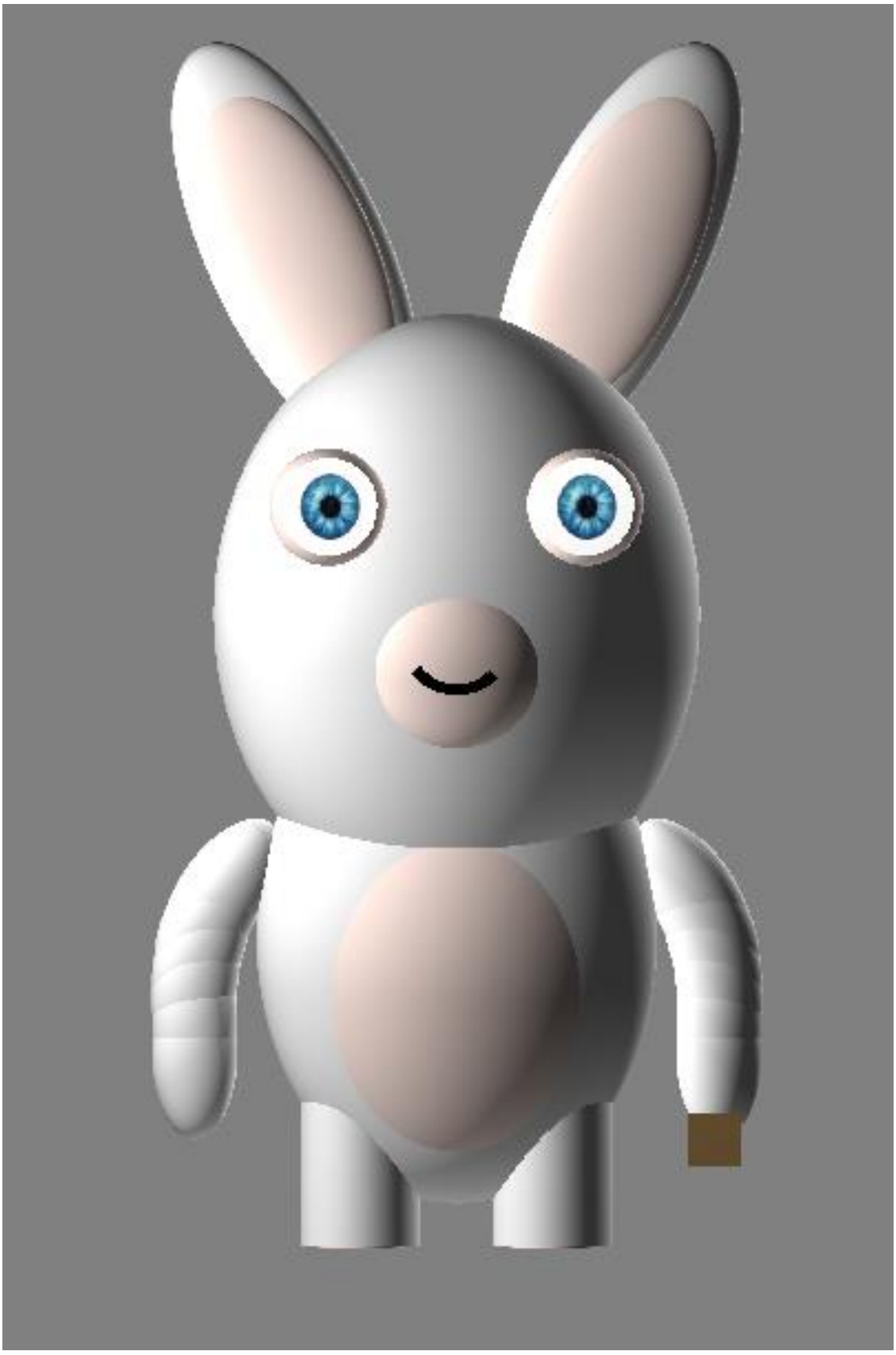
GIE Jimmy

LOGEROT Alexis

Rapport Projet Synthèse d'image



SOMMAIRE :

