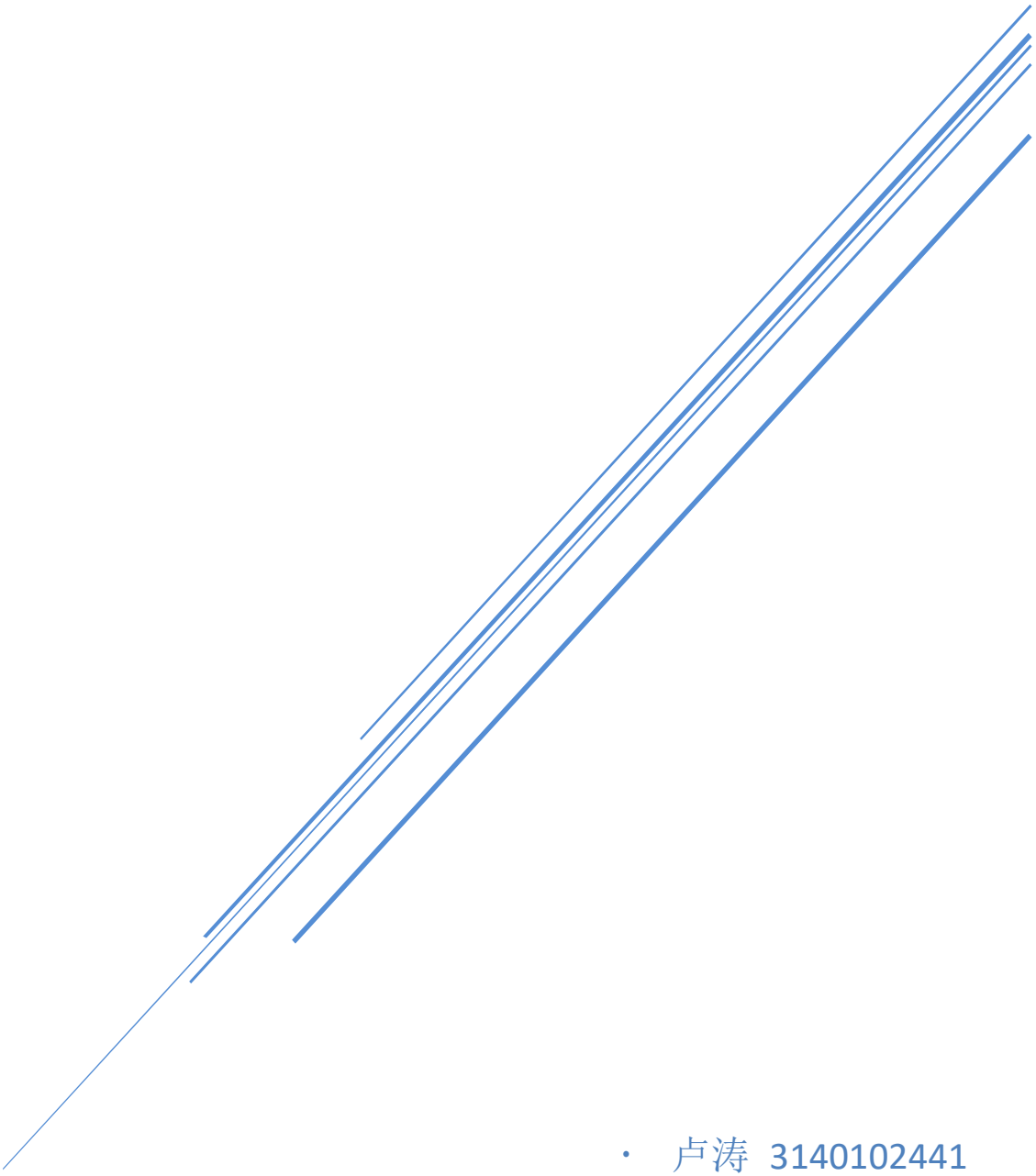
# [RUNNING CAR]

