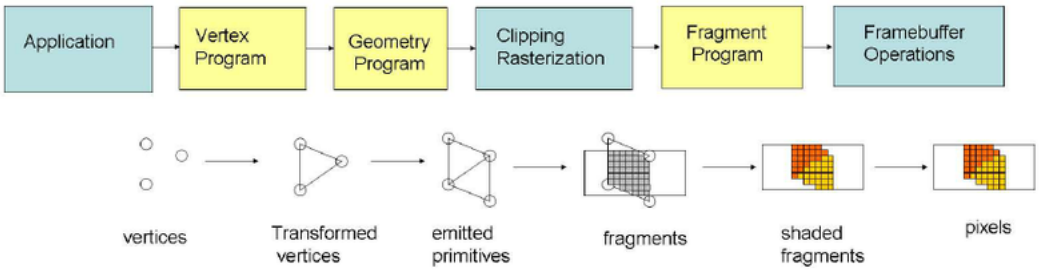
document of shader

GLSL shaders

shader: input data ——> shaders to transform——> output graphics

graphics pipeline in OpenGL



P5.js

Inspired by examples from Elise : pixellation of the picture
 and noting the dark space, I was considering if I could compare each pixel in the two pictures in finding difference game, then the differences could be automatically noted out.

p5.js Web Editor

A web editor for p5.js, a JavaScript library with the goal of making coding accessible to artists, designers, educators, and beginners.

https://editor.p5js.org/elisec/sketches/Z-2yzhOYx

p5.js Web Editor

A web editor for p5.js, a JavaScript library with the goal of making coding accessible to artists, designers, educators, and beginners.

https://editor.p5js.org/elisec/sketches/-Hj-3P-x8

```
var pixelsize = 1;

let img;
let img1, img2;
let bgcolor = 0;
let capture;

function preload() {
  img = loadImage("assets/pic.jpg");

}

function setup() {
  // disables scaling for retina screens which can create inconsistent scaling between displays
  pixelDensity(1);

  createCanvas(img.width, img.height);
  image(img, 0, 0);


  drawgrid();
}

function draw() {
  //background(220);
}

function drawgrid() {
```

```
  //this draws a grid of shapes based on the canvas pixels

  loadPixels();

  background(255);
  noStroke();
  //trying to traversal from left to right, and compare pixel(i,j) with(i+width/2,j);
  for (var j = 0; j < height; j += pixelsize) {
    for (var i = 0; i < width / 2; i += pixelsize) {
      let index1 = 4 * (i + j * width);
      let r1 = pixels[index1];
      let g1 = pixels[index1 + 1];
      let b1 = pixels[index1 + 2];
      fill(r1, g1, b1, 255);
      ellipse(i, j, pixelsize, pixelsize);
      let index2 = 4 * (i + width / 2 + j * width);
      let r2 = pixels[index2];
      let g2 = pixels[index2 + 1];
      let b2 = pixels[index2 + 2];
      fill(r2, g2, b2, 255);
      ellipse(i + width / 2, j, pixelsize, pixelsize);

      //compare pixels
      let av1 = (r1 + g1 + b1) / 3;
      let av2 = (r2 + g2 + b2) / 3;
      let dif = abs(av1 - av2);
      if (dif > 45) {
        fill(0, 0, 255, 20);
        ellipse(i, j, pixelsize * 10, pixelsize * 10);
      }
    }
  }
}
```

The most important part I learned from this project is to traversal from left to right and compare left side with right side.

the traversal calculation is as below:

```
for (var j = 0; j < height; j += pixelsize) {
    for (var i = 0; i < width / 2; i += pixelsize) {
      let index1 = 4 * (i + j * width);
      let r1 = pixels[index1];
      let g1 = pixels[index1 + 1];
      let b1 = pixels[index1 + 2];
      //fill(r1, g1, b1, 255);
      //ellipse(i, j, pixelsize, pixelsize);
      let index2 = 4 * (i + width / 2 + j * width);
      let r2 = pixels[index2];
      let g2 = pixels[index2 + 1];
      let b2 = pixels[index2 + 2];
      //fill(r2, g2, b2, 255);
      //ellipse(i + width / 2, j, pixelsize, pixelsize);
```

