CS4621/5621 Fall 2015

Basics of OpenGL/GLSL
Textures Basics

Professor: Kavita Bala
Instructor: Nicolas Savva

with slides from Balazs Kovacs, Eston Schweickart, Daniel Schroeder, Jiang Huang and Pramook Khungurn over the years.

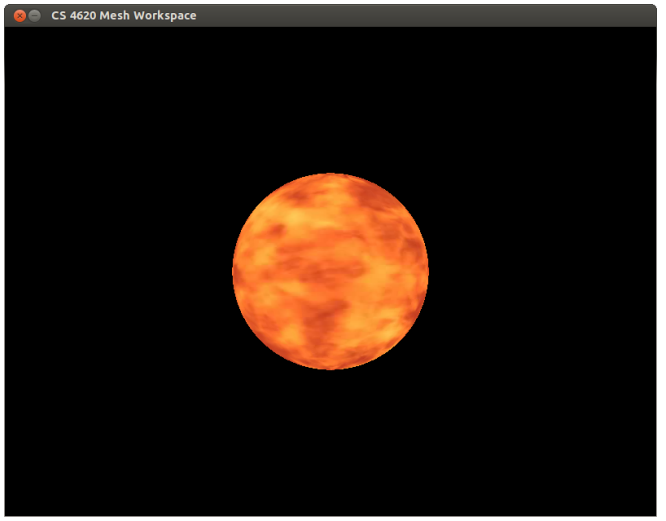
Announcements

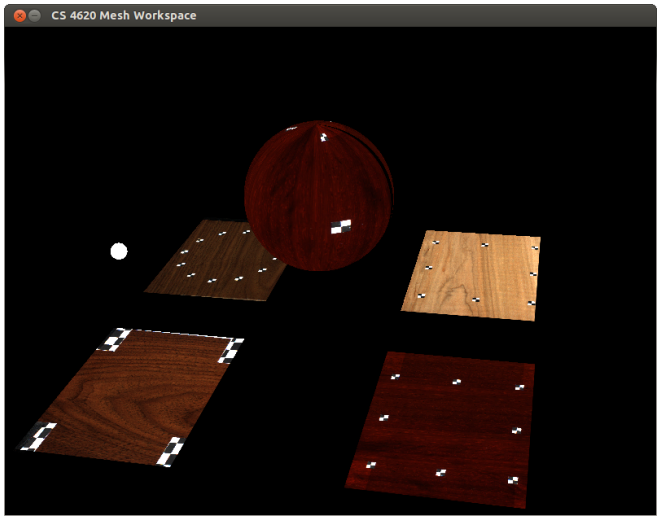
- PPA1 GPURay due tonight
- Note on *rasterization* vs *raytracing*
- PPA2 Shaders and Textures (out soon)
- Final projects

Today

- PPA2 Overview (Shading and Textures)
- Time evolution
- Textures basics
- Shadow shader