## [系统设计说明报告]




- 卢涛 3140102441
- 尹健文 3140105206

# 图形学大作业——RUNNING CAR

卢涛　3140102441　尹建文 3140105206

## 一、项目介绍

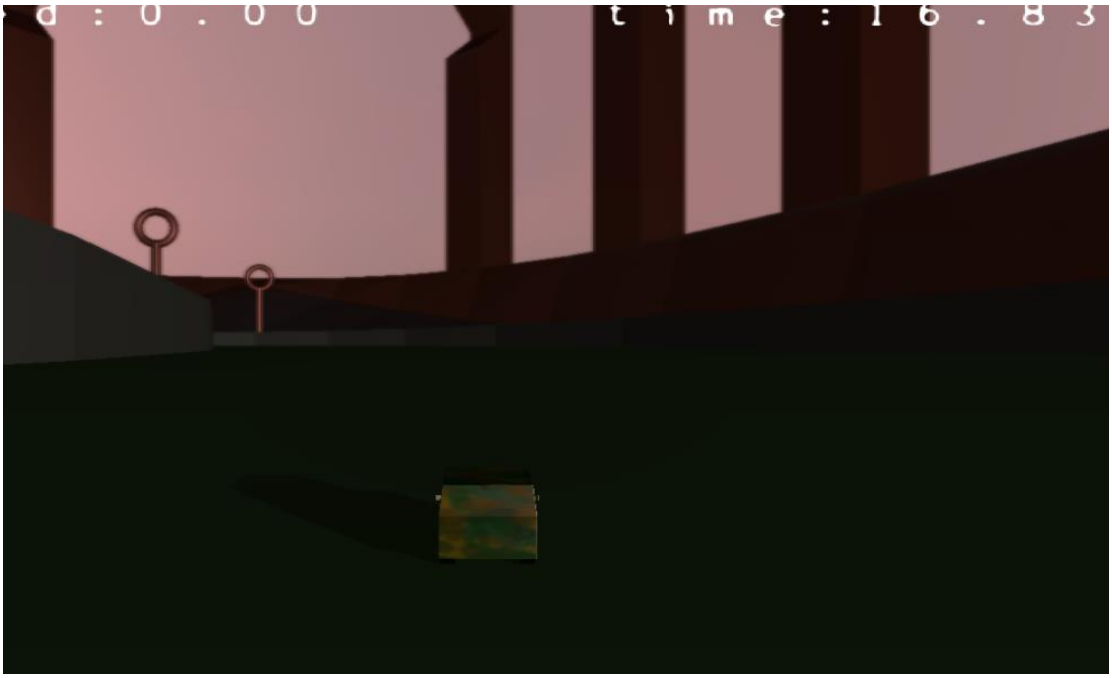
本次大作业，我们组做的游戏叫 Running Car，是一款赛车类游戏，游戏的主要内容就是玩家可以控制赛车在场景内驰骋，并含有光照模型、碰撞检测、运动模糊、实时阴影、体积光等效果。

操作说明：up、down、left、right 控制赛车前进的方向，space 打开车灯，v 切换视角，shift 可以减速并加大转弯半径以达到漂移的效果，ctrl 进行加速，f 进行波的释放。

