

**Project Report**

**（ 2019 / 2020 1st Semester）**

****

|  |  |
| --- | --- |
| **Course** | Computer Graphics |
| **Date** | 2019.11.13 |
| **School** | School of Computer Science |
| **Teacher** | Dr. Huang Rui |

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Xin Chengkun | ID | B17040217 |
| School | NJUPT | Major | Computer science |

## 第一章 介绍及研究背景

1.1 OpenGL概述

OpenGL(英语：Open Graphics Library)是用于渲染2D、3D矢量图形的跨语言、跨平台的应用程序编程接口(API)，常用于CAD、虚拟现实、科学可视化程序和电子游戏开发。它是一种可以对图形硬件设备特性进行访问的软件库。

一个用来渲染图像的OpenGL程序需要执行的主要操作如下：

1.从OpenGL的几何图元中设置数据，用于构建形状。

2.使用不同的着色器(shader)对输入的图元数据执行计算操作，判断它们的位置、颜色、，以及其他渲染属性。

3.将输入图元的数学描述转化为与屏幕位置对应的像素片元(fragment)。这一步也称为光栅化(rasterization)。

4.最后，针对光栅化过程产生的每个片元，执行片元着色器(fragment shader)，从而决定这个片元的最终颜色和位置。

5.如果有必要，还需要对每个片元执行一些额外的操作，例如判断片元对应的对象是否可见，或者将片元的颜色与当前屏幕位置的颜色进行融合。

此外，OpenGL被设计为只有输出的，所以它只提供渲染功能。核心API没有窗口系统、音频、打印、键盘/鼠标或其他输入设备的概念。虽然这一开始看起来像是一种限制，但它允许进行渲染的代码完全独立于他运行的操作系统，允许跨平台开发。然而，有些集成于原生窗口系统的东西需要允许和宿主系统交互。

1.2 Particle System

着色器使我们能够利用现代提供的大规模并行性图形处理器。由于它们具有变换顶点位置的能力，因此可以用于直接在着色器本身中实现动画。这样，如果动画算法可以适当并行化以在着色器，它就可以提供一个意想不到的效果。

本项目是基于OpenGL的天空盒和模型加载，并在此基础上实现粒子系统。如果一个着色器是帮助我们实现动画的，那么他必须包含两个方面，一是计算位置，二是提供下一帧的位置。着色器最初不是设计用于写入任意缓冲区的（当然帧缓冲区除外）。但是随着版本的不断更新，OpenGL提供了通过着色器存储缓冲区对象和图像执行此操作的功能——加载/存储。从OpenGL 3.0开始，我们还可以发送顶点或几何着色器的值将变量输出到任意缓冲区（或多个缓冲区）。此功能就是对粒子系统特别有用变换反馈。

变换反馈提供了一种捕获顶点（或几何）着色器输出的方法到缓冲区以供后续遍历使用。最初在3.0版中引入OpenGL，此功能特别适合于粒子系统，因为它尤其具有以下特点：

1.使我们能够进行离散仿真。

2.可以更新粒子在顶点内的位置着色器，并在后续遍（或同一遍）中渲染更新的位置。

3.更新的位置可以与输入动画的下一帧相同的方式使用。

因此，使用变换反馈实现粒子系统是本项目的最佳选择。

## 第二章 理论与算法

2.1 模型加载

2.1.1 Assimp

在日常的图形程序中，通常都会使用非常复杂且有趣的模型，然而我们不能对这些复杂的图形手工定义所有的顶点、法线和纹理坐标。因此，我们需要的是将这些模型(Model)导入(Import)到程序当中。

但是模型的文件格式有很多种，每一种都会以它们自己的方式来导出模型数据，像是Wavefront的.obj这样的模型格式，只包含了模型数据以及材质信息，比如模型颜色和漫反射/镜面光贴图。而以XML为基础的Collada文件格式则非常的丰富，包含模型、光照、多种材质、动画数据、摄像机、完整的场景信息等等。Waveform的.obj格式通常被认为是一个易于解析的模型格式。总而言之，不同种类的文件格式有很多，它们之间通常并没有一个通用的结构。所以如果我们想从这些文件格式中导入模型的话，就必须要去对每一种需要导入的文件格式写一个导入器。Assimp库应运而生。

Assimp(Open Asset Import Library，开放式资产导入库)是一个非常流行的模型导入库，能够导入很多种不同模型的文件格式（也能导出部分格式）。Assimp会将所有的模型数据加载至Assimp的通用数据结构中。当Assimp加载完模型之后，程序就能够从Assimp的数据结构中提取所需要的所有数据。由于Assimp的数据结构保持不变，所以不论导入的是什么格式的文件，它都能够将我们需要的数据从这些文件格式中抽象出来。

