PROJET TUTEURÉ

CLUSTERING : L’union fait la force

Sommaire

1. But du projet

Montrer que la combinaison de plusieurs ordis (de faible puissance de calcul) permet la réalisation d'une tache nécessitent une puissance de calcul importante.
L'application que nous avons choisi pour montrer cette puissance est une application qui crée des images par lancer de rayon (RAYTRACING qui sera expliqué dans la suite).

2. Moyens proposés par l'IUT

Des vieux ordinateurs dont l'IUT n'a plus besoin, qui ne fonctionnent plus ou qui sont incomplets stockés dans le sous-sol, des pièces détachées dont on ne connaît pas l'état de marche. Tout le mat1érielest entreposé dans un petit local dont nous avons constamment l'accès. Nous avons aussi l'accès à l'Internet, ainsi qu'un hub huit ports afin de connecter nos machines . Au niveau logiciel, Fréderic, l'administrateur réseau nous a fourni DEBIAN version 3.0 sur CD.

3. Travail de groupe

Répartition des tâches

Dans un premier temps, tout le groupe s'est occupé de l'assemblage des ordinateurs. Pour organiser le travail, nous avons mis en place un système de mémoire sous forme de cahier dans lequel nous récapitulons chaque tâche accomplie.
Ensuite, nous avons formé deux groupes, afin de mieux avancer sur le travail. Un groupe s'occupant du raytracing et l'autre du pvm. Celui du raytracing avait pour but de trouver un programme en c/c++ fonctionnant sur les vieux ordinateurs afin de modifier son code pour développer le partage de calcul. Le second groupe s'est chargé d' installer pvm et d' assimiler son fonctionnement dans le but de fournir les primitives de fonctions nécessaires au partage du raytracing.

Répartition temporelle

La salle étant petite, nous avons mis au point un système de rotation de l'occupation de l'espace de travail entre les groupes.
Une réunion est fréquemment organisée pour permettre un échange de nos travaux et définir les objectifs à atteindre pour les prochaines séances.

4. Confection des ordinateurs

Recherche des pièces & inventaire

Nom	Processeur	RAM	Disque dur	Système d'exploitation	Mode graphique	Adresse IP	Carte graphique	Carte réseau
PT1	Pentium 90	40Mo	535Mo	Linux Debian 3	Non	192.168.1.2 /24	S3 Trio 32	3Com Ether-link III
PT2	486 DX4	32Mo	400Mo	Linux Debian 3	Non	192.168.1.3 /24	Cirrus logic	3Com Ether-link 16
PT3	Pentium 133	64Mo	1700Mo	Linux Debian 3	Non	192.168.1.4 /24	S3 Trio 64	3Com Ether-link III
PT4	Pentium 100	40Mo	1700Mo	Linux Debian 3	Non	192.168.1.1 /24 10.1.26.202 /24	S3 Trio 64V+	3Com Ether-link III 3Com Ether-link III
PT5	Pentium 133	16Mo	1000Mo	Linux Debian 3	Non	192.168.1.5 /24	Cirrus logic	3Com Ether-link III
PT6	Celeron 433	128Mo	Mo	Linux Mandrake	Oui	192.168.1.6 /24	Intégrée	Intégrée

