Insper

Computação Gráfica

Aula 21: SDF (Signed Distance Function)

Signed Distance Function (SDF)

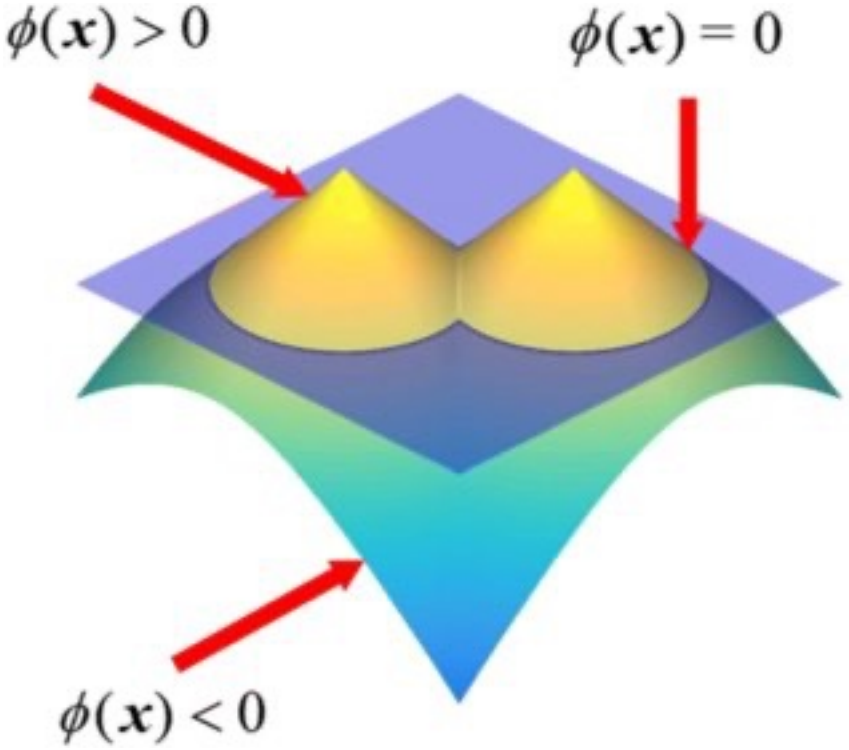
Em matemática, a Função de Distância com Sinal (Signed Distance Function) é a distância ortogonal de um determinado ponto x ao limite de um conjunto Ω em um espaço métrico, com o sinal determinado por x estar ou não no interior de Ω.

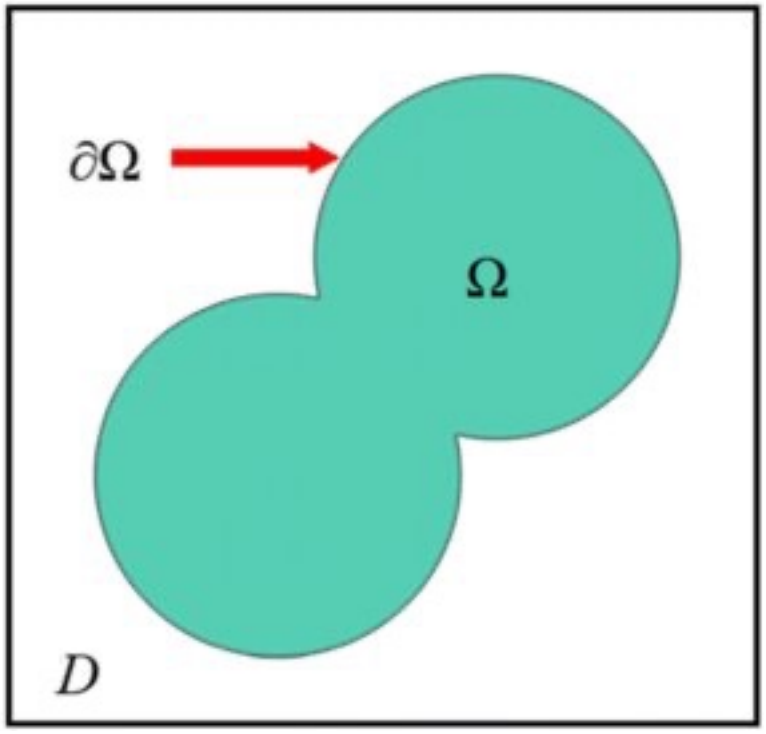
A função tem valores positivos nos pontos x dentro de Ω, diminui em valor quando x se aproxima do limite de Ω onde a função de distância com sinal é zero e assume valores negativos fora de Ω. No entanto, a convenção alternativa às vezes também é adotada (isto é, negativo dentro de Ω e positivo fora).

Adaptado de: https://en.wikipedia.org/wiki/Signed_distance_function

Signed Distance Function (SDF)

Explicar...



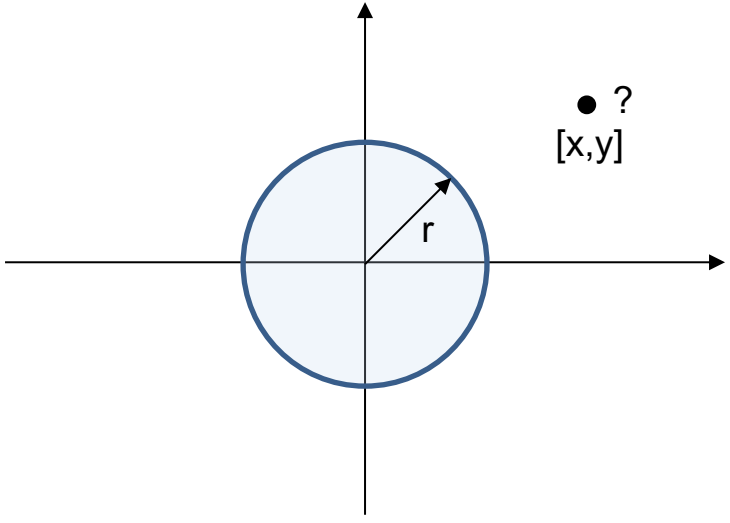


Stress-based shape and topology optimization with cellular level set in B-splines, por :Yelin Song & Qingping Ma & Yu He & Mingdong Zhou & Michael Yu Wang

Função para círculo 2D

Imagine que estou testando um ponto *uv*.

