CS380: Computer Graphics
Texture Mapping

Sung-Eui Yoon
(윤성의)

Course URL:
http://sglab.kaist.ac.kr/~sungeui/CG

Outline

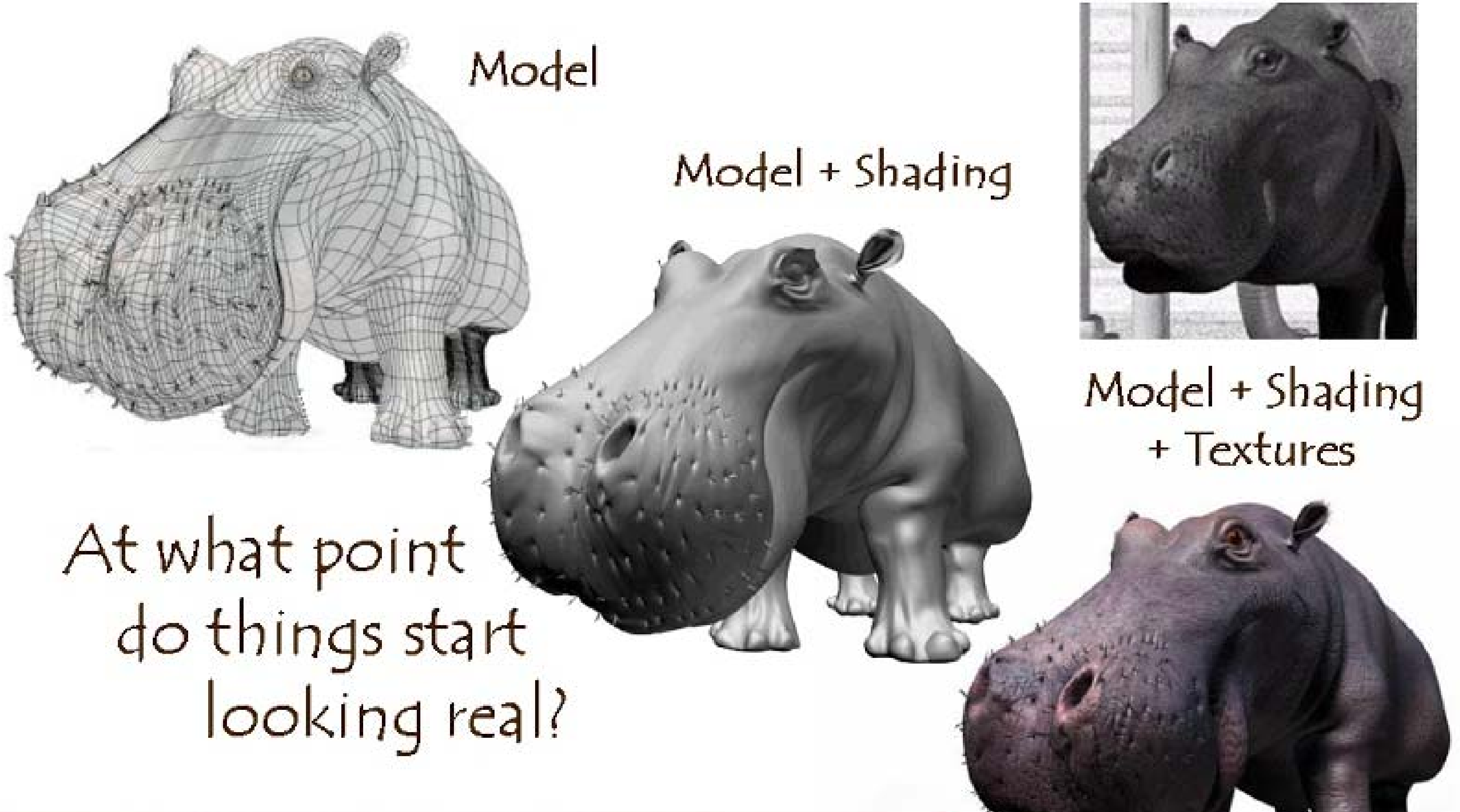
- Texture mapping overview
- Perspective-correct interpolation
- Texture filtering

Texture Mapping

- **Requires lots of geometry to fully represent complex shapes of models**
- **Add details with image representations**

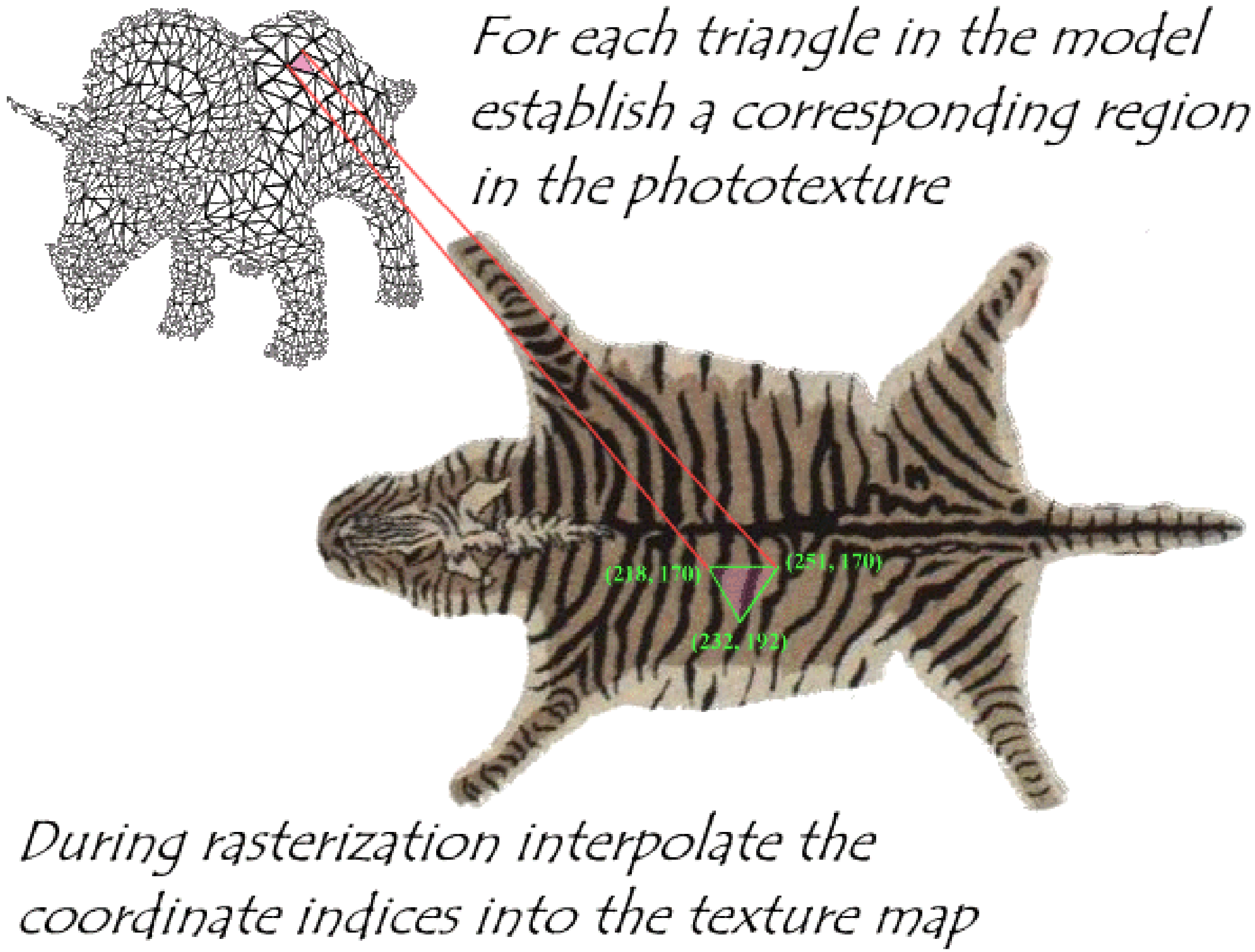
Excerpted from MIT EECS 6.837,
Durand and Cutler

The Quest for Visual Realism



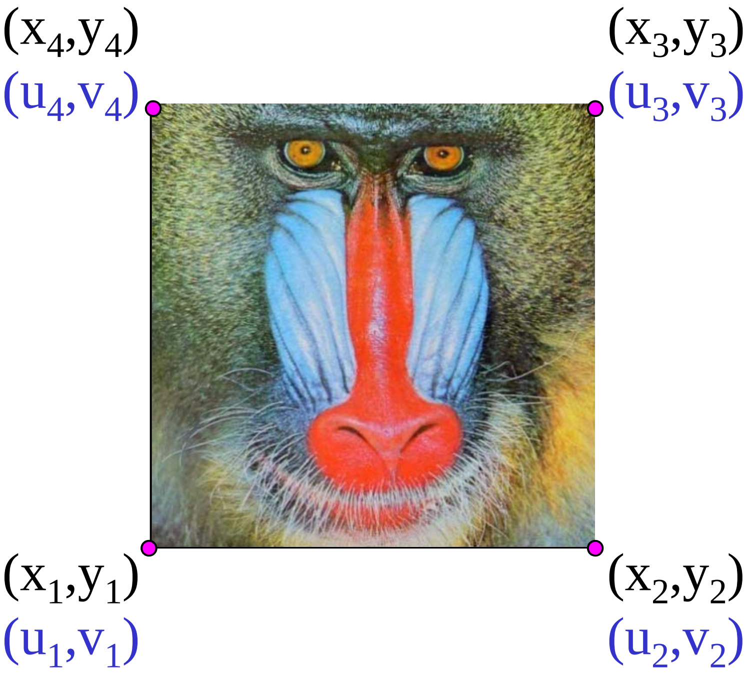
For more info on the computer artwork of Jeremy Birn
see http://www.3drender.com/jbirn/productions.html

Photo-Textures



Excerpted from MIT EECS 6.837, Durand and Cutler

Texture Maps in OpenGL



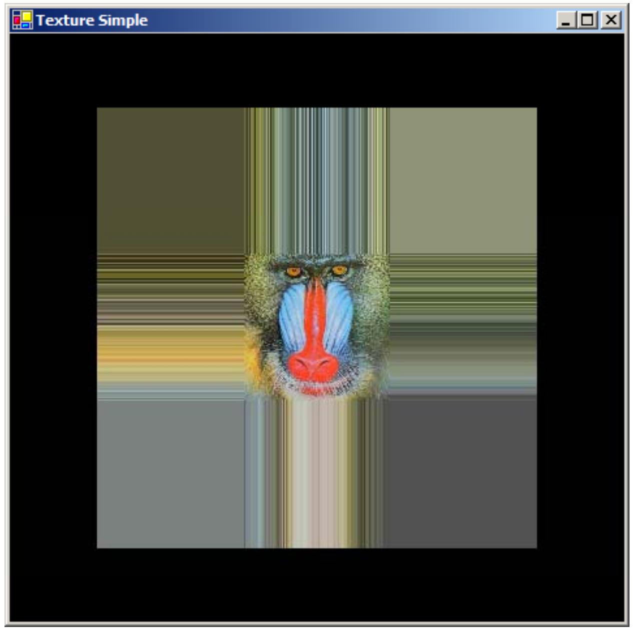
- Specify normalized texture coordinates at each of the vertices (u, v)
- Texel indices (s,t) = (u, v) · (width, height)

```
glBindTexture(GL_TEXTURE_2D, texID)
glBegin(GL_POLYGON)
  glTexCoord2d(0,1); glVertex2d(-1,-1);
  glTexCoord2d(1,1); glVertex2d( 1,-1);
  glTexCoord2d(1,0); glVertex2d( 1, 1);
  glTexCoord2d(0,0); glVertex2d(-1, 1);
glEnd()
```

