Project COMELEC

« Implémentation d'une architecture matérielle du lancer de rayon »

Réalisé par :

Jefferson Ferreira

Halina Mendoza

Mai 2011

Sommaire

1. Introduction 3

2. Méthode du lancer de rayon 3

3. Principes mathématiques 3

3.1. Calcul des intersections 3

3.1.1. Intersection Rayon – Plan 3

3.1.2. Intersection Rayon – Triangle 4

4. Principes physiques 5

4.1. Réflexion d'un rayon incident 5

4.2. Calcul de la réfraction ou transmission d'un rayon de lumière 5

4.3. Modèle d'illumination 6

5. Algorithmique du lancer de rayon 7

5.1. Considération pour l'implémentation en matériel 7

5.2. Source des images 7

5.3. Division spatiale du contexte 8

5.3.1. Algorithme de Kd-Tree 8

5.3.2. Calculer la meilleure position pour diviser un paquet de triangles 9

5.3.3. Pseudo code de construction de l'arbre kd 10

5.3.4. Algorithme pour trouver les paquets de triangles interceptés pour un rayon 11

5.4. Génération et enregistrement des rayons secondaires 12

5.5. Pseudo code utilisé pour le lancer de rayon 13

5.6. Cadrage 14

6. Architecture proposée 14

6.1. Calcul de la capacité du bus de transfert des données et du numéro des processeurs dédiés à ajouter 14

6.2. Capacité du mémoire « tampon » à ajouter à chaque processeur 15

6.3. Implémentations existantes 16

6.4. Comparaison avec le planning de l'architecture proposée 16

7. Références 17

8. Annexes 18

Annexe I: Nombre des opérations – KD Tree 18

Annexe II : Nombre des opérations – Lancer de rayon 20

1. Introduction

Le lancer de rayon est un algorithme capable de générer des images 3D à partir d'une scène prédéterminé. Il s'agit principalement de générer des rayons de lumière d'une position dans l'écran vers des sources de lumières. Ces rayons partent de chaque pixel jusqu'à qu'ils interceptent un objet. S'il y a des interceptions, le rayon peut se réfracter ou se réfléchir. Tout l'ensemble de rayons contribue pour l'intensité lumineuse du pixel qu'est en train d'être traitée. Puis, le changement à l'escale de couleurs permet de trouver le couleur RGB à ce point là.

2. Méthode du lancer de rayon

La méthode de lancer de rayon consiste en trouver le « vrai » couleur d'un pixel dans une image 3D à travers d'un rayon de lumière arrivant à l'œil dans ce position – là. C'est-à-dire, quand un rayon arrive à l'œil, on retrace le chemin parcouru en arrière. Au moment que le rayon arrive dans une surface quelconque, selon son matériel, le rayon peut se refléter, se réfracter ou se refléter complètement. Ensuite, on calcule la contribution de chaque lumière sur l'intensité lumineuse sur cette position. L'intensité finale est composé pour la composition des intensités de tous les points d'intersection rencontrés dans les trajets de cet ensemble de rayons en considérant aussi les possibles régions des ombres qui peuvent succéder dans la scène.

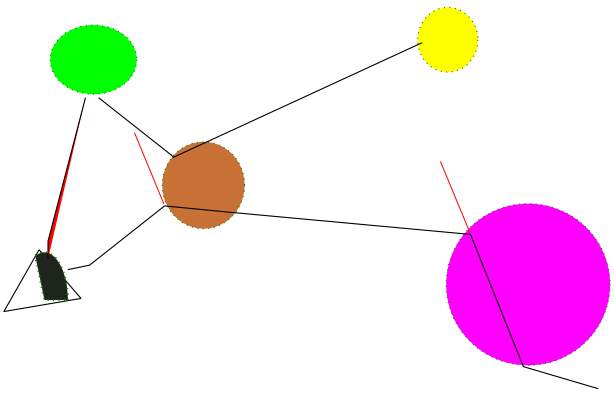

Figure 1: Rayons provenant de l'œil humain. Rayon réfléchis (rouge), Rayon réfractés (noir).

3. Principes mathématiques

3.1. Calcul des intersections

3.1.1. Intersection Rayon – Plan

En 3D une ligne est parallèle à un plan π ou l'intersecte en un point individuel. Soit L donné par l'équation paramétrique: $P(s) = P_0 + s(P_1 - P_0) = P_0 + s \cdot \vec{u}$ et le plan π donné par le point V_0 en son vecteur normal $\vec{n}$. Il faut vérifier si L est parallèle à π en testant si $\vec{n} \cdot \vec{u} = 0$ ce que signifie que la direction du vecteur $\vec{u}$ de la ligne est perpendiculaire au plan normal $\vec{n}$. Si cela est vrai, alors L et π sont parallèles et ils ne s'interceptent pas ou L est totalement dans le plan π. Pour la vérification il suffit de tester s'un point de L (cf. P_0) est contenu en π.

Si la ligne et le plan ne sont pas parallèles alors L et π s'intersectent en un point unique $P(s_I)$ qui est calculé (voir figure 2) de la façon suivante:

Dans le point d'intersection le vecteur $P(s) - V_0 = \vec{w} + s \cdot \vec{u}$, où $\vec{w} = P_o - V_o$, est perpendiculaire à n. Cela est équivalent à la condition: $\vec{n} \cdot (\vec{w} + s \cdot \vec{u}) = 0$. Avec des manipulations on a:

$$s_I = \frac{-\vec{n} \cdot \vec{w}}{\vec{n} \cdot \vec{u}} = \frac{\vec{n} \cdot (V_o - P_o)}{n \cdot (P_1 - P_o)}$$

Si la ligne L est un segment borné de P_0 à P_1, alors il faut vérifier s'il y a une intersection entre le segment et le plan. Pour un rayon positif, il y a une intersection avec le plan quand $s_I >= 0$.

