Implicit Curve and Surfaces

Prof. Fabio Pellacini
[some slide ideas by Prof. Wojciech Jarosz]

Implicit curves and surfaces

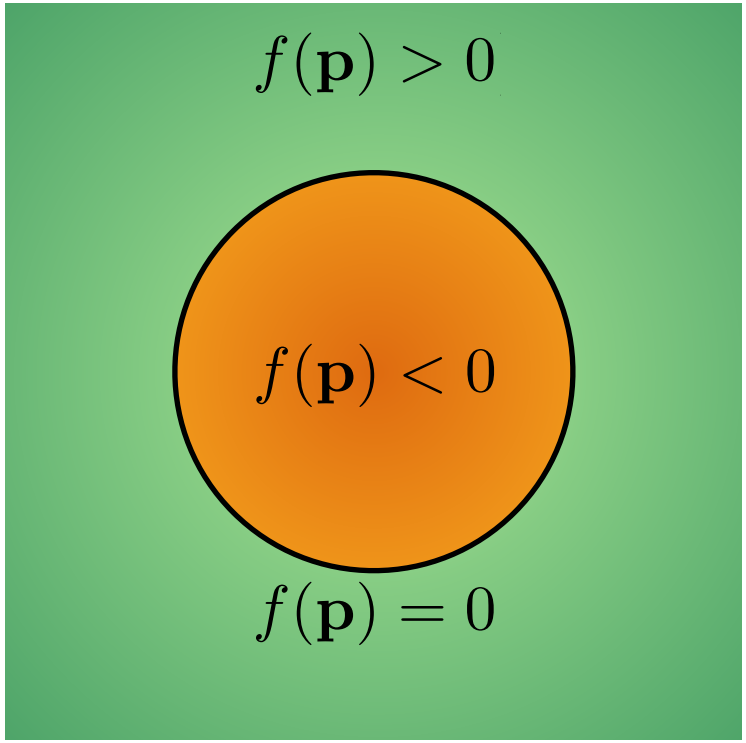
- Parametric models, meshes and subdivs are shape representations that explicitly represent the curve points and surface boundaries
- Implicit curves and surfaces are an alternative shape representation where boundaries are implicitly defined as the isovalues of a function

$$S = \{\mathbf{p} | f(\mathbf{p}) = 0\}$$

Signed Distance Functions

- The most used implicit representation is to consider functions that encode the signed distance to the boundary
  - The sign indicates whether we are outside or inside the surface

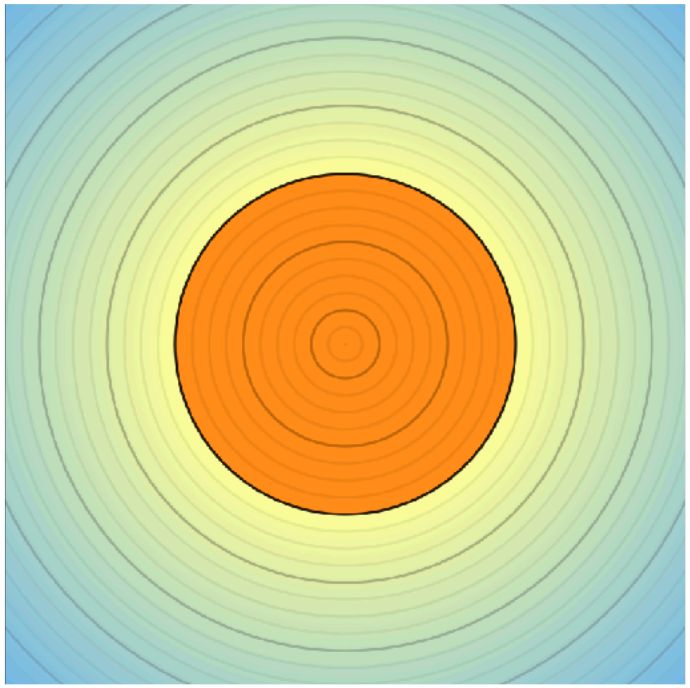
$$O = \{\mathbf{p} | f(\mathbf{p}) > 0\} \quad \text{outside}$$
$$S = \{\mathbf{p} | f(\mathbf{p}) = 0\} \quad \text{curve/surface}$$
$$I = \{\mathbf{p} | f(\mathbf{p}) < 0\} \quad \text{indice}$$



Signed Distance Functions

- Example for circles and spheres

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



Signed Distance Functions

- Normals of SDFs are computed as the gradient of the function

$$\mathbf{n} = \nabla f(\mathbf{p}) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right) (\mathbf{p})$$

- Since the gradient is often not available analytically, we estimate it numerically by finite differences

$$n_x \approx \frac{f(p_x + h) - f(p_x - h)}{2h}$$

$$n_y \approx \frac{f(p_y + h) - f(p_y - h)}{2h}$$

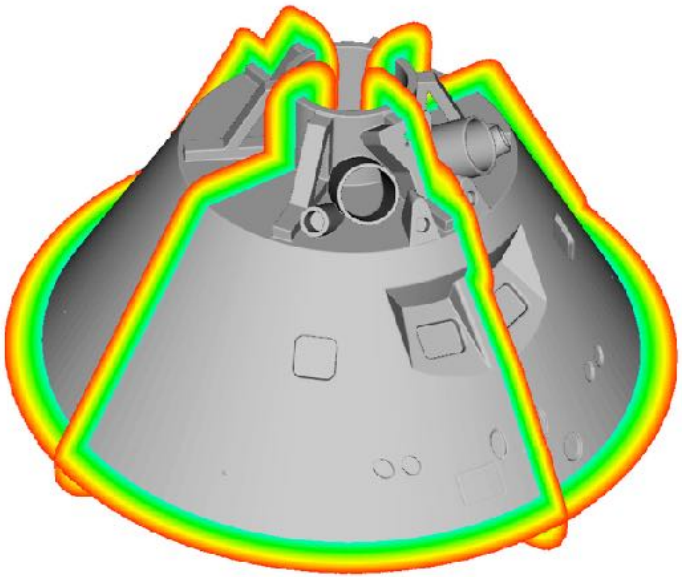
$$n_z \approx \frac{f(p_z + h) - f(p_z - h)}{2h}$$

Signed Distance Functions

- As for many numerical approximation we need to pay attention to numerical precision, in this case for the choice of increment when computing normals

