A Gentle Introduction

3 - Fragment Shaders

Image Processing

Carl Bateman
WebGL Workshop

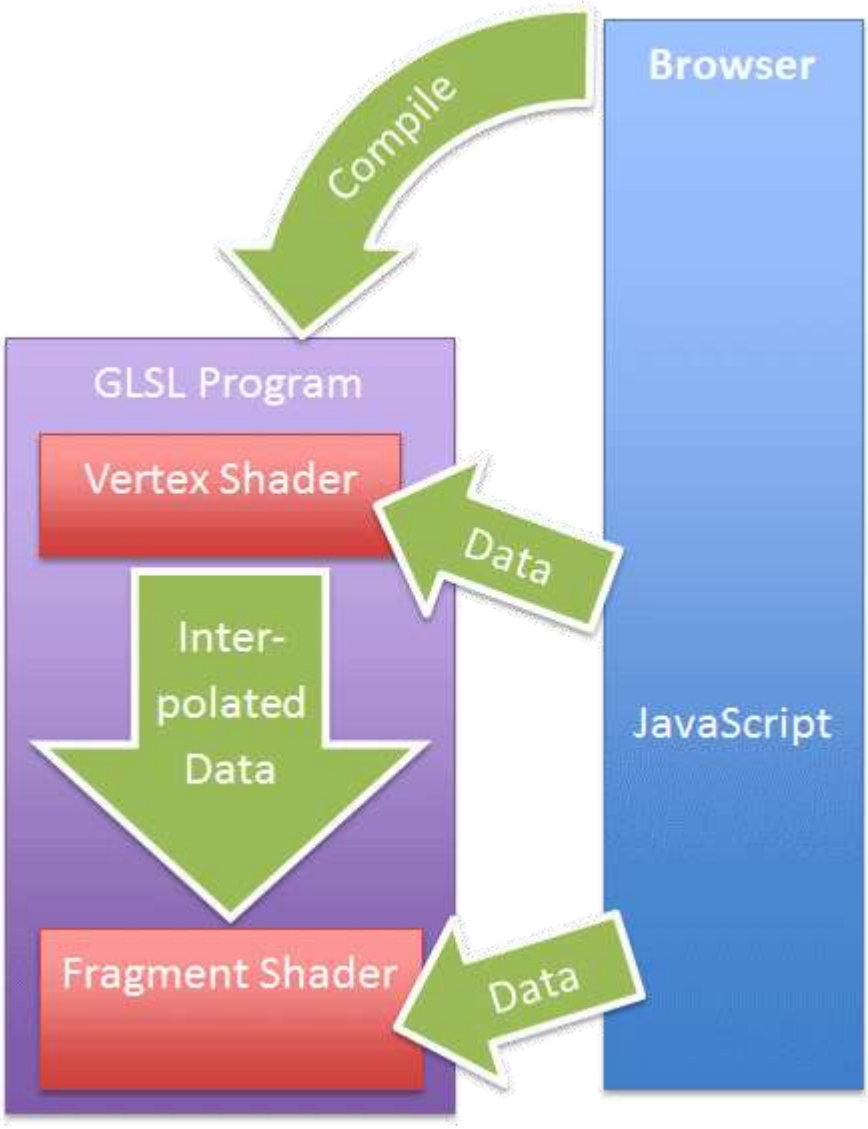
Table of Contents

Part 1. Background

WebGL

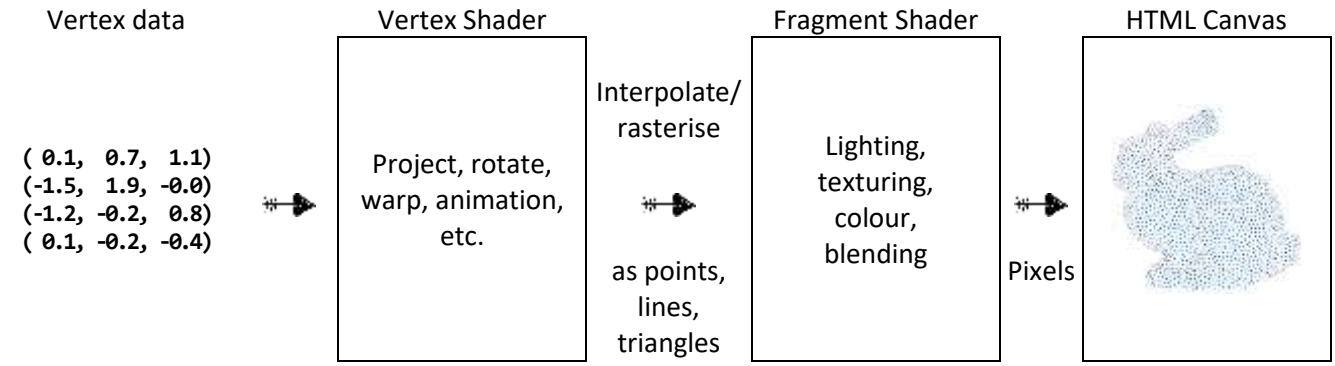
- Canvas based
- JavaScript API
- Renders 2D and 3D graphics
- Giving access to the GPU
- Integrated into all web standards

