Drawing 3D Fractals in Real Time

An Introduction to Distance Estimators

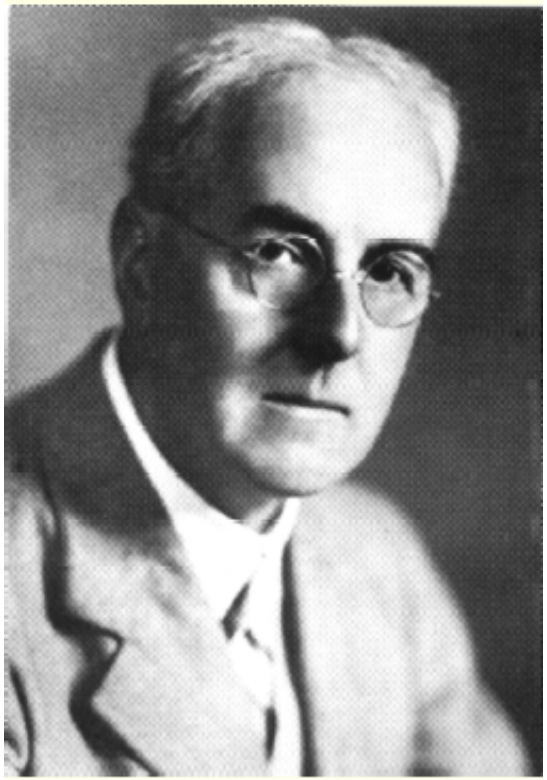
Giovanni Viglietta

JAIST – March 19, 2019

Outline

- Fractals in nature
- Fractal dimension
- The Mandelbrot set
- Rendering 3D objects
- Distance estimators
- Generalizing the complex numbers
- Rendering 3D Mandelbrot-like sets
- Open problems

The coastline paradox



The English mathematician Lewis F. Richardson (1881–1953), in his analysis of war, searched for a relation between the probability of two countries going to war and the length of their common border. However, in collecting data, he found considerable disagreements in the various published border lengths.

The coastline paradox

He then measured the length of the coast of Britain, and noticed that the length increases as the "ruler" gets shorter (e.g., 2300 km, 2800 km, 3500 km).

Can we take the limit as the real length? No: Richardson showed that the coast length diverges as the ruler gets shorter!

Similar paradoxes in nature

Many other objects in nature exhibit the same paradox:
frost crystals, broccoli, the human brain...

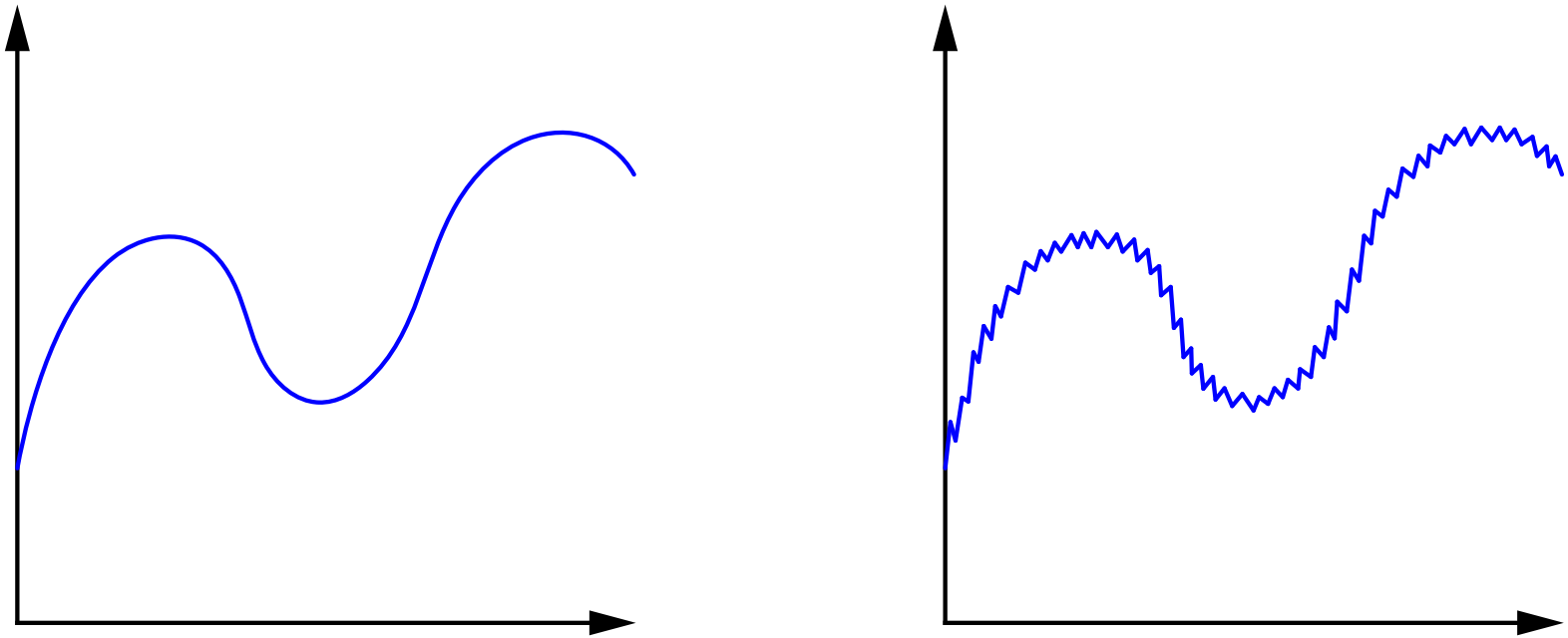
Similar paradoxes in nature

Many other objects in nature exhibit the same paradox:
frost crystals, broccoli, the human brain...

Similar paradoxes in nature

Many other objects in nature exhibit the same paradox:
frost crystals, broccoli, the human brain...

Traditional calculus

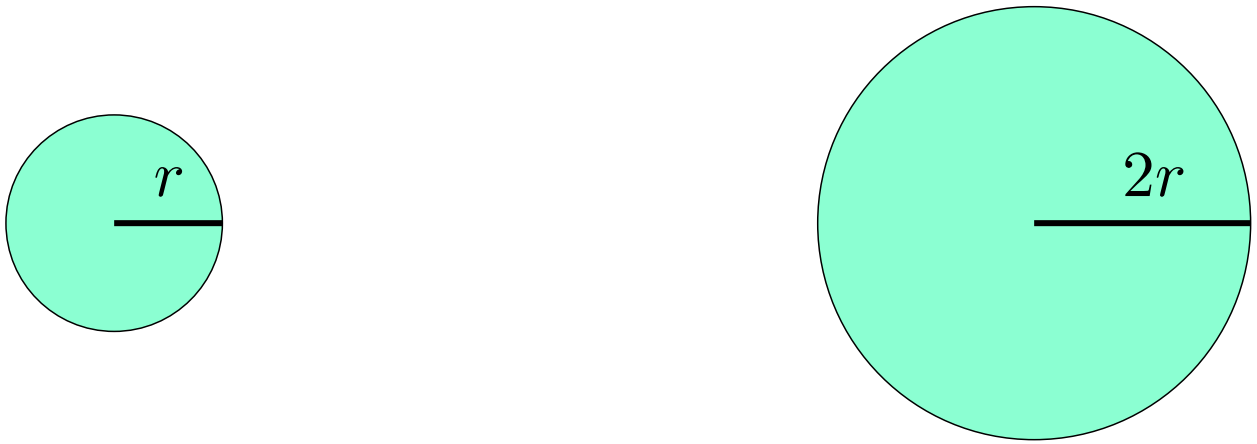


The central assumption of traditional calculus is that objects tend to look smooth if you zoom in far enough. This idealized notion does not capture the roughness of most real objects.

Fractal dimension

Between 1967 and 1975, Benoit Mandelbrot synthesized centuries of observations on undifferentiable, infinitely self-similar functions, formulating the notion of fractal dimension. It describes how a shape scales differently from the space in which it is embedded.

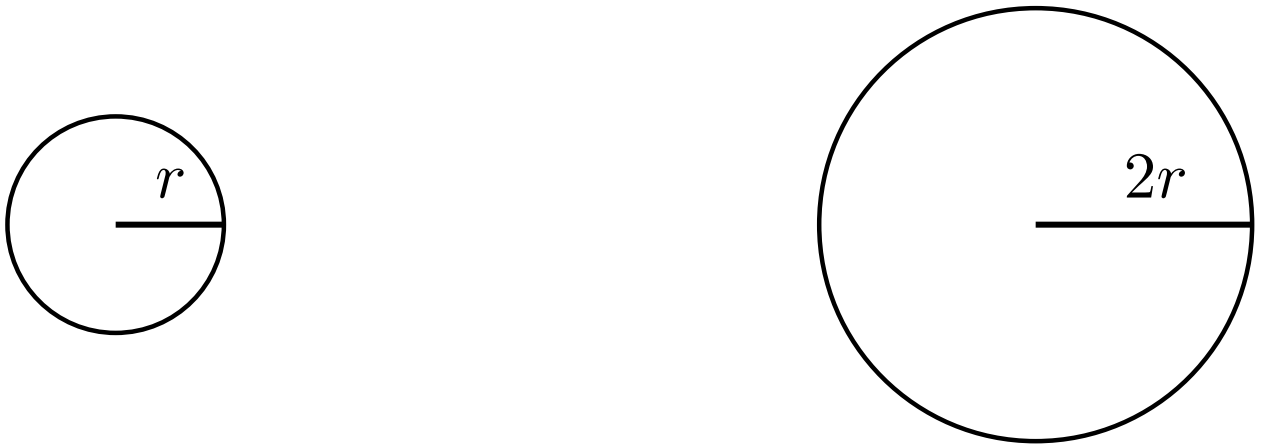
Fractal dimension



If a disk is scaled by a factor of 2, its area goes from πr^2 to $4\pi r^2$.

Its fractal dimension is $\log_2 4 = 2$.

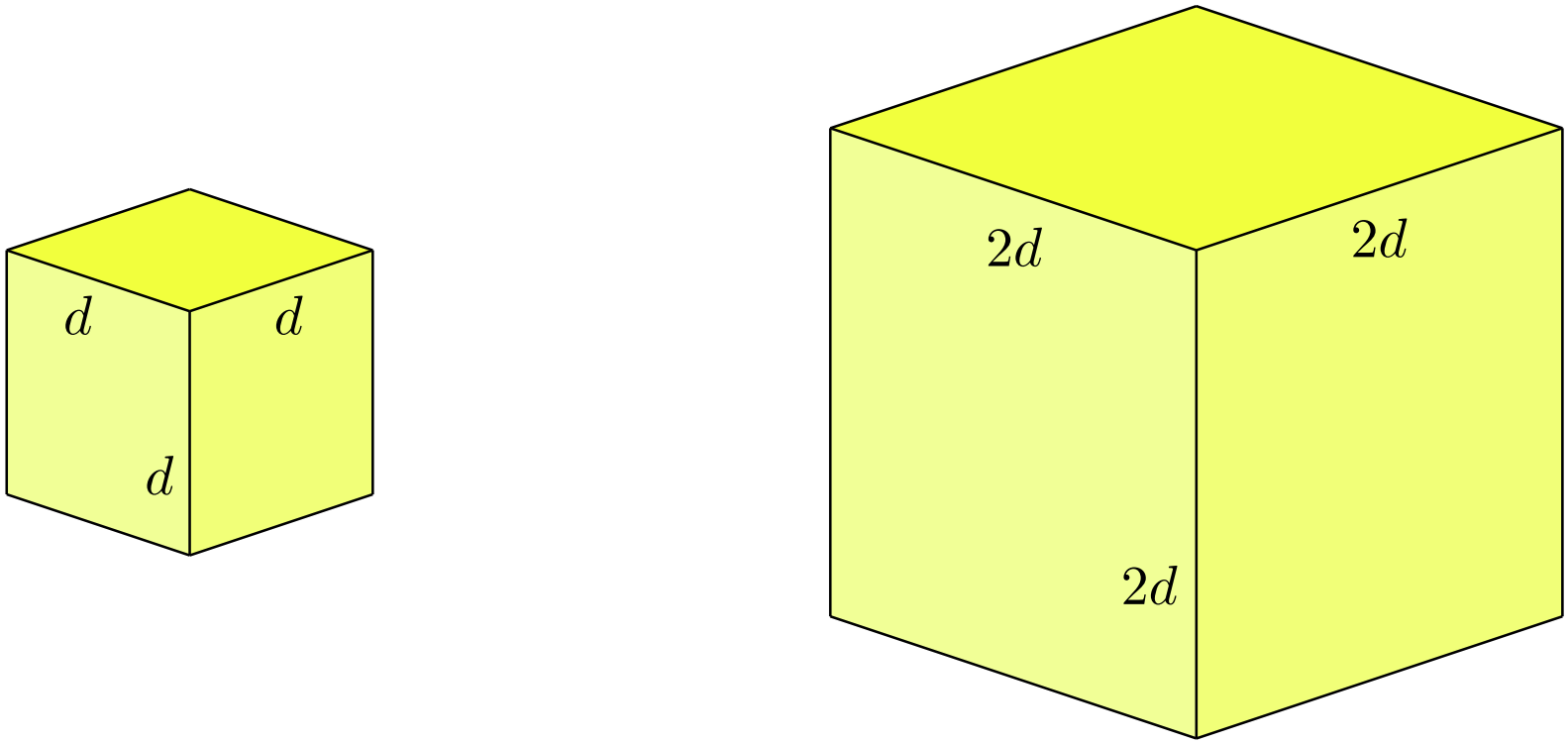
Fractal dimension



If a circle is scaled by 2, its length goes from $2\pi r$ to $2 \cdot 2\pi r$.

Its fractal dimension is $\log_2 2 = 1$.

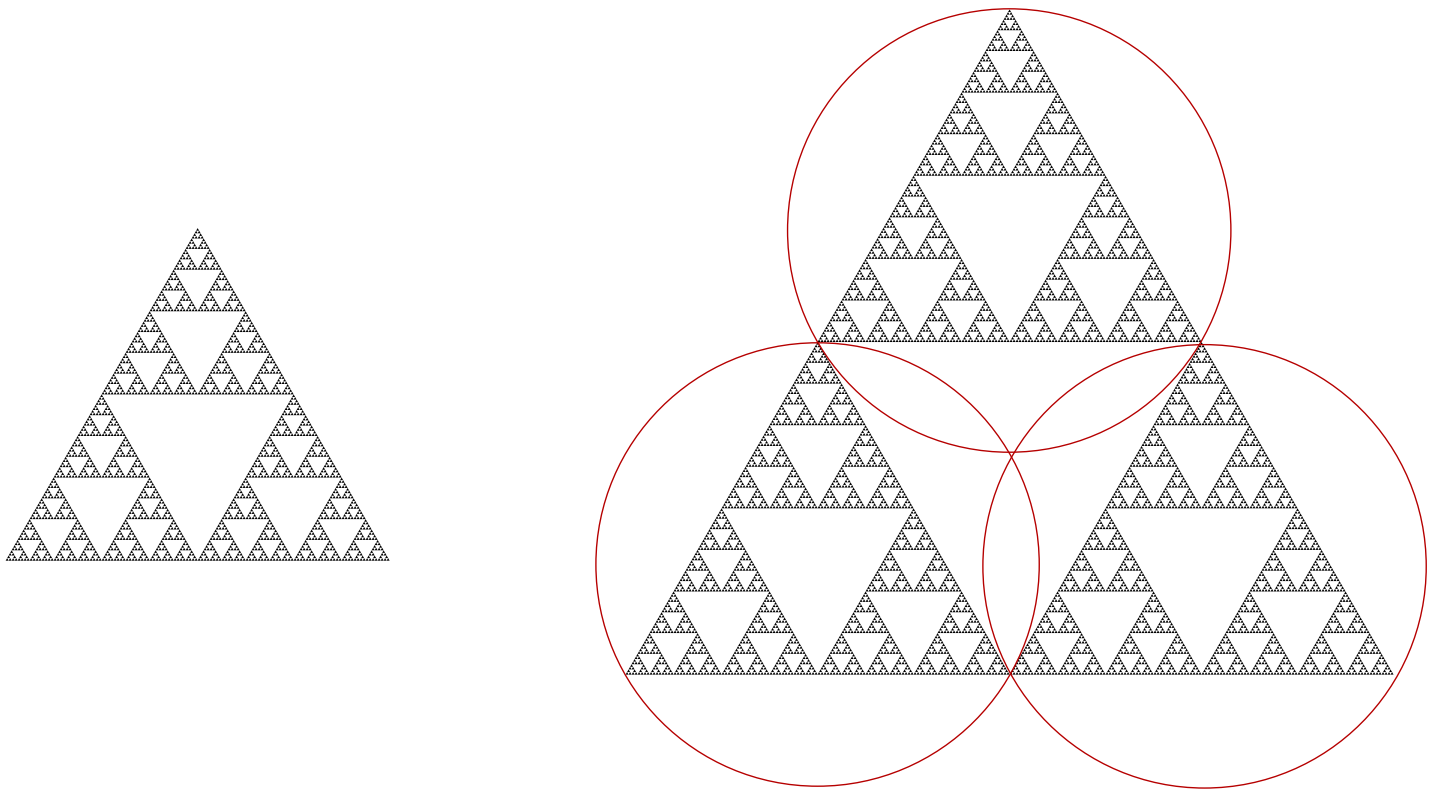
Fractal dimension



If a cube is scaled by a factor of 2, its volume goes from d^3 to $8d^3$.

Its fractal dimension is $\log_2 8 = 3$.

Fractal dimension

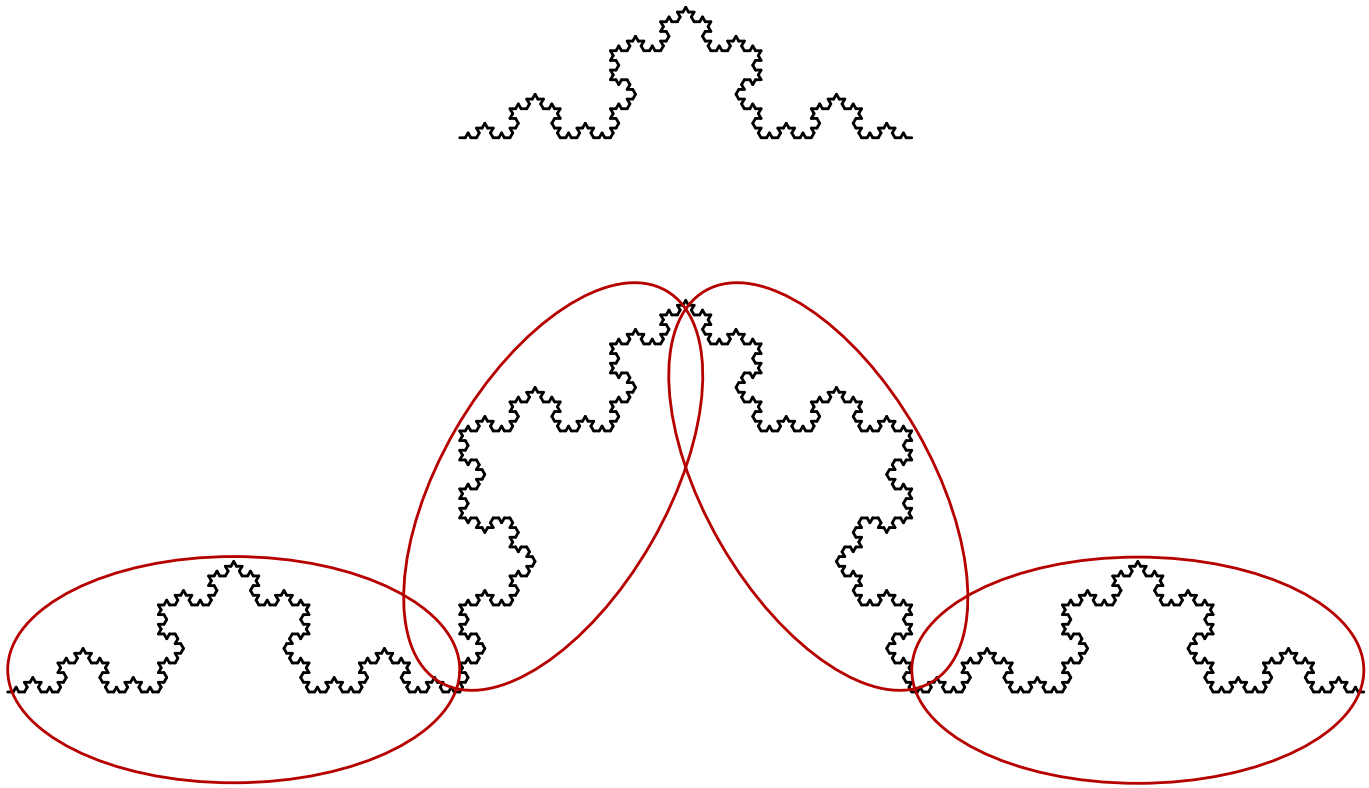


If a Sierpinski triangle is scaled by 2, its area is scaled by 3.

Its fractal dimension is $\log_2 3 \approx 1.585$.

Its dimension is not an integer, hence it is a fractal!

Fractal dimension



If a Koch snowflake is scaled by 3, its length is scaled by 4.

Its fractal dimension is $\log_3 4 \approx 1.262$.

Its dimension is not an integer, hence it is a fractal!

Fractal dimension

Approximate fractal dimensions of real objects:

- Coast of Britain: ~ 1.25
- Coast of Norway: ~ 1.52
- Outline of clouds: ~ 1.6
- Surface of oceans: ~ 2.3
- Surface of broccoli: ~ 2.7
- Surface of human brain: ~ 2.79
- Surface of human lungs: ~ 2.97

The Mandelbrot set

For each complex number c, define the function $f_c(z) = z^2 + c$.

