2023
Game Programming, Project #2 Physics Engine

組員: 00957116王嘉羽 00957117黃菁蕙

- 分工方式 & 呈現方式:

  我們是先將這次作業的物理引擎寫成多個class黨，之後再用各自喜歡的繪圖引擎呈現。所以作業繳交會有2個資料夾，是各自的程式碼檔案，裡面除了命名方式和繪圖引擎的地方不一樣之外，其他地方基本上相等(包括fancy idea)。這份說明文件，會先介紹物理引擎的程式碼說明，再各自介紹繪圖引擎(這份文件只有嘉羽的)。

- 實作要點 & 程式內容:

  有一個檔案專門存物理物件的相關資訊 PhyObj，存的內容包括質量、速度、角速度..等等。裡面也提供一些函式像是 applyLinearForce 和applyRotJ，當對物體施力的時候，可以去呼叫那兩個函式，他會去更新加速度。

```
class PhyObj
{
public:
    glm::mat3 I_inv;
    glm::quat rot{1, 0, 0, 0};
    glm::vec3 v{0, 0, 0};
    glm::vec3 w{0, 0, 0};
    glm::vec3 pos{0,0,0};
    glm::vec3 lin_a{0,0,0};
    glm::vec3 rot_a{0,0,0};
    float m, k;
    PhyObj(float _m, float _k);
    PhyObj(float _m);
    void applyLinearForce(const glm::vec3 &F);
    void applyRotJ(const glm::vec3 &J);
    virtual void update(float dt);
    bool isOpenDragForce = 1;
    bool isOpenGravity = 1;
};
```

```
void PhyObj::applyLinearForce(const glm::vec3 &F)
{
    lin_a += F / m;
}
```

```
void PhyObj::applyRotJ(const glm::vec3 &J)
{
    // J = I x w = r x F
    // 求 I , F
    glm::mat3 Rot = toMat3(rot);
    glm::mat3 Rot_inv = transpose(Rot);
    rot_a += Rot_inv * I_inv * Rot * J;
    std::cout << rot_a[1] << "\n";
}
```

置於，施力的方式我們的算法是採用以下方式計算，J那邊的力/100是因為，如果不除的話他轉太快了...有點線下調的感覺...

```
glm::vec3 J = cross(impactPoint, F/100.0f);

phyObj -> applyLinearForce(dot(F, impactPoint) * impactPoint / dot(impactPoint, impactPoint));
phyObj -> applyRotJ(J);
```

施力後會更新他的加速度，在main函式display的地方，我們會每秒用加速度更新他的速度

```
void PhyObj::update(float dt)
{
    //給空氣阻力 (平移 + 旋轉)
    if(isOpenDragForce){
        glm::vec3 dragforce = -v * k;
        applyLinearForce(dragforce);
        w *= 0.995;
    }
    v += lin_a * dt;
    w += rot_a * dt;

    pos += (v * dt);
    if (glm::length(w) != 0)
        rot = glm::angleAxis(glm::length(w), glm::normalize(w)) * rot;

}
```

