Algorithmic art

Notes: there are a number of sources of friction you may not expect when using WebGL.
You'll need some understanding of HTML and CSS.
GPU usage and battery become important.
3D model file size is ***very*** important for mobile – does the device have enough RAM, is there enough bandwidth, etc..

Two sides
JavaScript and GLSL

GLSL/Shader Side

http://www.alanzucconi.com/2016/07/01/signed-distance-functions/

http://www.iquilezles.org/www/articles/raymarchingdf/raymarchingdf.htm

http://iquilezles.org/www/articles/distfunctions/distfunctions.htm

Book of Shadows

http://editor.thebookofshaders.com

http://www.iquilezles.org/apps/graphtoy/
https://thebookofshaders.com/glossary/

Some On-line editors

https://www.shadertoy.com
https://shaderfrog.com/app
http://glslsandbox.com
http://shdr.bkcore.com

Shader Material Editors

http://cyos.babylonjs.com – for materials useful templates
http://www.kickjs.org/example/shader_editor/shader_editor.html

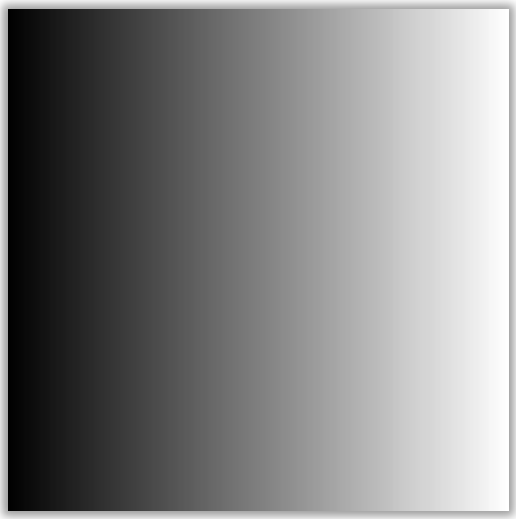
Some Background

linear
y = mx + c
quadratic
y = x*x*x + x*x - x
sinusoidal
y = sin(x)
y = cos(x)

Basic

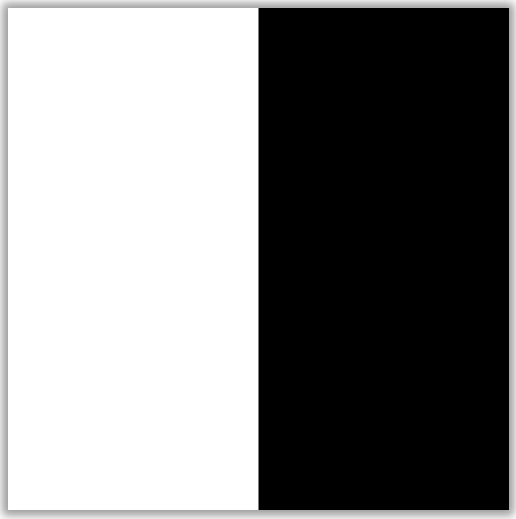
```
precision mediump float;

#define PI 3.1415926

uniform vec2 u_resolution;
uniform vec2 u_mouse;
uniform float u_time;

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

```
    vec3 color = vec3(0.);
    color = vec3(st.x);

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



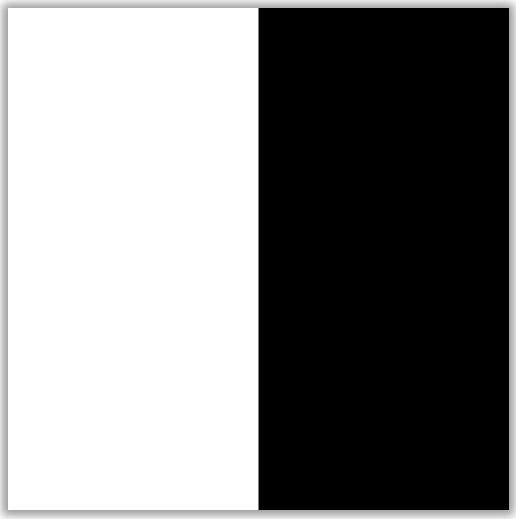
Step non-function

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

    vec3 color = vec3(0.);
    st.x = st.x > 0.5 ? 0.0: 1.0;
    color = vec3(st.x);

