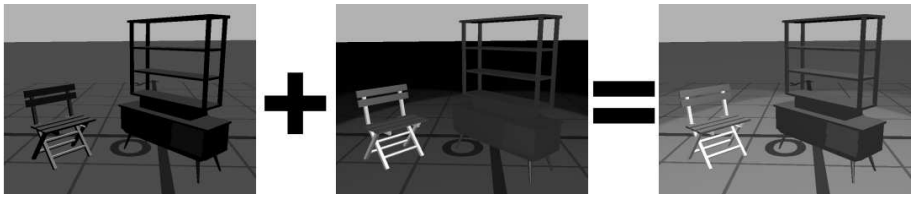


FIG. 6.9 – *Une scène bien éclairée résulte de l'addition d'éclairages insuffisants... A gauche, la scène n'est éclairée que par la source de lumière que nous avons baptisée « soleil » ; tout est visible, mais sans relief. Au centre, c'est le contraire : seuls les trois sources de lumière de type « spot » (qui n'éclairent pas dans toutes les directions mais seulement dans un cône de diffusion) sont actives.*

A défaut de règle absolue, on peut quand même énoncer quelques principes. D'abord, il vaut mieux multiplier les sources de lumière faibles que tout miser sur un ou deux éclairages violents. En second lieu, on obtient toujours des résultats plus lisibles lorsque l'essentiel des sources de lumière est placé en retrait de la caméra et pas trop loin d'elle. Toutefois, pour que les ombres portées aident bien à la compréhension de l'image, il faut prévoir au moins une source de lumière latérale qui projette des ombres.

En l'occurrence, nous avons opté pour quatre sources de lumière (c'est peu). Trois d'entre elles sont des « spots », la quatrième est un « soleil ». C'est très différent.

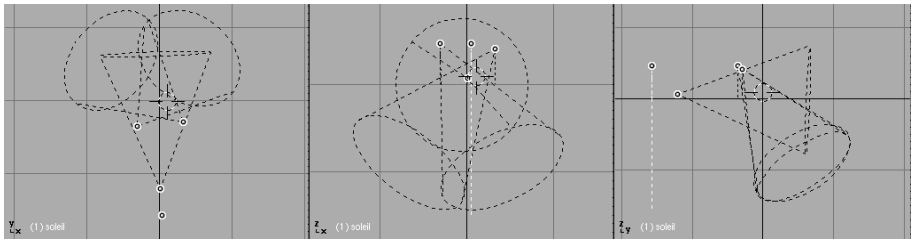


FIG. 6.10 – *Les sources de lumière de la scène, vues de dessus, de face et de profil. Blender symbolise les spots par des cônes, et le « soleil » par la ligne pointillée qui part d'un point.*

La lumière d'un spot est censée venir d'un point précis dans l'espace, et par ailleurs Blender affecte à chacune un cône de diffusion qui joue un peu le rôle d'un abat-jour ; c'est à cause de ces cônes que l'on peut remarquer à l'arrière-plan de l'image de synthèse (fig. 6.3) les contours de halos lumineux. Un spot a donc une orientation et un emplacement précis, et il joue un rôle très similaire aux projecteurs d'une salle de spectacles. Le moyen le plus compréhensible de le mettre en place est de le manipuler comme on l'a vu au chapitre précédent pour... les caméras : rien n'interdit de regarder la scène depuis le spot (on le sélectionne, puis on tape Ctrl-0) et d'utiliser cette faculté pour régler son orientation (bien entendu, il faudra ensuite rétablir la caméra comme référence des images en perspective, en la sélectionnant et en tapant à nouveau Ctrl-0 ; voir chapitre précédent).

La lumière d'un soleil, elle, fait davantage appel aux capacités d'abstraction de l'amateur de 3D. Blender la symbolise à l'écran non plus par un cône, mais par une ligne partant d'un point précis. Eh bien, l'emplacement de ce point n'a aucune importance : la lumière d'un soleil est toujours censée venir de l'infini, et est constituée de rayons parallèles à la ligne partant du point, dont seule l'orientation a de l'importance. En l'occurrence, la lumière de notre soleil est verticale, comme si le soleil était au zénith.

Entraînez-vous à manipuler tout cela. Amusez-vous à décaler ou à faire pivoter légèrement ces sources de lumière, puis à recalculer des images de synthèse.

6.2.2 Intensité des sources de lumière, ombres portées

Ce n'est pas tout de mettre en place des sources de lumière, il faut aussi pouvoir les paramétrer, notamment pour en régler l'intensité. Comme assez souvent avec Blender, la seule difficulté consiste à identifier l'emplacement des réglages pertinents.

Commencez par sélectionner la source de lumière dont vous voulez modifier

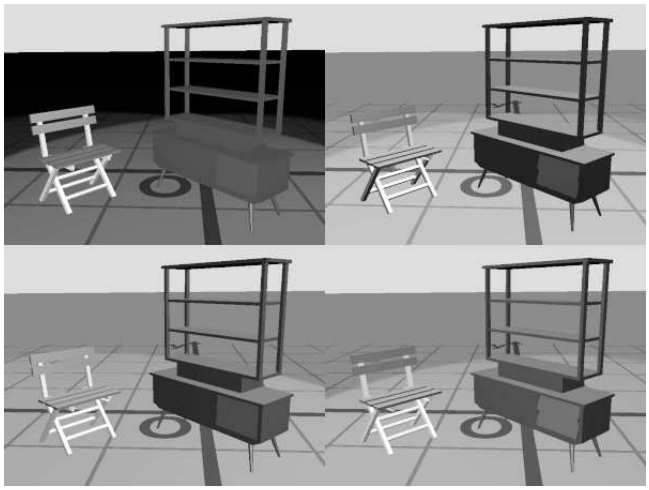


FIG. 6.11 – *Pour mieux comprendre le rôle de chacune de nos quatre sources de lumière, nous avons affiché quatre images de synthèse où l'une d'entre elles manque. En haut à gauche, il manque le « soleil » ; en haut à droite, la lumière rasante (la seule qui soit presque horizontale) ; en bas à gauche, il manque le projecteur gauche ; en bas à droite, le projecteur droit.*

le réglage. Ensuite, cliquez sur la petite sphère ombrée dans le premier ensemble d'icônes de la barre de menu de la fenêtre de boutons (vous avez tout intérêt à agrandir la fenêtre des boutons par Ctrl-flèche haut pour être certain de voir cette barre dans son intégralité). Cela aura pour effet de changer le deuxième groupe d'icônes, et c'est là que vous devrez sélectionner si ce n'est déjà fait le dessin de la petite lampe (fig. 6.12).

FIG. 6.12 – *Le menu habituel de la fenêtre de boutons, et celui auquel il faut aboutir pour travailler sur le paramétrage des sources de lumière.*

Cela fait, la fenêtre de boutons prend un tout autre aspect. Elle comporte désormais quatre signets, mais nous n'utiliserons que les trois premiers : Preview, Lamp, Shadow and Spot.

Le premier signet, Preview, permet éventuellement de changer le type de source de lumière pour transformer un spot en soleil et inversement. Par ailleurs, il comporte une fenêtre de visualisation qui donne une (vague) idée de l'intensité de

l'éclairage.

Cette dernière peut être réglée grâce au bouton Energy du second signet (intitulé quant à lui Lamp). On peut faire Majuscule-clic sur lui pour augmenter ou diminuer l'intensité de cet éclairage (la valeur de référence est de 1.0 ; on peut l'augmenter un peu, mais il est généralement plus utile de la réduire d'un tiers, d'une moitié ou des trois quarts lorsque l'on combine les sources de lumière). On peut aussi se contenter de déplacer le petit curseur à droite (mais attention à ne pas aller trop loin !). Les trois curseurs R, G et B placés juste en dessous dans le même signet permettent de remplacer la lumière blanche par une lumière colorée. A moins de rechercher des effets très particuliers, le réglage de base (valeur de 1.0 pour chacune des trois composantes), qui correspond à la lumière blanche, est généralement le plus pratique.

Enfin, dans le troisième signet intitulé Shadow and Spot, seul le bouton Ray Shadow (éventuellement abrégé en Ray Shad) nous intéresse : s'il est enfoncé, la source de lumière engendrera des ombres portées (sous réserve qu'on ait décidé de les afficher dans les options de rendu).

Quant aux autres réglages contenus dans les différents signets, il n'est peut-être pas urgent d'essayer de les comprendre...

6.3 Ajouter des éléments à la scène

6.3.1 Les formes primitives

A ce stade de votre apprentissage, vous commencez très certainement à avoir envie de vous essayer à concocter vos propres scènes en 3D. Patience, vous y êtes presque, mais pour le moment vous ignorez encore comment faire apparaître à partir de rien la moindre forme dans Blender.

Presque tout ce que l'on peut ajouter à une scène de Blender (forme géométrique, source de lumière, caméra...) est obtenu à partir du menu « Add ». Ce dernier est normalement accessible par la barre de menu de la fenêtre « User Preferences », tout en haut de l'écran. Mais la plupart des utilisateurs de Blender préfèrent faire apparaître ce menu sous le curseur, au milieu d'une fenêtre 3D, simplement en tapant la barre d'espace.

Parmi les formes géométriques disponibles dans le sous-menu « Mesh » (on les appelle souvent des primitives), il faut surtout noter le plan (Plane, très utile pour recevoir les ombres portées), le cube, la sphère (UVSphere est une sphère classique, dotée de parallèles et de méridiens), le cylindre et le cône. Bien entendu, ces objets aux formes supposées rondes sont en fait des assemblages de polygones, et si vous demandez une de ces formes, Blender vous laissera décider du nombre de facettes à gérer en indiquant un paramètre dans une petite fenêtre (fig. 6.14).

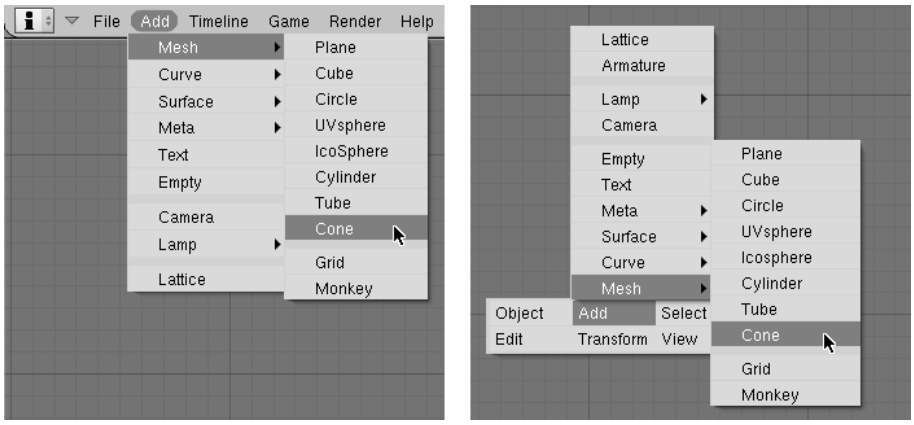


FIG. 6.13 – *Ajout d'un cône dans la scène à l'aide des deux formes du menu « Add » : à gauche, dans la barre de menu de la fenêtre « User Preferences », à droite simplement en pressant la barre d'espace lorsque le pointeur de la souris est au milieu d'une fenêtre 3D.*

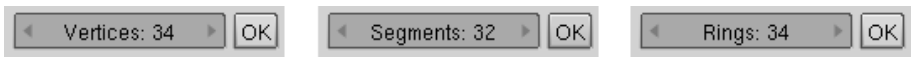


FIG. 6.14 – *Les paramètres demandés par Blender pour la création d'un objet rond (Majuscule-clic sur la valeur permet de la modifier manuellement). « Vertices » (pluriel de l'anglais vertex, qui désigne un sommet) indique le nombre de points sur le pourtour d'un cône ou d'un cylindre, « Segments » et « Rings » correspondent aux méridiens et aux parallèles d'une sphère, respectivement.*

Au moment où un objet vient d'être ajouté à une scène, il a un aspect très particulier : il apparaît en noir et non en mauve, sauf ses sommets qui sont quant à eux coloriés en jaune. Nous apprendrons au chapitre suivant à quoi cela correspond. Pour le moment, contentez-vous de savoir qu'il faut immédiatement repasser à la représentation normale en tapant la touche Tabulation, ou en passant d'« Edit mode » à « Object mode » dans la barre de menu d'une fenêtre 3D (fig. 6.15).

6.3.2 Paramétrage et nommage des objets

Vous pouvez évidemment modifier les proportions des formes que vous ajoutez à la scène grâce aux manipulations que nous avons décrites dans le chapitre précédent, mais il existe une technique alternative qui est souvent bien plus rapide. Quand un objet est sélectionné et que le pointeur de la souris se trouve au-dessus d'une fenêtre 3D, tapez simplement la touche « N » (pour « Numerical Values ».

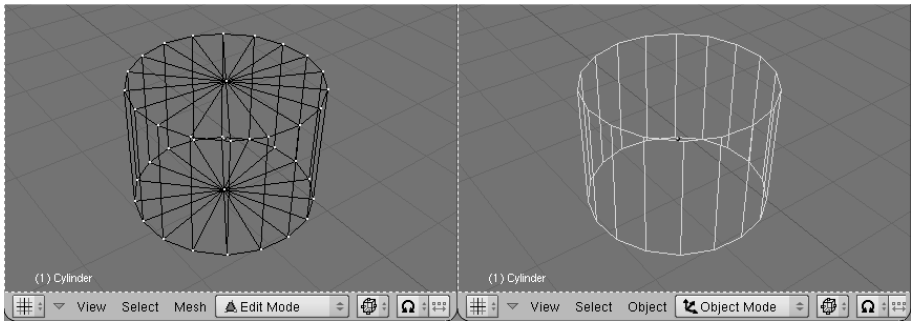


FIG. 6.15 – *A gauche, un cylindre en « Edit mode », tel qu'il se présente immédiatement après sa création. A droite, le même en « Object mode », sous son aspect normal. Remarquez que ces mentions apparaissent dans la barre de menu, où on peut donc les modifier.*

Cela fait immédiatement apparaître en surimpression une petite fenêtre intitulée « Transform properties », pourvue de plusieurs zones de saisie (fig. 6.16 ; comme d'habitude, vous pouvez modifier la valeur de n'importe laquelle d'entre elles en faisant un Majuscule-clic dessus). La modification des paramètres SizeX, SizeY et SizeZ vous permettra de transformer les cubes en parallélépipèdes ou d'allonger les cylindres bien plus facilement qu'avec la souris.

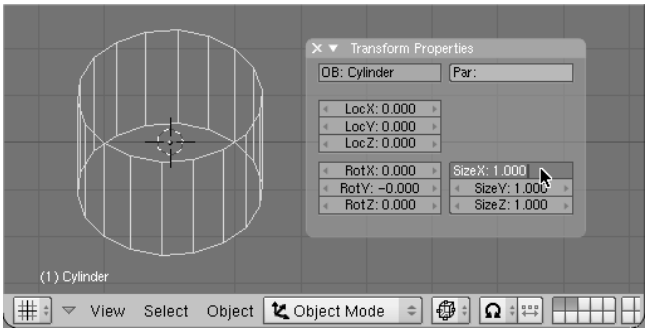


FIG. 6.16 – *La fenêtre « Transform properties » apparaît lorsque l'on tape sur la touche... N (on a déjà fait plus mnémotechnique, mais c'est ainsi) et permet de modifier l'emplacement et les proportions d'un objet, ainsi que son orientation et même son nom.*

Au passage, signalons que la zone de saisie en haut à gauche de cette fenêtre permet de donner à chaque objet que vous créez un nom plus intelligible que « cube » ou « cylindre ». Le nom que vous inscrivez là apparaîtra au bas de chaque fenêtre 3D chaque fois que vous sélectionnerez l'objet... et pourra vous aider à

faire transiter un objet d'une scène à une autre (nous en reparlerons).

Le X en haut à gauche de la fenêtre permet de la refermer, comme vous l'aviez probablement deviné.

6.3.3 Fusion de plusieurs formes en un seul maillage

On peut toujours sélectionner plusieurs objets en même temps avec Majuscule-clic droit pour pouvoir les manipuler simultanément. Mais il peut se révéler encore plus utile de les transformer en un tout indivisible, qu'on pourra toujours sélectionner d'un seul clic droit. Pour réunir plusieurs objets (sélectionnés simultanément) en un seul maillage, il suffit d'avoir recours à la commande « Join objects » du menu « Object ». Blender vous demandera confirmation de cette opération, car elle est définitive : les objets ne pourront plus être modifiés un par un. N'ayez donc pas recours à cette technique avant d'être pleinement satisfait de l'ensemble que vous voulez ainsi souder à tout jamais. Regardez-le bien sous tous les angles... et surtout prenez l'habitude d'enregistrer votre travail sous un nom spécifique (multipliez les numéros de version : scene1.blend, scene2.blend, etc.) avant et après chacune de ces fusions : revenir à une vieille version est en la matière le seul moyen d'annuler les effets d'une fausse manœuvre.

6.3.4 La mise en couleur d'une forme

Les objets mis en place dans la scène grâce au menu « Add » n'ont pas de couleur définie au départ (ils apparaissent en gris sur les images de synthèse). Pour imposer une couleur à un objet, vous devez d'abord le sélectionner, puis passer en mode « Shading » en cliquant sur la petite sphère ombrée dans le premier groupe d'icônes de la barre de menu d'une fenêtre 3D (fig. 6.17 ; nous avons déjà vu une manipulation similaire pour le réglage des sources de lumière).

FIG. 6.17 – *Le menu habituel de la fenêtre de boutons, et celui auquel il faut aboutir pour effectuer la mise en couleur d'une forme.*

Cela a pour effet de remplacer l'ensemble des boutons de la fenêtre. Si la forme sélectionnée n'a encore reçu aucune mise en couleur, il vous faudra aussi cliquer sur le bouton « Add New » de l'onglet « Material » pour faire apparaître les curseurs permettant la mise en couleur.

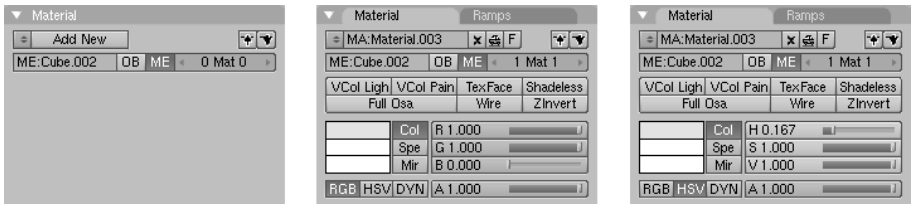


FIG. 6.18 – *L'onglet « Material », à gauche tel qu'il se présente pour une forme non encore mise en couleur, au centre et à droite sous son aspect normal. Notez qu'au centre « RGB » est validé, tandis qu'à droite c'est « HSV » ; dans les deux cas, les réglages correspondent à du jaune pur.*

Hélas, Blender a été écrit par des informaticiens, et bien évidemment il effectue la mise en couleur selon le modèle « red, green, blue » de la synthèse additive, auquel personne à part les écrans de télé ne comprend rien... Vous avez tout intérêt à cliquer en bas à gauche sur « HSV » pour adopter le modèle « hue, saturation, value » (voisin de celui dont nous avons parlé lors de la définition du format SGML), qui est quand même un peu moins incompréhensible (fig. 6.18). Poussez les curseurs S et V tout à fait à droite, de façon à obtenir des couleurs pures, puis choisissez la nuance qui vous convient avec le curseur H (la prévisualisation vous aidera à y comprendre quelque chose). Quand vous l'aurez trouvé, si elle vous paraît trop agressive ou trop sombre, vous pourrez arranger cela sans trop de peine, par tâtonnements, avec les curseurs S et V.

6.3.5 Passage d'un objet d'une scène à l'autre

Il vous arrivera sans doute souvent de vouloir reprendre dans une scène un objet que vous aviez mis au point pour une autre. Ne cherchez pas le copier-coller dans les menus de Blender, qui n'emploie pas cette technique. Pour récupérer un objet dans une autre scène que celle où vous travaillez, il faut simplement aller le chercher avec la commande « Append » du menu « File ». Vous pourrez chercher la scène contenant l'objet qui vous intéresse dans l'arborescence du disque dur, de façon classique, et quand vous l'aurez trouvée, Blender l'ouvrira exactement comme si c'était un dossier du disque dur. Vous y trouverez une demi-douzaine de sous-dossiers dont un nommé « Object », dans lequel il faudra entrer. C'est là que vous trouverez les objets de la scène visitée, ou plutôt... leurs noms, que vous aurez donc eu intérêt à rendre explicites préalablement ! Un clic central sur le nom de l'objet désiré, et voilà : l'objet se retrouvera dans votre scène aux coordonnées qu'il avait dans l'autre... et aussi, attention, sur le calque qu'il occupait précédemment : cela peut donner à tort l'impression que cette importation d'objet

a échoué.

6.3.6 Et maintenant, au travail !

A ce stade de votre apprentissage, vous ne pouvez pas encore vraiment créer vos propres scènes... en tout cas pas sans y consacrer beaucoup plus de temps que nécessaire (vous apprendrez à travailler plus vite dans les chapitres qui suivent). Mais qu'à cela ne tienne : lancez-vous ! Vous en savez déjà assez pour commencer à imiter des objets du monde réel en partant de formes géométriques simples : parallélépipèdes, cônes, cylindres, sphères. Surtout, si vous ne répugnez pas à retomber un peu en enfance en jouant au Lego et au Meccano sur ordinateur, vous pouvez déjà obtenir des effets graphiques intéressants rien qu'en combinant ces formes, et en jouant astucieusement sur leurs proportions, leur emplacement et leur orientation.

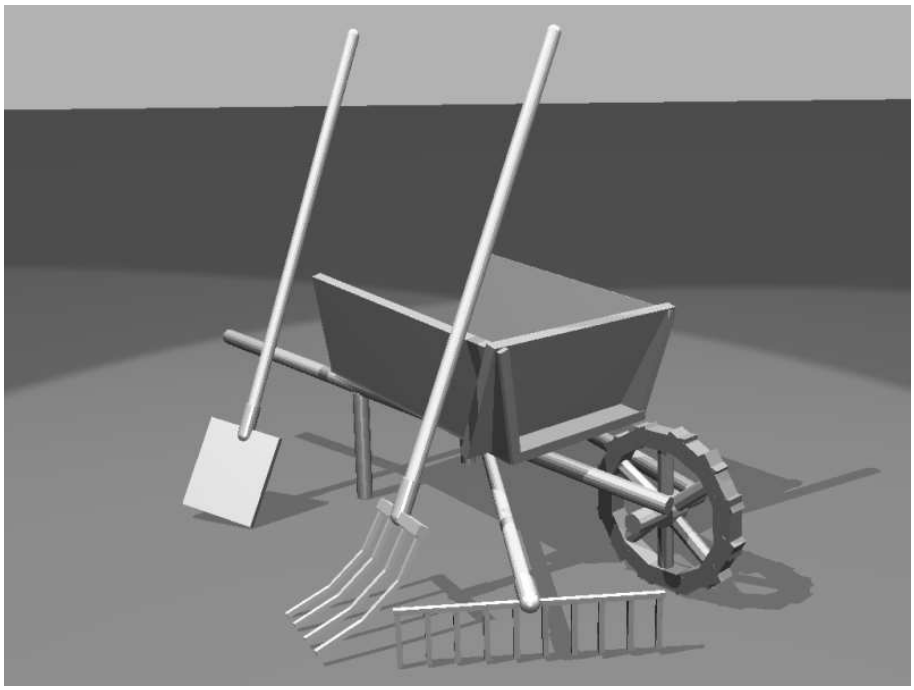


FIG. 6.19 – *Les éléments de cette scène ont été exclusivement modelés avec des sphères, des cylindres et des parallélépipèdes. C'est évidemment loin d'être parfait (la roue est faite de parallélépipèdes !), mais ce genre de Lego ou de Meccano est tout à fait à votre portée désormais.*

Vous allez certainement vite vous rendre compte de l'extrême méticulosité nécessaire pour réaliser manuellement la mise en place précise des composants

d'une scène. Sentez votre douleur, et réjouissez-vous-en ! Ce sera pour vous le début d'une réflexion critique sur les vertus comparées des manipulations à la souris (que vous connaissez désormais) et de celles du code informatique (dont vous découvrirez bientôt la souplesse et la puissance).

Chapitre 7

Le modelage des formes

7.1 Le mode d'édition de points

7.1.1 Une interface déjà connue pour l'essentiel

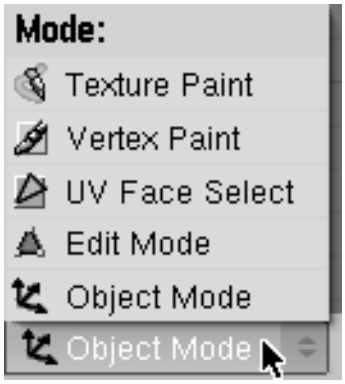


FIG. 7.1 – *Le menu des modes de travail des fenêtres 3D de Blender. On passe facilement du mode de déplacement des objets (« Object Mode ») à celui d'édition des points (« Edit Mode »), et inversement, en tapant sur la touche Tabulation.*

Les fenêtres 3D de Blender comportent dans leur barre de menu un sous-menu pourvu d'icônes, qui affiche habituellement une triple flèche symbolisant un repère XYZ et la mention « Object Mode ». Ce mode de travail, qui permet la mise en place des objets, est pratiquement le seul que nous avons employé jusqu'à présent. Nous sommes tout de même très brièvement passés en « Edit Mode » dans le chapitre précédent, lors de la création des objets, et nous allons maintenant y revenir pour nous attaquer au modelage des formes.

Pour ce faire, nous pouvons cliquer sur le sous-menu des modes (fig. 7.1) et choisir « Edit Mode » dans le menu, mais il est beaucoup plus simple et plus rapide de taper simplement sur la touche de tabulation (la même manipulation sert d'ailleurs pour revenir au mode normal de manipulation des objets). Notez que cette modification affecte toutes les fenêtres 3D simultanément.

Dans le mode habituel, un objet sélectionné apparaît sous la forme d'un assemblage de traits mauves. En « Edit Mode », les traits deviennent noirs, mais tous les sommets des polygones constituant l'objet sont clairement signalés par des points de couleur (mauves normalement, jaunes si ces points sont sélectionnés). Bonne nouvelle : tout ce que vous savez faire en « Object Mode » avec des objets et qui peut être transposé à des points (sélections multiples avec Majuscule-clic droit, déplacements avec la touche G, changements d'échelle avec S, rotations avec R) est effectué rigoureusement de la même façon avec des points en « Edit Mode » : vous avez déjà appris l'essentiel de ce qu'il faut savoir en la matière.

Toutefois, vous allez probablement devoir consacrer un certain temps d'apprentissage à acquérir de la virtuosité dans le maniement des vues 3D. En effet, les points d'une forme mathématique (cube, cylindre...) sont au départ parfaitement alignés sur des directions parallèles aux axes, d'où de grands risques de confusion quand on travaille sur des vues en projection : par exemple, les quatre points de la face supérieure d'un cube sont totalement confondus avec ceux de la face inférieure dans la vue de dessus. Il faut donc apprendre à travailler sur plusieurs projections simultanément, à vérifier ce que l'on fait en passant de temps à autre en vue en perspective (touche 5 du pavé numérique), à tourner autour de l'objet (avec les touches 2, 4, 6 et 8 du pavé numérique), à jongler avec les taux d'agrandissement, etc. Nous avons déjà vu tout cela, mais ces manipulations doivent être parfaitement maîtrisées lorsque l'on s'attaque au modelage des formes.

En particulier, prenez l'habitude de taper très souvent sur la touche Z pour passer d'une représentation « fil de fer » à la visualisation en « faces cachées » : cette dernière vous permet de masquer les points des faces situées à l'arrière-plan, ce qui peut se révéler très pratique dès que les sommets d'un objet commencent à être nombreux.

7.1.2 Déformation des maillages

Vous avez donc désormais la possibilité de déplacer n'importe quel point de n'importe quel maillage sélectionné, ce qui va vous permettre d'en changer très notablement la forme sans la moindre difficulté : les cubes peuvent ainsi devenir des pyramides tronquées, et les sphères se laissent très facilement modifier en « patatoïdes » (fig. 7.2). Toutefois, notez bien que pour le moment, vous n'ajoutez ni ne retranchez aucun sommet, et les maillages que vous déformez comportent

toujours le même nombre d'arêtes éventuellement disposées en méridiens et parallèles (même si ce dernier terme risque fort de devenir abusif...).

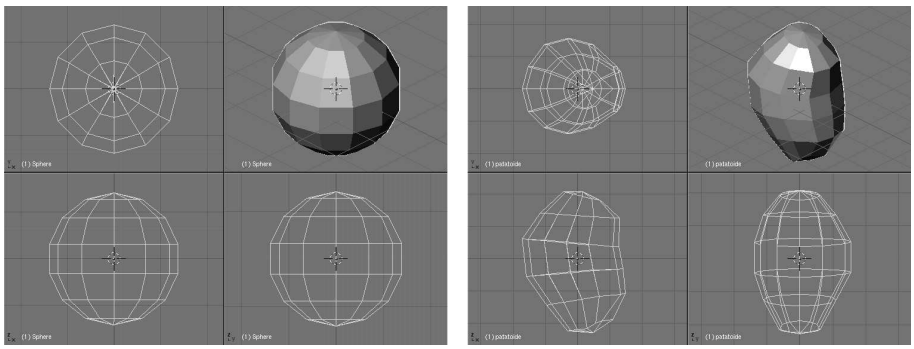


FIG. 7.2 – *Le modelage porte parfois bien son nom, et peut effectivement conduire à déformer les figures géométriques comme si elles étaient faites en pâte à modeler. Notez qu'en l'espèce, le maillage comporte toujours le même nombre de points, toujours disposés en « parallèles » et méridiens.*

7.2 Le rectangle de sélection et son usage

On a assez rarement besoin de traiter un maillage comme de la pâte à modeler. La plupart du temps, on va au contraire prendre grand soin de conserver des symétries et des alignements lorsque l'on modifie les maillages. Pour la parfaite régularité de ces opérations, le mieux est de travailler en sélectionnant dans le maillage des ensembles de points représentant des unités logiques – par exemple l'un des parallèles d'une sphère. Il est toujours possible de sélectionner tous ces points un par un (avec Majuscule-clic droit), mais il existe fort heureusement une manipulation plus pratique : la définition d'un rectangle de sélection.

Pour sélectionner une série de points dans une zone rectangulaire, commencez par choisir une vue ou ces points sont bien détachés des autres. Par exemple, les parallèles d'une sphère constituent des lignes bien distinctes dans les projections de face ou de profil, et c'est donc avec l'une de ces dernières qu'il faudra travailler. Quand vous avez mentalement repéré le rectangle de sélection que vous voulez tracer, commencez toujours par sélectionner l'un des points de cet ensemble (cela ne paraît pas présenter d'intérêt ; en fait, c'est une façon de s'assurer qu'aucun point extérieur à la zone considérée n'est resté sélectionné par erreur ou distraction). Cela fait, placez le pointeur de la souris dans le coin supérieur gauche de la zone que vous avez repérée, et tapez la touche B (pour « Border selection »). Ensuite, cliquez (clic gauche) et maintenez la pression sur le bouton de la souris

en déplaçant le curseur jusqu'au coin inférieur droit de la zone qui vous intéresse (au cours de cette opération, Blender dessine très discrètement un rectangle pointillé autour de la zone de sélection). Quand vous avez fini d'englober les points qui vous intéressaient, relâchez la pression sur le bouton de la souris : instantanément, tous les points inclus dans la zone prennent la couleur jaune, signe qu'ils sont bien tous sélectionnés et vont pouvoir être manipulés globalement.

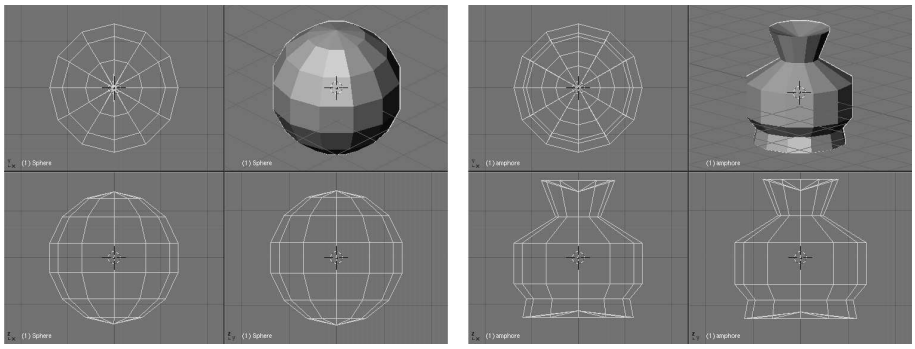


FIG. 7.3 – *Tous les objets qui paraissent pouvoir être moulés sur un tour de potier (en anglais :* lathe*) sont faciles à réaliser par la déformation d'une sphère comportant un nombre adéquat de parallèles et de méridiens. Bien entendu, on modèle ces objets en sélectionnant les points parallèle par parallèle.*

Notez bien que ce mode de sélection ne mène pas aux mêmes résultats selon que l'on choisit d'y procéder sur une vue en fil de fer ou en faces cachées. En fil de fer, tous les points dont la projection se trouve dans le rectangle de sélection seront pris en compte (y compris si, en projection, ils sont superposés à d'autres) ; en faces cachées, au contraire, seuls les points situés au premier plan seront sélectionnés. Il faut y prendre bien garde lorsque l'on travaille sur un objet symétrique à partir d'une vue de profil.

7.3 La création de nouveaux points et de nouvelles faces

7.3.1 Extrusion de faces

Même si elles sont assurément utiles, toutes ces possibilités de déplacement des sommets des primitives géométriques sont insuffisantes pour décrire des formes découpées et complexes. Il va donc nous falloir apprendre à ajouter des points et des faces là où nous en avons besoin.

La technique la plus courante pour y parvenir est nommée extrusion. Elle consiste à conférer une épaisseur au périmètre d'une face, d'un polygone, ou même d'un ensemble de faces non disposées dans un même plan : tous les sommets de ce périmètre vont se trouver dupliqués à quelque distance et le périmètre lui-même sera encerclé par une sorte de « palissade » de faces nouvellement créées (fig. 7.4). En outre, si les points sélectionnés suffisaient à eux seuls à délimiter des faces avant l'extrusion, ces faces seront surélevées.

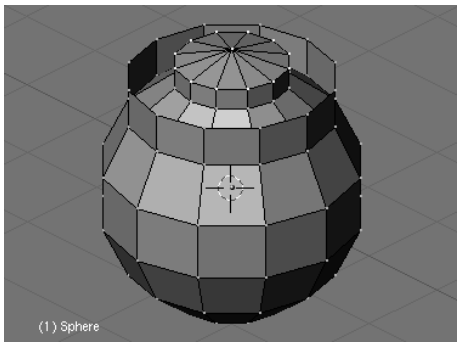


FIG. 7.4 – *Cette sphère a fait l'objet de deux extrusions : les treize points enserrant les douze triangles de sa calotte polaire ont été surélevés ; les douze points de son deuxième parallèle ont subi le même traitement, mais comme ils ne délimitaient pas des faces à eux seuls, l'extrusion a seulement engendré une sorte de palissade circulaire.*

Pour aboutir à ce résultat, il faut commencer par sélectionner un périmètre de points, puis placer le curseur au-dessus d'une fenêtre 3D permettant de visualiser le résultat (pour une extrusion en hauteur, on choisira donc une vue de face ou de profil). Il faut alors taper la touche E (pour extrusion, bien entendu), ce qui fera apparaître sous le curseur un sous-menu de Blender consacré à l'extrusion ; en l'occurrence, il faudra cliquer sur l'option « Region », après quoi l'extrusion proprement dite (la création de la palissade, en quelque sorte) aura lieu en suivant le déplacement du curseur et, comme d'habitude, il faudra valider l'opération par un clic gauche ou y renoncer en tapant la touche « Echap ».

Bien entendu, les points créés par l'extrusion peuvent être manipulés, globalement ou individuellement : on n'est évidemment pas obligé de toujours procéder aux extrusions en ligne droite.

A noter qu'au terme de l'extrusion, les points du nouveau périmètre sont eux-mêmes sélectionnés. Cela permet éventuellement de faire suivre par une deuxième, une troisième extrusions, etc., indéfiniment. On peut ainsi, à partir d'un tube ou d'un cylindre, créer en un tournemain assez de points pour définir la forme d'un tuyau coudé.

7.3.2 Extrusion de points et créations de faces

Mais on entre vraiment dans le domaine de la création en 3D maillée lorsque les formes que l'on crée ne cherchent plus à avoir le moindre rapport avec les primitives géométriques. Le principe est on ne peut plus simple : on met en place trois points, et on définit une face triangulaire ; puis on ajoute un point à côté (ou au-dessus !) de ce triangle, et en l'unissant à deux des points déjà définis, on crée un nouveau triangle qui touche le premier par un côté. Et ainsi de suite... à l'infini.

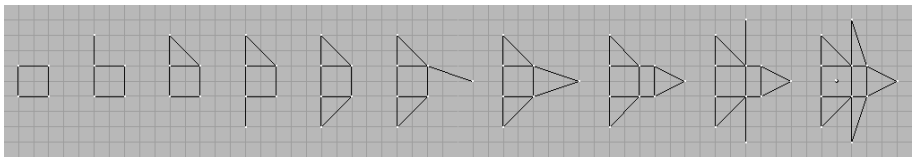


FIG. 7.5 – *Elaboration d'une forme complexe à partir d'un simple carré par ajout graduel de points, puis de triangles. N'importe quelle forme de 3D maillée peut être élaborée selon ce principe.*

A dire vrai, avec Blender, c'est juste un peu plus compliqué : un point n'est pas créé à partir du néant, mais à partir d'un sommet quelconque d'une face quelconque... pourvue qu'elle soit déjà définie. On a donc coutume de commencer le modelage libre d'un objet à partir d'un cube, voire d'un simple plan. On en sélectionne un point, puis on en crée un autre à peu près comme au paragraphe précédent : on tape la touche E (toujours pour extrusion), ce qui fait apparaître le même sous-menu que précédemment, mais on clique cette fois l'option « Only Vertices » ; il apparaît alors un nouveau point qui suit les mouvements du curseur ; on valide sa nouvelle position par un clic gauche, ou on annule l'opération avec la touche « Echap ».

Ayant ainsi défini un ou des nouveaux points, il suffit de délimiter une nouvelle zone en sélectionnant trois d'entre eux, puis de taper F (pour face) pour faire apparaître un nouveau triangle ayant les trois points sélectionnés pour sommets (il est préférable de vérifier sa présence en mode d'affichage « faces cachées », en tapant la touche Z). Blender admet aussi que l'on définisse des quadrilatères par la même méthode, mais il faut s'attendre à des résultats douteux si l'on sélectionne quatre points qui ne figurent pas dans le même plan.

7.3.3 Suppression de faces

Bien entendu, quand on travaille à élaborer un maillage libre, il arrive couramment que l'on se rende compte en cours de route qu'on s'y est mal pris. On peut

donc souhaiter supprimer des points, ou encore les conserver mais en organisant différemment le découpage de la surface en triangles.

