A Gentle Introduction

2 - Fragment Shaders

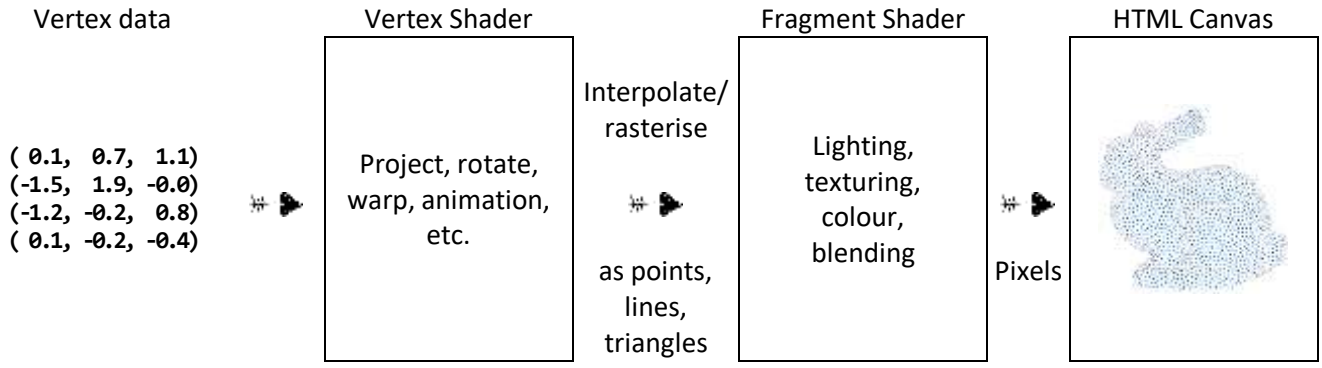
Carl Bateman
WebGL Workshop

Table of Contents

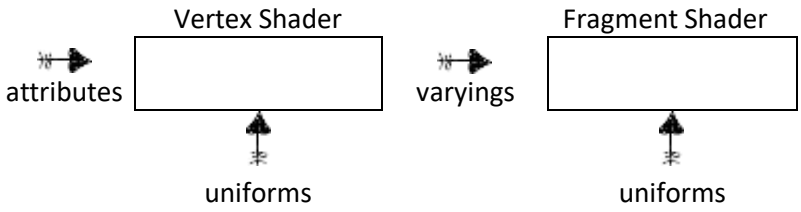
Part 1. Background

Graphics Pipeline

Simplified!

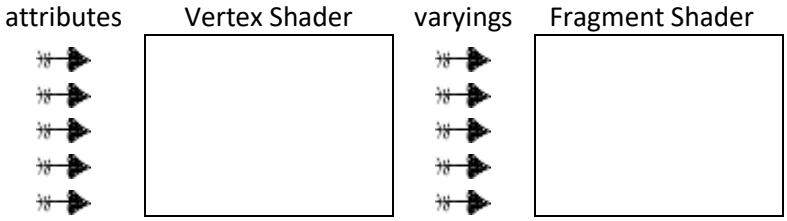


Sharing Data



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

Khronos reference sheets

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

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

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);
}
```

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);
}
```

Part 2. Exercises

Important!!!

1. **Always** when you get a failed but interesting visual effect, screen grab it and copy all your code into a well named repo.

2. Otherwise, accept you will waste days trying unsuccessfully to recreate it, only to go insane from frustration and lack of sleep!!!

