Computer Graphics Programming II

- Agenda:
  - Remaining GLSL API
    - Attributes
    - Uniforms
      - Uniform matrices
      - Texture uniforms
  - Render-to-texture
    - Copy-to-texture
    - Framebuffer objects
  - Environment Mapping

Attributes

- Shaders can access built-in attributes
  - `gl_Color`, `gl_Normal`, `gl_Vertex`, etc.
- Can also create custom named attributes
  - Define in shader with `attribute` key-word
  - In application, custom attributes are numbered
    - Set with `glVertexAttrib[1234][sifd ui us]`
    - Normalized attributes are set with `glVertexAttrib4N[bsi ub us ui]v`
      - Normalized values, like colors, have range [0.0, 1.0]
    - Associate name from shader with number in application with `glBindAttribLocation`.

Uniforms

- Shaders can access built-in uniforms
  - `gl_ModelViewMatrix`, `gl_LightSource`, etc.
- Can also create custom named uniforms
  - Define in shader with `uniform` key-word
  - In application, uniforms are numbered
    - Get number with `glGetUniformLocation`

```
GLint glGetUniformLocation(GLuint handle,
   const GLchar *name);
```

Setting Uniforms

- Must bind program (with `glUseProgram`) to set its uniforms
- Set with `glUniform[1234][if]{v}`

```
void glUniform4fv(GLint uniform, GLsizei count,
   const GLfloat *data);
```

  - Uniforms can be arrays, and `count` is the number of elements
- Use `glMatrixUniform[234]fv` for matrix uniforms

```
void glMatrixUniform4fv(GLint uniform,
   GLsizei count, GLboolean transpose,
   const GLfloat *data);
```

Texture Uniforms

- Textures accessed through sampler uniforms
- Different sampler type for each target:
  - `sampler1D`, `sampler2D`, `sampler3D`
  - `samplerCube`, `samplerRect`
  - `sampler1DShadow`, `sampler2DShadow`
    - We'll use these next term.

```
uniform sampler2D normal_map;
```

- Set uniform to texture unit number
  - Set just like any other integer uniform

Texture Fetch Functions

- Sample texture using `texture` function
  - One function for each texture target
  - Each function takes a sampler uniform and a texture coordinate as parameters

```
tex_color = texture2D(tex_sampler,
                      gl_TexCoord[0]);
```

  - Separate versions also available for projective texturing

```
tex_color = texture2DProj(tex_sampler,
                          gl_TexCoord[0]);
```

    - Do **not** divide by texture coordinate's w by hand!!!

Render-to-texture

- Several methods exist in OpenGL to render to a texture.
  - Render to the framebuffer, then copy the results to a texture.
  - Use the *new* framebuffer objects extension.
  - Render to a pixel buffer (pbuffer), then bind the pbuffer to a texture.
    - This method is platform dependent (i.e., is different on Linux, Windows, and Mac OS) and will *not* be covered in this course.

Why render to a texture?

- Many, many effects can be created by rendering to one or more textures, then using those textures to render the final scene.

Copy to texture

- Easiest and least efficient form of render-to-texture.
- Draw to the backbuffer, copy resulting image to texture with either `glCopyTexImage2D` or `glCopyTexSubImage2D`.
- *That's it.*

Problems with copy-to-texture

- Must perform extra copies.
- Must perform extra buffer clears.
- If the window is obscured or off the screen, the texture may be corrupted.
- The window must be at least as large as the desired texture.

Framebuffer Objects

- The framebuffer object (FBO) interface has a fairly steep learning curve.
  - We're just going to scratch the surface today, and we'll continue next week.
  - The ARB spent two years developing this interface.
  - It builds on the familiar texture interfaces, but is still *very* different.
- Now that I've stricken terror into your hearts...

Creating an FBO

- The first step is to create the FBO.
  - Use `glGenFramebuffersEXT` and `glBindFramebufferEXT`.
- Attach one or more renderable objects to it.
  - There are several functions available to do this. More on this later.
  - Conceptually, this is similar to attaching shader objects to a program object.
  - Example: Attach an RGBA texture to the FBO.

