高级计算机图形学

中国科学技术大学计算机学院
黄章进
zhuang@ustc.edu.cn

第八章之第三节

OpenGL中的纹理映射

基本内容

- 介绍OpenGL中的纹理函数以及相应的选项

OpenGL与纹理

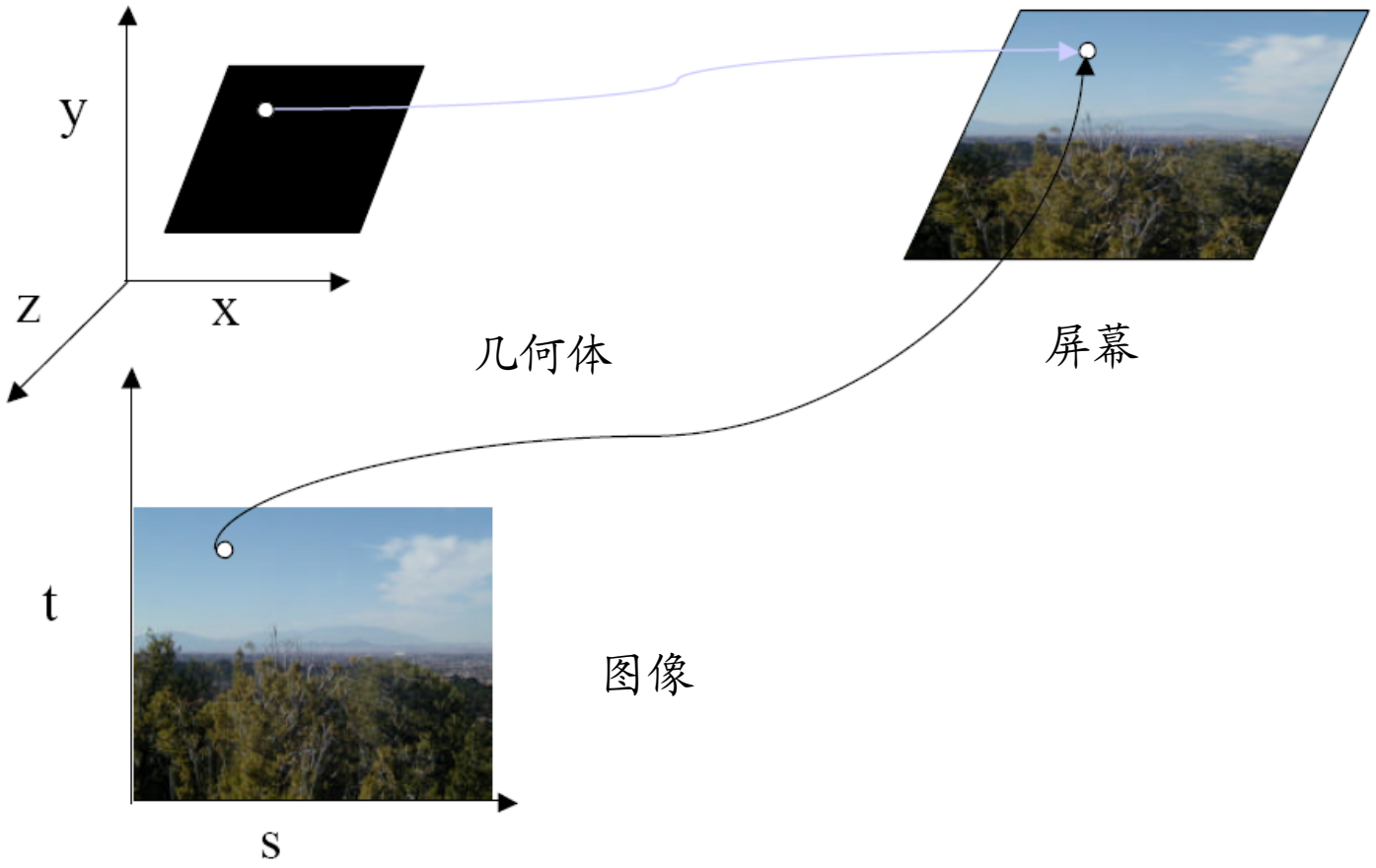
- OpenGL支持许多纹理映射选项
  - 第一版包含了把一维或二维纹理映射到一维至四维图形对象的函数
  - 现在的版本提供了三维纹理，但需要高端硬件的支持
- 本节只讨论从二维纹理到曲面的映射

基本策略

应用纹理需要下面三个步骤

- 指定纹理
  - 读入或生成图像
  - 赋给纹理
  - 激活纹理映射功能
- 给每个顶点赋纹理坐标
  - 由应用程序建立适当的映射函数
- 指定纹理参数
  - 环绕(wrapping), 滤波(filtering)

