PRESENTATION PROJET IMAGES

RÉALISÉ PAR :

- EL MAGHOUM FAYÇAL
- DIALLO MAMOUDOU
- IDIR AMZAL
- TAHIR ILYAS
- AMRANI SALIM
- EMIR-MOUNGONDO CRISTAN

GROUPE 2

ANNÉE UNIVERSITAIRE: 2023-2024

SOMMAIRE

PRESENTATION DU SUJET

Notre sujet se divise en 2 parties:

1. D’une part on nous a demander de plaquer une texture, en la déformant, sur une forme géométrique, de manière à ce que depuis un point de vue on reconnaisse l’image non déformée représenté par la texture.

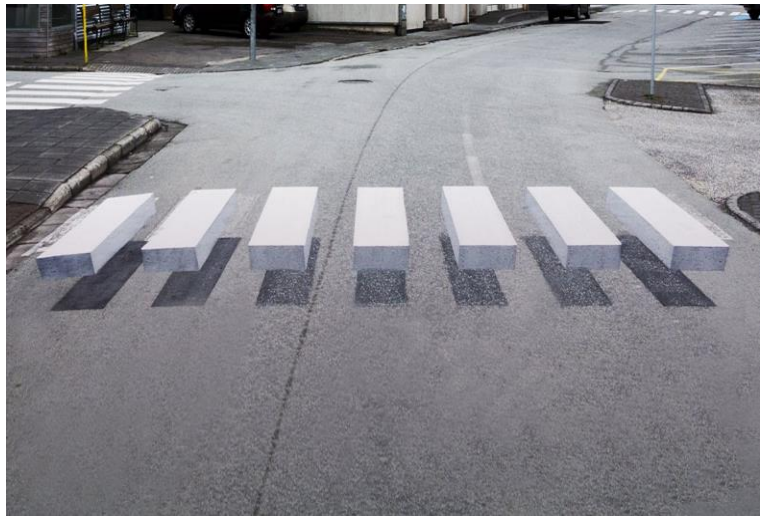
2. Et d’autre part on nous a demandé d’appliquer le principe d’anamorphose pour un support plan sur lequel est posé un miroir cylindrique.

DEFINITION ANAMORPHOSE

L’anamorphose est une transformation mathématique d’une image pour qu’elle s’apparente à une illusion d’optique

PRINCIPES DES ANAMORPHOSES

Le principe des anamorphoses est de plaquer une texture sur un support donné de manière à ce que l'image représentée par la texture soit perçue non déformée dans un miroir de forme géométrique donnée pour un observateur situé à un endroit précis.



Organisation

Étant donné que notre sujet avait deux parties on a décidé de faire deux groupes :

Anamorphose forme géométrique

- IDIR Amzal "Cube"
- TAHIR Ilyas "Cone"
- AMRANI Salim "Pyramide triangulaire"
- EMIR-MOUNGONDO Cristan "Pyramide base 4 et N"

ANAMORPHOSE SUR UN MIROIR CYLINDRIQUE

- EL MAGHOUM FAYÇAL
- DIALLO MAMOUDOU

Partie 1 :

ANAMORPHOSE SUR DES FORMES GEOMETRIQUES

LE CÔNE

```
glBegin(GL_TRIANGLE_FAN);
glTexCoord2f(0.5f, 0.5f);
glVertex3f(0.0f, height / 2.0f, 0.0f);

for (int i = 0; i <= segments; ++i)
{
    float theta = (2.0f * 3.1415926f * float(i)) / float(segments);
    float x = radius * cosf(theta);
    float y = radius * sinf(theta);

    float texCoordX = (x / radius + 1.0f) * 0.5f;
    float texCoordY = (y / radius + 1.0f) * 0.5f;

    glTexCoord2f(texCoordX, texCoordY);
    glVertex3f(x, -height / 2.0f, y);
}

glEnd();
```

```
// Base du cone avec texture
glColor3f(1.0,1.0,1.0);
glBegin(GL_TRIANGLE_FAN);
for (int i = 0; i <= segments; ++i)
{
    float theta = (2.0f * 3.1415926f * float(i)) / float(segments);
    float x = radius * cosf(theta);
    float y = radius * sinf(theta);

    //float texCoordX = (x / radius + 1.0f) * 0.5f;
    //float texCoordY = (y / radius + 1.0f) * 0.5f;

    //glTexCoord2f(texCoordX, texCoordY);
    glColor3f(1.0,1.0,1.0);
    glVertex3f(x, -height / 2.0f, y);
}

glEnd();
```

