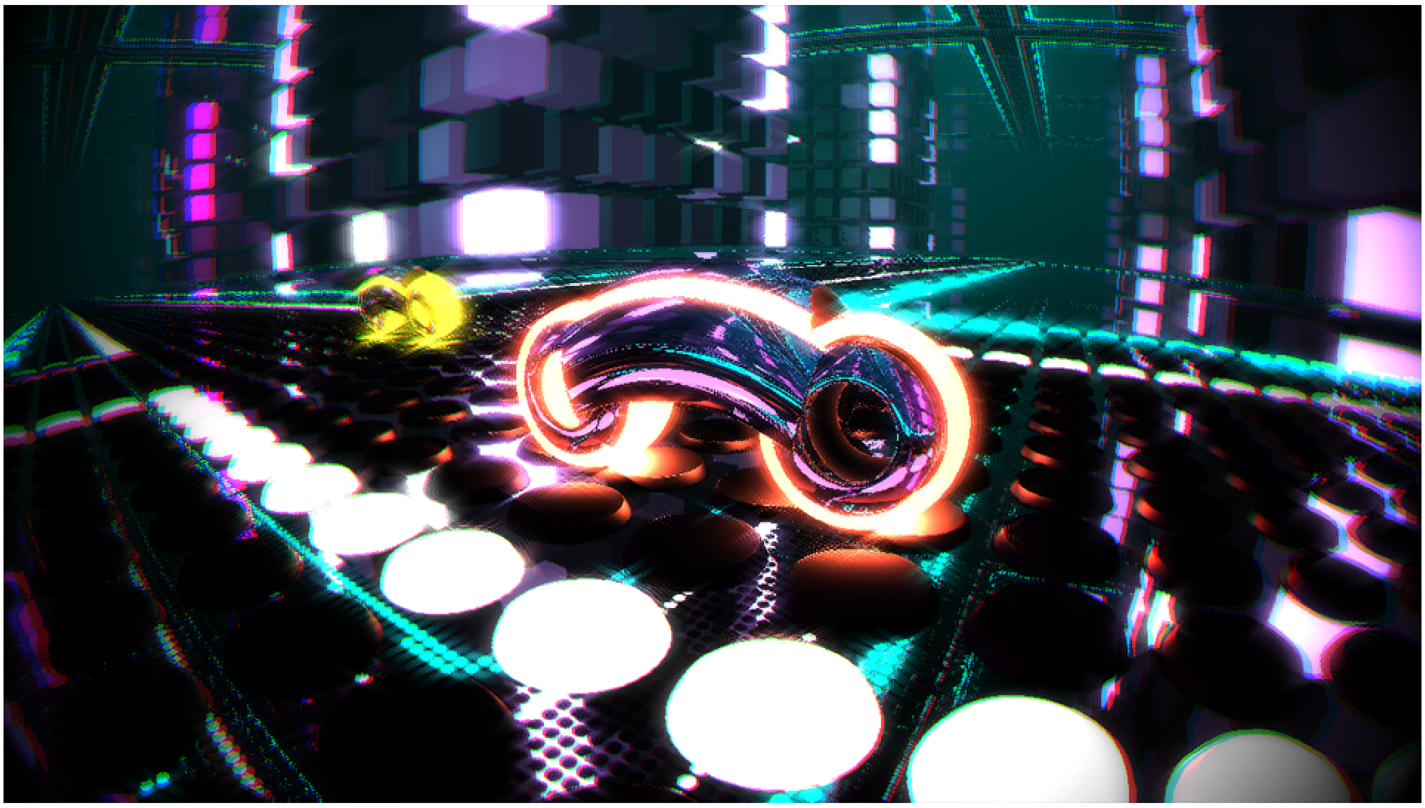
Making of Highway 4k

This is a short making of the Highway 4k intro that took part in Assembly 2013's 4k intro competition and won 1st place.



Getting the right look

I didn't really have a clear vision on what to do at the beginning of the project. I simply started writing raymarcher renderer for WebGL, which would support a few primitives and booleans. Since I was completely new to WebGL, OpenGL and GLSL I wasn't really sure what was possible and what was not. The 4k intro slisesix by Rgba was my starting point and I tried adding all the features I could think of to the renderer. After I had the basics down I started thinking about the actual intro. I decided pretty early on that it would be an endless city with floating roads and bikes driving on them.

I tried quite a few looks with shadows and ambient occlusion, but I wasn't really happy with them. They produced some artifacts and shadows that didn't really fit in with the bright bikes. So I dropped them and focused on some simple shaders to get the look I wanted. I spent a lot of time fine-tuning the shaders and colors to get the final look. After that, I focused on different types of cameras and editing.

Once I had the visuals finished I asked King Thrill to make the music for it. The graphics left slightly less than 1 kilobyte for the music, so King Thrill had to make some heavy sacrifices with the instruments and music. We were pretty surprised the first time we got the music and the visuals to fit into 4 kilobytes. I'd been afraid that we would have had to settle for a single instrument with two or three tracks.

After the music was done, I synchronized the buildings and road to pulse in sync with the beat and did some final tweaks to the cameras and editing. This still left me with some extra space, so I filled that with code that hid the scroll bars and mouse cursor. With those final touches, I managed to get the intro to hit exactly 4096 bytes, and I submitted it to the Assembly 4k intro competition.

Tools

Google Closure Compiler was used to minimize Javascript. I didn't really do any manual optimizing to the code.

