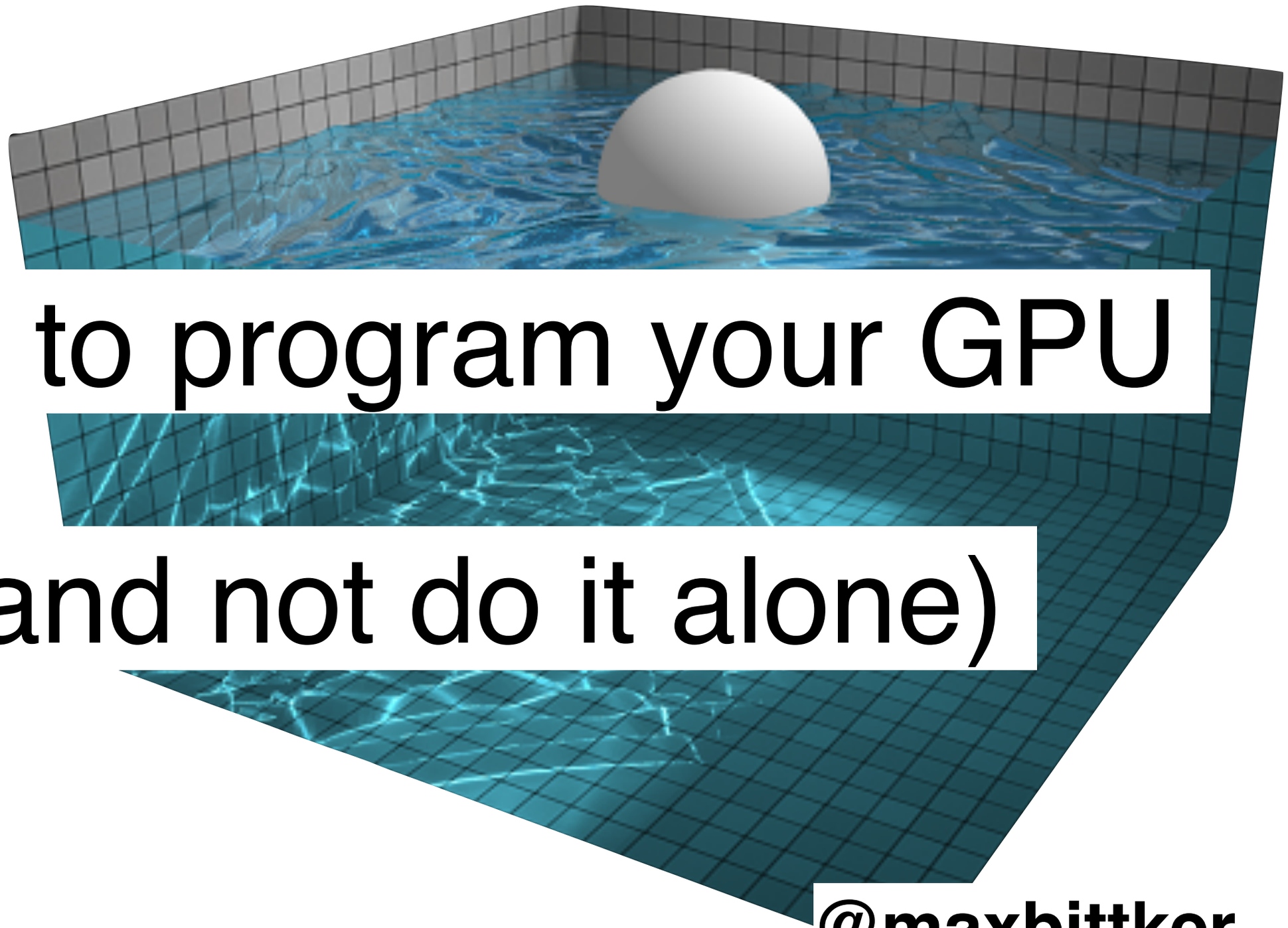


How to program your GPU

(and not do it alone)

@maxbittker

“Not programming alone”

```
var wow = require('wow');
```

```
import perfect from 'perfect';
```

Javascript Bundlers

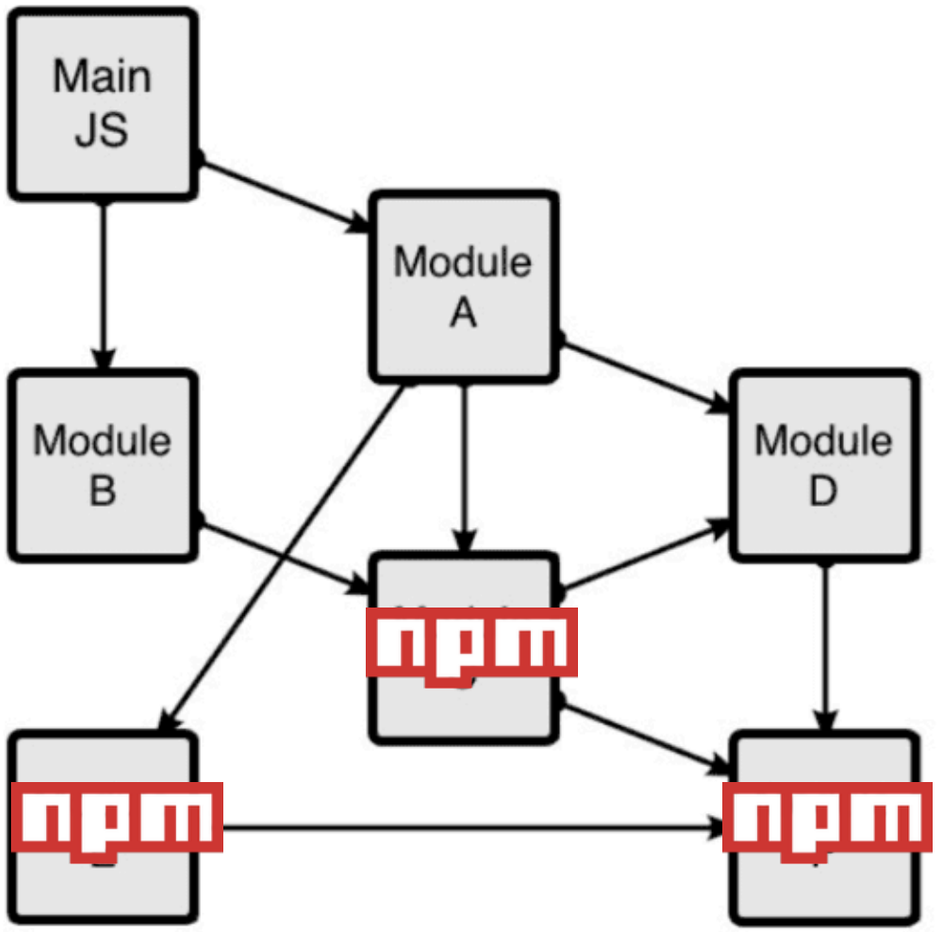
Implementation of modules for a language that doesn't have them!*

