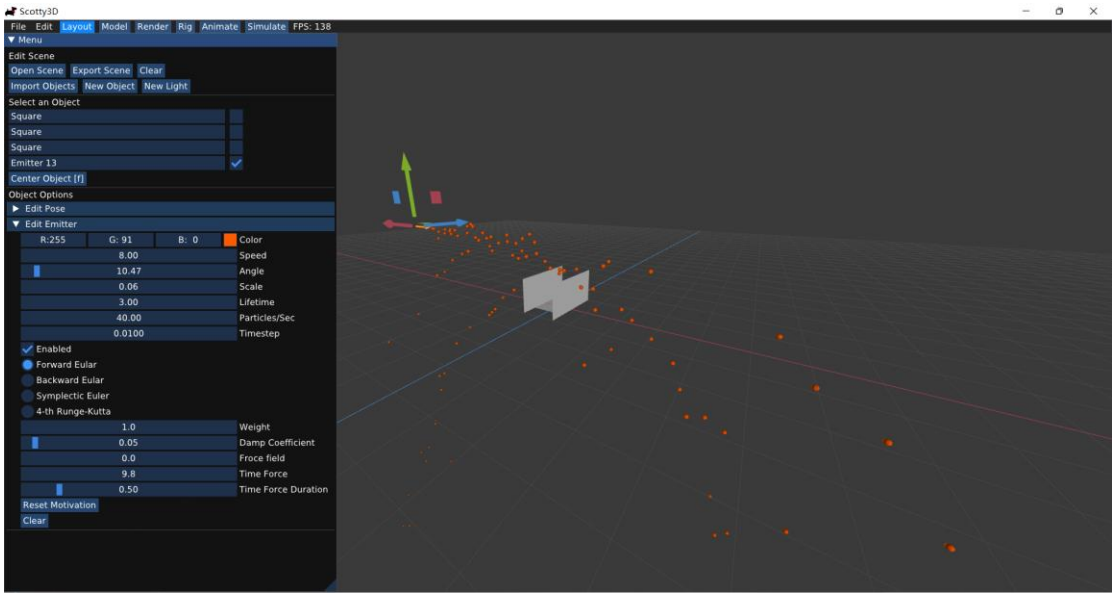
计算机图形学大作业------小球模拟方向

计算机试验班 001 王嘉禾 2193211079

1.实验环境

scotty3D

2.最终界面展示



3.补全的函数逻辑

(1) particles.cpp --- Scene_Particles::Particle::update

输入：

const PT::Object& scene 场景对象
float lifetime 小球生命周期（超出生命周期直接删除不再做计算）
float dt 此次进入函数需要计算的时间周期
float radius 小球半径（用于实现精准碰撞）
int method 时间积分的实现方法（0,1,2,3 分别对应四种方法）
float weight 小球质量（用于模拟更符合物理现实的碰撞）
float dampexp 碰撞体的阻尼系数（用于模拟更符合物理现实的碰撞）
float forceexp 力场 1（在固定空间中存在）的大小
float timeforceexp 力场 2（一定时间后消失）的大小
float timeforcedur 力场 2 持续时间

实现逻辑简述：

最小时间单位为 eps, 剩余未计算的时间为 tLeft, 因此仅当 tLeft>eps 时执行，否则直接删除。执行时，首先**调用 scene.hit()判断是否发生碰撞**。

若发生碰撞，则 hit()函数返回碰撞的位置 position，该位置的法向量 normal 和碰撞发生的距离 distance(可计算出时间)，由此计算碰撞后前进的方向和前进时间，**直接使用前向欧拉法**可计算出最终位置（由于在所有的时间单位 eps 中，发生碰撞的次数很少，因此**几次碰撞中的计算误差在所有的计算中几乎可以忽略**，可以使用

简单方法）

若不发生碰撞，则根据 method 参数，调用四种方法之一实现时间积分，其中小球的加速度由另一个函数 acc 求得，并且与 update()函数的输入中的各种物理参数有关。