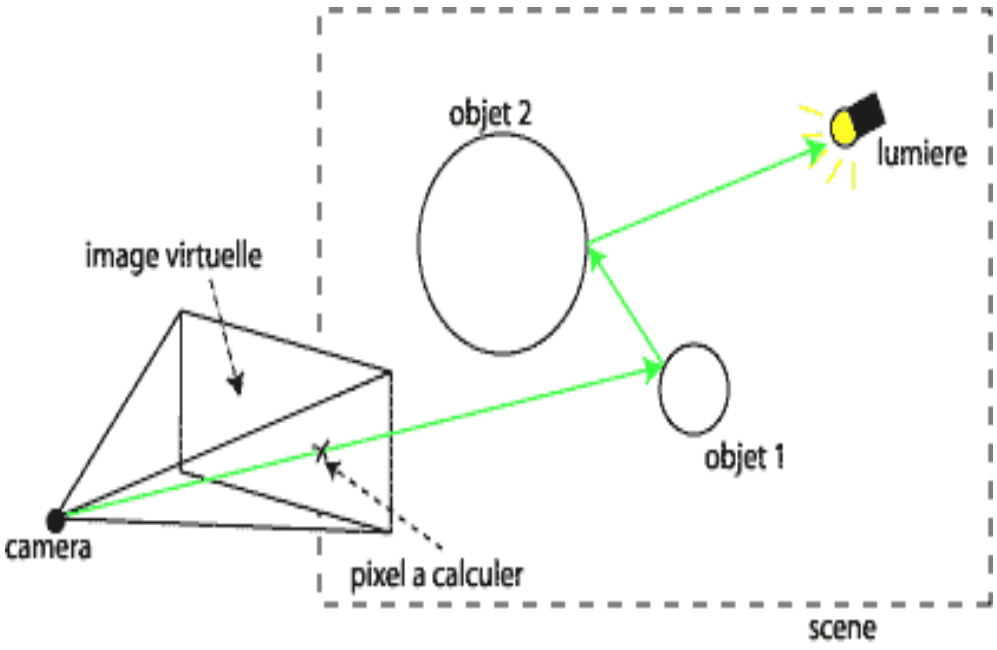
Problèmes rencontrés

Dans la confection des ordinateurs nous nous sommes heurté à plusieurs problèmes. En effet dans un premier temps, la recherche des pièces encore valables a été plutôt « périlleuse », il fallait trier parmi les nombreuses et vieilles machines les pièces encore en état de marche et rassembler le tout. Une fois les ordinateurs montés, il fallait installer l'OS (système d'exploitation). Cette étape a été aussi difficile. L'installation de Debian avec le CDROM est compromis car nous ne pouvions pas booter dessus. Il a fallu faire une détection de CDROM à partir de disquettes de boot. Ces disquettes permettaient de lancer l'installation de DEBIAN à partir des Cds. Ce problème résolu, l'installation de DEBIAN fut très longue car il fallait choisir chaque package (fonctionnalité). Les soucis ne s'arrêtent pas ici. Les fonctionnalités graphiques de DEBIAN sur les vieilles machines laissent beaucoup à désirer ! Ainsi tous les ordinateurs ont été basculés en mode texte (console) à l'exception d'un seul, celui de madame Briolle qui est assez performant pour afficher le résultat du lancer de rayons en mode graphique.

