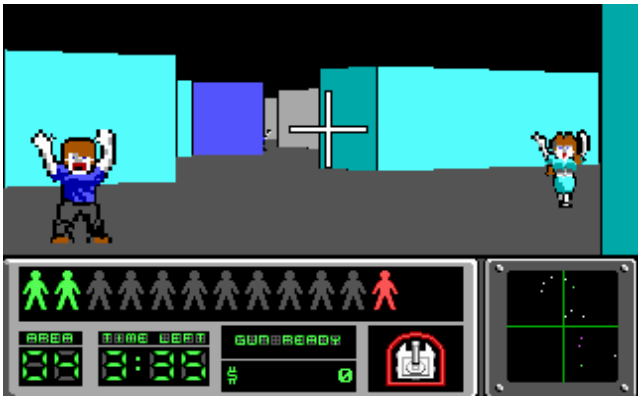
POLYTECH NANTES

École d'ingénieurs de l'université de Nantes

PROJET MULTIMEDIA

24/05/2011

L'évolution de la 3D temps réel dans le monde du jeu-vidéo



Illustrations : le premier moteur d'Id Software (Hovertank 3D, 1991) … et le dernier (Rage, 2011)

Binôme étudiant : Olivier CATRY – Alexandre GALERNE

Tuteur enseignant : Nicolas NORMAND

Projet Multimédia

L'EVOLUTION DE LA 3D TEMPS REEL DANS LE MONDE DU JEU-VIDEO

Sommaire

INTRODUCTION **2**

I – 10 ANS DE MOTEURS 3D EN TEMPS REEL **3**

Ray Casting 3
Back-face Culling 6
Voxels 10
Ombrage gouraud et éclairage basé sur les surfaces 14
A mi-chemin 17

II – 10 ANS D'ARTIFICES GRAPHIQUES EN TEMPS REEL **18**

Shaders 18
Techniques de Mapping 20
HDR 25
Screen space ambiant occlusion 28

III – QUELQUES TECHNIQUES EXPERIMENTALES **30**

CONCLUSION **34**

BIBLIOGRAPHIE **35**

GLOSSAIRE **37**

INTRODUCTION

Voici vingt ans que le premier jeu-vidéo en 3D est sorti. Ceci a marqué le début de la volonté de faire en temps réel des rendus graphiques en 3D tels qu'on les voyait à l'époque au cinéma ou dans d'autres domaines. Il fallait chercher de nouveaux algorithmes afin de générer ces rendus le plus rapidement possible. Cette recherche a duré jusqu'à l'apparition des premières cartes graphiques qui ont rendu possible l'application d'algorithmes plus gourmands grâce à une architecture spécialement conçue pour les traiter.

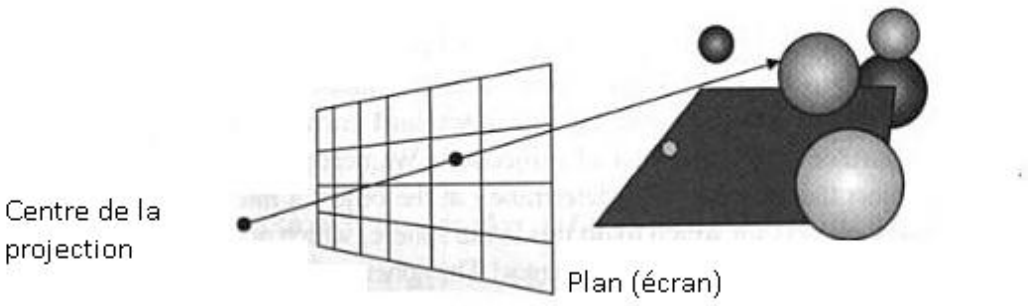
Ce dossier a pour but, dans un premier temps, de découvrir chronologiquement un certain nombre des techniques de rendu 3D en temps réel avec quelques algorithmes. En un second temps, et en suivant toujours une chronologie, les artifices 3D de nouvelle génération sont présentés, en présentant le principe de la technique employée. Enfin, nous découvrirons pêle-mêle des techniques et des moteurs spécialement conçus pour répondre à des besoins particuliers de rendus graphiques. Afin de rester le plus pragmatique possible, ce dossier s'efforce de présenter les applications accessibles au grand public, plutôt que les démonstrations technologiques qui sont souvent exécutées sur des machines très haut de gamme.

I – 10 ANS DE MOTEURS 3D EN TEMPS REEL

Ray Casting

PRINCIPE

La technique de *Ray Casting* (Lancer de rayon) consiste à tracer des lignes qui suivent les rayons de lumière, depuis les différents points d'un objet jusqu'à l'œil. L'intersection entre ces lignes et un plan de projection est dessiné sur ce plan. L'ensemble de ce qui est dessiné sur le plan est ce qui est affiché à l'écran.



[1] Schéma illustrant le principe du Ray Casting

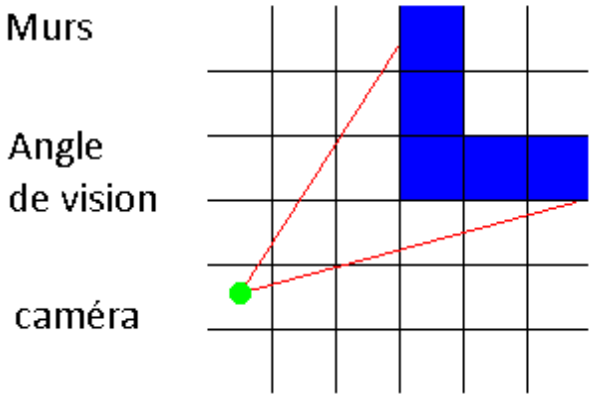
HISTORIQUE ET APPLICATION

Au début des années 90, la puissance des machines était encore trop faible pour réaliser du *Ray Casting* sur des formes 3D entièrement texturées. La technique imaginée par John Carmack en 1992 a été de créer une perspective 3D dans une carte en 2D. Chaque segment vertical du monde est calculé en fonction de sa distance à la caméra. Il y avait alors une complexité d'un seul calcul pour chaque ligne verticale, ce qui pouvait être calculé en temps réel [3].

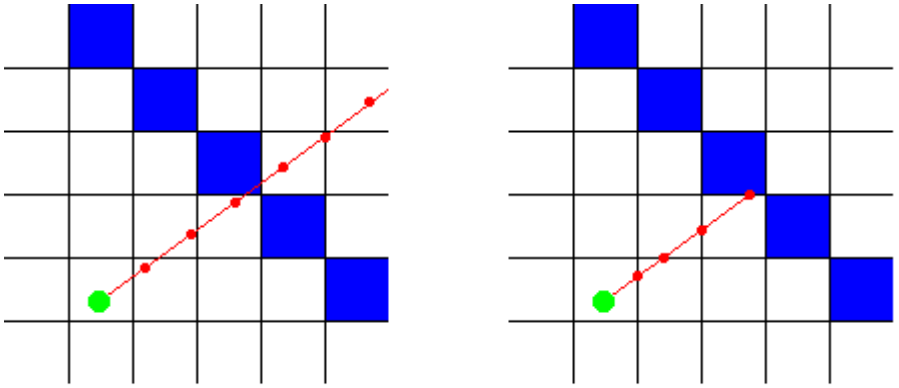
L'un des premiers jeux à appliquer cette technique était Wolfenstein 3D. L'astuce consistait à forcer toutes les lignes verticales à être de la même taille. Le monde créé ne pouvait donc pas contenir d'escalier et la caméra ne se déplaçait que sur deux dimensions.

Le principe était donc de projeter des lignes depuis la caméra, et dès qu'une ligne croise un mur, une ligne verticale est alors affichée, centrée verticalement, et sa taille est calculée en fonction de sa distance à la caméra.

La caméra se déplace dans le plan 2D. Elle peut pivoter sur une dimension, et peut se déplacer sur deux. Elle dispose d'un FOV déterminé qui limite le nombre de rayons à calculer. Le nombre de rayon est fini (taille en pixel de la largeur de l'écran)



Il n'est pas possible de déterminer si le rayon coupe un mur sur une infinité de points. Il fallait donc découper le rayon et trouver le premier point qui se trouvait dans un mur. Mais les machines de l'époque ne pouvait pas non plus réaliser ce calcule avec un découpage trop fin du rayon, ainsi, il était possible de manquer un mur (figure A). La technique trouvée consistait à découper le rayon selon la grille (figure B) : pour chaque intersection avec la grille, si celle-ci est adjacente à un mur, alors il s'agit de l'intersection avec le mur.



Pour appliquer une texture, il faut alors quantifier la largeur d'un mur (par exemple sur 256 pixels si l'on a des textures de 256 pixels de large). Le moteur calcule le numéro de la colonne au sein du mur qui est touchée par le rayon, et la colonne correspondante sur la texture est appliquée au segment affiché.

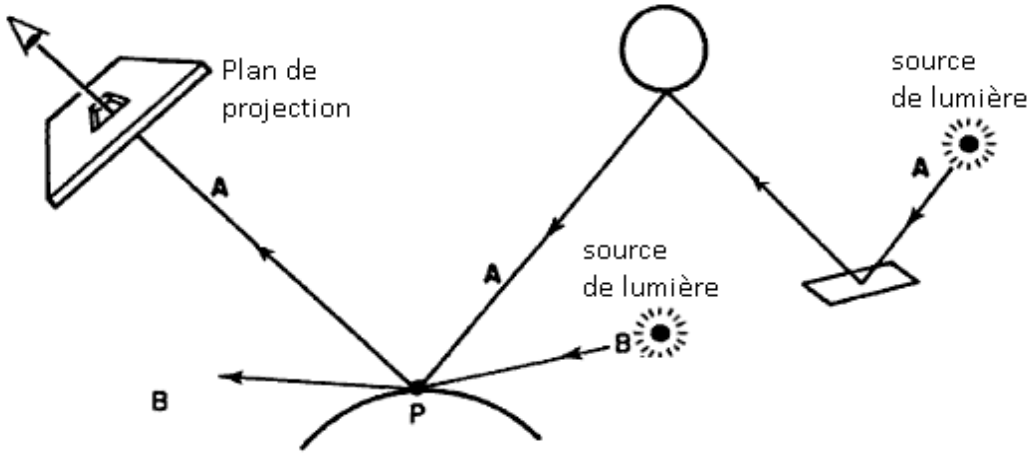
Une extension de la technique consiste à afficher des sprites 2D. Ceux-ci sont systématiquement alignés sur la grille, donc le rayon peut les croiser. Mais le rayon doit continuer son chemin, car le sprite peut être en partie transparent, laissant visible le décor derrière. Le rayon peut donc indiquer l'affichage d'un mur et de plusieurs sprites.

