南京工程学院

**实 验 报 告**

课程名称 虚拟现实2020

实验项目名称 纹理映射

实验学生班级 数嵌172

实验学生姓名 朱广锋

学　　　　号 202170638

同组学生姓名 无

实验时间 2020/5/31

实验地点

1. 实验主题

利用VC集成开发环境，配置OpenGL库函数， 建立基本的OpenGL应用程序，完成实验四的任务。

1. 实验准备

1.打开Visual Studio并且设置好工作目录；

2.下载OpenGL安装包所需文件（http://d.download.csdn.net/down/2560229/ssagnn23），主要包括GL.H GLAUX.H GLU.H glut.h

GLAUX.LIB GLU32.LIB glut32.lib glut.lib OPENGL32.LIB

glaux.dll glu32.dll glut32.dll glut.dll opengl32.dll

3.复制并配置OpenGL库函数到指定的目录（.h、.lib、.dll分别放到MSVC include、lib和系统Path路径如System32），检查复制后是否文件已经存在于指定目录下。

1. 主要数据源、库函数、变量、涉及函数及其解释

|  |
| --- |
| // Textbind.cpp  #pragma comment(lib, "legacy\_stdio\_definitions.lib ")  #include <GL/glut.h>  #include <GL/GLAUX.H>  #include <cstdlib>  constexpr int checkImageWidth = 64;  constexpr int checkImageHeight = 64;  constexpr int flyImageWidth = 128;  constexpr int flyImageHeight = 128;  static GLubyte checkImage[checkImageHeight][checkImageWidth][4];  static GLubyte otherImage[checkImageHeight][checkImageWidth][4];  static GLubyte flyImage[flyImageHeight][flyImageWidth][4];  static GLuint texName[4];  #define USE\_FLY\_TEXTURE true  #define USE\_PIC\_TEXTURE false  /// <summary>  /// 32X32 bits  /// </summary>  GLubyte fly[] = {  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  0x03, 0x80, 0x01, 0xC0, 0x06, 0xC0, 0x03, 0x60,  0x04, 0x60, 0x06, 0x20, 0x04, 0x30, 0x0C, 0x20,  0x04, 0x18, 0x18, 0x20, 0x04, 0x0C, 0x30, 0x20,  0x04, 0x06, 0x60, 0x20, 0x44, 0x03, 0xC0, 0x22,  0x44, 0x01, 0x80, 0x22, 0x44, 0x01, 0x80, 0x22,  0x44, 0x01, 0x80, 0x22, 0x44, 0x01, 0x80, 0x22,  0x44, 0x01, 0x80, 0x22, 0x44, 0x01, 0x80, 0x22,  0x66, 0x01, 0x80, 0x66, 0x33, 0x01, 0x80, 0xCC,  0x19, 0x81, 0x81, 0x98, 0x0C, 0xC1, 0x83, 0x30,  0x07, 0xe1, 0x87, 0xe0, 0x03, 0x3f, 0xfc, 0xc0,  0x03, 0x31, 0x8c, 0xc0, 0x03, 0x33, 0xcc, 0xc0,  0x06, 0x64, 0x26, 0x60, 0x0c, 0xcc, 0x33, 0x30,  0x18, 0xcc, 0x33, 0x18, 0x10, 0xc4, 0x23, 0x08,  0x10, 0x63, 0xC6, 0x08, 0x10, 0x30, 0x0c, 0x08,  0x10, 0x18, 0x18, 0x08, 0x10, 0x00, 0x00, 0x08 };  /// <summary>  /// 生成苍蝇纹理  /// </summary>  void makeFlyImages(void)  {  for (int i = 0; i < flyImageWidth; i++) {  for (int j = 0; j < flyImageHeight; j++) {  auto offset = i % 32 \* 32 + j % 32;  \*(GLuint\*)&flyImage[i][j] = fly[offset / 8] & (1 << (7-(offset % 8))) ? 