Signed Distance Functions

- SDFs for surfaces are represented in all points in a volume
- To keep memory usage limited, real implementations store SDFs only in a narrow band around the surface using sparse hash grids



[https://al-ro.github.io/projects/sdf/]

Signed Distance Functions

- Pros: easy and robust determination of inside/outside
  - used in collision detection for animation
- Pros: easy to determine if a point is on the surface
  - used in raytracing
- Cons: hard to enumerate points on the surfaces
- Cons: hard to modify directly

- Applications: Great for procedural modeling
- Applications: Very useful in deep learning

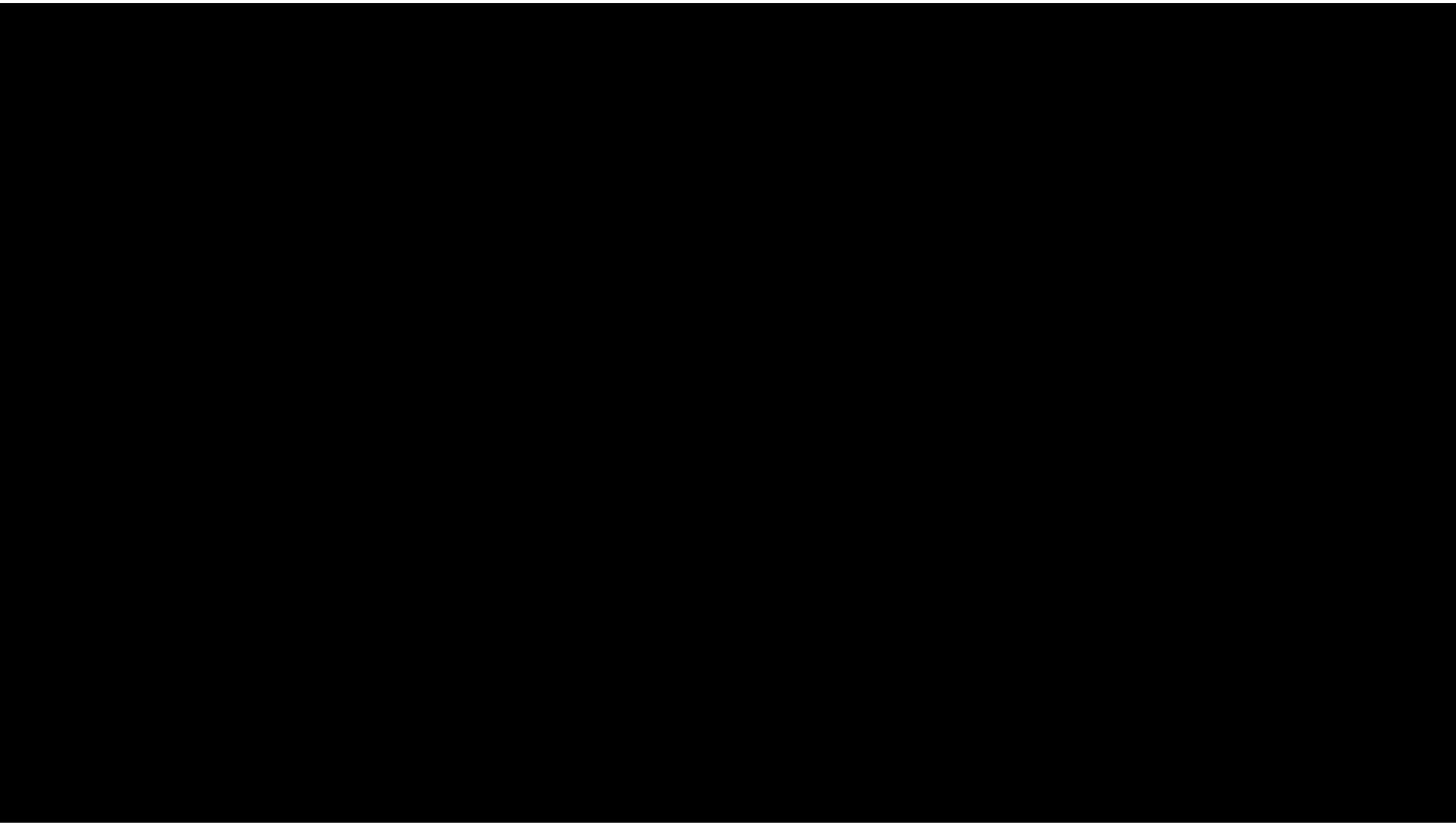
Implicit Modeling

[Inigo Quilez – https://www.shadertoy.com/view/4tByz3]

[Inigo Quilez – https://www.shadertoy.com/view/ldScDh]

[Inigo Quilez – https://www.shadertoy.com/view/4ttSWf]

[Inigo Quilez – https://www.shadertoy.com/view/3lsSzf]

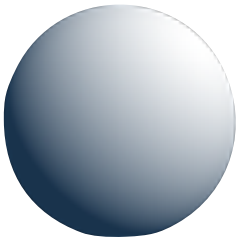


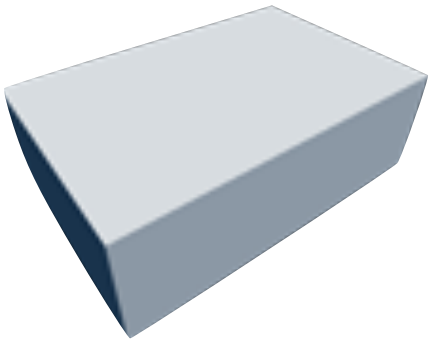
[Inigo Quilez – https://www.shadertoy.com/view/3lsSzf]

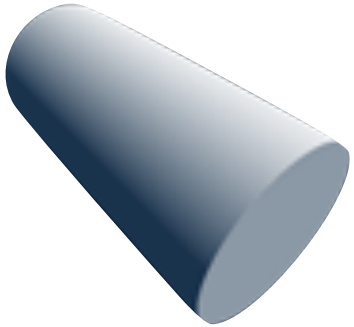
Primitive Shapes

- Create objects by combining basic shapes and sculpting
- Primitive shapes are all shapes you can analytically compute distance
- A good list can be found at http://www.iquilezles.org

```
float sdSphere(vec3f p, float s) { return length(p)-s; }
float sdBox(vec3f p, vec3f b) { vec3 q = abs(p) - b;
  return length(max(q,0.0)) + min(max(q.x,max(q.y,q.z)),0.0); }
float sdCylinder(vec3f p, float h, float r) {
  vec2 d = abs(vec2(length(p.xz),p.y)) - vec2(h,r);
  return min(max(d.x,d.y),0.0) + length(max(d,0.0)); }
```

