计算机图形学

真实感图形绘制

------纹理映射技术

王长波　教授

华东师范大学计算机学院

• 提出问题

– 真实感图形绘制：

• 计算机图形输出设备上

– 如何表现物体表面的丰富细节?

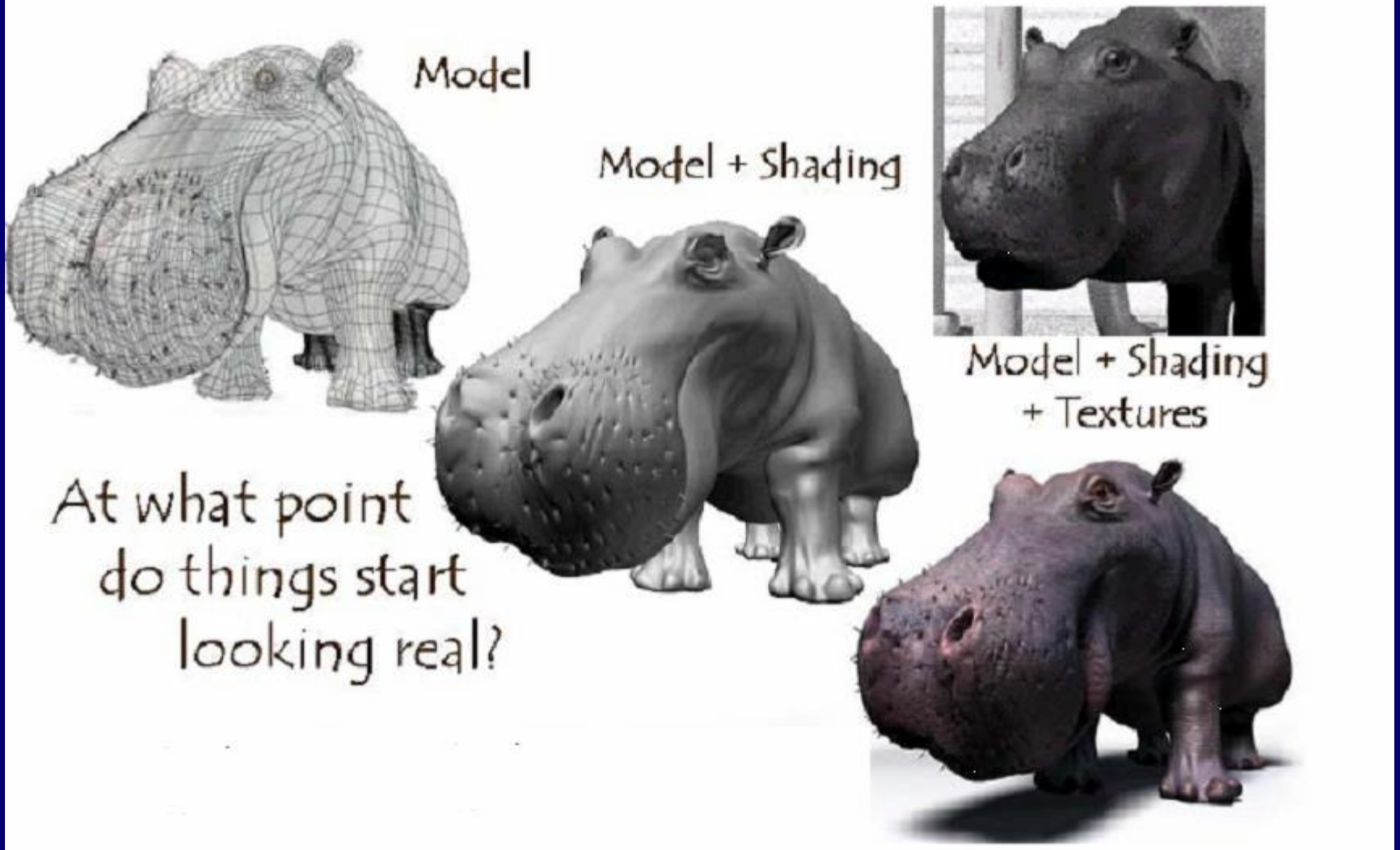
•更精细的几何；
•更精确的光照计算；

计算量太大!

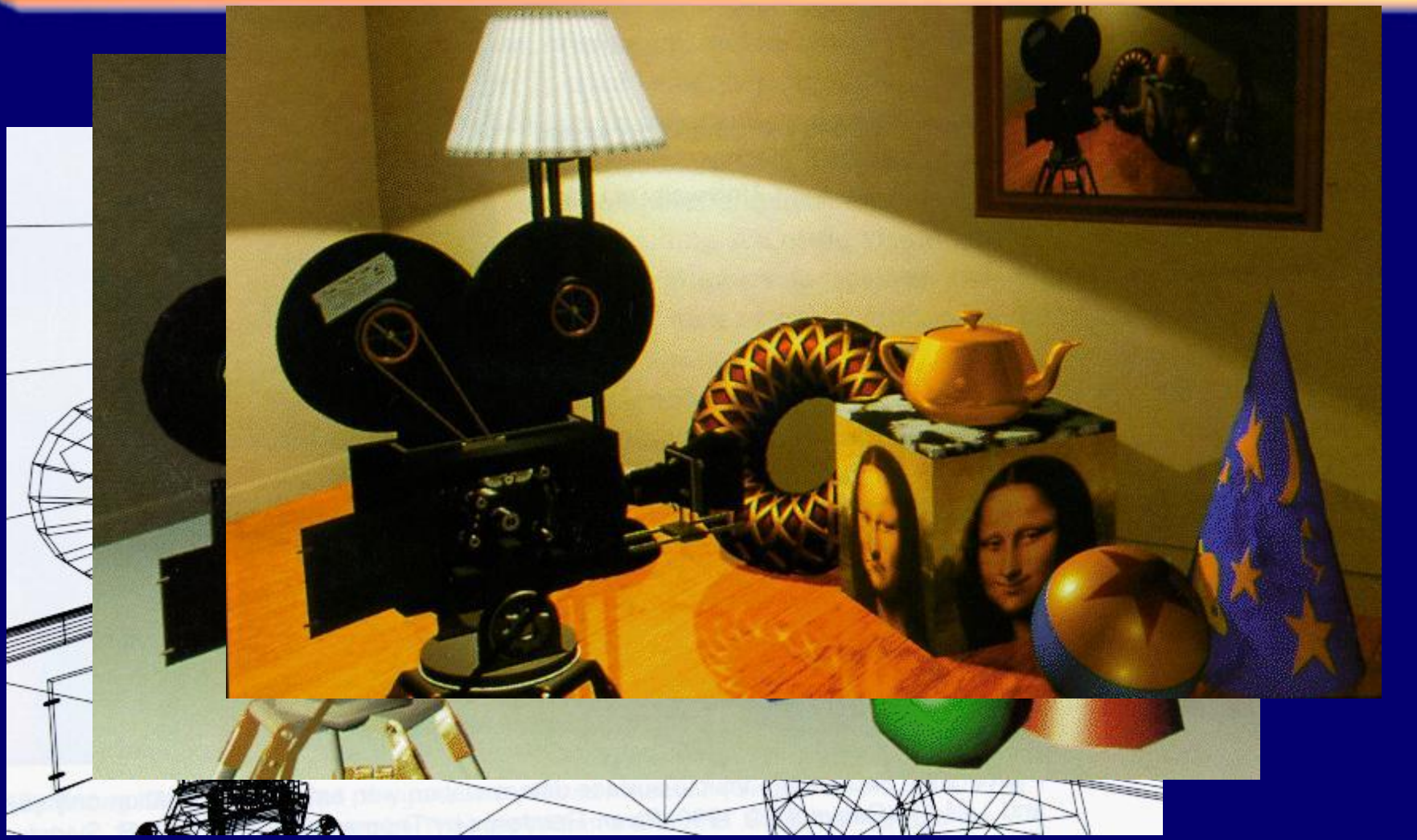
可以采用纹理映射技术!

纹理映射技术是真实感图形的重要绘制技术之一.

"All it takes is for the rendered image to look right" —— Jim Blinn



Example 1



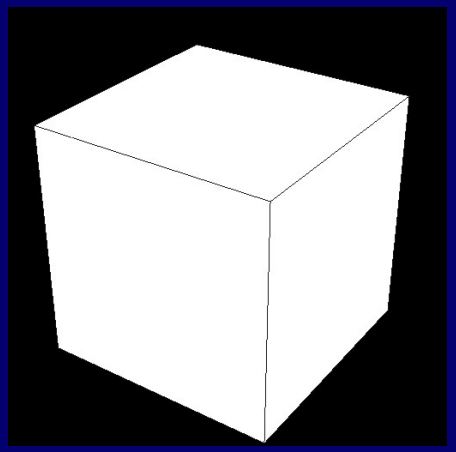
Example 2

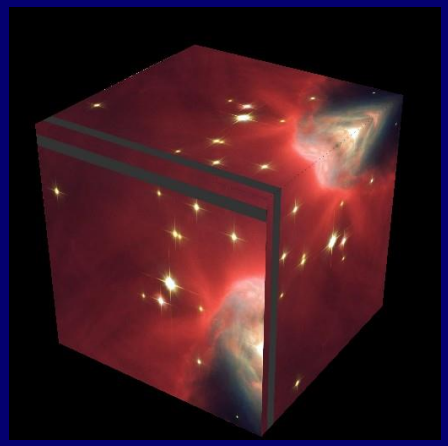
Example 3

提 要

- 纹理映射的基本理论
  - 基本概念
  - 实现纹理映射的基本过程
  - 纹理几何映射关系处理
- 纹理映射的实现
  - 纹理映射的D3D实现
  - 纹理映射的OpenGL实现
- 高级纹理映射技术介绍
  - 纹理优化处理技术
  - 多层纹理映射
  - 凸凹纹理映射
  - 环境映照技术

What is texture mapping?





