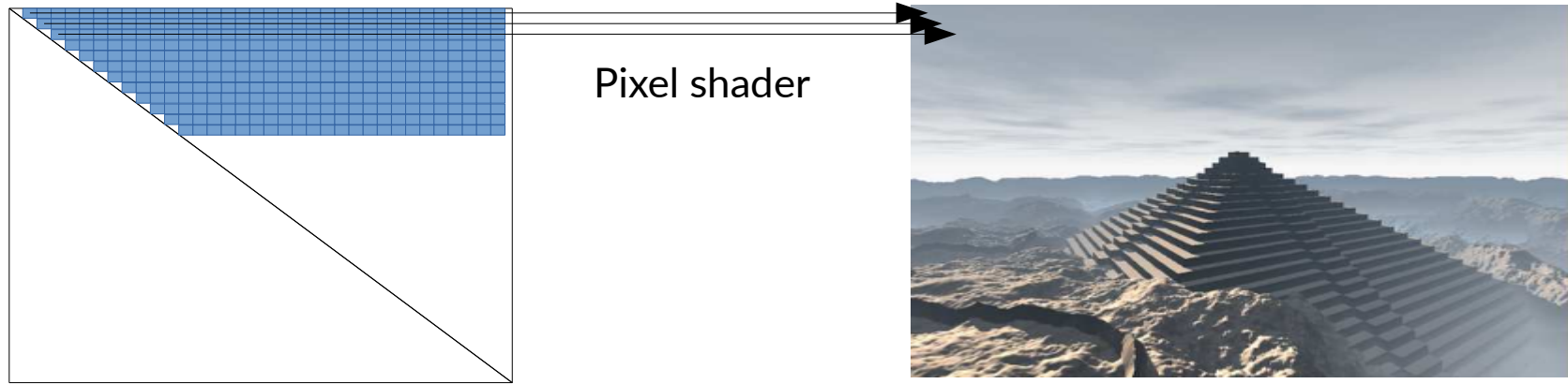


Smooth introduction to shaders

by maq / floppy

Each pixel runs the same shader program



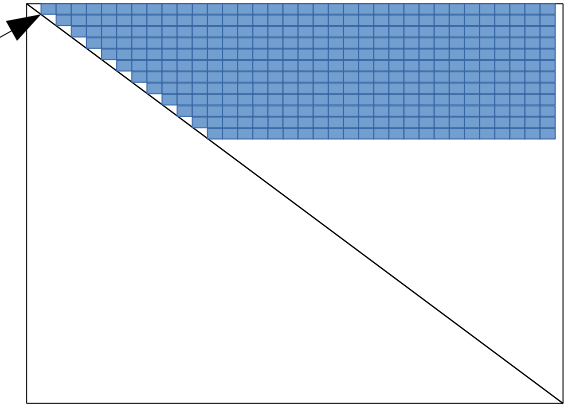
LOOP WAY

```
for( each pixel p )
```

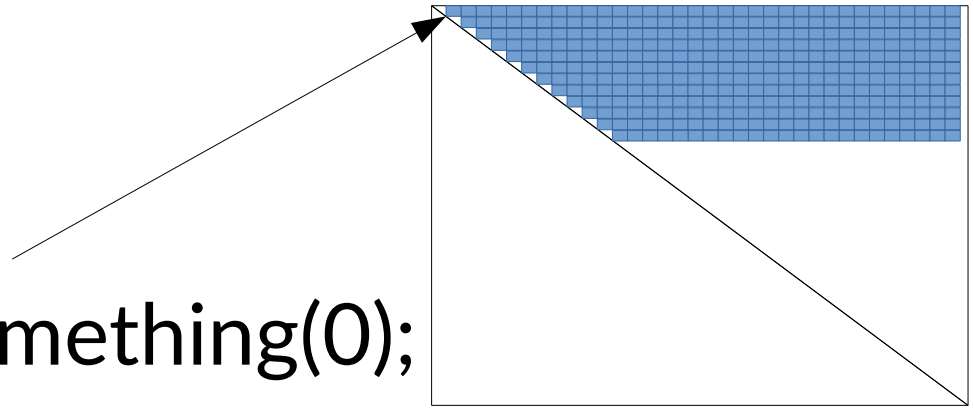
```
{
```

```
    outputColor = doSomething( p );
```

```
}
```



SHADER WAY



```
OutputColor(0) = doSomething(0);
```

```
OutputColor(1) = doSomething(1);
```

```
OutputColor(2) = doSomething(2);
```

```
OutputColor(3) = doSomething(3);
```

```
OutputColor(4) = doSomething(4);
```

Part 1: Shapes

Part 2: Deformations

Part 3: Ray Marching

Distance functions



Toon Cloud by AntoineC

<https://www.shadertoy.com/view/4t23RR>

Plane Deformation



Deform - square

tunnel by IQ

<https://www.shadertoy.com/view/Ms2SWW>

<http://iquilezles.org/www/articles/deform/deform.htm>

Shader Toy

- Authors: Inigo Quilez (IQ), Pol Jeremias (BeautyPi)
- WebGL application
- Database for shaders
- Source of inspiration for many people
- You may learn a lot here
- Not only simple shaders
(multipass games, demos, etc.)

Version 0.4

Shader Toy v0.4 by Inigo Quilez 2009 (tq/rgba), <http://www.iquilezles.org>

* 46 shader, 18 contributors: Adrian Boeing, Auld, Dangquifer/Slexars, Dave Hoskins, Lars Huttar, L.Mallet, Maciej Mityka, Mic, Psycho/Loonies, Paulo Falcao, Simon Green/NVida, TGCC/BUFlame, Tigrou, Viktor Korsun, W.tutani, XT95/FRequency, xT1m/BUFlame and iq/rgba (send me your shaders if you want to contribute - for now I keep the right to decide which shaders to upload based on some sort of "quantity vs variety" criteria)
* Please remember it probably makes sense to credit the authors if you reuse these shaders in your blog, software, demo, or portfolio. Also, the authors definitely want you to let them know if you plan to use their shaders in your applications or products.
* You can directly link to Shader Toy with a shader preset parameter like this: "<http://www.iquilezles.org/apps/shadertoy/?p=Radial%20Blur>" or "<http://www.iquilezles.org/apps/shadertoy/?p=radialblur>"

Inputs: tex0, tex1, tex2, tex3

Shader: Deform
Determin by iq (2009)
A GLSL version of the oldschool 2D deformation effect

Render: 160.5 fps 27.41 640 x 384

```
Source
1 #version 150
2 #define PI 3.14159265358979323846264338327
3 #endif
4
5 uniform float time;
6 uniform vec2 resolution;
7 uniform vec4 mouse;
8 uniform sampler2D tex0;
9 uniform sampler2D tex1;
10
11 void main(void)
12 {
13     vec2 p = -1.0 + 2.0 * gl_FragCoord.xy / resolution.xy;
14     vec2 m = -1.0 + 2.0 * mouse.xy / resolution.xy;
15
16     float a1 = atan(p.y-m.y,p.x-m.x);
17     float r1 = sqrt(dot(p-m,p-m));
18     float a2 = atan(p.y+m.y,p.x+m.x);
19     float r2 = sqrt(dot(p+m,p+m));
20
21     vec2 uv;
22     uv.x = 0.2*time + (r1-r2)*0.25;
23     uv.y = sin(2.0*a1-a2);
24
25     float w = r1*r2*0.8;
26     vec3 col = texture2D(tex0,uv).xyz;
27
28     gl_FragColor = vec4(col/(1+w),1.0);
29 }
```

<http://www.iquilezles.org/apps/shadertoy/index2.html>

http://ShaderToy.com

The screenshot shows the ShaderToy website interface. At the top, there is a search bar and navigation links: "Welcome magflp | Browse New Loved Profile Logout". The main content area is split into two panels. The left panel displays a 2D gradient image transitioning from blue at the bottom to yellow at the top. Below the image is a video player control bar showing "12.71" and "60.0 fps". Underneath the video player are three input fields: "Name of your shader", "Tags, comma separated. For exam", and "Describe your shader". At the bottom of these fields is a "draft" dropdown menu and a "Submit" button. The right panel is a code editor titled "Image" with a "Shader Inputs" dropdown. It contains the following GLSL code:

```
1 void mainImage( out vec4 fragColor, in vec2 fragCoord )
2 {
3     vec2 uv = fragCoord.xy / iResolution.xy;
4     fragColor = vec4(uv, 0.5+0.5*sin(iTime), 1.0);
5 }
```

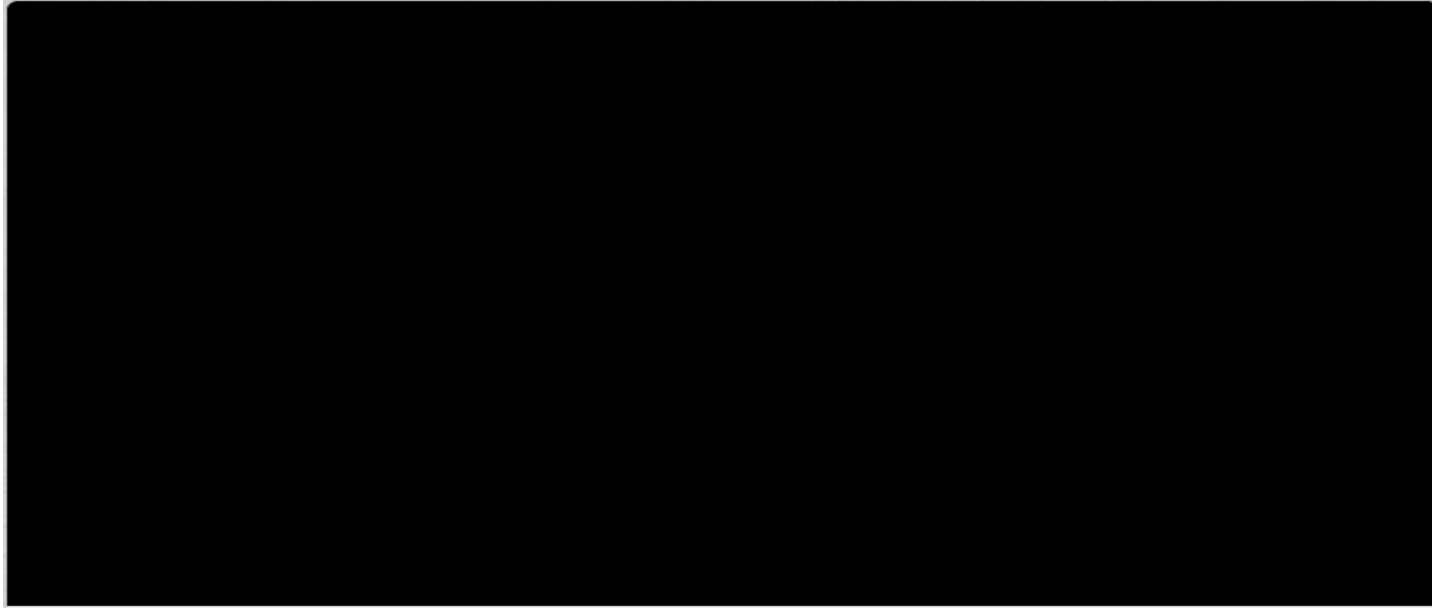
At the bottom of the code editor, it shows "132 chars" and a font size selector set to "XL".

Alternatives?

- PolyCube <https://rice.moonedit.com/>
- GLSL SandBox: <https://glslsandbox.com/>
- More?

Part 1: Shapes

1: Color



```
1 void mainImage( out vec4 fragColor, in vec2 fragCoord )  
2 {  
3     vec3 col = vec3(0.0, 0.0, 0.0);  
4     fragColor = vec4( col, 1.0);  
5 }  
6
```

