JT Nimoy

Follow 1.3K Followers About

How to Extract Isolines in p5.js

JT Nimoy Jul 30, 2019 · 10 min read ★

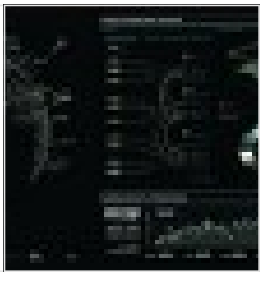
It has been a decade since I posted a portfolio entry about how I worked on Tron Legacy and then my site got slashdotted and redditted, ddossing me off my server, which was a long overdue upgrade anyway. My soon-to-be-fiancé who had just freshly fished me out of rehab, was egging me on as i pushed it to jtnimoy.net , and I really was not expecting the kind of audience that happened. In fact, you are still watching me today specifically because of that post alone, which is both creepy and fascinating business. I wish to respect Disney's interests in every way, but also continue with my artistic exploration of the area without feeling like I'm shackled to a mouse.

And so here I am contributing a 2D isosurface extraction library to p5js to commemorate the occasion in a rather self-congratulatory way. Furthermore, I hope that the cleverly named p5.marching.js will eventually rock 3D marching cubes, and perhaps 4D. So with future plans set down, let's mess around in p5js. I want to produce a similar luffa sponge that was seen in my own Tron work. I know half the internet tried it. So I'm jealous and want to join you.

Tron Legacy

jtnimoy - Tron Legacy I spent a half year writing software art to generate special effects for Tron Legacy, working at...

jtnimoy.net

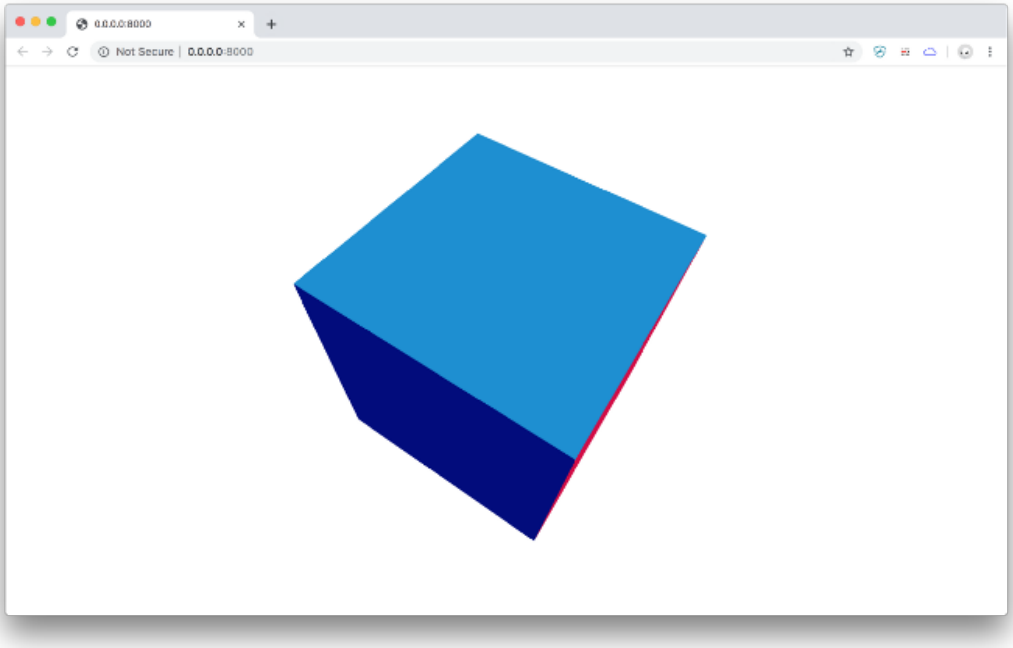


Check it out, Here is the most basic p5js sketch that makes a spinning 3D thingy.

```
push();

rotateZ(frameCount*0.03);

rotateY(frameCount*0.02);

rotateX(frameCount*0.01);
```

```
box( 400,400,400);

pop();
```

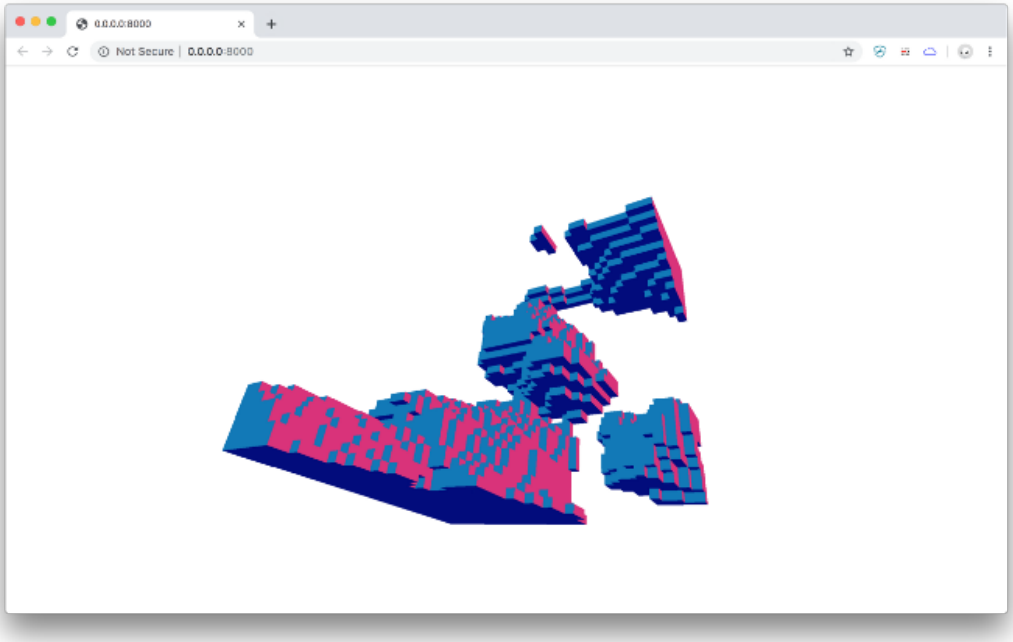
Notice the shader is already there. It's useful for debugging, but also elegant for general 3D. i love p5js. but please don't tell anyone I said that, i'm trying to be a punk.