Como saber se este ponto está dentro de um círculo de raio r?



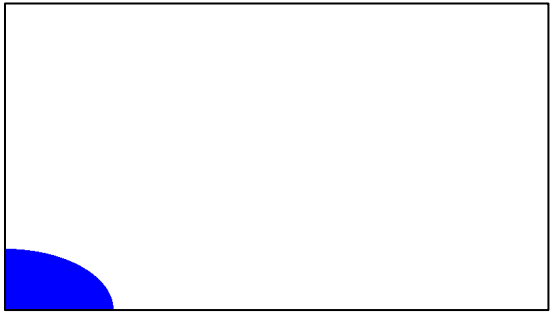

```
float sdfCircle(vec2 uv, float r) {
    float d = length(uv) - r;
    return d;
}
```

Signed Distance Function (SDF)

Exemplo em GLSL para Shadertoy

```
vec3 sdfCircle(vec2 uv, float r) {
    float d = length(uv) - r;
    return d > 0. ? vec3(1.) : vec3(0., 0., 1.);
}

void mainImage( out vec4 fragColor, in vec2 fragCoord )
{
    vec2 uv = fragCoord/iResolution.xy;
    vec3 col = sdfCircle(uv, .2);
    fragColor = vec4(col,1.0);
}
```

O que acontece?



Referência: https://inspirnathan.com/posts/47-shadertoy-tutorial-part-1/

Signed Distance Function (SDF)

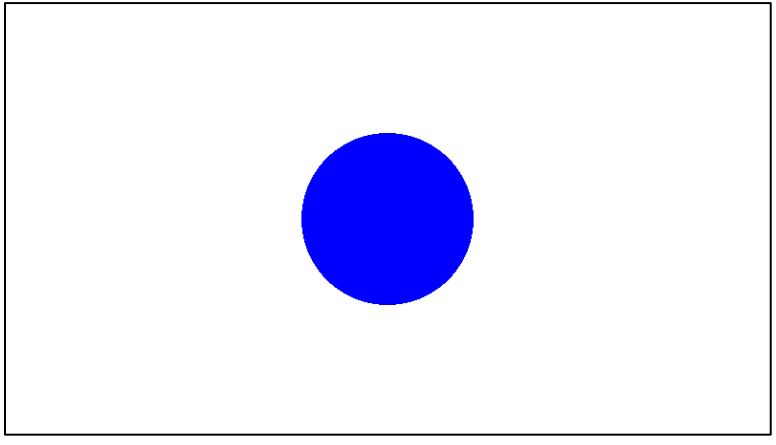
Como melhorar?

Centralizar:

```
uv -= 0.5;
```

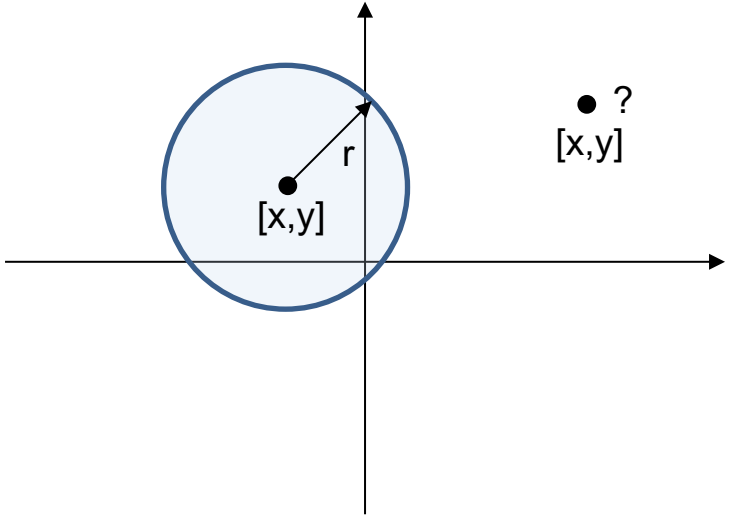
Acertar a razão de aspecto:

```
uv.x *= iResolution.x/iResolution.y;
```



Círculo em outras posições

Como podemos fazer o círculo aparecer em outra posição?

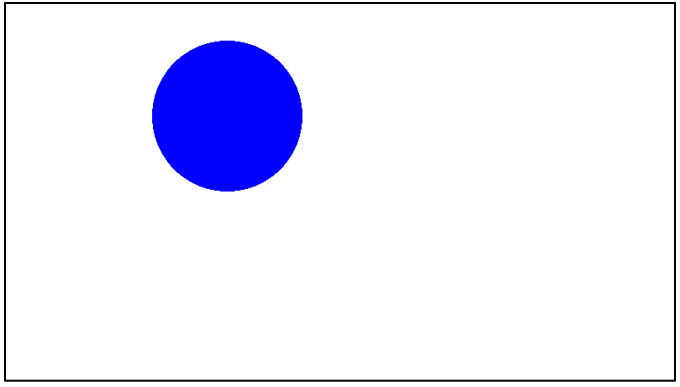



```
float sdfCircle(vec2 uv, float r, vec3 c) {
    float d = length(uv - c) - r;
    return d;
}
```

Exemplo

Um exemplo com o seguinte ponto:

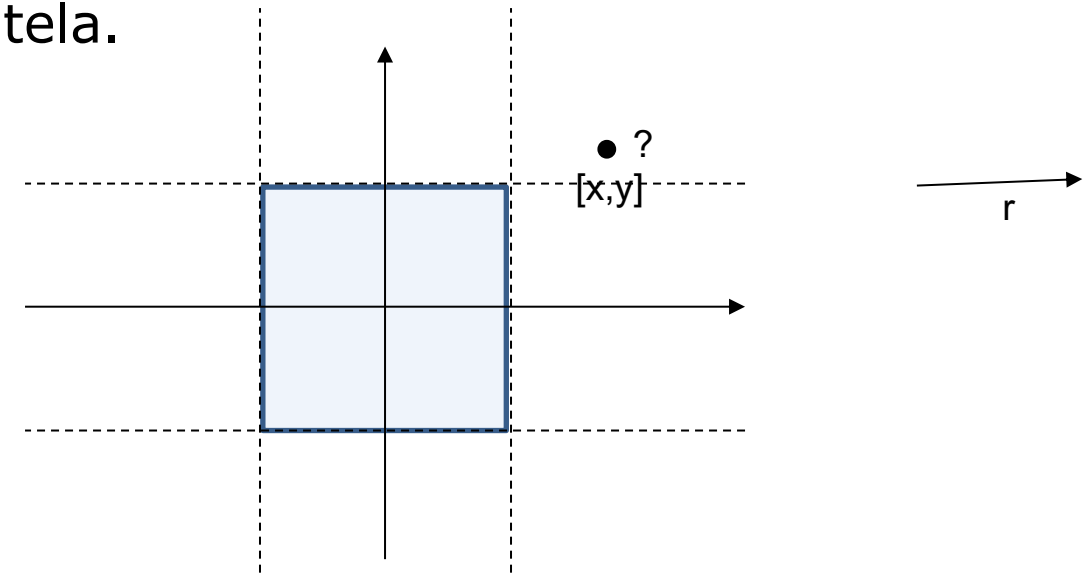
```
vec3 sdfCircle(vec2 uv, float r, vec2 c) {
    float d = length(uv - c) - r;
    return d > 0. ? vec3(1.) : vec3(0., 0., 1.);
}

void mainImage( out vec4 fragColor, in vec2 fragCoord ) {
    vec2 uv = fragCoord/iResolution.xy;
    uv -= 0.5;
    uv.x *= iResolution.x/iResolution.y;
    vec3 col = sdfCircle(uv, .2, vec2(-0.3, 0.2));
    fragColor = vec4(col,1.0);
}
```



Referência: https://inspirnathan.com/posts/47-shadertoy-tutorial-part-1/

Atividade em Aula: Faça um quadrado

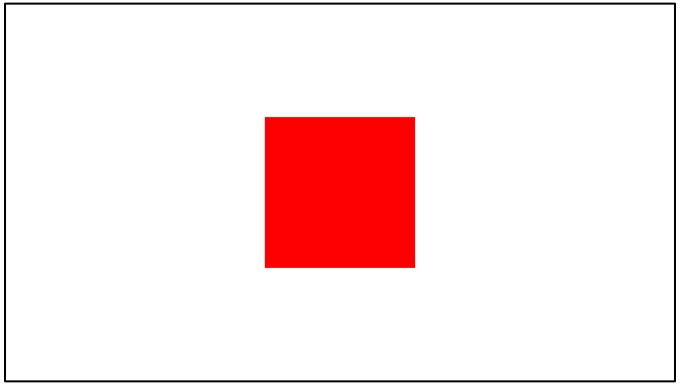
Usando os conceitos aprendidos de SDF. Desenhe um quadrado na tela.




```
bool sdfSquare(vec2 uv, float size, vec2 c) {
    float x = uv.x - c.x;
    float y = uv.y - c.y;
    float d = max(abs(x), abs(y)) - size;
    return d;
}
```

Exemplo Completo

```
vec3 sdfSquare(vec2 uv, float size, vec2 c) {
    float x = uv.x - c.x;
    float y = uv.y - c.y;
    float d = max(abs(x), abs(y)) - size;
    return d > 0. ? vec3(1.) : vec3(1., 0., 0.);
}


void mainImage( out vec4 fragColor, in vec2 fragCoord ) {
    vec2 uv = fragCoord/iResolution.xy;
    uv -= 0.5;
    uv.x *= iResolution.x/iResolution.y;
    vec3 col = sdfSquare(uv, 0.2, vec2(0.0, 0.0));
    fragColor = vec4(col,1.0);
}
```



Referência: https://inspirnathan.com/posts/47-shadertoy-tutorial-part-1/

Transformando objetos (Rotação)

Podemos aplicar a matriz de rotação em um objeto.

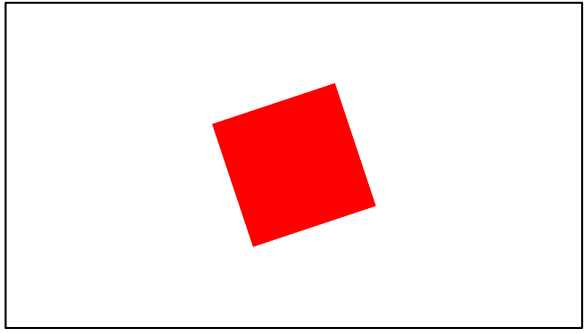
$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Cuidado que as matrizes no GLSL são *column-first*

```
vec2 rotate(vec2 uv, float th) {
    return mat2(cos(th), sin(th), -sin(th), cos(th)) * uv;
}
```

Exemplo com rotação animada

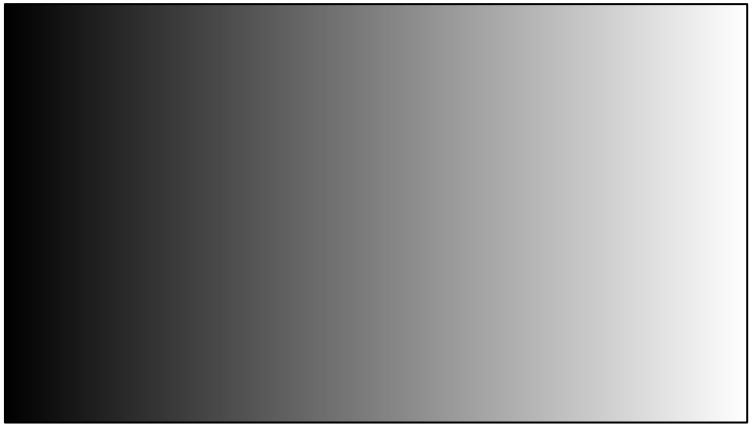
```
vec2 rotate(vec2 uv, float th) {
    return mat2(cos(th), sin(th), -sin(th), cos(th)) * uv;
}

vec3 sdfSquare(vec2 uv, float size, vec2 c) {
    float x = uv.x - c.x;
    float y = uv.y - c.y;
    vec2 rotated = rotate(vec2(x,y), iTime);
    float d = max(abs(rotated.x), abs(rotated.y)) - size;
    return d > 0. ? vec3(1.) : vec3(1., 0., 0.);
}

void mainImage( out vec4 fragColor, in vec2 fragCoord ) {
    vec2 uv = fragCoord/iResolution.xy;
    uv -= 0.5;
    uv.x *= iResolution.x/iResolution.y;
    vec3 col = sdfSquare(uv, 0.2, vec2(0.0, 0.0));
    fragColor = vec4(col,1.0);
}
```



