CS 4204 Computer Graphics

Texture Mapping

Yong Cao

Virginia Tech

Objectives

Introduce Mapping Methods

- Texture Mapping
- Environment Mapping
- Bump Mapping

Consider basic strategies

- Forward vs backward mapping
- Point sampling vs area averaging

The Limits of Geometric Modeling

Although graphics cards can render over 10 million polygons per second, that number is insufficient for many phenomena

- Clouds
- Grass
- Terrain
- Skin

Modeling an Orange

Consider the problem of modeling an orange (the fruit)

Start with an orange-colored sphere

- Too simple

Replace sphere with a more complex shape

- Does not capture surface characteristics (small dimples)
- Takes too many polygons to model all the dimples

Modeling an Orange (2)

Take a picture of a real orange, scan it, and "paste" onto simple geometric model

- This process is known as texture mapping

Still might not be sufficient because resulting surface will be smooth

- Need to change local shape
- Bump mapping

Three Types of Mapping

Texture Mapping

- Uses images to fill inside of polygons

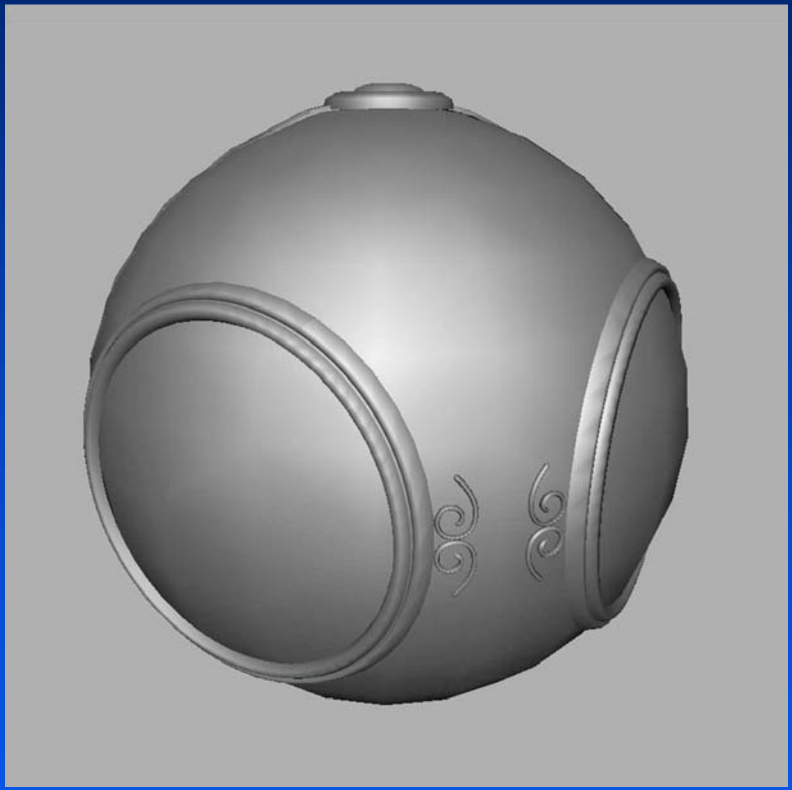
Environment (reflection mapping)

- Uses a picture of the environment for texture maps
- Allows simulation of highly specular surfaces

