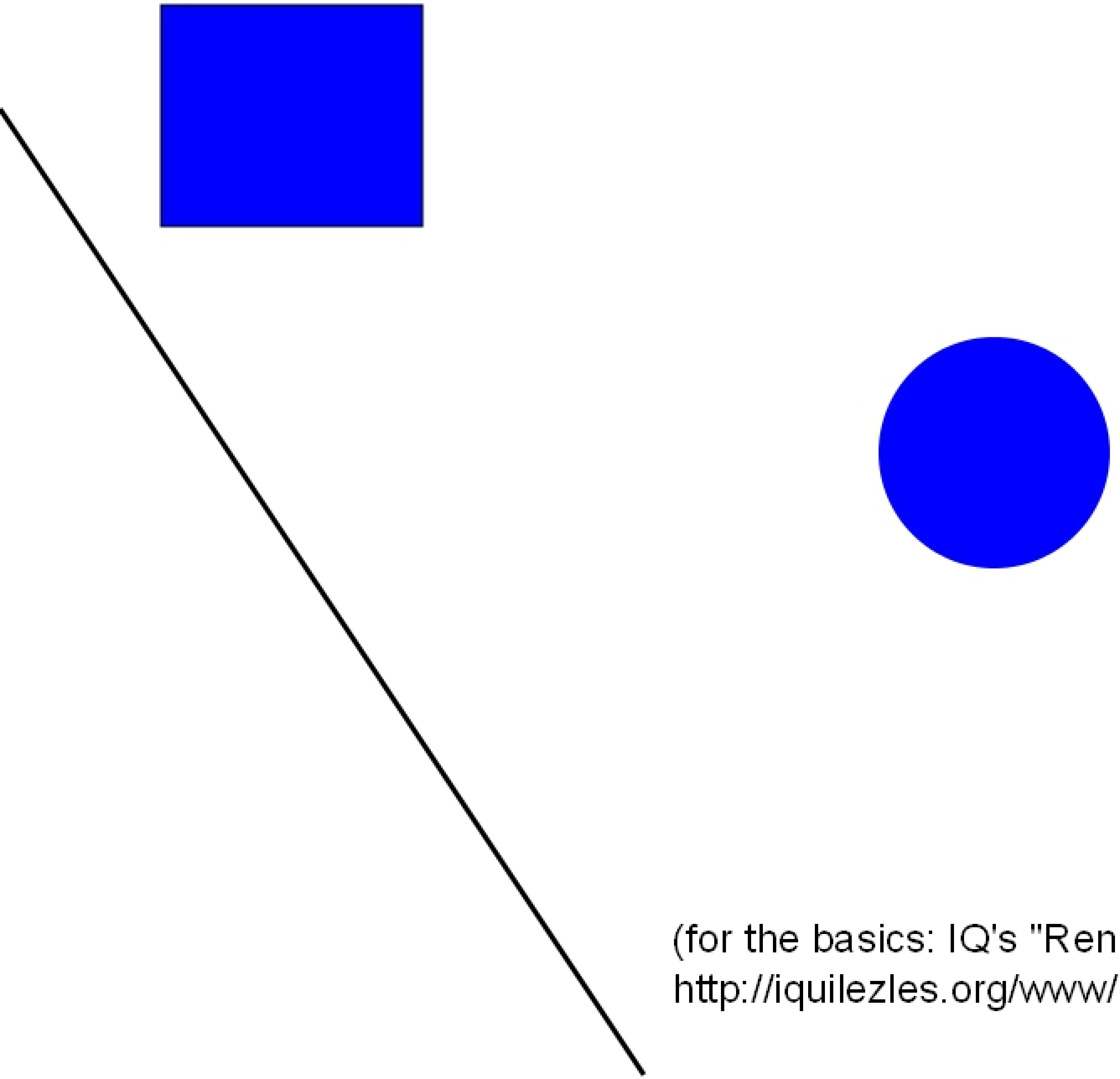
Rendering Mandelbox fractals faster with Cone Marching

