OpenGL 编程

纹理映射

纹理映射

- OpenGL纹理映射功能支持将一些像素数据经过变换将其附着到各种形状的多边形表面



载入纹理

- 启用纹理：
  - glEnable(GL_TEXTURE_2D)；//启用纹理
- 载入纹理：glTexImage2D函数
  - 参数1：为指定的目标GL_TEXTURE_2D
  - 参数2：多重细节层次，不使用设为0
  - 参数3：GL_RGB(3)/GL_RGBA(4)
  - 参数4,5：二维纹理像素的宽度和高度，新旧版本规定不一致
    - 最好是2的整数次方

载入纹理

- 参数4,5：二维纹理像素的宽度和高度,最好是2的整数次方
  - gluScaleImage把图象缩放至所指定的大小
  - 纹理的大小有限制，可用函数获取

```
GLint max;
glGetIntegerv(GL_MAX_TEXTURE_SIZE,
&max);
```

  - 图形程序常用256*256大小的纹理，很多硬件可以使用8位的整数来表示纹理坐标

载入纹理

- 参数6：纹理边框的大小
  - 如不使用纹理边框，可设为0
- 参数7,8,9：
  - 与glDrawPixels函数的最后三个参数的使用方法相同(内容、类型和地址指针)

- **举例**：有幅大小为width*height，格式为Win.系统中使用最普遍的24位BGR，保存在pixels中的像素图象,载入为纹理的命令
  - glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_BGR_EXT, GL_UNSIGNED_BYTE, pixels);

纹理坐标

- 绘制一个三角形时：
  - 只需要指定三个顶点的颜色
  - 系统会自动计算其它点的颜色
- 纹理的使用方法也与此类似：
  - 只要指定每个顶点在纹理图象中所对应的像素位置，OpenGL会自动计算顶点以外的其它点在纹理图象中所对应的像素位置

纹理坐标

- 绘制一条线段，设置一端点为红色，另端点为绿色
  - glShadeMode(GL_SMOOTH) ：渐变
  - glShadeMode(GL_FLAT) ：单一
- 纹理映射：
  - 一端用“纹理图象中最左下角的颜色”作为颜色，另一端用“纹理图象中最右上角的颜色”作为它的颜色，则OpenGL会自动在纹理图象中选择合适位置的颜色，填充到线段的各个像素

纹理坐标

- 纹理坐标：二维纹理的精确表示法
  - 最左下角坐标(0, 0)，最右上角坐标(1, 1)
  - 每个像素位置可用两个浮点数来表示
  - glTexCoord*系列函数来指定纹理坐标

```
glBegin( /* ... */ );
 glTexCoord2f(/*...*/);glVertex3f(/*...*/);
 /* ... */
glEnd();
```

  - 切换到纹理矩阵后可进行各种（旋转、平移等）变换glMatrixMode(GL_TEXTURE)

纹理参数

- 纹理参数的设置：glTexParameter*
  - GL_TEXTURE_MAG_FILTER：指当纹理图象被使用到一个大于它的形状上时应该如何处理：
    - GL_NEAREST：选最接近的一个像素的颜色作为需要绘制的像素颜色，会出现锯齿
    - GL_LINEAR：使用纹理中坐标最接近的若干个颜色，通过加权平均算法得到需要绘制的像素颜色，效果较好，运算量大

纹理参数

- GL_TEXTURE_MIN_FILTER：指当纹理图象被使用到一个小于（或等于）它的形状上时应该如何处理：
  - GL_NEAREST，GL_LINEAR，GL_NEAREST_MIPMAP_NEAREST，GL_NEAREST_MIPMAP_LINEAR，GL_LINEAR_MIPMAP_NEAREST和GL_LINEAR_MIPMAP_LINEAR

    后四个涉及到mipmap，前两个选项则和GL_TEXTURE_MAG_FILTER中的类似，此参数似乎是必须设置的

纹理参数

- GL_TEXTURE_WRAP_S：指当纹理坐标第一维坐标值大于1.0或小于0.0时，应该如何处理
  - GL_CLAMP：前者表示“截断”，即超过1.0的按1.0处理，不足0.0的按0.0处理
  - GL_REPEAT：后者表示“重复”，即对坐标值加上一个合适的整数得到一个在[0.0, 1.0]范围内的值，然后用这个值作为新的纹理坐标
- GL_TEXTURE_WRAP_T：当纹理坐标的第二维坐标值大于1.0或小于0.0时，应该如何处理
- 参数设置代码：

