Image Processing
Frame Buffer Objects

3-12-15
Outline

- Image Processing: Examples
- Render to Texture

Read:
- Angel, Chapter 7, 7.10 - 7.13

Lab3  new due date: Friday, Mar. 13th
Project#1 new due date: Friday, Mar. 27th
Lab4  due: Tuesday, Mar. 31st
At init time:
Get a context (gl) from the canvas element.
for each shader
    create shader
    look up attribute and uniform locations
for each shape
    initialize buffers with the shape
for each texture
    create textures and/or fill them with data.

At draw time:
for each shape
    if the last shader used is different than the shader needed for this shape call gl.useProgram
    for each attribute needed by shader
        call gl.enableVertexAttribArray, gl.bindBuffer and gl.vertexAttribPointer for each attribute
            needed by shape with the attribute locations for the current shader.
    for each uniform needed by shader
        call gl.uniformXXX with the desired values using the locations for the current shader
    call gl.drawArrays or if the data is indexed called gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER,
        bufferOfIndicesForCurrentShape) followed by gl.drawElements

Common Optimizations
1) Often you don't need to set every uniform. For example if you are drawing 10 shapes
   with the same shader and that shader takes a viewMatrix or cameraMatrix it's likely that
   viewMatrix uniform or cameraMatrix uniform is the same for every shape so just set it once.
2) You can often move the calls to gl.enableVertexAttribArray to initialization time.
Recap: Image Processing

- Use the fragment shader to do image processing
  - Image filtering
  - Pseudo Color
- Use multiple textures
  - Matrix operations
- Introduction to GPGPU
Example: Sobel Filter

- Very popular edge detection filter
- Approximate:
  \[
  \frac{\partial}{\partial x} \approx \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad \frac{\partial}{\partial y} \approx \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}
  \]
- Output is $|\nabla a|$, computed as follows:
  \[
  \nabla a = \begin{bmatrix} \frac{\partial a}{\partial x} & \frac{\partial a}{\partial y} \end{bmatrix}, \quad |\nabla a| = \sqrt{(\frac{\partial a}{\partial x})^2 + (\frac{\partial a}{\partial y})^2}
  \]
- Sobel filter is non-linear
  - Square and square root (more exact computation)
  - Can also use absolute value (faster computation)
Sample Filter Computation

- One part (of the two) of the Sobel filter
- Detects vertical edges
Sobel x operator

To calculate the pixel in the right side of the equation (the one with coordinates 1,1) the following equation is used:

output pixel [1,1] = ([0,0] x -1) + ([0,1] x 0) + ([0,2] x 1) + ([1,0] x -2) + ([1,1] x 0) + ([1,2] x 2) + ([2,0] x -1) + ([2,1] x 0) + ([2,2] x 1)

Recent GPUs and graphics cards support textures up to 8K x 8K.

For scalar imaging, we can do twice as well using all four color components.

\[
\begin{bmatrix}
R & G \\
B & A
\end{bmatrix}
\]
Indexed and Pseudo Color

- Display luminance (2D) image as texture map
- Treat pixel value as independent variable for separate functions for each color component

```cpp
void main()
{
    vec4 color = texture2D(texture, fTexCoord);
    if(color.g<0.5) color.g = 2.0*color.g;
    else color.g = 2.0 - 2.0*color.g;
    color.b = 1.0-color.b;
    gl_FragColor = color;
}
```
Top View of 2D Sinc

hatImage1.html

hatImage2.html
The Next Step

- Need more storage for most GPGPU calculations
- Example: filtering
- Example: iteration
- Need shared memory
- Solution: Use texture memory and off-screen rendering
Examples: WebGL Programming Guide, Chapter 10

Fog.html

↑↓: Increase/decrease the fog distance

use of fog

LookAtBlendedTriangles.html

blending
Examples: WebGL Programming Guide, Chapter 10

HUD.html

RotateObject.html

Head Up Display

Rotate textured object with the mouse
Framebuffer Objects

- Look at methods that use memory on the graphics card
- Introduce off screen rendering
- Learn how to create framebuffer objects
  - Create a renderbuffer
  - Attach resources
Discrete Processing in WebGL

- Recent GPUs contain large amounts of memory
  - Texture memory
  - Framebuffer
  - Floating point
- Fragment shaders support discrete operations at the pixel level
- Separate pixel (texel) pipeline
Accessing the Framebuffer

- Pre 3.1 OpenGL had functions that allowed access to the framebuffer and other OpenGL buffers
  - Draw Pixels
  - Read Pixels
  - Copy Pixels
  - BitBlt
  - Accumulation Buffer functions
- All deprecated
We have already seen that we can write pixels as texels to texture memory.

Texture objects reduce transfers between CPU and GPU.

Transfer of pixel data back to CPU slow.

Want to manipulate pixels without going back to CPU.

Image processing.

GPGPU.
Framebuffer Objects

Framebuffer Objects (FBOs) are buffers that are created by the application. They are not under the control of the window system and cannot be displayed. You can attach a renderbuffer to an FBO and render off-screen into the attached buffer. The attached buffer can then be detached and used as a texture map for an on-screen render to the default frame buffer.
Textures are shared by all instances of the fragment shade.

If we render to a texture attachment we can create a new texture image that can be used in subsequent renderings.

