2/7/1991

Travail de 8ème semestre. Laboratoire d'Infographie. Calcul d'interréflexions diffuses (Radiosité).

Guy Moreillon
moreillo@disuns2.epfl.ch

Assistant: Enrico Gobbetti

Introduction

En infographie, le réalisme d'une image dépend pour beaucoup de l'illumination des objets qui composent la scène. Une des techniques les plus convainquantes est celle de la radiosité.

Le but de ce projet est de développer une application basée sur cette méthode. Le projet est divisé en deux parties, la première réalisée pendant le semestre d'été, la seconde pendant le travail pratique de diplôme.

Définition de l'application

Première partie

Dans cette 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 utilisé avec *gview* (fourni par SG) pour visualiser la scène sous tous les angles.

Deuxième partie

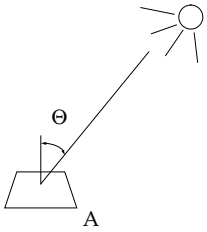
Dans la deuxième étape, l'application est rendue interactive. L'utilisateur peut visualiser la scène pendant le calcul, modifier les couleurs des objets, l'intensité des sources de lumière, les caractéristiques des matériaux. Il peut également forcer certains polygones à se subdiviser afin d'améliorer la qualité de l'illumination d'une surface.

Description de l'algorithme

Principe

Algorithme principal

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



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

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:

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

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

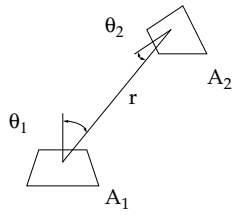
Un facteur de forme correspond à la quantité d'énergie quittant une surface i pour arriver sur une surface j. D'après la réciprocité des facteurs de forme, on a:

$$F_{ij}A_i = F_{ji}A_j \quad \text{(EQ 3)}$$

et en divisant (EQ 2) par A_i, on obtient:

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

ou, sous forme de matrice:



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

avec, pour les facteurs de forme:

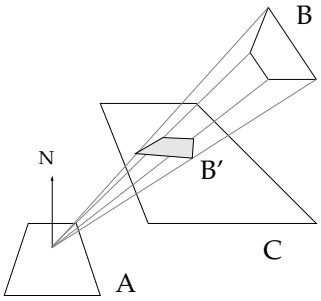
$$F_{A_i A_j} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \frac{\cos\theta_i \cos\theta_j}{\pi r^2} dA_j dA_i \quad \text{(EQ 6)}$$

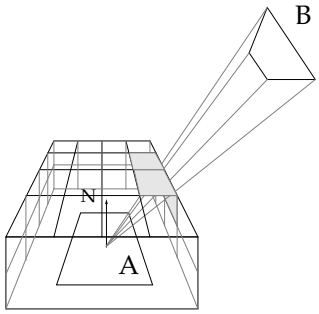
A partir de ces résultats, deux méthodes d'évaluation d'un solution au système d'équations sont envisageables. La première consiste à évaluer la totalité des facteurs de forme F_{ji}, résoudre le système grâce à la méthode de Gauss-Siedel ou une autre méthode similaire, puis afficher le résultat du calcul. Cette technique a plusieurs désavantages. Elle occupe un espace mémoire important (en $O(n^2)$, où n est le nombre de patch) 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. 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

Cette méthode permet de visualiser la scène au fur et à mesure du calcul de son illumination. 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.



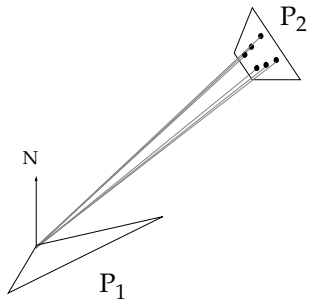


Calcul de radiosité

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. 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 obtiens 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 calculée 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. 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:

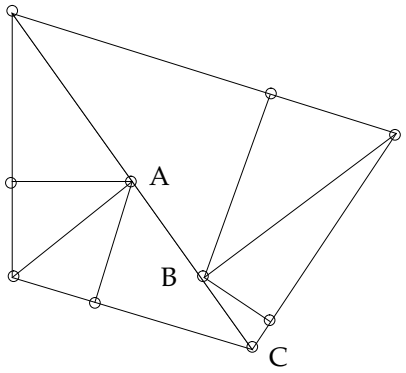
B_1	= La radiosité au point 1
B_2	= La radiosité au point 2 (source)
ρ_1	= La réflectivité de la surface 1
A_2	= L'aire de la surface 2
n	= Le nombre d'échantillons pris sur la source
δ_i	= La fonction de visibilité pour l'échantillon i

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).