L'algorithme de l'intersection du rayon avec les plans d'une région de l'espace est implémenté de telle façon qu'il calcule tous les possibilités d'intersection avec les plans de la région et vérifie si ces points sont dans la surface de la région et si le rayon sort de la région en utilisant ce point. Si toutes ces contritions sont accomplies ce point aura un statut $valid = true$ et sera utilisé pour trouver le prochain paquet de triangules dans l'arbre de primitives.

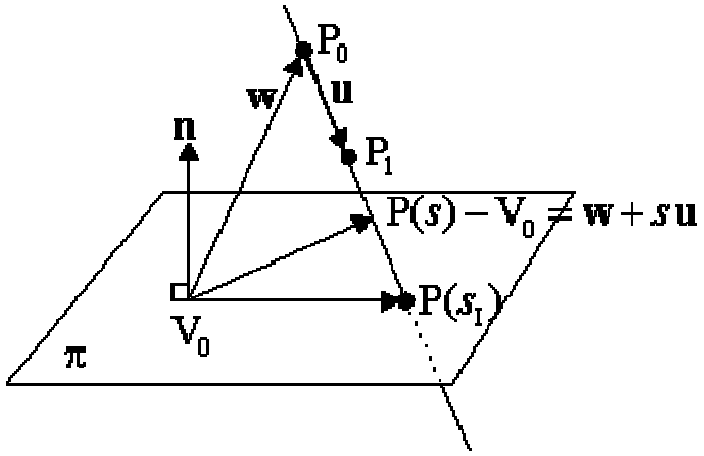

Figure 2: Intersection rayon - plan

3.1.2. Intersection Rayon – Triangle

Dans la bibliographie, il y a plusieurs méthodes pour le calcul de l'intersection entre un rayon et un triangle. La méthode choisie est celle de Moller et Tumbore [3] qui utilise les coordonnés « barycentrales » au but de trouver le point d'intersection.

Un point P d'un triangle est exprimé en coordonnés barycentrales de la façon suivante :

$$P(u, v) = (1 - u - v) * V_0 + u * V_1 + v * V_2$$

U et V sont les coordonnés du barycentre, Vo, V_1, V2, les sommets du triangle. U et V doivent remplir certaines conditions pour les considérer valables :

$$u \geq 0\,, \qquad v \geq 0\,, \qquad u + v \leq 1$$

S'on fait une égalité entre cette représentation et la représentation paramétrique du point, on trouve :

$$O + t * D = (1 - u - v) * V_0 + u * V_1 + v * V_2$$

En l'arrangeant, on peut l'exprimer avec un produit des vecteurs

$$[-D \quad V_1 - V_0 \quad V_2 - V_0] * \begin{bmatrix} t \\ u \\ v \end{bmatrix} = O - V_0$$

Si on nomme $E_1 = V_1 - V_0$, $E_2 = V_2 - V_0$, $T = -V_0$, et les mets dans l'expression précédente, on trouve :

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{((D \times E_2) * E_1)} * \begin{bmatrix} ((T \times E_1) * E_2) \\ ((D \times E_2) * T) \\ ((T \times E_1) * D) \end{bmatrix}$$

Avec la résolution du système précédent, si u et v remplissent les conditions définis, le point appartient au triangle et t est la longueur de son vecteur de direction.

4. Principes physiques

4.1. Réflexion d'un rayon incident

Un rayon réfléchi suit le comportement suivant :

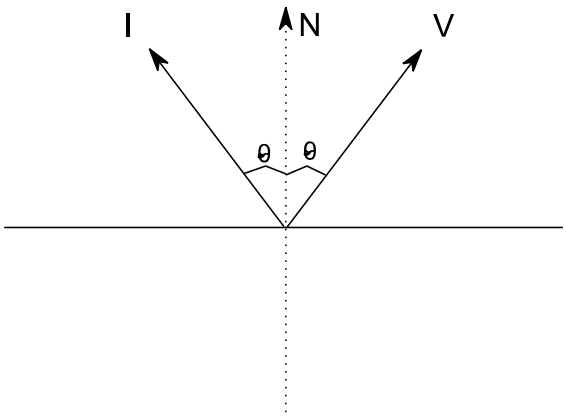

Figure 3: Réflexion d'un rayon incident I

Le rayon I est le rayon incident qui sort de l'objet jusqu'à la position de la camera. Le vecteur V est la direction du rayon réfléchi par le point d'intersection. Avec le vecteur N normal de la surface il est possible trouver la direction du vecteur V :

$$\cos\theta = \frac{I.N}{\|I\|\|N\|}$$

$$perpI_N = I - \left(\frac{I.N}{\|N\|^2}\right)N = I - (I.N)N$$

$$V = I - 2 * perpI_N = 2(I.N)N - I$$

4.2. Calcul de la réfraction ou transmission d'un rayon de lumière

La réfraction d'un rayon a lieu quand le rayon de lumière traverse une surface des différents matériaux.

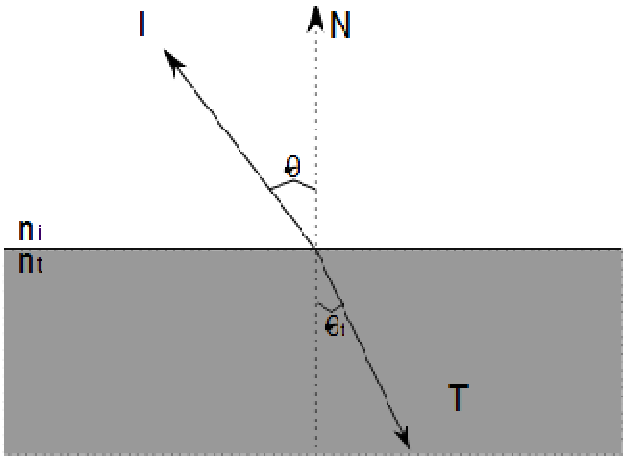

Figure 4: Réfraction d'un rayon incident I

Afin de trouver une relation entre le changement de la direction d'un rayon, à travers des milieux différents, la loi de Snell-Descartes est utilisée pour la réfraction. L'indice de réfraction «n », relation entre la vitesse de la lumière dans le vide et la vitesse dans le milieu actuel, caractérise chaque milieu selon sa capacité de modifier le parcours du rayon entrant. Dans la Figure 4, on a un rayon incident I avec direction Φ_i entre dans un milieu d'indice n_t. La direction du rayon réfracté est donnée par :