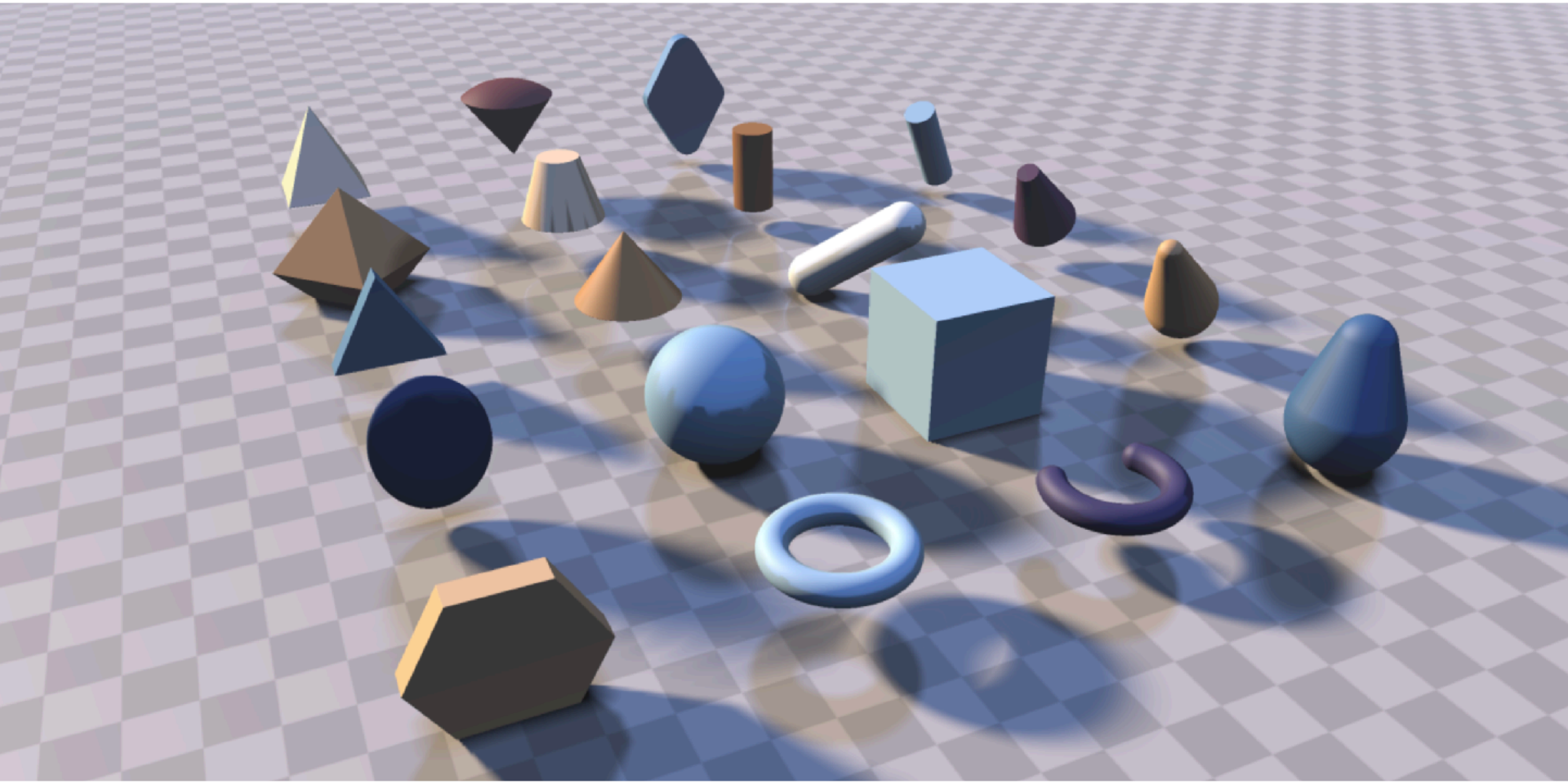






Primitive Shapes

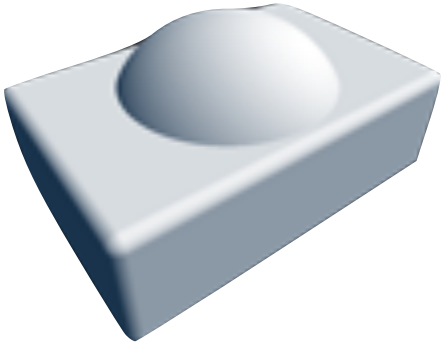
- Primitive shapes are all shapes you can analytically compute distance



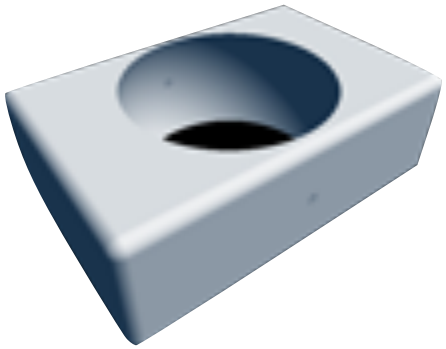
Boolean Operations

- Boolean operations in modeling are defined over object volumes
- We are interested in computing the surface of the resulting volume
- With SDFs we can trivially compute booleans

