ISAMM Institut Supérieur des Arts Multimédia de la Manouba

UNITY PHYSICS

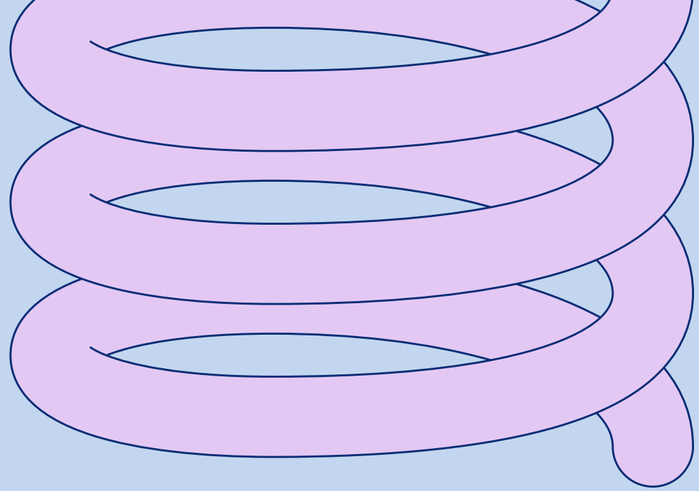
Skin Simulation

Encadré par : M Ajlani Hosni

Réalisé par : Chaouachi Eya
Horri Ahmed

Année universitaire: 2023/2024

PLAN



I. INTRODUCTION

- Contexte , Problématique et objectifs

1. Contexte

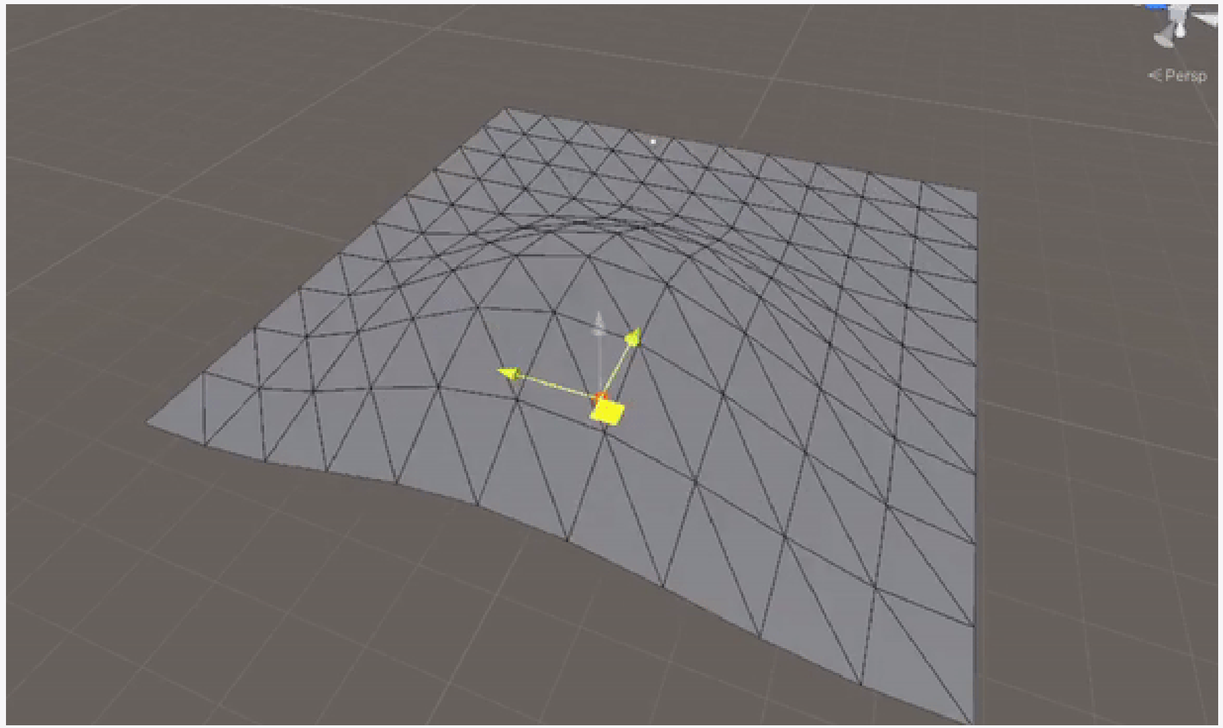
- La simulation de la peau est une avancée technologique majeure dans le domaine de la simulation. Cette technologie trouve des applications diverses, notamment dans les industries du divertissement, de la santé et de la recherche scientifique. La simulation de la peau peut être utilisée pour créer des effets visuels plus réalistes dans les jeux vidéo, simuler des interventions chirurgicales, et faciliter la recherche sur les propriétés biomécaniques de la peau.

