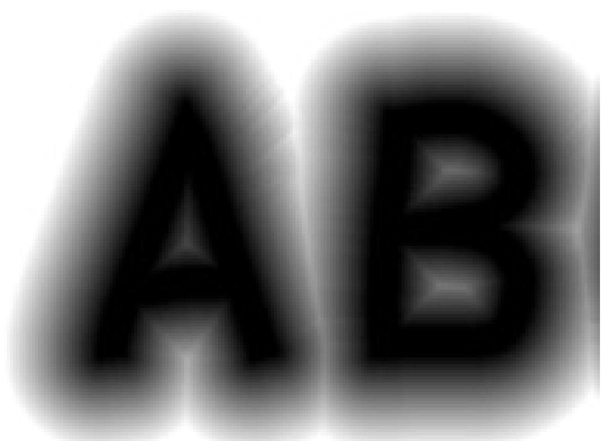
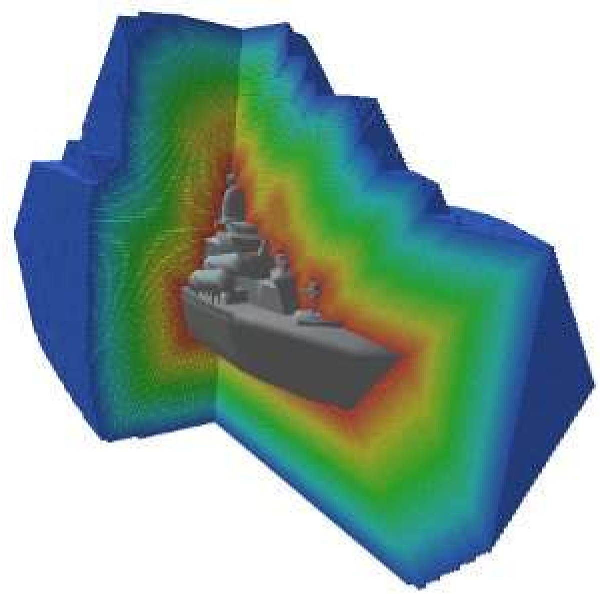
Sphere Tracing Distance Fields

Distance Fields

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

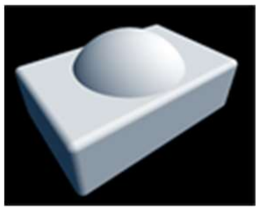


$\mathbb{R}^3 \to dist(\mathbb{R}^3)$

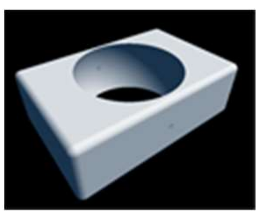
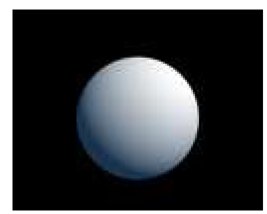