(2) tri_mesh.cpp ------ Triangle::hit

输入：

const Ray& ray 其中 ray 中有三个参数

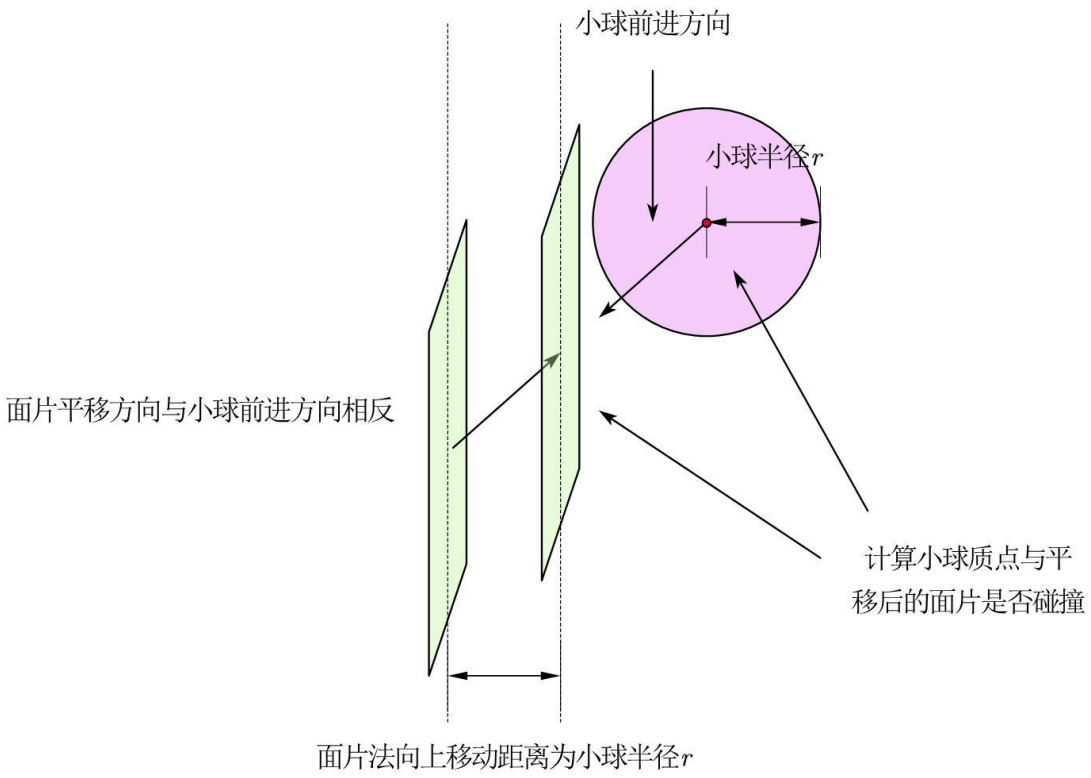
ray.point 射线的出发点

ray.dir 射线的方向

ray.r 小球的半径

实现逻辑简述：

由于小球存在半径，不能只计算质点（小球中心）与面片的碰撞情况，而应计算小球边缘的碰撞情况。方法是计算式将面片迎着小球前进方向的反方向移动，并且使法向上移动的距离等于小球的半径，具体情况如下图所示。



找到三角形面片**面向射线的法向量 tnorm**，并与小球前进方向 ray.dir 点乘得到点积 dot，可算出平移的距离为-ray.dir / dot * r，此时就可以将小球视为质点调用 Möller-Trumbore 算法计算碰撞。

该算法的本质是列出碰撞发生时的**连立方程组**，判断**碰撞点是否在射线上**（可能求出在射线的反方向）以及**碰撞点是否在三角形面片内**，当满足时即可返回碰撞点和该点距离射线起点的距离。若不成立，则返回 false 说明未发生碰撞。

4.实现算法

(1) 四种时间积分

前向欧拉法

$$x_{t+\Delta t} = x_t + \Delta t * v_t$$
$$v_{t+\Delta t} = v_t + \Delta t * a_{t,x_t}$$

代码实现

```
if(method == 0) {
    // forward eular
    acc = getacc(t, timeforcedur, pos, velocity, accbypos, accbytime);
    pos += eps * velocity;
    velocity += eps * acc;
```

后退欧拉法（前向欧拉法改变运算顺序即可）

$$v_{t+\Delta t} = v_t + \Delta t * a_{t,x_t}$$
$$x_{t+\Delta t} = x_t + \Delta t * v_{t+\Delta t}$$

代码实现

```
} else if(method == 1) {
    // backward eular
    acc = getacc(t, timeforcedur, pos, velocity, accbypos, accbytime);
    velocity += eps * acc;
    pos += eps * velocity;
```

半隐式欧拉法（在前向欧拉法中，将加速度的空间自变量改为变化后的空间坐标）

$$x_{t+\Delta t} = x_t + \Delta t * v_t$$
$$v_{t+\Delta t} = v_t + \Delta t * a_{t,x_{t+\Delta t}}$$

代码实现

```
} else if(method == 2) {
    // semi symplectic eular
    pos += eps * velocity;
    acc = getacc(t, timeforcedur, pos, velocity, accbypos, accbytime);
    velocity += eps * acc;
```

4 级 4 阶龙格库塔法

$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$
$$k_1 = hf(x_i, y_i)$$
$$k_2 = hf\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1\right)$$
$$k_3 = hf\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_2\right)$$
$$k_4 = hf(x_i + h, y_i + k_3)$$