- Canvas
- Shader program (GLSL)
- Vertex shader
- Fragment shader (pixels)

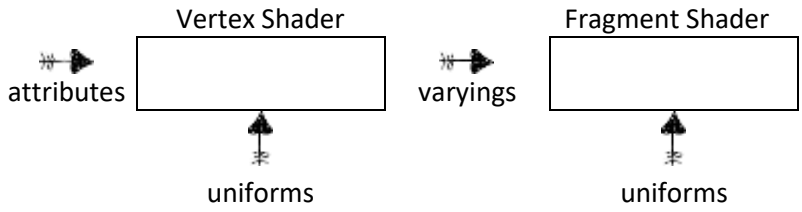


Graphics Pipeline

Simplified!

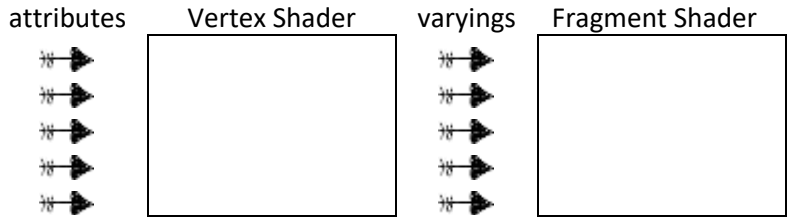


Sharing Data



SIMD

Single **I**nstruction **M**ultiple **D**ata
Shader programs run on the GPU, with the same set of instuctions (the vertex shader) working on each data item at the same time.



Programs can't access adjacent pixels or vertices (there are workrounds).

GLSL

GLSL (GLSLang) is a short term for the official OpenGL Shading Language. **GLSL** is a C/C++ similar high level programming language for several parts of the graphic card. With **GLSL** you can code (right up to) short programs, called shaders, which are executed on the GPU.

Khronos reference sheets

https://www.khronos.org/developers/reference-cards/

OpenGL ES Shading Language Reference

http://www.shaderific.com/GLSL

Support

https://caniuse.com/#search=webgl

WebGL (Web Graphics Library) is a JavaScript API for rendering interactive 3D and 2D graphics within any compatible web browser without the use of plug-ins. WebGL does so by introducing an API that closely conforms to OpenGL ES 2.0 that can be used in HTML5 `<canvas>` elements.

https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API

WebGL programs consist of control code written in JavaScript and shader code that is written in OpenGL ES Shading Language (GLSL ES), a language similar to C or C++, and is executed on a computer's graphics processing unit (GPU).

https://en.wikipedia.org/wiki/WebGL

GLSL WebGL1

Vertex shader

```
attribute vec3 aVertex;
attribute vec3 aColor;
varying vec3 vColor;

void main() {
  vColor = aColor;
  gl_Position = vec4(aVertex, 1.0);
}
```

Fragment shader

```
precision highp float;

varying vec3 vColor;

void main(void) {
  gl_FragColor = vec4(vColor, 1.0);
}
```

GLSL WebGL2

Vertex shader

```
#version 300 es

layout (location=0) in vec4 vertex;
layout (location=1) in vec3 color;

out vec3 vColor;

void main() {
  vColor = color;
  gl_Position = vertex;
}
```

