XIAMEN UNIVERSITY

COMPUTER GRAPHICS

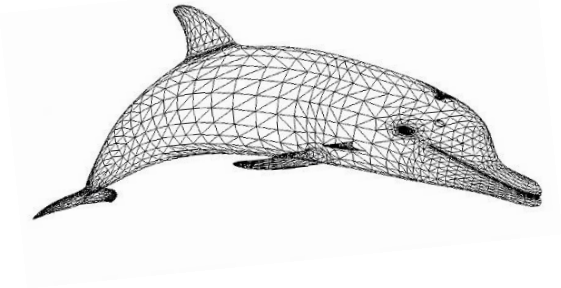
第五章纹理贴图

陈中贵
厦门大学信息学院
http://graphics.xmu.edu.cn

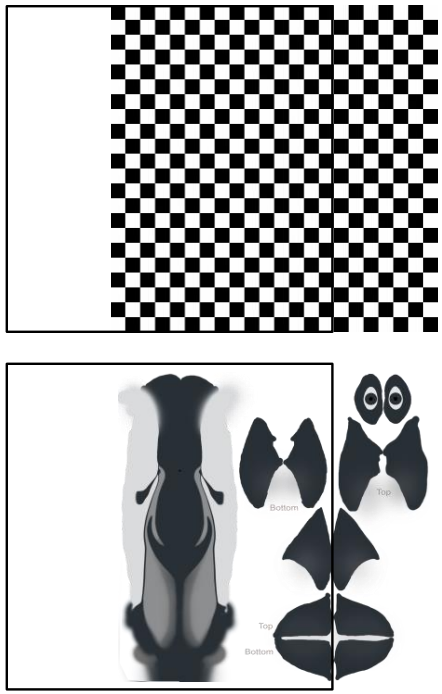
纹理贴图 = 在模型表面上覆盖图像

- 使用两张不同的图像给同一个海豚模型添加纹理

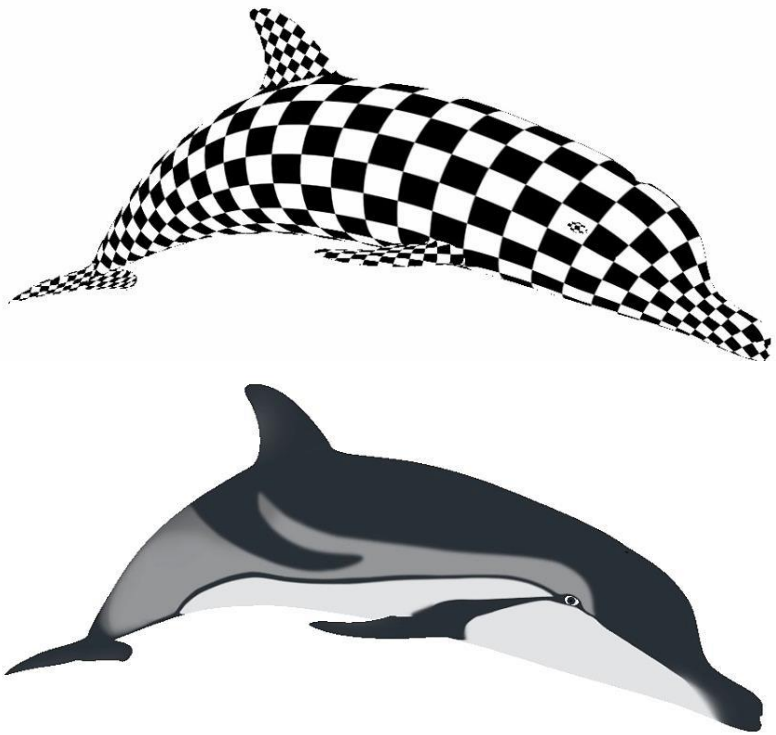
三角网格模型



