Computer Graphics Programming I

- Agenda:
  - Assignment #2 due
  - Finish lighting
    - Color materials
  - Texturing, part 1
    - Loading textures
    - Specifying texture coordinates
  - Start assignment #3

Materials

- Since `glMaterial` cannot be called inside begin / end, that interface is limited to one material per object.
  - Could split object into multiple begin / end pairs, but it is *much* more efficient to do a single block of drawing.
- Obvious interface deficiency. What to do?

Materials

- Since `glMaterial` cannot be called inside begin / end, that interface is limited to one material per object.
  - Could split object into multiple begin / end pairs, but it is *much* more efficient to do a single block of drawing.
- Obvious interface deficiency. What to do?
  - Enter “color material”.
  - Allows use of `glColor` calls to set certain material properties.

Color Material

- Enable `GL_COLOR_MATERIAL`.
- Set the per-face mode with `glColorMaterial`
  - Can set different mode for front and back faces.
  - Mode can be any of `GL_EMISSION`, `GL_AMBIENT`, `GL_DIFFUSE`, `GL_SPECULAR`, or `GL_AMBIENT_AND_DIFFUSE`
    - Default mode is `GL_AMBIENT_AND_DIFFUSE`.
- Modify the selected property with `glColor` calls inside or outside begin / end.

Drawing Spot Lights

- Even more important that point lights!
  - Not only does it have a position, but it also has a direction.
- How would you represent it?

Drawing Spot Lights

- Even more important that point lights!
  - Not only does it have a position, but it also has a direction.
- How would you represent it?
  - Draw a point for the light, as before.
  - Draw a line from the point in the direction the light faces.
  - Alternately, can draw a wire-frame cone for the spot cone, but this is *usually* overkill.

What is texture mapping?

- Application of an image onto a surface.
  - Many different kinds of images can be used as textures.
  - Texture mapping has been *the* fundamental drawing operation for at least the last 10 years.
- Images can come from a variety of sources.
  - Hand-drawn
  - Photos
  - Procedurally generated
  - Etc.

Kinds of Images

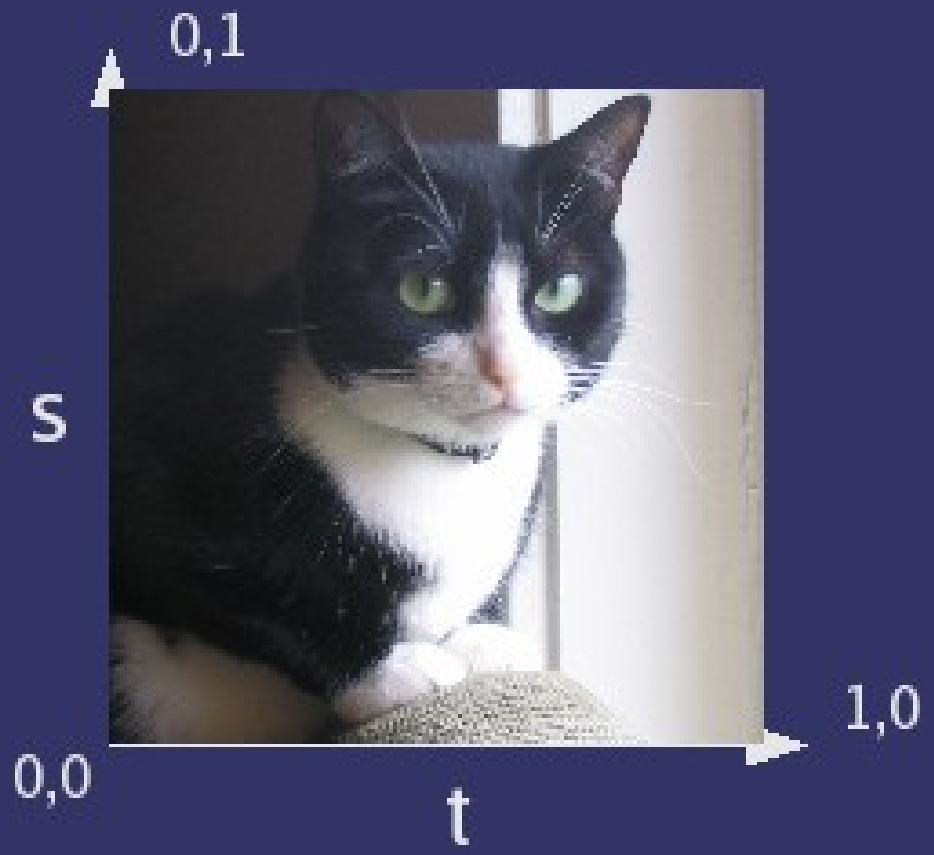
- 2D textures are by far the most common
- 1D textures have existed since OpenGL 1.0, but are not commonly used.
- 3D (aka volumetric) textures have been available since OpenGL 1.2.
  - Early hardware (e.g., Radeon 8500, Geforce) had limited support.
- Cubemap (aka cubic) textures have been available since OpenGL 1.3
  - *Very* useful!