EXEMPLE

Wolfenstein 3D est l'un des premiers jeux 3D texturé, utilisant la technique de Ray Casting. A gauche, la version PC de 1992 et à droite la version pour téléphone mobile actuelle.

EVOLUTION

Le *Ray Casting* sera de nouveau utilisé avec la volonté de diffuser la lumière.



[2] Certains rayons de lumière émis par la source lumineuse A sont réfléchis avec une certaine perte par différents objets et coupent le plan de projection. Le point P n'est donc pas affiché seulement en fonction de l'éclairage unique de sa lumière d'émission précalculée, mais aussi en fonction de la lumière qu'il reçoit et qu'il diffuse.

Back-face Culling

Principe

La technique de *Back-face Culling* consiste à déterminer quels triangles sont vus par la caméra afin de limiter le nombre de calculs de triangles à afficher.

Historique et application

Après 1992, Id-Software voulait profiter de l'évolution de la puissance des machines pour développer un nouveau moteur 3D qui ne se baserait plus sur le *Ray Casting,* afin de permettre à la caméra de se déplacer sur trois dimensions au lieu de deux. La rotation continuerait de se faire sur une dimension.

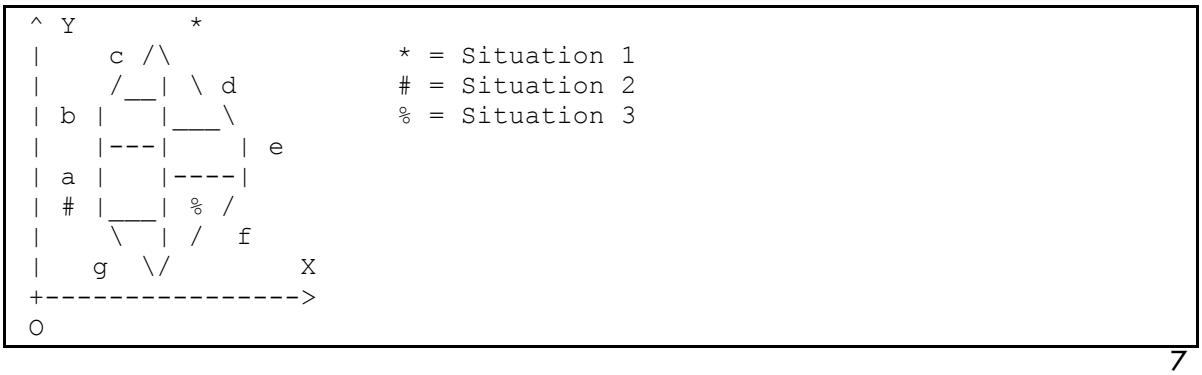
La technique utilise un dérivé de la méthode de *BSP Tree* (*Binary Space Partition Tree* – Arbre de Partition binaire de l'espace, nous reviendrons sur la réelle application de cette méthode plus loin) nommée sectorisation. Brève explication de son fonctionnement dans ces types de moteurs :

L'espace peut être subdivisé en un ensemble de plans sur deux dimensions (parallèles au sol ou au plafond). Le premier plan, celui qu'on l'on assimile à l'univers, permet de contenir plusieurs plans. Chacun de ces plans permet de contenir d'autres plans. En continuant ainsi récursivement, on se rend compte que cette structure est un arbre.

[4] Le plan A (l'univers, c'est à dire ce qui contient toute la carte) contient indirectement le plan B (c'est à dire que le plan B est inclus dans un plan plus vaste qui est contenu directement dans la carte). Le plan B contient le plan C (le plan C ne dépasse par du plan B, vu de haut). Le plan D n'est pas contenu dans le plan B, et n'est donc pas non plus dans le plan C.

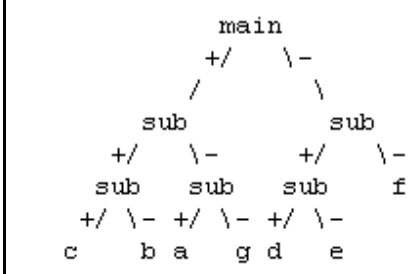
On reconnaît selon cette description une architecture en arbre.

Ceci permet de connaître l'ordre dans lequel il faut dessiner les plans : avec un parcours en largeur ou un parcours suffixé de l'arbre. Un exemple avec une scène "vue de haut" et différents angles de vue [5] :



ordre de dessin	Situation 1	2	3
1	g	d	c
2	a	e	b
3	b	f	a
4	c	c	g
5	f	b	d
6	e	g	e
7	d	a	f

Le BSP-Tree associé :



La partie suivante de ce dossier reviendra sur le BSP-Tree et sur une exploitation de ce procédé avec le partage en secteurs du moteur Build.

Analysons comment est déterminée la question de l'orientation des polygones. Pour cela, on choisi trois points du polygone. L'orientation du triangle ainsi extrait déterminera l'orientation du polygone.

La méthode pour déterminer si un triangle est face à la caméra est de calculer le cosinus entre deux vecteurs : l'un est le vecteur normal au triangle, et l'autre est le vecteur vers la caméra. On sait alors si le triangle fait face à la caméra et s'il doit être affiché.

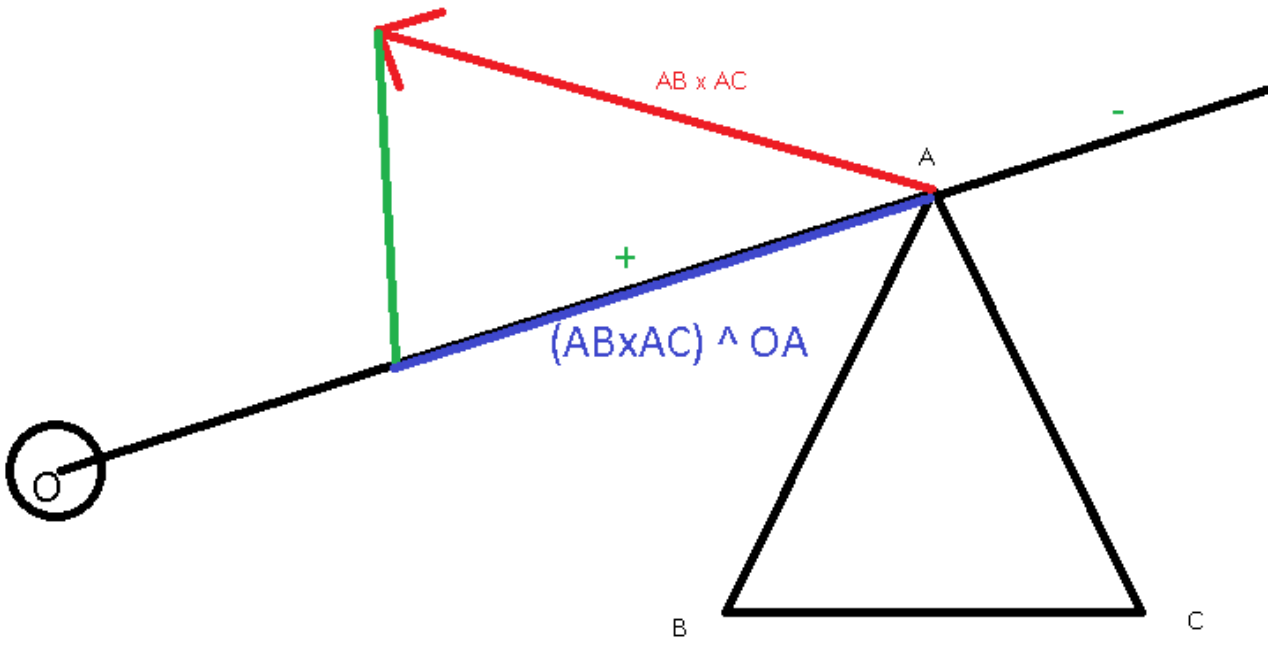
EXEMPLE D'IMPLEMENTATION

```
Vector3f cameraToFace = new Vector3f(cameraPosition.x-face[0].x,
                                     cameraPosition.y-face[0].y,
                                     cameraPosition.z-face[0].z);
Vector3f ab = new Vector3f(face[1].x-face[0].x,
                           face[1].y-face[0].y,
                           face[1].z-face[0].z);
Vector3f cb = new Vector3f(face[1].x-face[2].x,
                           face[1].y-face[2].y,
                           face[1].z-face[2].z);
Vector3f faceNormal = cross(cb,ab);
triangleNormal = faceNormal;
float result = dot(cameraToFace,faceNormal);
return result > 0;
```

- *cameraPosition* est la position de la caméra et *face* est le triangle à afficher. La face contient trois triplets de coordonnées qui sont celles des sommets du triangle. L'ordre a une importance puisqu'il permet de déterminer le sens du vecteur normal (produit vectoriel)

- Le vecteur *cameraToFace* est calculé et est le vecteur d'un point de la face à la caméra.
- On crée les vecteurs *ab* et *cd* qui sont les vecteurs représentant AB et AC avec A,B et C appartenant au triangle.
- Le produit vectoriel (cross) *faceNormal* de ces vecteurs est le vecteur normal du triangle.
- Le produit scalaire (dot) de *cameraToFace* et *faceNormal* a donc une valeur, dont le signe indique si le triangle fait face ou non à la caméra : + = oui , - = non.

Schéma explicatif dans un espace 3D :



O est la caméra, et A B et C sont trois points du polygone dont on cherche à connaître l'orientation.

