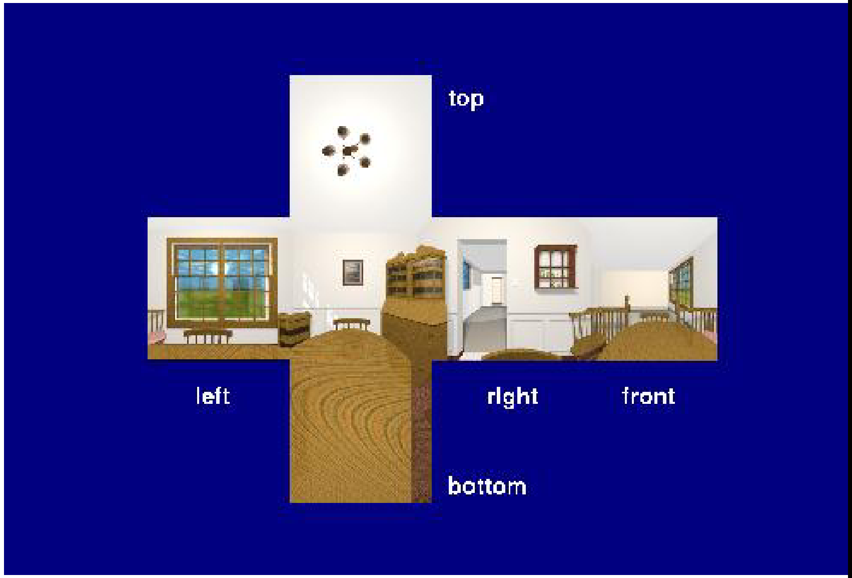
# Texture Mapping

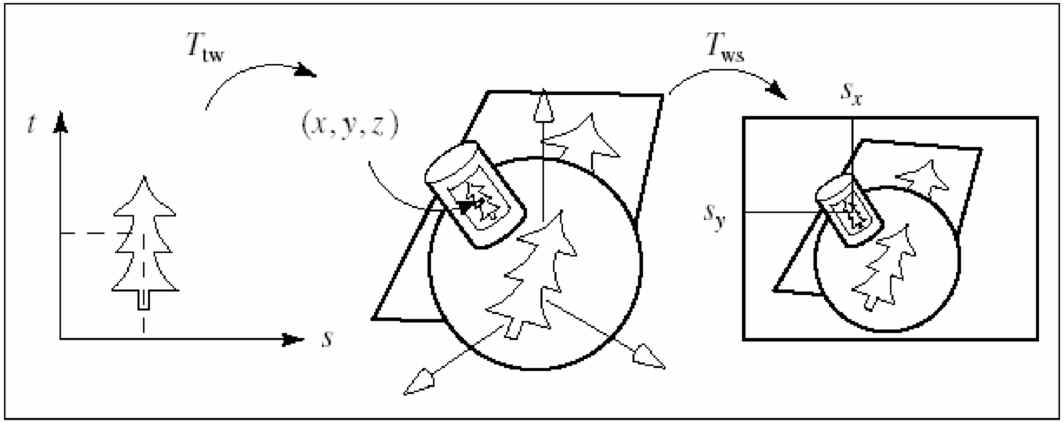
## *Pasting textures on surfaces: Hill 8.5*




# Systems involved

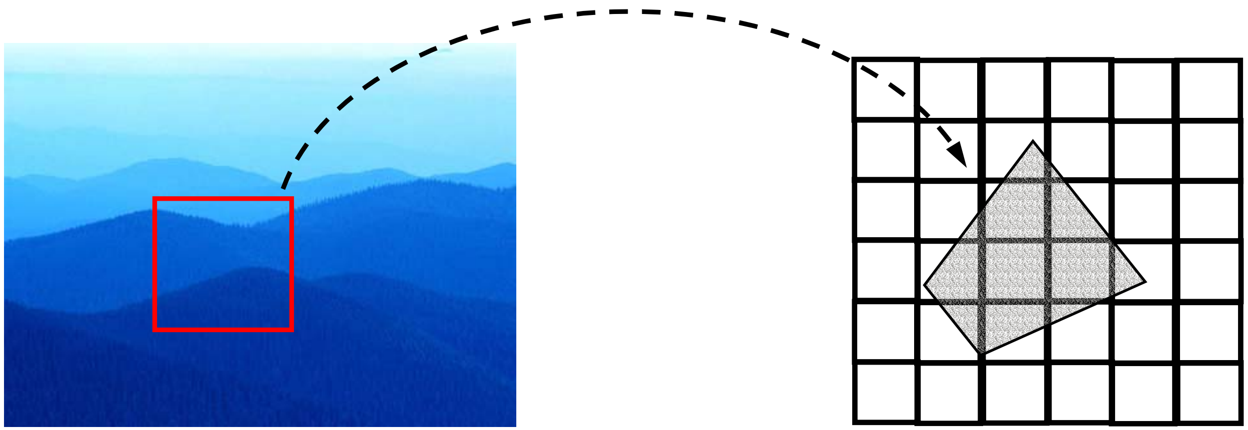
User Defined Viewing+Projection

FIGURE 8.35 Drawing texture on several objects of different shape.




(sx,sy) = Tws(Ttw(s,t))