纹理映射

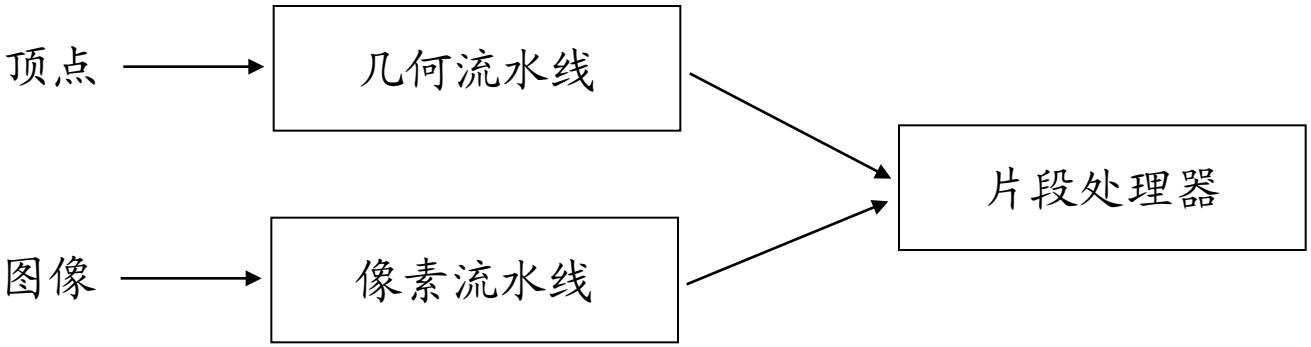


纹理示例

- 纹理(下方)是 256 × 256 的图像，它被映射到一个矩形上，经透视投影后的结果显示在上方

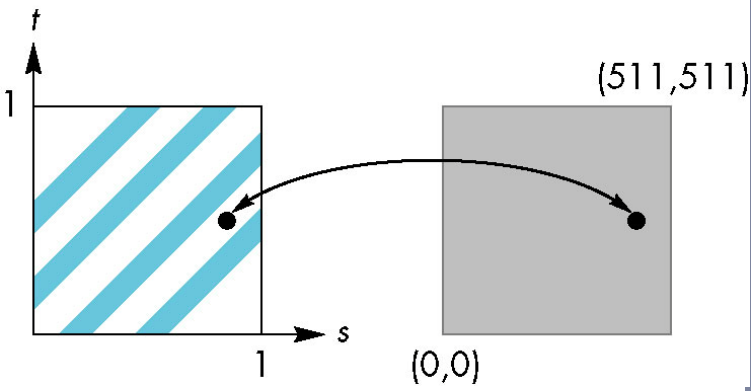
纹理映射与OpenGL流水线

- 图像与几何分别经过不同的流水线，在片段处理时合二为一
  - “复杂的”纹理并不影响几何的复杂性



指定纹理图像

- 利用CPU内存中的纹理元素数组定义纹理图像
  - **GLubyte my_texels[512][512];**
- 定义纹理图像所用的像素图
  - 扫描图像
  - 由应用程序代码创建
- 激活纹理映射
  - **glEnable(GL_TEXTURE_2D);**
  - OpenGL支持一至四维纹理映射

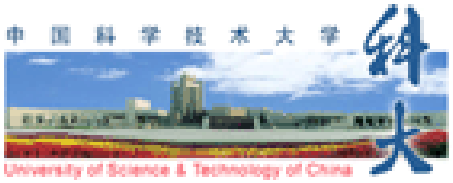


把图像定义为纹理

*void **glTexImage2D**(GLenum target, GLint level, GLint internalFormat, GLsizei width, GLsizei height, GLint border, GLenum format, GLenum type, const GLvoid *pixels);*

- target: 纹理的类型，如GL_TEXTURE_2D
- level: 用于mipmapping，0表示单分辨率纹理图像
- internalFormat: 纹素的分量格式，GL_RGB，GL_RGBA等
- width和height: pixels的宽度和高度，$2^m + 2b$，b为边框宽
- border: 边框宽度，0（没有边框）或1
- format: 图像数据pixels的格式， GL_RGB，GL_RGBA等
- type: pixels分量的数据类型， GL_BYTE, GL_INT等
- pixels: 指向纹素数组的指针，描述纹理图像及边框

转化纹理图像




```
glTexImage2D(GL_TEXTURE_2D,0,GL_RGB,512,512,0,
   GL_RGB,GL_UNSIGNED_BYTE,my_texels);
```