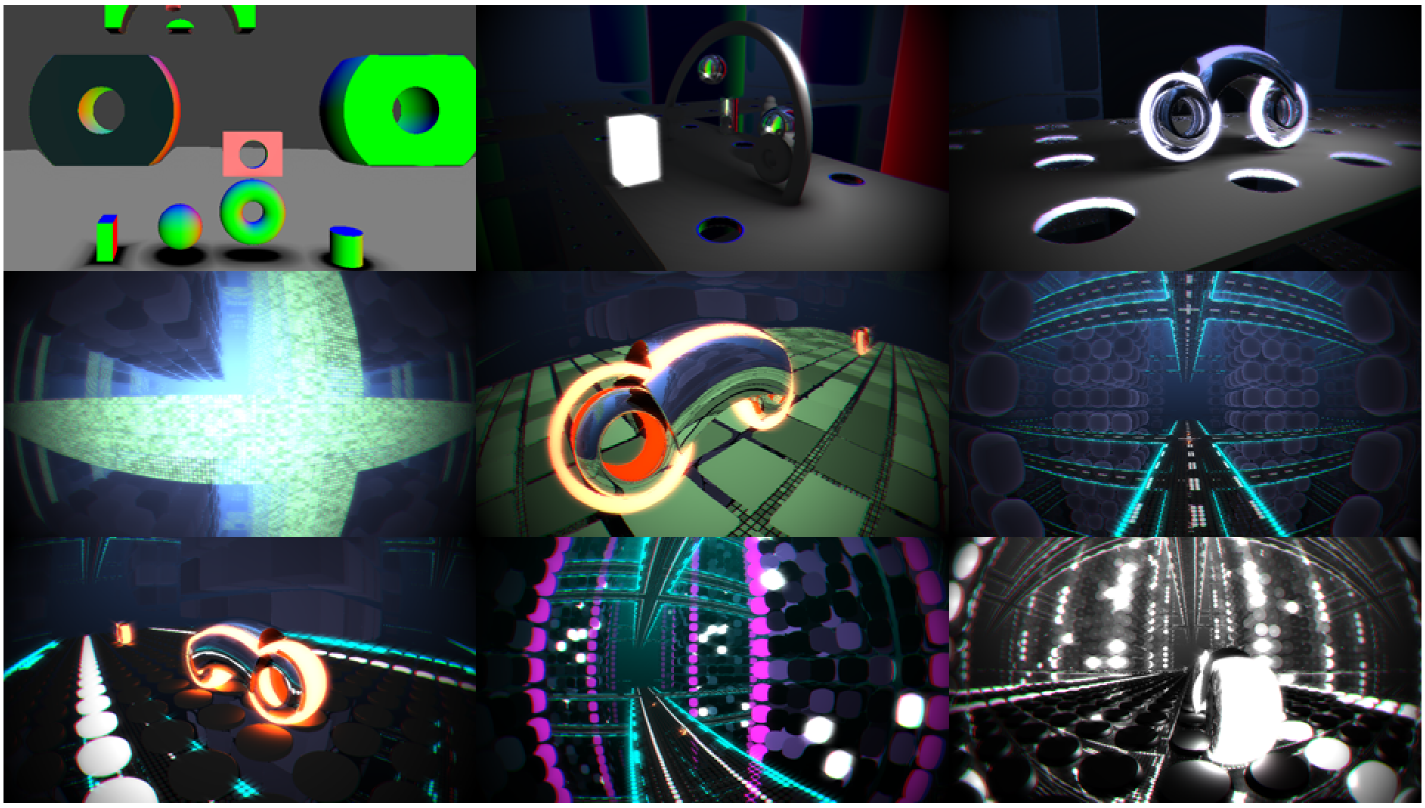
Shader Minifier was used to minify the GLSL shader. I didn't do any editing by hand. It's possible that some bytes may have been gained through manual editing, but I doubt that there were any more than that.

jsexe was used to compress the final code into a PNG image and add the html/javascript bootstrapper that decompressed the PNG and evaluated the javascript.

Notepad++ was used for writing all the code.

Sonant Live was used for making and playing the music. Another, probably better, option would have been to use SoundBox Lite, which is based on Sonant Live but has some improvements. Sadly, I overlooked this tool until late into production, making switching to it impossible; I would have needed to rewrite code and port the song.

I also wrote a short Java tool to shorten all the WebGL function names and parameters. While Jsexe also has the capability to do this, it came out late into production and I'd already written the Java tool.



Thoughts

This was my first real-time production and it took me quite a while to get hang of all the tools and techniques involved. I'd only previously worked on pre-rendered graphics and had never touched WebGL or OpenGL before this. From start to finish, the production time was approximately 300 hours. While this estimate includes the time required for me to learn how to use new tools, techniques, and what have you, it doesn't include King Thrill's time spent writing the music.

In the beginning it was very challenging to estimate how much space everything would take after optimization and compression. The first versions, with fairly complete visuals and some dummy music, used over 6 depressing kilobytes. However, after going through the shader code with a fine-tooth comb, I managed to significantly reduce the size. I learned a lot of techniques, such as hashing webgl functions names, that radically dropped the size of the demo. While working on Highway 4k, I also wrote my 1k intro, Kornell Box, which gave me a lot of new ideas on how to optimize the code. After writing the 1k intro, squeezing everything, including the final music, into 4k felt like a breeze.

No doubt this intro has a lot more room for optimization, but after 300 hours of work, it looked good enough in my eyes. I decided to leave improvements for future productions.

Javascript

This is the Javascript source code before the WebGL functions and variables were renamed as well as before the code was run through Closure Compiler. I defined one extern *S(x)* for the closure compiler so I could reuse the *String.fromCharCode* from jsexe, which saved few bytes. The final command was *java -jar compiler.jar --js main.js --js_output_file output.js --compilation_level ADVANCED_OPTIMIZATIONS --externs extern.js*