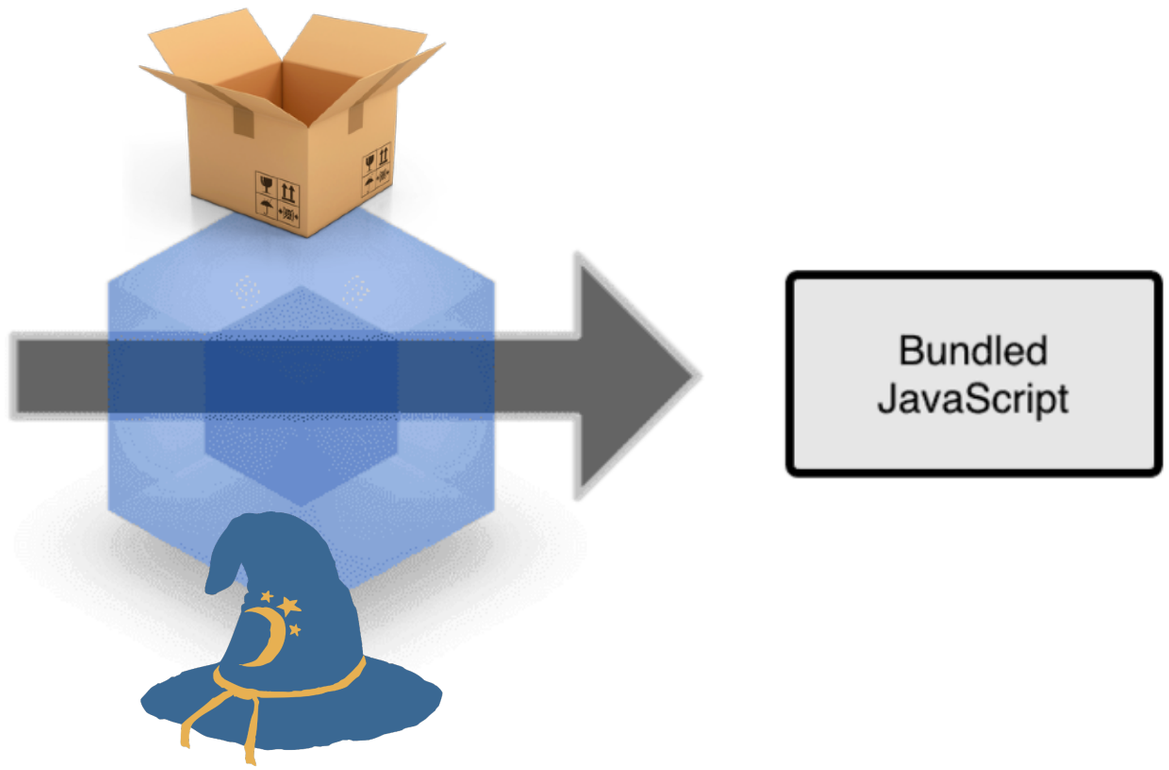
Webpack

Browserify

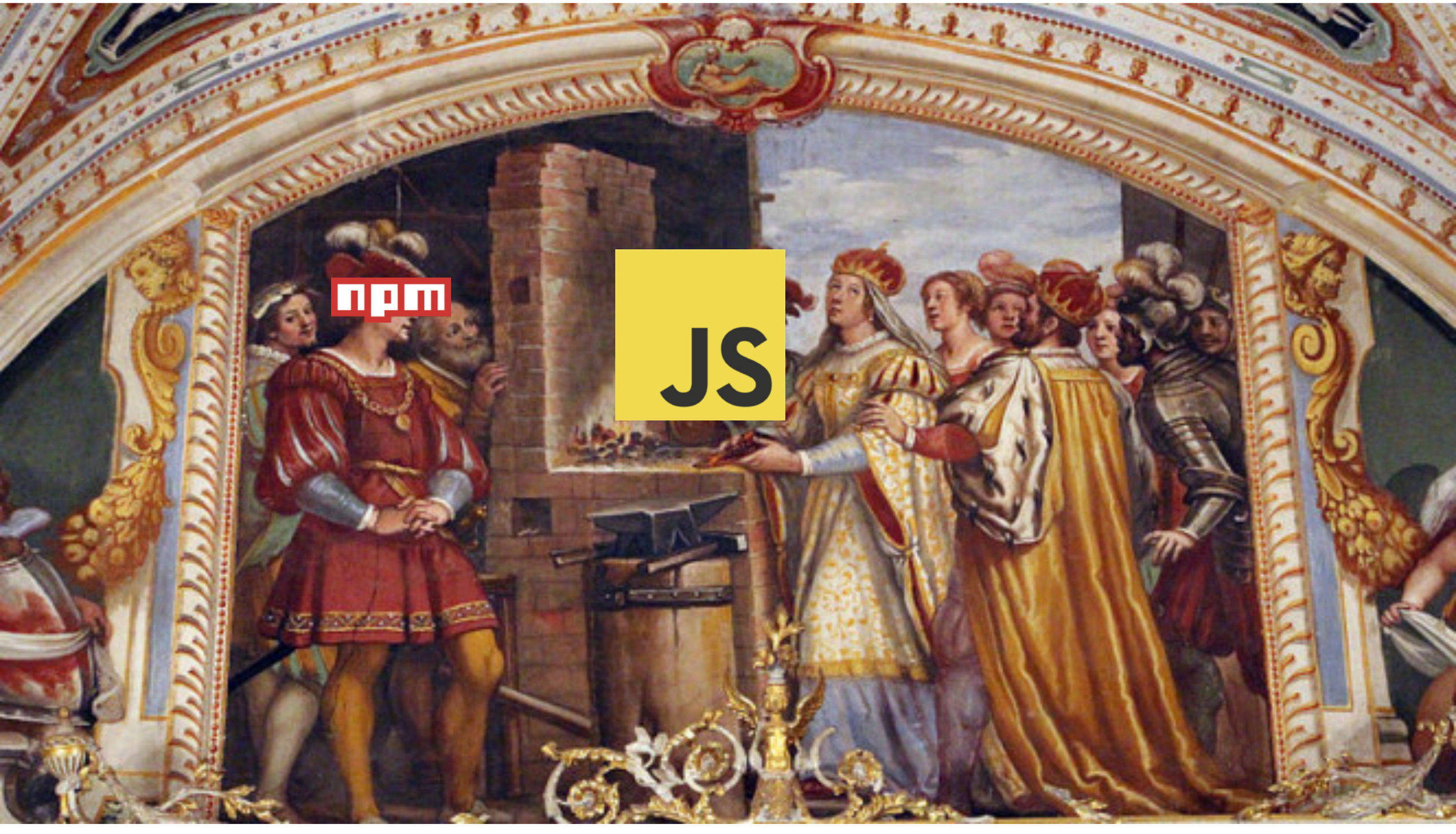
*since es6, it has them

Javascript Bundlers





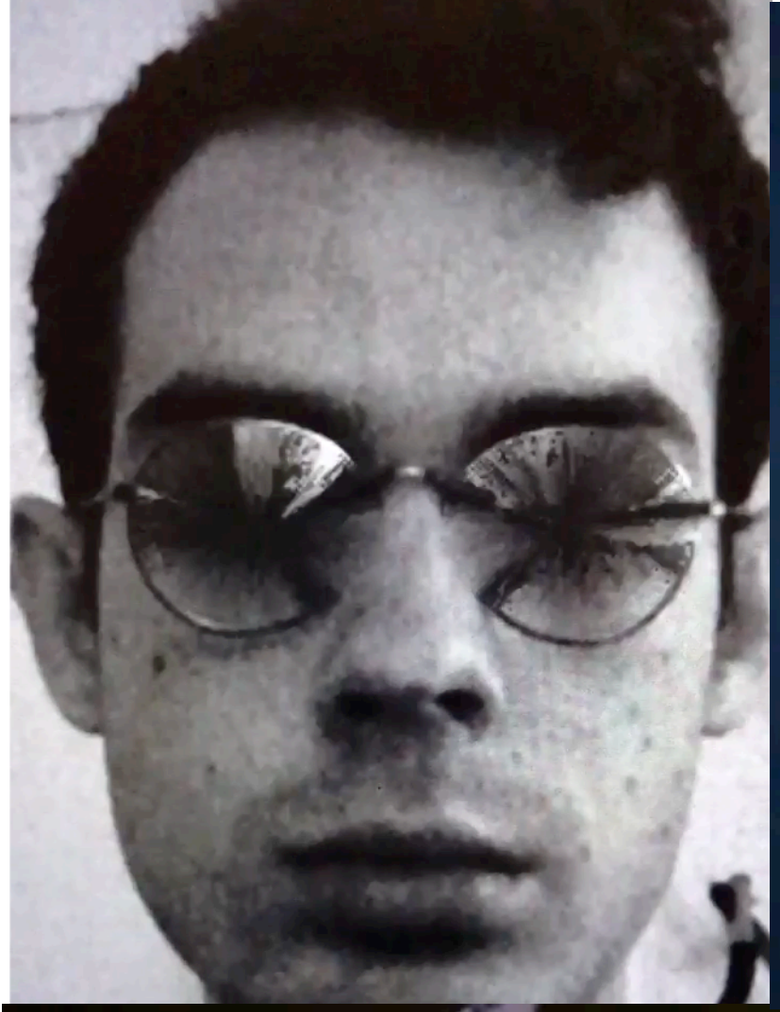
Modules == Code Reuse == Collaboration == Spread of Ideas

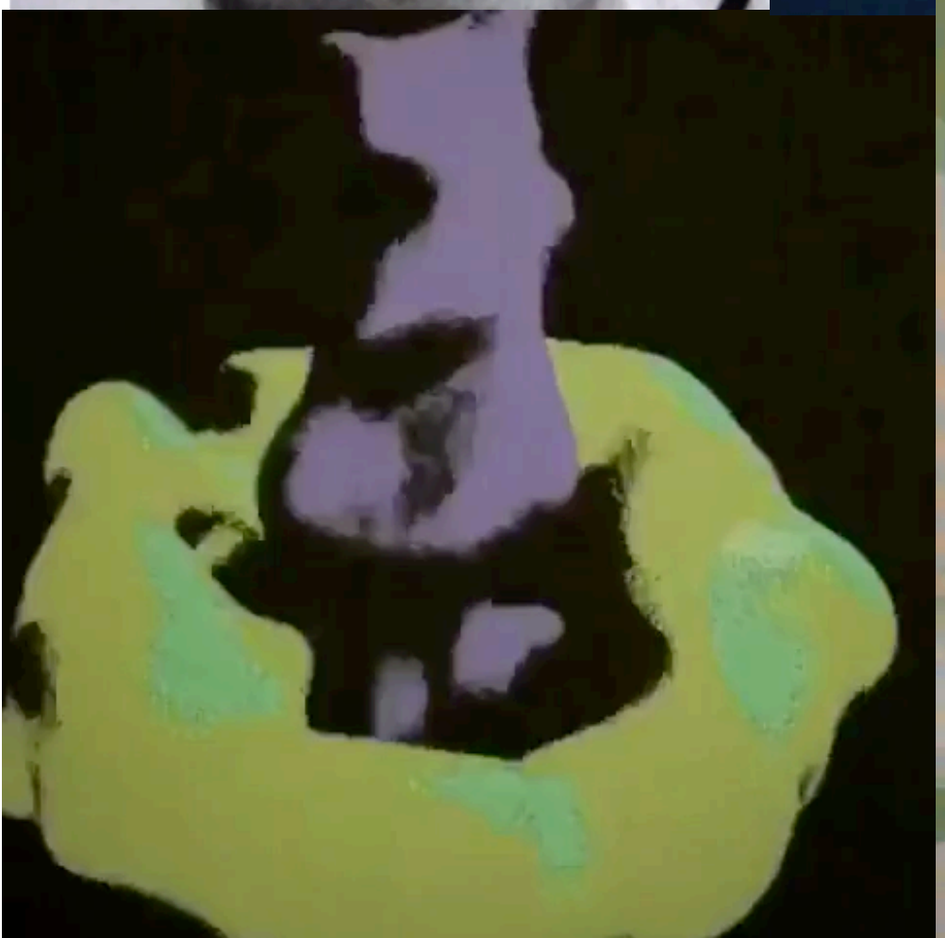


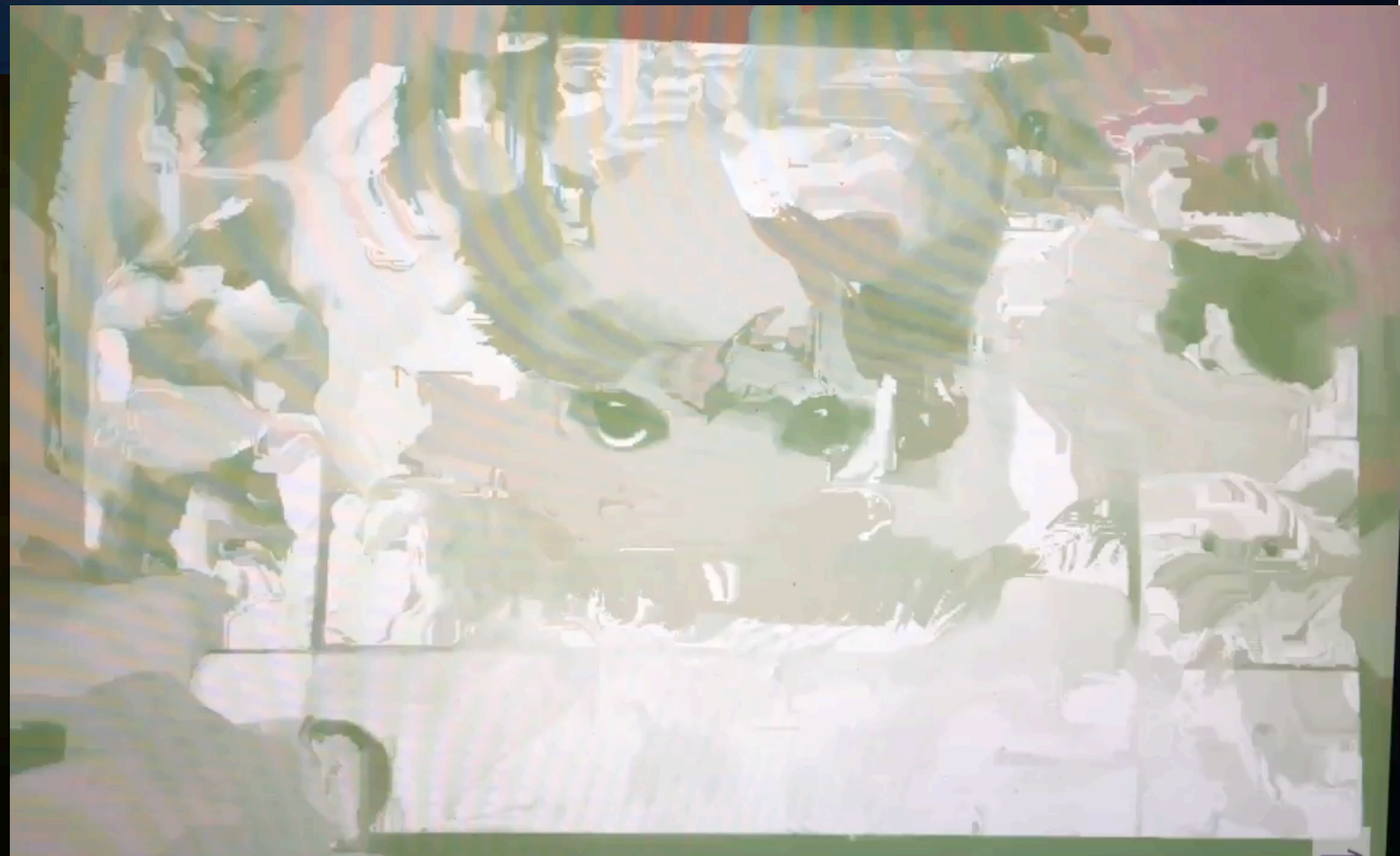
“Javascript Renaissance”