Okay, now that we've established the basic cube, let's introduce some perlin noise, and render it as 3D volumetric. Notice the code just got excitingly more complex. Yes, I'm adding 2 noises together. If noise() were 4D, i would be able to properly do the thing. But I would rather not patch in an external js lib right now. I'd rather keep it p5js. So we settle for a couple calls to 3D noise, and genuflect.

```
let voxels = [];
let VOXEL_RESOLUTION = 32;

//construct 3D array
voxels = [];
for(let z=0;z<VOXEL_RESOLUTION;z++){
    let thisSlice = [];
    for(let y=0;y<VOXEL_RESOLUTION;y++){
        let thisRow = [];
        for(let x=0;x<VOXEL_RESOLUTION;x++){
            thisRow.push(0);
        }
        thisSlice.push(thisRow);
    }
    voxels.push(thisSlice);
}

//populate array
for(let z=0;z<VOXEL_RESOLUTION;z++){
    for(let y=0;y<VOXEL_RESOLUTION;y++){
        for(let x=0;x<VOXEL_RESOLUTION;x++){
            voxels[x][y][z] = noise(x*0.01+ frameCount * 0.01,y*0.01
+ frameCount * 0.02 ,z*0.01 + frameCount * 0.07) + noise(x*0.02 +
frameCount * 0.03,y*0.03+ frameCount * 0.05 ,z*0.04 + frameCount *
0.01);
```

```
        }
    }
}

push();
rotateZ(frameCount*0.07);
rotateY(frameCount*0.05);
rotateX(frameCount*0.03);
scale(16);
translate(-VOXEL_RESOLUTION/2,-VOXEL_RESOLUTION/2,-
VOXEL_RESOLUTION/2);
noFill();
for(let z=0;z<VOXEL_RESOLUTION;z++){
    for(let y=0;y<VOXEL_RESOLUTION;y++){
        for(let x=0;x<VOXEL_RESOLUTION;x++){
            push();
            translate(x,y,z);
            if(voxels[x][y][z] > 1.1)box(1,1,1);
            pop();
        }
    }
}
pop();
```

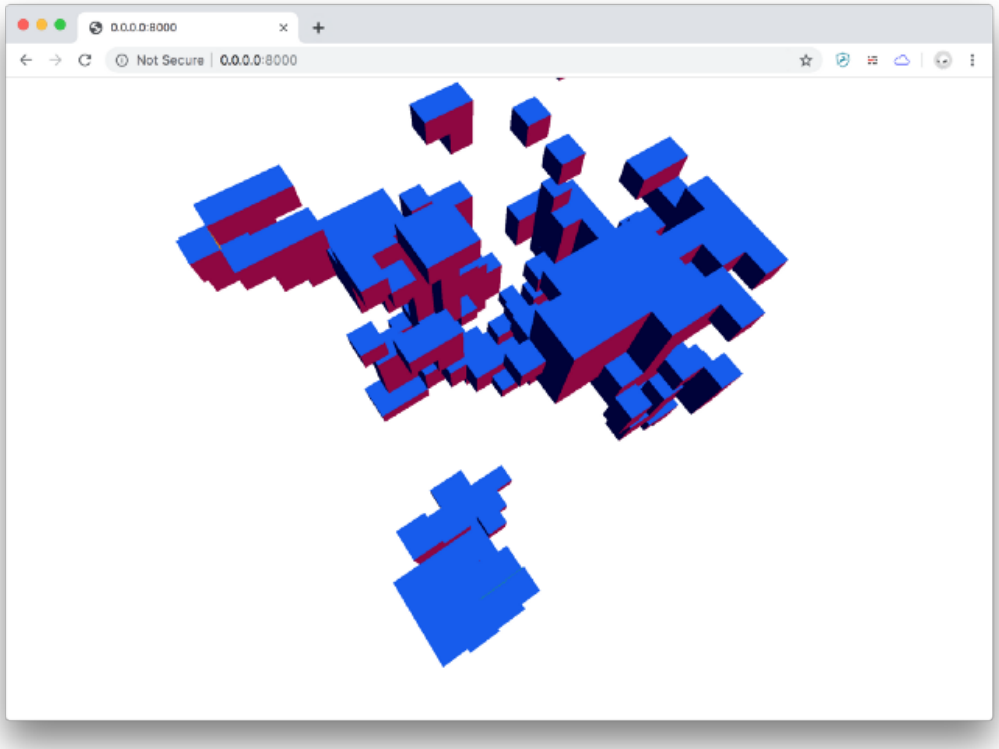


Wow, the organic complexity emerging here is curvatious and lovely to explore with the eye. If you are running the code, you see it animating — undulating in and out like a lava lamp that knows no constant volume or mass. It looks outer-dimensional. You are craving marijuana but your toddler is present in the room so nevermind.

Let us now adjust the noise() so we have something to work with while we apply the raster-to-vector part.

```
let VOXEL_RESOLUTION = 16; // lower resolution

...

voxels[x][y][z] = noise(x*0.1 + frameCount * 0.01 , y*0.1 +
frameCount * 0.02 ,z*0.1 + frameCount * 0.07 ) + noise(x*0.2 +
```

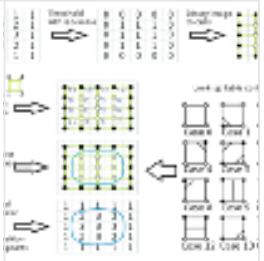
```
frameCount * 0.03 , y*0.3 + frameCount * 0.05 ,z*0.4 + frameCount *
0.01 ); // fiddle with noise function

...

scale(32); // change size
```



Marching squares

Marching squares is a computer graphics algorithm that generates contours for a two-dimensional scalar field...

en.wikipedia.org



No, we're not going to implement this in a medium.com article. Rather, I already did so we can move foward. Check out the new p5js library I wrote just for this article . . .

jtnimoy/marching

marching squares for p5.js conversión de trama a vector, isosuperficies 光栅到矢量转换, isosurfaces वेक्टर रूपांतरण के लिए...

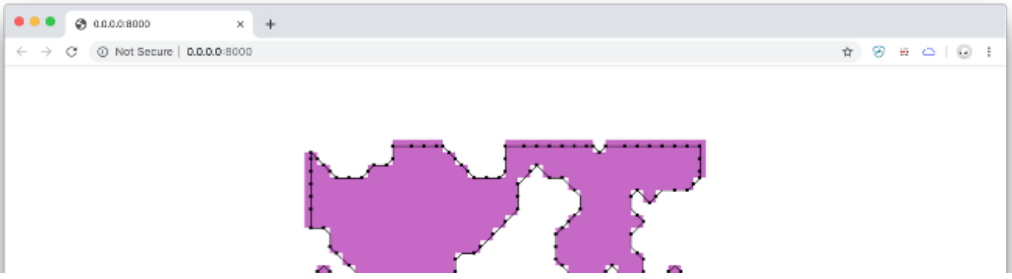
github.com

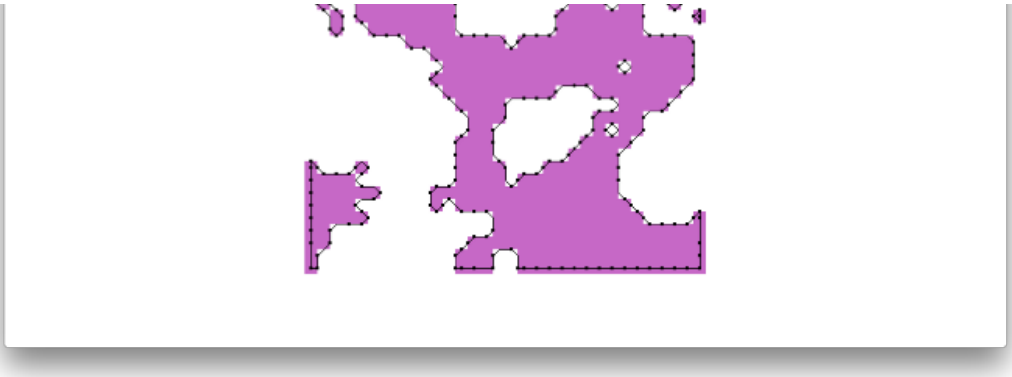
Now you can convert pixel data into vectors. Here is how to use it.

```
let lines = marchingSquares(voxels,1.1);

ret.map( function(i) { line(i[0], i[1], i[2], i[3]); } );
```