Operations on Distance Fields

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

Operations on Distance Fields

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

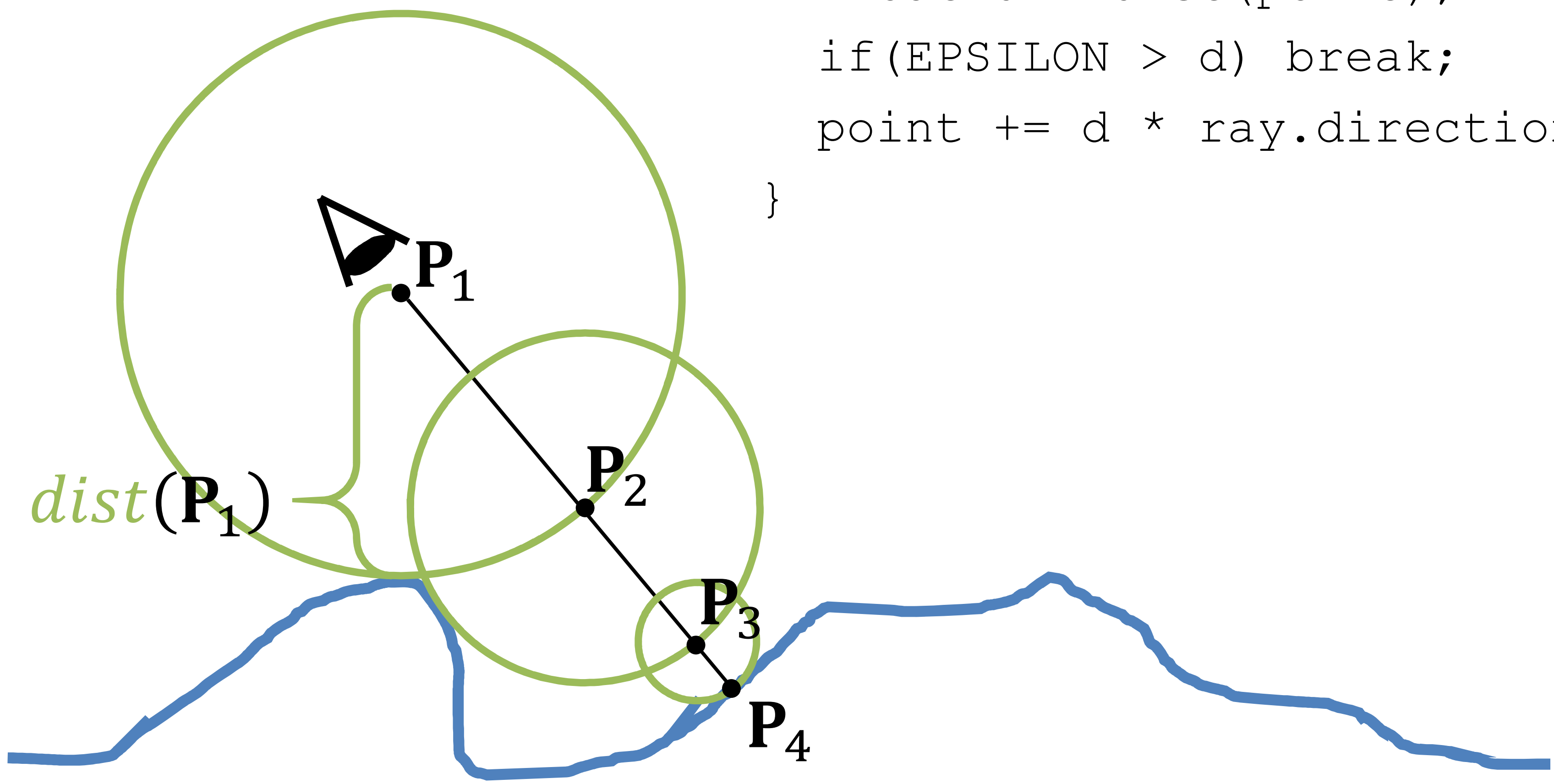
 $= dist(repeat(\mathbb{R}^3))$

- 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 $\vec{\mathbf{c}}$

Sphere Tracing Distance Fields

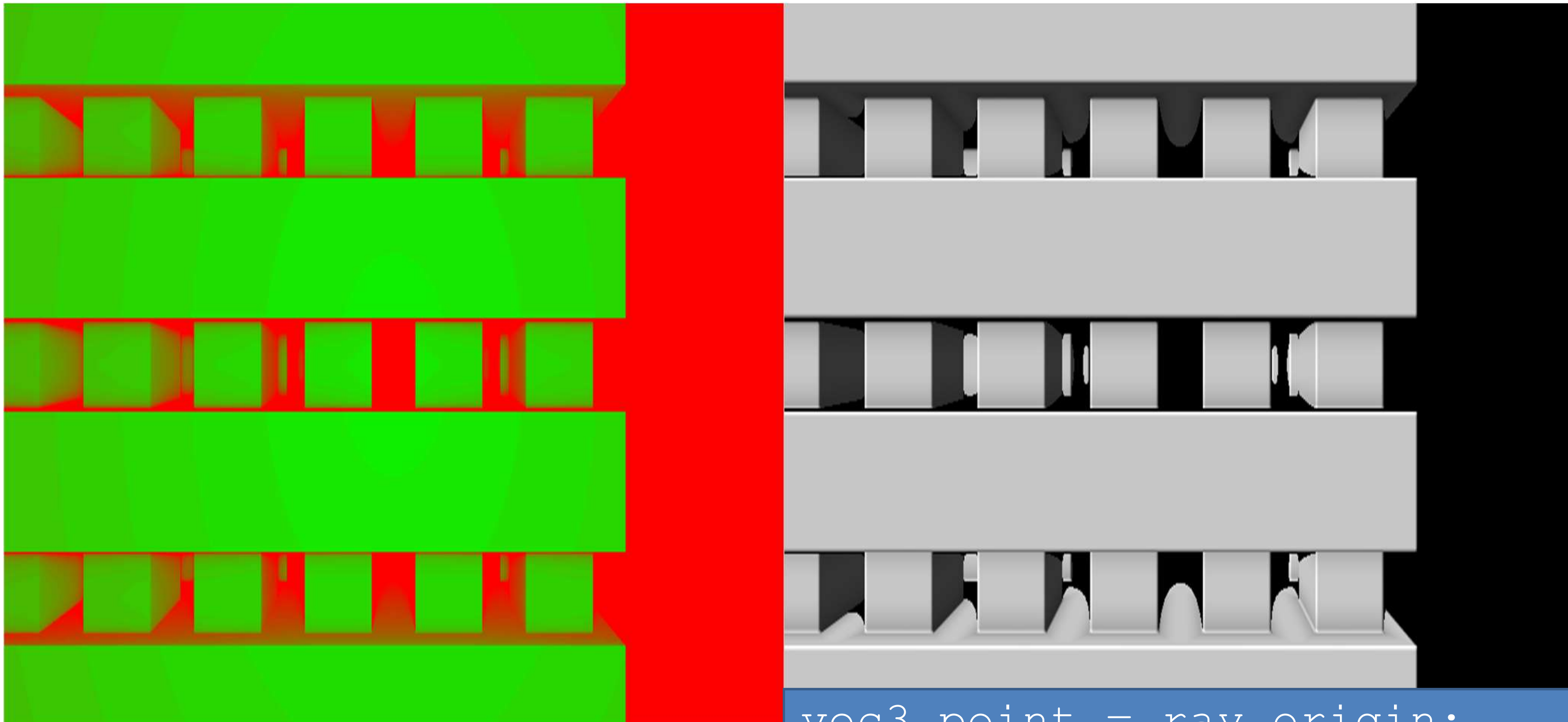
- $dist(\mathbf{P}_i)$

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