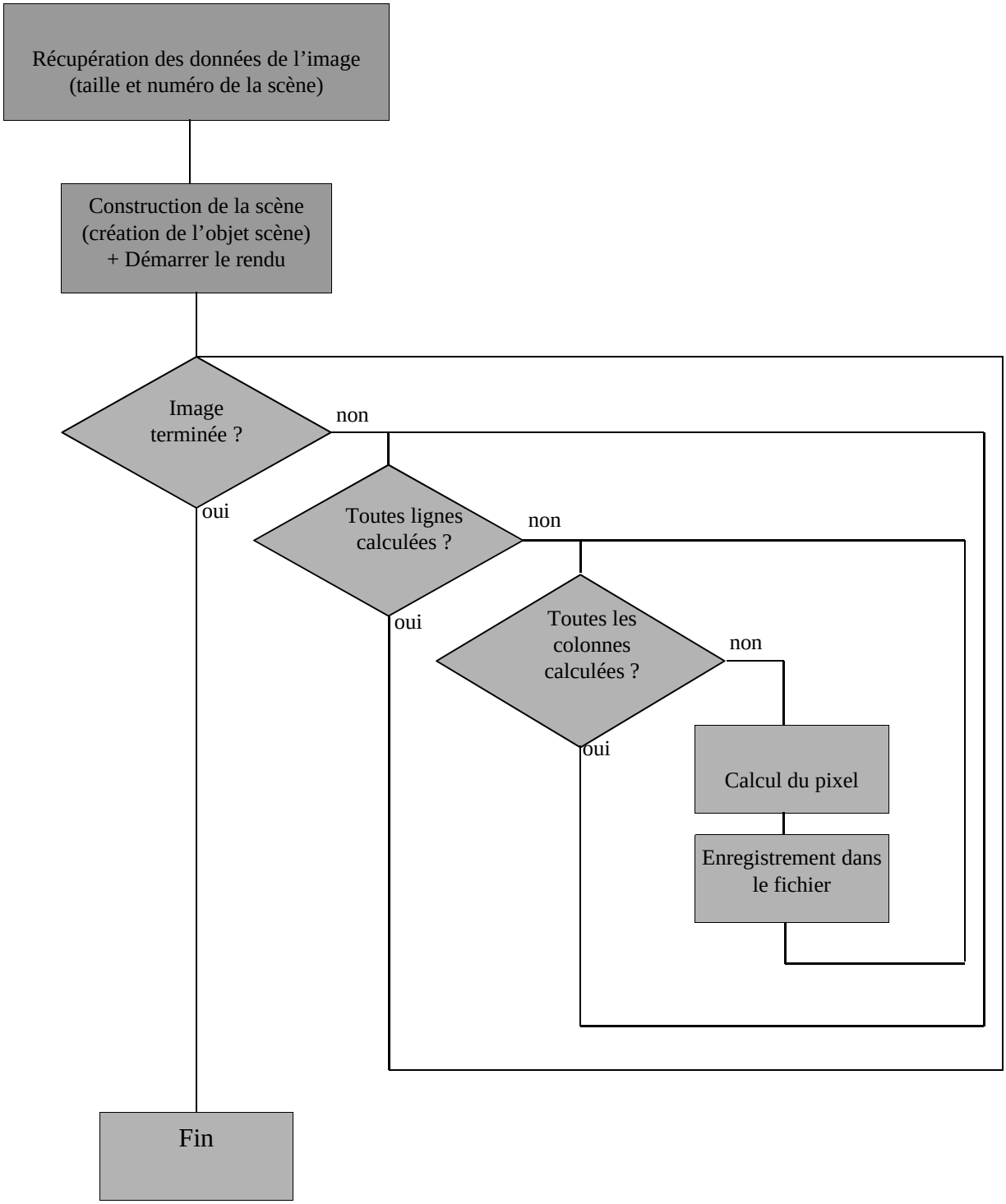
5. Raytracing

- La vision naturelle : Des rayons sont émis depuis une ou plusieurs sources lumineuses, viennent frapper des objets, et sont réfléchis et/ou transmis selon les caractéristiques (le matériau) de ces objets. Une partie de ces rayons arrive enfin sur l'œil (après avoir été réfléchis ou directement transmis), ce qui forme une image sur la rétine. Cette technique est inapplicable par informatique. La réalité est qu'il y a une trop grande quantité (voir une infinité) de rayons émises par une source de lumière pour qu'une machine quelconque puisse calculer tous les lancers.

- Principe général du lancer de rayons : Cette technique tente de reconstituer le parcours inverse de la lumière, depuis la caméra en direction des sources lumineuses. Pour cela, on place une image virtuelle dans la scène devant l'observateur (c'est cette image qui constitue le résultat final). Pour chacun des pixels qui constituent l'image, on lance un rayon partant du point de vue (l'observateur) passant par le centre du pixel. La couleur du pixel traversé va être déterminée en suivant le cheminement du rayon lancé jusqu'aux sources lumineuses de la scène 3D. Cette technique permet donc de ne calculer que les rayons qui seraient vus par l'œil, ce qui rend les temps de calcul acceptables.



Le principe de fonctionnement simplifié d'un raytracer est le suivant :



6. PVM

PVM est un ensemble logiciel permettant de construire virtuellement un calculateur parallèle à partir d'un ensemble de machines géographiquement distribués et hétérogènes, c'est-à-dire, composées de processeurs différents, fonctionnant sous des systèmes d'exploitation éventuellement différents. Plus précisément, il s'agit d'une bibliothèque de communications (inter-processus) par échange de messages.

En d'autres termes, une fois installé, PVM permet de faire communiquer entres elles différentes machines au moyen de *messages* que s'échangent les processus.

Le développement de PVM débuta en 1989 au laboratoire d'Oak Ridge et est encore aujourd'hui largement utilisé. Ce projet est à la base de la définition d'un standard dans le domaine : MPI (*Message Passing Interface*, http://www.mpi-forum.org).

Sous PVM, un utilisateur peut regrouper un ensemble de machines séquentielles, vectorielles aussi bien que parallèles en un large ordinateur à mémoire distribuée. On utilise le terme de *machine virtuelle* pour désigner cette machine logique à mémoire distribuée. On appelle également hôte (*host*) l'un de ses constituants de la machine virtuelle. Un hôte est une machine qui accepte d'héberger et de traiter une application PVM.

Cette bibliothèque permet d'utiliser un réseau de machines comme un calculateur parallèle avec de notables avantages par rapport à une machine parallèle ``classique" :

- Si l'une des machines composant le réseau tombe en panne, les autres peuvent continuer de fonctionner, la machine virtuelle est simplement diminuée d'une machine.
- La montée en puissance d'une machine ou l'ajout d'un ensemble de nouvelles machines est effectuée de manière transparente pour la machine virtuelle.
- Des machines usuellement incompatibles pour des raisons architecturales ou logicielles (systèmes d'exploitation différents (HP-UX, Linux, Solaris, Windows)) peuvent coopérer pour le traitement d'une application PVM.