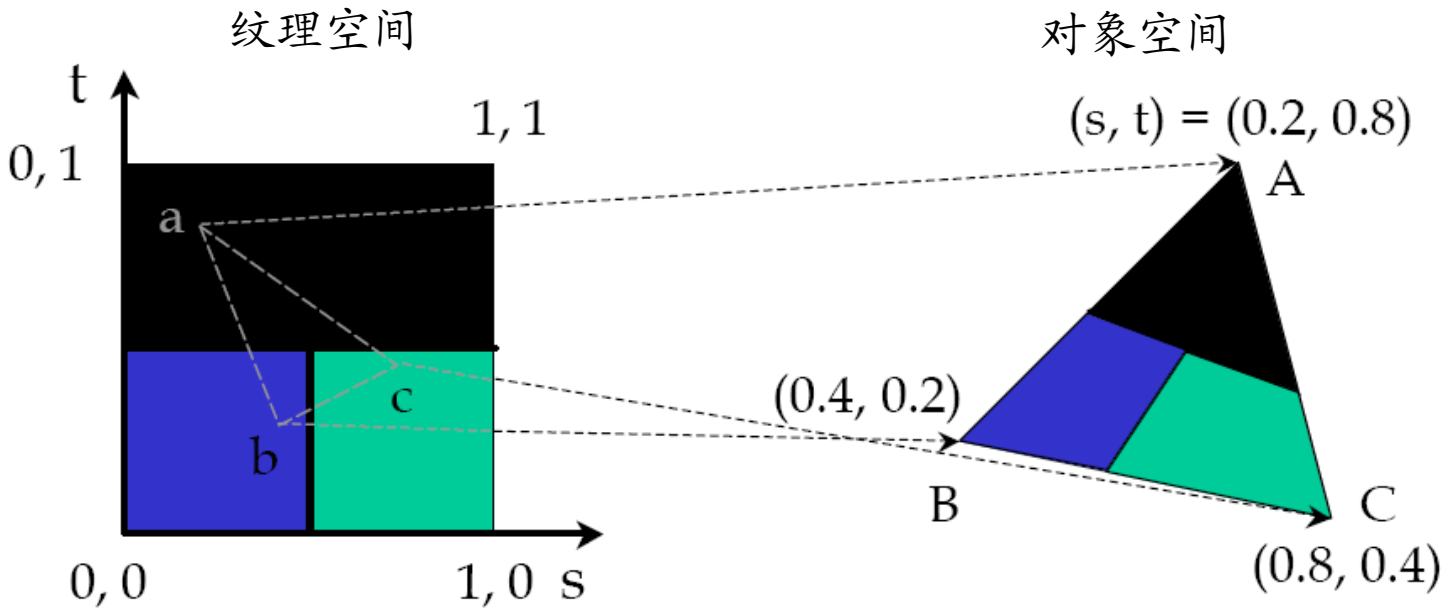
- OpenGL需要纹理的尺寸（不包括可选的边框宽度）为2的幂次（OpenGL2.0以上无此限制）
- 如果不是2的幂次，可用下述函数进行转化

*int **gluScaleImage**(GLenum format, GLint widthin, GLint heightin, GLenum typein, const void *datain, GLint widthout, GLint heightout, GLenum typeout, void *dataout);*

  - *datain* 源图像
  - *dataout* 目标图像
- 当放缩时图像被插值和滤波

映射纹理

- 基于参数纹理坐标，是状态量，内部表示为四维(s,t,r,q)。取值于[0,1]，维数小于4时，默认有t=r=0, q=1
- **glTexCoord*()**指定每个顶点对应的纹理坐标



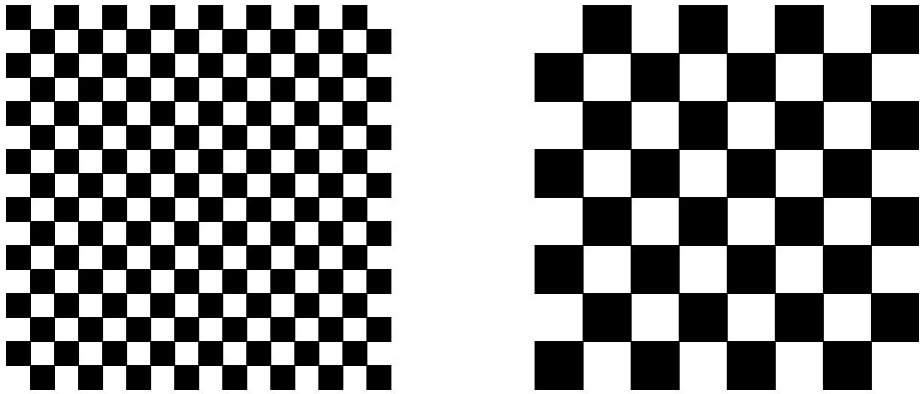
典型代码


```
glBegin(GL_POLYGON);
   glColor3f(r0, g0, b0); //if no shading used
   glNormal3f(u0, v0, w0); // if shading used
   glTexCoord2f(s0, t0);
   glVertex3f(x0, y0, z0);
   glColor3f(r1, g1, b1);
   glNormal3f(u1, v1, w1);
   glTexCoord2f(s1, t1);
   glVertex3f(x1, y1, z1);
        .
        .
glEnd();
```

注意为了提高效率，可以采用顶点数组

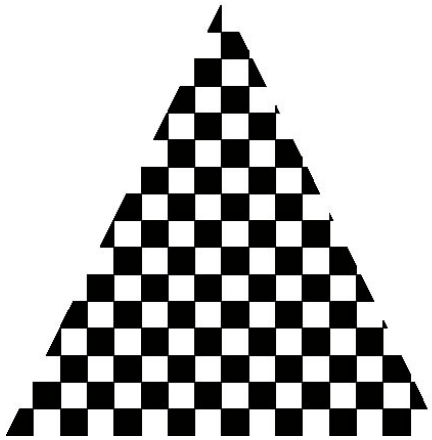
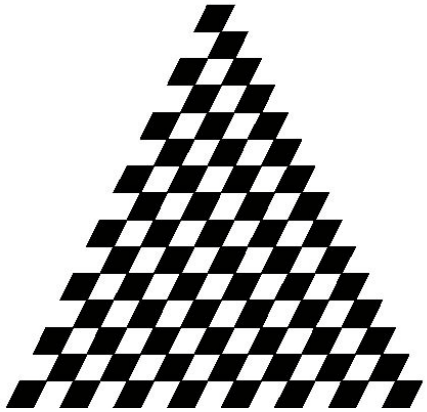
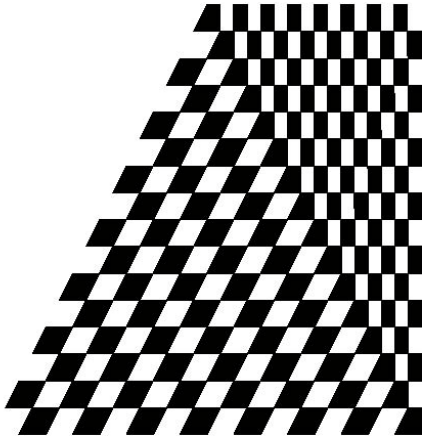
插值

