CS559: Computer Graphics

Lecture 21: Texture mapping

Li Zhang

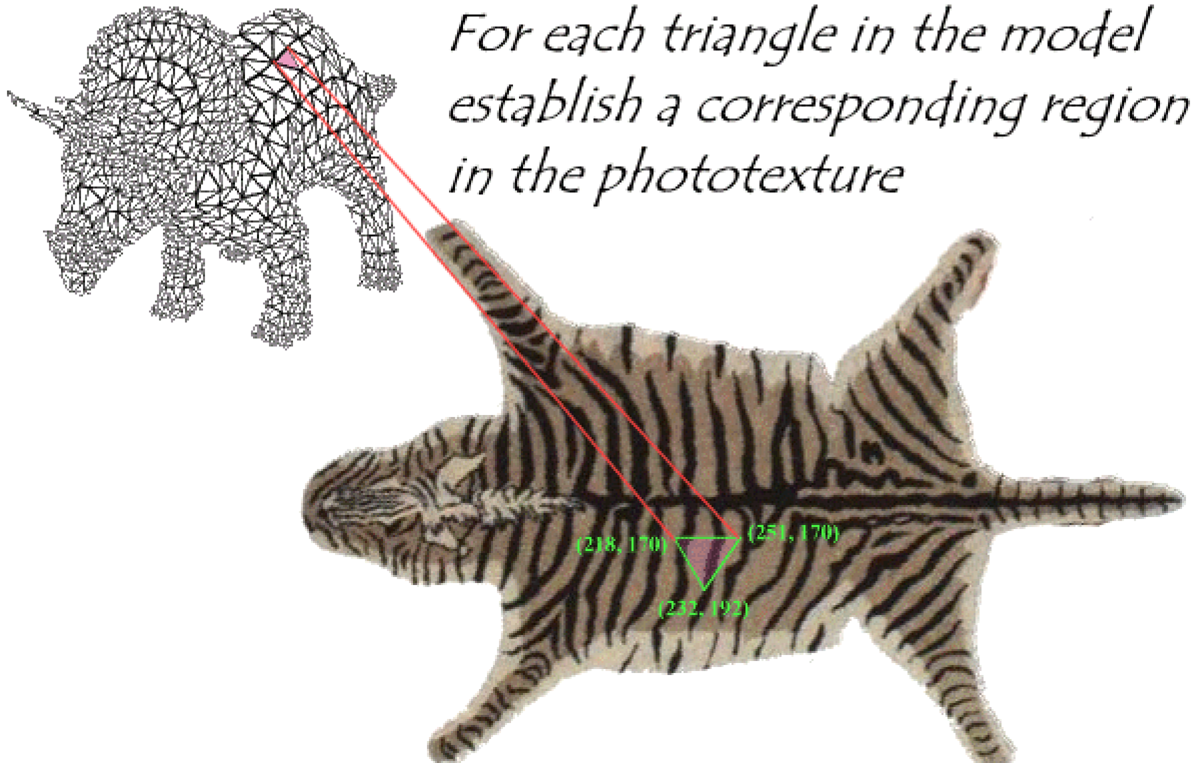
Spring 2010

Many slides from Ravi Ramamoorthi, Columbia Univ, Greg Humphreys, UVA and Rosalee Wolfe, DePaul tutorial teaching texture mapping visually

Photo-textures

The concept is very simple!

For each triangle in the model establish a corresponding region in the phototexture

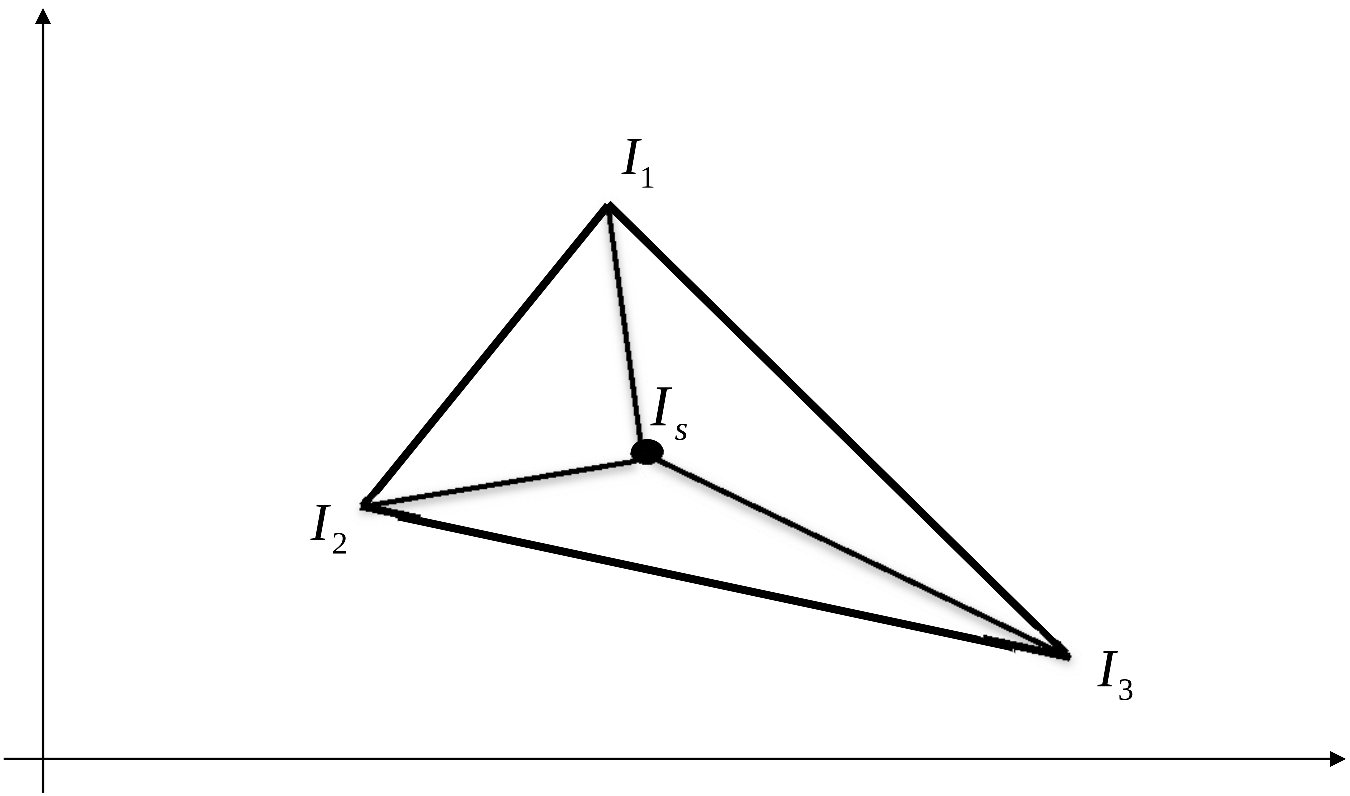


During rasterization interpolate the coordinate indices into the texture map

Slides from Leonard Mcmillan

1st idea: Gouraud interp. of texcoords

Using barycentric Coordinates

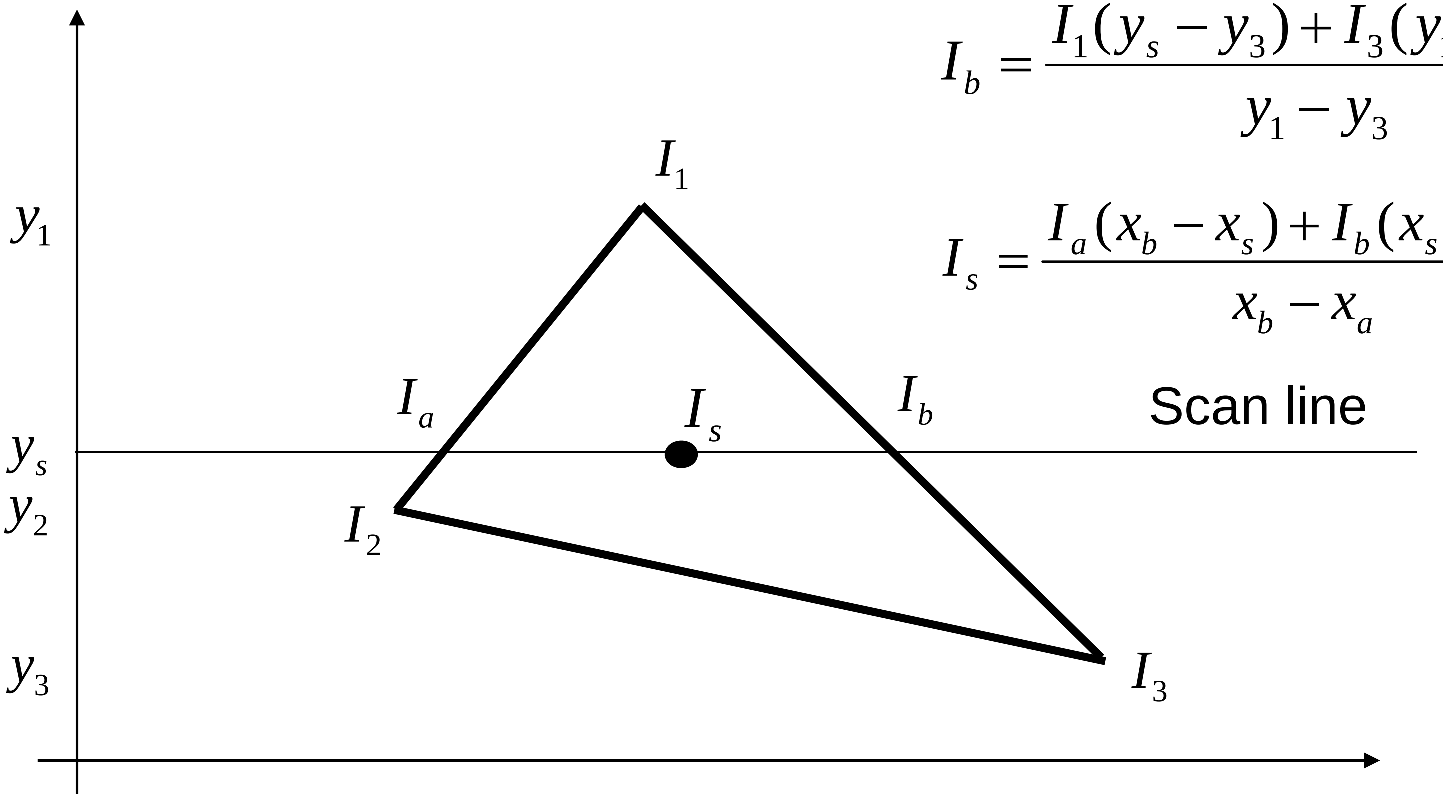


1st idea: Gouraud interp. of texcoords

$$I_a = \frac{I_1(y_s - y_2) + I_2(y_1 - y_s)}{y_1 - y_2}$$

$$I_b = \frac{I_1(y_s - y_3) + I_3(y_1 - y_s)}{y_1 - y_3}$$

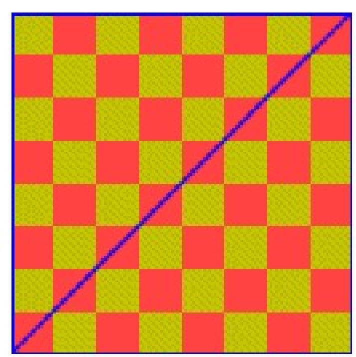
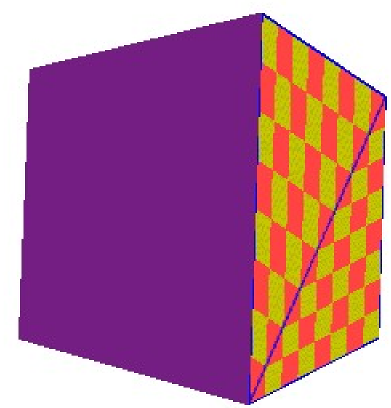
$$I_s = \frac{I_a(x_b - x_s) + I_b(x_s - x_a)}{x_b - x_a}$$



