CPSC 453

Textures and Images

Demos 3 and 4

UNIVERSITY OF CALGARY

HW2

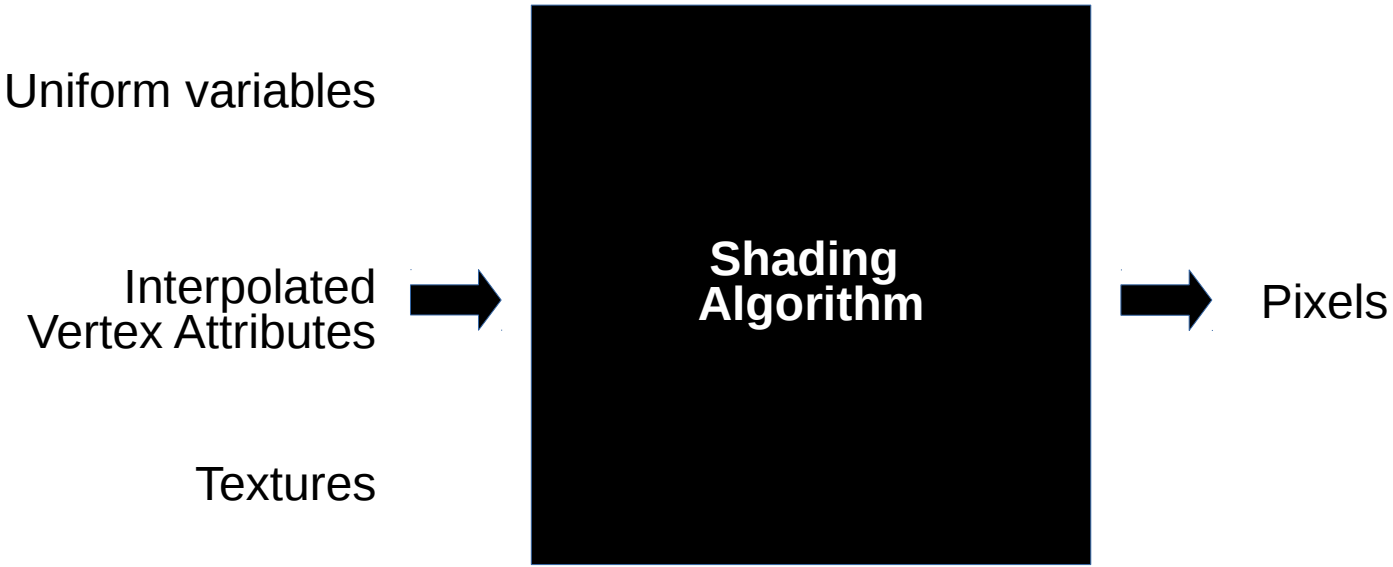
UNIVERSITY OF CALGARY

Demo 3

Game of Life

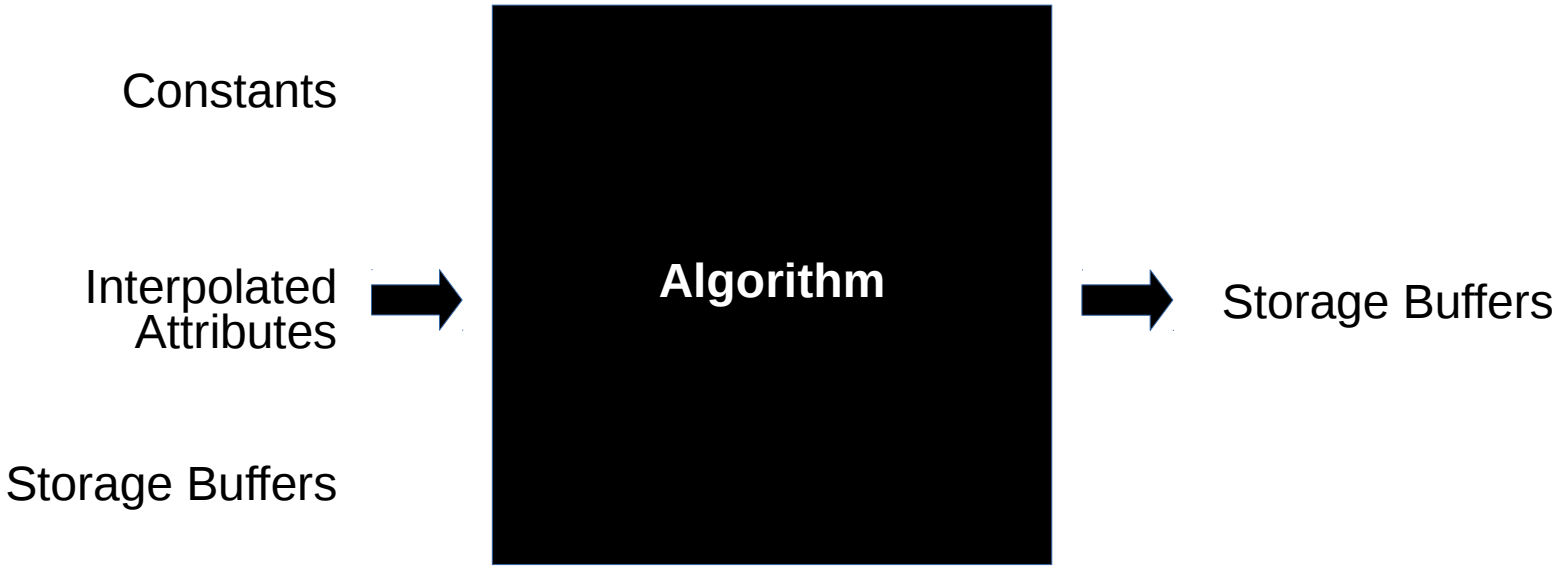
UNIVERSITY OF CALGARY

Fragment Shader

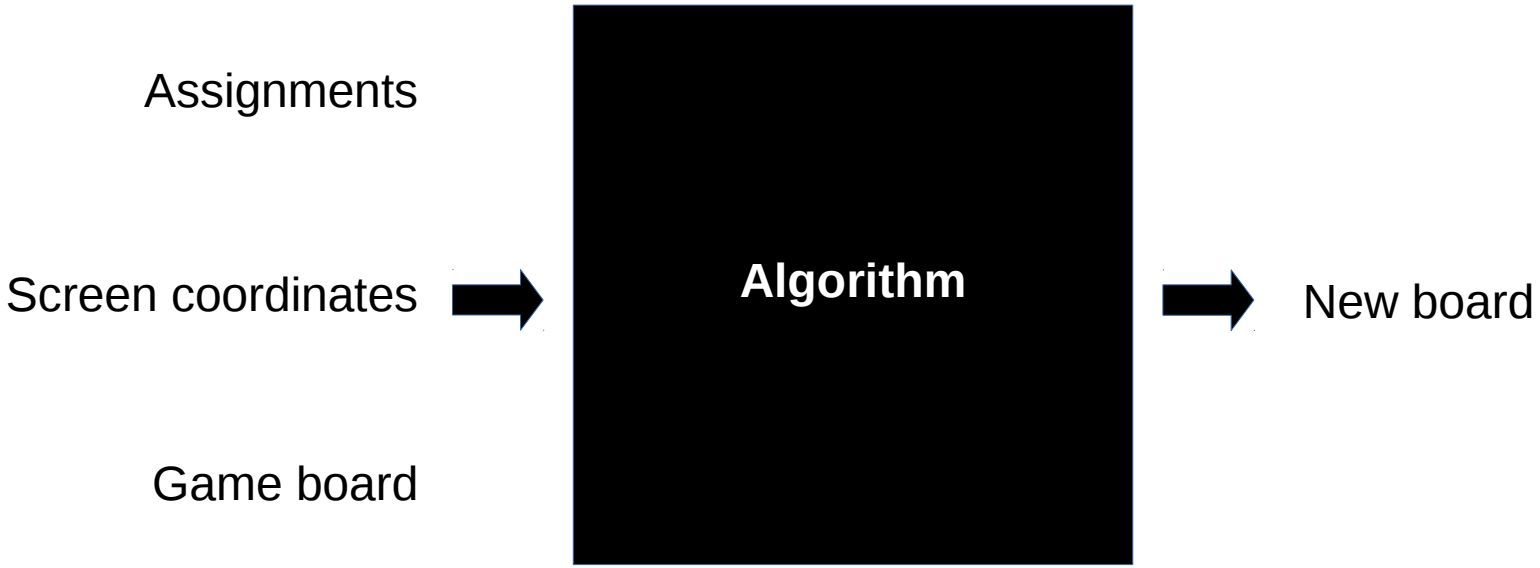


UNIVERSITY OF CALGARY

Fragment Shader

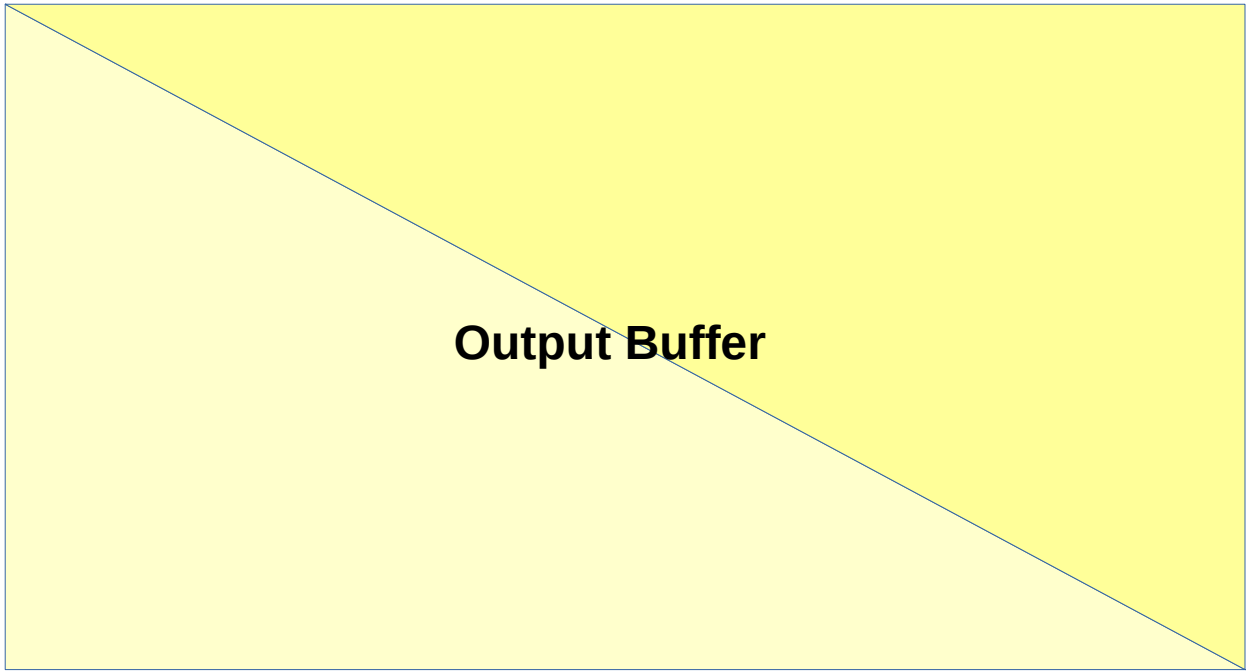


Game of Life


Assignments
Screen coordinates
Game board
Algorithm
New board

UNIVERSITY OF CALGARY

Hacking the Fragment Shader



UNIVERSITY OF CALGARY

Hacking the Fragment Shader

```
in vec2 coordinate;
out vec4 colour;

void main() {

    // do things with coordinates

    colour = something;
    }
```

UNIVERSITY OF CALGARY

Hacking the Fragment Shader

```
in vec2 coordinate;      // [-1:1]
out vec4 colour;

void main() {

    vec2 address = coordinate + /* pixel offset */

    // do things with coordinates

    colour = something;
    }
```

UNIVERSITY OF CALGARY

Hacking the Fragment Shader

```
out vec4 colour;

void main() {

    vec2 coordinates = gl_FragCoord.xy; // pixels

    // do things with coordinates

    colour = something;
    }
```

UNIVERSITY OF CALGARY

Hacking the Fragment Shader

```
layout(binding=0) uniform sampler2D source;   // old board
out vec4 colour;

void main() {

     vec2 address = gl_FragCoord.xy;

     int sum = texture2D( source, address );

     // grab more old values

     if ( /* condition */ )
