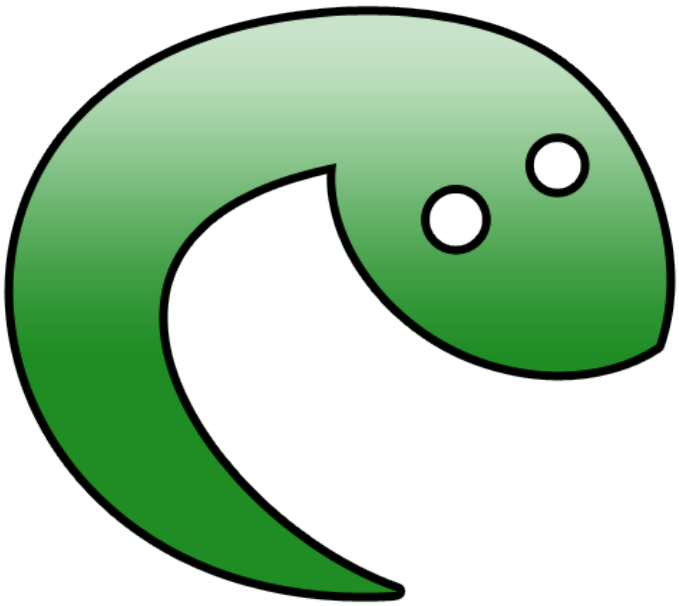


Triton

Guide mathématique

- ***Berthoux Vincent***
- ***Ferrari Romain***
- ***Legendre Olivier***
- ***Potiquet Fabien***

Table des matières

I Notre monde 3D

Notre logiciel utilise la librairie de rendu OpenGL, ce qui nous impose certaines contraintes : nous ne pouvons afficher que des triangles, triangles qui seront projetés sur l'écran par la carte graphique. Cette approche a été utilisée du fait de sa rapidité et de l'omniprésence des cartes 3D aujourd'hui.

Cette approche nous oblige à utiliser des points en coordonnées cartésiennes. OpenGL distingue 3 espaces importants :

- l'espace monde/vue (modelview), qui représente notre monde.
- l'espace oeil ou caméra.
- l'espace écran représente nos points projetés sur l'écran et prêts à être affichés.

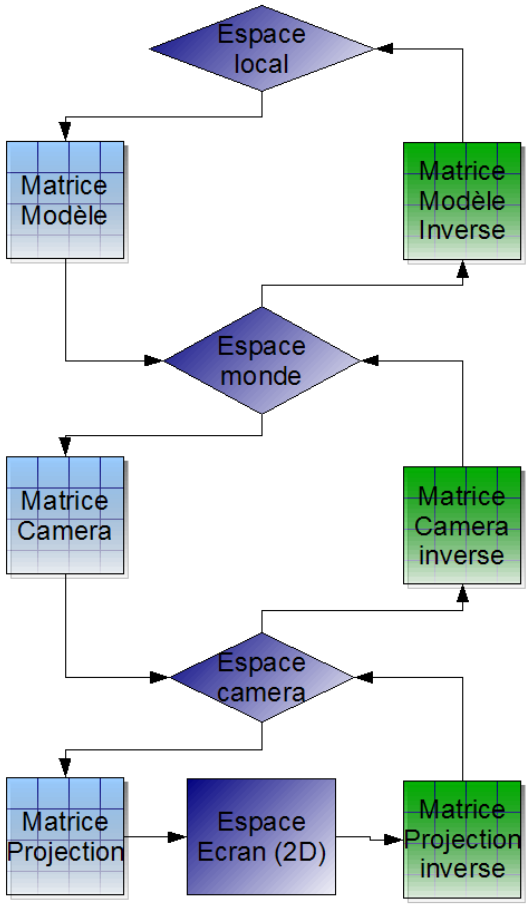
Pour passer d'un espace à un autre, des matrices de passages sont utilisées :

$B = P.A$

$B, A \in \mathbb{R}_4^2$ nos points sont exprimés dans des coordonnées homogènes, comme expliqué ci-après.

$P \in M(\mathbb{R})_4$

Nous avons étendu cette logique en séparant l'espace monde de l'espace local, propre à chaque objet de notre monde. Nous avons donc, au final, 4 espaces distincts avec 6 matrices pour passer d'un espace à un autre comme décrit dans le schéma suivant.



II Primitives

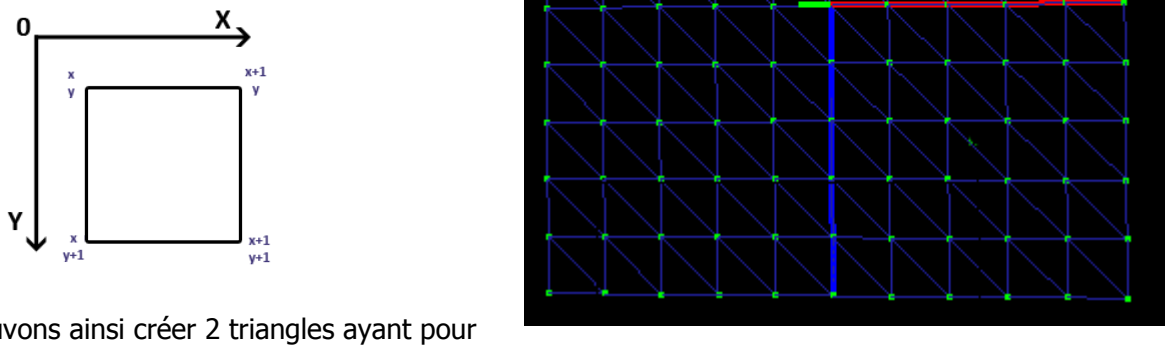
Pour créer nos primitives, nous partons d'équations paramétriques de figures géométriques connues, tels que la sphère et le cône. Nous les échantillonnons : nous prenons les pas Δu et Δv , les plus petits possibles, telles que $\begin{matrix} \forall m,n\in\mathbb{N},\forall a,b,c,d\in\mathbb{R} \\ a<b \\ c<d \end{matrix} soient\ I=[a,b] J=[c,d]$.

