

TD d'infographie n°1

Cours d'infographie
Master 2 : Science informatique option ISI

Radiosité

Calcul de l'illumination globale d'une scène par la méthode de Radiosité. Application sous OpenGL.

Le but de ce travail est de calculer l'éclairage global d'une scène virtuelle (simple). Pour ce faire nous utiliserons la méthode dite de radiosité. Elle consiste grossièrement à calculer tous les échanges lumineux diffus entre toutes les surfaces de la scène. Dans notre application, nous exploiterons une approche « par point » issu de l'article « A Ray Tracing Algorithm for Progressive Radiosity » de Wallace, Emquist & Haines, SIGGRAPH 89. La méthode fonctionne en positionnant des points sur les surfaces et en faisant un échange lumineux entre disques (représentés par les points) et chaque point illuminé. Puis la technique de la radiosité progressive est employée. Elle consiste à sélectionner, parmi les points distribués sur les surfaces, celui qui a la radiosité latente la plus grande. La radiosité latente consiste en la radiosité non encore envoyée sur les autres surfaces. Une fois le point émetteur sélectionné, on procède à « l'échange lumineux » entre ce point et tous les autres de la scène. On itère ce processus jusqu'à ce que la radiosité latente maximale descende en dessous d'un seuil fixé par le programme.

Le facteur de forme entre un point de lumière émetteur A posé sur un disque de rayon r et un point lumineux récepteur B est donné par la formule suivante :

$$F_{A \rightarrow B} = \frac{\cos(\theta_i) \cos(\theta_j)}{\pi d^2 + \pi r^2} V(A \rightarrow B)$$

où θ_i (resp. θ_j) est l'angle formé par la normale en A (resp. en B) et la direction d'émission, d la distance entre A et B et enfin V la visibilité entre ces deux points.

Lors de l'échange lumineux entre A (l'émetteur) et B (le récepteur), la radiosité de B, ainsi que sa radiosité latente, s'incrémente de :

$$\Delta B_B = B_{lat, A} * \rho_B * F_{A \rightarrow B}$$

où B_{lat} est la radiosité latente, et ρ_B le coefficient de réflexion diffuse de B

Ressource : Le programme de base

Vous disposez d'un ensemble de classes en C++ qui vous permettront de réaliser la méthode de radiosité. Rien de sophistiqué concernant le C++ n'apparaît dans ces classes sauf pour la classe *Artefact* qui est une classe abstraite de laquelle hérite la classe *Quad*, et la classe *Scene*, classe abstraite, dont hérite *SceneCornel*.

Les fichiers *tools.[ch].pp* permettent d'avoir des fonctions utilitaires. Les fichiers *mii2_rad.[hc].pp* contiennent l'application glut (IHM principalement). Le fichier *mii2_rad.hpp* contient toutes les « variables globales » et les constantes que vous utiliserez dans le programme.

La classe *Rayon* (fichiers *rayon.[hc].pp*) permet de représenter un rayon. De même la classe *Vector3D* (fichiers *vector3d.[hc].pp*) est une classe permettant la représentation d'un vecteur à trois dimensions. **Attention, dans cette implémentation, le produit vectoriel est noté $*$ et le produit scalaire $\wedge$.** Enfin la classe *ImageTexture* (fichiers *imageTexture.[hc].pp*) représente une image qui peut être chargée en tant que texture (c'est exactement la même classe que dans les TD de GPGPU).

La classe *PointLight* permet de représenter les points lumineux appelés aussi VPL (pour Virtual Point Light) sur lesquels la radiosité va être calculée et échangée. On y retrouve des éléments géométrique (position, normale), ainsi qu'un rayon qui symbolise le disque recouvrant la surface en ce point. Enfin, le *PointLight* est munie d'une valeur (triplet RGB sous la forme d'un *Vector3D*) de radiosité ainsi qu'une valeur de radiosité latente (aussi un triplet RGB).

Plus important, la classe *Artefact* (fichiers *artefact.[ch].pp*) représente les éléments géométriques paramétriques de la scène (les objets). Cette classe est une classe abstraite pure (certaines fonctions notées **virtual** ne sont pas implémentées). Les différents éléments géométrique de la scène doivent hériter de cette classe comme la classe *Quad*. La classe *Artefact* contient les propriétés radiométriques de l'objet (coefficient de réflexion diffuse, émissivité), la texture de radiosité ainsi que des identifiants et deux entiers (*dimension_w* et *dimension_h*) indiquant le valeur de la discrétisation sur chacun des deux paramètres (surfaces paramétriques). Il est donc important de comprendre que seuls des surfaces paramétriques pourront être représentées dans cette application.