          colour = something;

     }
```

Hacking the Fragment Shader

```
layout(binding=0) uniform sampler2D source;
out vec4 colour;

void main() {

     vec2 address = /* pixel coordinates */;

     int sum = texture2D( source, /* unit square */ );

     // grab more old values

     if ( /* condition */ )
          colour = something;

     }
```

UNIVERSITY OF CALGARY

Hacking the Fragment Shader

```
layout(binding=0) uniform sampler2D source;
out vec4 colour;
uniform vec2 scalar;        // vec2( 1/width, 1/height );

void main() {

     vec2 address = gl_FragCoord.xy * scalar;

     int sum = texture2D( source, address );

     // grab more old values

     if ( /* condition */ )
          colour = something;

     }
```

UNIVERSITY OF CALGARY

Hacking the Fragment Shader

```
layout(binding=0) uniform sampler2D source;
out vec4 colour;
uniform vec2 scalar;

void main() {

     vec2 address = gl_FragCoord.xy * scalar;

     int sum = texture2D( source, address );

     address = (gl_FragCoord.xy + vec2( 1.0, 1.0 )) * scalar;

     sum += texture2D( source, address );
     // etc...

     if ( /* condition */ )
          colour = something;

     }
```

UNIVERSITY OF CALGARY

Hacking the Fragment Shader

```
layout(binding=0) uniform sampler2D source;
out vec4 colour;
uniform vec2 scalar;

void main() {

     vec2 address = gl_FragCoord.xy * scalar;

     /* int */ = /* vec4 */

