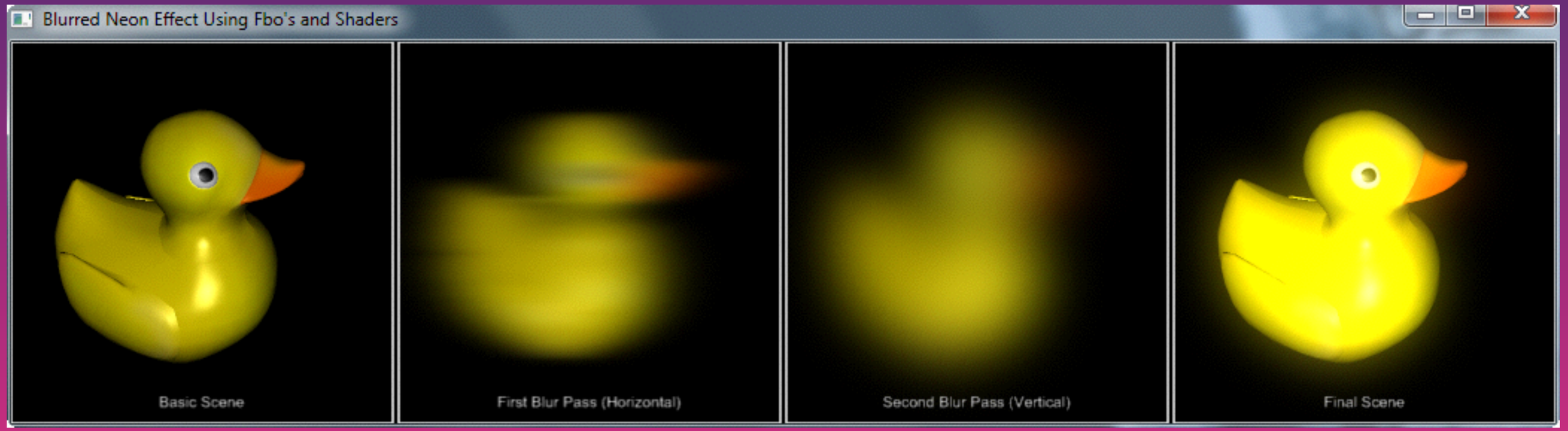
Shaders

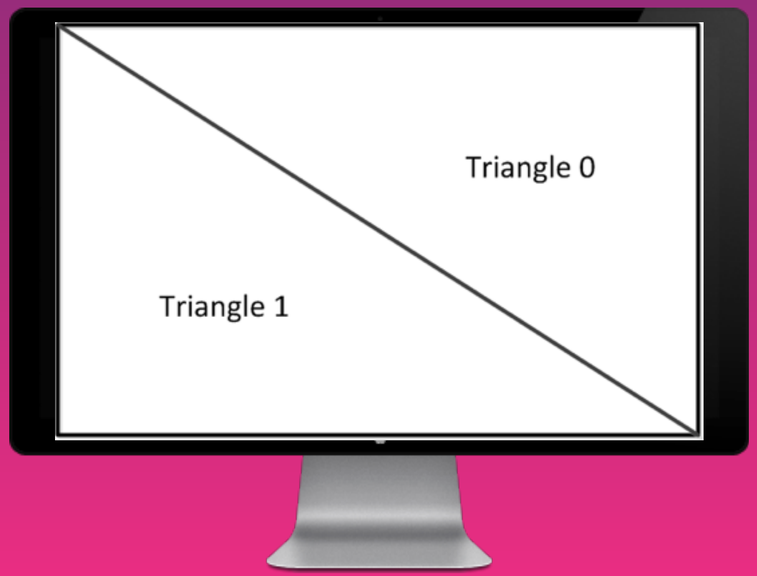
Part 3

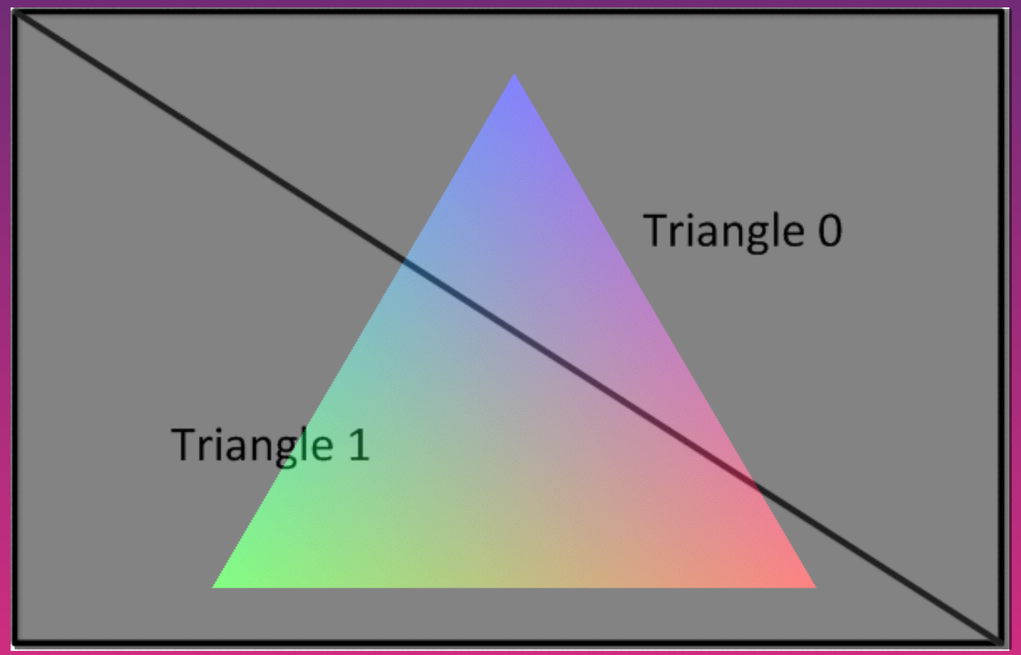
Screen shaders.