$$n_i \sin\theta_i = n_t \sin\theta_t$$

$$T = -N\cos\theta_i - \frac{\sin\theta_t}{\sin\theta_i}[I - (N.I)N]$$

$$T = -N\cos\theta_i - \frac{n_i}{n_t}[I - (N.I)N]$$

$$T = -N\left(\sqrt{1 - \frac{n_i}{n_t}\sin\theta_l}\right) - \frac{n_i}{n_t}[I - (N.I)N]$$

$$\text{Si } \sin^2\theta_l = 1 - \cos^2\theta_l = 1 - (N.I)^2$$

$$T = \left(\frac{n_l}{n_i}.N.I - \sqrt{1 - \left(\frac{n_i}{n_t}(1 - N.I)\right)^2}\right)N - \frac{n_i}{n_t}I$$

N est la normal de la surface normalisé, Φ_t la direction du rayon réfracté et n_i, l'indice de réfraction du milieu d'où le rayon sort.

4.3. Modèle d'illumination

Une fois que les vecteurs de contribution physiques sont calculés, il faut trouver l'intensité lumineuse pour un pixel déterminé.

Afin de le calculer, le modèle d'illumination de Whitted [3] est utilisé. Il fait le rapport entre la contribution de la réflexion spéculaire, transmission, réflexion diffusée et l'illumination globale à l'intensité totale de la lumière dans un point précis.

$$I = I_a + k_d * \sum_{j=1}^{j} (\bar{N}.\bar{L_J}) + k_s * S + k_t * T$$

Où les lettres représentent :

I : Intensité totale

I_a : Intensité de lumière globale

K_d : Coefficient de réflexion diffusé

N : Normal unitaire de la surface

L : Vecteur en direction d'une source de lumière

S : Intensité de la lumière incidente dans de la direction du rayon réfléchi

K_t : Coefficient de transmission

T : Intensité de la lumière dans la direction du rayon réfracté

5. Algorithmique du lancer de rayon

Suite à la description des propriétés du lancer de rayon, cette section présente l'algorithme implémenté. Pour arrêter la boucle de l'ensemble de rayons calculés, un maximum de 5 réflexions par rayon est défini.

L'algorithme consiste en suivre tout rayon partant de l'œil et, une fois qu'il arrive à intercepter un objet de la scène, calculer les rayons générés par cette intersection et la contribution sur l'intensité lumineuse des sources de lumière à ce point. Après l'algorithme fera le même procède pour chaque nouveau rayon jusqu'au moment que les intersections sont finis ou que le nombre maximal de réflexions est atteint.

5.1. Considération pour l'implémentation en matériel

Le but de ce travail est le démarrage d'une architecture en matériel qui implémente le lancer de rayon. En sachant que la gestion des ressources en matériel est différente de celle du logiciel, certaines considérations doivent être prises comme l'usage de l'arithmétique en virgule fixe au lieu de virgule flottante. En autre au lieu d'un algorithme récursif, plus présente dans les implémentations en logiciel, il faut l'implémentation des algorithmes itératifs pour que le lancer de rayon soit synthétisable en matériel.

5.2. Source des images

Il y a deux fichiers responsables pour les paramètres du lancer de rayon. Le premier est un fichier .dsc dont sa structure est la suivante :

- Première ligne : Le mot « DSC »
- Deuxième ligne : la taille de la fenêtre d'observation de la scène – longueur largeur
- Troisième ligne : le nombre de matériaux présents dans la scène et le nombre de lumières

- De la quatrième ligne jusqu'à ligne « numero_materiaux + 3 » : les matériaux dont attributs sont spécifies dans l'ordre suivante : red, green, bleu, coefficient réflexion, coefficient intensité, indice de réfraction
- De la ligne « numero_materiaux + 4 » jusqu'à ligne « numero_lumieres + numero_materiaux +3 » : les lumières dont attributs sont spécifies dans l'ordre suivant : posx, posy, posz, red, green, bleu
- Ligne « numero_lumieres + numero_materiaux +4 » : le nombre de triangles
- Ligne « numero_lumieres + numero_materiaux +5 » jusqu'à la ligne « numero_lumieres + numero_materiaux + numero_triangles + 5 » : le matériel de chaque triangle

Les images à traiter sont formées par triangles et stockés dans un fichier .off dont sa structure est la suivante :

- Première ligne : Le mot « OFF »
- Deuxième ligne : La quantité des points, des triangles et des arêtes. Ce dernier est toujours zéro.
- De la troisième ligne jusqu'à la ligne « numero_points + 3 » : Les points des sommets des triangules
- De la ligne « numero_points + 4 » jusqu'à la ligne « numero_triangles + numero_point + 5» : Le nombre de sommets des polygones et le nombre de lignes où se trouvent les sommets.

5.3. Division spatiale du contexte

5.3.1. Algorithme de Kd-Tree

Les arbres KD sont des arbres binaires dans lesquels chaque nœud contient un point en dimension k. Chaque nœud non terminal divise l'espace en deux demi-espaces. Les références des triangles situés dans chacun des deux demi-espaces sont stockées dans les branches gauche et droite du nœud courant. Par exemple, si un nœud donné divise l'espace selon un plan normal à la direction (Ox), tous les triangles qui ont au moins une partie de sa coordonnée x inférieure à la coordonnée du point de division considéré seront stockés dans la branche gauche du nœud. De manière similaire, les triangles qui ont, au moins une partie de ses coordonnées x supérieures à celle du point considéré seront stockés dans la branche droite du nœud.

