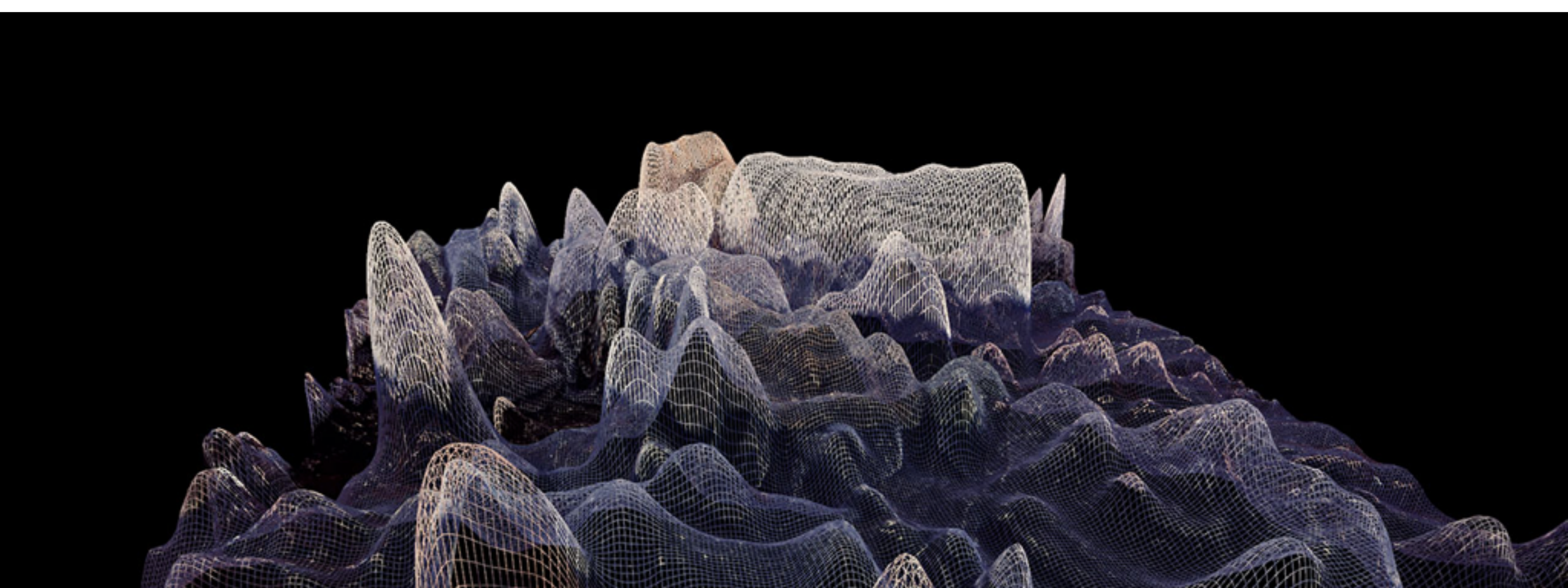
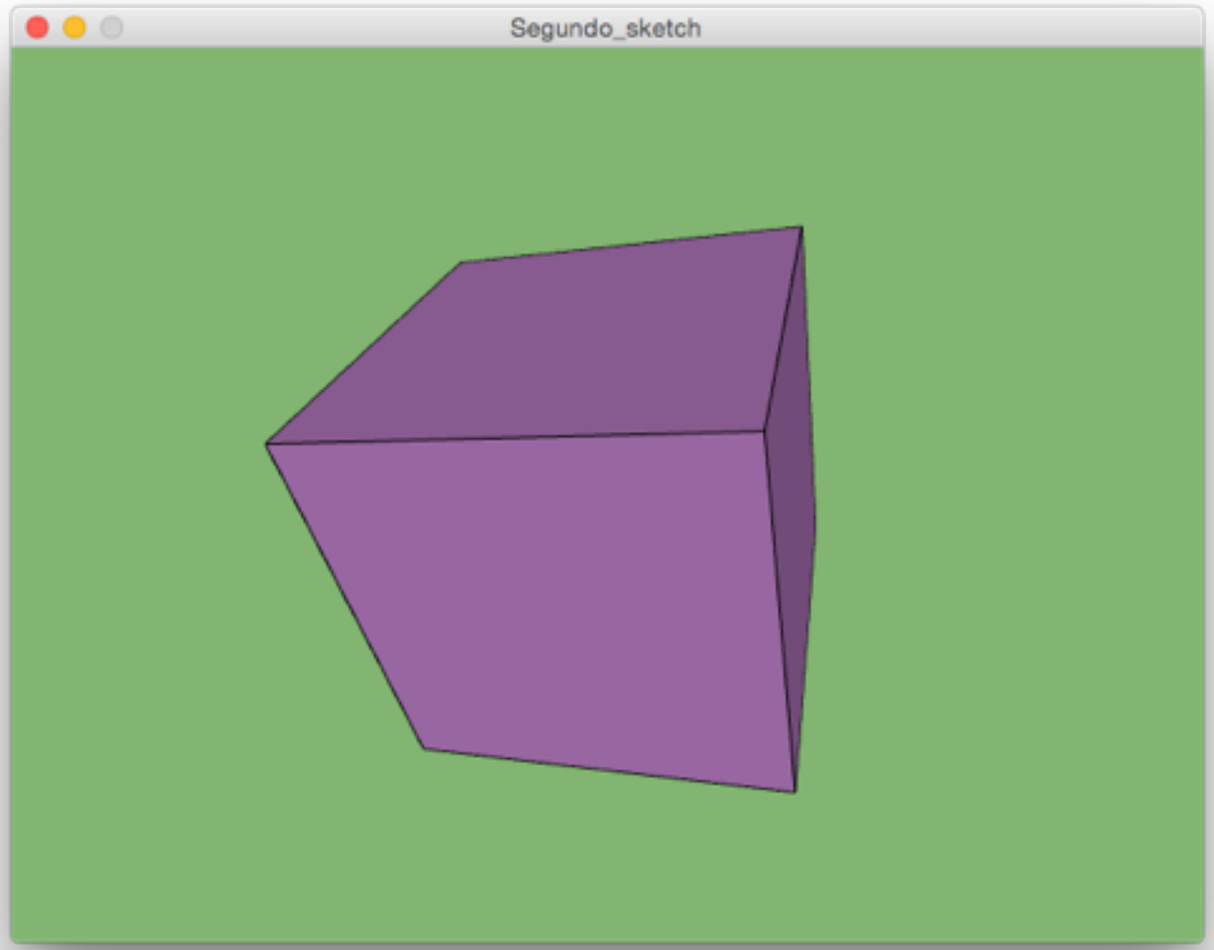
Render 3D y shaders en Processing