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT)
```

纹理对象

- 纹理数据通常比较大，载入过程较慢
  - 把这些像素数据从主内存传送到专门的图形硬件，可能还涉及格式转化
- 应尽量减少载入纹理的次数
  - 如只有一幅纹理，则应该在第一次绘制前就载入它，以后就不需要再次载入了
  - 在每次绘制时候需要使用两幅或多幅纹理时，如果使用下面的代码，效率非常低

```
glTexImage2D(/* ... */); //载入一幅纹理
// 使用一幅纹理
```

  - 需要有一种机制，能够在不同的纹理之间进行快速地切换

纹理对象

- 把每一幅纹理放到一个纹理对象中，通过创建多个纹理对象来达到同时保存多幅纹理的目的
  - 在第一次使用纹理前，把所有的纹理都载入，然后在绘制时只需要指明究竟使用哪一个纹理对象就可以了
  - 使用纹理对象和显示列表有相似之处：使用一个正整数来作为纹理对象的编号

纹理对象

- 分配纹理对象：glGenTextures
  - 分配一个纹理对象的编号

```
GLuint texture_ID;
glGenTextures(1, &texture_ID);
```

  - 分配5个纹理对象的编号

```
GLuint texture_ID_list[5];
glGenTextures(5, texture_ID_list);
```

  - 零：默认的纹理对象，不会分配该编号
- 销毁纹理对象：glDeleteTextures

纹理对象

- glBindTexture指定当前所用纹理对象：
  - 两个参数：GL_TEXTURE_2D，编号
  - 后续操作：
    - glTexImage*函数指定纹理像素
    - glTexParameter*函数指定纹理参数
    - glTexCoord*函数指定纹理坐标
  - 如不使用glBindTexture函数，上述函数默认在编号为0的纹理对象上进行操作
  - 该函数可实现不同纹理之间进行切换，避免反复载入纹理

纹理对象

- 典型代码：
  - // 在程序开始时：分配好纹理编号，并载入纹理

```
glGenTextures( /* ... */ );
glTexImage2D(/* ... */);
// 载入第一幅纹理
glTexImage2D(/* ... */);
// 载入第二幅纹理
```

  - // 在绘制时，切换并使用纹理，不需要再进行载入

```
glBindTexture(GL_TEXTURE_2D, texture_ID_1); // 指定第一幅纹理
// 使用第一幅纹理
glBindTexture(GL_TEXTURE_2D, texture_ID_2); // 指定第二幅纹理
// 使用第二幅纹理
```

纹理映射

- 例：给定纹理，实现映射

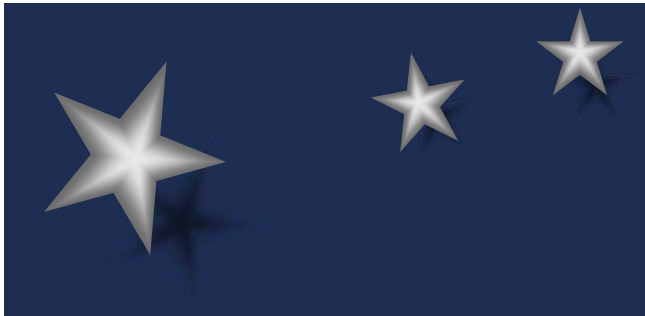


- 代码

