

**本科毕业设计说明书**

**基于OpenGL的简单渲染引擎设计与实现**

学院（部）： 计算机科学与工程学院

专业班级： 计算机科学与技术14-2

学生姓名： 程 亮

指导教师： 程 建

2018年 5 月 25 日

基于OpenGL的简单渲染引擎设计与实现

摘要

渲染引擎根据是否能够被主流计算机即时计算出结果分为即时渲染引擎和离线渲染引擎。[PC机](https://baike.baidu.com/item/PC%E6%9C%BA)及游戏机上的即时3D画面就是用即时渲染引擎运算生成的，而电影中应用的3D画面则是用离线渲染引擎来实现以达到以假乱真的效果。

渲染引擎对物质的抽象主要分为多边形和[NURBS](https://baike.baidu.com/item/NURBS/550944)两种。在即时引擎中多边形实现已经成为了事实上的标准，因为任何多边形都可以被最终分解为容易计算和表示的三角形。而在离线引擎中为了追求最好的视觉效果会使用大量的NURBS曲线来实现多边形很难表现出的细节和灵活性。

OpenGL是一套应用程序编程接口(API)，借助这个API我们开发人员就可以开发出对图形硬件具有访问的能力的程序。我们可以使用OpenGL开发出运行效率较高的图形程序或游戏，因为OpenGL非常接近底层硬件并且OpenGL使得我们不必去关注图形硬件的细节。既然我们开发人员不必关注图形硬件的细节，那么我们需要关注什么呢？我们需要关注OpenGL如何绘制，按照专业术语就是根据物体的规格参数及相关属性，借助虚拟照相机和光照生成一幅该物体的图像。OpenGL程序与平台是无关的，所以OpenGL API中不包含任何输入函数或窗口函数，原因是因为这两种函数都要依赖于特定的平台，例如Windows，Linux或是其他系统。

本文中我们利用OpenGL，C++编程语言来实现一个简单的渲染引擎,并借此学习图形学中的一些知识。

**关键词：渲染，引擎，OpenGL，图形学，C++**

THE DESIGN OF FACE RECOGNITION SYSTEM BASED ON HAAR ALGORITHM

ABSTRACT

Face recognition is a popular topic in the world and the research of this topic is very important, it covers lots of disciplines and areas like process of digital images, recognition of patterns, computer vision, neural networks, mathematics, physiology, psychology, etc. At the same time the technology of face recognition is not perfect, it’s affected by the changes of many factors such as facial expressions, gestures and so on. This paper discusses how to implement the real-time recognition of multiple faces with high accuracy on the .NET platform, generally describes some common face recogonition methods and analyzes the Haar eigenvalues and PCA methods, and improves the quality of image which concerns the information of facial feature based on PCA methods, and develop a simple face recognition system written with C# language, this system has a high accuracy of recognition.

**KEYWARDS：face recognition，.net，haar，pca，c#**

目录

[摘要 I](#_Toc514276524)

[**ABSTRACT** II](#_Toc514276525)

[1. 绪论 1](#_Toc514276526)

[1.1 计算机图形学介绍 1](#_Toc514276527)

[1.2 计算机图形学的确立和发展 1](#_Toc514276528)

[1.3 OpenGL介绍 1](#_Toc514276529)

[1.4 常用模型和算法 2](#_Toc514276530)

[1.4.1 渲染管线 2](#_Toc514276531)

[1.4.2 颜色模型 3](#_Toc514276532)

[1.4.3 Phong氏光照模型 4](#_Toc514276533)

[1.5 MongoEngine技术架构 4](#_Toc514276534)

[1.5.1 所用到的库 4](#_Toc514276535)

[1.5.2 运行环境 4](#_Toc514276536)

[2. 总体设计 5](#_Toc514276537)

[2.1 需求分析 5](#_Toc514276538)

[2.2 架构设计 5](#_Toc514276539)

[3. 详细设计与实现 7](#_Toc514276540)

[3.1 Camera组件 7](#_Toc514276541)

[3.2 Shader组件 8](#_Toc514276542)

[3.3 Texture组件 9](#_Toc514276543)

[3.4 Mesh组件 10](#_Toc514276544)

[3.5 Material组件 11](#_Toc514276545)

[3.6 Loader组件 12](#_Toc514276546)

[3.7 Light组件 12](#_Toc514276547)

[3.8 Scene组件 15](#_Toc514276548)

[4. 实现效果 15](#_Toc514276549)

[5. 总结 15](#_Toc514276550)

[总结 16](#_Toc514276551)

[参考文献 18](#_Toc514276552)

[致谢 20](#_Toc514276553)

1. 绪论

## 计算机图形学介绍

计算机图形学（CG，Computer Graphics）是随着计算机技术的发展，特别是图形显示器的发展而产生和发展起来的，是计算机技术和和电视技术和图形处理技术相互融合的结果。当我们使用计算机的时候，离我们最近的就是图形化的用户界面，这就是计算机图形学给我们的最直接感受。近些年来，计算机图形学已经在游戏、电影、商业、广告、军事、教学、培训等领域获得了广泛的应用。社会的需求反过来也同样会促进计算机图形学的快速发展，计算机图形学已经形成了一个巨大的产业。

