1/3/1993

Travail de Diplôme. Laboratoire d'Infographie, EPFL. Calcul d'interréflexions diffuses (Radiosité).

Guy Moreillon

Assistant: Enrico Gobbetti
Professeur: Daniel Thalmann

Introduction

La recherche en infographie, et plus précisément en rendu d'image, tends principalement vers un but: obtenir, par simulation, des images synthétiques d'objets quelconques dont la qualité et le réalisme s'approche le plus possible de celui des images réelles. Une des applications de telles images est la représentation de bâtiments d'après leur plans, avant leur construction.

Le but de ce projet est donc de fournir à l'utilisateur un moyen de visualiser de manière réaliste une construction architecturale modélisée informatiquement. Cela implique une simulation de l'illumination dans le modèle aussi proche que possible de la réalité. Or, pour calculer au mieux ces effets d'illumination, qui sont principalement de nature diffuse, la technique de la radiosité est certainement la plus appropriée. Elle permet en effet d'approximer aussi bien les illuminations directes que les illuminations indirectes, ce qui est indispensable dans la représentation d'un bâtiment, où la lumière ambiante et l'atmosphère lumineuse ont une grande importance.

De plus, afin d'améliorer la qualité de la visualisation, il est intéressant de pouvoir se déplacer interactivement dans le modèle, et mieux encore, de pouvoir interagir avec lui, en le modifiant à volonté. L'interaction avec la scène et sa manipulation est donc le deuxième but de ce projet.

En conclusion, l'application finale doit permettre d'accepter une description géométrique d'une scène et d'en calculer l'illumination en tenant

compte des interréflexions diffuses, mais pas des transparences ou des réflexions spéculaires. De plus elle doit permettre de visualiser la scène sous tous les angles de manière interactive. Enfin elle doit donner à l'utilisateur la possibilité de modifier les caractéristiques non géométriques des objets, de manière interactive également. Cette dernière restriction quant à la géométrie des objets est imposée par la technique de calcul envisagée qui nécessite de reprendre le calcul au départ si un objet et déformé ou déplacé, ce qui ne serait pas envisageable dans une application interactive.

Ce projet étant relativement important, il a été réalisé en deux parties, la première pendant le semestre d'été 1991, la seconde pendant le travail de diplôme en automne 1991. Pour des raisons de cohérence, ce rapport traite des deux parties simultanément.

Première partie

Dans la première étape, il s'agit d'écrire la base de l'application, c'est à dire les routines de calcul. Le résultat est un programme qui prend en entrée une description de scène et qui produit un fichier contenant la liste des polygones formants celle-ci, avec la couleur de chaque sommet de chaque polygone. Ce fichier peut alors être fourni à *gview* (application de démonstration de Silicon Graphics) pour visualiser la scène sous tous les angles.

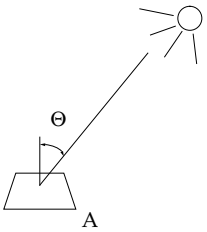
Deuxième partie

Dans la deuxième étape, l'application est rendue interactive. Il s'agit de donner la possibilité a l'utilisateur de se déplacer dans la scène pendant le calcul et de modifier les caractéristiques des objets interactivement. Ces caractéristiques comprennent la couleur et la texture, le coefficient de réflectivité du matériau ainsi que la constante d'émission de l'objet. Les effets dus à la modification de l'une de ces caractéristiques doivent se répercuter dans la scène aussitôt la modification effectuée.

Description des algorithmes de calcul

Algorithmes de calcul

Calcul de l'équilibre énergétique



On utilise ici un modèle d'illumination globale, où chaque surface émet et/ou réfléchit de la lumière (énergie). L'intensité de la lumière réfléchie par une surface mate, selon le modèle l'illumination diffuse de Lambert, est exprimée par [1]:

$$I_{out} = \rho \int_{2\pi} I_{in}(\Theta) \cos\theta d\omega \qquad \text{(Eq. 1)}$$

où:

I_{out} = Intensité de la lumière réfléchie
ρ = Réflectivité diffuse de la surface
θ = Angle entre la normale de la surface et la direction de la lumière incidente Θ
$I_{in}(\Theta)$ = Intensité de la lumière arrivant dans la direction Θ

Dans le cas d'un modèle d'illumination locale, le calcul de l'expression est simple, puisque la lumière provient uniquement des sources de lumière. Dans notre cas cependant, il faut tenir compte de la lumière émanant de toutes les surfaces de la scène, parfois de manière indirecte.

Considérons la scène comme étant un assemblage de morceaux de surface (ou *patch*). La relation entre la *radiosité* (quantité d'énergie émanant d'une surface par unité de surface et de temps) d'une patch et la radiosité de toutes les autres patchs est exprimée par [1]:

$$B_i A_i = E_i A_i + \rho_i \sum_{j=1}^{n} B_j F_{ji} A_j \qquad \text{(Eq. 2)}$$

où:

B_i = Radiosité de la patch i
E_i = Emission de la patch i
A_i = Aire de la patch i
F_{ji} = Facteur de forme de j à i
ρ_i = Réflectivité de la patch i
n = Nombre de patchs

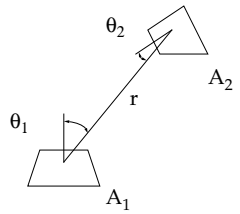
Un facteur de forme correspond à la quantité d'énergie quittant une surface i pour arriver sur une surface j. D'après la réciprocité des facteurs de forme, on a [2]:

$$F_{ij} A_i = F_{ji} A_j \qquad \text{(Eq. 3)}$$

et en divisant (Eq. 2) par A_i, on obtient [2]:

$$B_i = E_i + \rho_i \sum_{j=1}^{n} B_j F_{ij} \qquad \text{(Eq. 4)}$$

ou, sous forme de matrice [2]:



$$\begin{bmatrix} 1-\rho_1 F_{11} & -\rho_1 F_{12} & \cdots & -\rho_1 F_{1n} \\ -\rho_2 F_{21} & 1-\rho_2 F_{22} & \cdots & -\rho_2 F_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ -\rho_n F_{n1} & -\rho_n F_{n2} & & 1-\rho_n F_{nn} \end{bmatrix} \begin{pmatrix} B_1 \\ B_2 \\ \cdots \\ B_n \end{pmatrix} = \begin{pmatrix} E_1 \\ E_2 \\ \cdots \\ E_n \end{pmatrix} \qquad \text{(Eq. 5)}$$

avec, pour les facteurs de forme [3][4]:

$$F_{A_iA_j} = \frac{1}{A_i}\int_{A_i}\int_{A_j}\frac{\cos\theta_i\cos\theta_j}{\pi r^2}dA_j dA_i \qquad \text{(Eq. 6)}$$

A partir de ces résultats, deux méthodes d'évaluation d'un solution au système d'équations sont envisageables. La première consiste à évaluer la totalité des facteurs de forme F_{ji}, résoudre le système grâce à la méthode de Gauss-Siedel ou une autre méthode similaire, puis afficher le résultat du calcul [4]. Cette technique a plusieurs désavantages. Elle occupe un espace mémoire important (en $O(n^2)$, où n est le nombre de patchs) et elle n'est pas adaptée à une application interactive, puisqu'il faut attendre la fin du calcul pour pouvoir visualiser la scène.

La seconde technique est une méthode itérative [2]. Son algorithme est le suivant:

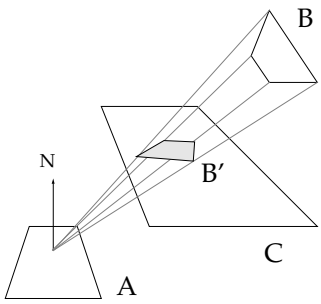
1. Déterminer la patch qui émet la plus grande quantité d'énergie
2. Calculer son effet sur toutes les autres patchs, en calculant les facteurs de forme nécessaires au passage
3. Recommencer au point 1 tant qu'il reste de l'énergie à distribuer

Il est alors aisé d'insérer dans ce schéma les opérations nécessaires à l'interaction avec le calcul de la scène [7]. On obtient ainsi:

1. Afficher l'état courant de la scène, et effectuer les modifications demandées par l'utilisateur sur les objets
2. Déterminer la patch qui émet la plus grande quantité d'énergie (en valeur absolue!)
3. Calculer son effet sur toutes les autres patchs, en calculant les facteurs de forme nécessaires au passage
4. Recommencer au point 1 tant qu'il reste de l'énergie a distribuer

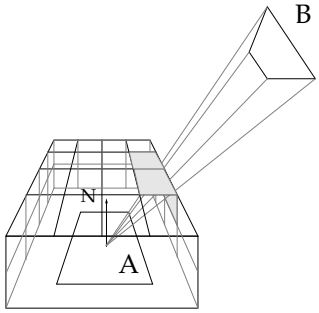
Les quantités d'énergie considérées ici peuvent être négatives aussi bien que positives. En effet, si la constante d'émission d'un objet est diminuée pendant le calcul après que la contribution originale de cet objet aie été prise en compte, la quantité d'énergie à répartir dans la scène en provenance de cet objet est alors négative. Le fait que l'on choisisse comme patch émettrice la patch qui émet le plus d'énergie permet d'avoir rapidement une bonne estimation de l'illumination globale de la scène. Par la suite, plus le calcul avance, plus l'estimation est précise.

Cette méthode permet de visualiser la scène au fur et à mesure du calcul de son illumination, et d'obtenir rapidement une nouvelle solution lors d'une modification locale. En effet, si la première méthode est utilisée (calculer une solution exacte en une fois), une modification, si infime soit-elle, implique la réévaluation totale de la solution, bien que son effet sur la scène soit limité à un nombre restreint d'objets alentours. Avec la méthode incrémentale par contre, lorsqu'une modification est effectuée, c'est uniquement l'effet de la modification qui est calculé et qui est additionné à la solution courante. Ce calcul est naturellement beaucoup plus rapide que le calcul de la scène entière.

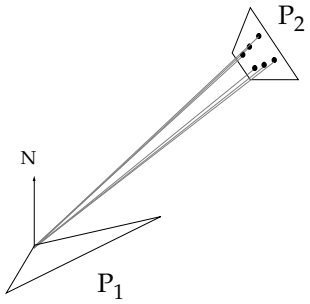


Calcul de la radiosité et des ombres portées

Pour le calcul de l'effet d'une patch sur une autre, là encore, deux méthodes sont possibles. La première, la plus ancienne, consiste à utiliser la technique de l'hémicube [4]. Cette méthode de calcul est basée sur le fait que le facteur de forme entre un morceau de surface A et un morceau de surface B est égal au facteur de forme entre le morceau de surface A et la projection B' de B sur une surface quelconque C placée dans le faisceau défini par AB (voir figure). Afin de profiter de cette propriété géométrique, on place la moitié supérieure d'un cube (ou hémicube) sur la patch réceptrice courante. La surface de l'hémicube est subdivisée en un certain nombre d'éléments (pixels d'hémicube) pour lesquels le facteur de forme avec la patch est précalculé. On projette alors la patch source sur l'hémicube de la patch réceptrice et on additionne les facteurs de forme de tous les pixels recouverts par la surface projetée. On obtient ainsi une estimation du facteur de forme pour ces patchs. Une fois la radiosité de chaque patch calculée, la radiosité aux sommets de chaque patch est estimée en prenant la moyenne des radiosités des patchs adjacentes, et la scène est visualisée en faisant un shading de Gouraud sur ces patchs. Cette méthode comporte cependant des désavantages. Des problèmes d'aliasing peuvent apparaître au niveau des pixels de l'hémicube et causer des irrégularités visibles dans l'illumination d'une surface.



