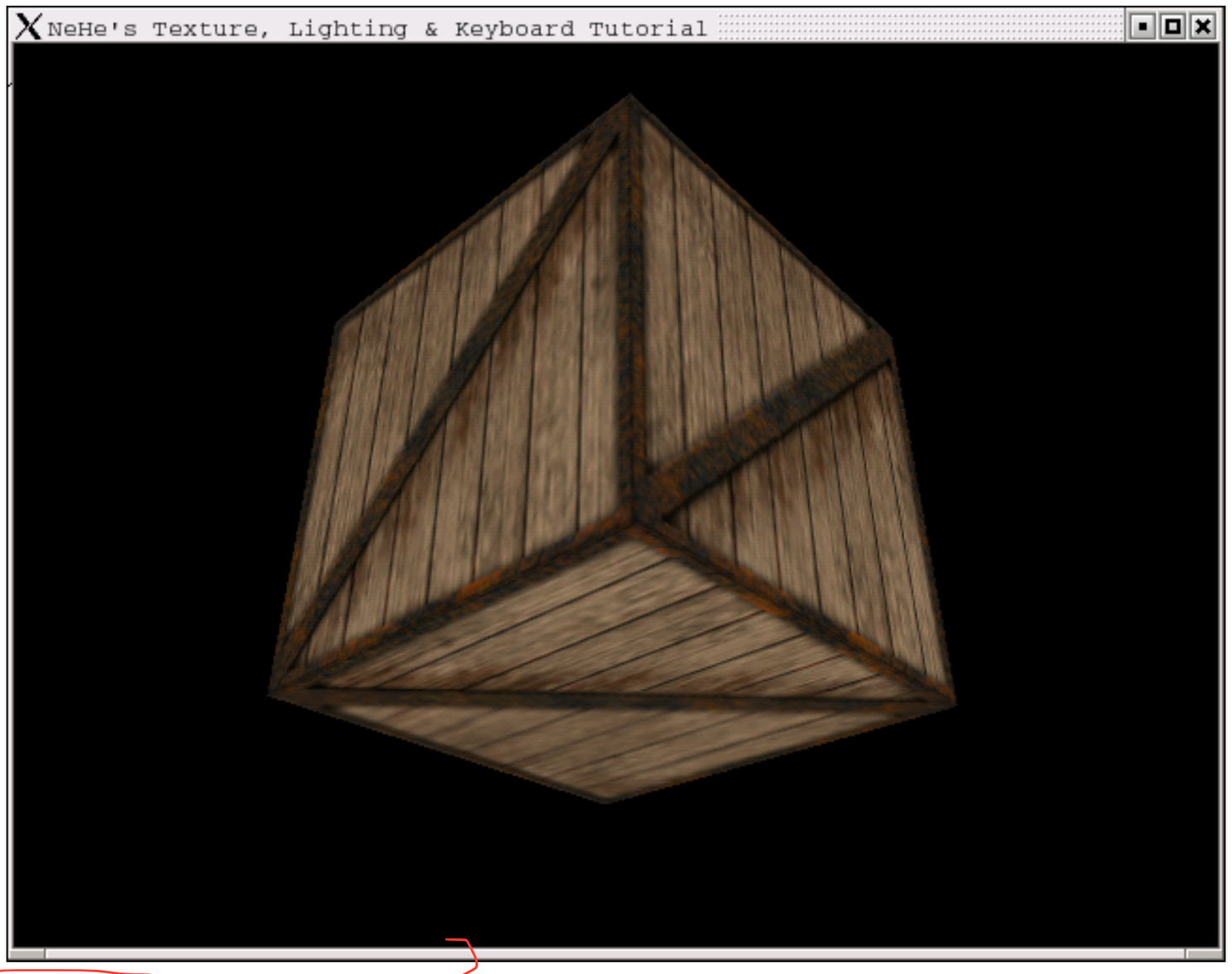
纹理滤波、光源和键盘控制



这一课我会教您如何使用三种不同的纹理滤波方式。教您如何使用键盘来移动场景中的对象，还会教您在OpenGL场景中应用简单的光照。这一课包含了很多内容，如果您对前面的课程有疑问的话，先回头复习一下。进入后面的代码之前，很好的理解基础知识十分重要。