$$\left\{\begin{matrix} \Delta u*n=b-a \\ \Delta v*m=d-c \end{matrix}\right\}$$

Ainsi en utilisant le pseudo-algorithme suivant :

```
Pour i <-1 à n Faire
        Pour j <-1 à m faire
                Ajouter point au tableau correspondant à l'équation paramétrique avec les valeurs
                 i*Δu   et   j*Δv   passées en paramètre
        fin Pour
fin Pour
```

Nous obtenons donc une image correspondant à une sorte de tableau en 2 dimensions de points. Ceci est juste l'idée qu'en a le programme, nos points sont répartis selon l'équation paramétrique de la primitive étudiée. Lors d'une deuxième étape, nous devons nous charger de relier les points entre eux afin de créer des faces (ici, des triangles). Pour cela rien de plus simple grâce à notre disposition en tableau :



Nous pouvons ainsi créer 2 triangles ayant pour sommets : < x,y | x+1,y | x+1, y+1 > et <x,y | x+1,y+1 | x,y+1> . L'adresse d'un point dans le tableau est donnée par la formule : $\forall x,y\in\mathbb{N}\ f(x,y)=x+y*m$, m étant la même variable que celle utilisée dans le pseudo-algorithme. Si nous partons du principe que le tableau « boucle » sur lui-même alors nous pouvons fermer nos primitives échantillonnées.

Ci dessous, la liste des équations paramétriques utilisées pour les primitives

1. Sphère

$$\left.\begin{array}{c} x=r*\sin(\theta)*\sin(\phi) \\ y=r*\sin(\theta)*\cos(\phi) \\ z=r*\cos(\theta) \end{array}\right\} \wedge \begin{array}{c} \theta\in[0;\pi] \\ \phi\in[-\pi;\pi] \end{array}$$

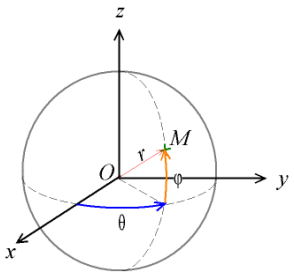


Illustration 1: image provenant de la wikipedia

r étant le rayon de la sphère.

2. Tore

$$\left.\begin{array}{c} x=(c+a*\cos(v))*\cos(u) \\ y=(c+a*\cos(v))*\sin(u) \\ z=a*\sin(v) \end{array}\right\} \wedge u,v\in[0;2\pi]$$

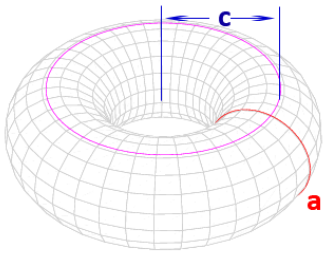
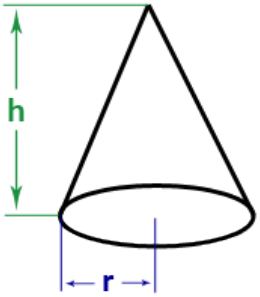


Illustration 2: image provenant de la Wikipedia

a étant le rayon du disque éloigné et c le rayon de rotation de ce même disque.

3. Cône

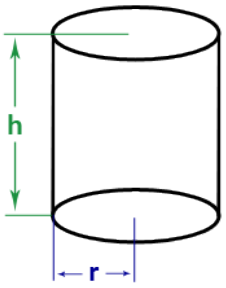
$$\left.\begin{array}{c} x=\frac{h-u}{h}*r*\cos(\theta) \\ y=\frac{h-u}{h}*r*\sin(\theta) \\ z=u \end{array}\right\} \wedge \begin{array}{c} \theta\in[0;2\pi] \\ u\in[0;h] \end{array}$$



h étant la hauteur du cône et r le rayon du disque générateur.

4. Tube/Cylindre

$$\left.\begin{array}{c} x=r*\cos(\theta) \\ y=r*\sin(\theta) \\ z=z \end{array}\right\} \wedge \begin{array}{c} z\in[0;h] \\ \theta\in[0;2\pi] \end{array}$$



h étant la hauteur du cylindre et r son rayon.