Consider the iterations

$f_c(0)$,

$f_c(f_c(0))$,

$f_c(f_c(f_c(0)))$,

$f_c(f_c(f_c(f_c(0))))$,

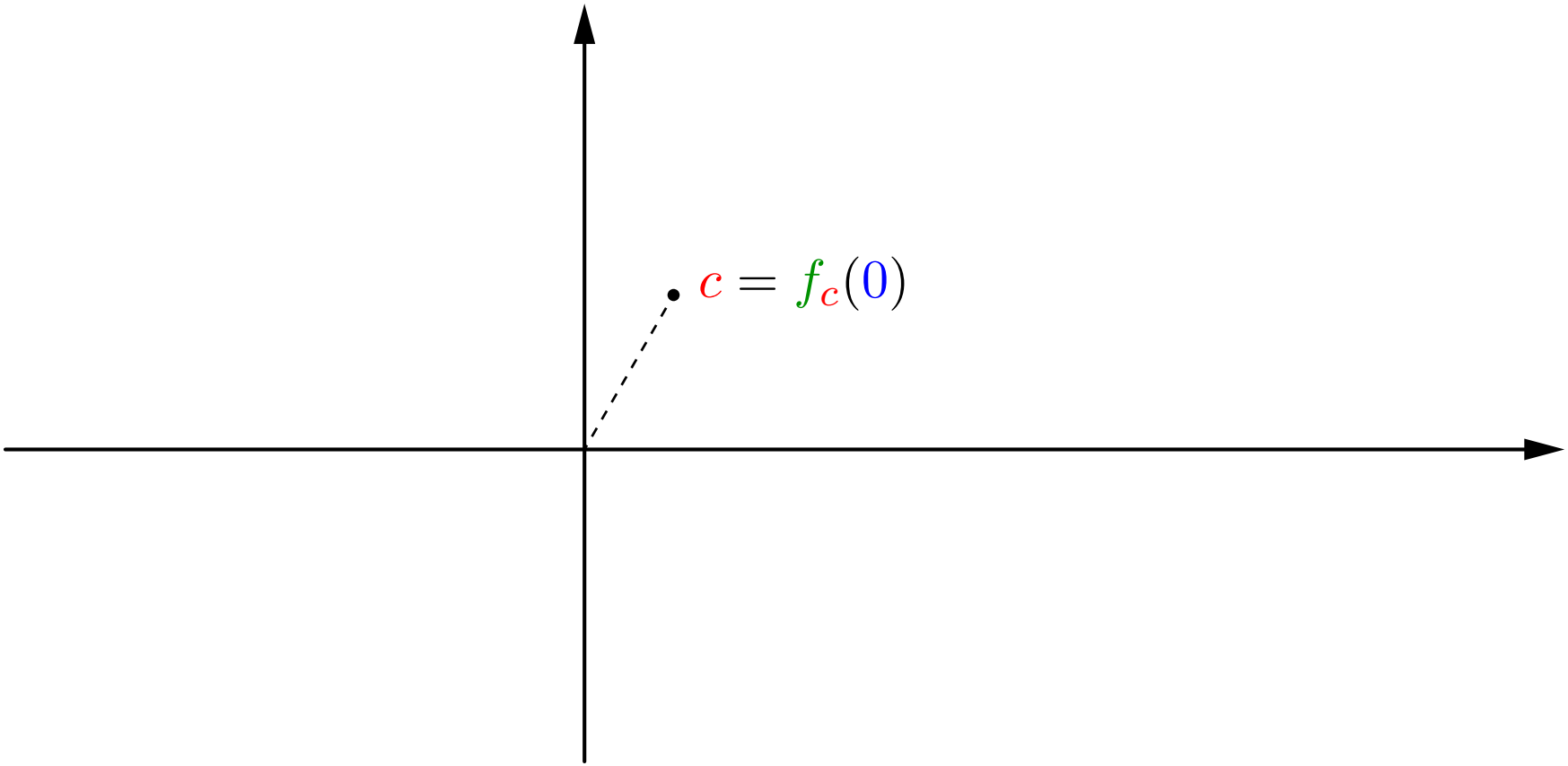
$\vdots$

$f_c^n(0)$.

Definition

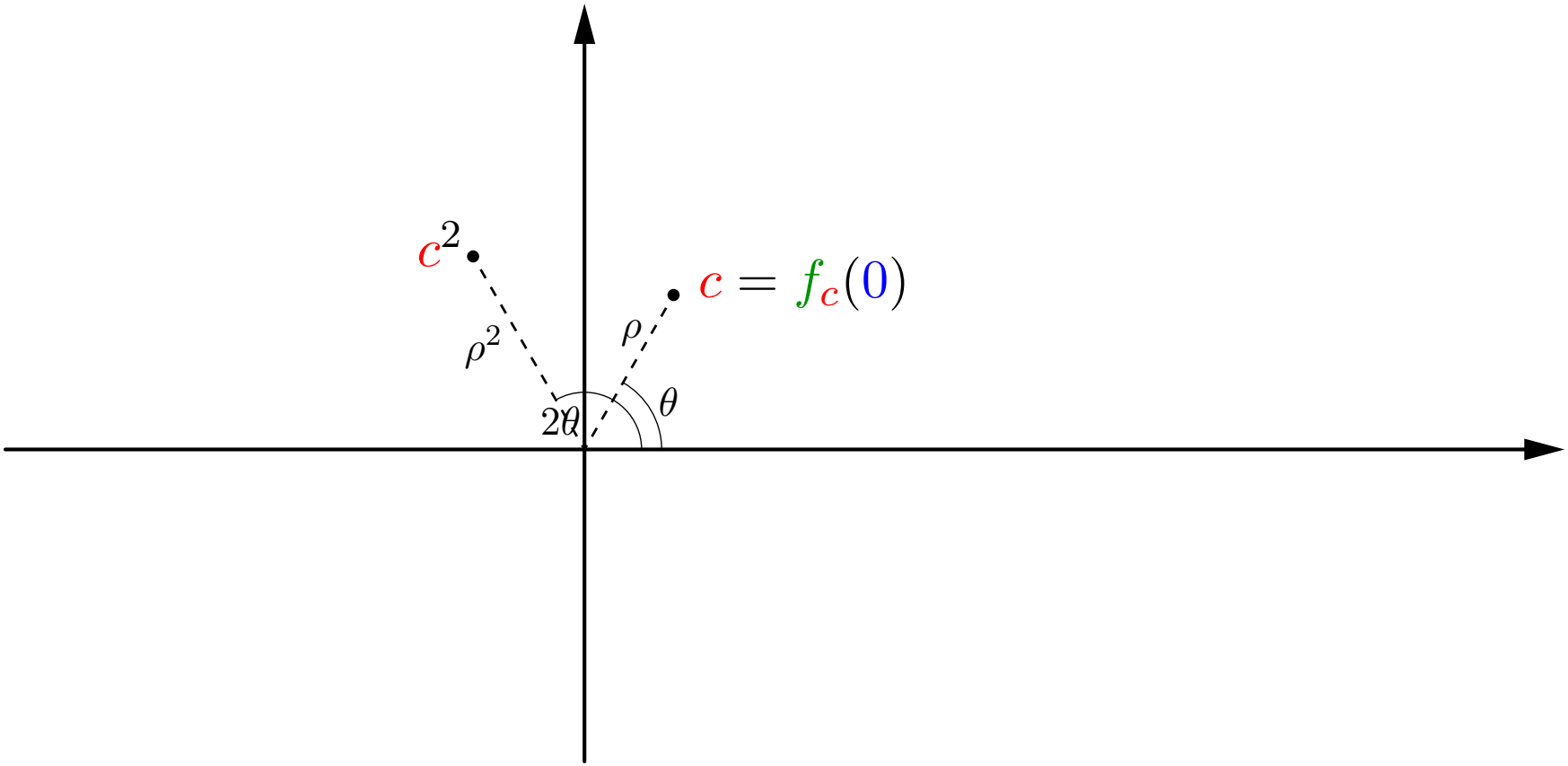
The complex number c is in the Mandelbrot set if and only if the sequence of moduli $|f_c^n(0)|$ is bounded.

The Mandelbrot set



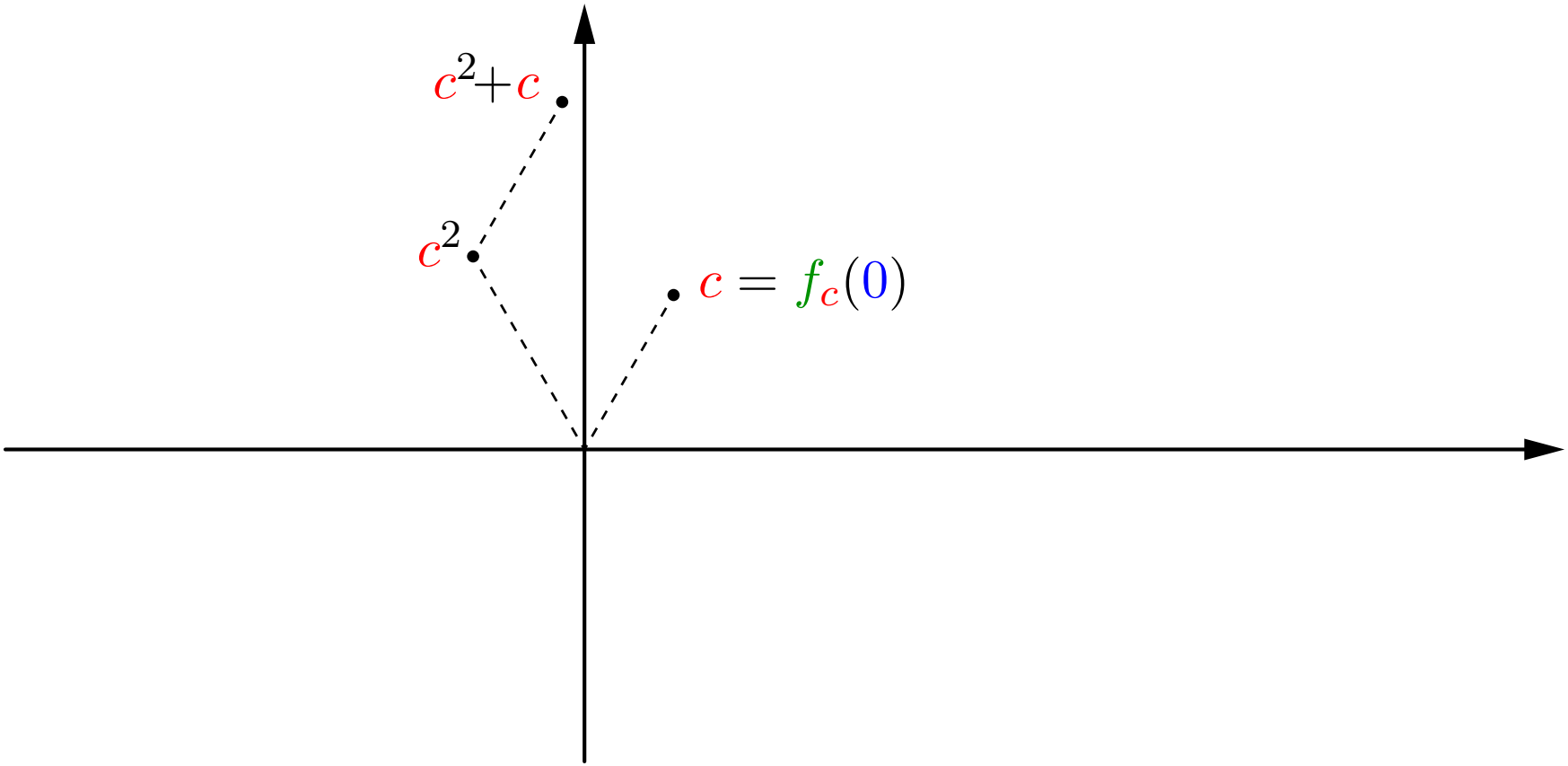
Is c in the Mandelbrot set?

The Mandelbrot set



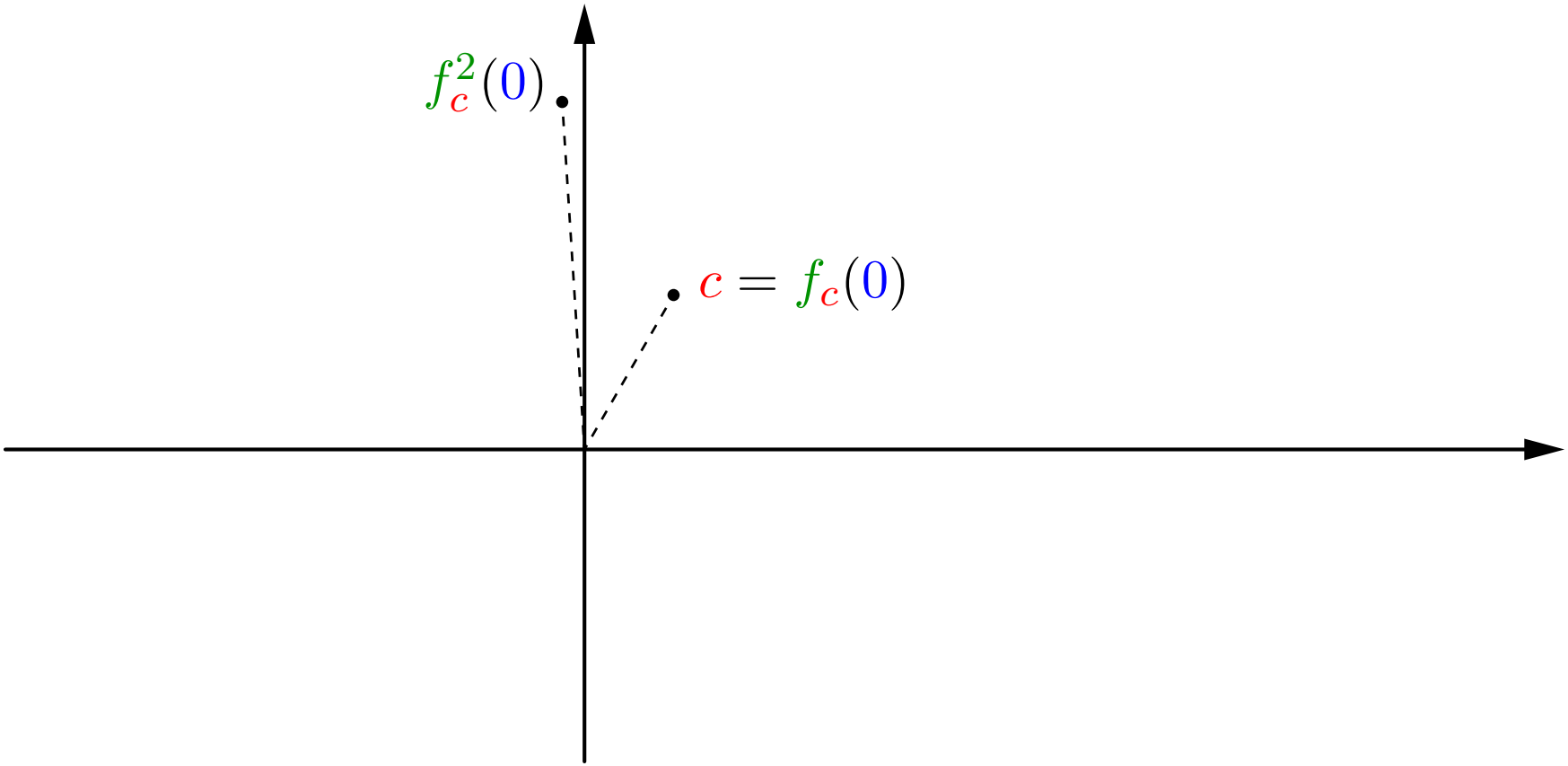
Let us iterate f_c on 0...

The Mandelbrot set



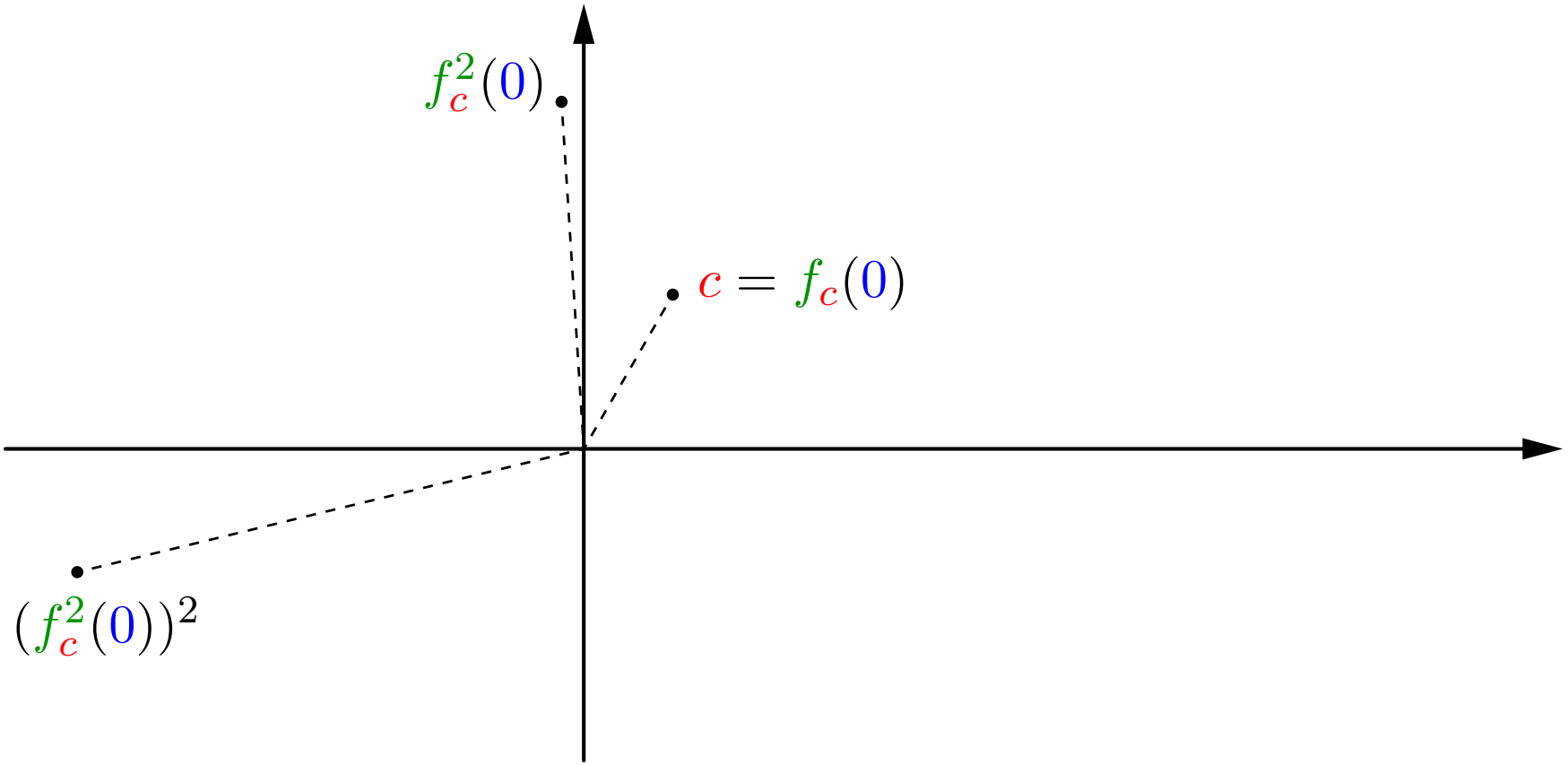
Let us iterate f_c on 0...

The Mandelbrot set



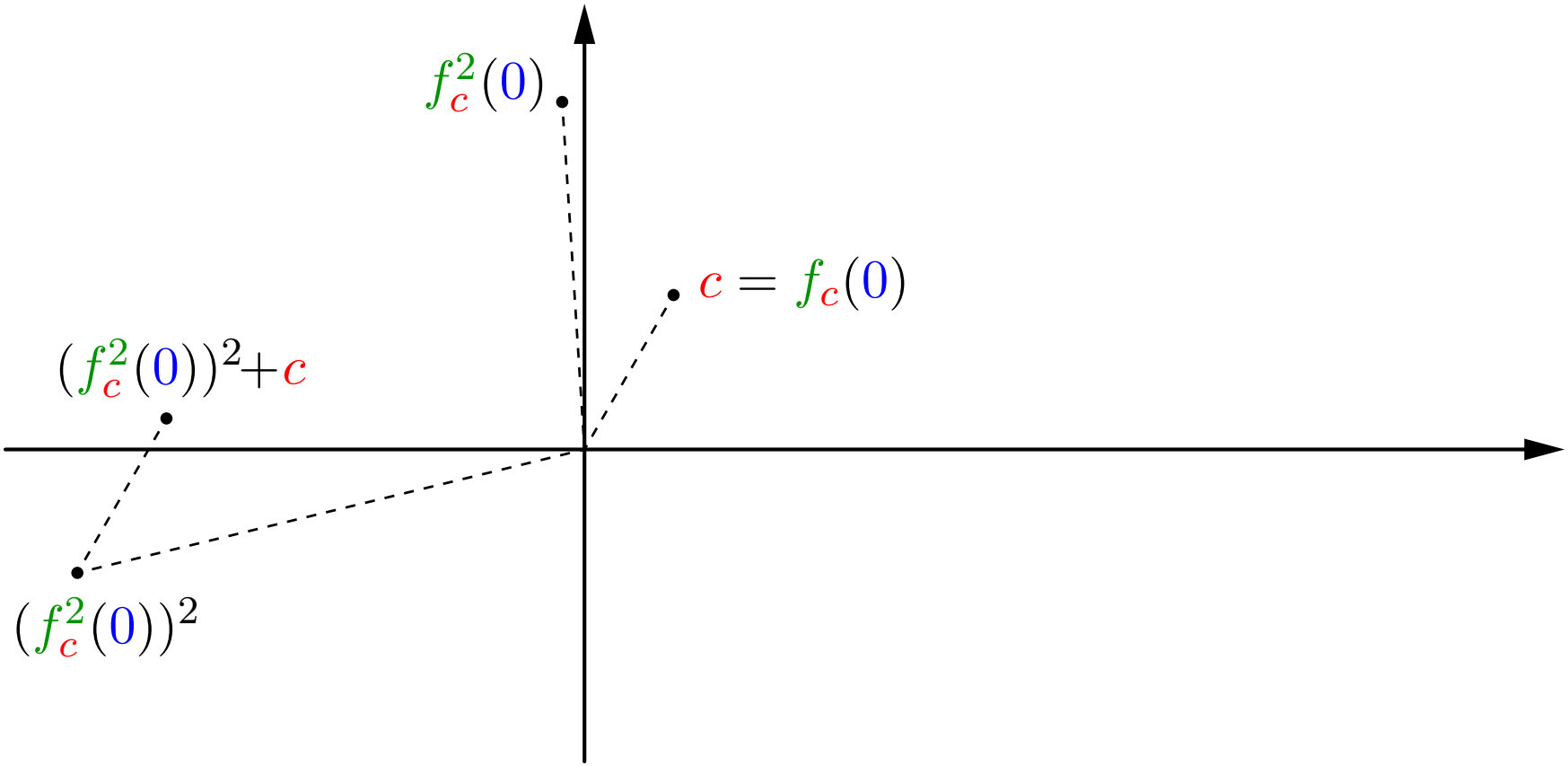
Let us iterate f_c on 0...

