Compte-rendu Projet Synthèse Image

BAZIN Thomas - DORT Jules - FERNANDES Alexandre -
GALLAY Jules - LARDI Nicolas - PERRIN Matthieu

Janvier 2019

Sommaire

1 Introduction

Pour ce projet, nous devons créer un programme capable de déformer un maillage 3d quelconque. Nous devons développer plusieurs primitives de déformations 3d qui pourront être appliquée à tour de rôle sur un objet. Ces primitives sont la compression, la rotation différentielle, le vortex et le pliage. Ces primitives pourront être appliquées interactivement sur l'objet, et nous devrons pouvoir charger un objet depuis un fichier.

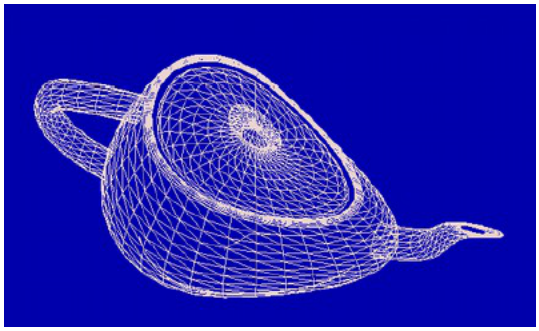
2 Méthode

Pour réaliser les déformations, nous avons opté pour la méthode de déformation globale. Cette déformation agit sur tous les points constituants l'objet. C'est une déformation locale qui déforme l'espace tangent du solide (ses vecteurs différentiels).

Nous avons choisi cette méthode de déformation car elle nous a semblé être la plus pratique pour faire nos tests. En repartant des exercices effectués en TP, nous avions déjà créé plusieurs représentations d'objets avec stockage de leurs points et c'était donc la solution la plus facile pour nous.

Les déformations sont effectuées dans le vertex shader grâce à des fonctions qui sont appelées juste avant l'affichage. Ainsi nous n'impactons pas la représentation initiale de l'objet dans le programme principal, et nous n'avons pas à stocker l'emplacement des points de l'objet déformé ce qui évite donc la création et la destruction de grands tableaux à chaque nouvelle opération de déformation.

2.1 Compression



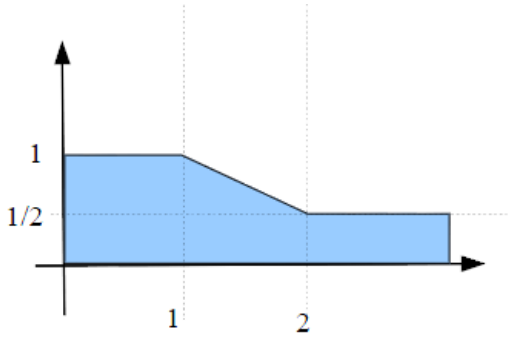
La compression d'un point donné d'un objet s'effectue de la manière suivante :

$$s(x) = \begin{cases} 1 & x < x_0 \\ 1 - \frac{1}{2}\frac{x - x_0}{x_1 - x_0} & x_0 \leq x \leq x_1 \\ \frac{1}{2} & x_1 < x \end{cases}$$

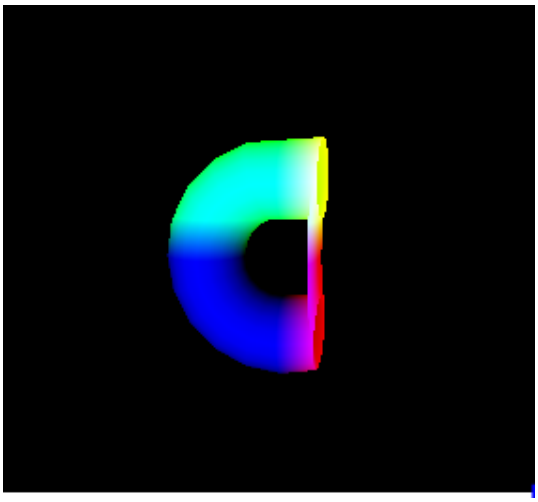
$$P' = \begin{bmatrix} s(p_x) & 0 & 0 \\ 0 & s(p_y) & 0 \\ 0 & 0 & s(p_z) \end{bmatrix} P$$

Ici, x0 et x1 sont les bornes inférieures et supérieures de la compression. Si le paramètre x est avant ces bornes, il ne bouge pas, autrement il est divisé par 2. La partie située entre les bornes correspond au changement de la valeur de x.