PVM fournit des mécanismes pour démarrer automatiquement des *tâches* sur la machine virtuelle, et pour leur permettre de *communiquer* et de se *synchroniser* les unes par rapport aux autres. Une tâche est associée à un processus UNIX.

Une application peut être parallélisée en utilisant les mécanismes d'*échanges de messages* (*message-passing*) communs à la majorité des ordinateurs à mémoire distribuée. En envoyant et en recevant des messages, plusieurs tâches d'une application peuvent coopérer pour résoudre un problème en parallèle.

Le système PVM est composé de deux parties. La première est un *démon*, appelé *pvmd3* et souvent abrégé *pvmd*. Ce démon réside sur chaque élément de la machine virtuelle. Pour exécuter une application PVM, un utilisateur doit tout d'abord établir sa machine virtuelle en démarrant PVM. L'application PVM peut alors être démarrée depuis n'importe quelle machine appartenant à la machine virtuelle. Plusieurs utilisateurs peuvent définir chacun une machine virtuelle différente avec la possibilité pour une machine d'être *host* pour plusieurs machines virtuelles. Chaque utilisateur peut exécuter plusieurs applications PVM simultanément.

La seconde partie du système PVM est constituée par une librairie des routines d'interface de PVM (*libpvm3.a*). Cette librairie contient les routines utilisables par les utilisateurs pour démarrer des tâches, les synchroniser, échanger des messages... Les applications doivent donc être liées avec cette librairie pour utiliser PVM.

Lorsque l'environnement a été mis en place, à partir d'une connexion sur une quelconque machine appartenant à cette machine virtuelle, on peut effectuer des calculs sur tout ou partie de cette machine virtuelle. Cela ne peut être fait qu'à la condition que les machines acceptent des connexions comme *rlogin* ou *rsh* (*remote shell*) ou *rcp* sans demande explicite de mot de passe. Ceci nécessite la mise en place d'un certain nombre de fichiers pour conserver un niveau acceptable de sécurité.

7. Raytracing & PVM

Notre but n'étant pas de développer un raytracer, nous avons choisi de chercher les sources d'un programme existant en licence GNU ce qui nous permet de les modifier à notre guise.

Problème rencontré : une multitude de programme de lancer de rayons existent mais il ne fonctionnait pas sur les machines que nous possédons car elles ont peu de mémoire.
Nous avons donc voulu en adapter un, mais cette opération c'est révélée très compliquée.
Finalement, nous avons trouvé un programme développé par Micha Riser (mriser@gmx.net) en langage C++.
Nous ne connaissions pas ce langage : il est proche du C mais orienté objet (comme le JAVA).
Il a donc été facile de s'y adapter.

