Computer Graphics Programming I

- Agenda:
  - Quiz #2.
  - More texture mapping:
    - Texture combiners (part 1 of 3)
    - Texture coordinate generation
    - Environment mapping
  - Assignment #2 due.
  - Assignment #3 assigned.

Texture Combiners

- Before OpenGL 1.3, there was *very* little math that could be done with texture colors and interpolated colors.
  - `GL_REPLACE` – Use texture color, ignore interpolated color.
  - `GL_MODULATE` / `GL_ADD` – Multiply (or add) texture color and interpolated color.
  - `GL_DECAL` – Use texture color to blend constant and interpolated colors.
  - `GL_BLEND` – Use texture alpha to blend texture and interpolated colors.

GL_DECAL

- Math done by GL_DECAL (and GL_BLEND) depends on the texture's internal format.
  - Internal format set by glTexImage{123}D.

Format	Red	Green	Blue	Alpha
ALPHA	R_f	G_f	B_f	$A_f A_t$
LUMINANCE	$R_f * (1 - L_t) + R_c L_t$	$Gf * (1 - L_t) + G_c L_t$	$B_f * (1 - L_t) + B_c L_t$	A_f
LUMINANCE_ALPHA	$R_f * (1 - L_t) + R_c L_t$	$Gf * (1 - L_t) + G_c L_t$	$B_f * (1 - L_t) + B_c L_t$	$A_f A_t$
INTENSITY	$R_f * (1 - I_t) + R_c I_t$	$Gf * (1 - I_t) + G_c I_t$	$B_f * (1 - I_t) + B_c I_t$	$A_f * (1 - I_t) + A_c I_t$
RGB	$R_f * (1 - R_t) + R_c R_t$	$Gf * (1 - G_t) + G_c G_t$	$B_f * (1 - B_t) + B_c B_t$	A_f
RGBA	$R_f * (1 - R_t) + R_c R_t$	$Gf * (1 - G_t) + G_c G_t$	$B_f * (1 - B_t) + B_c B_t$	$A_f A_t$

GL_BLEND

- Math done by GL_BLEND depends on the texture's internal format.

Format	Red	Green	Blue	Alpha
ALPHA	*Invalid / Undefined*	-	-	-
LUMINANCE	*Invalid / Undefined*	-	-	-
LUMINANCE_ALPHA	*Invalid / Undefined*	-	-	-
INTENSITY	*Invalid / Undefined*	-	-	-
RGB	R_t	G_t	B_t	A_f
RGBA	$R_f * (1 - A_t) + A_t R_t$	$G_f * (1 - A_t) + A_t G_t$	$B_f * (1 - A_t) + A_t B_t$	$A_t A_f$

Setting Texture Environments

- All parameters set via `glTexEnv{if}[v]`

  `glTexEnvi(GLenum target, GLenum pname, GLint param)`

  - `target` must be `GL_TEXTURE_ENV`.
  - `pname` can be either `GL_TEXTURE_ENV_MODE` or `GL_TEXTURE_ENV_COLOR`.
    - Use the float-vector version for color.
    - Use the integer version for texture environment mode.

Texture Coordinate Generation

- Most commonly refered to as “texgen.”
- Causes GL to generate per-vertex texture coordinates.
  - Somewhat like the way it generates per-vertex colors during lighting.
- Enabled per texture coordinate.

  `glEnable(GL_TEXUTRE_GEN_S)`
- Generation controlled by several parameters.
  - Set using `glTexGen{ifd}[v]`.

Texgen Modes

- Three possible texgen modes pre-1.3:
  - `GL_OBJECT_LINEAR` – Calculates distance to a plane in object-space.
  - `GL_EYE_LINEAR` – Calculates distance to a plane in eye-space.
  - `GL_SPHERE_MAP` – Calculates a reflection vector (s & t only) into a 2D “sphere” texture.

```
glTexGeni(GL_S, GL_TEXTURE_GEN_MODE,
  GL_EYE_LINEAR);
```

Distance to a plane?

- `GL_OBJECT_LINEAR` and `GL_EYE_LINEAR` each use a plane in their calculations:
  - `GL_EYE_PLANE` – reference plane for all eye-space texgen calculations.
  - `GL_OBJECT_PLANE` – reference plane for all object-space texgen calculations.

```
glTexGenfv(GL_S, GL_EYE_PLANE, plane);
```

More Texgen Modes

- Two texgen modes added in OpenGL 1.3:
  - `GL_NORMAL_MAP` – Generates a vector (s, t, and r) equivalent to the vertex's eye-space normal.
  - `GL_REFLECTION_MAP` – Generates a vector (s, t, and r) equivalent to the vertex's ideal eye-space refelction vector.
- These modes are mostly useful with cubic textures.

Texture Matrix

- The texture matrix transforms texture coordinates just like you would expect.
  - Select with `glMatrixMode(GL_TEXTURE)`
- Can be used to move textures without changing the texture coordinates.
- Can be used to “swizzle” texture coordinates.
  - Use texture matrix to convert (s, t) to (s, 0, 0, t).

Cubic Texture Mapping

- Uses the s, t, and r texture coordinates.
- Texels aren't selected in the usual way:
  - Imagine a unit cube centered at the origin.
  - Texture coordinate is a vector from the origin.
  - The location where the vector intersects the cube determines the texel that is selected.
    - This means that the wrap modes are irrelevant.

What is cubic texturing good for?

- Complex lighting calculations
  - Create a special texture that encodes our lights.
  - Use an interpolated normal to look-up the color in the texture.
    - `GL_NORMAL_MAP` texgen for the win.
- Normalizing interpolated values.
  - Create a special texture that encodes the normalized value of each texel location.
- Environment mapping.