Le *Back-face Culling* repose ainsi sur le principe de *secteur* plus évolué que celui des moteurs d'ancienne génération et utilise la méthode de *BSP Tree* afin de déterminer sous forme d'arbre l'ordre et la sélection des plans à afficher, tout en vérifiant l'orientation des triangles afin de déterminer s'ils doivent être affichés.

EXEMPLE

Doom (à gauche) est le premier produit utilisant le nouveau moteur d'ID Software, le Id Tech 1, et exploitant la technologie de Back-face Culling. Heretic et de nombreux autres jeux ont été développés sur ce moteur, et un portage utilisant les librairies modernes (DirectX, OpenGL ...) a été réalisé. A droite, Heretic sur le portage Zdoom, montrant l'exploitation du Back-face Culling par les librairies modernes.

Voxels

PRINCIPE

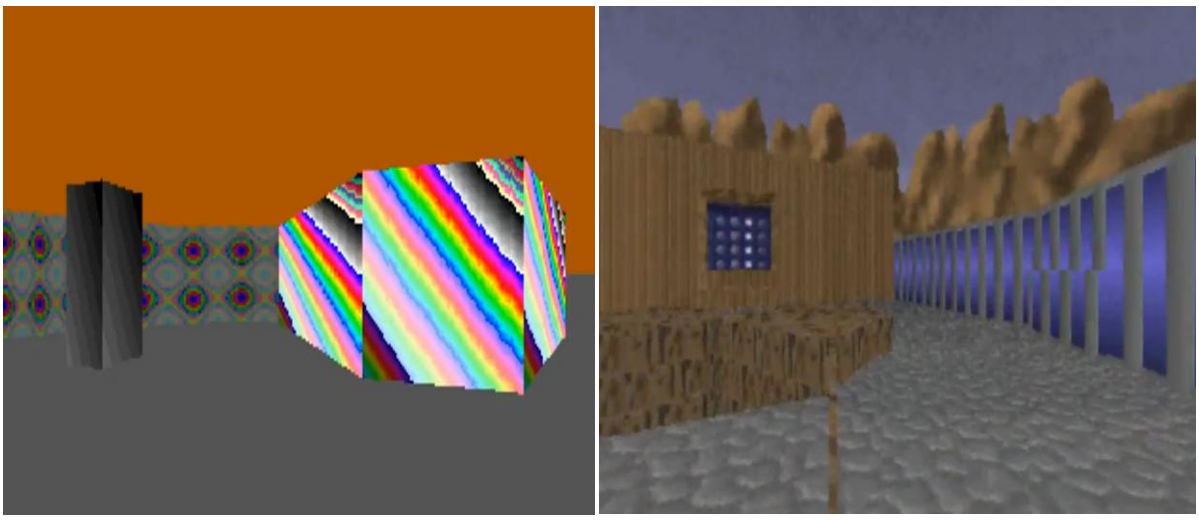
Les Voxels (Volumetric pixel) sont des pixels en 3D, alignés sur une grille 3D, qui forment un volume. On peut faire une analogie avec une image bitmap constituée de pixels en 2D alignés sur une grille 2D.

HISTORIQUE ET APPLICATION

Pourquoi et comment les voxels ont été appliqués dans les jeux-vidéo ?

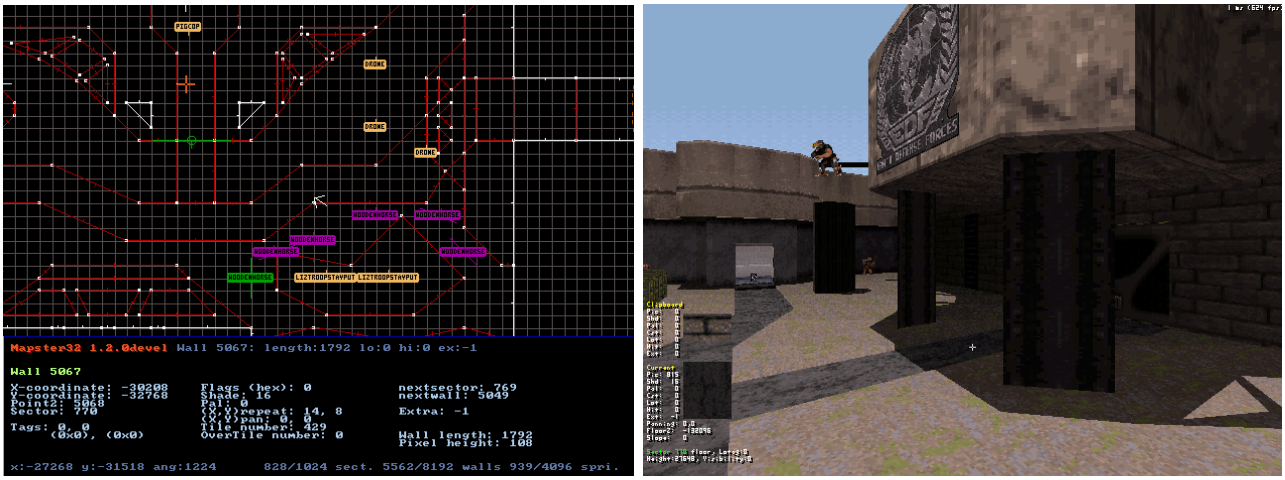
[7] Pendant le développement de Doom par Id Software, Ken Silverman, agé alors de 17 ans, a réalisé son propre jeu de tir à la première personne basé sur le même principe que le moteur de Wolfenstein 3D : Walken.

En Mars 1993, son moteur a été amélioré avec le support de murs anguleux. Puis Ken a créé un nouveau moteur 3D qu'il nomme "Build", basé sur le principe de *ray casting* sur une grille 3D.



A gauche le moteur de *ray casting* 2D avec murs anguleux, et à droite, le moteur de *ray casting* 3D. On notera les erreurs d'affichage liées au phénomène évoqué précédemment.

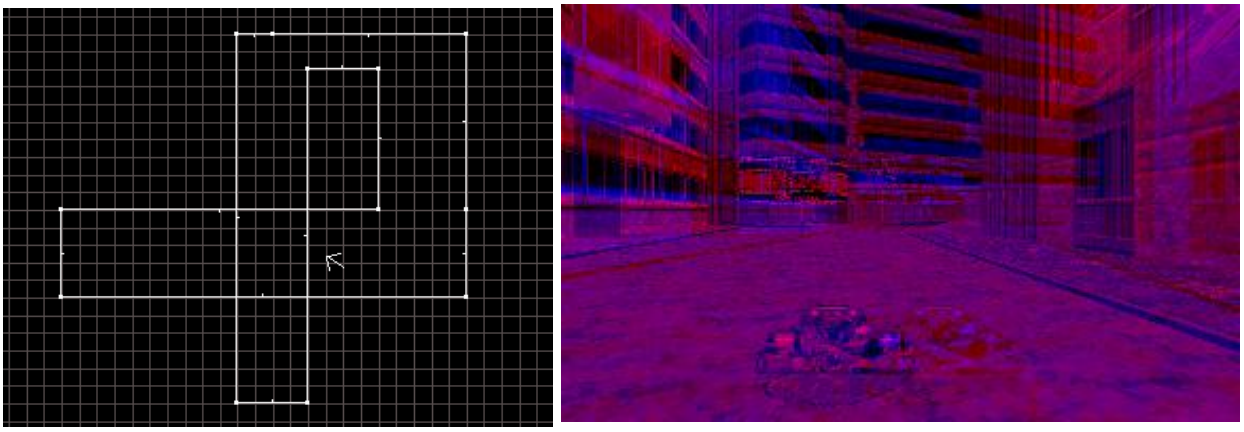
En Juin 1993, il entre en contact avec John Carmack d'Id Software, et exploite son idée de sectorisation (expliqué précédemment, le *BSP Tree*). Il abandonne alors la conception en grille et reprogramme son moteur. Les secteurs sont toujours décrits en 2D et décrivent deux plans encadrant l'espace visible. Il est alors possible en 3D de modifier l'espace entre ces deux plans, et, sur le moteur de Silverman, de modifier leur inclinaison. Le tout pouvait être géré en temps réel par l'intermédiaire de *sector effectors*, des entités programmables qui modifient en temps réel les attributs de leur secteur.



A gauche, la conception "vue de dessus" d'un niveau avec le logiciel Build, basé sur le moteur Build. Le curseur (viseur orange) est placé sur le secteur 770 décrit par une série de sommets. Ce secteur est contenu dans un et un seul secteur, lui même contenu dans un et un seul secteur qui contient plusieurs secteurs. A droite, la même situation "vue en 3D" depuis la flèche blanche. Le curseur blanc indique le même secteur que celui précédemment indiqué. On reconnaît bien les secteurs de la vue 2D, et les techniques pour rendre transparente la structure d'arbre (ex : la route semble couper le secteur ombragé, mais elle est en fait constituée de deux secteurs) (Duke Nukem 3D - 1996)

Note : Le moteur Build a eu la particularité totalement imprévue par son conepteur de permettre à deux espaces de se confondre. Les coordonnées X et Y des deux espaces ne doivent être visibles en 3D qu'au sein d'un seul des deux espaces. Ceci permet la technique du *Room over room* qui a été un argument de vente du moteur.

N'oublions pas également que le moteur Build permettait l'affichage en relief, quinze ans avant l'engouement pour cette technique dans le domaine du cinéma.

