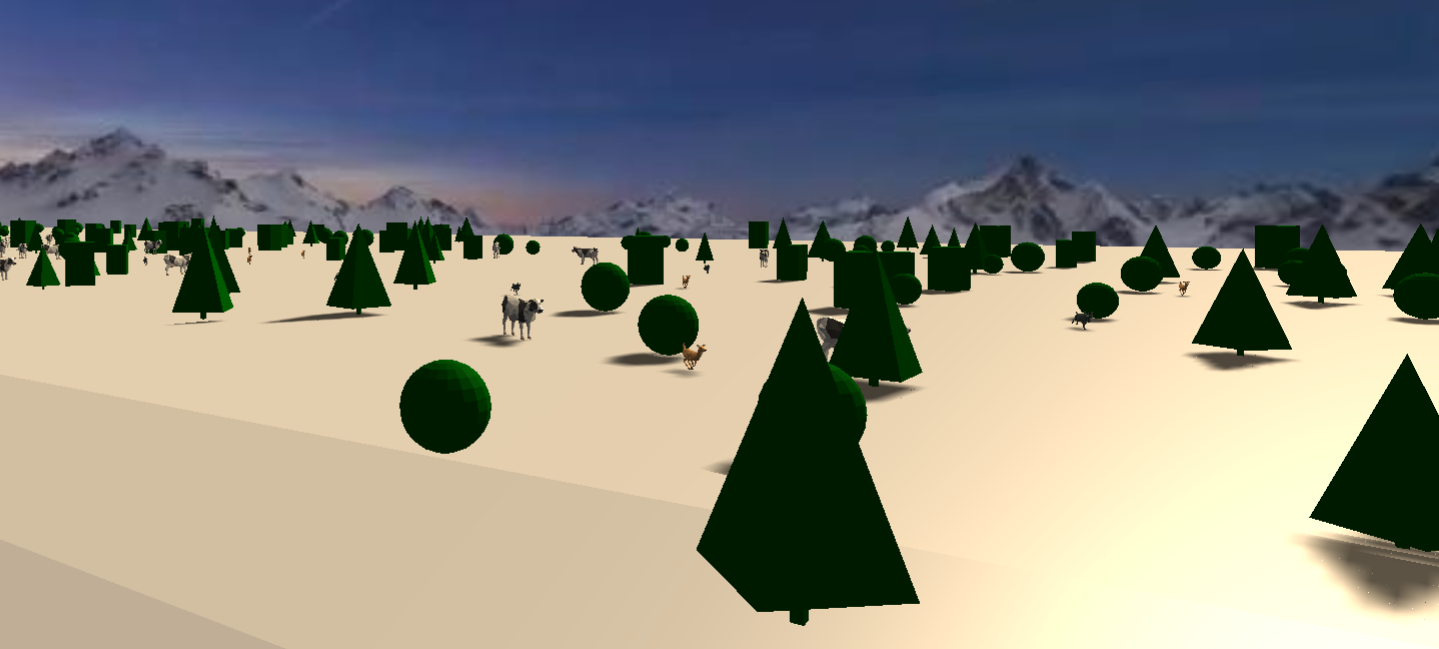
2018学年计算机图形学

**ComputerGraphics-Project**

**Music Parkour**（3D音乐跑酷）



**-powered by Web GL**

**组员信息**

|  |  |  |  |
| --- | --- | --- | --- |
| 成员信息 | 姓名 | 专业 | 学号 |
| 组长 | 张文龙 | 信息安全 | 3160102489 |
| 组员 | 邹家豪 | 信息安全 | 3160102313 |
| 组员 | 刘子威 | 信息安全 | 3160105715 |
| 组员 | 张思淡 | 信息安全 | 3160102245 |
| 组员 | 李天昊 | 信息安全 | 3160104528 |
| 2019.01.20 | | | |

目录

[1. 项目摘要： 2](#_Toc536494791)

[2. 项目详请： 3](#_Toc536494792)

[2.1 项目要素 3](#_Toc536494793)

[2.2 游戏逻辑 3](#_Toc536494794)

[2.3 操作方法 3](#_Toc536494795)

[2.4 项目工作 3](#_Toc536494796)

[3. 实现方法 4](#_Toc536494797)

[3.1 实现基本体素**建模**（长方体、球、圆柱、圆锥、多⾯棱柱、多⾯棱台） 4](#_Toc536494798)