Artifacts

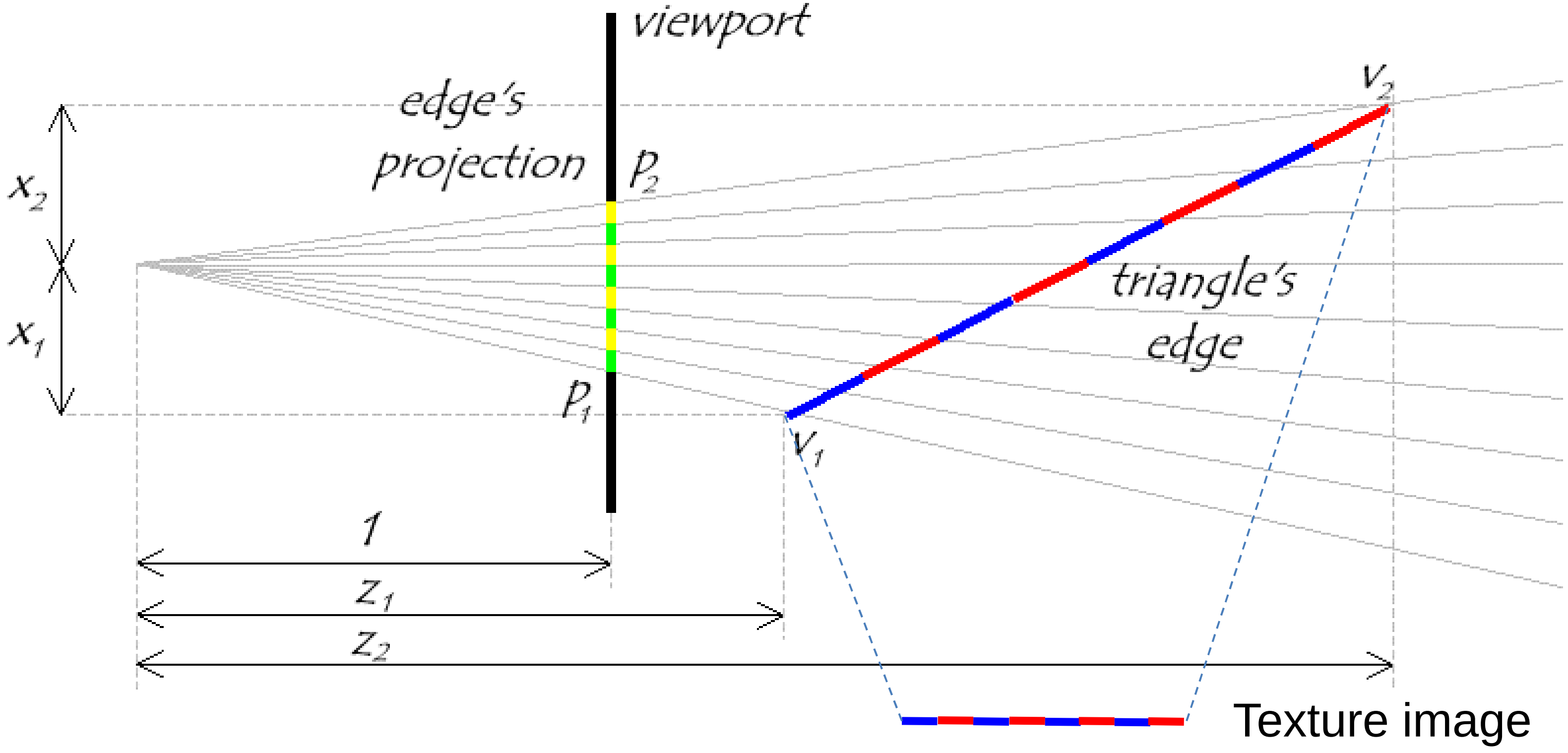
- McMillan’s demo of this is at
  http://graphics.lcs.mit.edu/classes/6.837/F98/Lecture21/Slide05.html
- Another example
  http://graphics.lcs.mit.edu/classes/6.837/F98/Lecture21/Slide06.html
- What artifacts do you see?
- Why?
- Hint: problem is in interpolating parameters



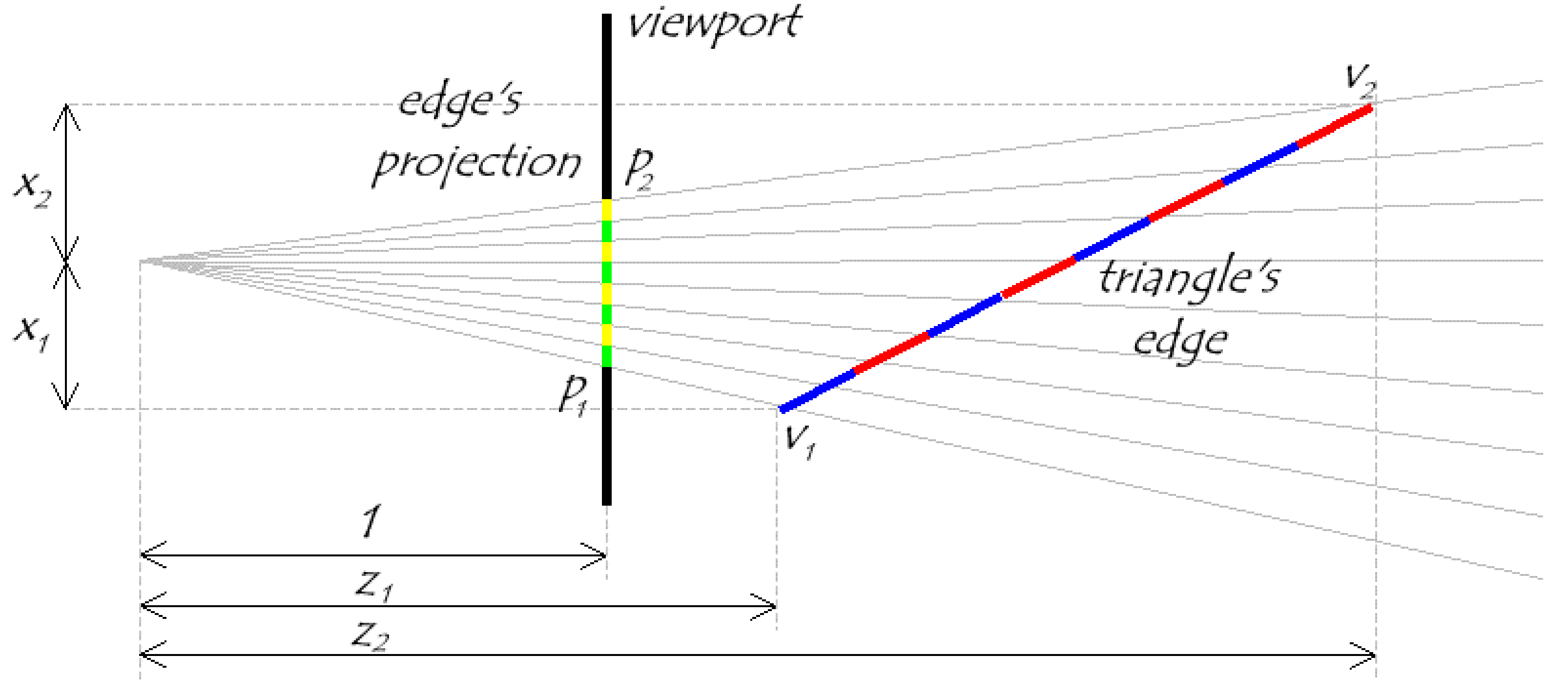



Interpolating Parameters

- The problem turns out to be fundamental to interpolating parameters in screen-space
  - *Uniform steps in screen space ≠ uniform steps in world space*



Linear Interpolation in Screen Space



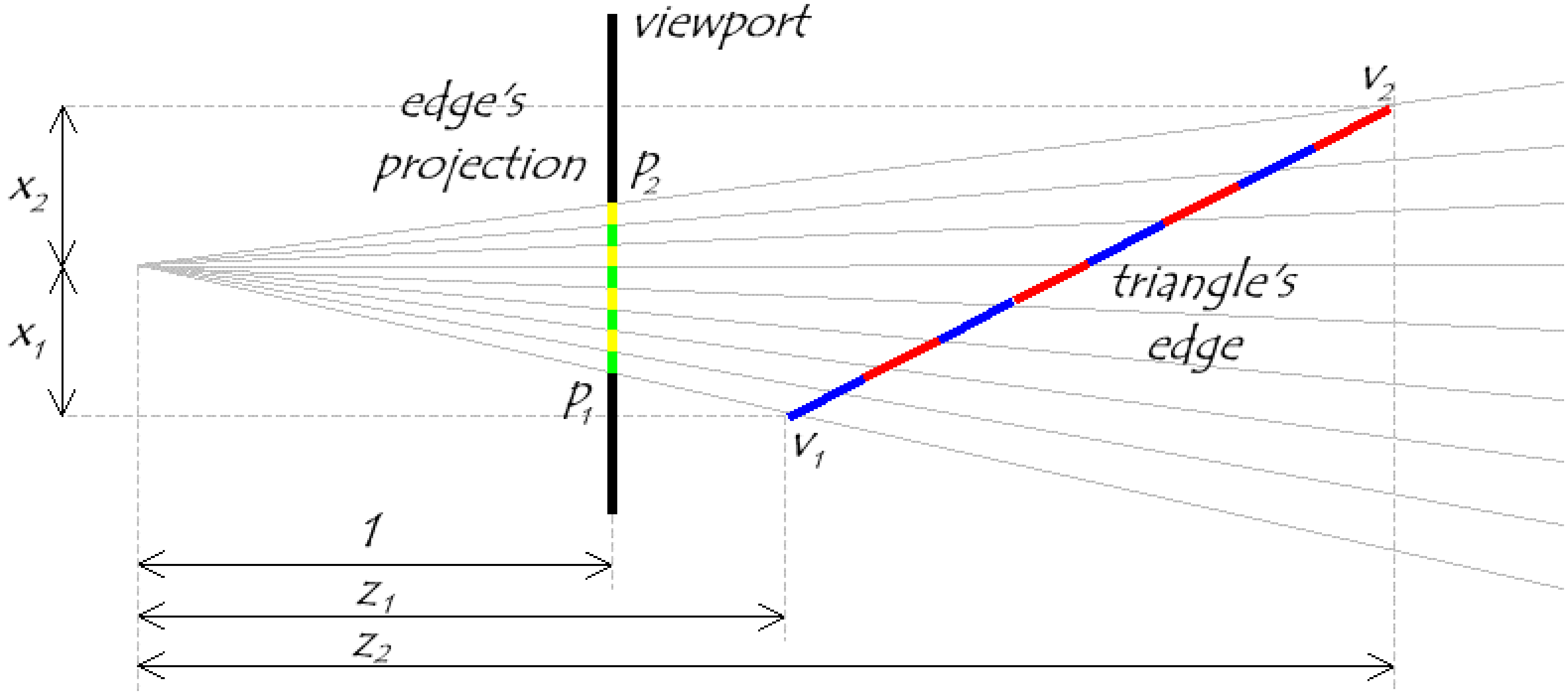
Compare linear interpolation in screen space

$$p(t) = p_1 + t(p_2 - p_1) = \frac{x_1}{z_1} + t\left(\frac{x_2}{z_2} - \frac{x_1}{z_1}\right)$$

Without loss of generality, let's assume that the viewport is located 1 unit away from the center of projection. That is $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$

Slides from Jingyi Yu

Linear Interpolation in 3-Space



to interpolation in 3-space:

$$\begin{bmatrix} x \\ z \end{bmatrix} = \begin{bmatrix} x_1 \\ z_1 \end{bmatrix} + s\left(\begin{bmatrix} x_2 \\ z_2 \end{bmatrix} - \begin{bmatrix} x_1 \\ z_1 \end{bmatrix}\right) \qquad P\left(\begin{bmatrix} x \\ z \end{bmatrix}\right) = \frac{x_1 + s(x_2 - x_1)}{z_1 + s(z_2 - z_1)}$$

Slides from Jingyi Yu

How to make them Mesh

Still need to scan convert in screen space... so we need a mapping from *t* values to *s* values. We know that the all points on the 3-space edge project onto our screen-space line. Thus we can set up the following equality:

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

and solve for *s* in terms of *t* giving:

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

Unfortunately, at this point in the pipeline (after projection) we no longer have z_1 and z_2 lingering around (Why? Efficiency, don't need to compute 1/z all the time). However, we do have $w_1 = 1/z_1$ and $w_2 = 1/z_2$.

$$s = \frac{t\ \frac{1}{w_1}}{\frac{1}{w_2} + t\ (\frac{1}{w_1} - \frac{1}{w_2})} = \frac{t\ w_2}{w_1 + t\ (w_2 - w_1)}$$

Slides from Jingyi Yu

Interpolating Parameters

We can now use this expression for *s* to interpolate arbitrary parameters, such as texture indices (*u, v)*, over our 3-space triangle. This is accomplished by substituting our solution for *s* given *t* into the parameter interpolation.