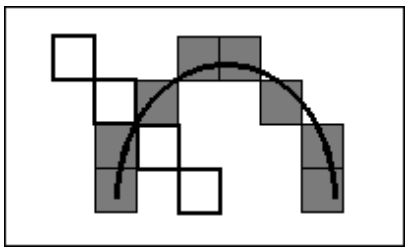


A gauche, le *Room over room* et à droite, le relief exploitant le R&B stéréo (séparation Rouge et bleu)

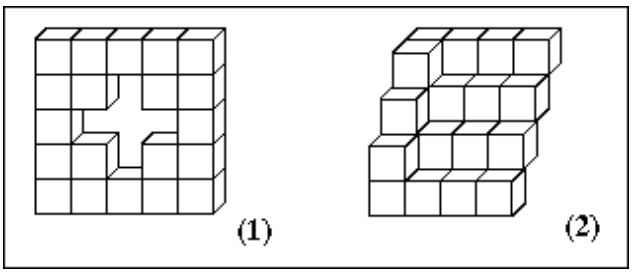
[8] Les entitées représentées dans le monde 3D sont restées des sprites 2D jusqu'en 1996 avec la sortie de Duke Nukem 3D pour lequel le temps a manqué pour l'implémentation des voxels. On est alors resté sur une conception 3D avec la contrainte "uniquement deux plans sur l'axe Z", que l'on appelle la 2.5D, même si celle-ci pouvait être contournée avec l'utilisation de sprites 2D plats. C'est alors Blood, basé sur le moteur Build qui a exploité les Voxels et qui a changé la conception 3D, permettant de s'éloigner du concept de 2.5D

Pour tourner autour d'un sprite 2D, celui ci changeait d'apparence en fonction de l'angle de vue. C'est ce que l'on nommait les sprites à 8 faces. Ces sprites seront également remplacés par la technique des voxels qui permettront une gestion totalement 3D.

[9] La voxelisation repose sur la discrétisation 3D : L'espace 3D est réduit à une grille 3D. Chaque cube de la grille prend pour valeur 0 (pas de voxel) ou 1 (un voxel) ou une autre valeur permettant de qualifier le voxel. Les voxels sont utilisés pour discrétiser le volume que l'on veut représenter. La question de la connexité se pose : en effet, permettre une 14-connexité laisse la possibilité à deux volumes 3D de se confondre sans exploiter les mêmes voxels. Exemple en 2D avec la 8-connexité :



Ce phénomène est lié à la séparation. On parle alors de k-séparation. Il s'agit de l'adjacence maximale existante entre les voxels discrétisant le volume. Deux voxels peuvent avoir des faces, des arêtes ou des sommets en commun. On peut donc avoir une 6, 18 ou 26-séparation. La 6-séparation ne permet pas ce phénomène de croisement sans voxel commun.



[9] A gauche, un volume 6-connexe qui n'est pas 6-séparant (il y a une 3-adjacence au maximum) et à droite, un volume toujours 6-connexe (observer l'escalier de gauche pour comprendre sa formation) qui n'est pas 18-séparant.

EXEMPLE

Une pierre tombale dans le jeu Blood (1997). Ce volume présente des plans parallèles au sol ou au plafond qu'il n'était pas possible de construire sur les moteurs précédents.

Ombrage gouraud et éclairage basé sur les surfaces

Principe

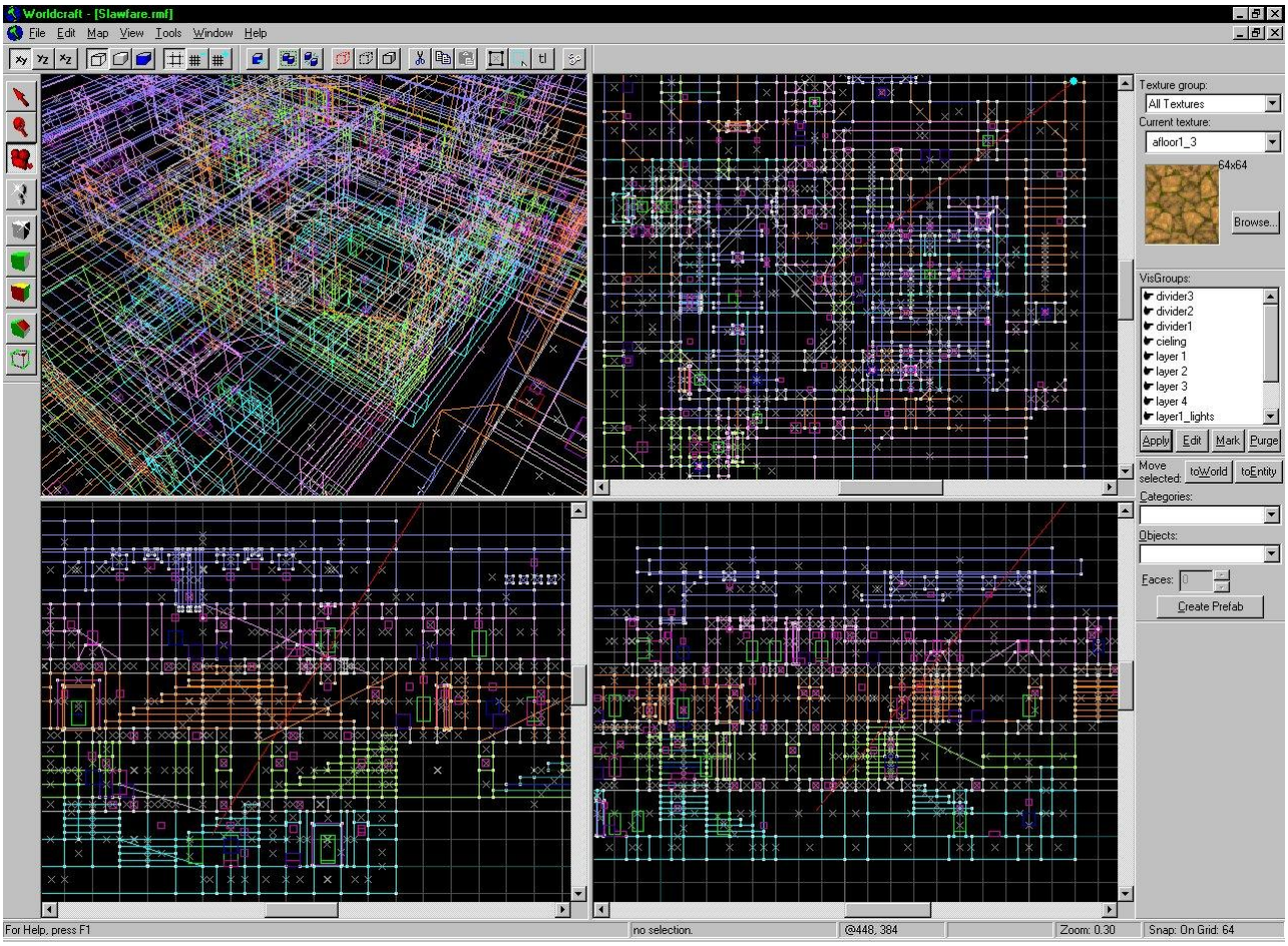
Cette technique permet d'éclairer le décor avec une précision au texel près en fonction de plusieurs sources lumineuses.

Historique et application

Jusqu'en 96, l'éclairage dans de nombreux jeux était fait à la main, par l'intermédiaire d'un changement de luminosité des textures appliquées aux murs et aux secteurs.

C'est John Carmack qui, ayant déjà créé le premier jeu en pseudo-3D texturées en 1992, a créé le premier jeu en 3D texturées en 1996. Le quake engine se débarasse alors de la conception en secteurs, pour une conception des niveaux en 3 dimensions.

La technique utilisée pour l'affichage des polygones se base sur le BSP-Tree, qui rappelons-le, est une technique par laquelle les polygones sont rangés dans un arbre selon leur position. Un polygone cachant totalement d'autres polygones empêche l'affichage de ces derniers. La conception des cartes repose sur une technique de partitionnement, obligeant notamment les concepteurs à créer des tunnels à angle droit afin d'empêcher la vision simultanée de plusieurs parties du niveau.

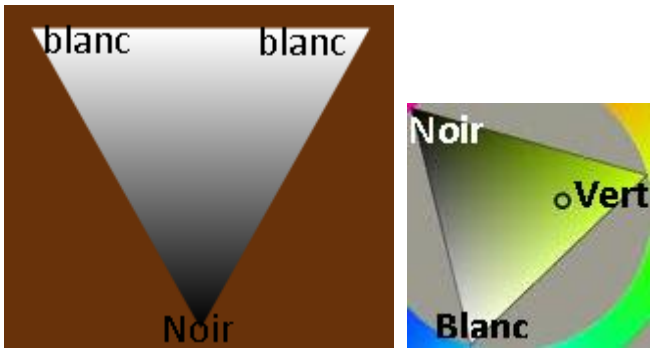


Worldcraft, l'éditeur de carte du jeu Half-life (1998) basé sur une version améliorée du moteur de Quake 1, montre la différence fondamentale entre la conception 3D et la conception 2D telle que celle de Build.

[10] C'est le premier moteur incluant un préprocesseur pour la construction des cartes, afin d'optimiser le rendu sur les machines de l'époque. Il s'est alors posé la question de la technique d'éclairage : le niveau n'est pas figé, et il n'est plus question de définir des zones d'intensité lumineuses différentes.

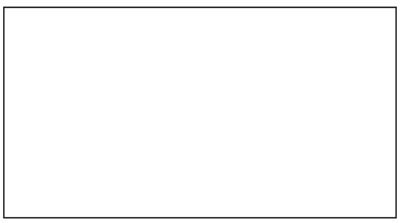
Ombrage Gouraud

[10][11] La technique d'éclairage appliqué aux éléments mobiles avec peu de polygones, tels que les personnages ou les petits objets, utilise la technique d'ombrage Gouraud (*Gouraud shading*). Cette technique consiste à calculer la distance à la source lumineuse de chaque vertex de la surface, et d'appliquer un dégradé de couleur sur chaque triangle en fonction de la valeur de l'éclairage de chacun de ses sommets.

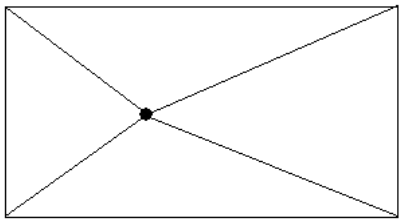


Par exemple, sur chaque sommet de ces triangles, la valeur de l'éclairage est retenue (d'usage, elle est enregistrée au format RVB), et une coloration linéaire est appliquée. Ici, les valeurs sont plutôt extrêmes, mais un volume présente de nombreux triangles et donc de nombreux vertex, pour lesquels la valeur de la lumière reçue admet des variations légères.

Cette technique nécessite le découpage de tous les polygones en triangles. L'augmentation du nombre de vertex est une des limites d'une telle technique d'éclairage.