Structure de l'arbre

L'arbre kd est composé par nœuds qui ont les paramètres suivants:

Paramètres	Fonction
kdTreeNode * left	Un pointeur au nœud fils de la gauche
kdTreeNode * right	Un pointeur au nœud fils de la droite
kdTreeNode * next	Un pointer pour le prochain nœud d'arbre situé dans le même niveau.
TrigRefList * trig	Contient des références pour les triangles stockés dans un nœud feuille. Si le nœud n'est pas une feuille, cette référence est NULL
Int pnum	Une variable qu'identifie le nombre de triangles stockés dans le nœud
ifixed x0, x1	Un indicateur des plans limitant du nœud dans l'axis x
ifixed y0, y1	Un indicateur des plans limitant du nœud dans l'axis y
ifixed z0, z1	Un indicateur des plans limitant du nœud dans l'axis z
int spx	Un indicateur par rapport quel axis les fils du nœud sont divisés. Les valeurs possibles sont 0 pour l'axis x, 1 pour l'axis y et 2 pour l'axis z

Tableau 1:Nœuds de l'arborescence KD-Tree

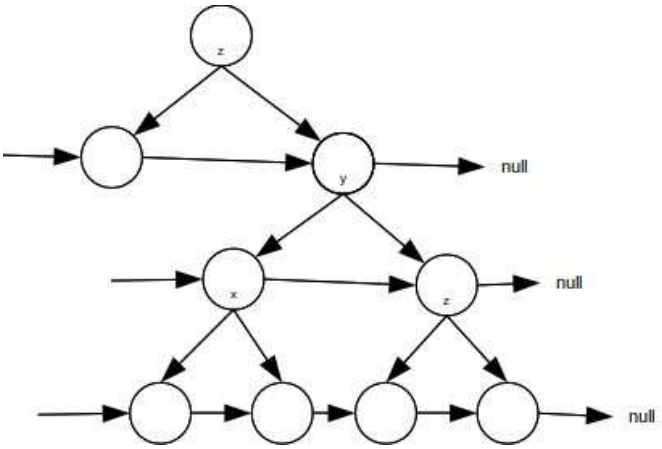

Figure 5: Structure de l'arbre Kd-Tree implémenté

5.3.2. Calculer la meilleure position pour diviser un paquet de triangles

Pour calculer la position dans laquelle un nœud sera divisé par rapport à un axis un modèle de coût [11] est employé. Le modèle de coût correspond à la computation de la fonction de coût pour toutes les positions possibles d'une division par rapport a un axis et le choix de la position qui a le coût minimal. La formule pour calculer le coût est la suivante :

$$C = 1\frac{1}{SA(N_{curr})}[SA(LCN_{curr})(Trig_{left} + Trig_{both}) + SA(RCN_{curr})(Trig_{right} + Trig_{both})]$$

Où:

$SA(N_{curr})$ - area de la surface du nœud courant

$SA(LCN_{curr})$ - area de la surface du candidat à fils de la gauche du nœud courant

$SA(RCN_{curr})$ - area de la surface du candidat à fils de la droite du nœud courant

$Trig_{left}$ - nombre de triangles à gauche de la valeur candidat à possible division

$Trig_{right}$ - nombre de triangles à droite de la valeur candidat à possible division

$Trig_{both}$ - nombre de triangles qu'appartient aux deux cotés de la possible division

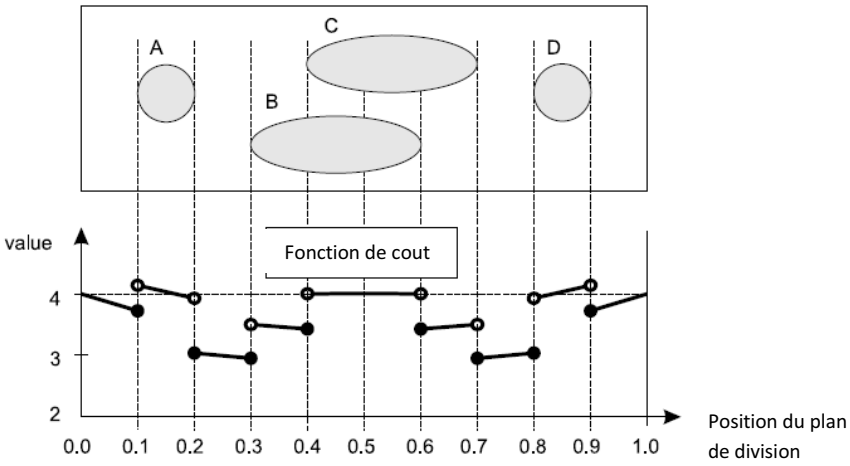

Figure 6: La valeur de la fonction de coût (2 dimensions) pour quatre objets. Les positions limites des objets ont grand influence dans la sélection du point de division avec le coût minimale

5.3.3. Pseudo code de construction de l'arbre KD

```
Paramètres: adresse du premier élément de la liste liée de références de triangles, adresse du premier élément du pool de références de triangles pas encore utilisées,  adresse du premier élément des nœuds de l'arbre KD pas encore utilisés, adresse de la racine de l'arbre KD et profondeur maximale de l'arbre KD.

L'algorithme démarre avec la racine de l'arbre

Faire  // on teste s'il y a de nouveaux nœuds pour être traités (au moins une nouvelle

//division spatiale)

       Faire  // On teste pour tous les nœuds  d'un certain niveau s'ils vont générer des fils

       Si le niveau maximal n'est pas atteint

              L'axis avec le majeur intervalle sera divisé

              Pour tous les sommets des triangles dedans l'intervalle faire

                     Prendre la valeur du sommet

                     Si ce sommet n'existe dans la liste des valeurs candidats pour la division

                            Insérer cette valeur dans la liste

              Pour toutes les valeurs dedans la liste de candidats pour la division

                     Compter le nombre des triangles compris à gauche de la valeur Candidate à division

                     Compter le nombre des triangles compris à droite de la valeur Candidate à division

                     Calculer l'area de la surface de l'espace du nœud courant

              Lowcost = MAXTRIANGLES ; //coût pour partager l'espace du nœud. Pour le démarrage

                                    //la valeur admise est la plus pire possible

              Pour toutes les valeurs dans la liste de candidats à division faire

                     Calculer l'area de la surface des nœuds candidats à gauche et à droite

                     Calculer le cout de la division

                     Si le coût de la division est plus petit que lowcost

                            Lowcost = splittingcost //le nouveau cout calculé
```

```
                                Bestleft = nlefttring   //nombre de triangles à gauche

                                Bestlright = nrighttring //nombre de triangles à droite

                        Si lowcost est moins que le cout de ne diviser pas

                                On divise tous les triangles du nœud courant en deux régions délimitées par la
                                meilleure position de division

                                On crée les deux fils du nœud courant

                                Lier les fils avec des références aux nœuds précédents et suivants

                        Nœud courant= nœud suivant référencié

                Tandis que le nœud courant existe // traite tout les nœuds d'un niveau de l'arbre

                Si le nœud a des fils //il y eu au moins une division

                        Nœud_fils_suivant_reference = NULL

                Noeud_courant = premier_noeud_fils

                premier_noeud_fils = NULL

                noeud_fils = NULL

                Profondeur de l'arbre augment

Tandis que le nœud courant existe    //il y a au moins un nouveau fils pour être traité
```

