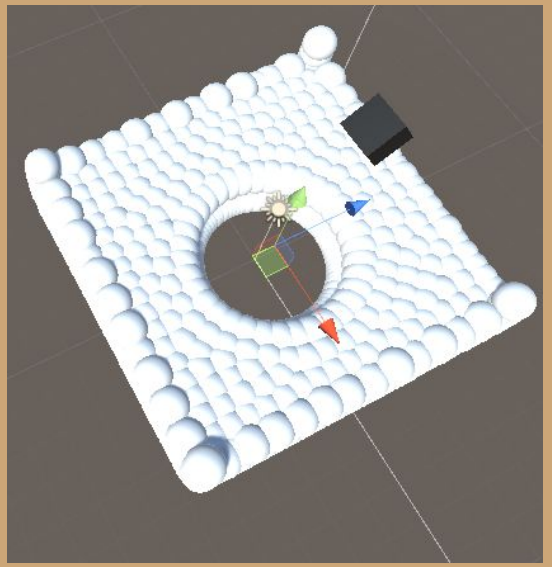
Fluid Simulation

Raphaël Jocteur
Créajeux 2019

Sommaire

- Objectifs
- Problématiques
- Mise en pratique
- Démonstration



❖ Objectifs

Permettre de simuler la physique d'un fluide :

- ❏ Créer des milliers de "particules" qui forment notre fluide
- ❏ Créer un système physique entre ses "particules"
- ❏ Créer des collisions avec des éléments extérieurs aux "particules"

❖ Problématiques

❖ Créer des milliers de “particules” qui forment notre fluide

➔ Moyens de paralléliser les calculs sur un grand nombre d’éléments :

- languages GPU (CUDA, ...)
- UNITY ECS*
- UE4 ECS*

Le choix a été fait de retenir UNITY ECS.

*ECS : Entity Components System

❖ Créer un système physique entre ces “particules”

Wikipedia lien :

$$\frac{\partial \vec{v}}{\partial t} + \left(\vec{v} \cdot \overrightarrow{\operatorname{grad}}\right) \vec{v} = -\frac{\overrightarrow{\operatorname{grad}} P}{\rho} + \overrightarrow{f_{\mathrm{m}}} + \nu \Delta \vec{v}$$

Sph Fluid in computer Graphics pdf1 and pdf2 :

for all *particle i* **do**
  find neighbors j
for all *particle i* **do**
  $\rho_i = \sum_j m_j W_{ij}$
  compute p_i using ρ_i (e.g. Eq. (9))
for all *particle i* **do**
  $\mathbf{F}_i^{pressure} = -\frac{m_i}{\rho_i} \nabla p_i$ (e.g. Eq. (6))
  $\mathbf{F}_i^{viscosity} = m_i \nu \nabla^2 \mathbf{v}_i$ (e.g. Eq. (8))
  $\mathbf{F}_i^{other} = m_i \mathbf{g}$
  $\mathbf{F}_i(t) = \mathbf{F}_i^{pressure} + \mathbf{F}_i^{viscosity} + \mathbf{F}_i^{other}$
for all *particle i* **do**
  $\mathbf{v}_i(t + \Delta t) = \mathbf{v}_i(t) + \Delta t \mathbf{F}_i(t)/m_i$
  $\mathbf{x}_i(t + \Delta t) = \mathbf{x}_i(t) + \Delta t \mathbf{v}_i(t + \Delta t)$

❖ Créer des collisions avec des éléments extérieurs

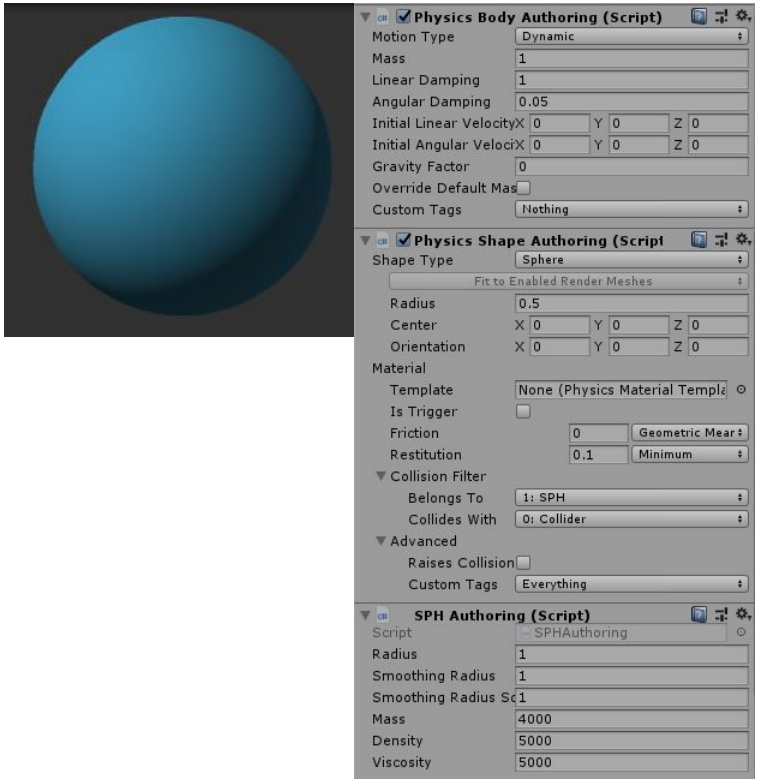
Utilisation de unity physics en Bêta :