Texture Coordinates

- Each vertex has associated texture coordinates
  - Like colors or normals
  - Coordinates have between 1 and 4 “dimensions”
  - Coordinates can be specified or generated by OpenGL
- Coordinates are interpolated along polygon edges, then across each scan line
  - Each fragment's coordinate is used to lookup a texel.
  - This interpolation is what we would want for normals for Phong shading...

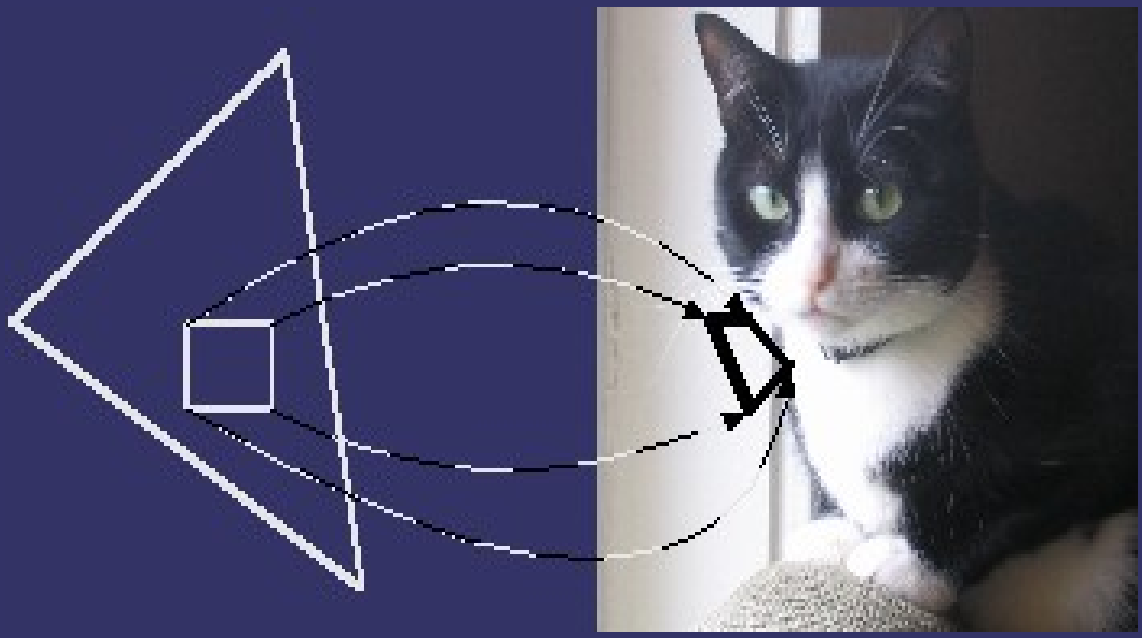
Texture Coordinates (cont.)