Wrapping

- **The behavior of texture coordinates outside of the range [0,1) is determined by the texture wrap options.**

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, wrap_mode )
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, wrap_mode )
```

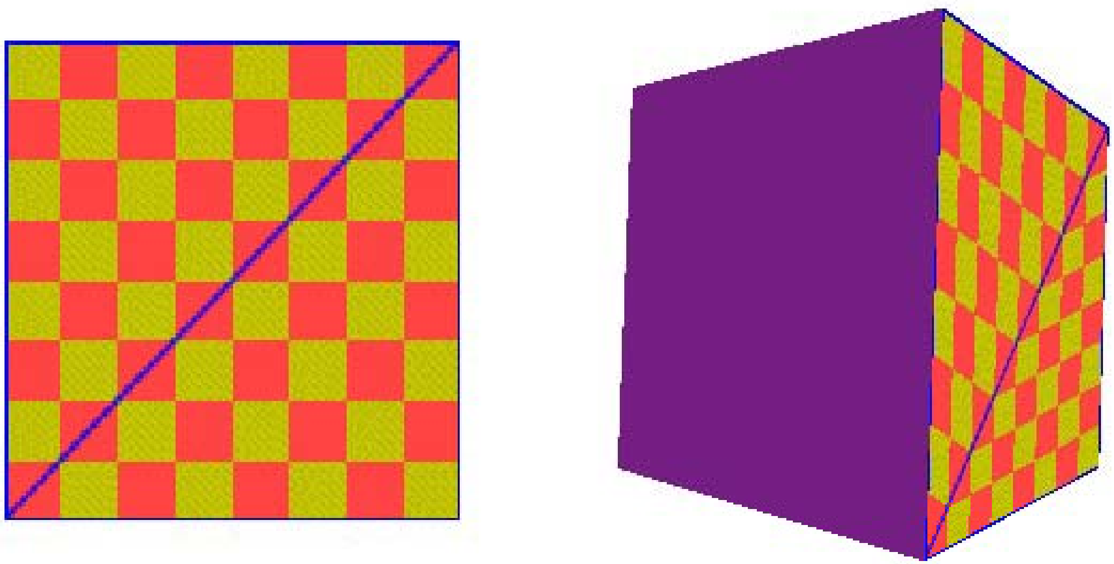


GL_CLAMP

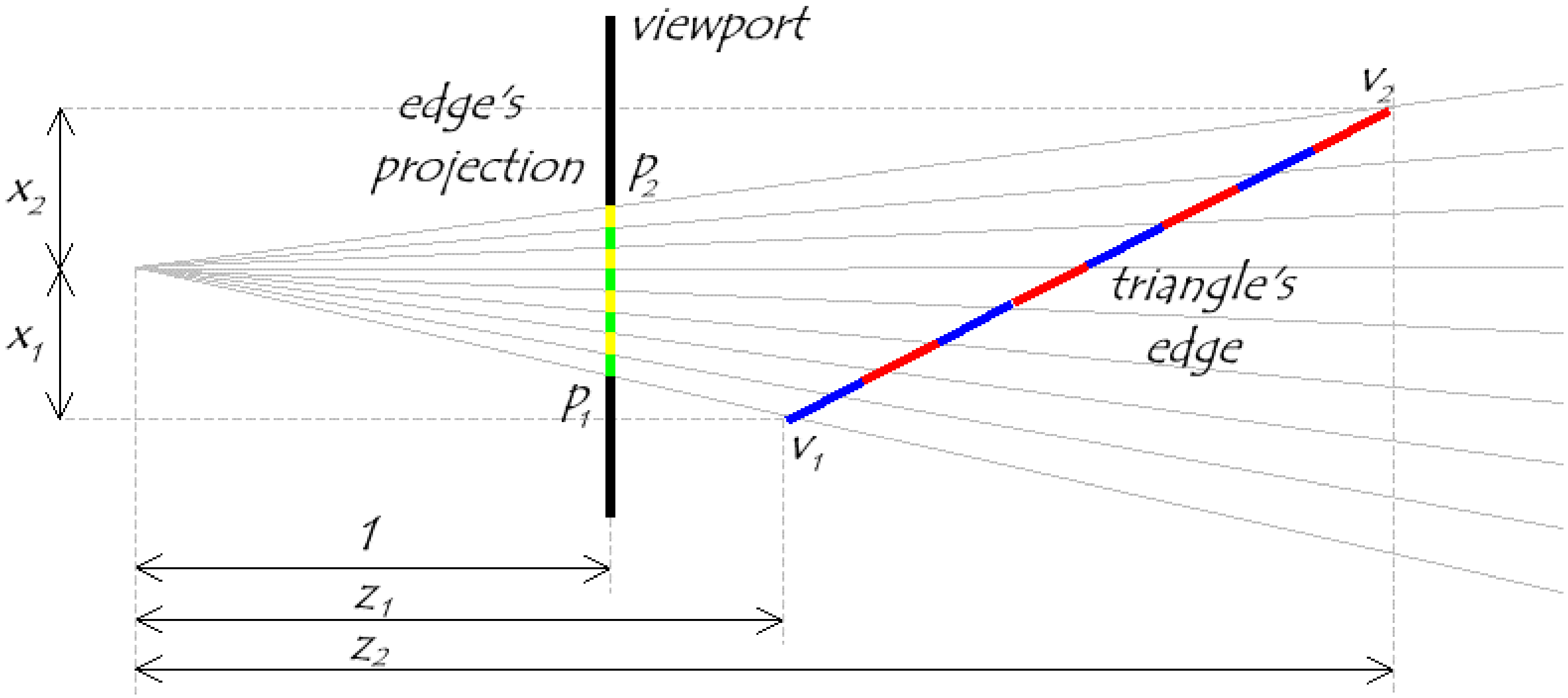
GL_REPEAT

Linear Interpolation of Texture Coordinates

- Simple linear interpolation of u and v over a triangle in a screen space leads to unexpected results
  - Distorted when the triangle's vertices do not have the same depth

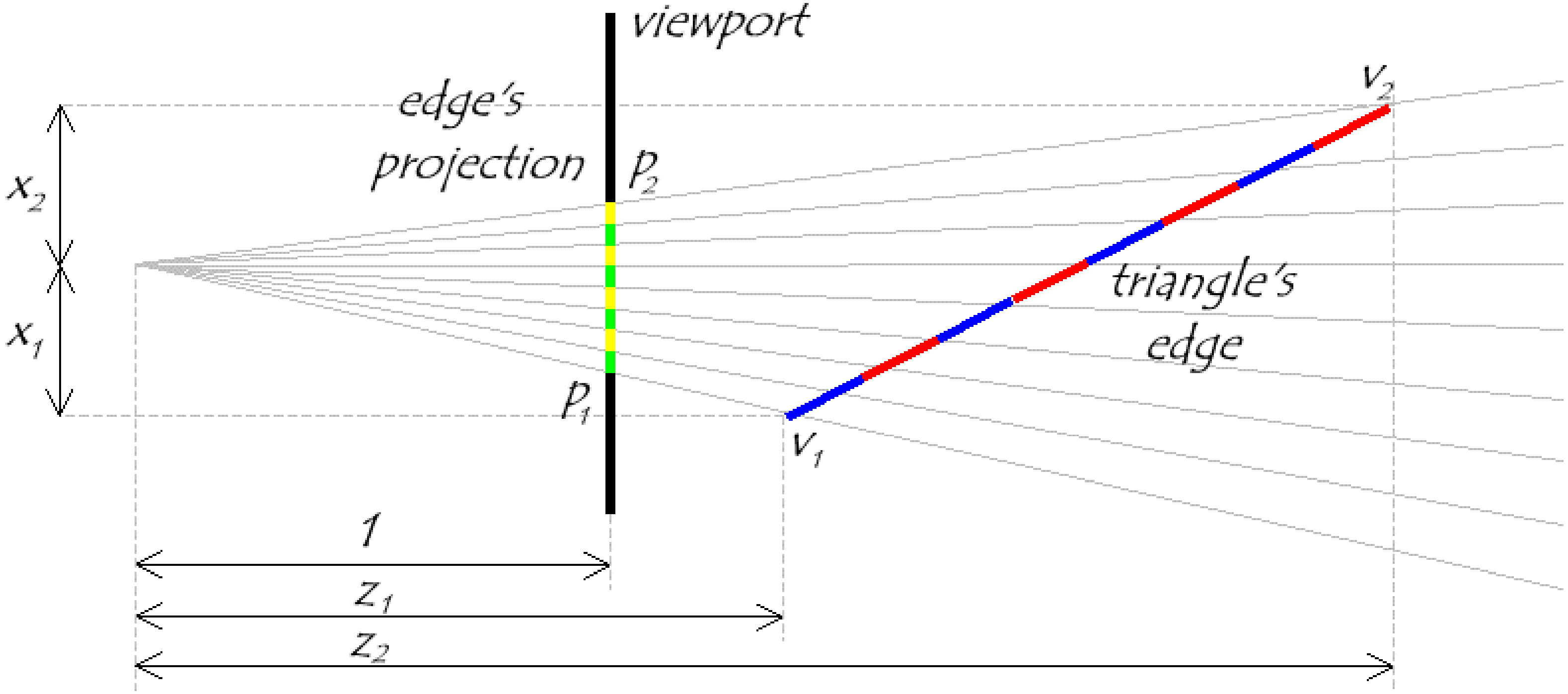


Linear Interpolation of Texture Coordinates



- **Uniform steps along the edge projection in screen space do not correspond to uniform steps along the actual edge in eye space**

Linear Interpolation of Texture Coordinates



screen space

$$p(\tau_s) = p_1 + \tau_s (p_2 - p_1)$$

$$= {}^{x_1}\!/_{z_1} + \tau_s \left({}^{x_2}\!/_{z_2} - {}^{x_1}\!/_{z_1}\right)$$

world space

$$\dot{v}(\tau_e) = \dot{v}_1 + \tau_e (\dot{v}_2 - \dot{v}_1)$$

$$p(\dot{v}(\tau_e)) = \frac{x(\tau_e)}{z(\tau_e)} = \frac{x_1 + \tau_e (x_2 - x_1)}{z_1 + \tau_e (z_2 - z_1)}$$

Correcting the Interpolation

- **We want to interpolate in world space, but in terms of our screen space τ_s**
  - **So we solve** $p(\tau_s) = p(\bar{v}(\tau_e))$ **for τ_e in terms of τ_s:**

$$p(\tau_s) = x_1/z_1 + \tau_s\left(x_2/z_2 - x_1/z_1\right) = \frac{x_1 + \tau_e(x_2 - x_1)}{z_1 + \tau_e(z_2 - z_1)} = p(\bar{v}(\tau_e))$$

$$\tau_e = \frac{\tau_s z_1}{z_2 + \tau_s(z_1 - z_2)}$$

- **In screen space, we don't have z_1 and z_2. But before the perspective divide we do have $w_1 = z_1$ and $w_2 = z_2$:**

$$\tau_e = \frac{\tau_s w_1}{w_2 + \tau_s(w_1 - w_2)}$$

Correcting the Interpolation

- **Plug this value of τ_e into the equation to linearly interpolate parameters like (u,v) in eye space:**

$$u(\tau_e) = u_1 + \tau_e (u_2 - u_1)$$

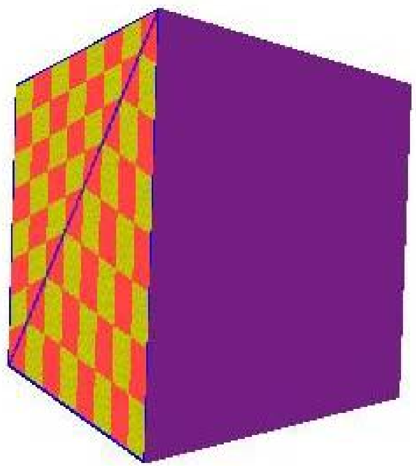
$$u(\tau_s) = u_1 + \frac{\tau_s w_1}{w_2 + \tau_s (w_1 - w_2)}(u_2 - u_1) = \frac{u_1 w_2 + \tau_s (u_2 w_1 - u_1 w_2)}{w_2 + \tau_s (w_1 - w_2)}$$

$$u(\tau_s) = \frac{u_1/w_1 + \tau_s (u_2/w_2 - u_1/w_1)}{1/w_1 + \tau_s (1/w_1 - 1/w_2)}$$