## 计算机图形学的确立和发展

计算机图形学诞生可以追溯到上世纪六十年代初，是与计算机硬件技术特别是图形显示器制造技术的发展密不可分的。

1950年美国麻省理工学院的旋风一号计算机配备了世界上的第一台显示器，使计算机摆脱了纯数值计算的单一用途，能够进行简单的图形显示，但还不能进行交互操作，被称为“被动式”的计算机图形学。

1963年美国麻省理工学院的Ivan E.Sutherland完成了《Sketchpad:A Man-Machine Graphical Communication System》博士学位论文。该论文首次使用了Computer Graphics作为术语，证明了交互式计算机图形学是一个可行的、有应用价值的研究领域，从而确立了计算机图形学作为一个崭新的学科的独立地位。

## OpenGL介绍

OpenGL（全写Open Graphics Library）是指定义了一个跨编程语言、跨平台的[编程接口](https://baike.baidu.com/item/%E7%BC%96%E7%A8%8B%E6%8E%A5%E5%8F%A3)规格的专业的图形[程序接口](https://baike.baidu.com/item/%E7%A8%8B%E5%BA%8F%E6%8E%A5%E5%8F%A3)。它用于[三维图像](https://baike.baidu.com/item/%E4%B8%89%E7%BB%B4%E5%9B%BE%E5%83%8F)（二维的亦可），是一个功能强大，调用方便的底层图形库。

OpenGL™ 是行业领域中最为[广泛](https://baike.baidu.com/item/%E5%B9%BF%E6%B3%9B)接纳的 2D/3D 图形 [API](https://baike.baidu.com/item/API/10154)，其自诞生至今已催生了各种计算机平台及设备上的数千优秀应用程序。OpenGL™ 是独立于视窗操作系统或其它操作系统的，亦是网络透明的。在包含CAD、内容创作、能源、娱乐、游戏开发、制造业、制药业及[虚拟现实](https://baike.baidu.com/item/%E8%99%9A%E6%8B%9F%E7%8E%B0%E5%AE%9E)等行业领域中，OpenGL™ 帮助程序员实现在 PC、[工作站](https://baike.baidu.com/item/%E5%B7%A5%E4%BD%9C%E7%AB%99)、[超级计算机](https://baike.baidu.com/item/%E8%B6%85%E7%BA%A7%E8%AE%A1%E7%AE%97%E6%9C%BA)等硬件设备上的高性能、极具冲击力的高视觉表现力图形处理软件的开发。

OpenGL是个与硬件无关的[软件接口](https://baike.baidu.com/item/%E8%BD%AF%E4%BB%B6%E6%8E%A5%E5%8F%A3)，可以在不同的平台如[Windows 95](https://baike.baidu.com/item/Windows%2095)、[Windows NT](https://baike.baidu.com/item/Windows%20NT)、[Unix](https://baike.baidu.com/item/Unix)、[Linux](https://baike.baidu.com/item/Linux)、MacOS、OS/2之间进行移植。因此，支持OpenGL的软件具有很好的移植性，可以获得非常广泛的应用。由于OpenGL是图形的底层图形库，没有提供几何实体图元，不能直接用以描述场景。

但是，通过一些转换程序，可以很方便地将[AutoCAD](https://baike.baidu.com/item/AutoCAD)、3DS/3DSMAX等3D图形设计软件制作的DXF和3DS模型文件转换成OpenGL的顶点数组。

在OpenGL的基础上还有Open Inventor、Cosmo3D、Optimizer等多种高级图形库，适应不同应用。其中，Open Inventor应用最为广泛。该软件是基于OpenGL面向对象的工具包，提供创建交互式3D图形应用程序的对象和方法，提供了预定义的对象和用于交互的事件处理模块，创建和编辑3D场景的高级应用程序单元，有打印对象和用其它图形格式交换数据的能力。

## 常用模型和算法

### 渲染管线

所谓渲染管线就是OpenGL的管道当中各个部分的功能以及如何在管道当中形成了我们想要的最终的一幅图。MongoEnginecai用的是可编程渲染管线，其渲染管线主要包括以下流程：

图1-1 可编程渲染管线

准备顶点数据：将顶点数据包括位置、法线、纹理坐标等等从CPU上传送到GPU上。

顶点着色器：顶点着色器中输入值是我们之前传入到GPU上的顶点数据，通过一系列运算后，输出顶点在世界坐标系下的位置，同时可以向接下来的着色器传输必要数据。这个阶段是可编程的。

图元装配：将上一阶段顶点着色器的输出值根据图元类型进行装配。

几何着色器：几何着色器的输入是一个图元（如点或三角形）的一组顶点。几何着色器可以在顶点发送到下一着色器阶段之前对它们随意变换。这个阶段是可编程的。

测试和混合：测试包括深度测试，模板测试等，混合指的是当前像素值和原先的像素值进行组合，可以用不同的混合方式达到不同的效果。

片段着色器：在片段着色器中进行一系列计算，输出的片段的最终颜色。这个阶段是可编程的。

光栅化：光栅化的过程是将我们计算出来的每个片段的颜色同屏幕上的每个像素点对应，决定每个像素的颜色，即我们最终看到的样子。

概括来讲，渲染管线是从模型数据到图像生成过程的一种描述。Vertex Shader能对顶点数据写处理算法，而 Fragment Shader允许我们对像素数据写处理算法。