Use a double buffering strategy for operations such as convolution.
Steps

- Create an Empty Texture Object
- Create a FBO
- Attach renderbuffer for texture image
- Bind FBO
- Render scene
- Detach renderbuffer
- Bind texture
- Render with new texture
Empty Texture Object

texture1 = gl.createTexture();
gl.activeTexture( gl.TEXTURE0 );
gl.bindTexture( gl.TEXTURE_2D, texture1 );

gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, 512, 512, 0, gl.RGBA, gl.UNSIGNED_BYTE, null);

gl.generateMipmap(gl.TEXTURE_2D);
gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST_MIPMAP_LINEAR );
gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST )
Creating a FBO

- We create a framebuffer object in a similar manner to other objects
- Creating an FBO creates an empty FBO
- Must add needed resources
  - Can add a renderbuffer to render into
  - Can add a texture which can also be rendered into
- For hidden surface removal we must add a depth buffer attachment to the renderbuffer
var framebuffer = gl.createFramebuffer();
gl.bindFramebuffer(gl.FRAMEBUFFER, framebuffer);
framebuffer.width = 512;
framebuffer.height = 512;
// renderbuffer = gl.createRenderbuffer();
// gl.bindRenderbuffer(gl.RENDERBUFFER, renderbuffer);
// gl.renderbufferStorage(gl.RENDERBUFFER,
// gl.DEPTH_COMPONENT16, 512, 512);
// Attach color buffer
gl.framebufferTexture2D(gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0, gl.TEXTURE_2D, texture1, 0);
// gl.framebufferRenderbuffer(gl.FRAMEBUFFER, gl.DEPTH_ATTACHMENT, gl.RENDERBUFFER, renderbuffer);
// check for completeness
var status = gl.checkFramebufferStatus(gl.FRAMEBUFFER);
if(status != gl.FRAMEBUFFER_COMPLETE) alert('Frame Buffer Not Complete');
Rest of Initialization

- Same as previous examples
  - Allocate VAO
  - Fill VAO with data for render to texture
- Initialize two program objects with different shaders
  - First for render to texture
  - Second for rendering with created texture
Examples of render-to-texture

Render a triangle to texture, then use this texture on a rectangle

Introduce buffer pingponging
Program Objects and Shaders

- For most applications of render-to-texture we need multiple program objects and shaders
  - One set for creating a texture
  - Second set for rendering with that texture
- Applications that we consider later such as buffer pingponging may require additional program objects
Program Object 1 Shaders

pass through vertex shader:

```cpp
attribute vec4 vPosition;
void main()
{
    gl_Position = vPosition;
}
```

fragment shader to get a red triangle:

```cpp
precision mediump float;
void main()
{
    gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
}
```
// vertex shader
attribute vec4 vPosition;
attribute vec2 vTexCoord;
varying vec2 fTexCoord;
void main()
{
  gl_Position = vPosition;
  fTexCoord = vTexCoord;
}

// fragment shader
precision mediump float;

varying vec2 fTexCoord;
uniform sampler2D texture;
void main()
{
  gl_FragColor = texture2D( texture,
                          fTexCoord);
}
First Render (to Texture)

```javascript
gl.useProgram( program1 );
var buffer1 = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, buffer1 );
gl.bufferData( gl.ARRAY_BUFFER, flatten(pointsArray), gl.STATIC_DRAW );

// Initialize the vertex position attribute from the vertex shader
var vPosition = gl.getAttribLocation( program1, "vPosition" );
gl.vertexAttribPointer( vPosition, 2, gl.FLOAT, false, 0, 0 );
gl.enableVertexAttribArray( vPosition );

// Render one triangle
gl.viewport(0, 0, 64, 64);
gl clearColor(0.5, 0.5, 0.5, 1.0);
gl.clear(gl.COLOR_BUFFER_BIT );
gl.drawArrays(gl.TRIANGLES, 0, 3);
```
Set Up Second Render

// Bind to default window system framebuffer

    gl.bindFramebuffer(gl.FRAMEBUFFER, null);
    gl.disableVertexAttribArray(vPosition);
    gl.useProgramProgram(program2);

// Assume we have already set up a texture object with null texture image

    gl.activeTexture(gl.TEXTURE0);
    gl.bindTexture(gl.TEXTURE_2D, texture1);

// set up vertex attribute arrays for texture coordinates and rectangle as usual
Data for Second Render

```javascript
var buffer2 = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, buffer2);
gl.bufferData(gl.ARRAY_BUFFER, new flatten(vertices), gl.STATIC_DRAW);

var vPosition = gl.getAttribLocation( program2, "vPosition" );
gl.vertexAttribPointer( vPosition, 2, gl.FLOAT, false, 0, 0 );
gl.enableVertexAttribArray( vPosition );

var buffer3 = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, buffer3);
gl.bufferData( gl.ARRAY_BUFFER, flatten(texCoord), gl.STATIC_DRAW);

var vTexCoord = gl.getAttribLocation( program2, "vTexCoord" );
gl.vertexAttribPointer( vTexCoord, 2, gl.FLOAT, false, 0, 0 );
gl.enableVertexAttribArray( vTexCoord );
```
Render a Quad with Texture

```glsl
gl.uniform1i( gl.getUniformLocation(program2, "texture"), 0);

gl.viewport(0, 0, 512, 512);
gl clearColor(0.0, 0.0, 1.0, 1.0);
gl.clear( gl.COLOR_BUFFER_BIT);

gl.drawArrays(gl.TRIANGLES, 0, 6);
```
Dynamic 3D Example
Buffer Ping-pong

- Iterative calculations can be accomplished using multiple render buffers
- Original data in texture buffer 1
- Render to texture buffer 2
- Swap buffers and rerender to texture
From Geometry to Pixels

- At end of the geometric pipeline, vertices have been assembled into primitives
- Must clip out primitives that are outside the view frustum
  - Algorithms based on representing primitives by lists of vertices
- Must find which pixels can be affected by each primitive
  - Fragment generation
  - Rasterization or scan conversion