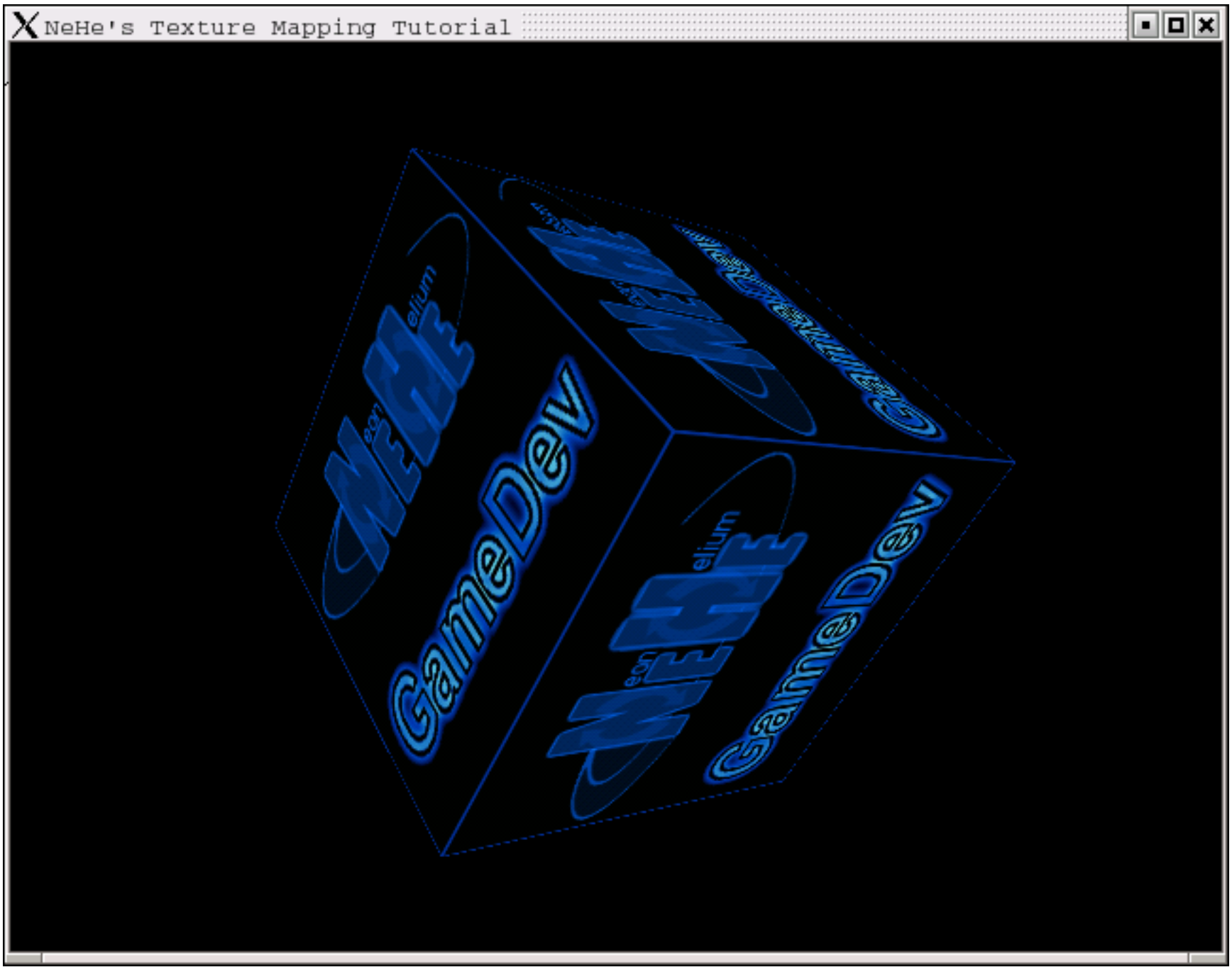
纹理映射



学习texture map纹理映射（贴图）有很多好处。比方说您想让一颗导弹飞过屏幕。根据前几课的知识，我们最可行的办法可能是很多个多边形来构建导弹的轮廓并加上有趣的颜色。使用纹理映射，您可以使用真实的导弹图像并让它飞过屏幕。您觉得哪个更好看？照片还是一大堆三角形和四边形？使用纹理映射的好处还不止是更好看，而且您的程序运行会更快。导弹贴图可能只是一个飞过窗口的四边形。一个由多边形构建而来的导弹却很可能包括成百上千的多边形。很显然，贴图极大的节省了CPU时间。

我们要在第一课的代码上增加几行就可以了。

我们将要增加一个loadGLTextures()函数来处理有关纹理操作的。我们将在NeHeWidget类中增加三个变量xRot、yRot、zRot来处理立方体的旋转。还有一个用来存储纹理的texture[1]。

NeHeWidget类

（由nehewidget.h展开。）

```
protected:
  void loadGLTextures();
```

在这个函数中我们会载入指定的图片并生成相应当纹理。

```
protected:

  bool fullscreen;
  GLfloat xRot, yRot, zRot;
  GLuint texture[1];

};
```

