CMP205: Computer Graphics

Lecture 8: Texture Mapping II

Mohamed Alaa El-Dien Aly
Computer Engineering Department
Cairo University
Fall 2013

Agenda

- 2D Textures
- MIP Maps
- Bump Mapping
- Displacement Mapping
- Environment Mapping
- Shadow Mapping

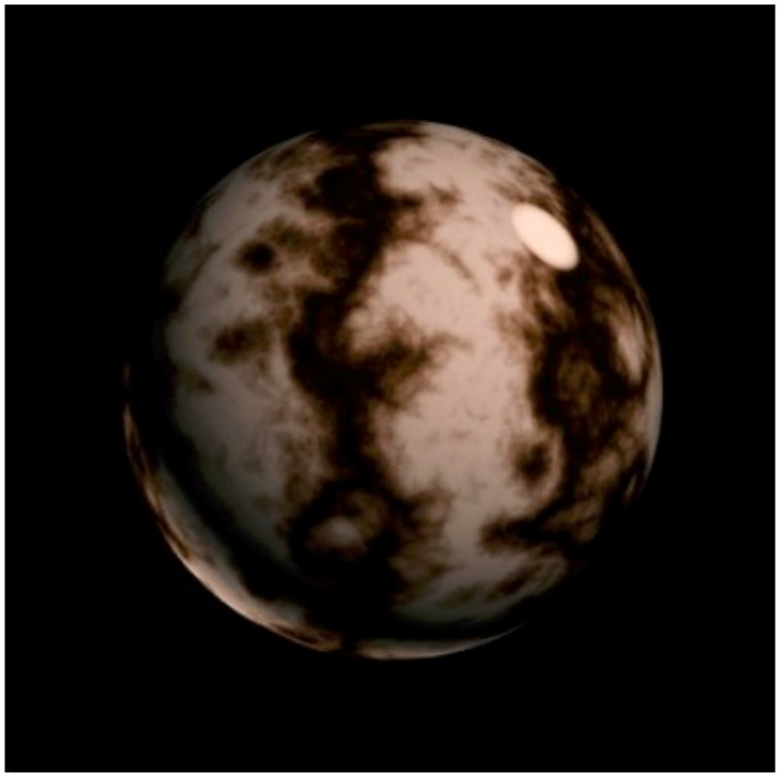
Acknowledgment: Some slides adapted from Steve Marschner and Maneesh Agrawala

Texture Mapping

The technique of controlling the appearance parameters of surfaces (e.g. shading, surface normals, color, … etc) as a function of position on the surface

Texture Mapping Types

Procedural Textures



- Call a function for each point
- Useful for creating noise, turbulence, … etc

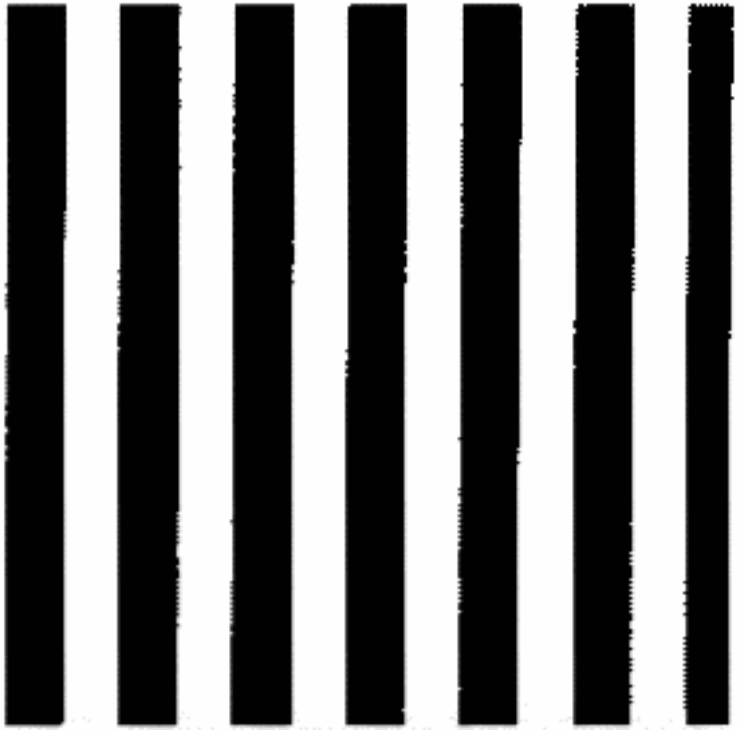
Lookup Textures

- Make a correspondence between each point and an entry in a lookup table
- Useful for reflections, mapping images, … etc.

Procedural Textures

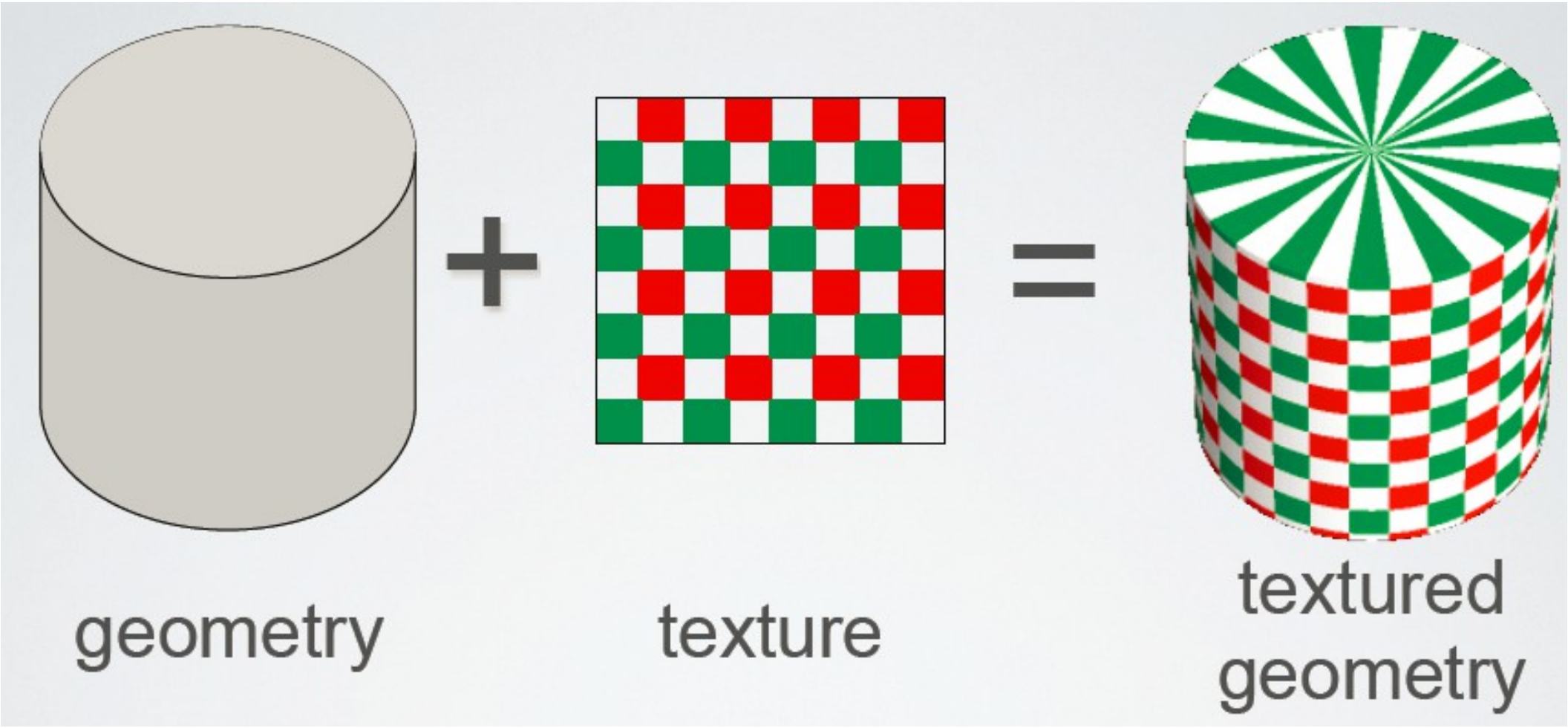
```
color Stripe(p, w)

if sin(π * x / w) > 0
    return c1
else
    return c2
```