PPA2 Textures and Shading

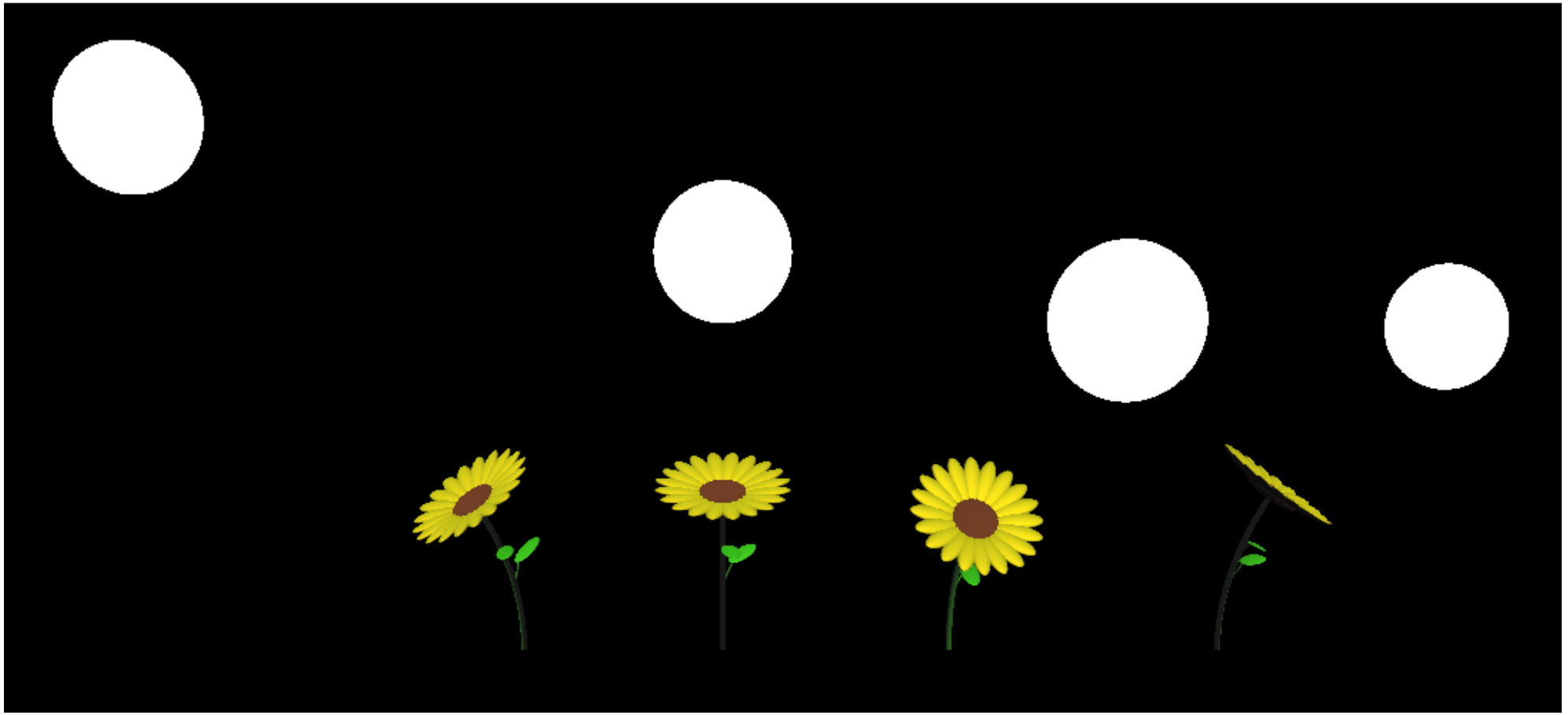




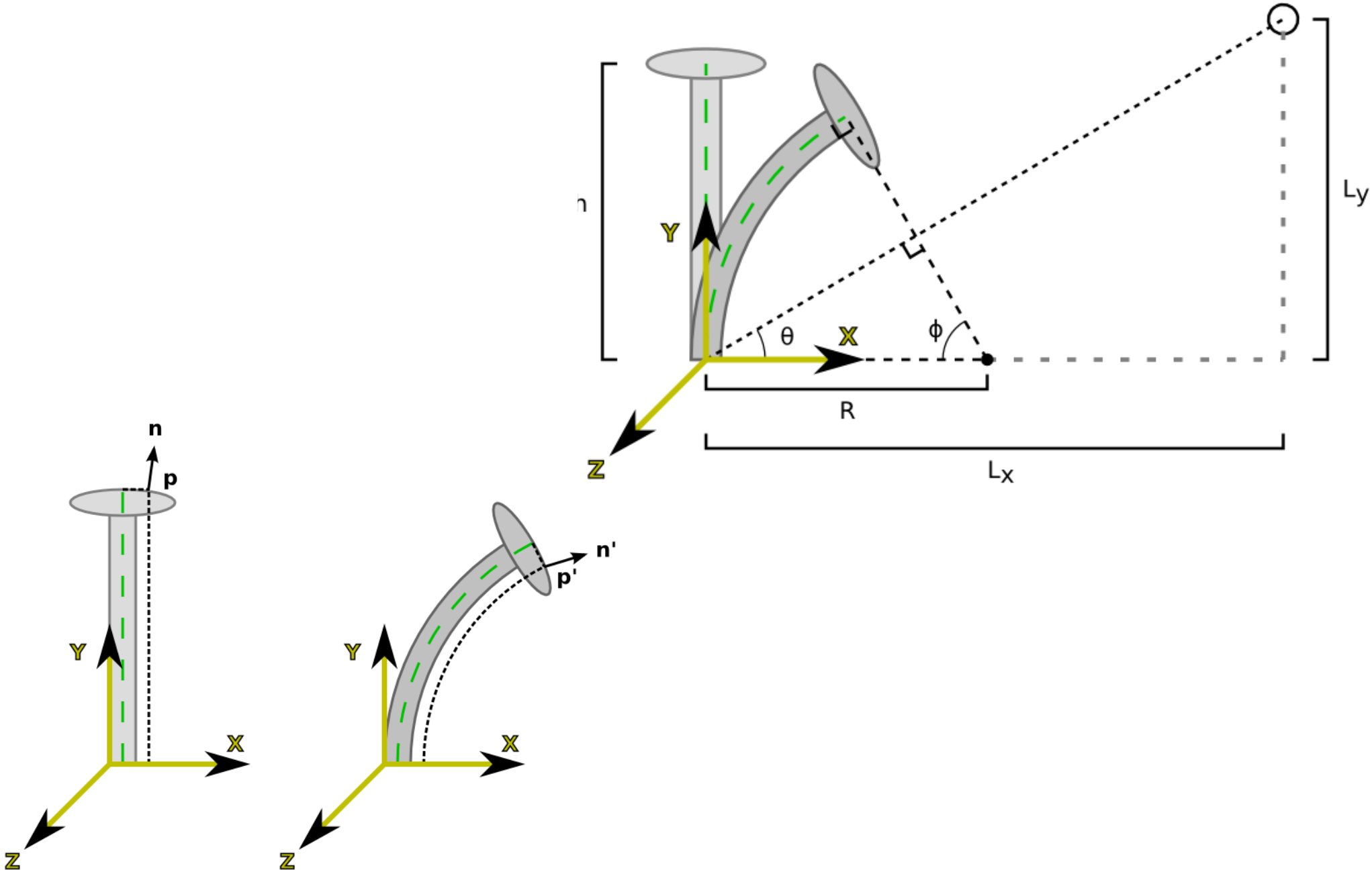
PPA2 Textures and Shading Tasks

- Deformable flower shader
- Animated fire shader
- Finished-wood shader

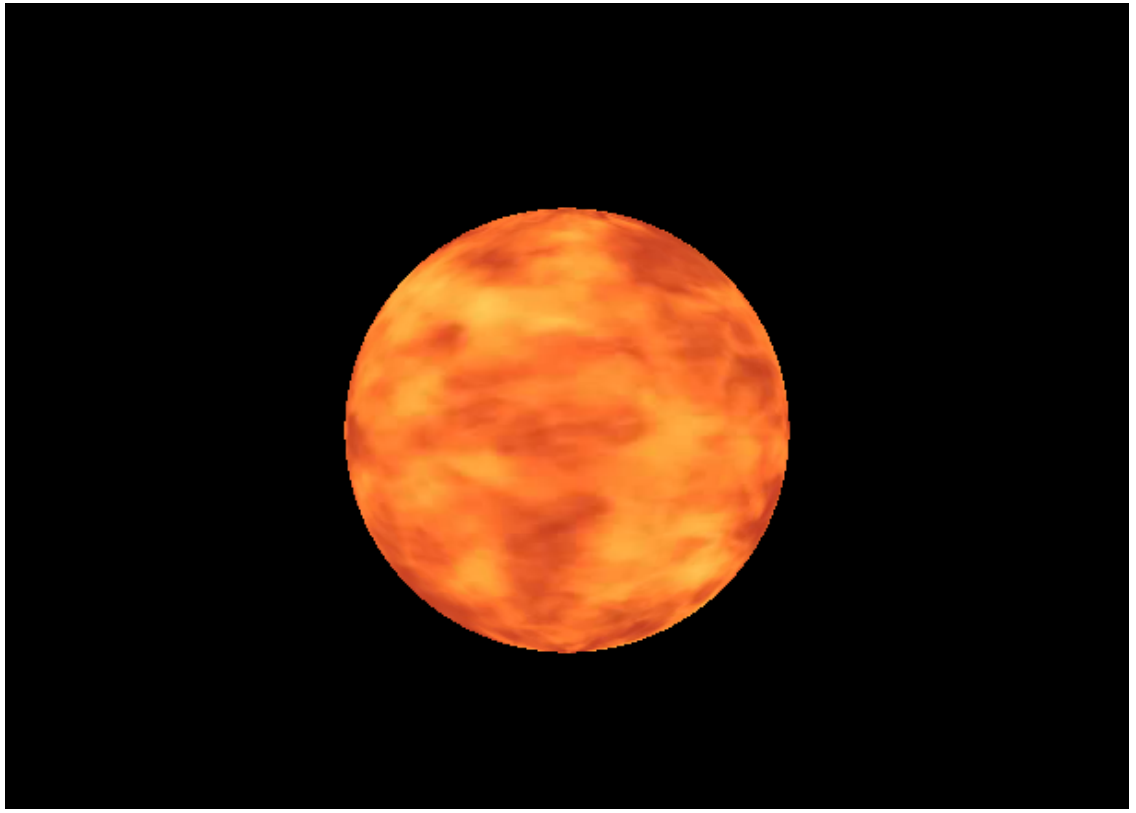
Deformable Flower Shader

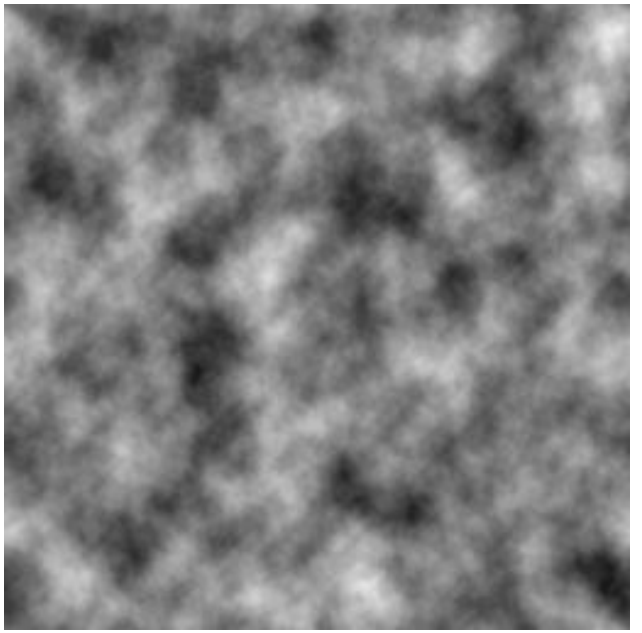


Deformable Flower Shader

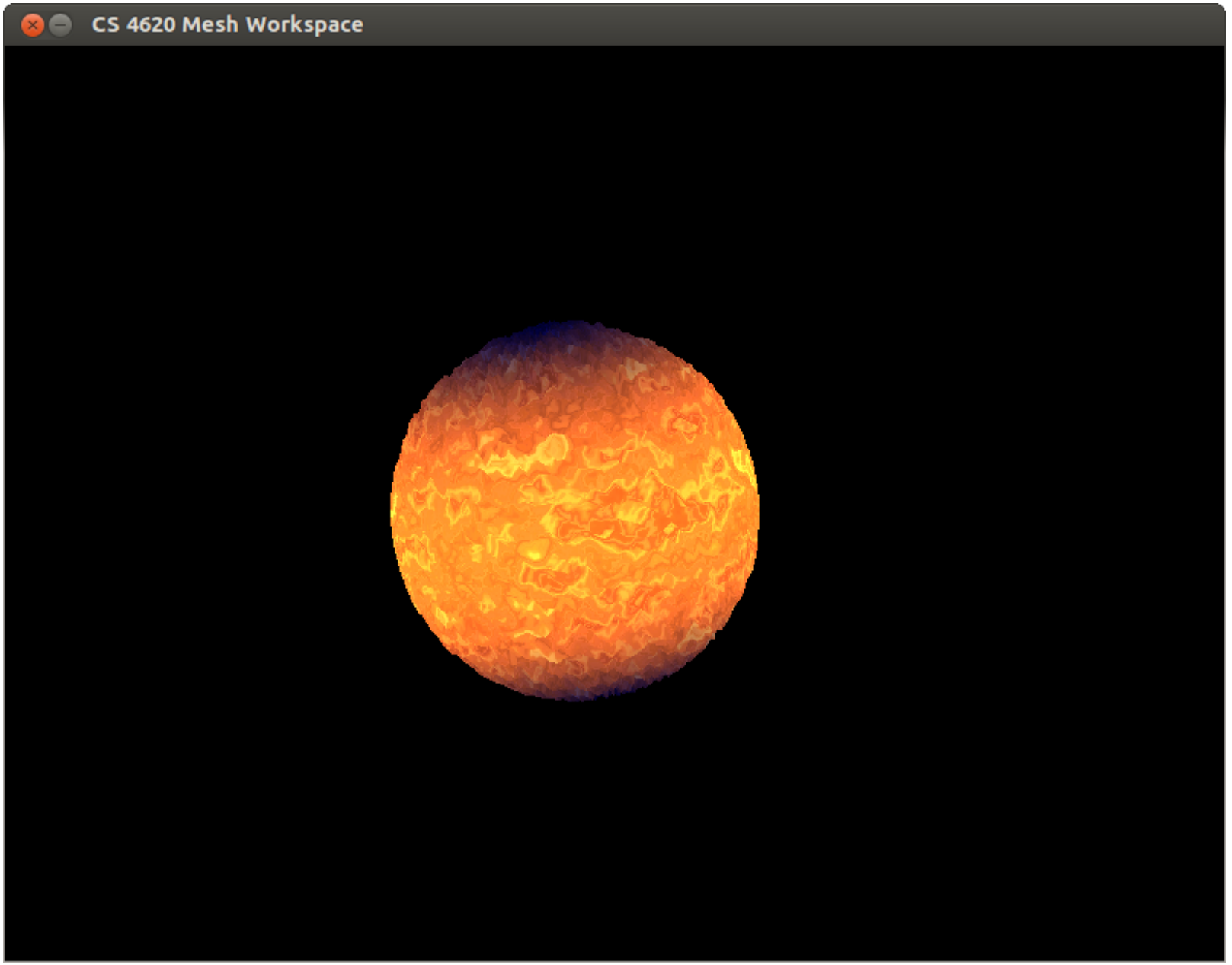


Fire Shader

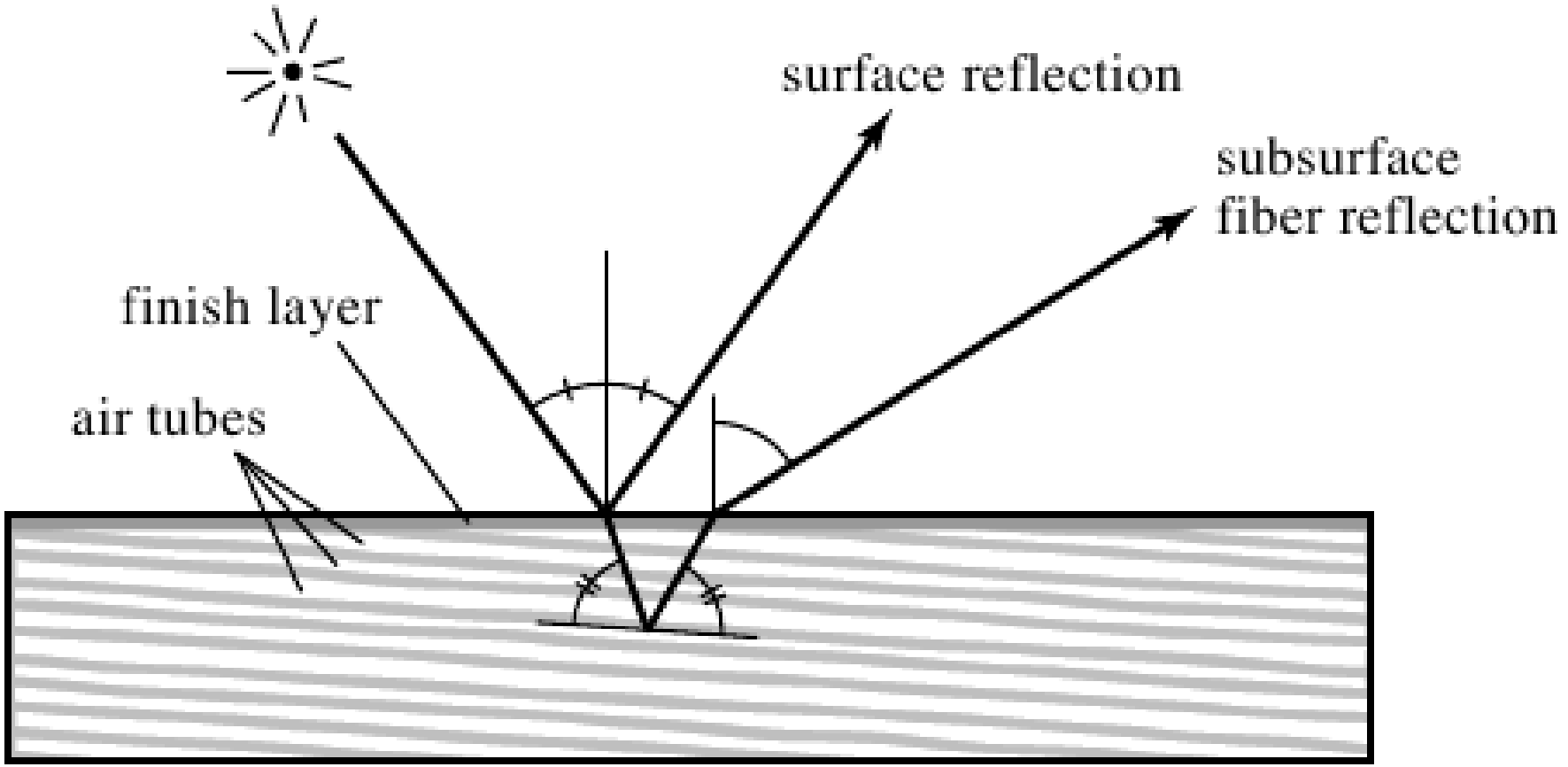




Fire Shader



Finished-wood Shader



Finished-wood Shader Model

$$f_r(v_i, v_r) = f_s(v_i, v_r) + T_i T_r(\rho_d + f_f(u, v_i, v_r))$$