PhyObj是存所有物理量的地方，但是由於每個人的質量分布矩陣(I)不同，所以不同形狀的物體會寫成不同class去繼承PhyObj。
像是Cube、Sphere 和 Irregular，裡面對I的定義各不相同。球和四方體是套wiki上的公式，不規則形狀則是用高斯散度定理推倒的，原理大概是先算出他的體積，且假設密度=1，去找他的質心進而推出I矩陣。

```
glm::vec3 cross_AB = glm::cross(A,B);
xyz += 1/3.0f*cross_AB*((A*A*A)/20.0f + (A*A*B)/20.0f + (A*A*a)/4.0f + A*B*B/20.0F + A*B*a/4.0f + (A*a*a)/2.0f + B*B*B/20.0f + B*B*a/4.0f +  B*a*a/2.0f + a*a*a/2.0f);

xy += 1/2.0f*cross_AB.x*((A.x*A.x*A.y)/20.0f    + (A.x*A.x*B.y)/60.0f + (A.x*A.x*a.y)/12.0f + (A.x*A.y*B.x)/30.0f + (A.x*A.y*a.x)/6.0f + (A.x*B.x*B.y)/30.0f + (A.x*B.x*a.y)/12.0f +
      (A.x*B.y*a.x)/12.0f + (A.x*a.x*a.y)/3.0f  + (A.y*B.x*B.x)/60.0f + (A.y*B.x*a.x)/12.0f + (A.y*a.x*a.x)/6.0f + (B.x*B.x*B.y)/20.0f + (B.x*B.x*a.y)/12.0f +
      (B.x*B.y*a.x)/6.0f  + (B.x*a.x*a.y)/3.0f  + (B.y*a.x*a.x)/6.0f  + (a.x*a.x*a.y)/2.0f);

zy += 1/2.0f*cross_AB.z*((A.z*A.z*A.y)/20.0f    + (A.z*A.z*B.y)/60.0f + (A.z*A.z*a.y)/12.0f + (A.z*A.y*B.z)/30.0f + (A.z*A.y*a.z)/6.0f + (A.z*B.z*B.y)/30.0f + (A.z*B.z*a.y)/12.0f +
      (A.z*B.y*a.z)/12.0f + (A.z*a.z*a.y)/3.0f  + (A.y*B.z*B.z)/60.0f + (A.y*B.z*a.z)/12.0f + (A.y*a.z*a.z)/6.0f + (B.z*B.z*B.y)/20.0f + (B.z*B.z*a.y)/12.0f +
      (B.z*B.y*a.z)/6.0f  + (B.z*a.z*a.y)/3.0f  + (B.y*a.z*a.z)/6.0f  + (a.z*a.z*a.y)/2.0f);

xz += 1/2.0f*cross_AB.x*((A.x*A.x*A.z)/20.0f    + (A.x*A.x*B.z)/60.0f + (A.x*A.x*a.z)/12.0f + (A.x*A.z*B.x)/30.0f + (A.x*A.z*a.x)/6.0f + (A.x*B.x*B.z)/30.0f + (A.x*B.x*a.z)/12.0f +
      (A.x*B.z*a.x)/12.0f + (A.x*a.x*a.z)/3.0f  + (A.z*B.x*B.x)/60.0f + (A.z*B.x*a.x)/12.0f + (A.z*a.x*a.x)/6.0f + (B.x*B.x*B.z)/20.0f + (B.x*B.x*a.z)/12.0f +
      (B.x*B.z*a.x)/6.0f  + (B.x*a.x*a.z)/3.0f  + (B.z*a.x*a.x)/6.0f  + (a.x*a.x*a.z)/2.0f);
```

上圖是將不規則形拆成很多小三角形，對他作積分然後去求I。

```
I[0][0] = xyz.y + xyz.z;
I[1][1] = xyz.x + xyz.z;
I[2][2] = xyz.x + xyz.z;

I[0][1] = I[1][0] = -xy;
I[0][2] = I[2][0] = -xz;
I[1][2] = I[2][1] = -zy;
```

再來是碰撞，在Cube和Sphere中也有覆載自己的update函式，裡面處理重力和碰到地面的反彈。

```
void Sphere::update(float dt)
{
    //給重力
    if(isOpenGravity) applyLinearForce({0, -9.8 * m, 0});
    // collision
    if (pos[1]<= (0 + 0.00001) + rad/2.0)
    {
        v[1] = -v[1] * 0.6;
        if (fabs(v[1]) < 5)
            v[1] = 0;
        pos[1] = rad/2.0;
    }
    PhyObj::update(dt);
}
```