LE CÔNE

LA PYRAMIDE TRIANGULAIRE

- Faire une pyramide traingulaire en 3D
- Plaquer une texture d'une façon precise pour que l'image soit visible lorsqu'on regarde la pyramide de haut

LA PYRAMIDE TRIANGULAIRE

```
glBegin(GL_TRIANGLE_FAN);
glTexCoord2f(0.5f, 0.5f);
glVertex3f(0.0f, height / 2.0f, 0.0f);

for (int i = 0; i <= segments; ++i) {
    float theta = (2.0f * PI * float(i)) / float(segments);
    float x = baseSize * cosf(theta);
    float y = baseSize * sinf(theta);

    float texCoordX = (x / baseSize + 1.0f) * 0.5f;
    float texCoordY = (y / baseSize + 1.0f) * 0.5f;

    glTexCoord2f(texCoordX, texCoordY);
    glVertex3f(x, -height / 2.0f, y);
}

glEnd();
```

LA PYRAMIDE TRIANGULAIRE

LA PYRAMIDE À BASE CARÉE

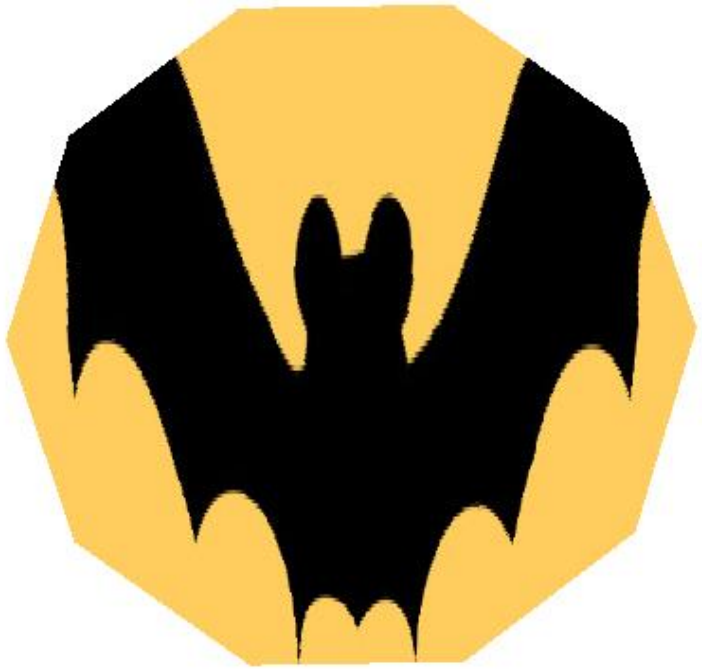
```
// Dessine les côtés de la pyramide
glBegin(GL_TRIANGLE_FAN);
glTexCoord2f(0.5f, 0.5f);
glVertex3f(0.0f, height / 2.0f, 0.0f);

for (int i = 0; i <= segments; ++i)
{
    float theta = (2.0f * PI * float(i)) / float(segments);
    float x = baseSize * cosf(theta);
    float y = baseSize * sinf(theta);

    float texCoordX = (cosf(theta) + 1.0f) * 0.5f;
    float texCoordY = (sinf(theta) + 1.0f) * 0.5f;

    glTexCoord2f(texCoordX, texCoordY);
    glColor3f(1.0, 1.0, 1.0);
    glVertex3f(x, -height / 2.0f, y);
}
glEnd();
```