CPU

GPU



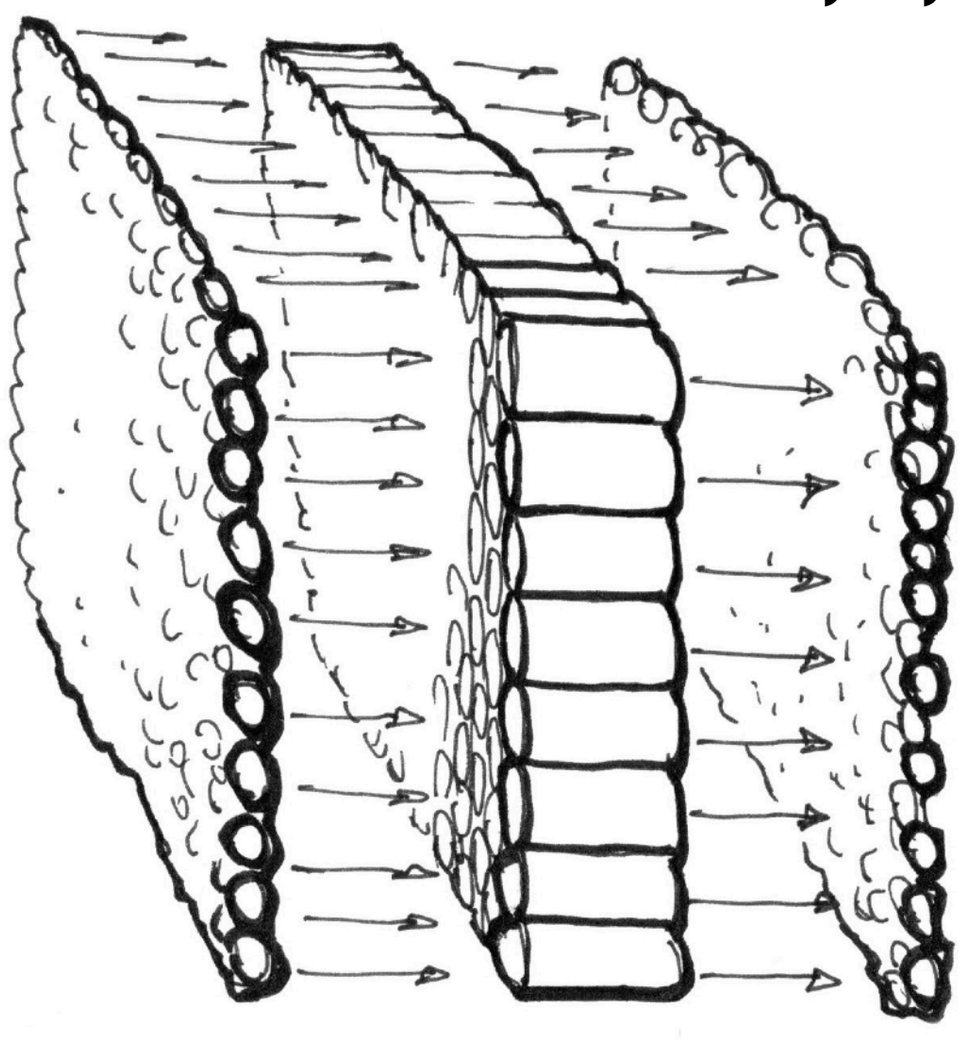




Shader

A function that runs on the GPU, in parallel

Coordinate
X, Y



Color
R,G,B,A

Image Source: The Book of Shaders, Patricio Gonzalez Vivo & Jen Lowe

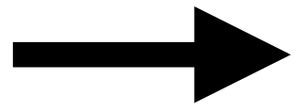
GLSL

Graphics Library Shading Language

Limited, C-like language, compiled to run on the GPU

GLSL

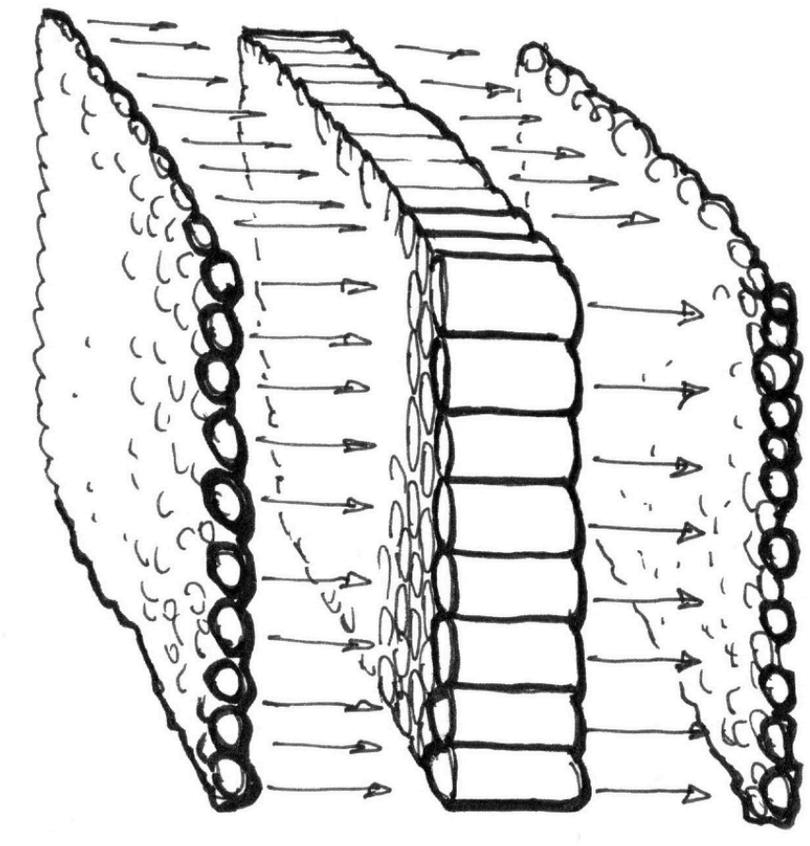
Coordinate
X, Y

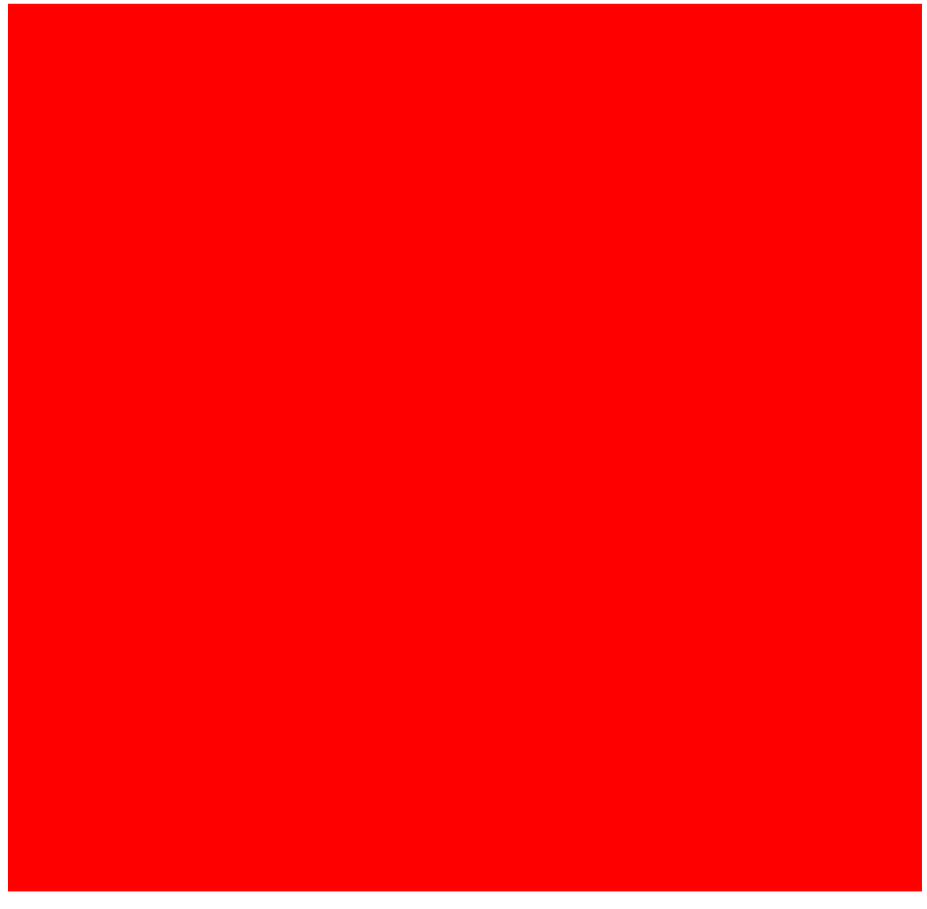


Color
R,G,B,A

```
void main() {
    gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
}
```

Red, Green, Blue, Alpha



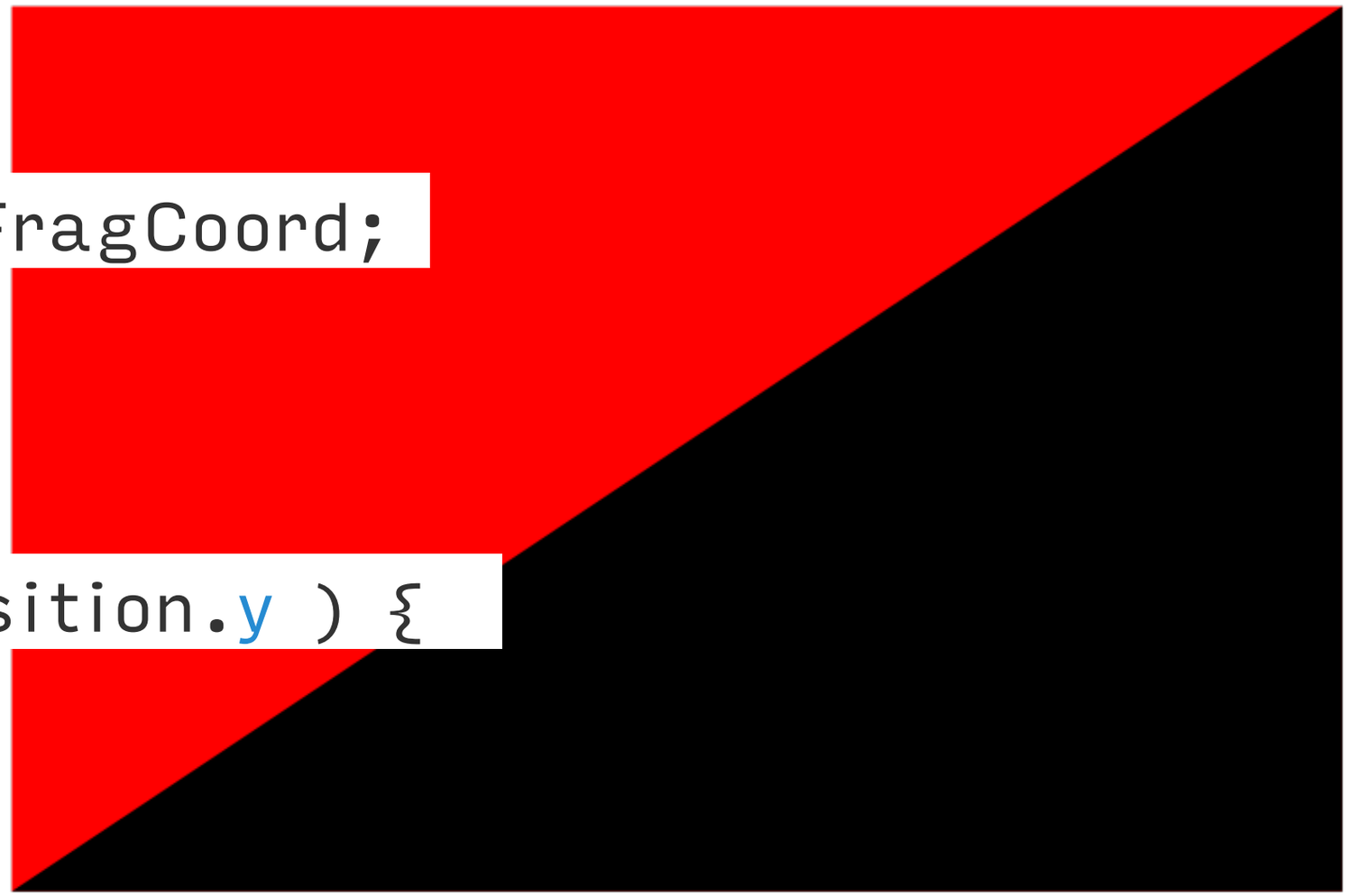


GLSL

Coordinate
X, Y

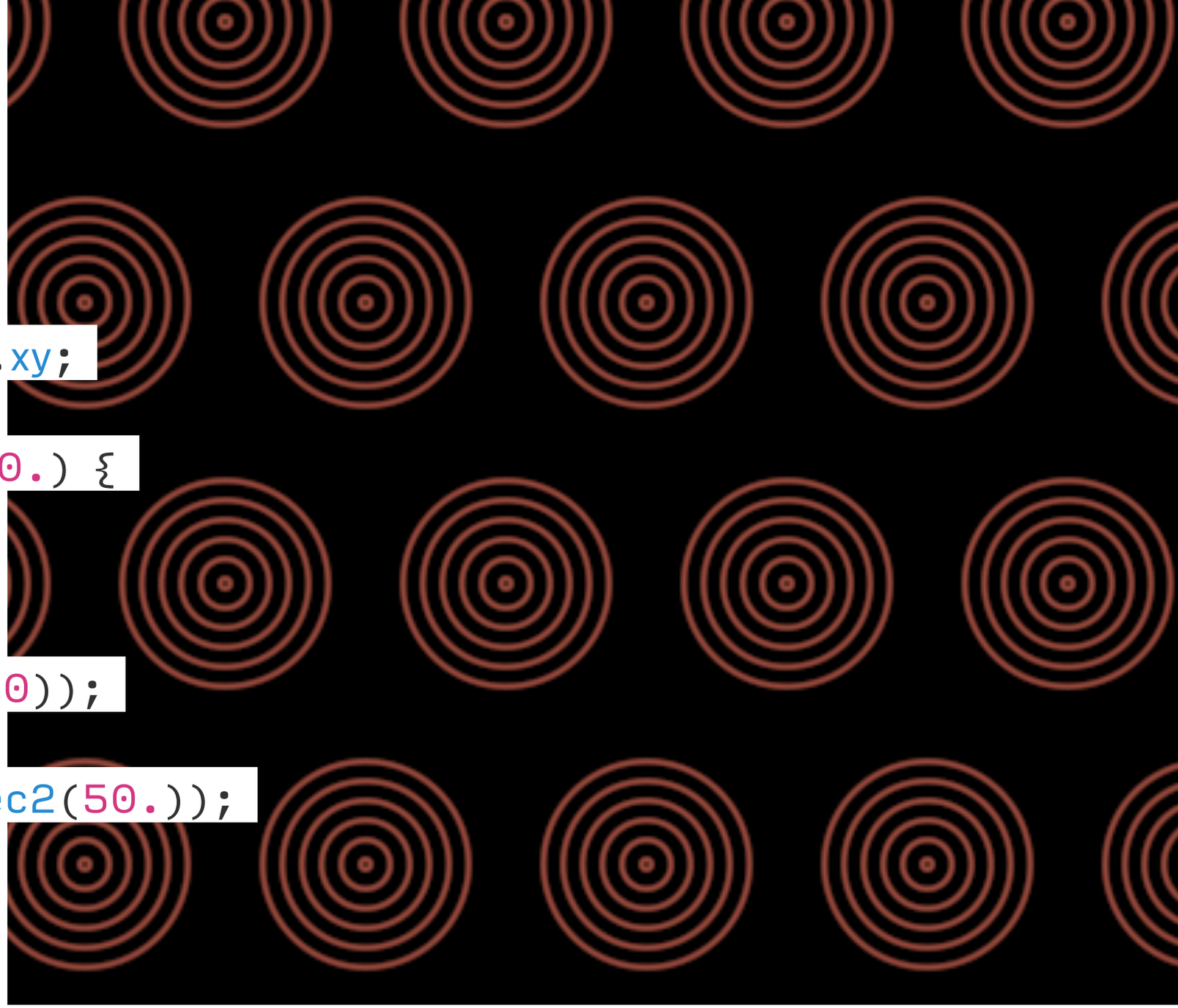
Color
R,G,B,A

```
void main() {
    vec2 position = gl_FragCoord;

    float red = 0.0;

    if ( position.x > position.y ) {
        red = 1.0;
    }

    gl_FragColor = vec4(red, 0.0, 0.0, 1.0);
}
```