### 颜色模型

光是波长在可见光谱范围内的电磁波。可见光的波长大约在400-700nm，正是这样的电磁波让人产生了颜色的感觉，但光本身有颜色，颜色是外来的光线刺激人的视觉器官而产生的主观感觉。根据人眼的视觉颜色感知模型，我们将红绿蓝作为三原色，其他所有的颜色都可以由这三种颜色以一定的比例混合得到。

现实世界中有无数种颜色，每一个物体都有它们自己的颜色。我们需要使用（有限的）数值来模拟真实世界中（无限）的颜色，所以并不是所有现实世界中的颜色都可以用数值来表示的。然而我们仍能通过数值来表现出非常多的颜色，甚至你可能都不会注意到与现实的颜色有任何的差异。颜色可以数字化的由红色(Red)、绿色(Green)和蓝色(Blue)三个分量组成，它们通常被缩写为RGB。仅仅用这三个值就可以组合出任意一种颜色。

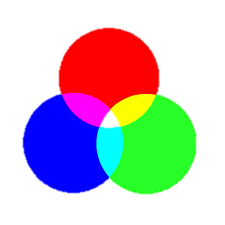
MongoEngine中用到的颜色模型就是RGB模型，同样RGB模型也是显示器的颜色物理模型。我们用一个三维向量来表示一种颜色，并将每个颜色分量规范化到0-1.0范围内。

图1-2 RGB三原色

### Phong氏光照模型

现实世界的光照是极其复杂的，而且会受到诸多因素的影响，这是我们有限的计算能力所无法模拟的。因此OpenGL的光照使用的是简化的模型，对现实的情况进行近似，这样处理起来会更容易一些，而且看起来也差不多一样。这些光照模型都是基于我们对光的物理特性的理解。其中一个模型被称为冯氏光照模型(Phong Lighting Model)。冯氏光照模型的主要结构由3个分量组成：环境(Ambient)、漫反射(Diffuse)和镜面(Specular)光照。

环境光照(Ambient Lighting)：即使在黑暗的情况下，世界上通常也仍然有一些光亮（月亮、远处的光），所以物体几乎永远不会是完全黑暗的。为了模拟这个，我们会使用一个环境光照常量，它永远会给物体一些颜色。

漫反射光照(Diffuse Lighting)：模拟光源对物体的方向性影响(Directional Impact)。它是冯氏光照模型中视觉上最显著的分量。物体的某一部分越是正对着光源，它就会越亮。

镜面光照(Specular Lighting)：模拟有光泽物体上面出现的亮点。镜面光照的颜色相比于物体的颜色会更倾向于光的颜色。



图1-3 phong氏光照模型

## MongoEngine技术架构

### 所用到的库

MongoEngine中用到了一些现有的c++库，模型解析用的是assimp，数学库用的是glm,文本渲染用的是freetype，播放声音用的是irrklang，版本控制用的是git。

