# 1 Signed Distance Fields

Signed Distance Fields (SDFs), very briefly and getting straight to the point, represent a distance from a certain shape. 'Signed' part refers to the fact that when we are **inside** a shape our distance is negative, **outside** a shape distance becomes positive and when we're on the border of that shape distance is equal to zero.

Let's first discuss a very simple shape, a circle, where every point has exactly the same distance from its center. Now let's think about what does it mean to have 'positive' and 'negative' distance based on our previous definitions.

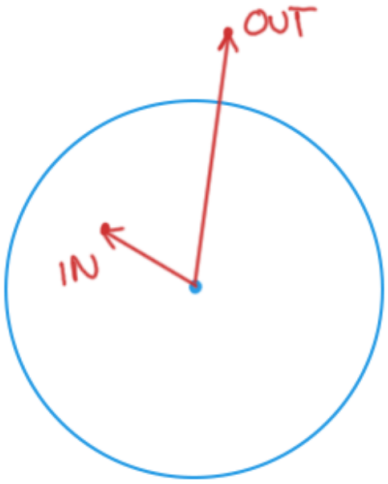


Figure 1: Determining SDF of a circle

If we're starting from the center it's rather easy to determine the distance from that center to some other point, we just subtract two vectors and check the resulting vector's length like so:

$$||\vec{P} - \vec{C}|| = d \tag{1}$$

Where $C$ is the center of our circle, $P$ is some point we want to check and result $d$ is our calculated distance from $C$ to $P$. This is all nice, but the problem is that way our distance will always be positive. What can we do to ensure that when we're **inside** the circle our distance becomes negative, otherwise its positive?

Fortunately with such a simple shape like circle we don't really have to spend a lot of time figuring it out. Because our circle has a certain radius, we can just subtract that from our distance $d$, that way we'll get something negative when our $d$ is less than radius (i.e. point is inside the circle)

$$||\vec{P} - \vec{C}|| - r = SDF \tag{2}$$

With that it's not so hard to imagine a simple check for when we're inside of a circle:

$$||\vec{P} - \vec{C}|| - r \leq 0 \tag{3}$$

This was a brief introduction to SDFs and how to think of them, of course with more complex shapes the equations for SDF also become much more involved. There are, however, plenty of 'out of the box' and ready-to-use functions for plenty of shapes on the internet.

## 2 Rounded Rectangle

While you could draw a rounded rectangle with just four circles and regular rectangle (and in most cases that's probably all you need). It's important to know other approaches, like SDFs (Signed Distance Fields) which I want to, very briefly, discuss here.

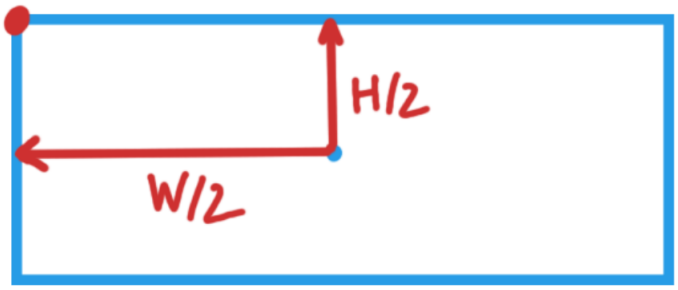


Figure 2: Determining SDF of a rectangle

Following the same reasoning from our previous section - How do we determine if our point is **inside** or **outside** a rectangle? Again, we can use the center to easily determine the distance of a point from that center:

$$||\vec{P} - \vec{C}|| = d \tag{4}$$

Where $C$ is the center of our rectangle (usually just half of width and height) $P$ is the point we're checking and resulting $d$ is the distance. This, again, gives us a distance that's always positive, this part is probably the most difficult to graps and understand for beginners (which also includes me). Let's first expand our distance formula to see what's happening:

$$||\vec{P} - \vec{C}|| = \sqrt{(P_x - C_x)^2 + (P_y - C_y)^2} = d \tag{5}$$

It may help to draw it out yourself, that way you can understand it more. What this is, well, it's a simple equation for distance! However in this particular context it tells us so much more, for example, what happens if one of those terms is equal to 0? Well that just means that our point is perfectly up or perfectly to the right. Also when one of our terms, let's say $P_x - C_x$ becomes negative, that means our point is somewhere inside the rectangle.

We can devide our distance in two cases then (outside and inside) and join them together. First part is to determine whether a point is inside or outside of a rectangle, to achieve that we can simply do the following:

$$\sqrt{(max(P_x - C_x, 0))^2 + (max(P_y - C_y, 0))^2} = d_o \tag{6}$$

That way each term is something either bigger than or equal to 0, this very beautifully describes distance on the outside of our rectangle. This is not the end just yet, now we need to determine the distance on the inside of the rectangle. You can imagine that it will be something similar, however our value is either less than or equal 0 now because we're on the inside (recall SDF definition):

$$min(max(P_x - C_x, P_y - C_y), 0) = d_i \tag{7}$$

If our expression $P_x - C_x$ becomes bigger than 0 that means our point is outside the rectangle, we're looking for values smaller than 0. Why are those expression inside $max$ function? Well, we're looking at a rectangle, not a circle, we need to get a bigger distance between its width and height, at least that's how I think about it. Otherwise we would probably be looking at something that resembles a circle.

This is it! That's the entire formula for an SDF of a rectangle/box:

$$d_o + d_i = SDF \tag{8}$$

To get something rounded you just need to 'shrink' your rounded rectangle by corner radius i.e. you just need to add and subtract from two equations here. Because I do realize that 'obvious' is the most dangerous word in mathematics, I'll leave some examples you can play with on 'Shadertoy.com'.

# 3 Code Examples

## 3.1 SDF of a circle

```
void mainImage(out vec4 fragColor, in vec2 fragCoord)
{
    // @Note: [0;1]
    vec2 uv = (fragCoord/iResolution.xy * 2.0) - 1.0;

    // @Note: Fix aspect ratio.
    uv.x *= iResolution.x/iResolution.y;

    // @Note: Preparatory things for circle rendering.
    vec4 colour = vec4(0.0, 0.0, 0.80, 1.0);
    vec2 center = vec2(0.0, 0.0);
    float radius = 0.7;
    float smoothing = 0.006;

    // @Note: SDF
    float dist = length(uv - center) - radius;
    float m = smoothstep(0.0, smoothing, dist);

    fragColor = mix(colour, vec4(1.0, 1.0, 1.0, 1.0), m);
}
```

## 3.2 SDF of a rounded rectangle

```
// @Note: Entire SDF of a rounded rectangle.
float SDF(in vec2 p, in vec2 center, in vec2 half_size, in float r)
{
    vec2 q = abs(p - center) - half_size + r;
    return min(max(q.x,q.y), 0.0) + length(max(q, 0.0)) - r;
}

void mainImage(out vec4 fragColor, in vec2 fragCoord)
{
    // @Note: NDC [-1;1]
    vec2 uv = (fragCoord/iResolution.xy * 2.0) - 1.0;

    // @Note: Preparatory things for rectangle rendering.
    vec2 half_size = vec2(0.8, 0.7);
    vec2 center = vec2(0.0, 0.0);
    vec4 rect_color = vec4(0.36, 0.81, 0.98, 1.0);
    float r = 0.12;
    float edge_softness = 0.02;

    float d = SDF(uv, center, half_size, r);

    // @Note: Something to add pseudo-AA
    float m = smoothstep(0.0, edge_softness, d);

    fragColor = mix(rect_color, vec4(1.0, 1.0, 1.0, 1.0), m);
}
```