La seconde méthode possible est basée sur le lancer de rayon [1]. Afin d'éviter les problèmes d'aliasing, les radiosités sont calculées directement aux sommets des patchs. Les ombres portées sont calculées par lancer de rayon. D'après [1], la radiosité en un point 1 due à l'illumination par une surface 2 est exprimée par:



$$B_1 = \rho_1 B_2 \frac{A_2}{n} \sum_{i=1}^{n} \delta_i \frac{\cos\theta_{1i} \cos\theta_{2i}}{\pi r^2 + \frac{A_2}{n}} \qquad \text{(Eq. 7)}$$

où:

- B_1 = La radiosité au point 1
- B_2 = La radiosité au point 2 (source)
- ρ_1 = La réflectivité de la surface 1
- A_2 = L'aire de la surface 2
- n = Le nombre d'échantillons pris sur la source
- δ_i = La fonction de visibilité pour l'échantillon i

La valeur de δ_i est déterminée en lançant un rayon depuis le point 1 vers le point i choisi sur la source 2. Si le rayon est intercepté par un objet quelconque, le point 1 est dans l'ombre portée du point i de la source et δ_i est nulle, alors que si le rayon atteint la surface 2, le point 1 est éclairé par l'élément de surface autour du point i sur la surface 2 et δ_i vaut 1 (pas d'ombre portée à cet endroit).

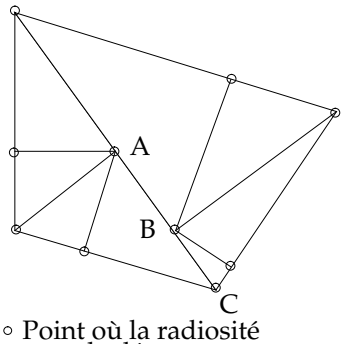
Algorithme choisi

Etant donné la nature interactive de l'application finale, la méthode itérative s'impose par rapport à une méthode de calcul global. D'autre part, les problèmes d'aliasing soulevés par la méthode de l'hémicube ont dirigé le choix de la méthode de calcul des radiosités (ou plus précisément des ombres portées) vers le lancer de rayon.

Subdivision adaptative

Contrairement à la plupart des programmes basés sur la radiosité, les patchs, c'est à dire les polygones constituant les objets pures, ne sont pas ici des carrés, mais des triangles. Il s'agit en effet du plus petit élément de surface plane définissable exclusivement à l'aide de points. Il a l'avantage de ne pas poser de problèmes de coplanarité entre les points, comme ce peut être le cas avec des carrés, et il permet de construire ou d'approximer n'importe quelle surface.

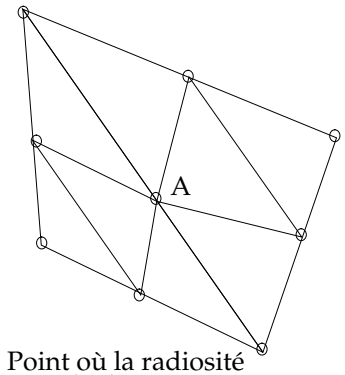
Pour calculer l'effet d'une patch émettrice sur une autre patch, on calcule l'effet de la patch émettrice sur chaque sommet du triangle (patch) récepteur. Une interpolation est ensuite faite à l'affichage entre ces trois couleurs. Cette méthode est efficace si les patchs sont petites et que la différence de radiosité entre les sommets de la patch est faible, mais lorsque celle-ci est plus grande, il est nécessaire d'avoir plus d'information sur la façon dont varie cette radiosité sur la surface de la patch. Pour cela, il faut la subdiviser récursivement jusqu'à obtenir soit une patch sur laquelle la radiosité varie dans des limites raisonnables, soit une patch de taille minimale. Le critère forçant une subdivision est simple: il suffit que la différence de radiosité entre une paire de sommets du triangle soit supérieure à une limite fixée. Si c'est le cas, les trois côtés sont subdivisés.



◦ Point où la radiosité est calculée

Figure 1

Lorsqu'un triangle est subdivisé, il est important qu'il le soit de manière unique afin de ne pas avoir de discontinuité d'illumination entre ce triangle et son voisin (voir figure 1). En effet, la radiosité au point A et au point B est nécessairement différente, et cela se verra au moment de l'affichage, car le gradient de radiosité entre le point A et le point C ne sera pas le même qu'entre le point B et le point C. Aussi un triangle est-il subdivisé en quatre éléments en créant trois nouveaux points et en les reliant entre eux (voir figure 2). Cela ne garanti pas une continuité de l'illumination entre deux polygones, car aucune information de connectivité n'est maintenue entre les éléments d'un objet, mais au niveau des subdivisions, on obtient effectivement un gradient uniforme entre les éléments d'une subdivision. De plus, le point A étant partagé par les deux éléments de la subdivision, sa radiosité ne doit être calculée qu'une seule fois.



◦ Point où la radiosité est calculée

Figure 2

Un problème de discontinuité dans l'illumination à la frontière de deux subdivisions peut cependant se présenter dans le cas où un seul côté d'un triangle nécessite une subdivision. Puisque c'est le triangle qui est subdivisé, les autres côtés sont également coupés en deux. Mais dans ce cas, les valeurs de radiosité des points crées sur ces côtés ne doivent pas être recalculés sous peine d'introduire une discontinuité. En effet (voir figure 3), il est possible que le segment AB nécessite une subdivision sans que cela soit nécessaire pour AC. Le triangle ABC est néanmoins subdivisé. Si

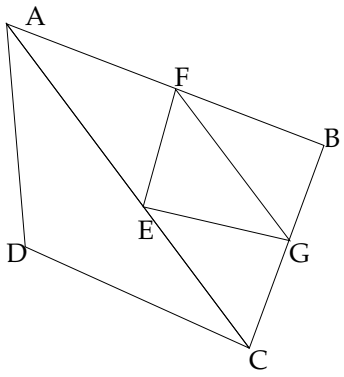


Figure 3

l'on admet à présent que le triangle ACD ne nécessite aucune subdivision, on aura alors à l'affichage une interpolation linéaire de l'intensité lumineuse entre A et C du côté du triangle ACD, mais une interpolation différente sur le même segment du côté du triangle ABC puisque la valeur de la radiosité au point E aura été calculée et non interpolée. C'est pour cette raison que lorsqu'un triangle est subdivisé, seuls les points médians des segments qui doivent être subdivisés voient leur radiosité calculée, les autres recevant une radiosité estimée par la moyenne de la radiosité aux deux extrémités du segment.

Le manque d'informations de connectivité entre les polygones vient du fait qu'il est difficile de leur faire partager des points ou des côtés et cela pour des raisons de normales. En effet, considérons deux polygones partageant un côté. Admettons que ces deux polygones aient la même direction de normale, c'est à dire qu'ils se trouvent dans un plan (voir figure 4a). Il est évident que ces deux polygones peuvent se partager les deux points aux extrémités du côté, y compris la radiosité en chacun de ces points. Dans ce cas, aucun problème de discontinuité d'un polygone à un autre ne se pose.

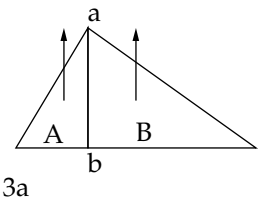


Figure 3a

Considérons maintenant les mêmes deux polygones, mais cette fois avec des normales différentes (voir figure 4b). Dans ce cas, la radiosité au point *a* lorsque ce point appartient à la face A est différente de la radiosité au même point *a*, lorsque celui-ci appartient à la face B, en raison des normales différentes. Le côté ne peut alors pas être partagé par les deux polygones, puisque les radiosités aux points extrêmes sont différentes.

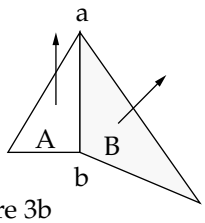
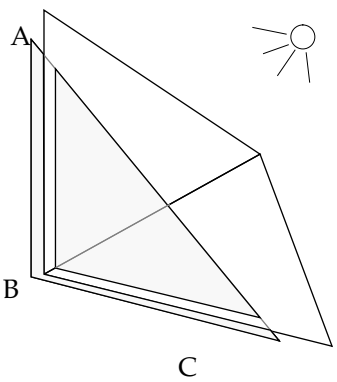


Figure 3b

On ne peut donc prévoir une connectivité qu'entre les polygones voisins de même normale, ce qui implique, pour dicerner les cas, l'introduction d'une phase de préparation des données ainsi que des structures de données plus complexes. Cette sophistication n'a pas été envisagée ici, car elle rejoint un problème plus vaste de création, pré-traitement et traitement des subdivisions expliqué ci-dessous.

Un autre problème survient en effet avec le procédé automatique de subdivision: il existe des cas dans lesquels le polygone devrait être subdivisé, mais ne l'est pas parce que l'évaluation des radiosités aux sommets, tout en étant correcte, n'implique pas une subdivision. Par exemple, considérons le cas très simple d'un grand triangle au dessus duquel est placée une source de lumière. Si la source de lumière est proche et centrée par rapport au triangle, ou de faible intensité, la différence de radiosité au trois sommets du triangle sera suffisamment faible pour ne pas engendrer une subdivision automatique de ce triangle. Pourtant, il est évident qu'elle devrait être faite, pour tenir compte de l'illumination plus intense au centre du triangle.



D'autres cas plus délicats se présentent lorsque les sommets du polygone sont cachés par d'autres alors que physiquement ils ne devraient pas l'être. C'est souvent le cas dans les scènes prévues pour d'autres systèmes de rendu où les polygones s'intersectent alors qu'ils devraient se rejoindre au même points. Dans la figure, les points A, B et C ne sont pas illuminés par la source de lumière, par conséquent tout le triangle est sombre alors qu'il devrait être clair.

Une solution à ce problème est de mesurer les valeurs de radiosité à l'*intérieur* du triangle et ne pas se limiter à ses sommets. Ainsi, bien que le résultat obtenu ne soit pas exact (dans le deuxième cas d'erreur en particulier), on obtiendra une bien meilleure estimation de la situation réelle. Cette méthode a cependant un gros désavantage: elle consomme énormément de temps, puisque pour chaque échantillon calculé à l'intérieur du triangle, il est nécessaire d'échantillonner la patch émettrice, ce qui signifie environ une dizaine de radiosités point à point à calculer, et autant de rayons à lancer dans la scène.

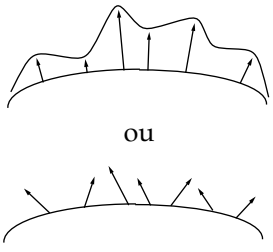
Une autre amélioration consisterait à tenter de découper les polygones originaux de telle manière que les subdivisions suivent plus exactement les limites des zones d'ombre, évitant ainsi des ombres en 'dents de scie' telles qu'on en voit dans certains cas pathologiques. De plus, une telle technique générerait moins de subdivisions que celle présentée au dessus. Ces deux raffinements sont cependant plus complémentaires que mutuellement exclusifs, l'un pouvant être utilisé dans certains cas plutôt que l'autre et vice-versa.

Trouver des solutions à ces problèmes de gestion des subdivisions et de la géométrie des objets demande un travail de recherche approfondi, en dehors des limites de ce projet. Le bénéfice d'un tel travail serait cependant non négligeable puisqu'il pourrait aboutir à un processus automatique de subdivision capable de générer un découpage beaucoup plus efficace et surtout beaucoup plus exact de la scène. Le calcul de l'illumination, qui est avant tout basé sur la géométrie des polygones en serait naturellement grandement amélioré.

Application de textures

Le réalisme d'une image peut être grandement accru par l'application de textures aux objets, aussi l'utilisation de textures a-t-elle été considérée pour cette application.

Les textures correspondent à des déformations géométriques ou chromatiques *localisées*, voir *microscopiques*, des surfaces. La surface d'un tissus par exemple, bien que lisse à une échelle macroscopique, ne l'est pas du tout à l'échelle microscopique. Cette rugosité est une texture de surface. D'autre part, bien que le tissus aie une teinte générale, sa couleur varie en fonction des motifs imprimés dessus. Ces variations forment une texture de couleur.

