游戏项目设计分报告

作者	学号
陶新野	3160103871

贡献总览

我个人在本次游戏项目中的贡献主要体现在整体架构中的物理解算和数据池端，具体来说，主要实现了：

- 物体抽象类及它的众多派生类
- 物体内存缓存池
- 游戏特效渲染设计
- 其他基础工具编写

作为第一轮迭代的两个程序员之一，还有很多工作与同伴一起完成，并共同搭建了最终的代码框架，进行了小幅度的代码重构。

分模块设计报告

物体抽象类：Object

物体抽象主要提供了共用基类接口设计，作为游戏全局所有可绘制实体的父类接口，它必须具有绘制和绘制更新的方法声明，同时，为了实现物体受控运动的游戏逻辑，还需要提供一系列运动参数的set方法，另外，为了在全局操作中获得物体位置，并提供每个物体和被其他物体附着的属性，Object类还需提供详尽的get方法。

具体接口设计如下：

```
// 绘制及更新方法
virtual void Update(float);
virtual void Draw(const Camera&, const LightCollection&);
// Get accessors
virtual btVector3 GetOrigin(void);
virtual btTransform GetTransform(void);
virtual btVector3 GetVelocity(void);
// Set accessors
virtual void SetVelocity(const btVector3&);
// 其他共用接口
virtual void Attach(std::weak_ptr<Object>);
```

在接下来的具体派生物体实现中，主要的代码关键就在于

1. 实现从抽象的物体形状计算出用于OpenGL渲染的三角形细分曲面坐标以及法线值。
2. 将物体整体运动信息与实际参与解算的物体类进行绑定，实现较高层次的逻辑操控。

特殊物体介绍：Cloth

为了充分利用Bullet物理引擎强大的软体支持，我们封装了织物物体类Cloth。

织物体与普通的刚体对象不同，很难实现精准的整体操控，在实际设计中，我使用附着点方式进行抽象。将织物整体的运动着力点限制在有限的几个片元顶点上，在这之后对这些着力点进行进一步运动力学操作，甚至是附着在其他可控制刚体上。

对于织物来说，更为复杂的构建过程在于三角细分曲面的计算，在实际编码中，我们借用了Bullet物理引擎中实现的凸包计算机：`btConvexHullComputer` 来进行构建。

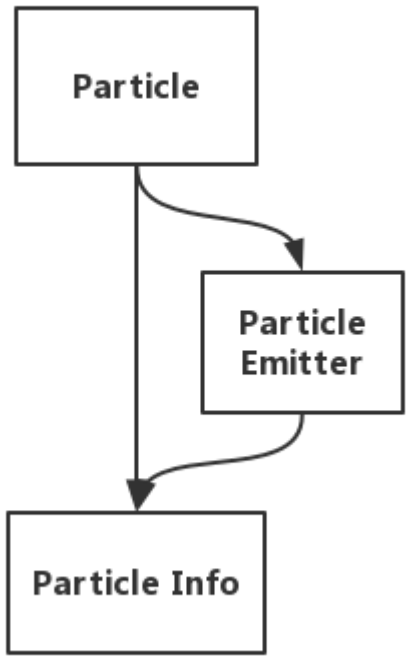
构建伪码如下：

```
func NewCloth(){
    Create anchors // at rigid body or at absolute control point
    Create Cloth
    Attach Cloth to anchors
    Create ConvexHullComputer
    use computer to solve triangle meshes
}
```

特殊物体介绍：Particle

如前所述，在游戏中使用了大量粒子对象，而对这些粒子对象的管理也是通过Object基类来进行派生实现的。实际上，我使用粒子发射点作为粒子物体的靶点，更便于管理大量同质粒子对象。

粒子作为一个单一整体，需要存放一个序列的实际粒子信息，同时还需要包含随时间而发射粒子的粒子生成器来实际产生这些粒子信息。这一模块的整体结构关系如图：



在粒子系统初始化的过程中，首先将粒子发射器按照指定要求初始化，再对固定大小的序列进行生成填充。

在粒子系统更新的过程中，首先这一更新单个粒子信息的最新状态，再按照设定的时间间隔生成新的粒子。

以上两个过程的伪码说明如下：

```
func New():
    particleEmitter = new ParticleEmitter(config)
    for slot in particles:
        slot = particleEmitter.newParticle()

func Update(delta):
    for slot in particles:
        slot = slot + delta
    _delta += delta
    if _delta > duration:
        slot = particleEmitter.newParticle()
```

