浙江大学

2017 图形学大程

Project 题目：基于 WebGL 的 3D 跳伞与神庙逃亡游戏

组长学号：3150103626　　组长姓名：谢文韬

组员学号：3150103644　　组员姓名：邵瑞辰

组员学号：3150104927　　组员姓名：徐畅

组员学号：3150104890　　组员姓名：张耀心

组员学号：3150102163　　组员姓名：吴涛

指导老师：张宏鑫

助教老师：李传康

目录

一、总体概述

1.游戏背景

绝地求生大逃杀是一款非常火爆的游戏，由于其对于真实战场的高度还原而广受追捧，而其中开局从初生岛到战场的跳伞过程也深入人心。而神庙逃亡是一款两三年前非常火爆的手游，其游戏场景也成为了玩家心中的经典。图形学大项目的作业是通过自己编写引擎实现一个完整的 CG 场景，因此把两款游戏中的部分相结合，来检测编写的底层引擎是否达到了构建整个 CG 场景，实现游戏逻辑的功能。

2.基本概况

本项目由小组共同讨论制定，采用 WebGL 实现，语言选用 javascript 为主，CSS 为辅助。构建基于引擎的完整 3D 游戏。玩家开始从空中坠落，落地后被怪物追逐后在道路上逃跑，通过吃金币进行加分，落入水中或踩到障碍游戏结束。

3.实现功能

（1）基本功能：

a. 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；

b.具有基于 OBJ 的三维网格导入功能

c. 具有基本材质、纹理的显示和编辑能力；

d. 具有基本几何变换功能（旋转、平移、缩放等）；

e. 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）；

f. 能通过玩家的奔跑和跳伞的方式在场景中漫游

g.实现了 render 连续播放功能

（2）扩展功能：

a.基于此引擎的完整游戏

b.基本 H5 平台和 WebGL 实现

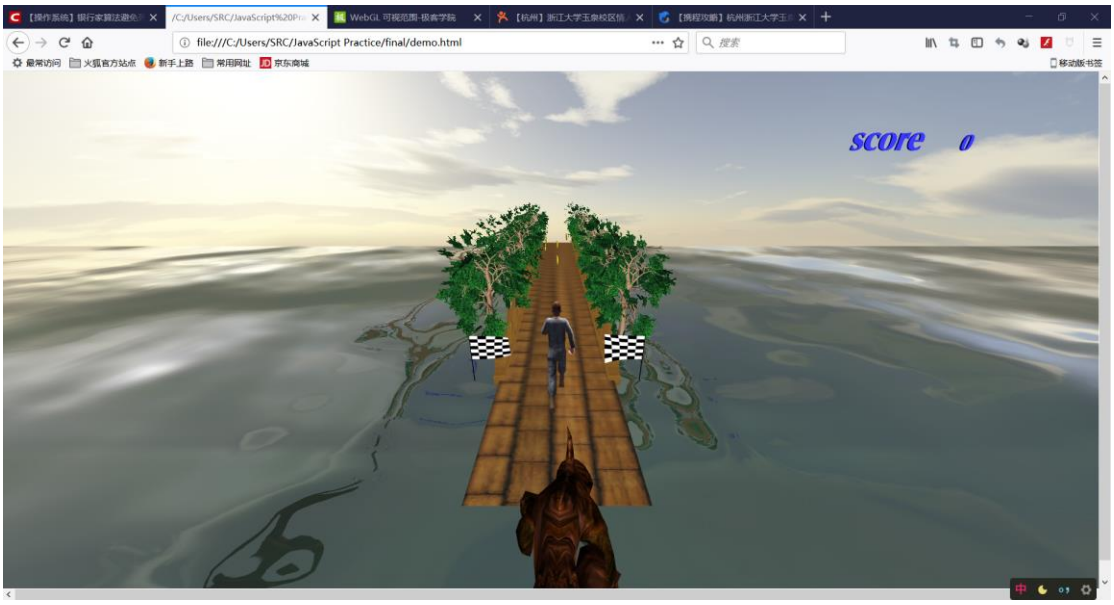
c.实现实时碰撞检测

d.具有 NURBS 曲面建模能力

e.人物和怪物骨骼动画（借助开源代码）

4.主 demo 的技术架构

Project 是基于 webgl 实现的跳伞+神庙逃亡游戏，其主要原理是构建一幅三维地图，初始由高空向地面坠落，在下降过程中可以调整人的体态，落地后即开始神庙逃亡游戏，总体的游戏画面如下：



即中间一条长度固定的道路，两边是随机生成的树木，被一座无限大的湖泊所包围，道路上会出现各种道具，比如金币、树桩、加速、护盾等等，玩家需要躲避障碍物，在各种功能道具的帮助下获取更多的金币，所得积分会实时显示在右上角。

为了实现这些功能，封装了大概十来个类，如 Road、Tree、Water、RoadSide、Coin 等等，主要的碰撞检测都在 Road 的 Update 方法中。

二、基本实现

1.3D 建模

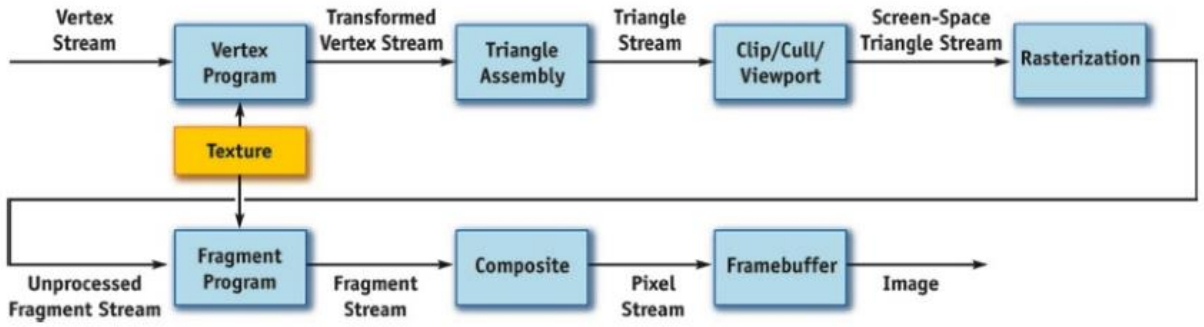
（1）渲染管线

着色程序分为两类：vertex shader program（顶点着色程序）和 fragment shader program（片断着色程序）。为了清楚的解释顶点着色和片断着色的含义，我们首先从阐述 GPU 上的两个组件：Programmable Vertex Processor（可编程顶点处理器，又称为顶点着色器）和 Programmable Fragment Processor（可编程片断处理器，又称为片断着色器）。

顶点和片段处理器被分离成可编程单元，可编程顶点处理器是一个硬件单元，可以运行顶点程序，而可编程片段处理器则是一个可以运行片段程序的单元。

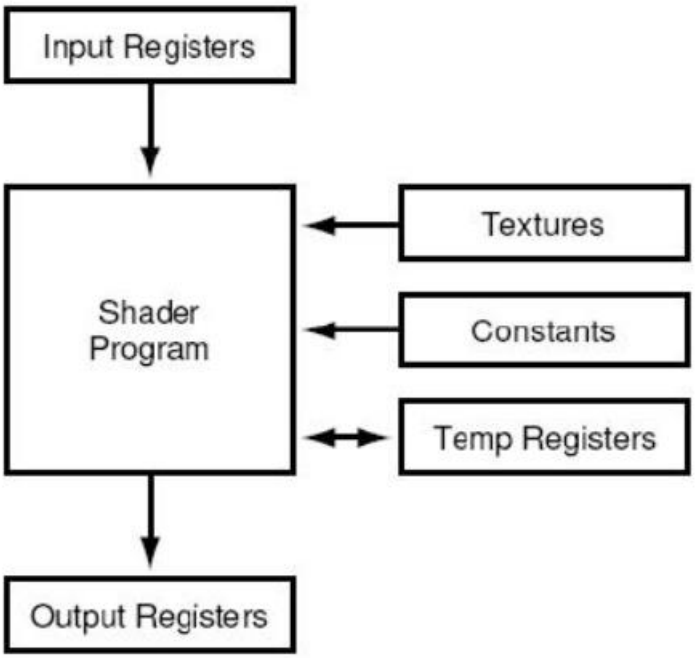
顶点和片段处理器都拥有非常强大的并行计算能力，并且非常擅长于矩阵（不高于 4 阶）计算，片段处理器还可以高速查询纹理信息（目前顶点处理器还不行，这是顶点处理器的一个发展方向）。