Using an FBO

- Once the FBO has all of its attachments:
  - Make sure the FBO is acceptable to the driver / hardware with `glCheckFramebufferStatusEXT`.
    - Some hardware can't handle some combinations of attachments.
    - Some combinations of attachments are just plain wrong (i.e., attaching a depth texture to a color attachment).
  - Bind the framebuffer with `glBindFramebufferEXT`.
  - Reset viewport and draw!

Using an FBO (cont.)

- When done rendering to FBO, bind the 0 object to resume rendering to window.
- To use textures that were rendered to, simply bind and use as usual.
  - You **cannot** use `GL_GENERATE_MIPMAPS` with FBO-rendered textures.
  - Instead, use new function `glGenerateMipmapEXT` to generate the mipmap stack on-demand.

Renderbuffers and textures

- Two broad types of objects can be attached to an FBO.
  - A texture. Most textures are both texturable and renderable.
  - A renderbuffer. Renderbuffers are *only* renderable.
    - If you won't need to texture from it, prefer to use a renderbuffer.

Texture attachments

- Created as always using `glTexImage2D` et. al.
  - Typically the `pixels` parameter will be `NULL`.
- Different attachment function depending texture dimensionality.
  - `glFramebufferTexture1DEXT` – Attach a 1D texture.
  - `glFramebufferTexture2DEXT` – Attach a 2D texture or a cube map face.
  - `glFramebufferTexture3DEXT` – Attach a slice of a 3D texture.

Renderbuffers

- Created using `glGenRenderbuffersEXT` and `glRenderbufferStorageEXT`.
  - Analogous to `glGenTextures` and `glTexImage2D`.
  - Only way to supply data to a renderbuffer is by rendering to it.
- Attach to FBO using `glFramebufferRenderbufferEXT`.

Dimensions and dimensionality

- The dimensions (i.e., height and width) of all attachments **must** match.
  - This requirement will be relaxed in a future extension.
- The dimensionality (i.e., 1D or 2D) of all attachments must match.
  - A 2D “slice” of a 3D texture is attached, so it is treated as a 2D texture for this purpose.