Using Cubic Textures

- Most commands use `GL_TEXTURE_CUBE_MAP` as the `target`.
- `glTexImage2D` uses a different target enum for each face of the cube:
  - `GL_TEXTURE_CUBE_MAP_POSITIVE_X`
  - `GL_TEXTURE_CUBE_MAP_NEGATIVE_X`
  - `GL_TEXTURE_CUBE_MAP_POSITIVE_Y`
  - `GL_TEXTURE_CUBE_MAP_NEGATIVE_Y`
  - `GL_TEXTURE_CUBE_MAP_POSITIVE_Z`
  - `GL_TEXTURE_CUBE_MAP_NEGATIVE_Z`

Using Cubic Textures (cont.)

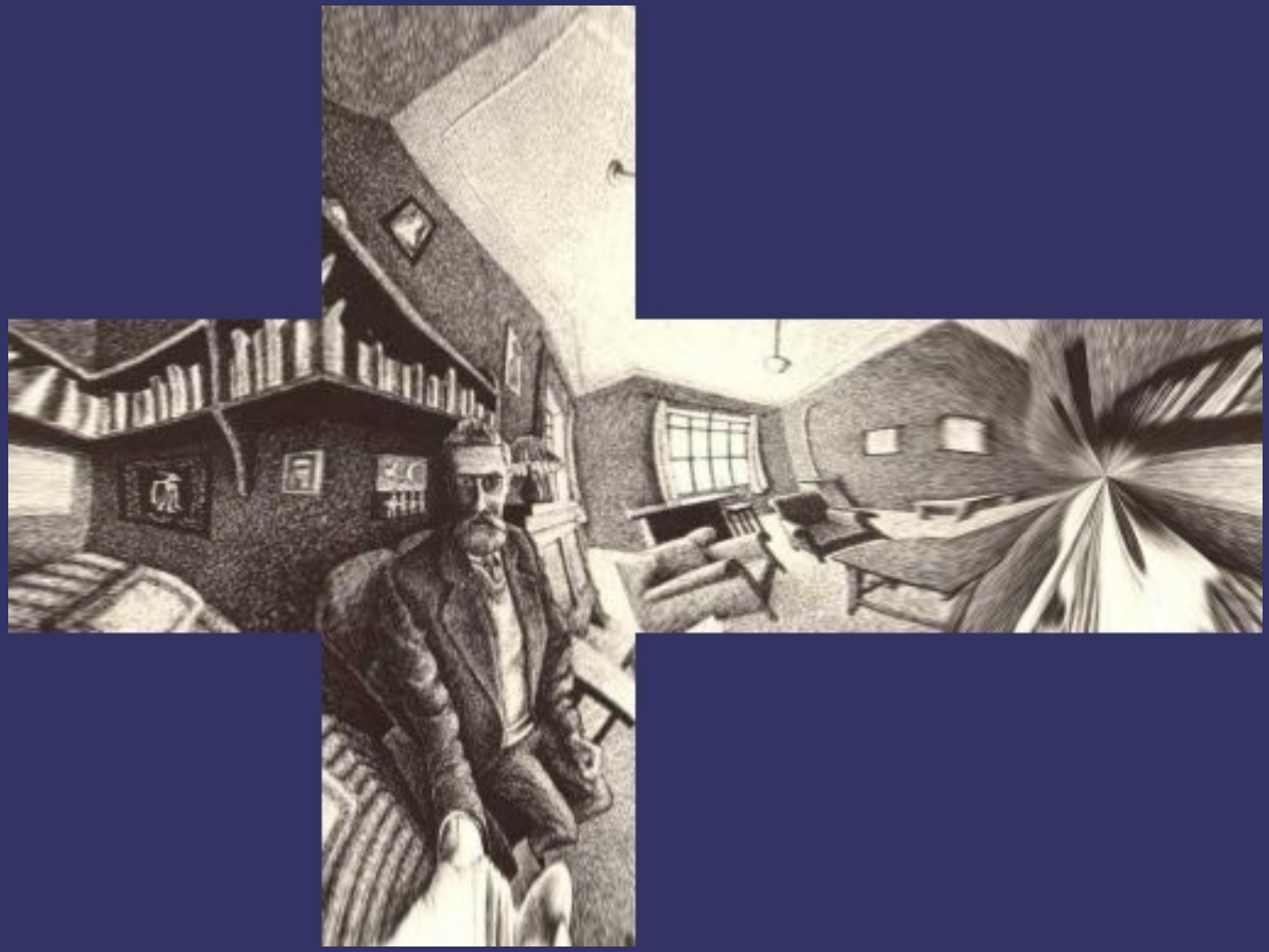
- All 6 faces must:
  - Be loaded.
  - Be square.
  - Be the same size.
  - Be mipmap complete (if mipmapping is used).
- If any of these conditions is not met, the texture will be disabled.

Environment Mapping

- Two common types of environment mapping:
  - Sphere environment mapping – Specially encode the reflection in a 2D texture. Imagine photographing a reflective sphere placed in a scene.
    - Difficult to generate source texture
    - Unequal distribution of texels
  - Cubic environment mapping – Each face of the cube represents one view of the scene.
    - Larger data
    - Easier to generate source textures

Sample Sphere Map

Sample Cube Map



Original image from http://brainwagon.org/?p=72

References

http://developer.nvidia.com/object/cube_map_ogl_tutorial.html

- A bit dated, but still a good foundation reference for cube mapping.

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

http://www.debevec.org/ReflectionMapping/

- *Very* good historical reference.

Next week...

- More texturing...
  - Additive specular reflections
  - Projective textures
  - Point sprites
  - Multi-texture
  - Texture combiners, part 2
- Assignment #3 due.

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.