2. Problématique

- Cependant, malgré les progrès réalisés dans le domaine de la simulation, la modélisation précise de la déformation de la peau dans des zones spécifiques reste un défi. Les techniques actuelles ne parviennent pas toujours à reproduire de manière réaliste les mouvements et les plis de la peau dans des conditions variées.

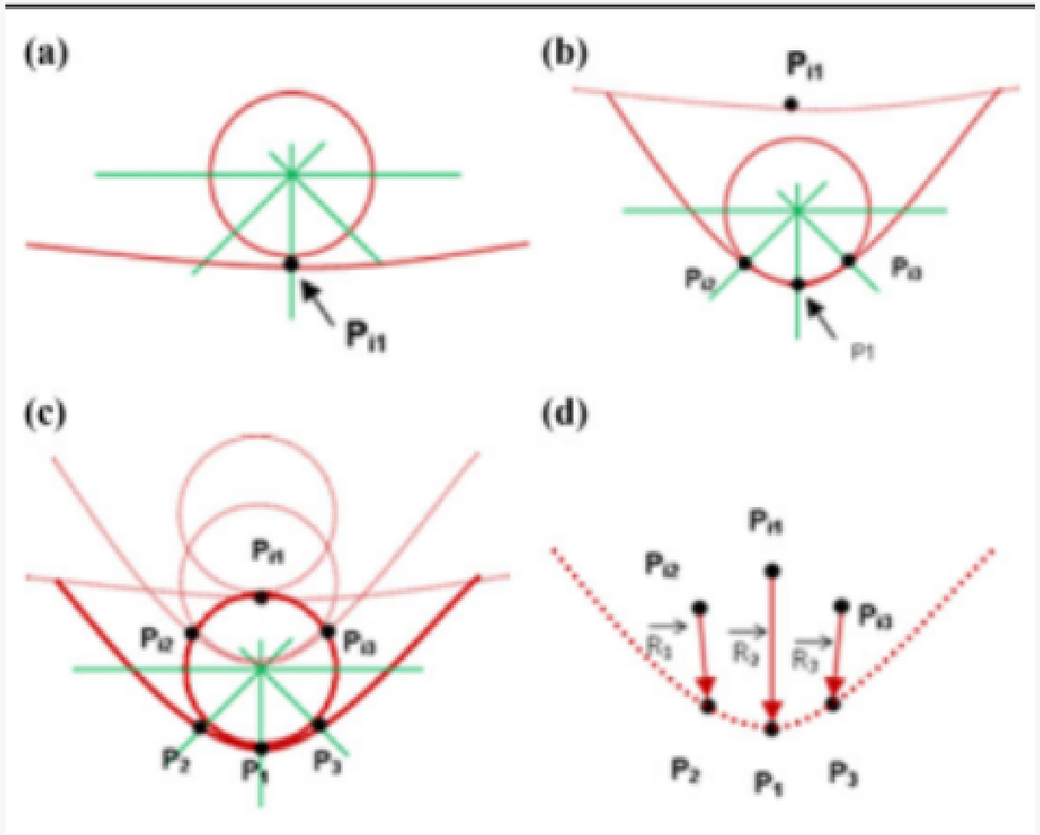
3. Objectifs

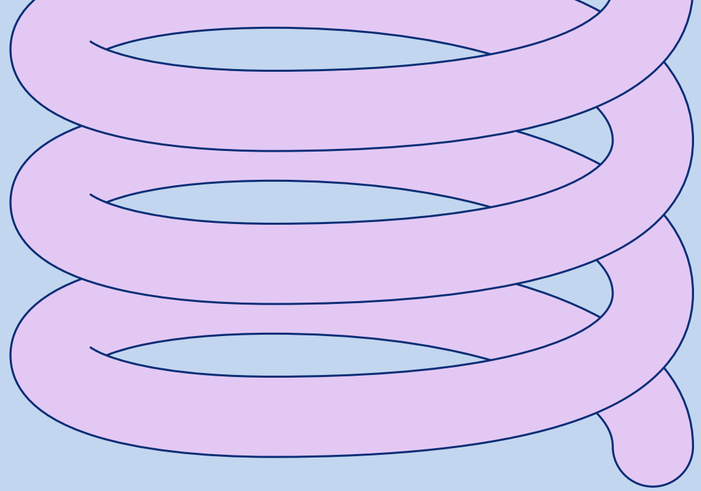
- L'objectif principal de notre projet est de développer une simulation de la peau hautement réaliste qui peut être intégrée dans divers contextes, offrant une déformation précise dans des zones sélectionnées.