Seven/Fulcrum

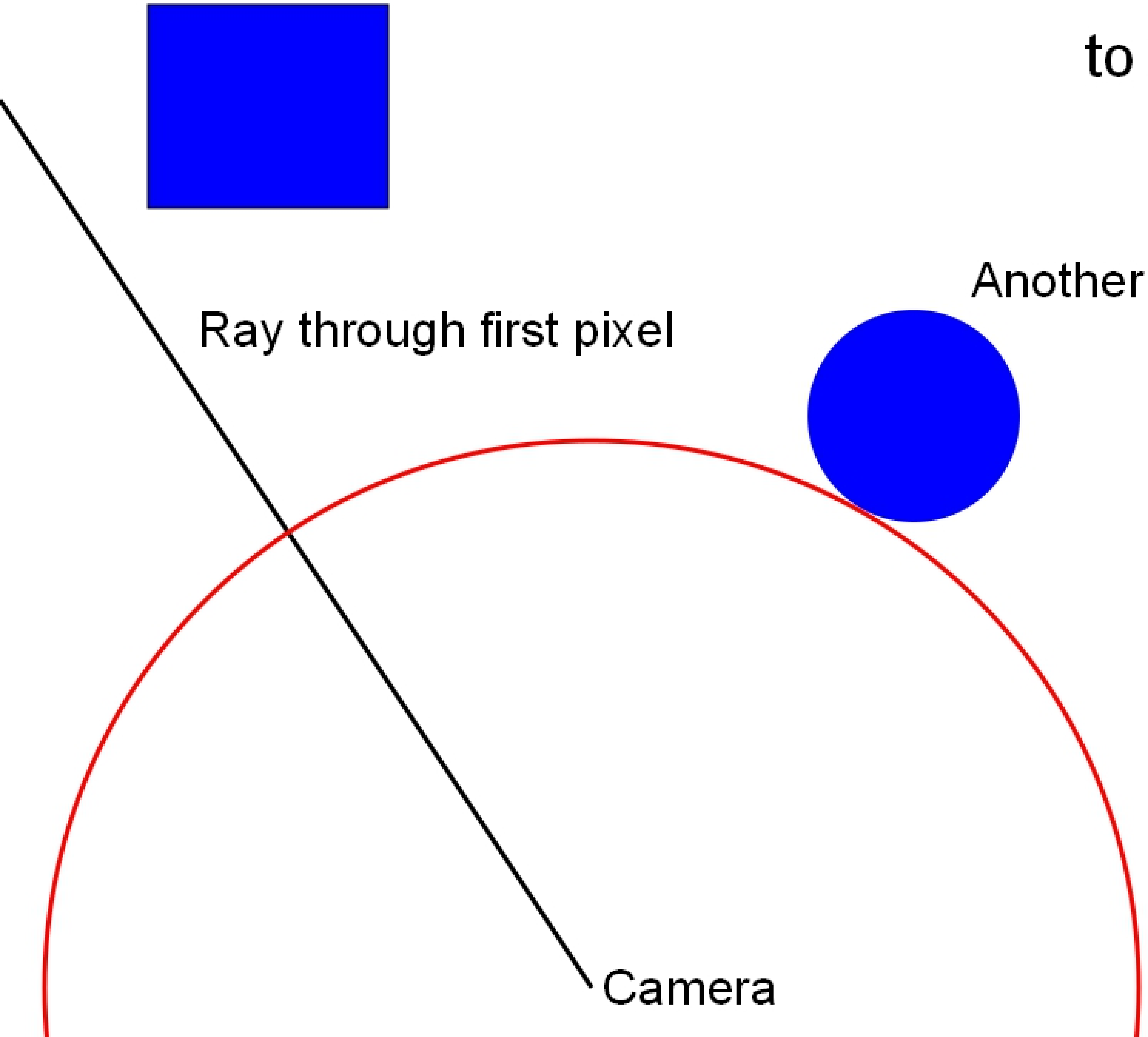
This is the longer version of the presentation given at Revision 2012

Raymarching:
A quick refresh.



(for the basics: IQ's "Rendering Worlds With 2 Triangles"
http://iquilezles.org/www/material/nvscene2008/rwwtt.pdf)

