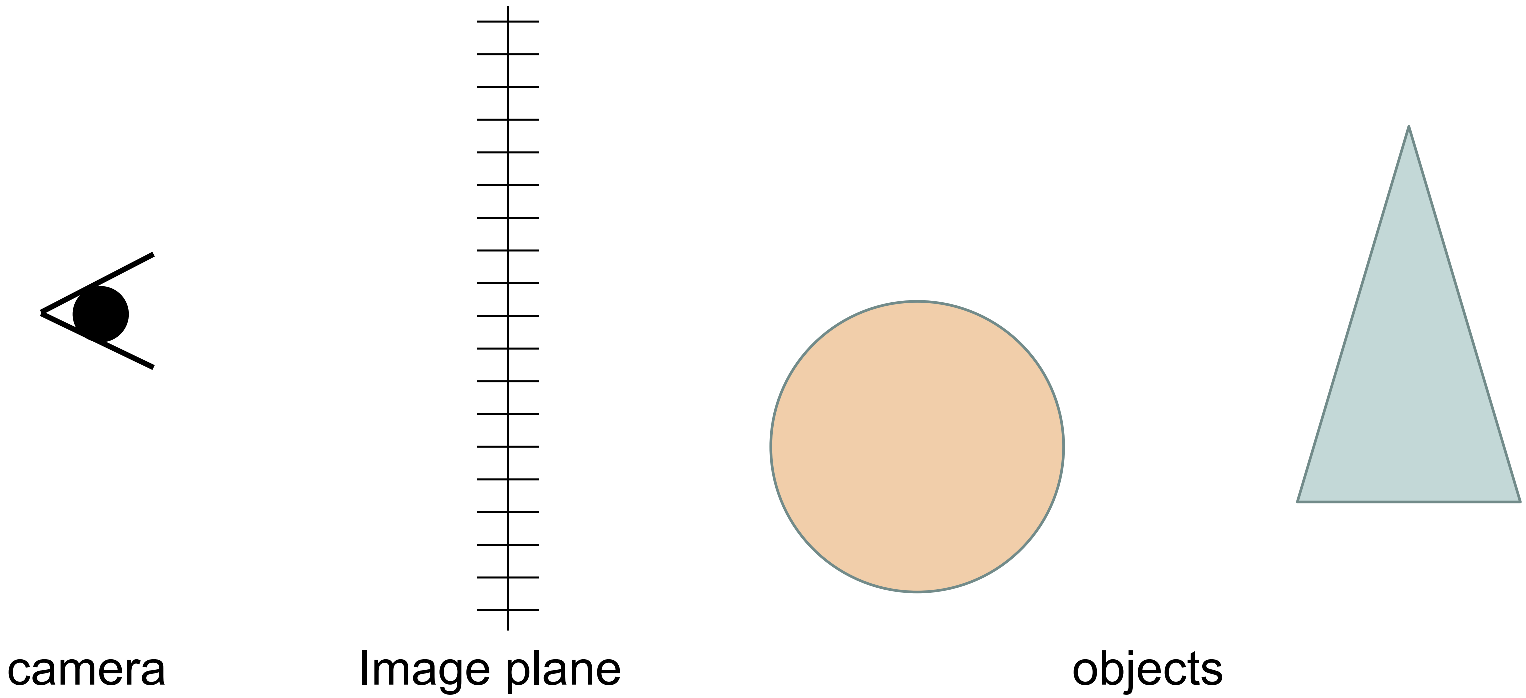
Advanced image synthesis

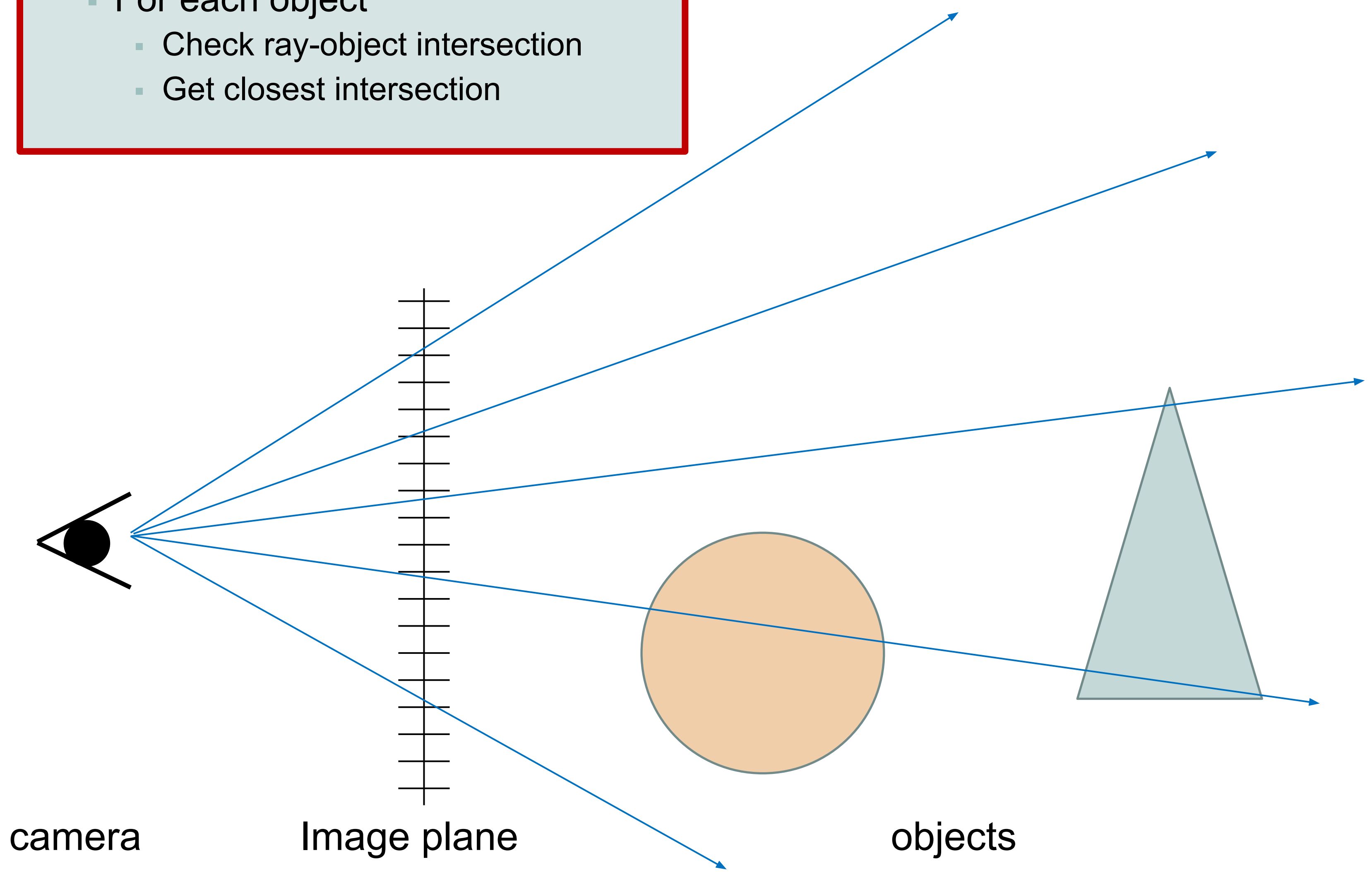
Ray-casting basics

- For each pixel
  - Compute pixel ray
  - For each object
    - Check ray-object intersection
    - Get closest intersection



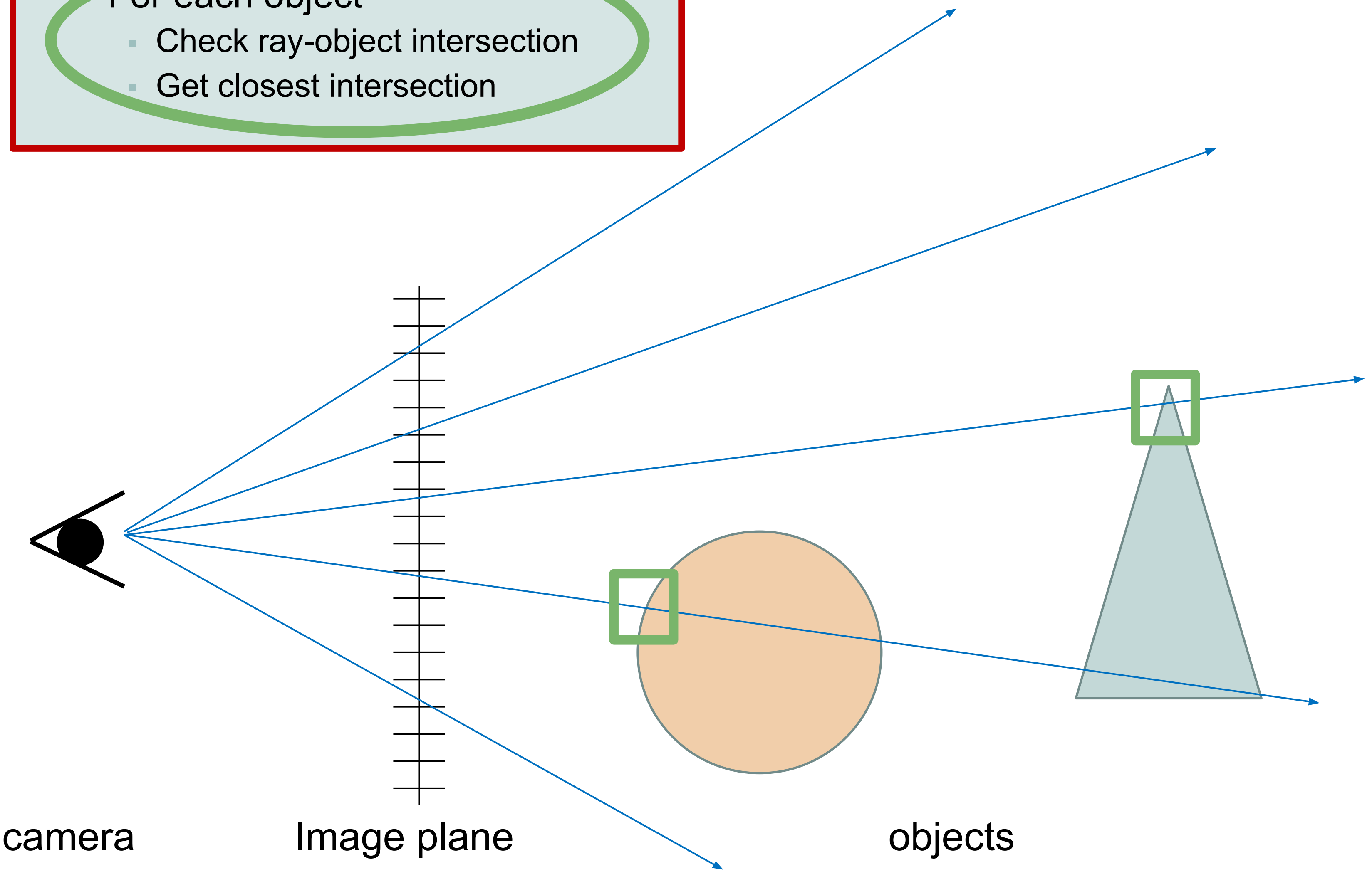
Ray-casting basics

- For each pixel
  - Compute pixel ray
  - For each object
    - Check ray-object intersection
    - Get closest intersection



Ray-casting basics

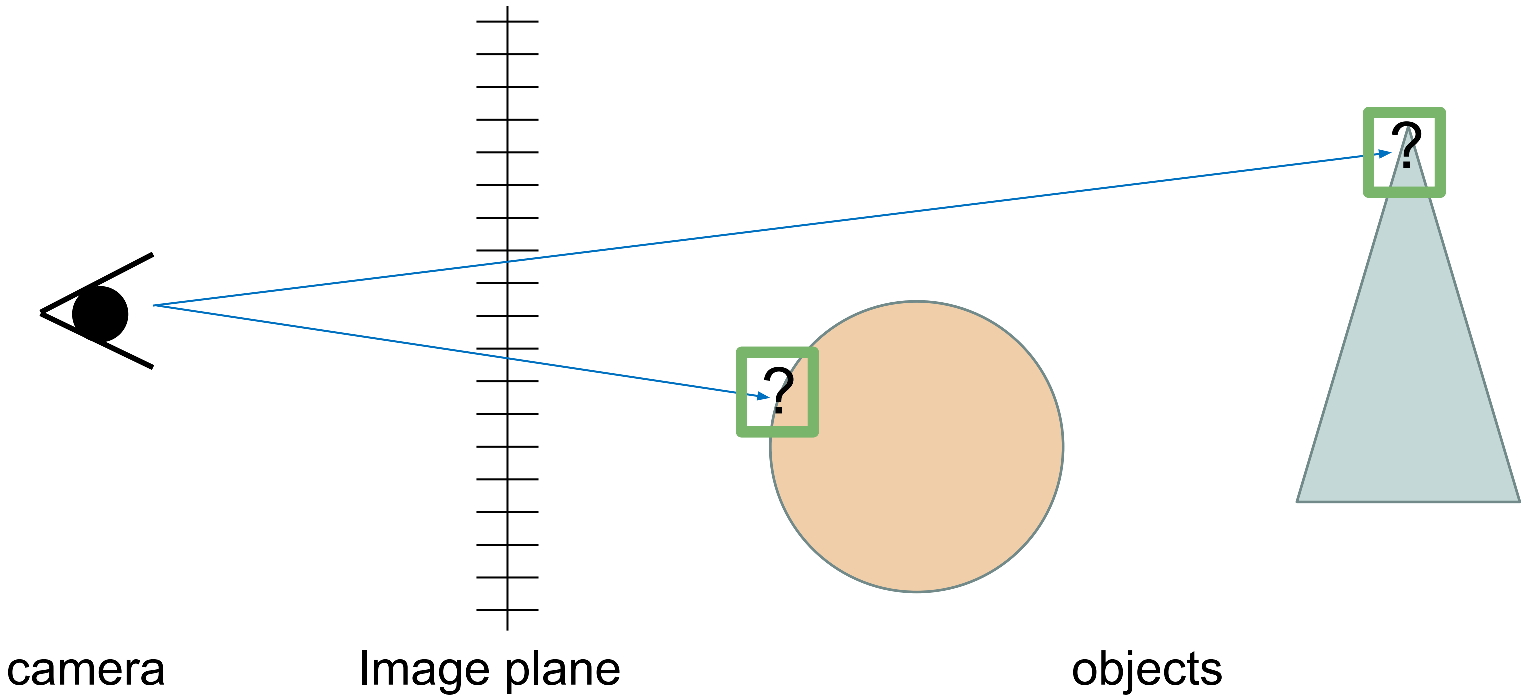
- For each pixel
  - Compute pixel ray
  - For each object
    - Check ray-object intersection
    - Get closest intersection



Ray-casting basics

- For each pixel
  - Compute pixel ray
  - For each object
    - Check ray-object intersection
    - Get closest intersection

And then?

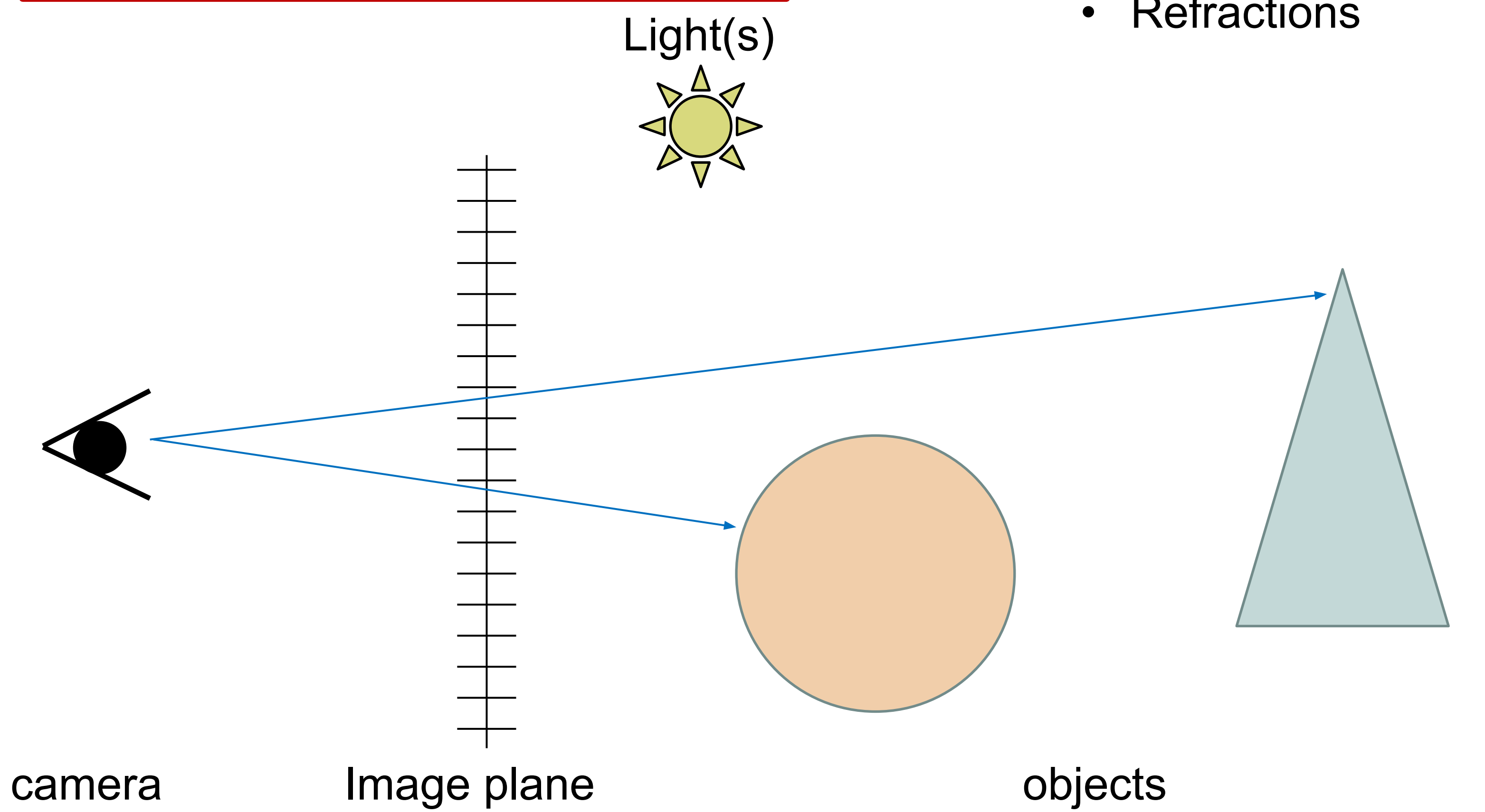


Ray-casting basics

- For each pixel
  - Compute pixel ray
  - For each object
    - Check ray-object intersection
    - Get closest intersection

And then?
Shade!

- Shadow rays
- Reflections
- Refractions



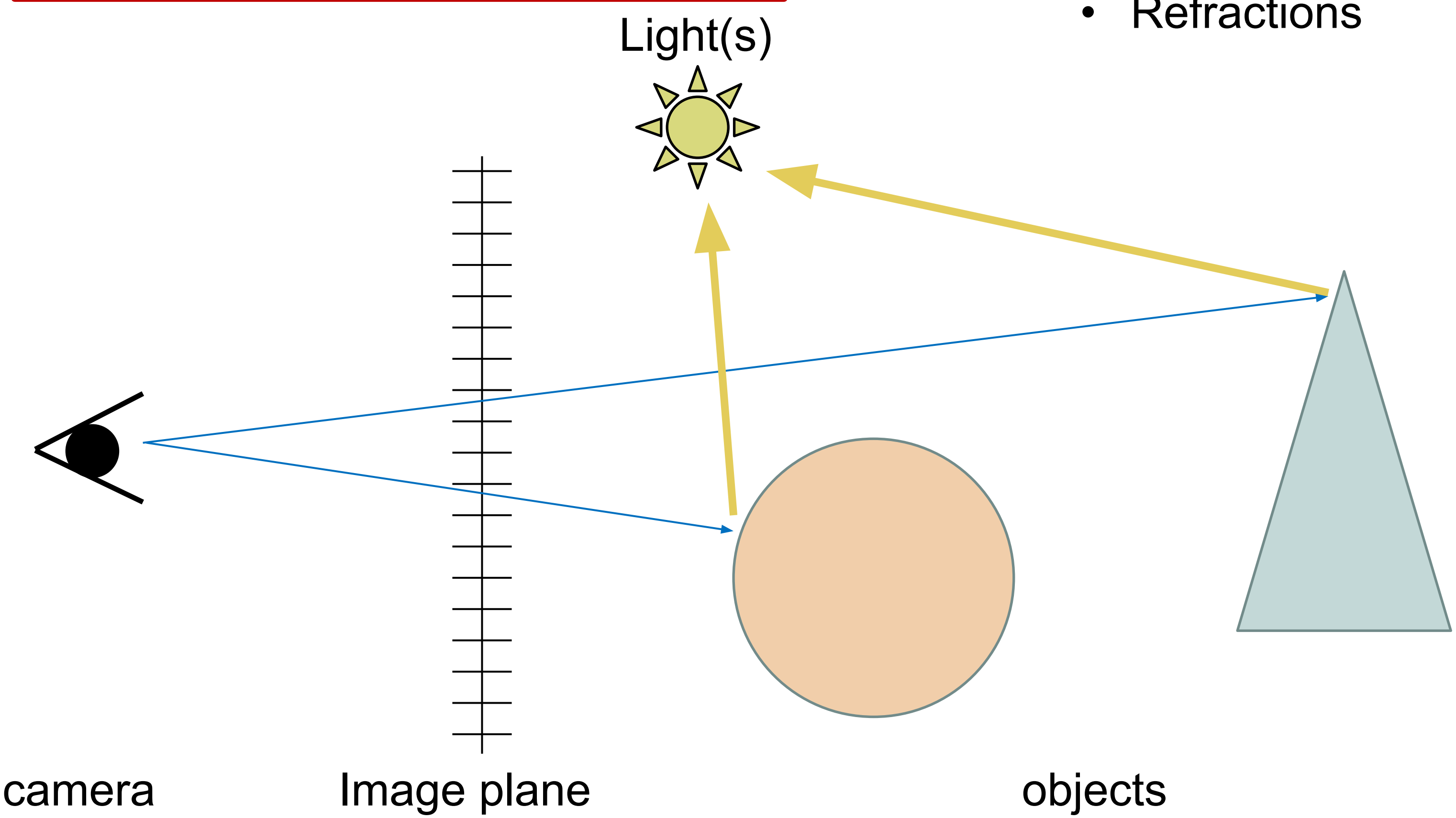
Ray-casting basics

- For each pixel
  - Compute pixel ray
  - For each object
    - Check ray-object intersection
    - Get closest intersection

And then?

Shade!