球的碰撞比較單純，如果球心的-半徑比0還低的話，那就給一個向上的速度，讓他往上彈。
如果掉下來的速度比5還小，那就直接歸0，有點像是設一個門檻當作平衡點。

```
void Cube :: update(float dt){
    // collision
    // v.y-=9.8*dt;
    if(isOpenGravity)applyLinearForce({0, -9.8 * m, 0});
    glm::vec3 mny = {1e9, 1e9, 1e9};
    std::array<glm::vec3, 8> tmp;
    for (int i = 0; i < 8; ++i)
    {
        tmp[i] = toMat3(rot) * corner[i] + pos;
    }
    std::sort(tmp.begin(), tmp.end(), cmp);
    bool flag = false, flag2 = false;

    mny = tmp[0];
    if (mny.y < 0)
    {
        float kk = 21.5;
        glm::vec3 r = mny - pos;
        glm::vec3 cf = {0, -mny.y * kk, 0};
        applyLinearForce(glm::dot(cf, r) * r / glm::dot(r, r));
        flag = true;
        w *= 0.99;
        pos.y -= mny.y;
    }

    if (fabs(tmp[0].y - tmp[2].y) <= 0.15 && fabs(tmp[0].y) <= 0.1)
        flag2 = true;
    for (int i = 0; i < 8 && !flag2; ++i)
    {
        mny = tmp[i];
        if (mny.y <= 0)
        {
            glm::vec3 r = mny - pos;
            glm::vec3 J = glm::cross(r, {0, 0.08 * m, 0});
            applyRotJ(J);
        }
    }
    PhyObj::update(dt);
    if (flag)
    {
        v.y = fmax(v.y, 0.0);
    }
}
```

長方體就比較困難，碰撞的寫法是，先找到當前位置的最低點，並且看他是否比0還小，如果是的話那就給一個向上的力。
為了防止抖動，所以我們加入了特別的判斷，即如果有3個點都碰到地面，那就當作他停下來了，就不在作反彈的動作。

最後是子彈射擊方式介紹，我們在畫面的正中間加入了準心，準心指到的位置即子彈射擊點。我們會去判斷準心射線和物體的面有沒有交集，如果有的話會計算出那個點並進行射擊。

```
void Object::shoot(const float& F_,const glm::vec3& start, const glm::vec3& dir)
{
    //因為是正交矩陣，所以 inverse = transpose
    glm::mat4 tp = glm::translate(glm::mat4(1),phyObj->pos)*glm::toMat4(phyObj->rot)*glm::scale(glm::mat4(1),sz);

    float min_pnt = 1e9;
    const vector<glm::vec3>& tpvec = graphicObj->getVerticesByID(GraphicObjID);
    for(int i = 0; i < tpvec.size(); i += 3){
        glm::vec3 a = tp*glm::vec4{tpvec[i],1};
        glm::vec3 b = tp*glm::vec4{tpvec[i+1],1};
        glm::vec3 c = tp*glm::vec4{tpvec[i+2],1};
        glm::vec3 U = b - a;
        glm::vec3 V = c - a;
        float det_mtx = dot(V, cross(U, - dir));
        float s = dot(start - a, cross(U, -dir)) / det_mtx;
        float t = dot(V, cross(start - a, -dir)) / det_mtx;
        float r = dot(V, cross(U, start - a)) / det_mtx;

        if(s >= 0 && t >= 0 && s + t <= 1 && r >= 0){
            min_pnt = fmin(min_pnt, r);
        }
    }
    if(min_pnt != 1e9){
        glm::vec3 impactPoint = (start + dir * min_pnt)- phyObj->pos;
        //cout << impactPoint.x  << " " << impactPoint.y << " " << impactPoint.z << "\n";
        glm::vec3 F = dir * F_;
        applyForce(F,impactPoint);
    }
}
```