- Coordinates range from 0 to 1 in each dimension.
  - Dimensions are named s, t, r, and q.
- The origin in OpenGL is always the lower left corner!



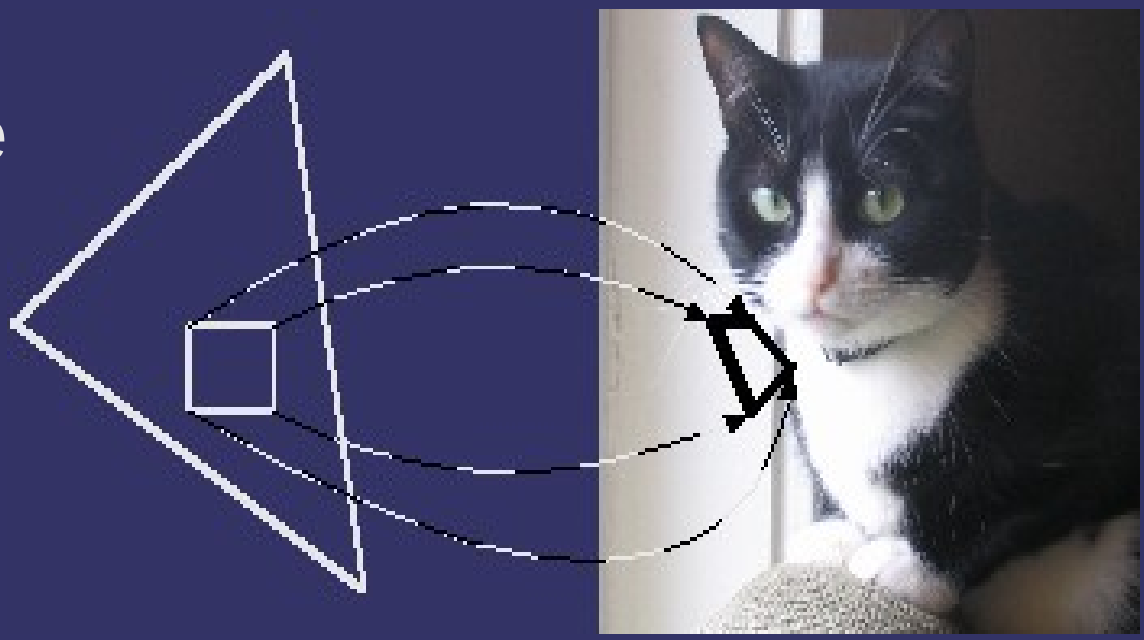
Texel Fetch

- Each fragment's texture coordinate selects a texel...but there's a problem here!