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

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

Therefore, if we **premultiply all parameters that we wish to interpolate in 3-space by their corresponding *w* value** and add a new plane equation to interpolate the *w* values themselves, we can interpolate the numerators and denominator in screen-space. We then need to perform a divide a each step to get to map the screen-space interpolants to their corresponding 3-space values. This is a simple modification to the triangle rasterizer that we developed in class.

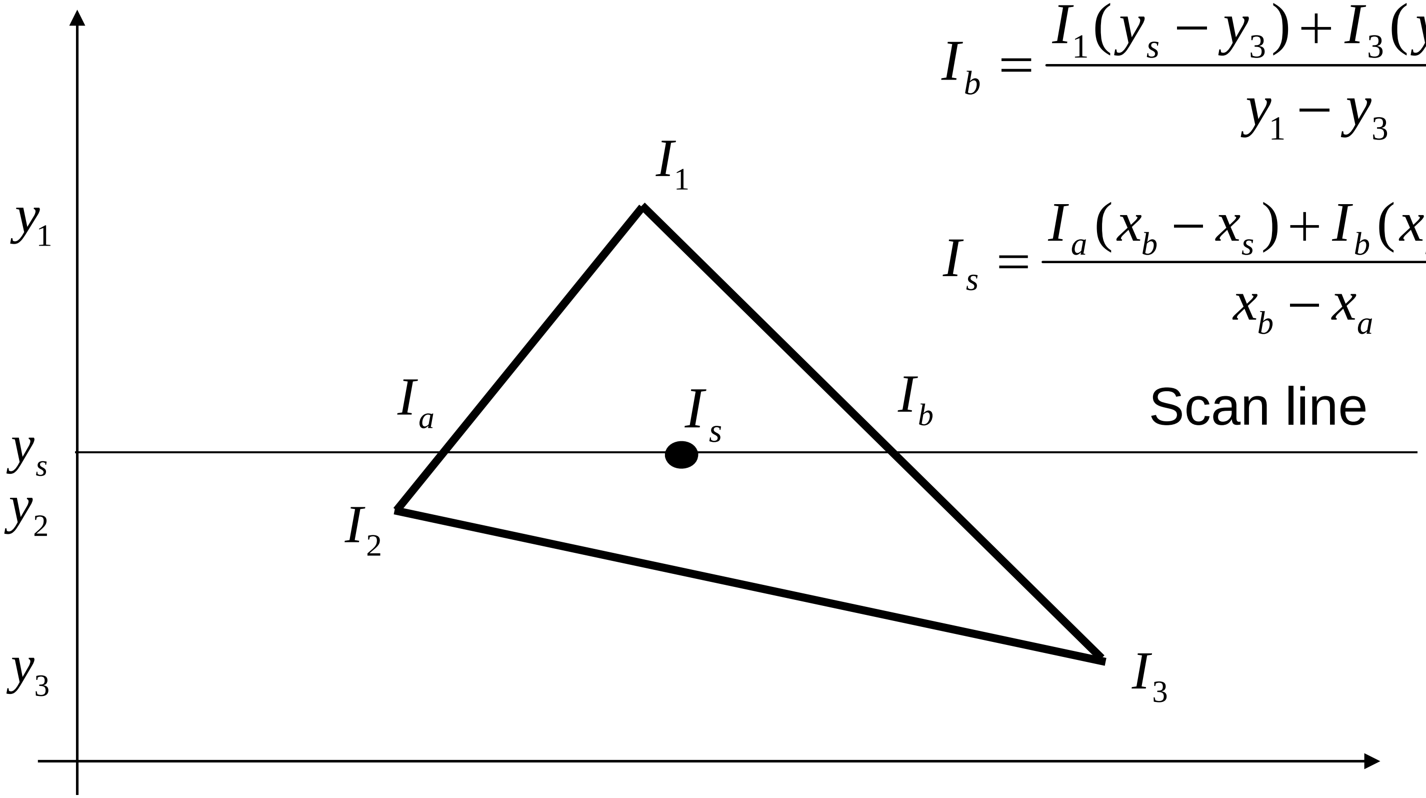
Slides from Jingyi Yu

1st idea: Gouraud interp. of texcoords

$$I_a = \frac{I_1(y_s - y_2) + I_2(y_1 - y_s)}{y_1 - y_2}$$

$$I_b = \frac{I_1(y_s - y_3) + I_3(y_1 - y_s)}{y_1 - y_3}$$

$$I_s = \frac{I_a(x_b - x_s) + I_b(x_s - x_a)}{x_b - x_a}$$

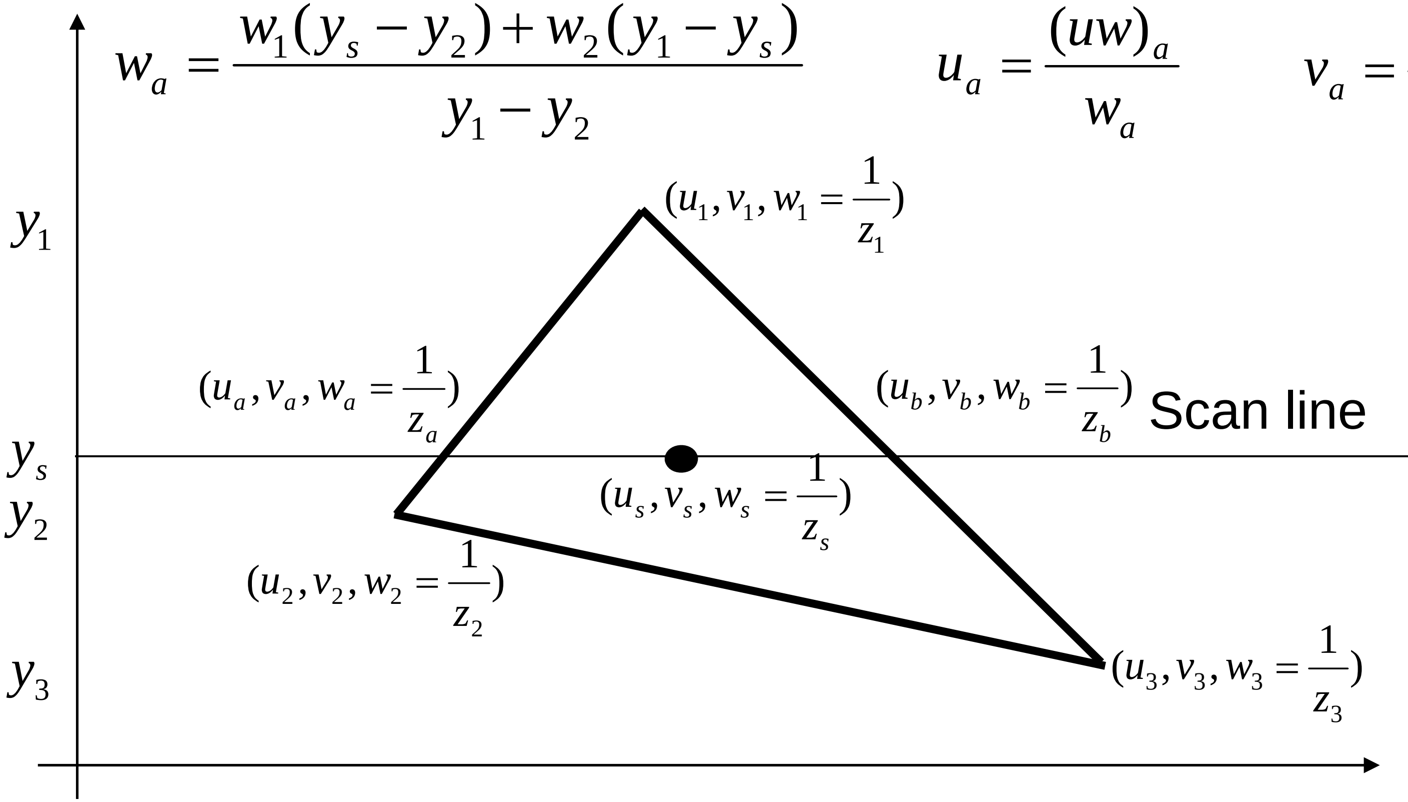


Replace I to uw, vw, and w, then compute (uw/w, and vw/w)

1st idea: Gouraud interp. of texcoords

$$(uw)_a = \frac{u_1 w_1 (y_s - y_2) + u_2 w_2 (y_1 - y_s)}{y_1 - y_2} \qquad (vw)_a = \frac{v_1 w_1 (y_s - y_2) + v_2 w_2 (y_1 - y_s)}{y_1 - y_2}$$

$$w_a = \frac{w_1 (y_s - y_2) + w_2 (y_1 - y_s)}{y_1 - y_2} \qquad u_a = \frac{(uw)_a}{w_a} \qquad v_a = \frac{(vw)_a}{w_a}$$

