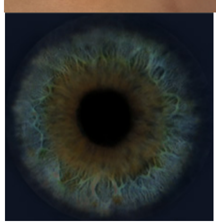
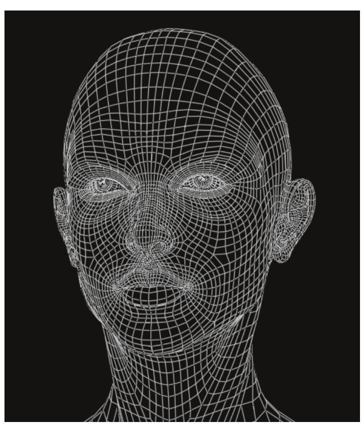
Texture Mapping

Pasting textures (2D images) on surfaces





Conceptual Steps Involved

Texture to Object Mapping

- User defines where the texture maps onto an object's surface
- In our pipeline this happens at the vertex level

Texture to Screen Mapping (through object)

- The rendering system has to project the texture in some way to the screen

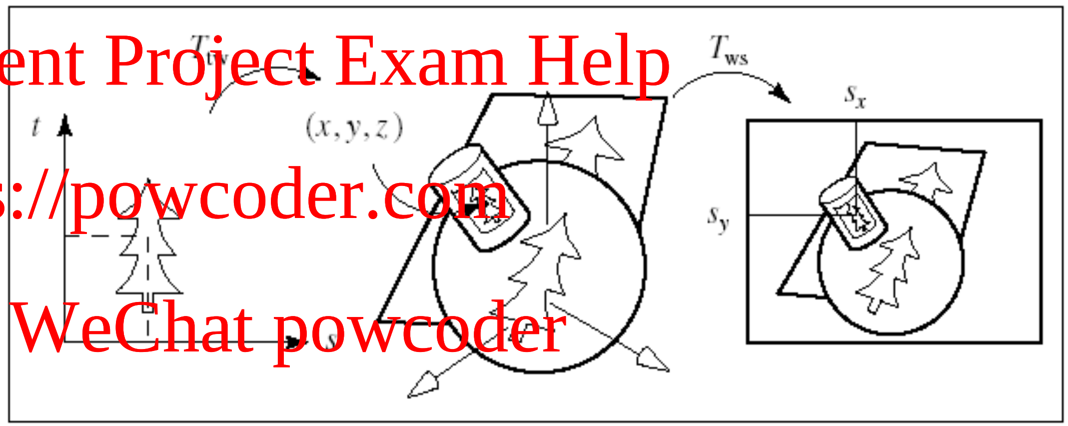
- That is, each pixel on the screen has to get the right piece of the texture

- With programmable OpenGL we can manipulate textures at the fragment level, i.e during the second step

The two steps as coordinate system transformations

User Defined Viewing+Projection

FIGURE 8.35 Drawing texture on several objects of different shape.



Systems and transformations involved

- (s,t) : 2D Texture space
- (s_x,s_y) : 2D Screen space
- ${}_sT_w$: world to screen (viewing and projection)
- ${}_wT_t$: texture to world

$$(s_x,s_y) = {}_sT_w({}_wT_t(s,t))$$

Approach one: Texture to Screen

$(s_x,s_y) = {}_sT_w({}_wT_t(s,t))$

For each pixel covered by the texture we would have to calculate coverage (overlap)

Approach two: Screen to texture

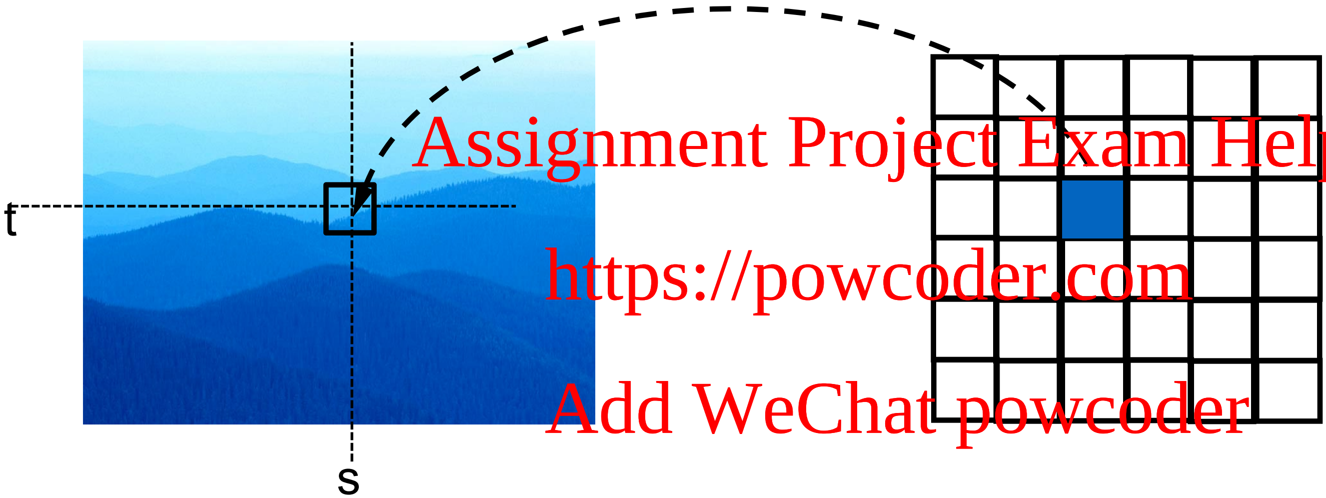
$(s,t) = {}_tT_w({}_wT_s(s_x,s_y))$

For each texel covered by the pixel's projection we would have to compute coverage

We also need to invert the projection process

OpenGL Approach: Screen to texture

In programmable OpenGL



For each fragment we compute texture coordinates with which we can fetch texels.

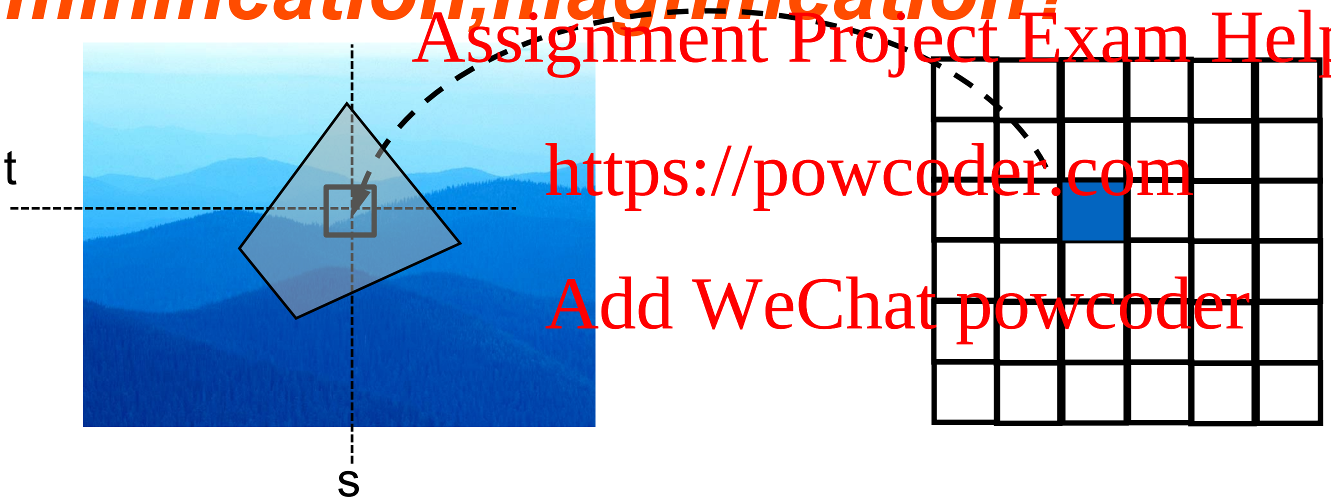
For each fragment we can fetch as many texels as we want (texture lookup). For simple cases, one texture lookup per pixel might be enough.

We keep texture coordinates per vertex and the rasterize interpolates them for intermediate pixels : (s,t)

Fetching and using a single texel corresponds to what?