but more practically,

```
function draw() {

    background(255);

    //update just the first 2D slice
    let z = 0 ;
        for(let y=0;y<VOXEL_RESOLUTION;y++){
            for(let x=0;x<VOXEL_RESOLUTION;x++){
                voxels[z][y][x] =
                    noise(x*0.2 + frameCount * 0.003 , y*0.3 );
        }
    }

    push();
    translate(width/2 - (VOXEL_RESOLUTION * DISPLAY_SCALE) / 2,
              height/2 - (VOXEL_RESOLUTION * DISPLAY_SCALE) / 2);

    push();
    scale(DISPLAY_SCALE);
    let z = 0;

    //draw just the first 2D slice
    for(let y=0;y<VOXEL_RESOLUTION;y++){
        for(let x=0;x<VOXEL_RESOLUTION;x++){
            push();
            translate(x,y);
            fill(0,255,255);
            noStroke();
            if(voxels[z][y][x] >=  0.5 )rect(0,0,1,1);
            pop();
        }
    }
    pop();

    strokeWeight(1);
    let sc = DISPLAY_SCALE;//scale
    //draw the marching squAre output

    //run the marching squares
    let ret = marchingSquares(voxels[z] , 0.5 );
    ret.map( function(i) {
        noFill();
        stroke(0,0,0);
        line(i[0] * sc , i[1] * sc , i[2] * sc, i[3] * sc);
        fill(255,0,0);
        noStroke();
        rect( i[0] * sc-2 , i[1] * sc-2 , 4,4  );
    });

    pop();
}
```



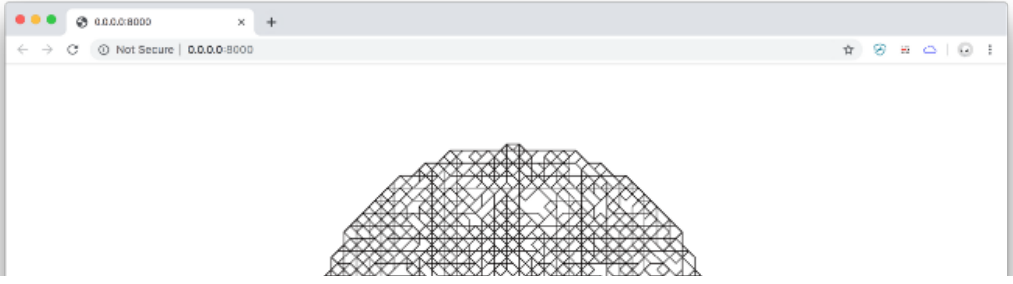


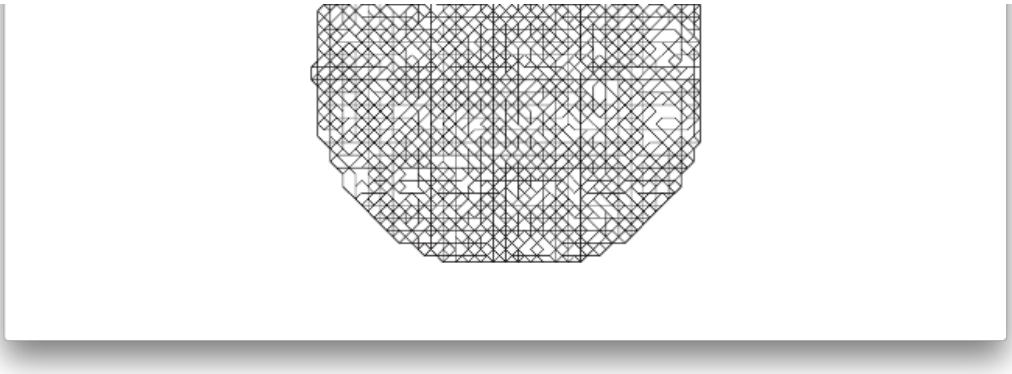
Now is a good time to introduce a circular filter.

```
let z = 0 ;
    for(let y=0;y<VOXEL_RESOLUTION;y++){
        for(let x=0;x<VOXEL_RESOLUTION;x++){
            //get distance from center
            let d = dist(x,y,VOXEL_RESOLUTION/2,VOXEL_RESOLUTION/2);
            //test using distance, against radius
            if(d > VOXEL_RESOLUTION/2){
                //cut it out
                voxels[z][y][x] = 0;
            }else{
                //it's inside the circle
                voxels[z][y][x] = noise(x*0.2 +
                        frameCount * 0.003 , y*0.3 );
            }
        }
   }
```

Now removing the point dots and disabling the raster part. And tweaking the colors again. As we do.

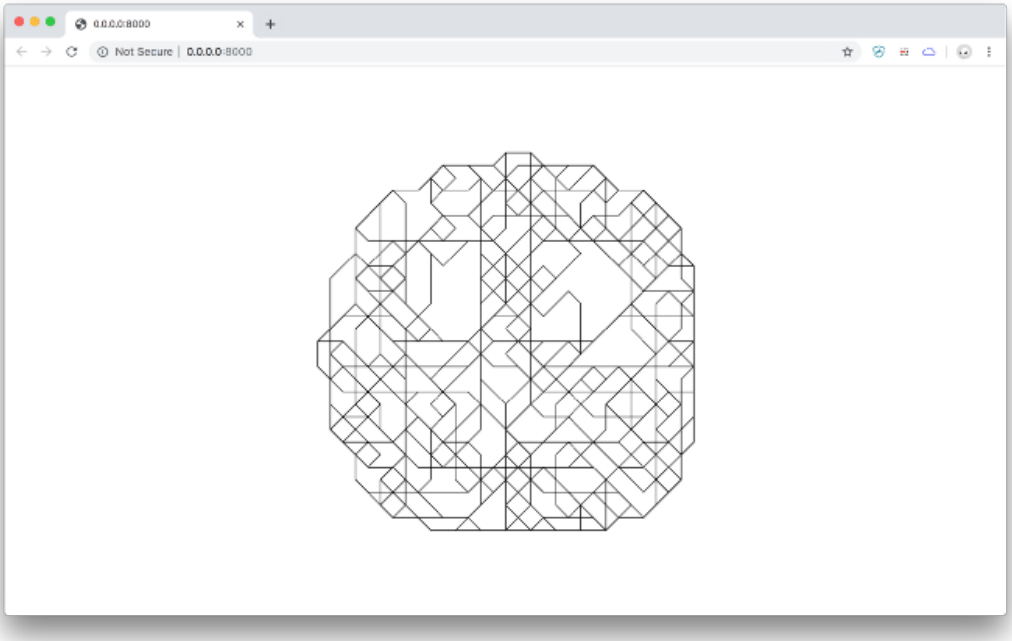
```
function draw() {

    background(255);

    for(let z=0;z<VOXEL_RESOLUTION;z++){
 for(let y=0;y<VOXEL_RESOLUTION;y++){
     for(let x=0;x<VOXEL_RESOLUTION;x++){
  let d = dist(x,y,VOXEL_RESOLUTION/2,VOXEL_RESOLUTION/2);
  if(d > VOXEL_RESOLUTION/2){
      //cut it out
      voxels[z][y][x] = 0;
  }else{
      //it's inside the circle
      voxels[z][y][x] = noise(x*0.2 + frameCount * 0.03 , y*0.2,
z*0.5 );
  }

     }
 }
    }

    push();
    translate(width/2 - (VOXEL_RESOLUTION * DISPLAY_SCALE) / 2,
       height/2 - (VOXEL_RESOLUTION * DISPLAY_SCALE) / 2);

    push();
    scale(DISPLAY_SCALE);
    /*z = 0;
    for(let y=0;y<VOXEL_RESOLUTION;y++){
 for(let x=0;x<VOXEL_RESOLUTION;x++){
     push();
     translate(x,y);
     fill(200,100,200);
     noStroke();
     if(voxels[z][y][x] >=  0.5 )rect(0,0,1,1);
     pop();
 }
    }*/
    pop();

    strokeWeight(1);
    let sc = DISPLAY_SCALE;//scale
    //draw the marching squAre output
    for(let z=0;z<VOXEL_RESOLUTION;z++){

 let ret = marchingSquares(voxels[z] , 0.5 );//run the marching
squares
 ret.map( function(i) {
     noFill();
     stroke(z * (VOXEL_RESOLUTION/255),0,0);
     line(i[0] * sc , i[1] * sc , i[2] * sc, i[3] * sc);
     //fill(0);
     //noStroke();
     //rect( i[0] * sc-2 , i[1] * sc-2 , 4,4  );
 });
    }
    pop();
}
```