Blurred Neon Effect Using Fbo's and Shaders
Basic Scene
First Blur Pass (Horizontal)
Second Blur Pass (Vertical)
Final Scene


HOLD TO EXECUTE BREACH
DART OVERLAY
100 %

Anatomy of a screen shader.


Triangle 0
Triangle 1


Triangle 0
Triangle 1

Rendering to texture.

Frame Buffer Objects (FBOs)

Creating an FBO.

1. Generate and bind a frame buffer .

```
GLuint frameBuffer;
glGenFramebuffers(1, &frameBuffer);
glBindFramebuffer(GL_FRAMEBUFFER, frameBuffer);
```

2. Create an empty texture to render to.

```
glGenTextures(1, &frameBufferTexture);
glBindTexture(GL_TEXTURE_2D, frameBufferTexture);
glTexImage2D(GL_TEXTURE_2D, 0,GL_RGB, 800, 600, 0,GL_RGB,
GL_UNSIGNED_BYTE, NULL);

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
```

3. Bind the texture to the FBO and unbind FBO for later use.

```
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0,
GL_TEXTURE_2D, frameBufferTexture, 0);

glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

Rendering to an FBO.

Bind the FBO before doing your rendering. Calling glClear after you bind it, will clear the texture.

```
glBindFramebuffer(GL_FRAMEBUFFER, frameBuffer);
glClear(GL_COLOR_BUFFER_BIT);
```

Then, do your rendering as usual.

Rendering the FBO texture to screen.

Vertex shader.

```
attribute vec4 position;
attribute vec2 texCoord;
varying vec2 texCoordVar;
void main()
{
    gl_Position =  vec4(position.xy, 0.0, 1.0);
    texCoordVar = texCoord;
}
```

No need for projection or modelview
matrices as we are just drawing two fullscreen triangles.
But we need to make sure that gl_Position is a homogenous coordinate
(vec4 with 1.0 as w).

Fragment shader.

```
uniform sampler2D texture;
varying vec2 texCoordVar;
void main()
{
    gl_FragColor = texture2D( texture, texCoordVar);
}
```

Just draw the texture as is.

After you do your scene rendering, unbind the FBO, use your screen shader program and draw two fullscreen triangles textured with the FBO texture.
You will need to bind the position and tex coordinate attributes of your screen shader program.

```
glBindFramebuffer(GL_FRAMEBUFFER, 0);
glUseProgram(screenShaderProgram);

glEnable(GL_TEXTURE_2D);
glBindTexture(GL_TEXTURE_2D, frameBufferTexture);

GLfloat triUVs[] = {
    1.0f, 1.0f,
    1.0f, 0.0f,
    0.0f, 0.0,

    0.0f, 0.0f,
    0.0f, 1.0f,
    1.0f, 1.0f
};

GLfloat tris[] = {
    1.0f, 1.0f,
    1.0f, -1.0f,
    -1.0f, -1.0f,

    -1.0f, -1.0f,
    -1.0f, 1.0f,
    1.0f, 1.0f
};

glVertexAttribPointer(screenPositionAttribute, 2, GL_FLOAT, false, 0, tris);
glEnableVertexAttribArray(screenPositionAttribute);
glVertexAttribPointer(screenTexcoordAttribute, 2, GL_FLOAT, false, 0, triUVs);
glEnableVertexAttribArray(screenTexcoordAttribute);

glDrawArrays(GL_TRIANGLES, 0, 6);
```

Some basic screen shaders.

Inverting the screen colors.