- Utilise un système ecs pour gérer la physics
- Embarque avec lui la plupart des colliders de base
- Peu documenté et difficilement accessible

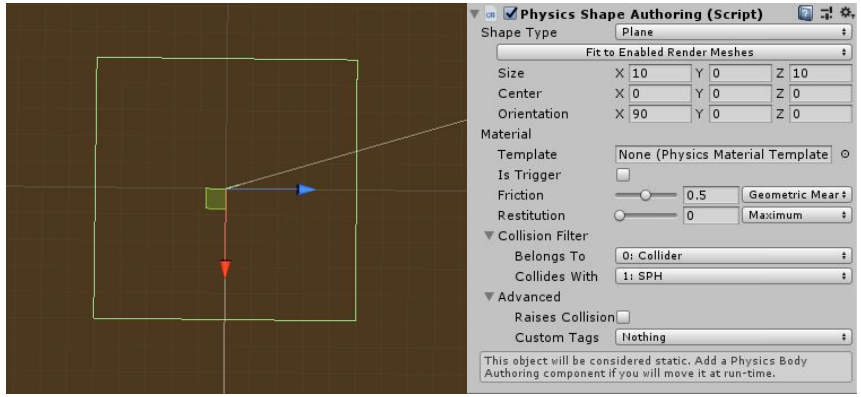
Ce choix a été fait pour se faciliter la gestion de la vélocité et du recalage des particules, ainsi qu'avoir un premier aperçu des nouveautés du moteur physics d'unity.

❖ Mise en pratique

❖ Une particule



❖ Un Collider



❖ Systems

Actuellement 3 systèmes ont été créés

- SpawnSystem : gère l'ajout de particules dans la scène
- SPHSystem : gère la physique de fluide (calcul de la viscosité, de la pression et de la gravité)
- ToySystem : permet de déplacer un collider sphérique dans la scène.

❖ Exemple du Toy System

job

```
EntityQuery m_ToyQuery;

public static GameObject toy;

[BurstCompile]
[RequireComponentTag(typeof(Toy), typeof(PhysicsCollider))]
1 référence
struct UpdateToyPositionJob : IJobForEachWithEntity<Toy,  PhysicsCollider>
{

    [ReadOnly] public float3 position;
    [ReadOnly] public float radius;

    2 références
    public unsafe void Execute(Entity entity, int index, [ReadOnly]ref Toy toy, ref PhysicsCollider physicsCollider)
    {

        if (physicsCollider.ColliderPtr->Type != ColliderType.Sphere)
        {
            return;
        }

        var scPtr = (Unity.Physics.SphereCollider*)physicsCollider.ColliderPtr;
        var sphereGeometry = scPtr->Geometry;
        sphereGeometry.Center = position;
        sphereGeometry.Radius = radius;
        scPtr->Geometry = sphereGeometry;
    }
}
```

init and update

```
UI ui;

1 référence
protected override void OnCreate()
{

    ui = GameObject.FindObjectOfType<UI>();

    m_ToyQuery = GetEntityQuery(new EntityQueryDesc
    {
        All = new[] {
            ComponentType.ReadOnly<Toy>(),
            ComponentType.ReadWrite<PhysicsCollider>(),
        }
    });
}

1 référence
protected override JobHandle OnUpdate(JobHandle inputDependencies)
{
    if (toy == null) return inputDependencies;

    var UpdateToyPositionJob = new UpdateToyPositionJob
    {
        position = toy.transform.position,
        radius = ui.radius
    };
    var UpdateToyPositionJobHandle = UpdateToyPositionJob.Schedule(m_ToyQuery, inputDependencies);

    inputDependencies = UpdateToyPositionJobHandle;

    return inputDependencies;
}
```

❖ Démonstration

❖ Et après

❖ Pouvoir simuler plusieurs fluides différents

- Adapter les formules mathématiques pour des fluides non homogènes
- Optimiser le rendu pour gagner en performances
- Ajouter des options pour pouvoir modifier en direct les fluides (radius, Time step, destruction de particules ...)

❖ Remerciements

❖ Sources

- https://fr.wikipedia.org/wiki/Fluide_(mati%C3%A8re)
- https://people.inf.ethz.ch/~sobarbar/papers/Sol14/2014_EG_SPH_STAR.pdf
- https://matthias-research.github.io/pages/publications/sca03.pdf
- https://github.com/leonardo-montes/Unity-ECS-Job-System-SPH
- https://www.youtube.com/watch?v=ILfUuBLfzGI
- https://docs.unity3d.com/Packages/com.unity.entities@0.1/manual/ecs_entities.html