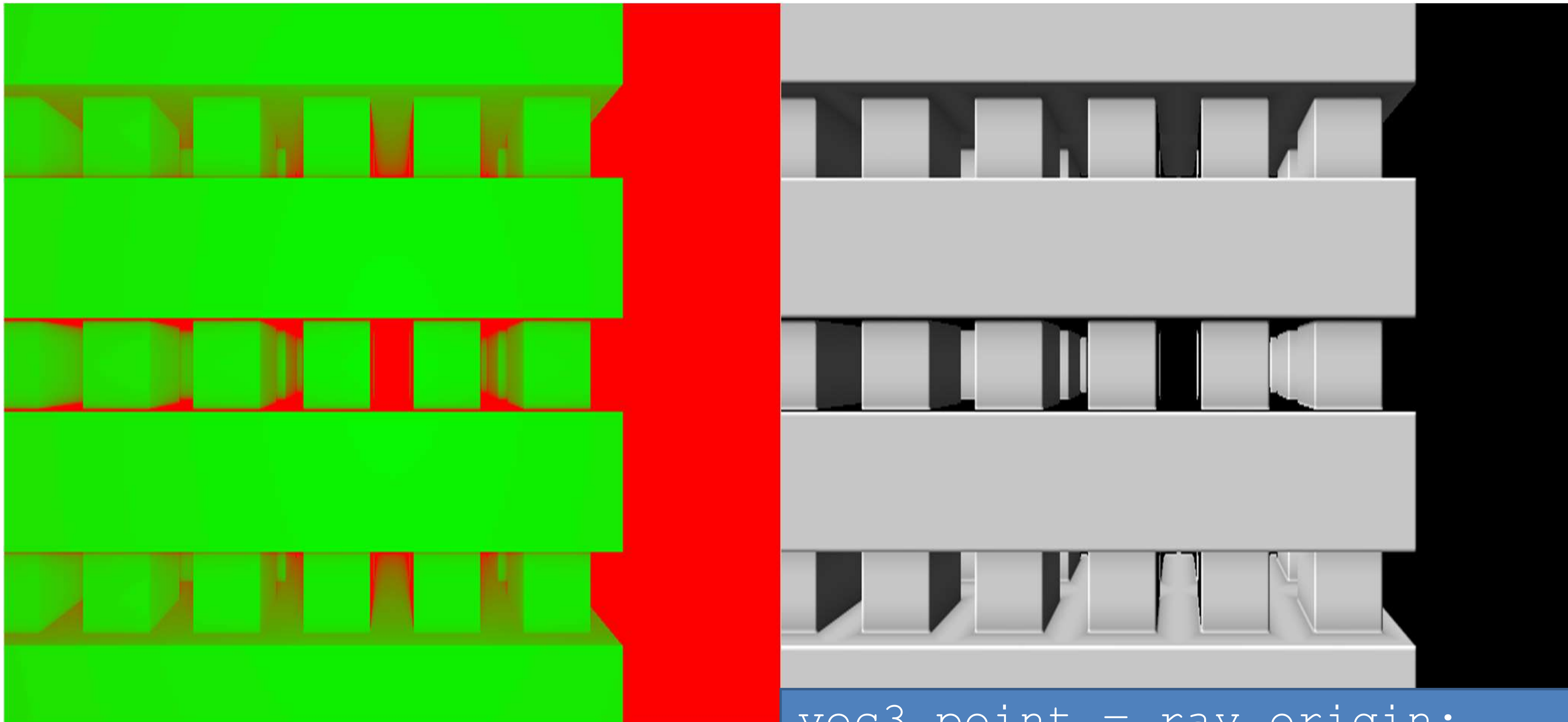
Performance vs Quality

maxSteps = 32



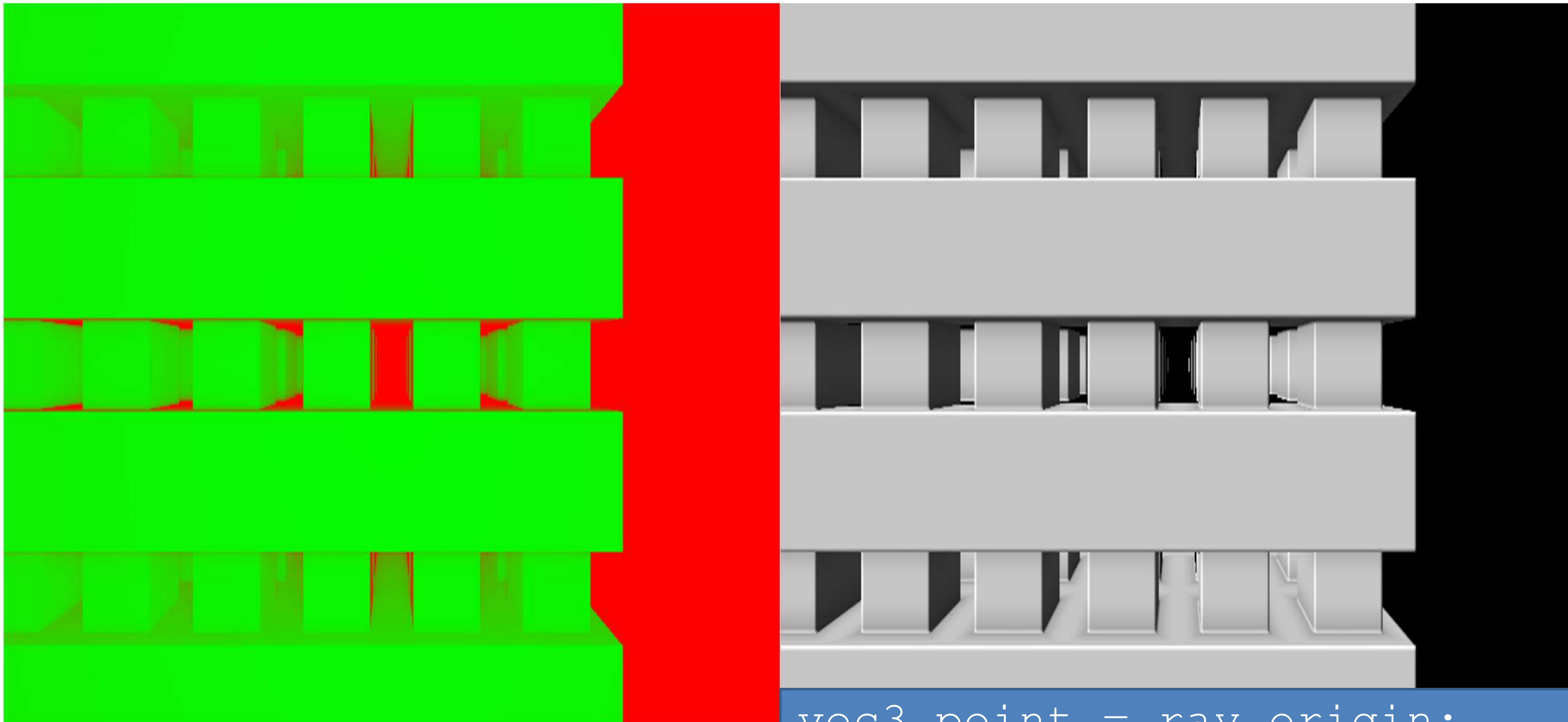
```
vec3 point = ray.origin;
while(--maxSteps) {
  float d = dist(point);
  if(EPSILON > d) break;
  point += d * ray.direction;
```

maxSteps = 64



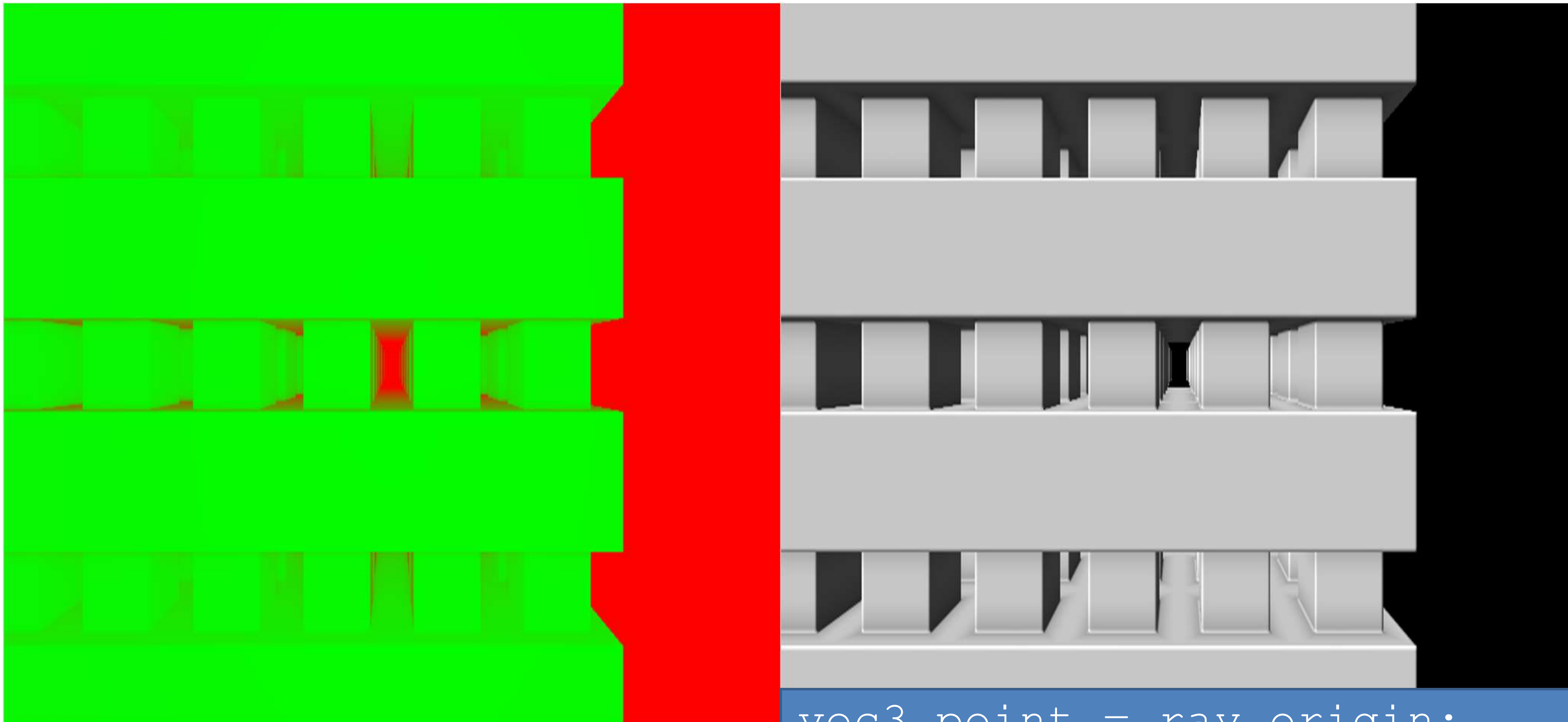
```
vec3 point = ray.origin;
while(--maxSteps) {
  float d = dist(point);
  if(EPSILON > d) break;
  point += d * ray.direction;
```