Approach two: Screen to texture

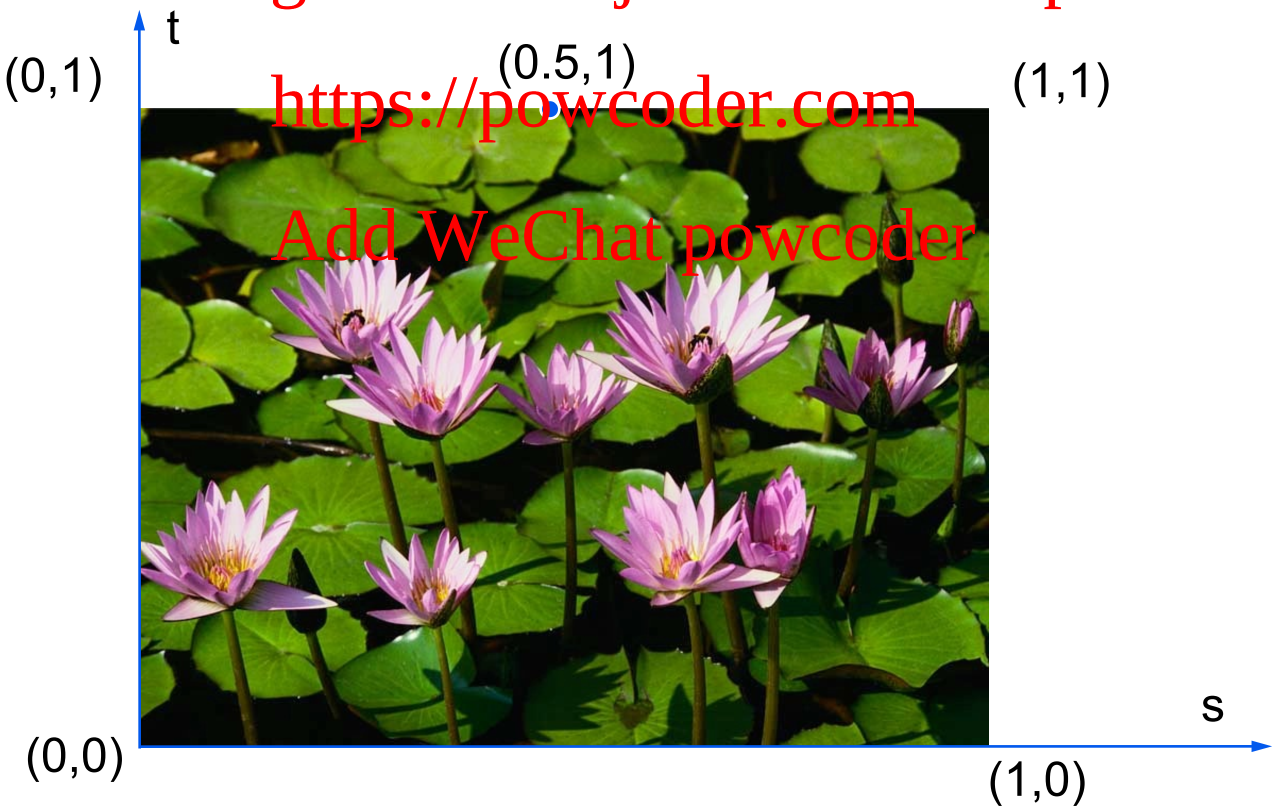
How do we address texture minification,magnification?



- Filtering, we will discuss it later

2D Textures image abstraction

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 correspondence between (s,t) and and some object coordinate system.

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

- By hand.

- Notice: we want to map a 2D image on the surface of the object
- Most often we need to parametrize the object in 2D

Mapping from texture to a 2D parametric form of the object space

Linear transformation

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

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

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

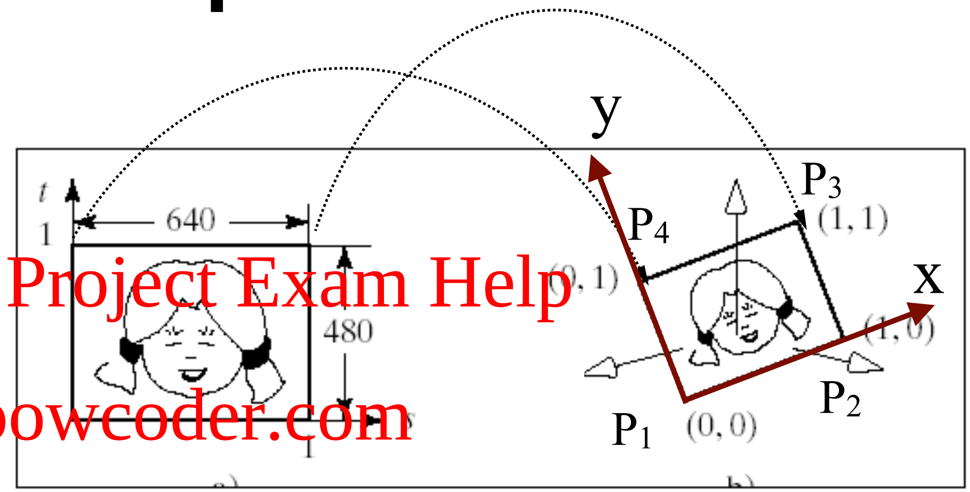
s in [0,1]

t in [0,1]

Example: Image to a quadrilateral

Chose a convenient object 2D system

- Origin P_1
- x axis P_2-P_1
- y axis P_4-P_1



Now parameterize it

A point P on the object in *u*-*v* coordinates is:

$$P_x(u) = P_{1x}(1-u) + P_{2x}u \quad = P_{1x} + u\,(P_{2x} - P_{1x})$$

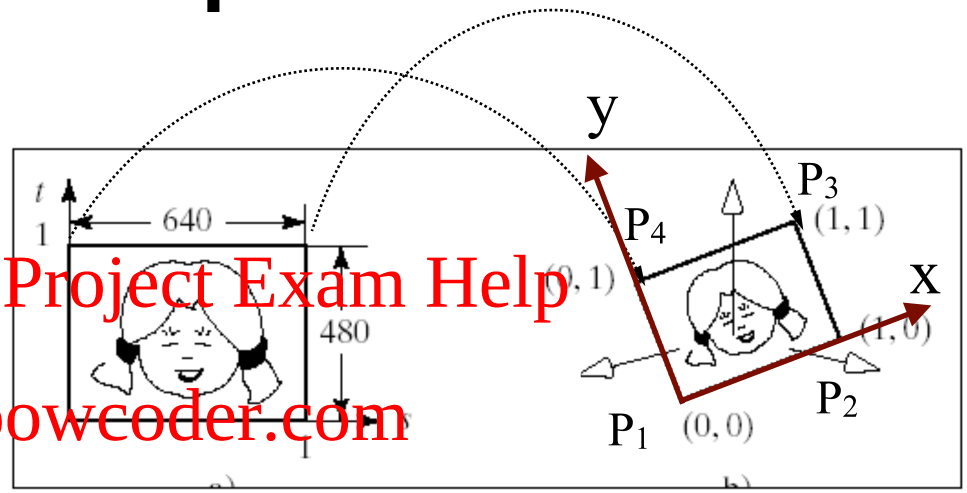
$$P_y(v) = P_{1y}(1-v) + P_{4y}v \quad = P_{1y} + v\,(P_{4y} - P_{1y})$$

with u,v in $[0,1]$

A couple of sanity checks (in (P_1, x,y) coordinate system) :