Oof that's pretty dense. Well, it's all the z-slices layered on one another so it's no wonder. Although this looks nice, let's first lower the resolution so it's not choking the browser on my tiny backpacking macbook. Yes, I'm still homeless.

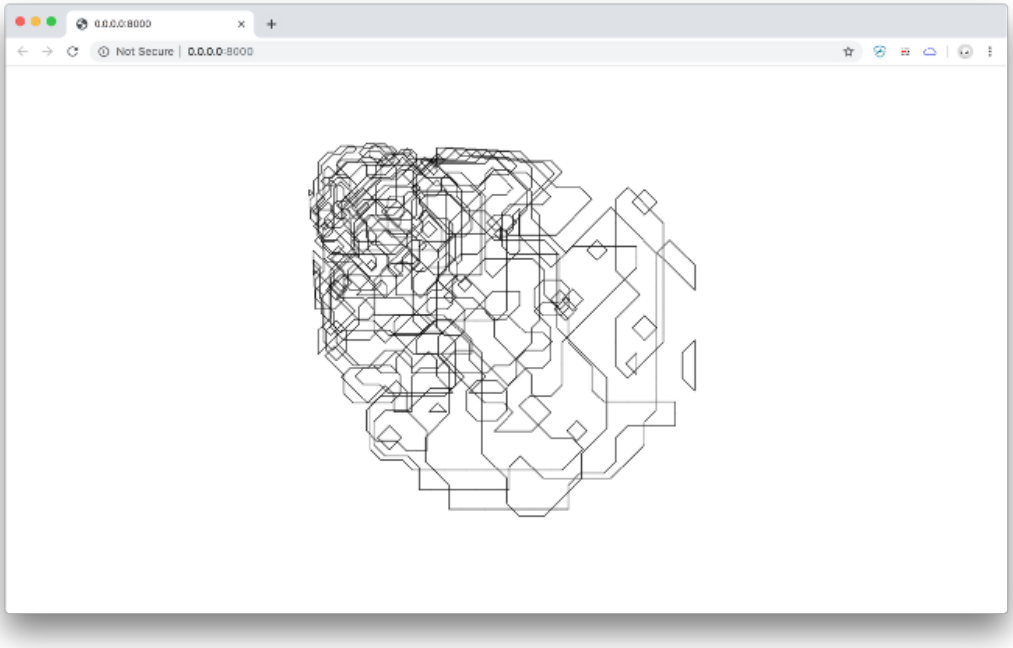
```
let VOXEL_RESOLUTION = 16;

let DISPLAY_SCALE = 32;
```



Great, now let's introduce some rudimentary foreshortening in absence of `line()` in WEBGL mode. We need to render our lines in 2D mode. That's ok, there are plenty of techniques for perspective besides the one people consider the proper one. Here's how I'm choosing to do it. Quick and dirty 3D.

```
let FORESHORTENING = 0.05;

...

ret.map( function(i) {
            noFill();
            stroke(z * (VOXEL_RESOLUTION/255),0,0);
            let x1 = i[0] + (0-i[0]) * z*FORESHORTENING;
```

```
            let y1 = i[1] + (0-i[1]) * z*FORESHORTENING;
            let x2 = i[2] + (0-i[2]) * z*FORESHORTENING;
            let y2 = i[3] + (0-i[3]) * z*FORESHORTENING;

line(x1 * sc , y1 * sc , x2 * sc, y2 * sc);

});
```



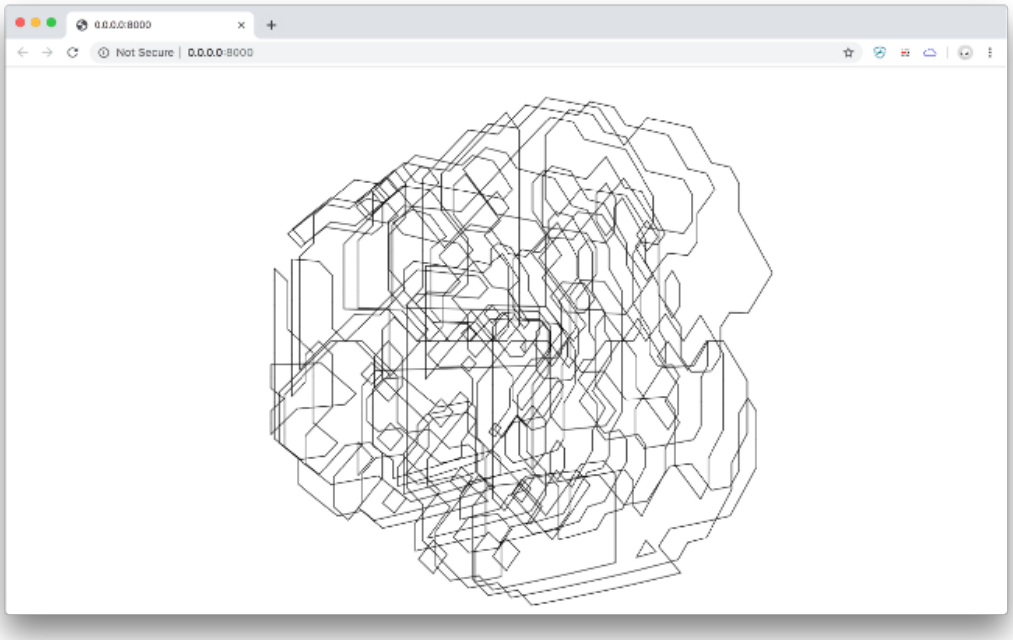
Great, now with that foreshortening in place, we will rotate about the Y axis (so rotate XZ). In the past, I've called that turntabling. I guess you could also call that trollying. Whatever you want.