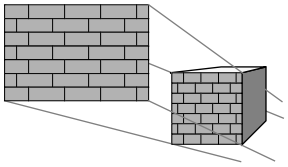


Texture de normale

Ces deux types de textures sont simulées en infographie par deux techniques appropriées.

La première est une méthode qui apporte de petites modifications à la surface de l'objet à texturer, ou plus simplement à la normale de cette surface. Ces modifications ont pour effet de moduler légèrement la valeur de l'intensité lumineuse calculée sur la surface, donnant ainsi un effet de rugosité à l'objet.

La seconde consiste à modifier la couleur de l'objet à texturer, au lieu de la normale. Cela permet de donner des textures de marbre ou de bois à un



Texture de couleur

objet par exemple. Les textures sont définies dans l'espace aussi bien que dans le plan, cependant ces dernières donnent parfois de moins bons résultats car il est nécessaire de les projeter d'une manière ou d'un autre sur la surface de l'objet alors que les premières sont définies en tout point de l'espace.

Les textures de perturbations de la surface ne sont cependant pas très intéressantes dans notre cas puisqu'une modification minime de la surface d'un objet ne modifie en rien son émission diffuse d'énergie, c'est à dire sa radiosité (ni en intensité, ni bien sûr, en direction).

Les textures de couleurs, en revanche, offrent des effets plus intéressants puisqu'elles permettent de donner l'impression que l'objet est formé d'un matériau spécifique, comme du bois, du granite ou du marbre. C'est ce type de texture qui est utilisé dans notre cas.

Il existe au moins deux façons de traiter ces textures dans le calcul de la radiosité. Reprenons (Eq. 1):

$$I_{out} = \rho \int_{2\pi} I_{in}(\Theta) \cos\theta d\omega \qquad \text{(Eq. 8)}$$

Ici, l'équation est valable pour une longueur d'onde (couleur) de la lumière quelconque, c'est à dire que le facteur ρ est dépendant de celle-ci. Dans notre modèle, nous ne considérons qu'un nombre limité de longueurs d'ondes (trois) et donc un nombre limité de facteurs. Chacun de ces facteurs est décomposé en deux sous-facteurs:

$$\rho_i = \rho_{obj} \times \rho_{col} \qquad \text{(Eq. 9)}$$

avec:

ρ_{obj} = Le facteur moyen de l'objet
ρ_{col} = Le facteur lié à la longueur d'onde

Lorsqu'une texture de couleur est appliquée à l'objet,(Eq. 9) devient:

$$\rho_i = \rho_{obj} \times f(x, y, z) \qquad \text{(Eq. 10)}$$

où $f(x, y, z)$ est une fonction variant entre 0 et 1 dépendante de la longueur d'onde et des coordonnées du point pour lequel on calcule la lumière émise. $f(x, y, z)$ est la fonction de texture.

La manière la plus exacte de tenir compte des textures consiste à effectivement calculer f pour chaque échantillon pris sur la patch émettrice, et l'utiliser dans le calcul. Cette méthode permettrait des effets spectaculaires comme des projections de diapositives sur un mur en utilisant une source de lumière texturée, cependant les risques d'obtenir des effets d'aliasing dus au nombre limité d'échantillons prélevés sur la patch émettrice et le temps de calcul supplémentaire requis fait que cette solution n'est pas rentable.

La deuxième façon de procéder consiste à prendre pour f une constante égale à la couleur moyenne de la texture employée. En effet, si l'on part

de l'hypothèse que les perturbations de couleurs engendrées par la fonction de texture sont locales en comparaison de la taille de l'objet, on peut définir:

$$F = \frac{1}{V}\int_{V} f(x, y, z) \qquad \text{(Eq. 11)}$$

où V est le volume dans lequel la texture est définie. On peut alors approximer I_{out} par:

$$I'_{out} = \rho_{obj} F \int_{2\pi} I_{in}(\Theta) \cos\theta d\omega \qquad \text{(Eq. 12)}$$

La valeur de radiosité utilisée à l'affichage est alors:

$$I = \frac{f(x, y, z)}{F} \times I'_{out} \qquad \text{(Eq. 13)}$$

Ainsi, si l'hypothèse citée ci-dessus est vérifiée, l'effet recherché sera obtenu sans coût de calcul supplémentaire. Cette méthode est celle qui a été retenue.

Calcul des effets dus aux modifications effectuées par l'utilisateur

L'application finale étant interactive, il faut considérer l'effet des modifications apportées par l'utilisateur à la scène.

Lorsqu'une modification des caractéristiques d'un objet est faite par l'utilisateur, c'est l'effet de cette modification uniquement qui doit être propagé dans la scène: il n'est pas envisageable de recalculer entièrement l'illumination. Il s'agit donc d'évaluer la *différence* entre la radiosité émise par l'objet non modifié et celle émise par l'objet après la modification, pour pouvoir ensuite corriger les valeurs de radiosité des patchs constituant l'objet et propager cette différence dans la scène.

Si l'on pose pour chaque point de l'objet:

B	= La radiosité du point avant la modification
B'	= La radiosité du point après la modification
B_r	= La radiosité totale reçue par le point
∂B	= La différence de radiosité à envoyer dans la scène
ρ	= La réflectivité de la surface avant la modification (Scalaire)
ρ'	= La réflectivité de la surface après la modification
e	= La constante d'émission de la surface avant la modification (Scalaire)
e'	= La constante d'émission de la surface après la modification
C	= La couleur de la surface avant la modification
C'	= La couleur de la surface après la modification

on a alors:

$$B' = e'C' + (\rho'C') \mid B_r \qquad \text{(Eq. 14)}$$

et:

$$\partial B = e'C' - eC + (\rho'C' - \rho C) \mid B_r \qquad \text{(Eq. 15)}$$

avec:

$$A \mid B = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \Bigg| \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} a_1 \times b_1 \\ a_2 \times b_2 \\ a_3 \times b_3 \end{pmatrix} \qquad \text{(Eq. 16)}$$

Les valeurs des composantes de ∂B peuvent bien entendu être négatives, puisqu'il s'agit d'une différence de radiosité.

La prise en compte des effets d'une modification est donc faite de la manière suivante: pour chaque point appartenant à l'objet modifié, on calcule B' et ∂B. On affecte B' comme nouvelle valeur de radiosité à ce point, et on additionne ∂B à la quantité d'énergie que le point doit encore envoyer dans la scène. Cette valeur est initialement égale à la valeur d'émission du point, et plus tard dans le calcul, à la quantité d'énergie renvoyée par la surface en ce point. Ainsi, la prochaine fois que la patch contenant ce point deviendra la patch émettrice, elle aura une quantité d'énergie à envoyer qui reflétera exactement les modifications faites par l'utilisateur.

Dans le cas de l'assignation d'une texture à un objet, il est nécessaire de tenir compte de la modification dans le calcul de la scène. En effet, lorsqu'une texture est liée à un objet, cet objet prend comme couleur la couleur moyenne de la texture. Cela implique que lors d'un changement de texture ou de l'assignation d'une texture à un objet qui n'en n'a pas, il est nécessaire de calculer les effets du changement de couleur dans la scène tout comme si l'utilisateur avait explicitement changé la couleur de l'objet.

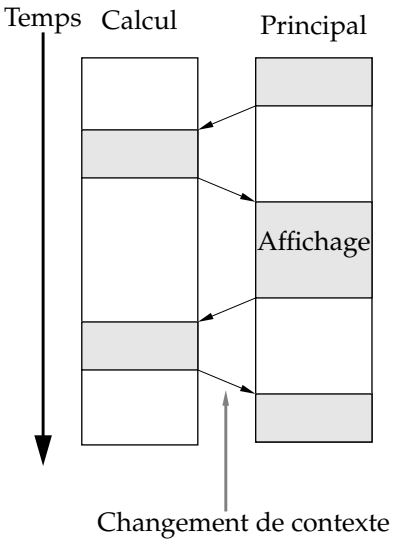
Un problème s'est rapidement posé à l'exécution lors de certains tests: une modification d'une unité ne correspond pas à deux modifications d'une demi-unité. Par exemple augmenter l'émissivité d'un objet de 1.0 ne donne pas les même résultats que de l'augmenter deux fois de suite de 0.5. Cela vient du fait que certaines optimisations sont faites dans le calcul de radiosité. En particulier, si une patch a une quantité d'énergie à émettre qui est inférieur à un certain seuil, elle n'est pas sélectionnée comme patch émettrice. De plus, si l'effet d'une patch sur une autre est inférieur à un seuil, elle n'est pas prise en compte dans le calcul, évitant ainsi de devoir lancer un rayon d'ombre qui coûterait cher pour une contribution pratiquement imperceptible. Ces deux optimisations font que certaines contributions qui sont prises en compte lors de modifications importantes ne le sont plus lors de modification de faible intensité, ce qui peut engendrer des erreurs visibles.

Il s'agit d'un problème délicat, puisqu'une augmentation de précision entraînerait un ralentissement du calcul, ainsi qu'un rallongement de celui-ci. La mise en oeuvre actuelle permet déjà un grand nombre de manipulations ne mettant pas en danger la cohérence de l'illumination de

la scène, aussi la recherche d'une solution à ce problème n'a pas été entreprise dans le cadre de ce projet.

Calcul en parallèle par tranche de temps

L'application devant être interactive, il est nécessaire de prévoir un moyen d'effectuer les calculs de manière transparente, en tâche de fond. On est donc dans une situation où un processus non prioritaire (le calcul) doit profiter de tous les temps morts laissé par un processus prioritaire (l'interface avec l'utilisateur).

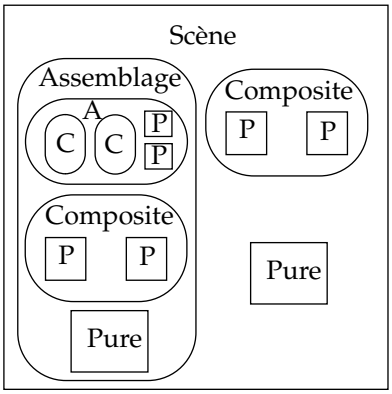


Une solution propre consisterait ici a créer un processus de calcul qui serait interrompu à intervalles de temps réguliers pour rendre la main à l'interface, et qui reprendrait son cours aussitôt après. Un problème apparait cependant rapidement: il n'est pas possible avec cette méthode d'assurer la cohérence des données à tout moment, ce qui signifie que l'utilisateur recevrait des données inexactes la majorité du temps. Une solution utilisant des coroutines est plus adaptée à ce problème. Le processus de calcul est codé sous la forme d'une coroutine, à laquelle un quota de temps est alloué, et qui rend la main au programme principal lorsque ce quota est épuisé. Puisque c'est la coroutine qui rend la main, elle le fait toujours à un moment où les données sont dans un état cohérent.

Le programme principal se présente alors sous la forme d'une boucle infinie dans laquelle on attend des événements. Une horloge, réglée adéquatement, génère les événements provoquant la mise à jour des données affichées. Les autres événements sont générés par l'utilisateur par l'intermédiaires des périphériques. Si aucun événement n'est généré, la coroutine de calcul est appelée avec un quota de temps (a moins que le calcul ne soit terminé, auquel cas, rien ne se passe).

Optimisations

Représentation conceptuelle des objets

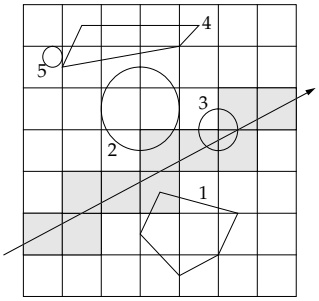


Les optimisations présentées ci-dessous utilisent des notions concernant la représentation des objets dans l'application.Elles sont présentées dans le paragraphe ci-dessous.

