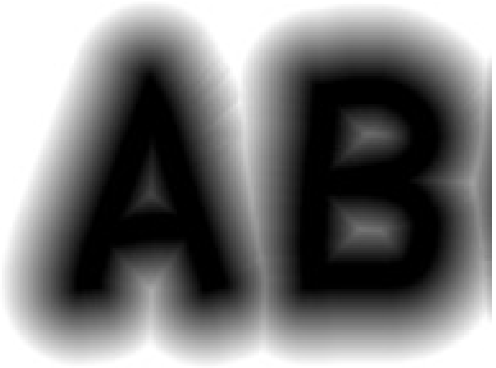
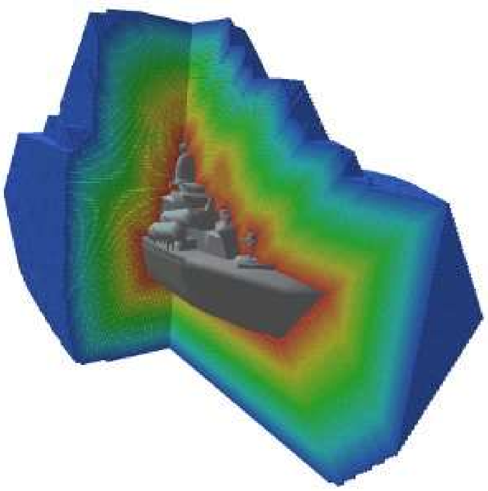
Sphere Tracing Distance Fields

Distance Fields

$\mathbb{R}^2 \rightarrow dist(\mathbb{R}^2)$

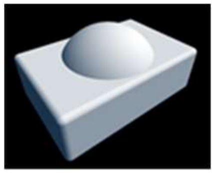
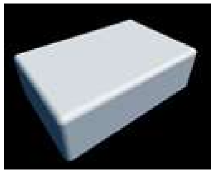
$\mathbb{R}^3 \rightarrow dist(\mathbb{R}^3)$



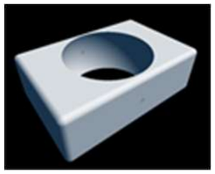
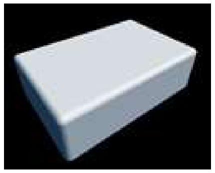


Operations on Distance Fields

- Given $dist_1(\mathbb{R}^3)$ and $dist_2(\mathbb{R}^3)$

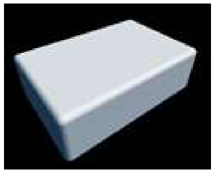
 $= union\Bigg($  , $\Bigg)$

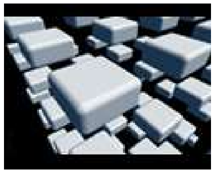
- The union is $\min(dist_1(\mathbb{R}^3), dist_2(\mathbb{R}^3))$

 $= substraction\Bigg($  , $\Bigg)$

- The subtraction is $\max(-dist_1(\mathbb{R}^3), dist_2(\mathbb{R}^3))$

Operations on Distance Fields

- Given $dist(\mathbb{R}^3) =$ 

 $= dist(repeat(\mathbb{R}^3))$

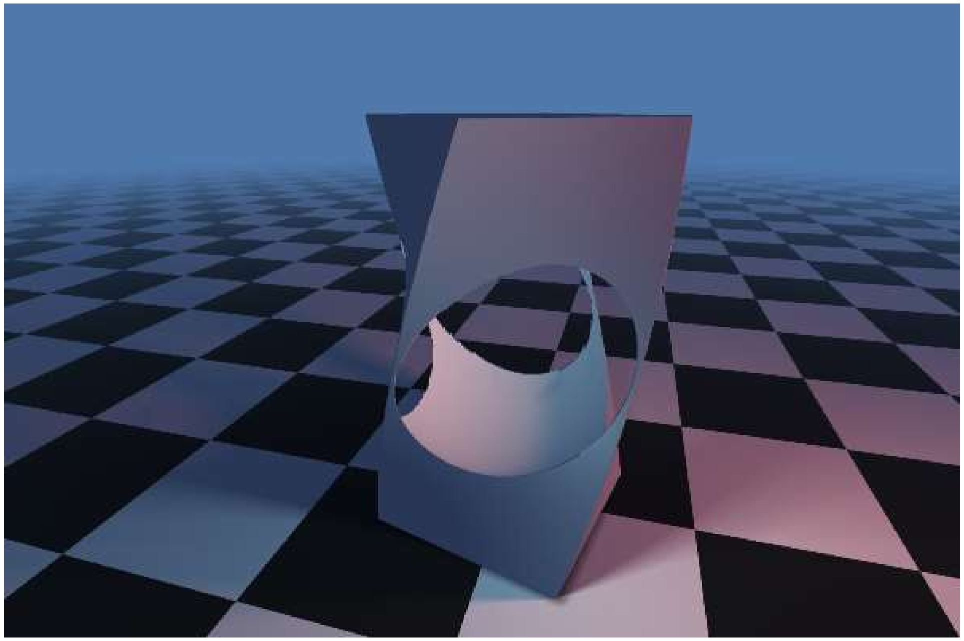
- Repeat is $\mathrm{mod}\left(\mathbf{P}, \vec{\mathbf{b}}\right) - \frac{1}{2}\vec{\mathbf{b}}$

  were $\mathrm{mod}(\vec{\mathbf{a}}, \vec{\mathbf{c}})$ is component-wise $\vec{\mathbf{a}}$ modulo $\vec{\mathbf{c}}$

Space Warping

- Manipulate input point

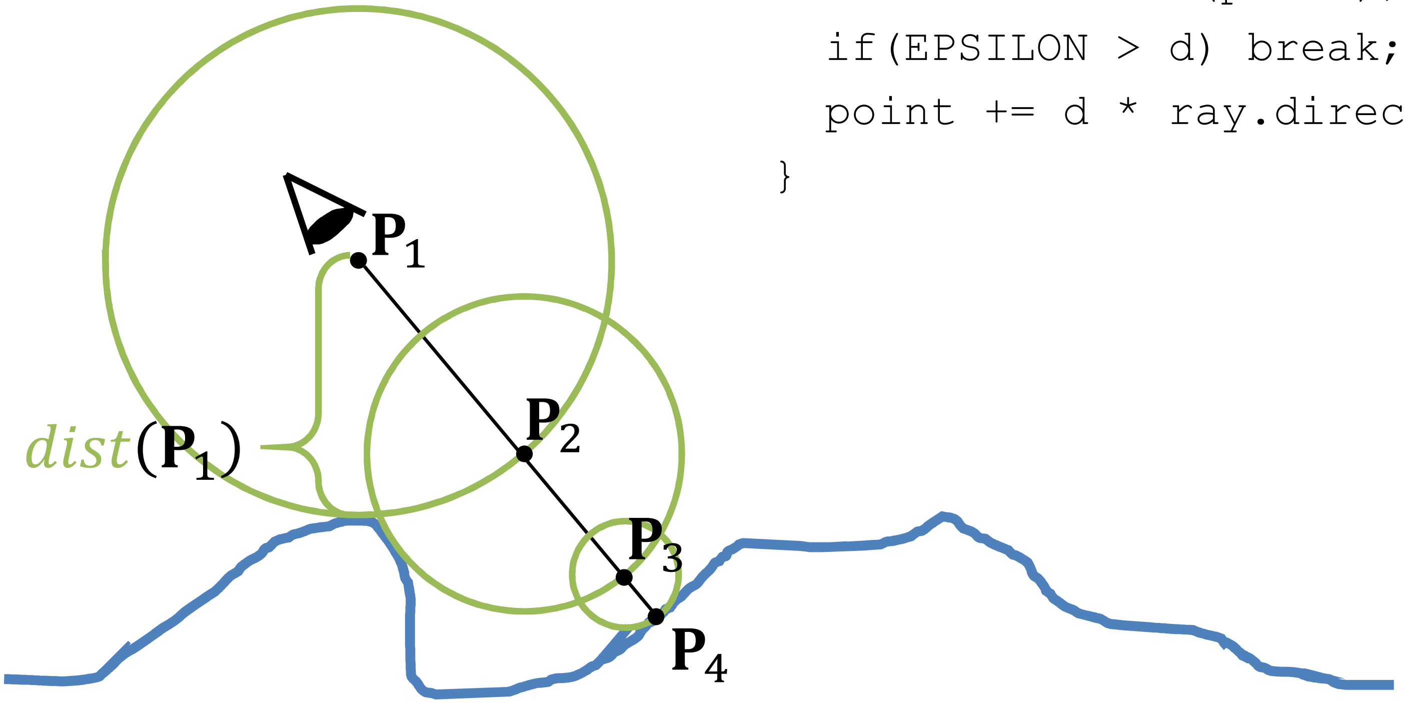
```
float twistedCube(vec3 p)
{
  vec3 q = rotateY(p, 0.5 * p.y);
  return cube(q);
}
```