Striped Texture

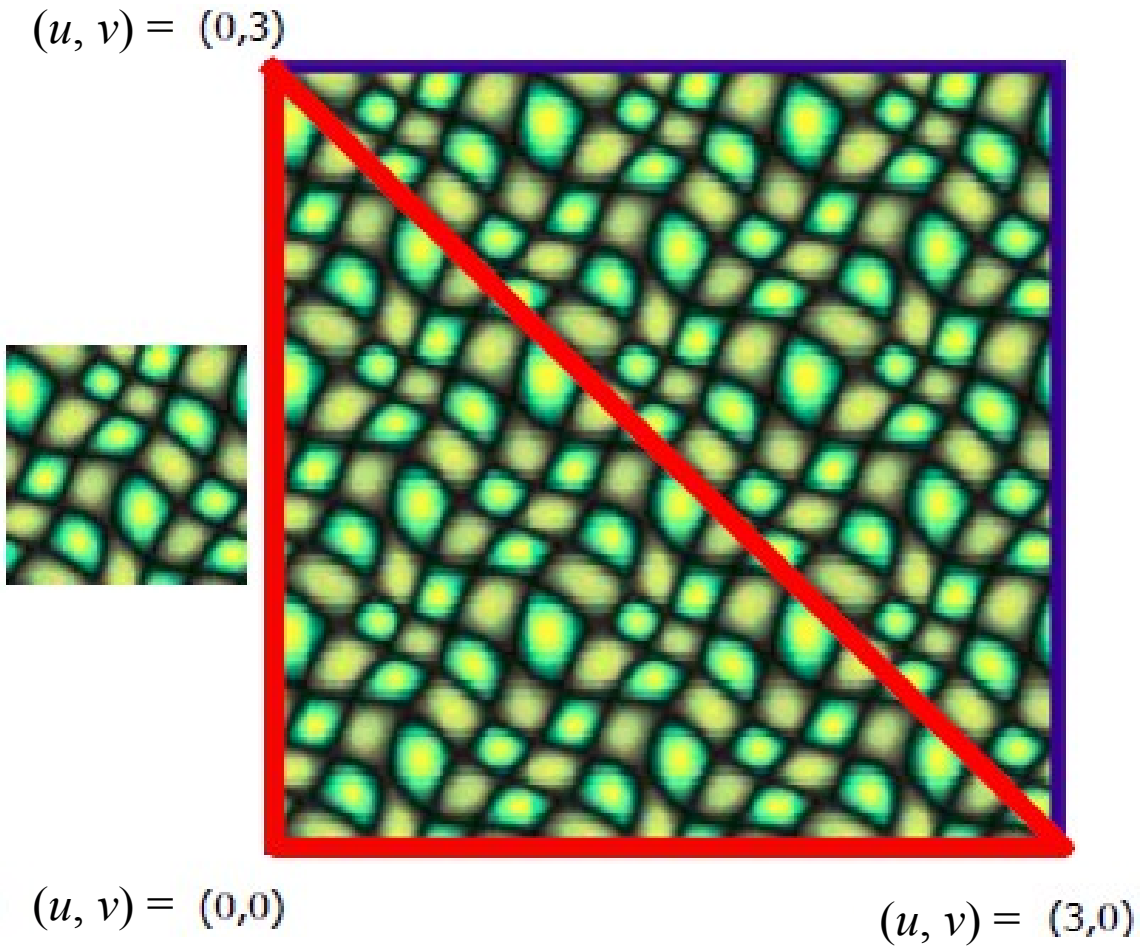
2D Textures



How do we decide where on the geometry each color from the image should go?

Texture Coordinates

- To put textures onto objects, we need to include it in modeling
- Specify texture coordinates for each vertex in object space
- Barycentric interpolation at interior points

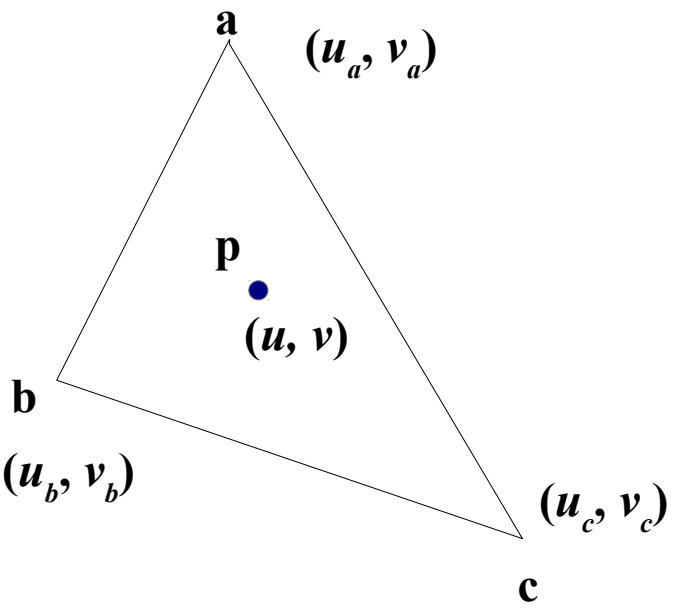


Texture Coordinates

- To put textures onto objects, we need to include it in modeling
- Specify texture coordinates for each vertex in object space
- Barycentric interpolation at interior points

$$p(\beta,\gamma)=a+\beta(b-a)+\gamma(c-a)$$

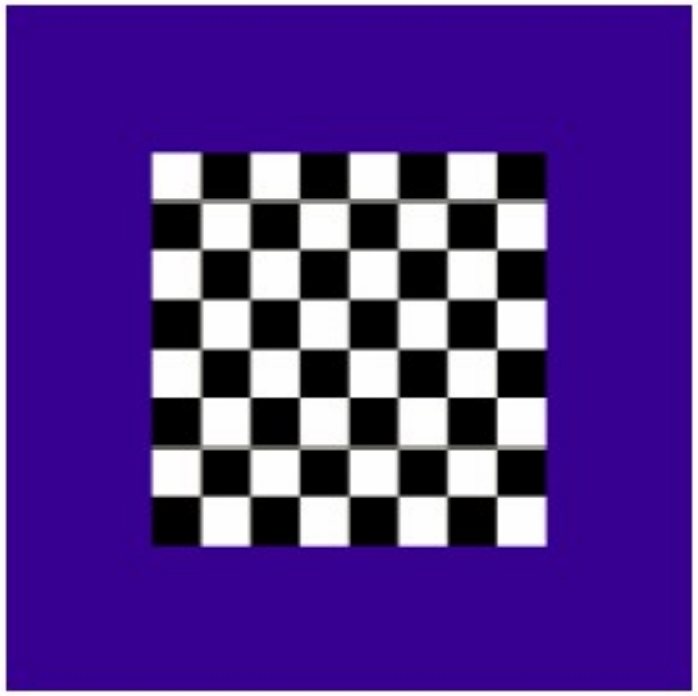
$$u(\beta,\gamma)=u_a+\beta(u_b-u_a)+\gamma(u_c-u_a)$$
$$v(\beta,\gamma)=v_a+\beta(v_b-v_a)+\gamma(v_c-v_a)$$



Can we interpolate in *screen space* i.e. using pixel coordinates for p, a, b, and c?

