浙江大学

计算机图形学作业报告

作业名称：	纹理映射
姓　　名：	郭炅
学　　号：	3170105370
电子邮箱：	793881807@qq.com
联系电话：	18888923171
指导老师：	张宏鑫

2019 年 12 月 3 日

纹理映射

一、 作业已实现的功能简述及运行简要说明

1. 功能简述

- 读取自绘制图片，在茶壶表面绘制自己学号最后两位
- 对桌面单一使用 Crack.bmp 的纹理效果
- 对桌面实现 Crack.bmp 和 Spot.bmp 的叠加效果
- 可以效果展示进行键盘控制

2. 运行简要说明

- 运行 Visual Studio 2017 工程即可

二、 作业的开发与运行环境

- Windows10
- Visual Studio2017
- Glut
- Glew

三、 系统或算法的基本思路、原理、及流程或步骤等

1. 基本思路

- 加载 bmp 位图，根据读入的数据生成 Texture 并存储
- 初始化 ARB 多重纹理属性
- 设置 ARB 纹理，配置不同的光照环境
- 绑定纹理，绘制图形，设置张贴纹理的坐标映射
- 在图形的坐标变化中设置可变参数
- 设置键盘操作对应的参数变化，以实现键盘操作

2. 步骤及原理

- 首先加载 bmp 图形并转换为纹理
- 设置纹理属性，绑定纹理，绘制图形
- 配置键盘操作

四、 具体如何实现，包括关键（伪）代码、主要用到函数与算法等

1. Bmp 图片数据读取

```
filePtr = fopen(filename, "rb");
```

- 使用 fopen 函数打开文件

```
// 读入 bitmap 文件图
fread(&bitmapFileHeader, sizeof(BITMAPFILEHEADER), 1, filePtr);
```

- 读入该文件的头部信息

```
// 读入 bitmap 信息头
fread(bitmapInfoHeader, sizeof(BITMAPINFOHEADER), 1, filePtr);
// 将文件指针移至 bitmap 数据
fseek(filePtr, bitmapFileHeader.bfOffBits, SEEK_SET);
// 为装载图像数据创建足够的内存
bitmapImage = new unsigned char[bitmapInfoHeader->biSizeImage];
//Size followed by number of items.That should work OK then!
fread(bitmapImage, bitmapInfoHeader->biSizeImage, 1, filePtr);
```

- 判断是 bmp 图片后，读取其头部信息块
- 将文件指针移动到 bmp 数据块区
- 读取 bmp 文件实际的数据内容，存储至数组 *bitmaoImage*

```
//由于 bitmap 中保存的格式是 BGR，下面交换 R 和 B 的值，得到 RGB 格式
for (imageIdx = 0;imageIdx < bitmapInfoHeader->biSizeImage; imageIdx += 3)
{
    tempRGB = bitmapImage[imageIdx];
    bitmapImage[imageIdx] = bitmapImage[imageIdx + 2];
    bitmapImage[imageIdx + 2] = tempRGB;
}
```

- 交换 RGB 通道数据

2. 纹理加载

```
glBindTexture(GL_TEXTURE_2D, texture[i]);
// 指定当前纹理的放大/缩小过滤方式
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_WRAP_S,GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,GL_REPEAT);

glTexImage2D(GL_TEXTURE_2D,
             0,                         //mipmap 层次(通常为，表示最上层)
             GL_RGB,                    //我们希望该纹理有红、绿、蓝数据
             bitmapInfoHeader.biWidth,  //纹理宽带，必须是 n，若有边框+2
```

```
                bitmapInfoHeader.biHeight, //纹理高度，必须是n，若有边框+2
                0,                         //边框(0=无边框，1=有边框)
                GL_RGB,                    //bitmap 数据的格式
                GL_UNSIGNED_BYTE,          //每个颜色数据的类型
                bitmapData);               //bitmap 数据指针
```

- 将图片转换为纹理，绑定纹理标号至 texture 数组，方便接下来的调用

3. 初始化 ARB

```
    const GLubyte *extensions = glGetString(GL_EXTENSIONS);
    bool multiTexturingSupported = strstr((const char *)extensions, "GL_ARB
_multitexture ") != NULL;

    GLint maxTextureUnits = 0;
    glGetIntegerv(GL_MAX_TEXTURE_UNITS_ARB, &maxTextureUnits);
    printf("Texture Units available = %d\n", maxTextureUnits);

    glMultiTexCoord1fARB = (PFNGLMULTITEXCOORD1FARBPROC)wglGetProcAddress("
glMultiTexCoord1fARB");
    glMultiTexCoord2fARB = (PFNGLMULTITEXCOORD2FARBPROC)wglGetProcAddress("
glMultiTexCoord2fARB");
    glMultiTexCoord3fARB = (PFNGLMULTITEXCOORD3FARBPROC)wglGetProcAddress("
glMultiTexCoord3fARB");
    glMultiTexCoord4fARB = (PFNGLMULTITEXCOORD4FARBPROC)wglGetProcAddress("
glMultiTexCoord4fARB");
    glActiveTextureARB = (PFNGLACTIVETEXTUREARBPROC)wglGetProcAddress("glAc
tiveTextureARB");
    glClientActiveTextureARB = (PFNGLCLIENTACTIVETEXTUREARBPROC)wglGetProcA
ddress("glClientActiveTextureARB");
```