判斷交集的方式，是像老師上課說的，將三角形移到UV平面上，再判斷射線有沒有打入裡面。有的話就計算出那個點，並且對他施力。

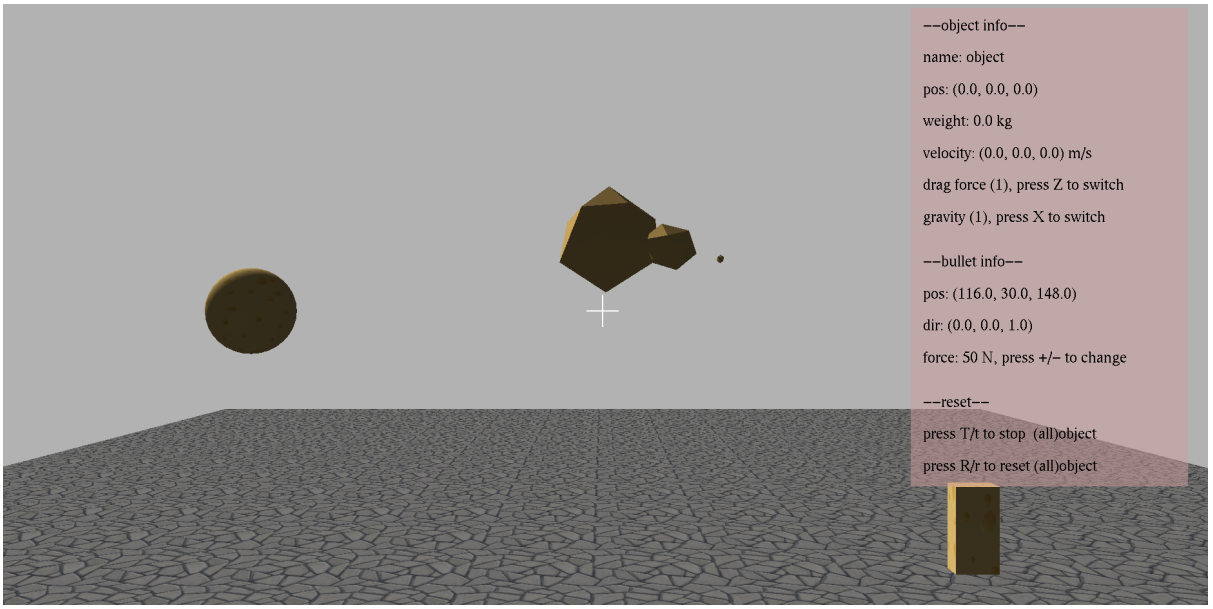
這邊的 applyForce 會呼叫 applyLinearForce 和 applyRotJ;

```
void Object::applyForce(const glm::vec3 &F,const glm::vec3 &impactPoint)
{
    glm::vec3 J = cross(impactPoint, F/100.0f);

    phyObj -> applyLinearForce(dot(F, impactPoint) * impactPoint / dot(impactPoint, impactPoint));
    phyObj -> applyRotJ(J);
}
```

物理引擎的部分大概就這樣了，接下來介紹繪圖引擎。

- 繪圖引擎介紹
  初始畫面呈現:

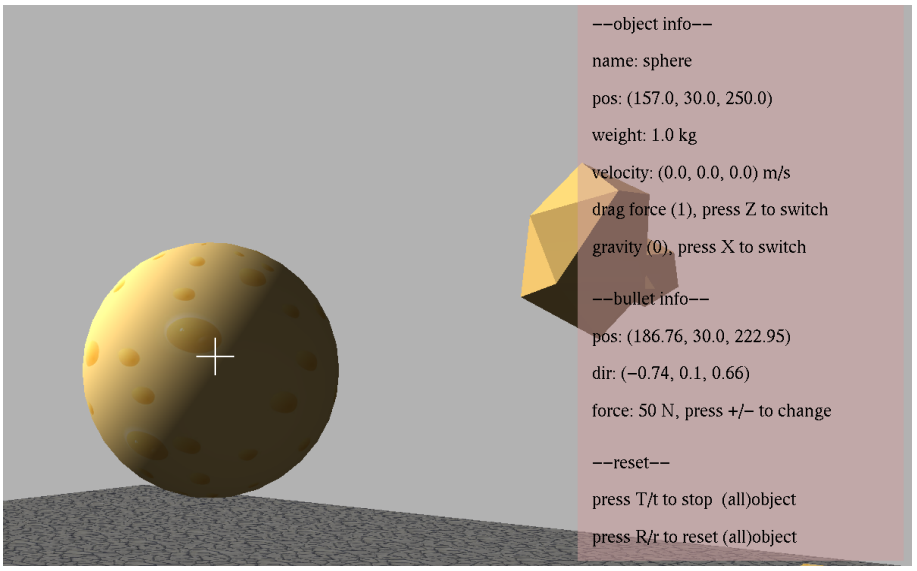


之前的專案是第三人稱視角，這次嘗試寫了第一人稱。如果有玩過minecraft應該會很熟悉，我是參考他的做法。

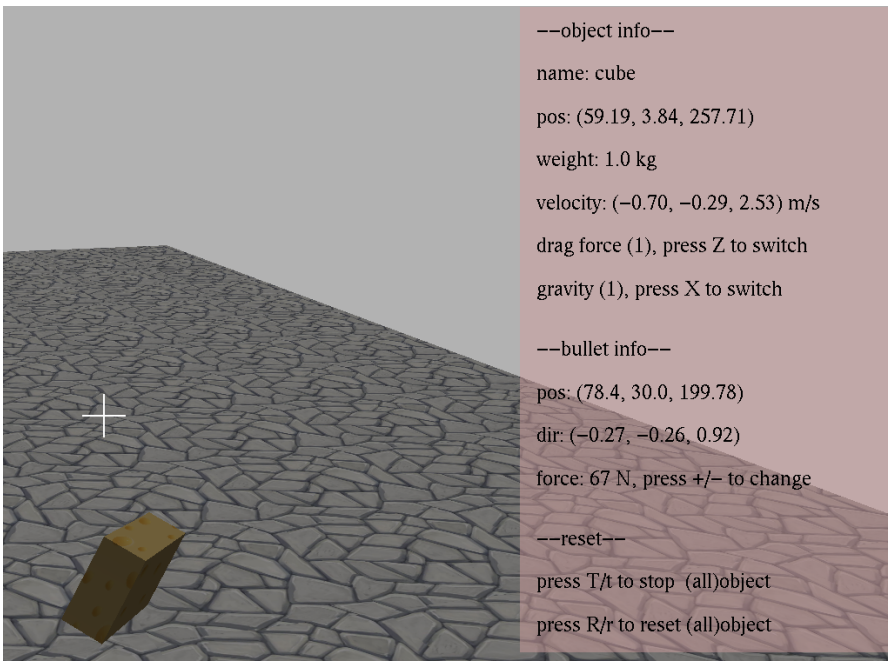
<u>案鍵說明:</u>

w/W: 人向上
s/S: 人向下
a/A: 人向左
d/D: 人向右
滑鼠向左/右/上/下移動: 往左/右/上/下方看

右邊的框框是會顯示當前被碰到的物體資訊如下圖。



object info是碰到的物體資訊，bullet info是顯示人當前的位置，子彈射向的方向和子彈力道。其中可以用z/x去控制這個物體是否要受阻力或是重力影響(不規則物體永遠不受重力影響)，也可以透過+/-去控制當前子彈的施力。



物體被打到也會即時顯示當前的資訊，由於有時候物體常常飛走(打太用力)，所以有設計stop 和 reset 按鍵，stop會讓物體停止不動(如果有重力還是會往下)，reset會將他移動到起始的位置。

心得：

沒有學過物理引擎之前，我寫類似碰撞的東西都寫得很假，像是碰到地板的時候，固定往上彈 x 像素之類的，頂多加個random讓他每次往上彈都不一樣。不過有了物理引擎之後，所有東西都看起來很真，而且也不像以前一樣需要寫很多if else，只需要套公式讓他自行計算位移和旋轉量。物理引擎真是個酷東西！！但由於我高中物理沒有學好，所以在推敲物體公式的時候很吃力，好在是兩人一組，菁蕙物理非常好，非常罩！！而且我們還有一位祕密組員 - 冠宏，他一步一步帶我們計算物理公式，多虧了冠宏才有今天的我們！非常謝謝冠宏！