CS559: Computer Graphics

Lecture 27: Texture Mapping

Li Zhang

Spring 2008

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

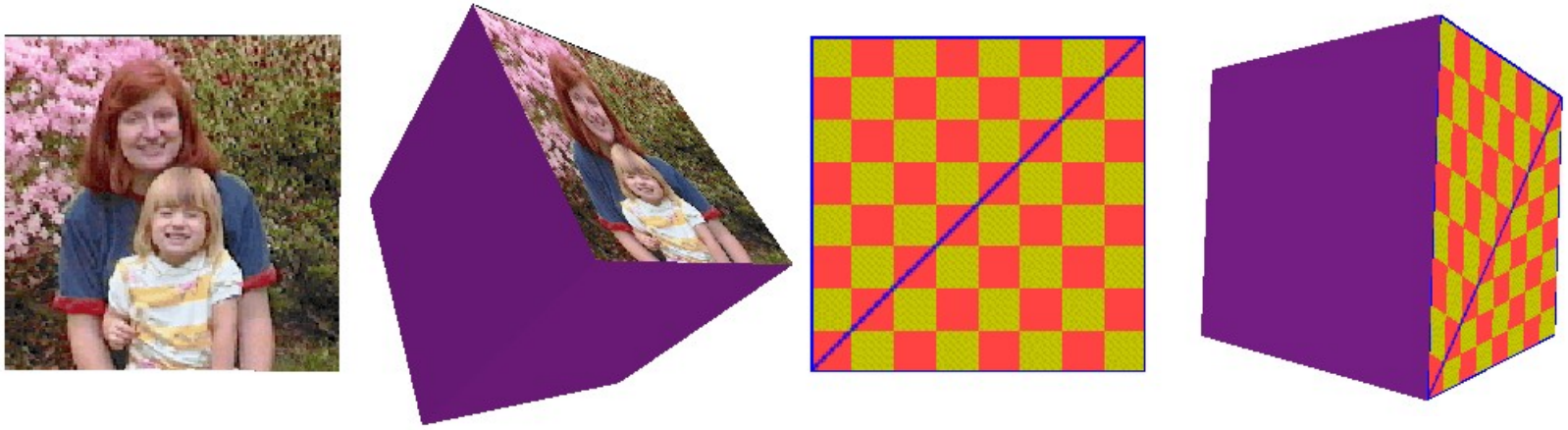
Today

- Continue on Texture mapping

- Reading
  - Redbook: Ch 9
  - (highly recommended) Moller and Haines: *Real-Time Rendering, 3e*, Ch 6
    - Linux: /p/course/cs559-lizhang/public/readings/6_texture.pdf
    - Windows: P:\course\cs559-lizhang\public\readings\6_texture.pdf
  - (optional) Shirley: Ch 11.4 – 11.8