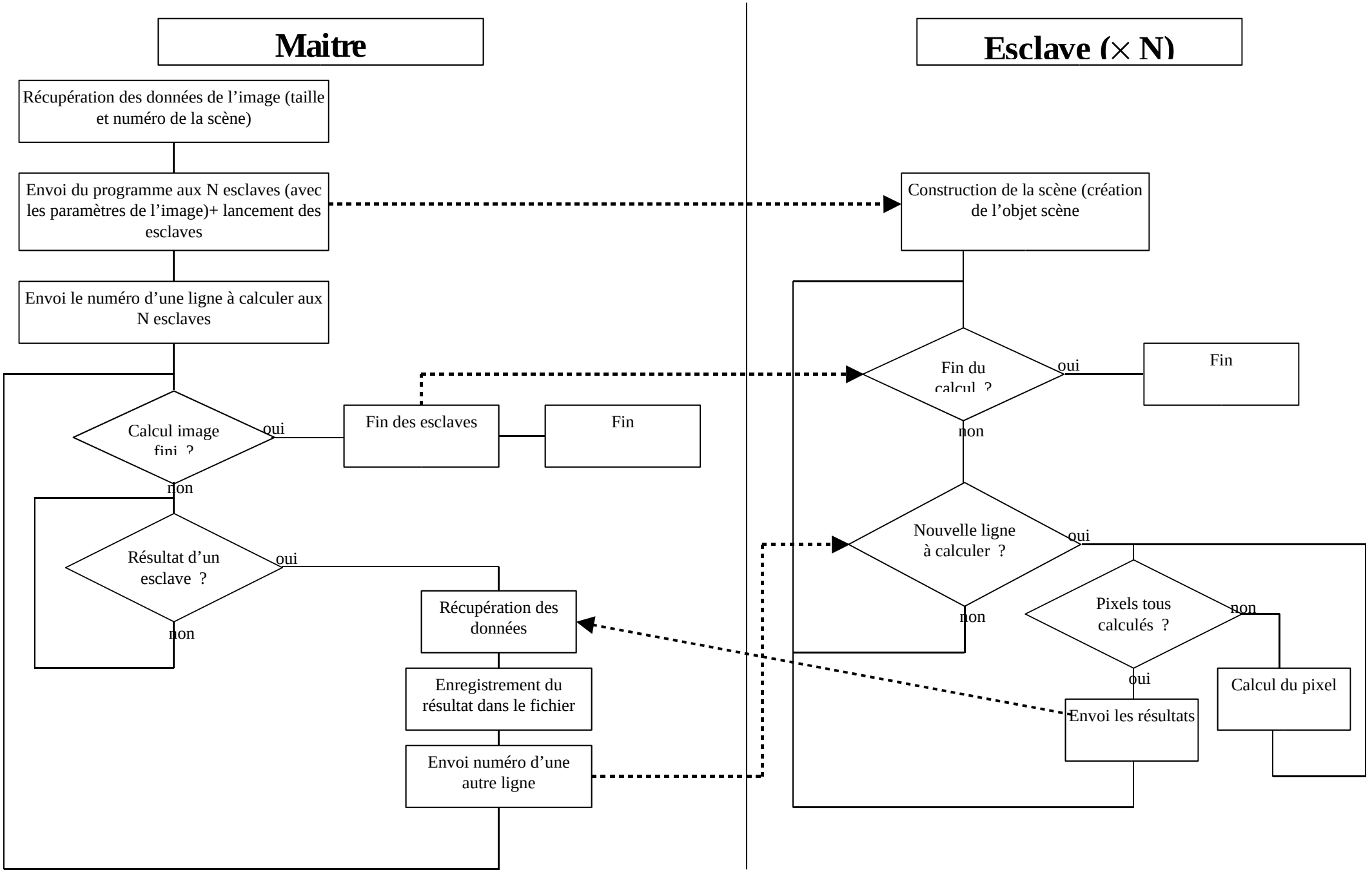
Rôle du maître

Le maître a pour but de distribuer le calcul de l'image aux esclaves. Par conséquent il doit aussi réunir les données calculées par les esclaves dans le fichier de sortie.
Pour obtenir de bonnes performances (ne pas encombrer le réseau), le maître ordonne à chaque esclave de calculer une ligne complète avant de lui renvoyer les données.

Rôle de l'esclave

L'esclave doit effectuer le calcul des lignes demandées par le maître et lui renvoyer les résultats.

Le fonctionnement simplifié du programme une fois modifié est sur le schéma de la page suivante .


Maitre
Esclave (× N)
Récupération des données de l'image (taille et numéro de la scène)
Envoi du programme aux N esclaves (avec les paramètres de l'image)+ lancement des esclaves
Envoi le numéro d'une ligne à calculer aux N esclaves
Construction de la scène (création de l'objet scène
Fin du calcul ?
oui
Fin
non
Calcul image fini ?
oui
Fin des esclaves
Fin
non
Nouvelle ligne à calculer ?
oui
non
Résultat d'un esclave ?
oui
non
Pixels tous calculés ?
non
oui
Calcul du pixel
Récupération des données
Enregistrement du résultat dans le fichier
Envoi numéro d'une autre ligne
Envoi les résultats

8. Annexes

Sources de documentation

PVM : Laboratoire d'Informatique du Havre
(http://www-lih.univ-lehavre.fr/~guinand/Enseignement/Maitrise/TP/TP1/)

Installation pvm

Première méthode : Installation par compilation

Création de répertoire et de variables d'environnement :

$>mkdir /root/PVM
On crée un répertoire pvm afin de ranger PVM (pvm3.4.3)

Copie les lignes de commande dans le répertoire de démarrage .bashrc:
$>echo export PVM_ROOT=/root/PVM/pvm3 >> $HOME/.bashrc
$>cat /root/PVM/pvm3/lib/bashrc.stub>> $HOME/.bashrc
Permet de créer les variables d'environnement
$>source .bashrc

compilation de PVM dams le répertoire PVM
$>make

Problème rencontré : cette solution est compliquée à mettre en œuvre et nous n'avons pas réussi à la développer sur tous les ordinateurs.

