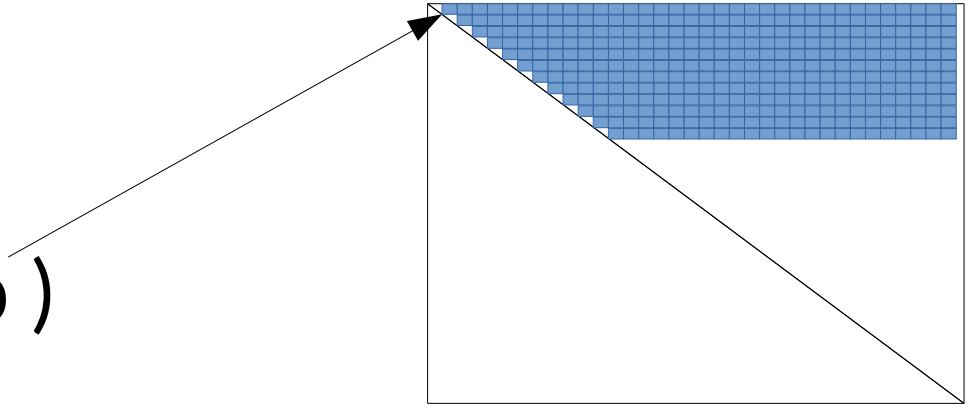
Smooth introduction to shaders

by maq / floppy

Each pixel runs the same shader program

http://www.iquilezles.org/www/material/nvscene2008/rwwtt.pdf

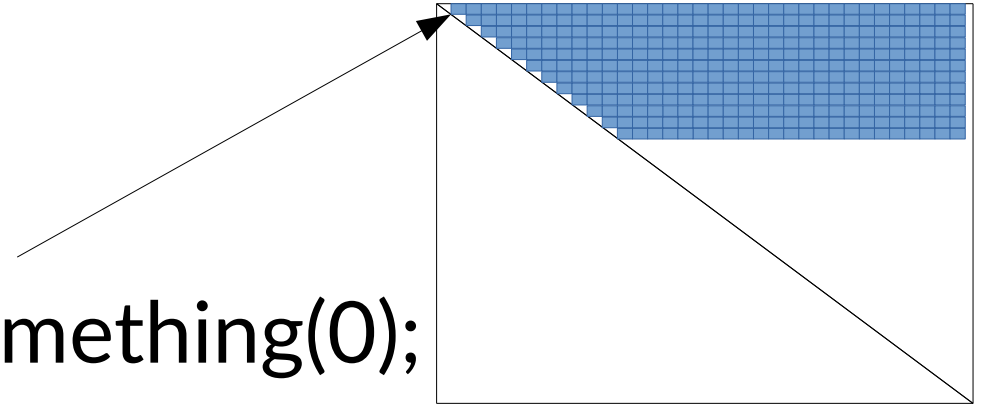
LOOP WAY



```
for( each pixel p )
{
    outputColor = doSomething( p );
}
```

http://www.iquilezles.org/www/material/nvscene2008/rwwtt.pdf

SHADER WAY



OutputColor(0) = doSomething(0);

OutputColor(1) = doSomething(1);

OutputColor(2) = doSomething(2);

OutputColor(3) = doSomething(3);

OutputColor(4) = doSomething(4);

http://www.iquilezles.org/www/material/nvscene2008/rwwtt.pdf

Part 1: Shapes

Part 2: Deformations

Part 3: Ray Marching

Distance functions

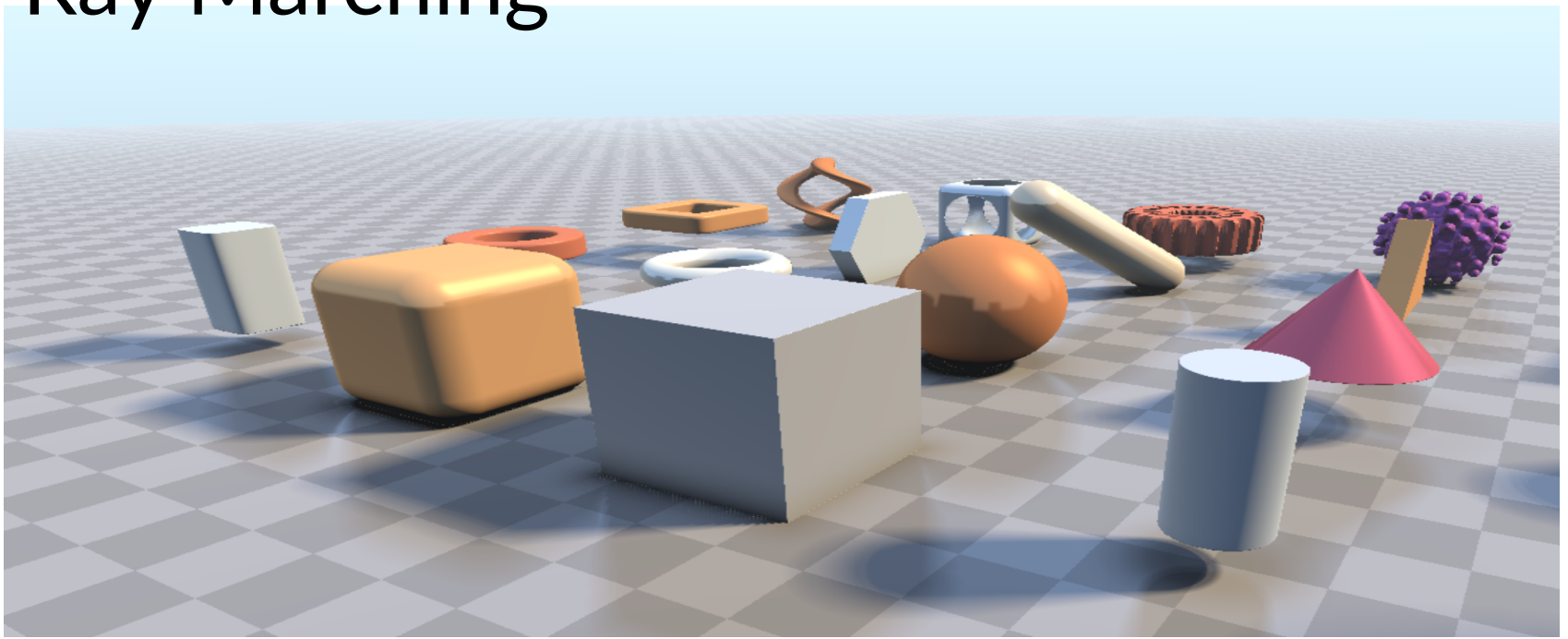
Toon Cloud by AntoineC
https://www.shadertoy.com/view/4t23RR

Plane Deformation

Deform - square tunnel by IQ
https://www.shadertoy.com/view/Ms2SWW

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

Ray Marching



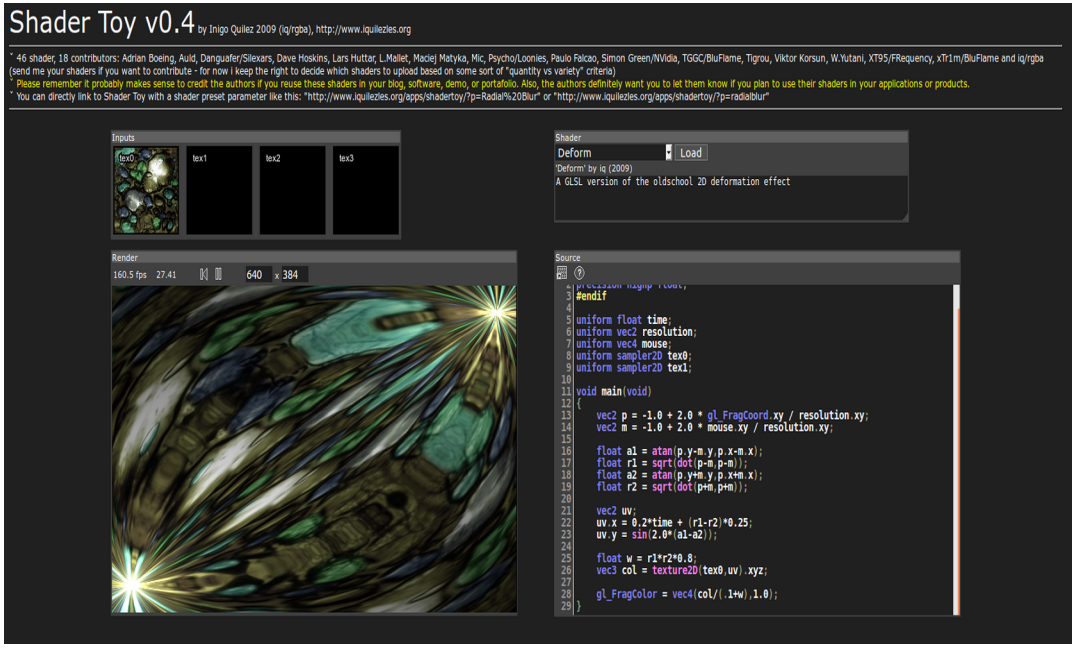
https://www.shadertoy.com/view/Xds3zN

Raymarching – Primitives
by IQ

Shader Toy

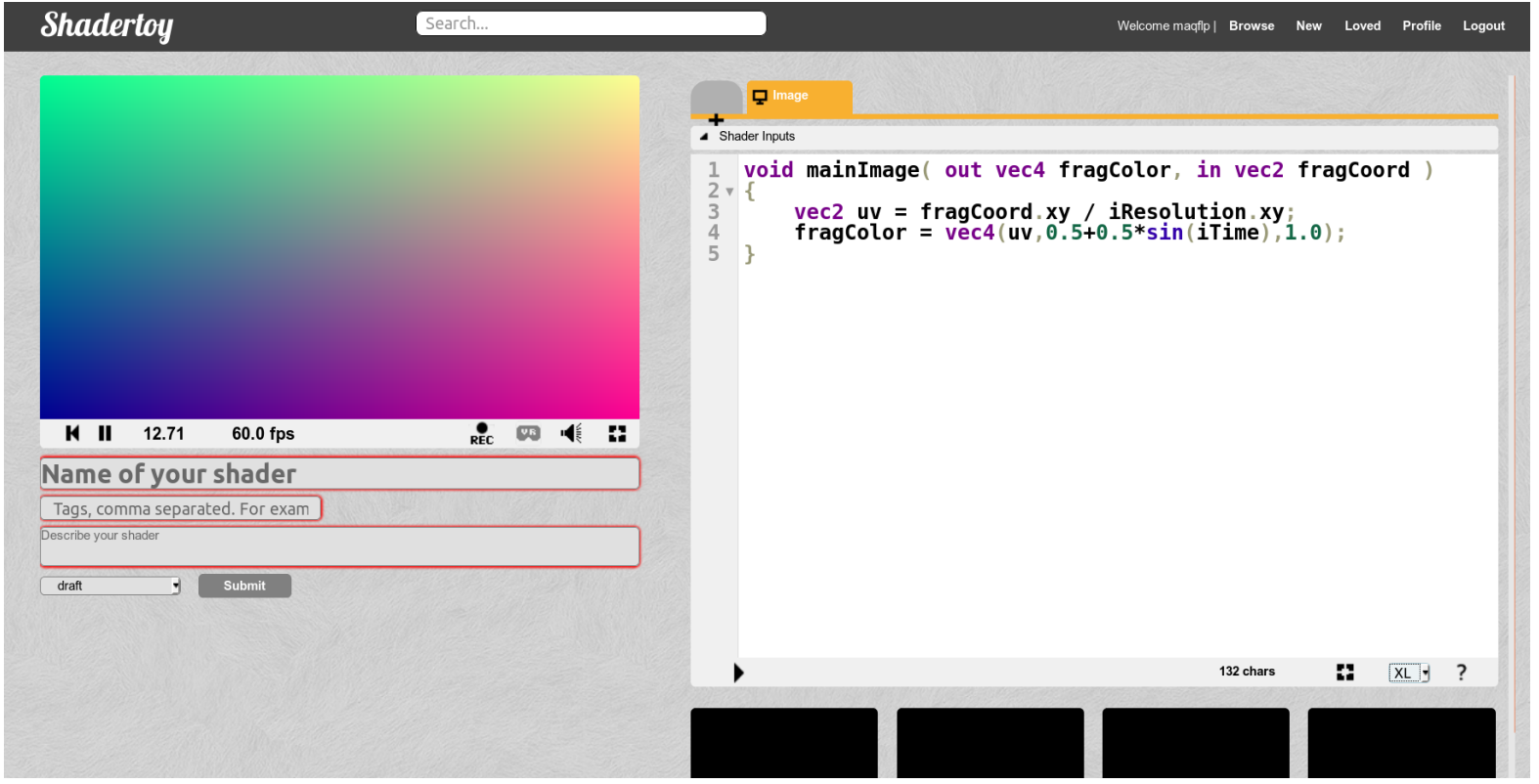
- Authors: Inigo Quilez (IQ), Pol Jeremias (BeautyPi)
- WebGL application
- Database for shaders
- Source of inspiration for many people
- You may learn a lot here
- Not only simple shaders
  (multipass games, demos, etc.)

Version 0.4



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

http://ShaderToy.com



Alternatives?