3. Share

Book of shaders

https://thebookofshaders.com/

https://thebookofshaders.com/edit.php

http://editor.thebookofshaders.com/

Slow

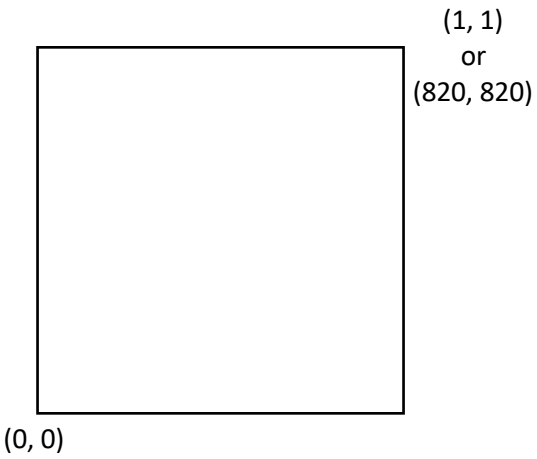
1. Change colour

```
void main() {
  gl_FragColor = vec4(.5, 1.0, .5, 1.0);
}
```


2. Gradient colour

```
void main() {
  gl_FragColor = vec4(gl_FragCoord.x, 1., .5, 1.0);
}
```


Not very gradienty. This is because the canvas (render) area runs from (0, 0) to (1, 1), gl_FragCoord runs to (0, 0) to (820, 820) in screen co-ordinates.



The Book of Shaders editor provides us with several uniform† values:

† uniforms are constant values passed to the shader (GPU) from the controlling program (CPU). These can be used (but not changed).

`uniform vec2 u_resolution;`	The height and width of the render area, as (x, y)
`uniform vec2 u_mouse;`	The x and y position of the mouse over the render
`uniform float u_time;`	A constantly increasing time value (in ms)

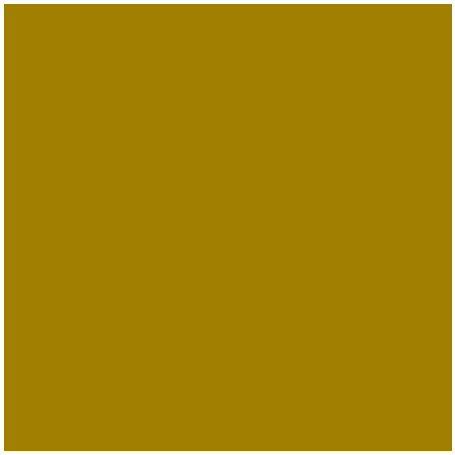
```
void main() {
    vec2 st = gl_FragCoord.xy/u_resolution.xy;

    gl_FragColor = vec4(st.x, .5, 0.0, 1.0);
}
```

Scale input

```
void main() {
    vec2 st = gl_FragCoord.xy/u_resolution.xy;
    vec2 ms = u_mouse.xy/u_resolution.xy;

    gl_FragColor = vec4(ms.x, ms.y, 0.0, 1.0);
}
```


3. Change colour with time

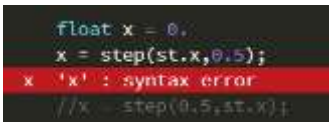
```
void main() {
    gl_FragColor = vec4(cos(u_time), .5, 0.0, 1.0);
}
```



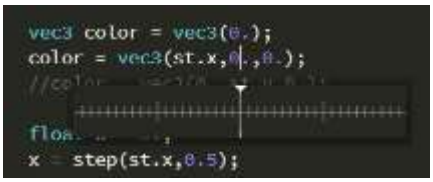
Features

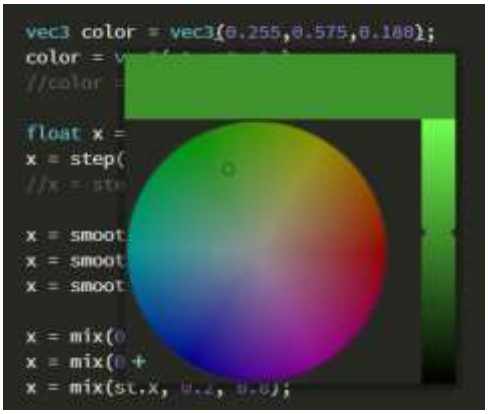
As you're editing you may have notice some of the funky built-in functionality:

