CG项目个人报告

课程名称	计算机图形学		
年级	16级	方向	数字媒体
学号	16340072	姓名	何颢尧

负责的工作

- 上网查找CG相关技术
- 上网查找贴图资源
- 搭建基本的代码框架
- 流体
- 粒子系统
- 光照、阴影
- 昼夜变换，天空盒
- 碰撞检测
- 雪人运动
- 模板测试
- 第三人称摄像机
- OpenGL模型导入
- 3dsMax模型构建，场景设计，模型导出
- 3dsMax设计模型包围盒

心得体会

OpenGL导入模型obj格式

3dsMax软件导出obj格式的模型文件包括.obj和.mtl。.obj文件储存模型各网格的顶点坐标、法向量、纹理坐标和切线空间的向量。.mtl文件储存模型各网格的材质，材质包括纹理贴图、漫反射参数、镜面反射参数、模糊度和透明度。在OpenGL实现模型导入时，需要自行实现从mtl，obj文件读取相关信息。assimp库提供读取obj文件与mtl文件信息的函数,利用assimp库函数实现Mesh类与Model类，Mesh对应网格，网格包含顶点信息以及材质信息，Model对应模型，每个模型拥有多个Mesh对象。在Model类内实现各网格信息读取，Mesh类实现在OpenGL中渲染函数。注意：3dsMax导出的mtl中的某些参数可能与在3dsMax软件设置的参数值有区别，要自行区分设置。

OpenGL导入模型问题

若一次性导入一个很大的模型时，可能会出现溢出。建议利用3dsMax软件分开导出模型，然后在OpenGL中分开导入，即新建多个Model对象。

水面模拟

水面建模，水面外观与外观非常相似，不同点在于水面外观高度时随着时间变化的。首先将平面看作普通的均匀网格，每一个网格点(x,y)对应一个水面高度值。水面是动态的随着时间变化的，因此其网格高度值是一个与时间相关的函数H(x,y,t).（正弦函数适合模拟水面的波动）

一个简单的模型（1D）

假设水面开始静止不动，在某一个位置有一振动源作简谐振动，振动产生的波相四周传播。水面在任意位置(x,y)处高度H(x,y,t)就与传播到此的波的幅度相关。根据该针源信息以及时间t,我们就可以推导处距离该振源任意位置高度随时间的大小变换，即H(x,y,t)

假设某振源位于一维坐标系下原点处，振动产生的波沿着x轴方向传播，位于x1处点在时刻t1水面高度为H(x1,t1)与t1时刻传播到此处的波的幅度有关。

振源处振幅随时间变化函数H(t)为

A为振幅大小;T为波的振动周期

振源处发出的波沿着x轴传播。已知波的传播速度为S，波场L

在某一个固定时刻，距离振源距离l处振源大小H(l)为

振源处振幅随时间内变化函数H(t)为

结合上面两个式子，得到距离振源l处波的振幅大小为：

对于3D模型

二维平面上某点(x,y)处振幅大小随着时间的变化关系H(x,y,t)：

波长(Wavelength - L): 连续两个波峰之间的距离

振幅(Amplitude - A): 从平面到波峰的距离

波传传播速度(Speed - S): 波峰每秒钟移动距离

波浪传播方向(Directionn - D): 垂直于波阵面的向量

则水面任意点坐标为：

法向量计算

其中D·x为向量D中x分量，D·y为向量D中y分量。

OpenGL实现水面模拟问题优化

根据时间t计算各点水面的高度H(t)，在CPU中计算时，会延缓每帧的渲染时间，从而显得每帧渲染速度慢，视角移动时，会出现丢帧的不良好效果。改良的方法时，将振源的参数送往着色器，在GPU上计算，这样就能提高每帧的渲染速度。因为GPU上是并发计算的。

昼夜变换-天空盒

准备两个天空盒，一个是晴天的天空盒，一个是夜晚的天空盒，利用GLSL语言函数mix,`mix(texture(skybox1,texCoord, texture(skybox2,texCoord), rate)`,根据时间设置rate的值，使昼夜之间缓慢变换。再根据rate值改变光照强度。