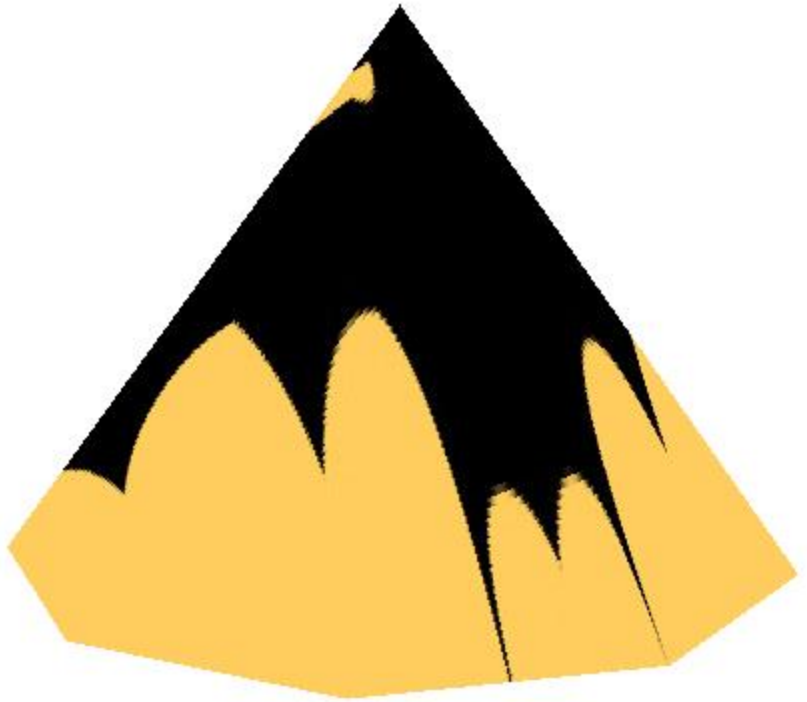
LA PYRAMIDE À BASE CARÉE

LA PYRAMIDE À BASE D'UN POLYGONE RÉGULIER

```
glBegin(GL_TRIANGLES);

for (int i = 0; i < sides; ++i) {
    float angle1 = 2.0f * PI * static_cast<float>(i) / static_cast<float>(sides);
    float angle2 = 2.0f * PI * static_cast<float>(i + 1) / static_cast<float>(sides);

    // Triangle adjacent to the base
    glTexCoord2f(0.5f, 0.5f);
    glVertex3f(0.0f, -height / 2.0f, 0.0f);

    glTexCoord2f(cos(angle1) * 0.5f + 0.5f, sin(angle1) * 0.5f + 0.5f);
    glVertex3f(baseRadius * cos(angle1), -height / 2.0f, baseRadius * sin(angle1));

    glTexCoord2f(cos(angle2) * 0.5f + 0.5f, sin(angle2) * 0.5f + 0.5f);
    glVertex3f(baseRadius * cos(angle2), -height / 2.0f, baseRadius * sin(angle2));

    // Triangle forming the sides
    glTexCoord2f(0.5f, 0.5f);
    glVertex3f(0.0f, height / 2.0f, 0.0f);

    glTexCoord2f(cos(angle1) * 0.5f + 0.5f, sin(angle1) * 0.5f + 0.5f);
    glVertex3f(baseRadius * cos(angle1), -height / 2.0f, baseRadius * sin(angle1));

    glTexCoord2f(cos(angle2) * 0.5f + 0.5f, sin(angle2) * 0.5f + 0.5f);
    glVertex3f(baseRadius * cos(angle2), -height / 2.0f, baseRadius * sin(angle2));
}

glEnd();
```

LA PYRAMIDE À BASE D'UN POLYGONE RÉGULIER





LE CUBE

- Faire un cube en 3D
- Plaquer une texture d'une façon precise pour que l'image soit visible lorsqu'on regarde le cube en diagonale
- Faire en sorte d'appliquer deux images sur le cube