"All it takes is for the rendered image to look right" —— Jim Blinn

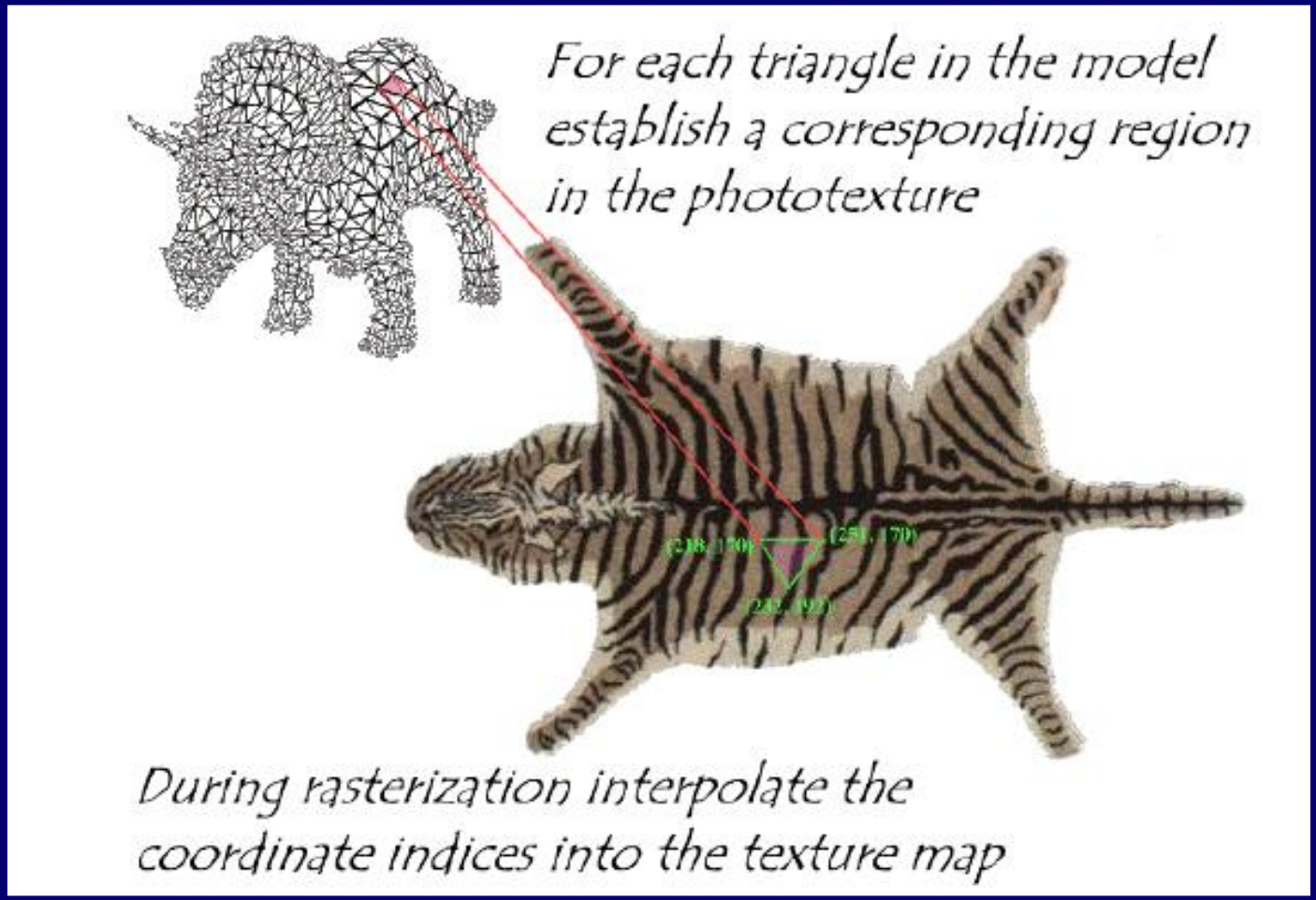
- 纹理贴图是一个用图像、函数或其它数据源来改变表面在每一处的外观的过程。

- 例如，我们不必用精确的几何去表现一块砖墙，而只需把一幅砖墙的图像贴到一个多边形上。

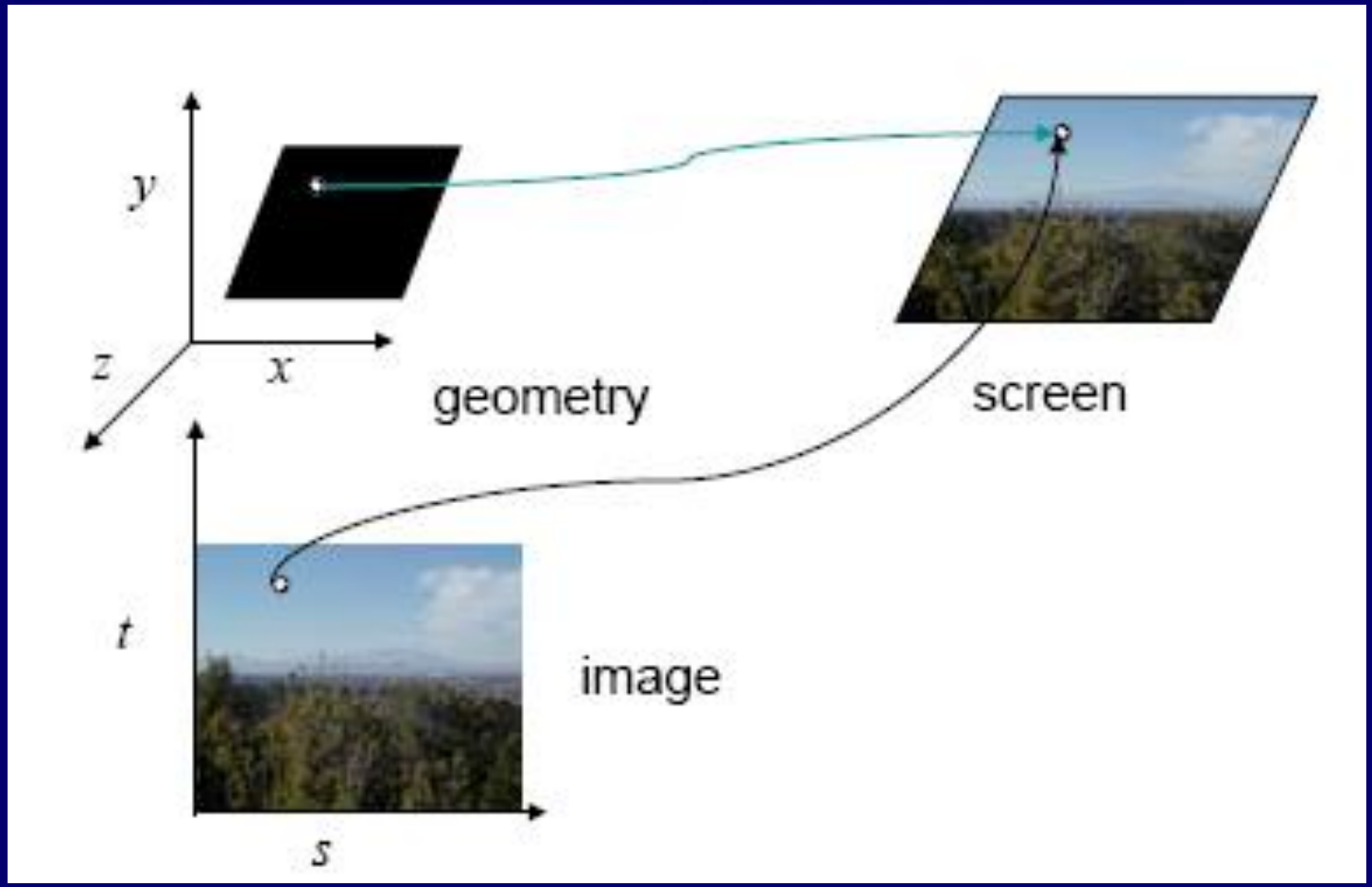
- 除非观察者非常靠近墙，否则我们并不会觉得缺少几何细节。

- 既节省了大量的造型工作量，也节省了内存空间，加快了绘制速度。

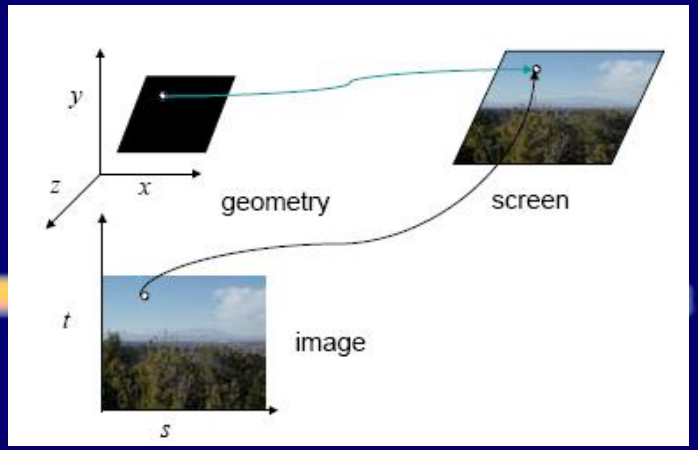
基本原理


For each triangle in the model establish a corresponding region in the phototexture
During rasterization interpolate the coordinate indices into the texture map

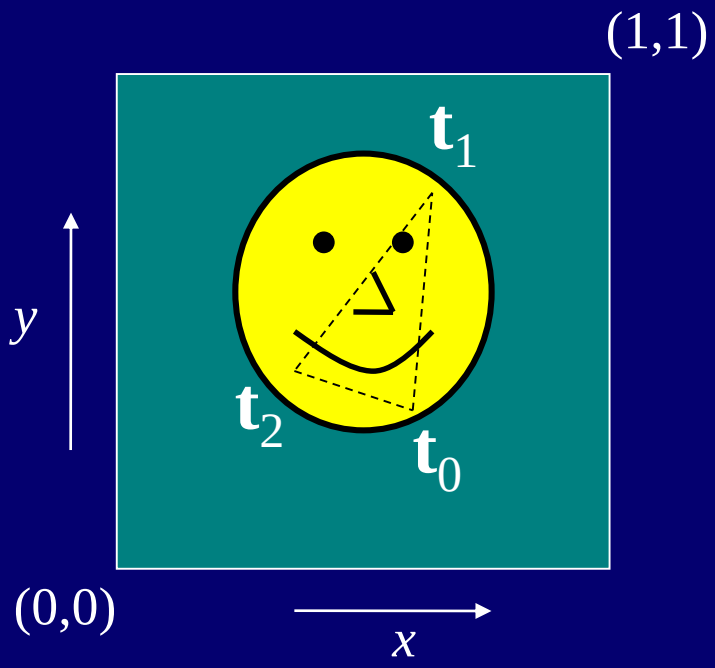
纹理映射----几何和图片之间的对应关系



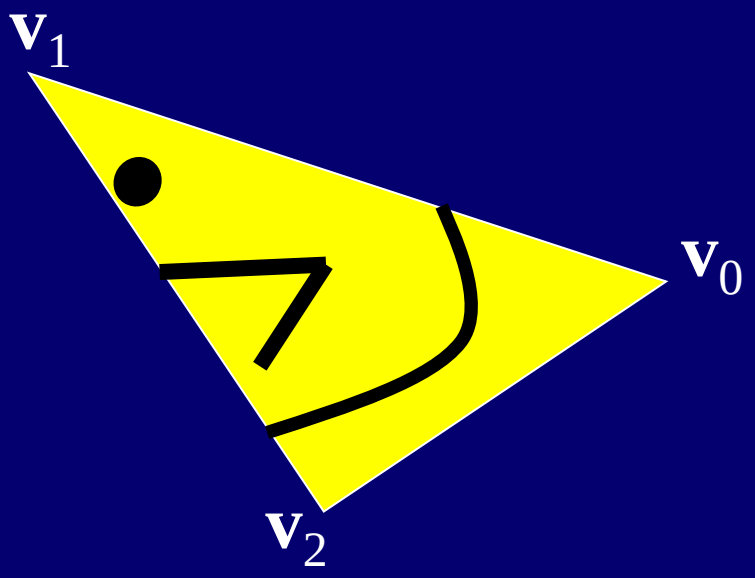
如何映射?



- 通过纹理坐标:
  - 纹理坐标:
    - 每个像素的纹理坐标就是简单的(u , v) 坐标， 它指定了像素正准备被映射到的纹理的纹理单元。
  - 对应纹理图像,左下角为 (0,0) 右上角为 (1,1);
  - 可以设定顶点在图像空间上的纹理坐标[t_x t_y], 该象素就画对应该u, v坐标的RGB颜色;
  - 一般一个四边形的纹理坐标: (0,0)(0,1)(1,0)(1,1)



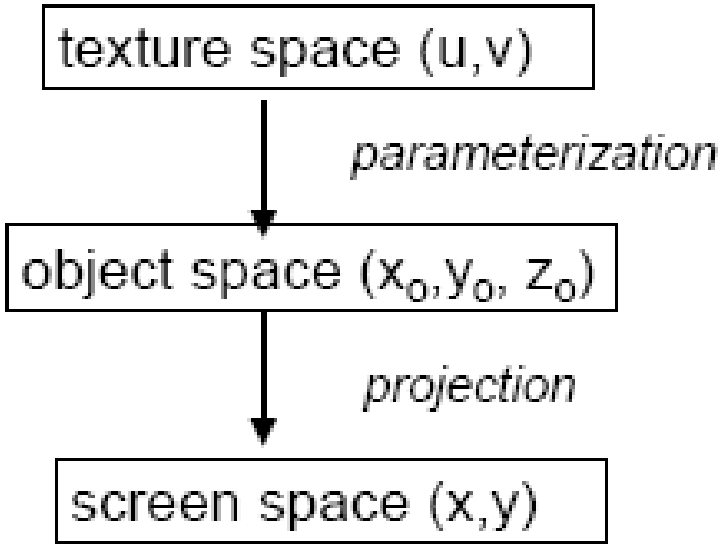
Texture Space



Triangle (in any space)

纹理空间与模型空间的对应

- A means to define surface characteristics of an object using an image
- Mapping a 2D image onto a 3D surface
- Coordinate spaces in texture mapping
  - Texture space (u, v)
  - Object space (x_o,y_o,z_o)
  - Screen space (x, y)

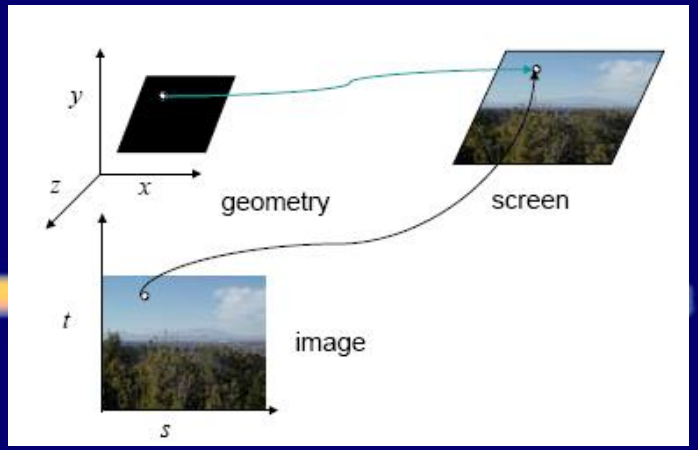


基本实现步骤:

- Three steps
  - -Specify texture
    - Read or generate image
    - Assign to texture
    - Enable texturing
  - - Assign texture coordinates to vertices
  - - Specify texture parameters
    - •Wrapping, filtering, etc.

1. 读入纹理位图

- 一般为二维图片
  - 可以是一张图片(8位, 24位, 32位)
  - 也可以是实际生成的一张图片
  - 长宽尺寸为2的倍数



2. 如何映射

- 利用纹理坐标来定义从图片到几何的映射:
  - 要将该纹理映射到一个三角形上, 需设置三角形的三个顶点在图像空间上的纹理坐标[t_x t_y], 再对应计算每个几何像素在纹理图像上的对应RGB颜色;
  - 一般一个四边形的纹理坐标: (0,0)(0,1)(1,0)(1,1)
  - 3D Game Studio.

3. 纹理插值

- 我们只指定了三角形顶点处的纹理坐标,中间每个象素的纹理坐标(t_i, t_j)可用线性插值;
- 一般双线性插值;
- 从而三角形上每个象素点都可对应地到纹理图片上去取颜色.

