OpenGL实例：纹理映射

作者：凯鲁嘎吉 - 博客园 http://www.cnblogs.com/kailugaji/

更多请查看：计算机图形学

1. 介绍

用于指定一维、二维和三维纹理的函数分别为：

```
Void glTexImage1D(GLenum target, Glint level, Glint components, GLsizei width, Glint border, GLenum format, GLenum type, const GLvoid *texels);
Void glTexImage2D(GLenum target, Glint level, Glint components, GLsizei width, GLsizei height, GLint border, GLenum format, GLenum type, const GLvoid *texels);
Void glTexImage3D(GLenum target, Glint level, Glint components, GLsizei width, GLsizei height, GLsizei depth, Glint border, GLenum format, GLenum type, const GLvoid *texels);
```

其中，参数target取值一般为GL_TEXTURE_1D, GL_TEXTURE_2D和GL_TEXTURE_3D，分别与一维、二维和三维的纹理相对应。参数Level表示纹理多分辨率层数，通常取值为0，表示只有一种分辨率。参数components的可能取值为1～4的整数以及多种符号常量(如GL_RGBA)，表示纹理元素中存储的哪些分量（RGBA颜色、深度等）在纹理映射中被使用，1表示使用R颜色分量，2表示使用R和A颜色分量，3表示使用RGB颜色分量，4表示使用RGBA颜色分量。参数width，height，depth分别指定纹理的宽度、高度、深度。参数format和type表示给出的图像数据的数据格式和数据类型，这两个参数的取值都是符号常量（比如format指定为GL_RGBA,type指定为GL_UNSIGNED_BYTE，参数texels指向内存中指定的纹理图像数据。

在定义了纹理之后，需要启用纹理的函数：

```
glEnable(GL_TEXTURE_1D);
glEnable(GL_TEXTURE_2D);
glEnable(GL_TEXTURE_3D);
```

在启用纹理之后，需要建立物体表面上点与纹理空间的对应关系，即在绘制基本图元时，在glVertex函数调用之前调用glTexCoord函数，明确指定当前顶点所对应的纹理坐标，例如：

```
glBegin（GL_TRIANGLES）;
   glTexCoord2f(0.0,  0.0); glVertex2f(0.0,  0.0);
   glTexCoord2f(1.0,  1.0); glVertex2f(15.0,  15.0);
glTexCoord2f(1.0,  0.0); glVertex2f(30.0,  0.0);
glEnd();
```

其图元内部点的纹理坐标利用顶点处的纹理坐标采用线性插值的方法计算出来。

在OpenGL中，纹理坐标的范围被指定在[0,1]之间，而在使用映射函数进行纹理坐标计算时，有可能得到不在[0,1]之间的坐标。此时OpenGL有两种处理方式，一种是截断，另一种是重复，它们被称为环绕模式。在截断模式（GL_CLAMP）中，将大于1.0的纹理坐标设置为1.0，将小于0.0的纹理坐标设置为0.0。在重复模式（GL_REPEAT）中，如果纹理坐标不在[0,1]之间，则将纹理坐标值的整数部分舍弃，只使用小数部分，这样使纹理图像在物体表面重复出现。例如，使用下面的函数：

```
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
```

分别指定二维纹理中s坐标采用截断或重复处理方式。

另外，在变换和纹理映射后，屏幕上的一个像素可能对应纹理元素的一小部分（放大），也可能对应大量的处理元素（缩小）。在OpenGL中，允许指定多种方式来决定如何完成像素与纹理元素对应的计算方法（滤波）。比如，下面的函数可以指定放大和缩小的滤波方法：

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
```

其中，glTexParameteri函数的第一个参数指定使用的是一维、二维或三维纹理；第二个参数为GL_TEXTURE_MAG_FILTER或GL_TEXTURE_MIN_FILTER，指出要指定缩小还是放大滤波算法；最后一个参数指定滤波的方法。

补充：透视投影函数

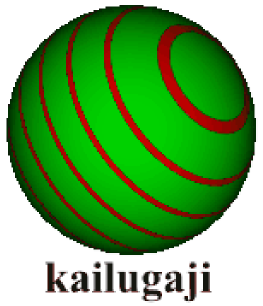
```
void gluPerspective(GLdouble fovy, GLdouble aspect, GLdouble zNear, GLdouble zFar);
```

它也创建一个对称透视视景体，但它的参数定义于前面的不同。其操作是创建一个对称的透视投影矩阵，并且用这个矩阵乘以当前矩阵。参数fovy定义视野在X-Z平面的角度，范围是[0.0,180.0]；参数aspect是投影平面宽度与高度的比率；参数zNear和Far分别是远近裁剪面沿Z负轴到视点的距离，它们总为正值。

2. 实例一