纹理图片

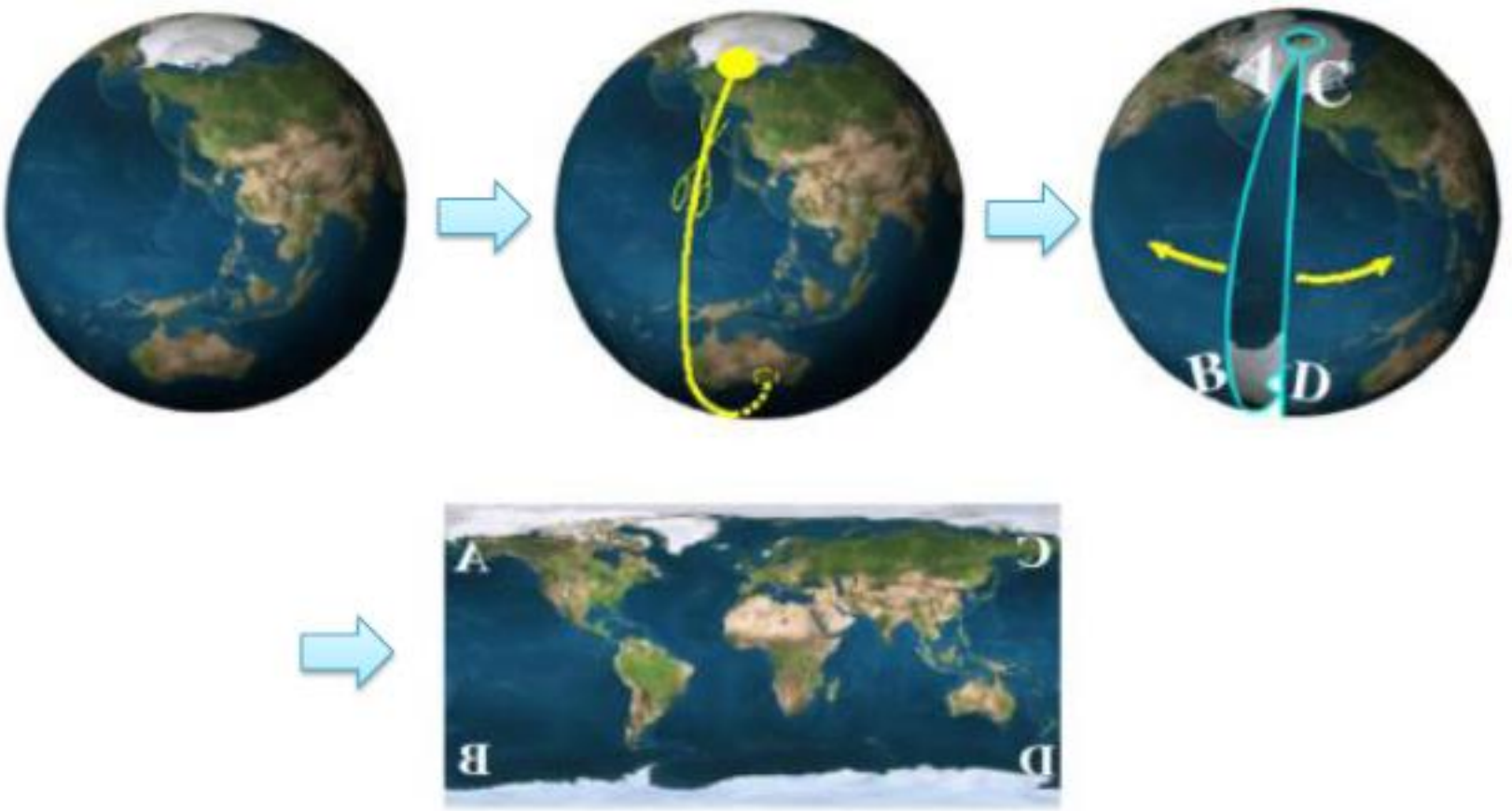


添加纹理后的模型



纹理映射

- 以世界地图为例：



数据和机制

- 为了在 OpenGL/GLSL 中有效地完成纹理贴图，需要协调好以下几个不同的数据和机制
  - 纹理图像
  - 用于保存纹理图像的纹理对象（在本章中我们仅考虑 2D 图像）
  - 特殊的统一采样器变量，以便着色器访问纹理
  - 用于保存纹理坐标的缓冲区
  - 用于将纹理坐标传递给管线的顶点属性
  - 显卡上的纹理单元