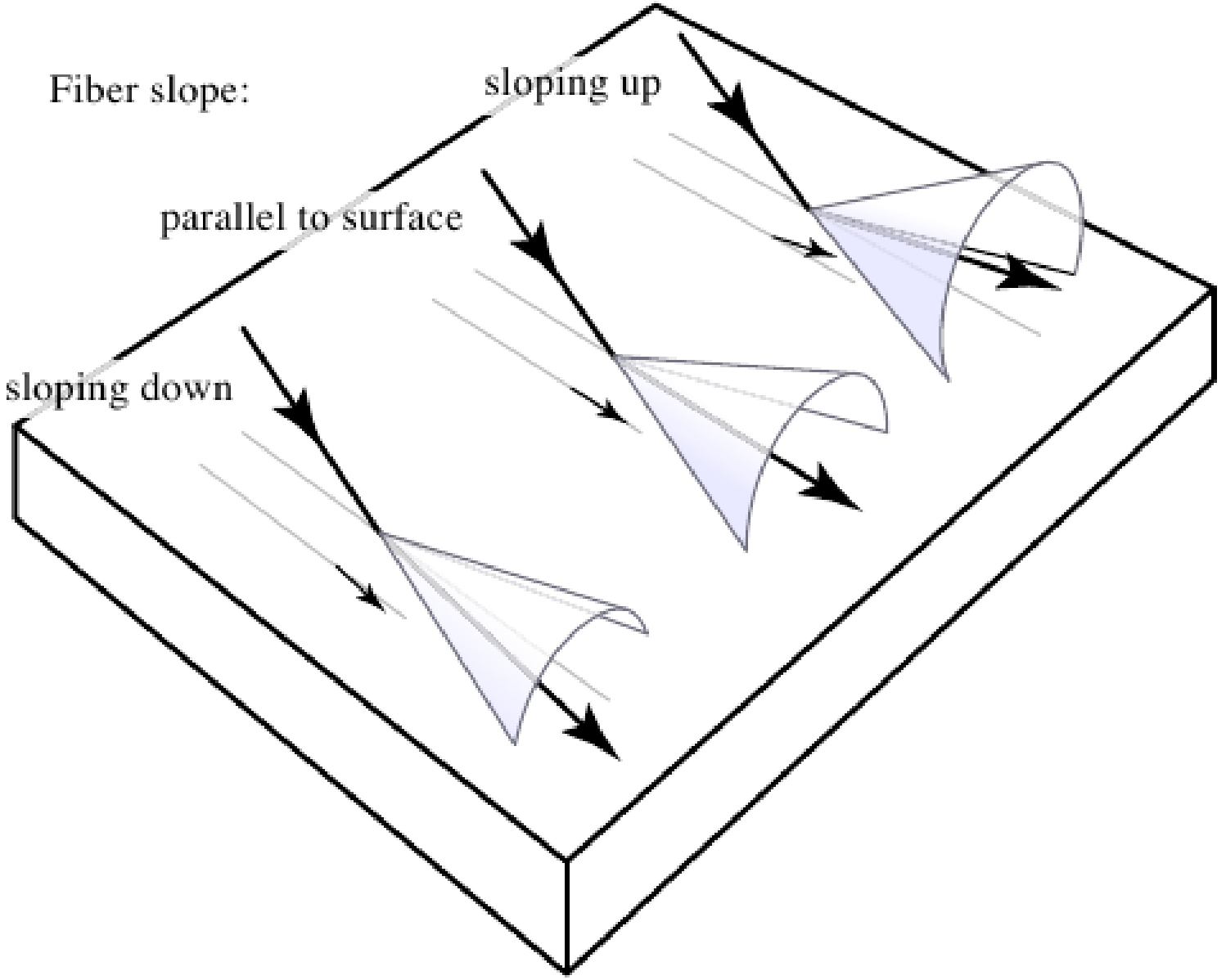
v_i is the incoming light direction, v_r is the reflected light direction, T_i and T_r are attenuation factors for the surface finish using Fresnel reflection factors, ρ_d is the diffuse color, u is the fiber direction, and f_f is

$$f_f(u, v_i, v_r) = k_f \frac{g(\beta, \psi_h)}{0.5 * cos^2(\psi_d)}$$

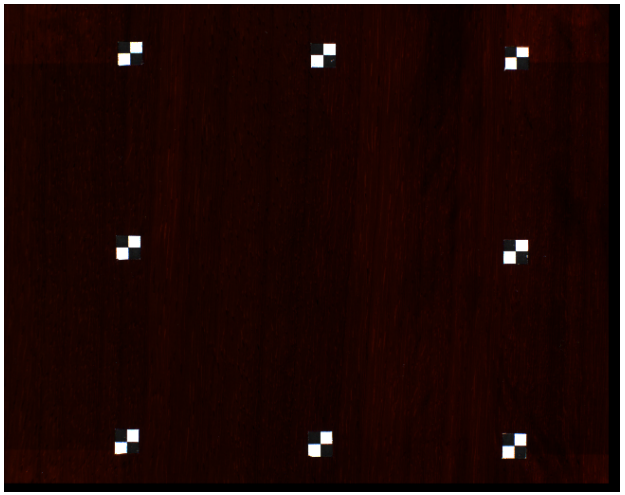
where:

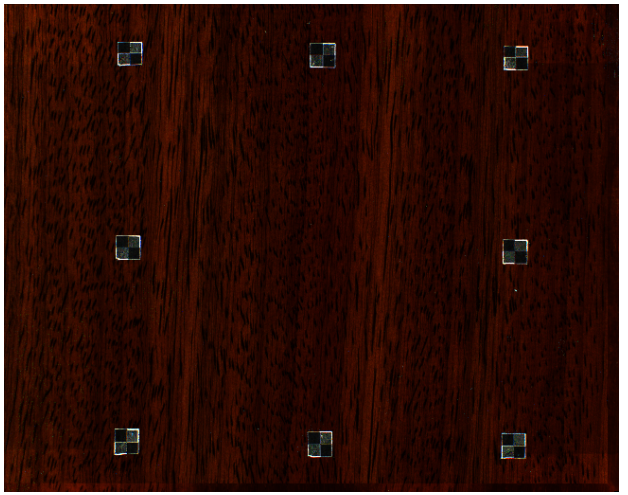
$$\psi_d = \psi_r - \psi_i$$
$$\psi_h = \psi_r + \psi_i$$
$$\psi_r = sin^{-1}(s(v_r) \cdot u)$$
$$\psi_i = sin^{-1}(s(v_i) \cdot u)$$