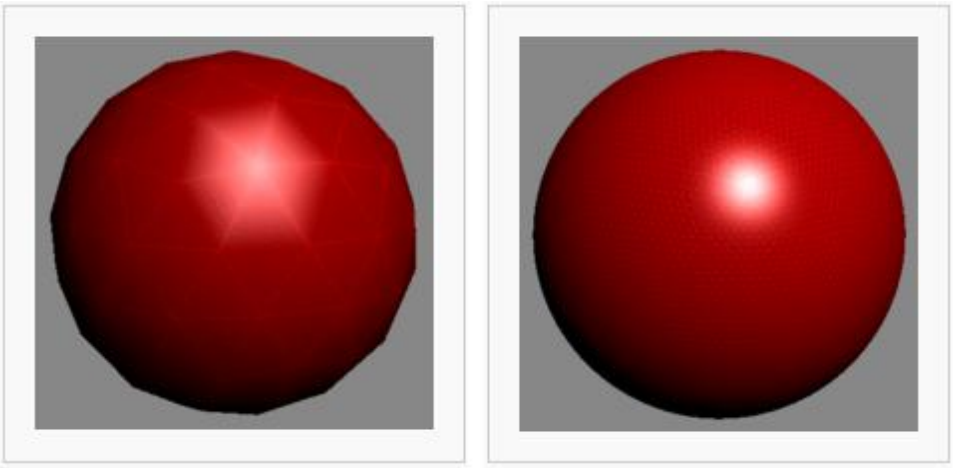


Le polygone avant d'ajouter un vertex pour l'éclairage



Le polygone partagé en quatre triangle avec un vertex en plus

Le second problème est la gestion des projections d'ombres : la bordure d'une ombre ne peut pas suivre une trajectoire rectiligne avec une telle technique, car elle dépend de l'emplacement des vertex.

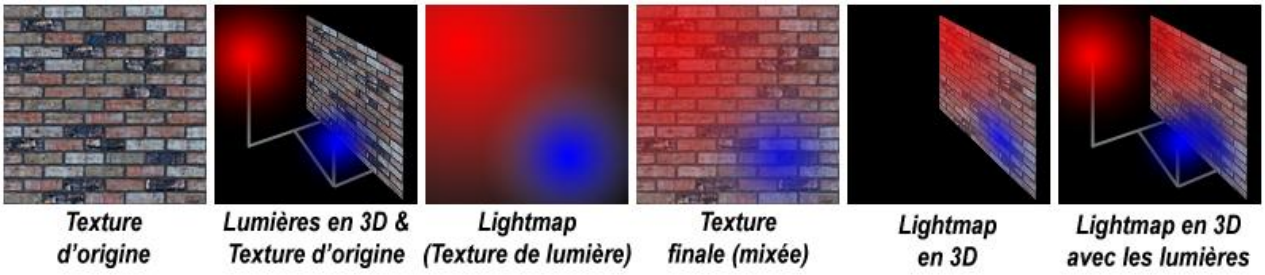


La technique d'ombrage de Gouraud appliquée à deux sphères. Celle de droite étant composée de plus de polygones que celle de gauche.

Éclairage basé sur les surfaces

Avec le préprocesseur, il est devenu possible d'appliquer une technique d'éclairage plus onéreuse car celle-ci serait alors appliquée lors de la construction de la carte et non gérée en temps réel. Bien que des techniques ont été depuis découvertes pour palier à cette contrainte, ce type d'éclairage ne peut s'appliquer qu'à des éléments fixes, ou il faut sinon accepter que l'éclairage reste statique sur un élément en mouvement.

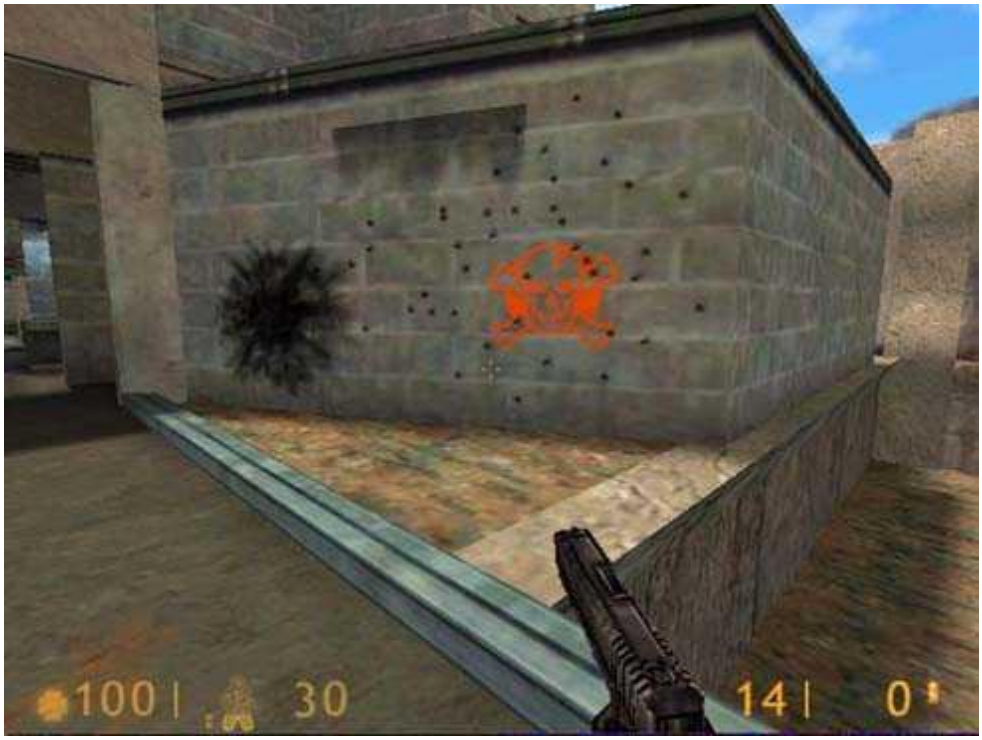
Cette technique d'éclairage ne se base plus sur l'éclairage des vertex mais sur celui des texels. Chaque groupe de texels (le choix de la taille du groupe influe sur la rapidité du preprocessing) enregistre l'éclairage qu'il reçoit parmi plusieurs sources de lumière. L'éclairage ne depend alors plus du découpage des polygones mais de la taille de leurs textures. On conserve alors l'aspect rectiligne des bordures des ombres. On peut enfin appliquer un filtre sur les valeurs pour supprimer l'effet d'escalier.



L'idée de ce genre de texture unique par polygone, fournissant des informations en plus de la texture qui lui est appliquée, sera réutilisée dans de nombreus autres cas pour tous les moteurs à partir de cette

époque. Il s'agit de la technique de mapping. Nous venons de voir le light-mapping, mais nous verrons aussi le bump-mapping ou encore le parallax-mapping.

EXEMPLE



Half-life (1998) basé sur une version améliorée du moteur de Quake 1 (notamment d'une diminution de la taille des groupes de texels) utilise la technique de mapping pour l'éclairage, mais aussi pour l'application de tag et de divers textures telles que les impacts ou les marques d'explosion, qui s'appliquent sur les groupes de texels. Ce sont des *decals*.

A mi-chemin

Cette première partie a montré comment les moteurs 3D ont évolué en 10 ans, en partant du ray casting, une conception en deux dimensions, et un affichage en une dimension, en passant par le Back-face culling avec sa conception toujours en deux dimensions mais un affichage en trois dimensions, et en arrivant aux moteurs à conception en trois dimensions et affichage en trois dimensions. Cette dernière technique que l'on utilisait aux alentours de l'an 2000 est à la base des moteurs 3D actuels.

Pourtant, 10 ans se sont écoulés depuis l'an 2000, une interpolation simple par rapport à l'histoire des années 90 devrait nous faire douter d'une grande avancée des moteurs 3D. Alors, comment a-t-on maintenu une évolution à l'image de celle des années 90 en conservant des bases similaires sur le moteur de rendu 3D ?

Les avancées dans la recherche en imagerie et en 3D ont révélé de nombreuses améliorations graphiques que l'on peut greffer à des éléments 3D comme on a pu le voir dans de nombreuses œuvres cinématographiques des années 80 et 90, mais tout ceci ne pouvait pas être à l'époque géré en temps réel. L'accroissement de la puissance des CPU, l'amélioration des cartes graphiques et de leur CPU et les ajouts dans les librairies graphiques (DirectX, openGL etc.) ont permis ces greffes aux moteurs de jeux-vidéo afin de les exploiter en temps réel.

La suite de ce dossier présente certaines de ces améliorations graphiques et leur application dans les jeux-vidéo des années 2000 que l'on nommait pour cela des jeux "nouvelle génération" (Next Gen.)

II – 10 ANS D'ARTIFICES GRAPHIQUES EN TEMPS REEL

Shaders

PRINCIPE

[13] Pour rappel, les volumes sont constitués d'un ensemble de polygones. Au sein de chacun de ces polygones, on ajoute un point afin que ces polygones deviennent un ensemble de triangles. Un volume est donc constitué d'un ensemble de points repérés dans l'espace, que l'on appelle vertex, et qui sont les sommets des triangles constituant ce volume.

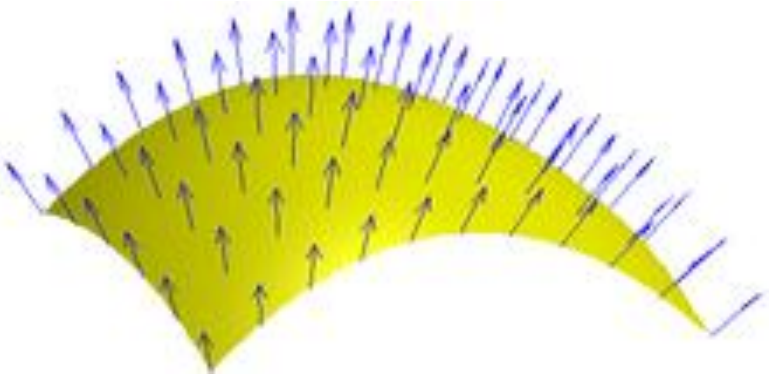
Ces vertex sont caractérisés par de nombreux attributs, dont par exemple les caractéristiques lumineuses ou la texture. La technique de *vertex shading* consiste à changer la valeur des attributs du vertex en fonction de l'emplacement de l'angle de vue. Les *geometric shaders* peuvent être appliqués après les *vertex shaders* à des mesh et se servent des triangles eux-mêmes comme primitive. Les *pixel shaders* sont des shaders qui s'appliquent directement aux pixels (et non aux texels) qui constituent la projection du volume à l'écran.