The music player and WebGL texture generation are probably good starting points for further optimization. Perhaps some clever tricks could be used to avoid creating floating point texture altogether and just make the two shaders work on framebuffer.

```
// Google Closure Compiler handles inlining of the smaller functions etc.
var gl, mCanvas;
var finalBuffer, shaderProgram, postProgram, buffer, texture;
var mWaveWords = 44100 * 2 * 164;

var mChnBufWork = new Int32Array(mWaveWords),
    mMixBufWork = new Int32Array(mWaveWords);

function oscillators(type, value) {
    return type == 0 ? Math.sin(value * 6.28318) : (value % 1) - .5;
}

function getnotefreq(n){
    return Math.pow(1.059463094, n - 128) / 256;
}

// Optimized Sonant Live player
function generateMusic(instr) {
    var panFreq = 1 / (4594 << 8);
    for(var b = 0; b < mWaveWords; b++) mChnBufWork[b] = 0;
    var currentpos = 0;
    for(var p = 0; p < 48; ++p) {
        var cp = instr[11][p];
        for(var row = 0;row < 32; ++row) {
            if(cp) {
                var n = instr[12][cp - 1][row];
                if(n) {
                    var c1 = c2 = 0;
                    var o1t = getnotefreq(n + (7 - 8) * 12);
                    var o2t = getnotefreq(n + (instr[2] - 8) * 12) * (1 + .0008 * 16);

                    for (var j = instr[5] + instr[6] + instr[7] - 1; j >= 0; --j) {
                        var k = j + currentpos;
                        // Envelope
                        var e = 1;
                        if(j < instr[5])
                            e = j / instr[5];
                        else if(j >= instr[5] + instr[6])
                            e -= (j - instr[5] - instr[6]) / instr[7];
                        // Oscillator 1
                        var t = o1t;
                        if(instr[0]) t *= e * e;
                        c1 += t;
                        var rsample = oscillators(instr[1],c1) * 255;
                        // Oscillator 2
                        t = o2t;
                        if(instr[3]) t *= e * e;
            c2 += t;
            rsample += oscillators(instr[4],c2) * 193;
            rsample *= (e / 255) * 10 * instr[8];
            // Add to channel buffer
            mChnBufWork[k*2] += rsample;
            mChnBufWork[k*2+1] += rsample;
          }
        }
      }
      currentpos += 4594;
    }
  }

  var p = ((instr[9] * 4594) >> 1) * 2;
  var t = instr[10] / 255;

  for(var b = 0; b < mWaveWords - p; b += 2) {
    var k = b + p;
    mChnBufWork[k] += mChnBufWork[b+1] * t;
    mChnBufWork[k+1] += mChnBufWork[b] * t;
  }

  for(var b = 0; b < mWaveWords; b++) mMixBufWork[b] += mChnBufWork[b];
};

function createAudio() {
  // This creates wav header
  var wave = atob("UklGRjhuuQFXQVZFZm10IBAAAAABAAIARKwAABCxAgAEABAAZGF0YRRuuQE=");
  for (var b = 0; b < mWaveWords; b++)
    // Here we are reusing String.fromCharCode used by jsexe to unpack png
    //wave += S(mMixBufWork[b] & 255, (mMixBufWork[b] >> 8) & 255);
    wave += String.fromCharCode(mMixBufWork[b] & 255, (mMixBufWork[b] >> 8) & 255);
  return new Audio("data:audio/wav;base64," + btoa(wave)); // This line crashes Chrome and Opera
};

// Here is the actual song data, music uses 5 different instruments
generateMusic([0,1,7,0,0,0,1075,4611,192,4,61,[1,1,1,1,1,1,1,1,2,2,2,3,2,2,2,3,2,2,2,3,2,2,2,2,2,3,2
,2,2,3,1,1,1,3,2,2,2,3,2,2,3,3,2,2,2,3,1,1],[[144,132,132,132,0,132,132,132,132,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,130,130,130,130],[168,0,132,132,0,132,132,132,132,0,132,132,0,132,132,132,144,0,
132,132,0,132,132,132,132,0,132,132,0,132,132,132],[168,0,128,128,0,128,128,128,128,0,128,128,0,128,
128,128,149,0,130,130,0,130,130,130,130,0,142,130,0,130,156,130]]]);
generateMusic([0,1,8,0,1,0,2193,4611,127,4,61,[0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,2,1,1,1,2,0,0,0,0,0,0,1
,1,1,2,1,1,1,2,1,1,1,2,1,1,2,2,0,0,0,0,0,0],[[156,0,0,156,0,0,158,0,156,0,0,156,0,0,159,0,156,0,0,
156,0,0,161,0,156,0,154,0,156,0,163,0],[152,0,0,152,0,0,159,0,152,0,0,152,0,0,154,0,154,0,0,154,0,0,
156,0,156,0,154,0,156,0,166,0]]]);
generateMusic([1,0,7,1,0,0,2193,11102,196,0,0,[1,1,1,1,1,1,1,1,1,1,1,0,1,1,1,1,1,1,1,1,0,0,1,1,1,1,1
,1,1,1,1,1,1,0,1,1,1,1,1,1,1,1,0,1,1,1,0,0],[[132,0,0,0,132,0,0,0,132,0,0,0,132,0,0,0,132,0,0,0,132,
0,0,0,132,0,0,0,132,0,0,0]]]);
generateMusic([1,1,6,1,0,0,1075,4298,255,4,63,[0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,0,0,0,1,1
,1,1,1,0,0,0,1,1,1,1,1,1,1,1,1,1,1,0,0,0],[[0,0,0,0,156,0,0,0,0,0,0,0,156,0,0,0,0,0,0,0,156,0,0,0,
0,0,0,0,156,0,0,156]]]);
generateMusic([0,0,1,7,0,1,14824,47416,50525,67,4,63,[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,
0,0,0,1,1,1,2,0,0,1,2,1,1,1,2,1,1,2,2,1,0,0,2,1,0],[[156,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0],[168,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,166,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]]]);

var audio = createAudio();

// Generate route which the bikes drive along
var route = [-1,0];
var xx = -1, y = 0, xd = 1, yd = 0, temp = 0;
for (var i = 0; i<250; i++) {
  var r = (xx*xx*7+y*y*3) % 10;
  if (r > 7) {
    // Turn left
    temp = xd;
```

```javascript
        xd = yd;
        yd = temp;
    }
    if (r < 1) {