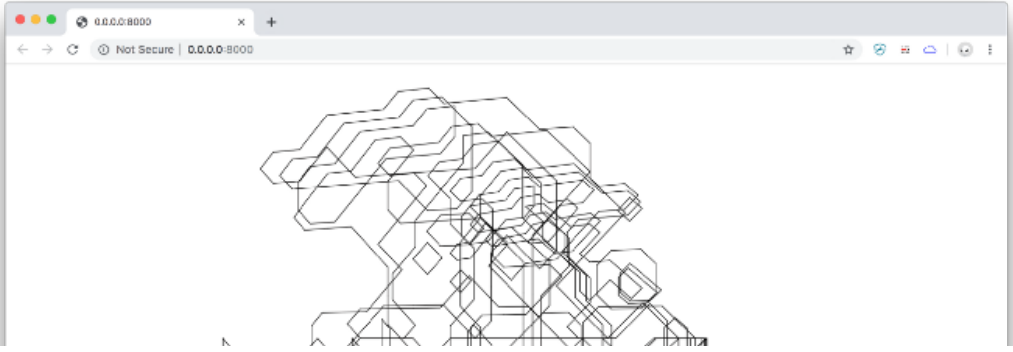
Note the use of trigonometry. I'm converting from cartesian to polar, increasing theta to rotate, then converting back to cartesian. I look up these equations. No one should ever be forced to do them on a whiteboard while wearing business casual wear. That is simply not how code is written. Sorry not sorry.

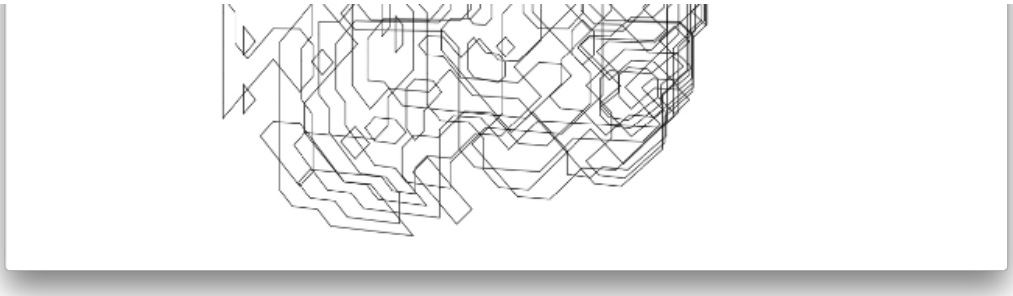
```
ret.map( function(i) {
     noFill();
     stroke(z * (VOXEL_RESOLUTION/255.0),0,0);

     let x1 = i[0];
     let y1 = i[1];
     let x2 = i[2];
     let y2 = i[3];
     let z1 = z - halfRes;
     let z2 = z - halfRes;

//translate to origin for pivoting
     x1-= halfRes;
     y1-= halfRes;
     x2-= halfRes;
     y2-= halfRes;

//convert to polar
     let radius1 = dist(x1,z1,0,0);
     let radius2 = dist(x2,z2,0,0);
```

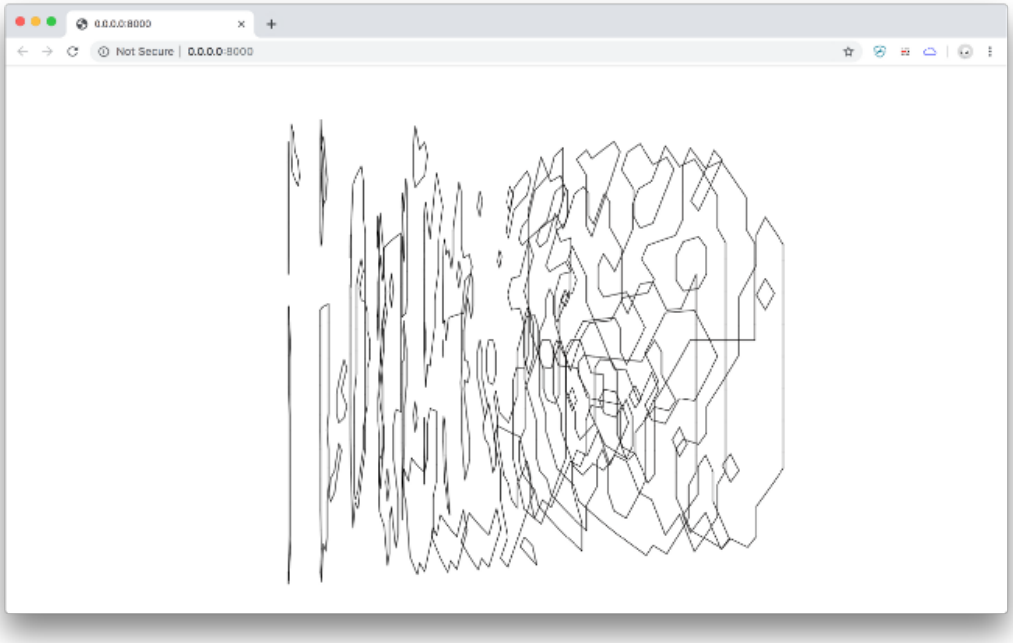
```
      let theta1 = atan2(x1,z1);
      let theta2 = atan2(x2,z2);

//rotate
      theta1 += frameCount * 0.1;
      theta2 += frameCount * 0.1;

//convert back to cartesian
      x1 = radius1 * sin(theta1);
      x2 = radius2 * sin(theta2);
      z1 = radius1 * cos(theta1);
      z2 = radius2 * cos(theta2);

      //translate back from origin into center framing
      x1+= halfRes;
      y1+= halfRes;
      x2+= halfRes;
      y2+= halfRes;

      //project to 2-point perspective
      x1 = x1 + (halfRes-x1) * z1*FORESHORTENING;
      y1 = y1 + (halfRes-y1) * z1*FORESHORTENING;
      x2 = x2 + (halfRes-x2) * z2*FORESHORTENING;
      y2 = y2 + (halfRes-y2) * z2*FORESHORTENING;
      line(x1 * sc , y1 * sc , x2 * sc, y2 * sc);

});
```

Now we're looking 3D.