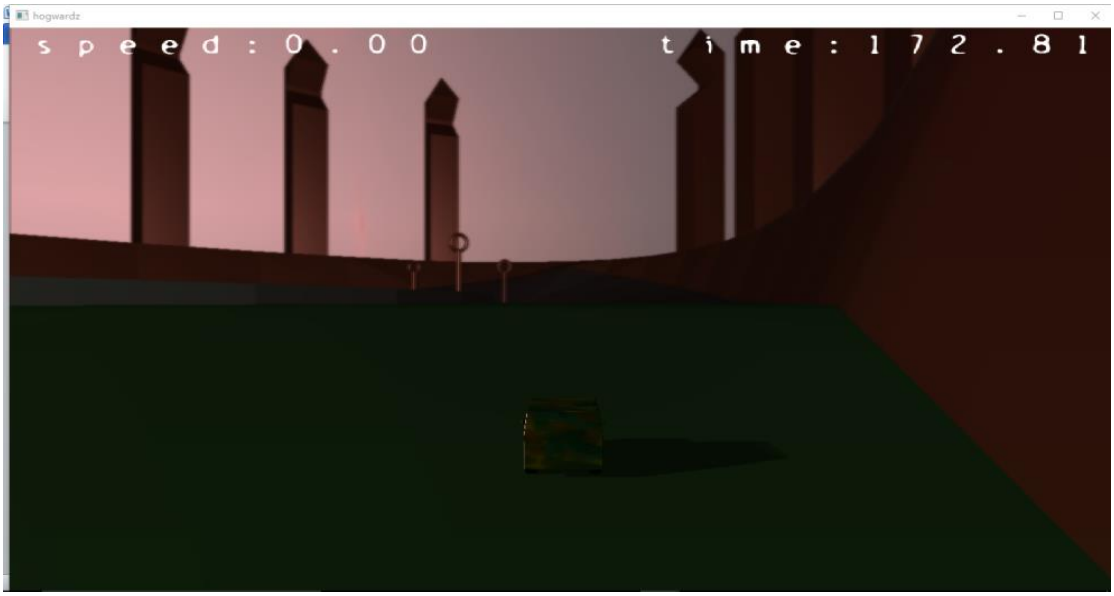
实现工具：opengl3.3

## 二、实现的功能

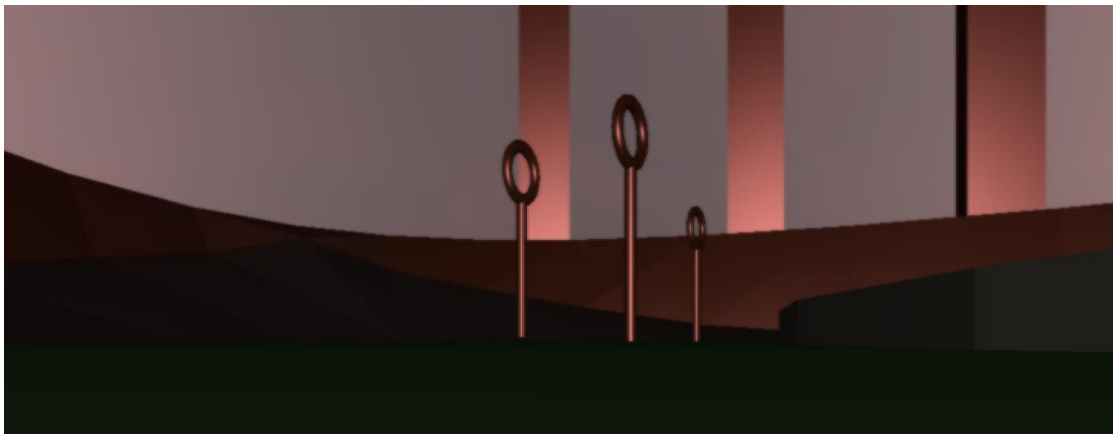
**1 基于 OpenGL，具有基本体素（立方体、球、圆柱、圆锥、多面棱柱、多面棱台）的建模表达能力**