5. Harmonie sphérique

$$\forall\, m_0, m_1, m_2, m_3, m_4, m_5, m_6, m_7 \in \mathbb{N}$$

$$r=\sin(m_0*\phi)^{m_1}+\cos(m_2*\phi)^{m_3}+\sin(m_4*\theta)^{m_5}+\cos(m_6*\theta)^{m_7}$$

r étant une composante de position en coordonnées sphériques. Nous convertissons ces coordonnées sphériques en coordonnées cartésiennes en réutilisant l'équation paramétrique de la sphère avec (r,ϕ,θ) . Plus d'information :

http://astronomy.swin.edu.au/~pbourke/surfaces/sphericalh/

6. Super Surface

$$r(x)=\left[\left|\frac{1}{a}*\cos\left(\frac{m.x}{4}\right)\right|^{n2}+\left|\frac{1}{b}*\sin\left(\frac{m.x}{4}\right)\right|^{n3}\right]^{\frac{-1}{n1}}$$

$$\left\{\begin{array}{c} x=r(\theta)*\cos(\theta)*r(\phi)*\cos(\phi) \\ y=r(\theta)*\sin(\theta)*r(\phi)*\cos(\phi) \\ z=r(\phi)*\sin(\phi) \end{array}\right\}\wedge \begin{array}{c} \phi\in[-\pi;\frac{\pi}{2}] \\ \theta\in[-\pi;\pi] \\ \forall a,b,m,n1,n2,n3\in\mathbb{R}^6-\{0\} \end{array}$$

plus d'info sur les Super Surface : http://astronomy.swin.edu.au/~pbourke/surfaces/supershape3d/

III Sélections

Afin de sélectionner des parties d'objet de notre monde 3D nous utilisons différents algorithmes de sélections. Nous pouvons sélectionner 2 entités différentes : des points et des triangles. Dans certains cas nous simplifions les tests en stipulant simplement qu'un triangle est sélectionné si tous ses points sont sélectionnés.

1. Sélection aléatoire

Cet algorithme sélectionne aléatoirement des points ou des triangles de notre objet. Utile lors de l'utilisation, cette méthode ne présente pas un grand intérêt outre mesure.

2. Sélection par « Picking »

Cette méthode de sélection est l'une des plus complexe implémentée dans Triton. A partir d'un simple clic sur l'écran, cette méthode est capable de nous dire précisément quels sont les triangles, sous le pointeur de la souris, que nous pouvons donc sélectionner.

La première étape est de récupérer une droite qui constitue l'ensemble des points sous le pointeur de notre souris. Pour cela nous prenons les coordonnées du clic :

$p_{click}\begin{pmatrix} x \\ y \end{pmatrix}$ en coordonnées écran, et donc dépendant de la résolution. Afin de pouvoir récupérer notre droite dans le monde 3D nous introduisons 2 points :

$$P_{vect}=\begin{pmatrix} \frac{x}{résolutionHorizontale} \\ \frac{y}{résolutionVerticale} \\ 0 \end{pmatrix}, P_{point}=\begin{pmatrix} \frac{x}{résolutionHorizontale} \\ \frac{y}{résolutionVerticale} \\ 1 \end{pmatrix}$$ En ayant ces 2 points nous pouvons donc trouver une droite d'origine u et de vecteur directeur v tel que :

$$u=M^{-1}_{projection}.P_{point}$$
$$v=M^{-1}_{projection}.P_{vect}$$ De cette manière nous avons donc une droite affine en espace monde. De la même manière nous pouvons passer cette droite en espace local en multipliant la matrice modèle par les composants de la droite. Avec la droite en espace local nous sommes prêts à chercher les intersections entre la droite et les triangles. Ce test se déroule en 2 parties : le test d'intersection entre la droite et le plan dans lequel est inclus le triangle puis un test pour savoir si le point d'intersection est dans ce même triangle.

a. Intersection droite/Plan

Soit 4 Vecteurs :

$$\vec{n}=\begin{pmatrix}a\\b\\c\end{pmatrix}, A=\begin{pmatrix}x\\y\\z\end{pmatrix}, \vec{V}=\begin{pmatrix}x'\\y'\\z'\end{pmatrix}, O=\begin{pmatrix}x_0\\y_0\\z_0\end{pmatrix}, I=\begin{pmatrix}x''\\y''\\z''\end{pmatrix}$$

$$\vec{n}, A, \vec{V}, O, I \in \mathbb{R}^3$$

$\vec{n}$ étant le vecteur normal du plan P, O un point de ce même plan, A un point de l'espace et $\vec{V}$ un vecteur de l'espace. est le point d'intersection entre le plan P et la droite D. Le plan P peut être exprimé par l'équation cartésienne du plan : $P=a.x+b.y+c.z+d=0$

De même, la droite D peut être exprimée par l'équation paramétrique suivante à l'aide du point A servant d'origine et de son vecteur directeur V :

$D=\begin{Bmatrix}X=t.x'+x\\Y=t.y'+y\\Z=t.z'+z\end{Bmatrix}$ L'intersection de la droite et du plan doit ainsi répondre à :

$a.(t.x'+x)+b.(t.y'+y)+c.(t.z')+d=0$ Nous cherchons donc à extraire le t de l'équation afin de pouvoir le ré entrer dans l'équation paramétrique de la droite et obtenir les coordonnées du point d'intersection :

$$a.t.x'+a.x+b.t.y'+b.y+c.t.z'.c.z+d=0$$
$$t.(a.x'+b.y'+c.z')+(a.x+b.y+c.z)+d=0$$

Nous pouvons ici introduire 2 produits scalaires :

$$t.\vec{n}.\vec{V}+\vec{n}.A+d=0$$
$$t.\vec{n}.\vec{V}=-d-\vec{n}.A$$
$$t=\frac{-d-\vec{n}.A}{\vec{n}.\vec{V}}$$

d n'étant pas fourni par défaut, nous devons le calculer par nous même en repartant de l'équation cartésienne sachant que O est un point du plan, nous arrivons donc à :