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

Part 1: Shapes

1: Color

```
void mainImage( out vec4 fragColor, in vec2 fragCoord )
{
    vec3 col = vec3(0.0, 0.0, 0.0);

    fragColor = vec4( col, 1.0);
}
```

Red

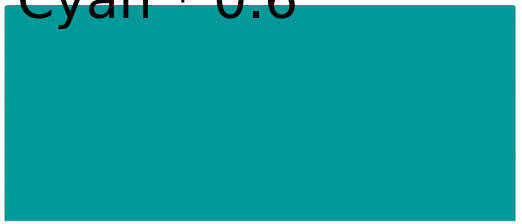
start

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

Cyan

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

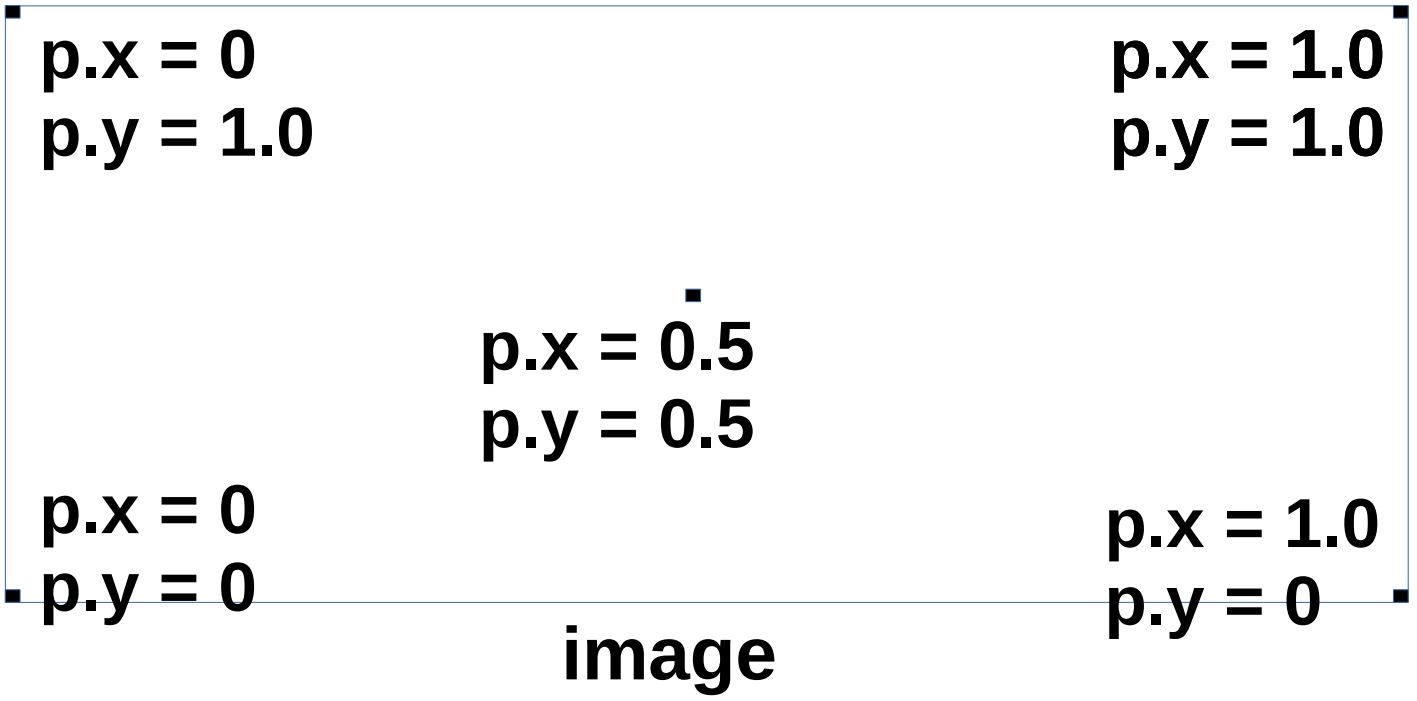
Cyan * 0.6



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

2: Position

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



2: Position

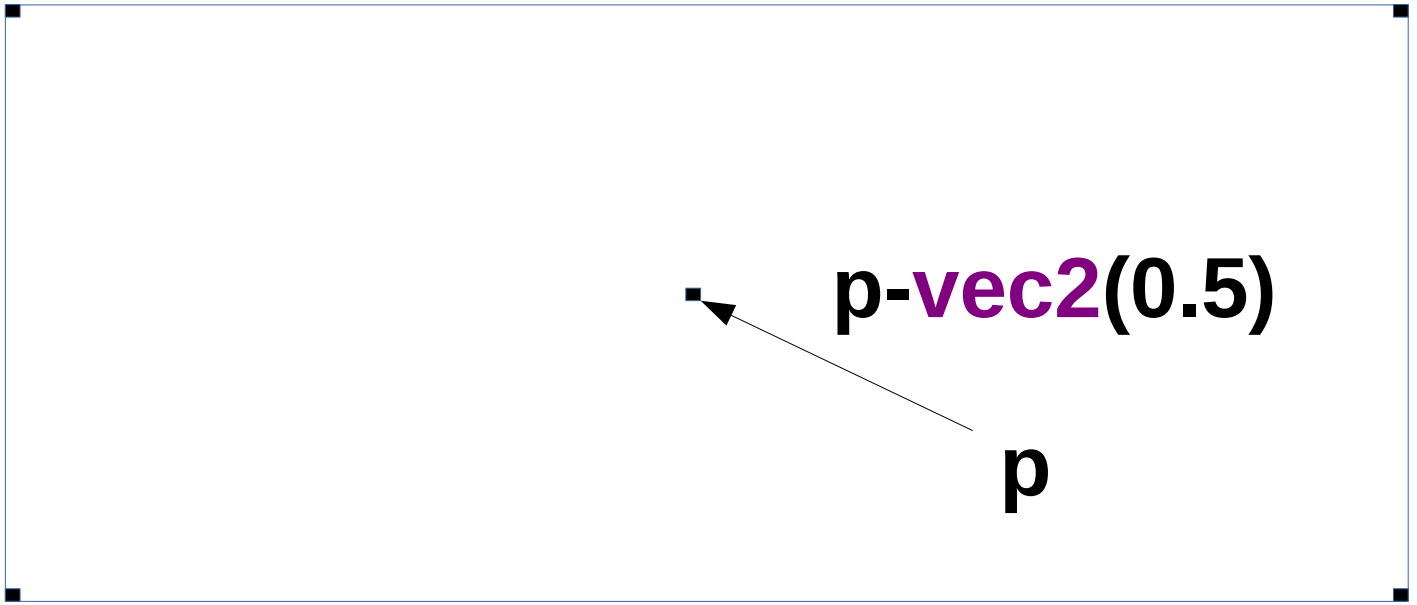
p.x=0.0

p.x=1.0

```
3     vec2 p = fragCoord.xy / iResolution.xy;
4
5     vec3 col = vec3(0, 1.0, 1.0) * p.x;
```

3: Distance

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



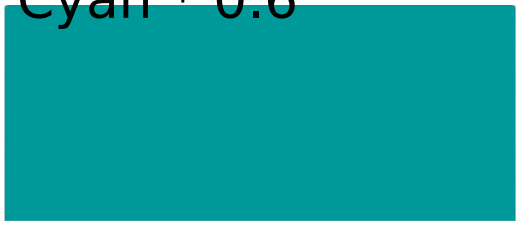
Red

```
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

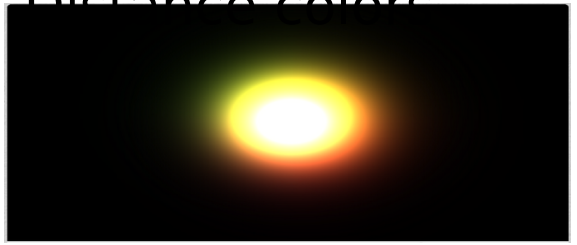
start

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

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);
```

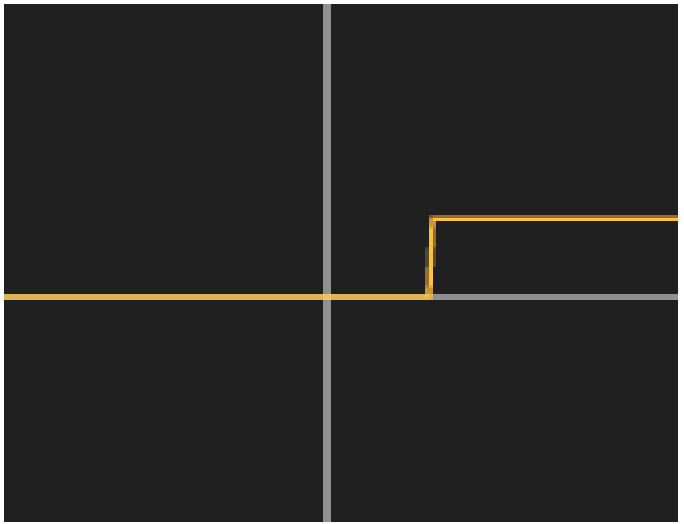
Distance colors



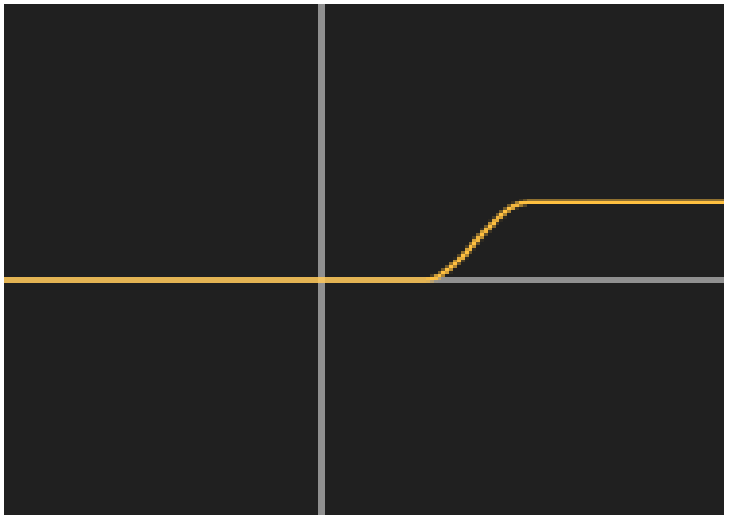
start

```
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);
}
```

Step functions



step(1,x)

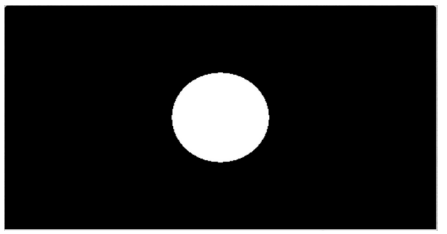


smoothstep(1,2,x)

http://www.iquilezles.org/apps/graphtoy/

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);
```

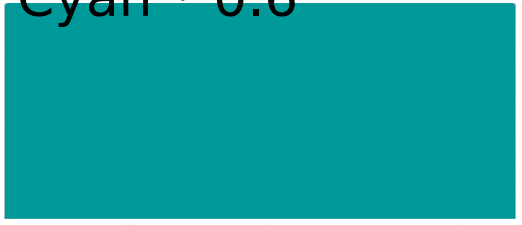
Red

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

Cyan

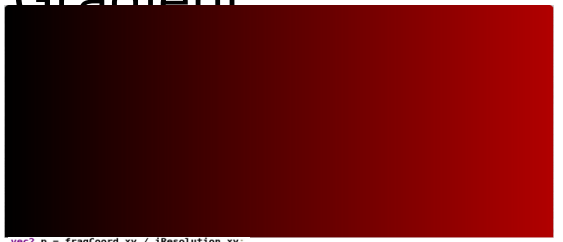
```
vec3 col = vec3(0, 1.0, 1.0);
```

Cyan * 0.6



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

Gradient

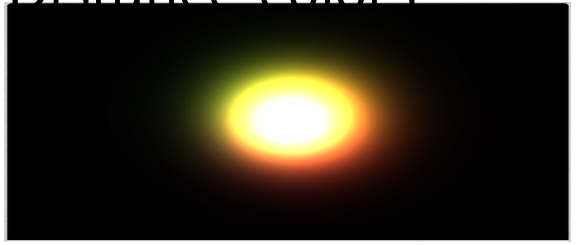


```
vec2 p = fragCoord.xy / iResolution.xy;
vec3 col = vec3(1.0, 0.0, 0.0) * p.x;
fragColor = vec4( col * 0.7, 1.0);
```

Radial

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

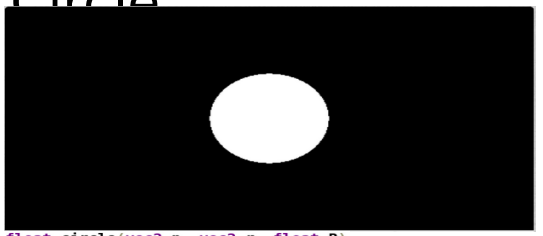
Distance colors