### 运行环境

MongoEngine的开发环境是windows10操作系统，VisualStudio2017集成开发环境，用的编程语言是C++。

1. 总体设计

## 需求分析

MongoEngine作为一个简单的渲染引擎，完成的功能有：

1. 加载资源，包括shader,texture,mesh等；
2. 解析obj模型；
3. 冯氏光照系统的实现；
4. 场景节点树的构建；

## 架构设计

MongoEngine主要的架构如图1-3所示：

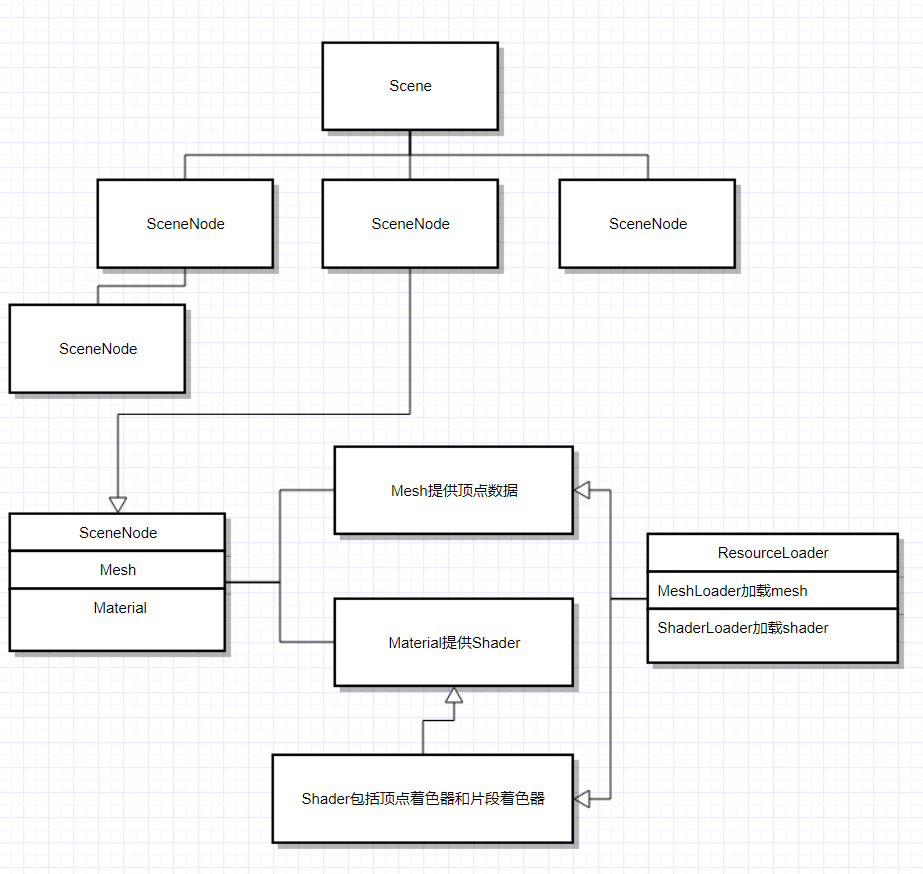


图1-3 MongoEngine架构

MongoEngine主要由几个组件组成，包括:

1. Camera组件：负责观察点的移动，在场景中漫游，以便更好的观察整个场景。
2. Shader组件：负责读取shader文件，创建shader，并编译，设置着色器中的uniform变量。
3. Texture组件：对纹理的抽象形成的一种数据结构。
4. Mesh组件：负责提供定点数据。
5. Material组件：对shader的封装。
6. Loader组件：MongoEngine的资源加载系统，可以加载包括shader、纹理、mesh等资源。
7. Light组件：对光源的抽象，不同光源持有不同的光源参数。
8. Scene组件：负责整个场景的初始化，持有场景节点树，在渲染循环中通过深度优先遍历便利整个节点树，并渲染节点，并且持有整个场景中的光源集合，负责对光源的更新。
9. 详细设计与实现

## Camera组件

当我们渲染一个场景时，需要通过一个摄像机来观察整个场景，并且这个摄像机要能根据我们的鼠标键盘输入在场景中移动，OpenGL中并没有摄像机的概念，所以我们需要自己通过抽象来实现一个摄像机系统。

当我们在讨论摄像机的时候，实际上就是在讨论场景中的顶点的世界坐标在摄像机定义的摄像机空间中的坐标，这个时候我们需要定义好这个坐标空间的三个轴和坐标原点即可，分别是摄像机在世界空间中的位置，观察方向，右方向，上方向。

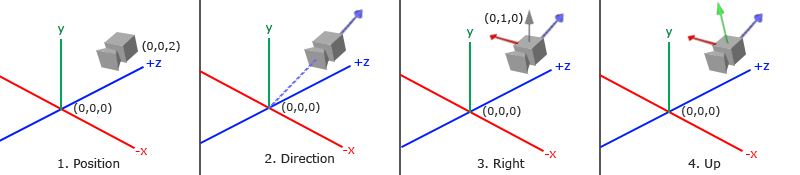


图3-1摄像机空间

摄像机类所持有的参数信息:

|  |
| --- |
| glm::vec3 Position;  glm::vec3 Front;  glm::vec3 Up;  glm::vec3 Right;  glm::vec3 WorldUp;  GLfloat Yaw;  GLfloat Pitch;  GLfloat MovementSpeed;  GLfloat MouseSensitivity;  GLfloat Zoom; |

当我们创建好这个坐标空间后，就可以利用glm的lookat方法构建一个view矩阵，这个矩阵就是我们所需要的变换矩阵。

glm::lookAt(rightVec, posVec, upVec);

当我们需要用键盘鼠标控制摄像机移动时，需要向opengl注册鼠标键盘输入的回调函数，并在回调函数中处理我们的摄像机的相关参数。

|  |
| --- |
| void ProcessMouseMovement(GLfloat xoffset, GLfloat yoffset, GLboolean constrainPitch = true)  {  xoffset \*= this->MouseSensitivity;  yoffset \*= this->MouseSensitivity;  this->Yaw += xoffset;  this->Pitch += yoffset;  if (constrainPitch)  {  if (this->Pitch > 89.0f)  this->Pitch = 89.0f;  if (this->Pitch < -89.0f)  this->Pitch = -89.0f;  }  this->updateCameraVectors();  }  void ProcessMouseScroll(GLfloat yoffset)  {  if (this->Zoom >= 1.0f && this->Zoom <= 45.0f)  this->Zoom -= yoffset;  if (this->Zoom <= 1.0f)  this->Zoom = 1.0f;  if (this->Zoom >= 45.0f)  this->Zoom = 45.0f;  } |

当我们需要用到摄像机的view矩阵时只需要通过其暴漏在外部的接口调用即可。

|  |
| --- |
| glm::mat4 GetViewMatrix()  {  return glm::lookAt(this->Position, this->Position + this->Front, this->Up);  } |