特殊物体介绍：Hero

以上众多特化物体在游戏中只起到从属道具的作用，而对于游戏的核心：玩家和NPC，我们使用人物贴图来创造更有内涵的图像形象。

在具体实现中，由于对骨骼动画掌握欠缺，使用了简单的立方体来作为人物的基础模型，考虑到这与游戏整体复古怀旧风格不矛盾，坚持了这一设计。

使用简单的立方体作为任务基础，简化了对人物动作细节的刻画。但同时，为了使得立方体人物具有我们期望的动作形态，同时能与其他物体实现交互，必须将其加入整体的物理世界。为了解决这一问题，我使用了六向角约束对作为任务的扁立方体进行控制。既使得其正常与其他道具发生碰撞，也不会产生不正常的旋转和平移情况，人物动作完全由控制器操控。

物体缓存池：TempCollection

物体缓存池，由字面意思来看就是用于存放临时物体的内存管理器。在游戏设计中，由于大量临时对象如射击特效、临时人物特效等需要快速创建和销毁，这种内存管理设施很有必要。

具体来说，临时缓存池的入池函数传入基类物体实例，并附带保留物体的时间长度，在游戏主循环中对缓存池进行刷新，检查超过保留时间的物体对象并执行对象内存删除。

循环更新方法的伪码如下：

```
function Update(delta_time):
    for object in objects:
        object.elapsed_time += delta_time
        if object.elapsed_time > duration:
            delete object
```

作为高度动态性的内存管理器，最重要的是保证物体销毁时同时销毁所有外端引用，为了实现这种机制，我使用了C++11提供的智能指针，采用shared_ptr作为数据索引指针，weak_ptr作为外端引用指针。

对于迭代器返回，我们提供接口`objectBegin(void)`，其返回一个自定义结构体的原生迭代器，同时要求此结构体继承于全局的可迭代基类，此基类规定了所有返回的迭代类型必须具备接口`get()`，而get方法则是由此结构返回weak_ptr供外层安全引用。

具体使用了继承于全局共同基类的TempWrapper类迭代器：

```
struct TempWrapper: public ObjectWrapper{
    friend TempCollection;
    // other util data
    float duration_;
    Timer::TimingId timer_;
    // data holder
    std::shared_ptr<Object> data_ref_;
 public:
    TempWrapper(Object* naked, float duration);
    TempWrapper(std::shared_ptr<Object> shared, float duration);
    ~TempWrapper()
    std::weak_ptr<Object> get(void);
};
```

将这一类型的迭代器传出后，使用get方法得到供外端使用的weak_ptr指针引用。一旦在内存池中销毁对象，对应的弱引用也随之失效。

物体控制器：Character

在完备的物体抽象设计基础上，对面向对象的物体封装可操控的命令句柄就显得更为简单，具体来说，我们的游戏中需要提供移动、跳跃、攻击几个基本动作操作命令。接口设计如下：

```
void Move(bool, float);
void Rotate(bool, float);
void Jump(float);
void LaserAttack(void);
void BoxAttack(void);
```

其中布尔量标识移动方向，初步实现了两种攻击方式。

每一种可量化的移动方式都对应了输入参数的时间值，在程序中，使用每一帧刷新间的绝对时间差作为这一参数的输入，使得人物的运动状态不随计算效率和帧率的变化发生大的偏移。同时，为了避免键盘时间短时间大量触发导致人物运动不自然，我们设置了运动速度上限，这一上限与人物的内部数值配置有关，同时，每一帧停止运动响应后对响应人物进行reset操作，清除冗余的速度信息，使得人物运动符合玩家预期的同时也遵循物理引擎的计算规则。

Character是一个拥有Move，Rotate或是Jump等操作方式的角色（这样的角色可以是人类角色、可以是怪物等等）。为了使得这些行为有承担方也有发出方，我们进一步设计了Controller，其作用就是抽象出它们的运动逻辑并操控这一类Character。

特效渲染

除了基本的光照和纹理渲染外，为了实现我们构想的游戏艺术化画面，还需要特殊的特效渲染方法，在实际开发过程中，我主要贡献了两个风格化特效渲染，下面分别介绍。

粒子特效渲染

粒子是图形学世界中常常用于模拟更复杂结构体的体素型物体。在我们的游戏中，直接使用点精灵方式渲染裸粒子，既简化了美术调试过程，也契合了本款游戏复古迷幻的风格路线。

具体来说，我们对场景中的三种成分使用了粒子渲染的方式，分别是火焰道具，攻击特效，以及生命值变化特效。它们分别具有不同的特性：火焰粒子无光晕渲染，攻击特效具有较强光晕渲染，生命值表示具有特别的粒子形状（加号和减号）。

具体实现中，在片元着色器内使用前一遍提供的PointCoord信息可以简单的通过对应形状的函数曲线进行着色，并使用Alpha通道进行光晕渲染。值得注意的是，在此处使用的Alpha叠加渲染策略，对与粒子渲染的顺序有较高的要求，只有在由后至前依序渲染的情况下才能得到最佳的实际效果。

