Rapport du TP noté

Table des matières

1	Partie Modélisation		
	1.1	Lissage du model	2
	1.2	Filtre de Taubin	2
2	2 Partie Rendu		3
	2.1	Appliquer le Cel Shading	3
	2.2	Positionner la lumière à la place de la caméra	4
	2.3	Contour noir	4
3	Animation		
	2.1	Interpolation linéaire	5

1 Partie Modélisation

1.1 Lissage du model

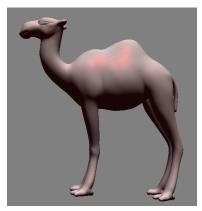
j'ai rajouté à la fonction smooth le lissage laplacien en récupérant les voisins de chaque sommets puis en appliquant à la position de chaque sommets la position moyenne des voisins (plus le lambda).

```
vector<vector<unsigned int>> voisins;
this->collectOneRing(voisins);
for (unsigned int i =0; i < vertices.size(); i++) {

    Vertex &vertex = vertices[i];
    Vec3 barycentre = Vec3(0,0,0);
    for (int j = 0; j < voisins[i].size(); j++) {
        barycentre += vertices[voisins[i][j]].position;
    }
    vertex.position += lambda * ((1.0/voisins[i].size())*barycentre - vertex.position);
}</pre>
```



(a) Modèle de base avec du bruit



(b) Modèle avec 5 itérations de lissage Laplacien

Figure 1 – Comparaison du modèle 3D avec et sans lissage

On constate que faire une itération du lissage réduit considérablement le bruit généré et que 2 ou 3 itérations sufissent à retrouver un modèle presque identique à celui de départ. Mais cela nous fait perdre en volume et à forcer le lissage on obtient un modèle très fin.

1.2 Filtre de Taubin

2 Partie Rendu

2.1 Appliquer le Cel Shading

J'ai ajouté shader.frag la fonction suivante après le calcul de dotLN :

```
float dotLN = max(dot (L, N), 0.);
for(int k = 1; k <= levels; k++) {
    float kfloat = float(k);
    float levelsfloat = float(levels);
    float niveau = kfloat * (1.0 / levelsfloat);
    bool flag = false;
    if(dotLN < niveau) {
        dotLN = niveau;
        flag = true;
        break;
    }
}</pre>
```

Cela permet de modifier la valeur de dotLN suivant le nombre de niveaux

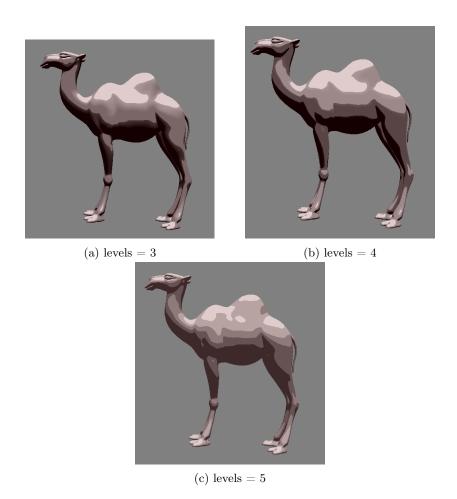


FIGURE 2 – Comparaison du modèle 3D avec différents niveaux de coloration

2.2 Positionner la lumière à la place de la caméra

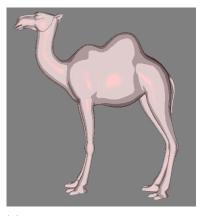
L = V;

P est un Vec3 déjà défini qui contient la position de la caméra vec3 P = vec3 (gl_ModelViewMatrix * p)

On utilise donc V qui est la matrice inverse de P



(a) Modèle 3D avec la lumière positionnée en haut du modèle



(b) Modèle 3D avec la lumière positionnée face au modèle, à la place de la caméra

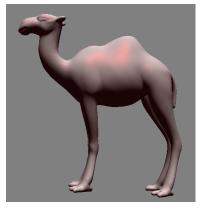
FIGURE 3 – Comparaison du placement de la lumière (en haut ou en face)

On peut constater et vérifier que cela a bien fonctionné en regardant les ombres portées sur le modèle 3D.

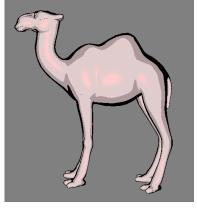
2.3 Contour noir

```
if(dot(N, V) < 0.5) I = vec4(0.,0.,0.,0.);
```

J'ai rajouté à la fin du fichier .frag un bout de code pour vérifier que si le produit scalaire de N et V et inférieur à 0.5 alors on remplace I par un vecteur nul



(a) Modèle 3D sans contour



(b) Modèle 3D avec contour

3 Animation

3.1 Interpolation linéaire

J'ai commencé à faire des boucles pour modifier les positions de current_mesh mais sans succès. Ci-après le début de mon code :

```
for (int i = 0; i < V0.size(); i++) {
    V[i].position = V0[i].position + V1[i].position;
}

const vector<Vertex> & V2 = mesh_pose_2.getVertices ();

for (int i = 0; i < V0.size(); i++) {
    V[i].position = (V0[i].position + V1[i].position + V2[i].position) / 3;
}</pre>
```