The Mandelbrot set



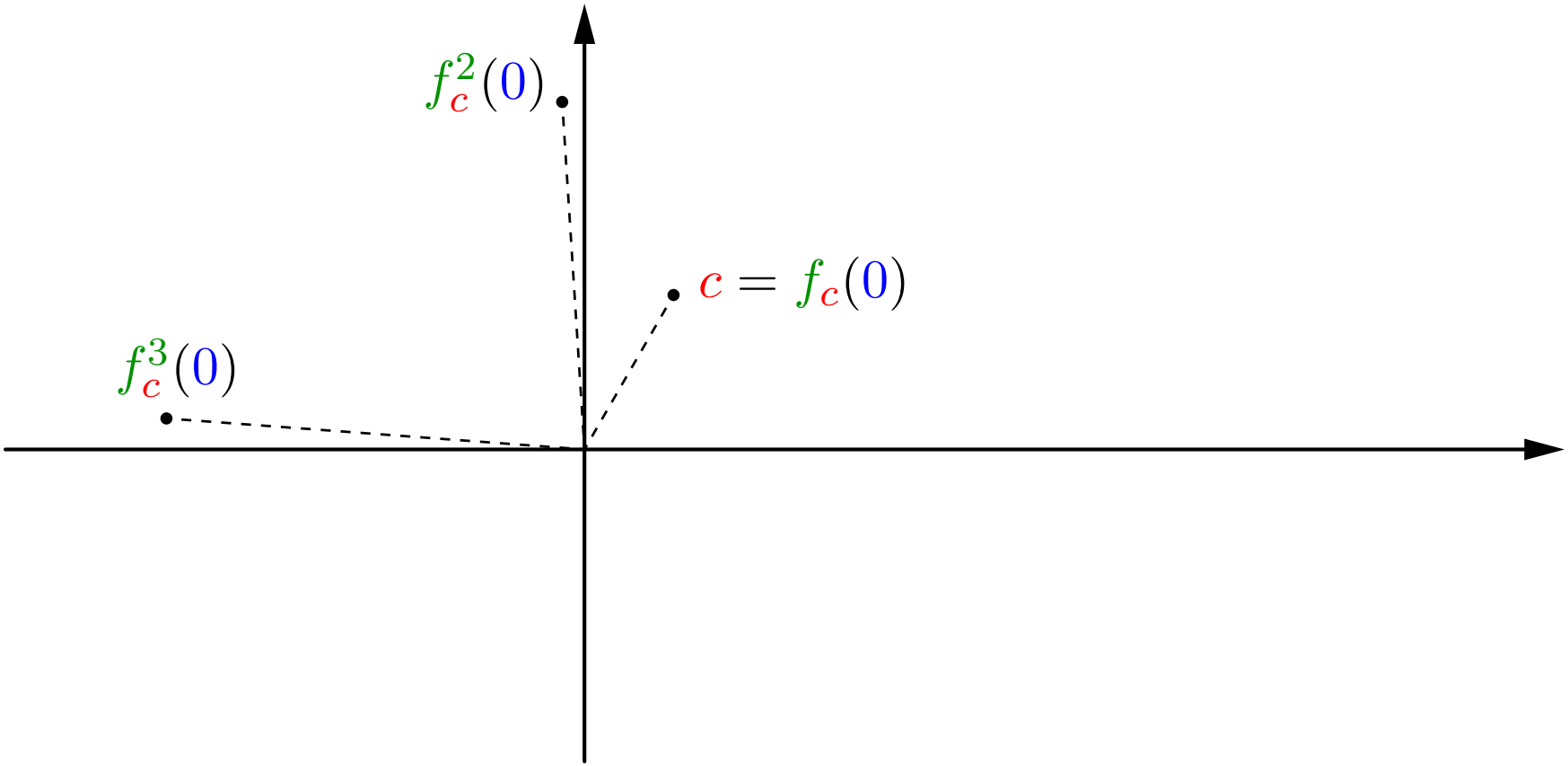
Let us iterate f_c on 0...

The Mandelbrot set



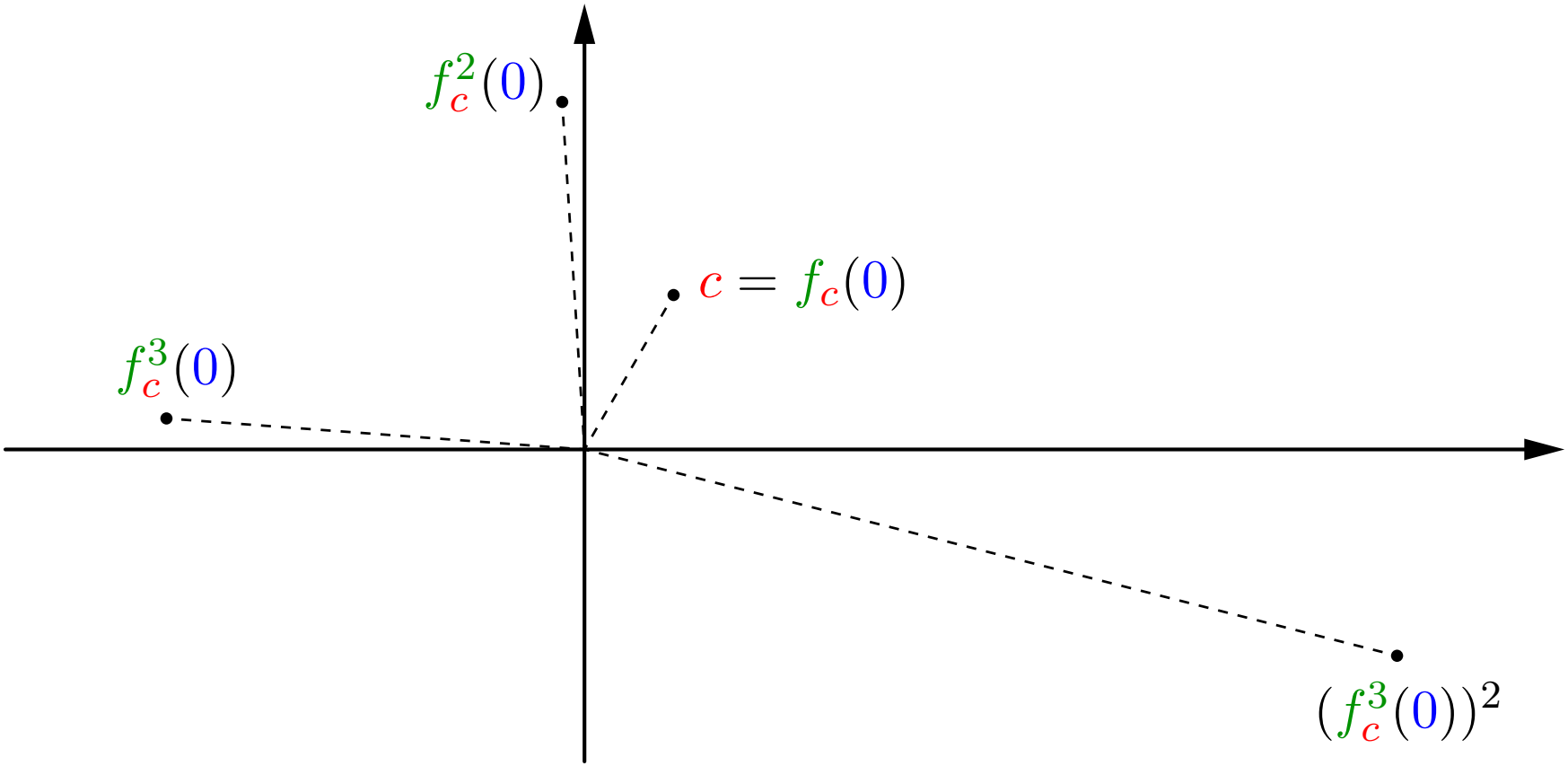
Let us iterate f_c on 0...

The Mandelbrot set



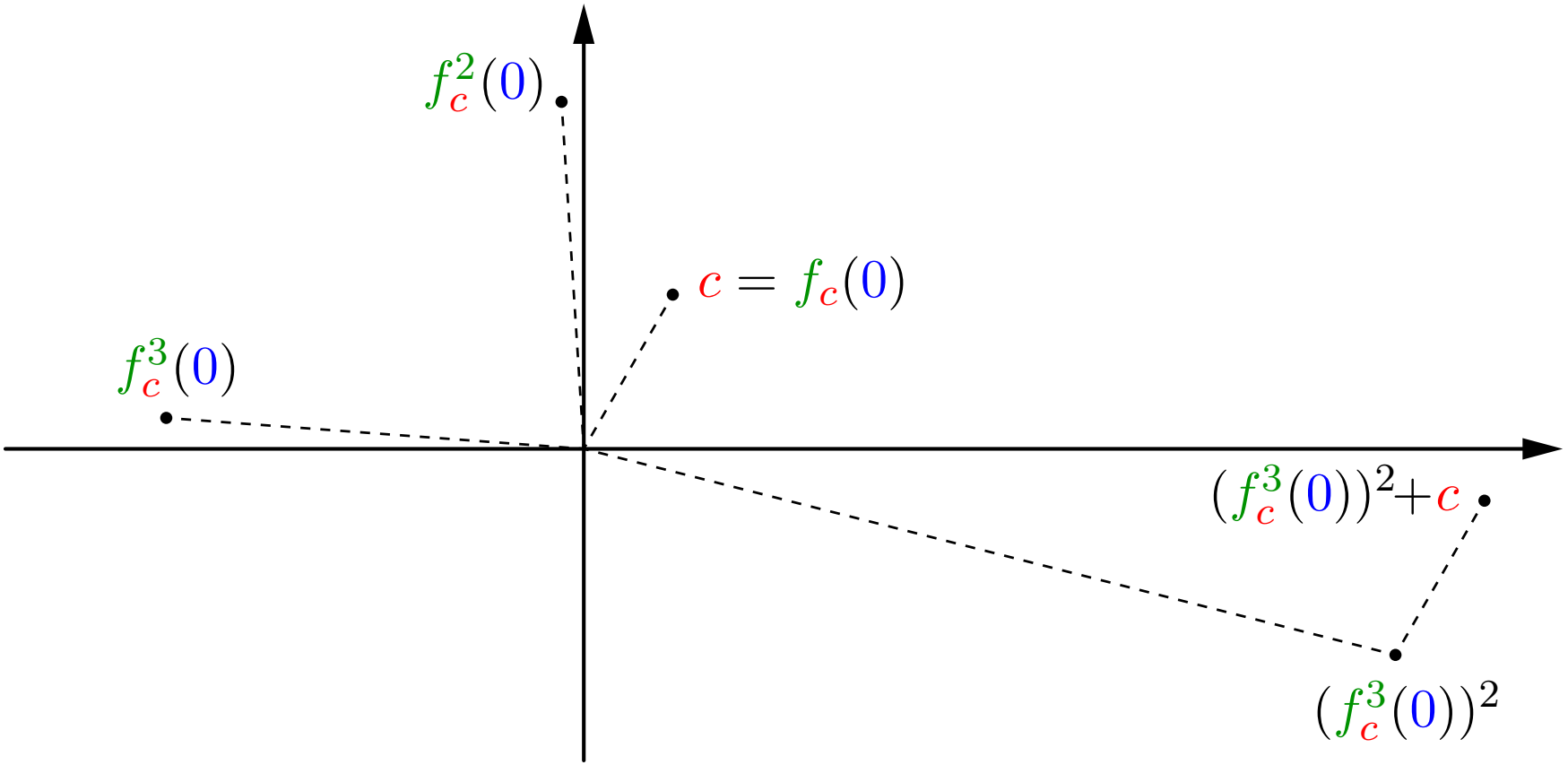
Let us iterate f_c on 0...

The Mandelbrot set



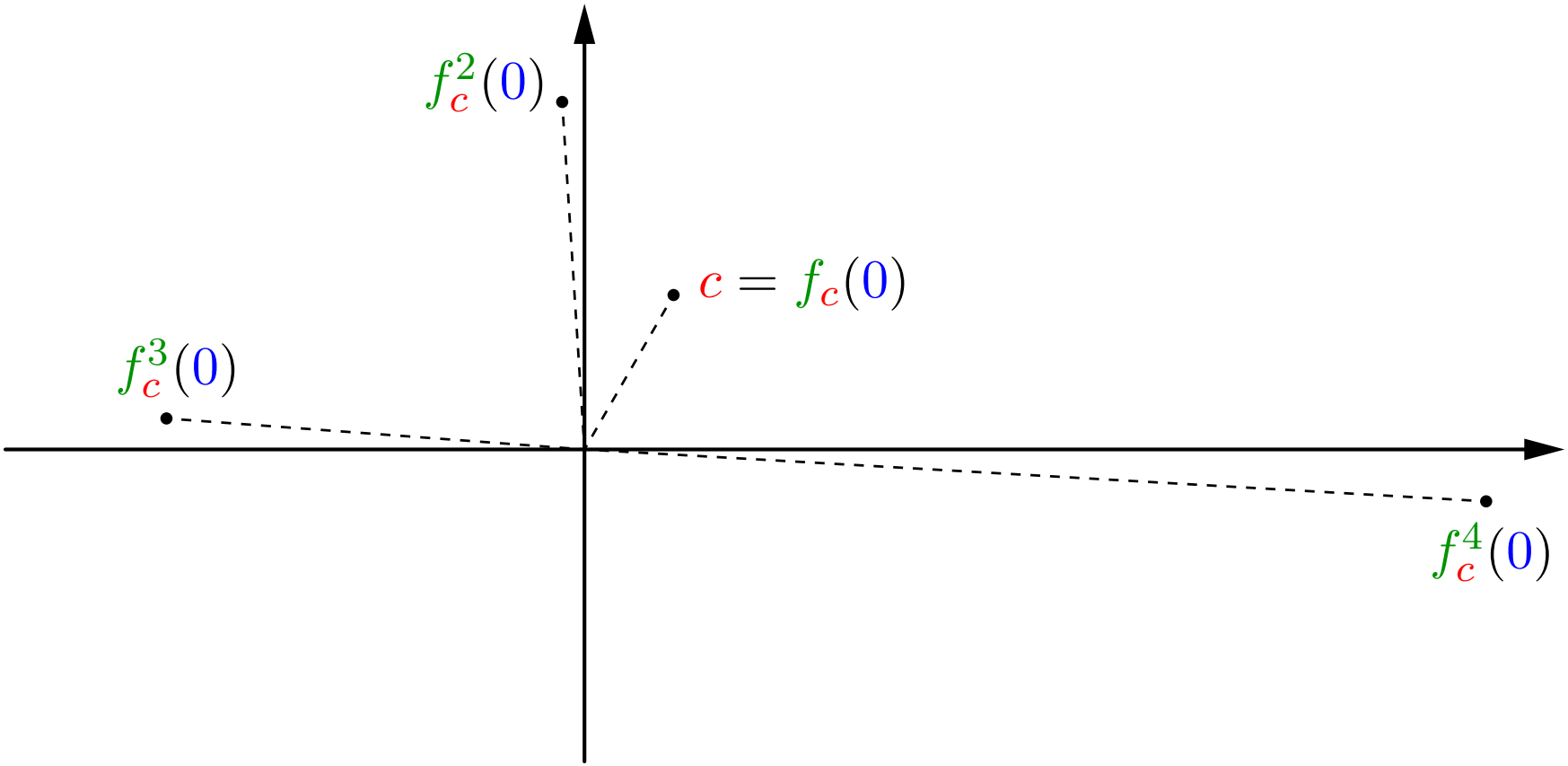
Let us iterate f_c on 0...

The Mandelbrot set



Let us iterate f_c on 0...

The Mandelbrot set



The moduli $|f_c^n(0)|$ are going to diverge,

hence c is not in the Mandelbrot set.

The Mandelbrot set

Is $c = -1$ in the Mandelbrot set?

Iterate $f_{-1}(z) = z^2 - 1$:

$f_{-1}(0) = -1$,

$f_{-1}(f_{-1}(0)) = 0$,

$f_{-1}(f_{-1}(f_{-1}(0))) = -1$,

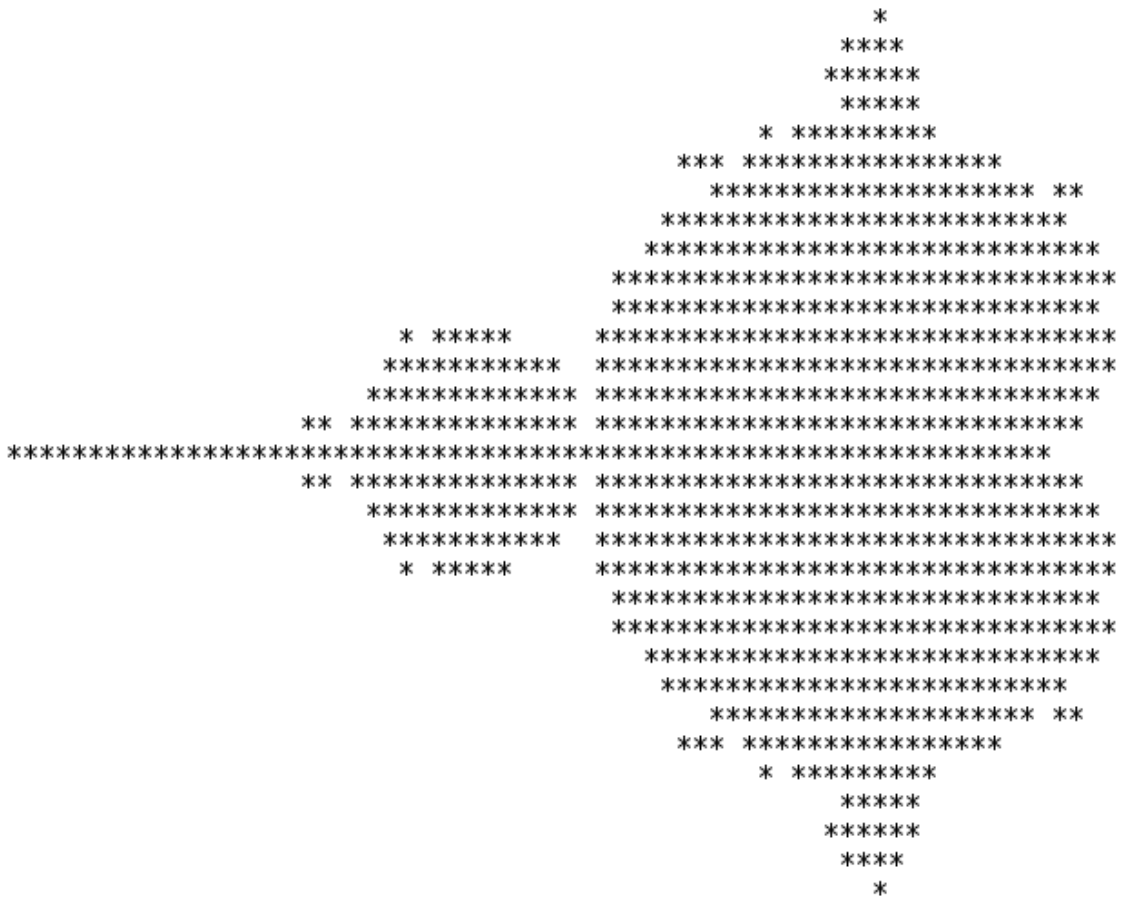
$f_{-1}(f_{-1}(f_{-1}(f_{-1}(0)))) = 0$,

$\vdots$

The moduli $|f^n_{-1}(0)|$ are bounded,

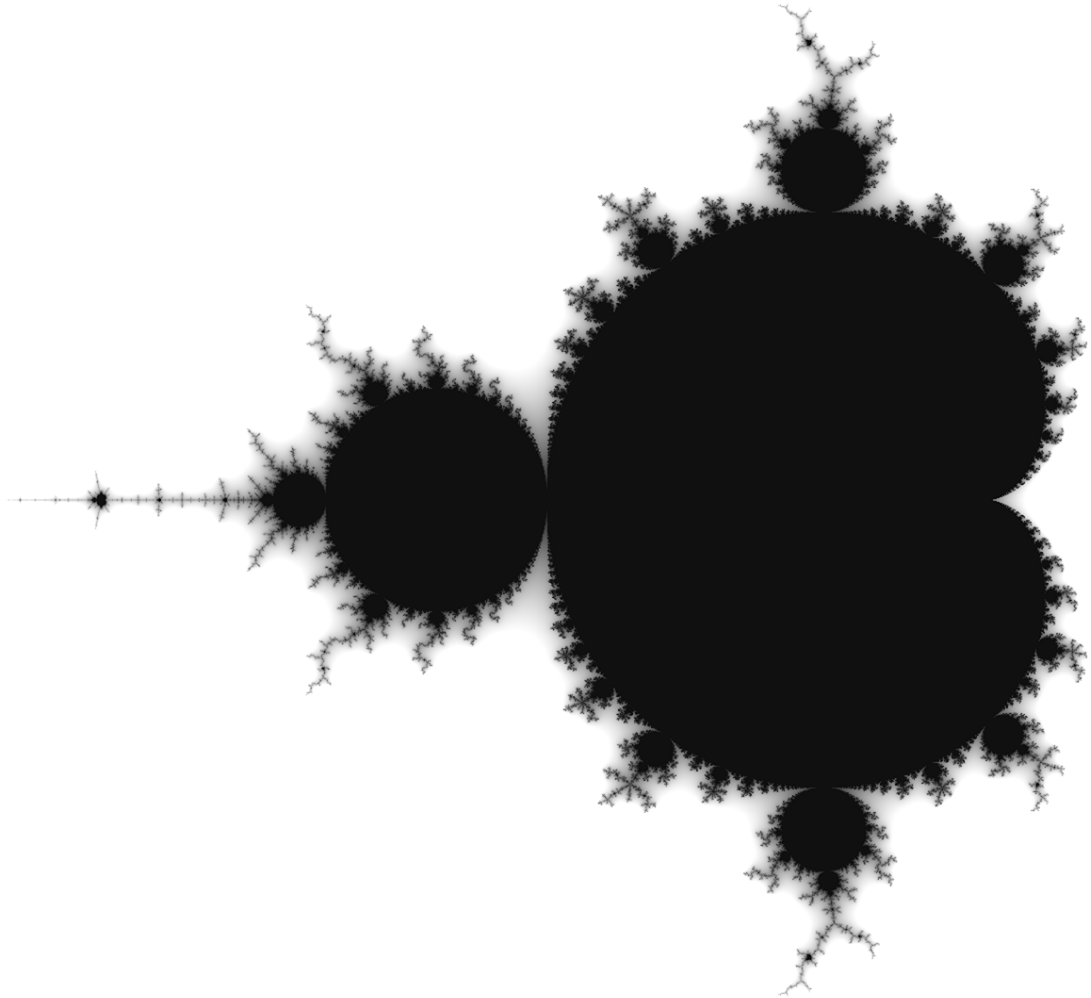
hence -1 is in the Mandelbrot set.

