Informatique graphique avancée, animation et rendu

Compte-rendu TP4

Sélection

Louis Jean

Master 2 IMAGINE
Université de Montpellier
N° étudiant : 21914083

22 octobre 2024

Table des matières

1 Introduction

Ce TP a pour objectif la mise en œuvre d'une sélection via des clics utilisateurs de sommets d'un maillage. La base de code du TP précédent (ARAP) a été réutilisée. Voir 2.5 pour le mode d'emploi.

2 Sphère de sélection

2.1 Trouver le point 3D associé au clic utilisateur

Pour trouver le point 3D associé à une position 2D dans l'écran, j'ai suivi plusieurs étapes :

- Récupérer les coordonnées 2D (dans l'espace de la fenêtre) du clic utilisateur
- Récupérer la coordonnée z du point, en lisant le depth buffer à la position du pixel cliqué
- Normaliser ces coordonnées (les ramener entre -1 et 1)
- Récupérer les matrices modèle-vue et projection de l'application
- Multiplier ces matrices pour obtenir la matrice MVP
- Inverser la matrice MVP
- Multiplier les coordonnées 2D normalisées par cette matrice inverse pour obtenir les coordonnées du point 3D correspondant (mais vecteur 4D)
- Diviser les coordonnées de ce point par sa 4ème composante w pour obtenir des coordonnées homogènes
- Retourner seulement les 3 premières composantes x, y, z du vecteur 4D représentant le point 3D

Pour mener à bien ces étapes, je me suis servi de la librairie `Eigen`, déjà présente dans la base de code. J'ai récupéré la position du clic grâce à un callback déjà implémenté.

```
Vec3 get3DPositionFromClick(float mouseX, float mouseY) {
    GLint viewport[4];
    GLdouble modelView[16], projection[16];
    glGetIntegerv(GL_VIEWPORT, viewport);
    glGetDoublev(GL_MODELVIEW_MATRIX, modelView);
    glGetDoublev(GL_PROJECTION_MATRIX, projection);
    float depth;
    glReadPixels(mouseX, viewport[3] - mouseY, 1, 1, GL_DEPTH_COMPONENT, GL_FLOAT, &depth);
    Eigen::Matrix4d modelViewMatrix = Eigen::Map<Eigen::Matrix4d>(modelView);
    Eigen::Matrix4d projectionMatrix = Eigen::Map<Eigen::Matrix4d>(projection);
    Eigen::Matrix4d invProjModelViewMatrix = (projectionMatrix * modelViewMatrix).inverse();
    Eigen::Vector4d normalizedPos((2.0f * mouseX) / viewport[2] - 1.0f,  // x normalisé
                      1.0f - (2.0f * mouseY) / viewport[3],  // y normalisé
                      2.0f * depth - 1.0f,                   // profondeur normalisée
                      1.0f);
    Eigen::Vector4d worldPos = invProjModelViewMatrix * normalizedPos;
    if (worldPos.w() != 0.0) {
        worldPos /= worldPos.w();
    }
    return Vec3(worldPos.x(), worldPos.y(), worldPos.z());
}
```

Dans ma première implémentation, je ne divisais pas les coordonnées obtenues par w, ce qui fait que j'avais un genre d'offset bizarre entre la position du clic et la position du centre de la sphère. J'ai mis pas mal de temps à comprendre ce problème.

2.2 Structure de données

En me basant sur la sélection rectangulaire déjà implémentée, j'ai créé un fichier `SphereSelectionTool.h`, qui introduit une structure appropriée pour stocker les informations de la sphère de sélection.

```
struct SphereSelectionTool {
    Vec3 center;
    float radius;
    bool isActive;
    bool isAdding;

    SphereSelectionTool() : radius(1.0f), isActive(false) {}

    void initSphere(float x, float y, float z) {
        center[0] = x;
        center[1] = y;
        center[2] = z;
        radius = 1.0f;
    }

    void setCenter(Vec3 center) {
        this->center = center;
    }

    void setRadius(float radius) {
        this->radius = radius;
    }

    bool contains(Vec3 point) {
        float distance = (point - center).length();
        return distance <= radius;
    }
```

La fonction `contains`, la plus utile, vérifie si un point donné se situe dans ou en dehors de la sphère, en vérifiant si la norme du vecteur reliant le centre de la sphère à ce point est plus petite que la rayon de la sphère.

