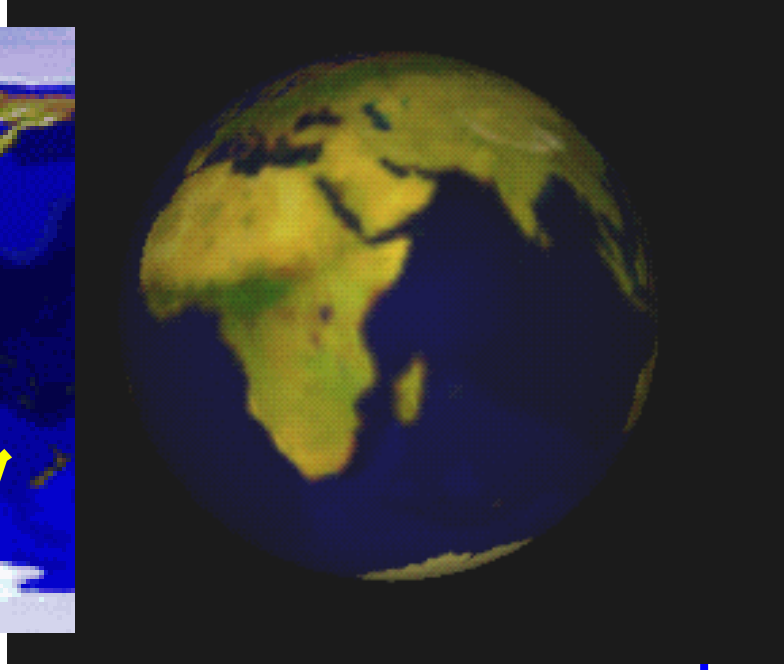
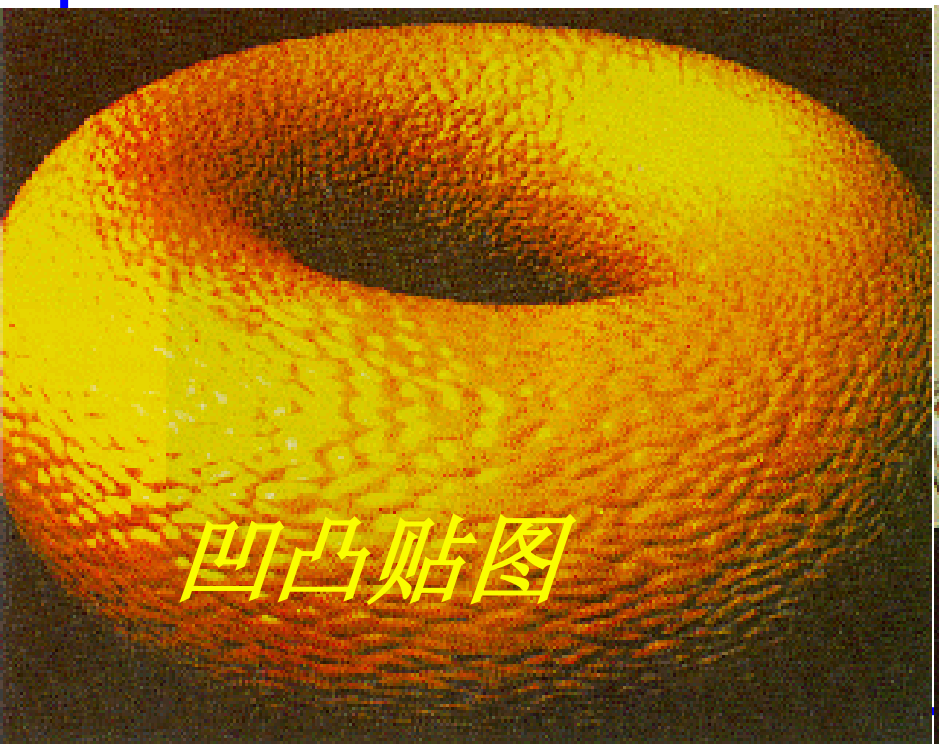
纹理映射


图像贴图




凹凸贴图


环境贴图

纹理映射

- 到目前为止，绘制的物体要么是纯色，要么是顶点颜色的光滑过渡
  - 类似于绘画
- 相反，纹理映射可以给物体的表面属性带来变化
  - 类似于表面修饰，例如贴墙纸或桶上贴标签

为什么需要纹理映射？

- 纹理表达了表面属性的变化，这些属性包括：颜色、法向量、镜面高光、透明度、表面位移等
- 如果能够捕捉这些复杂的细节，计算机生成的画面将更加真实
- 使用几何造型来表达这些细节非常困难，因为将会极大地加重计算需求
- 纹理映射是一种以较低代价“伪造”表面细节的有效手段

什么是纹理映射？

- 纹理映射是将一张纹理变换到三维物体表面的过程
- 类似于在三维曲面建立一个映射函数
  - 该函数的参数域可能是1D, 2D 或 3D
  - 函数值可以用数组或者是代数函数表示

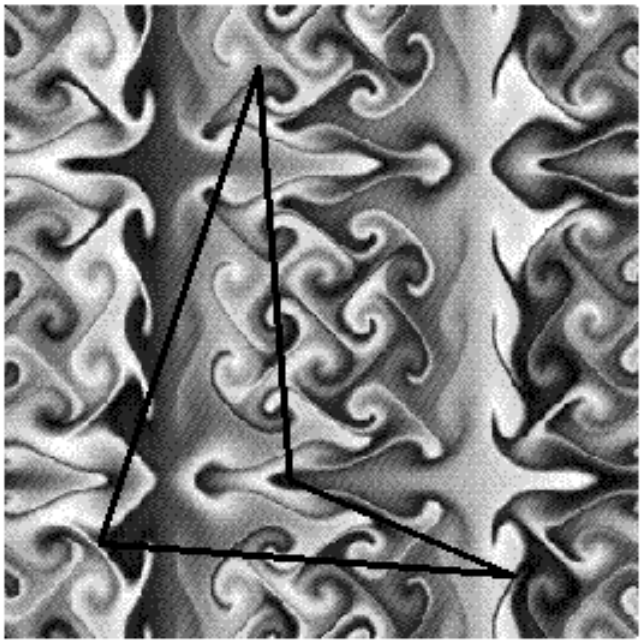
物体的类型?

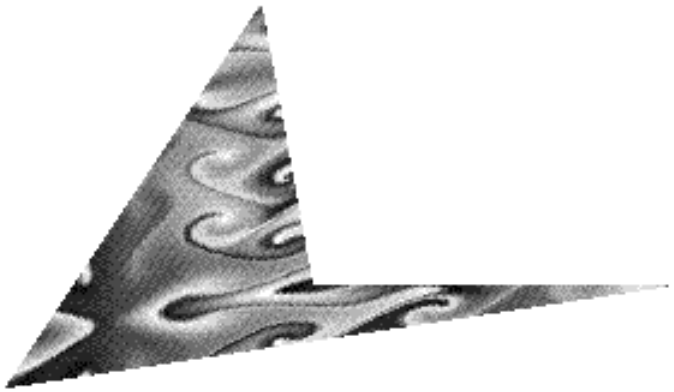
- 通常来说，任意3D物体的纹理映射是困难的
  - 例如：扭曲 (试图将一张平面纹理映射到一个球体上)
- 对多边形和参数曲面相对简单
- 下面仅讨论多边形的纹理映射

什么是一张典型的2D 纹理?