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

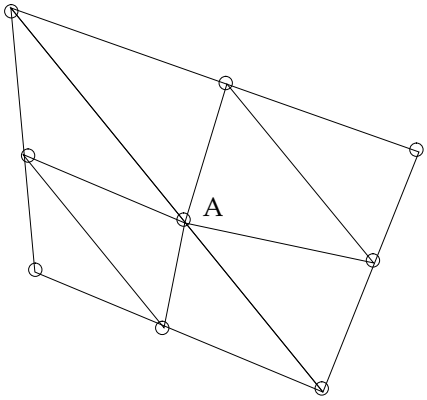
Contrairement à la plupart des programmes basés sur la radiosité, les patchs, c'est à dire les Polygones constituant les objets pures, ne sont pas ici des carrés, mais des triangles. Il s'agit en effet du plus petit élément de surface plane définissable exclusivement à l'aide de points. Il a l'avantage 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. Ce choix permet également d'introduire plus aisément une technique de raffinement du calcul: la subdivision adaptative des patchs.



◦ Point où la radiosité est calculée

Figure 1

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. Ça n'est qu'à l'affichage qu'une interpolation est faite 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 2

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 pure, 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.

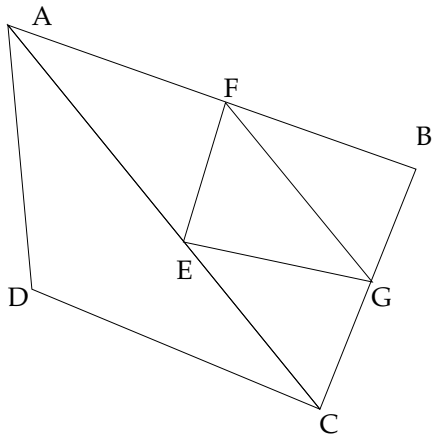
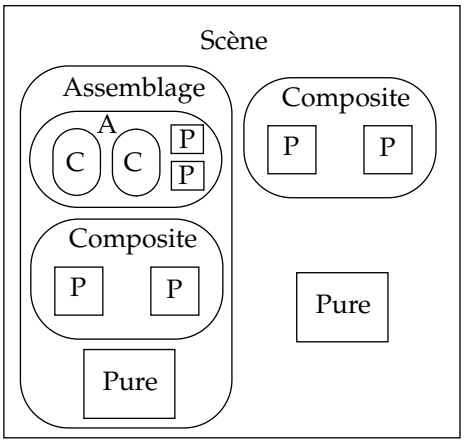


Figure 3

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 l'on considère à présent que le triangle ACD ne nécessite aucune subdivision, on aura alors à l'affichage une interpolation uniforme 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.

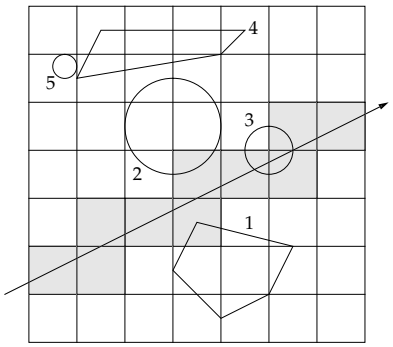
Structure hiérarchique des objets



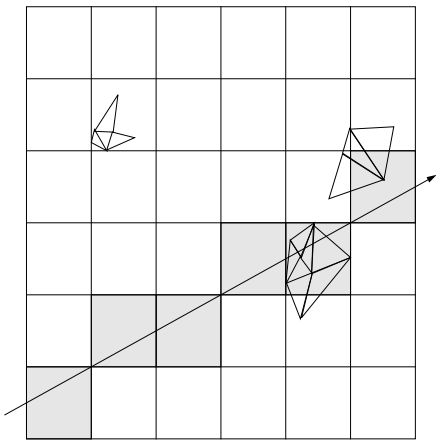
La structure hiérarchique des objets, est la suivante (voir figure): au plus haut niveau, une scène est composée d'*Assemblages*, de *Composites* ou de *Pures*. Un Assemblage est constitué lui-même d'Assemblages, de Composites ou de Pures. Un Composite ne peut être composé que de Pures, et enfin, un 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.