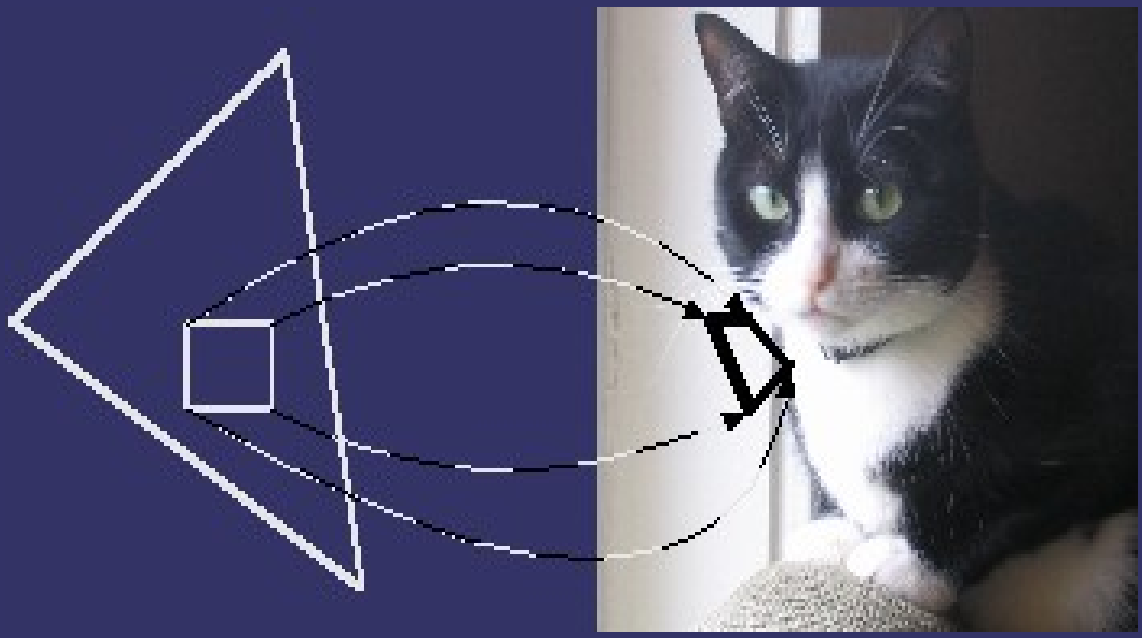
Texel Fetch

- Each fragment's texture coordinate selects a texel...but there's a problem here!
  - As the polygon gets smaller, each fragment represents more area in the texture.
  - How can we select just one texel for the fragment when the fragment covers multiple texels?



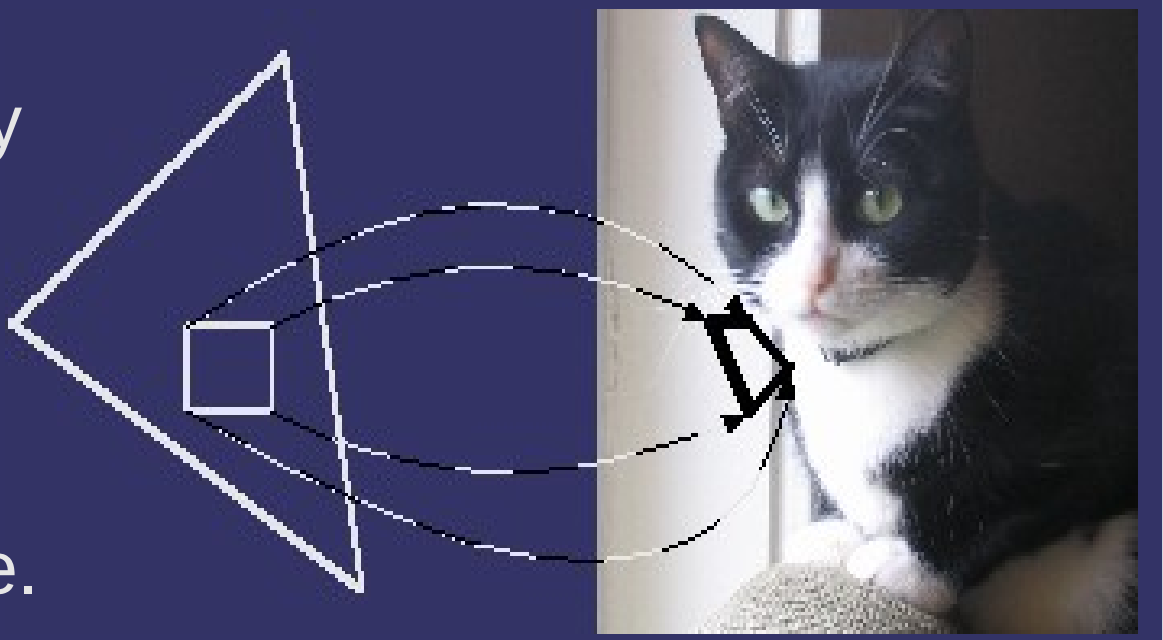
Texel Fetch

- Could read all texels covered by the fragment and average them together.
- What's the problem with this?



Texel Fetch

- Could read all texels covered by the fragment and average them together.
- What's the problem with this?
  - If the polygon is small enough, the whole texture is covered by one texel.
  - Reading the whole texture for one fragment would destroy performance.