```
void mainImage( out vec4 fragColor, in vec2 fragCoord )
{
    vec2 p = fragCoord.xy / 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);
}
```

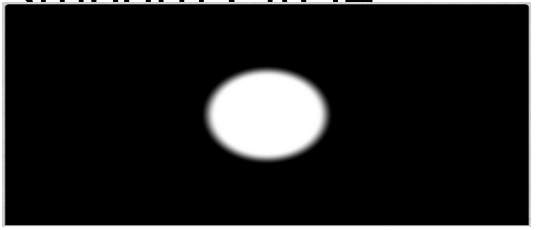
Circle

start



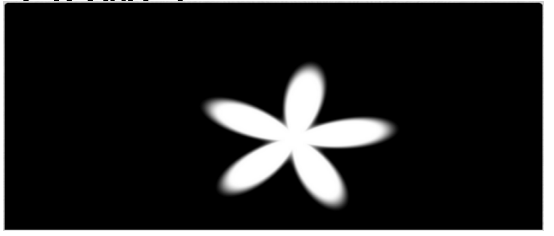
```
float circle(vec2 p, vec2 r, float R)
{
    return step(length(p-r), R);
}
col *= circle( p, vec2(0.0,0.0), 0.4);
```

Smooth Circle

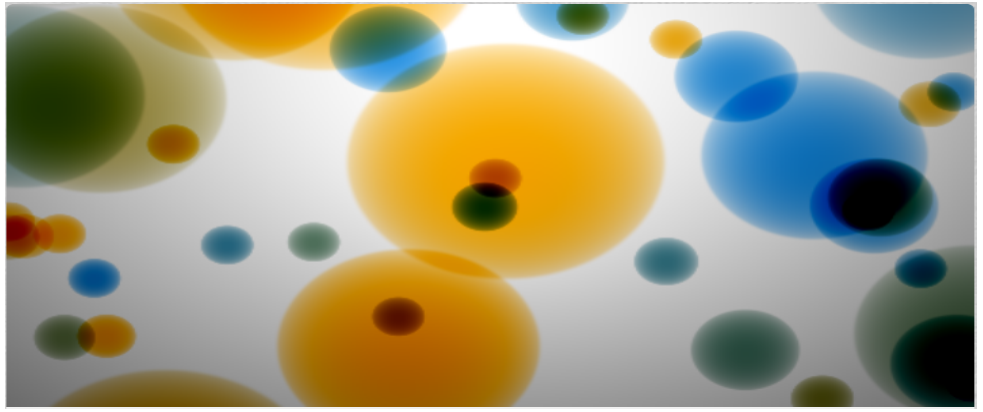


```
float circle(vec2 p, vec2 r, float R)
{
    return 1.0-smoothstep(R-0.05, R+0.05,length(p-r));
}
```

Flower



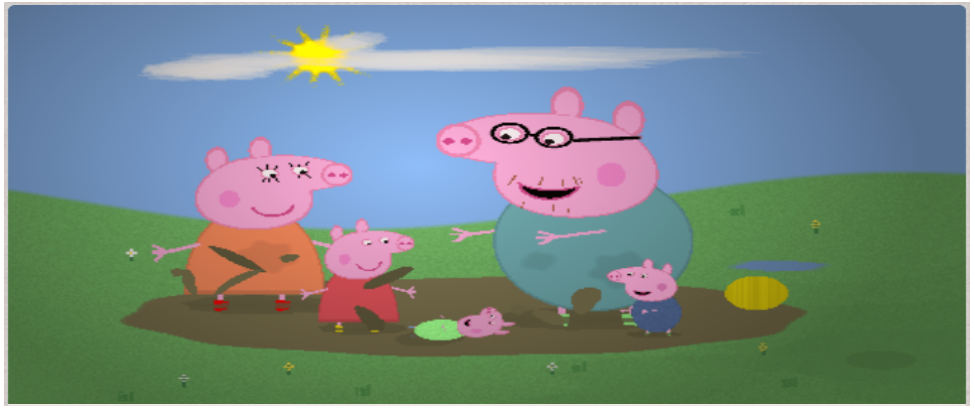
```
float flower(vec2 p, vec2 r, float R)
{
    float phi = atan(p.y, p.x);
    R*= 0.2+0.6*(1.0+sin(phi*5.0+0.7*iTime));
    return smoothstep(R, R+0.15,length(p-r));
}
```



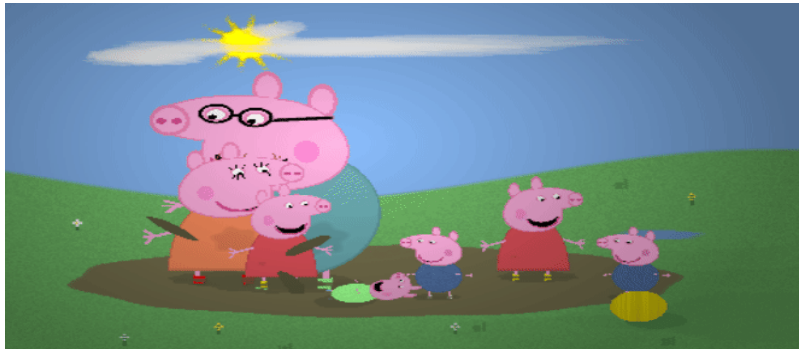
https://www.shadertoy.com/view/4dl3zn

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

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



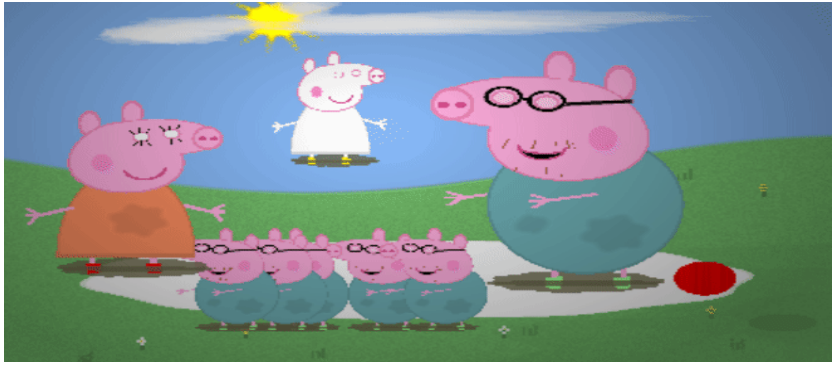
https://www.shadertoy.com/view/Xsf3z8



Bartek 2

Robert 7

Karol 8

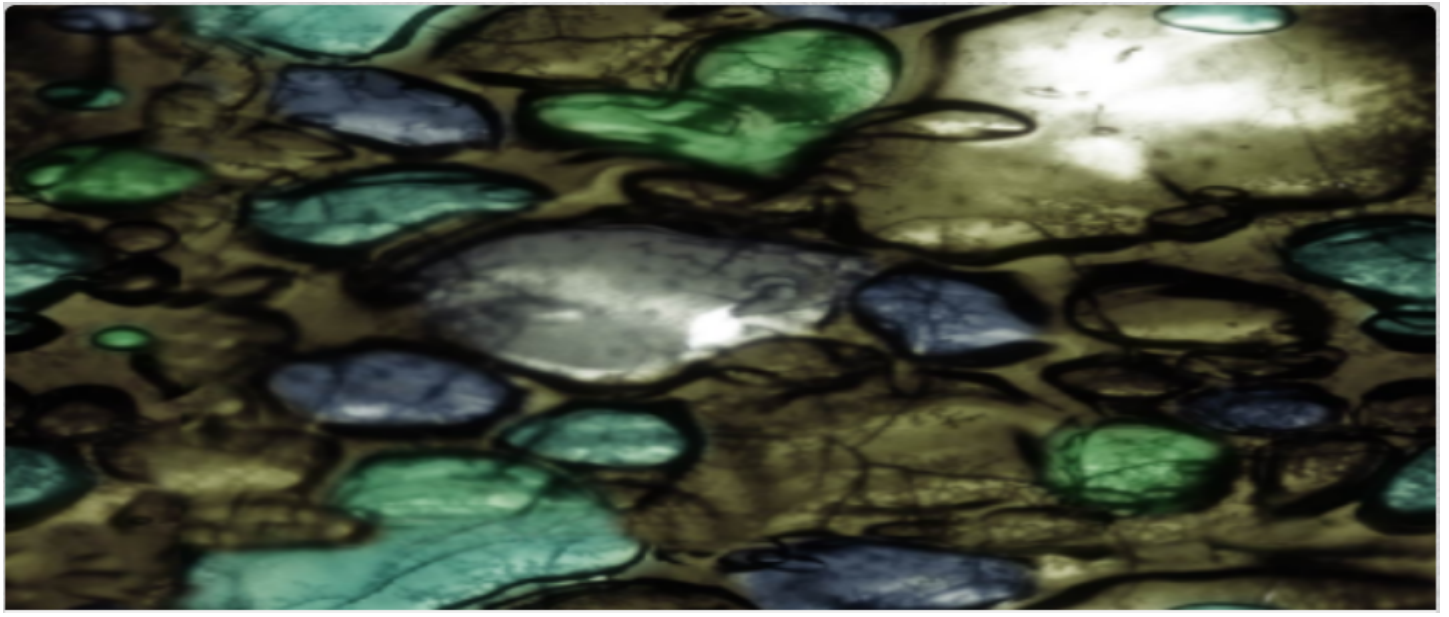


Heart - 2D by IQ

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

```
void mainImage( out vec4 fragColor, in vec2 fragCoord )
{
    vec2 p = (2.0*fragCoord-iResolution.xy)/min(iResolution.y,iResolution.x);

    // background color
    vec3 bcol = vec3(1.0,0.8,0.7-0.07*p.y)*(1.0-0.25*length(p));

    // animate
    float tt = mod(iTime,1.5)/1.5;
    float ss = pow(tt,.2)*0.5 + 0.5;
    ss = 1.0 + ss*0.5*sin(tt*6.2831*3.0 + p.y*0.5)*exp(-tt*4.0);
    p *= vec2(0.5,1.5) + ss*vec2(0.5,-0.5);

    // shape
#if 0
    p *= 0.8;
    p.y = -0.1 - p.y*1.2 + abs(p.x)*(1.0-abs(p.x));
    float r = length(p);
    float d = 0.5;
#else
    p.y -= 0.25;
    float a = atan(p.x,p.y)/3.141593;
    float r = length(p);
    float h = abs(a);
    float d = (13.0*h - 22.0*h*h + 10.0*h*h*h)/(6.0-5.0*h);
#endif

    // color
    float s = 0.75 + 0.75*p.x;
    s *= 1.0-0.4*r;
    s = 0.3 + 0.7*s;
    s *= 0.5+0.5*pow( 1.0-clamp(r/d, 0.0, 1.0 ), 0.1 );
    vec3 hcol = vec3(1.0,0.5*r,0.3)*s;

    vec3 col = mix( bcol, hcol, smoothstep( -0.01, 0.01, d-r) );

    fragColor = vec4(col,1.0);
}
```

Part 2: Deformations



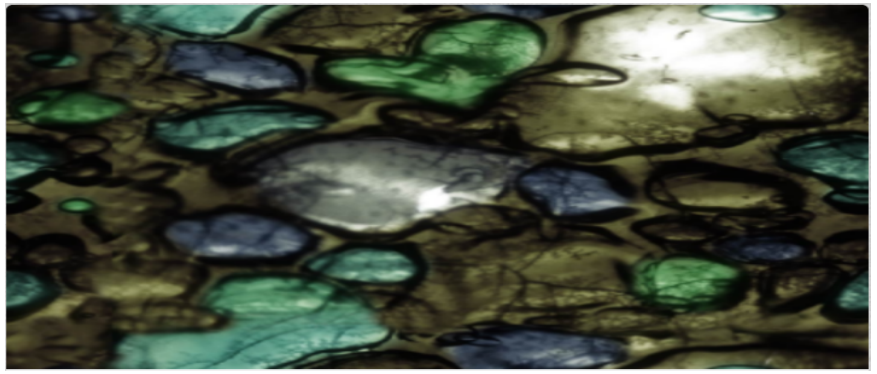
```
void mainImage( out vec4 fragColor, in vec2 fragCoord )
{
    vec2 uv = fragCoord.xy / iResolution.xy;

    vec3 col = texture(iChannel0, uv+iTime*0.1 ).rgb;

    fragColor = vec4(col, 1.0);
}
```

Modify **texture position uv** according to:

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

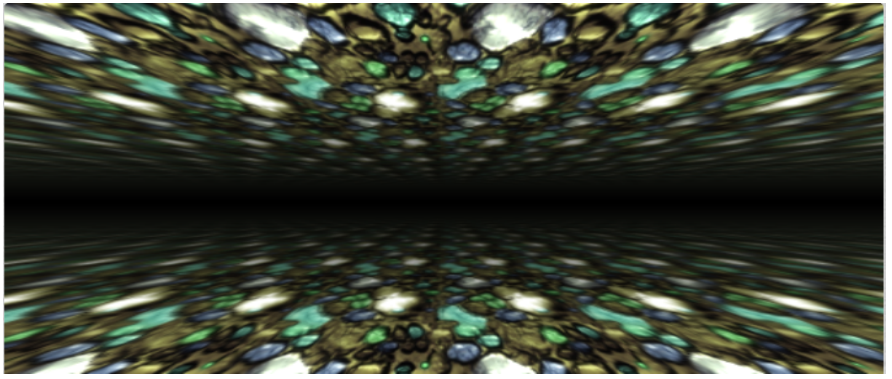
etc.

Deformations

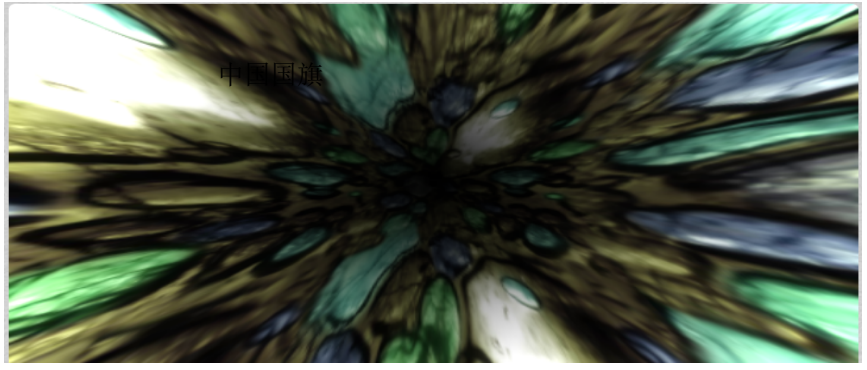