[3.1.1树形的四棱锥体（带立方体的底） 4](#_Toc536494799)

[3.1.2长方体 4](#_Toc536494800)

[3.1.3球体 5](#_Toc536494801)

[3.1.4圆柱体 5](#_Toc536494802)

[3.1.5圆锥 6](#_Toc536494803)

[3.1.6圆台 6](#_Toc536494804)

[3.1.7 **有树枝的树** 6](#_Toc536494805)

[3.2 实现三维**模型**导入 8](#_Toc536494806)

[3.2.1 Json格式模型导入 8](#_Toc536494807)

[3.2.2 场景中导入模型的随机分布 8](#_Toc536494808)

[3.2.3 查看场景中模型导入效果 8](#_Toc536494809)

[3.3 添加并播放**音乐** 9](#_Toc536494810)

[3.3.1 按钮触发打开文件 9](#_Toc536494811)

[3.3.2音频解码与播放 10](#_Toc536494812)

[3.4障碍物—台阶的**建模** 11](#_Toc536494813)

[3.4.1高度数据采样 11](#_Toc536494814)

[3.4.2形状生成方式 13](#_Toc536494815)

[3.5 **贴图**效果简单实现 13](#_Toc536494816)

[3.5.1 构造球面包围盒并实现贴图 13](#_Toc536494817)

[3.6 **碰撞**检测 14](#_Toc536494818)

[3.7 **光照** 15](#_Toc536494819)

[3.8 初始界面与游戏**UI**设计 16](#_Toc536494820)

# 项目摘要：

本次计算机图形学的大作业中，我们小组实现了一个名叫“Music Parkour” 即“音乐跑酷”的音乐3D游戏（以下简称MP）。该游戏基于*WebGL*平台开发，所用到的编程语言有*HTML*，*Javascript*，*CSS*等。其中整个大项目程序以*Javascript*为主，所有的3D场景构建，大部分游戏逻辑设计以及音乐播放效果最终都由它来实现。

为发挥成员间的协作优势，我们小组就此项目在线下和线上都开展了较为活跃的讨论与交流。我们在Github也专门设立了的一个开源项目库：网址是*https://github.com/JorhoZou/ComputerGraphics-Project*，可供参考。

# 项目详请：

## 2.1 项目要素

在整个小组项目的设计与开发中，我们充分利用了《计算机图形学》课上掌握的知识要领，通过逐步搭建3D场景，最终构造出了一个较为完整的CG世界。该CG项目所涉及到的图形学基本要素大致可以罗列成以下几点：场景构造与漫游，三维建模与导入， 材质纹理与贴图，光照模型与阴影，几何变换与视角，碰撞检测与游戏，跨平台移植等，满足了图形学课程大项目中的所有基本要求和部分高级要求。这些要求会在后面的详细设计描述中一一体现。

## 2.2 游戏逻辑

为使我们的CG成品具备更好的现场交互性，我们在原本的3D图像场景表达基础上添加了另一维度的要素，即音乐。最终设计出来的音乐游戏，能给场景体验者营造出一种身临其境的效果。即，在场景中，体验者可以一边漫游3D场景一边听音乐（漫游的控制基于玩家对音乐节奏的把握）。

## 2.3 操作方法

用户用浏览器打开游戏WebGL的HTML本地（index.html）页面时，需单击页面上的“打开文件”按钮从本地打开指定的音乐（Track01.mp3）。随后单击OK，开始游戏。游戏会生成整个3D场景，随即播放音乐，同时游戏的主角会从场景的一头跑向另一头，途中有大量的台阶样障碍物。这些障碍物不是随机的，而是根据音乐的节奏手工建模而成。玩家需要把握实时播放的音乐节奏，踩点并敲击跳跃键（“8”），越过障碍物。由于游戏引入了碰撞检测，如果玩家的游戏节奏把握不当，主角可能会卡在某一较高的台阶侧面上，不能继续奔跑，宣告游戏结束。玩家需要通过多练习，才能顺利通关，获得良好的游戏体验。此外，游戏时，用户也可以自由地放大缩小，平移相机，转动方向等操作。 （如果无法利用“8”跳跃，首先需要按一下Shift键）