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

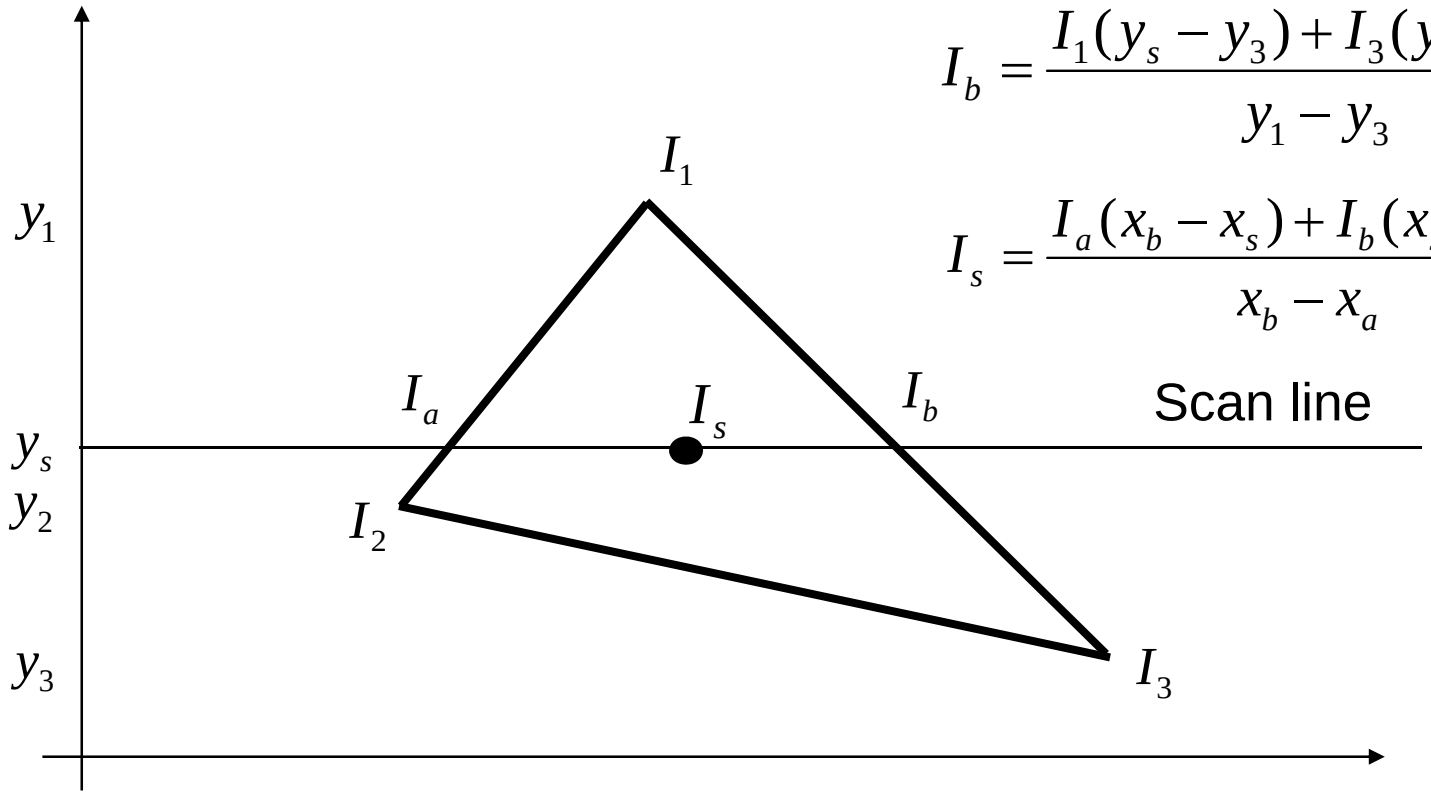


1st idea: Gouraud interp. of texcoords

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

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

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

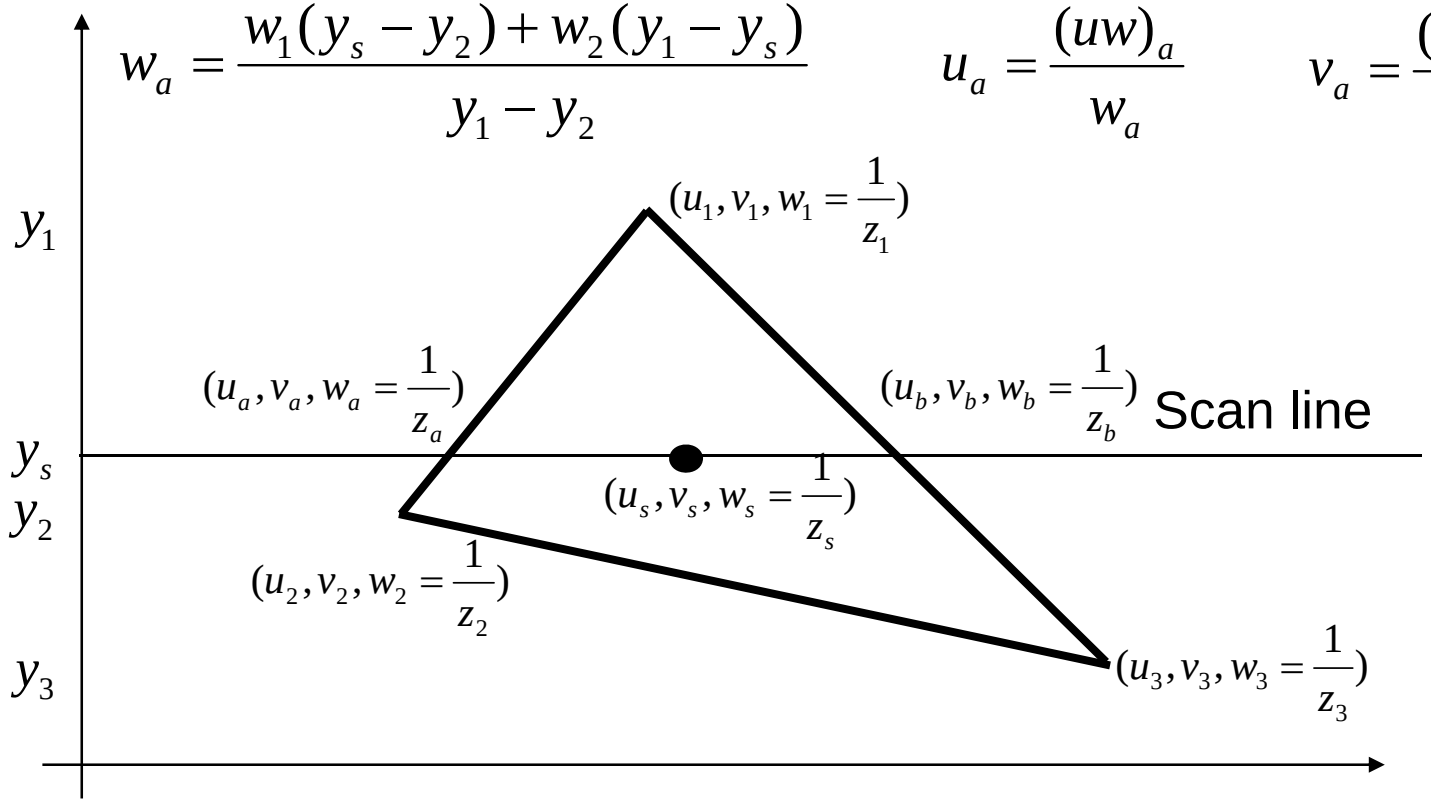


Replace I to uw, vw, and w, then compute (uw/w, and vw/w)

