Université de Montpellier
Master 2 - IMAGINE

ClayMotion

Modélisation de sculpture avec utilisation du "Leap Motion"

Rapport de projet
Projet informatique — HAI928I

Étudiants :
M. Khélian LARVET
M. Sylvain LECLERC

Encadrants :
Mme. Noura FARAJ
Mme. Nancy RODRIGUEZ
M. Olivier STRAUSS

Année : 2022

Table des matières

Description du projet

Présentation

Notre principal objectif pour ce projet était de créer une simulation interactive permettant de manipuler la matière avec nos mains. Nous avons donc choisi dans un premier temps de nous diriger vers une simulation de poterie où l'on peut générer, sculpter, déformer des maillages en temps réel.

La poterie étant un domaine exigeant et difficile à mettre en place en 3D et surtout en temps réel, nous nous sommes davantage dirigés vers des opérateurs de manipulation. Notre application s'apparente donc plus à un logiciel tel que "Blender".

Un utilisateur ayant notre application peut donc :
— **Créer** des maillages primitifs avec un certain mouvement ;
— **Visualiser** la structure des maillages créés avec un "Shader Wireframe"
— **Interagir** avec les maillages créés à l'aide du "LeapMotion" ("Grab").
— **Modifier** les maillages créés avec des operateurs (déformation, lissage) ;

Technologies utilisées

Nous avons utilisé **Unity** et un **capteur "LeapMotion"** afin de faciliter notre travail dans la reconnaissance des mains et dans le suivi des mains dans l'espace. Ces technologies nous permettent donc d'accéder facilement aux positions de chaque membre composant notre main, leurs vitesses ainsi que leurs directions.

Mode d'emploi

Installation

Notre application a été développée sous Unity v2022.1.16f1 et LeapMotion Tracking v6.2.1.
Assurez vous d'avoir installé le logiciel "UltraLeap Tracking" à l'adresse suivante :
`https://developer.leapmotion.com/tracking-software-download`

Attention, il faut bien configurer le logiciel dans la catégorie tracking avec "Desktop".

Fonctionnalités

Voici l'ensemble des fonctionnalités de notre application :

- Si vous orientez votre paume de main gauche vers la caméra, celle-ci affichera un menu qui sera alors "cliquable" avec la main droite. Lorsqu'une option est activée ou sélectionnée, son cadre s'illumine. Un code couleur est appliqué sur ces options :
  - Les options vertes indiquent la génération.
  - Les options bleues indiquent la modification.
  - Les options rouges indiquent la suppression.

- Si aucun maillage n'a été créé, les seules options activables seront les "vertes". Vous aurez alors le choix entre la création d'un cube, d'une sphère ou d'un cylindre. Une fois l'option sélectionnée, effectuez un "Pinch" (pouce et index joint) dans vos deux mains pour commencer la création. Tant que vous maintenez le "Pinch" le maillage sera modifiée (en blanc). Une fois le "Pinch" laché, l'objet sera créé (en marron clair) et vous pourrez intéragir avec.

- Une fois qu'au moins un maillage a été créé, les opérateurs de modifications deviennent activables :

  - L'option représentée par "quatre flèches" permet de faire apparaître les poignées de rotation sur l'objet le plus proche. Ces poignées peuvent être manipulées pour effectuer des rotations précises.

  - L'option représentée par "un cercle et un point" représente l'opérateur de déformation local. Celui-ci permet d'indiquer dans un premier temps l'objet le plus proche qui va pouvoir être modifié (cube rouge). Puis, si un "Pinch" est effectué assez proche du maillage cible, celui-ci va pouvoir "étirer" les sommets du maillage tant que le "Pinch" est actif.

  - L'option représentée par "une vague" représente l'opérateur de lissage local. Celui-ci permet d'indiquer dans un premier temps d'indiquer l'objet le plus proche qui va pouvoir être modifié (cube rouge). Puis, si un "Pinch" est effectué assez proche du maillage cible, celui ci va automatiquement lisser les points dans le rayon d'action bleu et passer rouge pour indiquer son application.

- La croix rouge permet de supprimer tous les maillages créés.