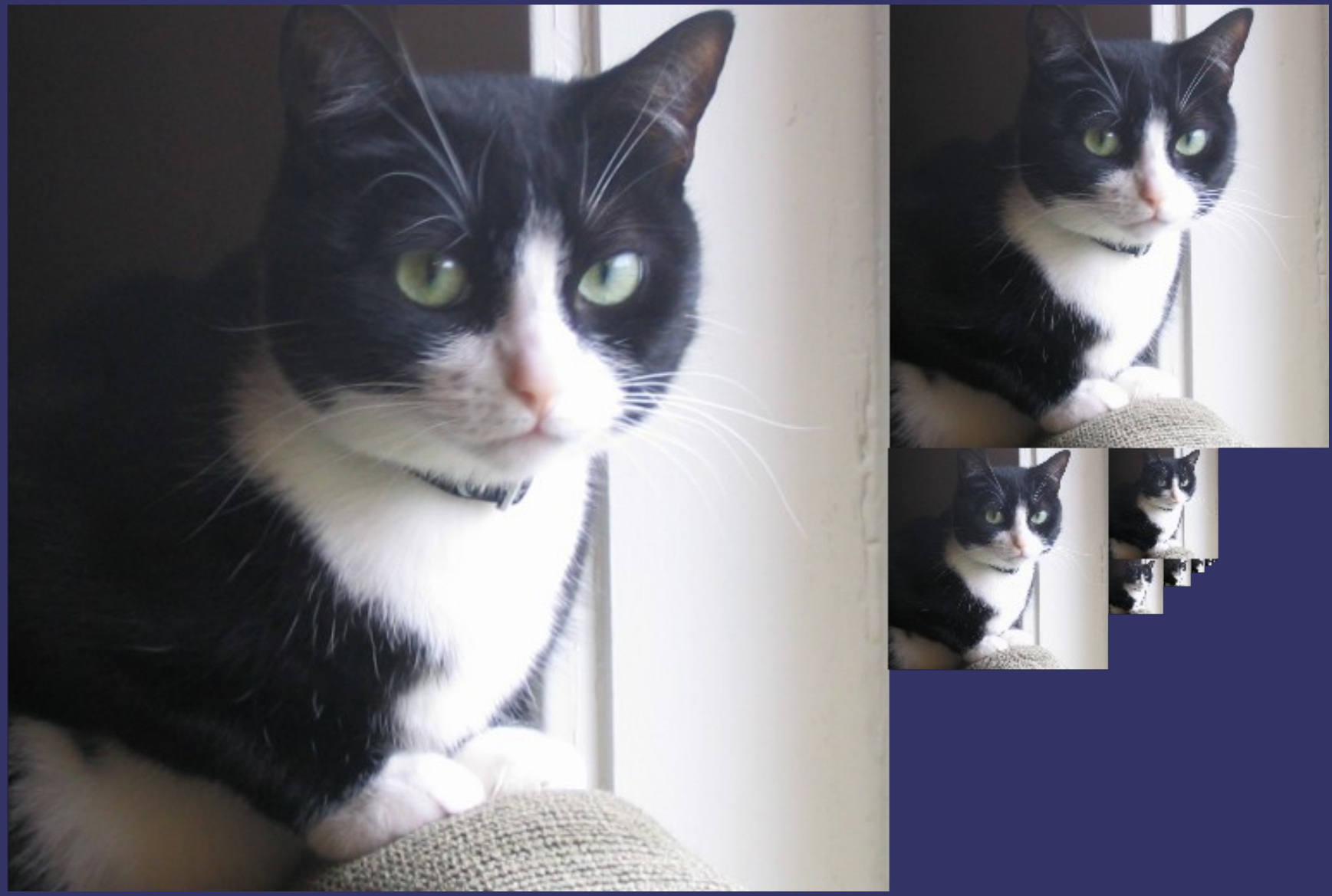
Linear and Bilinear Filtering

- We can approximate some of this *much* cheaper.
  - The texture coordinate selects somewhere between the texels.
  - Linear filtering selects the two nearest texels and calculates the weighted average.
  - Bilinear filtering selects the four nearest texels and calculates the weighted average.
  - Still not very good.