Find distance to closest object

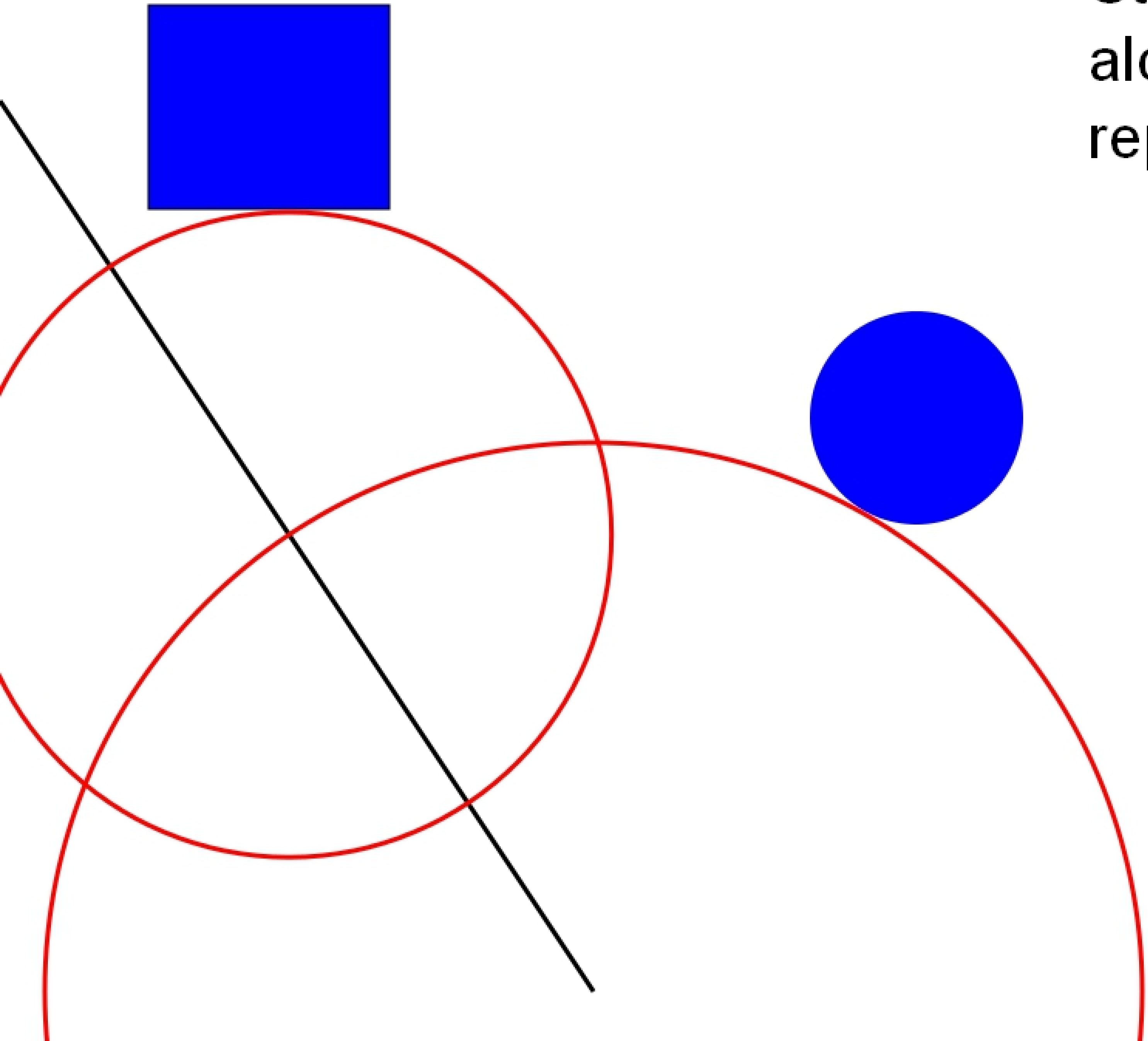


Object to draw