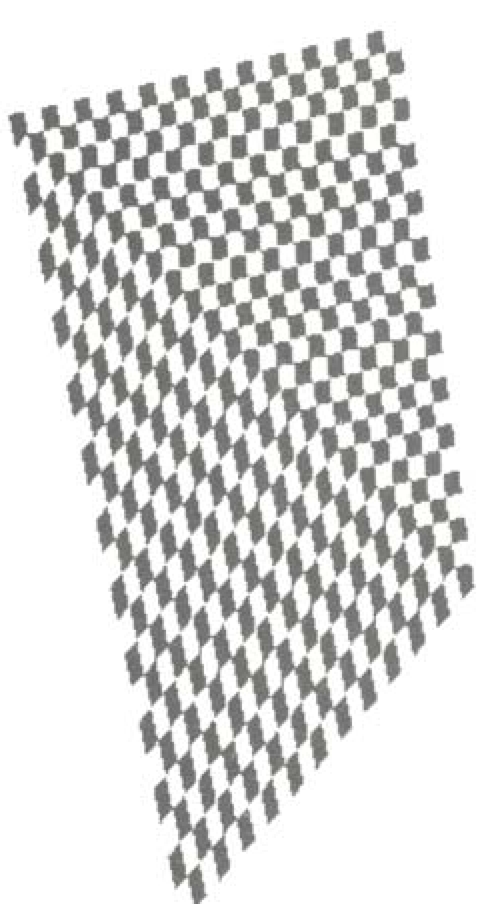


Do same thing for point b. From a and b, interpolate for s

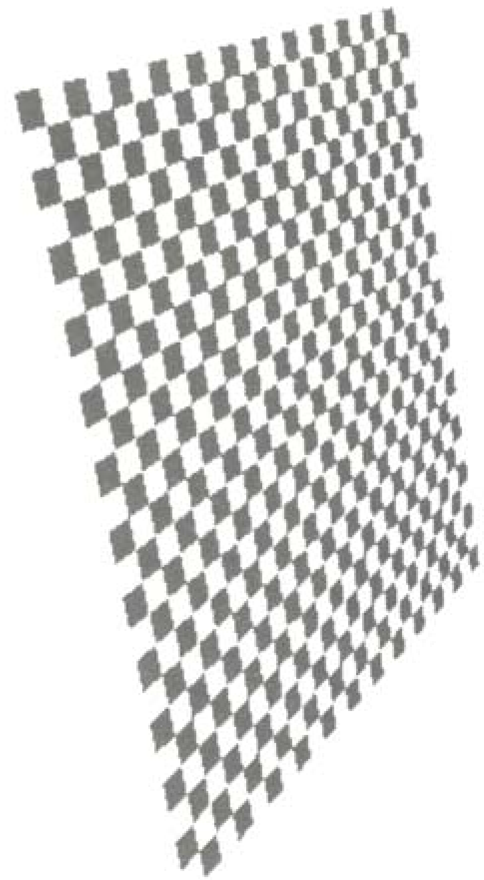
Perspective-Correct Interpolation

- In short…
  - Rather than interpolating *u* and *v* directly, interpolate *uw* and *vw* and *w*, and compute $u = uw/w$ and $v = vw/w$ for each pixel
    - These do interpolate correctly in screen space
    - Need to keep $w \propto 1/z$
    - This unfortunately involves a divide per pixel
- http://graphics.lcs.mit.edu/classes/6.837/F98/Lecture21/Slide14.html

Texture Mapping



Linear interpolation
of texture coordinates



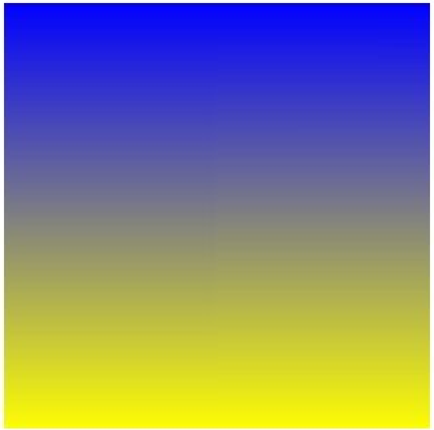
Correct interpolation
with perspective divide

http://graphics.lcs.mit.edu/classes/6.837/F98/Lecture21/Slide14.html

Hill Figure 8.42

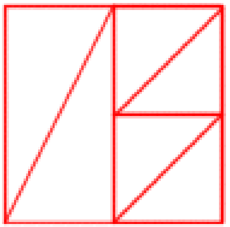
Why don’t we notice?

Traditional screen-space Gourand shading is wrong. However, you usually will not notice because the transition in colors is very smooth (And we don't know what the right color should be anyway, all we care about is a pretty picture). There are some cases where the errors in Gourand shading become obvious.



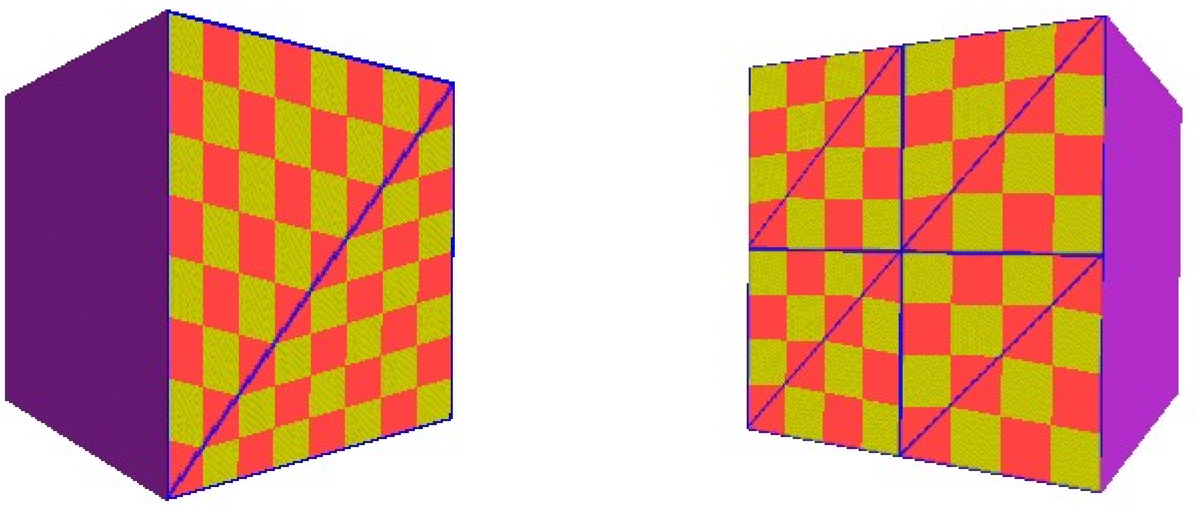
- When do we notice?
  - When switching between different levels-of-detail representations
  - At "T" joints.

A "T" joint



Dealing with Incorrect Interpolation

You can reduce the perceived artifacts of non-perspective correct interpolation by subdividing the texture-mapped triangles into smaller triangles (why does this work?). But, fundamentally the screen-space interpolation of projected parameters is `inherently flawed`.



http://groups.csail.mit.edu/graphics/classes/6.837/F98/Lecture21/Slide15.html

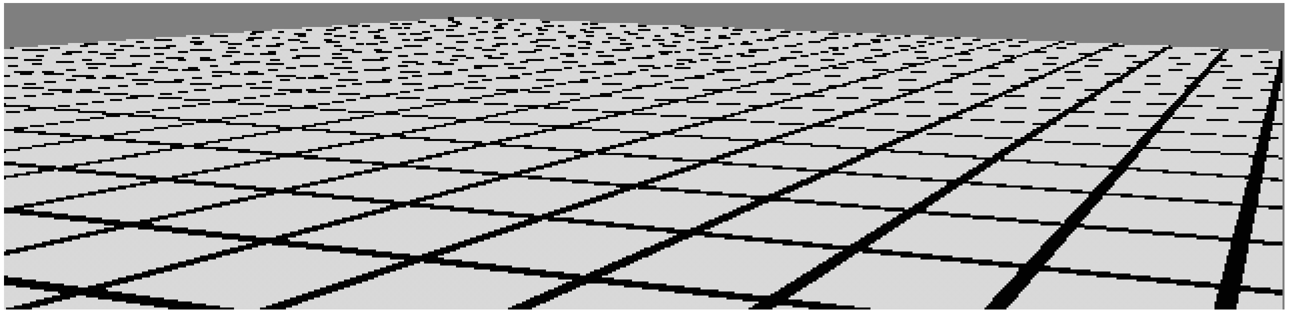
Outline

- Types of mappings
- Interpolating texture coordinates
- *Texture Resampling*
- Broader use of textures

Texture Map Filtering

- Naive texture mapping aliases badly
- Look familiar?

```
int uval = round(u * W);
int vval = round(v * H);
int pix = texture.getPixel(uval, vval);
```



Nearest Neighbor Sampling

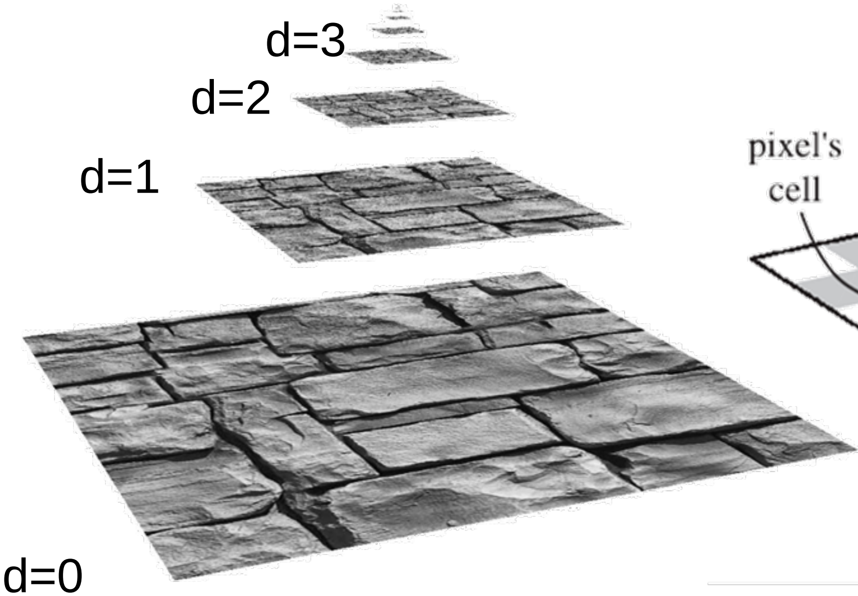
Texture Map Filtering

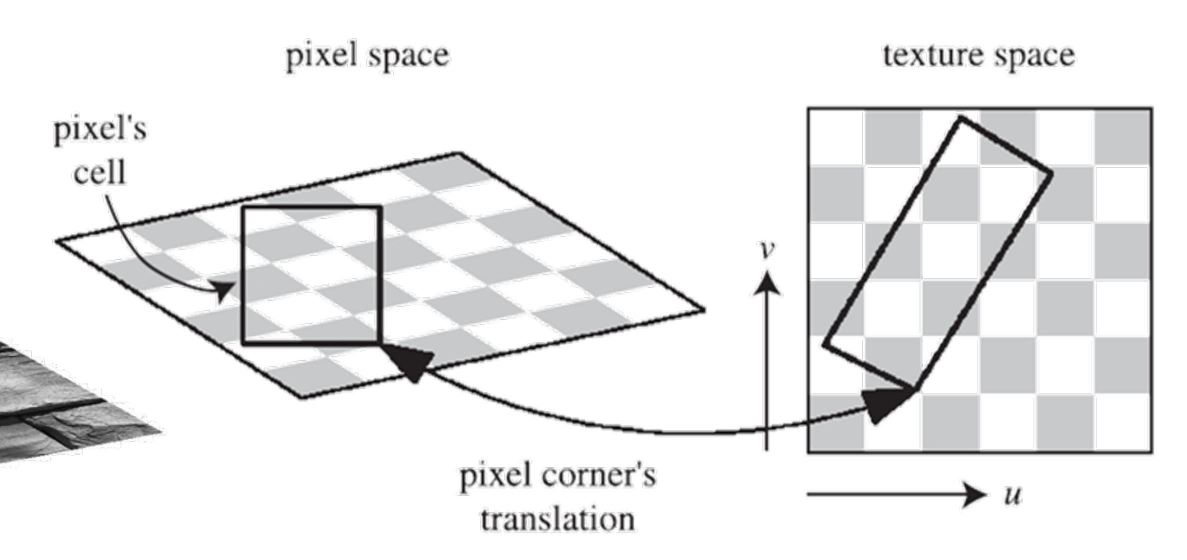
- Naive texture mapping aliases badly
- Look familiar?

```
int uval = round(u * W);
int vval = round(v * H);
int pix = texture.getPixel(uval, vval);
```

- Actually, each pixel maps to a region in texture
  - |PIX| < |TEX|
    - Easy: interpolate (bilinear) between texel values
  - |PIX| > |TEX|
    - Hard: average the contribution from multiple texels
  - |PIX| ~ |TEX|
    - Still need interpolation!

Mipmap





Let d = |PIX| be a measure of pixel size

Sample (u,v,d)

Using tri-linear interpolation

What’s the memory overhead?