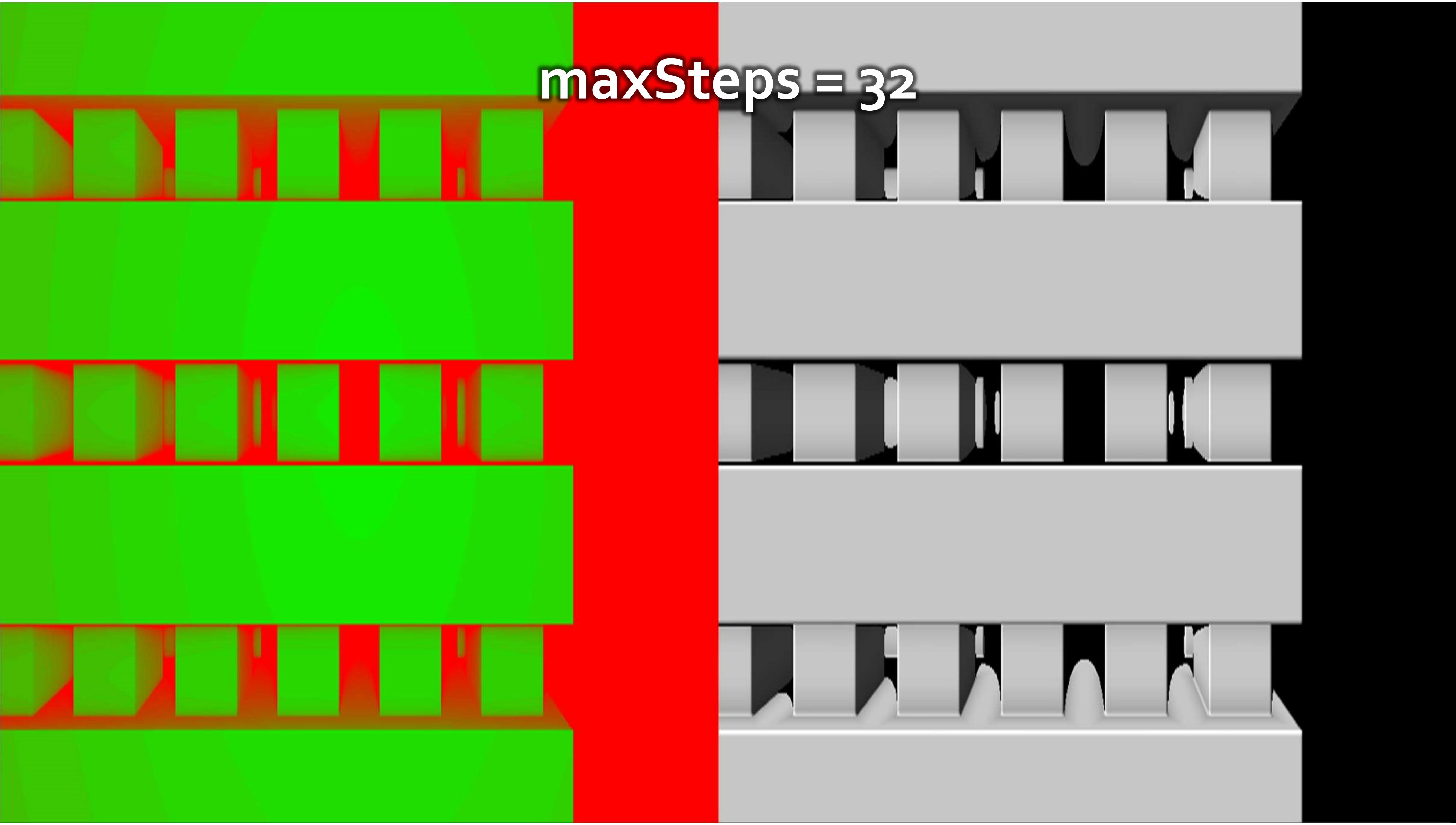
Sphere Tracing Distance Fields

- $\mathit{dist}(\mathbf{P}_i)$

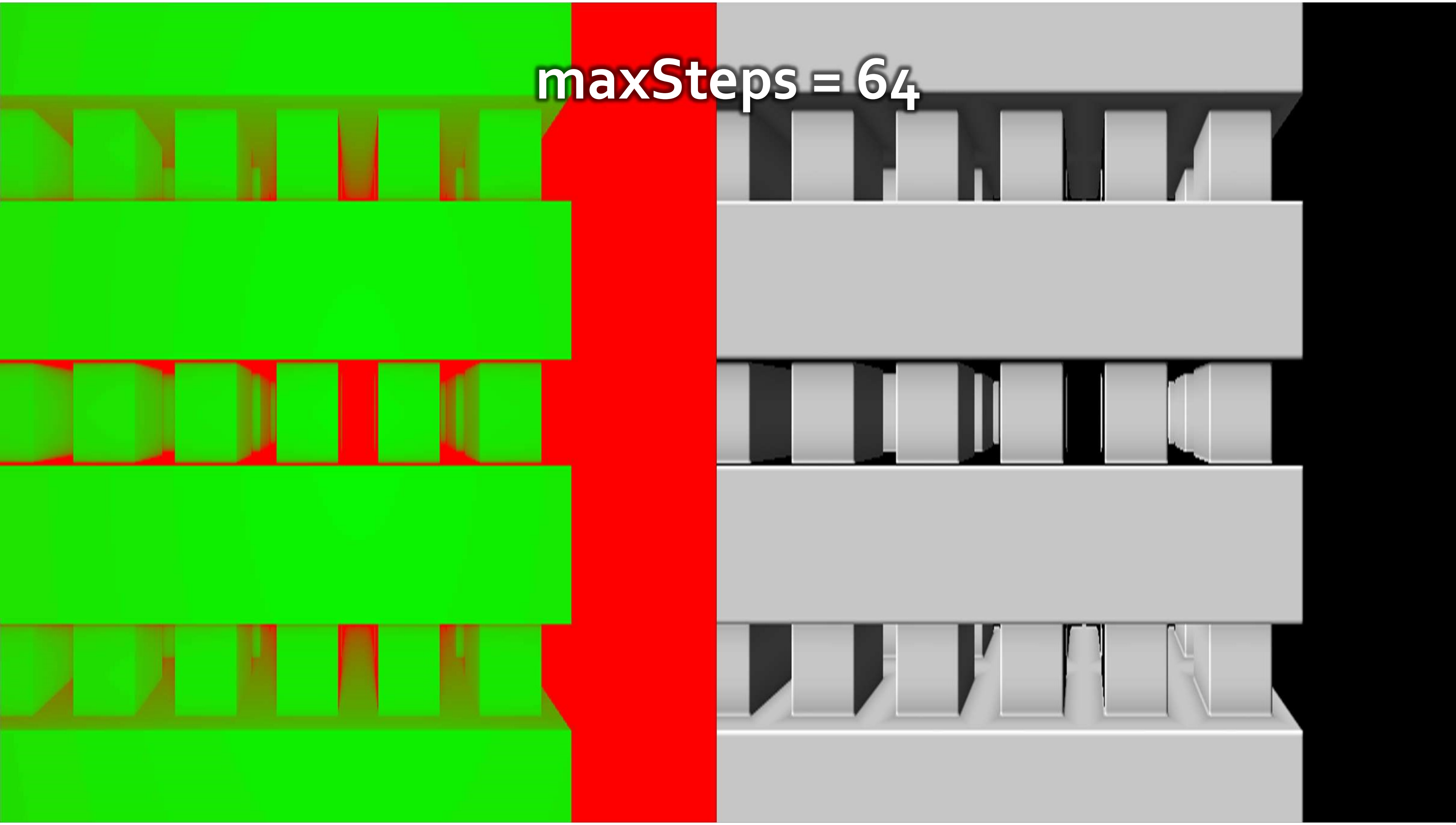
```
vec3 point = ray.origin;
while(--maxSteps) {
  float d = dist(point);
  if(EPSILON > d) break;
  point += d * ray.direction;
}
```