Rien de plus simple que la suppression d'un point : on le sélectionne, on tape X (ou la touche Suppr), on choisit « Vertices » (mot anglais pour « sommets », comme nous l'avons déjà vu) dans le menu « Erase » qui apparaît sous le curseur – et voilà, le point est supprimé. Fort logiquement, toutes les faces qui avaient ce point pour sommet sont également éliminées.

Il est beaucoup plus délicat de retravailler un maillage sans supprimer des points. Si l'on sélectionne tous les points du périmètre d'une face, et qu'on tape de nouveau la touche X, on peut choisir l'option « Faces » dans le menu « Erase » qui apparaît alors sous le curseur... mais cette option ne tient pas toutes ses promesses, et le plus courant est qu'elle élimine en plus de la face elle-même tous les points sélectionnés qui ne sont plus employés pour délimiter une autre face.

La solution est de relier arbitrairement le point dont on craint l'élimination à n'importe quel autre non concerné par l'opération de suppression de la face. Blender permet en effet de tracer de simples « arêtes virtuelles » entre deux points : on sélectionne ces derniers, et on tape F comme pour la création d'une face. Si par la suite (quand le point ainsi préservé aura été employé pour définir le périmètre d'une face) on souhaite se défaire de cette arête inutile, il suffira de sélectionner ses deux sommets et, là encore comme on le ferait pour une face, de taper X ; mais cette fois, c'est « Edges » (mot anglais pour arêtes) qu'il faudra choisir dans le menu « Erase » apparaissant sous le curseur.

7.3.4 Positionnement par rapport à la grille

Pour obtenir facilement des alignements parfaits lors des extrusions de points ou de faces, on peut s'aider du quadrillage affiché par Blender : lors de l'extrusion, alors que l'emplacement des points nouvellement créés n'a pas encore été validé par un clic gauche, il suffit d'enfoncer la touche Ctrl pour que ces points ne se déplacent plus que par sauts d'une unité entière parallèlement aux axes X, Y et Z. Ou, pour le dire autrement, lorsque l'on procède aux extrusions avec la touche Ctrl enfoncée, il est facile d'aligner les points sur le quadrillage (à la condition de procéder à l'extrusion de points disposés eux-mêmes sur le quadrillage). Hélas, cela ne fonctionne pas à tous les taux d'agrandissement, mais seulement sur les lignes représentant les unités entières – donc des valeurs exprimées en théorie en mètres. En d'autres termes, si on veut se servir de cette faculté pour dessiner un petit objet, il faudra commencer par le modeler beaucoup trop gros, à l'échelle métrique, et le réduire ensuite. Cela peut quand même rendre d'immenses services.

7.3.5 Modification du point de référence

Juste à la gauche des boutons des calques, la barre de menus des fenêtres 3D comporte un menu à icônes (fig. 7.6) dont nous ne nous sommes jamais servis, car l'option que Blender lui donne par défaut (« Bounding Box Center ») est très généralement la plus pratique. Son option « 3D Cursor » peut cependant être occasionnellement utile, et en particulier pour la modification d'une partie d'un maillage.

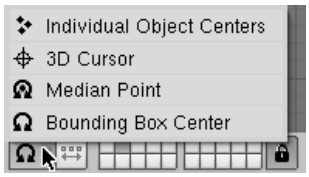


FIG. 7.6 – *Le menu des points de référence d'une fenêtre 3D. Attention, la modification de ce menu n'a d'effet que pour sa fenêtre 3D, et non pour toutes à la fois.*

Le curseur 3D dont il est question, c'est tout simplement la croix entourée d'un cercle rouge et blanc qui apparaît sous le curseur quand on effectue un clic gauche sur une fenêtre 3D, après quoi il reste en place. Sa fonction, vous l'aurez deviné, est de désigner un point précis de l'espace à trois dimensions (on ne sait donc pas exactement où il est placé si on ne le voit pas sur deux projections : vue de dessus et de face, par exemple). Eh bien, si l'on valide l'option « 3D Cursor », les rotations et les mises à l'échelle s'effectueront autour de ce point, et non plus à la moyenne des coordonnées de l'objet comme c'est habituellement le cas.

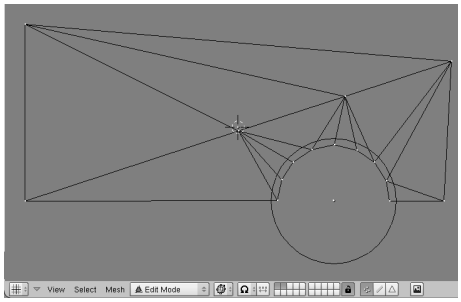


FIG. 7.7 – *Maillage d'une aile de voiture en cours d'élaboration. A l'évidence, l'espace circulaire réservé pour la roue est insuffisant. La solution paraît être d'effectuer une mise à l'échelle des points définissant le contour circulaire.*

Voilà qui est bien compliqué, il faut l'admettre, mais aussi fort utile dans des cas très particuliers, comme celui de la fig. 7.7. Nous sommes en présence d'un maillage qui a déjà demandé un certain travail, mais dont une partie – et une partie seulement – mérite d'être modifiée. En l'occurrence, il faudrait agrandir un arc de cercle sans toucher aux autres points du périmètre.

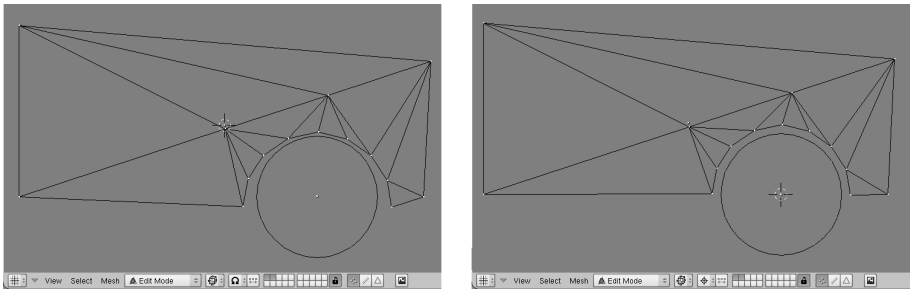


FIG. 7.8 – *A gauche, on a effectué la mise à l'échelle en prenant pour référence le centre du maillage. A droite, on a placé le curseur 3D au centre de la roue et validé l'option « 3D Cursor » avant de procéder à la mise à l'échelle.*

Il ne suffira pas pour cela de ne sélectionner que les points du contour qui pose problème, et de procéder à une mise à l'échelle de l'ensemble. En effet, si cette déformation est effectuée par rapport au centre de l'objet (fig. 7.8, partie gauche), l'arc de cercle sera certes agrandi, mais également éloigné par rapport à ce centre, d'où un décalage dont les défauts sont patents. Au contraire, si l'on place le curseur 3D au centre du cercle que l'on veut agrandir (et attention : dans le même plan, ce qui suppose de travailler sur deux vues 3D à la fois, faute de quoi le remède sera pire que le mal) et que l'on valide l'option « 3D Cursor », la déformation du maillage pourra alors donner en un tournemain exactement le résultat recherché (fig. 7.8, partie droite).

L'opération est aussi délicate à effectuer qu'utile. Nous vous recommandons de faire quelques tests pour vous entraîner en partant du fichier `07_aile_voiture.blend` inclus dans le dossier `blender_fichiers` du CD-ROM joint à cet ouvrage.

7.3.6 La démultiplication fractale

Lorsque l'on ne veut pas agrandir le maillage défini mais plutôt le préciser, il peut être utile de subdiviser des arêtes ou des faces qui existent déjà. La procédure est simple : on sélectionne tous les points de l'arête ou des faces à subdiviser, puis on tape W (abréviation probable de Wireframe, fil de fer... mais Blender ne mérite pas toujours les plus grands compliments pour son souci mnémotechnique).

Cela fait apparaître sous le curseur le menu « Specials », dans lequel on choisira simplement l'option « Subdivide ».

Cette possibilité peut se révéler occasionnellement utile, mais il est préférable de ne s'en servir que très occasionnellement : elle a en effet vite fait de multiplier jusqu'à l'absurde le nombre de points et de faces à gérer, ce qui a de fortes incidences sur les temps de réponse.

7.4 Validation d'un objet

Lorsqu'un maillage est satisfaisant pour les yeux, le travail n'est pourtant pas complètement terminé. Il reste à s'interroger sur son orientation, sa taille et son positionnement par rapport au point O, faute de quoi on risque d'avoir ensuite bien de la peine à l'intégrer dans une autre scène. Bon nombre des scènes enregistrées sur le CD-ROM joint comportent des repères fléchés porteurs des mentions O, x et y (fig. 7.9) et centrés sur le point O. Ce quadrillage de 10 x 10 m a pour fonction principale d'obliger à toujours réfléchir à ces questions.

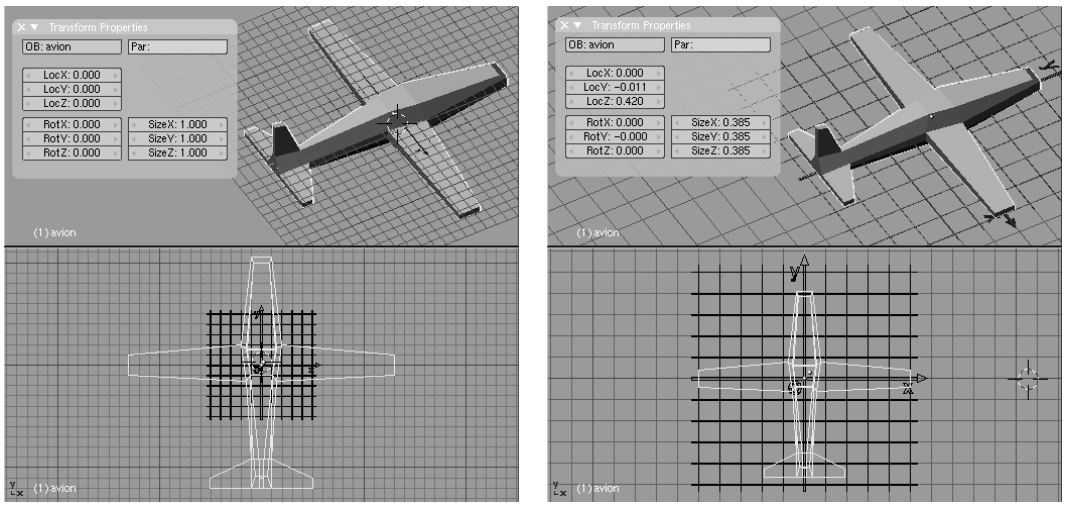


FIG. 7.9 – *A gauche, le maillage d'un petit avion, réalisé par... déformation d'un cube, en s'aidant du quadrillage de Blender : bien proportionné, mais beaucoup trop gros (26 m de large !). A droite, le même, mais ramené à une envergure plus raisonnable de 10 m.*

7.5 Exportation des données modelées

Quelles que soient les qualités de Blender, il peut être souhaitable d'exploiter les formes qu'il a modelées avec un autre logiciel de 3D ; en particulier, cet ouvrage traitera plus loin de leur intégration dans des images de synthèse produites par POV-Ray. Il va donc falloir convertir ces formes à un format moins difficile à exploiter que le format natif de Blender. Même si la commande « Export » du menu « File » possède d'autres options, c'est toujours au format Videoscape que nous aurons recours en import comme en export, du moins pour toutes les opérations réalisées avec l'interface de Blender (dans la partie suivante, nous emploierons le format SGML défini au début de cet ouvrage, mais Videoscape peut être converti depuis et en direction de la syntaxe SGML). Le format Videoscape présente l'avantage de conserver, outre les données géométriques, toutes les couleurs définies avec Blender.

Pour pouvoir exporter et gérer de façon simple un objet constitué de plusieurs maillages, il est nécessaire de les fusionner (voir, au chapitre précédent, « Fusion de plusieurs formes en un seul maillage »). Cette opération étant définitive, il est généralement préférable d'enregistrer toute la scène de Blender, d'abord sous le nom qu'on lui donne d'ordinaire, ensuite sous la forme d'un document provisoire. C'est avec ce dernier qu'on procédera à la fusion et à l'exportation. Il pourra ensuite être détruit. On retrouvera l'objet constitué de maillages indépendants dans le fichier Blender de départ, et on pourra donc éventuellement le retravailler.

L'opération d'exportation en elle-même n'a rien que de fort classique. Il suffit de sélectionner l'objet à exporter (et lui seul), puis de chercher dans le menu « File » la commande « Export », sous-option « Videoscape ». La fenêtre d'enregistrement de fichiers nous est désormais bien connue. Attention toutefois : les mises en garde que nous avons opérées au chapitre précédent (encadré « Gare au bug à l'enregistrement ! ») peuvent également être utiles lors de l'exportation (sauf que l'extension pertinente à donner au fichier enregistré est .obj et non plus .bmp).

Troisième partie

La 3D calculée et automatisée

Chapitre 8

Travailler la 3D au clavier

8.1 L'arborescence d'un disque dur

8.1.1 Quelques rappels terminologiques

Si ce que nous allons voir ici vous paraît élémentaire, vous ne vous trompez pas ! Il est quand même indispensable de faire certains rappels, en particulier terminologiques, pour les lecteurs qui ont découvert l'informatique de façon empirique avec une interface « conviviale » de type Windows. Sous prétexte de masquer la complexité, ces environnements graphiques se satisfont de simplifications totalement abusives, qui entraînent souvent des confusions intellectuelles très handicapantes. Par exemple, ces interfaces parlent de « couper-coller » lorsqu'un document est déplacé d'un dossier du disque dur à un autre... et du coup, bien des utilisateurs ne comprennent pas vraiment la différence entre un traitement de texte et un système d'exploitation (or, ça n'a rigoureusement rien à voir, et il est important de le comprendre !). Il est donc nécessaire avant de poursuivre de rappeler (ou hélas, d'enseigner) certains principes et termes informatiques dont la maîtrise sera indispensable pour la compréhension des prochains chapitres. Si les paragraphes qui suivent vous sont archi-connus, bravo, passez à la suite ! Mais s'ils vous causent la moindre difficulté de compréhension, faites-les vous expliquer par le premier bidouilleur venu ; c'est absolument indispensable.

Pour l'utilisateur d'une interface graphique, un document informatique se présente sous la forme d'une icône pourvue d'un nom assez bref et plus ou moins intelligible, du genre « Photo de mon beau-frère » ou « 10000327a.jpg ». Mais en termes techniques, ce document n'est rien d'autre qu'une longue suite de caractères informatiques enregistrés les uns derrière les autres : les **octets** (en anglais : **bytes**). Si l'interface graphique vous signale que ce document a une taille de 150 Ko (c'est-à-dire kilo-octets), cela signifie en termes techniques que c'est une longue suite d'environ 150 000 caractères informatiques (en fait, environ 153 600,

car un kilo-octet compte 1 024 octets et non 1 000, mais ne pinaillons pas). Un octet est assez assimilable à une lettre de l'alphabet (d'ailleurs, il sert en effet très couramment à enregistrer une lettre, un chiffre ou un signe de ponctuation)... à ceci près que l'alphabet informatique compte 256 de ces « lettres ».

Cette longue suite d'octets qu'on appelle un **fichier** (certains préfèrent le terme de **document** ; en anglais, on parle de **file**) est placée au milieu de centaines ou de milliers d'autres sur votre disque dur (ou sur un CD-ROM, une disquette, une clé USB...). Pour la récupérer sans risque d'erreur, on a besoin de connaître son **nom** (en l'occurrence, « Photo de mon beau-frère ») et son **emplacement** (pour l'interface graphique, l'emplacement est le nom du **dossier** (on dit aussi le **répertoire**, et en anglais, le **directory**) où se trouve l'icône du document, par exemple « Mes photos »).

Il est courant que le dossier en question soit en fait un sous-dossier (voire un sous-sous-dossier...). Par exemple, on peut trouver l'icône du dossier « Mes photos » dans le dossier « Mes documents ». Dans ce cas, en termes informatiques, on dira que « Mes documents, mes photos » est le **chemin d'accès** au fichier « Photo de mon beau-frère ».

Le nom du fichier lui-même est arbitraire, et s'il vous prenait l'envie absurde de renommer « Photo de mon beau-frère » en « Neuvième symphonie de Beethoven », ça ne changerait pas la nature ni le contenu du document informatique ; cela risquerait seulement de provoquer des confusions.

Pour un informaticien, « Photo de mon beau-frère » est un nom très mal choisi pour le fichier – comme d'ailleurs « Mes photos » un nom très mal choisi pour le dossier qui les contient. L'informaticien, lui, préférera un nom du genre `photo_beauf.jpg` : court, tout en minuscules, sans caractères accentués, sans caractères d'espacement (au besoin, on les remplace par un trait de soulignement (en anglais : underscore), c'est-à-dire le caractère qui se trouve le plus souvent au bas de la touche 8 de votre clavier), et terminé par un suffixe composé d'un point et de trois ou quatre lettres (plus rarement deux ou cinq). Ce suffixe, qu'on appelle une **extension**, est lui aussi arbitraire : si vous aviez l'idée saugrenue de renommer `photo_beauf.jpg` en `courrier_belle_mere.txt`, ça ne changerait ni la nature ni le contenu du document, ce serait seulement une source de confusion.

Cette extension a pour fonction de fournir rapidement une indication sur la nature du document informatique considéré. Par exemple, un fichier dont le nom est terminé par .jpg (en termes corrects, un fichier d'extension .jpg ; en termes courants, un fichier JPG) est a priori une image, généralement une photo (enregistrée selon les préconisations d'un comité professionnel appelé le Joint Photographic Expert Group, d'où le sigle JPEG ou JPG ; heureusement, l'explication des trois lettres d'une extension n'est pas toujours aussi compliquée !). Dans cet ouvrage, nous donnerons systématiquement des extensions pertinentes à tous les fichiers dont nous parlerons, et nous ne saurions trop vous recommander de prendre cette

habitude.

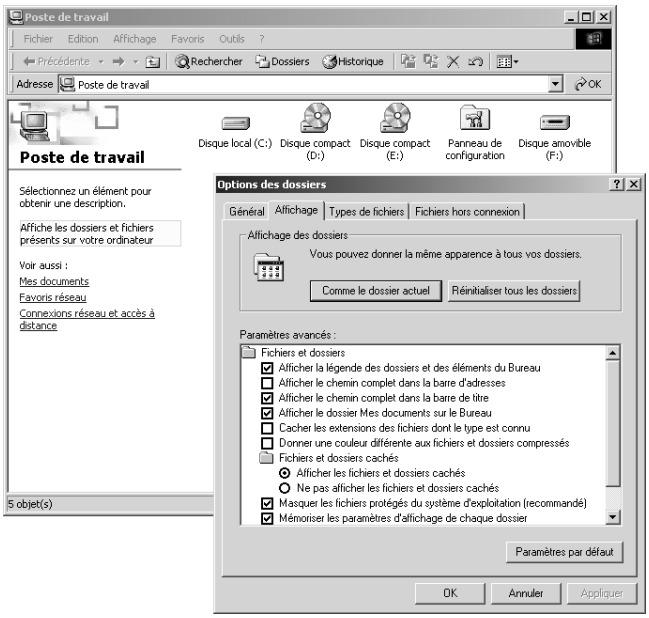


FIG. 8.1 – *Pour demander l'affichage des extensions sous Windows 2000 il faut décocher la case « Cacher les extensions des fichiers » dans l'onglet « Affichage » de la fenêtre « Option des dossiers », accessible depuis le menu « Outils » d'une fenêtre du bureau. C'est ce que chez Microsoft on appelle la « convivialité ».*

Certaines interfaces graphiques sont hélas configurées (par de sombres crétins du marketing qui jugeaient la chose trop compliquée pour leur petit cerveau) pour interdire aux fenêtres de l'interface graphique d'afficher les extensions, même quand on a eu la sagesse de les mettre ! C'est d'une stupidité révoltante et ce réglage absurde peut fort heureusement être modifié... hélas d'une façon que les mêmes crétins du marketing prennent grand soin de ne pas unifier d'une version de leur interface graphique à l'autre. Malheureusement pour vous, vous allez devoir rétablir le réglage pertinent vous-même, généralement en fouillant dans les sous-options d'un menu « Affichage », « Paramètres », « Outils »... de la fenêtre représentant le contenu d'un dossier quelconque. Si vous ne savez pas où trouver ce réglage, faites-vous aider de n'importe quel informaticien du dimanche, et profitez-en pour lui demander de vous répéter avec ses mots à lui tout ce que vous lisez dans ce chapitre : il s'agit de notions élémentaires, mais dont la maîtrise est indispensable.

8.1.2 Arborescence et syntaxe

La notion de chemin d'accès n'occasionne normalement pas de difficulté de compréhension aux êtres humains (sauf s'ils appartiennent au service de marketing de certains éditeurs informatiques...), mais c'est une source perpétuelle de problèmes de compatibilité entre ordinateurs. En effet, la façon de nommer et de disposer les dossiers et sous-dossiers (on appelle cela **l'arborescence**) est très mal unifiée d'une machine à l'autre. En outre, la façon de noter un chemin d'accès n'est pas la même sous Windows et sous les systèmes d'exploitation de tradition Unix, comme Mac OS/X ou Linux. Sous Windows, le nom complet d'un fichier (c'est-à-dire le nom précédé du chemin d'accès) est du type

```
c :\dossier\sous_dossier \fichier.ext
```

alors que dans la tradition Unix c'est plutôt du genre

```
/home/user/dossier/sous_dossier/fichier.ext
```

Outre que ces mentions bavardes ne commencent pas de la même façon, il faut noter une légère variante qui ne saute pas aux yeux d'un homme mais perturbe beaucoup les ordinateurs : Unix sépare les mots du chemin d'accès par une barre de fraction banale (un slash) et Windows par une barre de fraction inversée (un backslash).

Cela ne devrait pas être trop handicapant pour l'étude des pages qui suivent, ni même pour l'usage des convertisseurs du CD-ROM joint (ils sont normalement réglés pour remplacer automatiquement les slashes par des backslashes dans leur version pour Windows). Gardez tout de même le problème à l'esprit en lisant les pages qui suivent, car pour ne pas les surcharger inutilement, nous utiliserons systématiquement la syntaxe Unix (avec des barres de fraction normales) ; si vous travaillez sous Windows, il vous faudra donc penser seul à remplacer par des backslashes tous les slashes que nous aurons inscrit dans les lignes d'exemples (désolé de vous imposer cette corvée, vraiment... mais c'est le seul moyen de transposer à Windows le mode de fonctionnement efficace, puissant – et standard – des Unix).

Pour réduire au strict minimum les difficultés liées à cet exaspérant problème de syntaxe, nous avons pris le parti de ne pas démultiplier les niveaux hiérarchiques dans la partie de votre disque dur où vous effectuerez les manipulations. Vous travaillerez dans un dossier `3d_win`, `3d_mac` ou `3d_linux` selon le cas (vous l'installerez où cela vous chante sur votre disque dur, mais de préférence en haut de l'arborescence) et, sauf exception, tous les documents que vous manipulerez seront enregistrés dans un sous-dossier situé au premier niveau hiérarchique à l'intérieur de ce dossier (fig. 8.2). Cela vous permettra de faire référence aux documents d'un sous-dossier à partir d'un autre en utilisant une syntaxe beaucoup plus concise que celle du chemin d'accès complet. Par exemple, pour désigner le fichier `reperes.3db` du sous-répertoire `3db` quand vous travaillerez dans le

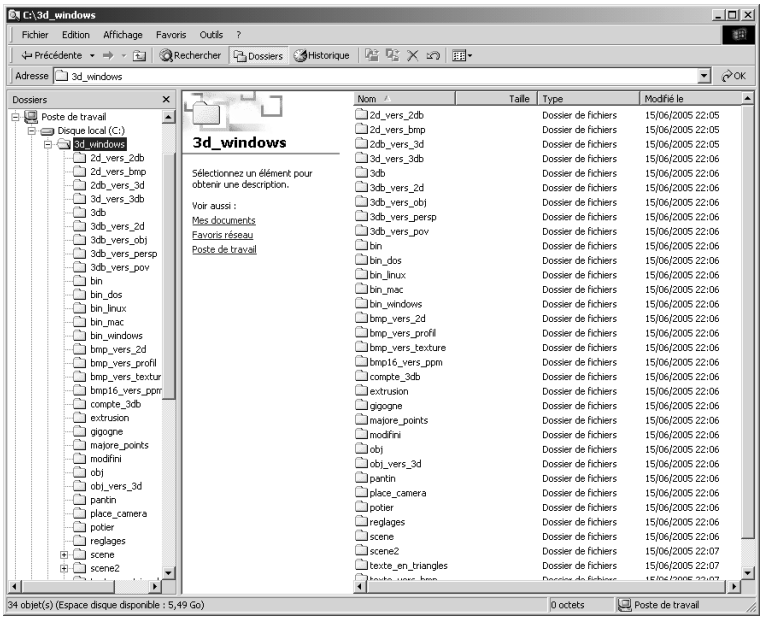


FIG. 8.2 – *Un dossier fourre-tout, avec de multiples sous-dossiers (on dit aussi sous-répertoires).*

sous-repertoire `scene`, vous écrirez simplement `../3db/reperes.3db` au lieu de quelque chose du genre

```
/Users/jeandupont/3d_mac/3db/reperes.3db
```

sur un Mac ou

```
c :\3d_win\3db\reperes.3db
```

avec Windows ; ce sera plus court et cela devrait fonctionner sans problème, quel que soit l'emplacement que vous aurez choisi pour copier le dossier de travail.

Nous reviendrons sur la syntaxe bizarre `../` dès le paragraphe suivant.

8.2 Cette bonne vieille ligne de commande

8.2.1 Pourquoi revenir à l'informatique de grand-papa

Tout ce que nous avons fait avec Blender était réalisé à la souris sur un bel écran coloré couvert de boutons comme un visage d'adolescent. Eh bien, il est temps de compléter ce « clicodrome » avec la bonne vieille ligne de commande chère à l'informatique pure et dure ! Ce n'est pas le passéisme qui va nous mener à y avoir recours, mais bel et bien la recherche de l'efficacité, voire d'une certaine forme de simplicité. Il faut bien comprendre qu'en informatique, ce qui est simple

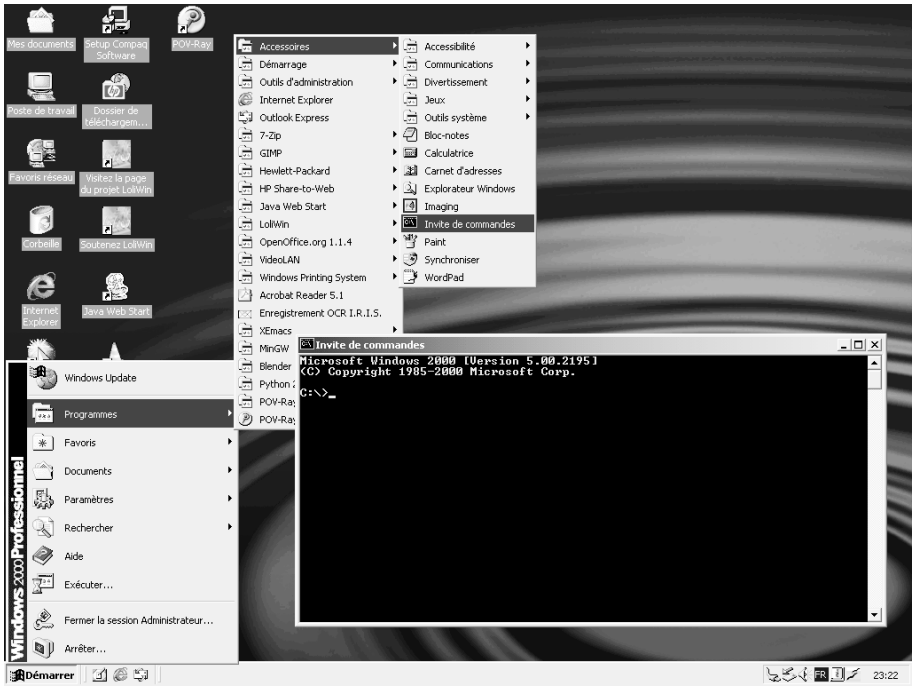


FIG. 8.3 – *La fenêtre noire de « l'invite de commandes » de Windows, avec les menus et sous-menus pour y accéder. Très laide et beaucoup moins élaborée que les shells d'Unix, elle n'en reste pas moins un moyen puissant d'automatiser les opérations.*

à manipuler pour l'utilisateur profane (l'interface graphique, la souris) ne l'est pas nécessairement pour l'informaticien qualifié. Réciproquement, ce qui est très rébarbatif pour l'utilisateur débutant (les commandes ésotériques dans une fenêtre noire) peut permettre au technicien d'automatiser en deux coups de cuiller à pot ce qui demanderait mille manipulations fastidieuses à la souris.

L'amateur de 3D, qui n'est ni un vrai profane ni un informaticien surqualifié mais un peu les deux à la fois, doit apprendre à se servir des deux techniques : il ne peut pas se passer de la souris pour l'aspect graphique de son travail, mais il ne peut pas davantage négliger la ligne de commande pour l'automatisation et le rendement. N'oubliez pas qu'une scène 3D demande la manipulation de dizaines de milliers de triangles ; ou on apprend à les traiter par des automatismes, ou l'on reste éternellement un béotien qui ne fait jamais mieux qu'empiler des cônes et des cylindres.

Par ailleurs, ce livre a pris pour parti de définir une méthode utilisable de façon similaire sur des ordinateurs très différents. Eh bien, il est beaucoup plus facile de définir des processus standards dans la fenêtre noire d'un terminal que dans le

capharnaüm que sont devenues les interfaces graphiques – là encore à la demande d'un marketing friand de gadgets graphiques toujours plus nouveaux... et toujours moins standards. Les petits convertisseurs inclus sur le CD-ROM joint n'ont pas grand-peine à produire les mêmes résultats sur une console Linux, un terminal Mac OS/X ou une « invite de commandes MS-DOS » de Windows, et c'est ce qui a permis à un amateur isolé comme l'auteur de les mettre au point. En revanche, c'est la croix et la bannière de réaliser une interface clicable identique avec les environnements graphiques associés à chacun de ces trois systèmes d'exploitation. C'est d'ailleurs en grande partie parce qu'il a tenu à une compatibilité totale Windows-Mac-Linux que Blender a dû se doter de son interface si particulière – et aussi, il faut bien le dire, si bizarroïde. Ne croyez surtout pas que ce qui paraît simple aux yeux d'un utilisateur novice est simple d'un point de vue technique : généralement, c'est tout le contraire.

8.2.2 L'invite et les paramètres

Principe général

La ligne de commande (on dit aussi le prompt, l'invite, le shell, le terminal, la console...) est le moyen le plus simple et le plus rapide (à défaut d'être le plus élégant) de donner un ordre précis à un ordinateur. Les informaticiens professionnels, qui n'aiment rien tant qu'inspirer une terreur sacrée aux êtres humains normaux, ont réussi à faire croire aux profanes qu'il s'agissait là d'une manipulation ésotérique, mystérieuse, dangereuse, radioactive, réservée à des techniciens ayant suivi sept années d'études supérieures. Ne jouez par leur jeu et ne vous laissez pas gagner par cette terreur superstitieuse. Par ailleurs, sachez qu'il existe aussi des informaticiens sympathiques (mais si), et si ce qui suit vous terrorise, vous pouvez demander à l'un d'entre eux de consacrer un petit quart d'heure à vous l'expliquer avec ses propres mots (montrez-lui ces pages, et soyez-en sûr, il comprendra du premier coup).

La ligne de commande, c'est en réalité bête comme chou (et ennuyeux comme la pluie, mais c'est un autre problème) : il s'agit tout simplement d'indiquer à l'ordinateur le nom d'un programme à exécuter (souvent un tout petit utilitaire, comme par exemple un convertisseur) et de compléter cette demande par quelques précisions (entre zéro et quatre, rarement au-delà) qu'on appelle des **paramètres** (par exemple, très souvent, le nom du fichier qui va subir un traitement). Une ligne de commande type a cet aspect :

```
nom_du_programme parametre1 parametre2 parametre3...
```

Dans le cas de très loin le plus courant, et qui sera presque le seul auquel nous aurons recours ici, le programme appelé sera un convertisseur, dont la fonction est de traduire un fichier informatique d'un mode de codification (celui de Blender,

par exemple) à un autre (celui de POV-Ray, par exemple). La ligne de commande prendra donc cette forme (nom du programme + deux paramètres) :

```
nom_du_convertisseur fichier_lu fichier_ecrit
```

Aucune difficulté jusqu'à ce stade. Poursuivons, ça va un peu se corser.

Syntaxes courantes

Le nom des convertisseurs est arbitraire. Mais comme il doit être tapé au clavier, les informaticiens se simplifient la vie (et compliquent celle des utilisateurs) en choisissant des noms très courts et donc peu intelligibles pour le profane, du genre `bl2pov` (exemple fictif ; il faudrait comprendre « Blender to POV-Ray », en prononçant le chiffre 2 à l'anglo-saxonne pour qu'il puisse signifier « to », c'est-à-dire « vers »). Pour notre part, nous avons donné à nos programmes des noms relativement plus intelligibles (mais évidemment, plus longs), du genre `blender_vers_pov` (l'exemple est toujours fictif). Notre ligne de commande prend donc cet aspect :

```
blender_vers_pov fichier_Blender fichier_POV-Ray
```

Mais nous savons déjà que pour diminuer un peu la pagaille sur le disque dur, il est d'usage d'inscrire à la fin du nom des fichiers une extension qui indique le type de document informatique auquel on a affaire. La ligne de commande n'est toujours pas bien compliquée, mais elle commence à devenir vraiment antipathique :

```
blender_vers_pov bidule.blend bidule.pov
```

Notez que dans la plupart des cas, le fichier lu et le fichier écrit portent le même nom : c'est leur extension qui permet de faire la différence entre les deux.

De plus en plus dur : la syntaxe que nous venons de voir est suffisante en théorie, mais seulement si le programme appelé et les deux fichiers mis en jeu se trouvent tous les trois, en même temps, dans le dossier où l'on a choisi de travailler. C'est rarement le cas : pour ne pas mélanger les torchons et les serviettes, on a coutume de ranger les programmes dans un répertoire spécifique, distinct du répertoire de travail. Dans la tradition Unix que nous respecterons, ce répertoire des programmes s'appelle `bin`. En revanche, il est courant que le fichier lu et le fichier écrit soient bien placés tous les deux dans un même répertoire de travail, et c'est depuis cet emplacement qu'il est le plus simple de taper une ligne de commande pertinente. Il suffira alors d'appeler comme précédemment les deux fichiers par leur seul nom, mais il faudra spécifier un chemin d'accès vers le programme. La ligne de commande aura donc cet aspect :

```
../bin/blender_vers_pov bidule.blend bidule.pov
```

Retenez bien cette syntaxe (nom du convertisseur précédé de `../bin/`, nom du fichier lu, nom du fichier à écrire) : ce sera de loin la plus courante... et vous voyez qu'elle n'est pas si terrible. La seule condition pour que vous puissiez l'em-

ployer est de bien vous placer dans le répertoire pertinent de l'arborescence (nous verrons plus loin comment).

La syntaxe ésotérique `../` avant le nom du répertoire `bin/` des convertisseurs et autres programmes signifie « remonte mentalement d'un cran dans la hiérarchie des dossiers et sous-dossiers avant de chercher à comprendre la suite » (`../` est sans doute moins intelligible que cette injonction circonstanciée... mais c'est rudement plus court à écrire !).

Notez que la syntaxe `../` (avec un slash) devient `..\` (avec un backslash) quand on travaille sous Windows et non sous Unix. Mais il serait insupportable que nous signalions à chaque fois dans les explications de ce livre cette exaspérante variante de syntaxe : si vous travaillez sous Windows, vous devrez opérer vous-même cette substitution sans que nous vous y invitions. Pensez-y !

Syntaxes pour virtuoses

La syntaxe que nous venons de voir est tout à fait efficace, et exige seulement qu'on se soit positionné à l'emplacement correct de l'arborescence avant d'y avoir recours. Certains informaticiens sans doute un peu virtuoses peuvent s'autoriser des syntaxes plus bavardes – et ce sera parfois le cas dans les scripts que nous avons rédigés pour vous faciliter le travail. Vous n'aurez pas en principe besoin de les lire, mais la curiosité (très utile à l'informaticien) peut vous y pousser. Voici donc quelques explications supplémentaires – que vous pouvez sauter si vous pensez avoir déjà assez souffert comme ça !

Lorsque l'on ne s'est pas placé à l'emplacement qui autorise la syntaxe brève que nous venons de voir, il est donc nécessaire d'indiquer où se trouve le programme, où se trouve le fichier au format de Blender, et où il faudra inscrire le fichier nouvellement produit au format de POV-Ray (par goût de la difficulté, nous supposerons ici que les deux fichiers ne sont pas voués à se retrouver dans le même sous-répertoire). C'est à cause de ces précisions que les lignes de commande deviennent épouvantablement longues. Dans un scénario catastrophe, il faudrait écrire quelque chose du genre /usr/local/bin/converters/bl2pov à la place du premier mot de la ligne de commande, /home/user/fichiers_blender/bidule.blend à la place du deuxième, et /home/user/fichiers_pov/bidule.pov à la place du troisième. Nous vous faisons grâce de la récapitulation (la ligne serait d'ailleurs trop large pour être imprimée !), mais cela continuerait de se conformer rigoureusement à la même logique : inscrire trois mots (à rallonge), à savoir le nom (complet) du convertisseur, suivi du nom (complet) du fichier à lire, suivi du nom (complet) du fichier à écrire. C'est laid, c'est long, c'est assez rébarbatif pour que l'utilisateur novice ait besoin de surmonter une répulsion bien naturelle quand il lit cela... mais c'est toujours bête comme chou, et utile, et puissant.

Par égard pour vous, nous l'avons déjà dit, nous avons pris le parti de ranger

tous les fichiers que nous manipulerons dans une série de sous-répertoires tous inclus au même niveau hiérarchique (le premier) à l'intérieur d'un même répertoire fourre-tout (fig. 8.2). Si on travaille au niveau du répertoire fourre-tout (qui englobe tous les sous-dossiers), la ligne de commande aura cet aspect (chacun des trois mots de la ligne de commande étant précédé du nom du sous-dossier où il se trouve) :

```
bin/blender_vers_pov blender/bidule.blend pov/bidule.pov
```

Dans la ligne que nous venons de voir, l'auteur a en outre choisi d'appeler tout simplement `blender` et `pov` les répertoires appelés « fichiers_blender » et « fichiers_pov » dans l'exemple kilométrique précédent. C'était plus clair mais plus bavard, et comme l'auteur aussi est un peu un informaticien, il aime bien comme tous ses congénères économiser les efforts de ses petits doigts sur le clavier... Vous verrez que vous y viendrez vite vous-même.

Attention mesdames et messieurs, nous avons besoin pour l'acrobatie suivante du plus grand silence. Vous y êtes ? Bien. Supposons maintenant qu'au lieu de travailler au niveau du répertoire fourre-tout, nous ayons choisi de nous placer dans le sous-répertoire « blender ». La ligne de commande pourrait alors prendre cet aspect mystérieux et terrible :

```
../bin/bl2pov bidule.blend ../pov/bidule.pov
```

D'une façon très similaire, si nous avions choisi de nous placer cette fois dans le sous-répertoire « pov », la ligne de commande deviendrait :

```
../bin/bl2pov ../blender/bidule.blend bidule.pov
```

Dans les deux cas, on a omis d'indiquer le chemin d'accès à l'un des fichiers car ce n'était pas utile : l'emplacement de ce fichier est celui où l'on a déjà choisi de se placer.