```
7    vec2 uv = p;
```

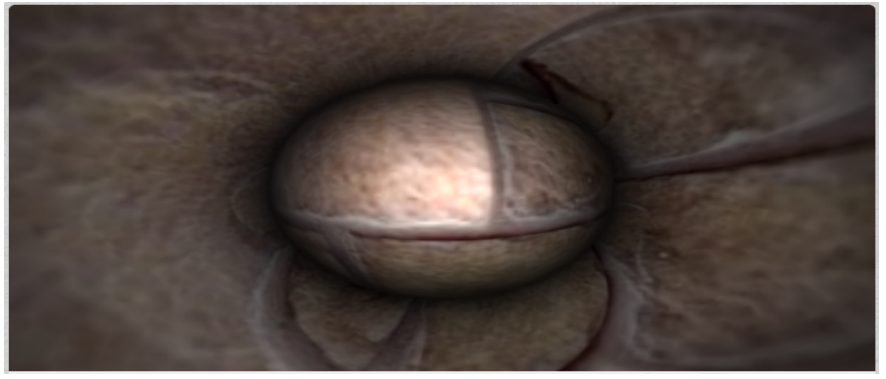
start



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

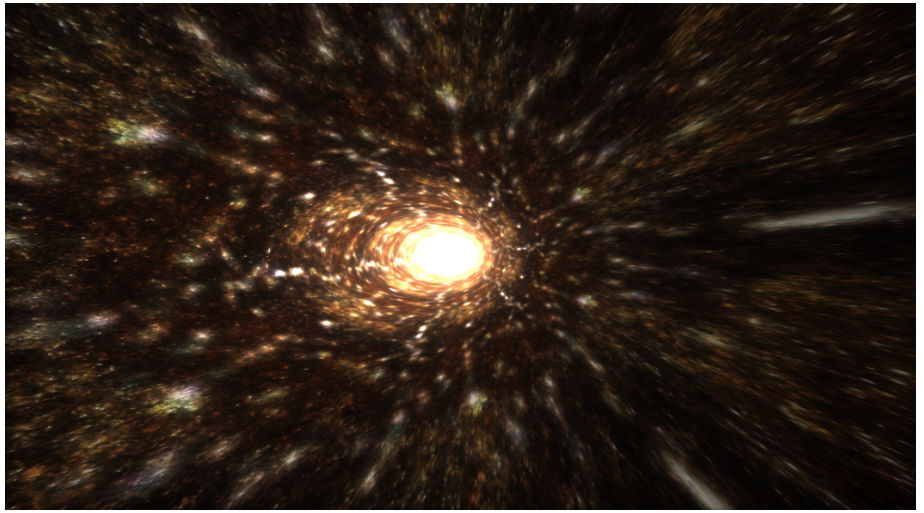


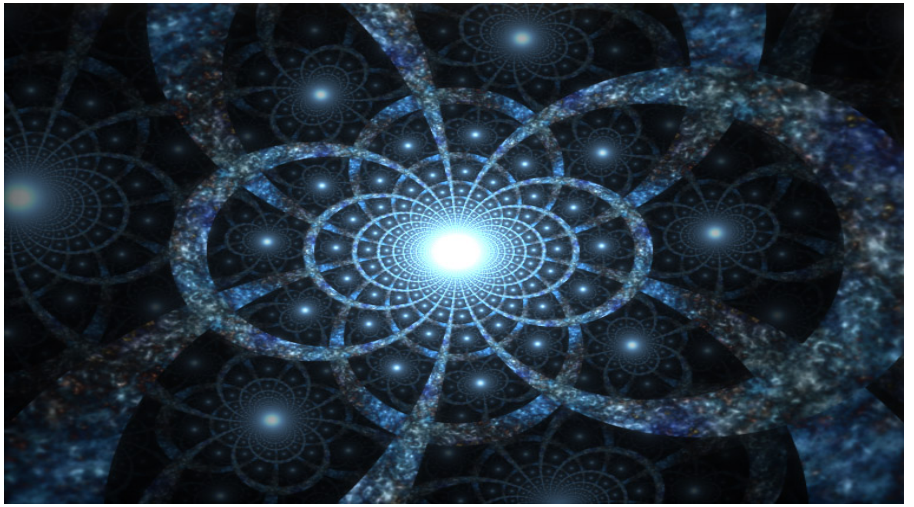
```
vec2 uv;
float d = length(p)+0.5;
uv=vec2(p)/d;
```

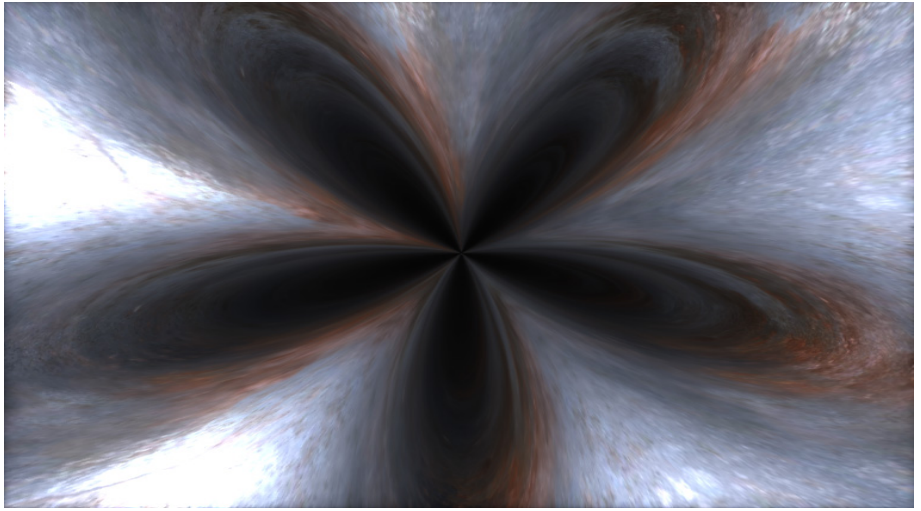


https://www.shadertoy.com/view/4dl3zS

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

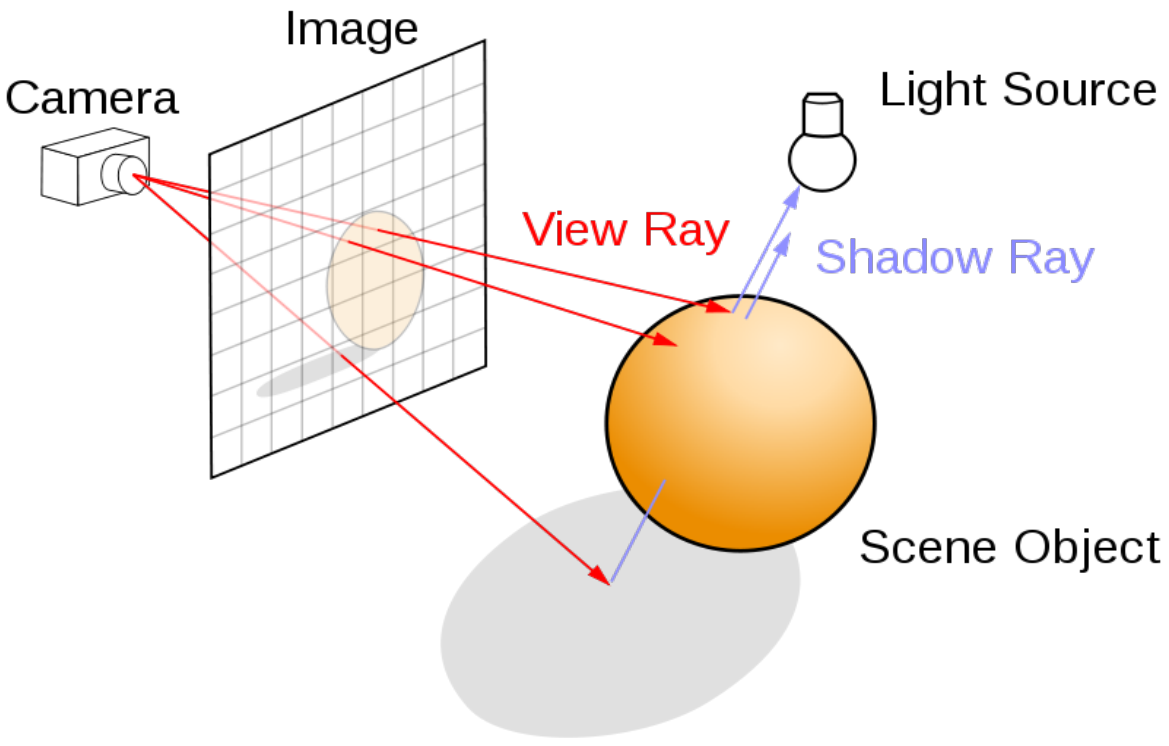
http://www.youtube.com/watch?v=foR64ZObAXA

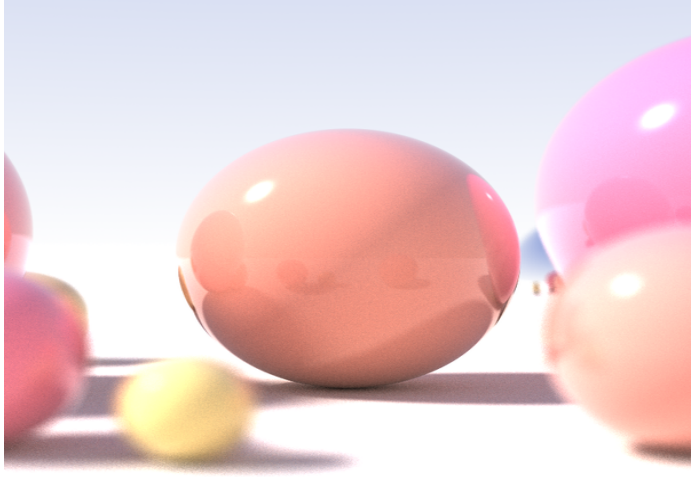
Sult 4kb intro tech

Sult .avi > 80 Mb file,

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

Ray Tracing



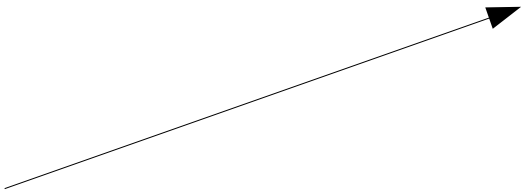


Ray Tracing

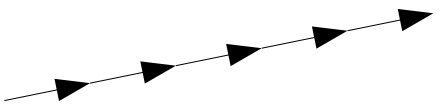
Problems: - soft shadows
- many objects, fractals