## 2.4 项目工作

在本次CG项目的开发中，我们小组做了以下工作：

（1）建立了的3D游戏对象可分为以下两部分：

第一部分，是天空地面，人物，花草树木等的建模，分为动态和非动态的。

第二部分，是根据背景音乐设置的跑道障碍物建模，时间轴要和音乐匹配。

并为场景添加了碰撞检测以及光照明，贴图效果。

（3）实现了模型（Json格式）的导入。

（4）实现了3D游戏场景中视角动态改动，以及场景漫游。

（5）可使程序相应来自键盘或者鼠标的不同操作。

（6）成功构建了HTML网页，能够控制游戏载入，也能播放音乐，网页分游戏未开始和开始两部分，且各自的效果不同。

# 实现方法

## 3.1 实现基本体素（长方体、球、圆柱、圆锥、多⾯棱柱、多⾯棱台）

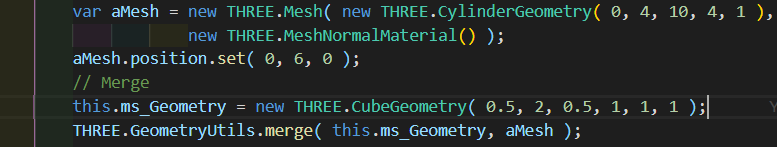
绘制树的形状首先从最简单的基本体素开始，最后构建更加复杂的模型

### 3.1.1树形的四棱锥体（带立方体的底）

效果如下：

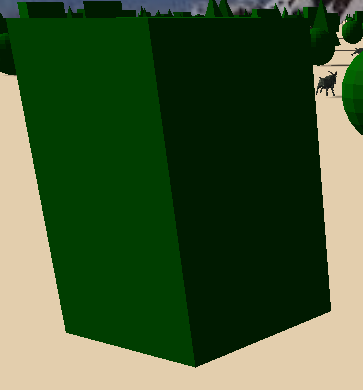
实现方法：



先生成一个四棱锥体，然后跟一个底的立方体合成。然后随机生成100个放在平面上。

### 3.1.2长方体：

之所以选择长方体而不是立方体是因为长方体在界面中会更好看。效果如下：

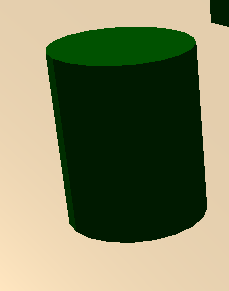


实现过程跟四棱锥体一样。

### 3.1.3球体：



### 3.1.4圆柱体：



### 3.1.5圆锥：

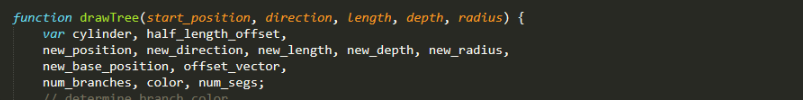


### 3.1.6圆台：



### 3.1.7 有树枝的树

之前实现的都是卡通风格的树木，这回尝试一个写实风格的树木。我们采取递归的方式，生成一颗有树枝的树。生成树木的***递归***定义如下：



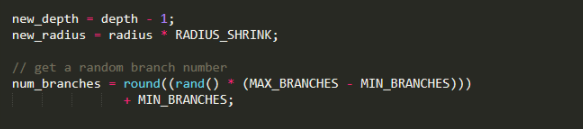
在位置start\_position处，沿着direction方向，生成一个长度为length，半径为radius的树枝，然后递归地生成以该树枝作为树干的树。depth表示当前的递归深度。