Une petite récréation pour nous détendre de toute cette magie noire. Y aura-t-il dans la salle un élève assez astucieux pour comprendre que la ligne de commande ci-dessous aurait pu être employée dans les deux derniers cas ?

```
../bin/bl2pov ../blender/bidule.blend ../pov/bidule.pov
```

(Réponse : ça aurait marché dans les deux cas, parce que dans les deux cas cela aurait conduit à remonter dans l'arborescence pour redescendre aussitôt à l'endroit où on se trouvait déjà... ce qui était bien sûr inutile, mais pas nuisible.)

Allez-y, servez-vous une aspirine, vous l'avez bien méritée. Mais de grâce, comprenez que ce que nous venons de dire n'est pas hors de portée de vos capacités intellectuelles, mais seulement ennuyeux. C'est seulement un bazar un peu technique auquel il faut s'habituer, au besoin en se faisant un peu aider au début par le premier informaticien qui passe. Ce n'est pas si difficile que ça, mais c'est vraiment beaucoup, beaucoup plus utile et puissant que vous ne le pensez.

8.3 L'éditeur de lignes

8.3.1 Séries de commandes (scripts et batches)

La toute première raison pour laquelle ces manipulations sont utiles et puissantes, c'est que leur automatisation est à la portée du premier crétin venu : il suffit d'enregistrer une bonne fois pour toutes dans un fichier de texte une suite de ces lignes si minutieuses à rédiger pour pouvoir lancer en une seule commande et même avec un seul mot une quantité indéfinie de telles opérations (quatre, cinq, plusieurs dizaines, plusieurs milliers même : cela n'a aucune limite). On appelle ces récapitulations de commandes à effectuer des **scripts** (plus précisément, sous Unix, des « scripts shell », auxquels on donne généralement l'extension .sh) ou, dans la tradition Windows, des **batches** (auxquels on donne obligatoirement l'extension .bat).

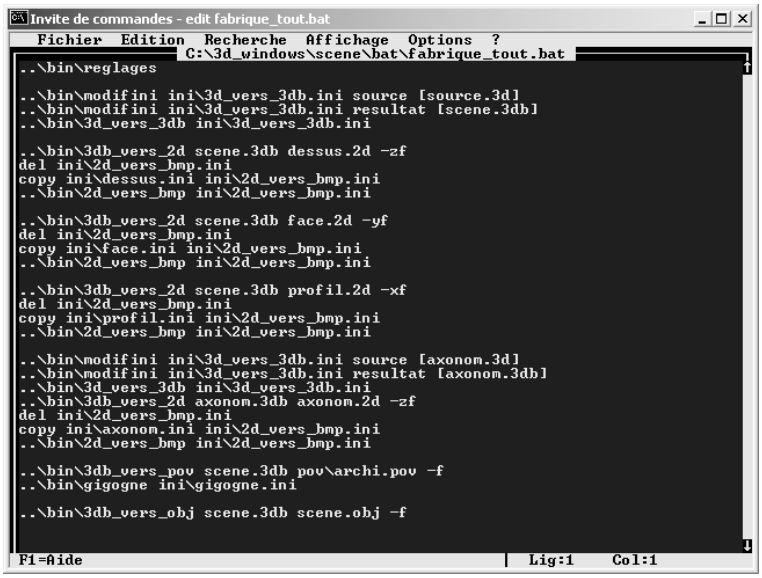


FIG. 8.4 – *Peut-être le batch le plus bavard que nous utiliserons. Rassurez-vous : vous n'aurez qu'à connaître son nom, pas à le rédiger vous-même... Sans chercher à en comprendre le détail, notez qu'il comporte de nombreuses lignes commençant par* `..\bin\`, *ce qui permet le lancement de petits programmes situés dans un autre répertoire que celui où l'on travaille, mais au même niveau dans l'arborescence.*

Les informaticiens savent en outre qu'il n'est pas très malin d'intercaler quelques instructions au milieu de ces lignes pour qu'elles adaptent leur comportement en fonction de certaines conditions (hélas, ça ne se fait pas de la même façon sous

Windows qu'ailleurs). Nous ne vous en demanderons pas tant : pour les besoins de l'automatisation de la 3D, nous nous limiterons à des scripts bêtes et méchants (généralement de quatre ou cinq lignes au maximum), qui foncent droit devant eux sans jamais se poser de question.

Si les scripts agissent de façon totalement idiote et mécanique, celui qui les rédige a au contraire besoin de se donner le temps de la réflexion : un script mal écrit (ou d'ailleurs lancé dans un contexte qui ne lui est pas adapté) peut mettre pas mal de pagaille dans votre disque dur. Cette réflexion nécessaire a lieu lors de la rédaction du script, alors qu'on manipule un logiciel spécialisé dans l'élaboration du code informatique, et que l'on appelle un **éditeur de lignes** (nous en avons déjà parlé au début de ce livre).

Toutes les opérations d'automatisation de la 3D peuvent être élaborées avec un éditeur de lignes, et elles ne sont jamais aussi simples à mettre en place qu'avec lui. L'éditeur de lignes est à la 3D informatisée ce que le tube de colle est à la maquette en carton : un outil qui ne paie pas de mine, mais avec lequel on accomplit l'essentiel du travail.

8.3.2 Les fichiers d'initialisation

Nous avons vu qu'une ligne de commande pouvait comporter un nombre indéfini de paramètres, et certaines opérations ont effectivement besoin d'un nombre de précisions assez abondant (notamment en 3D, où le moindre point est déjà défini par trois coordonnées). Dans ce cas, plutôt que d'écrire des lignes de commande à rallonge pourvues d'une douzaine de paramètres (ce qui ne pose pas de difficulté technique, mais est bien sûr illisible), nous utiliserons la technique merveilleusement simple et efficace du **fichier d'initialisation** (dont l'extension sera classiquement .ini). Il s'agit là encore de petits textes que l'on manipule avec un éditeur de lignes, et qui permettent des réglages très fins. La syntaxe de ces fichiers d'initialisation est tellement simple qu'elle se passe de commentaires :

```
Nom_du_parametre = Valeur_du_parametre
```

Vous devrez donc parfois ouvrir de tels fichiers d'initialisation avec un éditeur de lignes, y modifier un ou deux réglages, et enregistrer le tout avant de lancer une opération complexe. Avec la pratique, on se rend vite compte que cette façon de faire est en réalité beaucoup plus rapide et simple à manipuler qu'une interface graphique truffée de boutons, de cases à cocher, d'onglets et de sous-options.

Les fichiers d'initialisation n'ont d'ailleurs pas toujours besoin d'être modifiés manuellement par un être humain faisant appel à son intelligence : leur extrême simplicité leur permet d'être manipulés même par des automatismes. Notez bien l'astuce : ces fichiers permettent de régler l'action de certains programmes... mais ils peuvent eux-mêmes être modifiés par d'autres programmes. Le fichier d'initialisation d'un programme très complexe (par exemple, POV-Ray) peut être modifié

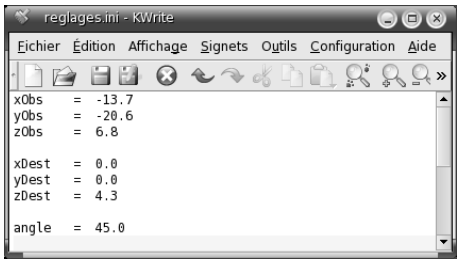


FIG. 8.5 – *Un fichier d'initialisation, très simple à comprendre comme à modifier (il concerne la mise en place d'une caméra).*

par un automatisme de quatre sous mis en place par un amateur ; variante de la même idée : vous pourrez écrire vous-même de tout petits scripts pour modifier automatiquement les fichiers d'initialisation mis au point par l'auteur. Sans vouloir brûler les étapes, sachez déjà que c'est là une des astuces qui permettent de provoquer la fabrication automatique des milliers d'images d'une animation. Le mariage de l'éditeur de lignes et des scripts les plus élémentaires donne accès à une incroyable puissance, vous vous en rendrez vite compte.

8.4 Le travail en ligne de commande

8.4.1 La logique d'une vieille imprimante

Le moyen le plus simple de prendre conscience de l'intérêt qu'il y a à travailler en ligne de commande est évidemment de s'en servir pour obtenir tout de suite des résultats. C'est donc ce que nous allons faire maintenant. Ouvrez une fenêtre permettant de travailler ainsi. On l'appelle « Invite de commandes » dans les diverses versions de Windows (fig. 8.3), « Terminal » sous Mac OS/X, et de noms très divers (xterm, rxvt, konsole...) sous Linux. Pour notre part, afin d'unifier un peu la terminologie, nous parlerons désormais toujours de **console**. Les consoles existent sous d'innombrables variantes, des plus rudimentaires comme celle de Windows (d'aucuns la qualifieraient même d'indigente, non sans raison) aux plus sophistiquées et ingénieuses comme le bash cher aux linuxiens... mais nous en ferons un usage tellement élémentaire que nous pourrons presque les considérer comme interchangeables. Hélas, pour les commandes les plus courantes, Windows et Unix ont souvent choisi des noms différents, et il va bien falloir quand même mentionner quelques variantes.

Le terminal de Mac OS/X
La console de Mac OS/X se nomme Terminal. Ce programme est normalement rangé dans le sous-dossier Utilitaires du dossier Applications de Mac OS/X.
Le Terminal a un aspect un peu différent de celui d'une console traditionnelle comme l'invite de commandes de Windows : le texte y figure en noir sur fond blanc et non le contraire, il n'y a pas de curseur clignotant (on s'en passe : il suffit de savoir que les commandes tapées au clavier apparaissent systématiquement sur la dernière ligne de l'écran). Enfin, la configuration par défaut de l'invite de Mac OS/X n'est pas très heureuse, puisqu'elle affiche très inutilement le nom complet de l'ordinateur sur lequel on travaille, souvent très long. A ces quelques détails près, le Terminal est une console on ne peut plus banale, et elle fonctionnera exactement comme nous le décrivons dans ces pages.

L'esprit général, fort heureusement, est toujours le même. La console joue assez exactement le rôle des imprimantes des tout premiers ordinateurs (à l'époque où ces derniers n'avaient même pas d'écran dans bien des cas), celles qui consommaient des longueurs kilométriques d'un papier perforé sur les côtés et plié en accordéon. Pour la plus grande joie des écologistes, les consoles ont permis de mettre fin à cet effroyable gaspillage de cellulose... mais elles continuent de fonctionner comme ces vieilles imprimantes, c'est-à-dire qu'elles affichent des lignes et ne les effacent pas – même si ces indications ont cessé d'être pertinentes, et il faut tout de suite en prendre conscience !

Par exemple, une console peut signaler sur la ligne 1 l'existence d'un problème ; l'informaticien tape alors sur la ligne 2, en réponse à l'invite, la manipulation permettant de le résoudre. Enfin, la console répond sur la ligne 3 en disant qu'en effet le problème est maintenant résolu. Cela nous donne donc très logiquement un affichage de ce type :

```
1 Problème constaté, traitement nécessaire...
2 invite> lancer solutionneur_de_problemes
3 Problème résolu. Tout va bien.
```

Notez bien : le problème est maintenant résolu, mais la ligne qui signale le problème n'a pas été effacée (la console se comporte comme une imprimante, et une imprimante n'efface rien), et cela peut vous donner l'impression fausse que le problème subsiste si vous ne prenez pas tout de suite l'habitude, comme l'informaticien moyen, de lire les comptes rendus informatiques... plutôt en commençant par le bas : ce qui est en bas est le plus récent, donc très souvent le plus pertinent ; ce qui est en haut est ancien, et parfois caduc. Prenez-y bien garde, ou vous ne comprendrez rien à ce qui se passe.

Cela dit, les consoles ne sont pas de vraies imprimantes, elles ne comportent qu'un nombre de lignes très limité (vingt à quarante, en général). Elles ne peuvent donc pas afficher des comptes rendus kilométriques... et au bout d'un temps par-

fois très bref, les messages récents entrant par le bas de la console font sortir et disparaître par le haut de la console les messages plus anciens. Attention : c'est le nombre de lignes affichées qui provoque cette élimination, et non la vétusté des messages. Une console peu utilisée peut parfaitement afficher un message obsolète depuis plusieurs heures, et une série de messages très abondante (plus de quarante lignes) défile parfois si vite qu'on n'a pas le temps de lire certaines lignes importantes. Or, ce n'est pas parce qu'un message a été effacé qu'il cesse forcément d'être pertinent. Il faut y prendre garde, mais ne vous affolez pas : les convertisseurs que nous utilisons sont peu bavards, et en principe, les messages importants resteront visibles tant que vous en aurez besoin.

Voilà à peu près tout ce qu'il faut savoir de la théorie des consoles. Il s'agit d'une technologie très rustique, certes, mais aussi très facile à mettre en œuvre, et on a bien tort de se passer des immenses services qu'elle peut rendre sous prétexte qu'elle n'est pas aussi jolie que des fenêtres de dialogue.

Encore un mot juste avant de passer aux exercices. Pratiquement dans tous les cas, la console a la forme d'un rectangle unicolore (généralement noir), avec un curseur clignotant qui indique l'endroit où apparaîtra la ligne de commande que vous allez taper au clavier. Ce curseur clignotant est précédé à sa gauche d'une mention qui varie selon la machine, le système d'exploitation... et le contexte. C'est en fait à l'apparition de cette mention qu'on se rend compte qu'on est en présence d'une invite (ou prompt). Dans nos explications, nous noterons systématiquement l'invite sous cette forme :

```
invite>
```

mais sur votre propre console, elle pourra avoir les aspects les plus divers, du genre `C :\3d_win\scene>` sous Windows, `[jeandupont@localhost] $` sous Linux, `[ordinateurdejean] $` sous Mac OS/X, sans parler des variantes où l'invite indique l'heure ou la date. Nous ne parlerons pas de ces variantes, nous noterons simplement `invite>`. Jetez-y quand même un œil : il n'est pas rare qu'elle ait été configurée pour fournir une indication sur le répertoire où on se trouve, et quand c'est le cas, c'est bien pratique.

8.4.2 Savoir où l'on est

En effet, la toute première chose à faire lorsque l'on se met ou remet à travailler avec une console, c'est de déterminer à quel endroit de l'arborescence on se trouve. Sous Windows, il suffit de taper `cd` (ce qui signifie en fait « choose directory », mais s'emploie aussi – sous Windows exclusivement ! – pour déterminer où l'on est) ; sous les Unix comme Mac OS/X ou Linux, ce n'est pas plus compliqué, mais il faut taper `pwd` (pour « print work directory »).

N'oubliez pas la touche Entrée !
Il ne suffit pas de taper ces deux ou trois lettres au clavier, il faut encore appuyer sur la grande touche « Entrée » (Enter) ou « Retour » (Return) à la droite du clavier alphabétique : bien entendu, chaque ligne de commande doit *toujours*, absolument toujours, être validée par l'appui de cette touche – il faut vraiment n'avoir jamais utilisé une console de sa vie pour l'ignorer ! C'est tellement systématique que nous ne prendrons plus *jamais* la peine de le préciser. Ne l'oubliez pas, ou vous attendrez des heures devant votre écran noir qu'il se passe enfin quelque chose !

En réponse à la commande `cd` sous Windows et `pwd` sous Unix, la console affichera une ligne souvent riche en backslashes dans le premier cas (du genre `c :\` ou `c :\3d_win\scene\`), et riche en slashes dans le deuxième (du genre `/home/jeandupont/` sous Linux et `/Users/jeandupont/` sous Mac OS/X) : à cette maudite différence de syntaxe près, toutes nos consoles peuvent rendre le même service.

8.4.3 Se placer dans un répertoire pertinent

A moins que votre machine soit vraiment configurée de façon très bizarre, une console nouvellement ouverte ne vous place pas n'importe où, mais au contraire à la position stratégique où nous vous avons recommandé d'installer le répertoire contenant les convertisseurs. Sous Windows, vous devriez ainsi systématiquement arriver en `c :\`, c'est-à-dire tout en haut de l'arborescence du disque dur principal c : – au même endroit que si vous aviez double-cliqué sur l'icône du disque dur c : dans la fenêtre représentant le contenu du Poste de travail. Sous Linux et Mac OS/X, l'ouverture d'une console vous mène normalement dans un répertoire portant le nom sous lequel vous vous connectez lorsque vous allumez l'ordinateur ; si ce nom est `jeandupont`, vous arriverez donc en `/home/jeandupont` sous Linux, et en `/Users/jeandupont` sous Mac OS/X (c'est-à-dire dans le répertoire appelé également jeandupont par le Finder du Mac ; hélas, le Finder d'un Mac OS/X francisé vous embrouillera un peu en plaçant ce répertoire à l'intérieur d'un dossier nommé Utilisateurs et non Users ; c'est sans conséquence, mais cela peut à bon droit vous surprendre).

Mais il se peut aussi que vous vous soyez déjà déplacé dans l'arborescence, et que vous n'ayez pas envie d'ouvrir une nouvelle console simplement pour vous retrouver à cet emplacement stratégique. Pas de problème. Sous Windows, tapez simplement la commande `cd \` (dans certains cas très particuliers, ça ne suffirait pas et il faudrait taper deux commandes : `c :` puis `cd \` ; cette manipulation en deux temps ne peut jamais donner un autre résultat que de vous placer en haut de l'arborescence du disque dur c :). Sous Linux et Mac OS/X, c'est encore plus court : l'ordre à taper est tout simplement `cd` (ne vous mélangez pas avec la syntaxe de Windows, où cette seule commande aurait un autre effet, que nous

avons déjà évoqué).

Bien. Maintenant que nous savons où nous sommes, nous pouvons nous déplacer dans l'arborescence. Incroyable mais vrai : c'est rigoureusement la même syntaxe sous Windows et sous Unix ! Pour entrer dans un sous-répertoire, la syntaxe est tout bêtement

```
cd nom_du_sous-repertoire
```

et pour en ressortir, tout bêtement

```
cd ..
```

car la syntaxe `..` que nous avons déjà évoquée signifie « étage supérieur » de l'arborescence.

L'informaticien descend-il vraiment du singe ?
Une convention aussi absurde qu'ancienne veut que le niveau le plus *élevé* dans la hiérarchie des dossiers et sous-dossiers (qu'on appelle aussi l'arborescence) soit appelé... la **racine**. Il faut bien comprendre que cette prétendue racine est censée se trouver *en haut* de l'arborescence... et non au pied d'un arbre. Par conséquent, les expressions « remonter dans l'arborescence » et « se rapprocher de la racine »... sont rigoureusement synonymes ; ce qui est sans doute plus intelligible pour les taupes et les vers de terre que pour les descendants de Lucy !
Cela dit, nous emploierons nous aussi cette terminologie incohérente... parce que toute la planète informatique en fait autant.

Vous en savez désormais assez pour vous placer au bon niveau hiérarchique dans un répertoire de travail. Ce répertoire, nous allons vous demander de le créer dans le dossier `3d_win`, `3d_linux` ou `3d_mac` (selon votre système d'exploitation) sur votre disque dur. Pour ce faire, il faut simplement que vous recopiiez là un dossier-type, `08_scene_type`, que vous trouverez dans le répertoire `sgml_fichiers` de votre disque dur. Attention : si votre système d'exploitation laisse l'attribut « Lecture seule » aux dossiers et fichiers qu'il recopie d'un CD-ROM, il vous faudra modifier manuellement ce réglage (nous avons déjà traité de cette question au chapitre 2, à propos de l'installation des convertisseurs sous Windows et sous Linux). Vous devrez d'ailleurs accomplir la même opération chaque fois que vous voudrez travailler avec un dossier recopié depuis `sgml_fichiers`. Notez que vous n'êtes pas du tout forcé de conserver ce nom de `08_scene_type` ; au contraire, chaque fois que vous recopierez ce répertoire (ou d'ailleurs l'un des autres dossiers de démonstration de `sgml_fichiers`) pour mettre en chantier une nouvelle scène, vous aurez bien sûr intérêt à lui donner immédiatement un intitulé correspondant à ce que vous voulez faire – quand ce ne serait que pour ne pas risquer que son contenu soit écrasé la prochaine fois que vous effectuerez une telle copie !

Quel que soit le nom que vous aurez choisi pour la copie du dossier `08_scene_type`, il vous faut à présent vous placer dans ce sous-répertoire à l'aide de la commande `cd` avant de passer à l'exercice décrit dans les paragraphes suivants.

8.4.4 Un dossier-type pour travailler en SGML

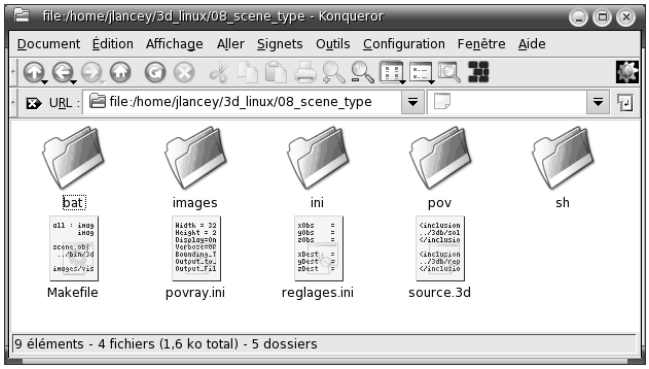


FIG. 8.6 – *Le contenu de départ du dossier permettant la représentation d'une scène (affiché ici sous KDE, l'une des interfaces graphiques de Linux).*

Prenons le temps d'examiner un peu le contenu de ce dossier (fig. 8.6). Comme vous pouvez le constater, le dossier `08_scene_type` englobe ses cinq propres sous-répertoires. Cela enfreint quelque peu la règle que nous nous étions fixée précédemment (disposer tous les sous-répertoires au premier niveau hiérarchique à l'intérieur de 3d_win, 3d_mac ou 3d_linux), mais nous n'entrerons guère à l'intérieur de ces sous-répertoires, qui servent en quelque sorte pour des opérations de cuisine interne. Nous travaillerons toujours au même niveau hiérarchique, celui de `08_scene_type`, et nous ne nous intéresserons guère au départ qu'à deux des fichiers stockés à ce niveau : `source.3d` et `reglages.ini` : c'est à partir de ces deux seuls fichiers (tous deux voués à être modifiés manuellement à l'aide d'un éditeur de lignes) que nous déterminerons l'aspect d'une scène 3D et de l'image de synthèse qui la représente à un moment donné.

Comment connaître le contenu d'un dossier ?
Il est bien sûr possible d'obtenir des informations sur le contenu d'un dossier en faisant apparaître avec les fonctions standard de l'interface graphique une fenêtre du type de celle de la fig. 8.6, mais il est généralement plus rapide d'obtenir ces informations directement depuis la ligne de commande. Pour cela, on tape sous Windows la commande `dir` et sous Mac OS/X ou Linux la commande `ls` (ou plutôt `ls -l`, qui affiche, outre le nom des fichiers, leur taille en octets). Les indications ainsi obtenues sont très généralement suffisantes, mais si le nombre de fichiers contenus dans le répertoire est trop abondant, il peut être nécessaire d'avoir recours à des commandes un peu plus complexes : `dir /p` sous Windows (notez que c'est bien `/p`, avec un slash normal, et non `\p`), et `ls -l | more` sous Linux ou Mac OS/X.

Le fichier `source.3d` est bien sûr un fichier SGML 3D. Il est au départ assez court pour que nous en citions intégralement le contenu :

```
<inclusion>
../3db/sol.3db
</inclusion>

<inclusion>
../3db/reperes.3db
</inclusion>

<inclusion>
../3db/chalet.3db
</inclusion>
```

Comme vous l'aurez sans doute compris à demi-mot, ces instructions se contentent d'inclure dans la scène, sans les modifier ni les déplacer, trois objets stockés dans le dossier `3db`. Ce dernier est situé au même niveau hiérarchique que `08_scene_type`, ce qui permet d'accéder à son contenu par la syntaxe laconique `../3db/` que l'on trouve en début de ligne : c'est avant tout pour aboutir à cette facilité d'écriture que nous avons pris le parti de placer presque tous les dossiers au même niveau hiérarchique dans l'arborescence. Le dossier `3db` contient une petite cinquantaine d'objets 3D parfois très simples, parfois fort complexes, mais tous relativement normalisés : sauf exception, ils sont tous stockés à la même échelle (l'unité utilisée pour définir leurs coordonnées est le mètre) et placés à proximité immédiate du point O (intersection des axes X, Y et Z). Au cours de votre apprentissage, vous pourrez inclure à votre guise dans vos scènes un ou plusieurs de ces objets. A terme, bien sûr, vous travaillerez avec des objets que vous aurez vous-même définis, mais vous pourrez parfaitement les ranger dans ce même répertoire `3db`.

En l'occurrence, les trois objets inclus dans la scène sont : premièrement, le « sol », c'est-à-dire une grande surface carrée (elle fait un kilomètre de large) qui sert essentiellement à recevoir les ombres portées et à créer un pseudo-horizon pour donner aux scènes un aspect intelligible ; les « repères », c'est-à-dire un quadrillage d'une dizaine de mètres de large (la distance entre deux lignes est d'un mètre) pourvu de deux axes fléchés X et Y (c'est fort utile tant que la scène n'est pas totalement au point pour comprendre l'échelle de l'image de synthèse et l'orientation de la caméra qui la produit) ; enfin, troisièmement, le « chalet », c'est-à-dire une petite maison déjà assez riche en détails.

Les deux autres fichiers présents dès le départ dans le dossier `08_scene_type` sont un `Makefile` (dont nous toucherons un mot un peu plus loin ; ce fichier d'usage très courant sous Unix peut rendre de grands services sous Linux et Mac OS/X, mais n'est hélas d'aucune utilité sous Windows) et le fichier d'initialisation de POV-Ray, `povray.ini`. Jusqu'à il y a fort peu, POV-Ray n'avait recours à ce

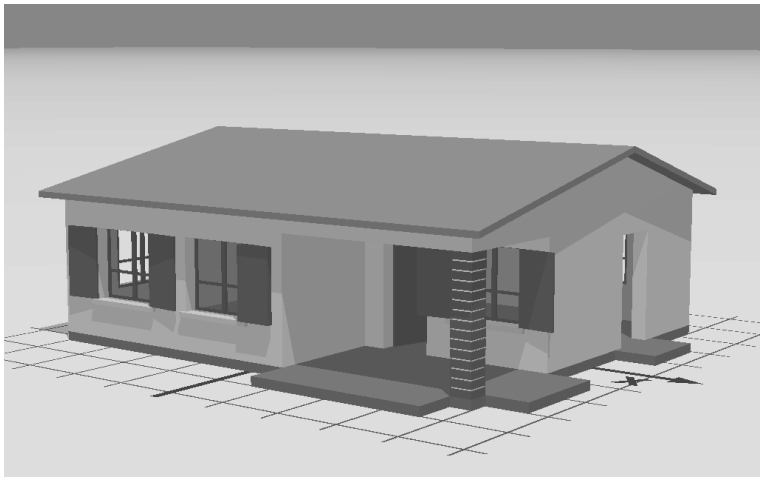


FIG. 8.7 – *Les trois objets de la scène-type : le sol, les repères et bien sûr le chalet. Le sol sert surtout à recevoir les ombres portées et à faire figurer une ligne d'horizon. Les repères sont à l'échelle métrique, et l'orientation des axes (notez la pointe de l'axe des X) permet de comprendre la position et l'orientation de la caméra.*

fichier que dans sa version pour Linux, mais sa toute dernière version pour Windows le prend elle aussi en compte, notamment pour régler la taille des images de synthèse ; en revanche, la version Macintosh de POV-Ray ignore superbement ce fichier d'initialisation, ce qui rend l'automatisation de POV-Ray sous Mac OS/X un peu plus difficile que sur les autres plates-formes. Nous y reviendrons.

Vous n'avez pas a priori à vous soucier des cinq sous-répertoires de `08_scene_type`, mais on peut tout de même toucher un mot à leur sujet. Les répertoires `bat` et `sh` jouent exactement le même rôle (stocker des scripts d'automatisation), mais le premier est conçu pour les batches de Windows et le second pour les scripts shell de Linux ou Mac OS/X. Le répertoire `ini` contient quelques fichiers d'initialisation utilisés par ces scripts (vous pouvez les lire avec un éditeur de lignes si la curiosité vous y pousse), et le répertoire `pov` contient des morceaux de code informatique de POV-Ray qui seront assemblés automatiquement (au besoin après avoir été mis à jour) pour écrire un descriptif de scène complet conformément à la syntaxe de ce programme. Enfin, le répertoire `images` est voué à stocker les diverses représentations de la scène 3D manipulée, mais c'est seulement avec POV-Ray pour Linux que les images de synthèse seront produites là ; les réglages par défaut de POV-Ray pour Windows et pour Mac auront pour effet de placer les images de synthèse directement dans `08_scene_type`, et nous nous en contenterons. En revanche, c'est bien dans le sous-répertoire `images` que les scripts

produiront des vues simplifiées de la scène en « fil de fer », au format BMP à seize couleurs ; nous verrons cela plus loin.

8.4.5 Modifier un fichier d'initialisation

Comme son nom l'indique, le dossier `08_scene_type` est voué à servir de squelette pour la production d'images de synthèse quelle que soit la configuration propre à votre machine. Toutefois, et en dépit de tous les efforts de standardisation qui ont été le souci constant de cet ouvrage, il vous faut encore opérer un tout petit réglage avant de commencer à travailler – en particulier avant de pouvoir lancer les scripts d'automatisation contenus dans les sous-répertoires `bat` et `sh`. Ouvrez donc avec un éditeur de lignes le fichier `reglages.ini` (comme nous l'avons déjà vu sur la fig. 8.5). Il est ainsi rédigé :

```
xObs = 15.0
yObs = -20.0
zObs = 5.0

xDest = 0.0
yDest = 0.0
zDest = 1.65

angle = 28.0

largeur = 12.0
unite = 2.0

RotZAxonom = -45.0
RotXAxonom = -45.0

systExpl = linux
permisDEcraser = TRUE
```

Pour le moment, seul l'avant-dernier paramètre (`systExpl`) nous intéresse, mais signalons quand même au passage (nous y reviendrons) que les sept premiers paramètres (`xObs, yObs, zObs, xDest, yDest, zDest` et `angle`) définissent la position, l'orientation et la focale de la caméra qui a été employée pour le calcul de l'image de synthèse de la fig. 8.7.

Regardez si la valeur de l'avant-dernier paramètre `systExpl` correspond bien à votre système d'exploitation et, si ce n'est pas le cas, modifiez le fichier pour inscrire la mention pertinente (qui sera `windows`, `macintosh` ou `linux` selon le cas). Enregistrez le fichier avant de le refermer. C'est tout, mais c'est indispensable pour passer à la suite.

8.4.6 Lancer des scripts

A présent que nous nous sommes placés dans le bon répertoire et avons correctement configuré le paramètre `systExpl` du fichier `reglages.ini`, il ne nous reste qu'à mettre en branle le processus automatisé qui va transformer tous ces vilains petits fichiers de texte en véritables images de synthèse. Il va suffire pour cela d'écrire une seule ligne de commande, mais selon une syntaxe légèrement différente sous Windows et sous les « Unix-like » comme Linux et Mac OS/X. Sous Windows, il va vous falloir taper

```
bat\fabrique_tout
```

tandis que sous Linux et Mac OS/X, ce sera

```
sh sh/fabrique_tout.sh
```

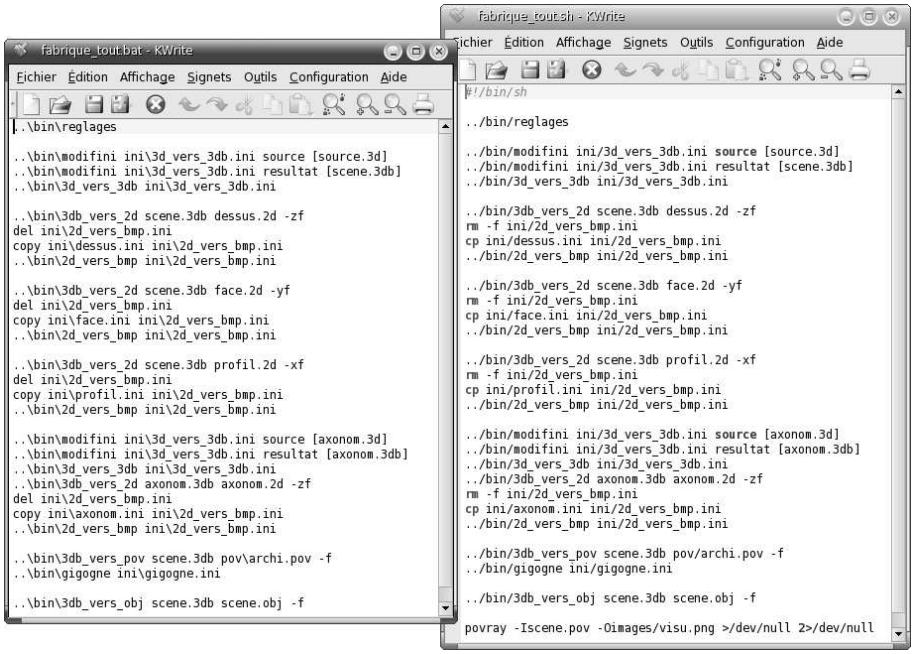


FIG. 8.8 – *Les batches du sous-dossier bat (à gauche) et les scripts shell du sous-dossier sh (à droite) jouent le même rôle, et ne diffèrent que par des différences de syntaxe minimes, liées aux spécificités des systèmes d'exploitation.*

L'une et l'autre syntaxe servent en fait à provoquer l'exécution d'un des deux petits scripts de la fig. 8.8, et donc à obtenir le même résultat que si vous aviez tapé manuellement (et sans erreur !) sur la ligne de commande les quelque vingt-cinq lignes qu'ils contiennent.

Bienvenue dans le monde de l'automatisation ! L'accomplissement de ces vingt-cinq commandes prend bien sûr un certain temps, comme disait le regretté Fer-

nand Raynaud. Une fois que vous aurez validé la ligne de commande qui convient avec la touche « Entrée » ou « Retour », il vous faudra attendre jusqu'à quelques dizaines de secondes pour que le système d'exploitation vous « rende la main » – c'est-à-dire, en patois informatique, pour qu'il affiche à nouveau une invite, vous signalant par là-même qu'il a terminé son travail et est donc de nouveau en attente d'une commande. Pendant ce temps (que vous pouvez parfaitement consacrer à surfer sur Internet si vous le jugez trop long !), sous Mac OS/X ou Linux, vous aurez probablement l'impression qu'il ne se passe rien du tout. Sous Unix, en effet, l'automatisation est dans l'ordre des choses, et l'on trouve tout à fait normal qu'un ordinateur travaille sagement dans son coin sans se manifester inutilement.

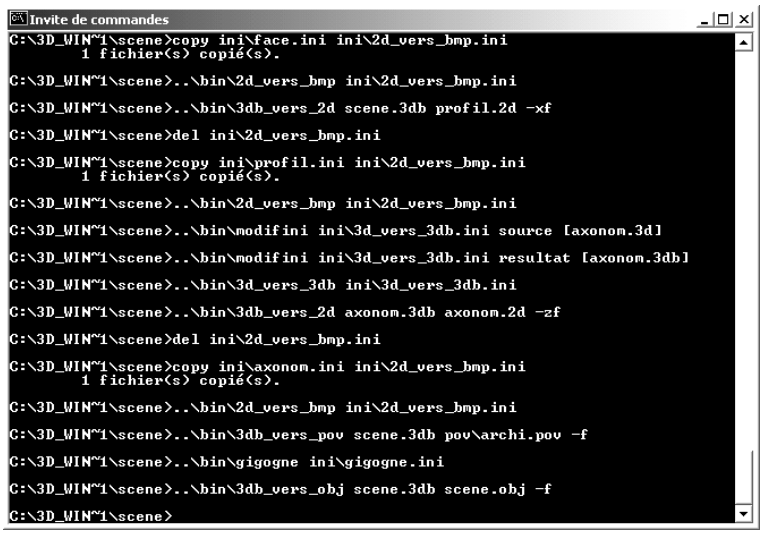


FIG. 8.9 – *La fenêtre de l'invite de commandes de Windows après exécution du batch fabrique_tout.bat.*

Sous Windows, en revanche, les opérations sont normalement toujours ponctuées de clics de souris et de fenêtres de dialogue, et la fenêtre de l'invite de commandes va elle aussi se rappeler à votre bon souvenir en faisant défiler un nombre impressionnant de lignes (fig. 8.9). Le style est différent, mais l'idée est la même : la machine travaille seule, fait tout le nécessaire, et vous pouvez vous en remettre à elle jusqu'à ce qu'elle ait terminé de fabriquer ce dont vous avez besoin.

Justement : que fait-elle durant ce « certain temps » ? Eh bien, essentiellement, elle lit le contenu de `source.3d` et `reglages.ini`, et elle en tient compte pour fabriquer des fichiers temporaires qui servent à en fabriquer d'autres, qui sont eux-mêmes exploités, et ainsi de suite... jusqu'à faire apparaître dans le répertoire `08_scene_type` les deux fichiers vers la fabrication desquels tout ce joli travail menait : `scene.obj` et `scene.pov` (fig. 8.10). Le premier, enregistré au format

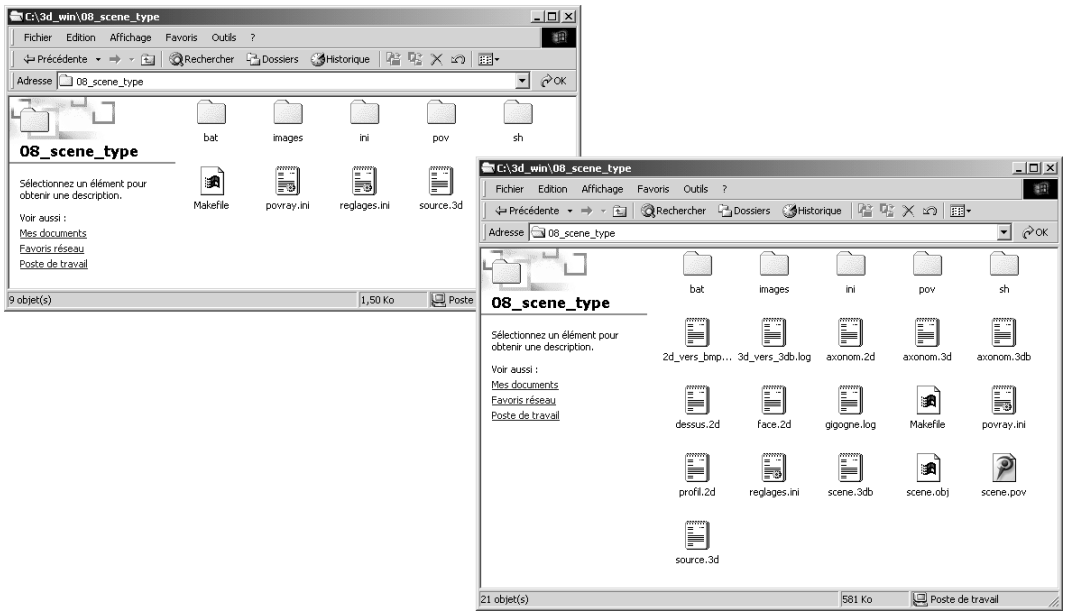


FIG. 8.10 – *Le contenu du dossier 08_scene_type, avant et après le lancement du script fabrique_tout. Parmi les fichiers produits, scene.obj et scene.pov nous intéressent particulièrement.*

OBJ de Videoscape, peut être exploité par Blender ; le second est au format de POV-Ray et voué à être lu par ce dernier programme pour produire une image de synthèse parfaitement paramétrée.