The Mandelbrot set



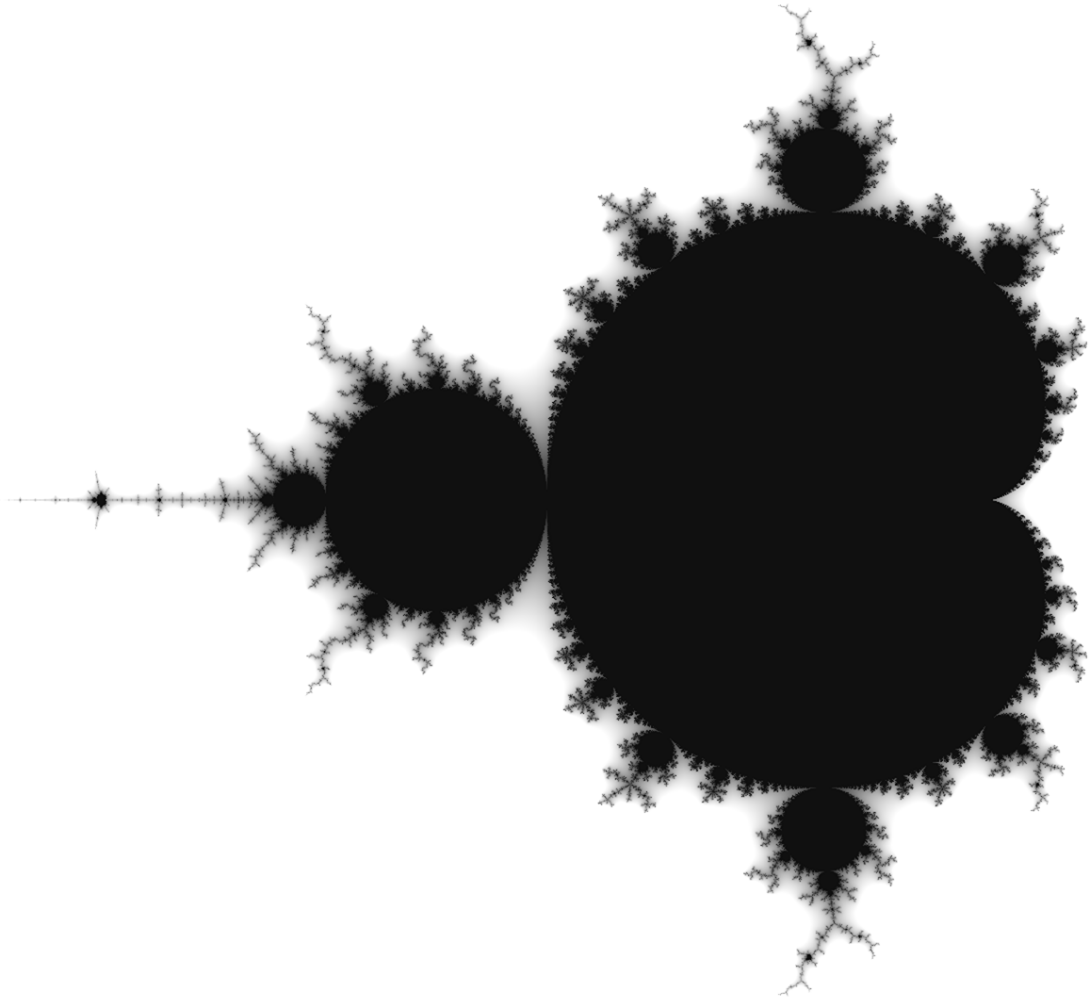
The first published picture of the Mandelbrot set, by Robert W. Brooks and Peter Matelski in 1978.

The Mandelbrot set



Increasing the resolution, a complex shape with infinitely self-similar features appears...

The Mandelbrot set



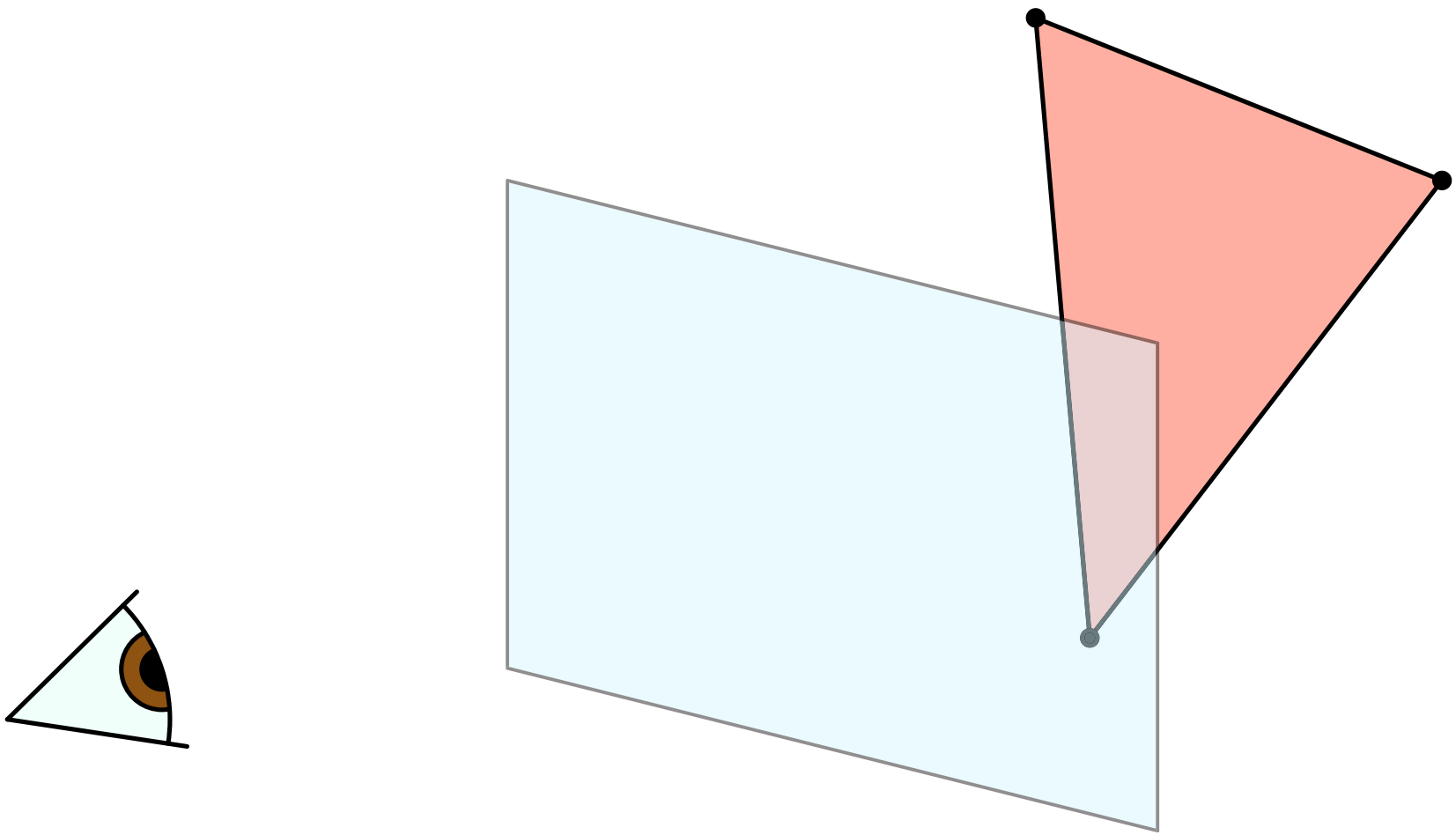
Fun fact: in 1998, Mitsuhiro Shishikura proved that the fractal dimension of the Mandelbrot set is 2, hence it is not a fractal!

Rendering of 3D objects on a 2D screen

Main rendering techniques:

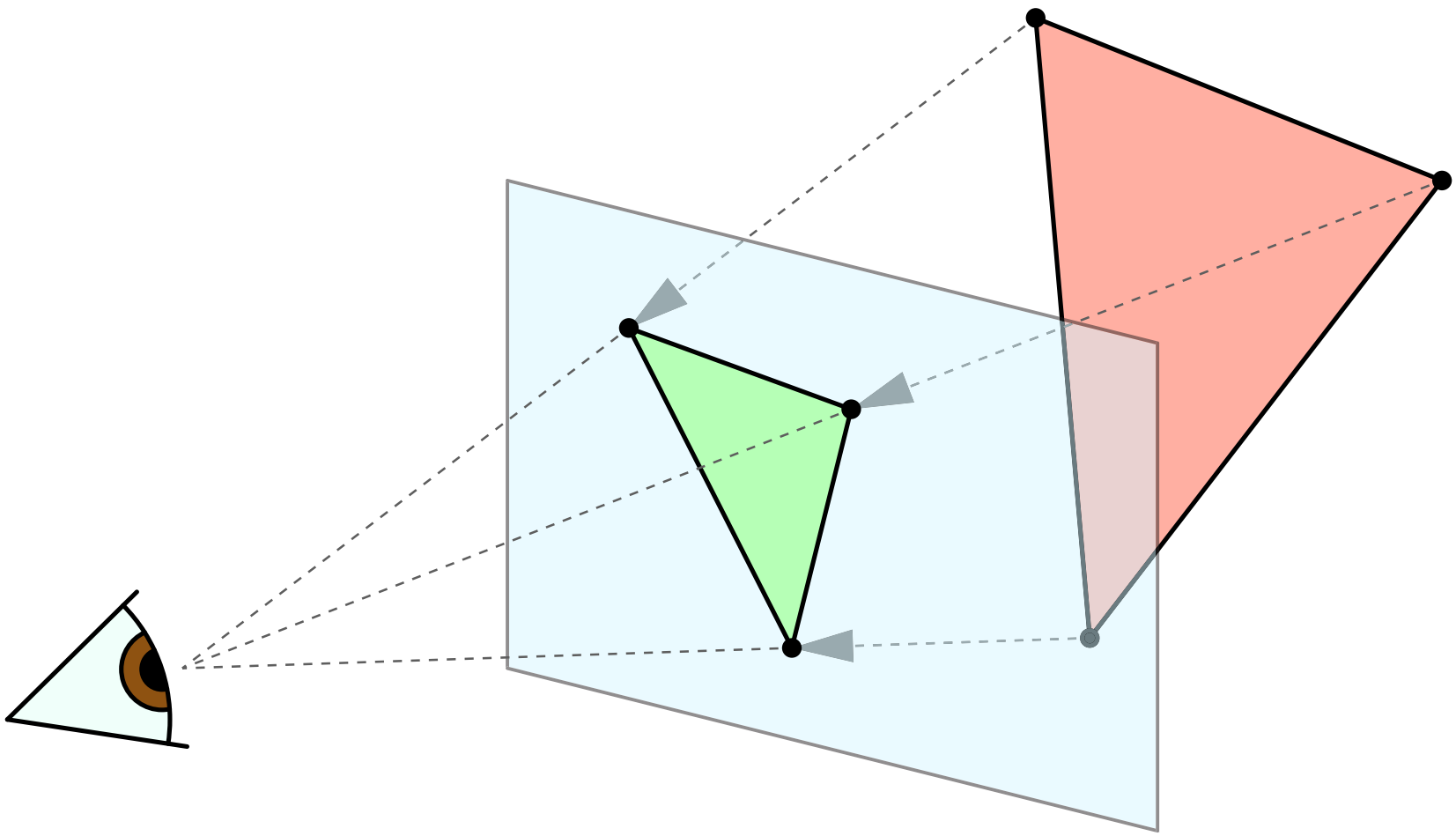
1. **Rasterization**
2.
3.

Rendering of 3D objects on a 2D screen



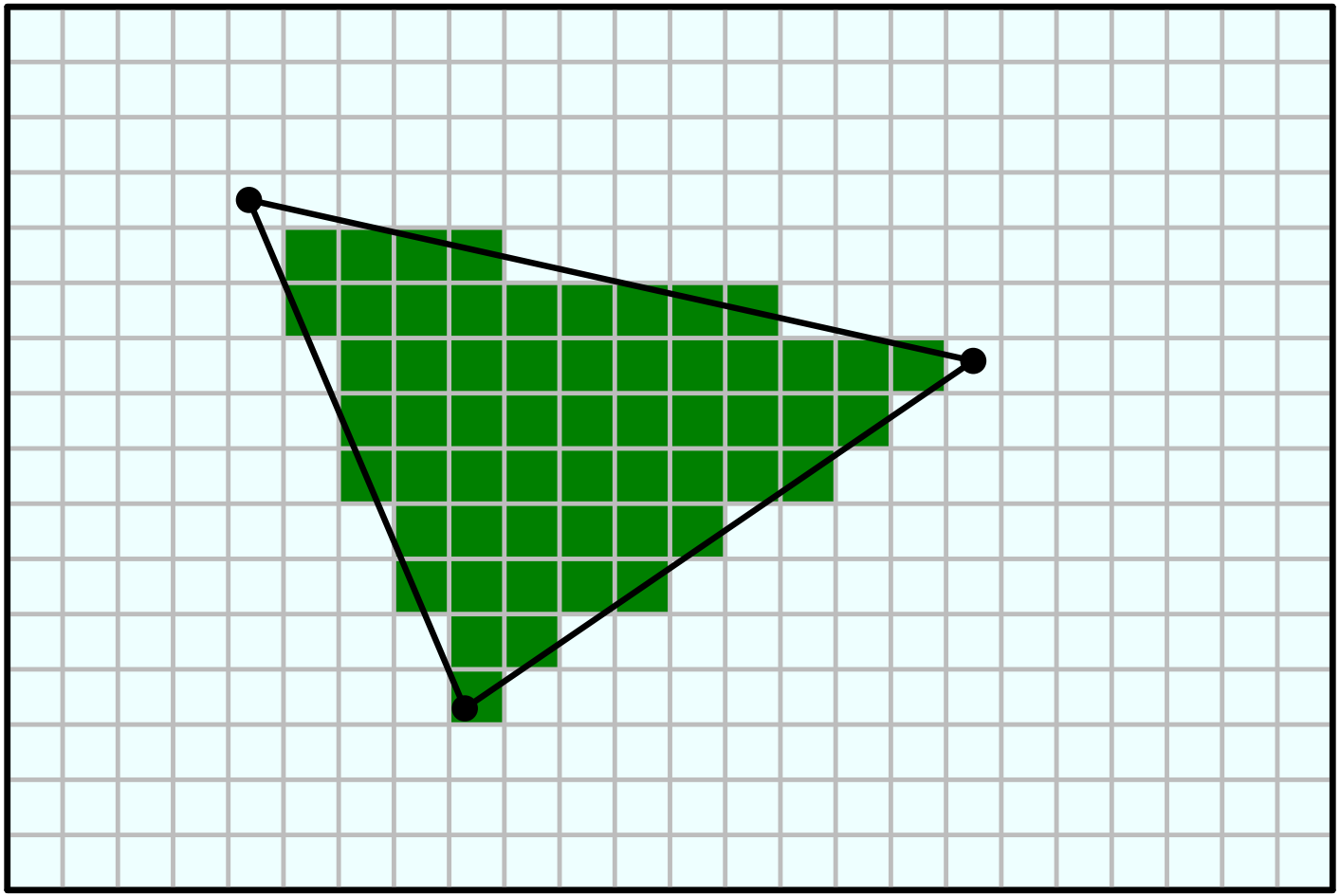
A 3D scene has to be represented on a 2D screen
as seen from an observer positioned behind the screen.

Rendering of 3D objects on a 2D screen



In the rasterization technique, the vertices of a 3D shape are first projected onto the screen...

Rendering of 3D objects on a 2D screen



...then, the rasterizer identifies the pixels that fall into the shape, and colors them appropriately.

Rendering of 3D objects on a 2D screen

Rasterization has always been the most popular technique.
Video games and real-time applications heavily rely on it.

Rendering of 3D objects on a 2D screen

Main rendering techniques:

1. **Rasterization**

   Not suitable for drawing fractals (infinitely many vertices!)

2.

3.

Rendering of 3D objects on a 2D screen

Main rendering techniques:

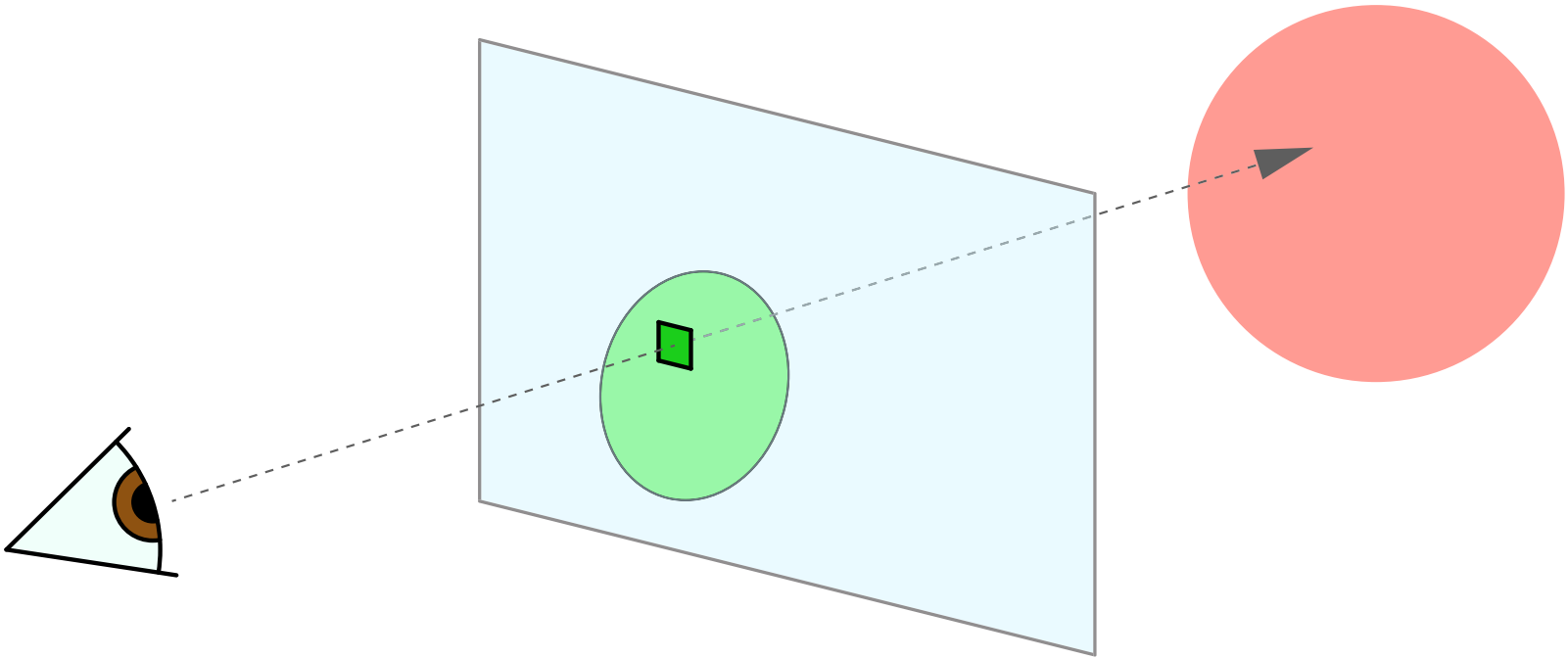
1. **Rasterization**

   Not suitable for drawing fractals (infinitely many vertices!)

2. **Ray tracing**

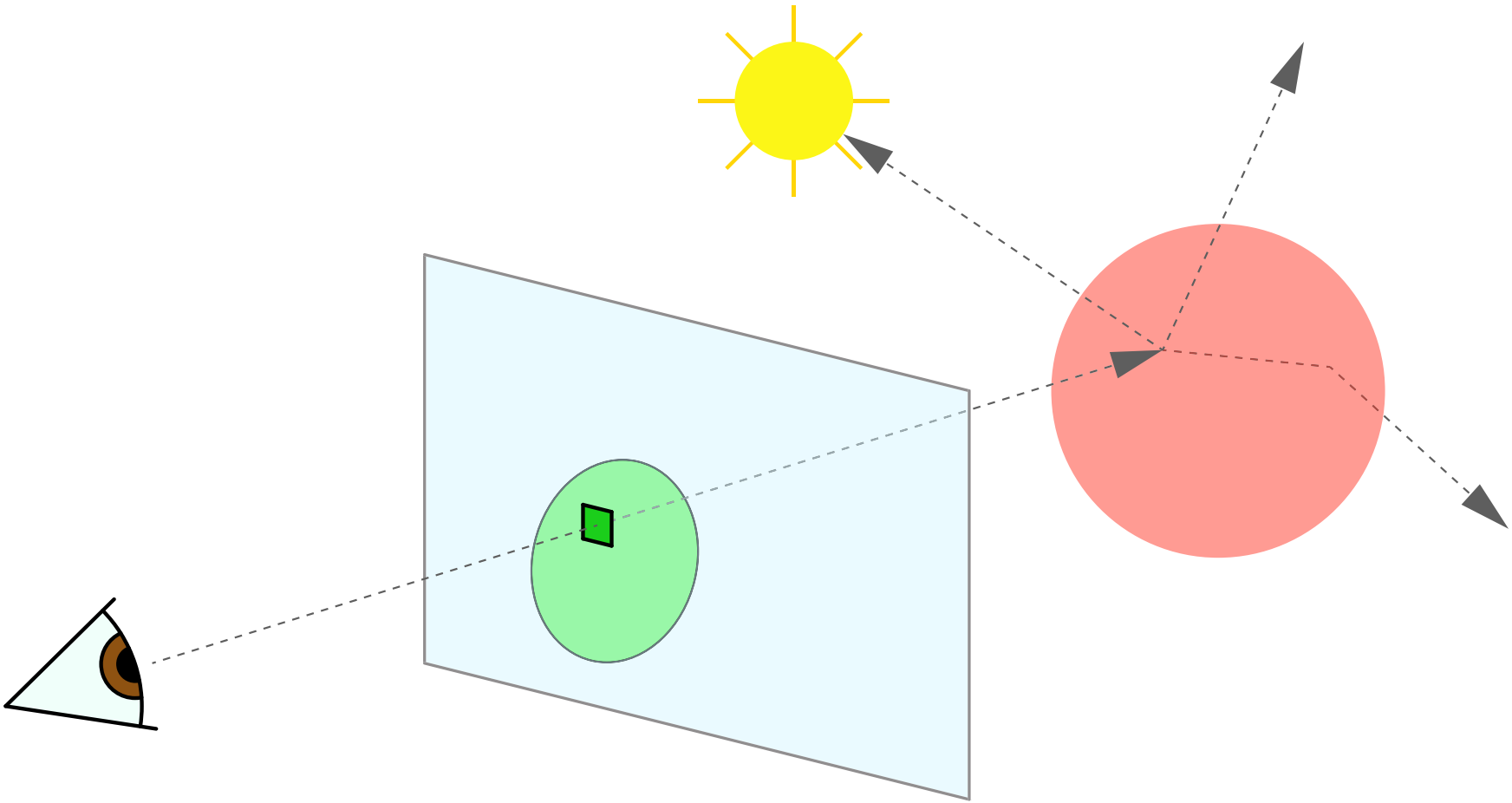
3.

Rendering of 3D objects on a 2D screen



In the ray tracing technique, a ray is cast for each pixel on screen, and the intersection with the closest object is computed.

Rendering of 3D objects on a 2D screen



Additional rays may be cast to compute shadows, reflections, refractions, caustics, etc.

Rendering of 3D objects on a 2D screen

Ray tracing gives the best results, but is very slow.
This is the preferred technique for animated movies.

Rendering of 3D objects on a 2D screen

Main rendering techniques:

1. **Rasterization**

   Not suitable for drawing fractals (infinitely many vertices!)

2. **Ray tracing**

   Not suitable for real-time rendering (heavy computations!)

3.