     address = (gl_FragCoord.xy + vec2( 1.0, 1.0 )) * scalar;

     /* int */ += /* vec4 */
     // etc...

     if ( /* condition */ )
          colour = something;

     }
```

UNIVERSITY OF CALGARY

Hacking the Fragment Shader

```
layout(binding=0) uniform sampler2D source;
out vec4 colour;
uniform vec2 scalar;

int grab( vec2 offset ) {

      vec2 address = (gl_FragCoord.xy + offset) * scalar;
      vec4 value = texture2D( source, address );
      if ( value.r > 0.5 )
            return 1;
      else
            return 0;
      }

void main() {

      int sum = grab( vec2( -1.0, 1.0 ) );
      sum += grab( vec2( 0.0, 1.0 ) );
      sum += grab( vec2( 1.0, 1.0 ) );
      // etc...

      if ( /* condition */ )
            colour = something;

      }
```

UNIVERSITY OF CALGARY

Hacking the Fragment Shader

```
layout(binding=0) uniform sampler2D source;
out vec4 colour;              // we just want one channel
uniform vec2 scalar;

int grab( vec2 offset ) {

      vec2 address = (gl_FragCoord.xy + offset) * scalar;
      vec4 value = texture2D( source, address );     // we just want one channel
      if ( value.r > 0.5 )
            return 1;
      else
            return 0;
      }

void main() {

      int sum = grab( vec2( -1.0, 1.0 ) );
      sum += grab( vec2( 0.0, 1.0 ) );
      sum += grab( vec2( 1.0, 1.0 ) );
      // etc...

      if ( /* condition */ )
            colour = something;

      }
```