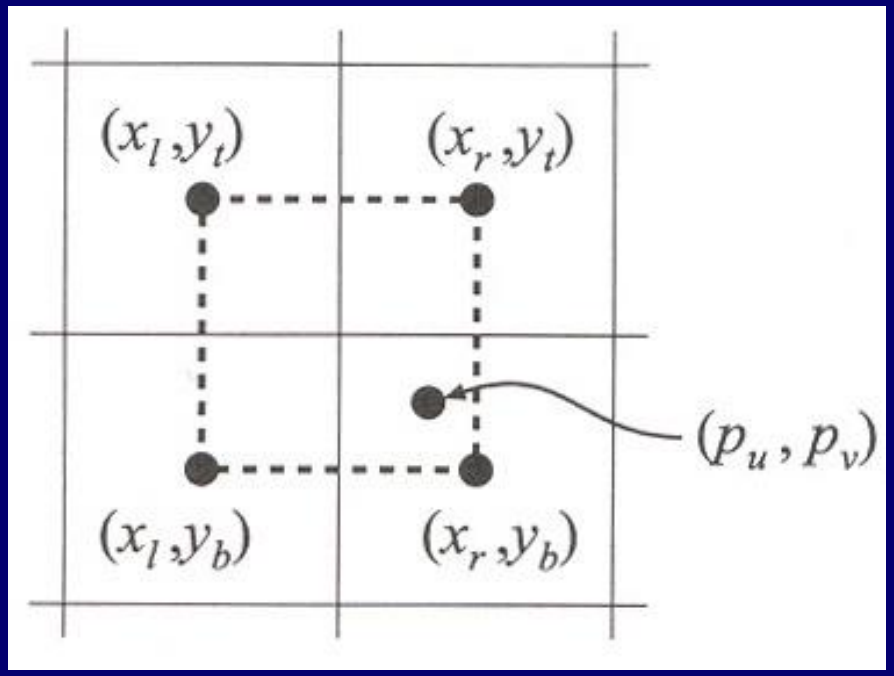
思考？

- 图像与几何体的尺寸不一致怎么办？
- 几何对应的纹理空间超出[0,1)纹理空间怎么办？
- 几何物体本身有颜色怎么办？

问题1：纹理图像缩放(纹理采样方式)

- 在硬件图形加速卡中，纹理图像的大小经常为$2^m \times 2^n$(或者$2^m \times 2^m$)的纹素，其中m和n为非负整数。
- 但是几何尺寸不一定正好是2的倍数,另外几何尺寸可能与纹理图像尺寸差别很大;
- 若投影得到的象素数目比原始纹理大，则需要把纹理图像放大(**magnification**);
- 若投影得到的象素数目比原始纹理小，则需要把纹理图像缩小(**minification**);

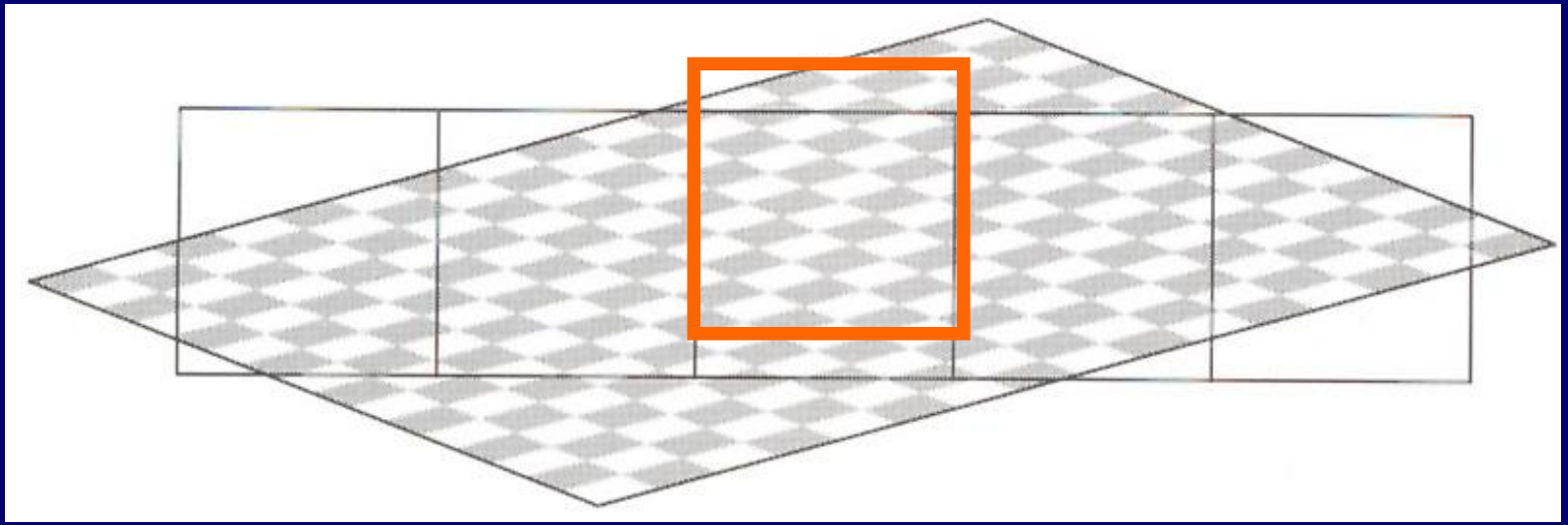
如:双线性插值(Bi-lnear Interpolation)



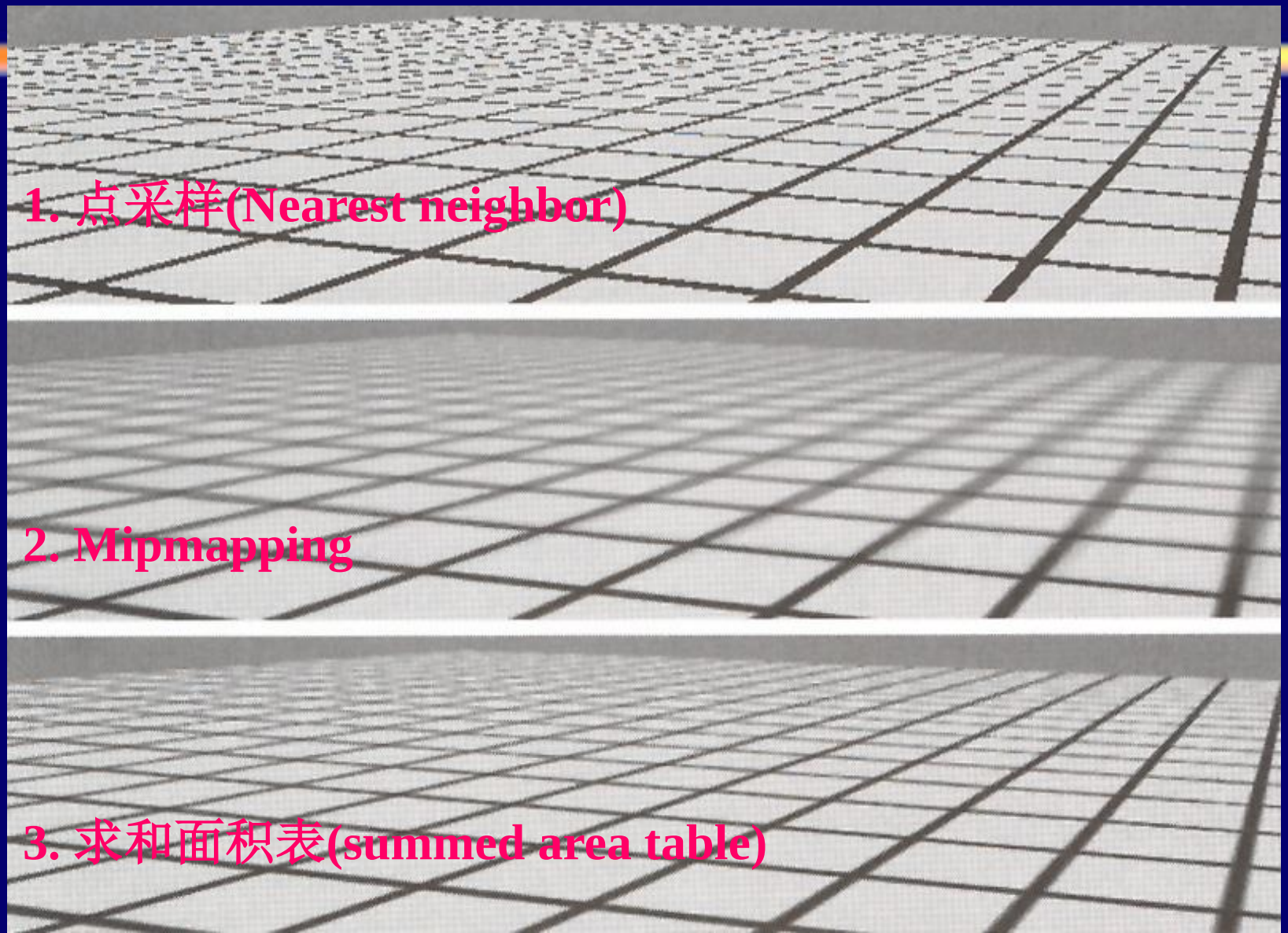
$$\mathbf{b}(p_u, p_v) = (1-u')(1-v')\mathbf{t}(x_l, y_b) + u'(1-v')\mathbf{t}(x_r, y_b)$$
$$+ (1-u')v'\mathbf{t}(x_l, y_t) + u'v'\mathbf{t}(x_r, y_t)$$

纹理缩小

- 当纹理图像缩小时，多个纹素可能覆盖一个象素单元。为了得到每个象素正确的颜色，应该综合考虑影响该象素的那些纹素。

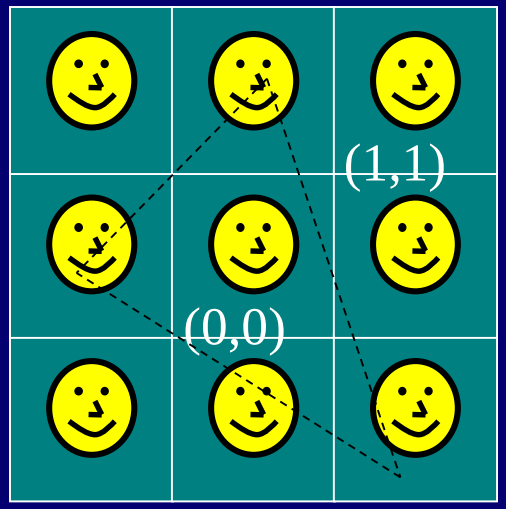


棋盘格纹理多边形通过一行象素，显示多个纹素可影响单个象素

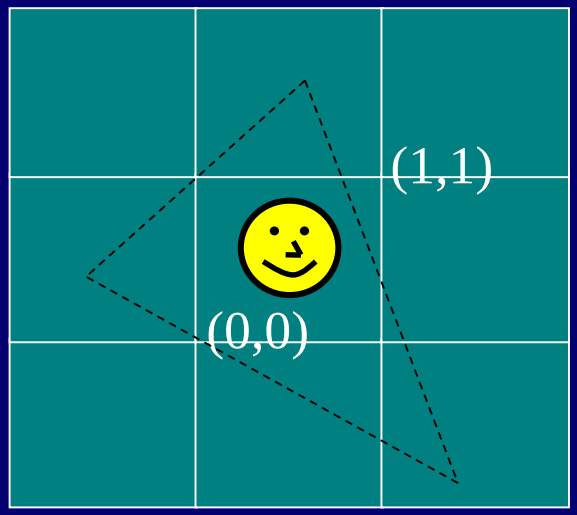

1. 点采样(Nearest neighbor)
2. Mipmapping
3. 求和面积表(summed area table)

- 最近邻域法：
  - 选择在象素中心可见的纹素。但会引起严重的走样现象，见上图。当这类表面相对视点移动时，走样现象更加明显，称为时间走样(temporal aliasing)。
- 双线性插值：
  - 效果仅比最近邻域法稍好，也会引起较严重的走样现象。
- Mipmap方式：
  - 对纹理进行预处理，建立多个纹素覆盖单个象素的快速逼近计算的数据结构。这样，一个采样点可以检索出一个或多个纹素的效果。

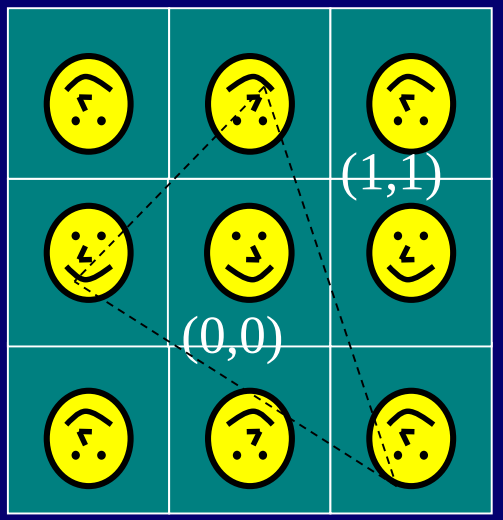
问题2：纹理重复方式



Tiling



Clamping



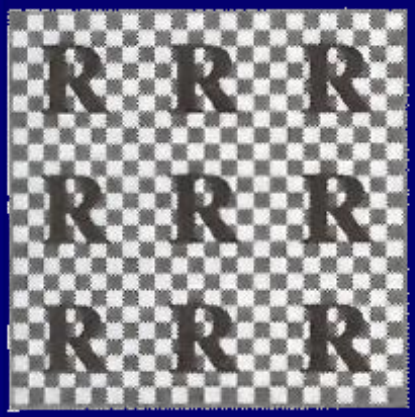
Mirroring

Texture Space

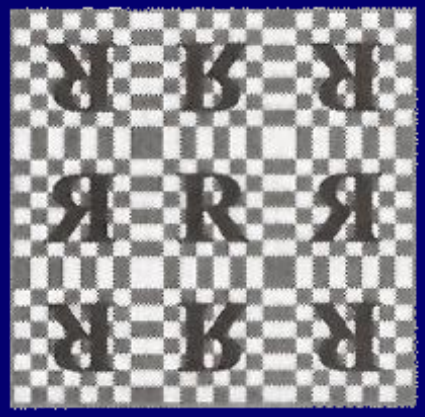
常用对应函数

- **wrap or tile:** 纹理图像在表面重复。例子：地上的大理石贴图
- **mirror:** 纹理图像在表面重复, 但每隔一幅进行翻转(flipped)。这样在纹理的边界处，纹理可以保持连续。
- **clamp :** 把[0,1)范围之外的进行截断。截断到[0,1)内的半个纹素。
- **border :**参数范围在[0,1) 之外的用单独定义的边界颜色或把纹理的边作为边界。用于在表面上印花样，地形绘制中相邻纹理的缝合。截断到[0,1)外的半个纹素。

例子

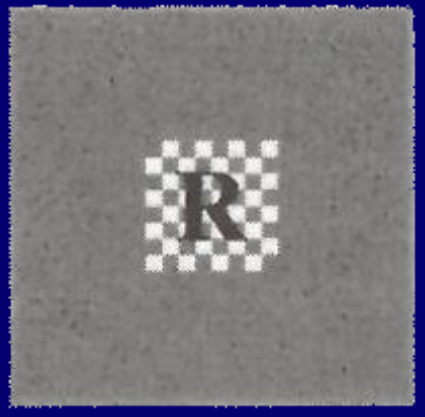


repeat



mirror

clamp



border

问题3：纹理与背景的叠加融合

- **Replace：** 把原来表面的颜色替换为纹理的颜色

- **Decal(印花)：** 与替换类似，但是若纹理中包含Alpha值，则用它与表面的颜色进行混合。

- **Modulate(调节)：** 把表面的颜色与纹理颜色相乘 .

