# 计算机图形学

第14组期末项目报告

**小组成员**

|  |  |
| --- | --- |
| 姓名 | 学号 |
| 张寅哲 | 16340295 |
| 郑国林 | 16340304 |
| 刘俊峰 | 16340150 |
| 谢玮鸿 | 16340256 |

## 项目介绍以及实现结果

**项目介绍：**本项目实现了一个**户外场景**下的**赛车游戏**，可以通过**键盘**控制赛车的移动，视角为**第二人称**视角。场景中有汽车，建筑，道路，天空等物体，拥有光照和阴影的效果。通过粒子系统模拟尾气效果，以及在场景边界加入水波效果。在汽车运动过程中，通过文本在屏幕上显示汽车的速度等所需信息。

**实现结果截图：**

具体的效果请看**doc**目录下的**演示视频**。

## 开发环境以及使用到的第三方库

**开发环境**

操作系统：**windows 10**

IDE: **visual studio 2017**

编译器： **msvc++**

**使用到的第三方库**

**imgui**

**glad**

**glfw**

**assimp**

**glm**

## 实现功能列表(Basic与Bonus)

#### Basic:

1.**视角**：处于第二人称视角操控汽车的运动，从汽车的后上方向汽车前方观察，视角跟随汽车移动

2.**光照**：采用phong局部关照明模型实现自然光的照射效果。

3.**纹理**：采用纹理贴图，实现汽车外观以及场景中实体的仿真。

4.**阴影**：采用shadow mapping技术，实时渲染自然光的阴影效果。

5.**模型导入**: 导入汽车和野外场景的obj文件，使其加载于屏幕上。

#### Bonus:

1. **天空盒**：场景的上方为天空盒，模拟自然天空的效果。

2. **文字**：屏幕上会通过文本显示一些汽车相关的信息。

3. **粒子系统**：通过粒子系统模拟汽车的尾气效果。

4.**光照明的优化**：通过改进phong局部光照模型，改善光照模型；使其更加真实。

5.**流体模拟**：在场景边界外模拟水波的效果。

6.**抗锯齿**：在渲染过程中使用多重采样的离屏渲染，减弱锯齿的效果。

## 功能点介绍

1. 游戏架构

游戏分为以下几个部分：

1). 游戏实体：包括汽车、地板、房子等各种物体。

2). 渲染器：用于渲染各种游戏实体。将游戏实体的指针以及着色器指针传入即可。

3). 游戏事件管理：用户按键事件等。

具体流程：在每一个游戏循环中，主线程将各种游戏实体传入到渲染器中，渲染器读取它们的状态，并渲染到画面，如果游戏事件触发，则改变当前实体的状态，在下一个游戏循环中渲染。

2.模型加载

模型加载的部分基本和learnopengl网站上的一致。但是，直接使用上面的源码无法加载出没有纹理的模型（有的模型只有diffuse color，并没有使用纹理贴图），所以，我们要在加载的时候读取mtl文件中存储的颜色信息。

aiColor4D diffuseColor;

aiGetMaterialColor(material, AI\_MATKEY\_COLOR\_DIFFUSE, &diffuseColor);

glm::vec3 dcolor = glm::vec3(diffuseColor.r, diffuseColor.g, diffuseColor.b);

这里我们利用assimp库，将mtl文件中的颜色信息读取到一个3维的向量中。

3. 光照和阴影

基本和作业一致。

4. 碰撞检测

AABB盒碰撞模型。在每个游戏实体中维护一个包围盒（在本项目中没有包含z轴），然后在每个游戏循环中判断，每个游戏实体的包围盒是否有重叠。如果有重叠则碰撞会发生。

bool BoundingBox**::**isCollided**(**const BoundingBox**&** src1**,** const BoundingBox**&** src2**)** **{**

bool axis1 **=** **false,** axis2 **=** **false;**

// axis x overlap

**if** **(**src2**.**minx **>** src1**.**minx **&&** src2**.**minx **<** src1**.**maxx**)** **{**

axis1 **=** **true;**

**}** **else** **if** **(**src2**.**maxx **>** src1**.**minx **&&** src2**.**maxx **<** src1**.**maxx**)** **{**

axis1 **=** **true;**

**}**

// axis y overlap

**if** **(**src2**.**miny **>** src1**.**miny **&&** src2**.**miny **<** src1**.**maxy**)** **{**

axis2 **=** **true;**

**}** **else** **if** **(**src2**.**maxy **>** src1**.**miny **&&** src2**.**maxy **<** src1**.**maxy**)** **{**

axis2 **=** **true;**

**}**

**return** **(**axis1 **&&** axis2**);**

这里是碰撞发生的逻辑。

关键在于如何找到每个游戏实体的包围盒。其实比较简单。在物体加载的时候，我们会加载出很多个mesh，只要我们找出这些mesh里最大的xy和最小的xy即可。然后通过模型的位置坐标对其进行offset。