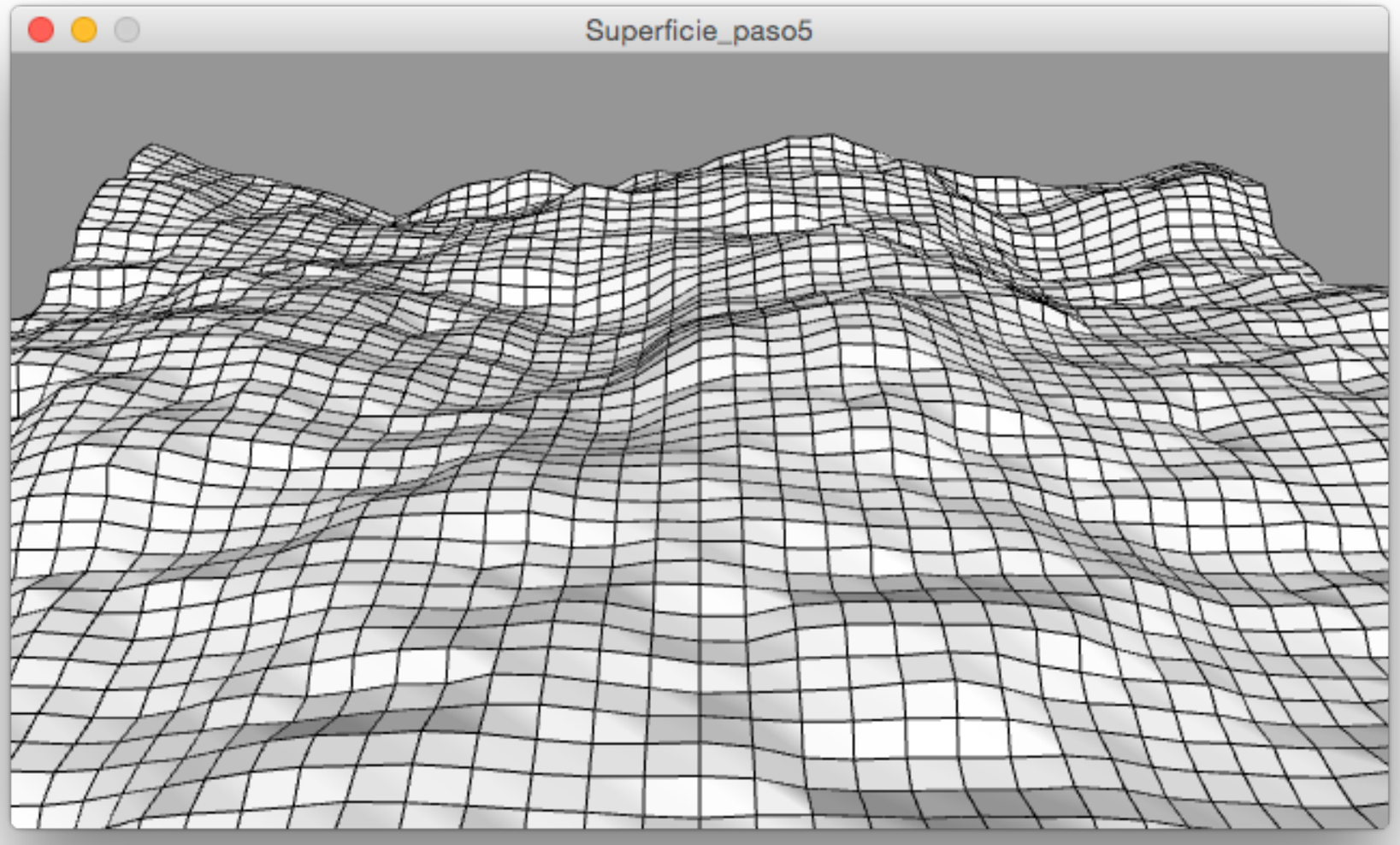


Un “hola mundo” 3D

```
float angulo = 0;
void setup() {
  size(640, 480, P3D);
  fill(#AD71B7);
}

void draw() {
  background(#81B771);

  lights();

  translate(width/2, height/2);
  rotateY(angulo);
  rotateX(angulo*2);
  box(200);

  angulo += 0.01;
}
```



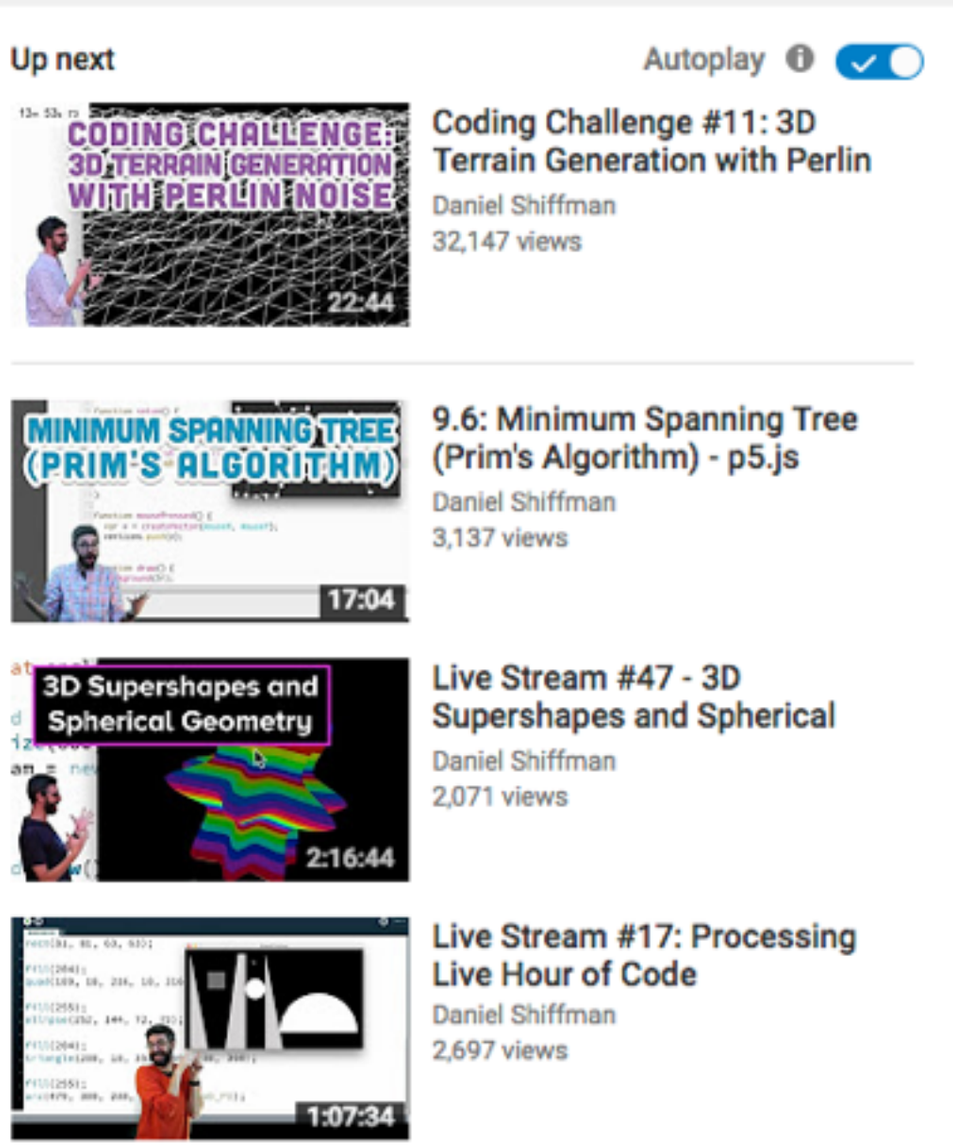
Algo un poco más complejo: una superficie 3-D