如上所述，顶点程序运行在顶点处理器上，片段程序运行在片段处理器上，哪么它们究竟控制了 GPU 渲染的哪个过程。下图展示了可编程图形渲染管线。



可以看出，顶点着色器控制顶点坐标转换过程；片段着色器控制像素颜色计算过程。这样就区分出顶点着色程序和片段着色程序的各自分工：Vertex program 负责顶点坐标变换；Fragment program 负责像素颜色计算；前者的输出是后者的输入。

下图展示了现阶段可编程图形硬件的输入/输出。输入寄存器存放输入的图元信息；输出寄存器存放处理后的图元信息；纹理 buffer 存放纹理数据，目前大多数的可编程图形硬件只支持片段处理器处理纹理；从外部宿主程序输入的常量放在常量寄存器中；临时寄存器存放着色程序在执行过程中产生的临时数据。



（2）顶点着色器和片段着色器

Vertex shader program（顶点着色程序）和 Fragment shader program（片断着色程序）分别被 Programmable Vertex Processor（可编程顶点处理器）和 Programmable Fragment Processo（可编程片断处理器）所执行。

顶点着色程序从 GPU 前端模块（寄存器）中提取图元信息（顶点位置、法向量、纹理坐标等），并完成顶点坐标空间转换、法向量空间转换、光照计算等操作，最后将计算好的数据传送到指定寄存器中；然后片断着色程序从中获取需要的数据，通常为“纹理坐标、光照信息等”，并根据这些信息以及从应用程序传递的纹理信息（如果有的话）进行每个片断的颜色计算，最后将处理后的数据送光栅操作模块。

（3）三维网格的导入：

a.基本原理

OBJ 格式的文件由若干个部分组成，包括顶点坐标部分、表面定义部分与材质定义部分等。每个部分定义了多个顶点、法线、表面等。

b.基本结构与关键字

OBJ 文件不需要任何种文件头(File Header)，尽管经常使用几行文件信息的注释作为文件的开头。OBJ 文件由一行行文本组成，注释行以符号“#”为开头，空格和空行可以随意加到文件中以增加文件的可读性。有字的行都由一两个标记字母也就是关键字(Keyword)开头，关键字可以说明这一行是什么样的数据。多行可以逻辑地连接在一起表示一行，方法是在每一行最后添加一个连接符()。注意连接符()后面不能出现空格或 Tab 格，否则将导致文件出错。

下列关键字可以在 OBJ 文件使用。在这个列表中, 关键字根据数据类型排列，每个关键字有一段简短描述。

顶点数据(Vertex data):

- v 几何体顶点(Geometric vertices)
- vt 贴图坐标点(Texture vertices)
- vn 顶点法线(Vertex normals)
- vp 参数空格顶点 (Parameter space vertices)

自由形态曲线(Free-form curve)/表面属性(surface attributes):

- deg 度(Degree)
- bmat 基础矩阵(Basis matrix)
- step 步尺寸(Step size)
- cstype 曲线或表面类型 (Curve or surface type)

元素(Elements):

- p 点(Point)
- l 线(Line)
- f 面(Face)
- curv 曲线(Curve)
- curv2 2D 曲线(2D curve)
- surf 表面(Surface)

自由形态曲线(Free-form curve)/表面主体陈述(surface body statements):

- parm 参数值(Parameter values)
- trim 外部修剪循环(Outer trimming loop)

- hole 内部整修循环(Inner trimming loop)
- scrv 特殊曲线(Special curve)
- sp 特殊的点(Special point)
- end 结束陈述(End statement)

自由形态表面之间的连接(Connectivity between free-form surfaces):

- con 连接 (Connect)

成组(Grouping):

- g 组名称(Group name)
- s 光滑组(Smoothing group)
- mg 合并组(Merging group)
- 对象名称(Object name)

显示(Display)/渲染属性(render attributes):

- bevel 导角插值(Bevel interpolation)
- c_interp 颜色插值(Color interpolation)
- d_interp 溶解插值(Dissolve interpolation)
- lod 细节层次(Level of detail)
- usemtl 材质名称(Material name)
- mtllib 材质库(Material library)
- shadow_obj 投射阴影(Shadow casting)
- trace_obj 光线跟踪(Ray tracing)
- ctech 曲线近似技术(Curve approximation technique)
- stech 表面近似技术 (Surface approximation technique)

c.OBJ 解析

解析 OBJ 和 MTL，则要用到 WebGL 与 HTML 中读取文件的函数，并在读取成功之后，在另一个函数中完成对 OBJ 文件的解析，而解析过程则要结合上文中对于 OBJ 文件和 MTL 文件原理的分析。具体过程如下：

```
    function readOBJFile(filename, gl, model, scale, reverse) {
        var request = new XMLHttpRequest();
        request.open("GET", filename, true);
        request.send();

        request.onreadystatechange = function () {
            if(request.readyState == 4 && request.status == 200){
                //获取到数据调用方法处理
```

```
                onReadOBJFile(request.responseText, filename, gl, model, scale,
reverse);
            }
        }
    }
```

我们要自己声明一个读取 obj 的函数，此处的函数名为 readOBJFile，之后会调用这个函数，如下：

```
//读取 OBJ 文件
        readOBJFile("resources/cube.obj", gl, model, 60, true);
```

首先来分析一下参数表。Filename 也就是 obj 文件的文件名，gl 的定义如下：

```
var gl = getWebGLContext(canvas);

        if(!gl){
            console.log("无法获取 WebGL 的上下文");
            return;
        }
```

可以看到，gl 是一个 WebGL 的上下文。Model 的定义如下：

```
//为顶点坐标、颜色和法向量准备空白缓冲区对象
        var model = initVertexBuffer(gl, program);
        if(!model){
            console.log("无法准备空白缓冲区");
            return;
        }
```

是一个空白缓冲区对象，

而剩下的 scale 是指导入的 obj 模型的规模，可以自己设置，是一个数。而 reverse 是一个布尔变量，如果是 true 的话则意为要翻转，否则不翻转。

代码中新建了一个 XMLHttpRequest 对象。该对象的作用是用于在后台和服务器之间进行交换。Onreadystatechange 是一个事件句柄，它的值（state_Change）是一个函数的名称，当 XMLHttpRequest 对象的状态发生改变时，会触发此函数。状态从 0 (uninitialized) 到 4 (complete) 进行变化。仅在状态为 4 时，我们才执行代码。而 status（HTTP 状态码）为 200 时表示请求已成功，请求所希望的响应头或数据体将随此响应返回。