maxSteps = 128



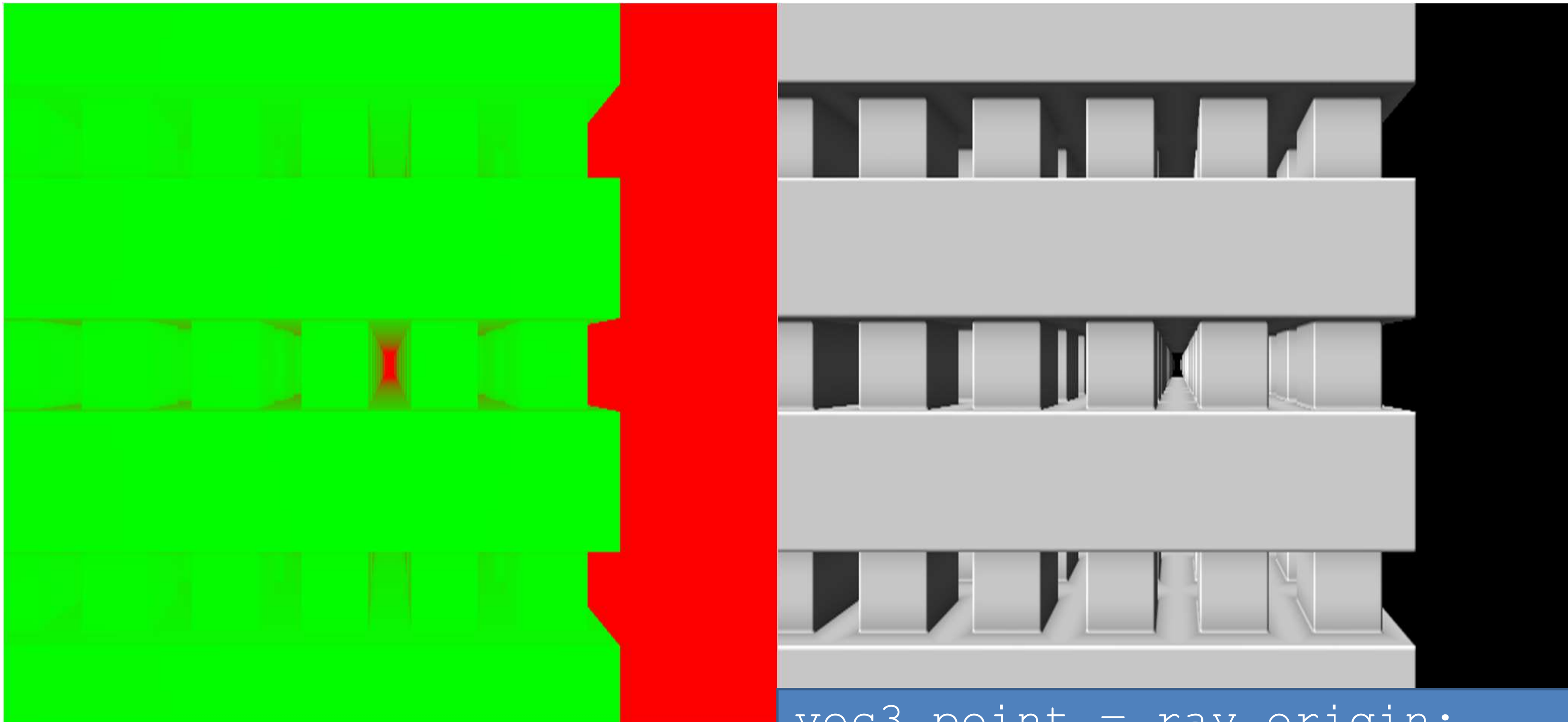
```
vec3 point = ray.origin;
while(--maxSteps) {
  float d = dist(point);
  if(EPSILON > d) break;
  point += d * ray.direction;
```

maxSteps = 256



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

maxSteps = 512



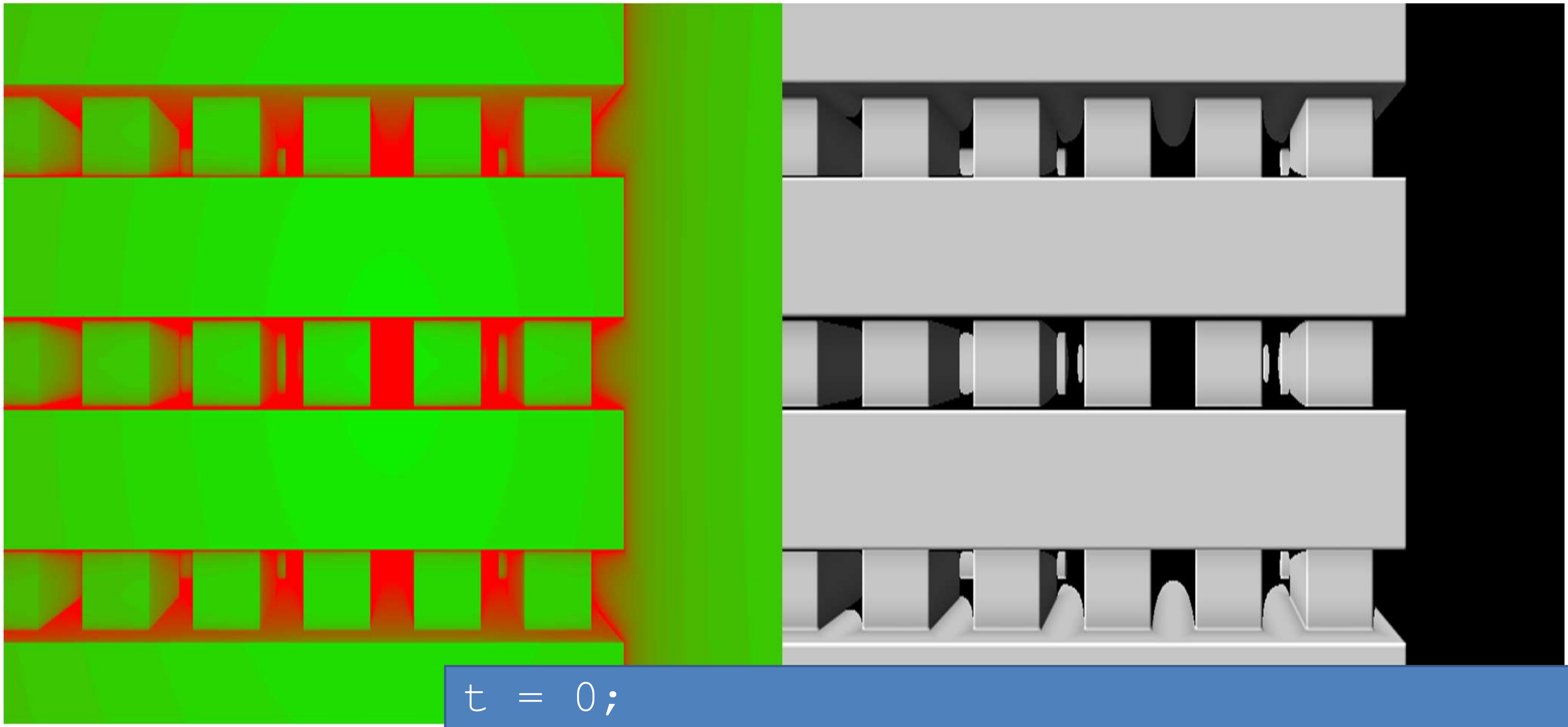
```
vec3 point = ray.origin;
while(--maxSteps) {
  float d = dist(point);
  if(EPSILON > d) break;
  point += d * ray.direction;
```