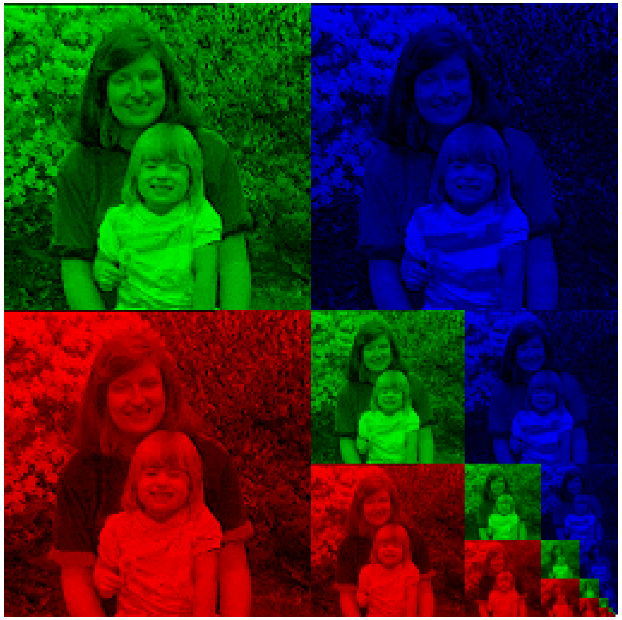
Option 1: use the longer edge of the quadrilateral formed by the pixel's cell to approximate the pixel's coverage

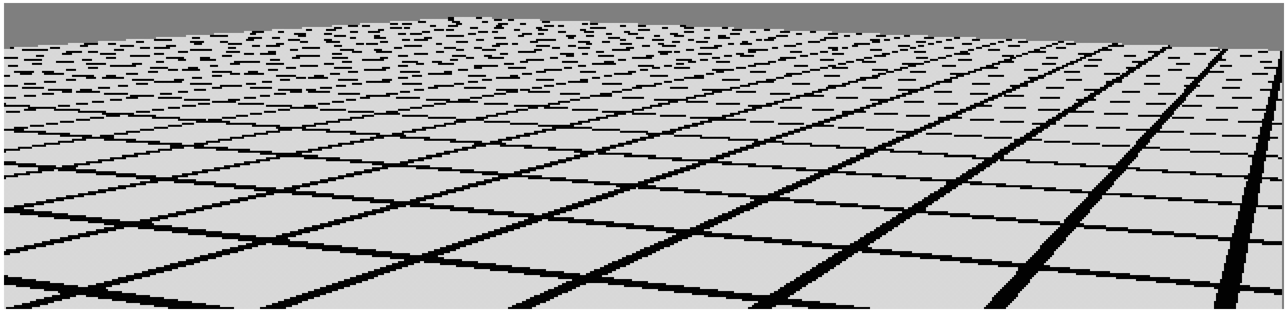
Option 2: use the max of |du/dx|, |du/dy|, |dv/x|, |dv/dy|

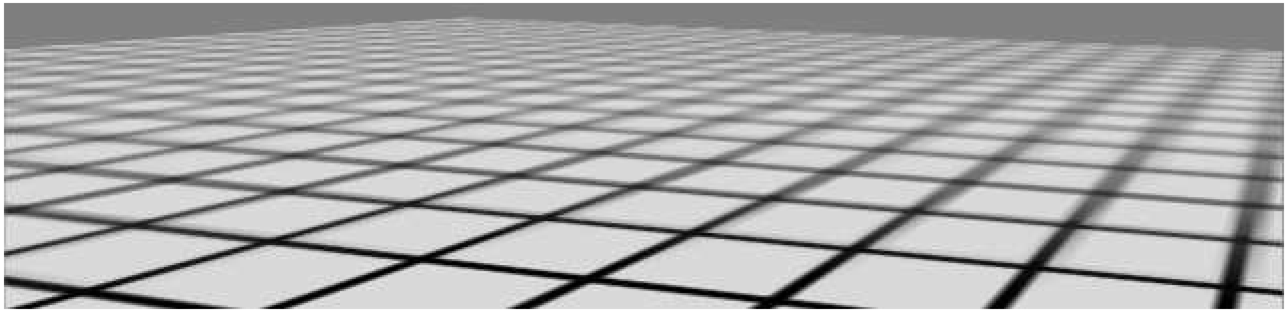
Then take logarithm

Storing MIP Maps

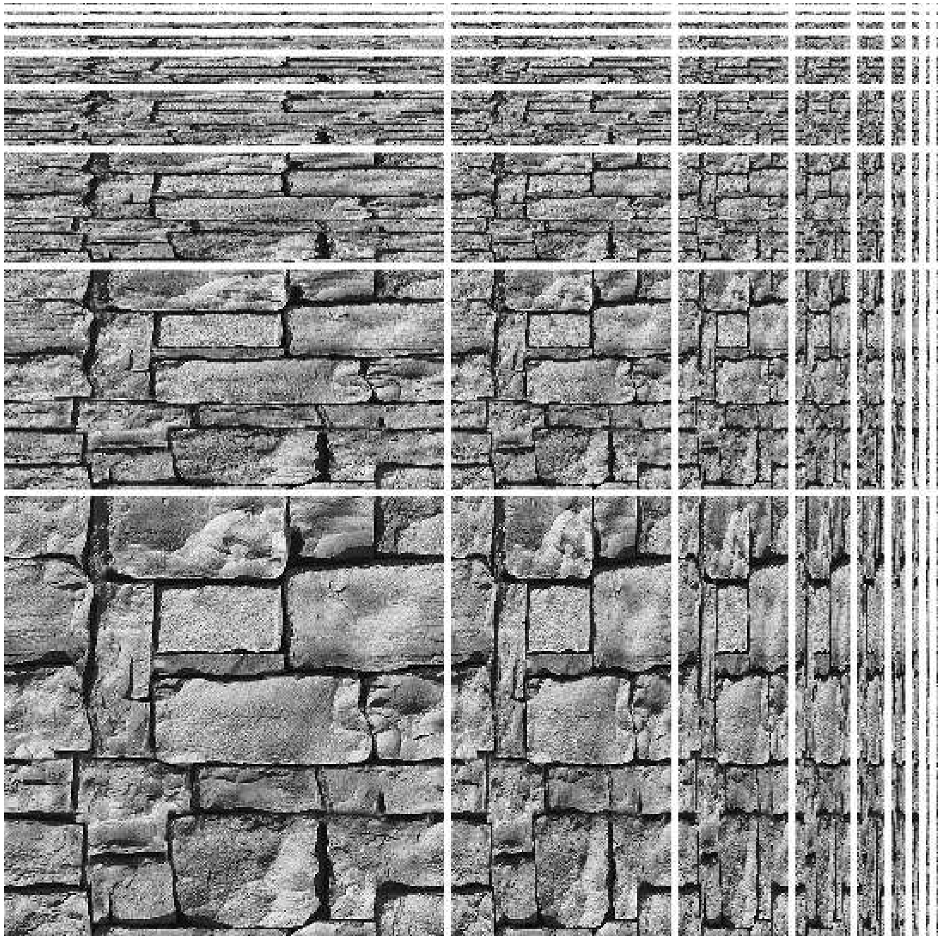
- One convienent method of storing a MIP map is shown below (It also nicely illustrates the 1/3 overhead of maintaining the MIP map).




Nearest Neighbor Sampling


Mipmap Sampling

Ripmap



Sample (u,v,du,dv)　　　Using Trilinaer => quadrilinear

What’s the memory overhead?

Summed Area Table

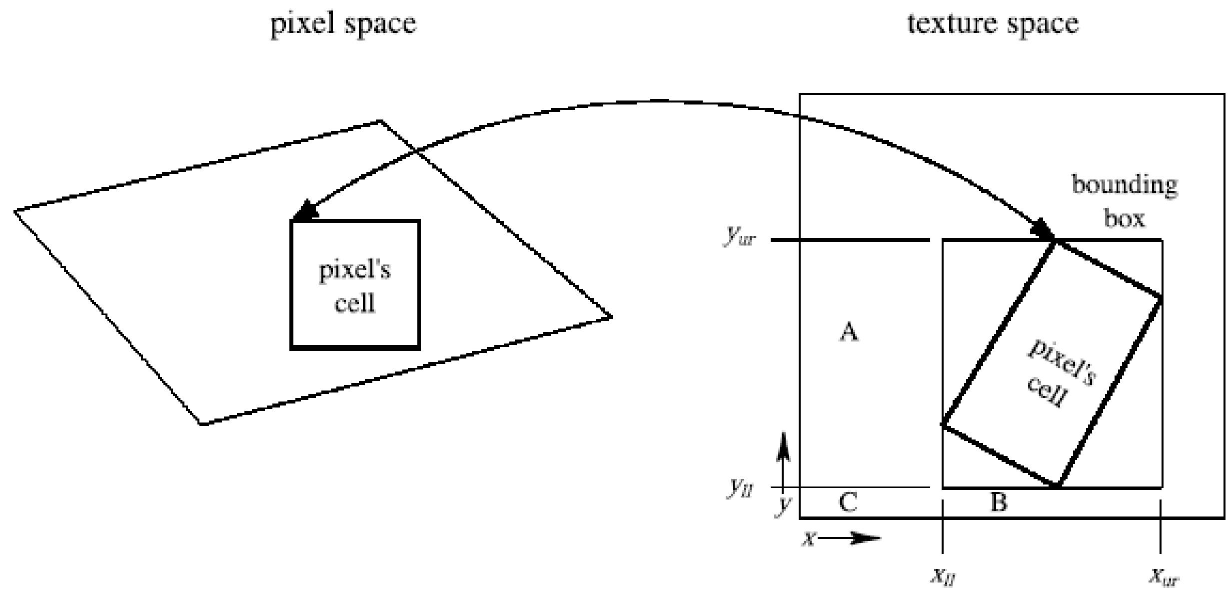


Figure 6.15: The pixel cell is back-projected onto the texture, bound by a rectangle, and the four corners of the rectangle are used to access the summed-area table.

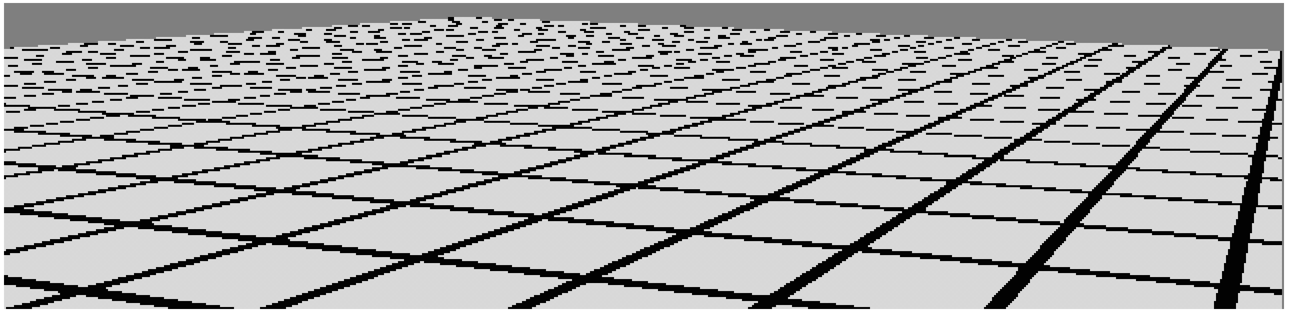
$$\mathbf{c} = \frac{\mathbf{s}[x_{ur}, y_{ur}] - \mathbf{s}[x_{ur}, y_{ll}] - \mathbf{s}[x_{ll}, y_{ur}] + \mathbf{s}[x_{ll}, y_{ll}]}{(x_{ur} - x_{ll})(y_{ur} - y_{ll})}$$