绘制一根树枝，就是简单地绘制一个cylinder。如何绘制生长在这根树枝上的其他树枝呢？

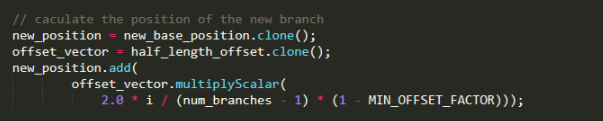
（1）我们随机地让3-4根子树枝生长在这根树枝上。

（2）子树枝的半径要比当前树枝小。

（3）子树枝的深度是当前的深度+1

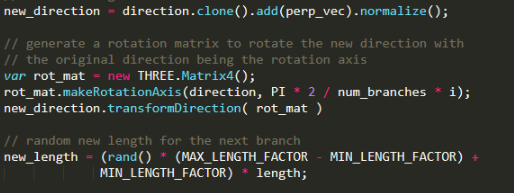


（4）我们在子树枝生长在这根树枝上的不同位置。

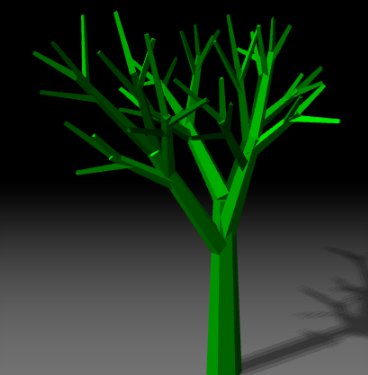


（5）子树枝的长度要比当前树枝小。

（6）子树枝的方向是在当前反向是旋转一定角度得到的。



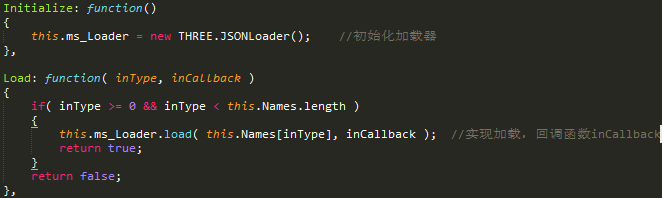
实现效果：



## 3.2 实现三维模型导入

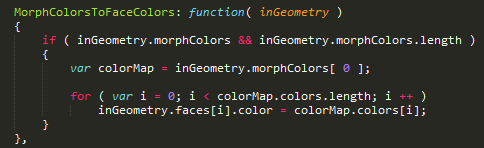
### 3.2.1 Json格式模型导入

Project中定义了一个名为“meshes.js”的类，用于使用three.js下的JSON加载器导入json格式模型。



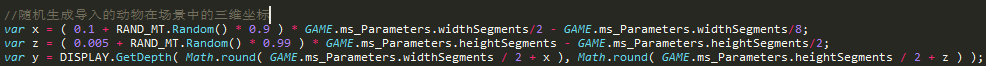
导入模型后根据Json模型特性进行相关贴图于渲染，相关函数有

AddMorph()、MorphColorsToFaceColors()等。



### 3.2.2 场景中导入模型的随机分布

导入的模型需要随机的分布在场景中，我们通过简单的random函数以及对三维坐标的范围限制实现上述效果，核心代码如下：



### 3.2.3 查看场景中模型导入效果



## 3.3 添加并播放音乐

此处添加的音乐是固定的，即文件夹中的Track01.MP3. 因为此工程中的BGM与3.4节中：障碍物——台阶的建模密切相关。

### 3.3.1 按钮触发打开文件

在游戏控制程序的主函数main内编写如下代码：

$("#loadfile")[0].onchange = function(){

        file = this.files[0];

        fr = new FileReader();

        alert(" Start Game"); //弹框，确认，中止后续程序执行

        fr.onload = function(e){

            mv.play(e.target.result);