I. Conception du lapin

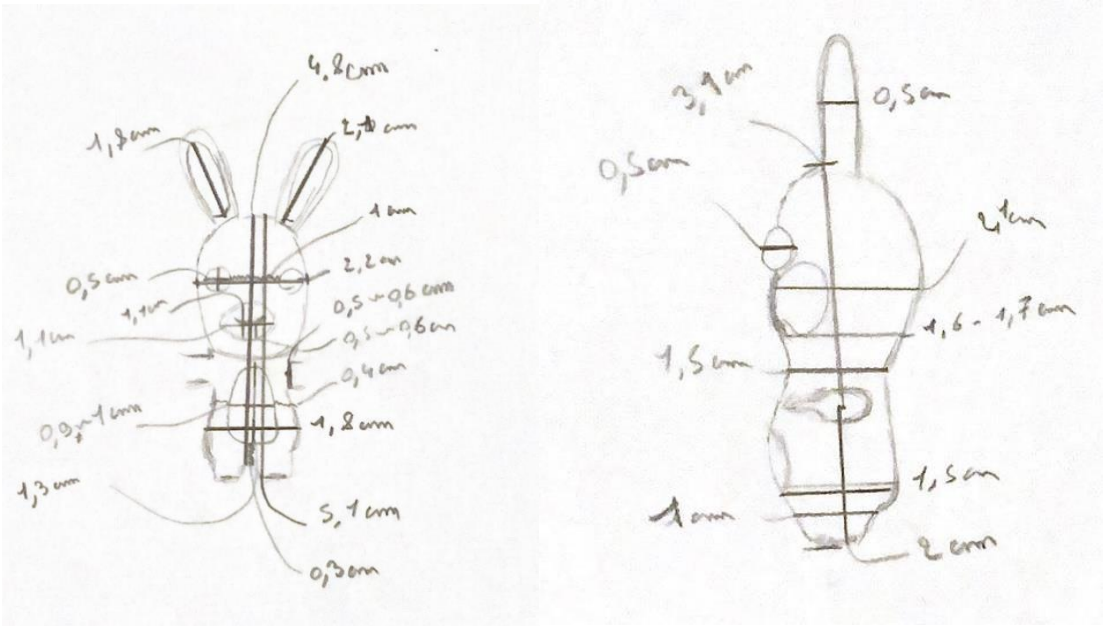
1. Idée de lapin :

Notre première idée de lapin était de faire Bonny, le robot lapin du jeu Five Night At Freddy's, un jeu d'horreur sur des robots tueurs, le Robot est constitué d'un squelette simple entouré d'une armature par-dessus qui lui donne ça forme de lapin.

À la suite de soucis de difficulté pour la création des formes complexe du lapin, nous avons décidé de prendre un model plus simple et nous avons conclue sur un modèle de lapin crétin qui fait aussi parti de la pop culture mais aussi plus simple à concevoir
Nous nous sommes donc aidés de ce modèle comme base pour définir les formes et dimensions de notre lapin :

2. Conception d'un lapin Crétin :

Avec le modèle montré précédemment, nous avons recrée un schéma à l'échelle pour pouvoir avoir une idée des dimensions qu'allait prendre notre lapin, nous avons aussi créer un schéma avec des formes simple pour voir quelles formes pouvais être utiles.



Croquis du lapin avec les dimensions

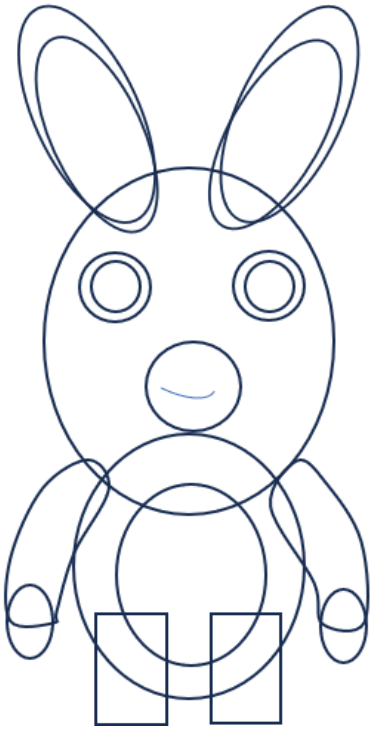


Schéma du lapin avec des formes simple

II. Création des Formes

1. Formes Glut :

A l'aide du schéma de forme simple que nous avons créé, nous avons pu définir quelles parties du corps serait créée à l'aide des formes glut de openGL : le Corps, les oreilles, la tête, les mains, et la forme rose en dessous de sa bouche. Toute les formes sont créées à l'aide d'un glutSolidSphere que l'on déformer et déplace (à l'aide de glScalef, glTranslatef et glRotatef) pour créer les formes voulues.