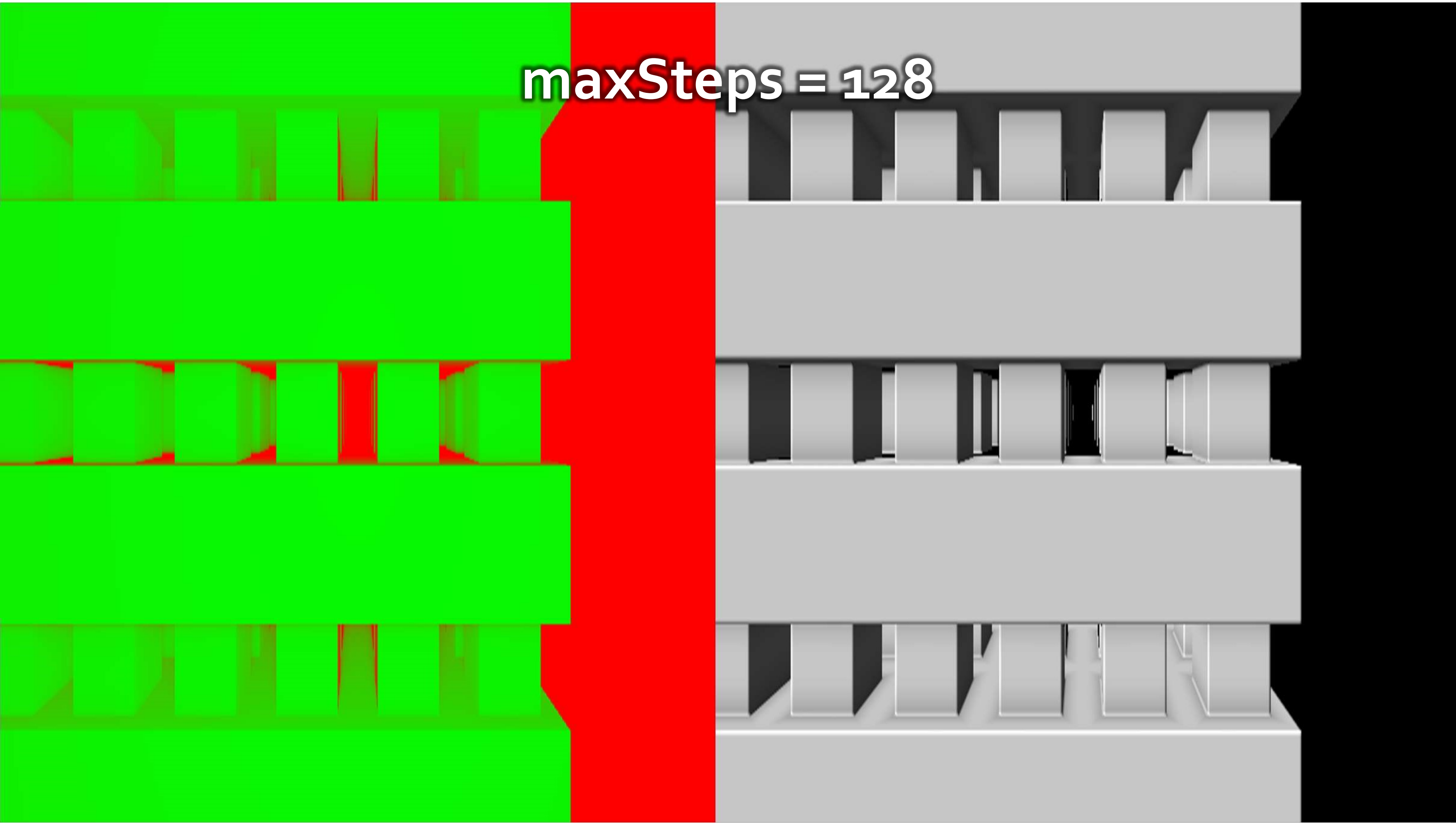
Quality vs. Speed

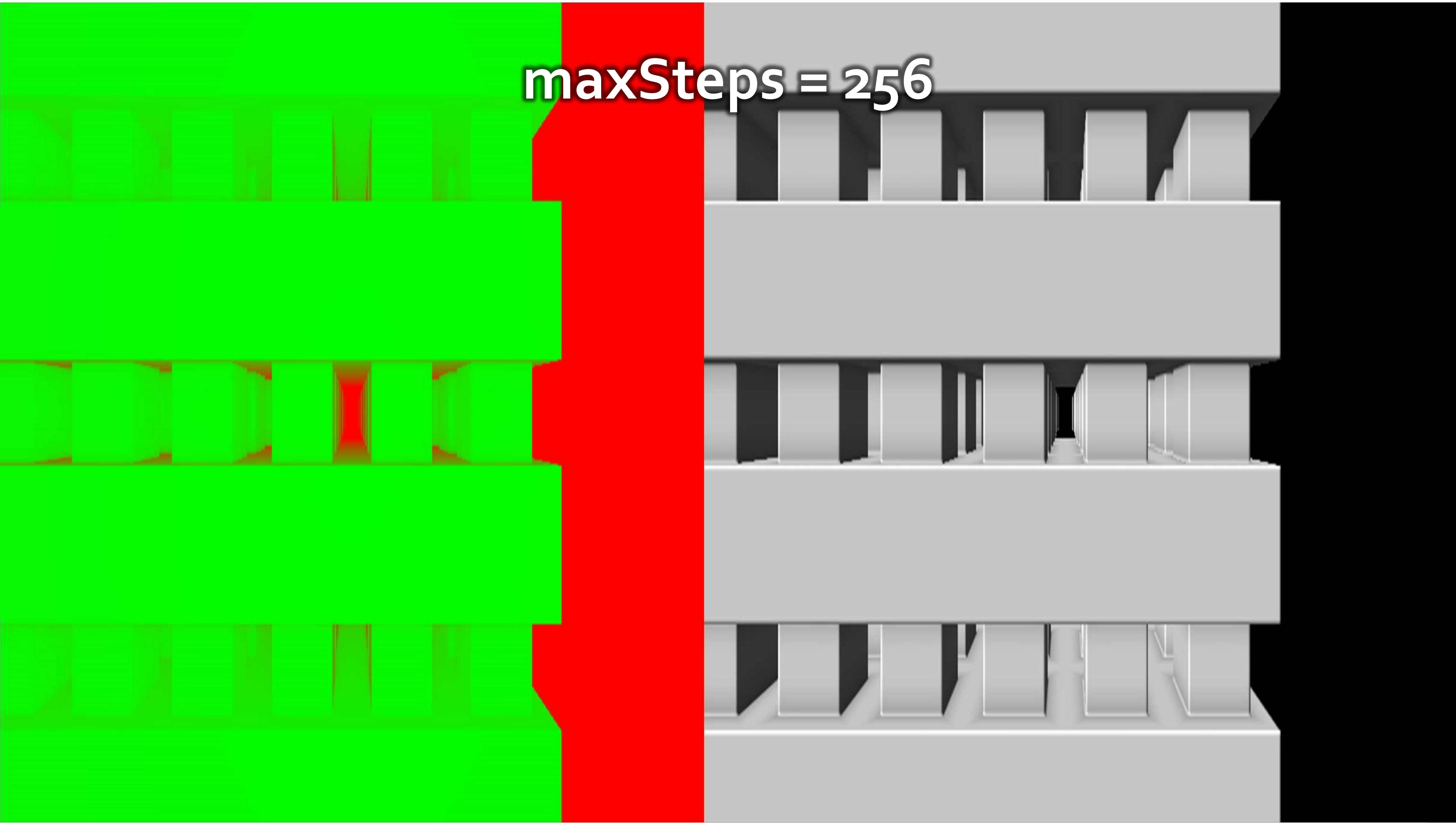


maxSteps = 32

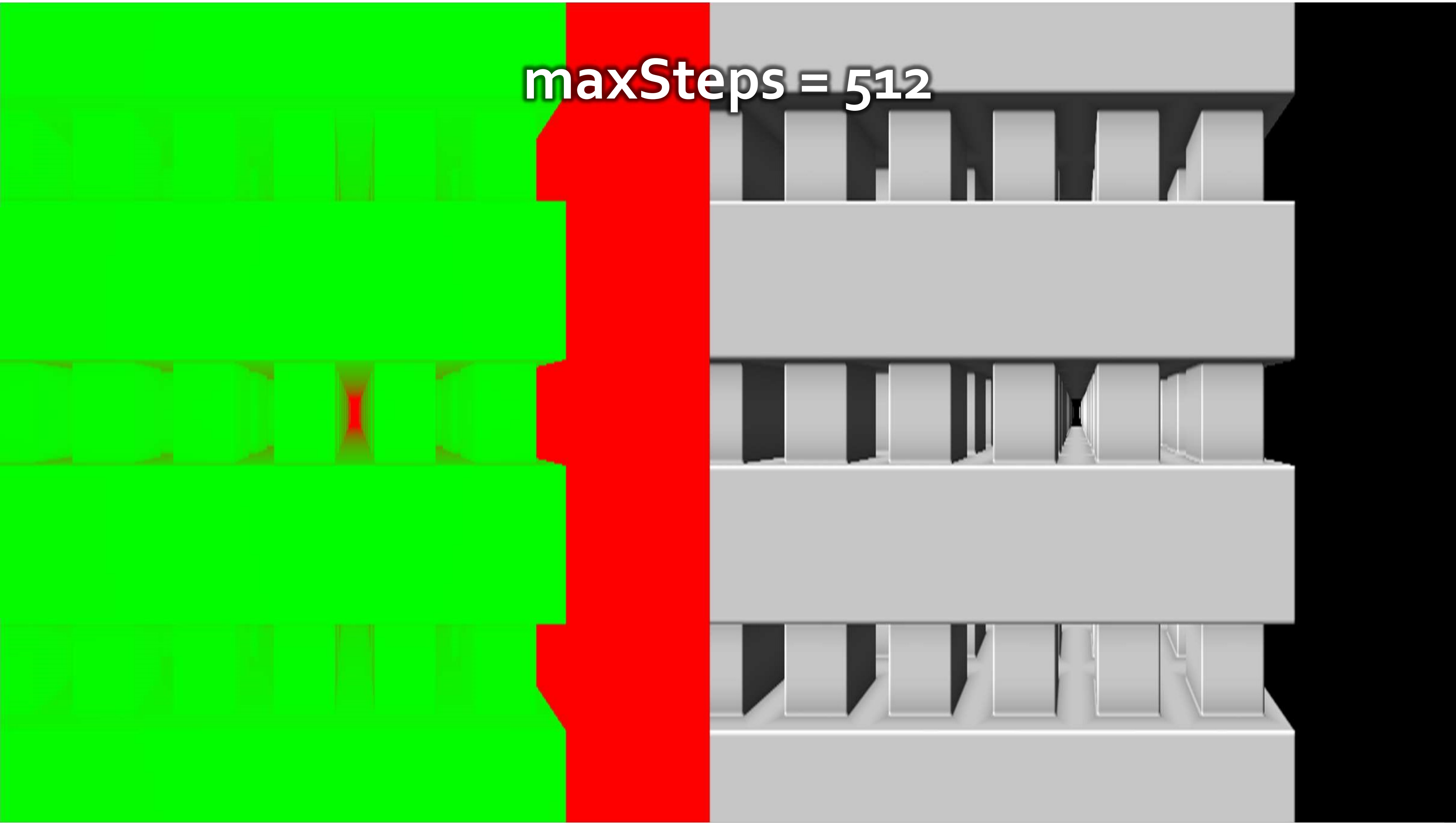


maxSteps = 64


maxSteps = 128



maxSteps = 256

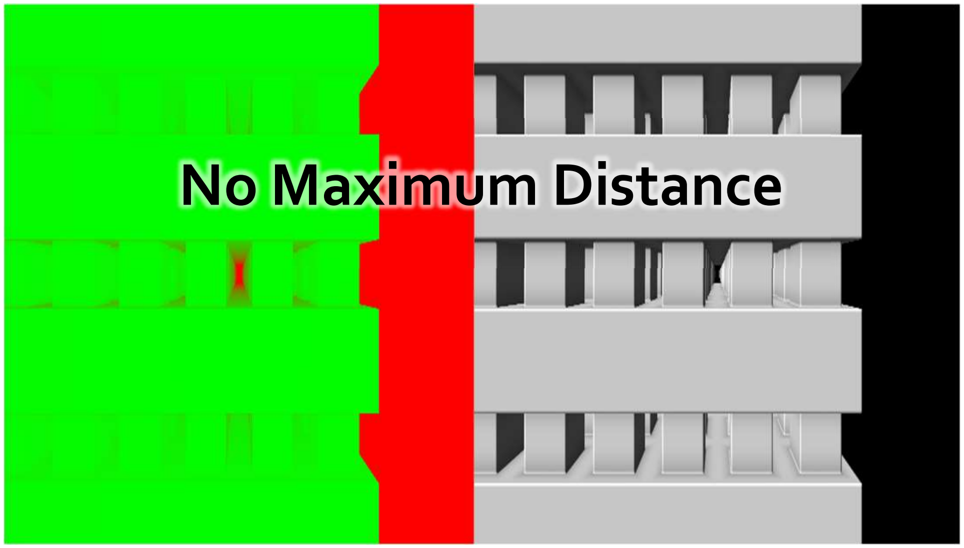


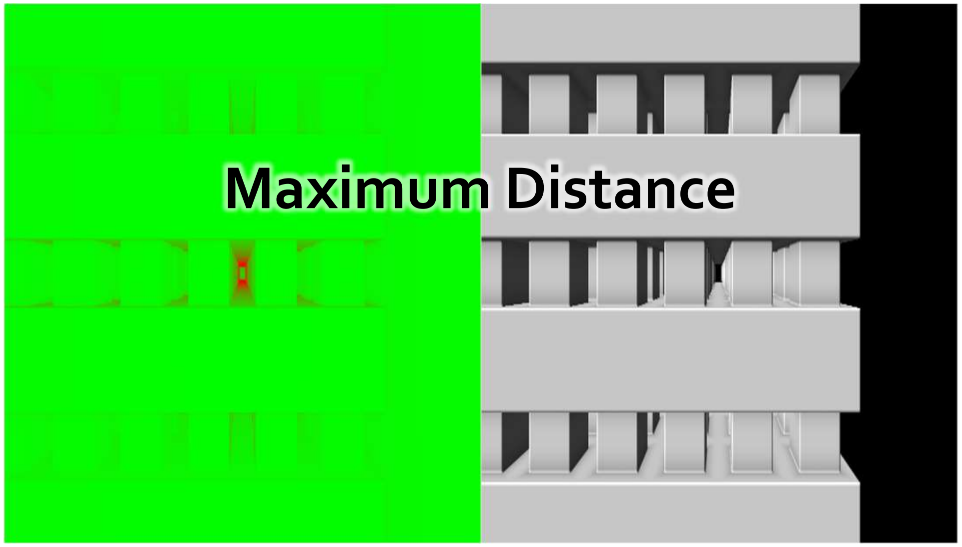
maxSteps = 512