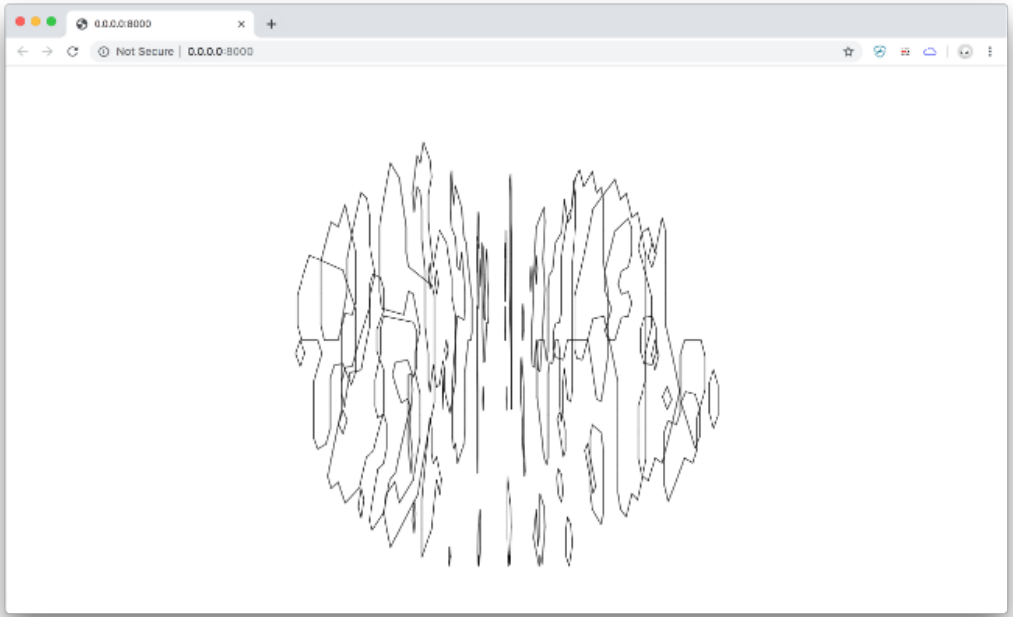


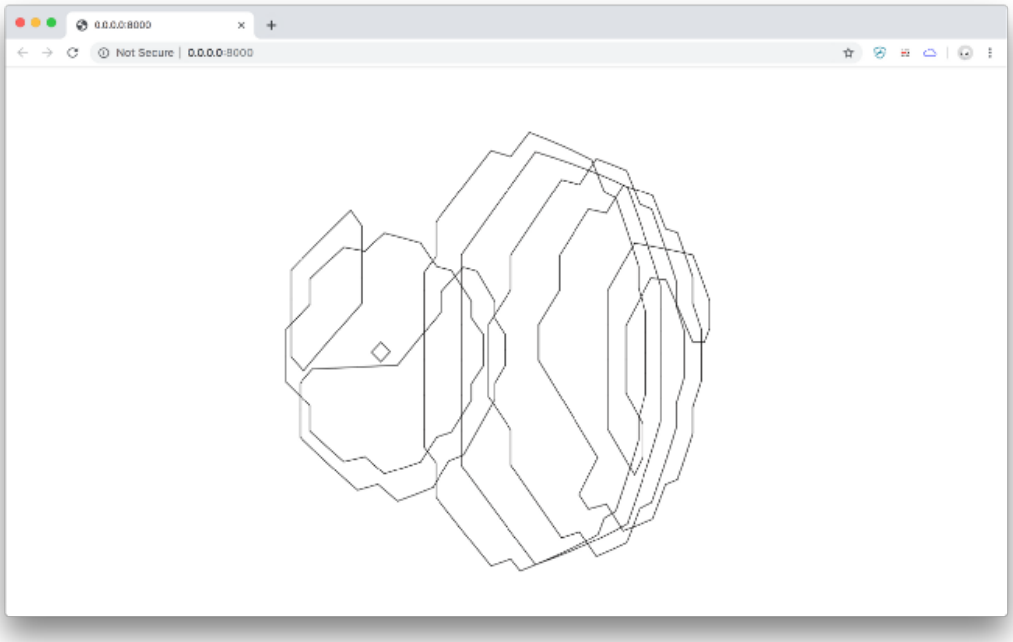


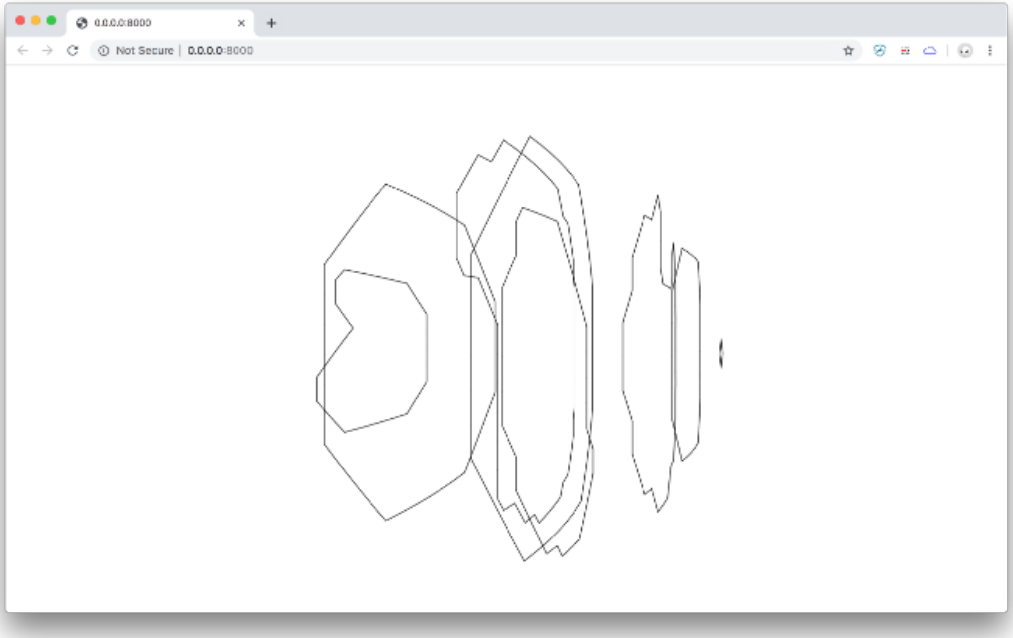


At this point, we should update the spherical culling distance function so that it includes Z. This will take us from a cylinder to a sphere.

```
let d =
dist(x,y,z,VOXEL_RESOLUTION/2,VOXEL_RESOLUTION/2,VOXEL_RESOLUTION/2);
```