Deuxième solution : Installation de PVM par package, elle beaucoup plus simple car c'est le le système d'exploitation qui gère cette installation.
Manipulation :
$>dselect
=>affiche un menu pour installer des packages

Primitives pvm utiles

pvm_mytid

C'est une instruction obligatoire en début de code PVM qui permet de raccrocher le processus à un démon `pvmd`. Sa syntaxe est:

```
int tid;
tid=pvm_tid();
```

Cela renvoie `tid` qui est l'identificateur de tâche du processus. Ce `tid` permet de définir de façon unique ce processus et de s'adresser à lui. Il y a un cas d'Erreur si cela renvoie `tid < 0` (démon PVM inexistant par exemple).

pvm_exit

C'est une instruction (presque) obligatoire en fin de code PVM. Elle permet de dire au démon `pvmd` que la tâche quitte PVM.

```
int info;
info=pvm_exit();
```

Il y a un cas d'Erreur si cela renvoie `info < 0`.

pvm_spawn

C'est l'instruction qui permet de lancer `n` tâches PVM exécutant un même code (`"programme"`):

```
int numt;
numt=pvm_spawn(char *task,char **argv,int flag,char
                          *where,int ntask,int *tids);
int numt,etid[n];
numt=pvm_spawn("programme",NULL,PvmTaskDefault,NULL,n,&etid
[0]);
```

Le `tid` de chacune des tâches est rangé dans une entrée du tableau `etid`. Cette instruction Renvoie `numt <= n` qui est le nombre de tâche(s) effectivement lancée(s).

Pour les autres champs:

- `argv` est la liste d'arguments (type ligne de commande UNIX) à passer à `"programme"`,
- `flag` donne des options à `pvm_spawn`,
- pour certaines de ces options `where` indique sur quel type d'hôte lancer les processus.

pvm_parent

Cette instruction retourne `tid` valant le `tid` de son processus parent (celui qui a crée le processus courant):

```
int tid;
tid=pvm_parent();
```

Elle peut également renvoyer `tid=PvmNoParent` si le processus courant n'a pas été crée par `pvm_spawn`.

pvm_initsend

C'est une instruction obligatoire avant l'envoie d'un message: elle initialise le tampon d'envoie des données:

```
int info;
info=pvm_initsend(int encoding);
```

`encoding` est un entier permettant de spécifier comment coder les données (surtout utile en milieur hétérogène). Ici on emploiera toujours `PvmDataDefault`. Il y a une Erreur si `pvm_initsend` renvoie `info < 0`.

pvm_pk...

Ce sont les fonctions qui effectuent le compactage de différentes données d'un processus, destinées à être envoyées, dans le tampon d'envoi:

```
int info;
info=pvm_pkint(int *ip,int nitem,int stride);
```

- `ip` est un tableau d'entiers d'au moins `nitem*stride` de long. `ip` peut être également un pointeur sur une variable entière.
- `nitem` est le nombre d'éléments de `ip` que l'on veut compacter dans le tampon d'envoi.
- `stride` est le décalage entre deux éléments de `ip` à compacter. Par exemple, si `int ip[6]; stride=2; nitem=3;` alors les éléments compactés de `ip` seront `ip [0], ip[2], ip[4]`.

Il y a eu une Erreur si cela renvoie `info < 0`. Il y a une fonction de compactage par type de données possible, donc on trouvera également `pvm_pkfloat`, `pvm_pkdouble`, `pvm_pkstr` etc.

pvm_send

Cette instruction envoie un message de façon non-bloquante:

```
int info;
info=pvm_send(int tid,int msgtag);
```

`tid` est l'identificateur de processus du destinataire du message. `msgtag` est un entier identifiant une classe de messages. Cela permet de filtrer les messages (à la réception). Il y a un cas d'Erreur si elle renvoie `info < 0`.

pvm_upk...

