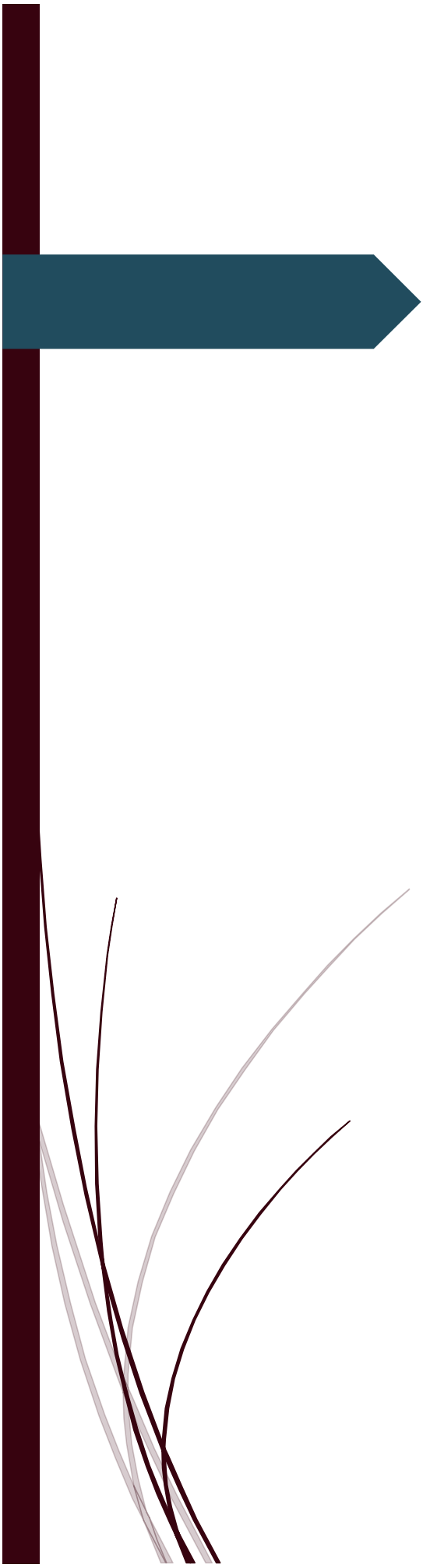


A Gentle Introduction

4 - Vertex Shaders, again

Carl Bateman
WebGL Workshop

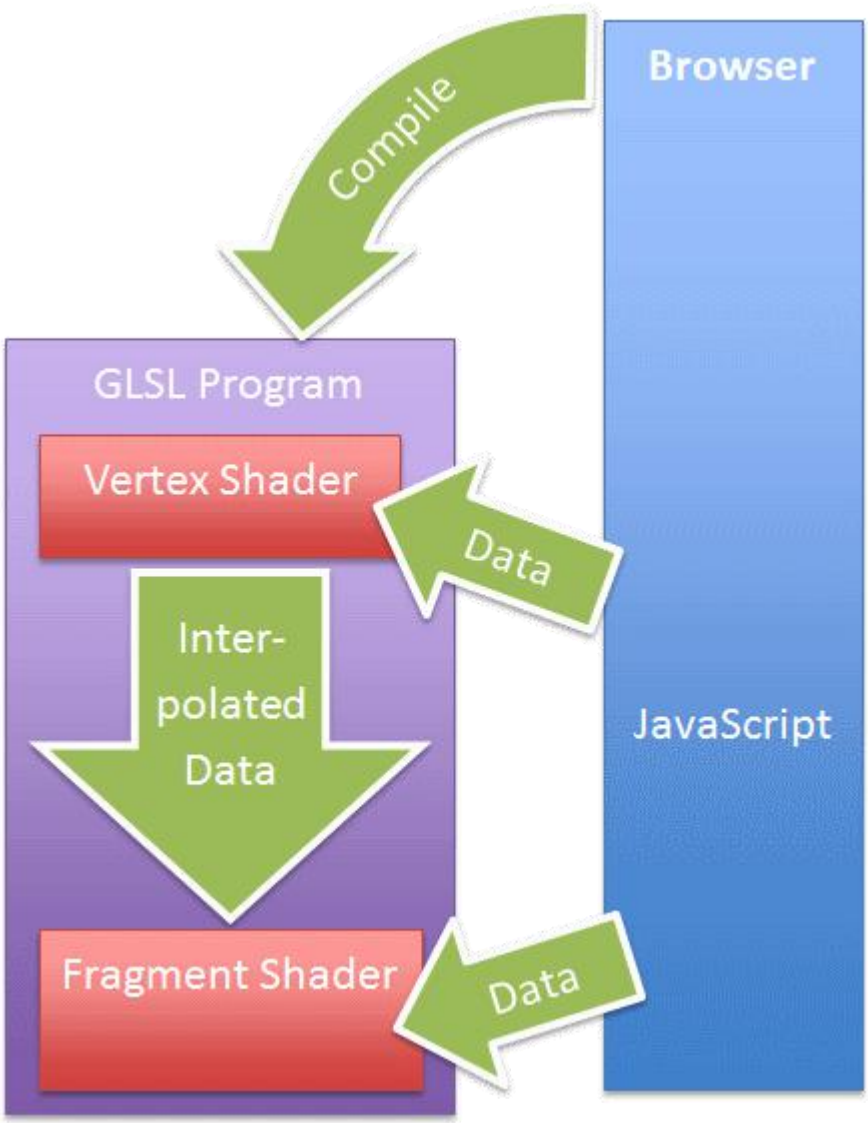
Table of Contents

Part 1. Background

WebGL

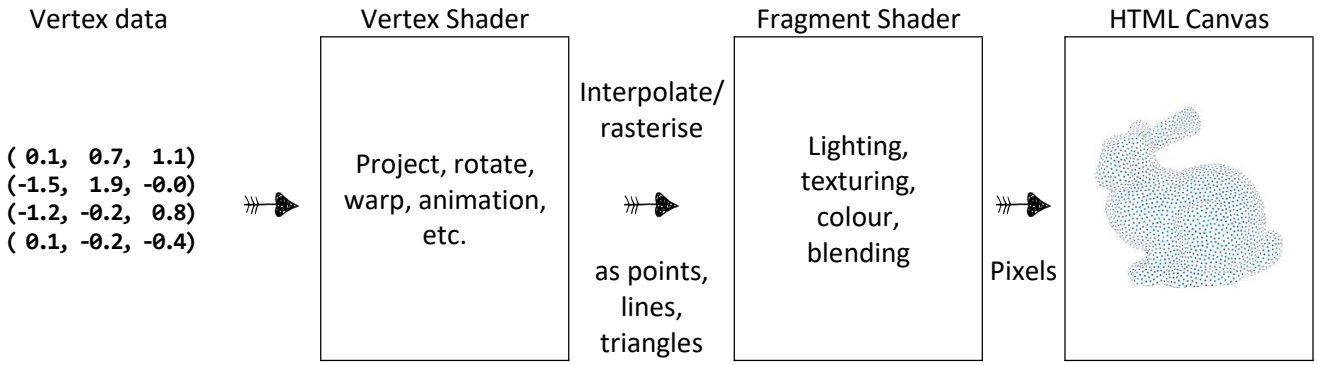
- Canvas based
- JavaScript API
- Renders 2D and 3D graphics
- Giving access to the GPU
- Integrated into all web standards

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

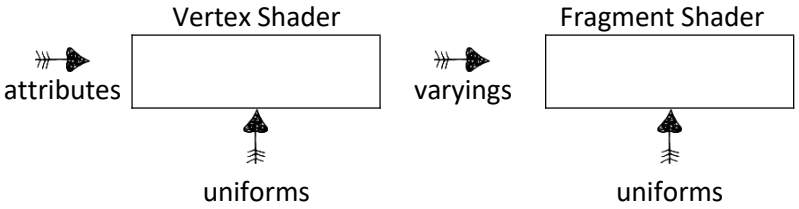


Graphics Pipeline

Simplified!