Cette structure dispose aussi d'une méthode `draw` qui se base sur la fonction `drawSphere` du programme principal, que je n'ai pas incluse ici car trop longue et pas très utile.

2.3 Sélection des handles

À partir de là, toujours en m'inspirant de la sélection rectangulaire, j'ai pu écrire le code de sélection des handles.

Tout d'abord, j'ai écrit la logique de sélection des sommets, dans la fonction `setTagForVertices-InSphere`. Ici, pour tous les sommets du maillage, on vérifie s'ils sont dans la sphère. Si c'est le cas, on les ajoute au handle courant.

```
void setTagForVerticesInSphere(bool tagToSet) {
    unsigned int nVertsInside = 0;
    for(unsigned int v = 0; v < mesh.V.size(); ++v) {
        const Vec3& p = mesh.V[v].p;
        if(sphereSelectionTool.contains(p)) {
            verticesAreMarkedForCurrentHandle[v] = tagToSet;
            ++nVertsInside;
        }
    }
    std::cout << "nVertsInside = " << nVertsInside << std::endl;
}
```

Ensuite, j'ai appelé cette fonction dans `addVerticesToCurrentHandle`.

```
void addVerticesToCurrentHandle() {
    if(activeHandle < 0 || activeHandle >= numberOfHandles) return;
    setTagForVerticesInRectangle(rectangleSelectionTool.isAdding);
    setTagForVerticesInSphere(sphereSelectionTool.isAdding);
}
```

Enfin, j'ai fait en sorte de pouvoir utiliser la sélection par sphère lorsque l'on est dans le mode handle (touche `N`) en appuyant sur `SHIFT` puis en cliquant sur le maillage. J'ai donc rajouté ceci dans la fonction `mouseButtonCallback`.

```
if(mods & GLFW_MOD_SHIFT || sphereSelectionTool.isActive) {
    Vec3 clickPosInWorld = get3DPositionFromClick(x, y);
    float xWorld = clickPosInWorld[0];
    float yWorld = clickPosInWorld[1];
    float zWorld = clickPosInWorld[2];
    if(viewerState == ViewerState_EDITINGHANDLE) {
        if(state == GLFW_RELEASE) {
            sphereSelectionTool.isActive = false;
            addVerticesToCurrentHandle();
        }
        else {
            if (button == GLFW_MOUSE_BUTTON_LEFT) {
                sphereSelectionTool.initSphere(xWorld,yWorld,zWorld);
                std::cout<<"x : "<<x<<" y : "<<y<<std::endl;
                sphereSelectionTool.isAdding = true;
                sphereSelectionTool.isActive = true;
            }
        ...
```

2.4 Changement du rayon

Pour que ce soit vraiment utilisable, il fallait permettre à l'utilisateur de changer la taille de la sphère de sélection sur le volet. J'ai donc fait un callback sur la molette de la souris, pour ajuster le rayon de la sphère en direct.

```
void scroll_callback(GLFWwindow* window, double xoffset, double yoffset) {
    if (yoffset > 0) {
        sphereSelectionTool.setRadius(sphereSelectionTool.radius + 0.1f);
    } else if (yoffset < 0) {
        sphereSelectionTool.setRadius(sphereSelectionTool.radius - 0.1f);
    }

    if (sphereSelectionTool.radius < 0.1f) {
        sphereSelectionTool.setRadius(0.1f);
    }
}
```

De cette manière, faire bouger la molette vers le haut augmente le rayon de la sphère, et faire bouger la molette vers le bas réduit le rayond de la sphère.

2.5 Utilisation de la sélection par sphère

Pour utiliser la sélection par sphère, veuillez d'abord passer en mode édition de handles en appuyant sur la touche `N`. Ensuite, appuyez sur la touche `SHIFT` et cliquez sur le maillage. Maintenez le clic, et faites défiler la molette pour changer le rayon de la sphère. Une fois la bonne taille souhaitée atteinte, relâchez le clic, et vos points seront sélectionnés.