Bump mapping

- Emulates altering normal vectors during the rendering process

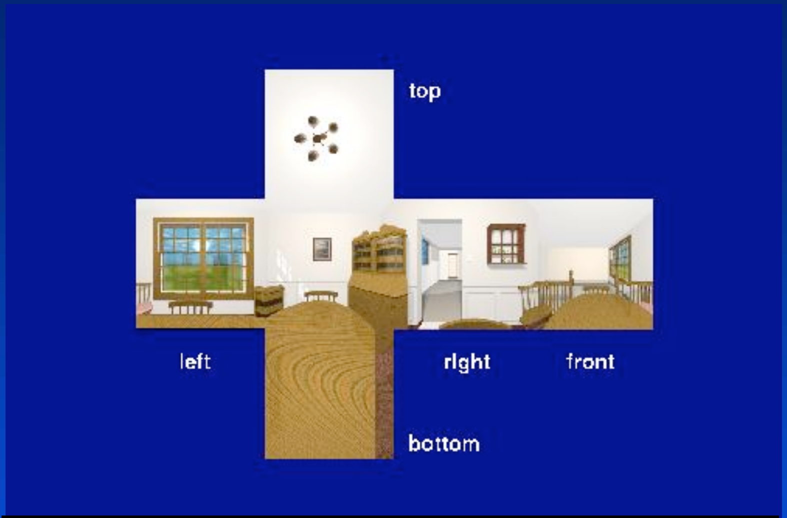
Texture Mapping



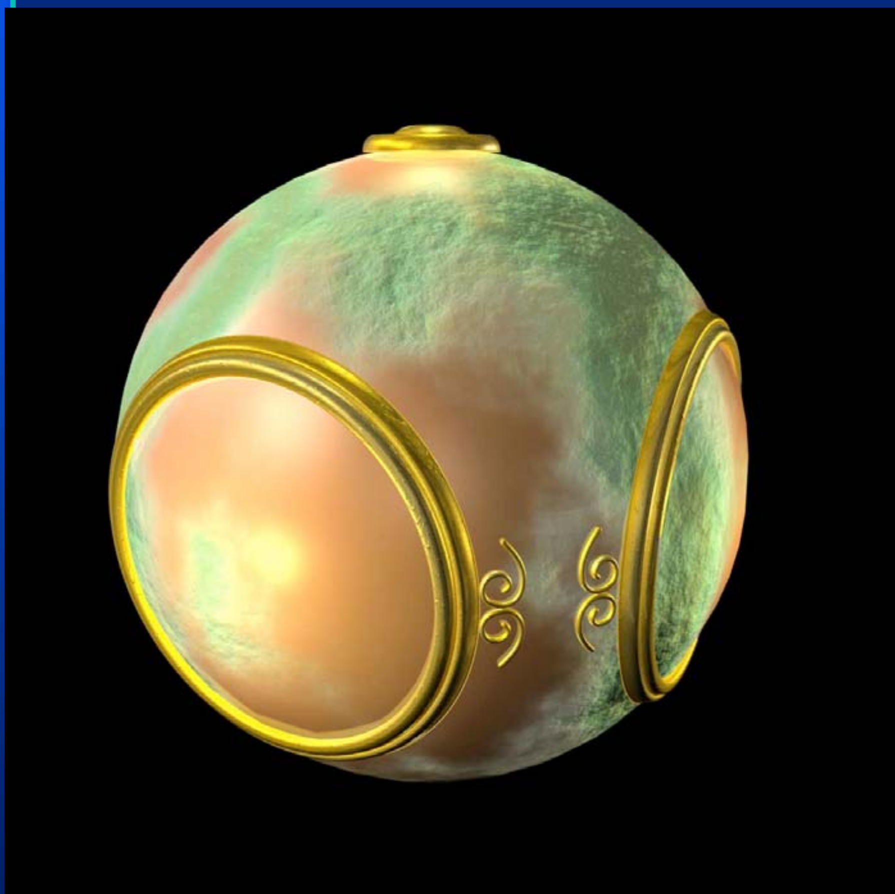
geometric model

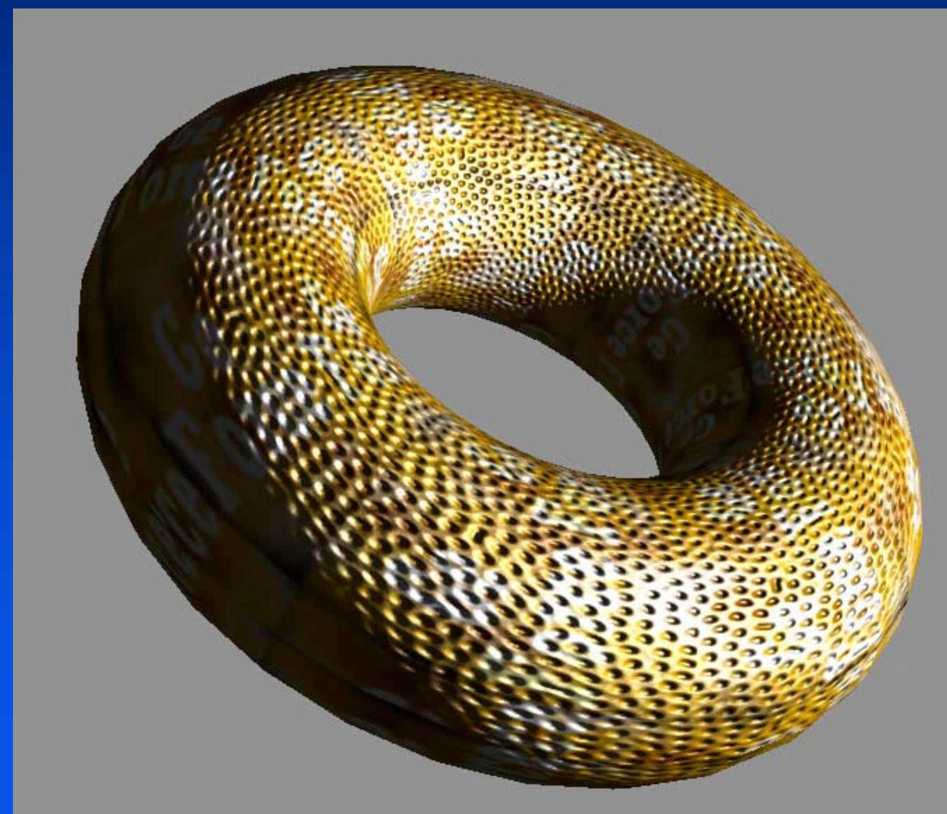
texture mapped

Environment Mapping



Bump Mapping

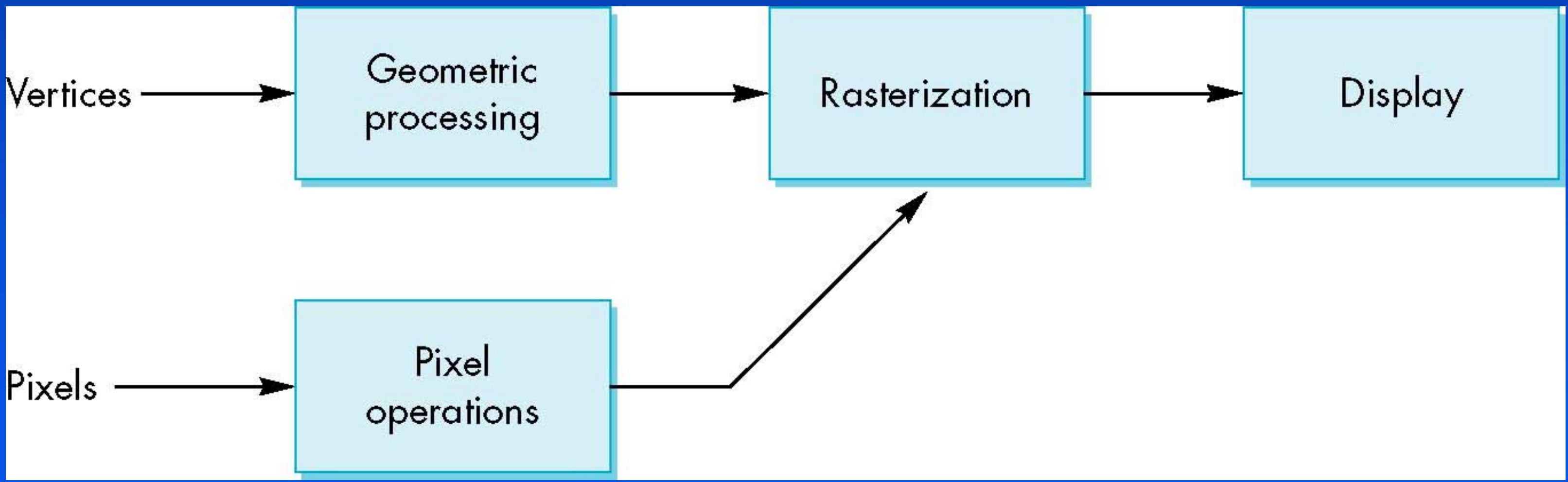




Where does mapping take place?

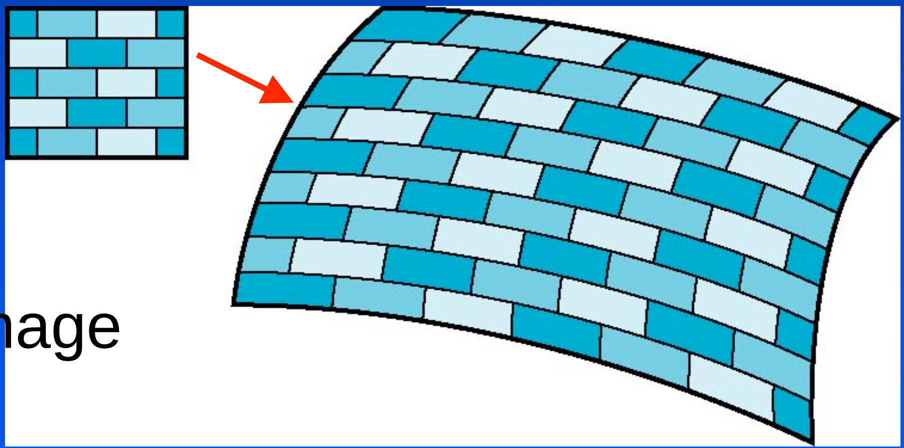
Mapping techniques are implemented at the end of the rendering pipeline

- Very efficient because few polygons make it past the clipper



Is it simple?