Pseudo Code 1 : Génération de l'arbre kd

5.3.4. Algorithme pour trouver les paquets de triangles interceptés pour un rayon

Le but de la construction d'un arbre KD est organiser les données pour parcourir un nombre minimale de triangles et vérifier l'occurrence d'intersection de ces triangles avec un rayon. Pour faire cela le pseudo code suivant est implémenté.

Pseudocode de la procède getTrianglePacket

```
Paramètres : rayon (ray), paquet dans lequel on est en train de traiter les triangles (currPack),  pointer pour la racine de la kdTree (tree)

Trouver le point sortant de rayon du currPack

Pour faire cela il faut

        Trouver toutes les points d'intersections des rayons dans le currPack

        Parmi ces intersections, trouver quelles sont dans la surface du currPack

        Finalement, trouver le point d'intersection d'où le rayon sort de la surface du currPack

        Si le point d'intersection se trouve dans les limites de l'espace

                Ne retourner aucun paquet (NULL)

Prendre le point d'intersection afin de l'utiliser pour parcourir l'arbre kd

Retourner le paquet suivant qu'aura les prochains triangles à être traités
```

Pseudo Code 2 : Obtention du prochain paquet de triangles

Pour prendre tous les paquets interceptés par le rayon il faut exécuter getTrianglePacket jusqu'à la sortie de l'espace indexable. Le pseudo code pour faire cela est le suivant

Dans le procède de vérifier s'il y a des intersections :

```
Tandis 1 faire //toujours vrai
        Paquet = getNextTrigPacket( ray, currPack, tree )
        Si le paquet est null
                Sortir de la boucle
        Sinon
                S'il y a une intersection dans le paquet
                        Signaler l'intersection
                        Sortir de la boucle
                Sinon
                        currPack = pack
```

Pseudo Code 3 : Obtention du prochain paquet de triangles

Pour exécuter cet algorithme de localisation de paquets il faut être dans un point dedans une région de l'espace indexable. Au début du procès du lancer de rayon, il y a un rayon sortant du camera qui va intercepter la région du contexte. En effet, étant donné un rayon sortant du camera, il faut trouver le premier paquet que ce rayon intercepte dans le contexte. Pour faire cela le code pseudo code suivant est implémenté :

```
Trouver les intersections du rayon avec tous les plans limites de la scène
Trouver les intersections valables (dedans la surface du cube de la scène)
Entre les deux intersections valables choisir la plus proche du camera
Parcourir l'arbre et trouver le premier paquet
```

Pseudo Code 4 : Obtention du premier paquet d'un rayon dans l'espace indexable

5.4. Génération et enregistrement des rayons secondaires

A cause de la récursivité originale de l'algorithme du lancer de rayon, il a fallu créer une arborescence qui héberge tous les rayons secondaires crées par le rayon initiale du camera.

La racine de l'arborescence est le rayon principal. Ensuite, on va calculer le rayon réfléchi et l'enregistrer dans le fils gauche. On continue le calcul jusqu'à arriver à la profondeur maximale établie. Sinon, on s'arrête quand le rayon n'intersecté aucune surface

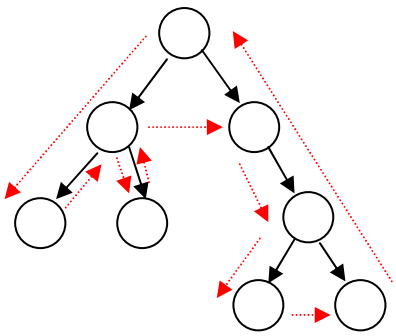

Figure 7: Parcours de l'arbre (ligne rouge)

Après, on demande au père s'il peut avoir un rayon réfracté. Si c'est possible, le rayon réfracté est crée dans le nœud du fils droit.

Puis, on vérifie s'il peut avoir des fils. S'il n'en peut pas, on revient au père et on pose la question au fils droit. Une fois que la vérification est faite, on continue à parcourir l'arborescence jusqu'au moment de retour à la racine. Afin de préciser les rayons secondaires, tout rayon réfléchi peut avoir deux fils. Par contre, un rayon réfracté peut avoir un rayon réfléchi et un fils droit quand le rayon sort d'un objet à l'extérieur et il n'ya pas réflexion totale. Le parcours est défini dans la figure 7.

La table suivante montre les éléments d'un nœud :

Paramètre	Fonction
Parent	Un pointeur au nœud père
Left	Un pointeur au nœud fils de la gauche
Right	Un pointeur au nœud fils de la droite
Visited	Un indicateur de passage temporel dans le nœud. Il est notamment utilisé dans le calcul de la contribution de l'illumination des rayons.
Type	Un indicateur du type de rayon : rayon du camera, rayon réfléchi, rayon réfracté et rayon avec réflexion totale.
Level	Un indicateur du niveau de l'arborescence
State	Un indicateur de l'emplacement des rayons ; soit dans l'espace vide ou soit dans l'intérieur d'un corps.
Ray	Soit le rayon du camera, rayon réfléchi ou rayon réfracté.