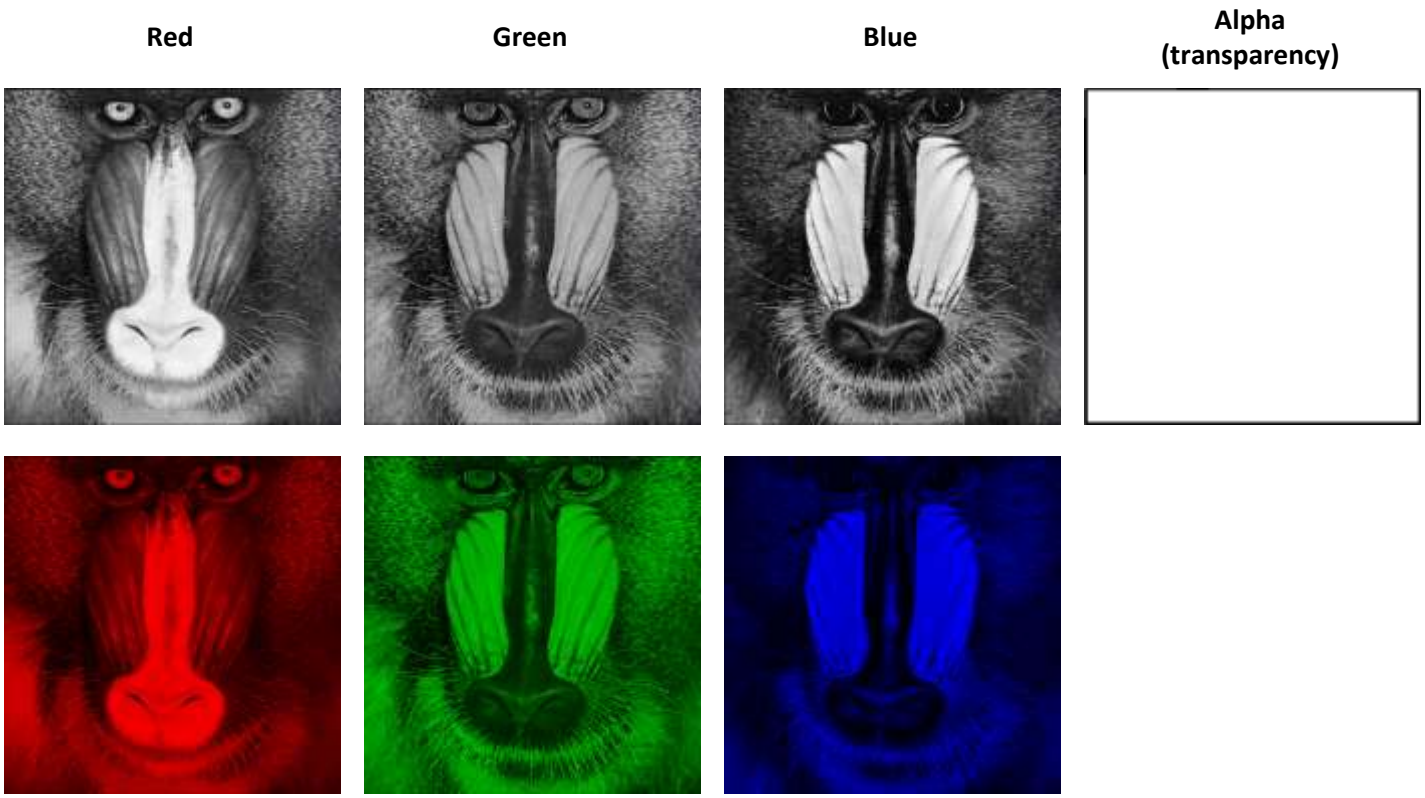
Fragment shader

```
#version 300 es
precision highp float;

in vec3 vColor;
out vec4 fragColor;

void main() {
  fragColor = vec4(vColor, 1.0);
}
```