Red

start



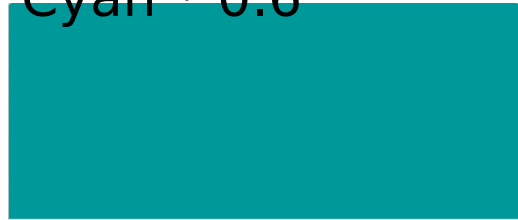
```
vec3 col = vec3(1.0, 0.0, 0.0);
```

Cyan



```
vec3 col = vec3(0, 1.0, 1.0);
```

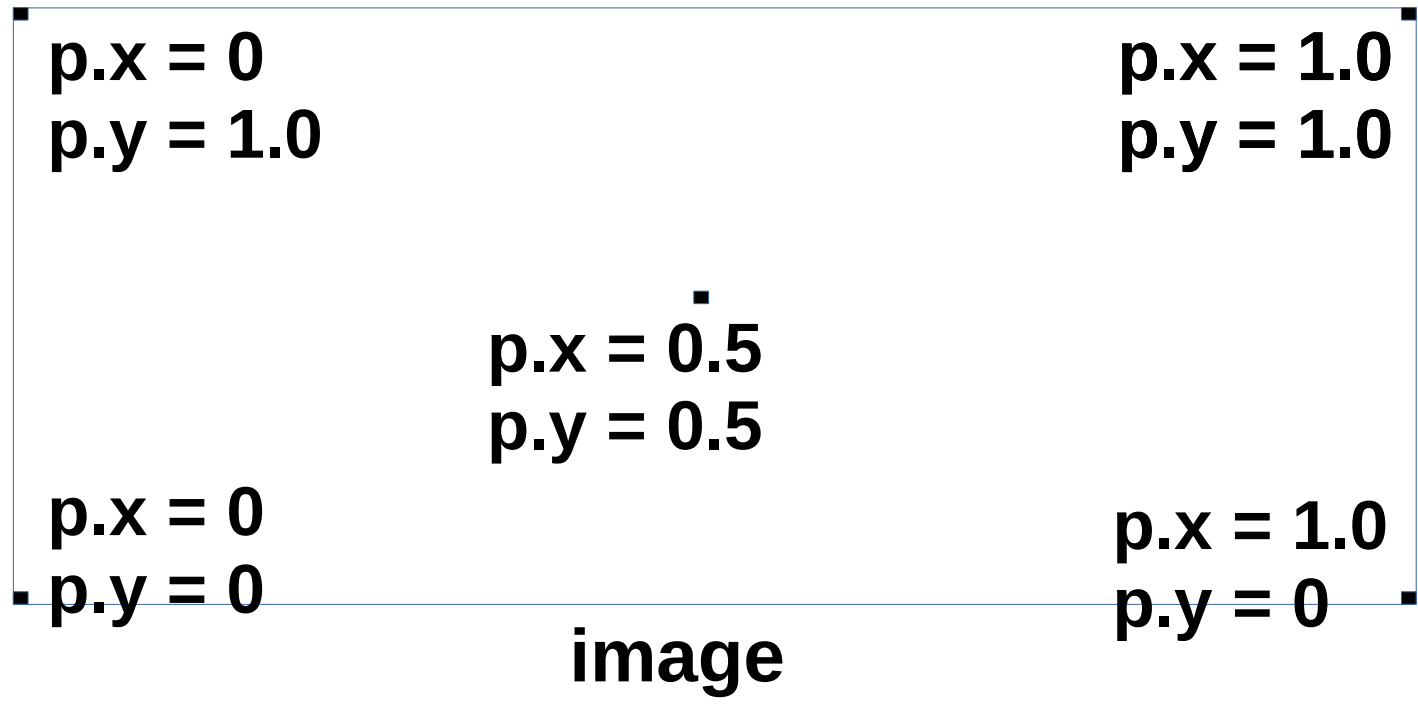
Cyan * 0.6



```
vec3 col = vec3(0, 1.0, 1.0)*0.6;
```

2: Position

vec2 p = fragCoord.xy / iResolution.xy;



2: Position

`p.x=0.0`



`p.x=1.0`

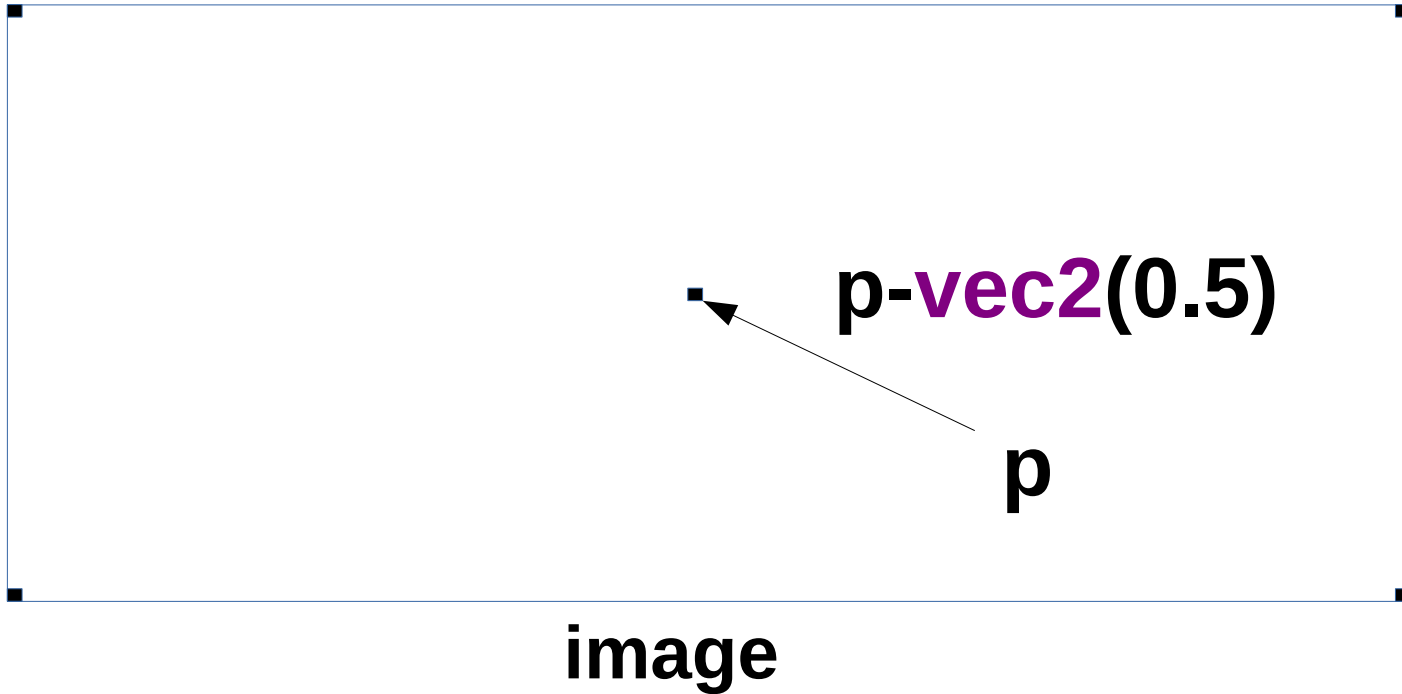
3
4
5

```
vec2 p = fragCoord.xy / iResolution.xy;
```

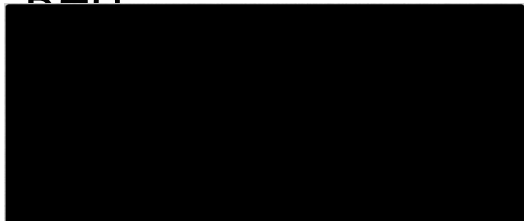
```
vec3 col = vec3(0, 1.0, 1.0) * p.x;
```

3: Distance

```
float dist = 1.0 - length( p-vec2(0.5) );
```

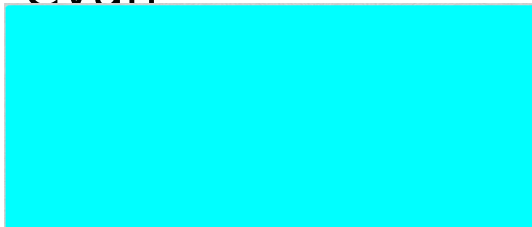


Red



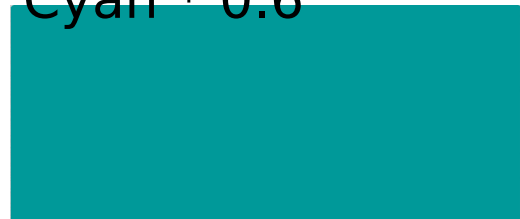
```
vec3 col = vec3(1.0, 0.0, 0.0);
```

Cyan



```
vec3 col = vec3(0, 1.0, 1.0);
```

Cyan * 0.6



```
vec3 col = vec3(0, 1.0, 1.0)*0.6;
```

Gradient

start



```
vec2 p = fragCoord.xy / iResolution.xy;
vec3 col = vec3(1.0, 0.0, 0.0) * p.x;
```

Radial

```
fragColor = vec4( col * 0.7, 1.0);
```

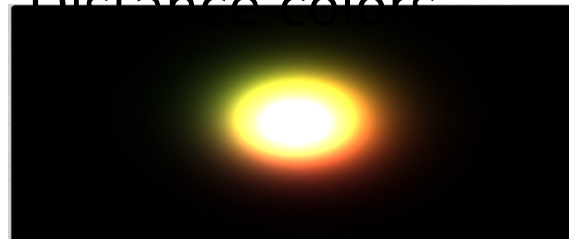


```
vec2 p = fragCoord.xy / iResolution.xy;
```

```
float dist = 1.0-length( p-vec2(0.5) );
```

```
fragColor = vec4(dist, 0, 0, 1.0);
```

Distance colors



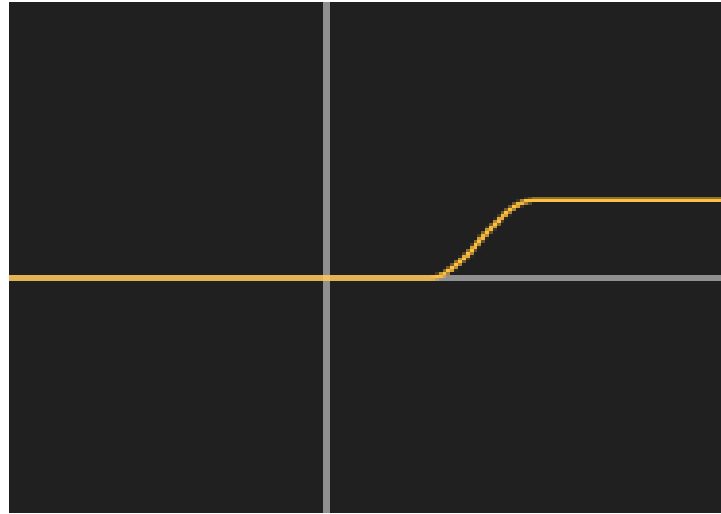
```
void mainImage( out vec4 fragColor, in vec2 fragCoord )
{
  vec2 p = fragCoord.xy / iResolution.xy;
  p.x*=1.5;
  vec3 col = vec3(p.x+0.1*sin(iTime*0.2),
                 p.y+0.05*sin(2.0*iTime), 0.2);
  float dist =
    exp(3.0-15.0*
      length( p-vec2(0.75+0.01*sin(iTime),
                 0.5+0.01*sin(0.2+0.3*iTime) ) ));
  fragColor = vec4(col*dist, 1.0);
}
```

start

Step functions



step(1,x)



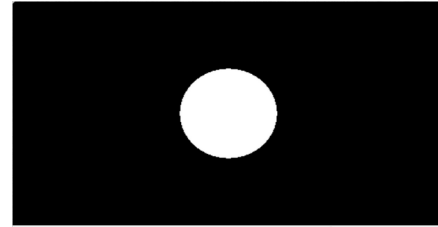
smoothstep(1,2,x)

Circle

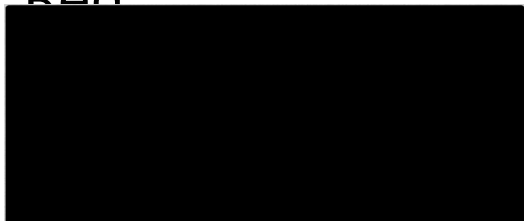
```
float circle(vec2 p, vec2 r, float R)
{
    return step(R, length(p-r));    // transition
}
```

```
vec3 colback = vec3(1.0);
vec3 colfor  = vec3(0.0);

float f = circle( p, vec2(0.0,0.0), 0.4);
vec3 col = mix(colback, colfor, f);
```

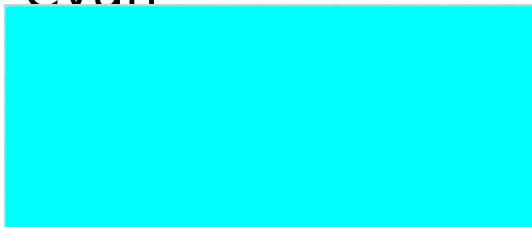


Red



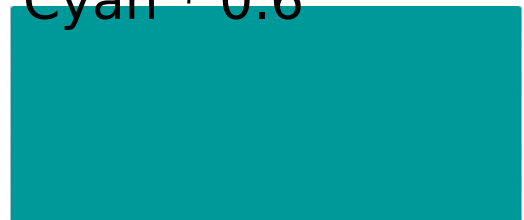
```
vec3 col = vec3(1.0, 0.0, 0.0);
```

Cyan



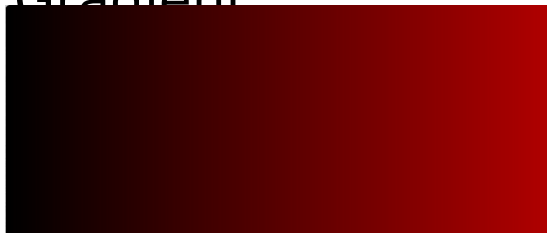
```
vec3 col = vec3(0, 1.0, 1.0);
```

Cyan * 0.6



```
vec3 col = vec3(0, 1.0, 1.0)*0.6;
```

Gradient



```
vec2 p = fragCoord.xy / iResolution.xy;
vec3 col = vec3(1.0, 0.0, 0.0) * p.x;
fragColor = vec4( col * 0.7, 1.0);
```

Radial



```
vec2 p = fragCoord.xy / iResolution.xy;
float dist = 1.0-length( p-vec2(0.5) );
fragColor = vec4(dist, 0, 0, 1.0);
```

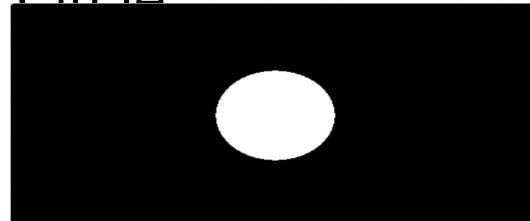
Distance colors



```
void mainImage( out vec4 fragColor, in vec2 fragCoord )
{
    vec2 p = fragCoord.xy;
    p.x*=1.5;
    vec3 col = vec3(p.x*0.1*sin(iTime*0.2),
                    p.y*0.05*sin(2.0*iTime), 0.2);
    float dist =
        exp(1.0-15.0*
            length( p-vec2(0.75+0.01*sin(iTime),
                        0.5+0.01*sin(0.2+0.3*iTime)) ));
    fragColor = vec4(col*dist, 1.0);
}
```