Les objets de la scène sont représentés par une structure hiérarchique, qui est la suivante (voir figure): au plus haut niveau, une scène est composée d'*Assemblages*, de *Composites* ou de *Pures*. Un Assemblage est constitué lui-même d'Assemblages, de Composites ou de Pures. Un Composite ne peut être composé que de Pures, et enfin, un Pure est composé exclusivement de *Polygones*. Cette hiérarchie est dictée par des besoins d'optimisations au niveau du lancer de rayon (voir plus bas). Un Assemblage correspond à un groupement d'objets relativement proches les uns des autres, alors qu'un Composite est un assemblage d'objets simples formant un objet *compact*. Compact signifie ici que la quantité d'espace vide entre les parties d'un objet composite, comprise dans le plus petit parallélipipède englobant l'objet, est relativement faible. Une chaise, par exemple, constituée de quatre pieds, un siège et un dossier, peut être définie

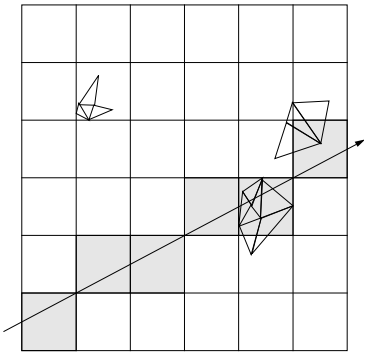
comme un objet compact. Une salle faite de quatre murs et ne contenant que quelques objets, par contre, ne devrait pas être déclarée compact.

Optimisation du lancer de rayon

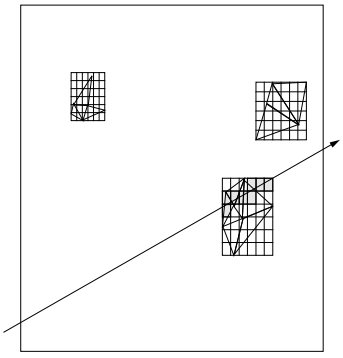


Equivalent 2D de la SUE

De nombreuses techniques d'accélération existent pour le lancer de rayon. La plus appropriée dans notre cas est la subdivision uniforme de l'espace [8]. Elle permet de déterminer l'objet le plus proche intersecté par un rayon en un temps virtuellement indépendant du nombre d'objets composant la scène. Il s'agit de diviser uniformément l'espace de la scène en *voxels* (*Vo*lume *El*ement c'est à dire ici parallélipipède rectangle), puis de déterminer pour chacun de ces voxels la liste des objets de la scène les intersectant. Ensuite, lorsqu'un rayon est envoyé dans la scène, il passe de voxel en voxel, et seuls les objets se trouvant dans le voxel courant sont considérés pour le calcul des intersections rayon-objet. Cela réduit considérablement le nombre de tests d'intersection par rapport à l'algorithme simpliste où tous les objets sont considérés pour chaque rayon. Par exemple, dans le cas de la figure de gauche, qui est l'équivalent en deux dimensions d'une configuration possible, seuls les objets 1, 2 et 3 seront considérés, dans cet ordre, pour les calculs d'intersection. Les cases grisées sont les seules qui puissent être visitées par le rayon, les autres n'étant pas traversées par lui, les objets qu'elles contiennent ou intersectent ne sont pas testés.


Grille de voxels pour la scène

Cet algorithme n'est malheureusement pas très efficace si les objets de la scène sont distribués non uniformément. En effet, plus le nombre de voxels vides que doit traverser un rayon est grand, plus le coût de la traversée de la grille de voxels devient important par rapport au temps gagné par ailleurs. C'est en partie pour palier à cet inconvénient qu'a été introduite la hiérarchie entre les objets décrite plus haut. En effet, tous les objets d'une scène étant constitués exclusivement de triangles, il est intéressant de ne définir des grilles de voxels que pour les objets compacts, et non pour la scène entière. Si cette dernière solution avait été choisie, l'algorithme aurait été totalement inefficace, car on aurait pu trouver de grandes quantités de polygones regroupés dans un même voxel, ce qui aurait ralenti le processus plus qu'autre chose. En définissant plusieurs grilles de voxels, une pour chaque objet *compact*, la taille des voxels est réduite à une échelle plus proche de celle des polygones, et le nombre maximum de triangles que l'on peux trouver dans un même voxel est grandement réduit. Ainsi seuls les objets Pures et les objets Composites bénéficient-ils d'une grille de voxels, les Assemblages eux n'étant pas considérés suffisamment *compacts* pour que l'utilisation d'une subdivision de l'espace pour ces objets soit rentable. Ces derniers sont néanmoins équipés comme les autres types d'objets d'un volume englobant (*bounding box*) qui permet de déterminer rapidement si un rayon a des chances de toucher les objets plus simples dont ils sont formés. L'utilisation d'objets Composites permet de rassembler un certains nombres d'objets Pures relativement simples, et de ne construire qu'une seule grille de voxels pour l'ensemble de ces objets, ce qui réduit la taille de la mémoire employée pour les grilles.


Grilles de voxels pour les objets

L'algorithme de lancer de rayon revient donc à la séquence suivante:

1. Parcourir la liste des objets constituant la scène ou l'objet courant.

2. Pour chaque objet, si le rayon touche le volume englobant alors:
   a) s'il s'agit d'un Assemblage, recommencer en 1) en prenant comme objet courant l'Assemblage en question.
   b) s'il s'agit d'un Composite ou d'un Pure, lancer le rayon dans la grille de voxels associée.
3. Si le rayon touche un polygone dans la grille, s'arrêter, le point est à l'ombre de la source de lumière.

Optimisation du calcul de radiosité

Une première optimisation est faite lors du calcul de l'effet d'une patch sur la scène. Pour effectuer ce calcul, il faut parcourir la totalité des objets de la scène et calculer l'effet de la patch émettrice sur chacun de ces objets. A chaque fois qu'un objet est considéré, il faut tester si cet objet se trouve devant la patch émettrice ou derrière. En effet, s'il se trouve derrière la patch, celle-ci n'aura aucun effet sur lui, et il donc inutile de considérer les objets et les polygones dont il est constitué pour ce calcul. Si par contre il se trouve devant, il convient de procéder au même test sur ses constituants.

Une autre optimisation efficace peut être faite lors du calcul de la radiosité pour une patch: avant l'envoi d'un rayon d'ombre, on vérifie que la contribution de la source à l'illumination de la patch est supérieure à une valeur minimale. Si elle ne l'est pas, il n'est pas nécessaire de lancer le rayon d'ombre, ce qui fait gagner beaucoup de temps car le lancer d'un rayon reste un calcul assez long. Cette technique permet d'accélérer de manière substantielle le calcul dans le cas d'une scène comportant des éclairages locaux, ou de petits objets qui ont très peu d'influence sur le reste de la scène. On peut également éviter le lancement de rayons inutiles en définissant quels objets projetteront des ombres, et lesquels n'en projetteront pas. De cette façon, de gros calculs peuvent être évités dans le cas des petits objets dont l'ombre est peu visible et dans le cas de sources de lumière puissantes qui n'ont à priori pas d'ombre portée propre discernable.

Interaction avec le calcul

Interface

Définition de l'interface

L'interface doit permettre deux choses à l'utilisateur: d'une part la visualisation de l'état du calcul pendant l'évaluation d'une solution, d'autre part la modification des caractéristiques des objets du modèle, toujours pendant le calcul.

Le premier des services de l'interface est la visualisation tri-dimensionnelle, avec la possibilité pour l'utilisateur de modifier les paramètres de visualisation interactivement.

Le second service consiste en un moyen d'édition interactif des caractéristiques des objets.

Visualisation tri-dimensionnelle en temps réel et navigation

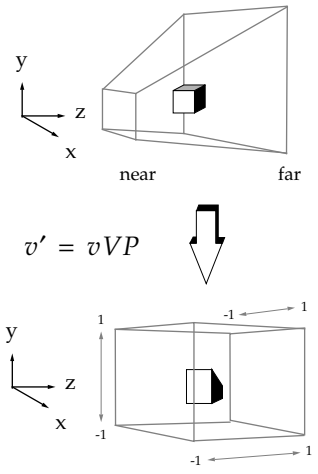
Naviguer dans une scène consiste à se déplacer à volonté à l'intérieur de la scène, en la visualisant sous différents angles. C'est l'utilisateur qui décide de la sa position et de son champ de vision. Pour cela il dispose en tout temps de la souris et, si possible, d'une spaceball. Dans le cas où cette dernière est disponible, elle sert uniquement au déplacement de l'observateur et à son orientation, alors que la souris a également d'autres fonctions décrites dans *Mode d'emploi page 25*. Les deux périphériques ne génèrent pas le même type d'informations, et doivent donc être traités séparément.

Afin de pouvoir expliquer la façon dont sont gérées les données reçues de la part de ces périphériques, il est d'abord nécessaire de présenter le fonctionnement de la librairie graphique de Silicon Graphics en ce qui concerne la gestion de la caméra (observateur). La *gl* (Graphics Library) utilise trois matrices pour la visualisation des scènes définies en trois dimensions: la matrice de perspective (*projection*), la matrice de visualisation (*viewing*) et la matrice de texture qui est décrite dans *Projection de textures page 17*. Tout point v donné à la *gl* est transformé d'abord dans l'espace de la caméra grâce à la matrice de visualisation, puis ses coordonnées sont normalisées en utilisant la matrice de perspective. On a donc:

$$v' = vVP \qquad \text{(Eq. 17)}$$

où:

V	= La matrice de visualisation
P	= La matrice de perspective
v	= Le point en coordonnées absolue dans le repère global
v'	= Le point en coordonnées normalisées



La matrice de perspective est calculée de telle manière que l'espace compris dans le volume de vision dans le repère local de la caméra soit transformé en un cube allant de -1 à 1 sur chaque axe du même repère (voir figure). Les objets de la scène sont ainsi modifiés de telle sorte que leur projection sur le plan d'équation z=-1 les fassent apparaître en perspective.

La matrice de visualisation, quant à elle, est égale à l'inverse de la matrice M permettant de passer du repère global au repère de la caméra. M est du type:

$$M = \begin{bmatrix} x_1 & x_2 & x_3 & 0 \\ y_1 & y_2 & y_3 & 0 \\ z_1 & z_2 & z_3 & 0 \\ t_x & t_y & t_z & 1 \end{bmatrix} \qquad \text{(Eq. 18)}$$

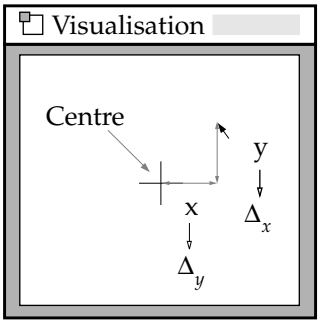
où les valeurs x_i (respectivement y_i et z_i) représentent le vecteur e_x (respectivement e_y et e_z) du repère de la caméra, et les valeurs t_i la position de la caméra.

Lorsqu'un ordre de mouvement est donné par l'utilisateur, c'est la matrice de visualisation qui doit être modifiée en conséquence.

La spaceball est le cas le plus simple à traiter car elle génère automatiquement une matrice quatre dimensions de delta-transformation correspondant à l'impulsion donnée par l'utilisateur sur la spaceball. C'est à dire qu'il suffit, pour tenir compte de l'ordre donné par l'utilisateur de prémultiplier la matrice de visualisation (*viewing*) courante par l'inverse de la matrice de delta-transformation:

$$V' = \Delta^{-1} V \qquad \text{(Eq. 19)}$$

La souris pose plus de problèmes, pour deux raisons. La première est que puisqu'il s'agit d'un périphérique à deux dimensions, elle génère des coordonnées en deux dimensions dans la fenêtre courante, et par conséquent ces coordonnées doivent être interprétées pour pouvoir calculer la nouvelle matrice de visualisation. La seconde raison est qu'il est nécessaire de prévoir un système permettant de fixer l'attitude de la caméra autour de l'axe des *Z*, étant donné que la souris ne permet pas de corriger cette attitude indépendamment de la position autour des deux autres axes (toujours parce qu'il s'agit d'un périphérique à deux dimensions).