Referência: https://inspirnathan.com/posts/47-shadertoy-tutorial-part-1/

Exemplo mix (LERP)

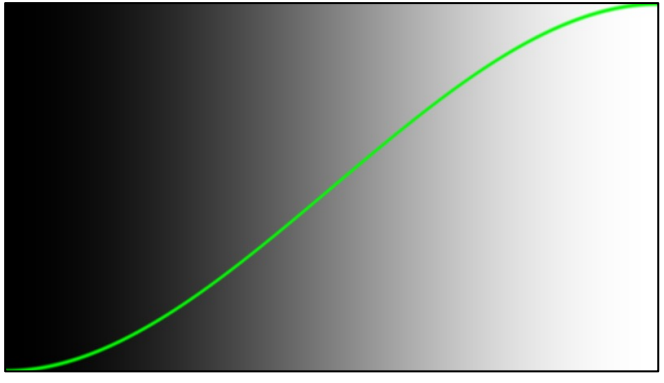
```
void mainImage( out vec4 fragColor, in vec2 fragCoord )
{
    vec2 uv = fragCoord/iResolution.xy;

    float interpolatedValue = mix(0., 1., uv.x);
    vec3 col = vec3(interpolatedValue);

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



Referência: https://inspirnathan.com/posts/47-shadertoy-tutorial-part-1/

Exemplo smoothstep (Hermite)

```
#define diameter 0.01

float smooth_step( float edge0, float edge1, float x ) {
    float p = clamp((x - edge0) / (edge1 - edge0), 0.0, 1.0);
    //float v = p * p * (3.0 - 2.0 * p); // smoothstep formula.
    float v = smoothstep( edge0, edge1, x ); // built-in
    return v;
}

float plot(vec2 st, float y)
{
    return smooth_step( y-diameter, y , st.y) -
    smooth_step( y , y+diameter, st.y);
}

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

    float y = smooth_step( 0.0, 1.0, st.x );
    // grey gradient
    vec3 color = vec3(y);

    // draw smoothstep curve in green
    float percent = plot(st,y);
    color = (1.0-percent)*color + percent*vec3(0.0,1.0,0.0);

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



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

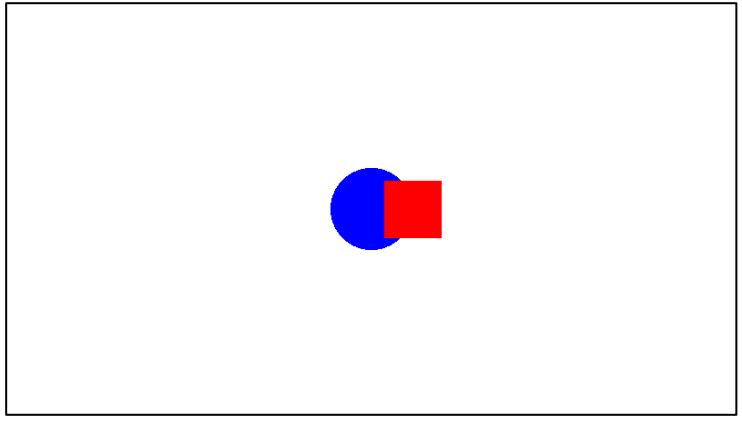
Atividade: Faça um degrade para fundo de tela

Usando os conceitos aprendidos em aula, faça um degrade

```
void mainImage( out vec4 fragColor, in vec2 fragCoord ) {
    vec2 uv = fragCoord/iResolution.xy;
    vec3 gradientStartColor = vec3(0.6, 0.3, 0.0);
    vec3 gradientEndColor = vec3(0.2, 0.8, 1.);
    vec3 col = mix(gradientStartColor, gradientEndColor, uv.y);
    fragColor = vec4(col,1.0);
}
```

Organizando código

```
float sdfCircle(vec2 uv, float r, vec2 c) {
  return length(uv - c) - r;
}

float sdfSquare(vec2 uv, float size, vec2 c) {
  float x = uv.x - c.x;
  float y = uv.y - c.y;
  return max(abs(x), abs(y)) - size;
}

vec3 drawScene(vec2 uv) {
  float circle = sdfCircle(uv, 0.1, vec2(0, 0));
  float square = sdfSquare(uv, 0.07, vec2(0.1, 0));

  vec3 col = vec3(1);
  col = mix(vec3(0, 0, 1), col, step(0., circle));
  col = mix(vec3(1, 0, 0), col, step(0., square));

  return col;
}

void mainImage( out vec4 fragColor, in vec2 fragCoord ) {
  vec2 uv = fragCoord/iResolution.xy;
  uv -= 0.5;
  uv.x *= iResolution.x/iResolution.y;

  vec3 col = drawScene(uv);
  fragColor = vec4(col,1.0);
}
```



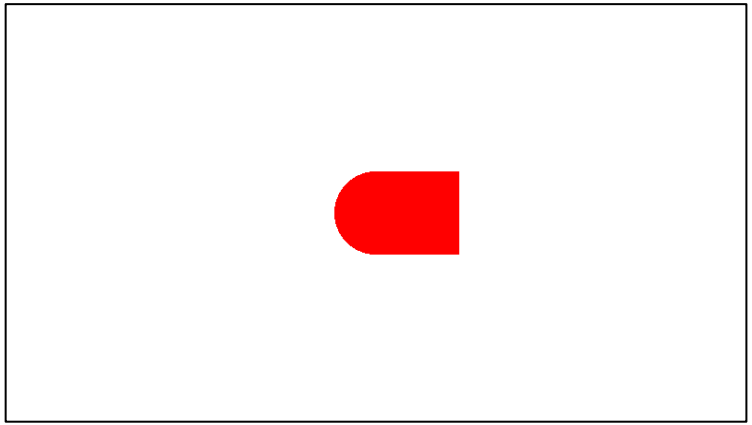
Combinando formas

Um dos truques interessantes do SDF é poder combinar as formas de diversas formas.

