2

Principes généraux

Dans ce chapitre, je vais expliquer quelques concepts de base pour comprendre schématiquement comment dessiner une image avec OpenGL et comment structurer les programmes.

2.1. Définition des objets à dessiner

La synthèse d'images consiste à dessiner des objets 3D, c'est-à-dire des formes définies dans l'espace. Ces formes ont une structure géométrique et une apparence qui est produite par un matériau réagissant avec la lumière. L'ensemble de ces objets, avec les sources de lumière et la caméra, est appelé la *scène 3D*. Je reviendrai très progressivement et en détails sur tous ces points dans chacun des chapitres du livre, mais voyons cela dans les grandes lignes en commençant par les objets 3D.

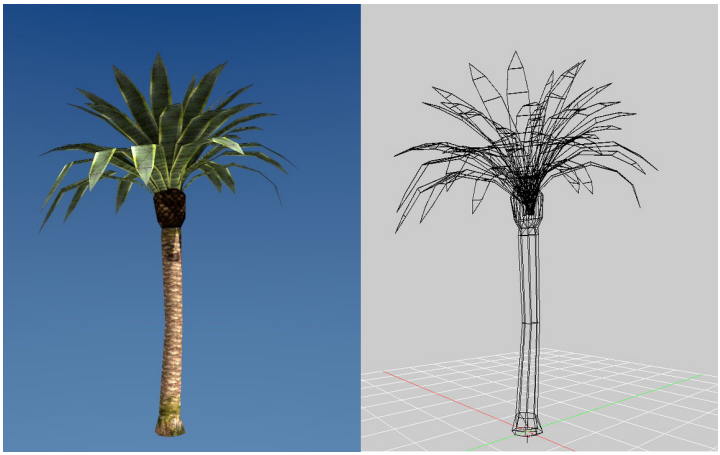
Formes et matériaux

Dans les API de dessin 3D en temps réel, les objets sont définis à l'aide de polygones plats, en général des triangles. Si les objets nous paraissent arrondis, c'est que les polygones qui les composent sont tout petits et assez nombreux. Le travail principal d'OpenGL et de la carte graphique consiste à dessiner tous ces polygones le plus rapidement possible.

La Figure 2.1 montre à droite les polygones constituant la structure géométrique de l'objet 3D à gauche. On peut remarquer quelques triangles et beaucoup de quadrilatères, notamment dans le tronc de l'arbre.

Les polygones, et en particulier les triangles, sont les formes géométriques les plus utilisées pour dessiner. Les polygones comme les quadrilatères sont tous automatiquement décomposés en triangles pour le dessin. OpenGL est également capable de dessiner des points isolés ainsi que des segments de droite. Ces formes très simples sont appelées *primitives géométriques*. Le chapitre Optimisation du dessin vous montrera qu'il existe quelques combinaisons de ces formes simples, par exemple pour regrouper des triangles autour d'un même point (voir notamment la Section 6.1, Primitives de dessin).

Figure 2.1 : *Structure en polygones d'un objet 3D*



Les scènes 3D réalistes n'emploient principalement que des triangles. Les points et segments y sont extrêmement rares. On les trouve plutôt dans les logiciels de conception 3D pour dessiner des objets en "fils de fer" comme le palmier de droite de la Figure 2.1. C'est pourquoi, dans la suite, je ne ferai référence qu'aux triangles, mais tout ce qui est écrit s'applique aussi aux autres primitives graphiques.

Les primitives géométriques sont définies par des coordonnées 3D x,y,z ainsi que je vous l'expliquerai au chapitre Premiers programmes OpenGL. Ce sont les extrémités des polygones. On les appelle *sommets*, appelés également *vertices* au pluriel et *vertex* au singulier. Il faut une quantité considérable de sommets et de triangles pour dessiner une scène complexe, puisque par exemple le palmier de cette figure compte 947 sommets et 876 triangles (après décomposition des quadrilatères en triangles).

Les sommets d'un objet peuvent être *transformés*, c'est-à-dire que leurs coordonnées sont recalculées, par exemple pour les déplacer à un autre endroit. Cela permet de redimensionner, de faire tourner, de déplacer un objet sans devoir le recréer. À la fin de cette phase, les sommets sont projetés sur l'écran, c'est-à-dire que le système calcule leur position sur l'écran, en deux dimensions, compte tenu de la mise en scène (rotations, déplacements et autres transformations), ainsi que de la perspective choisie pour le point de vue.

Les triangles sont colorés par un matériau. Pour ce palmier, il y a un matériau pour le tronc et un autre pour les feuilles. Un matériau définit la couleur de chaque point des triangles : vert clair, vert foncé, marron, etc. Les matériaux sont généralement sensibles

à la lumière. Ils sont plus ou moins clairs selon l'intensité et la direction de l'éclairage. C'est l'objet de calculs parfois complexes qui sont expliqués dans la partie Matériaux.