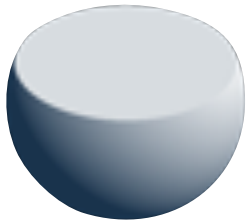
Union



Subtraction



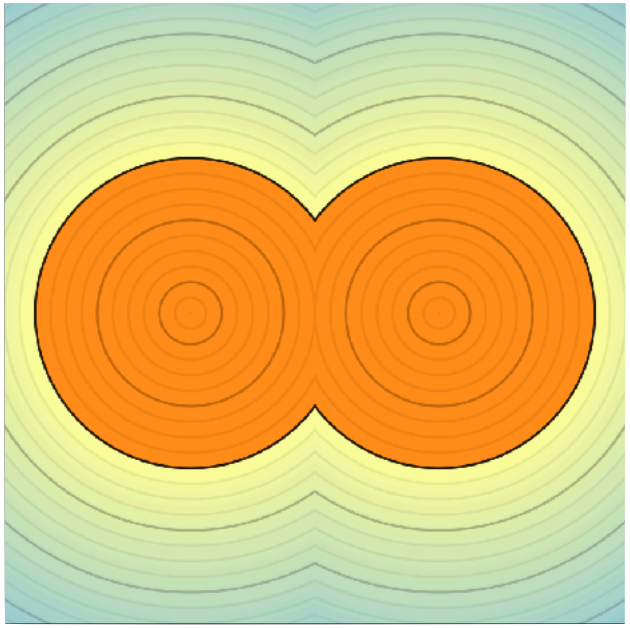
Intersection



Boolean Operations

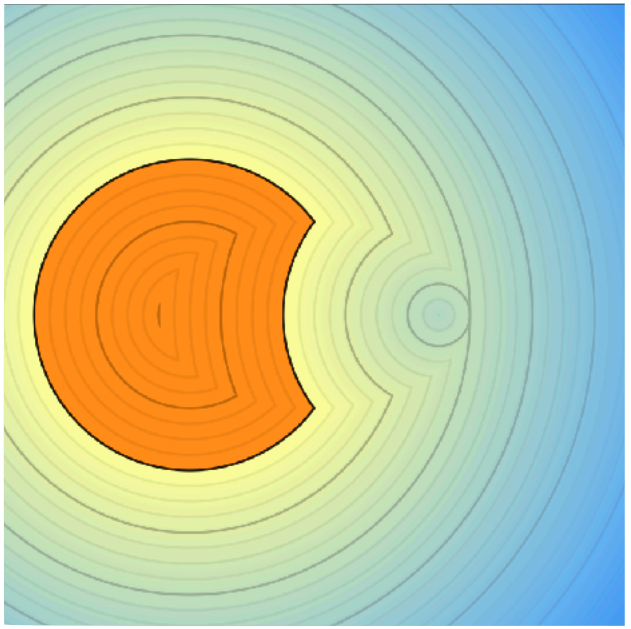
- Boolean can be defined by simply combining SDFs with min/max operations

Union



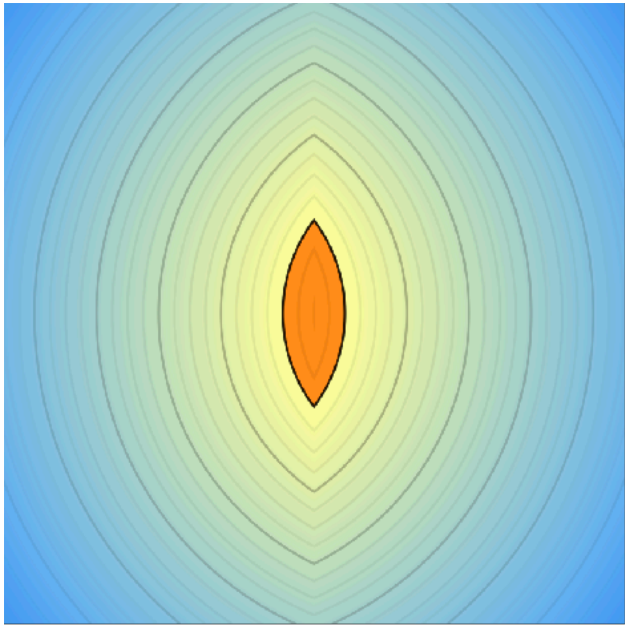
$f_u(\mathbf{p}) = \min(f_1(\mathbf{p}), f_2(\mathbf{p}))$

Subtraction



$f_s(\mathbf{p}) = \max(f_1(\mathbf{p}), -f_2(\mathbf{p}))$

Intersection

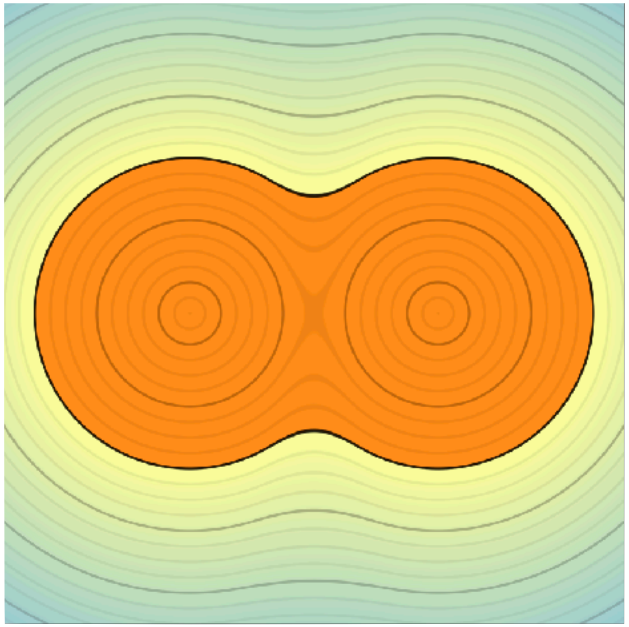


$f_i(\mathbf{p}) = \max(f_1(\mathbf{p}), f_2(\mathbf{p}))$

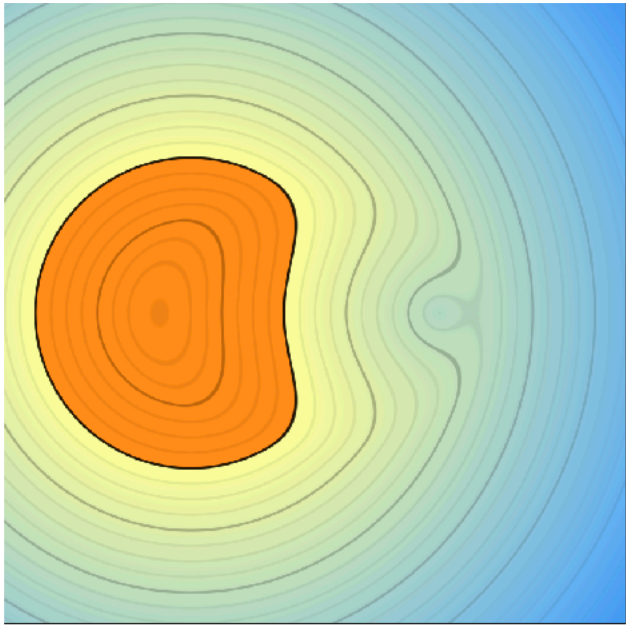
Smooth Boolean Operations

- Boolean operations can be generalized to perform smooth blending between shapes by introducing blending functions