# Texture to Screen

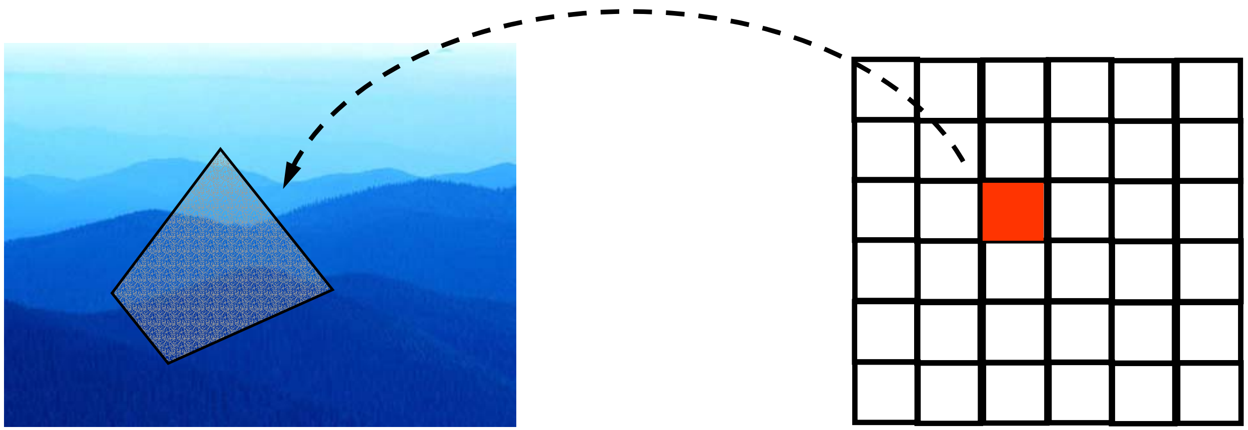


$(sx,sy) = Tws(Ttw(s,t))$

We would have to calculate pixel coverages

# Screen to texture

## *Better approach*

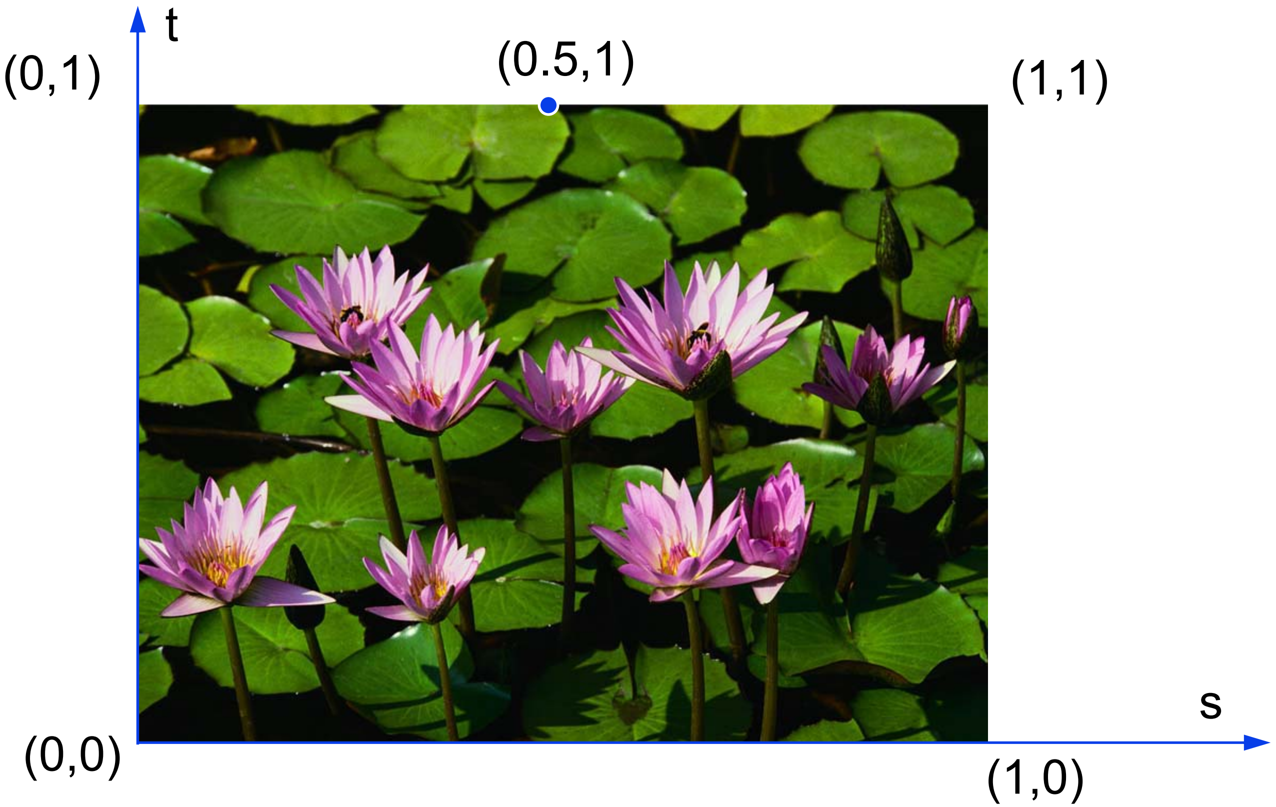


(s,t) = Twt(Tsw(sx,sy))

Requires inverting the projection matrix

# Textures are always images

**They are always assigned the shown parametric coordinates (s,t).**



# From texture to world (object)

***To apply a texture to an object we have to find a correspondance between (s,t) and and some object coordinate system.***

- Mapping via a parametric representation of the object space (points).
- By hand.

# Mapping from texture to a parametric representation of the object space

***Linear trasformation***

***Texture space (s,t) to object space (u,v)***

$u = u(s,t) = a_u\, s + b_u\, t + c_u$

$v = v(s,t) = a_v\, s + b_v\, t + c_v$

s in [0,1]

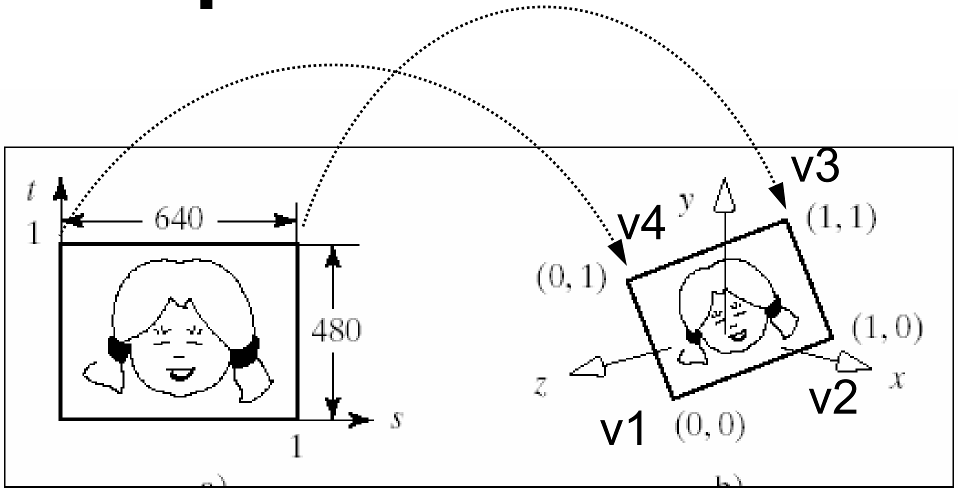
t in [0,1]

# Example: Image to a quadrilateral

***Simply***

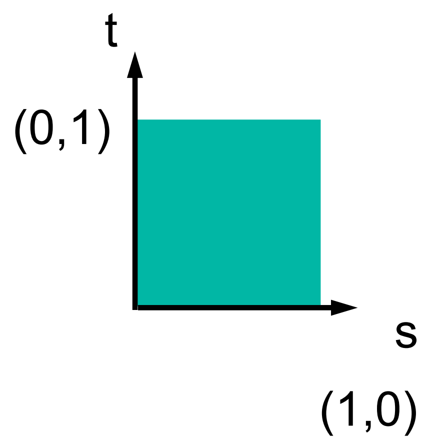
$u = u(s,t) = s$

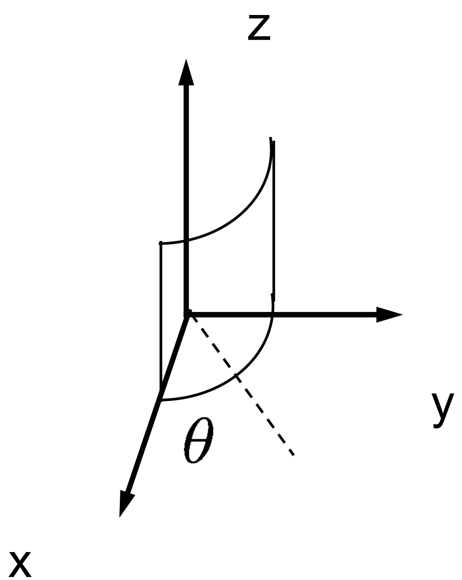
$v = v(s,t) = t$




```
glTexCoord2f(0,0) ;  glVertex3dv(v1) ;
glTexCoord2f(1,0) ;  glVertex3dv(v2) ;
glTexCoord2f(1,1) ;  glVertex3dv(v3) ;
glTexCoord2f(0,1) ;  glVertex3dv(v4) ;
```

# Example: Square texture to cylinder




$$Parametric\ \ form:$$

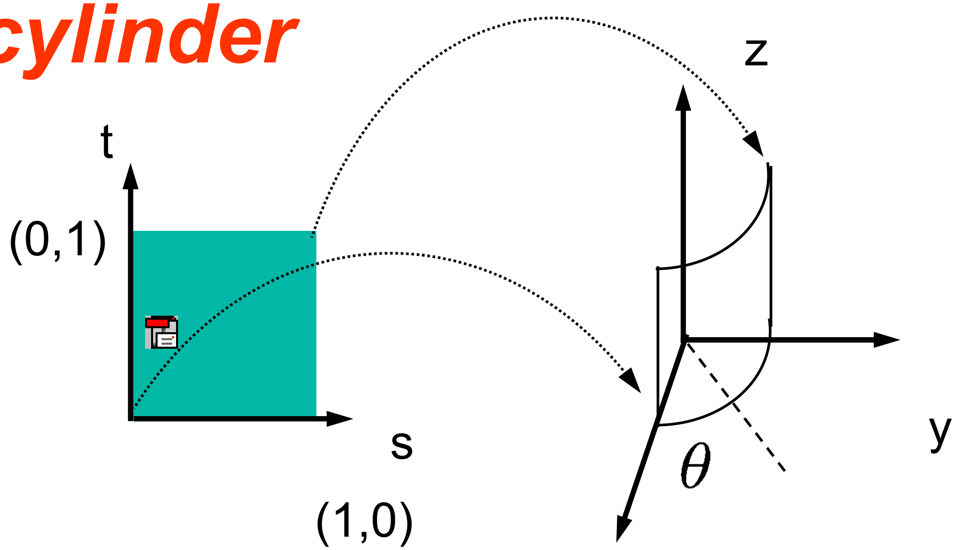
$$x = rcos\theta,\ \ y = rsin\theta,\ \ z$$