## Shader组件

着色器是运行在GPU上的小程序，这些程序是在图形渲染管线中的某些特定部分中运行的。着色器实际上是一种将输入转化为输出的程序。

Shader组件实际上是对opengl中的着色器程序的封装，在opengl中我们可以提供着色器程序的源代码字符串，然后调用opengl的函数进行编译链接，通过我们的封装我可以直接给一个文本字符串给shader组件，然后它会自动帮我们读取字符串，编译链接着色器程序，而我们所需要的就是一个编译链接成功的着色器程序id，当我们需要使用的时候，直接调用glUseProgram(ID)即可。

同时，我们的shader组件还负责设置着色器程序中的uniform变量的值，使得我们的程序可以同着色器程序进行沟通。

我们的shader组件需要对顶点着色器和片段着色器负责，其中顶点着色器程序的输入是顶点属性，输出值是顶点位置，片段着色器程序的输入是上一阶段的输出，输出值是片段的颜色。

我们的着色器组件的大致构成如下所示：

|  |
| --- |
| unsigned int ID;  Shader();  Shader(std::string name, std::string vsCode, std::string fsCode, std::vector<std::string> defines = std::vector<std::string>());  void Load(std::string name, std::string vsCode, std::string fsCode, std::vector<std::string> defines = std::vector<std::string>());  void Use();  bool HasUniform(std::string name);  void SetInt(std::string location, int value);  void SetBool(std::string location, bool value);  void SetFloat(std::string location, float value);  void SetVector(std::string location, glm::vec2 value);  void SetVector(std::string location, glm::vec3 value);  void SetVector(std::string location, glm::vec4 value);  void SetVectorArray(std::string location, int size, const std::vector<glm::vec2>& values);  void SetVectorArray(std::string location, int size, const std::vector<glm::vec3>& values);  void SetVectorArray(std::string location, int size, const std::vector<glm::vec4>& values);  void SetMatrix(std::string location, glm::mat2 value);  void SetMatrix(std::string location, glm::mat3 value);  void SetMatrix(std::string location, glm::mat4 value); |

## Texture组件

当我们在指定顶点属性时，可以指定顶点的UV坐标，这样我们就可以在纹理中根据纹理坐标进行采样，从而决定顶点的颜色。纹理是一张2D图片，用来添加物体的细节。

纹理坐标在x-y轴上，两个轴上的坐标范围被限定在0-1之间。

我们的Texture组件主要负责生成纹理，向纹理中传输纹理的像素数据，并可以设置纹理的环绕方式和纹理过滤的方式。

Texture组件的大致构成如下所示：

|  |
| --- |
| Texture();  ~Texture();  void Generate(unsigned int width, GLenum internalFormat, GLenum format, GLenum type, void\* data);  void Generate(unsigned int width, unsigned int height, GLenum internalFormat, GLenum format, GLenum type, void\* data);  void Generate(unsigned int width, unsigned int height, unsigned int depth, GLenum internalFormat, GLenum format, GLenum type, void\* data);  void Resize(unsigned int width, unsigned int height = 0, unsigned int depth = 0);  void Bind(int unit = -1);  void Unbind();  void SetWrapMode(GLenum wrapMode, bool bind = false);  void SetFilterMin(GLenum filter, bool bind = false);  void SetFilterMax(GLenum filter, bool bind = false); |

## Mesh组件

Mesh组件是MongoEnigne比较重要的一个组件，我们通过mesh组件向GPU上传输顶点数据，其中mesh类有几个子类，是MongoEnigne中自带的一些常用网格。

1. CircleMesh：圆形平面的网格类；
2. CubeMesh：长方体网格类；
3. PlaneMesh:平面网格类；
4. QuardMesh：面片网格类；
5. SphereMesh：球体网格类；

这里我们以比较复杂的球体网格类的实现为例：

SphereMesh中我们需要生成球体上面的每个点的坐标，用到的是球的参数方程。

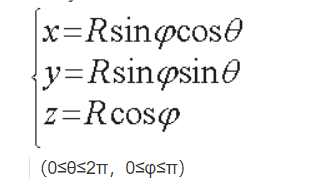


图3-2球体的参数方程

核心代码实现为：

|  |
| --- |
| Sphere::Sphere(unsigned int xSegments, unsigned int ySegments)  {  Name = "sphereMesh";  for (unsigned int y = 0; y <= ySegments; ++y)  {  for (unsigned int x = 0; x <= xSegments; ++x)  {  float xSegment = (float)x / (float)ySegments;  float ySegment = (float)y / (float)ySegments;  float xPos = std::cos(xSegment \* TAU) \* std::sin(ySegment \* PI); // TAU is 2PI  float yPos = std::cos(ySegment \* PI);  float zPos = std::sin(xSegment \* TAU) \* std::sin(ySegment \* PI);  Positions.push\_back(glm::vec3(xPos, yPos, zPos));  UV.push\_back(glm::vec2(xSegment, ySegment));  Normals.push\_back(glm::vec3(xPos, yPos, zPos));  }  }  bool oddRow = false;  for (int y = 0; y < ySegments; ++y)  {  for (int x = 0; x < xSegments; ++x)  {  Indices.push\_back((y + 1) \* (xSegments + 1) + x);  Indices.push\_back(y \* (xSegments + 1) + x);  Indices.push\_back(y \* (xSegments + 1) + x + 1);  Indices.push\_back((y + 1) \* (xSegments + 1) + x);  Indices.push\_back(y \* (xSegments + 1) + x + 1);  Indices.push\_back((y + 1) \* (xSegments + 1) + x + 1);  }  }  Topology = TRIANGLES;  Finalize();  } |