写成坐标与速度的关系为

$$x_{t+\Delta t} = x_t + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$
$$k_1 = \Delta t * v(t, x_t)$$

$$k_2 = \Delta t * v\left(t + \frac{1}{2}\Delta t, x_t + \frac{1}{2}k_1\right)$$

$$k_3 = \Delta t * v\left(t + \frac{1}{2}\Delta t, x_t + \frac{1}{2}k_2\right)$$

$$k_4 = \Delta t * v(t + \Delta t, x_t + k_3)$$

为了将速度 v 写成坐标 x 和时间 t 的函数，对速度 v 采用后退欧拉法

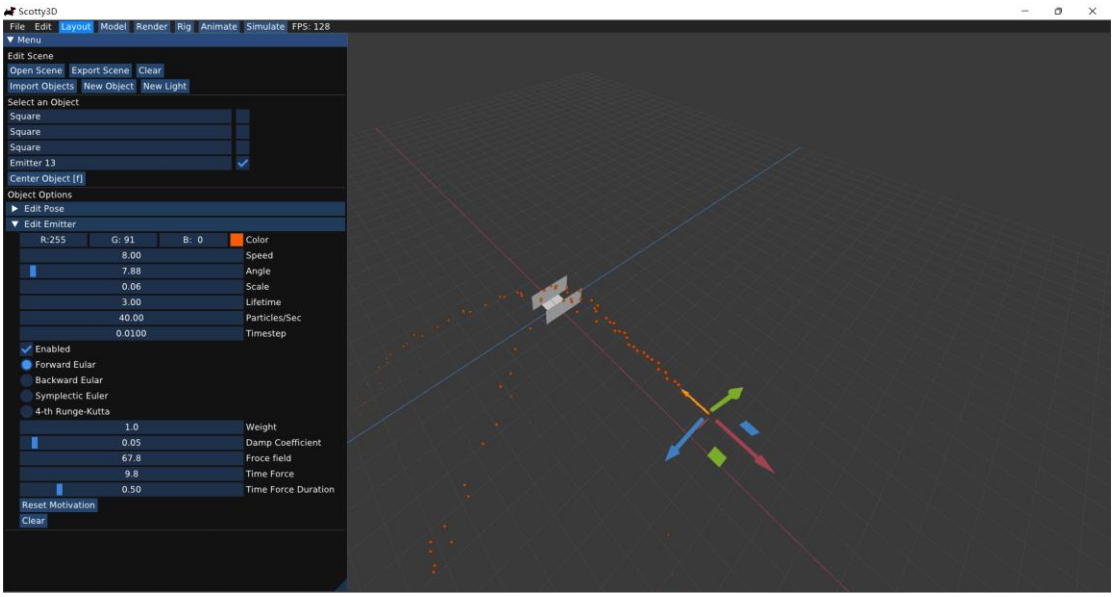
$$v(t + \Delta t, x_t + \Delta x) = v(t, x_t) + \Delta t * a(t + \Delta t, x_t + \Delta x)$$

而加速度 a 与空间坐标 x 和时间 t 有关，对于速度的迭代，也使用类似的加权平均的方法代码实现如下：

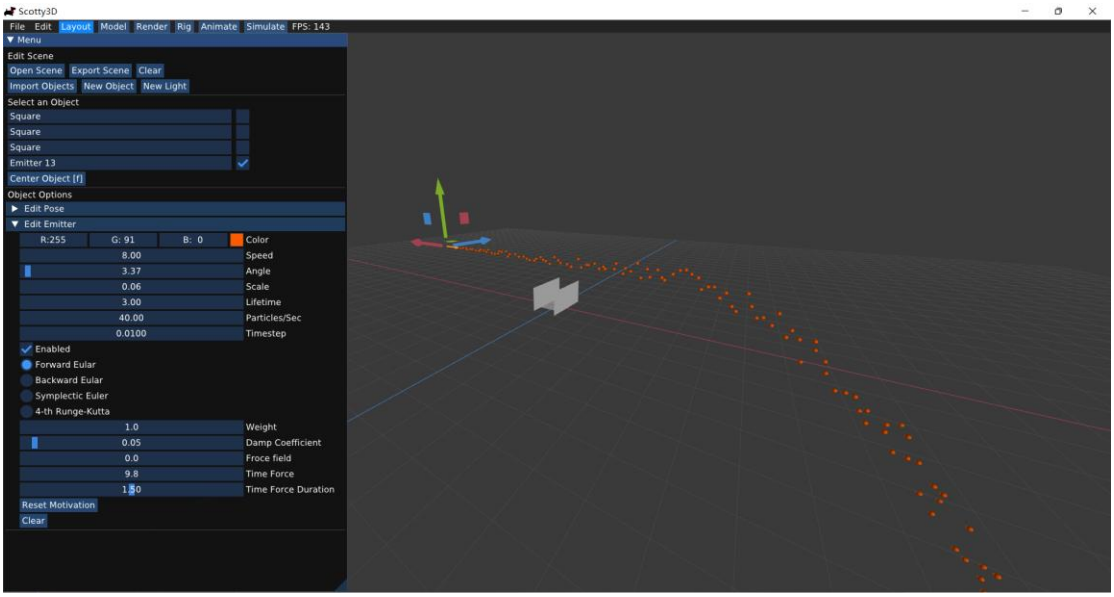
```
} else if(method == 3) {
    // Runge-Kutta
    Vec3 velocity_old = velocity;
    Vec3 vel0, vel1, vel2, vel3;
    acc = getacc(t, timeforcedur, pos, accbypos, accbytime);
    k1 = eps * velocity;
    vel0 = velocity + eps * acc;

    acc = getacc(t + eps / 2.0f, timeforcedur, pos + k1 / 2.0f, accbypos, accbytime);
    vel1 = velocity + eps * acc / 2.0f;
    k2 = eps * vel1;
    vel1 = velocity + eps * acc;

    acc = getacc(t + eps / 2.0f, timeforcedur, pos + k2 / 2.0f, accbypos, accbytime);
    vel2 = velocity + eps * acc / 2.0f;
    k3 = eps * vel2;
    vel2 = velocity + eps * acc;

    acc = getacc(t + eps, timeforcedur, pos + k3, accbypos, accbytime);
    vel3= velocity + eps * acc;
    k4 = eps * vel3;

    pos += (k1 + 2.0f * k2 + 2.0f * k3 + k4) / 6.0f;
    velocity = (vel0 + 2.0f * vel1 + 2.0f * vel2 + vel3) / 6.0f;
}
```