$$P(0,0) = P_1,\ \ P(1,1) \ = (P_{2x}, P_{4y}) = P_3\ ,\ P(1,0) = P_2,\ P(0,1) = P_4$$

Example: Image to a quadrilateral



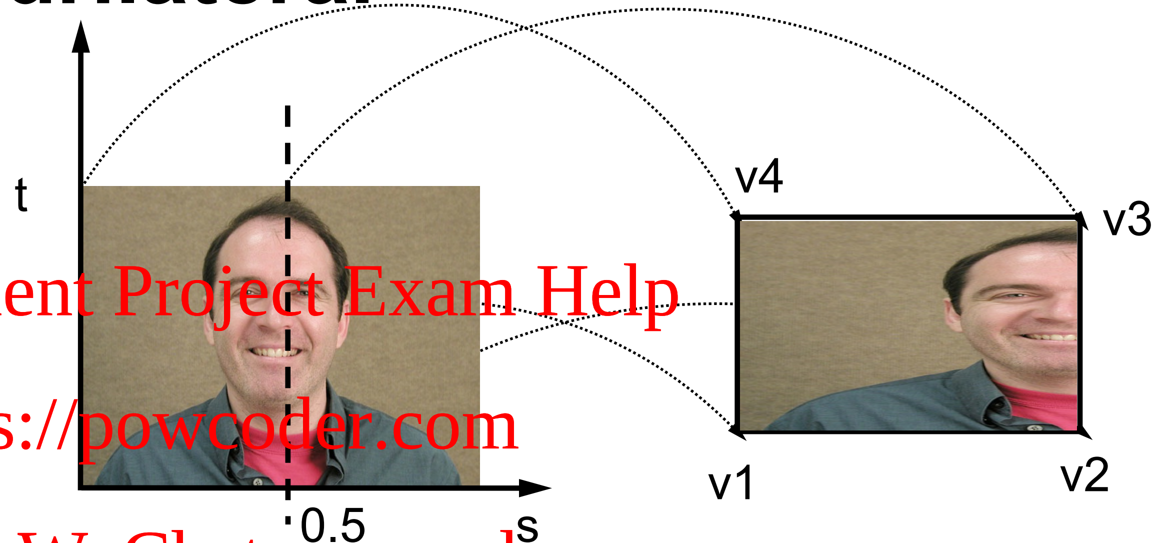
Then the mapping is simply

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

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

$$s,t \text{ in } [0,1]$$

Example 2: Piece of an image to a quadrilateral



Use only left part

$s = 0.5u$

$t = v$

u, v *in* $[0,1]$

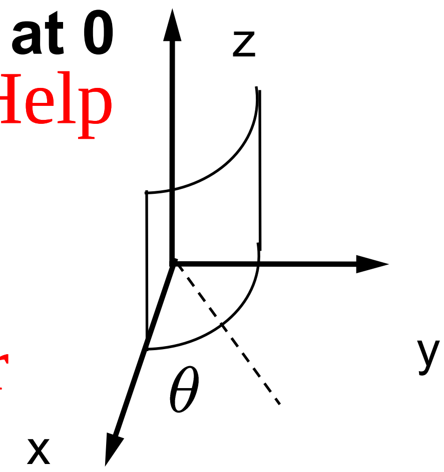
so for

v3 = P(u=1,v=1) the TexCoordinates are $(s,t) = (0.5,1)$

Advanced example: 2D Parameterization of a curve surface

Parameterizing the Cylinder

Cylinder has height h, radius r, centered at 0



2D parameterization of Object Space

Parametric form:

$$x = r cos\theta, \quad y = r sin\theta, \quad z$$

$$\theta \in [0, \pi/2], z \in [0, h]$$

Surface parameters:

$$u = \theta, v = z$$

with

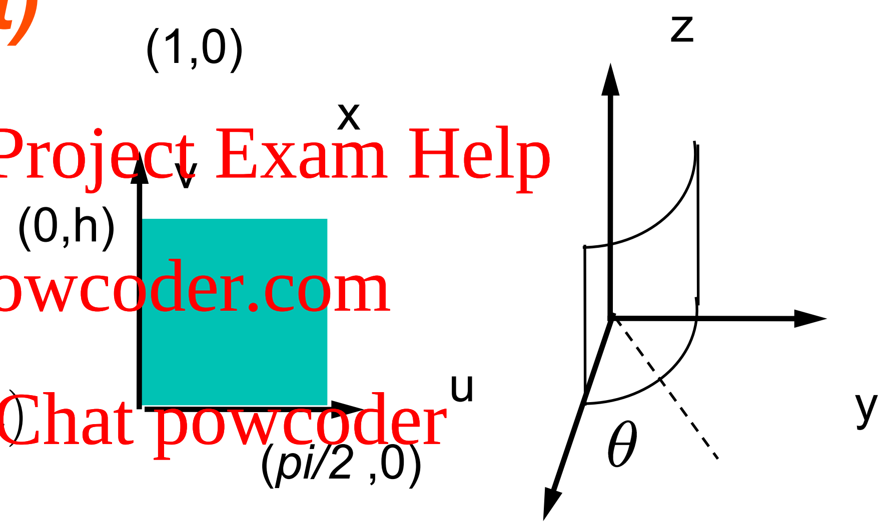
$$0 \le u \le \pi/2, \quad 0 \le v \le h$$

In (u,v) space quarter cylinder is a quad

Mapping a square texture to the quarter cylinder

Now map (u,v) to (s,t)

We chose



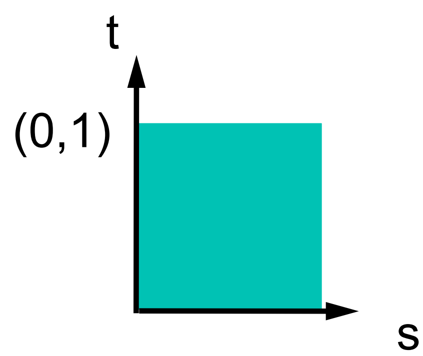
$$(s,t) = (0,0) \rightarrow (u,v) = (0,0)$$
$$(s,t) = (1,1) \rightarrow (u,v) = (\pi/2, h)$$

that is

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

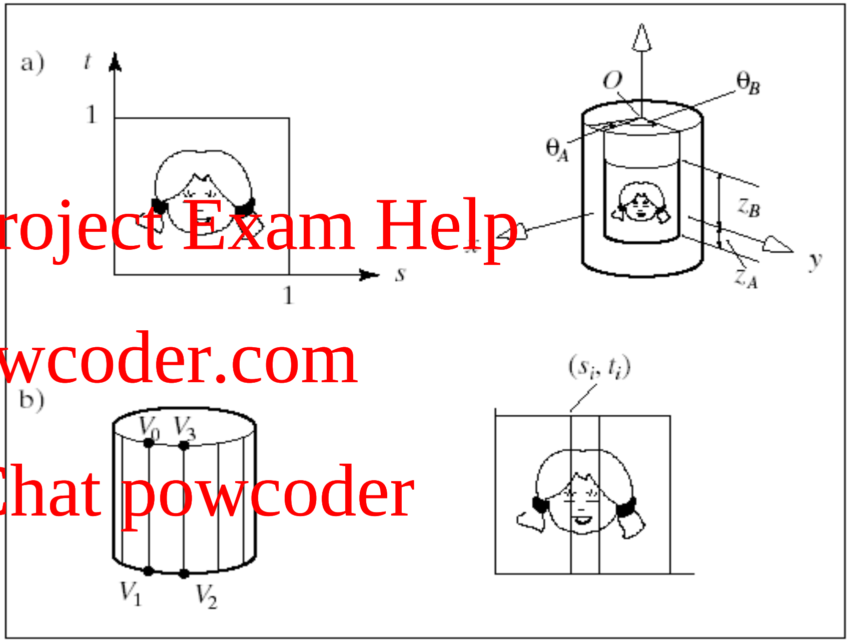
or inverted

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



Example: Wrapping textures on polygonal approximations of curved surfaces

However, we only have polygons in the graphics pipeline



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

Cylinder with N faces

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

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

How does that work with the graphics pipeline?

Only vertices go down the graphics pipeline.

Texture coordinates for interior points of polygons?

Calculate texture coordinates by interpolation along scanlines.

Rendering the texture

Scanline in screen space

- Generating s,t coordinates for each pixel

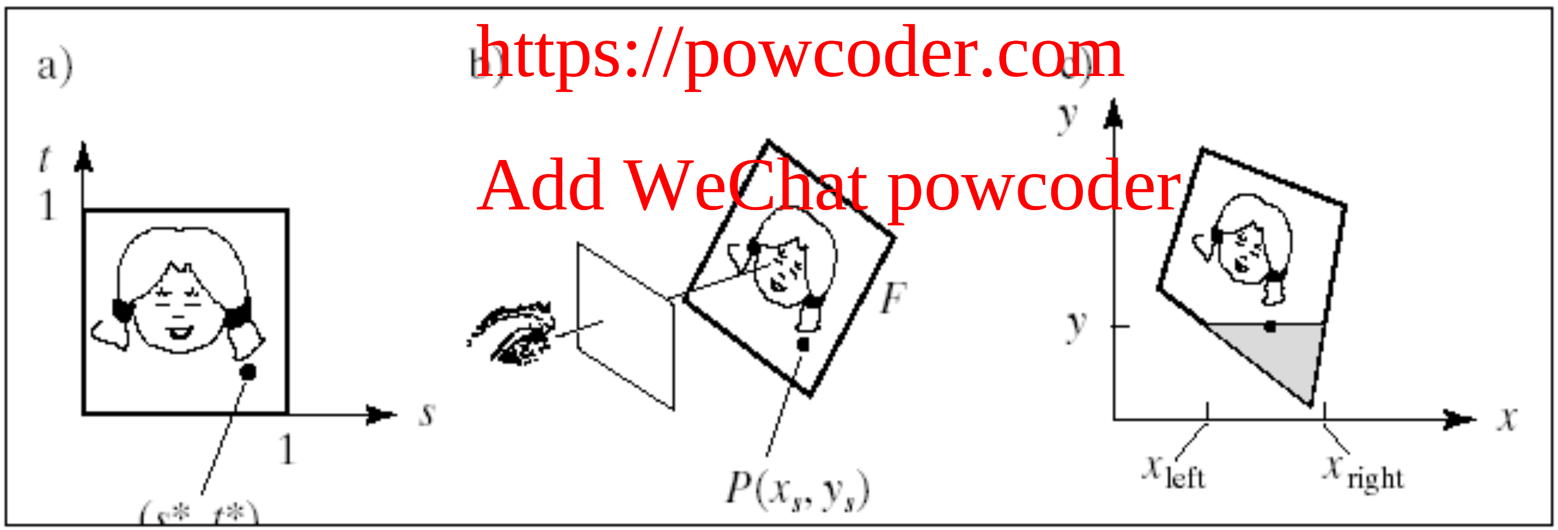


FIGURE 8.39 Rendering a face in a camera snapshot.