上面就是添加的三个变量xRot、yRot、zRot来处理立方体在三个方向上的旋转。texture[1]用来存储一个纹理。

（由nehewidget.cpp展开。）

```
NeHeWidget::NeHeWidget( QWidget* parent, const char* name, bool fs )
    : QGLWidget( parent, name )
{
  xRot = yRot = zRot = 0.0;
  fullscreen = fs;
  setGeometry( 0, 0, 640, 480 );
  setCaption( "NeHe's Texture Mapping Tutorial" );
  if ( fullscreen )
    showFullScreen();
}
```

我们需要在构造函数中给xRot、yRot、zRot赋初值，都是0.0。

```
void NeHeWidget::loadGLTextures()
{
  QImage tex, buf;
  if ( !buf.load( "./data/NeHe.bmp" ) )
```

载入纹理图片。这里使用了QImage类。

```
  {
    qWarning( "Could not read image file, using single-color instead." );
    QImage dummy( 128, 128, 32 );
    dummy.fill( Qt::green.rgb() );
    buf = dummy;
```

如果载入不成功，自动生成一个128*128的32位色的绿色图片。

```
  }
  tex = QGLWidget::convertToGLFormat( buf );
```

这里使用了QGLWidget类中提供的一个静态函数converToGLFormat()，专门用来转换图片的，具体情况请参见相应文档。

```
  glGenTextures( 1, &texture[0] );
```

创建一个纹理。告诉OpenGL我们想生成一个纹理名字(如果您想载入多个纹理，加大数字)。值得注意的是，开始我们使用 GLuint texture[1] 来创建一个纹理的存储空间，您也许会认为第一个纹理就是存放在 &texture[1] 中的，但这是错的。正确的地址应该是 &texture[0] 。同样如果使用 GLuint texture[2] 的话，第二个纹理存放在 texture[1] 中。

```
  glBindTexture( GL_TEXTURE_2D, texture[0] );
```

使用来自位图数据生成的典型纹理。告诉OpenGL将纹理名字texture[0]绑定到纹理目标上。2D纹理只有高度（在Y轴上）和宽度（在X轴上）。主函数将纹理名字指派给纹理数据。本例中我们告知OpenGL，&texture[0]处的内存已经可用。我们创建的纹理将存储在&texture[0]的指向的内存区域。

```
  glTexImage2D( GL_TEXTURE_2D, 0, 3, tex.width(), tex.height(), 0,
      GL_RGBA, GL_UNSIGNED_BYTE, tex.bits() );
```

这里真正的创建纹理。GL_TEXTURE_2D告诉OpenGL此纹理是一个2D纹理。数字零代表图像的详细程度，通常就由它为零去了。数字三是数据的成分数。因为图像是由红色数据，绿色数据，蓝色数据三种组分组成。 tex.width()是纹理的宽度。tex.height()是纹理的高度。数字零是边框的值，一般就是零。GL_RGBA 告诉OpenGL图像数据由红、绿、蓝三色数据以及alpha通道数据组成，这个是由于QGLWidget类的converToGLFormat()函数的原因。 GL_UNSIGNED_BYTE 意味着组成图像的数据是无符号字节类型的。最后tex.bits()告诉OpenGL纹理数据的来源。

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

上面的两行告诉OpenGL在显示图像时，当它比放大得原始的纹理大（GL_TEXTURE_MAG_FILTER）或缩小得比原始得纹理小（GL_TEXTURE_MIN_FILTER）时OpenGL采用的滤波方式。通常这两种情况下我都采用GL_LINEAR。这使得纹理从很远处到离屏幕很近时都平滑显示。使用GL_LINEAR需要CPU和显卡做更多的运算。如果您的机器很慢，您也许应该采用GL_NEAREST。过滤的纹理在放大的时候，看起来斑驳的很。您也可以结合这两种滤波方式。在近处时使用GL_LINEAR，远处时GL_NEAREST。

```
}
```

loadGLTextures()函数就是用来载入纹理的。

```
void NeHeWidget::paintGL()
{
  glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
  glLoadIdentity();
  glTranslatef(  0.0,  0.0, -5.0 );

  glRotatef( xRot,  1.0,  0.0,  0.0 );
  glRotatef( yRot,  0.0,  1.0,  0.0 );
  glRotatef( zRot,  0.0,  0.0,  1.0 );
```

根据xRot、yRot、zRot的实际值来旋转正方体。

```
  glBindTexture( GL_TEXTURE_2D, texture[0] );
```

选择我们使用的纹理。如果您在您的场景中使用多个纹理，您应该使用来glBindTexture(GL_TEXTURE_2D, texture[所使用纹理对应的数字]) 选择要绑定的纹理。当您想改变纹理时，应该绑定新的纹理。有一点值得指出的是，您不能在glBegin()和glEnd()之间绑定纹理，必须在glBegin()之前或glEnd()之后绑定。注意我们在上面是如何使用glBindTexture来指定和绑定纹理的。

```
glBegin( GL_QUADS );
```

