Master 2 Informatique : Projet 150h

Scalable Ambient Obscurance

Enseignant tuteur :
Flavien Bridault - Ingénieur de recherche - Ircad

Objectif :
Implémenter la solution proposée par les auteurs de l'article "*Scalable Ambient Obscurance*" dans le framework *fw4spl* en utilisant *Ogre* comme moteur de rendu

OUHMICH Farid - 7 décembre 2015

Remerciement

Avant tout développement sur cette expérience, il apparaît opportun de commencer ce rapport par des remerciements.
Aussi, je remercie Flavien BRIDAULT (Ingénieur de recherche à l'Ircad), mon tuteur qui m'a renseigné et aidé tout au long de cette expérience enrichissante avec beaucoup de patience et de pédagogie.

Table des matières

Introduction

Pour commencer nous expliciterons les principes de base de l'occlusion ambiante ainsi que son utilité pour améliorer les techniques de rendu. Nous poursuivrons en détaillant les différentes étapes qui compose cette version particulière nommée *Scalable Ambient Obscurance* ainsi que les spécificités qui la différencie d'autres méthodes de calcul d'occlusion. Puis nous évoquerons le contexte dans lequel on se situe avant de revenir sur les résultats obtenus suite à cette implémentation.

Occlusion Ambiante

Le principe de base de l'occlusion ambiante est que l'**ombrage** en un point est dépendant de la géométrie voisine, c'est pourquoi l'intérieur d'un tube ne sera jamais très éclairé par exemple, et ceci indépendamment du nombre ou du type de lumières présentes à l'extérieur du tube.

Philosophie Classique :

En calculant pour un sommet de la scène la proportion de l'environnement extérieur que celui-ci peut voir moins la proportion de l'environnement qui est caché par d'autres parties du modèle, on peut déterminer son **accessibilité**.

Calcul de l'accessibilité :

On commence par associer à chaque sommet de la scène un disque comme ci-dessous

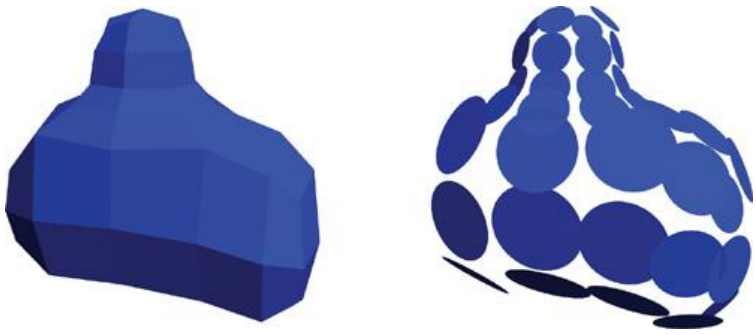


FIGURE 1 – Chaque cercle a comme centre la postion du sommet, comme orientation la normale de celui-ci et comme aire l'application de la formule de Héron

L'accessibilité en un point peut être vu comme la différence entre 1 (accessibilité maximale) et la proportion d'ombrage apportée par les autres sommets (valeur qu'on peut qualifier de **contribution** à l'occlusion)

Contribution :

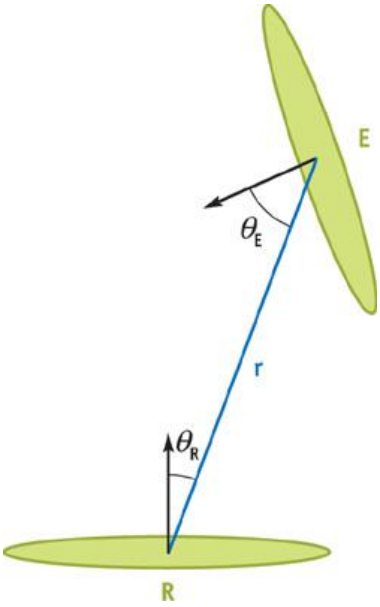


FIGURE 2 – E = cercle emetteur, r = recepteur

Pour chaque sommet de la scène on peut déterminer cette contribution avec la formule suivante :

$$contribution(recepeteur, emetteur) = \frac{r * \cos\theta_e * max\,(1, 4\cos\theta_r)}{\sqrt{\frac{A}{\pi} + r^2}}$$

On a alors l'accessibilité en un point qui se calcule très facilement :

$$accessibilite(p) = 1 - \sum^{emetteur\ e} contribution(p, e)$$

Cette valeur est maximale lorsqu'aucun autre objet ne se trouve dans l'hémisphère issu de la normale et diminuera plus les objets de cet hémisphère sont proche et plus leur contribution est importante.