- 用长方形数组表示的一个2D函数
  - 颜色数据
  - 光强数据
  - 颜色和透明数据
- 纹理中的最小数据单元称为texel; 在屏幕上，一个texel可能被映射到：
  - 单个pixel
  - 单个pixel的部分 (小的多边形)
  - 多个pixel (如果纹理非常小或者多边形被非常靠近地观察而变得很大)

例子





指定纹理坐标

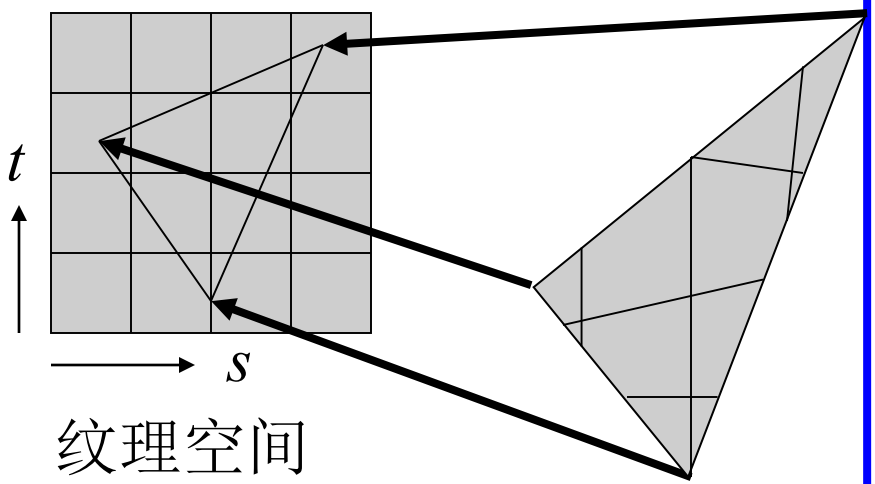
- 你必须为每个顶点提供纹理坐标
- 纹理图像本身覆盖了二维空间中的0/1坐标空间，通常使用s和t表示，以区别于三维空间中的 x, y , z坐标
- 一个顶点的纹理坐标决定了哪些 texel将被映射到该顶点

指定纹理坐标

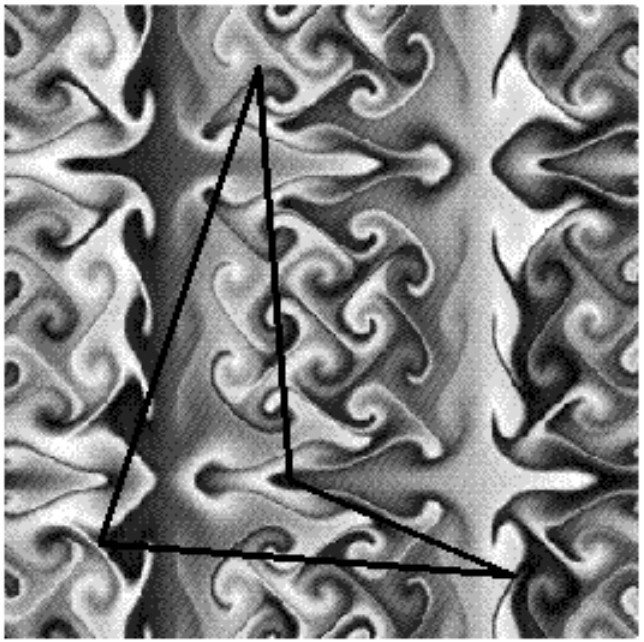
- 每个顶点的纹理坐标决定了纹理的哪个部分将被应用于该多边形
- 这个纹理的子集将被拉伸或挤压，以适合多边形的尺寸

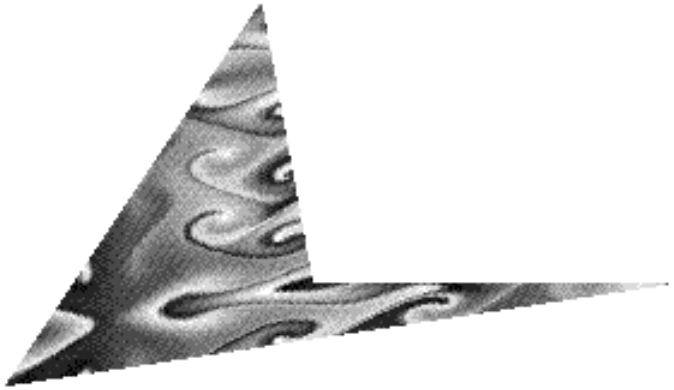
纹理插值

- 首先指定世界坐标中的顶点将被映射到纹理空间的哪些位置
- 其他的点将在世界空间中线性插值得到
  - 世界空间中的直线被映射为纹理空间中的直线



纹理示例





多边形的纹理映射

- 建立对应关系
- 找到组合2D-2D映射
- 在多边形扫描转换时使用这个映射来更新期望的属性（如颜色）

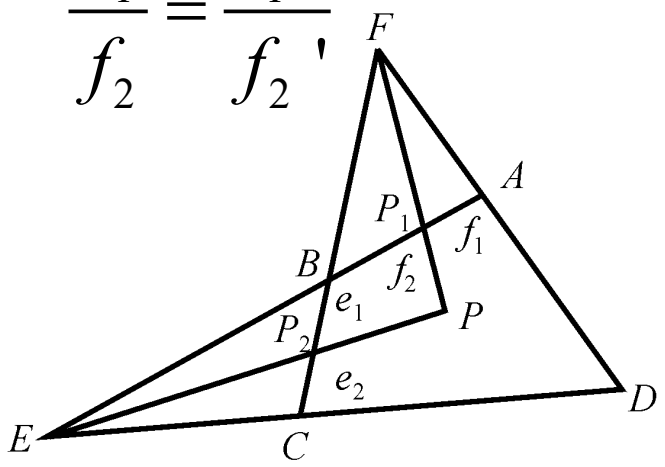
建立对应关系

- 通常我们对每个顶点指定纹理坐标
- 这些纹理坐标建立起图像和多边形之间的对应关系

找到组合2D-2D映射

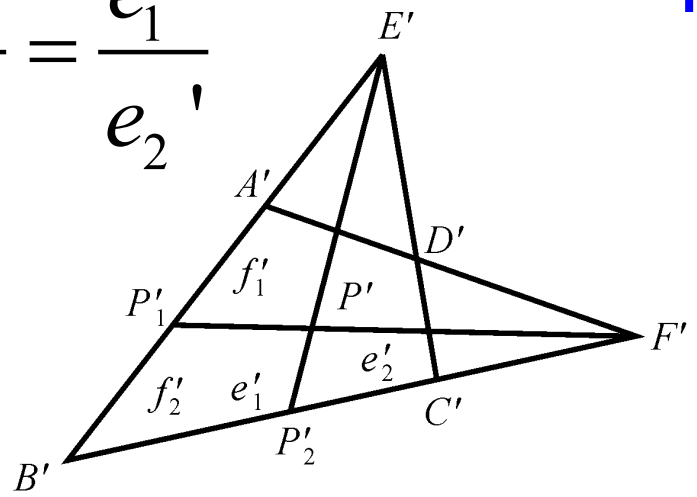
- 由于纹理在屏幕上最终显示为 2D, 因此可以将两个映射组合在一起 (从图像到三维空间、从三维到屏幕空间) 成为一个组合2D-2D映射
- 这完全避免了在3D进行纹理计算
- 同时也简化了图形流水线的硬件实现

$$\frac{f_1}{f_2} = \frac{f_1'}{f_2'}$$



(a) 目的四边形 S

$$\frac{e_1}{e_2} = \frac{e_1'}{e_2'}$$



(b) 源四边形 S'