C'est le principal, mais ce n'est pas tout. Le script (ou le batch) `fabrique_tout` a également produit quatre images de contrôle (vue de dessus, de face, de profil et axonométrie) dans le sous-répertoire `images` (au format BMP à seize couleurs ; vous pouvez les lire avec un logiciel de retouche d'images quelconque). Ce n'était pas indispensable, mais cela vous donnera une idée de tout le travail qu'un simple script d'une trentaine de lignes peut accomplir.

Enfin, sous Linux exclusivement (Linux est *le* système d'exploitation des amoureux de l'automatisation totale), le script a poussé le processus jusqu'au bout et directement produit une image de synthèse (`visu.png`, que les linuxiens trouveront dans le sous-répertoire `images`).

On le voit, cette technique d'automatisation est aussi efficace qu'élémentaire. Cela étant, si elle est tout à fait acceptable pour des amateurs, il en existe de bien plus performantes. Sous Unix, en particulier, le script `fabrique_tout` peut très avantageusement être remplacé par le `Makefile` qui se trouve dans le même répertoire `08_scene_type`. Pour lancer ce Makefile, il vous suffira (sous Linux ou Mac OS/X seulement : noblesse oblige) de taper la commande

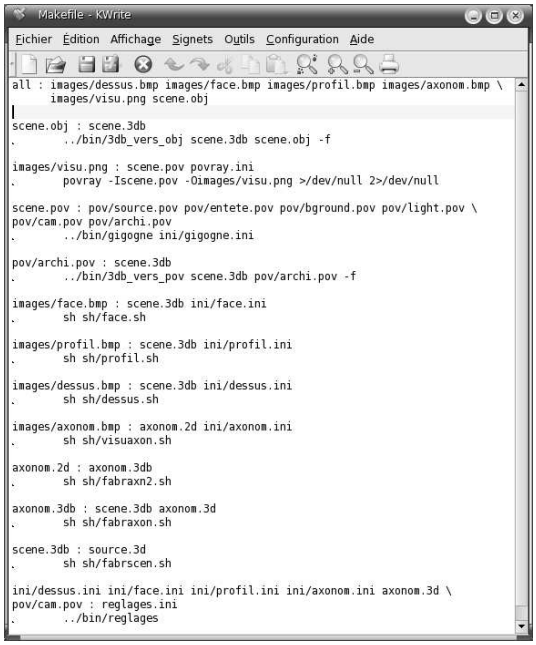


FIG. 8.11 – *Un Makefile d'Unix est à peine plus délicat à écrire qu'un script ou un batch, mais son comportement est bien plus subtil.*

`make`

Non seulement cette commande est capable de provoquer les mêmes vingt-cinq traitements que le script `fabrique_tout`... mais elle est aussi capable d'en négliger certains quand ils ne sont pas nécessaires, économisant ainsi un temps précieux. Cet ouvrage n'est pas assez épais pour expliquer la syntaxe pourtant relativement simple du Makefile (fig. 8.11). Ceux que le sujet intéresse trouveront toutes les explications utiles dans un ouvrage sur la programmation Unix – mais on peut parfaitement utiliser ce Makefile sans comprendre le moins du monde sa logique !

Pour que vous puissiez vous entraîner, nous avons inclus dans les sous-répertoires `bat` et `sh` des scripts servant à éliminer les fichiers que vous venez tout juste de produire (comme disait ma grand-mère : faire et défaire, c'est toujours travailler). Ces deux scripts peuvent être lancés de la même façon que `fabrique_tout` : sous Windows, il vous faut taper `bat\purge` et sous Linux ou Mac OS/X `sh sh/purge.sh`. Après quoi vous pourrez recommencer à tester l'action de `fabrique_tout` ou du `Makefile`. Les linuxiens et applemaniaques remarqueront que le fait de taper `make` une deuxième fois sans avoir procédé à la purge ne déclenchera aucun traitement. Ce n'est pas du tout une erreur : un `Makefile` sert justement à éviter de recommencer inutilement des opérations déjà accomplies.

8.5 Du clavier à l'image

8.5.1 Utiliser Blender comme visualiseur

Une fois que le script `fabrique_tout` a fait son travail, vous n'avez peut-être pas l'impression d'être beaucoup plus avancé qu'au départ, car les fichiers `scene.obj` et `scene.pov` ne sont que des fichiers de texte. Mais nos deux programmes préférés ne vont plus avoir aucune peine à transformer ces suites d'instructions informatiques en images. Commençons par nous en assurer avec Blender, que nous connaissons bien désormais.

Le fichier `scene.obj` n'est pas au format de Blender, mais de Videoscape. Blender peut l'exploiter, mais seulement en passant par la commande « Import » (option « Videoscape... ») de son menu « File ». Par ailleurs, un fichier de Videoscape comporte des objets, mais pas de caméras ni de sources de lumière. Rien ne vous interdit de les définir vous-même à votre convenance, mais lorsque l'on cherche juste à se faire une idée de l'aspect de ce que le script `fabrique_tout` a produit, il est préférable de partir d'une scène de Blender vide, mais comportant déjà une caméra, des sources de lumière et des réglages acceptables. Nous vous recommandons donc d'ouvrir le fichier `08_scene_vide` du répertoire `blender_fichiers` de notre CD-ROM, et d'effectuer l'importation à partir de lui.

8.5.2 La conversion vers POV-Ray, puis vers l'image de synthèse

Avec POV-Ray, le principe est sensiblement le même : il faut lancer le programme et lire le fichier produit par le script `fabrique_tout`, à savoir `scene.pov`... mais au départ, cette action conduira seulement à afficher à l'écran les lignes de code de ce fichier. Il faudra encore lancer la production d'une image de synthèse pour obtenir un résultat plus intelligible.

Sous Windows, cela s'opère d'une façon extrêmement classique : on lit le fichier par la commande « Open... » du menu « File », puis on clique sur l'icône « Run » de la barre d'outils. Une image de synthèse sera immédiatement calculée et affichée – et en outre stockée dans le même répertoire que le fichier `scene.pov`, sous le nom `scene.png`. Ne vous croyez pas obligé de relancer la production de cette image chaque fois que vous voulez la voir ! N'importe quel logiciel de retouche d'image peut exploiter le fichier `scene.png` – y compris d'ailleurs si vous avez quitté POV-Ray.

Sur Macintosh, bien que l'interface du logiciel soit différente, le procédé est sensiblement le même : on ouvre `scene.pov` classsiquement, par la commande « Open... » du menu « File ». Cependant, il peut être nécessaire de valider « All

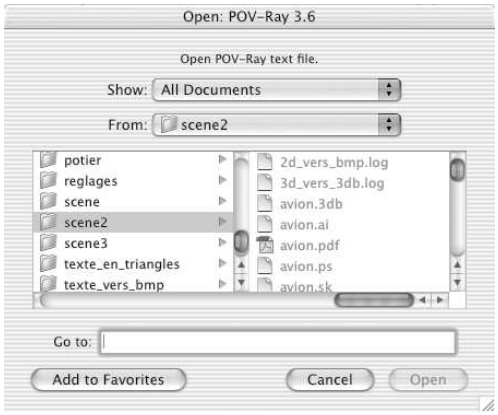


FIG. 8.12 – *L'extension .pov ne suffit pas pour que la version Mac de POV-Ray reconnaisse directement les fichiers qui lui sont destinés : il faut valider « All Documents » dans le sous-menu « Show » au moment de l'ouverture de chacun d'entre eux.*

Documents » au lieu de l'option par défaut « All Readable Documents » dans le sous-menu « Show » en haut de la fenêtre (fig. 8.12) : bien que POV-Ray fonctionne sous Mac OS/X, le logiciel a gardé le défaut congénital du système d'exploitation d'Apple (maintenu bec et ongles jusqu'à la version 9 du système), et les logiciels ne consentent a priori à ouvrir que les fichiers qu'ils ont enregistrés eux-mêmes... et justement, `scene.pov` a été produit par un script extérieur. Mais si l'on valide « All Readable Documents », le problème disparaît. Quant à la production de l'image, elle s'opère bien entendu avec le menu « Render ». Par défaut, POV-Ray pour Mac produit des images au format PICT, spécifique au Macintosh, mais nous verrons ultérieurement que ce réglage peut être modifié. Là encore, vous n'avez nul besoin de relancer la production de l'image de synthèse chaque fois que vous voulez la voir : Photoshop ou un programme équivalent lit évidemment le format PICT, et peut le faire même si vous avez quitté POV-Ray.

Enfin, sous Linux, la procédure est encore beaucoup plus simple... car il n'y a pas de procédure du tout ! En effet, le script `fabrique_tout` ou le `Makefile` ont directement produit l'image de synthèse (ce qui explique pourquoi POV-Ray pour Linux ne dispose d'aucune interface : il n'en a aucun besoin). Vous trouverez cette image sous le nom `visu.png` dans le sous-répertoire `images`, et vous l'ouvrirez avec votre logiciel de retouche préféré (c'est-à-dire très probablement Gimp).

8.5.3 La visualisation en « fil de fer »

A vrai dire, vous n'avez même pas vraiment besoin d'outils aussi lourds que Blender et POV-Ray pour vous faire une idée du contenu de vos fichiers SGML 3D. En effet, le script `fabrique_tout` produit directement quatre images de contrôle en mode « fil de fer », au format BMP à 16 couleurs. Vous les trouverez dans le sous-répertoire `images`, et elles seront mises à jour automatiquement à chaque lancement du script `fabrique_tout` (ou à chaque appel de la commande `make` sous Linux et Mac OS/X).

FIG. 8.13 – *La 3D ne se résume pas à l'image de synthèse. La visualisation des bâtiments en mode « fil de fer », très lisible lorsqu'on la fait sortir d'une imprimante à laser, est la préférée des architectes.*

A quoi ces images servent-elles ? Honnêtement, tant que Blender et POV-Ray vous donnent toute satisfaction, pas à grand-chose. Mais êtes-vous parfaitement sûr que vous ne rencontrerez jamais dans l'avenir de difficulté à faire tourner ces logiciels avec un nouvel ordinateur, une nouvelle carte graphique, un nouveau système d'exploitation ? Eh bien, si ces déboires venaient à se produire, vous pourriez quand même continuer à travailler jusqu'à la résolution du problème (jusqu'à la sortie d'une nouvelle version de ces programmes, par exemple) à l'aide de ces médiocres petites images. Quant à l'auteur, il lui est arrivé par le passé d'utiliser cette technique pour travailler avec des machines antédiluviennes, comme... des portables sous MS-DOS, bien évidemment incapables de faire tourner Blender. Même ces machines antiques peuvent mâcher le travail d'un amateur de 3D, car

les petits convertisseurs qui produisent ces images ne demandent aucune puissance.

En outre, pour produire ces images bitmap en 16 couleurs, il a d'abord fallu élaborer des données vectorielles en SGML 2D : ce sont les fichiers `dessus.2d`, `face.2d`, `profil.2d` et `axonom.2d`. Le format SGML est fait pour être traduit, et il n'est pas très difficile de convertir ces fichiers pour qu'une imprimante à laser les exploite – et ce type de documents fait le bonheur des architectes. D'ailleurs, c'est très exactement ainsi que la fig. 8.13 a été produite...

8.6 Le paramétrage de la scène

Le script `fabrique_tout` n'est qu'un automatisme informatique. Sa seule vertu est d'enchaîner toutes les commandes nécessaires pour produire en un seul mouvement des données totalement identiques, que ce soit pour Blender ou pour POV-Ray, et par ailleurs absolument conformes à un paramétrage que vous pouvez très facilement modifier à votre guise avec un éditeur de lignes – voire avec de petits scripts d'automatisation, comme nous le verrons.

Le contenu de la scène est évidemment contrôlé par le fichier SGML 3D `source.3d`, et éventuellement par toute la hiérarchie SGML qui en dépend grâce à l'emploi des balises `<inclusion>` ; reportez-vous au chapitre 4 pour adapter cette hiérarchie selon ce que vous voulez faire.

La position de la caméra de POV-Ray et le cadrage des vues en « fil de fer » du sous-dossier `images` sont quant à eux entièrement déterminés par le contenu du fichier `reglages.ini`, que nous vous recommandons d'ouvrir maintenant avec votre éditeur de lignes. Voici une explication détaillée de la demi-douzaine de paramètres qu'il contient :

```
xObs = 15.0
yObs = -20.0
zObs = 5.0
```

Ces trois valeurs correspondent aux coordonnées de l'« observateur », c'est-à-dire la caméra virtuelle de POV-Ray. Si vous définissez dans Blender une caméra ayant les mêmes coordonnées, vous obtiendrez bien entendu un angle de visualisation équivalent – et symétriquement, si vous déterminez avec Blender un point de vue qui vous paraît intéressant, vous pourrez en faire profiter POV-Ray rien qu'en indiquant dans `reglages.ini` les coordonnées de la caméra que vous aurez mise en place avec Blender (rappelons que sous Blender, la touche N vous donne accès aux coordonnées de n'importe quel objet sélectionné).

```
xDest = 0.0
yDest = 0.0
zDest = 1.65
```

Ces trois coordonnées sont celles d'un point vers lequel le centre de la caméra de POV-Ray se focalise (`dest` abrège bien sûr *destination*). Il n'est pas du tout obligatoire qu'un objet se trouve à cet endroit. Par ailleurs, c'est aussi ce point qui détermine le centre des quatre vues en « fil de fer ».

```
angle = 28.0
```

La valeur `angle`, exprimée en degrés, sert à calculer la focale de la caméra de POV-Ray. Un angle de 45° correspond à l'objectif de 50 mm d'un appareil photo 24 x 36.

```
largeur = 12.0
unite = 2.0
```

Ces deux paramètres concernent les vues en « fil de fer ». Le premier indique la largeur en mètres de la portion de la scène représentée par l'image. Le second concerne l'unité de largeur du quadrillage qui apparaît automatiquement sur les vues de dessus, de face et de profil. Comme ce quadrillage est réglé pour comporter un repère de couleur différente toutes les cinq unités, il est recommandé de toujours faire en sorte que le seul chiffre différent de zéro dans cette valeur soit un 1 ou un 2 : ainsi, les lignes de couleur différentes désigneront toujours un multiple de 5 (5 m, 5 cm, 5 km, si l'unité vaut 1.0, 0.01 ou 1 000, respectivement) ou de 10 (10 m, 10 cm ou 10 km, avec une unité valant 2.0, 0.02 ou 2 000, respectivement).

```
RotZAxonom = -45.0
RotXAxonom = -45.0
```

Ces deux valeurs concernent l'axonométrie – qui est en fait la projection d'une vue de dessus, mais effectuée à partir d'une copie de la scène que l'on a fait pivoter deux fois : d'abord autour de l'axe des Z (pour changer son orientation dans le plan horizontal), puis autour de l'axe des X (pour l'incliner). En général, vous n'avez pas intérêt à modifier la seconde valeur. En revanche, pour l'architecture, la visualisation de l'angle sud-est n'est pas toujours la meilleure ; vous pouvez choisir 45.0 pour voir l'angle sud-ouest, 135.0 pour l'angle nord-ouest, -45.0 pour l'angle sud-est et -135.0 pour l'angle nord-est.

```
systExpl = linux
permisDEcraser = TRUE
```

Nous l'avons déjà vu, vous devez impérativement modifier la valeur de `systExpl` avec le paramètre qui convient pour votre système d'exploitation : `windows`, `macintosh` ou `linux`. Sans cette modification, le script `fabrique_tout` ne pourra fonctionner correctement.

Quant au paramètre `permisDEcraser`, mieux vaut le laisser à TRUE (la valeur inverse serait bien sûr FALSE), faute de quoi le script `fabrique_tout` vous ennuiera en vous demandant l'autorisation d'écraser chacun des fichiers qu'il a fonction de mettre à jour.

Chapitre 9

Du dessin à main levée vers la 3D

9.1 Dessiner des silhouettes en triangles

9.1.1 Marier le dessin manuel et la console

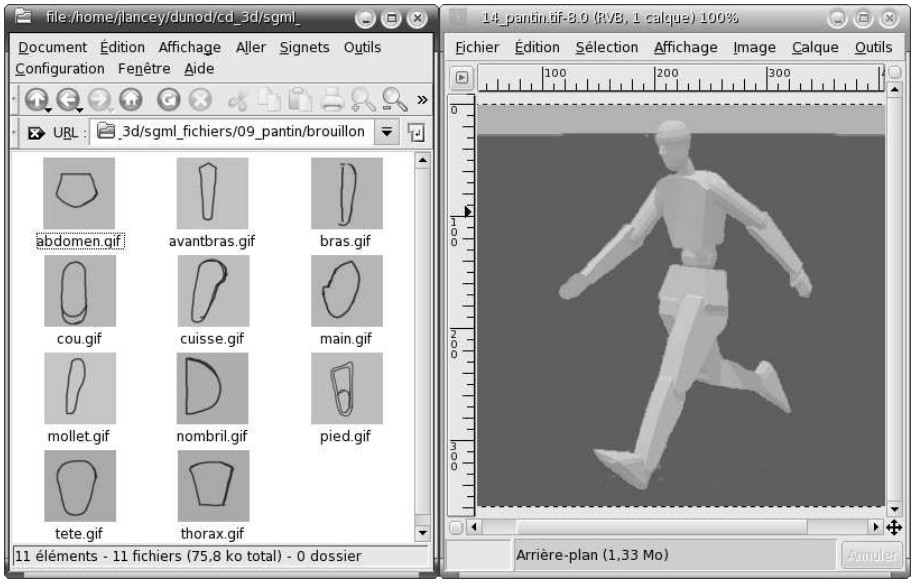


FIG. 9.1 – *Ce pantin articulé, dont vous pourrez étudier les composants dans le sous-répertoire* `09_pantin` *du dossier* `sgml_fichiers`*, est presque exclusivement constitué d'extrusions et de tours de potier, convertis à partir de dessins tracés à main levée et sans soin aucun (à gauche).*

Si vous vous sentez plus graphiste qu'informaticien, vous trouvez certainement la troisième partie de cet ouvrage assez rebutante : nous vous parlons de ma-

nipulation de code informatique, alors que vous auriez au contraire aimé pouvoir utiliser en 3D votre talent graphique. Eh bien, ce chapitre va peut-être vous réconcilier avec l'informatique austère, car vous verrez qu'elle constitue bel et bien le moyen *le plus simple*, et de loin, d'exploiter votre habileté au dessin à main levée. En particulier, la technique que nous expliquons ici peut vous permettre de définir un grand nombre de formes en 3D *avec un crayon sur du papier*. Un *vrai* crayon sur du *vrai* papier : vous n'avez pas besoin de faire les frais d'une tablette graphique coûteuse – le plus minable scanner la remplacera avantageusement.

Si vous êtes déjà un graphiste informatisé, vous n'avez pas non plus besoin d'abandonner les logiciels dont vous avez acquis la maîtrise, comme peut-être Photoshop ou Illustrator. Certes, Blender, ou surtout ses concurrents commerciaux, essayent parfois de s'approcher de l'ergonomie de ces remarquables logiciels. Eh bien, il est totalement inutile de vous demander si ces imitations sont ou non convaincantes : gardez le logiciel graphique que vous connaissez, et servez-vous en. Vous allez voir qu'avec nos convertisseurs de quatre sous, *tout* ce que vous savez faire avec quelque technique graphique que ce soit peut être exploité en 3D maillée. Et en plus, c'est simple.

9.1.2 Des besoins absolument élémentaires

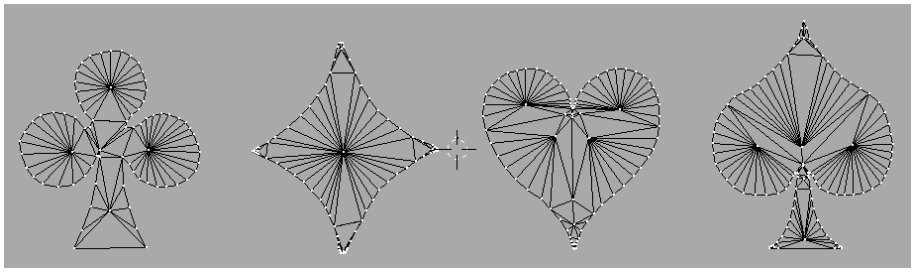


FIG. 9.2 – *Pour représenter les silhouettes courbes les plus diverses par un assemblage de triangles, il suffit de les découper en « parts de tarte » longues et étroites. Et la technique pour y parvenir est d'une simplicité déconcertante.*

Quelles que soient les qualités de Blender ou de ses concurrents, ils ne sauraient même approcher la facilité d'emploi du papier et du crayon quand il s'agit de dessiner à plat, en deux dimensions. Or, le dessin traditionnel est réellement utile à la 3D. Des motifs plats peuvent jouer un rôle décoratif ou signalétique (flèches directionnelles, pictogrammes...), et de très nombreuses formes en 3D (plates ou « faites au tour » comme des poteries) peuvent être très rapidement élaborées à partir d'ébauches planes. Mais surtout, tout ce que l'on sait dessiner sur

du papier (du croquis griffonné sur la page d'un carnet au schéma technique représentant des heures de travail à la planche à dessin), ou bien encore sur l'écran d'un ordinateur avec un logiciel de dessin quelconque, peut très facilement être converti en triangles. *N'importe quelle image peut être exploitée*, au moins à titre de brouillon, par les logiciels de 3D maillée.

Autre bonne nouvelle : le travail d'adaptation et de conversion en 3D des formes plates les plus diverses ne demande *aucune puissance* à votre ordinateur : il est réalisable même sur des machines absolument antiques, à la seule condition qu'elles puissent faire tourner un logiciel de retouche d'images. La puissance de ce logiciel est elle aussi assez indifférente. L'auteur travaille avec le logiciel libre Gimp, mais même une version vieille de dix ans d'un modeste freeware ferait parfaitement l'affaire, du moment qu'elle peut :

– afficher les images à divers taux d'agrandissement, et en particulier les forts grossissements (x 4 ou x 8) ;

– travailler chaque point d'une image (le terme correct n'est pas point mais pixel ; nous y reviendrons) au moyen d'un outil « crayon » bête et méchant, qui n'agisse que sur le point qu'on lui désigne (par opposition au comportement habituel d'un outil « pinceau », qui au contraire fait sciemment déborder sur les côtés la couleur choisie afin de créer un effet de flou ; dans le cas qui nous occupe, cet effet esthétique est tout à fait indésirable) ;

– convertir une image du mode couleurs au mode niveaux de gris, et inversement ;

– rééchantillonner une image pour lui donner des dimensions quelconques en pixels, que ce soit pour l'agrandir ou la rapetisser ;

– enregistrer en millions de couleurs au format BMP (aussi courant que Windows depuis que Windows existe).

Difficile de faire plus basique comme cahier des charges : pour trouver un logiciel de retouche qui ne sache pas faire tout cela, il faudrait vraiment remonter à la préhistoire de la micro-informatique. Bien entendu, si votre logiciel est capable de beaucoup d'autres choses (éclaircissement, rotation d'image, superposition de calques...), ce n'est aucunement un défaut ! Au contraire, plus vous serez un virtuose de la retouche d'images, plus votre logiciel facilitera votre travail de 3D. Mais même un savoir-faire élémentaire peut déjà rendre d'énormes services.

9.1.3 De la grille de pixels aux triangles

Les images manipulées par un logiciel de retouche, ou d'ailleurs toute image qui apparaît sur l'écran de votre ordinateur (même une photo, même le scan d'un

document), sont toujours affichées comme des mosaïques de petits points de couleur carrés, parfaitement alignés en colonnes et rangées. Soit dit en passant, il est préférable de parler de **pixels** (abréviation jargonnante de *picture elements*) plutôt que de points : même si le pixel est minuscule et facilement assimilable à un point sans épaisseur, il s'agit pourtant bel et bien d'une surface carrée. N'importe quelle image peut donc être considérée comme une grille de pixels, et à chaque pixel de cette grille correspond un numéro de colonne et un numéro de rangée. Si vous pouvez faire apparaître un motif quelconque sur l'écran de votre ordinateur, vous pouvez le transformer en grille de pixels, et même si le logiciel qui affiche cette image ne dispose d'aucune capacité d'enregistrement à un format graphique : il vous suffit de faire une copie d'écran pour avoir votre grille de pixels.

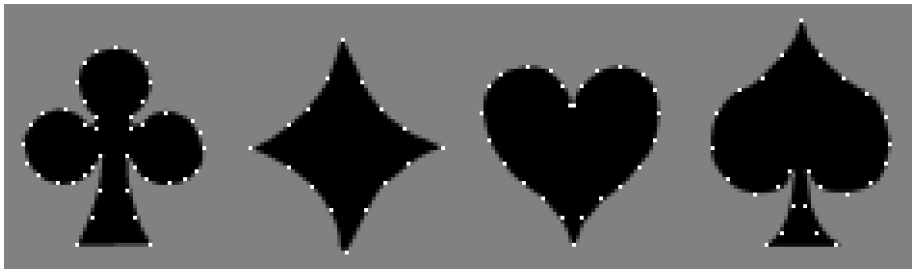


FIG. 9.3 – *Pour mettre en place les points caractéristiques, il est préférable d'afficher l'image à 400 % ou 800 %, afin que la grille de pixels soit bien lisible.*

L'astuce qui va nous permettre de passer de cette grille de pixels à des données mathématiques est simple comme bonjour. Il va seulement falloir inscrire sur cette image certains pixels faciles à repérer par leur couleur (fig. 9.3). Ces pixels auront bien sûr un positionnement parfaitement défini sur la mosaïque constituant l'image. L'astuce consistera à assimiler le numéro de leur colonne à une coordonnée X, et le numéro de leur rangée à une coordonnée Y. C'est tout. C'est en s'appuyant sur ce repérage qu'un convertisseur pourra très facilement définir (en SGML 2D) les innombrables triangles que la 3D maillée utilise pour représenter les formes les plus diverses.

A partir d'un simple repérage de pixels sur une image *absolument quelconque*, des automatismes informatiques peuvent effectuer toutes sortes de traitements et de calculs géométriques. En particulier, les convertisseurs du CD-ROM joint peuvent classer les points de haut en bas, ou encore, plus subtil, en fonction de leur position angulaire par rapport à un point plus ou moins central. Un périmètre de points très grossièrement circulaire pourra alors facilement être découpé en une multitude de triangles ayant tous ce point central pour sommet (fig. 9.2). Cet ensemble de triangles, aussi abondant et détaillé que nécessaire, sera très facile à

intégrer dans n'importe quelle scène de 3D maillée. Ce n'est pas plus malin que ça !

9.1.4 « Patatoïdes » et « parts de tarte »

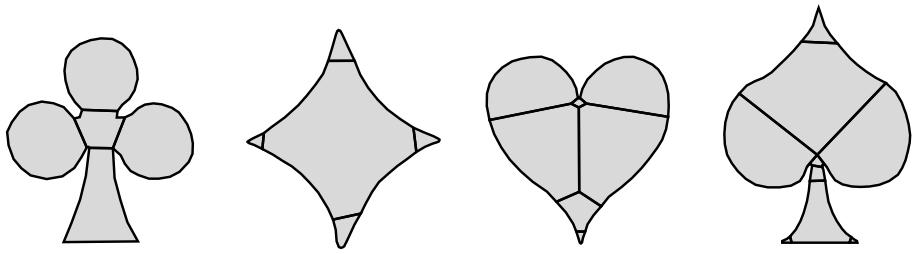


FIG. 9.4 – *Pour faciliter la conversion en triangles, il faut découper chaque silhouette en « patatoïdes », c'est-à-dire en surfaces vers le milieu desquelles on puisse placer un point d'où on pourra faire rayonner des segments de droite vers tout le périmètre, sans jamais couper ce dernier.*

Il faut bien comprendre que cette astuce informatique est dépourvue de toute sophistication. Les traitements automatiques que l'image subira sont totalement dépourvus de subtilité – en particulier d'aptitude à la reconnaissance de formes. Il va donc falloir mâcher quelque peu le travail des convertisseurs en découpant les silhouettes en formes simples (fig. 9.4), puis en enregistrant chacune d'entre elles dans un fichier indépendant, susceptible de faire l'objet d'un traitement automatique.

L'auteur a coutume d'appeler **patatoïde** les surfaces très grossièrement circulaires qui peuvent être découpées en une multitude de triangles longs et étroits – un peu comme on découpe un gâteau ou un fromage lorsque les convives sont nombreux. D'ailleurs, pour filer la métaphore, nous appellerons ces triangles étroits des **parts de tarte**. Espérons que ce vocabulaire culinaire saura vous mettre en appétit !

9.1.5 Le travail avec un logiciel de retouche

La toute première chose à faire avec la forme que vous souhaitez manipuler en 3D est évidemment de la dessiner... avec la technique qui vous agrée. Si vous voulez l'élaborer au crayon sur du papier, allez-y ! Cependant, les convertisseurs que nous employons ne savent pas travailler avec autre chose que le format BMP en millions de couleurs, et vous aurez donc éventuellement besoin de recourir à une conversion.

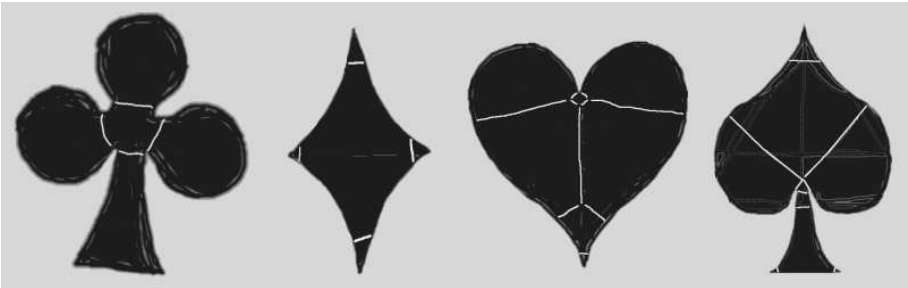


FIG. 9.5 – *Ebauche des formes souhaitées, dessinées à main levée et sans soin, ainsi qu'il est normal. La plupart des défauts visibles (le trèfle n'est pas vertical, le carreau est trop étroit, les formes ne sont ni bien proportionnées ni bien alignées) resteront sans conséquence, car il sera facile de les corriger ultérieurement à l'aide de balises* `<modifcoord>`.

Si vous avez élaboré votre dessin avec un logiciel de retouche, contentez-vous de l'enregistrer au format BMP. Si vous avez employé un logiciel n'ayant pas de fonctions d'enregistrement au format BMP, il vous suffit de faire apparaître à l'écran la forme qui vous intéresse et d'effectuer une copie d'écran (selon la manipulation adaptée à votre interface graphique : appui de la touche « Impr écran » sous Windows, Pomme-majuscule-3 sur Mac, capture d'écran de Gimp sous Linux, etc.) ; ensuite, ouvrez cette copie d'écran avec un logiciel de retouche, et là encore, enregistrez-la au format BMP. Quant au dessin sur papier, il vous faudra évidemment l'informatiser à l'aide d'un scanner – et si votre programme de scan ignore le format BMP, choisissez-en un autre (TIFF, ou JPG à la rigueur) ; vous effectuerez la conversion vers le format BMP plus tard, avec un logiciel de retouche.

Si jamais le fichier graphique que vous souhaitez exploiter n'est pas enregistré en millions de couleurs (si c'est une image en noir et blanc, par exemple), cela n'est pas grave : votre logiciel de retouche dispose forcément d'une fonction permettant de le passer en mode « RVB » ou « RGB » – ce qui est une autre façon de désigner le mode millions de couleurs.

Notez que nous allons devoir un peu endommager ce fichier informatique BMP pour y enregistrer des pixels de repérage. N'hésitez donc pas à travailler sur une copie pour conserver un original impeccable.

Pour poser des repères sur l'image, vous allez devoir travailler à l'échelle du pixel. Plutôt que de vous user les yeux, vous avez tout intérêt à afficher votre image à 400 %. Si vous n'êtes pas un fanatique de la précision, vous pouvez d'ailleurs la rééchantillonner pour lui donner une toute petite taille, de 128 pixels de large par exemple : de cette façon, vous pourrez voir l'image entière à 400 %, et il vous

sera bien plus facile de mettre en place les pixels de repère.

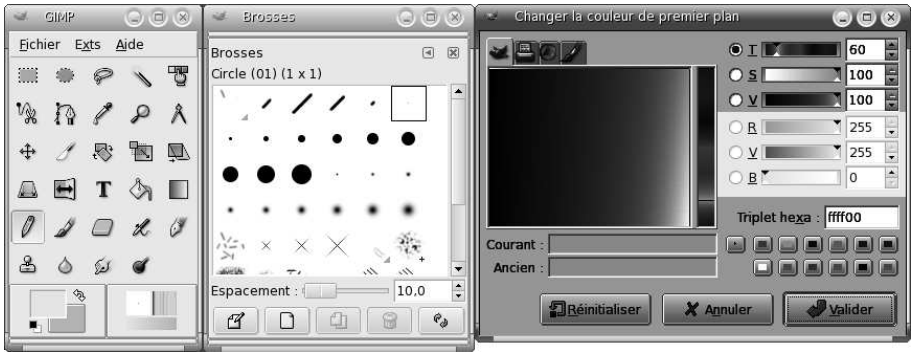


FIG. 9.6 – *Paramétrage du logiciel de retouche Gimp pour le placement des pixels repères. A gauche, choix de l'outil crayon. Au centre, choix de la brosse pixel. A droite (dans la zone que nous avons fait ressortir), choix du jaune pur : rouge 255, vert 255, bleu 0.*

Même si l'interface des logiciels de retouche n'est pas du tout unifiée, l'outil permettant de travailler pixel par pixel est généralement symbolisé par l'icône d'un crayon (ne confondez pas avec le pinceau). Vous devrez en outre choisir la forme de la « brosse » (c'est-à-dire de la surface qui sera modifiée sur l'image chaque fois que vous y effectuerez un clic de souris) ; la brosse qui nous intéresse est constituée d'un seul pixel... elle est donc minuscule et difficile à voir, mais votre logiciel en dispose forcément (fig. 9.6).

Il vous faut à présent marquer sur l'image le périmètre des patatoïdes avec des pixels parfaitement jaunes ; en outre, vous devrez marquer un pixel parfaitement cyan au centre de chaque patatoïde (plus exactement, à un endroit où le découpage des « parts de tarte » pourra être effectué sans affecter le périmètre du patatoïde ; pour mieux comprendre à quoi cela correspond, comparez les fig. 9.2 et 9.4).

Pour obtenir du jaune et du cyan purs avec des logiciels de retouche, nous devons hélas nous conformer au modèle colorimétrique rouge-vert-bleu – qui nous inspire pourtant une exécration cordiale. Dans ce modèle, le jaune pur est le résultat de l'addition du rouge et du vert purs, et le cyan pur le résultat de l'addition du vert et du bleu purs. En d'autres termes, le jaune est généralement obtenu par les valeurs rouge 255, vert 255, bleu 0 ; le cyan, lui, est obtenu avec rouge 0, vert 255, bleu 255. Nos convertisseurs ne sont pas du tout subtils, et c'est très exactement ces valeurs-là que vous devez choisir : tapez-les bien au clavier, ne vous contentez pas que le résultat vous paraisse jaune ou cyan à l'œil.

Une fois que vous aurez repéré les pixels pertinents sur l'image en leur donnant la couleur adéquate (jaune pour le pourtour de la forme, cyan pour son

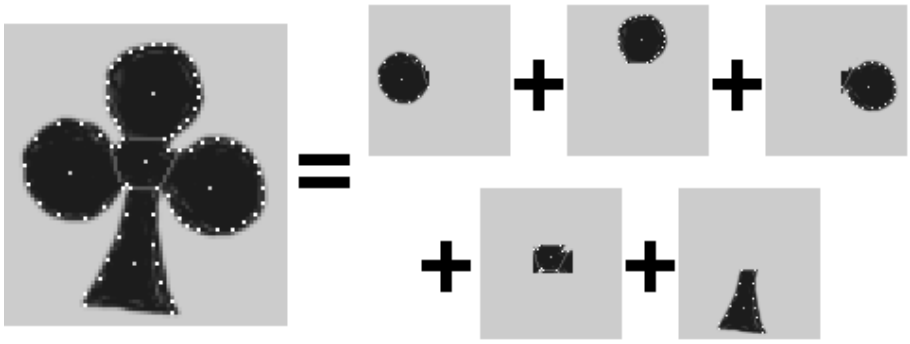


FIG. 9.7 – *Chaque « patatoïde » devra être enregistré en un fichier BMP indépendant pour permettre sa conversion automatique en données vectorielles.*

centre), il faudra démultiplier chaque image de départ en autant de copies qu'elle comporte de « patatoïdes », puis ne laisser subsister sur chacune de ces copies qu'un seul patatoïde, en effaçant les parties non pertinentes et surtout les pixels jaunes et cyan qu'elles peuvent contenir. Peu importe comment vous les effacez ; vous pouvez par exemple les barbouiller avec la couleur de fond (fig. 9.7). Inutile de fignoler du moment que vous faites bien disparaître de chaque image *tous* les pixels jaunes et cyan sans rapport avec le patatoïde qui vous intéresse (n'en oubliez surtout aucun !). Attention : l'image de chaque patatoïde doit impérativement conserver le cadrage initial de l'ensemble : même si le patatoïde qui vous intéresse n'occupe qu'un tout petit espace dedans, ne rognez surtout pas l'image !

Si tout cela vous paraît un peu abstrait, allez donc voir dans le sous-répertoire `09_clr_cart` du répertoire `sgml_fichiers` de notre CD-ROM. Il contient quatre fichiers bitmap décrivant les couleurs de cartes à jouer (fig. 9.5 ; ces dessins ont bien sûr été tracés à main levée, d'ailleurs sans aucun talent, mais ils sont tout à fait exploitables pour la 3D). Les pixels importants (de couleur jaune et cyan) y ont déjà été mis en place (vous avez tout intérêt à afficher les images à 400 % pour mieux les voir), et de gros traits rouges signalent les frontières des patatoïdes à découper. En outre, le sous-répertoire `09_trefle_decoupe` contient déjà les

cinq images correspondant au découpage du trèfle (fig. 9.7). Inspirez-vous de cet exemple pour découper vos propres formes.