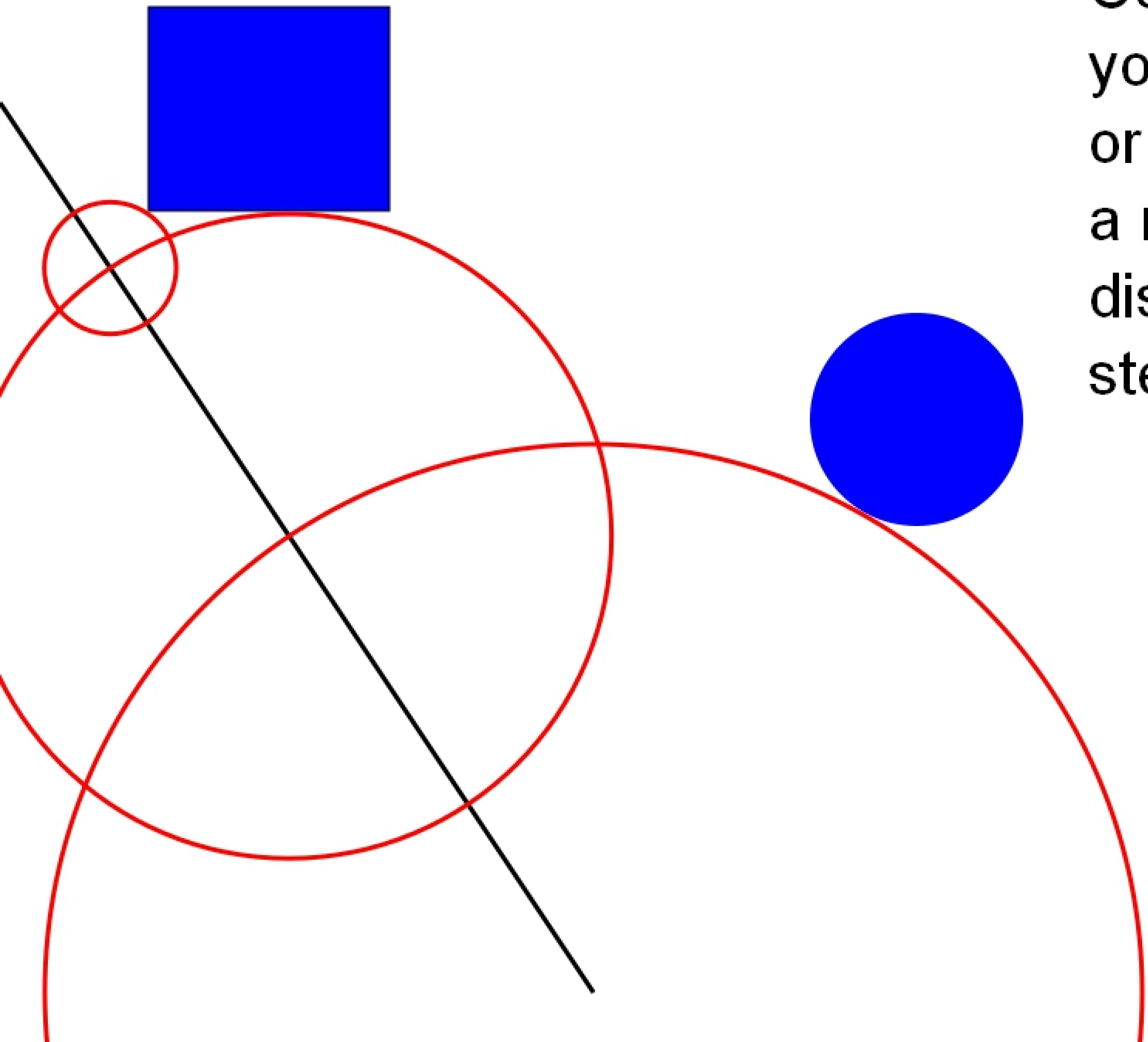
Another object to draw

Ray through first pixel

Camera

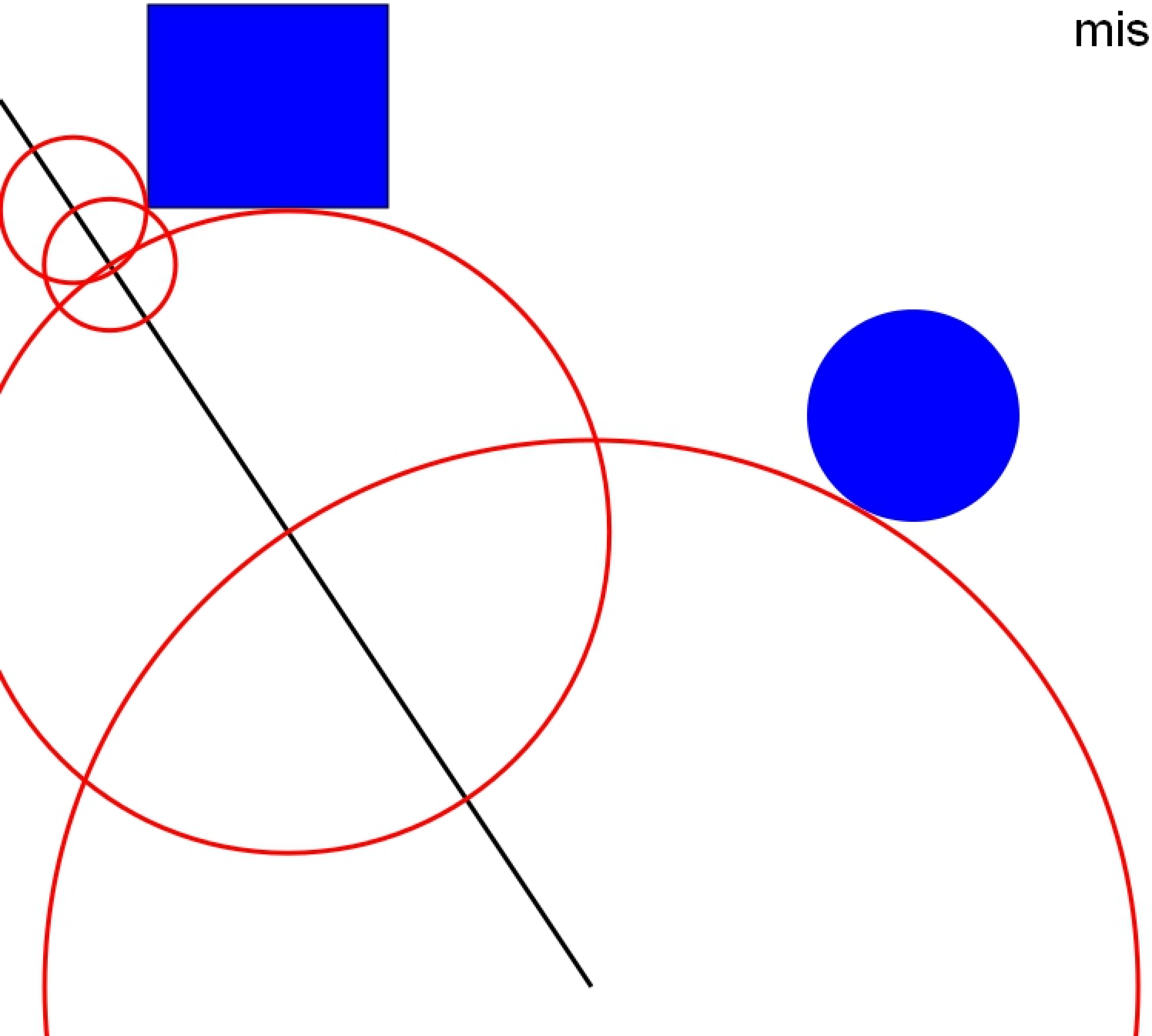