```
%创建几何体对象
GLUquadricObj* qObj = gluNewQuadric();
%绑定纹理
gluQuadricTexture(quadObj, GL_TRUE);
glBindTexture(GL_TEXTURE_2D, texName);
%二次曲面qobj
gluSphere(qObj, 2, 720, 450);
%销毁二次方程对象
gluDeleteQuadric(quadObj);
```

小结

- 利用纹理可以进行比像素操作命令更复杂的像素绘制
- 纹理包括1,2,3维纹理
- 使用纹理前，要启用纹理：注意纹理高度，最好是2的整次方，如不满足可用gluScaleImage进行缩放
- 载入纹理的开销比较大的，应该尽可能减少载入次数



OpenGL 编程

片段测试

片段测试

- 片段测试：测试每一个像素，只有通过测试的像素才会被绘制
- OpenGL提供了以下几种测试
  - 剪裁测试
  - Alpha测试
  - 模板测试
  - 深度测试

剪裁测试

- 剪裁测试：只有位于指定矩形内部的像素才能通过测试
  - 指定一个矩形的剪裁窗口
  - 启用剪裁测试后，只有在这个窗口之内的像素才被绘制，其它像素则会被丢弃
- 应用：《魔兽争霸3》
  - 如选中某士兵，则画面下方的一个方框内就会出现该士兵的头像
  - 剪裁测试保证了该头像无论如何绘制都不会越界而覆盖到外面的像素

剪裁测试

- 启用或禁用剪裁测试：
  - glEnable(GL_SCISSOR_TEST)
  - glDisable(GL_SCISSOR_TEST)
- 指定剪裁窗口：
  - glScissor(x, y, width, height);
  - OpenGL窗口坐标与Windows系统窗口有所不同

Alpha测试

- **Alpha测试：**只有Alpha值与设定值相比较，满足特定关系条件的像素才能通过测试
  - 始终通过（默认情况）、始终不通过、大于设定值则通过、小于设定值则通过、等于设定值则通过、大于等于设定值则通过、小于等于设定值则通过、不等于设定值则通过

Alpha测试

- 启用或禁用Alpha测试
  - glEnable(GL_ALPHA_TEST)
  - glDisable(GL_ALPHA_TEST)
- 设置Alpha测试条件
  - glAlphaFunc(GL_GREATER, 0.5f)
    - GL_ALWAYS（始终通过）
      GL_NEVER（始终不通过）
      GL_LESS（小于则通过）
      GL_LEQUAL（小于等于则通过）
      GL_EQUAL（等于则通过）
      GL_GEQUAL（大于等于则通过）
      GL_NOTEQUAL（不等于则通过）

Alpha测试

- 例子：一幅照片图片，一幅相框图片，如何将它们组合在一起呢？
  - 网络游戏《魔兽世界》的一幅桌面背景
  - 先绘制图片，再绘制相框的不透明部分

Alpha测试

- Alpha测试 Vs 混合
  - 透明或不透明，采用Alpha测试；需要绘制半透明像素时，可采用混合
  - 效率高：只比较大小，不需要计算
  - 裁剪/Alpha/模板和深度测试：透明像素不需要通过后两个测试，混合则不然
  - “透明”像素：未通过Alpha测试，深度值不会被改变，混合操作会改变深度

模板测试

- 模板测试：像素模板值与设定值比较，满足特定关系条件的像素才能通过测试
  - 模板缓冲区在 OpenGL初始化时指定，如使用GLUT工具包
    - glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_STENCIL)
    - Windows操作系统中，即使没有明确要求使用模板缓冲区，有时候也会分配模板缓冲区
    - 如确实没有分配，则所有像素全部通过该测试

模板测试

- 启用或禁用模板测试
  - glEnable(GL_STENCIL_TEST)
  - glDisable(GL_STENCIL_TEST)
- 模板测试过程：
  - 模板缓冲区中为每个像素保存了一个“模板值”，模板测试时，将设定的模板参考值与该像素的“模板值”进行比较，符合条件的通过测试，不符合条件的则被丢弃，不进行绘制
  - 条件的设置与Alpha测试中的条件设置相似
  - Alpha测试中是用浮点数，此测试用整数

模板测试

- 测试：
  glStencilFunc(GL_LESS, 3, mask)
  - 参数1，2：与glAlphaFunc参数类似
  - 参数3：如进行比较，则只比较mask中二进制为1的位
    - 例如，某个像素模板值为5（二进制101），而mask的二进制值为00000011，因为只比较最后两位，5的最后两位为01，其实是小于3的，因此会通过测试