Interpolation of texture coordinates

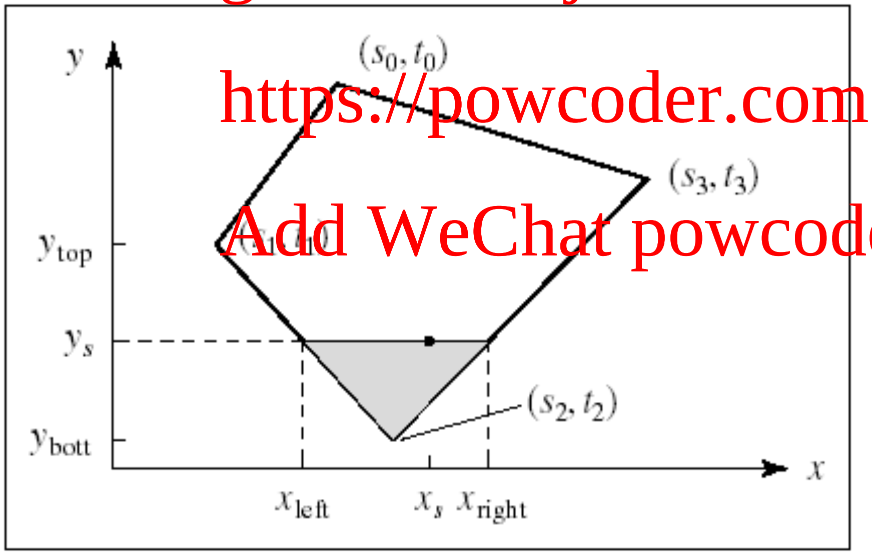
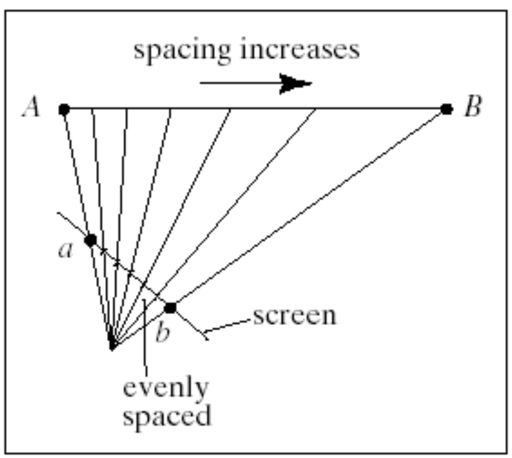


FIGURE 8.40 Incremental calculation of texture coordinates.

Problem

Perspective foreshortening

- Scanconversion takes equal steps along scanline (screen space), specifically by 1 i.e. x, x + 1, x+2 ,...
- Equal steps in screen space are not equal steps in world space



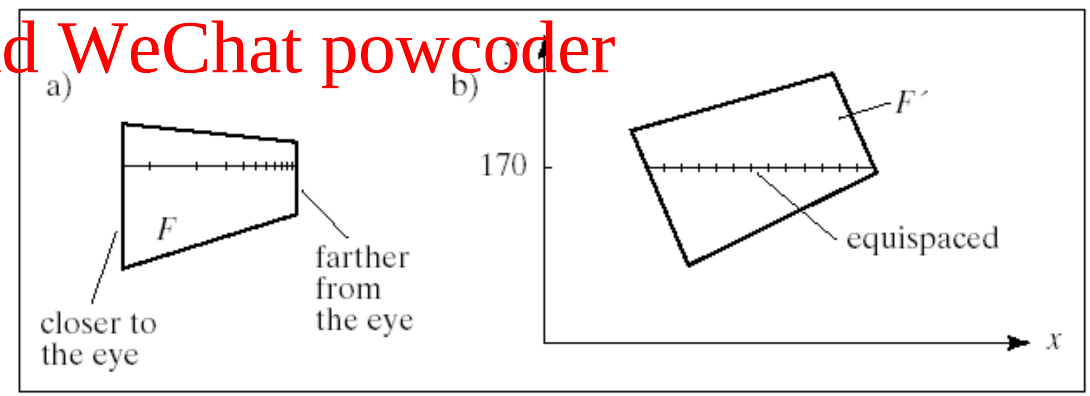
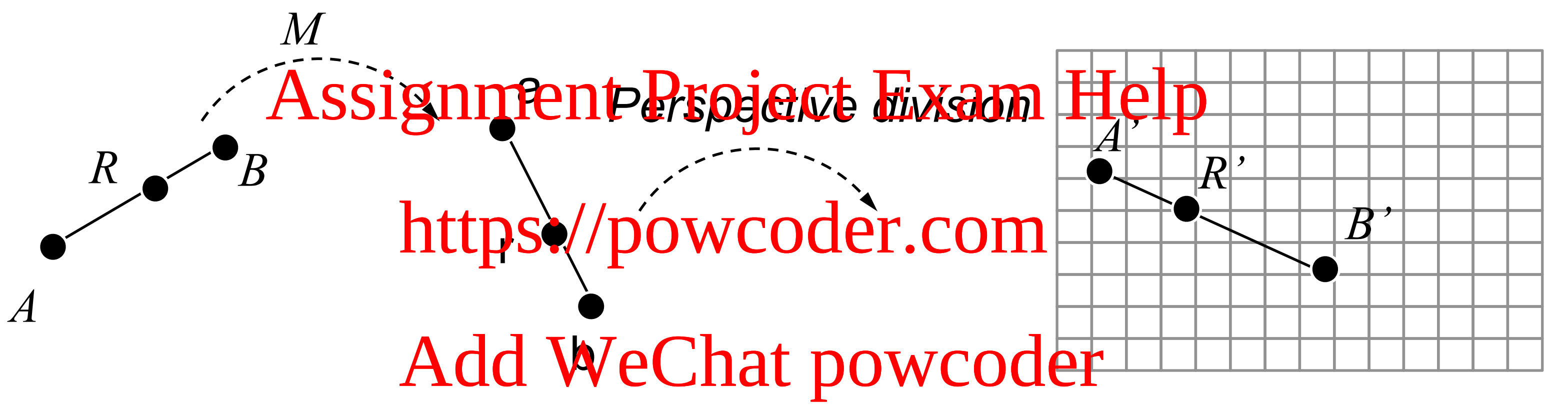


FIGURE 8.41 Spacing of samples with linear interpolation.

Reminder: Inbetween points



$R(g) = (1-g)A + gB, \quad g \in \Re$

$r = MR,$ where M is a WebGL perspective transformation

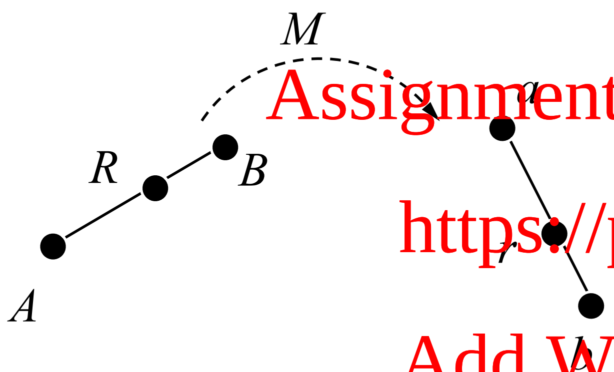
After perspective division (NDCS):

$R'(f) = (1-f)A' + fB', \quad f \in \Re$

How do g, f relate?

First step

Viewing to homogeneous space (4D)