Pour chaque forme du corps, nous avons créés un double de la forme que l'on rend plus petit pour créer la couleur rose caractéristique des lapins crétin

Exemple avec le corps qui est composé d'une partie blanche qui est un cercle déformé pour prendre la forme d'un ovale mais aussi de la même forme plus petite et rose qui dépasse du blanc pour créer Son ventre rose

2. Formes Primitives :

La fonction createSphere nous permet de créer la paupière de l'œil du lapin, Pour cette fonction le programme a besoin de 2 int n et m qui seront le nombre de méridiens horizontaux et verticaux (plus il y en a, plus la sphère sera détaillée et bien sphérique), du fl le rayon de la sphère et d'un autre float angle qui sera l'angle que prendra la sphère (un angle de 180° donnera une demi-sphère), la fonction va tout d'abord créer un tableau de coordonnés sphérique qu'il va calculer en fonction de notre nombre de méridiens horizontaux et de méridiens verticaux pour ensuite les transformer en coordonnées cartésiennes (x,y,z) pour que l'on puisse les utiliser dans la création de la sphère

```
void createSphere(int n, int m, float rayon, float angle)
{
    // Calcul du nombre de points et de faces
    int numPoints = n * m;
    int numFaces = (n - 1) * (m - 1);

    // Alloue de la mémoire pour les points de la sphère et les faces
    Point* sphere = new Point[numPoints];
    Face* face_sphere = new Face[numFaces];

    // Générer les points de la sphère
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            double theta = ((double)j / (double)(m - 1.0)) * angle;
            double phi = ((double)i / (double)n) * (2 * M_PI);

            // Coordonnées cartésiennes en fonction de theta et phi
            sphere[j + (i * m)].x = rayon * sin(theta) * cos(phi);
            sphere[j + (i * m)].z = rayon * sin(theta) * sin(phi);
            sphere[j + (i * m)].y = rayon * cos(theta);
        }
    }
```

Il va ensuite créer un autre tableau qui sera un stockage des index de points pour la face i (face [0] contiendra les 4 points qui compose sa face 0)

```
    // Génère les faces de la sphère en utilisant les indices des points
    int index = 0;
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < m - 1; j++) {
            Face f{ 0, 0, 0, 0 };

            f.i = (i * m) + j;
            f.l = (i * m) + j + 1;

            if (i != n - 2) {
                f.j = ((i + 1) * m) + j;
                f.k = ((i + 1) * m) + j + 1;
            } else {
                f.j = j;
                f.k = j + 1;
            }

            face_sphere[index++] = f;
        }
    }
```

Pour finir la fonction va créer à l'aide du tableau de face, du tableau de points et de GL_GUAD_STRIP la sphère comme n'importe quelle autre forme crée par primitive (il va placer le point à l'aide de glVertex3f qui va créer la forme de sphère petit à petit).

```
    // Dessine la sphère en utilisant les points, les normales et les faces
    glBegin(GL_QUADS);
    for (int i = 0; i < numFaces; i++) {
        // Dessine les faces avec les normales et les points correspondants
        glNormal3f(normals[face_sphere[i].i].x, normals[face_sphere[i].i].y, normals[face_sphere[i].i].z);
        glVertex3f(sphere[face_sphere[i].i].x, sphere[face_sphere[i].i].y, sphere[face_sphere[i].i].z);

        glNormal3f(normals[face_sphere[i].j].x, normals[face_sphere[i].j].y, normals[face_sphere[i].j].z);
        glVertex3f(sphere[face_sphere[i].j].x, sphere[face_sphere[i].j].y, sphere[face_sphere[i].j].z);

        glNormal3f(normals[face_sphere[i].k].x, normals[face_sphere[i].k].y, normals[face_sphere[i].k].z);
        glVertex3f(sphere[face_sphere[i].k].x, sphere[face_sphere[i].k].y, sphere[face_sphere[i].k].z);

        glNormal3f(normals[face_sphere[i].l].x, normals[face_sphere[i].l].y, normals[face_sphere[i].l].z);
        glVertex3f(sphere[face_sphere[i].l].x, sphere[face_sphere[i].l].y, sphere[face_sphere[i].l].z);
    }
    glEnd();

    // Libère la mémoire allouée
    delete[] sphere;
    delete[] face_sphere;
    delete[] normals;
}
```