9.1.6 La conversion du format bitmap vers SGML 2D

Lorsque vous disposez d'un fichier BMP avec les pixels repères jaunes et cyan pour chaque patatoïde, il ne vous reste qu'à demander à nos convertisseurs d'achever le travail en effectuant la conversion de ces formes vers le format SGML 2D. Il serait tout à fait possible de le faire en tapant quelques commandes à l'invite, mais le sous-répertoire `09_trefle_decoupe` contient déjà deux scripts qui peuvent vous épargner cette peine. Il y en a deux car l'un est un batch pour Windows, l'autre un script shell pour Linux (il fonctionnera aussi sous Mac OS/X). Vous pouvez exécuter celui des deux qui correspond à votre système d'exploitation si vous recopiez ce dossier dans le répertoire `3d_win, 3d_linux` ou `3d_mac` de votre disque dur (pensez à enlever l'attribut « lecture seule » !). Vous pouvez aussi, et nous vous le recommandons, le lire pour en comprendre la logique. Voici d'ailleurs un commentaire du script shell.

```
../bin/bmp_vers_2d trefle_centre.bmp trefle_centre.2d
../bin/bmp_vers_2d trefle_feuille_droite.bmp trefle_feuille_droite.2d
../bin/bmp_vers_2d trefle_feuille_gauche.bmp trefle_feuille_gauche.2d
../bin/bmp_vers_2d trefle_haut.bmp trefle_haut.2d
../bin/bmp_vers_2d trefle_queue.bmp trefle_queue.2d
```

Ces cinq lignes appellent le convertisseur `bmp_vers_2d` qui, comme son nom l'indique (mais si), a pour fonction de traduire au format SGML 2D un fichier bitmap comportant les pixels repères. Sur chaque ligne, le nom de ce programme est suivi de façon classique par le nom du fichier à lire (un fichier bitmap) et celui du fichier à écrire (c'est-à-dire a priori le même nom, mais avec l'extension .2d, puisqu'il s'agira d'un fichier SGML 2D).

```
cp ../bin/2d_vers_2db.ini .
../bin/modifini 2d_vers_2db.ini source [source.2d]
../bin/modifini 2d_vers_2db.ini resultat [trefle.2db]
../bin/modifini 2d_vers_2db.ini systExpl [linux]
../bin/2d_vers_2db
```

Ces cinq autres lignes sont beaucoup moins compliquées qu'elles n'en ont l'air. Toutes concernent le convertisseur `2d_vers_2db`, qui effectue la synthèse d'une hiérarchie SGML 2D. En anticipant un peu sur les explications du prochain chapitre, sachez que ce programme est de ceux que l'on paramètre à l'aide d'un fichier d'initialisation (d'extension .ini) à placer dans le répertoire d'où le programme est lancé. Donc, la première ligne effectue la recopie dans le répertoire courant d'un fichier d'initialisation-type correspondant à ce programme (`cp` est la commande Unix pour la copie de fichiers ; ce serait `copy` avec Windows). Ce

fichier va devoir être adapté. Si vous le faisiez vous-même avec votre éditeur de lignes, cela ne vous prendrait que quelques secondes – mais l'automatisation de cette opération avec le petit programme `modifini` donne une fausse impression de complexité : les trois lignes suivantes servent seulement à indiquer dans ce fichier les valeurs pertinentes pour trois paramètres ; premièrement, le fichier à lire (paramètre `source`, valeur `source.2d` ; il s'agit d'un fichier SGML 2D placé hiérarchiquement au-dessus des cinq que le script fabrique) ; deuxièmement, le fichier à écrire (paramètre `resultat`, valeur `trefle.2db` ; il s'agit d'un fichier SGML 2D d'un seul tenant, qui sera plus facile à manipuler et à traduire) ; troisièmement, le système d'exploitation (et vous constaterez sans surprise que dans le batch équivalent de ce script, c'est `windows` que nous avons indiqué au lieu de `linux`). Enfin, la dernière de ces cinq lignes lance le convertisseur.

```
rm -f trefle_*.2d
rm -f 2d_vers_2db.*
```

Ces deux dernières lignes effacent les fichiers dont on n'a plus l'usage avec la commande Unix `rm -f` (ce serait `del` avec Windows) et aussi une syntaxe un peu cryptique utilisant le caractère `*`. Ce n'est pas le sujet de ce livre que de vous expliquer cette syntaxe (pas bien compliquée pourtant). Il vous suffit de savoir que le script opère une purge que vous auriez parfaitement pu effectuer vous-même en faisant glisser des icônes dans la corbeille. Les fichiers éliminés sont le fichier d'initialisation (pourtant tout juste recopié) et les fichiers SGML 2D (pourtant tout juste créés eux aussi, mais déjà recopiés dans `trefle.2db`). En d'autres termes, après exécution du script, le seul résultat visible est l'apparition d'un fichier `trefle.2db` unique, qui sera très facile à manipuler.

Magique, non ? Notons quand même que ce traitement n'a pu être automatisé que grâce à la rédaction préalable d'un tout petit fichier SGML 2D (`source.2d`) conçu pour rassembler les cinq patatoïdes grâce aux balises `<inclusion>`. Voici le contenu de ce fichier, qui ne devrait pas vous poser de problèmes de compréhension :

```
<inclusion>
trefle_centre.2d
</inclusion>
<inclusion>
trefle_feuille_droite.2d
</inclusion>
<inclusion>
trefle_feuille_gauche.2d
</inclusion>
<inclusion>
trefle_haut.2d
</inclusion>
```

```
<inclusion>
trefle_queue.2d
</inclusion>
```

Peut-être trouvez-vous tout cela bigrement – et inutilement – complexe. Vous faites erreur : avec un peu d'habitude (et vous aurez acquis cette habitude dès que vous aurez converti deux ou trois formes en patatoïdes), vous vous rendrez compte que tout ce code informatique peut être écrit à toute vitesse, à grands renforts de copier-coller et de recherches-remplacements : si vous avez vraiment compris comment ça marche, vous n'aurez aucune peine à vous servir de votre éditeur de lignes pour le refaire à toute vitesse avec n'importe quelle forme. Pensez en outre que vous pourrez utiliser indéfiniment le code SGML ainsi produit, même dans quarante ans... Cela vaut bien quelques minutes d'efforts !

9.2 Traduction du format SGML 2D au format SGML 3D

Le fichier `trefle.2db` que nous venons de faire écrire par un script n'est pas encore directement exploitable en 3D, mais la manipulation pour y parvenir n'est qu'une simple formalité :

```
../bin/2db_vers_3d trefle.2db trefle.3d
```

Cet appel au convertisseur `2db_vers_3d` suffit à produire un fichier SGML 3D dont la géométrie est valide. Il est toutefois souhaitable de le modifier manuellement avec un éditeur de lignes pour lui conférer une couleur. Evitez le noir complet (qui resterait d'un noir absolu sous tous les éclairages), et choisissez plutôt un gris sombre en ajoutant au tout début du fichier `trefle.3d` les trois lignes suivantes :

```
<couleur>
240.0, 0.0, 0.2
</couleur>
```

Ainsi modifié, le fichier `trefle.3d` peut être inclus sans difficulté dans une scène 3D. Rien n'interdit, par exemple, de le déplacer dans le répertoire employé au chapitre précédent, et de l'inclure dans la scène exactement comme un autre des trois objets qui y figurent déjà, avec des balises `<inclusion>`. Cela étant, toutes les formes traduites par `2db_vers_3d` sont au départ incluses dans le plan XY (leur coordonnée Z est égale à 0.0), et il faut donc les déplacer avec des balises `<modifcoord>` pour qu'elles soient visibles.

Vous pouvez aussi rendre cette forme exploitable par Blender grâce au convertisseur `3db_vers_obj` :

```
../bin/3db_vers_obj trefle.3d trefle.obj
```

Blender pourra utiliser le fichier `trefle.obj` de la même façon que tous les fichiers enregistrés au format OBJ de Videoscape : il suffira de l'importer avec le menu « File », commande « Import », option « Videoscape... ».

9.3 L'extrusion

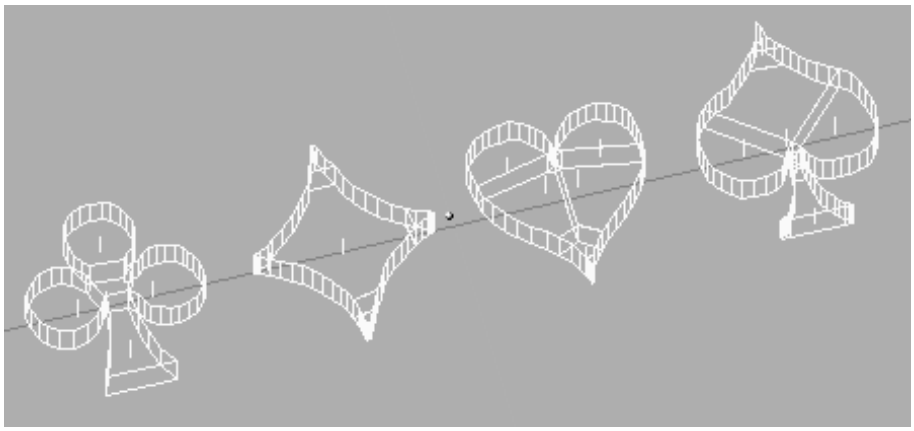


FIG. 9.8 – *Extrusions des formes précédents, visualisées en fil de fer après réimportation dans Blender. Le découpage en « patatoïdes » est encore visible, mais il sera indécelable avec la représentation en faces cachées.*

Nous avons vu au chapitre 7 comment procéder à une extrusion avec Blender, et la technique est tout à fait applicable après importation d'un fichier OBJ résultat d'une conversion, tel que celui dont nous venons de parler. Mais il est également possible de procéder à l'extrusion en ligne de commande, par exemple comme ceci :

```
../bin/extrusion trefle.2db trefle_extrude.3d
```

Là encore, le fichier SGML 3D obtenu peut être inclus dans une scène comme l'un quelconque des objets 3D manipulés au chapitre précédent. Il convient toutefois de remarquer que, par défaut, le convertisseur `extrusion` confère aux objets qu'il produit une épaisseur de 1.0, qui est très généralement excessive. Cela se corrige facilement. Par exemple, si l'on veut inclure le trèfle extrudé dans la scène en lui conférant une épaisseur de 15 cm (soit 0.15 m), il suffira d'écrire dans le fichier `source.3d` :

```
<modifcoord>
Tail 1.0,1.0,0.15
</modifcoord>
```

```
<inclusion>
trefle_extrude.3d
</inclusion>
<modifcoord>
Fin
</modifcoord>
```

C'est en fait très pratique que le convertisseur `extrusion` confère systématiquement une épaisseur de 1.0, sans s'interroger sur la taille de la surface qu'il traite. De cette façon, il suffit d'indiquer l'épaisseur que l'on souhaite sur la ligne contenant le mot-clé `Tail` – alors qu'il faudrait effectuer une règle de trois inutilement compliquée si l'épaisseur de l'objet au départ était de quelque autre valeur que ce soit.

9.4 Le tour de potier

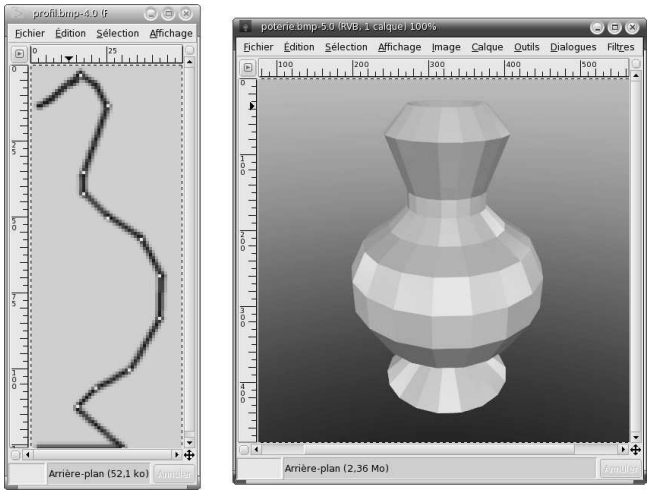


FIG. 9.9 – *Pour passer du profil de gauche à l'objet 3D de droite, il suffit d'utiliser deux convertisseurs.*

Nous avons vu comment nous pouvions aboutir à des extrusions de patatoïdes à partir d'une simple image bitmap judicieusement pourvue de pixels jaunes et cyan, en enchaînant dans l'ordre l'action des convertisseurs `bmp_vers_2d`, `2db_vers_3d` et `extrusion` (en fait, nous avons même intercalé le convertisseur `2d_vers_2db` pour réaliser la synthèse de plusieurs patatoïdes). Cela doit commencer à vous faire comprendre tout le parti qu'on peut tirer de l'usage

de convertisseurs, et le prochain chapitre vous donnera toutes les armes dans cet esprit.

Mentionnons sans attendre que, de façon encore plus simple, nos utilitaires peuvent engendrer des objets ronds, fabriqués comme sur le tour d'un potier, à l'aide des convertisseurs `bmp_vers_profil` et `potier` (fig. 9.9). Le premier exploite une image bitmap où deux points cyan définissent un axe vertical, et où un nombre quelconque de points jaunes décrivent un profil, et il exploite ces données pour écrire un descriptif de ce profil au format SGML 2D. Le second convertisseur, `potier`, part de ce descriptif SGML 2D et crée un objet SGML 3D ayant l'aspect d'une poterie de même profil. Le sous-répertoire `09_potier` du répertoire `sgml_fichiers` de notre CD-ROM, là encore doté de deux scripts d'automatisation, en fait la démonstration. Nous n'entrerons pas dans les détails, car cela nous conduirait à répéter sans grand changement ce que nous avons vu pour l'extrusion. L'étude des scripts du répertoire `09_potier`, ainsi que le prochain chapitre, devraient vous suffire à tirer parti de cette possibilité.

Chapitre 10

Extensions et convertisseurs

10.1 Logique employée par les programmes de l'auteur

Sauf exception, les convertisseurs et autres petits programmes mis au point par l'auteur et diffusés comme logiciels libres sur le CD-ROM joint sont de l'un de ces deux types : avec ou sans fichier d'initialisation.

Les programmes du premier type n'utilisent pas de fichier d'initialisation. Ce sont généralement des convertisseurs à la logique élémentaire, qui lisent un fichier (Ascii dans la plupart des cas) et en écrivent un autre. Leur syntaxe d'appel est alors invariablement :

```
nom_du_convertisseur fichier_a_lire fichier_a_ecrire
```

et certains d'entre eux acceptent en outre un paramètre optionnel `-f` (qui doit être le dernier) pour autoriser l'écrasement éventuel d'une version précédente (supposée caduque) de fichier_a_ecrire (faute de quoi le programme demandera l'autorisation d'écraser ce fichier).

Comme les programmes de ce type ne peuvent rien faire sans qu'on leur fournisse des noms de fichier en argument, l'appel de ces programmes sans argument entraîne l'affichage d'un message d'erreur qui explique brièvement la mission du programme et sa syntaxe. Appeler ces programmes sans argument est donc un moyen d'en obtenir un mode d'emploi sommaire, qui peut utilement compléter les explications que nous donnons ci-dessous pour chacun des convertisseurs.

Les programmes du second type, plus élaborés, ont besoin de recevoir davantage d'informations, et les cherchent donc dans un fichier d'initialisation Ascii – dont on peut d'ailleurs indiquer le nom en argument. Mais si on ne fournit aucun argument, ce qui est la procédure normale, le programme cherchera les informations dont il a besoin dans un fichier d'initialisation portant le même nom que lui mais d'extension .ini, et situé dans le répertoire courant. Par exemple, la com-

mande
`../bin/2d_vers_bmp`
cherchera les informations qui lui sont nécessaires dans un fichier `2d_vers_bmp.ini` situé dans le répertoire courant (et non pas dans `../bin/2d_vers_bmp.ini`). Au cas où un tel fichier n'existerait pas dans le répertoire courant, le programme le signalerait. On pourrait alors résoudre la difficulté en plaçant dans ce répertoire une copie du fichier d'initialisation-type du sous-répertoire bin, et éventuellement, en adaptant ce fichier (en l'ouvrant avec un éditeur de lignes pour y inscrire les mentions pertinentes).

La syntaxe des fichiers d'initialisation est très simple et n'appelle pas de commentaires.

Signalons toutefois qu'au cas où l'on souhaite qu'un fichier d'initialisation soit modifié par un script plutôt que par une modification manuelle dans un éditeur de texte, on dispose pour cela du petit programme modifini.

Remarque
Comme un grand nombre des programmes de l'auteur sont des convertisseurs, leur nom comporte souvent la mention « _vers_ ».

10.2 Extensions

aucune extension

En général, les fichiers ne comportant aucune extension sont des programmes exécutables sous Linux ou Mac OS/X (généralement, il s'agit de convertisseurs écrits en langage C). Les programmes exécutables sous Windows, eux, portent l'extension .exe.

.2d

Descriptif SGML (Ascii) d'objet 2D (polygones ou traits droits). Un fichier .2d fait couramment appel à des sous-fichiers du même type.

.2db

Descriptif SGML (Ascii) d'objet 2D, dépourvu d'instructions de modifications de coordonnées et d'inclusion de sous-fichiers (ce qui facilite les conversions). Un fichier .2db est presque toujours le produit de la synthèse d'une hiérarchie de fichiers .2d (synthèse réalisée avec le convertisseur 2d_vers_2db).

.3d

Descriptif SGML (Ascii) d'objet 3D (polygones de couleur exclusivement). Un fichier .3d fait couramment appel à des sous-fichiers du même type.

.3db

Descriptif SGML (Ascii) d'objet 3D, dépourvu d'instructions de modifications de coordonnées et d'inclusion de sous-fichiers (ce qui facilite les conversions). Un fichier .3db est presque toujours le produit de la synthèse d'une hiérarchie de fichiers .3d (synthèse réalisée avec le convertisseur 3d_vers_3db).

.bat

Batch (en Ascii), c'est-à-dire script de Windows. Il ne peut être employé ni sous Linux ni sous Mac OS/X.

.blend

Fichier de Blender. C'est apparemment le format le plus rassurant, parce qu'il conserve la géométrie des objets, les éclairages et jusqu'à la configuration de l'interface. En outre, Blender assure une compatibilité parfaite de ses fichiers entre ses divers portages (Windows, Mac, Linux) – et même, jusqu'à un certain point, entre ses diverses versions (sauf exception, une version récente de Blender récupère sans problème une scène enregistrée par une version ancienne... et même l'inverse est généralement vrai !). Cela étant, le format .blend n'est reconnu que par Blender, sa structure interne n'est compréhensible que pour ses développeurs, et choisir ce format pour l'archivage revient à se rendre prisonnier de Blender. Or, aucun logiciel n'est éternel, pas même un logiciel libre... Par ailleurs, la compacité n'est assurément pas au nombre des avantages de ce format.

.bmp

Image bitmap, non compressée. Les logiciels de retouche d'image peuvent employer le format BMP pour stocker des images en millions de couleurs (trois octets par pixel), et c'est ce qu'il faut faire pour employer les convertisseurs bmp_vers_2d, bmp_vers_profil et bmp_vers_texture.

Par ailleurs, le convertisseur 2d_vers_bmp produit des fichiers BMP en 16 couleurs seulement (deux pixels par octet)... ce qui est moins beau mais six fois plus rapide et moins encombrant.

Les images .bmp peuvent être lues avec n'importe quel logiciel de retouche d'images, qu'elles soient en mode seize couleurs ou millions de couleurs.

Pour l'archivage éventuel, il est recommandé d'employer plutôt le format GIF, beaucoup moins encombrant... mais attention à bien régler les paramètres de la conversion si l'on ne veut pas risquer d'endommager les pixels jaunes et cyan.

.exe

Programme exécutable sous Windows (et même, en ce qui concerne nos convertisseurs, sous MS-DOS).

.gif

Format d'image compressé utilisant un nombre limité de couleurs (on dit aussi une *palette*) : 16 ou 256 couleurs le plus souvent. On peut avoir recours à ce format pour archiver les images bitmap employées par les convertisseurs bmp_vers_2d, bmp_vers_profil et bmp_vers_texture... au prix de certaines précautions car il ne conserve pas forcément les couleurs jaune et cyan nécessaires à l'emploi de ces programmes. GIF peut servir sans problème pour archiver les projections ou perspectives en mode « fil de fer », ou n'importe quelle image produite par 2d_vers_bmp.

Les images .gif peuvent être lues avec n'importe quel logiciel de retouche d'images.

.ini

Fichier Ascii d'initialisation, généralement utilisé par un petit programme portant le même nom. C'est presque toujours un simple inventaire de paramètres de la forme

```
nom_du_paramètre = valeur_du_parametre
```

De tels fichiers sont voués à être modifiés manuellement avec un éditeur de texte pour adapter à des besoins précis les résultats du programme auquel ils correspondent. Ils peuvent aussi être modifiés par un script faisant appel au programme modifini.

.jpg

Format d'image compressé en millions de couleurs. POV-Ray peut l'utiliser pour représenter des objets texturés, et le convertisseur bmp_vers_texture est écrit dans cet esprit.

Les images .jpg peuvent être lues avec n'importe quel logiciel de retouche d'images.

.log

Compte rendu Ascii créé automatiquement par certains programmes. Les comptes rendus de 3d_vers_3db et 2d_vers_2db (`3d_vers_3db.log` et `2d_vers_2db.log`, respectivement) peuvent aider à débusquer les erreurs dans les imbrications d'inclusions de fichiers et de modifications de coordonnées.

.obj

Ce format, appelé aussi Videoscape, est utilisé par Blender en import et en export. Employé avec le convertisseur obj_vers_3d, il peut nous permettre d'exporter les formes définies par Blender vers le format SGML 3D (souvent avec l'idée d'opérer ultérieurement une nouvelle conversion vers POV-Ray à l'aide de 3db_vers_pov). Symétriquement, le format de Videoscape peut nous servir à faire

manipuler par Blender des formes ou des scènes 3D définies en SGML : le convertisseur 3db_vers_obj permettra de les traduire au format de Videoscape, que Blender peut importer.

.png
Image bitmap compressée. C'est généralement à ce format que POV-Ray produit ses images de synthèse. On peut ouvrir ces images avec la plupart des logiciels de retouche, et les employer pour effectuer une conversion vers le format JPG, d'usage plus courant.

.pov
Fichier source de POV-Ray (en Ascii). Comme ce format est généralement très bavard, il est recommandé de le compresser si on tient à l'archiver.
POV est l'abréviation de *Persistence Of Vision.*

.ppm
Image bitmap enregistrée sans compression selon les spécifications du format Portable Pixel Map. Les routines graphiques de Python sont capables d'afficher ce format.

.py
Script Python (en Ascii).

.sh
Script shell d'Unix (en Ascii). Utile avec Mac OS/X et Linux, il ne peut être employé sous Windows.

10.3 Convertisseurs et autres petits programmes

2d_vers_2db (et 2d_vers_2db.ini)
Ce convertisseur effectue la synthèse d'une hiérarchie de fichiers SGML 2D pour écrire un fichier .2db unique où les instructions de modification de coordonnées éventuelles auront été prises en compte, mais éliminées. Le fichier .2db résultant est particulièrement facile à exploiter, notamment pour une conversion automatique.

2d_vers_bmp (et 2d_vers_bmp.ini)
Ce convertisseur transforme une hiérarchie de fichiers SGML 2D en une image bitmap en 16 couleurs, et la rend plus lisible en traçant à l'arrière-plan des axes repères. Plus subtil qu'il en a l'air, il permet de travailler à n'importe quelle échelle,

même pour représenter à l'échelle du micron des objets ayant des dizaines de milliers de kilomètres de large. Outre son utilité propre, il permet au débutant de s'habituer rapidement à la logique des fichiers hiérarchiques et des modifications de coordonnées (déplacement, rotations, mises à l'échelle).

2db_vers_3d

Ce convertisseur traduit en SGML 3D les polygones stockés dans un fichier .2db. Bien que le fichier résultat soit techniquement en 3D, et donc facile à intégrer dans une hiérarchie SGML 3D, toutes les formes qu'il décrit sont plates, horizontales et d'épaisseur nulle (et incluses dans le plan XY tant qu'elles n'ont pas subi de modifications de coordonnées). Si l'on souhaite leur donner une épaisseur, il faut employer le programme extrusion et non ce convertisseur.

Attention : toutes les couleurs sont éliminées dans une conversion par 2db_vers_3d.

3d_vers_3db (et 3d_vers_3db.ini)

Ce convertisseur effectue la synthèse d'une hiérarchie de fichiers SGML 3D pour écrire un fichier .3db unique où les instructions de modification de coordonnées éventuelles auront été prises en compte, mais éliminées. Le fichier .3db résultant est particulièrement facile à exploiter, notamment pour une conversion automatique.

3db_vers_2d

Ce convertisseur traduit les contours des polygones 3D d'un fichier .3db en une projection 2D, constituée d'une série de traits droits au format SGML 2D (représentation dite « fil de fer »). Il est appelé par divers scripts du sous-répertoire scene pour fabriquer des projections en vue de dessus, de face et de profil et même des perspectives axonométriques en mode « fil de fer ».

3db_vers_obj

Ce convertisseur traduit un fichier .3db en fichier .obj conforme aux spécifications du format Videoscape, et que Blender peut exploiter grâce à la commande « Import Videoscape » de son menu File. Le fichier traduit conserve les couleurs, mais pas les textures.

3db_vers_persp

Ce convertisseur prend en compte la position d'un observateur et la direction dans laquelle il regarde, lit les traits de contour des polygones 3D d'un fichier SGML 3D (en .3db) et rédige un descriptif SGML 2D permettant de les afficher en perspective et en « fil de fer ». L'interface textuelle place_camera a recours à ce convertisseur, qui peut rendre de grands services grâce à elle, en particulier pour paramétrer une caméra de POV-Ray. Toutefois, ce petit programme est loin de me-

ner à une qualité de visualisation irréprochable. Comme toujours avec le format SGML 2D, l'image qu'il produit comporte seulement 16 couleurs (au maximum), et les teintes originales sont donc extrêmement simplifiées ; l'éclairage de la scène n'est pas du tout pris en compte. Le programme n'affiche un trait de contour que si ses deux extrémités se trouvent dans le champ visuel de l'observateur (dans le cas contraire, le trait sera simplement ignoré). En outre, les lignes se superposent en fonction de l'ordre de leur apparition dans le fichier .3db, et dans certains cas, cela peut donner l'illusion que les objets les plus lointains sont placés au premier plan. Enfin, la perspective produite est très élémentaire : elle est constituée de traits rigoureusement droits, alors qu'une véritable perspective photographique (comme celle des images de synthèse de POV-Ray) les arrondirait à proximité de l'observateur ; en cas de disproportion entre les objets représentés, cela peut conduire à d'apparentes erreurs de positionnement, en particulier près de l'intersection des axes ; fort heureusement, ces aberrations visuelles disparaîtront lors de la production d'une véritable image de synthèse par POV-Ray.

3db_vers_pov

Ce convertisseur traduit un fichier .3db en série d'instructions décrivant un objet 3D selon la syntaxe de POV-Ray. Il conserve les couleurs et les textures.

Pour que POV-Ray puisse exploiter ces instructions, il faudra au moins en ajouter quelques-unes relatives à l'éclairage et à la position d'une caméra virtuelle. Généralement, cela est réalisé par la fusion à l'aide du petit programme gigogne de plusieurs sous-fichiers .pov respectant la syntaxe de POV-Ray.

bmp_vers_2d

Ce convertisseur exploite les pixels jaunes et cyan d'une image bitmap en millions de couleurs pour rédiger le descriptif SGML 2D d'un polygone, lui-même géré comme un assemblage de triangles ayant une pointe en commun, à la façon des parts d'une tarte.

bmp_vers_profil

Ce convertisseur exploite les pixels jaunes et cyan d'une image bitmap en millions de couleurs pour rédiger le descriptif SGML 2D du profil d'un objet rond, lequel profil pourra être exploité par le convertisseur potier.

bmp_vers_texture

Ce convertisseur exploite les pixels jaunes et cyan d'une image bitmap en millions de couleurs pour rédiger le descriptif SGML 3D d'un polygone texturé (la texture employée ayant le même nom que l'image bitmap, mais l'extension .jpg).

bmp16_vers_ppm

Ce convertisseur traduit au format Portable Pixel Map en millions de couleurs une image .bmp stockée avec une palette de 16 couleurs seulement. On l'emploie principalement pour faire afficher par un script Python une image produite par 2d_vers_bmp.

compte_3db

Ce petit programme calcule et affiche le nombre de points et de faces gérées par un fichier .3db, permettant ainsi de se faire une idée précise de sa complexité.

extrusion

Ce convertisseur transforme un descriptif d'une ou plusieurs surfaces SGML 2D (stocké en .2db) en objet SGML 3D ayant le même aspect en vue de dessus, mais doté d'une épaisseur de 1.0. Dans la plupart des cas, les surfaces du descriptif SGML 2D de départ ont été produites préalablement par bmp_vers_2d.

Attention : les couleurs éventuellement conférées aux surfaces dans le descriptif SGML 2D sont éliminées lors de la conversion, et il faudra penser à compléter manuellement le fichier produit pour lui conférer une couleur.

gigogne (et gigogne.ini)

Ce convertisseur n'est pas exclusivement conçu pour la 3D. Il permet de rassembler dans un fichier Ascii unique divers fichiers de texte disposés logiquement dans une structure hiérarchique, et qui s'incluent les uns dans les autres à la façon de poupées gigognes (d'où le nom du convertisseur). Le balisage à employer est le suivant :

```
<inclusion>
fichier_a_inclure.txt
</inclusion>
```

En fait, ce convertisseur n'est rien d'autre qu'une simplification de 3d_vers_3db et 2d_vers_2db, permettant un traitement similaire à celui de la synthèse des fichiers SGML 3D et 2D pour n'importe quel type de texte. C'est grâce à ce convertisseur simplifié qu'est assemblé le fichier scene.pov (le seul qui puisse être exploité directement par POV-Ray).

majore_points

Ce petit programme permet de majorer (ou d'ailleurs de minorer) les numéros des points définis dans un fichier .2db ou .3db. Il ne sert que très occasionnellement, notamment pour faciliter la description en SGML 3D d'un relief dont on a stocké en SGML 2D les courbes de niveau (si cela ne vous évoque rien, vous pouvez l'oublier).

modifini

Ce petit programme permet (notamment à un script) de modifier un fichier d'initialisation sans qu'une manipulation manuelle dans un éditeur de texte soit nécessaire.

obj_vers_3d

Ce convertisseur traduit un fichier .obj, conforme aux spécifications du format Videoscape (et généralement produit par Blender avec la commande « Export Videoscape » de son menu File) en fichier SGML 3D. Le fichier converti ne conserve pas les couleurs.

place_camera

Cette petite interface en mode texte est bien utile pour faciliter la définition d'une caméra virtuelle, d'autant que les paramétrages qu'elle définit (positionnement, cadrage, focale) peuvent être exploités par POV-Ray. Elle a recours à 3db_vers_persp (voir ci-dessus), et les images simplifiées qu'elle peut produire sont donc affectées de toutes les imperfections liées à la logique de ce convertisseur.

potier

Ce convertisseur transforme un profil SGML 2D unique (stocké en .2db) en objet SGML 3D ayant grossièrement l'aspect d'un objet fabriqué sur le tour d'un potier. Dans la plupart des cas, ce profil a été produit préalablement par bmp_vers_profil.

Attention : la couleur éventuellement conférée au profil dans le descriptif SGML 2D est éliminée lors de la conversion, et il faudra penser à compléter manuellement le fichier produit pour lui conférer une couleur.

reglages

Ce petit programme n'est pas voué à être appelé manuellement, mais est employé par certains scripts. Il permet de mettre à jour une série de fichiers après la modification du fichier d'initialisation reglages.ini.

texte_en_triangles (et texte_en_triangles.ini)

Ce convertisseur traduit une mention textuelle en un assemblage de triangles au format SGML 2D (en .2db). Le résultat n'est assurément pas esthétique, mais il est suffisamment lisible et peu gourmand en nombre de triangles. Après conversion (par 2db_vers_3d ou extrusion) et ajout manuel d'une couleur, cela permet de faire figurer la mention textuelle dans une scène 3D, par exemple pour réaliser des panneaux indicateurs.

A noter que ce programme doit être lancé depuis le sous-répertoire qui porte le même nom que lui (`texte_en_triangles`), car il a besoin de lire un grand nombre de fichiers secondaires, en particulier ceux décrivant le dessin des lettres.

visuppm.py

Ce petit script Python aussi rustique que pratique affiche à l'écran (sous Windows et Linux, mais malheureusement pas avec le Python installé en standard par Mac OS/X) des images au format PPM (Portable Pixel Map). Le programme place_camera a recours à lui pour accélérer l'affichage d'une image correspondant au cadrage en vigueur.

Chapitre 11

Ce qu'il faut savoir de POV-Ray

11.1 Un programme sans interface

En caricaturant à peine, on peut considérer POV-Ray lui-même comme un convertisseur... plus complexe mais pas très différent dans son principe des modestes utilitaires que nous diffusons sur le CD-ROM joint. Sa fonction consiste bel et bien à transformer un fichier en un autre : partant d'un fichier de texte (d'extension .pov) écrit selon la syntaxe de POV-Ray, il écrit un fichier graphique (généralement, d'extension .png) voué à être repris par un programme de retouche d'images.

Comme POV-Ray est un convertisseur, on ne s'étonnera pas qu'il n'ait pas plus besoin d'interface que n'importe lequel de nos utilitaires en ligne de commande. Comme les plus complexes d'entre eux, POV-Ray peut d'ailleurs avoir recours à la technique du paramétrage par fichier d'initialisation (il s'agit du fichier `povray.ini` que l'on trouve dans plusieurs sous-dossiers de `sgml_fichiers`).

Si POV-Ray n'a pas besoin d'interface, pourquoi lui en avoir donné une ? La question mérite vraiment d'être posée : la version de POV-Ray pour Linux fonctionne bel et bien en ligne de commande, exclusivement ; non seulement c'est possible, mais c'est une bénédiction pour ceux qui souhaitent automatiser son action, en particulier pour le calcul des milliers d'images d'une animation.

Seulement, tous les utilisateurs d'ordinateurs ne savent pas se servir d'une console... Dans l'univers Mac, en particulier, on a très longtemps cherché à se convaincre qu'une souris simplissime, à un seul bouton, constituait le *nec plus ultra* de l'ergonomie informatique. C'est à notre sens une aberration, mais elle a tellement imprégné l'esprit des applemaniaques qu'encore aujourd'hui, alors que Mac OS est devenu un Unix des plus classiques, l'immense majorité de ses utilisateurs ignore jusqu'à l'existence du programme Terminal – et POV-Ray pour Mac ne fait rien pour lutter contre cette ignorance : cette version de notre moteur

de rendu préféré n'est pas du tout vouée à être automatisée, et la façon normale de lancer une opération avec elle reste l'usage de la souris et des menus déroulants (ce qui peut d'ailleurs transformer en vraie corvée le travail de mise au point d'une animation avec POV-Ray pour Mac).

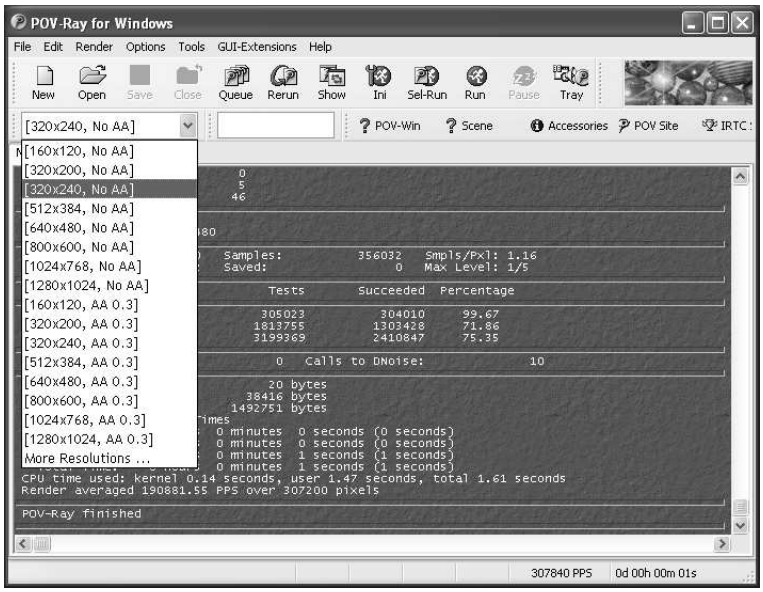


FIG. 11.1 – *La liste déroulante en haut à gauche de la fenêtre de POV-Ray pour Windows permet de choisir la taille de l'image de synthèse produite... à moins qu'un fichier d'initialisation* `povray.ini` *n'impose d'autres paramètres.*

Sous Windows, la philosophie générale n'est guère différente, mais enfin la tradition MS-DOS a quand même laissé quelques traces : si POV-Ray pour Windows trouve dans le répertoire d'un fichier .pov à traiter un fichier `povray.ini`, il prendra en compte ses paramétrages, y compris d'ailleurs s'ils contredisent ceux qu'affiche l'interface graphique (mieux vaut le savoir, car cela peut vous embrouiller quand vous travaillez avec les dossiers de `sgml_fichiers`). Il est même possible de lancer le rendu d'une image de synthèse en ligne de commande (et donc dans un batch) par la syntaxe suivante :

```
pvengine /render scene.pov /exit
```

ou plutôt, si l'on ne travaille pas dans le répertoire de POV-Ray :

```
"c :\Program Files\POV-Ray for Windows v3.6\bin\pvengine"
/render scene.pov /exit
```

Un peu longuet, non ? Mais enfin, ça marche...

En fait, l'interface graphique de POV-Ray pour Windows et Mac a pour principal intérêt de comporter un éditeur de lignes dédié à ce programme, et capable

d'afficher les instructions de son langage avec une **coloration syntaxique** adaptée : cela signifie que lorsque vous utilisez une telle interface pour lire un fichier `scene.pov` produit par le script `fabrique_tout`, les lignes de code n'apparaissent pas uniformément noires sur un fond blanc : les mots-clés de POV-Ray et les valeurs numériques sont affichées en d'autres couleurs – ce qui est plus joli, plus lisible, mais surtout peut aider à repérer un grand nombre d'erreurs de syntaxe si l'on se risque à modifier ce code manuellement. Cela étant, ne croyez pas que cela ait de quoi rendre jaloux les linuxiens avec leur POV-Ray sans interface : tous les éditeurs de texte courants sous Linux (vim, emacs, kwrite...) effectuent la coloration syntaxique et, en général, ils sont dotés d'un module pour POV-Ray (à défaut, le module consacré au code C++ ferait presque aussi bien l'affaire).

Par ailleurs, les interfaces graphiques de POV-Ray sont relativement pratiques pour paramétrer la taille et le format des images de synthèse produites. Sous Windows, le réglage de la taille s'effectue dans une petite liste déroulante, en haut à gauche de la fenêtre (fig. 11.1). Par défaut, le format de l'image de synthèse est PNG, ce qui convient parfaitement.

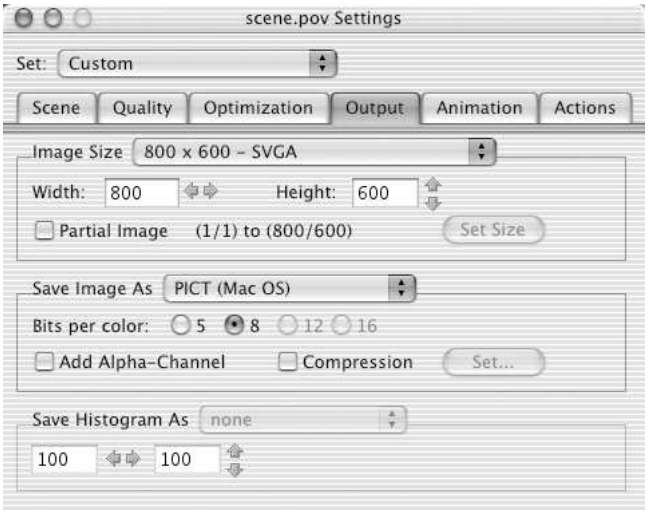


FIG. 11.2 – *La version Macintosh de POV-Ray ne produit pas directement d'images au format BMP, mais le format PICT est toujours lu par les logiciels de retouche pour Mac – qui peuvent d'ailleurs le traduire.*

Sur Mac, la taille de l'image est réglée par la manipulation suivante : menu « Edit », commande « Settings » (tout en bas du menu), onglet « Output », sous-menu « Image Size ». L'image produite est par défaut au format Pict, non standard mais reconnu par tous les programmes de retouche d'image tournant sur Mac, et qu'on peut de toute façon remplacer par PNG (fig. 11.2).