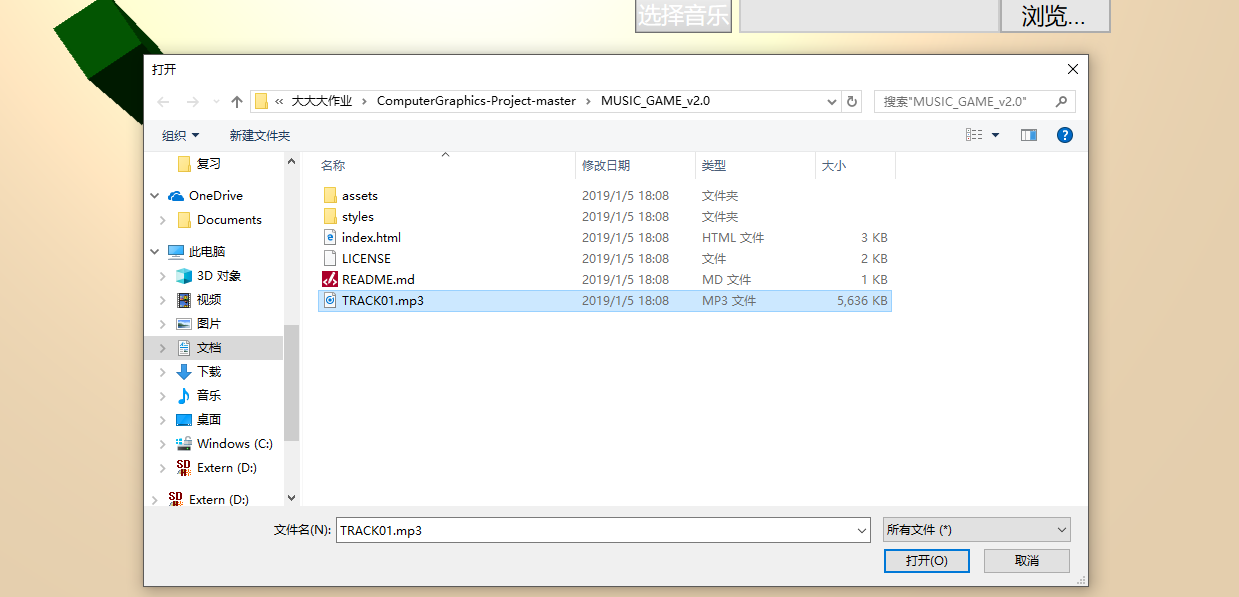
        }

        fr.readAsArrayBuffer(file);

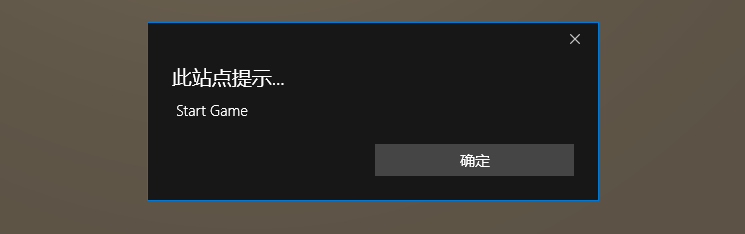
        MyStart(); //调用游戏初始化

    }

这段程序表示用户在HTML中单击“文件选择”按钮后，Javascript程序会触发loadfile，要求从本地文件夹中选择指定的音乐文件用于后续的播放。



Javascript的安全性策略导致，程序员编写程序的时候，程序不能主动地脱离浏览器到本地读取一个指定文件，这可能涉及到个人隐私。所以要用JS“打开文件”只能由用户手动打开。尽管HTML可以直接插入背景音乐，用户可以不用手动打开本地文件，在本CG项目中却没有采用是因为：这种方式下，在用户打开网页时，网页会直接播放音乐，开始游戏。用户没有准备时间，也就是缺少StartGame这个确定步骤：



在JS中可以轻松设定游戏开始按钮，并请求用户确认（同时中止后面所有程序运行）。只有用户单击“确定以后”游戏才会开始。这可以提升用户的游戏体验，并可以让用户在没有开始游戏前，自由的漫游整个场景。

### 3.3.2音频解码与播放

音频解码与播放的代码中以此调用以下函数

Musicvisualizer.prototype.load = function(url,fun){

由于是WebGL，基于网页的浏览，用户打开的本地音乐Track01被抽象成了URL资源，即JS把Track01和一个本地URL绑定起来。达到打开文件，获取读写流的效果

Musicvisualizer.prototype.decode = function(arraybuffer,fun){

以上这个函数可以用于实时地音频采样（把音频数据一个buf一个buf地解码），从中可以获得当前播放音乐的响度量化值，保存在arraybuffer中。

|  |  |
| --- | --- |
|  | for(var i=0;i<size;i++){  var rectHeight = arr[size-i]/768\*height;//音频数据最大值256  o.cap--;  timer++;  if(timer>=3000){  timer=0;  StageHeight=arr[50]/256+0.01\*RAND\_MT.Random()console.log(StageHeight+",");  }  实时的音乐响度信息可以从浏览器中自动输出来，  ***这可以辅助地面障碍物的建模***。因为这些数据是固定时间输出的，响度越大，当时的障碍物高度越高 |

