

VGP351 – Week 5

⇒ Agenda:

- Quiz #2
- Texture mapping, part 1



12-February-2009

© Copyright Ian D. Romanick 2009

What is texture mapping?



12-February-2009

© Copyright Ian D. Romanick 2009

What is texture mapping?

⇒ Classic definition:

Application of an image to a polygon or 3D model.



12-February-2009

© Copyright Ian D. Romanick 2009

Kinds of Images

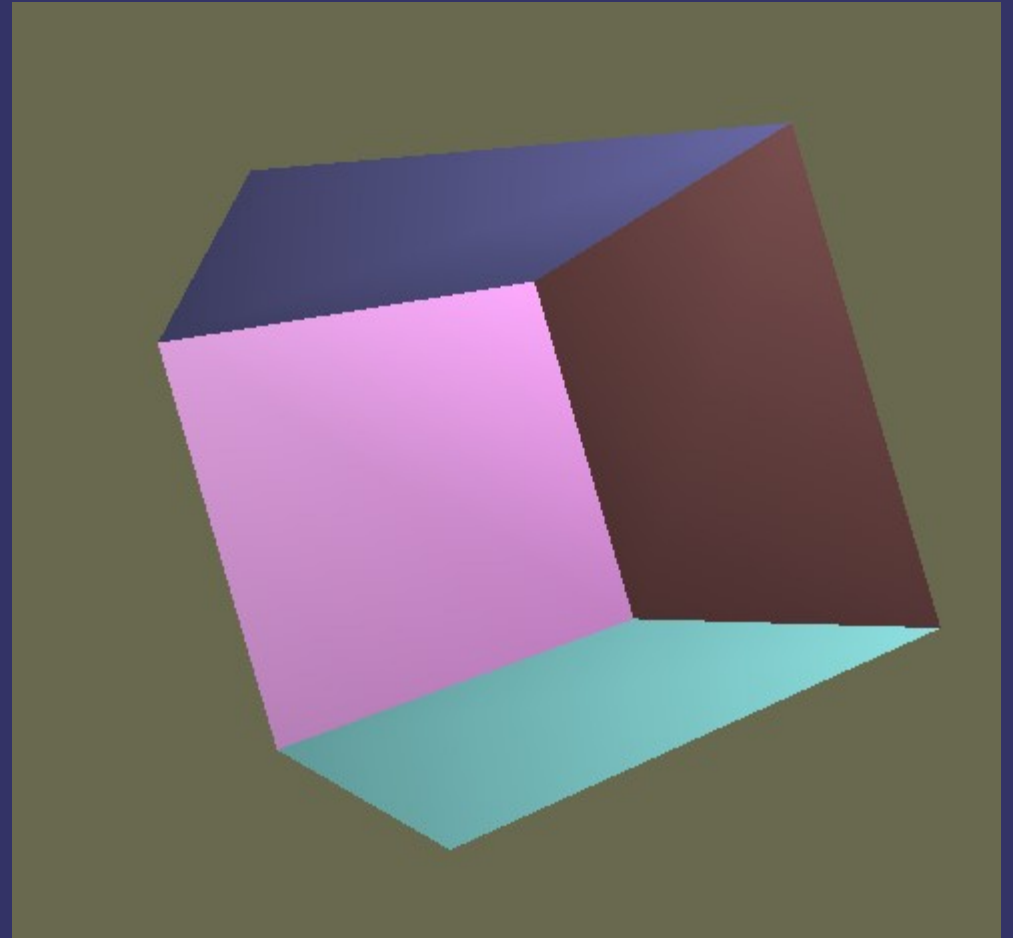
- Several *dimensionalities* are commonly used:
 - 1D – Usually used as large look-up tables or for color space conversions
 - 2D – Rectangular images...what we usually think of as a texture image
 - 3D (volumetric) – May be used to store voxel type data, volumetric light data, etc.
 - Cubemap (cubic) – 6 square, same-sized textures representing faces of a cube. Often used for environment maps