- ray/scene intersections cost a lot

Ray Marching

Ray Tracing Ray-scene intersections



Ray Marching Marching in the scene along the ray



Iterations

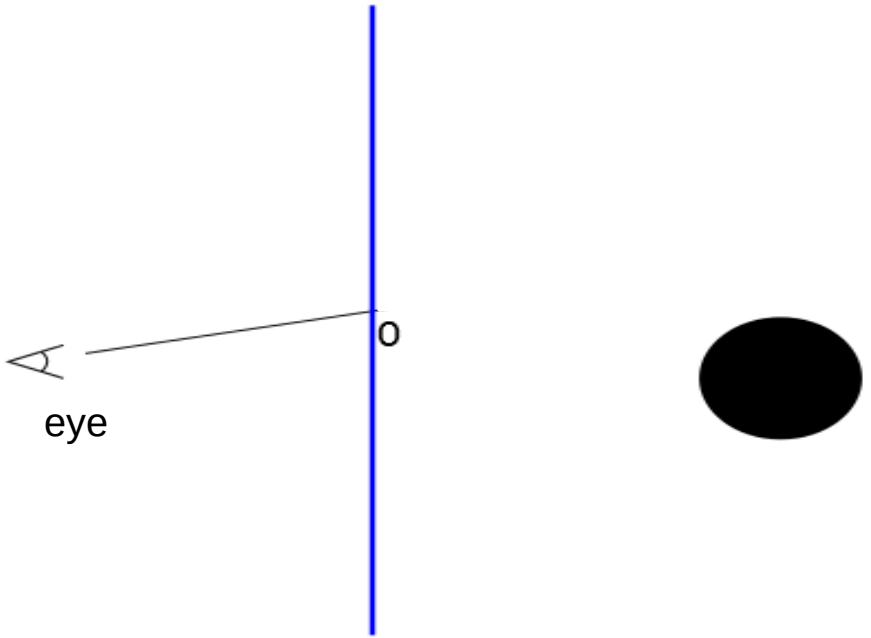
Ray:

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

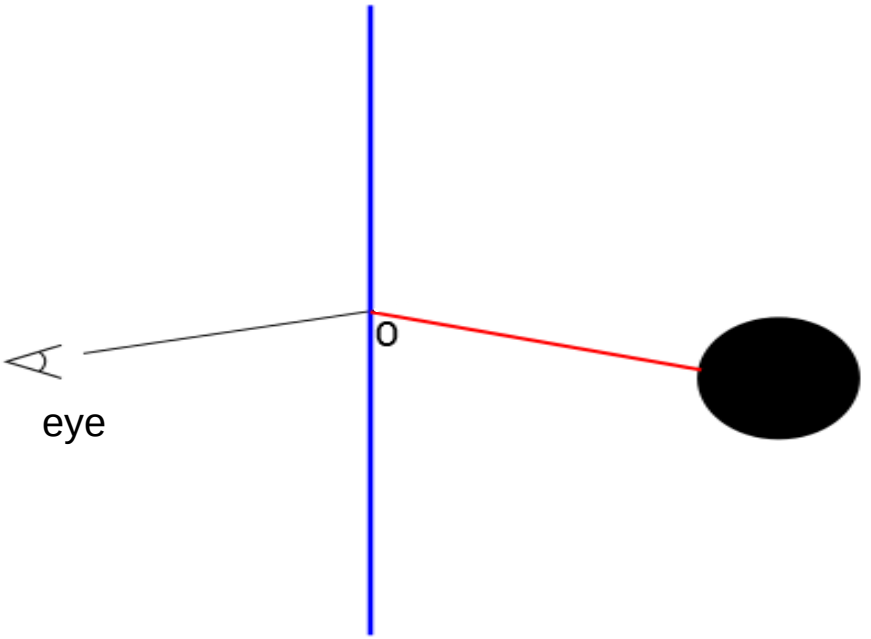
Seek for t_i minimum distance to any object in the scene:

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

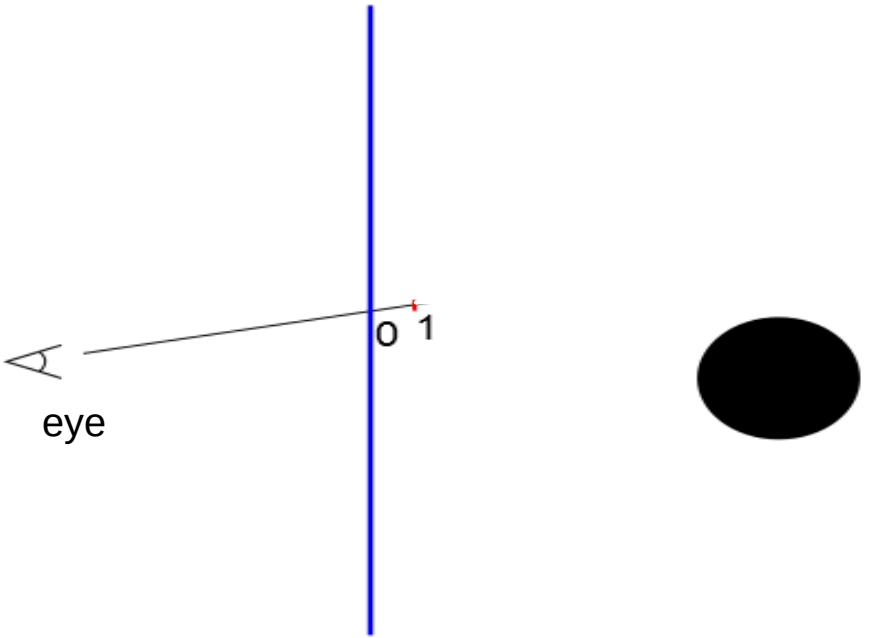
Ray Marching



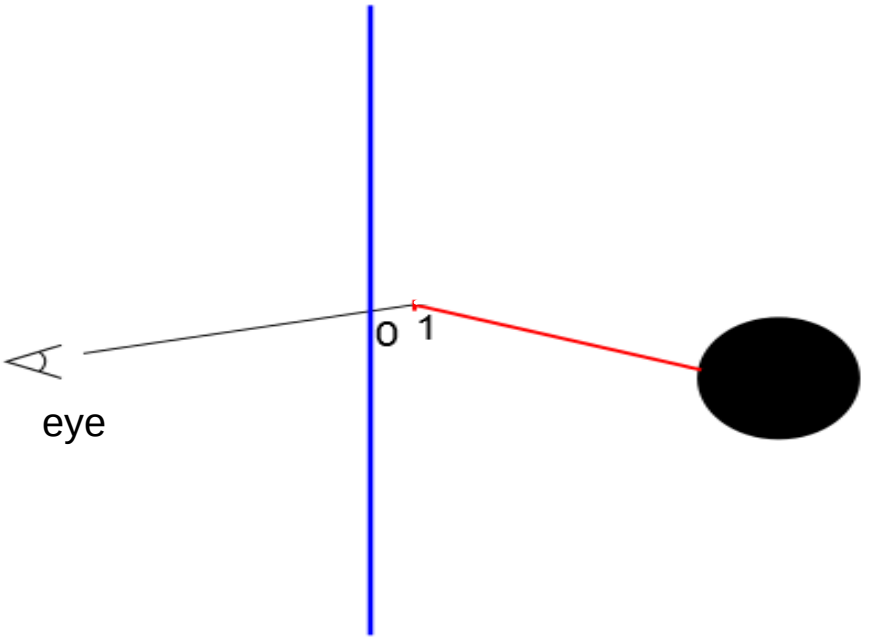
Ray Marching



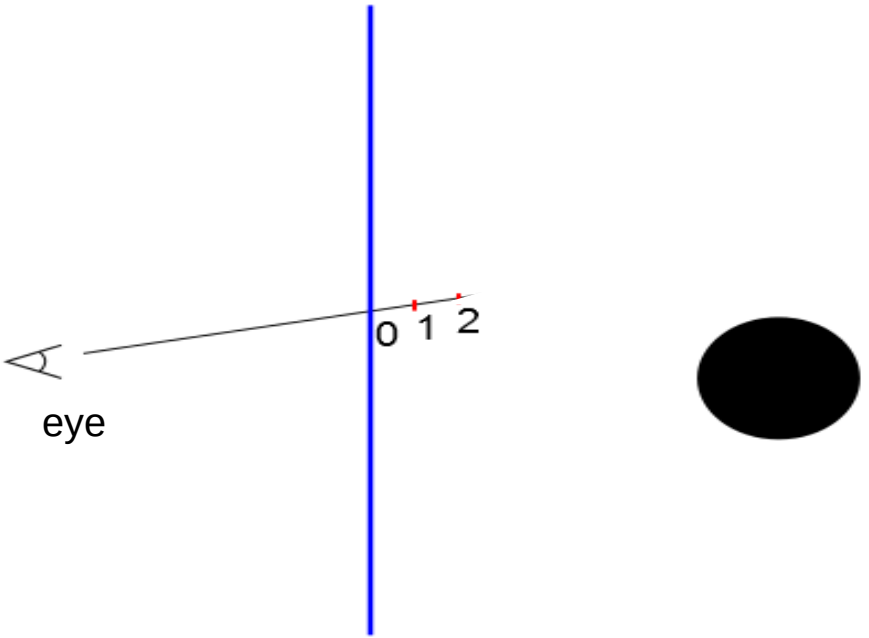
Ray Marching



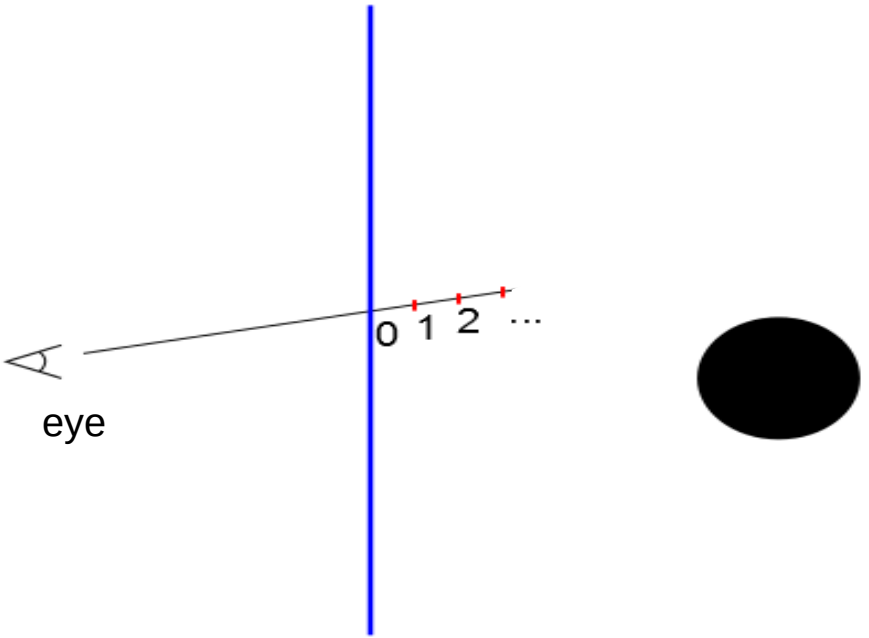
Ray Marching



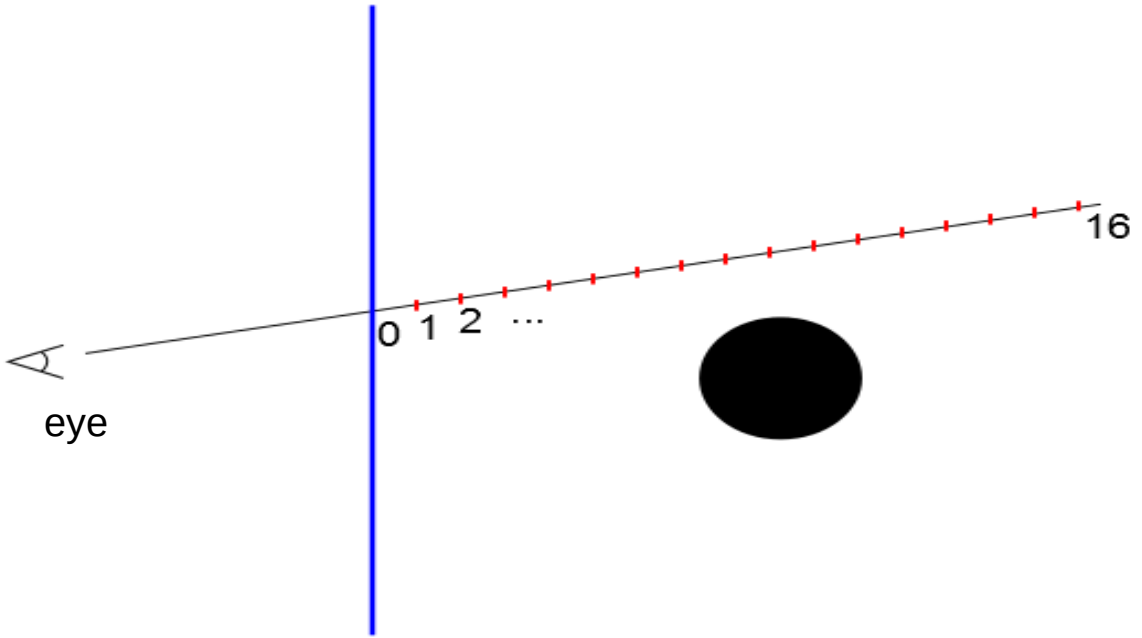
Ray Marching



Ray Marching



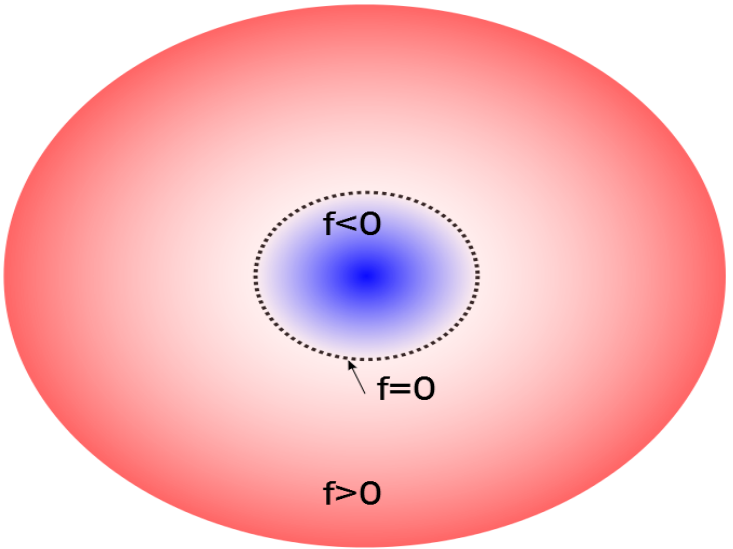
Ray Marching



Distance Fields