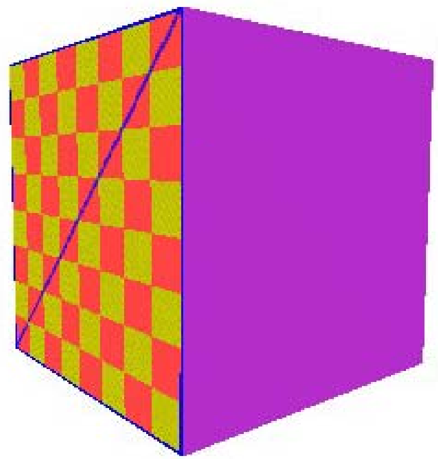
- **Linearly interpolate the numerator and denominator separately and do the divide once per pixel**

Perspective-Correct Interpolation

- **This method of interpolation is called perspective-correct interpolation**
  - **Actually it is simply correct interpolation**
  - **Not all 3D graphics APIs implement perspective-correct interpolation**



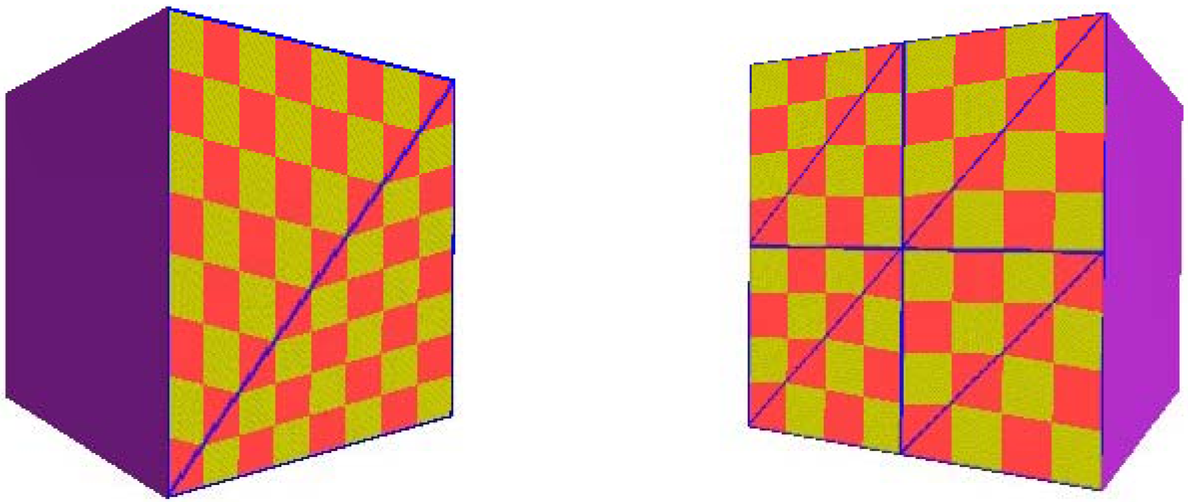
Linear interpolation



Perspective-correct interpolation

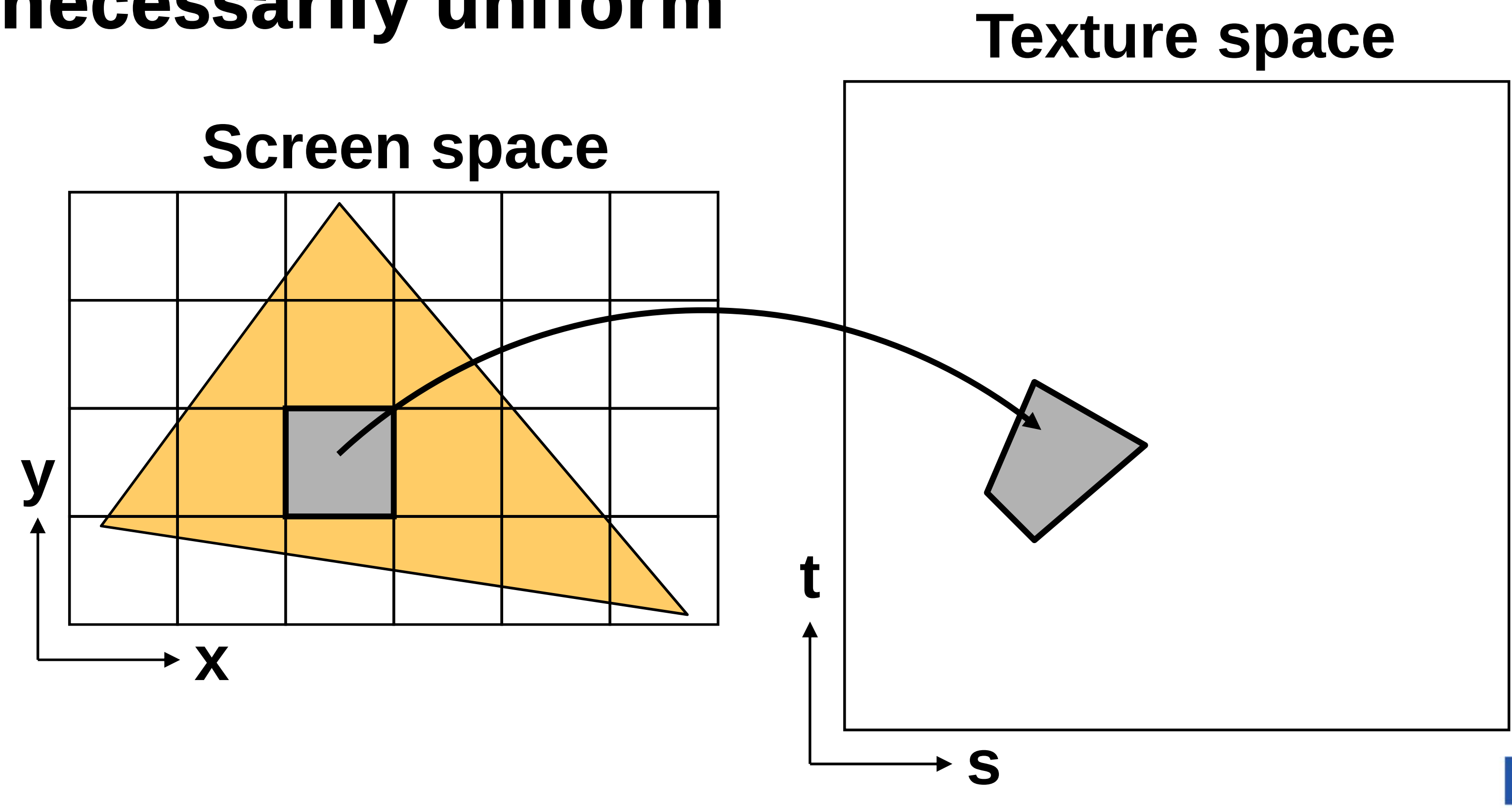
Dealing with Incorrect-Interpolation

- The perceived artifacts of non-perspective correct interpolation can be ameliorated by subdividing the texture-mapped triangles into smaller triangles
  - Why does this work?
- Screen-space interpolation of projected parameters is inherently flawed



Sampling Texture Maps

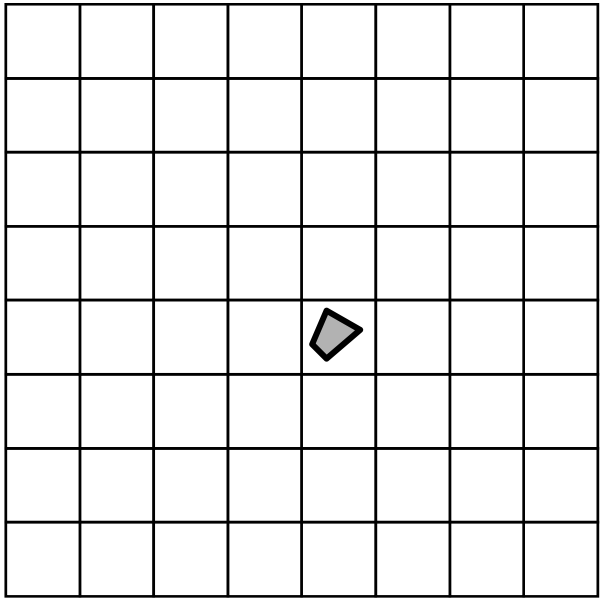
- The uniform sampling pattern in screen space cooresponds to some sampling pattern in texture space that is not necessarily uniform



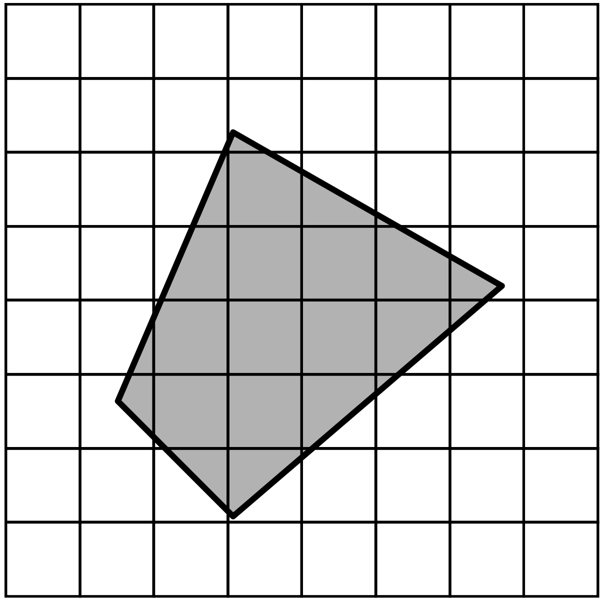
KAIST

Sampling Density Mismatch

- **Sampling density in texture space rarely matches the sample density of the texture itself**

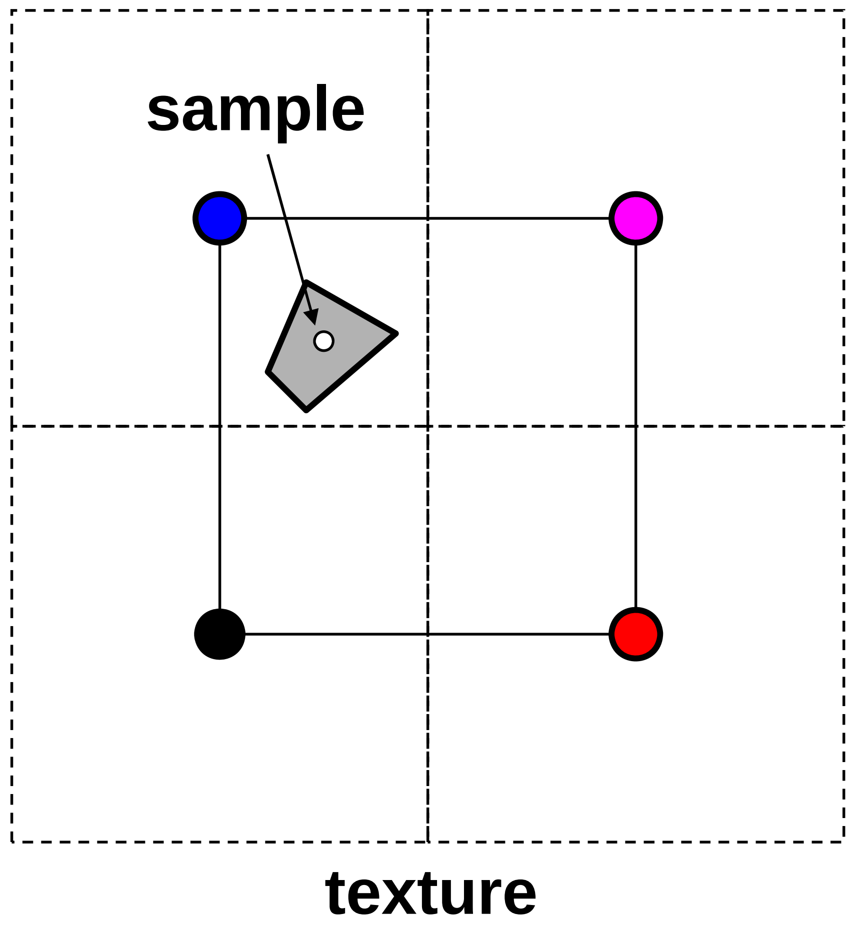


Oversampling (Magnification)



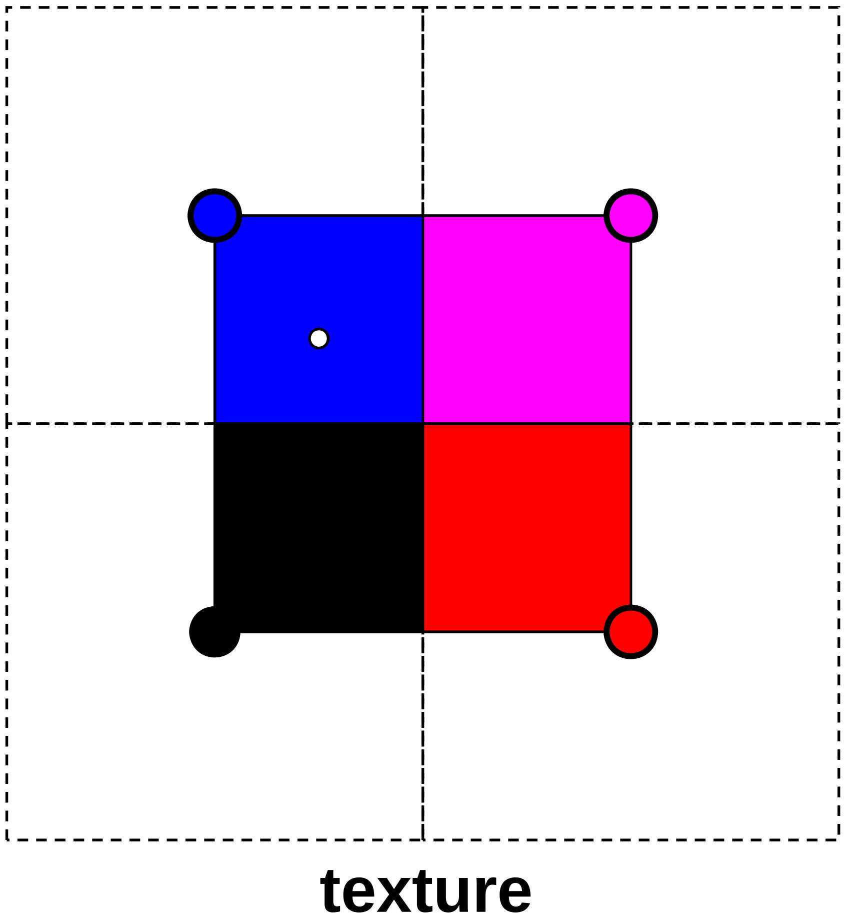
Undersampling (Minification)

Handling Oversampling



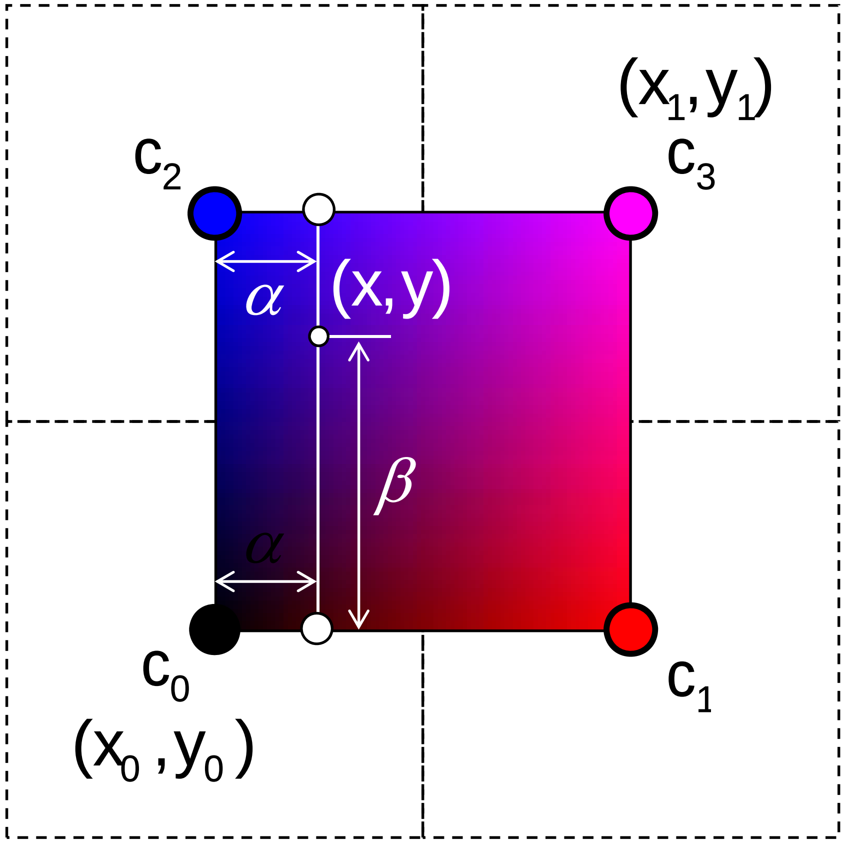
- **How do we compute the color to assign to this sample?**

Handling Oversampling



- How do we compute the color to assign to this sample?
- Nearest neighbor – take the color of the closest texel

Handling Oversampling



texture

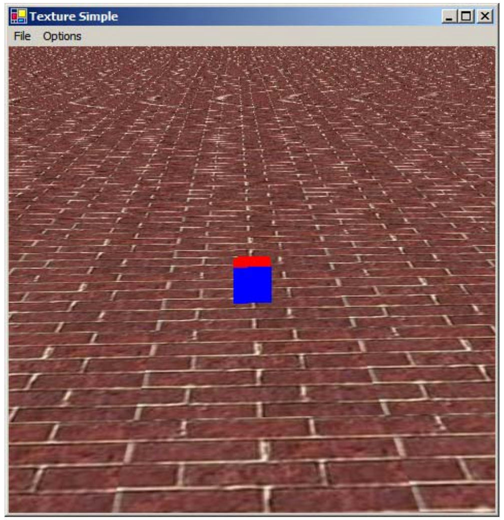
- **How do we compute the color to assign to this sample?**
- **Nearest neighbor – take the color of the closest texel**
- **Bilinear interpolation**

$$\alpha = \frac{x - x_0}{x_1 - x_0} \qquad \beta = \frac{y - y_0}{y_1 - y_0}$$

$$c = ((1-\alpha)c_0 + \alpha c_1)(1-\beta) + ((1-\alpha)c_2 + \alpha c_3)\beta$$

KAIST

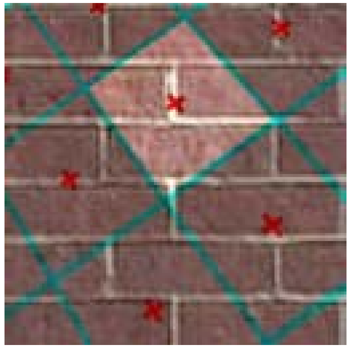
Undersampling



- Details in the texture tend to pop (disappear and reappear)
  - Mortar (white substances) in the brick