Rendering of 3D objects on a 2D screen

Main rendering techniques:

1. **Rasterization**

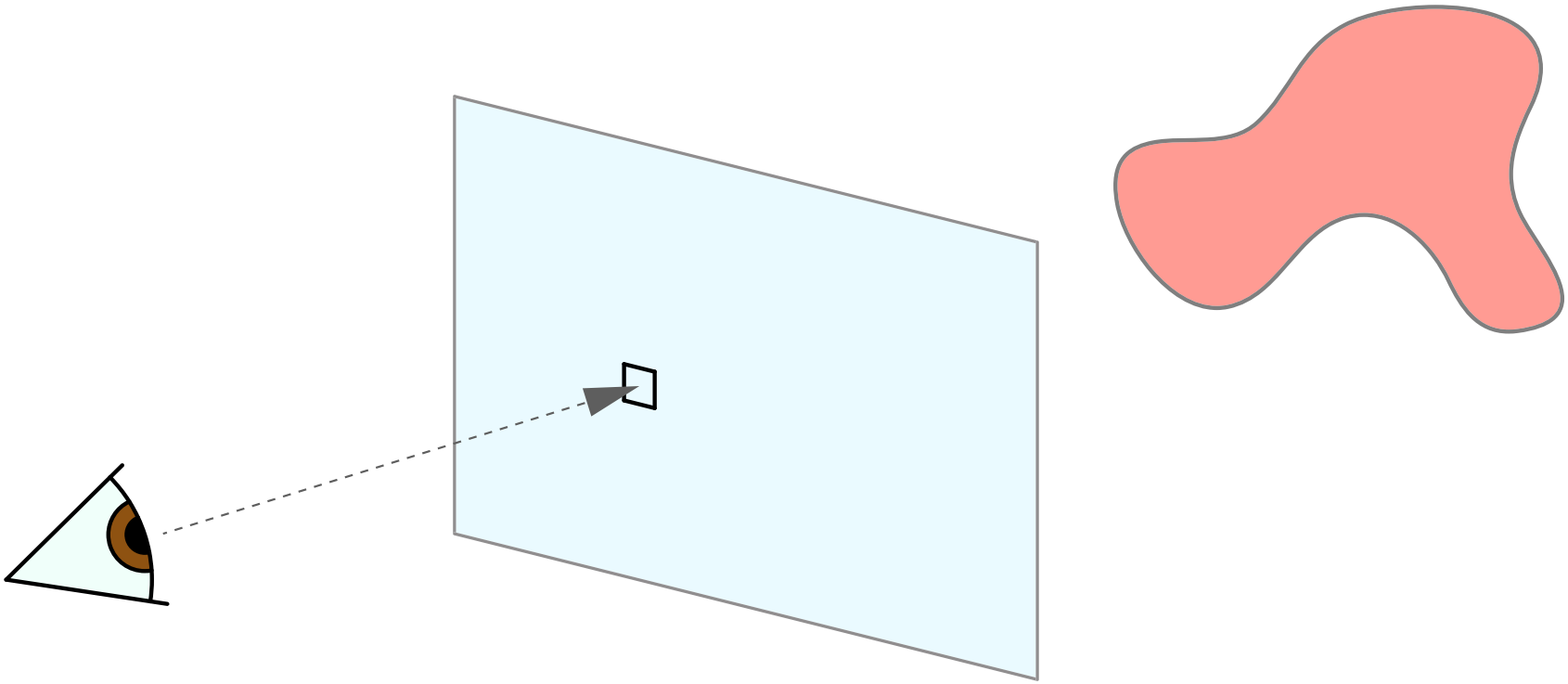
   Not suitable for drawing fractals (infinitely many vertices!)

2. **Ray tracing**

   Not suitable for real-time rendering (heavy computations!)

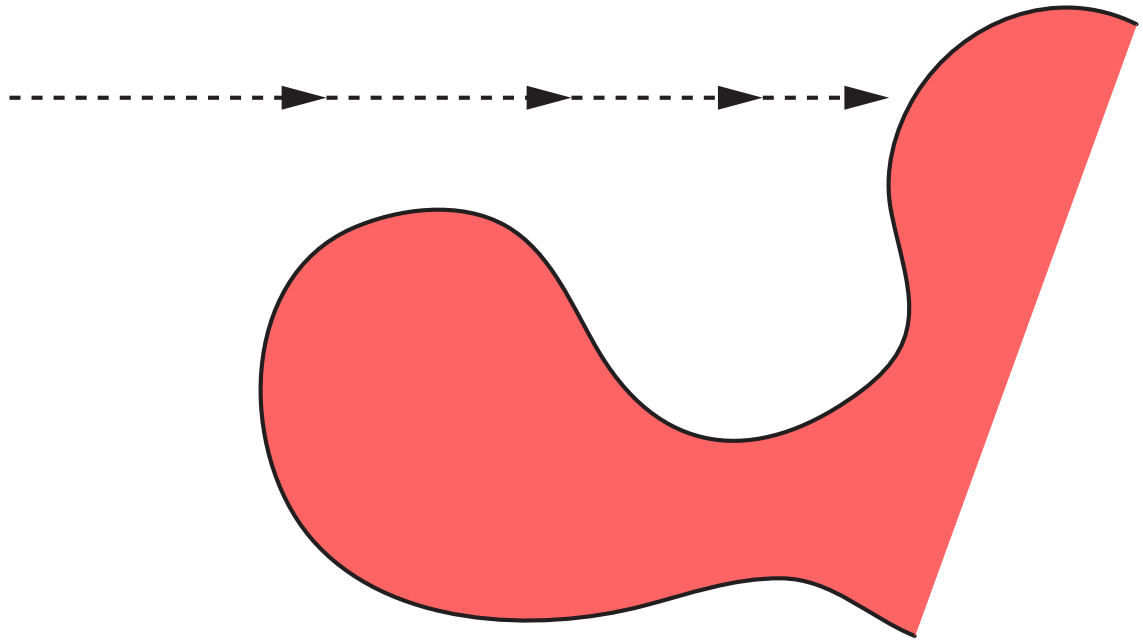
3. **Ray marching**

Ray marching



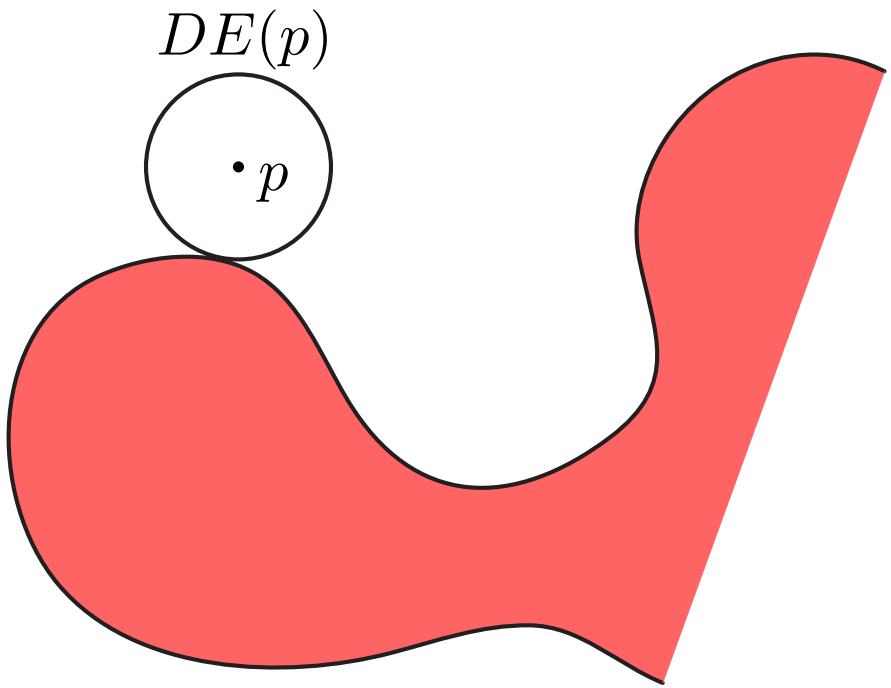
Ray marching starts like ray tracing:
a ray is cast for each pixel on screen.

Ray marching



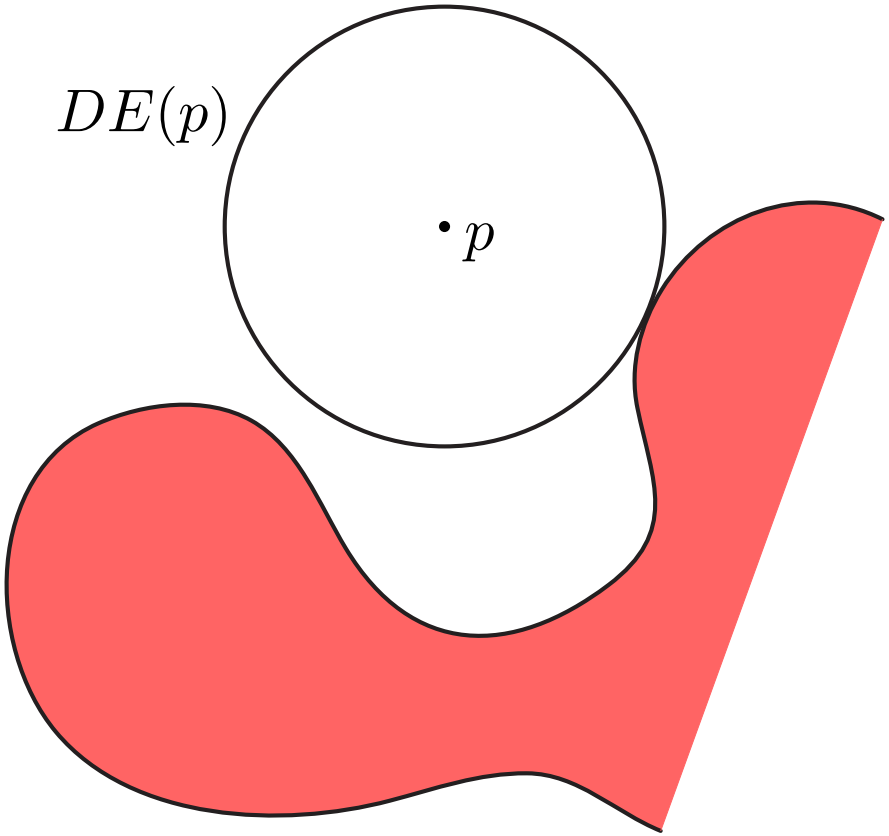
But instead of computing the exact intersection with a shape, it finds better and better approximations of it.

Ray marching



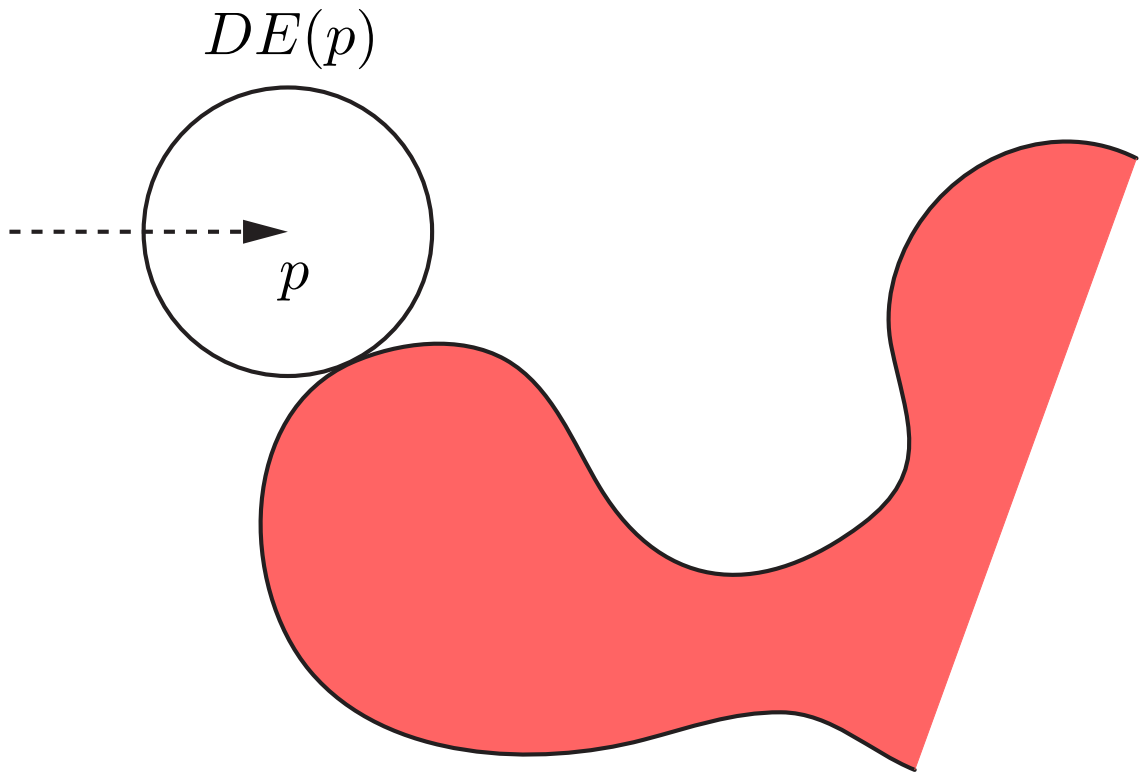
Suppose we are given a distance estimator for the shape: a function DE that tells the distance of each point from the shape.

Ray marching



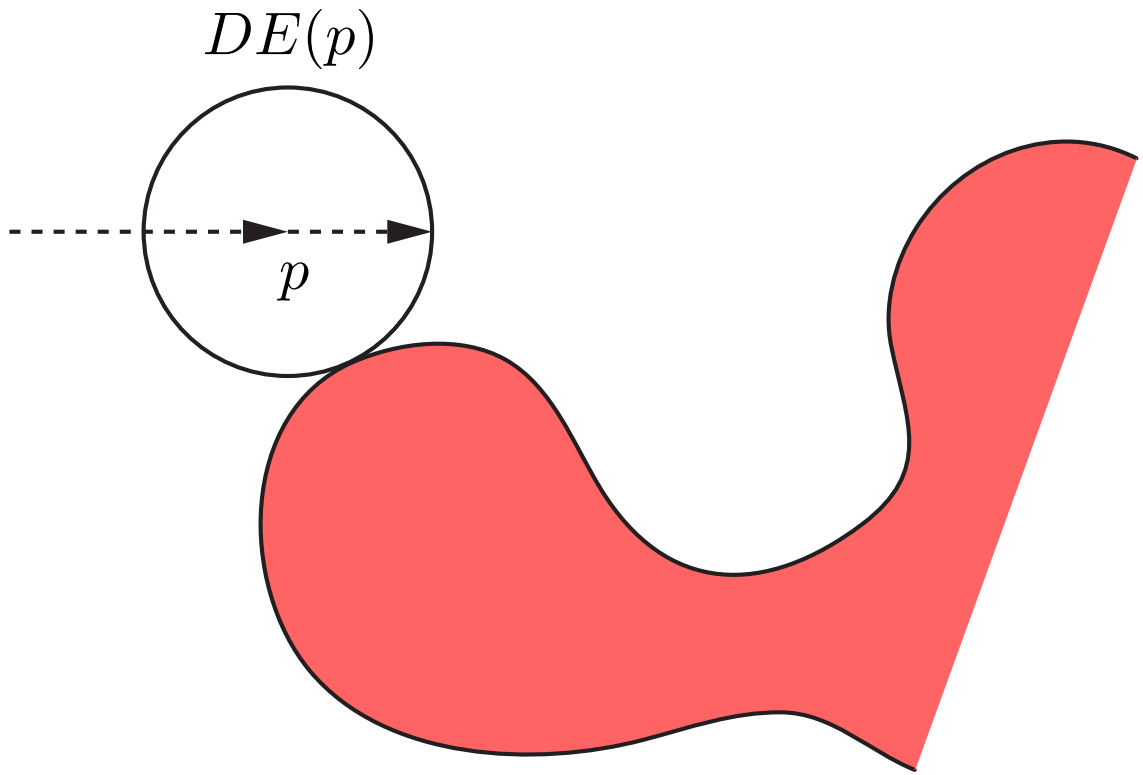
Suppose we are given a distance estimator for the shape: a function DE that tells the distance of each point from the shape.

Ray marching



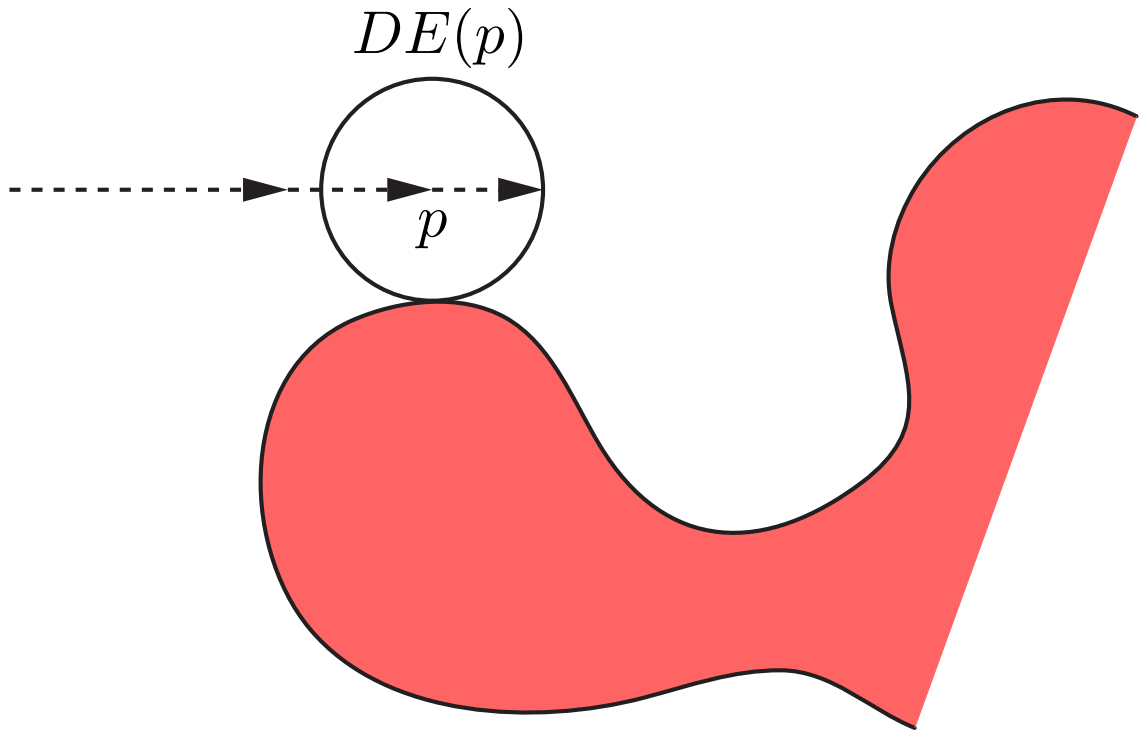
When our ray is at a point p, we can advance it by at least $DE(p)$ without penetrating the shape.

Ray marching



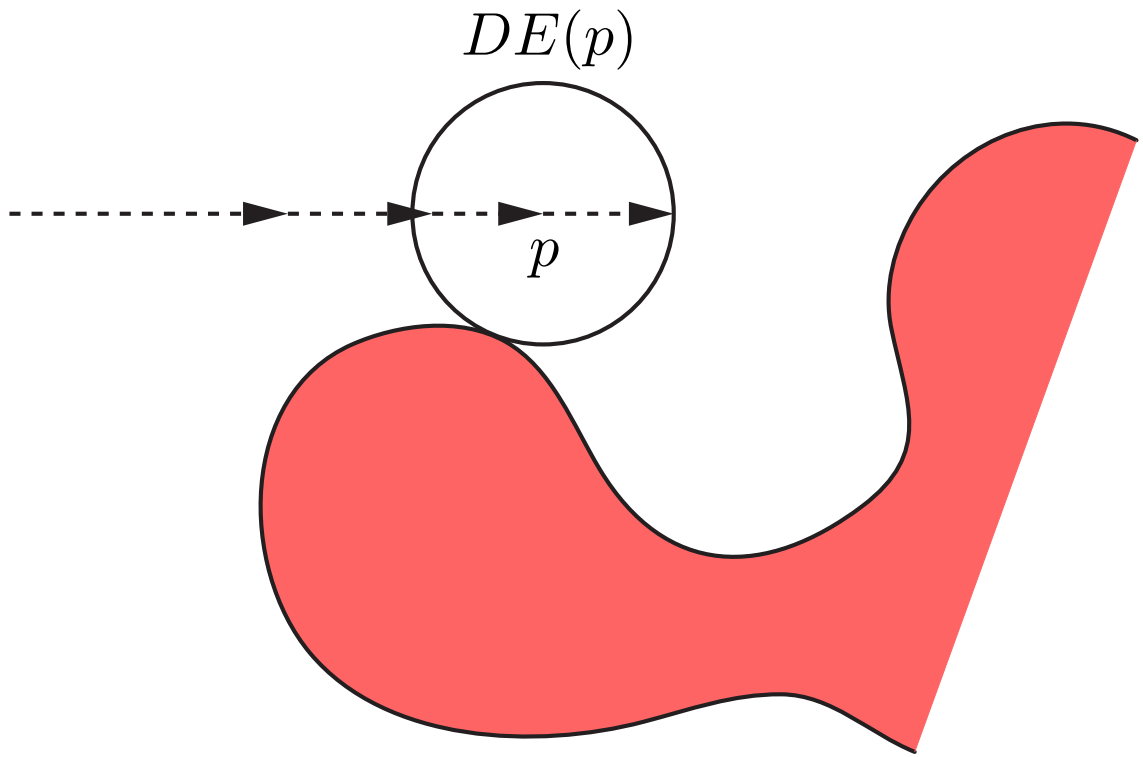
When our ray is at a point p, we can advance it by at least $DE(p)$ without penetrating the shape.

Ray marching



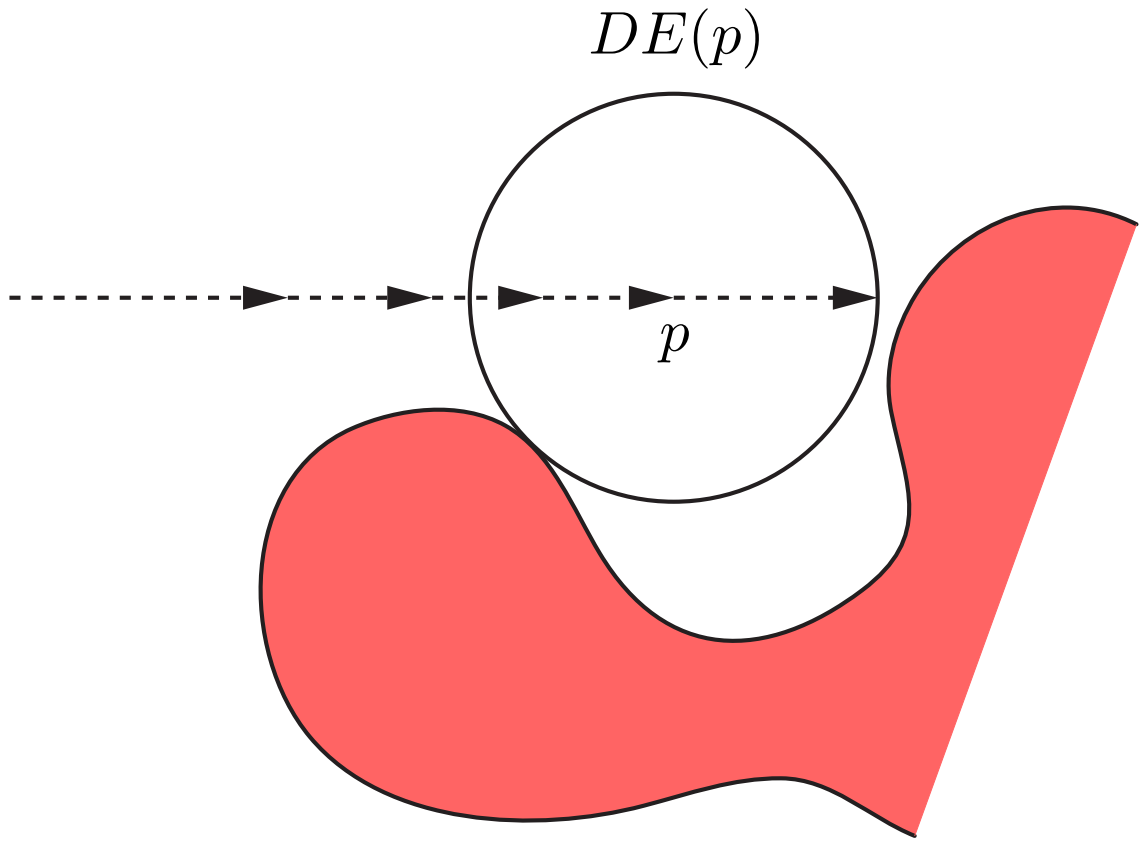
When our ray is at a point p, we can advance it by at least $DE(p)$ without penetrating the shape.