Maximum Distance

- Scene dependent
- Avoids most costly rays
- Could use bounding volume

```
vec3 point = ray.origin;
while(--maxSteps) {
  float d = dist(point);
  if(EPSILON > d) break;
  t += d;
  point = ray.origin + d * ray.direction;}
```

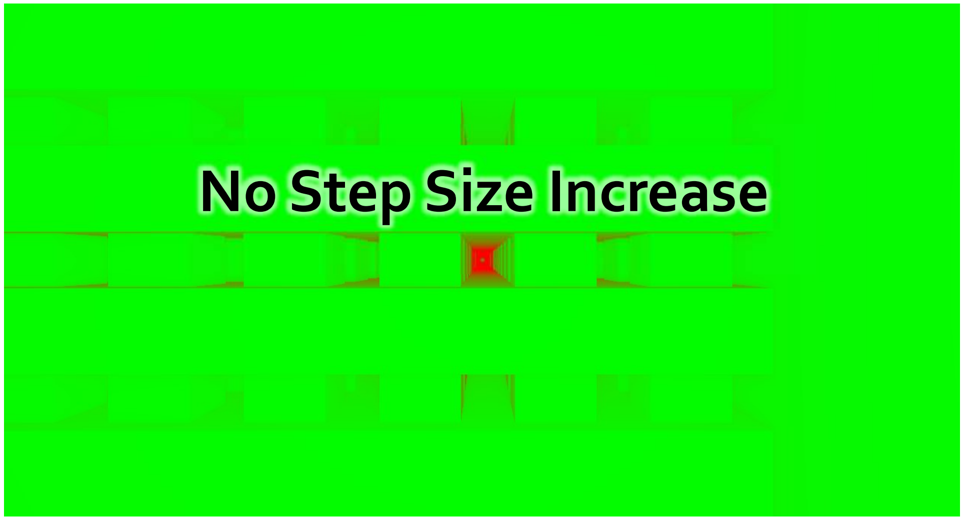