Texture Coordinates

Interpolation in screen space



texture source

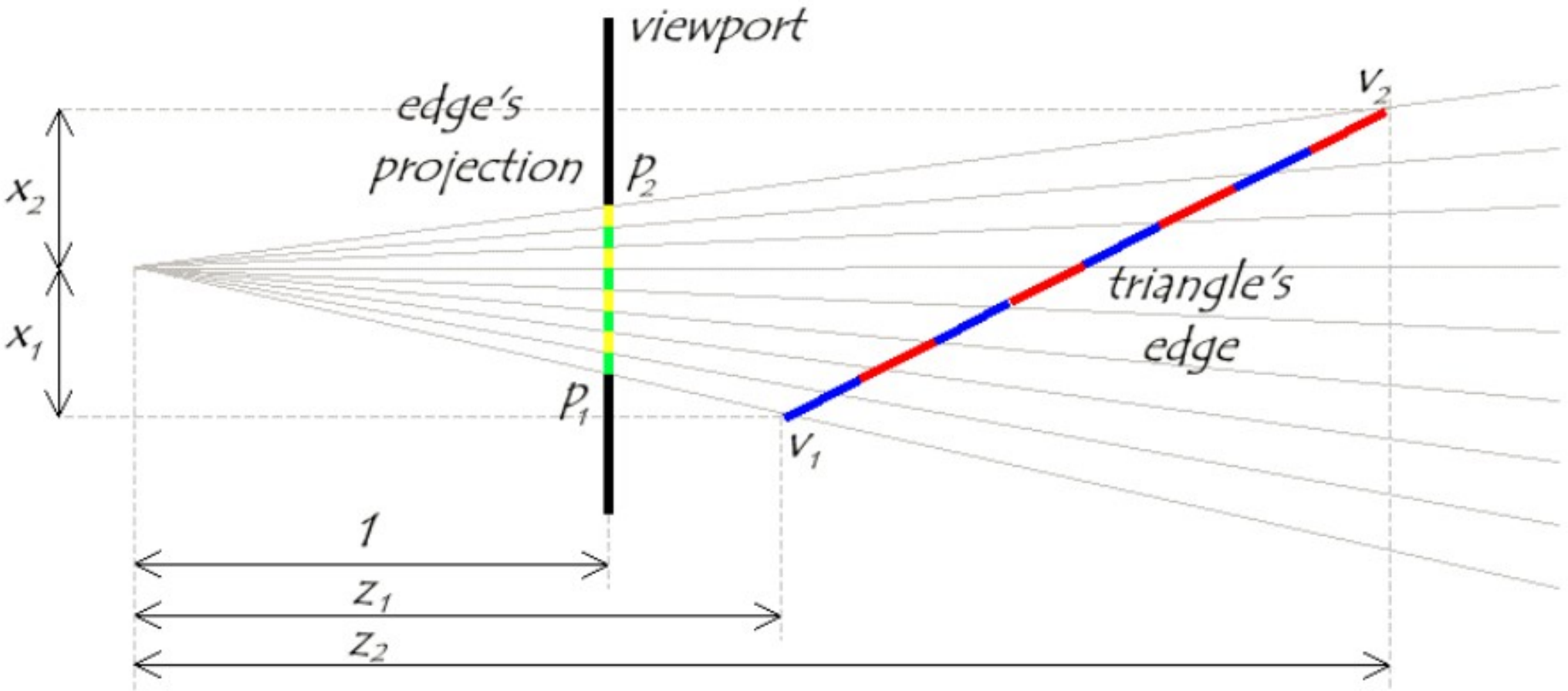
what we get

what we want

What goes wrong ?!

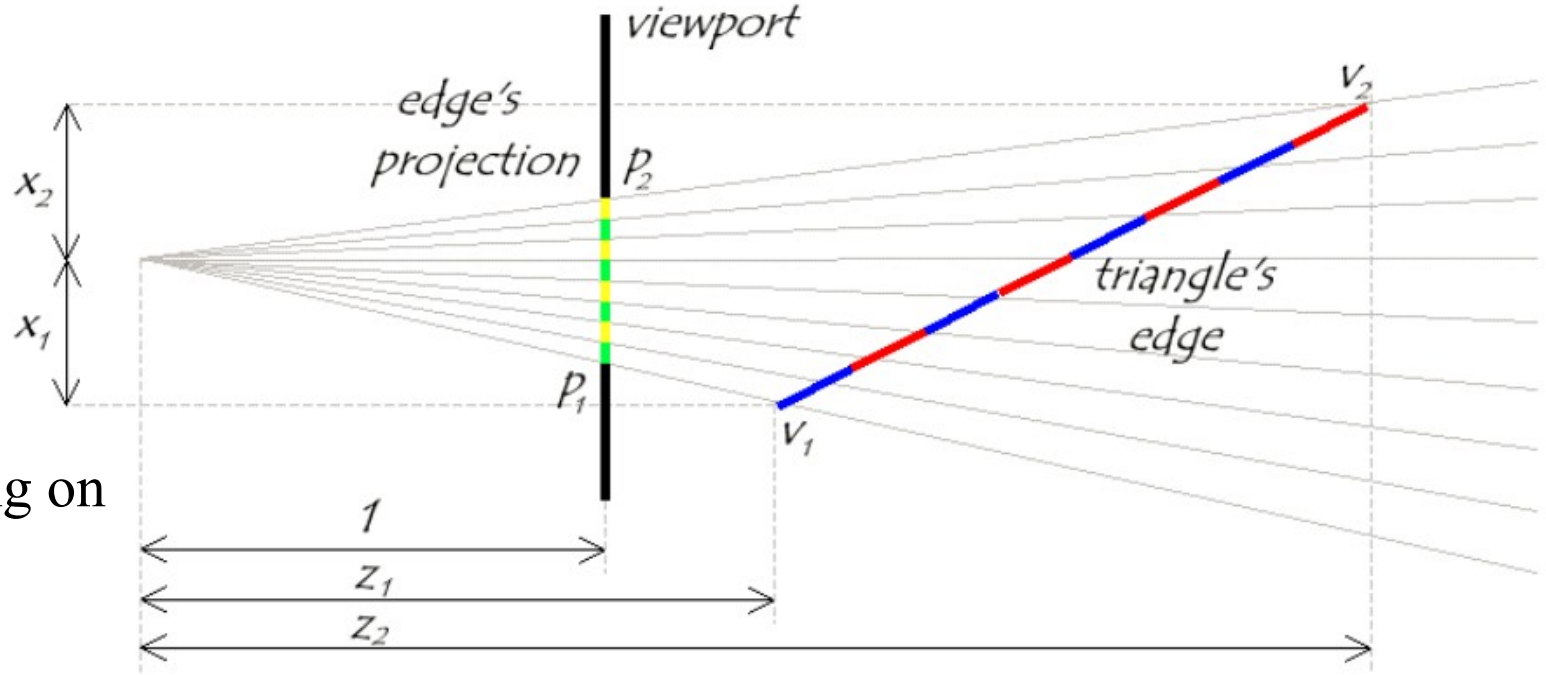
Texture Coordinates

Let's focus on linear interpolation i.e. just lines or on an edge of the triangle



Uniform steps in screen coordinates ≠ Uniform steps in world coordinates

Perspective Correct Interpolation



just focusing on the x and z coordinates

Interpolation in World Coordinates on the line v_1v_2

$$\begin{bmatrix} x_w \\ z_w \end{bmatrix}(s) = \begin{bmatrix} x_1 \\ z_1 \end{bmatrix} + s\left(\begin{bmatrix} x_2 \\ z_2 \end{bmatrix} - \begin{bmatrix} x_1 \\ z_1 \end{bmatrix}\right)$$

Screen Coordinates after projection

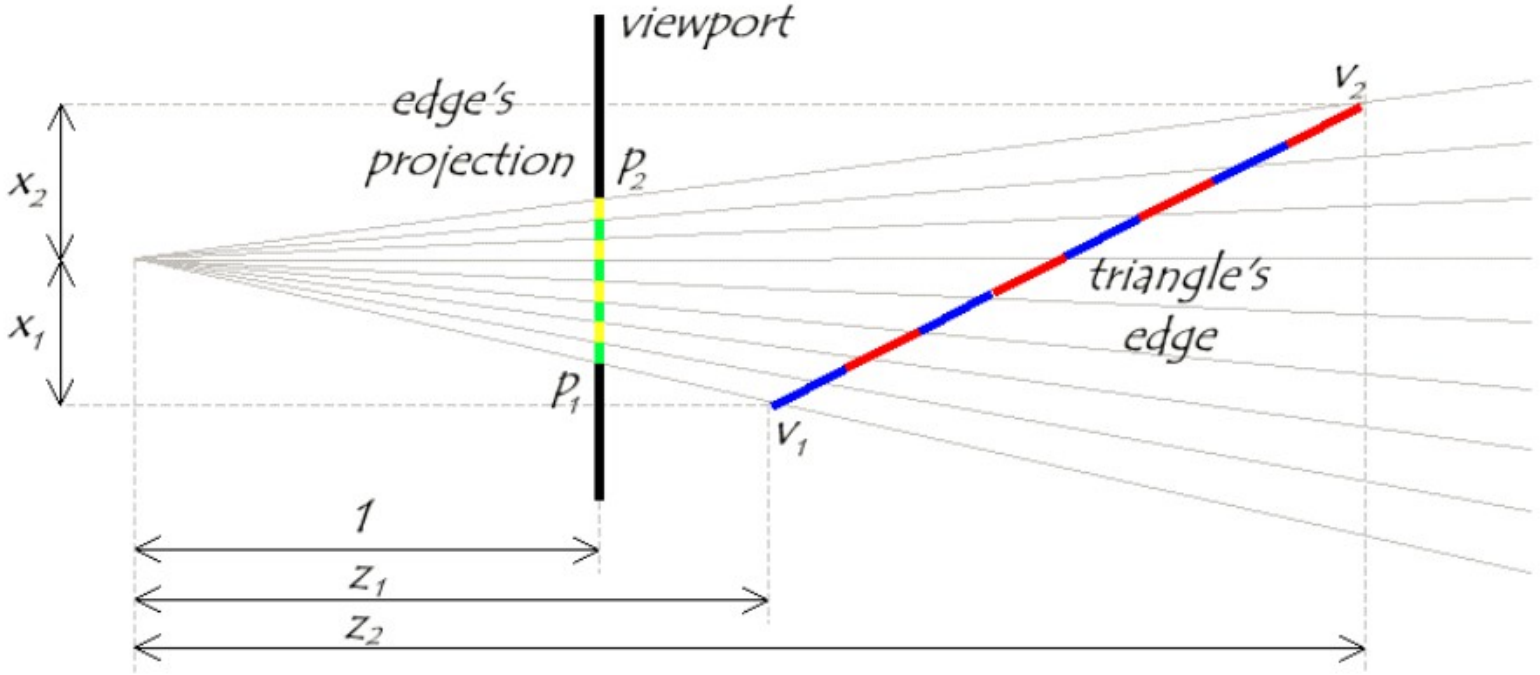
$$x = \frac{x_1 + s(x_2 - x_1)}{z_1 + s(z_2 - z_1)}$$