Ce schéma résume la situation dans un exemple en 2d, avec les bornes inférieures et supérieures égales respectivement à 1 et 2.

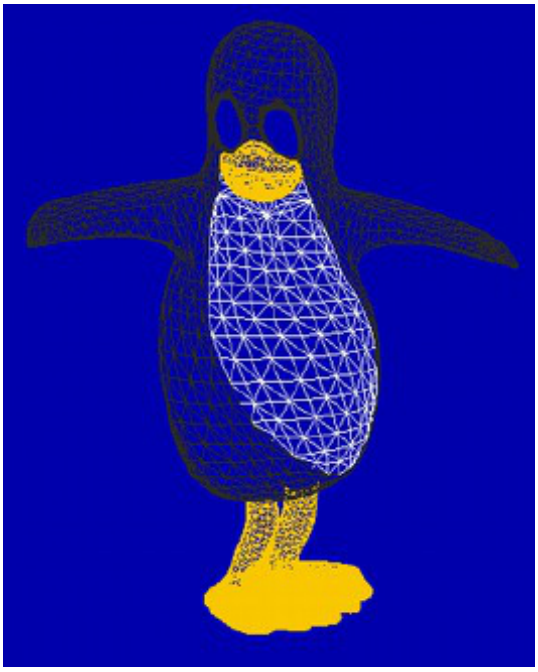


Au niveau du code, nous avons créé plusieurs fonctions qui effectuent une compression sur des coordonnées données du vecteur (ex : compressionX, CompressionY,...) pour pouvoir observer différents résultats. Grâce aux formules ci dessus, il suffit de 2 conditions pour obtenir la déformation souhaitée.



Compression x

2.2 Rotation différentielle



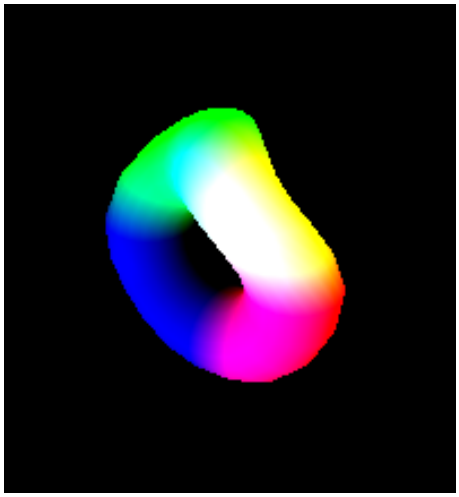
La rotation différentielle en un point s'effectue suivant cette formule :

$$r(z) = \begin{cases} 0 & z < z_0 \\ \frac{z - z_0}{z_1 - z_0}\theta_{max} & z_0 \leq z \leq z_1 \\ \theta_{max} & z_1 < z \end{cases}$$

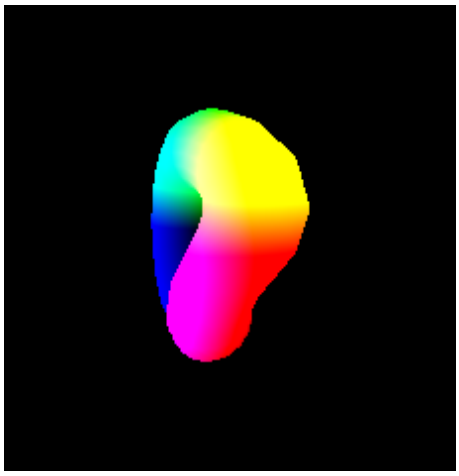
$$P' = \begin{bmatrix} \cos(r(p_z)) & -\sin(r(p_z)) & 0 \\ \sin(r(p_z)) & \cos(r(p_z)) & 0 \\ 0 & 0 & 1 \end{bmatrix} P$$

Le principe est similaire à celui de la compression avec z0 et z1 qui sont les bornes de la rotation. Cependant on rajoute ici un paramètre thêta qui symbolise l'angle de la rotation. Il faut donc d'abord calculer le paramètre r(pz) puis on multiplie les coordonnées du point en cours de traitement par la matrice ci-dessus.

Pour cela nous utilisons deux fonctions dans le code, la première impose les conditions pour la rotation, calcule r(pz) et appelle la deuxième fonction qui effectue le calcul des coordonnées du point avec la matrice. Comme pour la compression, nous avons plusieurs fonctions qui effectuent une rotation différentielle sur une coordonnée à la fois (rotationdiffX, rotationdiffY,...)