1. 天空盒

天空盒其实就是一个覆盖场景四周的长方体，但它的各个面上贴有表示天空的纹理图片，即四周的4面纹理的边与顶面纹理的边相连，同时四面纹理前后相连. 在

实际的渲染中，将这个立方体始终罩在摄像机的周围，让摄像机始终处于这个立方体的中心位置.要实现这个的话OpenGL中这种纹理叫做立方体贴图(Cubemap)。为了从立方体贴图中采样，我们要采用3d纹理坐标而不是我们之前用的2d纹理坐标，所以首先加载图片的时候要设置成GL\_TEXTURE\_CUBE\_MAP，而不是我们之前纹理中常用的2D纹理，最后，还要记得设置纹理的过滤和环绕方式，因为我们用的是纹理盒，所以除了平常使用的GL\_TEXTURE\_WRAP\_S和GL\_TEXTURE\_WRAP\_T之外，还需要设置GL\_TEXTURE\_WRAP\_R的属性。

另外在设置shaderr中的view矩阵的时候，为了保持摄像机和天空盒的距离，达到不能靠近天空盒或者说是始终被天空盒包围的效果，我们要把view矩阵中平移的部分去掉，也就是第三列设置成0。

关于着色器部分的话，要对一个3D盒子使用纹理盒有一个巨大的好处就是不需要额外指定纹理坐标。只要盒子是被放置在世界原点上，盒子本身的坐标就可以作为纹理坐标使用，因为在3D世界中位置本身就是一个向量，表示一个方向，我们要的就是这个方向。所以在顶点着色器中我们直接把输入的坐标当做纹理坐标传出，在片段着色器中，我们可以直接用这个坐标对纹理盒进行采样，同时我们可以在片段着色器中设置一些雾的效果，根据坐标的高度来计算出雾的颜色和采样得到的纹理颜色的混合。

1. 文字渲染

首先就是根据教程安装好那个freetype的库，能够用于加载字体并将他们渲染到位图以及提供多种字体相关的操作的软件开发库。在安装的时候还是最好通过自己的vs来进行编译一次，避免产生各种问题。

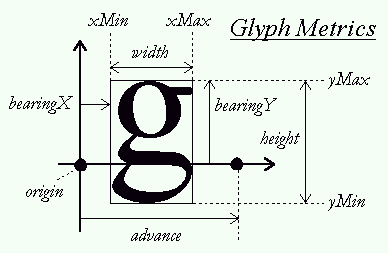
FreeType是一个能够用于加载字体并将他们渲染到位图以及提供多种字体相关的操作的软件开发库。它是一个非常受欢迎的跨平台字体库，它被用于Mac OS X、Java、PlayStation主机、Linux、Android等平台。FreeType的真正吸引力在于它能够加载TrueType字体。TrueType字体不是用像素或其他不可缩放的方式来定义的，它是通过数学公式（曲线的组合）来定义的。类似于矢量图像，这些光栅化后的字体图像可以根据需要的字体高度来生成。通过使用TrueType字体，你可以轻易渲染不同大小的字形而不造成任何质量损失。

FreeType所做的事就是加载TrueType字体并为每一个字形生成位图以及计算几个度量值(Metric)。我们可以提取出它生成的位图作为字形的纹理，并使用这些度量值定位字符的字形。

要加载一个字体，我们只需要初始化FreeType库，并且将这个字体加载为一个FreeType称之为面(Face)的东西。

使用FreeType加载的每个字形没有相同的大小（不像位图字体那样）。使用FreeType生成的位图的大小恰好能包含这个字符可见区域。例如生成用于表示’.’的位图的大小要比表示’X’的小得多。因此，FreeType同样也加载了一些度量值来指定每个字符的大小和位置。下面这张图展示了FreeType对每一个字符字形计算的所有度量值。