Interpolation in Screen Coordinates

$$x = \frac{x_1}{z_1} + t\left(\frac{x_2}{z_2} - \frac{x_1}{z_1}\right)$$

We want s in terms of t because we have t during rasterization

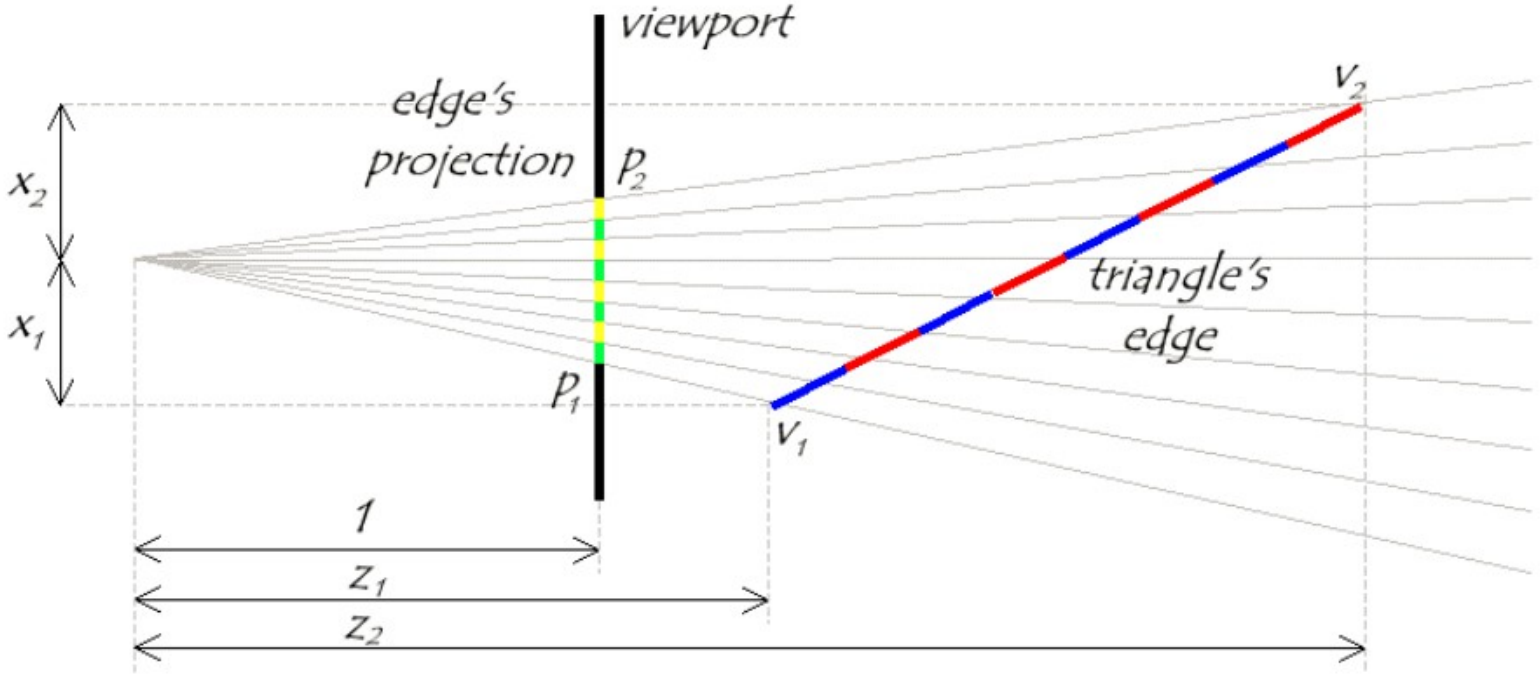
Perspective Correct Interpolation



$$\frac{x_1 + s(x_2 - x_1)}{z_1 + s(z_2 - z_1)} = \frac{x_1}{z_1} + t\left(\frac{x_2}{z_2} - \frac{x_1}{z_1}\right)$$

$$s = \frac{t\, z_1}{z_2 + t(z_1 - z_2)}$$

Perspective Correct Interpolation



$$s = \frac{t\, z_1}{z_2 + t(z_1 - z_2)}$$

Now use s to interpolate u & v

$$u = u_1 + s(u_2 - u_1)$$

$$u = \frac{\frac{u_1}{z_1} + t\left(\frac{u_2}{z_2} - \frac{u_1}{z_1}\right)}{\frac{1}{z_1} + t\left(\frac{1}{z_2} - \frac{1}{z_1}\right)} \quad \rightarrow \frac{u}{z} = \frac{u_1}{z_1} + t\left(\frac{u_2}{z_2} - \frac{u_1}{z_1}\right)$$

Interpolating u/z and $1/z$ in Screen Coordinates is correct !

Perspective Correct Interpolation

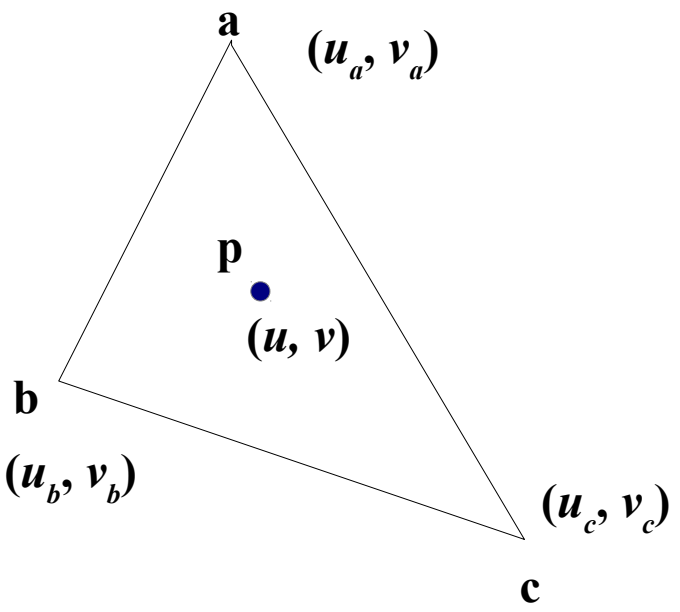
For triangles, use Barycentric interpolation for u/z and $1/z$:

$$\frac{u}{z} = \frac{u_a}{z_a} + \beta\left(\frac{u_b}{z_b} - \frac{u_a}{z_a}\right) + \gamma\left(\frac{u_c}{z_c} - \frac{u_a}{z_a}\right)$$

where $$\frac{1}{z} = \frac{1}{z_a} + \beta\left(\frac{1}{z_b} - \frac{1}{z_a}\right) + \gamma\left(\frac{1}{z_c} - \frac{1}{z_a}\right)$$

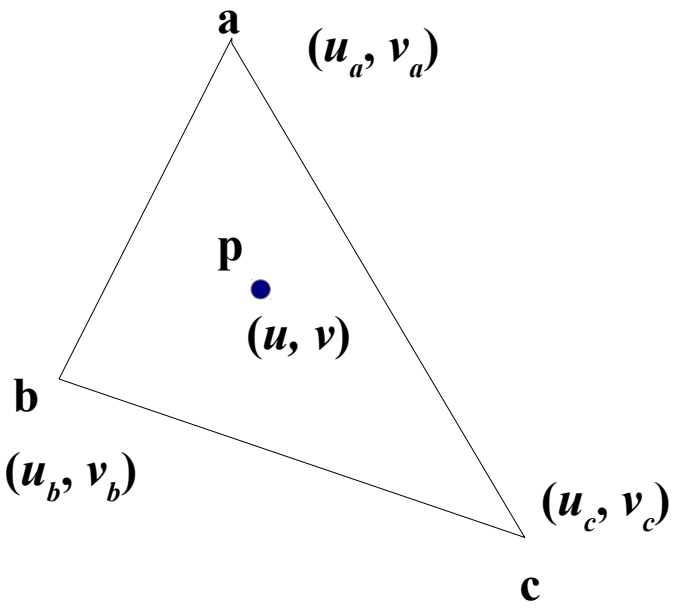
Then compute u from $\frac{u}{z}$

This requires *seven* divisions per value!