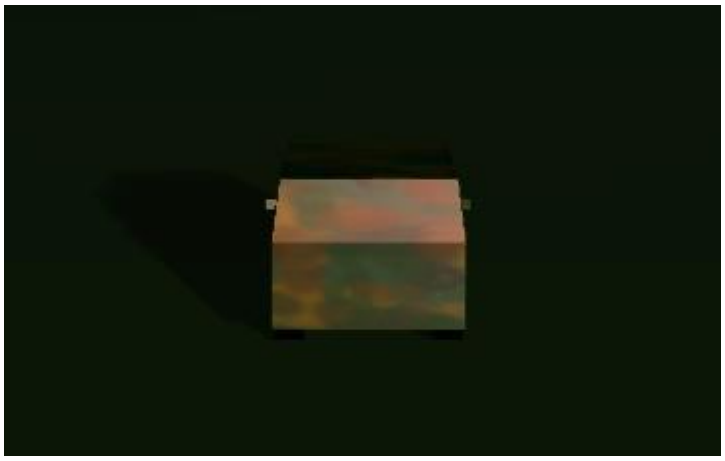
**2 具有基本三维网格导入导出功能（建议 OBJ 格式）**



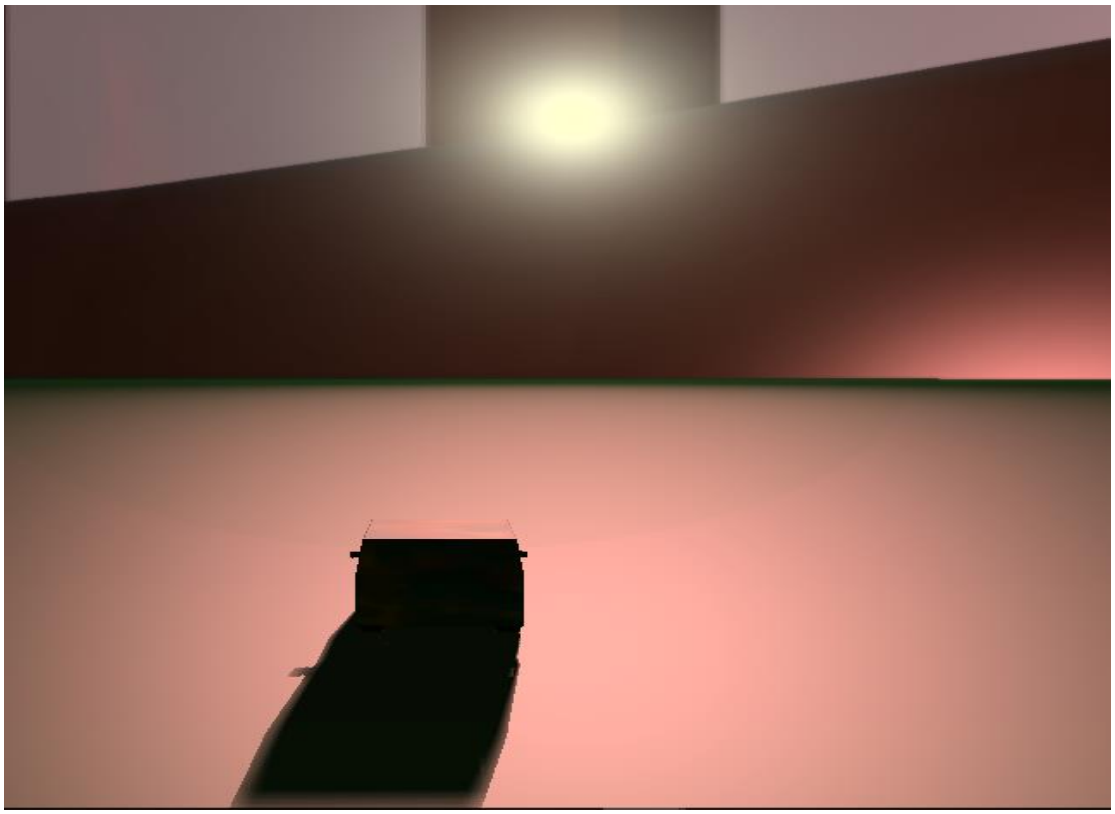

3 具有基本材质、纹理的显示和编辑能力



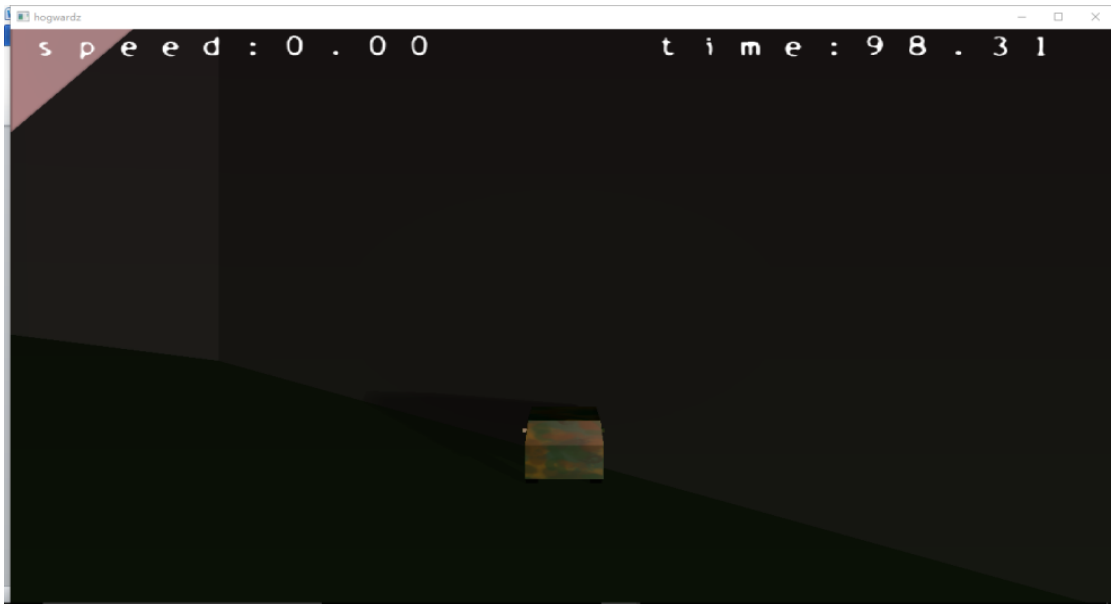
4 具有基本几何变换功能（旋转、平移、缩放等）



5 基本光照明模型要求，并实现基本的光源编辑（如调整光源的位置， 光强等参数）

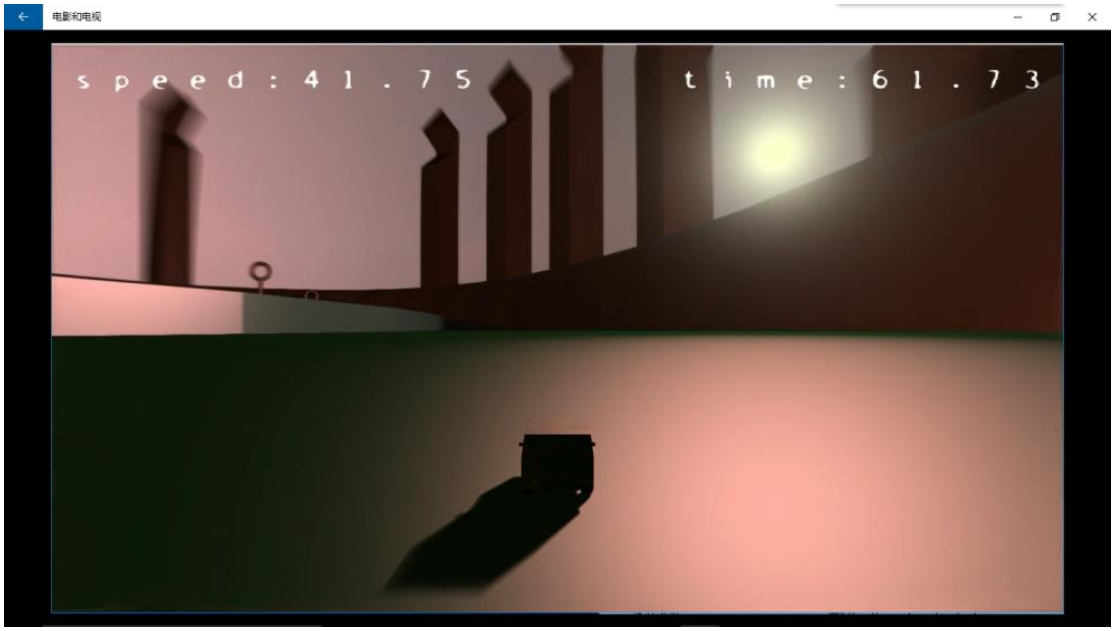


6 能对建模后场景进行漫游如 Zoom In/Out， Pan, Orbit, Zoom To Fit 等观察功能

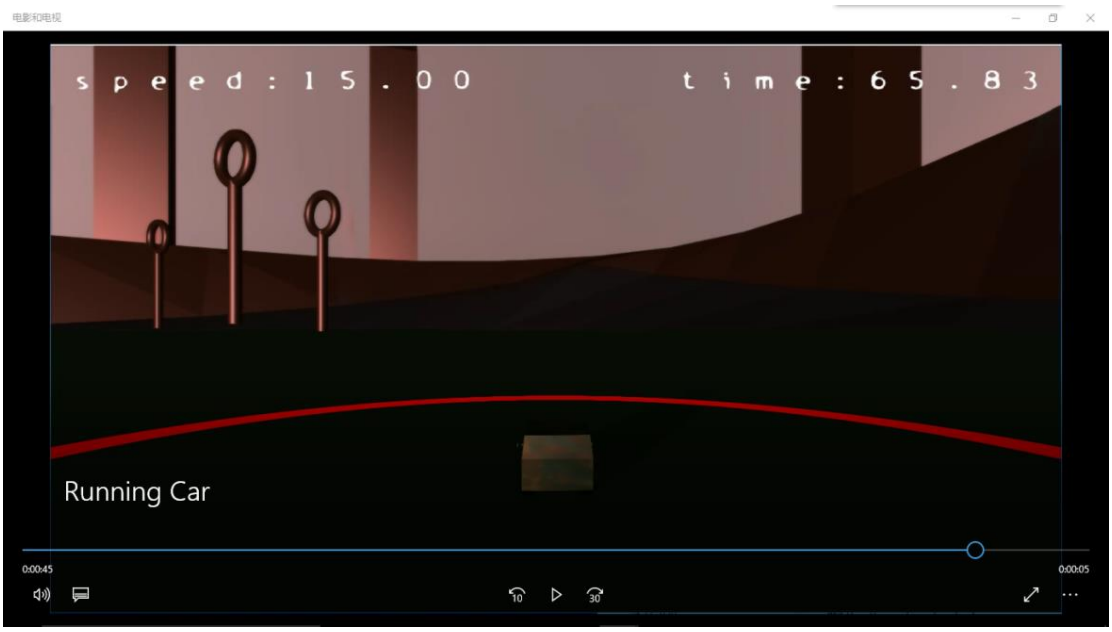