Circle

start



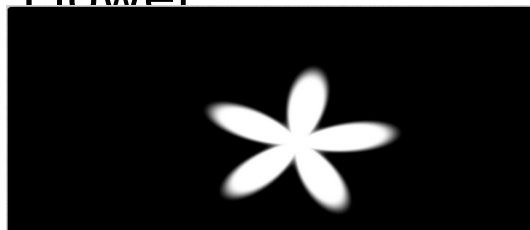
```
float circle(vec2 p, vec2 r, float R)
{
    return step(length(p-r), R);
}
col *= circle( p, vec2(0.0,0.0), 0.4);
```

Smooth Circle

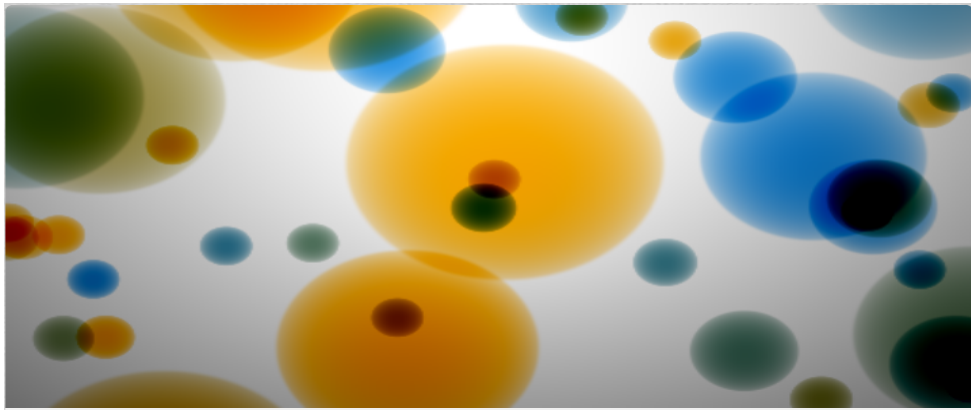


```
float circle(vec2 p, vec2 r, float R)
{
    return 1.0-smoothstep(R-0.05, R+0.05,length(p-r));
}
```

Flower



```
float flower(vec2 p, vec2 r, float R)
{
    float phi = atan(p.y, p.x);
    R*= 0.2+0.6*(1.0+sin(phi*5.0+0.7*iTime));
    return smoothstep(R, R+0.15,length(p-r));
}
```



<https://www.shadertoy.com/view/4dl3zn>



<https://www.shadertoy.com/view/4t23RR>



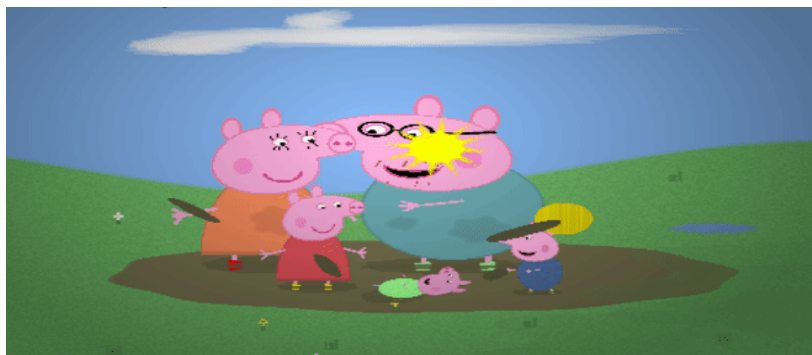
<https://www.shadertoy.com/view/lSxcWn>



<https://www.shadertoy.com/view/Xsf3z8>

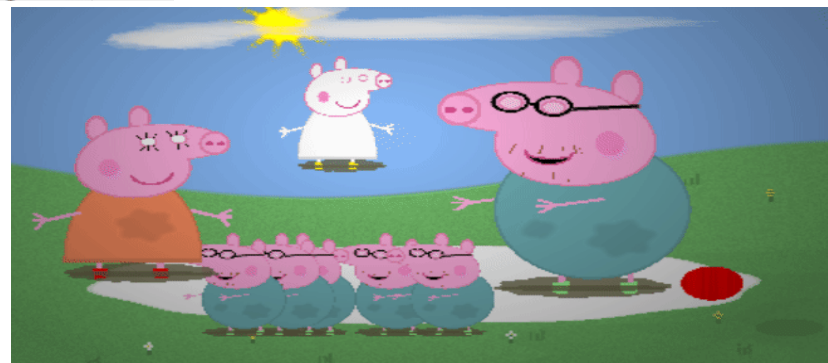


Bartek 2



Robert 7

Karol 8



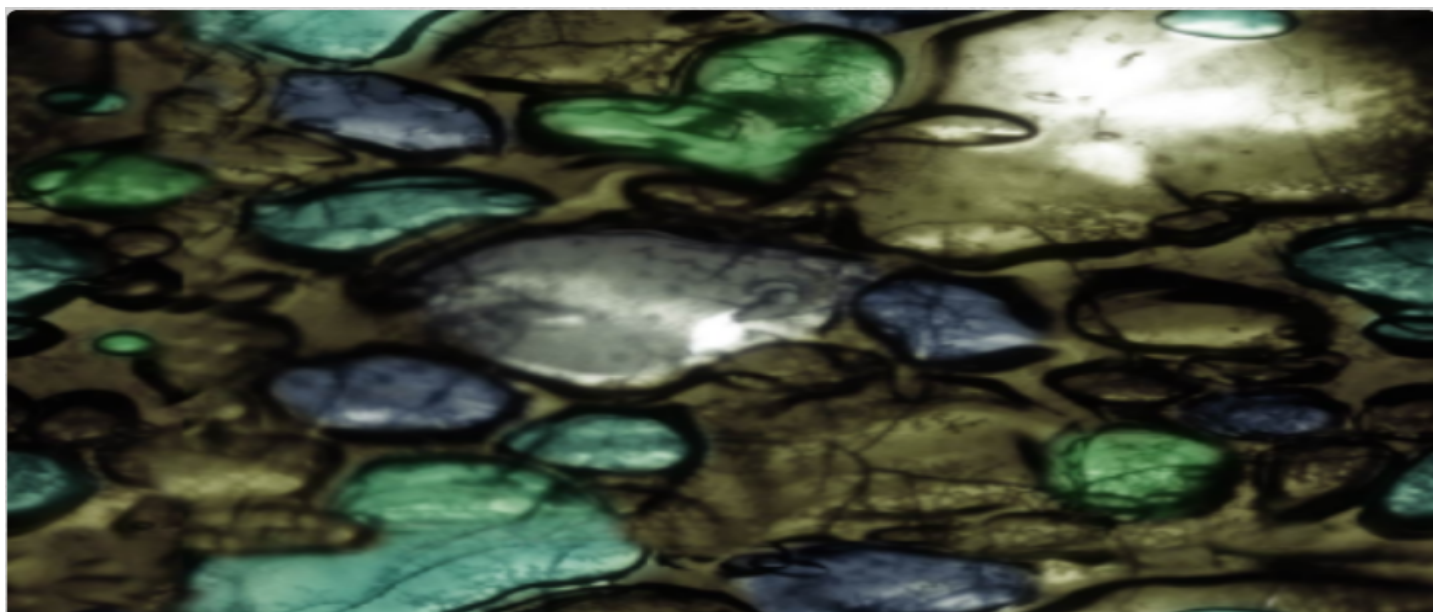
Heart - 2D by IQ



<https://www.shadertoy.com/view/XsfGRn>

```
4 void mainImage( out vec4 fragColor, in vec2 fragCoord )
5 {
6     vec2 p = (2.0*fragCoord-iResolution.xy)/min(iResolution.y,iResolution.x);
7
8     // background color
9     vec3 bcol = vec3(1.0,0.8,0.7-0.07*p.y)*(1.0-0.25*length(p));
10
11     // animate
12     float tt = mod(iTime,1.5)/1.5;
13     float ss = pow(tt,.2)*0.5 + 0.5;
14     ss = 1.0 + ss*0.5*sin(tt*6.2831*3.0 + p.y*0.5)*exp(-tt*4.0);
15     p *= vec2(0.5,1.5) + ss*vec2(0.5,-0.5);
16
17     // shape
18     #if 0
19     p *= 0.8;
20     p.y = -0.1 - p.y*1.2 + abs(p.x)*(1.0-abs(p.x));
21     float r = length(p);
22     float d = 0.5;
23     #else
24     p.y -= 0.25;
25     float a = atan(p.x,p.y)/3.141593;
26     float r = length(p);
27     float h = abs(a);
28     float d = (13.0*h - 22.0*h*h + 10.0*h*h*h)/(6.0-5.0*h);
29     #endif
30
31     // color
32     float s = 0.75 + 0.75*p.x;
33     s *= 1.0-0.4*r;
34     s = 0.3 + 0.7*s;
35     s *= 0.5+0.5*pow( 1.0-clamp(r/d, 0.0, 1.0 ), 0.1 );
36     vec3 hcol = vec3(1.0,0.5*r,0.3)*s;
37
38     vec3 col = mix( bcol, hcol, smoothstep( -0.01, 0.01, d-r ) );
39
40     fragColor = vec4(col,1.0);
41 }
```

Part 2: Deformations

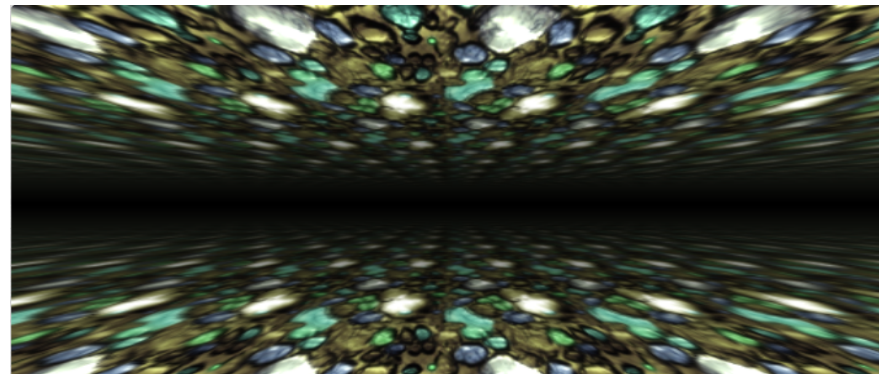
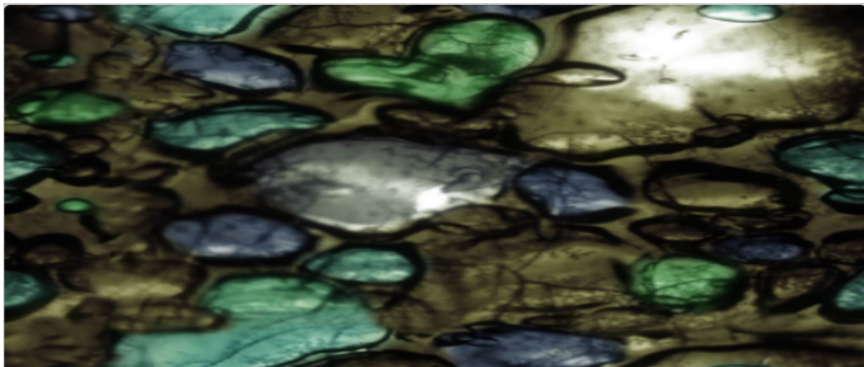


```
1 void mainImage( out vec4 fragColor, in vec2 fragCoord )  
2 {  
3     vec2 uv = fragCoord.xy / iResolution.xy;  
4     vec3 col = texture(iChannel0, uv+iTime*0.1 ).rgb;  
5     fragColor = vec4(col, 1.0);  
6 }  
7  
8
```

Modify **texture position uv** according to:

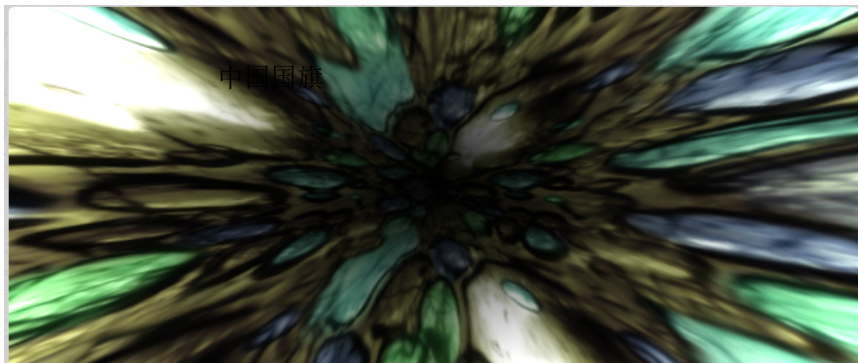
- pixel position
 - distance from some point
 - angle..
- etc.

