

# 3D版星噬

——基于OpenGL ES 2.0 的安卓端3D太空物理益智游戏

杨煜溟 李杰 - December 11, 2015



## 游戏简介

星噬是2009年独立游戏节的一款获奖作品，它凭借玄奇抽象的风格，唯美的画面，动听的音乐，以及简单有趣的玩法，吸引了大量的玩家。我们希望的是，在吸取此游戏的优点基础上，做出一款3D版的星噬，把游戏从二维的物理益智游戏，提升为3维的游戏，让玩家能够更加真实体验到行星在太空中运行的感觉，充分发挥玩家的空间想象力。



## 玩法概述

游戏图标



打开我们的游戏，首先出现的是一个欢迎界面，非常直观的表明了我们游戏的内容，如图所示：



之后是一个很经典的游戏菜单界面，通过此界面，我们可以选择开始游戏，获取游戏帮助，进行游戏相关设置等。如图所示：



我们总共设计了24个不同的关卡，玩家可以通过上下滑动右侧屏幕来选择关卡进行游戏，当前关卡会以数字显示在屏幕上。

游戏的主要内容是，玩家控制自己的星球，在太空中寻找比自己体积更小的星体来进行吞噬，同时规避体积比自己大的星体避免自己被吞噬掉。不同的关卡有不同的获胜条件，最常见的一个就是吞噬掉其他所有星球。