Ray marching



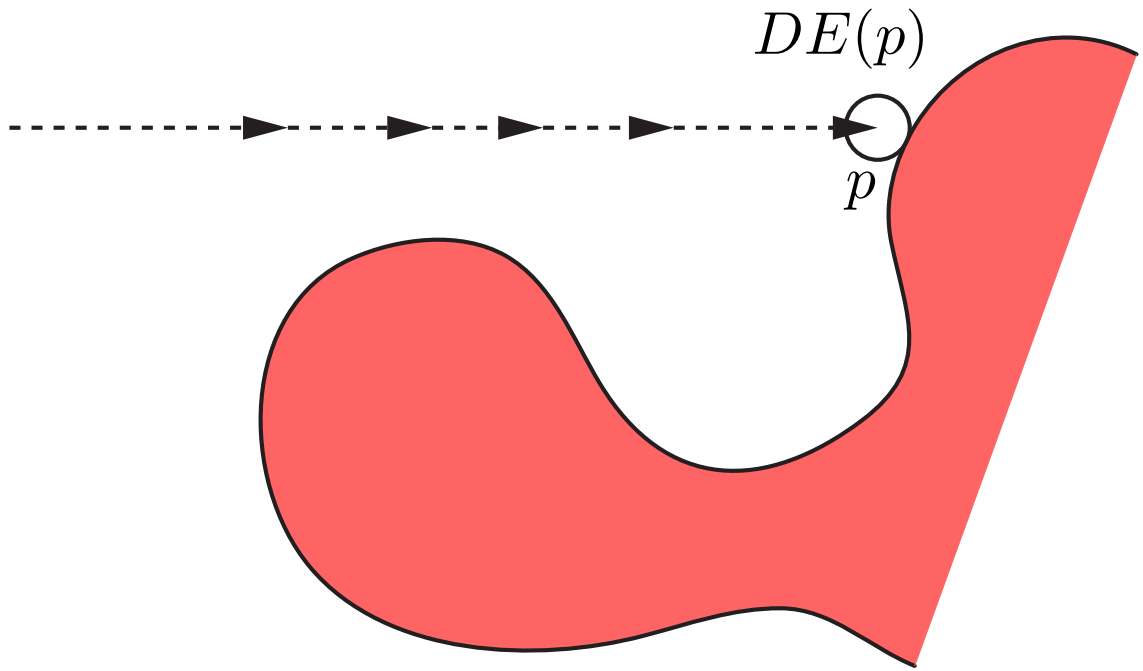
When our ray is at a point p, we can advance it by at least $DE(p)$ without penetrating the shape.

Ray marching



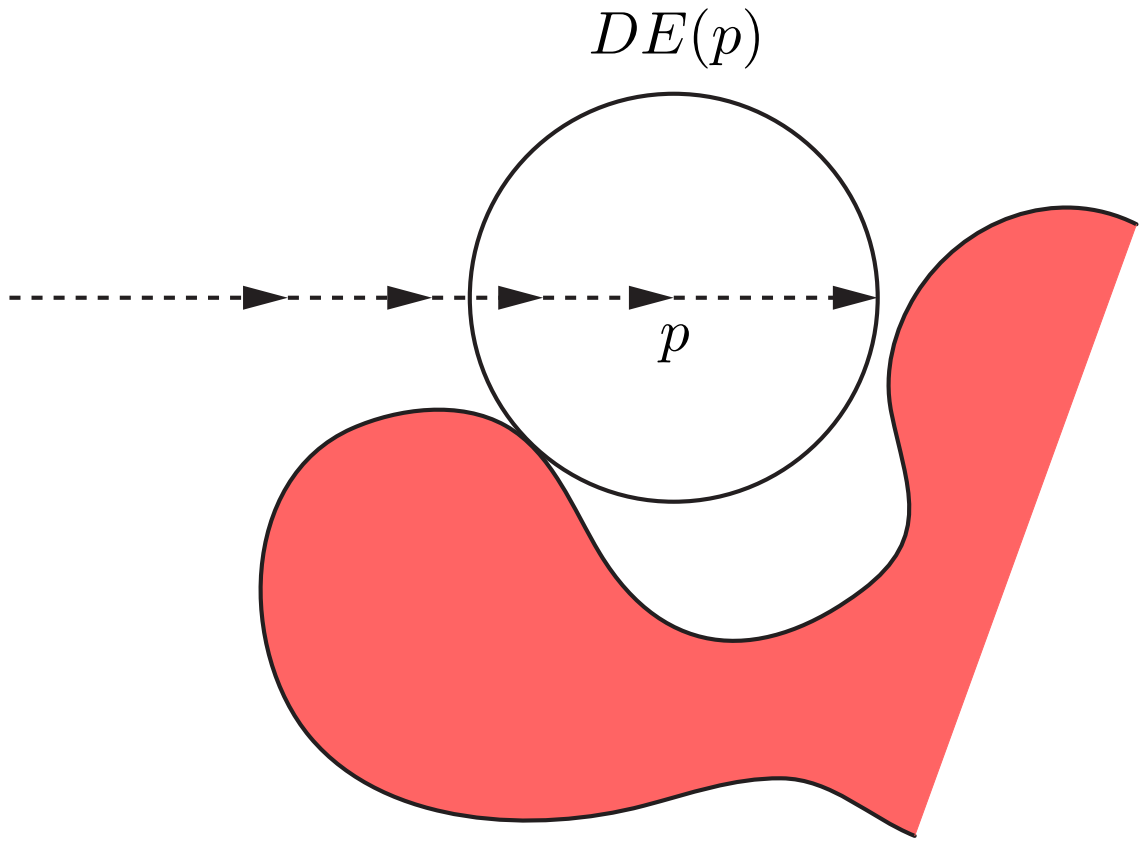
When our ray is at a point p, we can advance it by at least $DE(p)$ without penetrating the shape.

Ray marching



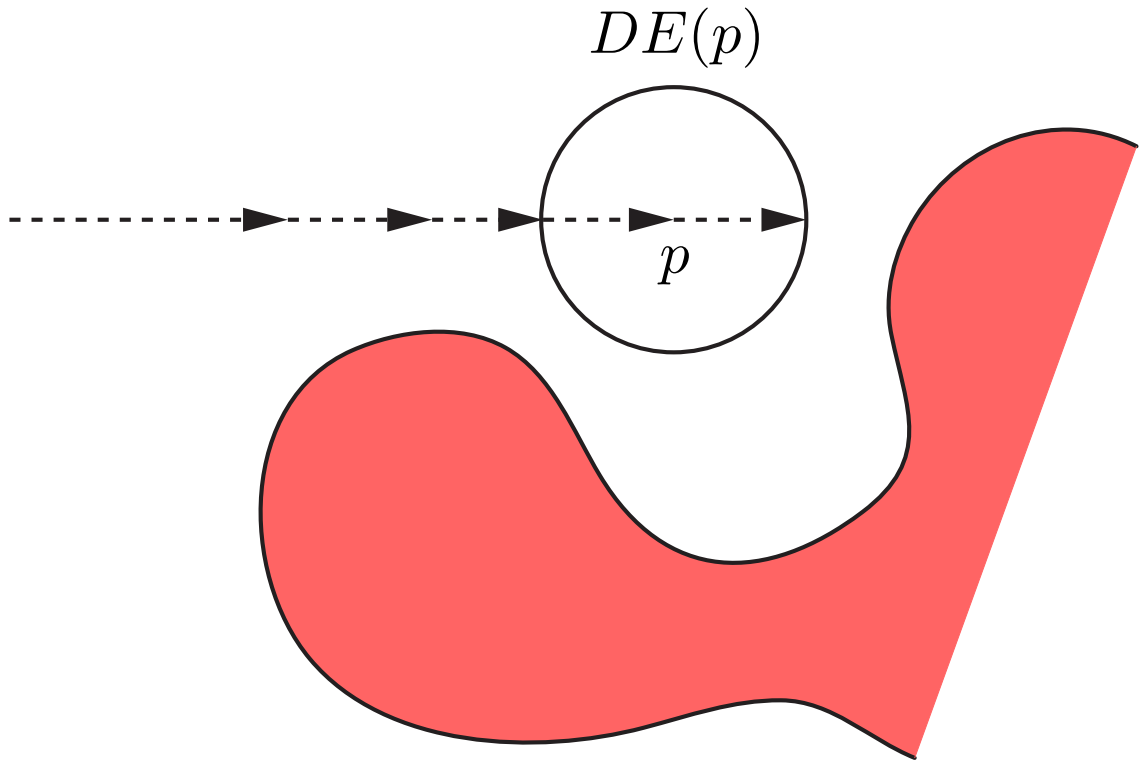
As soon as $DE(p)$ is smaller than some threshold,
we claim that the ray has hit the shape, and we stop.

Ray marching



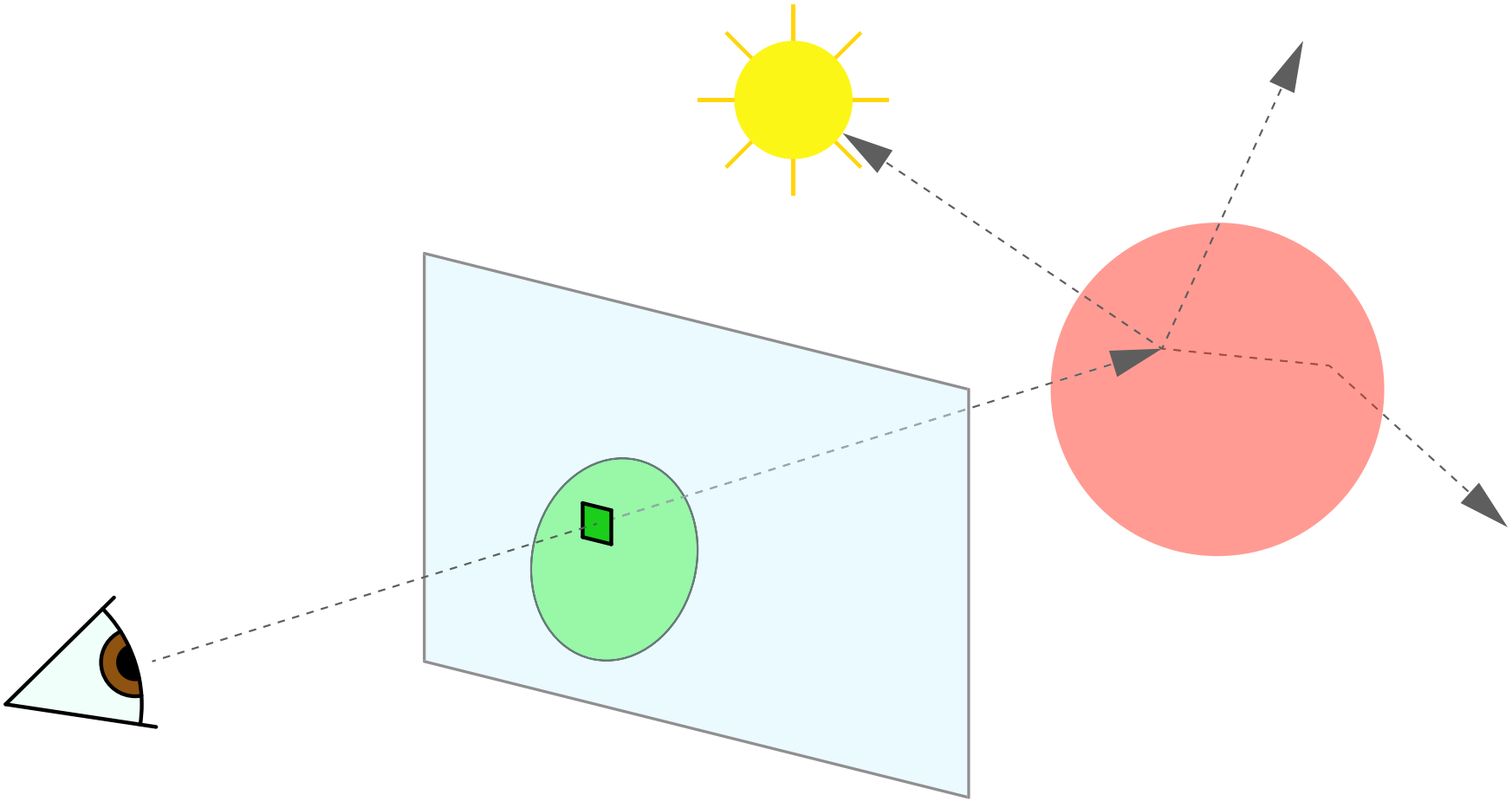
Note that $DE(p)$ does not have to be the exact distance, but it could be a lower bound (hence the name "estimator").

Ray marching



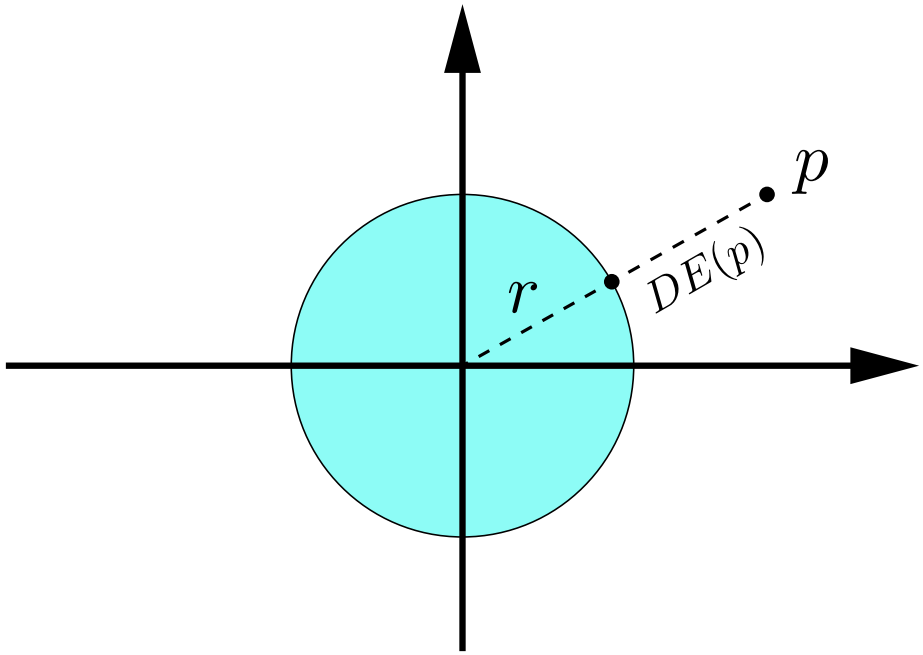
Note that $DE(p)$ does not have to be the exact distance, but it could be a lower bound (hence the name "estimator").

Ray marching



As with ray tracing, additional rays may be cast to compute shadows, reflections, refractions, caustics, etc.

Distance estimator of a sphere



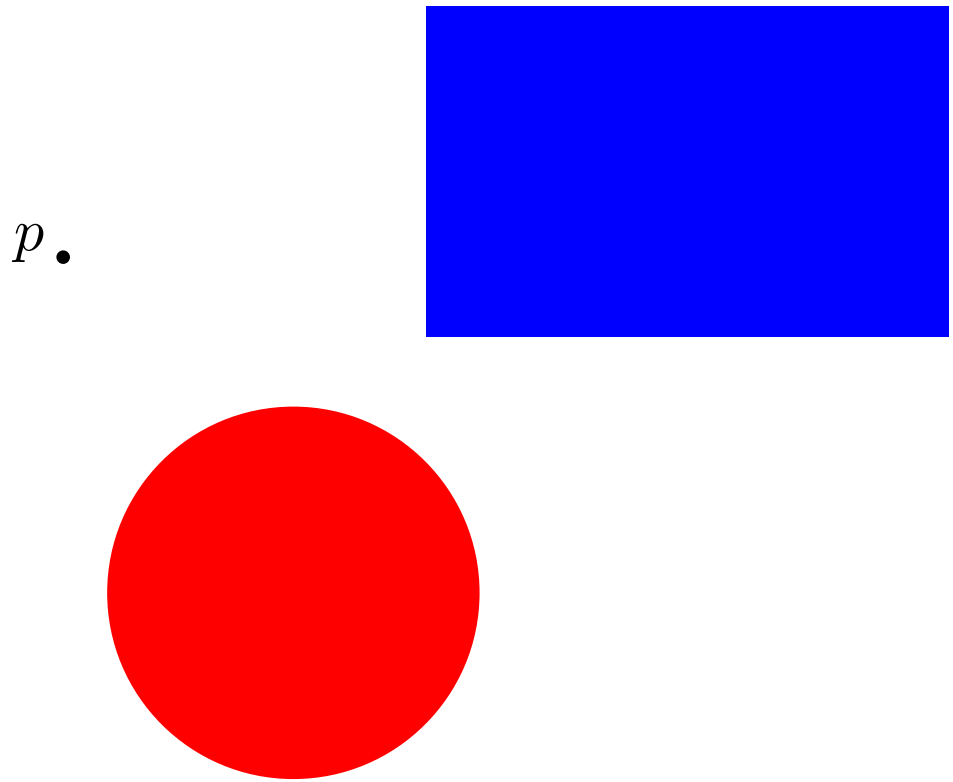
An exact distance estimator for a sphere of radius r is easy to compute: $DE(p) = \|p\| - r$.

More distance estimators

Exact distance estimators for some simple shapes:

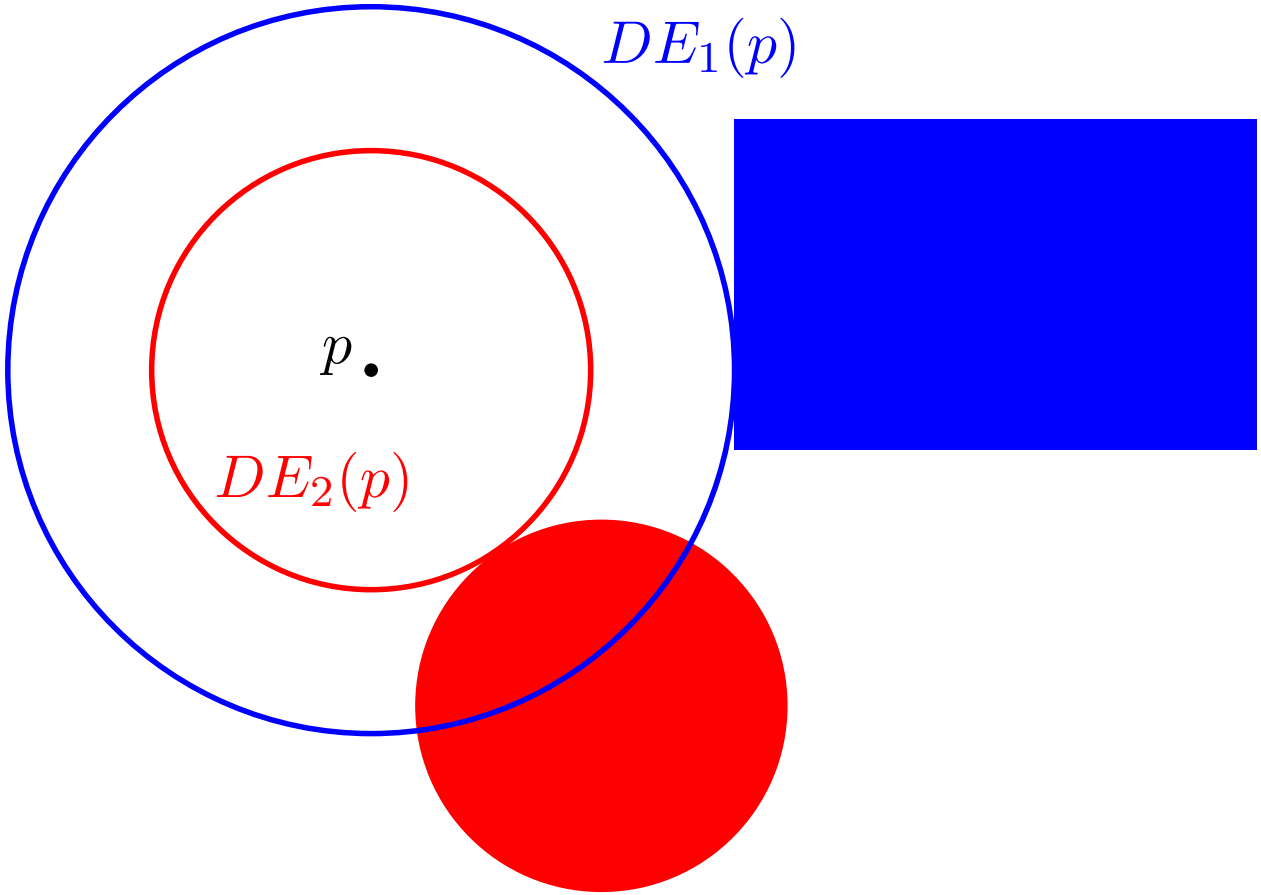
- **Sphere:** $DE(p) = \|p\| - r$
- **Cylinder:** $DE(p) = \|p_{xy}\| - r$
- **Cone:** $DE(p) = (\|p_{xy}\|, p_z) \cdot c$
- **Torus:** $DE(p) = \|(\|p_{xy}\| - r_1, p_z)\| - r_2$
- **Plane:** $DE(p) = p \cdot n + d$
- **Box:** $DE(p) = \| \max\{|p| - b, 0\}\|$

Combining shapes



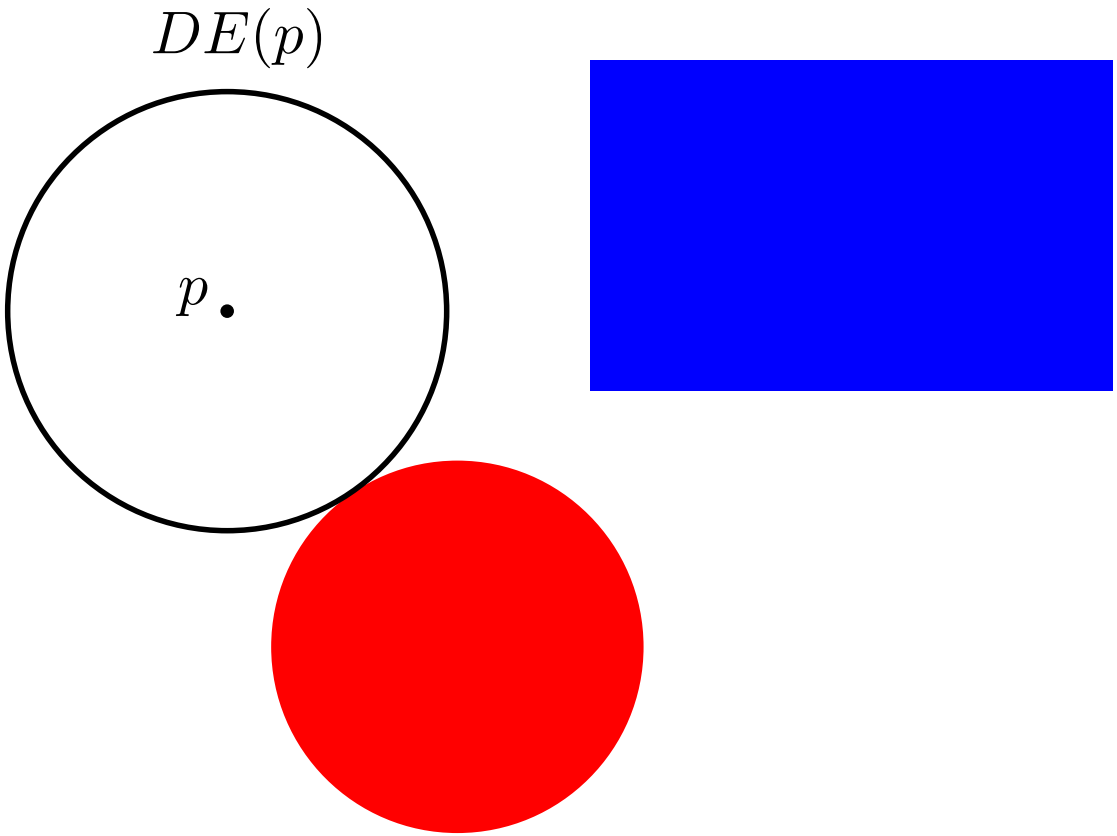
What if there is more than one shape in the scene?

