上海电力学院

虚拟现实技术

课程设计报告

题目 3D 人脸渲染

姓名 高健翔

专业 电子信息工程

班级 2015071

学号 20131921

一、目的

（1）熟悉 Opengl 工具

（2）学会在 linux 中运行 OpenGL 示例

（3）了解 OpenGL 并用自己的图片渲染 3D 人脸模型

二、环境

Deepin

Sublime text3

三、实现主要代码

```
#define WindowWidth   800
#define WindowHeight 800
#define WindowTitle   "OpenGL 纹理测试"

#include <GL/glut.h>
#include <stdio.h>
#include <stdlib.h>

//定义两个纹理对象编号
//GLuint texGround;
GLuint texWall;

#define BMP_Header_Length 54   //图像数据在内存块中的偏移量
//static GLfloat angle = 0.0f;     //旋转角度
//GLfloat diffuseMaterial[4] = { 0.5, 0.5, 0.5, 1.0 };

int s1=0,s2=0;
// 函数 power_of_two 用于判断一个整数是不是 2 的整数次幂
int power_of_two(int n)
{
     if( n <= 0 )
          return 0;
     return (n & (n-1)) == 0;
}

/* 函数 load_texture
* 读取一个 BMP 文件作为纹理
* 如果失败，返回 0，如果成功，返回纹理编号
*/
GLuint load_texture(const char* file_name)
{
     GLint width, height, total_bytes;
     GLubyte* pixels = 0;
     GLuint last_texture_ID=0, texture_ID = 0;
```