Tableau 2: Eléments d'un nœud dans l'arbre de gestion des rayons

5.5. Pseudo code utilisé pour le lancer de rayon

Voici dans les lignes suivantes le pseudo code utilisé pour le lancer de rayon :

```
Processus lancer_du_rayon (pointeur image, scenario  contexte)

Pour chaque pixel de l'écran,

        Générer un rayon qui partira du pixel actuel avec une direction perpendiculaire à l'écran.

        Appeler la fonction raytracing avec le rayon r et le contexte comme des arguments

        Trouver le couleur du pixel actuel

        Passer au  pixel suivant

Pixel suivant

Fin processus


Couleur Fonction raytracing (rayon r, scene s)

Initialisation du vecteur de couleur « C1»

Création d'un vecteur des rayons

Création d'un pointeur du type nœud
```

```
Initialisation de la liste des nœuds

Démarrer un rayon de la liste

Si le rayon demandé n'est pas nul.

        Initialiser le nœud

                Assignation d'un pointeur « courant » à la racine

                Tandis que l'on ne revient pas à la racine

                        C1= C1 + Contribution_rayon (rayon, contexte)

                        Courant= GenerationArbre (&liste, profondeur maximale, courant, contexte)

Retourner le couleur du pixel

Fin fonction
```

Pseudo Code 5 : Lancer de rayon

5.6. Cadrage

Étant donné que l'on travaille avec des numéros en virgule fixe, il y a eu des problèmes de précision, lesquels ont influencé lors de l'enregistrement de l'information du scenario. Pour gérer la situation, on a utilisé le cadrage; une opération que permet de changer le numéro de bits entiers et rationnel selon les besoins.

On a du utiliser les cadrages pour l'enregistrement et traitement des données, enregistrer les valeurs d'luminosité, l'exactitude de la racine carré et la séparation des régions pour l'arbre Kd. Finalement, dans l'implémentation, on a utilisé le type de donnée de point fixe du SystemC, lequel offre des fonctions de cadrage explicites.

6. Architecture proposée

Afin d'implémenter le lancer de rayon en matériel, on propose l'architecture de la fig. 8 qui montre les bloques de la maquette à utiliser. Le processeur est responsable pour calculer l'intersection d'un rayon venant du camera avec l'espace indexé et pour faire le management de chacun des coprocesseurs (fig. 9) en les fournissant un rayon (utilisé pour calculer l'intersection) et l'adresse du premier paquet de triangles. Les coprocesseurs sont responsables pour le raytracing. Pour avoir une idée de la capacité des dispositifs de traitement, on a fait des approximations selon le nombre d'opérations et une comparaison avec les implémentations existantes.

6.1. Calcul de la capacité du bus de transfert des données et du numéro des processeurs dédiés à ajouter

La capacité ainsi que la quantité maximale des processeurs fonctionnant en même en temps dépend de la relation entre le temps de calcul de la couleur d'un pixel et les accès de mémoire pour le faire. La méthodologie appliquée est de calculer le nombre des opérations à réaliser pour un numéro fixe des paquets de triangles.

D'abord, on a essayé de trouver des limites maximales, en nombre de triangles, dans les implémentations du lancer de rayon courantes ou un modèle pouvant caractériser un scenario quelconque. D'après [11], la quantité maximale des intersections pour un arbre kd de profondeur 16 est 40 intersections. En suite, on a choisi de fixer à 20 le numéro de paquets à traiter (quantité moyenne d'intersections). Etant donnée ses conditions, la table montre le numéro d'opérations à réaliser et les accès mémoire à faire.

Sachant qu'un accès à la mémoire, soit une lecture ou une écriture, prends deux fois plus de temps qu'une opération élémentaire du processeur, on peut trouver la quantité maximale des processeurs fonctionnant en forme parallèle (Voir Annexe I et II):

$$\#processeurs = \frac{\#Operacions\ processeur}{\#Accès\ memoire} = \frac{203000}{49600} = 4.09 \cong 4$$

S'on travaille avec 4 processeurs (f_{max}=100MHz), on pourrait avoir un pixel prêt en 0,6315 ms (1.583 KHz) et une image de 640 x 480 pixels en 194s.

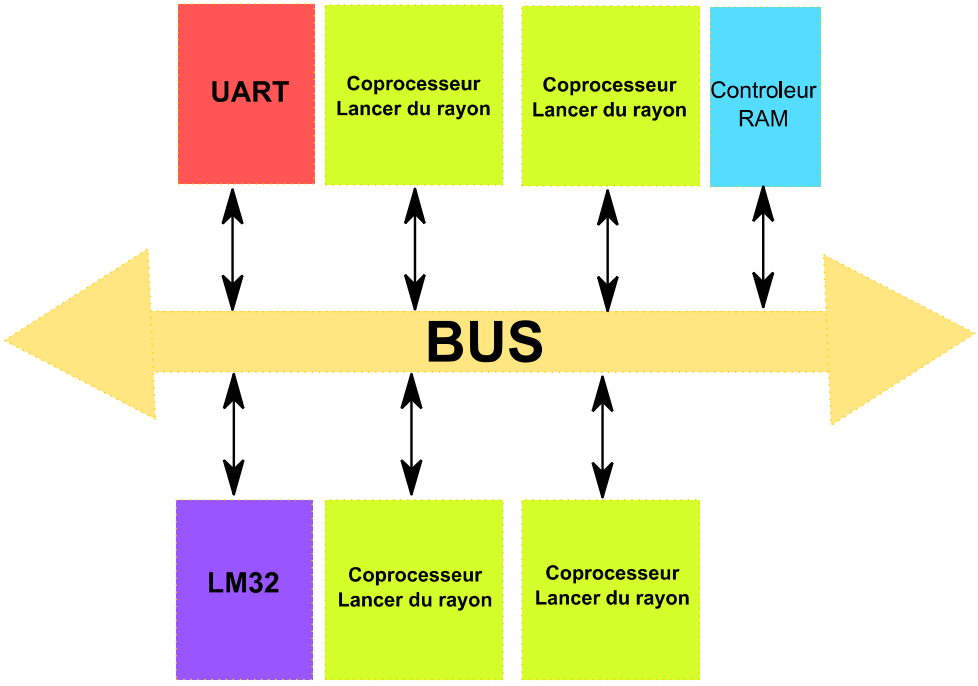

Figure 8: Schéma de l'architecture proposé avec quatre processeurs

6.2. Capacité du mémoire « tampon » à ajouter à chaque processeur

Pour une meilleure gestion du trafic dans les accès à la mémoire, on a décide d'ajouter un mémoire tampon qui va à stocker les triangles de chaque paquet et leur caractéristiques. Parmi les données, on trouve :