0XFFFFFFFF : 0XFF000000;  }  }  }  void makeCheckImages(void)  {  for (int i = 0; i < checkImageHeight; i++) {  for (int j = 0; j < checkImageWidth; j++) {  int c = ((((i & 0x8) == 0) ^ ((j & 0x8)) == 0)) \* 255;  checkImage[i][j][0] = (GLubyte)c;  checkImage[i][j][1] = (GLubyte)c;  checkImage[i][j][2] = (GLubyte)c;  checkImage[i][j][3] = (GLubyte)255;  c = ((((i & 0x10) == 0) ^ ((j & 0x10)) == 0)) \* 255;  otherImage[i][j][0] = (GLubyte)c;  otherImage[i][j][1] = (GLubyte)0;  otherImage[i][j][2] = (GLubyte)0;  otherImage[i][j][3] = (GLubyte)255;  }  }  }  void init(void)  {  glClearColor(0.0, 0.0, 0.0, 0.0);  glShadeModel(GL\_FLAT);  glEnable(GL\_DEPTH\_TEST);  makeFlyImages();  makeCheckImages();    glPixelStorei(GL\_UNPACK\_ALIGNMENT, 1);  glGenTextures(3, texName);  glBindTexture(GL\_TEXTURE\_2D, texName[0]);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_CLAMP); glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_CLAMP);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_NEAREST); glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_NEAREST);  glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA, checkImageWidth,  checkImageHeight, 0, GL\_RGBA, GL\_UNSIGNED\_BYTE,  checkImage);  glBindTexture(GL\_TEXTURE\_2D, texName[1]);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_CLAMP);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_CLAMP);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_NEAREST);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_NEAREST);  glTexEnvf(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_DECAL);  glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA, checkImageWidth,  checkImageHeight, 0, GL\_RGBA, GL\_UNSIGNED\_BYTE,  otherImage);  glBindTexture(GL\_TEXTURE\_2D, texName[2]);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_NEAREST);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_NEAREST);  //glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);  //glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_CLAMP);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_CLAMP);  //glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S,GL\_CLAMP\_TO\_EDGE);  //glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T,GL\_CLAMP\_TO\_EDGE);  //glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S,GL\_CLAMP\_TO\_BORDER);  //glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T,GL\_CLAMP\_TO\_BORDER);  //glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT);  //glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT);  //glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_MIRRORED\_REPEAT);  //glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_MIRRORED\_REPEAT);  glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA, flyImageWidth,  flyImageHeight, 0, GL\_RGBA, GL\_UNSIGNED\_BYTE,  flyImage);  auto texture = auxDIBImageLoad(TEXT("meme.bmp"));  glBindTexture(GL\_TEXTURE\_2D, texName[3]);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_CLAMP);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_CLAMP);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_NEAREST);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_NEAREST);  glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGB, texture->sizeX,  texture->sizeY, 0, GL\_RGB, GL\_UNSIGNED\_BYTE,  texture->data);  glEnable(GL\_TEXTURE\_2D);  }  void display(void)  {  glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);  glBindTexture(GL\_TEXTURE\_2D, texName[USE\_FLY\_TEXTURE ? 2 : 0]);  glBegin(GL\_QUADS);  glTexCoord2f(0.0, 0.0); glVertex3f(-2.0, -1.0, 0.0);  glTexCoord2f(0.0, 1.0); glVertex3f(-2.0, 1.0, 0.0);  glTexCoord2f(1.0, 1.0); glVertex3f(0.0, 1.0, 0.0);  glTexCoord2f(1.0, 0.0); glVertex3f(0.0, -1.0, 0.0);  glEnd();  glBindTexture(GL\_TEXTURE\_2D, texName[USE\_FLY\_TEXTURE ? 2 : USE\_PIC\_TEXTURE ? 3 : 1]);  glBegin(GL\_QUADS);  glTexCoord2f(0.0, 0.0); glVertex3f(0.0, -1.0, 0.0);  glTexCoord2f(0.0, 1.0); glVertex3f(0.0, 1.0, 0.0);  glTexCoord2f(1.0, 1.0); glVertex3f(1.41421, 1.0, -1.41421);  glTexCoord2f(1.0, 0.0); glVertex3f(1.41421, -1.0, -1.41421);  glEnd();  glFlush();  }  void reshape(int w, int h)  {  glViewport(0, 0, (GLsizei)w, (GLsizei)h);  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  gluPerspective(60.0, (GLfloat)w / (GLfloat)h, 1.0, 30.0);  glMatrixMode(GL\_MODELVIEW);  glLoadIdentity();  glTranslatef(0.0, 0.0, -3.6);  }  void keyboard(unsigned char key, int x, int y)  {  switch (key) {  case 27:  exit(0);  break;  default:  break;  }  }  int main(int argc, char\*\* argv)  {  glutInit(&argc, argv);  glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB | GLUT\_DEPTH);  glutInitWindowSize(500, 500);  glutInitWindowPosition(100, 100);  glutCreateWindow(argv[0]);  init();  glutDisplayFunc(display);  glutReshapeFunc(reshape);  glutKeyboardFunc(keyboard);  glutMainLoop();  return 0;  } |
| 添加代码段含义：  makeFlyImages设置苍蝇纹理  函数解释：  glGenTextures 生成纹理，根据参数设置n个纹理索引；  glBindTexture 将纹理索引绑定到目标上，第一个参数GL\_TEXTURE\_2D表示2D纹理；  glTexParameteri 纹理过滤函数，target GL\_TEXTURE\_2D指定操作2D纹理，pname表示操作名称， param表示值，GL\_TEXTURE\_WRAP\_S为S方向上的贴图模式，GL\_CLAMP将纹理坐标限制在0.0~1.0范围，超出部分会边缘拉伸填充，GL\_TEXTURE\_WRAP\_T设置T方向上的贴图模式，GL\_TEXTURE\_MAG\_FILTER设置放大过滤的规则，GL\_TEXTURE\_MIN\_FILTER设置缩小过滤的规则，GL\_NEAREST代表临近插值；  glTexImage2D 生成二维纹理图像，参数依次为目标target、详细程度编号level、纹理内部格式internalformat、纹理图像宽度width、高度height、边框高度border、纹理数据格式format、纹理数据类型type、指向图像数据指针pixels，internalformat和format设置为GL\_RGBA表示24色彩，type设置为GL\_UNSIGNED\_BYTE与pixels参数checkImage、otherImage、flyImage类型对应；  glTexEnvf(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_DECAL)指定纹理贴图和材质的混合方式为GL\_DECAL；  glEnable(GL\_TEXTURE\_2D) 启用2D纹理；  glBegin(GL\_QUADS) 绘制四边形；  glTexCoord2f 载入纹理，参数为ST坐标。 |