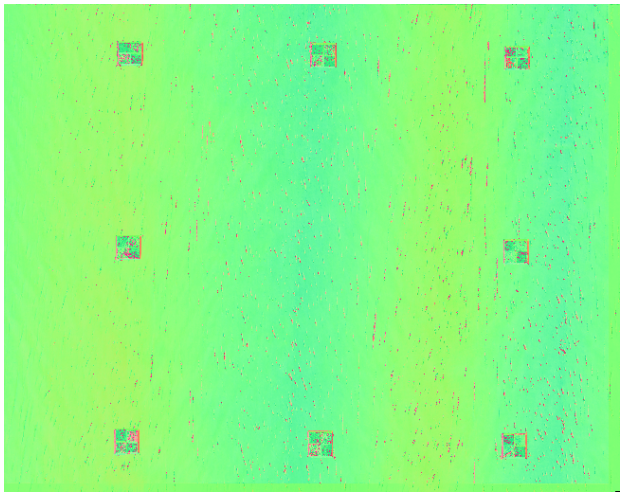
Finished-wood Shader



Finished-wood Shader







Demo: PPA2 Textures and Shading

Height Fields

- A height field is a function $y = f(x, z)$
  - y represents the height of a point for a location in the x-z plane.
- Heights fields are usually rendered as a rectangular mesh of triangles or rectangles sampled from a grid
  - samples $y_{ij} = f(x_i , z_j)$

Displaying a Height Field

First, generate a mesh data and use it to initialize data for a VBO:

```
float dx = 1.0/N, dz = 1.0/N;
for( int i = 0; i < N; ++i ) {
    float x = i*dx;
    for( int j = 0; j < N; ++j ) {
        float z = j*dz;
        float y = f( x, z );
        vertex[Index++] = vec3(      x, y, z );
        vertex[Index++] = vec3(      x, y, z + dz );
        vertex[Index++] = vec3( x + dx, y, z + dz );
        vertex[Index++] = vec3( x + dx, y, z );
    }
}
```

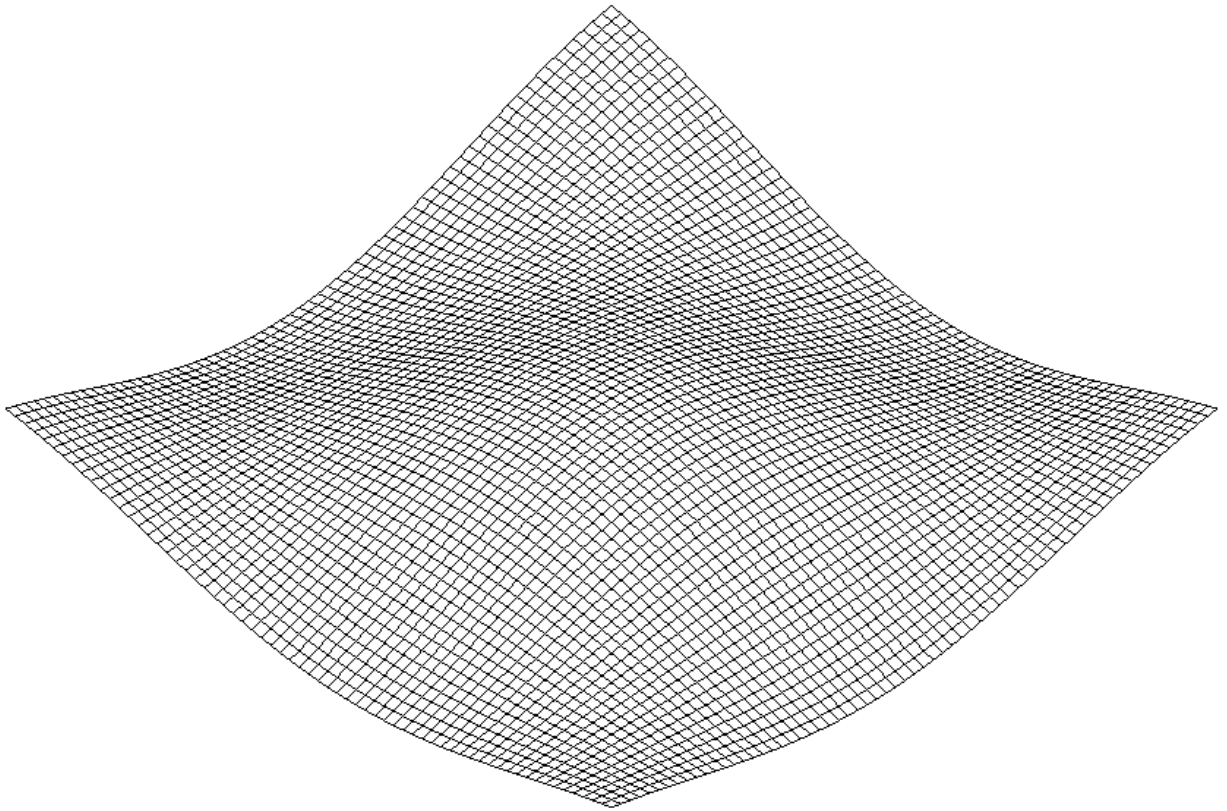
Finally, display each quad using:

```
for( int i = 0; i < NumVertices ; i += 4 )
    glDrawArrays( GL_LINE_LOOP, 4*i, 4 );
```

Time Varying Vertex Shader

```
in vec4 vPosition;
in vec4 vColor;

uniform float time; // in milliseconds
uniform mat4 ModelViewProjectionMatrix;

void main()
{
    vec4  v = vPosition;
    vec4  u = sin( time + 5*v );

    v.y = 0.1 * u.x * u.z;

    gl_Position = ModelViewProjectionMatrix * v;
}
```

Shaded Mesh



Adding Lighting

- Solid Mesh: create two triangles for each quad
- Display with:

```
glDrawArrays( GL_TRIANGLES, 0, NumVertices );
```

- For more interest looking results, we'll add lighting (and perhaps a texture)
- We'll do per-vertex lighting
  - leverage the vertex shader since we'll also use it to vary the mesh in a time-varying way

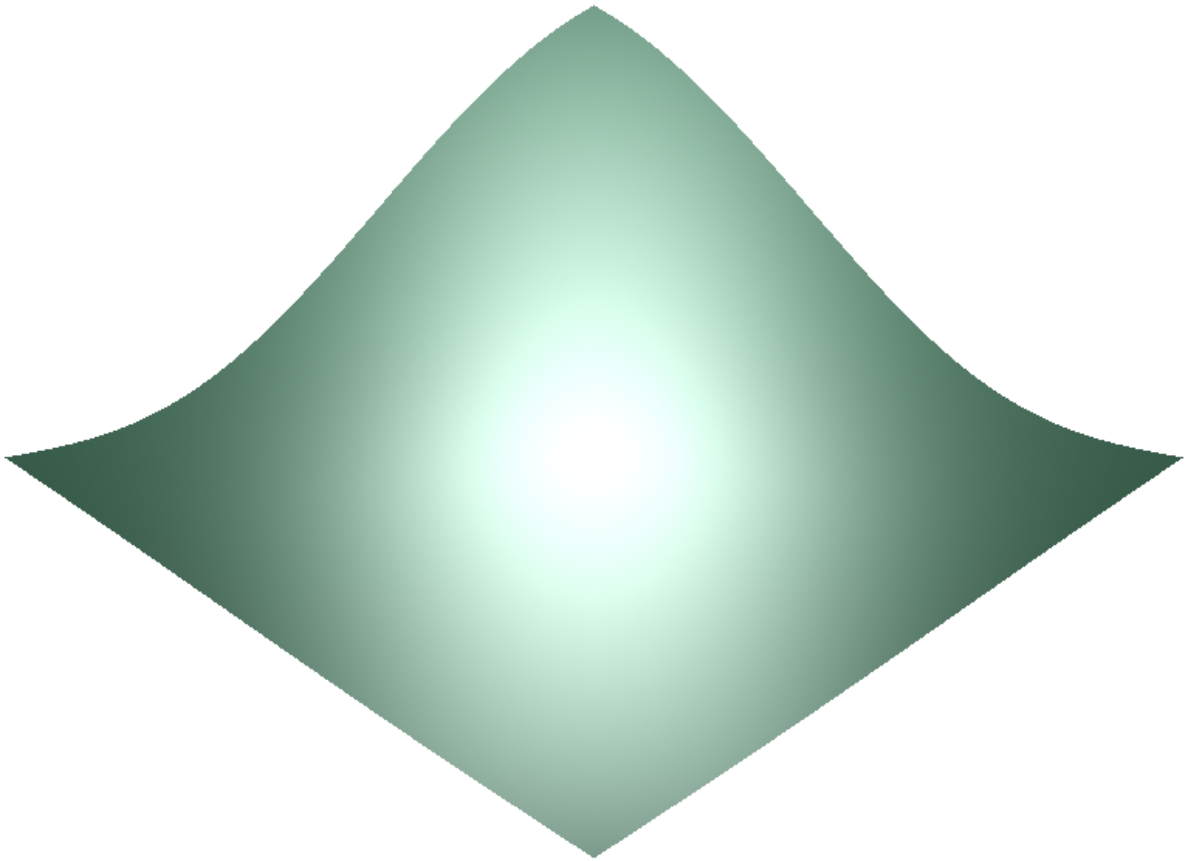
Mesh Shader

```
uniform float time, shininess;
uniform vec4
    vPosition, lightPosition, diffuseLight, specularLight;
uniform mat4
    ModelViewMatrix, ModelViewProjectionMatrix, NormalMatrix;


void main()
{
   vec4  v = vPosition;
   vec4  u = sin( time + 5*v );
   v.y = 0.1 * u.x * u.z;

   gl_Position = ModelViewProjectionMatrix * v;

   vec4 diffuse, specular;
   vec4 eyePosition = ModelViewMatrix * vPosition;
   vec4 eyeLightPos = lightPosition;
```