Although the idea is simple---map an image to a surface---there are 3 or 4 coordinate systems involved



2D image

3D surface

Coordinate Systems

Parametric coordinates

- May be used to model curves and surfaces

Texture coordinates

- Used to identify points in the image to be mapped

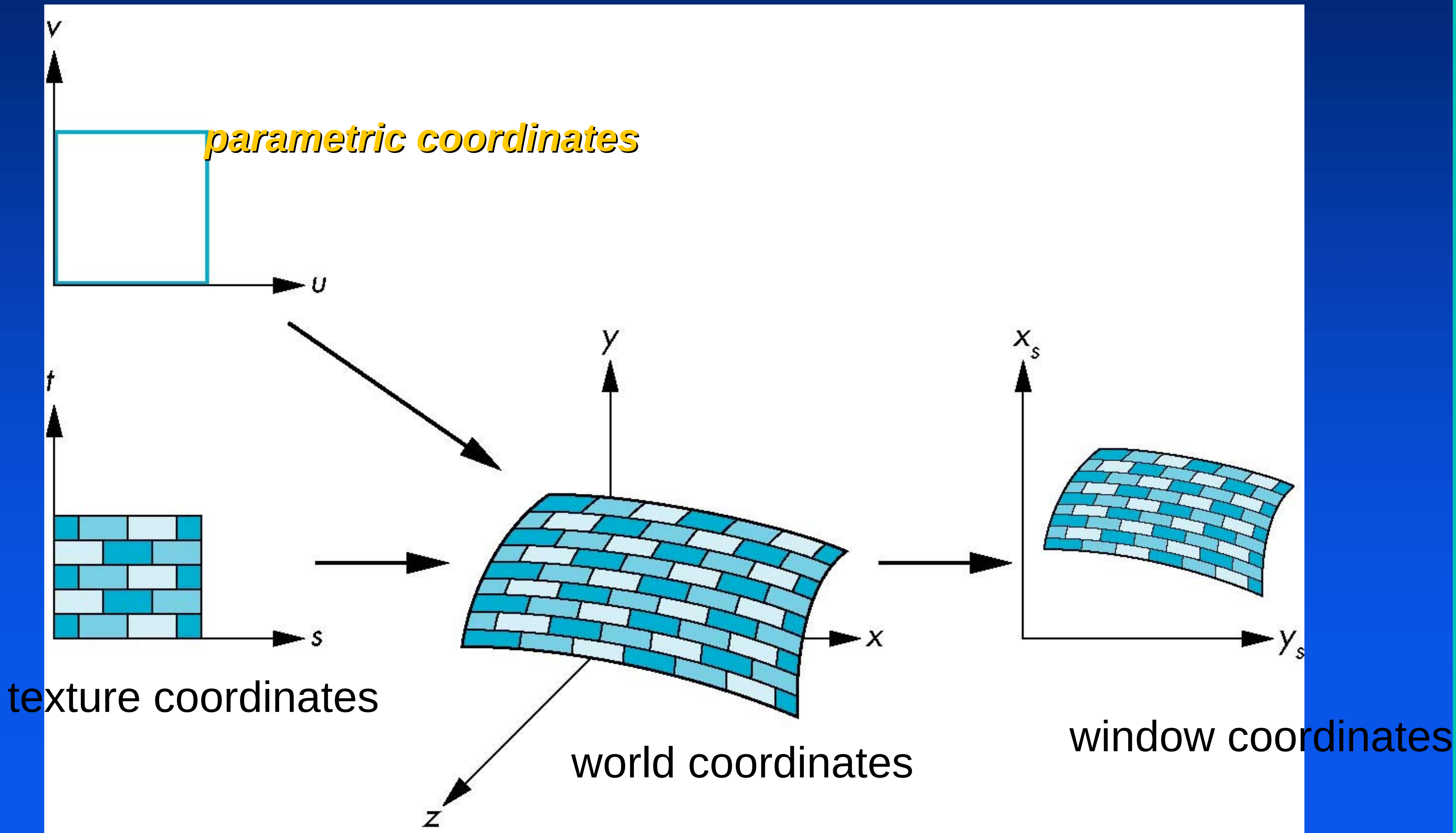
Object or World Coordinates

- Conceptually, where the mapping takes place

Window Coordinates

- Where the final image is really produced

Texture Mapping



Mapping Functions

Basic problem is how to find the maps

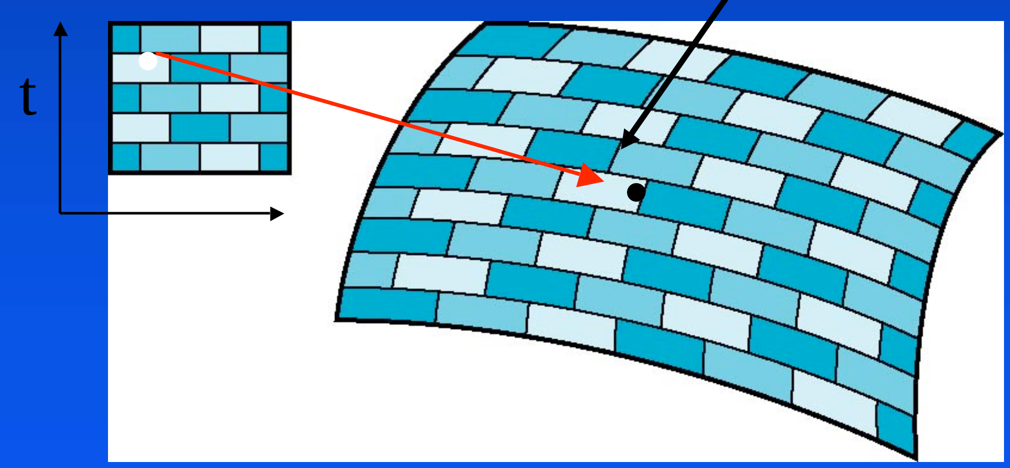
Consider mapping from texture coordinates to a point a surface

Appear to need three functions

x = x(s,t)

y = y(s,t)

z = z(s,t)



But we really want to go the other way

Backward Mapping

We really want to go backwards

- Given a pixel, we want to know to which point on an object it corresponds
- Given a point on an object, we want to know to which point in the texture it corresponds

Need a map of the form

$$s = s(x,y,z)$$

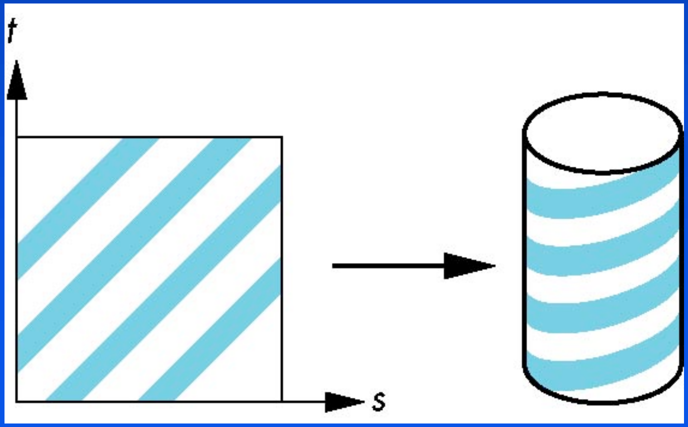
$$t = t(x,y,z)$$

Such functions are difficult to find in general

Two-part mapping

One solution to the mapping problem is to first map the texture to a simple intermediate surface

Example: map to cylinder



Cylindrical Mapping

parametric cylinder

$$x = r \cos 2\pi\, u$$
$$y = r \sin 2\pi u$$
$$z = v/h$$

maps rectangle in u,v space to cylinder
of radius r and height h in world coordinates

$$s = u$$
$$t = v$$

maps from texture space

Spherical Map

We can use a parametric sphere

$$x = r \cos 2\pi u$$
$$y = r \sin 2\pi u \cos 2\pi v$$
$$z = r \sin 2\pi u \sin 2\pi v$$