Combining shapes



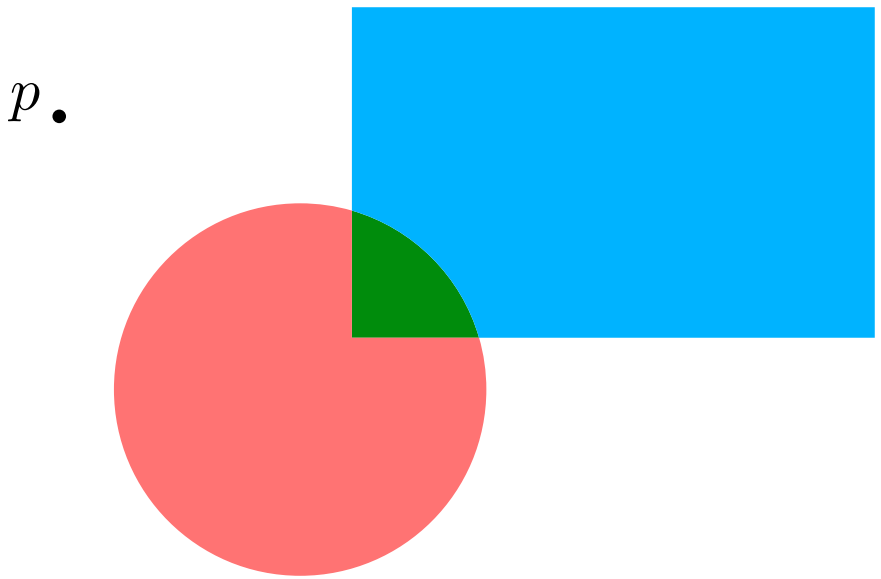
Suppose we have a distance estimator for each shape.

Combining shapes



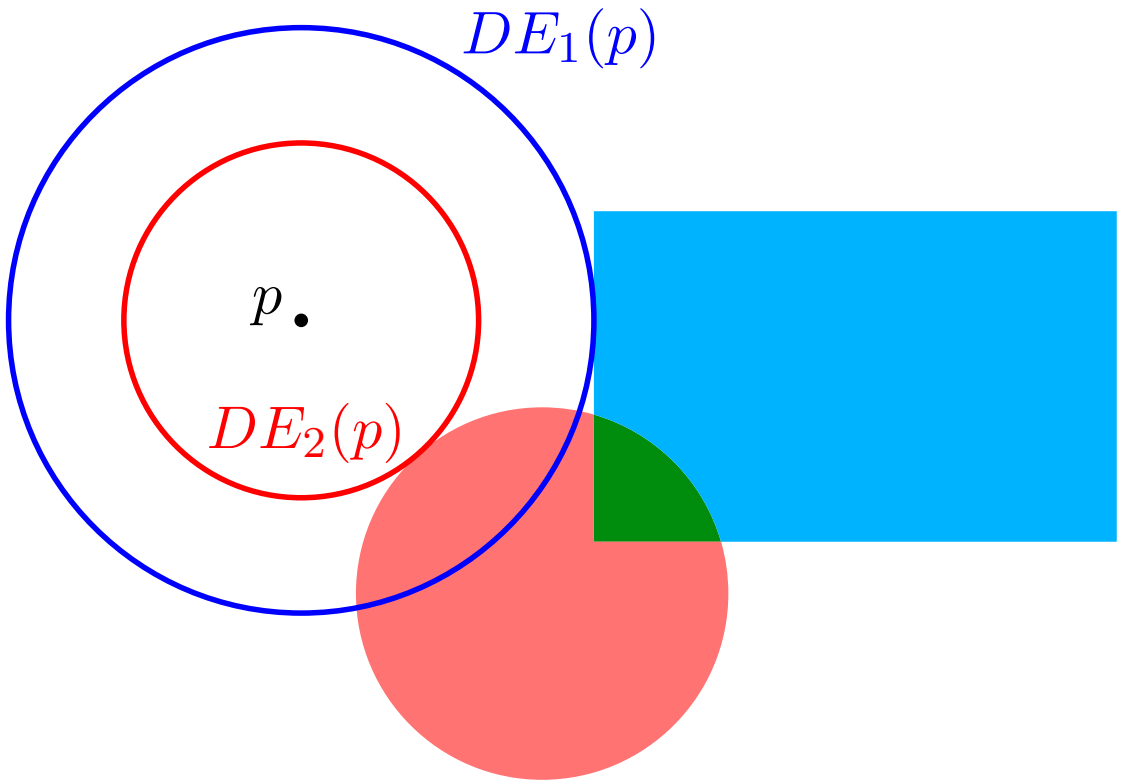
The combined distance estimator is the minimum.

Combining shapes



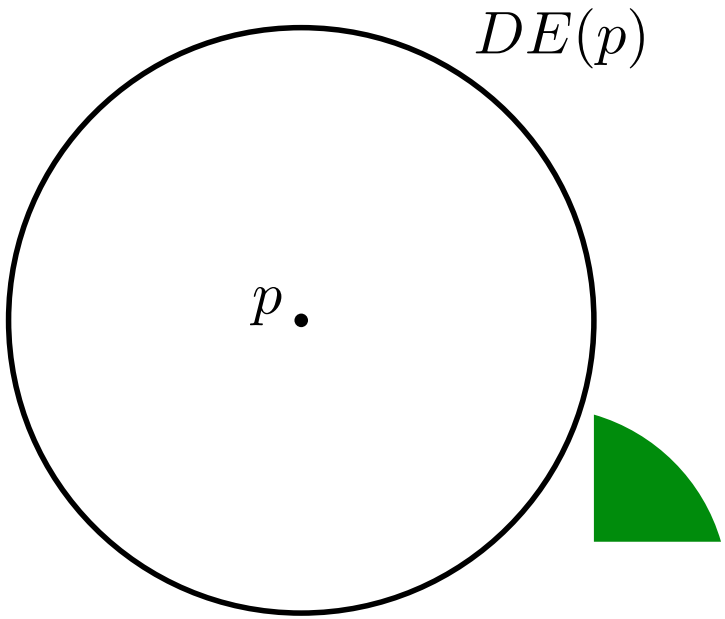
Similarly, the distance estimator of the intersection is the maximum.

Combining shapes



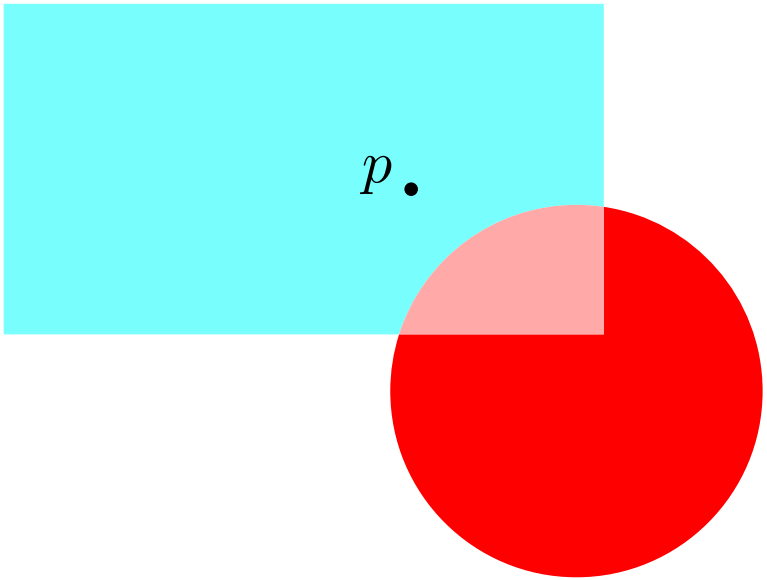
Similarly, the distance estimator of the intersection is the maximum.

Combining shapes



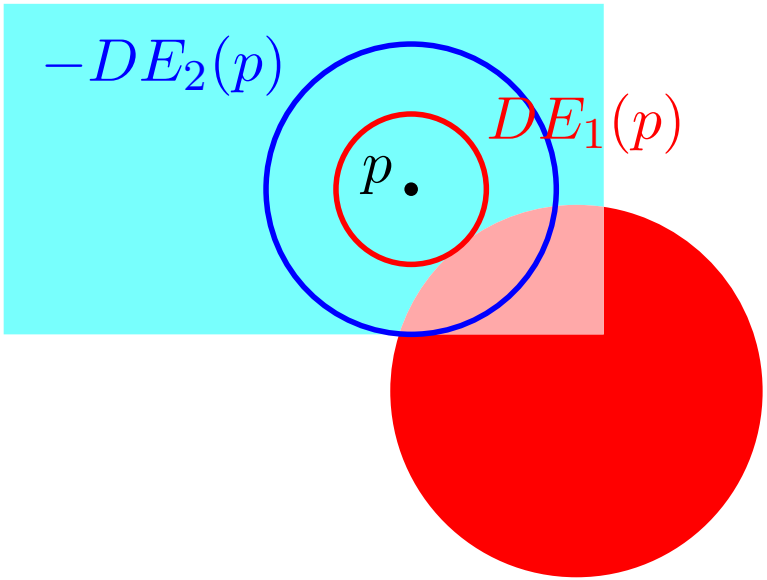
Similarly, the distance estimator of the intersection is the maximum.

Combining shapes



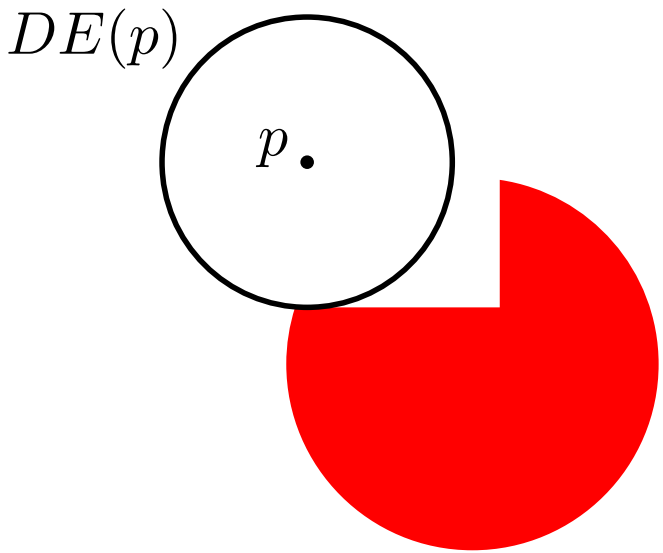
What if we want to subtract one shape from another?

Combining shapes



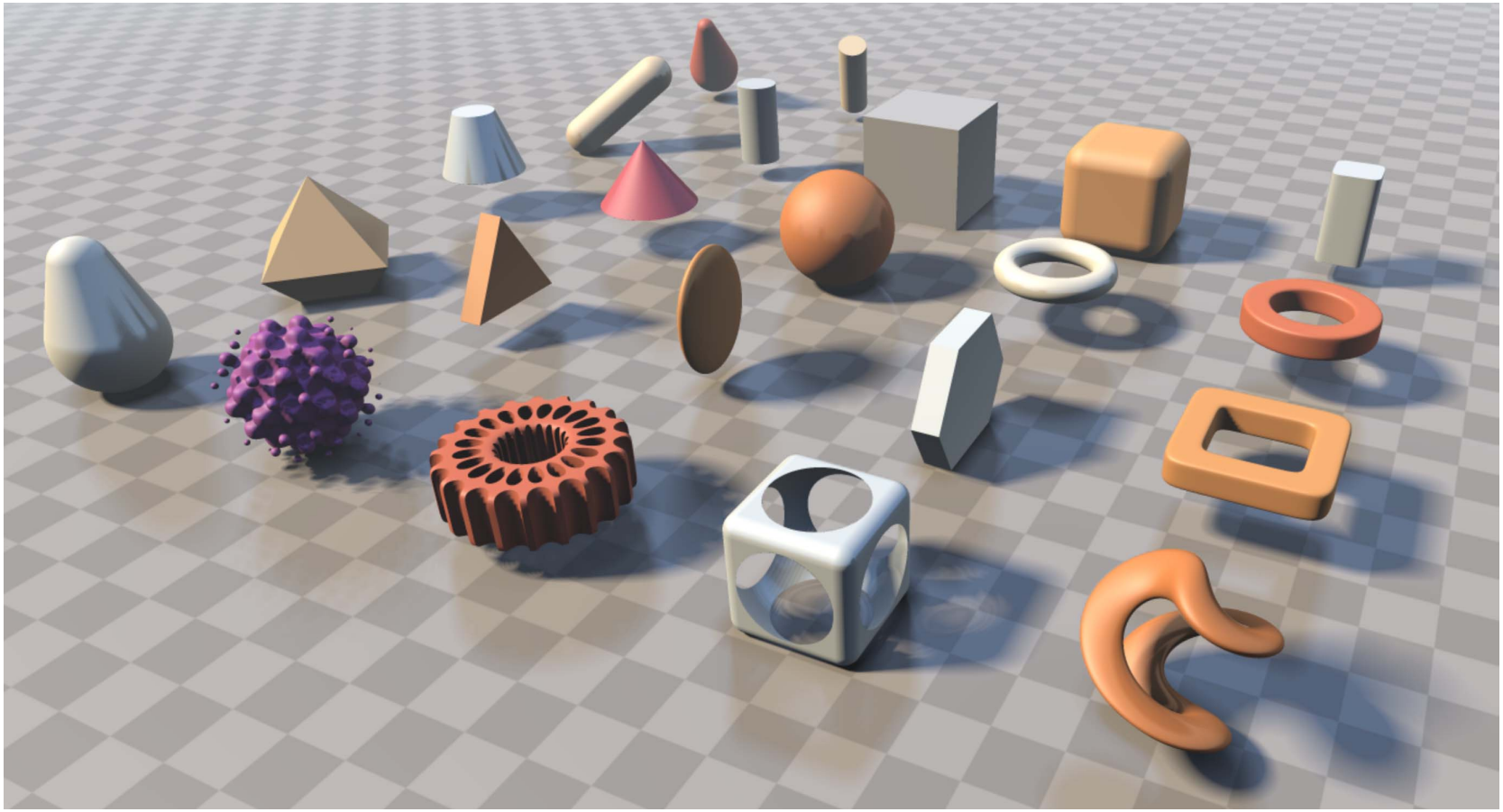
Negate the distance estimator of the shape to be subtracted, which gives a distance estimator for its complement shape.

Combining shapes



Then compute the maximum,
as we did for the intersection of shapes.

Combining shapes



Combining shapes

Infinite repetition

We already know the distance estimator for a sphere:

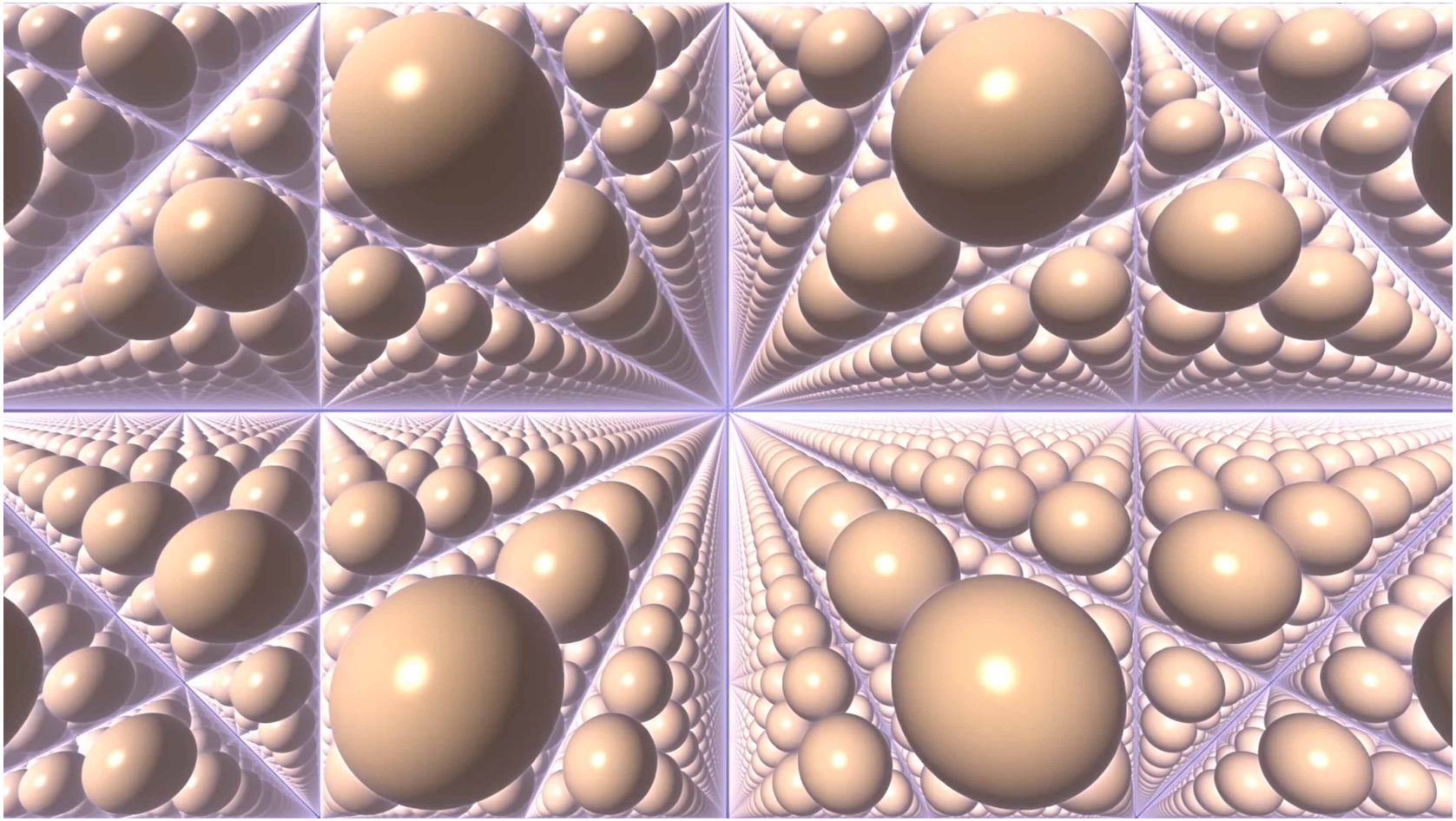
$$DE(p) = \|p\| - r.$$

Infinite repetition

Let us make a small modification, introducing a $\underline{\text{mod operator}}$:

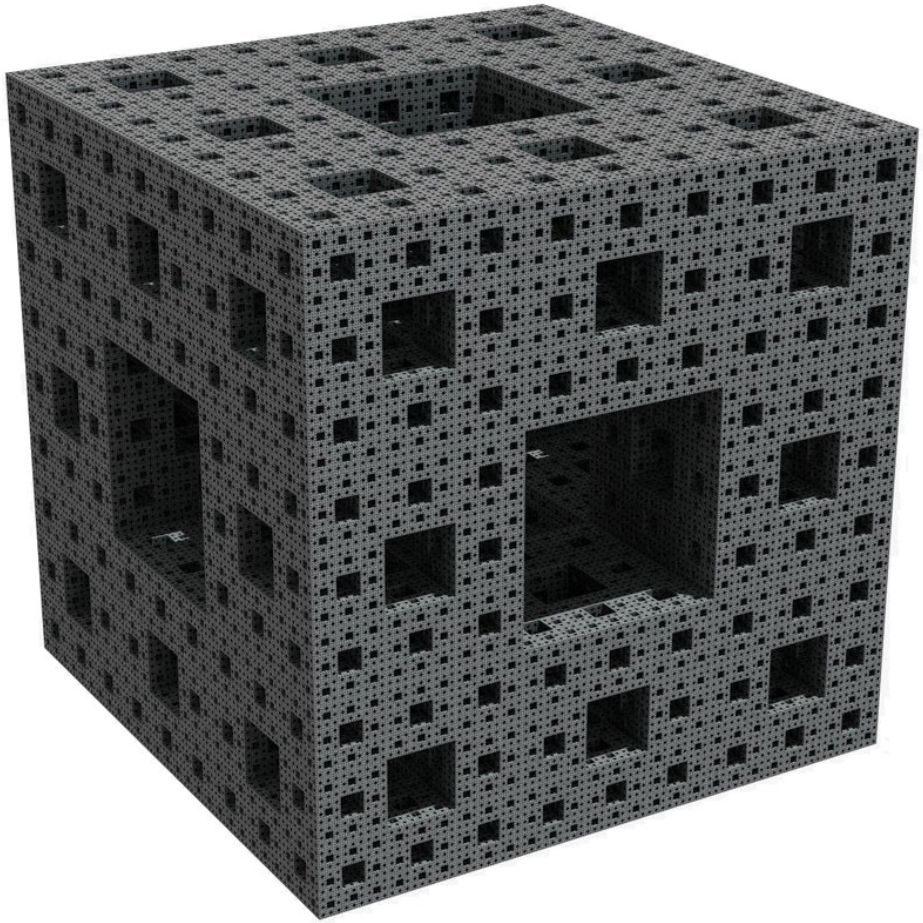
$$DE(p) = \|(p \bmod m) - m/2\| - r.$$

Infinite repetition



Suddenly we get infinitely many spheres, at roughly the same cost!
This would be unfeasible with rasterization or ray tracing.