- Shadow rays
- Reflections
- Refractions

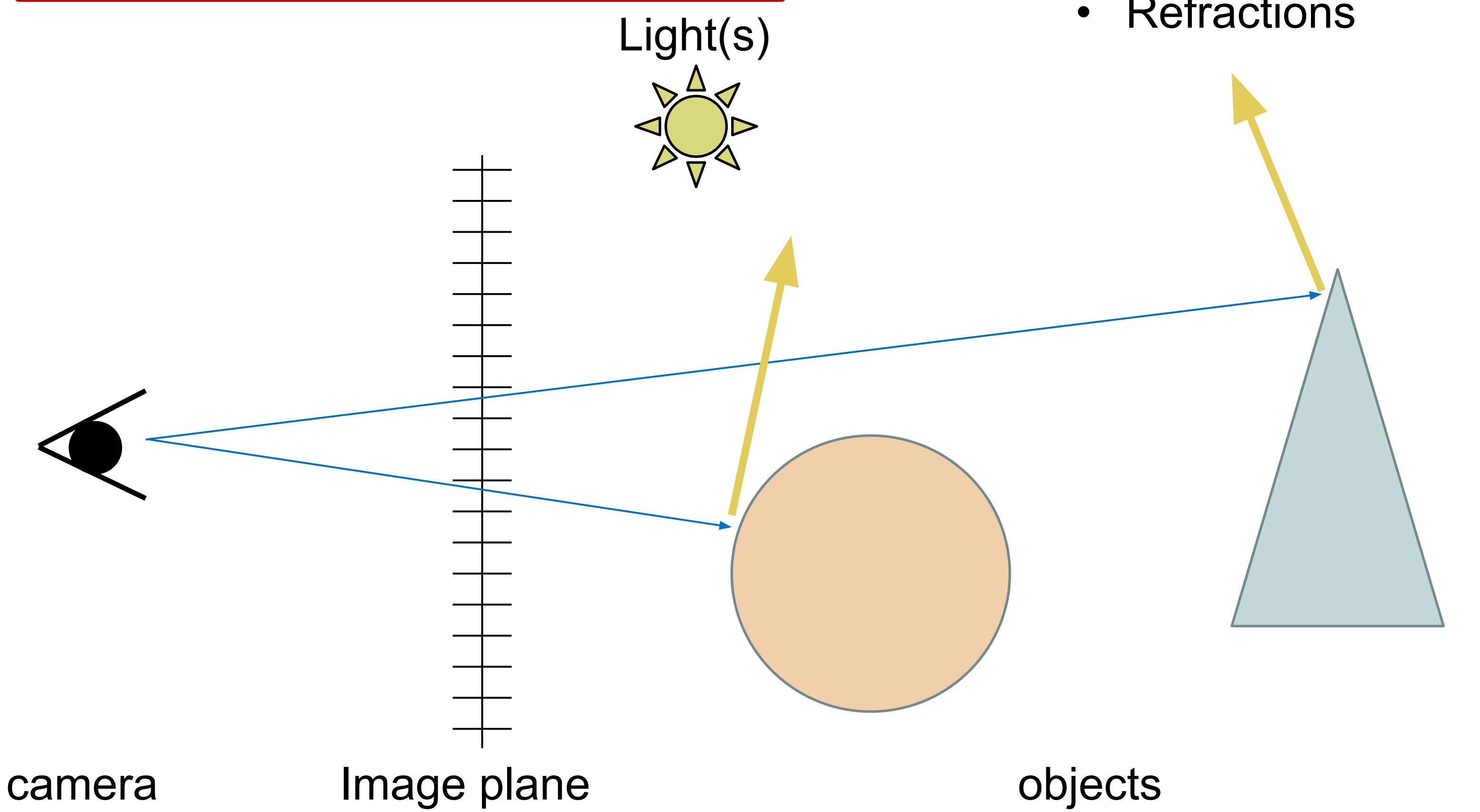


Ray-casting basics

- For each pixel
  - Compute pixel ray
  - For each object
    - Check ray-object intersection
    - Get closest intersection

And then?
Shade!

- Shadow rays
- Reflections
- Refractions

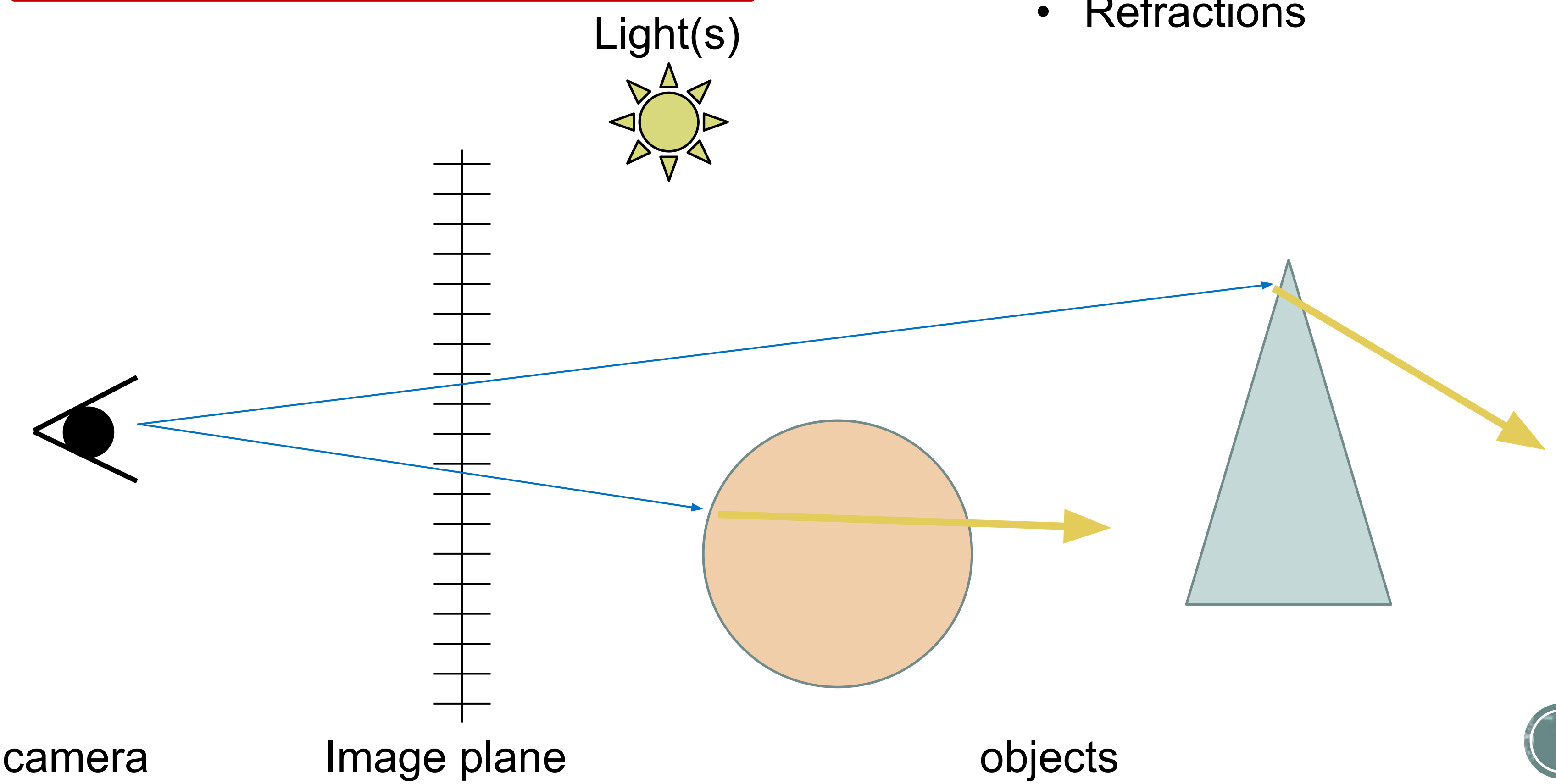


Ray-casting basics

- For each pixel
  - Compute pixel ray
  - For each object
    - Check ray-object intersection
    - Get closest intersection

And then?
Shade!

- Shadow rays
- Reflections
- Refractions



Ray-casting: summary

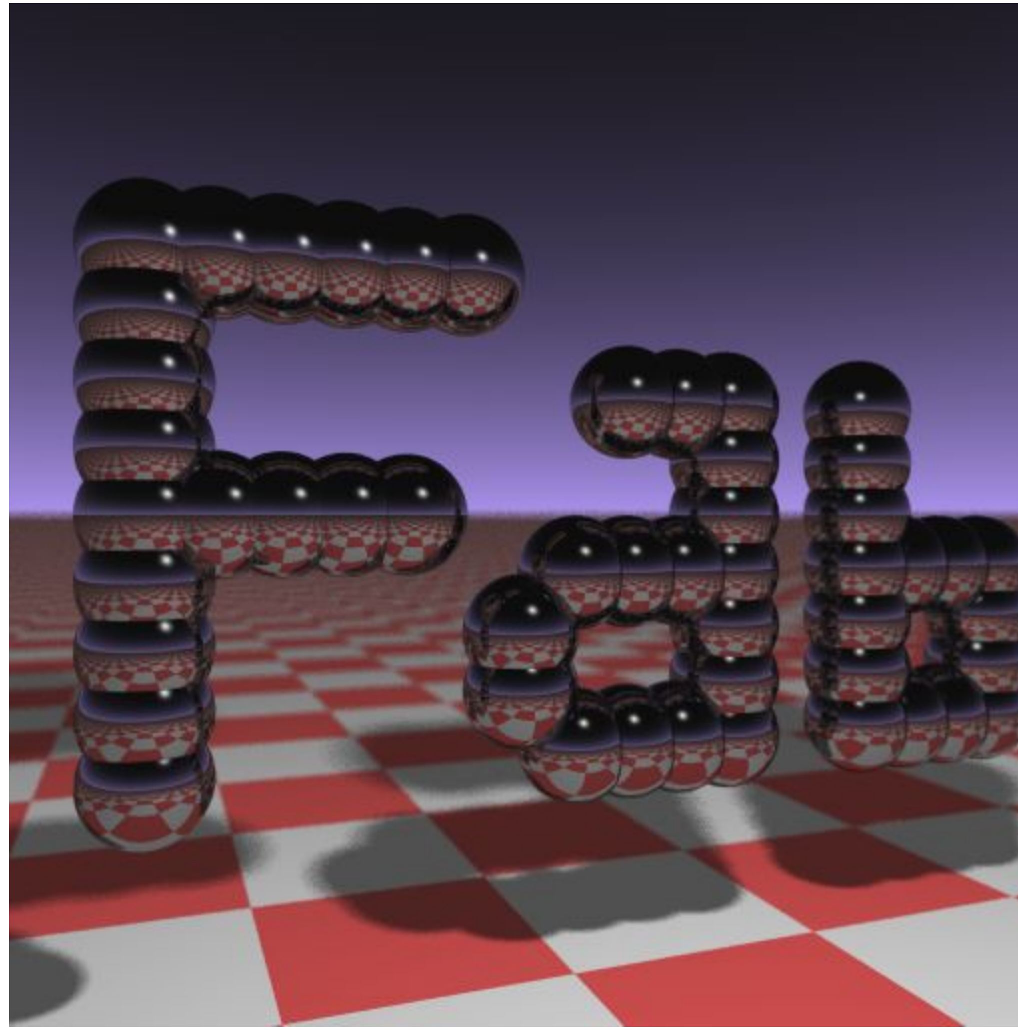
- For each pixel
  - Compute eye ray
  - For each object
    - Check ray-object intersection
    - Get closest intersection
    - Shade depending on light and normal vector

Finding intersection point and normal is the central part of ray-casting!

Ray tracing

Ray tracing

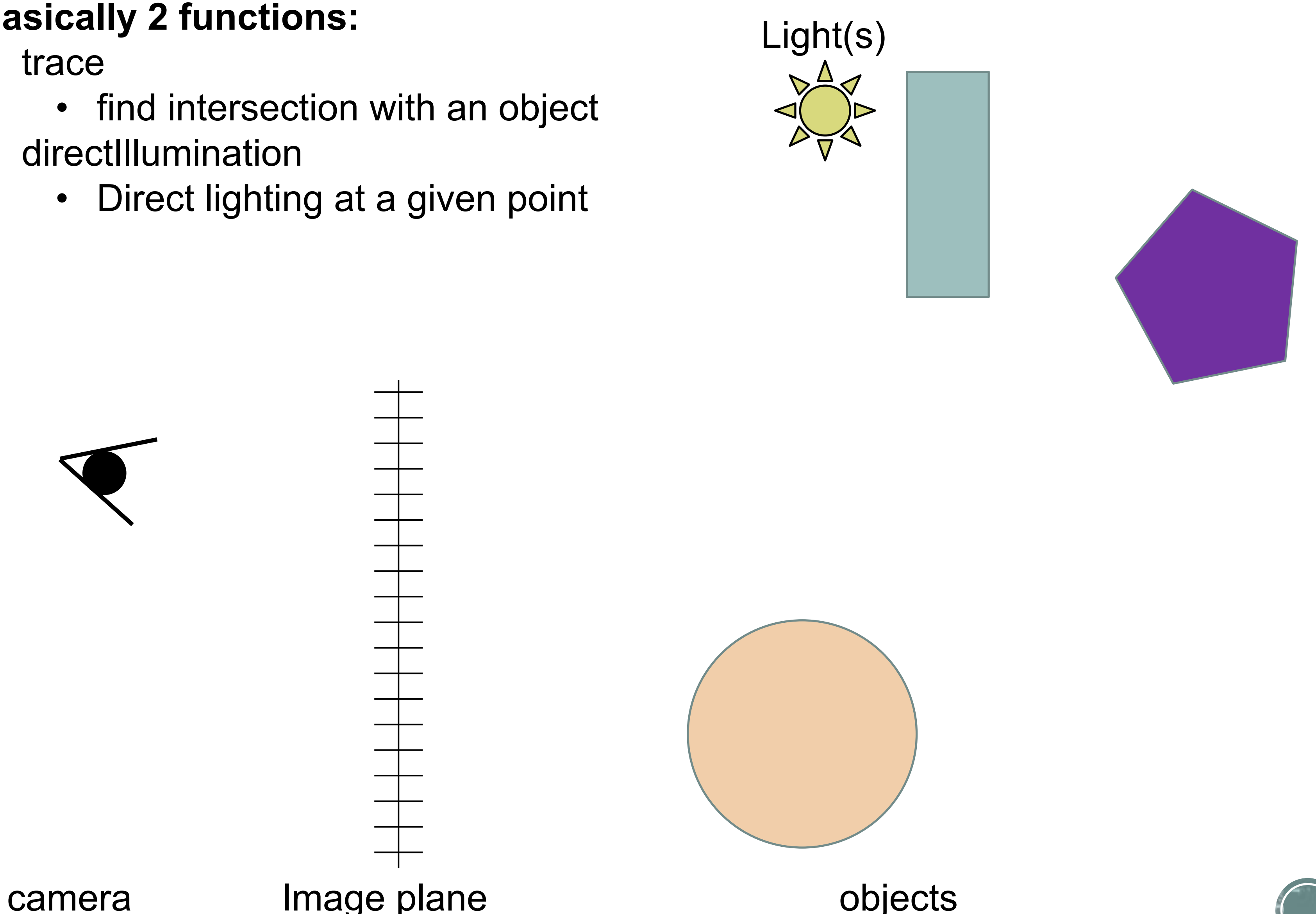
```
#include <stdlib.h>   // card > aek.ppm
#include <stdio.h>
#include <math.h>
typedef int i;typedef float f;struct v{
f x,y,z;v operator+(v r){return v(x+r.x
,y+r.y,z+r.z);}v operator*(f r){ return
v(x*r,y*r,z*r);}f operator%(v r){return
x*r.x+y*r.y+z*r.z;}v(){}v operator^(v r
){return v(y*r.z-z*r.y,z*r.x-x*r.z,x*r.
y-y*r.x);}v(f a,f b,f c){x=a;y=b;z=c;}v
operator!(){return*this*(1/sqrt(*this%*
this));}};i G[]={133022, 133266,133266,
133022, 254096, 131216, 131984, 131072,
258048,};f R(){return(f)rand()/RAND_MAX
;}i T(v o,v d,f&t,v&n){t=1e9;i m=0;f p=
-o.z/d.z; if(.01<p)t=p, n=v(0,0,1),m=1;
for(i k=19;k--;)for(i j=9;j--;)if(G[j]&
1<<k){v p=o+v(-k,0,-j-4);f b=p%d,c=p%p-
1,q=b*b-c;if(q>0){f s=-b-sqrt(q);if(s<t
&&s>.01)t=s,n=!(p+d*t),m=2;}}return m;}
v S(v o,v d){f t;v n;i m=T(o,d,t,n);if(
!m)return v(.7,.6,1)*pow(1-d.z,4);v h=o
+d*t,l=!(v(9+R(),9+R(),16)+h*-1),r=d+n*
(n%d*-2);f b=l%n;if(b<0||T(h,l,t,n))b=0
;f p=pow(l%r*(b>0),99);if(m&1){h=h*.2;
return((i)(ceil(h.x)+ceil(h.y))&1?v(3,1
,1):v(3,3,3))*(b*.2+.1);}return v(p,p,p
)+S(h,r)*.5;}i main(){printf("P6 512  "
"512 255 ");v g=!v(-6,-16,0),a=!(v(0,0,
1)^g)*.002,b=!(g^a)*.002,c=(a+b)*-256+g
;for(i y=512;y--;)for(i x=512;x--;){v p
(9,9,9);for(i r=64;r--;){v t=a*(R()-.5)
*99+b*(R()-.5)*99;p=S(v(17,16,8)+t,!(t*
-1+(a*(R()+x)+b*(y+R())+c)*16))*3.5+p;}
printf("%c%c%c",(i)p.x,(i)p.y,(i)p.z);}}
```



Ray tracing

Basically 2 functions:

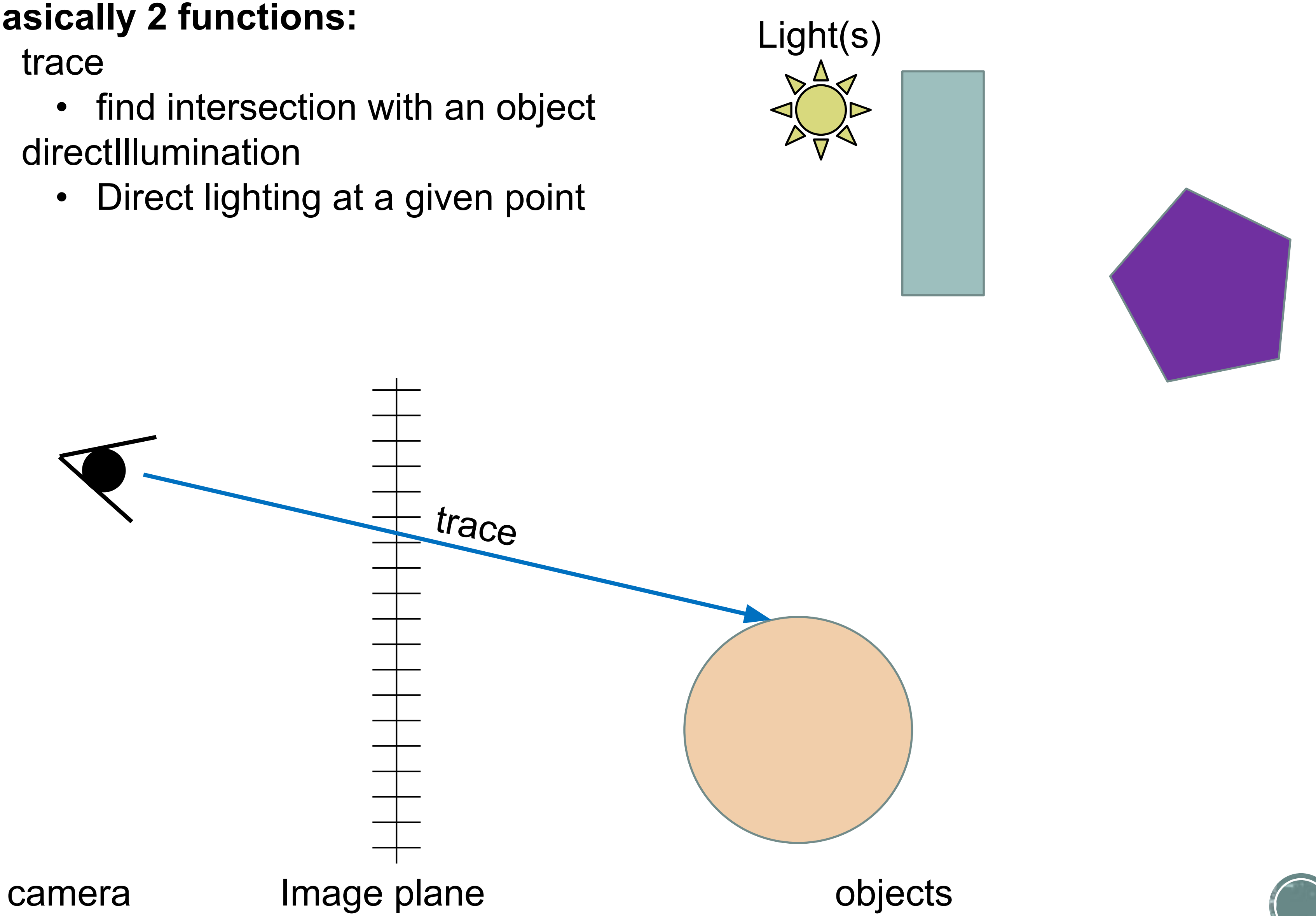
- trace
  - find intersection with an object
- directIllumination
  - Direct lighting at a given point



Ray tracing

Basically 2 functions:

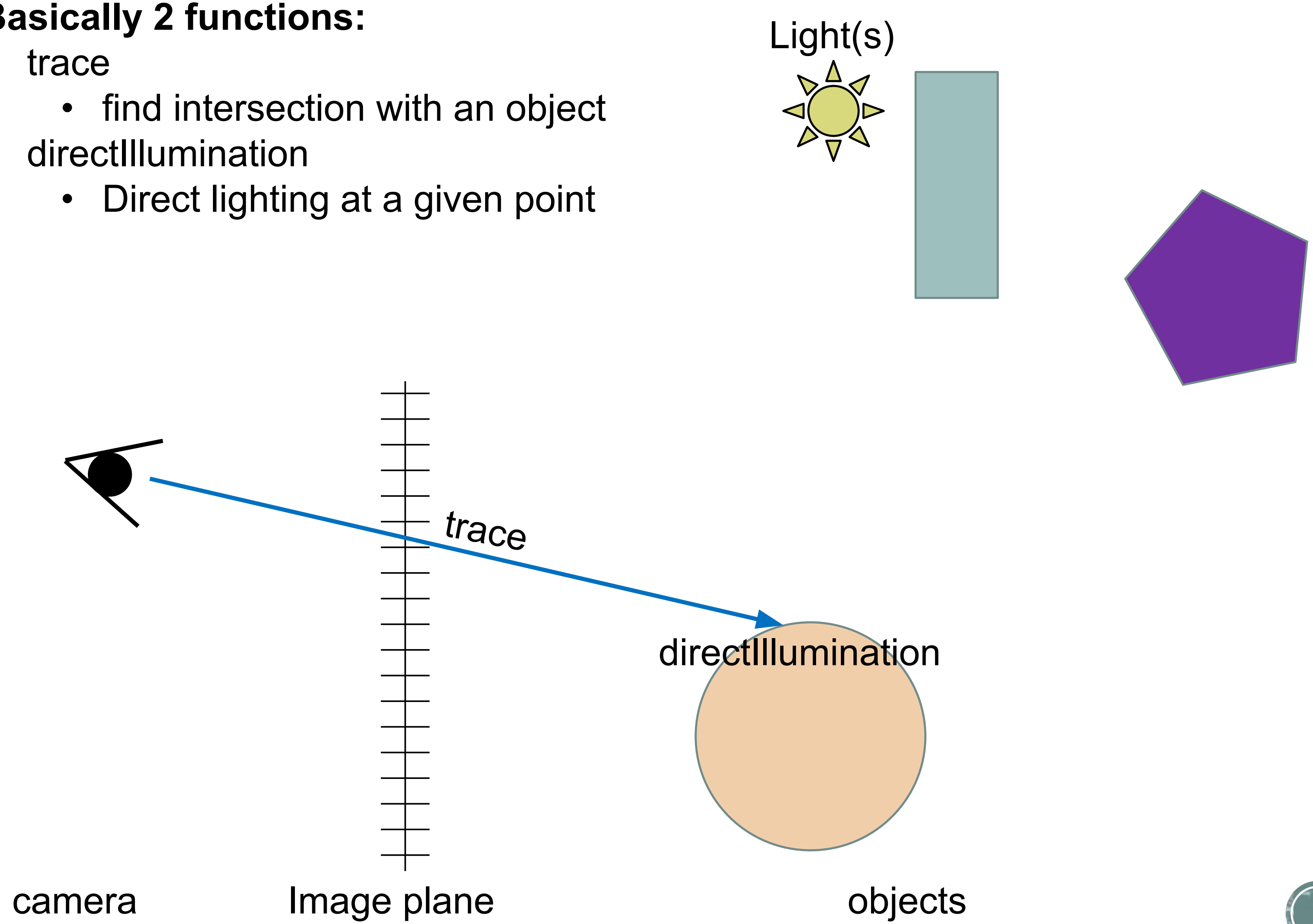
- trace
  - find intersection with an object
- directIllumination
  - Direct lighting at a given point



Ray tracing

Basically 2 functions:

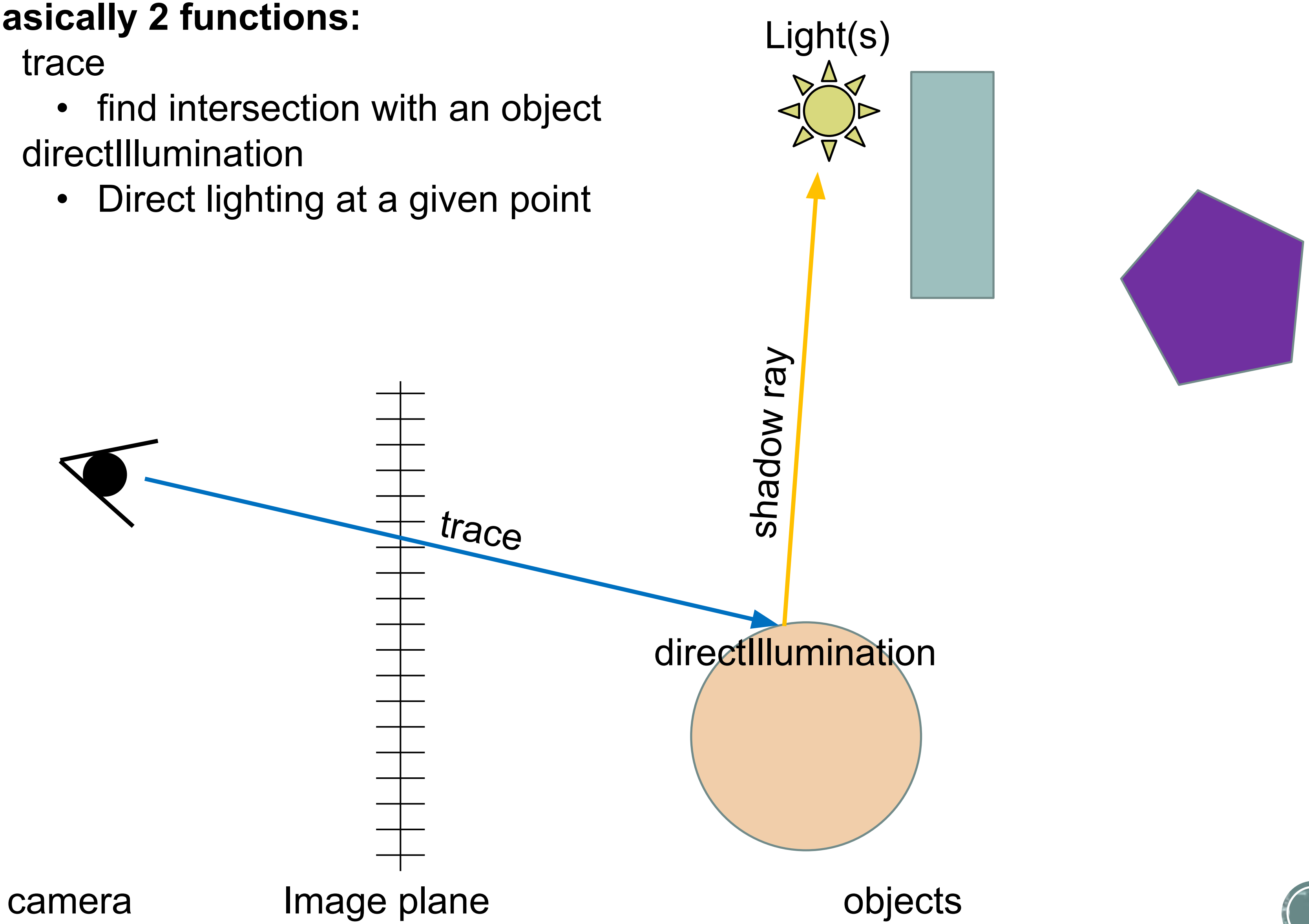
- trace
  - find intersection with an object
- directIllumination
  - Direct lighting at a given point



Ray tracing

Basically 2 functions:

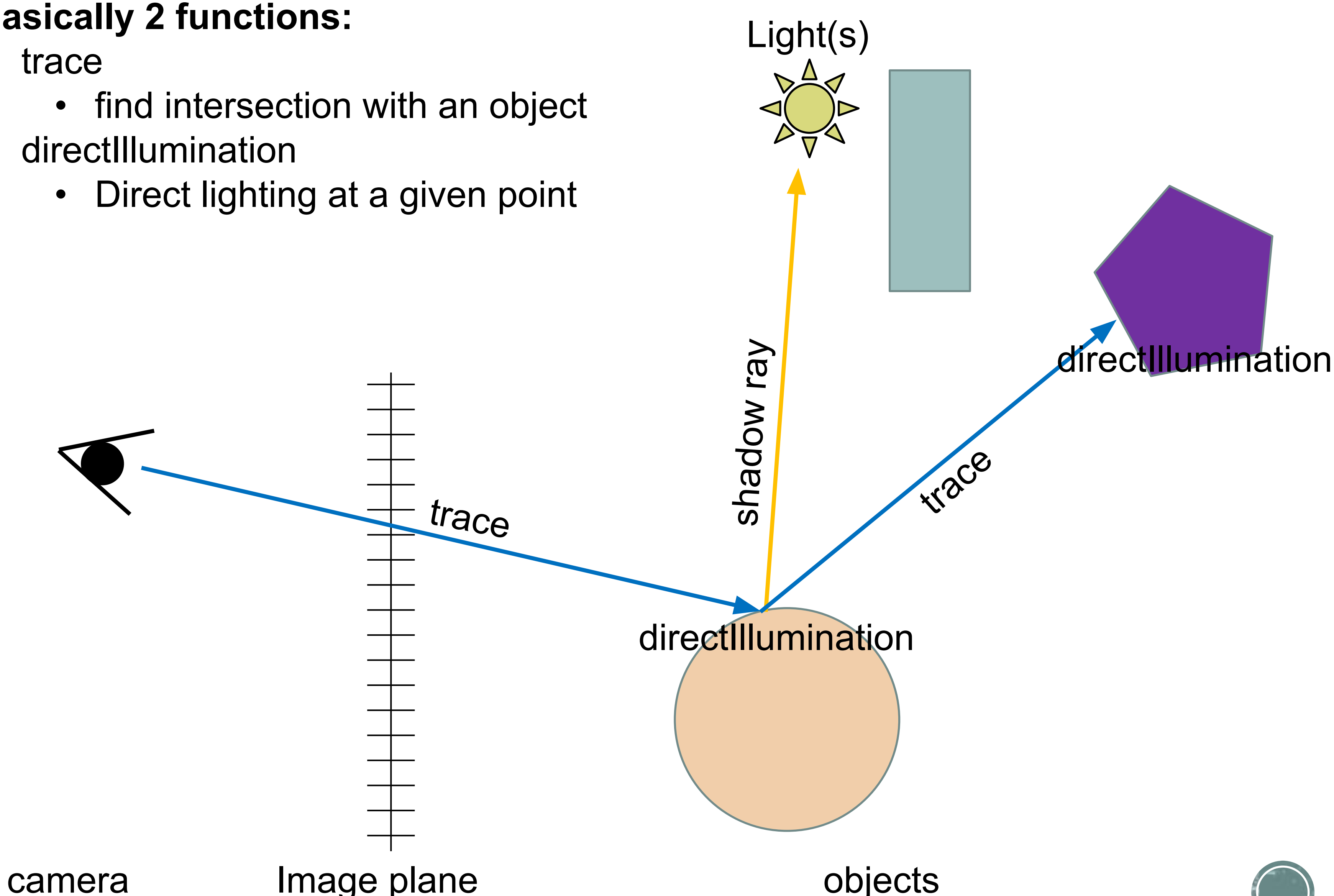
- trace
  - find intersection with an object
- directIllumination
  - Direct lighting at a given point



Ray tracing

Basically 2 functions:

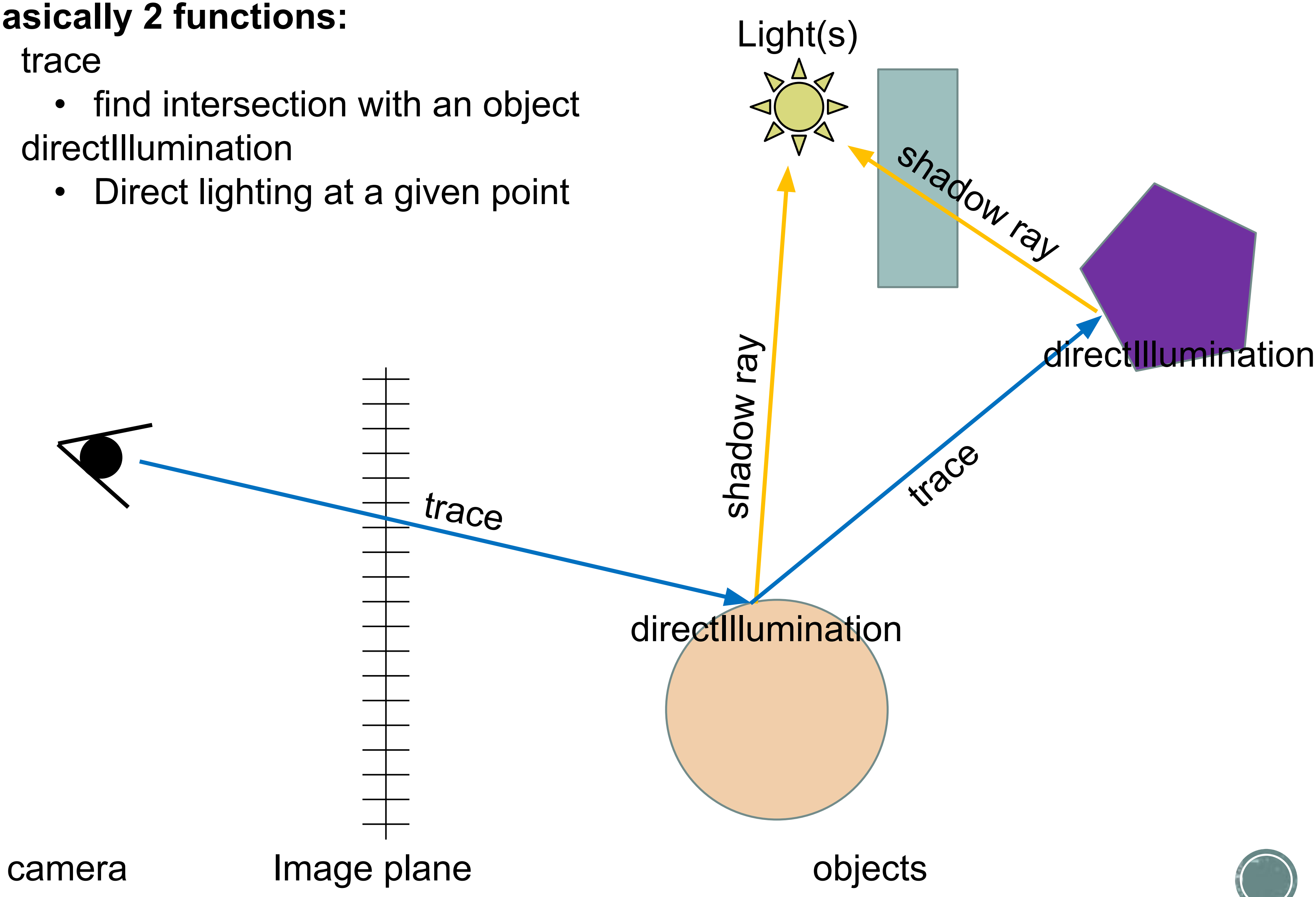
- trace
  - find intersection with an object
- directIllumination
  - Direct lighting at a given point



Ray tracing

Basically 2 functions:

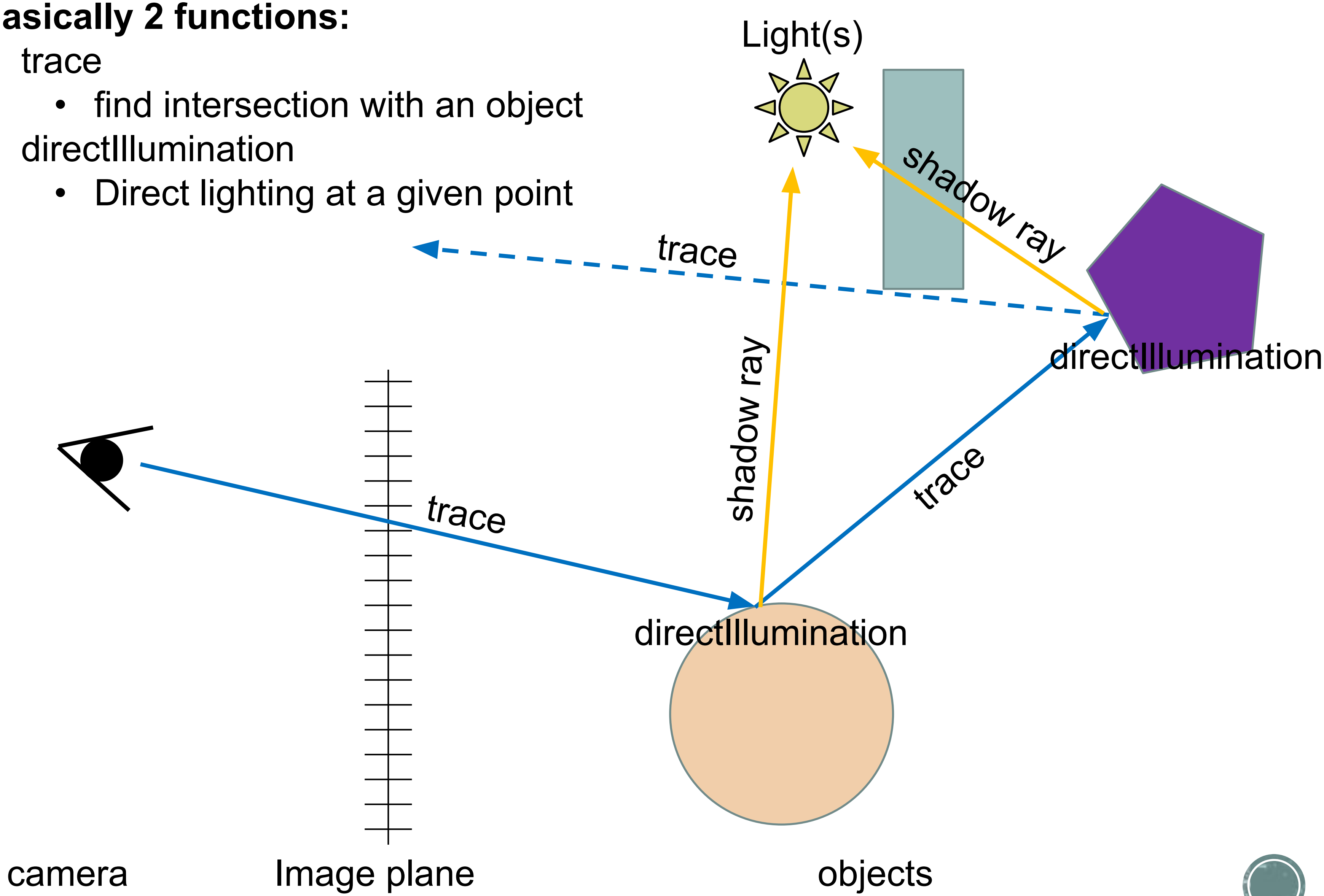
- trace
  - find intersection with an object
- directIllumination
  - Direct lighting at a given point



Ray tracing

Basically 2 functions:

- trace
  - find intersection with an object
- directIllumination
  - Direct lighting at a given point



Ray tracing

- color trace(ray) {
  - hit = intersectScene(ray)
  - if(hit) {
    - color = directIllumination(hit)
    - if hit is reflective
      - color += c_refl * trace(reflected ray)
    - if hit is transmissive
      - color += c_trans * trace(refracted ray)
  - } else
    - color = background_color
  - return color
- }

Ray tracing

- color trace(ray) {
  - hit = intersectScene(ray)   OK
  - if(hit) {
    - color = directIllumination(hit)
    - if hit is reflective
      - color += c_refl * trace(reflected ray)
    - if hit is transmissive
      - color += c_trans * trace(refracted ray)
  - } else
    - color = background_color
  - return color
- }

Ray tracing

- color trace(ray) {
  - hit = intersectScene(ray)
  - if(hit) {
    - color = directIllumination(hit) ?
    - if hit is reflective
      - color += c_refl * trace(reflected ray)
    - if hit is transmissive
      - color += c_trans * trace(refracted ray)
  - } else
    - color = background_color
  - return color
- }

Ray tracing

- color directIllumination(hit) {
  - color = (0,0,0)
  - for each light L {
    - T = cast shadow ray to L
    - if hit is not shadowed by L
      - color += Ambient+diffuse+specular terms(L,hit)
  - }
  - return color
- }

Ray tracing

- color directIllumination(hit) {
  - color = (0,0,0)
  - for each light L {
    - I = cast shadow ray to L
    - if hit is not shadowed by L
      - color += Ambient+diffuse+specular terms(L,hit)
  - }
  - return color
- }

?

Difference between eye ray and shadow ray ?

Ray tracing

- color directIllumination(hit) {
  - color = (0,0,0)
  - for each light L {
    - T = cast shadow ray to L
    - if hit is not shadowed by L
      - $\boxed{\text{color += Ambient+diffuse+specular terms(L,hit)}}$

?
  - }
  - return color
- }

Material properties (we will see it soon in details)

Ray tracing

- color trace(ray) {
  - hit = intersectScene(ray)
  - if(hit) {
    - color = directIllumination(hit)
    - if hit is reflective
      - color += c_refl * trace(reflected ray)
    - if hit is transmissive
      - color += c_trans * trace(refracted ray)
  - } else
    - color = background_color
  - return color
- }

OK

Ray tracing

- color trace(ray) {
  - hit = intersectScene(ray)
  - if(hit) {
    - color = directIllumination(hit)
    - if hit is reflective
      - color += c_refl * trace(reflected ray)
    - if hit is transmissive
      - color += c_trans * trace(refracted ray)
  - } else
    - color = background_color
  - return color
- }

Recursive!

Ray tracing

- color trace(ray) {
  - hit = intersectScene(ray)
  - if(hit) {
    - color = directIllumination(hit)
    - if hit is reflective
      - color += c_refl * trace(reflected ray)
    - if hit is transmissive
      - color += c_trans * trace(refracted ray)
  - } else
    - color = background_color
  - return color
- }

Recursive!

How to stop?

- Recursion depth
  - After a number of bounces
- Ray contribution
  - Reflected/transmitted contribution becomes too small

Ray tracing

- color trace(ray) {
  - hit = intersectScene(ray)
  - if(hit) {
    - color = directIllumination(hit)
    - if hit is reflective
      - color += c_refl * trace(reflected ray)
    - if hit is transmissive
      - color += c_trans * trace(refracted ray)
  - } else
    - color = background_color
  - return color
- }

Recursive!

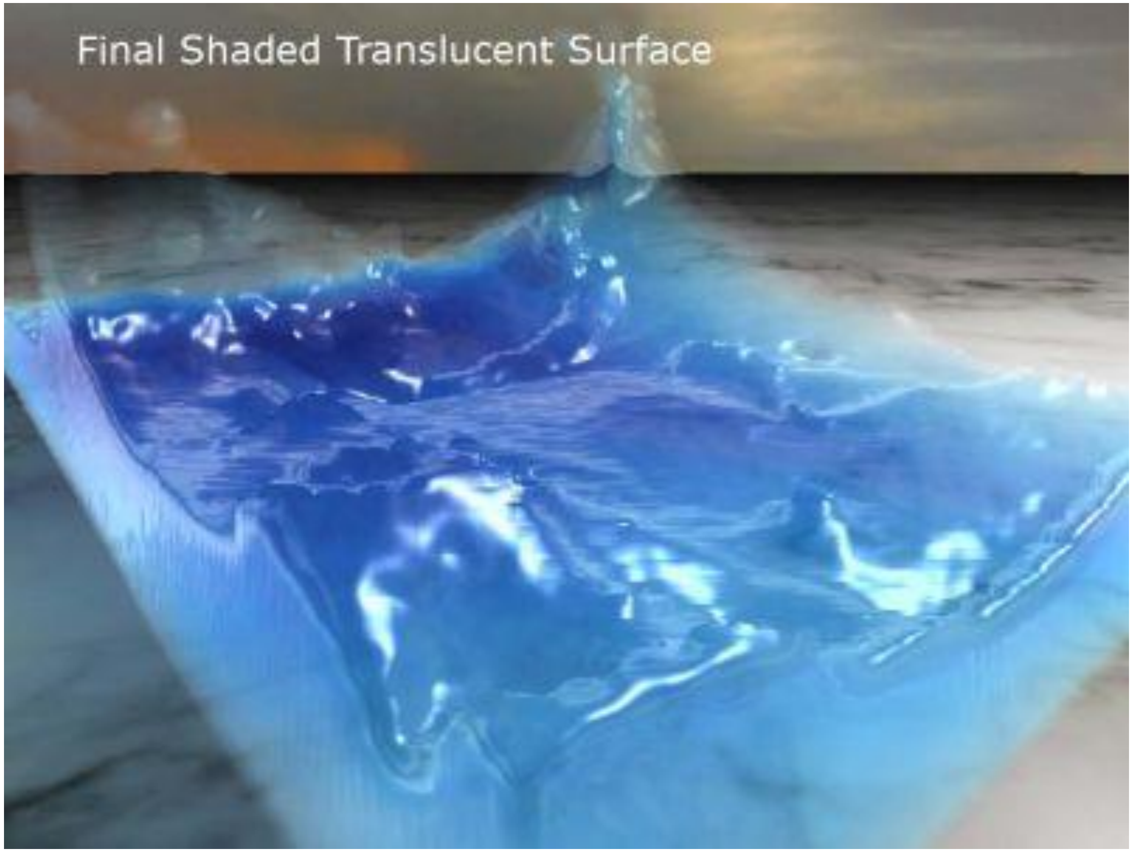
How to stop?

glossary

- Ray casting:
  - eye ray only
- Ray tracing:
  - all secondary rays
- Shadow ray:
  - surface to light ray (shadow test)
- Ray marching:
  - step by step surface intersection test

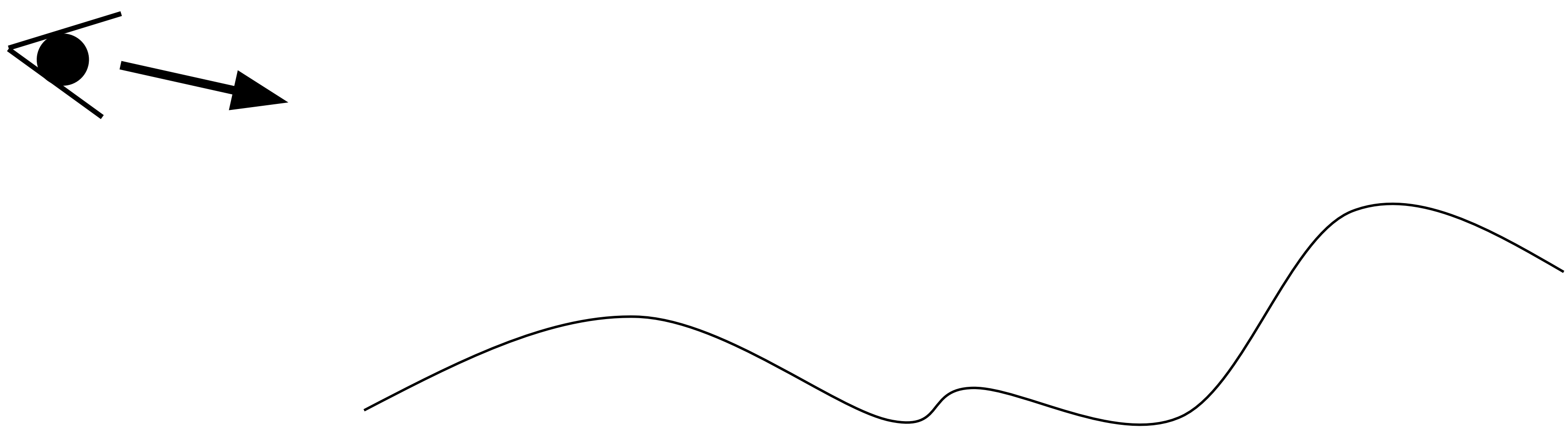
Ray-tracing

What if intersection cannot be computed analytically?