La fonction createSphereOeil est une fonction très similaire à la fonction précédente, et nous permet de créer une sphère complète en y appliquant une texture de pupille, ce qui la différencie de createSphere est qu'elle n'a pas besoin d'un angle donné car la sphère sera toujours complète, pour ce qui est du code à l'intérieur nous calculons toujours les coordonnées des points comme avant, mais ce qui change est que nous n'utilisons pas de tableau de face avec les indices des points pour les appeler directement dans le glvertex3f. Cela rend la ligne plus compliqué mais nous permet d'appliquer en plus à chaque point la texture plus simplement pour l'œil grâce aux 2 boucles i et j au lieux d'une seule boucle.

```
    // Dessine la sphère en utilisant des quadrilatères texturés
    glBegin(GL_QUADS);
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < m - 1; j++) {
            // Défini les coordonnées de texture et les sommets des quadrilatères
            glTexCoord2f(static_cast<float>(j) / (m - 1), static_cast<float>(i) / (n - 1));
            glVertex3f(sphere[j + (i * m)].x, sphere[j + (i * m)].y, sphere[j + (i * m)].z);

            glTexCoord2f(static_cast<float>(j + 1) / (m - 1), static_cast<float>(i) / (n - 1));
            glVertex3f(sphere[j + 1 + (i * m)].x, sphere[j + 1 + (i * m)].y, sphere[j + 1 + (i * m)].z);

            glTexCoord2f(static_cast<float>(j + 1) / (m - 1), static_cast<float>(i + 1) / (n - 1));
            glVertex3f(sphere[j + 1 + ((i + 1) * m)].x, sphere[j + 1 + ((i + 1) * m)].y, sphere[j + 1 + ((i + 1) * m)].z);

            glTexCoord2f(static_cast<float>(j) / (m - 1), static_cast<float>(i + 1) / (n - 1));
            glVertex3f(sphere[j + ((i + 1) * m)].x, sphere[j + ((i + 1) * m)].y, sphere[j + ((i + 1) * m)].z);
        }
    }
    glEnd();

    // Libère la mémoire allouée
    delete[] sphere;
}
```

Notre dernière Forme primitive est un cylindre pour créer les pieds du lapin, elle est créée avec la fonction Cylindre qui a besoin du rayon du cylindre, de la hauteur du cylindre et de méridien qui sera

le nombre de face du cylindre, la création se fait en 3 parties, le contour du cylindre, puis la face en haut et en bas.
Pour la partie centrale, le programme va utiliser des angles similaires aux création des sphères, avec la boucle for et l'angle, il va calculer l'angle nécessaire pour placer chaque segment qui va constituer le cylindre, la va calculer chaque point x,y par rapport aux coordonnées sphériques pour pouvoir créer les faces avec glVertex3f (il a besoin de 2 glVertex3f, un a la base du cylindre et un en haut du cylindre pour créer la droite)

```
void Cylindre(float rayon, float haut, int meridien)
{
    int i;
    float angle;
    float x, y;

    glBegin(GL_QUAD_STRIP);
    for (i = 0; i <= meridien; ++i)
    {
        angle = 2.0 * M_PI * i / meridien;
        x = rayon * cos(angle);
        y = rayon * sin(angle);

        // Normale pour le côté du cylindre
        float nx = cos(angle);
        float ny = sin(angle);
        float nz = 0.0f;
        glNormal3f(nx, ny, nz);

        glVertex3f(x, y, 0.0); // Point de la base du cylindre
        glVertex3f(x, y, haut); // Point du haut du cylindre
    }
    glEnd();
```