Musicvisualizer.prototype.play = function(path){

以上函数是用于实时播放音乐的。

此外我们还要在浏览器HTML中预先调入包含上述代码的JS文件

        <script src="assets/js/musicdecoder.js"></script>

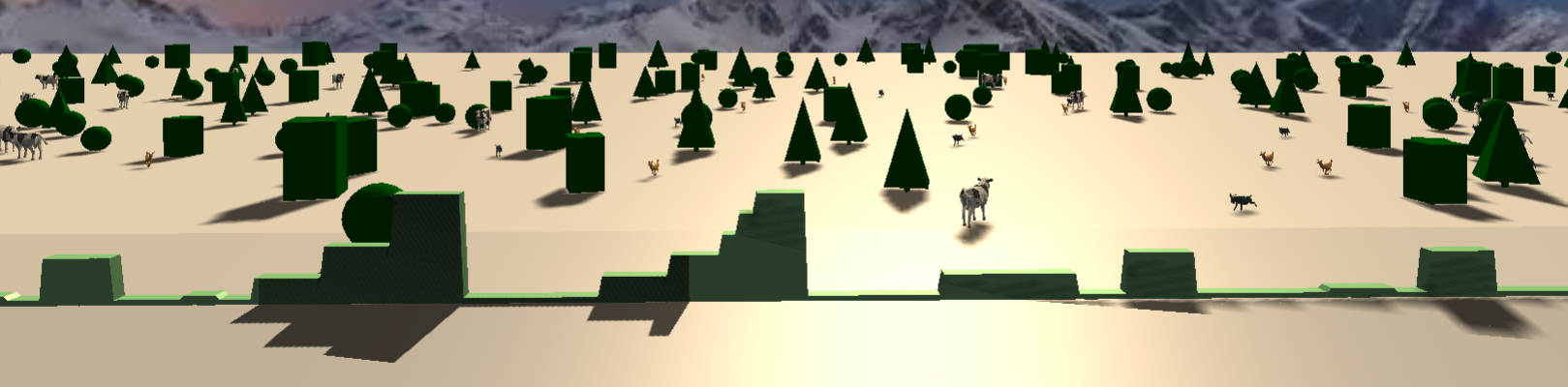
        <script src="assets/js/game/music.js"></script>

利用以上三步我们可以播放音乐了。

## 3.4障碍物——台阶的建模

### 3.4.1高度数据采样

这里我们利用了box2D实现了对整个台阶的建模：



        aGroundPoints.push( CONV.ToVector2D( { x: 3, y: -0.1 } ) );

        for( var i = 0; i <mystream.length; i += 1 )

        {

            //var Resampling = 0;

            //if(mystream[i]>0.2)Resampling=5;

            var x1 = j, x2 = j +0.255/2;

            y = (mystream[i]+0.1)\*2\*(1-0.001\*RAND\_MT.Random())-0.2;

            j+=0.33045/2;

            // var x1 = j, x2 = j +0.6;

            // y = mystream[i]\*3\*(1+0.1\*RAND\_MT.Random());

            // j+=0.7609;

     aEdgeShape.Set( new Box2D.b2Vec2( x1, 0 ), new Box2D.b2Vec2( x2, y ) );

GAME.ms\_B2DWorld.CreateBody( new Box2D.b2BodyDef() ).CreateFixture( aEdgeShape, 0.0 );

            aGroundPoints.push( CONV.ToVector2D( { x: x1, y: y } ) );

            aGroundPoints.push( CONV.ToVector2D( { x: x2, y: y } ) );

        }

其中mystream是一个手工调试而成的数组，他需要根据音乐的播放速度相匹配，

音乐播放到哪儿（时间t），主角跑到离起点的距离（s），这两个变量st是**一次函数**关系。mystream中的数据改变的是主角脚下的地面障碍物高度，数值越大，障碍物高度越大。