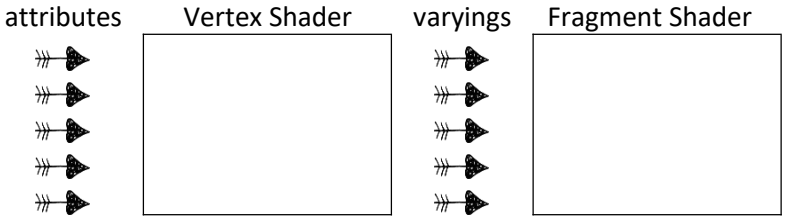
Sharing Data



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);
}
```


POWER
UNLIMITED POWER!
quickmeme.com

Part 2. Vertex Shaders

The **Vertex Shader** is the programmable Shader stage in the rendering pipeline that handles the processing of individual vertices.

Vertex shaders are fed Vertex Attribute data, as specified from a vertex array object by a drawing command.

A vertex shader receives a single vertex from the vertex stream and generates a single vertex to the output vertex stream. There must be a 1:1 mapping from input vertices to output vertices.

Vertex shaders typically perform transformations to post-projection space, for consumption by the Vertex Post-Processing stage. They can also be used to do per-vertex lighting, or to perform setup work for later shader stages.

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

Vertex shaders are written using GLSL (a strongly typed C-like language tailored to geometry)

GLSL code is **compiled** and **run on the GPU**

Client side (JavaScript - browser - CPU)

Server side (GLSL -GPU)

Data (vertices, colours, etc.) is passed to the shader program from JavaScript via **uniforms** (for single values) and **attributes** (for arrays of values). Uniforms and attributes cannot be changed by the shader program.

Data is processed in parallel:

Attributes -- each item in the array is passed to the shader program (not the whole array) and processed† in isolation in a single "instance" of the shader program (multiple attributes can be used, but only individual items are processed).

Uniforms -- a single value that can be used in all "instances" of the shader program (multiple uniforms can be used).

† (may be left unprocessed).

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

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

Built-in variables

The OpenGL Shading Language defines a number of special variables for the various shader stages. These built-in variables (or built-in variables) have special properties. They are usually for communicating with certain fixed-functionality. By convention, all predefined variables start with "`gl_`"; no user-defined variables may start with this.

In

- `gl_VertexID` the index of the vertex currently being processed

Out

- `gl_Position` the clip-space output position of the current vertex
- `gl_PointSize` the pixel width/height of the point being rasterized

Shake it all about

- `do_TheOkeyCokey` that's what it's all about

Supplied variables

On-line shader editors usually provide their own special variables to make life easier for the user (you). These may be declared in the shader. If they're not declared then they're probably injected "behind the scenes".

e.g. **http://www.pleek.net/vertexlove/**

```
// explicitly declared input
uniform   vec3  controls;     // RGB sliders
uniform   float time;         // IN SECONDS
attribute float vertexSize;

// explicitly declared output passed to fragment shader
varying   vec4  vertexColor;

void main() {
  vertexColor        = vec4(color, 1.0);
  vec4 finalPosition = vec4(position, 1.0);
  gl_Position        = projectionMatrix * modelViewMatrix * finalPosition;
  gl_PointSize       = 10.;
}