        // Turn right
        temp = xd;
        xd = -yd;
        yd = temp;
    }
    xx += xd;
    y += yd;
    route.push(xx);
    route.push(y);
}

var getPositionSimple= function(time, axis) {
    var step = Math.floor(time);
    return route[step*2+axis] * (1-time+step) + route[step*2+2+axis] * (time-step);
}

var normalDistribution = function(value) {
    return (1. + Math.cos(value)) / 3.14159 * 2.;
}

// Interpolate 500 points along the route weighted with normal distribution to create smooth turns
var getPositionIntepolated= function(spread, time, axis) {
    var total = 0.;
    var n = 250;
    for(var i = -n; i < n; i++)
        total += getPositionSimple(time + (spread * i)/n, axis) * normalDistribution(i*.01);
    return total/394.5;
}

// Calculate position, direction and curvature. Curvature is used for tilting bikes and camera
var getPosition3= function(time, corner) {
    var pos1ax = getPositionIntepolated(corner, time, 0);
    var pos1ay = getPositionIntepolated(corner, time, 1);
    var pos1bx = getPositionIntepolated(corner, time+.2, 0);
    var pos1by = getPositionIntepolated(corner, time+.2, 1);
    var dirAx = pos1bx - pos1ax;
    var dirAy = pos1by - pos1ay;
    var pos1bx = getPositionIntepolated(corner, time+.4, 0) - pos1bx;
    var pos1by = getPositionIntepolated(corner, time+.4, 1) - pos1by;
    return [pos1ax, pos1ay, dirAx, dirAy,
        -Math.atan2(dirAx*pos1by - dirAy*pos1bx, dirAx*pos1bx + dirAy*pos1by)]
}

// Create canvas, some space could be possibly saved by creating part of this in HTML
mCanvas = document.createElement("canvas");
document.body.appendChild(mCanvas);
document.body.style.overflow = "hidden";
mCanvas.style.cursor = "none";
mCanvas.style.position = "fixed";
mCanvas.style.left = mCanvas.style.top = 0;
mCanvas.width = 1280;
mCanvas.height = 720;

// Hash webgl context function names, below we still use full names, shortening with Java tool
// before running the code through Closure Compiler
for(k in gl = mCanvas.getContext('experimental-webgl'))
    gl[k.match(/^..|[A-Z]/g).join('')] = gl[k];

with (gl) {
    // Initialize textures and framebuffers, we need somewhere to store preliminary result that is
    // sent to post processing for bloom and chromatic aberration
    finalBuffer = gl.createBuffer();
    buffer = gl.createFramebuffer();
    gl.bindFramebuffer(gl.FRAMEBUFFER, buffer);

    texture = gl.createTexture();
    gl.bindTexture(gl.TEXTURE_2D, texture);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.CLAMP_TO_EDGE);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.CLAMP_TO_EDGE);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST);
    gl.getExtension("OES_texture_float");
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, 1280, 720, 0, gl.RGBA, gl.FLOAT, null);
    gl.framebufferTexture2D(gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0, gl.TEXTURE_2D, texture, 0);

    var s1= gl.createShader(gl.FRAGMENT_SHADER);
    var s3= gl.createShader(gl.FRAGMENT_SHADER);
    var s2= gl.createShader(gl.VERTEX_SHADER);

    // Fragment shader: Renderer, fshader.c
    gl.shaderSource(s1, "precision highp float;uniform vec2 X[8];mat3 v,i,f;vec3 m,r,l,g,a,d;float "
+"n,b,s,c;int p,e;void R(inout float v,float g){v=mod(abs(v),2.*g);v=v>g?2.*g-v:v;}vec3 M(vec3 v){v"
+".b+=1.;R(v.r,8.);R(v.g,8.);R(v.b,4.);if(v.r<v.g){float m=v.r;v.r=v.g;v.g=m;}return v;}void U(inou"
+"t float v,inout int g,float m,int n){if(abs(v)>abs(m))g=n;v=max(v,-m);}void E(float v,int g,float"
+" m,int b){float n=max(v,m);if(c>n){c=n;e=abs(v)<abs(m)?g:b;}}float B(vec3 v,vec3 g){vec3 b=abs(v)"
+"-g;return min(b.r>b.g?b.r>b.b?b.r:b.b:b.g>b.b?b.g:b.b,length(max(b,0.)));}float O(vec3 v,vec2 g){"
+"return length(vec2(length(v.rb)-g.r,v.g))-g.g;}void K(mat3 v,vec3 m,vec3 g){g-=m;float b=length(g"
+");if(b>.4){b-=.1;if(c>b)c=b;}else{vec3 i=g*v;i=vec3(abs(i.rg),i.b+.04);b=O(i-vec3(0,0,-.07),vec2("
+".15,.05));int r=4;U(b,r,length(i-vec3(.23,0,.022))-.16,4);U(b,r,length(i-vec3(0,0,-.15))-.2,3);U("
+"b,r,O(i-vec3(.228,0,0),vec2(.16,.038)),3);U(b,r,O(i-vec3(0,.043,-.07),vec2(.19,.012)),2);if(c>b){"
+"c=b;e=r;}i-=vec3(.1,0,.042);E(O(i-vec3(0,.031,0),vec2(.038,.011)),3,O(i,vec2(.03)),4);E(O(i,vec2("
+".15,.1)),2,length(i)-.06,2);}}float F(vec3 g){vec3 a=M(g),l=mod(a,vec3(1))-.5;float d=.39+sin(n*2"
+".)*.1;c=max(B(a-vec3(8,8,0),vec3(4,4,8)),mix(length(l)-d,B(l,vec3(d)),(sin(n)+1.)*.5));e=1;if(b>3"
+"5.&&b<142.){K(v,m,g);K(i,r,g);}if(b>25.&&b<150.)E(B(a-vec3(0,0,1),vec3(8,1,.005)),0,length(vec3(m"
+"od(a.r,.1)-.05,mod(a.g,.1)-.05,a.b-.98))-.045,0);return c;}float Y(){float i=.0005,v=.05;for(int "
+"b=0;b<156;b++){vec3 m=g+a*v;float n=F(m);if(n<.0002){p=e;d=normalize(vec3(n-F(m-vec3(.001,0,0)),n"
+"-F(m-vec3(0,.001,0)),n-F(m-vec3(0,0,.001))));return v;}v+=n<i?i:n;if(v>40.)break;i+=1.8e-06*float"
+"(b*b);}return 40.;}float P(vec3 v){v.b+=.07;vec3 m=v-g;float i=length(m);i*=i;return pow(clamp(do"
+"t(d,normalize(m/i+a)),0.,1.),20.)/i;}float D(vec3 v,float m){return pow(clamp(dot(d,normalize(v-g"
+")),0.,1.),m);}vec3 S(){vec3 n=M(g),f=vec3(1.2,.4,.25),c=floor(g);if(p==1){vec2 e=floor(g.rg/16.);"
+"float t=fract(sin(dot(e,vec2(12.9898,78.233)))*43758.5),o=fract(sin(dot(vec2(c.r,c.g+c.b*13.),vec"
