Rapport animation Florence Zara

Barros Mikel-ange

I) Présentation du TP

L'objectif de ce TP est de mettre en place une représentation réaliste de la chute d'un objet rigide en fonction de différentes forces. Pour cela, nous définirons notre objet rigide comme un ensemble de points et nous calculerons le barycentre de l'objet. Une fois cela fait, nous calculerons les différentes forces et moments appliqués à chaque point ainsi que le déplacement de chaque point à l'intérieur d'une boucle d'animation.

Mots clés : boucle d'animation, Objet rigide

II) Boucles d'animation : Principe et fonctionnement

Une boucle d'animation qu'est ce que c'est ? Voilà une question que l'on retrouve souvent dans la bouche de ceux qui commencent l'animation. Comme son nom l'indique une boucle d'animation est un processus exécutant une action en boucle. Cette action peut être l'application d'une force à un objet, le déplacement d'un objet selon une courbe définie, etc. Malgré ces différences de méthodes la boucle d'animation à toujours le même but, déplacer ou déformer ,au cours du temps , un objet selon une méthode prédéfinie afin de simuler un mouvement plus ou moins naturel.

Maintenant que nous avons défini la boucle d'animation un exemple sera sans doute plus parlant. Prenons l'exemple de ce TP.

Dans le cadre de ce TP, la boucle d'animation désigne la boucle permettant de calculer les forces appliquées sur un objet rigide. Pour cela, à chaque itération de notre boucle, nous calculons tout d'abord le tenseur d'inertie de notre modèle ainsi que sa vitesse angulaire.

```
void ObjetSimuleRigidBody::CalculStateX()
{
    _InertieTenseurInv    = _Rotation * _IbodyInv * _Rotation.TransposeConst();

    _VitesseAngulaire = _InertieTenseurInv * _MomentCinetique;
}
```

Une fois cela fait, nous allons pouvoir nous servir de ce résultat pour calculer la vitesse, la dérivée de la rotation du modèle, les forces appliquées (dans notre cas la force gravitationnelle) et le couple appliqué a l'objet.

```
void ObjetSimuleRigidBody::CalculDeriveeStateX(Vector gravite)
{
    _Vitesse = _QuantiteMouvement / _Mass;
    _RotationDerivee = StarMatrix(_VitesseAngulaire) * _Rotation;
    _Force = gravite * _Mass;
    _Torque = Vector();
    for(int i = 0; i < _Nb_Sommets; i++)
    {
        Vector ri = _Rotation * _ROi[i] + _BaryCentre;
        _Torque = _Torque + cross((ri - _BaryCentre), _Force);
    }


}
```

Enfin, nous allons pouvoir recalculer la position générale de l'objet, le moment cinétique, la rotation de l'objet, la quantité de mouvement et la nouvelle position de chaque point de l'objet.

```
void ObjetSimuleRigidBody::Solve(float visco)
{
    _Position          = _Position + _delta_t * _Vitesse;
    _Rotation          = _Rotation + _delta_t*_RotationDerivee;
    _QuantiteMouvement = _QuantiteMouvement + _delta_t*_Force;
    _MomentCinetique   = _MomentCinetique + _delta_t*_Torque;

    for( int i = 0 ; i < _Nb_Sommets; i++)
        P[i] = _Position + _BaryCentre + (_Rotation * _ROi[i]);
}
```

Une fois ces nouvelles positions connues, le traitement de la boucle d'animation est terminé et on peut passer à un nouveau tour de boucle qui effectuera à nouveau les mêmes calculs.

III) Objets Rigide : Principe et fonctionnement

Un objet rigide est défini par son barycentre (centre de masse) et la distance de chaque point à ce barycentre.
Une fois ces informations obtenues, il nous reste plus qu'a calculer les forces appliquées à notre modèle comme montré dans la partie boucle de simulation.
Enfin, nous calculons les collisions avec chaque objet de la scène et notre objet rigide et nous modifions les paramètres de moments cinétiques, quantité de mouvement et positions globales de l'objet.