原始的纹理映射代码

```
// 对每个顶点，我们有x,y,z 和 u,v
for (x=xleft; x < xright; x++) {
    if (z < zbuffer[x][y] ){
       z[x][y] = z;
            raster[x][y] = texture[u][v]; // 替换颜色
    }
    z=z+dz;
    u=u+dv;
    v=v+dv;
}
```

- **注意代码中是替换颜色，你也可以进行颜色调和**
- **除了颜色，你当然可以选择修改表面的其他属性**

纹理中的调和属性

- 表面颜色(漫反射系数)
  - 这是纹理映射中最常使用的参数。类似于在表面贴上一副图像（如在酒瓶上贴标签）
- *镜面和漫反射* **(环境贴图)**
  - 用于捕捉周围环境在特定表面的反射，常用于展现光亮的金属表面
- *法向量扰动* **(凹凸贴图)**
  - 用于生成粗糙的表面，如橘子皮

纹理中的调和属性

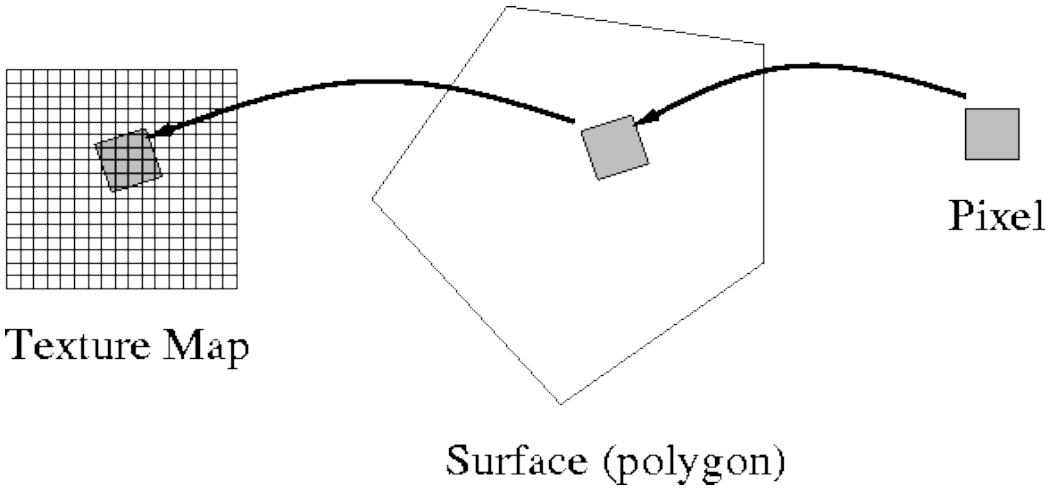
- *镜面反射*
  - 用于生成具有可变光亮区域的表面
- *透明度*
  - 用于生成具有变化透明度的表面，如云彩。更多地用于生成复杂物体，而不仅仅是在表面进行纹理贴图
- 除了*环境贴图*，物体上的纹理与它在世界中的位置无关。因此*环境贴图*将和*纹理映射*区分讨论