因此如果 readystate 为 4 且 status 为 200 时，说明对于 obj 的读取已经成功。接下来要做的，是对 obj 文件的解析。

```
//obj 文件读取成功后开始解析
    function onReadOBJFile(fileString, fileName, gl, obj, scale, reverse) {
        var objDoc = new OBJDoc(fileName); // 创建一个 OBJDoc 对象
        var result = objDoc.parse(fileString, scale, reverse); //解析文件
```

```
        if(!result){
            g_objDoc = null;
            g_drawingInfo = null;
            console.log("obj 文件解析错误");
            return;
        }else {
            //解析成功赋值给 g_objDoc
            g_objDoc = objDoc;
        }
    }
```

代码中首先创建了一个 OBJDoc 对象，并使用这个对象的 parse 函数来完成对 obj 文件的解析。也就是说，真正的对于 OBJ 文件的解析，并不在之前的 readOBJFile 函数中，也不在这个 onReadOBJFile 函数中，而在 OBJDoc 类以及该类的 parse 方法中。由于整个 OBJDoc 的代码规模较大，因此不在文档中全部列出，我们重点叙述 OBJDoc 类的构造方法和 parse 函数。

```
// OBJDoc object
    // Constructor
    var OBJDoc = function(fileName) {
        this.fileName = fileName;
        this.mtls = new Array(0);       // Initialize the property for MTL
        this.objects = new Array(0);    // Initialize the property for Object
        this.vertices = new Array(0);   // Initialize the property for Vertex
        this.normals = new Array(0);    // Initialize the property for Normal
    }
```

这是 OBJDoc 类的构造函数，可以看到里面有五个属性。为了解释这五个属性的意义，我们制作了以下表格：

filename	准备读取的 obj 文件的文件名
mtls	数组，用于保存之后解析出来的 obj 文件的材质
objects	数组，用于保存之后解析出来的 obj 文件的 object 数据
vertices	数组，用于保存之后解析出来的 obj 文件的模型的顶点坐标数据
normal	数组，用于保存之后解析出来的 obj 文件的顶点法线数据。

之后我们来看 parse 函数。switch 部分是解析的核心。对于 obj 文件中不同的关键字，会有不同的操作。为了直观的体现不同关键字进行了怎样的操作，以下将用表格来阐述：

关键词	作用和操作
#	注释，无特殊操作
mtllib	读取 mtl 文件，读取材质，并将材质放到

	mtls 数组中，该数组为 OBJDoc 的成员变量。该部分的详细过程将会在本章节的后半部分讲到。
o	意义同'g'
g	读取 object 的名称，并放入 objects 数组中，该数组为 OBJDoc 的成员变量。
v	读取顶点数据，并放入 vertices 数组中，该数组为 OBJDoc 的成员变量。
vn	读取法线数据，并放入 normals 数组中，该数组为 OBJDoc 的成员变量。
usemtl	读取当前使用的材质的名称，并保存到局部变量中，配合下面的读取面的操作。
f	读取面，并通过 parseFace 配合当前使用的材质来给当前的 obj 文件加入面。

该函数的返回值最终为 true，完成对于 obj 文件的解析和调用。

2.材质纹理

（1）MTL 文件基本结构和关键字

.mtl 文件（Material Library File）是材质库文件，描述的是物体的材质信息，ASCII 存储，任何文本编辑器可以将其打开和编辑。一个.mtl 文件可以包含一个或多个材质定义，对于每个材质都有其颜色，纹理和反射贴图的描述，应用于物体的表面和顶点。

以下是一个材质库文件的基本结构：

```
1. newmtl mymtl_1
2. # 材质颜色光照定义
3. # 纹理贴图定义
4. # 反射贴图定义
```

每个材质库可含多个材质定义，每个材质都有一个材质名。用 newmtl mtlName 来定义一个材质。对于每个材质，可定义它的颜色光照纹理反射等描述特征。主要的定义格式如下文所示：

a.环境反射有以下三种描述格式，三者是互斥的，不能同时使用。

Ka r g b ——用 RGB 颜色值来表示，g 和 b 两参数是可选的，如果只指定了 r 的值，则 g 和 b 的值都等于 r 的值。三个参数一般取值范围为 0.0~1.0，在此范围外的值则相应的增加或减少反射率;

Ka spectral file.rfl factor ——用一个 rfl 文件来表示。factor 是一个可选参数，表示.rfl 文件中值的乘数，默认为 1.0;

Ka xyz x y z ——用 CIEXYZ 值来表示，x，y，z 是 CIEXYZ 颜色空间的各分量值。y 和 z 两参数是可选的，如果只指定了 x 的值，则 y 和 z 的值都等于 r 的值。三个参数一般取值范围为 0~1。

b.漫反射描述的三种格式：

- Kd r g b
- Kd spectral file.rfl factor
- Kd xyz x y z

c.镜反射描述的三种格式:

- Ks r g b
- Ks spectral file.rfl factor
- Ks xyz x y z

d.滤光透射率描述的三种格式：

- Tf r g b
- Tf spectral file.rfl factor
- Tf xyz x y z

e.光照模型描述格式：

- illum illum_#

指定材质的光照模型。illum 后面可接 0~10 范围内的数字参数。

（2）MTL 文件读取

MTL 的读取与操作从代码结构上来看与 OBJ 的读取与操作十分类似，但是略有不同。

MTL 并没有 readMTL 函数，但是有 onReadMTLFile 函数。调用这个函数的时机是在上一节的对 OBJDoc.parse 函数的 switch 语句中，对于‘mtllib’的处理。

```
case 'mtllib':     // Read Material chunk
                    var path = this.parseMtllib(sp, this.fileName);
                    var mtl = new MTLDoc();   // Create MTL instance
                    this.mtls.push(mtl);
                    var request = new XMLHttpRequest();
                    request.onreadystatechange = function() {
                        if (request.readyState == 4) {
                            if (request.status != 404) {
                                onReadMTLFile(request.responseText, mtl);
                            }else{
                                mtl.complete = true;
                            }
                        }
```

```
                    }
                    request.open('GET', path, true);  // Create a request to
acquire the file
                    request.send();                   // Send the request
                    continue; // Go to the next line
```

我们按照代码的逻辑逐一进行分析。

首先调用 parseMtllib 函数。该函数用于得到 mtl 文件的路径。

```
OBJDoc.prototype.parseMtllib = function(sp, fileName) {
        // Get directory path
        var i = fileName.lastIndexOf("/");
        var dirPath = "";
        if(i > 0) dirPath = fileName.substr(0, i+1);

        return dirPath + sp.getWord();   // Get path
    }
```

之后构造了一个 MTLDoc 的对象。

```
var MTLDoc = function() {
        this.complete = false; // MTL is configured correctly
        this.materials = new Array(0);
    }

    MTLDoc.prototype.parseNewmtl = function(sp) {
        return sp.getWord();         // Get name
    }

    MTLDoc.prototype.parseRGB = function(sp, name) {
        var r = sp.getFloat();
        var g = sp.getFloat();
        var b = sp.getFloat();
        return (new Material(name, r, g, b, 1));
    }
```