$$Surface\ \ parameters:\ \ u = \theta, v = z$$

$$with\ \ 0 \le u \le \pi/2,\ \ 0 \le v \le 1$$

# Example : Square texture to cylinder

## *Square texture to cylinder*

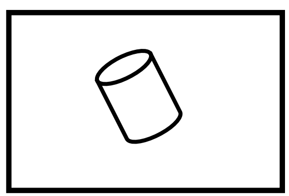


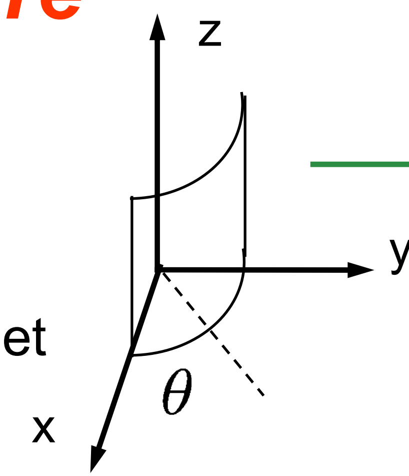

We pick the following linear transformation that maps $(s,t) = (0,0)$ to $(x,y,z) = (r,0,0)$ and $(s,t) = (1,1)$ to $(x,y,z) = (0,r,r)$.

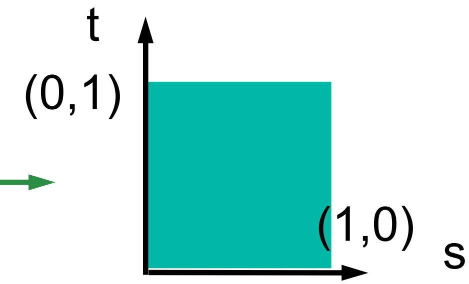
$$u = s\pi/2, \quad v = t$$

# Example : Square texture to cylinder

## *From screen to texture*






1. Inverse transform *(sx,sy)* to get world position *(x,y,z)*.
2. Then having *(x,y,z)*

$$u = tan^{-}1(y/x), \quad v = z$$

$$s = 2u/\pi, \quad t = v$$

$$Reminder: \quad u = s\pi/2, \quad v = t$$

$$x = rcos\theta, \quad y = rsin\theta, \quad z$$

$$Surface \;\; parameters: \quad u = \theta, v = z$$

# How does that work with the graphics pipeline?

***Only polygons***

***Only vertices go down the graphics pipeline.***

Interior points?

*Calculate texture coordinates by interpolation along scanlines.*

# Rendering the texture

## *Scanline in screen space*

- Generating s,t cordinates for each pixel

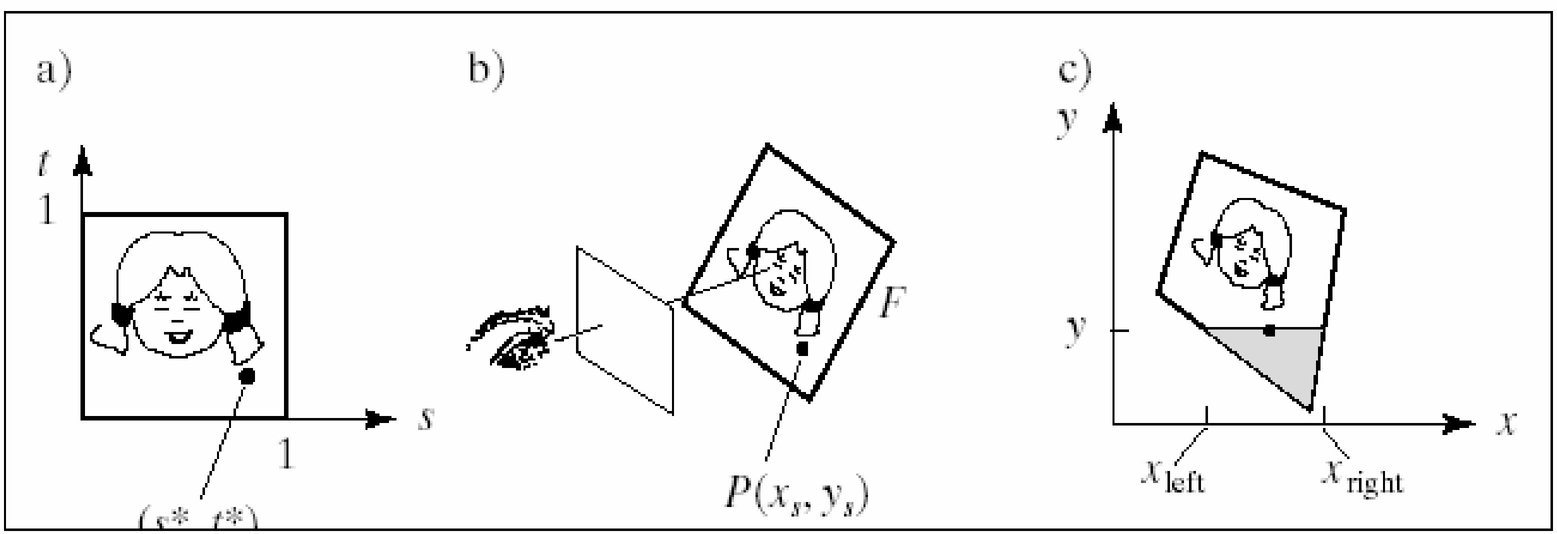



**FIGURE 8.39** Rendering a face in a camera snapshot.

# Interpolation of texture coordinates

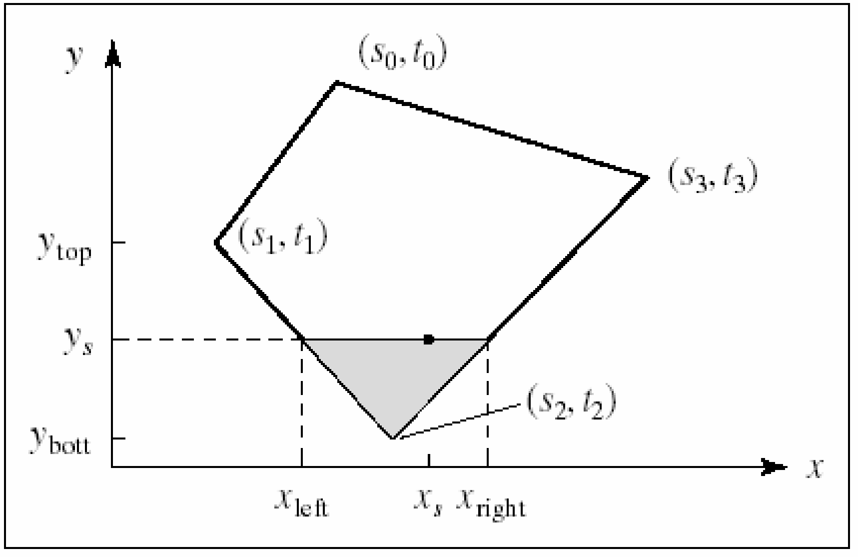
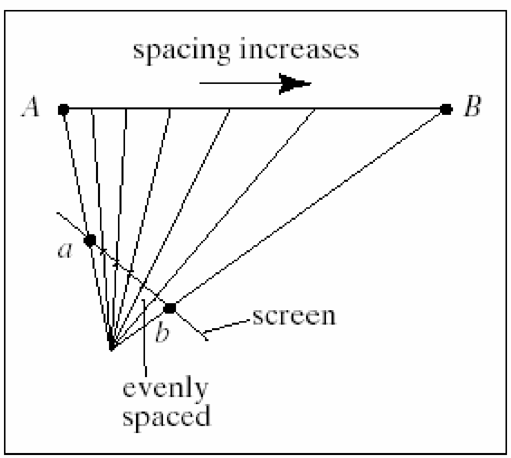



**FIGURE 8.40** Incremental calculation of texture coordinates.

# Problem

## *Perspective forshortening*

- Scanconversion takes equal steps along scanline (screen space)
- Equal steps in screen space not equal steps in world space