GLSL

```
void main() {
   vec2 pos = gl_FragCoord.xy;

  if(mod(pos.y, 200.) > 100.) {
    pos.x += 50.;
  }

  pos = mod(pos, vec2(100.0));

  float d = distance(pos, vec2(50.));

  float n = step(d,40.);
  n = min(n, sin(d*0.9));

  gl_FragColor.rgb = n * vec3(0.61,0.3,0.25);
}
```



// "The Drive Home" by Martijn Steinrucken - 2017

```glsl
#define S(x, y, z) smoothstep(x, y, z)
#define B(a, b, edge, t) S(a-edge, a+edge, t)*S(b+edge, b-edge, t)
#define sat(x) clamp(x,0.,1.)

#define streetLightCol vec3(1., .7, .3)
#define headLightCol vec3(.8, .8, 1.)
#define tailLightCol vec3(1., .1, .1)

#define CAM_SHAKE 1.
#define LANE_BIAS .5

vec3 ro, rd;

float N(float t) {
    return fract(sin(t*10234.324)*123423.23512);
}
vec3 N31(float p) {
    //  3 out, 1 in... DAVE HOSKINS
   vec3 p3 = fract(vec3(p) * vec3(.1031,.11369,.13787));
   p3 += dot(p3, p3.yzx + 19.19);
   return fract(vec3((p3.x + p3.y)*p3.z, (p3.x+p3.z)*p3.y, (p3.y+p3.z)*p3.x));
}
float N2(vec2 p)
{   // Dave Hoskins - https://www.shadertoy.com/view/4djSRW
    vec3 p3  = fract(vec3(p.xyx) * vec3(443.897, 441.423, 437.195));
    p3 += dot(p3, p3.yzx + 19.19);
    return fract((p3.x + p3.y) * p3.z);
}


float DistLine(vec3 ro, vec3 rd, vec3 p) {
    return length(cross(p-ro, rd));
}

vec3 ClosestPoint(vec3 ro, vec3 rd, vec3 p) {
    // returns the closest point on ray r to point p
    return ro + max(0., dot(p-ro, rd))*rd;
}

float Remap(float a, float b, float c, float d, float t) {
    return ((t-a)/(b-a))*(d-c)+c;
}

float BokehMask(vec3 ro, vec3 rd, vec3 p, float size, float blur) {
    float d = DistLine(ro, rd, p);
    float m = S(size, size*(1.-blur), d);

    return m;
}
```

```glsl
float SawTooth(float t) {
    return cos(t+cos(t))+sin(2.*t)*.2+sin(4.*t)*.02;
}

float DeltaSawTooth(float t) {
    return 0.4*cos(2.*t)+0.08*cos(4.*t) - (1.-sin(t))*sin(t+cos(t));
}

vec2 GetDrops(vec2 uv, float seed, float m) {
    float t = iTime+m*30.;
    vec2 o = vec2(0.);
    uv *= vec2(10., 2.5)*2.;
    vec2 id = floor(uv);
    vec3 n = N31(id.x + (id.y+seed)*546.3524);
    vec2 bd = fract(uv);

    vec2 uv2 = bd;
    bd -= .5;
    bd.y*=4.;

    bd.x += (n.x-.5)*.6;
    t += n.z * 6.28;
    float slide = SawTooth(t);
    float ts = 1.5;
    vec2 trailPos = vec2(bd.x*ts, (fract(bd.y*ts*2.-t*2.)-.5)*.5);

    bd.y += slide*2.// make drops slide down
    float d = length(bd);  // distance to main drop

    float trailMask = S(-.2, .2, bd.y); // mask out drops that are below the main
    trailMask *= bd.y; // fade dropsize
    float td = length(trailPos*max(.5, trailMask)); // distance to trail drops

    float mainDrop = S(.2, .1, d);
    float dropTrail = S(.1, .02, td);

    dropTrail *= trailMask;
    o = mix(bd*mainDrop, trailPos, dropTrail);      // mix main drop and drop trail

    return o;
}

void CameraSetup(vec2 uv, vec3 pos, vec3 lookat, float zoom, float m) {
    ro = pos;
    vec3 f = normalize(lookat-ro);
    vec3 r = cross(vec3(0., 1., 0.), f);
    vec3 u = cross(f, r);
    float t = iTime;

    vec2 offs = vec2(0.);

    vec2 dropUv = uv;
```

```glsl
    float x = (sin(t*.1)*.5+.5)*.5;
    x = -x*x;
    float s = sin(x);
    float c = cos(x);

    mat2 rot = mat2(c, -s, s, c);

    vec3 center = ro + f*zoom;
    vec3 i = center + (uv.x-offs.x)*r + (uv.y-offs.y)*u;

    rd = normalize(i-ro);
}

vec3 HeadLights(float i, float t) {
    float z = fract(-t*2.+i);
    vec3 p = vec3(-.3, .1, z*40.);
    float d = length(p-ro);

    float size = mix(.03, .05, S(.02, .07, z))*d;
    float m = 0.;
    float blur = .1;
    m += BokehMask(ro, rd, p-vec3(.08, 0., 0.), size, blur);
    m += BokehMask(ro, rd, p+vec3(.08, 0., 0.), size, blur);

    float distFade = max(.01, pow(1.-z, 9.));
   blur = .8;
    size *= 2.5;
    float r = 0.;
    r += BokehMask(ro, rd, p+vec3(-.09, -.2, 0.), size, blur);
    r += BokehMask(ro, rd, p+vec3(.09, -.2, 0.), size, blur);
    r *= distFade*distFade;
   return headLightCol*(m+r)*distFade;
}


vec3 TailLights(float i, float t) {
    t = t*1.5+i;
   float id = floor(t)+i;
    vec3 n = N31(id);
   float laneId = S(LANE_BIAS, LANE_BIAS+.01, n.y);

    float ft = fract(t);
    float z = 3.-ft*3.;  // distance ahead

    laneId *= S(.2, 1.5, z);   // get out of the way!
    float lane = mix(.6, .3, laneId);
    vec3 p = vec3(lane, .1, z);
    float d = length(p-ro);

    float size = .05*d;
    float blur = .1;
    float m = BokehMask(ro, rd, p-vec3(.08, 0., 0.), size, blur) +
              BokehMask(ro, rd, p+vec3(.08, 0., 0.), size, blur);
```

```glsl
    float refSize = size*2.5;
    m += BokehMask(ro, rd, p+vec3(-.09, -.2, 0.), refSize, .8);
    m += BokehMask(ro, rd, p+vec3(.09, -.2, 0.), refSize, .8);
    vec3 col = tailLightCol*m*ft;

    float b = BokehMask(ro, rd, p+vec3(.12, 0., 0.), size, blur);
    b += BokehMask(ro, rd, p+vec3(.12, -.2, 0.), refSize, .8)*.2;

    vec3 blinker = vec3(1., .7, .2);
    blinker *= S(1.5, 1.4, z)*S(.2, .3, z);
    blinker *= sat(sin(t*200.)*100.);
    blinker *= laneId;
    col += blinker*b;
    return col;
}

vec3 StreetLights(float i, float t) {
     float side = sign(rd.x);
    float offset = max(side, 0.)*(1./16.);
    float z = fract(i-t+offset);
    vec3 p = vec3(2.*side, 2., z*60.);
    float d = length(p-ro);
    float blur = .1;
    vec3 rp = ClosestPoint(ro, rd, p);
    float distFade = Remap(1., .7, .1, 1.5, 1.-pow(1.-z,6.));
    distFade *= (1.-z);
    float m = BokehMask(ro, rd, p, .05*d, blur)*distFade;
    return m*streetLightCol;
}

vec3 EnvironmentLights(float i, float t) {
    float n = N(i+floor(t));

    float side = sign(rd.x);
    float offset = max(side, 0.)*(1./16.);
    float z = fract(i-t+offset+fract(n*234.));
    float n2 = fract(n*100.);
    vec3 p = vec3((3.+n)*side, n2*n2*n2*1., z*60.);
    float d = length(p-ro);
    float blur = .1;
    vec3 rp = ClosestPoint(ro, rd, p);
    float distFade = Remap(1., .7, .1, 1.5, 1.-pow(1.-z,6.));
    float m = BokehMask(ro, rd, p, .05*d, blur);
    m *= distFade*distFade*.5;

    m *= 1.-pow(sin(z*6.28*20.*n)*.5+.5, 20.);
    vec3 randomCol = vec3(fract(n*-34.5), fract(n*4572.), fract(n*1264.));
    vec3 col = mix(tailLightCol, streetLightCol, fract(n*-65.42));
    col = mix(col, randomCol, n);
    return m*col*.2;
}

void mainImage( out vec4 fragColor, in vec2 fragCoord )
{
    float t = iTime;
    vec3 col = vec3(0.);
    vec2 uv = fragCoord.xy / iResolution.xy; // 0 <> 1

    uv -= .5;
    uv.x *= iResolution.x/iResolution.y;

    vec2 mouse = iMouse.xy/iResolution.xy;

    vec3 pos = vec3(.3, .15, 0.);

    float bt = t * 5.;
    float h1 = N(floor(bt));
    float h2 = N(floor(bt+1.));
    float bumps = mix(h1, h2, fract(bt))*.1;
    bumps = bumps*bumps*bumps*CAM_SHAKE;

    pos.y += bumps;
    float lookatY = pos.y+bumps;
    vec3 lookat = vec3(0.3, lookatY, 1.);
    vec3 lookat2 = vec3(0., lookatY, .7);
    lookat = mix(lookat, lookat2, sin(t*.1)*.5+.5);

    uv.y += bumps*4.;
    CameraSetup(uv, pos, lookat, 2., mouse.x);

    t *= .03;
    t += mouse.x;

    for(float i=0.; i<1.; i+=stp) {
       col += StreetLights(i, t);
    }

    for(float i=0.; i<1.; i+=stp) {
       float n = N(i+floor(t));
       col += HeadLights(i+n*stp*.7, t);
    }

    for(float i=0.; i<1.; i+=stp) {
       col += EnvironmentLights(i, t);
    }

    col += TailLights(0., t);
    col += TailLights(.5, t);

    col += sat(rd.y)*vec3(.6, .5, .9);

    fragColor = vec4(col, 0.);
}
```