Scale-invariant fractals



With similar methods, it is very easy to efficiently draw scale-invariant fractals such as the Menger sponge.

Generalizing the Mandelbrot set

The Mandelbrot set is defined from the iterations of the function

$$f_c(z) = z^2 + c,$$

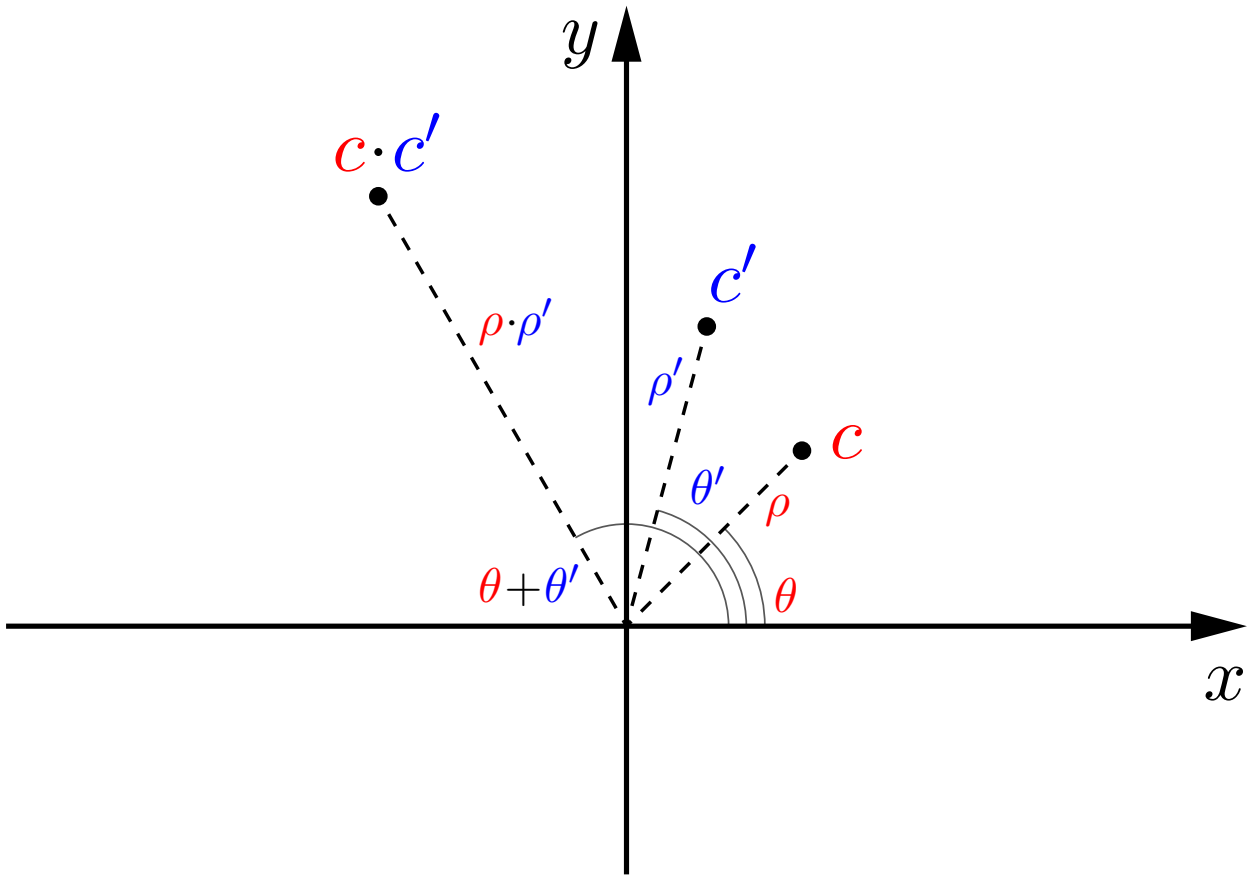
where z and c are complex numbers.

Is there a way to extend this notion to 3D?

- The term $+c$ poses no problem ($\mathbb{R}^3$ is a vector space)
- But how should we interpret the term z^2?

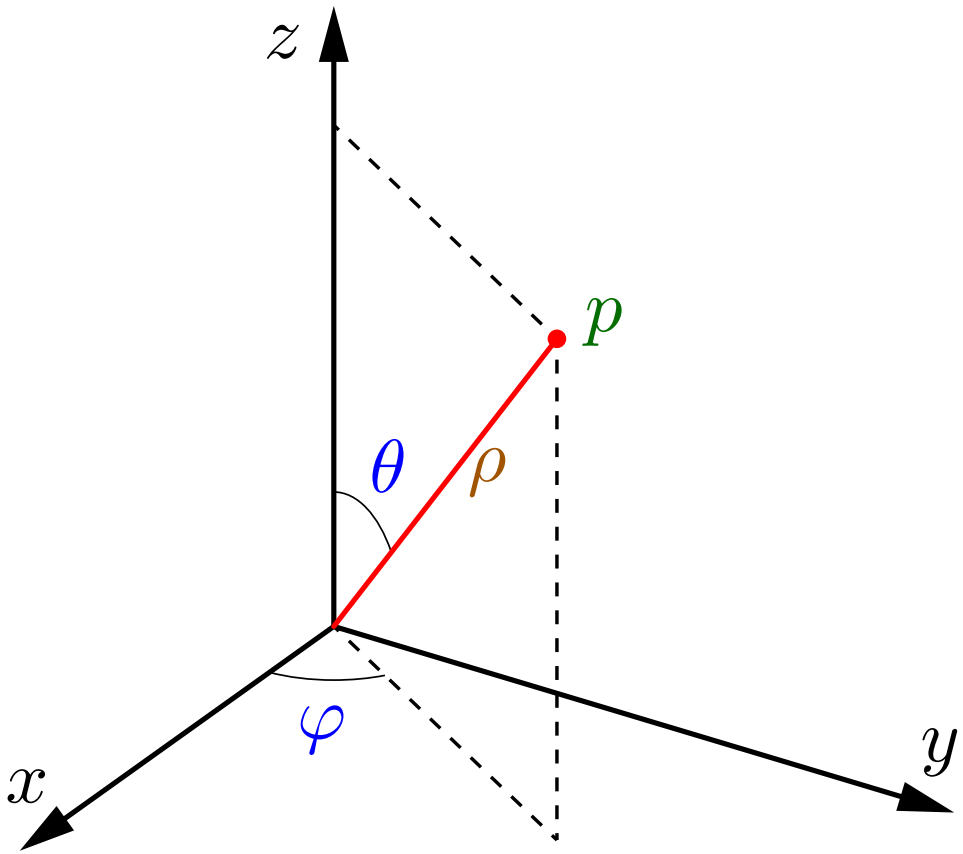
We need to generalize complex multiplication to some 3D space.

Generalizing complex multiplication



Recall the geometric interpretation of complex multiplication: in polar coordinates, multiply the norms and add the angles.

Generalizing complex multiplication



We can do the same thing in spherical coordinates, except that now we have two pairs of angles to add.

Distance estimator for the Mandelbrot set

To apply the ray marching technique to this 3D Mandelbrot set, we still need to define a distance estimator for it.

For the 2D Mandelbrot set, an approximated distance estimator can be defined based on the theory of electrostatic potential.

At the nth iteration of $f_c(0)$, we get the complex number $g_n(c) = f_c^n(0)$. The distance estimator is:

$$DE(c) = \lim_{n\to\infty} \frac{|g_n(c)| \cdot \ln |g_n(c)|}{2 \cdot |g'_n(c)|},$$

where $g'_n(c)$ is the complex derivative of $g_n(c)$ with respect to c.

John C. Hart, Daniel J. Sandin, Louis H. Kauffman
Ray tracing deterministic 3-D fractals
SIGGRAPH '89

Distance estimator for the Mandelbrot set

How do we compute the complex derivative of $g_n(c) = f_c^n(0)$*?*

From $f_c(z) = z^2 + c$, we have the recursion:

$$g_n(c) = f_c(g_{n-1}(c)) = (g_{n-1}(c))^2 + c.$$

Deriving with respect to c, we get:

$$g'_n(c) = 2 \cdot g_{n-1}(c) \cdot g'_{n-1}(c) + 1 = 2 \cdot f_c^{n-1}(0) \cdot g'_{n-1}(c) + 1.$$

Distance estimator for the Mandelbrot set

How do we compute the complex derivative of $g_n(c) = f_c^n(0)$*?*

From $f_c(z) = z^2 + c$, we have the recursion:

$$g_n(c) = f_c(g_{n-1}(c)) = (g_{n-1}(c))^2 + c.$$

Deriving with respect to c, we get:

$$g'_n(c) = 2 \cdot g_{n-1}(c) \cdot g'_{n-1}(c) + 1 = 2 \cdot f_c^{n-1}(0) \cdot g'_{n-1}(c) + 1.$$

Can we extend the same idea to the 3D Mandelbrot set?

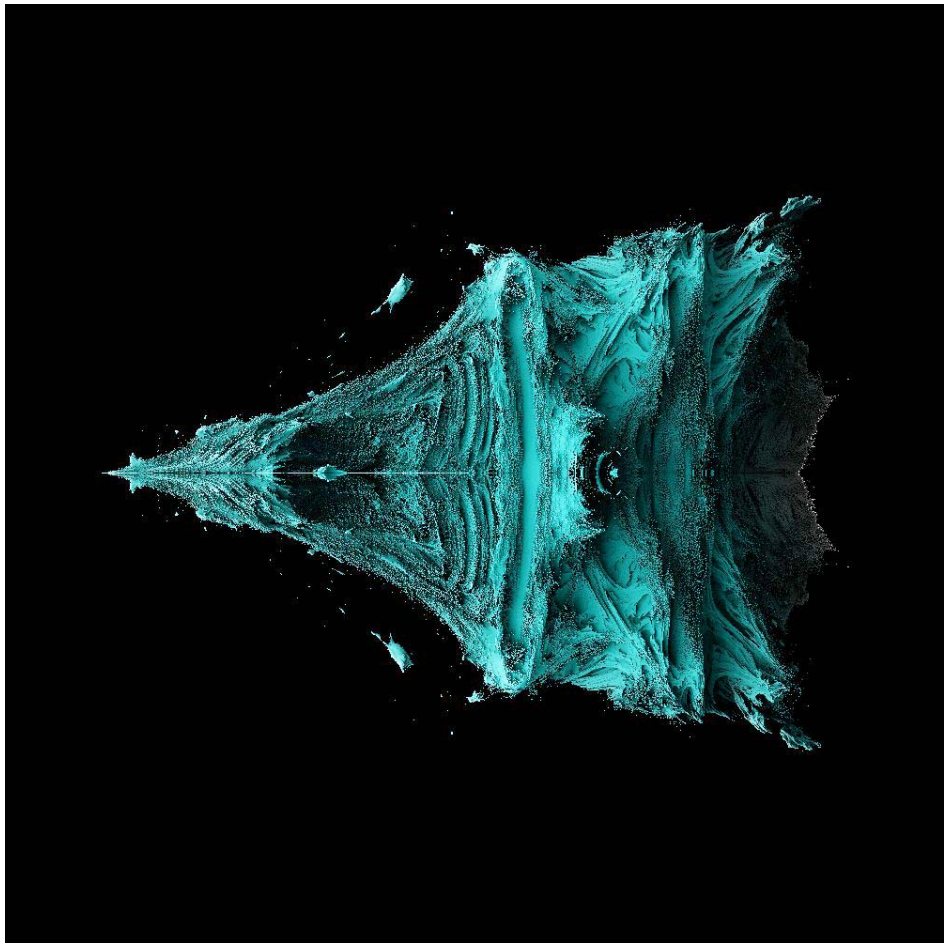
Problem 1: what is the potential of the 3D Mandelbrot set?

Tentative solution: use the same potential function as in 2D!

Problem 2: the complex derivative is a complex number, but for points with three components we would get a 3x3 Jacobian matrix.

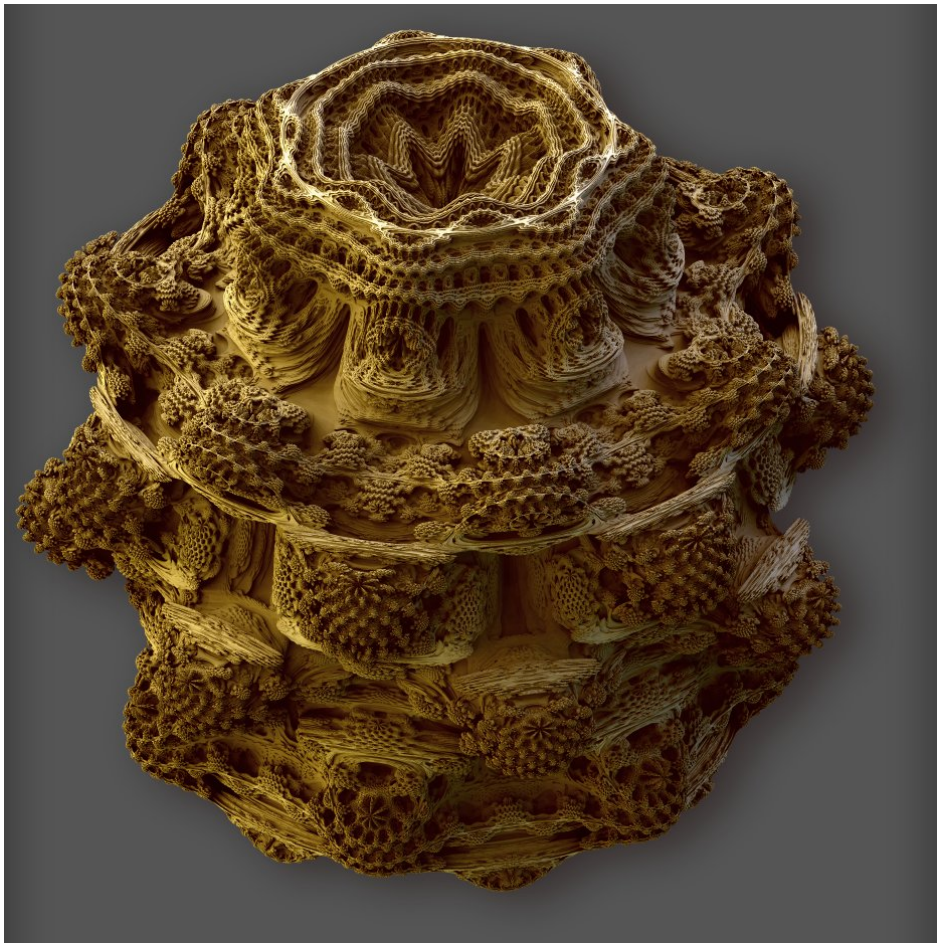
Tentative solution: use the same recursive formula as in 2D, and take norms $\|g'_n(c)\|$ instead of moduli $|g'_n(c)|$

The Mandelbulb



The result is not so aesthetically pleasing and does not exhibit a lot of self-similarity.

The Mandelbulb



But if we replace $f_c(z) = z^2 + c$ with $f_c(z) = z^8 + c$,
we obtain the Mandelbulb.

Another approach

Is there another way to extend complex multiplication to 3D?

Let us add to 1 and i a third component j. If we assume the distributive law, we only have to fill out this table:

$\times$	$\mathbf{1}$	$\mathbf{i}$	$\mathbf{j}$
$\mathbf{1}$	1	i	j
$\mathbf{i}$	i	-1	?
$\mathbf{j}$	j	?	?

Another approach

Is there another way to extend complex multiplication to 3D?

Let us add to 1 and i a third component j. If we assume the distributive law, we only have to fill out this table:

$\times$	$\mathbf{1}$	$\mathbf{i}$	$\mathbf{j}$
$\mathbf{1}$	1	i	j
$\mathbf{i}$	i	-1	$?$
$\mathbf{j}$	j	$?$	$?$

Unfortunately, if we also want associativity, we get

$$i \cdot (i \cdot j) = (i \cdot i) \cdot j = -j,$$

but trying to solve the equation $i \cdot ? = -j$ always causes j to be a linear combination of 1 and i, making the space 2-dimensional.

There is no solution in 3D.

Quaternions

A possible workaround is using 4 components instead of 3, which yields the quaternions $\mathbb{H}$:

$\times$	$\mathbf{1}$	$\mathbf{i}$	$\mathbf{j}$	$\mathbf{k}$
$\mathbf{1}$	1	i	j	k
$\mathbf{i}$	i	-1	k	$-j$
$\mathbf{j}$	j	$-k$	-1	i
$\mathbf{k}$	k	j	$-i$	-1

Quaternions

A possible workaround is using 4 components instead of 3, which yields the quaternions $\mathbb{H}$:

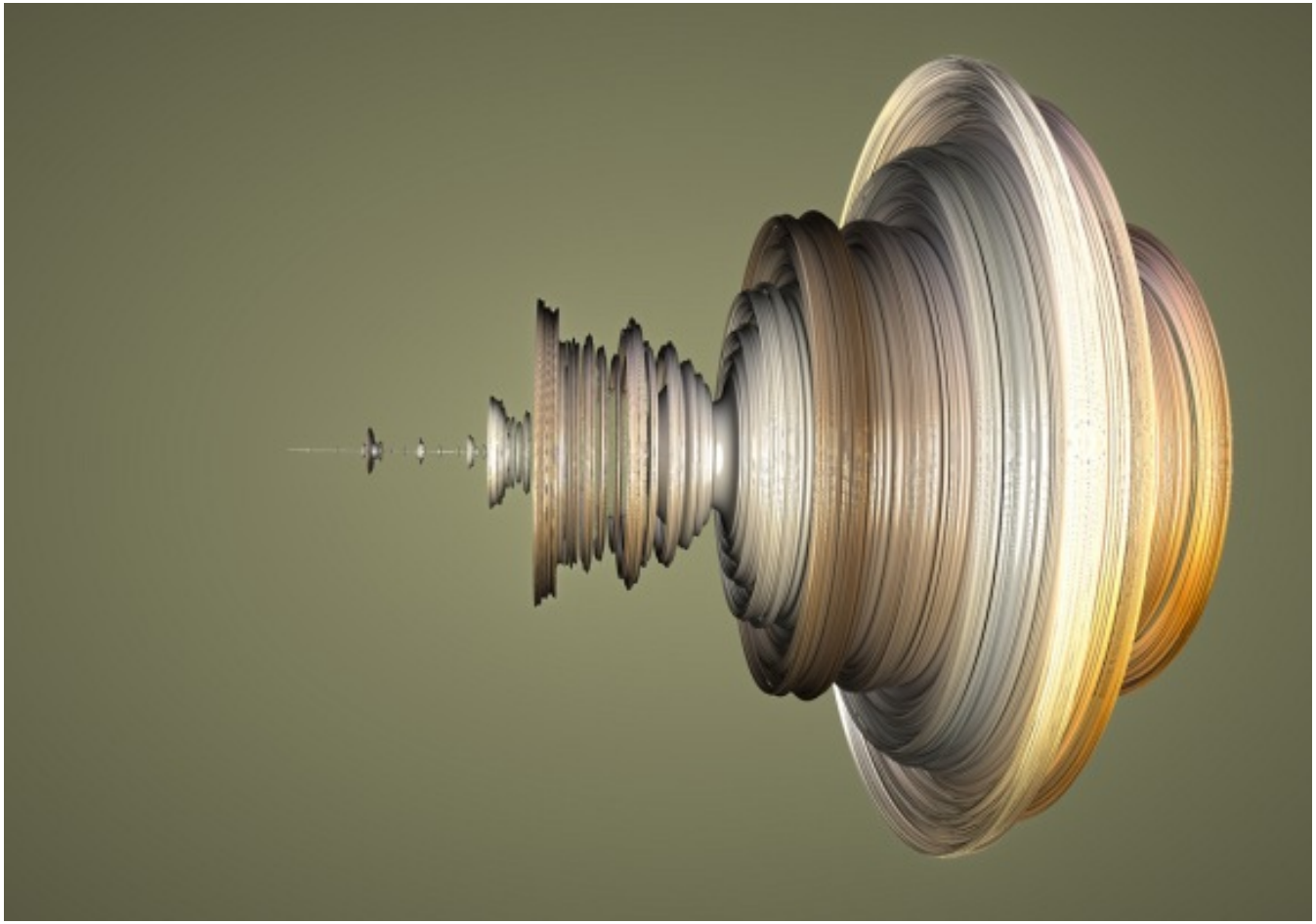
$\times$	$\mathbf{1}$	$\mathbf{i}$	$\mathbf{j}$	$\mathbf{k}$
$\mathbf{1}$	1	i	j	k
$\mathbf{i}$	i	-1	k	$-j$
$\mathbf{j}$	j	$-k$	-1	i
$\mathbf{k}$	k	j	$-i$	-1

Theorem (Frobenius, 1877)

The only finite-dimensional vector spaces on the real numbers with associative and distributive multiplication and division are $\mathbb{R}$, $\mathbb{C}$, $\mathbb{H}$.

Problem: how do we visualize a 4D object?

Quaternion Mandelbrot set



If we keep the k component at a fixed value,
we just get a solid of revolution.

Quaternion Mandelbrot set

But if we apply the transformation $p_y := -p_y$ at each iteration, we obtain a more interesting shape.

Quaternion Mandelbrot-like sets

By adding more parameters to the iteration function, we get higher-dimensional shapes that can be “sliced” in various ways.

Alan Norton
Generation and display of geometric fractals in 3-D
SIGGRAPH ’82

Summary and open problems

We illustrated the ray marching technique with distance estimators for real-time rendering of 3D fractals.

Advantages: good compromise between image quality and speed; many effects that are usually expensive come almost for free with ray marching: ambient occlusion, glow, soft shadows, etc.

Disadvantages: no dedicated hardware support; still slower than rasterization for scenes with many objects.

Summary and open problems

We illustrated the ray marching technique with distance estimators for real-time rendering of 3D fractals.

Advantages: good compromise between image quality and speed; many effects that are usually expensive come almost for free with ray marching: ambient occlusion, glow, soft shadows, etc.

Disadvantages: no dedicated hardware support; still slower than rasterization for scenes with many objects.

Future work:

- Provide theoretical foundations to these rendering heuristics.
  - Why does the 2D Mandelbrot potential function work in 3D?
  - For what other iteration functions does this approach work?
  - Why can we get rid of Jacobians in the distance estimators?

Summary and open problems

We illustrated the ray marching technique with distance estimators for real-time rendering of 3D fractals.

Advantages: good compromise between image quality and speed; many effects that are usually expensive come almost for free with ray marching: ambient occlusion, glow, soft shadows, etc.

Disadvantages: no dedicated hardware support; still slower than rasterization for scenes with many objects.

Future work:

- Provide theoretical foundations to these rendering heuristics.
  - Why does the 2D Mandelbrot potential function work in 3D?
  - For what other iteration functions does this approach work?
  - Why can we get rid of Jacobians in the distance estimators?
- Design video games around these concepts: navigation and interaction with fractal environments at various scales, etc.

Summary and open problems