浙江大学实验报告

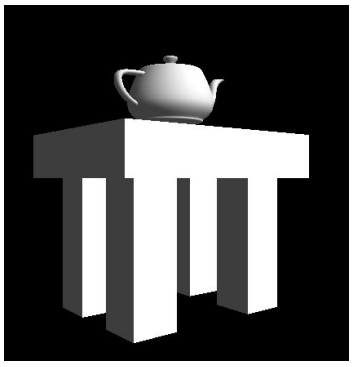
课程名称：____计算机图形学____ 指导老师：_____成绩：_____
实验名称：____OpenGL 纹理_____ 实验类型：_____基础实验_____同组学生姓名：_____

一、实验目的和要求

在 OpenGL 消隐和光照实验的基础上，通过实现实验内容，掌握 OpenGL 中纹理的使用，并验证课程中关于纹理的内容。

二、实验内容和原理

使用 Visual Studio C++编译已有项目工程。



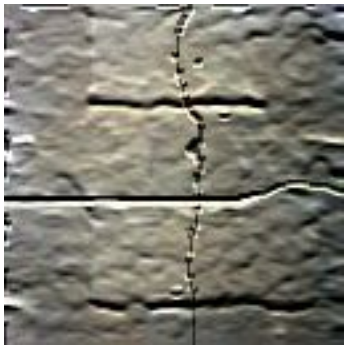
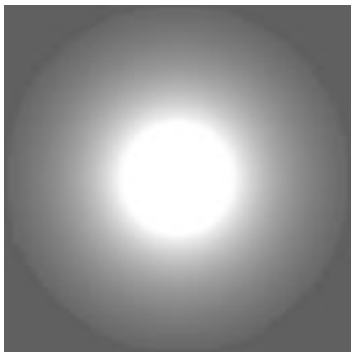
模型尺寸参见 OpenGL 消隐和光照实验。要求修改代码达到以下要求：

1. 通过设置纹理，使得茶壶纹理为：

2. 使得桌子纹理为：

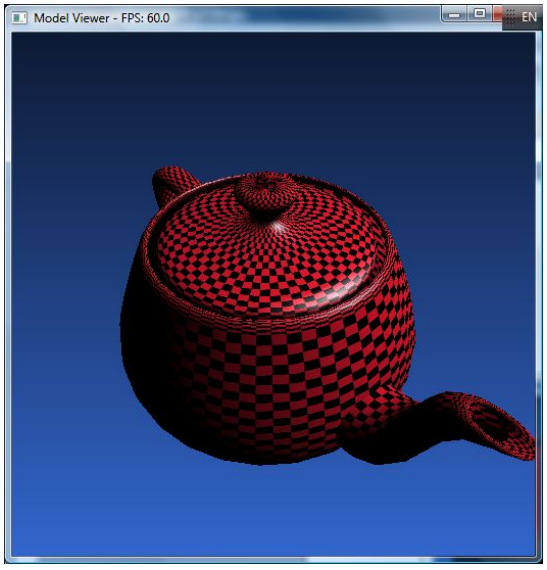
装 订 线

3. 对桌面上实现两张纹理的叠合效果：


+


4. 对茶壶实现纹理和光照效果的混合

5. 自己用代码产生一张纹理，并贴在茶壶表面，效果类似：



提示：glSolidCube()并不会为多边形指定纹理坐标，因此需要自己重写一个有纹理坐标的方块函数。

三、主要仪器设备

Visual Studio C++

glut-3.7.6-bin.zip
模板工程

四、操作方法和实验步骤

1.通过设置纹理，使得茶壶纹理为"Monet.bmp"

利用 readme_2.doc 中提供的读取纹理函数 LoadBitmapFile()与加载纹理的函数 texload()从图片"Monet.bmp"中获取纹理，然后在绘制茶壶前为茶壶加载已定义的纹理。init()函数为初始化纹理的函数，详细过程见代码：

```
// 读纹理函数
unsigned char *LoadBitmapFile(char *filename, BITMAPINFOHEADER *bitmapInfoHeader);

// 加载纹理的函数
void texload(int i,char *filename);

// 初始化纹理的函数：
void init()
{
    glGenTextures(4, texture);    // 第一参数是需要生成标示符的个数，第二参数是返回标示符的数组
    texload(0,"Monet.bmp");     // 从"Monet.bmp"中加载纹理
    ......
}
void Draw_Triangle()
{
    // 为茶壶加载纹理
    // 从已转载纹理中选择当前纹理
    glBindTexture(GL_TEXTURE_2D, texture[0]);
    glColor3f(1.0f,1.0f,1.0f);

    glPushMatrix();
    glTranslatef(0, 0, 4+1);
    glRotatef(90, 1, 0, 0);
    glutSolidTeapot(1);
    glPopMatrix();
    ......
}

void redraw()
{
    ......
    glEnable(GL_TEXTURE_2D);    // 开启纹理
    ......
}
```