我们定义了多种不同的星球：玩家星体，普通星体，恒星，斥星，混乱星，黑暗星。具体如下：

玩家星体：玩家控制的星体

普通星体：非常普通的星体

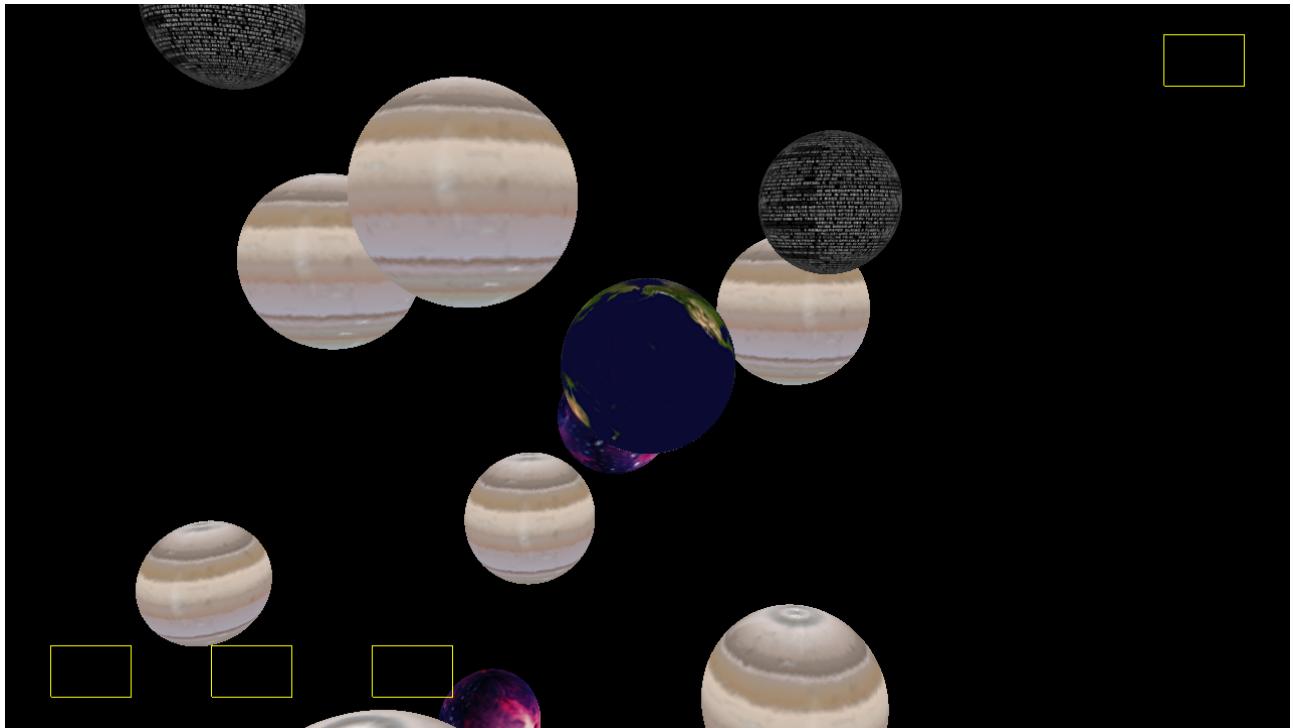
恒星：固定在星图中心，产生引力场，不会改变大小

斥星：产生斥力场

混乱星：当玩家星体接近一定范围时，用户的控制输入将会混乱，无法正常控制玩家星体

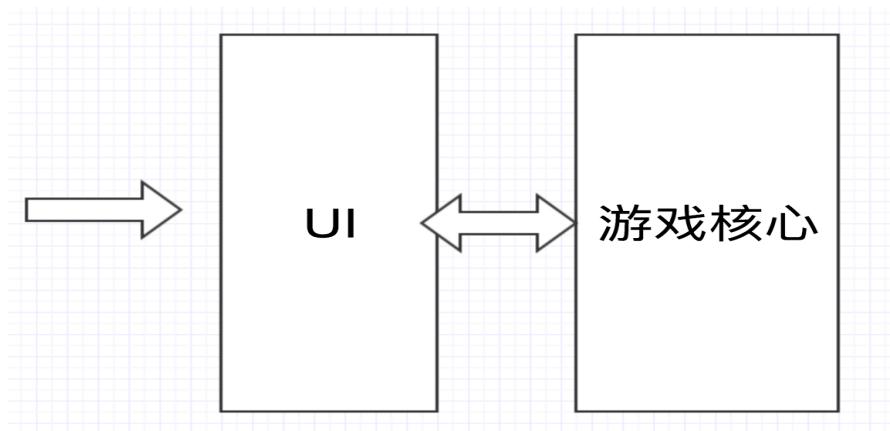
黑暗星：非常黑暗的星体，难以发现。

我们定义了4个操作：加速前进，减速，加速后退，以及通过触摸屏幕来调整前进的方向和视角。通过这几个简单的操作，我们就可以控制自己的星体在太空中灵活的驰骋，来达成获胜的条件。游戏界面截图如下：