Le calcul de l'occlusion ambiante peut s'avérer utile si l'on souhaite effectuer d'autres opérations comme illuminer la scène par exemple.

Extensions

Pour effectuer un éclairage de la scène basé sur le calcul de l'occlusion précedemment effectué, on peut combiner l'utilisation de l'éclairage ambiant comme source de lumière et une map (*environment map*) pour déterminer la couleur de chaque point.

Pour cela, en plus de récupérer la proportion de la scène visible depuis un point, on sauvegarde également la direction à partir de laquelle on recoit le plus de lumière, cette direction est appelée ***bent-normal***.

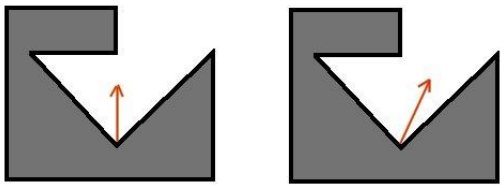


FIGURE 3 – à gauche la normale classique, à droite la *bent-normal*

La couleur de la lumière incidente en P sera alors determinée en moyennant la lumière incidente provenant du cône créé à partir de la *bent-normal*.

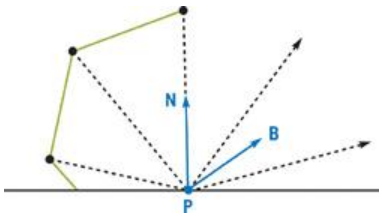


FIGURE 4 – on construit le cône à partir de la bent-normale B

Le résultat est que les fissures apparaissent plus sombres (comme elles devraient l'être en réalité) alors que les parties exposées recoivent plus de lumière et sont par conséquent plus brillantes.

On obtient des scènes beaucoup plus réalistes que si un modèle standard de calcul d'éclairage était utilisé.

Interessons nous maintenant à une technique plus récente et plus complète permettant de calculer l'occlusion ambiante, qui est la **Scalable Ambient Obscurance**.

Scalable Ambient Obscurance

On peut appeler cette technique SAO (*Scalable Ambient Obscurance*) ou encore de SSAO (*Screen-Space Ambient Obscurance*), l'idée de base est que le calcul de l'occlusion se fait indépendamment de la densité des pixels.

Cette technique prendra comme entrées le buffer de profondeur (*Z-Buffer*), le rayon d'échantillonage souhaité r ainsi que le nombre d'échantillons s pour rendre en sortie la visibilité ambiante en chaque point (stockée dans une texture qu'on nommera *Occlusion Texture Map*).
Elle est composée de 4 étapes que nous détaillerons les unes après les autres.

Calcul de profondeur

Comme évoqué précedemment, l'une des seules entrées nécessaires à ce calcul est le Z-buffer, qui est simplement obtenu en transformant une coordonée de l'espace en coordonnée écran, et en récupérant la composante z de cette position.

On pourra par exemple remplir un *Z-buffer* de la manière suivante :

```
\\ Fragment Shader
out float depth;
void main()
{
  depth = gl_FragCoord.z;
}
```

Les GPUs modernes sont hautement optimisés pour faire cette opération (la position *gl_ FragCoord* est automatiquement calculée), cependant les auteurs de l'article de base (ajout ref) se sont rendus compte que les opérations arithmétiques du Pipeline introduisaient des erreurs et qu'en les réduisant, on pouvait pousser la précision plus loin.
Pour cela ils proposent plusieurs solutions :

— Effectuer le calcul de la matrice de "Transformation" avec une double précision (cela comprendra la multiplication Model * View * Projec-

tion, les divisions et les opérations trigonométriques)
— Utiliser $zf = -\infty$ dans la matrice de Projection
— Multiplier les vecteurs à gauche $\vec{v}\,' = \vec{v}\,\mathbf{P}$ (contrairement à la convention OpenGl)

Génération de Mip-Map

On va chercher ici à convertir la valeur du buffer de profondeur (valeur *depth* comprise entre 0 et 1) en une profondeur "caméra" (valeur z inférieure à 0). On souhaite effectuer cette génération à plusieurs niveaux de précision (création d'une *Mip-Map*)