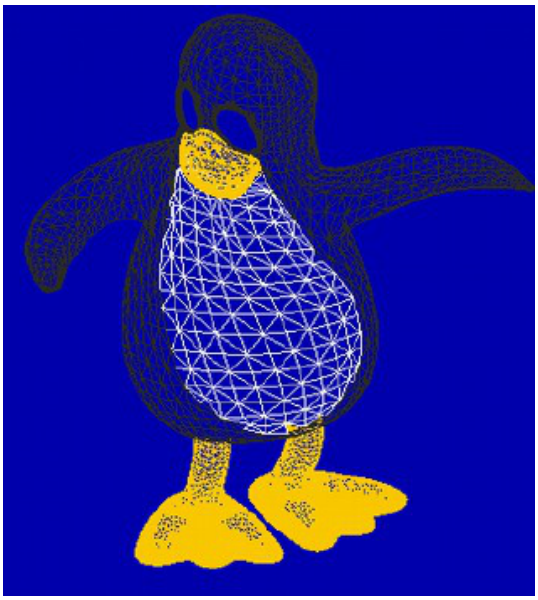


RotationDiffY



RotationDiffX

2.3 Vortex



Le vortex est très similaire à la rotation différentielle :

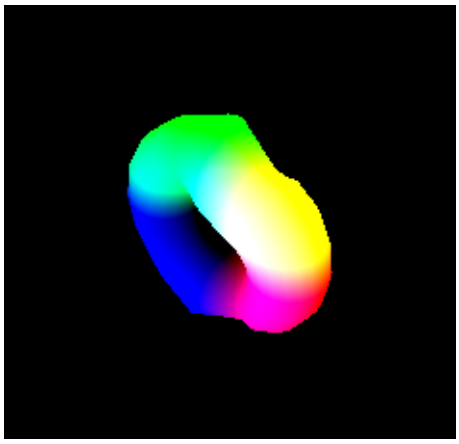
$$r(z) = \begin{cases} 0 & z < z_0 \\ \dfrac{z - z_0}{z_1 - z_0}\theta_{max} & z_0 \leq z \leq z_1 \\ \theta_{max} & z_1 < z \end{cases}$$

$$\alpha(P) = r(p_z)e^{-\left(p_x^2 + p_y^2\right)}$$

$$P' = \begin{bmatrix} \cos(\alpha(P)) & -\sin(\alpha(P)) & 0 \\ \sin(\alpha(P)) & \cos(\alpha(P)) & 0 \\ 0 & 0 & 1 \end{bmatrix} P$$

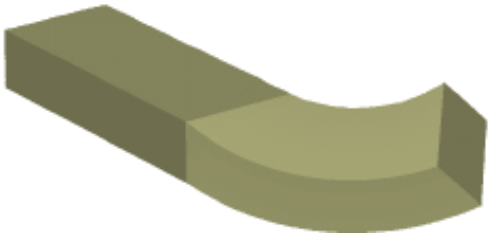
La seule différence réside dans le paramètre alpha qui rajoute une petite étape au processus de calcul.

La méthode employée dans le code est la même que pour la rotation différentielle avec là aussi plusieurs fonctions pour les différentes coordonnées.

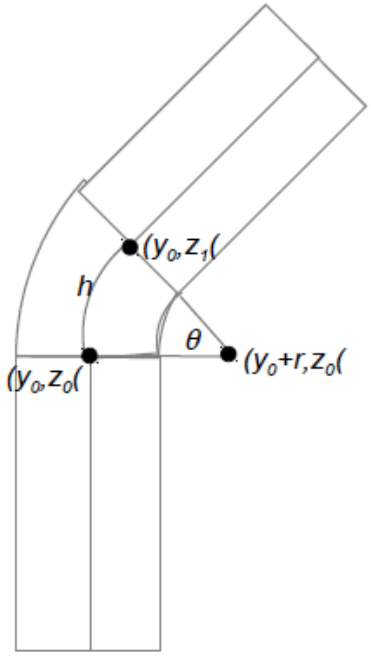


VortexY

2.4 Pliage



Le pliage se décompose en 3 parties comme on peut le voir sur ce schéma :