Error reporting

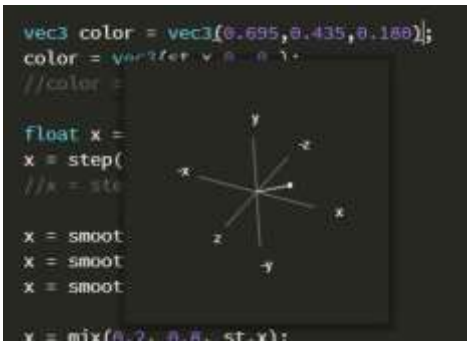


Controls





To switch to 3D xyz, click the "+" in the lower left.



Editing

ctrl-v, ctrl-x, ctrl-c

alt for column select

Shaping functions

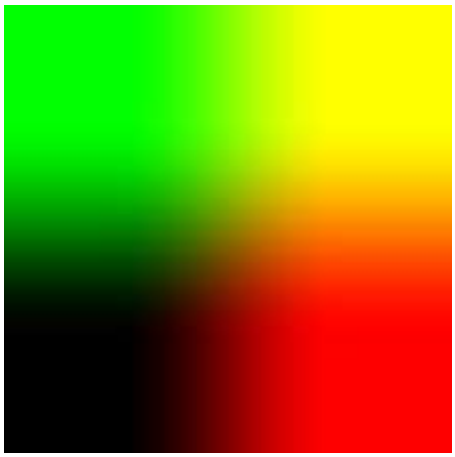
GLSL comes with many built-in functions, particularly geometry focussed:

step, sin, cos, fract, floor, ceil

length, distance, smoothstep

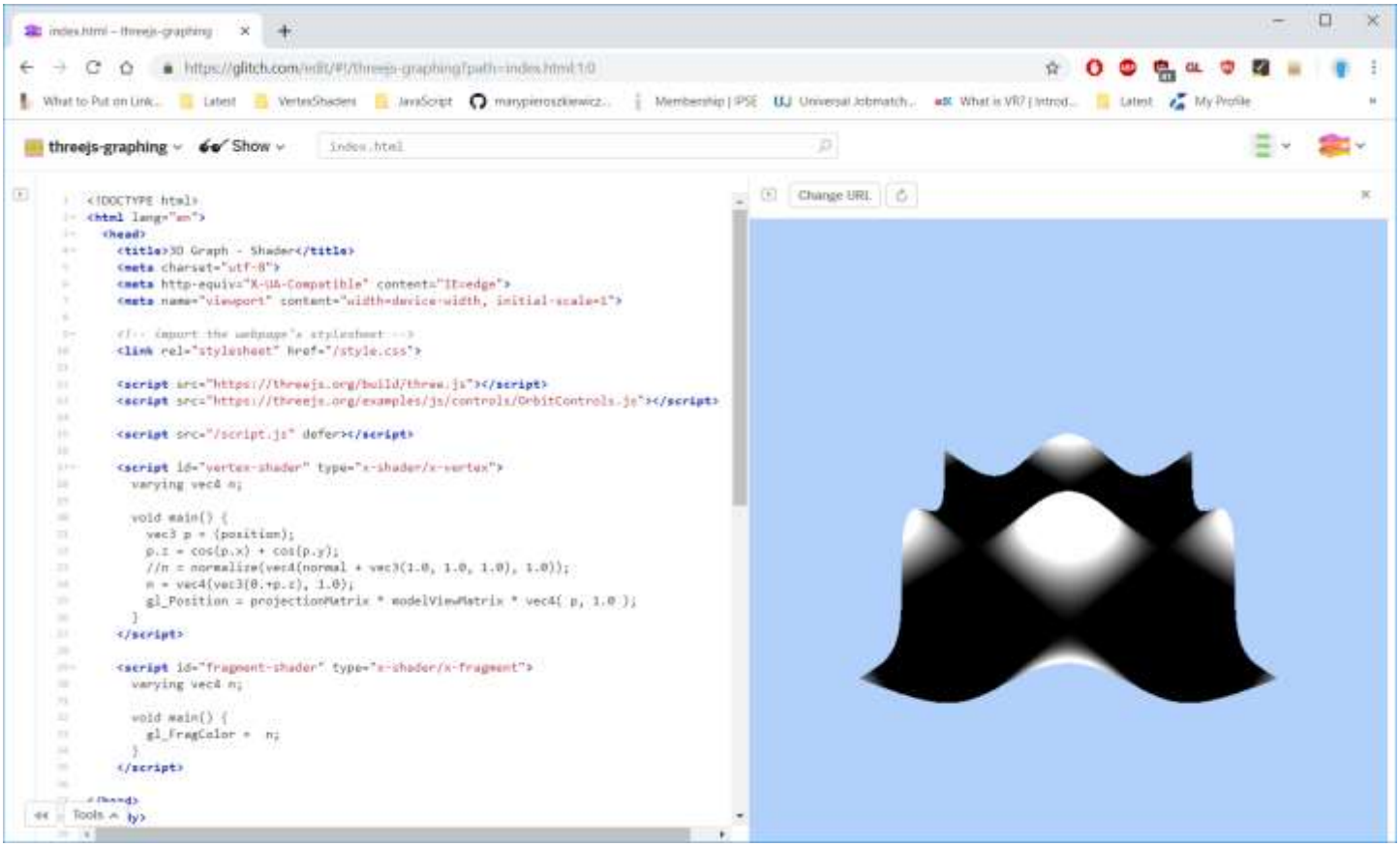
A good list can be found at: **http://www.shaderific.com/glsl-functions**

```
void main() {
    vec2 st = gl_FragCoord.xy/u_resolution.xy;

    vec2 c = step(.5, st);
    vec2 d = smoothstep(0.25, .75, st);

    gl_FragColor = vec4(d.x, d.y, 0.0, 1.0);
}
```