// "The Drive Home" by Martijn Steinrucken - 2017

```
#define S(x, y, z) smoothstep(x, y, z)
#define B(a, b, edge, t) S(a-edge, a+edge, t)*S(b+edge, b-edge, t)
#define sat(x) clamp(x,0.,1.)

#define streetLightCol vec3(1., .7, .3)
#define headLightCol vec3(.8, .8, 1.)
#define tailLightCol vec3(1., .1, .1)

#define CAM_SHAKE 1.
#define LANE_BIAS .5

vec3 ro, rd;
```

```
float N(float t) {
    return fract(sin(t*10234.324)*123423.23512);
}
```

```
vec3 N31(float p) {
    //  3 out, 1 in... DAVE HOSKINS
   vec3 p3 = fract(vec3(p) * vec3(.1031,.11369,.13787));
   p3 += dot(p3, p3.yzx + 19.19);
   return fract(vec3((p3.x + p3.y)*p3.z, (p3.x+p3.z)*p3.y, (p3.y+p3.z)*p3.x));
}
```

```
float N2(vec2 p)
{   // Dave Hoskins - https://www.shadertoy.com/view/4djSRW
    vec3 p3  = fract(vec3(p.xyx) * vec3(443.897, 441.423, 437.195));
    p3 += dot(p3, p3.yzx + 19.19);
    return fract((p3.x + p3.y) * p3.z);
}
```

```
float DistLine(vec3 ro, vec3 rd, vec3 p) {
    return length(cross(p-ro, rd));
}
```

```
vec3 ClosestPoint(vec3 ro, vec3 rd, vec3 p) {
    // returns the closest point on ray r to point p
    return ro + max(0., dot(p-ro, rd))*rd;
}
```

```
float Remap(float a, float b, float c, float d, float t) {
    return ((t-a)/(b-a))*(d-c)+c;
}
```

```
float BokehMask(vec3 ro, vec3 rd, vec3 p, float size, float blur) {
    float d = DistLine(ro, rd, p);
    float m = S(size, size*(1.-blur), d);

    return m;
}
```

```
float SawTooth(float t) {
    return cos(t+cos(t))+sin(2.*t)*.2+sin(4.*t)*.02;
}
```

```
float DeltaSawTooth(float t) {
    return 0.4*cos(2.*t)+0.08*cos(4.*t) - (1.-sin(t))*sin(t+cos(t));
}
```

```
vec2 GetDrops(vec2 uv, float seed, float m) {
    float t = iTime+m*30.;
    vec2 o = vec2(0.);
    uv *= vec2(10., 2.5)*2.;
    vec2 id = floor(uv);
    vec3 n = N31(id.x + (id.y+seed)*546.3524);
    vec2 bd = fract(uv);

    vec2 uv2 = bd;
    bd -= .5;
    bd.y*=4.;

    bd.x += (n.x-.5)*.6;
    t += n.z * 6.28;
    float slide = SawTooth(t);
    float ts = 1.5;
    vec2 trailPos = vec2(bd.x*ts, (fract(bd.y*ts*2.-t*2.)-.5)*.5);

    bd.y += slide*2.// make drops slide down
    float d = length(bd);  // distance to main drop

    float trailMask = S(-.2, .2, bd.y); // mask out drops that are below the main
    trailMask *= bd.y; // fade dropsize
    float td = length(trailPos*max(.5, trailMask)); // distance to trail drops

    float mainDrop = S(.2, .1, d);
    float dropTrail = S(.1, .02, td);

    dropTrail *= trailMask;
    o = mix(bd*mainDrop, trailPos, dropTrail);      // mix main drop and drop trail

    return o;
}
```

```
void CameraSetup(vec2 uv, vec3 pos, vec3 lookat, float zoom, float m) {
    ro = pos;
    vec3 f = normalize(lookat-ro);
    vec3 r = cross(vec3(0., 1., 0.), f);
    vec3 u = cross(f, r);
    float t = iTime;

    vec2 offs = vec2(0.);

    vec2 dropUv = uv;
```

```
    float x = (sin(t*.1)*.5+.5)*.5;
    x = -x*x;
    float s = sin(x);
    float c = cos(x);

    mat2 rot = mat2(c, -s, s, c);

    vec3 center = ro + f*zoom;
    vec3 i = center + (uv.x-offs.x)*r + (uv.y-offs.y)*u;

    rd = normalize(i-ro);
}

vec3 HeadLights(float i, float t) {
    float z = fract(-t*2.+i);
    vec3 p = vec3(-.3, .1, z*40.);
    float d = length(p-ro);

    float size = mix(.03, .05, S(.02, .07, z))*d;
    float m = 0.;
    float blur = .1;
    m += BokehMask(ro, rd, p-vec3(.08, 0., 0.), size, blur);
    m += BokehMask(ro, rd, p+vec3(.08, 0., 0.), size, blur);

    float distFade = max(.01, pow(1.-z, 9.));
   blur = .8;
    size *= 2.5;
    float r = 0.;
    r += BokehMask(ro, rd, p+vec3(-.09, -.2, 0.), size, blur);
    r += BokehMask(ro, rd, p+vec3(.09, -.2, 0.), size, blur);
    r *= distFade*distFade;

    return headLightCol*(m+r)*distFade;
}


vec3 TailLights(float i, float t) {
    t = t*1.5+i;
   float id = floor(t)+i;
    vec3 n = N31(id);
   float laneId = S(LANE_BIAS, LANE_BIAS+.01, n.y);

    float ft = fract(t);
    float z = 3.-ft*3.;  // distance ahead

    laneId *= S(.2, 1.5, z); // get out of the way!
    float lane = mix(.6, .3, laneId);
    vec3 p = vec3(lane, .1, z);
    float d = length(p-ro);

    float size = .05*d;
    float blur = .1;
    float m = BokehMask(ro, rd, p-vec3(.08, 0., 0.), size, blur) +
              BokehMask(ro, rd, p+vec3(.08, 0., 0.), size, blur);
```

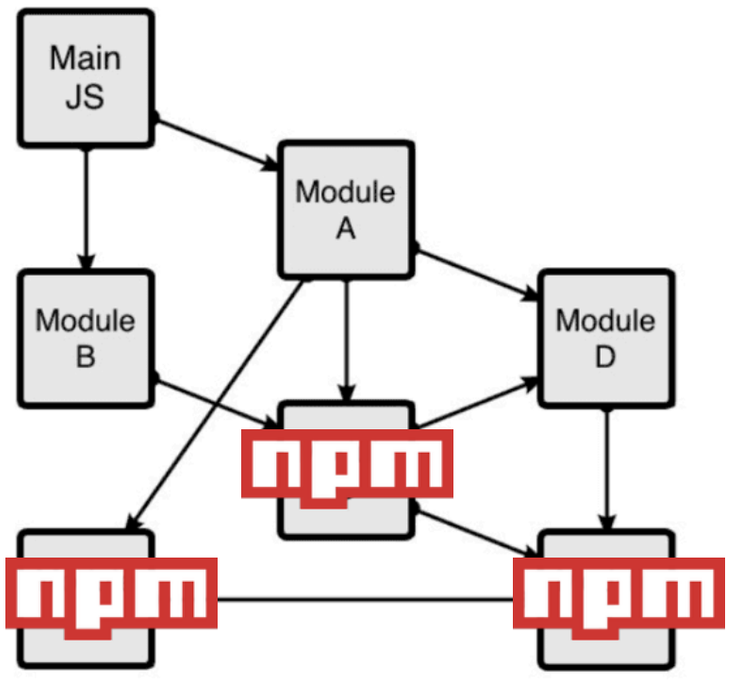
```
    float refSize = size*2.5;
    m += BokehMask(ro, rd, p+vec3(-.09, -.2, 0.), refSize, .8);
    m += BokehMask(ro, rd, p+vec3(.09, -.2, 0.), refSize, .8);
    vec3 col = tailLightCol*m*ft;

    float b = BokehMask(ro, rd, p+vec3(.12, 0., 0.), size, blur);
    b += BokehMask(ro, rd, p+vec3(.12, -.2, 0.), refSize, .8)*.2;

    vec3 blinker = vec3(1., .7, .2);
    blinker *= S(1.5, 1.4, z)*S(.2, .3, z);
    blinker *= sat(sin(t*200.)*100.);
    blinker *= laneId;
    col += blinker*b;
    return col;
}

vec3 StreetLights(float i, float t) {
     float side = sign(rd.x);
    float offset = max(side, 0.)*(1./16.);
    float z = fract(i-t+offset);
    vec3 p = vec3(2.*side, 2., z*60.);
    float d = length(p-ro);
    float blur = .1;
    vec3 rp = ClosestPoint(ro, rd, p);
    float distFade = Remap(1., .7, .1, 1.5, 1.-pow(1.-z,6.));
    distFade *= (1.-z);
    float m = BokehMask(ro, rd, p, .05*d, blur)*distFade;
    return m*streetLightCol;
}

vec3 EnvironmentLights(float i, float t) {
    float n = N(i+floor(t));

    float side = sign(rd.x);
    float offset = max(side, 0.)*(1./16.);
    float z = fract(i-t+offset+fract(n*234.));
    float n2 = fract(n*100.);
    vec3 p = vec3((3.+n)*side, n2*n2*n2*1., z*60.);
    float d = length(p-ro);
    float blur = .1;
    vec3 rp = ClosestPoint(ro, rd, p);
    float distFade = Remap(1., .7, .1, 1.5, 1.-pow(1.-z,6.));
    float m = BokehMask(ro, rd, p, .05*d, blur);
    m *= distFade*distFade*.5;

    m *= 1.-pow(sin(z*6.28*20.*n)*.5+.5, 20.);
    vec3 randomCol = vec3(fract(n*-34.5), fract(n*4572.), fract(n*1264.));
    vec3 col = mix(tailLightCol, streetLightCol, fract(n*-65.42));
    col = mix(col, randomCol, n);
    return m*col*.2;
}

void mainImage( out vec4 fragColor, in vec2 fragCoord )
{
    float t = iTime;
    vec3 col = vec3(0.);
    vec2 uv = fragCoord.xy / iResolution.xy; // 0 <> 1

    uv -= .5;
    uv.x *= iResolution.x/iResolution.y;

    vec2 mouse = iMouse.xy/iResolution.xy;

    vec3 pos = vec3(.3, .15, 0.);

    float bt = t * 5.;
    float h1 = N(floor(bt));
    float h2 = N(floor(bt+1.));
    float bumps = mix(h1, h2, fract(bt))*.1;
    bumps = bumps*bumps*bumps*CAM_SHAKE;

    pos.y += bumps;
    float lookatY = pos.y+bumps;
    vec3 lookat = vec3(0.3, lookatY, 1.);
    vec3 lookat2 = vec3(0., lookatY, .7);
    lookat = mix(lookat, lookat2, sin(t*.1)*.5+.5);

    uv.y += bumps*4.;
    CameraSetup(uv, pos, lookat, 2., mouse.x);

    t *= .03;
    t += mouse.x;

    for(float i=0.; i<1.; i+=stp) {
       col += StreetLights(i, t);
    }

    for(float i=0.; i<1.; i+=stp) {
        float n = N(i+floor(t));
        col += HeadLights(i+n*stp*.7, t);
    }

    for(float i=0.; i<1.; i+=stp) {
       col += EnvironmentLights(i, t);
    }

    col += TailLights(0., t);
    col += TailLights(.5, t);

    col += sat(rd.y)*vec3(.6, .5, .9);

    fragColor = vec4(col, 0.);
}
```

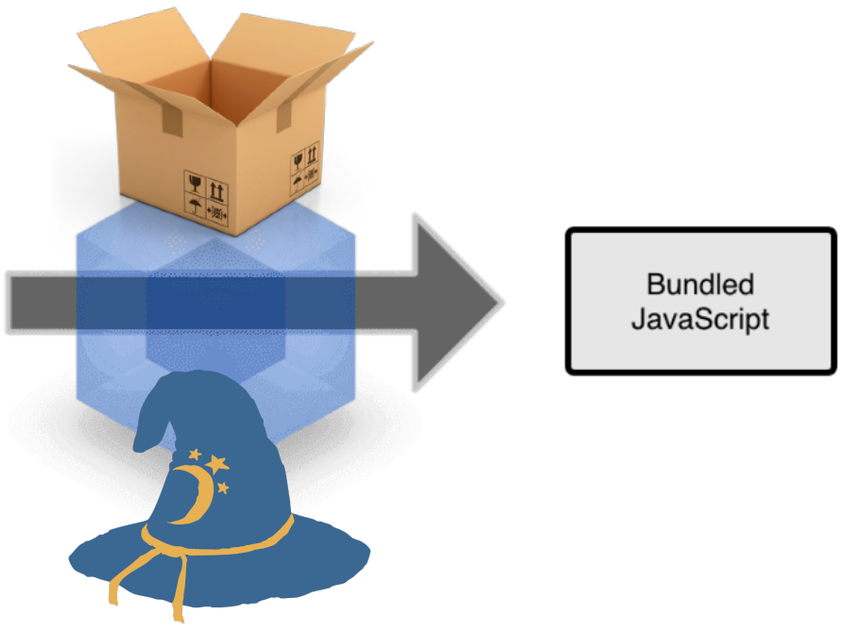
// "The Drive Home" by Martijn Steinrucken - 2017