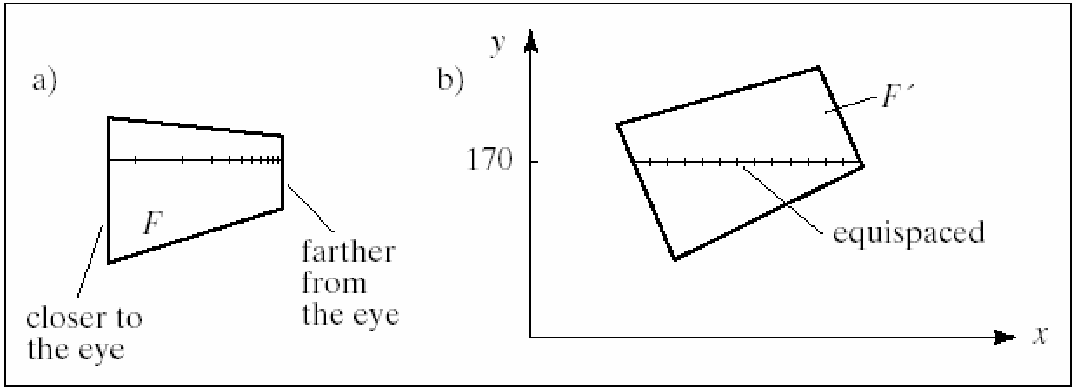
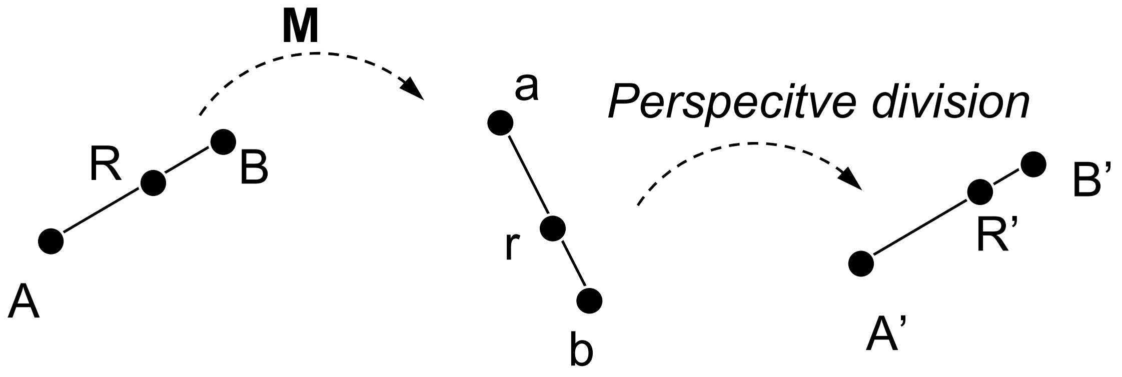




**FIGURE 8.41** Spacing of samples with linear interpolation.

# Inbetween points

***How do points on lines transform?***




R(g) = (1-g)A + gB

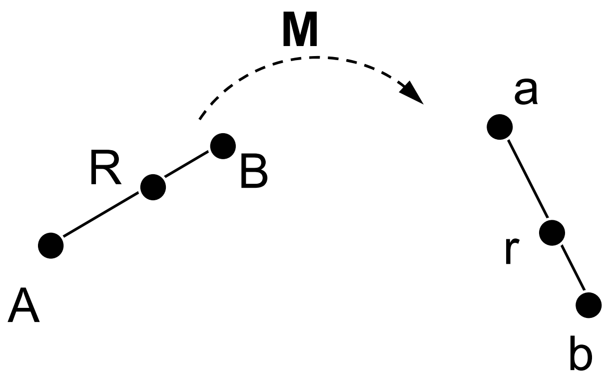
r = MR

R'(f) = (1-f)a' + fb' in cartesian coordinates

What is the relationship between g and f?

# First step

## *World to homogeneous space (4D)*




$$R = (1-g)A + gB$$

$$r = MR = M[(1-g)A + gB] = (1-g)MA + gMB \Rightarrow$$

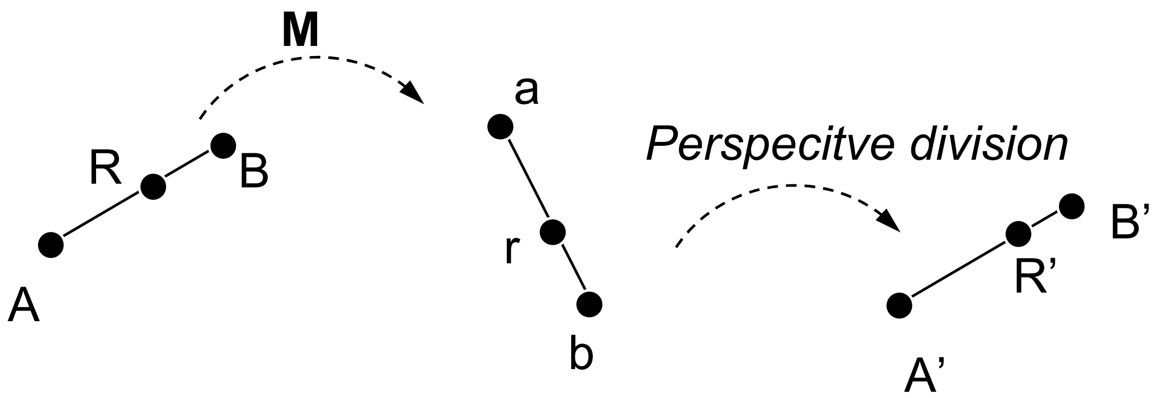
$$r = (1-g)a + gb$$

$$a = MA = (a_1, a_2, a_3, a_4)$$

$$b = MB = (b_1, b_2, b_3, b_4)$$

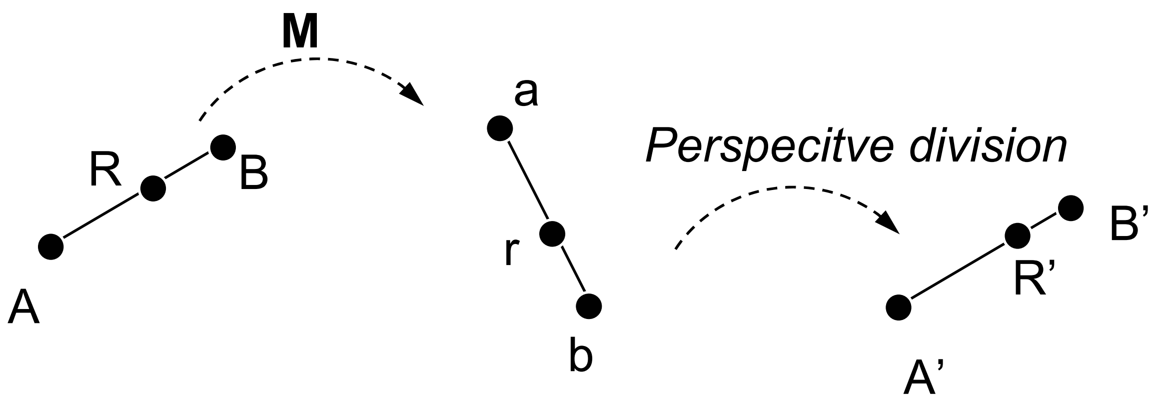
# Second step

## *Perspective division*




$$\left\{ \begin{array}{l} r = (1-g)a + gb \\ a = (a_1, a_2, a_3, a_4) \\ b = (b_1, b_2, b_3, b_4) \end{array} \right\} \Rightarrow \boxed{R'_1 = \frac{r_1}{r_4} = \frac{(1-g)a_1 + gb_1}{(1-g)a_4 + gb_4}}$$

# Putting all together




$$R'_1 = \frac{(1-g)a_1 + gb_1}{(1-g)a_4 + gb_4} = \frac{lerp(a_1, b_1, g)}{lerp(a_4, b_4, g)}$$

$$At \;\; the \;\; same \;\; time:$$

$$R' = (1-f)A' + fB' \Rightarrow$$

$$R'_1 = (1-f)\frac{a_1}{a_4} + f\frac{b_1}{b_4} = lerp(\frac{a_1}{a_4}, \frac{b_1}{b_4}, f)$$

# Relation between the fractions

$$\left.\begin{aligned} R'1(f) &= \frac{lerp(a1,b1,g)}{lerp(a4,b4,g)} \\ R'1(f) &= lerp\left(\frac{a1}{a4},\frac{b1}{b4},f\right) \end{aligned}\right\} \Rightarrow g = \frac{f}{lerp(\frac{b4}{a4},1,f)}$$

substituting this in $R(g) = (1-g)A + gB$ yields

$$R1 = \frac{lerp(\frac{A1}{a4},\frac{B1}{b4},f)}{lerp(\frac{1}{a4},\frac{1}{b4},f)}$$

**THAT MEANS**: For a given f in **screen space** and A,B in **world space** we can find the corresponding R (or g) in **world space** using the above formula.