Visualisation of functions in 3D

The project at **https://glitch.com/~threejs-graphing** allows you to see the effects of the functions in 3D.



Part 3. Practical Examples

There are several examples, showing how to incorporate shaders into a web page.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Hello!</title>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <!-- import the webpage's stylesheet -->
    <link rel="stylesheet" href="/style.css">

    <script src="https://unpkg.com/shader-doodle"></script>

  </head>
  <body>

    <shader-doodle>
      void main() {
        vec2 st = gl_FragCoord.xy / u_resolution.xy;
        vec3 color = vec3(st.x, st.y, abs(sin(u_time)));

        gl_FragColor = vec4(color, 1.0);
      }
    </shader-doodle>

  </body>
</html>
```

https://github.com/halvves/shader-doodle

Pre-Defined Uniforms

Default (`<shader-doodle />`)

- `uniform float u_time;` : shader playback time (in seconds)
- `uniform float u_delta;` : delta time between frames (in seconds)
- `uniform int u_frame;` : shader playback frame
- `uniform vec4 u_date;` : year, month, day and seconds
- `uniform vec2 u_resolution;` : viewport resolution (in pixels)
- `uniform vec2 u_mouse;` : mouse pixel coords (x & y)

Shadertoy (`<shader-doodle shadertoy />`)

- `uniform float iTime;` : shader playback time (in seconds)
- `uniform float iDelta;` : delta time between frames (in seconds)
- `uniform int iFrame;` : shader playback frame
- `uniform vec4 iDate;` : year, month, day and seconds
- `uniform vec2 iResolution;` : viewport resolution (in pixels)
- `uniform vec4 iMouse;` : -- mouse pixel coords. xy: current (if mousedown), zw : click.

https://glitch.com/edit/#!/shader-doodle-test?path=index.html:13:47

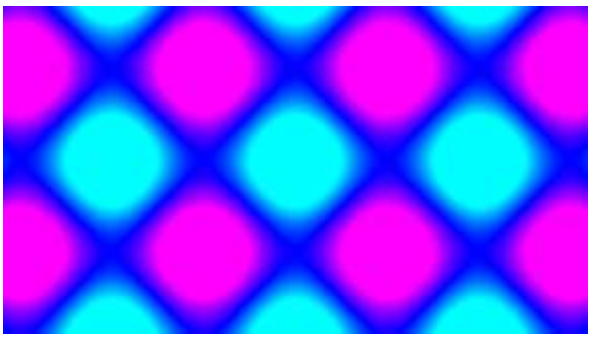
Part 4. Examples

Graphing with step