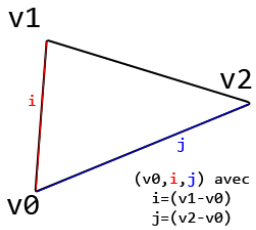
$$a.x_0+b.y_0+c.z_0=-d$$
$$\vec{n}.O=-d$$

Nous pouvons donc maintenant réinjecter d dans notre équation précédente et trouver notre point d'intersection :

$$t=\frac{\vec{n}.O-\vec{n}.A}{\vec{n}.\vec{V}}=\frac{\vec{n}.(O-A)}{\vec{n}.\vec{V}}$$
$$I=t.\vec{V}+A$$

b. Test intersection droite/triangle

Soit un triangle $v_{0,}v_{1,}v_2 \, avec \, v_0=\begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix}, v_1=\begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix}, v_2=\begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix}$, introduisons 2 vecteurs :



$$\vec{i}=\begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix}=\begin{pmatrix} x_1-x_0 \\ y_1-y_0 \\ z_1-z_0 \end{pmatrix}=\overrightarrow{v_0 v_1}$$

$$\vec{j}=\begin{pmatrix} x_j \\ y_j \\ z_j \end{pmatrix}=\begin{pmatrix} x_2-x_0 \\ y_2-y_0 \\ z_2-z_0 \end{pmatrix}=\overrightarrow{v_0 v_2}$$

grâce à ces 2 vecteurs nous pouvons trouver $\vec{n}=i \wedge j$ et ainsi trouver le point d'intersection du plan de vecteur normal n et de point v0. Ce point d'intersection est nommé $P=\begin{pmatrix} x_p \\ y_p \\ z_p \end{pmatrix}$ que nous pouvons exprimer en fonction de i et j tel que :

$P=s.\vec{i}+t.\vec{j}+v_0, s, t \in \mathbb{R}^2$. Nous introduisons le vecteur $\vec{w}=\begin{pmatrix} x_w \\ y_w \\ z_w \end{pmatrix}=\overrightarrow{v_0 P}$ qui nous permet de simplifier l'équation précédente : $\vec{w}=s.\vec{i}+t.\vec{j}$ Pour que P soit dans le triangle, s et t doivent satisfaire les conditions suivantes :

$$\begin{matrix} t \geq 0 \\ s \geq 0 \\ s+t \leq 1 \end{matrix}$$

Afin d'extraire s & t nous introduisons 2 nouveaux vecteurs : $\begin{matrix} i_n=\vec{n} \wedge \vec{i} \\ j_n=\vec{n} \wedge \vec{j} \end{matrix}$

nous pouvons donc maintenant calculer s et t :

$$\left\{ \begin{matrix} \vec{w}.j_n=s.\vec{i}.j_n+t.\vec{j}.j_n \\ w.\vec{j}_n=s.\vec{i}.j_n \\ s=\dfrac{\vec{w}.j_n}{\vec{i}.j_n} \end{matrix} \right\} et \left\{ \begin{matrix} \vec{w}.i_n=s.\vec{i}.i_n+t.\vec{j}.i_n \\ w.\vec{i}_n=t.\vec{j}.i_n \\ t=\dfrac{\vec{w}.i_n}{\vec{j}.i_n} \end{matrix} \right\}$$

3. La sélection par « Frustrum »

Le Picking est une technique de sélection très pratique pour sélectionner individuellement des faces, mais dès qu'il s'agit de sélectionner un grand nombre de faces, la tâche peut se révéler fastidieuse. Nous avons mis en place une technique qui permet de sélectionner tous les points situés dans un cadre dessiné à l'écran. Ainsi sont sélectionnés tous les éléments situés dans un pavé ou dans une pyramide dont 2 extrémité es seraient situées à l'infini. Avec cette technique, pour qu'un triangle soit sélectionné, ses 3 sommets doivent être sélectionnés.

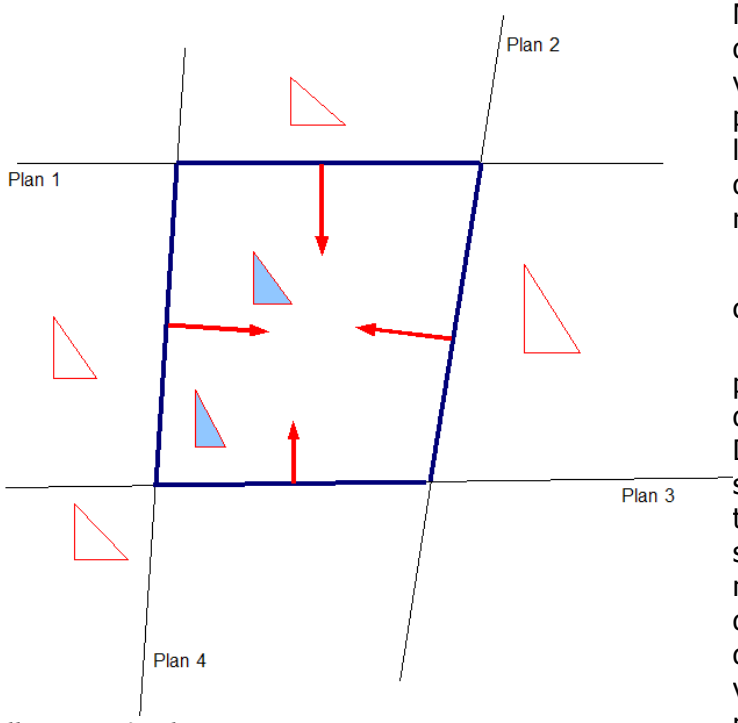


Illustration 3: Plan en coupe

Nous avons 4 plans, ici représentés par des droites. Les flèches rouges représentent les vecteurs normaux rentrants de ces mêmes plans. Pour déterminer si un objet est dans le cadre bleu, nous faisons des tests de demi-espace. Un test de demi-espace est réussi si :