- OpenGL应用双线性插值从给定的纹理坐标中求出适当的纹素
- 可以只应用纹理的一部分
  - 方法是只应用纹理坐标的一部分，如最大纹理坐标为(0.5,0.5)

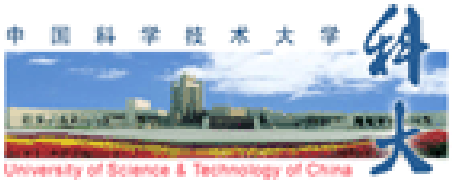


变形

- 对于四边形，从纹理坐标到顶点的对应是比较直接的
- 对于一般的多边形，程序员必须决定如何给顶点赋纹理坐标
  - 可能会出现变形

纹理参数

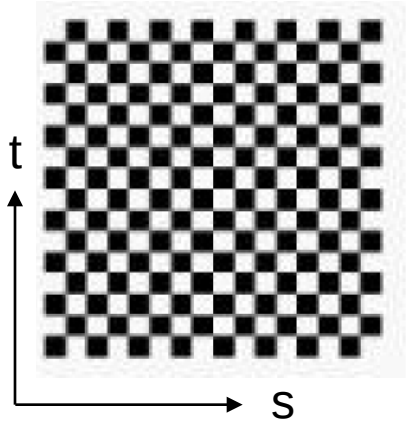


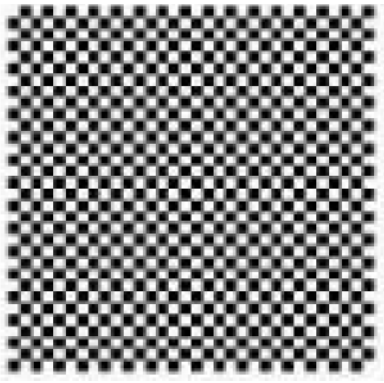
- OpenGL中有许多参数来确定纹理的应用方式
  - Wrapping参数确定当s, t的值超出[0,1]区间后的处理方法
  - Filter模式允许用区域平均方法来代替点采样方法
  - Mimmapping技术使得能以不同的分辨率应用纹理
  - 环境参数确定纹理映射与明暗处理的交互作用

Wrapping模式

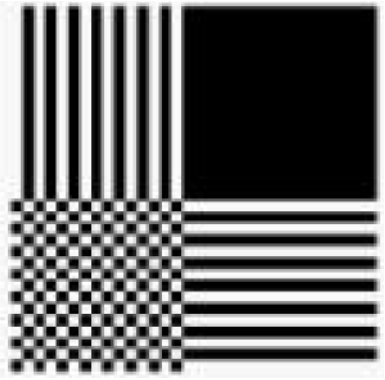
- 钳位（截断）： 若s,t>1就取1， 若s,t<0就取0
- 重复: 应用s,t模1的值