提 要

- 纹理映射的基本理论
  - 基本概念
  - 实现纹理映射的基本过程
  - 纹理几何映射关系处理
- **纹理映射的实现**
  - **纹理映射的D3D实现**
  - **纹理映射的OpenGL实现**
- 高级纹理映射技术介绍
  - 纹理优化处理技术
  - 多层纹理映射
  - 凸凹纹理映射
  - 环境映照技术

Direct3D纹理映射过程

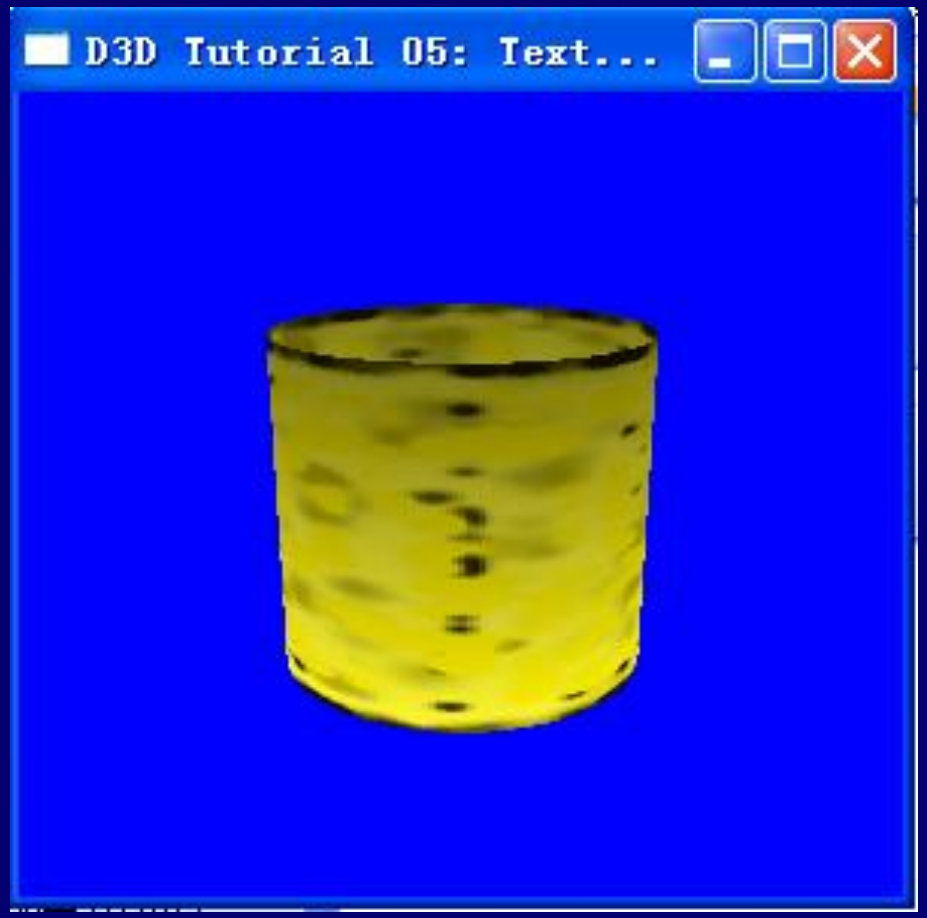
- 载入纹理
- 分配顶点纹理坐标
- 设置当前渲染纹理
- 设置纹理渲染状态
- 渲染顶点缓冲区, 绘制物体

- 载入纹理
  - CreateTexture（）
  - 载入纹理：LoadBmpTexture24Bit（）
    - LockRect和UnlockRect访问纹理资源
  - 从磁盘生成并载入纹理
    - D3DXCreateTextureFromFile
    - 程序结束时释放指针：g_pTexture->Release()

- 分配顶点纹理坐标
- 设置当前渲染纹理
  - SetTexture（）
- 设置纹理渲染状态
  - SetTextureStageState（）
- 渲染顶点缓冲区
  - SetStreamSource(0, g_pVB, 0, sizeof(CUSTOMVERTEX));
  - SetFVF(D3DFVF_CUSTOMVERTEX);
  - DrawPrimitive(D3DPT_TRIANGLESTRIP, 0, 2);

- 设置纹理采样方式
  - 调用函数SetSamplerState设置纹理采样方式
  - 四种：
    - Nearest-Point Sampling
    - Linear Texture Filtering
    - Anisotropic Texture Filtering
    - Texture Filtering with Mipmaps

- 程序演示



OpenGL纹理映射过程

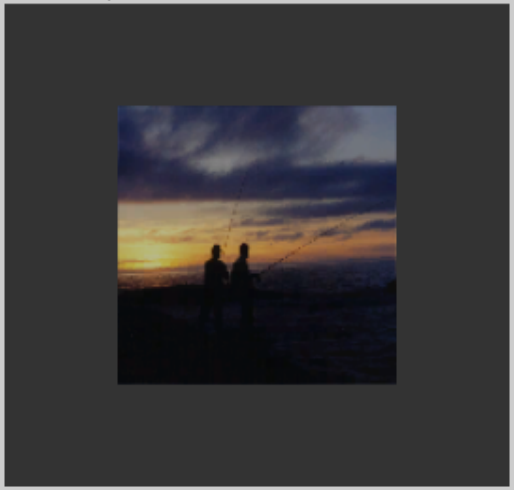
- 将纹理装入内存
  - ***LoadFromFile***调用完成图片解码读入的工作;
  - 图片格式有**.bmp/.jpg/.tif/.gif/.rgb**等。
- 将纹理发送给OpenGL管道:
  - **glGenTextures (n,*textures);**
- 分配顶点纹理坐标
  - 指定当前纹理坐标的命令是**glTexCoord*(s,t,r,q);**
- 纹理映射参数设置

- 对于这种超出纹理图范围的情况，可用:
  - **glTexParameter*(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S/T,GL_REPEAT/GL_CLAMP);**
  - 选择:
    - GL_REPEAT：当纹理比表面小时重复使用纹理以填满每个点。
    - GL_CLAMP：比1大的当作1，比0小的当作0。

- 对纹理的缩放等, OpenGL提供了滤波函数:
  - **glTexParameter*(GL_TEXTURE_2D,GL_TEXTURE_MAG/MIN_FILTER,GL_NEAREST/GL_LINEAR);**。
  - 放大和缩小的处理参数:
    - GL_NEAREST：取比较接近的那个像素。
    - GL_LINEAR：以周围四个像素的平均值做为纹理。
    - bilinear：二次插值，精度更高，但需要自己动手计算。

- 程序演示

Screen-space view



Texture-space view

Command manipulation window

```
GLfloat border_color[ ] = {  1.00 ,  0.00 ,  0.00 ,  1.00 };
GLfloat env_color[ ] = {     0.00 ,  1.00 ,  0.00 ,  1.00 };

glTexParameterfv(GL_TEXTURE_2D, GL_TEXTURE_BORDER_COLOR, border_color);
glTexEnvfv(GL_TEXTURE_ENV, GL_TEXTURE_ENV_COLOR, env_color);

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,  GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,  GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,  GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,  GL_REPEAT);
glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,  GL_MODULATE);

glEnable(GL_TEXTURE_2D);
gluBuild2DMipmaps(GL_TEXTURE_2D, 3, w, h, GL_RGB, GL_UNSIGNED_BYTE, image);
glColor4f(  0.60  , 0.60  , 0.60  , 1.00  );
glBegin(GL_POLYGON);
glTexCoord2f( 0.0  , 0.0  ); glVertex3f( -1.0 , -1.0 , 0.0  );
glTexCoord2f( 1.0  , 0.0  ); glVertex3f( 1.0  , -1.0 , 0.0  );
glTexCoord2f( 1.0  , 1.0  ); glVertex3f( 1.0  , 1.0  , 0.0  );
glTexCoord2f( 0.0  , 1.0  ); glVertex3f( -1.0 , 1.0  , 0.0  );
glEnd();
```

Click on the arguments and move the mouse to modify values.

练习

- 给定一张图片,绘制一个立方体, 利用VC++和D3D编写代码实现不同的纹理映射方式.
- 可以在Direct3D\Tutorials的Tut05_Textures上修改.

提 要

- 纹理映射的基本理论
  - 基本概念
  - 实现纹理映射的基本过程
  - 纹理几何映射关系处理
- 纹理映射的实现
  - 纹理映射的D3D实现
  - 纹理映射的OpenGL实现
- 高级纹理映射技术介绍
  - 纹理优化处理技术
  - 多层纹理映射
  - 凸凹纹理映射
  - 环境映照技术

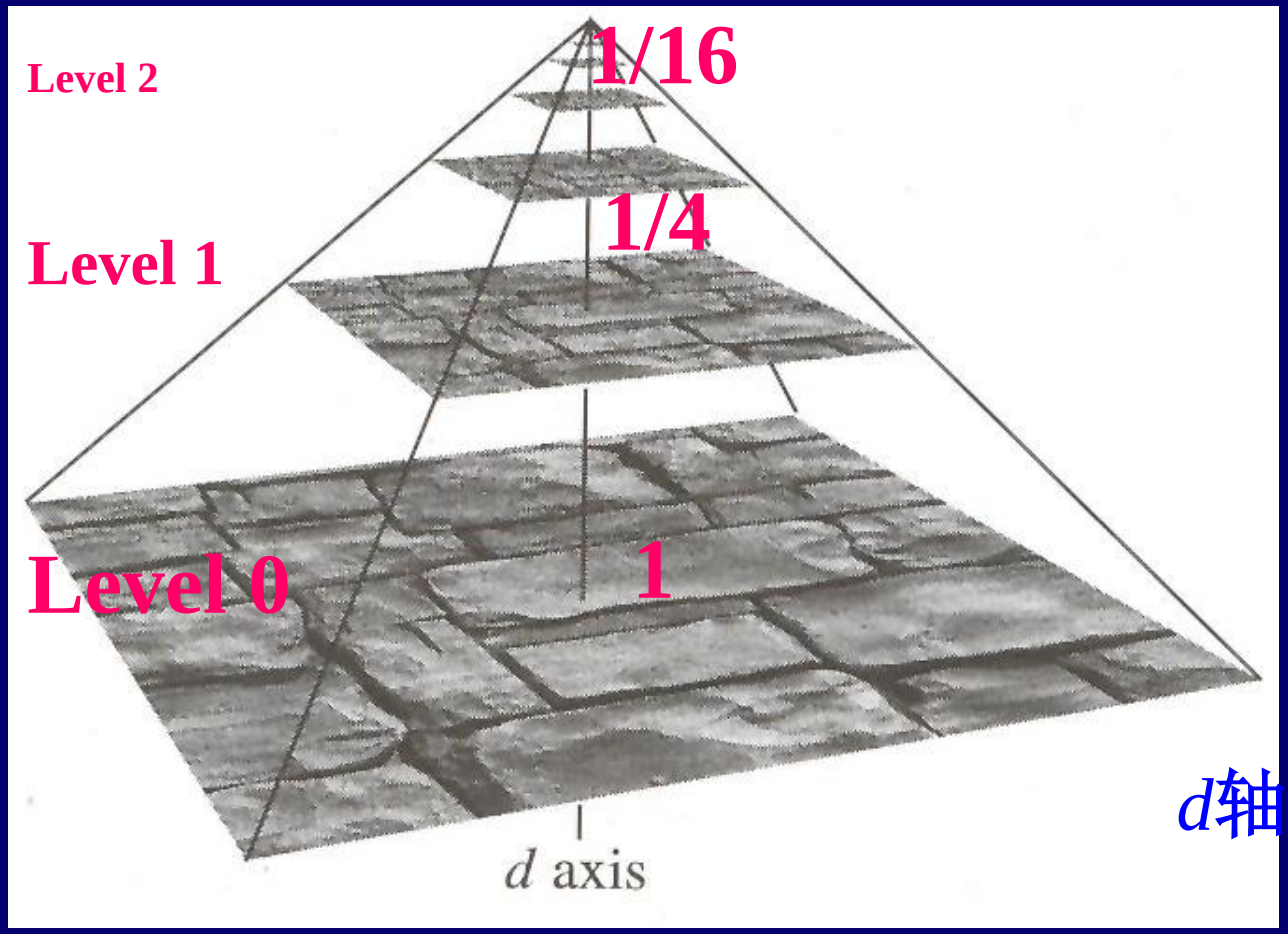
- 高级纹理映射技术
  - 纹理优化处理技术
    - Mipmapiing
    - **Texture Caching**
    - **纹理压缩**
  - 多层纹理映射
  - 凸凹纹理映射
  - 环境映照技术