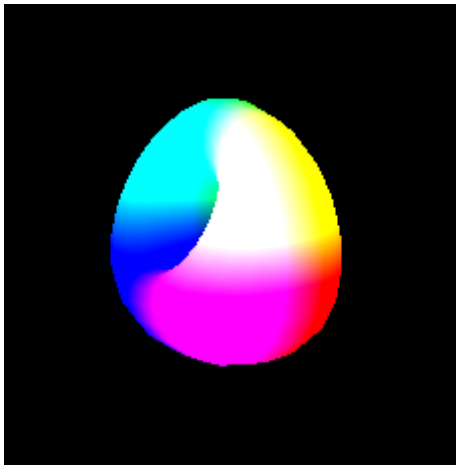
Nous avons besoin de 3 données pour effectuer cette déformation. La première est l'angle thêta du pliage, et les deux autres sont les bornes z0 et z1 qui jouent le même rôle que dans les déformations précédentes à savoir celui des bornes inférieures et supérieures de la zone à plier.

Avant z0, il ne se passe rien pour l'objet.

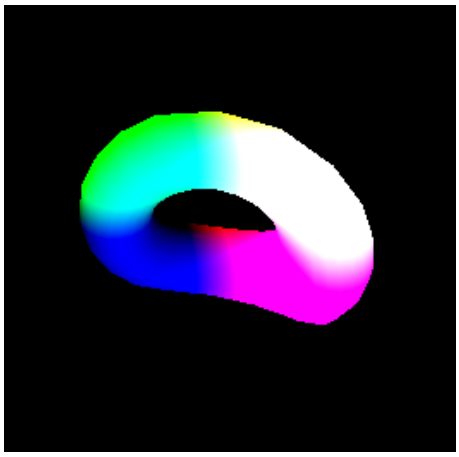
Après z1, nous devons effectuer une translation de z1-z0 sur tous les points de l'objet, puis une rotation d'angle thêta autour de l'axe qui nous intéresse (ici y0) auquel on ajoute le rayon défini par nos bornes, soit : (y0+r, z0).

Entre les deux bornes, le principe est le même. On effectue d'abord une translation de z-z0 puis une rotation d'angle thêta*(z-z0)/(z1-z0) autour de (y0+r, z0).

Nous avons la aussi plusieurs fonctions pour chaque coordonnée ce qui nous permet de faire un pliage suivant les 3 axes du repère.

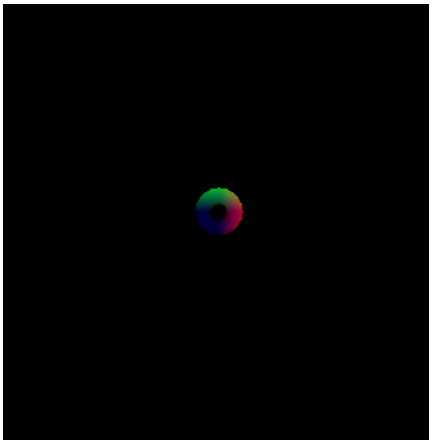


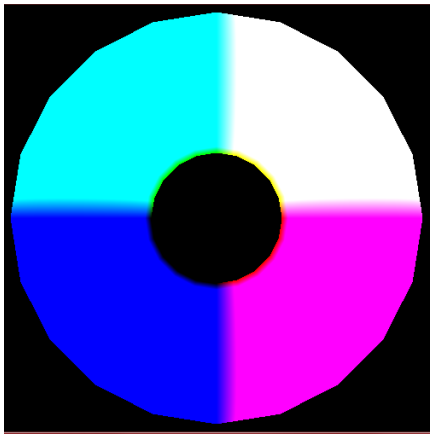
PliageX



PliageY

2.5 Agrandissement/Réduction





Cette opération est très simple à réaliser, il suffit de multiplier chaque coordonnée par le paramètre de changement de taille et le tour est joué.

3 Critique des résultats et améliorations possibles

Pour ce projet, nous avons réussi à mettre en place plusieurs primitives de déformation d'un objet avec un semblant d'interaction qui nous permet de toutes les tester sans redémarrer le programme.

Néanmoins, cette interaction reste limitée car les fonctions agissent sur des paramètres prédéfinis dans le programme, et les arguments de ces fonctions sont eux aussi inscrits dans le code. Il faut redémarrer le projet pour les modifier.

De plus, nous n'avons pas réussi à faire en sorte de pouvoir lire un maillage depuis un fichier ce qui nous aurait permis d'avoir un éventail beaucoup plus large d'objets à tester.