## Material组件

在现实世界中物体对光会产生不同的反应。我们在opengl中来模拟不同的物体时就需要为每个物体指定一个材质属性。

Material组件，实际上是对Shader组件的封装，在这里面我们会有对shader中的uniform变量设置的接口，同时为了效率，当我们设置非纹理类型的变量时，我们不直接设置shader中的变量，而是先将变量名和变量值通过map储存起来，当我们使用的时候再同统一向shder中设置对应的值。

并且我们在Material中还提供了一个Copy接口，通过这个接口可以返回一个深度拷贝的材质副本。

Material的大致构成如下所示：

|  |
| --- |
| Shader \* m\_Shader;  std::map<std::string, UniformValue> m\_Uniforms;  std::map<std::string, UniformValueSampler> m\_SamplerUniforms;  void Use();  Material();  Material(Shader\* shader);  Shader\* GetShader();  void SetShader(Shader\* shader);  Material Copy();  void SetBool(std::string name, bool value);  void SetInt(std::string name, int value);  void SetFloat(std::string name, float value);  void SetTexture(std::string name, Texture\* value, unsigned int unit = 0);  void SetTextureCube(std::string name, TextureCube\* value, unsigned int unit = 0);  void SetVector(std::string name, glm::vec2 value);  void SetVector(std::string name, glm::vec3 value);  void SetVector(std::string name, glm::vec4 value);  void SetMatrix(std::string name, glm::mat2 value);  void SetMatrix(std::string name, glm::mat3 value);  void SetMatrix(std::string name, glm::mat4 value); |

## Loader组件

Loader组件是MongoEnigne的资源加载器，负责加载我们需要用的资源，主要由MeshLoader、ShaderLoader、TextureLoader三大部分构成。

1. MeshLoader

MongoEngine中自带的mesh我们可以直接实例化得到，在它的构造函数中我们就填充了它的顶点数据，而MongoEnigne可以加载其他建模软件做好的模型文件obj文件，所以我们需要用MeshLoader来加载解析这样的obj文件。其中用到的库是assimp。

在模型的加载过程中，我们直接给出模型的文件名，然后通过assimp库加载出来，并且由于obj文件中已经包含了材质信息，所以我们通过meshLoader中加载出来的就是一个完整的SceneNode了，并且默认的父亲节点就是场景根节点。

MeshLoader组件的大致构成如下所示：

|  |
| --- |
| static std::map<std::string, Mesh\*> meshMap;  static std::map<std::string, Material\*> matMap;  static SceneNode\* LoadMesh(std::string path, bool setDefaultMaterial = true);  static SceneNode\* processNode(aiNode\* aNode, const aiScene\* aScene, std::string directory, bool setDefaultMaterial);  static Mesh\* parseMesh(aiMesh\* aMesh, const aiScene\* aScene, glm::vec3& out\_Min, glm::vec3& out\_Max);  static Material\* parseMaterial(aiMaterial\* aMaterial, const aiScene\* aScene, std::string directory);  static std::string processPath(aiString\* path, std::string directory); |

## Light组件

Light组件实际上是对光源的抽象，目前MogonEngine中支持三种光源，分别是平行光，点光源，聚光。

1. 平行光：

当一个光源位于很远的地方的时候，来自光源的每条光线我们就可以近似地把它们看作是平行的，不论观察者站在任何地方，看起来光线都来自同一个方向。

平行光的一个典型例子就是太阳光，太阳离我们并不是无限远，但我们在计算光照的时候可以把它视作为无限远。

对平行光的抽象如下所示：

|  |
| --- |
| class DirectionalLight  {  public:  glm::vec3 Direction = glm::vec3(1, -1.0, 1);  glm::vec3 Color = glm::vec3(1.0f);  float Intensity = 1.0f;  }; |

主要是光的方向和光的颜色，光的强度是为了让我们调节光的颜色。

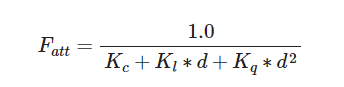
当我们在对平行光进行光照计算的时候，漫反射的时候我们需要会用视线的方向和平面法线的夹角的余弦值乘以光的颜色来作为漫反射的颜色。

图3-3 平行光

1. 点光源

点光源是处于世界中某一个位置的光源，它会朝着所有方向发光，但光线会随着距离逐渐衰减。随距离减少光强度的一种方式是使用一个线性方程，这样的一个方程可以让光的强度线性的随着距离的增长而减少，使得远处的物体要更暗，但这样的方程看起来会比较假。

在现实世界中灯照射时，近处一般会非常亮，但随着距离的增加光源的亮度一开始会下降的特别快，但在远处下降的就会非常缓慢，MongoEngine中使用的是下面这样的一个公式：