Perspective Correct Interpolation

Instead, we can compute β_w and γ_w in the world coordinates from their screen values β and γ (as we got s from t)



We can interpolate β_w / z in screen coordinates

$$\frac{\beta_w}{z} = \frac{\beta_a}{z_a} + \beta\left(\frac{\beta_b}{z_b} - \frac{\beta_a}{z_a}\right) + \gamma\left(\frac{\beta_c}{z_c} - \frac{\beta_a}{z_a}\right)$$

But we know that: $\beta_a = \beta_c = 0$ and $\beta_b = 1$

$$\frac{\beta_w}{z} = \frac{\beta}{z_b} \quad \rightarrow \quad \beta_w = \frac{\frac{\beta}{z_b}}{\frac{1}{z_a} + \beta\left(\frac{1}{z_b} - \frac{1}{z_a}\right) + \gamma\left(\frac{1}{z_c} - \frac{1}{z_a}\right)}$$

$$\rightarrow \quad \beta_w = \frac{z_a z_c \beta}{z_b z_c + \beta z_c (z_a - z_b) + \gamma z_b (z_a - z_c)}$$

Perspective Correct Interpolation

Instead, we can compute β_w and γ_w in the world coordinates from their screen values β and γ (as we got s from t)

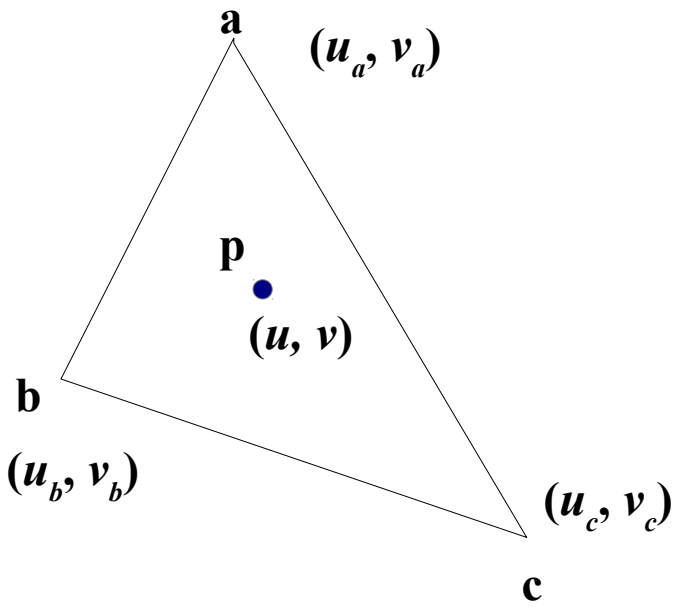
$$\beta_w = \frac{z_a z_c \beta}{z_b z_c + \beta z_c (z_a - z_b) + \gamma z_b (z_a - z_c)}$$

Similarly for γ_w

$$\gamma_w = \frac{z_a z_b \gamma}{z_b z_c + \beta z_c (z_a - z_b) + \gamma z_b (z_a - z_c)}$$

Then using: β_w and γ_w

$$u = u_a + \beta_w (u_b - u_a) + \gamma_w (u_c - u_a)$$



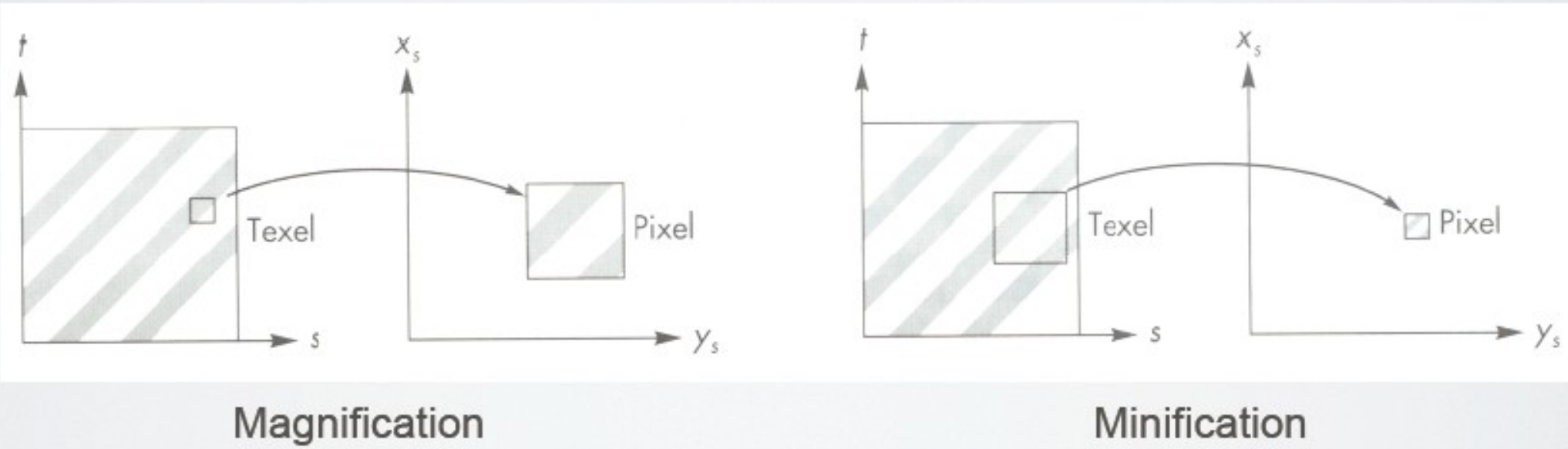
This requires *two* division per value!

Perspective Correct Interpolation

- Interpolating (u, v) in screen space is wrong ...
- However, interpolating (u/z, v/z) in screen space is right !
- We *might not* have z at this point, possibly $h=z/n$
- So
  - Interpolate $1/h$, u/h, v/h, or
  - Compute β_w and γ_w from the h's (instead of the z's) and interpolate u and v
- Either case, you will need h or z

$$P=\begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -fn \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