Deformations

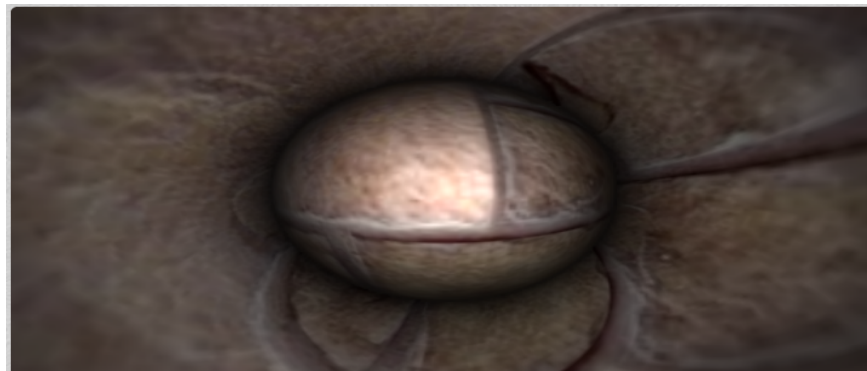


7 `vec2 uv = p;`

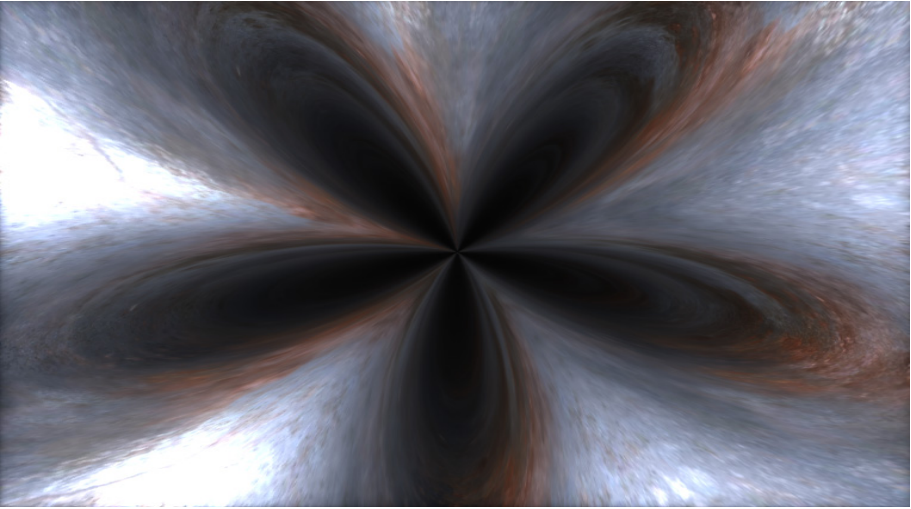
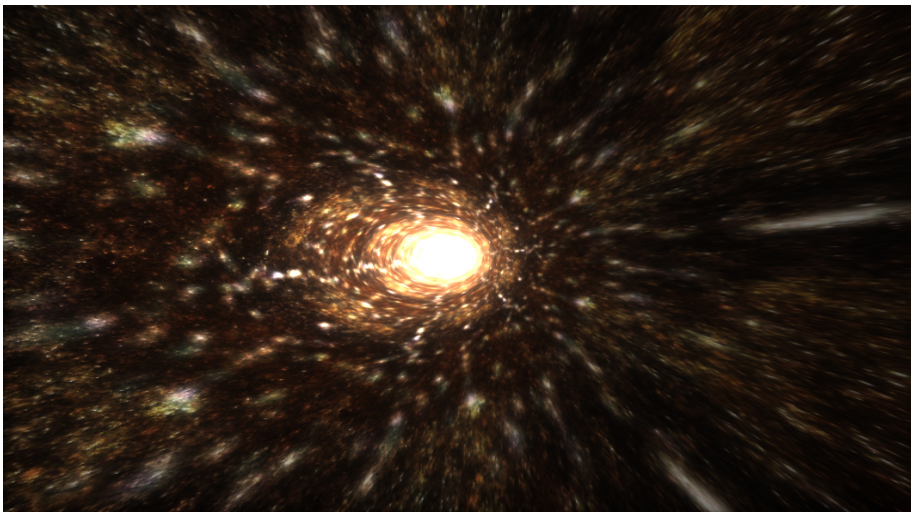
```
float d = abs(p.y+0.8);  
uv=vec2(p)/(d);
```



```
vec2 uv;  
float d = length(p)+0.5;  
uv=vec2(p)/d;
```



<https://www.shadertoy.com/view/4dl3zS>
`uv=vec2(p)/d+iMouse.xy/iResolution.xy;`



<http://www.iquilezles.org/www/articles/deform/deform.htm>

<https://www.shadertoy.com/view/4sXGzn>

<https://www.shadertoy.com/view/4sXGRn>

Part 3: Ray Marching

Example: Sult / Loonies



3rd place at
Breakpoint 2009

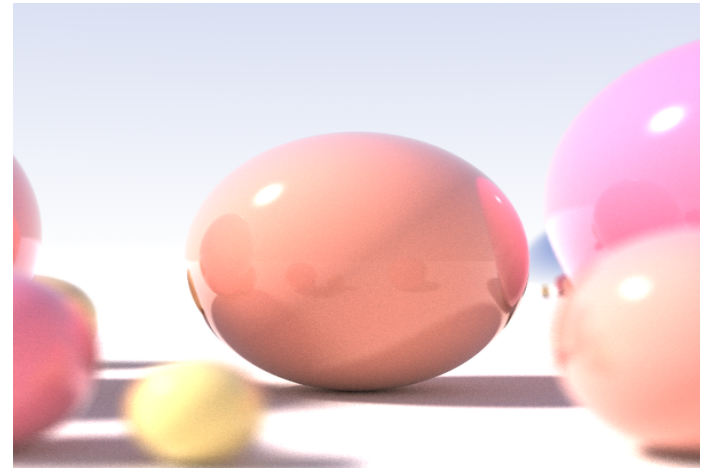
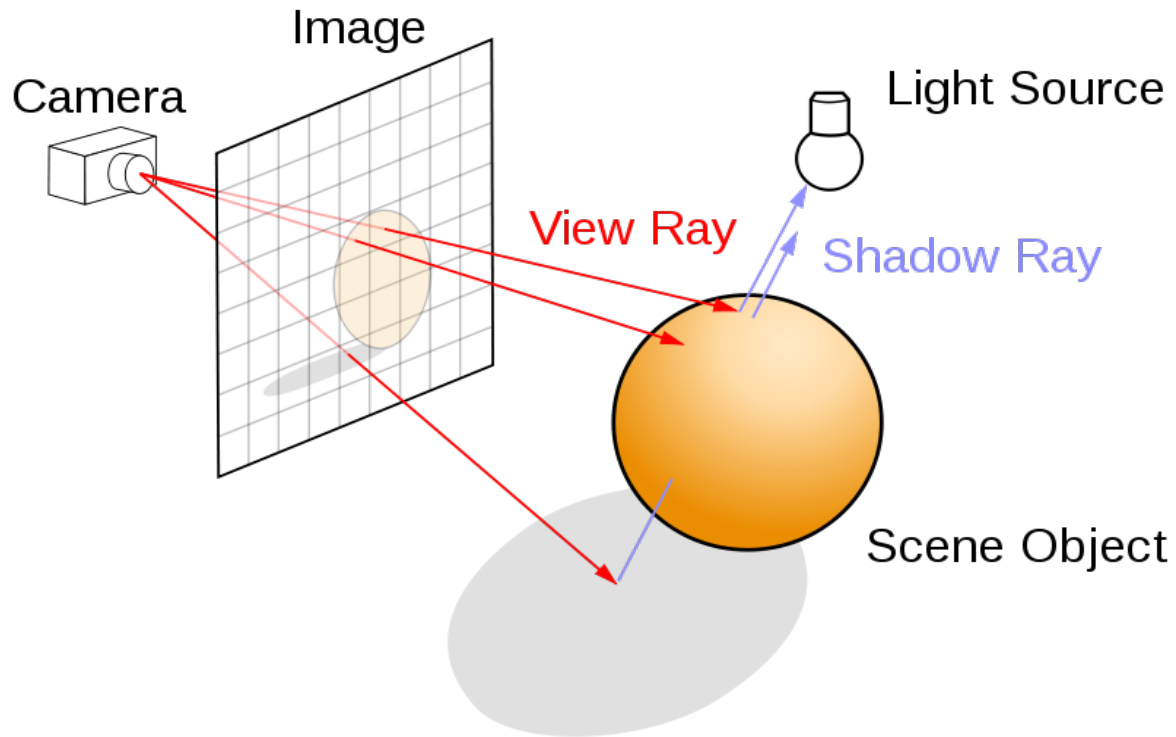
<http://www.youtube.com/watch?v=foR64ZOBAAXA>

Sult 4kb intro tech

Sult .avi > 80 Mb file,

- intro as a fragment shader
- **ray marching**
- FM synth
- .exe packed with Crinkler

Ray Tracing

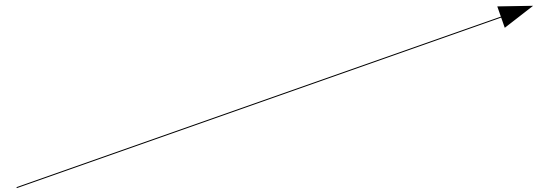


Ray Tracing

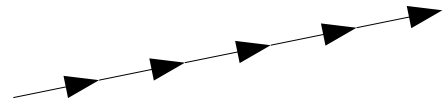
- Problems:
- soft shadows
 - many objects, fractals
 - ray/scene intersections cost a lot

Ray Marching

Ray Tracing Ray-scene intersections



Ray Marching **Marching in the scene** along the ray



Iterations

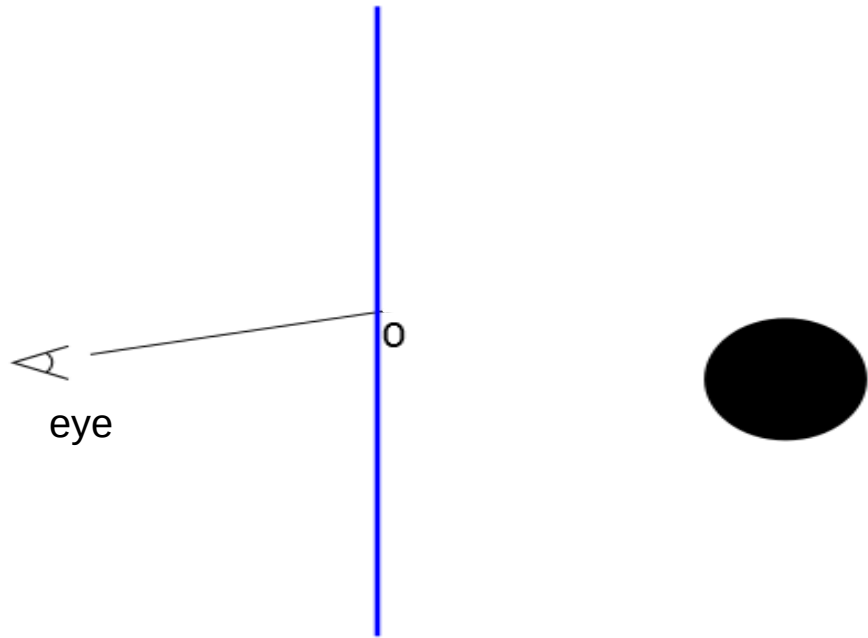
Ray:

$$\mathbf{r}(t) = \mathbf{r}_0 + t * \mathbf{dir}$$

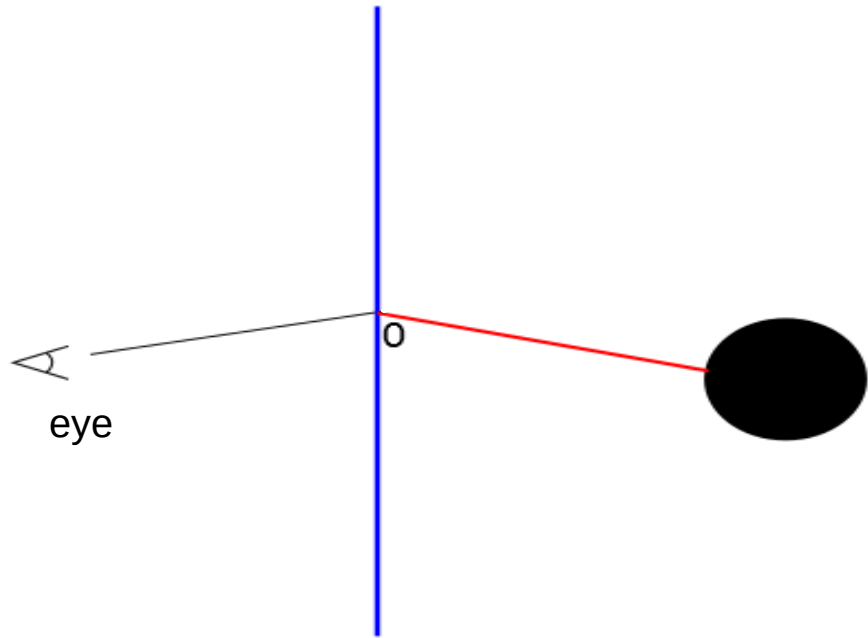
Seek for t_i minimum distance to any object in the scene:

$$t_i = t_{i-1} + f(\mathbf{r}(t_{i-1}))$$

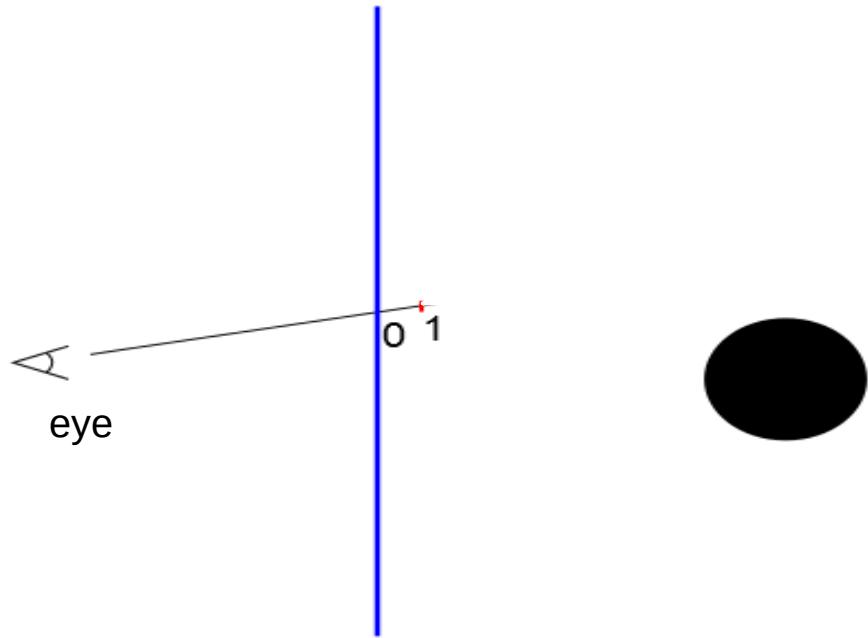
Ray Marching



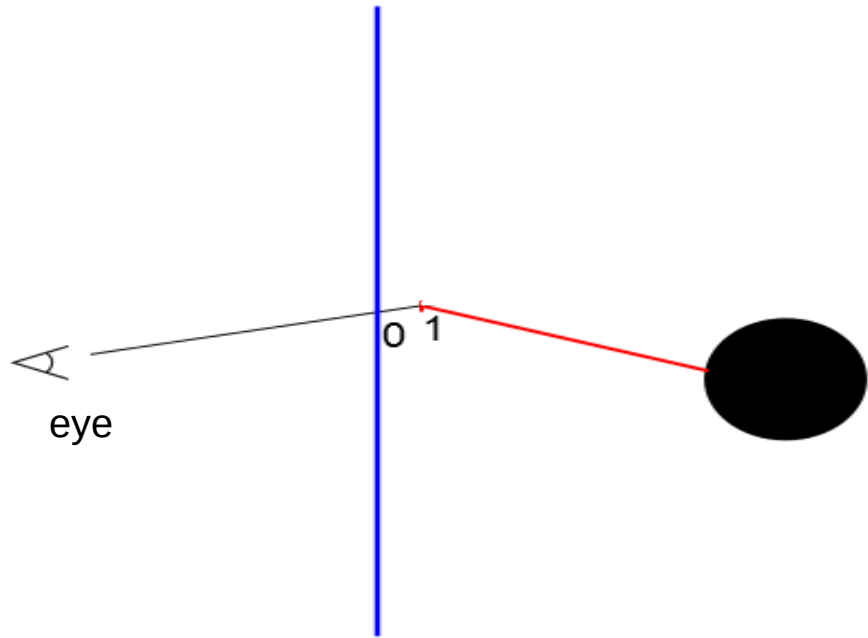
Ray Marching



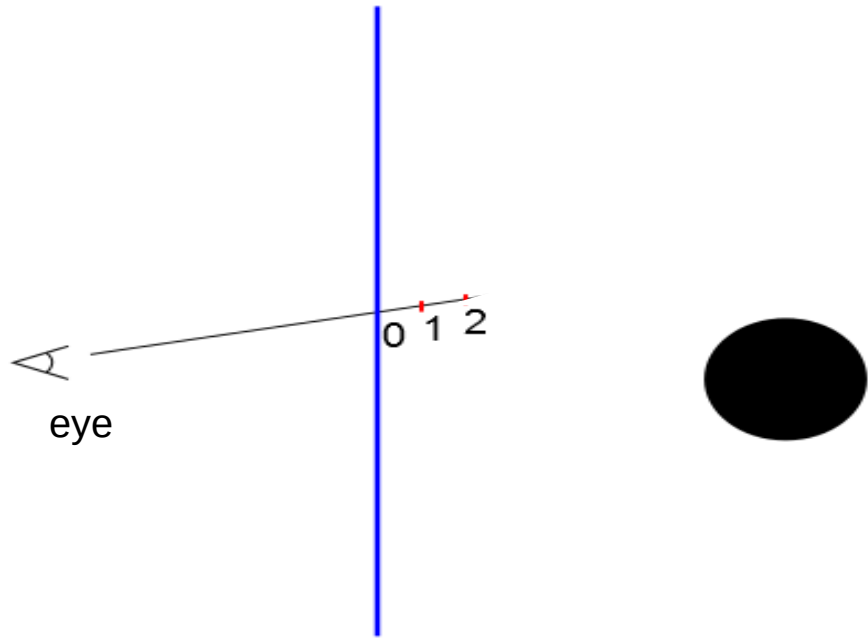
Ray Marching



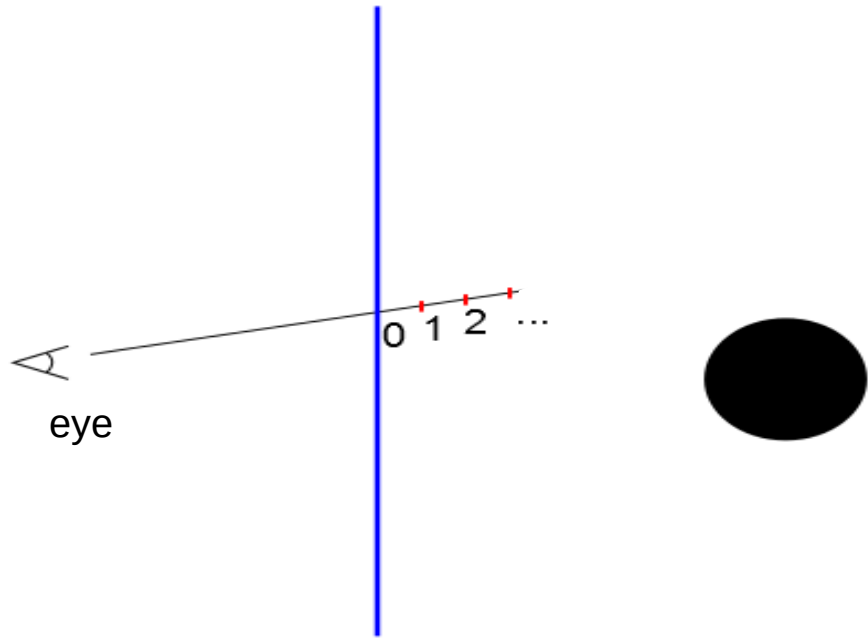
Ray Marching



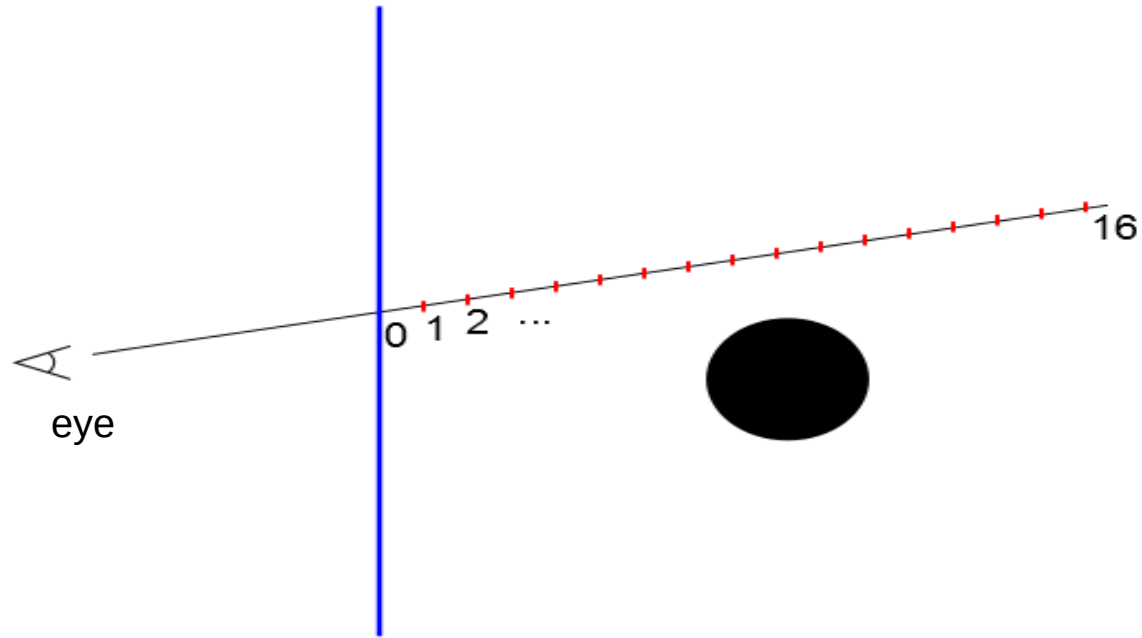
Ray Marching



Ray Marching



Ray Marching



Distance Fields

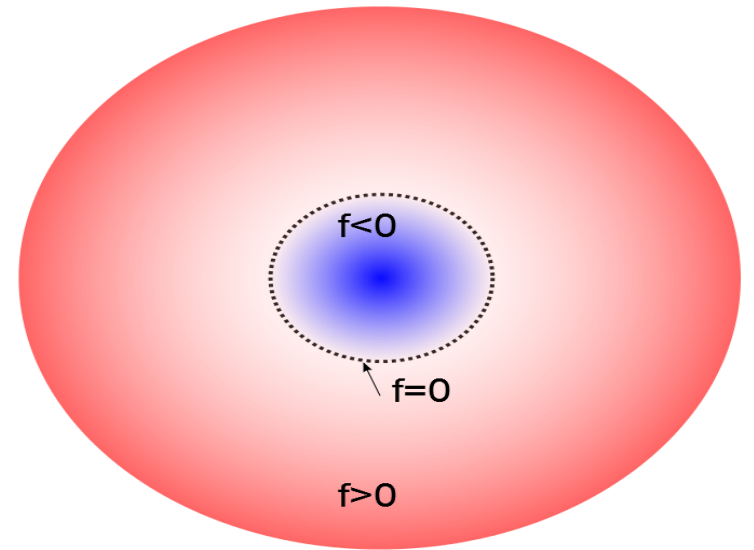
- Object as an *implicit function*
- For each position \mathbf{x} in the scene

$f(\mathbf{x}) < 0$ - point inside

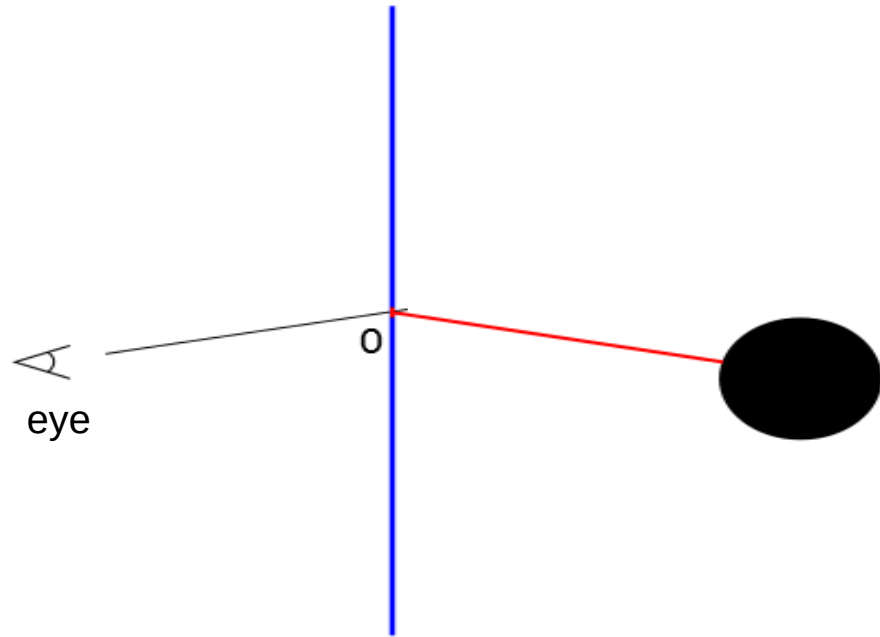
$f(\mathbf{x}) = 0$ - point on the surface