- Deux nœuds de l'arbre kd qui représentent 110 octets
- Les adresses du paquet courant et le couleur du pixel : 8 octets
- Un vecteur de 512 triangles : 15872 octets
- Une structure pour le triangle plus proche : 31 octets
- Un arbre binaire pour les rayons secondaire : 1736 octets
- Deux vecteurs pour les matérielles et les sources des lumières : 768 octets

En total la mémoire tampon doit avoir au minimum 18,09 Koctets.

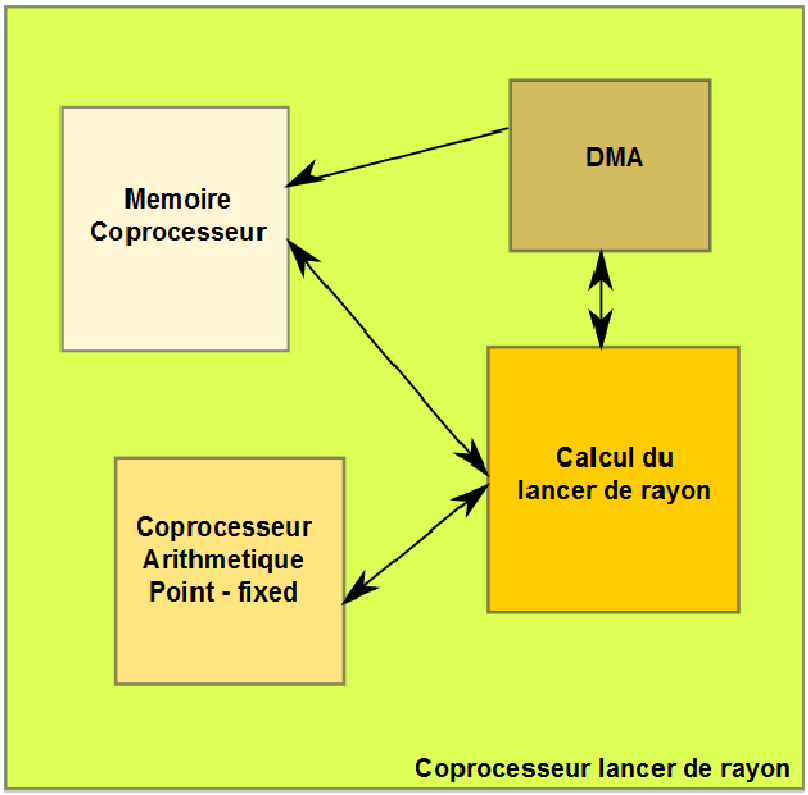

Figure 9: Schéma du coprocesseur du Lancer de rayon

6.3. Implémentations existantes

Au but de comparer l'architecture que l'on proposé, on a étudié les implémentations existantes du lancer de rayon.

Jusqu'à maintenant, il n'y a pas dans le marché des IPs dédiés au calcul de lancer de rayon malgré les prototypes existants depuis le débout des années 2000. Par contre, SiliconArts va commercialiser un ASIC qui fournira des fonctions propres au lancer de rayon, avec une mémoire interne de 512KB par noyau. Aussi, il serait possible de traiter dès 5M à 20M de rayons par second dans chaque noyau [13].

Intel travaille aussi sur les ASICs du lancer de rayon mais avec le but de les exploiter dans les jeux. En fait, ils ont réussi à faire le lancer de rayon en temps réel dans un ordinateur portable [14]. Ils ont du, quand même, utiliser une «cloud» de quatre serveurs, chaque un contenant son nouveau chip Knight Ferris, pour y parvenir. Le scenario traité avait près de 1 million de triangles, une fréquence de 83 trames par seconde et une résolution de 1280 x 720 pixels. Ainsi, l'option d'Intel permet de calculer un pixel en 0.013 µs, contraint de temps accorde avec la notion de «temps réel»

6.4. Comparaison avec le planning de l'architecture proposée

Après l'exam des solutions matérielles existantes pour le lancer de rayon, il est possible percevoir que paralléliser les procèdes a un rôle clé pour l'optimisation du temps de calcule.

Dans le cadre d'optimisation de l'architecture, il faudra redresser l'architecture vers un modèle reparti, étant donnée la quantité des rayons, objets à couper, les effets d'illumination, entre autres. Pour cela, il faut approfondir dans la parallélisassions du calcul. Le temps de traitement d'un pixel par des futures applications commerciales est

très inférieur à celui qu'on a obtenu. En autre, il faut vérifier si le procès de calcul d'une valeur de pixel serait mieux s'il y avait les interpolations des pixels au lieu de calculer chacun au fur et mesure.

Critères	Architecture Intel	Architecture proposé
Nombre de serveurs parallèles	4	0
Résolution	1280 x 720	640 x 480
Primitive de test	Triangles	Triangles
Temps de calcul d'un pixel	0.013µs	0.6315ms

Tableau 3: Comparaison entre l'architecture d'Intel et l'architecture proposée

7. Références

[1] Glassner, Andrew, "An Introduction to Ray Tracing". Academic Press. 1991. 3eme Ed.

[2] Suffern, Kevin. "Ray Tracing up from the Ground". CRC Press 2007.