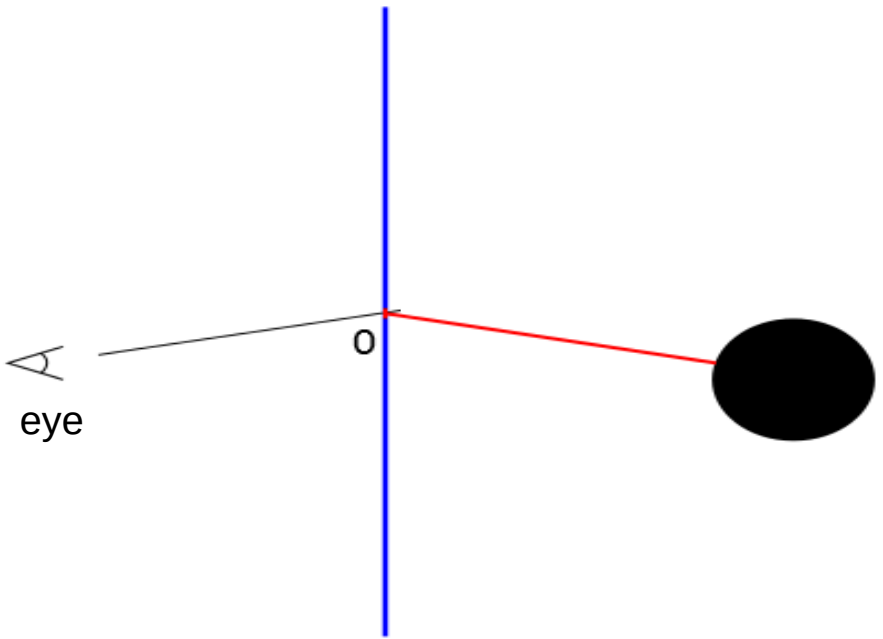
- Object as an *implicit function*
- For each position **x** in the scene

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

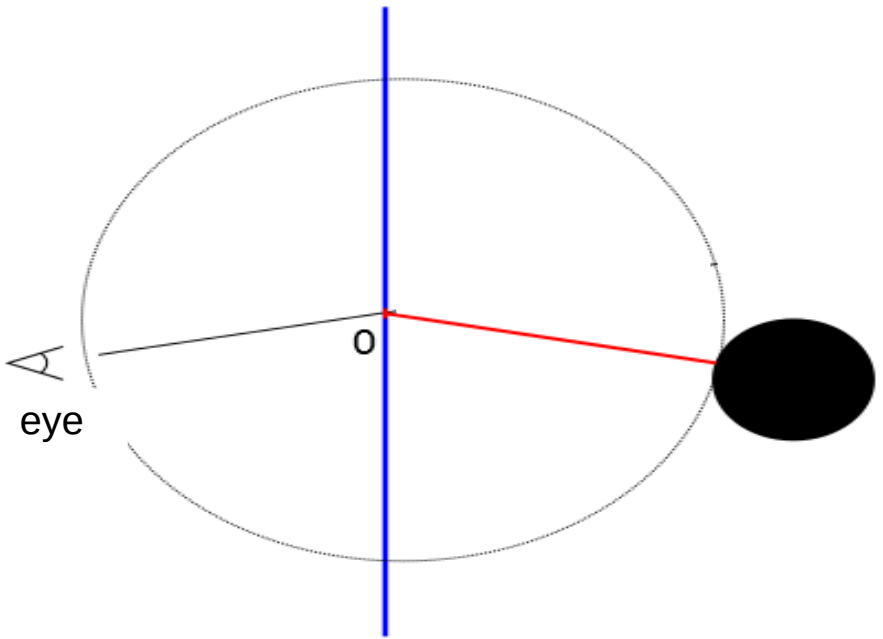


- f proportional to the distance

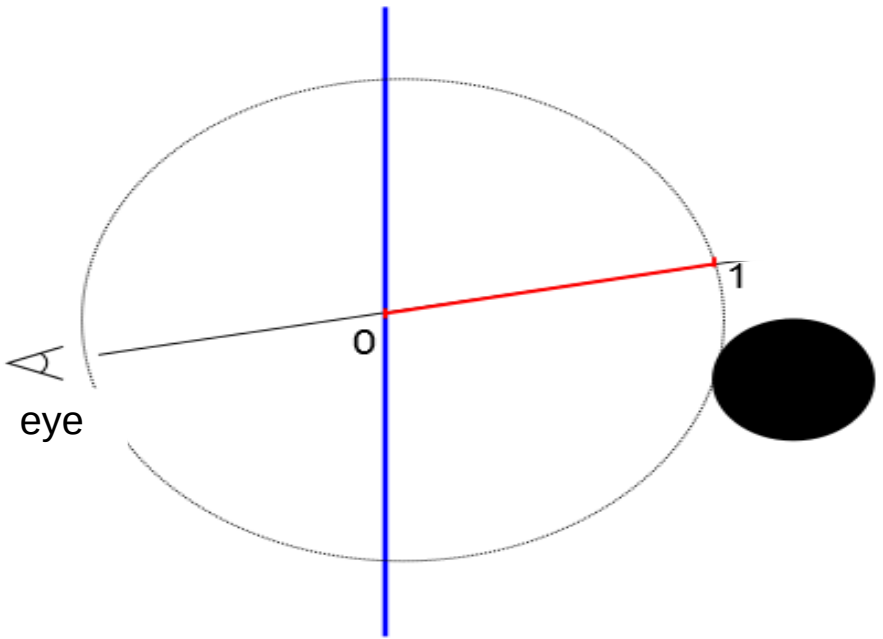
Sphere Tracing



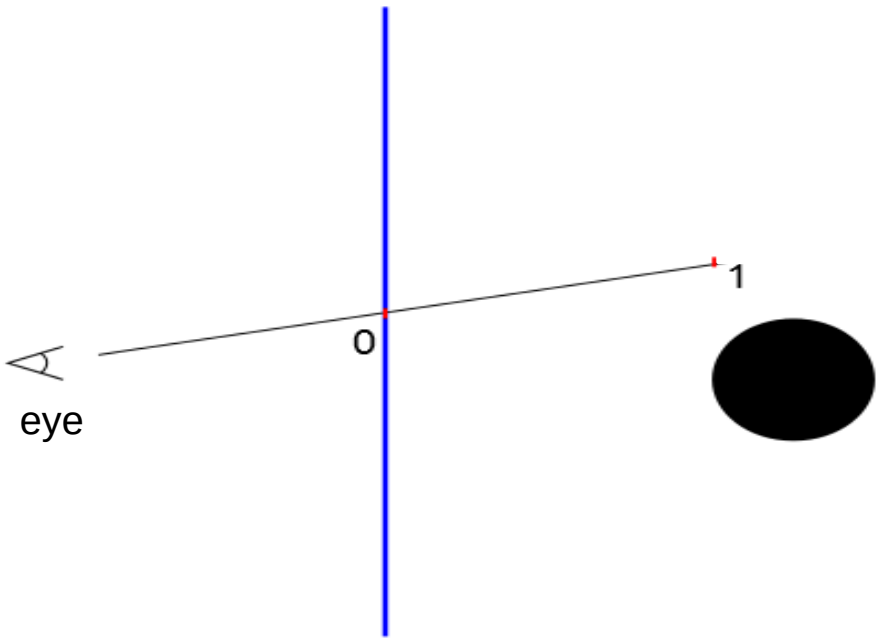
Sphere Tracing



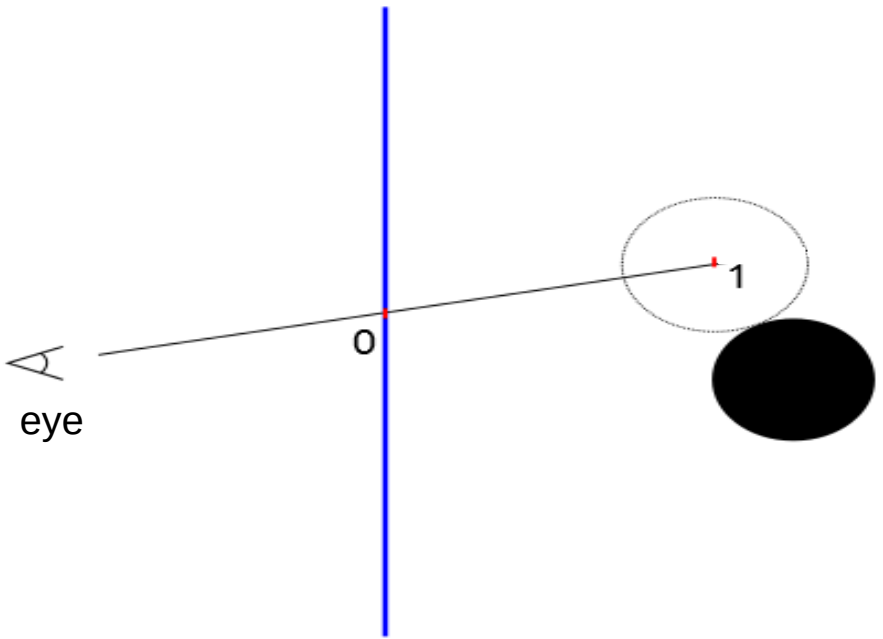
Sphere Tracing



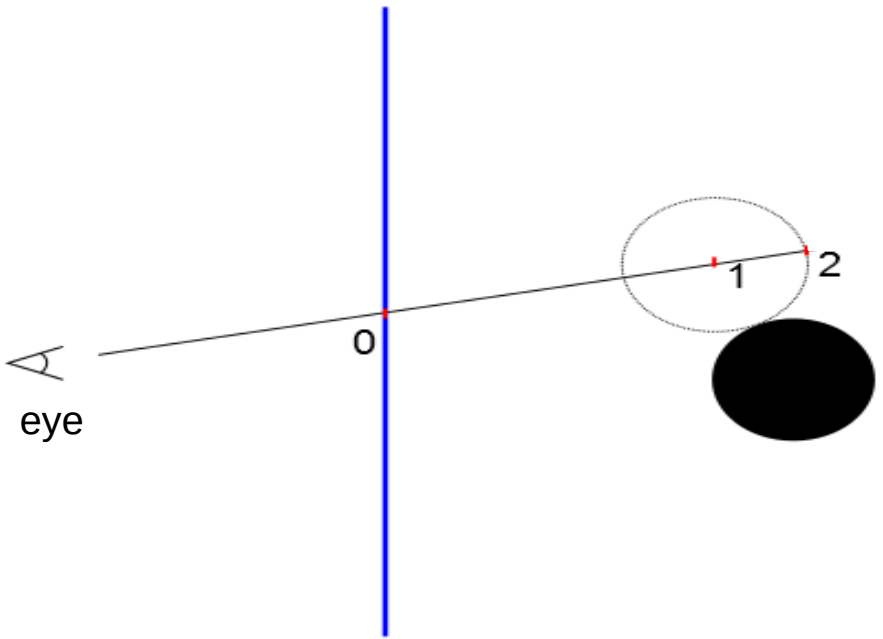
Sphere Tracing



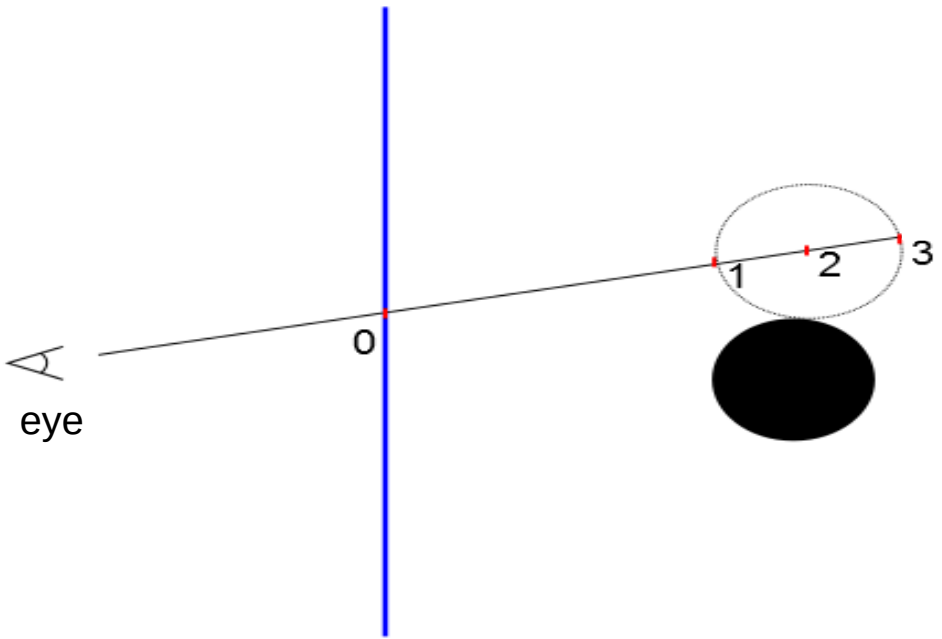
Sphere Tracing



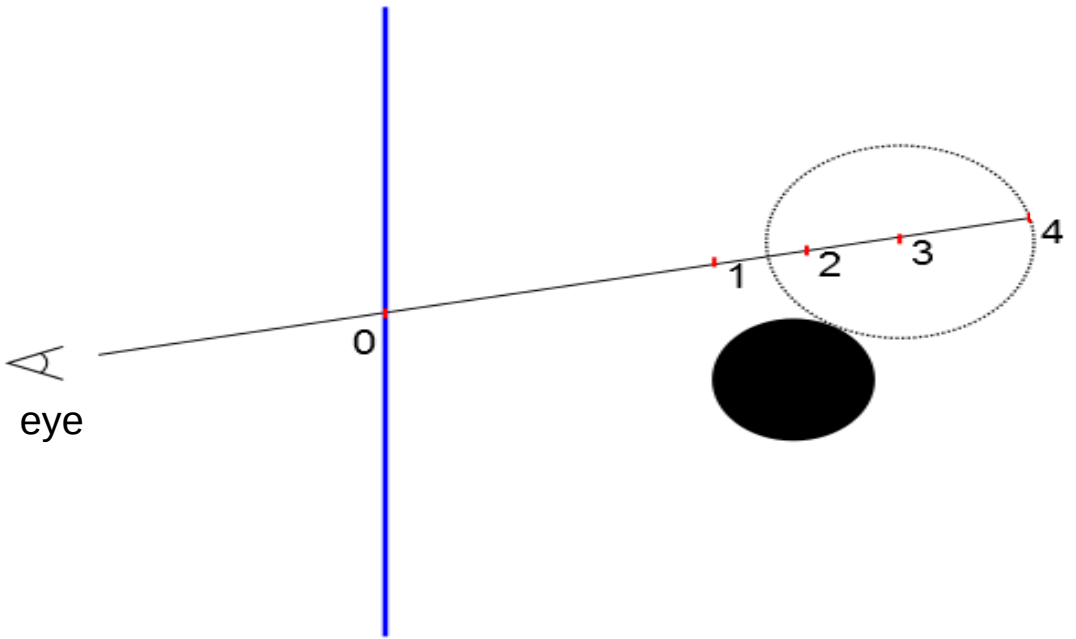
Sphere Tracing



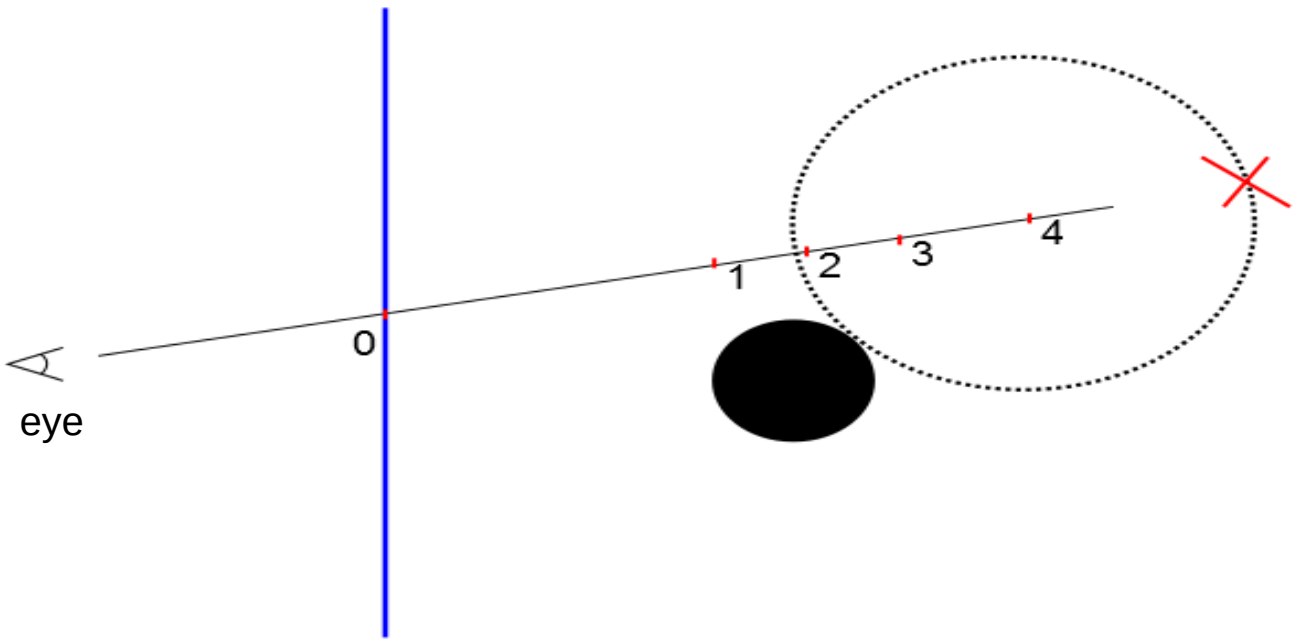
Sphere Tracing



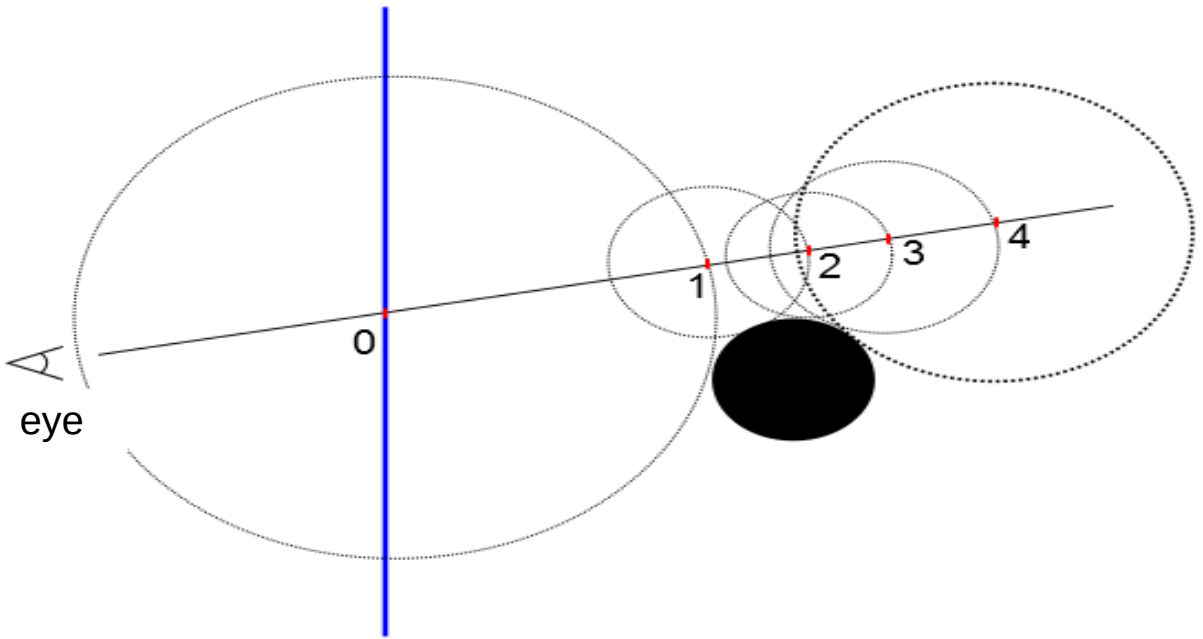
Sphere Tracing



Sphere Tracing



Sphere Tracing

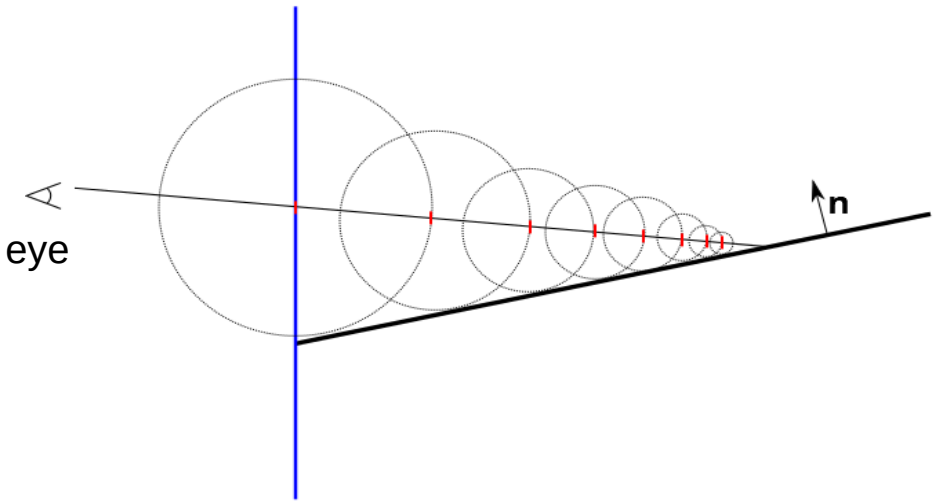


Sphere tracing algorithm