Caractéristiques significatives

Création de maillage

C'est le point d'entrée de notre application : pouvoir créer de la matière intuitivement et rapidement. Pour se faire nous avons mis en place nos propre générateur de maillage primitifs permettant de générer les tableaux de sommets, triangles et normales. Une fois le maillage créé, nous l'envoyons dans Unity sous la forme d'un "GameObject" pour avoir les composants de base ("Collider", "Render", "Transform").

L'utilisateur peut donc à l'aide d'un menu, sélectionner un objet primitif et le créer en effectuant un "Pinch" (pouce et index joint) dans chaque main. Tant que l'utilisateur reste dans l'état "Pinch" l'objet n'est pas créé (en blanc) et peut donc être modifié au niveau de la position, de la rotation et de la taille.

Création d'un cube

Création d'une sphère

Création d'un cylindre

```
// Vector3 : rightPinchPos, leftPinchPos
// Float : rightCoef, leftCoef
float size = (leftPinchPos - rightPinchPos).magnitude;

// Start Pinch
if (!isPinching && leftCoef < PINCH_DISTANCE_LOW && rightCoef < PINCH_DISTANCE_LOW){
    isPinching = true;
    // Création d'un objet temporaire (blanc) ...
}
// While Pinch
if (isPinching){
    // Modification de l'objet temporaire (blanc) ...
}
// End Pinch
if (isPinching && leftCoef >= PINCH_DISTANCE_HIGH && rightCoef >= PINCH_DISTANCE_HIGH){
    isPinching = false;
    GameObject go;
    go.transform.position = (leftPinchPos + rightPinchPos) / 2;
    go.transform.localScale = new Vector3(size, size, size);
    go.transform.forward = leftPinchPos - rightPinchPos;
    // ...
}
```

Opérateur de modification

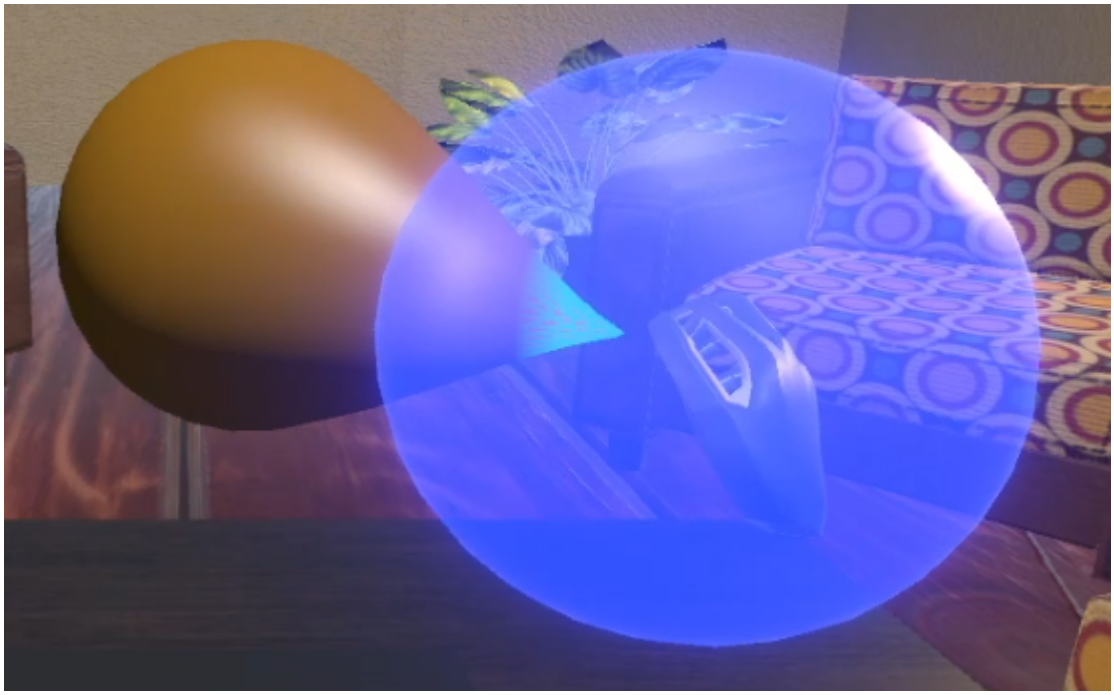
Une fois qu'un maillage est créé, les opérateurs de modifications deviennent utilisables. Notre application propose deux opérateurs : la déformation et le lissage.