- High-frequency details lead to strange patterns
  - Aliasing

Spatial Filtering

- To avoid aliasing we need to prefilter the texture to remove high frequencies
  - Prefiltering is essentially a spatial integration over the texture
  - Integrating on the fly is expensive: perform integration in a pre-process



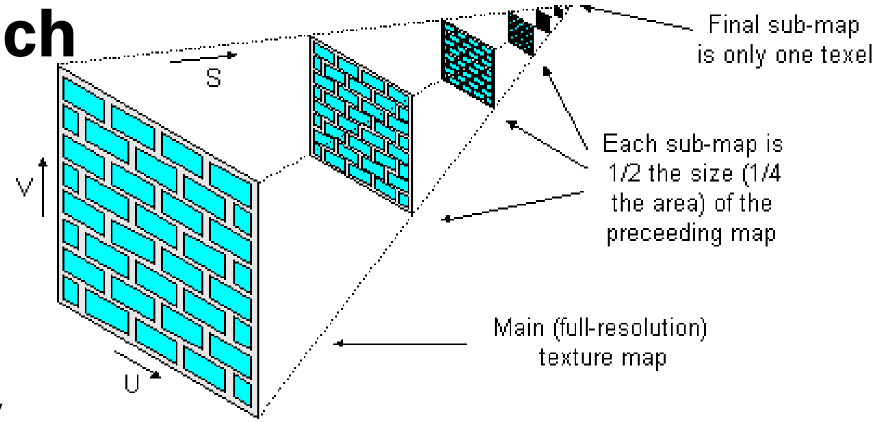
Samples and their extents

Proper filtering removes aliasing

KAIST

MIP Mapping

- **MIP is an acronym for the Latin phrase *multium in parvo*, which means "many in one place"**
  - **Constructs an *image pyramid***
  - **Each level is a prefiltered version of the level below resampled at half the frequency**

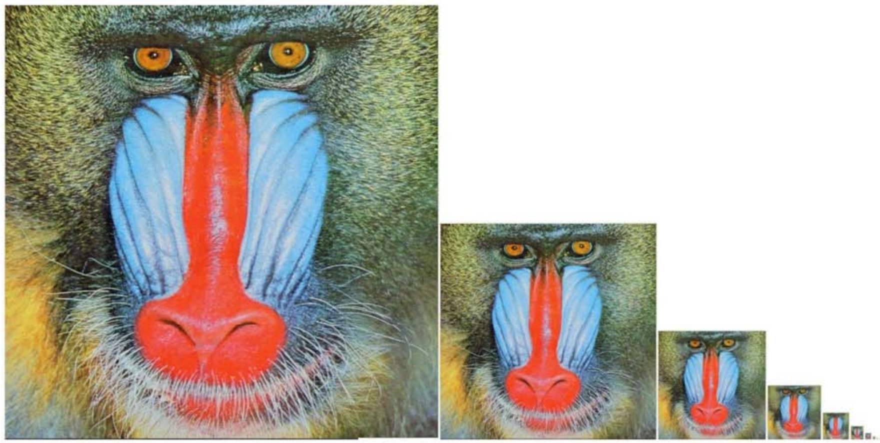


- **While rasterizing use the level with the sampling rate closest to the desired sampling rate**
  - **Can also interpolate between pyramid levels**
- **How much storage overhead is required?**

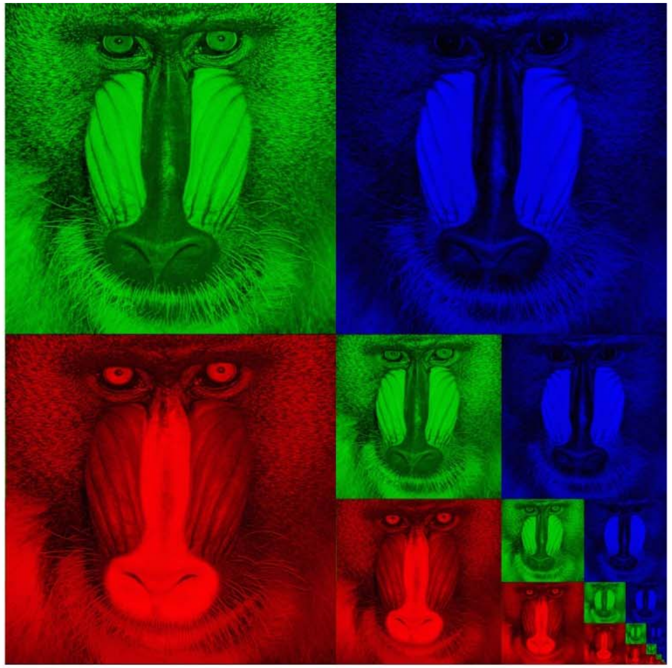
$$\text{mip map size} = \sum_{i=0}^{\infty} \left(\frac{1}{4}\right)^i = \frac{1}{1-\frac{1}{4}} = \frac{4}{3}$$

Storing MIP Maps

- **One convenient method of storing a MIP map is shown below**
  - **It also nicely illustrates the 1/3 overhead of maintaining the MIP map**



10-level mip map



Memory format of a mip map

Finding the MIP Level

- Use the projection of a pixel in screen into texture space to figure out which level to use

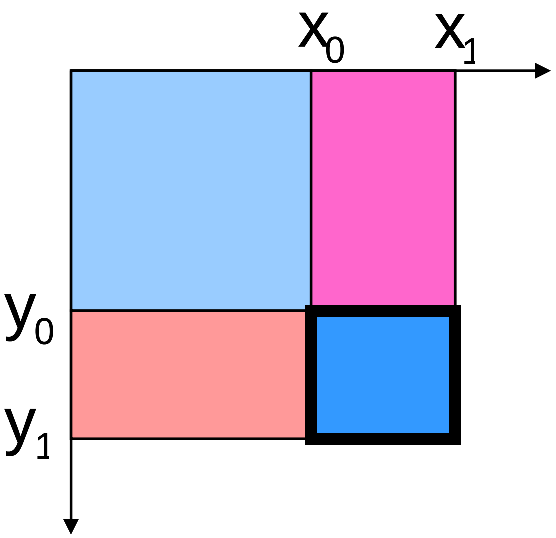
Summed-Area Tables

- **Another way performing the prefiltering integration on the fly**
- **Each entry in the summed area table is the sum of all entries above and to the left:**

1	6	8	3
0	0	3	7
4	7	8	8
5	0	9	9

→

1	7	15	18
1	7	18	28
5	18	37	55
10	23	51	78



What is the sum of the highlighted region?

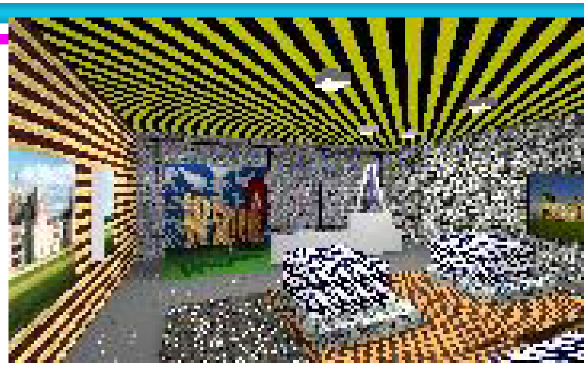
$$T(x_1,y_1) - T(x_1,y_0) - T(x_0,y_1) + T(x_0,y_0)$$

Divide out area $(y_1 - y_0)(x_1 - x_0)$

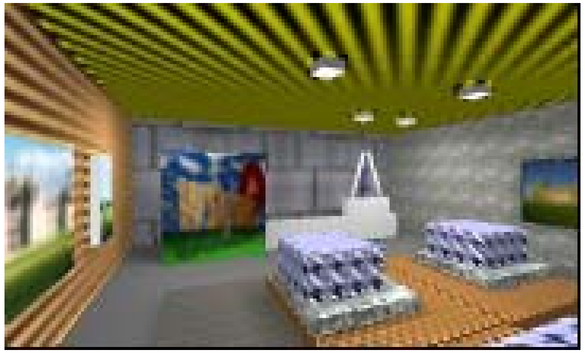
Summed-Area Tables

- **How much storage does a summed-area table require?**
- **Does it require more or less work per pixel than a MIP map?**

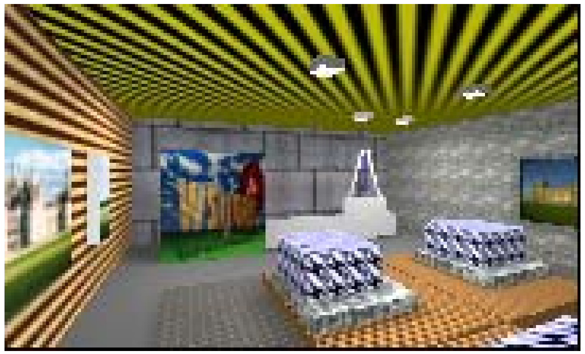
No Filtering



MIP mapping



Summed-Area Table



KAIST

Texture Filtering in OpenGL

- **Automatic creation**