问题

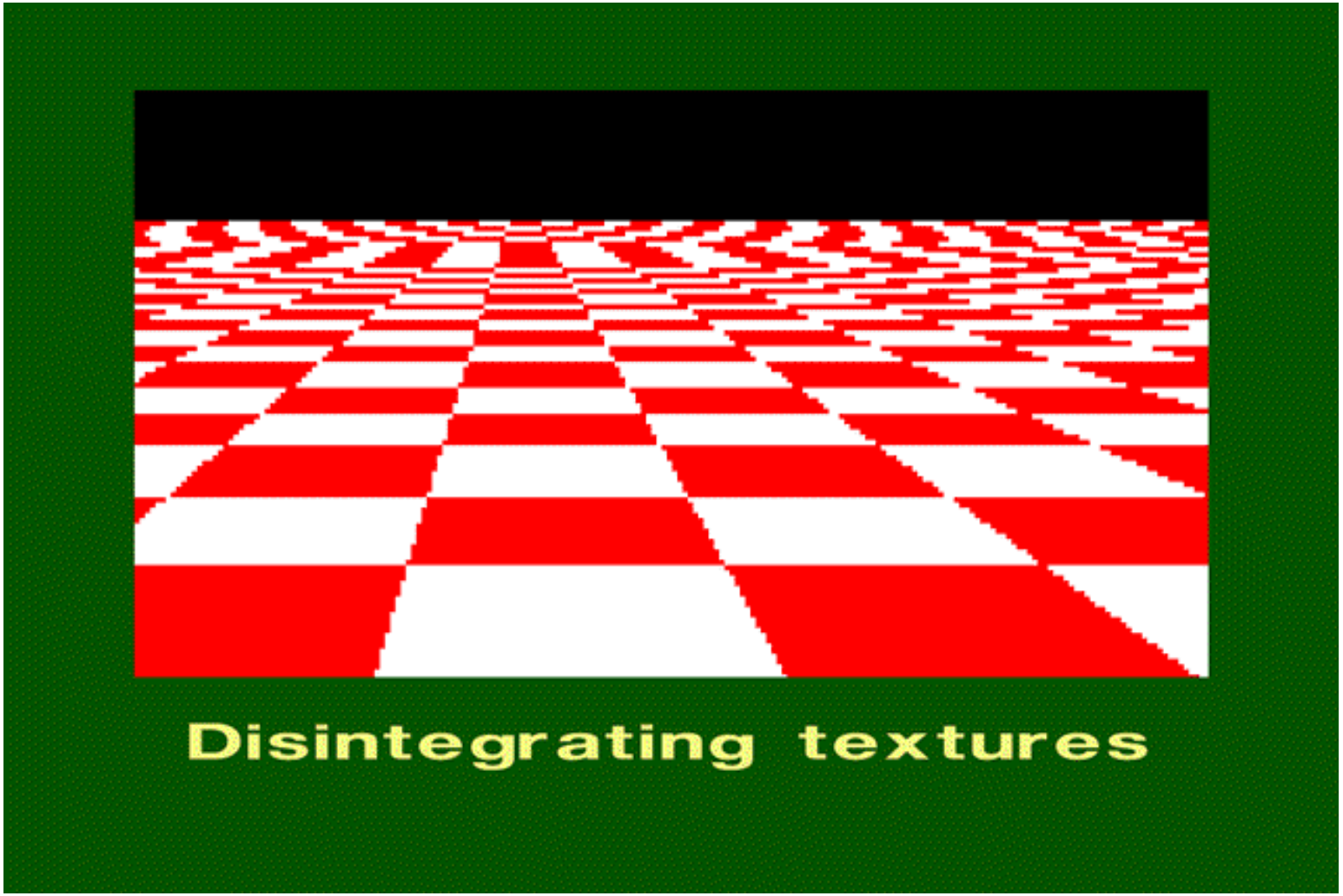
- 走样瑕疵
  - 由于点采样
- 透视扭曲
  - 由于没有考虑透视变换

纹理映射中的颜色计算

- 将纹理与多边形关联
- 将像素映射到多边形，然后映射到纹理贴图
- 使用覆盖纹理的加权平均来计算颜色



走样示例



解决走样

- 什么造成了走样瑕疵？
  - 高频信号
- 典型解决方法
  - 提高采样率
  - 用低频滤波预先对纹理过滤

优化

- 大多数情况下, 纹理是预先已知的，可以在预处理阶段创建过滤纹理的不同层次
- 在运行时刻，取出需要的层次纹理进行应用
- 这种技术称为mipmapping

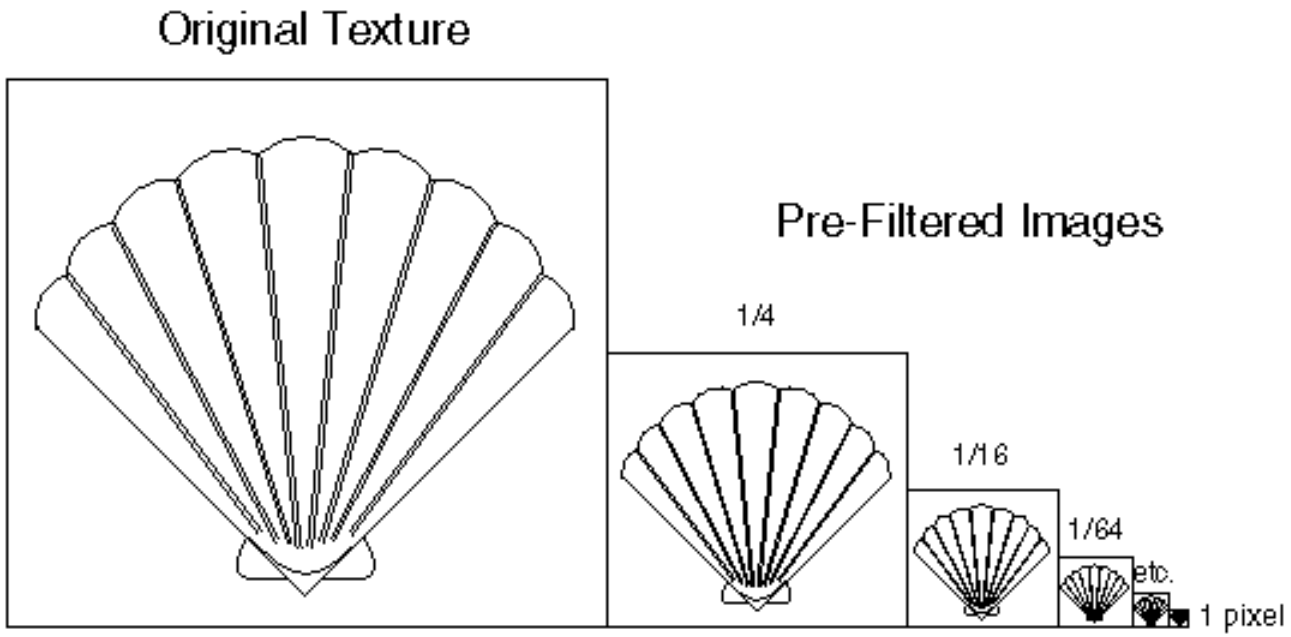
Mipmapping

- 与其他的物体一样，贴过纹理的物体也会从不同的距离进行观察
- 有时候，这将造成问题：
  - 一张低分辨率的纹理 (比如 32x32) 应用于一个大的多边形 (比如占用了512x512的屏幕区域)将显得非常斑块化
  - 反之，如果将一张高分辨率纹理应用于一个小的多边形，如何决定哪些 texel显示，哪些忽略？

Mipmapping

- 一个解决方法是为同一张纹理提供多个细节层次，然后根据多边形在屏幕上的显示尺寸，挑选出最佳匹配的细节层次纹理，并进行使用
- 这个技术被称为Mipmapping, 在这个层次里面的每张纹理被称为一张 mipmap
- OpenGL支持自动计算 mipmap，也可以接收来自于不同文件的mipmap

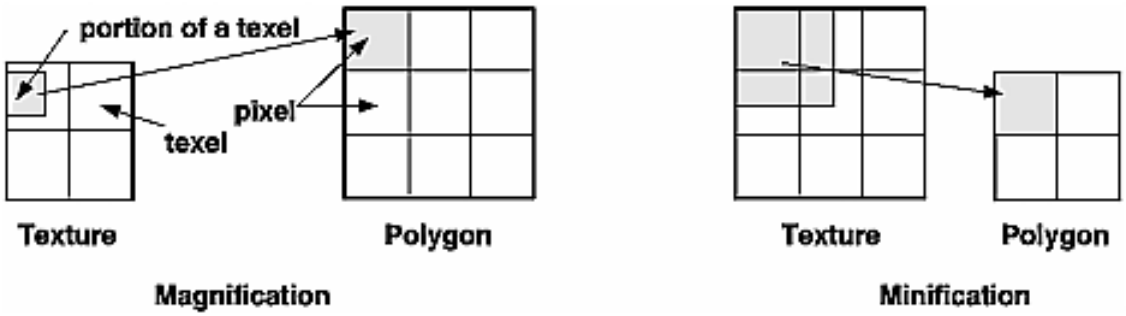
Mipmapping示例



纹理过滤

- 当纹理被映射到多边形时，单个texel很少会精确匹配单个像素
- 如果一个像素只匹配一个texel的部分，则该texel必须进行放大
- 如果一个像素匹配多个texel，则texel需要缩小

过滤示例


portion of a texel
pixel
texel
Texture
Polygon
Magnification
Texture
Polygon
Minification

纹理矩阵

- 将纹理装入到显卡通常比较昂贵
- 但是完成后，可以使用***纹理矩阵***对该纹理进行“变换”
  - 例如，改变平移量可以选择某张纹理的不同部分
- 如果纹理矩阵每帧都进行修改，则纹理可以表现为在物体上移动
- 这对于火焰、漩涡等的绘制特别有用