Multum in pavro

- Latin for “many things in one place.”
- Create multiple pre-filtered (averaged), down-sampled version of the “base” texture.
  - Down-sampled textures are called *mipmaps*.
  - The collection of mipmaps for a particular base texture is called its *mipmap stack*.
- As the texel area represented by a single fragment increases, use a smaller mipmap.
  - In smaller mipmaps, each texel represents more area from the base map.

Mipmap Example



Using Mipmaps

- Combine mipmapping ideas and linear / bilinear filtering ideas...
  - Filter the 4 nearest texels from the nearest mipmap
  - Filter 1 texel from each of the 2 nearest mipmaps
  - Filter the 4 nearest texels from each of the 2 nearest mipmaps.
    - This is called *trilinear filtering*.

Filtering Modes

- OpenGL has a name for each of these:
  - GL_NEAREST
  - GL_LINEAR
  - GL_NEAREST_MIPMAP_NEAREST
  - GL_NEAREST_MIPMAP_LINEAR
  - GL_LINEAR_MIPMAP_NEAREST
  - GL_LINEAR_MIPMAP_LINEAR
- We'll discuss how to select the filtering mode in a bit...

References

http://en.wikipedia.org/wiki/Texture_filtering

http://en.wikipedia.org/wiki/Mipmap

http://www.opengl.org/resources/code/samples/redbook/mipmap.c

http://www.graphicshardware.org/previous/www_1998/presentations/kirk/sld022.htm

http://www.sgi.com/products/software/performer/brew/anisotropic.html

Creating Textures

- Textures are identified by a unique texture object ID.
  - IDs can be generated by `glGenTextures`.
    - `glGenTextures` does not allocate memory for the texture.
  - IDs can also be “pulled from thin air”.
    - This is the push model in action!
    - An old trick is to use textures for letters in a font. Name the textures after the letters... `(GLint)'a'`, `(GLint)'b'`, etc.
    - In simple programs with few textures hard-code the names to 1, 2, 3, etc.

Creating Textures (cont.)

- Make a texture active with `glBindTexture`.
  - By default texture ID 0 is bound.
  - `glBindTexture` does not allocate memory for the texture.
- A texture ID is bound to a *texture target*.
  - The target determines what kind of texture (e.g., 1D, 2D, etc.) it is.
  - Each ID can only be associated with one target.

Creating Textures (cont.)

- Texture data is uploaded `glTexImage[123]D`
  - Since the size of the image is set by these functions, this is when the memory gets allocated.
  - Specified target must match dimensionality of the function. (e.g., `GL_TEXTURE_2D` cannot be passed to `glTexImage[123]D`).
  - Specified target must match target of bound texture ID.
  - Width and height must be powers of 2.
    - Restriction relaxed in OpenGL 2.1 or with `GL_ARB_texture_non_power_of_two`.

Texture Creation Example

```
glBindTexture(GL_TEXTURE_2D, id);

glTexImage2D(GL_TEXTURE_2D, level, GL_RGB,
             width, height,
             GL_RGB, GL_UNSIGNED_BYTE,
             pointer_to_image_data);
```

Updating Texture Data

- `glTexImage[123]D` are expensive because they allocate memory.
- To update a texture, use `glTexSubImage[123]D` instead.

```
glBindTexture(GL_TEXTURE_2D, id);
glTexSubImage2D(GL_TEXTURE_2D, level,
                x_offset, y_offset,
                width, height,
                GL_RGB, GL_UNSIGNED_BYTE,
                pointer_to_image_data);
```

Texture Completeness

- A texture must be “complete” or it will be disabled.
  - If a mipmap filter mode is selected, the texture must be *mipmap complete*, meaning that ever mipmap down to 1x1 must be set.
  - Cubic textures must be *cubemap complete*, meaning that all six sides must be set and, if necessary, be mipmap complete.
    - Cubic textures also must be square, and all sides must have the same dimensions.
- Not being complete is the cause of 99% of all newbie texturing problems.