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



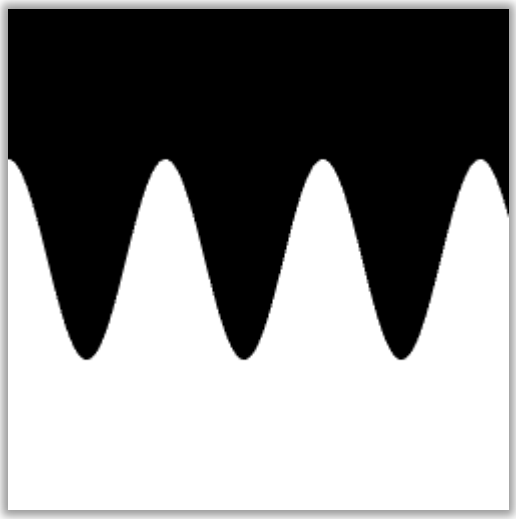
Step function

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

    vec3 color = vec3(0.);
    st.x = step(st.x, 0.5);
    color = vec3(st.x);

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



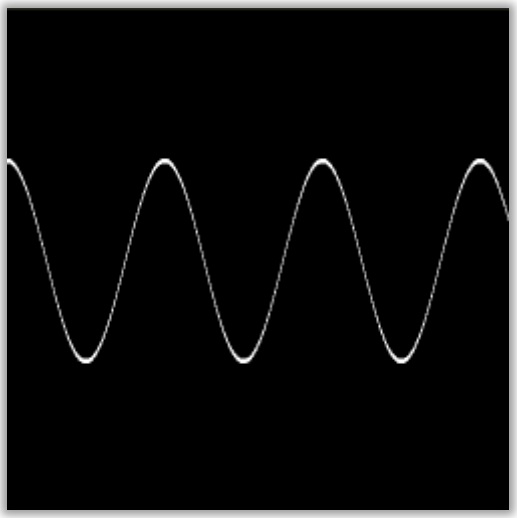
Sin

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

    vec3 color = vec3(0.);
    st *= 10.;
    float c = step(st.y, 2.*cos(2. * st.x) + 5.);
    color = vec3(c);

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



Sin stroke

```
float stroke(float pos, float limit, float width) {
    float d1 = step(pos, limit - width/2.);
    float d2 = step(pos, limit + width/2.);
    return d2-d1;
}

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

    vec3 color = vec3(0.);
    st *= 10.;
    float c = stroke(st.y, 2.*cos(2. * st.x) + 5., .1);
    color = vec3(c);

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

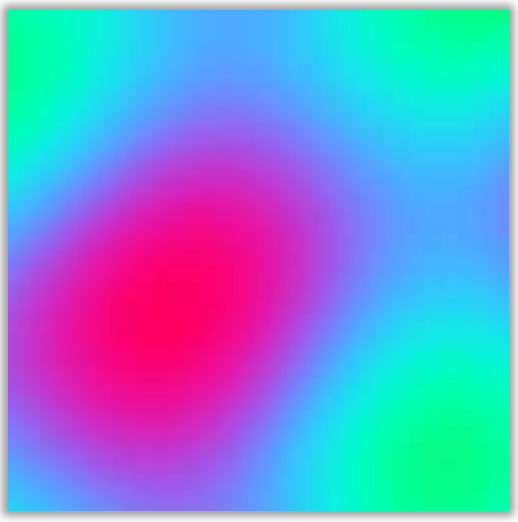


Pre-plasma

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

  float v = cos(PI * st.x) + sin(PI * st.y);

  gl_FragColor = vec4(vec3(v, 1.-v, 1.0), 1.0);
}
```

Plasma

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

  float v = cos(tau * st.x + sin(st.y *2.0 + u_time));
  v += cos(tau * st.y + sin(st.x *2.0 + u_time*0.1));
  v+=sin(length(st));
  v*=0.125;
  v+=0.125;
  v+=u_time*0.01;