Shadertoy

> Image shaders implement the **mainImage()** function in order to generate the procedural images by computing a color for each pixel. This function is expected to be called once per pixel, and it is responsability of the host application to provide the right inputs to it and get the output color from it and assign it to the screen pixel. The prototype is:
> **void mainImage(out vec4 fragColor, in vec2 fragCoord);**
> where **fragCoord** contains the pixel coordinates for which the shader needs to compute a color. The coordinates are in pixel units, ranging from 0.5 to resolution-0.5, over the rendering surface, where the resolution is passed to the shader through the ***iResolution*** uniform (see below). The resulting color is gathered in **fragColor** as a four component vector, the last of which is ignored by the client. The result is gathered as an "out" variable in prevision of future addition of multiple render targets.

> **Input Uniforms**
>
> Shader can be fed with different types of per-frame static information by using the following uniform variables:
> uniform vec3 iResolution;
> uniform float iTime;
> uniform float iTimeDelta;
> uniform float iFrame;
> uniform float iChannelTime[4];
> uniform vec4 iMouse;
> uniform vec4 iDate;
> uniform float iSampleRate;
> uniform vec3 iChannelResolution[4];
> uniform samplerXX iChanneli;

functions

origin point: from left bottom

clamp(): Constrain a value to lie between two further values;

`clamp()` returns the value of `x` constrained to the range `minVal` to `maxVal`. The returned value is computed as `min(max(x, minVal), maxVal)`

iTime:

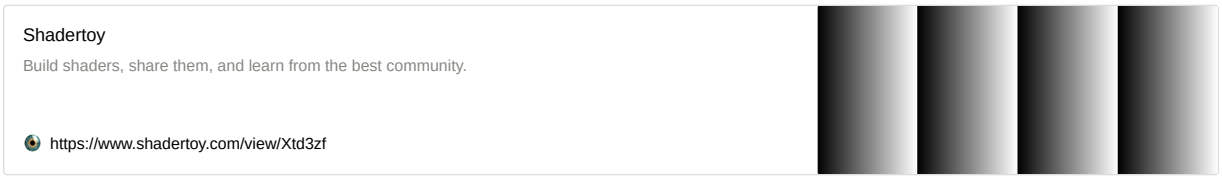
mod(): `mod()` returns the value of `x` modulo `y`. This is computed as `x - y * floor(x/y)`.

smoothstep(): Perform Hermite interpolation between two values

```
float smoothstep(float edge0, float edge1, float x)
//smoothstep() performs smooth Hermite interpolation between 0 and 1 when edge0 < x < edge1. This is useful in cases where a threshold f

    genType t;  /* Or genDType t; */
    t = clamp((x - edge0) / (edge1 - edge0), 0.0, 1.0);
    return t * t * (3.0 - 2.0 * t);
```

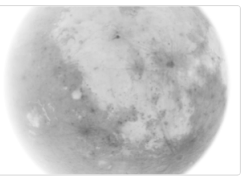
fract(): Compute the fractional part of the argument, `fract()` returns the fractional part of `x`. This is calculated as `x - floor(x)`.



The Book of Shaders

Gentle step-by-step guide through the abstract and complex universe of Fragment Shaders.

https://thebookofshaders.com/glossary/?search=fract

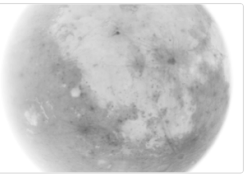


dot(): Calculate the dot product of two vectors. a(x1,y1)*b(x2,y2) =x1*x2+y1*y2; a*b = |a||b| cos(angle);

The Book of Shaders

Calculate the dot product of two vectors float dot(float x, float y) float dot(vec2 x, vec2 y) float dot(vec3 x, vec3 y) float dot(vec4 x, vec4 y) x specifies the first of two vectors y specifies the second of two vectors dot() returns the dot product of two vectors, x and y.