HISTORIQUE ET APPLICATION

[14] Fin 1999, Id Software a sorti l'Id Tech 3, avec notamment une gestion des shaders basée sur les vertex shaders. Le concepteur de niveau doit alors indiquer les caractéristiques variables sur les vertex d'un volume 3D afin de lui appliquer un rendu proche du matériau dont il est constitué. Il existe dans les fichiers de configuration de nombreux shaders pré-enregistrés. En fonction de l'angle de vue et de l'éclairage qu'ils reçoivent, la couleur de certains vertex va changer, afin que la surface ait des reflets.

Par exemple, il est possible d'appliquer un rendu "eau" à une surface dont la texture est de la terre, on a alors l'impression de voir de la boue.

Le problème de cette technique est qu'il faut déterminer l'orientation des vertex et l'angle que cela fait avec l'angle de vue (comme on l'a vu plus tôt sur la détermination de l'orientation d'un polygone). Il faut alors faire la moyenne du vecteur normal de chaque triangle et l'appliquer au vertex commun, et cela, pour tous les vertex du volume. On atteint alors une complexité trop importante.



[15] Vecteurs normaux des vertex d'une surface

[16] Pour cette raison, Id software a programmé une optimisation du calcule de l'inverse de la racine carré pour gérer ces shaders. Cette fonction a permis un grand pas en avant dans les moteurs 3D temps réel car elle permet de calculer quatre fois plus vite cette valeur que par la voie normale.

Avant Id software, Newton s'était déjà penché sur une approximation de l'inverse de la racine carré d'un nombre…

[17] Le principal concurents d'Id Software sur le plan moteur 3D était Epic Games avec leur série de moteur Unreal Engine. La version 2 a vu une amélioration qui s'est imposée peu à peu au cours de l'année 2003 et qui propose la gestion des shaders model 3.0. C'est un rassemblement des trois techniques de shaders que nous avons évoquées.

Le problème vient surtout des pixel shaders, dont le temps de calcul est proportionnel à la résolution de l'écran. Les constructeurs de carte graphique ont rendu possible ce rendu en temps réel grâce à un composant dont l'architecture est entièrement dédiée aux calculs de shaders. Nvidia a, par exemple, commencé ce genre de modèles à partie de la Geforce 3 (2001).

(nekocake.com, IGN.com)

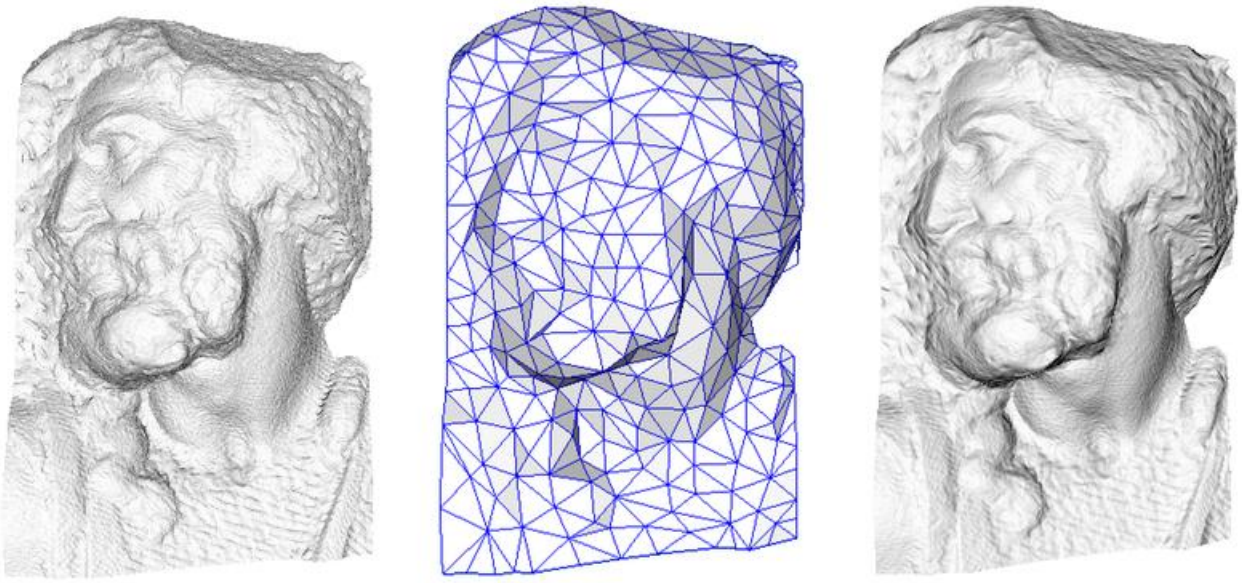
A gauche, un rendu non-texturé sous Qmap d'une carte de Quake 3 Arena (Id Tech 3, Decembre 1999) montrant l'application de l'éclairage modifié par les vertex shaders. A droite, un rendu de Splinter Cell Chaos Theory (unreal engine 2.5, 2004) avec les Shaders Model 3.0 activés, montrant la modification de l'éclairage appliqué aux surfaces en fonction de l'ambiance lumineuse, des textures et du point de vue, et également la gestion "au pixel près" des reflets entourant l'ombre du personnage rendu possible grâce aux pixel shaders.

Techniques de Mapping

PRINCIPES

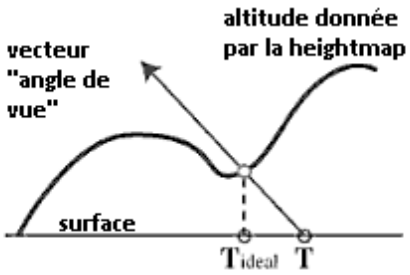
Le **mapping**, comme on l'a vu, est un principe selon lequel une texture invisible va être prise en compte pour modifier l'apparence de la texture visible appliquée à une surface. Cette technique était utilisée dans le cadre du light mapping, qui consistait à appliquer un éclairage géré au texel près sur une surface.

Le **Normal mapping** utilise un ensemble de textures avec son mapping d'éclairage appliquées à plusieurs polygones pour les plaquer sur un seul polygone, qui est une approximation de ces plusieurs polygones. Le mesh est alors constitué de moins de polygones, et son rendu texturé donne l'impression d'avoir plus de polygones. [19]



Le **bump mapping** utilise une texture invisible (heightmap) pour indiquer la profondeur de chaque texel au sein d'une surface (avec une variation sur une composante quelconque : la teinte, la luminance etc.). Cela va alors altérer l'éclairage de ces texels. On parle de *placage de relief*. [18]

Le **parallax mapping** utilise la même texture "heightmap" que pour le bump mapping, mais ne va pas altérer l'éclairage, mais l'affichage du texel. Il repose sur le principe qu'un texel est affiché différemment selon le point de vue et selon les texels voisins et leur profondeur relative. C'est un problème de parallaxe. [20]

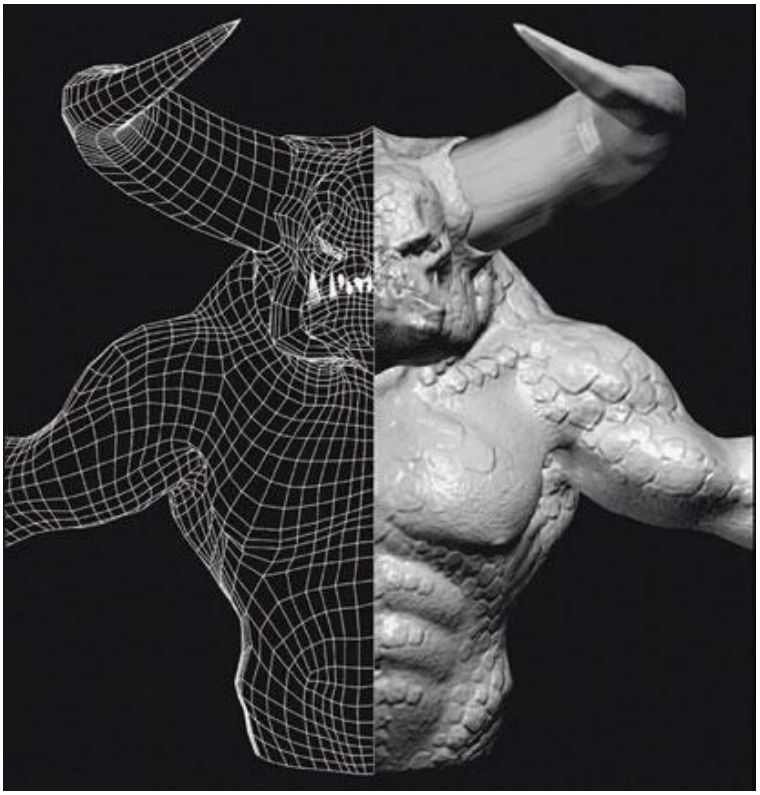


Le texel Tideal est projeté selon l'altitude qu'il doit avoir, donnée par la heightmap. Selon cet angle de vue, on doit donc voir le texel Tideal à l'emplacement T sur la surface. L'application de ceci à chacun des texels va donc générer une nouvelle texture "déformée" selon la heightmap et appliquée à la surface.

APPLICATION

En 2004, Id Software sort son nouveau moteur Id Tech, le numéro 4. C'est le dernier moteur Id Tech présenté dans ce dossier, car l'Id tech 5, dévoilé en 2007, supporte des jeux qui sont toujours en développement aujourd'hui en 2011.