我们要在第一课的代码上进行改动就可以了。

我们将要增加一个loadGLTextures()函数来处理有关纹理操作的。我们将增加一些变量，稍后我们对这些变量进行解释。

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;
  GLfloat zoom;
  GLfloat xSpeed, ySpeed;
  GLuint texture[3];
  GLuint filter;

  bool light;

};
```

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。zoom是场景深入屏幕的距离。xSpeed和ySpeed是立方体在X轴和Y轴上旋转的速度。texture[3]用来存储三个纹理。filter表明的是使用哪个纹理。light是说明现在是否使用光源。

（由nehewidget.cpp展开。）

```
GLfloat lightAmbient[4] = { 0.5, 0.5, 0.5, 1.0 };
GLfloat lightDiffuse[4] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat lightPosition[4] = { 0.0, 0.0, 2.0, 1.0 };
```

这里定义了三个数组，它们描述的是和光源有关的信息。

我们将使用两种不同的光。第一种称为环境光。环境光来自于四面八方。所有场景中的对象都处于环境光的照射中。第二种类型的光源叫做漫射光。漫射光由特定的光源产生，并在您的场景中的对象表面上产生反射。处于漫射光直接照射下的任何对象表面都变得很亮，而几乎未被照射到的区域就显得要暗一些。这样在我们所创建的木板箱的棱边上就会产生的很不错的阴影效果。

创建光源的过程和颜色的创建完全一致。前三个参数分别是RGB三色分量，最后一个是alpha通道参数。

因此，第一行有关lightAmbient的代码使我们得到的是半亮（0.5）的白色环境光。如果没有环境光，未被漫射光照到的地方会变得十分黑暗。

第二行有关lightDiffuse的代码使我们生成最亮的漫射光。所有的参数值都取成最大值1.0。它将照在我们木板箱的前面，看起来挺好。

第三行有关lightPosition的代码使我们保存光源的位置。前三个参数和glTranslate中的一样。依次分别是XYZ轴上的位移。由于我们想要光线直接照射在木箱的正面，所以XY轴上的位移都是0.0。第三个值是Z轴上的位移。为了保证光线总在木箱的前面，所以我们将光源的位置朝着观察者（就是您哪。）挪出屏幕。我们通常将屏幕也就是显示器的屏幕玻璃所处的位置称作Z轴的0.0点。所以Z轴上的位移最后定为2.0。假如您能够看见光源的话，它就浮在您显示器的前方。当然，如果木箱不在显示器的屏幕玻璃后面的话，您也无法看见箱子。最后一个参数取为1.0。这将告诉OpenGL这里指定的坐标就是光源的位置，以后的教程中我会多加解释。

```
NeHeWidget::NeHeWidget( QWidget* parent, const char* name, bool fs )
    : QGLWidget( parent, name )
{
  xRot = yRot = zRot = 0.0;
  zoom = -5.0;
  xSpeed = ySpeed = 0.0;

  filter = 0;

  light = false;

  fullscreen = fs;
  setGeometry( 0, 0, 640, 480 );
  setCaption( "NeHe's Texture, Lighting & Keyboard Tutorial" );

  if ( fullscreen )
    showFullScreen();
}
```

我们需要在构造函数中给各个变量赋初值。xRot、yRot、zRot是0.0。zoom是-5.0。xSpeed和ySpeed都是0。filter是0。light是false。