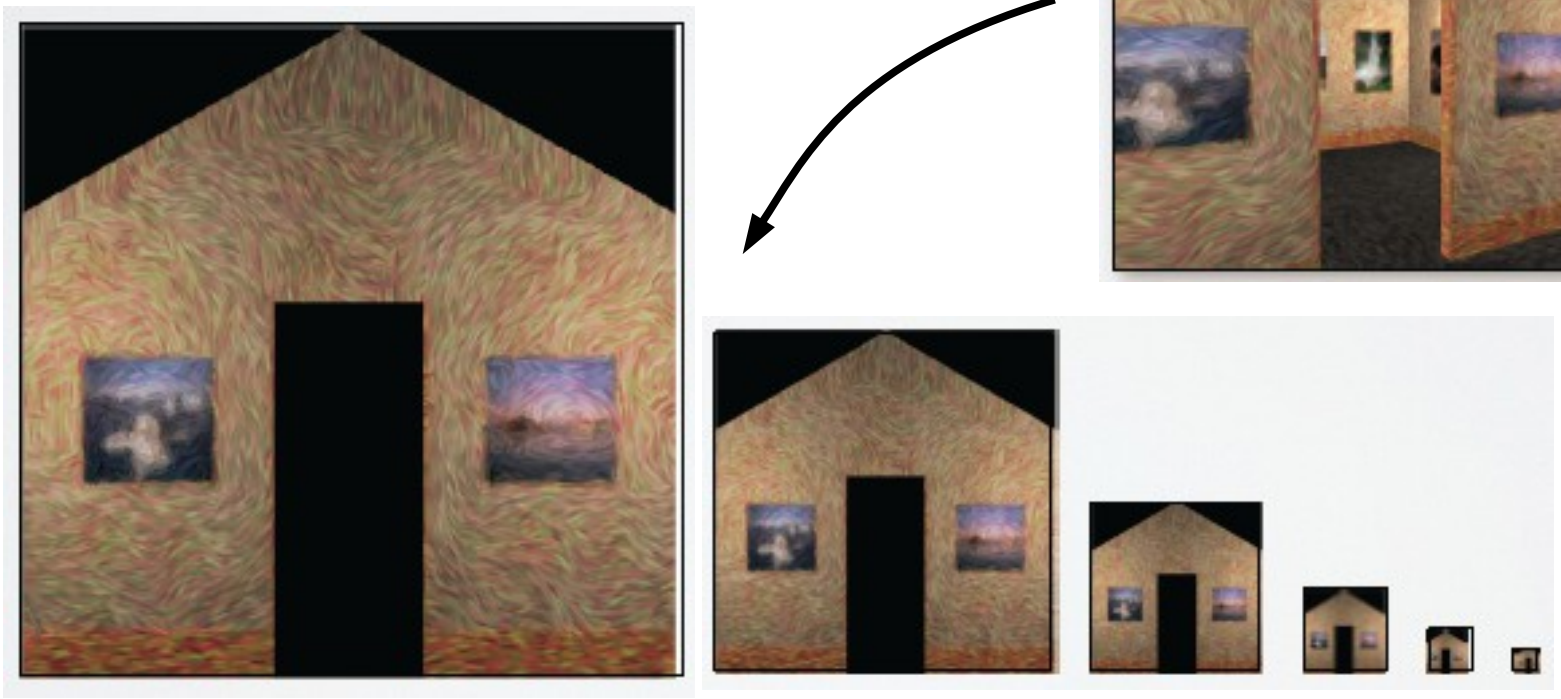
Texture Filtering



Some times pixels and texels have different sizes

MIP Maps

Use trilinear interpolation in space and scale



Create different scaled versions of the texture

MIP Maps

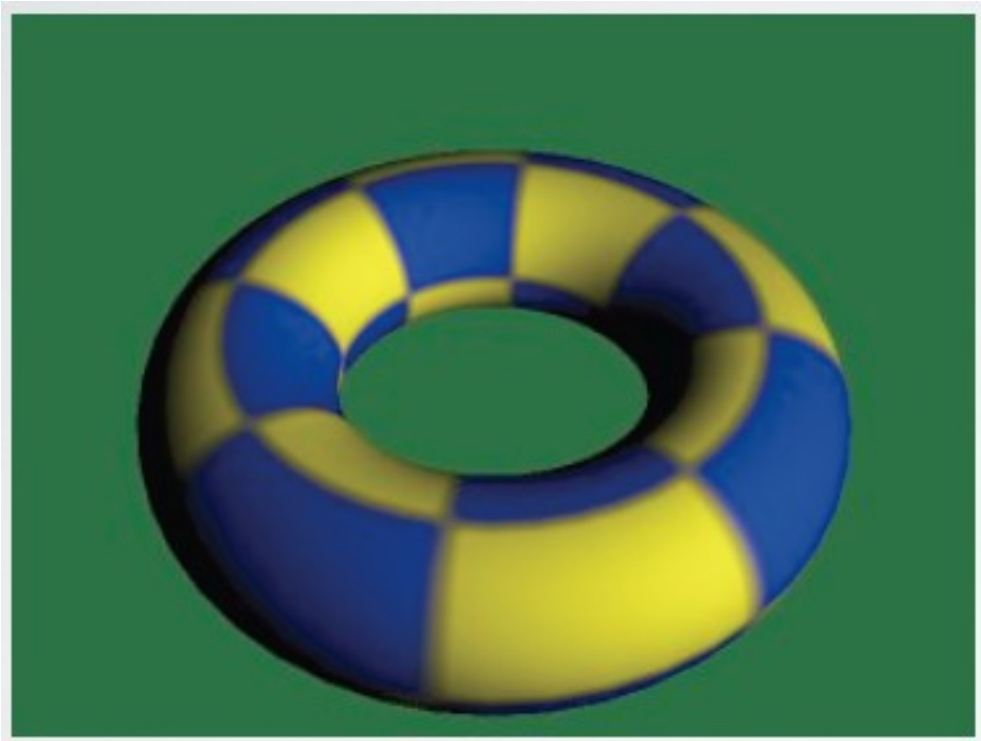
No filtering

MIP Map filtering

Variations

Can modulate any parameter in the Phong Shading Model

Bump Mapping