```
gluBuild2DMipmaps(GL_TEXTURE_2D, GL_RGBA, width, height,
                  GL_RGBA, GL_UNSIGNED_BYTE, data)
```

- **Filtering**

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, filter )
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, filter )
```

where filter is:

```
GL_NEAREST
GL_LINEAR

GL_LINEAR_MIPMAP_LINEAR
GL_NEAREST_MIPMAP_NEAREST
GL_NEAREST_MIPMAP_LINEAR
GL_LINEAR_MIPMAP_NEAREST
```

inter-level **intra-level**

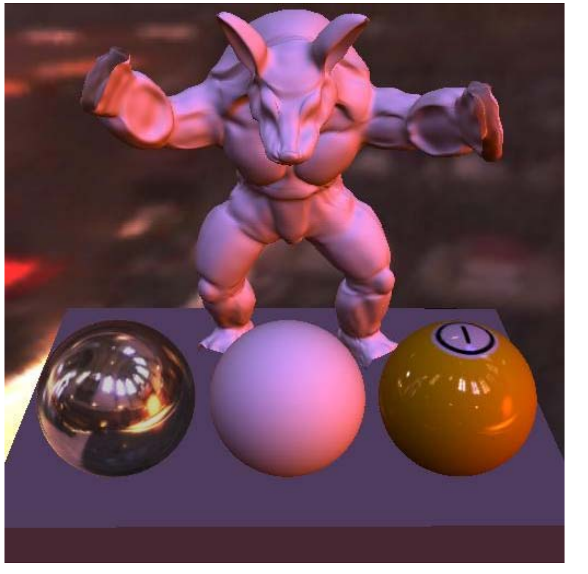
Uses of Texture Maps

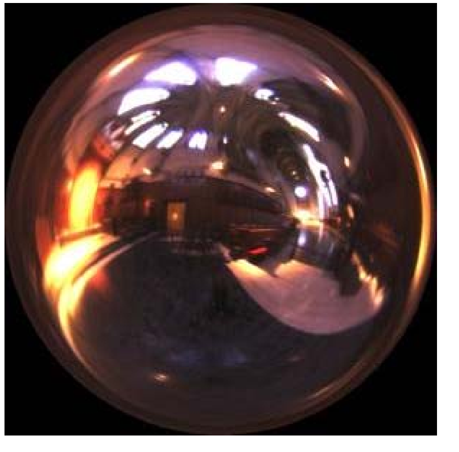
- Texture maps are used to add complexity to a scene
  - Easier to paint or capture an image than geometry
- Model light
- Model geometry, etc

KAIST

Modeling Lighting

- Light maps
  - Supply the lighting directly
  - Good for static environments
- Projective textures
  - Can be used to simulate a spot light
  - Shadow maps
- Environment maps
  - A representation of the scene around an object
  - Good for reflection