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





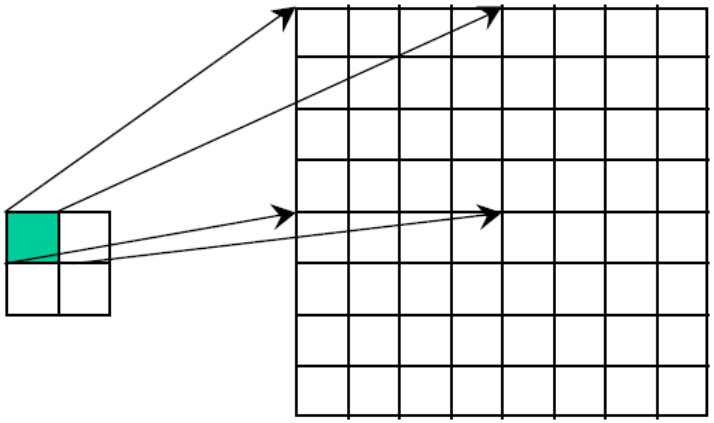
GL_REPEAT



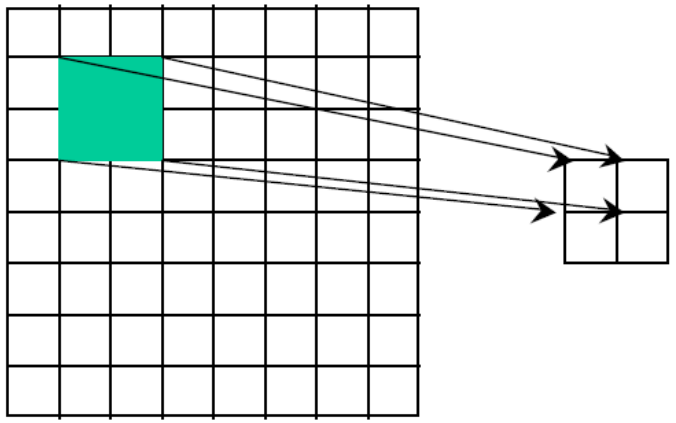
GL_CLAMP

纹理的放大和缩小

- 放大：多个像素对应一个纹素
- 缩小：多个纹素对应一个像素



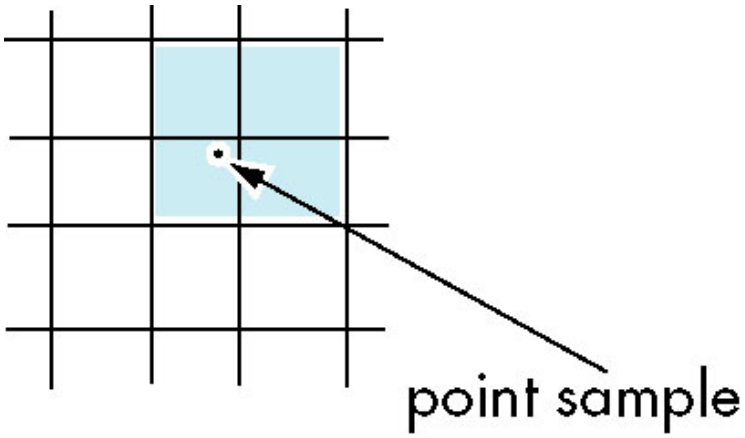
放大



缩小

解决方法

- 点采样：最近纹素的纹理值
- 线性滤波：最近点邻域（2 x 2）纹素加权平均的纹理值



滤波模式

滤波模式的指定

- **glTexParameteri(target, type, mode)**

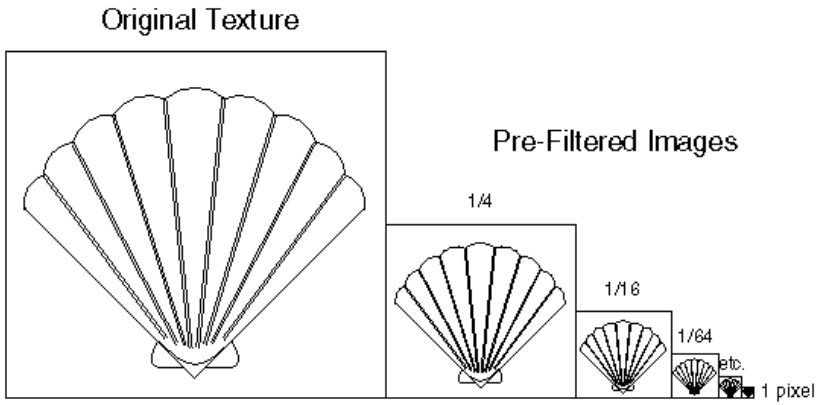
```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
                GL_NEAREST);

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

注意线性滤波需要一个额外的纹素边框来在边界处进行滤波，即 (border = 1)

纹理的Mipmap

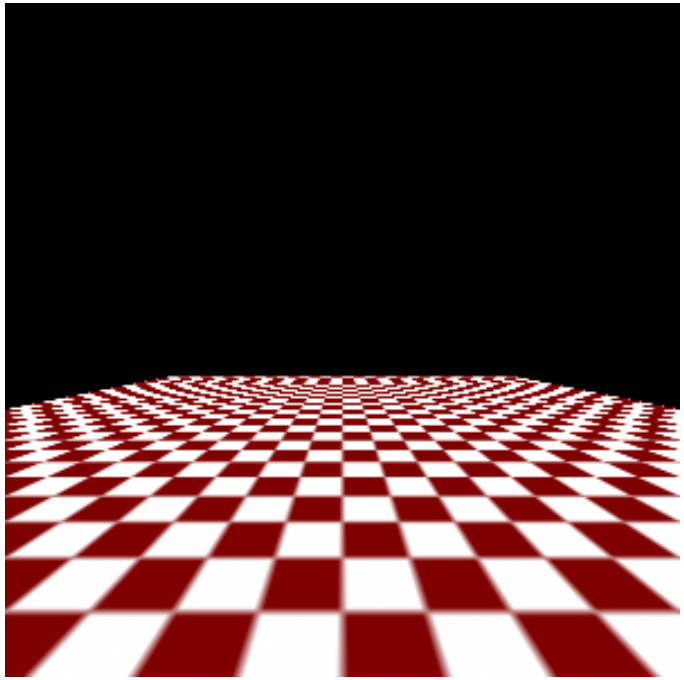
- Mipmap对纹理位图进行预先滤波，降低分辨率
- 可以减小对于非常小的要加纹理的对象的插值误差
- 在纹理定义时声明mipmap的层次，level=0,1,…