Mesh Shader (continued)

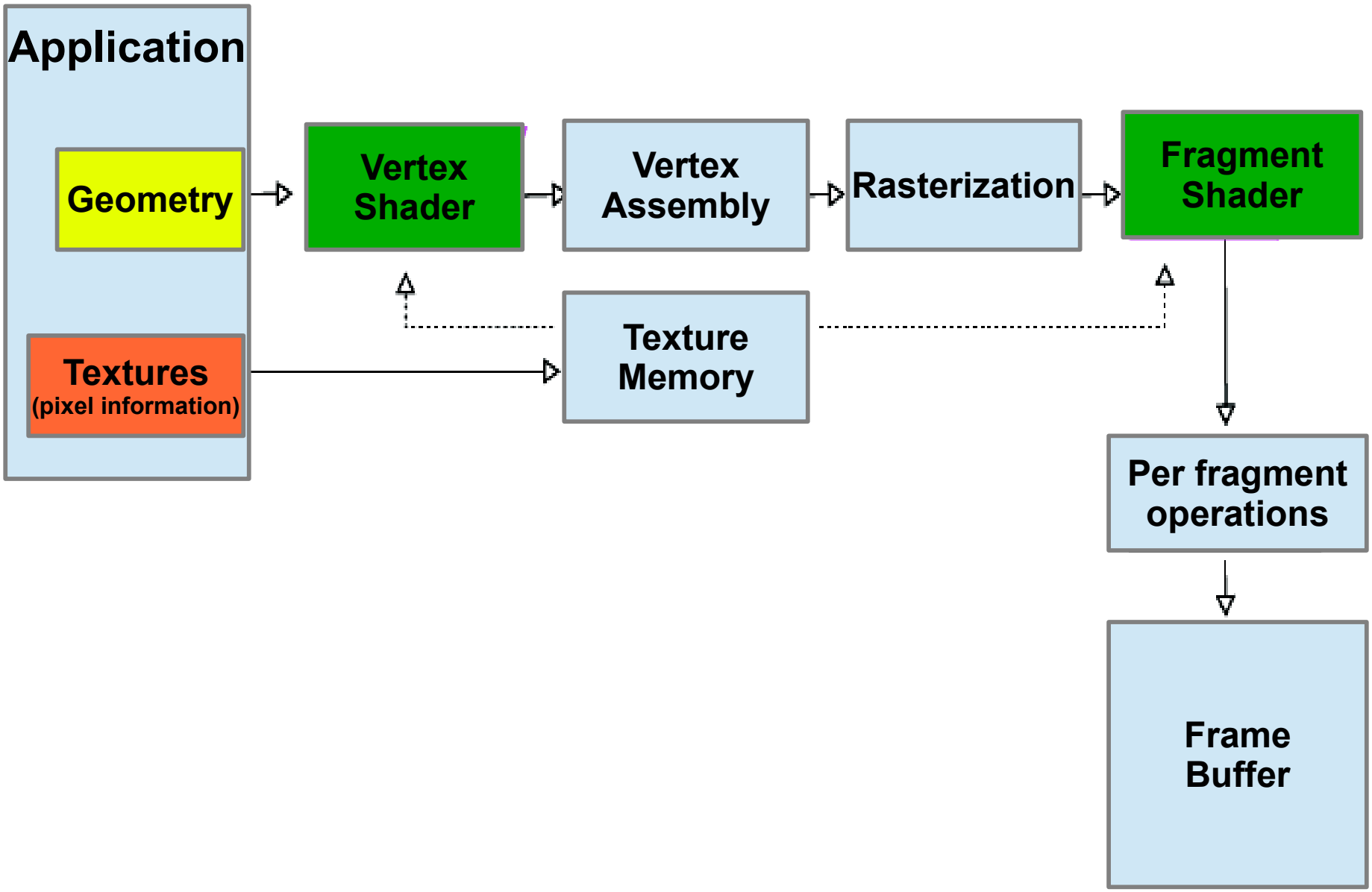
```
  vec3 N = normalize(NormalMatrix * Normal);
  vec3 L = normalize(vec3(eyeLightPos –
eyePosition));
  vec3 E = -normalize(eyePosition.xyz);
  vec3 H = normalize(L + E);


    float Kd = max(dot(L, N), 0.0);
    float Ks = pow(max(dot(N, H), 0.0), shininess);
    diffuse  = Kd*diffuseLight;
    specular = Ks*specularLight;
    color    = diffuse + specular;
}
```