纹理优化处理技术

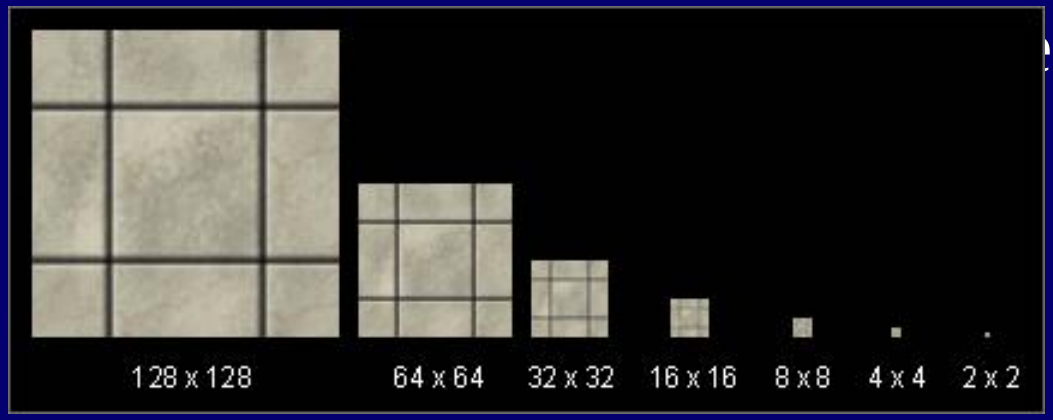
1. Mipmapping技术

--------避免图像的过度放大或缩小

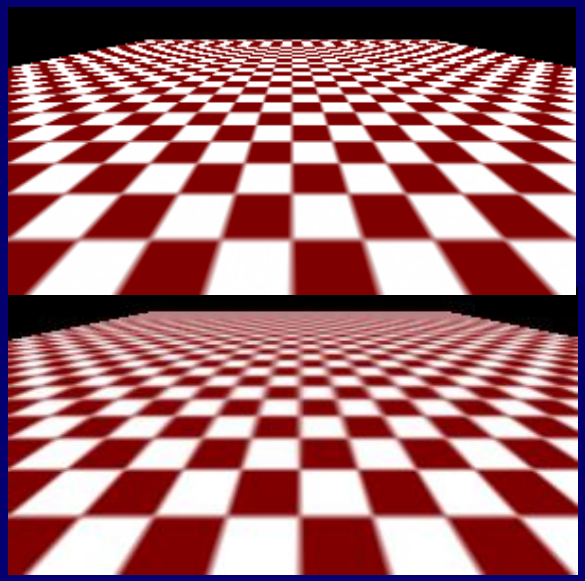
金字塔结构

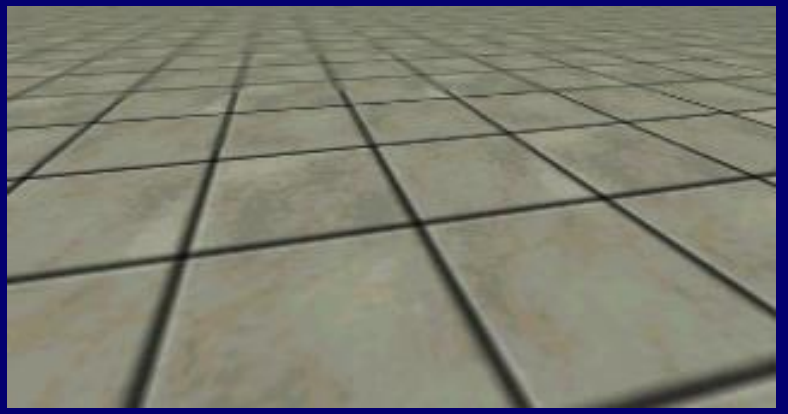


- **Mipmapping是纹理反走样最受欢迎的方法。目前，很多图形加速卡支持该功能;**
- **根据几何的大小, 调整纹理的大小, 以避免拉伸和缩小;**
- **等比性地缩放, 图片大小必须为2的n次。**
- 调用函数SetSamplerState设置即可.



Without MIP-mapping





With MIP-mapping

2. Texture Caching(纹理缓存)

- 复杂的应用需要相当多的纹理, 不一定把所有的纹理都一次性送到显存;

- 有多种纹理高速缓存技术:
  - 在速度和内存中的纹理数目之间取得平衡。例如：当贴了纹理的多边形离视点较远时，可只载入需要的子纹理。

使用纹理内存的一般原则：

- 尽量使纹理小：不要大于使得纹理需要放大。
- 尽量使得使用相同纹理的多边形成组。
- 采用Tiling或者Mosaicing技术：把一些小纹理拼成一块大纹理，这样可以避免纹理的切换，加快存取的速度。

3. 纹理压缩

- 一个直接针对纹理内存和带宽的解决方法是固定速率的纹理压缩(fixed-rate texture compression)。
- 通过硬件即时对压缩的纹理进行解压，所需的纹理内存可减少，从而增加了有效Cache的大小，同时减少了带宽需要。
- D3D提供了专门的纹理压缩函数。

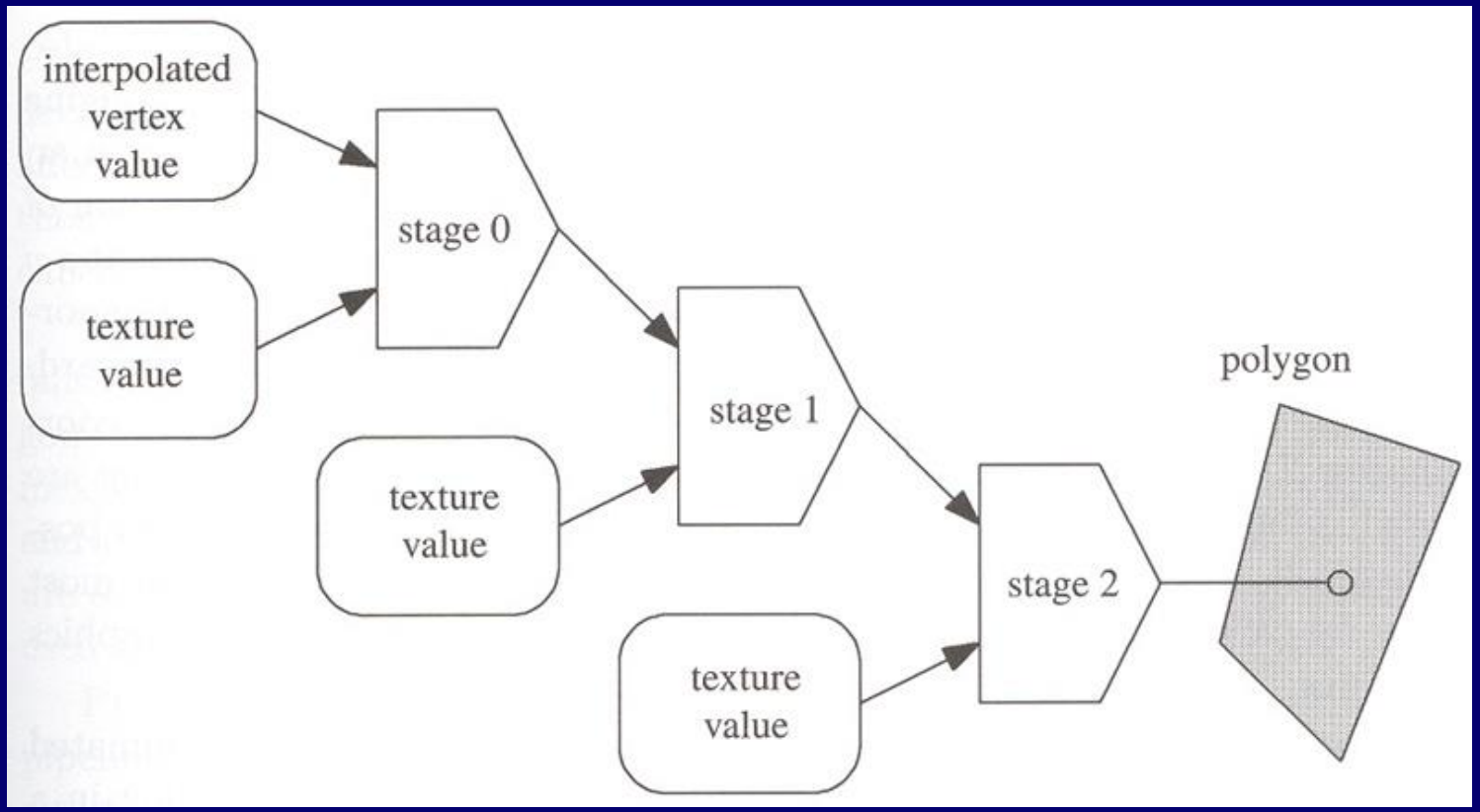
纹理压缩实现

- 查询是否支持DXT压缩
  - IsCompressedTextureFormatOK（）
- 生成压缩纹理
  - D3DXCreateTextureFromFileEx（）

4. 多重纹理绘制

- 图形学光照方程的各部分可以在不同的pass分别计算，每个pass是对前一个pass结果的修改,并最终得到一采样颜色。
- 多重绘制技术是指将图形绘制的每个pass绘制出的中间结果存成纹理,一层层地映射到几何物体上, 这种技术称为多重绘制技术(multipass rendering)。
- 多重绘制的基本思路是：每重绘制计算光照明方程的一部分，并采用frame buffer存贮中间结果。通过图像合成等技术，可以计算得到任意复杂的光照方程。
- 有很多效果可通过多重绘制来得到的，如：运动模糊、景深、反走样、软影、平面反射等等。
- 目标：用更多精致的表面光照方程。

过程


interpolated vertex value
texture value
stage 0
texture value
stage 1
texture value
stage 2
polygon

- **Quake III的10个pass**
  - **(Passes 1-4: accumulate bump map)**
  - **Pass 5: diffuse lighting**
  - **Pass 6: base texture (with specular component)**
  - **(Pass 7: specular lighting)**
  - **(Pass 8: emissive lighting)**
  - **(Pass 9: volumetric/atmospheric effects)**
  - **(Pass 10: screen flashes)**

  - **Quake III 可以同时支持十级的纹理映射。**

- 多层纹理映射的实现:
  - D3D支持8级纹理
  - 设置纹理层混合方式:
    - SetTextureStageState(0, D3DTSS_COLOROP, D3DTOP_ADD);//两个混合参数相加后输出
    - SetTextureStageState(0, D3DTSS_COLOROP, D3DTOP_MODULATE);//两个混合参数相乘后输出
  - 设置纹理层坐标索引： SetTextureStageState(0, D3DTSS_TEXCOORDINDEX, 0);
  - 设置渲染纹理： SetTexture(0, g_pTexture);
  - 经典的QuakeIII引擎定义了10个多步纹理映射技术。

- 程序演示

其它多重纹理技术(1)

- **Alpha mapping**
  - Alpha值可以用来模拟很多有趣的效果。一种效果为印花(decaling)，比如要把一朵花的图像印到一个茶壶上。印上去的不是整幅图像，而只是花的部分。
  - 另一个应用是用来模拟剪纸(cutout)效果。比如通过把一颗树的纹理贴到一个多边形上来模拟树。
  - 但是，当视点旋转时，由于纹理树没有厚度，会出现缺陷。一个解决方法为拷贝得到另一个多边形并沿树干转90度，构成一颗交叉树(cross tree)。

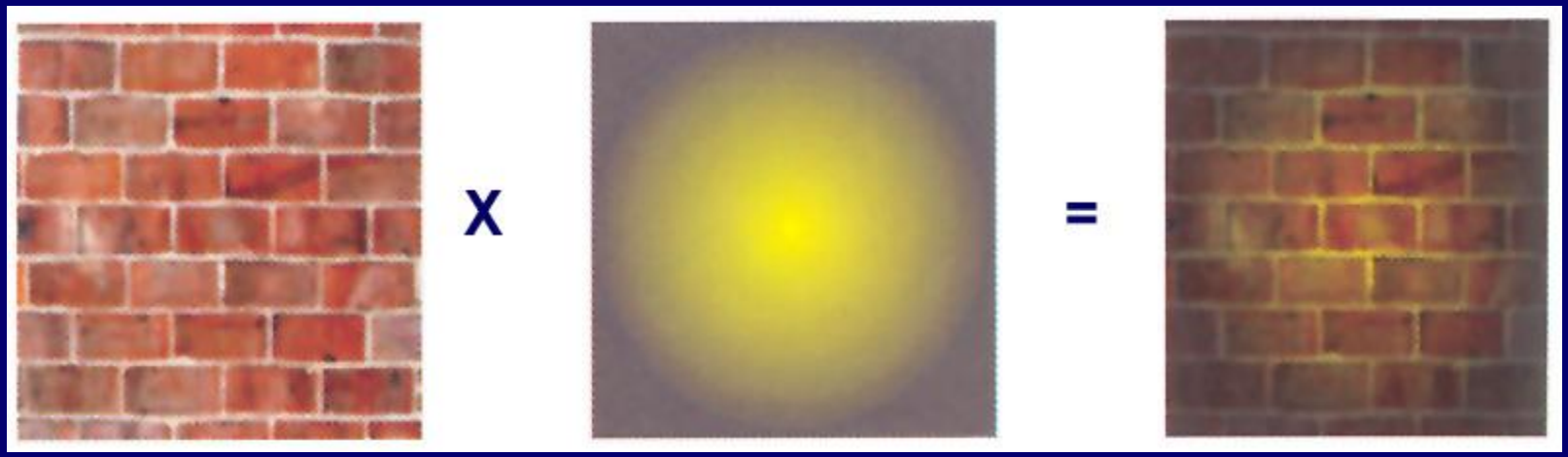
纹理树

alpha通道

Tree

其它多重纹理技术(2)

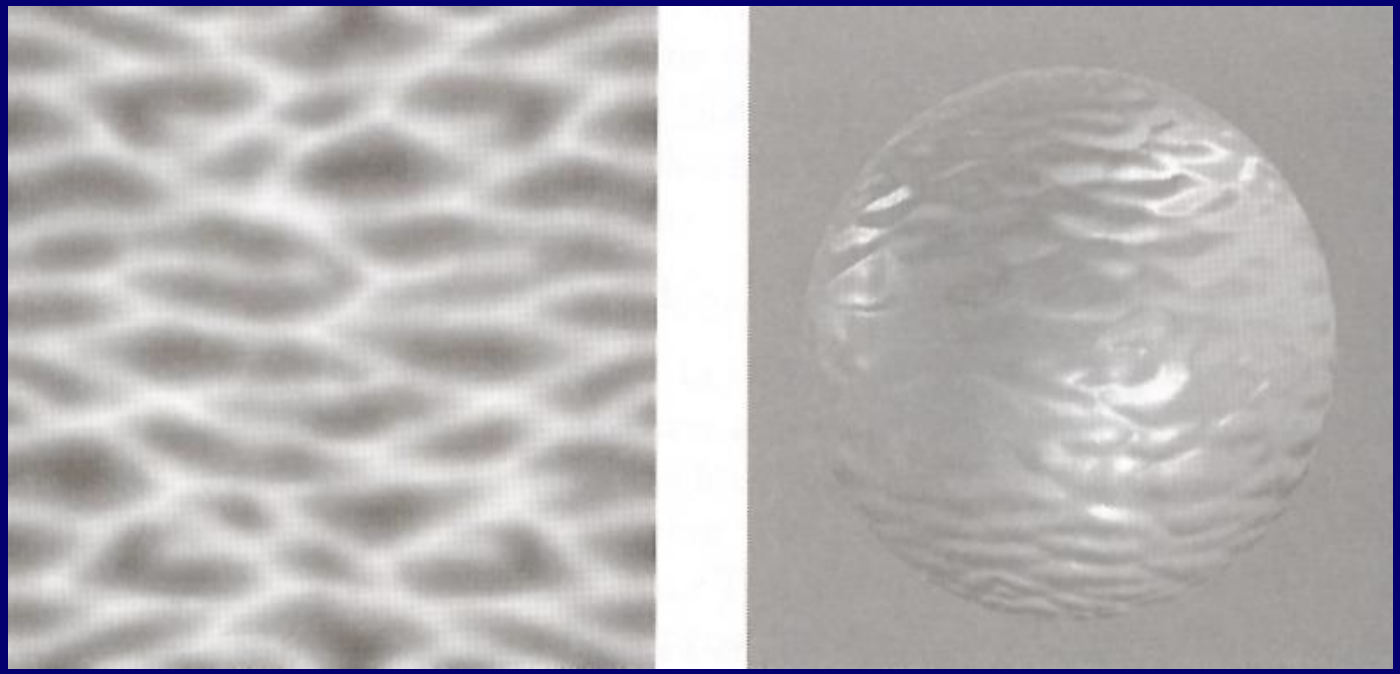
- **Light mapping**
  - 对于静态的光源，物体表面的漫反射部分从任何角度看都是相同的。由于这一与视点无关的性质，这部分光对表面的贡献可以用纹理附加到表面上来刻画。
  - 通过预先单独计算的捕获光照效果的纹理，还可以实现类似Phong Shading的效果。