Les deux autres parties du code permettes de créer les faces du haut et du bas du cylindre, à l'aide du GL_TRIANGLE_FAN qui va créer des triangles, nous allons d'abord placer un point au centre puis comme pour la face circulaire, nous allons utiliser un angle pour avoir la distance entre les points en fonction de méridien et calculer les coordonnées pour pouvoir créer les triangles entre le point centrale et les points placés sur le contour de la face (la différence entre la face du bas et du haut est seulement la valeur de z qui est la hauteur du cylindre)

```
    glBegin(GL_TRIANGLE_FAN);
    glNormal3f(0.0f, 0.0f, -1.0f); // Normale pour la base inférieure (orientée vers le bas)
    glVertex3f(0.0, 0.0, 0.0); // Centre de la base inférieure
    for (i = 0; i <= meridien; ++i)
    {
        angle = 2.0 * M_PI * i / meridien;
        x = rayon * cos(angle);
        y = rayon * sin(angle);
        glVertex3f(x, y, 0.0); // Points de la base inférieure
    }
    glEnd();

    glBegin(GL_TRIANGLE_FAN);
    glNormal3f(0.0f, 0.0f, 1.0f); // Normale pour la base supérieure (orientée vers le haut)
    glVertex3f(0.0, 0.0, haut); // Centre de la base supérieure
    for (i = 0; i <= meridien; ++i)
    {
        angle = 2.0 * M_PI * i / meridien;
        x = rayon * cos(angle);
        y = rayon * sin(angle);
        glVertex3f(x, y, haut); // Points de la base supérieure
    }
    glEnd();
     glNormal3f(0.0, 0.0, 1.0);
}
```

III. Lumières

Nous avons 2 types de lumières, une lumière ambiante rouge qui peut être activé avec des touches du clavier.

```
glPushMatrix();
GLfloat ColorAmbiant[] = {1, 0, 0, 1};
glLightfv(GL_LIGHT3, GL_AMBIENT, ColorAmbiant);
glPopMatrix();
```

Création d'une lumière ambiante avec glLightfv

```
switch (touche)
  {
  case 'T':
    glEnable(GL_LIGHT3);
    break;
  case 't':
    glDisable(GL_LIGHT3);
    break;
```

Code pour activer et désactiver la lumière avec la touche

Nous avons aussi plusieurs spots de lumière diffuse autour du lapin qui permet de créer des ombres sur la structure, et notamment sur les Forme crées par primitive grâce aux formes normales calculé en même temps que les formes correspondantes cela permet donc d'avoir de l'ombre aussi bien sur les objets glut que sur les formes primitives

```
glPushMatrix();
GLfloat posLight1[] = {-0.8, 0.8, 0.8, 1};
glLightfv(GL_LIGHT0, GL_POSITION, posLight1);
glEnable(GL_LIGHT0);
glPopMatrix();

glPushMatrix();
GLfloat posLight2[] = {1, 1, -1, 1};
GLfloat ColorDiffuse[] = {1, 1, 1, 1};
glLightfv(GL_LIGHT1, GL_POSITION, posLight2);
glLightfv(GL_LIGHT1, GL_DIFFUSE, ColorDiffuse);
glEnable(GL_LIGHT1);
glPopMatrix();

glPushMatrix();
GLfloat posLight3[] = {0, -1, 0, 1};
```

Lumière de face

- Lumière de dos

Lumière en dessous

IV. Animations

1. Animations Automatique :

Pour l'animation automatique nous avons eu l'idée de faire ouvrir et fermer les yeux du lapin, ou avec notre fonction createSphere nous pouvons choisir l'angle d'ouverture du lapin grâce à notre fonction animationOeil nous ajustons l'angle d'ouverture et de fermeture. Nous avons choisi de faire ouvrir l'œil a 2Pi/3 pour une ouverture plus réaliste, ici notre code nous permet de faire varier l'ouverture entre 2Pi/3 et 0

glutTimeFunc nous permet d'appeler la fonction en continue, pour que lorsque que l'on active une autre animation manuelle celle-ci ne soit pas stoppé.