Techniques d'accélération

Le lancer de rayon



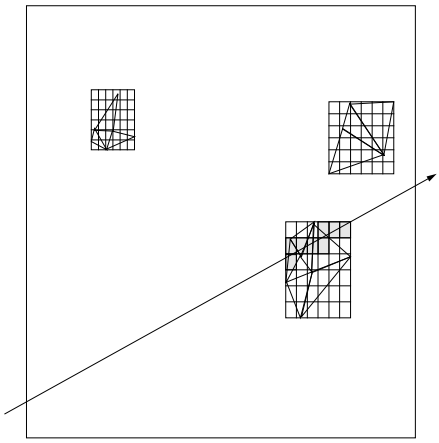
Equivalent 2D de la SUE



Grille de voxels pour la scène

De nombreuses techniques d'accélération existent pour le lancer de rayon. La plus appropriée dans notre cas est la subdivision uniforme de l'espace. 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.

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



Grilles de voxels pour les objets

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.

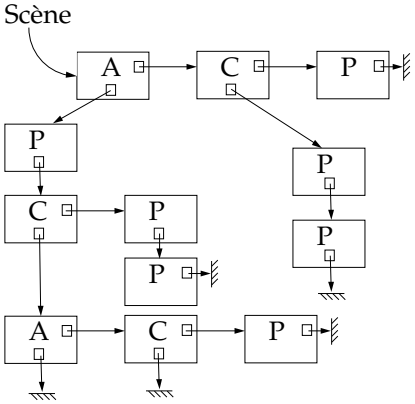
L'algorithme de lancer de rayon revient donc à la séquence suivante: parcourir la liste des objets constituant la scène. Pour chaque objet, si le rayon touche le volume englobant alors, s'il s'agit d'un Assemblage, tester les objets composant l'assemblage, s'il s'agit d'un Composite ou d'un Pure, lancer le rayon dans la grille de voxels associée. Si le rayon touche un polygone dans la grille, s'arrêter, le point est à l'ombre de la source de lumière.

Le calcul de radiosité

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.

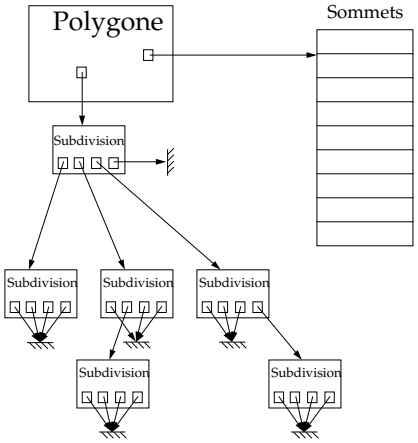
Réalisation

Structure des données



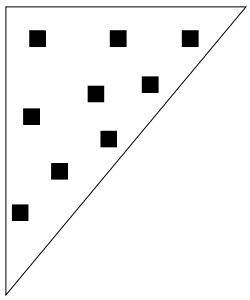
La structure interne des données est similaire à la représentation externe de la scène. Un objet est une structure à champs variables comprenant les informations communes à chaque objet, indépendamment de son type, ainsi qu'un champ formé d'une structure spécifique au type de l'objet (Assemblage, Composite 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 simplement 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.

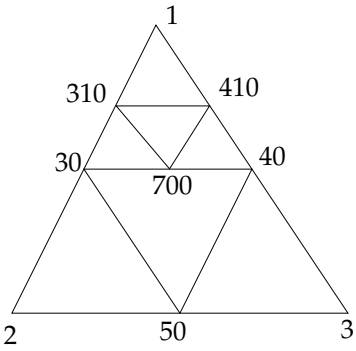


Un échantillon est une structure représentant un point d'échantillonnage sur la surface d'un polygone. Elle contient la position du point, un pointeur sur un élément de subdivision, et les coordonnées barycentriques du point dans l'élément. Le pointeur sur l'élément peut être nul si le polygone dans 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.

Problèmes de réalisation

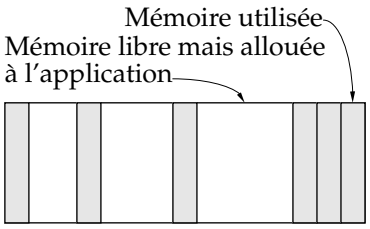
Création et gérance des subdivisions et des sommets associés



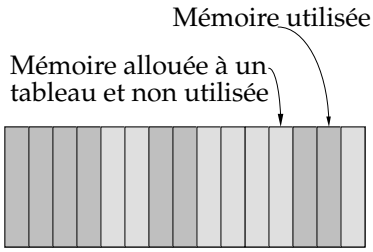
Numérotation des sommets

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 (voir figure), 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 hié-

rarchie 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.



Fragmentation de la mémoire



Allocation par blocs

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 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 décrémenté. 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.