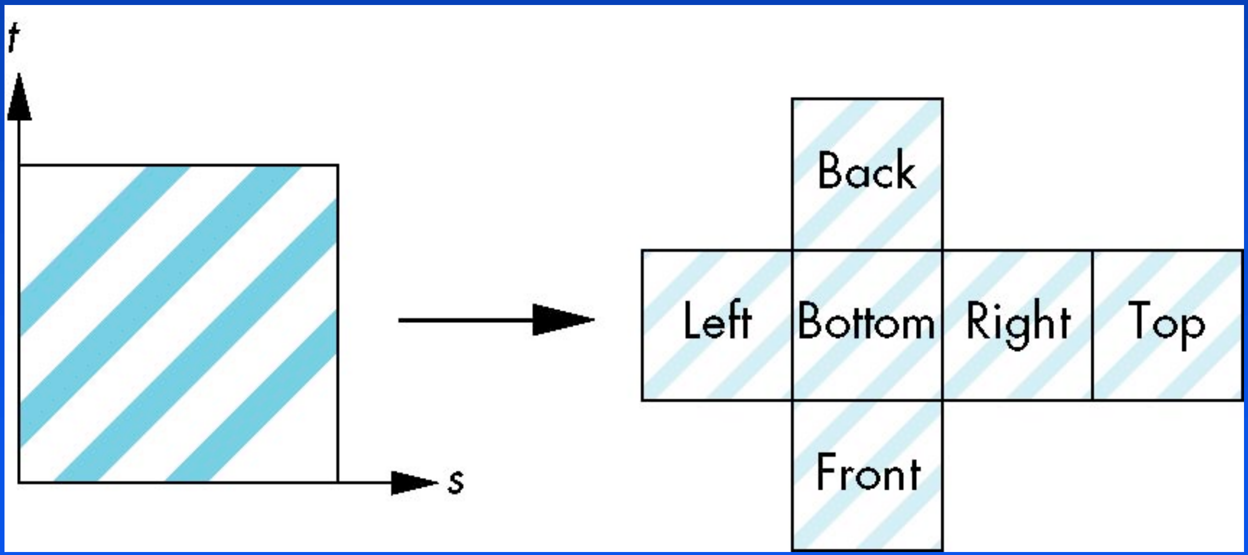
in a similar manner to the cylinder
but have to decide where to put
the distortion

Spheres are used in environmental maps

Box Mapping

Easy to use with simple orthographic projection

Also used in environment maps



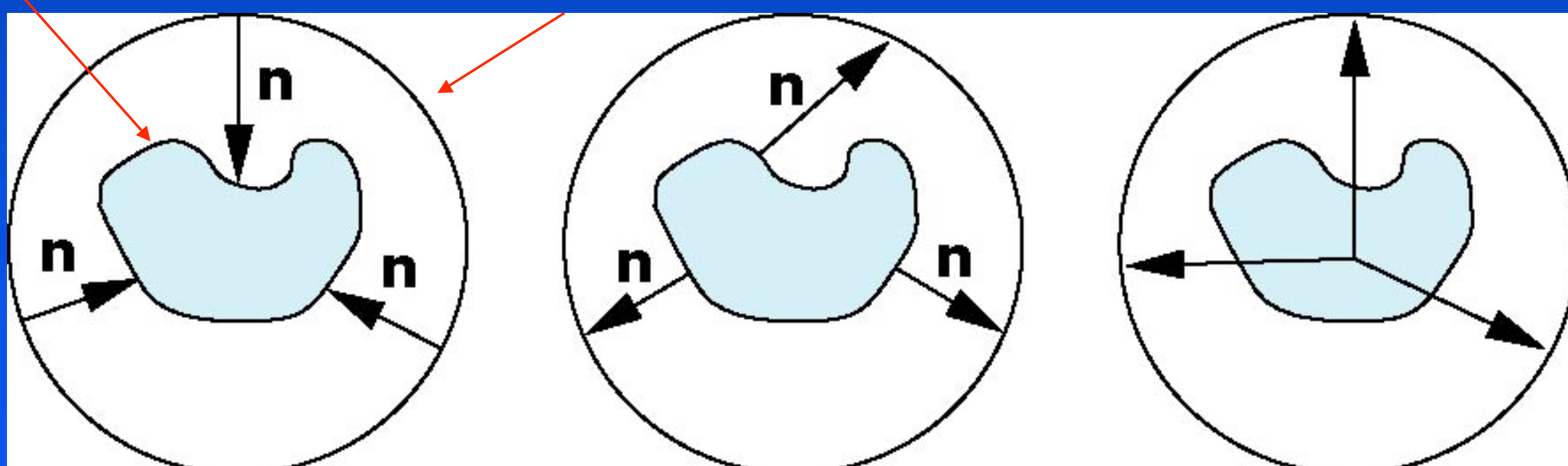
Second Mapping

Map from intermediate object to actual object

- Normals from intermediate to actual
- Normals from actual to intermediate
- Vectors from center of intermediate