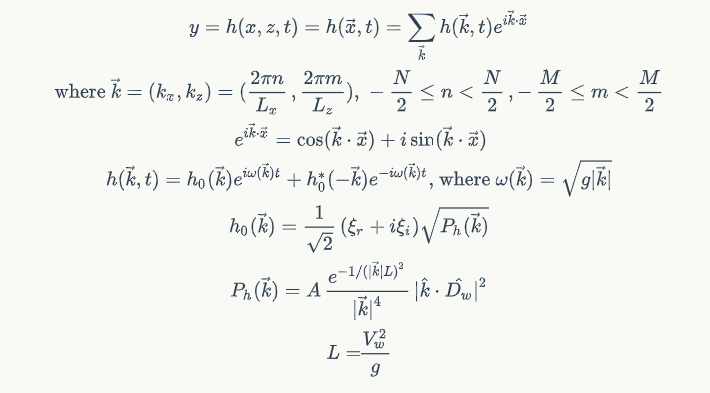
每一个字形都放在一个水平的基准线(Baseline)上（即上图中水平箭头指示的那条线）。一些字形恰好位于基准线上（如’X’），而另一些则会稍微越过基准线以下（如’g’或’p’）（译注：即这些带有下伸部的字母，可以见[这里](https://www.supremo.tv/typeterms/)）。这些度量值精确定义了摆放字形所需的每个字形距离基准线的偏移量，每个字形的大小，以及需要预留多少空间来渲染下一个字形。下面这个表列出了我们需要的所有属性。



1. 水波模拟

要模拟一个比较真实的流动水面不是一件容易的事情，需要考虑如何让波浪看起来自然、如何让水面反射等问题。在OpenGL中，可以将水面绘制成一个连续的高度场。如果以(x, y, z)来描述水面(xz平面)上的一个点，则它的高度y应该有`y = H(x, z, t)`的关系(t表示时间)。这次实现参考了[Ocean simulation part one: using the discrete Fourier transform](https://www.keithlantz.net/2011/10/ocean-simulation-part-one-using-the-discrete-fourier-transform/) 一文，基于傅里叶变换模拟较为真实的水面。

总结来说，计算水面高度的方法为



这堆公式涉及到的变量很多，我们需要控制这些变量来模拟出理想的水面效果。这些变量包括：

1) 水面大小 Lx, Lz

2) 顶点网格密度 N，M

3) 风的方向 Dw 以及 速度 Vw

4) 波浪幅度 A

5) 满足正态分布的两个独立变量 Er 和 Ei

其中函数h的计算，实际上就是二位傅里叶变换的求和，由于水面的波浪是实时计算更新的，为了能有更好的效率，利用二维快速傅里叶变换FFT降低计算复杂度。

总体来说，水面模拟基本涉及到的都是数学计算。在计算机图形学中，通过由三维顶点坐标组成的集合以及顶点之间的连接关系集合，表示一个三角形集合。三角形之间拼接成了水面，顶点的上下波动模拟出水面的波动效果。最终实现效果如下：



8．抗锯齿：

在渲染的时候，如果仔细观察一些边缘的位置，能够看到锯齿状的图案。这很明显不是我们想要在最终程序中所实现的效果。你能够清楚看见形成边缘的像素。这种现象被称之为走样(Aliasing)。有很多种抗锯齿（Anti-aliasing，也被称为反走样）的技术能够帮助我们缓解这种现象，从而产生更**平滑**的边缘。

我们使用多重采样的方式，多重采样所做的正是将单一的采样点变为多个采样点（这也是它名称的由来）。我们不再使用像素中心的单一采样点，取而代之的是以特定图案排列的4个子采样点(Subsample)。我们将用这些子采样点来决定像素的遮盖度。当然，这也意味着颜色缓冲的大小会随着子采样点的增加而增加。

MSAA真正的工作方式是，无论三角形遮盖了多少个子采样点，（每个图元中）每个像素只运行**一次**片段着色器。片段着色器所使用的顶点数据会插值到每个像素的**中心**，所得到的结果颜色会被储存在每个被遮盖住的子采样点中。当颜色缓冲的子样本被图元的所有颜色填满时，所有的这些颜色将会在每个像素内部平均化。