Textures/Images

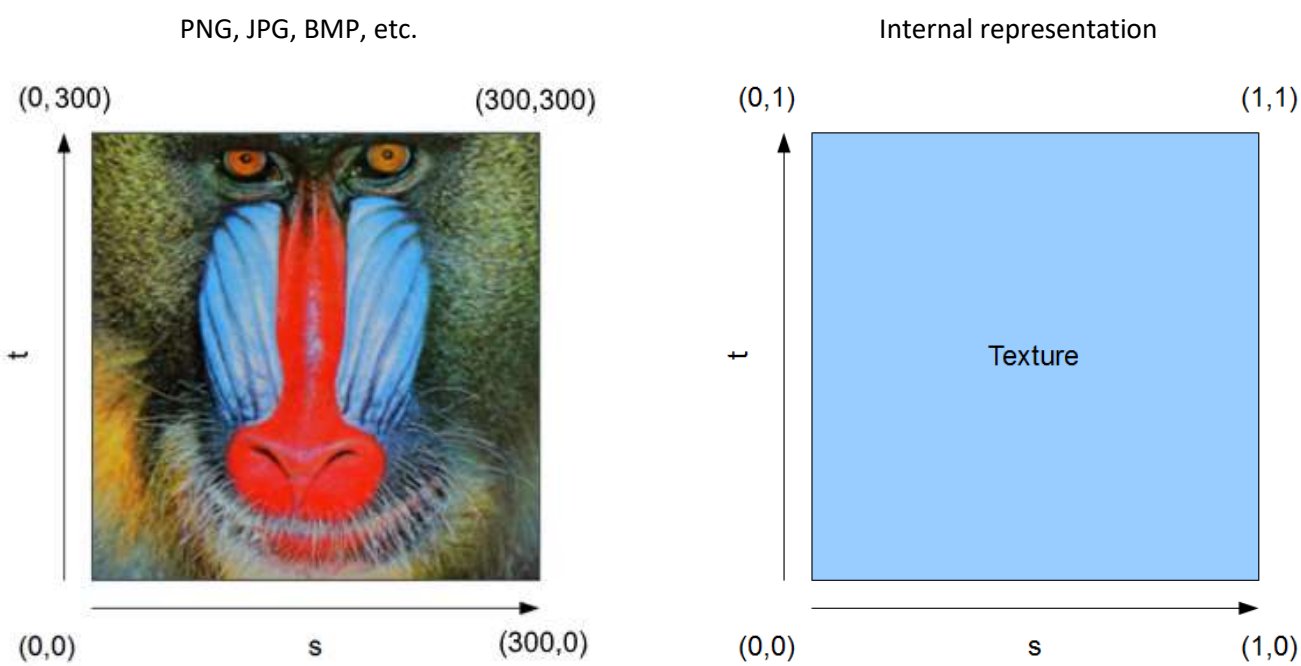
Test images (full colour)

The image is made up of 3 components or channels each specifying the intensity (from 0.0 to 1.0) of each colour (red, green and blue) and contributing to the final image. There is a 4th channel, the alpha, which controls the degree of transparency for the whole image



GLSL gives us control over these channels.

All textures, regardless of their actual dimensions are treated as being of size (1, 1). It is up to the code to map the texture as required.



The simplest shader will map the texture.

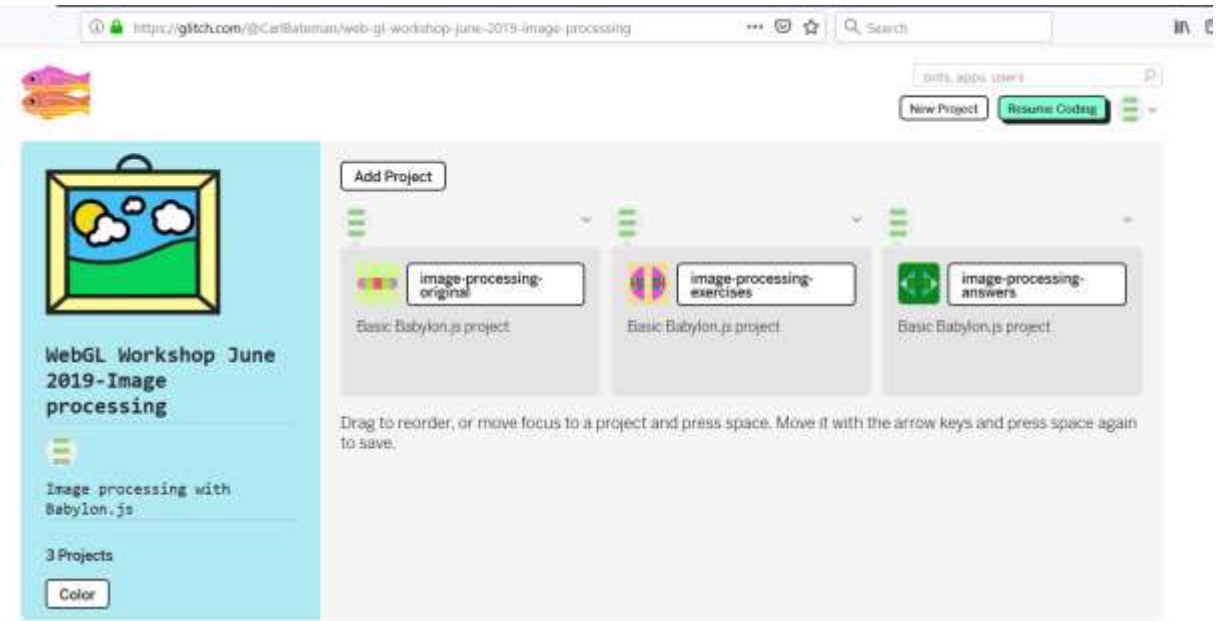
```
gl_FragColor = texture2D(textureSampler1, vUV);
```

The image consists of 300 × 300 pixels, if the shader tries to access a non-existent pixel (very likely) an interpreted value will be returned.