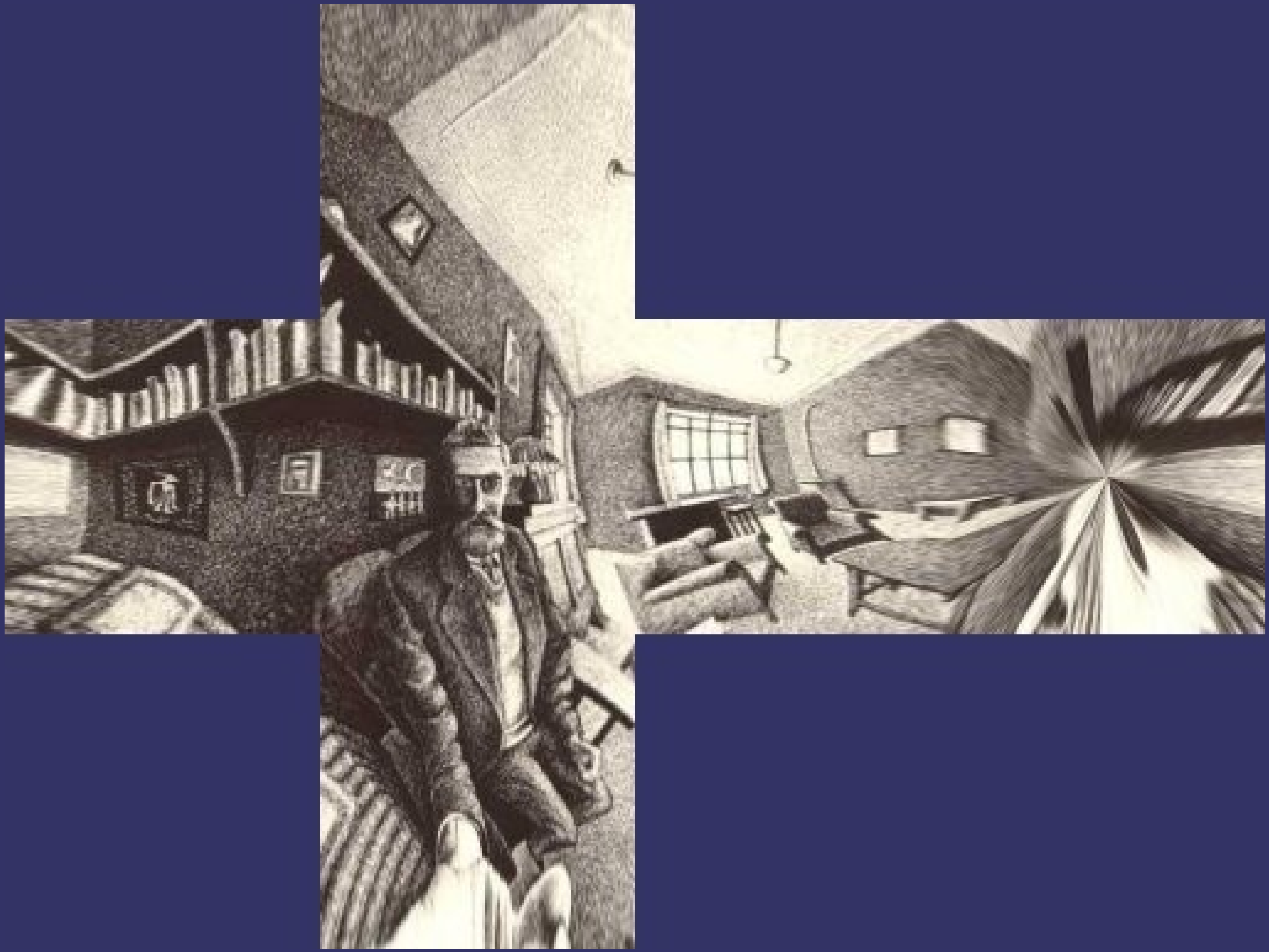
Break

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

Paraboloid

- View of environment as reflected by a convex parabolic mirror
  - The *outside* of a satellite dish, for example
  - Reflects 180° of the environment
  - Does *not* have the singularity of a sphere map

Paraboloid (cont.)