$a.x+b.y+c.z+d>0$ a,b,c étant les coordonnées du vecteur normal du plan et d une constante donnée pour chaque plan. Chaque point doit réussir les 4 tests de demi-espace afin d'être sélectionné. Dans le schéma, les 2 triangles bleus sont sélectionnés car leur 3 points ont réussi les tests de demi espace. De plus pour sélectionner uniquement les triangles qui nous font face, nous pouvons rajouter un dernier test afin de déterminer l'orientation de la face. Ce test s'effectue grâce à un vecteur $\vec{u}$ symbolisant la direction de notre regard et le vecteur normal du triangle (nous n'en utilisons qu'un seul), nous souhaitons que le vecteur normal du triangle soit orienté vers nous, donc pour que le test soit validé, il doit vérifier $\vec{n}.\vec{u}<0$.

Le test en lui même est plutôt simple. La difficulté réside dans la détermination des équations des 4 plans de test. Sachant que nous voulons simplifier la tâche de l'utilisateur, nous lui demandons de fournir 2 points en cliquant sur l'écran. Grâce à un simple « Drag'n'Drop » nous récupérons ces 2 points, Nous calculons ensuite la coordonnée de 2 autres points afin d'obtenir un rectangle :

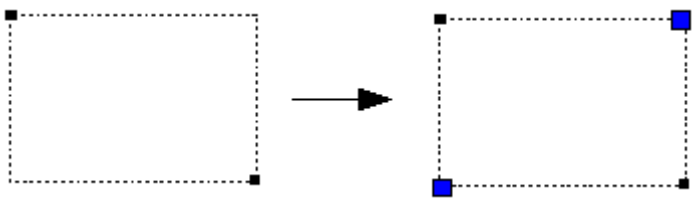


Illustration 4: 2 clicks

Avec nos 4 points sur l'écran, nous pouvons, comme pour le Picking, récupérer 4 droites affines dans l'espace Monde et calculer les coordonnées des plans de la manière suivante :

$$\vec{v}_{1,}\ \vec{v}_2\ les\ 2\ vecteurs\ directeurs\ de\ 2\ droites$$
$$P_1 \leq point\ d'\ origine\ de\ la\ droite\ ayant\ pour\ vecteur\ directeur\ \vec{v}_1$$
$$\vec{n}_1 \leq vecteur\ normal\ rentrant\ du\ plan\ à\ calculer.$$
$$n_1 = \vec{v}_1 \wedge \vec{v}_{2,}\ d_1 = p_1 . n_1$$

ou v1 et v2 sont coplanaires (donc typiquement en reprenant l'illustration 4, un rayon « noir » et un rayon « bleu » à chaque fois).

IV Transformations

Une fois nos points ou nos triangles sélectionnés sur une primitive de base, il est intéressant de pouvoir en modifier la position ou d'effectuer diverses opérations.

1. Transformation simple

La première transformation est comme son nom l'indique, simple. Elle consiste à appliquer à chaque point une transformation. La transformation est une combinaison d'homotétie, de rotation et de translation. Nous créons une matrice représentant ces 3 opérations puis nous multiplions cette matrice par chaque point pour obtenir la nouvelle position du point :

$$M_{homothétie} = \begin{pmatrix} x_s & 0 & 0 & 0 \\ 0 & y_s & 0 & 0 \\ 0 & 0 & z_s & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

$$M_{Translation} = \begin{pmatrix} 1 & 0 & 0 & x_t \\ 0 & 1 & 0 & y_t \\ 0 & 0 & 1 & z_t \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

$$M_{rotationX} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta_x) & -\sin(\theta_x) & 0 \\ 0 & \sin(\theta_x) & \cos(\theta_x) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$M_{rotationY} = \begin{pmatrix} \cos(\theta_y) & 0 & -\sin(\theta_y) & 0 \\ 0 & 0 & 0 & 0 \\ \sin(\theta_y) & 0 & \cos(\theta_y) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$M_{rotationZ} = \begin{pmatrix} \cos(\theta_z) & -\sin(\theta_z) & 0 & 0 \\ \sin(\theta_z) & \cos(\theta_z) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$M_{Transformation} = M_{Homothétie} * M_{Translation} * M_{rotationX} * M_{rotationY} * M_{rotationZ}$$

$$\forall Point \in \mathbb{R}^4, Point = M_{Transformation} * Point$$

2. Déplacement aléatoire

Afin de pouvoir donner un aspect « cabossé » aux objets modelés, nous avons choisi d'implémenter un manipulateur aléatoire de points qui fonctionne de 2 manières. Dans un mode purement aléatoire, et un autre ou le vecteur normal associé au point (ce vecteur étant calculé par la moyenne des vecteurs normaux de tous les triangles dont fait parti le point) est ajouté à un coefficient aléatoire prêt.

a. Mode aléatoire pur

Dans le mode aléatoire pur chaque point P est mis à jour selon la méthode suivante :

$\forall P\begin{pmatrix}x\\y\\z\end{pmatrix}, \begin{cases}x=x+rand()*c\\y=y+rand()*c\\z=z+rand()*c\end{cases}$ c étant un coefficient rentré par l'utilisateur et $rand()$ une fonction retournant un nombre pseudo-aléatoire.