https://thebookofshaders.com/glossary/?search=dot



max():

if d=max(vec3(-1,2,10),vec3(10,-10,40)); then d = vec3(10,2,40);

Prototype

Truchet Effect / Grid

Shader Coding: Truchet Tiling Explained!

https://www.youtube.com/watch?v=2R7h76GoIJM

RayMarching

Ray Marching for Dummies!

https://youtu.be/PGtv-dBi2wE

Basic of Raymarching

Input:

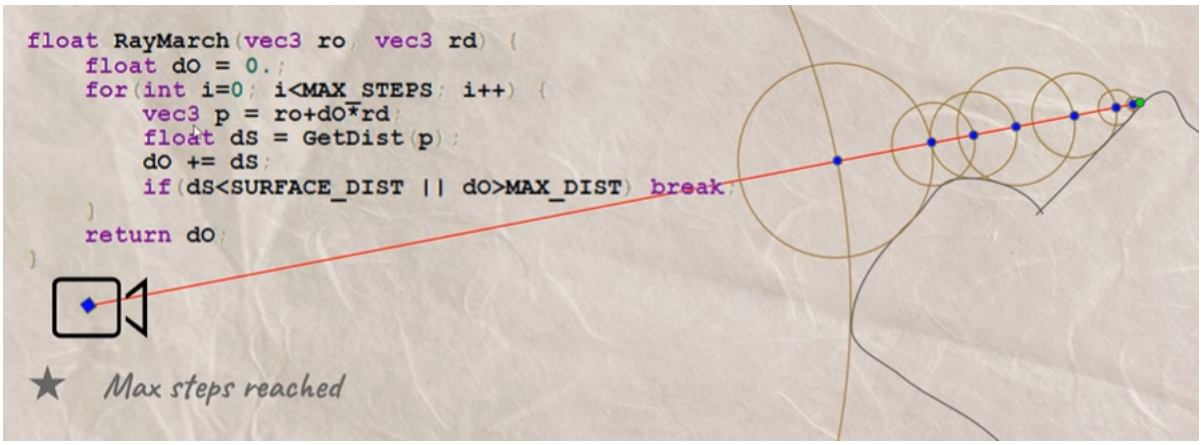
origin point & direction;

Process:

Along the ray, a step is performed, and at each step the distance to the object which is the closest to the current origin is detected, and that distance is used as the value for the step accumulation. This can be understood as drawing the largest circle along the ray that is tangent to the object and using the intersection of that circle with the farthest point of the ray as the next origin. When the radius of the circle is close to zero, it means that the ray cannot advance and has touched the object, and if the number of steps is out of max steps, it means that the ray has no intersection with the object.

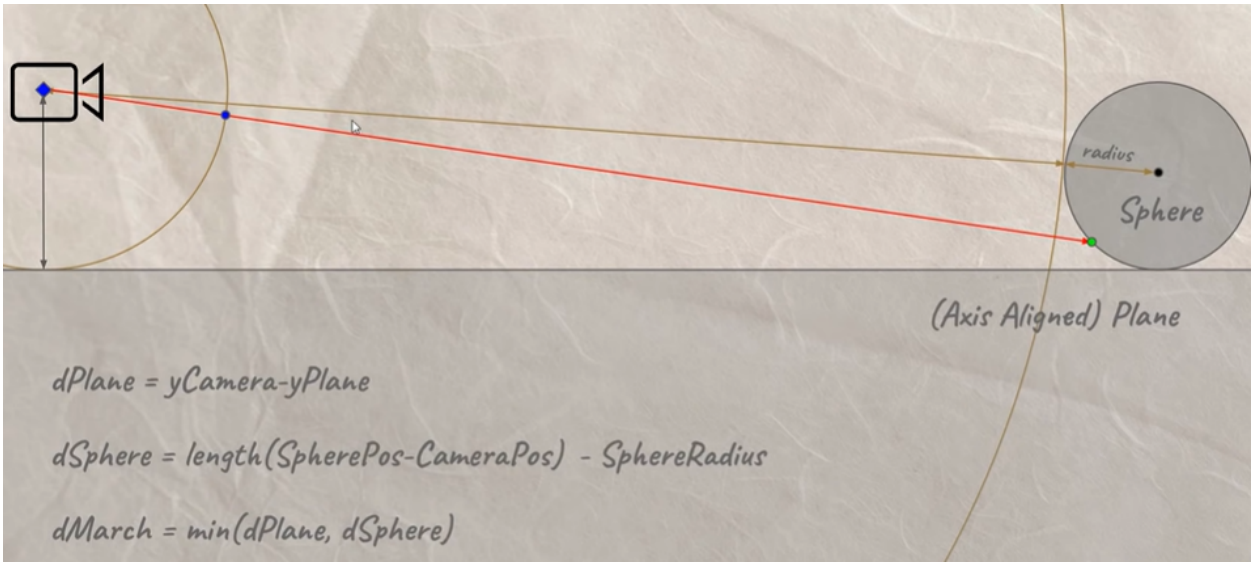
Output:

distance to object




```
#define MAX_STEPS 100
#define MAX_DIST 100.
#define SURF_DIST .01
//raymarching/ sphere tracing
//input: origin point & direction; output: distance to object
float RayMarch(vec3 ro, vec3 rd){
    float  dO=0.;//step?
    for(int i=0;i<MAX_STEPS;i++){
        vec3 p = ro + rd*dO;// location of step
        float dS = GetDist(p);//distance to the closest object
        dO += dS;//location/step++
        if(dO>MAX_DIST || dS<SURF_DIST) break;
    }
    return dO;
}
```

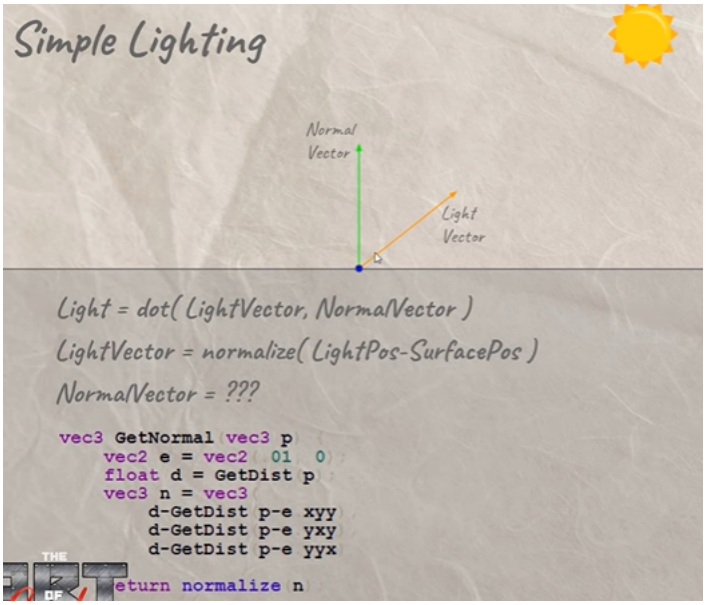
Distance from point to object




```
//distance from camera to object
float GetDist(vec3 p){
    vec4 s = vec4(0,1,6,1);//x,y,z:location; w:radius
    float sphereDist = length(p-s.xyz)-s.w;
    float planeDist = p.y;//distance from cam to plane
    float d = min(sphereDist,planeDist);//the min dist: to plane or to cam
    return d;
}
```

Lighting system

Lighting is about the angle of light ray.




```
//light system
    //normal vector,perpendicular to surface
vec3 GetNormal(vec3 p){
    float d = GetDist(p);
    vec2 e = vec2(.01,0);
    vec3 n = d-vec3(
        GetDist(p-e.xyy),
        GetDist(p-e.yxy),
        GetDist(p-e.yyx)
    );
    return normalize(n);
}
    //light vector
float GetLight(vec3 p){
    vec3 lightPos = vec3(0,5,6);
    lightPos.xz = vec2(sin(iTime),cos(iTime))*2.;
    vec3 l = normalize(lightPos-p);
    vec3 n = GetNormal(p);

    float dif = dot(n,l);
    return dif;
    // p = surfacePos
}
```

light = dot(lightVector,NormalVector);

> dot(): Calculate the dot product of two vectors. a(x1,y1)*b(x2,y2) =x1*x2+y1*y2; a*b = |a||b| cos(angle); for here, lightVector and NormalVector are both normalized(|a|,|b|=1).