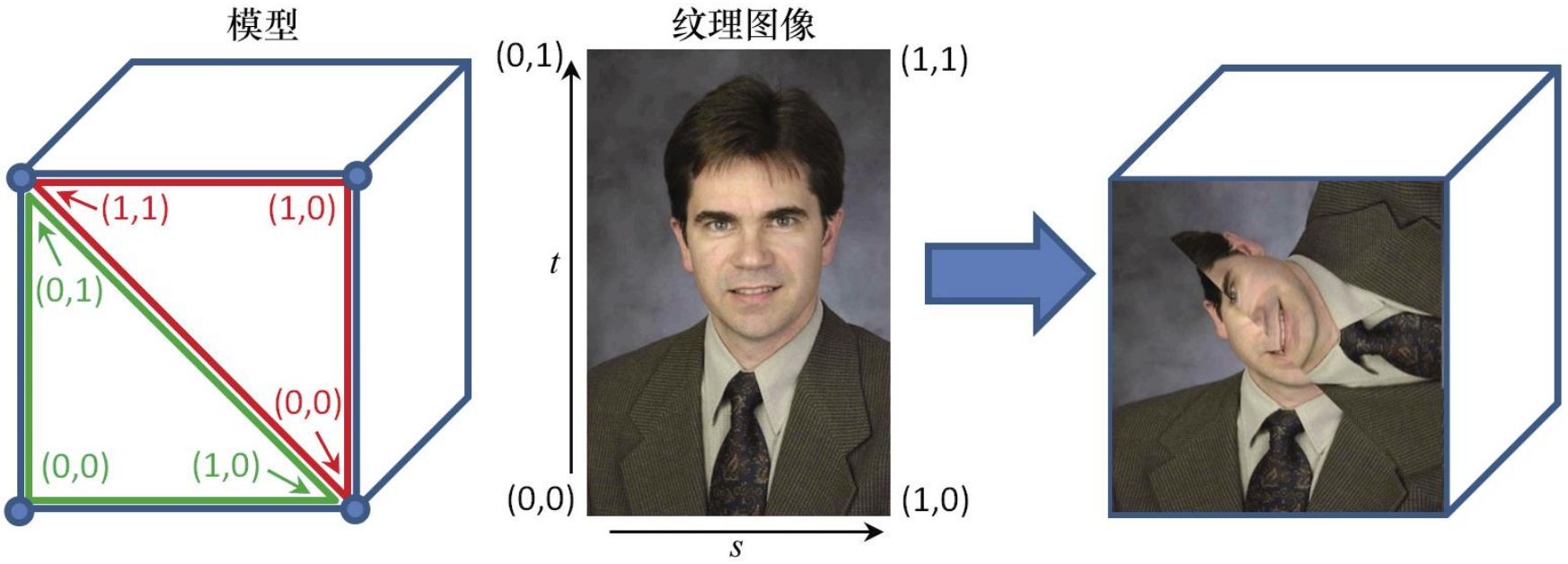
纹理图像

- 纹理图像可以是任何图像
- 通常存储在图像文件中，例如.jpg、.png、.gif 或.tiff 文件
- 从图像中提取颜色并将它们放入**OpenGL纹理对象**（用于保存纹理图像的内置OpenGL 结构）
- 使用SOIL2库将纹理图像加载到OpenGL纹理对象中：

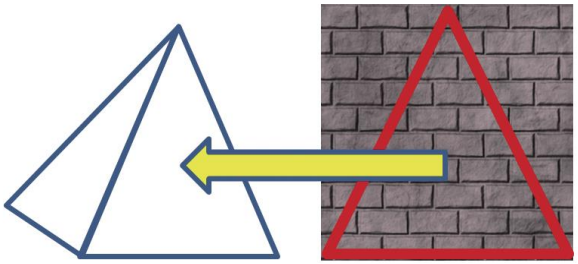
```
GLuint loadTexture(const char *texImagePath) {
    GLuint textureID;
    textureID = SOIL_load_OGL_texture(texImagePath, SOIL_LOAD_AUTO,
                    SOIL_CREATE_NEW_ID, SOIL_FLAG_INVERT_Y);
    if (textureID == 0)
       cout << "could not find texture file" << texImagePath << endl;
    return textureID;
}

. . .
GLuint myTexture = Utils::loadTexture("image.jpg");
```

纹理坐标

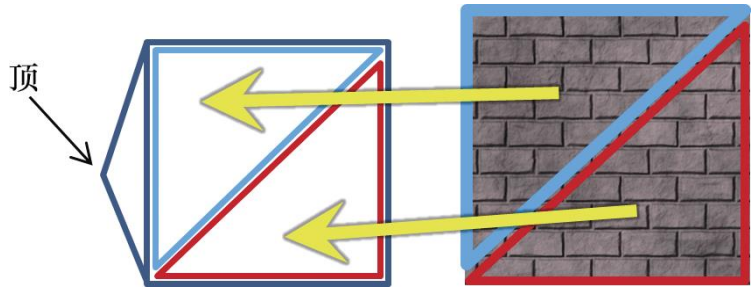
- 纹理图像中的像素被称为纹元（texel）
- 纹理坐标用于将 3D 模型上的点映射到纹理中的位置
- 纹理坐标（由 *s* 和 *t* 描述）将图像的部分（纹元）映射到模型正面的栅格化像素上
- 顶点之间的所有中间像素都已使用图像中间插值的纹元进行绘制



构建纹理坐标



使纹理图像的顶部中心对应四棱锥的顶



为四棱锥底面添加纹理

顶点	纹理坐标	
(-1.0,-1.0, 1.0)	(0, 0)	//前侧面
(1.0,-1.0, 1.0)	(1, 0)	
(0, 1.0, 0)	(.5, 1)	
(1.0,-1.0, 1.0)	(0, 0)	//右侧面
(1.0,-1.0,-1.0)	(1, 0)	
(0, 1.0, 0)	(.5, 1)	
(1.0,-1.0,-1.0)	(0, 0)	// 底面
(-1.0,-1.0,-1.0)	(1, 0)	
(0, 1.0, 0)	(.5, 1)	
...		