Mais, avant de poursuivre sur OpenGL, voici des explications sur différentes manières d'effectuer le *rendu* d'une image de synthèse, c'est-à-dire de calculer les couleurs des pixels affichés sur l'écran en fonction des primitives (triangles, segments et points) de la scène.

Algorithmes de synthèse d'images

L'algorithme simplifié d'OpenGL est le suivant :

```
pour chaque primitive de la scène {
    calculer la position écran des sommets vus de la caméra
    colorer les pixels occupés par la primitive
                    en fonction du matériau et de la lumière
}
```

Le principe est de projeter successivement les primitives géométriques sur l'écran afin de remplir et colorer les pixels concernés. Cette méthode demande un mécanisme supplémentaire pour que le résultat soit correct. Puisqu'on dessine les triangles successivement, il faut s'assurer que ceux du fond n'effacent pas ceux de l'avant-plan, même s'ils sont dessinés ultérieurement. L'ordre du dessin dépend de l'ordre de parcours des primitives dans la boucle extérieure, et malheureusement pas de l'éloignement par rapport à l'écran. Je vous montrerai comment faire avec OpenGL dans la section Superposition indépendante de l'ordre de dessin.

Une autre famille d'algorithmes suit une logique inverse par rapport à l'algorithme précédent :

```
pour chaque pixel de l'image {
    chercher la primitive la plus proche de la caméra
                    auquel appartient ce pixel
    colorer ce pixel en fonction du matériau et de la lumière
}
```

C'est-à-dire qu'au lieu de parcourir les triangles, lignes et points à la recherche des pixels sur lesquels ils sont projetés, on parcourt les pixels en cherchant quels formes de la scène les contiennent et on s'intéresse à la plus proche d'entre elles qui est sur la ligne de visée. Ce second algorithme est appelé *lancer de rayons*.

Cette seconde méthode permet d'appliquer les lois de l'optique relativement facilement et avec une très grande précision, par exemple calculer les reflets, les transparences et

les ombres portées. En effet, le calcul effectué sur chaque pixel, pour calculer sa couleur correspond directement avec la notion physique de rayon lumineux et dans ce cadre, il est très simple de calculer la couleur de rayons réfléchis et réfractés.

Le premier algorithme, celui d'OpenGL n'est pas approprié pour faire ces calculs optiques. En revanche, il est très intéressant pour ses faibles temps de calculs et ses résultats visuels convenables. Inversement, le lancer de rayons est très lent parce qu'il faut faire de très nombreuses recherches parmi toutes les primitives.

Note *› Une troisième méthode, appelée* calcul des radiosités, *est beaucoup plus complexe. Elle consiste à calculer l'équilibre lumineux dans la scène. Cet équilibre est analogue à l'équilibre thermique qui s'établit par exemple dans une salle de bains équipée d'un chauffage infrarouge. Lorsqu'on allume le chauffage, il commence par chauffer les différentes surfaces qui lui font face, puis celles-ci rayonnent à leur tour et les autres surfaces s'échauffent peu à peu jusqu'à arriver à une température stable. Dans le cas d'une scène 3D, les surfaces lumineuses envoient de la lumière aux surfaces réfléchissantes ; celles-ci réémettent alors une partie de cette lumière vers les autres surfaces en fonction de leurs orientations et des occultations, et ainsi de suite. Un équilibre s'établit et chaque surface est alors plus ou moins éclairée. L'équilibre est presque instantanément obtenu dans la nature, tandis qu'il faut des heures pour le calculer sur ordinateur. Toutefois, certaines simplifications permettent d'en approcher le résultat et de l'intégrer à une synthèse faite par OpenGL. Nous verrons cela dans la* *Section 19.4, Occlusion ambiante.*

2.2. Pipeline OpenGL

Pour résumer ce qui précède en une phrase, OpenGL est un logiciel spécialisé dans le dessin à haute vitesse d'un grand nombre de triangles.

Pour bien comprendre OpenGL, il faut s'intéresser aux aspects matériels de bas niveau. Les cartes graphiques contemporaines ont énormément évolué ces dernières années, ce qui a entraîné des changements dans OpenGL. Certaines fonctionnalités assez commodes qui existaient dans les deux premières versions majeures d'OpenGL sont maintenant déconseillées (*deprecated*) ou ont été carrément supprimées parce que trop difficiles à réaliser matériellement, ou trop lentes. Et inversement, de nouvelles possibilités sont apparues, mais elles sont très liées aux aspects matériels internes des cartes graphiques, alors qu'on cherche plutôt à s'abstraire des réalités matérielles dans les API et langages de programmation.

Nous devons donc nous pencher sur le fonctionnement des cartes graphiques pour comprendre ce qu'est OpenGL et comment il applique l'algorithme de la section précédente.