“A” can be texture coordinates, position, color, normal etc.

# Rendering images incrementally

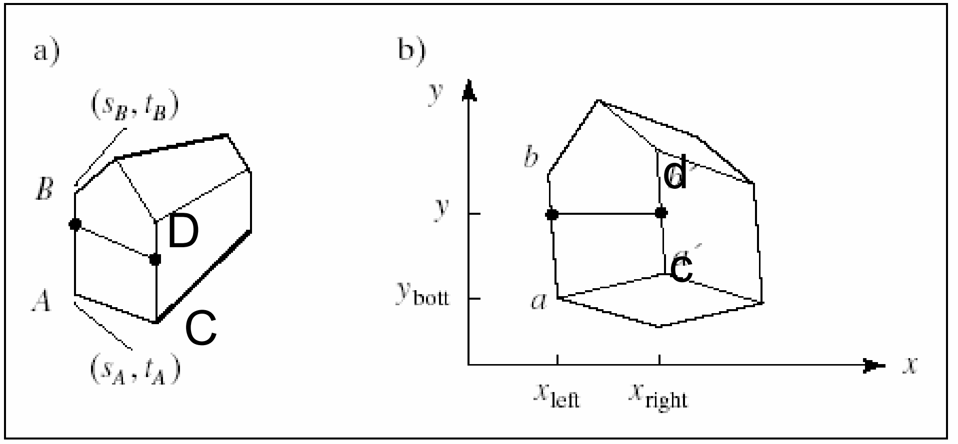
*A maps to a (homogeneous)*

*B maps to b*

*C maps to c*

*D maps to d*

*For scanline y:*




$$f = (y - y_{bott})/(y_{top} - y_{bott})$$

$$s_{left}(y) = \frac{lerp(\frac{s_A}{a_4}, \frac{s_B}{b_4}, f_l)}{lerp(\frac{1}{a_4}, \frac{1}{b_4}, f_l)}, s_{right}(y) = \frac{lerp(\frac{s_C}{c_4}, \frac{s_D}{d_4}, f_r)}{lerp(\frac{1}{c_4}, \frac{1}{d_4}, f_r)}$$

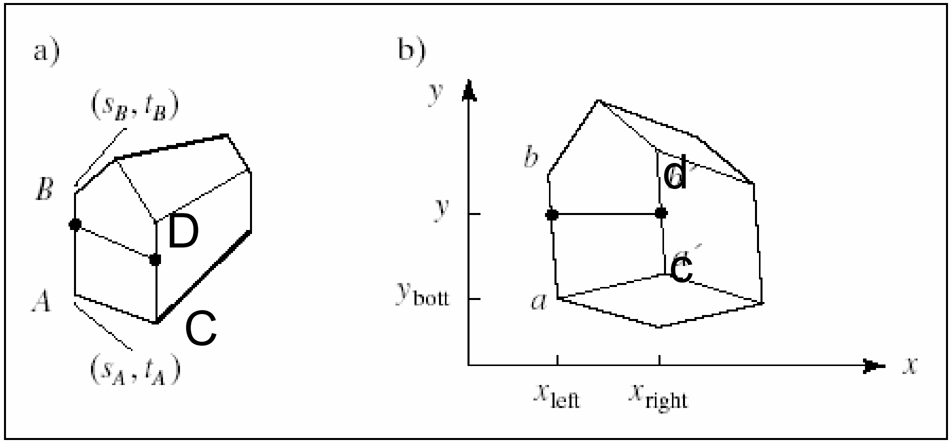
*Once we have $s_{left}$ and $s_{right}$ another hyperbolic interpolation fills in the scanline*

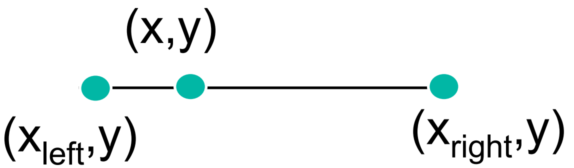
# Interpolation along the scanline

$$s_{left}(y) = \frac{lerp(\frac{s_A}{a_4}, \frac{s_B}{b_4}, f_l)}{lerp(\frac{1}{a_4}, \frac{1}{b_4}, f_l)},$$

$$s_{right}(y) = \frac{lerp(\frac{s_C}{c_4}, \frac{s_D}{d_4}, f_r)}{lerp(\frac{1}{c_4}, \frac{1}{d_4}, f_r)}$$

$$s(x, y) = \frac{lerp(\frac{s_{left}}{h_{left}}, \frac{s_{right}}{h_{right}}, f)}{lerp(\frac{1}{h_{left}}, \frac{1}{h_{right}}, f)}$$




What are the f, and h's?

# Interpolation along the scanline

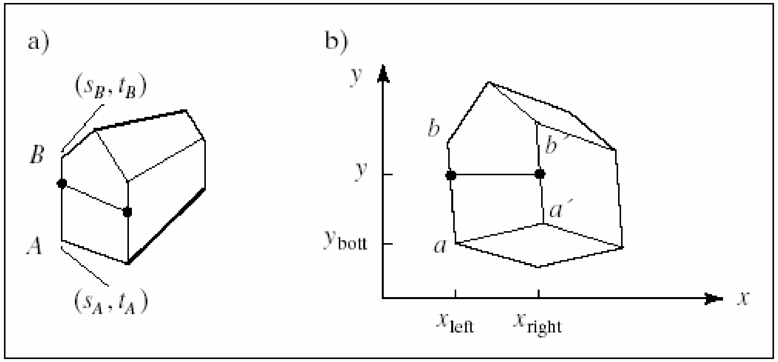
$$s_{left}(y) = \frac{lerp(\frac{s_A}{a_4}, \frac{s_B}{b_4}, f_l)}{lerp(\frac{1}{a_4}, \frac{1}{b_4}, f_l)}, s_{right}(y) = \frac{lerp(\frac{s_C}{c_4}, \frac{s_D}{d_4}, f_r)}{lerp(\frac{1}{c_4}, \frac{1}{d_4}, f_r)}$$

$$s(x, y) = \frac{lerp(\frac{s_{left}}{h_{left}}, \frac{s_{right}}{h_{right}}, f)}{lerp(\frac{1}{h_{left}}, \frac{1}{h_{right}}, f)}$$