El código terminado…

```
void setup() {
  size(640, 360, P3D);
  smooth(8);
  frameRate(120);
}

void draw() {
  background(150);
  lights();
  translate(width/2, height/2);
  rotateX(QUARTER_PI);
  beginShape(QUADS);
  for (int i = 0; i < 50; i++) {
    for (int j = 0; j < 50; j++) {
      float x0 = map(i, 0, 50, -width/2, width/2);
      float y0 = map(j, 0, 50, -height/2, height/2);
      float x1 = x0 + width/50.0;
      float y1 = y0 + height/50.0;
      float t = 0.0001 * millis();
      float z1 = 100 * noise(0.1 * i, 0.1 * j, t);
      float z2 = 100 * noise(0.1 * (i + 1), 0.1 * j, t);
      float z3 = 100 * noise(0.1 * (i + 1), 0.1 * (j + 1), t);
      float z4 = 100 * noise(0.1 * i, 0.1 * (j + 1), t);
      vertex(x0, y0, z1);
      vertex(x1, y0, z2);
      vertex(x1, y1, z3);
      vertex(x0, y1, z4);
    }
  }
  endShape();
}
```

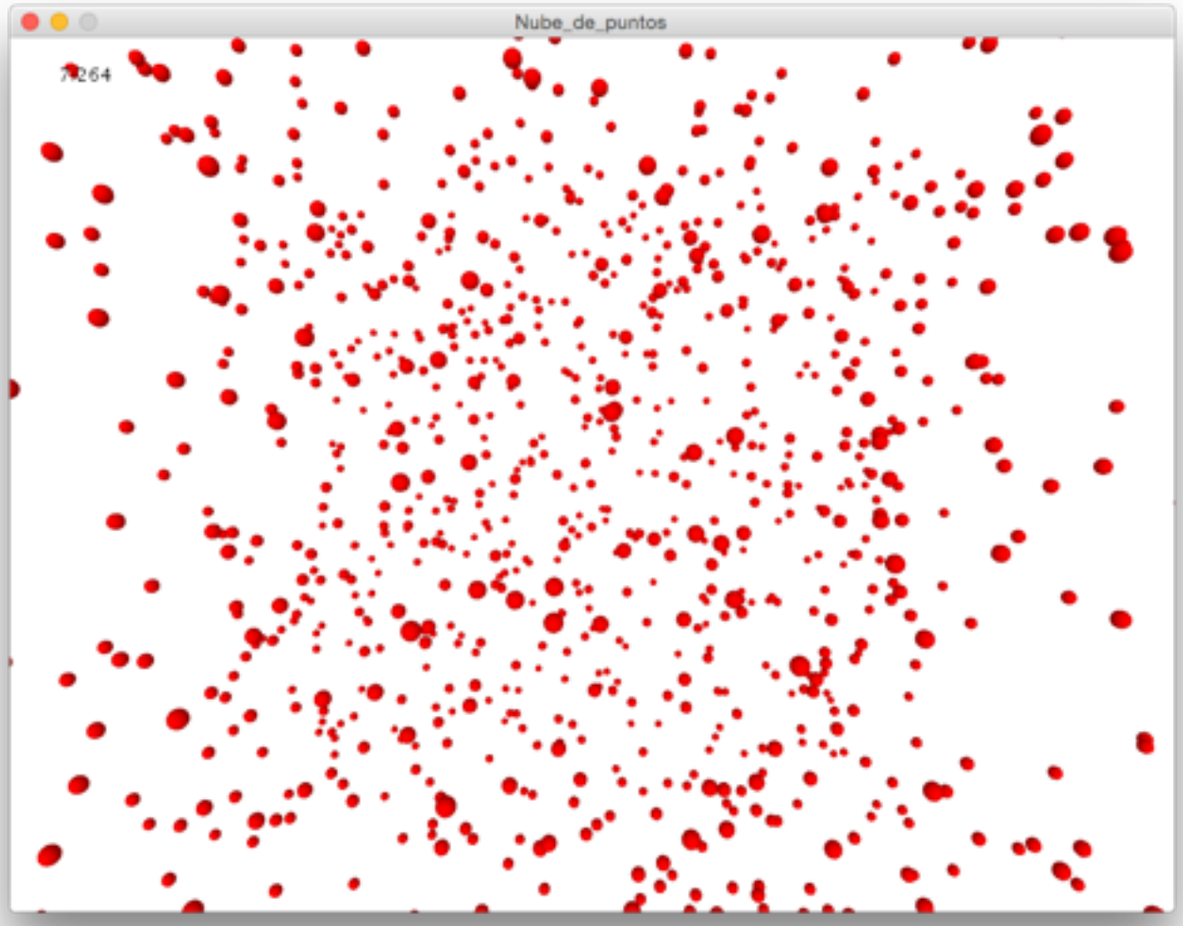
Inspirado por Daniel Shiffman y Coding Rainbow