+"2(12.9898,78.233)))*43758.5);f=mix(vec3(1.5,0.,1.5),vec3(.8,.8,1.2),t);if(n.r<5.&&mod(g.b+floor(t"
+"*6.9),6.)>2.)f*=2.;else if(o>.98)f=vec3(2);else f*=(s+.7*pow(o,4.))*(2.-floor(n.r)/4.);f+=(D(m,40"
+".)*vec3(1.2,.4,.25)+D(r,40.)*vec3(.9,.9,.1)+D(l,40.)*vec3(0,1,1))*.2;return f;}if(length(g-m)>.5)"
+"f=vec3(.95,.85,.1);if(p==2)return f*2.;if(p>2)return f*(dot(d,a)+1.)*.5;f=vec3(0);if(b>35.&&b<142"
+".)f=((P(m-v[0]*.1)+P(m+v[0]*.1))*vec3(1.2,.4,.25)+(P(r-i[0]*.1)+P(r+i[0]*.1))*vec3(.9,.9,.1))*.08"
+";if(n.g>.9||n.g>.8&&mod(n.r/.5,2.)<1.)f+=vec3(0,2,2);if(n.g>.4&&n.g<.5)f+=vec3(1);f+=D(l,1.)*vec3"
+"(.5,0,.5)*s;return f;}vec3 C(float v){vec2 m=(gl_FragCoord.rg-vec2(1280,720)*.5)/720.;float f=len"
+"gth(m),g=f*3.14159*v;return vec3(cos(g),m.r/f*sin(g),m.g/f*sin(g));}mat3 I(vec3 v,float m){vec3 g"
+"=normalize(v),r=normalize(vec3(-g.g,g.r,m));return mat3(g,r,cross(g,r));}float A(float v){return "
+"max(0.,4.-abs(b-v)*20.);}void main(){n=X[7].g;b=n*2.5;s=mod(2.4*(b+.12),1.)*5.;if(s>1.)s=1.25-s*."
+"25;s*=s;s=s*clamp(6.9-abs(80.-b)*.1,0.,1.);f=I(vec3(X[5],0),X[7].r);v=I(vec3(X[1],0),X[6].r-cos(n"
+")*.35);i=I(vec3(X[3],0),X[6].g-sin(n)*.35);l=vec3(X[4]*16.+f[1].rg*sin(n)*1.5,1.2+sin(n*.5+2.5));"
+"m=vec3(X[0]*16.+v[1].rg*sin(n)*.3,.07);r=vec3(X[2]*16.+i[1].rg*cos(n)*.3,.07);g=l;vec3 c=vec3(0);"
+"int e=b<45.?0:b<52.?1:b<55.?3:b<59.?5:b<63.6?6:b<67.5?1:b<70.5?2:b<73.5?3:b<76.?0:b<87.?1:2;if(b>"
+"93.)e=b<98.?0:b<101.?5:b<106.?3:b<112.?4:b<115.?2:b<120.?3:b<125.?1:b<130.?3:b<133.?4:b<135.?5:b<"
+"137.?6:b<138.?3:b<145.?1:0;if(e==2){g=b<80.?vec3(-112,-81.6,.4):b<100.?vec3(-192,-113.6,1.2):vec3"
+"(-81.6,-78.4,-.01);a=I(m-g,0.)*C(.15);}if(e==1){float t=.25+sin(b)*.15;g=vec3(t*cos(n*1.5),t*sin("
+"n*1.5),.1)+m;a=I(m-g,0.)*C(.4);}if(e==3){g=r+vec3(0,0,.05)+vec3(normalize(i[1].rg)*.13,0);a=I(m-g"
+",0.)*C(.6);}if(e==6){g=m+vec3(0,0,.01)+vec3(normalize(i[1].rg)*.05,0)+v[0]*.25;a=I(-v[0],0.)*C(.6"
+");}if(e==4)a=I(m-g,0.)*C(.8);if(e==5){g=(m+r)*.5+vec3(.2,.2,2);a=-C(.8).gbr;}if(e==0)a=f*C(max(b*"
+".1-14.9,.5));float t=Y();if(t!=40.){g+=a*t;if(p==4){a=reflect(a,d);float o=Y();if(o!=40.){g+=a*o;"
+"c=S();}c=mix(c,vec3(0,.2,.2)*(a.b+1.)*(a.b+1.),min(o/40.,1.))*vec3(.8,.5,.8);}else c=S();}c=mix(c"
+",vec3(0,.2,.2)*(a.b+1.)*(a.b+1.),min(t/40.,1.));if(.25-min(abs(1.-b),abs(127.-b))*.04+sin(b*400.)"
+"*.02>0.)c=vec3(pow((c.r+c.g+c.b)*.33,1.));gl_FragColor=vec4((b>159.5?vec3(0):b<2.?vec3(0):c+vec3("
+"1,.1,1)*A(25.)+vec3(.1,1,1)*A(35.)+vec3(.1,1,1)*A(142.)+vec3(1,.1,1)*A(150.))+vec3(1,1,.1)*A(159."
+"5)+vec3(1,1,.1)*A(2.),1);}");
    // Fragment shader: Post effects, fpost.c
    gl.shaderSource(s3, "precision highp float;uniform sampler2D S;void main(){vec2 f=gl_FragCoord."
+"rg;float t,v=length((f-vec2(1280,720)*.5)/720.);v*=v;f/=vec2(1280,720);vec4 r,g=vec4(0);for(int i"
+"=-9;i<10;i++)r=texture2D(S,f-vec2(1)*float(i)*.0025),t=1.-abs(float(i))*.1,g+=r*t*t*dot(r,r)*.013"
+"2;for(int i=-9;i<10;i++)r=texture2D(S,f-vec2(-1,1)*float(i)*.0025),t=1.-abs(float(i))*.1,g+=r*t*t"
+"*dot(r,r)*.0132;g+=vec4(texture2D(S,f-vec2(.01,0.)*v).r,texture2D(S,f-vec2(0.,.01)*v).g,texture2D"
+"(S,f).b,0);g*=1.-v;g.a=1.;gl_FragColor=g;}");
    // Vertex shader
    gl.shaderSource(s2, "attribute vec4 p;void main(){gl_Position=p;}");

    gl.compileShader(s1);
    gl.compileShader(s3);
    gl.compileShader(s2);

    shaderProgram = gl.createProgram();
    postProgram = gl.createProgram();

    gl.attachShader(shaderProgram, s1);
    gl.attachShader(shaderProgram, s2);
    gl.attachShader(postProgram, s3);
    gl.attachShader(postProgram, s2);

    gl.linkProgram(shaderProgram);
    gl.linkProgram(postProgram);

    gl.bindBuffer(gl.ARRAY_BUFFER, finalBuffer);
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array([1,1,1,-3,-3,1]), gl.STATIC_DRAW);
    gl.enableVertexAttribArray(0);
    gl.vertexAttribPointer(0, 2, gl.FLOAT, false, 0, 0);

    audio.play();

    setInterval(function() {
        // Calculate bike and camera positions
        var time = audio.currentTime*.4;
        var pos1 = getPosition3(time+5.5, .2); // Bike 1
        var pos2 = getPosition3(time+5.35 + Math.sin(time*.4) * .1, .2); // Bike 2
        var pos3 = getPosition3(time+5.15+Math.sin(time*.5)*.1, .75); // Camera

        // Draw scene
        gl.bindFramebuffer(gl.FRAMEBUFFER, buffer);
        gl.useProgram(shaderProgram);
        gl.bindBuffer(gl.ARRAY_BUFFER, finalBuffer);
        gl.uniform2fv(gl.getUniformLocation(shaderProgram, "X"),
            [pos1[0], pos1[1], pos1[2], pos1[3], // pos+dir 1
            pos2[0], pos2[1], pos2[2], pos2[3], // pos+dir 2
            pos3[0], pos3[1], pos3[2], pos3[3], // pos+dir 3
            pos1[4],pos2[4],pos3[4], time] // curvature 1 2 3 + time
        );
        gl.drawArrays(gl.TRIANGLES, 0, 3);

        // Send results to post processing
        gl.bindTexture(gl.TEXTURE_2D, texture);
        gl.bindFramebuffer(gl.FRAMEBUFFER, null);

        // Apply post processing
        gl.useProgram(postProgram);
        gl.bindBuffer(gl.ARRAY_BUFFER, finalBuffer);
        gl.uniform1i(gl.getUniformLocation(postProgram, "S"), 0);

        //gl.activeTexture(gl.TEXTURE0);
        gl.bindTexture(gl.TEXTURE_2D, texture);
        gl.drawArrays(gl.TRIANGLES, 0, 3);
    }, 16);
}
```