Since the entire texture is available, we can access any part of however we like.

Part 2. Exercises

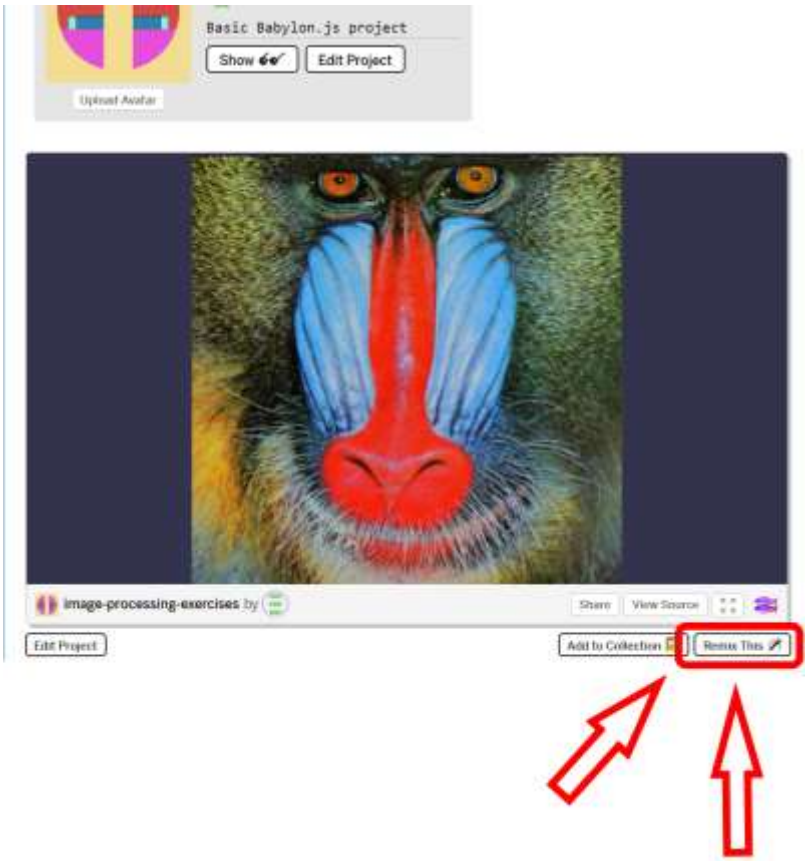
All files can be found at **https://glitch.com/@CarlBateman/web-gl-workshop-june-2019-image-processing**



Try to get the image shown. Each function is given, your mission is complete them.

Use h**ttps://glitch.com/~image-processing-exercises**

Answers are at **https://glitch.com/~image-processing-answers**

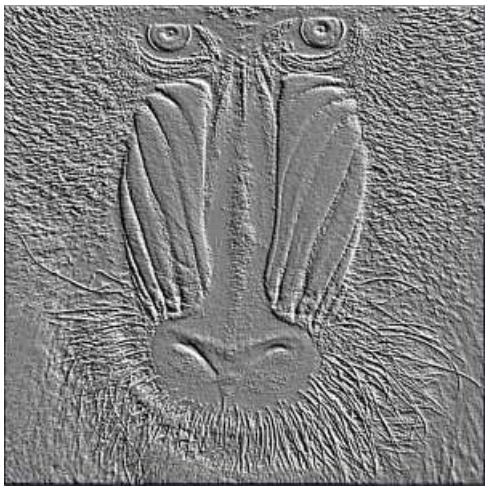


Grey scale

Negative

Red only

Emboss

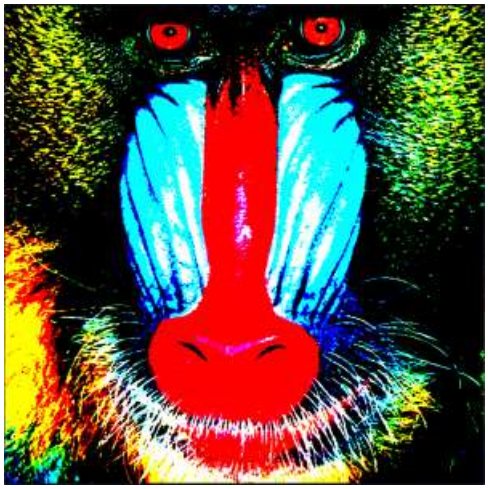


Edge detection

Reflect

Transition (fade)

Max Out



Two-tone (true black and white)

Blur

Part 3. If time allows

Convolution filters
Ping ponging