Que ce soit avec Mac ou Windows, l'image de synthèse sera a priori enregistrée dans le répertoire où se trouvait le fichier source (`source.pov`).

Quant à la syntaxe pour obtenir une image de synthèse avec POV-Ray pour Linux, c'est celle que vous pourrez lire dans le `Makefile` du dossier `08_scene_type`, c'est-à-dire tout simplement :

```
povray -Iscene.pov -Oimages/visu.png
```

Ca ne vous rappelle rien ? Le nom d'un programme, le nom d'un fichier à lire, le nom d'un fichier à écrire... Quant on vous disait que POV-Ray était un convertisseur ! Cela étant, amis linuxiens, prenez bien note de cette bizarrerie : les mentions `-I` et `-O` précèdent les noms des fichiers manipulés, et elles y sont accolées et non séparées par un caractère d'espacement (nous ignorons l'origine de cette bizarrerie, mais elle n'est nullement gênante).

Dans le `Makefile` de Linux, nous avons en outre fait suivre cette ligne de commande des mentions `>/dev/null 2>/dev/null`, classiques pour éviter de polluer l'affichage de la console par des messsages – presque toujours inutiles avec le script `fabrique_tout`. Mais si vous modifiez vous-même le code du fichier `scene.pov`, il peut être intéressant de supprimer ces mentions, afin de laisser POV-Ray vous signaler ses difficultés à vous comprendre...

Quels réglages dans le fichier povray.ini ?
Pour obtenir une image de synthèse avec la version pour Linux de POV-Ray, il faut impérativement qu'un fichier d'initialisation `povray.ini` se trouve dans le même répertoire que le fichier .pov qui décrit cette image. Avec la version pour Windows, ce n'est pas indispensable, mais si un tel fichier existe (et c'est notamment le cas dans notre dossier type `08_scene_type`, ses réglages seront pris en compte, quand bien même l'interface graphique paraîtrait les contredire.
Le fichier `povray.ini` que nous avons placé en `08_scene_type` est très laconique, et de surcroît vous n'avez a priori aucune raison de modifier ses réglages, en dehors des deux premiers : `Width` et `Height`, lesquels, comme vous l'aurez probablement compris si vous parlez anglais, indiquent les dimensions en pixels de l'image de synthèse produite. Si vous les modifiez, nous vous recommandons de garder les proportions classiques des images informatiques : 320 x 240, 640 x 480, 1024 x 768, etc.

11.2 Structure d'un fichier POV-Ray

11.2.1 Une syntaxe élémentaire pour utiliser POV-Ray

Au fil de sa déjà longue existence, POV-Ray est devenu un véritable langage de programmation dédié à l'image de synthèse. Ce langage se décline en de très nombreux dialectes adaptés au type d'image que l'on cherche à produire. Plus

l'image fait appel à des effets d'optique spectaculaires, plus la syntaxe se complique. Les fanatiques du photoréalisme peuvent consacrer des années entières à apprendre à la dominer par la lecture de la très riche documentation dont POV-Ray est doté.

Symétriquement, et c'est très heureux, quand on se borne comme l'auteur à chercher à afficher des triangles de couleur en perspective photographique et avec des ombres portées, la syntaxe de POV-Ray est relativement simple. Notre façon de l'écrire n'est certes pas celle qui mène aux résultats les plus spectaculaires. Mais elle permet d'obtenir des images de synthèse lisibles de façon particulièrement rapide, même dans le cas où la scène représentée comporte un très grand nombre de triangles. L'auteur estime donc que c'est un bon choix lorsqu'on travaille sur l'ordinateur de M. Tout-le-monde.

Vous n'êtes certes pas obligé de comprendre la syntaxe que nous préconisons, et d'autant moins que nos convertisseurs ont précisément pour fonction de vous dispenser de l'écrire vous-même ! Mais vous avez aussi le droit de vouloir consacrer plus d'énergie que l'auteur à la recherche du photoréalisme. Or, dans bien des cas, une légère modification du code POV-Ray produit par nos convertisseurs peut suffire à augmenter considérablement la richesse graphique d'une image. Voici donc quelques explications qui vous faciliteront le travail si vous voulez repérer les endroits où une modification du code peut mener à des effets particuliers.

L'ordre que nous suivrons pour cette explication est bien sûr celui du code POV-Ray présent dans chaque fichier `scene.pov` produit par notre script `fabrique_tout`.

11.2.2 Paramètres généraux

```
global_settings {
    assumed_gamma 1.0
    ambient_light rgb <1.0, 1.0, 1.0> * 4.0
}
```

Les deux premières instructions que l'on trouve dans n'importe quel fichier `scene.pov` sont réunies par la syntaxe de POV-Ray sous le nom `global_settings`, mais c'est leur seul point commun et ces deux commandes n'ont pas grand rapport logique.

Le **gamma** est un paramètre mathématique très utilisé par les logiciels de retouche d'image pour définir le degré d'éclaircissement que l'on veut faire subir à une photo sous-exposée, et POV-Ray l'utilise pour des raisons similaires : il s'agit de permettre de travailler avec un écran d'ordinateur de mauvaise qualité (comme la plupart des écrans de bureautique), qui affiche toutes les images de façon trop sombre. Avec un paramètre `assumed_gamma` à 1.0, POV-Ray n'effectuera aucun éclaircissement de l'image. En revanche, avec des valeurs de 1.2, ou 1.4, à la rigueur 1.8, POV-Ray éclaircira sensiblement toutes les images de synthèse qu'il

produira. Attention : ce paramétrage a pour fonction de corriger les défauts de votre moniteur, non de vous dispenser de réfléchir au paramétrage correct de vos sources de lumière. Si vous avez besoin de pousser ce paramètre au-delà de 2.0 pour obtenir des images lisibles, c'est le signe que les sources de lumière définies dans le fichier POV-Ray sont trop faibles compte tenu des dimensions de la scène (paramétrées comme des chandelles alors qu'il faudrait plutôt qu'elles simulent des lampes de 500 watts...) – ou encore cela signifie que votre moniteur est bon à jeter ; mais dans ce cas, vous n'avez sûrement pas eu besoin de POV-Ray pour vous en apercevoir !

Le paramètre `ambient_light` exprime une lumière fictive, venant de façon uniforme de toutes les directions. Avantage : grâce à ce paramètre, la scène offre une lisibilité minimale sous tous les angles, même très loin des sources de lumière, et d'ailleurs même si vous n'en définissez aucune. Inconvénient : ce paramètre atténue les contrastes (si la scène n'est éclairée que par lui, tous les objets apparaissent comme des surfaces planes). Ce paramètre est suivi d'une mention définissant : premièrement, la couleur de la lumière (en l'occurrence, `rgb <1.0, 1.0, 1.0>`, qui désigne une lumière blanche selon le modèle *red, green, blue*) ; secondement, son intensité (`* 4.0`).

11.2.3 Couleur de fond

```
background { color rgb <0.0, 1.0, 1.0> }
```

Le paramètre `background` définit la couleur de fond de l'image... celle du ciel, en somme. Cette couleur apparaîtra comme un aplat sur toutes les zones ne comportant aucun triangle. En l'occurrence, pour évoquer un ciel bleu, nous avons choisi du cyan pur (`rgb <0.0, 1.0, 1.0>`) : pas de composante rouge, composantes verte et bleue au maximum.

11.2.4 Sources de lumière

```
light_source {
    <66.0, 200.0, -55.0>
    color rgb <1.0, 1.0, 1.0> * 0.5
    // shadowless
}
```

Les paramètres `light_source`, que l'on peut multiplier à l'envi, décrivent des sources de lumière ponctuelles : elles proviennent d'un point précis de l'espace, dont on indique les coordonnées XYZ (en l'occurrence, `<66.0, 200.0, -55.0>`) – ou plutôt, pour un Européen, les coordonnées XZY : c'est la deuxième coordonnée citée, et non la troisième, qui indique à quelle hauteur se situe la source de lumière. Notez qu'au-delà d'une certaine distance, POV-Ray atténue

graduellement l'effet de ces sources d'éclairage. Il s'agit en l'espèce une nouvelle fois d'une lumière blanche, ce qu'exprime le paramètre `rgb <1.0, 1.0, 1.0>`. Son intensité a été minorée par le paramètre `* 0.5`, mais on aurait pu au contraire l'augmenter avec une valeur supérieure à 1.0.

La mention `shadowless` dans le code d'une source de lumière signifie que cette lumière traversera (fictivement) tous les objets qu'elle rencontrera et ne projettera donc pas d'ombres portées... sauf qu'en l'espèce, cette mention a été neutralisée par la double barre de fraction `//` qui la précède ; cette source de lumière projettera donc bien des ombres. On peut trouver saugrenu de faire ainsi figurer dans un code informatique une commande que l'on prend grand soin de neutraliser, mais c'est une pratique courante en programmation. La double barre de fraction est d'ailleurs une convention de notation héritée du langage C++ : toute fin de ligne précédée de `//` est censée être nulle et non avenue. Neutraliser ainsi un peu de code informatique sans l'effacer offre au programmeur la possibilité de changer d'avis sans avoir à se remémorer une syntaxe obscure. En l'occurrence, il suffirait d'effacer la double barre de fraction pour que la lumière provenant de cette source redevienne *shadowless*, donc cesse de projeter des ombres.

La scène type du sous-dossier `08_scene_type` comporte cinq sources de lumière, et une seule d'entre elles (la plus haut placée) projette des ombres. Notez que vous n'avez pas du tout intérêt à essayer de rendre plus régulier et symétrique le positionnement de ces sources de lumière : c'est précisément l'imprécision de leur positionnement qui évite que tous les côtés d'un objet soient affichés avec exactement la même nuance de couleur – ce qui empêcherait de les distinguer.

11.2.5 Caméra

```
camera {
    perspective
    location <0.500000, 1.600000, -5.000000>
    look_at <0.000000, 0.500000, 0.000000>
    angle 45.000000
}
```

Une caméra de type `perspective` comme celle-ci correspond à un objectif photographique de modèle banal. Soit dit en passant, POV-Ray en connaît quelques autres fort spectaculaires, notamment pour élaborer des vues à 360 °. On l'aura compris, le paramètre `location` indique la position de la caméra (là encore, les coordonnées sont citées dans l'ordre XZY), et le paramètre `look_at` désigne un point vers lequel la caméra s'oriente ; bien évidemment, c'est la direction de ce point qui compte, et peu importe qu'il désigne ou non le centre d'un objet présent sur la scène ; la distance à laquelle se situe ce point est elle aussi indifférente. A noter qu'une caméra de ce type ne subit jamais de dévers, et aucun

réglage n'est nécessaire pour que l'horizon apparaisse toujours parfaitement horizontal. Enfin, le paramètre `angle` permet de régler la focale ; la valeur indiquée est exprimée en degrés.

11.2.6 Maillages

Les lignes que nous venons de commenter étant passées, nous entrons dans la partie de loin la plus abondante du code de POV-Ray (elle représente couramment 99 % du fichier, avec des dizaines de milliers de lignes). Cette partie décrit les maillages et même plus précisément les triangles qui constituent la scène. Dans un fichier `scene.pov`, le code de ce type a été produit par notre convertisseur `3db_vers_pov` (lui-même lancé automatiquement par le script `fabrique_tout` ou le `Makefile` d'Unix) et constitue une traduction extrêmement fidèle du code SGML 3D dépendant hiérarchiquement du fichier `source.3d`.

Les triangles sont regroupés en maillages (*mesh*), de façon à ne pas répéter inutilement le code relatif à la couleur ou à la texture d'une multitude de triangles.

Maillages de couleur

```
mesh {
    triangle {
        <-0.583386, 0.001000, 4.625541>,
        <-0.562114, 0.001000, 4.673757>,
        <-0.574877, 0.001000, 4.670921>
    }
    // (eventuellement, d'autres triangles)
    texture {
        pigment { color rgb <0.166, 0.164, 0.466> }
    }
}
```

Les trois lignes pleines de chiffres qui suivent le mot-clé `triangle` indiquent tout simplement les coordonnées de ses sommets. La seule difficulté consiste à ne pas oublier que ces données sont énumérées dans l'ordre XZY et non XYZ.

Quant à la définition de la couleur affectant tout le maillage (du type `rgb <0.166, 0.166, 0.466>`), elle s'opère selon le classique et incompréhensible modèle colorimétrique rouge-vert-bleu de la synthèse additive (les trois valeurs énumérées sont bien sûr citées dans cet ordre). Sachez simplement que POV-Ray exprime toujours ces valeurs entre 0.0 (aucune lumière) et 1.0 (intensité maximale de la couleur), contrairement aux logiciels de retouche d'image qui travaillent généralement sur une fourchette comprise entre 0 et 255 (le principe est

cependant le même, et il vous suffirait de diviser par 255.0 les valeurs indiquées par un logiciel de retouche pour obtenir des paramètres que POV-Ray saurait exploiter). Notez que quand Blender définit les couleurs selon le modèle RGB, il utilise lui aussi des valeurs comprises entre 0.0 et 1.0 ; vous pouvez donc vous aider de Blender pour trouver les valeurs correspondant à une nuance précise.

Maillages texturés

Focalisé sur la 3D maillée *stricto sensu*, ce livre comporte cependant en annexe quelques explications sur la question complexe des textures – c'est-à-dire, pour simplifier, l'affichage de motifs photographiques sur des triangles en 3D. Vous pourrez vous y reporter pour mieux comprendre ce qui va suivre. Dans l'immédiat, sachez que notre convertisseur `3db_vers_pov` est capable d'écrire du code POV-Ray décrivant des triangles texturés selon la syntaxe décrite ci-dessous.

```
mesh {
    triangle {
        <0.007813, 0.496094, 0.000000>,
        <0.460938, 0.824219, 0.000000>,
        <-0.484375, 0.824219, 0.000000>
        uv_vectors
        <0.511719, 0.488235>,
        <0.964844, 0.982353>,
        <0.019531, 0.982353>
    }
    // (eventuellement, d'autres triangles)
    texture {
        uv_mapping pigment {
            image_map {
                jpeg "texture.jpg"
                map_type 0
                interpolate 0
            }
        }
    }
}
```

Nous retrouvons au début la syntaxe de définition des coordonnées des sommets d'un triangle, mais elle est cette fois suivie du mot-clé `uv_vectors` et de trois points définis sur la texture dans un système de coordonnées U et V (reportez-vous aux annexes de ce livre pour plus d'explications). Quant au code abondant qui termine ce maillage, la seule chose qu'il faut y remarquer est la présence du nom d'un fichier graphique décrivant la texture – en l'occurrence,

`"texture.jpg"`. Ce fichier sera bien sûr recherché dans le même répertoire que celui où se trouve le fichier .pov qui l'appelle, à moins que vous ne spécifiiez un chemin d'accès. POV-Ray peut travailler avec les formats graphiques Jpeg (JPG) et PNG.

11.3 Assemblage automatique d'un fichier POV-Ray

Nous venons de le voir, un fichier de 3D maillée de POV-Ray comporte cinq parties de longueur très inégale : paramètres généraux, couleur de fond, sources de lumière, caméra et enfin maillages. Le processus automatisé par notre script `fabrique_tout` rédige le fichier `scene.pov` en rassemblant le contenu de cinq autres fichiers de texte, stockés dans le sous-répertoire `pov` d'une scène et correspondant à ces cinq fractions de code : `entete.pov` pour les paramètres généraux, `bground.pov` pour la couleur de fond, `light.pov` pour les sources de lumière, `cam.pov` pour la caméra et `archi.pov` pour les maillages proprement dits. Les trois premiers ne sont pas modifiés par le processus – et si vous voulez les adapter, par exemple pour changer le nombre et l'emplacement des sources de lumière, vous devrez le faire à la main, directement dans la syntaxe de POV-Ray. Les deux derniers, en revanche, sont automatiquement modifiés par le processus mis en jeu par le script `fabrique_tout` : `archi.pov` n'est rien d'autre que la traduction de la hiérarchie de code SGML 3D dépendant du fichier `source.3d`; quant à `cam.pov`, il est automatiquement écrit (par le petit programme `reglages`) en fonction des indications du fichier `reglages.ini`, qu'il est très facile de modifier manuellement. Cela étant, la mise en place d'une caméra est quand même beaucoup plus facile à effectuer avec des repères visuels que par la saisie de coordonnées XYZ, et nous allons voir dans la section suivante qu'un petit utilitaire a été prévu pour permettre un tel réglage.

Sachez enfin que les cinq fichiers dont nous avons parlé sont réunis (par notre convertisseur `gigogne`, mais cela est automatisé dans le script `fabrique_tout`) en fonction des indications d'un autre fichier contenu dans le même sous-répertoire `pov` : `source.pov`, lui-même écrit selon une syntaxe SGML qui ne vous causera aucune difficulté de compréhension. Rien ne vous empêche de modifier ce fichier si vous voulez intercaler dans le code POV-Ray des instructions spécifiques.

11.4 Faciliter le placement de la caméra

Pour régler le cadrage de l'image de synthèse produite par POV-Ray, il est envisageable de modifier manuellement les paramètres du fichier `reglages.ini` (notamment le paramètre `angle`, qui correspond au facteur de zoom : 45.0 dé-

signe une focale normale, 60.0 donne une vision grand-angulaire, 25.0 correspond à l'usage d'un téléobjectif) ; ensuite, l'appel du script `fabrique_tout` opérera toutes les modifications nécessaires. Mais pour des réglages plus subtils, on peut lancer par la commande `../bin/place_camera` un petit programme qui permet un ajustement plus fin avec une interface en ligne de commande très rudimentaire, mais quand même tout à fait utilisable... et par ailleurs très facile à programmer.

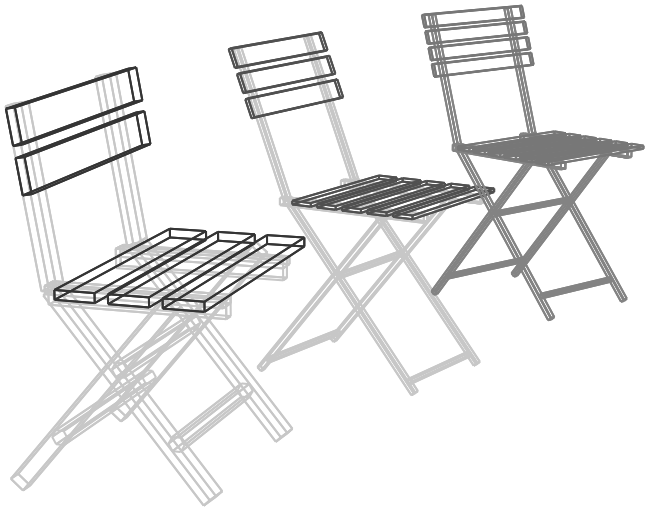


FIG. 11.3 – *La visualisation en perspective et en « fil de fer » obtenue avec le programme* `place_camera` *peut être employée pour définir le cadrage de l'image de POV-Ray.*

Les commandes de `place_camera` sont fort nombreuses mais très faciles à retenir :

– `image` fabrique (mais n'affiche pas !) un fichier bitmap en 16 couleurs (nommé `visu.bmp` et stocké dans le sous-répertoire `images`) correspondant aux paramètres actuels de visualisation en perspective ; comme cette commande fait appel à des utilitaires extérieurs, elle est sensiblement plus lente que les autres ;

– `visu` affiche (mais ne fabrique pas !) le fichier `visu.bmp` du sous-répertoire `images` – si toutefois les routines graphiques de Python sont présentes et accessibles, ce qui n'est notamment pas le cas dans la configuration de base du Python d'un Mac ; cela dit, il est toujours possible de lire ce fichier avec un logiciel de retouche d'images, c'est seulement un peu plus long et fastidieux ; si la configuration de votre machine permet que la commande `visu` fonctionne, vous devez

bien sûr enchaîner les deux commandes : `image`, puis `visu` ; notez que la fenêtre de visualisation qui sera alors affichée devra impérativement être refermée avant que vous puissiez faire suivre une autre commande (ne vous contentez pas de la faire passer à l'arrière-plan) ;

– `exporte` modifie le fichier `reglages.ini` conformément à la représentation que l'on a déterminée ; en d'autres termes, cette commande enregistre le résultat, et il suffira de relancer le script `fabrique_tout` après avoir quitté `place_camera` pour provoquer la fabrication par POV-Ray de l'image de synthèse correspondant au cadrage que l'on vient de régler (fig. 11.3) ;

– `quitte` permet de sortir de `place_camera` (attention : si `quitte` n'est pas précédé d'`exporte`, les réglages seront perdus, et cela n'est pas signalé !) ;

– `gauche` oriente la caméra plus à gauche ; `poilgauche` oriente la caméra légèrement plus à gauche ;

– `droite` oriente la caméra plus à droite ; `poildroite` oriente la caméra légèrement plus à droite ;

– `nord, sud, est, ouest, ne, no, se, so` orientent la caméra dans la direction correspondante (attention : une direction nord-sud est parallèle à l'axe des Y et non des X, et c'est une direction est-ouest qui est parallèle à l'axe des X) ;

– `plushaut` oriente la caméra vers le haut ; `poilhaut` oriente la caméra légèrement vers le haut ;

– `plusbas` oriente la caméra vers le bas ; `poilbas` oriente la caméra légèrement vers le bas ;

– `horizontal` remet la caméra à l'horizontale ;

– `zoom` augmente la focale ; `poilzoom` augmente légèrement la focale ;

– `mooz` diminue la focale ; `poilmooz` diminue légèrement la focale ;

– `angle45` reprend une focale moyenne (correspondant à une vision à 45 degrés) ;

– `plusvite` double la valeur d'un déplacement éventuel ;

– `moinsvite` divise par deux la valeur d'un déplacement éventuel ;

– `vitesseun` redonne au déplacement éventuel la valeur d'un mètre ;

– `avance` déplace la caméra vers l'avant à l'horizontale (la valeur du déplacement étant déterminée par ailleurs) ;

– `serapproche` déplace la caméra dans la direction visée, pas forcément à l'horizontale ;

– `recule` déplace la caméra vers l'arrière à l'horizontale ;
– `seloigne` déplace la caméra à l'opposé de la direction visée, pas forcément à l'horizontale ;
– `monte` déplace la caméra vers le haut, comme dans un ascenseur, sans en changer l'orientation (on fait généralement suivre de la commande `plusbas`) ;
– `descend` déplace la caméra vers le bas sans en changer l'orientation (on fait généralement suivre de la commande `plushaut`) ;
– `crabegauche` effectue un déplacement latéral vers la gauche, sans changer l'orientation de la caméra (on fait souvent suivre par la commande `droite`) ;
– `crabedroite` effectue un déplacement latéral vers la droite, sans changer l'orientation de la caméra (on fait souvent suivre par la commande `gauche`) ;
– enfin, `pointo` replace la caméra au point O (c'est-à-dire, entre autres, au niveau du sol ! il faut faire suivre de la commande `monte`) ;
– tous les autres ordres sont ignorés (en d'autres termes, les fautes de frappe restent sans conséquence).

La bonne compréhension d'un ordre par `place_camera` est toujours signalée sur la console par une brève réponse du programme (généralement, cette réponse indique la valeur actuelle d'un ou plusieurs paramètres que l'on vient de modifier). Une absence de réponse est le signe que le dernier ordre passé était incompréhensible, le plus souvent à cause d'une faute de frappe.

Si l'on a exporté (commande `exporte`) les résultats, il suffira de quitter `place_camera` (commande `quitte`), puis de lancer le script `fabrique_tout`, pour fabriquer un fichier de POV-Ray prenant exactement en compte le cadrage choisi.

L'utilisation de `place_camera` présente un léger inconvénient : comme cet utilitaire modifie `reglages.ini` et que ce fichier est aussi pris en compte pour la production des vues de face, de dessus et de profil et de la perspective axonométrique (fabriquées par le script `fabrique_tout` dans le sous-répertoire `images`), ces quatre dernières vues ne correspondent souvent plus à grand-chose d'utile après l'emploi de `place_camera`.

Mais l'utilisation de `place_camera` présente l'énorme avantage de permettre d'examiner la scène sous plusieurs angles très rapidement. En fait, dans l'immense majorité des cas, au lieu d'avoir recours au script `fabrique_tout` à la moindre modification de la scène, on peut se contenter d'enchaîner le script `fabrscen` (`bat\fabrscen` sous Windows, `sh sh/fabrscen.sh` sous Linux et Mac OS/X) et de relancer une visualisation avec `place_camera`. Cela fait gagner un temps fou.

Remarque
Les images en perspective produites avec `place_camera` suivent *chronologiquement* le déroulé de la hiérarchie de fichiers SGML 3D : les traits représentant le dernier objet appelé chronologiquement seront toujours tracés par-dessus les traits représentant les autres objets, sans que cela ait le moindre rapport avec la perspective (il ne s'agit pas d'une représentation en faces cachées). On aura donc parfois l'impression qu'un objet éloigné se trouve au premier plan, s'il a été appelé parmi les derniers dans la hiérarchie SGML 3D. Il suffira de relancer le script `fabrique_tout` pour se rendre compte que cet inconvénient n'a aucune influence sur l'image de synthèse produite au final.

Chapitre 12

Automatiser la 3D avec Python

12.1 Un exercice à la portée de l'amateur

Avec un logiciel adapté comme Blender, un amateur peut assurément élaborer de jolies images fixes rien qu'avec des manipulations à la souris. Cependant, nous l'avons vu dans les chapitres qui précèdent, dès qu'il est question de minutie et d'exactitude, les manipulations du code SGML 3D avec un bon éditeur de lignes peuvent se révéler beaucoup plus efficaces.

C'est ici que l'on touche à la vraie magie de l'informatique : l'élaboration de ce code SGML 3D peut très facilement être automatisée de A à Z – et en fait, avec n'importe quel langage de programmation, et même avec des notions absolument élémentaires de n'importe quel langage de programmation.

Non, l'automatisation intégrale de la 3D n'est pas du tout hors de la portée d'un simple amateur. L'auteur est même convaincu que l'amateur de 3D est beaucoup mieux armé que n'importe quel informaticien débutant pour s'attaquer à l'apprentissage de la programmation, parce qu'il dispose sur les autres de deux avantages inappréciables.

Premièrement, il lui est très facile de laisser divaguer son imagination pour choisir et moduler la difficulté des exercices qu'il s'impose (aujourd'hui, je vais créer une échelle ; un escalier monumental ; une piste d'atterrissage ; un château fort avec des créneaux et quatre tours ; un pont à haubans ; une salle de cinéma ; une fusée ; une gare de triage...). Deuxièmement, il peut juger de la qualité de son travail d'un seul coup d'œil sur une image de synthèse – au lieu de devoir se torturer les méninges à juger de la validité d'un algorithme, ce qui est le supplice habituel que doivent s'infliger les étudiants en informatique.

Si les triangles s'emboîtent mal, cela se voit du premier coup d'œil. Si l'objet est mal proportionné, cela se voit du premier coup d'œil. Mais surtout, si la géométrie de l'objet qu'on vient de créer est parfaite, on peut consacrer autant de

temps qu'on le veut à le regarder sous tous les angles pour s'en assurer et s'en réjouir. On peut même le montrer à tous ses commensaux – et là, encore, chacun d'entre eux n'a besoin que d'un coup d'œil pour se convaincre de la perfection géométrique de l'objet, même s'il a fallu à l'amateur une semaine de travail pour en venir à bout.

La 3D est peut-être le seul exercice informatique qui puisse facilement susciter des commentaires admiratifs : il n'y a aucun besoin de comprendre la difficulté de l'exercice pour juger de la qualité du résultat. Créer de la 3D par programmation est donc un exercice intellectuel extrêmement valorisant, qui procure très vite au programmeur amateur un orgueil dont il ressent, *voit* et peut même *montrer* la légitimité. Et c'est avant tout pour cette raison que l'apprentissage de la programmation par un amateur de 3D est beaucoup plus facile que pour n'importe qui d'autre.

Au reste, lorsque l'on dispose de cette récompense pour se payer de ses efforts, on se rend vite compte que la programmation n'est pas un art difficile. Il est certes minutieux et gourmand en temps, mais en compensation logique, rigoureux, cartésien, et donc d'apprentissage facile. En outre, il est extraordinairement puissant et efficace, et ce, dès le premier instant. Son seul véritable défaut, qui rebute tout le monde, est l'ennui qu'il suscite ordinairement. Et justement : ce défaut n'existe pas pour l'amateur de 3D.

12.2 Aussi simple que d'imprimer

Il n'y a pas si longtemps, il existait une méthode d'apprentissage du dessin dont le slogan était : « Si vous savez écrire, vous savez dessiner. » En effet, du moins selon son promoteur, la principale difficulté du dessin consiste à apprendre à tenir correctement un crayon – ce qu'a priori on sait faire si l'on sait écrire. Eh bien, d'une façon un peu similaire, on peut presque dire : « Si vous savez imprimer, vous savez créer des objets en 3D. » Vous allez comprendre pourquoi.

Quand nous parlons d'« imprimer », c'est là encore au sens où on l'entendait du temps des vieilles imprimantes matricielles utilisant de longues bandes de papier perforé en accordéon – dont nous avons déjà parlé pour expliquer la logique des consoles et de la ligne de commande. Par « imprimer », il faut donc comprendre « faire cracher une ligne de texte à l'imprimante » – ou afficher une ligne de texte sur la fenêtre de la console, ce qui revient au même. Si vous savez faire cracher une ligne de texte à une vieille imprimante, ou afficher une ligne de texte sur votre console, eh bien vous maîtrisez déjà l'essentiel.

Et si vous ne savez pas ? C'est le moment d'apprendre. En Python, cela se traduit par le mot magique `print` (jamais vous ne l'auriez imaginé). Sans rechercher le moins du monde l'originalité – cet exemple est un poncif de tous les

manuels de programmation –, nous allons donc vous demander d'écrire le programme Python suivant, composé en tout et pour tout de cette ligne :

```
print 'Bonjour tout le monde !'
```

Ouvrez donc un éditeur de lignes, saisissez cette ligne de code informatique (mais oui : c'en est bien une) et enregistrez-la sous le nom `helloworld.py` (vous pourriez en choisir un autre, mais la tradition informatique a imposé celui-là, et l'écriture d'un « Hello world » est à la programmation ce que le baptême est au christianisme : une cérémonie d'une grande solennité, extrêmement codifiée, et qui marque votre entrée définitive dans la communauté ; recueillez-vous : dans deux secondes, vous serez devenu à tout jamais un informaticien).

Avec la console, placez-vous ensuite dans le répertoire où vous avez enregistré ce fichier, et tapez

```
python helloworld.py
```

En d'autres termes, vous demandez à Python d'exécuter ce merveilleux programme. Ce qu'il fait aussitôt : juste sous la ligne de commande que vous venez de valider, il fait apparaître la mention « Bonjour tout le monde ! » Vous ne voyez sans doute pas encore à quel point cela est merveilleux. Patience ! Vous n'en êtes plus loin.

Vous allez à présent taper la même ligne de commande, mais en la faisant suivre d'une mention mystérieuse (le signe cabalistique `>` qui figure au milieu signifie habituellement « supérieur à », et sur la plupart des claviers, on le saisit en majuscules avec la touche située tout à fait en bas à gauche de la partie alphabétique) :

```
python helloworld.py > bonjour.txt
```

A votre grand désappointement sans doute, la mention « Bonjour tout le monde ! » n'apparaît plus à l'écran. C'est normal ! La mention que vous avez ajoutée en fin de ligne constitue en effet ce que l'on appelle une **redirection** : tout ce que le programme `helloworld.py` était capable d'écrire (imprimer) à l'écran a été *redirigé* dans un petit fichier de texte : `bonjour.txt`. Ouvrez ce fichier avec un éditeur de lignes : ô merveille, vous pourrez y lire une ligne qui dit « Bonjour tout le monde ! »...

Tout ça pour ça ? Eh oui. Mais réjouissez-vous : vous venez d'apprendre le secret qui vous ouvre la voie de l'automatisation de la 3D – ainsi d'ailleurs que de tout ce que l'informatique sait exprimer par des fichiers de texte... c'est-à-dire énormément de choses, vraiment. Tout ce qu'un programme (en l'occurrence un script Python) est capable d'afficher sur l'écran d'une console peut être enregistré dans un fichier de texte par cette manipulation toute simple : ajouter le signe `>` et le nom d'un fichier de texte à créer au bout de la ligne de commande avec laquelle on lance le programme.

En d'autres termes : si vous êtes capable d'écrire un script Python qui affiche à l'écran du code SGML 3D (et vous découvrirez que c'est très facile), vous pourrez

sans la moindre difficulté enregistrer ce résultat dans un fichier de texte d'extension .3d. Vous venez de le faire avec un texte d'une seule ligne, mais ce serait à peine plus difficile avec un fichier SGML 3D comportant des centaines de milliers de lignes définissant autant de triangles ! Dès lors, ce sera un jeu d'enfant de demander à nos convertisseurs de traduire cela au format .obj de Videoscape (que Blender peut lire) ou au format de POV-Ray. Nous ne vous avions pas menti : si vous savez imprimer des lignes sur l'écran noir de votre console, vous en savez assez pour automatiser la 3D. Et ce dès maintenant, pour ainsi dire.

Et comme si ce que nous venons d'apprendre n'était pas déjà assez merveilleux, prenez conscience que cette méthode, qui fonctionne pour produire du code SGML 3D, est évidemment applicable pour produire n'importe quel autre fichier de texte, par exemple : un fichier d'initialisation (.ini) ; un fichier de Videoscape (.obj), que Blender peut lire ; un fichier de POV-Ray (.pov) ; un batch (.bat) ou un script shell (.sh), contenant s'il le faut des dizaines de milliers de commandes minutieuses, modifiant des dizaines de milliers de fois le même fichier d'initialisation, lançant des dizaines de milliers de fois des convertisseurs ; régal des régals... vous pouvez même employer un script Python pour rédiger... un autre script Python (.py). Bref, si vous savez « imprimer » (en fait, écrire des lignes de texte), vous avez accès à l'essentiel des capacités d'automatisation dont dispose l'informatique. Ca vaut vraiment la peine d'apprendre !

12.3 Quelques scripts pour découvrir Python

12.3.1 Avant-propos

Evidemment, pour tirer le meilleur parti de Python, il faut en savoir un peu plus long que ce seul mot-clé `print`. Un peu plus, oui ; mais pas tant que ça. Un manuel exhaustif de Python peut comporter dans les huit cents pages... mais ce dont l'amateur de 3D a réellement besoin (itérations, boucles, formatage des données, traitements conditionnels, routines) est toujours traité au grand maximum dans les cinquante premières. C'est bel et bien à votre portée. Si vous vous prenez au jeu, il vous suffira de quelques heures pour commencer à vous amuser, de quelques semaines pour réaliser des scènes très abondantes avec des détails très minutieux... et de quelques mois pour acquérir un savoir-faire au-delà duquel la 3D maillée ne vous demandera jamais d'aller.

Ce livre ne prétend pas remplacer un manuel d'initiation à Python. Toutefois, pour vous mettre le pied à l'étrier, nous avons rédigé quelques scripts Python aussi laconiques que possible, et classés par ordre de difficulté croissante. Vous pouvez déjà les lancer pour vous faire une idée de leur vitesse d'exécution. Vous pouvez aussi les étudier, surtout si vous savez déjà programmer dans un autre langage,

auquel cas vous en comprendrez probablement la logique à demi-mot (cela dit, leur style n'a rien d'impérissable, et un vrai pythonien le trouverait sans doute très médiocre).

Mais surtout, dès que vous aurez acquis quelques bases en Python par la lecture d'un ouvrage d'initiation ou d'un des nombreux tutoriels Python disponibles sur le web, vous pourrez sans vergogne plagier nos scripts-types pour construire vos propres objets. Non seulement il n'y a aucune honte à le faire, mais c'est la seule bonne façon de travailler ! Peut-être l'ignorez-vous encore, mais 95 % au moins de n'importe quel programme professionnel est constitué de code informatique qui a déjà servi à d'autres usages. En outre, le plus souvent, l'équipe qui revendiquera la paternité du programme quand il sera terminé n'est pour rien dans l'écriture de ces 95 %. Le principe même de la civilisation informatique en général (et du logiciel libre en particulier) est de permettre aux nains de monter sur les épaules des géants – ou plus modestement dans notre cas, de permettre au débutant que vous êtes (peut-être) de monter sur les épaules de l'amateur un peu expérimenté qu'est l'auteur.

Vous trouverez ces scripts-types dans le sous-dossier `12_cone_en_python` du dossier `sgml_fichiers` de notre CD-ROM. Tous ces scripts, en effet, ont pour seule et unique fonction de décrire un cône au format SGML 3D. Cela peut paraître d'un intérêt bien limité... mais cet objet simplissime est un bon prétexte pour introduire graduellement des concepts de Python de complexité croissante.

12.3.2 A quoi ressemble un cône en SGML 3D

Pour comprendre le travail effectué par nos scripts, il faut commencer par se faire une idée du code SGML 3D qu'ils doivent produire. Il s'agit de définir un cône de rayon 1.0 et de hauteur 1.0, dont la base est centrée sur le point O (intersection des axes X, Y et Z). Nous avons arbitrairement décidé que le pourtour de ce cône ne serait constitué que de douze facettes triangulaires. Fort logiquement, il va donc nous falloir définir quatorze points : le premier, numéroté 1, sera au centre de la surface de base ; les douze suivants, disposés en cercle autour de lui (avec un écart de 360°/12 = 30°), seront numérotés de 2 à 13 ; enfin, le dernier, numéroté 14, constituera la pointe du cône.

Les lignes de code SGML 3D décrivant tout cela auront donc à peu près cet aspect :

```
<points3D>
1, 0.0, 0.0, 0.0
2, 1.0, 0.0, 0.0
3, 0.866, 0.5, 0.0
4, 0.5, 0.866, 0.0
5, 0.0, 1.0, 0.0
```

```
6, -0.5, 0.866, 0.0
7, -0.866, 0.5, 0.0
8, -1.0, 0.0, 0.0
9, -0.866,-0.5, 0.0
10,-0.5, -0.866, 0.0
11,-0.0, -1.0, 0.0
12, 0.5, -0.866, 0.0
13, 0.866,-0.5, 0.0
14, 0.0, 0.0, 1.0
</points3D>
```

Ceux qui ont un peu oublié leurs cours de géométrie se demanderont sans doute d'où sortent ces valeurs, et en particulier ce mystérieux 0.866 que l'on rencontre à tout bout de champ. Ce n'est pas très compliqué. Les points 2 à 13 découpent les 360 degrés de la circonférence de base par portions de 360°/12 = 30°. Le point 2 est sur une direction de 0° (0 x 30°), le point 3 de 30° (1 x 30°), le point 4 de 2 x 30° = 60°, le point 5 de 3 x 30° = 90°, et ainsi de suite. Pour chacune de ces valeurs angulaires, les coordonnées X et Y correspondent respectivement au cosinus et au sinus de l'angle – comme n'importe quelle calculette scientifique se fera une joie de vous le confirmer. Par exemple, pour le point 3, qui est situé sur un angle de 30°, la coordonnée X vaut cos(30°) = 0.866 (plus exactement 0.866025404), et la coordonnée Y sin(30°) = 0.5.