Pour convertir la valeur *depth* précedemment calculée en profondeur "caméra", on utilise la formule suivante :

$$z(depth) = \frac{c[0]}{depth * c[1] + c[2]} \text{ avec}$$

$$\begin{cases} c = [zn; -1; +1] & \text{lorsque } zf = -\infty \\ c = [zn * zf; zn - zf; zf] & \text{sinon} \end{cases}$$

On utilisera la formule précedemment citée pour générer le premier niveau de Mip (*niveau 0*) puis on génèrera les niveaux suivants en effectuant une interpolation comme celle-ci :

$$z^{m+1}(x, y) = z^m(2x + (y\&1 \oplus 1), 2y + (x\&1 \oplus 1))$$

& et $\oplus$ correspondent aux opérations *and* et *xor* sur les bits

Échantillonage

Cette passe distribue S échantillons dans une demi-sphère issue du point C (point vers lequel est dirigée la caméra).

Pour cela on commence par déterminer la **position du point C** dans l'espace [1] à partir de sa position (x_c, y_c) sur l'écran [2] :

— On commence par définir la profondeur de ce point grâce au niveau 0 de Mip calculé plus tôt :
$$z_c = z^0(x, y)$$
— Puis on récupère la position du point en utilisant la matrice de Transformation notée $\mathbf{T}$:

1. Camera-Space position
2. Screen-Space position

$$(x_c, y_c) = z_c * \left(\frac{1 - T_{0,2}}{T_{0,0}} - \frac{2(x + \frac{1}{2})}{w * T_{0,0}}, \frac{1 + T_{1,2}}{T_{1,1}} - \frac{-2(y + \frac{1}{2})}{h * T_{1,1}} \right) \quad (1)$$

w et h correspondent aux dimensions de l'image

On détermine ensuite la **normale** de la face contenant C en utilisant les gradients spatiaux :

$$\hat{n_c} = normalize \left(\frac{\delta C}{\delta y} \times \frac{\delta C}{\delta x} \right)$$

On calcule ensuite le **rayon** r' (relativement à l'écran) de la sphère dans laquelle s'effectuera l'échantillonage :

$$r' = -r * \frac{S}{z_c}$$

avec S étant la taille en pixel
d'un objet de 1m à une prondeur 1m

À partir de là on cherche à placer nos s échantillons en suivant une spirale issue du point C qui couvre le rayon r'.
Pour chaque **échantillon** q_i on peut déterminer sa **position** (x_i, y_i) sur l'écran avec la formule suivante :

$$(x_i, y_i) = (x_c, y_c) + h_i * \hat{u_i}$$

$$\text{avec : } \begin{cases} \alpha_i = \frac{1}{s}(i + 0.5) \\ h_i = r' \ \alpha_i \\ \theta_i = 2 \ \Pi \ \alpha_i \ \tau + \phi \\ \hat{u_i} = (\cos \theta_i \ , \ \sin \theta_i) \\ \phi = 30 \ x_c \ \wedge \ y_c + 10 \ x_c \ y_c \end{cases}$$

Le point (x_i, y_i) est donc issu d'une translation d'un vecteur $h_i * \hat{u_i}$ à partir du point C .

On détermine ensuite **le niveau de Mip** m_i où on ira lire la profondeur de ce point échantillon :

$$m_i = \lfloor \log_2 (h_i / q) \rfloor$$

avec q étant le "pas" d'augmentation de rayon pour lequel on change de niveau de Mip (les auteurs conseillent d'utiliser $2^3 \leq q \leq 2^5$ + ref bib)

On utilise ensuite notre pyramide Mip-Map pour déterminer la profondeur z_i de notre échantillon et on réutilise l'équation (1) pour trouver la position spatiale de notre point échantillon q_i.

On est alors apte à déterminer **le facteur d'occlusion** A de notre point C en sommant la contribution nos s points échantillons :

$$A(C) = max\left(0, 1 - \frac{2\,\sigma}{s} * \sum_{i=1}^{s} \frac{max(0, \vec{v_i} \cdot \hat{n_c} + z_c\,\beta)}{\vec{v_i} \cdot \vec{v_i} + \epsilon}\right)^k$$

les variables σ, β et k sont choisies "esthétiquement" et ϵ est la plus petite valeur possible pour ne pas avoir de division par 0
On a pour chaque échantillon $\vec{v_i} = q_i - C$

Cette étape se conclu en appliquant un **filtre bilatéral** 2×2 de reconstruction pour moyenner les valeurs de A et ainsi réduire la variance (relativement au 4-voisinage) .