对象的中的内容我们之后会用到。我们现在继续分析对于‘mtllib’的处理。

之后便是对该 mtl 文件的访问。如果访问成功，则调用 onReadMTLFile 函数。

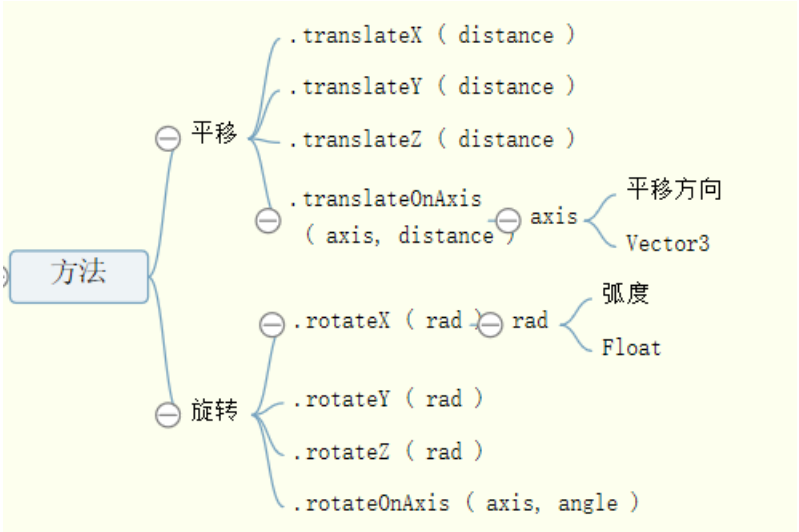
该函数的作用是对 mtl 文件的解析，同样我们用表格来展示该函数的内容。

关键词	作用和操作
#	注释，无特殊操作

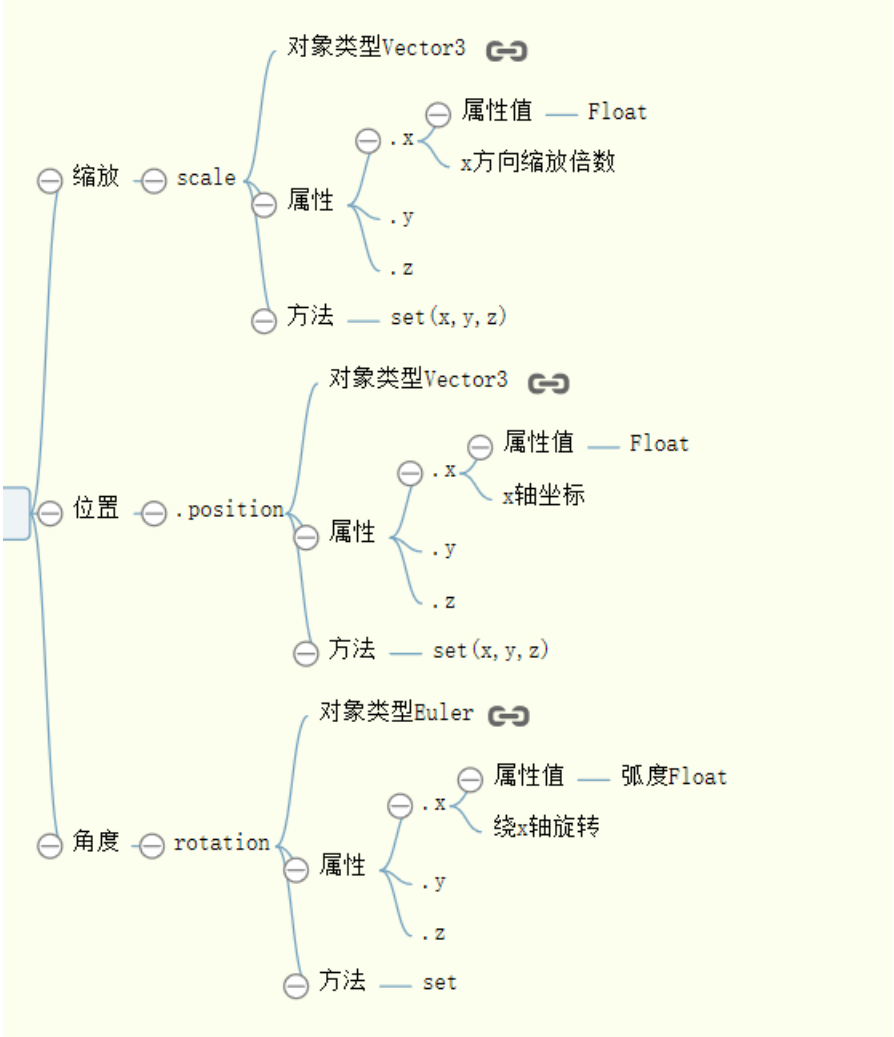
newmtl	mtl 为传进来的参数，是先前 new 的 MTLDoc 对象。此处调用了 parseNewmtl 函数，该函数最终获得被使用的 mtl 的名称。
Kd	该函数用于读取法线，并将 material 保存到成员变量 materials 数组中。一个 material 由名称和 r,g,b 三种颜色组成。

3.几何变换

Webgl 中的几何变换本质上和 OpenGL 没有区别，都是进行一系列矩阵变换，通过之前提到的着色器去实现。主要实现方法如下图所示：



所有三维实体和三维网格的实质属性如下图：



4.光照和相机

（1）光源编辑

光照模型的实现主要是配合顶点着色器等实现的 shader，通过一系列计算获取需要渲染的颜色值。

顶点着色器：添加变量

```
/**用于光照计算的变量 a_normal、u_lightColor、u_lightDirection**/
'attribute vec4 a_normal;' +//法向量变量
'uniform vec3 u_lightColor;' + //uniform 声明平行光颜色变量
'uniform vec3 u_lightDirection;' + //平行光方向
'varying vec4 v_color;'+//varying 声明顶点颜色插值后变量
```

着色器语言数据类型的相关转化、构造、访问相关问题，vec4、vec3 等都是和 float、int 一样是数据类型的标识关键字，float、int 在 C 语言中都是常见的类型， 着色器语言为了实现大规模的顶点运算增加了很多数据类型，vec4()、vec3()这时候的表达相当于一个 vec4、vec3 数据的构造函数，第 49 行代码中把一个 vec3 类型数据和一个 vec4 类型数据的一个分量 a 作为构造函数 vec4 的两个参数，来实现创建一个 vec4 类型数据。访问多元素数据的分量可使用点符号，从面对象的角度来看， 一个数据家是一个对象，数据的一个元素就是数据的一个分量，比如 a_color.a 表示 vec4 类型数据 a_color 的透明度分量 a。

```
42  // 顶点法向量归一化
43  '  vec3 normal = normalize(a_normal.xyz);' +
44  // 计算平行光方向向量和顶点法向量的点积
45  '  float dot = max(dot(u_lightDirection, normal), 0.0);' +
46  // 计算平行光方向向量和顶点法向量的点积
47  '  vec3 reflectedLight = u_lightColor * a_color.rgb * dot;' +
48  //颜色插值计算
49  '  v_color = vec4(reflectedLight, a_color.a);'
```

要想给着色器程序中声明的变量传递数据，首先要获取数据的地址，然后通过指针地址传递给变量。要想获取变量地址，不可能像普通 CPU 变成一样，要考虑 GPU 的特殊性，首先要通过 59 行代码调用初始化着色器函数，把着色器程序通过 CPU 与 GPU 的通信传递给 GPU 配置渲染管线， 执行执行初始化着色器函数的同时会返回一个 program 对象，通过对象 program 可以获取着色器程序中的变量，getAttribLocation()方法用来获取 attribute 关键字声明的定点变量地址，getAttribLocation()方法获取 uniform 关键字声明的统一变量地址。

```
60  /**
61   * 从 program 对象获取相关的变量
62   * attribute 变量声明的方法使用 getAttribLocation()方法
63   * uniform 变量声明的方法使用 getAttribLocation()方法
64   **/
65  var aposLocation = gl.getAttribLocation(program,'apos');
66  var a_color = gl.getAttribLocation(program,'a_color');
67  var a_normal = gl.getAttribLocation(program,'a_normal');
68  var u_lightColor = gl.getUniformLocation(program,'u_lightColor');
69  var u_lightDirection = gl.getUniformLocation(program,'u_lightDirection');
```