https://www.youtube.com/watch?v=ELpZW62HGVs

Navegando grandes volúmenes de datos

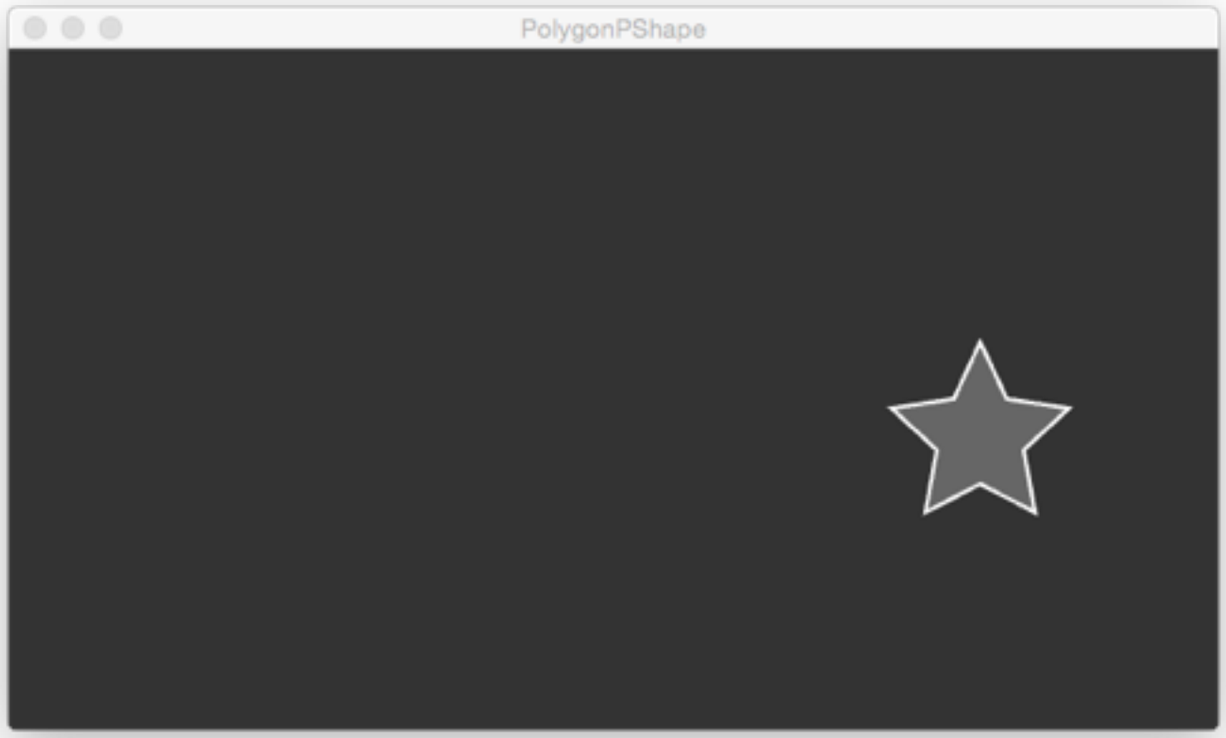
```
PVector[] coordenadas;
void setup() {
  size(800, 600, P3D);

  String[] lineas = loadStrings("puntos.txt");
  coordenadas = new PVector[lineas.length];
  for (int i = 0; i < lineas.length; i++) {
    String linea = lineas[i];
    String[] valores = linea.split(" ");
    float x = float(valores[0]);
    float y = float(valores[1]);
    float z = float(valores[2]);
    coordenadas[i] = new PVector(x, y, z);
  }
  noStroke();
}

void draw() {
  background(255);

  fill(0);
  text(frameRate, 30, 30);

  fill(255, 0, 0);
  lights();
  translate(width/2, height/2, -1000);
  rotateY(map(mouseX, 0, width, 0, TWO_PI));
  for (PVector v: coordenadas) {
    pushMatrix();
    translate(v.x, v.y, v.z);
    sphere(10);
    popMatrix();
  }
}
```



fps < 10?

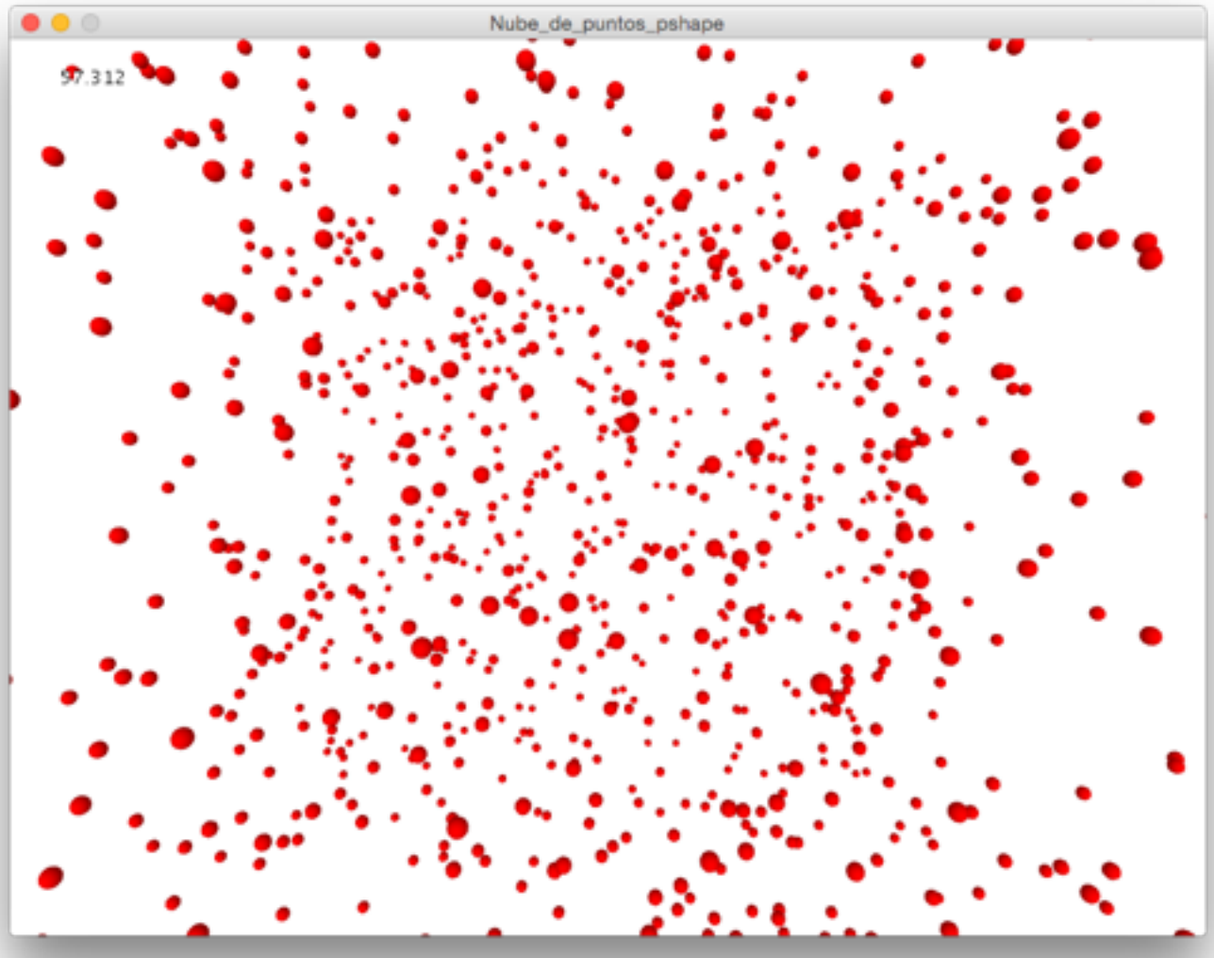
Empaquetando geometría con PShape

```
PShape star;

void setup() {
  size(640, 360, P2D);

  star = createShape();
  star.beginShape();
  star.fill(102);
  star.stroke(255);
  star.strokeWeight(2);
  star.vertex(0, -50);
  star.vertex(14, -20);
  star.vertex(47, -15);
  star.vertex(23, 7);
  star.vertex(29, 40);
  star.vertex(0, 25);
  star.vertex(-29, 40);
  star.vertex(-23, 7);
  star.vertex(-47, -15);
  star.vertex(-14, -20);
  star.endShape(CLOSE);
}

void draw() {
  background(51);
  translate(mouseX, mouseY);
  shape(star);
}
```