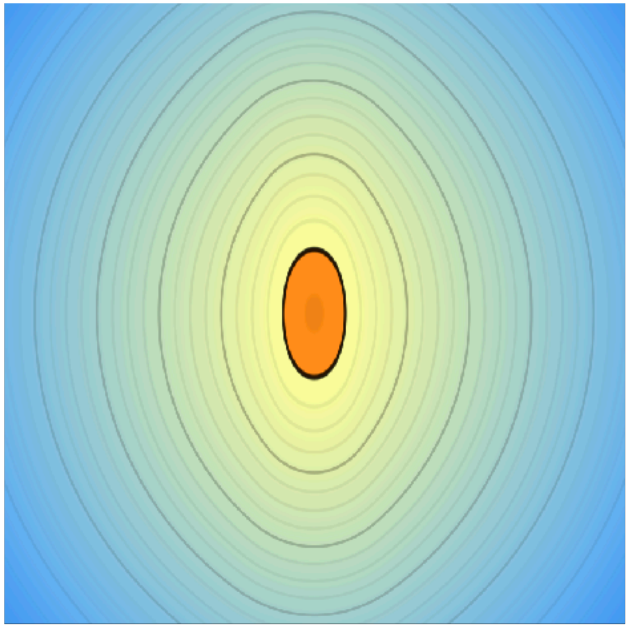
"Smooth" Union



"Smooth" Subtraction



"Smooth" Intersection



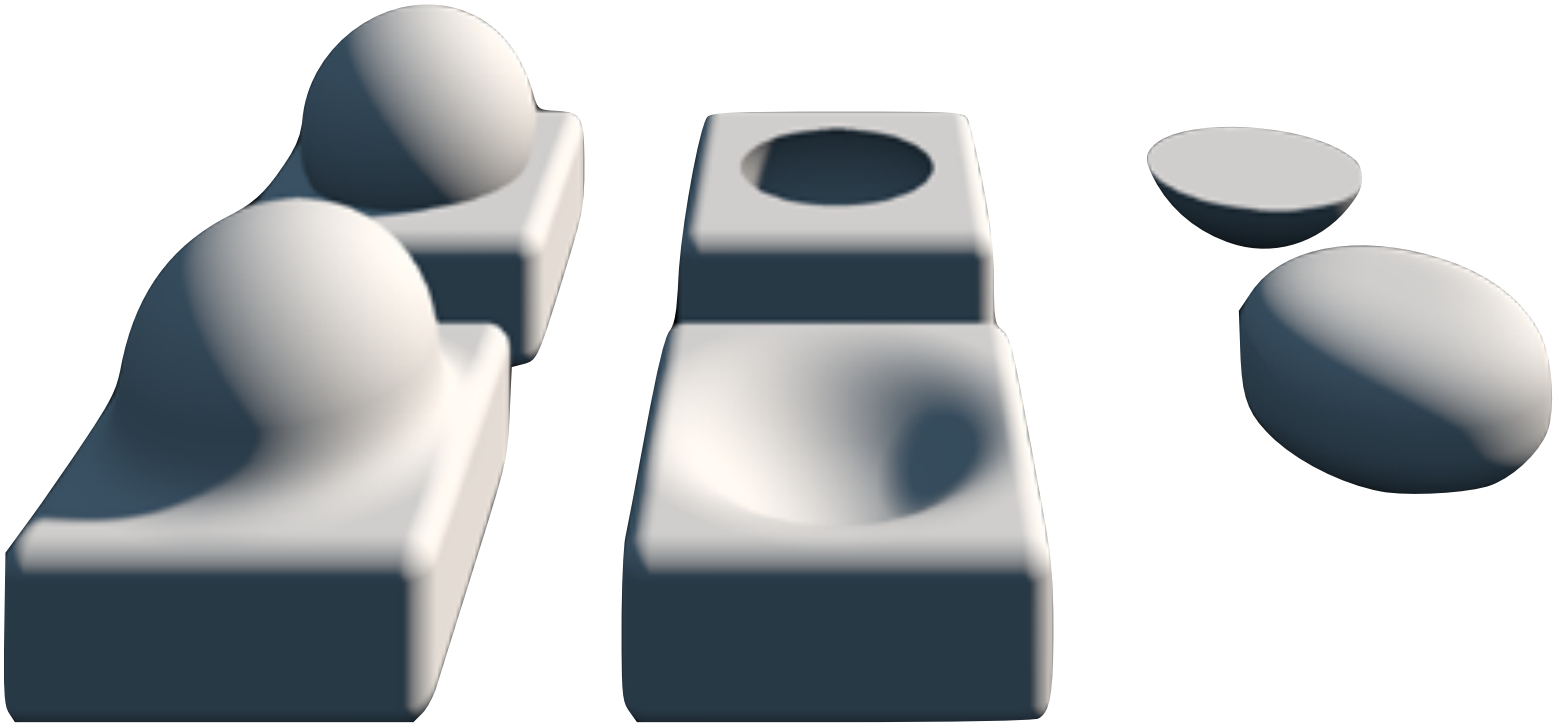
Smooth Boolean Operations

- Boolean operations can be generalized to perform smooth blending between shapes by introducing blending functions

Union

Subtraction

Intersection



Smooth Boolean Operations

- Boolean operations can be generalized to perform smooth blending between shapes by introducing blending functions

Boolean Operations

```
float opUnion( float d1, float d2 ) {  min(d1,d2); }

float opSubtraction( float d1, float d2 ) {
  return max(-d1,d2); }

float opIntersection( float d1, float d2 ) {
  return max(d1,d2); }

float opSmoothUnion( float d1, float d2, float k ) {
  float h = clamp( 0.5 + 0.5*(d2-d1)/k, 0.0, 1.0 );
  return mix( d2, d1, h ) - k*h*(1.0-h); }

float opSmoothSubtraction( float d1, float d2, float k ) {
  float h = clamp( 0.5 - 0.5*(d2+d1)/k, 0.0, 1.0 );
  return mix( d2, -d1, h ) + k*h*(1.0-h); }

float opSmoothIntersection( float d1, float d2, float k ) {
  float h = clamp( 0.5 - 0.5*(d2-d1)/k, 0.0, 1.0 );
  return mix( d2, d1, h ) + k*h*(1.0-h); }
```