actual

intermediate

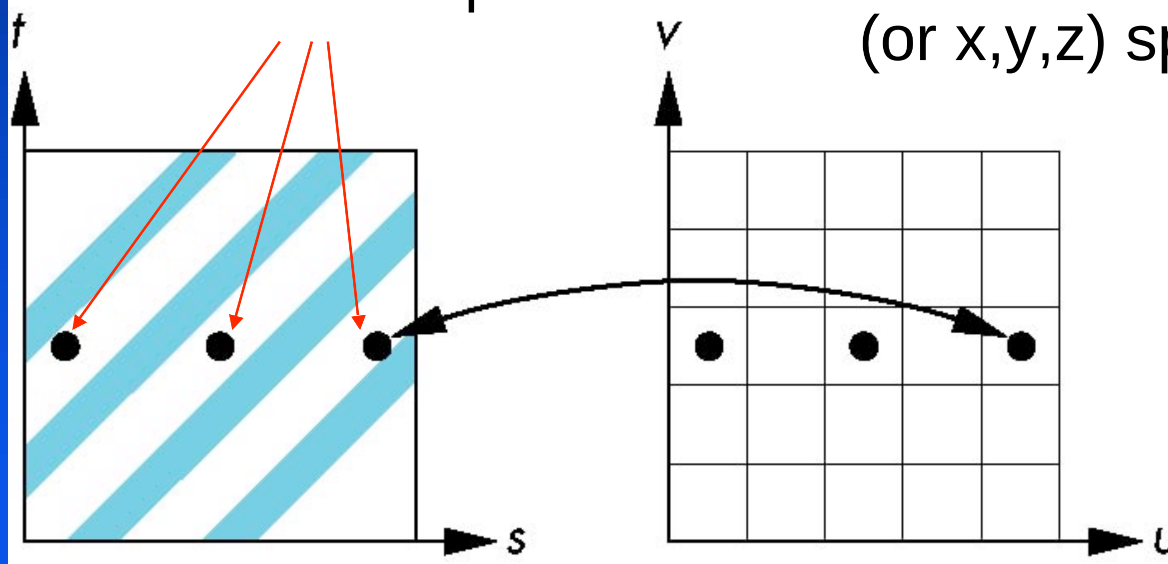


Aliasing

Point sampling of the texture can lead to aliasing errors

miss blue stripes

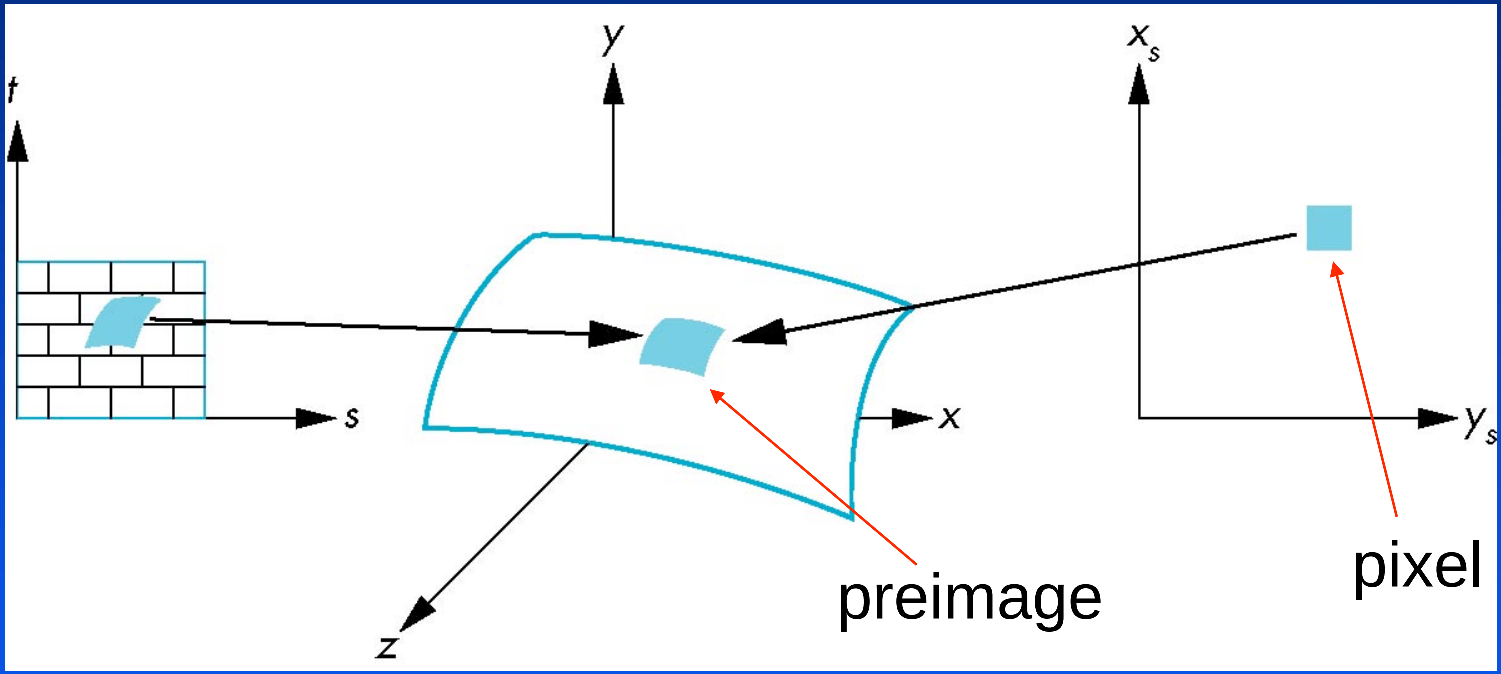
point samples in u,v
(or x,y,z) space



point samples in texture space

Area Averaging

***A better but slower option is to use* area averaging**



Note that *preimage* of pixel is curved

Objectives

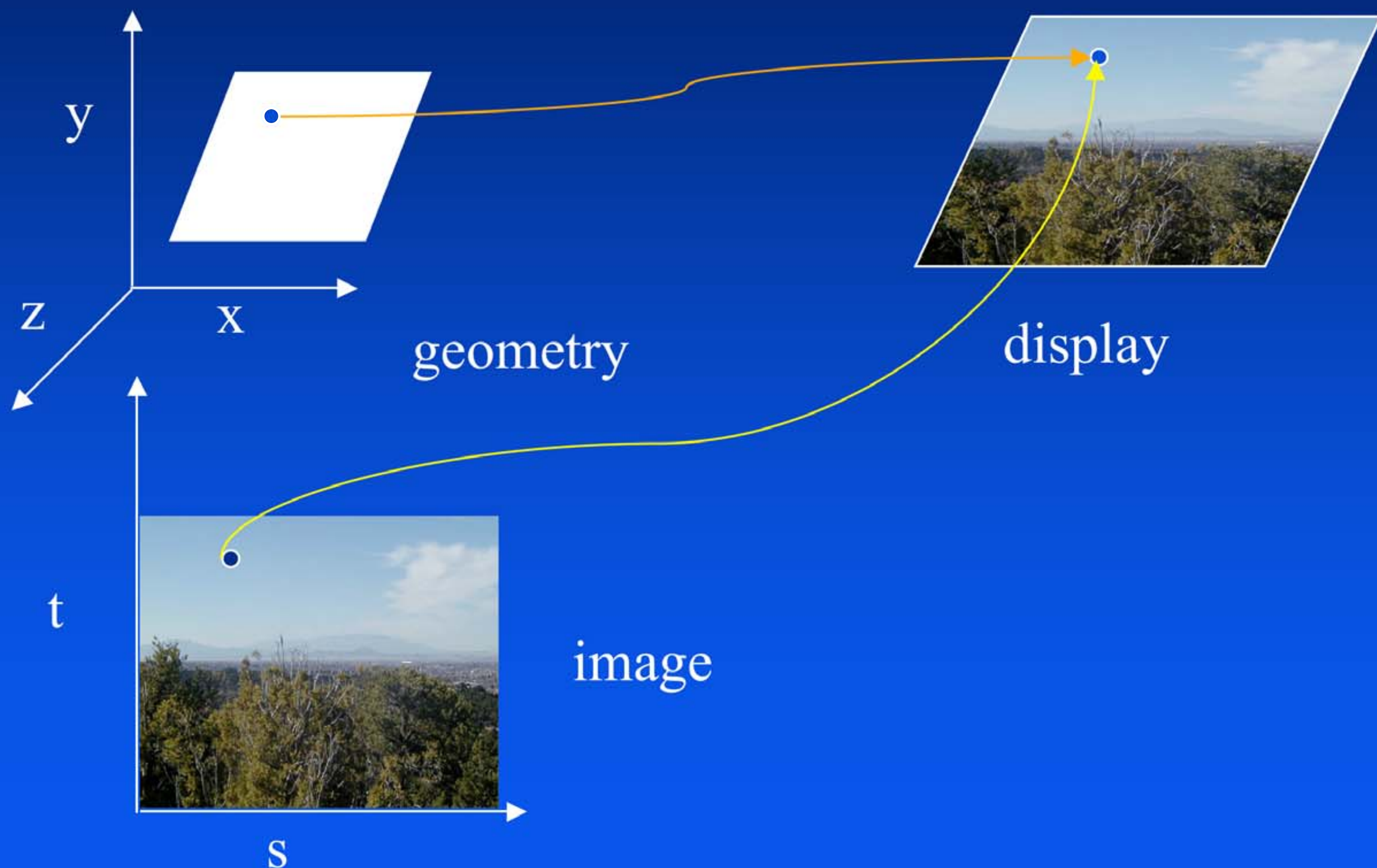
Introduce the OpenGL texture functions and options