粒子系统

1，粒子系统的基本思想是用许多形状简单且赋予生命的微小粒子作为基本元素来表示物体(一般由点或很小的多边形通过纹理贴图表示)，侧重于物体的总体形态和特征的动态变化。把物体定义为许多不规则，随机分布的粒子，且每个粒子均有一定的生命周期。随着时间的推移，旧的粒子不断消失(死亡)，新的粒子不断加入(生长)。粒子的这种出生，成长，衰老和死亡的过程，能够较好地反映模糊物体的动态特性。因此它比起传统的帧动画，效果要更加逼真而且节省资源。但其计算量较大，不适合大范围的特效(卡)，一般用于较小场景。 2，粒子系统实现的一般步骤：一，初始化，即创建很多粒子对象同时赋予初始属性。二，循环运行，渲染粒子<->更新粒子(如果有死亡就把其初始化为起始属性)。总的原理就是随机改变粒子的属性，然后渲染，这样循环。所谓的粒子编辑器无非就是在操作改变粒子属性罢了。

方向包围盒（OBB）碰撞检测

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）。包围体类型包括球体、轴对齐包围盒（AABB）、有向包围盒（OBB）、8-DOP以及凸壳。

方向包围盒（Oriented bounding box），简称OBB。方向包围盒类似于AABB，但是具有方向性、可以旋转，AABB不能旋转。如图3所示。

要计算两个OBB是否碰撞，只需要计算他们在图3上的4个坐标轴上的投影是否有重叠，如果有，则两多边形有接触。这也可以扩展到任意多边形

投影轴来自于多边形自身边的垂线。 判定方式：两个多边形在所有轴上的投影都发生重叠，则判定为碰撞；否则，没有发生碰撞。 OBB存在多种的表达方式，这里使用最常用的一种：一个中心点、2个矩形的边长、两个旋转轴（该轴垂直于多边形自身的边，用于投影计算）。代码如下所示：

```
(function (window) {

    var OBB = function (centerPoint, width, height, rotation) {

        this.centerPoint = centerPoint;
        this.extents = [width / 2, height / 2];
        this.axes = [new Vector2(Math.cos(rotation), Math.sin(rotation)), new Vector2(-1 *
Math.sin(rotation), Math.cos(rotation))];

        this._width = width;
        this._height = height;
        this._rotation = rotation;
    }

    window.OBB = OBB;
})(window);
```

然后基于这个数据结构，进行OBB之间的相交测试。为OBB扩展一个方法，即或者在任意轴上的投影半径：

```
OBB.prototype = {
    getProjectionRadius: function (axis) {
        returnthis.extents[0] * Math.abs(axis.dot(this.axes[0])) + this.extents[1] *
Math.abs(axis.dot(this.axes[1]));
    }
}
```

有了这些，就可以进行相交检测。由上面的判定方式，可以得出，两个矩形之间的碰撞检测需要判断四次（每个投影轴一次）。完整检测代码如下所示：

```
(function (window) {

    var CollisionDetector = {

        detectorOBBvsOBB: function (OBB1, OBB2) {
            var nv = OBB1.centerPoint.sub(OBB2.centerPoint);
            var axisA1 = OBB1.axes[0];
            if (OBB1.getProjectionRadius(axisA1) + OBB2.getProjectionRadius(axisA1) <=
Math.abs(nv.dot(axisA1))) return false;
            var axisA2 = OBB1.axes[1];
            if (OBB1.getProjectionRadius(axisA2) + OBB2.getProjectionRadius(axisA2) <=
Math.abs(nv.dot(axisA2))) return false;
            var axisB1 = OBB2.axes[0];
            if (OBB1.getProjectionRadius(axisB1) + OBB2.getProjectionRadius(axisB1) <=
Math.abs(nv.dot(axisB1))) return false;
            var axisB2 = OBB2.axes[1];
            if (OBB1.getProjectionRadius(axisB2) + OBB2.getProjectionRadius(axisB2) <=
Math.abs(nv.dot(axisB2))) return false;
            return true;

        }
    }

    window.CollisionDetector = CollisionDetector;
})(window)
```