Shader - Renderer

This shader generates most of the visuals that you see on the screen. Firefox didn't like reusing function and variable names so I had to rename them myself and force Shader Minifier to leave them untouched. It was minified with this command: *shader_minifier.exe -v --format none --no-sequence --field-names rgba --preserve-externals --no-renaming-list R,M,U,E,B,O,K,F,Y,P,D,S,C,I,A,main fshader.c*

```
precision highp float;

uniform vec2 X[8];

mat3 bike1tm;
mat3 bike2tm;
mat3 camtm;

vec3 bike1pos;
vec3 bike2pos;
vec3 campos;
vec3 rO;
vec3 rD;
vec3 hitNormal;

float time;
float timeScaled;
float pulse;
float hDistTemp;

int hMaterial;
int hMaterialTemp;

// ##################### Math #####################

// Mod around 0, might have been smarter to move entire scene further from 0,0,0 to make it simpler
void R(inout float x, float n) {
    x = mod(abs(x),2.*n);
    x = x>n ? 2.*n-x : x;
}

vec3 M(vec3 v) {
    v.z+=1.;

    R(v.x,8.);
    R(v.y,8.);
    R(v.z,4.);

    if (v.x < v.y) {
        float x = v.x;
        v.x = v.y;
        v.y = x;
    }

    return v;
}

// ##################### Booleans #####################

void U(inout float ad, inout int am, float bd, int bm) {
    if (abs(ad)>abs(bd))
        am = bm;
    ad = max(ad,-bd);
}
```

```
void E(float ad, int am, float bd, int bm) {
    float distance = max(ad,bd);
    if (hDistTemp > distance) {
        hDistTemp = distance;
        hMaterialTemp = abs(ad)<abs(bd)?am:bm;
    }
}

// #################### Primitives ####################

float B(vec3 p, vec3 b) {
    vec3 v = abs(p) - b;
    return min(v.x>v.y?v.x>v.z?v.x:v.z:v.y>v.z?v.y:v.z,length(max(v,.0)));
}

float O(vec3 p, vec2 t) {
    return length(vec2(length(p.xz)-t.x,p.y))-t.y;
}

// #################### Scene ####################

void K(mat3 tm, vec3 pos, vec3 p) { // Bike distance field
    p -= pos;
    float distance = length(p);
    if (distance>.4) {
        distance -= .1;
        if (hDistTemp > distance)
            hDistTemp = distance;
    } else {
        vec3 vp = p * tm;
        vp = vec3(abs(vp.xy), vp.z+.04);

        distance = O(vp-vec3(0,0,-.07), vec2(.15, .05));
        int material = 4;
        U(distance, material, length(vp - vec3(.23,0, .022))-.16, 4);
        U(distance, material, length(vp - vec3(0,0,-.15))- .2, 3);
        U(distance, material, O(vp - vec3(.228,0,0), vec2(.16, .038)), 3);
        U(distance, material, O(vp - vec3(0,.043,-.07), vec2(.19, .012)), 2);
        if (hDistTemp > distance) {
            hDistTemp = distance;
            hMaterialTemp = material;
        }
        vp-=vec3(.1,0,.042);
        E(O(vp - vec3(0,.031,0), vec2(.038, .011)), 3, O(vp, vec2(.03)), 4);
        E(O(vp, vec2(.15, .1)), 2, length(vp) - .06, 2);
    }
}

float F(vec3 p) { // Distance field
    vec3 mp = M(p),
    cp = mod(mp, vec3(1)) - .5;

    // Buildings
    float width = .39 + sin(time*2.) * .1;
    hDistTemp = max( B(mp-vec3(8,8,0),vec3(4,4,8)),
                    mix(length(cp) - width,
                        B(cp, vec3(width)),
                        (sin(time) + 1.) * .5));
    hMaterialTemp = 1;

    if (timeScaled > 35. && timeScaled < 142.) { // Bikes
        K(bike1tm, bike1pos, p);
        K(bike2tm, bike2pos, p);
    }
    if (timeScaled > 25. && timeScaled < 150.) // Road
        E(B(mp-vec3(0,0,1),vec3(8,1,.005)), 0,
            length(vec3(mod(mp.x, .1) - .05,mod(mp.y, .1) - .05,mp.z - .98)) - .045, 0);

    return hDistTemp;
}

// ##################### Raymarching #####################

float Y() {
    float minStep = .0005, dist = .05;
    for (int i = 0; i < 156; i++) {
        vec3 p = rO + rD * dist;
        float f = F(p);
        if (f <  .0002) {
            hMaterial = hMaterialTemp;
            hitNormal = normalize(vec3(
                f - F(p - vec3(.001, 0, 0)),
                f - F(p - vec3(0, .001, 0)),
                f - F(p - vec3(0, 0, .001))));
            return dist;
        }
        dist += f < minStep ? minStep : f;
        if (dist > 40.)
            break;
        minStep += 0.0000018 * float(i*i);
    }
    return 40.;
}

// ##################### Shading #####################

float P(vec3 light) {
    light.z += .07;
    vec3 lightDir = light - rO;
    float distance = length( lightDir );
    distance *= distance;
    return pow(clamp(dot(hitNormal, normalize(lightDir/distance + rD)),0.,1.), 20. ) / distance;
}

float D(vec3 light, float sharpness) {
    return pow(clamp(dot(hitNormal, normalize(light-rO)), 0., 1.), sharpness);
}

vec3 S() { // Shading happens here
    vec3 mp = M(rO), color = vec3(1.2, .4, .25), floored = floor(rO);

    if (hMaterial == 1) { // Buildings
        vec2 fp = floor(rO.xy/(16.));

        float r = fract(sin(dot(fp,vec2(12.9898,78.233))) * 43758.5453);
        float r2 = fract(sin(dot(vec2(floored.x, floored.y + floored.z * 13.),
                    vec2(12.9898,78.233))) * 43758.5453);

        color = mix(vec3(1.5,0.,1.5), vec3(.8,.8,1.2), r);

        if ((mp.x < 5. && mod(rO.z+floor(r*6.9),6.) > 2.)) // Corners
            color *= 2.;
        else if (r2 > .98) // Bright spots
            color = vec3(2);
        else // Standard
            color *= (pulse + .7 * pow(r2, 4.)) * (2.0 - floor(mp.x) / 4.);

        color += (D(bike1pos, 40.) * vec3(1.2, .4, .25) +
                D(bike2pos, 40.) * vec3(.9,.9,.1) +
                D(campos, 40.) * vec3(0,1,1)) * .2;

        return color;
    }

    if (length(rO-bike1pos) > .5) // Change color around bike 2
        color = vec3(.95,.85,.1);

    if (hMaterial == 2) // Tires