12-February-2009

© Copyright Ian D. Romanick 2009

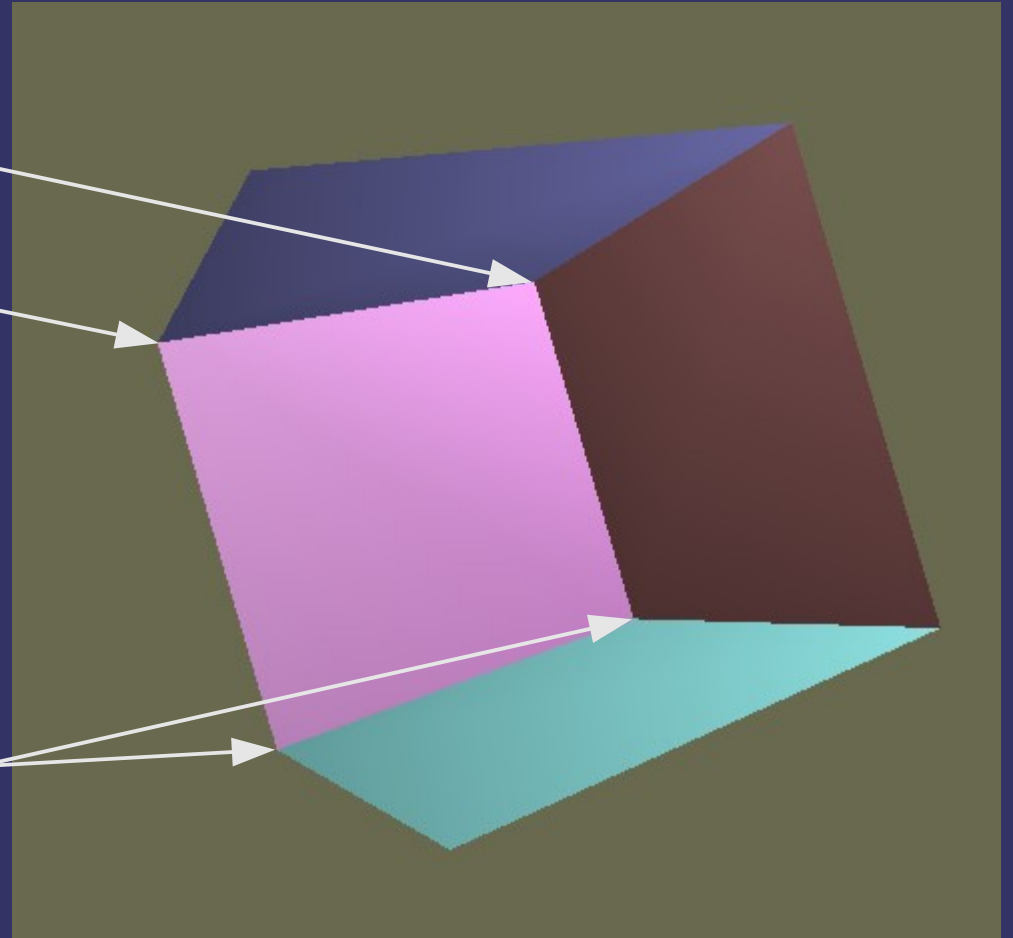
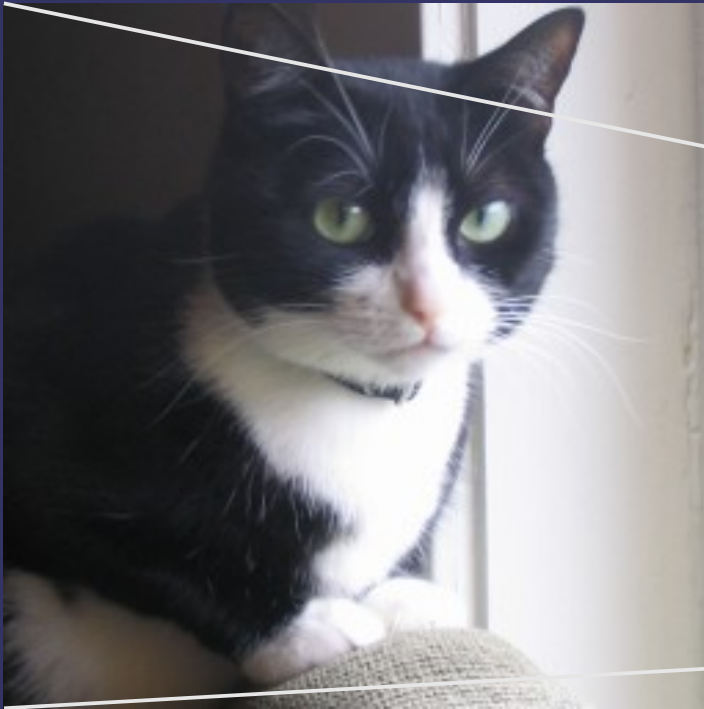
Texture Mapping