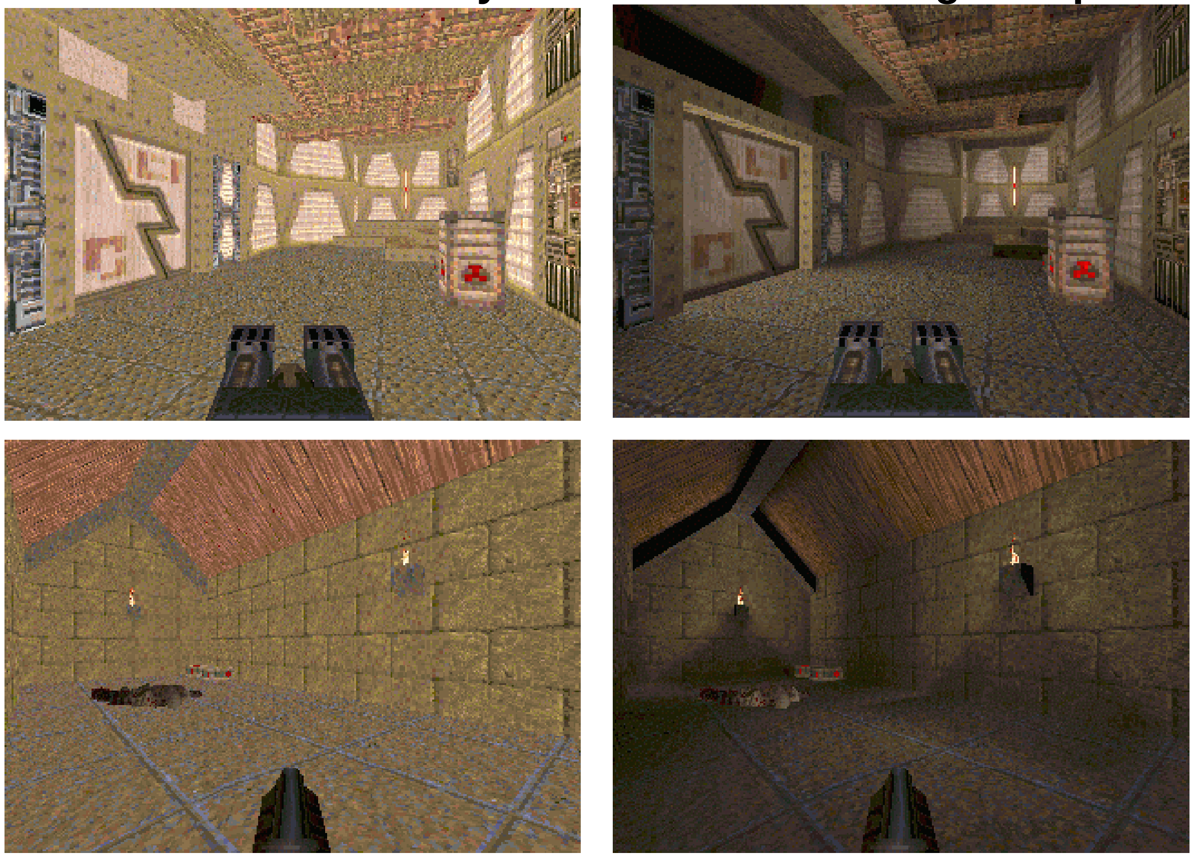
KAIST

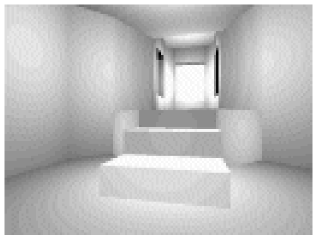
Light Maps in Quake

- **Light maps are used to store pre-computed illumination**

	Texture Maps	Light Maps
Data	RGB	Intensity
Resolution	High	Low

Textures Only **Textures & Light Maps**

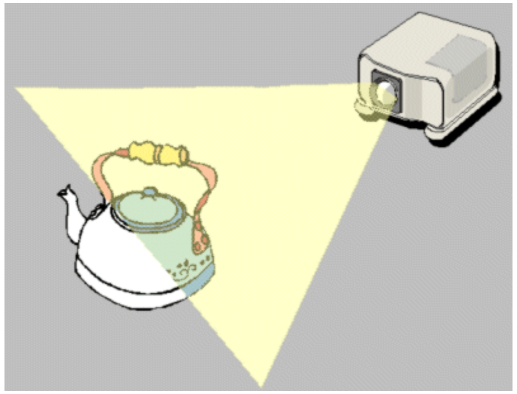


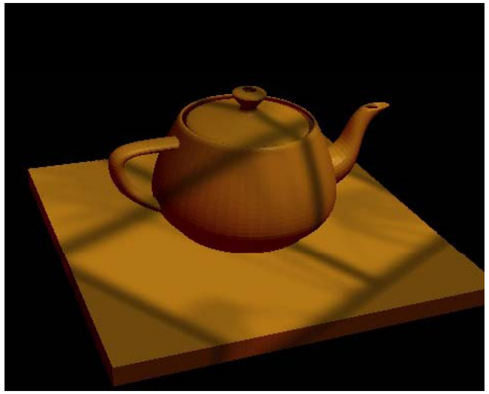


Light map image by Nick Chirkov

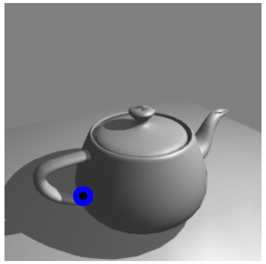
Projective Textures

- Treat the texture as a slide in a projector
  - A good model for shading variations due to illumination (cool spotlights)
- Projectors work like cameras in reverse
  - Camera: color of point in scene $\rightarrow$ color of corresponding pixel
  - Projector: color of pixel $\rightarrow$ color of corresponding point in the scene



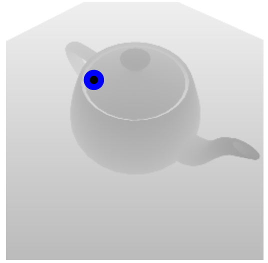


Shadow Maps

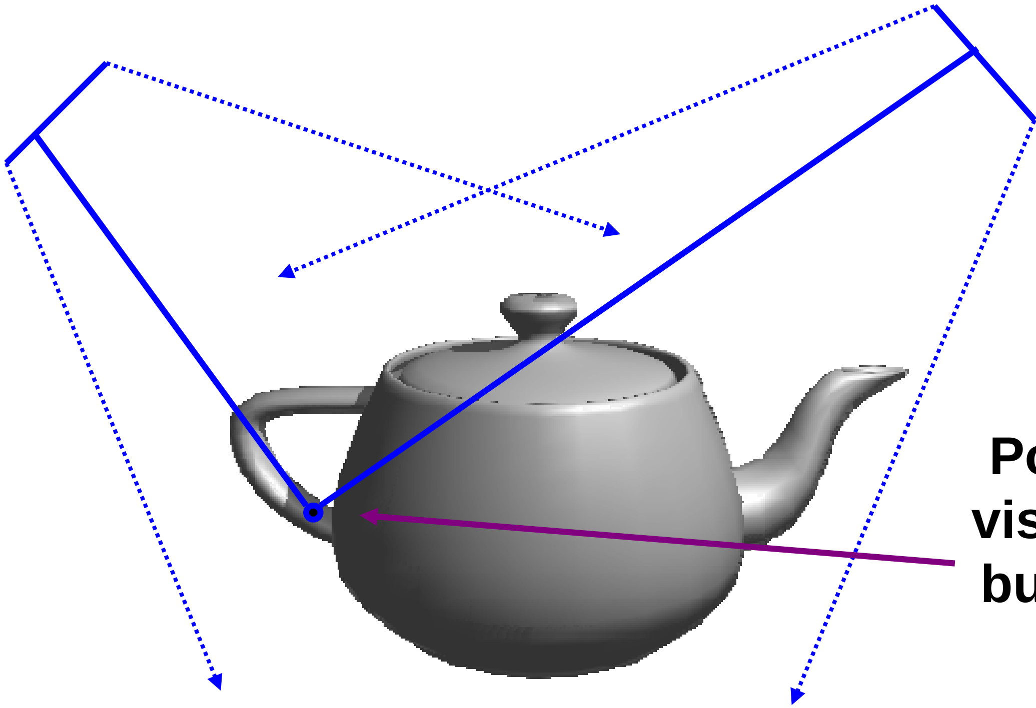


Eye

Use the depth map in the light view to determine if sample point is visible



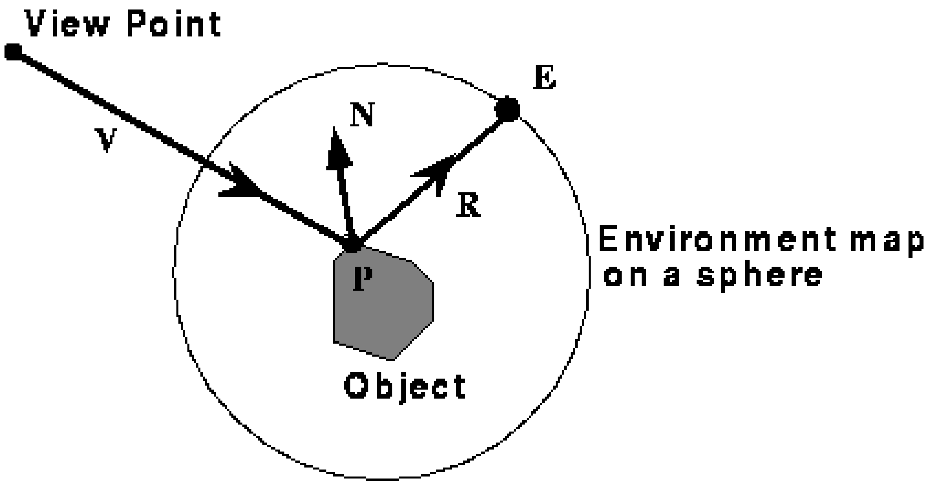
Light



Point in shadow visible to the eye, but not visible to the light

Environment Maps

- Simulate complex mirror-like objects
  - Use textures to capture environment of objects
  - Use surface normal to compute texture coordinates

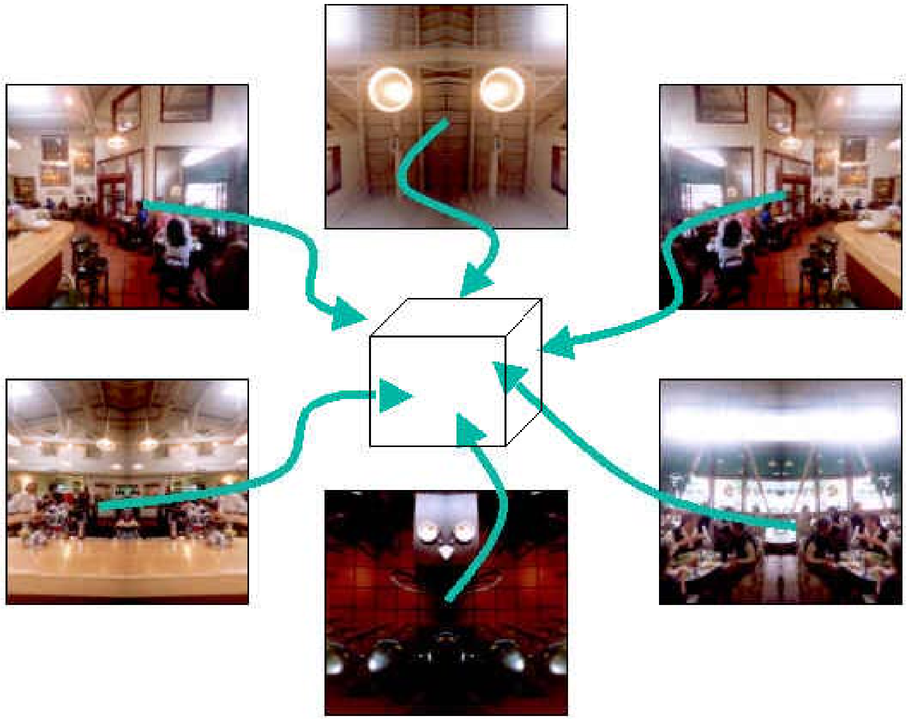


Environment Maps - Example

T1000 in Terminator 2 from Industrial Light and Magic

Cube Maps

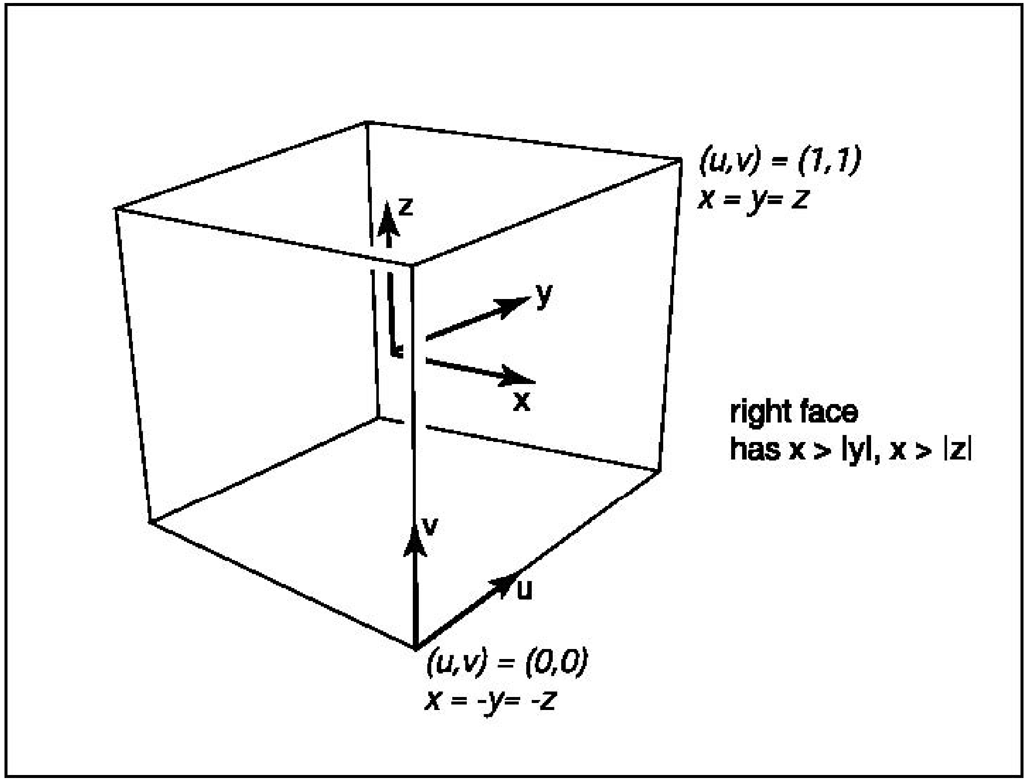
- Maps a viewing direction b and returns an RGB color
  - Use stored texture maps