  vec3 c = vec3(0.5) + 0.5 * cos(tau * (v + vec3(cos(u_time*0.1), 0.1, 0.3)));
  gl_FragColor = vec4(c, 1.0);
}
```



JavaScript side

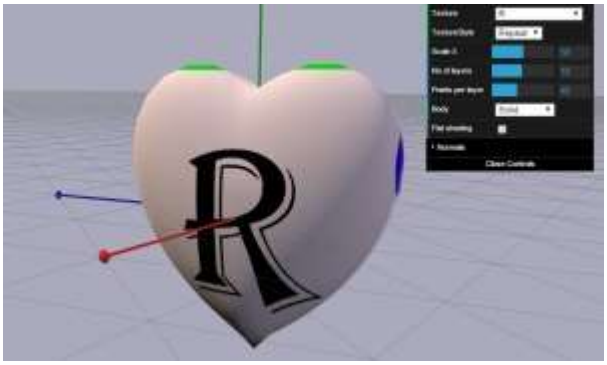
Three.js

To make life simpler I'll be using the Three.js library.

Heart

Three.js supports a number of "geometries", this is my attempt at implementing a 3D cardioid (heart) surface function, described as:

$2x^2 + y^2 + z^2 - 1)^3 - 0.1x^2z^3 - y^2z^3 = 0$

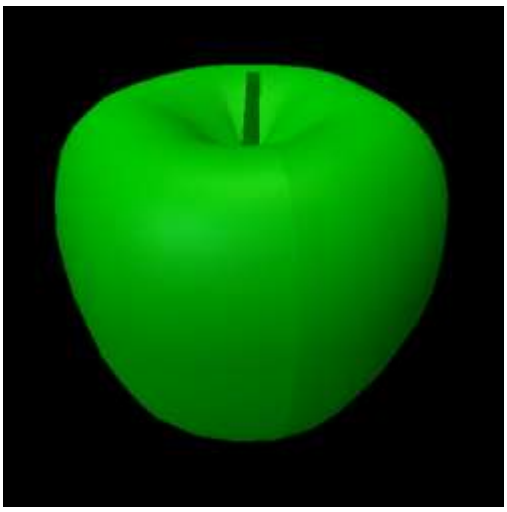


Apple

Three.js actually provides a function to do all the above.

Here, I pass the "apple" function to `ParametricBufferGeometry`.

```
function apple(u, t, optionalTarget) {
  // apple
  // http://www.3d-meier.de/tut3/Seite100.html
  var result = optionalTarget || new THREE.Vector3();
  u = u * 2 * Math.PI;
  var v = Math.PI - 2 * Math.PI * t;

  var x, y, z;

  var r1 = 4, r2 = 4;

  z = Math.cos(u) * (r1 + r2 * Math.cos(v)) + Math.pow(v / Math.PI, 20);
  x = Math.sin(u) * (r1 + r2 * Math.cos(v)) + 0.25 * Math.cos(5 * u);
  y = -2.3 * Math.log(1 - v * 0.3157) + 6 * Math.sin(v) + 2 * Math.cos(v);

  x += .001;
  y += .001;
  z += .001;

  return result.set(x, y, z);
}
```



JavaScript AND GLSL

It's October, so in Hallowe'en fashion I'll be using a skull for my little experiments… ha, ha, hah!

Skully

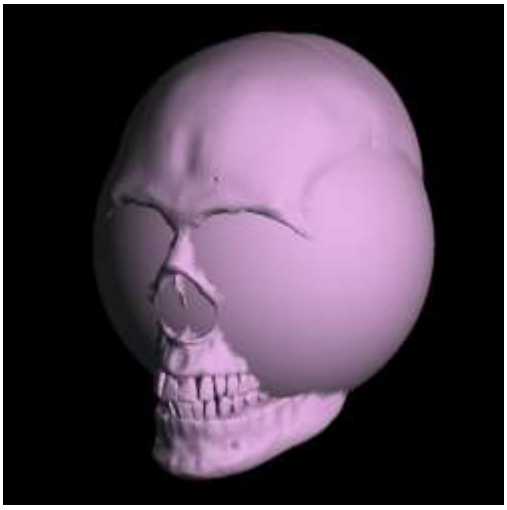
This is the unadulterated version. Three.js supports many 3D formats (see https://github.com/mrdoob/three.js/tree/dev/examples/js/loaders).

Spherised Skull

Points on the skull are pushed out onto the surface of a sphere.

This is done in the ***vertex shader***, the original data is left unchanged.

```
varying vec3 vPos;
varying vec3 vNormal;
uniform float radiusLimit;

void main() {
  vec3 sphere_pt = position;
  vec3 sphere_normal = normal;
  if(length(position) < radiusLimit) {
    sphere_pt = normalize(position) * radiusLimit;
    //sphere_normal = normalize(position);
  }
  vPos = (modelMatrix * vec4(sphere_pt, 1.0 )).xyz;

  vNormal = normalMatrix * sphere_normal;
  gl_Position = projectionMatrix * modelViewMatrix * vec4(sphere_pt,1.0);
}
```



Coloured Skull

The colour of each pixel is determined by passing its underlying position on the skull and performing a plasma calculation on it.

The code is in the ***fragment shader***.

```
varying vec3 vPos;
varying vec3 vNormal;
uniform float radiusLimit;

void main() {
  vec3 sphere_pt = position;
  vec3 sphere_normal = normal;
  if (length(position) < radiusLimit) {
    sphere_pt = normalize(position) * radiusLimit;
    sphere_normal = normalize(position);
  }
  vPos = (modelMatrix * vec4(sphere_pt, 1.0)).xyz;

  vNormal = normalMatrix * sphere_normal;
  gl_Position = projectionMatrix * modelViewMatrix * vec4(sphere_pt, 1.0);
}
```