2.自己实现绘制方块的函数 MySolidCube，通过对每个面指定纹理坐标使桌子的纹理为"Crack.bmp"

纹理的读取、加载基本同茶壶相同，但由于 glutSolidCube 没有对每个面指定纹理，必须自己重新实现绘制方块的函数 MySolidCube，并为每个面指定纹理坐标：

```
// 重新写画立方体的函数，为每一个面指定纹理坐标
static void MydrawBox(GLfloat size, GLenum type)
{
    ......
    for (i = 5; i >= 0; i--)
    {
        // 从已转载纹理中选择当前纹理
        glBindTexture(GL_TEXTURE_2D, texture[1]);
        // 将当前颜色设置为白色(重要)，可以试试设为红色效果如何
        glColor3f(1.0f,1.0f,1.0f);
        glBegin(type);
        glNormal3fv(&n[i][0]);
        // 指定纹理坐标
        glTexCoord2f(1.0,1.0);          glVertex3f(v[faces[i][0]][0],     v[faces[i][0]][1],
v[faces[i][0]][2]);
        glTexCoord2f(1.0,0.0);         glVertex3f(v[faces[i][1]][0],      v[faces[i][1]][1],
v[faces[i][1]][2]);
        glTexCoord2f(0.0,0.0);         glVertex3f(v[faces[i][2]][0],      v[faces[i][2]][1],
v[faces[i][2]][2]);
        glTexCoord2f(0.0,1.0);         glVertex3f(v[faces[i][3]][0],      v[faces[i][3]][1],
v[faces[i][3]][2]);
        glEnd();
    }
}

void MySolidCube(GLdouble size)
{
    MydrawBox(size, GL_QUADS);
}
```

3.对桌面上实现两张纹理的叠合效果

利用 OpenGL 扩展库 glew.h 和 glext.h 中的函数实现多重纹理（GL_ARB_multitexture）。

在读取与加载纹理后需要先初始化多重纹理单元：

```
int main (int argc,  char *argv[])
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGBA | GLUT_DEPTH | GLUT_DOUBLE);
    glutInitWindowSize(480,480);
    int windowHandle = glutCreateWindow("实验六_3090104732_桂义林");
```

```
    init();  // 加载纹理
    // 初始化多重纹理单元
    glMultiTexCoord1fARB =
        (PFNGLMULTITEXCOORD1FARBPROC) wglGetProcAddress("glMultiTexCoord1fARB");
    glMultiTexCoord2fARB =
        (PFNGLMULTITEXCOORD2FARBPROC) wglGetProcAddress("glMultiTexCoord2fARB");
    glMultiTexCoord3fARB =
        (PFNGLMULTITEXCOORD3FARBPROC) wglGetProcAddress("glMultiTexCoord3fARB");
    glMultiTexCoord4fARB =
        (PFNGLMULTITEXCOORD4FARBPROC) wglGetProcAddress("glMultiTexCoord4fARB");
    glActiveTextureARB =
        (PFNGLACTIVETEXTUREARBPROC) wglGetProcAddress("glActiveTextureARB");
    ......
}
```