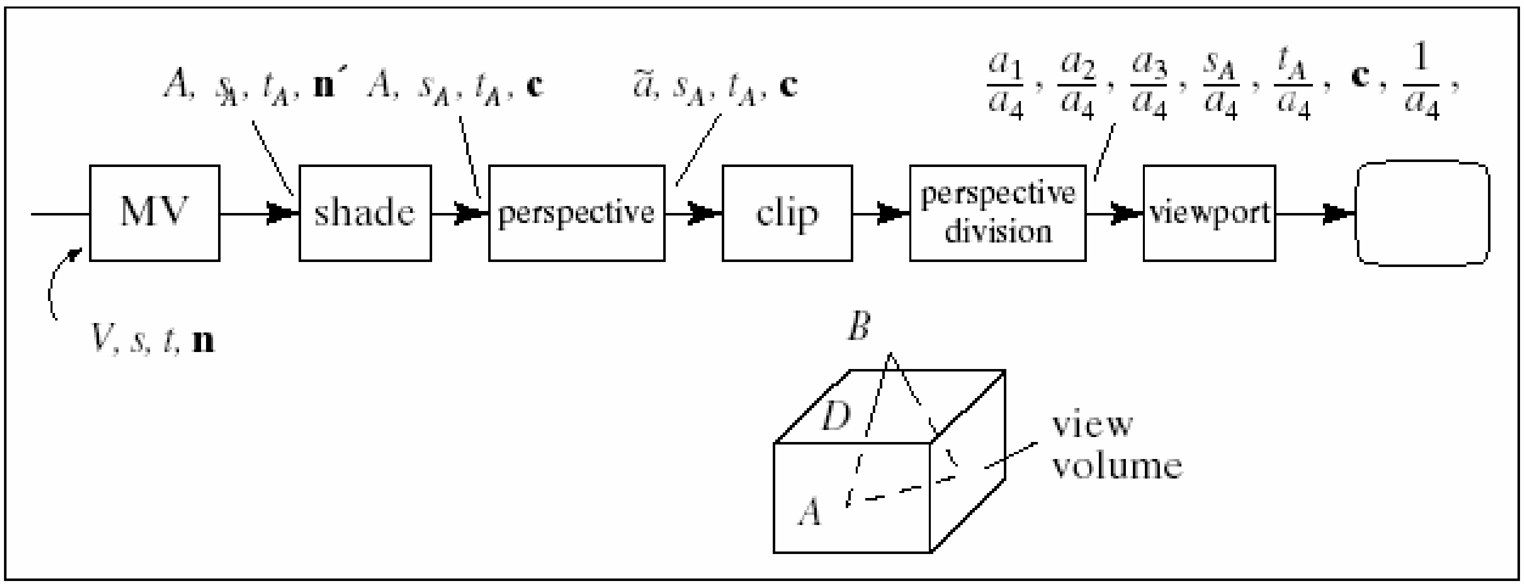
$$h_{left} = lerp(a_4, b_4, f_l)$$

$$h_{right} = lerp(c_4, d_4, f_r)$$

$$f = (x - x_{left})/(x_{right} - x_{left})$$



# Pipeline with hyperbolic interpolation



# What does the texture do?

## *Replace*

- Ir = texture_r(s,t)
- glTexEnvf(Gl_TEXTURE_ENV,GL_TEXTURE_ENV_MODE, GL_DECAL) ;

## *Modulate*

- I = texture(s,t)[Iaka + Idkd x lambert] + Isks x phong
- glTexEnvf(Gl_TEXTURE_ENV,GL_TEXTURE_ENV_MODE, GL_MODULATE) ;

# Bump Mapping

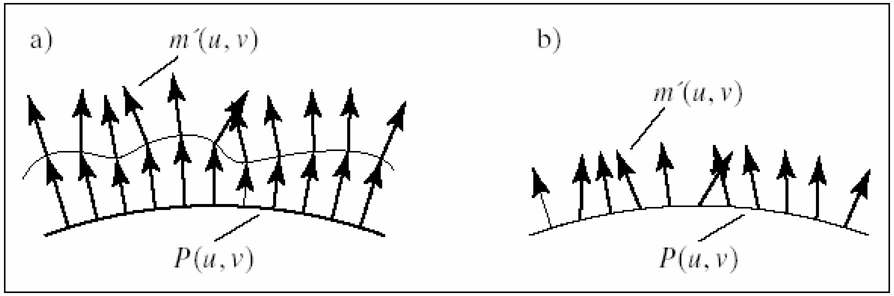



**FIGURE 8.50** On the nature of bump mapping.

P’(u,v) = P(u,v) + texture(u,v)m(u,v)

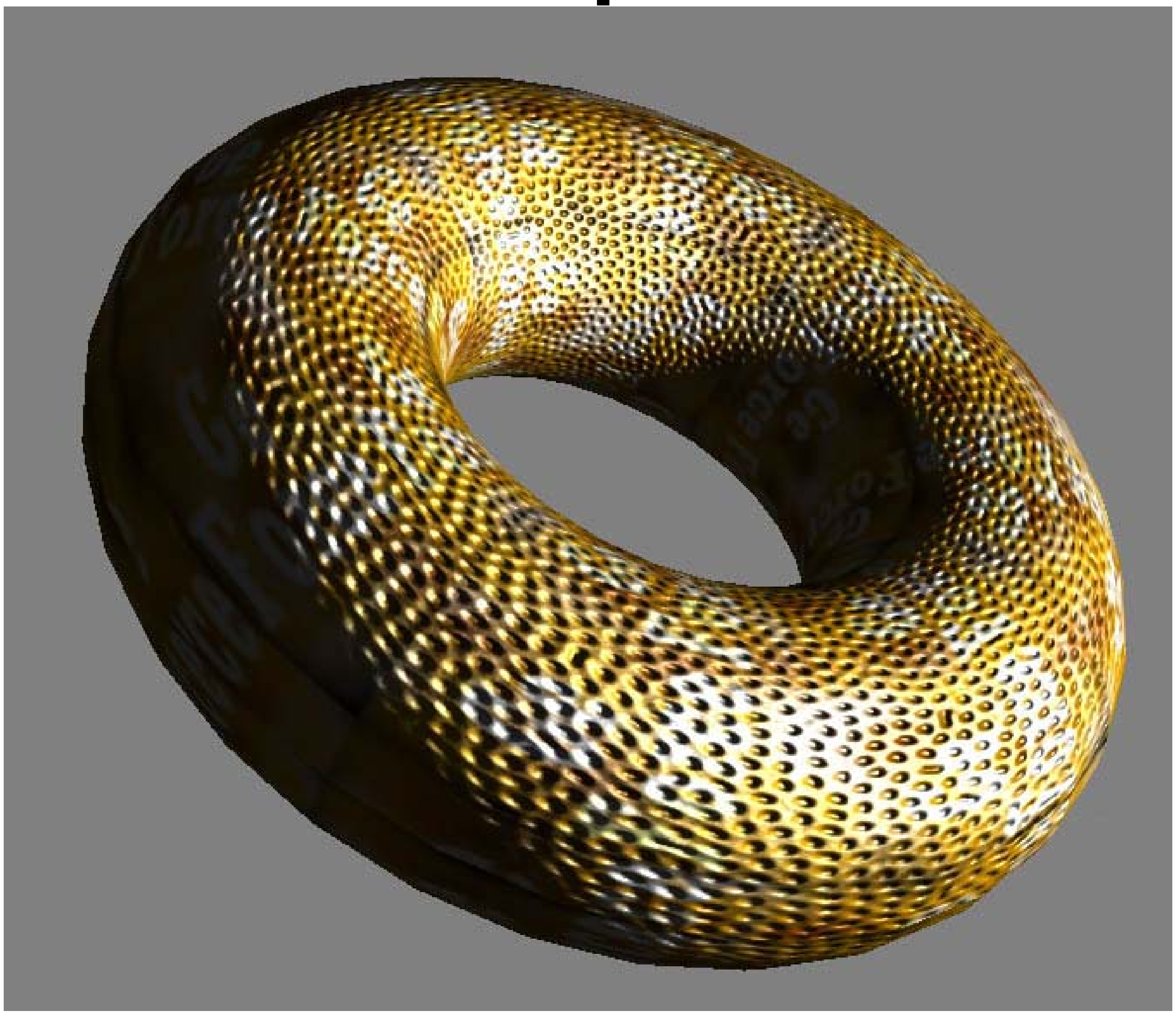
Approximation by Blinn

m’(u,v) = m(u,v) + [(m x P_v) texture_u – (m x P_u) texture_v]

Where _ indicates partial derivative.

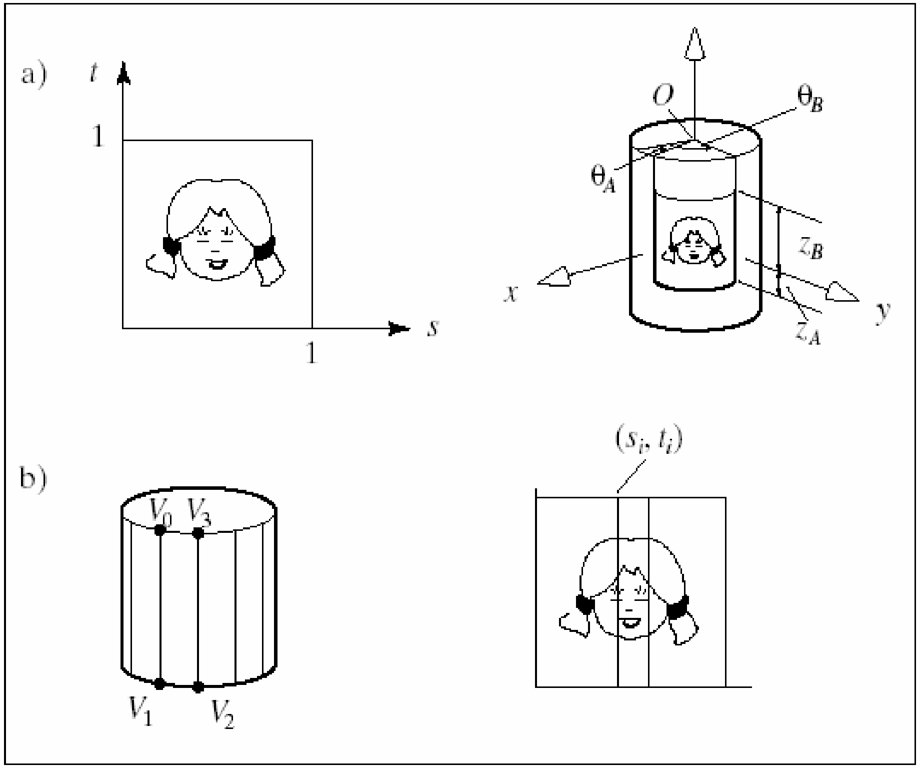
All functions evaluated at (u,v).