Reconstruction bilatérale

On se retrouve à ce niveau là avec un nuage de points présent là où l'occlusion est engendrée, on veut obtenir un résultat plus lisse, pour cela on utilise un flou gaussien. On décide donc d'appliquer 2 filtres 1-D composés de poids qu'on modulera à partir de la différence de profondeur.

On peut se permettre (pour diminuer le nombre de calculs) d'espacer les différentes origines des filtres de 3 pixels chacun (grâce au filtre bilatéral de l'étape précédente).

On se retrouve alors avec une carte qui nous donnera pour chaque pixel la valeur de son facteur d'occlusion A compris entre 0 et 1.

Différences par rapport à d'autres méthodes

Le but de la méthode explicitée ici, était d'améliorer le rendement de ce qui était alors considéré comme l'état de l'art en terme de calcul d'occlusion, à savoir la méthode nommée *AAO : Alchemy Ambient Obscurance*. [ref à ajouter]

Le modèle mathématique reste le même que pour cette méthode, ce qui change c'est l'implémentation. Contrairement à *AAO* qui nécessitait de prendre comme entrée les buffer de positions et de normales, notre méthode ne part que du buffer de profondeur.

La principale limitation connue pour les anciennes méthodes de calcul d'occlusion ambiante est que les performances chutaient énormément lorsqu'on récupérait des échantillons loin d'un pixel :
Pour une résolution assez faible (*720p*) *AAO* ne peut échantillonner qu'avec un cercle de **0.1m** autour de chaque pixel, et ceci en **2.3 ms** : On se limitait alors à des effets très locaux.
Dans le même temps, avec *SAO*, même avec une résolution plus elevée (*1080p*) on réussi à échantillonner dans un rayon de **1.5m**.

Contexte d'implémentation

Cette technique de calcul de l'occlusion ambiante a été intégrée à **OgreViewer**, une application créée à partir du framework **Fw4Spl**.

Fw4spl

Le framework *Fw4spl* (appelé *Forces*) a été developpé par l'Ircad pour permettre la création rapide d'applications, principalement dans le domaine médical. La philosophie de ce framework multi-plateforme est basée sur le concept *objet-service* et sur la communication *signaux-slot* que l'on peut retrouvé dans des langages comme *Qt*.

OgreViewer

L'application qui nous intéresse ici se nomme *OgreViewer* et permet d'observer (en $3D$) le corps de patients avec (entre autres) les paramètres suivants :

— la manière dont sera représentée chacun des organes (textures, profondeur ...)
— quels organes seront affichés
— la possibilité d'appliquer un filtre plein-écran

Cette application utilise *Ogre* comme moteur de rendu.

Ogre

Ce moteur de rendu se repose essentiellement sur le concept classique d'une scène hierarchique (notion d'héritage, noeud *Root*, ...)
Une des applications intéressantes offertes par ce moteur est l'utilisation de **compositor**. Il s'agit d'un script permettant de définir un effet "plein-écran" que l'on peut appliquer à un viewport. Pour définir nos compositors, on utilisera le schéma suivant :

1. Récupérer la scène de départ en la stockant dans une texture

2. Effectuer des opérations sur cette texture en l'interprétant comme un quad plein-écran
3. Afficher le résultat final en plein-écran ou bien le stocker dans une autre texture si l'on souhaite par exemple chainer des compositors

Détails d'implémentation

Nous avons choisi d'utiliser un **premier compositor** pour générer les différents niveaux de Mip, nommé *MipMap*.

La partie concernant les erreurs relatives aux opérations arithmétiques du pipeline n'ont pas étés prises en compte, on a préfèré directement récupérer la matrice de transformation calculée par Ogre.

```
\\ parametre d'appel du Vertex Shader dans le compositor MipMap
param_named_auto u_worldViewProj worldviewproj_matrix
```

```
\\ Vertex Shader
\\ "uniform" indique qu'il s'agit d'une variable calculee par ogre
uniform mat4 worldViewProj;
```

Le premier niveau de Mip (niveau 0) est directement calculé à partir de la scène originale, tandis que les autres niveaux seront determinés récursivement, comme évoquer plus haut.
La création de la pyramide nécessite d'ajouter une interface en *C++* qui puisse déterminer à quel moment le **second compositor** nommé *AO_samples* (qui effectuera l'échantillonage) sera appelé :

```
compChain->getCompositor("AO_Samples")->addListener(new SaoListener(↩
    m_ogreViewport));
```

Ajout illustration appel à notifyMaterialRender()

La texture contenant la pyramide est alors générée en copiant successivement les différents niveaux dans la cible :

```
\\ copie de la premiere texture dans le premier niveau de la ↩
    pyramide
\\ rt0 est notre texture en pyramide
\\ mip0 est une des sorties obtenues par le premier compositor
rt0.get()->getBuffer(0,0)->blit(mip0.get()->getBuffer());
```

Deux passes seront utilisées à la fin du second compositor pour effectuer la reconstruction.