## 架构分析

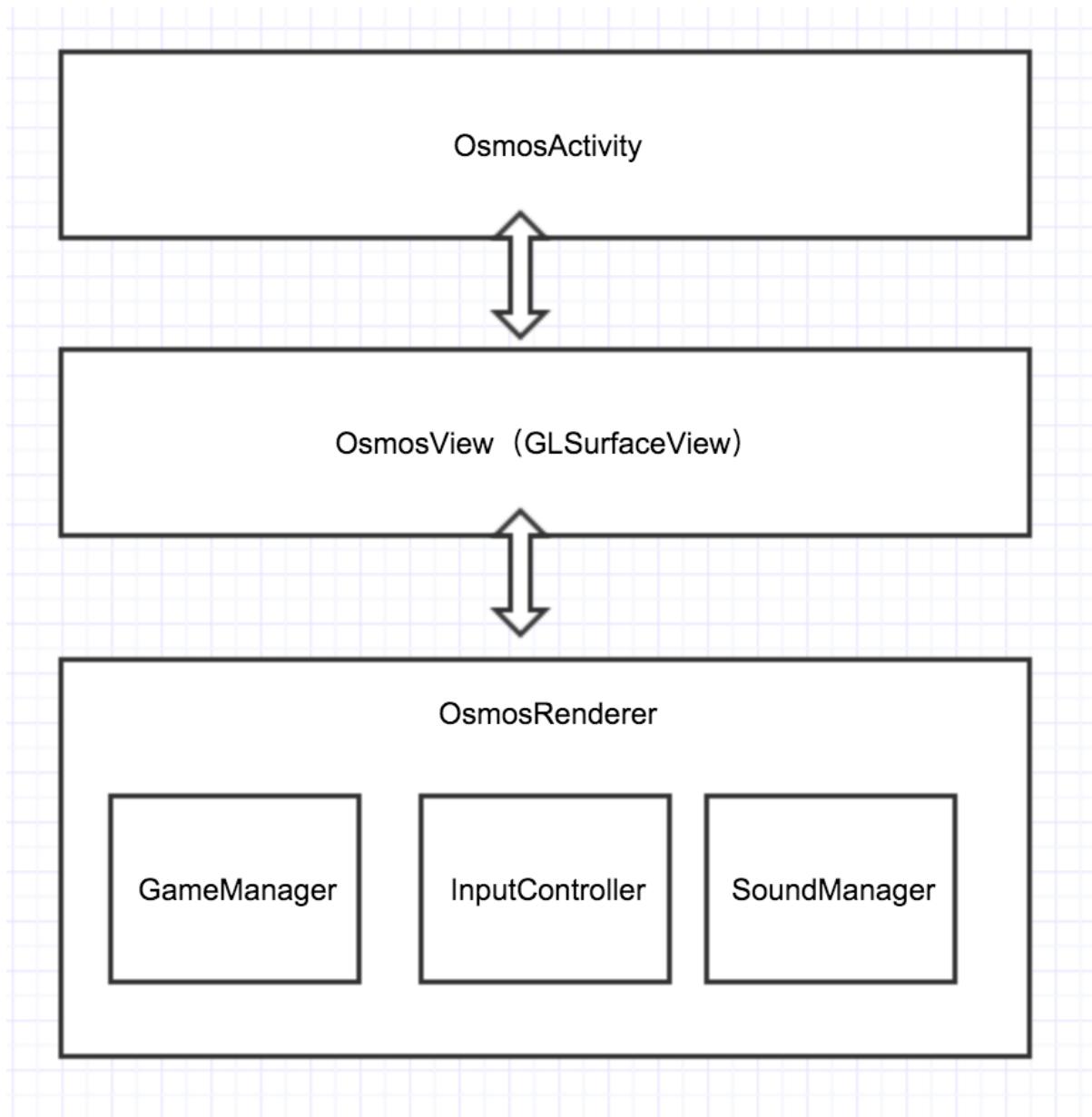
### 整体结构



## 游戏结构

对于如何绘制出3D的太空场景以及各种星体，我们打算利用原生的OpenGL ES 2.0，直接调用Android底层硬件，而非利用第三方的三维游戏引擎。这么做的原因，一是为了更加灵活，不必受限于别人的API；二是能够更好的了解Android的图形编程，加深我们对Android的理解。

我们的游戏核心结构如下：



其中OsmosActivity是管理游戏的Activity，其下拥有一个重载了GLSurfaceView的OsmosView，负责显示OpenGL绘制出的图形。在OsmosView之下，是OsmosRenderer模

---

块，此模块包涵了三个子模块：GameManager模块，InputController模块，以及SoundManager模块。游戏的核心部分的实现都在Osmos Renderer中，下面我们将对其做进一步的介绍。

GameManager负责管理游戏中的游戏对象，比如新的星体的生成，旧的星体的陨灭消失，以及当前的游戏状态（暂停还是游戏中），当前游戏地图（或者说星图）的大小范围等。

InputController负责管理用户的输入，监控用户是否按压了按钮，是否拖动屏幕等，然后会据此计算用户输入对游戏的影响，比如用户拖动了屏幕视窗，对应的角度应该有怎样的变化等，这些参数会保留下供Osmos Renderer使用。