然后在绘制时为不同的纹理单元中的纹理分别指定纹理坐标：

```
// 重新写画立方体的函数，为每一个面指定纹理坐标
static void MydrawBox(GLfloat size, GLenum type)
{
    ......
    for (i = 5; i >= 0; i--)
    {
        if (nTexMode == 0)
        {
            ......
        }
        else if (nTexMode == 1)
        {
            ......
        }
        else if (nTexMode == 2)    // 多重纹理模式
        {
            glActiveTextureARB(GL_TEXTURE0_ARB);
            glEnable(GL_TEXTURE_2D);
            // 从已转载纹理中选择当前纹理
            glBindTexture(GL_TEXTURE_2D, texture[1]);
            glActiveTextureARB(GL_TEXTURE1_ARB);
            glEnable(GL_TEXTURE_2D);
            // 从已转载纹理中选择当前纹理
            glBindTexture(GL_TEXTURE_2D, texture[2]);
            glBegin(type);
            glNormal3fv(&n[i][0]);
            // 为不同纹理单元中的纹理指定纹理坐标
            glMultiTexCoord2fARB(GL_TEXTURE0_ARB, 1.0, 1.0);
```

```
            glMultiTexCoord2fARB(GL_TEXTURE1_ARB, 1.0, 1.0);
            glVertex3f(v[faces[i][0]][0], v[faces[i][0]][1], v[faces[i][0]][2]);

            glMultiTexCoord2fARB(GL_TEXTURE0_ARB, 1.0, 0.0);
            glMultiTexCoord2fARB(GL_TEXTURE1_ARB, 1.0, 0.0);
            glVertex3f(v[faces[i][1]][0], v[faces[i][1]][1], v[faces[i][1]][2]);

            glMultiTexCoord2fARB(GL_TEXTURE0_ARB, 0.0, 0.0);
            glMultiTexCoord2fARB(GL_TEXTURE1_ARB, 0.0, 0.0);
            glVertex3f(v[faces[i][2]][0], v[faces[i][2]][1], v[faces[i][2]][2]);

            glMultiTexCoord2fARB(GL_TEXTURE0_ARB, 1.0, 1.0);
            glMultiTexCoord2fARB(GL_TEXTURE1_ARB, 1.0, 1.0);
            glVertex3f(v[faces[i][3]][0], v[faces[i][3]][1], v[faces[i][3]][2]);

            glEnd();
            glActiveTextureARB(GL_TEXTURE1_ARB);
            glDisable(GL_TEXTURE_2D);
            glActiveTextureARB(GL_TEXTURE0_ARB);
        }
    }
}
```

4.对茶壶实现纹理和光照效果的混合

在设置好光源和材质属性后，利用函数 glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, XXXX); 并设置参数 XXXX 为 GL_MODULATE 实现纹理和光照的混合效果：

```
void Draw_Triangle()
{
    // 设置茶壶材质
    glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, matAmb);
    glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, matDiff);
    glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, matSpec);
    glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 60.0);

    // 为茶壶加载纹理
    // 从已转载纹理中选择当前纹理
    if (nTeapotTex == 0)
    {
        glBindTexture(GL_TEXTURE_2D, texture[0]);  // "Monet.bmp"
    }
    else if (nTeapotTex == 1)
    {
```

```
        glBindTexture(GL_TEXTURE_2D, texture[3]);  // 自定义黑白格纹理
    }
    glColor3f(1.0f,1.0f,1.0f);

    // 纹理与光照混合
    // GL_MODULATE表示纹理与光照效果混合
    // GL_REPLACE则完全采用纹理颜色
    glTexEnvf(GL_TEXTURE_ENV,GL_TEXTURE_ENV_MODE,GL_MODULATE);

    glPushMatrix();
    glTranslatef(0, 0, 4+1);
    glRotatef(90, 1, 0, 0);
    glutSolidTeapot(1);
    glPopMatrix();
    ......
}
```

5.自己用代码产生一张红黑格子纹理，并贴在茶壶表面

图像的实质为一个二维矩阵，因此自定义纹理只须指定每个像素的颜色信息即可。程序中使用 RGBA 颜色模型，编写函数 void makeTexImage(void)生成自定义纹理，保存在二维数组 static GLubyte texImage[TEXW][TEXW][4]; 中：

```
// 自定义生成纹理的函数
void makeTexImage(void)
{
    int i, j, c;
    for (i = 0; i < TEXH; i++)
    {
        for (j = 0; j < TEXW; j++)
        {
            c = ((((i&0x8)==0)^((j&0x8))==0))*255;  // c 为0 或 255
            if (c == 0)  // c为0则设为黑色
            {
                texImage[i][j][0] = (GLubyte) c;
                texImage[i][j][1] = (GLubyte) c;
                texImage[i][j][2] = (GLubyte) c;
                texImage[i][j][3] = (GLubyte) 255;
            }
            Else  // c为255则设为红色
            {
                texImage[i][j][0] = (GLubyte) 225;
                texImage[i][j][1] = (GLubyte) 50;
                texImage[i][j][2] = (GLubyte) 50;
                texImage[i][j][3] = (GLubyte) 255;
```

```
            }
        }
    }
}
```