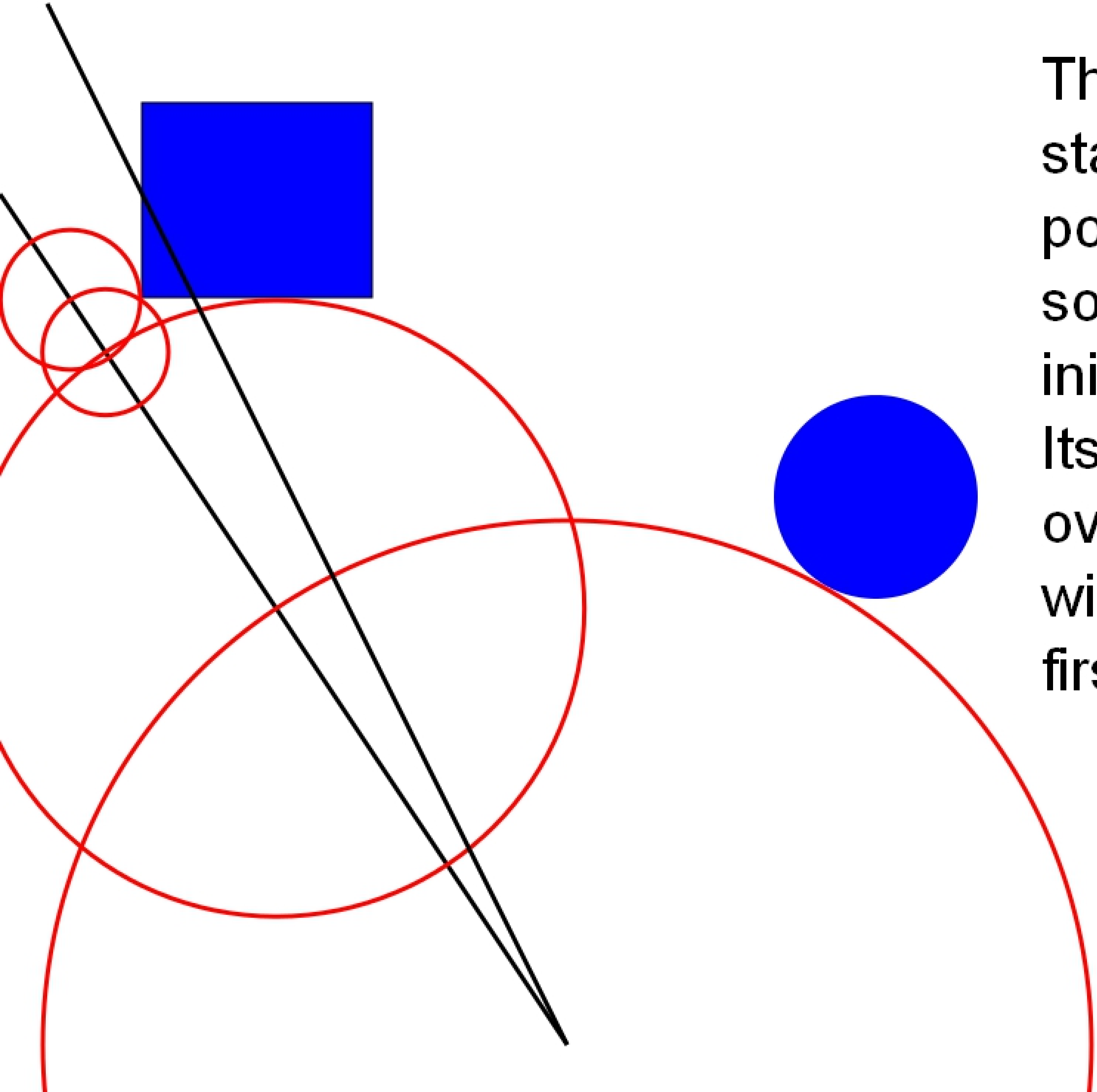
Step that distance along the ray and repeat