在openGL中要使用离屏渲染的方式进行多重采样，大体思路是为每个像素点产生多个缓冲区，最终渲染的时候将其平均。

最终对比：左图为未使用抗锯齿时的边缘，右图为使用抗锯齿的边缘

9．粒子系统：

基本达到的目标是能够跟随汽车移动，并且在汽车移动速度加快的时候能够与汽车拉开更大的距离。

事实上，一个微粒,从OpenGL的角度看就是一个总是面向摄像机方向且(通常)包含一个大部分区域是透明的纹理的小四边形.一个微粒本身主要就是一个精灵(sprite),当你把成千上万个这些微粒放在一起的时候,就可以创造出令人疯狂的效果。当处理这些微粒的时候,通常是由一个叫做粒子发射器或粒子生成器的东西完成的,从这个地方,持续不断的产生新的微粒并且旧的微粒随着时间逐渐消亡。

这里尾气粒子的基本的实现使用的粒子是相对于车很小的立方体，贴图使用于汽车尾气颜色相近的灰色贴图，其运动是离开车后向外扩散，并且有跟随效果。最后其存活时间存活时间是线性递减的，死亡后重新生成。

最终效果如下：



10.白天与黑夜的变换

后期想要做到一个白天与黑夜渐变的效果，首先是通过两个帧之间相减获得时间差，然后累计时间，得到一个虚拟的时间worldtime，然后通过这个abs((time - 12) / 12)来计算出factor，然后传给天空盒着色器，然后通过mix函数将计算出来的天空盒颜色与黑色通过这个factor进行混合，达到渐变成晚上的效果。另外为了达到更逼真的效果，还将这个factor传给渲染物体的着色器，将环境光ambient根据时间进行调节，使之看起来更加真实





11.车的运动和照相机跟随

为了让照相机跟随车的运动和转弯，首先要设定一个和车的相对高度relativeheight和相对方向relativedirection，然后相机的位置就是车的位置加上相对位置，相对位置的计算车的方向在相对方向的分量，以及相对高度得到

glm::vec3 relativePosition = glm::vec3(-relativeDirection \* car.direction.x, relativeHeight, -relativeDirection \* car.direction.z);

camera.position = car.position + relativePosition;

然后为了使相机跟随旋转，还要调整相机的yaw角

camera.yaw -= car.angle - preAngle;

12.模型的摆放

为了达到更加简洁的效果，在model类中可以设置一个objnum变量，一个model类可以含有相同模型的多个不同实例，主要是存放多个model矩阵，然后通过render进行渲染的时候就可以根据objnum和多个model矩阵进行渲染，这样可以避免同一个模型的多次加载。

至于位置的摆放的话，影响的音素比较多，和模型本身制作时的大小，位置都有关，可以通过对model矩阵进行scale，translate，rotate等变换进行摆放

## 遇到的问题和解决方案

1.天空盒中遇到的问题就是设置天空盒大小的问题，如果设置的太小的话会阻挡住盒子里加载的模型，但是我找的一个地形的模型中自带了一个天空盒，但是这个天空盒却没有那个无法靠近的那个效果，所以如果一直往边界走会越过这个模型自带的天空盒，所以还是要我们自己来实现天空盒的效果的，所以要想办法把那个模型中自带的天空盒给去掉。

2.模型加载的时候遇到的问题就是我找了很多的模型，在加载的时候很多都或多或少有一些问题，像有些3d模型的格式assimp库是无法加载的，有些的纹理路径等又不对，所以有的时候还需要自己打开mtl文件来修改一下路径之类的。并且关于纹理部分的话，有些模型又是没有纹理的，是通过直接用颜色信息的，所以在shader中还要考虑清有纹理和没纹理的情况来考虑。并且关于纹理的那个bindTexture也要十分谨慎，否则在着色器中通过sampler获取纹理的颜色信息的时候就很容易出错。

3. 本来想做一个换车的功能，但是不知道哪里出了问题，在访问纹理的时候会出现冲突，调试了很久都无法解决

## 小组成员分工

|  |  |  |  |
| --- | --- | --- | --- |
| 姓名 | 学号 | 完成功能 | 贡献比例 |
| 张寅哲 | 16340295 | 粒子系统的实现，阴影中深度贴图的调试，抗锯齿的实现，小组展示PPT的制作 | 25% |
| 郑国林 | 16340304 | 天空盒，白天黑夜变换，文字渲染，模型加载的一部分，模型收集与渲染摆放，阴影的调试，车的运动和照相机的跟随 | 25% |
| 刘俊峰 | 16340150 | 游戏架构设计；游戏基础部分实现：模型加载、光照、阴影  游戏物体控制：汽车控制；碰撞检测 | 25% |
| 谢玮鸿 | 16340256 | 水面波纹的模拟与实现 | 25% |