（2）相机编辑

在本项目中，相机是必不可少的部分，主要实现在于我们结合玩家视角的特色相机编辑方式，简单的相机视口原理就不再赘述。

三、特色设计

1.NURBS 曲面建模

（1）原理简述

NURBS 是非均匀有理 B 样条曲面（Non-Uniform Rational B-Splines）的缩写，是一种使用插值方法实现的曲面建模方法。Non-Uniform（非均匀性）是指一个控制顶点的影响力的范围能够改变，Rational（有理）是指每个 NURBS 物体都可以用有理多项式形式表达式来定义，B-Spline（B 样条）是指用路线来构建一条曲面，在一个或更多的点之间以内插值替换。

核心参数主要有如下：

a.多项式方程最大指数 degree
b.控制曲面形状的 control point
c.由前两个参数确定的 Knot vector
d.决定控制数目的 order

（2）具体实现

a.基本步骤：

计算基函数——计算基函数在控制点各处导数——计算 B 样条曲线

——计算 B 样条曲线在控制点的导数——计算有理曲线在各控制点处的的导数——计算有理 B 曲面的点

b.建模示例

考虑到我们要实现的是野外场景，而 NURBS 曲线只能用于生成光滑的曲面，无法用于生成棱角分明的石头、草等物体，我们使用 NURBS 曲线对神庙逃亡路上的标志旗、道路两旁的地面进行建模。

标志旗

初始的旗子控制点均在同一平面；改变旗子两个角位置上的控制点参数，营造出随风飘扬的效果。中间的曲面算法由 NURBS 曲面的插值衔接方式营造出一个自然光滑的曲面，使得其有真实感。

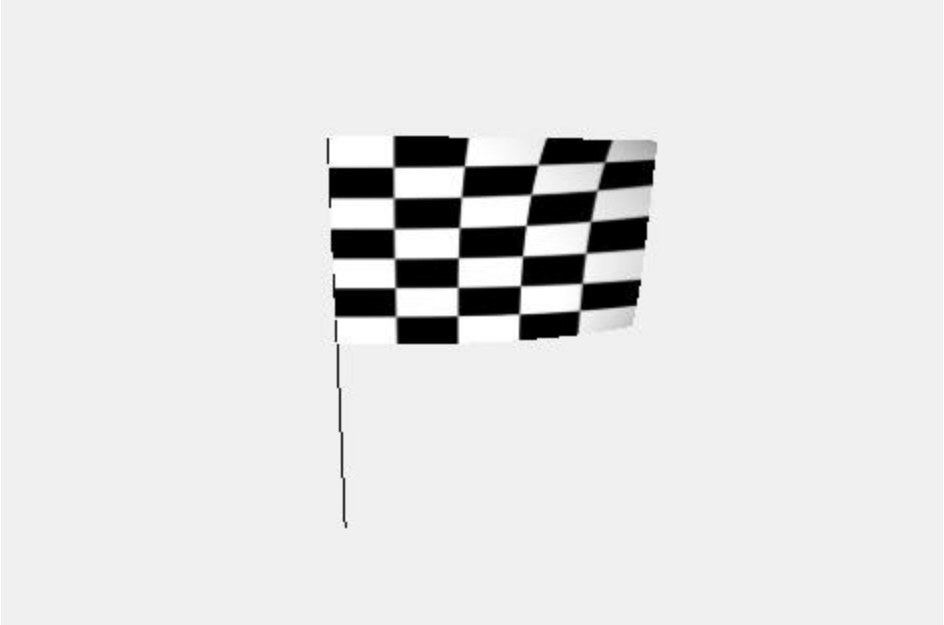


Figure 1 **标志旗效果图**

地形

NURBS 曲面运用在地形中，可以生成表面光滑的曲面，组合后可以形成斜坡、土堆等不同的地形效果。

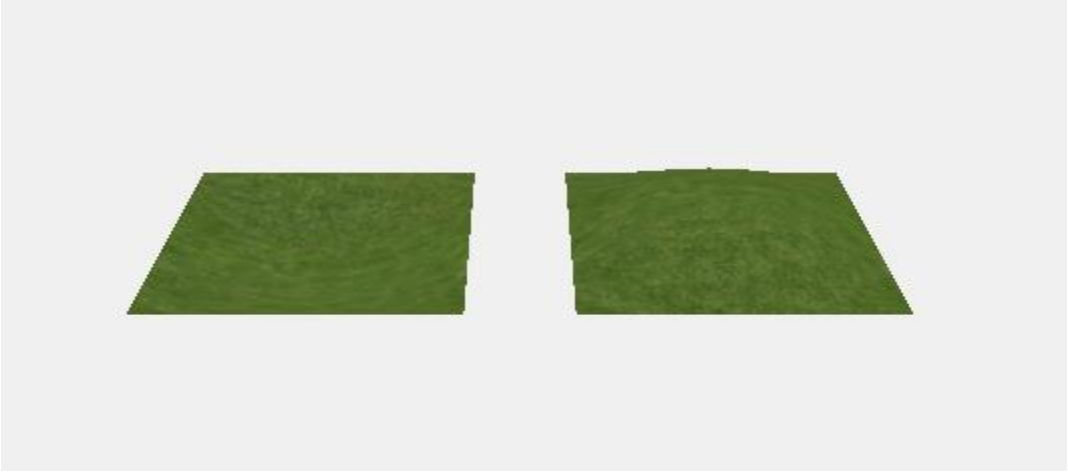


Figure 2 **地形效果图**

地形的接口标准等在地形建模部分进行说明。

（3）地形进阶

负责地形建模的同学对不同地形、地貌进行建模，给编辑地图的同学提供一个统一调用的接口：Terrain = function (x, y, z, size, terrain_type, texture_name, object_path, object_scale)。其中必需参数包括该地形的坐标、大小、地势类型、地貌类型，可选参数包括该地形上需要导入的物体（包括岩石、花草等物体），再加上设计地图的同学使用的

静态设计或动态随机算法，就可以生成最终需要的大地图。

实现过程包括生成地形、添加纹理、导入外部模型，其中生成地形是最主要的一步。生成地形的过程使用了 NURBS 曲面、二次曲面方程、随机数等不同算法，保证了地形的多样性。