browserify : JS

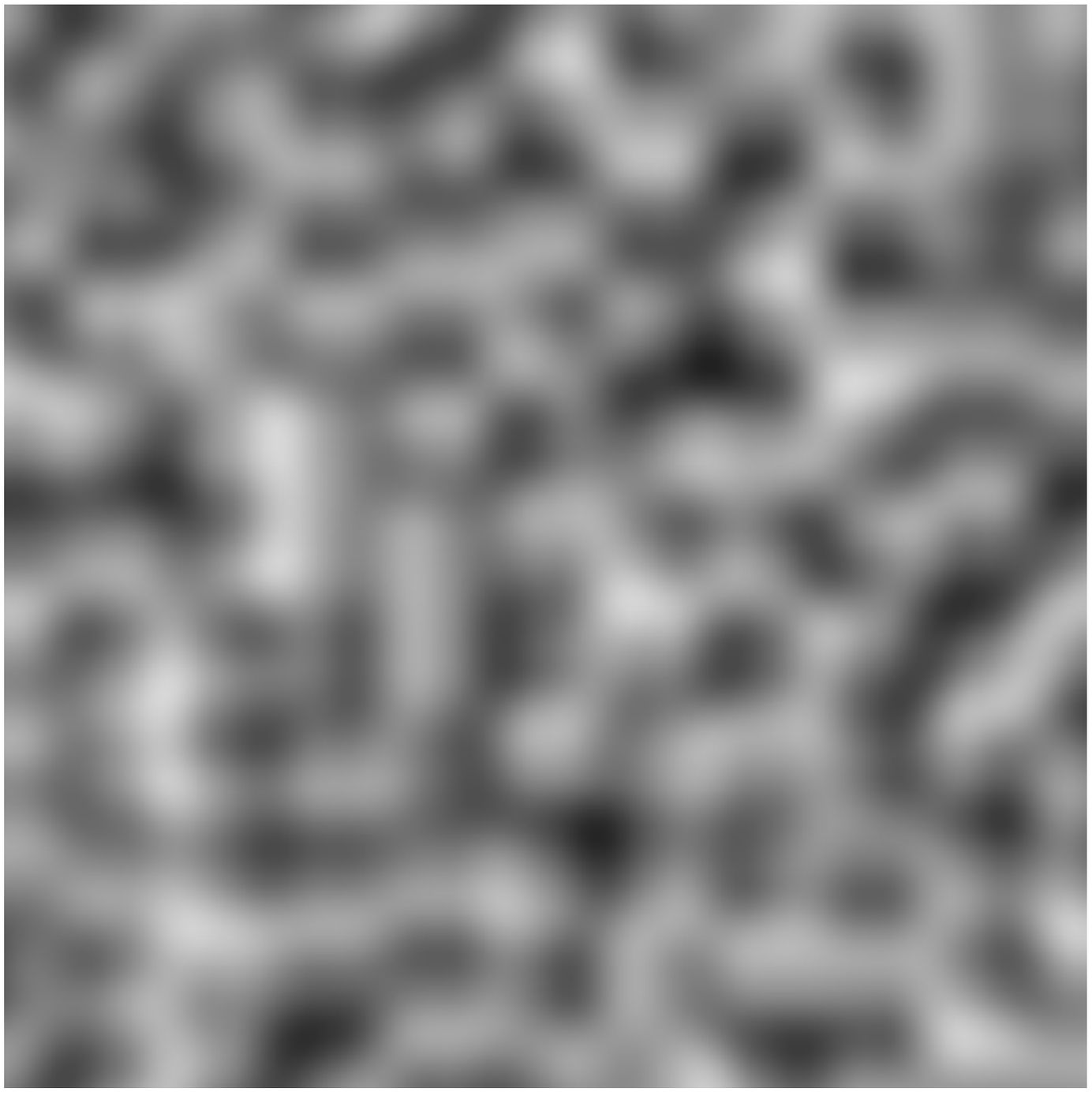
::