Le premier problème est résolu en interprétant la position de la souris par rapport au centre de la fenêtre d'affichage comme deux rotations conjuguées, l'une autour de l'axe des *Y*, et l'autre autour de l'axe des *X*. Ainsi la distance horizontale entre le curseur et le centre de la fenêtre de visualisation donne l'angle de rotation à appliquer à la caméra autour de l'axe des *Y* (l'axe pointant vers le haut pour l'utilisateur), alors que la distance verticale entre le curseur et le centre de la fenêtre de visualisation donne l'angle de rotation à appliquer à la caméra autour de l'axe des *X* (l'axe pointant vers la droite pour l'utilisateur).

Le deuxième problème est résolu en changeant légèrement l'interprétation du déplacement horizontal de la souris. Ce déplacement correspond en fait à une rotation autour de la droite de même direction que l'axe *Y* du repère *global* et passant par la position de la caméra. On assure ainsi que la caméra ne tangue pas.

On a donc finalement:

$$\Re' = \Delta_t \Delta_x R \Delta_y T \qquad \text{(Eq. 20)}$$

$$V' = M \Re'^{-1} \qquad \text{(Eq. 21)}$$

avec:

$$\Re = TR \qquad \text{(Eq. 22)}$$

et:

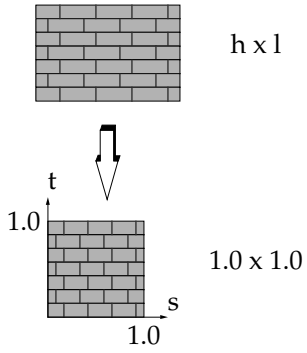
Δ_t = delta-translation

Δ_x = delta-rotation autour de l'axe des X
Δ_y = delta-rotation autour de l'axe des Y
M = matrice de modification

c'est à dire que $\Re$ représente la transformation, formée de la translation T et de la rotation R, du repère global dans le repère local de la caméra. La matrice de modification M est utilisée pour changer l'orientation du repère dans le cas où l'axe 'vers le haut' des données n'est pas l'axe des Y du repère global.

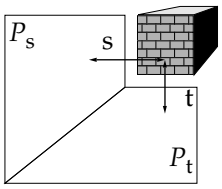
Dans le cas où l'utilisateur donne l'ordre d'avancer ou de reculer dans la scène, une translation selon l'axe des Z du repère de la caméra est conjuguée à la rotation autour de X et Y.

Projection de textures



L'affichage des textures se fait en utilisant une fonctionnalité de la *gl*: la projection d'images en deux dimensions sur un objet quelconque selon une direction quelconque. Cette direction de projection est définie en spécifiant deux plans selon lesquels l'image sera projetée sur les polygones affichés.

Considérons une image stockée en mémoire et définissant une texture. Cette image est d'abord projetée dans un repère où elle prend une taille de une unité par une unité (voir figure). On peut alors définir deux plans dans l'espace tels pour chaque point considéré pour l'affichage, la distance du point à ces deux plans est calculée. Ces distances représentent les coordonnées (s,t) dans l'espace de la texture du point dont il faut prendre la couleur. En fait, une matrice de changement d'échelle est d'abord appliquée aux coordonnées (s,t) afin de pouvoir garder l'échelle relative entre la hauteur et la largeur de l'image originale. Seule la partie décimale des coordonnées obtenues est ensuite utilisée afin d'effectuer une duplication à l'infini de l'image sur la surface affichée.



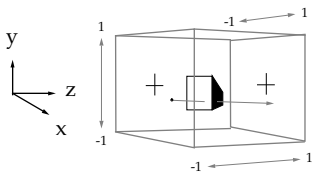
Dans notre cas, la direction de projection est déterminée par la direction de vision au moment où la texture est liée à l'objet. L'axe des X (respectivement Y) du repère de la caméra correspond à la normale au plan P_s (respectivement P_t). Les plans passent par le point inférieur gauche avant du volume englobant de l'objet texturé. De plus la matrice de changement d'échelle est multipliée par le facteur de taille spécifié par l'utilisateur, ce qui permet d'adapter la taille de la texture à celle de l'objet sur lequel elle est projetée.

Sélection des objets

La sélection d'un objet dans la scène se fait grâce à la souris. Il suffit de placer le pointeur sur l'objet désiré dans l'image tri-dimensionnelle courante et d'appuyer sur le bouton de la souris.

Pour déterminer l'objet sélectionné, on utilise les routines de lancer de rayon. En effet, il suffit de lancer un rayon dans la direction de vision déterminée par la souris et de prendre comme objet sélectionné l'objet le

plus proche rencontré dans cette direction. Déterminer le triangle correspondant dans l'objet est alors immédiat.



Pour calculer le point de départ du rayon et sa direction, on utilise le cube de vision en coordonnées uniformes. Le rayon a son origine dans le plan z=-1 et sa fin au même endroit dans le plan z=1 (voir figure). La position dans le carré de coordonnées uniformes dans chaque plan correspond à la position de la souris dans la fenêtre. On obtient donc deux points dans l'espace uniforme, qui transformés par la matrice de passage de l'espace uniforme à l'espace monde, permettent de trouver la direction et l'origine du rayon, ainsi que la longueur sur laquelle il nous intéresse (ce qui peut se trouver après la distance maximale ne doit pas être considéré pour le calcul d'intersection puisqu'il s'agirait d'objets placés hors du volume de vision, donc invisibles).

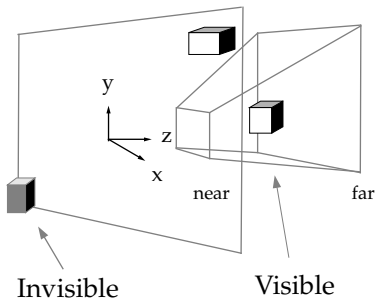
Si un objet est rencontré dans cette direction, le triangle touché devient le triangle sélectionné. Si ce triangle comporte des subdivisions, la subdivision sélectionnée est déterminée en fonction du point d'intersection du rayon et du triangle. L'objet contenant le triangle devient l'objet sélectionné.

Optimisations

Optimisation de l'affichage tri-dimensionnel

L'affichage des objets de la scène se fait en utilisant les primitives graphiques de la *gl*. Une fois que les paramètres de vision, c'est à dire les matrices de visualisation et de perspective, ont été définis, il suffit pour afficher un polygone de donner à la *gl* les trois sommets du triangle ainsi que leur couleur, en demandant une interpolation de ces couleurs sur la surface du triangle (*shading* de Gouraud). Cette procédure est suivie pour chaque triangle de chaque subdivision de chaque objet de la scène, et ceci chaque fois que la vue est mise à jour.

La *gl* s'occupe de n'afficher que les polygones visibles à l'écran, et de découper ceux qui ne le sont qu'à moitié. Néanmoins, donner la description complète d'un objet, polygone par polygone, coûte en temps de traitement, ce qui ralentit l'affichage. Par conséquent, afin d'obtenir de meilleures performances dans la cadence de mise à jour, il est préférable de ne donner à la *gl* que les polygones visibles ou partiellement visibles.



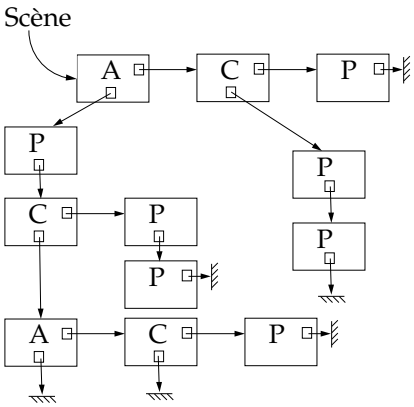
Cette optimisation peut être réalisée à différents niveaux. La version la plus simple consiste à ne considérer pour l'affichage que les objets se trouvant dans le demi-espace de vision, c'est à dire devant le plan ayant pour normale la direction de vision et passant par la position de la caméra. Il s'agit d'un test simple et peu coûteux qui permet d'éliminer en moyenne un peu moins de la moitié des objets de la scène. Une optimisation plus efficace consiste à éliminer les objets qui ne se trouvent pas dans le volume de vision de la caméra. Ce test est plus complexe à mettre en oeuvre puisqu'il s'agit de tester l'intersection entre le volume de vision (un cube, en coordonnées uniformes) et un tronçon de pyramide (le volume englobant de l'objet, en coordonnées uniforme). Par contre il permet d'éliminer en moyenne plus que la moitié des objets de la scène.

Dans les deux cas, le test est fait de manière récursive dans la hiérarchie de la scène: les constituants d'un assemblage ou d'un composite ne sont testés que si leur père à été déclaré visible par le test.

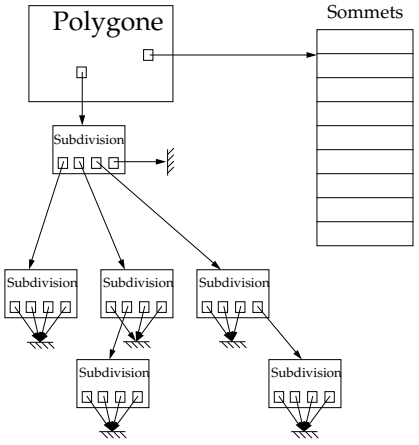
Dans notre cas, c'est la première solution qui a été mise en oeuvre.

Réalisation

Structure des données

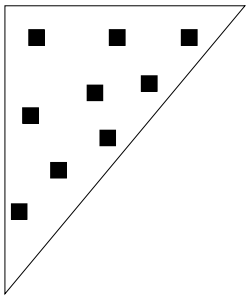


La structure interne des données est similaire à la représentation externe de la scène. Un objet est une structure à champs variables comprenant les informations communes à chaque objet, indépendamment de son type, ainsi qu'un champ formé d'une structure spécifique au type de l'objet (Assemblage, Composite ou Pure). Chaque objet peut avoir un frère qui est défini par un pointeur sur sa structure. Les fils d'un Assemblage ou d'un Composite sont liés de cette manière en une liste chaînée dont l'adresse de la tête est maintenue dans la structure de l'objet père (voir figure). La partie spécifique de l'Assemblage ne comporte que le pointeur sur ses fils, alors que celle du Composite comporte ce même pointeur ainsi qu'un pointeur sur une structure de grille de voxels. La partie spécifique du Pure comporte quant à elle un pointeur sur une structure de grille, ainsi que les informations concernant la surface de l'objet (couleur, réflectivité, émission et surface totale de l'objet) ainsi que le nombre de polygones dont il est formé et un tableau (dynamique) de ces polygones.



Un triangle peut être de deux types, un polygone ou une subdivision. Il est représenté par une structure à champs variables comprenant les informations indépendantes du type (polygone ou subdivision), c'est à dire son type et un pointeur sur l'objet le contenant, ainsi qu'un champ formé d'un structure spécifique au type. Un polygone est le type de triangle constituant les objets Pures. Sa structure contient les informations nécessaires au calcul d'intersection rayon-triangle, un tableau dynamique de sommets, contenant les trois sommets originaux ainsi que les éventuels sommets supplémentaires créés lorsque le triangle est subdivisé, un tableau d'échantillons utilisés pour l'échantillonnage du triangle lorsque celui-ci est la patch source ainsi qu'un pointeur sur une subdivision éventuelle. Une subdivision quant à elle, comprend un pointeur sur le triangle dont elle est descendante, et quatre éléments représentant les quatre triangles formant le triangle père, comprenant chacun les indices, dans la table du polygone, des trois sommets les définissant, la surface de l'élément ainsi qu'un pointeur sur une éventuelle subdivision de l'élément.