Hacking the Fragment Shader

```
layout(binding=0) uniform sampler2D source;
out float colour;             // expand if output is RGBA
uniform vec2 scalar;

int grab( vec2 offset ) {

      vec2 address = (gl_FragCoord.xy + offset) * scalar;
      float value = texture2D( source, address ).r;  // don't bother expanding out
      if ( value > 0.5 )
            return 1;
      else
            return 0;
      }

void main() {

      int sum = grab( vec2( -1.0, 1.0 ) );
      sum += grab( vec2( 0.0, 1.0 ) );
      sum += grab( vec2( 1.0, 1.0 ) );
      // etc...

      if ( /* condition */ )
            colour = something;

      }
```

UNIVERSITY OF CALGARY

Hacking the Fragment Shader

```
layout(binding=0) uniform sampler2D source;
out float colour;
uniform vec2 scalar;

uniform float dead = 0.0;          // can be changed
uniform float reborn = 1.0;
uniform float old = 0.51;

int grab( vec2 offset ) { /* etc. */ }

void main() {

      int centre = grab( vec2( 0.0, 0.0 ) );

      int sum = grab( vec2( -1.0, 1.0 ) );
      sum += grab( vec2( 0.0, 1.0 ) );
      sum += grab( vec2( 1.0, 1.0 ) );
      // etc...

      if ( /* condition */ )
            colour = dead;
      }
```

UNIVERSITY OF CALGARY

Hacking the Fragment Shader

```
/* etc. */

int grab( vec2 offset ) { /* etc. */ }

void main() {

      int centre = grab( vec2( 0.0, 0.0 ) );

      int sum = grab( vec2( -1.0, 1.0 ) );
      sum += grab( vec2( 0.0, 1.0 ) );
      sum += grab( vec2( 1.0, 1.0 ) );
      // etc...

      if ( (sum > 3) || (sum < 2) )
            colour = dead;

      else if ( centre == 1 )
            colour = old;

      else if ( sum == 3 )
            colour = reborn;

      else
            colour = dead;

      }
```

UNIVERSITY OF CALGARY

Hacking the Fragment Shader

```
/* etc. */

int grab( vec2 offset ) { /* etc. */ }

void main() {

      int centre = grab( vec2( 0.0, 0.0 ) );

      int sum = grab( vec2( -1.0, 1.0 ) );
      sum += grab( vec2( 0.0, 1.0 ) );
      sum += grab( vec2( 1.0, 1.0 ) );
      // etc...

      if ( (sum > 3) || (sum < 2) )            // if statements?!?!?!
            colour = dead;

      else if ( centre == 1 )
            colour = old;

      else if ( sum == 3 )
            colour = reborn;

      else
            colour = dead;

      }
```

UNIVERSITY OF CALGARY

Hacking the Fragment Shader

```
/* etc. */

int grab( vec2 offset ) { /* etc. */ }

void main() {

      int centre = grab( vec2( 0.0, 0.0 ) );    // tonnes of memory access

      int sum = grab( vec2( -1.0, 1.0 ) );
      sum += grab( vec2( 0.0, 1.0 ) );
      sum += grab( vec2( 1.0, 1.0 ) );
      // etc...

      if ( (sum > 3) || (sum < 2) )
            colour = dead;                      // short code paths

      else if ( centre == 1 )
            colour = old;

      else if ( sum == 3 )
            colour = reborn;

      else
            colour = dead;

      } // function ends
```