Assimp数据结构的简化模型如下：

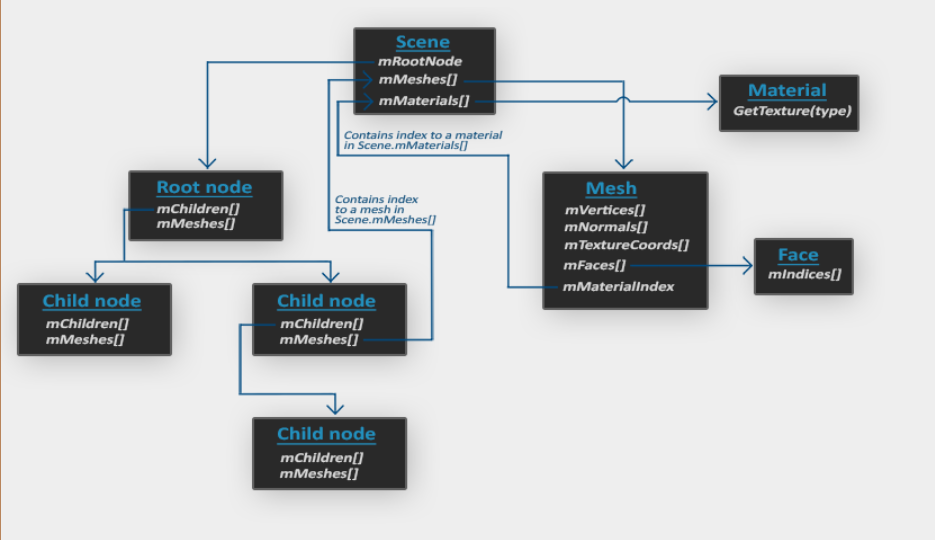


图2-1 Assimp的数据结构模型

当使用Assimp导入一个模型时，通常会将整个模型加载进一个场景(Scene)对象中，它会包含导入的模型/场景中的所有数据。Assimp会将场景载入为一系列的节点(Node)，每个节点包含了场景对象中所储存数据的索引，每个节点都可以有任意数量的子节点。

所以，首先要做的就是将一个物体加载到Scene对象中，遍历节点获取对应的Mesh对象（通过递归搜索每个节点的子节点），并处理每个Mesh对象来获取顶点数据、索引以及它的材质属性，生成第系列的网格数据，以便进行后续操作。

2.1.2 Model

Model应包含一个Mesh对象的vector，并且构造器需要一个文件路径。在导入模型的过程中，首先调用的函数是loadModel，它会直接从构造器中调用。在这个函数中，使用Assimp来加载模型至Assimp的一个Scene数据结构中。一旦有了这个对象，程序就能访问到加载后的模型中所有所需的数据了。

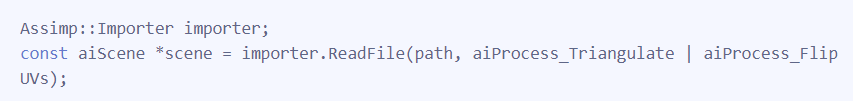


图2-5 加载模型代码

2.2 Particle

2.2.1 Particle的实现

Particle（粒子）从OpenGL的角度来看，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，其本身就是一个Sprite（精灵）。

要处理这些粒子，通常由一个叫做粒子发生器或粒子生成器的东西来完成。粒子发生器持续不断地产生新的微粒并且旧的微粒随着时间逐渐消亡。如果这个粒子发生器产生一个带着类似烟雾纹理的微粒，它的颜色亮度同时又随着与发生器距离的增加而变暗，那么就会产生出灼热的火焰的效果。

一个单一的微粒通常有一个生命值变量，并且从它产生开始就一直缓慢的减少。一旦它的生命值少于某个极限值（通常是0），程序就会杀掉这个粒子，这样下一个粒子产生时就可以让它来替换那个被杀掉的粒子。一个粒子发生器控制它产生的所有粒子并且根据它们的属性来改变它们的行为，通常具以下属性：

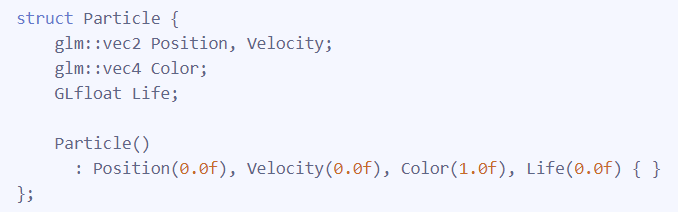


图2-6 粒子发生器属性

以下图火焰的粒子系统为例，粒子发射器可能在靠近发射器的地方产生一个粒子，并且有一个向上的速度，这样每个粒子都是朝着正y轴方向移动。那似乎有3个不同区域，只是相比其他的区域给了某个区域内的粒子更快的速度。我们也可以看到，y轴方向越高的粒子,它们的黄色或者说亮度就越低。一旦某个粒子到达某个高度的时候，它的生命值就会耗尽然后被杀掉，绝不可能直冲云霄。