凹凸纹理贴图(Bump Mapping)

凹凸纹理贴图(Bump Mapping)

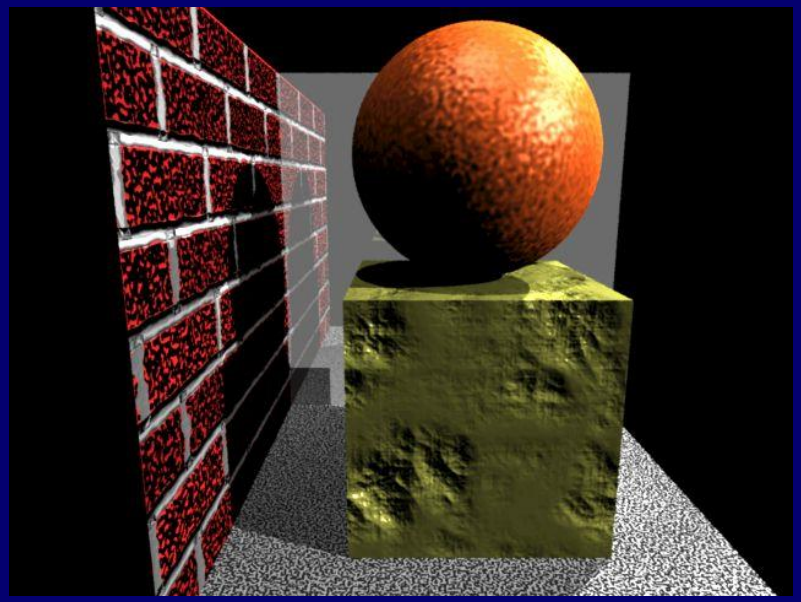
- 凹凸纹理贴图最早由Blinn于1978年提出，是一种模拟不平表面(凹凸、皱纹 、波浪起伏等)的一种方法。

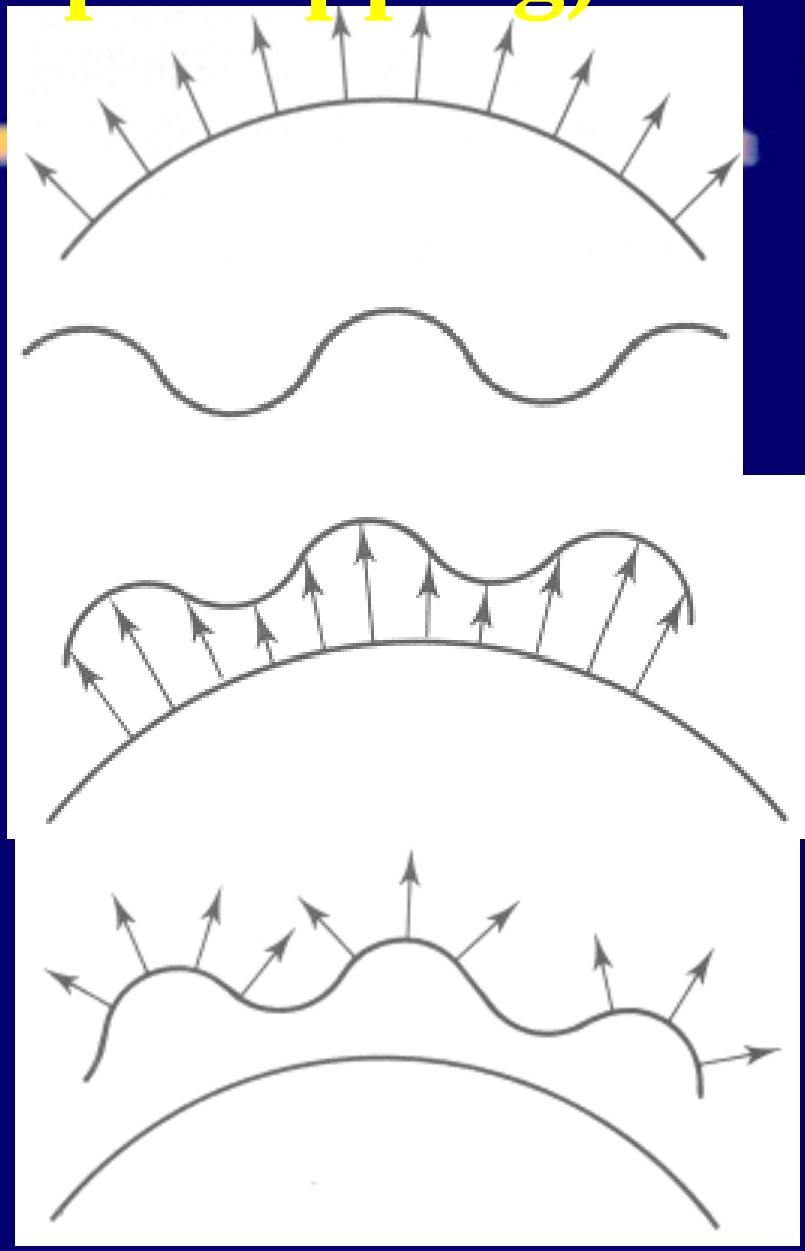


把波浪状的高度场凹凸图像应用于一个球，用Per-Pixel照明绘制。

凹凸纹理贴图(Bump Mapping)

- create illusion of complex geometry model
- control shape effect by locally perturbing surface normal





凹凸纹理可以模拟需要很多多边形才能实现的特征，如衣服的褶痕、动物的肌肉组织等。

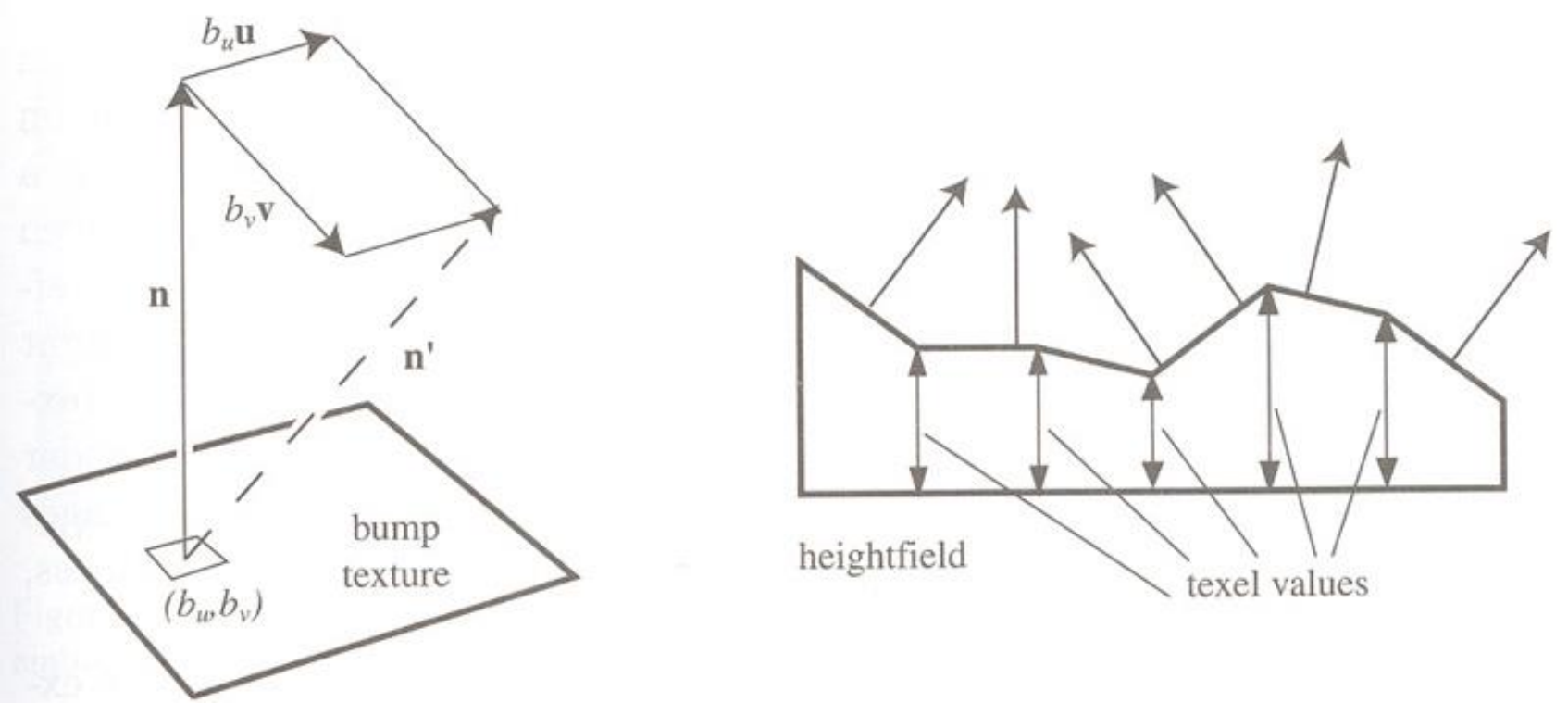
左图采用**Normal map**作为**bump**。

基本思想

- 凹凸纹理的基本思想是：用纹理去修改物体的法向而不是颜色。
- 物体表面的几何法向保持不变，我们仅仅改变光照明模型计算中的法向。
- 凹凸纹理没有物理意义上等价的操作。

Offset map

Height-field map



在左图中，法矢量**n**在**u**和**v**方向被凹凸纹理上的(b_u, b_v)值修改，得到**n'**。在右图中，一个高度场和它对**shading normal**中的影响。

- 凸凹纹理映射
  - 模拟粗燥物体表面的凸凹不平的细节；
  - 通过一张表示物体表面凸凹程度的纹理，对纹理坐标进行干扰；
  - 由三张纹理映射图组成:
    - 原始纹理图（0）
    - 凸凹纹理（1）
    - 环境映射图（2）

凹凸方式一：浮雕凹凸纹理贴图

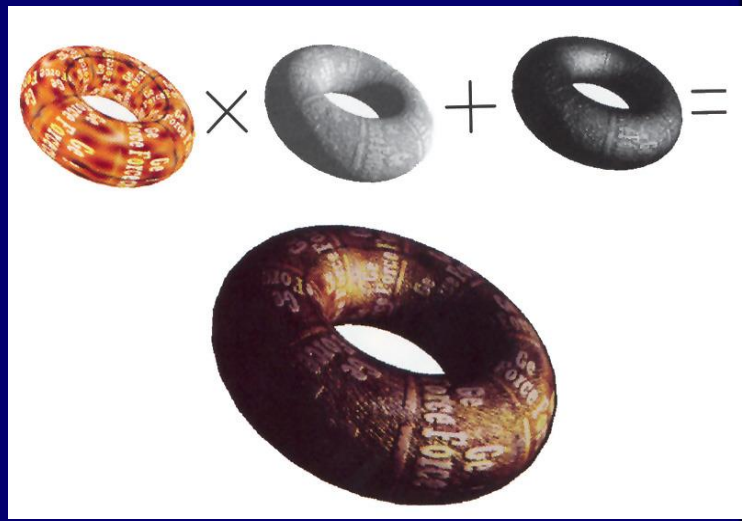
- 浮雕凹凸纹理贴图是最早用于实时系统的凹凸纹理贴图技术之一。它借鉴了二维图像处理中的相关技术：
  - 首先拷贝高度场图像，稍做偏移，然后与原图相减。