将纹理坐标载入缓冲区

- 将纹理坐标加载到 VBO 中：

```
float pyrTexCoords[36] =
{   0.0f, 0.0f, 1.0f, 0.0f, 0.5f, 1.0f,   0.0f, 0.0f, 1.0f, 0.0f, 0.5f, 1.0f,      // top and right faces
    0.0f, 0.0f, 1.0f, 0.0f, 0.5f, 1.0f,   0.0f, 0.0f, 1.0f, 0.0f, 0.5f, 1.0f,      // back and left faces
    0.0f, 0.0f, 1.0f, 1.0f, 0.0f, 1.0f,   1.0f, 1.0f, 0.0f, 0.0f, 1.0f, 0.0f       // base triangles
}

glBindBuffer(GL_ARRAY_BUFFER, vbo[1]);
glBufferData(GL_ARRAY_BUFFER,
                sizeof(pyrTexCoords),
                pyrTexCoords,
                GL_STATIC_DRAW);
```

在着色器中使用纹理

- 在着色器中声明一个采样器变量

```
layout (binding=0) uniform  sampler2D  samp;
```

  - layout (binding=0)指定此采样器与第 0 个纹理单元关联

- 在C++应用程序中，将纹理对象与纹理单元（在本例中为第 0 个单元）相关联

```
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, brickTexture);
```

- 发送顶点属性中的纹理坐标

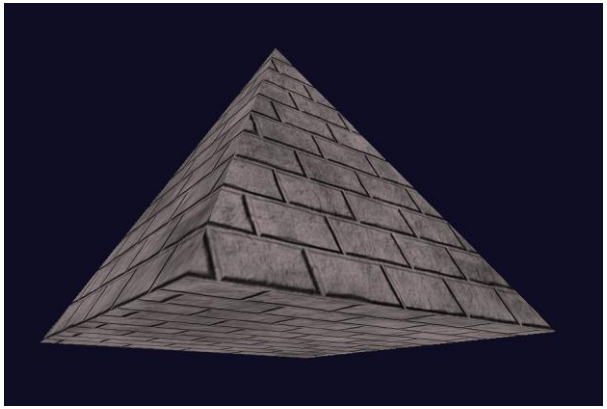
```
in vec2 tc;  // texture coordinates
```