12-February-2009

© Copyright Ian D. Romanick 2009

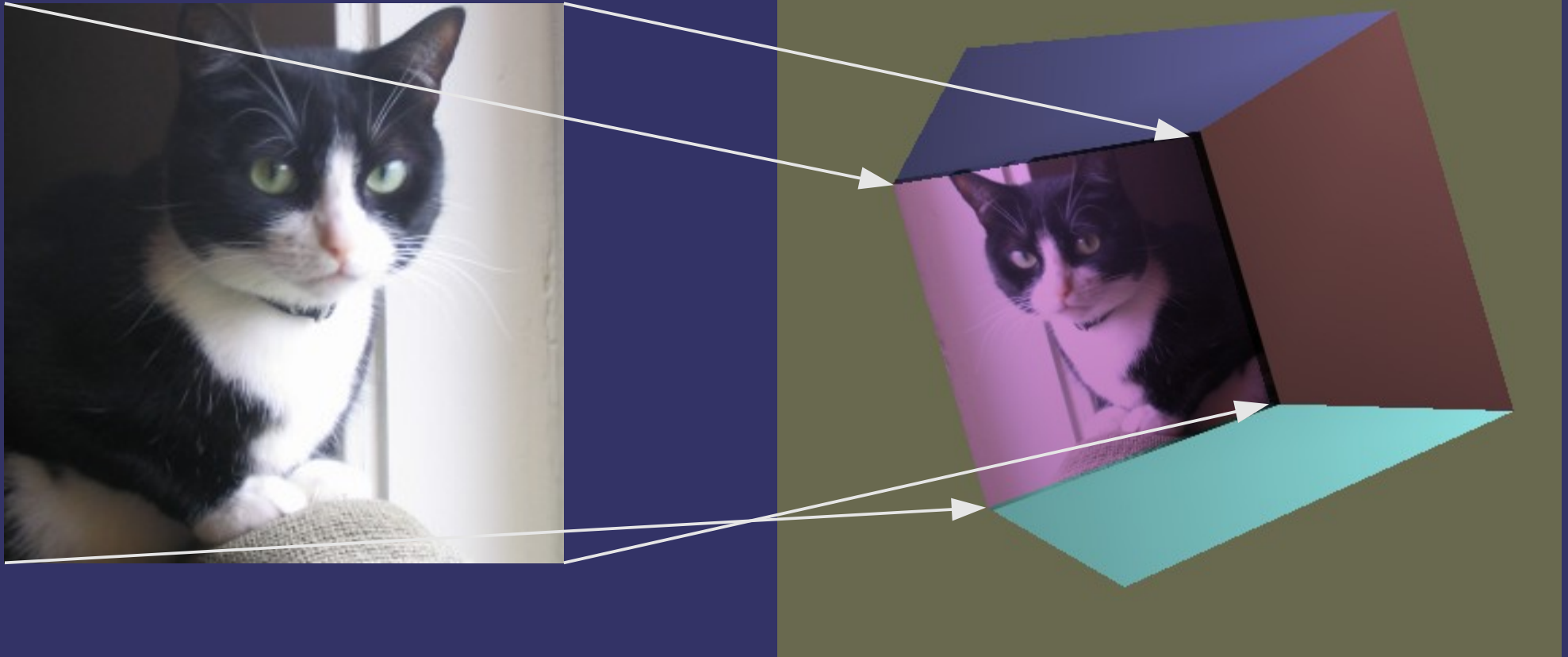
Texture Mapping



12-February-2009

© Copyright Ian D. Romanick 2009

Texture Mapping



12-February-2009

© Copyright Ian D. Romanick 2009

Texture Mapping

- Where does the *mapping* come from?
 - Numerous types of projections
 - Spherical
 - Cylindrical
 - Planar
 - Reflections
 - “Hand” edited coordinates