UNIVERSITY OF CALGARY

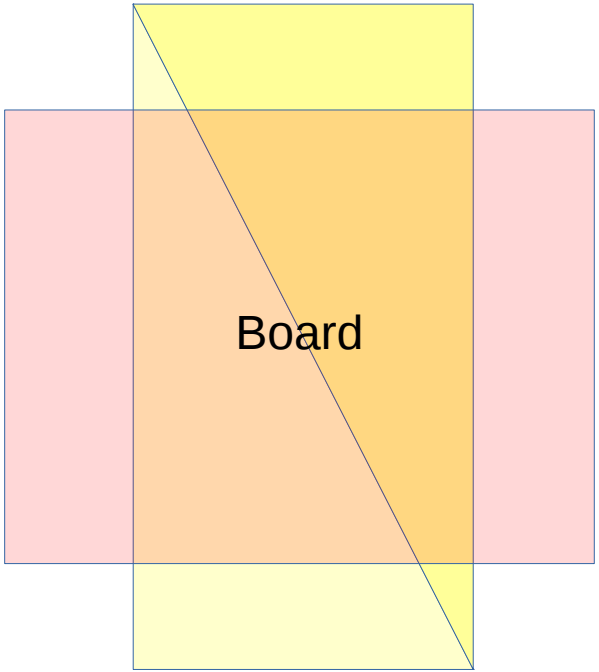
Hacking the Fragment Shader

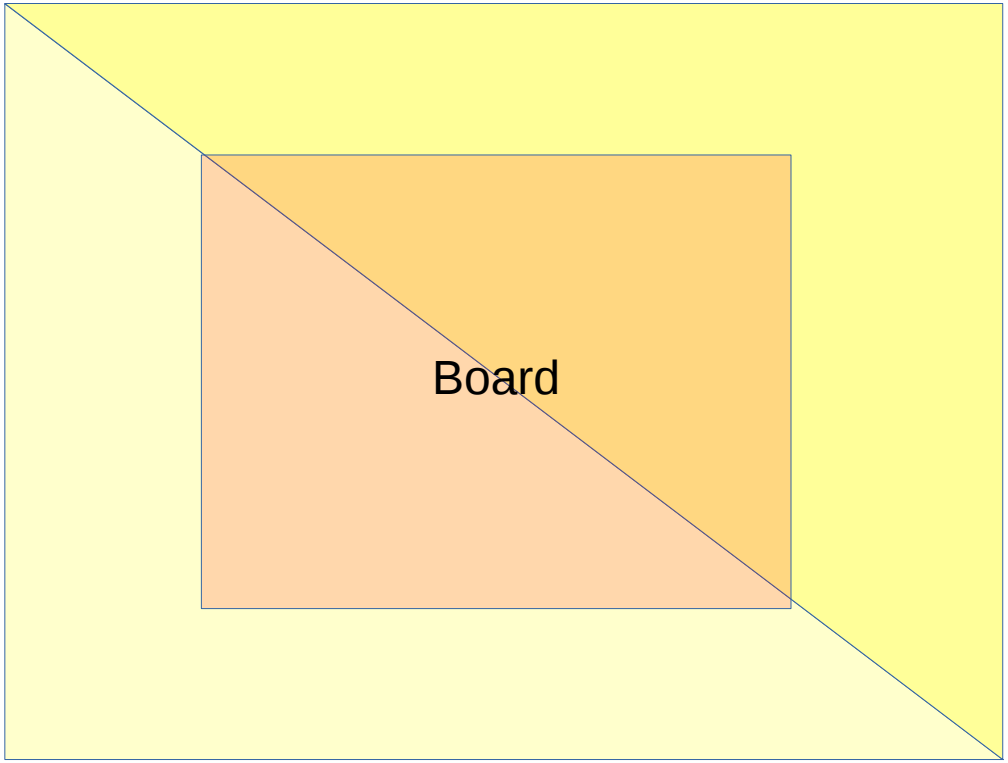
```
/* etc. */

int grab( vec2 offset ) { /* etc. */ }

void main() {

      int centre = grab( vec2( 0.0, 0.0 ) );

      int sum = grab( vec2( -1.0, 1.0 ) );
      sum += grab( vec2( 0.0, 1.0 ) );
      sum += grab( vec2( 1.0, 1.0 ) );

      sum += grab( vec2( -1.0, 0.0 ) );
      sum += grab( vec2( 1.0, 0.0 ) );

      if ( sum > 3 ) {                          // save precious bandwidth

            colour = dead;
            return;
            }

      sum += grab( vec2( -1.0, -1.0 ) );
      sum += grab( vec2( 0.0, -1.0 ) );
      sum += grab( vec2( 1.0, -1.0 ) );

      // etc...
```

Drawing the Board

```
layout(binding=0) uniform sampler2D source;
in vec2 coordinate;    // [-1:1]
out vec4 colour;

void main() {

    colour = texture2D( source, coordinate*.5 + .5 );

    }
```

Drawing the Board





Pixels must be 1:1

Drawing the Board

```
layout(binding=0) uniform sampler2D source;
out vec4 colour;
uniform vec2 scalar;      // figure this out CPU-side
uniform vec2 offset;

void main() {

    vec2 address = (gl_FragCoord.xy * scalar) + offset; // MAD
    colour = texture2D( source, address );

    }
```

UNIVERSITY OF CALGARY

Drawing the Board

```
layout(binding=0) uniform sampler2D source;
out vec4 colour;
uniform vec2 scalar;
uniform vec2 offset;

void main() {

    vec2 address = (gl_FragCoord.xy * scalar) + offset;
    colour = texture2D( source, address );    // float -> vec4

    }
```

Drawing the Board

```
layout(binding=0) uniform sampler2D source;
out vec4 colour;
uniform vec2 scalar;
uniform vec2 offset;

uniform vec4 alive = vec4( 1.0, 1.0, 0.9, 1.0 );  // flexibility
uniform vec4 dead = vec4( 0.0, 0.0, 0.1, 1.0 );

void main() {

     vec2 address = (gl_FragCoord.xy * scalar) + offset;
     float value = texture2D( source, address ).r;

     colour = mix( dead, alive, value );

     }
```

UNIVERSITY OF CALGARY

Drawing the Board

```
layout(binding=0) uniform sampler2D source;
out vec4 colour;
uniform vec2 scalar;
uniform vec2 offset;

uniform vec4 alive = vec4( 1.0, 1.0, 0.9, 1.0 );
uniform vec4 dead = vec4( 0.0, 0.0, 0.1, 1.0 );

void main() {

    vec2 address = (gl_FragCoord.xy * scalar) + offset;
    float value = texture2D( source, address ).r;     // OOB = OK!

    colour = mix( dead, alive, value );

    }
```

UNIVERSITY OF CALGARY

How to load Textures?

Render to Texture?

UNIVERSITY OF CALGARY

How to load Textures?