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

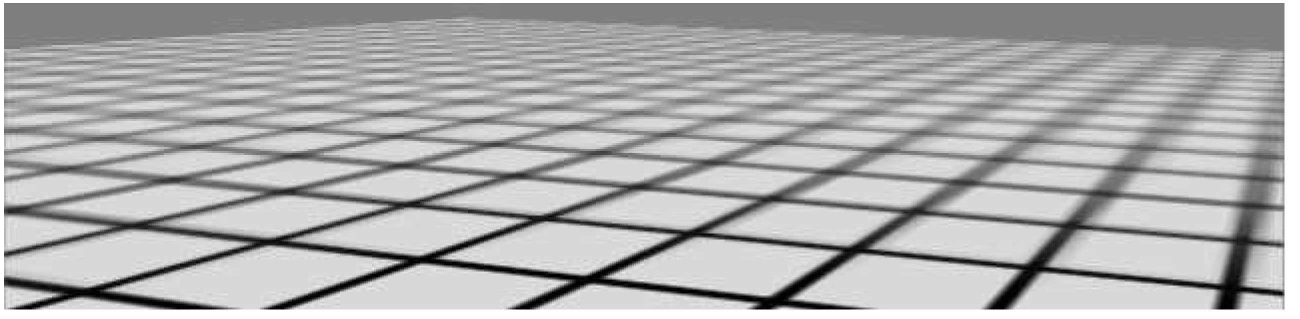
→

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

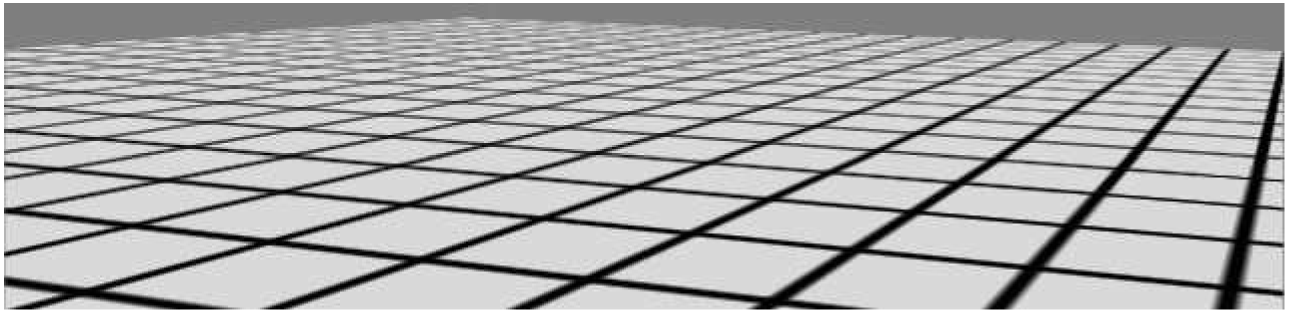
What's the memory overhead?



Nearest Neighbor Sampling



Mipmap Sampling



Summed Area Table

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?
- What sort of low-pass filter does a summed-area table implement?

No Filtering

MIP mapping

Summed-Area Table

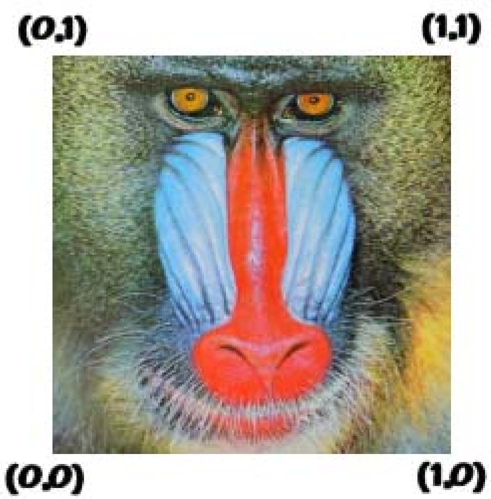
- Other filtering method, see *Real-Time Rendering* chapter 6.

Outline

- Types of mappings
- Interpolating texture coordinates
- Texture Resampling
- *Texture mapping OpenGL*
- Broader use of textures

Simple OpenGL Example

- Specify a texture coordinate at each vertex *(s, t)*
- Canonical coordinates where *s* and *t* are between 0 and 1




```
public void Draw() {
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glLoadIdentity();
        glTranslated(centerx, centery, depth);
        glMultMatrixf(Rotation);
                 :
        // Draw Front of the Cube
        glEnable(GL_TEXTURE_2D);
        glBegin(GL_QUADS);
             glTexCoord2d(0, 1);
             glVertex3d( 1.0, 1.0, 1.0);
             glTexCoord2d(1, 1);
             glVertex3d(-1.0, 1.0, 1.0);
             glTexCoord2d(1, 0);
             glVertex3d(-1.0,-1.0, 1.0);
             glTexCoord2d(0, 0);
             glVertex3d( 1.0,-1.0, 1.0);
        glEnd();
        glDisable(GL_TEXTURE_2D);
      :
        glFlush();
```

glTexCoord works like glColor

Slides from Jingyi Yu

Initializing Texture Mapping

- Generate an artificial pattern or load in an image

```
#define checkImageWidth 64
#define checkImageHeight 64
static GLubyte checkImage[checkImageHeight][checkImageWidth][4];

void makeCheckImage(void) {
	int i, j, c;
	for (i = 0; i < checkImageHeight; i++) {
		for (j = 0; j < checkImageWidth; j++) {
			c = ((((i&0x8)==0)^((j&0x8))==0))*255;
			checkImage[i][j][0] = (GLubyte) c;
			checkImage[i][j][1] = (GLubyte) c;
			checkImage[i][j][2] = (GLubyte) c;
			checkImage[i][j][3] = (GLubyte) 255;
		}
	}
}
```

Initializing Texture Mapping

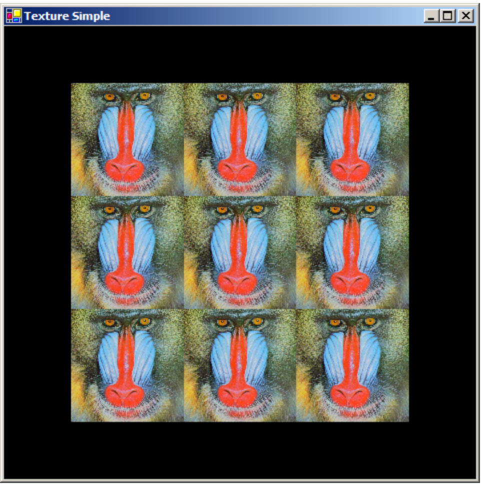
```
static GLubyte image[64][64][4];
static GLuint  texname;

void init(void) {
        glClearColor (0.0, 0.0, 0.0, 0.0);
        glShadeModel(GL_FLAT);
        glEnable(GL_DEPTH_TEST);
        //load in or generate image;
        …
        glPixelStorei(GL_UNPACK_ALIGNMENT, 1);

        glGenTextures(1, &texName);
        glBindTexture(GL_TEXTURE_2D, texName);

        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
        glTexPrameteri(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, checkImageWidth, checkImageHeight, 0,
                GL_RGBA, GL_UNSIGNED_BYTE, image);
}
```

Level index in the Pyramid

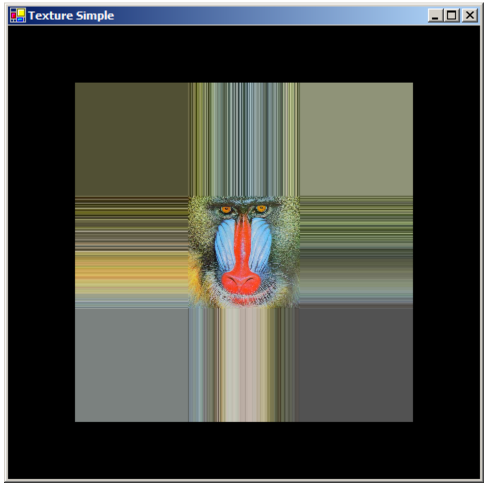
OpenGL Texture Peculiarities

- The width and height of Textures in OpenGL must be powers of 2
- The parameter space of each dimension of a texture ranges from [0,1) regardless of the texture's actual size.
- The behavior of texture indices outside of the range [0,1) is determined by the texture wrap options.

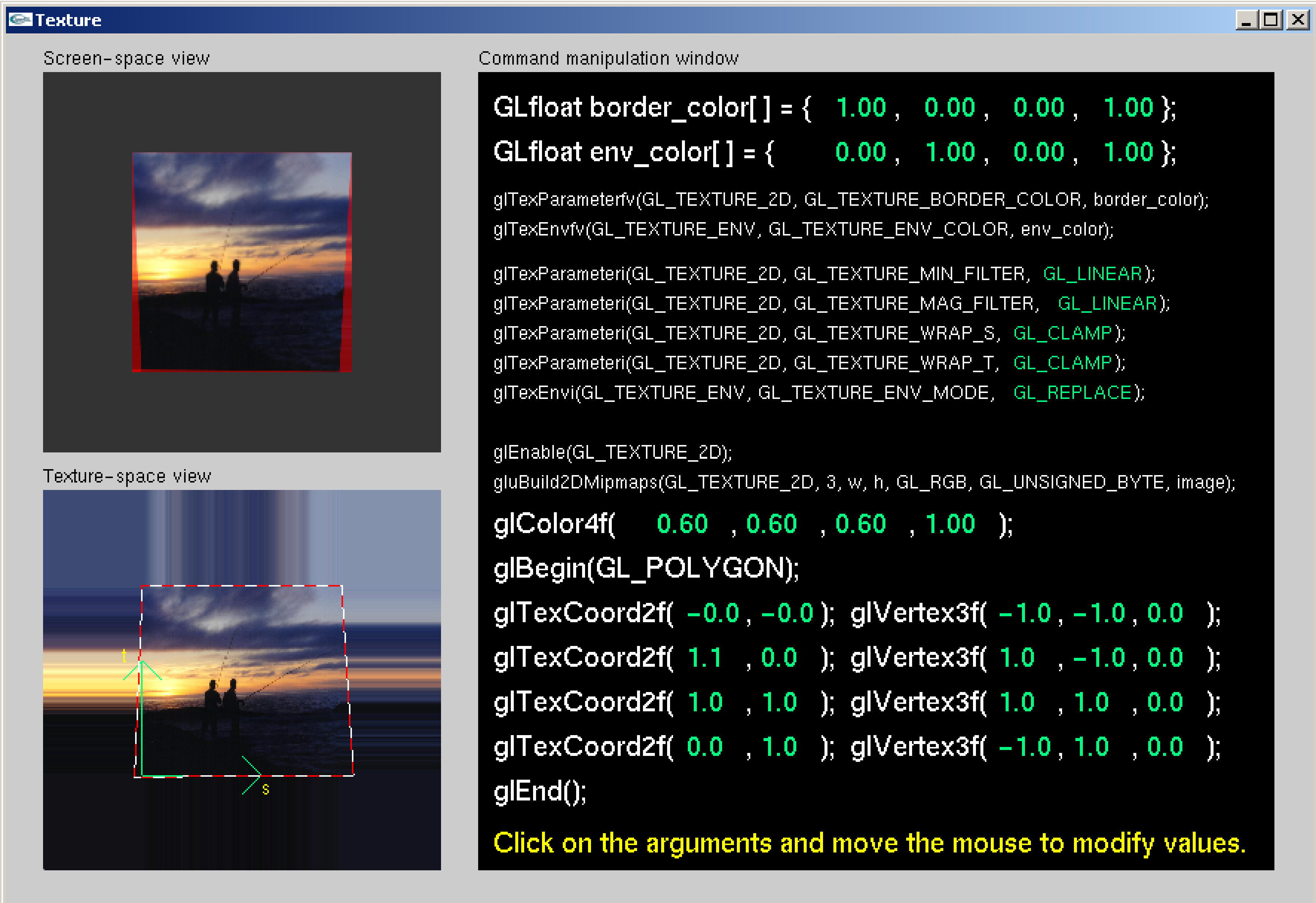
```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
```




```
// Draw Front of the Cube
glEnable(GL_TEXTURE_2D);
glBegin(GL_QUADS);
glTexCoord2d(-1, 2);  glVertex3d( 1.0, 1.0,
1.0);
glTexCoord2d(2, 2);  glVertex3d(-1.0, 1.0,
1.0);
glTexCoord2d(2, -1);  glVertex3d(-1.0,-1.0,
1.0);
glTexCoord2d(-1, -1);  glVertex3d( 1.0,-1.0,
1.0);
glEnd();
glDisable(GL_TEXTURE_2D);
```




```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
```



http://www.xmission.com/~nate/tutors.html

Texture Function

glTexEnv(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE)