```
#include <GL/glut.h>
#include <cstdlib>
#include <cstdio>
#include <cmath>
#define stripeImageWidth 32

GLubyte stripeImage[4 * stripeImageWidth];

// create 2D texture
void makeStripeImage(void)
{
        int j;
        for (j = 0; j < stripeImageWidth; j++)
        {
                stripeImage[4 * j + 0] = (GLubyte)((j <= 4) ? 255 : 0);
                stripeImage[4 * j + 1] = (GLubyte)((j > 4) ? 255 : 0);
                stripeImage[4 * j + 2] = (GLubyte)0;
                stripeImage[4 * j + 3] = (GLubyte)255;
        }
}

static GLfloat xequalzero[] = { 1.0, 1.0, 1.0, 1.0 };
static GLfloat slanted[] = { 1.0, 1.0, 1.0, 1.0 };
static GLfloat *currentCoeff;
static GLenum currentPlane;
static GLint currentGenMode;
static float roangles;
void init(void)
{
        glClearColor(1.0, 1.0, 1.0, 1.0);
        glEnable(GL_DEPTH_TEST);
        glShadeModel(GL_SMOOTH);
        makeStripeImage();
        glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
        glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_WRAP_S, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
        glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
        glTexImage1D(GL_TEXTURE_1D, 0, 4, stripeImageWidth, 0, GL_RGBA, GL_UNSIGNED_BYTE, stripeImage);
        glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
        currentCoeff = xequalzero;
        currentGenMode = GL_OBJECT_LINEAR;
        currentPlane = GL_OBJECT_PLANE;
        glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, currentGenMode);
        glTexGenfv(GL_S, currentPlane, currentCoeff);
        glEnable(GL_TEXTURE_GEN_S);
        glEnable(GL_TEXTURE_1D);
        glEnable(GL_LIGHTING);
        glEnable(GL_LIGHT0);
        glEnable(GL_AUTO_NORMAL);
        glEnable(GL_NORMALIZE);
        glMaterialf(GL_FRONT, GL_SHININESS, 64.0);
        roangles = 45.0f;
}

void display(void) {
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glPushMatrix();
        glRotatef(roangles, 0.0, 0.0, 1.0);
        glutSolidSphere(2.0, 32, 32);
        glPopMatrix();
        glFlush();
};

void reshape(int w, int h) {
        glViewport(0, 0, (GLsizei)w, (GLsizei)h);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        if (w <= h)
                glOrtho(-3.5, 3.5, -3.5*(GLfloat)h / (GLfloat)w, 3.5*(GLfloat)h / (GLfloat)w, -3.5, 3.5);
        else
                glOrtho(-3.5*(GLfloat)w / (GLfloat)h, 3.5*(GLfloat)w / (GLfloat)h, -3.5, 3.5, -3.5, 3.5);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
}

void idle()
{
        roangles += 0.05f;
        glutPostRedisplay();
}

int main(int argc, char** argv) {
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
        glutInitWindowSize(256, 256);
        glutInitWindowPosition(100, 100);
        glutCreateWindow(argv[0]);
        glutIdleFunc(idle);
        init();
        glutDisplayFunc(display);
        glutReshapeFunc(reshape);
        glutMainLoop();
        return 0;
}
```

结果:

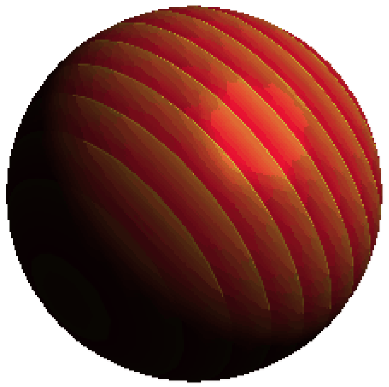


3. 实例二