SoundManager顾名思义，是控制游戏音效等模块，当发生特定的游戏事件时，会通过SoundManager来播放指定的音效。

最后，Osmos Renderer需要协调其所包含的各个子模块，大致功能如下：

- 1.负责渲染绘制GameManager所管理的所有需要动态变化游戏物体，如各种星体，以及不需要动态改变的物体，比如界面的按钮等。
- 2.接收InputController的参数，计算用户触摸拖动后所需要改变的各种透视矩阵、投影矩阵、星体的位置、速度等。
- 3.监测各个游戏物体的位置、速度等信息，进行引力场、斥力场等物理法则的应用，以及碰撞检测，各个星体的吸收，边界监测等，并对游戏状态进行更新。

## 实现细节

### UI设计

欢迎界面设置短暂停留，向玩家展示游戏的总体印象。之后进入游戏主界面，MainActivity采用横向的线性布局。左侧添加四个TextView做为选项，采用垂直线性布局。右侧加入一个滑动选择器，可通过上下滑动进行设置关卡。

为游戏开始按钮设置监听器，当出发按键操作时响应事件为打开OsmosActivity，之后可正式开始游戏。当游戏胜利或失败后，会进入相应的Activity，当点击屏幕时，弹出提示框询问是否继续。选择否跳回主界面。

此外，游戏记录了最高得分并且提供给玩家帮助选项，可通过点击对应按钮进行查看。UI部分具体细节不做过多叙述。

## 星体绘制及优化

星体的绘制其实很简单，只需要加载对应的模型文件（对星体而言，即：球），然后贴上星体表面的纹理。为了能让Android手机在有限的资源下顺畅的绘制出众多3D对象，我们在星体绘制上做了大量的优化，让星体尽可能的共用资源。比如有两个一大一小的同种星球，我们是用同样的数据在两个地方以不同的矩阵绘制一次，而不是为每个星体都分配单独的数据。此外，我们还压缩了纹理图片，使其尽可能小。

## 物理法则

星噬的一大亮点就是做为一款物理游戏，将各种引力等实现的非常好，我们的游戏也在物理法则上下了不少功夫。

首先是引力场（有恒星产生）和斥力场（由斥星）产生。我们是用的公式其实就是著名的万有引力公式：

$$F = \frac{GMm}{r^2}$$

唯一不同的是，我们的游戏由于尺度和真实世界不同，所以G值的数量级做了适当的修改。这是我们使用的G：

```
final double G_CONST = 0.67f;
```

以引力的实现为例，我们先求得一个星球距离场源星球的距离和方向向量：

```
double distance = source.getWorldLocation().distance(target.getWorldLocation());  
Point3F direction = sourcePos.subtract( targetPos ).normalize();
```

然后代入公式，求得当前引力的影响程度：

```
POINT3F v = target.getTargetVelocity(),  
double factor = ( G_CONST * sourceM / Math.pow(distance, 2) );
```

最后用这个影响因素与方向向量的乘积做为加速度，并对星球做出改变。

为了简化游戏，我们规定，恒星与斥星各自力场互相不起作用。

## 星体吸收

星体吸收是个动态变化的问题，需要根据两个星体的相对位置以及各自尺寸进行更新。我们星体吸收的设计是保证吸收前后两个星体的体积保持相同，通过此前提，我们可以列出两个等式：

$$\left\{ \begin{array}{l} \frac{4}{3}\pi r_{10}^3 + \frac{4}{3}\pi r_{20}^3 = \frac{4}{3}\pi r_{11}^3 + \frac{4}{3}\pi r_{21}^3 \quad (\text{same volume}) \\ d = r_{11} + r_{21} \quad (\text{distance between them is same}) \end{array} \right.$$

我们已知吸收前的两个星体半径 $r_{10}$ 和 $r_{20}$ ，以及吸收时两星体中心距离 $d$ ，这样就能分别求出吸收后两个星体新的半径，分别对其更新半径：