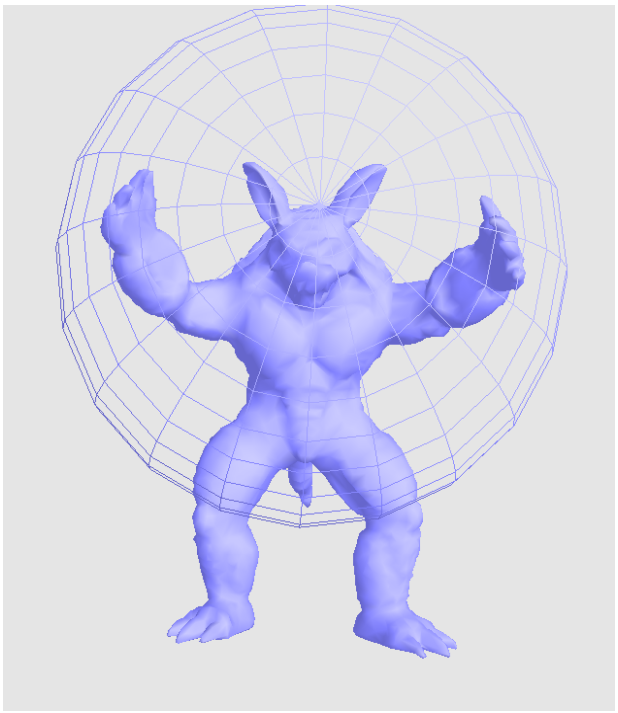
2.6 Résultats

Voici deux exemples de sélection par sphère, illustrant le fonctionnement de l'outil de sélection.

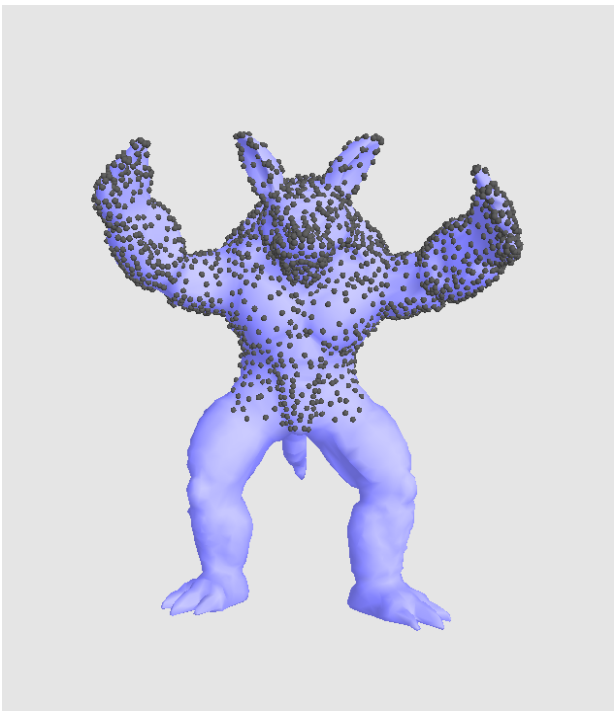
(a) Sphère de sélection

(b) Points sélectionnés

Figure 1: Premier exemple de sélection par sphère



(a) Sphère de sélection



(b) Points sélectionnés

Figure 2: Second exemple de sélection par sphère

3 Suite du TP

Malgré le fait que je n'ai pas eu le temps de poursuivre avec les étapes suivantes du TP, cette première question m'a permis de renforcer mes compétences en manipulation de code existant, en intégrant des nouvelles fonctionnalités dans un environnement complexe. J'ai notamment compris comment manipuler les transformations entre les espaces 2D et 3D, ce qui me sera sans doute très utile plus tard.

4 Conclusion

En m'appuyant sur la sélection rectangulaire déjà existante, j'ai pu réaliser un outil de sélection sphérique flexible.

Merci pour le temps et l'attention que vous avez consacrés à la lecture de ce compte-rendu.