- 最后，在片段着色器中使用纹理坐标从纹理中查找正确的纹素

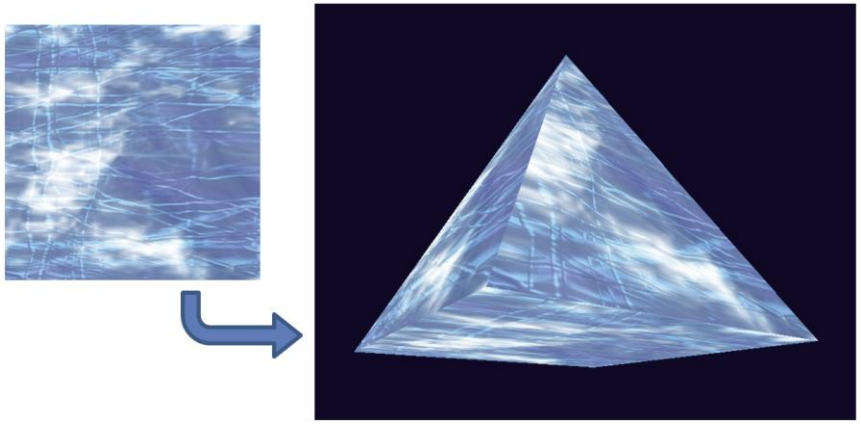
```
color = texture(samp, tc);
```

示例程序

- 程序 5.1 砖纹理的四棱锥



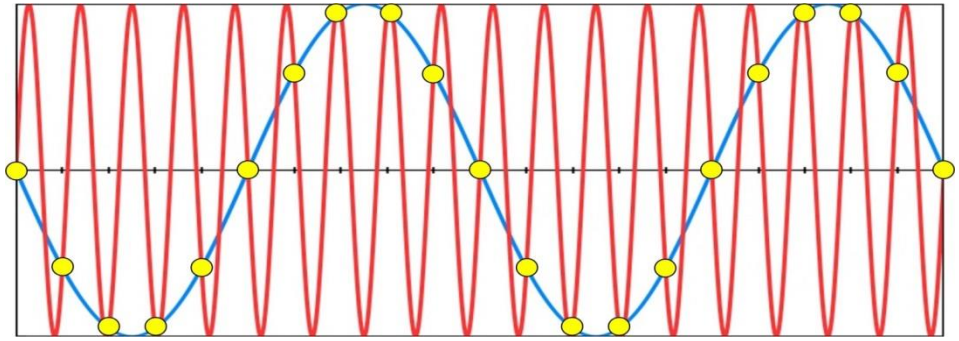
使用砖图像纹理贴图后的四棱锥



使用“冰”图像纹理贴图后的四棱锥

纹理瑕疵

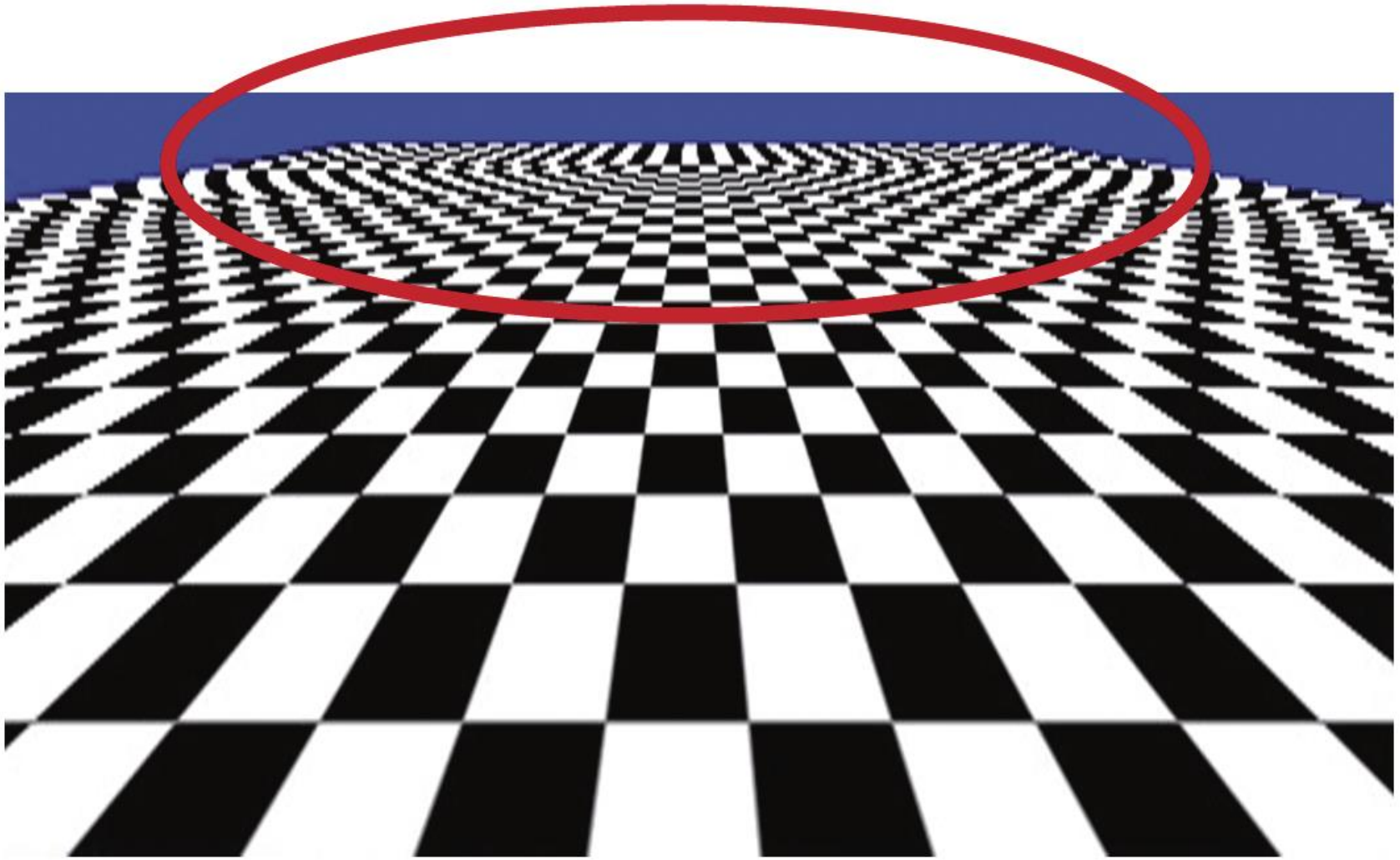
- 可能发生在以下情况：
  - 纹理图像分辨率过低（模糊、拉伸）
  - 纹理图像分辨率过高（！）
- 为什么这会成为一个问题？
  - 因为采样错误或走样