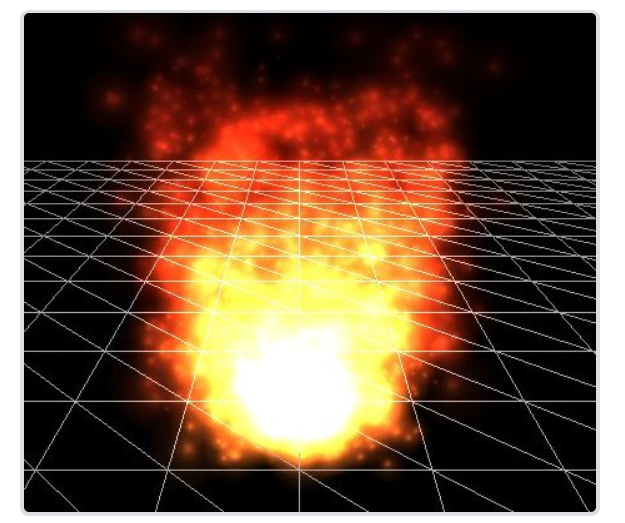


图2-7 粒子系统火焰效果

我们利用这样的系统，就可以创造出有趣的效果比如火焰、青烟、烟雾、魔法效果和炮火残渣等。

粒子系统 (particle system) 是迄今为止被认为模拟不规则模糊物体最为成功的一种图形生成算法。粒子系统采用了一套完全不同于以往造型、绘制系统的方法来构造、绘制景物, 它并不是一个简单的静态模型, 而是一种过程计算模型。如果存在有效的物理或生理模型, 物体的行为建模将变得十分简单, 只要实现当前的几何模型即可。

一个粒子系统由许多个称为粒子的简单体素构成, 每个粒子均具有形状、大小、位置、颜色、透明度、运动速度、运动方向、生命周期等属性, 而一个粒子究竟有什么样的属性, 主要取决于粒子系统用来模拟什么。粒子系统是不断进化的, 是动态变化的, 随着时间的推移, 系统中不断有新粒子的加入, 旧粒子的死亡, 系统中“存活”的粒子其位置及生命值亦随时间变化而变化, 其正常运行的关键是确定粒子的初始属性、粒子的变化规律和绘制等因素。粒子系统的这些属性与变化规律的可控性使粒子系统可以用来模拟许多动态自然景象 (如雨、雪、烟、云、火焰等) 。

在粒子生命期的每一刻, 都要完成以下五步工作:

1) 粒子源产生新粒子并赋予粒子属性后加入系统中。

2) 根据粒子的动态属性对粒子进行移动和变换, 同时更新粒子属性。

3) 判断粒子的生命值。

4) 删除那些已经超过其生命周期的粒子。

5) 绘制并显示由有生命的粒子组成的图形。

2.2.2 变换反馈(Transform Feedback)

变换反馈是实现粒子系统常用的方法之一。在这种方法中，我们不用描述粒子运动的方程式来更新粒子的位置，而采用一种基于每个时间所涉及的力来逐步地求解运动方程进而渲染出粒子。一个最常用的方法就是欧拉方法。此种方法将位置、t时刻的瞬时速度基于上一时刻的位置、速度和加速度混合在了一起。

式(2-1)

式(2-2)

为了使代码能够正常运行，我们将使用一种有时称为缓冲区的“乒乓”的技术。我们包含了两组顶点缓冲区，并在每一帧交换它们使用的数据。例如，使用buffer A提供位置和速度来作为顶点着色器的输入。顶点着色器使用Euler方法更新位置和速度，并使用变换反馈的方法将结果发送到buffer B。然后在第二遍渲染的时候，使用buffer B来渲染粒子。在接下来的动画中，重复这个动作，并交换这两个buffer的值。

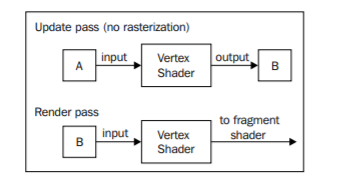


图2-8 变换反馈原理图

## 第三章 实验

3.1 实验要求

基础部分：编译Assimp，导入一个球体。加入天空盒并使球体有镜面效果。

附加部分：利用变换反馈或计算Shader实现粒子系统。

3.2 关键步骤实现过程

确定项目后，根据需求，按照粒子系统工作原理对粒子进行操作，将有生命粒子绘制在窗口中即实现粒子系统的实时更新。（如图3-1）

3.2.1 粒子初始化

每个粒子都有一定的生命期，赋予每个粒子一个生命值T；由于粒子做匀速运动，v为常数；有生命的粒子T>0。

窗口中的固定位置产生粒子源，其初始位置(x,y,z)的坐标值为：

X={-width/2, 0, width/2};