Opérateur de déformation locale

Cet opérateur permet de "tirer" la matière proportionnellement dans un certain sens. En effet, lorsque cet outil est actif, si l'utilisateur "Pinch" avec la main droite, un rayon d'action apparaît et tous les points du maillage cible se trouvant dans ce rayon d'action vont être déplacé. Ce déplacement est effectué selon un coefficient, définit par la distance à la zone de pincement initiale, et une direction définit par le déplacement entre la zone de départ et la zone d'arrivée du pincement :

```
// Vector3 : PinchStart, PinchEnd, V
// float : RAYON_ACTION

// Pour chaque sommet V du maillage compris dans le RAYON_ACTION
float coef = 1 - ((PinchStart - V).magnitude / RAYON_ACTION);
Vector3 targetDirection = PinchEnd - V;
V += targetDirection * coef;
```



Application d'une déformation locale

Opérateur de lissage locale

Cet opérateur permet de lisser la surface d'un objet en fonction un lissage Laplacien. En effet, lorsque cet outil est actif, un rayon d'action apparaît et si l'utilisateur "Pinch" ce rayon devient rouge pour indiquer son activation. Lors de son activation, tous les points du maillage cible se trouvant dans ce rayon d'action vont subir un lissage. Ce lissage consiste en un déplacement effectué pour chaque sommet en direction de la moyenne des positions de son 1-voisinage avec un certain coefficient que nous avons fixé à 0.5 Ce coéficient pourrait dépendre de la distance par rapport à la zone d'activation du lissage, comme pour l'opérateur précédent, mais cette fonctionnalité n'est que peu pertinente :

```
List<List<int>> oneRing = CollectOneRing(tmpVertices, tmpTriangles);

// Pour tous les sommets i dans le rayon d'action :
resVertices[i] = SmoothVertex(i, 0.5f, oneRing, tmpVertices);
// ...
```

```
// public Vector3 SmoothVertex(...)

List<int> oneRingVerts = oneRing[vertInd]; // Liste des 1-voisins du sommet vertInd
Vector3 smoothPos = vertices[vertInd];     // Initialiser un sommet smooth (coef de 1.0)

// Pour chaque 1-voisin du sommet, ajouter sa position au sommet smooth
foreach (int oneRingVert in oneRingVerts) { smoothPos += vertices[oneRingVert]; }

// Moyenne des sommets adjacents (+1 car vertInd n'est pas compté dans le oneRing)
smoothPos /= (oneRingVerts.Count + 1);

// Appliquer le coefficient de lissage (de 0.5) sur le sommet smooth
smoothPos = Vector3.Lerp(vertices[vertInd], smoothPos, coef);
```

Avant lissage

Après lissage

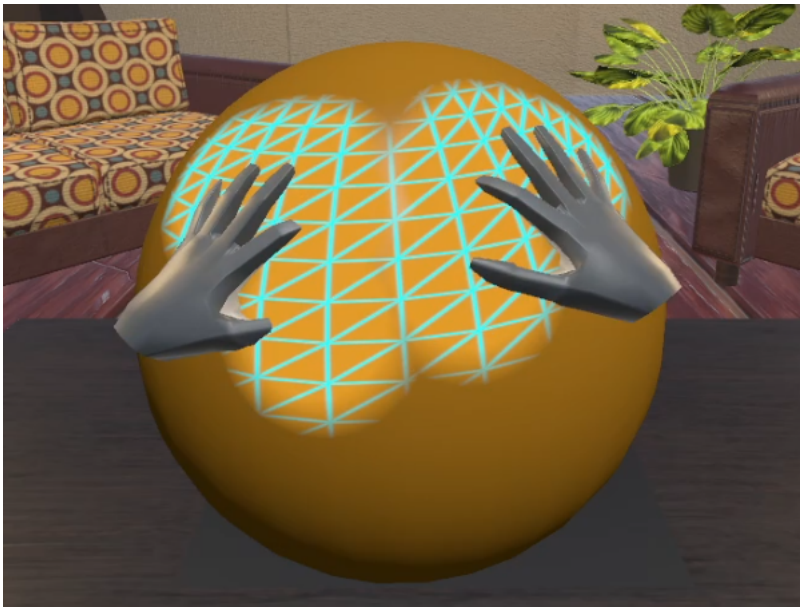
Le "Shader Wireframe"