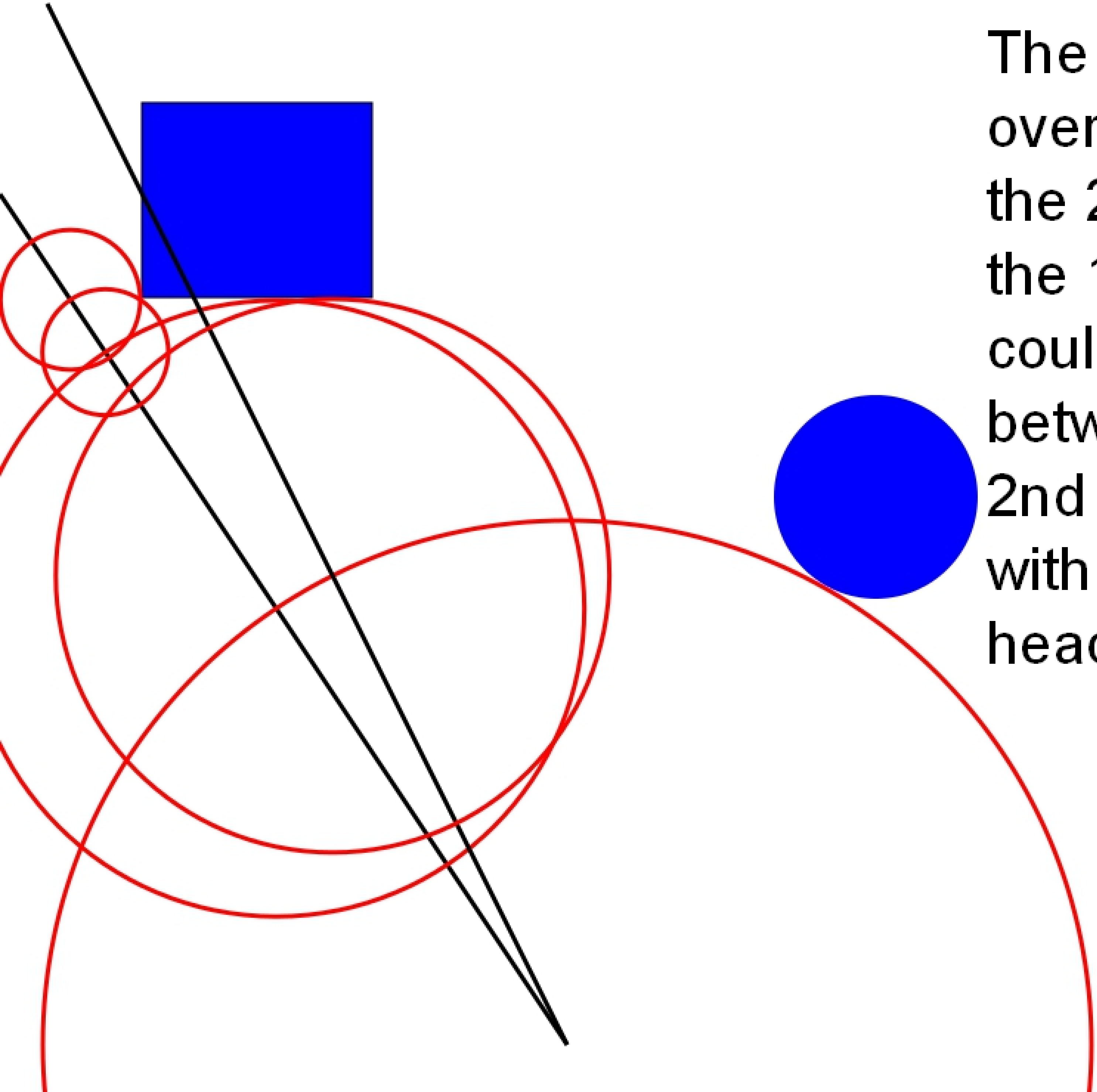
Continue until you hit an object or (if miss) reach a maximum distance or nr of steps