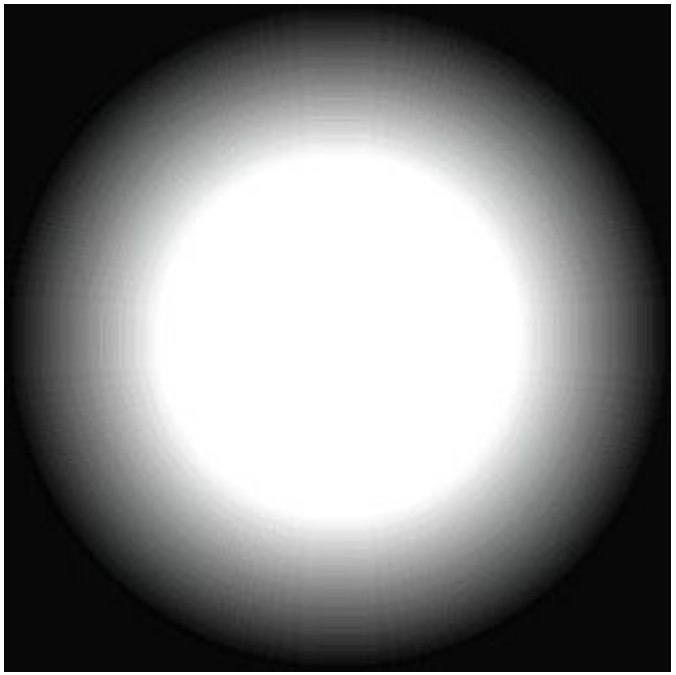
多纹理

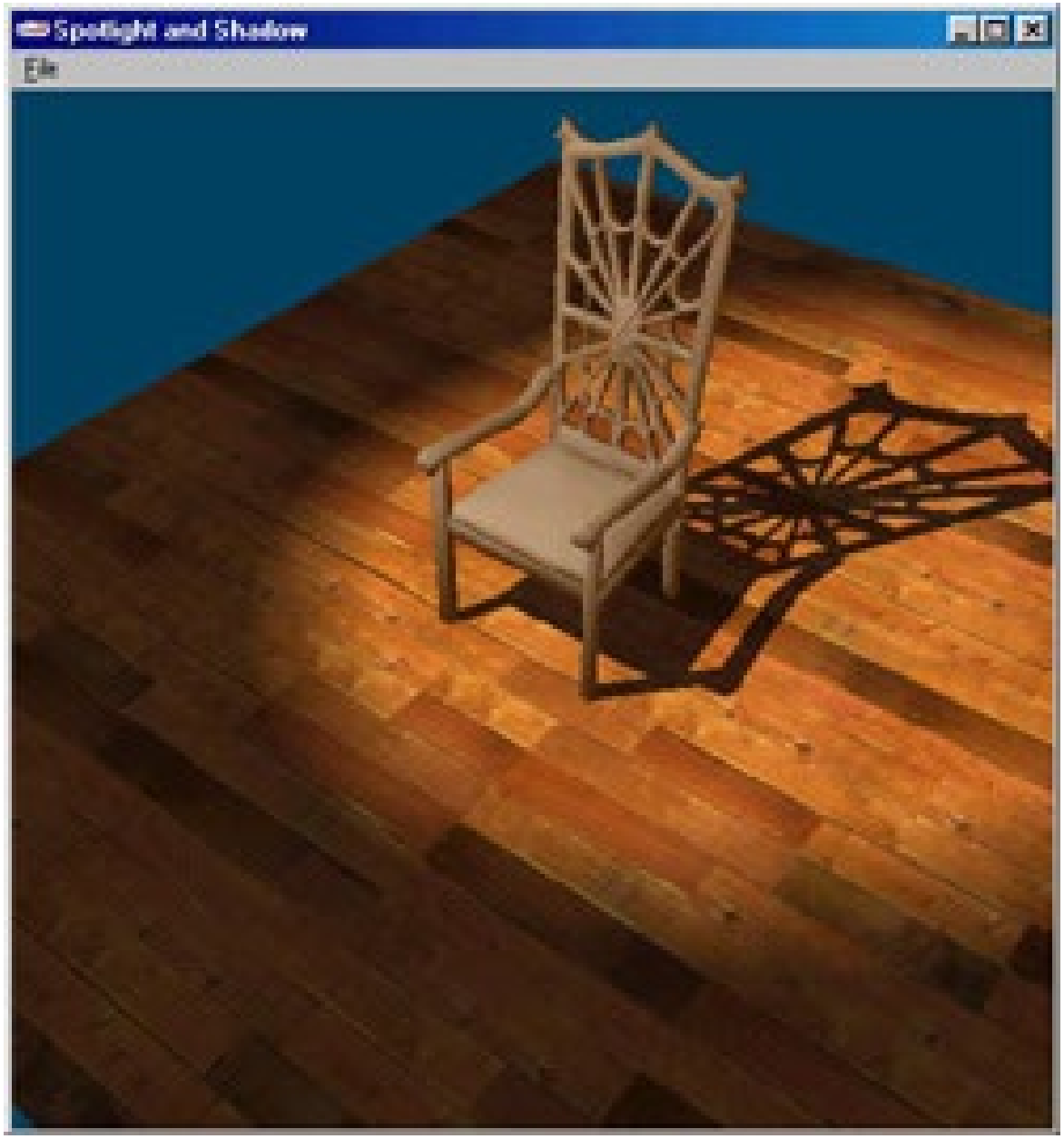
- 有时候可能需要在一个多边形上应用多张纹理
- 例如：光照贴图、纹理过渡
- 注意：大部分，但不是全部的OpenGL实现，支持多纹理！

光照贴图

- 除了使用纹理中的颜色进行贴墙纸外，我们也可以使用纹理颜色与已有颜色进行混合
- 一张“light map”是一副黑白纹理; 白色texel使得下面的像素显得更光亮，黑色则反之

光照贴图示例




Spotlight and Shadow
File

动画纹理

- 除了使用固定图片，有时也可以使用动画帧（例如Flash动画）作为纹理
- 这种情况下，多边形表面的纹理将随着时间而变化

OpenGL 纹理映射

- 创建纹理对象
  - 1D, 2D, 3D
  - 指明纹理中保存的内容：颜色、深度等
- 指明纹理如何被使用：
  - 替换、调和或混合
- 打开纹理映射
- 在场景中同时指明几何坐标和纹理坐标

指明纹理

- glTexImage2D(target, level, internalformat, width, height, border, format, type, *texels);
- E.g.:
  - target: GL_TEXTURE_2D
  - Level : 0 (for no mip maps)
  - Internalformat: GL_RGB
  - Width,height: 2^m+2*b, 256 by 256 with b=0
  - Border b: 0
  - Format: GL_RGB
  - Type: GL_INT
  - texels: 真正的纹理图像

指明mipmaps

- 需要指明使用mipmap层级图像
- 也可以指明缩放因子 (在纹理图像和多边形尺寸之间)
- 提供每层的mipmap图像
- 使用 glTexParameteri() 和 glTexImage2D()
  - glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_BASE_LEVEL, 2);
  - glTexImage2D(..,2,…);

自动生成mipmap

- 根据给定的最高分辨率图像，OpenGL可以自动生成预先过滤的低层次mipmap
- gluBuild2DMipmaps()
- gluBuild2DMipmapsLevels()
  - 只创建一个子集

OpenGL过滤

- 可以指明使用哪种过滤
- OpenGL的纹理过滤原始但是快速
- 使用函数glParameteri()指明过滤，参数为 GL_TEXTURE_MAG_FILTER 和 GL_TEXTURE_MIN_FILTER
- 过滤示例：GL_NEAREST, GL_LINEAR, GL_NEAREST_MIPMAP_LINEAR (minification)
- 例子：glParameteri (GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);

其他函数

- glGenTextures();
  - 生成唯一的名字
- glBindTexture();
  - 如果未创建则创建纹理对象
  - 如果已创建则激活该纹理对象
  - 参数0则停止使用所有纹理对象
- glEnable(GL_TEXTURE_2D);
  - 激活纹理映射
- glTexCoord2i(texcoordinate);

调和表面属性

- 除了替换颜色，你也可以调和颜色或其他多边形的属性
- glTexEnv(GL_TEXTURE_ENV, pname, param);
- Pname 和 param 指明纹理如何影响表面绘制
- 例子：glTexEnv (GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);