1st idea: Gouraud interp. of texcoords

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

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

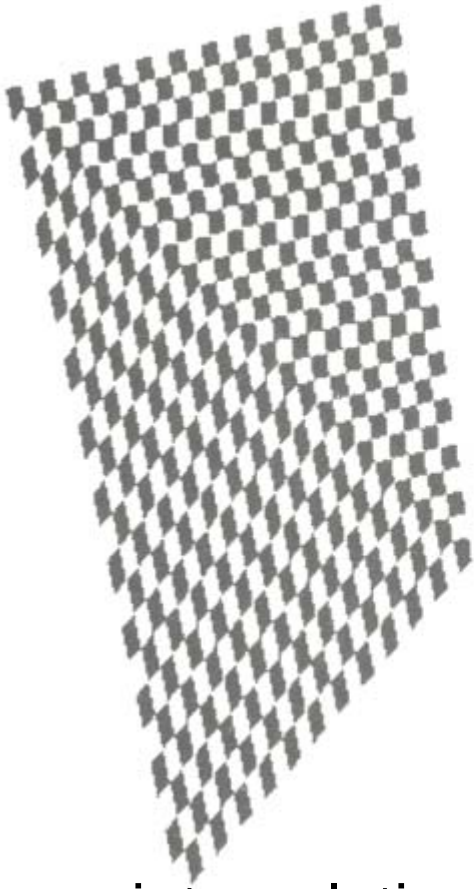


Do same thing for point b. From a and b, interpolate for s

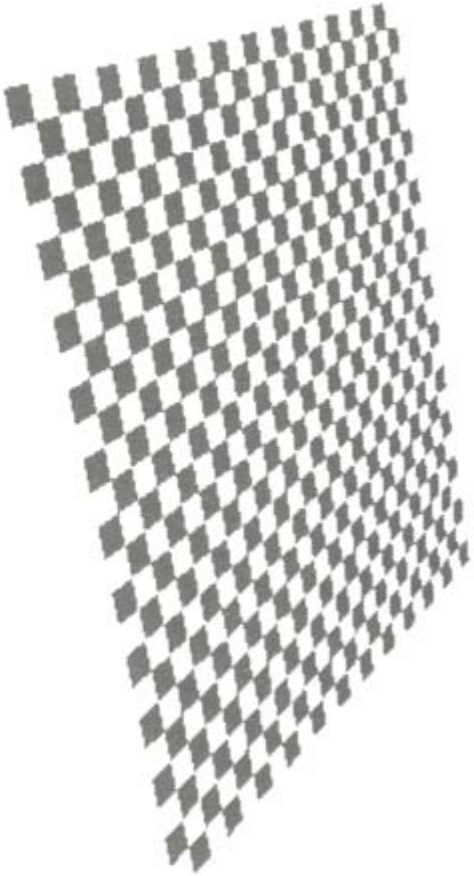
Perspective-Correct Interpolation

- In short…
  - Rather than interpolating *u* and *v* directly, interpolate u / z and v / z
    - These do interpolate correctly in screen space
    - Also need to interpolate $1/z$ and multiply per-pixel
  - In practice: we keep $w \propto 1 / z$
  - So…interpolate *uw* and *vw* and *w*, and compute $u = uw/w$ and $v = vw/w$ for each pixel
    - 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

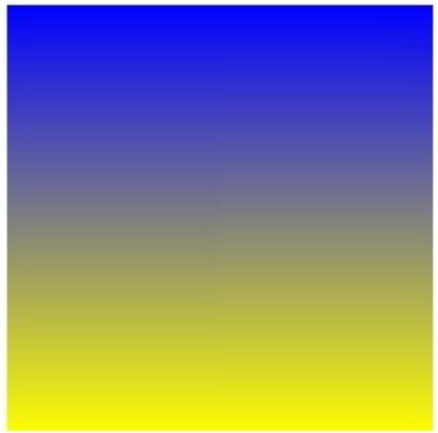
http://graphics.lcs.mit.edu/classes/6.837/F98/Lecture21/Slide14.html

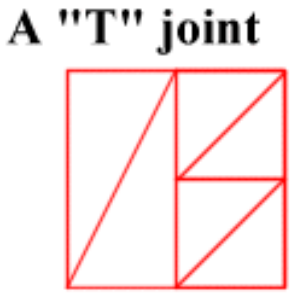
Hill Figure 8.42

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.

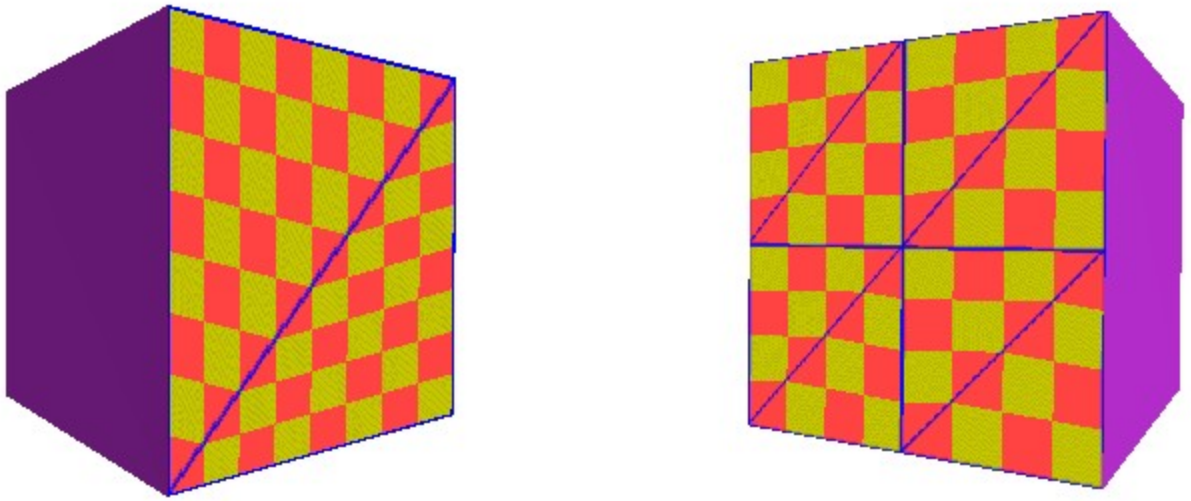
- When do we notice?
  - When switching between different levels-of-detail representations
  - At "T" joints.