走样示例。原始波形为红色，再现的错误波形（由于采样不足）为蓝色。

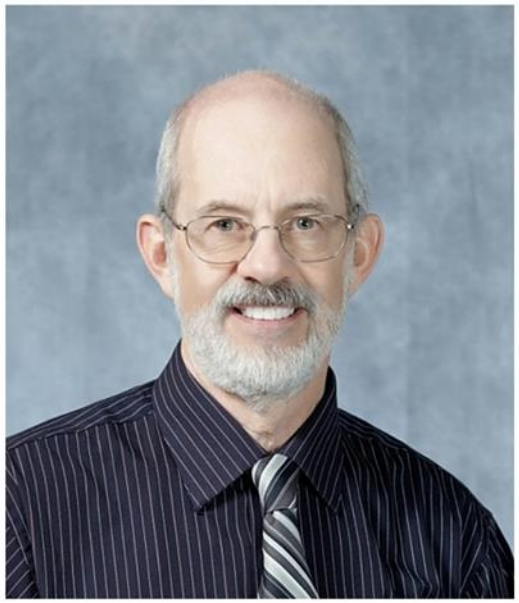
纹理瑕疵

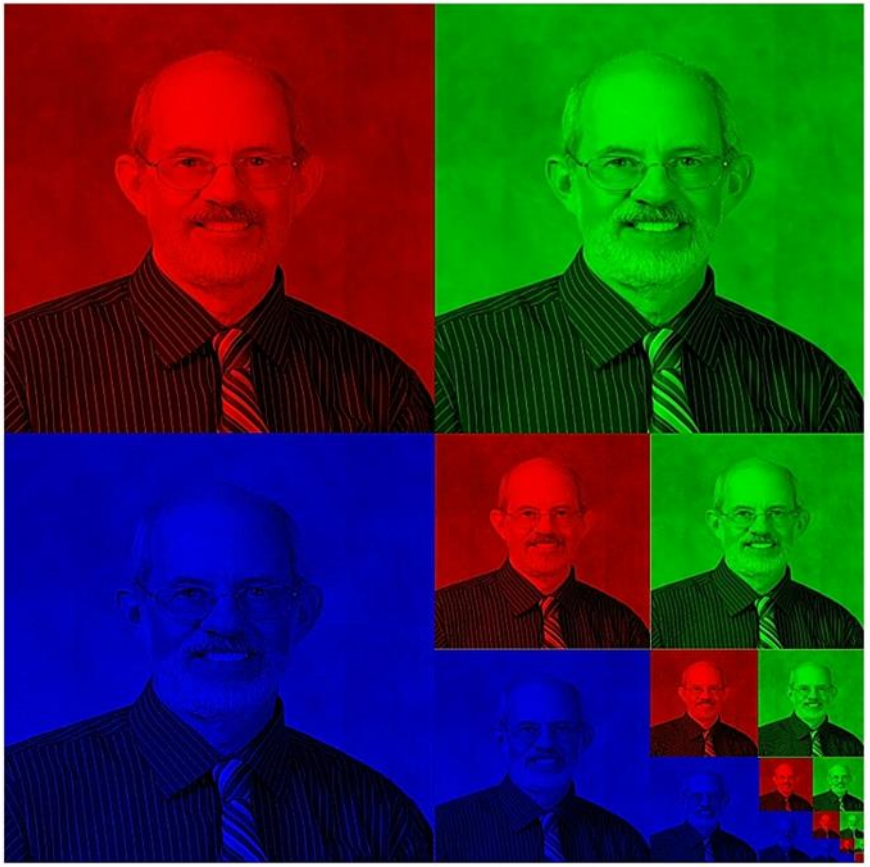
- 纹理贴图中的走样问题



反走样技术

- 多级渐远纹理贴图 -- mipmapping

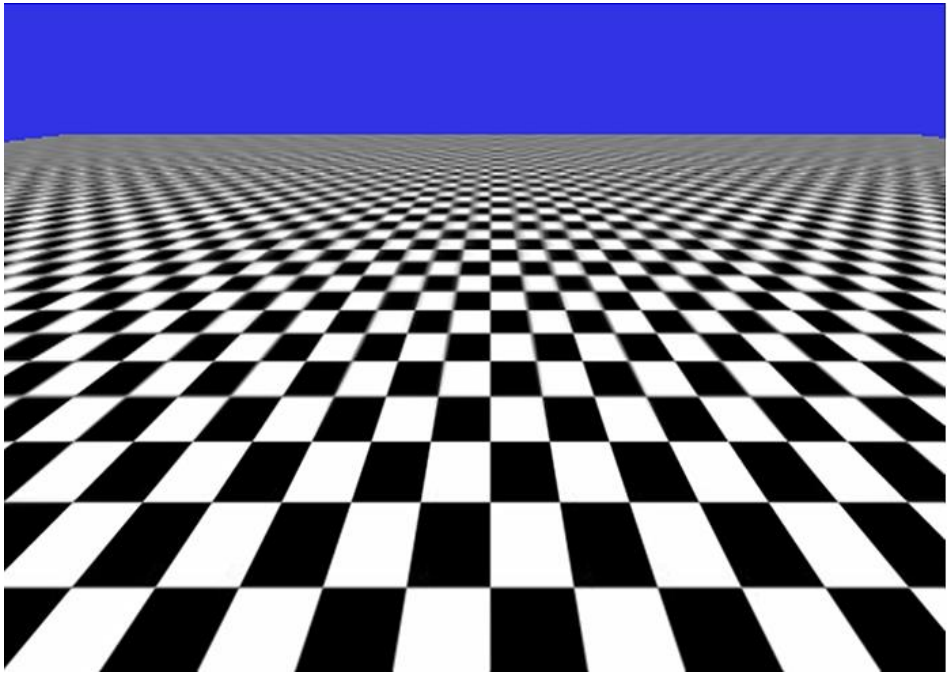




- 纹理图像文件的多个分辨率存储在一起。一个图片需要增加33%的存储空间。

OpenGL 多级渐远纹理支持

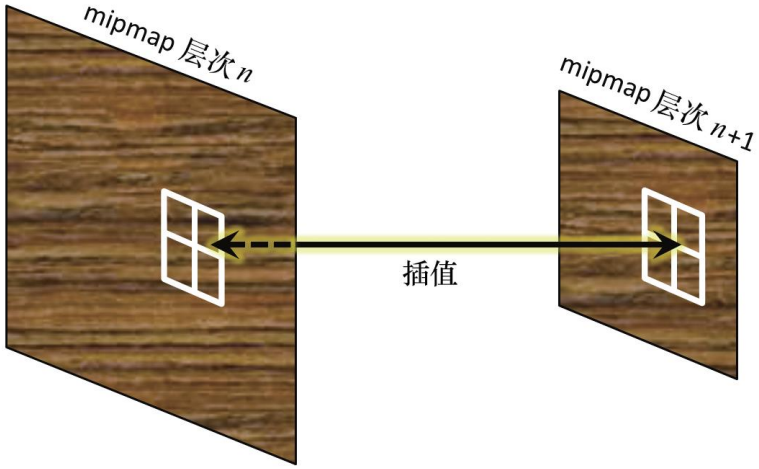
```
glBindTexture(GL_TEXTURE_2D, brickTexture);
glTexParameteri(    GL_TEXTURE_2D,
                    GL_TEXTURE_MIN_FILTER,
                    GL_LINEAR_MIPMAP_NEAREST );
glGenerateMipmap(GL_TEXTURE_2D);
```



OpenGL使用纹理映射中最接近绘制区域的分辨率

OpenGL 多级渐远纹理支持

- OpenGL 中通过GL_TEXTURE_MIN_FILTER 参数设置采样方法
  - GL_NEAREST_MIPMAP_NEAREST：选择具有与纹元区域最相似的分辨率的多级渐远 纹理。然后，它获得所需纹理坐标的最近纹元。
    GL_LINEAR_MIPMAP_NEAREST：选择具有与纹元区域最相似的分辨率的多级渐远纹 理。然后，它取最接近纹理坐标的 4 个纹元的插值。这被称为“线性过滤”。
  - GL_NEAREST_MIPMAP_LINEAR：选择具有与纹元区域最相似的分辨率的 2 个多级渐远纹理。然后，它从每个多级渐远纹理获取纹理坐标的最近纹元并对其进行插值。这被 称为“双线性过滤”。