LE CUBE

```
glBegin(GL_POLYGON);
glTexCoord2f(0.0, 0.5); glVertex3f(-0.5, 0.5, 0.5);
glTexCoord2f(0.0, 1.0); glVertex3f(-0.5, -0.5, 0.5);
glTexCoord2f(0.5, 1.0); glVertex3f(0.5, -0.5, 0.5);
glTexCoord2f(0.5, 0.5); glVertex3f(0.5, 0.5, 0.5);
glEnd();
```

```
glBegin(GL_POLYGON);
glTexCoord2f(0.5, 0.5); glVertex3f( 0.5, 0.5, 0.5);
glTexCoord2f(0.5, 1.0); glVertex3f( 0.5,-0.5, 0.5);
glTexCoord2f(1.0, 1.0); glVertex3f( 0.5,-0.5,-0.5);
glTexCoord2f(1.0, 0.5); glVertex3f( 0.5, 0.5,-0.5);
glEnd();
```

```
glBegin(GL_POLYGON);
glTexCoord2f(0.0, 1.0); glVertex3f( 0.5, 0.5,-0.5);
glTexCoord2f(0.0, 0.5); glVertex3f( 0.5,-0.5,-0.5);
glTexCoord2f(0.5, 0.5); glVertex3f(-0.5,-0.5,-0.5);
glTexCoord2f(0.5, 1.0); glVertex3f(-0.5, 0.5,-0.5);
glEnd();
```

```
glBegin(GL_POLYGON);
glTexCoord2f(0.5, 1.0); glVertex3f(-0.5, 0.5,-0.5);
glTexCoord2f(0.5, 0.5); glVertex3f(-0.5,-0.5,-0.5);
glTexCoord2f(1.0, 0.5); glVertex3f(-0.5,-0.5, 0.5);
glTexCoord2f(1.0, 1.0); glVertex3f(-0.5, 0.5, 0.5);
glEnd();
```

```
glBegin(GL_POLYGON);
glTexCoord2f(0.5, 0.0); glVertex3f(-0.5, 0.5,-0.5);
glTexCoord2f(1.0, 0.0); glVertex3f(-0.5, 0.5, 0.5);
glTexCoord2f(0.5, 0.5); glVertex3f( 0.5, 0.5, 0.5);
glTexCoord2f(0.0, 0.0); glVertex3f( 0.5, 0.5,-0.5);
glEnd();
```

```
glBegin(GL_POLYGON);
glTexCoord2f(0.5, 0.5); glVertex3f(-0.5,-0.5,-0.5);
glTexCoord2f(0.0, 0.0); glVertex3f(-0.5,-0.5, 0.5);
glTexCoord2f(0.5, 0.0); glVertex3f( 0.5,-0.5, 0.5);
glTexCoord2f(1.0, 0.0); glVertex3f( 0.5,-0.5,-0.5);
glEnd();
```

LE CUBE

Partie 2 :

ANAMORPHOSE SUR UN MIRROIR CYLINDRIQUE

MIRROIR CYLINDRIQUE

L’anamorphose du miroir cylindrique est composée de 2 éléments un cylindre (représente le miroir) et un plan où sera placer la texture (image déformé)

MIRROIR : CYLINDRE

```
void drawCylinder()
{
    GLUquadric *quad = gluNewQuadric();
    gluQuadricTexture(quad, GL_TRUE);

    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, texture);

    glPushMatrix();
    glTranslatef(0.0f, 1.5f, -1.6f);
    glRotatef(90.0f, 1.0f, 0.0f, 0.0f);
    glRotatef(180.0f, 0.0f, 0.0f, 1.0f);
    // Activation du mappage de l'environnement

    glEnable(GL_TEXTURE_GEN_S);
    // glEnable(GL_TEXTURE_GEN_T);

    // Configuration du plan de réflexion (ici, nous utilisons GL_SPHERE_MAP)
    glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);
    // glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);

    gluCylinder(quad, 0.5, 0.5, 2.5, 32, 32);

    glPopMatrix();

    // Désactivation du mappage de l'environnement après le rendu
    glDisable(GL_TEXTURE_GEN_S);
    glDisable(GL_TEXTURE_GEN_T);

    glBindTexture(GL_TEXTURE_2D, 0);
    glDisable(GL_TEXTURE_2D);

    gluDeleteQuadric(quad);
}
```

- Ce code dessine un cylindre texturé avec un effet de mappage environnemental sphérique, créant ainsi un effet de réflexion similaire à une miroir.
- gluCylinder(quad, 0.5, 0.5, 2.5, 32, 32); dessine le cylindre avec un rayon de 0.5 et une hauteur de 2.5,
- gluQuadricTexture(quad, GL_TRUE); active le mappage de texture sur le Quadric, indiquant qu'une texture sera appliquée à la surface générée par le Quadric
- La texture est appliquée automatiquement grâce à la méthode glBindTexture(..)