        return color * 2.;

    if (hMaterial > 2) // Bike innards + Reflection second time + Inner tire
        return color * (dot(hitNormal, rD) + 1.) * .5;

    color = vec3(0);

    if (timeScaled > 35. && timeScaled < 142.)
        color = ((P(bike1pos - bike1tm[0]*.1) +
                P(bike1pos + bike1tm[0]*.1)) * vec3(1.2, .4, .25) +
                (P(bike2pos - bike2tm[0]*.1) +
                P(bike2pos + bike2tm[0]*.1)) * vec3(.9,.9,.1))
                * .08;

    if (mp.y > .9 || (mp.y > .8 && (mod(mp.x/.5,2.) < 1.))) // Outline + Stripes
        color += vec3(0,2,2);
    if ((mp.y > .4 && mp.y < .5)) // Line
        color += vec3(1);

    color += D(campos, 1.) * vec3(.5,0,.5) * pulse;

    return color;
}

// #################### Cameras ####################

vec3 C(float fov) { // Fisheye
    vec2 pos = (gl_FragCoord.xy-vec2(1280,720)*.5)/720.; // Screen space: -.5 .. .5
    float radius = length(pos);
    float theta = radius * 3.14159 * fov;
    return vec3(cos(theta),(pos.x/radius)*sin(theta),(pos.y/radius)*sin(theta));
}

mat3 I(vec3 dir, float tilt) { // Transformation matrix
    vec3 vx = normalize(dir);
    vec3 vy = normalize(vec3(-vx.y, vx.x, tilt));
    return mat3(vx, vy, cross(vx, vy));
}

float A(float timePos) { // Flash screen
    return max(0., 4. - abs(timeScaled-timePos) * 20.);
}

// #################### Main ####################

void main( void ) {
    time = X[7].y;
    timeScaled = time * 2.5;

    // Pulse is synced with audio
    pulse = mod(2.4 * (timeScaled+.12),1.) * 5.;
    if (pulse>1.)
        pulse = 1.25 - pulse * .25;
    pulse *= pulse;
    pulse = pulse * clamp(6.9-abs(80.-timeScaled)*.1,0.,1.);
```

```
// Calculate transformation matrices for bikes and camera
camtm   = I(vec3(X[5],0), X[7].x);
bike1tm = I(vec3(X[1],0), X[6].x-cos(time)*.35);
bike2tm = I(vec3(X[3],0), X[6].y-sin(time)*.35);

// Positions for bikes and camera
campos   = vec3(X[4]*16. + camtm[1].xy   * sin(time)  * 1.5, 1.2+sin(time*.5+2.5));
bike1pos = vec3(X[0]*16. + bike1tm[1].xy * sin(time) * .3,   .07);
bike2pos = vec3(X[2]*16. + bike2tm[1].xy * cos(time) * .3,   .07);

rO = campos;

vec3 color = vec3(0);

// This had to be split into two parts, because otherwise it caused out of memory error when
// compiling, I guess there might be some neater way of doing this
int camera = // 0=free, 1=bike, 2=drive by, 3=follow, 4=warp, 5=sky, 6=rear view
    timeScaled < 45.  ? 0  :
    timeScaled < 52.  ? 1  :
    timeScaled < 55.  ? 3  :
    timeScaled < 59.  ? 5  :
    timeScaled < 63.6 ? 6  :
    timeScaled < 67.5 ? 1  :
    timeScaled < 70.5 ? 2  :
    timeScaled < 73.5 ? 3  :
    timeScaled < 76.  ? 0  :
    timeScaled < 87.  ? 1  : 2;
if (timeScaled > 93.)
    camera =
    timeScaled < 98.  ? 0  :
    timeScaled < 101.  ? 5  :
    timeScaled < 106.  ? 3  :
    timeScaled < 112.  ? 4  :
    timeScaled < 115.  ? 2  :
    timeScaled < 120.  ? 3  :
    timeScaled < 125.  ? 1  :
    timeScaled < 130.  ? 3  :
    timeScaled < 133.  ? 4  :
    timeScaled < 135.  ? 5  :
    timeScaled < 137.  ? 6  :
    timeScaled < 138.  ? 3  :
    timeScaled < 145.  ? 1  : 0;

if (camera==2) { // Drive by cam
    rO = timeScaled < 80. ?
            vec3(-112, -81.6, .4) : // 1st
            timeScaled < 100. ?
                vec3(-192, -113.6, 1.2) : // 2nd
                vec3(-81.6, -78.4, -.01); // 3rd
    rD = I(bike1pos-rO,0.) * C(.15);
}
if (camera==1) { // Bike cam
    float radius = .25 + sin(timeScaled) * .15;
    rO = vec3(radius * cos(time*1.5), radius * sin(time*1.5), .1) + bike1pos;
    rD = I(bike1pos-rO,0.) * C(.4);
}
if (camera==3) { // Follow cam
    rO = bike2pos + vec3(0,0,.05) + vec3(normalize(bike2tm[1].xy)*.13, 0);
    rD = I(bike1pos-rO,0.) * C(.6);
}
if (camera==6) { // Rear view cam
    rO = bike1pos + vec3(0,0,.01) + vec3(normalize(bike2tm[1].xy)*.05, 0) + bike1tm[0]*.25;
    rD = I(-bike1tm[0],0.) * C(.6);
}
```

```
    if (camera==4) // Warp cam
        rD = I(bike1pos-rO,0.) * C(.8);

    if (camera==5) { // Sky cam
        rO = (bike1pos + bike2pos) * .5 + vec3(.2,.2,2);
        rD = -C(.8).yzx;
    }
    if (camera==0) // Free cam
        rD = camtm * C(max(timeScaled*.1-14.9,.5));

    float dist = Y(); // Raymarch into scene
    if (dist != 40.) {
        rO += rD * dist;
        if (hMaterial == 4) {
            rD = reflect(rD, hitNormal);
            float dist2 = Y(); // Raymarch reflection
            if (dist2 != 40.) {
                rO += rD * dist2;
                color = S();
            }

            color = mix(color, vec3(0,.2,.2) * (rD.z+1.) * (rD.z+1.), min(dist2 / 40., 1.))
                        * vec3(.8,.5,.8); // Sky
        } else
            color = S();
    }

    color = mix(color, vec3(0,.2,.2) * (rD.z+1.) * (rD.z+1.), min(dist / 40., 1.)); // Sky

    if (.25-min(abs(1.-timeScaled),abs(127.-timeScaled))*.04+sin(timeScaled*400.)*.02 > .0)
        color = vec3(pow((color.x+color.y+color.z)*.33,1.)); // Black and white

    // Add screen flashes to final color
    gl_FragColor = vec4((timeScaled > 159.5 ?
                            vec3(0) :
                            timeScaled < 2. ?
                            vec3(0) :
                            color
                            + vec3(1,.1,1) * A(25.)
                            + vec3(.1,1,1) * A(35.)
                            + vec3(.1,1,1) * A(142.)
                            + vec3(1,.1,1) * A(150.))
                        + vec3(1,1,.1) * A(159.5)
                        + vec3(1,1,.1) * A(2.)
                        ,1);
}
```