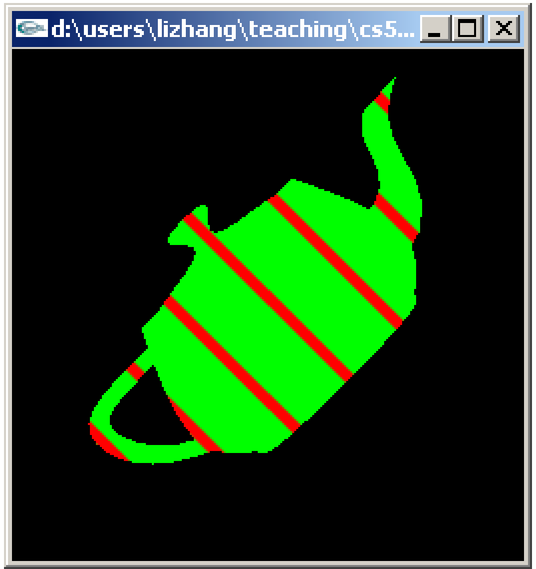
C = Ct, A = Af

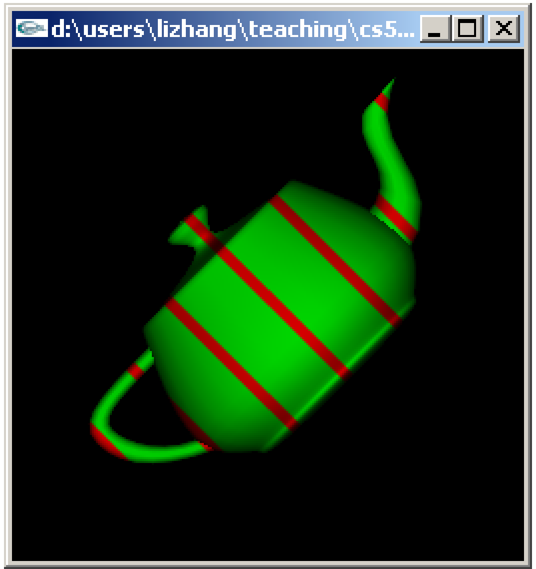
t: texture f:fragment

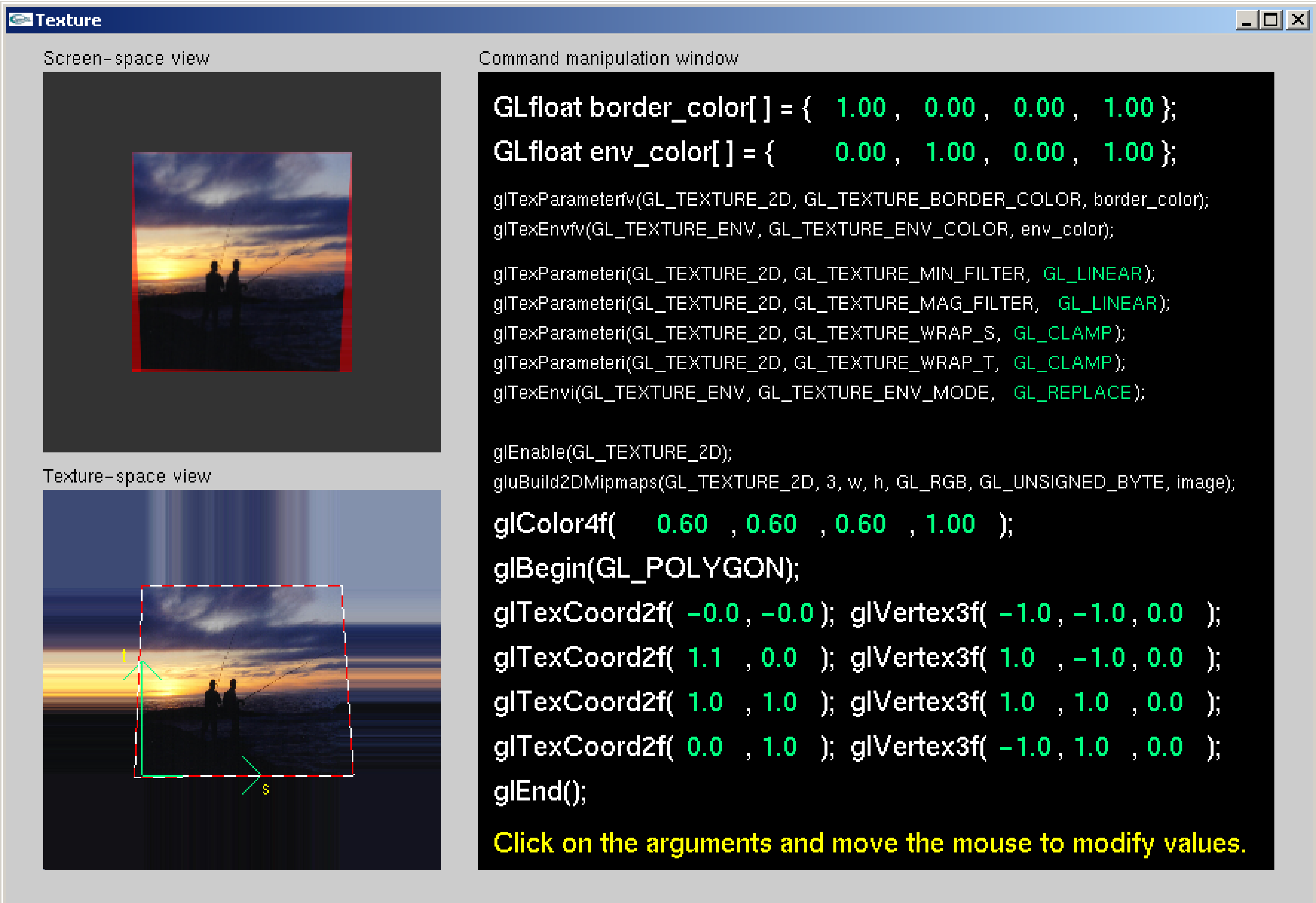
glTexEnv(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE)

C = Cf*Ct, A = Af

t: texture f:fragment







http://www.xmission.com/~nate/tutors.html

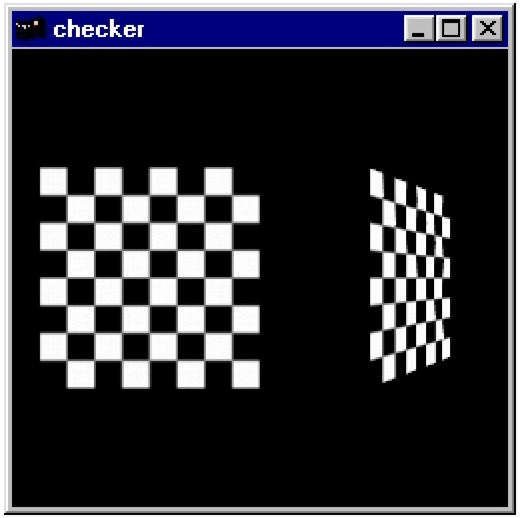
More texture function options

Base Internal Format	Replace Texture Function	Modulate Texture Function
GL_ALPHA	C = Cf, A = At	C = Cf, A = AfAt
GL_LUMINANCE	C = Lt, A = Af	C = CfLt, A = Af
GL_LUMINANCE_ALPHA	C = Lt, A = At	C = CfLt, A = AfAt
GL_INTENSITY	C = It, A = It	C = CfIt, A = AfIt
GL_RGB	C = Ct, A = Af	C = CfCt, A = Af
GL_RGBA	C = Ct, A = At	C = CfCt, A = AfAt

Table 9-3 : Decal and Blend Texture Function

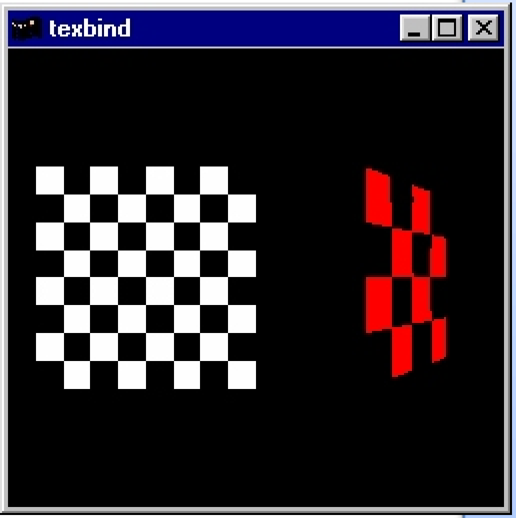
Base Internal Format	Decal Texture Function	Blend Texture Function
GL_ALPHA	undefined	C = Cf, A = AfAt
GL_LUMINANCE	undefined	C = Cf(1-Lt) + CcLt, A = Af
GL_LUMINANCE_ALPHA	undefined	C = Cf(1-Lt) + CcLt, A = AfAt
GL_INTENSITY	undefined	C = Cf(1-It) + CcIt, A = Af(1-It) + AcIt,
GL_RGB	C = Ct, A = Af	C = Cf(1-Ct) + CcCt, A = Af
GL_RGBA	C = Cf(1-At) + CtAt, A = Af	C = Cf(1-Ct) + CcCt, A = AfAt

Using one texture for different objects




```
static GLubyte image[64][64][4];
static GLuint  texname;

void display(void) {
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glEnable(GL_TEXTURE_2D);
        glBindTexture(GL_TEXTURE_2D, texName);

        glBegin(GL_QUADS);
        glTexCoord2f(0.0, 0.0); glVertex3f(-2.0, -1.0, 0.0);
        glTexCoord2f(0.0, 1.0); glVertex3f(-2.0, 1.0, 0.0);
        glTexCoord2f(1.0, 1.0); glVertex3f(0.0, 1.0, 0.0);
        glTexCoord2f(1.0, 0.0); glVertex3f(0.0, -1.0, 0.0);

        glTexCoord2f(0.0, 0.0); glVertex3f(1.0, -1.0, 0.0);
        glTexCoord2f(0.0, 1.0); glVertex3f(1.0, 1.0, 0.0);
        glTexCoord2f(1.0, 1.0); glVertex3f(2.41421, 1.0, -1.41421);
        glTexCoord2f(1.0, 0.0); glVertex3f(2.41421, -1.0, -1.41421);
        glEnd();
        glFlush();

        glDisable(GL_TEXTURE_2D);
}
```

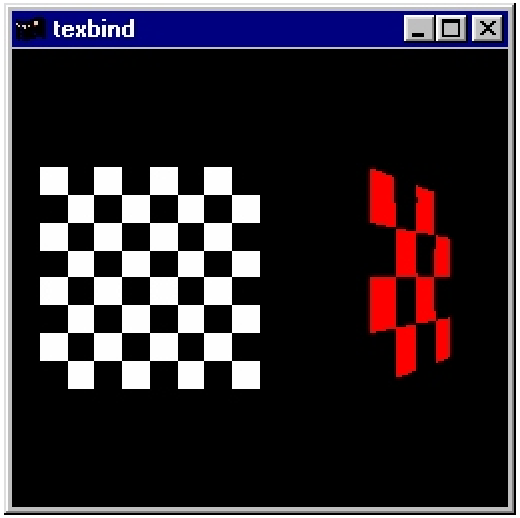
Using several texture images




```
static GLubyte image0[64][64][4], image1[64][64][4];
static GLuint  texname[2];

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, checkImageWidth, checkImageHeight, 0,
                GL_RGBA, GL_UNSIGNED_BYTE, image0);

        glBindTexture(GL_TEXTURE_2D, texName[1]);
        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, checkImageWidth, checkImageHeight, 0,
                GL_RGBA, GL_UNSIGNED_BYTE, image1);
}
```