……数组保存在rhythms.js文件中

0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.0,

0.00,0.00,0.00,0.00,0.00,0.00,0.00,

0.00,0,0.00,

0.00,

0.00,

0.00, 0.00,

0.02． 0.02,

0.02,0.0,

0.0, 0.00,

0.13,0.13,0.13,0.13,0.00,0.00,0.00,0.00,0.0, 0.0,

0.0, 0.0, 0.0,

0,0, 0.02,

0.02,0.02,0.0,

0,0

0.13,0.13,0.13,0.13,

0,0,

0,0, 0, 0, 0, 0, 0, 0, 0,

0.02,

0.02,

0.02,

0.0,0,

0,

0.13,

0.13，0.13,0.13,

0,

0,

0, 0, 0, 0, 0, 0, 0, 0,

0,

0.02,

0.02,

0.02,

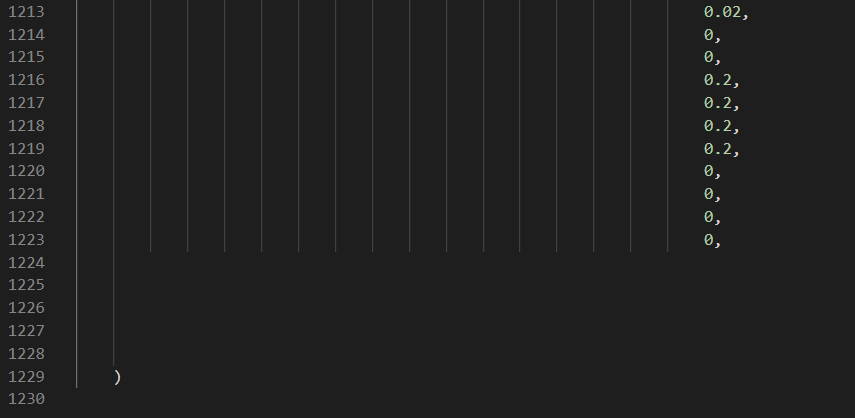
0.0,

0,

0.13, 0.13, 0.13, 0.13, 0.13,

//一下还有1000多个采样，

……



var x1 = j, x2 = j +0.255/2;

建模代码中的0.33045/2;代表每块生成砖块的长度。长度越短，意味着对音乐的采样率越高，主角跑过同样的距离s，需要生成的台阶数目就要越多。同时也越能够协调音乐节奏和障碍物出现的时机。唯一的坏处就是数据量会增大，采样细致程度（程序中设定的是0.33045/2;）与整首音乐的采样次数成**反比例函数关系。**

**例如：要生成如下障碍物台阶**

**我们需要采样大约12次，采样数据一样的会被拼接起来形成一个跟长的台阶。它的数据是：**

0.1,0.1,0.1,0.1,0.2,0.2,0.2,0.3,0.4,0.5,0.5,0.5,

**而紧跟在它后面的另一个障碍物台阶**

**则可以采用8个同样的数据即可：**

0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1,

### 3.4.2形状生成方式

var x1 = j, x2 = j +0.255/2;

y = (mystream[i]+0.1)\*2\*(1-0.001\*RAND\_MT.Random())-0.2;

j+=0.33045/2;

j表示当前地址离起始地址之间的距离，步长为0.33045/2，表示一个台阶的下底长度。

y = (mystream[i]+0.1)\*2\*(1-0.001\*RAND\_MT.Random())-0.2;

y表示台阶在实际场景中的高度。这样可以方便地对所有的台阶统一等比例放大缩小。其中乘以1-0.001\*RAND\_MT.Random()可以塑造凹凸不平地效果，使模型不显得单调。

var x1 = j, x2 = j +0.255/2;中的0.255/2是上底的长度。如果该数据变成0.33045/2，则会从梯形变成矩形。

aGroundPoints.push( CONV.ToVector2D( { x: x1, y: y } ) );

aGroundPoints.push( CONV.ToVector2D( { x: x2, y: y } ) );

最后将一个台阶的定点数据压入整个模型表达的数据栈中。

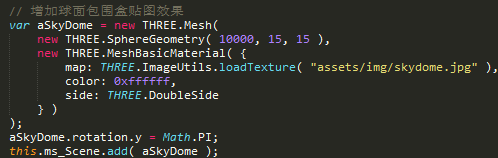
for( var i = 0; i <mystream.length; i += 1 )