Aqui veremos as principais possibilidades.

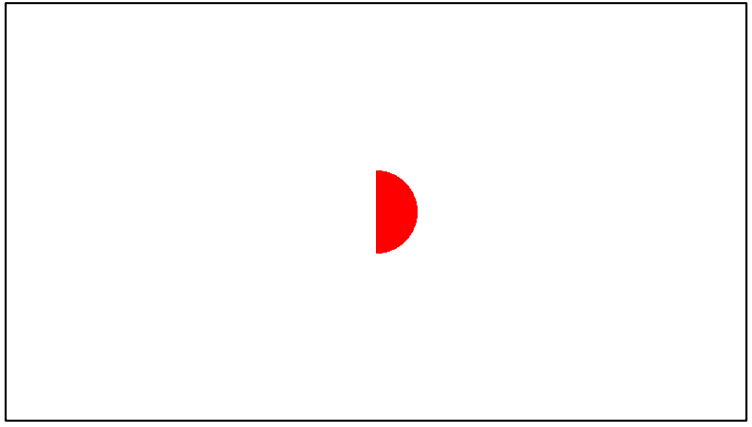
União

```
float sdfCircle(vec2 uv, float r, vec2 c) {
  return length(uv - c) - r;
}

float sdfSquare(vec2 uv, float size, vec2 c) {
  float x = uv.x - c.x;
  float y = uv.y - c.y;
  return max(abs(x), abs(y)) - size;
}

vec3 drawScene(vec2 uv) {
  float circle = sdfCircle(uv, 0.1, vec2(0, 0));
  float square = sdfSquare(uv, 0.1, vec2(0.1, 0));

  vec3 col = vec3(1);
  float res = min(circle, square);
  col = mix(vec3(1, 0, 0), col, step(0., res));

  return col;
}

void mainImage( out vec4 fragColor, in vec2 fragCoord ) {
  vec2 uv = fragCoord/iResolution.xy;
  uv -= 0.5;
  uv.x *= iResolution.x/iResolution.y;

  vec3 col = drawScene(uv);
  fragColor = vec4(col,1.0);
}
```



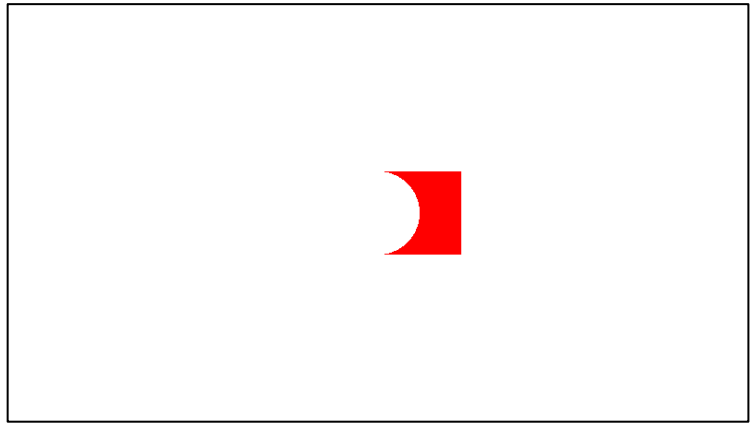
Intersecção

```
float sdfCircle(vec2 uv, float r, vec2 c) {
  return length(uv - c) - r;
}

float sdfSquare(vec2 uv, float size, vec2 c) {
  float x = uv.x - c.x;
  float y = uv.y - c.y;
  return max(abs(x), abs(y)) - size;
}

vec3 drawScene(vec2 uv) {
  float circle = sdfCircle(uv, 0.1, vec2(0, 0));
  float square = sdfSquare(uv, 0.1, vec2(0.1, 0));

  vec3 col = vec3(1);
  float res = max(circle, square);
  col = mix(vec3(1, 0, 0), col, step(0., res));

  return col;
}

void mainImage( out vec4 fragColor, in vec2 fragCoord ) {
  vec2 uv = fragCoord/iResolution.xy;
  uv -= 0.5;
  uv.x *= iResolution.x/iResolution.y;

  vec3 col = drawScene(uv);
  fragColor = vec4(col,1.0);
}
```



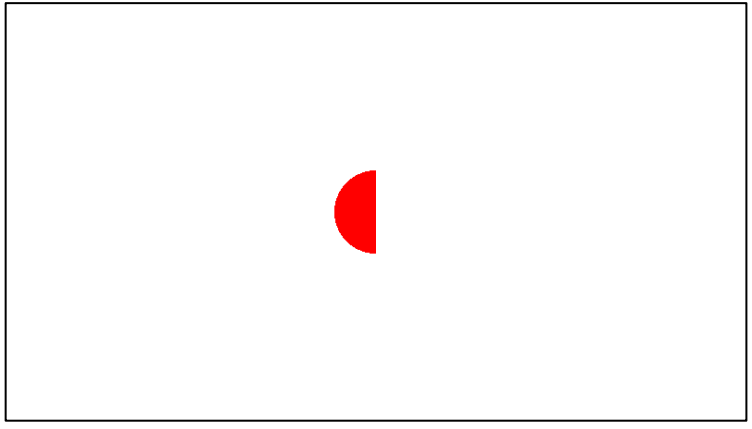
Subtrair o círculo do quadrado

```
float sdfCircle(vec2 uv, float r, vec2 c) {
  return length(uv - c) - r;
}

float sdfSquare(vec2 uv, float size, vec2 c) {
  float x = uv.x - c.x;
  float y = uv.y - c.y;
  return max(abs(x), abs(y)) - size;
}

vec3 drawScene(vec2 uv) {
  float circle = sdfCircle(uv, 0.1, vec2(0, 0));
  float square = sdfSquare(uv, 0.1, vec2(0.1, 0));

  vec3 col = vec3(1);
  float res = max(-circle, square);
  col = mix(vec3(1, 0, 0), col, step(0., res));

  return col;
}

void mainImage( out vec4 fragColor, in vec2 fragCoord ) {
  vec2 uv = fragCoord/iResolution.xy;
  uv -= 0.5;
  uv.x *= iResolution.x/iResolution.y;

  vec3 col = drawScene(uv);
  fragColor = vec4(col,1.0);
}
```