```
void main() {
  vec2 st = gl_FragCoord.xy/u_resolution.xy;
  st.x *= u_resolution.x/u_resolution.y;
  st.x *= u_resolution.x/u_resolution.y;
  st *= 10.;
  st -= 5.;

  float x = step(cos(st.x), st.y);
  float y = step(sin(st.x), st.y);

  vec3 color = vec3(0.);
  color = vec3(x, y, 1.);

  gl_FragColor = vec4(color, 1.0);
}
```

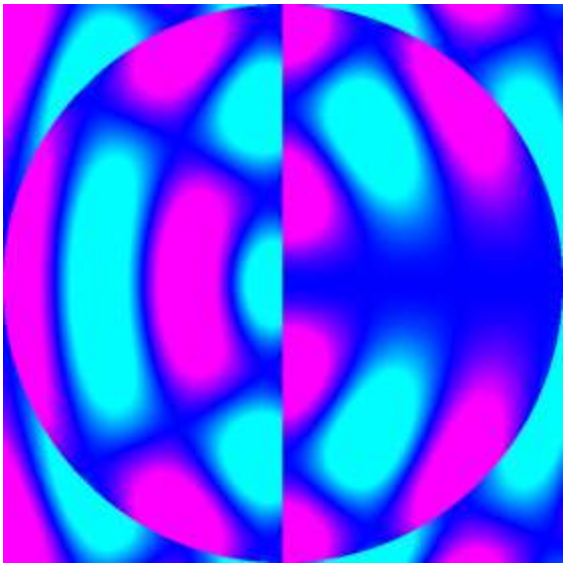
Repeating patterns

```
void main() {
  vec2 st = gl_FragCoord.xy/u_resolution.xy;
  st.x *= u_resolution.x/u_resolution.y;
  st *= 20.;
  st -= 10.;

  float l = length(st);
  float c = cos(st.x);
  float s = sin(st.y);

  vec3 color = vec3(0.);
  color = vec3(s, c, 1);
  color += vec3(-c, -s, 1);

  gl_FragColor = vec4(color, 1.0);
}
```



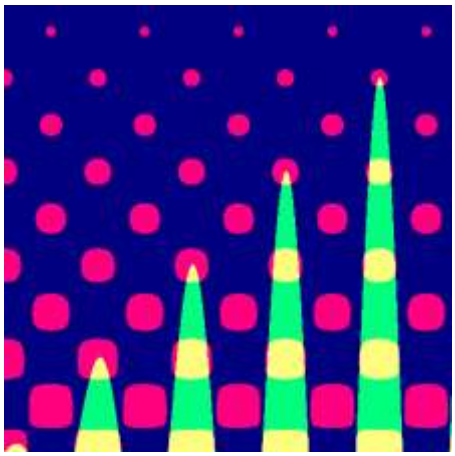
Surprise