```
#include <GL/glut.h>
#include <cstdlib>
#include <cstdio>
#include <cmath>
#define stripeImageWidth 32
#define stripeImageHeight 32
GLubyte stripeImage[stripeImageWidth][stripeImageHeight][4];
void makeStripeImage(void)
{
        int i, j;
        for (i = 0; i < stripeImageWidth; i++)
        {
                for (j = 0; j < stripeImageHeight; j++)
                {
                        stripeImage[i][j][0] = (GLubyte)(i * 7 - 1);
                        stripeImage[i][j][1] = (GLubyte)(j * 4 - 1);
                        stripeImage[i][j][2] = (GLubyte)50;
                        stripeImage[i][j][3] = (GLubyte)255;
                }
        }
}
static GLfloat xequalzero[] = { 1.0, 1.0, 1.0, 1.0 };
static GLfloat slanted[] = { 1.0, 1.0, 1.0, 1.0 };
static GLfloat *currentCoeff;
static GLenum currentPlane;
static GLint currentGenMode;
static float roangles;
void init(void)
{
	glClearColor(1.0, 1.0, 1.0, 1.0);
	glEnable(GL_DEPTH_TEST);
	glShadeModel(GL_SMOOTH);
	makeStripeImage();
	glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
	glTexImage2D(GL_TEXTURE_2D, 0, 4, stripeImageWidth, stripeImageHeight, 0, GL_RGBA, GL_UNSIGNED_BYTE, stripeImage);
	glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
	currentCoeff = xequalzero;
	currentGenMode = GL_OBJECT_LINEAR;
	currentPlane = GL_OBJECT_PLANE;
	glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, currentGenMode);
	glTexGenfv(GL_S, currentPlane, currentCoeff);
	glEnable(GL_TEXTURE_GEN_S);
	glEnable(GL_TEXTURE_2D);
	glEnable(GL_LIGHTING);
	glEnable(GL_LIGHT0);
	glEnable(GL_AUTO_NORMAL);
	glEnable(GL_NORMALIZE);
	glMaterialf(GL_FRONT, GL_SHININESS, 64.0);
	roangles = 60.0f;
	//光照
	GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
	GLfloat mat_shininess[] = { 50.0 };
	GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };    //光源位置
	GLfloat white_light[] = { 1.0, 1.0, 1.0, 1.0 };
	GLfloat Light_Model_Ambient[] = { 0.2, 0.2, 0.2, 1.0 };
	glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);    //设置材料反射指数（纯镜面反射）
	glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);  //设置材料反射指数（材料反射指数）
	glLightfv(GL_LIGHT0, GL_POSITION, light_position);   //建立光源 （光源位置）
	glLightfv(GL_LIGHT0, GL_DIFFUSE, white_light);       //建立光源 （漫反射光分量强度）
	glLightfv(GL_LIGHT0, GL_SPECULAR, white_light);      //建立光源 （折射光分量强度）
	glLightModelfv(GL_LIGHT_MODEL_AMBIENT, Light_Model_Ambient);
}
void display(void) {
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
	glPushMatrix();
	glRotatef(roangles, 0.0, 1.0, 0.0);
	glutSolidSphere(3.0, 32, 32);
	glPopMatrix();
	glFlush();
}
void reshape(int w, int h) {
	glViewport(0, 0, (GLsizei)w, (GLsizei)h);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	if (w <= h)
		glOrtho(-3.5, 3.5, -3.5*(GLfloat)h / (GLfloat)w, 3.5*(GLfloat)h / (GLfloat)w, -3.5, 3.5);
	else
		glOrtho(-3.5*(GLfloat)w / (GLfloat)h, 3.5*(GLfloat)w / (GLfloat)h, -3.5, 3.5, -3.5, 3.5);
	glMatrixMode(GL_MODELVIEW);
```

```
        glLoadIdentity();
}
void idle()
{
        roangles += 0.05f;
        glutPostRedisplay();
}
int main(int argc, char** argv) {
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
        glutInitWindowSize(256, 256);
        glutInitWindowPosition(100, 100);
        glutCreateWindow("balloon");
        glutIdleFunc(idle);
        init();
        glutDisplayFunc(display);
        glutReshapeFunc(reshape);
        glutMainLoop();
        return 0;
}
```

结果:



kailugaji