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

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

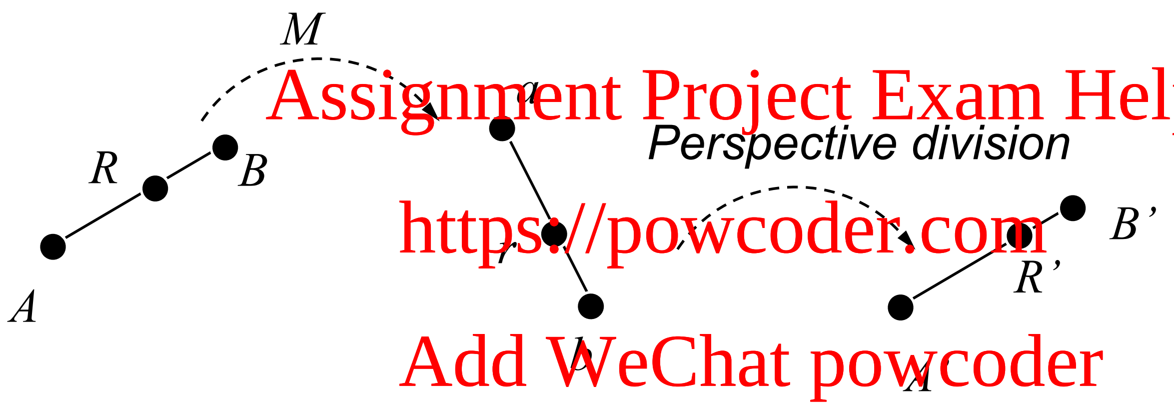
$$r = (1-g)a + gb$$

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

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

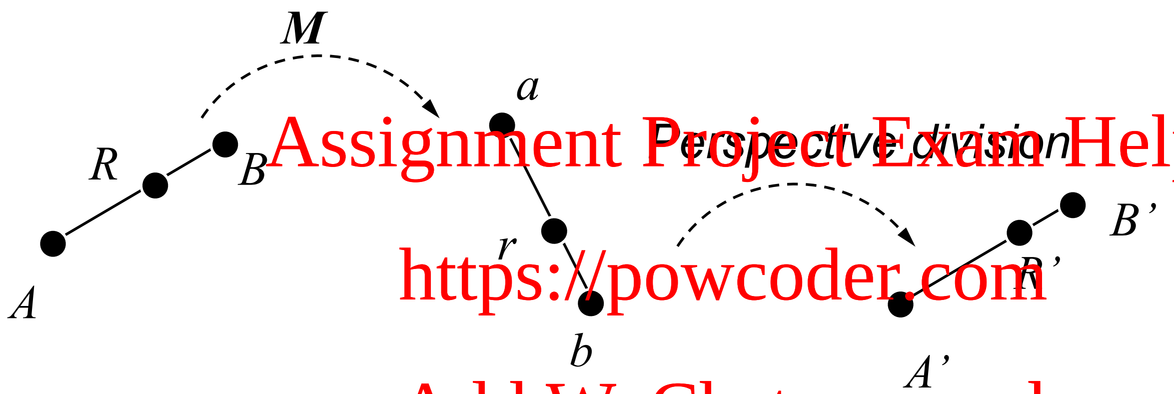
Second step

Perspective division



$$\left.\begin{aligned} r &= (1-g)a + gb \\ \\ r &= (r_1, r_2, r_3, r_4) \\ a &= (a_1, a_2, a_3, a_4) \\ b &= (b_1, b_2, b_3, b_4) \end{aligned}\right\} \rightarrow \boxed{R'_1 = \frac{r_1}{r_4} = \frac{(1-g)a_1 + gb_1}{(1-g)a_4 + gb_4}}$$

Putting all together



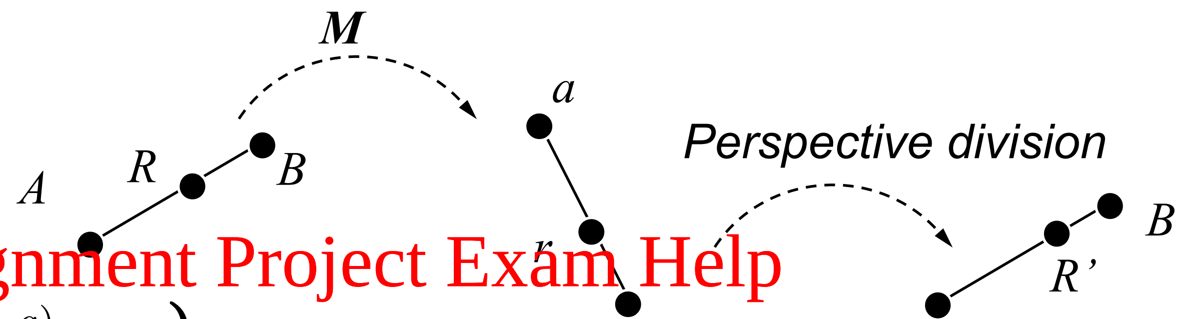
$$R'_1 = \frac{(1-g)a_1 + gb_1}{(1-g)a_4 + gb_4} = \frac{lerp(a_1, b_1, g)}{lerp(a_4, b_4, g)}$$

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

$$R' = (1-f)A' + fB' \Rightarrow R'_1 = (1-f)A'_1 + fB'_1$$

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

Relation between the fractions



$$\left.\begin{aligned} R_1'(g) &= \frac{lerp(a_1, b_1, g)}{lerp(a_4, b_4, g)} \\ R_1'(f) &= lerp\left(\frac{a_1}{a_4}, \frac{b_1}{b_4}, f\right) \end{aligned}\right\} \rightarrow g = \frac{f}{lerp\left(\frac{b_4}{a_4}, 1, f\right)}$$

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

$$R_1 = \frac{lerp\left(\frac{A_1}{a_4}, \frac{B_1}{b_4}, f\right)}{lerp\left(\frac{1}{a_4}, \frac{1}{b_4}, f\right)}$$

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

"A,B" can be texture coordinates, position, color, normal etc.

Rendering images incrementally

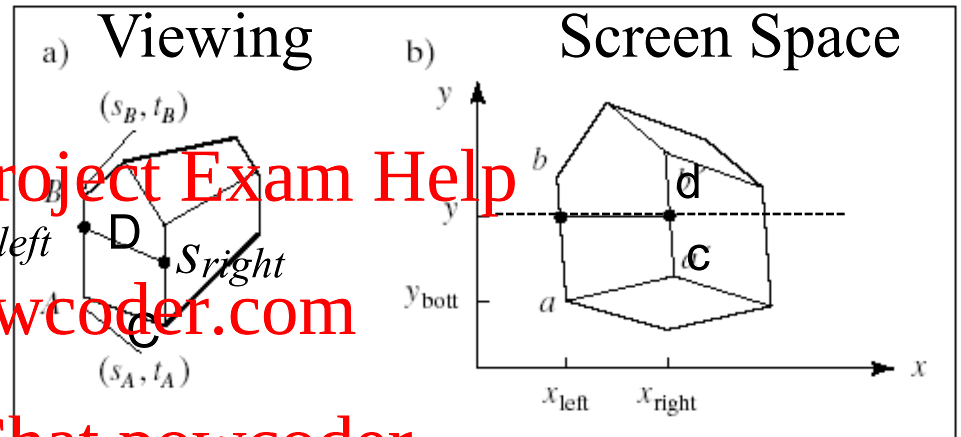
A maps to a (homogeneous)

B maps to b

C maps to c

D maps to d

For scanline y and two edges:



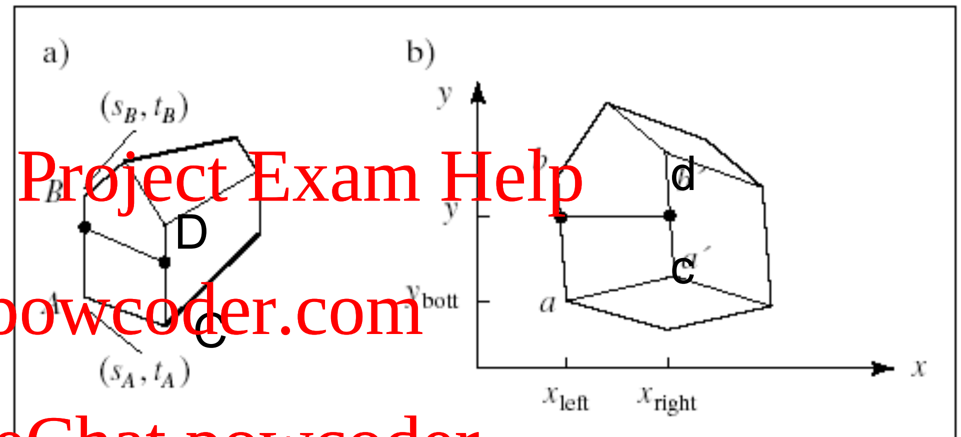
$$f_{edge} = (y - y_{bott}) / (y_{top} - y_{bott}) \text{ so for the left and right edges:}$$

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

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

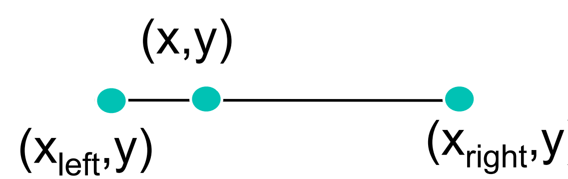
Interpolation along the scanline

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



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

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



What are the f, and h's?