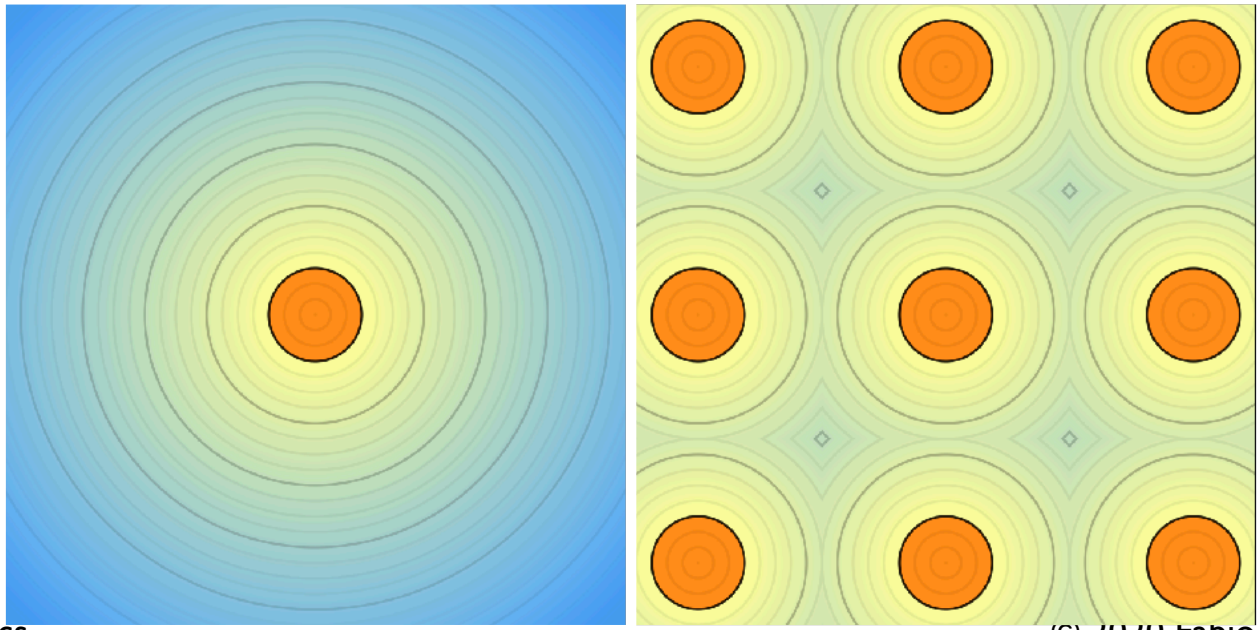

Transforms and Repetition

- We can transform an SDF by querying the function with the point inversely transformed

$$Tf(\mathbf{p}) = f(T^{-}1\mathbf{p})$$

- We can repeat a shape by taking the modulus of the point

$$f_r(\mathbf{p}, \mathbf{c}) = f(\mathrm{mod}(\mathbf{p} + \mathbf{c}/2, \mathbf{c}) - \mathbf{c}/2)$$