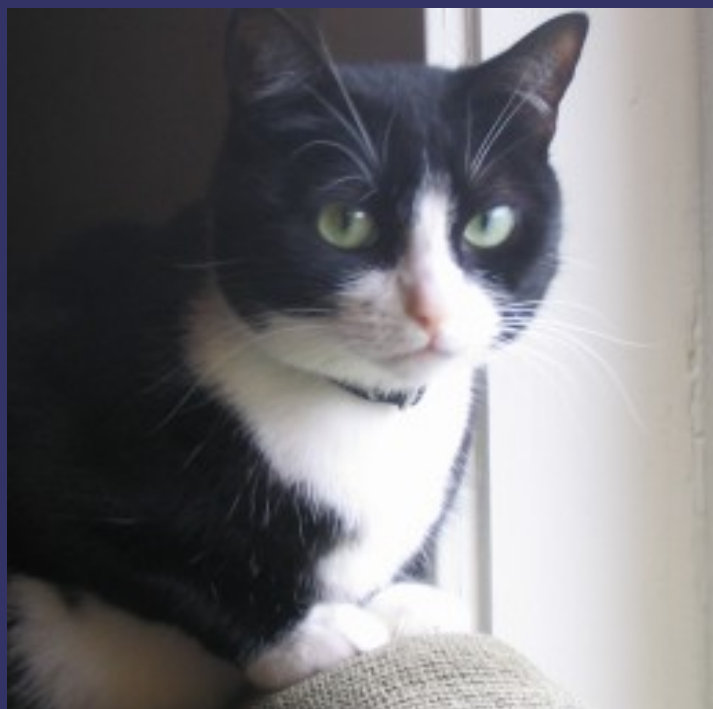
12-February-2009

© Copyright Ian D. Romanick 2009

Cylindrical Mapping

1, 0

1, 1



0, 0

1, 0

u

v



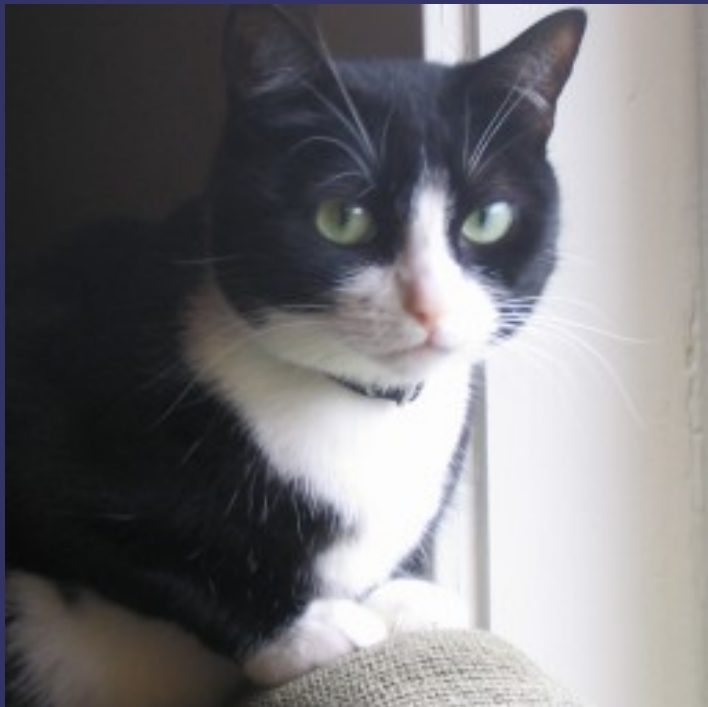
12-February-2009

© Copyright Ian D. Romanick 2009

Cylindrical Mapping

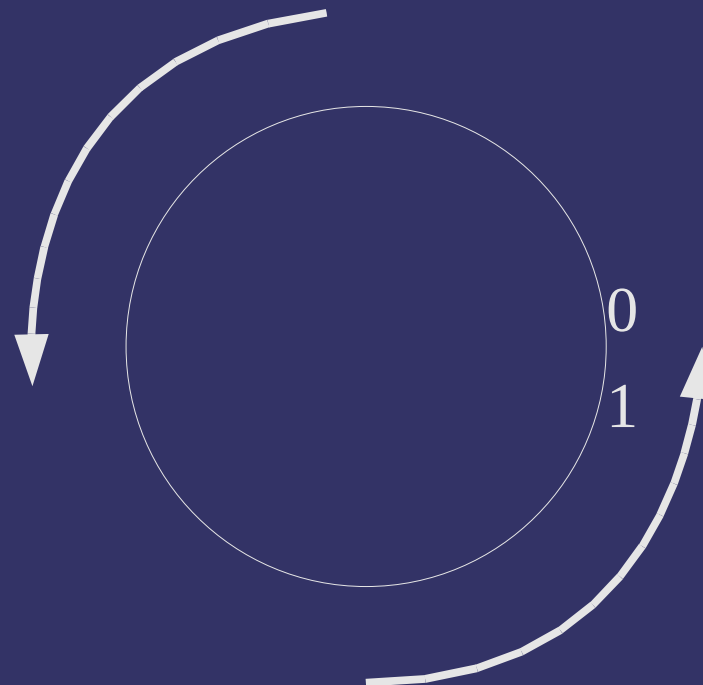
1, 0

1, 1



0, 0

1, 0



12-February-2009

© Copyright Ian D. Romanick 2009

Cylindrical Mapping

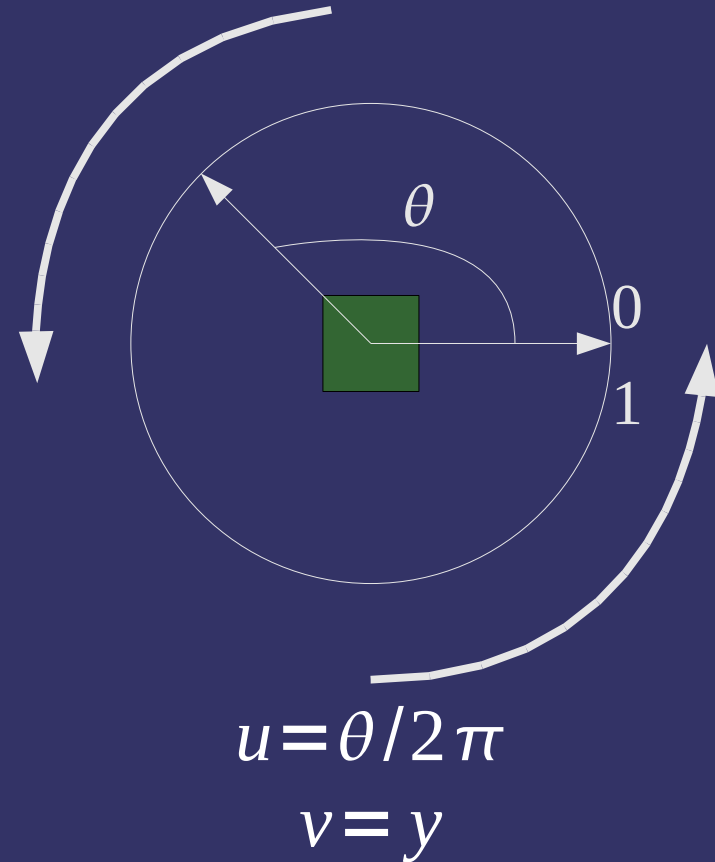
1, 0

1, 1



0, 0

1, 0



12-February-2009

© Copyright Ian D. Romanick 2009

Cylindrical Mapping

```
vec2 cylinder_map(vec3 position)
{
    vec2 tc;

    tc.s = atan(position.x, position.z) / 360.0;
    tc.t = position.y;
    return tc;
}
```



12-February-2009

© Copyright Ian D. Romanick 2009

Explicit Texture Coordinates

- Most commonly, texture coordinates are generated by the 3D modeling package
 - These coordinates are stored in the model file, and supplied, by you, to OpenGL



12-February-2009

© Copyright Ian D. Romanick 2009

Explicit Texture Coordinates

- Most commonly, texture coordinates are generated by the 3D modeling package
 - These coordinates are stored in the model file, and supplied, by you, to OpenGL
 - Coordinates are supplied using vertex shader attributes



12-February-2009

© Copyright Ian D. Romanick 2009

Point Sprites

- Special mode for rendering points that automatically generates useful texture coordinates
 - Upper left of point gets (0, 0, 0, 0) and lower right gets (1, 1, 0, 0)
 - Enable in GL with:

```
glEnable(GL_POINT_SPRITE);
```
 - Add a fragment shader variable called `gl_PointCoord`



12-February-2009

© Copyright Ian D. Romanick 2009

Coordinate Interpolation

⇒ Linear interpolation:

$$u_\alpha = (1 - \alpha)u_0 + \alpha u_1$$

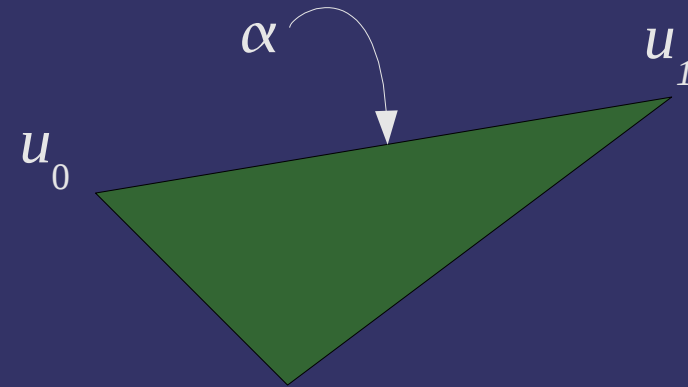


Image from <http://wwwx.cs.unc.edu/~sud/courses/236/a6/>

12-February-2009

© Copyright Ian D. Romanick 2009

Coordinate Interpolation

⇒ Linear interpolation:

$$u_{\alpha} = (1 - \alpha)u_0 + \alpha u_1$$

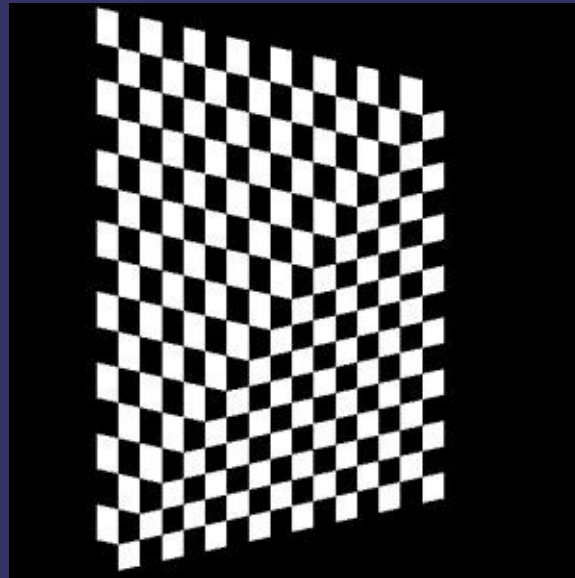
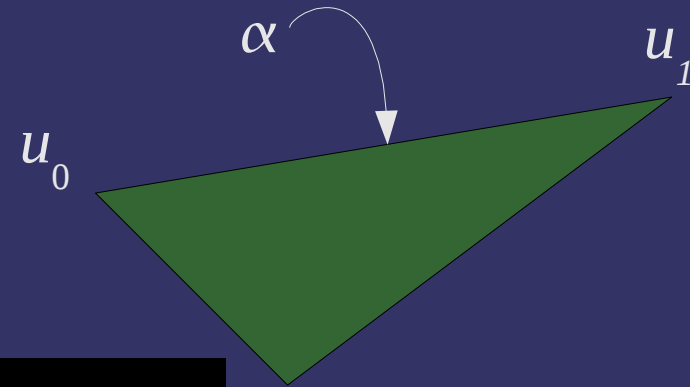


Image from <http://wwwx.cs.unc.edu/~sud/courses/236/a6/>

12-February-2009

© Copyright Ian D. Romanick 2009

Coordinate Interpolation

⇒ Perspective correct interpolation:

$$u_{\alpha} = \frac{(1-\alpha)\frac{u_0}{z_0} + \alpha\frac{u_1}{z_1}}{(1-\alpha)\frac{1}{z_0} + \alpha\frac{1}{z_1}}$$

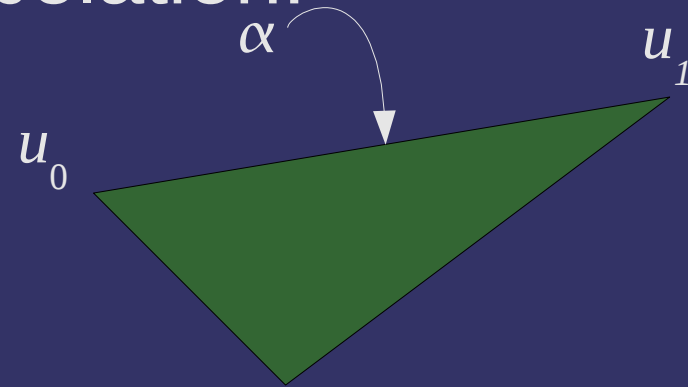


Image from <http://wwwx.cs.unc.edu/~sud/courses/236/a6/>

12-February-2009

© Copyright Ian D. Romanick 2009

Coordinate Interpolation

⇒ Perspective correct interpolation:

$$u_{\alpha} = \frac{(1-\alpha)\frac{u_0}{z_0} + \alpha\frac{u_1}{z_1}}{(1-\alpha)\frac{1}{z_0} + \alpha\frac{1}{z_1}}$$

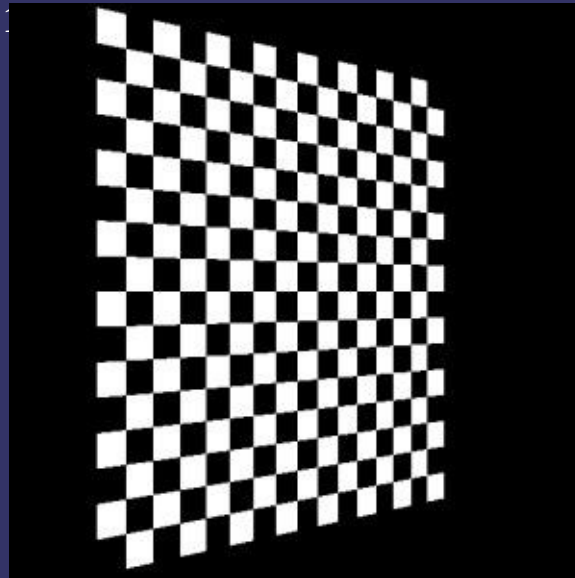
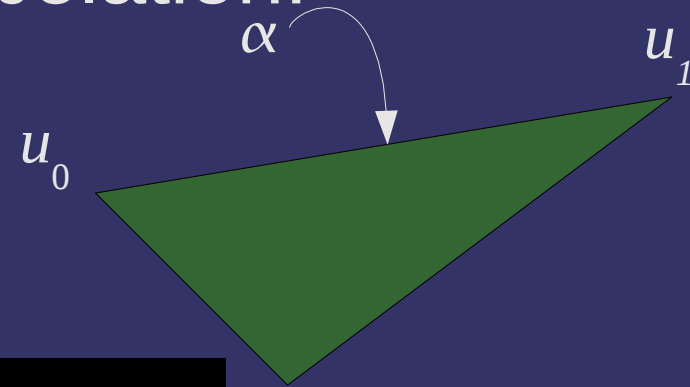