~~Render to Texture?~~

Texture Types

```
class SimpleTexture {

      // ...

        private:
        GLuint id = 0;          // named texture
        uint width;             // dimensions
        uint height;

                                // how to handle sampling
        GLenum upsampling       = GL_LINEAR;
        GLenum downsampling     = GL_LINEAR_MIPMAP_LINEAR;
        GLenum wrapping         = GL_REPEAT;
        GLuint sampler = 0;     // the named reference

        GLenum format;          // pixel format
                                // what type of texture? 1D, 2D, RECTANGLE, ...
        GLenum type = GL_TEXTURE_2D;

      // ...
```

UNIVERSITY OF CALGARY

Texture Types

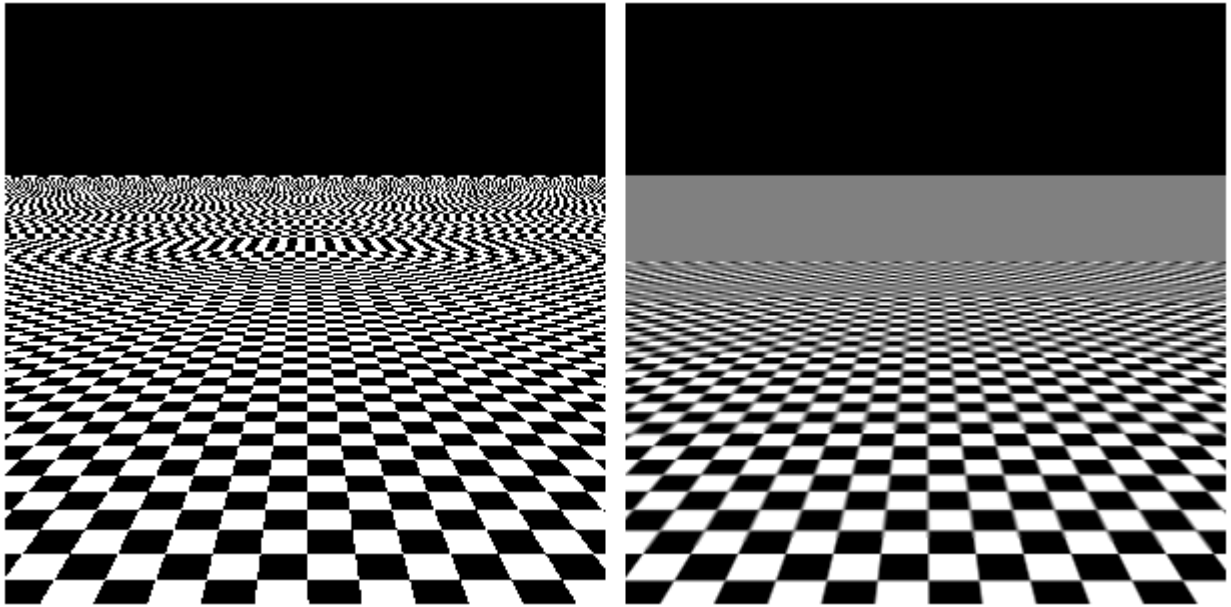
There are a number of different types of textures. These are:

- `GL_TEXTURE_1D`: Images in this texture all are 1-dimensional. They have width, but no height or depth.
- `GL_TEXTURE_2D`: Images in this texture all are 2-dimensional. They have width and height, but no depth.
- `GL_TEXTURE_3D`: Images in this texture all are 3-dimensional. They have width, height, and depth.
- `GL_TEXTURE_RECTANGLE`: The image in this texture (only one image. No mipmapping) is 2-dimensional. Texture coordinates used for these textures are not normalized.
- `GL_TEXTURE_BUFFER`: The image in this texture (only one image. No mipmapping) is 1-dimensional. The storage for this data comes from a Buffer Object.
- `GL_TEXTURE_CUBE_MAP`: There are exactly 6 distinct sets of 2D images, all of the same size. They act as 6 faces of a cube.
- `GL_TEXTURE_1D_ARRAY`: Images in this texture all are 1-dimensional. However, it contains multiple sets of 1-dimensional images, all within one texture. The array length is part of the texture's size.
- `GL_TEXTURE_2D_ARRAY`: Images in this texture all are 2-dimensional. However, it contains multiple sets of 2-dimensional images, all within one texture. The array length is part of the texture's size.
- `GL_TEXTURE_CUBE_MAP_ARRAY`: Images in this texture are all cube maps. It contains multiple sets of cube maps, all within one texture. The array length * 6 (number of cube faces) is part of the texture size.
- `GL_TEXTURE_2D_MULTISAMPLE`: The image in this texture (only one image. No mipmapping) is 2-dimensional. Each pixel in these images contains multiple samples instead of just one value.
- `GL_TEXTURE_2D_MULTISAMPLE_ARRAY`: Combines 2D array and 2D multisample types. No mipmapping.

https://www.khronos.org/opengl/wiki/Texture

UNIVERSITY OF CALGARY

Aliasing / Moire Patterns



https://textureingraphics.wordpress.com/what-is-texture-mapping/anti-aliasing-problem-and-mipmapping/