```
void NeHeWidget::loadGLTextures()
{
  QImage tex, buf;
  if ( !buf.load( "./data/Crate.bmp" ) )
  {
    qWarning( "Could not read image file, using single-color instead." );
    QImage dummy( 128, 128, 32 );
    dummy.fill( Qt::green.rgb() );
    buf = dummy;
  }
  tex = QGLWidget::convertToGLFormat( buf );

  glGenTextures( 3, &texture[0] );
```

这一部分，上一章讲过了。我们这里创建了3个纹理。

```
  glBindTexture( GL_TEXTURE_2D, texture[0] );
  glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST );
  glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST );
  glTexImage2D( GL_TEXTURE_2D, 0, 3, tex.width(), tex.height(), 0,
      GL_RGBA, GL_UNSIGNED_BYTE, tex.bits() );
```

第六课中我们使用了线性滤波的纹理贴图。这需要机器有相当高的处理能力，但它们看起来很不错。这一课中，我们接着要创建的第一种纹理使用GL_NEAREST方式。从原理上讲，这种方式没有真正进行滤波。它只占用很小的处理能力，看起来也很差。唯一的好处是这样我们的工程在很快和很慢的机器上都可以正常运行。您会注意到我们在MIN和MAG时都采用了GL_NEAREST,你可以混合使用GL_NEAREST和GL_LINEAR。纹理看起来效果会好些，但我们更关心速度，所以全采用低质量贴图。MIN_FILTER在图像绘制时小于贴图的原始尺寸时采用。MAG_FILTER在图像绘制时大于贴图的原始尺寸时采用。

```
  glBindTexture( GL_TEXTURE_2D, texture[1] );
  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, 3, tex.width(), tex.height(), 0,
      GL_RGBA, GL_UNSIGNED_BYTE, tex.bits() );
```

这个纹理与第六课的相同，线性滤波。唯一的不同是这次放在了texture[1]中。因为这是第二个纹理。如果放在texture[0]中的话，它将覆盖前面创建的GL_NEAREST纹理。

```
  glBindTexture( GL_TEXTURE_2D, texture[2] );
  glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR );
  glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_NEAREST );
```

这里是创建纹理的新方法。Mipmapping！您可能会注意到当图像在屏幕上变得很小的时候，很多细节将会丢失。刚才还很不错的图案变得很难看。当您告诉OpenGL创建一个 mipmapped的纹理后，OpenGL将尝试创建不同尺寸的高质量纹理。当您向屏幕绘制一个mipmapped纹理的时候，OpenGL将选择它已经创建的外观最佳的纹理(带有更多细节)来绘制，而不仅仅是缩放原先的图像（这将导致细节丢失）。

我曾经说过有办法可以绕过OpenGL对纹理宽度和高度所加的限制——64、128、256，等等。办法就是gluBuild2DMipmaps。据我的发现，您可以使用任意的位图来创建纹理。OpenGL将自动将它缩放到正常的大小。

因为是第三个纹理，我们将它存到texture[2]。这样本课中的三个纹理全都创建好了。 gluBuild2DMipmaps(GL_TEXTURE_2D, GL_RGB, tex.width(), tex.height(), GL_RGBA, GL_UNSIGNED_BYTE, tex.bits());

```
 这一行生成 mipmapped 纹理。我们使用三种颜色（红，绿，蓝）来生成一个2D纹理。tex.width()是位图宽度，tex.height()是位图高度，extureImage[0]->sizeY 是位图高度，GL_RGB
}
```

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, zoom );

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

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

根据filter变量来决定使用哪个纹理。

```
  glBegin( GL_QUADS );
    glNormal3f( 0.0, 0.0, 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 );
    glTexCoord2f( 0.0, 1.0 ); glVertex3f( -1.0,  1.0,  1.0 );

    glNormal3f( 0.0, 0.0, -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 );
    glTexCoord2f( 0.0, 0.0 ); glVertex3f(  1.0, -1.0, -1.0 );

    glNormal3f( 0.0, 1.0, 0.0 );
    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 );
    glTexCoord2f( 1.0, 1.0 ); glVertex3f(  1.0,  1.0, -1.0 );

    glNormal3f( 0.0, -1.0, 0.0 );
    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 );
    glTexCoord2f( 1.0, 0.0 ); glVertex3f( -1.0, -1.0,  1.0 );

    glNormal3f( 1.0, 0.0, 0.0 );
    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 );
    glTexCoord2f( 0.0, 0.0 ); glVertex3f(  1.0, -1.0,  1.0 );

    glNormal3f( -1.0, 0.0, 0.0 );
    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 );
    glTexCoord2f( 0.0, 1.0 ); glVertex3f( -1.0,  1.0, -1.0 );
  glEnd();
```

这里绘制正方体的方法，上一章已经讲解过了。