```
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 **n**) going through point **n**d

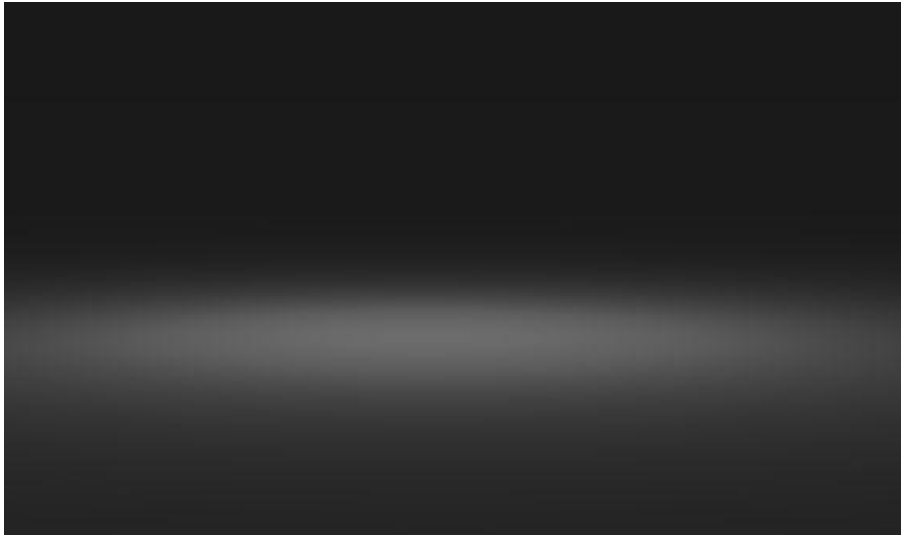


$$f(\mathbf{r}) = \mathbf{r} * \mathbf{n} - d$$

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

Distance Functions

Surface (normal **n**) going through point **n**d



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

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

Distance Functions

Ball with radius R

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

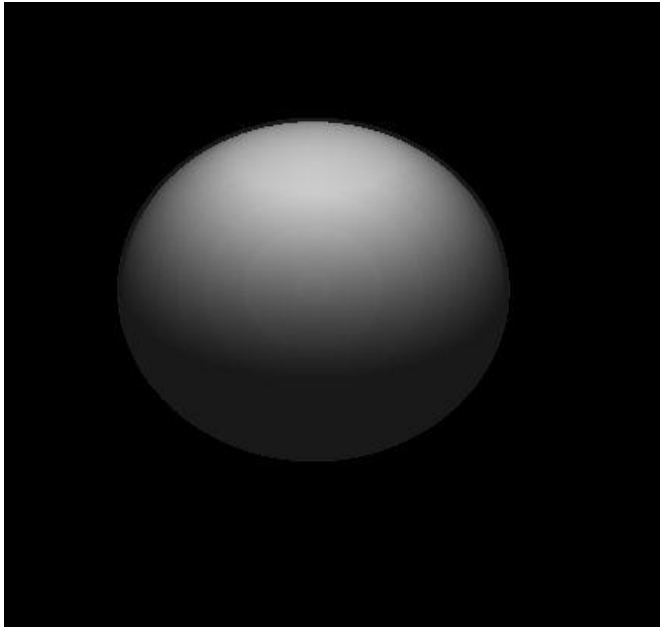
```
// code
float sphere(vec3 r, float R)
{
	return length(r) - R;
}
```

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

Distance Functions

Ball with radius R

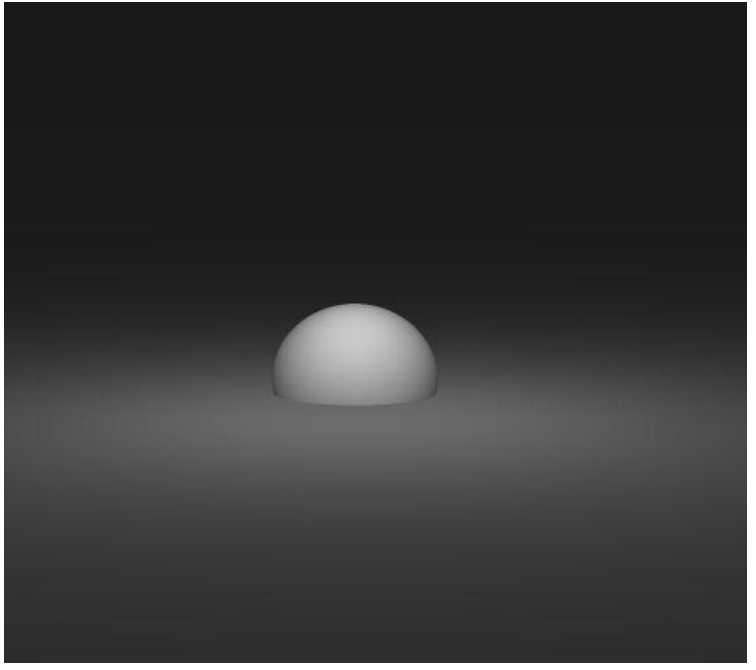
$$f(\mathbf{r}) = |\mathbf{r}| - R$$



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

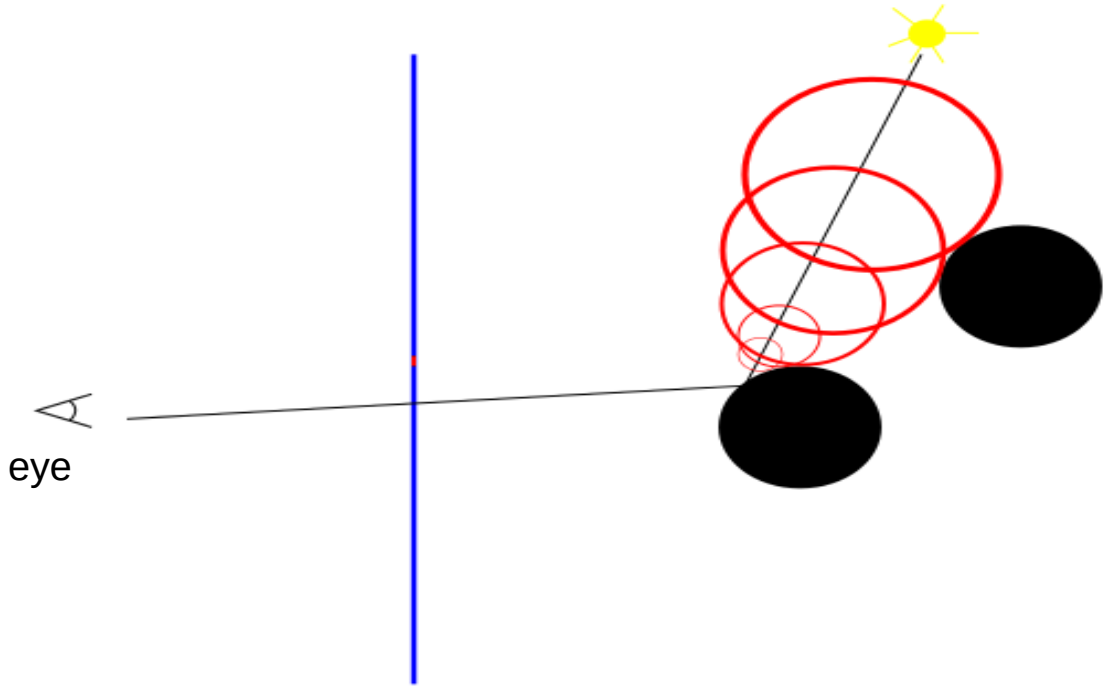
Marching from contact point to light source