Elles correspondent aux instructions `pvm_pk...`:

```
int info;
info=pvm_upkint(int *ip,int nitem,int nstride);
```

Elles ont les mêmes arguments que pour les fonctions `pvm_pk...`. Il y a une Erreur si elles renvoient `info < 0`.

pvm_recv

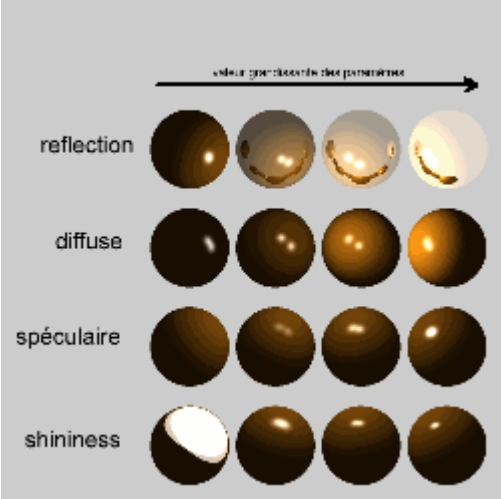
Cette instruction reçoit un message de type `msgtag` de l'expéditeur `tid`:

```
int bufid;
bufid=pvm_recv(int tid,int msgtag);
```

Elle bloque le processus receveur tant que le message n'est pas reçu. Si `tid` vaut -1, peut recevoir de n'importe qui. Si `msgtag` vaut -1, peut recevoir de n'importe qui. Il y a un cas d'erreur si `pvm_recv` renvoie `info < 0`.

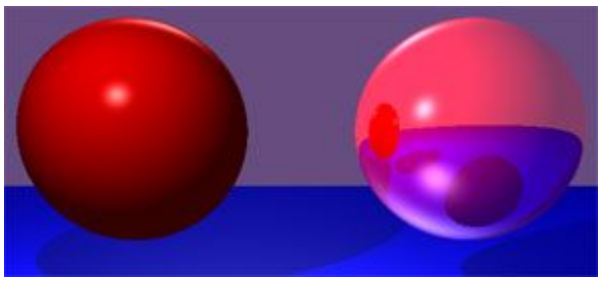
Principe du raytracing

Modélisation des phénomènes optiques



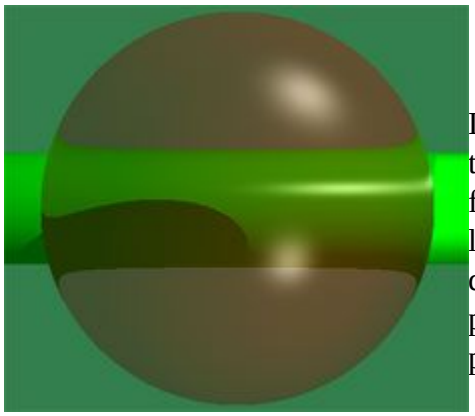
Le lancer de rayons utilise les propriétés physiques et mathématiques réelles de la lumière, ce qui permet d'obtenir des résultats photo-réalistes. Les phénomènes optiques qui sont rendus par le moteur de lancer de rayons sont la réflexion, la réfraction et les ombres.

La réflexion



Si la surface d'un objet est réfléchissante, l'algorithme de lancer de rayons doit non seulement calculer la valeur de l'illumination sur la surface, mais aussi ajouter les reflets des autres objets de la scène. Pour ce faire, il suffit uniquement de connaître l'angle auquel part le rayon réfléchi, puis de lancer un nouveau rayon dans cette direction pour déterminer ce qui se reflète. Le calcul de l'angle de réflexion est immédiat dès que l'on connaît la valeur de la normale à la surface. On a le vecteur directeur du rayon réfléchi par la formule

La réfraction



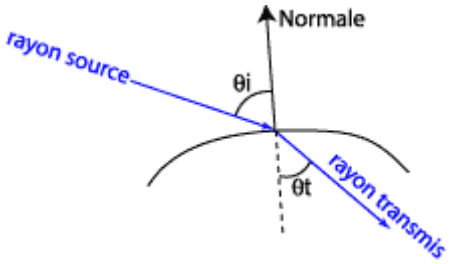
La réfraction permet de prendre en compte la transparence des objets de la scène. Un rayon qui vient frapper un objet transparent va être dévié en fonction de l'indice de réfraction du matériau de cet objet, mais aussi de celui de celui du matériau d'où arrive le rayon (la plupart du temps l'air, dont l'indice de réfraction est proche de 1).