Ce n'est pas uniquement par cruauté mentale que nous vous infligeons ce douloureux rappel d'un indigeste cours de mathématiques (si cela peut vous rassurer, l'auteur lui-même n'a jamais eu la moyenne sur ses copies de géométrie !). C'est surtout qu'en 3D informatisée, il est bien utile de connaître cette logique chaque fois que l'on veut travailler sur n'importe quoi de rond... et cela se produit vraiment souvent (tellement souvent que l'auteur, pourtant nul en géométrie à sa sortie du collège, a fini par savoir tout ça par cœur... car c'est vraiment beaucoup plus facile à apprendre quand ça sert à produire de jolies images).

Fort heureusement, Python est capable d'effectuer tout seul ces fastidieux calculs de sinus et de cosinus (du moins quand on lui a fait avaler ses routines mathématiques grâce à une ligne `from math import *`, à placer au tout début du script). La seule chose qui complique un peu l'exercice est que Python, comme d'ailleurs la plupart des langages de programmation, exprime a priori les angles en radians et non en degrés. Rappelons que le tour complet d'un cercle représente 360 degrés mais 2.0 * pi radians. D'où des lignes de code Python de ce type :

```
angleRad = (2.0 * pi) * numPoint / nbPtsSurCirc
```

ce qui signifierait en bon français : la valeur de l'angle en radians (`angleRad`; cette appellation est arbitraire, comme celle de toutes les variables employées) est égale à la valeur de la fraction angulaire (une circonférence complète, soit 2 * pi radians, divisée par `nbPtsSurCirc`, c'est-à-dire le nombre de points

sur la circonférence) multipliée par le numéro du point (`numPoint`). Le code informatique de la ligne que nous venons de voir est certes un peu télégraphique, mais relativement compréhensible pour l'auteur qui a écrit cette ligne... et tout à fait digeste pour notre ami Python !

Voilà pour les coordonnées des points. Reste à énumérer les périmètres des faces :

```
<faces>
1,2,3,4,5,6,7,8,9,10,11,12,13,2
14,2,3
14,3,4
14,4,5
14,5,6
14,6,7
14,7,8
14,8,9
14,9,10
14,10,11
14,11,12
14,12,13
14,13,2
</faces>
```

Rien de bien compliqué : la première ligne exprime la surface circulaire de base, constituée de douze « parts de tarte » ayant le même point 1 pour sommet, et les douze lignes suivantes décrivent les douze facettes triangulaires du pourtour du cône, qui ont toutes le point 14 pour sommet.

Bonjour M. Python. Votre mission, si vous l'acceptez, consistera à écrire avec élégance toutes ces lignes SGML 3D. Comme toujours, si ce script ou l'un de ses semblables venait à planter lamentablement, l'auteur nierait énergiquement y être pour quoi que ce soit. Le lecteur de ces lignes ouvrira manuellement une console pour faire l'essai dans cinq secondes. Bonne chance.

12.3.3 Calcul géométrique et d'itérations : cone1.py

Notre premier script, `cone1.py`, ne comporte qu'une cinquantaine de lignes, qui pourraient toutes avoir été écrites par n'importe quel amateur ayant lu le premier chapitre d'un manuel d'initiation à Python. Ça ne l'empêche pas de remplir à peu près correctement sa mission, comme on peut s'en assurer par la ligne de commande suivante :

```
python cone1.py > cone1.3d
```

Et bien sûr, il faut ouvrir `cone1.3d` avec un éditeur de lignes pour juger du résultat.

Les deux principales difficultés d'écriture de ce script sont l'usage du mot clé `str` pour transformer une valeur numérique en une mention textuelle (une « chaîne de caractères », en patois informatique), et la curieuse syntaxe

```
for [variable] in range([maximum]) :
```

Cette ligne (où les mentions entre crochets représentent des noms de variables) exprime ce qu'on appelle une **itération**, c'est-à-dire un traitement répété un nombre précis de fois. Pour de plus amples informations, cherchez `str`, `for` et `range` dans l'index d'un manuel d'initiation à Python.

Le résultat obtenu dans le fichier `cone1.3d` est tout à fait utilisable par nos convertisseurs. Il est toutefois assez inélégant, en raison de la présence de valeurs notées de façon bizarre, du type `6.12303176911e-017`. Si vous comparez avec les valeurs-types que nous avons indiquées plus haut, vous constaterez que c'est tout simplement... 0.0 que l'on s'attendait à trouver à cet emplacement. Rien d'étonnant : `6.12303176911e-017` est une notation informatique courante pour exprimer de toutes petites valeurs, en l'occurrence 6.12303176911 multiplié par 10 à la puissance -17, c'est-à-dire... divisé par cent millions de milliards. `6.12303176911e-017`, ça ne vaut donc pas exactement zéro, mais ça n'en est vraiment pas loin !

Si on ne vous a jamais dit que les ordinateurs ne calculent pas juste, eh bien, voilà, c'est chose faite. Vous pensez peut-être que des erreurs aussi infimes sont totalement négligeables... Vous vous trompez. Mais cela, comme disait Rudyard Kipling, c'est une autre histoire.

12.3.4 Formatage de données : cone2.py

Notre deuxième script, `cone2.py`, est à peine plus long que le premier dont il s'inspire très largement. Pour le lancer, la ligne de commande est

```
python cone2.py > cone2.3d
```

Ce script pourrait encore avoir été écrit par un débutant complet, mais dont les connaissances se seraient enrichies sur un point bien précis : la question du formatage des données – qui sert, en particulier, à limiter au raisonnable le nombre de chiffres inscrits après la virgule (en fait, après le point décimal). Si vous ouvrez le fichier `cone2.3d` avec un éditeur de lignes et le comparez avec le fichier `cone1.3d` produit au paragraphe précédent, vous pourrez constater que les coordonnées des points sont désormais exprimées de façon nettement plus lisible : le nombre de chiffres après la virgule est limité à 6, et la notation incompréhensible (`6.12303176911e-017`) a fait place à une notation nettement plus intelligible (`0.000000`). Le formatage des données, c'est cela.

Ouvrez maintenant le script `cone2.py` avec votre éditeur de lignes. Le formatage des données y est opéré à l'aide d'une syntaxe a priori tout à fait absconse :

```
print "%d,%f,%f,%f" % (numPtAStocker, x, y, z)
```

Au début, nous retrouvons notre ami le mot-clé `print` : il s'agit donc d'« imprimer » une ligne (c'est-à-dire, en fait, de l'afficher à l'écran). Jusque là, c'est simple.

La mention qui suit, `"%d,%f,%f,%f"` est beaucoup moins compréhensible (sauf pour ceux qui ont déjà rencontré ce type de syntaxe, notamment avec les instructions printf et sprintf chères au langage C). En fait, cette mention joue un peu le rôle d'un formulaire de Sécurité sociale : elle réserve dans la mémoire de l'ordinateur un espace d'un nombre adéquat de « cases informatiques » (des octets) en fonction de la nature des mentions qu'on doit y inscrire – un peu comme un formulaire de Sécurité sociale réserve quatre petits carrés pour noter une année de naissance, cinq pour inscrire un code postal, etc.).

En l'occurrence, la mention `"%d,%f,%f,%f"` signifie qu'il faut s'attendre à des lignes du type `3,0.25,6.333,-2.56` (ces valeurs sont choisies arbitrairement pour l'exemple), c'est-à-dire comportant un nombre entier (%d), une première virgule, un nombre décimal (%f), une deuxième virgule, un deuxième nombre décimal (%f), une troisième virgule et enfin un troisième nombre décimal (%f ; nous préférons parler de « nombre décimal » ou, mieux, de « nombre à partie décimale » que de « nombre à virgule » – puisque justement en informatique on sépare la partie décimale par un point et non par une virgule ; attention aux confusions !).

Ces abréviations, il faut bien le dire, sont tout sauf claires : `%d` abrège *digit*, qui signifie chiffre (mais ici, il faut quand même comprendre nombre entier) et `%f` abrège *float* ou plutôt *floating point number*, c'est-à-dire nombre à partie décimale (attention au faux ami : `%d` est l'initiale de *digit* et non de *décimal*, et ne sert justement pas à introduire un nombre à partie décimale). Admettons-le, il est difficile de concevoir une syntaxe plus illisible... mais elle a été employée dès la préhistoire de l'informatique et ne sera probablement pas remise en cause plus vite que la messe en latin !

Tant que nous y sommes, signalons qu'il existe aussi une notation `%s`. Elle signifie *string* – c'est-à-dire littéralement ficelle, mais les informaticiens francophones préfèrent traduire ce mot par chaîne, ou « chaîne de caractères » ; en clair, `%s` signifie que le « formulaire informatique » s'attend à ce qu'on inscrive une petite mention textuelle à cet endroit. C'est en quelque sorte l'équivalent informatique de la case à saisie libre (« Renseignements divers ») d'un formulaire de Sécurité sociale.

Poursuivons. Après la mention incompréhensible (entre guillemets) `"%d,%f,%f,%f"`, la ligne de code Python exprimant le formatage des données comporte à nouveau un signe `%` (ne l'oubliez pas !), et enfin (entre parenthèses cette fois) une liste de quatre variables : `(numPtAStocker, x, y, z)`. Peut-être l'avez-vous déjà compris : c'est le contenu de ces quatre variables qui sera inscrit, dans l'ordre où ces variables sont citées, dans les « cases » du « formulaire informatique » – et

donc sur la ligne finalement « imprimée » (stockée dans le fichier `cone2.3d`). Et en effet, `numPtAStocker` (lire « numéro du point à stocker ») est un nombre entier (`%d`), tandis que les trois coordonnées x, y et z sont des nombres à partie décimale (`%f`).

Oui, tout cela est fichtrement – et inutilement – compliqué. Mais c'est aussi d'un usage si courant que les informaticiens s'y habituent très vite, et finissent même par se convaincre que c'est là une syntaxe particulièrement élégante ! Vous vous y ferez vous aussi.

Si vous trouvez un manuel d'initiation à Python qui vous explique toute cette fichue syntaxe, c'est assurément un excellent livre – mais dans la plupart des cas ces ouvrages considèrent que la question du formatage des données est déjà connue du lecteur. Cette syntaxe absconse est en effet employée par un grand nombre de langages de programmation, et chaque manuel d'initiation laisse à l'autre le soin de l'expliquer. Du coup, même de bons manuels d'initiation aux langages informatiques peuvent être totalement muets sur le sujet. En outre, comme vous l'aurez constaté, elle ne fait appel à aucun mot-clé du langage Python... de sorte qu'il ne faut pas espérer la voir figurer dans l'index des mots-clés qui est pourtant la principale bouée de sauvetage du débutant en Python. Tout cela constitue un vrai bonheur pour les débutants ! C'est pourquoi nous avons choisi d'en parler ici, bien que ça n'ait pas grand rapport avec la 3D.

12.3.5 Routines, structures (classes) et arguments : cone3.py

Notre troisième script, `cone3.py`, n'est certes pas l'œuvre d'un expert, mais ce n'est plus non plus un script de débutant complet, et cette fois nous vous recommandons de vous inspirer de son style – qui est typiquement celui d'un amateur qualifié. Pour lancer ce script, la ligne de commande est

```
python cone3.py 12 > cone3.3d
```

Notez la différence subtile par rapport aux deux exemples précédents : le nom du script est suivi du nombre 12 – qui indique le nombre de points à placer sur la circonférence de la base de notre cône, et donc le nombre de facettes qu'il comportera. Vous entrez dans la cour des grands : ce script unique est déjà capable de produire des objets différents ! Il suffit pour cela de remplacer 12 par une autre valeur. Par exemple :

```
python cone3.py 4 > pyramide.3d
```

produira un « cône à quatre faces » – c'est-à-dire une pyramide (n'hésitez pas à ouvrir le fichier `pyramide.3d` pour vous en convaincre). Vous pouvez aussi essayer une valeur plus élevée que 12 et vous extasier sur l'abondance des lignes que peut produire un petit script de rien du tout... Cela dit, nous vous déconseillons fortement de dépasser une valeur de 36 ; ce n'est pas Python que cela gênerait, mais cela augmenterait sans grande utilité graphique la taille du fichier produit.

La principale différence entre le script `cone3.py` et les deux exemples précédents, c'est qu'il ne suit plus une logique purement séquentielle : au lieu d'être lisible comme un article de journal (enfin, peut-être pas avec la même facilité...) du début à la fin, il est découpé en **routines** (au début) et en **programme principal** (à la fin). Les routines sont des bouts de programme qui peuvent être appelés à plusieurs reprises lors de l'exécution du programme principal, permettant à ce dernier de ne pas se répéter inutilement. Voici par exemple la routine `fermeBalise`, qui transforme une mention textuelle du type `balise` en véritable balise fermante conforme à la syntaxe SGML, du type `</balise>` :

```
def fermeBalise (nomBalise) :
    ligne = "</%s>" % (nomBalise)
    print ligne
```

Nos lecteurs les plus attentifs auront reconnu au premier coup d'œil la syntaxe de formatage `%s` dont nous parlions dans l'exemple précédent.

Ce n'est pas le sujet de ce livre que de vous apprendre à écrire et à utiliser de telles routines : pour votre apprentissage, vous devrez impérativement vous rabattre sur un manuel d'initiation à Python ou un tutoriel du web. Sachez seulement que ce n'est pas la mer à boire, et que vous pourrez parfaitement vous inspirer de ce script pour écrire vos premières routines. Sachez aussi que l'on peut parfaitement utiliser des routines que l'on n'a pas écrites soi-même sans chercher à en comprendre la logique interne – et cela pourra vous servir avec le script `cone5.py` dont nous parlerons plus loin : vous pourrez l'adapter à vos besoins sans avoir besoin de comprendre le moins du monde les routines qu'il contient : il vous suffira de les conserver sans y changer une virgule.

L'usage de routines n'est pas la seule chose à remarquer dans le script `cone3.py`. Vous pourrez aussi noter, tout au début, la définition d'une structure de données :

```
class Point :
    def __init__(self, num = 0, x = 0.0, y = 0.0, z =
0.0) :
        self.num = num
        self.x = x
        self.y = y
        self.z = z
```

Là encore, reportez-vous à un manuel ou à un tutoriel de Python pour comprendre de quoi il retourne (cherchez les mots-clés `class, init` et `self`)... mais ne vous croyez pas obligé d'absorber tout ce que ces ouvrages pourront vous raconter sur la question. Les classes de Python sont un outil extrêmement puissant, et permettent de s'élever à un niveau d'abstraction phénoménal... dont l'amateur de 3D n'a pas du tout besoin, en tout cas pas avant de s'attaquer à des projets extrêmement ambitieux. Si vous parvenez simplement, comme dans l'exemple ci-dessus, à utiliser le concept de classe pour réunir en un ensemble logique un

petit paquet de variables (en l'occurrence, le numéro d'un point 3D et ses trois coordonnées), cela peut suffire à vous rendre d'immenses services – et comme le démontre cet exemple, une demi-douzaine de lignes sont suffisantes pour cela. En patois informatique, cela signifie que le langage Python, quoique orienté objets, est tout à fait utilisable en logique procédurale (si cette dernière phrase vous fait le même effet que de l'hébreu, rassurez-vous, c'est absolument normal ; mais il est possible que ce livre soit aussi lu par quelques informaticiens qui comprendront à demi-mot où nous cherchons à les mener).

La dernière chose que l'on peut remarquer dans ce script `cone3.py`, c'est la syntaxe pour récupérer les paramètres de la ligne de commande (dans les exemples que nous avons donnés, les valeurs 12 ou 4 qui exprimaient le nombre de points sur la circonférence du cône) :

```
import sys
(...)
nbArgs = len(sys.argv)

if nbArgs >= 2 :
    nbPtsSurCirc = int(sys.argv[1])
else :
    nbPtsSurCirc = 8
```

Tant que vous n'aurez pas ouvert un manuel d'initiation à Python, ces quelques lignes vous paraîtront sans nul doute totalement incompréhensibles... mais à un certain moment de votre apprentissage, vous nous remercierez de les avoir écrites pour vous (l'auteur, lui, a perdu beaucoup de temps à fouiller son manuel pour y parvenir seul, et il en garde un souvenir cuisant). Notez qu'elles comportent un test qui impose une valeur de 8 au nombre de points sur la circonférence de notre cône, dans le cas où l'utilisateur aurait appelé le script en omettant d'indiquer cette valeur.

12.3.6 Itération inverse : cone4.py

Pour lancer ce script, presque semblable au précédent, la ligne de commande est

```
python cone4.py 12 > cone4.3d
```

Vous avez à peine besoin de vous intéresser à cette étape. La seule différence avec le script précédent, c'est que la surface de base de notre triangle est désormais exprimée par la ligne

```
1,2,13,12,11,10,9,8,7,6,5,4,3,2
```

au lieu de

```
1,2,3,4,5,6,7,8,9,10,11,12,13,2
```

afin de nous conformer à la règle que nous avons évoquée au chapitre 4 (encadré « Dans quel ordre appeler les points d'un périmètre ») : les points du périmètre doivent être appelés dans le sens géométrique (l'inverse de celui des aiguilles d'une montre) avec la face considérée sous un angle d'où elle est visible (en l'occurrence, depuis le dessous du cône et non en vue de dessus). On peut à bon droit juger que nous faisons ici montre d'un purisme idéal pour embrouiller le débutant... Mais c'est pour introduire la syntaxe de l'**itération inverse** que l'auteur a eu, là encore, beaucoup de mal à trouver dans son propre manuel d'initiation, bien qu'elle soit très utile à l'amateur de 3D pour résoudre ce problème précis :

```
for numPoint in range(nbPtsSurCirc - 1, 0, -1) :
```

Ne cherchez pas à la comprendre tout de suite... mais vous verrez qu'un jour vous nous remercierez d'avoir pensé à vous l'indiquer. Vous trouverez les explications pertinentes dans votre manuel au mot-clé `range`.

12.3.7 Usage d'un fichier d'initialisation : cone5.py

Notre dernier script, `cone5.py`, est plus de trois fois plus long que les précédents, et peut donc vous effrayer par sa complexité. Mais l'idée n'est pas que vous acquerriez la compétence pour l'écrire vous-même. Vous n'avez pas non plus besoin d'en comprendre le détail. Il vous suffit de savoir que ce script est capable d'exploiter un fichier d'initialisation (nommé en l'occurrence `cone5.ini`). Sur la base de ce squelette que vous modifierez et surtout enrichirez à votre guise (avec les mêmes compétences que celles que réclamaient les exemples précédents), vous pourrez mettre au point vous-même des scripts dont l'action sera paramétrée par un tel fichier ; ce qui, spécialement pour le débutant, est une bénédiction : un fichier d'initialisation est beaucoup plus facile à modifier que le script Python le plus élégant.

Pour lancer ce script, la ligne de commande est particulièrement laconique :

```
python cone5.py
```

En effet, comme ce script a recours à un fichier d'initialisation, il ira y chercher tous les paramètres qui lui sont utiles et n'a donc nul besoin qu'on lui en indique sur la ligne de commande. Voici d'ailleurs le contenu du fichier `cone5.ini` :

```
nomFichResultat = cone5.3d
nbPtsSurCirc = 18
couleurCone = 60.0,1.0,0.8
```

En d'autres termes, ce fichier d'initialisation comporte trois paramètres : `nomFichResultat`, le nom du fichier résultat (le fichier SGML 3D que le script produira) ; `nbPtsSurCirc`, le nombre de points sur la circonférence du cône ; et enfin `couleurCone`, c'est-à-dire la couleur de l'objet 3D produit (définie selon la syntaxe « teinte, saturation, niveau de gris » que nous avons étudiée au chapitre 4 ; en l'occurrence, il s'agit d'un jaune vif).

Ces trois mots (`nomFichResultat, nbPtsSurCirc, couleurCone`) doivent bien sûr être repris dans le programme Python pour que les valeurs qui leur sont associées soient correctement exploitées. Vous pouvez les rechercher dans le script à l'aide de votre éditeur de lignes pour repérer les endroits où ils sont exploités.

La commande la plus importante pour permettre une telle utilisation est la suivante :

```
vrcIni = litFichIni('cone5.ini')
```

c'est-à-dire, en bon français, « lis le fichier d'initialisation `cone5.ini` et stocke les paramètres qu'il contient dans le **dictionnaire** `vrcIni` (lisez : « vrac d'initialisation »). La seule compétence Python que vous devez acquérir pour en tirer parti est la maîtrise de ce concept de dictionnaire (cherchez ce mot-clé dans votre manuel Python). A l'usage des informaticiens, mentionnons qu'un dictionnaire de Python est fort peu différent des hachages (*hash*) du langage Perl ou des « paires clé-valeur » d'autres outils informatiques comme awk... et à l'usage de tout un chacun, disons que c'est une astuce informatique très pratique pour manipuler avec un seul mot (en l'occurrence, `vrcIni`) un véritable fourre-tout de paramètres divers. Les lignes du script `cone5.py` qui utilisent ce fourre-tout sont celles où vous lisez des mentions comme `vrcIni['nomFichResultat']` ou `vrcIni['nbPtsSurCirc']`. Même si cette syntaxe peut paraître un peu rebutante pour le débutant, cette possibilité facilite considérablement l'écriture des scripts les plus minutieux, et vous avez donc tout intérêt à l'étudier... d'autant que les routines que nous avons écrites pour vous dans `cone5.py` vous ont vraiment mâché le travail.

Par parenthèse, l'auteur qui a écrit ces routines n'est pas du tout un expert en Python, et sa seule méthode a été de transposer sans réfléchir le mode de fonctionnement auquel il était habitué en Perl et en C. Ces routines fonctionnent et c'est tout ce qu'on leur demande, mais il est probable que des pythoniens qualifiés sauraient les réécrire de façon beaucoup plus laconique et élégante ; qu'ils ne s'en privent surtout pas.

Chapitre 13

Vers l'animation

13.1 Animer, c'est automatiser et compresser

Nous terminerons ce livre, sinon en apothéose, du moins sur un joli bouquet final, en touchant un mot de la production de séquences vidéo en images de synthèse – et si nous ne parlons pas de films d'animation, c'est bien par horreur des grands mots ; car en fait, c'est bien de cela qu'il s'agit, au moins en termes techniques.

Si vous avez compris par la lecture des chapitres précédents comment automatiser de A à Z la production d'une image de synthèse, vous disposez en fait déjà de la plupart des armes nécessaires pour vous attaquer à l'animation. Car, contrairement aux êtres humains, les ordinateurs peuvent sans difficulté ni fatigue recommencer dix mille fois de suite ce qu'on les a dressés à faire ne fût-ce qu'une fois. Et pour l'essentiel, le calcul d'une animation se résume à multiplier quelques milliers de fois la production d'une image de synthèse : ce sera des milliers de fois plus long, des milliers de fois plus encombrant sur le disque dur... mais ça ne demandera aucun accroissement de puissance de la part de votre ordinateur – et de votre part, pas une goutte d'huile de coude de plus ni même aucune augmentation notable de vos compétences informatiques. Si donc vous avez compris comment commander à votre ordinateur de réaliser quelques milliers de fois de suite la production d'une image de synthèse unique (comme nous l'avons vu précédemment avec le script `fabrique_tout`), vous n'avez plus grand-chose à apprendre avant de vous attaquer à l'animation.

Il nous reste cependant deux difficultés à mentionner. Premièrement, nous devons définir une méthode pour que les milliers d'images que nous produirons comportent des milliers de variantes minimes – et surtout régulières, pour la fluidité de l'animation ; cela est tout à fait à la portée d'un modeste script. Secondement, nous devrons trouver un moyen de convertir les milliers d'images produites en un

fichier informatique unique, aussi standard que possible, et beaucoup moins volumineux que l'addition de ces milliers d'images ; et là, malheureusement, il faut admettre qu'il est difficile de résoudre ce problème sans faire appel à des logiciels commerciaux fort chers, comme Premiere d'Adobe ou Final Cut Pro d'Apple. Difficile... mais pas impossible. Les linuxiens pourront trouver dans le monde du logiciel libre tout le nécessaire pour automatiser de A à Z – et sans bourse délier – le processus de fabrication d'une vidéo informatique. Les autres devront sans doute ouvrir leur portefeuille, ce qui est désagréable, avoir recours à la copie illégale de logiciels, ce que nous désapprouvons totalement – ou chercher de l'aide auprès d'associations d'utilisateurs de logiciels libres, ce qui est la voie royale vers le nirvana informatique : l'auteur de ces lignes est d'ailleurs passé par là...

13.2 Des scripts pour l'animation

13.2.1 Un exemple complet sous Linux

Vous trouverez sur le CD-ROM joint, dans le dossier `3d_linux`, un sous-répertoire du nom de `video_autour_objet`, assez semblable dans sa structure au dossier-type qui nous a servi pour les manipulations du chapitre 8, mais déjà doté de scripts pour la production de longues séries d'images et leur compression.

Vous trouverez aussi, dans le dossier `3d_win`, un autre sous-répertoire `video_autour_obje` qui constitue une pâle imitation du premier. Cela étant, il peut suffire à vous donner des idées de ce qu'il est possible de faire dans le monde du logiciel libre – et par ailleurs, tel qu'il est, il est déjà suffisant pour produire sans les compresser toutes les images de l'animation (fig. 13.1).

A ce jour, l'auteur n'a pas trouvé le moyen d'obtenir un résultat similaire en Python avec Mac OS/X – même si ce serait probablement assez facile en ayant recours à des doses homéopathiques d'AppleScript pour faire sauter les limitations liées à l'absence d'interface en ligne de commande pour la version Macintosh de POV-Ray. Si vous êtes applemaniaque, vous n'avez probablement pas les mêmes réticences idéologiques que l'auteur quant à l'emploi de cet outil propriétaire, et vous parviendrez probablement à adapter les scripts que nous avons écrits pour Linux.

Dans tous les cas, l'idée est de produire une animation très simple (mais redoutablement efficace !) consistant à faire tourner une caméra virtuelle autour d'un point central (où, bien sûr, on aura placé un objet), en la maintenant constamment braquée vers lui. Ce n'est pas follement original, mais c'est très suffisant pour présenter une maquette d'architecture sous tous les angles. Vous pouvez vous faire une idée du type de petit film que cela peut produire en lisant l'animation

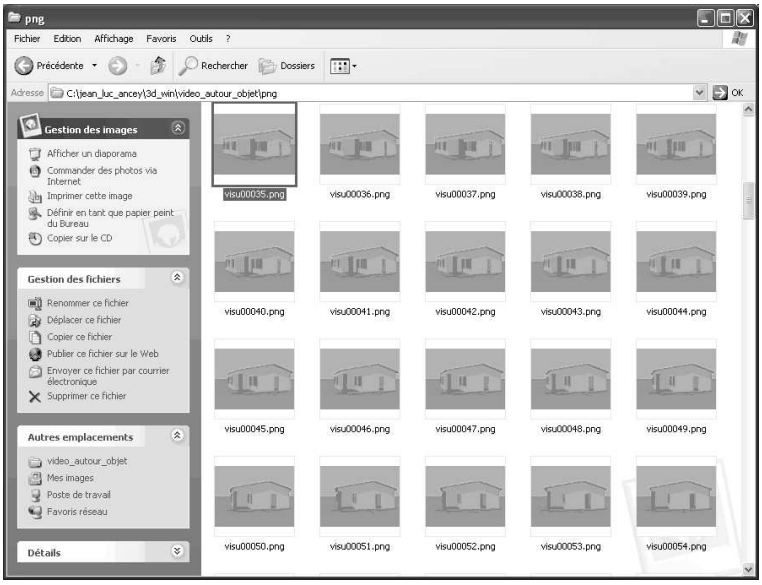


FIG. 13.1 – *Les images constituant l'animation produite par les scripts du sous-dossier* `video_autour_objet` *(vues ici avec le mode « Miniatures » de Windows XP).*

`chalet.mpeg` (chemin d'accès sur le CD-ROM : dossier `a_voir`, sous-dossier `animations`, sous-dossier `mpeg`).

Comment cela marche-t-il ? Rien de bien sorcier si vous avez assimilé les chapitres qui précèdent. Comme d'habitude avec les dossiers ayant la structure de `08_scene_type` (bien entendu, `video_autour_objet` en est une adaptation), l'aspect général de la scène dépend de la hiérarchie SGML 3D chapeautée par le fichier `source.3d`. Le fichier `reglages.ini` que nous avons appris à connaître sera utilisé par les scripts, mais il est inutile de vous en préoccuper, car les paramètres relatifs à l'animation proprement dite ne sont pas inclus dans lui, mais dans le fichier `video.ini` du sous-répertoire `ini`. En voici le contenu :

```
distObs = 15.0
zObs = 1.6
xDest = 0.0
yDest = 0.0
zDest = 1.6
dureeAnim = 0.5
imagesParSeconde = 30
```

Il faut lire : la caméra est située à 15 m (paramètre `distObs`, distance d'observation) d'un point de coordonnées 0.0, 0.0, 1.6 (paramètres `xDest, yDest,`

`zDest`) autour duquel elle va tourner et sur lequel elle restera braquée. Cette caméra est située à une hauteur de 1.60 m (paramètre `zObs` ; en l'espèce, la caméra regarde à l'horizontale, mais ce n'est nullement une obligation). Il s'agit de produire une animation d'une durée de 0.5 seconde (paramètre `dureeAnim` ; c'est très peu, mais suffisant pour faire des essais, et rien n'interdit de modifier ensuite ce paramètre) à la cadence de 30 images par seconde (aux normes européennes, la valeur serait de 25, mais l'utilitaire libre `ppmtompeg` que nous utilisons pour la compression se conforme par défaut aux standards américains).

Pour lancer le processus de fabrication (qui mènera directement à la production d'un fichier Mpeg), un linuxien peut se contenter de taper (bien sûr depuis le répertoire `video_autour_objet`)

```
sh produit_video.sh
```

Ce script shell très bref ne sert en fait qu'à lancer trois autres scripts, rédigés quant à eux en Python (et aussi, accessoirement, à noter dans un fichier `compte_rendu_video.txt` le temps nécessaire à leur exécution, qui peut atteindre plusieurs heures si la durée de l'animation se compte en dizaines de secondes). Ces trois scripts, dont vous avez bien évidemment tout intérêt à étudier la logique, portent des noms qui résument bien leur fonction.

Le premier, `produit_serie_png.py`, lance POV-Ray le nombre de fois nécessaire pour fabriquer au format PNG toutes les images de l'animation, les numéroter, et déplacer légèrement à chaque fois la caméra autour du point central. Ce déplacement de caméra prend très prosaïquement la forme d'une modification des paramètres `xObs` et `yObs` dans le fichier `reglages.ini` (avec notre petit utilitaire `modifini`). La modification une fois effectuée, le script relance POV-Ray pour produire une image de plus.

Le second script Python, `conv_png_ppm.py`, convertit ces nombreux fichiers PNG tout juste produits au format PPM (à l'aide d'un utilitaire libre généralement installé avec une bonne distribution Linux et nommé `pngtopnm`). Rien de bien malin.

Enfin, le troisième script Python, `conv_ppm_mpeg.py`, lance le compresseur libre `ppmtompeg` (là encore disponible dans toute bonne distribution Linux), et surtout le paramètre selon une syntaxe assez absconse – que vous n'avez pas besoin de connaître, puisque le script se charge de la respecter à votre place.

Quant à la pâle imitation pour Windows, on peut la lancer (bien sûr depuis le répertoire `video_autour_objet`) par la commande

```
python produit_serie_png.py > produit_video.bat
```

car l'idée n'est pas de fabriquer directement les images, mais seulement... un fichier .bat capable de les produire. Par prudence, mieux vaut y jeter un coup d'œil avant de le lancer (fig. 13.2). Si tout a bien fonctionné, vous reconnaîtrez la logique que nous avons décrite quelques lignes plus haut en parlant de la version de ce script pour Linux (appels nombreux à `modifini` avant de relancer POV-Ray

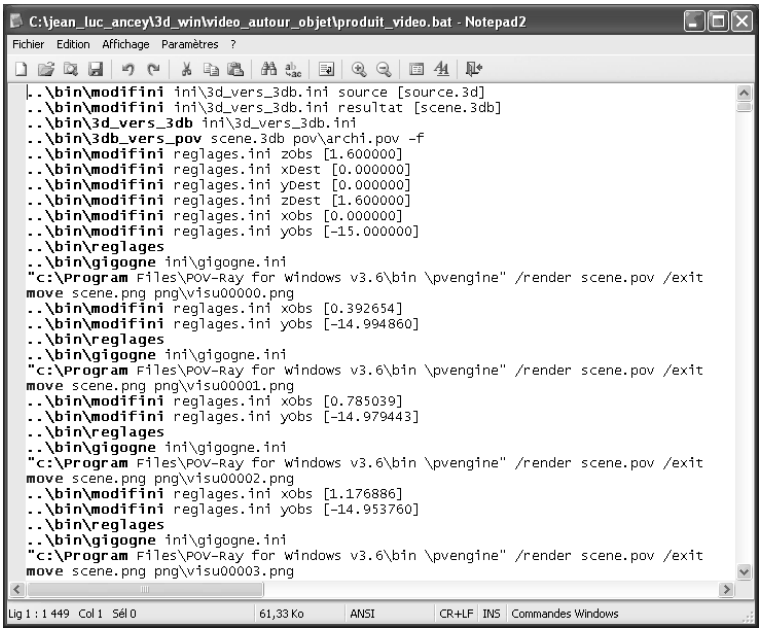


FIG. 13.2 – *La version pour Windows de notre script* `produit_serie_png.py` *ne fabrique pas les images directement, mais écrit un batch sur lequel on a tout intérêt à jeter un coup d'œil de vérification avant de lancer le long processus qu'il décrit.*

pour chaque image).

13.2.2 Autres types d'animation concevables

Vous l'aurez compris, toutes les actions effectuées par les scripts dont nous venons de parler sont... de simples manipulations de fichiers de texte. Elles ont lieu sur l'écran noir d'une console, et si le script affiche les images POV-Ray en cours de fabrication, c'est uniquement pour vous permettre de voir en cours de route à quel stade de l'opération l'ordinateur en est ; en réalité, tout cela peut être fait... sans aucun affichage graphique.

Ou pour le dire autrement, *vous n'avez aucun besoin de savoir programmer une interface graphique pour lancer la production d'une séquence d'animation* : si vous savez exprimer un paramètre d'une scène 3D par des lignes de texte (et nous vous avons appris à exprimer la position d'une caméra dans un fichier .ini, un éclairage dans le sous-fichier `light.pov`, la géométrie d'un objet dans un fichier SGML 3D...), vous pouvez envisager d'écrire un script qui modifie automatiquement ce paramétrage, et donc vous appuyer sur cette possibilité pour

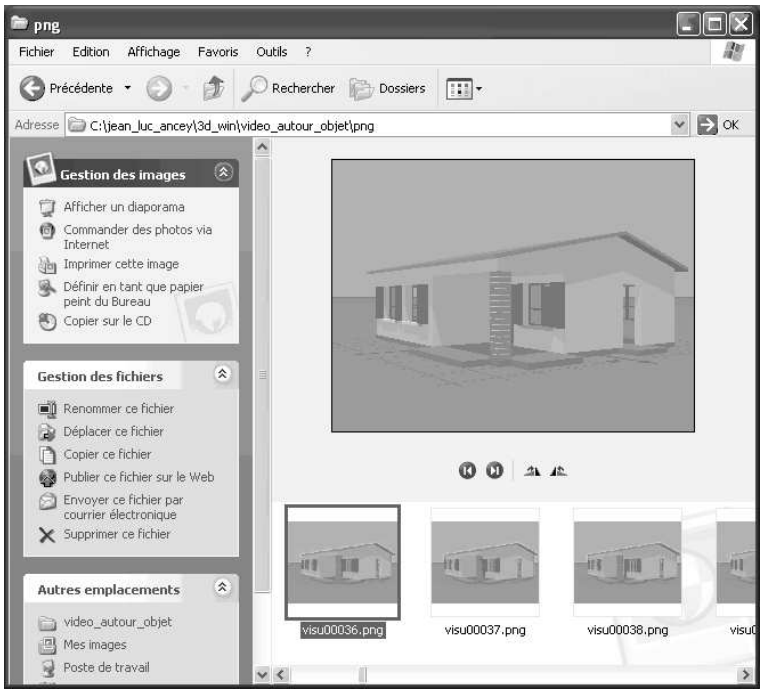


FIG. 13.3 – *A défaut d'effectuer la compression vidéo lui-même, Windows XP est doté d'une mode de visualisation « Pellicule » qui vous permet de faire défiler la série d'images à grands renforts de clics de souris.*

produire les séquences d'animation les plus diverses.

Le fait que le format SGML 3D s'exprime dans une hiérarchie de petits fichiers vous aidera considérablement dans cette optique, car il vous suffit de modifier les quatre ou cinq lignes d'un élément charnière de cette hiérarchie pour transformer le positionnement ou l'orientation (ou même les proportions, la couleur, l'aspect...) de tous les objets qui lui sont raccordés hiérarchiquement.

A titre d'exemple, ouvrez avec un éditeur de lignes et regardez à quoi ressemblent les fichiers dont le nom se termine par `_artic.3d` dans le sous-dossier `09_pantin` du répertoire `sgml_fichiers` de notre CD-ROM : ces fichiers laconiques, qui décrivent l'orientation de chacun des éléments du pantin articulé, sont toujours composés en tout et pour tout d'une dizaine de lignes pleines de balises `<inclusion>` et `<modifcoord>`. N'importe lequel de ces fichiers peut en un tournemain être intégralement réécrit par un script pour la production de chacune des milliers d'images d'une animation. Apprenez assez de Python pour simplement écrire de façon pertinente cette dizaine de lignes, et vous n'aurez aucun mal à animer ce pantin de façon parfaitement régulière.

13.3 Quelques exemples d'animations

La réalisation d'animations en images de synthèse mériterait à elle seule de faire l'objet de tout un livre. L'auteur de cet ouvrage ne s'y est jamais intéressé qu'en marge de ses recherches sur l'aspect géométrique et surtout algorithmique de la 3D, et les seules animations qu'il a réalisées lui-même sont extrêmement rudimentaires. Mais c'est peut-être précisément pour cette raison qu'il peut être intéressant d'y jeter un œil : si vous voyez ce qu'un amateur comme l'auteur a pu faire en ne consacrant presque aucune attention à la question, vous pourrez vous faire une idée de ce à quoi vous pouvez vous attaquer vous-même si le sujet vous passionne... Quoi qu'il en soit, le sous-dossier `animations` du répertoire `a_voir` de notre CD-ROM contient quelques modestes essais aux formats AVI, MPEG et même FLC (un format aujourd'hui tombé en désuétude, mais qui était pourtant très adapté à la 3D maillée). Nous vous recommandons en particulier de vous intéresser aux mouvements de caméra dans le fichier `scene.mpeg` : dans le même plan-séquence ininterrompu, vous verrez la caméra passer en rien de temps de la vitesse d'un avion à réaction à celle d'un cycliste asthmatique. Cet effet assez bluffant est très facile à réaliser par script... et beaucoup trop difficile à expliquer pour que nous nous y essayions ici (il fait appel à rien de moins que la technique des... *courbes de Bézier définies dans l'espace-temps* !). Nous ne citons cet exemple que pour vous convaincre qu'avec deux sous de science algorithmique, l'amateur de 3D peut s'attaquer à des effets très spectaculaires. Et c'est bien ce que nous vous conseillons de faire.