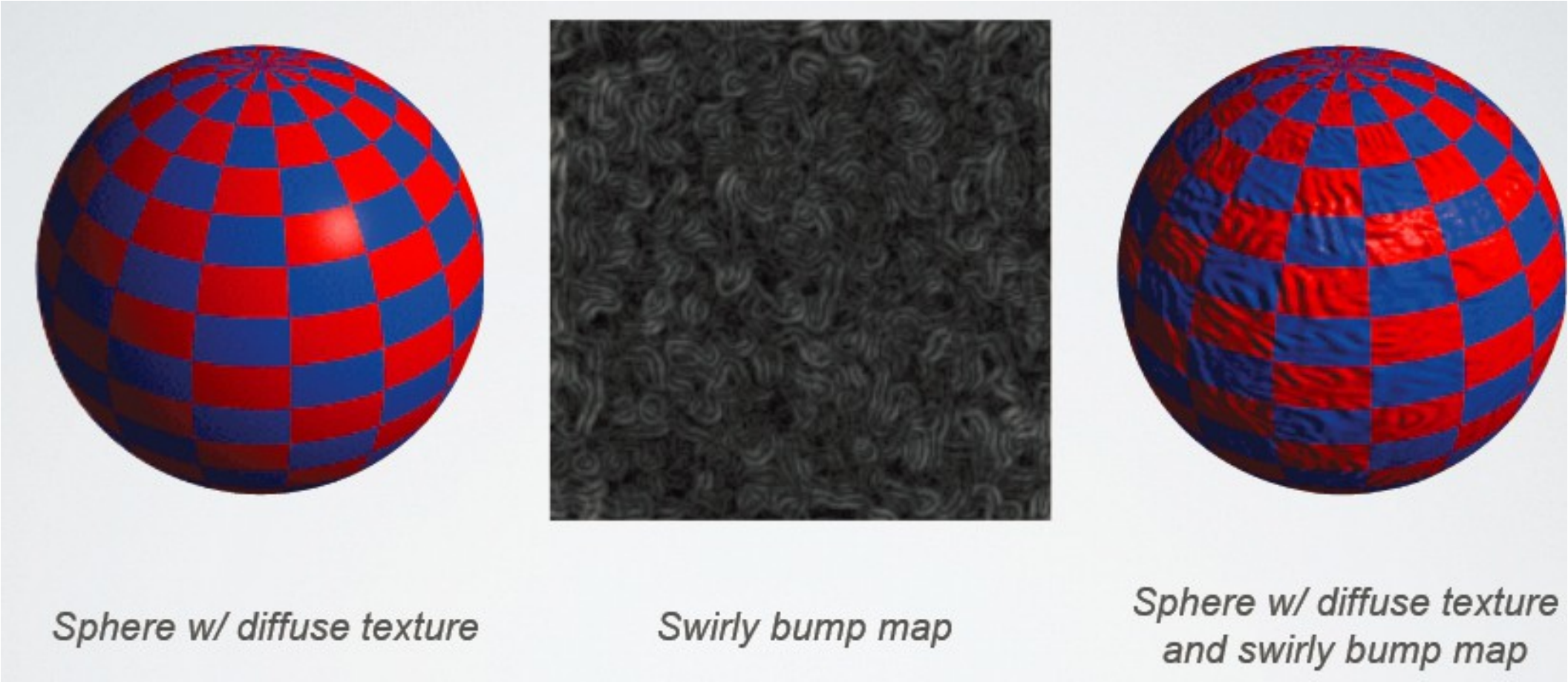


No bump mapping

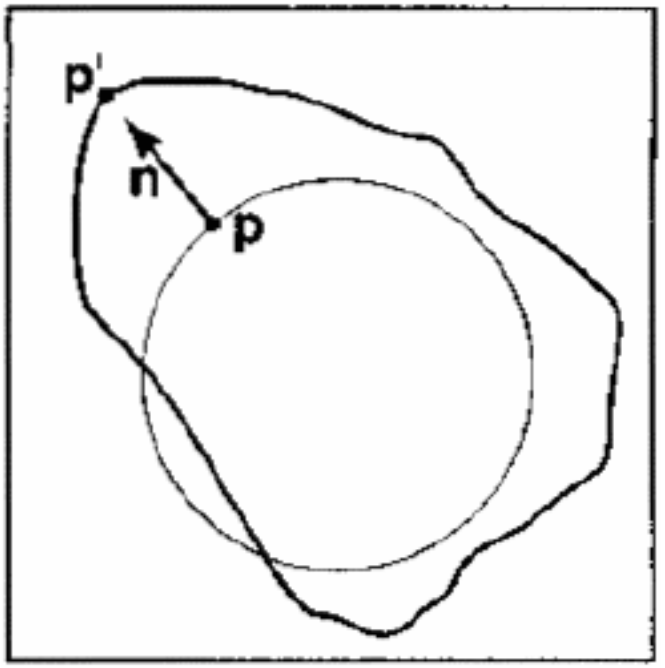
With bump mapping

Map to perturb the *surface normals*

Bump Mapping

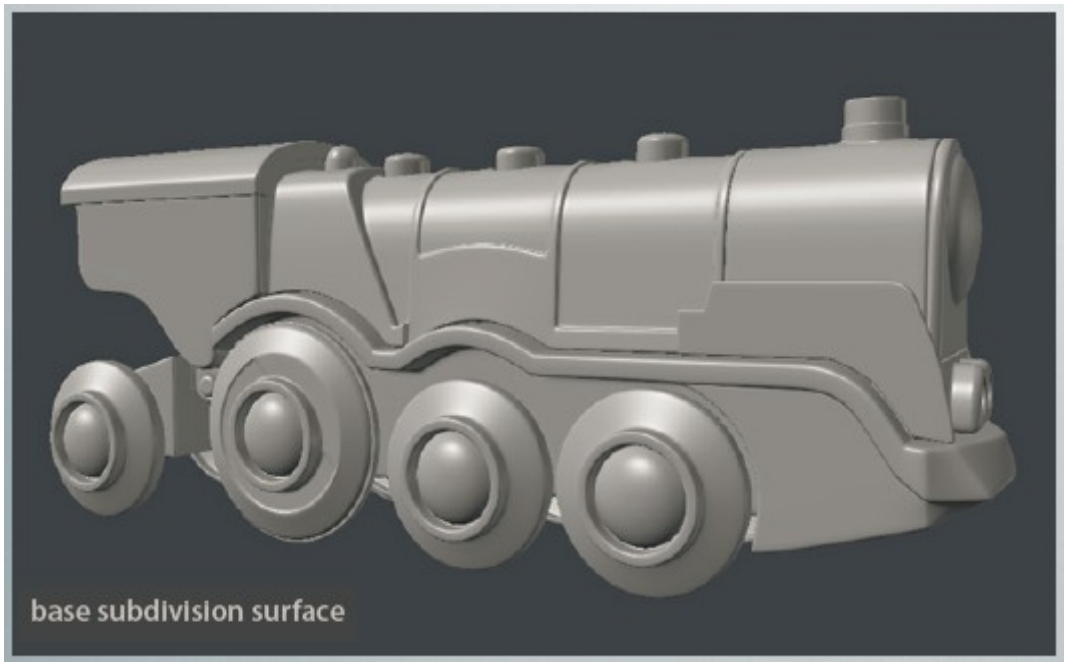


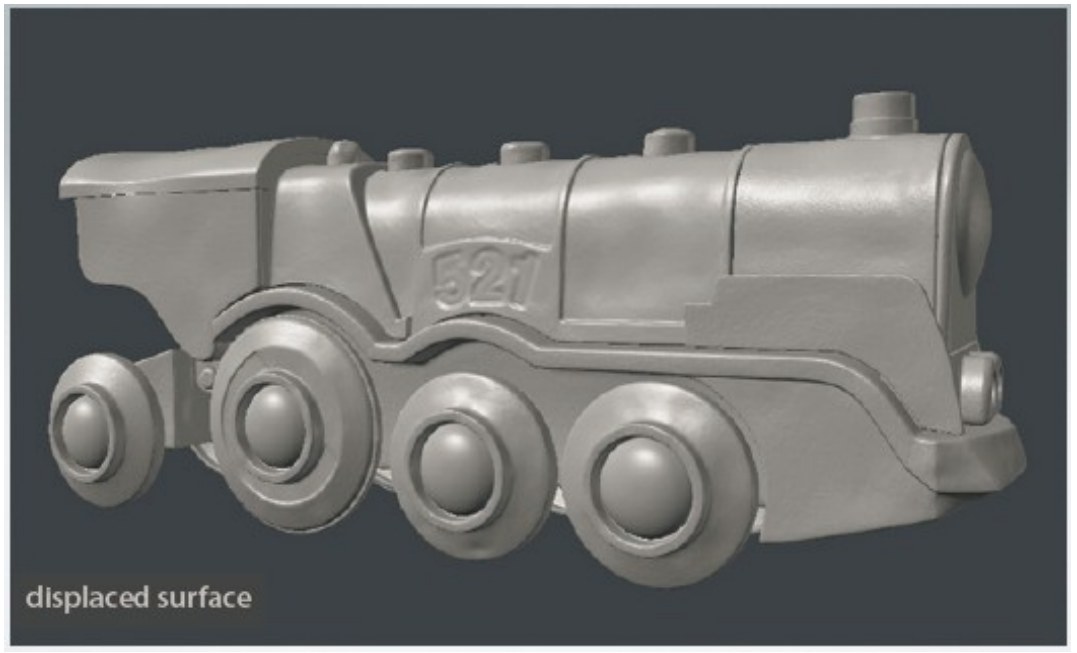
Displacement Mapping



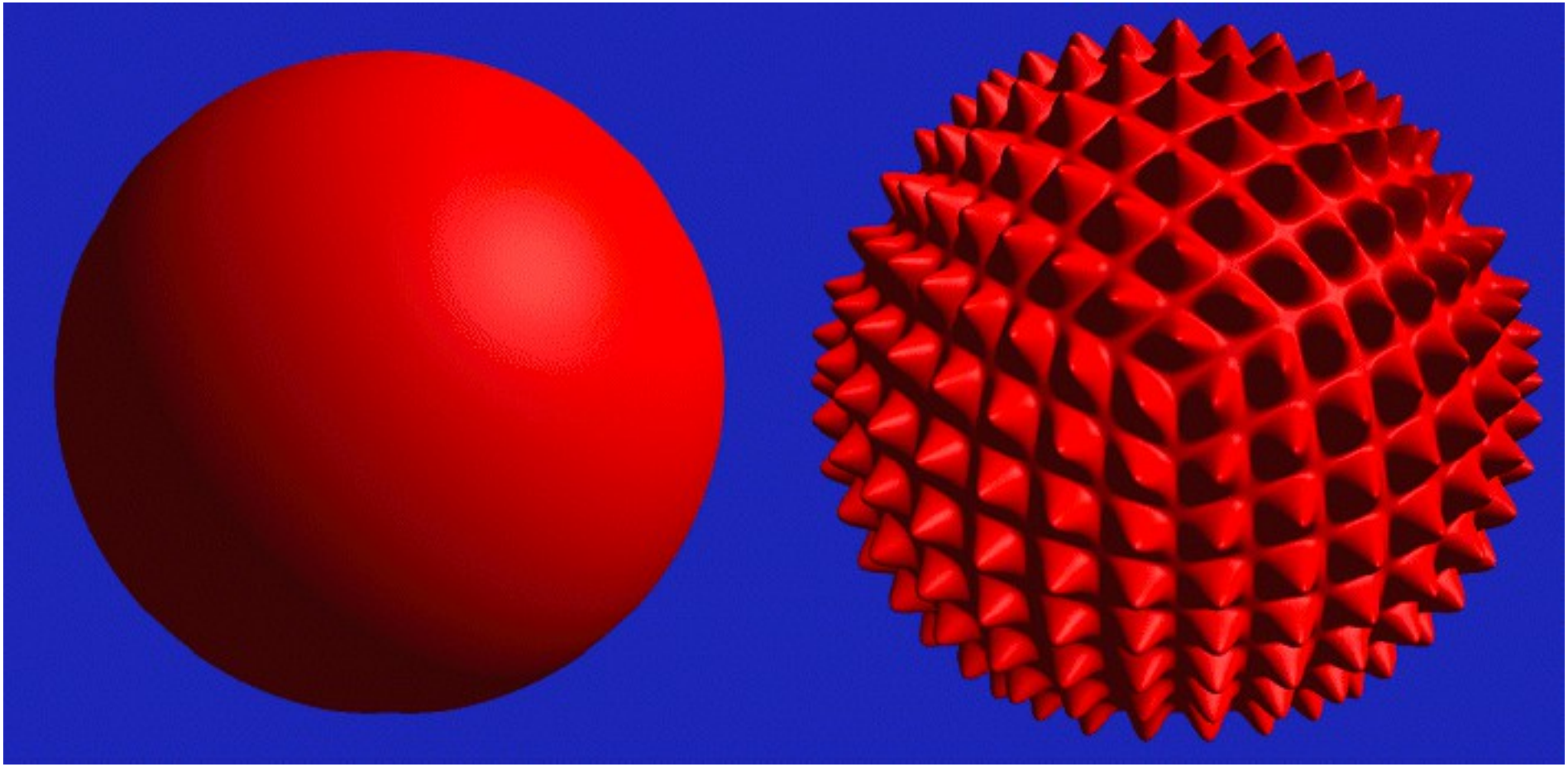
Move points on the surface in the direction of the surface normal