KAIST

Cube Maps

- **Maps a viewing direction b and returns an RGB color**
  - **Assume b = (x, y, z),**



- Identify a face based on magnitude of x,y,z

-For the right face, compute texture coord. (u,v)

u = (y+x)/(2x)
v = (z+x)/(2x)

Environment Maps - Problems

- Expensive to update dynamically
- Not completely accurate

images from NVIDIA

Reflection of swimming pool is wrong

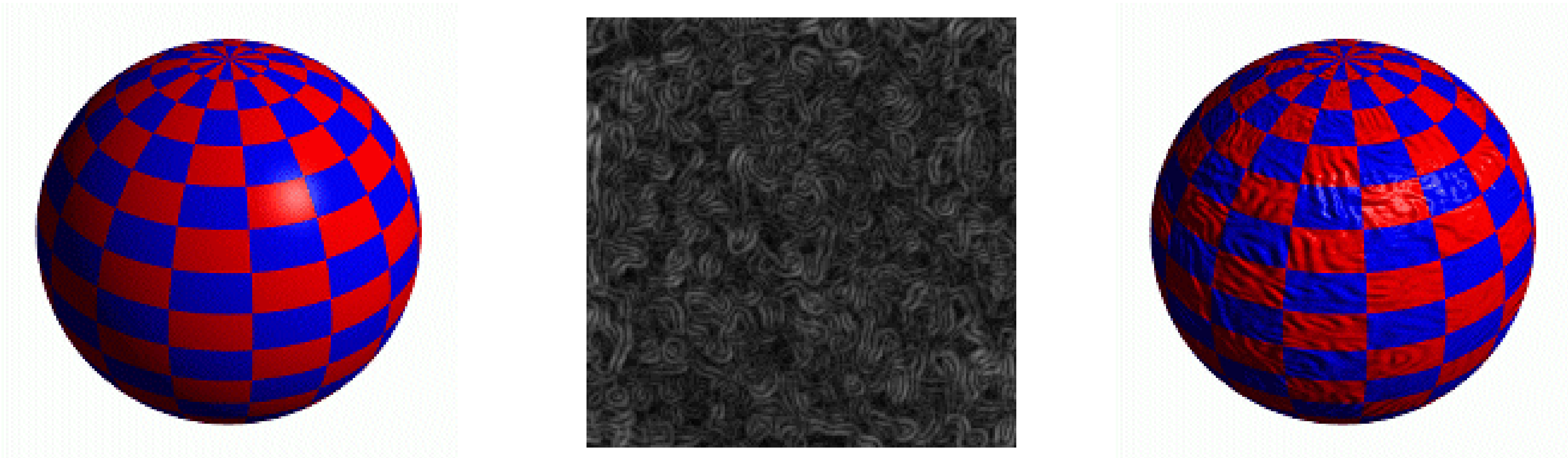
KAIST

Modeling Geometry

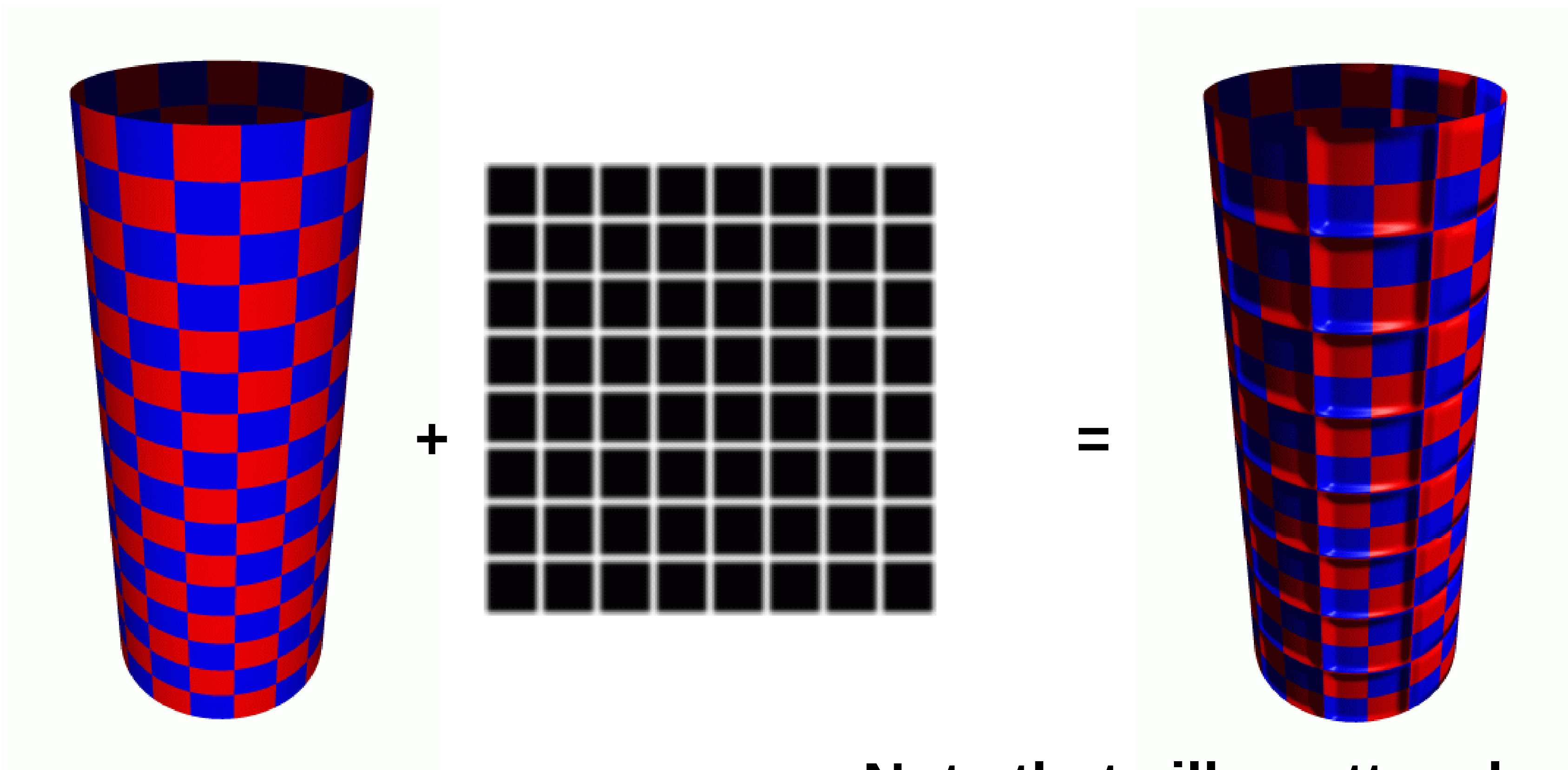
- Store complex surface details in a texture rather than modeling them explicitly
- Bump maps
  - Modify the existing normal
- Normal maps
  - Replace the existing normal
- Displacement maps
  - Modify the geometry
- Opacity maps and billboards
  - Knock-out portions of a polygon using the alpha channel

Bump Mapping

- Modifies the normal not the actual geometry
  - Texture treated as a heightfield
  - Partial derivatives used to change the normal
  - Causes surface to appear deformed by the heightfield



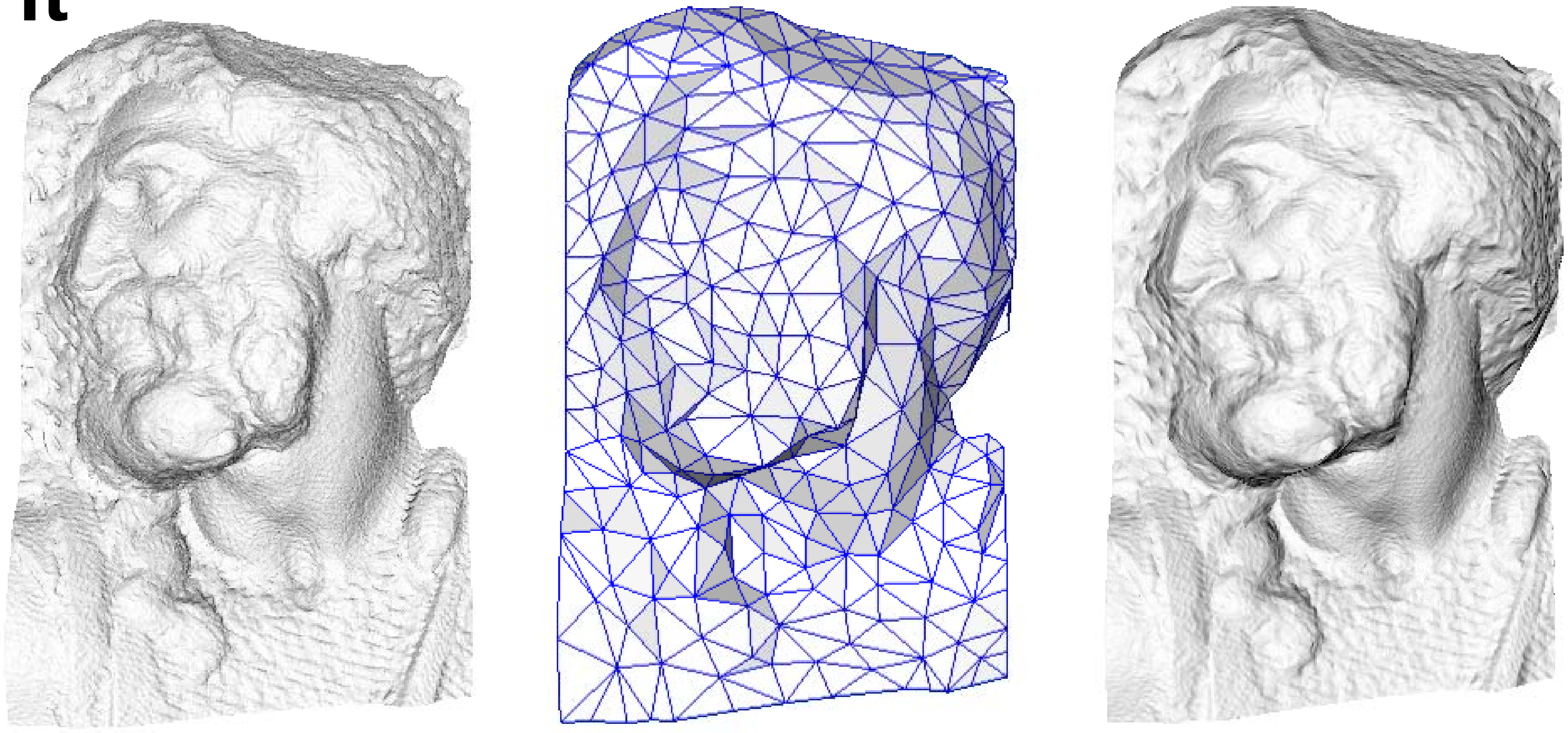
More Bump Map Examples



Note that silhouette edge of the object not affected!