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

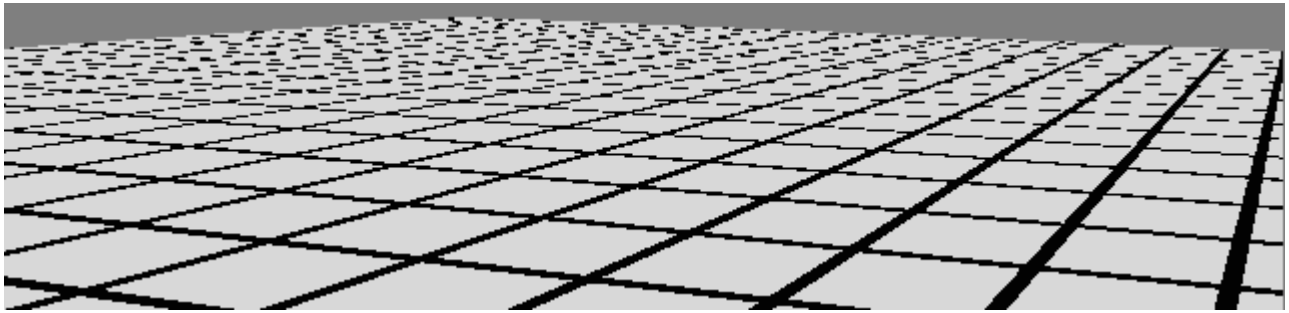
Outline

- 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

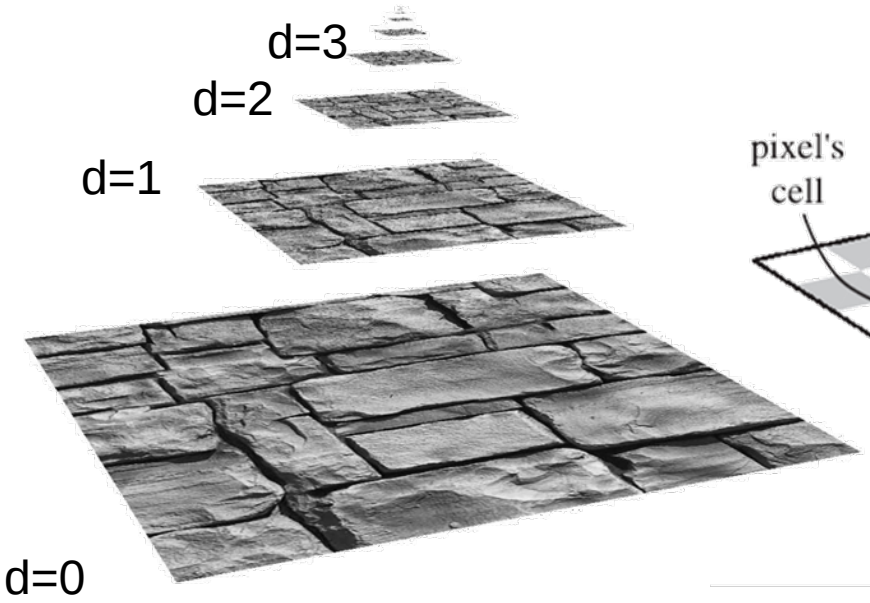
Texture Map Filtering

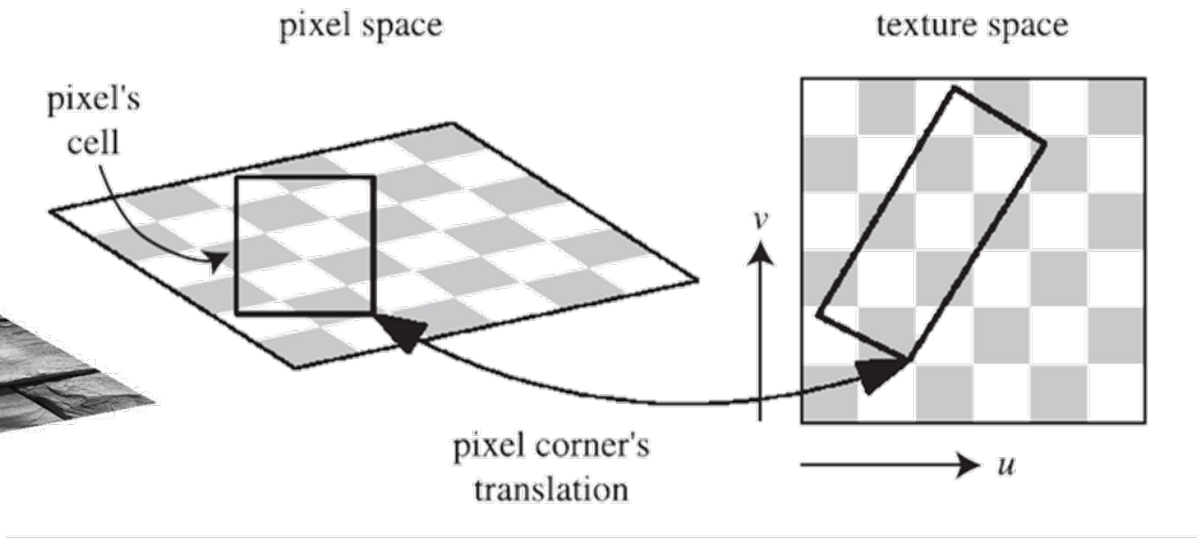
- Naive texture mapping aliases badly
- Look familiar?

```
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?

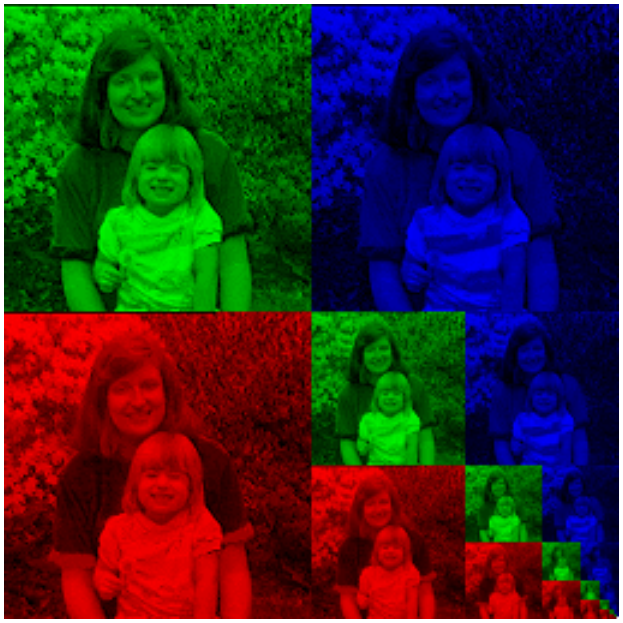