Usando PShape para optimizar el código

```
PVector[] coordenadas;
PShape esferas;
void setup() {
  size(800, 600, P3D);

  esferas = createShape(GROUP);
  String[] lineas = loadStrings("puntos.txt");
  coordenadas = new PVector[lineas.length];
  for (int i = 0; i < lineas.length; i++) {
    String linea = lineas[i];
    String[] valores = linea.split(" ");
    float x = float(valores[0]);
    float y = float(valores[1]);
    float z = float(valores[2]);
    coordenadas[i] = new PVector(x, y, z);
    PShape esfera = createShape(SPHERE, 10);
    esfera.translate(x, y, z);
    esfera.setFill(color(255, 0, 0));
    esfera.setStroke(false);
    esferas.addChild(esfera);
  }
  noStroke();
}

void draw() {
  background(255);

  fill(0);
  text(frameRate, 30, 30);

  fill(255, 0, 0);
  lights();
  translate(width/2, height/2, -1000);
  rotateY(map(mouseX, 0, width, 0, TWO_PI));
  shape(esferas);
}
```



fps = 60!

Unnamed soundsculpture, por Daniel Franke (onformative)

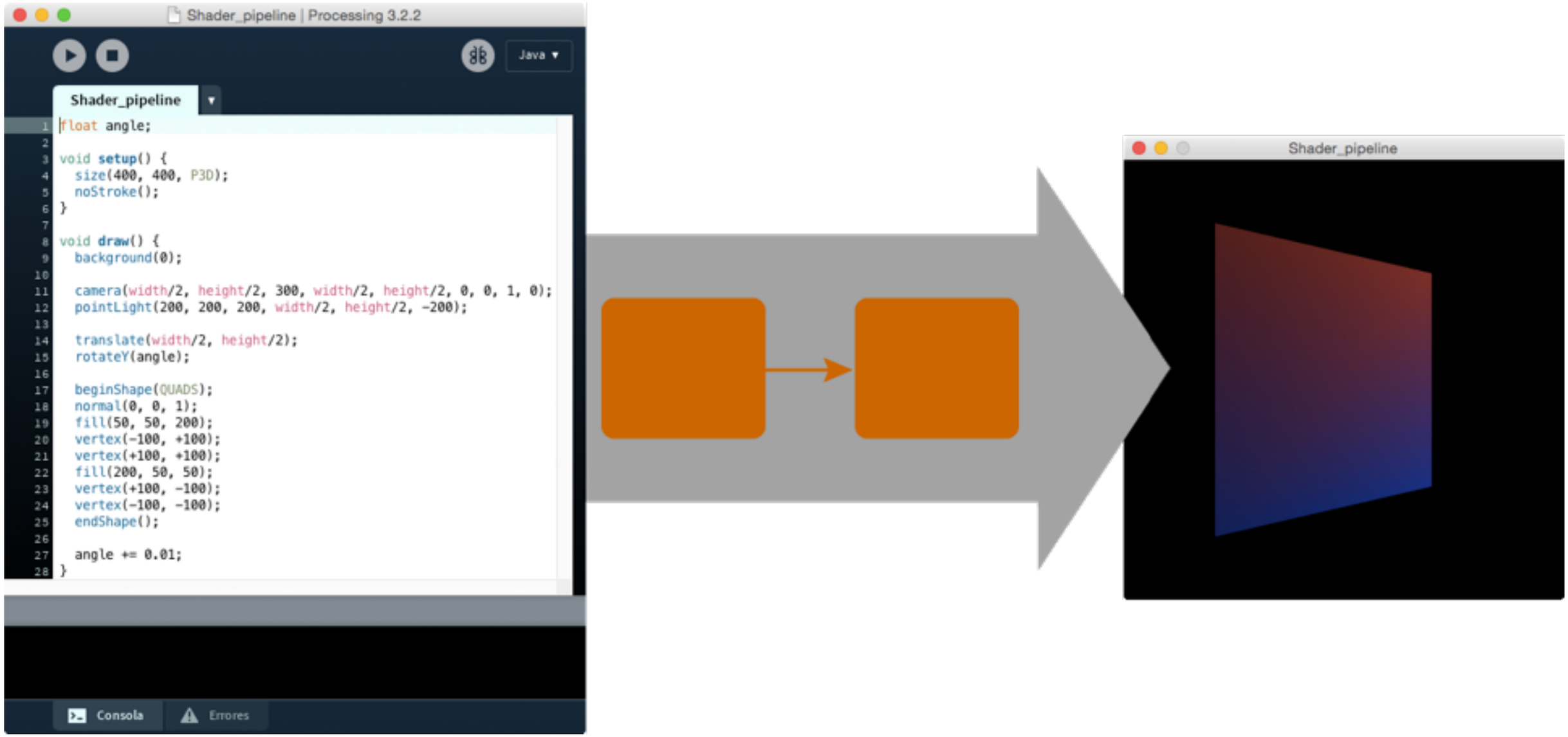
https://vimeo.com/38840688

Documentación

https://vimeo.com/38505448

¡GLSL Shaders!