Ray marching height-fields

$$P(t) = r_o + r_d * t$$

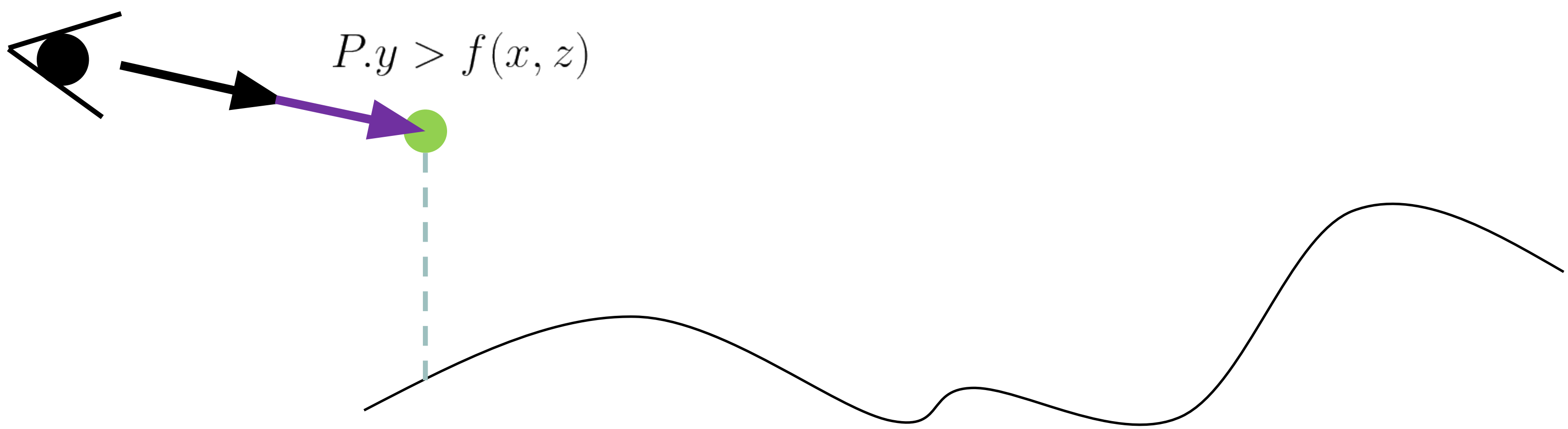


camera

Height field $f(x, z) = y$

Ray marching height-fields

$P(t) = r_o + r_d * t$

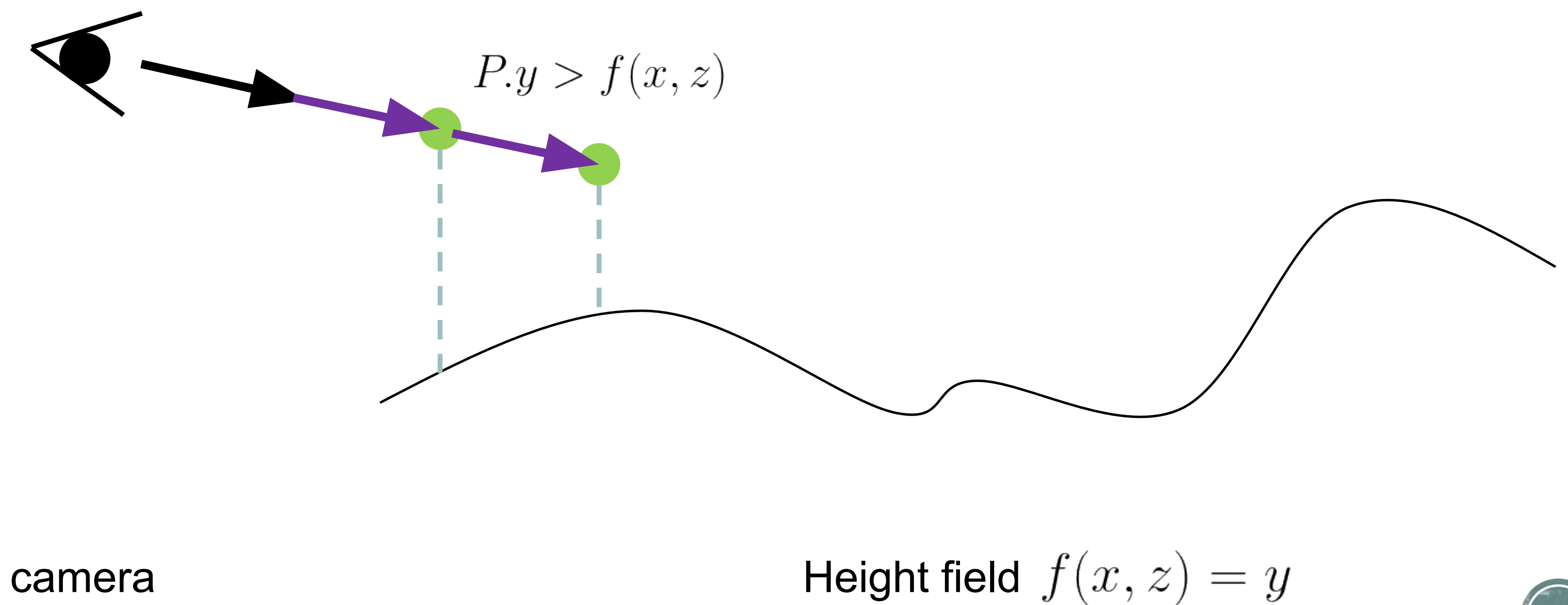


camera

Height field $f(x, z) = y$

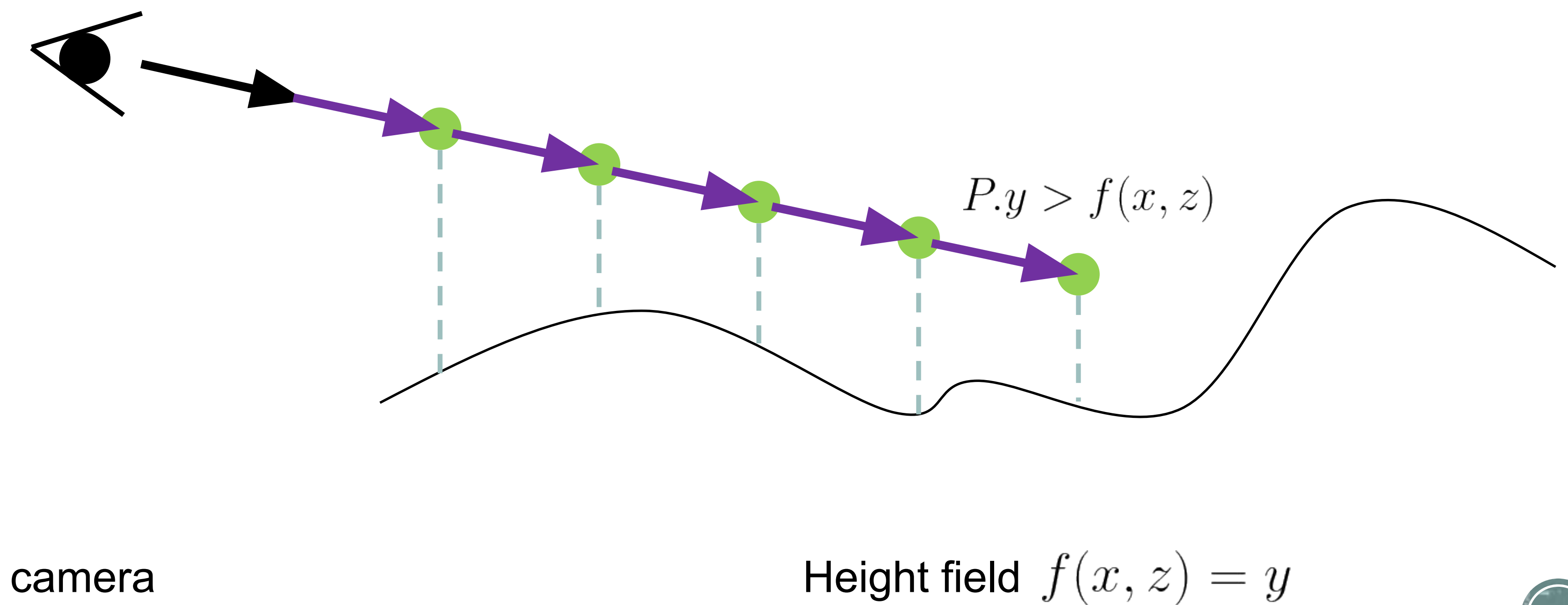
Ray marching height-fields

$$P(t) = r_o + r_d * t$$



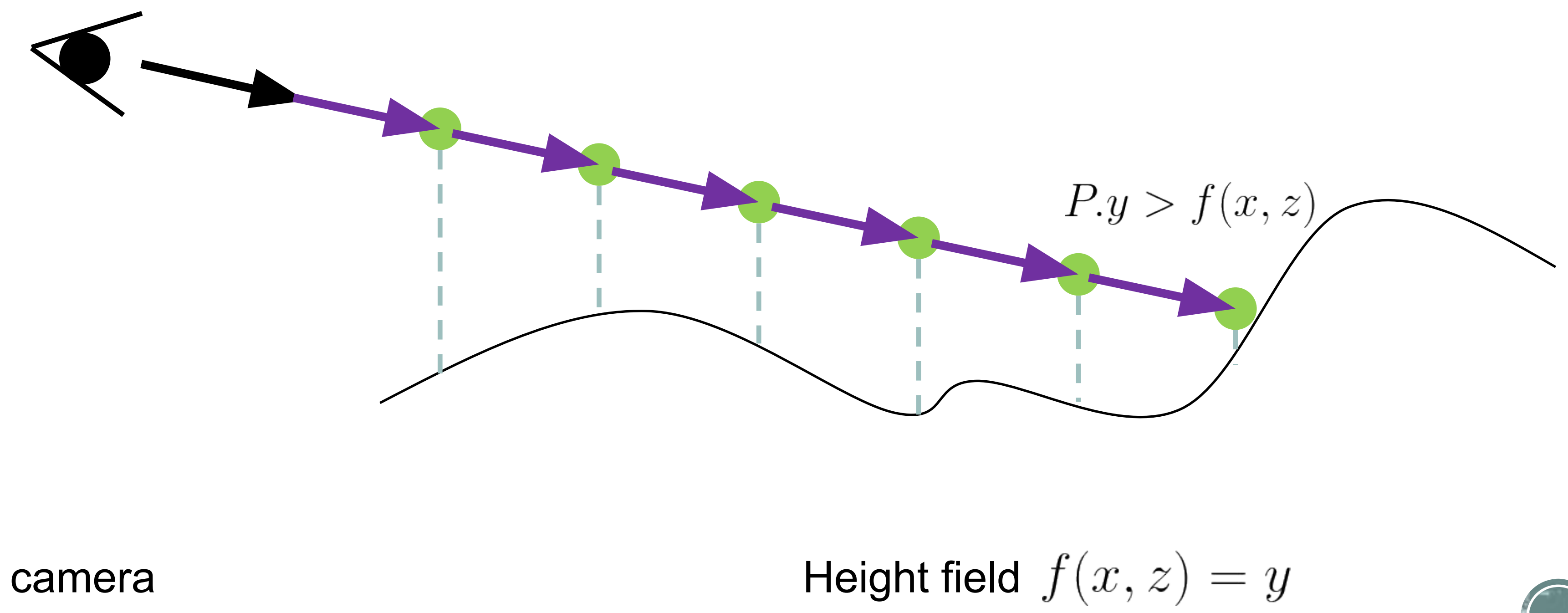
Ray marching height-fields

$$P(t) = r_o + r_d * t$$



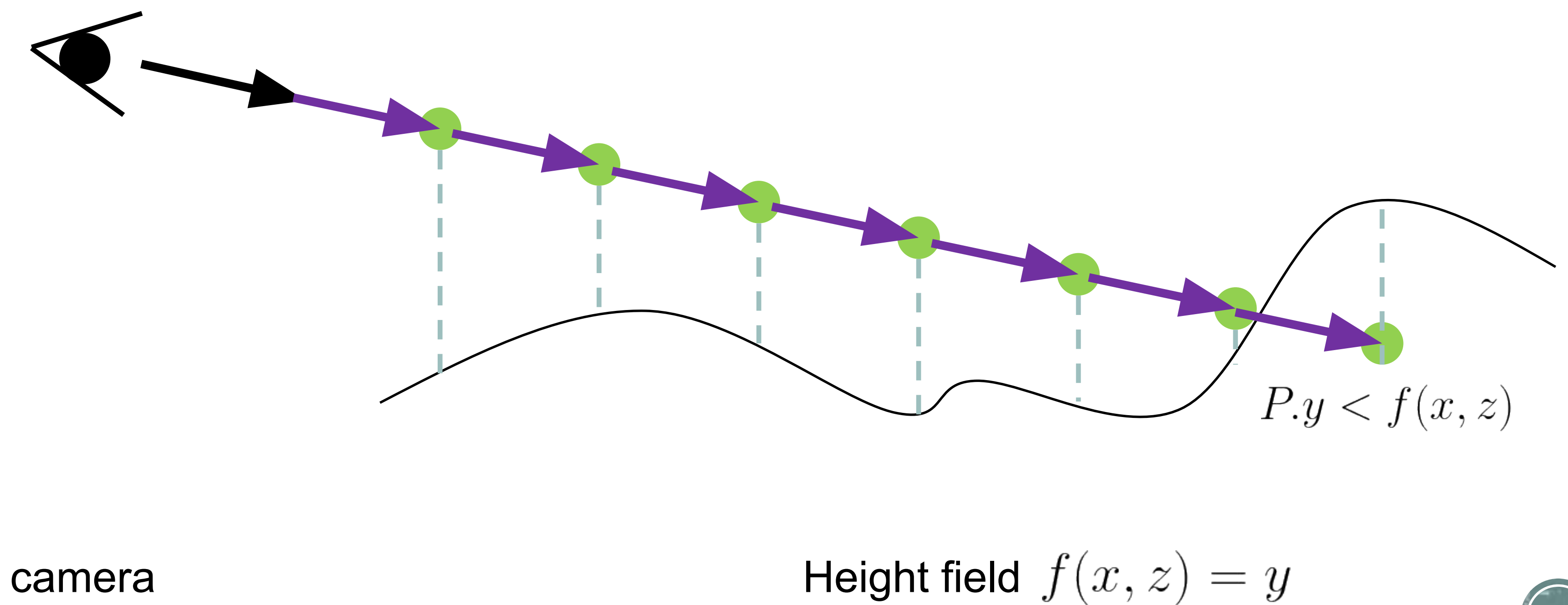
Ray marching height-fields

$P(t) = r_o + r_d * t$



Ray marching height-fields

$$P(t) = r_o + r_d * t$$

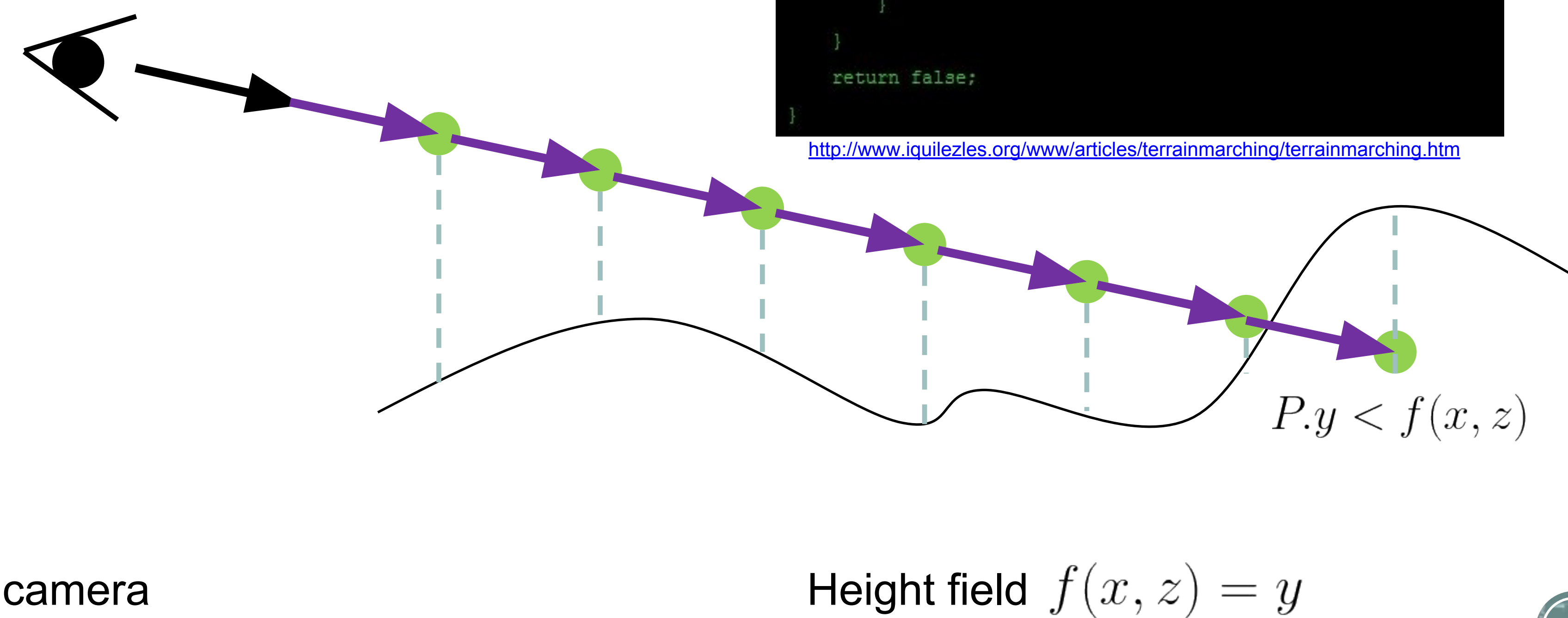


Ray marching height-fields

$$P(t) = r_o + r_d * t$$

```
bool castRay( const vec3 & ro, const vec3 & rd, float & resT )
{
    const float delt = 0.01f;
    const float mint = 0.001f;
    const float maxt = 10.0f;
    for( float t = mint; t < maxt; t += delt )
    {
        const vec3 p = ro + rd*t;
        if( p.y < f( p.x, p.z ) )
        {
            resT = t - 0.5f*delt;
            return true;
        }
    }
    return false;
}
```

http://www.iquilezles.org/www/articles/terrainmarching/terrainmarching.htm

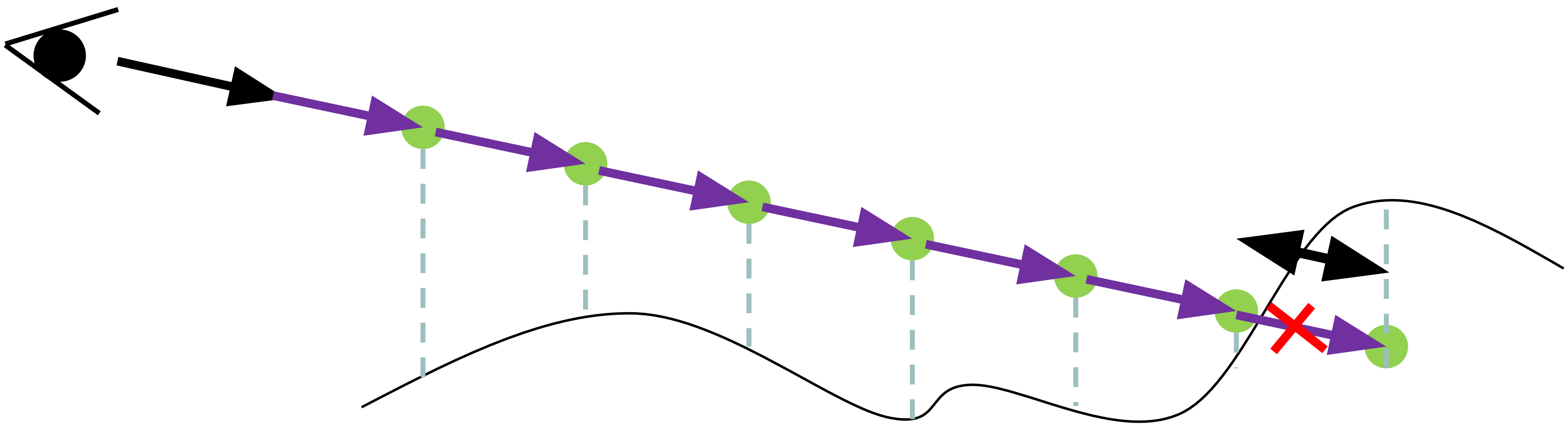


Ray marching height-fields

$$P(t) = r_o + r_d * t$$

Optimizations:

- Interpolate between the 2 last positions



camera

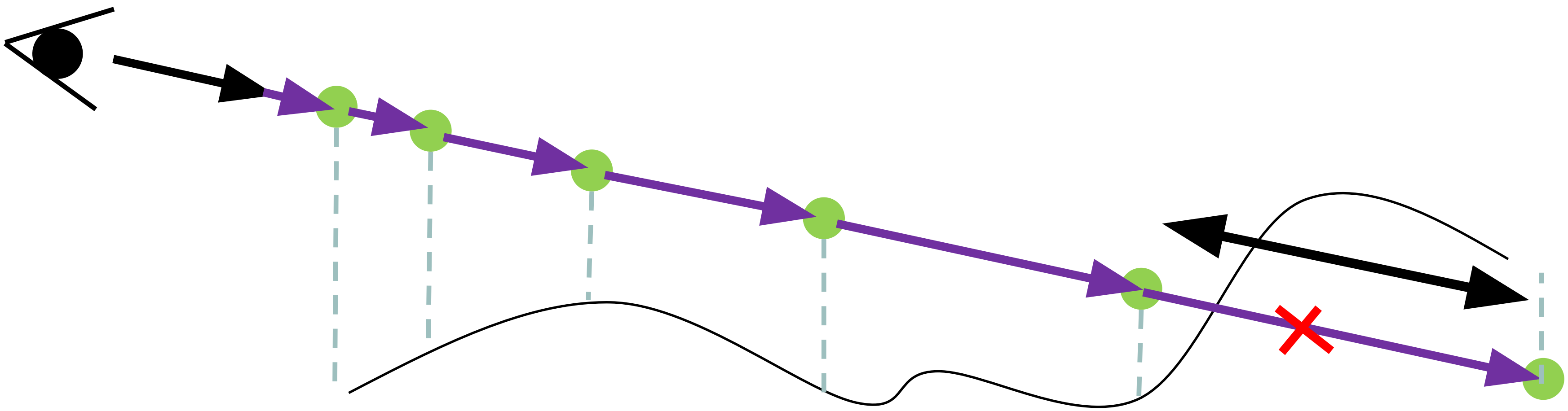
Height field $f(x, z) = y$

Ray marching height-fields

$$P(t) = r_o + r_d * t$$

Optimizations:

- Interpolate between the 2 last positions
- Increase deltaT with distance from eye



camera

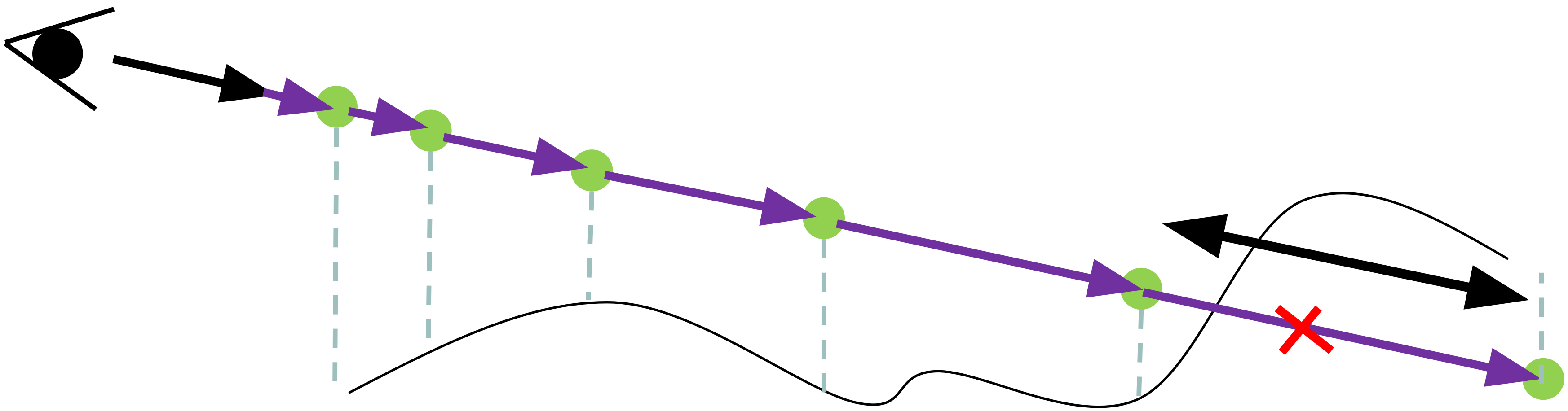
Height field $f(x, z) = y$

Ray marching height-fields

$P(t) = r_o + r_d * t$

Optimizations:

- Interpolate between the 2 last positions
- Increase deltaT with distance from eye
- See: http://www.iquilezles.org



camera

Height field $f(x, z) = y$

Ray marching height-fields

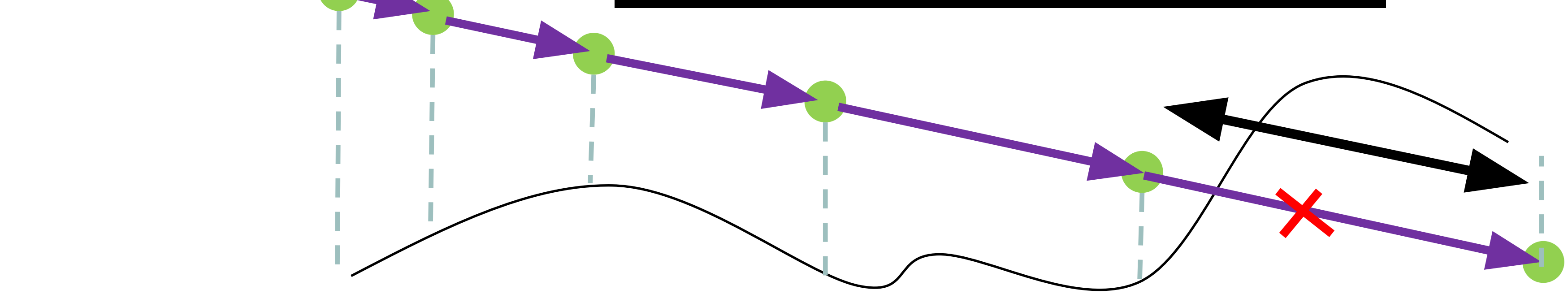
$P(t) = r_o + r_d * t$

Optimizations:

- Interpolate between the 2 last positions
- Increase deltaT with distance from eye
- See: http://www.iquilezles.org

Normal computation:

```
vec3 getNormal( const vec3 & p )
{
    const vec3 n = vec3( f(p.x-eps,p.z) - f(p.x+eps,p.z),
                         2.0f*eps,
                         f(p.x,p.z-eps) - f(p.x,p.z+eps) );
    return normalize( n );
}
```