+

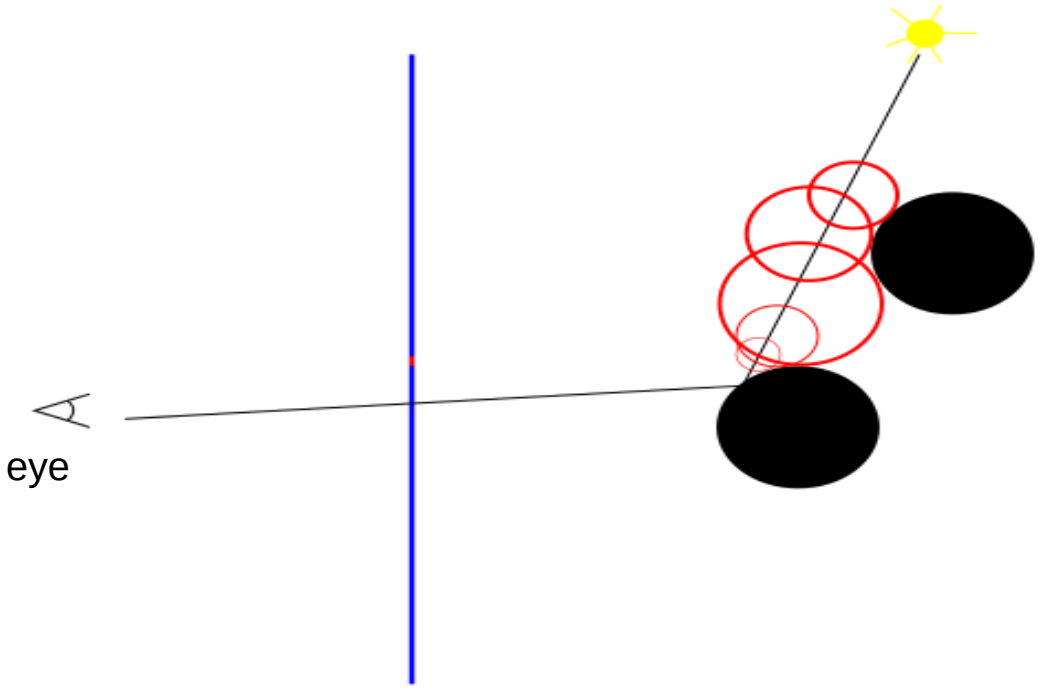
sum distances from the scene

(easy implementation)

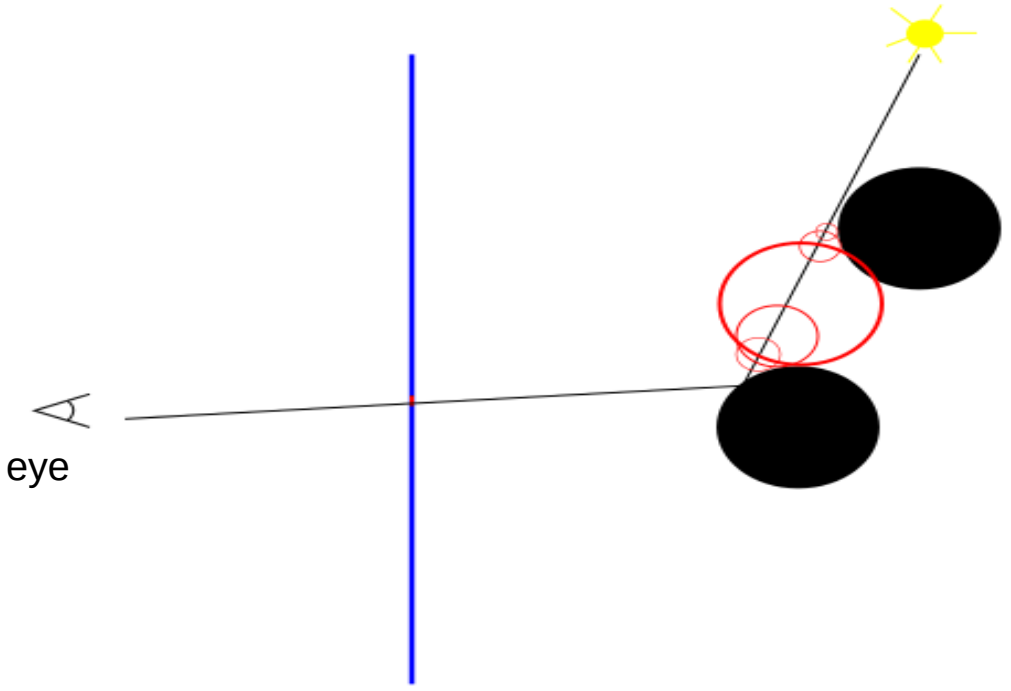
Soft Shadows



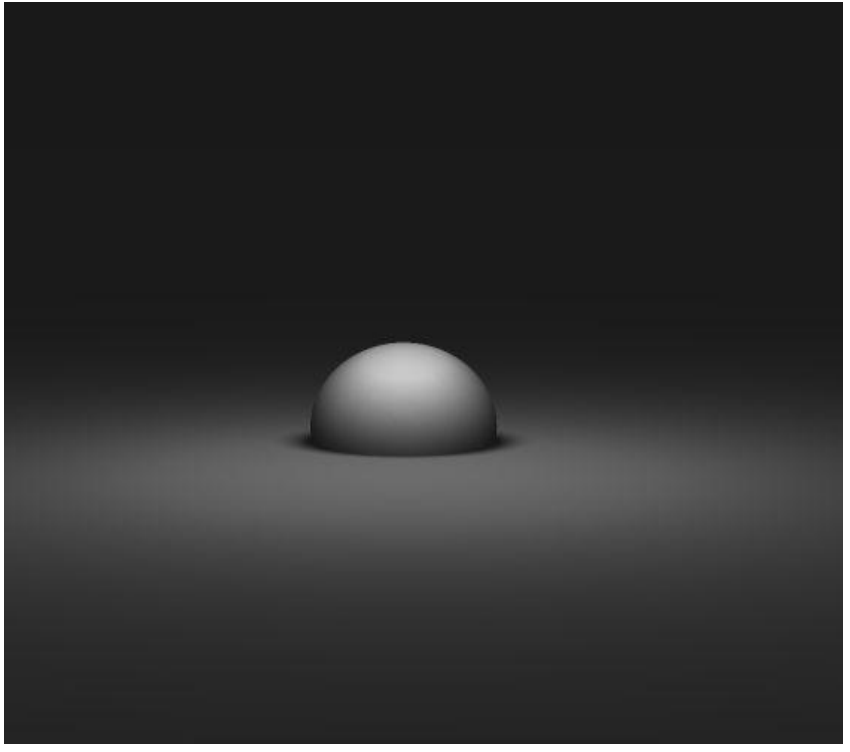
Soft Shadows



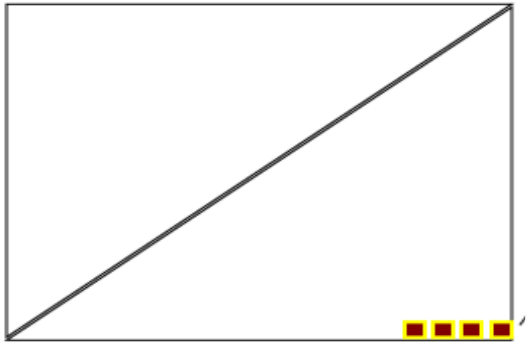
Soft Shadows

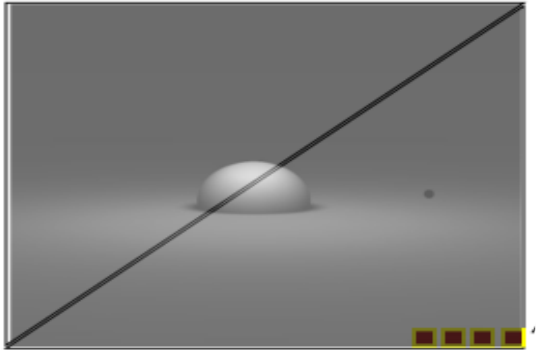


Soft Shadows



RM in Pixel shaders





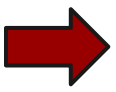
1) two triangles (fullscreen)

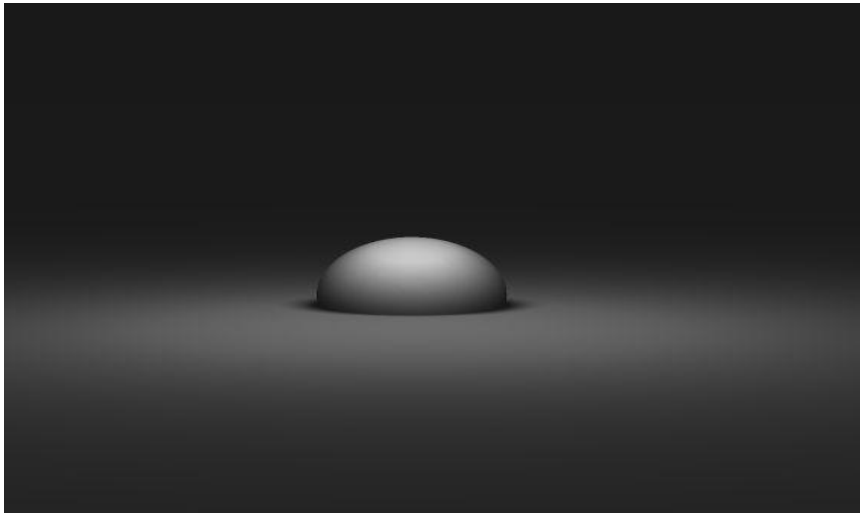
2) each pixel runs the same pixel shader

3) sphere tracing in GLSL pixel shader

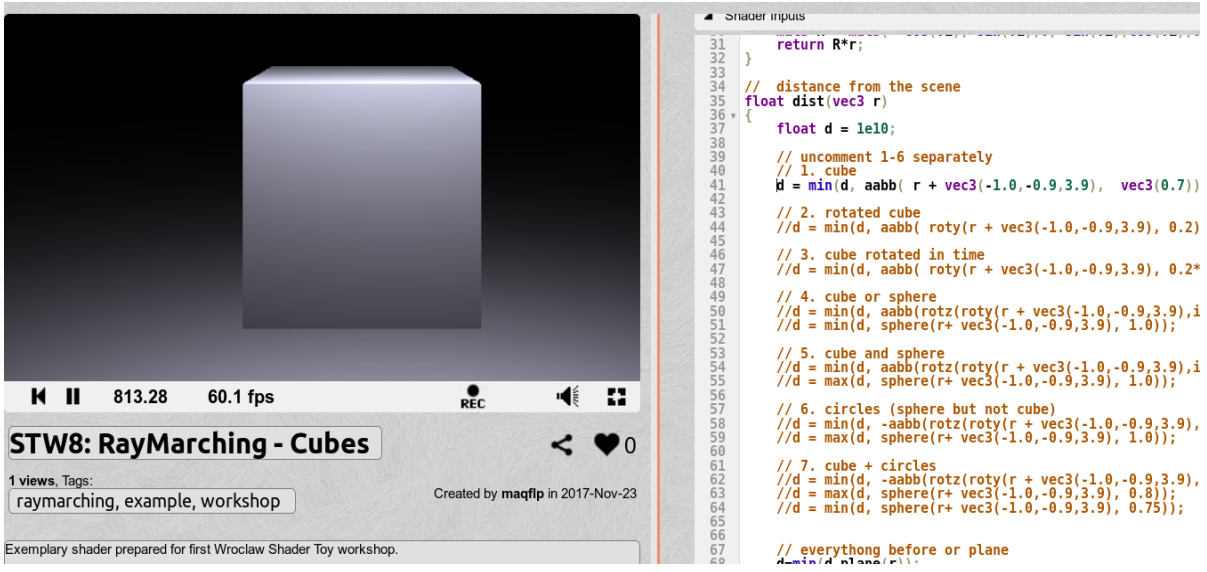
Kod

```
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;
}
```





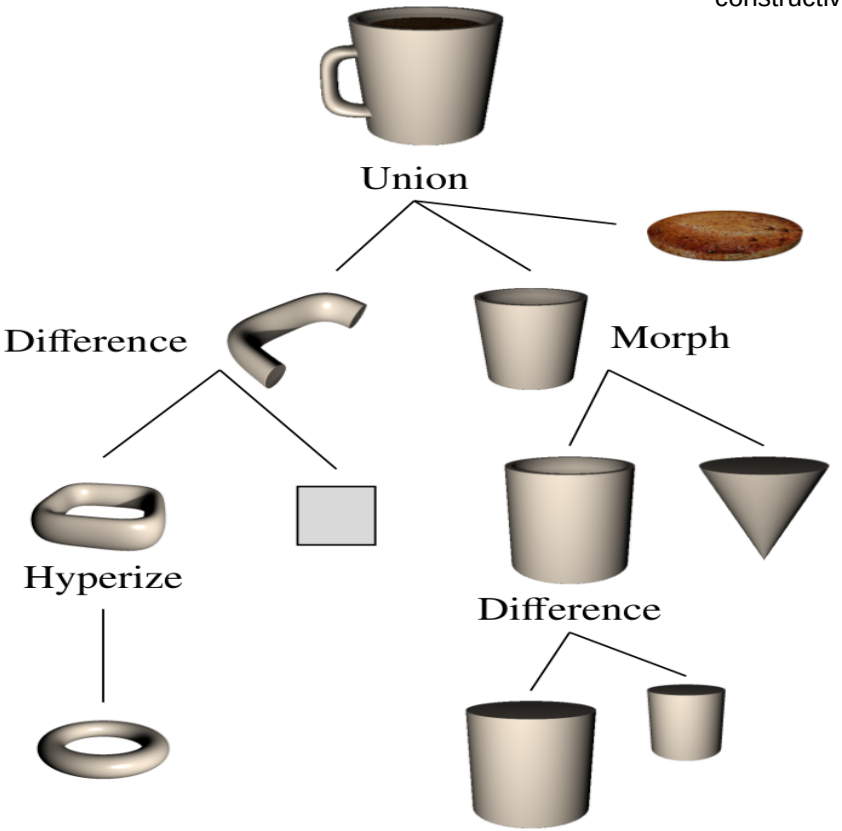
Example



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