Option 1: use the longer edge of the quadrilateral formed by the pixel's cell to approximate the pixel's coverage

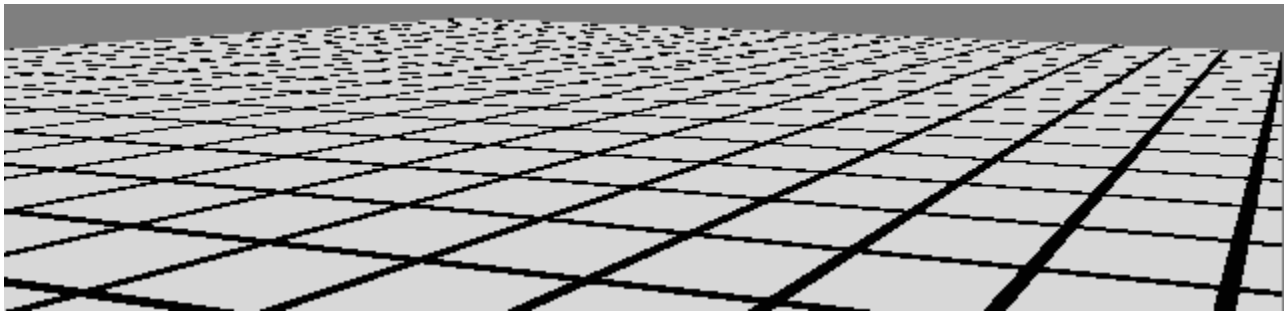
Option 2: use the max of |du/dx|, |du/dy|, |dv/x|, |dv/dy|

Then take logarithm

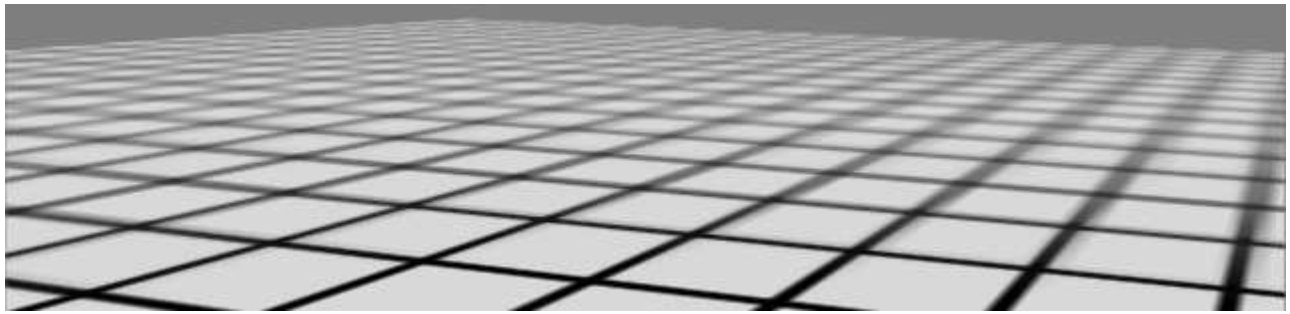
Storing MIP Maps

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



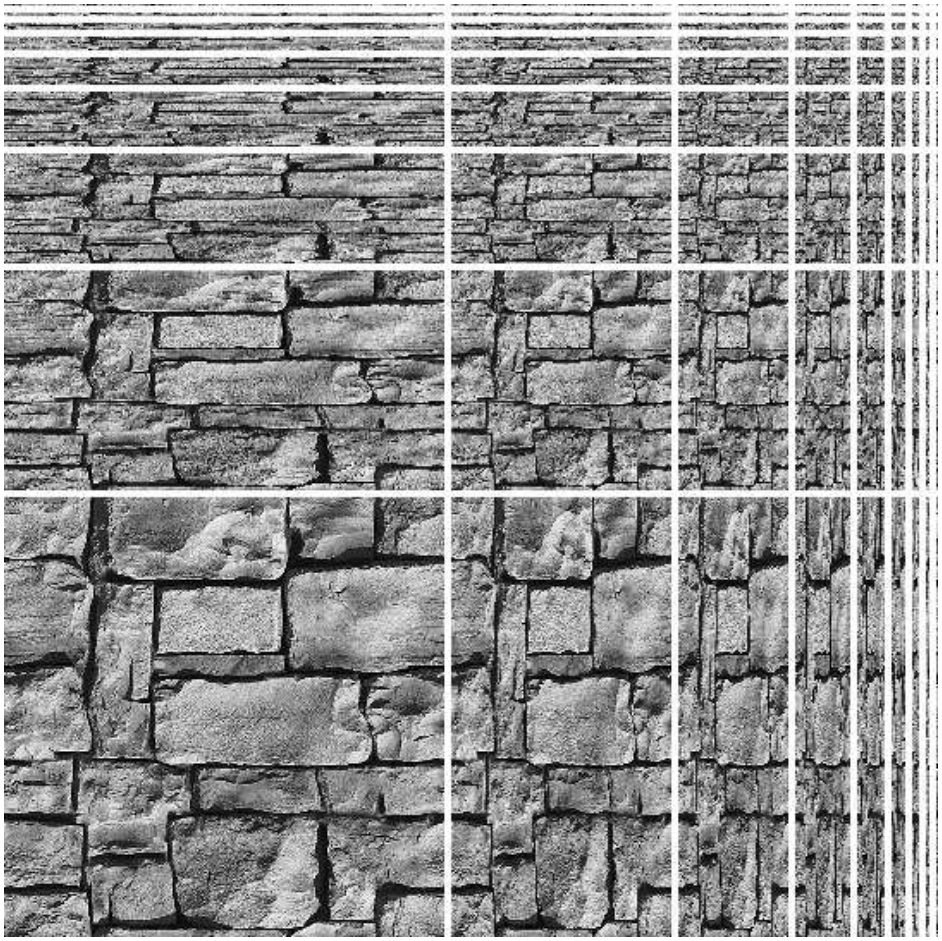


Nearest Neighbor Sampling



Mipmap Sampling

Ripmap