El pipeline gráfico



Processing/CPU Shaders/GPU Pantalla

```
float angle;

void setup() {
  size(400, 400, P3D);
  noStroke();
}

void draw() {
  background(0);
  camera(width/2, height/2, 300, width/2, height/2, 0, 0, 1, 0);
  pointLight(200, 200, 200, width/2, height/2, 200);
  translate(width/2, height/2);
  rotateY(angle);

  beginShape(QUADS);
  normal(0, 0, 1);
  fill(50, 50, 200);
  vertex(-100, +100);
  vertex(+100, +100);
  fill(200, 50, 50);
  vertex(+100, -100);
  vertex(-100, -100);
  endShape();

  angle += 0.01;
}
```

projection matrix

lighting parameters

modelview matrix

uniform variables

vertex attributes: positions, colors and normals

connectivity

a. vertex shader

b. primitive assembly

varyings: colors and normals

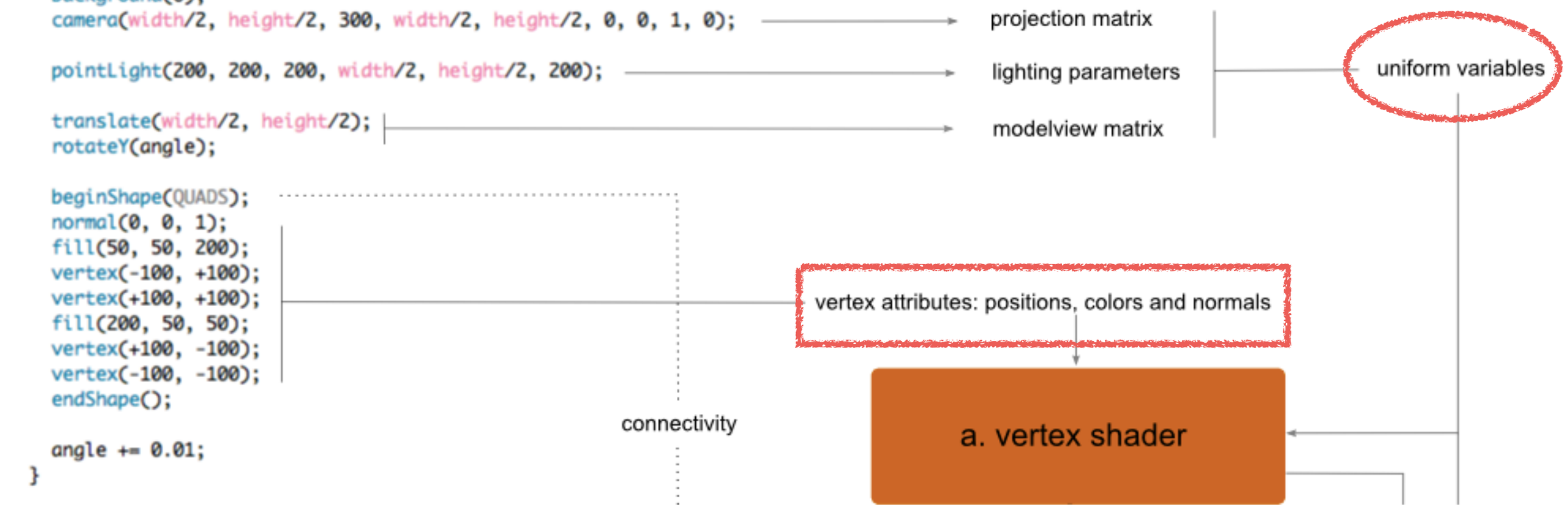
c. rasterization and interpolation