Using several texture images

– Swithcing using glBindTexture(GL_TEXTURE_2D, texName);

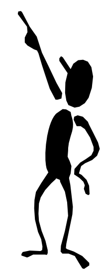



```
void display(void) {
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        glBindTexture(GL_TEXTURE_2D, texName[0]);
        glBegin(GL_QUADS);
        glTexCoord2f(0.0, 0.0); glVertex3f(-2.0, -1.0, 0.0);
        glTexCoord2f(0.0, 1.0);  glVertex3f(-2.0, 1.0, 0.0);
        glTexCoord2f(1.0, 1.0);  glVertex3f(0.0, 1.0, 0.0);
        glTexCoord2f(1.0, 0.0);  glVertex3f(0.0, -1.0, 0.0);
        glEnd();

        glBindTexture(GL_TEXTURE_2D, texName[1]);
        glBegin(GL_QUADS);
        glTexCoord2f(0.0, 0.0); glVertex3f(1.0, -1.0, 0.0);
        glTexCoord2f(0.0, 1.0); glVertex3f(1.0, 1.0, 0.0);
        glTexCoord2f(1.0, 1.0); glVertex3f(2.41421, 1.0, -1.41421);
        glTexCoord2f(1.0, 0.0); glVertex3f(2.41421, -1.0, -1.41421);
        glEnd();

        glFlush();
}
```

OpenGL Mipmap

Incorporating MIPmapping into OpenGL applications is surprisingly easy.

```
// Boilerplate Texture setup code
glTexImage2D(GL_TEXTURE_2D, 0, 4, texWidth, texHeight, 0, GL_RGBA,GL_UNSIGNED_BYTE, data);
gluBuild2DMipmaps(GL_TEXTURE_2D, 4, texWidth, texHeight, GL_RGBA, GL_UNSIGNED_BYTE, data);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
                                            GL_LINEAR_MIPMAP_LINEAR
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);
```



OpenGL also provides a facility for specifying the MIPmap image at each level using multiple calls to the glTexImage*D() function. This approach provides more control over filtering in the MIPmap construction.

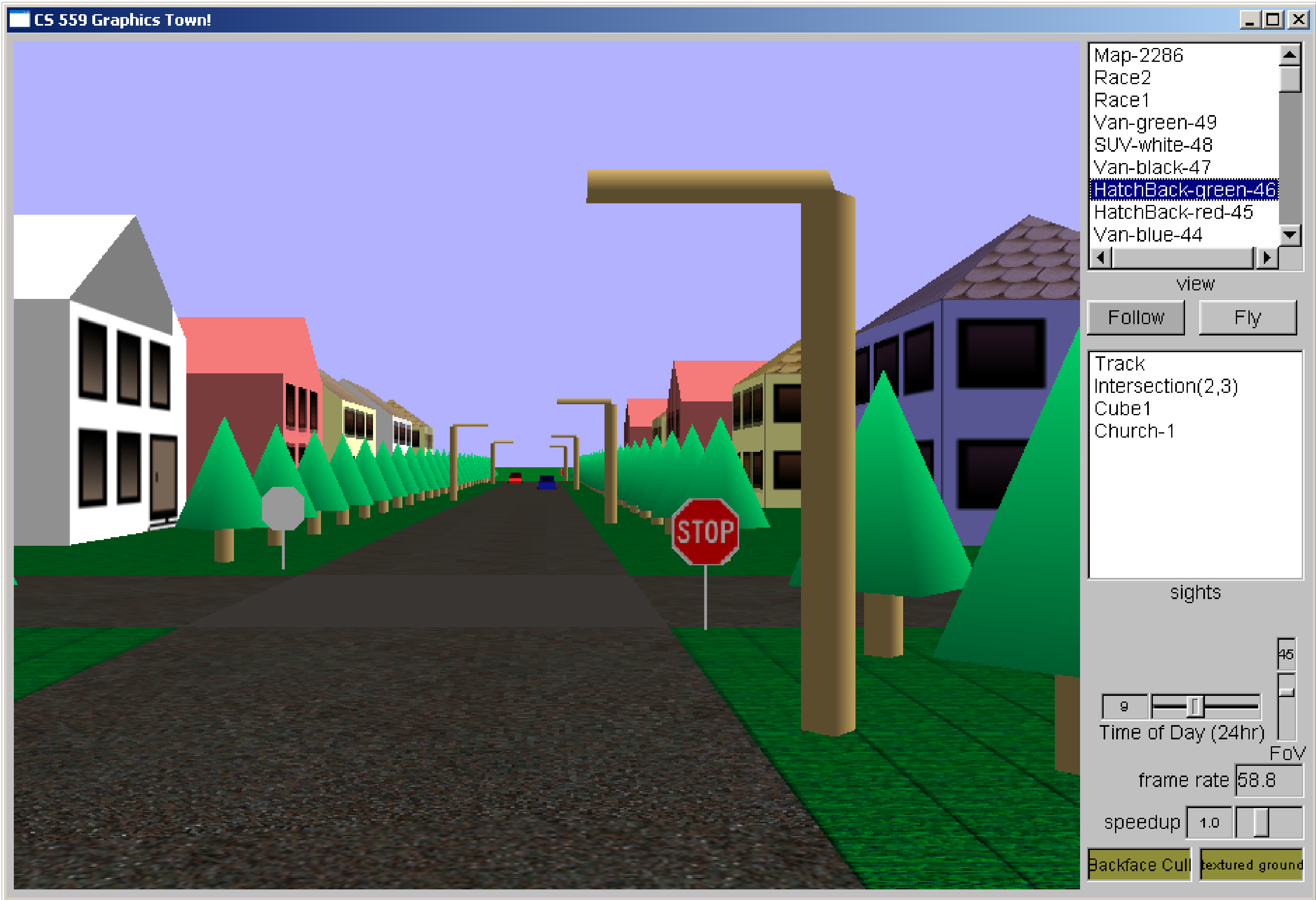
The gluBuildMipmaps() utility routine will automatically construct a mipmap from a given texture buffer. It will filter the texture using a simple box filter and then subsample it by a factor of 2 in each dimension. It repeats this process until one of the texture's dimensions is 1. Each texture ID, can have multiple levels associated with it. GL_LINEAR_MIPMAP_LINEAR trilinearly interploates between texture indices and MIPmap levels. Other options include GL_NEAREST_MIPMAP_NEAREST, GL_NEAREST_MIPMAP_LINEAR, and GL_LINEAR_MIPMAP_NEAREST.

Slides from Jingyi Yu

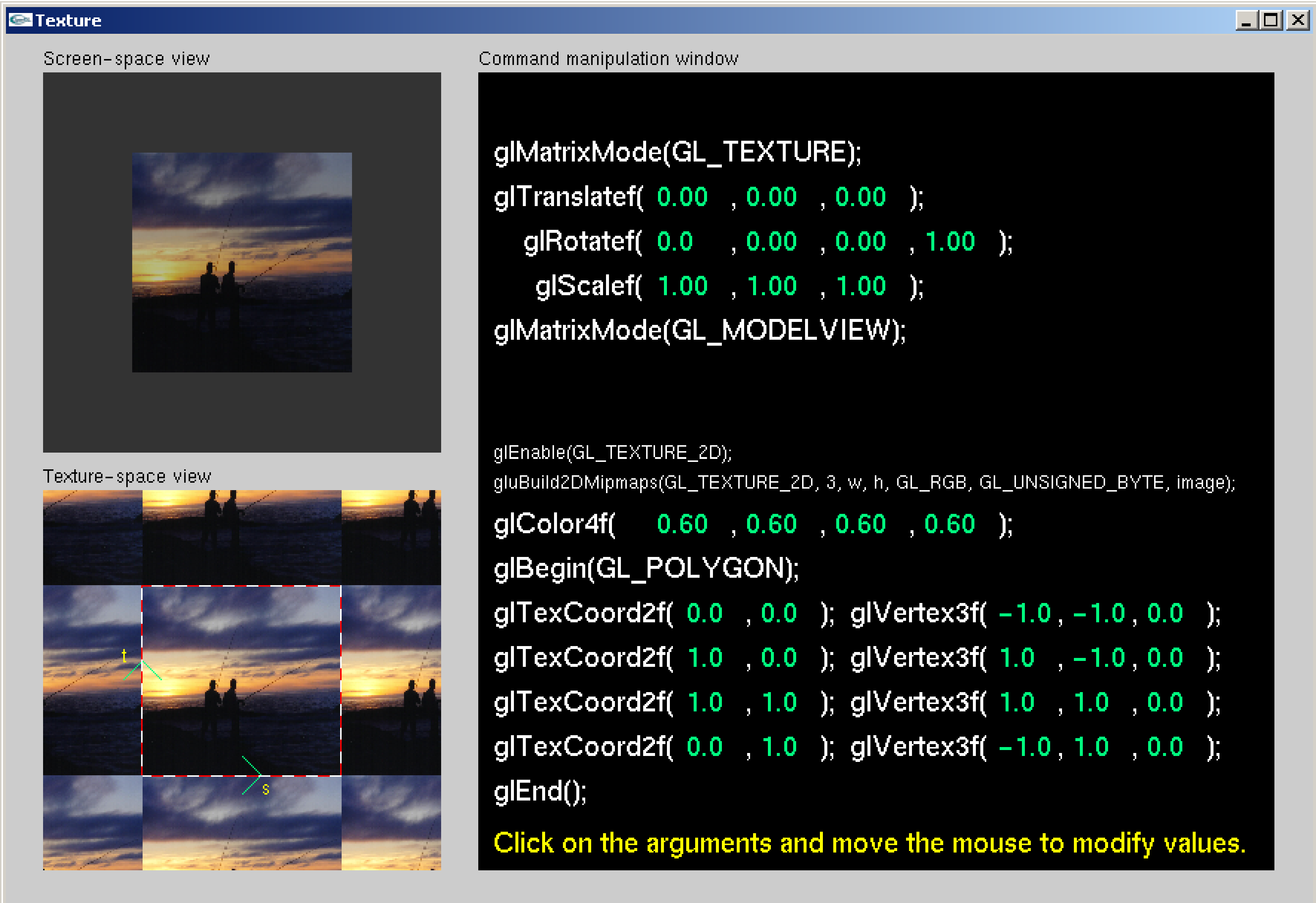
Initializing Texture Mapping

- Red book chapter on Texture mapping, Example 9-1
- All Red book example source code can be found at http://www.opengl.org/resources/code/samples/redbook/
- Course Tutorial 10, http://pages.cs.wisc.edu/~cs559-1/Tutorial10.htm

Project 3



CS 559 Graphics Town!
STOP
Map-2286
Race2
Race1
Van-green-49
SUV-white-48
Van-black-47
HatchBack-green-46
HatchBack-red-45
Van-blue-44
view
Follow
Fly
Track
Intersection(2,3)
Cube1
Church-1
sights
45
9
Time of Day (24hr)
FoV
frame rate
58.8
speedup
1.0
Backface Cull
textured ground



http://www.xmission.com/~nate/tutors.html

Texture animation

- Basic idea: treat texture coordinate like color
  - Moving water texture to simulate flow
  - zoom, rotation, and shearing image on a surface

Other Issues with Textures

- Tedious to specify texture coordinates for every triangle
- Textures are attached to the geometry
- Can't use just any image as a texture
- **The "texture" can't have projective distortions**
  Reminder: linear interpolation in image space is not equivalent to linear interpolation in 3-space (This is why we need "perspective-correct" texturing).
  The converse is also true.
- Makes it hard to use pictures as textures