```
uniform sampler2D texture;
varying vec2 texCoordVar;
void main()
{
    gl_FragColor = vec4(1.0-texture2D( texture, texCoordVar).xyz, 1.0);
}
```

Make everything black and white.

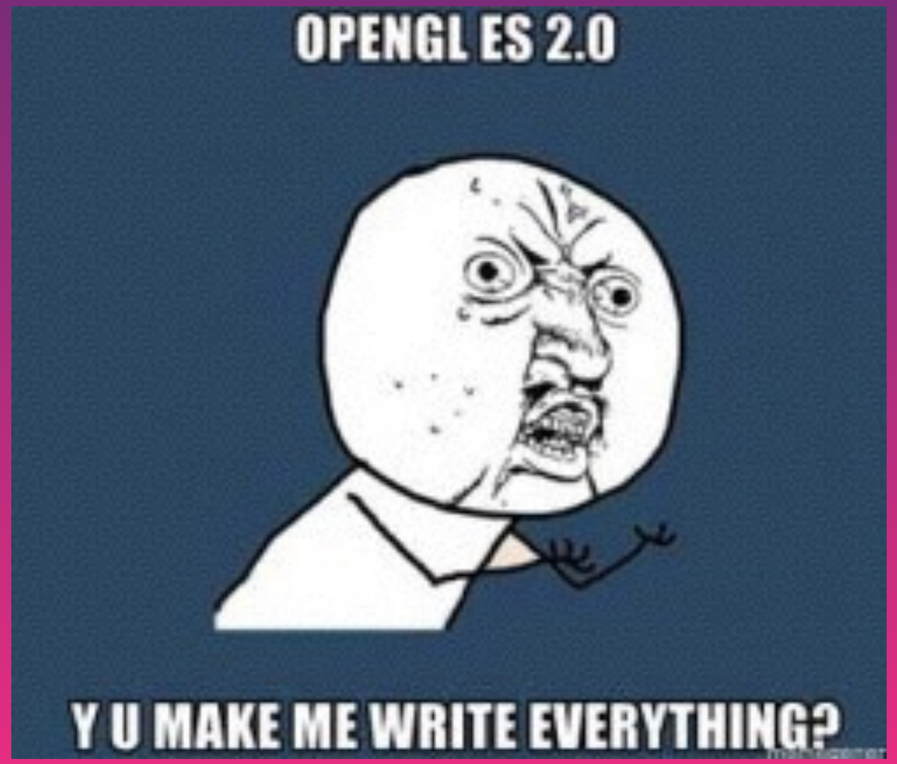
```
uniform sampler2D texture;
varying vec2 texCoordVar;
void main()
{
    vec4 texColor = texture2D( texture, texCoordVar);
    float brightness = (texColor.x+texColor.y+texColor.z)/3.0;
    gl_FragColor = vec4(brightness, brightness, brightness, 1.0);
}
```

Make it wavy!

```
uniform sampler2D texture;
uniform float time;
varying vec2 texCoordVar;
void main()
{
    gl_FragColor = texture2D( texture, vec2(texCoordVar.x+(sin((texCoordVar.y
+time)*15.0) * 0.1), texCoordVar.y));
}
```

You'll need to pass in the time elapsed as the "time" uniform.

OpenGL ES2 / mobile compatibility.



Must use shaders + manual matrix uniforms for projection and modelview!

No built-in matrix operations!

No glOrtho, glMatrixMode, glPushMatrix, glPopMatrix, glTranslate, glRotate, glScale, glLoadIdentity.

Projection matrices

Manual ortho projection matrix.

```
Matrix projectionMatrix;

float l = -1.331f;
float r = 1.33f;
float t = 1.0f;
float b = -1.0f;
float f = 1.0f;
float n = -1.0f;

projectionMatrix.identity();
projectionMatrix.m[0][0] = 2.0/(r-l);
projectionMatrix.m[1][1] = 2.0/(t-b);
projectionMatrix.m[2][2] = -2.0/(f-n);
```

You will need to pass the projection matrix to your entities' shaders as uniforms. If this projection matrix doesn't change, you can bind it once to your shader programs in your main app, otherwise you'll need to pass it to the entities like the camera matrix.

Modelview matrices

Instead of pre-translating everything for the camera view, you need to pass the camera's matrix to your entities.

The modelview matrix you pass to your entities' shaders is their model matrix multiplied by the inverse of the camera matrix.

You must use shaders/vertex attribute pointers for all drawing!

There are no **GL_QUADS** in OpenGL ES2, so you must draw using triangles (or lines or points).

Additional things to note if compiling for mobile.

When compiling for mobile platforms, you'll need to include **SDL_opengles2.h** instead of **SDL_opengl.h**

You also need to set SDL to use the OpenGL ES2 profile (before you create the GL context).

```
SDL_GL_SetAttribute(SDL_GL_CONTEXT_PROFILE_MASK, SDL_GL_CONTEXT_PROFILE_ES);
SDL_GL_SetAttribute(SDL_GL_CONTEXT_MAJOR_VERSION, 2);
SDL_GL_SetAttribute(SDL_GL_CONTEXT_MINOR_VERSION, 0);
```

You should also prefix your GLSL shaders with the following to make sure they work both with OpenGL ES2 and desktop OpenGL.

```
#ifdef GL_ES
precision mediump float;
#else
#define lowp
#define mediump
#define highp
#endif
```

See SDL documentation for the details of setting up an SDL2 project for mobile platforms.

That’s it!