7 Awesomeness 指数：展示项目本身所独有的炫酷特点，包括但不限于 有感染力的视觉特效

体积光

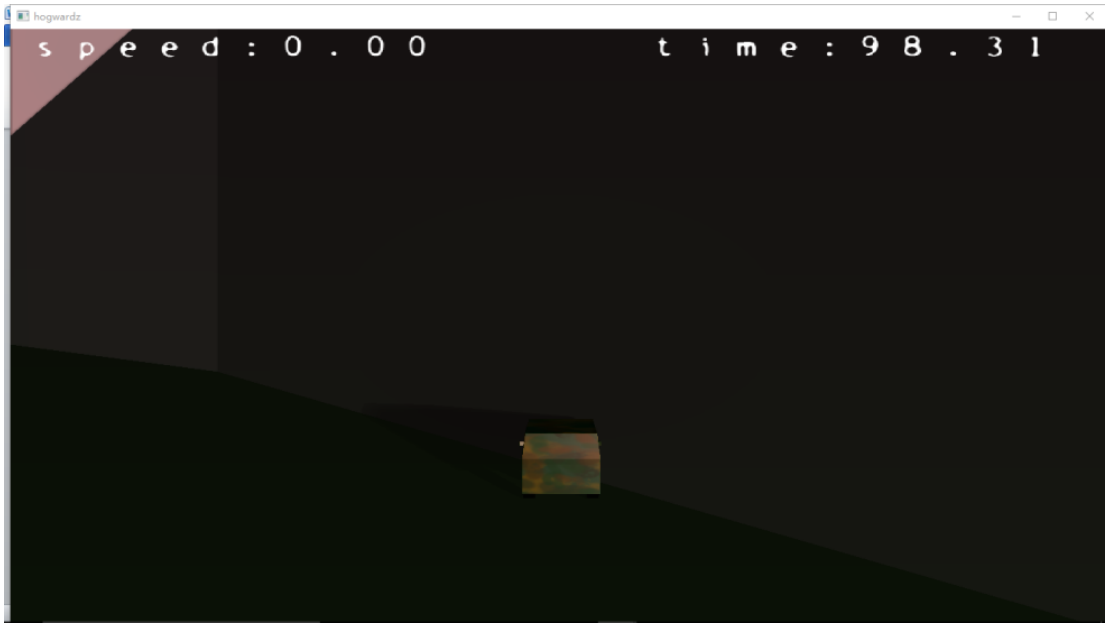



动态模糊（左上角）

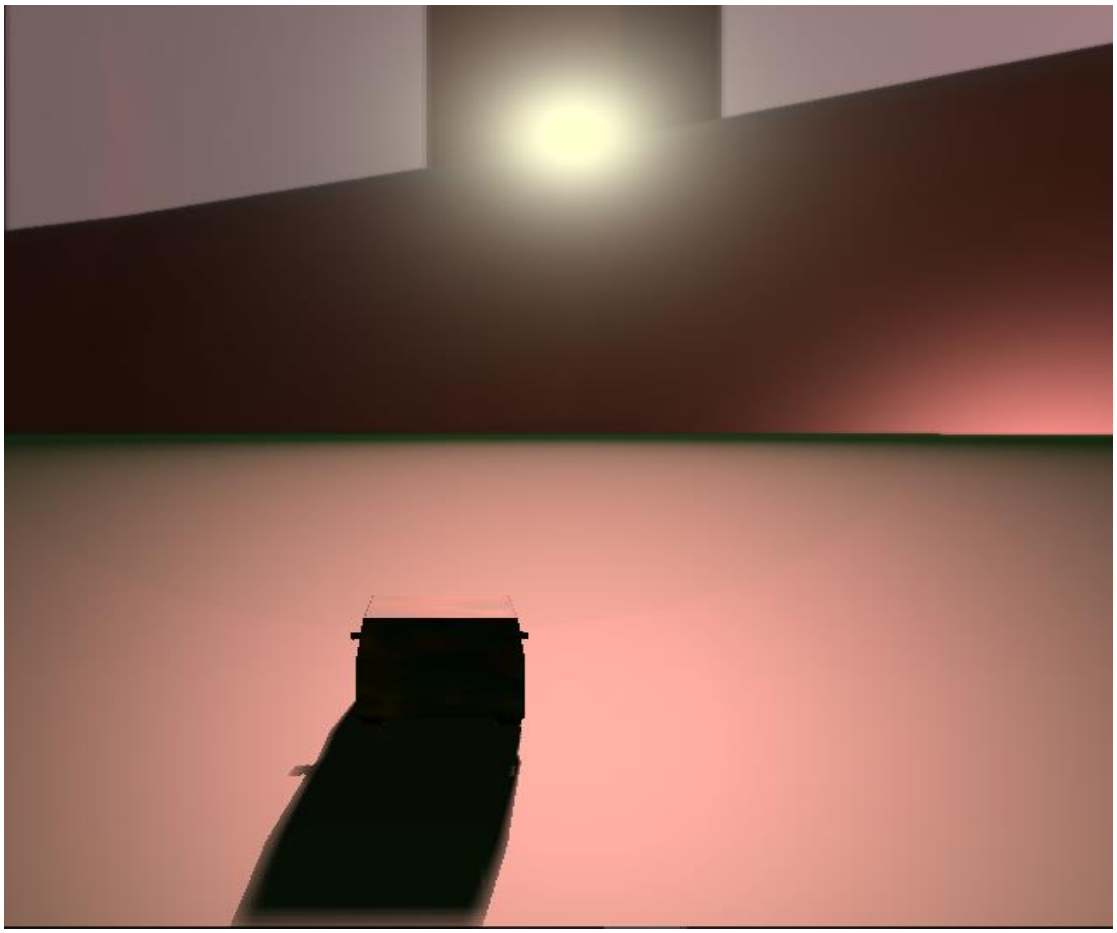



动态波

8 漫游时可实时碰撞检测(加分点)



**9 光照明模型细化，可任选实现实时阴影、Caustics、位移纹理、全局 光照明（光子跟踪）、辐射度、AO 叠加等（加分点）**



**使用 PCSS 技术实现**

## 三、实现方法

### 1 建模、obj 导入、纹理贴图

这几项功能都是最基本的，我们是在 Blender 里面做好模型以及贴图，车包括车身和轮子，赛道是模型加贴图，然后再导入的，这里就不细讲。