Image from <http://wwwx.cs.unc.edu/~sud/courses/236/a6/>

12-February-2009

© Copyright Ian D. Romanick 2009

Creating Textures

- ⇒ In OpenGL, textures are named objects

```
void glGenTextures(GLsizei n,  
                  GLuint *textures);
```

```
void glDeleteTextures(GLsizei n,  
                      const GLuint *textures);
```

- ⇒ “Bind” a texture for use:

```
void glBindTexture(GLenum target,  
                  GLuint texture);
```

- `target` selects which dimensionality we're talking about
- Binding creates the object, but it still has no storage



12-February-2009

© Copyright Ian D. Romanick 2009

Creating Textures

⇒ Texture targets:

- `GL_TEXTURE_1D` – 1D texture
- `GL_TEXTURE_2D` – 2D texture
- `GL_TEXTURE_3D` – 3D textures
- `GL_TEXTURE_RECTANGLE_ARB` – Special kind of 2D texture
- `GL_TEXTURE_CUBE_MAP` – Cubic texture
 - There are other cubic texture targets. We'll discuss those next week with environment mapping



12-February-2009

© Copyright Ian D. Romanick 2009

Creating Textures

- ⇒ Storage is created and *optionally* initialized with:

```
void glTexImage1D(GLenum target, GLint level,  
                 GLint internalFormat, GLsizei width,  
                 GLint border, GLenum format, GLenum type,  
                 const GLvoid *pixels);
```

- Variations for 2D and 3D textures also exist

- ⇒ Storage is updated with:

```
void glTexSubImage1D(GLenum target,  
                    GLint level, GLint xoffset, GLsizei width,  
                    GLenum format, GLenum type,  
                    const GLvoid *pixels);
```



12-February-2009

© Copyright Ian D. Romanick 2009

Creating Textures

⇒ format and type describe the source data

- format can be one of: GL_RED, GL_GREEN, GL_BLUE, GL_ALPHA, GL_RGB, GL_BGR, GL_RGBA, GL_BGRA, GL_LUMINANCE, and GL_LUMINANCE_ALPHA
- type can be one of: GL_UNSIGNED_BYTE, GL_BYTE, GL_UNSIGNED_SHORT, GL_SHORT, GL_UNSIGNED_INT, GL_INT, GL_FLOAT, GL_UNSIGNED_SHORT_5_6_5, GL_UNSIGNED_SHORT_5_6_5_REV, GL_UNSIGNED_SHORT_4_4_4_4, GL_UNSIGNED_SHORT_4_4_4_4_REV, GL_UNSIGNED_SHORT_5_5_5_1, GL_UNSIGNED_SHORT_1_5_5_5_REV, GL_UNSIGNED_INT_8_8_8_8, and GL_UNSIGNED_INT_8_8_8_8_REV

A few less common types have been omitted for brevity



12-February-2009

© Copyright Ian D. Romanick 2009

Creating Textures

➤ Internalformat describes how the texture should be stored

- Can be one of: GL_ALPHA, GL_ALPHA4, GL_ALPHA8, GL_ALPHA12, GL_ALPHA16, GL_LUMINANCE, GL_LUMINANCE4, GL_LUMINANCE8, GL_LUMINANCE12, GL_LUMINANCE16, GL_LUMINANCE_ALPHA, GL_LUMINANCE4_ALPHA4, GL_LUMINANCE6_ALPHA2, GL_LUMINANCE8_ALPHA8, GL_LUMINANCE12_ALPHA4, GL_LUMINANCE12_ALPHA12, GL_LUMINANCE16_ALPHA16, GL_INTENSITY, GL_INTENSITY4, GL_INTENSITY8, GL_INTENSITY12, GL_INTENSITY16, GL_RGB, GL_R3_G3_B2, GL_RGBA, GL_RGBA2, GL_RGBA4, GL_RGBA8, GL_RGBA12, or GL_RGBA16



Creating Textures

- Storage is created and initialized from framebuffer data with:

```
void glCopyTexImage1D(GLenum target,  
    GLint level, GLenum internalformat,  
    GLint x, GLint y, GLsizei width,  
    GLint border);
```

- Storage is updated from framebuffer data with:

```
void glCopyTexSubImage1D(GLenum target,  
    GLint level, GLint xoffset,  
    GLint x, GLint y, GLsizei width);
```



12-February-2009

© Copyright Ian D. Romanick 2009

Texture Units

- A texture unit is the piece of hardware that accesses a texture image
- Many OpenGL texture operations are per-object, but some are per-unit
 - Select the unit with:

```
void glActiveTexture(GLenum texture);
```

Enum is `GL_TEXTURE n` , where n is unit number



12-February-2009

© Copyright Ian D. Romanick 2009

Texture Units

- A texture unit is the piece of hardware that accesses a texture image
- Many OpenGL texture operations are per-object, but some are per-unit
 - Select the unit with:

```
void glActiveTexture(GLenum texture);
```
- Use this API to set per-unit texture objects as well!



12-February-2009

© Copyright Ian D. Romanick 2009

Texture Parameters

⇒ Set texture object parameters with:

```
void glTexParameterI(GLenum target,  
                    GLenum pname, GLint param);
```

```
void glTexParameteriv(GLenum target,  
                    GLenum pname, const GLint *params);
```



12-February-2009

© Copyright Ian D. Romanick 2009

Texture Wrapping

- ⇒ Texture images have coordinates on the range $[0, 1]$
 - What happens if the requested texel coordinate is outside that range?