Maximum Distance

- Scene dependent

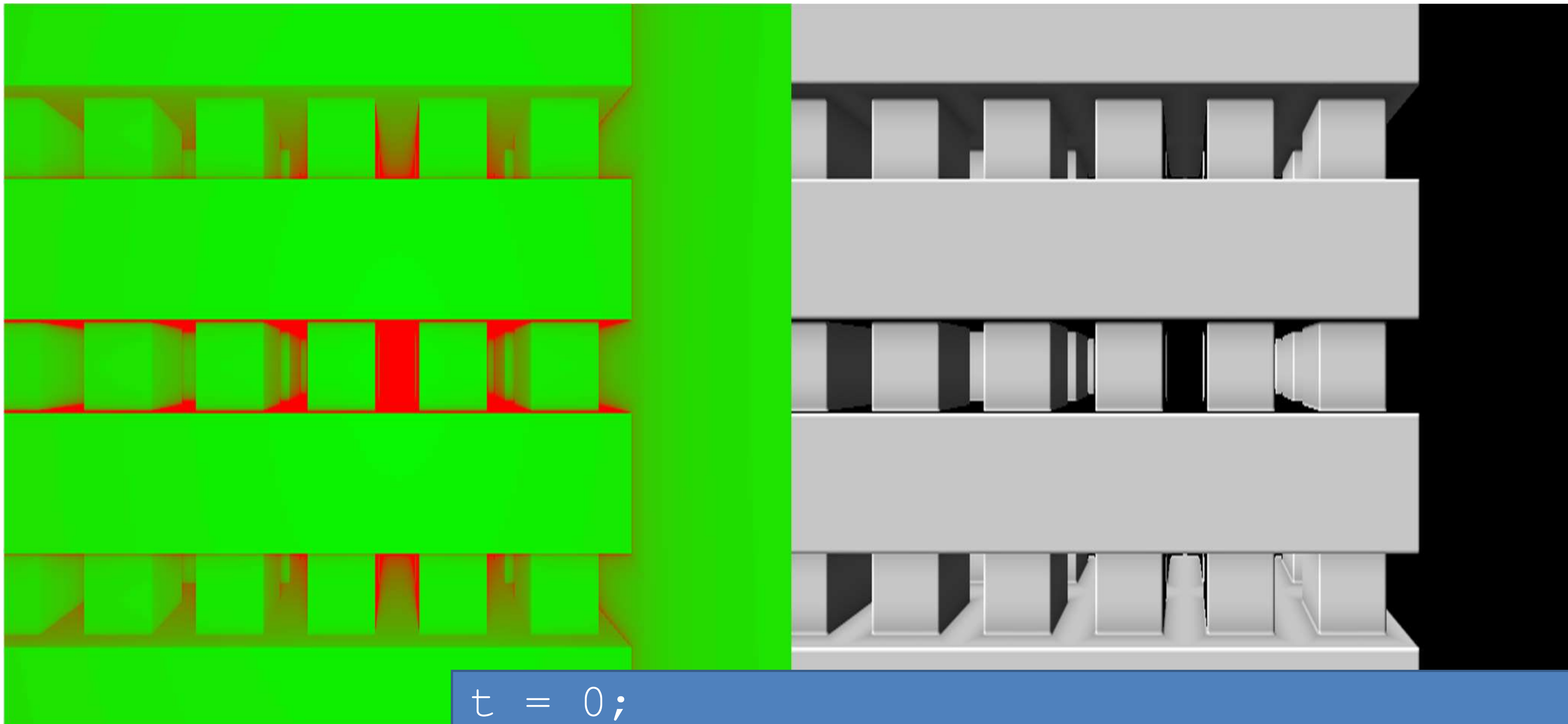
```
t = 0;
while(--maxSteps && (t < maxDistance)) {
  d = dist(point);
  if(EPSILON > d) break;
  t += d;
  point = ray.origin + t * ray.direction;
```

maxSteps = 32



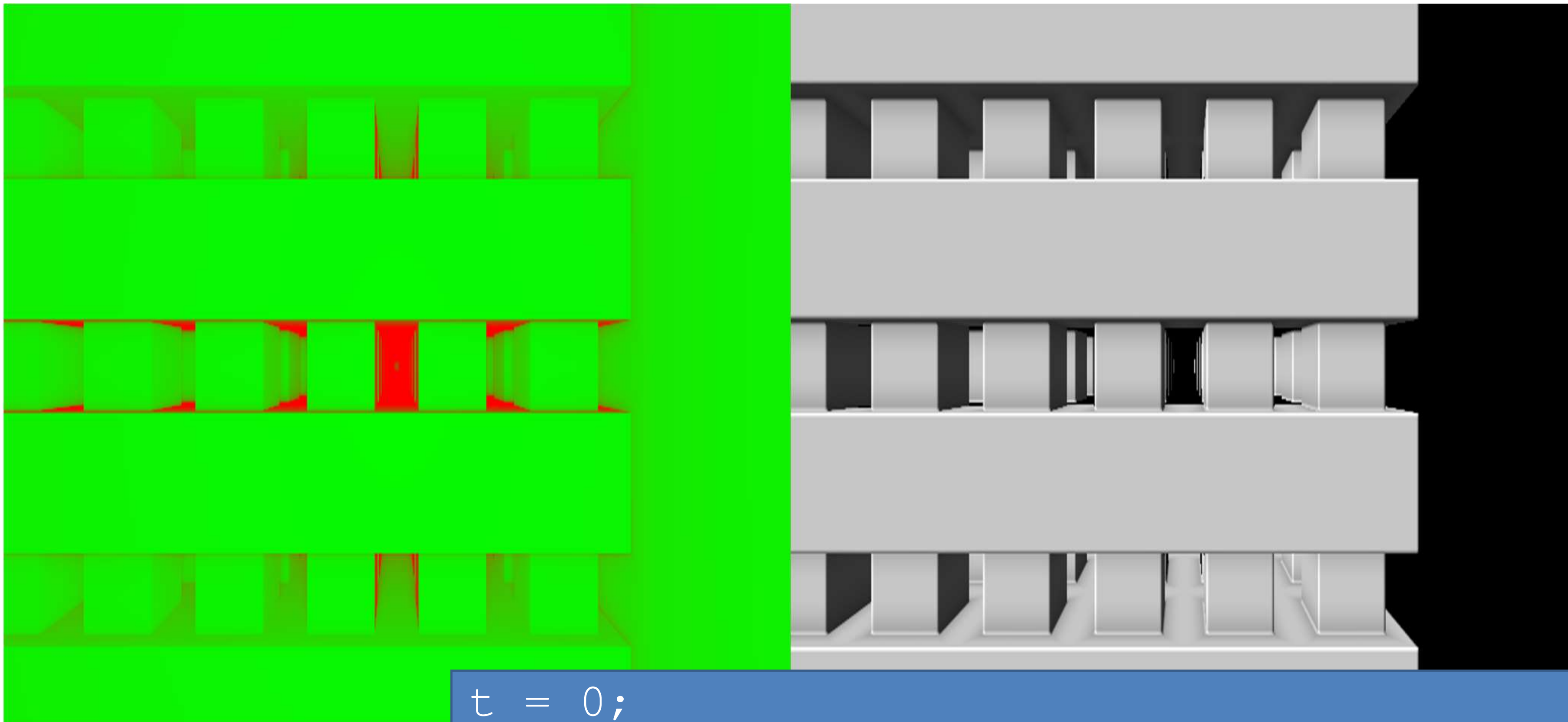
```
t = 0;
while(--maxSteps && (t < maxDistance)) {
  d = dist(point);
  if(EPSILON > d) break;
  t += d;
  point = ray.origin + t * ray.direction;
```