$f(\mathbf{x}) > 0$ - point outside

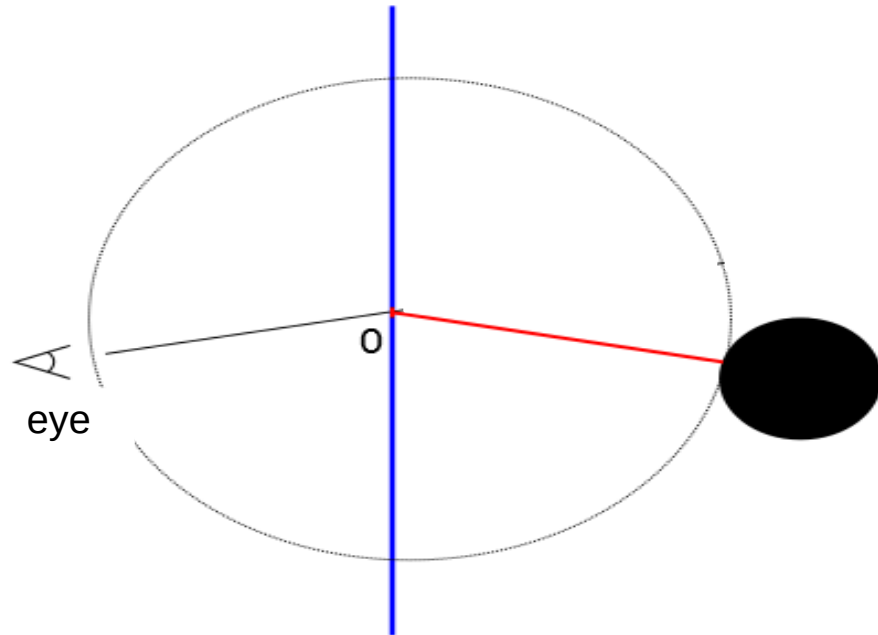
- f proportional to the distance



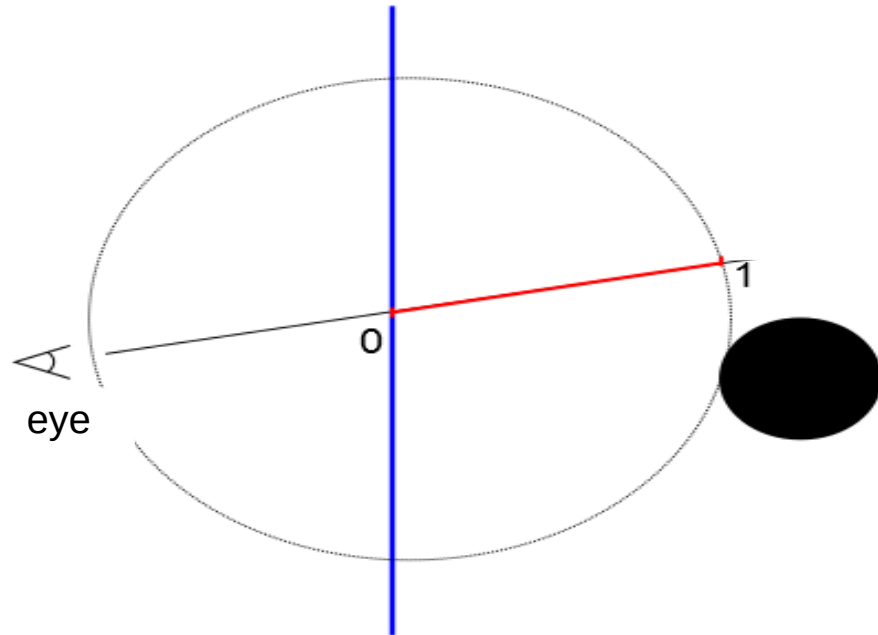
Sphere Tracing



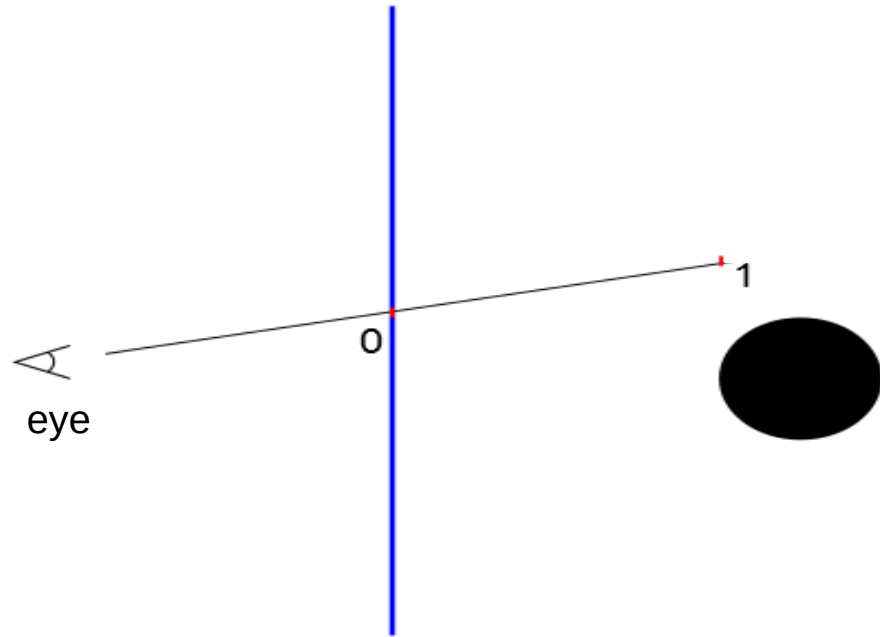
Sphere Tracing



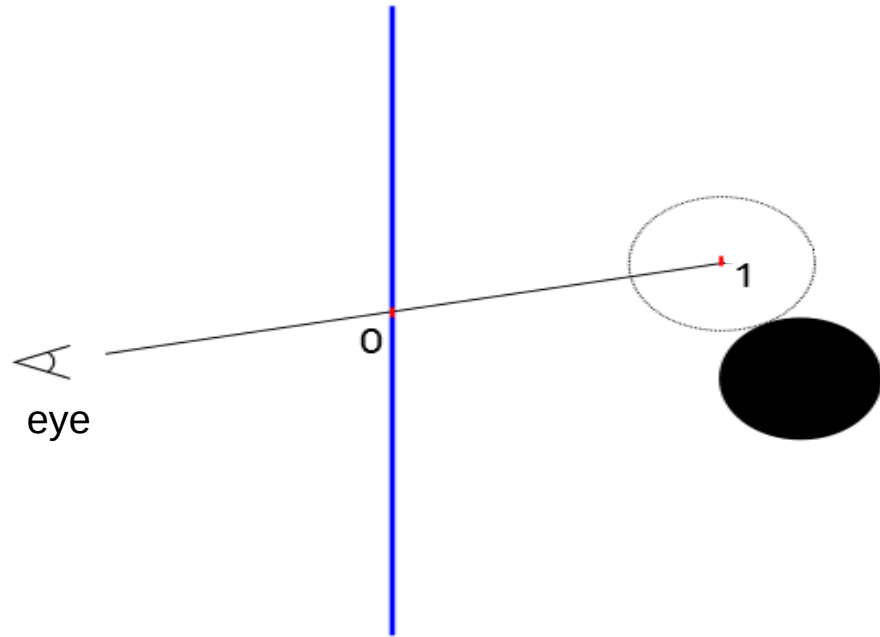
Sphere Tracing



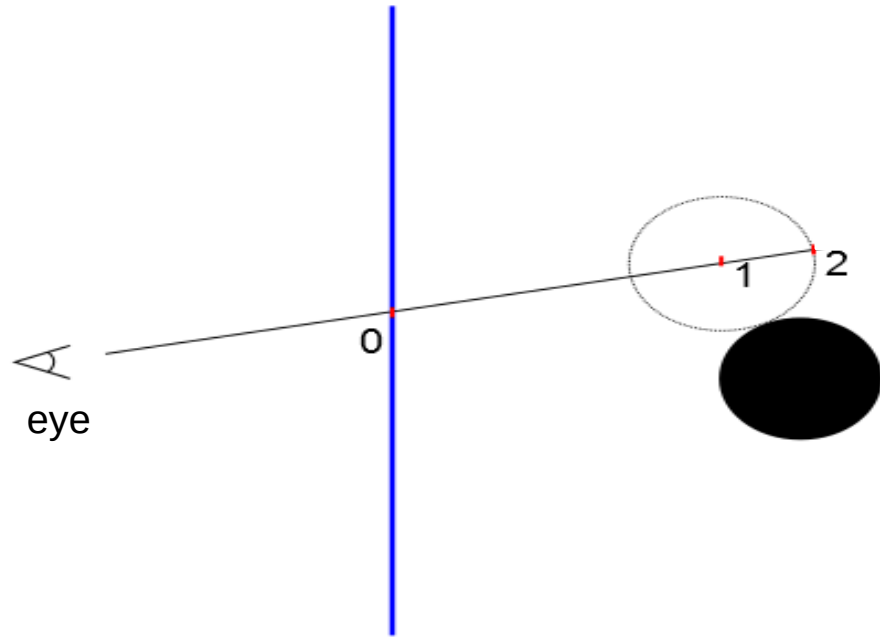
Sphere Tracing



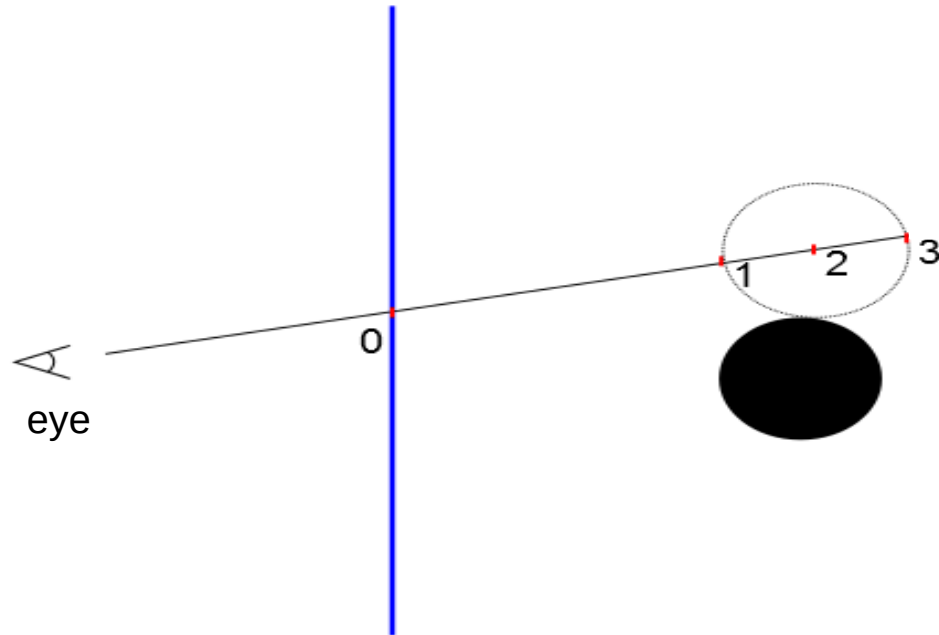
Sphere Tracing



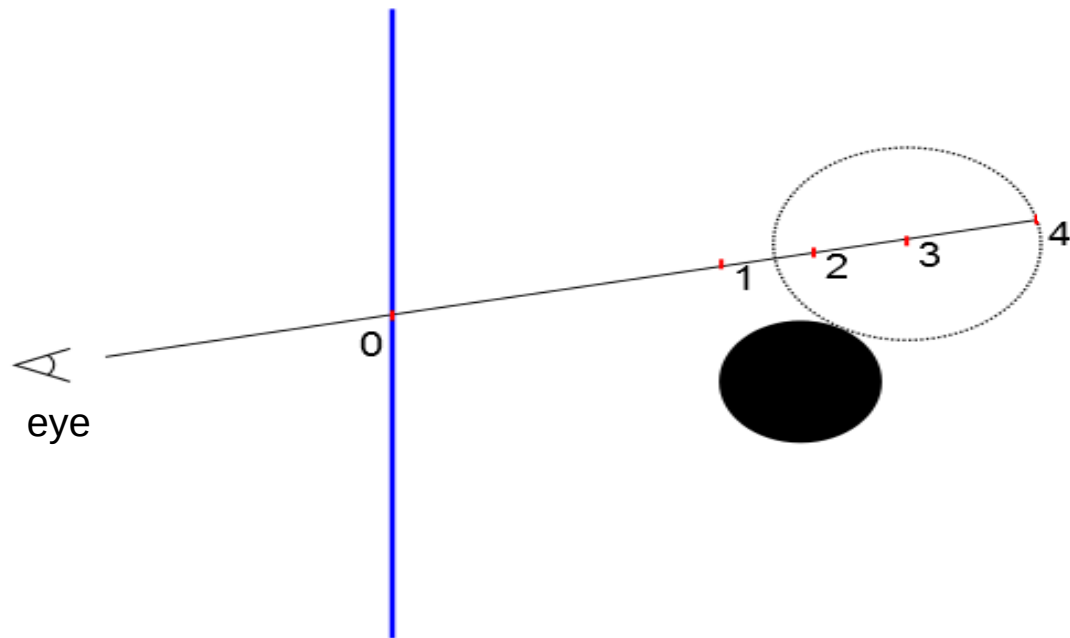
Sphere Tracing



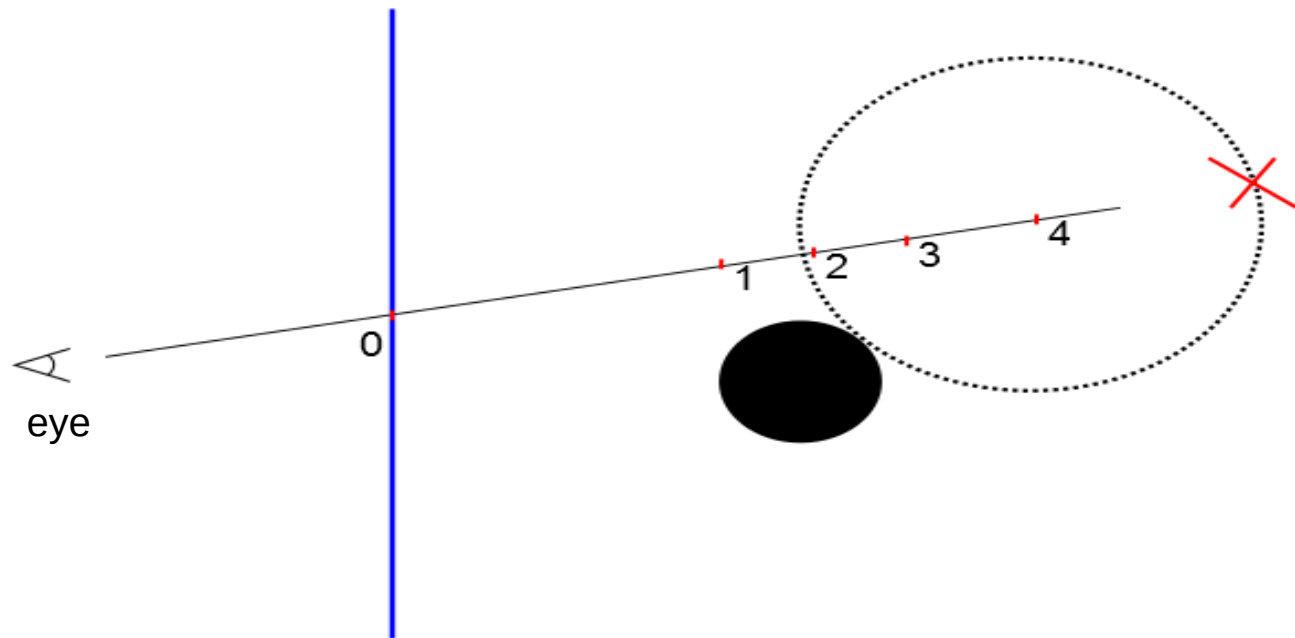
Sphere Tracing



Sphere Tracing



Sphere Tracing



Sphere Tracing

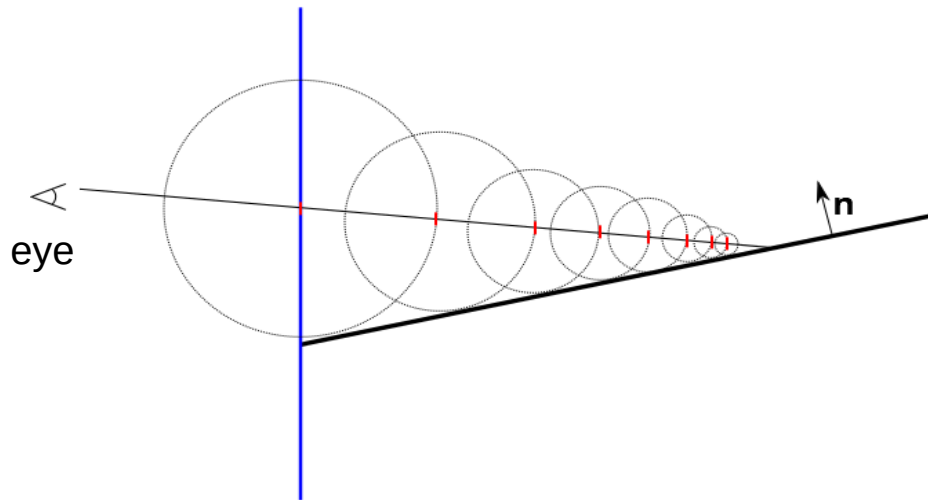


Sphere tracing algorithm

```
float d;  
vec3 p = p0;  
for(int i=0; i<ITER; i++) // 100  
{  
    d = f(p);  
    if(d < EPS) // 0.002  
        break;  
    p = p + dir * d;  
}
```

Distance Functions

Surface (normal \mathbf{n}) going through point \mathbf{nd}



$$f(\mathbf{r}) = \mathbf{r} \cdot \mathbf{n} - d$$

J.C. Hart, *Sphere Tracing: Simple Robust Antialiased Rendering of Distance-Based Implicit Surfaces*
(Course Notes, SIGGRAPH 1993)