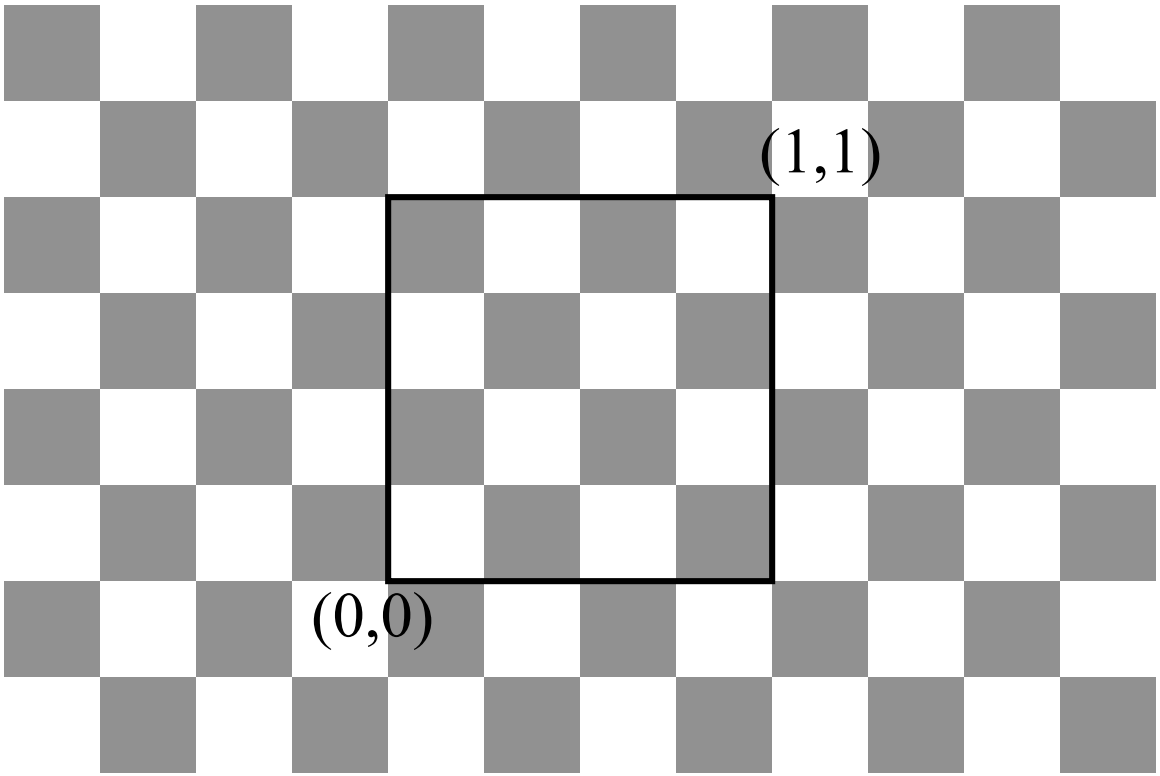
纹理的技巧

- 如果纹理不够大，不能够覆盖一个多边形，可以使用参数 GL_REPEAT 进行平铺：
  - glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
- 也可以截断一张纹理 (例如拉伸最后的像素覆盖所有)

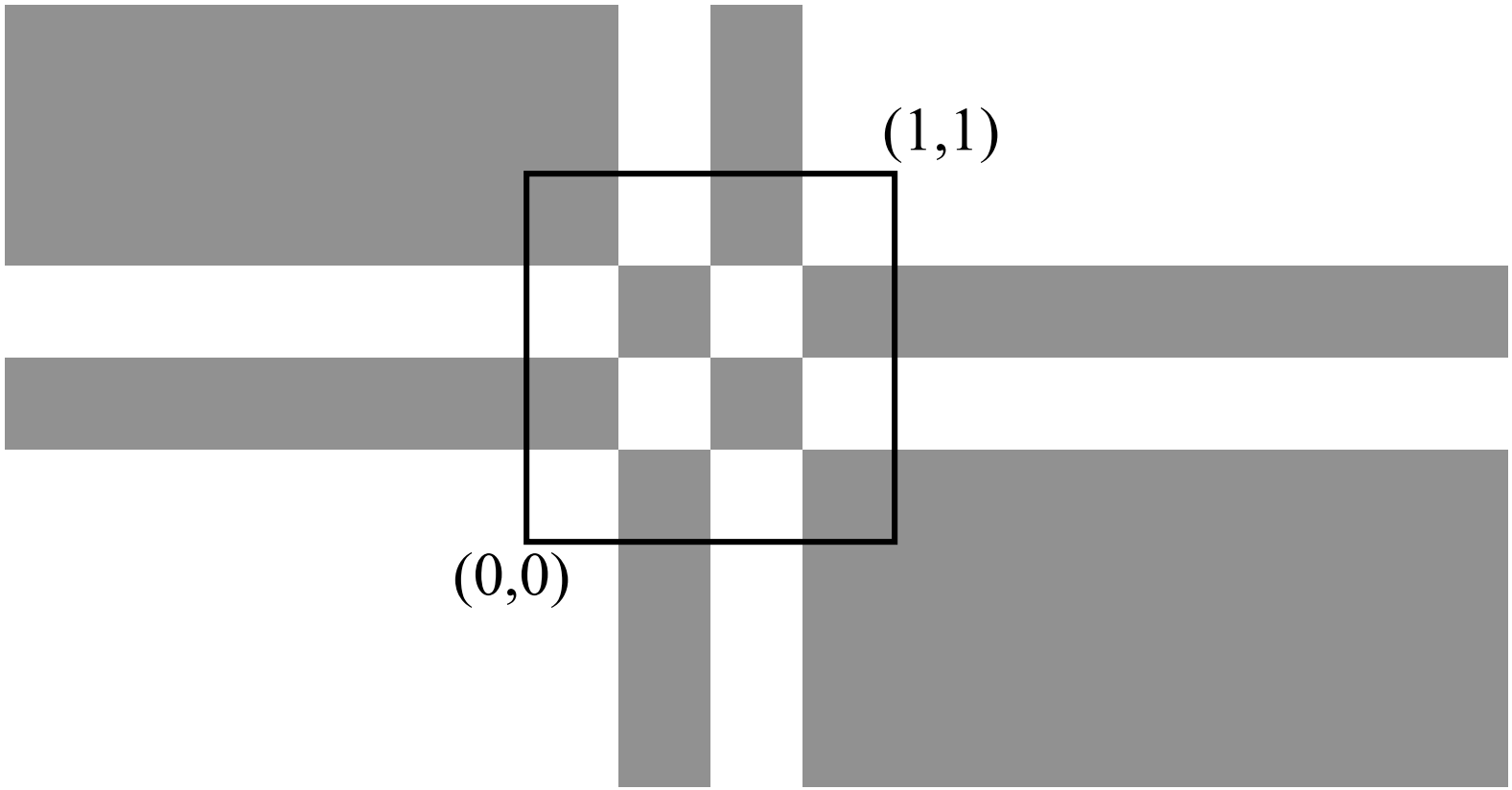
边界

- 可以控制当一个点映射到的纹理坐标落在了纹理图像外面时如何进行：
  - 所有的纹理都假定在纹理空间 (0,0) 到 (1,1) 之间
- 重复: 假定纹理超出部分进行平铺
  - glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT)
- 截断: 截断到边缘: 纹理坐标截断为合法值，然后使用
  - glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP)
- 也可以指定一个边界颜色:
  - glTexParameterfv(GL_TEXTURE_2D, GL_TEXTURE_BORDER_COLOR, R,G,B,A)