glslify : GLSL

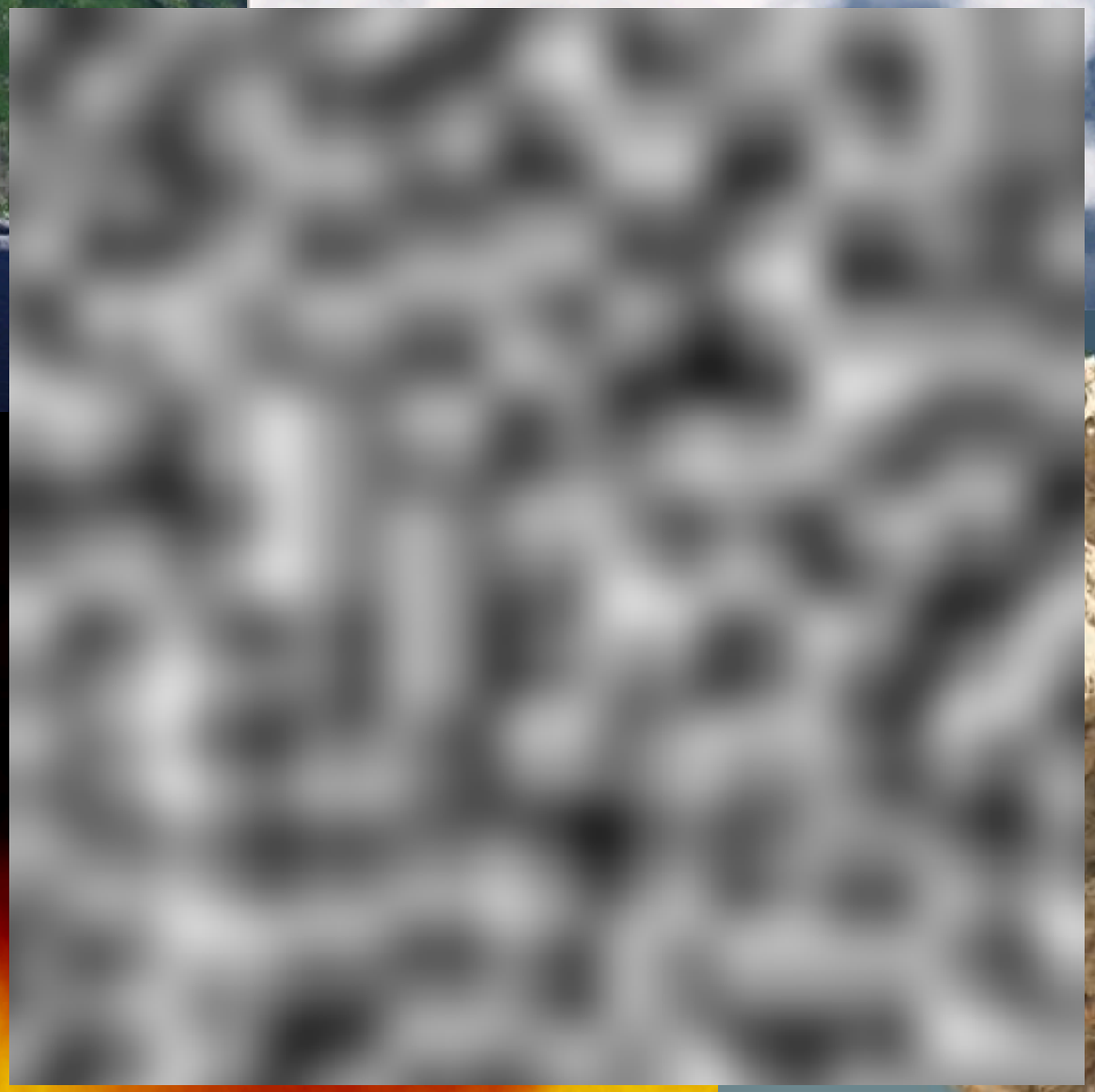


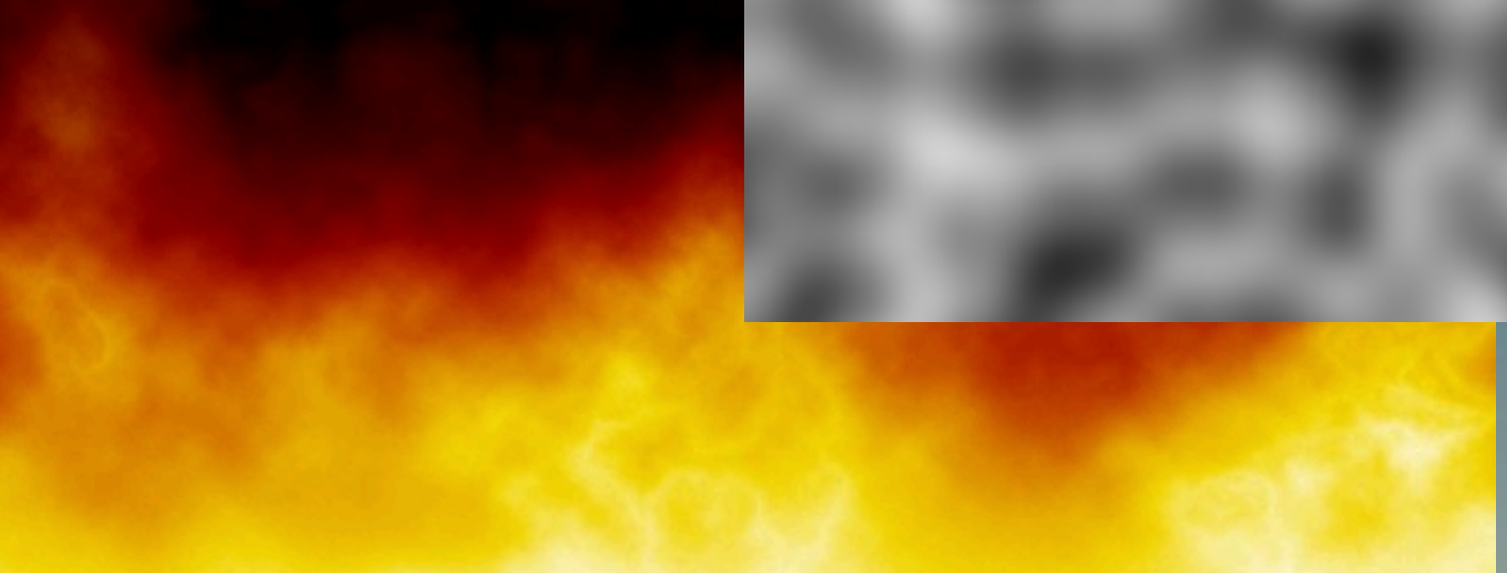


Perlin Noise



Perlin Noise



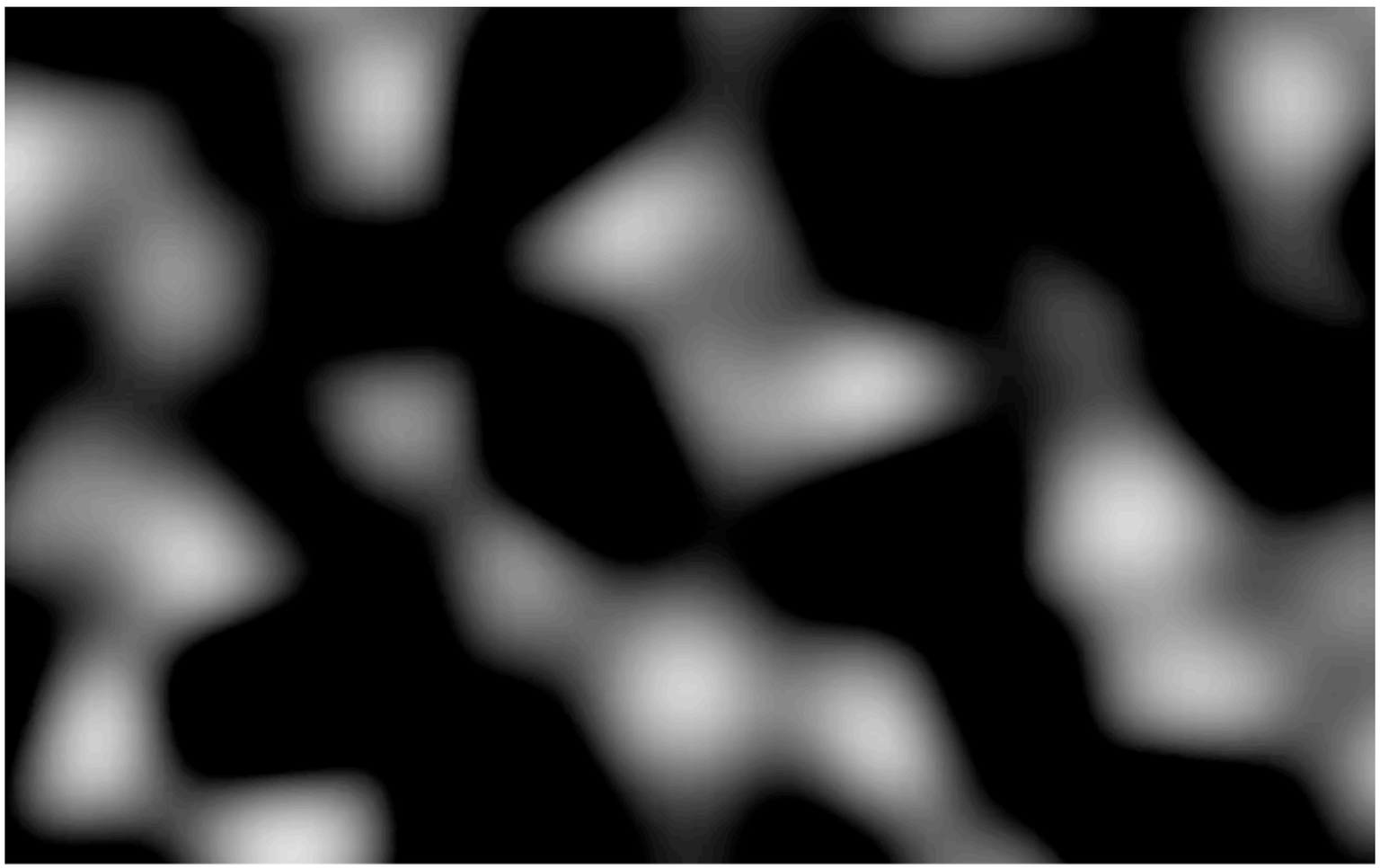


Perlin Noise

GLSLIFY

>npm install glsl-noise

```
#pragma glslify: noise = require("glsl-noise/simplex/3d")

void main() {
  float n = noise(vec3(gl_FragCoord.xy * 0.005, time));
  gl_FragColor.rgba = vec4(n);
}
```



```
#define GLSLIFY 1
precision mediump float;

uniform vec3  iResolution;

vec3 mod289_1540259130(vec3 x) {
  return x - floor(x * (1.0 / 289.0)) * 289.0;
}

vec2 mod289_1540259130(vec2 x) {
  return x - floor(x * (1.0 / 289.0)) * 289.0;
}

vec3 permute_1540259130(vec3 x) {
  return mod289_1540259130(((x*34.0)+1.0)*x);
}

float snoise_1540259130(vec2 v)
  {
  const vec4 C = vec4(0.211324865405187,
                      0.366025403784439,
                     -0.577350269189626,
                      0.024390243902439);
// First corner
  vec2 i  = floor(v + dot(v, C.yy) );
  vec2 x0 = v -   i + dot(i, C.xx);

// Other corners
  vec2 i1;
  //i1.x = step( x0.y, x0.x ); // x0.x > x0.y ? 1.0 : 0.0
  //i1.y = 1.0 - i1.x;
  i1 = (x0.x > x0.y) ? vec2(1.0, 0.0) : vec2(0.0, 1.0);
  // x0 = x0 - 0.0 + 0.0 * C.xx ;
  // x1 = x0 - i1 + 1.0 * C.xx ;
  // x2 = x0 - 1.0 + 2.0 * C.xx ;
  vec4 x12 = x0.xyxy + C.xxzz;
  x12.xy -= i1;

// Permutations
  i = mod289_1540259130(i);

 vec3 p = permute_1540259130( permute_1540259130( i.y
+ vec3(0.0, i1.y, 1.0 ))
    + i.x + vec3(0.0, i1.x, 1.0 ));

  vec3 m = max(0.5 - vec3(dot(x0,x0), dot(x12.xy,x12.xy),
dot(x12.zw,x12.zw)), 0.0);
  m = m*m ;
  m = m*m ;

// Gradients: 41 points uniformly over a line, mapped onto a
diamond.
// The ring size 17*17 = 289 is close to a multiple of 41
(41*7 = 287)

  vec3 x = 2.0 * fract(p * C.www) - 1.0;
  vec3 h = abs(x) - 0.5;
  vec3 ox = floor(x + 0.5);
  vec3 a0 = x - ox;

// Normalise gradients implicitly by scaling m
// Approximation of: m *= inversesqrt( a0*a0 + h*h );
  m *= 1.79284291400159 - 0.85373472095314 * ( a0*a0
+ h*h );

// Compute final noise value at P
  vec3 g;
  g.x  = a0.x  * x0.x  + h.x  * x0.y;
  g.yz = a0.yz * x12.xz + h.yz * x12.yw;
  return 130.0 * dot(m, g);
}

void main() {
  float n = snoise_1540259130(gl_FragCoord.xy * .005);
  gl_FragColor.rgba = vec4(n);
}
```

Our Actual Source Code

Leveraging libraries isn't just a convenience

An ecosystem is a guide and a tutor

Thank you!

Learn to write GLSL online:

“The Book Of Shaders”

@maxbittker

hughsk / glsl-noise

Watch 4 | Unstar 73 | Fork 7

Code | Issues 1 | Pull requests 0 | Projects 0 | Wiki | Insights

webgl-noise shaders ported to work with glslify

3 commits | 1 branch | 0 releases | 2 contributors | MIT

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

hughsk Merge pull request #2 from wwwtyro/patch-1 ... Latest commit 7870430 on Dec 26, 2015

classic	first commit	5 years ago
periodic	first commit	5 years ago
simplex	first commit	5 years ago
LICENSE	first commit	5 years ago
README.md	Update README.md	2 years ago
package.json	first commit	5 years ago

README.md

glsl-noise stability frozen

webgl-noise ported to an NPM package so that you can require it from glslify.

npm install glsl-noise

Usage

```
// Require as many or as little as you need:
#pragma glslify: snoise2 = require(glsl-noise/simplex/2d)
#pragma glslify: snoise3 = require(glsl-noise/simplex/3d)
#pragma glslify: snoise4 = require(glsl-noise/simplex/4d)
#pragma glslify: cnoise2 = require(glsl-noise/classic/2d)
#pragma glslify: cnoise3 = require(glsl-noise/classic/3d)
#pragma glslify: cnoise4 = require(glsl-noise/classic/4d)
#pragma glslify: pnoise2 = require(glsl-noise/periodic/2d)
#pragma glslify: pnoise3 = require(glsl-noise/periodic/3d)
#pragma glslify: pnoise4 = require(glsl-noise/periodic/4d)

attribute vec3 position;

// And just treat them as functions like
// you normally would:
void main() {
  gl_FragColor = vec4(snoise3(position), 1.0);
}
```

Pull Requests!

Versions!

Readmes!

Examples!

Tests!

Forking!

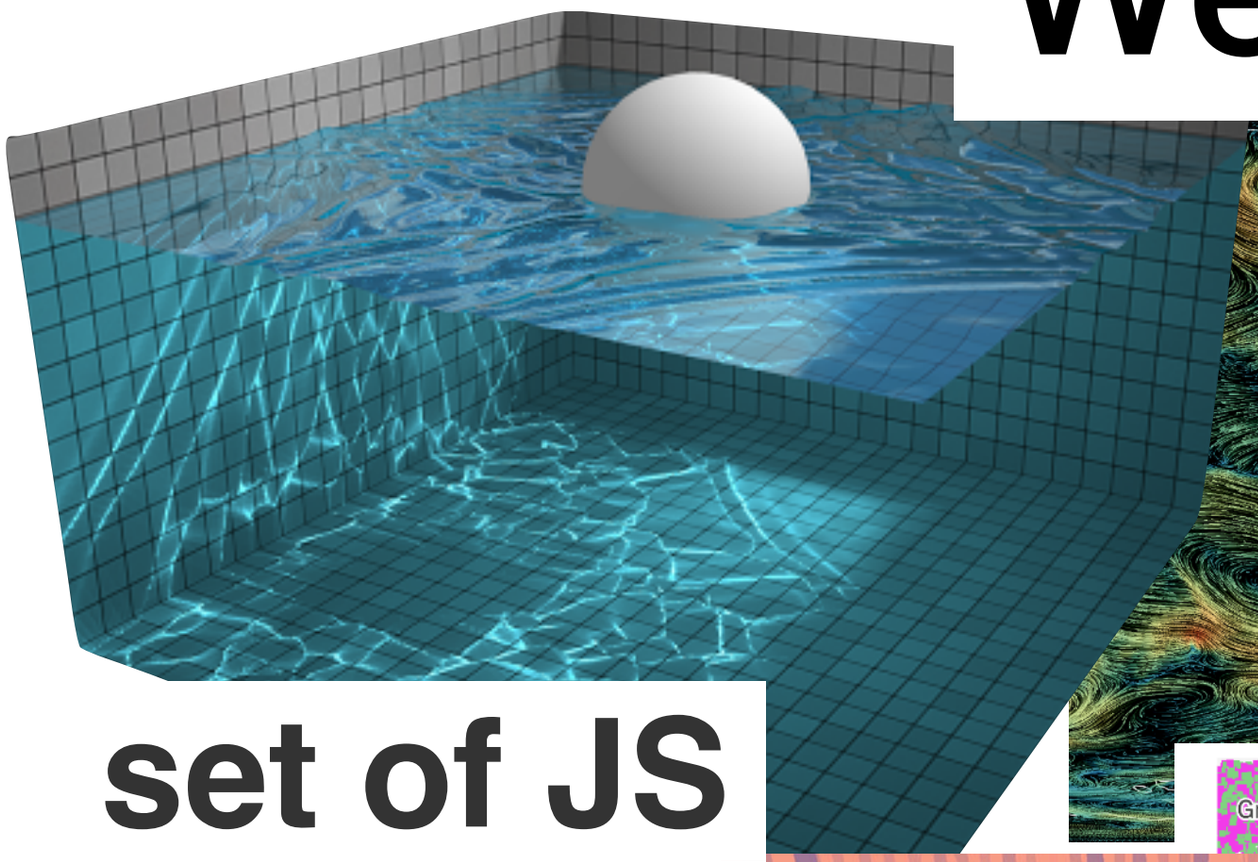
Licenses!

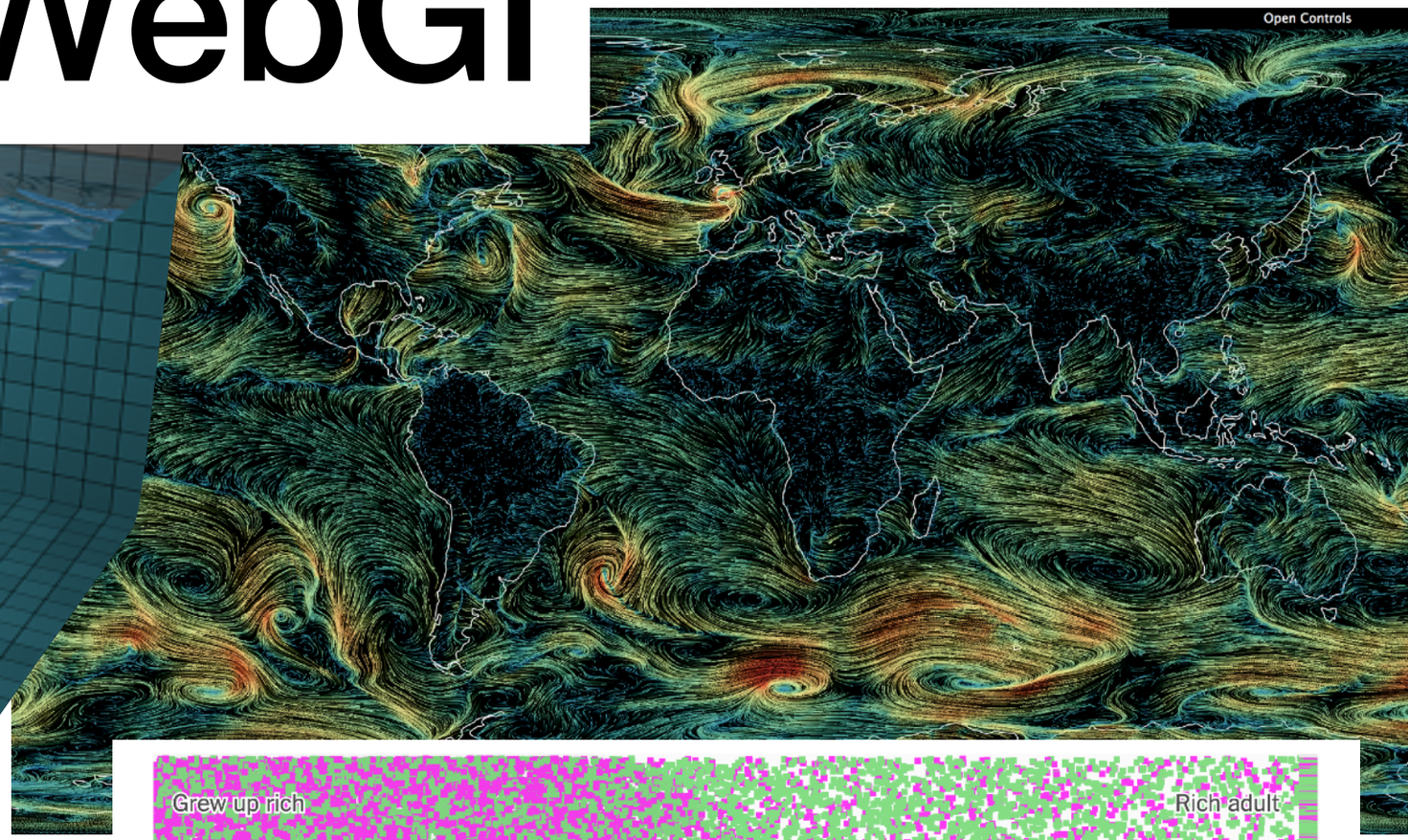
History!