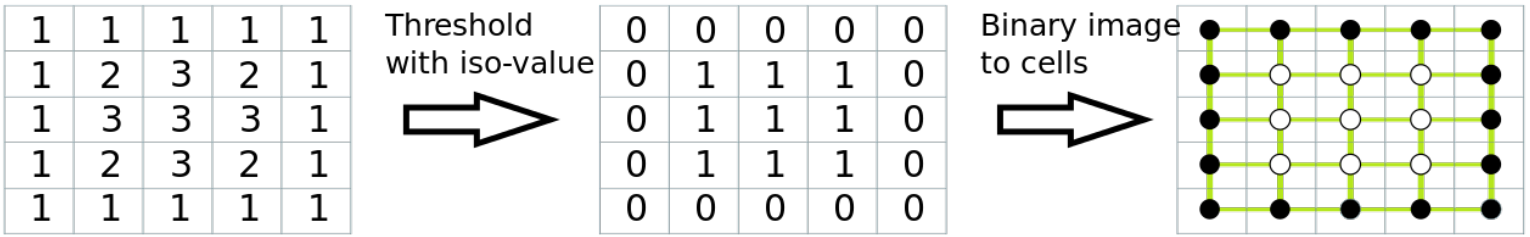


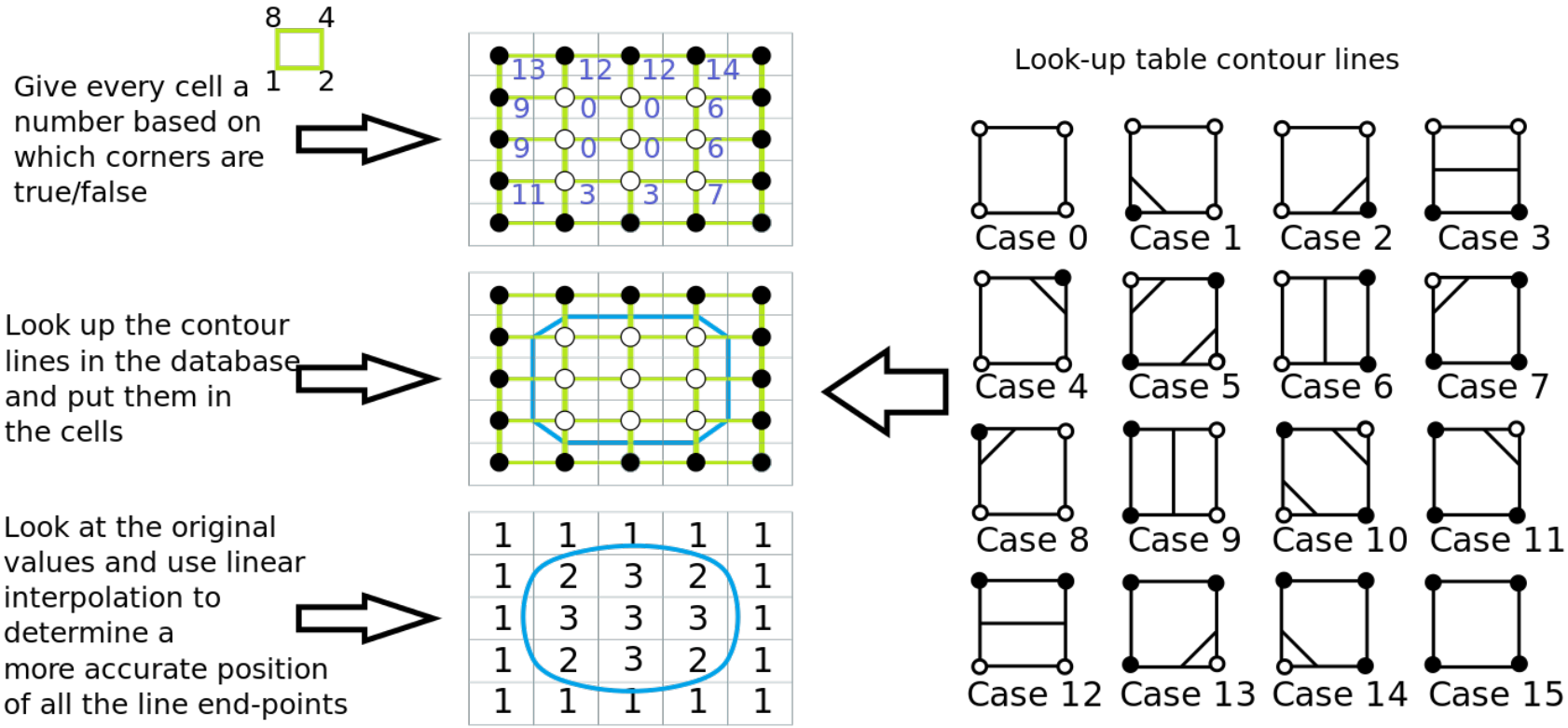
Rendering Implicit Surfaces

Rendering Implicit Surfaces

- Method 1: convert SDF to poly lines (2D) or meshes (3D)
  - Many well-studied algorithms to do this
  - Different tradeoffs
  - All quite tricky to implement correctly, so won't cover in details
- Method 2: retrace SDFs directly
  - Fairly robust method
  - Still not we'll studied for all cases

Marching squares



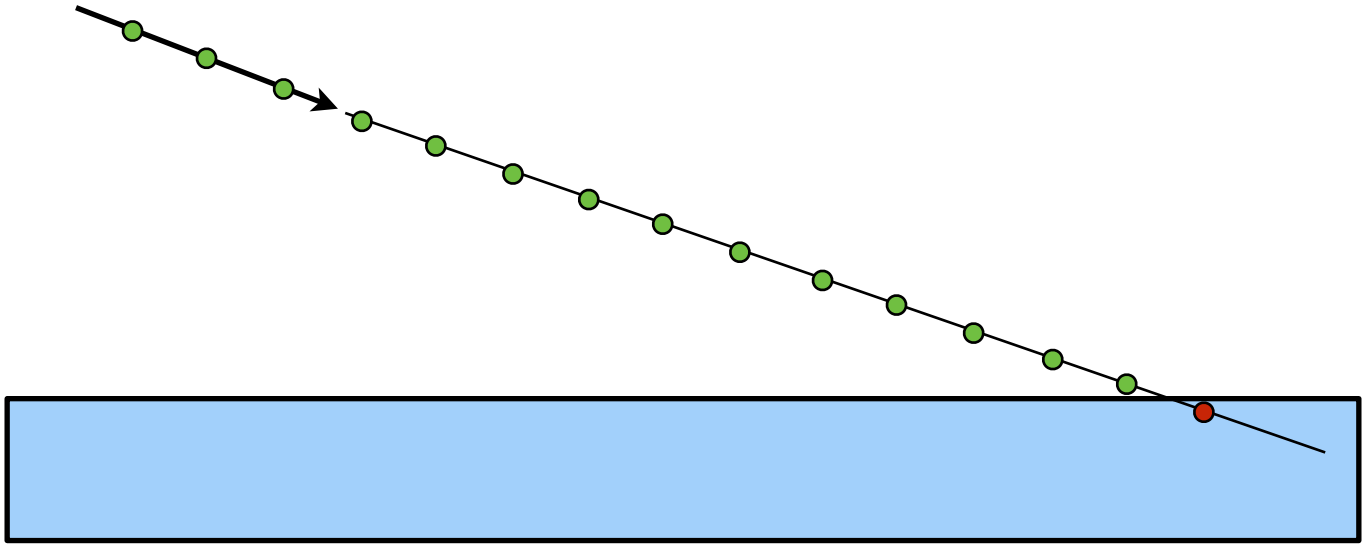


Ray tracing implicits

- For models that are boolean combination of analytics primitives, we could compute the solution numerically since method exists for all those surfaces
- But that would be restrictive and would not work for general SDF
- The most common solution is to solve the problem numerically
  - Several methods exist in practice

Ray marching

- The simplest method is to walk along the ray at fixed increments and stop when the SDF is negative
- This method works in all cases, but it is not very accurate nor very fast