Mipmaps

http://www.tomshardware.com/reviews/ati,819-2.html

UNIVERSITY OF CALGARY

Mipmaps

http://www.tomshardware.com/reviews/ati,819-2.html

Creating Textures

```
SimpleTexture::SimpleTexture( uint w, uint h, GLenum f ) {

        glGenTextures( 1, &id );        // generate the texture and sampler
        glGenSamplers( 1, &sampler );

// ...

        format = f;

        switch (f) {                    // how many channels do we have?

                case GL_R8:
                case GL_R16:
                case GL_R16F:
                case GL_R32F:
                        perPixelChan = 1;
                        break;

                case GL_RGB8:
                        perPixelChan = 3;
                        break;

                case GL_RGBA8:
                case GL_RGBA16F:
                case GL_RGBA32F:
                        perPixelChan = 4;
                        break;

// ...
```

UNIVERSITY OF CALGARY

Sampler Preparation

```
// handle sampler settings
bool SimpleTexture::setDownsampler( GLenum value ) {

        // check for validity
        switch (value) {

                case GL_NEAREST:
                case GL_LINEAR:
                case GL_NEAREST_MIPMAP_NEAREST:
                case GL_LINEAR_MIPMAP_NEAREST:
                case GL_NEAREST_MIPMAP_LINEAR:
                case GL_LINEAR_MIPMAP_LINEAR:

                        downsampling = value;
// ...
        }

bool SimpleTexture::setWrapping( GLenum value ) {

        // check for validity
        switch (value) {

                case GL_CLAMP_TO_BORDER:
                case GL_CLAMP_TO_EDGE:
                case GL_MIRRORED_REPEAT:
                case GL_MIRROR_CLAMP_TO_EDGE:
                case GL_REPEAT:

                        wrapping = value;
// ...
        }
```

Allocating Blank Textures

```
// allocate blank storage, if possible
bool SimpleTexture::load() {

       // ...

         glBindTexture( type, id );
         if ( OpenGL::error( "glBindTexture" ) )
                 return false;

         glTexStorage2D( type, 1, format, width, height );
         if ( OpenGL::error( "glTexStorage2D" ) )
                 return false;

         glBindTexture( type, 0 );                        // unbind and mark as ready to go

       // ...
         }
```

Allocating Existing Textures

```
// stuff some values into the texture
bool SimpleTexture::load( vector<float> data ) {

        // ...

         // determine the proper format
         GLenum components;
         switch (perPixelChan) {

                 case 1:
                         components = GL_RED;
                         break;
                 case 3:
                         components = GL_RGB;
                         break;
                 case 4:
                         components = GL_RGBA;
                         break;
              // ...
                 }

         glBindTexture( type, id );
         if ( OpenGL::error( "glBindTexture" ) )
                 return false;

         glTexImage2D( type, 0, format, width, height, 0, components, GL_FLOAT,
                         data.data() );
         if ( OpenGL::error( "glTexImage2D" ) )
                 return false;

         glBindTexture( type, 0 );                       // unbind and mark as ready to go
        // ...
        }
```

Binding the Texture to the Shader

```
// associate a texture with an input
bool ShaderProgram::setTexture( string variable, shared_ptr<SimpleTexture> tex ) {

        // ...

        // grab the texture's location, if possible
        GLint location = glGetUniformLocation( id, variable.c_str() );
        if ( location < 0 )
                return false;

        // textures require a bit more work
        glUniform1i(            location, textureSlot );

        glActiveTexture(        GL_TEXTURE0 + textureSlot );
        if ( OpenGL::error( "glActiveTexture" ) )
                return false;

        glBindTexture(          tex->type, tex->id );
        if ( OpenGL::error( "glBindTexture" ) )
                return false;

        if ( tex->loadSampler( textureSlot ) ) {

                textureSlot++;          // only increment on success
                return true;
                }
        else
                return false;

        }
```

Setting Sampler Settings

```
// load the sampler settings
bool SimpleTexture::loadSampler( GLuint unit ) {

        // bind the sampler first
        glBindSampler( unit, sampler );
        if ( OpenGL::error( "glBindSampler" ) )
                return false;

        // set up the parameters
        glSamplerParameteri( sampler, GL_TEXTURE_MIN_FILTER, downsampling );
        glSamplerParameteri( sampler, GL_TEXTURE_MAG_FILTER, upsampling );
        glSamplerParameteri( sampler, GL_TEXTURE_WRAP_S, wrapping );
        glSamplerParameteri( sampler, GL_TEXTURE_WRAP_T, wrapping );
        glSamplerParameteri( sampler, GL_TEXTURE_WRAP_R, wrapping );

        // cheat a bit and only error check here
        return !OpenGL::error( "glSamplerParameter" );

        }
```

Deleting Textures

```
// just trash the texture
SimpleTexture::~SimpleTexture() {       glDeleteTextures( 1 , &id ); }
```

Summary

Preparation

- Generate name for texture
- Generate name for sampler (optional)
- Allocate storage for texture