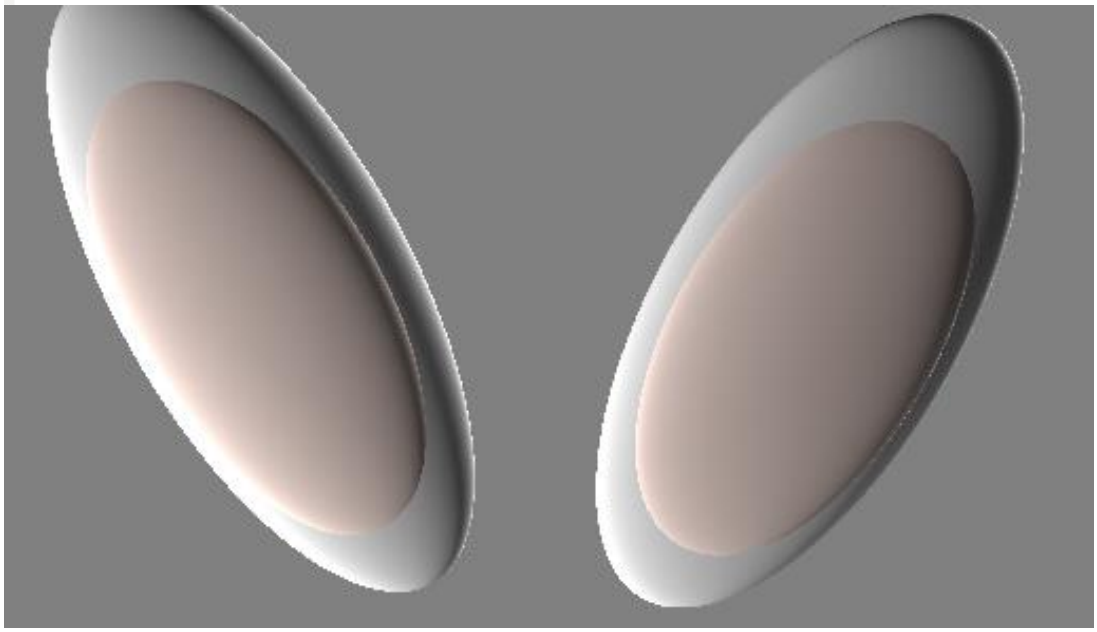
```
void animationOeil(int value)
{
    if (angleOuvert <= (2*M_PI/3) || angleOuvert >= M_PI )
    {
        value *= -1; // Change la direction de l'animation

    }
    angleOuvert += value * (M_PI / 150); // Ajuste l'angle progressivement

    glutPostRedisplay(); // Redessine la sphère avec le nouvel angle
    glutTimerFunc(16, animationOeil, value); // Déclenche une nouvelle mise à jour après 16ms (environ 60 FPS)
}
```

2. Animations Manuelles :

Pour les animations avec des touches, nous avons 2 animations, une première animation qui va faire bouger les oreilles avec la touche 'c' et 'C' qui va permettre le mouvement des oreilles, l'animations marche sous forme de système on/off, lorsque la touche est pressée, l'animation des oreilles va se faire en continue jusqu'à une nouvelle appuie de la touche qui va arrêter les oreilles.

La fonction de l'animation va tout d'abord vérifier si elle est déjà activée ou non pour lancer l'animation ou non, il va ensuite à l'aide d'une condition if else dans quelle sens bouge l'oreille pour la déplacer dans le bon sens en changeant la valeur translationX pour que l'oreille bouge, lorsqu'elle atteint la distance max donnée, le booléen mouvement est modifié et l'oreille part dans l'autre sens

```
// Fonction pour animer le mouvement de l'oreille
void animationOreilles()
{
    if (animation)
    { // Vérifie si l'animation est activée
        if (mouvement)
        { // Vérifie la direction du mouvement
            TranslationX += vitesse; // Augmente la translation vers la droite
            if (TranslationX >= maxTranslation) { // Vérifie si l'oreille a atteint la limite droite
                mouvement = false; // Change la direction du mouvement vers la gauche
            }
        } else
        {
            TranslationX -= vitesse; // Diminue la translation vers la gauche
            if (TranslationX <= -maxTranslation)
            { // Vérifie si l'oreille a atteint la limite gauche
                mouvement = true; // Change la direction du mouvement vers la droite
            }
        }

        glutPostRedisplay(); // Demande de redessiner la fenêtre pour refléter les changements de l'oreille
    }

   // glutTimerFunc(10, animateEar, 0); // Définit la prochaine itération de l'animation après 10 millisecondes
}
```



Pour ce qui est de la deuxième animation avec les touches, c'est une animation qui va faire passer le bâton tenu par le lapin d'une main à l'autre,