```
void main() {
  vec2 st = gl_FragCoord.xy/u_resolution.xy;
  st.x *= u_resolution.x/u_resolution.y;
  st *= 20.;
  st -= 10.;

  float l = length(st);
  float c = cos(mod(l, 10.));
  float s = cos(mod(st.x, 10.));

  vec3 color = vec3(0.);
  color = vec3(s, c, 0);
  color += vec3(-c, -s, 1);

  gl_FragColor = vec4(color, 1.0);
}
```



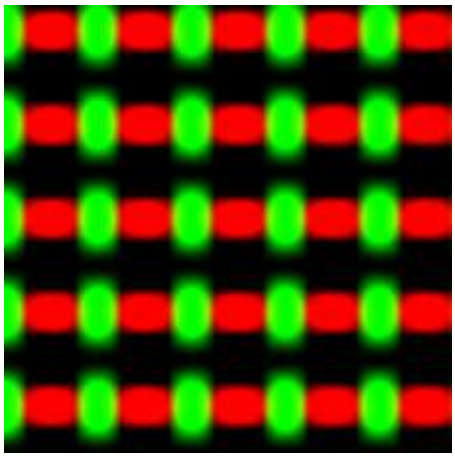
Spotty and spikey

```
void main() {
    vec2 st = gl_FragCoord.xy/u_resolution;

    float y = cos(st.x*30.);
    float x = cos(st.y*30.);

    vec3 color = vec3(y);

    color = (1.0-pct)*color+pct*vec3(0.0,1.0,0.0);
    color  = vec3(step(st.y, x*y));
    color  += vec3(step(st.y, st.x*y));

    color  = vec3(step(st.y, x*y), step(st.y,
st.x*y), .5);

    gl_FragColor = vec4(color,1.0);
}
```



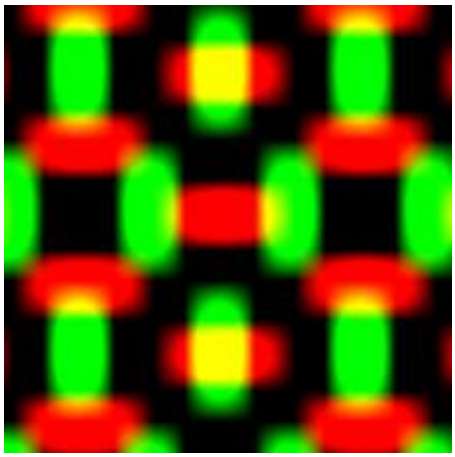
Tartan or a surprising mix

```
void main() {
    vec2 st = gl_FragCoord.xy/u_resolution;

    float y = cos(st.x*30.);
    float x = cos(st.y*30.);

    vec3 color = vec3(y);
    color  = vec3(x*y, y  , .5);
    color -= vec3(x  , y*x, .5);

    gl_FragColor = vec4(color,1.0);
}
```



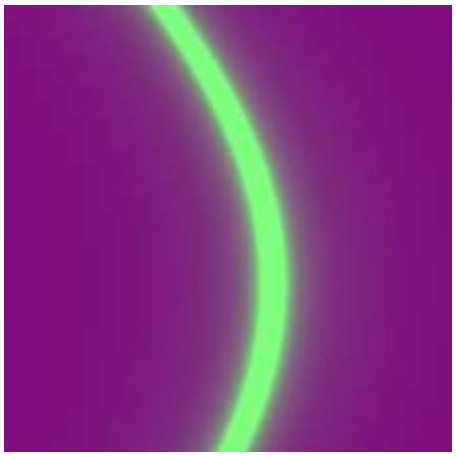
A pattern