以下为相应的着色器代码：

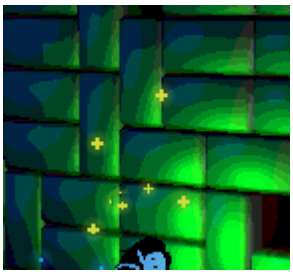
普通光晕方块粒子：

```
vec2 radiant = gl_PointCoord - vec2(0.5, 0.5);
float radiusSquare = dot(radiant, radiant);
if(abs(radiant.x) < 0.07 && abs(radiant.y) < 0.07){
    FragColor = vec4(vColor, 1.0);
}
else if(radiusSquare < 0.25){ // from 0.0625 to 0.25
    FragColor = vec4(vColor, 0.25 - radiusSquare);
}
else{
    discard;
}
```

加号型生命值表示粒子

```
vec2 radiant = gl_PointCoord - vec2(0.5, 0.5);
float radiusSquare = dot(radiant, radiant);
if(abs(radiant.x) < 0.07 && abs(radiant.y) < 0.2 || abs(radiant.y) <
0.07 && abs(radiant.x) < 0.2){
    FragColor = vec4(vColor, 1.0);
}
else if(radiusSquare < 0.25){ // from 0.0625 to 0.25
    FragColor = vec4(vColor, 0.25 - radiusSquare);
}
else{
    discard;
}
```

将如上的渲染逻辑应用进游戏粒子系统中可以得到如下图的实际效果：



卡通化特效渲染

卡通化作为非真实感渲染的一个分类，出现过更多有意思的技巧。这些渲染着色的方法以较低的运算成本得到了非常出色的风格化效果。在本游戏项目中，游戏风格最初定为复古卡通像素化，针对这种风格，有几大着色特点：

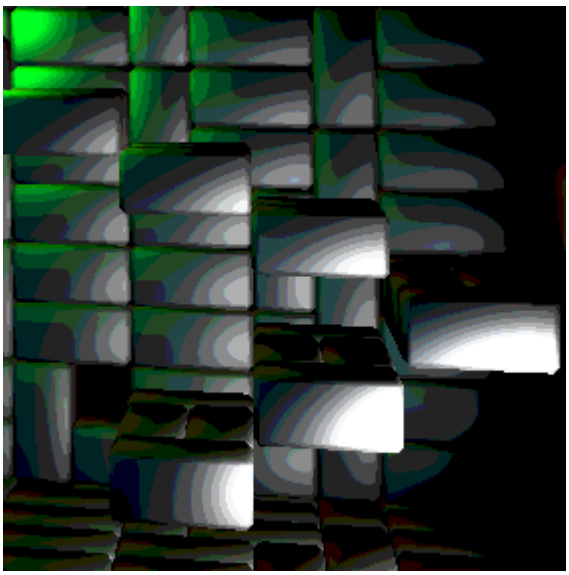
- 锐利阴影
- 高亮减弱
- 轮廓描边

在我们的实际实现中，使用多tone硬着色方法，在着色器输出端将连续光照映射至离散调色板色值，实现高光减弱和填充风格的渲染效果。

具体的实现细节可以由以下的着色器代码显示：

```
int r = int(color.r * 255) / level;
int g = int(color.g * 255) / level;
int b = int(color.b * 255) / level;
vec3 result;
result.r = float(r * level) / 255.0;
result.g = float(g * level) / 255.0;
result.b = float(b * level) / 255.0;
return result;
```

将以上着色逻辑实际应用后得到如下图形效果：



课程设计个人总结

此次小学期历时十多天，在这短短十多天里，我和我的团队一起完成了许多难度不小的工作，也学会了运用各种协作工具来加速团队产品的迭代，可以说是一次以小学期为名的“黑客马拉松”活动。体验本身无疑是新鲜刺激的，工作过程也是卓有乐趣的，而带来的收获也同样是令人回味。

其中学习到的各种工具和方法早已在前文中讨论，这里更多想到的还是宝贵的团队协作经历中得到的收获。和我以前想象的不同，在一个高难度的项目流程中，要调动起每一个成员的工作热情，充分利用好每个人不同的强项能力，不是一件轻松的事情。设计精巧的协作环境是一部分因素，整体项目的分解和规划也是一个因素，作为工作个体为团队着想的同理心更是重要。要将自己锤炼成为一个善于合作的技术人才是一项长远而意义重大的系统工程。希望这次小学期的经历能在以后的工作学习中长期发挥指导作用，为迈向工业界生产做好自我提升。

最终，经过许多挫折和耐心的学习，我们克服了大部分的障碍，得到了现在呈现出的小游戏成果。也许就游戏本身而言，除了画面有创新、构建链完备外，它不是一款有实际竞争力的产品，但作为这一小段时间的辛劳成果，它无疑标志了我们在小学期乃至在大二学习生活中能力的质变性提升和成熟。也象征着我们向一线程序工程师迈出的一小步，未来的路无疑还很漫长，只期望今日今时的这份热情能一直伴我前行。