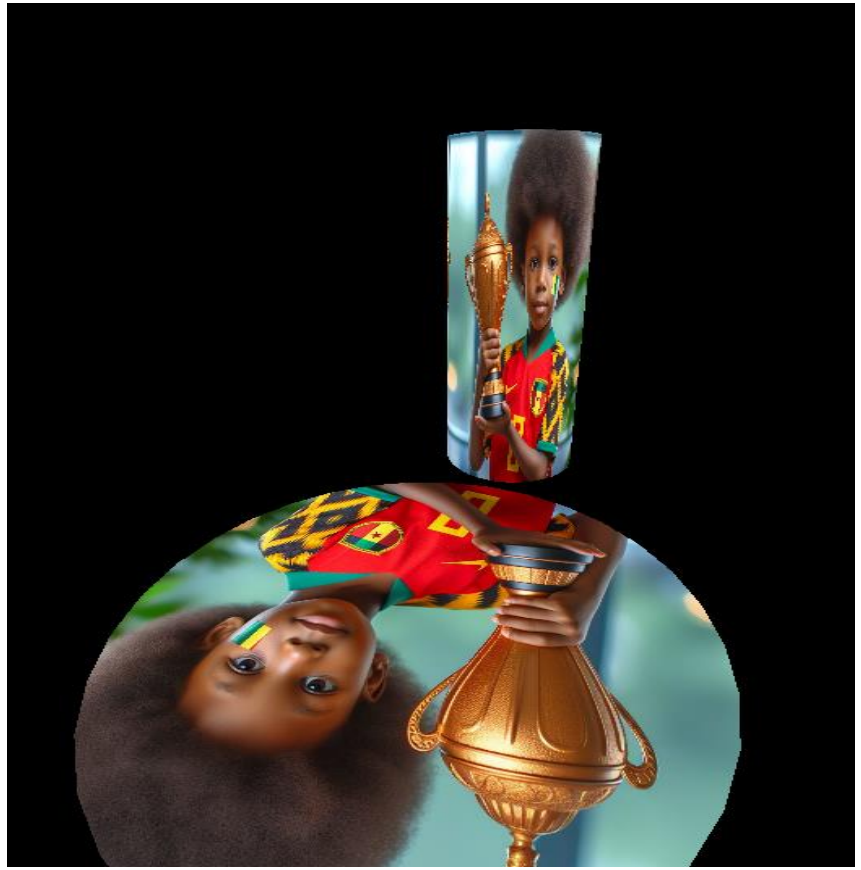
MIRROIR : LE PLAN

```
void drawCircularSurface()
{
    GLUquadric *quad = gluNewQuadric();
    gluQuadricTexture(quad, GL_TRUE);

    glDisable(GL_TEXTURE_GEN_S);
    glDisable(GL_TEXTURE_GEN_T);

    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, texture);

    glPushMatrix();
    glTranslatef(0.0f, -1.0f, 0.5f);
    glRotatef(90.0f, 1.0f, 0.0f, 0.0f);
    glRotatef(180.0f, 0.0f, 0.0f, 1.0f);
    gluDisk(quad, 0.0, 1.5, 32, 1);

    glPopMatrix();

    glBindTexture(GL_TEXTURE_2D, 0);
    glDisable(GL_TEXTURE_2D);

    gluDeleteQuadric(quad);
}
```

- Cette fonction crée une surface circulaire texturée en utilisant un Quadric (objet de type Quadric qui sera utilisé pour dessiner la surface circulaire). Elle sert à appliquer une texture spécifique à cette surface circulaire, effectuer des transformations pour positionner et orienter la surface, puis enfin dessiner un disque.

- La texture est appliquée automatiquement grâce à la méthode glBindTexture(..)

MIRROIR : LE PLAN



CONCLUSION

Ce projet d'anamorphose en OpenGL a été une expérience intéressante pour nous. Nous avons réussi à appliquer des textures sur différentes formes telles que le cube, le cône, etc., et à observer comment les images se déforment avec un miroir cylindrique.

Dans la partie consacrée à l'anamorphose sur les formes, nous avons rencontré des défis tels que le calcul des coordonnées de texture et des points de construction des figures. Quant à la partie du miroir, nous avons fait face à des défis tels que la réflexion Plan-Cylindre (miroir) et le mappage. Pour résoudre ces problèmes, nous avons adopté une approche de gestion des coordonnées de texture et des transformations appropriées.

Merci à nos enseignants pour leur temps et leur soutien.

Merci de votre attention.

DEMONSTRATION