值到采样完mystream，即i =mystream.length后才建模完毕。

## 3.5 贴图效果简单实现

### 3.5.1 构造球面包围盒并实现贴图

这里我们使用了Mesh类中的模型依赖于Three.js先构建了一个大的球面，并在其上实现了贴图效果。核心代码以及实现效果如下

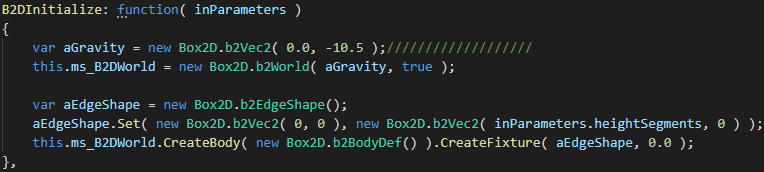




可以看到我们成功将我们的背景图贴入了我们场景的球面包围盒上。

## 3.6 碰撞检测

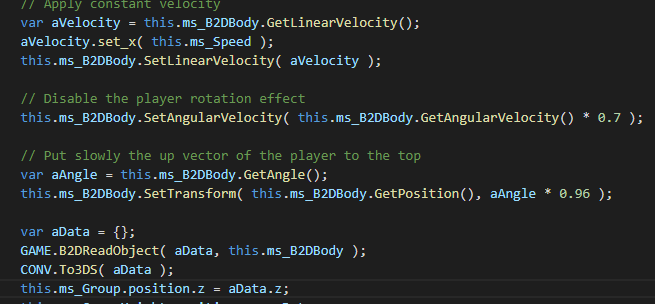
碰撞检测我们使用了Box2d引擎，

Box2d世界的初始化以及重力等参数的设置：

在生成台阶的同时对Box2d的世界进行架构：

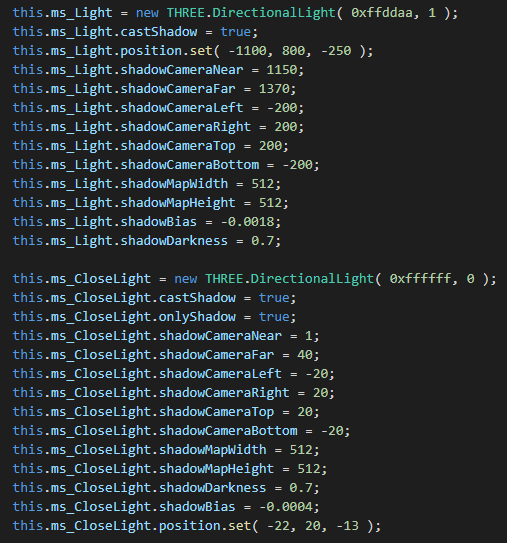


最后每隔一段时间更新box2d的世界，并根据box2d世界中物体的位置来更新我们正在跑的动物的位置：



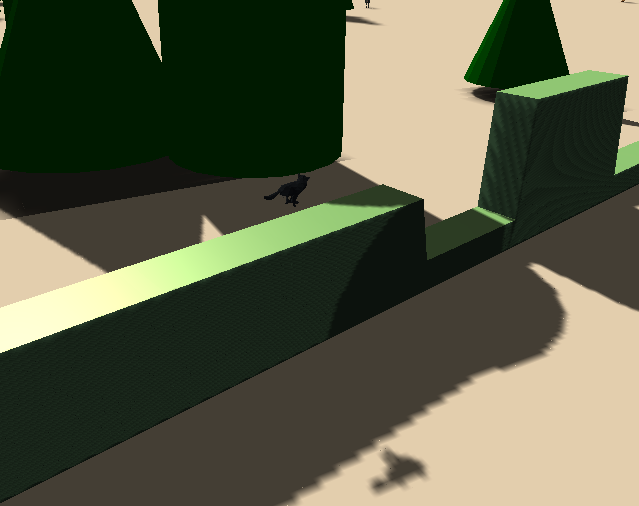
## 3.7 光照

光照我们最初是只用一条平行光以覆盖整个世界，但这将导致我们正在奔跑的动物没有影子，因而我们再加了一处较近的平行光，以获得较为清晰的影子。



另外我们的光源并不是直接与整个世界绑定的，而是与正在奔跑的动物绑定在一起，使得阴影是随着动物的奔跑而出现并变化的。

效果：

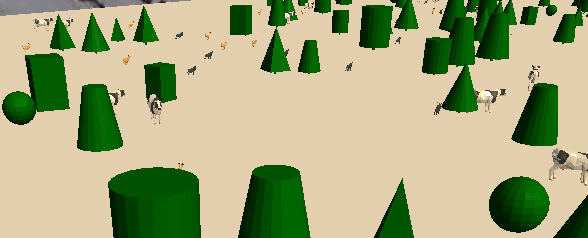


# 3.8 初始界面与游戏界面

因为游戏界面的camera是不断变化的（跟随着动物），因而我们在初始时把整个世界除了台阶和动物之外呈现。

而为了方便，采取的办法是：在初始界面新建一个canvas，在其上依然使用游戏的世界，但通过参数传递把一些不需要的功能如：跑动的动物，台阶，移动的相机给disable掉，从而得到一个可以漫游的世界。等到选择音乐完毕，则把原来的canvas给关闭，对新的canvas进行绘画，从而得到游戏的界面。

初始界面：（静态可漫游）：



游戏界面：（相机会移动，导致漫游操作困难）：