maxSteps = 64



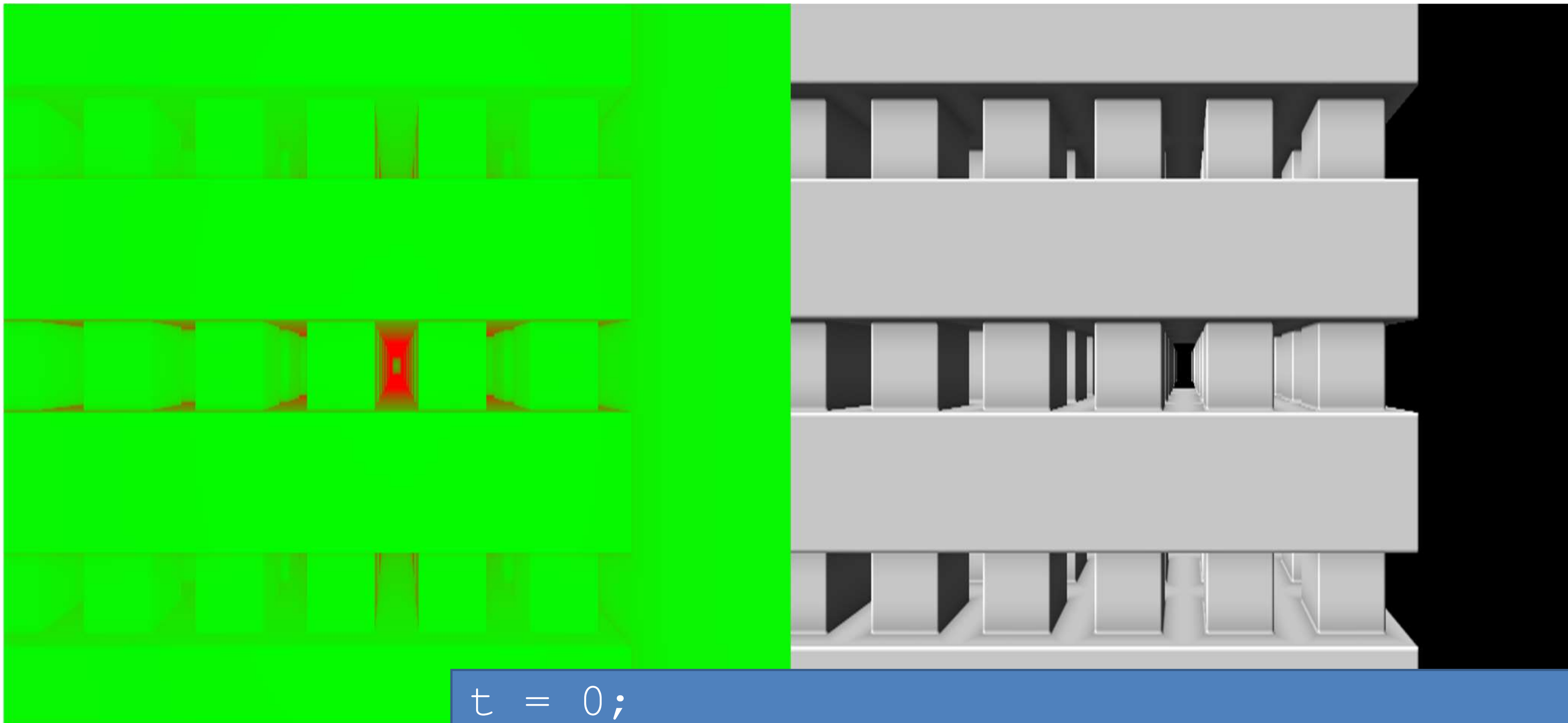
```
t = 0;
while(--maxSteps && (t < maxDistance)) {
  d = dist(point);
  if(EPSILON > d) break;
  t += d;
  point = ray.origin + t * ray.direction;
```

maxSteps = 128



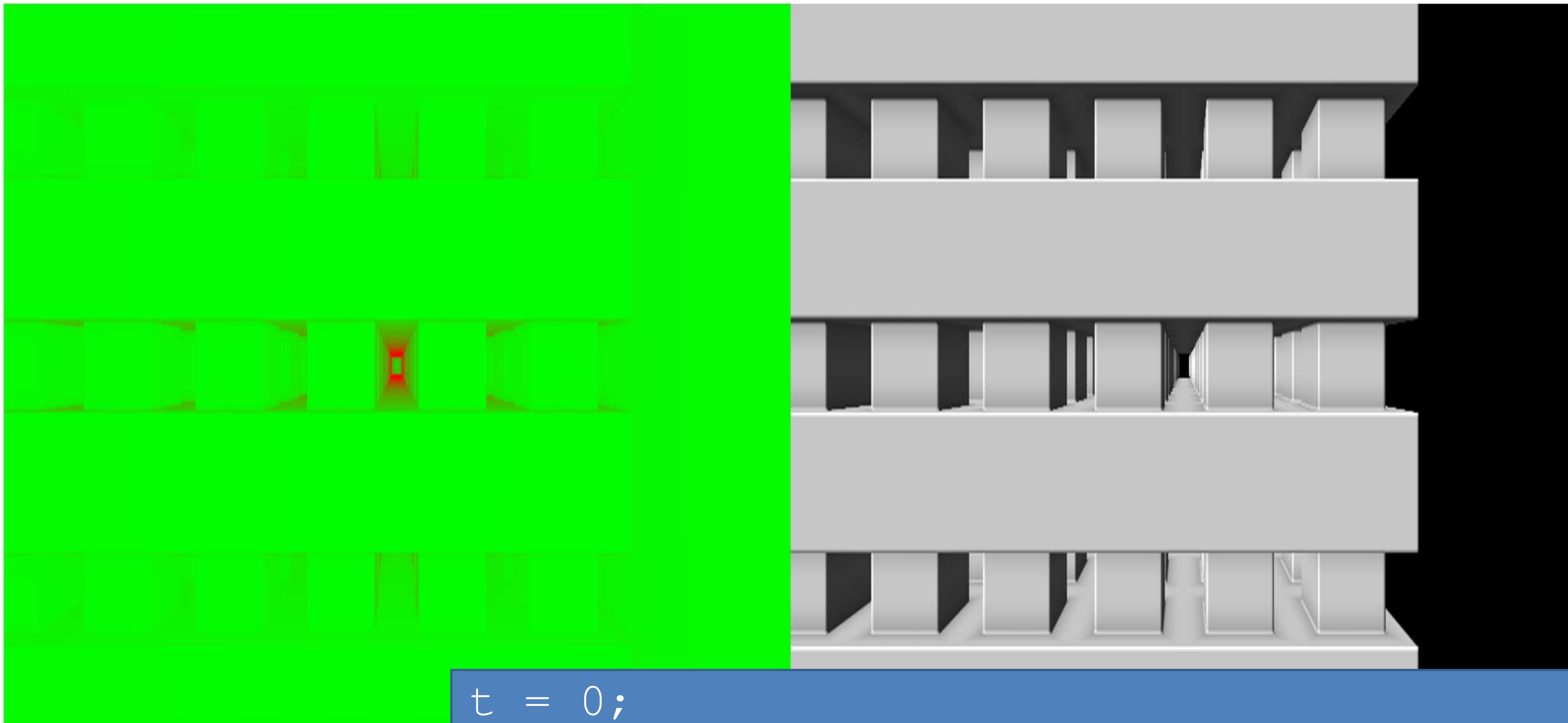
```
t = 0;
while(--maxSteps && (t < maxDistance)) {
  d = dist(point);
  if(EPSILON > d) break;
  t += d;
  point = ray.origin + t * ray.direction;
```