(2) 外力引入

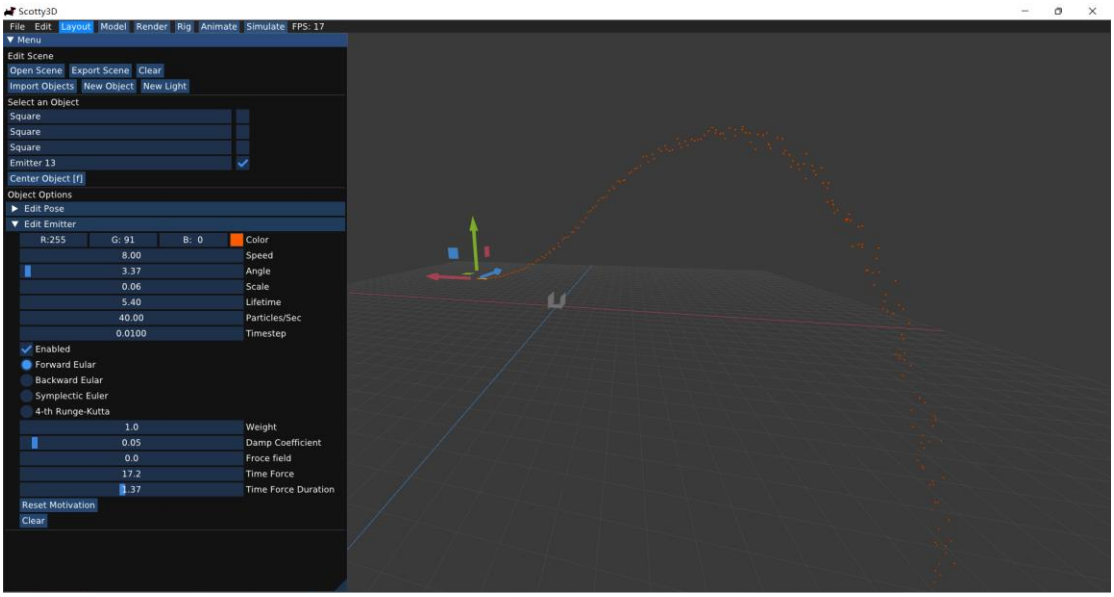
外力 1：存在于**两个竖直挡板之间**，方向为平行于挡板且竖直分量为 0，写成向量形式为**（0,0,1）**，大小可调节，在（-100,100）之间，负号代表方向为反向。当小球进入两挡板之间时，会受到水平方向上的力。**获得的水平加速度大小与力的大小和小球的质量有关**，效果如图所示。



外力 2：与时间相关，从小球发射开始，**到一定时间后结束**，该时间可调节。力的方向为竖直方向，写成向量形式为**(0,1,0)**，大小可调节，范围为（-50,50），负号代表方向相反，即力的方向向下。显然，当力的大小为 9.8 且小球质量为 1 时，该外力刚好平衡重力，使得小球在一段时间内作“悬浮前进”，效果如下图所示。