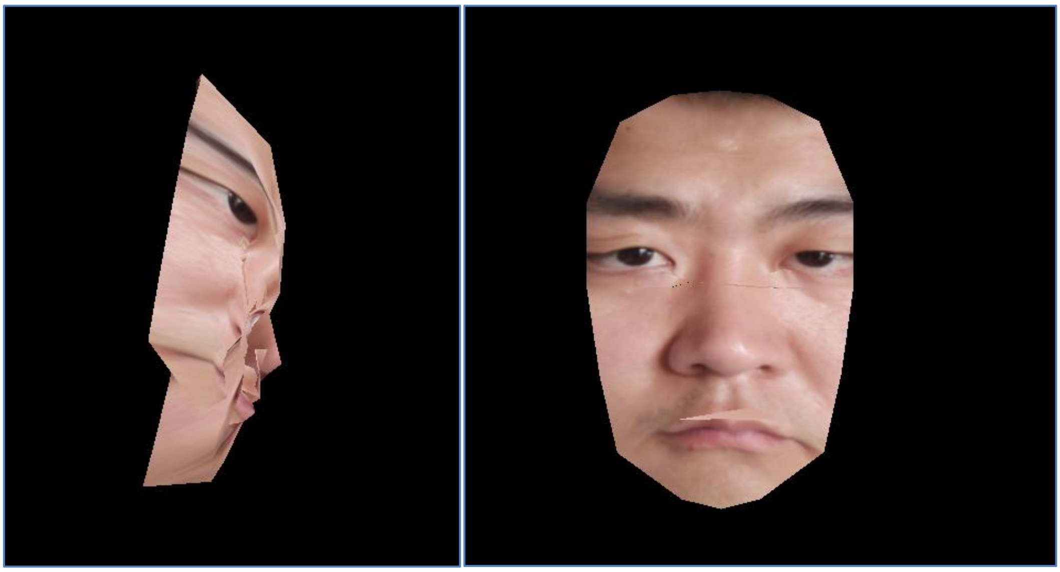
```
// 打开文件，如果失败，返回
FILE* pFile = fopen(file_name, "rb");
if( pFile == 0 )
    return 0;

// 读取文件中图象的宽度和高度
fseek(pFile, 0x0012, SEEK_SET);
fread(&width, 4, 1, pFile);
fread(&height, 4, 1, pFile);
fseek(pFile, BMP_Header_Length, SEEK_SET);

// 计算每行像素所占字节数，并根据此数据计算总像素字节数
{
    GLint line_bytes = width * 3;
    while( line_bytes % 4 != 0 )
        ++line_bytes;
    total_bytes = line_bytes * height;
}

// 根据总像素字节数分配内存
pixels = (GLubyte*)malloc(total_bytes);
if( pixels == 0 )
{
    fclose(pFile);
    return 0;
}

// 读取像素数据
if( fread(pixels, total_bytes, 1, pFile) <= 0 )
{
    free(pixels);
    fclose(pFile);
    return 0;
}

// 对就旧版本的兼容，如果图象的宽度和高度不是的整数次方，则需要进行缩放
// 若图像宽高超过了 OpenGL 规定的最大值，也缩放
{
    GLint max;
    glGetIntegerv(GL_MAX_TEXTURE_SIZE, &max);
    if( !power_of_two(width)
        || !power_of_two(height)
        || width > max
        || height > max )
    {
        const GLint new_width = 256;
        const GLint new_height = 256; // 规定缩放后新的大小为边长的正方
形

        GLint new_line_bytes, new_total_bytes;
        GLubyte* new_pixels = 0;

        // 计算每行需要的字节数和总字节数
        new_line_bytes = new_width * 3;
        while( new_line_bytes % 4 != 0 )
```

```
                    ++new_line_bytes;
                new_total_bytes = new_line_bytes * new_height;

                // 分配内存
                new_pixels = (GLubyte*)malloc(new_total_bytes);
                if( new_pixels == 0 )
                {
                    free(pixels);
                    fclose(pFile);
                    return 0;
                }

                // 进行像素缩放
                gluScaleImage(GL_RGB,
                    width, height, GL_UNSIGNED_BYTE, pixels,
                    new_width, new_height, GL_UNSIGNED_BYTE, new_pixels);

                // 释放原来的像素数据，把 pixels 指向新的像素数据，并重新设置 width
和 height
                free(pixels);
                pixels = new_pixels;
                width = new_width;
                height = new_height;
            }
        }

        // 分配一个新的纹理编号
        glGenTextures(1, &texture_ID);
        if( texture_ID == 0 )
        {
            free(pixels);
            fclose(pFile);
            return 0;
        }

        // 绑定新的纹理，载入纹理并设置纹理参数
        // 在绑定前，先获得原来绑定的纹理编号，以便在最后进行恢复
        GLint lastTextureID=last_texture_ID;
        glGetIntegerv(GL_TEXTURE_BINDING_2D, &lastTextureID);
        glBindTexture(GL_TEXTURE_2D, texture_ID);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
        glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0,
            GL_BGR_EXT, GL_UNSIGNED_BYTE, pixels);
        glBindTexture(GL_TEXTURE_2D, lastTextureID);  //恢复之前的纹理绑定
        free(pixels);
        return texture_ID;
    }
    void keyboard (unsigned char key, int x, int y)
```

```
{
switch (key) {
case 's':
s1 = (s1 + 5) % 360;
glutPostRedisplay();
break;
case 'S':
s1 = (s1 - 5) % 360;
glutPostRedisplay();
break;
case 'e':
s2 = (s2 + 5) % 360;
glutPostRedisplay();
break;
case 'E':
s2 = (s2 - 5) % 360;
glutPostRedisplay();
break;
case 27:
exit(0);
break;
default:
break;
}
}
static GLfloat spin = 0.0;
void init(void)
{
glClearColor(0.0,0.0,0.0,0.0);
glShadeModel(GL_FLAT);
}
void display(void)
{
    // 清除屏幕
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    // 设置视角
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
        //gluPerspective(75, 1, 1, 21);
       gluPerspective(20, 1, 3, 21);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(-4, 7,7, 0, 0, 0, 0, 0, 1);
    //glRotatef(angle, 0.0f, 0.0f, 1.0f); //旋转
        //glRotatef(spin,0.0,0.0,0.0);
        glRotatef(s1,0.0,0.0,1.0);
        glRotatef(s2,1.0,0.0,0.0);
        glBindTexture(GL_TEXTURE_2D, texWall);
       //glBegin(GL_QUADS);
      glBegin(GL_TRIANGLES);

/*glClear(GL_COLOR_BUFFER_BIT);
glPushMatrix();glRotatef(spin,0.0,1.0,0.0);
glColor3f(0.5,0.0,1.0);
glPolygonMode(GL_FRONT_AND_BACK,GL_FRONT);
glBindTexture(GL_TEXTURE_2D, texWall);
```

```
glBegin(GL_TRIANGLES);*/
    //  绘制底面以及纹理
glTexCoord2d(0.501041,0.005000);glVertex3f(0.000000f,1.061000f,-0.371000f);
glTexCoord2d(0.587917,0.079375);glVertex3f(0.174000f,0.800000f,-0.024000f);
glTexCoord2d(0.600208,0.014375);glVertex3f(0.217000f,1.039000f,-0.371000f);
glTexCoord2d(0.501041,0.005000);glVertex3f(0.000000f,1.061000f,-0.371000f);
glTexCoord2d(0.587917,0.079375);glVertex3f(0.174000f,0.800000f,-0.024000f);


int main(int argc, char* argv[])
{
    // GLUT 初始化
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA);
    glutInitWindowPosition(20, 20);
    glutInitWindowSize(WindowWidth, WindowHeight);
    glutCreateWindow(WindowTitle);
    glEnable(GL_DEPTH_TEST);

    glEnable(GL_TEXTURE_2D);      //  启用纹理
    //texGround = load_texture("2.bmp");   //加载纹理
    //texWall = load_texture("lby.bmp");
       texWall = load_texture("zhengmian.bmp");
    glutDisplayFunc(&display);     //注册函数
        glutReshapeFunc(reshape);
        glutMouseFunc(mouse);
        glutKeyboardFunc(keyboard);
    //glutIdleFunc(&myIdle);
    glutMainLoop(); //循环调用
    return 0;
}
```

四、效果



五、总结

本次课程设计，入门和了解 OpenGL 工具并使用 OpenGL 完成一些简单到复杂的图像设计和处理。这学期最大的收获是，感谢仝老师给了很多方面的指导和指引，以及对学习和未来职业规划上思维的扩展，更好地利用 github 上面丰富的开源资源，在工作学习中做更深入的研究。