Basic Stragegy

Three steps to applying a texture

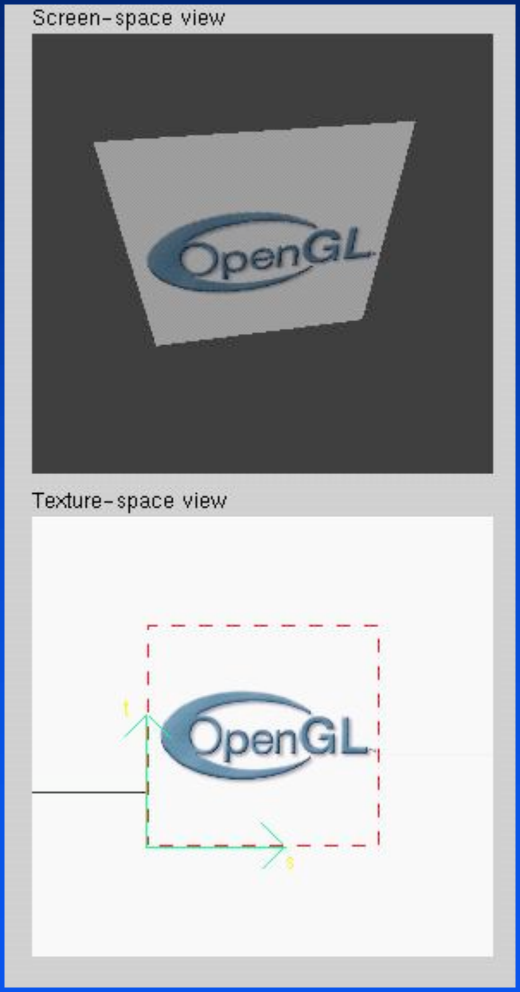
1. specify the texture
   – *read or generate image*
   – *assign to texture*
   – *enable texturing*
2. assign texture coordinates to vertices
   – *Proper mapping function is left to application*
3. specify texture parameters
   – *wrapping, filtering*

Texture Mapping



Texture Example

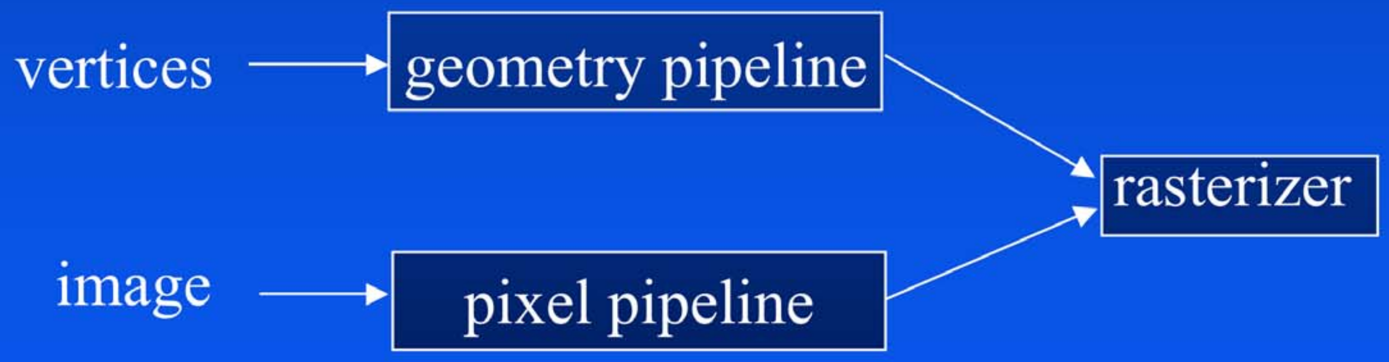
The texture (below) is a 256 x 256 image that has been mapped to a rectangular polygon which is viewed in perspective



Texture Mapping and the OpenGL Pipeline

Images and geometry flow through separate pipelines that join at the rasterizer

- "complex" textures do not affect geometric complexity



Specifying a Texture Image

Define a texture image from an array of texels (texture elements) in CPU memory

```
Glubyte my_texels[512][512];
```

Define as any other pixel map

- Scanned image
- Generate by application code

Enable texture mapping

- `glEnable(GL_TEXTURE_2D)`
- OpenGL supports 1-4 dimensional texture maps

Define Image as a Texture

```
glTexImage2D( target, level, components,
   w, h, border, format, type, texels );
```

`target:` **type of texture, e.g.** `GL_TEXTURE_2D`

`level:` **used for mipmapping (discussed later)**

`components:` **elements per texel**

`w, h:` **width and height of** `texels` **in pixels**

`border:` **used for smoothing (discussed later)**

`format and type:` **describe texels**

`texels:` **pointer to texel array**

```
glTexImage2D(GL_TEXTURE_2D, 0, 3, 512, 512, 0, GL_RGB,
GL_UNSIGNED_BYTE, my_texels);
```

Converting A Texture Image

- ***OpenGL requires texture dimensions to be powers of 2***
- ***If dimensions of image are not powers of 2***

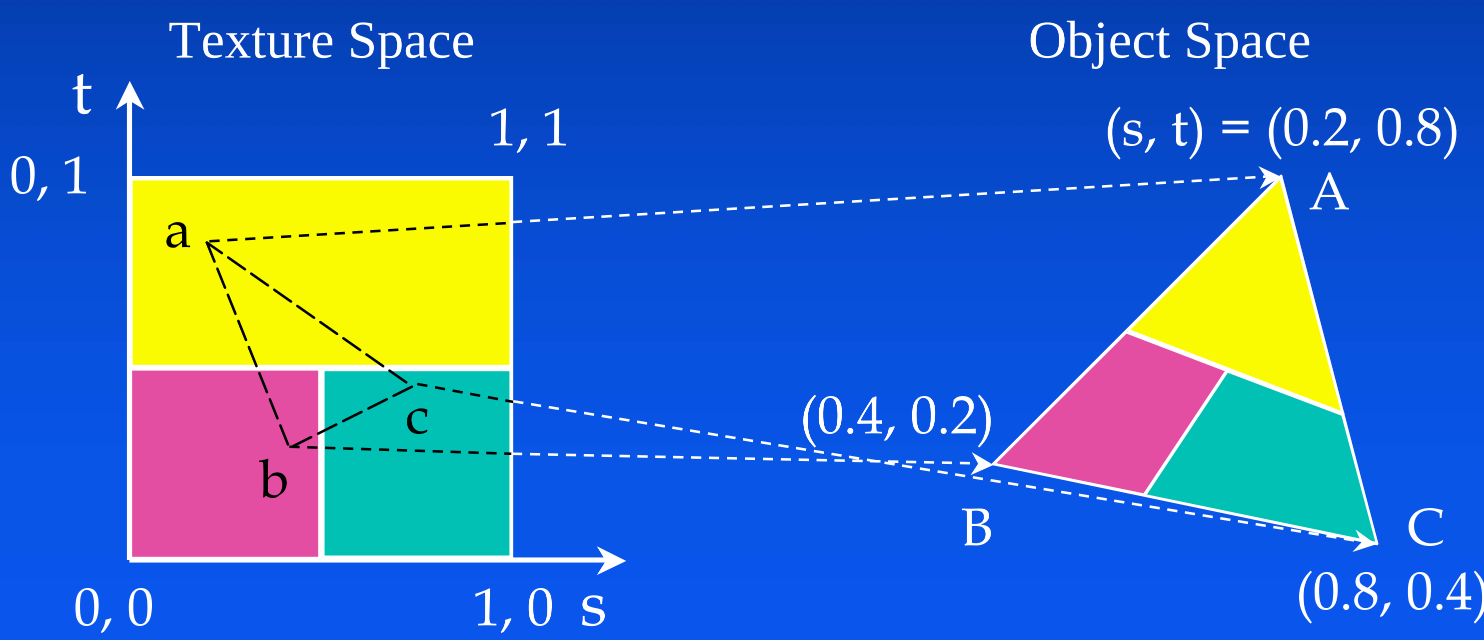
```
gluScaleImage( format, w_in, h_in,
   type_in, *data_in, w_out, h_out,
   type_out, *data_out );
```