[3] Moller, Tomas et Ben Trumbore, "Fast, Minimum Storage Ray / Triangle". 1997 http://www.acm.org/jgt/papers/MollerTrumbore97/

[4] Lengyel, Eric. "Mathematics for 3D Game Programming & Computer Graphics" CRM Inc. 2004. Ed 2eme.

[5] Whitted, Turner. "An improved Illumination Model for Shaded Display". CACM, 1980, pp 343-349

[6] Shirley, Peter et Keith Morlet. "Realistic Ray – Tracing". , 2003, Ed. 2nd.

[7] Shevtsov, Soupikov, and Kapustin. "Highly parallel fast kd-tree construction for interactive ray tracing of dynamic scenes." Computer Graphics Forum, 26(3):395–404, 2007

[8] Woop S., Marmitt G., Slusallek P. "B-KD Trees for Hardware Accelerated Ray Tracing of Dynamic Scenes". In Proceedings of Graphics Hardware (2006).

[9] Kun Zhou, Qiming Hou, Rui Wang, Baining Guo. "Real-Time KD-Tree Construction on Graphics Hardware" ACM Transactions on Graphics (SIGGRAPH Asia 2008);

[10] Foley, Tim. Sugerman, Jeremy "KD-tree acceleration structures for a GPU raytracer". Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware, July 30-31, 2005, Los Angeles, California.

[11] Havran,V. "Heuristic Ray Shooting Algorithms" Ph.D. thesis, Czech Technical University, November 2000;

[12] Keller, A. Slomka, F. "Fixed Point Hardware Ray Tracing". Universittat Ulm, 2007.

[13] SiliconArts : http:// www.siliconarts.com

[14] Pohl, D. "Ray traced games in the Cloud using MIC". Feb 23, 2011. http://blogs.intel.com/research/2011/02/ray_traced_games_in_the_cloud.php

[15] Mancini S. "Architectures matérielles pour la synthèse d'image par lancer de rayon" Thèse de doctorat, Ecole Nationale Supérieure des Télécommunications, Janvier 2000

8. Annexes

Annexe I: Réquisits de mémoire et nombre des opérations – KD Tree

Structure	Elément	Axis	Total
Un triangle de l'arbre kd	v1	16	48
	v2	16	48
	v3	16	48
	small	16	48
	big	16	48
	materialId	-	8
	Total (bits)	-	248
	Total (octets)		31
	Total triangles MB		1,773834MB
Structure de l'arbre kd	left*		32
	right*		32
	next1*		32
	trig*		32
	pnum*		16
	x0	16	48
	x1	16	48
	y0	16	48
	y1	16	48
	z0	16	48
	z1	16	48
	split		8
	Total		440
	Total en octets		55
	Total KD-Tree (Profondeur 16) MB		3,4375
Référence de l'arbre kd	triangle*		32
	next*		32
	total		64
	total en octets		8
	total références	3	180000
	total octets références (MB)		1,373291

Operations						
Opération	**Accès Mémoire**	**Comparaison**	**Addition**	**Soustraction**	**Multiplication**	**Division**
Intersection rayon – nœud	1	180	50	18	42	6
Traverser arbre kd	16	16	-	-	-	-

Obtenir le premier paquet des triangles	17	234	56	28	48	6
Obtenir un paquet quelconque des triangles	17	203	50	18	42	6

Annexe II : Nombre des opérations – Lancer de rayon

Fonction	Sous-opérations	#Operations	#Occurrences	Total
Racine Carré	Addition			96
	Soustraction			132
	Multiplication			32
	Comparaison			32
	Total			292
Intersection avec un triangle	Addition	1	9	9
	Soustraction	1	15	15
	Multiplication	1	27	27
	Division	1	3	3
	Comparaison	1	7	7
	Total			61
Intersection total	Addition	1	40	40
	Comparaison	1	81	81
	Intersection avec un Triangle	61	40	2440
	Total			2561
Calcul d'un vecteur normal d'un Triangle	Soustraction vecteur	3	2	6
	Soustraction	1	3	3
	Multiplication	1	6	6
	Division	1	1	1
	Racine Carré	292	1	292
	Total			308
Rayon se réfléchit sur la surface ?	Comparaison	1	2	2
	Intersection totale	2561	1	2561
	Total			2563
Rayon se réfracte sur la surface ?	Comparaison	1	2	2
	Intersection totale	2561	1	2561
	Total			2563
Créer rayon réfléchi	Addition vecteurs	3	1	3
	Soustraction vecteurs	3	3	9
	Produit Scalaire	5	1	5
	Produit Scalaire x Vecteur	3	2	6
	Soustraction	1	3	3
	Multiplication	1	7	7
	Division	1	1	1
	Comparaison	1	2	2
	Normale Triangle	308	1	308
	Total			344
Créer rayon réfracté	Addition vecteurs	3	1	3
	Soustraction vecteurs	3	1	3
	Produit Scalaire	5	5	25

	Produit Scalaire x Vecteur	3	4	12
	Addition	1	4	4
	Soustraction	1	2	2
	Multiplication	1	3	3
	Division	1	1	1
	Comparaison	1	9	9
	Racine Carré	292	1	292
	Normale Triangle	308	1	308
	Total			662
Génération de l'arbre des rayons secondaires	Comparaison	1	10	10
	Rayon Réfléchi	2563	1	2563
	Rayon Réfracté	2563	1	2563
	Créer rayon	662	1	662
	Total			5798
Calcul de l'intensité lumineuse	Addition	1	104	104
	Soustraction	1	96	96
	Multiplication	1	69	69
	Division	1	1	1
	Comparaison	1	119	119
	Racine Carré	292	1	292
	Normale Triangle	308	1	308
	Intersection des Triangles	61	40	2440
	Total			3429
Lancer de rayon	Comparaison	1	28	28
	LC	3429	22	75438
	Génération des rayons secondaires	5797	22	127534
	Total			203000

Accès de mémoire : 2 clocks * 20 paquets * 40 Triangles * 31 = 49600

Où 20 paquets sont obtenus de la quantité moyenne d'intersections par rayon

40 triangles est la quantité moyenne de triangles par paquet