  - GL_LINEAR_MIPMAP_LINEAR：选择具有与纹元区域最相似的分辨率的 2 个多级渐远纹理。然后，它取各自最接近纹理坐标的 4 个纹元，并计算插值。这被称为“三线性过滤”，如右图：



各向异性过滤（Anisotropic Filtering， AF）

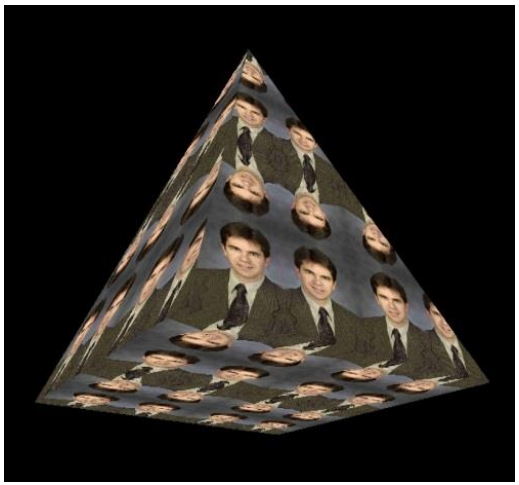
- 当对象倾斜时，Mipmapping可能会导致细节丢失，因为其基本体沿一个轴（即宽度与高度）看起来比另一个轴更小。
- AF通过以各种矩形分辨率（如256x128、64x128）对纹理进行采样来减少细节损失（同时仍能减少混叠和“闪光”）。
- AF在计算上更昂贵，而且并非所有显卡都支持它

OpenGL通过OpenGL扩展支持AF...

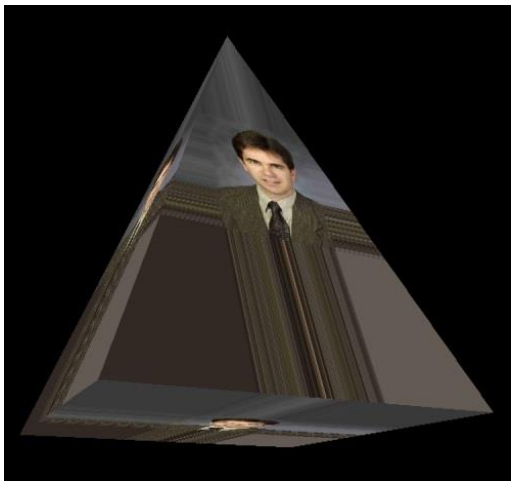
```
glTexParameteri(GL_TEXTURE_2D,
                GL_TEXTURE_MIN_FILTER,
                GL_LINEAR_MIPMAP_LINEAR);
glGenerateMipmap(GL_TEXTURE_2D);
// if also anisotropic filtering
if (glewIsSupported("GL_EXT_texture_filter_anisotropic")) {
    GLfloat anisoSetting = 0.0f;
    glGetFloatv(GL_MAX_TEXTURE_MAX_ANISOTROPY_EXT, &anisoSetting);
    glTexParameterf(GL_TEXTURE_2D,
                    GL_TEXTURE_MAX_ANISOTROPY_EXT, anisoSetting);
}
```