Résultats

Voici un détail du temps passé par notre algorithme sur chacune des étapes :

— Calcul de profondeur : pour un modèle simple, on est aux alentours de **450μs**, pour un modèle plus complexe, il faut compter **1.5ms**

— Èchantillonage : cette étape dure approximativement **2.4ms**

— Les deux étapes de flou durent chacune environ **590μs**

On arrive alors à un total compris entre **4et5ms** pour obtenir la carte d'occlusion ambiante de la scène.

Les résultats sont assez comparables avec ceux obtenus par les auteurs de l'article, l'étape de calcul de profondeur est cependant moins chronophage chez eux, ce qui est principalement dû à la manière dont il récupère le buffer de pronfodeur le calcul de d.

Voici quelques résultats en image :
— Sur le palais de Sponza :

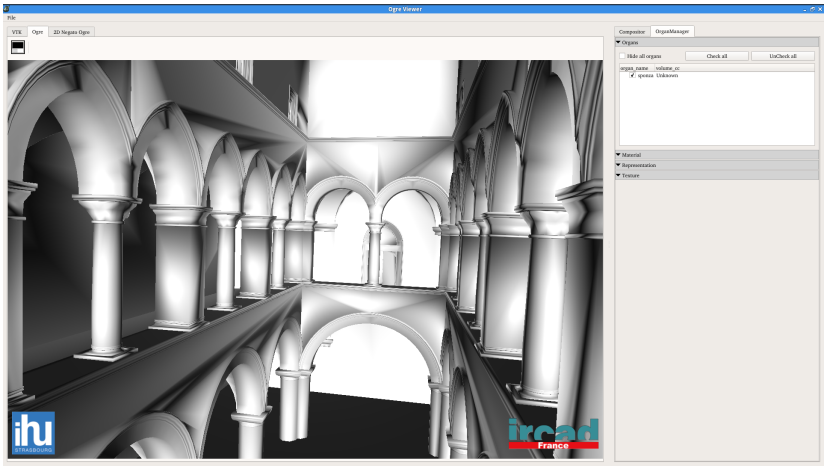


FIGURE 5 – image originale

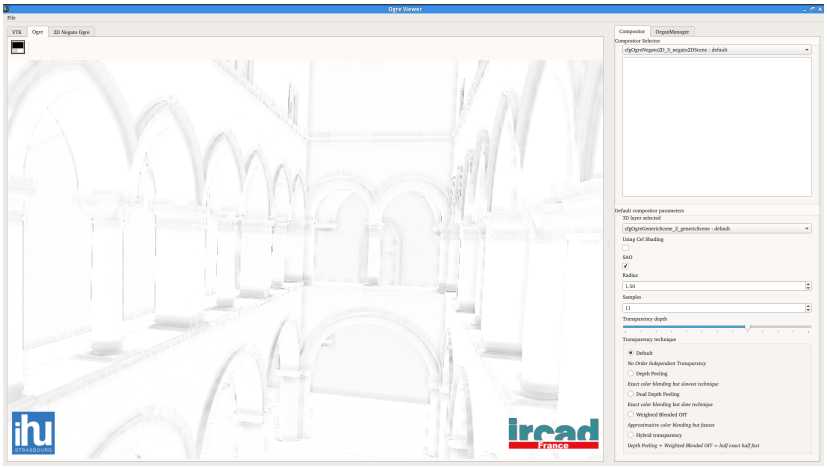


FIGURE 6 – application SAO avec r = 1.5 et s = 11

— Sur un corps humain :