### 2 基本几何变换、漫游、碰撞检测

前进、后退使用平移矩阵实现，转弯通过旋转矩阵，旋转车身和 camera 达到转弯的效果，碰撞检测是通过判断前一帧与后一帧车身连线线段与每个面片是否有交点，若有交点，则判断发生碰撞，根据碰撞前的速度大小和方向，决定碰撞后的速度大小和方向。此外，我们还做了上坡的碰撞检测，若面角度小于 30，

则车可以开上坡，而不会被挡住。

### 3 光源

真实光源是一个点，每个面片将其与 camera 连线，再计算该直线到真实光源的距离，距离越近，亮度越大，由此可以实现一个带光晕的光源。

### 4 体积光

在片段着色器中计算片段、光源两点的连线与聚光灯正射方向之间的夹角α大• 小，设定聚光灯光圆锥体的张角为一个常数 Θ，若 α 大于 Θ/2，则不点亮，若小于，则点亮。这样就完成了基本的一个聚光灯的效果。

用两个角度值来表示一个聚光灯的张角：较大的角Θ表示聚光灯能照射到的圆锥体空间的张角，若α大于Θ/2，则不被照亮；较小的角 β 表示聚光灯能完全 照亮的圆锥体空间的张角，若 α 小于 β/2，则光强为一个较大的常数；当 α 在两角之间,则用一个平滑插值 smoothStep 函数来计算这些中间区域的光强。这样就完成了一个光亮区域边缘软化的聚光灯效果。

在光束圆锥空间上实时绘制一个半透明的圆锥，这样就可以绘制一个简单的丁达尔现象。

在绘制光束圆锥体的时候将结果绘制到一个独立帧缓存绑定的纹理上去，然后用高斯模糊将其打散,在显示最后结果的纹理上再将打散之后的光束圆锥体画上去，这样就绘制出了一个比较真实的丁达尔现象。

在片段着色器中绘制光束圆锥体时，根据片段与光源的距离去改变该片段的不透明度，在绘制被照亮的物体时，根据片段与光源的距离去改变光照的强度，岂可获得一个光亮有限且随距离柔和递减的光束。

在片段着色器中绘制光束圆锥体时，根据片段的法线方向，去改变不透明度，即可将一个整体的光束打撒成若干束光合成一个圆锥体光束的效果。（这一效果属于参数错误时意外所得）。