  - Force Gravitationnelle
  - Force de Frottement de l'Air
  - Forces des Ressorts
  - Force de Viscosité (Résistance de l'Air)
  - Feedback Force haptique

- Les vibrations des particules au niveau d'un point de la peau s'orientent selon la direction du déplacement, reflétant ainsi la déformation cutanée





II. MISE EN ŒUVRE

Conception et modélisation de la simulation , Intégration dans Unity

Conception et modélisation de la simulation dans Unity :

Création des sphères représentant le tissu

A. le maillage

1. crée le maillage initial de particules qui représentent le tissu.

```
{
    spheres = new GameObject[gridSizeX, gridSizeY];
    velocities = new Vector3[gridSizeX, gridSizeY];
    forces = new Vector3[gridSizeX, gridSizeY];

    for (int i = 0; i < gridSizeX; i++)
    {
        for (int j = 0; j < gridSizeY; j++)
        {
            Vector3 position = new Vector3(i * espacement, - j * espacement, 0);
            spheres[i, j] = GameObject.CreatePrimitive(PrimitiveType.Sphere);
            spheres[i, j].transform.localScale = Vector3.one * 0.4f;

            spheres[i, j].transform.position = position;
            spheres[i, j].name = "Objet [" + i + "," + j + "]";
        }
    }
}
```

- 2. initialisation de la force du ressort :

```
Vector3 CalculateSpringForce(int i, int j)
{
    Vector3 springForce = Vector3.zero;
```

- 3. Calcul des forces :

```
void CalculateForces(int i, int j)
{
    forces[i, j] = Vector3.zero;

    Vector3 forceGravity = gravity * masse;
    Vector3 forceFrottementAir = -frottementAir * velocities[i,j];

    forces[i, j] += forceGravity;
    forces[i, j] += forceFrottementAir;

    forces[i, j] += CalculateSpringForce(i, j);
    forces[i, j] += CalculateWindForce(i, j);
}
```

- 4. Calcul des forces des ressorts horizontaux et verticaux :

```
// 4 Ressorts horizontales et verticales
if (i > 0)
{
    springForce += CalculateSpringForce(i, j, i - 1, j, longueurRessort1, K1, amortissementRessort1);
    DrawSpring(i, j, i - 1, j);
}
if (i < gridSizeX - 1)
{
    springForce += CalculateSpringForce(i, j, i + 1, j, longueurRessort1, K1, amortissementRessort1);
    DrawSpring(i, j, i + 1, j);
}
if (j > 0)
{
    springForce += CalculateSpringForce(i, j, i, j - 1, longueurRessort1, K1, amortissementRessort1);
    DrawSpring(i, j, i, j - 1);
}
if (j < gridSizeY - 1)
{
    springForce += CalculateSpringForce(i, j, i, j + 1, longueurRessort1, K1, amortissementRessort1);
    DrawSpring(i, j, i, j + 1);
}
```

- 5. Calcul des forces des ressorts diagonaux :

```
// Ressorts diagonaux


if ((i % 2 == 0 && j % 2 != 0) || (i % 2 != 0 && j % 2 == 0))
{
    if (j < gridSizeY - 1 && i > 0)
    {
        springForce += CalculateSpringForce(i, j, i - 1, j + 1, longueurRessort2, K2, amortissementRessort2);
        DrawSpring(i, j, i - 1, j + 1);
    }
    if (j < gridSizeY - 1 && i < gridSizeX - 1)
    {
        springForce += CalculateSpringForce(i, j, i + 1, j + 1, longueurRessort2, K2, amortissementRessort2);
        DrawSpring(i, j, i + 1, j + 1);
    }
    if (j > 0 && i > 0)
    {
        springForce += CalculateSpringForce(i, j, i - 1, j - 1, longueurRessort2, K2, amortissementRessort2);
        DrawSpring(i, j, i - 1, j - 1);
    }
    if (j > 0 && i < gridSizeX - 1)
    {
        springForce += CalculateSpringForce(i, j, i + 1, j - 1, longueurRessort2, K2, amortissementRessort2);
        DrawSpring(i, j, i + 1, j - 1);
    }
}
```

si i est pair et j est impair ou si i est impair et j est pair

- 7. Dessin des ressorts pour la visualisation :

```
void DrawSpring(int i1, int j1, int i2, int j2)
{
    Debug.DrawLine(spheres[i1,j1].transform.position, spheres[i2,j2].transform.position,Color.red);
}
```

- 8. Application des forces :

```
void ApplyForce(int i, int j, Vector3 force)
{
    // Update velocity using F = ma => a = F/m


    if (i == 0 || j == 0 || i == gridSizeX || j == gridSizeY)

        return;
    Vector3 acceleration = forces[i,j] / masse;
    velocities[i, j] += acceleration * Time.deltaTime;

    // Update position using Euler method: x = x0 + v * dt
    spheres[i, j].transform.position += velocities[i, j] * Time.deltaTime;
}
```

```
if (i == 0 || j == 0 || i == gridSizeX || j == gridSizeY)
```

Cette condition vérifie si la particule se trouve sur le bord du tissu. Si l'un de ses indices (i ou j) est égal à 0 (indiquant la première ligne ou colonne) ou égal à gridSizeX ou gridSizeY (indiquant la dernière ligne ou colonne), la méthode se termine prématurément sans appliquer de force.