  - **`data_in`** is source image
  - **`data_out`** is for destination image
- ***Image interpolated and filtered during scaling***

Mapping a Texture

Based on parametric texture coordinates

`glTexCoord*()` ***specified at each vertex***



Typical Code

```
glBegin(GL_POLYGON);
  glColor3f(r0, g0, b0); //if no shading used
  glNormal3f(u0, v0, w0); // if shading used
  glTexCoord2f(s0, t0);
  glVertex3f(x0, y0, z0);
  glColor3f(r1, g1, b1);
  glNormal3f(u1, v1, w1);
  glTexCoord2f(s1, t1);
  glVertex3f(x1, y1, z1);
        .
        .
glEnd();
```

Note that we can use vertex arrays to increase efficiency

Interpolation

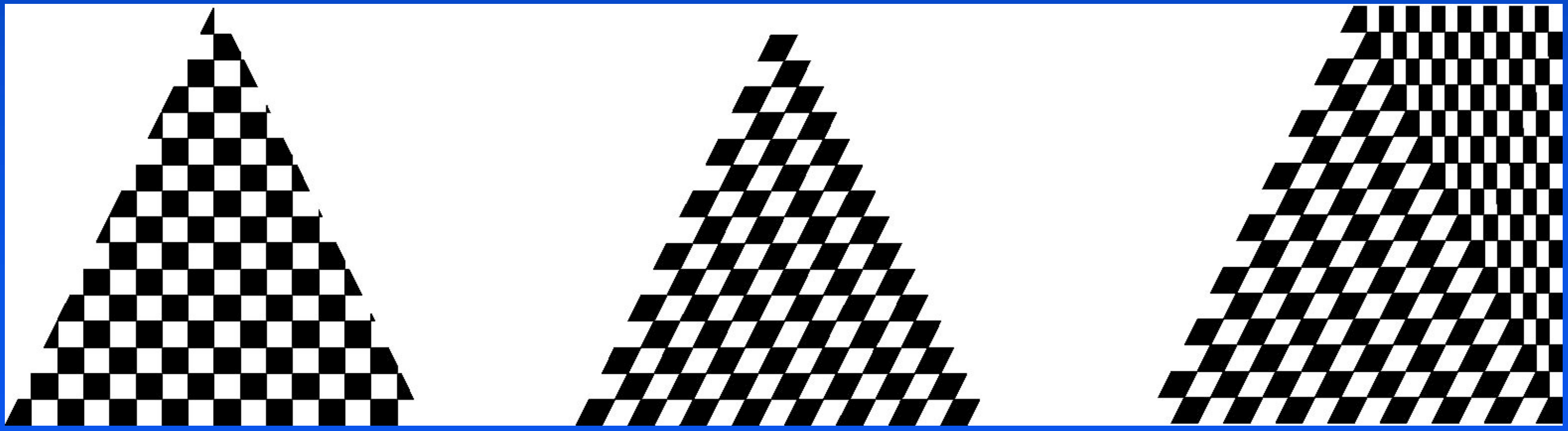
OpenGL uses interpolation to find proper texels from specified texture coordinates

Can be distortions

good selection of tex coordinates

poor selection of tex coordinates

texture stretched over trapezoid showing effects of bilinear interpolation



Texture Parameters

OpenGL has a variety of parameters that determine how texture is applied

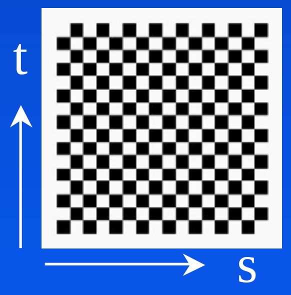
- Wrapping parameters determine what happens if s and t are outside the (0,1) range
- Filter modes allow us to use area averaging instead of point samples
- Mipmapping allows us to use textures at multiple resolutions
- Environment parameters determine how texture mapping interacts with shading

Wrapping Mode

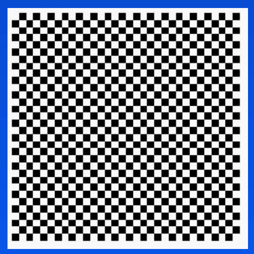
Clamping: if $s,t > 1$ use 1, if $s,t < 0$ use 0

Wrapping: use s,t modulo 1

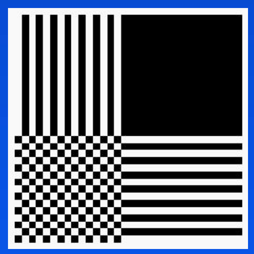
```
glTexParameteri( GL_TEXTURE_2D,
     GL_TEXTURE_WRAP_S, GL_CLAMP )

glTexParameteri( GL_TEXTURE_2D,
     GL_TEXTURE_WRAP_T, GL_REPEAT )
```



texture



GL_REPEAT
wrapping

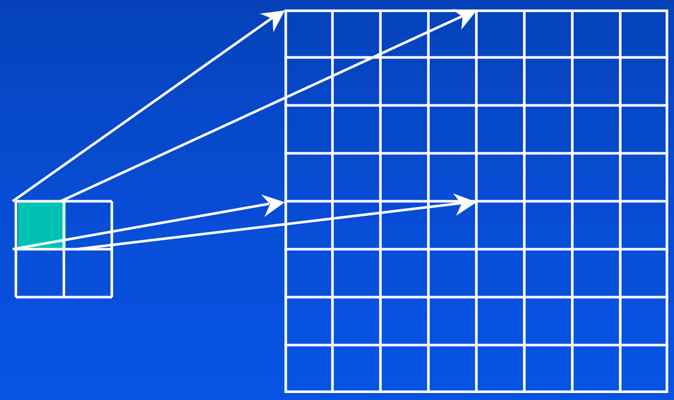


GL_CLAMP
wrapping

Magnification and Minification

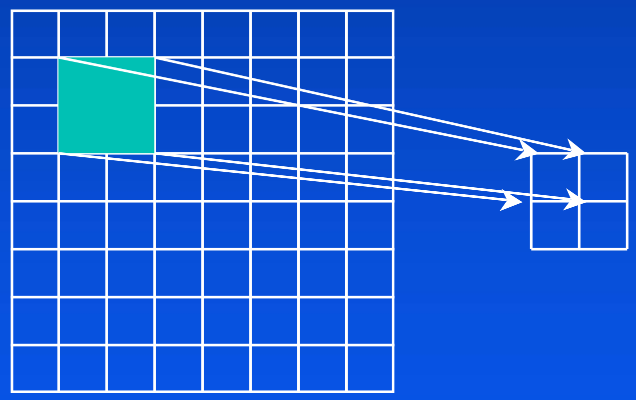
More than one texel can cover a pixel (*minification*) or more than one pixel can cover a texel (*magnification*)