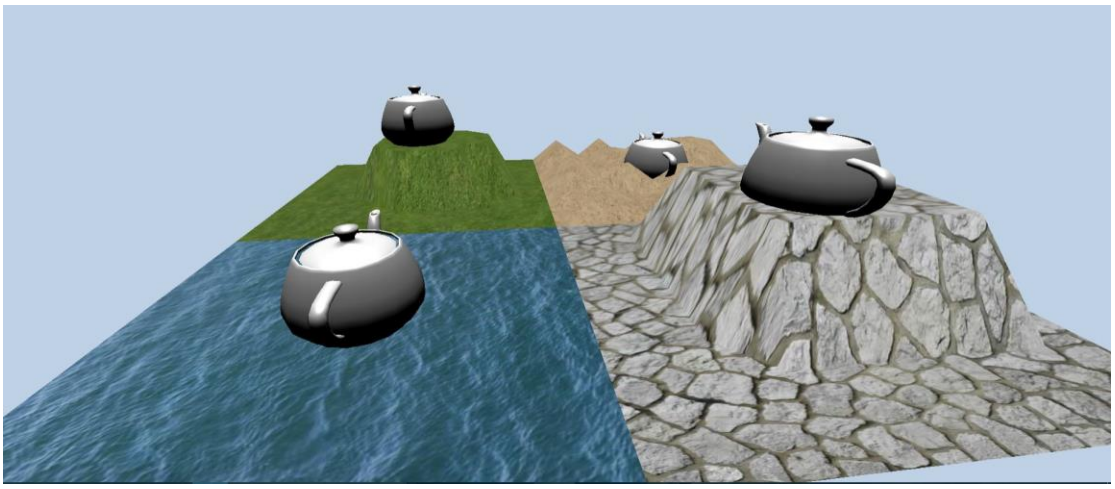
（4）地形生成 demo

草地对应圆台。

湖泊对应平地

沙漠对应随机生成的无规则地面

砖石地对应棱台地面



至此，我们可以根据需要或者随机生成不同的地貌和材质的地形。

2.实时 AABB 碰撞检测

由于这个游戏是有着很多道具的存在，因此碰撞检测尤为必要，人物模型跟不同的道具碰撞会产生不同的效果，吃金币会增加分数，吃加速会进入加速状态，吃护盾会产生护盾效果，而超出道路边缘入水中或者碰撞到树干都会导致游戏结束，考虑到人物一直是在正方向上移动(东南西北)，因此采用 AABB 包围盒即可，包围体是一个简单的几何空间，里面包含着复杂形状的物体。为物体添加包围体的目的是快速的进行碰撞检测或者进行精确的碰撞检测之前进行过滤（即当包围体碰撞，才进行精确碰撞检测和处理）。

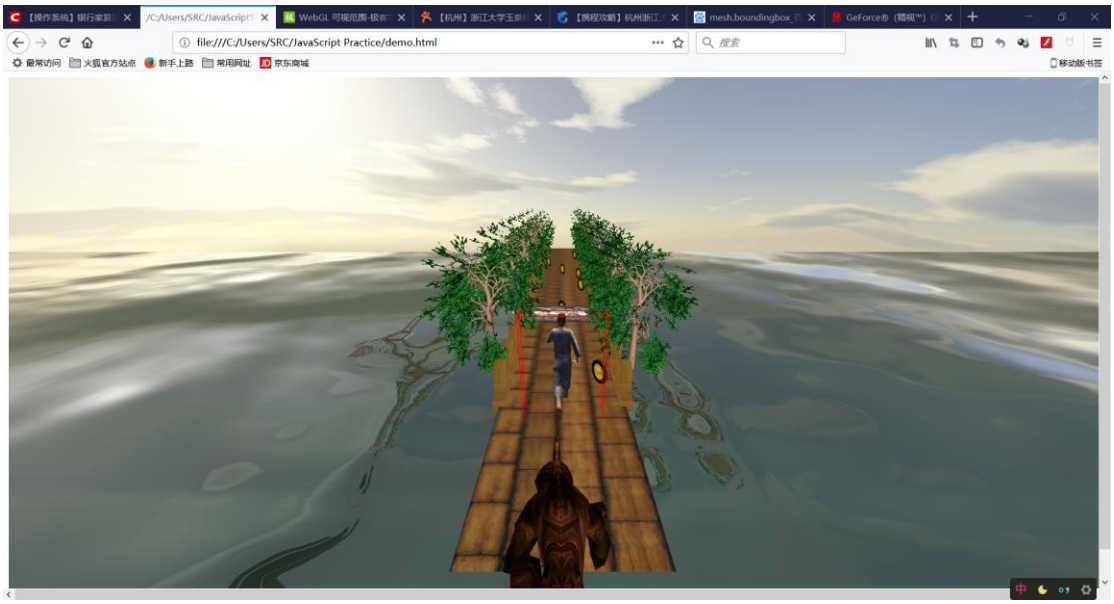
AABB 盒的表示方法有很多，总结起来有如下的三种情况：

（1）Max-min 表示法：使用一个右上角和左下角的点来唯一的定义一个包围体

（2）Center-radious 表示法：我们用 center 点来表示中点，radious 是一个数组，保存了包围盒在 x 方向，y 方向，z 方向上的半径。

（3）Min-Width 表示方法：我们用 min 来定义左下角的点，使用 width 来保存在 x，y，z 方向上的长度。

这里我们采用第二种方法，可以得到如下效果：

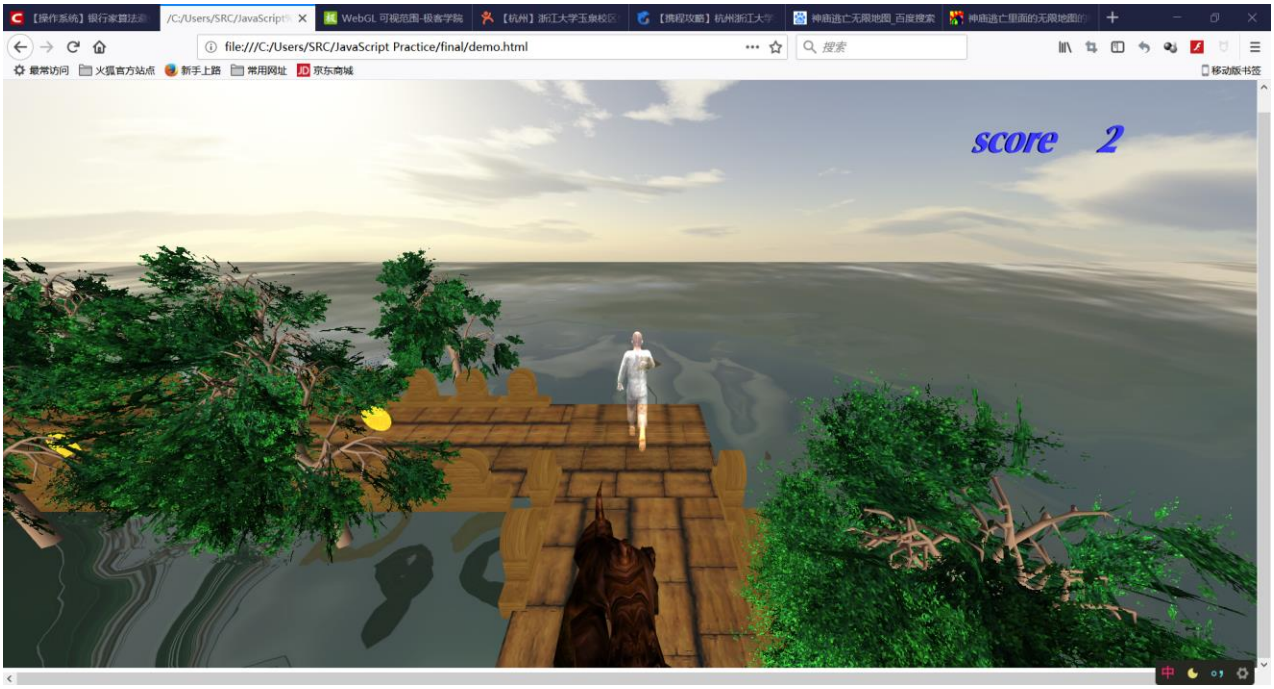


这样每一个需要碰撞检测的物体，我们都用 AABB 包围盒将其封装起来，只需要在一个函数里，依次判断 Item 是否与人物模型碰撞即可，一旦碰撞，再根据 Item 的 type 进行相应的操作。

3.无限地图方法

神庙逃亡这款游戏，最重要的特点就是，地图是无限延伸的，玩家永远走不到尽头，所以这就对于地图的构建造成了一定的困难，一个简单的思路是，使地图闭合，这样玩家就会永远循环地图，但是这会对游戏性造成一定的影响，所以这里我们采用了一种相对高级的方法。