$c\in\mathbb{R}, rand()\in[-1;1]$

b. *Mode aléatoire avec vecteur Normal*

$\forall P\begin{pmatrix}x\\y\\z\end{pmatrix}, \vec{n}\begin{pmatrix}a\\b\\c\end{pmatrix}, m\in\mathbb{R}$

$m=coefficient*rand()$ Ici $coefficient$ correspond au c de l'équation ci-dessus. Il est à noter $\begin{cases}x=x+a*m\\y=y+b*m\\z=z+c*m\end{cases}$ que le mode aléatoire par vecteur normal donne des résultats plus esthétiques, le mode aléatoire pur bougeant les points dans n'importe quelle direction, tous les triangles finissent par se chevaucher dans une sorte de « bouillie ». Le mode normal respecte davantage la topologie globale de l'objet (dans la limite d'un coefficient c relativement petit).

3. Pseudo-lissage

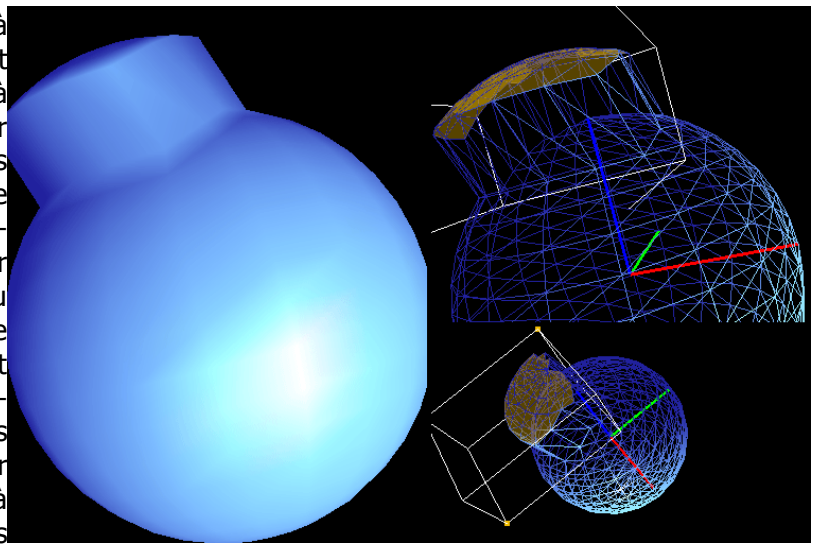
Il peut être utile, dans certaines circonstances, de pouvoir atténuer les arêtes trop raides de certains modèles. Pour cela nous utilisons une sorte de « floutage » qui pourrait être comparé à celui appliqué aux images. L'idée est de modifier la position du point par rapport à la position des points voisins. Nous ne cherchons pas ici un voisinage complexe, mais des points vérifiant la relation suivante :

$\forall P, P'\in\mathbb{R}^3, \{P\ \Re\ P' \equiv P \text{ et } P' \text{ sont des sommets différents d'un même triangle}\}$

R est une relation symétrique non réflective et non transitive. Le lissage s'effectue alors en appliquant la formule suivante :

$\forall p\in\mathbb{R}^3, \forall p'\in\mathbb{R}^3 tq\ p\ \Re\ p' : p=p+c.\sum\overrightarrow{pp'}$ c étant un coefficient d'atténuation rentré par l'utilisateur.

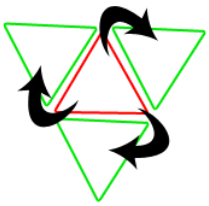
4. Extrusion



L'extrusion est une opération complexe à réaliser. Le principe de base est plutôt simple, mais beaucoup moin à implémenter. Sur un objet, pour extruder un sous-ensemble de triangles, nous devons tout d'abord trouver le bord de ce sous ensemble, "elever" ce sous-ensemble et creer des faces pour combler le vide. Le vide est comblé au niveau du bord, on crée des faces entre le non élévé, et l'élevé. Une image valant mieux qu'un long discour... En marront-moche nous discernons les faces sélectionnées. Nous pouvons voir qu'elles ont été surélevées par rapport à la sphère initiale. Des nouveaux triangles

font le lien entre les faces surélevées et la sphère originale.

a. *Trouver le Bord*



Comment trouver le bord? La méthode choisie consiste à trouver les relations entre les triangles. Pour chaque triangle (ici en rouge), on trouve ses triangles voisins en se basant sur les ID de Vertex. Une fois toutes nos relations entre les triangles trouvées, chaque triangle ayant une arrête sans relation place cette même arrête dans un stock. Nous trions ensuite les arrêtes de ce stock afin qu'elles se suivent, jusqu'à former un cercle. Une fois arrivé à cette étape, nous avons trouvés la liste des points à extruder.

b. *Elever les vertices*

là c'est simple, pour tous les points du bord, on crée un nouveau point à partir du précédent en ajoutant un vecteur qu'on appellera $\overrightarrow{extr}$ (vecteur d'extrusion, dont il sera discuté plus tard). On a donc un nouvel étage de points surélevés. Nous pouvons faire une extrusion en plusieurs étages, et donc répeter l'opération de surélèvement. A chaque étage nous pouvons en profiter pour faire une transformation comme mise à l'échelle/rotation pour les points, pour obtenir une extrusion « courbe » (si le nombre d'étage est important).