Shader - Post-processing

This is a pretty simple post-processing shader that adds bloom, vignette and chromatic aberration. To add some flavor to the bloom, it’s calculated in an X-like shape. It also saves some processing time as it doesn’t have to sample as many pixels as box blur for example.

```
precision highp float;

uniform sampler2D S;

/*
    Applies following effects:
        Vignette
        Blur in X shape to get glow effect
        Chromatic aberration
*/

void main() {
    // Calculate screen position and vignette strength
    vec2 p = gl_FragCoord.xy;
    float dist, vignette = length((p-vec2(1280,720)*.5)/720.);
    vignette *= vignette;
    p/=vec2(1280,720);

    // Apply blur
    vec4 val, sum = vec4(0);
    for (int i=-9; i<10; i++) {
        val = texture2D(S, p - vec2(1) * float(i) * .0025);
        dist = 1. - (abs(float(i)) * .1);
        sum += val * dist * dist * dot(val,val) * .0132;
    }
    for (int i=-9; i<10; i++) {
        val = texture2D(S, p - vec2(-1,1) * float(i) * .0025);
        dist = 1. - (abs(float(i)) * .1);
        sum += val * dist * dist * dot(val,val) * .0132;
    }

    // Apply chromatic aberration
    sum += vec4(texture2D(S, p-vec2(.01,.0)*vignette).x,
                texture2D(S, p-vec2(.0,.01)*vignette).y,
                texture2D(S, p).z,
                0);

    // Apply vignette
    sum *=(1.-vignette);
    sum.w = 1.;
    gl_FragColor = sum;
}
```

Credits

These aren't comprehensive credits but I hope I remembered to name most of the people who shared their code, tricks and advice on the web and thus helped create this intro.

- http://www.bitsnbites.eu/?p=112
- http://daeken.com/superpacking-js-demos
- http://www.p01.org/
- http://www.ctrl-alt-test.fr/
- http://pouet.net/prod.php?which=59298
- http://www.iquilezles.org/
- https://developer.mozilla.org/en-US/demos/detail/soundbox-lite
- http://sonantlive.bitsnbites.eu/
- http://stackoverflow.com/questions/4200224/random-noise-functions-for-glsl
- https://www.shadertoy.com/
- http://pouet.net/prod.php?which=51074