其中d表示片段距离光源的距离；

常数项一般保持为1，它的作用是为了让分母永远不会比1小；

一次项与距离值相乘，以线性的方式减少光的强度；

二次项会和距离的平方相乘，让光的强度以二次递减的方式减少。

当我们抽象一个点光源的时候，我们主要关注它的位置和颜色，同样强度参数是为了让我们更方便地控制光的强度。

抽象如下所示：

|  |
| --- |
| class PointLight  {  public:  glm::vec3 Position = glm::vec3(0.0f);  glm::vec3 Color = glm::vec3(1.0f);  float Intensity = 1.0f;  }; |

图3-4 点光源

1. 聚光

聚光是处于世界某个位置的一个光源，它只朝一个特定的方向而不是所有的放心投射光线，所以只有在聚光方向的特定半径内的物体才会被照亮，现实中的聚光例子就是手电筒。

当我们在抽象聚光的时候，我们需要一个空间位置、一个方向、一个切光角。对于一个片段，我们计算它是否在聚光的切光方向上，如果是的话，我们就会照亮相应的片段。

聚光的抽象如下所示：

|  |
| --- |
| class SpotLight  {  public:  glm::vec3 Position = glm::vec3(0, 1, 0);  glm::vec3 Direction = glm::vec3(0, -1, 0);  glm::vec3 Color = glm::vec3(1.0f);  float Intensity = 1.0f;  float CutOff = glm::cos(glm::radians(12.5f));  float OuterCutOff = glm::cos(glm::radians(17.5f));  bool RenderMesh = false;  }; |

图3-5 聚光

## Scene组件

Scene组件是MongoEnigne中的核心组件，其持有了整个场景的场景树，在场景中每个物体都被抽象为一个SceneNode，由Scene持有整个场景树的根节点，并且Scene组件持有所有的光源的引用，场景的节点树结构如下图所示：

图3-6 场景树

当我们实例化SceneNode后，就可以将它加入到场景树中，将它置为场景中某个节点的子物体即可。

深度遍历的核心代码如下：

|  |
| --- |
| void Scene::render(SceneNode\* node)  {  if (node != nullptr)  {  for (int i = 0; i < node->GetChildCount(); i++)  {  if (node->GetChildByIndex(i) != nullptr)  {  node->GetChildByIndex(i)->Render();  }  render(node->GetChildByIndex(i));  }  }  } |

当我们渲染场景时，会以深度优先遍历的方式遍历整个场景树，当访问到每个节点的时候，我们会调用该节点的render接口，渲染该节点。

同时在每个渲染循环中我们都会先更新Uniform缓冲区张的数据，然后渲染每个节点，渲染灯光，最后渲染天空盒。

Scene组件作为一个全局单例类，负责管理场景中所有的实体，包括物体节点，灯光，相机，声音播放工具。

1. 实现效果
2. 总结

总结

毕业设计对于我们大学生来说至关重要，它是大学期间最后的一个环节也是意义重大的一个环节,要做好一份软件的毕业设计，需要紧紧地抓住当今的热门趋势，乐于学习新的计算机以及互联网的前沿知识。当今随着计算机技术发展的迅速趋势，人脸识别技术在当今计算机领域中越来越广泛地引起关注，它也一步步在生活中投入了使用。因此人脸识别这个热门的课题对于我们这群还未踏入社会的学生来说，非常具有挑战性，研究的意义也是很重大的。

此次的人脸识别系统毕业设计让我得到了很多感悟，当我选择了人脸识别这个有意义的课题后，指导老师程老师就给我提供了有用的思路和想法，这之后的一个多月里，我就自己一个人学习理解着相应的人脸识别理论，并最终开发实现了要做的人脸识别功能。这次的人脸识别毕业设计不仅是我大学所学习的专业基础知识的检验，也培养了我的动手解决问题能力，综合检验了我的大学实践能力，我学习着如何在遇到一个难题时，去寻找思路，去设计实施，并最终解决整个问题。在此次开发人脸识别软件之前，我搜索了大量关于机器视觉识别的网页资源和书籍，查找了别人所写的很多人脸识别论文资料以及源程序，了解了特征值以及很多的图像处理算法，率先完成了大体的设计计划。在设计开发的过程中，结合了各种资料中的算法并加入了自己的一些思想，也学习了如何优化代码，另外我还在Visual Studio开发平台上学习了如何使用一系列的组件，最后开发出一个合格的软件程序，可以实现对视频中的人脸进行实时的简单识别，这样的实践经历将会成为我未来工作能力的一部分。

这次设计出来的人脸识别系统存在着以下一些不足和可改进之处：

（1）软件会将第一次检测的陌生人识别为已识别过的面孔，这是由于PCA特征脸方法的限制造成的：因为特征脸方法是使用整体方法（直接使用图像的全部像素）来进行人脸识别的，这样是把整个图像看成了一个高维向量，但当每个人都只有一张图像时，子空间的协方差估计方法会发生错误，所以人脸识别也出错了。这样的缺点可以通过使用其他的算法比如LBP方法解决，这个方法注重于提取图像局部特征，通过这种方法，可获得一个低维隐式，这样最后就不会再出现误将陌生人识别为熟人的情况。