// implicit input - inserted by Three.js
color, position, projectionMatrix, modelViewMatrix, finalPosition
```

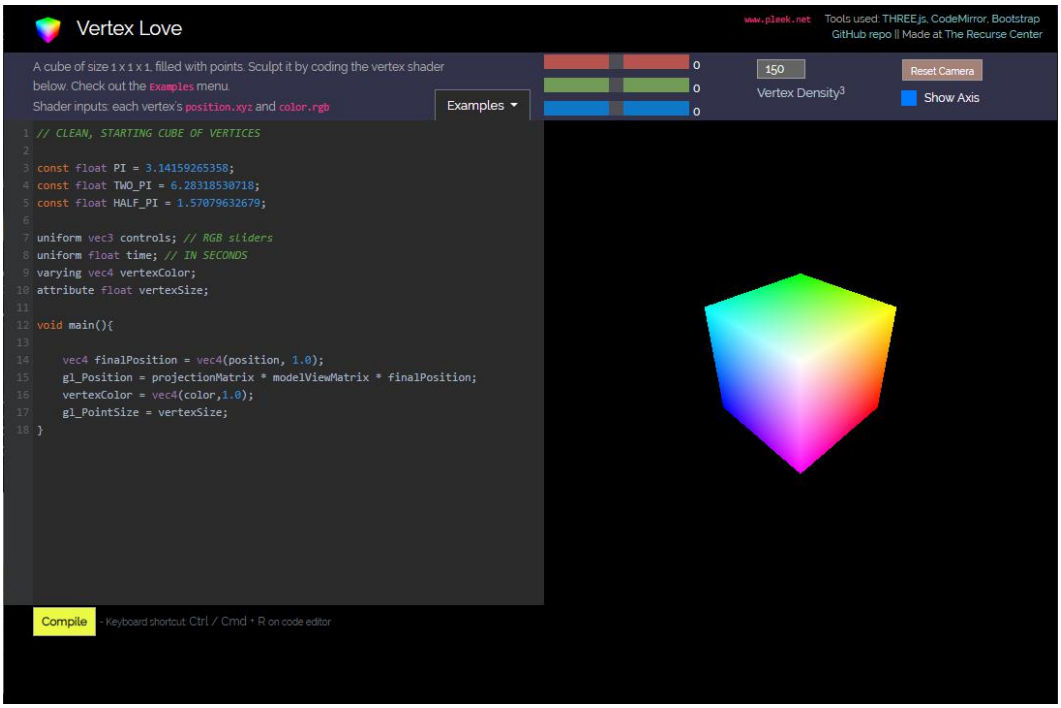
These values are sent from the JavaScript side (browser/CPU/client) to the the GLSL side (GPU/server) with code similar to this:

```
// set uniforms directly
gl.uniformMatrix4fv(shaderProgram.projectionMatrixLocation, false, projectionMatrix);
gl.uniformMatrix4fv(shaderProgram.modelViewMatrixLocation, false, modelViewMatrix);
gl.uniform1f(shaderProgram.time, (Date.now() - start) / 1000);

// bind buffer then set attributes
gl.bindBuffer(gl.ARRAY_BUFFER, positionBuffer);
gl.vertexAttribPointer(shaderProgram.positionLocation, positionBuffer.itemSize, gl.FLOAT, false, 0, 0);