# Example



# Calculating texture coordinates

# Wrapping textures on curved surfaces




$$s = \frac{\theta - \theta_a}{\theta_b - \theta_a}, t = \frac{z - z_a}{z_b - z_a}$$

Cylinder with N faces

Left edge at azimuth $\theta = 2\pi i / N$

Upper left vertex texture coordinates $s_i = \dfrac{2\pi i / N - \theta_a}{\theta_b - \theta_a}, t_i = 1.$

# Automatic calculation of Texture Coordinates

```
glEnable(GL_TEXTURE_GEN_S);

glEnable(GL_TEXTURE_GEN_T);


glTexGeni(GL_S, GL_TEXTURE_GEN_MODE,
    GL_OBJECT_LINEAR);

Glfloat coeff[4] = {1,0,0,0} ;

glTexGenfv(GL_S, GL_OBJECT_PLANE, coeff);
```

Same for T

# GL_OBJECT_LINEAR

***Linear combination of coordinates***

S or T = p0*x + p1*y+p2*z+p4*w

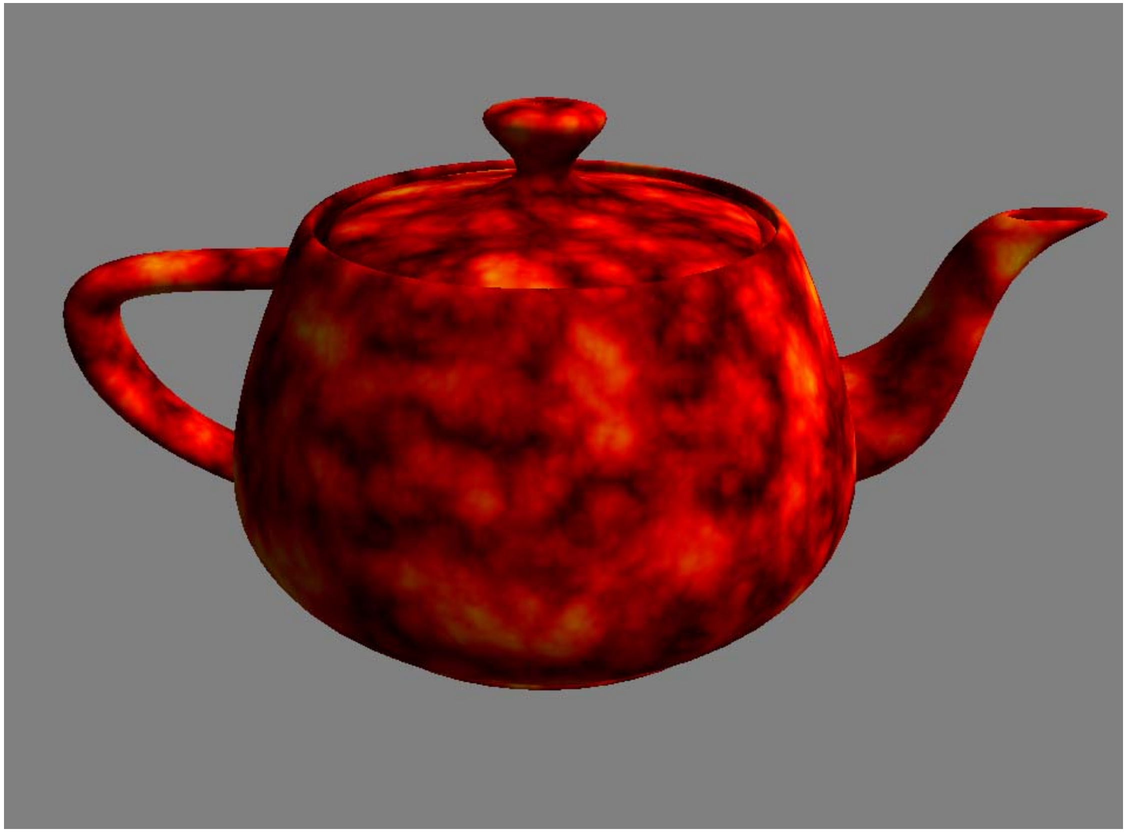
If (p0,p1,p2,p3) correctly normalized then
distance to plane (p0,p1,p2,p3)

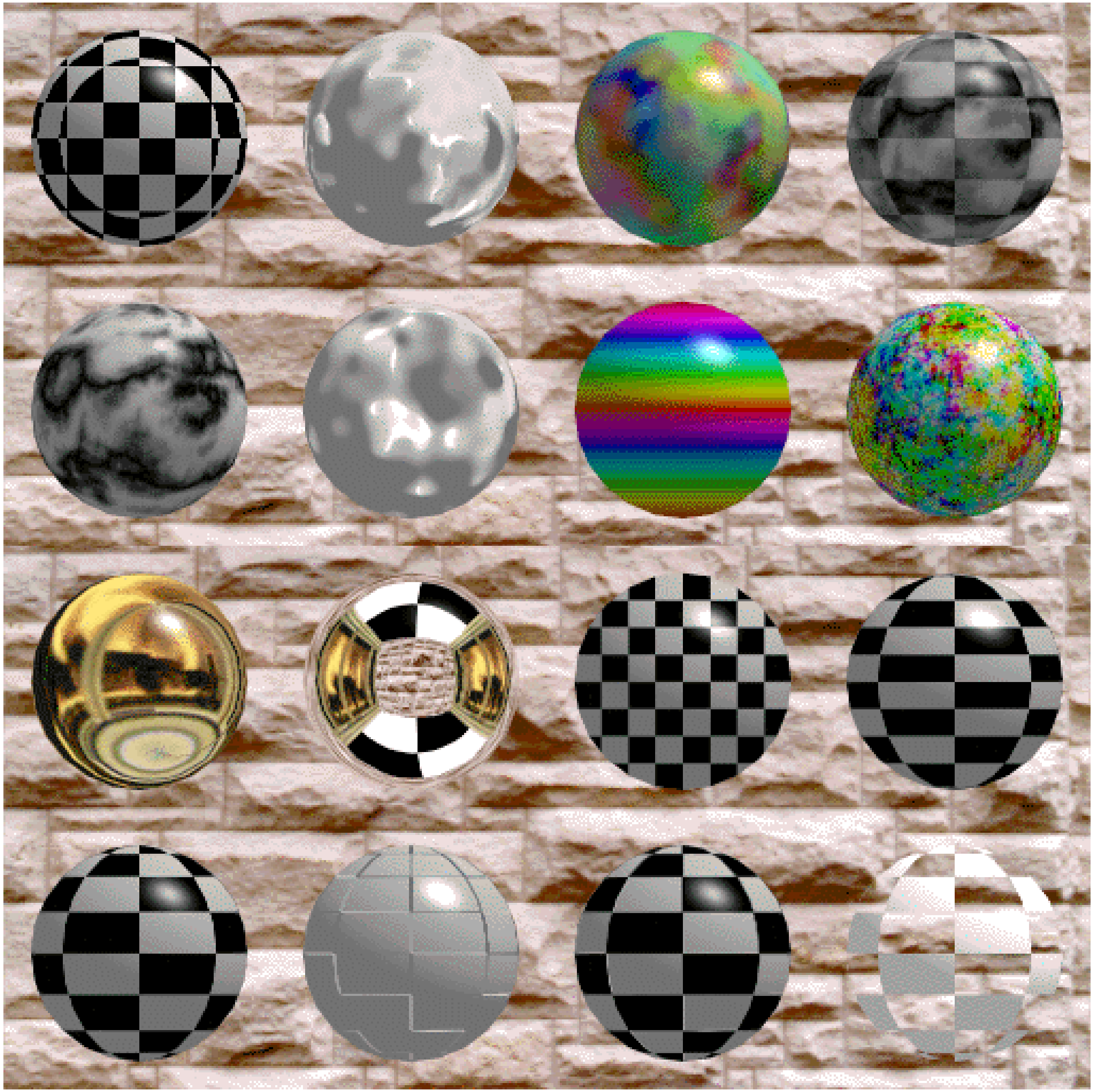
E.g. (1,0,0,0) distance from plane x = 0.