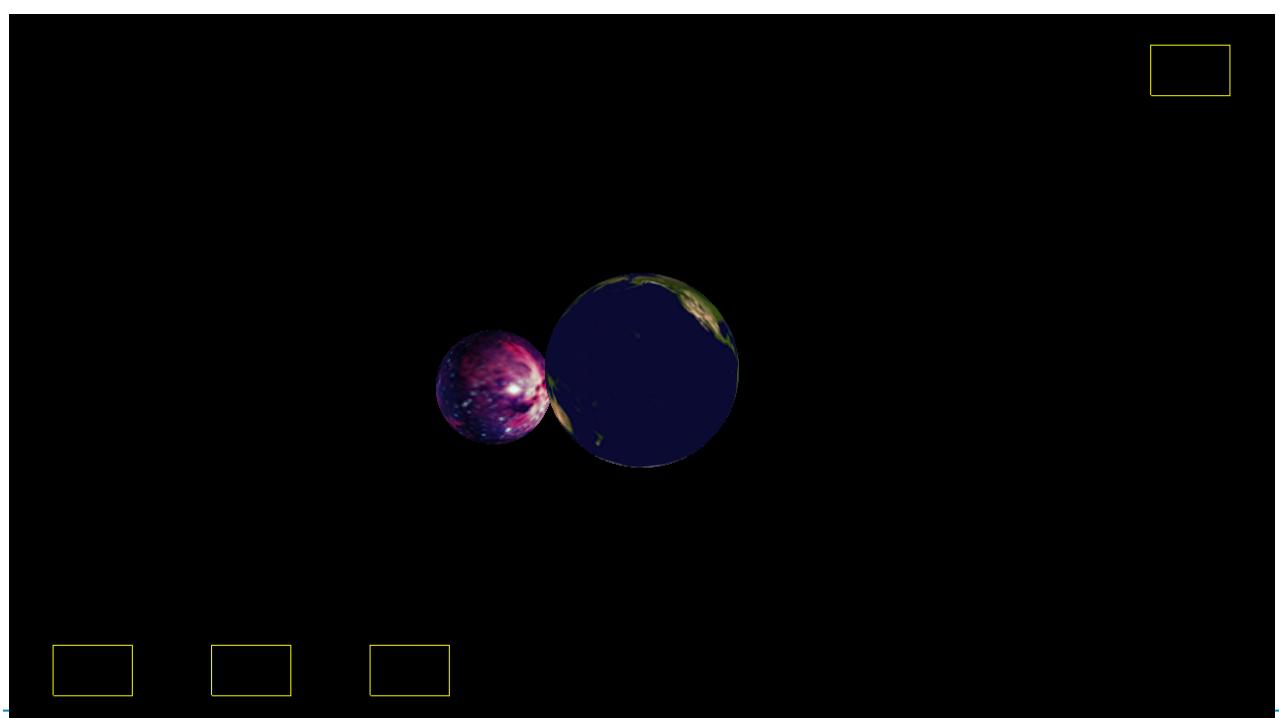
```
bigPlanet.set_radius(Math.sqrt(wholeVolume / 3 / distance - Math.pow(distance, 2) / 12.0) + distance / 2);
smallPlanet.set_radius(distance - bigPlanet.get_radius());
```