也可以使合力数值向上，使得小球作重力反转后的抛物运动，效果如下图所示。



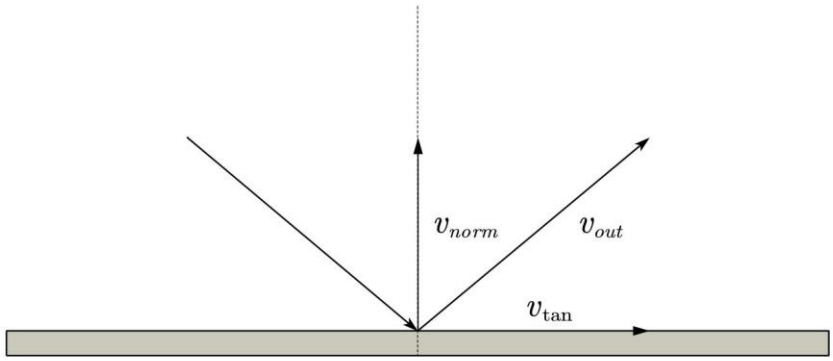
由于外力仅影响加速度，因此加速度与时间和位置有关，可写成 a=g(t,x)，因此有

$$\frac{dx}{dt} = v \qquad \frac{dv}{dt} = g(t,x)$$

则**引入的两个外力只产生加速度的叠加**，即 a=g+acc1(t)+acc2(x)，acc1 和 acc2 分别为两个外力产生的加速度，通过获取加速度的函数 getacc(t,x)实现，只需判断**是否处于力场空间中以及小球存活时间是否在一定范围内**，见 particles.cpp ------ getacc()

(3) 阻尼和质量的实现

由于碰撞瞬间的摩擦力难以计算，因此采用阻尼系数(dampexp)进行模拟，使小球产生**切向速度的衰减**。对于碰撞产生的能量损失，**将碰撞体视为一个质量较大的物体，使小球与其作完全弹性碰撞**，以此模拟现实中墙壁有一定形变能力的情况：当小球质量很小时，墙壁等效视为十分坚硬，小球碰撞不产生能量损失，而当小球质量较大时，碰撞瞬间墙壁发生一定形变，吸收部分能量。代码实现中小球质量范围为[0.1,100]将碰撞体视为质量为 200 的物体，具体计算过程如下所示。