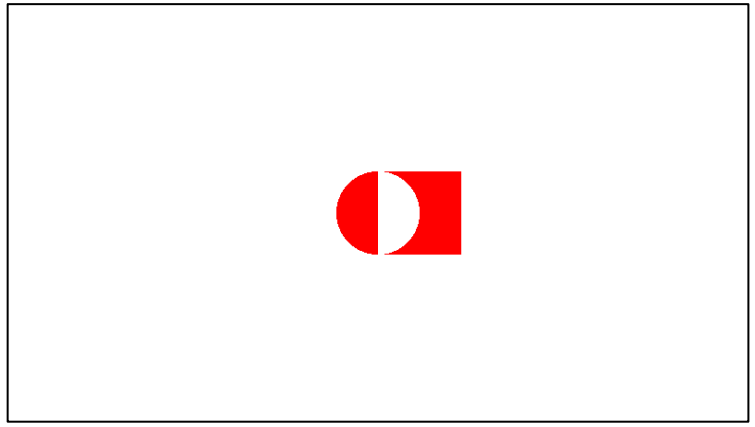
Subtrair o quadrado do círculo

```
float sdfCircle(vec2 uv, float r, vec2 c) {
  return length(uv - c) - r;
}

float sdfSquare(vec2 uv, float size, vec2 c) {
  float x = uv.x - c.x;
  float y = uv.y - c.y;
  return max(abs(x), abs(y)) - size;
}

vec3 drawScene(vec2 uv) {
  float circle = sdfCircle(uv, 0.1, vec2(0, 0));
  float square = sdfSquare(uv, 0.1, vec2(0.1, 0));

  vec3 col = vec3(1);
  float res = max(circle, -square);
  col = mix(vec3(1, 0, 0), col, step(0., res));

  return col;
}

void mainImage( out vec4 fragColor, in vec2 fragCoord ) {
  vec2 uv = fragCoord/iResolution.xy;
  uv -= 0.5;
  uv.x *= iResolution.x/iResolution.y;

  vec3 col = drawScene(uv);
  fragColor = vec4(col,1.0);
}
```



Ou exclusivo (XOR)

```
float sdfCircle(vec2 uv, float r, vec2 c) {
  return length(uv - c) - r;
}

float sdfSquare(vec2 uv, float size, vec2 c) {
  float x = uv.x - c.x;
  float y = uv.y - c.y;
  return max(abs(x), abs(y)) - size;
}

vec3 drawScene(vec2 uv) {
  float circle = sdfCircle(uv, 0.1, vec2(0, 0));
  float square = sdfSquare(uv, 0.1, vec2(0.1, 0));

  vec3 col = vec3(1);
  float res = max(min(circle, square), -max(circle, square));
  col = mix(vec3(1, 0, 0), col, step(0., res));

  return col;
}

void mainImage( out vec4 fragColor, in vec2 fragCoord ) {
  vec2 uv = fragCoord/iResolution.xy;
  uv -= 0.5;
  uv.x *= iResolution.x/iResolution.y;

  vec3 col = drawScene(uv);
  fragColor = vec4(col,1.0);
}
```



Resumindo

```
res = min(d1, d2); // união

res = max(d1, d2); // intersecção

res = max(-d1, d2); // subtração - d1 menos d2

res = max(d1, -d2); // subtração - d2 menos d1

res = max(min(d1, d2), -max(d1, d2)); // xor
```

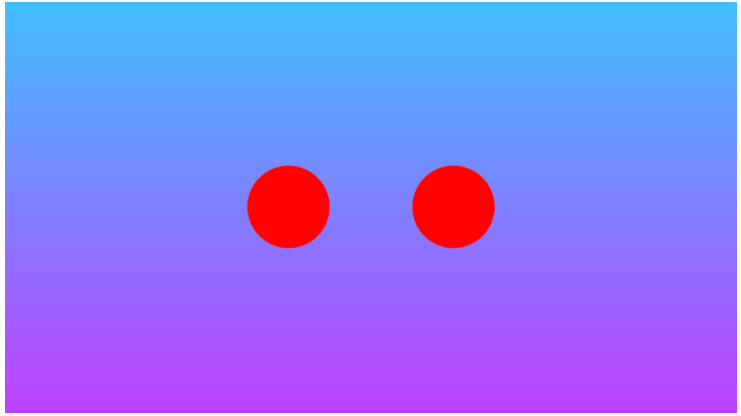
Posicionamento 2D

Inspirado originalmente no trabalho de Inigo Quilez. A seguir serão apresentadas algumas estratégias de posicionar e exibir padrões de imagens.

opSymX

Repete o padrão na horizontal.

```
float opSymX(vec2 p, float r)
{
  p.x = abs(p.x);
  return sdCircle(p, r, vec2(0.2, 0));
}

vec3 drawScene(vec2 uv) {
  vec3 col = getBackgroundColor(uv);

  float res; // result
  res = opSymX(uv, 0.1);

  res = step(0., res);
  col = mix(vec3(1,0,0), col, res);
  return col;
}
```

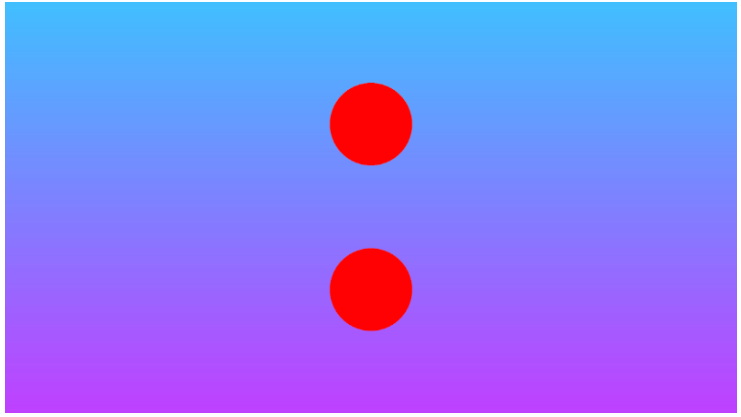


Referência: https://inspirnathan.com/posts/47-shadertoy-tutorial-part-1/

opSymY

Repete o padrão na horizontal.

```
float opSymY(vec2 p, float r)
{
  p.y = abs(p.y);
  return sdCircle(p, r, vec2(0, 0.2));
}

vec3 drawScene(vec2 uv) {
  vec3 col = getBackgroundColor(uv);

  float res; // result
  res = opSymY(uv, 0.1);

  res = step(0., res);
  col = mix(vec3(1,0,0), col, res);
  return col;
}
```