Enfin la classe Scene (fichiers Scene.[hc]pp) représente la scène virtuelle elle-même. C'est elle qui va ordonnancer les calculs, dessiner la scène et, réaliser l'algorithme de radiosit . Cette classe doit  tre h rit e pour chaque sc ne virtuelle que l'on souhaite cr er (et donc remplir la fonction makeScene). Une classe (SceneCornel) vous permet d'avoir un exemple d'impl mentation de sc ne.

Pour le moment l'application se contente de n'afficher en OpenGL que les surfaces avec, pour couleur, la valeur de leur coefficient de diffusion. Vous pouvez activer ou d sactiver cette visualisation en appuyant sur la touche F1. La touche F2 vous permet de voir directement les points lights. Enfin la touche F3 vous permet de voir les textures de radiosit . Les fl ches ainsi que les touches 'Page Up' et 'Page Down' vous permettrons de vous d placer au sein de la sc ne.

Exercice 1 : Impl mentation de la radiosit 

1. T l chargez sur le site Internet <http://www-igm.univ-mlv.fr/~biri/> : l'archive du programme. D zippez, d tarrez et compilez.
2.  tudiez chacune des classes et plus particuli rement PointLight, Artefact, Quad, Scene et SceneCornel.
3.  tape num ro 1 : g n ration des VPL.
 1. A l'initialisation, la sc ne doit demander   toutes les surfaces de g n rer les PointLights. Il faut donc appeler et impl menter la fonction generateAllVPL de la classe Scene.
 2. Il faut donc  galement impl menter la fonction generateVPL de la classe Quad. Cette fonction devra renvoyer les VPL sous forme d'un vecteur C++ (vector). La fonction prend pour argument un r el qui repr sente une estimation de la distance devant s parer chaque VPL. Ainsi, pour le Quad, si cet argument vaut 0.1, alors les dimensions du quadrilat re doivent  tre divis  par cette valeur pour savoir le nombre de point   cr er sur chaque dimension. Ces nombres de points doivent  tre stock s dans les variables dimension_w et dimension_h de la classe Artefact. La fonction g n re les PointLight et les renvoie dans un vecteur C++.
4.  tape num ro 2 : calcul de la radiosit .
 1. R alisation de la fonction de s lection du PointLight  metteur. Cette fonction selectOneEmitter appartient   la classe Scene. Pour l'impl menter, vous devrez  galement impl menter la fonction getEnergyLatente de la classe PointLight. La fonction selectOneEmitter renvoie l'identifiant de l'objet et du PointLight qui a le plus d' nergie latente. La fonction retourne aussi un bool en exprimant si la radiosit  latente est sup rieur ou non   la constante MINIMUM_RADIOSITY
 2. Impl mentez la fonction shootOneRad de la classe Scene. Cette fonction doit r aliser le « tir » de la radiosit  latente d'un PointLight vers l'ensemble des autres VPL (sauf lui m me). Vous devrez pour ce faire impl mentez la fonction exchangeRad de la classe PointLight. **Ne prenez pas en compte les probl mes de visibilit .** A la fin du « tir », il faut  galement annuler la radiosit  latente sur le PointLight  metteur (fonction removeLatRad   appeler et impl menter).
 3. Enfin, impl mentez la fonction loopRadiosity qui r alise des « tirs » de radiosit  tant que celle-ci ne d passe pas le seuil d fini par MINIMUM_RADIOSITY. A la fin de la boucle, vous devriez pouvoir voir les couleurs des VPL
5.  tape num ro 3 : affichage des textures de radiosit 
 1. Impl mentez la fonction generateAllTextureRad de la classe Scene
 2. Impl mentez la fonction generateTexture de la classe Artefact (devrait en fait  tre dans la classe Quad). Cette fonction remplit une image (texture_rad) avec les couleurs des VPL. Faites bien attention   convertir en unsigned char et   l'ad quation entre l'espace param trique et l'espace image.

Exercice 2 : Prise en compte des occlusions

1. Faites en sorte qu'  chaque  change de radiosit  entre un  metteur et un r cepteur, on v rifie que les deux soient bien visibles. Pour ceci vous pourrez vous appuyer sur la classe Rayon et les fonctions virtuelles getIntersection et isIntersected.
2. Impl mentez une nouvelle classe h ritant de sc ne avec une sc ne mettant en valeur les occlusions.