```
void main() {
  vec2 st = gl_FragCoord.xy/u_resolution.xy;
  st.x *= u_resolution.x/u_resolution.y;
  st *= 10.;
  st += 10.;

  float x = st.x;
  float y =st.y;

  vec3 color = vec3(0.);
  //color = vec3(cos(st.xyx));
  //color = vec3(cos(x+y)-cos(y-x));
  color = vec3(sin(x+y+y)-cos(x-y-y), .0,.0);

  x += .5;
  y -= .5;
  color -= vec3(.0,sin(2.*x+y)-cos(2.*x-y),.0);

  gl_FragColor = vec4(color,1.0);
}
```



A line

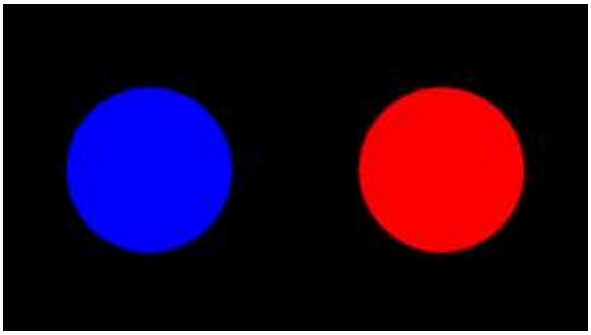
```
void main() {
  vec2 uv = gl_FragCoord.xy / u_resolution.xy;
  float col=0.0;
  float i=1.0;
  vec2 spec = vec2(0.1, 0.6);
  uv.x += sin(i * 20.0 + spec.x * 30.0 + uv.y * 1.5)
* spec.y;
  col += abs(0.044/uv.x) * spec.y;
  gl_FragColor = vec4(.5, col, .5, 1.0);
}
```



3d plane

Two ways to draw a circle

```
void main() {
  vec2 st = gl_FragCoord.xy/u_resolution.xy;
  st.x *= u_resolution.x/u_resolution.y;
  st *= 10.;
  st -= 5.;
  st.x += 2.5;

  vec3 color = vec3(0.);
  float s = step( st.y * st.y + st.x * st.x, 2. );
  color = vec3(0, 0, s);


  st.x -= 5.;
  float l = length(st);
  float t = step(l, sqrt(2.));
  color += vec3(t, 0, 0);