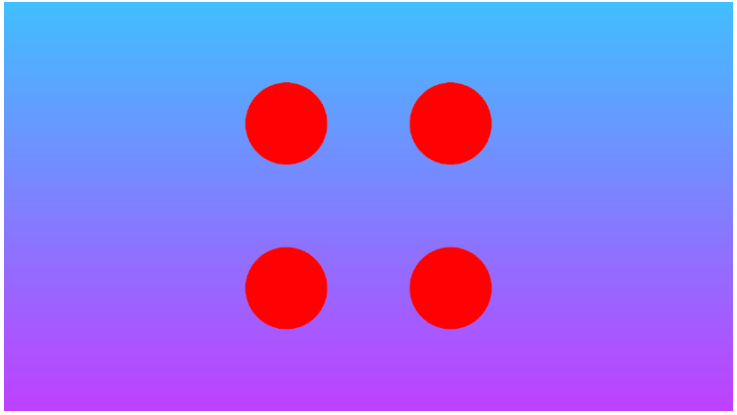


Referência: https://inspirnathan.com/posts/47-shadertoy-tutorial-part-1/

opSymXY

Repete o padrão na horizontal.

```
float opSymXY(vec2 p, float r)
{
  p = abs(p);
  return sdCircle(p, r, vec2(0.2));
}

vec3 drawScene(vec2 uv) {
  vec3 col = getBackgroundColor(uv);

  float res; // result
  res = opSymXY(uv, 0.1);

  res = step(0., res);
  col = mix(vec3(1,0,0), col, res);
  return col;
}
```

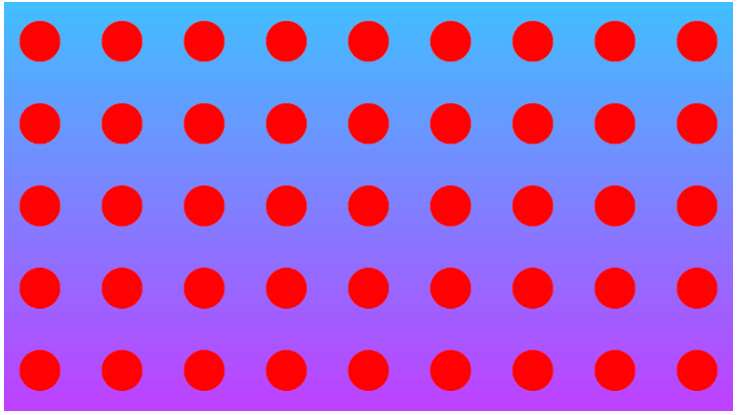


Referência: https://inspirnathan.com/posts/47-shadertoy-tutorial-part-1/

opRep

Repete o padrão na horizontal.

```
float opRep(vec2 p, float r, vec2 c)
{
  vec2 q = mod(p+0.5*c,c)-0.5*c;
  return sdCircle(q, r, vec2(0));
}

vec3 drawScene(vec2 uv) {
  vec3 col = getBackgroundColor(uv);

  float res; // result
  res = opRep(uv, 0.05, vec2(0.2, 0.2));

  res = step(0., res);
  col = mix(vec3(1,0,0), col, res);
  return col;
}
```

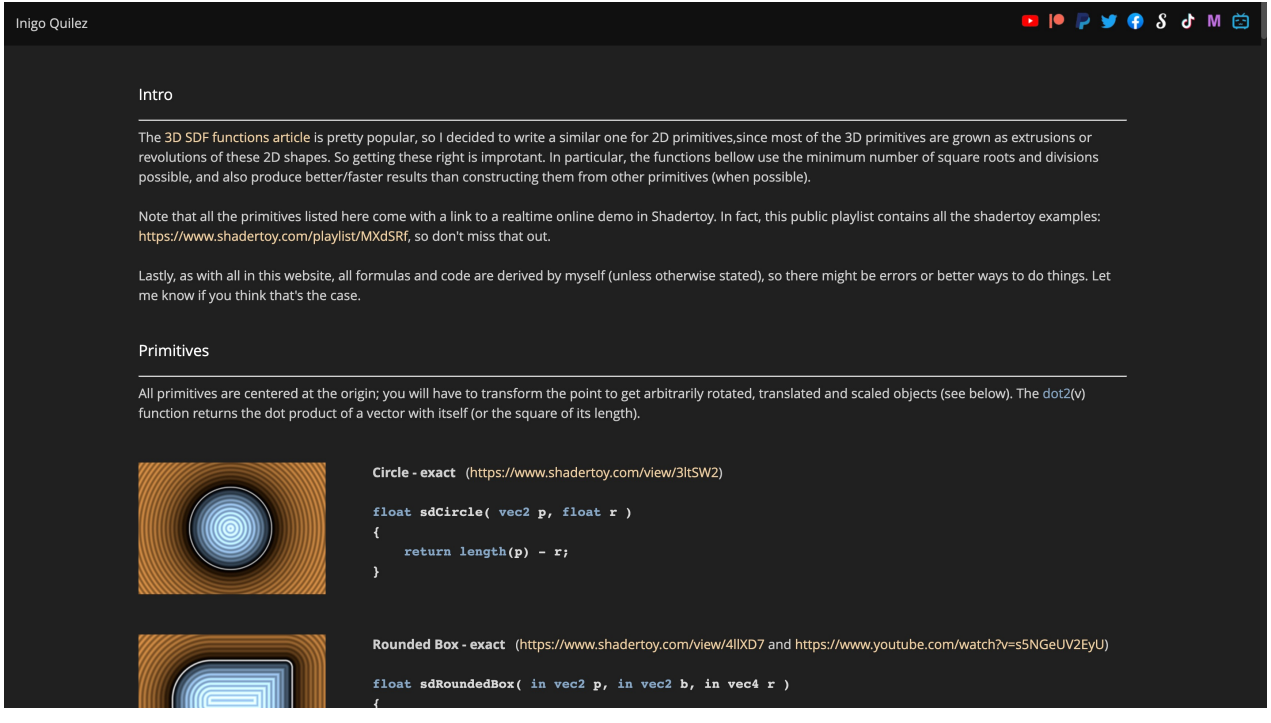


Referência: https://inspirnathan.com/posts/47-shadertoy-tutorial-part-1/

Funções SDF prontas

Muitas funcionalidades para SDF já existem. Um bom repositório é o site do Inigo Quilz:

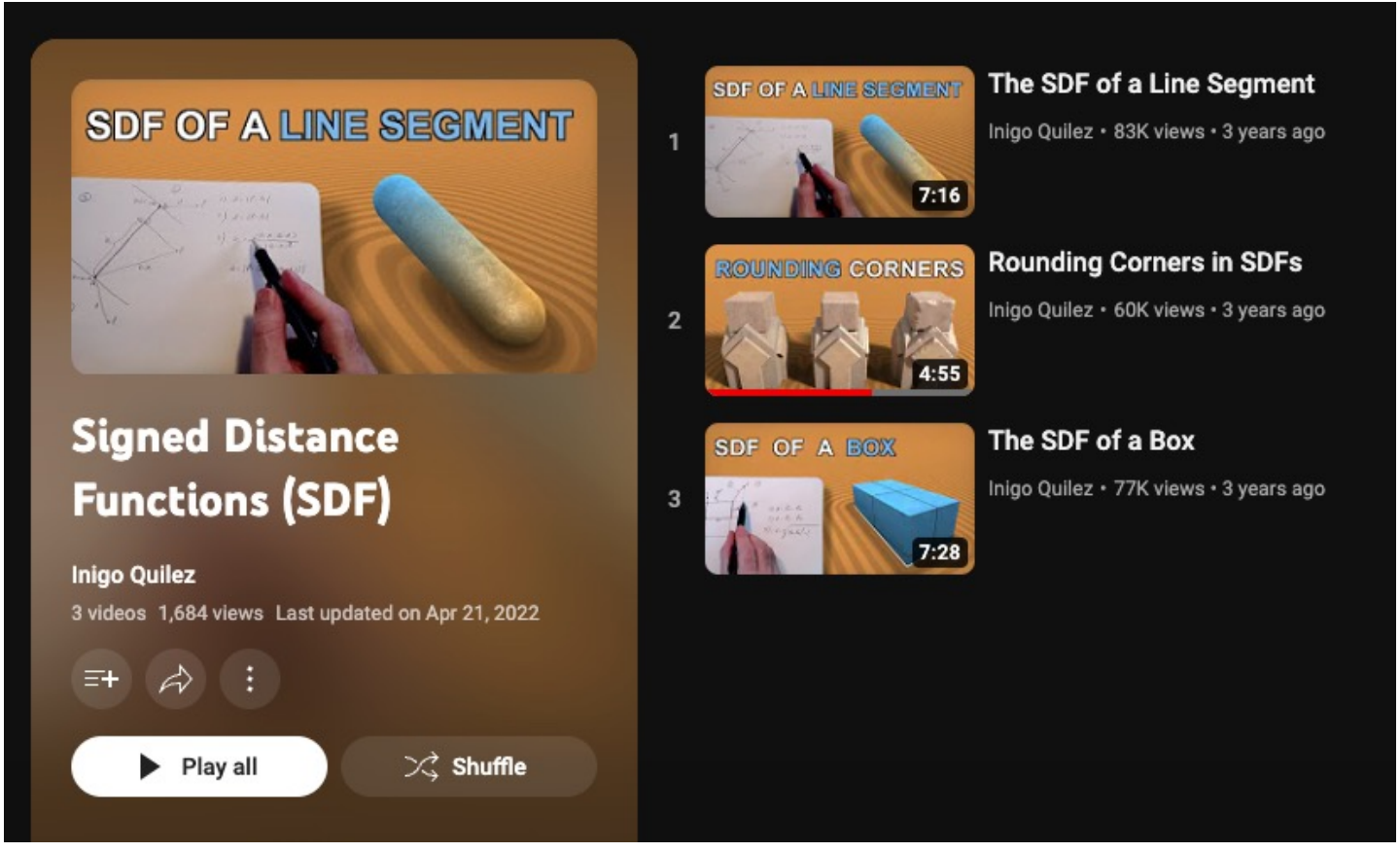
https://iquilezles.org/articles/distfunctions2d/



Insper

Vídeos sobre SDFs

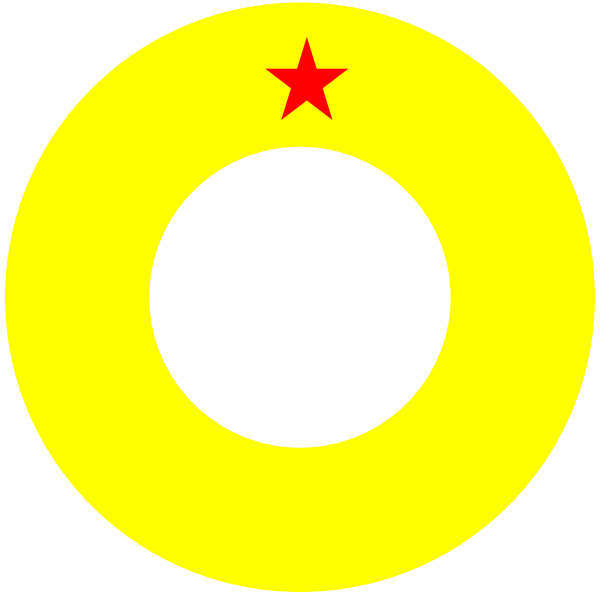
https://www.youtube.com/playlist?list=PL0EpikNmjs2AUFqRi3vmpkrO3j-zWuoyq



Insper

Projeto 2.1

Crie uma animação 2D no Fragment Shader de uma estrela (qualquer tipo) sobre um anel. A volta toda deve demorar 5 segundos. A velocidade na parte superior deve ser zero.



Referências

Baseado:
https://www.shadertoy.com/

Usando:
https://inspirnathan.com/posts/49-shadertoy-tutorial-part-3

Documentações:
https://iquilezles.org/
https://thebookofshaders.com/

Insper

Computação Gráfica

Luciano Soares
<lpsoares@insper.edu.br>