生成自定义纹理后，通过如下代码加载纹理：

```
// 定义纹理的函数：
void init()
{
    glGenTextures(4, texture);     // 第一参数是需要生成标示符的个数，第二参数是返回标示符的数组
    texload(0,"Monet.bmp");
    texload(1,"Crack.bmp");
    texload(2,"Spot.bmp");

    // 下面生成自定义纹理
    makeTexImage();
    glBindTexture(GL_TEXTURE_2D, texture[3]);
    glPixelStorei(GL_UNPACK_ALIGNMENT, 1); //设置像素存储模式控制所读取的图像数据的行对齐方式.
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA,TEXW,TEXH, 0, GL_RGBA, GL_UNSIGNED_BYTE, texImage);
    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_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
}
```

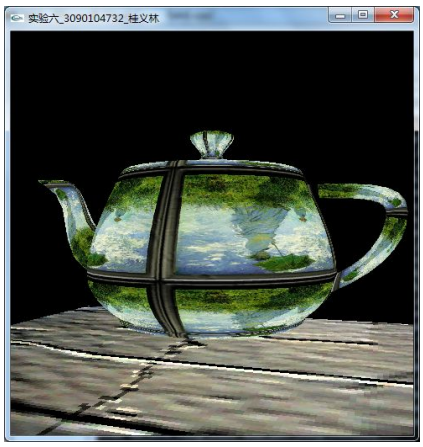
在绘制时贴纹理的步骤同前面几步过程相同。

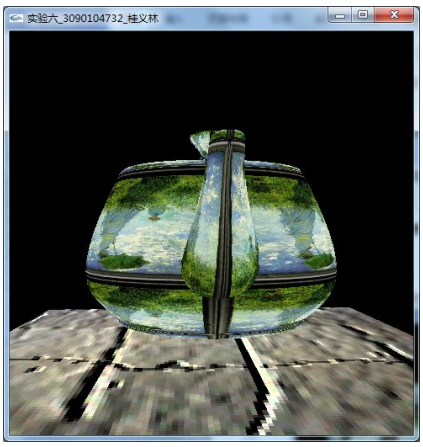
五、实验数据记录和处理

无

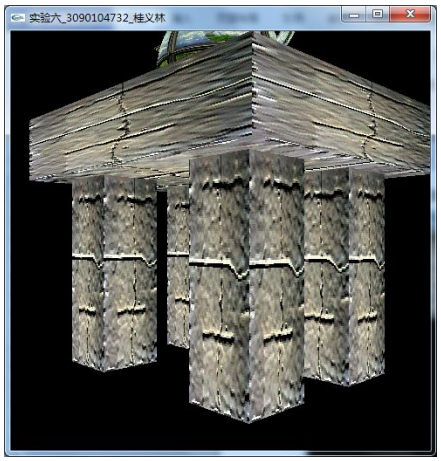
六、实验结果与分析

1.通过设置纹理，使得茶壶纹理为"Monet.bmp"

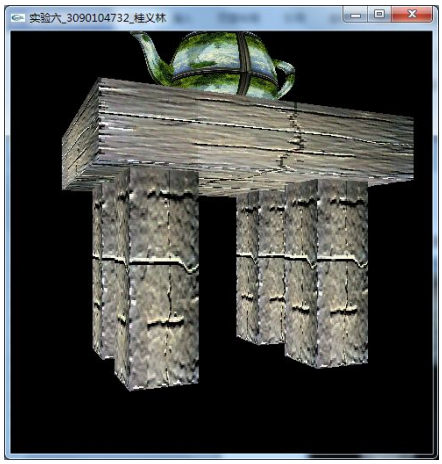




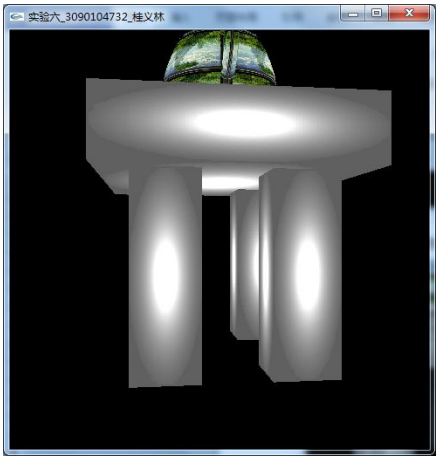
2.自己实现绘制方块的函数 MySolidCube，通过对每个面指定纹理坐标使桌子的纹理为"Crack.bmp"