Mesh Display



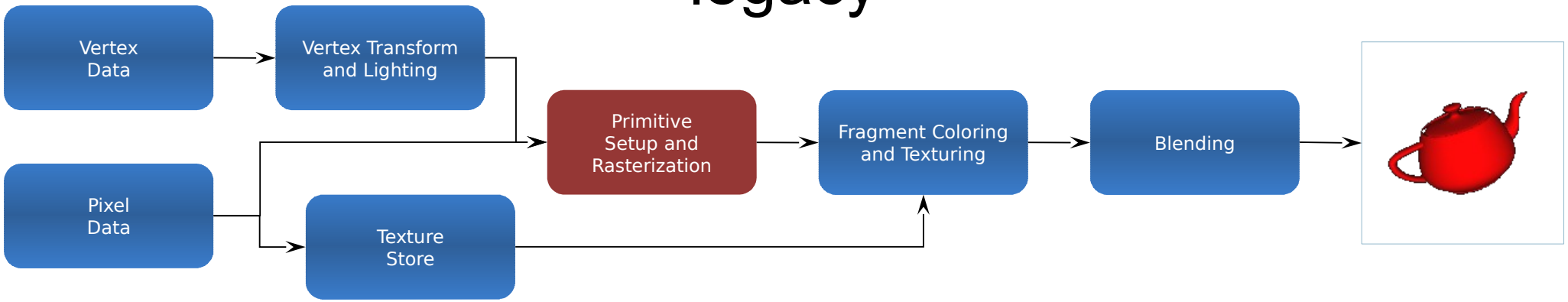
Animated Texture Demo

Texturing in GLSL/Pipeline

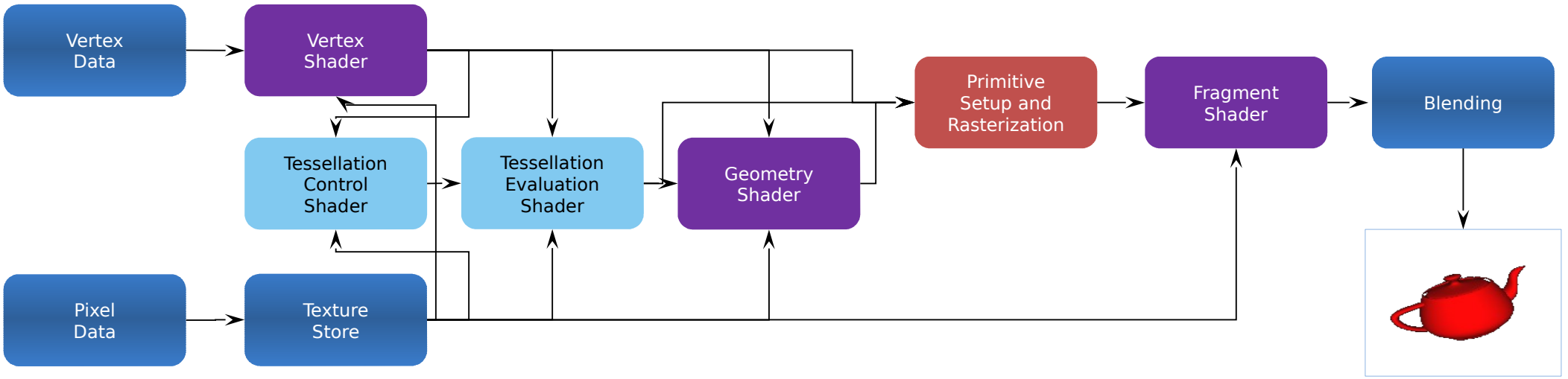


Texture Mapping and the OpenGL Pipeline

legacy

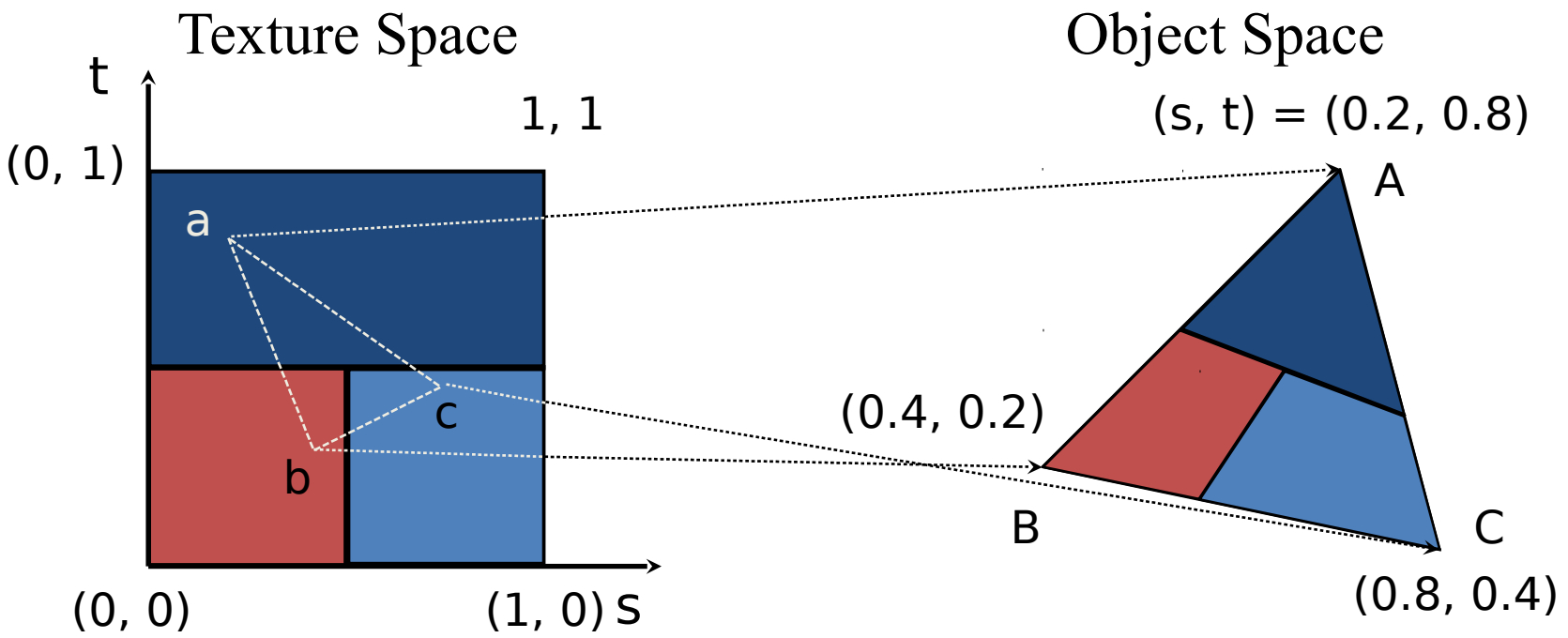


modern



Mapping a Texture

- Based on parametric texture coordinates
- coordinates needs to be specified at each vertex



Applying Textures

- Basic steps to applying a texture:
  1. specify the texture
  2. read or generate image
  3. assign to texture
  4. enable texturing
  5. assign texture coordinates to vertices
  6. specify texture parameters
  7. wrapping, filtering

Using textures

In order to use texture mapping in your application, you will need to do the following steps:

- Create a texture object and load texel data into it.
- Include texture coordinates with your vertices.
- Associate a texture sampler with each texture map you intend to use in your shader.
- Retrieve the texel values through the texture sampler from your shader.

Texture Targets

Target (GL_TEXTURE*	Sampler Type	Dimensionality
1D	`sampler1D`	1D
1D_ARRAY	`sampler1DArray`	1D array
2D	`sampler2D`	2D
2D_ARRAY	`sampler2DArray`	2D array
2D_MULTISAMPLE	`sampler2DMS`	2D multisample
2D_MULTISAMPLE_ARRAY	`sampler2DMSArray`	2D multisample array
3D	`sampler3D`	3D
CUBE	`samplerCube`	cube-map texture
ARRAY	`samplerCubeArray`	cube-map array
RECTANGLE	`samplerRect`	2D rectangle
BUFFER	`samplerBuffer`	1D buffer

Texture Format

Sized Internal Format	Base Internal Format	R Bits	G Bits	B Bits	A Bits	Shared Bits
GL_R8	GL_RED	8				
GL_R8_SNORM	GL_RED	s8				
GL_R16	GL_RED	16				
GL_R16_SNORM	GL_RED	s16				
GL_RG8	GL_RG	8	8			
GL_RG8_SNORM	GL_RG	s8	s8			
GL_RG16	GL_RG	16	16			
GL_RG16_SNORM	GL_RG	s16	s16			
GL_R3_G3_B2	GL_RGB	3	3	2		
GL_RGB4	GL_RGB	4	4	4		
GL_RGB5	GL_RGB	5	5	5		
GL_RGB565	GL_RGB	5	6	5		
GL_RGB8	GL_RGB	8	8	8		
GL_RGB8_SNORM	GL_RGB	s8	s8	s8		
GL_RGB10	GL_RGB	10	10	10		
GL_RGB12	GL_RGB	12	12	12		
GL_RGB16	GL_RGB	16	16	16		
GL_RGB16_SNORM	GL_RGB	s16	s16	s16		
GL_RGBA2	GL_RGBA	2	2	2	2	
GL_RGBA4	GL_RGBA	4	4	4	4	
GL_RGB5_A1	GL_RGBA	5	5	5	1	
GL_RGBA8	GL_RGBA	8	8	8	8	

Format Token	Component Layout
GL_UNSIGNED_BYTE_3_3_2	
GL_UNSIGNED_BYTE_2_3_2_REV	
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_10_10_10_2	
GL_UNSIGNED_INT_2_10_10_10_REV	
GL_UNSIGNED_INT_10F_11F_11F_REV	
GL_UNSIGNED_INT_5_9_9_9_REV	

Textures in the CS4620 Framework

- Loading texture files as texture maps (glTexImage2D(...))
- Setting up the texture parameters (glTexParameteri(...))
- Manages the texture units (glBindTexture(...))

Provides wrapper classes for working with 1D, 2D
(and 2D MIP-Mapped textures)

Simple interface for using textures with GLSL

Textures in OpenGL

- **glEnable(GL_TEXTURE_2D)**
  turn on the 2D texture store

- **glTexImage2D(...)**
  declares a texture's size, color components (RGBA, etc), data type (byte, float...), pixel data

- **glBindTexture(...)**
  "bind" the given texture to the active store
  - Programmable shaders can bind textures to uniforms

Textures in OpenGL

GlTexParameteri(...)

Used to set texture configuration:

How are the texture values interpolated?

GL_NEAREST vs GL_LINEAR :

GL_NEAREST rounds to nearest texel to get the color
GL_LINEAR linearly interpolates the colors of the texels

Does the texture repeat itself?

GL_REPEAT vs GL_CLAMP

Say we have a texture coordinate of (-0.1,1.1)
GL_CLAMP changes it to (0.0,1.0)
GL_REPEAT changes it to (0.9,0.1)

More options for Texture Parameters can be found here:
https://www.opengl.org/sdk/docs/man/html/glTexParameter.xhtml
Examples of use can be found in the Texture class in the framework

Texture Borders

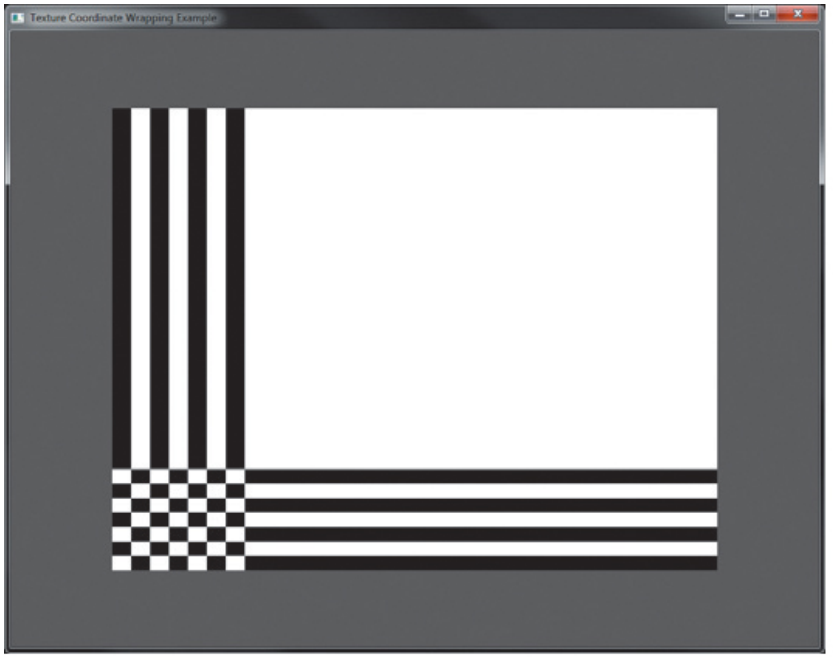
GL_CLAMP_TO_EDGE whenever outside the range 0.0 to 1.0, texels on the edge of the texture are used

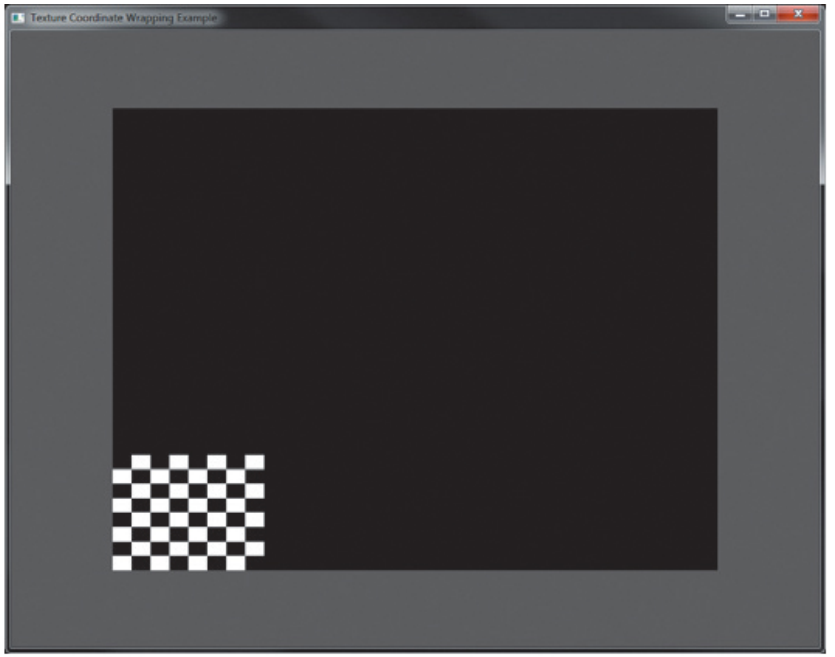
•GL_CLAMP_TO_BORDER attempts to read outside the texture will result in the constant border color (GL_TEXTURE_BORDER_COLOR) for the texture being used