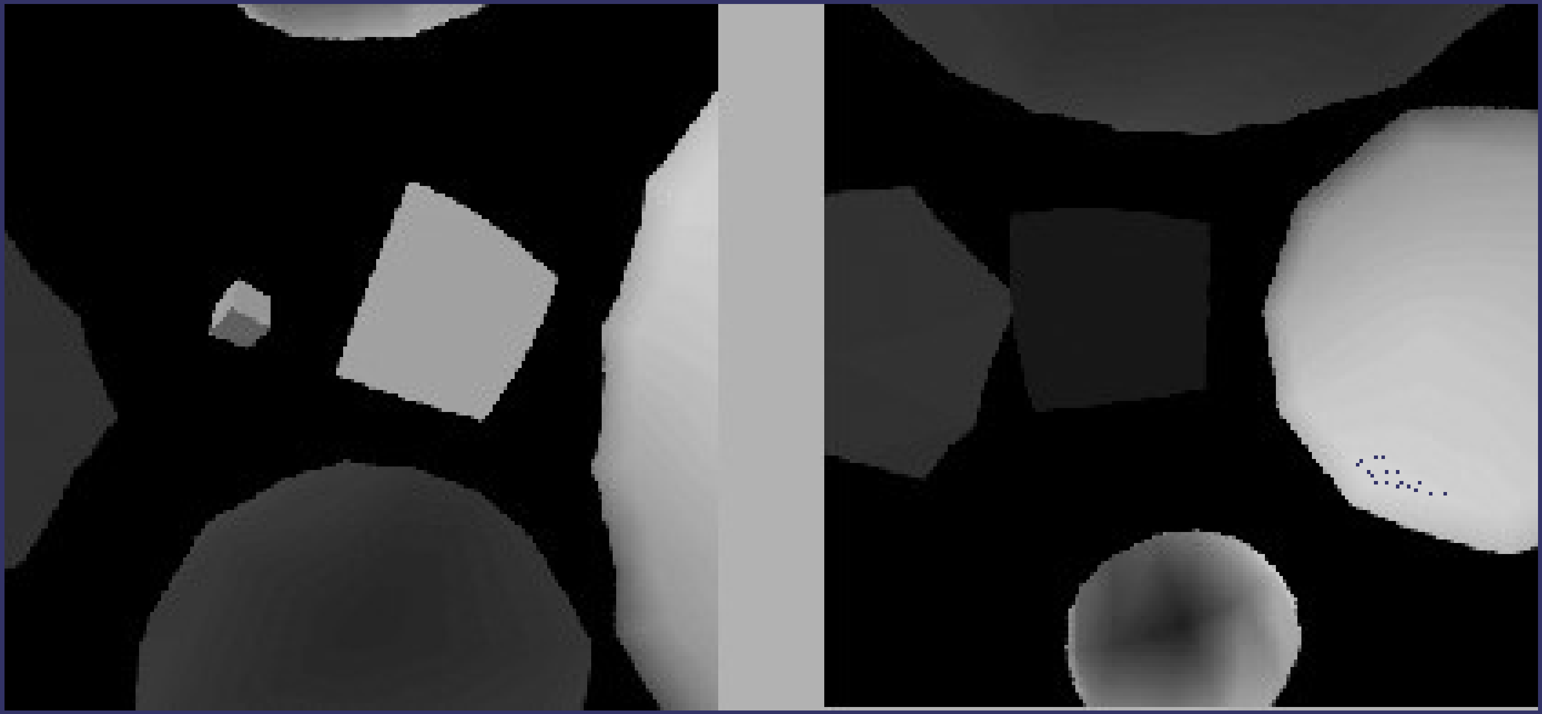
- Can easily convert refection vector to 2D texture coordinate for paraboloid map

$$\begin{pmatrix} s \\ t \\ 1 \\ 1 \end{pmatrix} = A \cdot P \cdot S \cdot M_n^T \cdot R^T$$

$$A = \begin{pmatrix} \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, P = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}, S = \begin{pmatrix} -1 & 0 & 0 & d_x \\ 0 & -1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

  - *d* is the view direction vector
    - (0 0 -1) or (0 0 1) depending on direction we're looking
  - M_n is the transformation matrix for normals

Sample Parabolic Map



Original image from
http://opengl.org/resources/code/samples/sig99/advanced99/notes/node185.html

Real-time Generation

- Cube maps are easy...

Real-time Generation

- Cube maps are easy...
  - Draw six images from center of environment
  - Each image uses one cube face as the near plane
- But you have to draw SIX TIMES

Real-time Generation

- Dual-parabolic maps are easy...

Real-time Generation

- Dual-parabolic maps are easy...
  - Draw two images from center of environment
  - Transform vertices as usual w/modelview-projection matrix
  - Divide X, Y, Z by W
    - Call magnitude of this vector *L*
  - Normalize and divide resulting X and Y by (Z + 1)
  - Final Z is *L* remapped to view volume
  - Final W is 1.0.

References

http://opengl.org/resources/code/samples/sig99/advanced99/notes/node184.html

Jason Zink. "Dual Paraboloid Mapping in the Vertex Shader." GameDev.net, 1996. http://www.gamedev.net/reference/articles/article2308.asp

Wolfgang Heidrich and Hans-Peter Seidel. "View-independent environment maps." In *Proceedings of the SIGGRAPH/Eurographics Workshop on Graphics Hardware*, 1998. http://www.cs.ubc.ca/~heidrich/Papers/GH.98.pdf

Next week...

- Improving the reflection model
  - Using environment maps as better lights
  - Fresnel reflection
  - BRDF introduction
- Assignment #1 due
- Quiz #1

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.