```
glTexImage2D( GL_TEXTURE_2D, level, width,
  height, … )
```

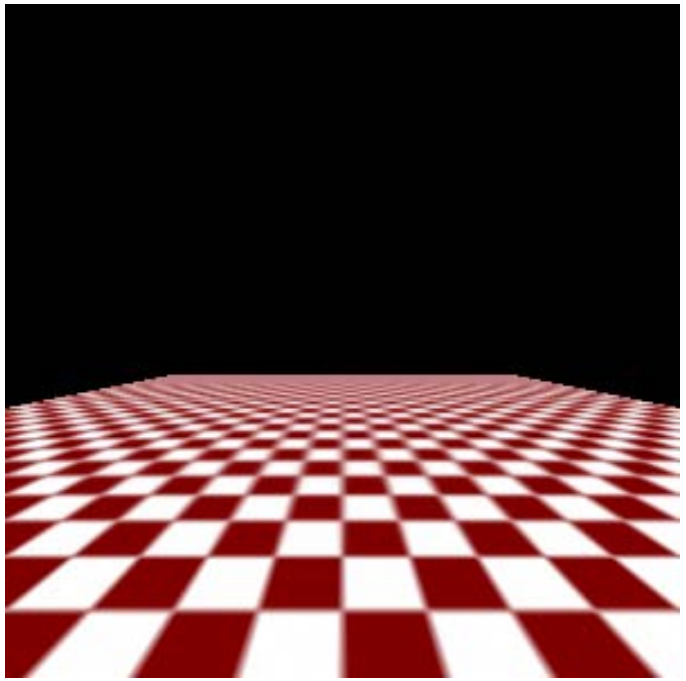



```
GLubyte image0[64][64][3];
GLubyte image1[32][32][3];
…
GLubyte image5[1][1][3];
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 64, 64,
    GL_RGB, GL_UNSIGNED_BYTE, image0);
…
glTexImage2D(GL_TEXTURE_2D, 5, GL_RGB, 1, 1,
    GL_RGB, GL_UNSIGNED_BYTE, image5);