12-February-2009

© Copyright Ian D. Romanick 2009

Texture Wrapping

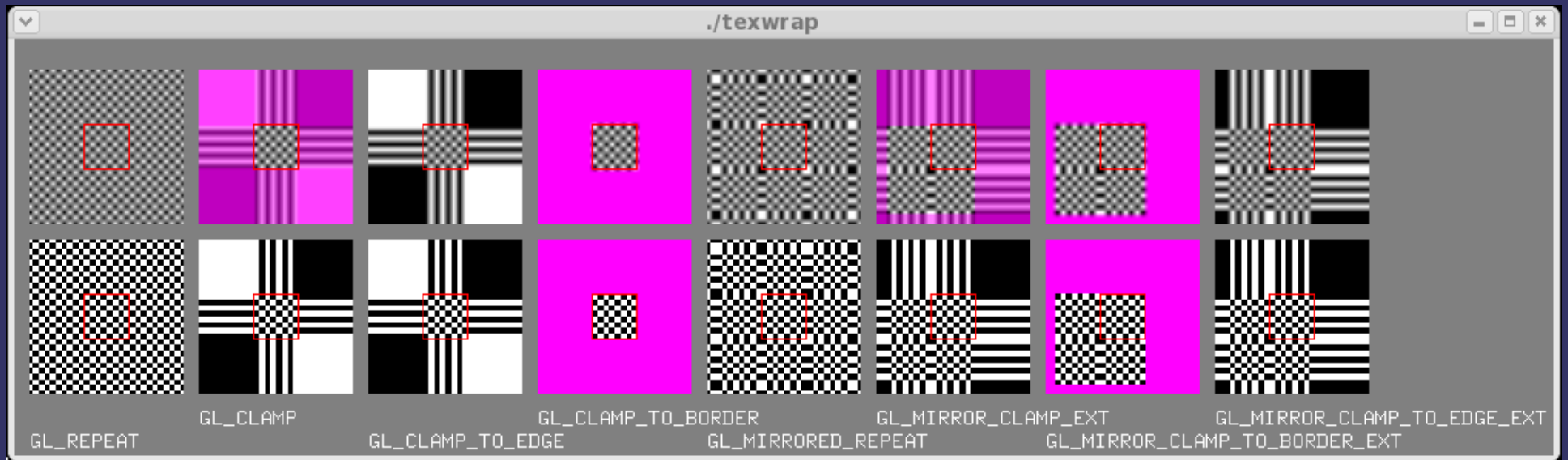
- Texture images have coordinates on the range $[0, 1]$
 - What happens if the requested texel coordinate is outside that range?
 - It depends on the wrap mode!
- Wrap mode is set independently for each dimension
- 8 possible modes
 - Not all implementations support all 8
 - OpenGL 1.5 and later only require 5



12-February-2009

© Copyright Ian D. Romanick 2009

Texture Wrapping



12-February-2009

© Copyright Ian D. Romanick 2009

Texture Wrapping

⇒ Select the wrap mode with `glTexParameter`:

```
glTexParameteri(GL_TEXTURE_2D,  
                GL_TEXTURE_WRAP_S,  
                GL_CLAMP_TO_BORDER);
```

```
glTexParameteri(GL_TEXTURE_2D,  
                GL_TEXTURE_WRAP_T,  
                GL_REPEAT);
```



12-February-2009

© Copyright Ian D. Romanick 2009

Texture Wrapping

- ⇒ `GL_CLAMP`, `GL_CLAMP_TO_BORDER`, and their mirrored counter parts use a texture “border” color

```
const GLfloat color[4] = {  
    0.0, 1.0, 0.0, 1.0  
};  
  
glTexParameterfv(GL_TEXTURE_2D,  
                GL_TEXTURE_BORDER_COLOR,  
                color);
```



12-February-2009

© Copyright Ian D. Romanick 2009

Samplers

- In GLSL, textures are accessed through special data types call *samplers*
 - There is a sample type for each texture target: `sampler1D`, `sampler2D`, `samplerRect`, `sampler3D`, and `samplerCube`
 - Samplers are uniforms
 - Set the sampler uniform to the number of the texture *unit*
`glUniform1i(tex_sampler, 1);`



12-February-2009

© Copyright Ian D. Romanick 2009

Texture Sample Functions

- Textures are accessed using special GLSL functions
 - There is a many variations of these functions
 - The function name must match the sampler type
 - See the GLSL quick reference

http://www.opengl.org/sdk/libs/OpenSceneGraph/glsl_quickref.pdf

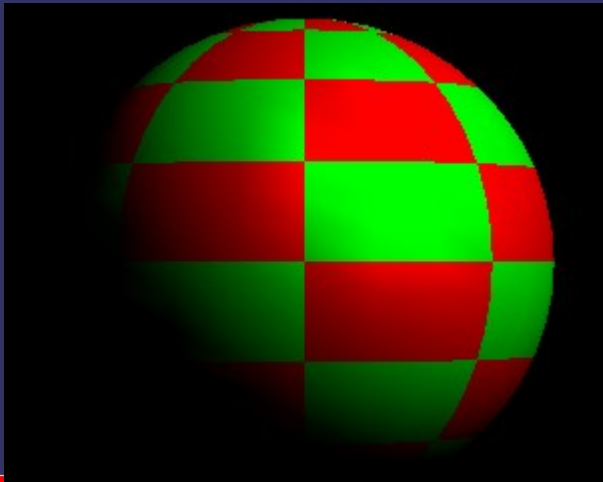


12-February-2009

© Copyright Ian D. Romanick 2009

Specular Lighting

- ⇒ We perform lighting in the vertex shader and texturing in the fragment shader
 - VS passes a single color to FS, and FS combines it with the texture color
 - Why is this wrong?