Un sommet, de son côté, comprend des informations telles que son identificateur, sa position dans l'espace, la radiosité totale qu'il émet, la radiosité qu'il a accumulé depuis le début du calcul ou depuis que la patch dont il fait partie a été la patch source, ainsi qu'un indice de tableau identifiant la valeur de radiosité en ce point due à la patch source courante.



Un échantillon est une structure représentant un point d'échantillonnage sur la surface d'un polygone. Elle contient la position du point, un pointeur sur un élément de subdivision, et les coordonnées barycentriques du point dans l'élément. Le pointeur sur l'élément peut être nul si le polygone dans lequel l'échantillon a été pris n'a pas été subdivisé. Dans ce cas, les coordonnées barycentriques du point sont définies par rapport au polygone en question. Ces coordonnées barycentriques sont utilisées pour estimer la valeur de la radiosité au point d'échantillonnage en fonction de la radiosité au sommets du triangle. Les échantillons sont choisis de manière régulière et uniforme sur la surface du polygone (voir figure).

Une grille de voxels est une structure contenant toutes les informations nécessaires à la définition et au stockage d'un grille de subdivision uniforme de l'espace, c'est à dire le nombre de voxels suivant les trois axes, la taille d'un voxel suivant les mêmes trois axes, l'origine de la grille ainsi qu'un tableau dynamique à trois dimensions contenant les listes de pointeurs sur les triangles se trouvant dans les voxels associés.

Implémentation de la coroutine de calcul

Le langage C ne permettant pas de définir explicitement des coroutines, la coroutine de calcul a été implémentée comme une procédure qui stocke son état interne dans la structure de description de la scène qu'elle doit calculer. Ceci permet de dissocier complètement le processus de calcul du processus d'affichage.

La procédure de calcul est composée de plusieurs boucles imbriquées qui ont comme condition d'arrêt la condition requise pour le calcul, conjuguée avec la condition d'arrêt sur le quota de temps. A l'entrée de chaque boucle, l'état de la variable de boucle correspondante est testée et initialisée si nécessaire. A la sortie de chaque boucle les variables sont mises à jour uniquement si la condition de sortie de la boucle interne indique qu'il n'y a pas eu épuisement du quota de temps. Dans le cas contraire la condition d'arrêt se propage vers le haut dans les boucles imbriquées et provoque le retour à la procédure appelante. La condition d'arrêt sur le quota de temps est évaluée dans la boucle la plus interne, après l'exécution de l'opération de calcul de base. Celle-ci est considérée comme atomique, c'est à dire qu'il s'agit de la plus petite opération de calcul laissant les données dans un état cohérent. Dans notre cas, cette opération correspond au calcul de l'effet d'une patch sur une autre.

Les tranches de temps allouées au processus de calcul doivent lui permettre d'effectuer un nombre raisonnable d'opérations de calcul de base avant l'épuisement du quota afin que le temps perdu dans le passage du programme principal à la routine de calcul et l'entrée dans la routine ne soit pas trop grand par rapport au temps passé à calculer. Des tranches de temps d'environ 100ms donnent de bon résultats: en moyenne 3 à 5 calculs par tranche. Il est préférable de ne pas diminuer la taille de la tranche jusqu'au point où seulement un calcul de base peut être effectué pendant le quota de temps imparti, car cela signifie que ce quota peut être largement dépassé et par conséquent que la cadence de mise à jour peut ne pas être respectée. Un nombre de 4 ou 5 calculs par tranche assure que le

dépassement de quota reste raisonnable (au maximum un quart ou un cinquième de tranche). Il intéressant de noter que le nombre de calculs par tranche peux varier énormément, suivant que les patchs réceptrices doivent être subdivisées ou non. On arrive ainsi à 50 ou 60 calculs par tranche pour des tranches de 100ms.

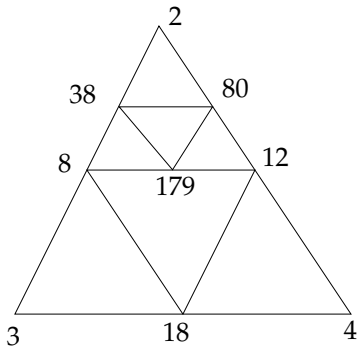
Problèmes de réalisation

Création et gestion des subdivisions et des sommets associés

Lors de la création d'une subdivision, trois nouveaux points sont théoriquement créés pour la définition des quatre éléments de la subdivision: les points milieux des trois côtés du triangle à subdiviser. Pratiquement cependant, il n'est pas avantageux de réellement créer chacun de ces points. En effet, comme il a été montré plus haut, deux subdivisions peuvent utiliser le même point, et par conséquent, créer deux occurrences de ce point serait une perte de place mémoire ainsi que de temps car les radiosités pour ce point devraient être calculées deux fois. Par conséquent, il est avantageux de stocker les sommets créés pour les subdivisions d'un polygone avec les sommets originaux de ce polygone dans un tableau dynamique. Par la suite, lorsqu'une subdivision est créée, chaque sommet dont a besoin la subdivision est recherché dans la table grâce à son identificateur, et s'il n'est pas trouvé il est créé et ajouté à la fin de cette table. La recherche du sommet dans la table est séquentielle car le tri du tableau des sommets en fonction de leur identificateur impliquerait des déplacement de sommets dans le tableau. Or il serait peu rentable de parcourir la hiérarchie des subdivisions pour mettre à jour les indices des sommets utilisés à chaque insertion d'un nouveau sommet dans la table. L'utilisation d'une table d'indiçage pour palier à ce problème serait prohibitive en emplacement mémoire et de plus, étant donné le faible nombre moyen de sommets stockés pour un même polygone, le gain de temps apporté par une recherche dichotomique serait vraisemblablement perdu dans le temps utilisé pour l'insertion des éléments dans la table.

:							
6	16						
5	11	17					
4	7	12	18				
3	4	8	13	19			
2	2	5	9	14	20		
1	1	3	6	10	15	21	
	1	2	3	4	5	6	..

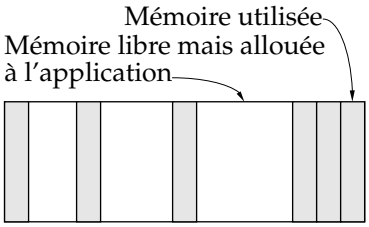
Table 1



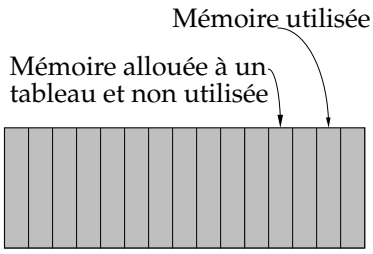
Numérotation des sommets

Il existe des restrictions quant à la création des identificateurs des points. L'identificateur doit être unique, et il doit être calculable en fonction des identificateurs de ses parents. Une solution à ce problème est d'utiliser une table qui a la forme de la table 1. Le nombre indexé par les identificateurs des deux parents est l'identificateur du fils, qui est nécessairement unique (voir figure).

Lors de la création d'un nouveau sommet, il est nécessaire d'agrandir la table de sommets afin de pouvoir l'y placer. Ceci peut provoquer des problèmes au niveau de la gestion de la mémoire, puisqu'il est nécessaire de réallouer de l'espace mémoire pour un tableau plus grand. Si l'espace déjà existant n'est pas suffisant, le gestionnaire de mémoire se chargera de demander de la mémoire au système et de recopier le tableau dans un bloc de mémoire fraîchement alloué. La mémoire qui vient d'être libérée peut être utilisée pour allouer d'autres petits blocs de mémoire, cependant si cela se répète trop souvent, la mémoire se fragmentera petit à petit, et l'espace mémoire demandé au système par l'application augmentera dans des proportions démesurées. Afin d'éviter une telle situation, la mémoire pour les tableaux évoluant dynamiquement est allouée par



Fragmentation de la mémoire



Allocation par blocs

blocs de plusieurs éléments. Lorsqu'un tableau doit s'étendre au-delà de ses capacités, un nouveau bloc lui est attribué. Lorsqu'un élément doit y être ajouté et qu'il reste de la place, il est simplement recopié dans le premier espace libre du tableau et le nombre de places libres pour ce tableau est décrementé. De cette manière, le nombre de réallocation provoquant une demande d'espace mémoire supplémentaire au système est grandement réduit. L'espace mémoire inutilisé est ainsi maintenu dans des limites raisonnables.

Calcul des intersections rayon-triangle et initialisation des grilles de voxels

Deux problèmes cruciaux sont le calcul du point d'intersection entre un rayon et un triangle et le calcul de l'intersection entre un triangle et un voxel. Le premier survient dans le lancer de rayon, le second dans l'initialisation des grilles de voxels, lorsqu'il faut déterminer quels voxels un triangle particulier intersecte. Les deux problèmes sont très proches l'un de l'autre car un voxel peut être considéré comme un point dont les coordonnées sont des intervalles. Il suffit donc de résoudre le premier problème pour que le deuxième le soit également grâce à l'analyse d'intervalle [5].

Le calcul du point d'intersection entre un rayon et un triangle revient à projeter l'origine du rayon sur le plan du triangle selon la direction du rayon, puis de déterminer si le point projeté se trouve bien à l'intérieur du triangle. Pour faire cela, il faut utiliser les coordonnées barycentriques du point dans le triangle [5].

Considérons le triangle T de sommets $\underline{a}$, $\underline{b}$et $\underline{c}$. Définissons les vecteurs $\underline{e}_a = \underline{a} - \underline{c}$ et $\underline{e}_b = \underline{b} - \underline{c}$. Le triangle est ainsi dans le plan de normale $\underline{n} = \underline{e}_a \times \underline{e}_b$ contenant le point $\underline{c}$. Posons:

$$\underline{u} = \frac{1}{|\underline{n}|}\underline{n} \qquad \text{(Eq. 23)}$$

et considérons un point $\underline{p}$ dont la position relative par rapport au point $\underline{c}$ est $\tilde{\underline{p}} = \underline{p} - \underline{c}$. Alors [5]:

- $\underline{p}$ est dans le plan du triangle si et seulement si $\tilde{\underline{p}} \cdot \underline{u} = 0$
- En admettant que la condition précédente soit vraie, $\underline{p}$ est dans le triangle si et seulement si $\tilde{\underline{p}} = \alpha\underline{e}_a + \beta\underline{e}_b$ avec α, β et $\alpha + \beta$ appartenant à [0,1]. Dans ce cas, on a $\underline{p} = \alpha\underline{a} + \beta\underline{b} + \gamma\underline{c}$ avec α, β et $\gamma = \alpha + \beta$ appartenant à [0,1].

α, β et γ sont les coordonnées barycentriques de $\underline{p}$ dans le triangle T. D'après [5], on a:

- $\alpha = \tilde{\underline{p}} \cdot \underline{s}_a$ avec $\underline{s}_a = \dfrac{\underline{e}_b \times (\underline{e}_a \times \underline{e}_b)}{|\underline{e}_a \times \underline{e}_b|}$
- $\beta = \tilde{\underline{p}} \cdot \underline{s}_b$ avec $\underline{s}_b = \dfrac{\underline{e}_a \times (\underline{e}_b \times \underline{e}_a)}{|\underline{e}_a \times \underline{e}_b|}$
- $\gamma = \tilde{\underline{p}} \cdot \underline{s}_c$ avec $\underline{s}_c = \underline{s}_a + \underline{s}_b$

Afin d'accélérer le calcul du test d'appartenance, on garde avec le triangle les valeurs de $\underline{s}_a$ et $\underline{s}_b$.