Interpolation along the scanline

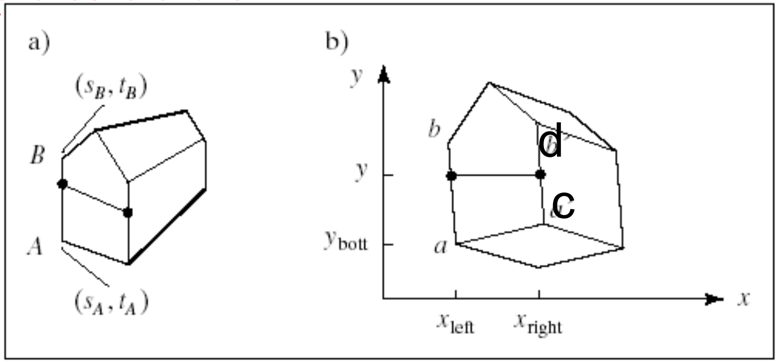
$$s_{left}(y)=\frac{lerp(\frac{s_A}{a_4},\frac{s_B}{b_4},f_l)}{lerp(\frac{1}{a_4},\frac{1}{b_4},f_l)},\; s_{right}(y)=\frac{lerp(\frac{s_C}{c_4},\frac{s_D}{d_4},f_r)}{lerp(\frac{1}{c_4},\frac{1}{d_4},f_r)}$$

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

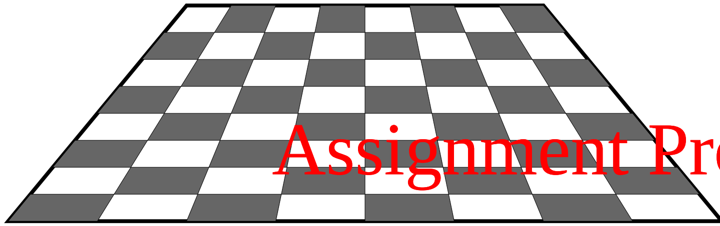
$$h_{left}=lerp(a_4,b_4,f_l)$$

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

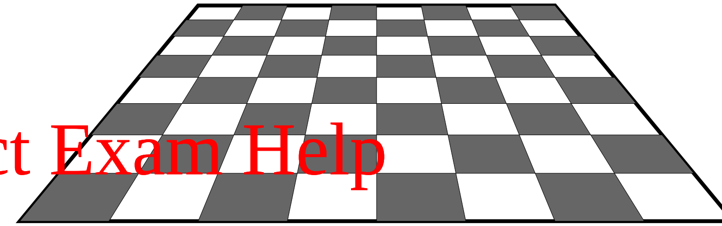
$$f=(x-x_{left})/(x_{right}-x_{left})$$



Example: Checkerboard image on a flat quad in the x-y plane



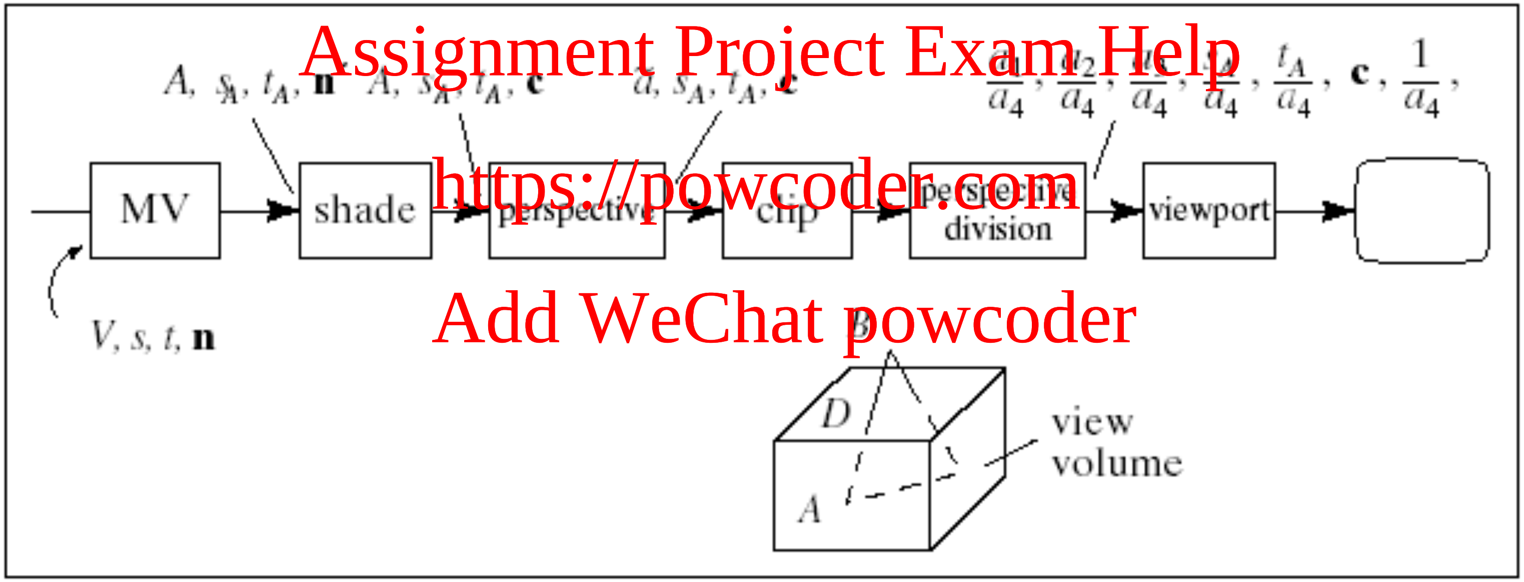
Without hyperbolic interpolation



With hyperbolic interpolation

- Left would be correct for a trapezoid that is parallel to the image plane
- You can think of it as follows:
  Linear interpolation pastes the image on the projected object, hyperbolic pastes the image on the object before the projection

Pipeline with hyperbolic interpolation



What does the texture do?

Textures are accessed in the fragment shader

- vec4 texColor = texture(texID, vec2(TexCoord)) ;

- and we can do what we like them (assign them as the fragment's color, blended them with other values, anything)

Texture mapping in OpenGL

Creating a texture

```
void glTexImage2D(
    GLenum target,          // must be GL_TEXTURE_2D
    GLint level,
    GLint internalformat,   // e.g. GL_RGB
    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 !!
  );
```

MUST MATCH!!

Need to load an image. Various libraries exist for that

Textures Have Many Parameters

Dealing with out of range tex coordinates

```
void gl.texParameterf(
    GLenum target,        // e.g. GL_TEXTURE_2D
    GLenum pname,         // GL_WRAP_S
    GLint param           // value e.g. GL_CLAMP
  );
```


e.g.

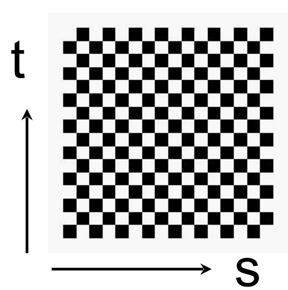
```
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S,
GL_REPEAT);
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T,
GL_REPEAT);
```

Wrapping Mode

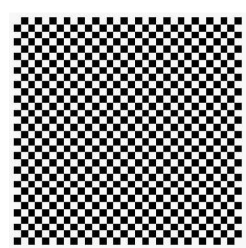
Clamping: if $s,t > 1$ use 1, if $s,t < 0$ use 0

Wrapping: use s,t modulo 1

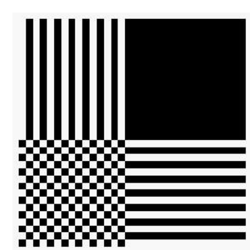
```
gl.texParameteri(gl.TEXTURE_2D,
        gl.TEXTURE_WRAP_S, gl.CLAMP )
gl.texParameteri( gl.TEXTURE_2D,
        gl.TEXTURE_WRAP_T, gl.REPEAT )
```

texture



gl.REPEAT wrapping



gl.CLAMP wrapping

Texture filtering

Texture images consist of pixels (texels)

Therefore:

- *Magnification*: a pixel on the screen covers only part of a texel (a texel stretches to cover multiple pixels)

- *Minification*: a pixel on the screen covers more than one texels (a texel is squeezed to fit in an area smaller than a pixel)