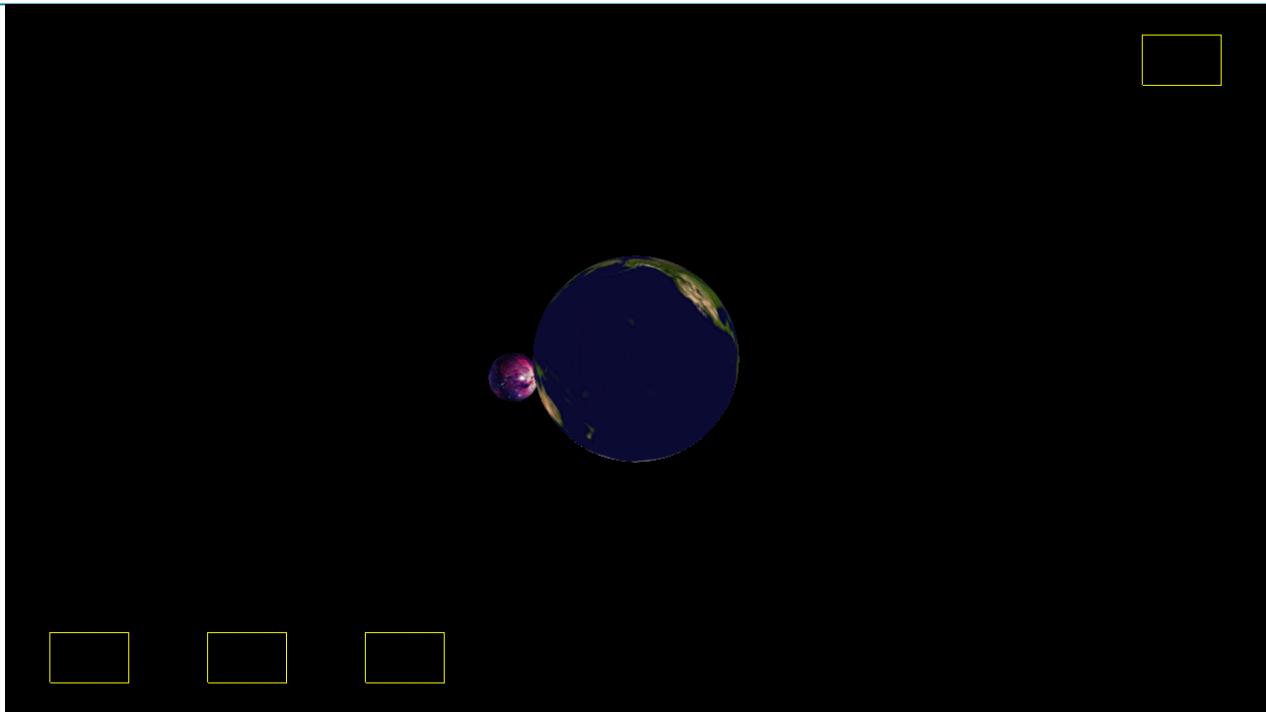
当然，对于两个星体间距小于较大星体半径这种特殊情况，我们直接将小星体半径设为0，将所有体积赋予大星体。

之后，会根据动量守恒定律对星体速度进行更新，如图，其中 $M_1$ 和 $m_1$ 分别是星体的体积：

```
// v3 = v1.multiply( (float )M1 ).add( v2.multiply( (float)m1 ) );
v3 = v3.multiply( (float) (1/(M1+m1)) );
```