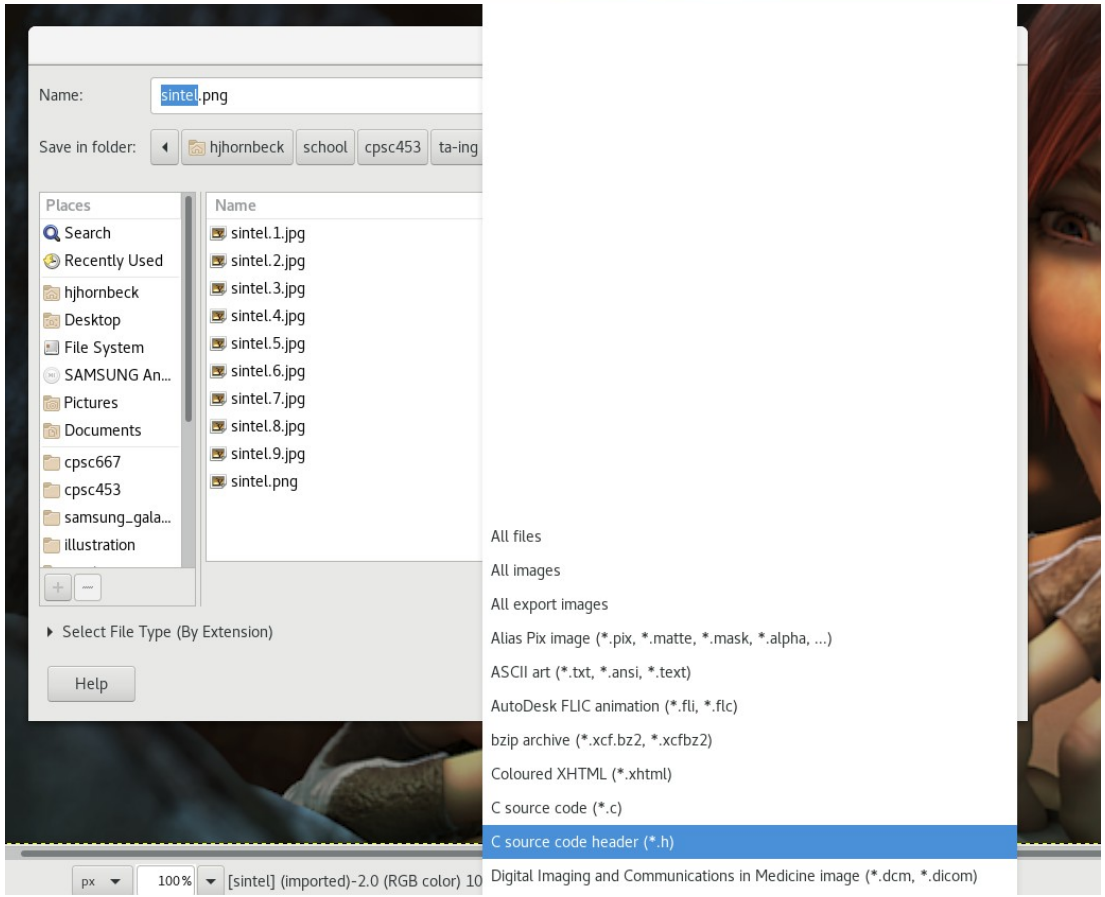
Rendering

- Attach the texture to a shader texture slot
- Bind the texture, bind the sampler, configure it (optional)
- Draw

Allocating Existing Textures

```
// stuff some values into the texture
bool SimpleTexture::load( vector<float> data ) {

        // ...

        // determine the proper format
        GLenum components;
        switch (perPixelChan) {

                case 1:
                        components = GL_RED;
                        break;
                case 3:
                        components = GL_RGB;
                        break;
                case 4:
                        components = GL_RGBA;
                        break;
             // ...
                }

        glBindTexture( type, id );
        if ( OpenGL::error( "glBindTexture" ) )
                return false;

        glTexImage2D( type, 0, format, width, height, 0, components, GL_FLOAT,
                        data.data() );
        if ( OpenGL::error( "glTexImage2D" ) )
                return false;

        glBindTexture( type, 0 );                       // unbind and mark as ready to go
        // ...
        }
```

GIMP


Name:
sintel.png
Save in folder:
hjhornbeck
school
cpsc453
ta-ing
Places
Search
Recently Used
hjhornbeck
Desktop
File System
SAMSUNG An...
Pictures
Documents
cpsc667
cpsc453
samsung_gala...
illustration
Name
sintel.1.jpg
sintel.2.jpg
sintel.3.jpg
sintel.4.jpg
sintel.5.jpg
sintel.6.jpg
sintel.7.jpg
sintel.8.jpg
sintel.9.jpg
sintel.png
Select File Type (By Extension)
Help
All files
All images
All export images
Alias Pix image (*.pix, *.matte, *.mask, *.alpha, ...)
ASCII art (*.txt, *.ansi, *.text)
AutoDesk FLIC animation (*.fli, *.flc)
bzip archive (*.xcf.bz2, *.xcfbz2)
Coloured XHTML (*.xhtml)
C source code (*.c)
C source code header (*.h)
Digital Imaging and Communications in Medicine image (*.dcm, *.dicom)
px
100%
[sintel] (imported)-2.0 (RGB color) 10

UNIVERSITY OF CALGARY

Demo 4

Difference

STB Library

```
// in a C/C++ source file!

#define STB_IMAGE_IMPLEMENTATION
#include "stb_image.h"


void Difference::loadImage( const char* file, uint index ) {

        // ...

        // call STB
        int width, height, channels;

        float* pixels = stbi_loadf(file, &width, &height, &channels, 0);
        // unsigned char* out = stbi_load( /* ... */ );

        // ...

        // free up the buffer
        stbi_image_free( pixels );
```

https://github.com/nothings/stb/blob/master/stb_image.h