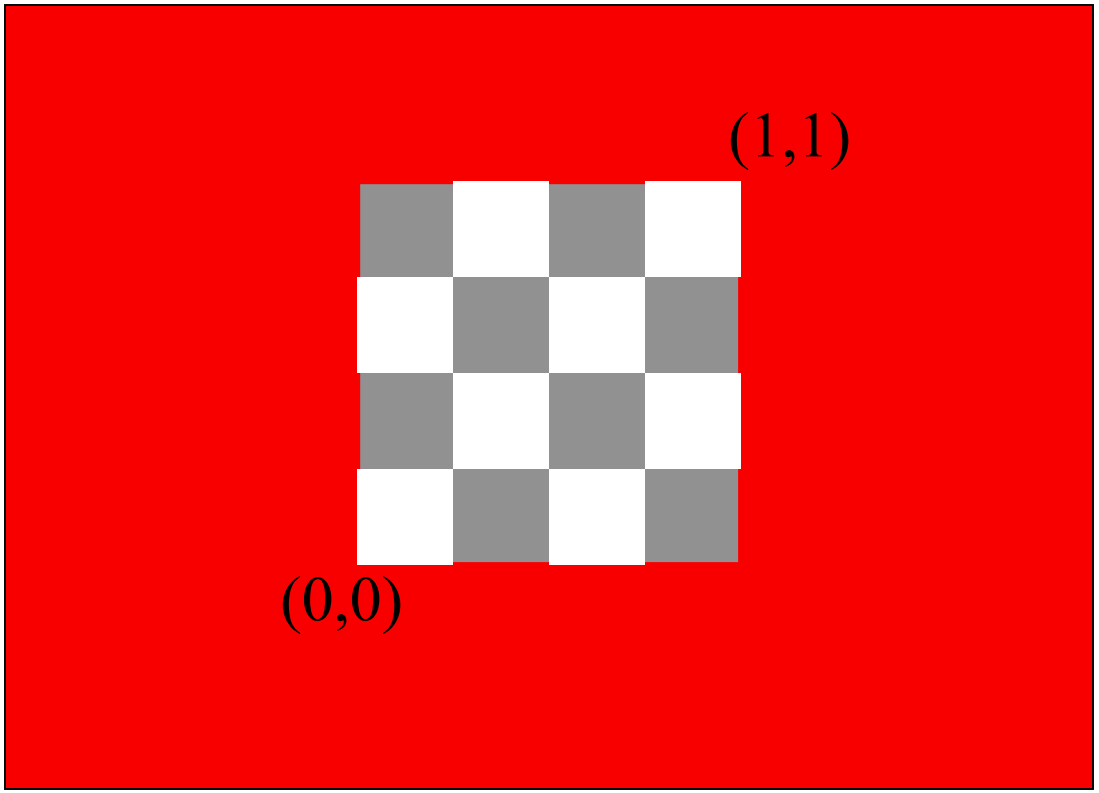
重复边界



截断边界

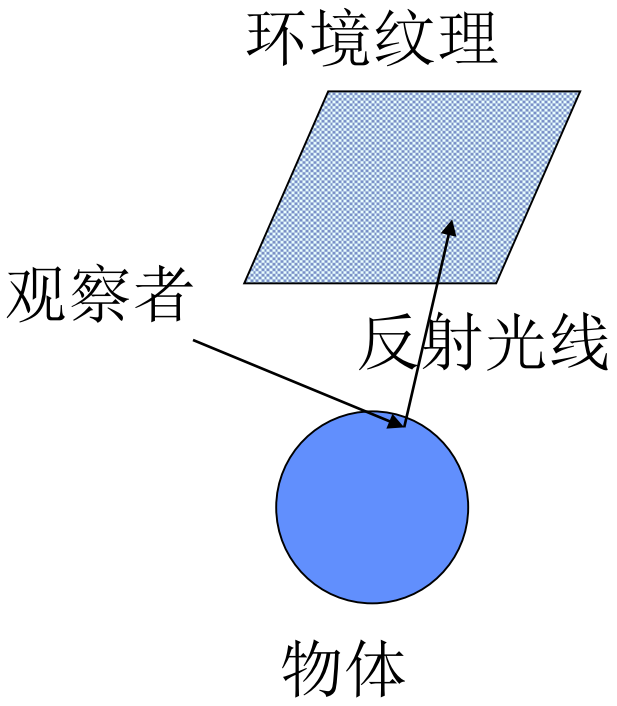


边界颜色

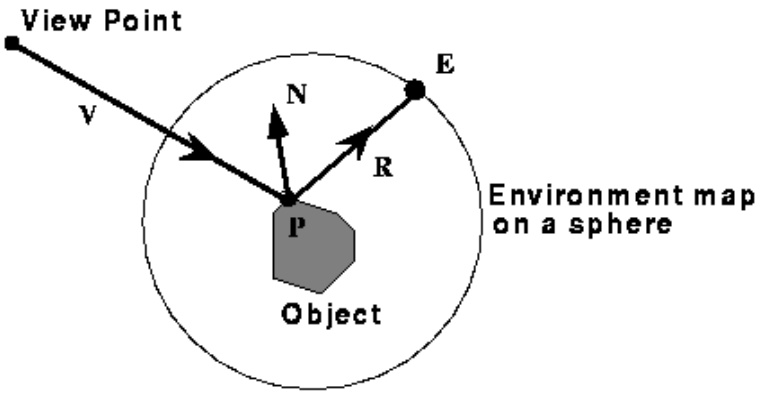


环境贴图

- 环境贴图可以生成光亮表面的反射现象
- 纹理按照反射光线的方向进行变换，从环境纹理映射到物体表面
- 反射光线: $\boldsymbol{R}=2(\boldsymbol{N}\cdot\boldsymbol{V})\boldsymbol{N}-\boldsymbol{V}$



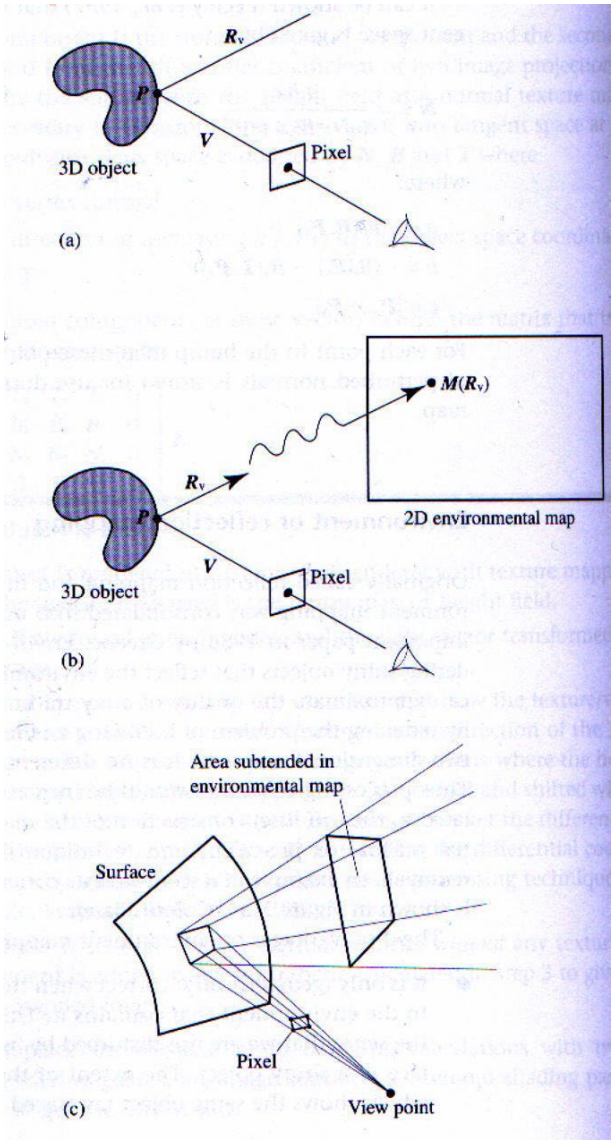
环境纹理



- 使用了反射光线的***方向***作为索引访问环境纹理
- 可以模拟反射，但是并不完全精确。该方法假定所有的反射光线都自同一点出发，并且场景中的所有物体都与该点距离相等

球形贴图

- 该贴图落在球面，坐标映射变得简单
- 我们使用经度和维度做为(u,v)