On détermine l'angle de déviation du rayon transmis lors d'un changement de matériau par la formule

$$\frac{\sin(\theta_i)}{\sin(\theta_t)} = \frac{n_1}{n_2}$$

La valeur du vecteur directeur du rayon transmis est obtenue avec la formule

$$\vec{transmis} = \frac{n_1}{n_2}\vec{source} - \left(\cos(\theta_t) + \frac{n_1}{n_2}(\vec{source} \bullet \vec{normale})\right)\vec{normale}$$



Rayon d'ombre

Pour déterminer si le point d'intersection entre la surface et le rayon contribue directement à la valeur du pixel considéré, on envoie un rayon en direction de chaque source lumineuse, ce qui permet de déterminer si un objet opaque de la scène n'occulte pas cette source.

La contribution des sources de lumière

Si un point n'est pas occulté d'une source de lumière par un objet opaque, cette source de lumière aura une contribution sur la couleur perçue de l'objet. L'illumination finale du point sera la somme d'une composante diffuse et d'une composante spéculaire, et de la contribution de la lumière ambiante.

La composante diffuse d'une illumination représente le fait que l'énergie d'un faisceau lumineux s'étale d'autant plus sur la surface d'un objet si son angle d'incidence est proche de la tangente à l'objet. Pour un objet de couleur S ayant la propriété d'émettre le taux rd de lumière diffuse, éclairé par une source lumineuse de couleur C, on obtient la contribution diffuse suivante :

$$C.\rho_d.S.\left|\vec{normale} \bullet \vec{L}\right|$$

avec L le vecteur de lumière incidente, calculé par normalisation du vecteur allant de la position de la lumière vers le point d'intersection.

La composante spéculaire de l'illumination représente le fait que pour une surface réfléchissante, plus l'œil (la caméra) se trouve en face des rayons réfléchis, plus il recevra de lumière. Pour un objet ayant la propriété d'émettre le taux rs de lumière spéculaire, éclairé par une source lumineuse de couleur C, on obtient la contribution diffuse suivante :

$$C.\rho_s.(\overrightarrow{reflechi} \bullet \vec{L})$$

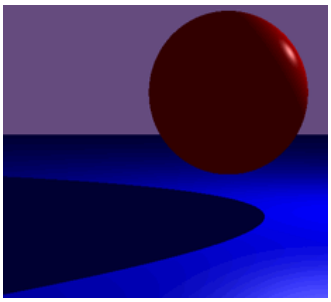
La normalisation de la lumière : On a vu que la l'intensité en un point est la somme de 3 contributions : diffuse, spéculaire et ambiante. Il peut donc arriver qu'une ou plusieurs des composantes de cette intensité dépasse la valeur maximale qu'elle peut prendre (la valeur de l'intensité d'une composante varie entre 0 et 1 : 0 pour éteinte et 1 pour pleine puissance). Il convient donc de normaliser ces valeurs pour qu'elles soient cohérentes. Nous avons choisi de faire du clamping, à savoir que si une composante dépasse la valeur maximale, on la remplace par la valeur maximale. Une autre solution serait de déterminer la composante dont l'intensité est maximale, puis de diviser les valeurs de toutes les composantes par cette valeur maximale ; ceci créerait des images plus sombres mais représentant mieux les dégradés dans les zones intenses de la scène.

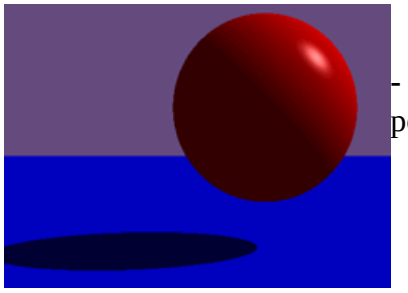
Les sources de lumière

Une scène comporte des objets divers, mais aussi des sources lumineuses qui peuvent être de 3 types différents. On distingue la lumière ambiante, la lumière ponctuelle, et la lumière directionnelle.

- La lumière ambiante : c'est la lumière qui est reçue de façon uniforme par tout point de la scène, indifféremment de la position des autres sources lumineuses et des objets de la scène. Cette lumière ambiante permet d'évider que les zones non éclairées soient complètement noires, ce qui donne un résultat plus réaliste étant donné que tout objet reçoit en réalité une lumière minimale par réflexion d'autres objets.

- La lumière ponctuelle : c'est une source lumineuse placée en un point de l'espace, et qui éclaire uniformément dans toutes les directions. Une lumière ponctuelle a une influence diffuse et spéculaire sur la couleur de l'objet.





- La lumière directionnelle : c'est une source de lumière ponctuelle, mais qui n'émet que dans une direction.