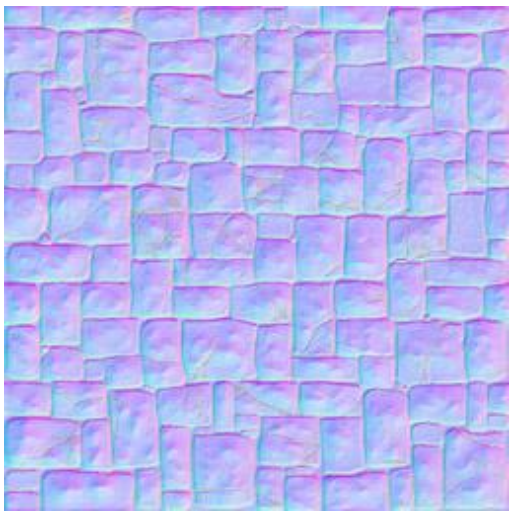
La grande avancée avec ce moteur était le "shadow volume" (ombres volumétriques), qui est une projection "au pixel près" de l'ombre projetée par un volume. La problématique est la complexité de l'algorithme, car plus un mesh était composé de nombreux polygones, et plus les calculs de son ombre projetée étaient importants. Le moteur utilise donc le normal mapping afin de réduire le nombre de polygones constituant le mesh tout en préservant le détail de sa texture.



(iphonefreakz.com)

Cette modélisation d'une créature du jeu Doom 3 (Id Tech 4, 2004) montre l'éclairage appliqué au mesh, qui a d'abord été calculé avec un grand nombre de polygones, et la lightmap a été appliquée au même mesh avec moins de polygones.

Quelques mois auparavant, Epic Games s'était penché sur le bump mapping, implémenté dans l'amélioration de leur principal moteur : l'unreal engine 2. Toujours dans la même optique de réduire le nombre de polygones, la technique générait un éclairage différent sur la texture en fonction de l'angle de la lumière et de la texture hightmap.



Normal map d'une texture de brique après l'application de son heightmap couplé à l'angle et les caractéristiques de la lumière reçue. La texture originale n'a plus qu'à être appliquée.

En 2005, une équipe de développement a sorti une amélioration de son moteur dont le jeu phare a impressionné le public pour ses graphismes : Le Lithtech Jupiter EX de Monolith Production était l'un des premiers moteurs à inclure abondamment la technique de parallax mapping. On notera son application en temps réel aux decals mais aussi aux textures générée lors du preprocessing de la carte.

[21] La technique est très proche de celle du lancer de rayon, et on s'est trouvé à nouveau confronté aux mêmes problèmes d'appréciation de distance qu'en 1992 avec la discrétisation des intervalles de vérification. Le travail doit donc s'effectuer sur une grille afin d'être certain de rencontrer la surface de la heightmap lors du lancer de rayon. L'autre problème de la discrétisation est que la surface rendue est voxelisée : un texel qui apparaît à une plus forte altitude qu'un autre est relié à cet autre texel par un plan orthogonal à la surface. Le parallax mapping apparaît donc comme une surface souffrant d'un "effet d'escalier". Un filtre doit alors être appliqué de façon à lisser après projection des texels.

Exemple de parallax mapping appliqué à une surface texturée et bump-mappée.

EXEMPLES

(insidemacgames.com, tomshardware.com)

A gauche, un rendu de Doom 3 (Id Tech 4, 2004) montrant l'application du normal mapping à un modèle. On remarque le détail de la texture basée sur un modèle fait de nombreux polygones, appliquée sur un modèle réduit à peu de polygones (constater le sommer du crane.)

A droite, le bump mapping et parallax mapping appliqué à un *decal* dans le jeu F.E.A.R. (Lithtech Jupiter Ex, 2005) montrant les limites de cette technique lorsqu'elle doit être appliquée aux bordures d'un polygone.

Exemple de surface traîtée avec shader models, bump mapping et parallax mapping sur le jeu Stalker Call of Pripyat (X-Ray Engine 1.6, 2009)

pcgameshardware.com

Le futur du parallax mapping : la tesselation ou le displacement mapping (deux techniques différentes) qui ont encore du mal à s'imposer à cause de l'augmentation du nombre de vertex obligatoires pour gérer ces techniques.

HDR

(high dynamic range imaging, imagerie à grande gamme dynamique)

PRINCIPE

[22] La technique de HDR repose sur un stockage supplémentaire par pixel (on travaille sur la projection à l'écran) des informations de couleur. Le principe repose sur une variation de l'exposition afin d'extraire le plus d'information possible des zones trop ou pas assez exposées de l'image d'origine. L'image est reconstituée à partir des sous-images d'exposition différente. Le type de reconstitution fait l'objet d'une décision sur la sélection des sous-images.

APPLICATION

Le HDR est appliqué dans le milieu de la photographie depuis les années 1980, mais son application en temps réel s'est faite en Octobre 2005 avec une amélioration du moteur Source de Valve dans un niveau jouable servant de démonstration technologie : Lost Coast.

[23] Le HDR de Valve est caractérisé par de nombreuses techniques. Parmi celles-ci :

- Le **bloom** simule la surexposition de la lumière sur une zone de lumière dans une ambiance plus sombre que celle-ci.
- Le **HDR Skybox** enregistre plusieurs sous-images de différentes expositions du ciel afin que celui-ci s'adapte à l'ambiance lumineuse de la zone où se trouve la caméra.

- Le **HDR light map** applique l'ambiance lumineuse à des surfaces qui ne reçoivent pas directement la lumière, on s'approche de la réflexion de la lumière dont on parlait au début de ce dossier.

Ces trois captures montrent trois différentes expositions lumineuses. Celles-ci sont choisies en fonction de l'ambiance lumineuse dans laquelle se trouve la caméra. La conséquence est un effet visuel proche de la

réaction de l'oeil humain lorsqu'il passe d'une zone à une autre avec des ambiances lumineuses très différentes. Par exemple, le fait d'être dans un tunnel sombre et de passer d'un coup à un extérieur illuminé va faire conserver quelques instants l'exposition à laquelle l'œil s'est réglé pour mieux voir dans le tunnel sombre, l'éblouissant alors lorsqu'il sort à l'extérieur, avant que l'exposition soit amoindrie afin de s'habituer à la lumière extérieur.

EXEMPLE

(s2.n4g.com)

Une capture d'écran du jeu Crysis (Cry-Engine 2, 2007) alliant **shader models** (l'eau), **normal mapping** (les rochers dont les bords mettent en évidence la réduction du nombre de polygones), **bump mapping et parralax mapping** (la surface de galets) et **HDR** (différence ombre/lumière entre la zone sous les feuilles et le ciel abondamment éclairé, *bloom* sur les bords du bateau et sur les rochers)

Screen space ambiant occlusion

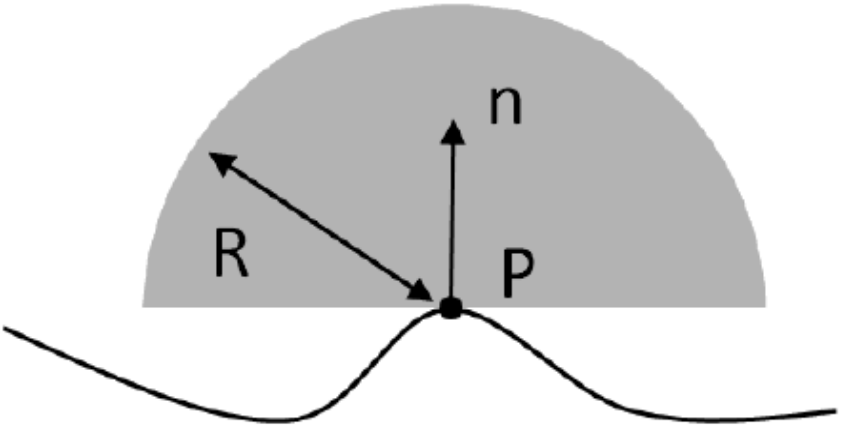
(SSAO)

PRINCIPE

On essaie de plus en plus de résoudre notre problème initial de l'éclairage, selon lequel la lumière diffusée sur une surface ne provient pas d'un seul point mais est émise depuis toutes les surfaces pouvant faire rebondir la lumière.

Le *screen space ambiant occlusion* essaie de simuler une solution, en se basant sur un hémisphère selon le vecteur normal à chaque point d'une surface (les points étant décrits selon une discrétisation de l'espace). Le rayon des hémisphères est défini par un radius donné.

Le principe est que toutes les surfaces qui sont contenues dans un hémisphère interviennent dans l'illumination du point concerné par l'hémisphère, en plus de la lumière diffuse.



Coupe 2D d'un hémisphère de rayon R appliqué au point P selon son vecteur normal n.

APPLICATION

[24] Dans la pratique, la complexité des calculs est bien moins importante que la véritable gestion de rebond des rayons de lumière, mais toujours trop importante pour les machines actuelles dû au phénomène de lancer de rayon au sein d'un hémisphère pour déterminer les surfaces qui y sont inclues. Nvidia utilise donc une technique d'approximation de l'espace et une gestion en 2D des lancers de rayon au sein de l'hémisphère.

EXEMPLE

Le SSAO est une technique très récente qui est appliquée dans peu de jeux à ce jour (environ 50), car les cartes graphiques qui peuvent le gérer de façon efficace sont des cartes haut-de-gamme.

A gauche, Mafia II (Illusion Engine, Aout 2010) montre une implémentation en temps réel du SSAO avec les zones ombrées autour du personnage qui ne sont pas des ombres projetées mais bien le résultat du SSAO.

A droite, Half life 2 (Source Engine, 2004) montre les premières techniques d'Ambiant Occlusion appliquées lors du pre-processing de la carte. Le résultat est relativement similaire à celui de Mafia II qui est appliqué en temps réel.

DEMI-TOUR

On notera que cette partie du dossier s'est particulièrement penchée sur la gestion de la lumière et sur le rendu des textures. C'est encore aujourd'hui sujet à de nombreuses améliorations, comme on l'a vu avec la tesselation ou des techniques plus avancées que le SSAO, qui nous rapprochent petit à petit de la gestion totale de la diffusion et l'émission de la lumière.