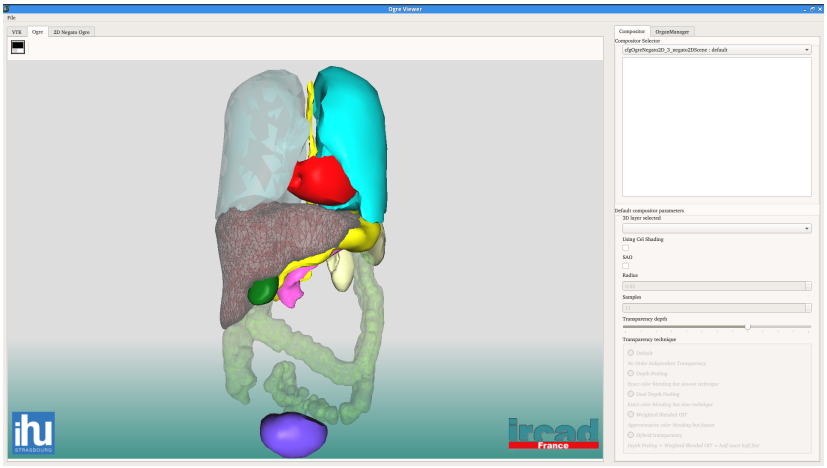


FIGURE 7 – image originale

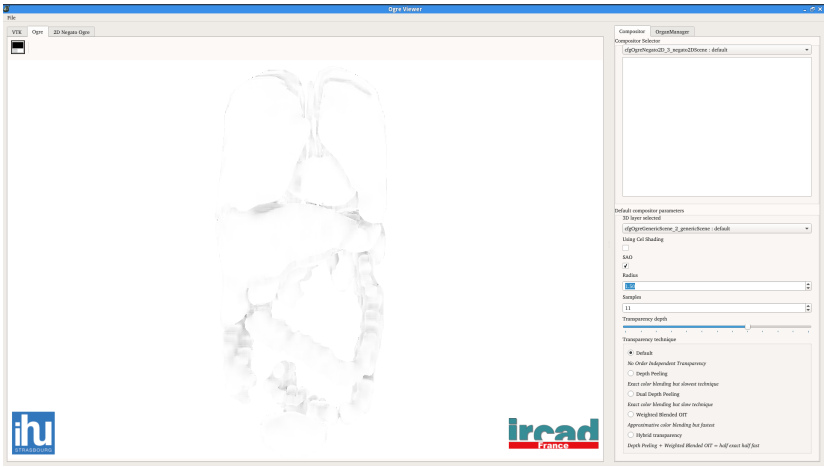
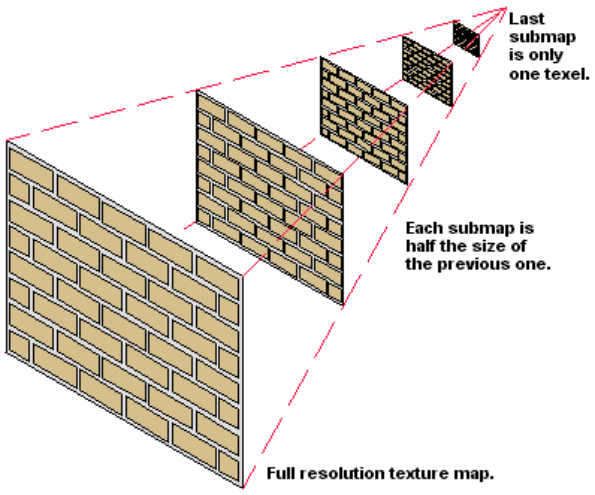


FIGURE 8 – application SAO avec r = 1.5 et s = 11

Annexe

Mip-Map

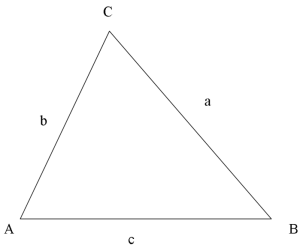
Le *Mip Mapping* est une technique permettant d'adapter la texture utilisée pour représenter une image donnée en fonction du niveau de détail souhaité.
On part d'une image I et on génère une pyramide de textures en réduisant à chaque étape les dimensions par 2.



Le niveau de détail sera donc adapté à la distance de l'objet, lorsque l'oeil est proche de l'objet, on utilisera une texture avec une haute définition, et lorsqu'il s'éloigne on préfèrera approximer le résultat et donner une réprésentation "grossière" de l'objet.

Heron

Soit un triangle ABC :



On peut calculer son aire grâce à la formule d'Héron :

$$A = \sqrt{s\,(s-a)\,(s-b)\,(s-c)}$$
$$\text{avec } s = \frac{(a+b+c)}{2}$$