Afin de visualiser la structure de nos objets de manière interactive avec le "LeapMotion", nous avons eu l'idée de faire un "Shader Wireframe" qui s'activerait à une certaine distance des doigts du "LeapMotion".

Pour ce faire, nous avons dans un premier temps utilisé un "Geometry Shader" pour calculer la distance entre un fragment et les arêtes du triangle. Puis nous avons fait en sorte d'envoyer les coordonnées de chaque doigts du "LeapMotion" au "Fragment Shader". Enfin, nous avons mis en place deux composante :

— une coloration "Wireframe" basée sur la distance entre les fragments et les arêtes des triangles.
— un rendu "Blinn Phong" basique ("ambiant", "diffuse", "specular").

Ces deux composantes sont liées par une interpolation linéaire basée sur la distance entre les doigts et les triangles du maillage.



Visualisation avec notre "Shader Wireframe"

Difficultés et perspectives d'amélioration

Difficultés rencontrées

Notre première difficulté a été **l'apprentissage du "LeapMotion"** et de ses divers scripts. En effet, nous avons appris à manipuler les interactions définies dans le "LeapMotion" permettant par exemple d'attraper un objet pour le déplacer. Nous avons également appris à utiliser les positions des différents membres de la main pour effectuer des mouvements particuliers. Le plus simple étant le "Pinch" lorsque l'index et le pouce se rejoignent.

La seconde difficulté fût la **création d'un "Shader Wireframe"** en Unity. En effet, nous avons mis du temps à nous adapter à la syntaxe Unity qui est différente des shaders GLSL, notamment pour le passage des valeurs entre le CPU et le GPU.

Une autre difficulté a été celle de **créer nos propres maillages primitifs**. En effet, nous avons toujours manipulé des objets dont les sommets sont partagés par les triangles. Or dans Unity, chaque triangle possède 3 uniques sommets. Cet aspect à été particulièrement problématique lorsque nous avons voulu implémenter un opérateur de lissage qui nécessite le 1-voisinage.

Enfin, nous avons dû **simplifier au maximum nos opérateur de modification** pour les voir en temps réel. En effet, en fonction de la taille des maillages, les opérateurs peuvent être lourd à exécuter. C'est pourquoi nous avons choisi d'appliquer nos opérateurs de manière local avec des maillages légèrement subdivisés.

Perspectives d'améliorations

Plusieurs améliorations restent possibles sur ce projet afin de le rendre plus complet et attrayant. Avec par exemple :

— **L'utilisation d'un casque VR** : Le "LeapMotion" pouvant s'utiliser sur un casque VR, celui ci nous permettrait d'avoir un espace d'interaction plus grand avec de nouvelles possibilités.

— **Ajouter de nouveaux opérateurs de modification** : Pour l'instant, notre application compte seulement deux opérateurs de modification (déformation, lissage). Mais nous pouvons en ajouter de nouveaux pour proposer de nouvelles manières de modifier un maillage.

— **Ajouter de manière interactive des options** : Notre espace étant restreint avec un seul capteur, nous n'avons pas réussi à rajouter de manière intuitive des options. Ces options permettraient par exemple de modifier la portée des opérateurs ou encore le nombre de subdivisions pour les maillages primitifs de départ.

— **Ajouter des fonctions pour combiner les maillages et les sauver** : Pour l'instant notre application est statique et ne permet pas vraiment de modéliser des objets pour les sauver. Mais nous pourrions rajouter cette dimension à notre application.