La dernière partie de ce dossier présente brièvement quelques techniques supplémentaires appliquées aux moteurs 3D dans d'autres domaines que l'éclairage ou le rendu des textures. L'objectif n'est pas d'en expliquer le fonctionnement global comme on l'a fait précédemment, ni de faire une énumération de ces techniques. Il s'agit de se pencher sur les motivations et les réponses aux problèmes posés aux développeurs de moteurs de jeux-vidéo.

III – QUELQUES TECHNIQUES EXPERIMENTALES

1. Un problème s'est posé quand aux distances de vue en augmentation constante. Une solution a été la modification du nombre de polygone dynamiquement. Plus un objet est loin, moins il est constitué de polygones. Pour cela, on enregistre la distance de l'objet à la caméra et on procède à une technique de fading lors de la transition d'un état de X à un état de Y polygones constituant l'objet.

Une extension de la technique inclut la réduction du nombre de texels constituant la texture (diminution de la résolution des textures), mais aussi la diminution de la résolution des ombres, des shaders etc.

Assassin's Creed (moteur Anvil, 2011) gère très efficacement cette technique en affichant une ville complète. Les objets proches sont détaillés, tandis que les objets distants sont peu détaillés.

On peut étendre la technique à un affichage sous forme de sprites 2D des modèles lorsqu'ils sont distants. C'est en partie la technique utilisée par The Creative Assembly dans leur moteur dont ils gardent le secret de fabrication :

Chaque soldat d'Empire : Total War (moteur Warscape, 2009) est par défaut affiché en trois dimensions. Mais lorsqu'un soldat est au delà d'une distance définie de la caméra, son modèle 3D n'est plus affiché, laissant place à un sprite 2D à N faces (cf la technique d'affichage des sprites 2D à N face en première partie de ce dossier)

2. L'engouement pour les décors forestiers a posé un problème sur la complexité d'affichage de nombreux éléments liés à la végétation. Les arbres et autres éléments naturels sont constitués de nombreux polygones et de très nombreux sprites 2D qui font en général office de feuillages. Une solution a été développée par Speed Tree incluant un moteur qui génère e façon procédurale de la végétation et la gère avec des algorithmes efficaces. Le moteur peut être greffé à un moteur graphique. [25]

Oblivion (moteur Gamebryo, 2006) utilise le moteur Speed Tree afin d'afficher de très grandes étendues de forêts très détaillées. Le moteur se permet une projection des ombres des feuilles sur le sol et sur les brins d'herbe à l'aide d'une technique dite d'o*mbres canopées*.

D'autres jeux conçoivent leur propre moteur de rendu pour la gestion des forêts, comme The Hunter (moteur Avalanche 2.0, 2009)

3. On a voulu afficher les rayons de soleils qui apparaissent dans des zones poussiéreuses ou remplies de particules qui révèlent la lumière. La solution se nomme *sun shafts* ou *god rays* et repose sur la technique de *lumière volumétrique*. La technique repose sur le *ray casting* et est mise en évidence par une modification du contraste pour tout ce qui est affiché et qui passe à travers un volume de lumière. [26]

Les sun-shafts de Stalker Call of Pripyat (xrEngine, 2010) ont une gestion en temps réel, obligée par les mouvements de la végétation et par le cycle jour-nuit.

4. On a vu la volonté de gérer les propriétés physiques des objets : attraction des corps, physique des fluides, mais aussi des comportements tout simples tels que les chutes, rebonds etc. Deux grosses sociétés proposent aujourd'hui une solution sous forme de moteur à greffer aux moteurs de jeux : le moteur Havok et le moteur physX. Ce dernier a la particularité d'exploiter l'architecture de la carte graphique, car il a été conçu par Nvidia, constructeur de cartes graphiques. Aujourd'hui, des démos technologiques ont été présentées à l'E3 de Los Angeles révélant une hypothétique gestion de la physique des fluides en temps réel dans les jeux-video. Concrètement, les machines actuelles ont des difficultés à afficher avec une bonne fréquence d'affichage des particules avec le moteur PhysX, et Havok continue de s'imposer comme le moteur physique le plus performant, mais ne propose pas une physique aussi poussée.

Garry's mod est une modification pour les jeux du moteur Source et permet de réaliser des expériences avec son moteur physique : Havok. Il est possible de modifier les propriétés (poids, élasticité etc.), de relier des éléments selon diverses liaisons (corde, axe, slider, attachement fixe etc.) ou encore de programmer des circuits électroniques avec des portes logiques afin de créer des robots virtuels.

Le moteur physique concurrent de Havok, le PhysX de Nvidia, appliqué à quelques jeux tels que, dans cet exemple, Mirror's Edge (Unreal Engine 3, 2008)

CONCLUSION

Nous avons découvert les origines des moteurs 3D temps réel avec les jeux des années 90. De nombreux problèmes se posaient à l'époque à cause de la puissance des machines. Il aura fallu attendre les premières cartes graphiques pour commencer le développement d'artifices graphiques afin de toujours tendre vers une simulation graphique la plus crédible possible de la réalité. Enfin, ce dossier s'est terminé par la découverte de techniques et de moteurs inclassables dans les deux premières parties, et qui pourraient faire l'objet de futurs projets multimédia. (Implémentation, étude de la technique employée etc.)

Il existe de nombreux moyens pratiques de découvrir le monde de la 3D temps réel à condition de disposer du matériel nécessaire. C'est un sujet extrêmement vaste et un dossier comme celui-ci ne pouvait en présenter qu'une infime partie.

BIBLIOGRAPHIE

[1] James FOLEY. *Computer graphics: principles and practice*. 2004.

[2] Andrew GLASSNER. *An Introduction to ray tracing*. 2002.

[3] Lode VANDEVENNE. *Lode's Computer Graphics Tutorial Raycasting*. 2007.

http://lodev.org/cgtutor/raycasting.html
page consultée le 10 Avril 2011

[4] *Doom engine code review*. 2010

http://www.fabiensanglard.net/doomIphone/doomClassicRenderer.php
page consultée le 9 Avril 2011

[5] Matthew MASTRACCI. *BSP-Tree Primer*. 1998.

http://grack.com/downloads/graphics/bsp/bsp.txt

[6] JOHNA. *Packface culling*. 2006.

http://wiki.processing.org/w/Backface_culling

[7] Ken SILVERMAN. *Ken Silverman's Build Engine Page*.

http://advsys.net/ken/build.htm
page consultée le 18 Avril 2011

[8] Ken SILVERMAN. *Ken Silverman's Voxlap Page*.

http://advsys.net/ken/voxlap.htm
page consultée le 18 Avril 2011

[9] Arie KAUFMAN Daniel COHEN Roni YAGEL. *Fundamentals of Voxelization*. 1993.

http://www.cs.sunysb.edu/~vislab/projects/volume/Papers/Voxel/index.html
page consultée le 18 Avril 2011

[10] Michael ABRASH. *Quake's lighting model*. 2000.

http://downloads.gamedev.net/pdf/gpbb/gpbb68.pdf

[11] Images d'exemple de l'article Gouraud Shading de wikipedia

http://en.wikipedia.org/wiki/Gouraud_shading
page consultée le 19 Avril 2011

[12] Images d'exemple de l'article Lightmap de wikipedia

http://fr.wikipedia.org/wiki/Lightmap
page consultée le 19 Avril 2011

[13] *Vertex Shaders*. Nvidia.

http://www.nvidia.com/object/feature_vertexshader.html
page consultée le 3 Mai 2011

[14] *Shader Concepts*.

http://robotrenegade.com/q3map2/docs/shader_manual/shader-concepts.html
page consultée le 3 Mai 2011

[15] Images d'exemple de l'article Fast inverse square root de wikipedia

http://en.wikipedia.org/wiki/Fast_inverse_square_root
page consultée le 3 Mai 2011

[16] Tomer MARGOLIN. *Magical Square Root Implementation In Quake III*. 2005.

http://www.codemaestro.com/reviews/9

[17] *Pixel Shaders*. Nvidia.

http://www.nvidia.com/object/feature_pixelshader.html
page consultée le 3 Mai 2011

[18] *Simple Bumpmapping*

http://www.paulsprojects.net/tutorials/simplebump/simplebump.html
page consultée le 3 Mai 2011

[19] Images d'exemple de l'article Normal mapping de wikipedia

http://fr.wikipedia.org/wiki/Normal_mapping
page consultée le 3 Mai 2011

[20] Tomas MOLLER Eric HAINES Naty HOFFMAN. *Real-time rendering*. 2008. p.191

[21] *Per-Pixel Displacement Mapping with Distance Functions*. NVIDIA.

http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter08.html
page consultée le 4 Mai 2011

[22] Pierre-Henry MULLER. *Exposé sur la photographie HDR*. 2008.

http://www.photo-hdr.com/tutoriaux/expose-sur-la-photographie-hdr

[23] Geoff RICHARDS. Half Life 2: Lost Coast HDR overview. 2005.

http://www.bit-tech.net/gaming/pc/2005/06/14/hl2_hdr_overview/1

[24] Louis BAVOIL Michel SAINZ. *Screen space ambiant occlusion*. Nvidia. 2008.

[25] *Speed tree Overview*. SpeedTree.

http://www.speedtree.com/apps
page consultée le 6 Mai 2011

[26] *Volemtric Light*. Nvidia.

http://http.developer.nvidia.com/GPUGems3/gpugems3_ch13.html
page consultée le 6 Mai 2011

GLOSSAIRE

FOV : Le Field Of View (Champs de vision) est la valeur donnée l'ouverture de l'angle de vision qui permet de voir une partie du monde.

Sprite : Une entité repérée ponctuellement dans l'espace, en principe en partie transparente, qui est placée dans l'univers virtuel.

Vertex : (vertices au pluriel en anglais) "couture entre polygones". Il s'agit du coin d'intersection de plusieurs polygones.

Texel : Texture Element. L'unité de texture. Il s'agit du plus petit élément composant une texture sur une surface. Cet élément peut être constitué de plusieurs pixels.

RVB : triplet correspondant aux quantités de rouge, vert et bleu d'une couleur.

Mesh : un mesh est un volume constitué de vertex qui sont les sommets de polygones convexes tels que des triangles ou certains quadrilatères.

Depth buffer : la technique actuelle pour l'affichage "au pixel près". Chaque pixel contient une information sur sa distance à la caméra.