Can use point sampling (nearest texel) or linear filtering (2 x 2 filter) to obtain texture values



Texture Polygon

Magnification



Texture Polygon

Minification

Filter Modes

Modes determined by

- **`glTexParameteri( target, type, mode )`**

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXURE_MAG_FILTER,
                GL_NEAREST);

glTexParameteri(GL_TEXTURE_2D, GL_TEXURE_MIN_FILTER,
                GL_LINEAR);
```

Note that linear filtering requires a border of an extra texel for filtering at edges (border = 1)

Mipmapped Textures

Mipmapping *allows for prefiltered texture maps of decreasing resolutions*

Lessens interpolation errors for smaller textured objects

Declare mipmap level during texture definition

```
glTexImage2D( GL_TEXTURE_*D, level, … )
```

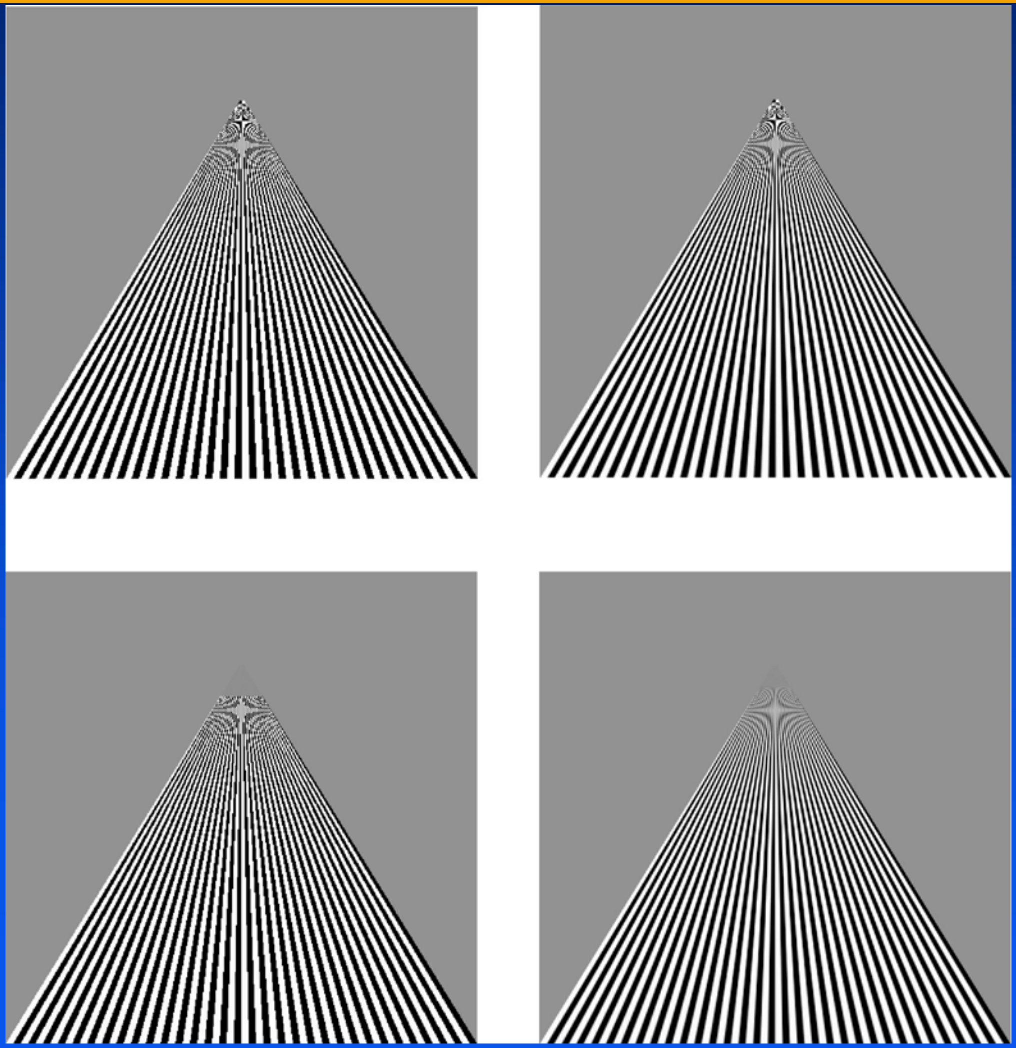
GLU mipmap builder routines will build all the textures from a given image

```
gluBuild*DMipmaps( … )
```

Example

point sampling

linear filtering



mipmapped point sampling

mipmapped linear filtering

Texture Functions

Controls how texture is applied

`glTexEnv{fi}[v]( GL_TEXTURE_ENV, prop, param )`

`GL_TEXTURE_ENV_MODE` ***modes***

- **`GL_MODULATE:`** modulates with computed shade
- **`GL_BLEND:`** blends with an environmental color
- **`GL_REPLACE:`** use only texture color
- **`GL(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);`**

Set blend color with *`GL_TEXTURE_ENV_COLOR`*

Perspective Correction Hint

Texture coordinate and color interpolation

- either linearly in screen space
- or using depth/perspective values (slower)

Noticeable for polygons "on edge"

```
glHint( GL_PERSPECTIVE_CORRECTION_HINT, hint )
```

where **`hint`** is one of

- ***`GL_DONT_CARE`***
- ***`GL_NICEST`***
- ***`GL_FASTEST`***

Generating Texture Coordinates

OpenGL can generate texture coordinates automatically

`glTexGen{ifd}[v]()`

specify a plane

- generate texture coordinates based upon distance from the plane

generation modes

- **`GL_OBJECT_LINEAR`**
- **`GL_EYE_LINEAR`**
- **`GL_SPHERE_MAP`** (used for environmental maps)

Texture Objects

Texture is part of the OpenGL state

- If we have different textures for different objects, OpenGL will be moving large amounts data from processor memory to texture memory

***Recent versions of OpenGL have* texture objects**

- one image per texture object
- Texture memory can hold multiple texture objects

Applying Textures II

specify textures in texture objects

set texture filter

set texture function

set texture wrap mode

set optional perspective correction hint

bind texture object

enable texturing

supply texture coordinates for vertex

- coordinates can also be generated