Il suffit alors d'appliquer ces mêmes calculs à un voxel en utilisant l'analyse d'intervalles afin de déterminer si celui-ci intersecte le triangle. On a pour cela les formules suivantes [5]:

$$\overline{\underline{V}} - \underline{c} = (\overline{V}^x - c_x, \overline{V}^y - c_y, \overline{V}^z - c_z)$$

où $\overline{V}^x$, $\overline{V}^y$ et $\overline{V}^z$ sont des intervalles, $\underline{c}$ est un vecteur et $\overline{\underline{V}}$ est un voxel. De plus [5]:

$$\overline{\underline{V}} \cdot \underline{u} = c_x \overline{V}^x + c_y \overline{V}^y + c_z \overline{V}^z$$

On test alors d'abord si le voxel intersecte le plan du triangle. On ramène le voxel dans le repère local du triangle:

$$\tilde{\overline{\underline{V}}} = \overline{\underline{V}} - \underline{c}$$

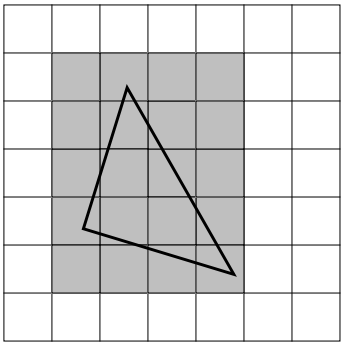
puis, si

$$0 \in \tilde{\overline{\underline{V}}} \cdot \underline{u}$$

alors le voxel intersecte effectivement le plan du triangle. En admettant que cette condition soit vraie, pour chaque $\underline{s} \in \{\underline{s}_a, \underline{s}_b, \underline{s}_c\}$ on test si:

$$\overline{\underline{V}} \cdot \underline{s} \cap [0, 1] \neq \emptyset$$

Si c'est le cas pour chacun d'eux, le voxel intersecte le triangle. Il suffit donc pour déterminer quels triangles intersectent quels voxels dans la grille, de procéder de la manière suivante pour chaque triangle: déterminer les voxels candidats à l'aide du volume englobant du triangle (voir figure), puis pour chacun de ces voxels tester s'il intersecte le triangle et si c'est le cas, ajouter le triangle à la liste des triangles de ce voxel.



Voxels à tester

Voxels contenant le triangle

Conclusion

Les buts définis pour ce projet ont été atteints, ce qui prouve la faisabilité d'une application utilisable en dehors d'un contexte expérimental, dans le domaine de l'architecture. Une application a été développée, elle permet de calculer l'illumination d'une scène en tenant compte des interréflexions diffuses. De plus elle permet de visualiser et de modifier cette scène de manière interactive. Le calcul est relativement rapide pour ce type d'application et les résultats sont de bonne qualité.

De multiples améliorations pourraient cependant être apportées.

Le plus gros travail devrait être fait sur l'algorithme de subdivision automatique comme décrit dans *Subdivision adaptative page 6*.

Des algorithmes pourraient être développés pour traiter les données reçues en entrée: il s'agirait par exemple d'éliminer les cas d'intersection de polygones décrit dans *Subdivision adaptative page 6* en découpant effectivement l'un des polygones, voir les deux, afin que les limites des polygones se juxtaposent exactement.

D'un autre côté, il serait possible d'analyser la géométrie de la scène afin de construire automatiquement une hiérarchie optimale pour le lancer de rayon.

Une optimisation supplémentaire pourrait être employée principalement pour les modèles de bâtiments comportant plusieurs pièces: il s'agirait de définir pour chaque objet la liste des objets sur lesquels il peut avoir une influence. Ainsi, les objets d'une pièce n'auront d'effet que sur les autres objets de cette même pièce et non sur les objets de la pièce à côté. Cela peut poser des problèmes lorsqu'une porte ouverte sépare les deux pièces car à ce moment elles interagissent par l'intermédiaire de cette ouverture, cependant c'est un problème contournable à l'aide d'approximations. Le bénéfice d'une telle méthode devrait être particulièrement efficace dans un bâtiment comportant un grand nombre de pièces.

L'entrée des données devrait être modifiée pour accepter directement les formats standard du type AutoCAD ou celui d'un autre modeleur couramment utilisé. Un architecte pourrait alors l'utiliser directement pour visualiser de manière plus réaliste le bâtiment qu'il modélise.

Mode d'emploi

Lancement du programme

Le lancement du programme se fait grâce à l'ordre suivant:

```
> rad
```

Deux fenêtres apparaissent alors à l'écran. Celle de gauche est la fenêtre de visualisation tri-dimensionnelle, celle de droite est la fenêtre de contrôle de l'application.

Navigation dans une scène

Il existe deux moyens pour naviguer dans une scène: soit utiliser la spaceball, soit utiliser la souris. Dans les deux cas, c'est la caméra qui se déplace dans la scène en fonction des ordres donnés, et non pas la scène qui est déplacée autour d'un observateur fixe. La différence est purement nominale dans le cas de la souris, mais ne l'est pas dans celui de la spaceball.

Utilisation de la souris

L'utilisation de la souris est relativement simple: une rotation conjuguée autour de l'axe X et de l'axe Y de la caméra est effectuée en plaçant le pointeur de la souris dans la fenêtre de visualisation, puis en appuyant sur le bouton de gauche de la souris. La position du pointeur par rapport au centre de la fenêtre détermine les angles de rotation à appliquer à la caméra: plus le pointeur est éloigné du centre, plus la rotation est rapide.

Pour avancer dans la scène, il suffit d'appuyer sur le bouton du milieu de la souris. Le placement de la souris a le même effet que dans le cas précédent, mais une translation selon l'axe des Z de la caméra est ajoutée.

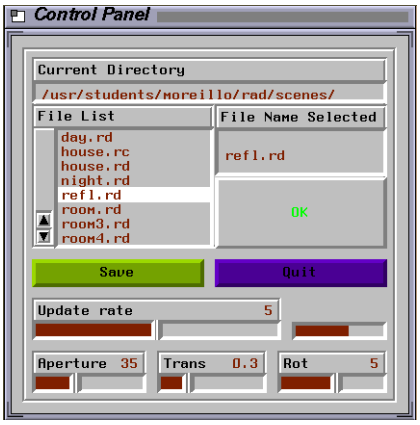
Reculer dans la scène se fait de la même manière qu'avancer, mais en enfonçant le bouton de gauche en même temps que le bouton du milieu.

Utilisation de la spaceball

L'utilisation de la spaceball est intuitive, mais il faut se rappeler que c'est la caméra qui est déplacée, non pas la scène.

Manipulation des scènes

Les fichiers de stockage d'une scène peuvent être de deux types. Le premier, le fichier initial, est la description initiale de la scène. Il doit être créé par l'utilisateur selon la grammaire définie dans *Format du fichier d'entrée page 29*. Son nom doit se terminer par l'extension ".rd". Le second, le fichier intermédiaire, est un fichier binaire contenant la description de l'état courant du calcul pour la scène sauvegardée. Son nom doit se terminer par l'extension ".rc".



Dans la partie supérieure de la fenêtre de contrôle se trouve la liste de fichiers disponibles dans le répertoire courant des scènes.

Charger une scène

Pour charger une scène, il suffit de cliquer sur le nom de la scène dans la liste puis de cliquer avec la souris sur le bouton ***OK*** se trouvant juste à coté. Le fichier de description de la scène est alors lu, et sitôt la lecture terminée, la vue tri-dimensionnelle de la scène est affichée. De plus, si le calcul n'a pas encore commencé, ou n'est pas encore terminé (lorsqu'une scène est rechargée après avoir été sauvegardée par exemple), il reprend là où il avait été interrompu.

Si la scène courante n'a pas été sauvegardée au moment où le bouton ***OK*** est enfoncé, une fenêtre d'avertissement apparait au centre de l'écran. Trois boutons permettent de sauvegarder l'état de la scène courante et continuer l'opération (bouton ***Save***), ne pas sauvegarder l'état de la scène et continuer l'opération (bouton ***Don't Save***) ou annuler l'opération (bouton ***Cancel***). Il suffit de cliquer sur le bouton désiré.

Sauvegarder une scène

Une scène peut être sauvegardée à tout moment. Pour cela, cliquer sur le bouton marqué ***Save*** en dessous de la liste des scènes. C'est l'état du calcul au complet qui est écrit sur fichier, pour pouvoir être repris plus tard. **Le nom du fichier de sauvegarde est toujours le nom du fichier duquel la scène a été lue, éventuellement modifié pour avoir une extension ".rc".** Cela signifie que si la scène avait été chargée depuis un fichier intermédiaire, ce fichier sera écrasé par le nouveau.

Contrôle de l'application

Quitter l'application

Pour quitter l'application, cliquer sur le bouton marqué ***Quit*** à droite du bouton ***Save***. Si l'état courant de la scène n'a pas été sauvegardé au moment où le bouton ***Quit*** est enfoncé, la même fenêtre d'avertissement que ci-dessus apparait.

Certains paramètres de l'application peuvent être modifiés interactivement.

Cadence de mise à jour

Le premier est la cadence de mise à jour de la vue tri-dimensionnelle. Elle peut être fixée grâce à l'ascenseur horizontal ***Update Rate*** se trouvant sous le bouton ***Save***. Il permet de faire varier la cadence de 0 à 10 images par secondes. La cadence effective n'est cependant pas nécessairement celle demandée, elle dépend d'abords de la rapidité de la station.

Angle d'ouverture

Le second est l'angle d'ouverture de la caméra. Il peut varier entre 5° et 90° et il est fixé par l'ascenseur ***Aperture*** en dessous de l'ascenseur

Update Rate. Il permet d'avoir une vue plus ou moins restreinte autour de la caméra.

Vitesse de translation

Le troisième est la distance parcourue à chaque pas lors de la navigation à la souris. Il varie entre 0.1 et 10 unités et est fixé par l'ascenseur ***Trans***.

Vitesse de rotation

Le dernier est l'angle maximum de rotation par pas lors de l'utilisation de la souris pour la navigation. Il varie entre 1° et 20° et est fixé par l'ascenseur ***Rot***.

Changement de mode (Navigation/Sélection)

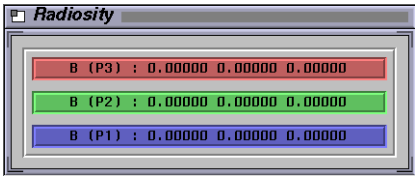
Un menu apparait dans la fenêtre de visualisation lorsque la souris est dans cette fenêtre et que le bouton de droite de la souris est enfoncé. Il permet de quitter l'application ou de choisir parmi deux modes: Navigation ou Sélection. Le mode de navigation est le mode par défaut, il permet de se déplacer avec la souris dans la scène. Le mode de sélection change l'interprétation des événements venant de la souris: elle permet alors de sélectionner un objet dans la scène, mais plus de se déplacer en l'utilisant.

La spaceball n'est pas influencée par le mode de fonctionnement, elle permet toujours de naviguer.

Modification de la scène

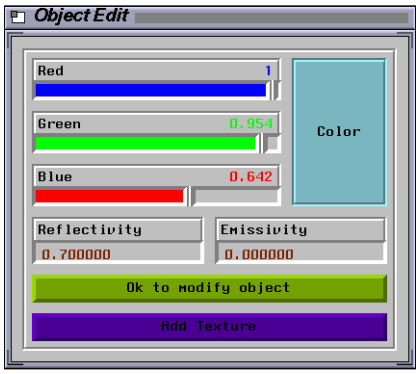
Sélection d'un objet

En mode sélection, la souris permet de sélectionner un objet dans la scène. Pour cela, il suffit de placer le pointeur de la souris sur l'objet à sélectionner dans l'image de la scène et d'enfoncer le bouton gauche de la souris. L'objet devient alors transparent, et le polygone sélectionné dans l'objet est affiché en fil de fer avec ses subdivisions éventuelle. De plus le triangle sélectionné (triangle original ou subdivision) est affiché en un dégradé de couleur avec un sommet rouge, un vert et le troisième bleu. Ces couleurs correspondent à celles des radiosités affichées dans la fenêtre des radiosités (voir ci-dessous).