A cette étape, nous avons un problème. Nous avons élevé le bord, mais nous devons aussi élever les vertices qui ne sont pas au bord, mais pour lesquels nous n'avons pas besoin de creer de faces. Ces vertices sont appellés vertices invariants. A la différence de ceux du bord, on ne crée pas de nouveau point pour leur nouvel position, on les remplacent par leur nouvelle position. Ces vertex sont trouvés pendant la phase de recherche de bord, si ils sont dans une face et pas au bord, alors ils sont invariants.

c. *Créer les faces*

Pour créer des faces, on ajoute des triangles jouant sur la liste de points crées et les points « source ».

Dernière étape, il faut modifier les faces originales. On ne touche pas aux points invariants de ces faces, ils sont déjà bien placés, mais il faut changer les points appartenant au bord, sinon on aura des triangles acrochés aux fondations mais avec des points sur le tois, ce qui est emmerdant. Comme non dit plus tôt, on conserve la chaine de points constituant le bord, celle de la première itération (rappelons qu'il peut y avoir plein d'itérations, et qu'on jongle entre 2 chaines de bord pour creer les faces). Pour corriger on va chercher pour toutes les faces et tous ses points, si il est dans l'ensemble bord de la première itération, si on trouve sa position dans l'ensemble bord1, on prend l'ID à la même position dans l'ensemble bord de la dernière itération. De cette manière on aura toutes les faces avec des points ayant la même "itération".

d. *Discution autour du vecteur d'extrusion.*

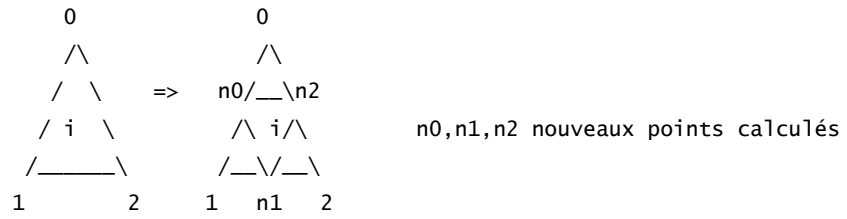
Le concept de vecteur d'extrusion a été ammené pour réhausser les points, mais ce vecteur d'extrusion peut avoir plusieurs forme.

Par défaut ce vecteur est le vecteur normal moyen de toutes les faces du sous-ensemble. Il peut aussi être le vecteur normal de chaque vertex. Cela peut être le vecteur position du vertex (sachant qu'on peut modifier le "centre" du repère) et pour finir, cela peut être le vecteur normal de chaque face. A noter que dans ce dernier cas il n'y a pas de vérifications des liens entre les faces, chaque face est extrudé indépendemment.

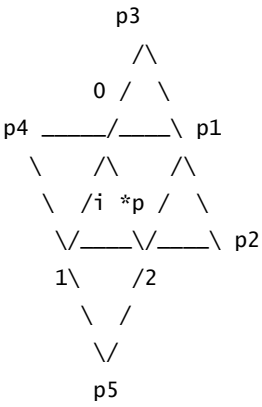
5. Subdivision

La subdivision sert à augmenter le nombre de triangles dans un objet en créant de nouveaux points et

faces. L'algorithme implémenté dans Triton est l'algorithme dit du Papillon fonctionnant sur des triangles (à la différence du Catmull-Clark qui fonctionne sur des quads). La subdivision découpe les triangles comme décrit dans le schéma ci-dessous :



i, la face initiale se retrouve au centre des nouvelles faces, qui incorpore les 3 nouveaux points. Pour subdiviser une face on regarde les triangles voisins pour pouvoir interpoler la position des nouveaux points.



pour calculer le point *p il nous faut les points de la face initiale (i) ainsi que les points p_1, p_2, p_3, p_4, p_5 . Donc avec tous ces point nous utilisons la formule « magique » :

$$p = \frac{1}{2}(F_0 + F_2) + 2w(F_1 + p_1) - w(p_2 + p_3 + p_4 + p_5), w \in \mathbb{R}$$

Une fois nos nouveaux points calculés, nous recréons 3 nouvelles faces. Afin d'éviter de créer un nombre trop important de points, nous marquons chaque face traitée : si nous tombons sur une face marqué lors de la recherche de voisins, nous récupérons directement le point déjà calculé. De plus, vu la face avant subdivision se trouve au milieu des 3 nouvelles faces, il est possible de déduire les nouvelles relations afin de refaire une passe de subdivision plus rapidement (nous évitons ainsi d'avoir à retrouver les relations entre les faces, opération très longue $\Theta(n^2)$).

6. Duplication

Pour chaque copie

$$Point_{dupliqué} = M_{duplication} * Point_{original}$$
$$M_{duplication} = M_{duplication} * M_{transformation}$$

fin Pour

V RayTracing

La méthode de rendu utilisée en mode d'édition est une méthode dite par « Rasterisation » : on se débrouille pour projeter nos triangles sur une surface 2D, puis ont relie nos différents points pour dessiner nos triangles (étape effectuée par OpenGL). Le RayTracing adopte une approche complètement opposée, on part de la camera pour aller voir se qui se passe dans le monde 3D. Pour chaque pixel qu'on souhaite rendre on lance un rayon dans le monde et on teste les objets du monde pour savoir si il existe une intersection. En cas d'intersection le pixel prend la couleur de l'objet touché par le rayon. Ceci est le principe global du rayTracing.