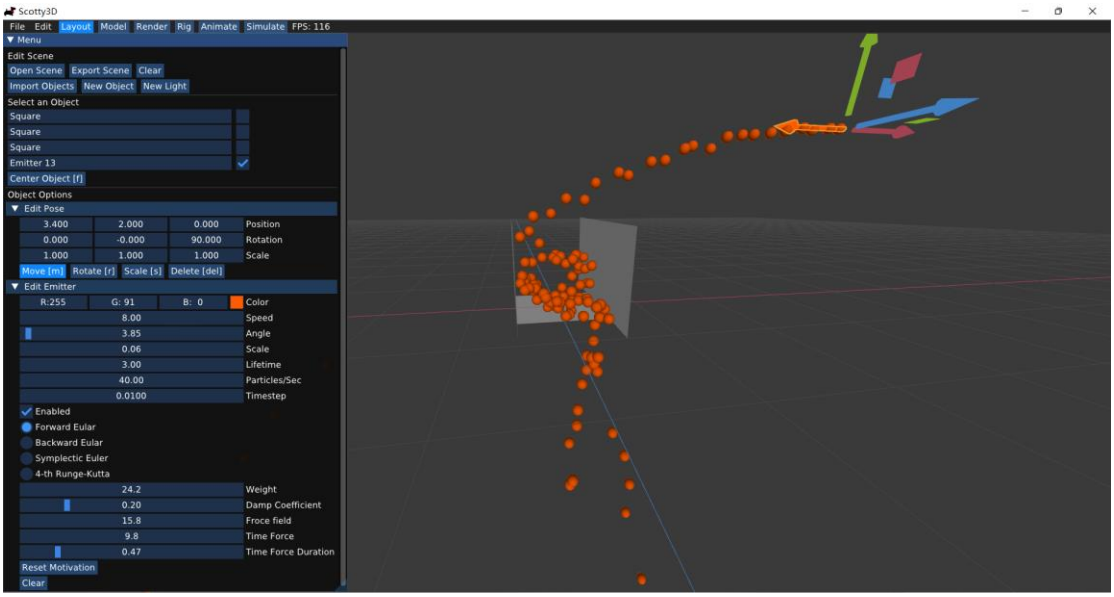


设$v_{\tan}$和v_{norm}为仅考虑反射时出射的切向和法向速度

考虑阻尼系数dex: $v_{\tan} = (1 - dex) * v_{\tan}$

考虑能量损失: $v_{norm} = \frac{200 - w}{200 + w} * v_{norm}$ 小球质量$w \in [0.1, 100]$

实现效果如下图所示，当小球质量和墙壁阻尼系数均较大时，明显看到小球的反弹能力很弱，受外力影响也很小。

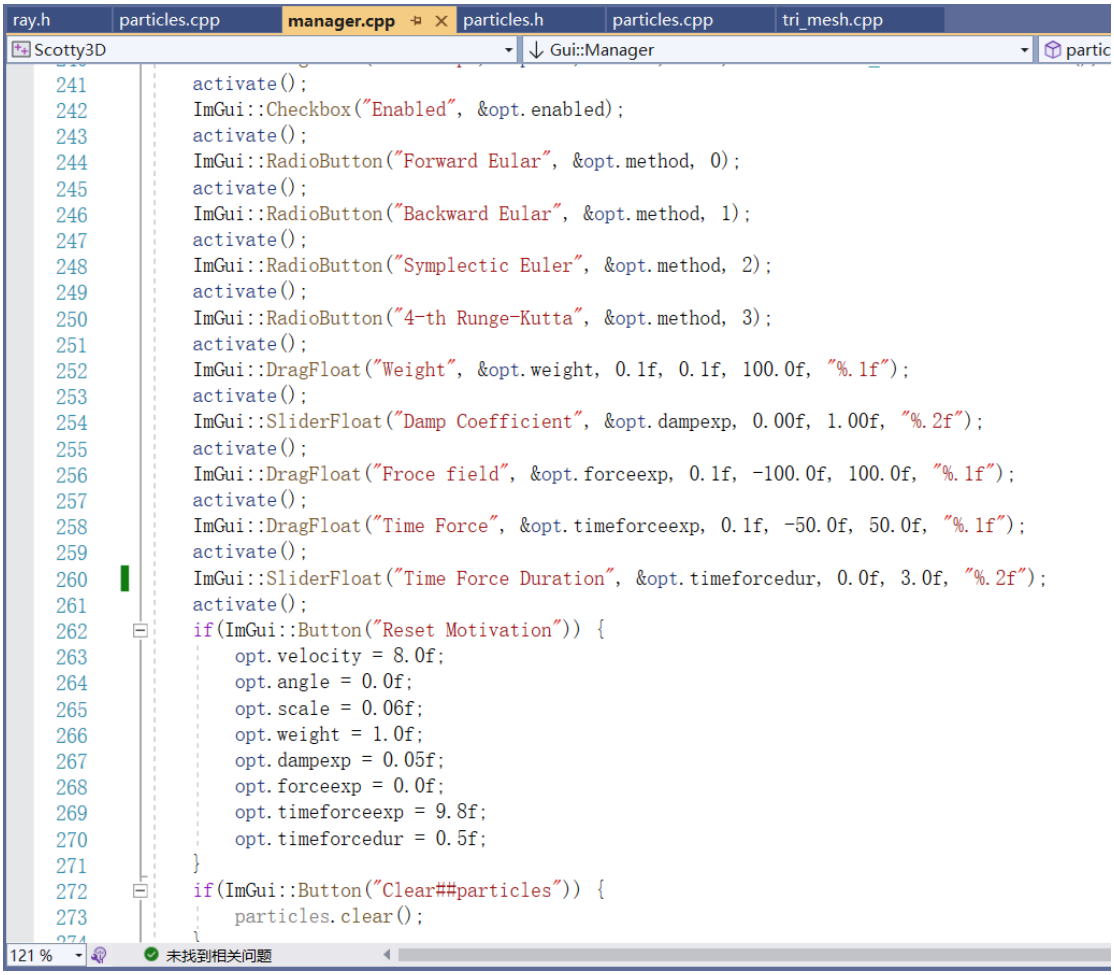


5.可视化交互界面的实现

（1） 在 **src/scene/particles.h** 中，找到 Options 结构体，加入所需的参数 method（方法编号），weight（小球质量），dampexp（阻尼系数），forceexp（外力 1 大小），timeforceexp（外力 2 大小），timeforcedur（外力 2 持续时间）

```
struct Options {
    char name[MAX_NAME_LEN] = {};
    Spectrum color = Spectrum(1.0f);
    float velocity = 8.0f;
    float angle = 0.0f;
    float scale = 0.06f;
    float lifetime = 15.0f;
    float pps = 5.0f;
    float dt = 0.01f;
    int method = 0;
    float weight = 1.0f;
    float dampexp = 0.05f;
    float forceexp = 0.0f;
    float timeforceexp = 9.8f;
    float timeforcedur = 0.0f;
    bool enabled = false;
};
```

（2） 在 **src/gui/manager.cpp** 中，找到 **Manager::particles_edit_gui(Undo& undo, Scene_Particles& particles)**函数，在函数中使用 gui 编辑可视化界面，并与 opt 中的参数链接，使得 gui 界面可以修改 opt 中的参数，如图所示。