模板测试

- 设置模板值：glClear函数
  - 可将所有像素的模板值复位
    - glClear(GL_STENCIL_BUFFER_BIT)
  - 同时复位颜色值和模板值
    - glClear(GL_COLOR_BUFFER_BIT | GL_STENCIL_BUFFER_BIT)
  - 使用glClearStencil函数来指定复位后的“模板值”

模板测试

- 每个像素的“模板值”会根据模板测试的结果和深度测试的结果而进行改变

  glStencilOp(fail, zfail, zpass)

  – 指定了三种情况下“模板值”该如何变化：
    - 参数1：模板测试未通过时该如何变化
    - 参数2：模板测试通过，但深度测试未通过时该如何变化
    - 参数3：表示模板测试和深度测试均通过时该如何变化

  如没有起用模板测试，则认为模板测试总是通过

模板测试

- 变化可以是：

GL_KEEP（不改变，这也是默认值）
GL_ZERO（回零）
GL_REPLACE（使用测试条件中的设定值来代替当前模板值）
GL_INCR（增加1，但如果已经是最大值，则保持不变）
GL_INCR_WRAP（增加1，但如果已经是最大值，则从零重新开始）
GL_DECR（减少1，但如果已经是零，则保持不变），
GL_DECR_WRAP（减少1，但如果已经是零，则重新设置为最大值）
GL_INVERT（按位取反）

模板测试

- 新版OpenGL：
  - 允许为多边形的正面和背面使用不同的模板测试条件和模板值改变方式，
  - glStencilFuncSeparate函数和glStencilOpSeparate函数。对应于glStencilFunc和glStencilOp
  - 只在最前面多了一个参数face，用于指定当前设置的是哪个面。可以选择GL_FRONT GL_BACK, GL_FRONT_AND_BACK

模板测试

- 模板缓冲区 **Vs** 深度缓冲区
  - 无论是否开启深度测试，像素被绘制时，总会重新设置该像素的深度值，除非设置glDepthMask(GL_FALSE)
  - 模板测试如果不启用，则像素的模板值会保持不变
- 尽量不使用glStencilFuncSeparate和glStencilOpSeparate函数（时间）

模板测试

- 例：把绘制区域限制在一个不规则的区域内。如绘制一个湖泊、周围的树木及倒影。为了保证倒影被正确的限制在湖泊表面
  - 关闭模板测试，绘制地面和树木
  - 开启模板测试，使用glClear设置所有像素模板值为0
  - 设置glStencilFunc(GL_ALWAYS, 1, 1)
    glStencilOp(GL_KEEP, GL_KEEP, GL_REPLACE);
    绘制湖泊水面。这样一来，湖泊水面的像素的“模板值”为1，而其它地方像素的“模板值”为0
  - 设置glStencilFunc(GL_EQUAL, 1, 1);
    glStencilOp(GL_KEEP, GL_KEEP, GL_KEEP)绘制倒影

模板测试

- 例子：空间中有一个球体，一个平面镜。我们站在某个特殊的观察点，可以看到球体在平面镜中的镜像，并且镜像处于平面镜的边缘，有一部分因为平面镜大小的限制，而无法显示出来

深度测试

- **深度测试：**只有像素深度值与新的深度值比较，满足特定关系条件的像素才能通过测试
  - 缓冲区保存像素的某个值，当需要进行测试时，将保存的值与另一个值进行比较，以确定是否通过测试
  - 如用GLUT包，调用glutInitDisplayMode函数时在参数中加上GLUT_DEPTH

深度测试

- 深度测试 Vs 模板测试
  - 在缓冲区保存像素某个值，将保存值与另一个值进行比较，确定是否通过测试
  - 前者根据顶点空间坐标计算出深度，用这个深度与像素的“深度值”进行比较，
  - 后者设定一个值，在测试时用这个设定值与像素“模板值”进行比较
  - 前者应用频繁，几乎所有三维场景绘制都使用了深度测试， OpenGL所有实现都对深度测试提供了硬件支持

深度测试

- 启用或禁用深度测试
  - glEnable(GL_DEPTH_TEST)
  - glDisable(GL_DEPTH_TEST)
- 测试条件
  - 与Alpha测试中的条件设置相同（八种）
  - 用glDepthFunc函数
    glDepthFunc(GL_LESS)

小结

- OpenGL提供的测试：剪裁测试、Alpha测试、模板测试、深度测试
- OpenGL会对每个即将绘制的像素进行以上四种测试，每个像素只有通过一项测试后才会进入下一项，而只有通过所有测试的像素才会被绘制，没有通过测试的像素会被丢弃掉，不进行绘制
- 每项测试都可以单独开启或关闭，如某项测试被关闭，则所有像素都可顺利通过该测试