maxSteps = 256



```
t = 0;
while(--maxSteps && (t < maxDistance)) {
  d = dist(point);
  if(EPSILON > d) break;
  t += d;
  point = ray.origin + t * ray.direction;
```

maxSteps = 512



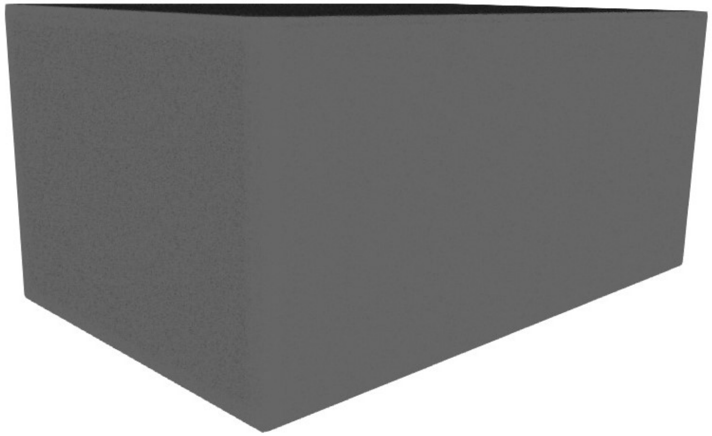
```
t = 0;
while(--maxSteps && (t < maxDistance)) {
  d = dist(point);
  if(EPSILON > d) break;
  t += d;
  point = ray.origin + t * ray.direction;
```

Links

- Overview
  9bitscience.blogspot.de/2013/07/raymarching-distance-fields_14.html
- Distance functions
  www.iquilezles.org/www/articles/distfunctions/distfunctions.htm

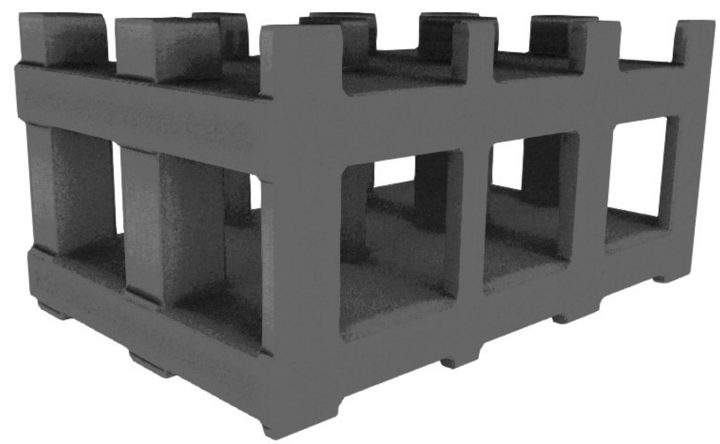
A Box

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



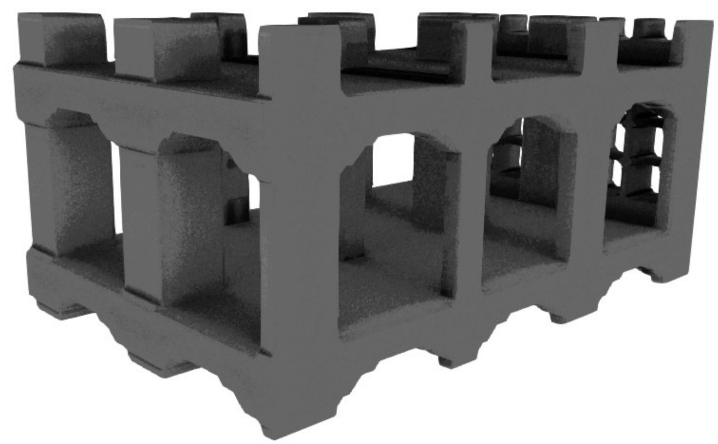
Cutting with Booleans

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



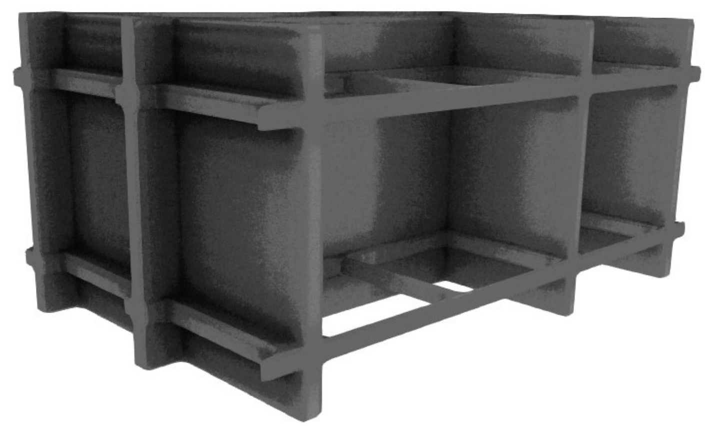
More Booleans

```
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)
```



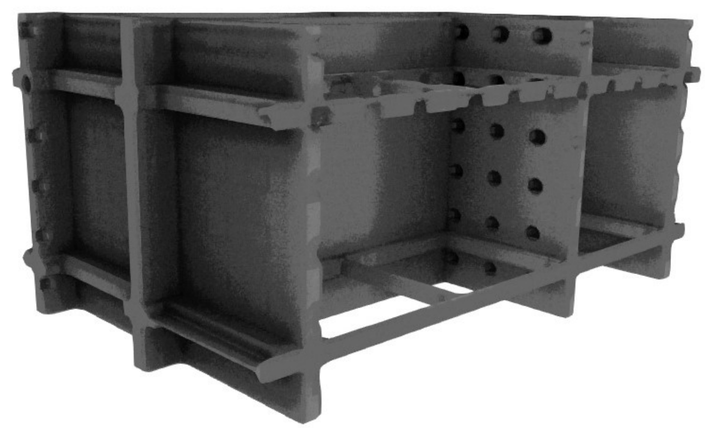
Repeated Booleans

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



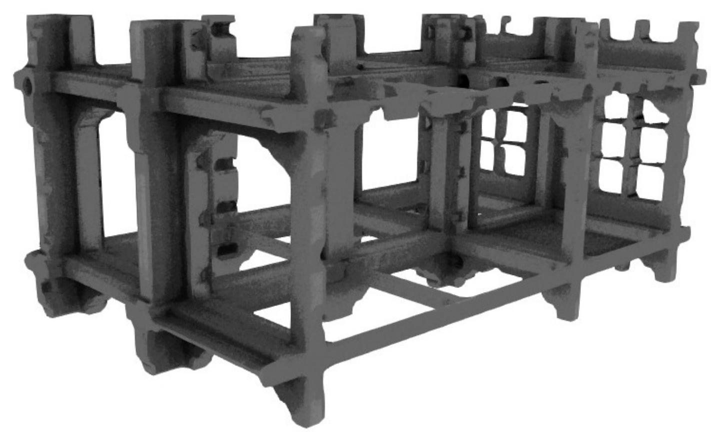
Cutting Holes

```
d = Box(pos)

e = fmod(pos + N, M)

floorD = Box(e)

floorD = min(floorD,holes())

d = max(d, -floorD)
```