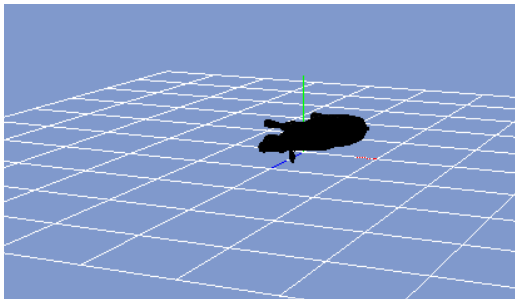
```
double height = 0.0;
bool coll = false;
double dist = 0.0;
int sommetc = -1;

for(int i = 0; i < _Nb_Sommets; i++)
{
    if (P[i].y < height)
    {
        coll = true;
        double tmp = distance2(Point(P[i].x,P[i].y,P[i].z),
                               Point(P[i].x,height,P[i].z));

        if(dist < tmp)
        {
            dist = tmp;
            sommetc = i;
        }
    }
}

if(coll)
{
    _Position.y += dist;
    _MomentCinetique = cross(P[sommetc] - _BaryCentre, -_Force) + _MomentCinetique;
    _QuantiteMouvement = -0.4 * _QuantiteMouvement;
}
```

Cette méthode , nous permet de simuler la chute et les interactions des objets rigides avec les autres objets du monde. Ainsi, nous obtenons des résultats réalistes pour une scène simple

IV) Exemples de résultats

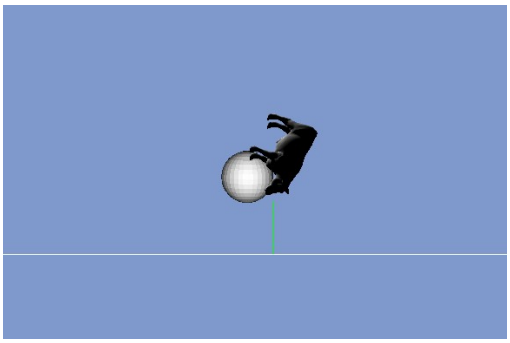
- Collisions avec le sol :

Ici, la collision avec le sol est définie quand un point passe sous le niveau du sol, pour cela on calcul pour tous les points de notre objet, sa position par rapport au sol. Et nous en modifions les paramètres en fonction du point touchant le sol.



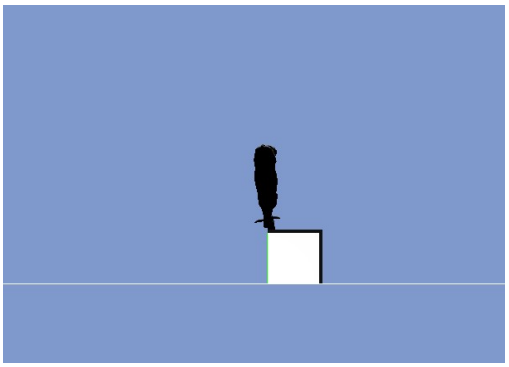
- Collisions avec une sphère :

Ici, pour détecter une collision avec une sphère on calcul la distance entre le point en collision et le centre de la sphère. Ensuite, on repousse l'objet dans la direction entre le centre de la sphère et le point de collision. Puis l'on calcul le moment cinétique par rapport à ce point la aussi.



- Collisions avec un cube :

Pour détecter la collision avec le cube on défini le cube à partir de ses points pmin et pmax et on vérifie si un point est à l'intérieur du cube ou non.
Si il est à l'intérieur, on repousse l'objet dans la direction voulue de la même manière que dans les parties précédentes.



Conclusion sur les collisions :

Les collisions avec le sol semblent bien fonctionner. Les collisions avec une sphère ou un cube semblent aussi fonctionner. Cependant, un problème se pose… si l'objet rigide rebondi au sol et qu'une toute petite partie, touche un objet en retombant alors il ne va pas tomber mais remonter sur l'objet avant de retomber.

V) Améliorations possibles

Dans un premier temps, nous pourrions lier notre modèle pour les objets rigides au modèle masse ressort fait l'année dernière cette amélioration permettrait d'obtenir un modèle plus réaliste permettant de simuler de nombreux cas, tels que recouvrir un objet d'un tissu ou encore effectuer un rebond sur un trampoline.
Il faudrait aussi améliorer les collisions avec différents objets afin d'obtenir un résultat plus réaliste et surtout éviter les bugs décris dans la partie précédente.

VI) Conclusion

Si la mise en place du système semble aux premiers abords plus simple que pour le modèle masse ressort. Il devient vite indéniable que la grande difficulté du modèle réside dans la partie collision avec son environnement. En effet, de telles collisions bien que simple en apparence peuvent poser un véritable problème dés lors que l'on voudrait se pencher dessus pour obtenir un résultat réaliste. Et ce même pour des formes simples telles que le cube ou la sphère, les résultats sont souvent incohérent avec la réalité.

Cependant, ce projet c'est montré intéressant en plusieurs points :
Il m'a permis de découvrir un nouveau modèle physique (bien que simplifié) et de voir comment l'intégrer à un environnement. Il m'a aussi permis d'améliorer mes compétences dans le domaines de la programmation 3D et dans la compréhension de formules scientifiques.