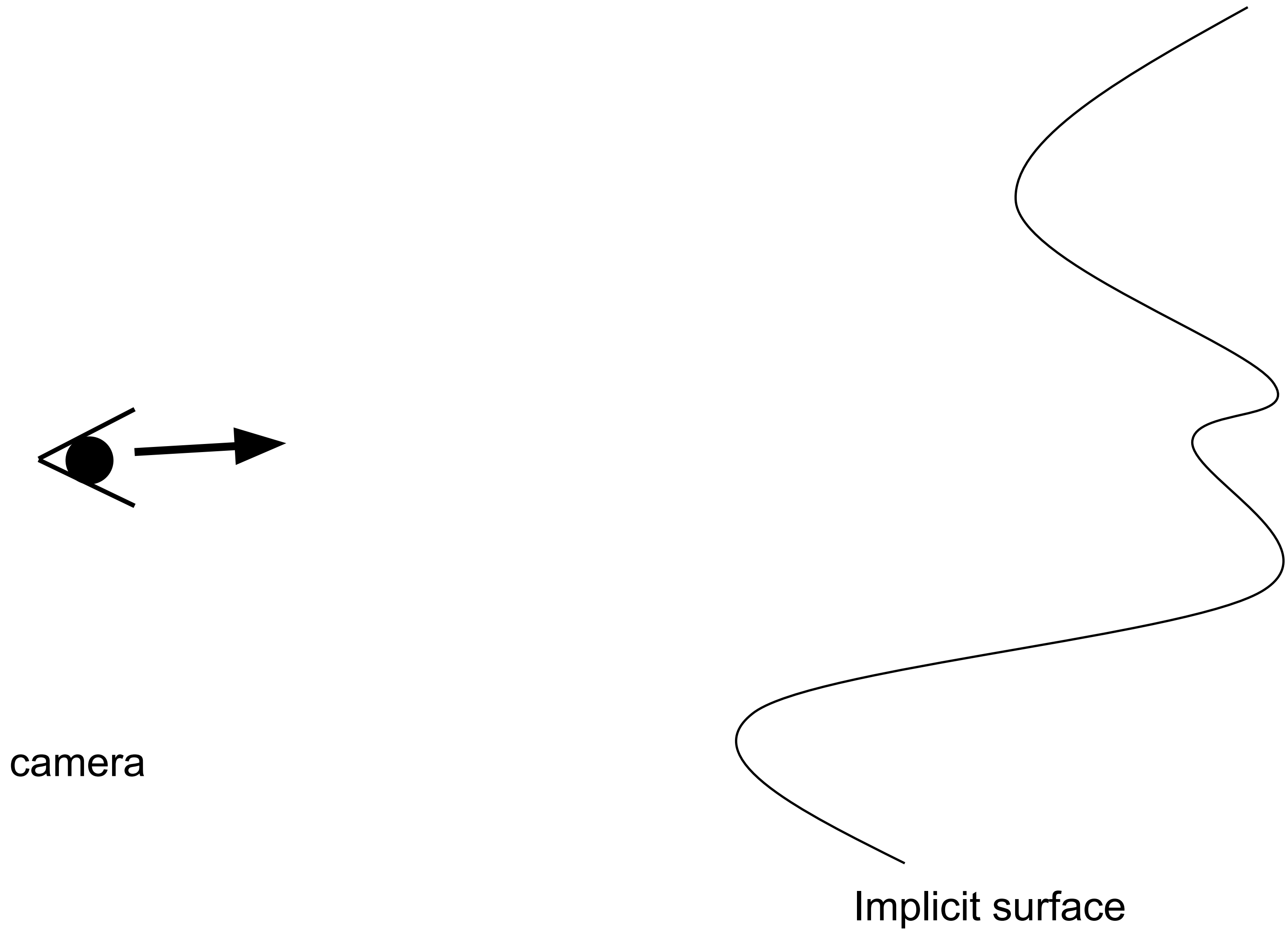


camera

Height field $f(x,z) = y$

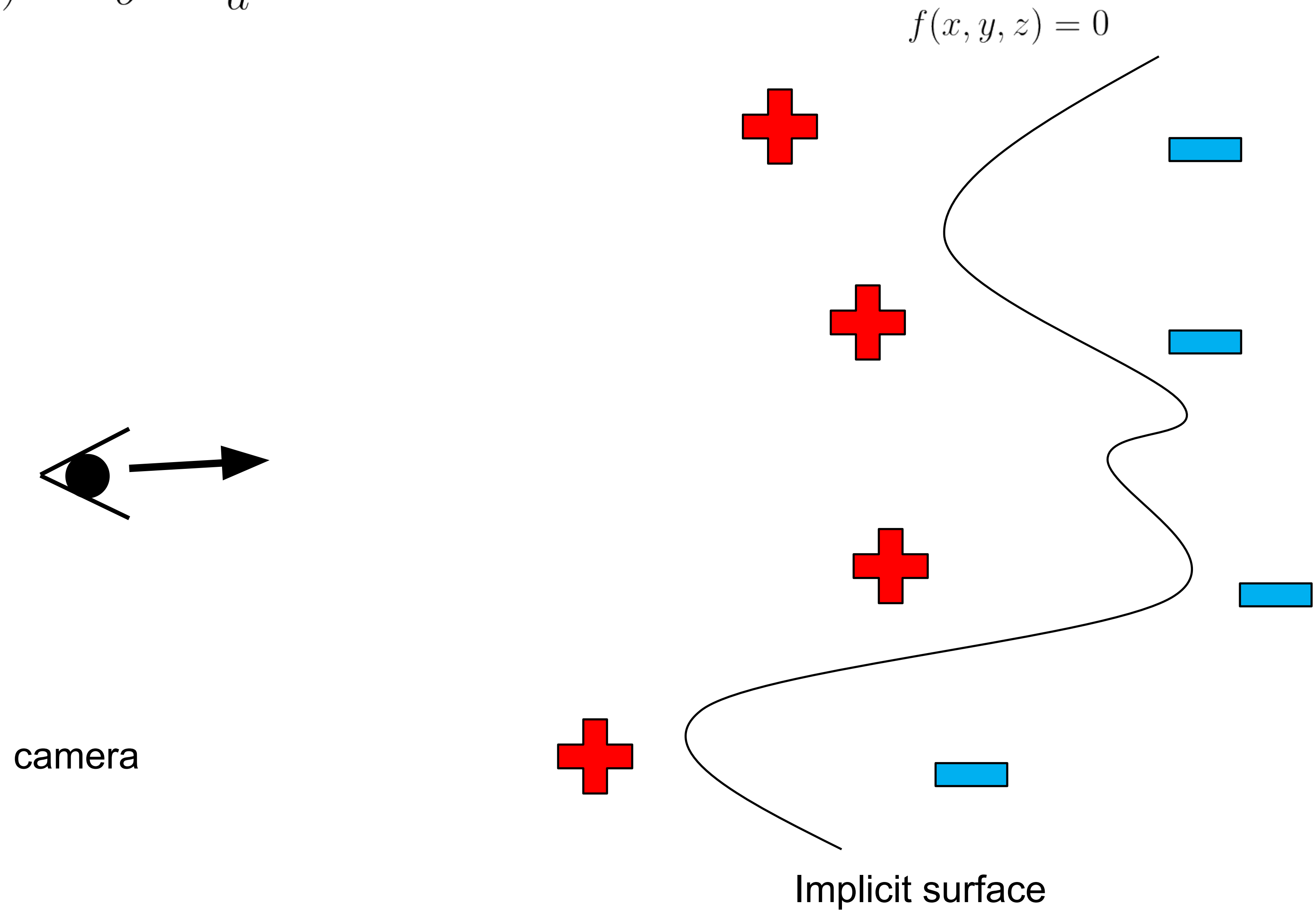
Ray marching implicit surfaces

$$P(t) = r_o + r_d * t$$



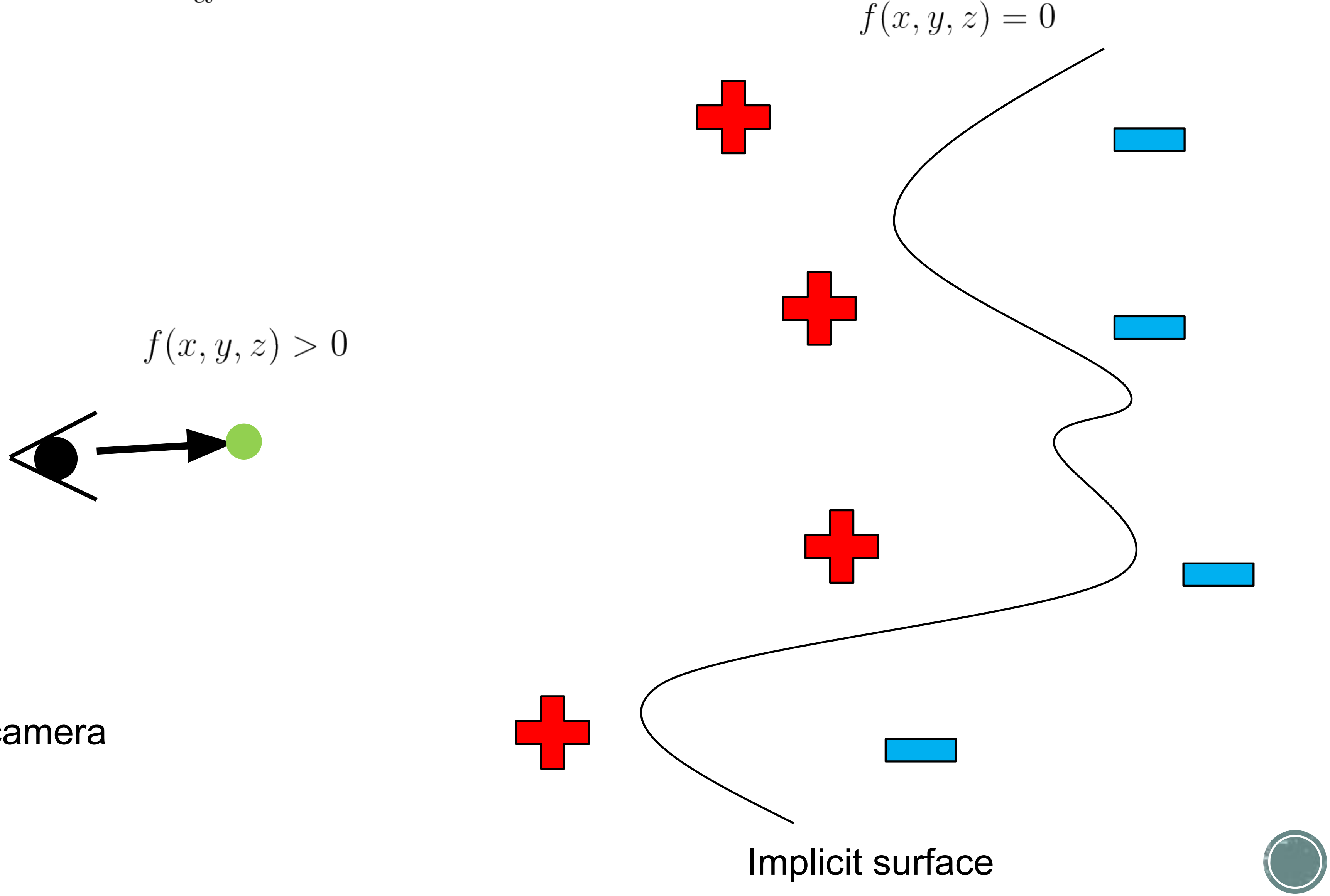
Ray marching implicit surfaces

$$P(t) = r_o + r_d * t$$



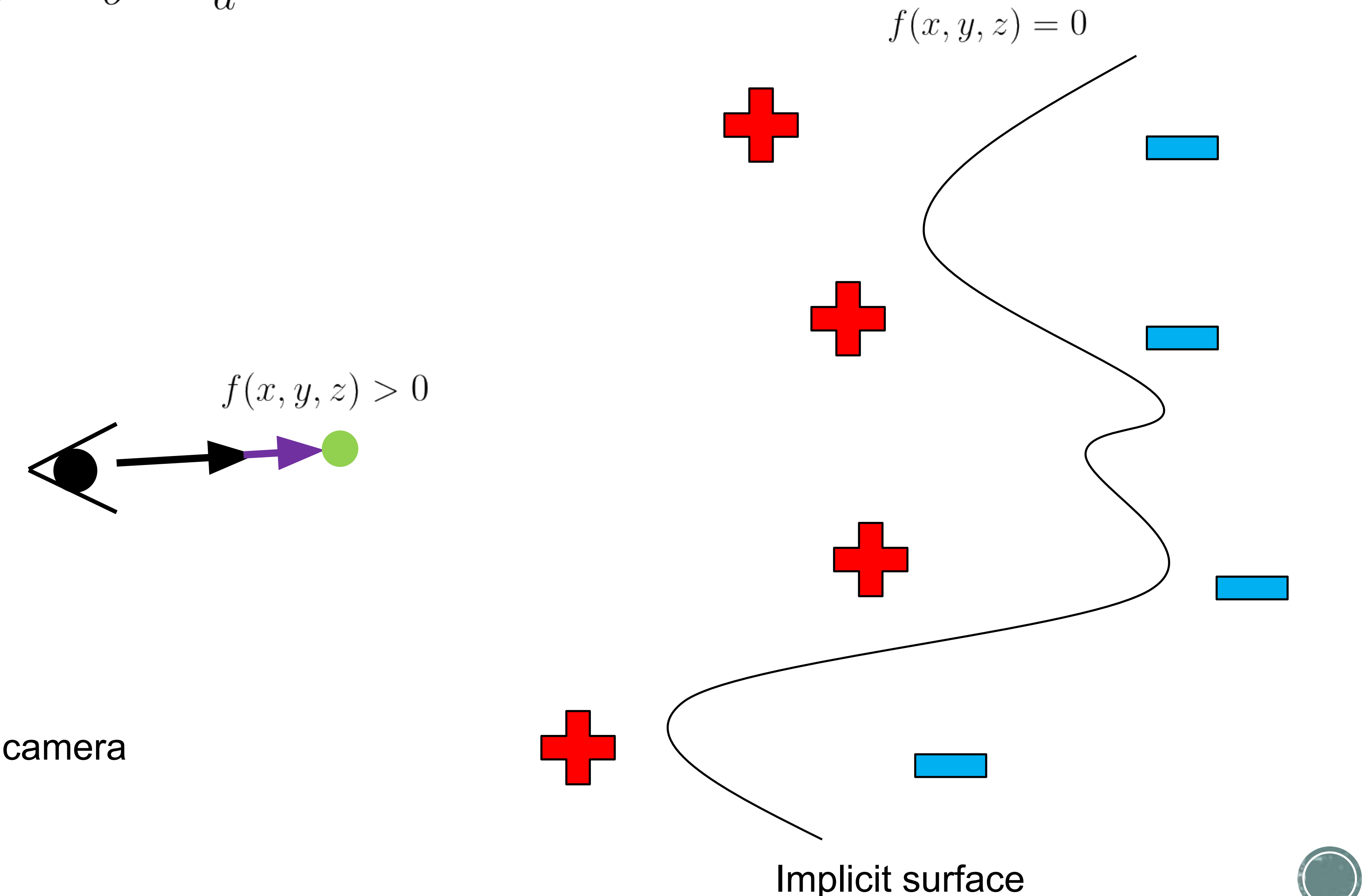
Ray marching implicit surfaces

$$P(t) = r_o + r_d * t$$



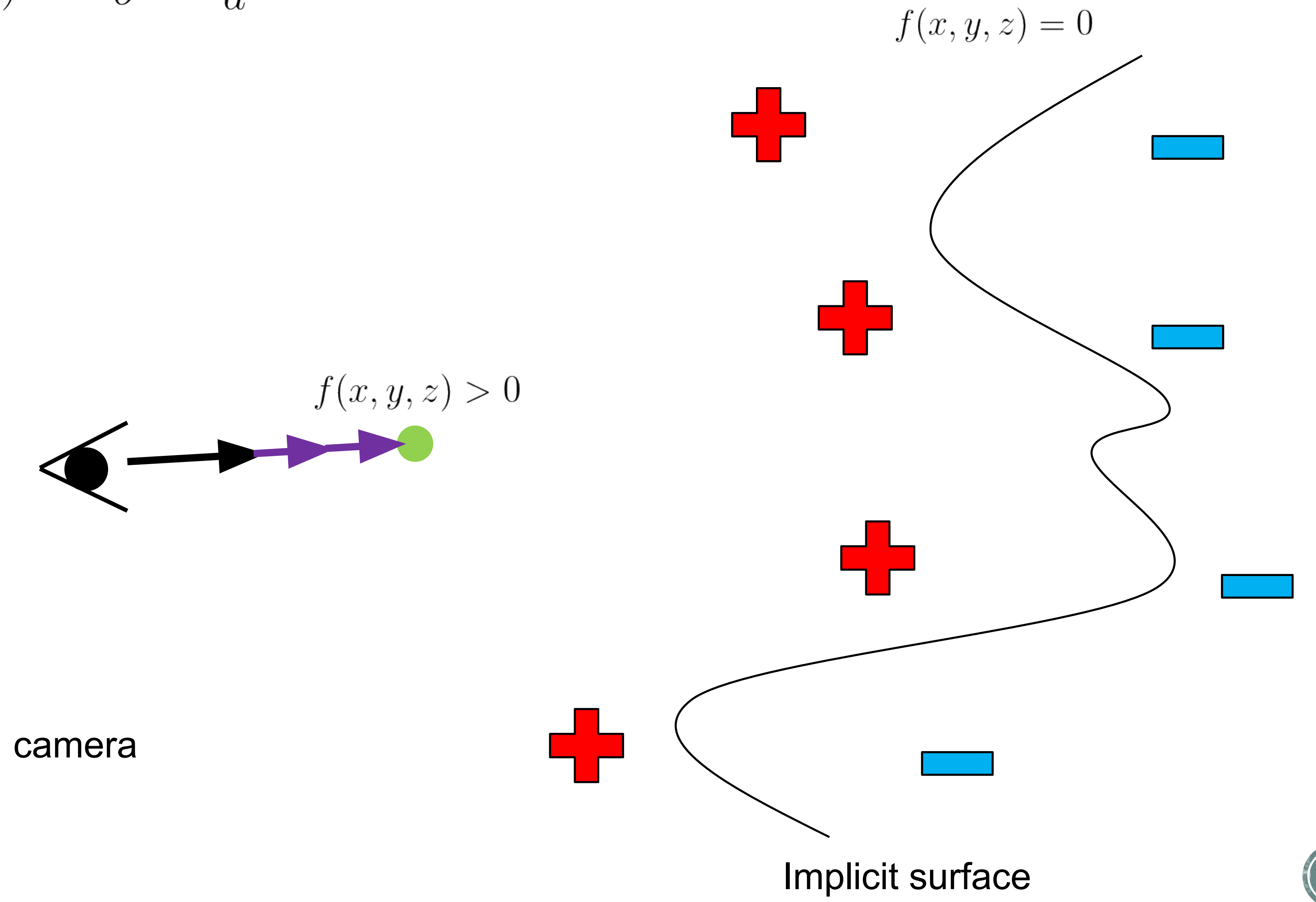
Ray marching implicit surfaces

$$P(t) = r_o + r_d * t$$



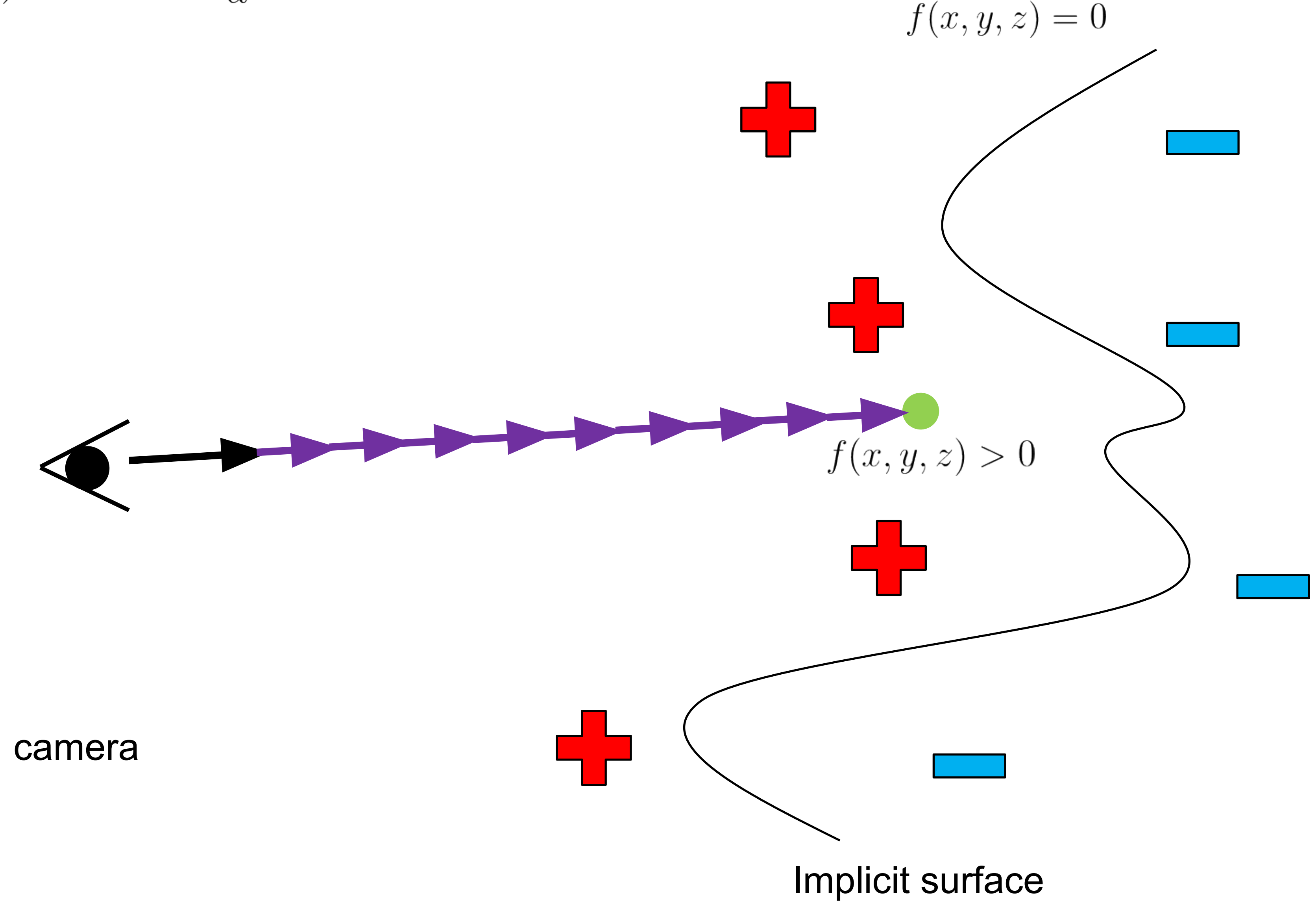
Ray marching implicit surfaces

$$P(t) = r_o + r_d * t$$



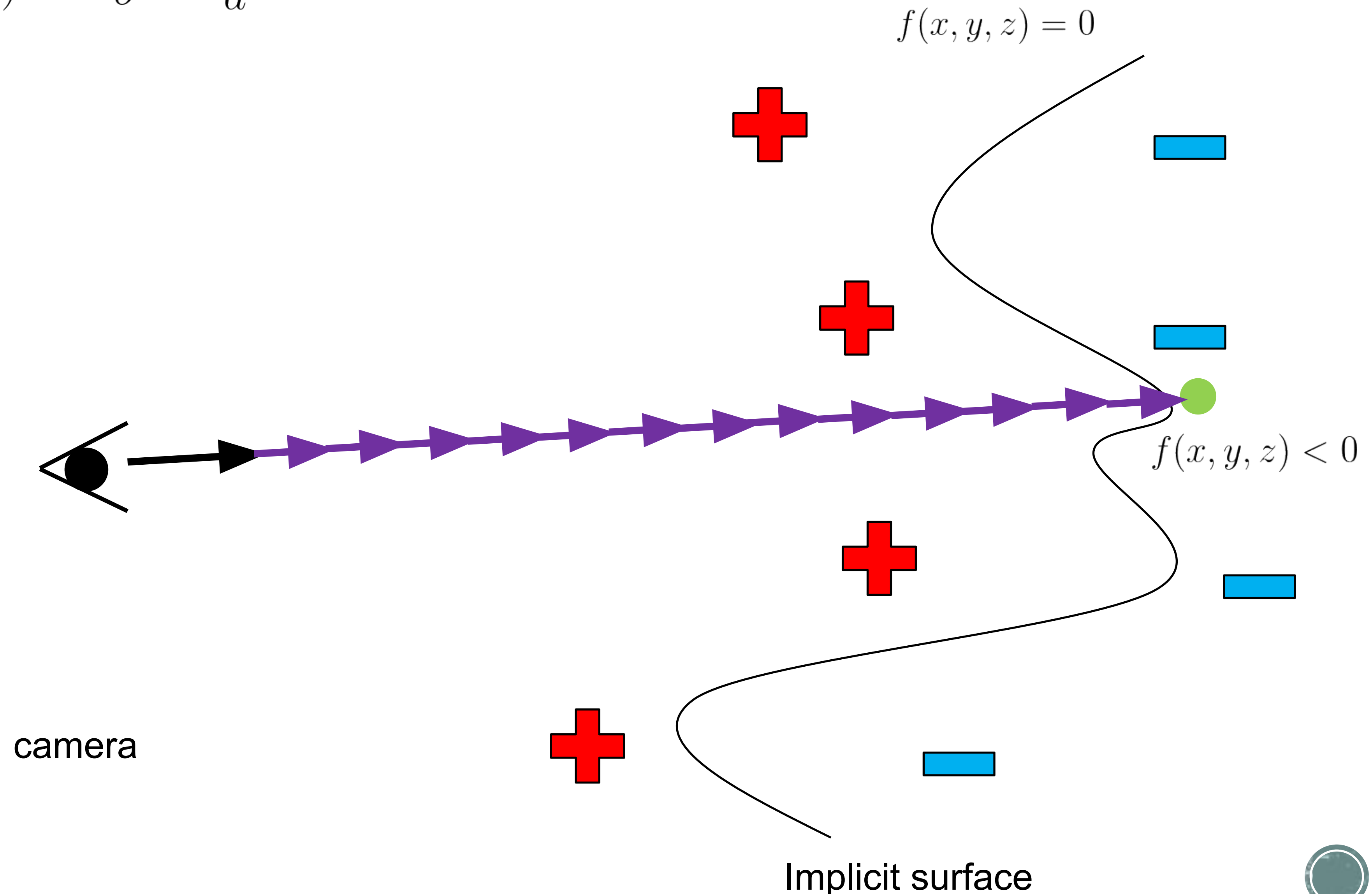
Ray marching implicit surfaces

$$P(t) = r_o + r_d * t$$



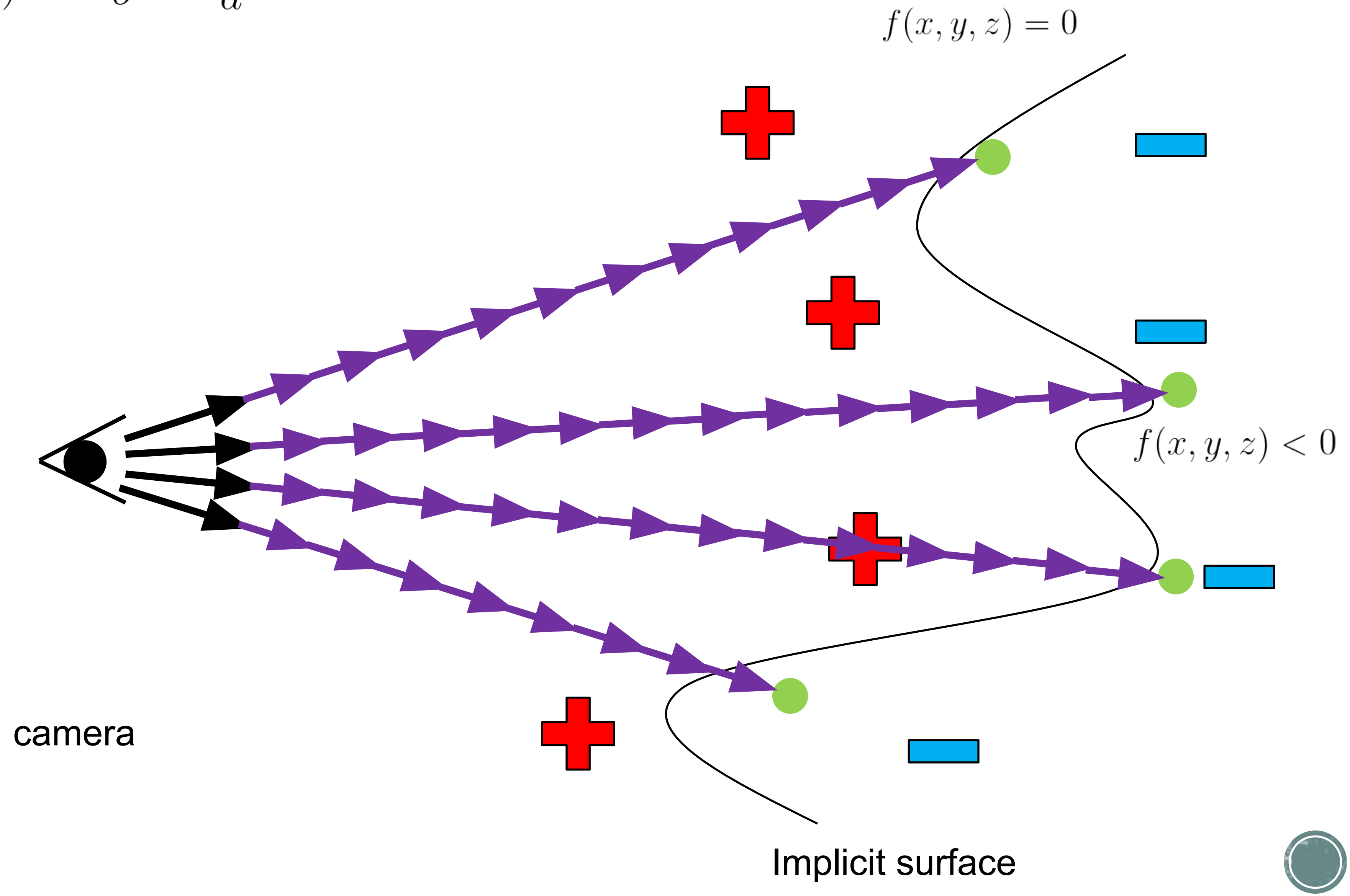
Ray marching implicit surfaces

$$P(t) = r_o + r_d * t$$



Ray marching implicit surfaces

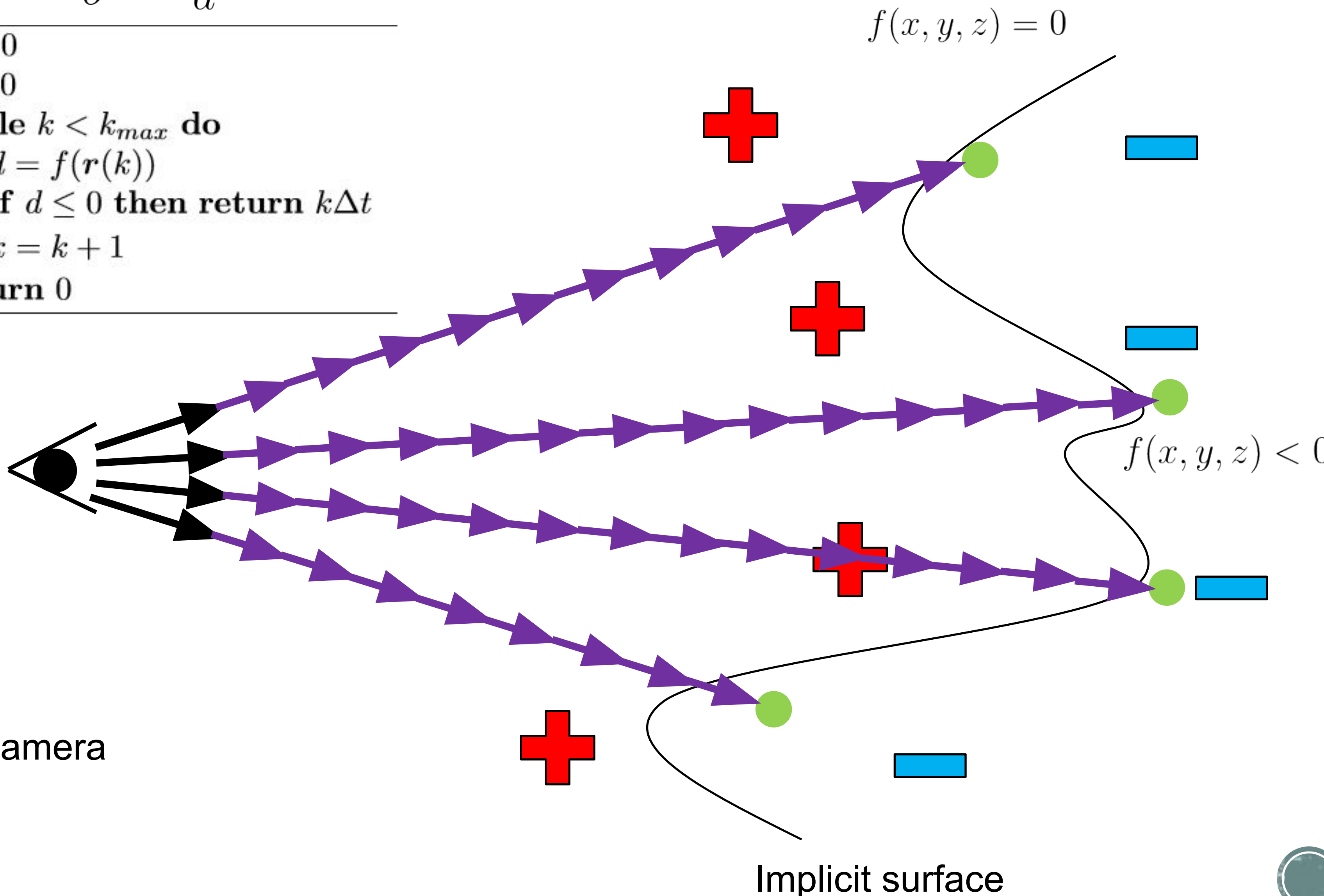
$$P(t) = r_o + r_d * t$$



Ray marching implicit surfaces

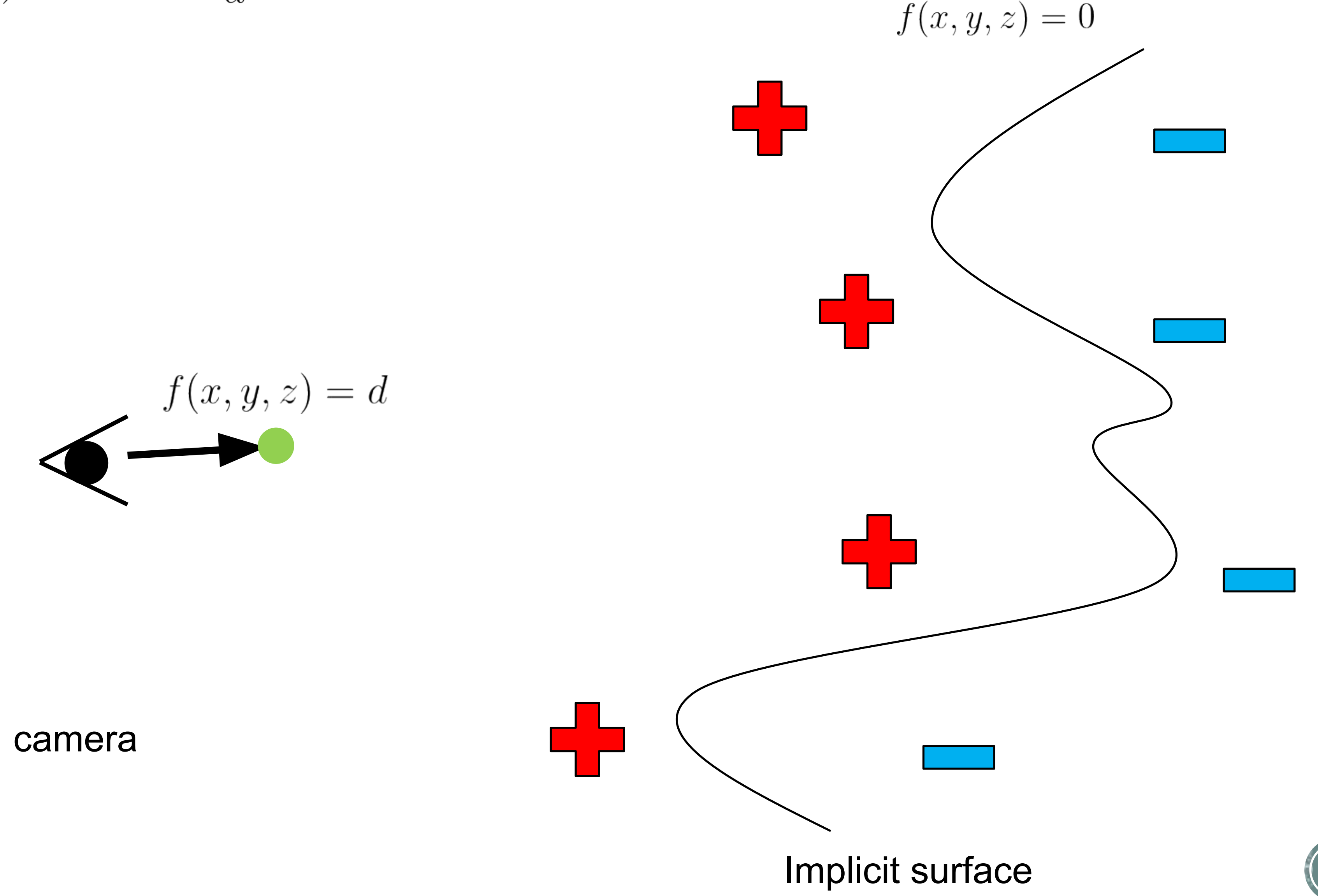
$$P(t) = r_o + r_d * t$$

$k = 0$
$d = 0$
while $k < k_{max}$ **do**
  $d = f(\boldsymbol{r}(k))$
  if $d \leq 0$ **then return** $k\Delta t$
  $k = k + 1$
return 0