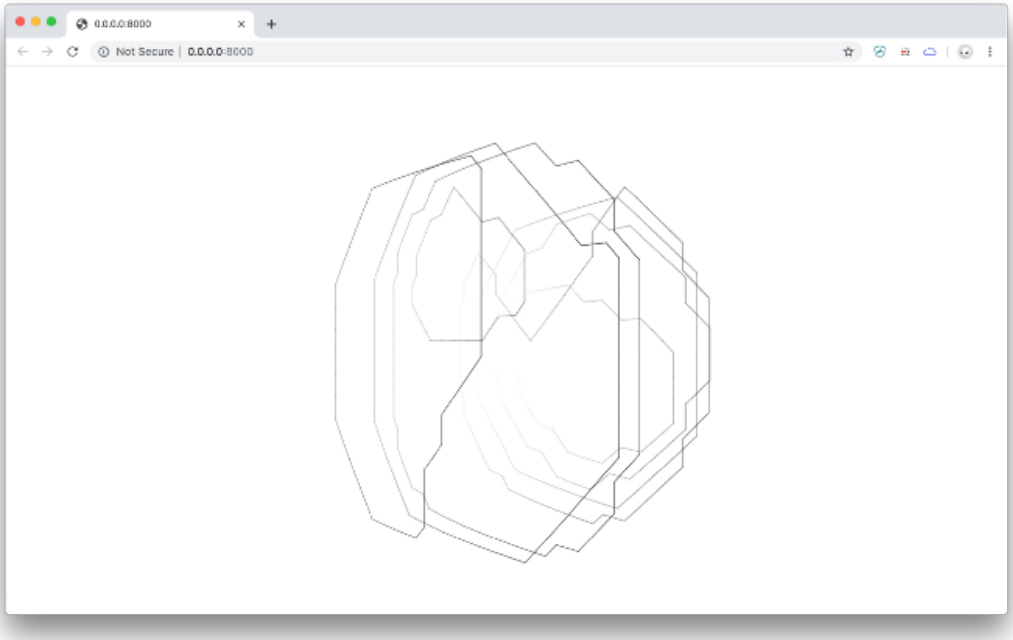


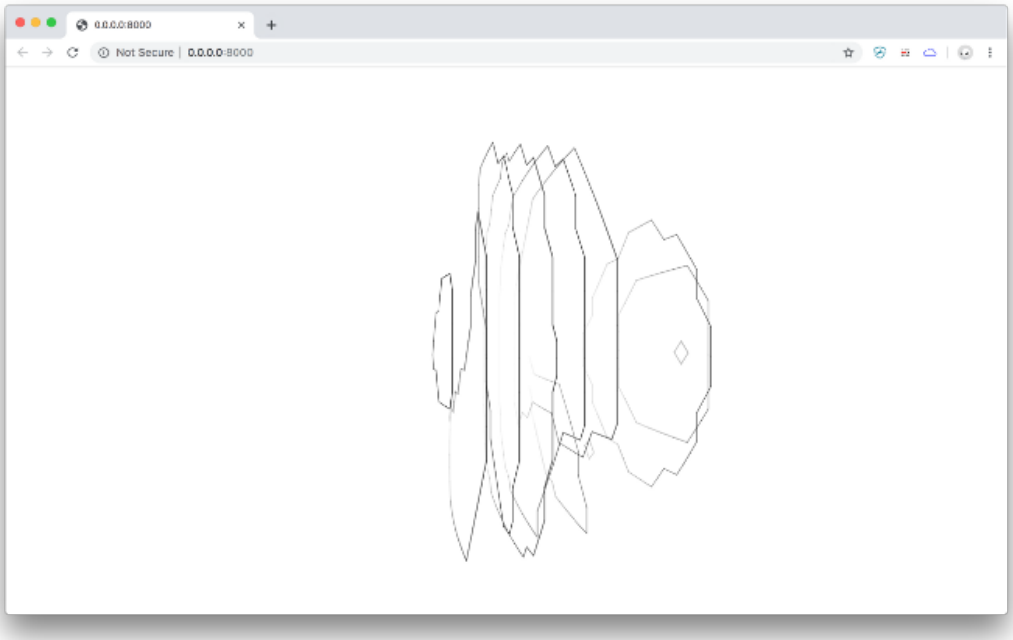


Okay, it's looking like the movie content! So now we have 3D line rendering, raster-to-vector conversion of noise volume, and spherical volumetric culling. Let's add one more depth effect: fog.

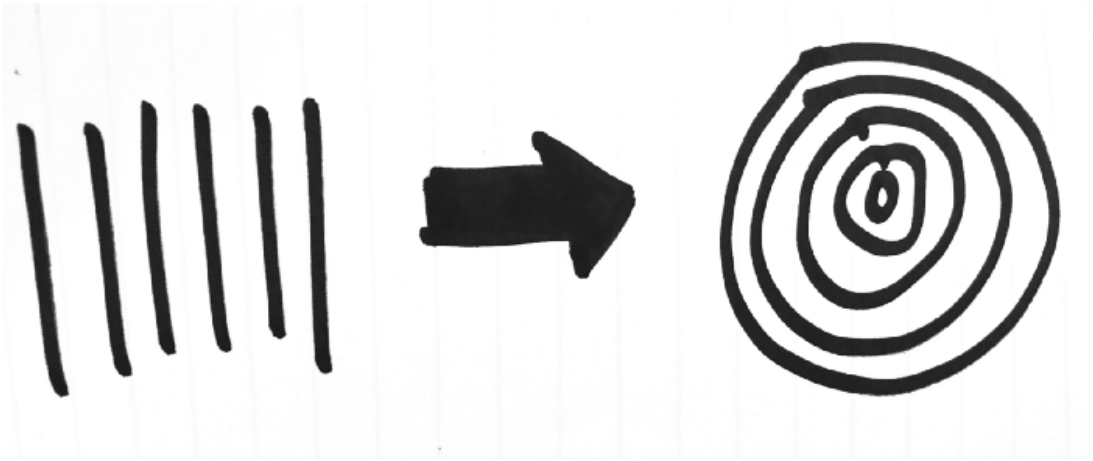
```
function draw() {
    blendMode(NORMAL);

...

let halfRes = VOXEL_RESOLUTION * 0.5;
blendMode(MULTIPLY);
strokeWeight(1);
```

```
...

stroke((z1+halfRes) * 16);
line(x1 * sc , y1 * sc , x2 * sc, y2 * sc);
```

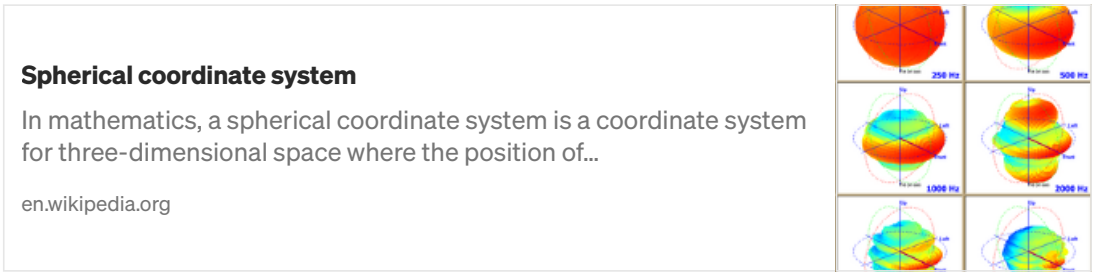




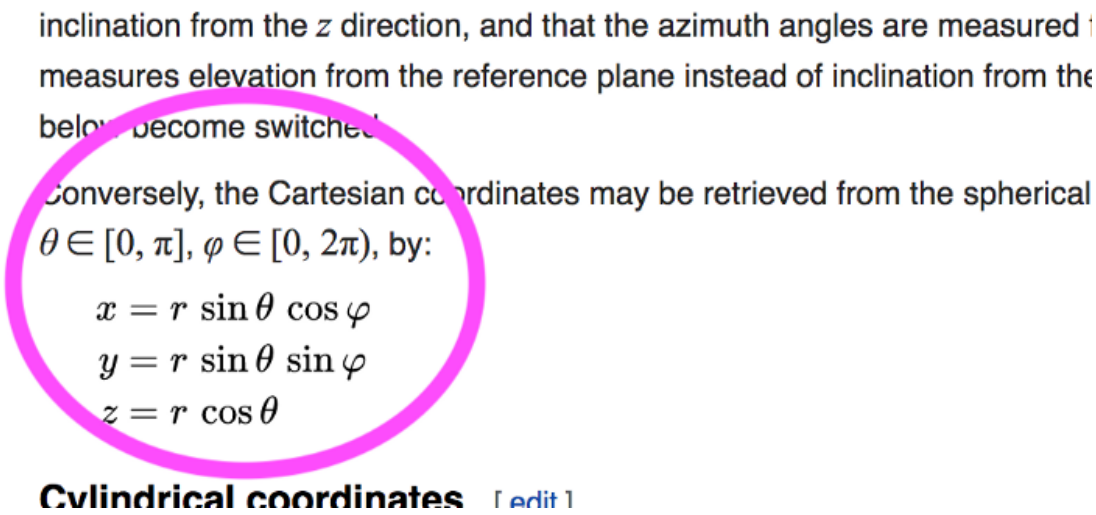
Ah, Remember Toxi's metaballs from back in the day? Ok, now let's change the cartesian isosurface to a sphere-coordinate-space isosurface, so the lines wrap around like onion skin layers.



so first, we need to provide 2D slices that have been sampled off the globe space. Time for sphere coordinates. A lot of hobbyists have skipped this part in their attempts to duplicate what they saw.