Normal Mapping

- **Replaces the normal rather than tweaking it**



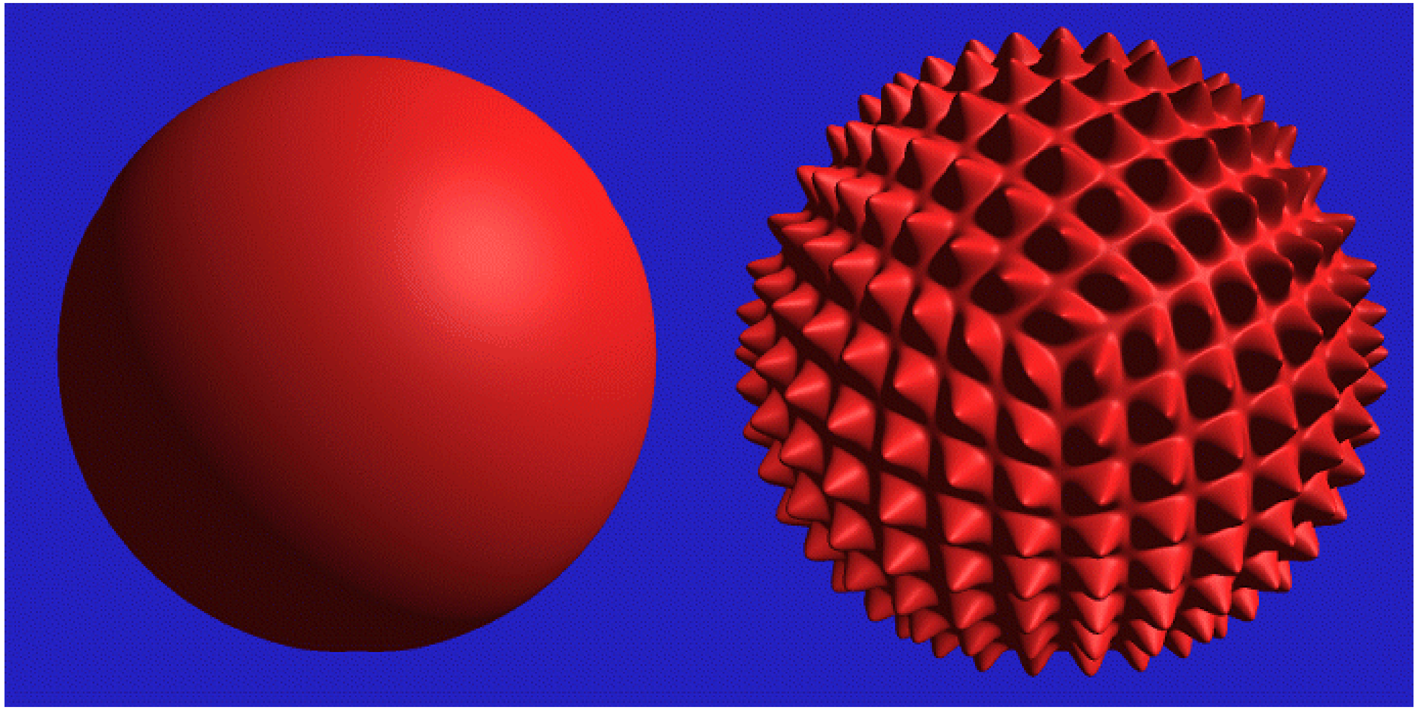
original mesh
4M triangles

simplified mesh
500 triangles

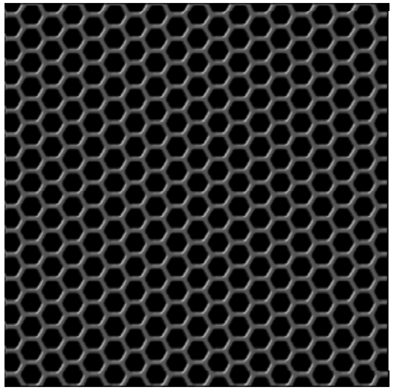
simplified mesh
and normal mapping
500 triangles

Displacement Mapping

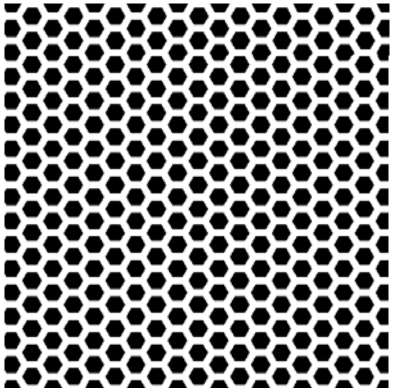
- **Texture maps can be used to actually move surface points**



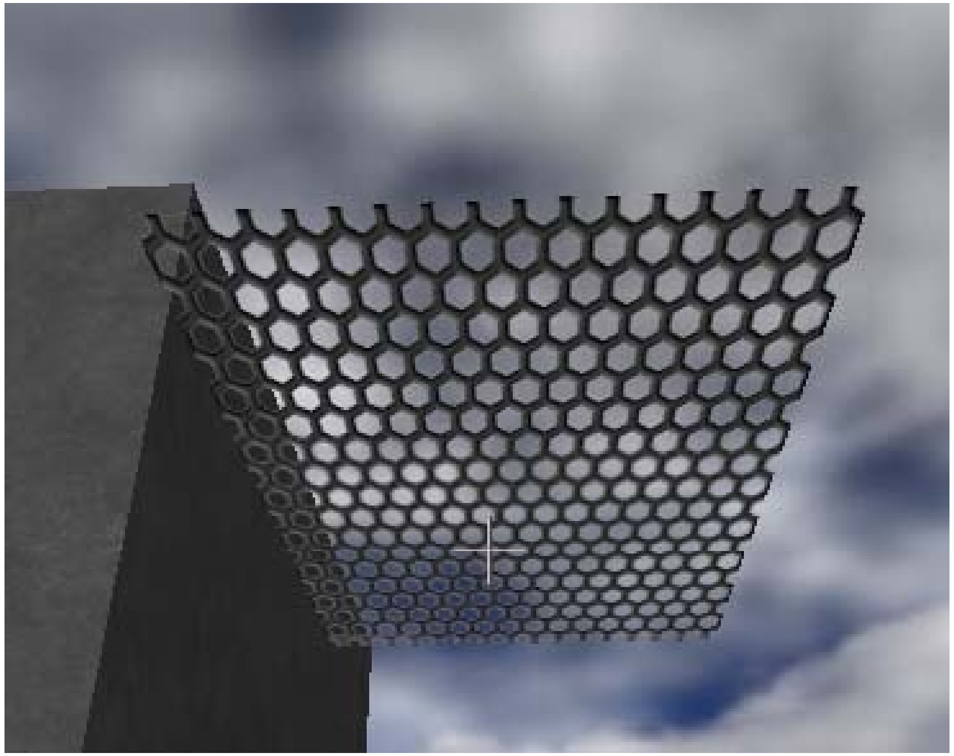
Opacity Maps



RGB channels

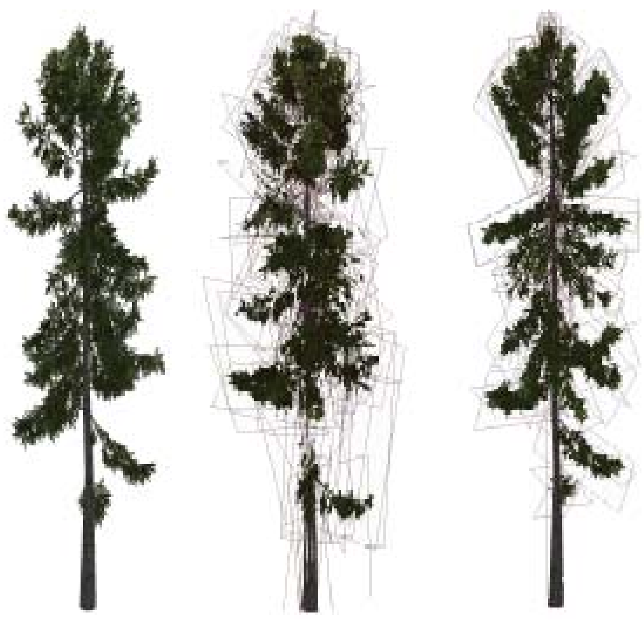


alpha channel



Use the alpha channel to make portions of the texture transparent

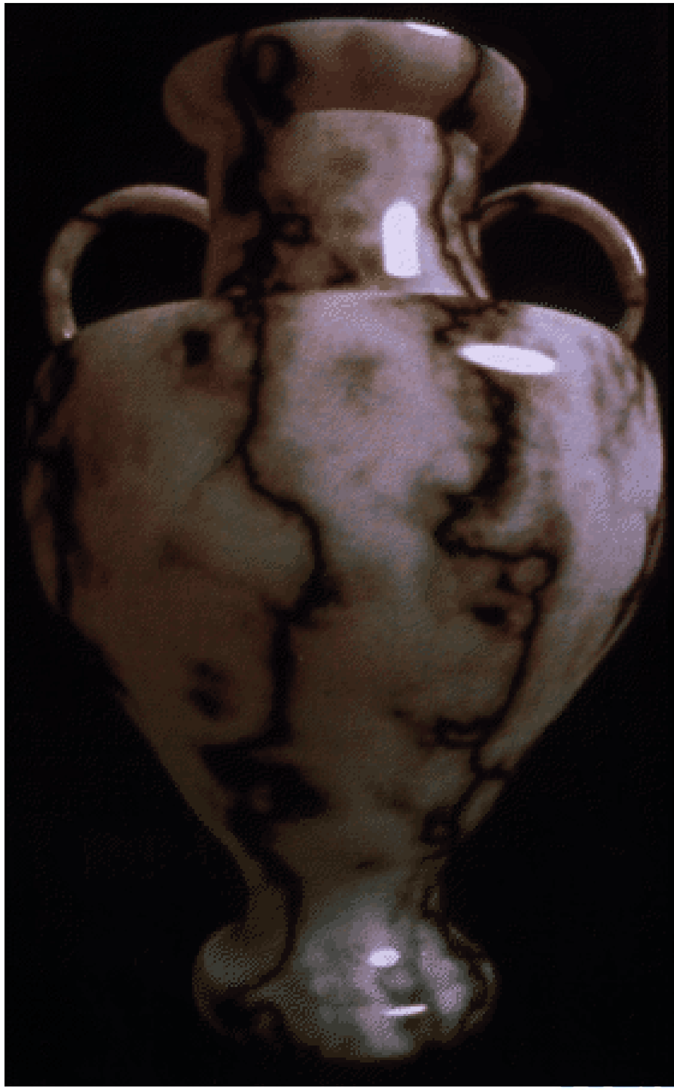
Billboards



Replace complex geometry with polygons texture mapped with transparent textures

3D or Solid Textures

- Solid textures are three dimensional assigning values to points in 3 space
  - Very effective at representing some types of materials such as marble and wood
  - The object is “carved” out of the solid texture
- Generally, solid textures are defined procedural functions rather than tabularized or sampled functions as used in 2D



Next Time

- Visibility