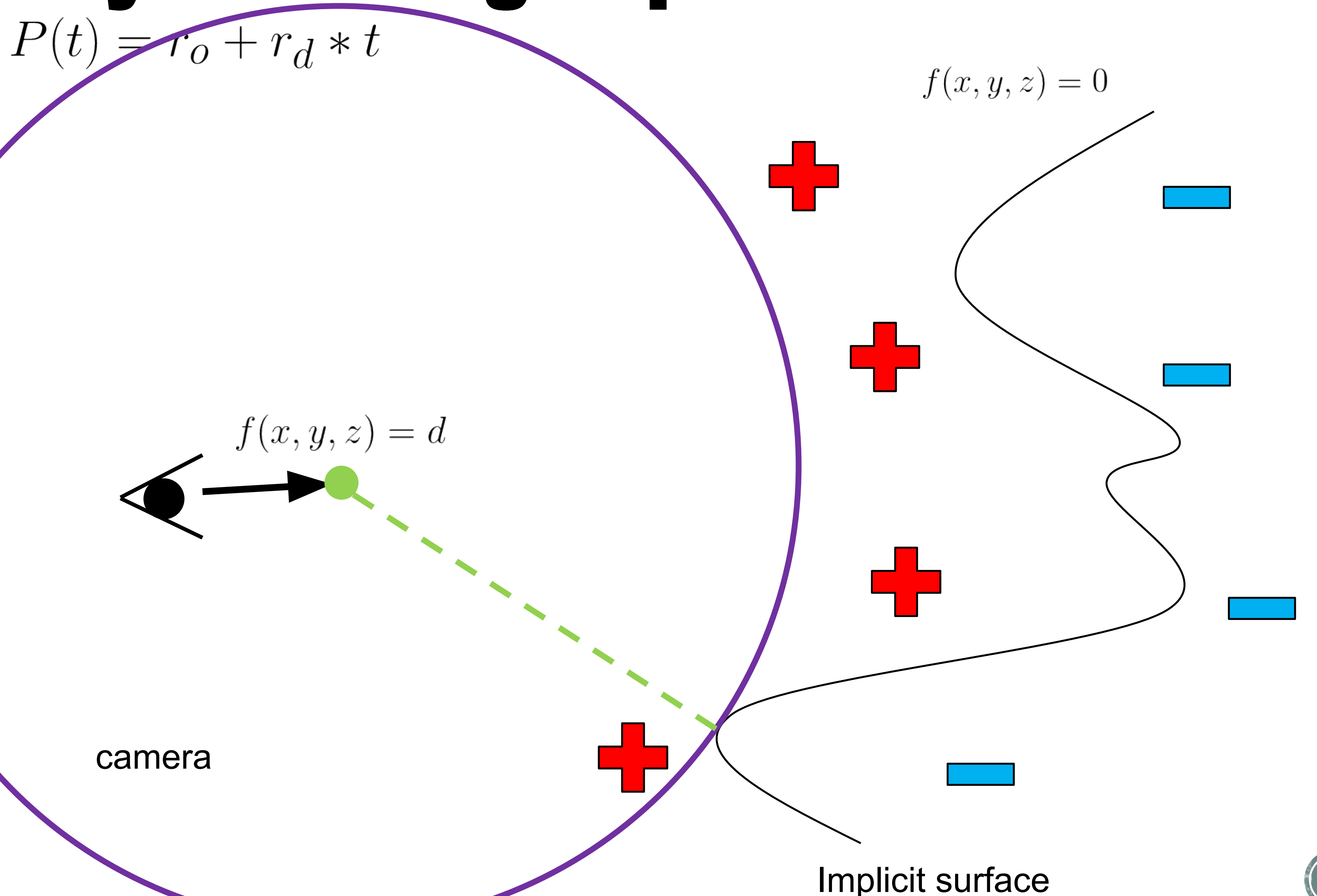
Ray marching implicit surfaces

$$P(t) = r_o + r_d * t$$



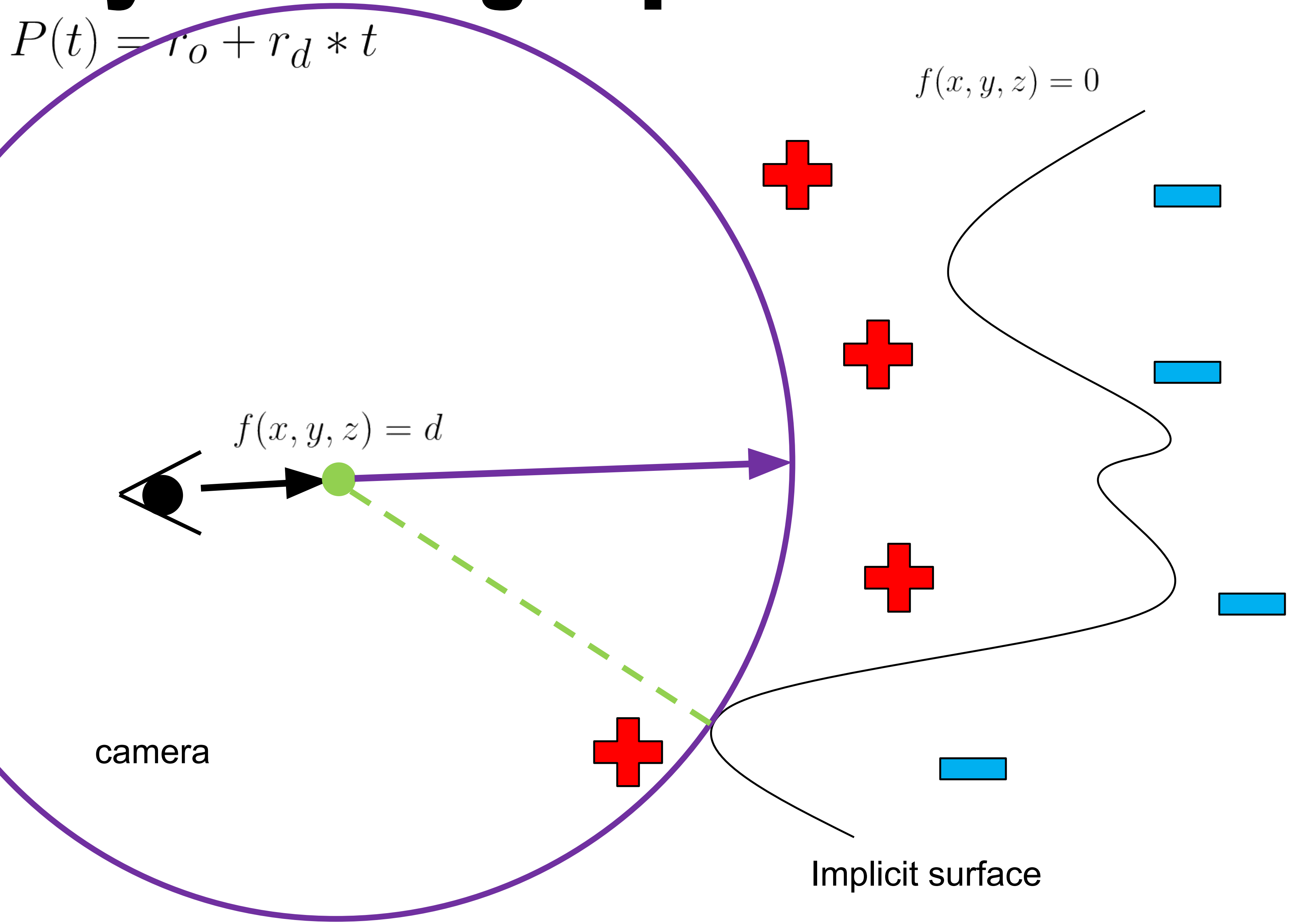
Ray marching implicit surfaces

$$P(t) = r_o + r_d * t$$



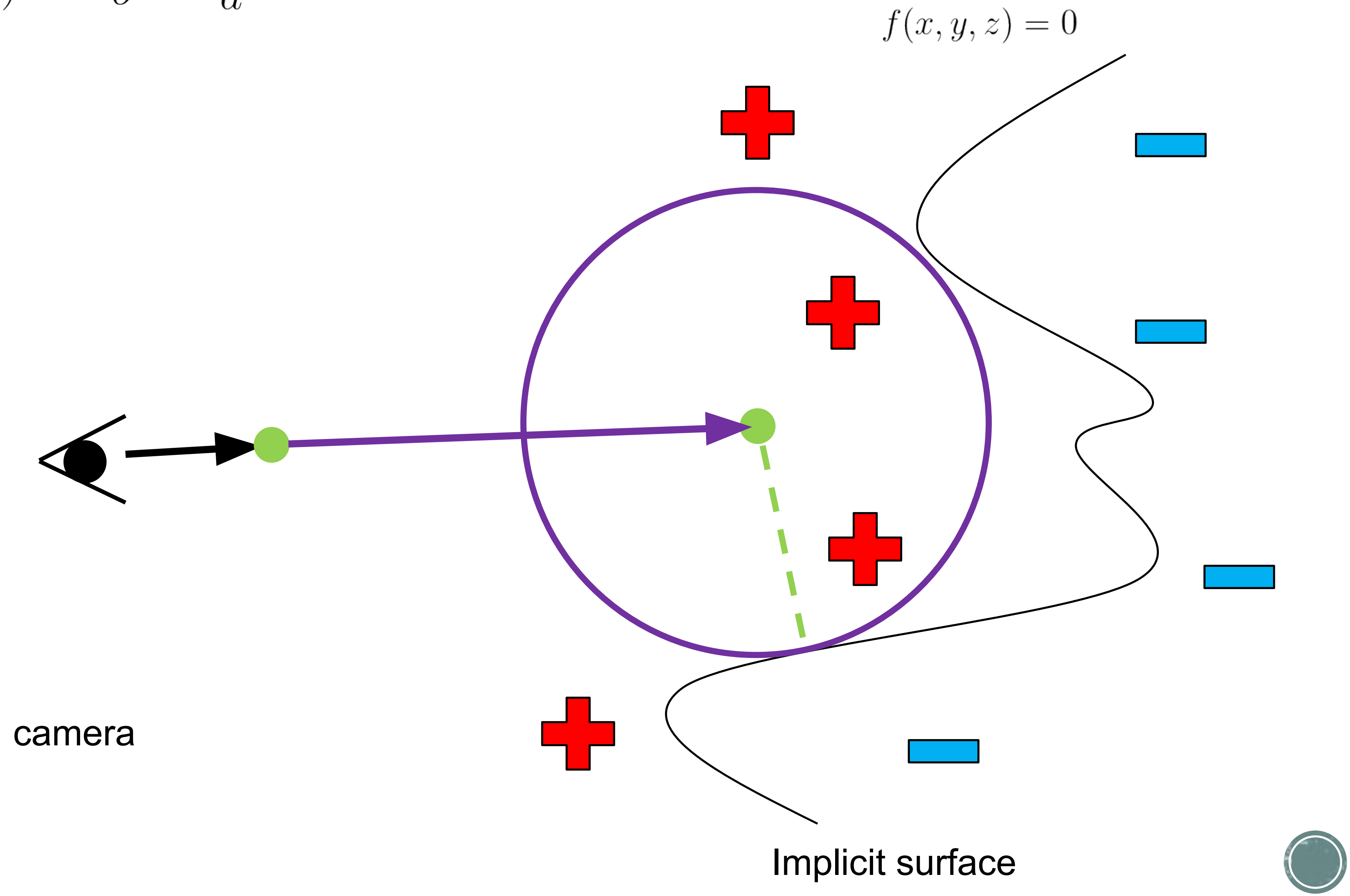
Ray marching implicit surfaces

$$P(t) = r_o + r_d * t$$



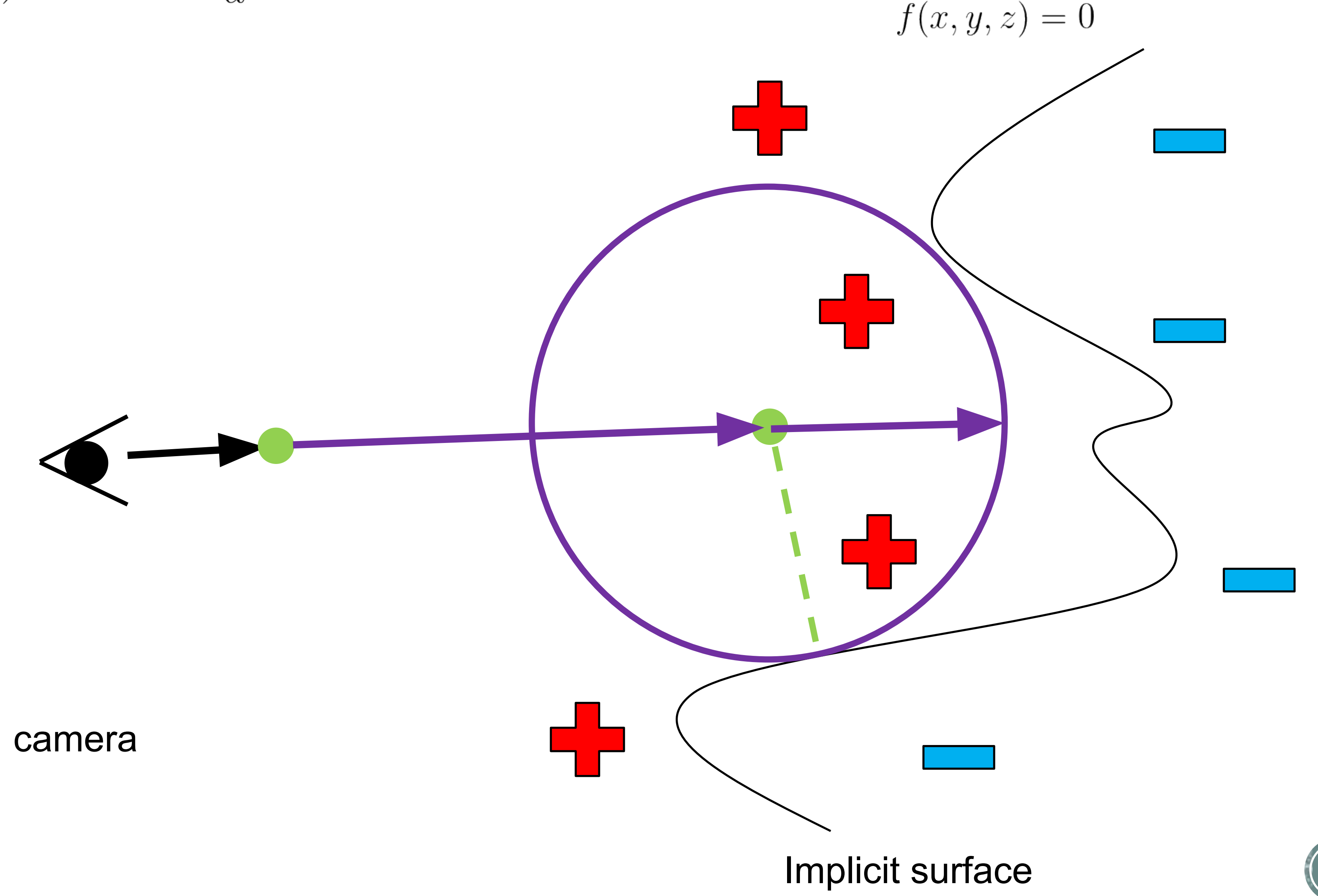
Ray marching implicit surfaces

$$P(t) = r_o + r_d * t$$



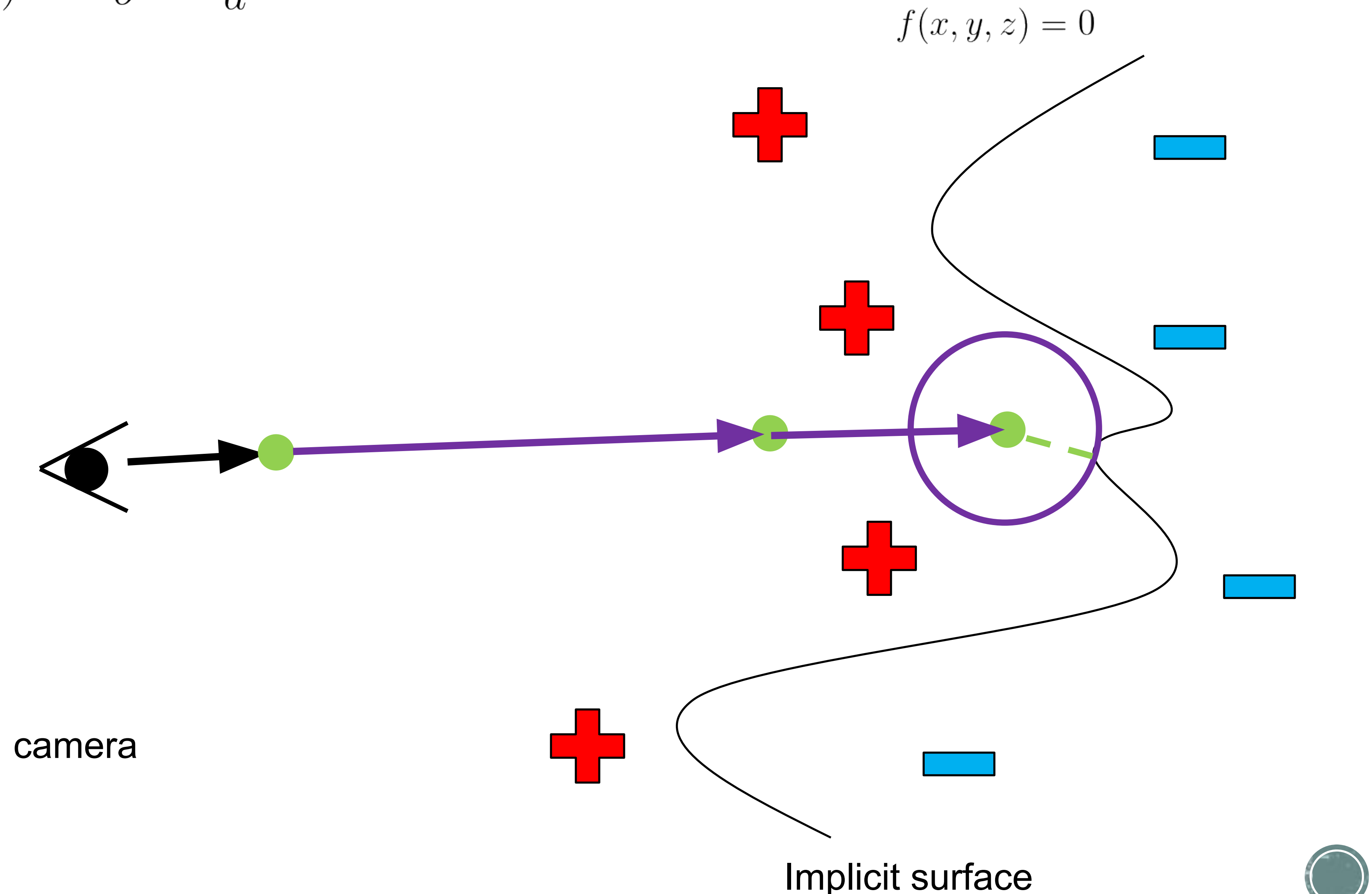
Ray marching implicit surfaces

$$P(t) = r_o + r_d * t$$



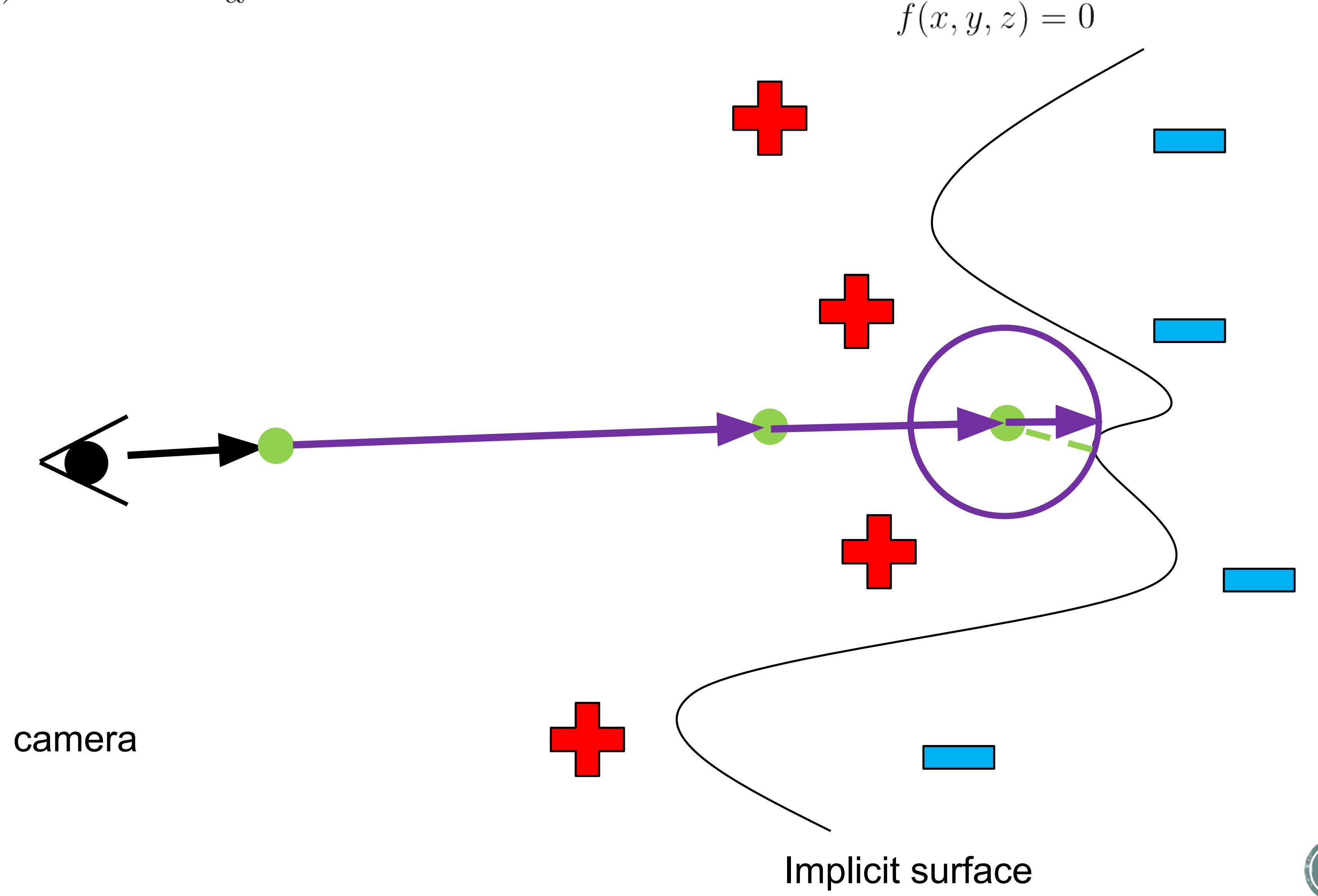
Ray marching implicit surfaces

$$P(t) = r_o + r_d * t$$



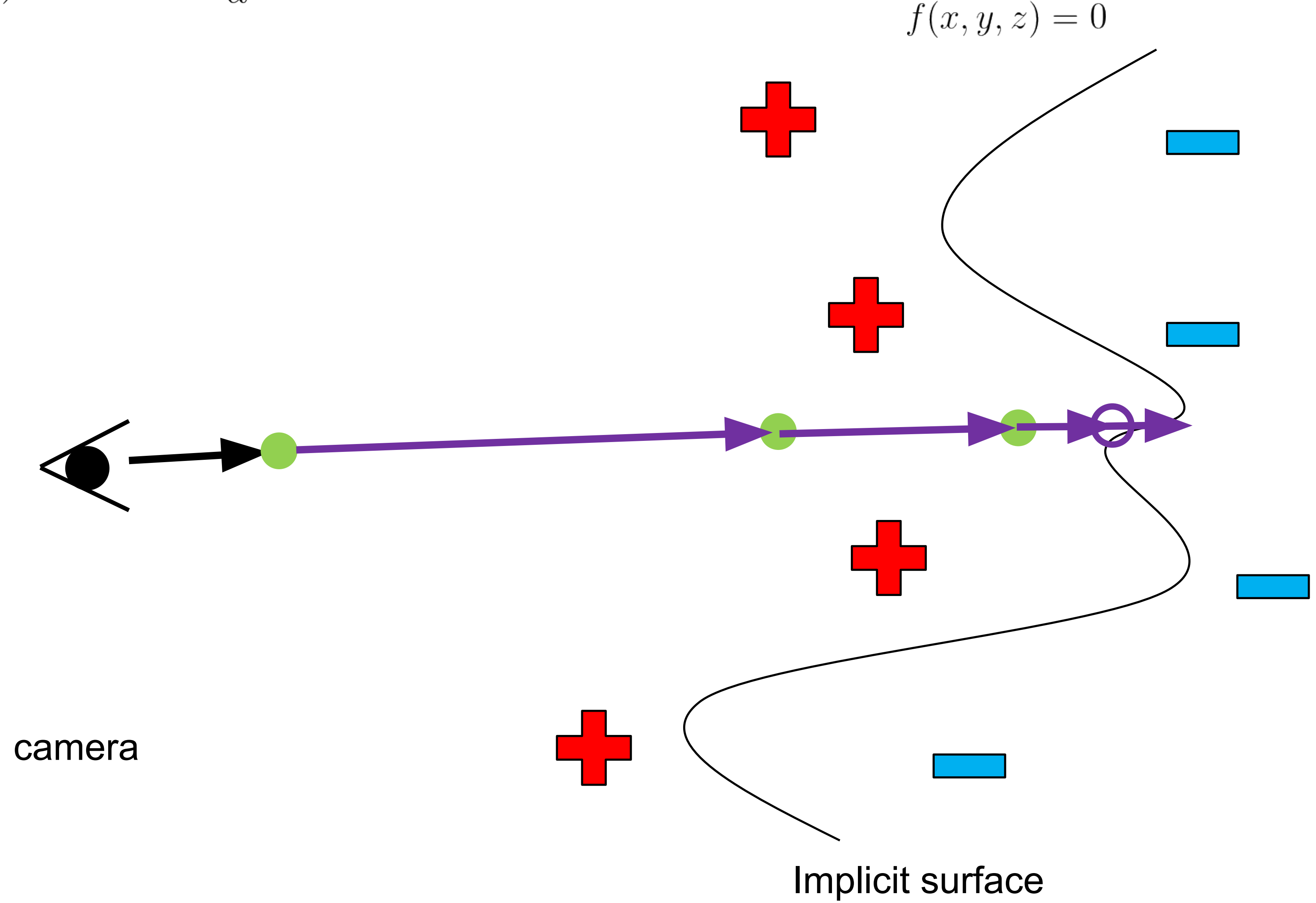
Ray marching implicit surfaces

$$P(t) = r_o + r_d * t$$



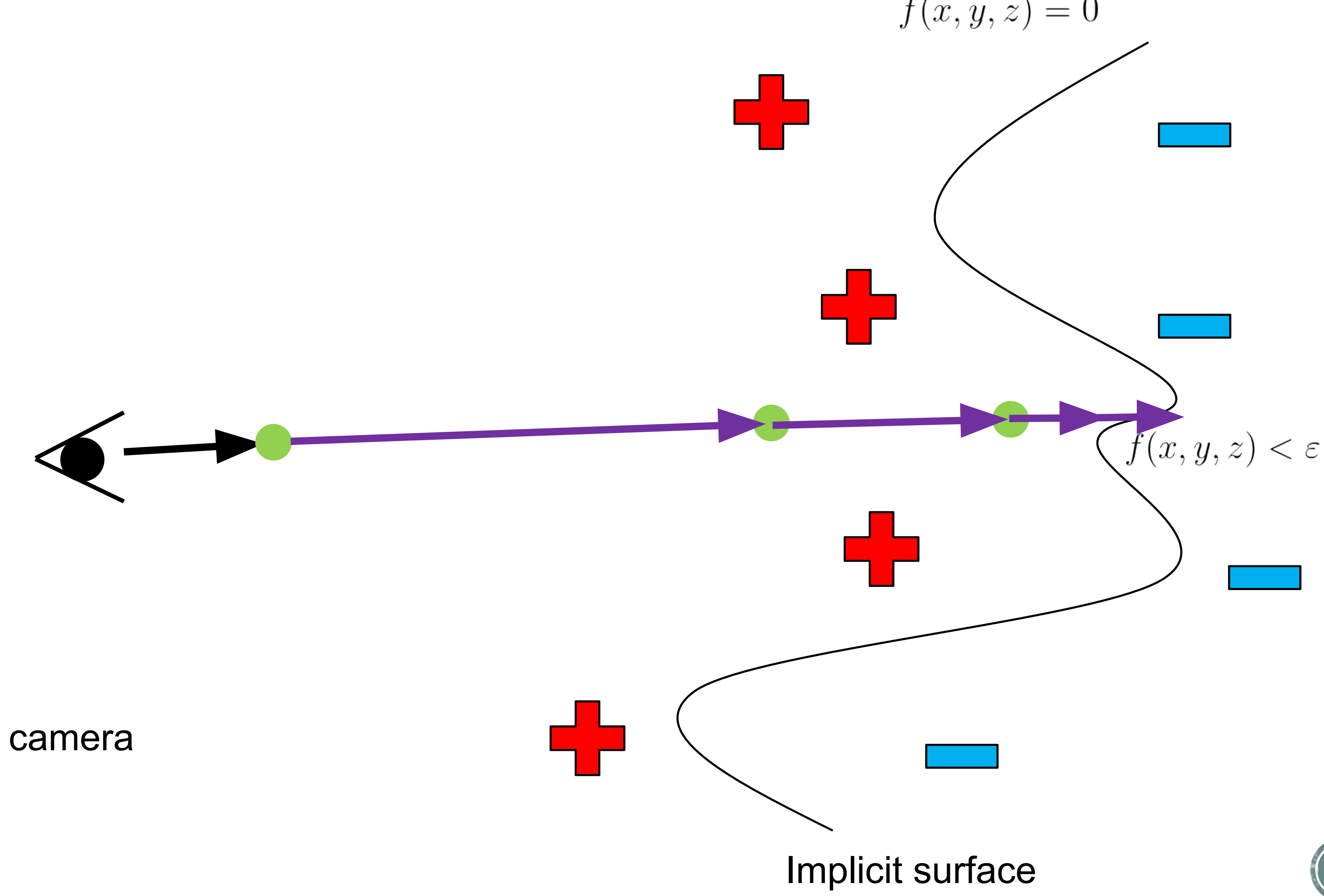
Ray marching implicit surfaces

$P(t) = r_o + r_d * t$



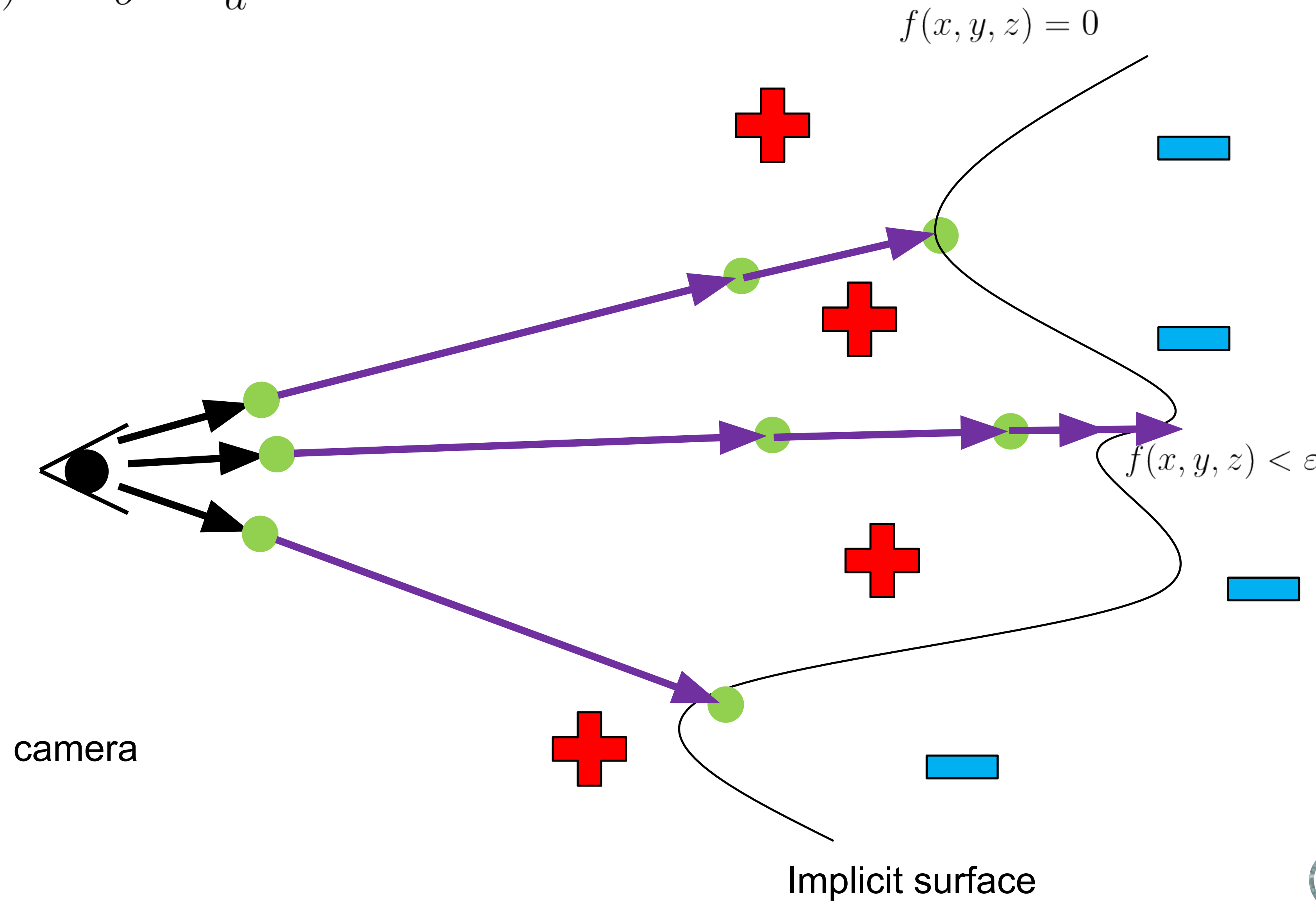
Ray marching implicit surfaces

$P(t) = r_o + r_d * t$



Ray marching implicit surfaces

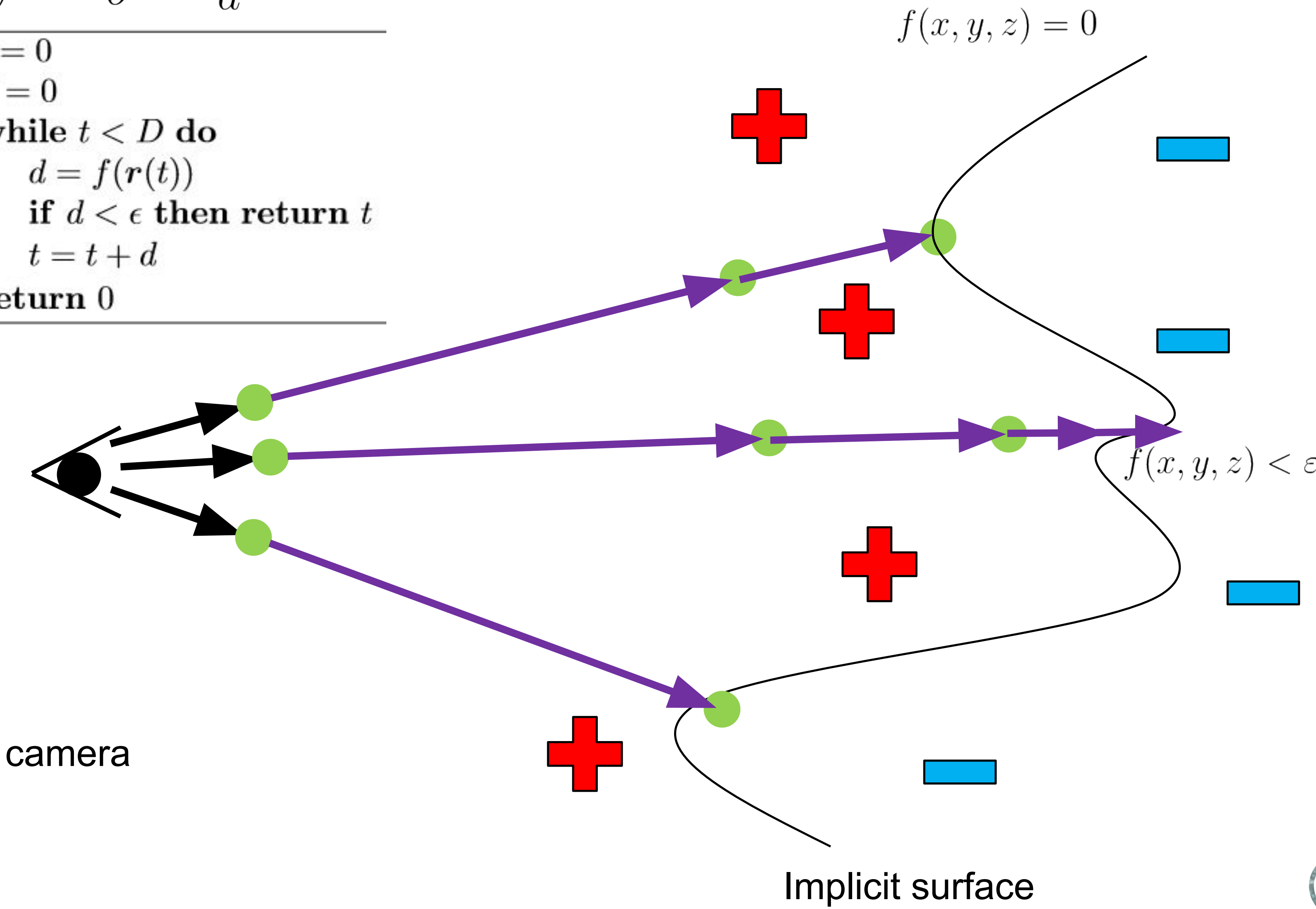
$$P(t) = r_o + r_d * t$$



Ray marching implicit surfaces

$$P(t) = r_o + r_d * t$$

$t = 0$
$d = 0$
while $t < D$ **do**
  $d = f(\boldsymbol{r}(t))$
  if $d < \epsilon$ **then return** t
  $t = t + d$
return 0



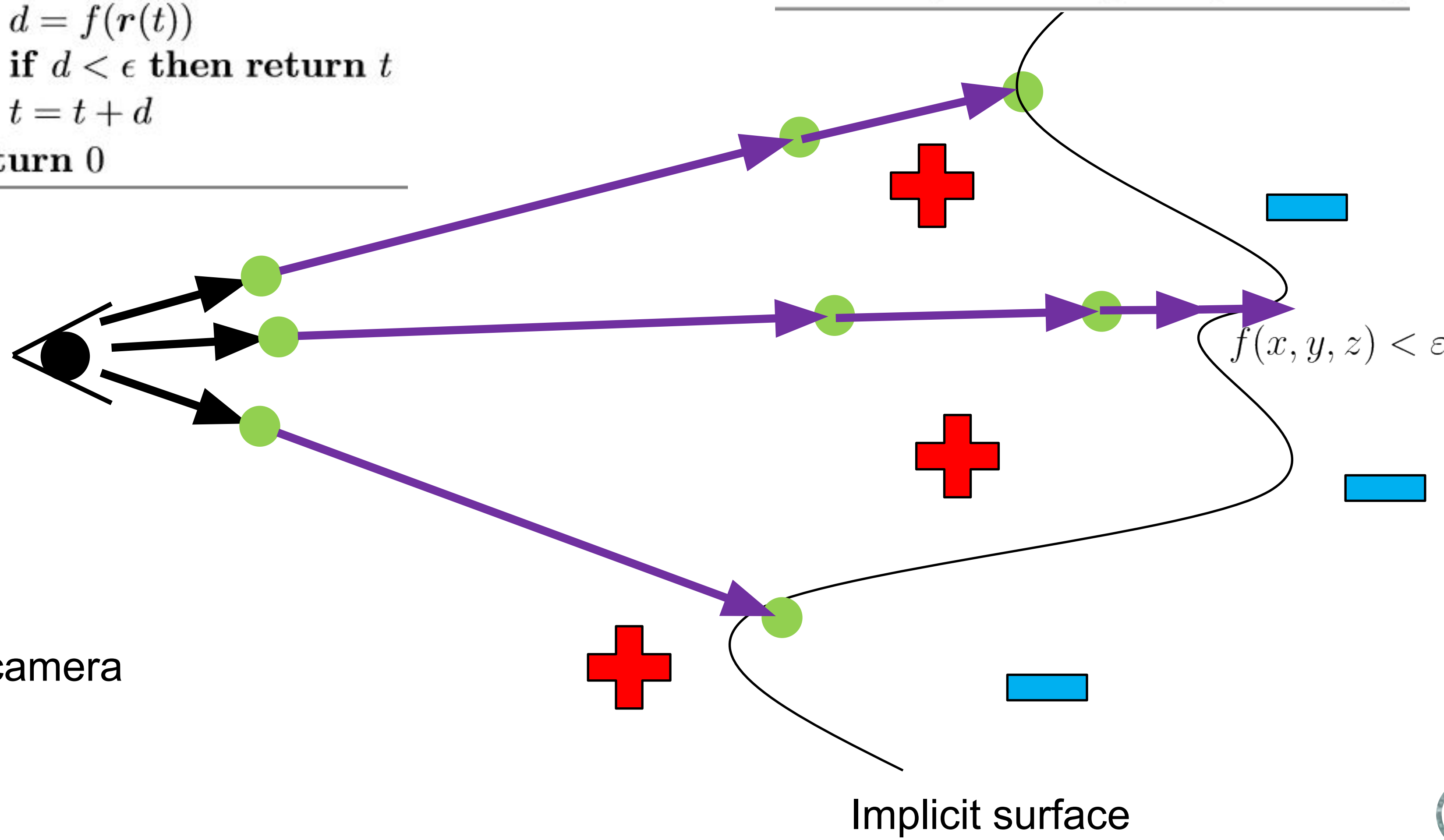
Ray marching implicit surfaces

$$P(t) = r_o + r_d * t$$

$t = 0$
$d = 0$
while $t < D$ **do**
  $d = f(\boldsymbol{r}(t))$
  if $d < \epsilon$ **then return** t
  $t = t + d$
return 0

Normal computation:

$$n_x = f(x+\epsilon, y, z) - f(x-\epsilon, y, z)$$
$$n_y = f(x, y+\epsilon, z) - f(x, y-\epsilon, z)$$
$$n_z = f(x, y, z+\epsilon) - f(x, y, z-\epsilon)$$



Common implicit surfaces

Sphere - signed

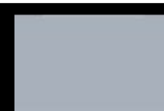
```
float sdSphere( vec3 p, float s )
{
  return length(p)-s;
}
```

Common implicit surfaces

Sphere - signed

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

Plane - signed



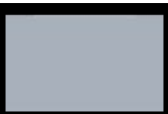
```
float sdPlane( vec3 p, vec4 n )
{
  // n must be normalized
  return dot(p,n.xyz) + n.w;
}
```

Common implicit surfaces

Sphere - signed

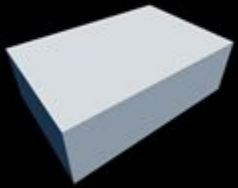
```
float sdSphere( vec3 p, float s )
{
  return length(p)-s;
}
```

Plane - signed



```
float sdPlane( vec3 p, vec4 n )
{
  // n must be normalized
  return dot(p,n.xyz) + n.w;
}
```

Box - signed



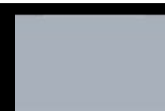
```
float sdBox( vec3 p, vec3 b )
{
  vec3 d = abs(p) - b;
  return min(max(d.x,max(d.y,d.z)),0.0) +
         length(max(d,0.0));
}
```

Common implicit surfaces

Sphere - signed

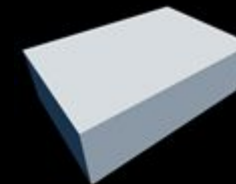
```
float sdSphere( vec3 p, float s )
{
  return length(p)-s;
}
```

Plane - signed



```
float sdPlane( vec3 p, vec4 n )
{
  // n must be normalized
  return dot(p,n.xyz) + n.w;
}
```

Box - signed



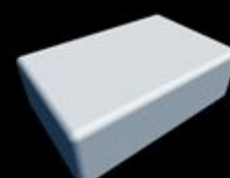
```
float sdBox( vec3 p, vec3 b )
{
  vec3 d = abs(p) - b;
  return min(max(d.x,max(d.y,d.z)),0.0) +
         length(max(d,0.0));
}
```

Torus - signed

```
float sdTorus( vec3 p, vec2 t )
{
  vec2 q = vec2(length(p.xz)-t.x,p.y);
  return length(q)-t.y;
}
```

Round Box - unsigned

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



Cone - signed

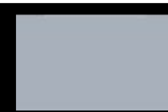
```
float sdCone( vec3 p, vec2 c )
{
    // c must be normalized
    float q = length(p.xy);
    return dot(c,vec2(q,p.z));
}
```

Common implicit surfaces

Sphere - signed

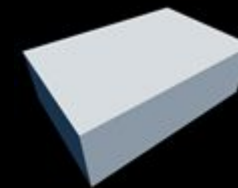
```
float sdSphere( vec3 p, float s )
{
  return length(p)-s;
}
```

Plane - signed



```
float sdPlane( vec3 p, vec4 n )
{
  // n must be normalized
  return dot(p,n.xyz) + n.w;
}
```

Box - signed



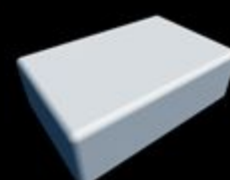
```
float sdBox( vec3 p, vec3 b )
{
  vec3 d = abs(p) - b;
  return min(max(d.x,max(d.y,d.z)),0.0) +
         length(max(d,0.0));
}
```

Torus - signed

```
float sdTorus( vec3 p, vec2 t )
{
  vec2 q = vec2(length(p.xz)-t.x,p.y);
  return length(q)-t.y;
}
```

Round Box - unsigned

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



Cone - signed

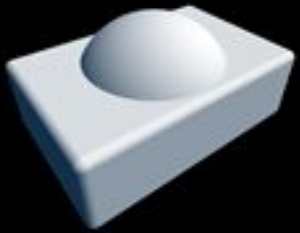
```
float sdCone( vec3 p, vec2 c )
{
    // c must be normalized
    float q = length(p.xy);
    return dot(c,vec2(q,p.z));
}
```

Complete list: http://www.iquilezles.org/www/articles/distfunctions/distfunctions.htm

Distance operations

Union

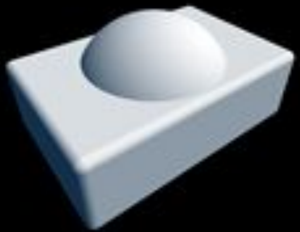
```
float opU( float d1, float d2 )
{
    return min(d1,d2);
}
```



Complete list: http://www.iquilezles.org/www/articles/distfunctions/distfunctions.htm

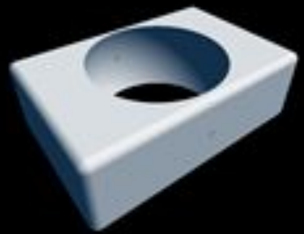
Distance operations

Union



```
float opU( float d1, float d2 )
{
    return min(d1,d2);
}
```

Substraction



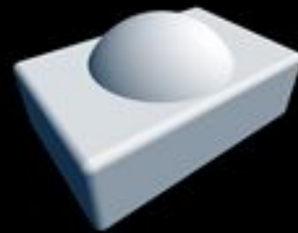
```
float opS( float d1, float d2 )