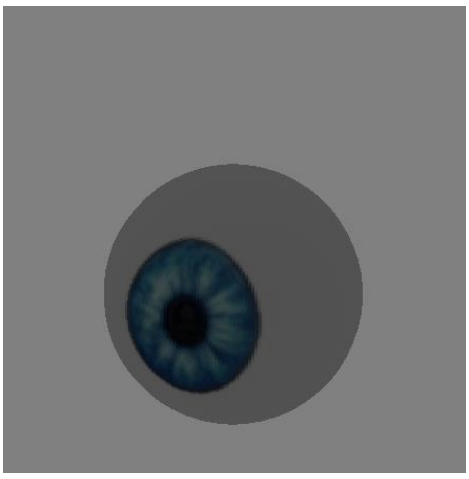
```
void animeBaton()
{
    // Si le booléen boolBaton est vrai
    if (boolBaton == true)
    {
        // Si l'angle animé est inférieur à 360 degrés
        if (angleanime < 360)
        {
            angleanime += 5; // Augmente l'angle animé de 5 degrés
            glutPostRedisplay(); // Demande le redessin de la scène pour refléter le changement d'angle
        }
    }

    // Si le booléen boolBaton est faux
    if (boolBaton == false)
    {
        // Si l'angle animé est supérieur à 180 degrés
        if (angleanime > 180)
        {
            angleanime -= 5; // Diminue l'angle animé de 5 degrés
            glutPostRedisplay(); // Demande le redessin de la scène pour refléter le changement d'angle
        }
    }
}
```

La fonction va vérifier dans quel bras est le bâton avec le booléen boolBaton pour ensuite avec un autre if vérifier quand le bâton aura atteint sa destination (tant que la valeur de l'angle pour le changement de main n'a pas atteint la valeur du if, la valeur pour déplacer le bâton va augmenter ou diminuer en fonction de la main visée)

V. Textures

Notre première texture utilisée est une texture de pupilles que l'on va appliquer sur notre sphereŒil pour avoir un œil réaliste :

Imagine de la texture



Texture sur l'œil

Nous avons créé une texture personnalisée d'une pupille avec un logiciel de retouche d'image pour avoir un résultat concluant et détaillé sans difficulté.

```
    // Dessine la sphère en utilisant des quadrilatères texturés
    glBegin(GL_QUADS);
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < m - 1; j++) {
            // Défini les coordonnées de texture et les sommets des quadrilatères
            glTexCoord2f(static_cast<float>(j) / (m - 1), static_cast<float>(i) / (n - 1));
            glVertex3f(sphere[j + (i * m)].x, sphere[j + (i * m)].y, sphere[j + (i * m)].z);

            glTexCoord2f(static_cast<float>(j + 1) / (m - 1), static_cast<float>(i) / (n - 1));
            glVertex3f(sphere[j + 1 + (i * m)].x, sphere[j + 1 + (i * m)].y, sphere[j + 1 + (i * m)].z);

            glTexCoord2f(static_cast<float>(j + 1) / (m - 1), static_cast<float>(i + 1) / (n - 1));
            glVertex3f(sphere[j + 1 + ((i + 1) * m)].x, sphere[j + 1 + ((i + 1) * m)].y, sphere[j + 1 + ((i + 1) * m)].z);

            glTexCoord2f(static_cast<float>(j) / (m - 1), static_cast<float>(i + 1) / (n - 1));
            glVertex3f(sphere[j + ((i + 1) * m)].x, sphere[j + ((i + 1) * m)].y, sphere[j + ((i + 1) * m)].z);
        }
    }
    glEnd();

    // Libère la mémoire allouée
    delete[] sphere;
}
```

glVertex3f et glTexCoord2f, créer une série de quads (quatre vertices définissant un carré) pour former la sphère. Pour chaque vertex du quad, des coordonnées de texture sont définies. Les coordonnées de texture indiquent à comment appliquer la texture sur la surface. Ces coordonnées varient entre 0.0 et 1.0, où (0, 0) correspond au coin en haut à gauche de la texture et (1, 1) au coin en bas à droite. static_cast<float>(j) / (m - 1) et static_cast<float>(i) / (n - 1) calculent les coordonnées de texture en fonction des indices i et j de la boucle. Ces valeurs sont normalisées entre 0.0 et 1.0 en divisant par m - 1 et n - 1. Définition des vertices (glVertex3f) : Pour chaque vertex du quad, les coordonnées x, y, z de la sphère sont spécifiées à l'aide de glVertex3f. Les coordonnées sphere[j + (i * m)].x, .y, .z, sphere[j + 1 + (i * m)].x, etc., correspondent aux vertices du quad.