d. fragment shader

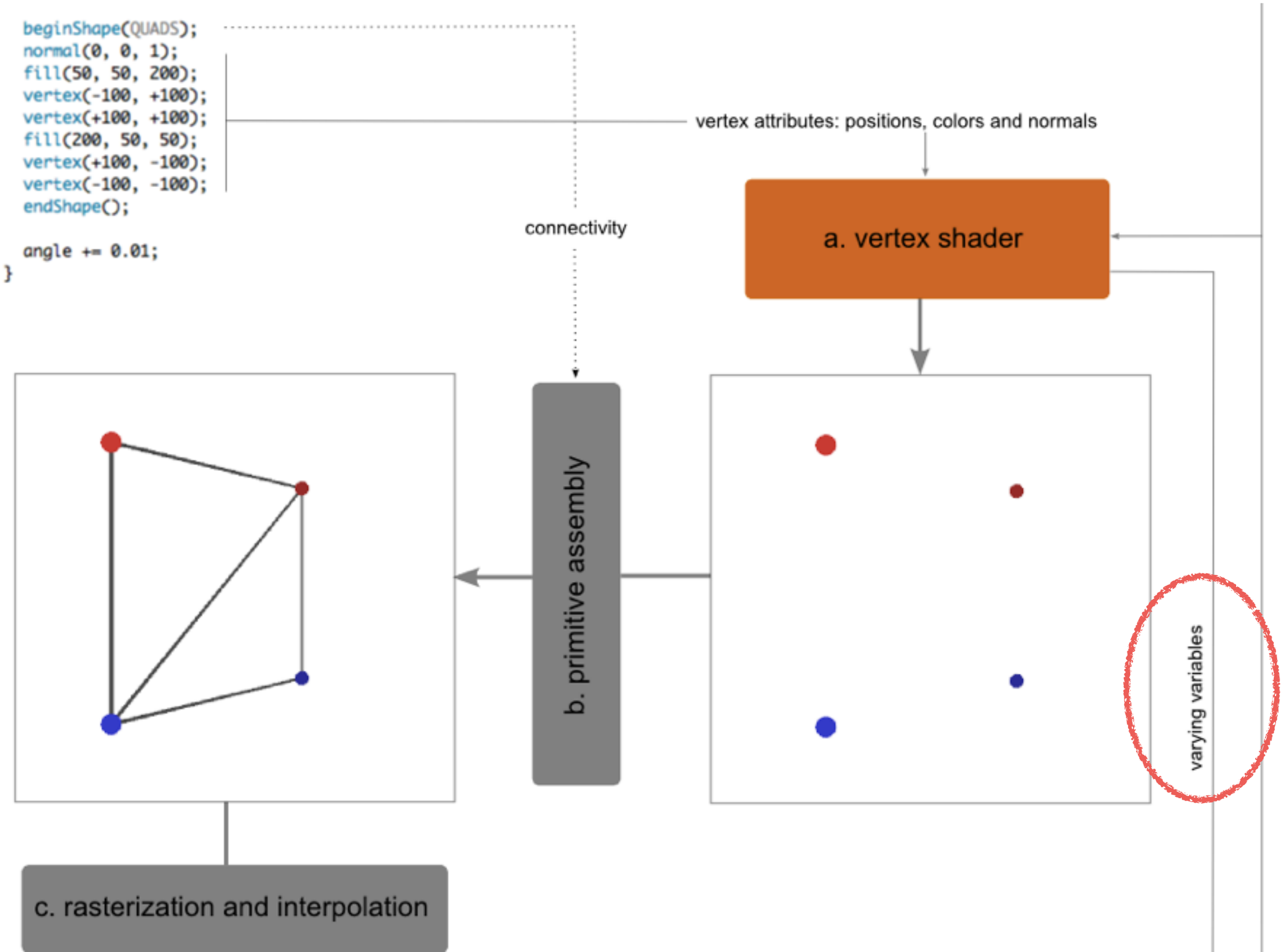
e. blending, depth oclussion

Variables uniformes y atributos de vértices

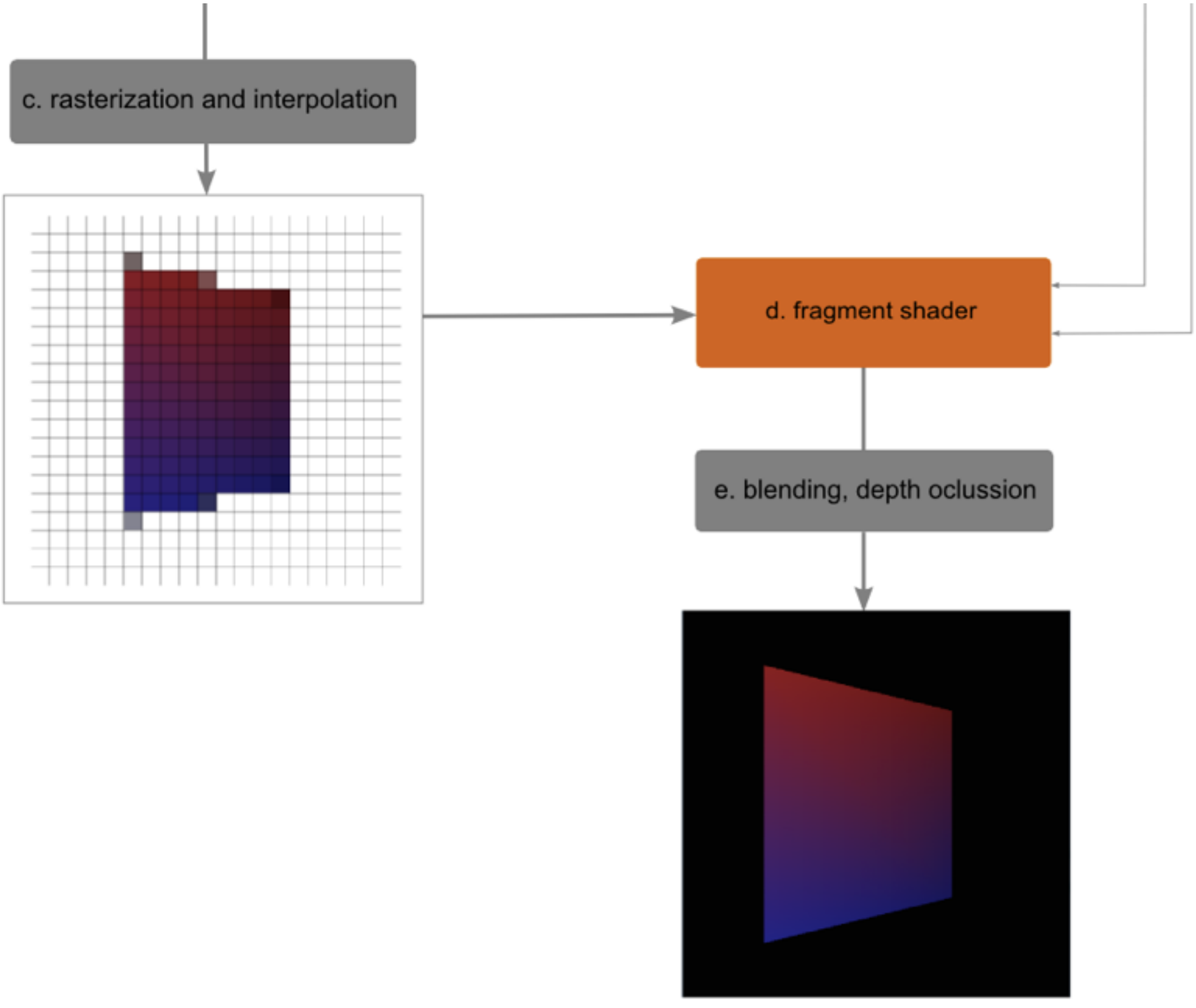
```
float angle;

void setup() {
  size(400, 400, P3D);
  noStroke();
}

void draw() {
  background(0);
  camera(width/2, height/2, 300, width/2, height/2, 0, 0, 1, 0);

  pointLight(200, 200, 200, width/2, height/2, 200);

  translate(width/2, height/2);
  rotateY(angle);

  beginShape(QUADS);
  normal(0, 0, 1);
  fill(50, 50, 200);
  vertex(-100, +100);
  vertex(+100, +100);
  fill(200, 50, 50);
  vertex(+100, -100);
  vertex(-100, -100);
  endShape();

  angle += 0.01;
}
```



El shader de vértices

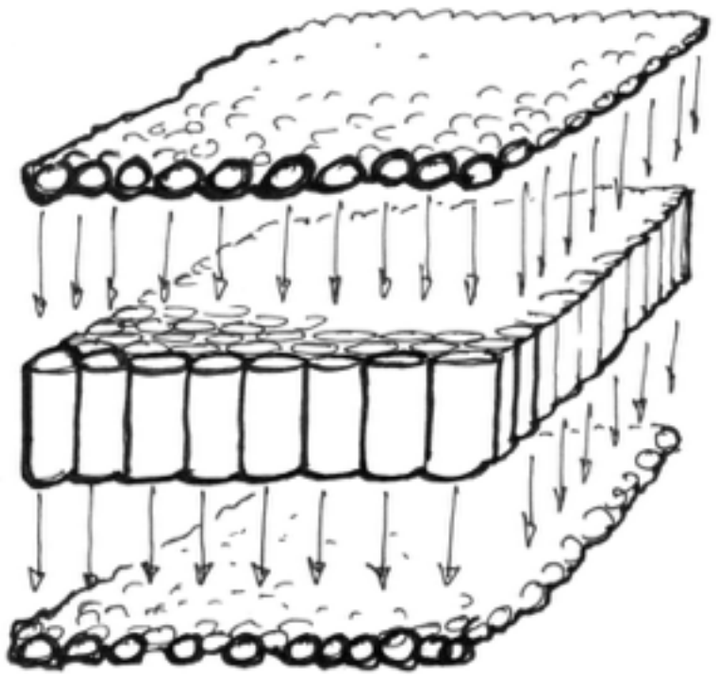


El shader de fragmentos



Parallelismo en el GPU

CPU



GPU

(cortesía de Patricio Gonzalez Vivo, The Book of Shaders)