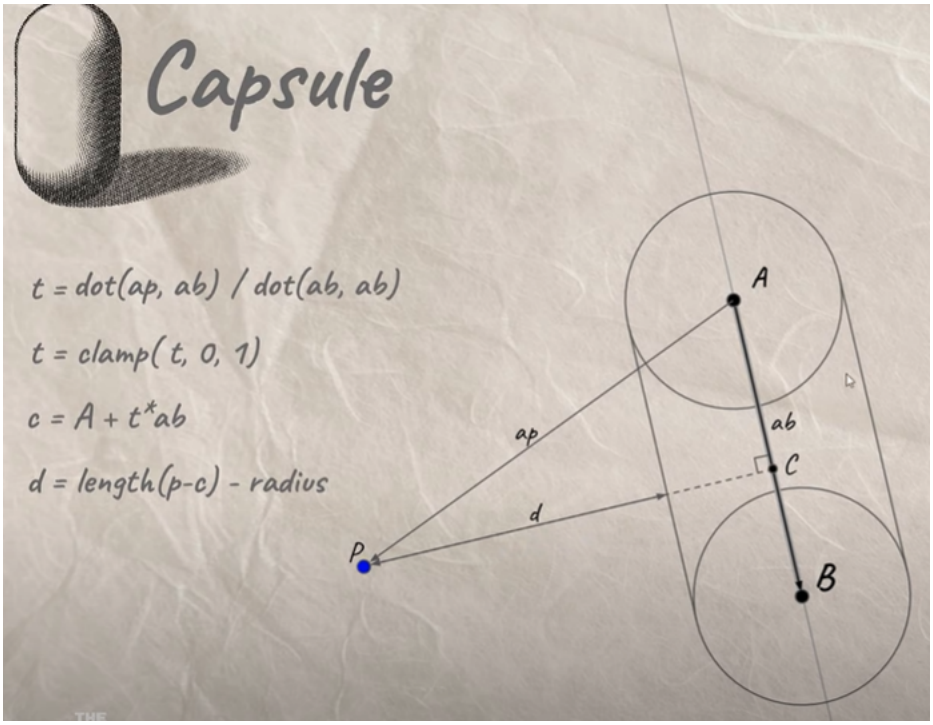
shadow: add in GetLight()

```
float d = RayMarching(p+n*SURF_DIST, l);
if(d < length(lightPos - p))dif *= .1;
```

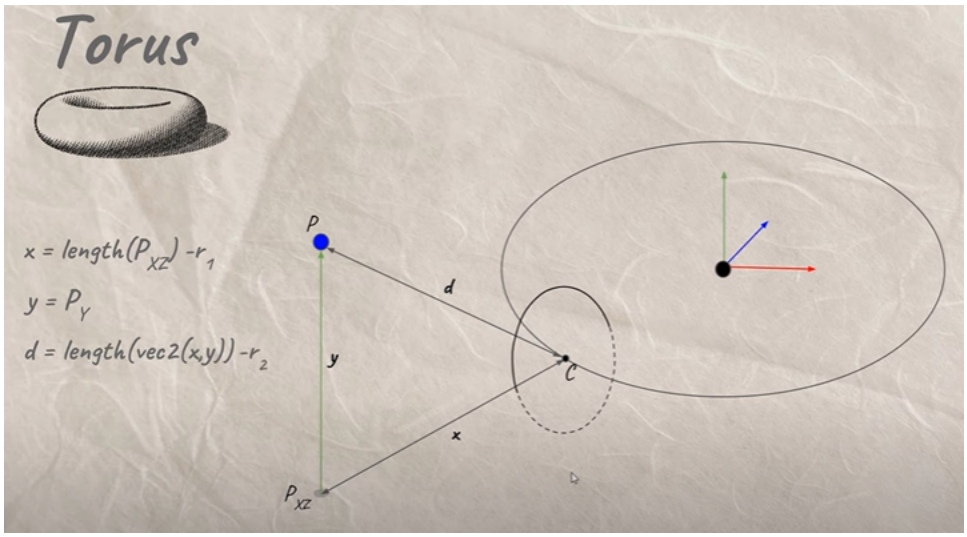
Main coding

```
//set camera and light/ray direction
vec3 ro = vec3(0,1,0);//position of camera
vec3 rd = normalize(vec3(uv.x,uv.y,1));//ray direction /camera
```