(3) 修改 **Particle::update** 函数，传入 opt 中新增的参数，用于实现新的模拟过程，在 **src/scene/particles.cpp** 中调用，在 **src/student/particles.cpp** 中实现，均作修改

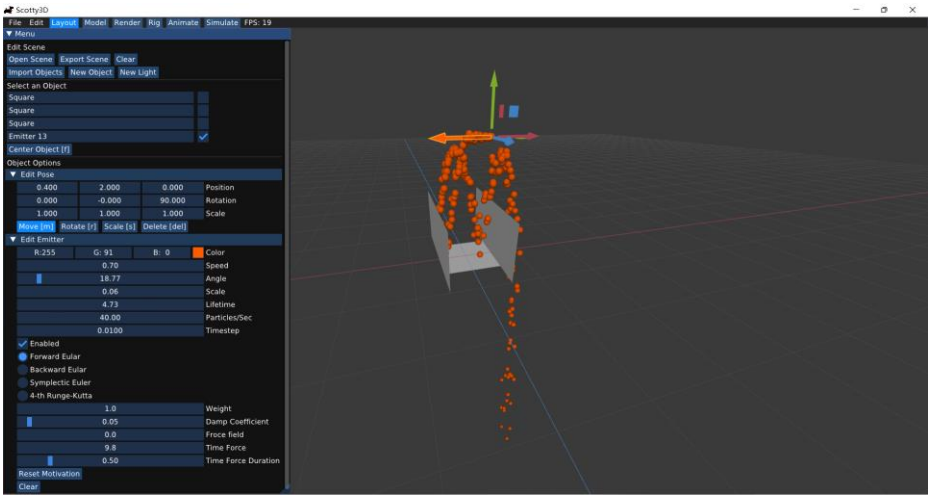
```
for(Particle& p : particles) {
    if(p.update(scene, opt.lifetime, dt, radius * opt.scale, opt.method, opt.weight,
                opt.dampexp, opt.forceexp, opt.timeforceexp, opt.timeforcedur)) {
        next.emplace_back(p);
    }
}
```

```
bool Scene_Particles::Particle::update(const PT::Object& scene, float lifetime, float dt,
                                       float radius, int method, float weight, float dampexp,
                                       float forceexp, float timeforceexp, float timeforcedur) {
```