Texture Parameters

- Set texture object parameters with `glTexParameter[if]` or `glTexParameter[if]v`
  - Just like lights!
  - Set filter mode, coordinate wrap mode, border color, and other parameters this way.

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

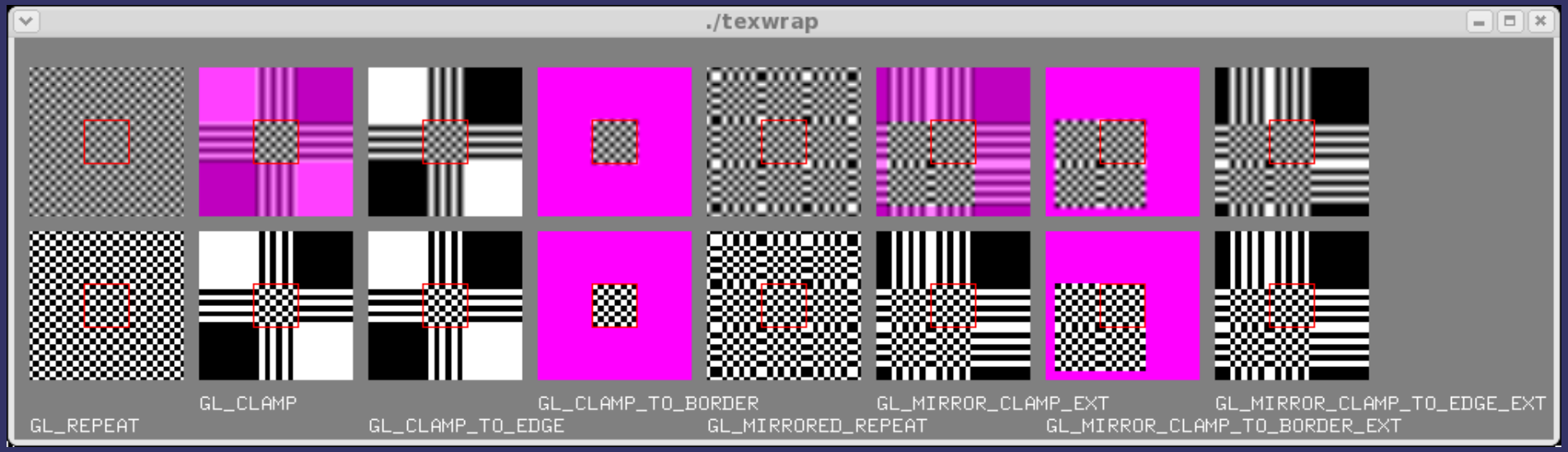
Texture Wrapping

- Texture *images* only have range [0, 1].
  - What happens if the requested texel coordinate is outside that range?

Texture Wrapping

- Texture *images* only have range [0, 1].
  - What happens if the requested texel coordinate is outside that range?
  - It depends on the wrap mode!
- Wrap mode is set independently for each dimension.
- 8 possible modes, not all implementations support all 8.
  - OpenGL 1.5 and 2.0 only require 5.
  - Remaining 3 were rejected for inclusion in 2.0.

Wrap Mode Demo



Next week...

- More texture mapping:
  - Texture combiners (part 1 of 3)
  - Texture coordinate generation
  - Environment mapping
- Assignment #3 due.
- Assignment #4 assigned.
- Maybe another quiz? >:)

Legal Statement

- This work represents the view of the authors and does not necessarily represent the view of IBM or the Art Institute of Portland.
- OpenGL is a trademark of Silicon Graphics, Inc. in the United States, other countries, or both.
- Khronos and OpenGL ES are trademarks of the Khronos Group.
- Other company, product, and service names may be trademarks or service marks of others.