Y=T\*V;

Z={-width/2, 0, -width};

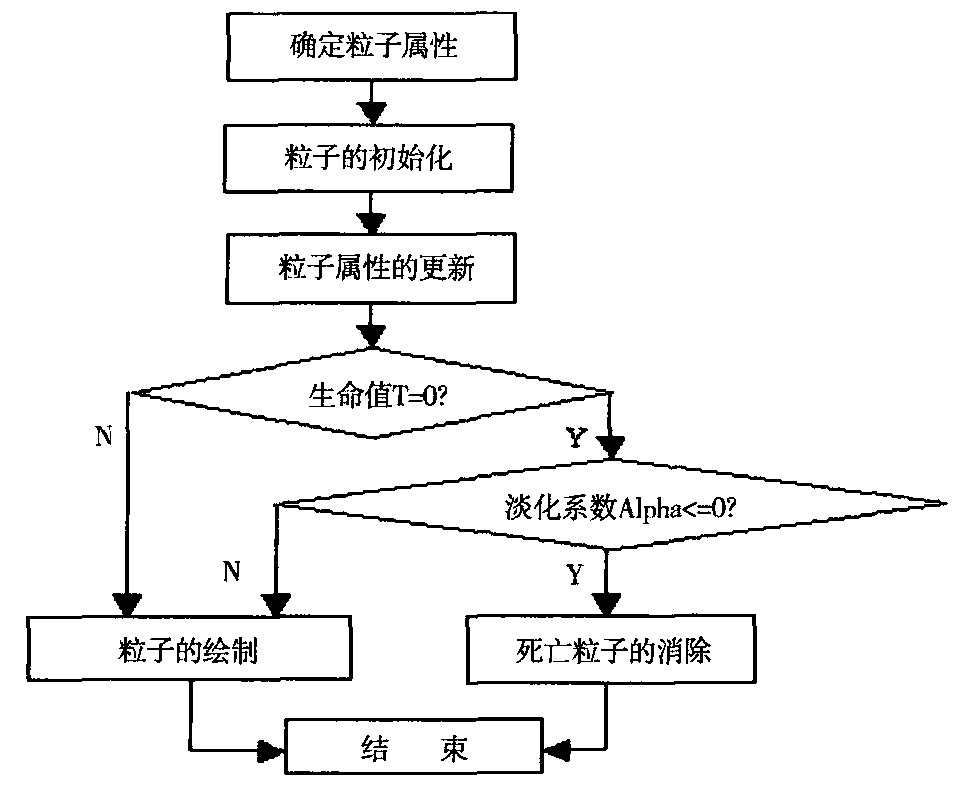


图3-1 粒子系统工作流程图

3.2.2 死亡粒子的消除

用粒子系统实现此项目，就不断会有新粒子的产生与旧粒子的死亡，应及时把已经死亡的粒子从系统中删除以释放系统资源。当粒子的生命值T<=0时，表示粒子已经死亡，应将其从系统中删除。

## 第四章 结论

通过实验灵活运用OpenGL图形API函数，自行设计出基于Assimp的模型导入并通过天空盒纹理图案实现反射的效果。同时也学会了配置OpenGL图形函数API，并在屏幕上显示基本的3D图形。通过对变换矩阵的一些小操作，就是将去除变换矩阵的位移量，可以实现3D模型只在天空盒内部移动，而不会移到天空盒外边。这样，就实现了天空盒当背景的效果。

然而本次实验较难的部分就是附加的粒子系统，我没有实现。主要原因是能力没有达到，不懂OpenGL如何调试，使得自己写出的代码运行不出正确的结果而找不出原因的所在。除此之外，粒子系统常用方法的原理掌握得不好，导致不能在很短的时间内学以致用。我对于深度测试的概念也不是很理解，希望以后能够更加努力，加强对这些知识的理解与运用。

## 第五章 参考文献

[1] Joey de Vries.learnopengl[EB/OL].(2014.06)[2019.11.14]. https://learnopengl.com

[2]Dave Shreiner,Graham Sellers．OpenGL编程指南[M].王锐，译.原书第8版.北京：机械工业出版社,2014：1-20 ．

[3] 王海伦. 基于OpenGL的场景动态效果模拟的研究与实现[D].上海:复旦大学,2011.

[4] David Wolff．OpenGL4 Shading Language Cookbook[M]．Second Edition．UK.：Packt Publishing Ltd.,2013：315-343．

[5] 徐利明，姜昱明. 基于粒子系统与OpenGL的实时雨雪模拟[D].陕西西安:西安电子科技大学计算机学院.2004