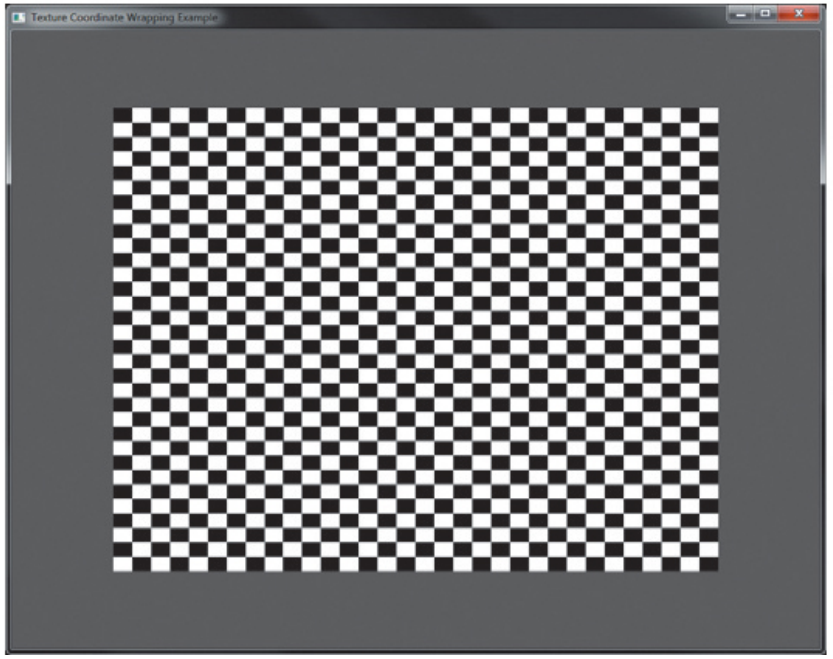
• GL_REPEAT the texture is wrapped and considered to repeat infinitely (fractional part of the uvcoord is used whereas integer part is discarded)

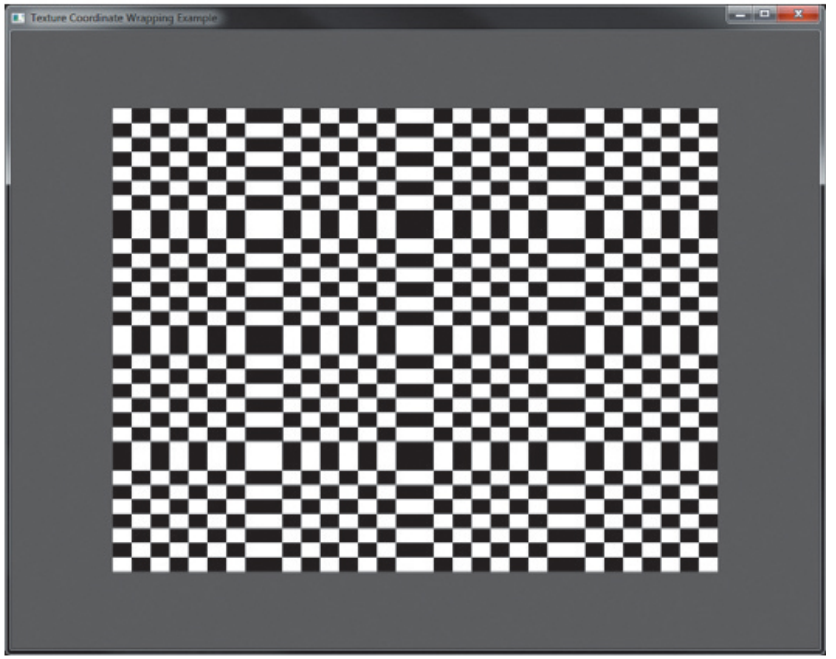
• GL_MIRRORED_REPEAT texture repeated in a mirrored fashion (even integer uvcoords part fractional part considered whereas coordinates with odd integer part have their fractional part subtracted from 1.0)

Texture Borders









GLSL Data Types

- Scalar types: `float, int, bool`
- Vector types: `vec2, vec3, vec4`
  `ivec2, ivec3, ivec4`
  `bvec2, bvec3, bvec4`
- Matrix types: `mat2, mat3, mat4`
- **Texture sampling: `sampler1D, sampler2D, sampler3D, samplerCube`**

Texturing in GLSL

Basic shader syntax:

- sampler2D - a new data type
- vec4 texture2D(sampler2D,vec2) – a new built-in function

Texturing in GLSL (Vertex Shader)

Figure out the coordinate that we want to sample from by using an attribute variable

```
attribute vec2 in_TexCoord;
varying vec2 coord;

void main() {
   gl_Position = ...

   coord = vec2(in_TexCoord);
}
```

Texturing in GLSL (Fragment Shader)

Take the coordinate data from the vertex shader and sample the appropriate pixel from the desired texture

```
varying vec2 coord;
uniform sampler2D sampler;

void main() {
   gl_FragColor = texture2D(sampler, coord);

}
```

Textures in CS 4620 Framework

In onEntry() method // Initialization step :

```
// Load the 2D texture
texture = new GLTexture(TextureTarget.Texture2D, true);

try {
   texture.setImage2DResource("path/to/image.png",
false);
} catch (Exception e) {
   // No image found!
   System.out.println(e.getMessage());
   System.exit(1);
}
```

Textures in CS 4620 Framework

Inside draw() method // Render step:

```
program.use(); // Activate the shader

// Activate the texture and bind the active texture
unit to the sampler uniform

texture.use(TextureUnit.Texture0,
   program.getUniform("sampler"));

glDrawElements(..); // Render your scene

// clean up
texture.unuse();
GLProgram.unuse();
```

Texture Object

```
GLuint textures[1];
glGenTextures( 1, textures ); //reserve names for texture objects


glActiveTexture( GL_TEXTURE0 ); //select active texture unit
glBindTexture( GL_TEXTURE_2D, textures[0] ); //create texture object
//create, activate bind and unbind texture
//(use dimensionality of target)


glTexImage2D( GL_TEXTURE_2D, 0, GL_RGB, TextureSize,
              TextureSize, GL_RGB, GL_UNSIGNED_BYTE, image );
//mutable texture image storage


glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT );
glTexParameteri( GL_TEXTURE_2D,
                 GL_TEXTURE_MAG_FILTER, GL_NEAREST );
glTexParameteri( GL_TEXTURE_2D,
                 GL_TEXTURE_MIN_FILTER, GL_NEAREST );
```

Vertex Shader

```
in vec4 vPosition;
in vec4 vColor;
in vec2 vTexCoord;

out vec4 color;
out vec2 texCoord;

void main()
{
    color       = vColor;
    texCoord    = vTexCoord;
    gl_Position = vPosition;
}
```

Fragment Shader

```
in vec4 color;
in vec2 texCoord;

out vec4 fColor;

uniform sampler texture;

void main()
{
  fColor = color * texture( texture,
texCoord );
}
```

Multitextures

Multitextures (client side)

```
...
// For the first texture, we will use texture unit 0

// Get the uniform location
GLint tex1_uniform_loc = glGetUniformLocation(prog,"tex1");
// Set it to 0
glUniform1i(tex1_uniform_loc, 0);
// Select texture unit 0
glActiveTexture(GL_TEXTURE0);
// Bind a texture to it
glBindTexture(GL_TEXTURE_2D, tex1);

// Repeat the above process for texture unit 1
GLint tex2_uniform_loc = glGetUniformLocation(prog, "tex2");
glUniform1i(tex2_uniform_loc, 1);
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, tex2);
...
```

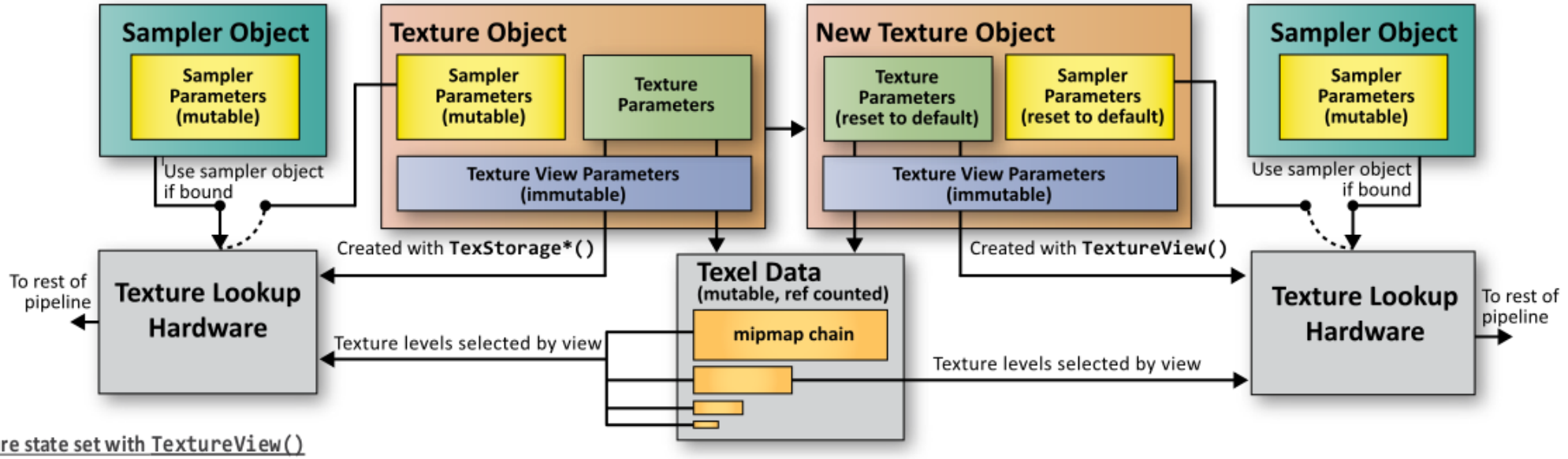
Multitexture (Vertex Shader)

```
#version 330 core

layout (location = 0) in vec2 in_position;
layout (location = 1) in vec2 in_tex_coord;

out vec2 tex_coord0;
out vec2 tex_coord1;

uniform float time;

void main(void)
{
    const mat2 m = mat2(vec2(cos(time), sin(time)),
                        vec2(-sin(time),
cos(time)));
    tex_coord0 = in_tex_coord * m;
    tex_coord1 = in_tex_coord * transpose(m);
    gl_Position = vec4(in_position, 0.5, 1.0);
}
```