Lorsqu'un objet est sélectionné, deux fenêtres sont affichées en dessous de la fenêtre de contrôle: la fenêtre des radiosités affichant les valeurs des radiosités au trois sommets du triangle sélectionné, et la fenêtre de modification de l'objet. Les radiosités sont affichées pour les trois canaux rouge, vert et bleu, à ne pas confondre avec le code des couleurs utilisé pour établir la correspondance entre les sommets et les valeurs. La radiosité affichée sur un fond rouge par exemple est celle du point rouge affiché dans la fenêtre de visualisation. Cette radiosité comporte trois composantes: la composante rouge (premier nombre), la verte (deuxième nombre) et la bleue (dernier nombre).

La nature de la deuxième fenêtre affichée dépend de l'objet sélectionné. Si l'objet n'a pas de texture, c'est la fenêtre d'édition standard qui est affi-



chée, sinon, c'est la fenêtre d'édition des textures qui est affichée à la place.

Modification d'un objet sans texture

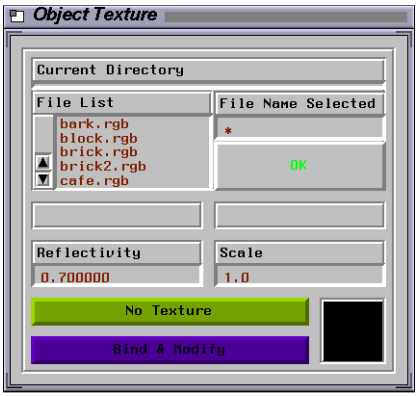
La fenêtre de modification d'un objet sans texture permet de modifier sa couleur, sa réflectivité et son émission.

La couleur est définie par les trois ascenseurs horizontaux ***Red***, ***Green*** et ***Blue*** qui permettent d'ajuster les composantes de rouge, vert et bleu de la couleur.

La réflectivité et l'émission sont définies par les valeurs placées dans les champs d'entrée de texte ***Reflectivity*** et ***Emissivity***. Pour changer une valeur, cliquer avec la souris dans le champ correspondant, taper une nouvelle valeur, et **taper Return**.

Une fois les paramètres fixés aux valeurs désirées, cliquer sur le bouton ***Ok to modify object***. Les modifications seront effectuée dans la scène dès que l'état du calcul le permettra.

Pour appliquer une texture à un objet, cliquer sur le bouton ***Add Texture***, la fenêtre des textures apparaîtra.



Modification d'un objet texturé

Lorsqu'un objet texturé est sélectionné, il est affiché sans texture en plus d'être transparent.

La fenêtre des textures permet de choisir une texture parmi une liste, de définir son facteur de taille, de l'appliquer à l'objet et de changer la réflectivité de la surface.

Pour choisir une texture, cliquer sur son nom dans la liste des textures, puis cliquer sur le bouton ***OK*** placé à côté. La texture est alors chargée en mémoire et affichée (après réduction) dans le petit carré placé en bas à droite de la fenêtre.

La modification du facteur de taille se fait en écrivant sa taille dans le champ ***Scale***. Un facteur de 1.0 signifie qu'un texture carrée se projettera sur un carré d'une unité de côté sur l'objet texturé. Ne pas oublier de taper Return à la fin.

La modification de la réflectivité est faite de la même manière que précédemment.

Une fois toute les modifications effectuées, cliquer sur le bouton ***Bind & Modify***. Cela aura pour effet d'effectivement modifier l'objet. La texture choisie sera projetée sur l'objet suivant la direction de vision courante.

Pour ôter toute texture à un objet, cliquer sur le bouton ***No Texture***, la fenêtre de modification standard apparaîtra.

Conseil d'utilisation

Voici quelques conseils d'utilisations pour obtenir des performances optimales de la part de l'application.

Construire des modèles simples. Plus le modèle est complexe, plus le temps de calcul sera grand et plus la cadence d'affichage sera faible. Eviter les polygones inutiles et les détails superflus. Eviter les polygones de très grandes tailles, ils risquent de ne pas être subdivisés adéquatement.

Ne pas sous-estimer les sources de lumière. Au début du calcul les sources de lumières semblent toujours trop faibles, parce que l'éclairage est principalement direct. Plus tard dans le calcul cependant, les interréflexions augmentent beaucoup les illuminations (surtout dans les espaces entièrement fermés).

Utiliser un minimum de textures. Les textures ralentissent l'affichage, surtout si elles sont projetées sur de grandes surfaces.

Faire des modifications réalistes. L'algorithme est conçu pour simuler des situations réelles, aussi les modifications doivent-elles être aussi réalistes que possible (rendre une chaise lumineuse n'est pas une opération très courante dans le monde réel).

Format du fichier d'entrée

Le fichier d'entrée est un fichier texte répondant à la grammaire suivante:

Scène:==	Chaîne **UpAxis** Axe Composition
Axis:==	**x_axis** |
	y_axis |
	z_axis
Composition:==	Complexe |
	Composition Complexe
Complexe:==	**Object** Chaîne Assemblage **|**
	Object Chaîne Composite |
	Simple
Assemblage:==	**Assembly** {Composition}
Composite:==	**Composite** {Objets}
Objets:==	Simple |
	Objets Simple
Simple:==	**Object** Chaîne Pure
Pure:==	**Pure** {Caractéristique Géométrie}
Caractéristique:==	Base |
	Texture Base
Base:==	**Reflectivity** Réel **Color** Vecteur **Emission** Réel **Shadows** Etat
Texture:==	**Texture File** NomFichier **Direction** Vecteur **Scale** Réel
Etat:==	On |

	Off
Géométrie:==	Fichier |
	Données
Fichier:==	**File** NomFichier
Données:==	**Data** {TabPolys}
TabPolys:==	Poly |
	TabPolys Poly
Poly:==	Vecteur Vecteur Vecteur
Vecteur:==	**<** Réel Réel Réel **>**
Chaîne:==	**"**Chaîne_de_caractères **"**
NomFichier:==	Chaîne_de_caractères. Chaîne_de_caractères
Cacahuète:==	Ecorce Graine Graine

De plus, tous les caractères d'une ligne suivant un point-virgule (';') sont considérés comme faisant partie d'un commentaire et sont ignorés. Pour des exemples, voir les fichiers de description de scène donnés en annexe.

Bibliographie

[1] Wallace, Elmquist, Haines, “A Ray Tracing Algorithm for Progressive Radiosity”, Computer Graphics, Vol. 23, Num. 3, Juillet 1989

[2] Cohen, Chen, Wallace, Greenberg, “A Progressive Refinement Approach to Fast Radiosity Image Generation”, Computer Graphics, Vol. 22, Num. 4, Août 1988

[3] Cohen, Greenberg, Immel, Brock, “An Efficient Radiosity Approach for Realistic Image Synthesis”, IEEE CG&A, Mars 1986

[4] Cohen, Greenberg, “The Hemi-Cube, A Radiosity Solution for Complex Environments”, Siggraph, Vol. 19, Num. 3, Juillet 1985

[5] Spackman, “SMART Theory, Octtree Decomposition”, U. of Edinburgh, Mai 1991

[6] Snyder, Barr, “Ray Tracing Complex Models Containing Surface Tessellations”, ACM, 1987

[7] Chen, “Incremental Radiosity: An Extension of Progressive Radiosity to an Interactive Image Synthesis System”, Computer Graphics, Vol. 24, Num. 4, Août 1990

[8] Glassner, “Space Subdivision for Fast Ray Tracing”, IEEE CG&A, Octobre 1984

[9] Tanaka, Iwata, “ARTS: Accelerated Ray-Tracing System”, IEEE CG&A, Avril 1986

[10] Buckalew, Fussel, “Illumination Networks: Fast Realistic Rendering with General Reflectance Functions”, Computer Graphics, Vol. 23, Num. 3, Juillet 1989

[11] Baum, Rushmeier, Winget, “Improving Radiosity Solutions Through the Use of Analytically Determined Form-Factors”, Computer Graphics, Vol. 23, Num. 3, Juillet 1989

[12] Puech, Sillion, Vedel, “Improving Interaction with Radiosity-based Lighting Simulation Programs”, ACM, 1990

[13] Airey, Rohlf, Brooks, “Towards Image Realism with Interactive Update Rates in Complex Virtual Building Environments”, ACM, 1990

[14] Goral, Torrance, Greenberg, Battaile, “Modeling the Interaction of Light Between Diffuse Surfaces”, Computer Graphics, Vol. 18, Num. 3, Juillet 1984

[15] Recker, George, Greenberg, “Acceleration Techniques for Progressive Refinement Radiosity”, ACM, 1990

Remerciements

Je tiens à remercier tout particulièrement les personnes suivantes:

Enrico Gobbetti, pour son intérêt, ses conseils, ses directives et sa disponibilité

Francis Balaguer, pour son support et *Animator*

Les membres du LIG pour leur aide et leur bonne humeur

Daniel Thalmann, pour avoir accepté mon projet

Eric Chaubert, pour son enthousiasme

Martin Kunz, pour son sens inné du funboard

Stéphane 'Barberator' Barbey, pour sa philosophie

Vincent 'Vincenzo' Robert, pour son sens de l'humour

Bernard 'Jesus' Rytz, pour son calme et son chalet

Didier 'Wagen' Wagenknecht, pour son support technique

Frank 'Gland Blanc' Budel, pour l'animation

Jean-Marc 'Rony McVandel' Vandel, pour sa présence et sa barbe

Philippe Borgeaud, pour son optimisme face à la malchance

Isabelle Couve, pour sa présence féminine

Alex Merigay, pour son invisibilité et sa technique de squash

Laurent Stoeckli, pour son expertise des bières et ses batchs

Victor, pour ses batchs

Et bien d'autres encore...

Table des matières

Introduction 1
Première partie 2
Deuxième partie 2
Description des algorithmes de calcul 2
Algorithmes de calcul 2
Calcul de l'équilibre énergétique 2
Calcul de la radiosité et des ombres portées 5
Algorithme choisi 6
Subdivision adaptative 6
Application de textures 8
Calcul des effets dus aux modifications effectuées par l'utilisateur 10
Calcul en parallèle par tranche de temps 12
Optimisations 12
Représentation conceptuelle des objets 12
Optimisation du lancer de rayon 13
Optimisation du calcul de radiosité 14
Interaction avec le calcul 14
Interface 14
Définition de l'interface 14
Visualisation tri-dimensionnelle en temps réel et navigation 15
Projection de textures 17
Sélection des objets 17
Optimisations 18
Optimisation de l'affichage tri-dimensionnel 18
Réalisation 19
Structure des données 19
Implémentation de la coroutine de calcul 20
Problèmes de réalisation 21
Création et gestion des subdivisions et des sommets associés 21
Calcul des intersections rayon-triangle et initialisation des grilles de voxels 22
Conclusion 24
Mode d'emploi 25
Lancement du programme 25
Navigation dans une scène 25
Utilisation de la souris 25
Utilisation de la spaceball 25
Manipulation des scènes 25
Charger une scène 26
Sauvegarder une scène 26
Contrôle de l'application 26

Quitter l'application 26
Cadence de mise à jour 26
Angle d'ouverture 26
Vitesse de translation 27
Vitesse de rotation 27
Changement de mode (Navigation/Sélection) 27
Modification de la scène 27
Sélection d'un objet 27
Modification d'un objet sans texture 28
Modification d'un objet texturé 28
Conseil d'utilisation 29
Format du fichier d'entrée 29
Bibliographie 31
Remerciements 32
Table des matières 33