glTexParameteri(GL_TEXTURE_2D, GL_TEXURE_MIN_FILTER,
                GL_NEAREST_MIPMAP_NEAREST);
```

有无mipmap的对比



无mipmap



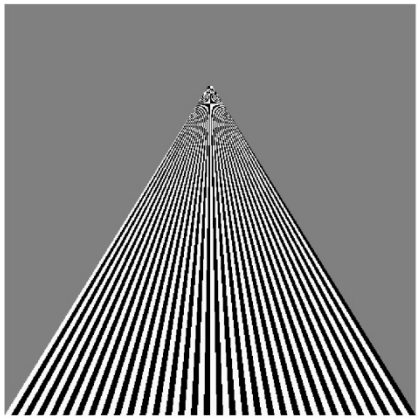
有mipmap

自动生成mipmap

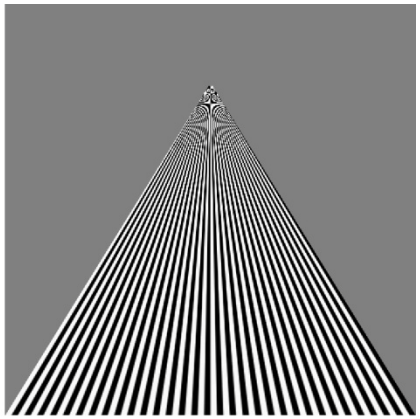
- GLU中提供了mipmap建立界面，可以从给定图像建立起所有的mipmap纹理
  - gluBuild2DMipmaps(…)
- 缩小滤波器参数:
  - GL_NEAREST_MIPMAP_NEAREST
  - GL_NEAREST_MIPMAP_LINEAR
  - GL_LINEAR_MIPMAP_NEAREST
  - GL_LINEAR_MIPMAP_LINEAR

示例

点采样



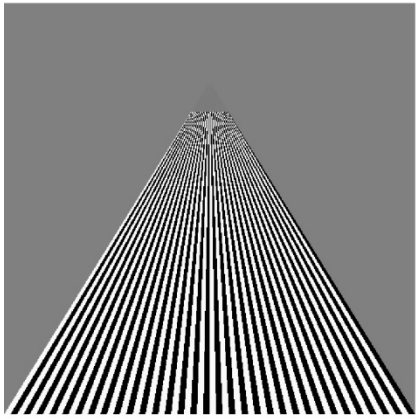
(a)



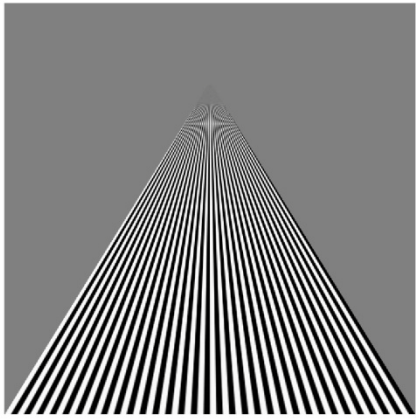
(b)

线性滤波

mipmapped
后的点取样



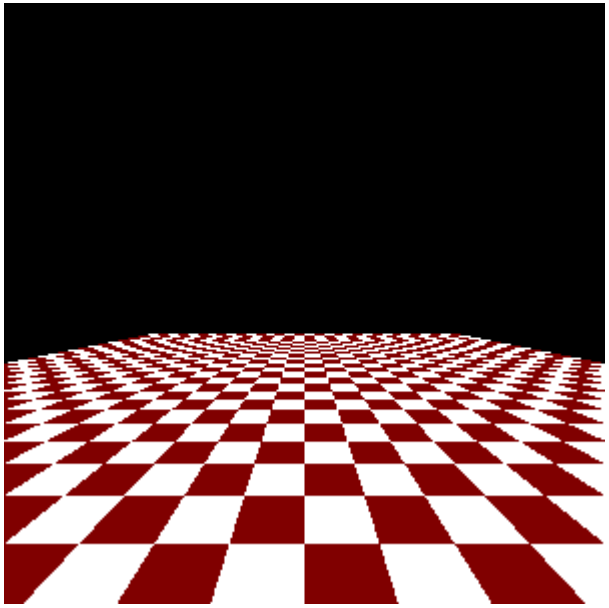
(c)



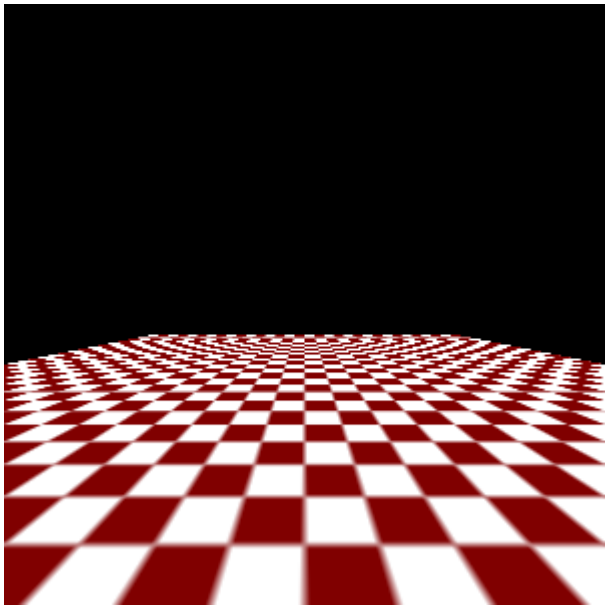
(d)

mipmapped
的线性滤波

更多例子

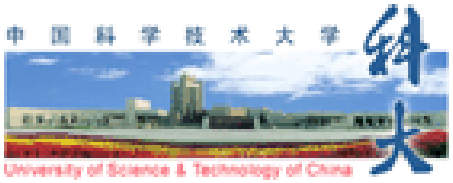


GL_NEAREST



GL _ LINEAR

纹理环境函数



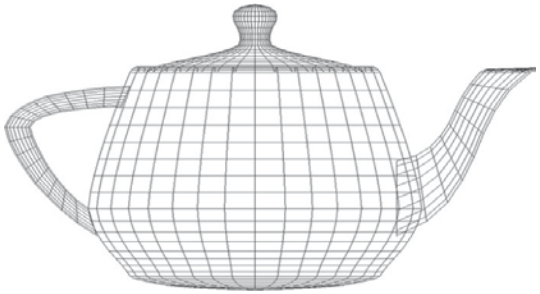
- 控制纹素和片段颜色如何融合
  - glTexEnv{fi}[v](GL_TEXTURE_ENV, mode, param);
  - GL_TEXTURE_ENV_MODE 设置模式
    - GL_MODULATE: 与计算的明暗值调制在一起
    - GL_BLEND: 与环境颜色融合在一起
    - GL_REPLACE: 只应用纹理的颜色
    - GL_DECAL: 与GL_REPLACE类似
  - 应用GL_TEXTURE_ENV_COLOR设置融合颜色

透视校正提示

- 纹理映射还与投影类型相关
- 纹理坐标与颜色插值
  - 要么是关于屏幕空间线性确定的
  - 或者要应用深度/透视值(较慢)
- 在多边形边界上，走样现象非常明显
  - **glHint(GL_PERSPECTIVE_CORRECTION_HINT, hint)**
  - 其中hint可取值GL_DONT_CARE, GL_NICEST, GL_FASTEST

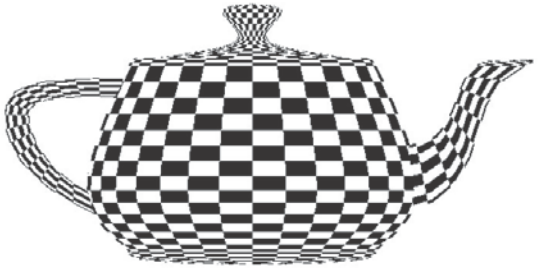
纹理坐标设置

- 纹理坐标内部表示为四维向量，可以用纹理变换矩阵进行变换（缩放、旋转），得到纹理与对象、照相机或光源一起运动的效果