Distance Functions

Surface (normal \mathbf{n}) going through point \mathbf{nd}

$$f(\mathbf{r}) = \mathbf{r} \cdot \mathbf{n} - d$$



J.C. Hart, *Sphere Tracing: Simple Robust Antialiased Rendering of Distance-Based Implicit Surfaces*
(Course Notes, SIGGRAPH 1993)

Distance Functions

Ball with radius R

$$f(\mathbf{r}) = |\mathbf{r}| - R$$

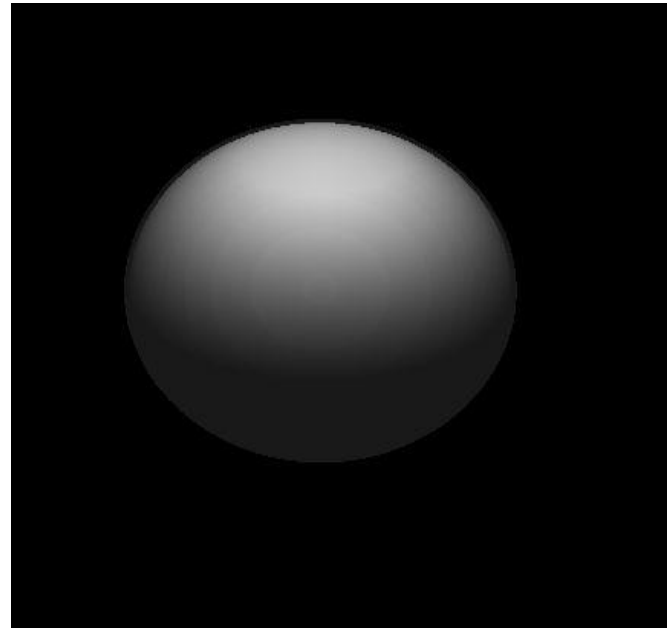
```
// code  
float sphere(vec3 r, float R)  
{  
    return length(r) - R;  
}
```

J.C. Hart, *Sphere Tracing: Simple Robust Antialiased Rendering of Distance-Based Implicit Surfaces*
(Course Notes, SIGGRAPH 1993)

Distance Functions

Ball with radius R

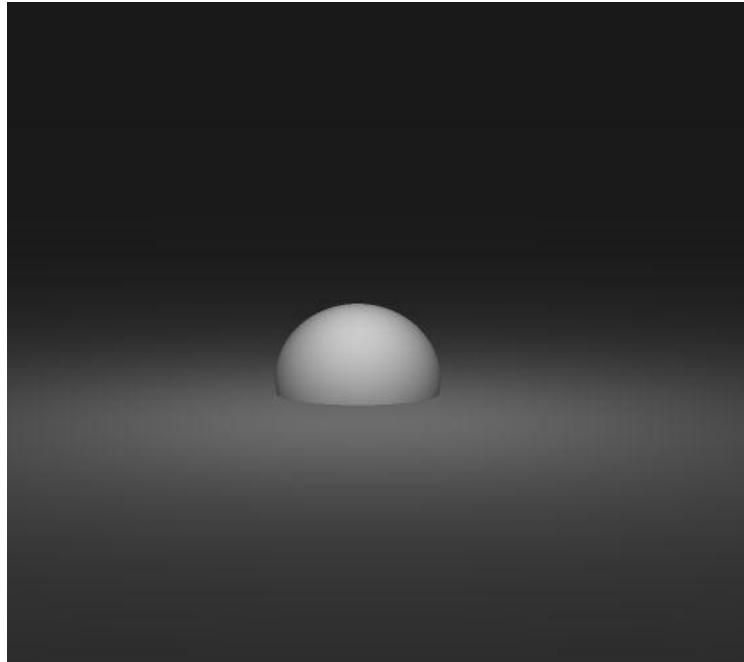
$$f(\mathbf{r}) = |\mathbf{r}| - R$$



J.C. Hart, *Sphere Tracing: Simple Robust Antialiased Rendering of Distance-Based Implicit Surfaces*
(Course Notes, SIGGRAPH 1993)

Minimum of two DFs

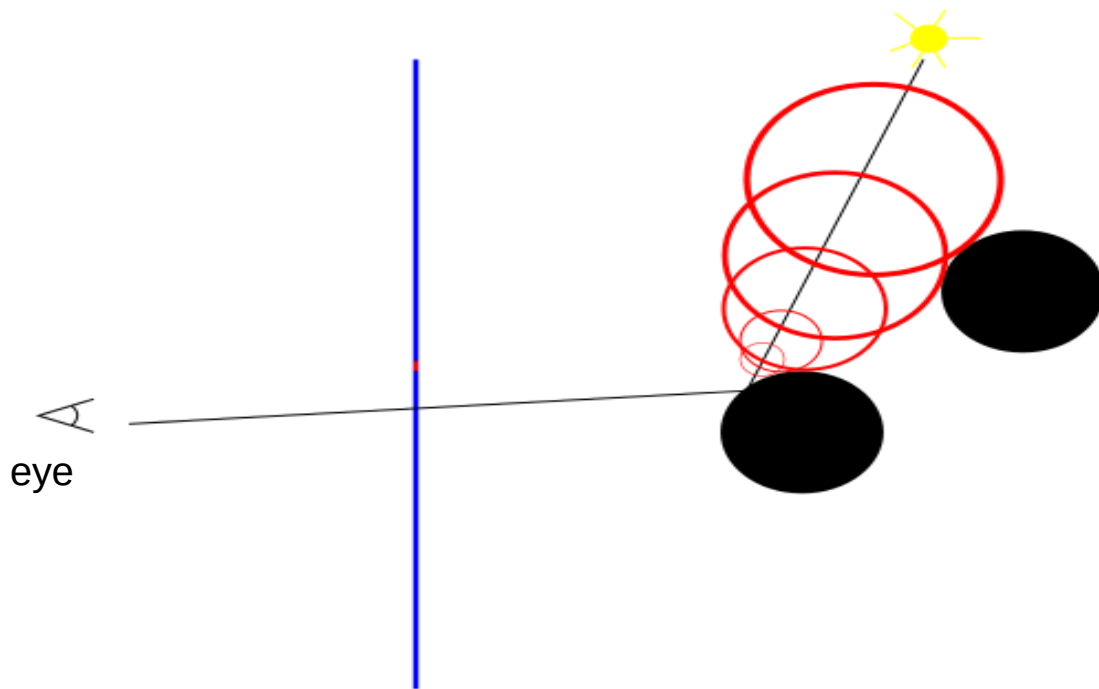
is their sum...



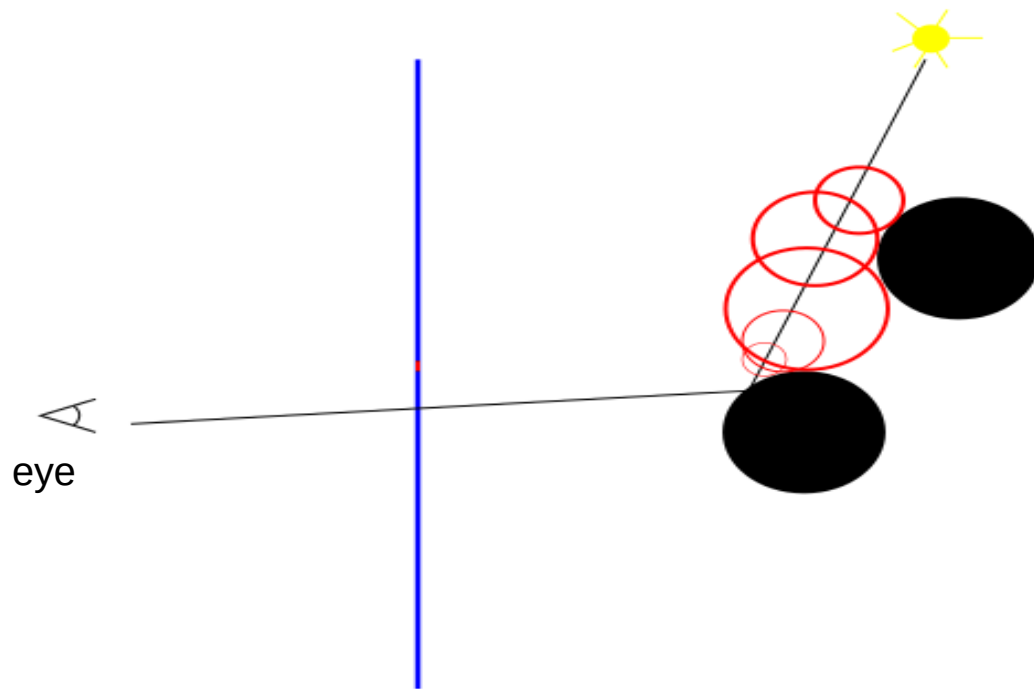
Soft Shadows

Marching from contact point to light source
+
sum distances from the scene
(*easy* implementation)