gl.bindBuffer(gl.ARRAY_BUFFER, colorBuffer);
gl.vertexAttribPointer(shaderProgram.vertexColorLocation, colorBuffer.itemSize, gl.FLOAT, false, 0, 0);
```

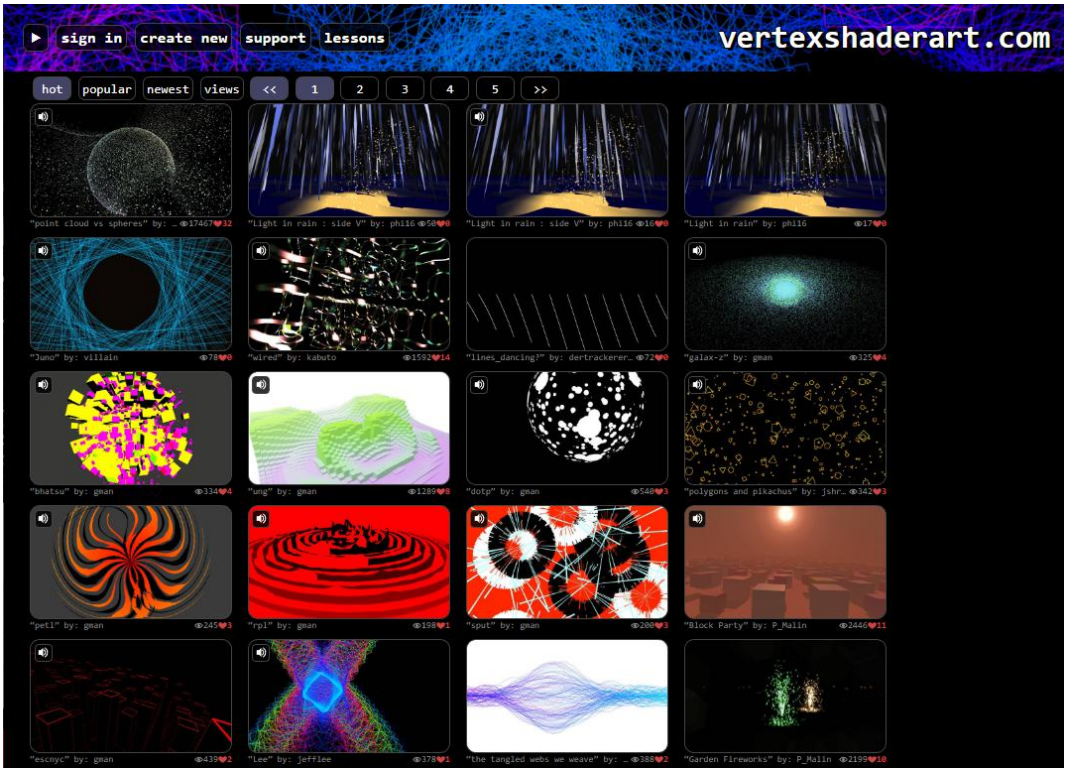
Part 3. Exercises

Take a look at:

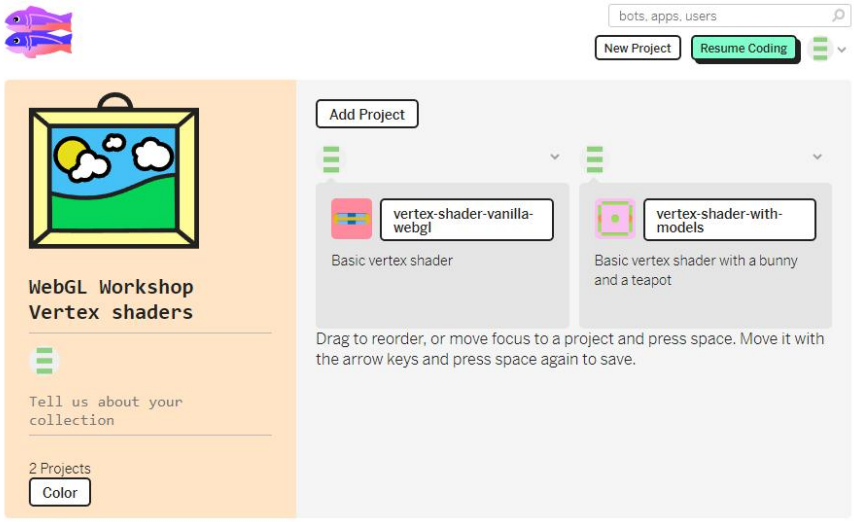
http://www.pleek.net/vertexlove/



https://www.vertexshaderart.com/



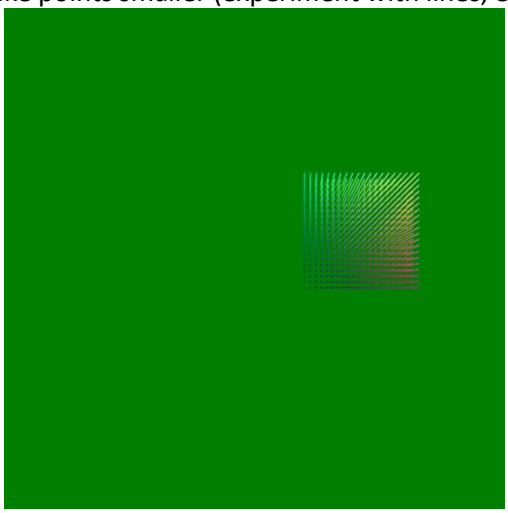
https://glitch.com/@CarlBateman/web-gl-workshop-vertex-shaders



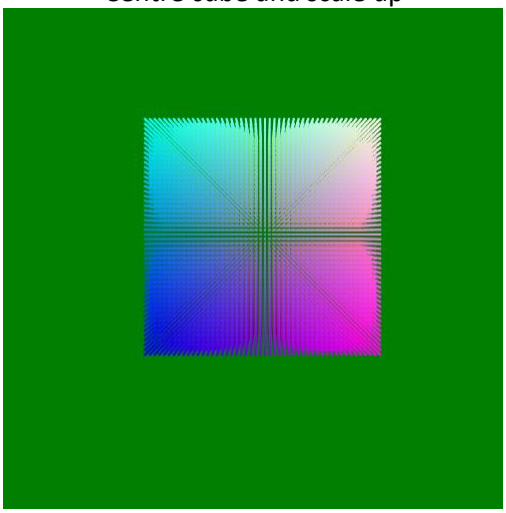
https://glitch.com/~vertex-shader-vanilla-webgl
https://glitch.com/~vertex-shader-with-models