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

What's the memory overhead?

Summed Area Table

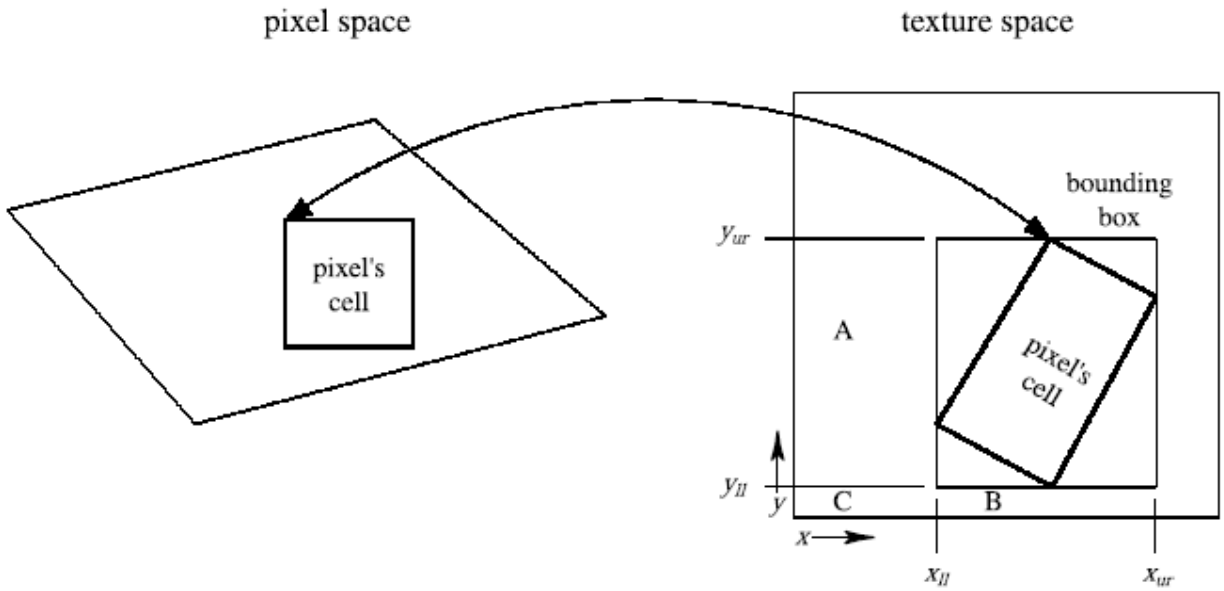


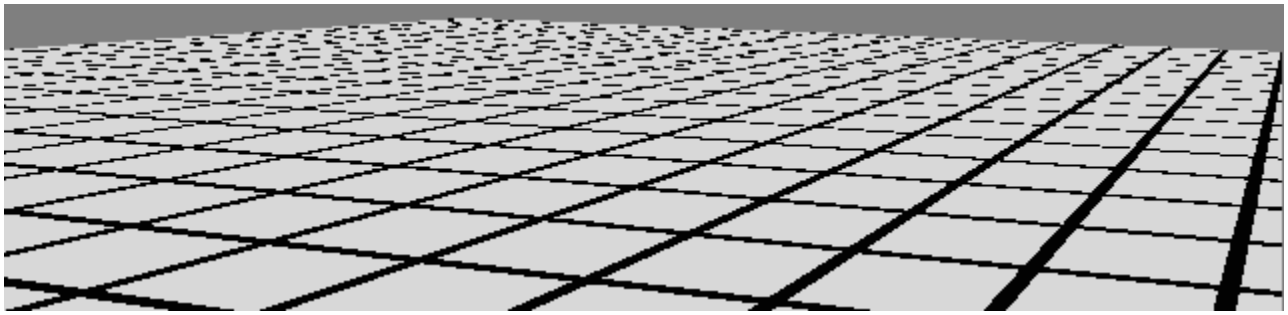
Figure 6.15: The pixel cell is back-projected onto the texture, bound by a rectangle, and the four corners of the rectangle are used to access the summed-area table.

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

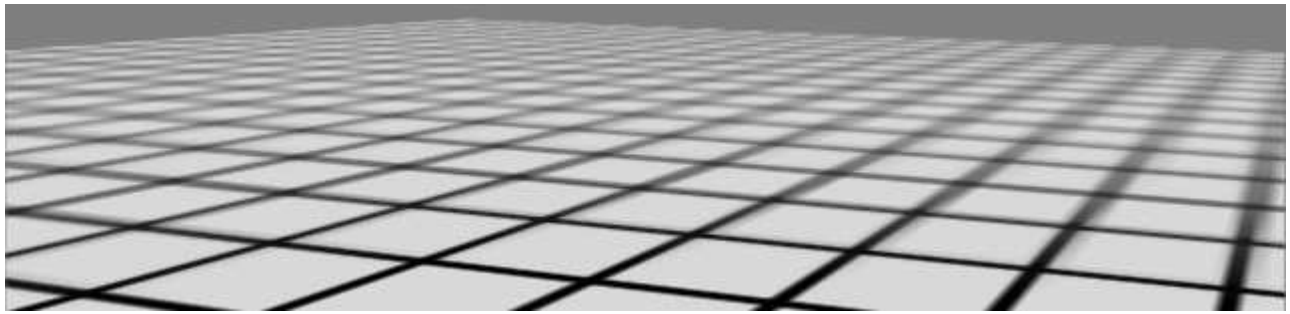
1	6	8	3
0	0	3	7
4	7	8	8