```
  xRot += xSpeed;
  yRot += ySpeed;
```

将xRot和yRot的旋转值分别增加xSpeed和ySpeed个单位。xSpeed和ySpeed的值越大，立方体转得就越快。

```
}

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

  glEnable( GL_TEXTURE_2D );
  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 );

  glLightfv( GL_LIGHT1, GL_AMBIENT, lightAmbient );
  glLightfv( GL_LIGHT1, GL_DIFFUSE, lightDiffuse );
  glLightfv( GL_LIGHT1, GL_POSITION, lightPosition );

  glEnable( GL_LIGHT1 );
```

这里开始设置光源。第一行设置环境光的发光量，光源GL_LIGHT1开始发光。这一课的开始处我们我们将环境光的发光量存放在lightAmbient数组中。现在我们就使用此数组（半亮度环境光）。

接下来我们设置漫射光的发光量。它存放在lightDiffuse数组中（全亮度白光）。

然后设置光源的位置。位置存放在lightPosition 数组中（正好位于木箱前面的中心，X- 0.0，Y- 0.0，Z方向移向观察者2个单位，位于屏幕外面）。

最后，我们启用一号光源。我们还没有启用GL_LIGHTING，所以您看不见任何光线。记住：只对光源进行设置、定位、甚至启用，光源都不会工作。除非我们启用GL_LIGHTING。

```
}

void NeHeWidget::keyPressEvent( QKeyEvent *e )
{
  switch ( e->key() )
```

```
  {
  case Qt::Key_L:
    light = !light;
    if ( !light )
    {
      glDisable( GL_LIGHTING );
    }
    else
    {
      glEnable( GL_LIGHTING );
    }
    updateGL();
    break;
```

按下了L键，就可以切换是否打开光源。

```
  case Qt::Key_F:
    filter += 1;;
    if ( filter > 2 )
    {
      filter = 0;
    }
    updateGL();
    break;
```

按下了F键，就可以转换一下所使用的纹理（就是变换了纹理滤波方式的纹理）。

```
  case Qt::Key_Prior:
    zoom -= 0.2;
    updateGL();
    break;
```

按下了PageUp键，将木箱移向屏幕内部。

```
  case Qt::Key_Next:
    zoom += 0.2;
    updateGL();
    break;
```

按下了PageDown键，将木箱移向屏幕外部。

```
  case Qt::Key_Up:
    xSpeed -= 0.01;
    updateGL();
    break;
```

按下了Up方向键，减少xSpeed。

```
  case Qt::Key_Down:
    xSpeed += 0.01;
    updateGL();
    break;
```

按下了Dowm方向键，增加xSpeed。

```
  case Qt::Key_Right:
    ySpeed += 0.01;
    updateGL();
    break;
```

按下了Right方向键，增加ySpeed。

```
  case Qt::Key_Left:
    ySpeed -= 0.01;
    updateGL();
    break;
```

按下了Left方向键，减少ySpeed。

```
  case Qt::Key_F2:
    fullscreen = !fullscreen;
    if ( fullscreen )
    {
      showFullScreen();
    }
    else
    {
      showNormal();
      setGeometry( 0, 0, 640, 480 );
    }
    update();
    break;
  case Qt::Key_Escape:
    close();
  }
}
```

这一课完了之后，您应该学会创建和使用这三种不同的纹理映射过滤方式。并使用键盘和场景中的对象交互。最后，您应该学会在场景中应用简单的光源，使得场景看起来更逼真。

本课程的源代码。

---

http://www.qiliang.net mailto:cavendish@qiliang.net 2003年2月24日