- 9. Vérification des collisions entre le point d'interface haptique (HIP) et le tissu :

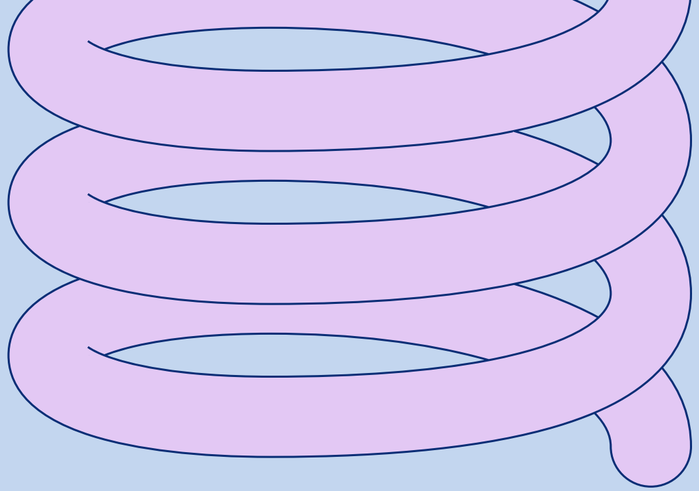
```
void CheckCollisionsWithHIP()
{
    // Perform raycasting to detect collisions between the HIP and the Tissu
    RaycastHit hit;
    Vector3 hipPosition = hapticInterfacePoint.transform.position;

    for (int i = 0; i < tissu.gridSizeX; i++)
    {
        for (int j = 0; j < tissu.gridSizeY; j++)
        {
            Vector3 particlePosition = tissu.spheres[i, j].transform.position;

            // Perform raycast from the HIP to the particle
            if (Physics.Raycast(hipPosition, particlePosition - hipPosition, out hit))
            {
                // Handle collision between the HIP and the particle
                HandleCollision(hit, i, j);
            }
        }
    }
}
```

Cette fonction parcourt toutes les particules du tissu et vérifie s'il y a une collision entre chaque particule et le point d'interface haptique. Si une collision est détectée, une force est appliquée à la particule en fonction de la direction de la collision et de la magnitude de force spécifiée.

- 9. Gérer une collision détectée entre le point d'interface haptique (HIP) et une particule du tissu

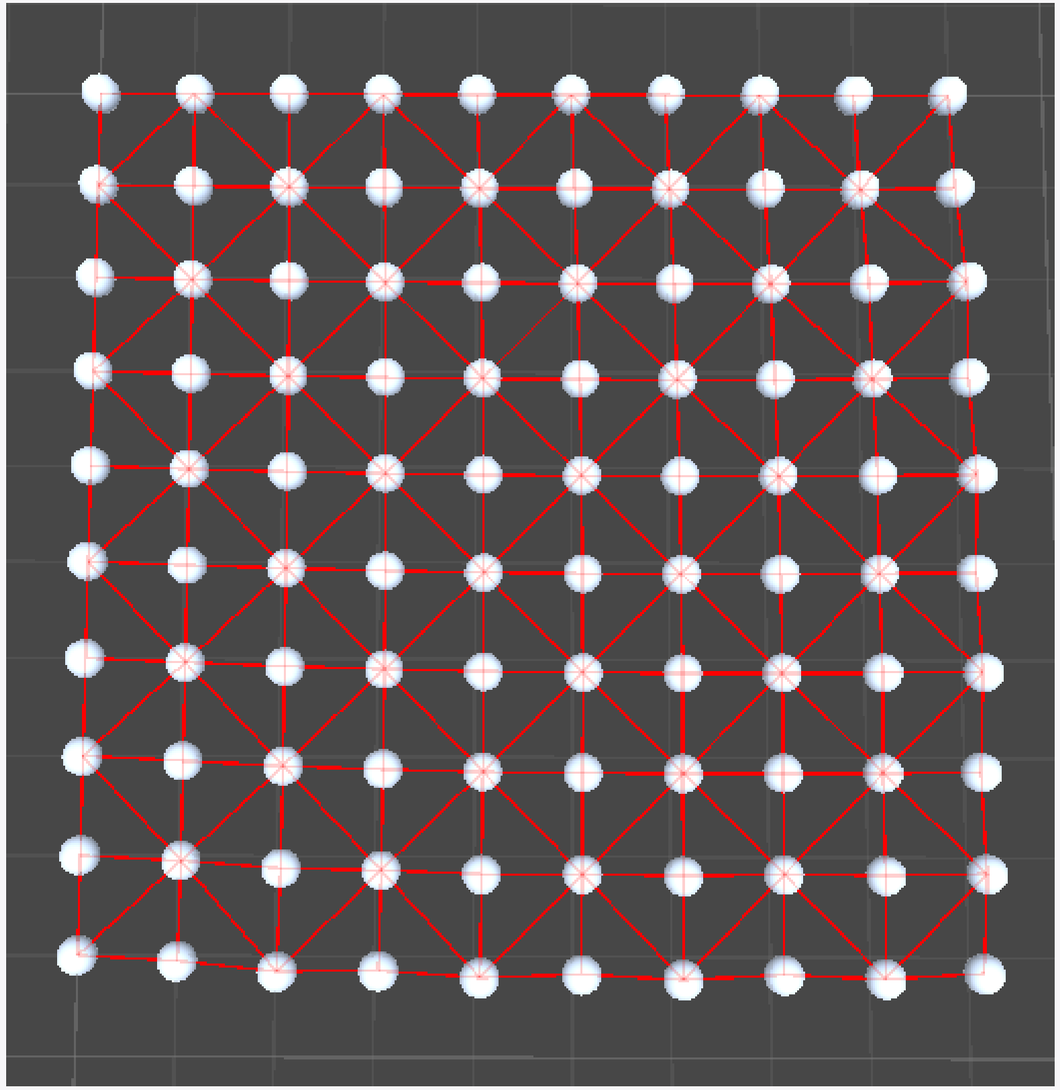
```
void HandleCollision(RaycastHit hit, int particleIndexX, int particleIndexY)
{
    // Example: Apply a force to the particle based on the collision
    Vector3 forceDirection = hit.normal.normalized; // Direction opposite to the collision normal
    float forceMagnitude = 10f; // Adjust this value based on the desired force strength
    Vector3 force = forceDirection * forceMagnitude;

    // Apply the force to the particle
    // Assuming Tissu script has a method to apply forces to particles
    tissu.ApplyForce(particleIndexX, particleIndexY, force);
}
```