Capsule



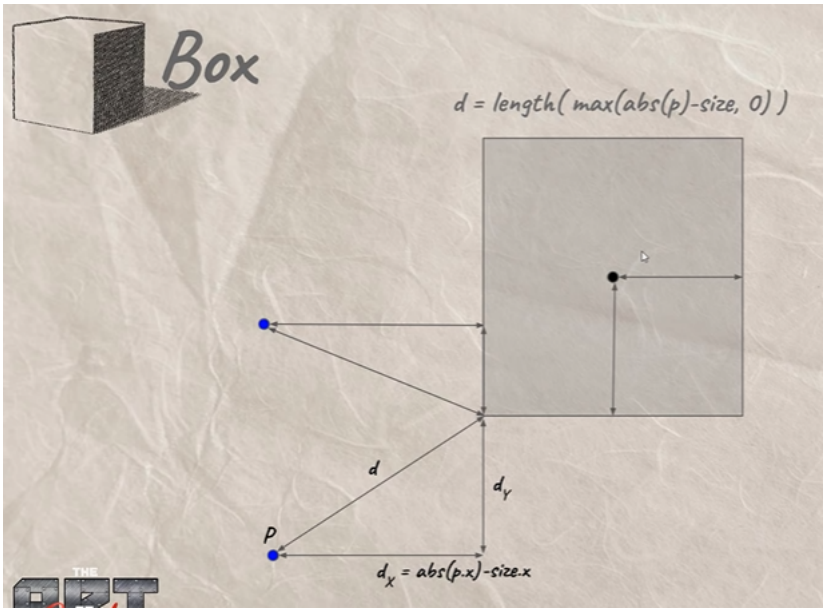

```
float sdCapsule(vec3 p, vec3 a, vec3 b, float r){
    vec3 ab = b-a;
    vec3 ap = p-a;
    float t = dot(ab,ap)/dot(ab,ab);
    t = clamp(t,0.,1.);
    vec3 c = a + t*ab;
    return length(p-c)-r;
}
//distance from camera to object
float GetDist(vec3 p){
    vec4 s = vec4(0,1,6,1);
    float sphereDist = length(p-s.xyz)-s.w;
    float planeDist = p.y;
    float cd = sdCapsule(p,vec3(0,2,6),vec3(1,2,10),.2);
    //float d = min(sphereDist,planeDist);
    float d = min(cd,planeDist);
    return d;
}
```