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) ;
```



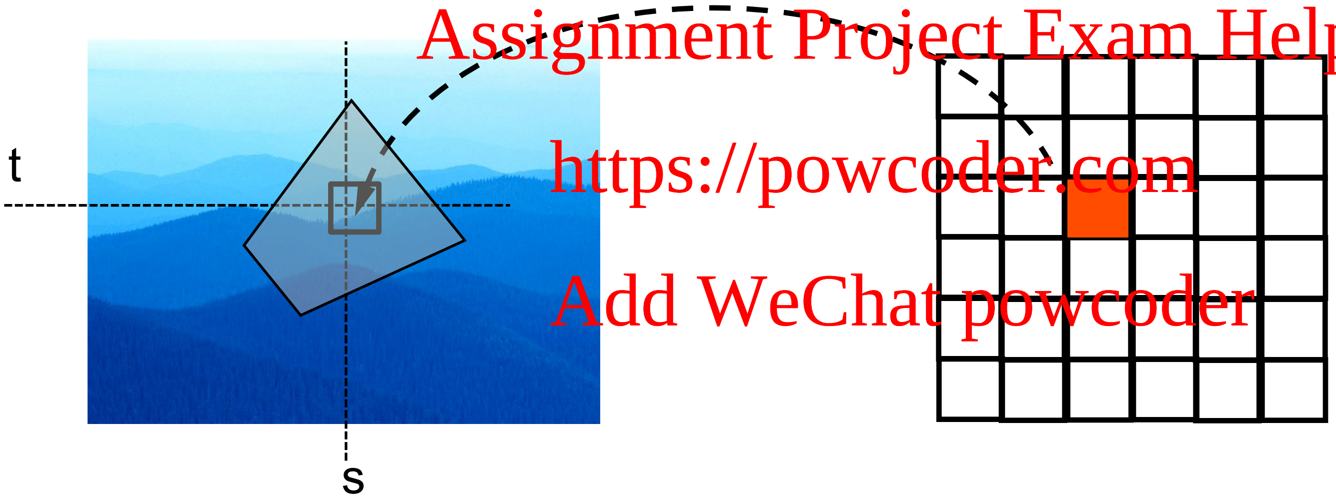
```
GL_TEXTURE_MAG_FILTER: GL_NEAREST or GL_LINEAR
```



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

Filtering textures

Addresses texture minification,magnification



- vec4 texColor = texture(s,t) returns what really?

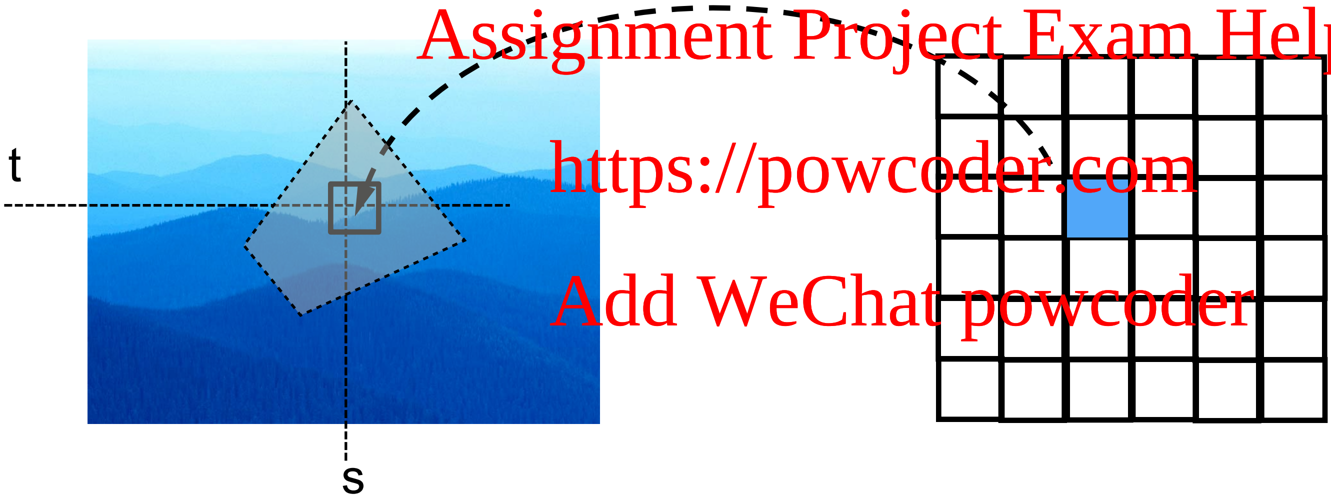
FILTERING

- GL_NEAREST: no filtering, return the texture element closest (in Manhattan distance) to the texture coordinates provided

- GL_LINEAR: Returns the weighted average of the four texture elements that are closest to the texture coordinates provided

Filtering textures

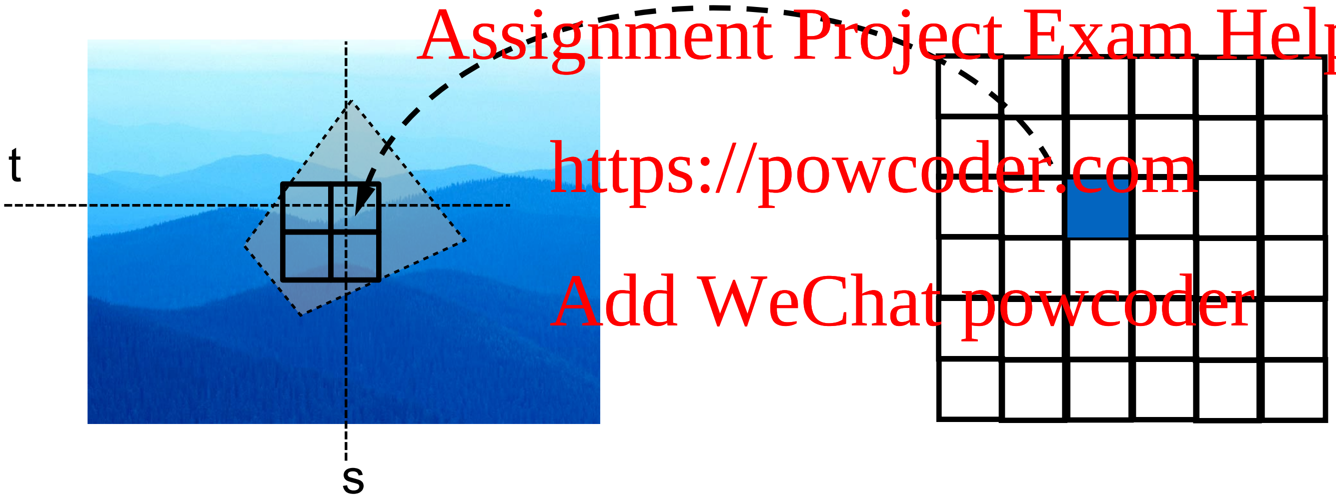
Addresses texture minification,magnification



- vec4 texColor = texture(s,t) returns what?
- Nearest: returns the color a single pixel square

Filtering textures

Addresses texture minification,magnification



- vec4 texColor = texture(s,t) returns what ?
- Linear: returns the average of the nearest four texels
- They capture better how the pixel actually covers texels

The University of New Mexico

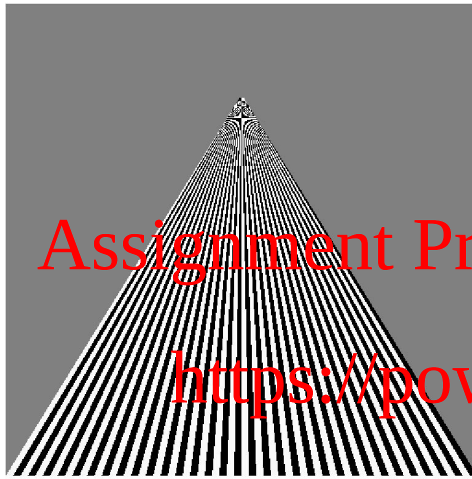
Mipmapped Textures

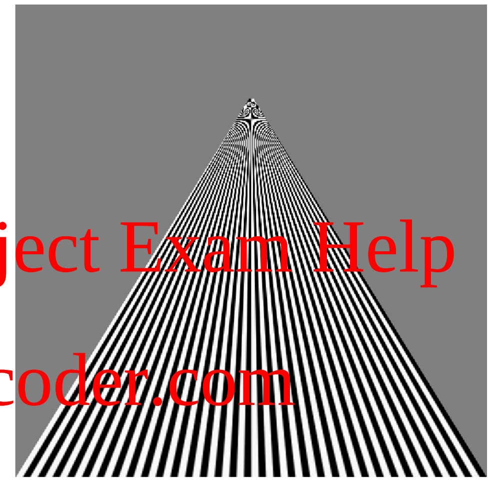
- *Mipmapping* allows for prefiltered texture maps of decreasing resolutions
- Lessens interpolation errors for smaller textured objects
- Declare mipmap level during texture definition

```
gl.texImage2D(gl.TEXTURE_*D, level, … )
```

Example

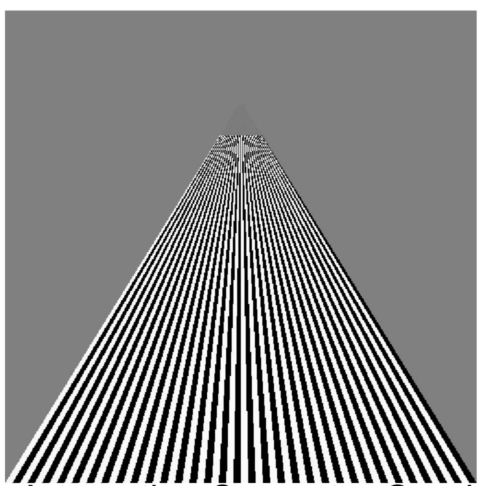
point sampling

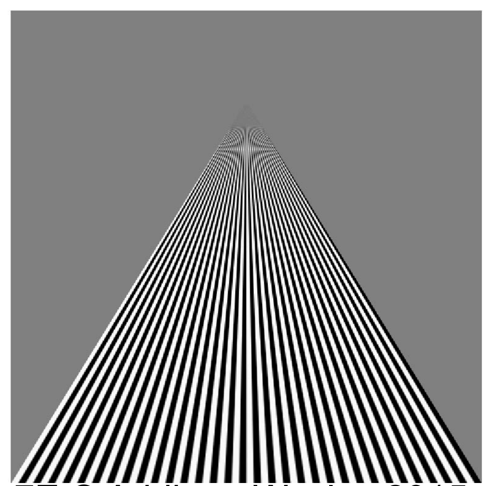




linear filtering

mipmapped point sampling





mipmapped linear filtering

Texture Coordinate Transforms

Texture coordinates are, in fact, 2D coordinates in texture space and they can be transformed with affine transformations

Texture Objects

Copying an image from main memory to video memory is very expensive (`gl.texImage2D`).

- *Create texture names*

- *Bind (create) texture objects to texture data:*
  - Image arrays + texture properties

- *Bind and rebind texture objects.*

Texture Object Creation