5	0	9	9

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

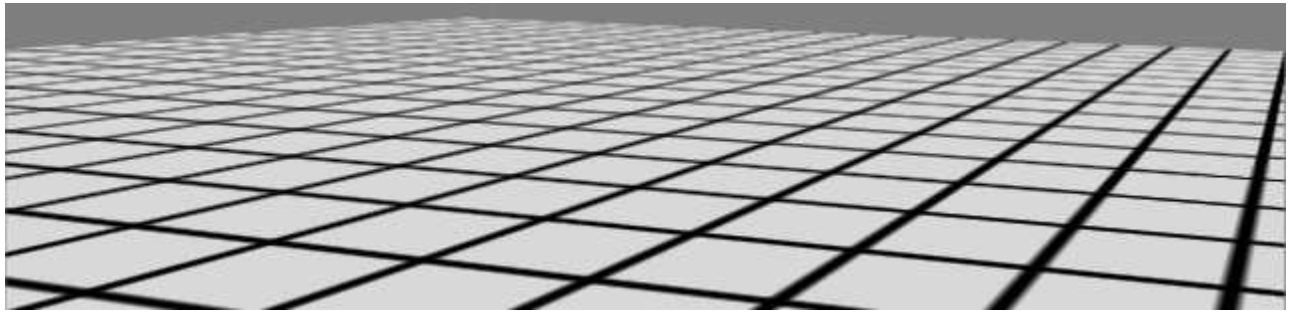
What's the memory overhead?



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

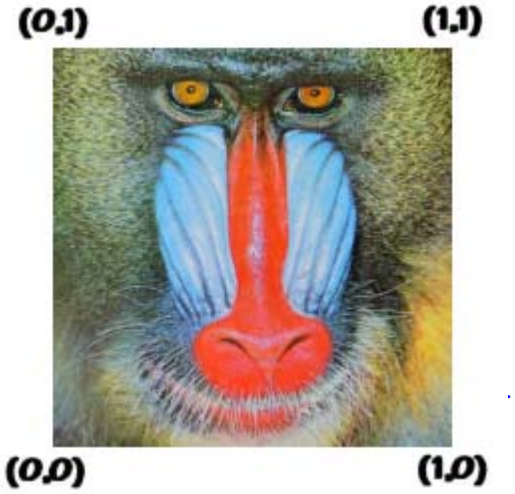
- Other filtering method, see *Real-Time Rendering* chapter 6.

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

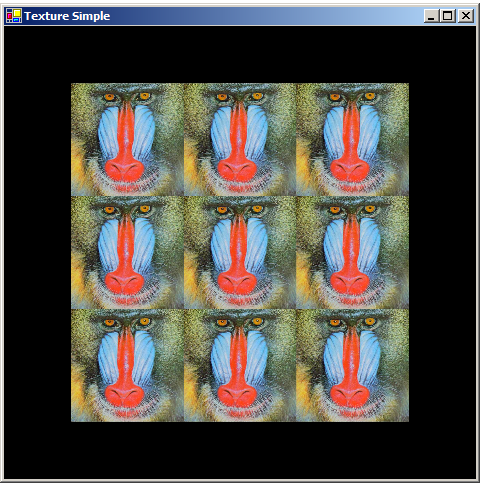
```
static GLubyte image[64][64][4];
static GLuint  texname;