- 确定限定支持的拓展
- 如果支持 ARB，则加载 ARB 拓展，允许使用线管函数

4. 设置纹理属性

```
       glEnable(GL_TEXTURE_2D);
        glBindTexture(GL_TEXTURE_2D,texture[2]);
        glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
        glPushMatrix();
        // glTranslatef(0,0,1);
        glRotatef(90,1,0,0);
        glutSolidTeapot(1);
        glPopMatrix();
        glDisable(GL_TEXTURE_2D);
```

- 对茶壶只需要设定需要绑定的相关纹理即可

```
        glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE); //设置
纹理受光照影响
        glActiveTextureARB(GL_TEXTURE0_ARB);
        glEnable(GL_TEXTURE_2D);
        glBindTexture(GL_TEXTURE_2D, texture[0]);

        if(mode==3){
            glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL); //设
置纹理受光照影响
            glActiveTextureARB(GL_TEXTURE1_ARB);
            glEnable(GL_TEXTURE_2D);
            glBindTexture(GL_TEXTURE_2D, texture[1]);
        }
```

- 绘制桌子时，根据是否叠加纹理设定第二种纹理
- 原始的桌面纹理放置于 TEXTURE0 通道，叠加的阴影纹理放置于 TEXTURE1 通道，同时使用 glTexENvf() 函数设定纹理与环境，纹理于纹理之间的渲染叠加方式，对两种纹理分别使用上述的属性，即可以实现要求的效果。
- 在所有工作进行前需要激活纹理
- 绑定纹理坐标绘制结束后需要关闭纹理

5. 图形绘制

```
        glBegin(GL_QUADS);
        glMultiTexCoord2fARB(GL_TEXTURE0, 1, 1);
        glMultiTexCoord2fARB(GL_TEXTURE1, 1, 1);
        glVertex3f(0.5f, 0.5f, 0.5f);
        glMultiTexCoord2fARB(GL_TEXTURE0, 1, 0);
        glMultiTexCoord2fARB(GL_TEXTURE1, 1, 0);
        glVertex3f(0.5f, -0.5f, 0.5f);
        glMultiTexCoord2fARB(GL_TEXTURE0, 0, 0);
        glMultiTexCoord2fARB(GL_TEXTURE1, 0, 0);
        glVertex3f(-0.5f, -0.5f, 0.5f);
        glMultiTexCoord2fARB(GL_TEXTURE0, 0, 1);
        glMultiTexCoord2fARB(GL_TEXTURE1, 0, 1);
        glVertex3f(-0.5f, 0.5f, 0.5f);
        glEnd();
```

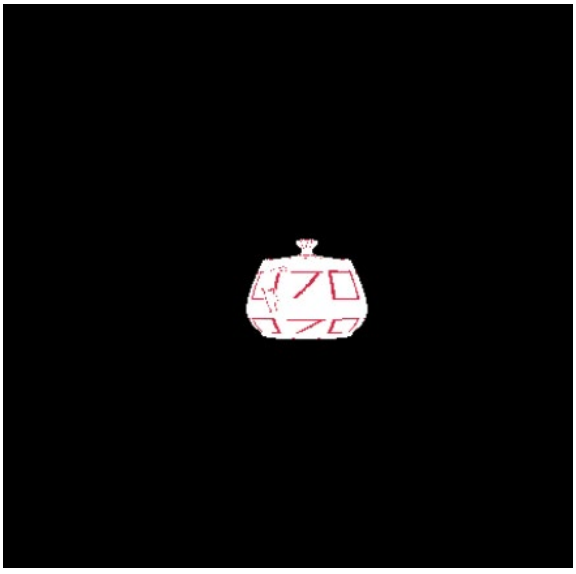
- 使用六个四边形面片组合形成完整的立方体进行显示，同时需要对两种纹理的坐标都进行绑定

```
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
        glPushMatrix();
        glTranslatef(0, 0, 3.5);
```

```
        glScalef(5, 4, 1);
        Texture_desk();
        glPopMatrix();

        glPushMatrix();
        glTranslatef(1.5, 1, 1.5);
        Draw_Leg();
        glPopMatrix();

        glPushMatrix();
        glTranslatef(-1.5, 1, 1.5);
        Draw_Leg();
        glPopMatrix();

        glPushMatrix();
        glTranslatef(1.5, -1, 1.5);
        Draw_Leg();
        glPopMatrix();

        glPushMatrix();
        glTranslatef(-1.5, -1, 1.5);
        Draw_Leg();
        glPopMatrix();
```

- 实际的绘制需要对标准立方体进行变形，组合产生桌子

五、 实验结果与分析



- 茶壶表面贴上了我的学号纹理，并且可以根据键盘操作旋转

- 按动键盘 2，切换至桌子，可以看到桌子贴上了相关的纹理

- 安东键盘 3，切换纹理，此时时两个纹理的叠加效果，可以看到在原有的桌面的纹理上贴上了相关的纹理图案

六、 结论与心得体会

- 学到了如何使用纹理贴图构建复杂的图案，从而使得绘制出的图形更加美丽

七、 参考文献