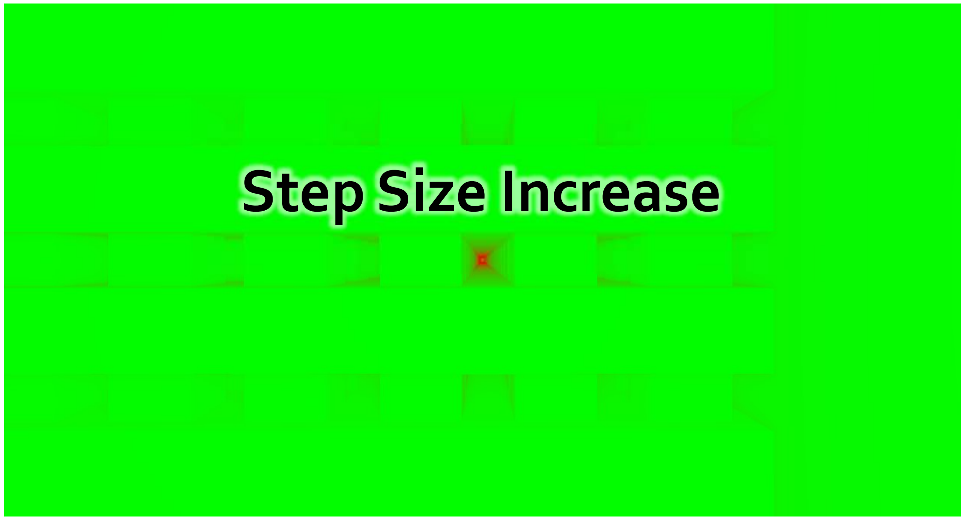


Step Size Increase

- Screen error decreases with distance

```
for(steps = 0, t = 0; (steps < maxSteps)&&(t < maxDistance); ++steps) {
  float d = dist(point);
  if(EPSILON > d) break;
  t += max(d, t * 0.001); // some increase factor
  point = ray.origin + t * ray.direction; }
```