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);
        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, image);
}
```

Level index in the Pyramid

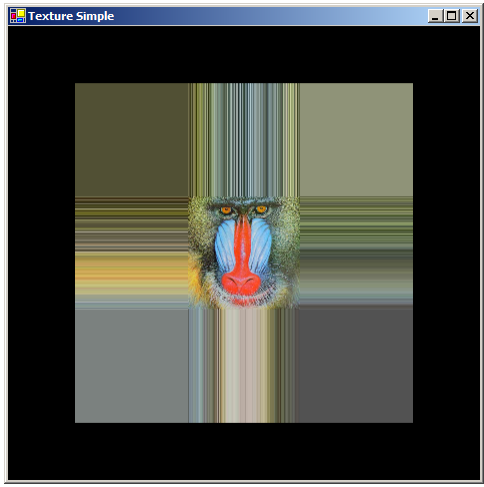
OpenGL Texture Peculiarities

- 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);
```



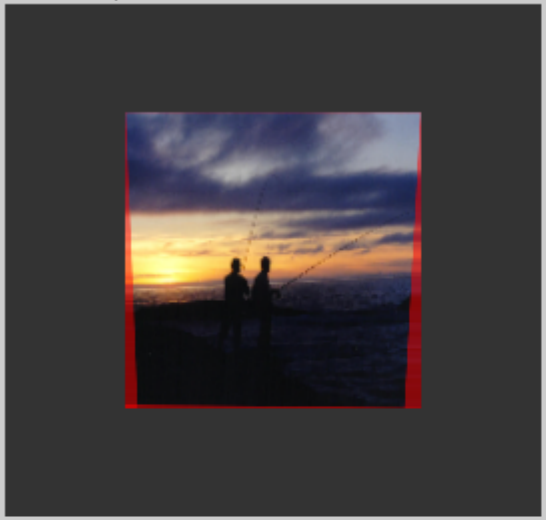

```
// Draw Front of the Cube
glEnable(GL_TEXTURE_2D);
glBegin(GL_QUADS);
glTexCoord2d(-1, 2);  glVertex3d( 1.0, 1.0, 1.0);
glTexCoord2d(2, 2);  glVertex3d(-1.0, 1.0, 1.0);
glTexCoord2d(2, -1);  glVertex3d(-1.0,-1.0, 1.0);
glTexCoord2d(-1, -1);  glVertex3d( 1.0,-1.0, 1.0);
glEnd();
glDisable(GL_TEXTURE_2D);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



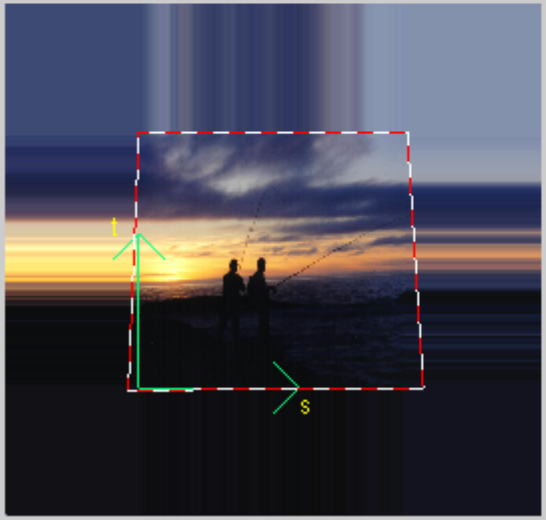

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
```

Texture

Screen-space view



Texture-space view



Command manipulation window

```
GLfloat border_color[ ] = {  1.00 ,  0.00 ,  0.00 ,  1.00 };
GLfloat env_color[ ] = {     0.00 ,  1.00 ,  0.00 ,  1.00 };

glTexParameterfv(GL_TEXTURE_2D, GL_TEXTURE_BORDER_COLOR, border_color);
glTexEnvfv(GL_TEXTURE_ENV, GL_TEXTURE_ENV_COLOR, env_color);

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,  GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,  GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,  GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,  GL_CLAMP);
glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,  GL_REPLACE);

glEnable(GL_TEXTURE_2D);
gluBuild2DMipmaps(GL_TEXTURE_2D, 3, w, h, GL_RGB, GL_UNSIGNED_BYTE, image);
glColor4f(  0.60  , 0.60  , 0.60  , 1.00  );
glBegin(GL_POLYGON);
glTexCoord2f( -0.0 , -0.0 ); glVertex3f( -1.0 , -1.0 , 0.0  );
glTexCoord2f( 1.1  , 0.0  ); glVertex3f( 1.0  , -1.0 , 0.0  );
glTexCoord2f( 1.0  , 1.0  ); glVertex3f( 1.0  , 1.0  , 0.0  );
glTexCoord2f( 0.0  , 1.0  ); glVertex3f( -1.0 , 1.0  , 0.0  );
glEnd();
```

Click on the arguments and move the mouse to modify values.

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