原图　　看起来从左上角打光的浮雕效果　　看起来从右下角打光的浮雕效果

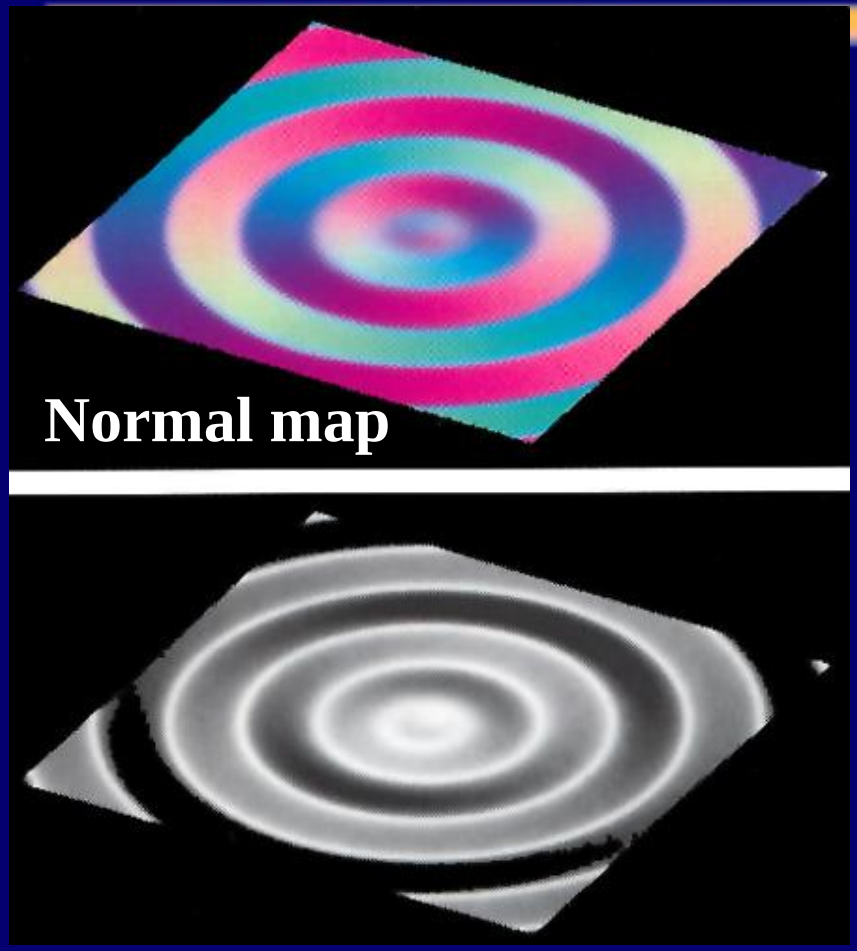
凹凸方式二: 点积凹凸纹理贴图

- 点积凹凸纹理(有时也称dot3 bump mapping)概念简单，是现代图形硬件的主流方法。

• 它存贮的不是高度或斜率，而是把真实的表面法向量(*x*, *y*, *z*)存贮在法向图(**normal map**)中。每个8bits分量映射到[-1,1]。例如： *x*分量的0表示-1.0, 255表示1.0。



• 对于每个象素，我们把单位化后的光矢量与凹凸纹理中的法向进行点积。而点积是一种特殊的texture blending函数，DirectX提供的操作称为D3DTOP_DOTPRODUCT3.



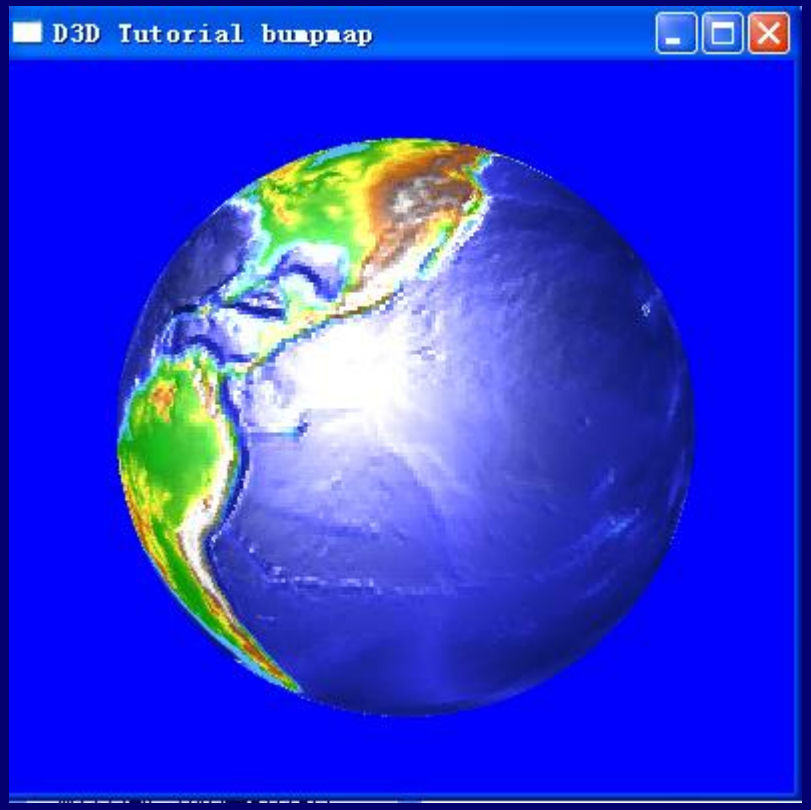
- 对于每个象素，我们把单位化后的光矢量与凹凸纹理中的法向进行点积。而点积是一种特殊的texture blending函数，DirectX提供的操作称为D3DTOP_DOTPRODUCT3.
- 由于点积正好是漫反射分量，因而使得结果呈现凹凸状。

采用Normal Map得到的凹凸纹理贴图效果。

实现

- 凸凹纹理被用来表示物体表面相邻象素的高度差:
  - 载入一张轮廓图片来生成凸凹纹理
  - 通过程序计算凸凹纹理
- 凸凹纹理状态设置
  - SetTextureStageState(1, D3DTSS_COLOROP, D3DTOP_BUMPENVMAPLUMINANCE);
- 凸凹纹理的计算
  - $D_u' = D_u M_{00} + D_v M_{01}$
  - $D_v' = D_u M_{01} + D_v M_{11}$
  - SetTextureStageState(1, D3DTSS_BUMPENVMAT00, F2DW(0.5f));

- 程序演示



环境映照

- cheap way to achieve reflective effect
  - generate image of surrounding
  - map to object as texture

Sphere Mapping

- 最早由Williams在1983年提出，是第一个商用图形卡支持的EM方法。
- 纹理图像通过正投影观察一个纯反射球面的外形来得到，故得到的纹理称为球面图(sphere map)。
- 将几何体参数化到一个球上, 然后进行映射。

Sphere Mapping

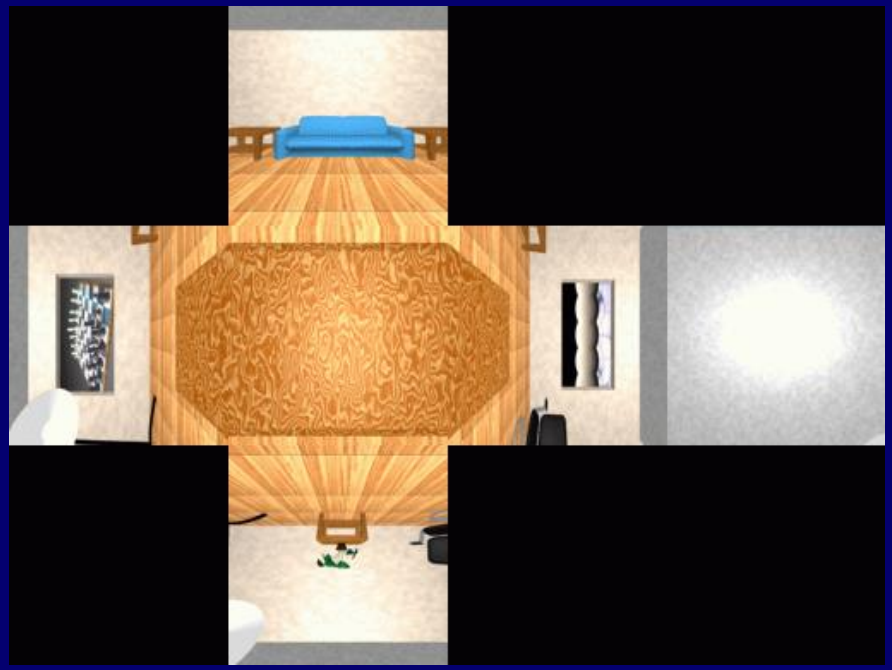
- texture is distorted fish-eye view
  - point camera at mirrored sphere
  - spherical texture coordinates

立方体环境映照
(Cubic Environment Mapping)

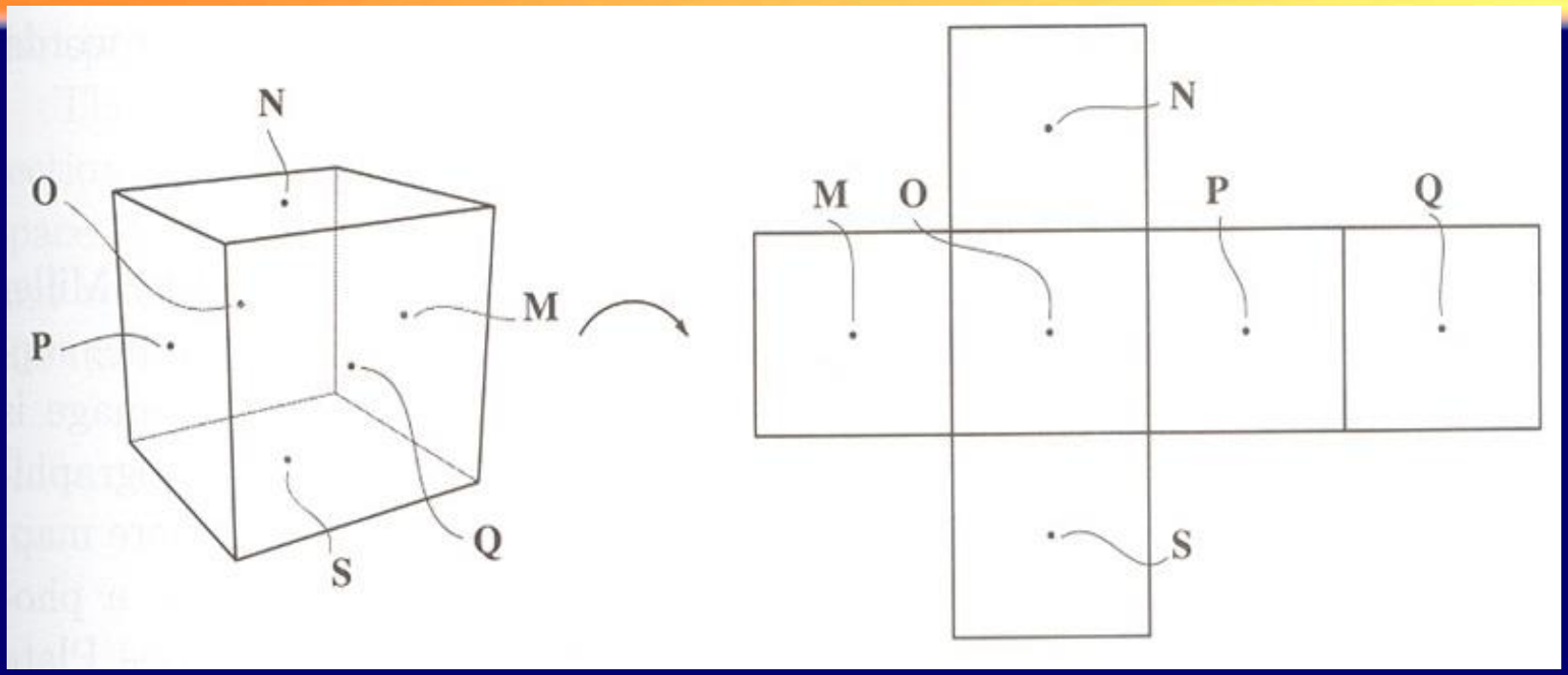
- 立方体环境映照由Greene于1986年提出，该方法速度快、适用性强，是图形硬件中最受欢迎的方法。
- 立方体环境映照通过把摄像机置于立方体的中心，然后把环境投影到立方体的面上。立方体上的图像作为环境图。
- 在实用中，场景需要绘制6次（立方体的每个面一次），摄像机置于立方体的中心，以90度的视角看立方体的面。

Cube Mapping

- 6 planar textures, sides of cube
  - point camera outwards to 6 faces
    - use largest magnitude of vector to pick face
    - other two coordinates for (s,t) texel location