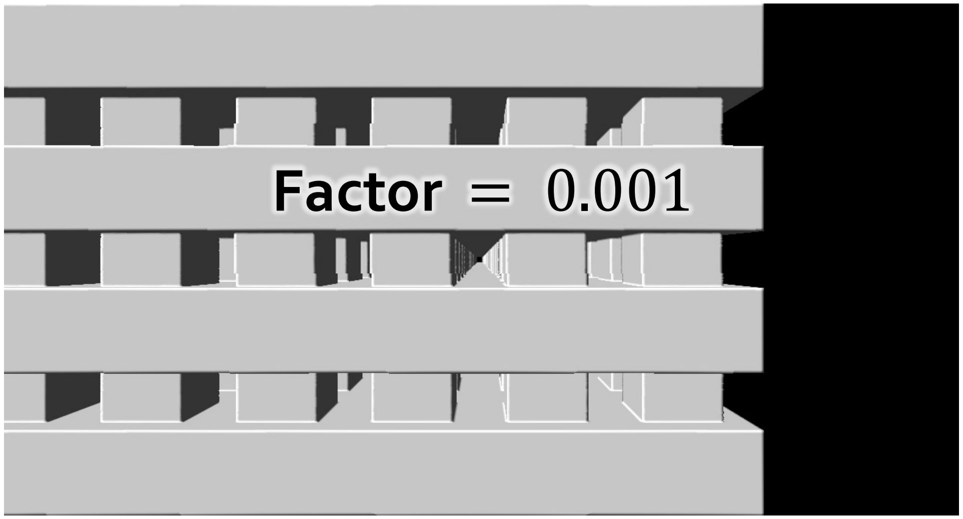


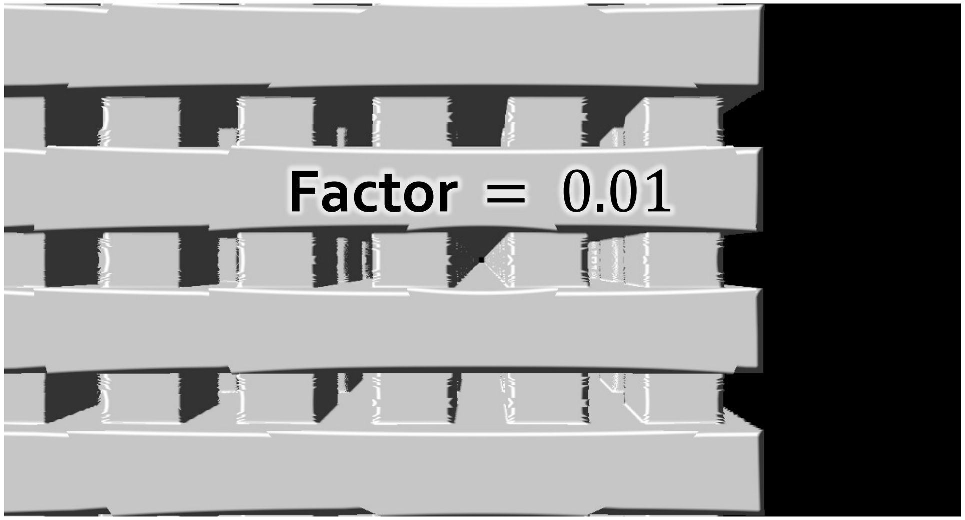


Step Size Increase

- Screen error decreases with distance

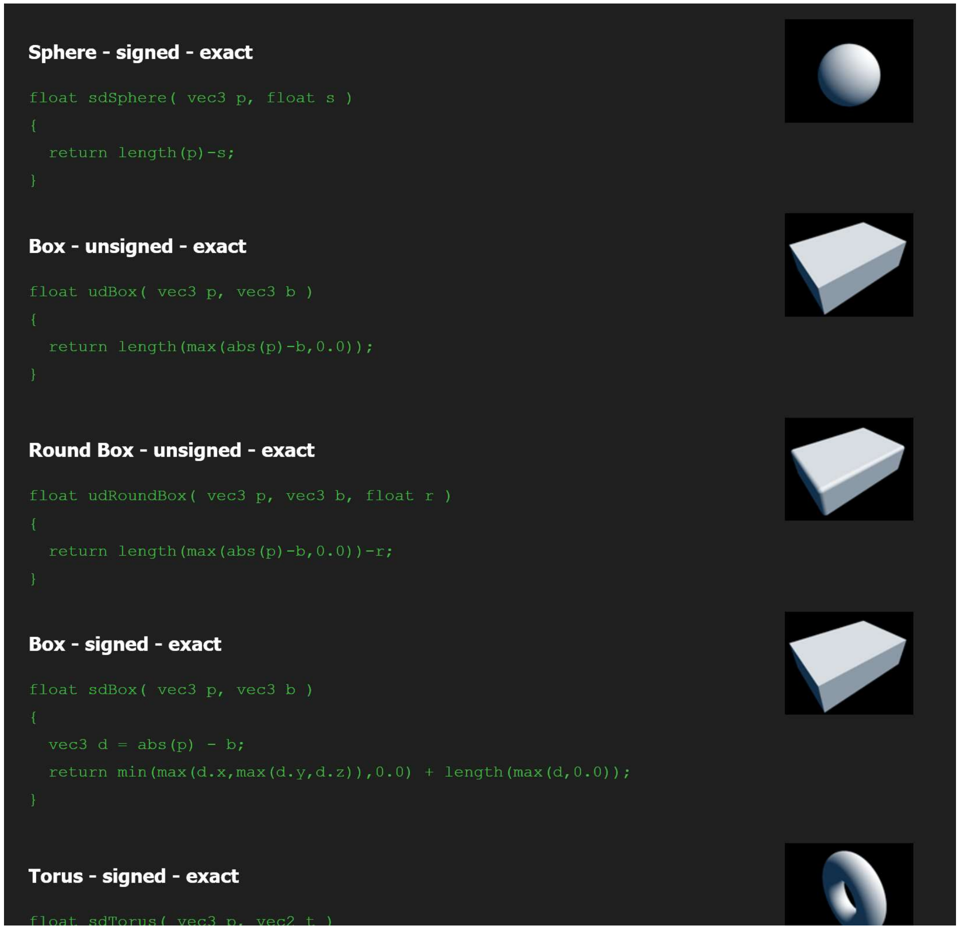
```
for(steps = 0, t = 0; (steps < maxSteps)&&(t < maxDistance); ++steps) {
  float d = dist(point);
  if(EPSILON > d) break;
  t += max(d, t * 0.001); // some increase factor
  point = ray.origin + t * ray.direction; }
```





Links

- Overview
  9bitscience.blogspot.de/2013/07/raymarching-distance-fields_14.html
- Distance functions
  iquilezles.org/www/articles/distfunctions/distfunctions.htm
- Distance function glsl lib
  mercury.sexy/hg_sdf



Sphere - signed - exact

```
float sdSphere( vec3 p, float s )
{
  return length(p)-s;
}
```

Box - unsigned - exact

```
float udBox( vec3 p, vec3 b )
{
  return length(max(abs(p)-b,0.0));
}
```

Round Box - unsigned - exact

```
float udRoundBox( vec3 p, vec3 b, float r )
{
  return length(max(abs(p)-b,0.0))-r;
}
```

Box - signed - exact

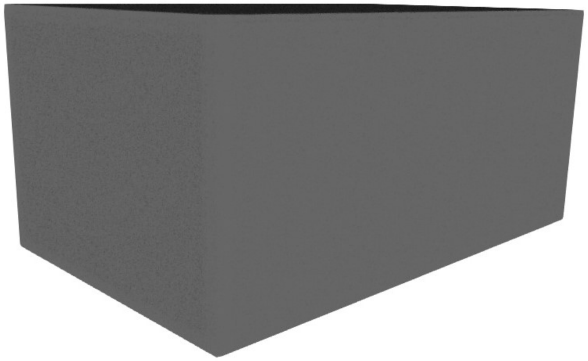
```
float sdBox( vec3 p, vec3 b )
{
  vec3 d = abs(p) - b;
  return min(max(d.x,max(d.y,d.z)),0.0) + length(max(d,0.0));
}
```

Torus - signed - exact

```
float sdTorus( vec3 p, vec2 t )
```

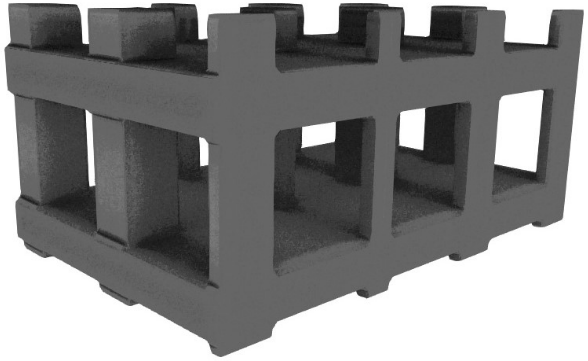
A Box

```
Box(pos, size)
{
  a = abs(pos-size) - size;
  return max(a.x,a.y,a.z);
}
```