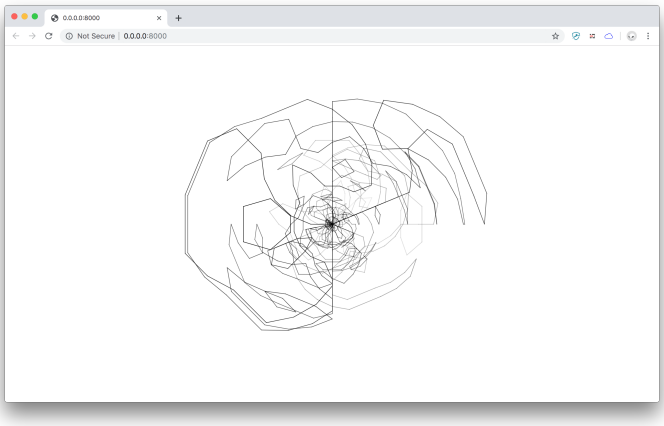
Specifically, I'm going to that article so i can be reminded of this trig, and not much more:

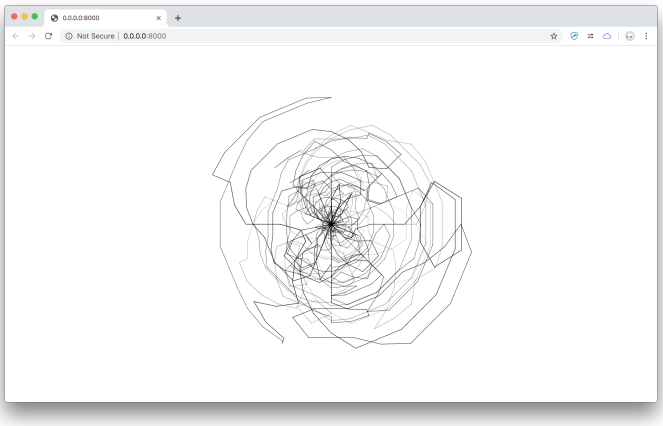


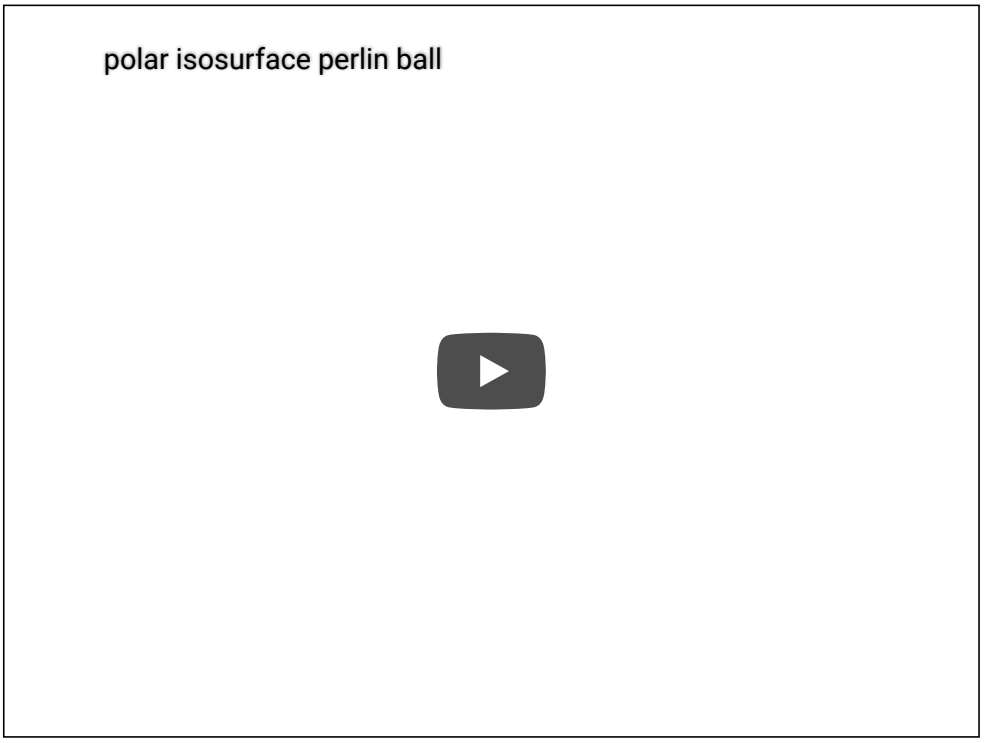
This is how we will interpret the XYZ data as bulbous today . . .

```
//convert to onion skin
            let x1n = (x1 / float(VOXEL_RESOLUTION)) * Math.PI;
            let x2n = (x2 / float(VOXEL_RESOLUTION)) * Math.PI;
            let y1n = (y1 / float(VOXEL_RESOLUTION)) * Math.PI * 2;
            let y2n = (y2 / float(VOXEL_RESOLUTION)) * Math.PI * 2;

x1p = z1 * sin(x1n) * cos(y1n);
            y1p = z1 * sin(x1n) * sin(y1n);
            z1p = z1 * cos(x1n)

x2p = z2 * sin(x2n) * cos(y2n)
            y2p = z2 * sin(x2n) * sin(y2n)
            z2p = z2 * cos(x2n)

x1=x1p;
            x2=x2p;
            y1=y1p;
            y2=y2p;
            z1=z1p;
            z2=z2p;

//translate back from origin into center framing
            x1+= halfRes;
            y1+= halfRes;
            x2+= halfRes;
            y2+= halfRes;

...

let VOXEL_RESOLUTION = 16;

let DISPLAY_SCALE = 50;
```

```
let FORESHORTENING = 0.05;

...

voxels[z][y][x] = noise(x*0.2 + frameCount * 0.008 , y*0.2, z*0.5 );
```







There are many ways to spherify, and this one . . . is fast and bulbous. Got me?

Conclusion

I had a lot of fun authoring p5.marching.js , I got a kick out of seeing the look again after a long time of not wanting to look at it, and I felt free in writing this article to just be myself with you. For the future, let's think about how Ai starts to blend and mix with vector based artwork like this. I'm tired of deepdream and other deeply *raster* projects.

Please let me know what you think of this article in the comments. I am still homeless, broke, and harassed all day irl. Anything helps.

Web Development Processing Foundation P5js JavaScript Webgl