为了将纹理正确的映射到四边形上，您必须将纹理的右上角映射到四边形的右上角，纹理的左上角映射到四边形的左上角，纹理的右下角映射到四边形的右下角，纹理的左下角映射到四边形的左下角。如果映射错误的话，图像显示时可能上下颠倒，侧向一边或者什么都不是。

glTexCoord2f的第一个参数是X坐标。0.0是纹理的左侧。0.5是纹理的中点，1.0是纹理的右侧。
glTexCoord2f的第二个参数是Y坐标。0.0是纹理的底部。0.5是纹理的中点，1.0是纹理的顶部。

所以纹理的左上坐标是X：0.0，Y：1.0f，四边形的左上顶点是X：-1.0，Y：1.0。其余三点依此类推。

试着玩玩glTexCoord2f的X、Y坐标参数。把1.0改为0.5将只显示纹理的左半部分，把0.0改为0.5将只显示纹理的右半部分。

```
glTexCoord2f( 0.0, 0.0 ); glVertex3f( -1.0, -1.0,  1.0 );
glTexCoord2f( 1.0, 0.0 ); glVertex3f(  1.0, -1.0,  1.0 );
glTexCoord2f( 1.0, 1.0 ); glVertex3f(  1.0,  1.0,  1.0 );
glTexCoord2f( 0.0, 1.0 ); glVertex3f( -1.0,  1.0,  1.0 );
```

前面。

```
glTexCoord2f( 1.0, 0.0 ); glVertex3f( -1.0, -1.0, -1.0 );
glTexCoord2f( 1.0, 1.0 ); glVertex3f( -1.0,  1.0, -1.0 );
glTexCoord2f( 0.0, 1.0 ); glVertex3f(  1.0,  1.0, -1.0 );
glTexCoord2f( 0.0, 0.0 ); glVertex3f(  1.0, -1.0, -1.0 );
```

后面。

```
glTexCoord2f( 0.0, 1.0 ); glVertex3f( -1.0,  1.0, -1.0 );
glTexCoord2f( 0.0, 0.0 ); glVertex3f( -1.0,  1.0,  1.0 );
glTexCoord2f( 1.0, 0.0 ); glVertex3f(  1.0,  1.0,  1.0 );
glTexCoord2f( 1.0, 1.0 ); glVertex3f(  1.0,  1.0, -1.0 );
```

顶面。

```
glTexCoord2f( 1.0, 1.0 ); glVertex3f( -1.0, -1.0, -1.0 );
glTexCoord2f( 0.0, 1.0 ); glVertex3f(  1.0, -1.0, -1.0 );
glTexCoord2f( 0.0, 0.0 ); glVertex3f(  1.0, -1.0,  1.0 );
glTexCoord2f( 1.0, 0.0 ); glVertex3f( -1.0, -1.0,  1.0 );
```

底面。

```
glTexCoord2f( 1.0, 0.0 ); glVertex3f(  1.0, -1.0, -1.0 );
glTexCoord2f( 1.0, 1.0 ); glVertex3f(  1.0,  1.0, -1.0 );
glTexCoord2f( 0.0, 1.0 ); glVertex3f(  1.0,  1.0,  1.0 );
glTexCoord2f( 0.0, 0.0 ); glVertex3f(  1.0, -1.0,  1.0 );
```

右面。

```
    glTexCoord2f( 0.0, 0.0 ); glVertex3f( -1.0, -1.0, -1.0 );
    glTexCoord2f( 1.0, 0.0 ); glVertex3f( -1.0, -1.0,  1.0 );
    glTexCoord2f( 1.0, 1.0 ); glVertex3f( -1.0,  1.0,  1.0 );
    glTexCoord2f( 0.0, 1.0 ); glVertex3f( -1.0,  1.0, -1.0 );
```

左面。

```
  glEnd();

  xRot += 0.3;
  yRot += 0.2;
  zRot += 0.4;
```

现在改变xRot、yRot、zRot的值。尝试变化每次各变量的改变值来调节立方体的旋转速度，或改变+/-号来调节立方体的旋转方向。

```
}

void NeHeWidget::initializeGL()
{
  loadGLTextures();
```

载入纹理。

```
  glEnable( GL_TEXTURE_2D );
```

启用纹理。如果没有启用的话，你的对象看起来永远都是纯白色，这一定不是什么好事。

```
  glShadeModel( GL_SMOOTH );
  glClearColor( 0.0, 0.0, 0.0, 0.5 );
  glClearDepth( 1.0 );
  glEnable( GL_DEPTH_TEST );
  glDepthFunc( GL_LEQUAL );
  glHint( GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST );
}
```

现在您应该比较好的理解纹理映射（贴图）了。您应该掌握了给任意四边形表面贴上您所喜爱的图像的技术。一旦您对2D纹理映射的理解感到自信的时候，试试给立方体的六个面贴上不同的纹理。

当您理解纹理坐标的概念后，纹理映射并不难理解。

本课程的源代码。

[上一课：向三维进军] [Qt OpenGL教程主页] [下一课：纹理滤波、光源和键盘控制]

---

http://www.qiliang.net mailto:cavendish@qiliang.net 2002年12月26日