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. 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 $\underline{n} = \underline{e}_a \times \underline{e}_b$ ou

$$\underline{u} = \frac{1}{|\underline{n}|}\underline{n}$$

contenant le point $\underline{c}$. Considérons un point $\underline{p}$ dont la position relative par rapport au point $\underline{c}$ est $\tilde{\underline{p}} = \underline{p} - \underline{c}$. Alors:

- $\underline{p}$ 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:

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

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:

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

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

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

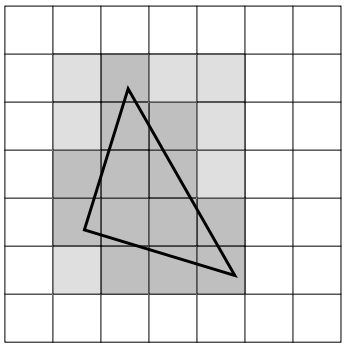
puis, si

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

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

Améliorations possibles

Le format du fichier d'entrée (voir plus loin) est relativement simple, et ne permet pas de manipulation des objets définis. Il serait utile de pouvoir déplacer ces objets, les tourner dans l'espace et les faire changer d'échelle grâce à des ordres simples. Une autre alternative serait de pouvoir lire en entrée directement un fichier au format *Animator* qui utilise des objets au format *SurfMan*, plutôt que d'utiliser un convertisseur comme c'est le cas actuellement. Dans ce dernier cas, il serait également intéressant de prévoir un algorithme permettant de créer automatiquement une hiérarchie optimale des objets afin que le lancer de rayon soit plus efficace.

D'autres améliorations pourraient être faites dans le sens d'une optimisation de l'algorithme comme par exemple la possibilité de spécifier si un objet crée une ombre portée, ou s'il est sensible aux ombres portées. Cela permet d'éviter de lancer un grand nombre de rayons souvent inutiles ou de petite importance.

Une autre optimisation 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.

Mode d'emploi

Lancement du programme

La ligne de commande a le format suivant:

```
> rad [-options] DescriptionFile GfoFile
```

Le fichier `DescriptionFile` est le fichier d'entrée décrivant la scène (voir plus bas pour le format de ce fichier). Le fichier `GfoFile` est le fichier produit par le programme, il est au format *gview*. Les options sont au nombre de deux. Elles peuvent être spécifiées les deux dans n'importe quel ordre, une seule au choix ou aucune. Elles sont les suivantes:

`-noshadows`

qui empêche le calcul des ombres portées pour toute la scène, et

`-minsize` *`float`*

qui spécifie la taille minimale des côtés des subdivisions. Si cette dernière options n'est pas utilisée, une valeur sera calculée automatiquement mais elle risque de ne pas être aussi bien adaptée à la scène qu'une valeur spécifiée.

Une fois l'application lancée, certains messages apparaissent, reflétant le stade d'exécution du programme:

```
> rad scene5.des scene5.gfo
Reading description file...
Reading the geo file: off/cube.geo
Done reading file.
Initializing...
Starting Computation
----\
Writing results in file scene5.gfo
```

Si la syntaxe du fichier d'entrée n'est pas correcte, un message d'erreur apparait, indiquant la ligne à laquelle se trouve l'erreur. L'exécution du programme est alors interrompue. Une fois le calcul lancé, le programme affiche un '-' chaque fois qu'une centaine de subdivisions ont été créées, et chaque fois que la patch source change, un '/' ou un '\' est affiché alternativement.

Une fois le calcul terminé, le résultat est écrit dans un fichier texte, et des statistiques sur le lancer de rayon sont affichées:

```
Ray-Tracing Statistics:
Number of rays shot:                  33361
Number of hits on polygons:            4894 (14.67%)
Number of trials on bounding boxes: 159469
Number of hits on bounding boxes:     53189 (33.35%)
Number of intensity cuts:             39751

Total elapsed time is 22 seconds
```

Format du fichier d'entrée

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

Scène:==	Chaîne Composition
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:==	**Reflectivity** Réel **Color** Vecteur **Emission** Réel
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

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

Table des matières

Introduction 1
Définition de l'application 1
Première partie 1
Deuxième partie 2
Description de l'algorithme 2
Principe 2
Algorithme principal 2
Calcul de radiosité 4
Subdivision adaptative 5
Structure hiérarchique des objets 6
Techniques d'accélération 6
Le lancer de rayon 6
Le calcul de radiosité 7
Réalisation 7
Structure des données 7
Problèmes de réalisation 8
Création et gérance des subdivisions et des sommets associés 8
Calcul des intersections rayon-triangle et initialisation des grilles de voxels 9
Améliorations possibles 11
Mode d'emploi 12
Lancement du programme 12
Format du fichier d'entrée 13
Bibliographie 14
Table des matières 15