Using one of the above online editors:

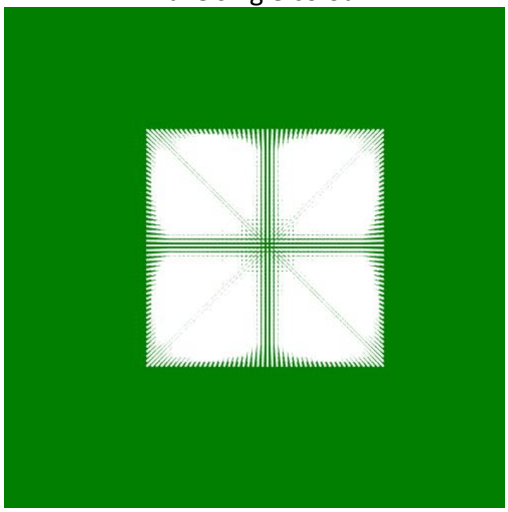
Make points smaller (experiment with lines, etc.)



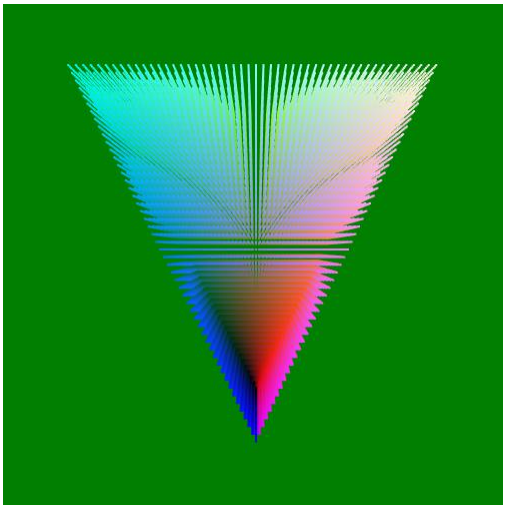
Centre cube and scale up



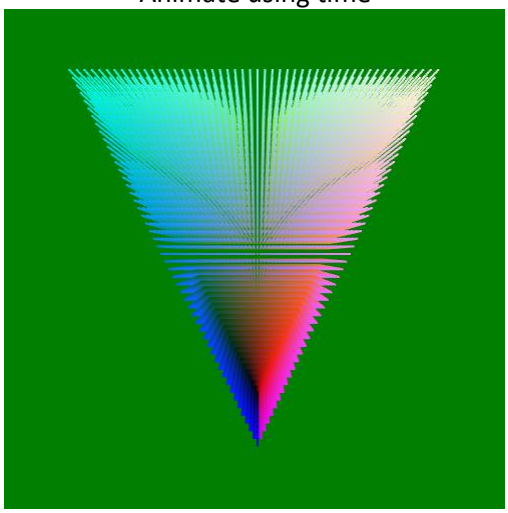
Make single colour



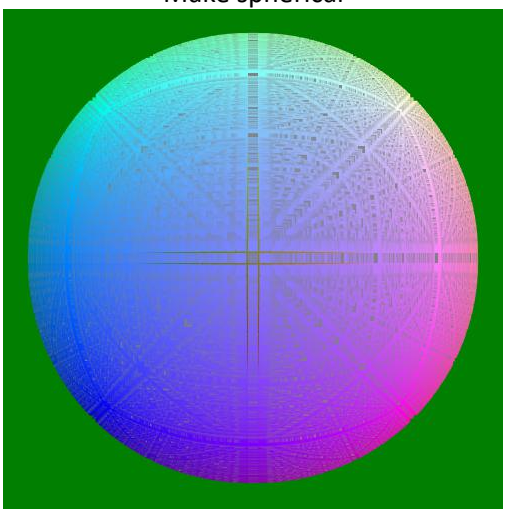
Narrow the bottom



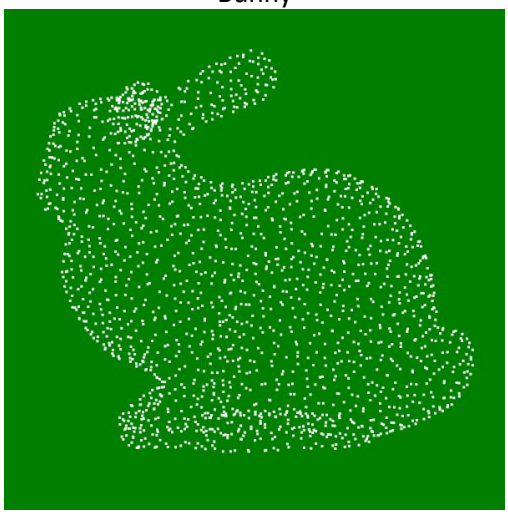
Animate using time



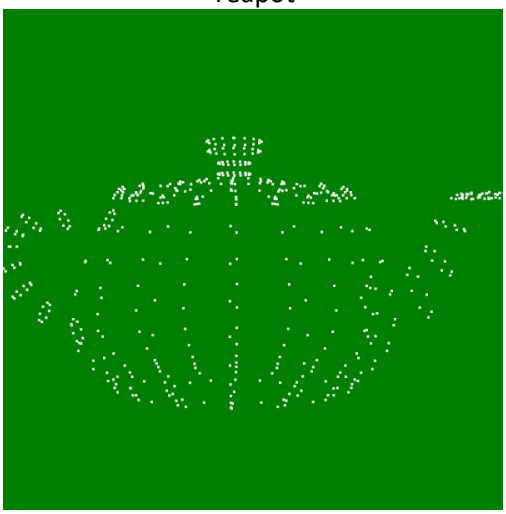
Make spherical



Bunny



Teapot



Some functions

http://www.shaderific.com/glsl-functions

GLSL is aimed at graphics and geometric functions. Some you might like to try:

sin, cos, tan, pow, exp, sqrt, sign, floor, ceil, fract, mod, min, max, clamp, length, distance

Part 4. Practical Examples

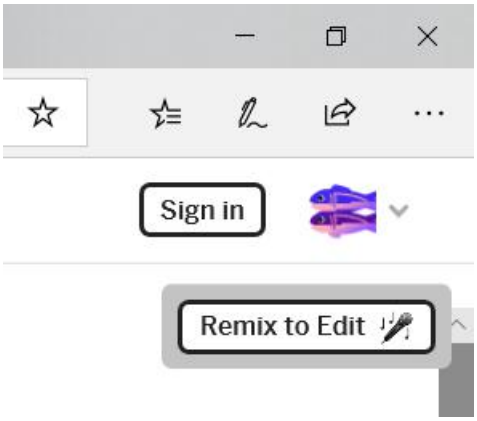
https://glitch.com/@CarlBateman/web-gl-workshop-vertex-shaders

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

Remix!

Select a project from the collection

Click on the Remix to Edit button in the top right.



This will create your own version of the project to work on.

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

Some functions

http://www.shaderific.com/glsl-functions

Cool

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