Cette fonction calcule la direction et la magnitude de la force à appliquer à une particule du tissu en réponse à une collision avec le point d'interface haptique, puis applique cette force à la particule via la méthode ApplyForce de la classe Tissu.

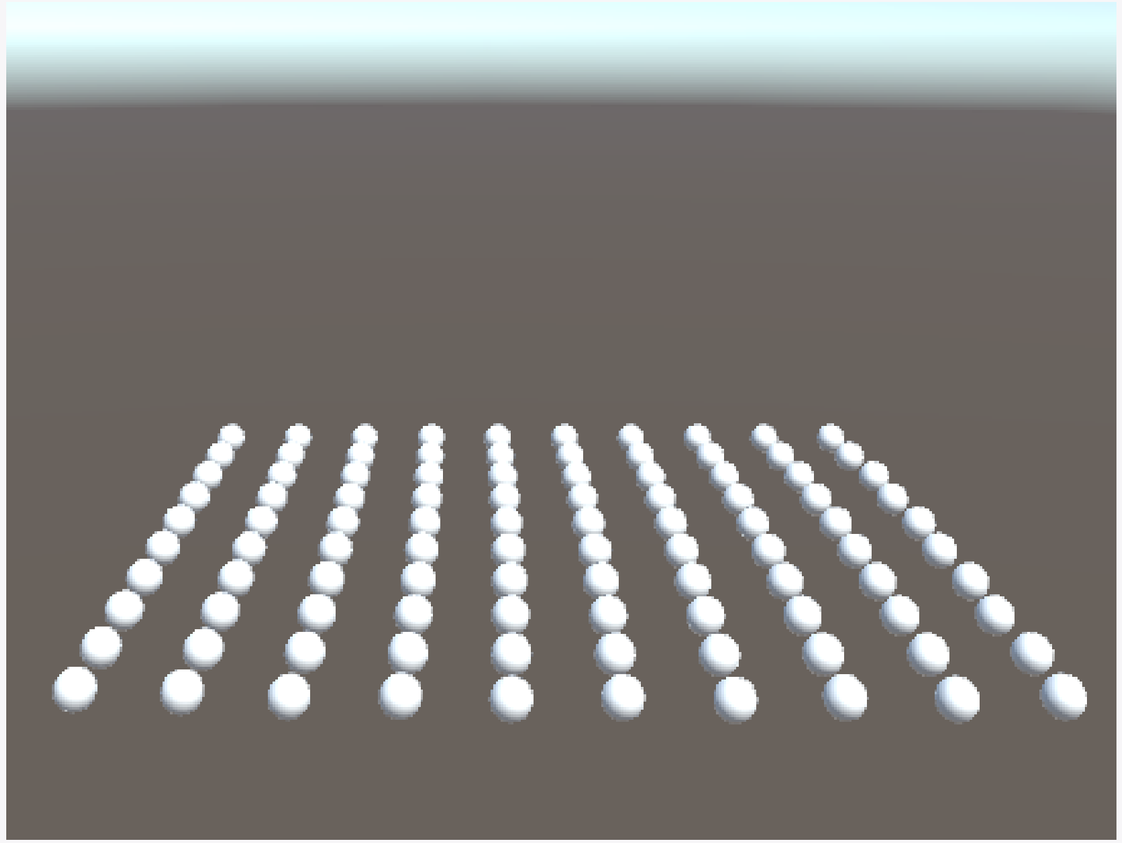


III. RÉALISATION

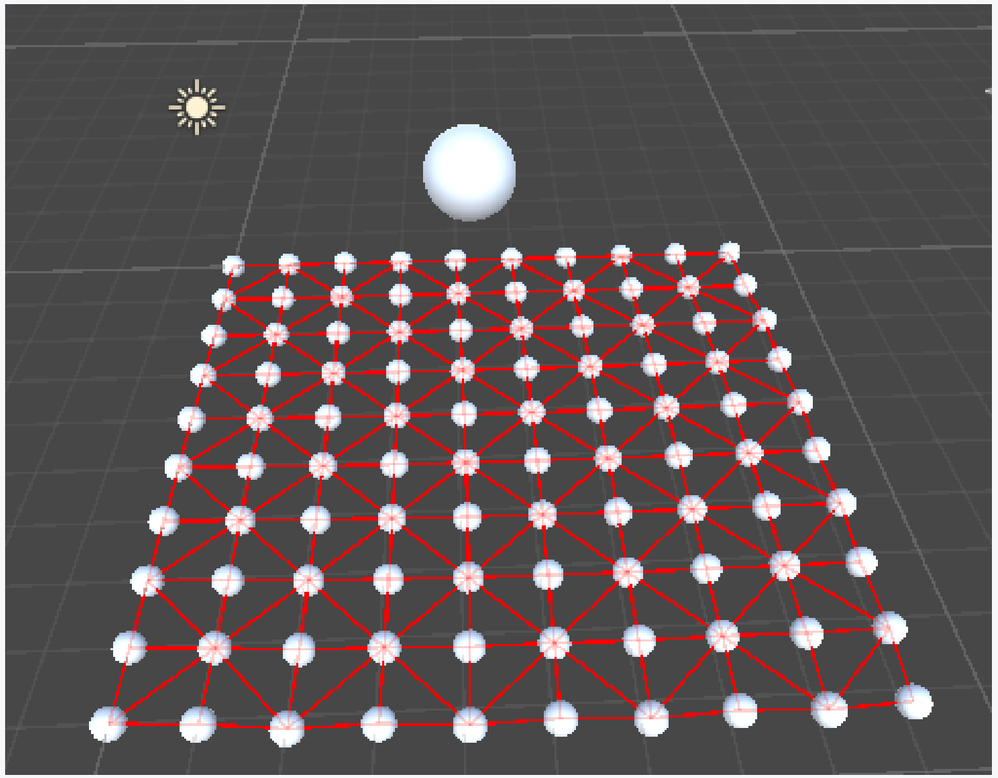
Conception et modélisation de la simulation , Intégration dans Unity