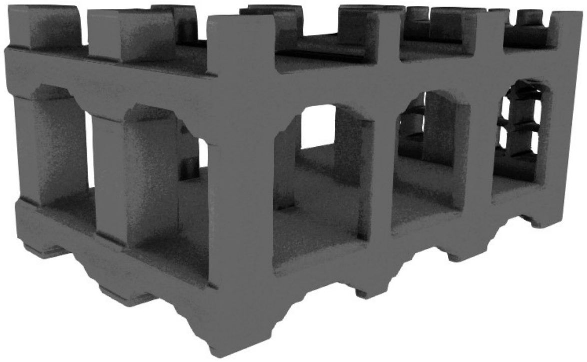
Cutting with Booleans

```
d = Box(pos)
c = fmod(pos * A, B)
subD = max(c.y,min(c.y,c.z))
d = max(d, -subD)
```



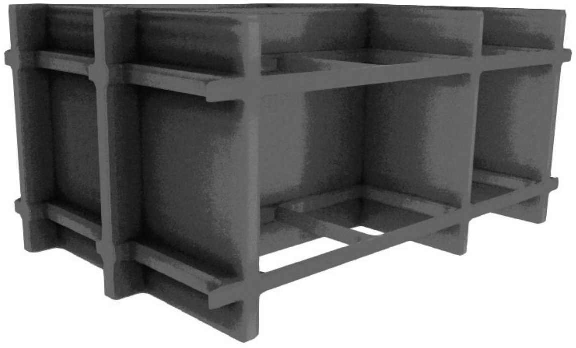
More Booleans

```
d = Box(pos)
c = fmod(pos * A, B)
subD = max(c.y,min(c.y,c.z))
subD = min(subD,cylinder(c))
subD = max(subD, Windows())
d = max(d, -subD)
```



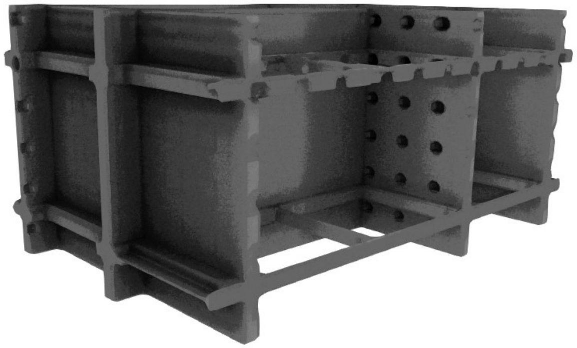
Repeated Booleans

```
d = Box(pos)
e = fmod(pos + N, M)
floorD = Box(e)
d = max(d, -floorD)
```



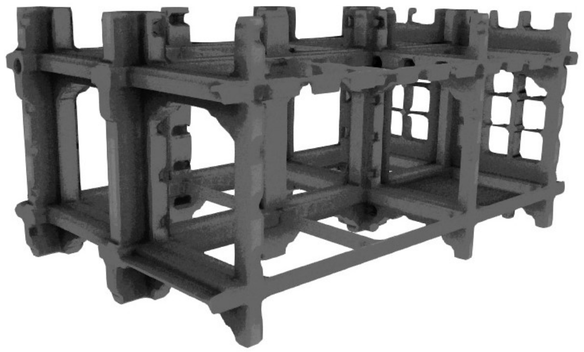
Cutting Holes

```
d = Box(pos)
e = fmod(pos + N, M)
floorD = Box(e)
floorD = min(floorD,holes())
d = max(d, -floorD)
```



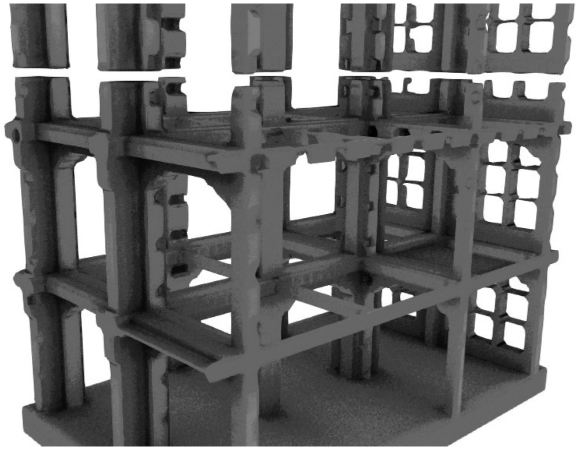
Combined Result

```
d = Box(pos)
c = fmod(pos * A, B)
subD = max(c.y,min(c.y,c.z))
subD = min(subD,cylinder(c))
subD = max(subD, Windows())
e = fmod(pos + N, M)
floorD = Box(e)
floorD = min(floorD,holes())
d = max(d, -subD)
d = max(d, -floorD)
```



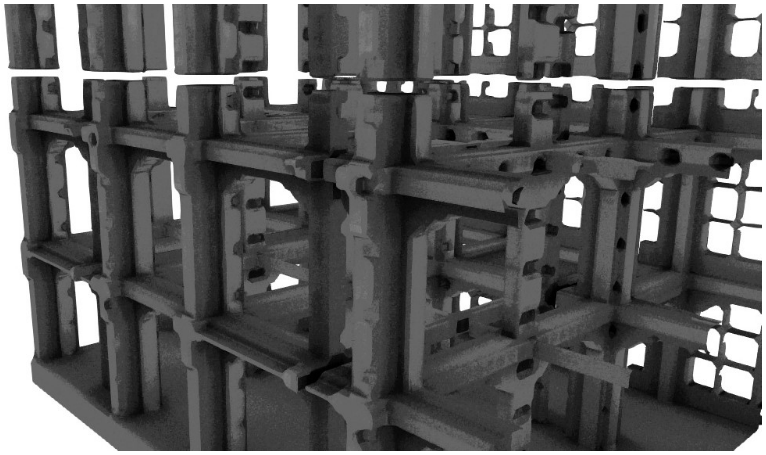
Repeating the Space

```
pos.y = frac(pos.y)
d = Box(pos)
c = fmod(pos * A, B)
subD = max(c.y,min(c.y,c.z))
subD = min(subD,cylinder(c))
subD = max(subD, Windows())
e = fmod(pos + N, M)
floorD = Box(e)
floorD = min(floorD,holes())
d = max(d, -subD)
d = max(d, -floorD)
```