环绕和平铺

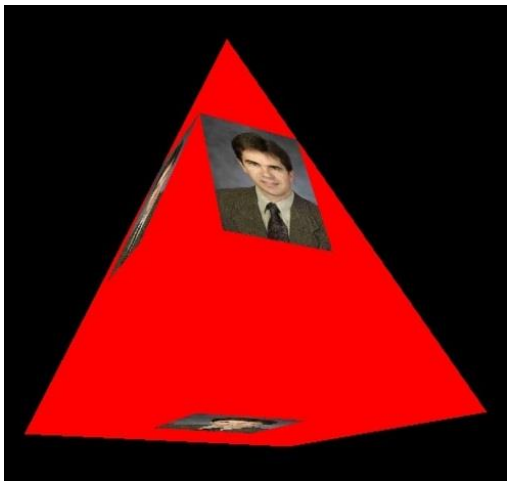
- 当纹理坐标超出（0,1）范围时，OpenGL有几个选项：



GL_REPEAT



GL_CLAMP_TO_EDGE



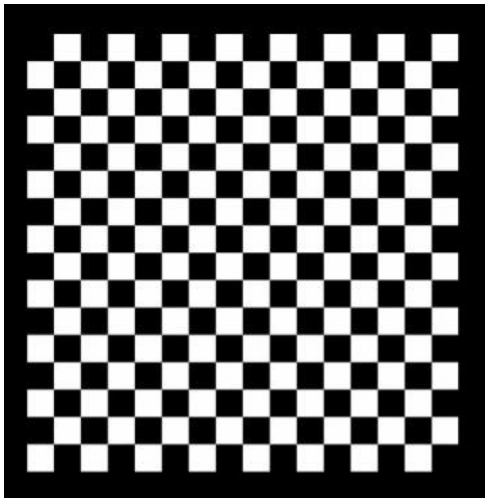
GL_CLAMP_TO_BORDER

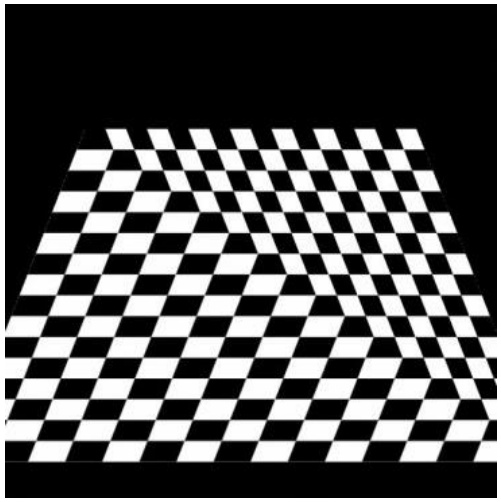
for example:

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_BORDER);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_BORDER);
float redColor[4] = { 1.0f, 0.0f, 0.0f, 1.0f };
glTexParameterfv(GL_TEXTURE_2D, GL_TEXTURE_BORDER_COLOR, redColor);
```

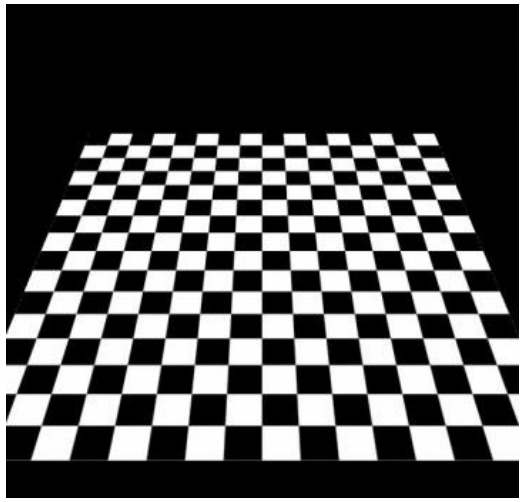
透视变形

- OpenGL在3D空间中对“倾斜”对象进行纹理处理时，会自动校正透视失真。





透视失真



OpenGL 透视矫正

- 可以在着色器中禁用此校正。例如：

```
noperspective out vec2 texCoord;        (in the vertex shader)

noperspective in vec2 texCoord;         (in the fragment shader)
```