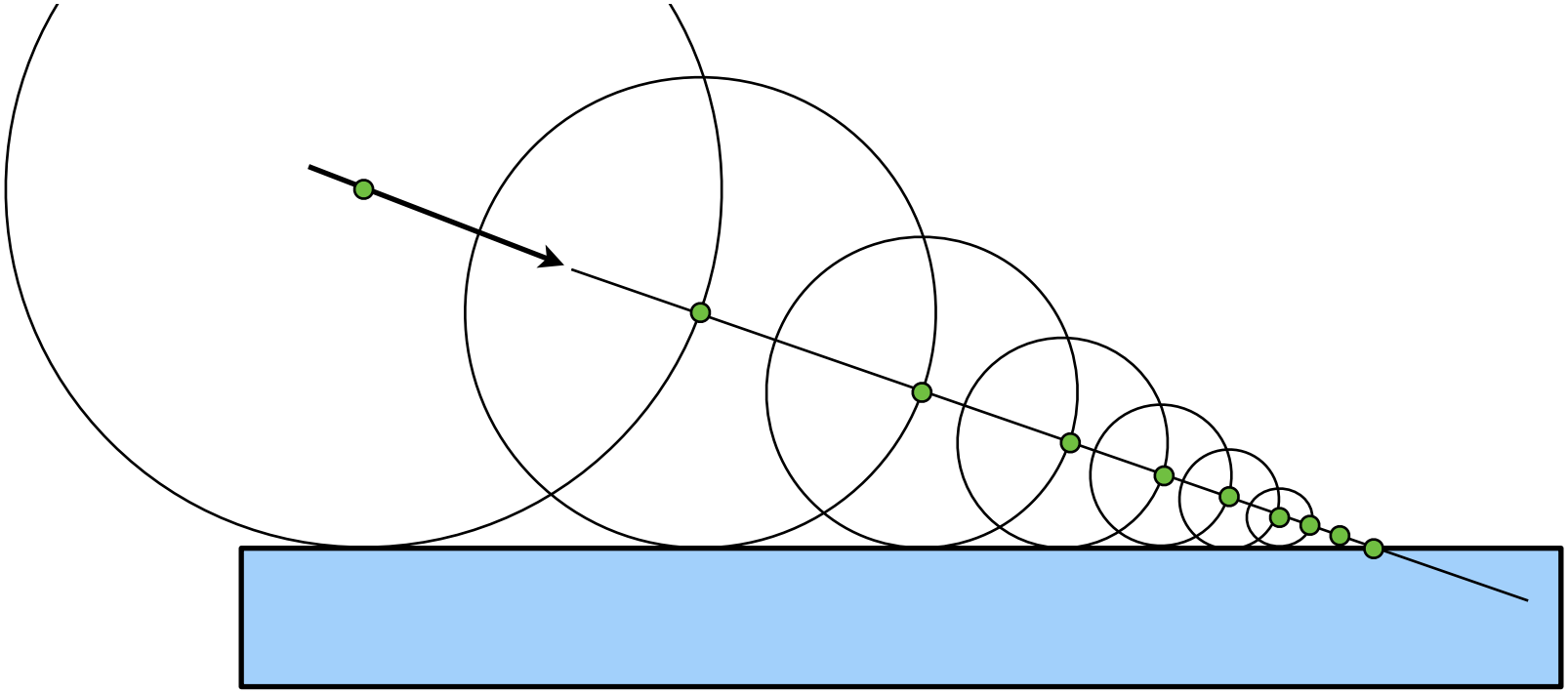
Ray marching

- The simplest method is to walk along the ray at fixed increments and stop when the SDF is negative
- This method works in all cases, but it is not very accurate nor very fast

```
bool raytrace(ray3f ray, SDF* sdf, float* dist) {
  auto t = ray.tmin;
  while(t < ray.tmax) {
    auto p = eval_ray(ray, t);
    if(eval_sdf(sdf, p) < 0) { *dist = t; return true; }
    t += step_size;
  }
  return false;
}
```


Sphere tracing

- The key to make ray marching faster and more accurate is to step along the ray each time with the longest distance we know it is safe
- That distance is just the SDF itself evaluated at the point we are at



Sphere tracing

- The key to make ray marching faster and more accurate is to step along the ray each time with the longest distance we know it is safe
- That distance is just the SDF itself evaluated at the point we are at

```
bool spheretrace(ray3f ray, SDF* sdf, float* dist) {
  auto t = ray.tmin;
  while(t < ray.tmax) {
    auto p = eval_ray(ray, t);
    auto d = eval_sdf(sdf, p);
    if(d < epsilon) { *dist = t; return true; }
    t += d;
  }
  return false;
}
```

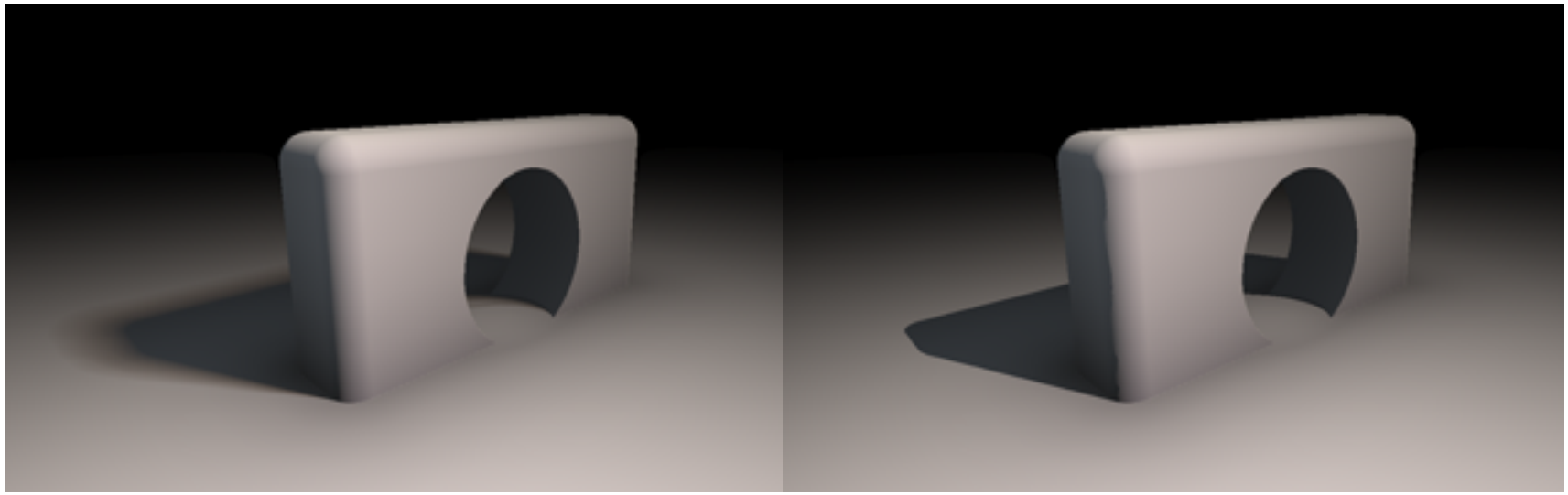

Texture mapping

- SDFs do not define texture coordinates and support booleans
- Instead of applying 2D texture, we apply 3D textures to them
- 3D texture are defined in the volume so we can look them up at any position
- Just like real-world objects that have the inside filled in

[http://lernertandsander.com/cubes]

Lighting SDFs

- Since we can retraced them, we can use any lighting algorithm
- We can also cheat lighting by taking advantage of the SDF nature



Dreams