### 5 动态模糊

我们实现动态模糊的过程是，将清晰的世界直接绘制到一张绑定到某个帧缓

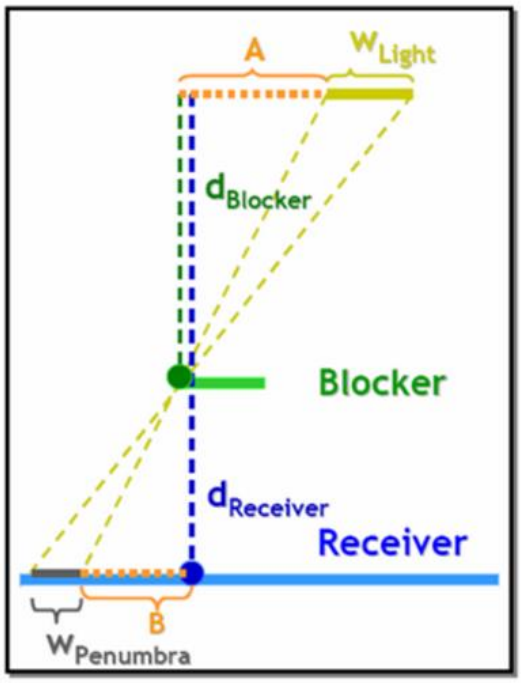
冲区的颜色缓冲区中，这一过程没有记录任何速度信息，然后对该纹理进行径向多重采样，绘制到屏幕中，采样率正比于待绘制的片元偏移中心的程度。

没有记录深度，所以不管远近快慢，只要是与中心等距的片元都有相同的模糊程度。

### 6 PCSS 技术实现实时光影

PCSS 技术是基于 Shadow Mapping 的软化阴影技术。

现实中的光源都是有一定体积的，所以这样的体光源照射物体投下的影子就是有全影和半影的区分。




$$w_{Penumbra} = \frac{(d_{Receiver} - d_{Blocker}) \cdot w_{Light}}{d_{Blocker}}$$

- Assumes that blocker, receiver, and light are parallel

根据简单的相似三角形原理，已知光源的尺寸（W light），光源到遮挡物（blocker）的距离，光源到被投影物体（receiver）的距离，就可计算出半影的宽度。

而传统的 ShadowMapping 技术下的影子是二元的，这个片段要么是距离光源最近的片段，就被点亮，要么就不是距离光源最近的片段，被遮挡住，非黑即白，俗话说，这个影子很硬，而且由于深度贴图是离散的方格，所以最后投射的影子是一定有锯齿的。于是 PCF 技术应运而生。简单的说，PCF 技术就是对深度贴图的多重采样，与 ShadowMappinp 相比，每一个片段的阴影暗度是受该片段对应

的深度贴图上的一点及其周围若干点的深度决定的。简单的说，这个片段多暗不光由其本身决定，还由它周围的片段决定，周围被遮住的片段所占的比例越高，它的暗度就越大。而采样的范围通常是个定值，所以影子的边缘都一样的软，不管什么部位边缘的样子都是一样的。
这很不科学。

所以这个采样区域的大小必须是动态的，而之前说到的半影的大小正好可以与其对应。所以一切具备，只需要获取三个值：光源的尺寸（W light）、光源到遮挡物（blocker）的距离、光源到被投影物体（receiver）的距离，就可以绘制科学的柔化的阴影了，而光源的尺寸可视作常量，光源到被投影物体的距离在着色器中很容易计算，而光源到被投影物体的距离就不那么简单了，如果只是从一个深度贴图上采样得到，会产生一个问题，在影子外围本应该投影上一个虚化的半影的片段由于找不到它对应的深度贴图上的点，无法得到它对应的 光源至遮挡物的距离，计算无法正常进行下去，所以光源至遮挡物的距离也是一个多重采样的过程，只不过是一个较为特殊的多重采样：将采样区域扩大（这可以是个定值），在深度贴图上找到的深度小于原始片段深度的采样点被认为是可接受的，在多次采样后将这些可接受的采样点的深度做平均，得到一个有效的，对应该片段的 光源到遮挡物的平均距离。这样之后，三个值都齐全了，计算可以正常进行下去。

说明：有关代码的原理来源于 NVIDIA DX10 SDK：PCSS，着色器程序在 DX10 的 HLSL 代码上改编（减少了若干影响性能的函数，并翻译成 GLSL）。