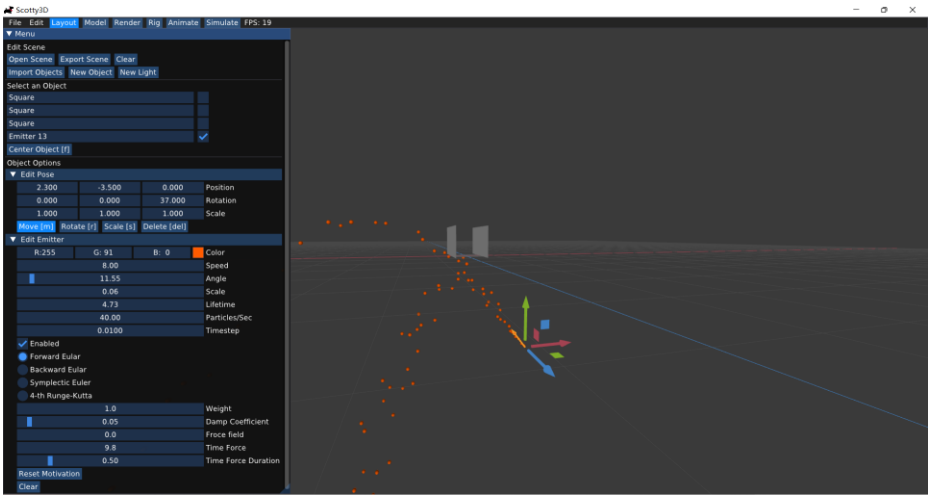
6.实验结果

实验实现成功，在之前的报告中已经列出了部分实验成果截图，此处增加几种情况。

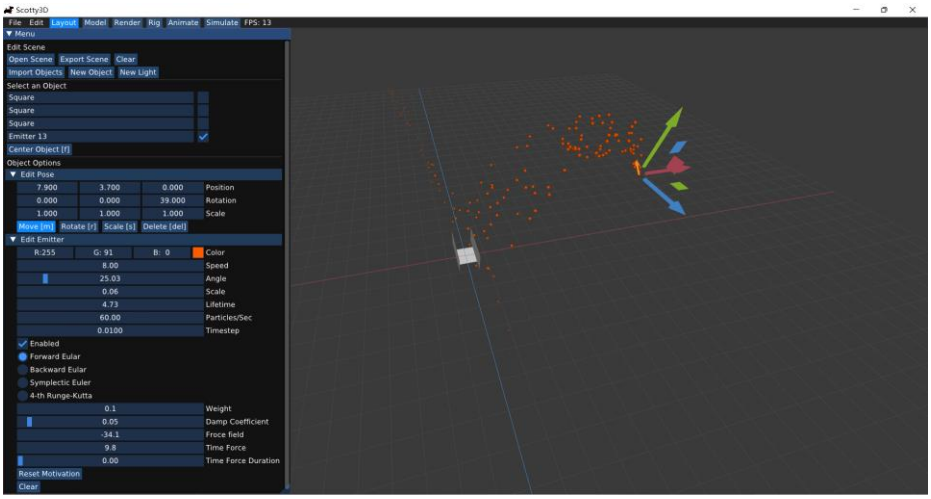
（1） 小球初速度很低，几乎作自由落体



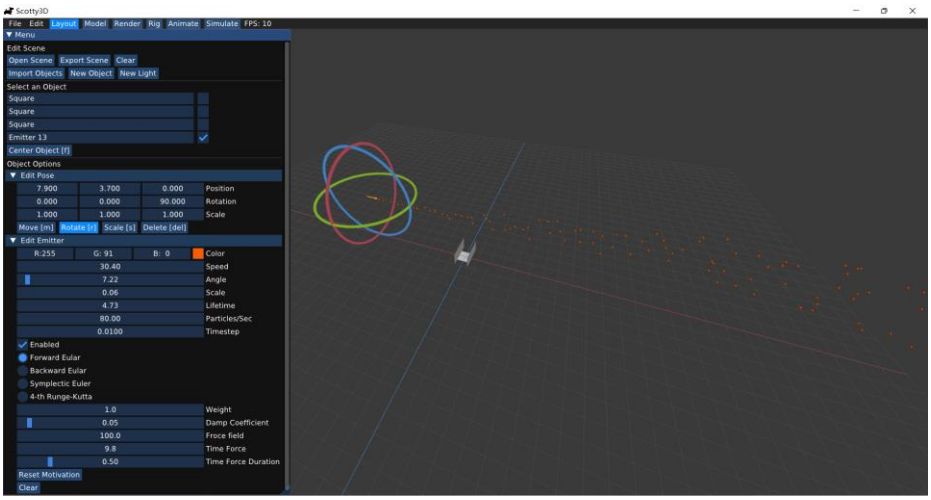
（2） 小球从底部向上发射



（3） 小球质量很小而力场较大，小球几乎刚进入力场就被吹飞



(4) 小球速度极大，即使力场很大也可以直接冲破力场



7.实验总结

(1) 实验中遇到的困难

该实验整体困难较小，困难主要集中在**阶段一的碰撞检测部分以及阶段二的 ui 界面交互**部分。

碰撞检测部分的困难主要在算法的理解和对于 hit()函数输入输出内容的梳理，我的方法是**运行时将一些输入输出或中间量打印出来验证**，算法部分理解物理意义之后其实不是很难。

由于没有接触过 gui 和不知道工程运行的逻辑，使得开始时不知道怎么编写 ui 界面，解决方案是**根据 ui 界面中已有 speed，angle，scale 等参数，和“edit pose” “edit emitter”等标题，在工程文件中进行搜索**，定位到了 particles.cpp 中的 options 结构体以及 manager.cpp 中的 particles_edit_gui 函数，再**根据函数的调用关系梳理 gui 的运作过程**，就变得清晰明了。

阶段一的前两个实验，渲染实验和几何实验，渲染实验主要的困难在于对于**环境的使用不熟练，一些语法经常使用错误**，最终是询问同学解决的。几何实验的困难主要在于逻辑的梳理，在代码实现中出现了不少步骤错误和对象错误（**缺少了一些操作或对错误的边进行的操作**），最终将每个过程一一梳理并对比代码，得以解决。

(2) 对课程的建议

实验课的操作内容在我看来质量较高，三个实验的完成过程其实都不是特别顺利，都有探索价值，唯一的建议是**希望能有一个对于实验环境以及工程环境的细节说明书**。例如在第一个实验中，对 fshader.glsl 的编辑，里面用到了 Material 结构体中的参数和 Light 结构体中的参数，但这些参数的具体意义其实并不清楚，最终是通过询问同学解决的。又例如实验三中，Vec3 这个类型是自定义的类，我最终找到了定义类的代码查看了里面的函数和运算方式，在这之前其实并不清楚 Vec3 这个类型该怎么使用，甚至以为是 C++自带的库，但又没有 include。这些问题其实在群里也都有助教解决，但是如果有一个完整的说明书，或许可以在减少助教工作量的同时减少同学们做实验的障碍。

8.参考资料

除助教给出的文档外，参考的资料有

https://en.wikipedia.org/wiki/Runge-Kutta_methods
https://en.wikipedia.org/wiki/Semi-implicit_Euler_method
《数值分析》--- 李乃成 梅立泉 科学出版社，2011