{
    return max(-d1,d2);
}
```

Complete list: http://www.iquilezles.org/www/articles/distfunctions/distfunctions.htm

Distance operations

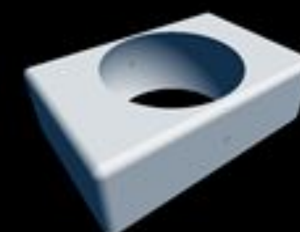
Union

```
float opU( float d1, float d2 )
{
    return min(d1,d2);
}
```



Substraction

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



Intersection

```
float opI( float d1, float d2 )
{
    return max(d1,d2);
}
```

Complete list: http://www.iquilezles.org/www/articles/distfunctions/distfunctions.htm

Domain operations

Repetition

```
float opRep( vec3 p, vec3 c )
{
    vec3 q = mod(p,c)-0.5*c;
    return primitve( q );
}
```

Complete list: http://www.iquilezles.org/www/articles/distfunctions/distfunctions.htm

Domain operations

Repetition

```
float opRep( vec3 p, vec3 c )
{
    vec3 q = mod(p,c)-0.5*c;
    return primitve( q );
}
```

Scale

```
float opScale( vec3 p, float s )
{
    return primitive(p/s)*s;
}
```

Complete list: http://www.iquilezles.org/www/articles/distfunctions/distfunctions.htm

Domain operations

Repetition

```
float opRep( vec3 p, vec3 c )
{
    vec3 q = mod(p,c)-0.5*c;
    return primitve( q );
}
```

Scale

```
float opScale( vec3 p, float s )
{
    return primitive(p/s)*s;
}
```

Rotation/Translation

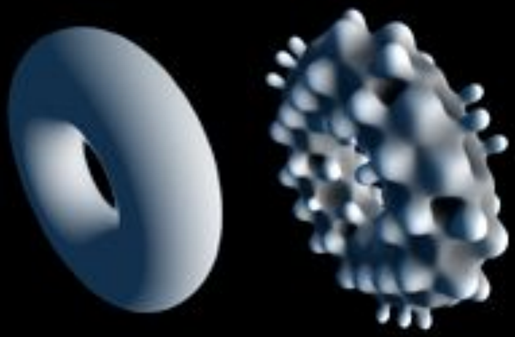
```
vec3 opTx( vec3 p, mat4 m )
{
    vec3 q = invert(m)*p;
    return primitive(q);
}
```

Complete list: http://www.iquilezles.org/www/articles/distfunctions/distfunctions.htm

Distance deformations

Displacement

```
float opDisplace( vec3 p )
{
    float d1 = primitive(p);
    float d2 = displacement(p);
    return d1+d2;
}
```

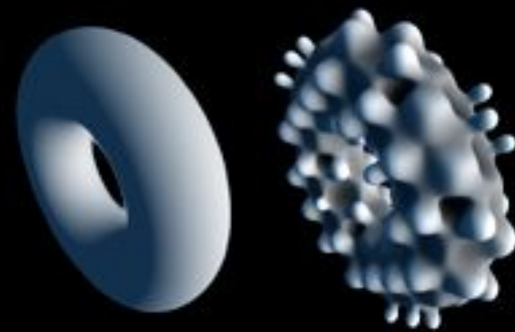


Complete list: http://www.iquilezles.org/www/articles/distfunctions/distfunctions.htm

Distance deformations

Displacement

```
float opDisplace( vec3 p )
{
    float d1 = primitive(p);
    float d2 = displacement(p);
    return d1+d2;
}
```



Blend

```
float opBlend( vec3 p )
{
    float d1 = primitiveA(p);
    float d2 = primitiveB(p);
    return smin( d1, d2 );
}
```

```
// polynomial smooth min (k = 0.1);
float smin( float a, float b, float k )
{
    float h = clamp( 0.5+0.5*(b-a)/k, 0.0, 1.0 );
    return mix( b, a, h ) - k*h*(1.0-h);
}
```

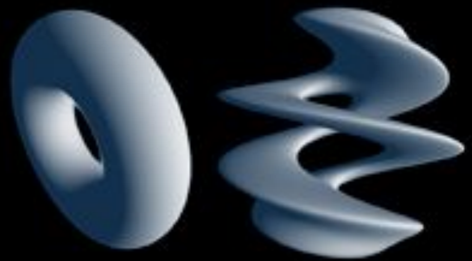
for instance…

Complete list: http://www.iquilezles.org/www/articles/distfunctions/distfunctions.htm

Domain deformations

Twist

```
float opTwist( vec3 p )
{
    float c = cos(20.0*p.y);
    float s = sin(20.0*p.y);
    mat2  m = mat2(c,-s,s,c);
    vec3  q = vec3(m*p.xz,p.y);
    return primitive(q);
}
```

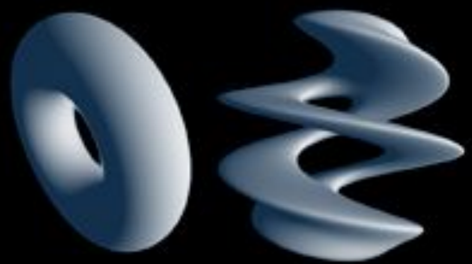


Complete list: http://www.iquilezles.org/www/articles/distfunctions/distfunctions.htm

Domain deformations

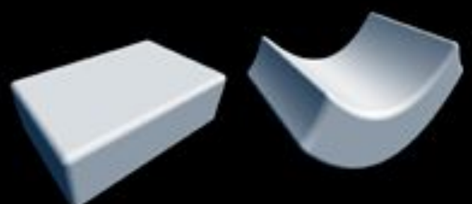
Twist

```
float opTwist( vec3 p )
{
    float c = cos(20.0*p.y);
    float s = sin(20.0*p.y);
    mat2  m = mat2(c,-s,s,c);
    vec3  q = vec3(m*p.xz,p.y);
    return primitive(q);
}
```



Cheap Bend

```
float opCheapBend( vec3 p )
{
    float c = cos(20.0*p.y);
    float s = sin(20.0*p.y);
    mat2  m = mat2(c,-s,s,c);
    vec3  q = vec3(m*p.xy,p.z);
    return primitive(q);
}
```



Complete list: http://www.iquilezles.org/www/articles/distfunctions/distfunctions.htm

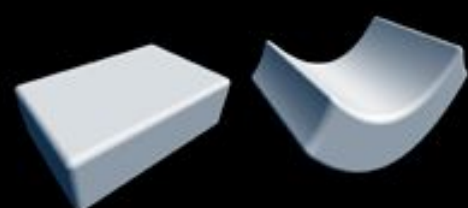
Domain deformations

Twist

```
float opTwist( vec3 p )
{
    float c = cos(20.0*p.y);
    float s = sin(20.0*p.y);
    mat2  m = mat2(c,-s,s,c);
    vec3  q = vec3(m*p.xz,p.y);
    return primitive(q);
}
```

Cheap Bend

```
float opCheapBend( vec3 p )
{
    float c = cos(20.0*p.y);
    float s = sin(20.0*p.y);
    mat2  m = mat2(c,-s,s,c);
    vec3  q = vec3(m*p.xy,p.z);
    return primitive(q);
}
```



Demos:
https://www.shadertoy.com/view/Xds3zN
https://www.shadertoy.com/view/Mss3zM

Complete list: http://www.iquilezles.org/www/articles/distfunctions/distfunctions.htm

References

- MIT:
  - http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-837-computer-graphics-fall-2012/lecture-notes/
- Standford:
  - http://candela.stanford.edu/cs348b-14/doku.php
- Siggraph:
  - http://blog.selfshadow.com/publications/s2014-shading-course/
  - http://blog.selfshadow.com/publications/s2013-shading-course/
- Image synthesis & OpenGL:
  - http://romain.vergne.free.fr/blog/?page_id=97
- Path tracing and global illum:
  - http://www.graphics.stanford.edu/courses/cs348b-01/course29.hanrahan.pdf
  - http://web.cs.wpi.edu/~emmanuel/courses/cs563/write_ups/zackw/realistic_raytracing.html
- GLSL / Shadertoy:
  - https://www.opengl.org/documentation/glsl/
  - https://www.shadertoy.com/
  - http://www.iquilezles.org/
- http://fileadmin.cs.lth.se/cs/Education/EDAN30/lectures/L2-rt.pdf
- http://csokavar.hu/raytrace/imm6392.pdf