（2）在人脸检测的过程中，人脸面向摄像头的角度和距离是有限制的：要想检测出人脸，脸部必须要正对摄像头，同时不能距离摄像头太远。当人脸角度为半侧以及全侧时，软件无法检测人脸，这是因为本次设计使用的算法没有使用对应的倾角检测特征，而只有正面检测特征。

（3）软件会受到光照条件的干扰和影响，尤其在光线较弱的地方，很有可能会导致摄像头中脸部的检测以及识别失败，因此本系统的识别并不稳定，在不同光照条件下识别出的人脸信息可能会出错，这也是因为PCA方法有不足性：PCA方法提取的是图像的总体特征，这种方法保留了图像中的所有信息，而当外界的光线和光照出现一定变化时，识别的性能会受到重大影响，识别能力会大大降低。

（4）在进行软件人脸识别训练过程中，最好要对一个人进行多次训练，因为PCA方法在训练样本足够多的情况下，测试样本与训练样本的相关性会更高，识别测试时会更加精确。

这次的毕业设计让我认识到，我在大学期间积累的专业知识还是不足，在任何时候都不能只依赖着以前学过的书本知识，现在自己还有太多的东西不了解，我的专业水平也还没达到高水准。我明白了学习需要靠自己长期的积累，在以后的工作中也应该不断地学习和总结，将学习到的理论知识与开发设计里的动手实践经验相结合，不断提高自己的专业水平和实践能力。

参考文献

[1]徐安东，谭浩强.Visual C#程序设计基础[M].北京：清华大学出版社，2012.1:20-40.

[2]Karli Watson,Jacob Vibe Hammer,Jon D Reid,et al.C#入门经典（第6版）[M].齐立波，等，译.北京：清华大学出版社，2014.1:103-126.

[3]John Sharp.Visual C# 2010从入门到精通:Step by Step[M].周靖，译.北京：清华大学出版社，2010.8:117-151.

[4]Daniel M Solis.C#图解教程（第4版）[M].姚琪琳，等，译.北京：人民邮电出版社，2013.7:36-82.

[5]于仕琪，刘瑞祯.OpenCV教程：基础篇[M].北京：北京航空航天大学出版社，2007.6:63-85.

[6]Bradski G,Kaehler A.学习OpenCV（中文版）[M].于仕琪，译.北京：清华大学出版社，2009.10:18-125.

[7]Gonzalez R C,Woods R E.数字图像处理（第3版）[M]. 阮秋琦，等，译.北京：电子工业出版社，2011.5:62-285.

[8]J R Parker.图像处理与计算机视觉算法及应用（第2版）[M].景丽，译.北京：清华大学出版社，2012.5:215-230.

[9]Mark S Nixon,Alberto S Aguado.特征提取与图像处理（第2版）[M].李仁发，译.北京：电子工业出版社，2010.10:147-192.

[10]Milan Sonka,Vaclav Hlavac,Roger Boyle.图像处理、分析与机器视觉（第3版）[M].艾海舟，等，译.北京：清华大学出版社，2011.1:79-113.

[11]Turk M,Pentland A P. Eigenfaces for Recognition[J].Journal of Cognitive Neuroscience,1991,3(1):71-86.

[12]Adam D.Tibbalds. Three Dimensional Human Face Acquisitions for Recognition[D].Phd.Thesis Cambridge,University of Cambridge,1998.

[13]Belhumeur P N,Hespanha J P,Kriegman D J. Eigenfaces vs. Fisherfaces : Recognition Using Class Specific Linear Projection[J].IEEE Trans.On Pattern Analysis and Machine Intelligence,1997(19):711-720.

[14]Yambor W,Draper B,Beveridge J.R.. Analysis of PCA-based Face Recognition Algonthms[R]:Eigenveetor Selection and Distance Measures.Second Workshop on Empirical Evaluation Methods in Computer Vision,2000.

[15]Yang Jian,David Zhang. Two-Dimensional PCA:A New Approach to Appearance-Based Face Representation and Recognition[J].IEEE Trans.on PAMI,2004,226(4):131-137.

[16]WenyiZhao. Robust image based 3D face recognition [D].PhD.Thesis.University of Maryland,College Park,1999.

[17]Comon P. Independent Component Analysis-A New Concept[J].Signal Proeessing,1994,36(3):287-314.

[18]Draper BA,Baek K,Bartlett MS,et al. Recognizing Faces with PCA and ICA[J].Computer Vision and Image Understanding,2003,91(1-2):115-137.

[19]Moghaddam B,Jebara T,Pentland A. Bayesian Face Recognition[J].Pattern Recognition,2000.33(11):1771-1782.

[20]S.C.Chen,Y.L.Zhu. Subpattern-based Principal Component Analysis[J].Pattern Recognition,2004,37(1):1081-1083.

致谢

在这次毕业设计的过程中，我得到了指导老师程老师的精心指导，从一开始选择毕业设计的题目时，程老师就给了我许多珍贵的意见，启发了我，从而让我选择了一个有趣而又新颖的设计项目。在这过去的两个月时光里，衷心感谢程老师对我的帮助和支持。

在大学学习期间，我的老师们也对我给予了很多热情的帮助，在此向他们表达我的诚挚的谢意。还有，我要特别感谢我的舍友干瑞杰，他在我测试人脸识别系统时提供了不少的援助。