```
function loadFileTexture(tex, filename)
{
    tex.textureWebGL  = gl.createTexture();
    tex.image = new Image();
    tex.image.src = filename ;
    tex.isTextureReady = false ;
    tex.image.onload = function() { handleTextureLoaded(tex);
// The image is going to be loaded asyncronously (lazy) which
could be
    // after the program continues to the next functions.
OUCH!
}
```

Texture Object Creation

```
function handleTextureLoaded(textureObj) {
    gl.bindTexture(gl.TEXTURE_2D, textureObj.textureWebGL);
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA,
             gl.UNSIGNED_BYTE, textureObj.image);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER,
             gl.LINEAR);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER,
             gl.LINEAR_MIPMAP_NEAREST);
    gl.generateMipmap(gl.TEXTURE_2D);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S,
       gl.CLAMP_TO_EDGE); //Prevents s-coordinate wrapping
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T,
       gl.CLAMP_TO_EDGE); //Prevents t-coordinate wrapping
    gl.bindTexture(gl.TEXTURE_2D, null);
}
```

Using Textures

```
gl.activeTexture(gl.TEXTURE0) ;
gl.bindTexture(gl.TEXTURE_2D,texture1) ;
gl.uniform1i(gl.getUniformLocation(program, "stexture1"), 0);

drawCube() ; //
```



```
gl.activeTexture(gl.TEXTURE0) ;
gl.bindTexture(gl.TEXTURE_2D,texture2) ;
gl.uniform1i(gl.getUniformLocation(program, "stexture1"), 0);

drawSphere() ;  // different texture for the sphere
```

Fragment shader

```
precision mediump float;

uniform sampler2D stexture1;

varying vec4 fColor;
varying vec2 fTexCoord ;

void
main()
{

    gl_FragColor = vec4(fColor.rgb,1.0);
    gl_FragColor = texture2D( stexture1, fTexCoord );

}
```

Multiple Textures

```
gl.activeTexture(GL_TEXTURE0) ;
gl.bindTexture(GL_TEXTURE_2D,texture1) ;
gl.uniform1i(gl.getUniformLocation(program, "stexture1"), 0);
```



```
gl.activeTexture(gl.TEXTURE1) ;
gl.bindTexture(GL_TEXTURE_2D,texture2) ;
gl.uniform1i(gl.getUniformLocation(program, "stexture2"),1);

drawSphere() ;  // two active textures
```

Fragment Shader

```
precision mediump float;

uniform sampler2D stexture1;
uniform sampler2D stexture2;

varying vec4 fColor;
varying vec2 fTexCoord ;

void
main()
{
    gl_FragColor = vec4(fColor.rgb,1.0) ;
    vec4 c1 = texture2D( stexture1, fTexCoord );
    vec4 c2 = texture2D( stexture2, fTexCoord );
    gl_FragColor = mix(c1,c2,0.5);

}
```

Filters!


```
gl.texParameterf(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER,
                 gl.NEAREST);
```



```
gl.texParameterf(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER,
                 g.NEAREST);
```


DO NOT FORGET TO SET THE FILTERS!! You get black textures because of the default settings

Procedural textures

Fragment shaders can generate textures on the fly

Define textures through a process (function) instead of predefined samples

– 2D: T = F(s,t), 3D: T = F(x,y,z)

Advantages

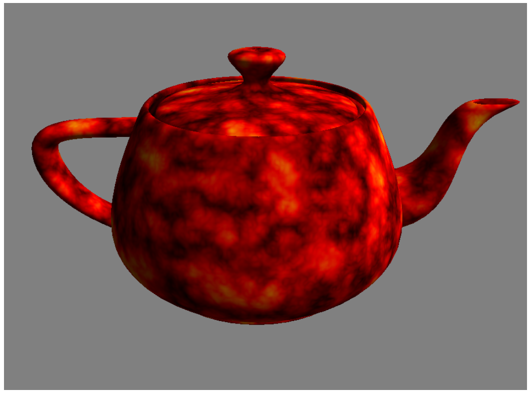
Process can be parameterized

Needs less memory especially for the 3D case

No predefined resolution

Disadvantages

Slower texture lookup

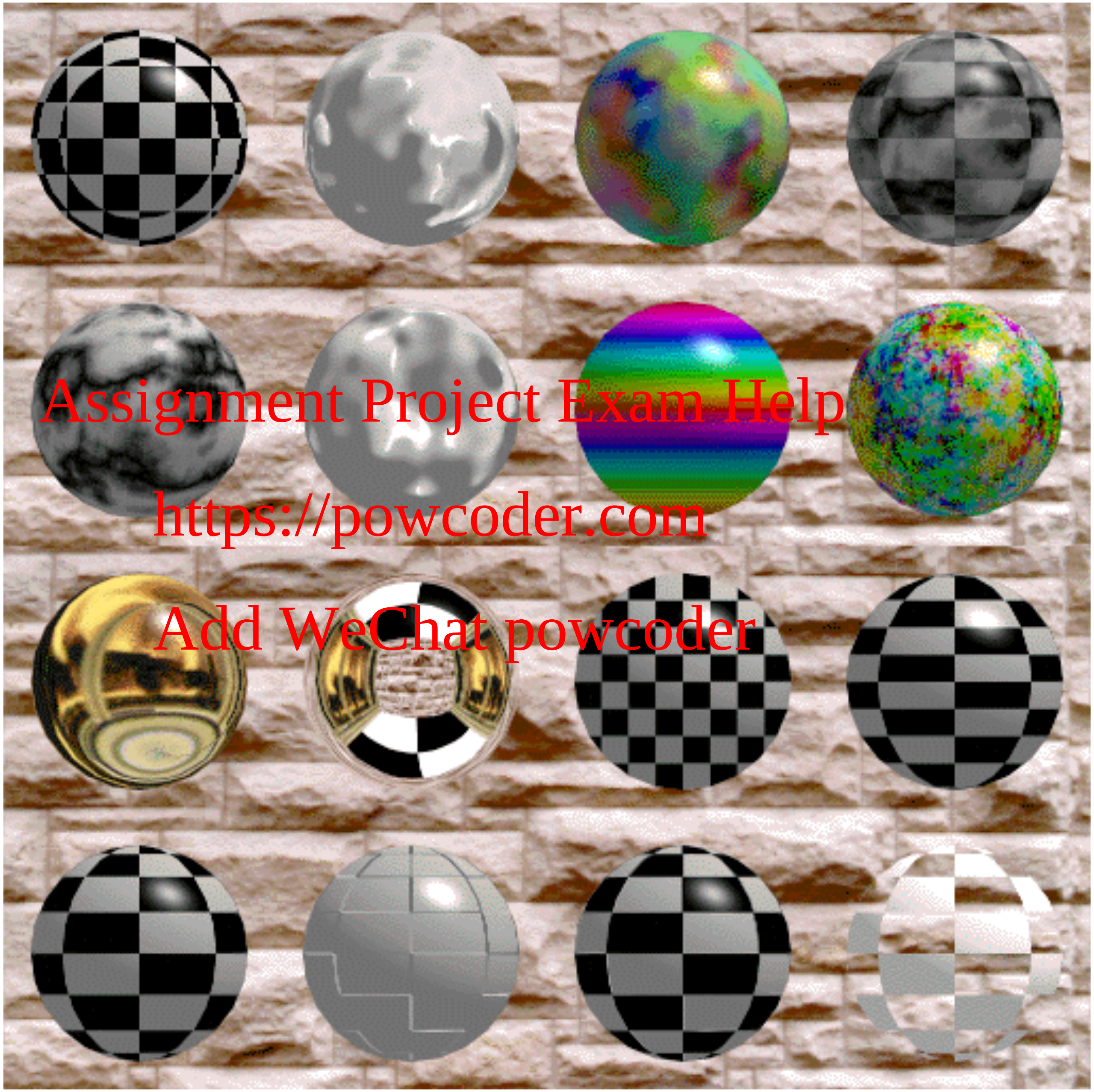


In practice

Combinations of both approaches

Complex objects will use multiple textures

some based on images some procedural



Common use of Textures: Light maps

For static objects we can simulate lighting by blending textures