Multitexture (Fragment Shader)

```
#version 330 core

in vec2 tex_coord0;
in vec2 tex_coord1;

layout (location = 0) out vec4 color;

uniform sampler2D tex1;
uniform sampler2D tex2;

void main(void)
{
    color = texture(tex1, tex_coord0) +
texture(tex2, tex_coord1);
}
```

Texture View and Texture Object State



Texture state set with `TextureView()`

```
enum internalformat // base internal format
enum target         // texture target
uint minlevel       // first level of mipmap
uint numlevels      // number of mipmap levels
uint minlayer       // first layer of array texture
uint numlayers      // number of layers in array
```

Sampler Parameters (mutable)

- TEXTURE_BORDER_COLOR
- TEXTURE_COMPARE_{FUNC,MODE}
- TEXTURE_LOD_BIAS
- TEXTURE_{MAX,MIN}_LOD
- TEXTURE_{MAG,MIN}_FILTER
- TEXTURE_WRAP_{S,T,R}

Texture Parameters (immutable)

- TEXTURE_WIDTH
- TEXTURE_HEIGHT
- TEXTURE_DEPTH
- TEXTURE_FIXED_SAMPLE_LOCATIONS
- TEXTURE_COMPRESSED
- TEXTURE_COMPRESSED_IMAGE_SIZE
- TEXTURE_IMMUTABLE_FORMAT
- TEXTURE_SAMPLES

Texture Parameters (mutable)

- TEXTURE_SWIZZLE_{R,G,B,A}
- TEXTURE_MAX_LEVEL
- TEXTURE_BASE_LEVEL
- DEPTH_STENCIL_TEXTURE_MODE

Texture View Parameters (immutable)

- <target>
- TEXTURE_INTERNAL_FORMAT
- TEXTURE_SHARED_SIZE
- TEXTURE_VIEW_{MIN,NUM}_LEVEL
- TEXTURE_VIEW_{MIN,NUM}_LAYER
- TEXTURE_IMMUTABLE_LEVELS
- IMAGE_FORMAT_COMPATIBILITY_TYPE
- TEXTURE_{RED,GREEN,BLUE,ALPHA,DEPTH}_TYPE
- TEXTURE_{RED,GREEN,BLUE,ALPHA,DEPTH,STENCIL}_SIZE

OpenGL reference

Textures and Samplers [8]

void **ActiveTexture**(enum *texture*);
texture: TEXTURE*i* (where *i* is [0, max(MAX_TEXTURE_COORDS, MAX_COMBINED_TEXTURE_IMAGE_UNITS)-1])

Texture Objects [8.1]

void **GenTextures**(sizei *n*, uint **textures*);

void **BindTexture**(enum *target*, uint *texture*);
target: TEXTURE_{1D, 2D}[_ARRAY], TEXTURE_{3D, RECTANGLE, BUFFER}, TEXTURE_CUBE_MAP[_ARRAY], TEXTURE_2D_MULTISAMPLE[_ARRAY]

void **BindTextures**(uint *first*, sizei *count*, const uint **textures*);
target: See BindTexture

void **BindTextureUnit**(uint *unit*, uint *texture*);

void **CreateTextures**(enum *target*, sizei *n*, uint **textures*);
target: See BindTexture

void **DeleteTextures**(sizei *n*, const uint **textures*);

boolean **IsTexture**(uint *texture*);

Sampler Objects [8.2]

void **GenSamplers**(sizei *count*, uint **samplers*);

void **CreateSamplers**(sizei *n*, uint **samplers*);

void **BindSampler**(uint *unit*, uint *sampler*);

void **BindSamplers**(uint *first*, sizei *count*, const uint **samplers*);

void **SamplerParameter{i f}**(uint *sampler*, enum *pname*, T *param*);
pname: TEXTURE_*X* where *X* may be WRAP_{S, T, R}, {MIN, MAG}_FILTER, {MIN, MAX}_LOD, BORDER_COLOR, LOD_BIAS, COMPARE_{MODE, FUNC} [Table 23.18]

void **SamplerParameter{i f}v**(uint *sampler*, enum *pname*, const T **param*);
pname: See SamplerParameter{if}

void **SamplerParameterI{i ui}v**(uint *sampler*, enum *pname*, const T **params*);
pname: See SamplerParameter{if}

void **DeleteSamplers**(sizei *count*, const uint **samplers*);

boolean **IsSampler**(uint *sampler*);

Sampler Queries [8.3]

void **GetSamplerParameter{i f}v**(uint *sampler*, enum *pname*, T **params*);
pname: See SamplerParameter{if}

void **GetSamplerParameterI{i ui}v**(uint *sampler*, enum *pname*, T **params*);
pname: See SamplerParameter{if}

Pixel Storage Modes [8.4.1]

void **PixelStore{i f}**(enum *pname*, T *param*);
pname: [Tables 8.1, 18.1] [UN]PACK_*X* where *X* may be SWAP_BYTES, LSB_FIRST, ROW_LENGTH, SKIP_{IMAGES, PIXELS, ROWS}, ALIGNMENT, IMAGE_HEIGHT, COMPRESSED_BLOCK_WIDTH, COMPRESSED_BLOCK_{HEIGHT, DEPTH, SIZE}

(Continued on next page) ▶

...

◀ Textures and Samplers (cont.)

void **TexStorage2D**(enum *target*, sizei *levels*, enum *internalformat*, sizei *width*, sizei *height*);
target: TEXTURE_{RECTANGLE, CUBE_MAP}, TEXTURE_{1D_ARRAY, 2D}
internalformat: See TexStorage1D

void **TexStorage3D**(enum *target*, sizei *levels*, enum *internalformat*, sizei *width*, sizei *height*, sizei *depth*);
target: TEXTURE_3D, TEXTURE_{CUBE_MAP, 2D}[_ARRAY]
internalformat: See TexStorage1D

void **TextureStorage1D**(uint *texture*, sizei *levels*, enum *internalformat*, sizei *width*);
internalformat: See TexStorage1D

void **TextureStorage2D**(uint *texture*, sizei *levels*, enum *internalformat*, sizei *width*, sizei *height*);
internalformat: See TexStorage1D

void **TextureStorage3D**(uint *texture*, sizei *levels*, enum *internalformat*, sizei *width*, sizei *height*, sizei *depth*);
internalformat: See TexStorage1D

void **TexStorage2DMultisample**(enum *target*, sizei *samples*, enum *internalformat*, sizei *width*, sizei *height*, boolean *fixedsamplelocations*);
target: TEXTURE_2D_MULTISAMPLE

void **TexStorage3DMultisample**(enum *target*, sizei *samples*, enum *internalformat*, sizei *width*, sizei *height*, sizei *depth*, boolean *fixedsamplelocations*);
target: TEXTURE_2D_MULTISAMPLE_ARRAY

void **TextureStorage2DMultisample**(uint *texture*, sizei *samples*, enum *internalformat*, sizei *width*, sizei *height*, boolean *fixedsamplelocations*);

void **TextureStorage3DMultisample**(uint *texture*, sizei *samples*, enum *internalformat*, sizei *width*, sizei *height*, sizei *depth*, boolean *fixedsamplelocations*);

Invalidate Texture Image Data [8.20]

void **InvalidateTexSubImage**(uint *texture*, int *level*, int *xoffset*, int *yoffset*, int *zoffset*, sizei *width*, sizei *height*, sizei *depth*);

void **InvalidateTexImage**(uint *texture*, int *level*);

Clear Texture Image Data [8.21]

void **ClearTexSubImage**(uint *texture*, int *level*, int *xoffset*, int *yoffset*, int *zoffset*, sizei *width*, sizei *height*, sizei *depth*, enum *format*, enum *type*, const void **data*);
format, type: See TexImage3D, pg 2 this card

void **ClearTexImage**(uint *texture*, int *level*, enum *format*, enum *type*, const void **data*);
format, type: See TexImage3D, pg 2 this card

Texture Image Loads/Stores [8.26]

void **BindImageTexture**(uint *index*, uint *texture*, int *level*, boolean *layered*, int *layer*, enum *access*, enum *format*);
access: READ_ONLY, WRITE_ONLY, READ_WRITE
format: RGBA{32,16}F, RG{32,16}F, R11F_G11F_B10F, R{32,16}F, RGBA{32,16,8}UI, RGB10_A2UI, RG{32,16,8}UI, R{32,16,8}UI, RGBA{32,16,8}I, RG{32,16,8}I, R{32,16,8}I, RGBA{16,8}, RGB10_A2, RG{16,8}, R{16,8}, RGBA{16,8}_SNORM, RG{16,8}_SNORM, R{16,8}_SNORM [Table 8.26]

void **BindImageTextures**(uint *first*, sizei *count*, const uint **textures*);