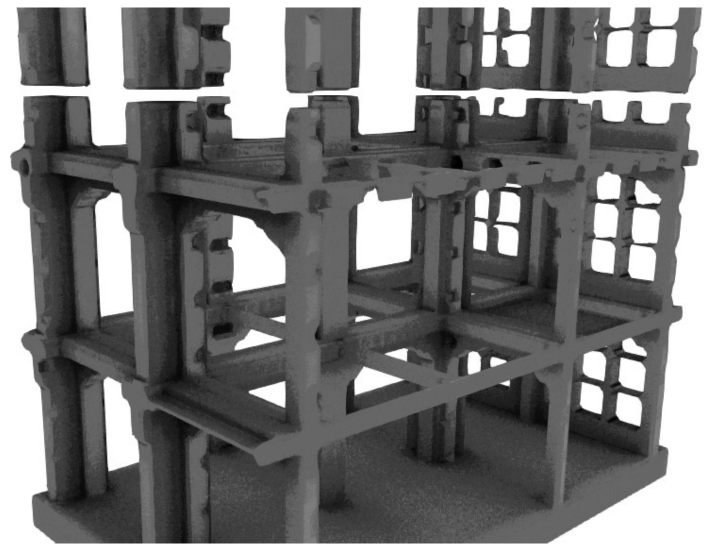
Combined Result

```
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)
```



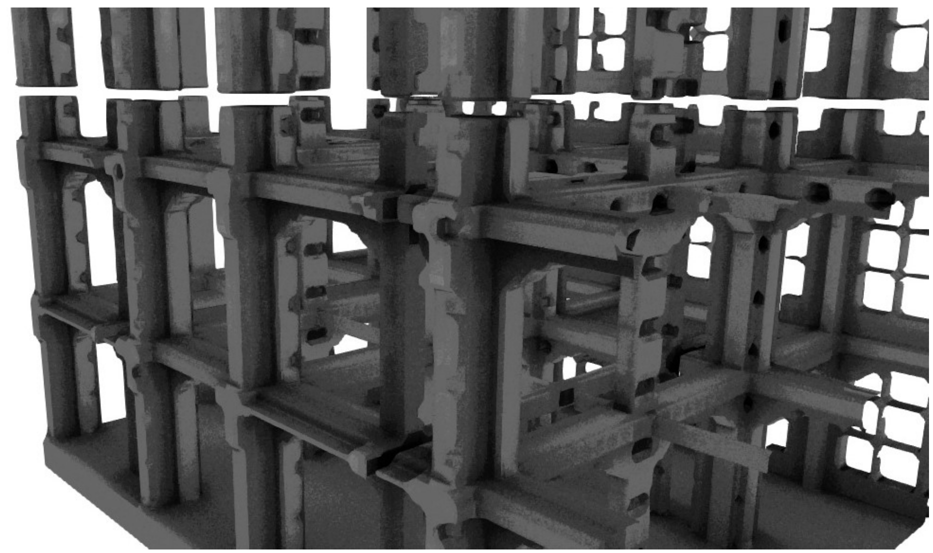
Repeating the Space

```
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

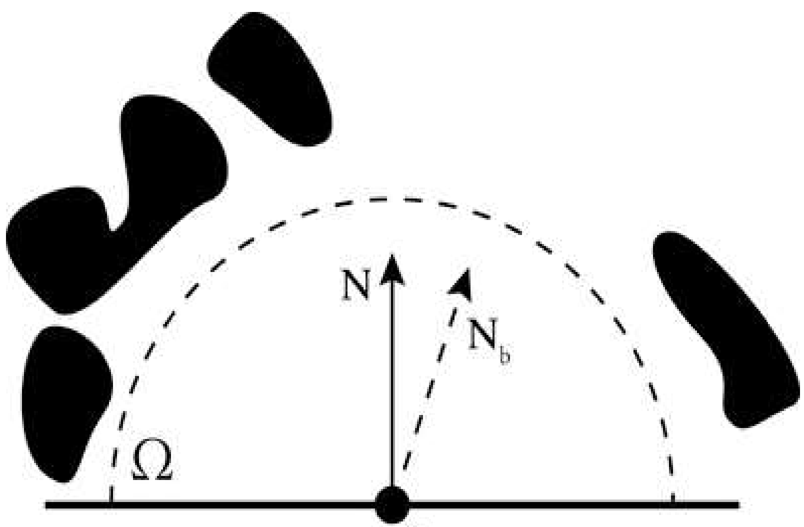
```
pos.xy = frac(pos.xy)
d = Box(pos)
c = fmod(pos * A, B)
subD = max(c.y,min(c.y,c.z))
subD = min(subD,cylinder(c))
subD = max(subD, Windows())
e = fmod(pos + N, M)
floorD = Box(e)
floorD = min(floorD,holes())
d = max(d, -subD)
d = max(d, -floorD)
```



Shading

Ambient Occlusion (AO)

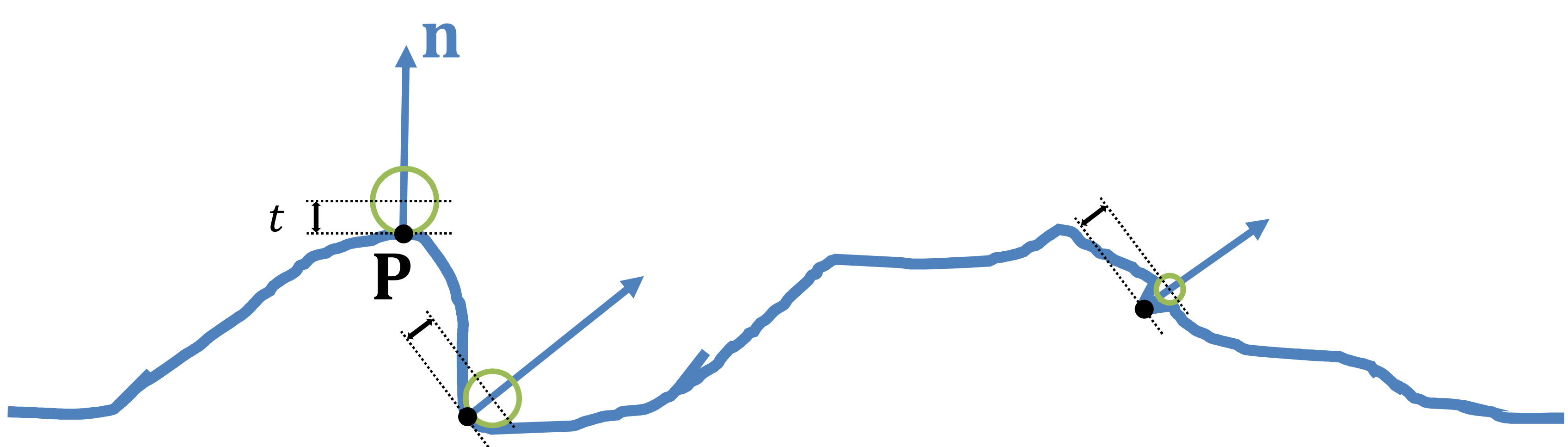
- Cheap approximation of global illumination
- % of hemisphere that is blocked
- Integrate binary visibility function V



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

AO with Distance Fields

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



AO with Distance Fields

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