Repeating the Space

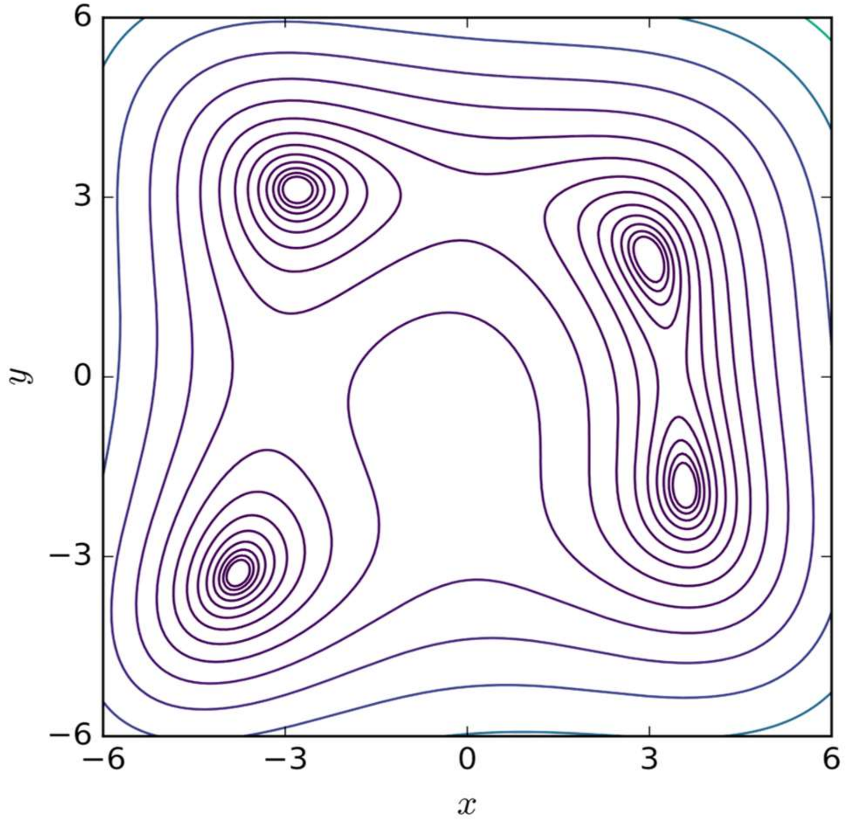
```
pos.xy = frac(pos.xy)
d = Box(pos)
c = fmod(pos * A, B)
subD = max(c.y,min(c.y,c.z))
subD = min(subD,cylinder(c))
subD = max(subD, Windows())
e = fmod(pos + N, M)
floorD = Box(e)
floorD = min(floorD,holes())
d = max(d, -subD)
d = max(d, -floorD)
```

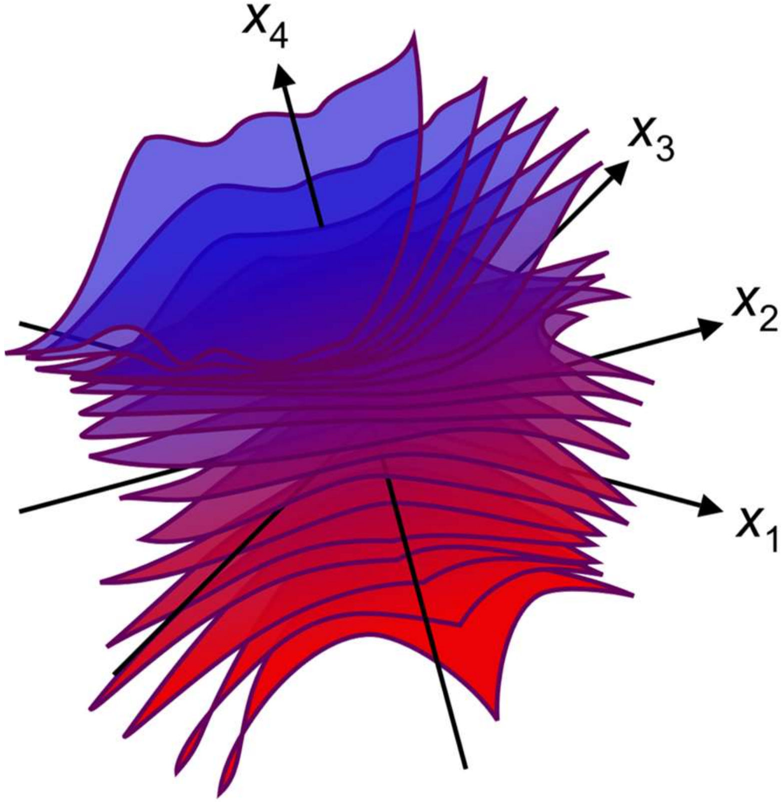


Shading

Distance Fields Contain Infinite Level Sets

- A level set is a set where the function takes on a given constant value *c*



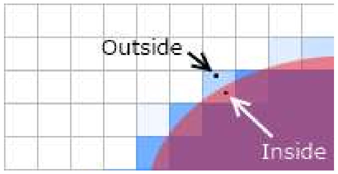


Gradient of a Level Set

Fog

Anti-aliasing

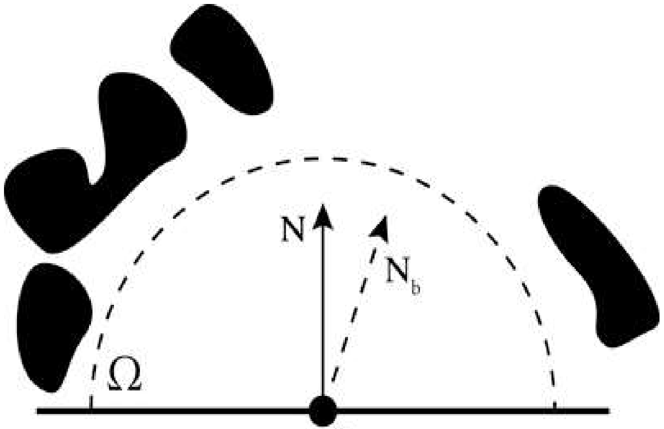
- An interesting optimization could be to only perform AA if the iteration count is above average, as it likely indicates the edge of an object.



Fake Soft Shadows

Ambient Occlusion (AO)

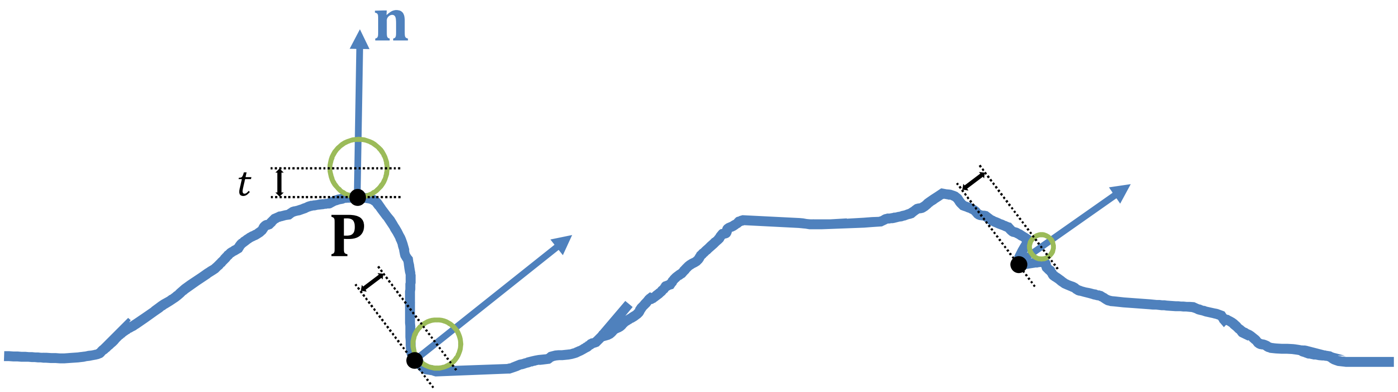
- Cheap approximation of global illumination
- % of hemisphere that is blocked
- Integrate binary visibility function V



$$A_p = \frac{1}{\pi}\int_{\Omega} V_{p,\omega}(N \cdot \omega)\, d\omega$$

Approximate AO with Distance Fields

- Sample distance field along normal
- if $t < dist(\mathbf{P} + t\mathbf{n})$ some occlusion is present
  - Occlusion proportional to $t - dist(\mathbf{P} + t\mathbf{n})$



AO with Distance Fields

- Sample distance field along normal
- if $t < dist(\mathbf{P} + t\mathbf{n})$ some occlusion is present
  - Occlusion proportional to $t - dist(\mathbf{P} + t\mathbf{n})$
  - Repeat for a number of samples
  - Apply weighted sum or other combination