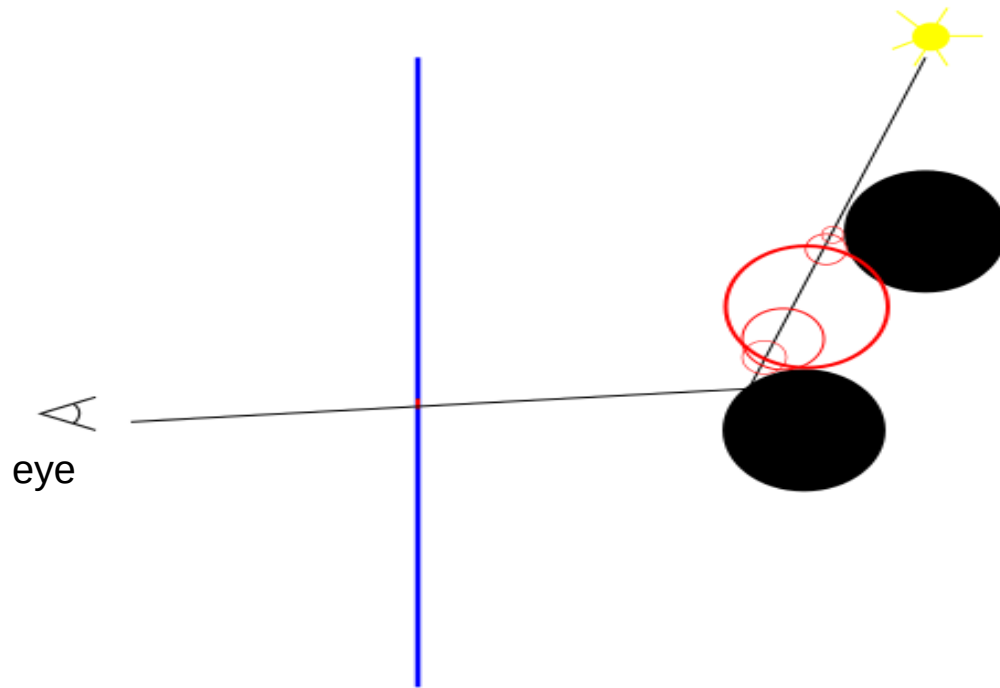
Soft Shadows



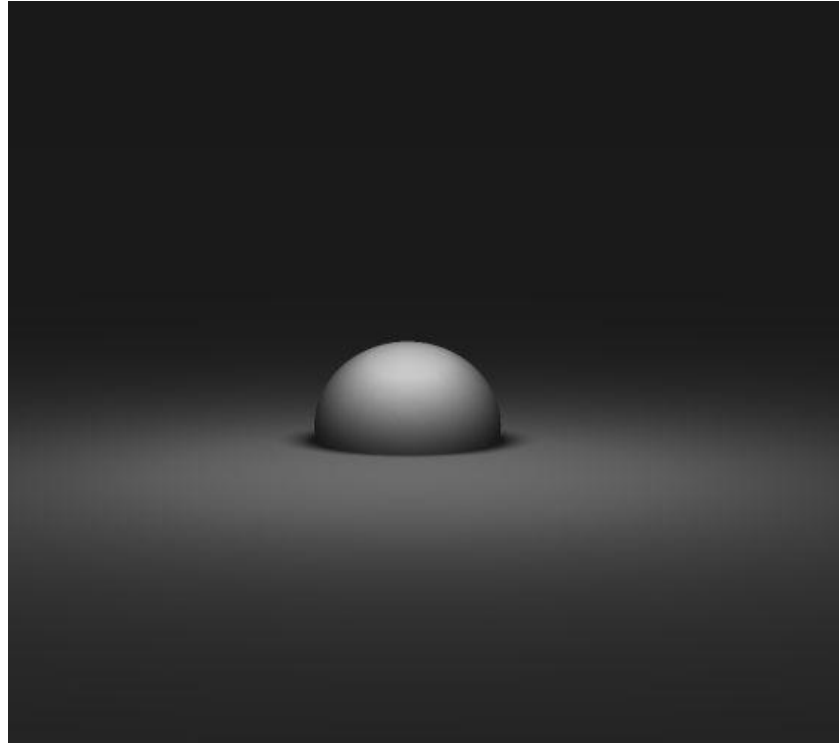
Soft Shadows



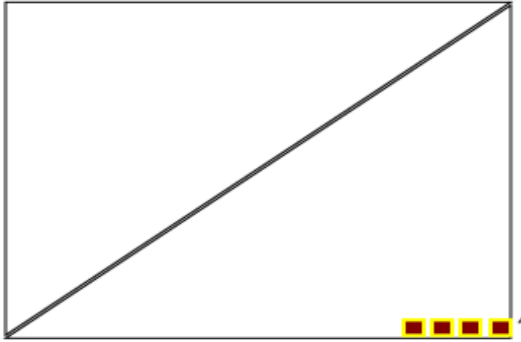
Soft Shadows



Soft Shadows

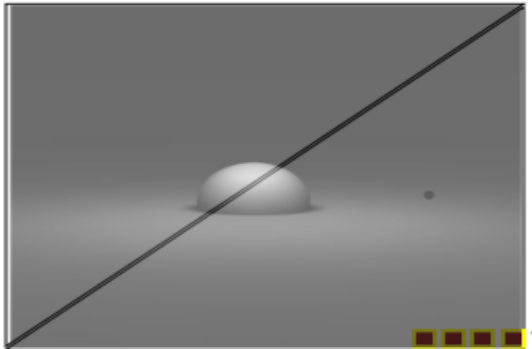


RM in Pixel shaders



1) two triangles (fullscreen)

2) each pixel runs the same **pixel shader**



3) sphere tracing in GLSL **pixel shader**

```

varying vec3 r;
uniform float t;
uniform vec4 m;
float plane(vec3 r)
{ return r.y; }
float sphere(vec3 r, float R)
{ return length(r) - R; }

float f(vec3 r) // distance function
{
    float d=1e10;
    vec3 posr = r;
    d = sphere(r+vec3(0+m.x,-0.1+m.y,3.9), 0.5);
    d = min (d, plane(r));
    return d;
}
// credit: las / pouet thread
float shadow(vec3 p, vec3 l, float d, int i)
{
    float o;
    for (o = 0; i > 0.; i--) o += f(p+l*i*d).x;
    return clamp(o, 0.0, 1.0);
}
// credit: http://www.pouet.net/topic.php?which=7920&page=10, rar
vec3 normal(vec3 p)
{
    #define DR 2e-4
    vec3 dr = vec3(DR,0,0);

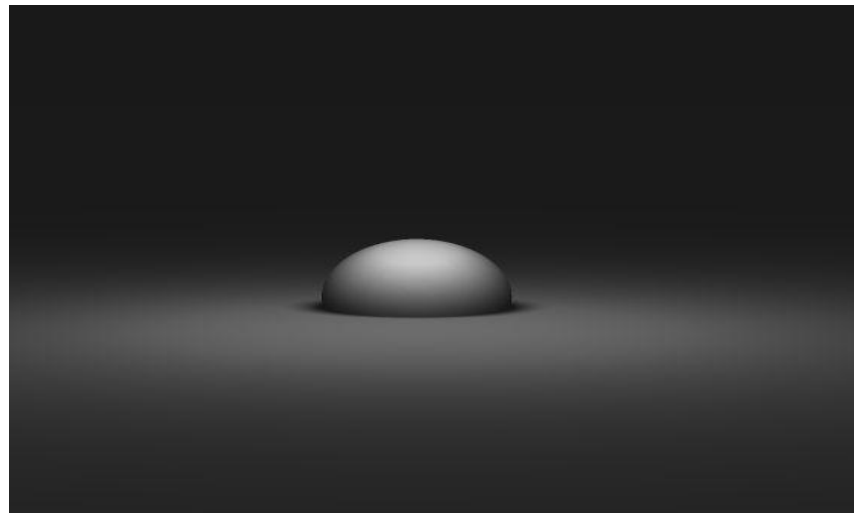
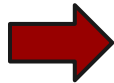
```

```

        return ( vec3( f(p+dr.xyz).x, f(p+dr.yxz).x, f(p+dr.yzx).x ) - f(p).x ) / DR;
    }
#define ITER 90
#define EPS 0.002
void main()
{
    vec3 camera = vec3(-0.1,1.8,-15.0);
    vec3 p = vec3(r.x, r.y+1.2, -10.2);
    vec3 lightv = vec3(0.0,2.5,-5.0);

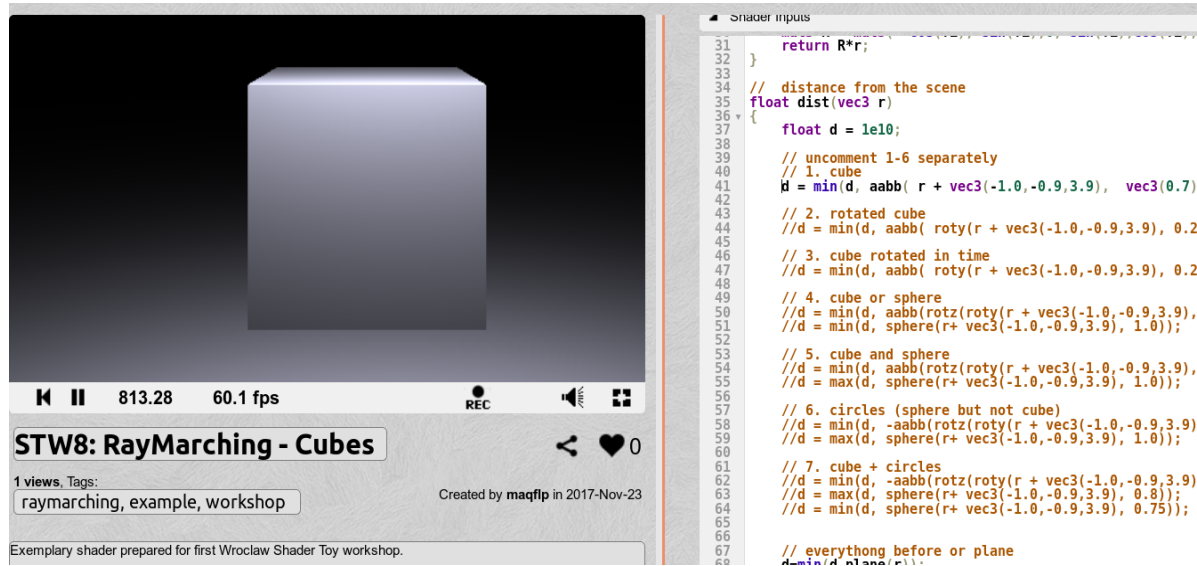
    vec3 dir = normalize(p-camera);
    int mat=1;
    float d;
    for(int i=0; i<ITER; i++)
    {
        d = f(p);
        if(d < EPS)
            break;
        p = p + dir * d;
    }
    vec3 n = normal(p);
    float light = 1.0 + 0.5*(dot(n,lightv));
    light = light / pow(length(lightv-p),2);
    vec3 lightdir = normalize(lightv-p);
    light *= shadow(p, lightdir, 0.02, 10);
    vec4 color = 0.1+0.9*light * vec4(1,1,1,1);
    gl_FragColor = color;
}

```



Kod

Example



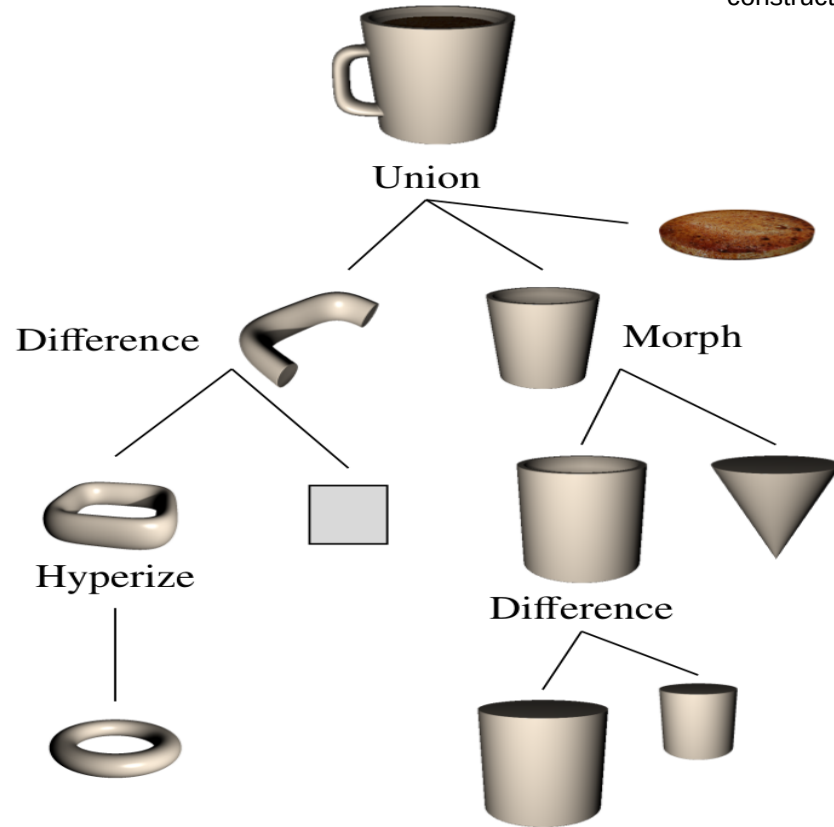
The image shows a screenshot of a raymarching scene. On the left, a 3D cube is rendered in a dark environment. The scene is displayed in a window with a play button, a pause button, and a frame rate of 60.1 fps. Below the scene, the title "STW8: RayMarching - Cubes" is visible, along with a share icon and a heart icon. The code editor on the right shows the following shader code:

```
31 }
32 return R*r;
33 }
34 // distance from the scene
35 float dist(vec3 r)
36 {
37     float d = 1e10;
38     // uncomment 1-6 separately
39     // 1. cube
40     d = min(d, aabb( r + vec3(-1.0,-0.9,3.9),  vec3(0.7)))
41
42     // 2. rotated cube
43     //d = min(d, aabb( roty(r + vec3(-1.0,-0.9,3.9), 0.2)
44
45     // 3. cube rotated in time
46     //d = min(d, aabb( roty(r + vec3(-1.0,-0.9,3.9), 0.2*
47
48     // 4. cube or sphere
49     //d = min(d, aabb(rotz(roty(r + vec3(-1.0,-0.9,3.9),i
50     //d = min(d, sphere(r+ vec3(-1.0,-0.9,3.9), 1.0));
51
52     // 5. cube and sphere
53     //d = min(d, aabb(rotz(roty(r + vec3(-1.0,-0.9,3.9),i
54     //d = max(d, sphere(r+ vec3(-1.0,-0.9,3.9), 1.0));
55
56     // 6. circles (sphere but not cube)
57     //d = min(d, -aabb(rotz(roty(r + vec3(-1.0,-0.9,3.9),
58     //d = max(d, sphere(r+ vec3(-1.0,-0.9,3.9), 1.0));
59
60     // 7. cube + circles
61     //d = min(d, -aabb(rotz(roty(r + vec3(-1.0,-0.9,3.9),
62     //d = max(d, sphere(r+ vec3(-1.0,-0.9,3.9), 0.8));
63     //d = min(d, sphere(r+ vec3(-1.0,-0.9,3.9), 0.75));
64
65
66
67 // everything before or plane
68 d = min(d, plane(r));
```

<https://www.shadertoy.com/view/XtlfR7>

Coffee Mug with CSG

constructive solid geometry



Tim Reiner, Gregor Muckl, Carsten Dachsbacher, Interactive Modeling of Implicit Surfaces using a Direct Visualization Approach with Signed Distance Functions

Thank you for your attention!

Shaders are at:
maciejmatyka.blogspot.com

[maq/floppy](#)