Alors, à vos claviers !

Paris, le 5 août 2005.

Quatrième partie

Annexes

Chapitre 14

La 3D maillée en couleurs

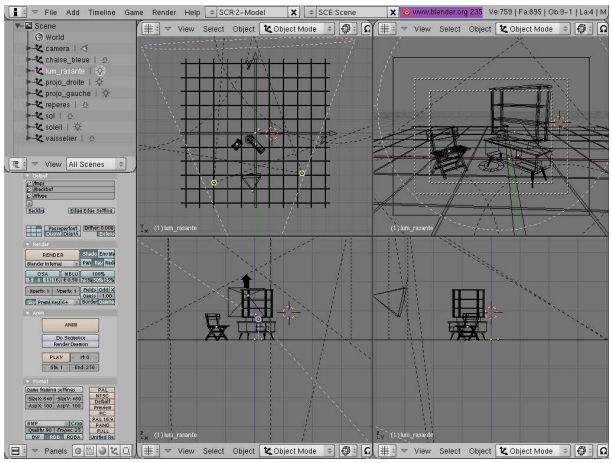


FIG. 14.1 – *L'une des meilleures configurations de l'interface de Blender.*

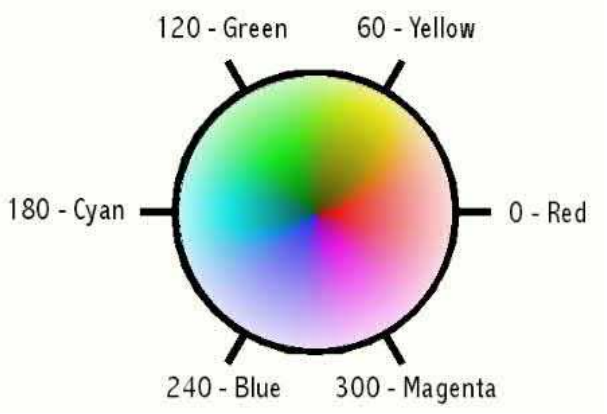


FIG. 14.2 – *Le cercle chromatique du modèle « teinte, saturation, niveau de gris » (voir en 4.2.3).*

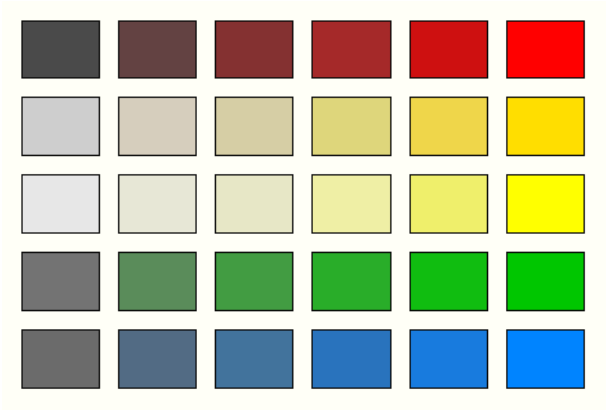


FIG. 14.3 – *Différents niveaux de saturation de la couleur (voir en 4.2.3).*

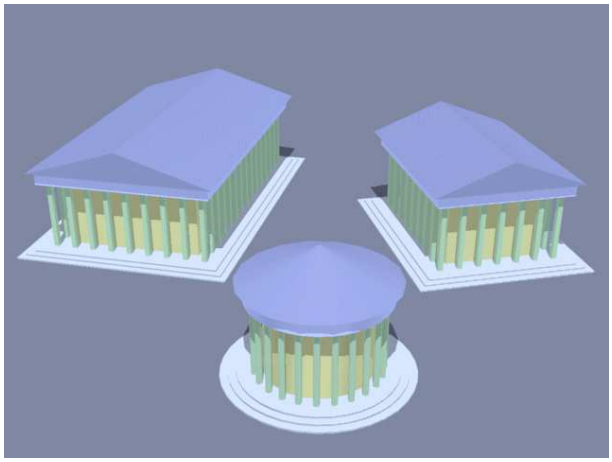


FIG. 14.4 – *Les formes architecturales les plus classiques sont aussi les plus faciles à imiter.*

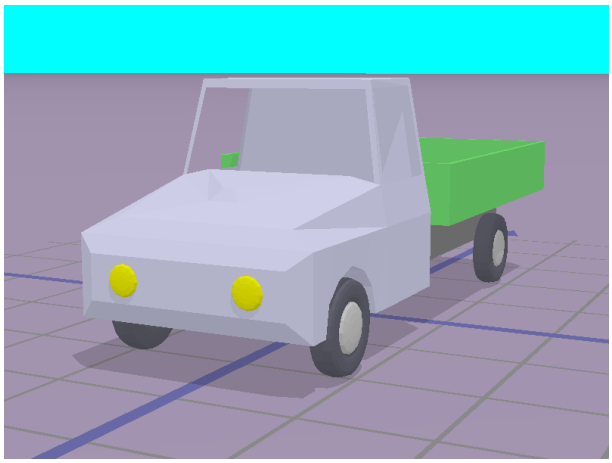


FIG. 14.5 – *La cabine de ce camion a été modelée manuellement avec Blender.*

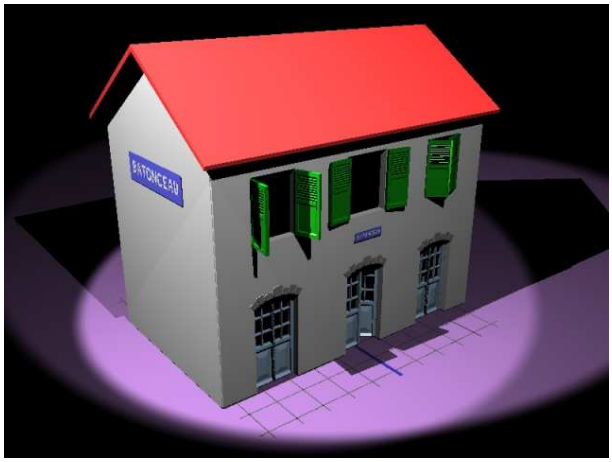


FIG. 14.6 – *Cette petite gare a été modelée en une demi-journée avec Blender. La façade est une extrusion.*

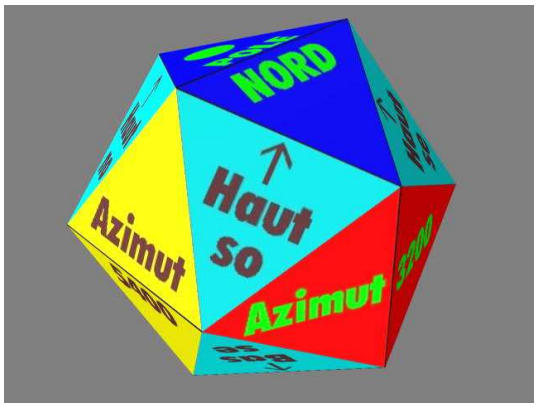


FIG. 14.7 – *Application de vingt textures différentes sur le même objet (voir le sous-dossier* `icosaedre` *du répertoire* `sgml_fichiers` *de notre CD-ROM).*

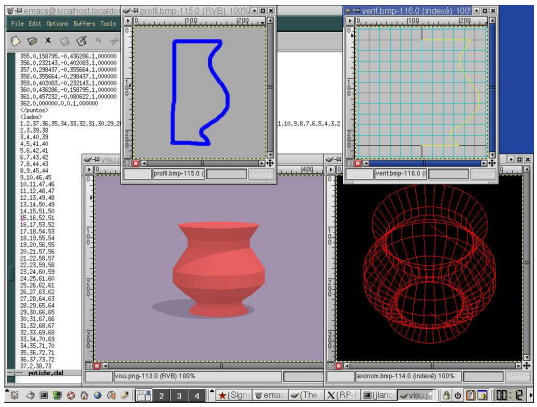


FIG. 14.8 – *La technique du tour de potier utilisée à partir d'un dessin à main levée. Tout le code informatique a été généré automatiquement par des convertisseurs.*

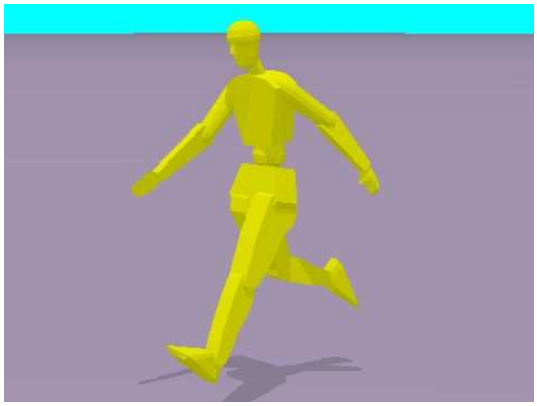


FIG. 14.9 – *Ce pantin articulé, conçu pour l'animation, a été dessiné en 2D à main levée (voir chapitre 9).*

FIG. 14.10 – *Trois objets notablement différents, produits en rien de temps par un script unique.*

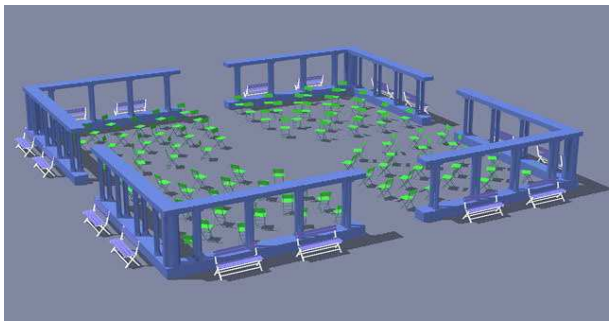


FIG. 14.11 – *L'un des principaux avantages d'une codification SGML de la 3D est la facilité à mettre au point des positionnements minutieux et réguliers.*

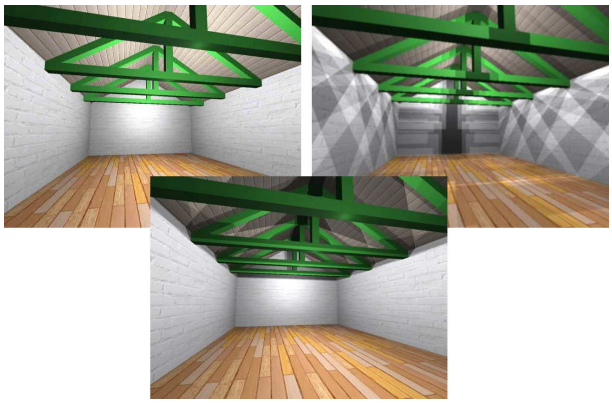


FIG. 14.12 – *Divers essais d'éclairage sur une scène texturée. Les sources de lumière ne peuvent pas apparaître sans que l'image soit retouchée.*

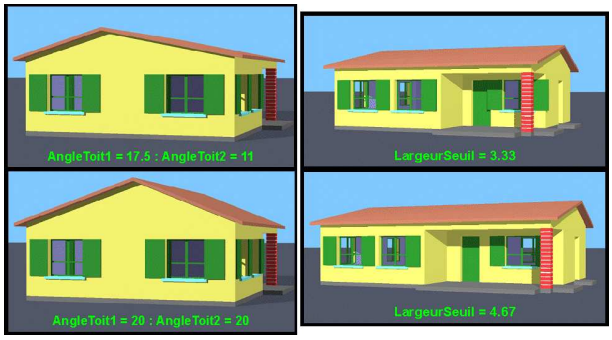


FIG. 14.13 – *Les scripts sont particulièrement efficaces pour gérer l'architecture : la simple modification d'un paramètre chiffré peut suffire à modifier des séries entières d'éléments.*

FIG. 14.14 – *Le positionnement impeccable de la texture de briques n'aurait pas été possible sans l'écriture par un script d'un code SGML extrêmement précis.*

FIG. 14.15 – *Maquette de la cathédrale de Cochabamba (Bolivie), réalisée sans aucun plan... mais avec toute une panoplie de scripts !*

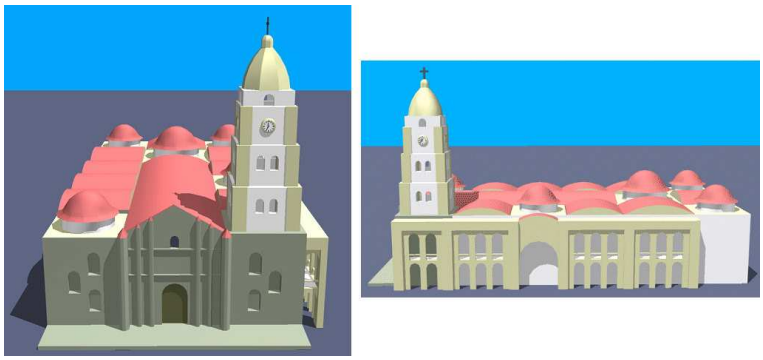


FIG. 14.16 – *Une maquette d'architecture ne peut être exacte sans une réflexion approfondie sur sa structure.*

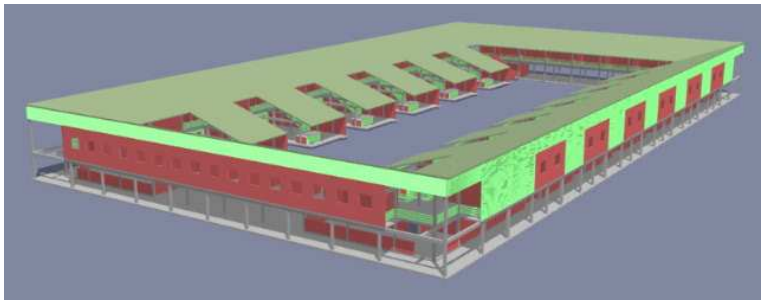


FIG. 14.17 – *Le projet de logement social de l'architecte bolivienne Liliana Collazos. Les images qui suivent ont toutes été produites par des scripts.*

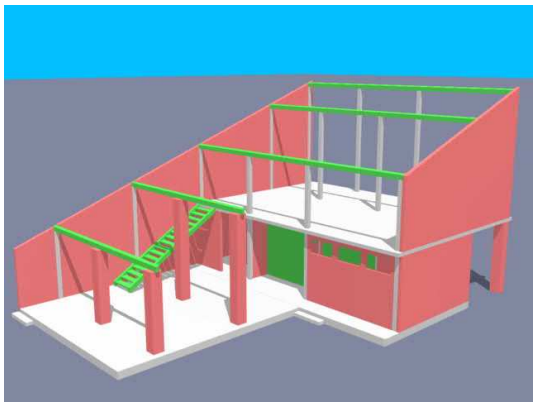


FIG. 14.18 – *Premier brouillon du projet de Liliana Collazos...*

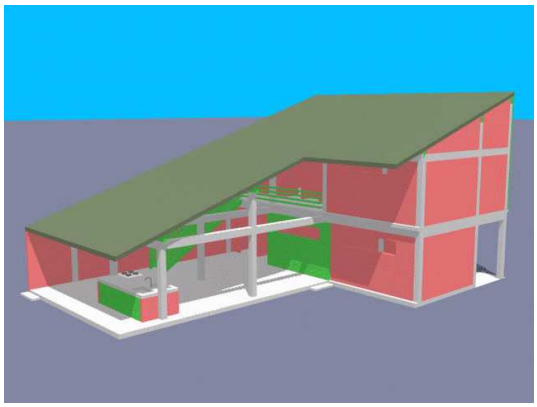


FIG. 14.19 – *... et version définitive. D'innombrables corrections de détail ont dû être intégrées.*

FIG. 14.20 – *L'usage d'un script a permis très simplement de choisir l'écartement optimal de ces volets d'aération.*

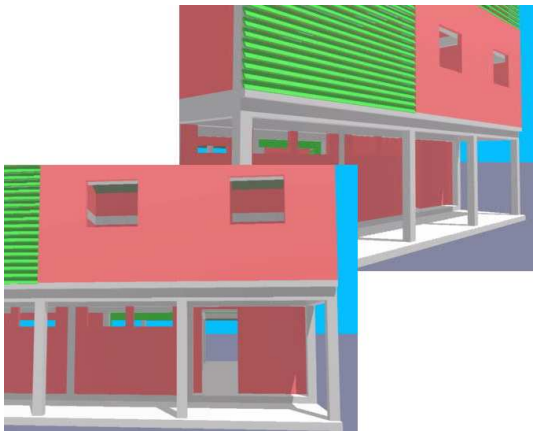


FIG. 14.21 – *Vues de détail de la façade.*

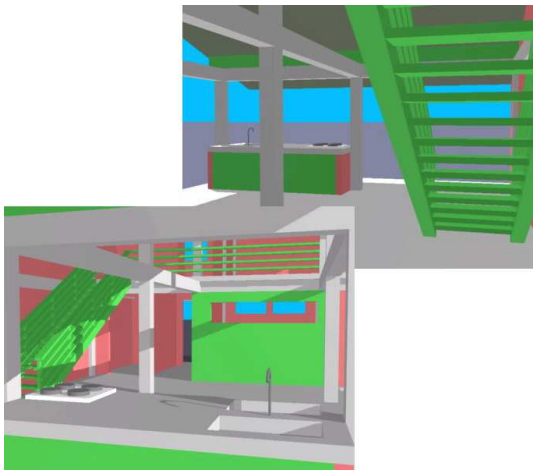


FIG. 14.22 – *Etude sur l'ensoleillement du logement à divers moments de la journée.*

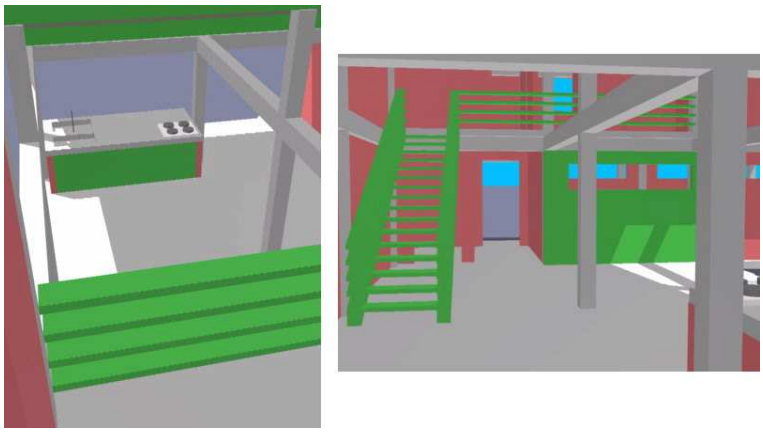


FIG. 14.23 – *Les images de synthèse permettent en quelque sorte la visite d'un bâtiment avant même sa construction.*

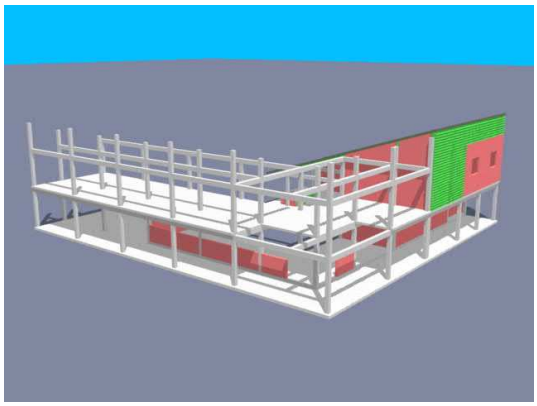


FIG. 14.24 – *En image de synthèse comme dans la réalité, le gros œuvre précède toujours les finitions.*

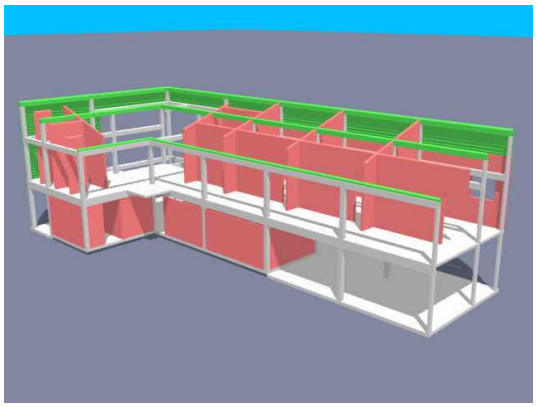


FIG. 14.25 – *Le toit a été sciemment enlevé pour mieux faire comprendre la structure du bâtiment.*

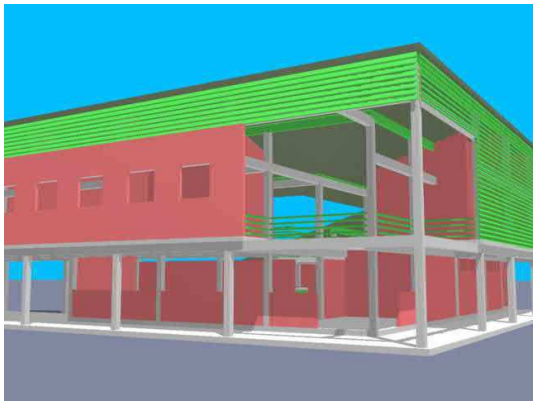


FIG. 14.26 – *Vue de détail de l'angle du pâté de maisons. Notez la précision des positionnements.*

FIG. 14.27 – *Un effet de perspective, aussi impressionnant que facile à produire.*

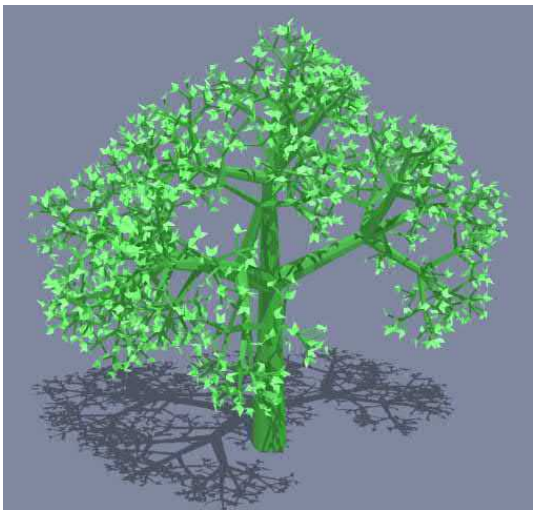


FIG. 14.28 – *Les motifs végétaux, beaucoup trop riches en détails, ne peuvent être fabriqués manuellement. Mais avec un script et un petit peu d'astuce informatique, on en vient facilement à bout.*

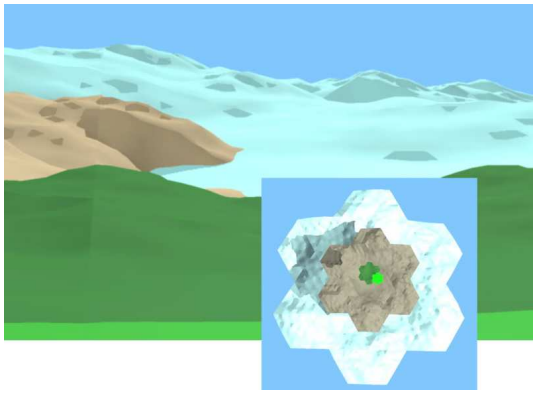


FIG. 14.29 – *Le paysage à l'arrière-plan est tout simplement une vue en perspective des motifs répétitifs que la vignette représente en vue de dessus.*

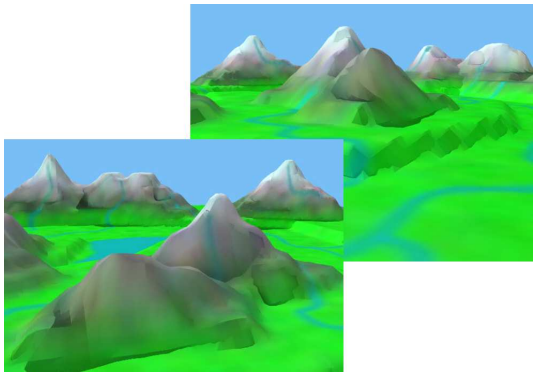


FIG. 14.30 – *Deux vues sur le même paysage informatique...*

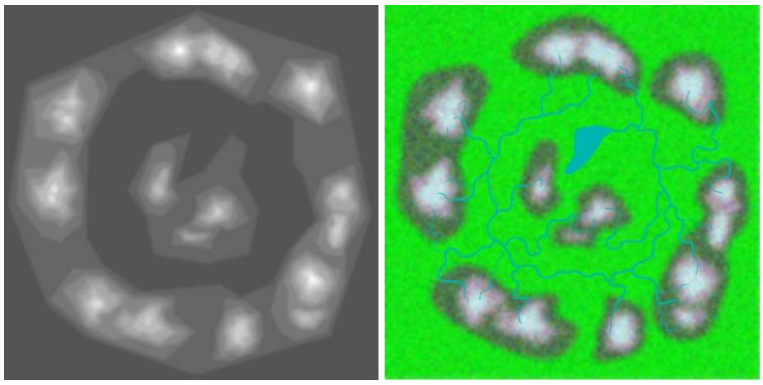


FIG. 14.31 – *... créé par programmation à partir de motifs simples, élaborés en 2D : des superpositions de surfaces grises suffisent à définir le relief, et une petite texture, retravaillée avec un logiciel de retouche, complète l'effet.*

FIG. 14.32 – *La 3D fabriquée par scripts dans toute sa splendeur : toute cette salle a été fabriquée de façon automatique à partir de la série de photos informatisées qui décorent les murs.*

Chapitre 15

Compléments d'information

15.1 Les textures

Pour faire simple, une texture n'est rien d'autre qu'une photographie informatisée (généralement enregistrée au format Jpeg) représentée dans un espace en trois dimensions (au besoin en lui donnant la taille d'une affiche). Comme n'importe quelle image numérique, la texture est fondamentalement plate et rectangulaire... et le problème est de la représenter en 3D sur un relief qui n'est généralement pas plat ni rectangulaire ! C'est un problème très similaire à celui que l'on rencontre lorsque l'on colle du papier peint sur des angles de murs : il peut être nécessaire de plier le papier, voire d'en découper certains morceaux pour les jeter purement et simplement.

La question des textures est très vaste. La 3D use et même abuse des textures pour donner un aspect plus réaliste aux scènes qu'elle produit, avec des techniques qui relèvent de la création graphique au sens large, beaucoup plus que de la géométrie. Nous n'avons pas l'intention d'aborder ici en détail toutes ces questions, qui demandent beaucoup plus de maîtrise des logiciels de retouche d'images que de connaissances de 3D maillée. Cependant, les notions mises en jeu sont tellement similaires à celles que nous avons vues précédemment, et surtout la technique de repérage de pixels précis sur une image bitmap (que nous avons décrite pour expliquer l'emploi des convertisseurs `bmp_vers_2d` et `bmp_vers_profil`) simplifie si radicalement cette question complexe qu'il serait dommage de ne pas en toucher un mot.

Nous avons appris à décrire en SGML 3D un relief quelconque, en indiquant les coordonnées 3D des sommets des polygones constituant son enveloppe. Pour employer facilement des textures, le secret est de définir sur l'image en 2D un nombre de points correspondant, auxquels on donnera les mêmes numéros d'ordre. Les convertisseurs feront le reste en affichant en perspective des triangles

ou des polygones découpés dans cette image, et en leur faisant éventuellement subir des déplacements, des mises à l'échelle ou des rotations, exactement de la même façon que pour des formes non texturées.

Les coordonnées définies sur la texture sont bien sûr exprimées en deux dimensions et non en trois (par convention, on nomme généralement ces coordonnées *u* et *v* pour ne pas les confondre avec *x* et *y*), mais il y a une autre différence importante : les unités employées ne désignent pas des mètres ni d'autres unités de longueur, mais un positionnement proportionnel sur l'image. Le point de coordonnées u = 0.0 et v = 0.0 désigne le coin inférieur gauche de l'image, le point de coordonnées u = 1.0 et v = 0.0 désigne le coin inférieur droit, le point de coordonnées u = 0.5 et v = 0.5 désigne le centre de l'image, et ainsi de suite, sans considérer le moins du monde si l'image a la forme d'un carré ou d'un rectangle horizontal ou vertical.

Cela étant dit, la syntaxe SGML 3D pour l'emploi de textures n'a plus rien pour surprendre. Il suffit de savoir que les balises `<texture>` et `</texture>` sont inscrites pour encadrer le nom du fichier Jpeg employé comme texture, et que les balises `<pointsUV>` et `</pointsUV>` encadrent les coordonnées *u* et *v* des points définis sur l'image.

Voici à titre d'exemple le code nécessaire à la représentation d'une affiche verticale de 1.2 m de largeur et de 1.8 m de hauteur, disposée à partir du niveau du sol dans le plan XZ à la verticale du point O (cet exemple type peut être facilement adapté pour intégrer n'importe quelle photographie dans une scène 3D) :

```
<texture>
affiche.jpg
</texture>

<pointsUV>
1, 0.0, 0.0
2, 1.0, 0.0
3, 1.0, 1.0
4, 0.0, 1.0
</pointsUV>

<points3D>
1, -0.6, 0.0, 0.0
2, 0.6, 0.0, 0.0
3, 0.6, 0.0, 1.8
4, -0.6, 0.0, 1.8
</points3D>

<faces>
```

```
1, 2, 3, 4
</faces>
```

Ce code est certes un peu bavard (encore qu'il le soit beaucoup moins que sa traduction pour POV-Ray !). Avec un peu de pratique, il est néanmoins assez facile à écrire... et encore bien plus à adapter : l'exemple que nous venons de donner est suffisant pour tous les cas où l'on souhaite seulement faire figurer une photo non recadrée dans un rectangle.

Mais le convertisseur `bmp_vers_texture` peut rendre cela considérablement plus simple encore, en permettant de découper la texture exactement de la même façon que celle que nous avons vue précédemment avec le convertisseur `bmp_vers_2d`. Nous ne donnerons que des explications laconiques, celles que nous venons de fournir étant suffisantes pour comprendre la logique du code SGML 3D produit automatiquement par le convertisseur (et qui sera de structure strictement identique à celui que nous venons de voir).

Pour toutes les opérations que nous allons voir, il est plus simple de placer ou d'enregistrer tous les fichiers manipulés (y compris la texture elle-même) directement dans le répertoire de fabrication de la scène.

Nous supposerons pour l'exemple que la texture est nommée `maTexture.jpg`.

1) Ouvrir `maTexture.jpg` avec un logiciel de retouche d'images.

2) En enregistrer une copie au format BMP et sous le nom `maTexture.bmp` (le fichier Jpeg original ne doit pas être modifié, et les opérations suivantes seront effectuées sur cette copie).

3) Rééchantillonner l'image pour que sa plus petite dimension soit de 128 pixels (cela n'est pas nécessaire mais permet un positionnement plus facile des pixels jaunes et cyan).

4) Afficher l'image à 400 %.

5) Prendre l'outil crayon, la brosse pixel et la couleur jaune pure (rouge et vert à 255, bleu à 0).

6) Dessiner autour de la forme que l'on veut afficher un périmètre de pixels jaunes. Exactement comme nous l'avons vu précédemment, cette forme doit être un « patatoïde » grossièrement circulaire, mais rien n'interdit d'enchaîner dans un fichier SGML 3D les définitions de texture pour obtenir des découpages aux formes plus irrégulières.

7) Placer au centre du périmètre grossièrement circulaire un pixel cyan unique (rouge à 0, vert et bleu à 255).

8) Enregistrer, quitter le logiciel de retouche.

9) Dans le répertoire de travail, lancer le convertisseur avec la syntaxe suivante :

```
../bin/bmp_vers_texture maTexture.bmp maTexture.3d
```

Le fichier `maTexture.3d` qui vient d'être produit peut être intégré dans la scène de façon classique. Sans modifications de coordonnées, il apparaîtra sous la forme d'un découpage plat et horizontal de taille proche d'un mètre, au niveau du sol, à proximité du point O. Pour éviter des bizarreries à l'affichage, il est recommandé de le décoller un peu du sol, par exemple comme ceci :

```
<modifcoord>
Posi 0.0, 0.0, 0.002
</modifcoord>
<inclusion>
maTexture.3d
</inclusion>
<modifcoord>
Fin
</modifcoord>
```

Dans tous les cas, les informations de texture seront parfaitement respectées en cas de conversion vers POV-Ray... mais ignorées par une conversion vers Blender. La facilité à gérer les textures avec POV-Ray est d'ailleurs sans nul doute un argument de poids pour le préférer à tout autre logiciel d'image de synthèse.

15.2 Le texte en 3D

On a couramment besoin d'intégrer de courtes mentions textuelles dans une scène (par exemple pour y faire figurer des panneaux d'information : *« Sortie »*, *« Attention à la marche »*, etc.) Le programme `texte_en_triangles`, avec ses compléments inclus dans le sous-répertoire du même nom, peut transformer à cette fin une mention textuelle en un assemblage de polygones peu esthétique, mais suffisamment lisible et peu gourmand en mémoire (et aussi, accessoirement, libre des droits qui affectent la plupart des polices de bonne qualité typographique). Le fichier informatique produit est au format SGML 2D (et même en .2db, puisqu'il ne comporte ni modification de coordonnées, ni inclusion de sous-fichiers), et il ne peut donc être inclus dans une scène 3D avant d'avoir été converti avec `2db_vers_3d`, ou encore avec `extrusion` si l'on souhaite lui donner une épaisseur.

La syntaxe d'appel est relativement simple, à la condition que le programme soit lancé depuis le sous-répertoire `texte_en_triangles` :

```
../bin/texte_en_triangles "SORTIE"
```

crée un fichier `texte.2db` (c'est du moins le nom que lui donne par défaut le fichier d'initialisation `texte_en_triangles.ini`) qui définit tous les triangles nécessaires pour dessiner les lettres du mot *« SORTIE »*. Par défaut, le point O est situé en bas à gauche de la première lettre, mais on peut aussi taper

```
../bin/texte_en_triangles "SORTIE" -c
```

pour que le bas du texte (en termes typographiques, la « ligne de base ») soit centrée sur le point O.

La largeur d'un tel texte est arbitraire, et généralement supérieure à plusieurs unités. On peut imposer une largeur de 1.0 (qui facilitera ensuite les modifications de coordonnées, puisqu'il suffira d'imposer la largeur désirée comme facteur de mise à l'échelle, sans avoir à tâtonner ni à effectuer de règle de trois). Cela se fait avec la syntaxe :

```
../bin/texte_en_triangles "SORTIE" -1
```

ou encore, pour obtenir un texte d'une largeur de 1.0 mais centré sur le point O, donc compris entre x = -0.5 et x = 0.5 :

```
../bin/texte_en_triangles "SORTIE" -1c
```

Enfin, on peut obtenir un texte disposé sur plusieurs lignes en intercalant le code traditionnel en informatique pour représenter le retour à la ligne (la lettre n précédée d'un *backslash*). Par exemple :

```
../bin/texte_en_triangles "Il est dangereux\nde se pencher
au dehors"
```

Dans l'hypothèse probable où l'on voudrait plaquer une telle mention textuelle sur un fond de couleur dans une scène 3D, il ne faudrait pas oublier de l'en détacher ne fût-ce que d'un demi-millimètre, pour éviter des bizarreries d'affichage.

Signalons aux curieux que les nombreux petits fichiers .2d du répertoire `texte_en_triangles` sont eux-mêmes des fichiers SGML 2D. Il serait donc relativement facile de modifier le dessin de ces lettres ou d'en créer de nouvelles (le signe de l'euro, par exemple), mais c'est une autre histoire (il faudrait aussi modifier le fichier `chasses.txt`, voire des paramètres du fichier d'initialisation `texte_en_triangles.ini`).

15.3 Quelques ouvrages de complément

Gérard Swinnen (2004), *Apprendre à programmer avec Python,* O' Reilly, ISBN 2-84177-294-2.

Alex Martelli (2004), *Python en concentré*, O' Reilly, ISBN 2-84177-290-X.

Jean-Pierre Couwenbergh (1998), *La synthèse d'images,* Marabout informatique, ISBN 2-501.03039-7.

Oscar Riera Ojeda et Lucas H. Guerra (1999), *Modèles virtuels d'architecture,* Evergreen (Benedikt Taschen Verlag GmbH), ISBN 3-8228-7107-9.

Benoît Rittaud (2000), *La géométrie classique,* Le Pommier-Fayard, ISBN 2-746-50037-X.

Neil Matthew et Richard Stones (2001), *Programmation Linux,* Eyrolles, ISBN 2-212-09129-X.

15.4 Logiciels employés pour la réalisation de cet ouvrage

A la seule exception de POV-Ray (dont la philosophie n'est pourtant guère différente), cet ouvrage a exclusivement été conçu avec des logiciels libres, pour la plupart liés au projet GNU/Linux. La rédaction et la saisie ont été effectuées principalement avec GNU/Emacs, et parfois (sous Windows) avec Notepad2. Nous avons abondamment eu recours à LaTeX dans l'élaboration du brouillon, en particulier pour la numérotation et le référencement des figures. La plus grosse partie de la cotation du texte a été effectuée avec OpenOffice. Les images bitmap ont été retravaillées avec Gimp, les images vectorielles avec Skencil. Les convertisseurs et autres utilitaires du CD-ROM joint ont été compilés avec GCC pour Linux, Mac OS/X et Cygwin. Le master du CD-ROM joint a été gravé avec K3B.

Les outils de mise en page des éditions Dunod sont les seuls logiciels non libres employés pour la réalisation de cet ouvrage.

15.5 Remerciements

Le peu de géométrie que je sais m'a été transmis par le regretté M. Passelergue et par Mlle Graviche, mes professeurs de mathématiques en classes de seconde et première ; je leur suis d'autant plus reconnaissant que mon prof de troisième, lui, n'avait jamais rien su me faire comprendre de cette discipline, et s'était même exclusivement consacré à m'en dégoûter (honte aux enseignants qui découragent l'intelligence de leurs élèves). Je remercie aussi le capitaine d'artillerie Didier Debray, dont les tonitruantes harangues lors d'exercices de topographie sous le soleil du Var m'ont fait comprendre à quel point il est honteux de s'enfuir devant une difficulté trigonométrique !

Je n'aurais jamais eu le courage de me mettre à la programmation sans les explications de Pierre Caumon, et je me serais certainement laissé rebuter par Linux et le langage C sans celles de Kavous Bojnourdi.

Merci enfin à Thierry Stoehr pour m'avoir fait connaître LaTeX. Puisse ce médiocre ouvrage transmettre un peu du savoir et de la pédagogie de toutes ces excellentes personnes.

J'ennuierais le lecteur en citant les noms de tous ceux qui m'ont encouragé, voire houspillé, pour que je me décide à mener ce manuscrit à son terme. Je rends quand même particulièrement hommage à la persévérance de Didier Sanz, qui m'a incité à m'y mettre pendant plusieurs années, et à la patience d'Isabelle Charansol, qui m'a laissé Dieu sait combien de fois squatter son Macintosh pour y effectuer d'innombrables tests et compilations de listings.

Index

Table des matières