  gl_FragColor = vec4(color, 1.0);
}
```



Part 5. Resources

Vertex editor only

http://www.pleek.net/vertexlove/
https://www.vertexshaderart.com/

Vertex and fragment editor

https://shaderfrog.com/app/editor
http://shdr.bkcore.com/
http://www.kickjs.org/example/shader_editor/shader_editor.html
https://cyos.babylonjs.com/
http://bkcore.com/blog/3d/shdr-online-glsl-shader-editor-viewer-validator.html

Node editor

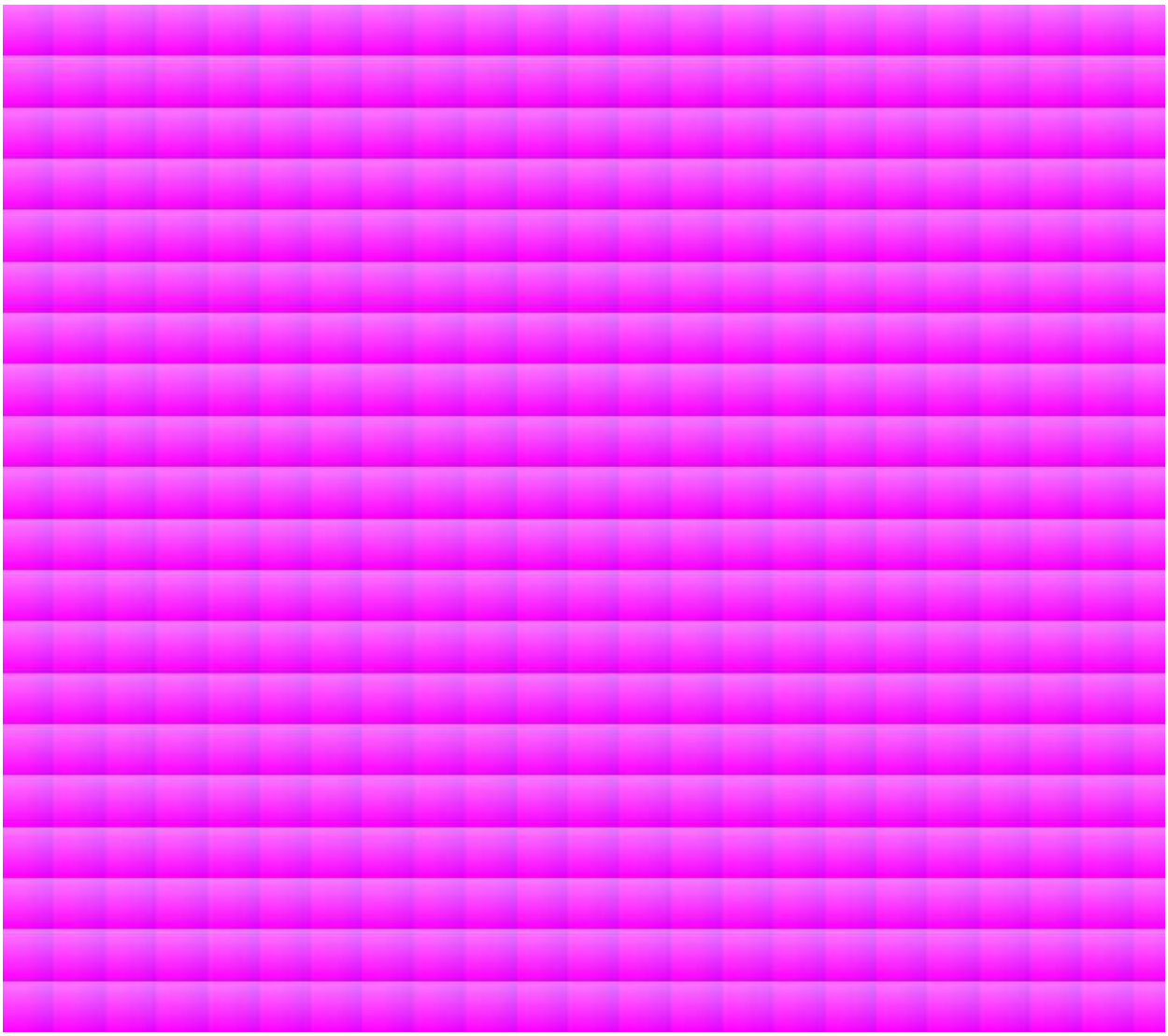
https://www.gsn-lib.org/index.html#projectName=public3dshader&graphName=NormalTrans
https://victhorlopez.github.io/editor/

Background info

http://www.shaderific.com/glsl
https://www.awwwards.com/inspiration/5981b0a1e13823534b28b8cb
https://medium.com/@Zadvorsky/into-vertex-shaders-594e6d8cd804

Cool

https://glitch.com/~shader-doodle-test



```
vec2 f01(vec2 st) {
   return vec2(cos(st));
}

vec2 f02(vec2 st) {
   float l = length(st);
   return vec2(l,l);
}

vec2 f03(vec2 st) {
   float l = length(st)*5.;
   float lm = mod(l, 1.);

   vec2 ss = step(.5, st);
   float x = mod(st.x, .5);
   float y = mod(st.y, .5);

   float s = sin(st.x);
```

```
    float c = cos(st.y);

    //return vec2(mod(s, .5), mod(c, .5));//ss;//vec2(step(l,.5),step(l, .5));
    return vec2(cos(x), sin(y));//ss;//vec2(step(l,.5),step(l, .5));
}

vec2 f04(vec2 st) {
    float l = length(st);
    float l1 = mod(l * 10., 4.);
    float s = cos(l1);
    float c = step(s, .5);

    return vec2(s,s);
}


void main() {
    vec2 st = gl_FragCoord.xy/u_resolution.xy;
    st.x *= u_resolution.x/u_resolution.y;

    vec3 color = vec3(0.);
    color = vec3(f03(st*10.),1.);

    gl_FragColor = vec4(color,1.0);
}
```