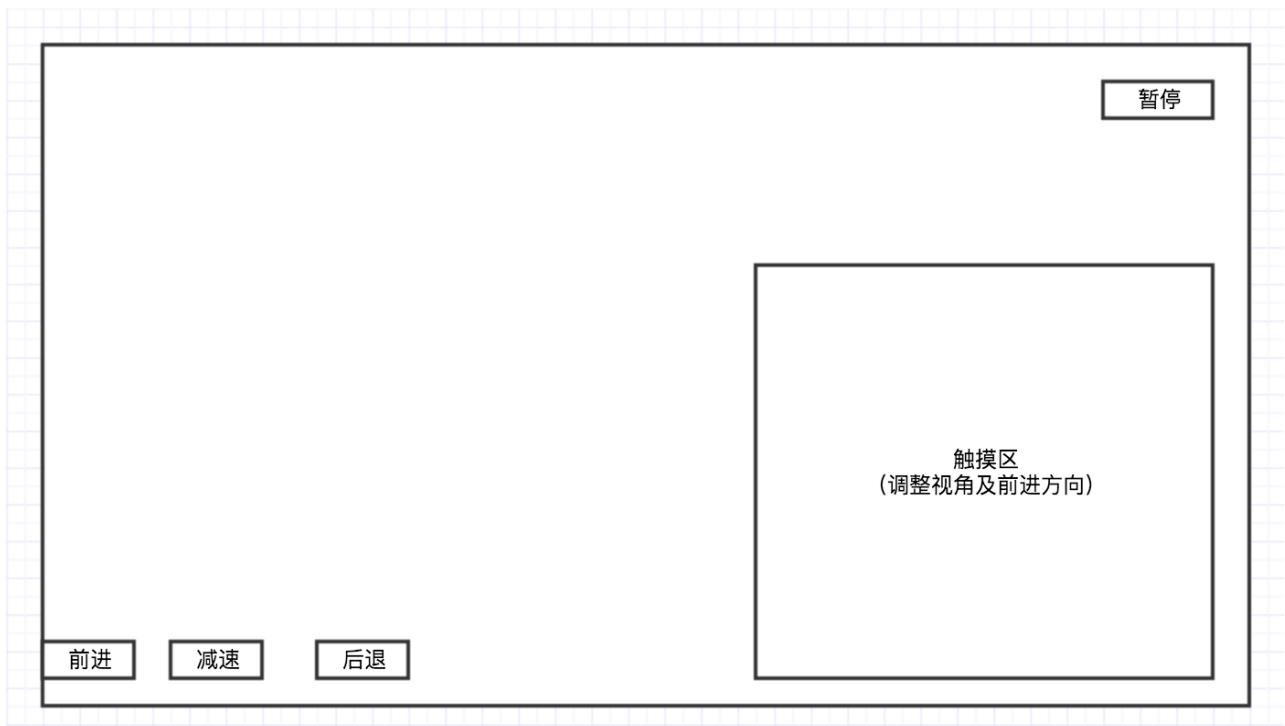
最终我们实现的星体吸收效果如图所示：





## 操作实现

正如之前在开题报告中所言，我们打算充分利用安卓手机触摸屏的优势，通过玩家对触摸屏不同的操作来实现对游戏物体的控制。我们的游戏设置为横版的，其输入结构如图：



一共有4个按键，其功能如图所示；还有右下角一块区域是不可见的触摸区，其功能和许多经典的FPS游戏一样，通过将手指放在其中拖动，可以旋转视角，选择前进的方向。当玩家的手指在触摸区中拖动时，会记录拖动的x, y方向的偏移量，并将此偏移量转化为视角转动的角度保存下来供进一步计算OpenGL绘制所需的必要矩阵。

---

当玩家用双指在屏幕上收缩拖动时，会拉长或缩短视角，玩家可以很方便的调整视角远近，既能远观，也可细察。

## 开发困难及解决

### 困难一：OpenGL ES 2.0 环境搭建

我们要让通过OpenGL ES来绘制游戏场景，首先要做的就是搭建一个类似简单游戏引擎的框架，然后把我们游戏对象的数据传入其中，让其为我们绘制出来。

解决方案：

参考了多个使用了OpenGL ES 2.0 的安卓程序实例，综合多方考虑，最后设计出了上述所描述的结构图。经测试，此设计简洁有效，在后续的游戏编程中给了我们很大方便，完美达成我们的目标。

### 困难二：纹理贴图以及绘制纹理

我们的游戏要求我们不仅绘制出一些几何物体，还需要一定的对象表达能力，这就要求我们需要进行纹理贴图。

解决方案：

纹理贴图花费了我们大量的时间，纹理老是显示不正常。期间也尝试过网上找的一些第三方的库，但是用的不顺手，问题没有解决。后来花费大量时间重新调试自己写的加载代码，终于找到了一个很蠢的错误，解决问题。

### 困难三：星体位置及视角不符预期

编写游戏时，我们发现，一个星体的位置会时不时的不符预期。令我们疑惑的是不符预期的行为并不是总是出现，这个问题几乎困扰了我们游戏的整个研发周期。

解决方案：

对代码仔细检查后，确认自己写的部分是没有问题的。最后我们确定发生错误的可能地方是调用安卓API的地方。经过仔细的调试，发现他原本提供的函数与表面看起来的功能有细微差别，容易让人误解。最后，我们解决了这个问题。

### 团队合作：

杨煜溟： UI设计及流程控制

李杰： 游戏界面绘制及游戏引擎