3.对桌面上实现两张纹理的叠合效果



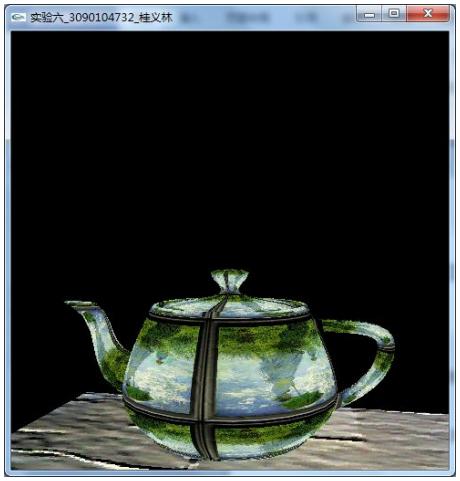
"Crack.bmp"



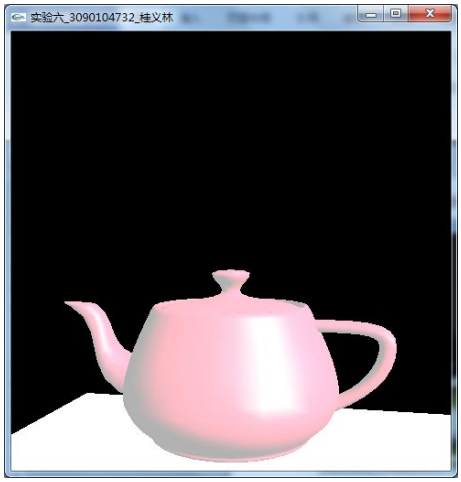
"Spot.bmp"

叠合后的效果：

4.对茶壶实现纹理和光照效果的混合

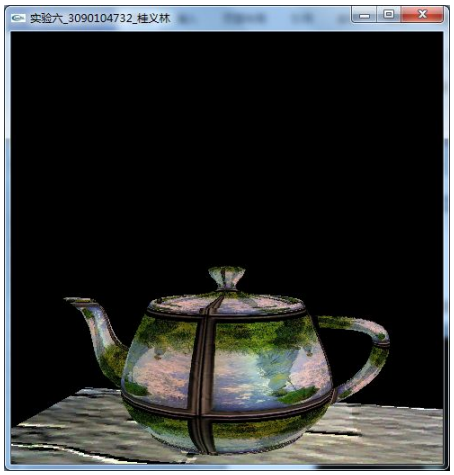


纹理效果

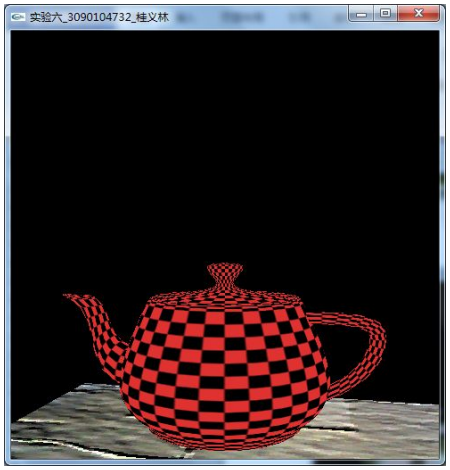


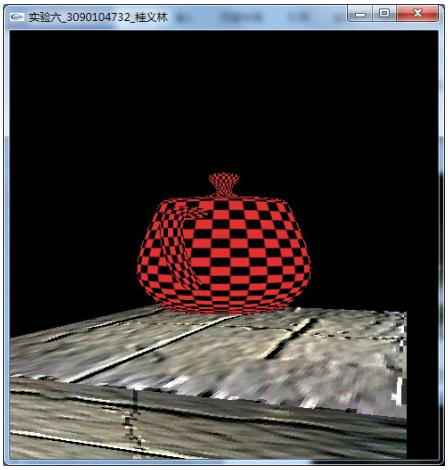
光照效果

纹理、光照融合后：



5.自己用代码产生一张红黑格子纹理，并贴在茶壶表面





七、讨论、心得

通过本次实验我掌握了 OpenGL 中纹理映射的设置方法，通过重写方块函数并为每个面指定纹理坐标进一步加深了对纹理映射的理解。在实现多重纹理的过程中，我一开始总是出现内存访问错误，经过调试和资料查阅后发现在使用 OpenGL 的 GL_ARB_multitexture 前要先对纹理单元进行初始化，且初始化要在 glut 初始化函数 glutInit()后，否则代码编译通过，但运行时会出错。在光照与纹理融合实验中，我通过设置参数 GL_MODULATE 观察融合后与融合前的区别，发现融合后的效果并不是十分明显，可能与纹理及我设置的茶壶的材质有关。