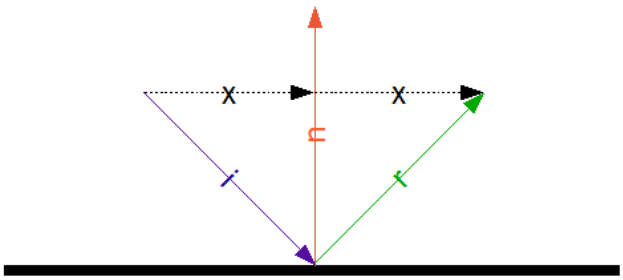
Triton implémente un rayTraceur naïf (comprendre : lent, de qualité douteuse) fonctionnant uniquement avec des triangles, Triton étant un modeleur polygonal, il semblait difficile d'inclure d'autres primitives non échantillonnées. Ce moteur de rendu gère les reflexions, la réfraction et les calculs d'illumination directe.

1. Reflection

Si un objet est réflechissant, nous lançons un rayon secondaire afin de connaître la couleur du rayon réfléchi. La determination de ce rayon s'effectue à l'aide de la formule suivante :

$$\vec{r} = \vec{n} + \vec{x}$$
$$\vec{x} = \vec{n} * \cos(-\vec{i}\,;\vec{n}) + \vec{v}$$
$$\vec{r} = \vec{n} + \vec{n} * \cos(-\vec{v}\,;\vec{n}) + \vec{v}$$
$$\vec{r} = \vec{i} - 2.\vec{n}(\vec{n}.\vec{i})$$

où $\vec{n}$ et le vecteur normal sortant du triangle atteint par le rayon $\vec{i}$ et $\vec{r}$ le rayon réfléchi.

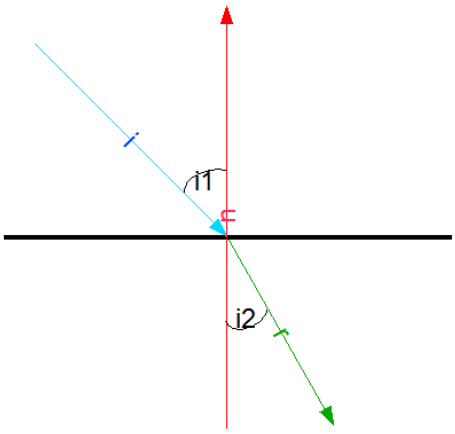


2. Refraction

Si un objet est transparent, nous envoyons un autre rayon secondaire pour connaître la couleur réfractée :

$$n1.sin(i1) = n2.sin(i2)$$
$$\sin(i2) = \frac{n1}{n2}\sin(i1)$$
$$\frac{n1}{n2}\vec{i} - [\cos(\theta) + \frac{n1}{n2}(\vec{n}.\vec{i})].\vec{n}$$
$$\cos(\theta) = \sqrt{1 - \left(\frac{n1}{n2}\right)^2 (1 - (\vec{n}.\vec{i})^2)}$$
$$\vec{r} = \vec{n} * \cos(\theta) - \vec{i}.(-\vec{n})$$

où n1 & n2 sont 2 coefficients de réfraction des milieux, $\vec{i}$ le rayon de vue incident, $\vec{r}$ le rayon réfracté, $\vec{n}$ le vecteur normal sortant du triangle considéré.



3. Calcul d'illumination directe

Pour calculer nos couleurs, nous encodons nos couleurs sous la forme suivante :

$$c = \begin{pmatrix} r \\ g \\ b \end{pmatrix}, r, g, b \in [0;1]$$

Nous distinguons 3 parties pour l'éclairage :

- l'éclairage ambiant.
- l'éclairage diffus.
- l'éclairage spéculaire.

Pour chaque lumière dans notre monde nous lançons un rayon du point d'intersection vers la lumière, si aucune objet n'est situé entre le point et la lumière, nous calculons la couleur de l'illumination grâce aux équations suivante :

$$c_{globalAmbiant} = Global_{ambiant} * Obj_{ambiant}$$

$$c_{ambiant} = lumière_{ambiant} . Obj_{ambiant}$$

$$\vec{l} = Position\ lumière - Position\ point$$

$$c_{diffuse} = Obj_{diffuse} * \vec{n} . \vec{l}$$

$$\vec{h} = Position\ Point\ de\ vue - Position\ point$$

$$c_{speculaire} = lumière_{speculaire} . Obj_{speculaire} * \vec{n} . \vec{h}^{Obj_{specularity}}$$

$$c_{illumination} = (c_{diffuse} + c_{ambiant}) * attenuation + c_{speculaire}$$

La couleur finale d'illumination étant la somme des couleurs trouvés grâce à chaque lampe.

$$c_{finalillumination} = c_{globalAmbiant} + \sum c_{illumination}$$

4. Mélange des couleurs

Une fois nos différentes couleurs calculées (illumination directe, réfraction, réflexion), nous mélangeons ces 3 couleurs de la manière suivante afin de produire la couleur finale du pixel qui sera sauvegardée :

$$c_{finale} = (c_{finalillumination} * (1-\alpha) + c_{refraction} * \alpha) * (1-r) + c_{reflexion} * r$$

α étant le coefficient de transparence (coefficient alpha) de l'objet toutché par le rayon, r le coefficient de réflexion de ce même objet. Cette formule a été trouvée de manière empirique et ne respecte nullement des critères physiques/réalistes. Néanmoin le rendu obtenu étant plutôt satisfaisant en vue de notre objectif, nous l'avons gardé.