# Procedural texture

Volumetric textures

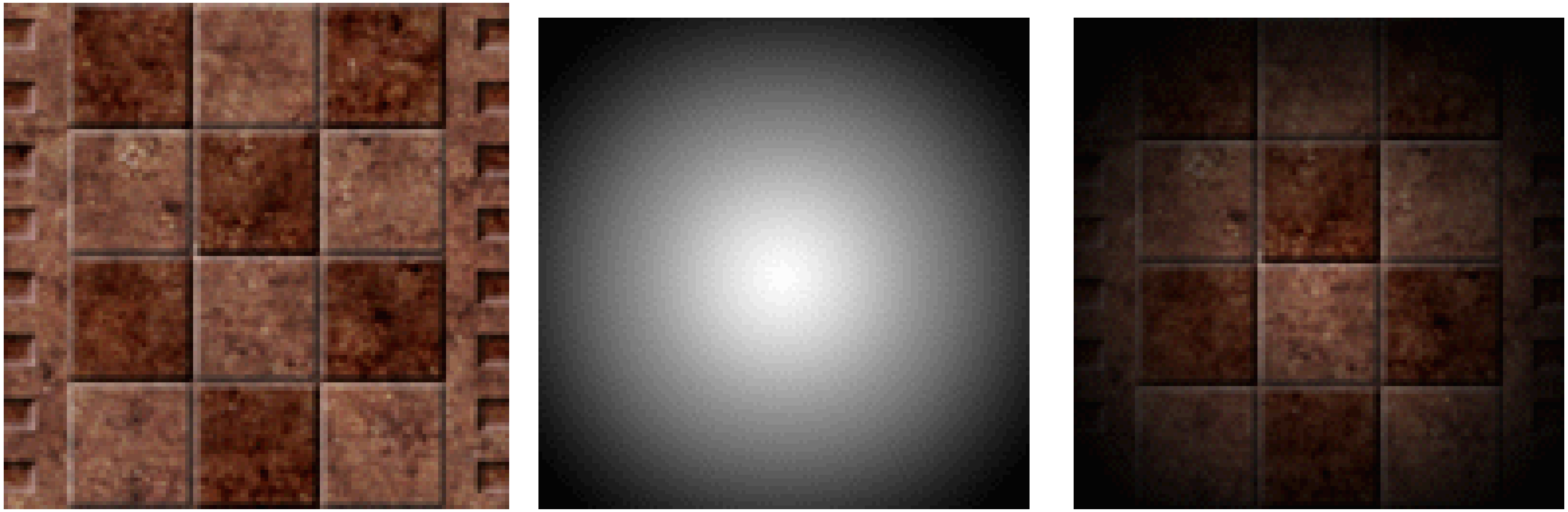
C = B(x,y,z)





# Light maps

***Static objects***



# Texture filtering

***Texture images consist of pixels (texels).***

***Therefore:***

- *Maginfication*: a pixel on the screen may cover only part of a texel.
- *Minification*: a pixel on the screen may cover more than one texels.

***Solution: Filtering***

# Texture filtering in OpenGL

```
glTexParametei(GL_TEXTURE_2D,
     GL_TEXTURE_MAG_FILTER, GL_NEAREST) ;

glTexParametei(GL_TEXTURE_2D,
     GL_TEXTURE_MIN_FILTER, GL_NEAREST) ;


GL_TEXTURE_MAG_FILTER: GL_NEAREST or GL_LINEAR

GL_TEXTURE_MIN_FILTER: GL_NEAREST, GL_LINEAR,
                    GL_NEAREST_MIPMAP_NEAREST,
                    GL_LINEAR_MIPMAP_NEAREST,
                    GL_LINEAR_MIPMAP_LINEAR,
```

# Texture mapping in OpenGL

```
void glTexImage2D(
   GLenum target,        // must be GL_TEXTURE_2D
   GLint level,
   GLint internalformat,    // e.g. 3
   GLsizei width,
   GLsizei height,
   GLint border,
   GLenum format,        // e.g. GL_RGB
   GLenum type,          // e.g. GL_UNSIGNED_BYTE
   const GLvoid *pixels     // size powers of 2 !!
 );
```

# Texture Parameters

```
void glTexParameterf(
    GLenum target,        // e.g. GL_TEXTURE_2D
    GLenum pname,         // GL_WRAP_S
    GLfloat param         // value e.g. GL_CLAMP
  );
```

# Texture effect

```
void glTexEnvf(
    GLenum target,       // GL_TEXTURE_ENV
    GLenum pname,        // GL_TEXTURE_ENV_MODE
    GLfloat param        // GL_MODULATE, GL_DECAL,
                         // and GL_BLEND
  );
```

# Enabling texture mapping

```
glEnable(GL_TEXTURE_2D) ;

glDisable(GL_TEXTURE_2D) ;
```

# Texture mapping example

```
void initTexture(void){
   glTexImage2D(GL_TEXTURE_2D, 0, 3, Img.m_width,
         Img.m_height, 0, GL_RGB, GL_UNSIGNED_BYTE,
         Img.m_data);
   glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
      GL_REPEAT);
   glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
      GL_REPEAT);
   glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_NEA
   REST);
   glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,
   GL_NEAREST);
   glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
   GL_DECAL);
   glEnable(GL_TEXTURE_2D);

}
```

# Texture Objects

***Copying an image from main memory to video memory is very expensive*** **(`glTexImage2D`).**

- *Create texture names*
- *Bind (create) texture objects to texture data:*
  - Image arrays + texture properties
- *Bind and rebind texture objects.*

# Naming texture objects

```
void glGenTextures(GLsizei n, GLuint
  *textureNames) ;
```

Returns `n` unused names `textureNames[0]…[n])`.

```
GLboolean glIsTexture(GLuint textureName) ;
```

# Creating Texture Objects

```
glBindTexture(Glenum target, GLuint
textureName) ;
```

***Three things:***

- If textureName > 0 and not already assigned a new texture object is created.
- If textureName assigned, the textureObject becomes active.
- If textureName is 0 OpenGL stops using textures and returns to the default unnamed texture.

# Initial creation

```
glBindTexture(Glenum target, GLuint
textureName) ;
```

Target:

```
GL_TEXTURE_1D, GL_TEXTURE_2D , GL_TEXTURE_3D,
Gl_TEXTURE_CUBE_MAP.
```

# Example: Creating the textures

```
void init(void) {
       glGenTextures(2,texName) ;
       glBindTexture(GL_TEXTURE_2D, texName[0]) ;
       glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
       GL_CLAMP);
       glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
       GL_CLAMP);
       glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
       GL_NEAREST);
       glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
       GL_NEAREST);
       glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA,
              GL_UNSIGNED_BYTE, image1) ;

       glBindTexture(GL_TEXTURE_2D, texName[1]) ;
       …
       glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA,
              GL_UNSIGNED_BYTE, image2) ;

}
```

# Example: Using the textures

```
void display(void) {
        …
    glBindTexture(GL_TEXURE_2D, texName[0]) ;
    glBegin(GL_QUADS) ;
    glTexCoord2f(0.0,0.0) ;
        …
    glEnd() ;

    glBindTexture(GL_TEXURE_2D, texName[1]) ;
    glBegin(GL_QUADS) ;
    glTexCoord2f(0.0,0.0) ;
        …
    glEnd() ;
}
```