首先是生成不同的道路模型，形成简单的 prefab，组装出道路，并与当前的道路拼接，如下图所示：



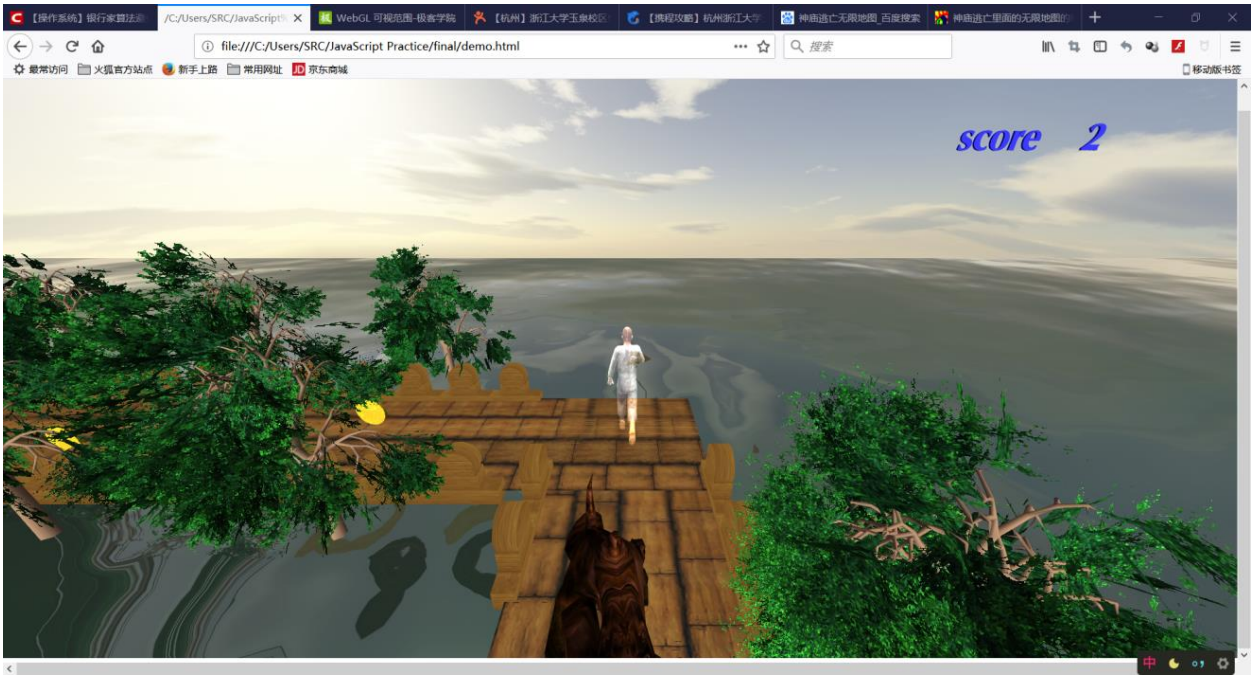
组装道路可以衍生出一个三叉树，进而随机生成一个闭环的无限道路

4 可视 FPS 优化

由于整个场景需要渲染的东西很多，所以如果不对 FPS 进行优化，就会出现画面卡顿的现象，因此采用了两种方法实现优化。

（1）删除看不到的场景元素

由于人是不断的向前方奔跑，所以路上的景观(金币、树木等等)在移出照相机视野后就直接删除，避免继续不必要的渲染耗时，基本原理就是如果 Item 的 position 位于相机的 position 后方，即视野盲区，那么就删除 Item，而对于道路的删除，因为在拐角处还是能够看到原来的道路，所以依然按照上面的方法删除是有损用户体验的，因此，在 Road 中引入 hasChildren 这一属性，它代表人走到道路中心时，会生成下一条道路，而这是删除最开始的道路就不会影响玩家的体验，因为它已经完全不在视野范围内了。



（2）网格优化

在 webgl 中，一切皆是 mesh，即网格，无论是道路、树木还是人物模型，全部是由一个一个的小三角形拼接而成，而小三角形的数目就是影响 render 性能的重要因素，因为在我们的神庙逃亡场景中，树木、人物模型相对复杂，需要相当大数量的三角形，因此，我们通过距离的远近来动态调节模型的精细程度，远处的模型，三角形面数较少，近处的模型三角形面数较多，这样并不会对用户体验造成多少影响，而且还大大加速了渲染过程。

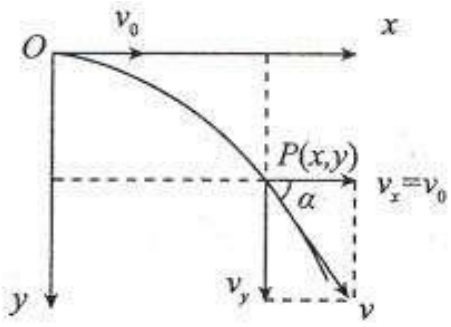
5.跳伞的物理系统建模

（1）坐标位移

跳伞是本游戏中的亮点之一，如果对于大逃杀中的场景做一些物理建模，可以得到如下的平抛运动图，其中算上正比于速度平方的空气阻力：

$$v_{xnew} = v_{xold} - kv_{xold}^2 cos\alpha * \Delta t$$
$$v_{ynew} = v_{yold} - \left(g - kv_{yold}^2 sin\alpha\right)\Delta t$$

$$\Delta x = v_{xnew} * \Delta t$$
$$\Delta y = v_{ynew} * \Delta t$$



$$F_f \propto kv^2$$

通过动态更新 Vxnew 和 Vynew,xnew 和 ynew 就能记录这个过程中的人位置，由于计算了空气阻力，所以达到一定的高度会进入平衡状态，使得游戏体验更加真实。

（2）人的动作

在大逃杀游戏中，可以看到人的跳伞动作会随着身体和水平方向的夹角做出不同的手臂动作，我们通过设置人的手臂和驱赶的夹角模拟这一过程：

大逃杀中：

本游戏中：

6.天空盒和水面

（1）天空盒

构造一个 10*10*10 的立方体外壳，在其表面上进行贴图，可以发现它的大小并不很大，那是因为我们在实践中发现，图形学是有限世界的表示，天空这种无限的概念如果用很大的盒子去表现，贴图的清晰度要求更高，更加消耗资源，而如果我们使用一个小的天空盒，通过天空盒和人一起运动的方法，人的视角里，天边永远一样远，就可以说骗过了人的眼睛。

（2）水面

水面是一个复杂的三维网格，通过加入镜面反射属性可以造成一些漂亮的倒影效果，包括树木。而水面大小我们设置正好延伸到天边，可以达到水天相接的效果，而且水天一色，很有感觉。

7.骨骼动画的表现

人和怪物的动作通过一个 json 导入，json 本质上也是 obj，mtl 加上骨骼动画的信息的组合，这里我们调用了 three 的 jsonloader 去解析复杂的骨骼动画，以实现人物和怪物动作的生动性，包括跳伞过程中的人手，包括人的奔跑等等：