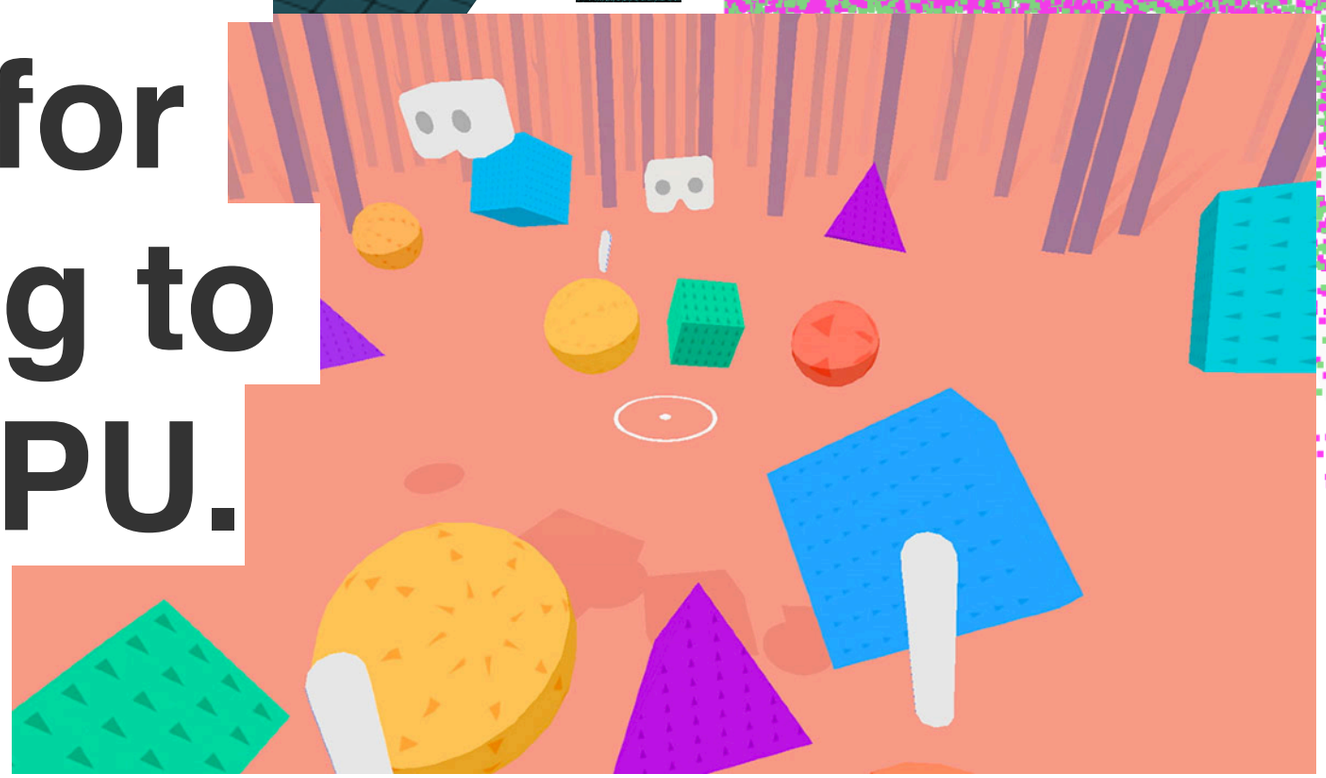
Cool GLSL libraries you can install from npm:

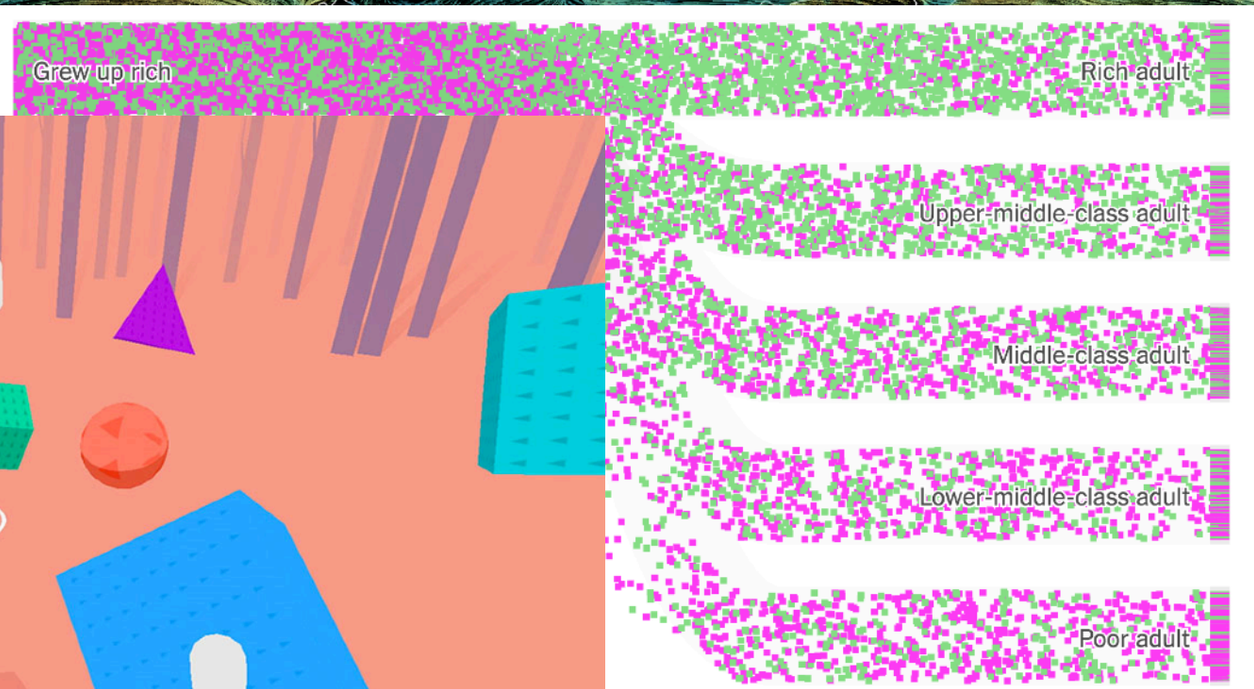
glsl-specular-beckmann
glsl-specular-cook-torrance
glsl-diffuse-oren-nayar
glsl-diffuse-lambert
glsl-specular-ward
glsl-specular-gaussian
glsl-specular-phong
glsl-specular-blinn-phong
glsl-perturb-normal
glsl-face-normal
glsl-checker
glsl-earth
glsl-easings
matcap
glsl-inverse
glsl-determinant
glsl-transpose
glsl-frobenius
glsl-look-at
glsl-camera-ray
glsl-raytrace
glsl-sdf-normal
glsl-sdf-sphere
glsl-sdf-box
glsl-sdf-primitives
glsl-sdf-ops
glsl-ruler
glsl-turntable-camera
glsl-combine-smooth
glsl-luma
glsl-gamma
glsl-aastep
glsl-dither
glsl-noise
glsl-fractal-brownian-noise
glsl-worley
glsl-random
glsl-fog
glsl-fxaa
glsl-lut
glsl-range
glsl-scale-linear
glsl-scale-log
glsl-square-frame
glsl-cornell-box
glsl-read-float
glsl-smooth-min
glsl-film-grain
glsl-hash-blur
glsl-fast-gaussian-blur
glsl-halftone
glsl-crosshatch-filter
glsl-ascii-filter
glsl-hsv2rgb
glsl-hsl2rgb
glsl-blend-overlay
glsl-blend-soft-light
glsl-map
glsl-edge-detection
glsl-atmosphere
glsl-godrays
glsl-cos-palette
glsl-gradient-palette
glsl-vignette
glsl-solid-wireframe
glsl-domain-coloring
glsl-sat
glsl-numerify
glsl-quad
glsl-gaussian
glsl-zoom
screen-projected-lines
glsl-fft
glsl-rfft

WebGl



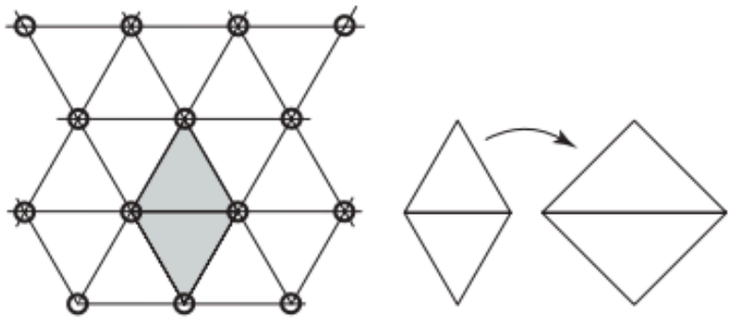




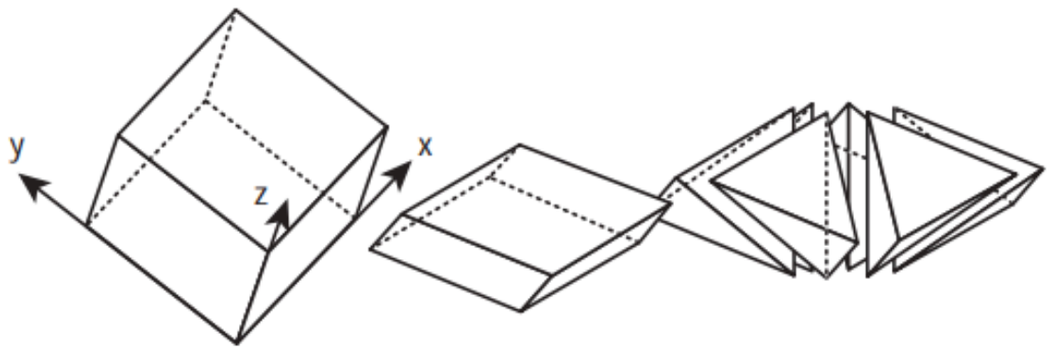


set of JS APIs for talking to the GPU.

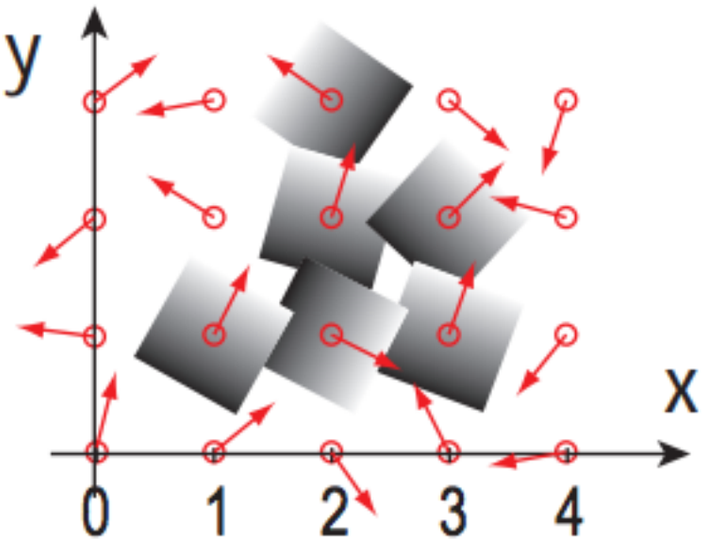
rhombus, a shape that can be thought of as a square that has been squashed along its main diagonal.

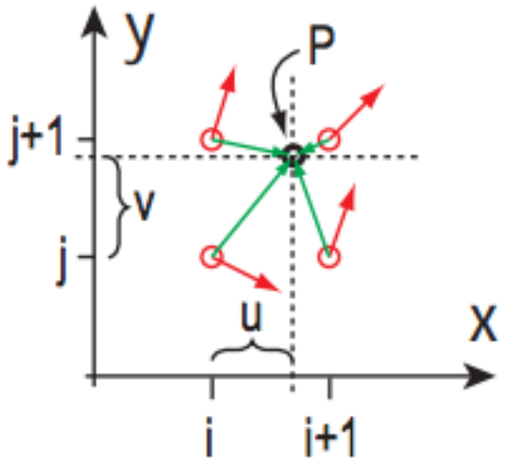


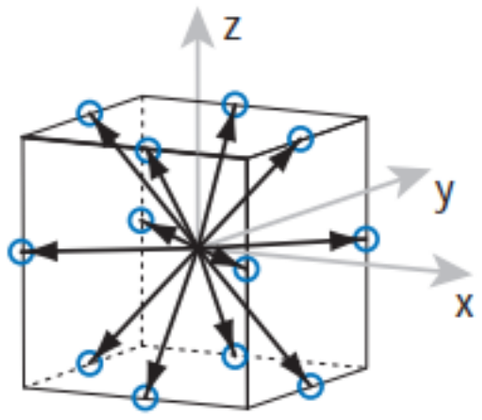
In three dimensions, the simplex shape is a slightly skewed tetrahedron, six of which mal
cube that has been squashed along its main diagonal.



ent, with the value zero at its associated grid point.







$g_0 = (0, 1, 1), g_1 = (0, 1, -1),$
$g_2 = (0, -1, 1), g_3 = (0, -1, -1),$
$g_4 = (1, 0, 1), g_5 = (1, 0, -1),$
$g_6 = (-1, 0, 1), g_7 = (-1, 0, -1),$
$g_8 = (1, 1, 0), g_9 = (1, -1, 0),$
$g_{10} = (-1, 1, 0), g_{11} = (-1, -1, 0)$