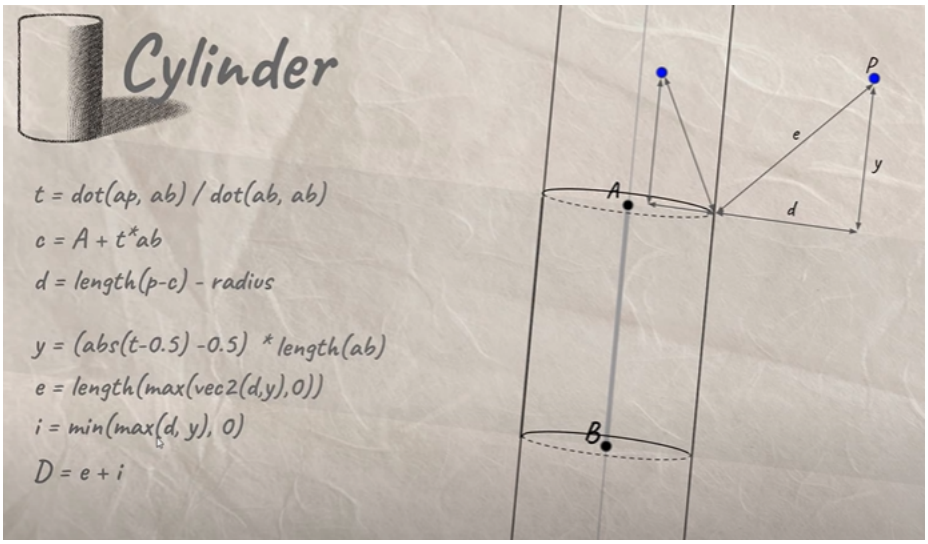

```
//r contains R and r
float sdTorus(vec3 p, vec2 r){
    float x = length(p.xz)-r.x;
    return length(vec2(x,p.y))-r.y;

}

//in GetDist():
float td = sdTorus(p-vec3(2,.3,6),vec2(1.,.3));//p-vec3(),move
```




```
float dBox(vec3 p, vec3 s){
    return length(max(abs(p)-s,0.));
}
```




```
float sdCylinder(vec3 p, vec3 a, vec3 b, float r){
    vec3 ab = b-a;
    vec3 ap = p-a;
    float t = dot(ab,ap)/dot(ab,ab);
    //t = clamp(t,0.,1.);
    vec3 c = a + t*ab;
    float x = length(p-c)-r;
    float y = (abs(t-.5)-.5)*length(ab);
    float e = length(max(vec2(x,y),0.));
    float i = min(max(x,y),0.);
    return e+i;
}
```

References

How To - Shadertoy BETA

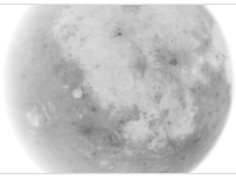
This is a very brief introduction to how the shaders in Shadertoy interface with the rendering platform of your choice (WebGL in the case of the web client, Opengl 2, Opengl 3 or Opengl 4 in case of the native app, OpenGL ES in case of the iOS version).

https://www.shadertoy.com/howto

The Book of Shaders

This is a gentle step-by-step guide through the abstract and complex universe of Fragment Shaders. Patricio Gonzalez Vivo (1982, Buenos Aires, Argentina) is a New York based artist and developer. He explores interstitial spaces between organic and synthetic, analog and digital, individual and collective.

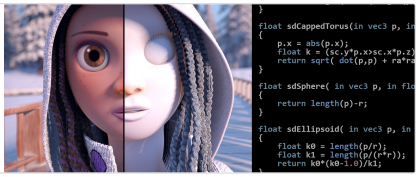
https://thebookofshaders.com/



Inigo Quilez

The 3D SDF functions article is pretty popular, so I decided to write a similar one for 2D primitives, since most of the 3D primitives are grown as extrusions or revolutions of these 2D shapes. So getting these right is improtant.

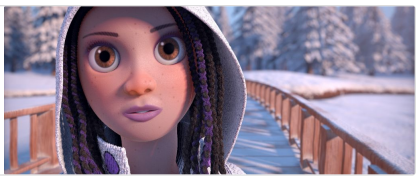
https://iquilezles.org/articles/distfunctions2d/



Shadertoy

Build shaders, share them, and learn from the best community.

https://www.shadertoy.com/view/WsSBzh



p5.js shaders

Interested in shaders? Check out our documentation website for using shaders inside p5.js

https://itp-xstory.github.io/p5js-shaders/#/

p5.js Web Editor

A web editor for p5.js, a JavaScript library with the goal of making coding accessible to artists, designers, educators, and beginners.

https://editor.p5js.org/andoncemore/collections/XLotQtB42

p5jsShaderExamples/4_image-effects at gh-pages · aferriss/p5jsShaderExamples

You can't perform that action at this time. You signed in with another tab or window. You signed out in another tab or window. Reload to refresh your session. Reload to refresh your session.

https://github.com/aferriss/p5jsShaderExamples/tree/gh-pages/4_image-effects

aferriss/
p5jsShaderExamples

A collection of heavily commented WebGL shaders created with p5.js and GLSL

4 Contributors 1 Issue 675 Stars 97 Forks