1. 实验任务

任务1：建立纹理映射应用程序

 在VSStudio 菜单栏上选择File->New->Project，选择Win32应用程序，输入项目名称TextureProj

 点击“OK”后，在后续的对话框中选中Empty project，然后点击Finish。

 右击项目名TextureProj，选择属性(property)，再选择链接器(Linker)中的输入选项(Input)，附加依赖项(Additional Dependencies):opengl32.lib glu32.lib glaux.lib

 右击项目名TextureProj下的源文件，选择添加(ADD)，再选择新建项 (New Item)，在弹出来的对话框中选择C++ File(.cpp)，输入文件名Texbind.cpp,然后点击添加(ADD)，检查源文件是否建立成功。

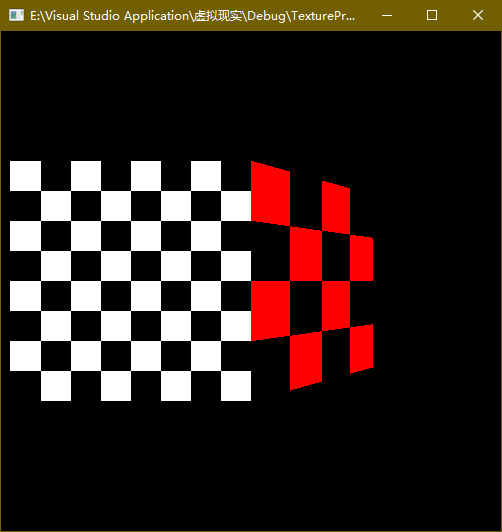
任务2：在Texbind.cpp中输入代码，人工生成一类棋盘格图像checkImage，读取一个外部图像otherImage, 然后作为纹理贴图贴到两个正方形两个表面。

任务3：在Texbind.cpp中输入代码，人工生成一苍蝇图像FlyImage，映射到一个正方形上，并设置纹理的重复与截取方式。

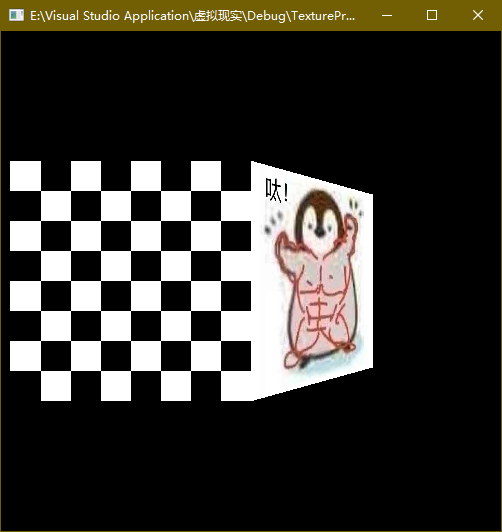
1. 主要分析和解释

运行结果：

任务1 如图为使用生成纹理的绘制结果；



任务2 如图读取外部图像纹理的绘制结果；



 归纳纹理映射的四个步骤，以及总结四个步骤各自相关的对应函数？用框表的形式表达出来。

|  |  |
| --- | --- |
| 生成或加载纹理 | makeCheckImages、auxDIBImageLoad |
| 绑定纹理对象 | glBindTexture |
| 纹理过滤，指定纹理和材质混合方式 | glTexParameteri、glTexEnvf |
| 绘制场景,提供纹理和几何坐标 | glTexImage2D、glTexCoord2f |

 为何glBindTexture函数对每一个纹理对象都用了两次，每次的含义是什么？用两次的作用是什么？

第一次初始化使用glBindTexture()函数绑定纹理，并确定纹理类型为2D；

显示阶段，使用glBindTexture()将纹理绑定至显示上下文，与顶点一起进入纹理贴图阶段。

 尝试解释void makeCheckImages(void)函数中每个语句的含义：

遍历checkImage和otherImage的每个元素，((((i & 0x8) == 0) ^ ((j & 0x8)) == 0))表示格子宽高为8 ，最终生成8\*8数量、8\*8大小的棋盘。生成otherImage的方法类似，只是格子宽高变为16。

任务3 如图为生成苍蝇图像的绘制结果。

|  |  |
| --- | --- |
|  | GL\_NEAREST  GL\_CLAMP |
|  | GL\_LINEAR  GL\_CLAMP |
|  | GL\_LINEAR  GL\_REPEAT |
|  | GL\_NEAREST  GL\_REPEAT |

1. 展望

本次实验， 主要实践了Opengl纹理映射的基本步骤，让我理解了纹理的生成与加载方法，纹理的绑定方法，纹理过滤与混合的方法，以及绘制时载入纹理的步骤。

实验中曾遇到一些问题，如设置苍蝇纹理时没有注意下标的范围导致的越界异常、加载了非DIB纹理图像的错误等，不过都在得到报错提示后修改正确。

教师评阅：

|  |  |
| --- | --- |
| 评阅项目及内容 | 得分 |
| 1．考勤（10分） |  |
| 2．实验完成情况（50分） |  |
| 3．报告撰写内容（40分） |  |
| 合 计 |  |
| 成绩评定 |  |