Tutorial de shaders en Processing

Processing

Cover
Download
Exhibition

Reference
Libraries
Tools
Environment

Tutorials
Examples
Books
Handbook

Overview
People

Shop

Shaders

Andres Colubri

(The code for this tutorial is available here.)

Everything that Processing draws on the screen with the P2D and P3D renderers is the output of an appropriate "default shader" running behind the scenes. Processing handles these default shaders transparently so that the user doesn't need to worry about them, and she or he can continue to use the well-known drawing functions and expect the same visual results as with previous versions of Processing. However, Processing incorporates a new set of functions and variables that allows advanced users to replace the default shaders with her or his own. This opens up many exciting possibilities: rendering 3D scenes using more sophisticated lighting and texturing algorithms, applying image post-processing effects in real-time, creating complex procedural objects that would be very hard or impossible to generate with other techniques, and sharing shader effects between desktop, mobile, and web platforms with minimal code changes.

https://processing.org/tutorials/pshader/

Código del tutorial en GitHub

codeanticode / pshader-tutorials

Unwatch 4 | Star 23 | Fork 15

Code | Issues 0 | Pull requests 0 | Projects 0 | Wiki | Pulse | Graphs | Settings

Tutorial for the shader API in Processing 2.0 — Edit

21 commits | 1 branch | 0 releases | 1 contributor

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

codeanticode added texture matrix in Cinder project — Latest commit d40eb02 on Nov 26, 2015

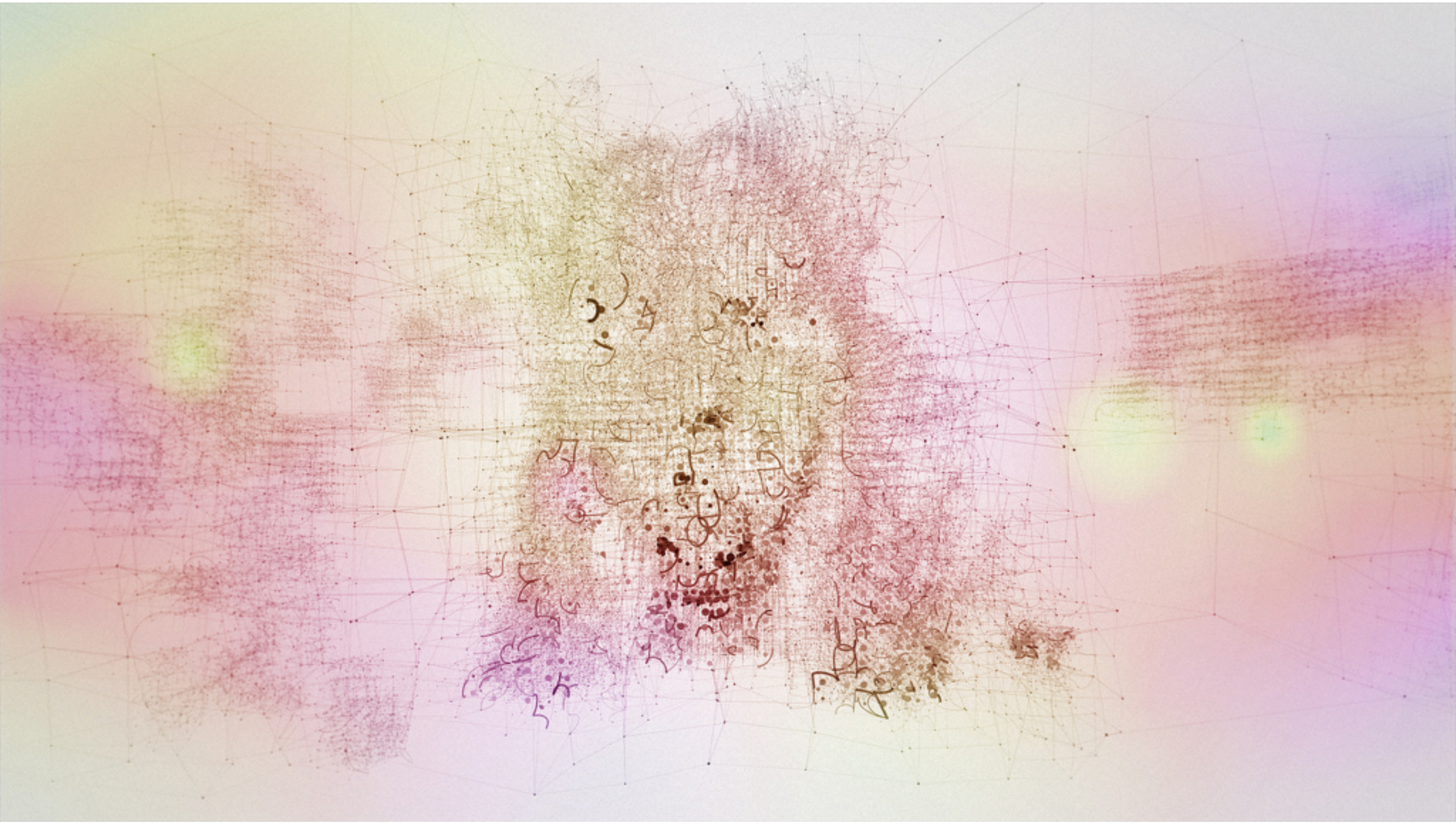
intro	removed DS_Store	4 years ago
ports	added texture matrix in Cinder project	11 months ago
README	uploaded first version of intro examples	4 years ago

README

```
Code examples that accompany the PShader tutorials for the Processing website. They might move into the
built-in examples later.
```

https://github.com/codeanticode/pshader-tutorials

Los videos musicales de Raven Kwok



https://vimeo.com/ravenkwok