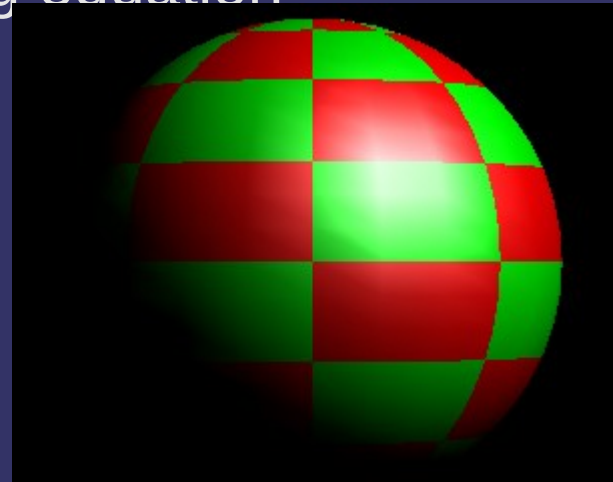
12-February-2009

© Copyright Ian D. Romanick 2009



Specular Lighting

- We perform lighting in the vertex shader and texturing in the fragment shader
 - VS passes a single color to FS, and FS combines it with the texture color
 - Why is this wrong?
 - Texture color is typically a diffuse property
 - It usually supplies C_d in the lighting equation

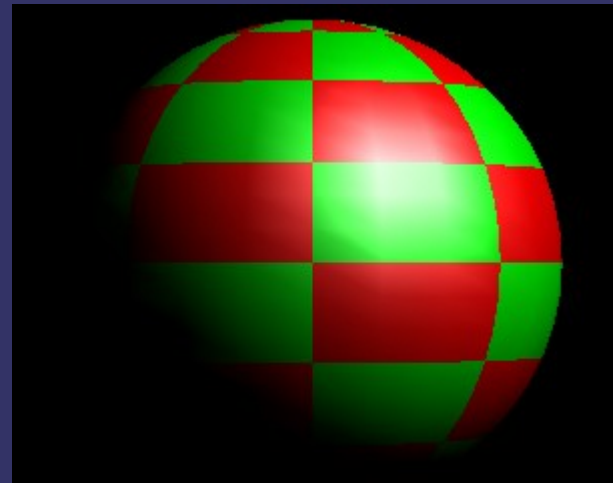
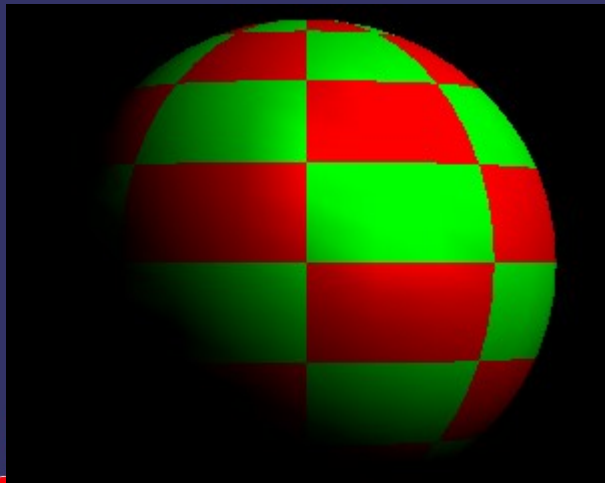


12-February-2009

© Copyright Ian D. Romanick 2009

Specular Lighting

⇒ How can we fix this?



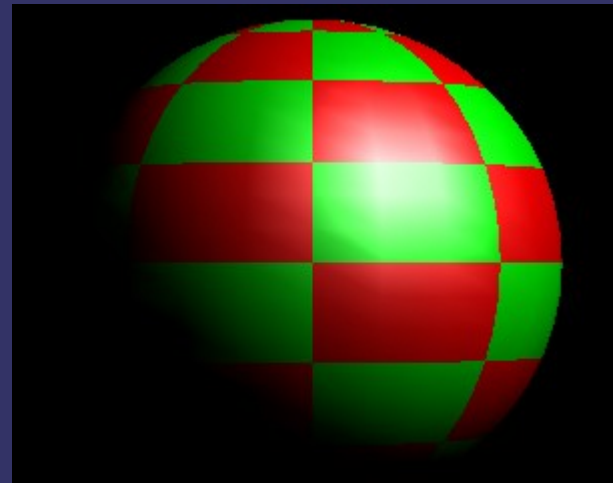
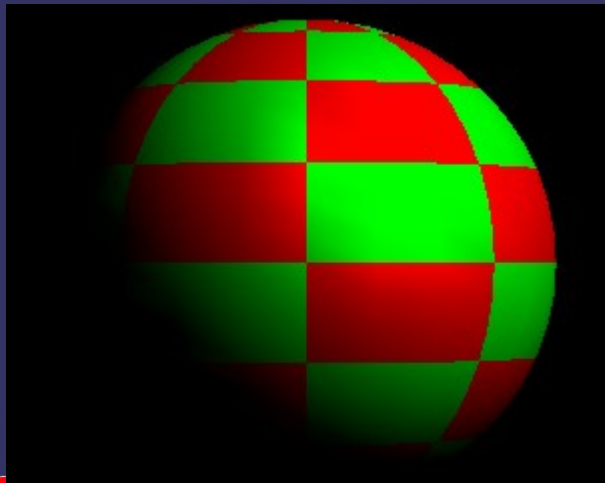
12-February-2009

© Copyright Ian D. Romanick 2009

Specular Lighting

⇒ How can we fix this?

- Perform lighting per-pixel in the fragment shader
- Send diffuse color and specular color *separately* from the vertex shader to the fragment shader



12-February-2009

© Copyright Ian D. Romanick 2009

Next week...

- More texture mapping
 - Sampling and filtering
 - Environment mapping
 - Compression
- Assignments:
 - All of assignment #1 due
 - Start assignment #2



12-February-2009

© Copyright Ian D. Romanick 2009

Legal Statement

This work represents the view of the authors and does not necessarily represent the view of Intel 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.



12-February-2009

© Copyright Ian D. Romanick 2009