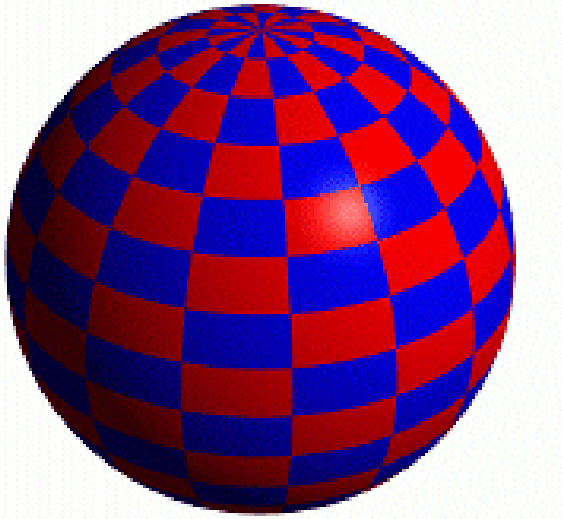


立方体贴图

- 贴图落在包围物体的立方体表面
  - 典型的，立方体各面与坐标轴对齐
- 为了生成这些贴图：
  - 对立方体的每个面，以该面为成像平面，从对象的中心进行绘制
    - 绘制可以任意复杂 (离线绘制)
  - 或者，在物体的位置放置相机实拍6张相片
- 假定立方体表面与坐标轴对齐，并且纹理坐标落在 [0,1]x[0,1]。主要需要决定的因素：
  - 使用哪个面；
  - 使用哪个纹理坐标；
  - 使用通过中心的射线；只有立方体的角点和边缘需要特别处理

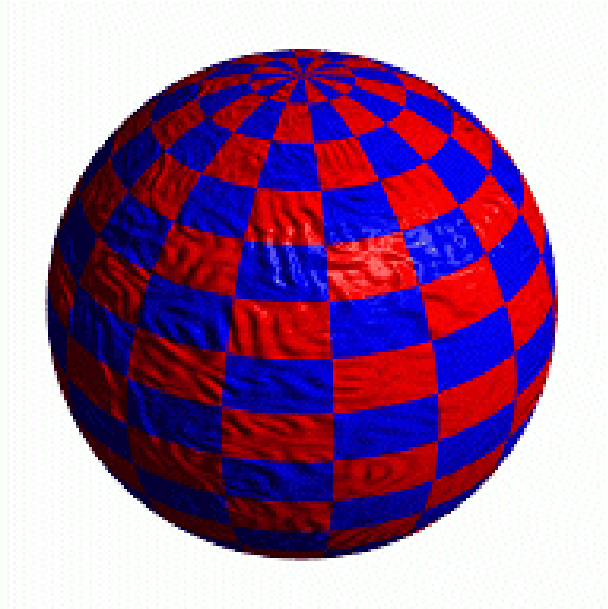
凹凸贴图

凹凸贴图假定绘制时使用逐像素的照明模型 (如Phong色调或光线跟踪).



带有普通纹理的球体

漩涡状凹凸贴图

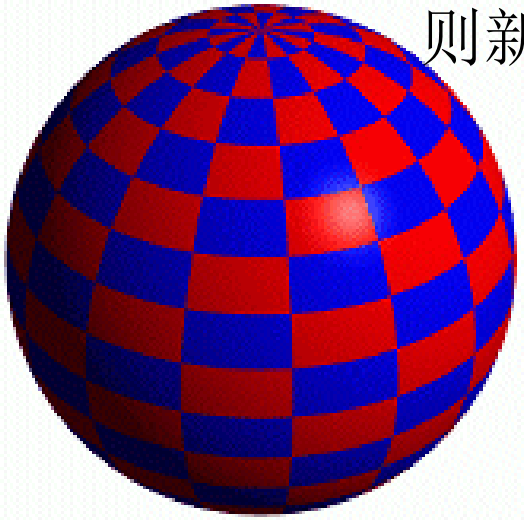


加入了凹凸贴图和普通纹理的球体

凹凸贴图

凹凸贴图假定绘制时使用逐像素的照明模型(如Phong色调或光线跟踪).

假定$S(u, v)$是原表面，N(u,v)是法向量, $P(u, v)$是小的扰动,



则新的表面可以表达为：

$$S(u, v)=Q(u, v)+P(u, v)\mathrm{N}(\mathrm{u,v})$$

$$S_u = Q_u + P_u \cdot \frac{N}{|N|} + P \cdot (\frac{N}{|N|})_u$$

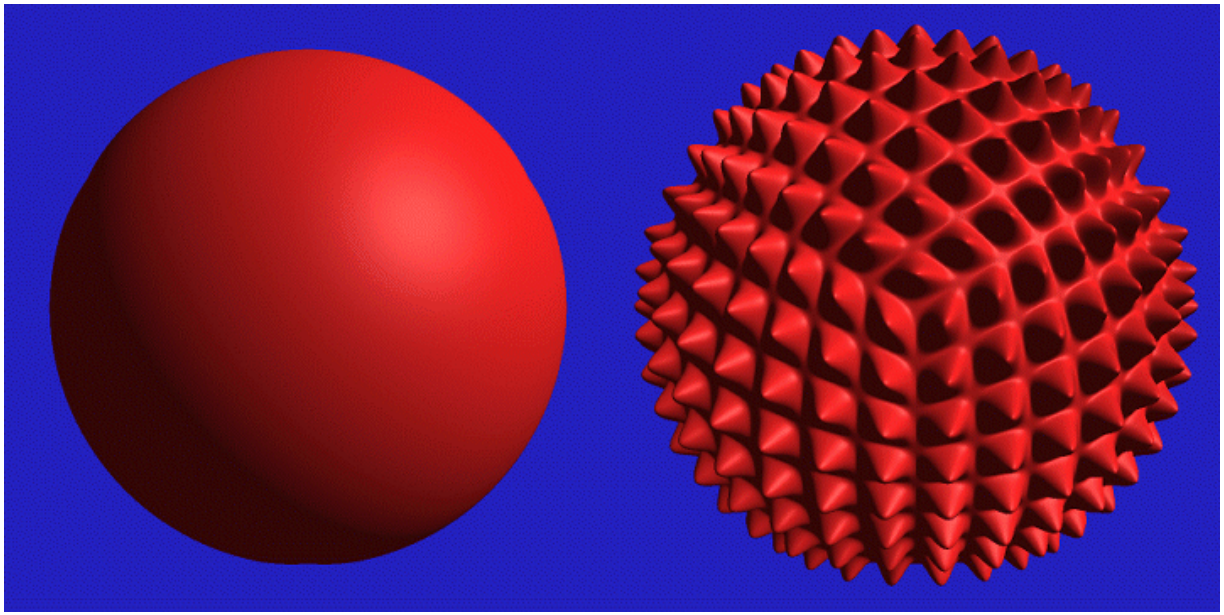
$$S_v = Q_v + P_v \cdot \frac{N}{|N|} + P \cdot (\frac{N}{|N|})_v$$

$$S_u = Q_u + P_u \cdot \frac{N}{|N|} \qquad S_v = Q_v + P_v \cdot \frac{N}{|N|}$$

$$N_S = S_u \times S_v = N + \frac{P_u (N \times Q_v)}{|N|} + \frac{P_v (Q_u \times N)}{|N|}$$

位移贴图

可以使用纹理贴图来真正改变表面点的位置，这被称为位移贴图。它和凹凸贴图的本质区别在哪？



几何顶点必须在可见性判别之前进行位移。

过程式纹理映射

- 除了查找图像, 也可以将纹理坐标做为参数传递给一个函数来动态计算纹理值
  - Renderman, Pixar渲染语言支持这个特性
  - 在新的图形硬件上的顶点着色器也支持
- 优点:
  - 近乎无限的分辨率，非常小的存储开销
  - 对许多特效非常有效
- 缺点是多数情况下比较慢

谢谢！