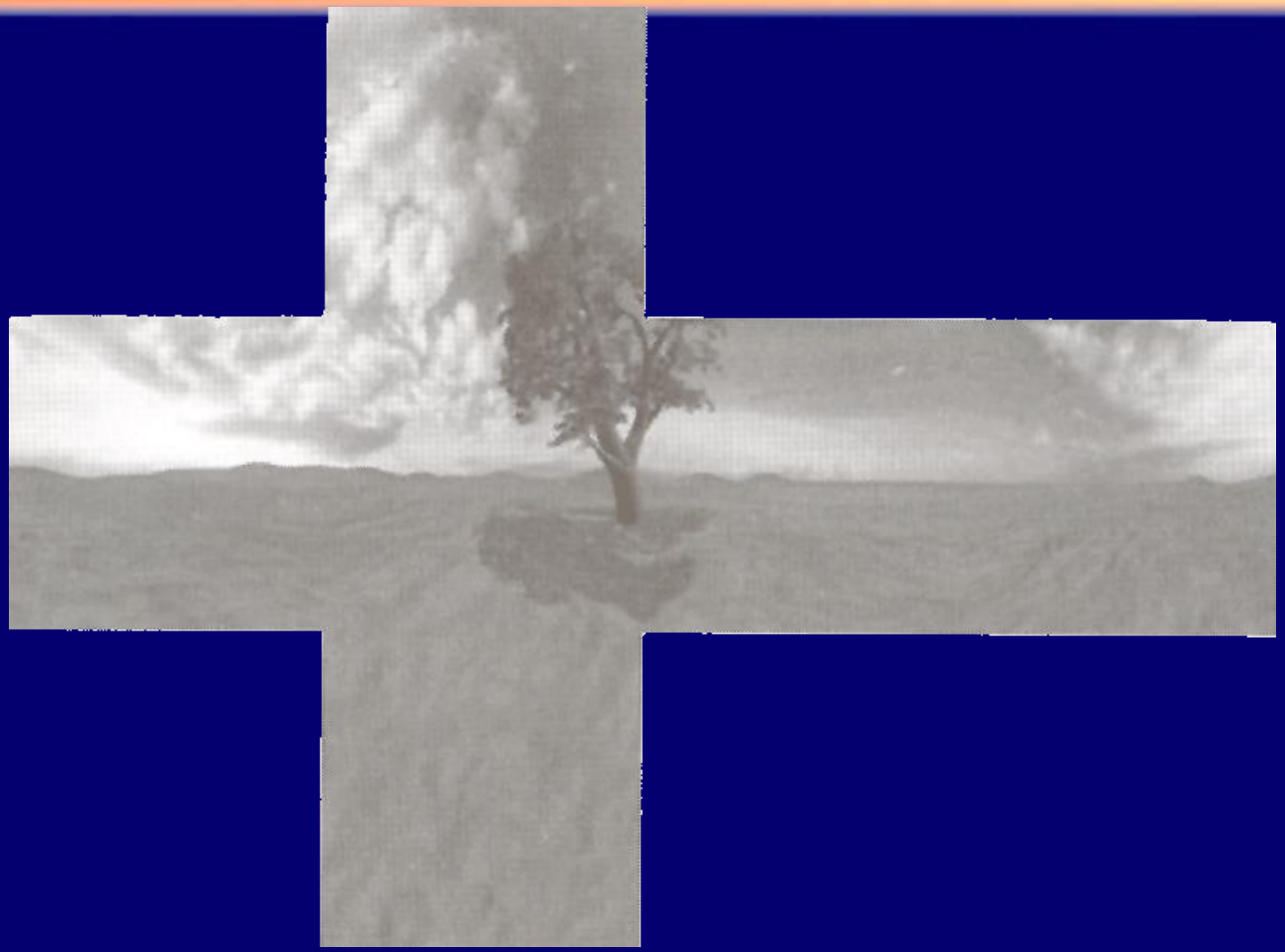


Greene环境映照示意图



左边的立方体展开为右边的环境图

一幅典型的环境图



例子

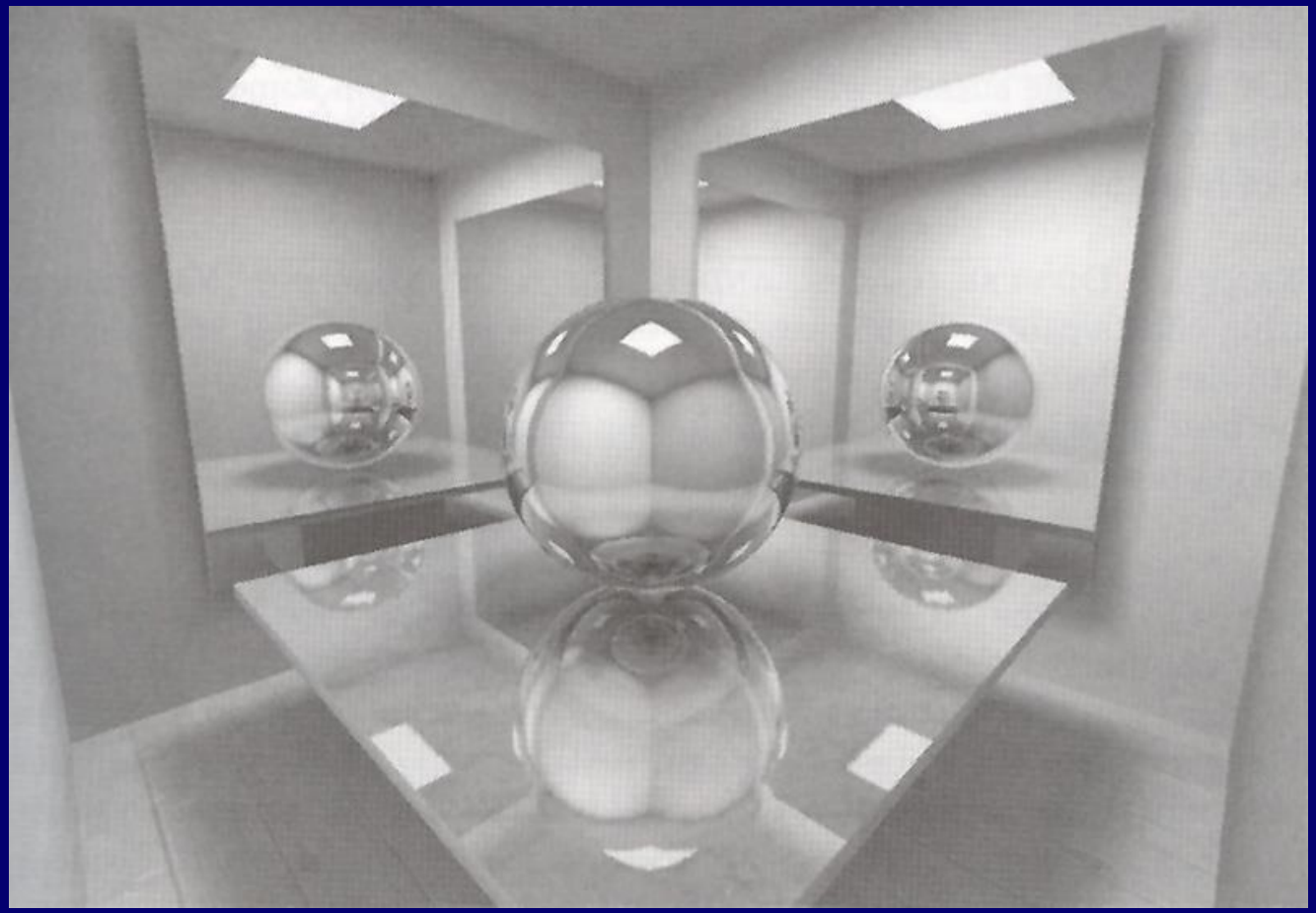
Xbox游戏的一幅静态图像，*Project Gotham Racing*。为了模拟车身上的动态反射效果，在车位置处逐帧建立一立方体环境映照。

采用环境映照的光照明
(Lighting Using Environment Mapping)

- 环境映照的一个重要用途为生成镜面反射和折射。我们前面讲过，因为Gouraud Shading只在顶点处计算光照效果，所以会丢失高光。环境映照可以通过纹理上的光照来克服该问题。通过该方法，我们可以用固定的花费模拟任何数量光源的高光(on per-pixel basis)。而且光源也可以不一定是点，可以是有大小和形状的。

Per-pixel的镜面高光　　光照环境映照　　整个周围场景的环境映照。

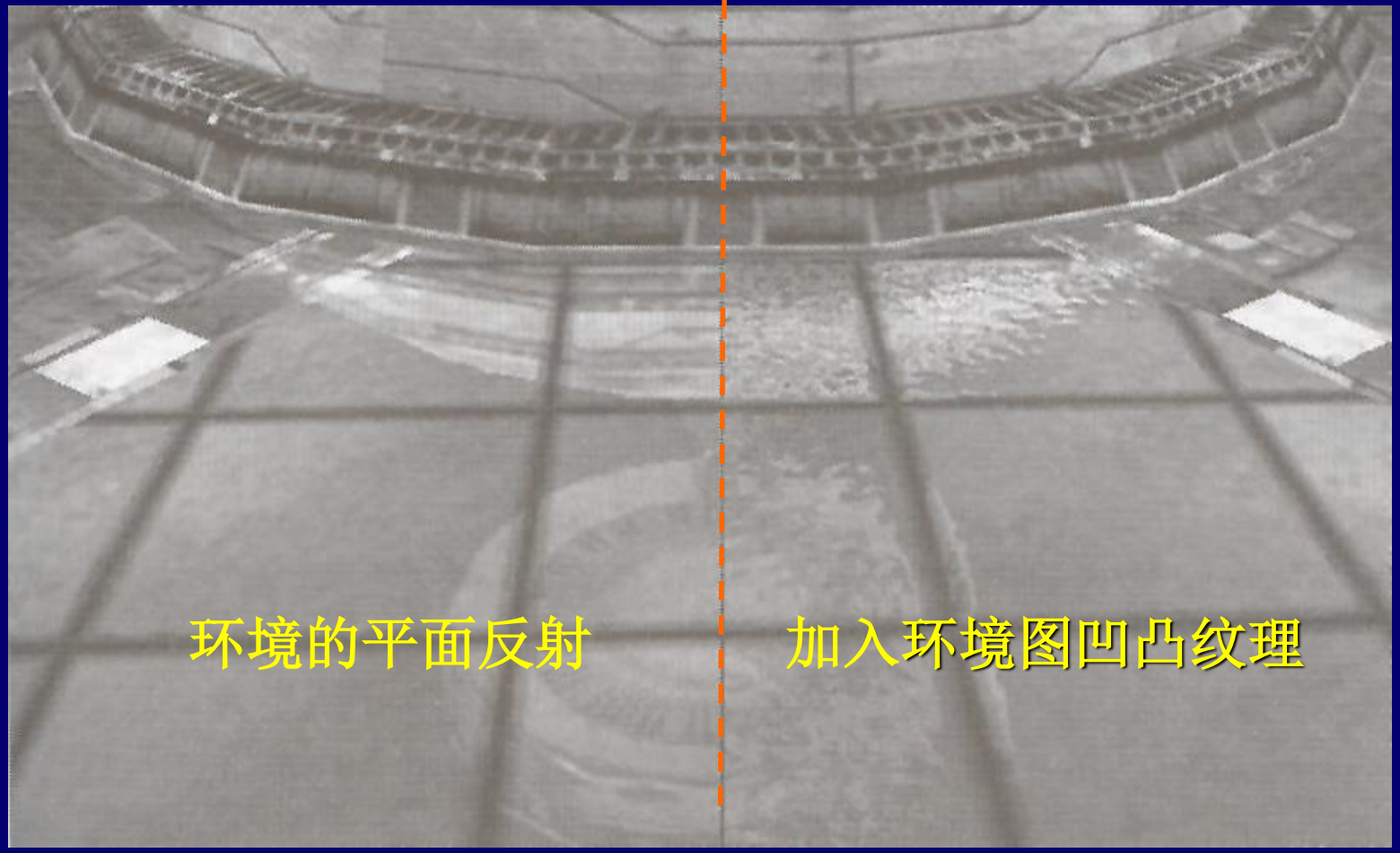
用环境映照实现的递归反射

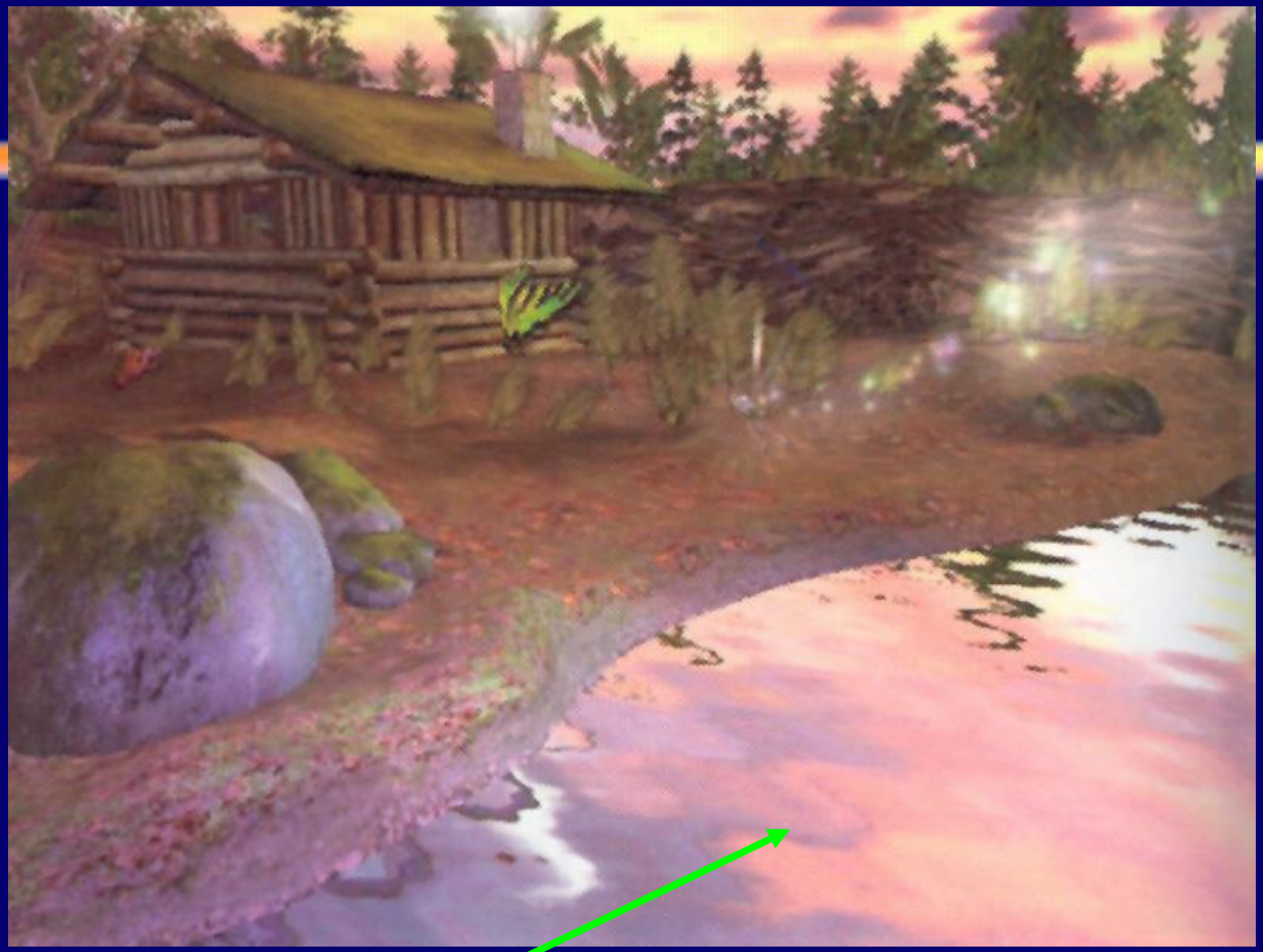


环境映照技术的实现过程

- 球面环境映照
  - 设置球面环境纹理
  - 设置球面环境纹理坐标生成的方式
  - 打开球面环境纹理坐标生成的方式
  - 正确设置物体表面的法向，绘制物体
- 立方体环境映照
  - 六张图片拼成整个空间

其他效果





水面反射天空的效果采用环境图凹凸纹理技术

Normal mapping和cubic environment mapping相结合