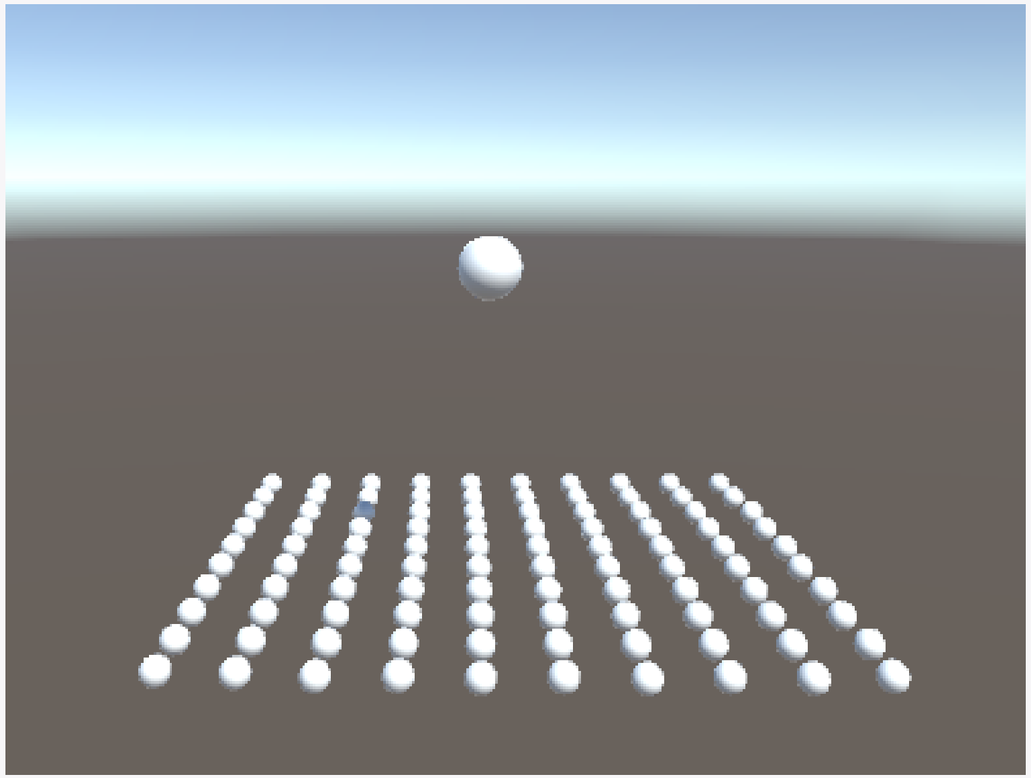
Scéne Maillage



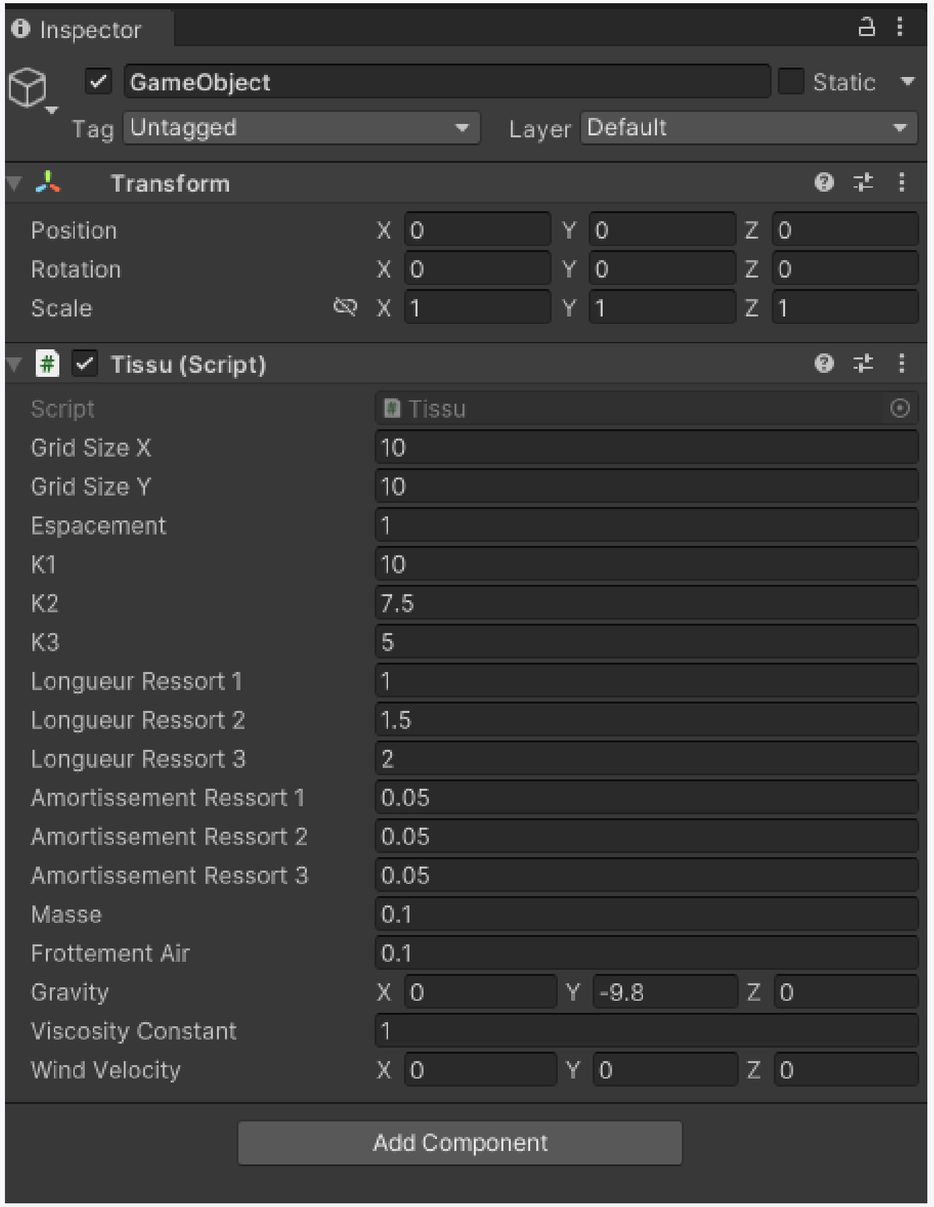
GamePlay Maillage



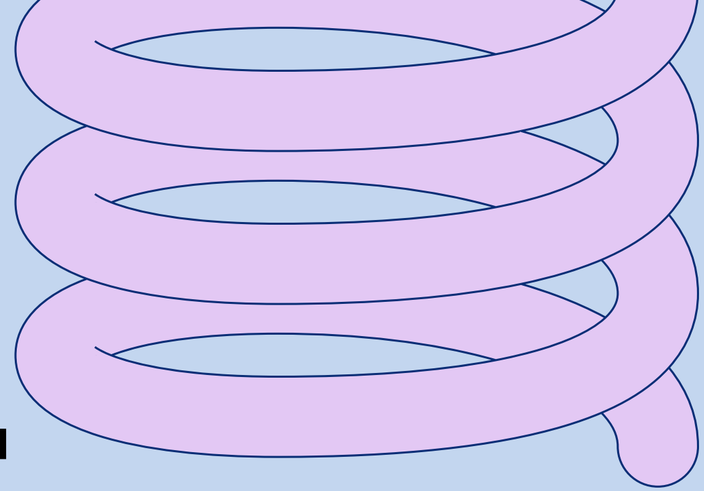
Scéne Maillage



GamePlay Maillage

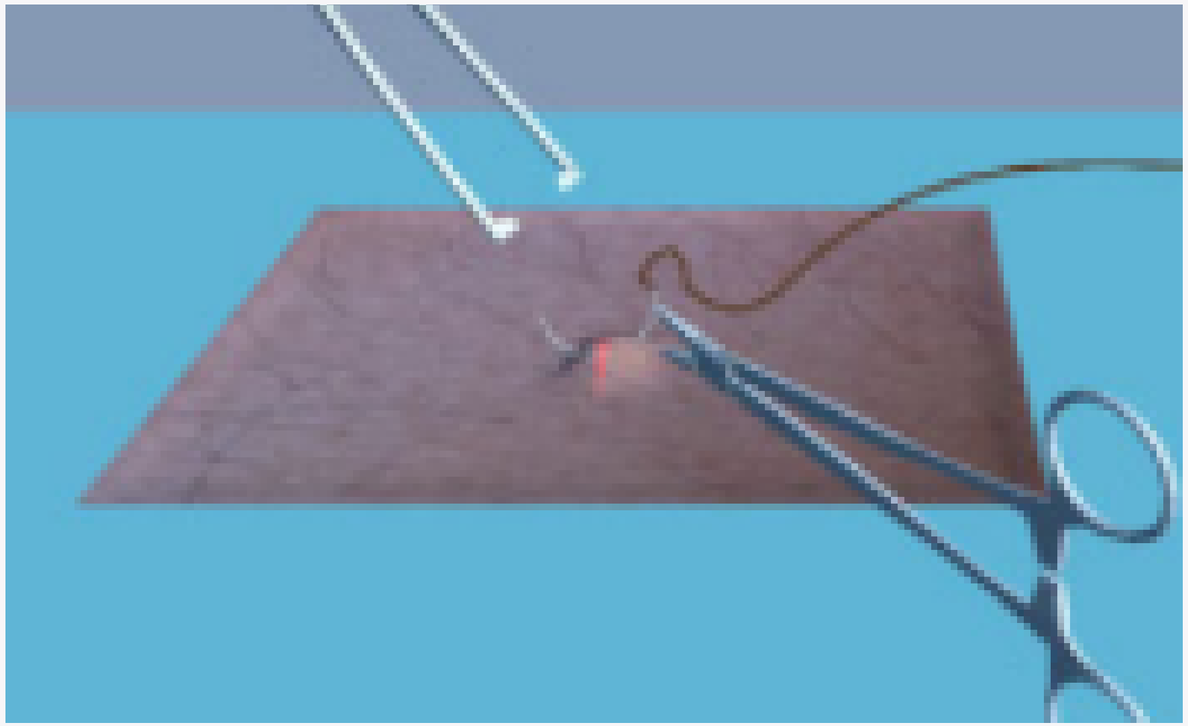

Inspector
GameObject
Static
Tag Untagged
Layer Default
Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Tissu (Script)
Script Tissu
Grid Size X 10
Grid Size Y 10
Espacement 1
K1 10
K2 7.5
K3 5
Longueur Ressort 1 1
Longueur Ressort 2 1.5
Longueur Ressort 3 2
Amortissement Ressort 1 0.05
Amortissement Ressort 2 0.05
Amortissement Ressort 3 0.05
Masse 0.1
Frottement Air 0.1
Gravity X 0 Y -9.8 Z 0
Viscosity Constant 1
Wind Velocity X 0 Y 0 Z 0
Add Component

Inspector



VI. CONCLUSION ET PERSPECTIVES

- Notre projet a souligné l'importance de comprendre les propriétés viscoélastiques de la peau. L'intégration de méthodes mathématiques a été cruciale pour obtenir une représentation précise de la déformation cutanée.
- Les principes établis ici contribuent à repousser les frontières de la modélisation virtuelle et à créer des expériences plus immersives et réalistes. Il ouvre également des opportunités pour des applications innovantes dans des domaines tels que la médecine, l'industrie du divertissement et la recherche scientifique.



Merci pour votre Attention