```
glMatrixMode(GL_TEXTURE);
```

- 用glTexCoord*()函数给多边形网格赋恰当的纹理坐标并不容易



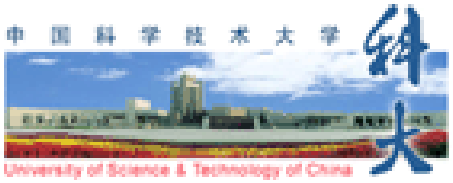
自动生成纹理坐标

- OpenGL能自动生成纹理坐标

*void **glTexGen**{ifd}(GLenum coord, GLenum pname, TYPE param);*
*void **glTexGen**{ifd}**v**(GLenum coord, GLenum pname, TYPE *param);*

  - coord：纹理坐标，可取GL_S, GL_T, GL_R, 或 GL_Q
  - pname：纹理坐标的生成函数，取
    - GL_TEXTURE_GEN_MODE
    - GL_OBJECT_PLANE, GL_EYE_PLANE
  - param：当pname为GL_TEXTURE_GEN_MODE时，取
    - GL_OBJECT_LINEAR
    - GL_EYE_LINEAR
    - GL_SPHERE_MAP：用于环境映射
    - 否则应为数组，由纹理生成平面的系数组成

自动生成纹理坐标（续）



- 指定平面a x + b y + c z + d w = 0
  - 根据顶点(x,y,z,w)到平面的距离生成纹理坐标(s,t)，由于ax+by+cz+dw与该距离成比例，有

$$s = a_s x + b_s y + c_s z + d_s w$$

$$t = a_t x + b_t y + c_t z + d_t w$$

```
GLfloat planes[]={0.5,0.0,0.0,0.5}; /* s=x/2+1/2 */
GLfloat planes[]={0.0,0.5,0.0,0.5}; /* t=y/2+1/2 */
glTexGeni(GL_S,GL_TEXTURE_GEN_MODE,GL_OBJECT_LINEAR);
glTexGeni(GL_T,GL_TEXTURE_GEN_MODE,GL_OBJECT_LINEAR);
glTexGenfv(GL_S, GL_OBJECT_LINEAR, planes);
glTexGenfv(GL_T, GL_OBJECT_LINEAR, planes);
glEnable(GL_TEXTURE_GEN_S);
glEnable(GL_TEXTURE_GEN_T);
```

对象坐标系和视点坐标系

- (a)对象坐标系，纹理固定在对象上
- (b)视点坐标系，纹理图案随变换变化，感觉像对象穿过纹理场

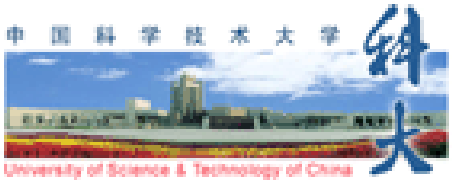
(a)

(b)

纹理对象

- 纹理也是状态的一部分，每类纹理（指1D,2D,3D）只有一个当前纹理
  - 每次执行glTexImage2D时，另一个纹理图像被载入纹理内存并替换掉原来的纹理
  - 如果不同的对象具有不同的纹理，那么OpenGL需要从处理器内存向纹理内存传送大量数据
- 最新版本的OpenGL提供了纹理对象功能
  - 每个纹理对象是一个图像
  - 纹理内存可以保存多个纹理对象

纹理对象（续）



- *void **glGenTextures**(GLsizei n, GLuint *textureNames);*
  - 产生n个未使用的纹理对象名称
- *GLboolean **glIsTexture**(GLuint textureName);*
  - 检查一个纹理名称是否被使用

- *void **glBindTexture**(GLenum target, GLuint textureName);*
  - 创建或切换纹理对象，target为GL_TEXTURE_2D等
- *void **glDeleteTextures**(GLsizei n, const GLuint *textureNames);*
  - 删除纹理对象

综述：应用纹理框架

- 在纹理对象中指定纹理
- 设置纹理滤波方式
- 设置纹理环境函数
- 设置纹理的wrap模式
- 设置可选的透视校正提示
- 绑定纹理对象
- 激活纹理映射功能
- 为每个顶点指定纹理坐标
  - 纹理坐标可以自动生成

多重纹理

- 通常一个几何对象上只有一个纹理
- 有许多渲染效果需要多次应用纹理
  - 例：已有纹理的表面上有其它物体的阴影
  - 此时需要给阴影加上纹理
- 多重纹理的工作流程

初始化时，定义了两个纹理对象object0和object1

```
glActiveTexture(GL_TEXTURE0); /* unit 0 */
glEnable(GL_TEXTURE_2D);
glBindTexture(GL_TEXTURE_2D, object0);
glTexEnvironment(…); /* how to apply texture 0 */
glActiveTexture(GL_TEXTURE1); /* unit 1 */
glEnable(GL_TEXTURE_2D);
glBindTexture(GL_TEXTURE_2D, object1);
glTexEnvironment(…); /* how to apply texture 1 */
```

每个纹理单元使用不同的纹理坐标

```
GLfloat point[3], s, t, ss, tt;
glMultiTexCoord2f(GL_TEXTURE0, s, t);
glMultiTexCoord2f(GL_TEXTURE1, ss, tt);
glVertex3fv(point);
```

纹理生成

- 纹理的主要一个应用就是不必应用几何建模方法，而提供几何对象上的细节
- 高端图形系统可以实时生成三维纹理映射
  - 对于每一帧图像，纹理被映射到对象上，几乎与不进行纹理映射的对象显示频率相同
- 现在的微机显卡也包含了大量纹理内存，从而可以使得游戏开发人员应用纹理映射的方法创建复杂的动画效果