四、效果展示

1.初始界面，按键开始

2.跳伞过程

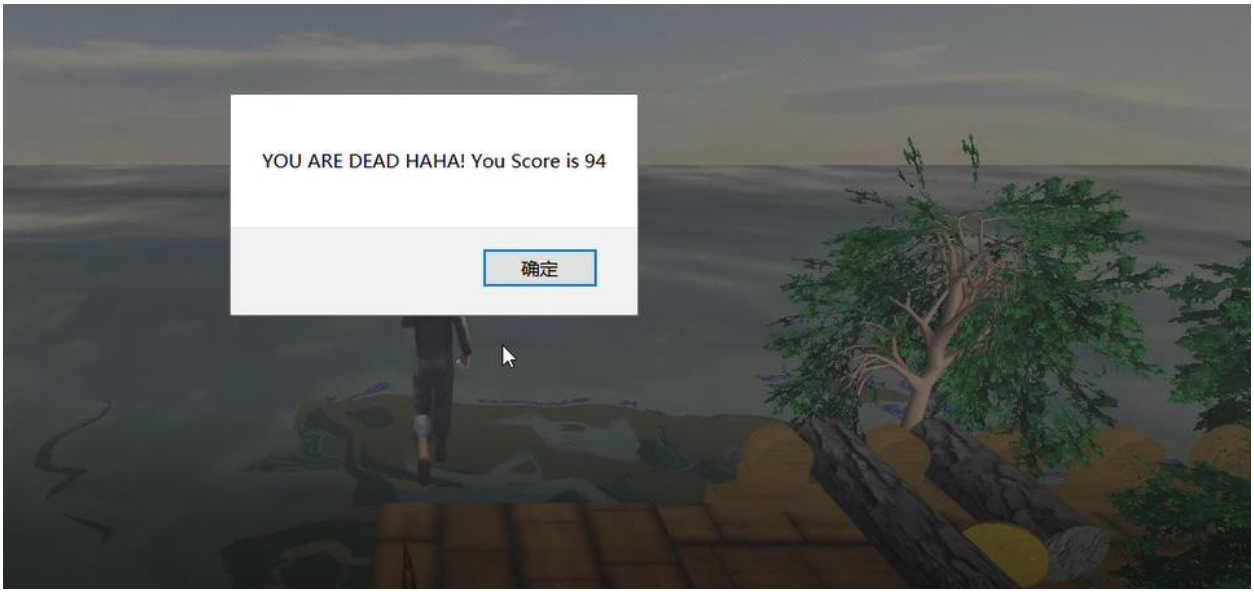

3.起点处的 NURBS 曲面旗

4.通过光照和透明组合实现的护盾效果和极速效果

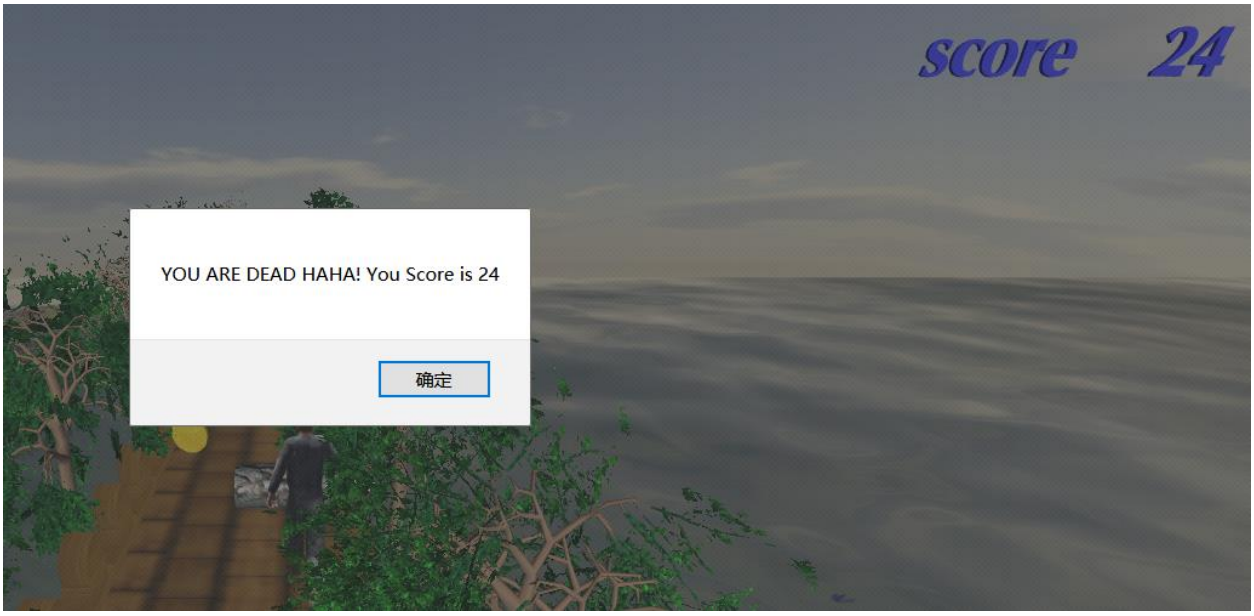

5.吃了金币之后自动加分

6.两种失败方法，撞树和掉河

掉河：



撞树：



7.随机路障、树木、金币和道具

8.其余效果可以参见 demo 视频。

五、工作量和分工

1.代码结构层次

主要展示：demo.html

引擎代码：script 文件夹中

2.代码工作量

Demo：1968 行

引擎 js：3320 行，有效行数约 2000 行

3.组员分工

（1）谢文韬：统筹整个工程，实时碰撞检测，游戏主逻辑优化，shader 和各类模型的实现，跳伞逻辑的实现

（2）邵瑞辰：工程整体逻辑实现，render 和 camera 的编写，光照模型的实现，水面和天空盒的设计

（3）徐畅、张耀心：NURBS 曲面建模，以及 OBJ，MTL 等三维网格的导入内容

（4）吴涛：本来被分配设计跳伞的相机变化，但是未能完成好任务，由谢文韬和邵瑞辰实现。

六、总结

通过本学期计算机图形学知识的学习和大程的实现，我们对于图形学的系统建模有了一定程度的了解，但是发现具体知识应用到实际的过程中还是有很大差距的，我们开始认为有了 glut 就可以制作想要的游戏，可以实现自己写游戏的梦想，以至于一开始指定计划的时候百度到了一些模型，就想着

可以自己实现一个绝地求生大逃杀，但是后来发现要考虑的问题还有很多，特别是自己编程而不是依赖现有的引擎。

系统的底层 GPU 的 shader 编程让我们对于底层的硬件有了一定的了解，但是通过 shader 去实现各种着色效果，包括光照，纹理的映射，发现要考虑的问题特别多。特别是对于我们做游戏而言，流畅不卡的游戏体验式最重要的，所以我们一直在试图去优化 FPS。最开始一整个地图全部导入，全部进行 render，游戏体验极差，通过讨论优化了方案，通过可视 FPS 的优化和 Mesh 的优化，我们的帧数有了明显的提高。

地面游戏的内容我们最初想的是绝地求生大逃杀，但是游戏逻辑上复杂太多，而且绘制场景很不容易。所以我们决定从一般的曲面做起，去实现那些草地、山坡的地形，而队友也很给力，通过课上学的 NURBS 建模方法实现了随机的曲面建模，可以做出山地、草地类似的效果，这样我们的场地可以像拼图一样一块块搭建起来，只要保证衔接的平滑，就能生成一个复杂的场景。

我们试图实现人在曲面上的行走，但是发现获取曲面的表面让人可以逼真地走有些困难，所以最后选择了平面行走。但是喜人的是，本来以为要做一个僵硬的第一人称视角，后来借助了引擎中的骨骼动画可以让人在第三人称动起来，所以很有意思。

老师和同学们在验收的时候提了很多有建设性的建议，比如一个同学指出我们拐弯的动作略显僵硬，由于时间有限难以在考试周来临之际去优化，之后我们是有方案去优化的。老师提出我们缺少一个实时计分的功能，我们通过 css 配合实现了，可以在 demo 中看到。

总结下来，我们的项目管理还是不够紧凑，中间方案修改了很多次，对接和拼接也做的不是很好，以至于最后几天负责拼代码的特别劳累。虽然有个别同学有些不上心，但是作为组长还是要找一下自己的原因，在统筹任务的时候可以更加有条理一些，催的更紧一些。不过自己通过努力达到了效果，也是非常有收获的。