Displacement Mapping





Displacement Mapping



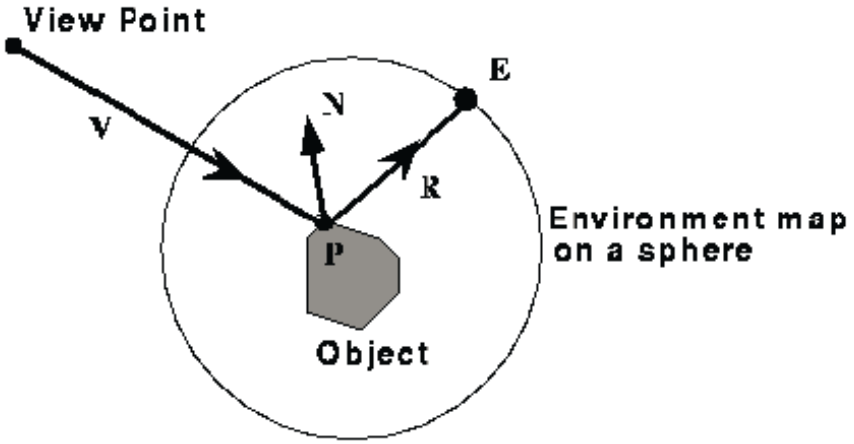
Environment Maps

Texture storing the background of the environment (scene)

Mapping the texture onto the tea pot gives the illusion of reflecting the background

Performs crude *reflections* with Environment Maps

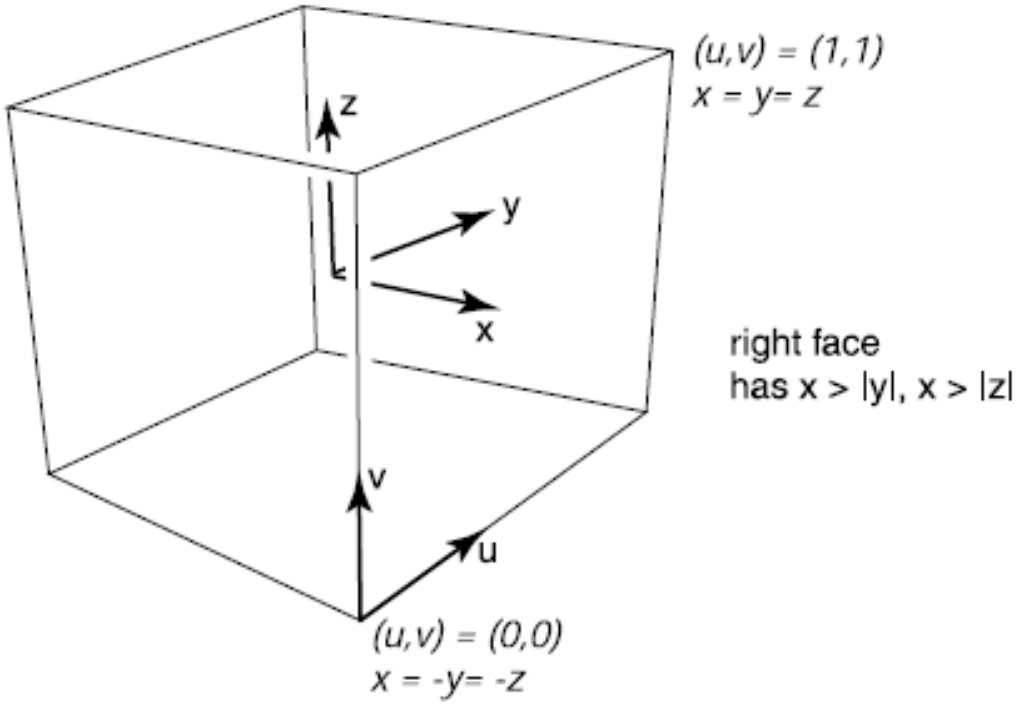
Environment Maps



Spherical Texture

Spherical Parametrization

Environment Maps



right face
has x > |y|, x > |z|

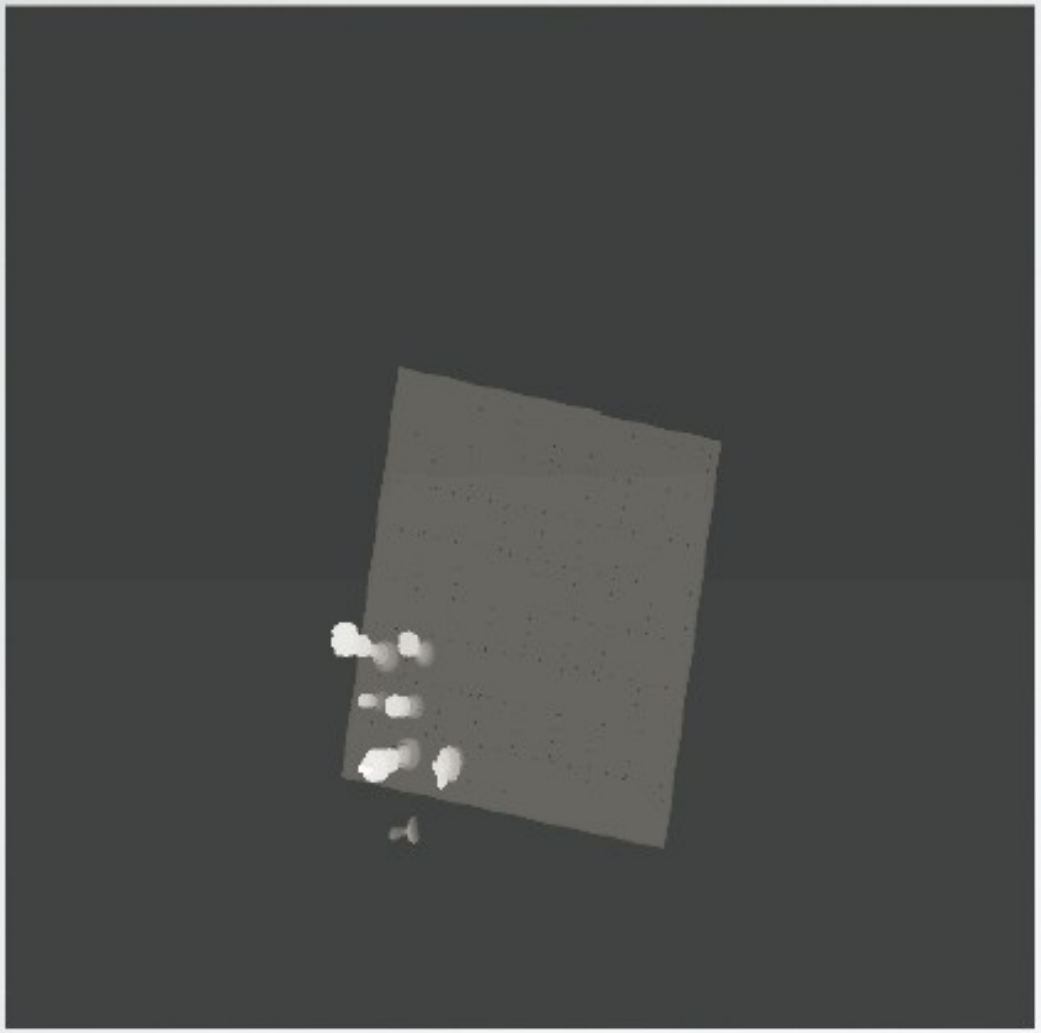
Cubic Parametrization

Shadow Maps

- Render the scene from the viewpoint of the light source
  - Keep *z*-values of pixels in a *shadow* depth map

- For every point being rendered, compute its depth from the light (by rendering again it from the light source)
  - Compare *z*-value (w.r.t. light source) with shadow map
    - If smaller → pixel is closer to light → pixel is lit
    - If larger → pixel is further from light → pixel is shadow

Shadow Maps



Shadow Buffer

Image w/ Shadows

Recap

- Texture Mapping
  - Procedural Textures
  - Lookup Textures
- 2D Textures
- MIP Maps
- Bump Mapping
- Displacement Mapping
- Environment Mapping
- Shadow Mapping