Here the first ray misses

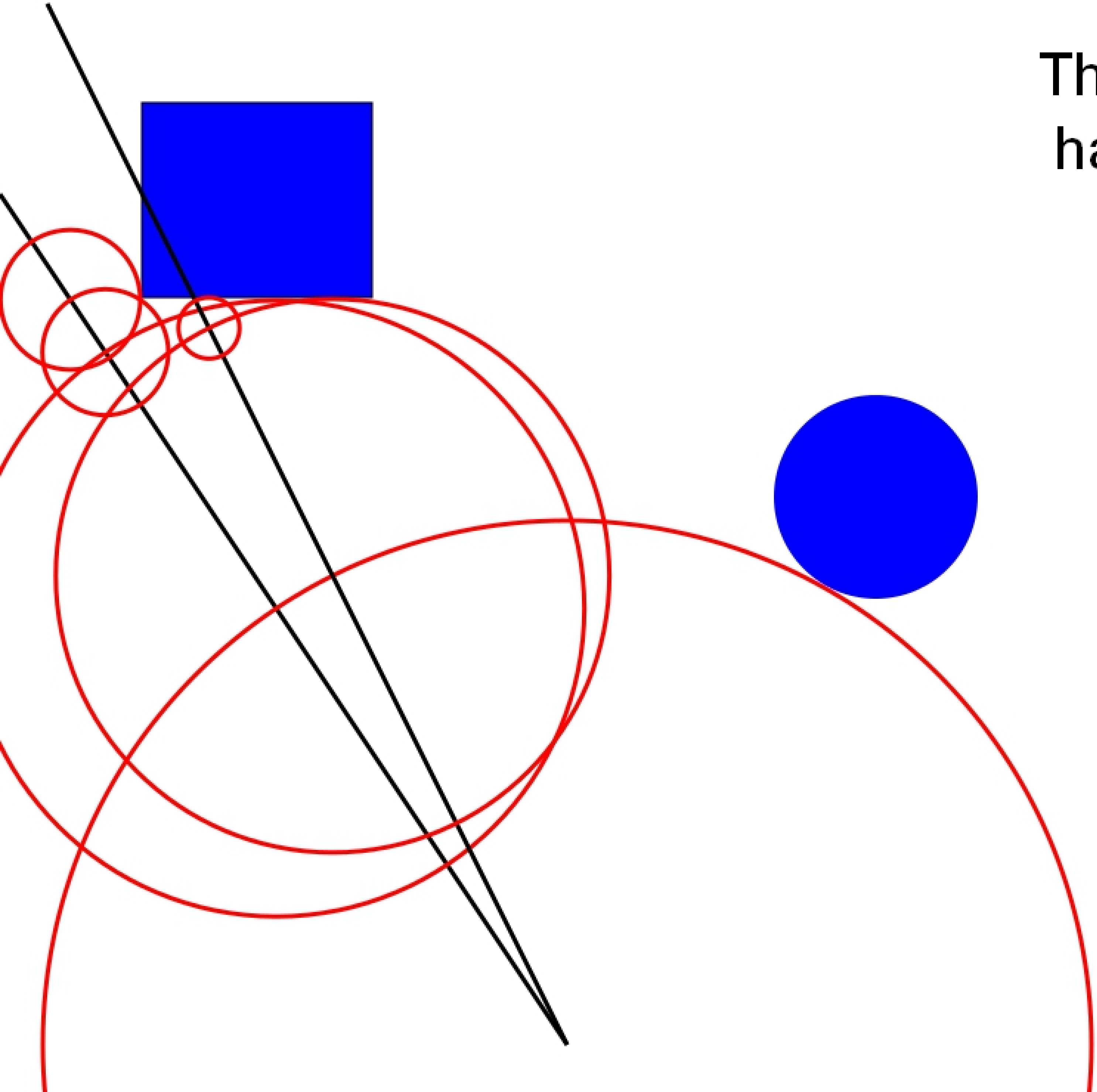




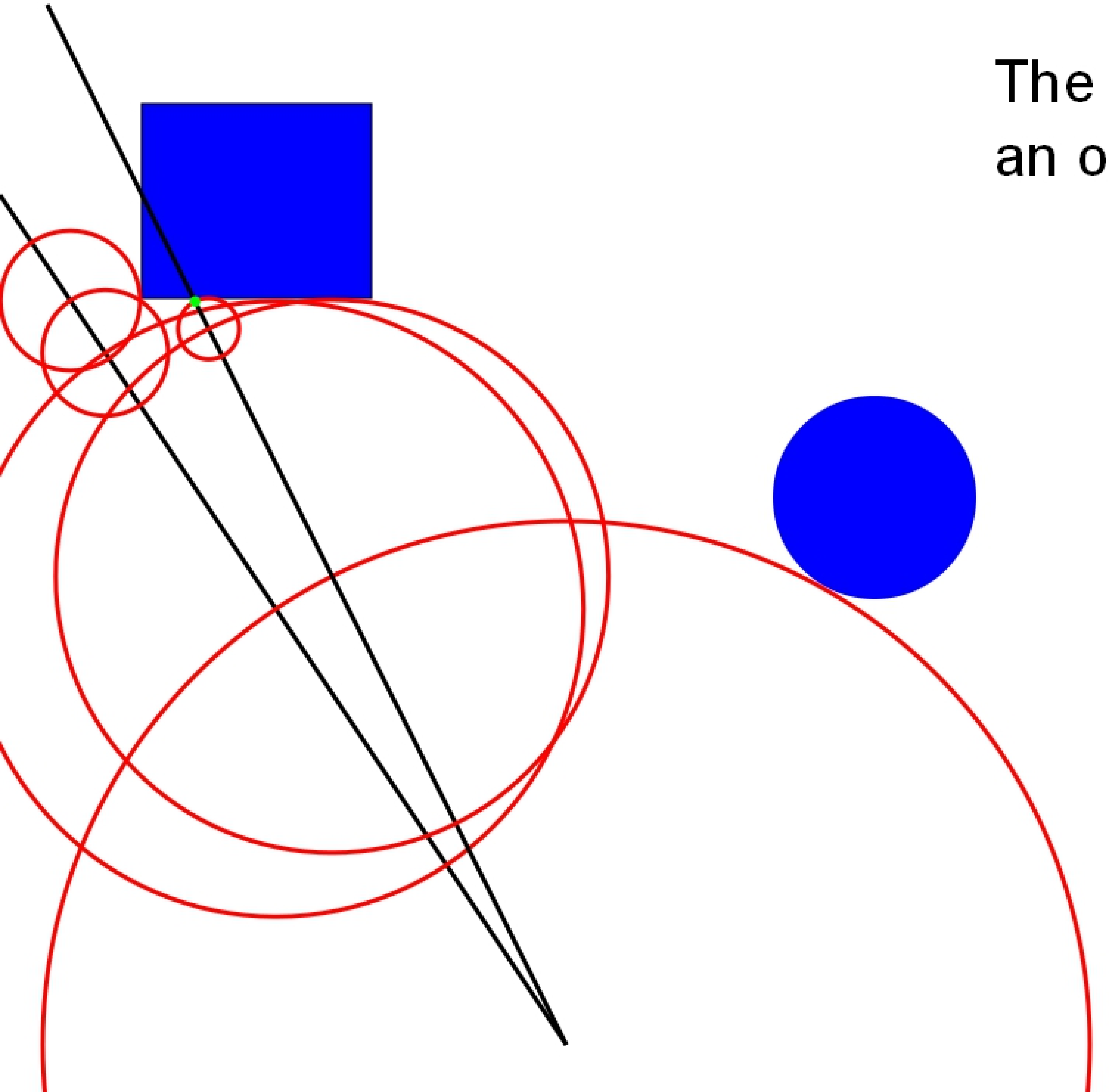
The second ray starts at the same point (camera), so has the same initial distance. Its first sphere overlaps 100% with that of the first ray.



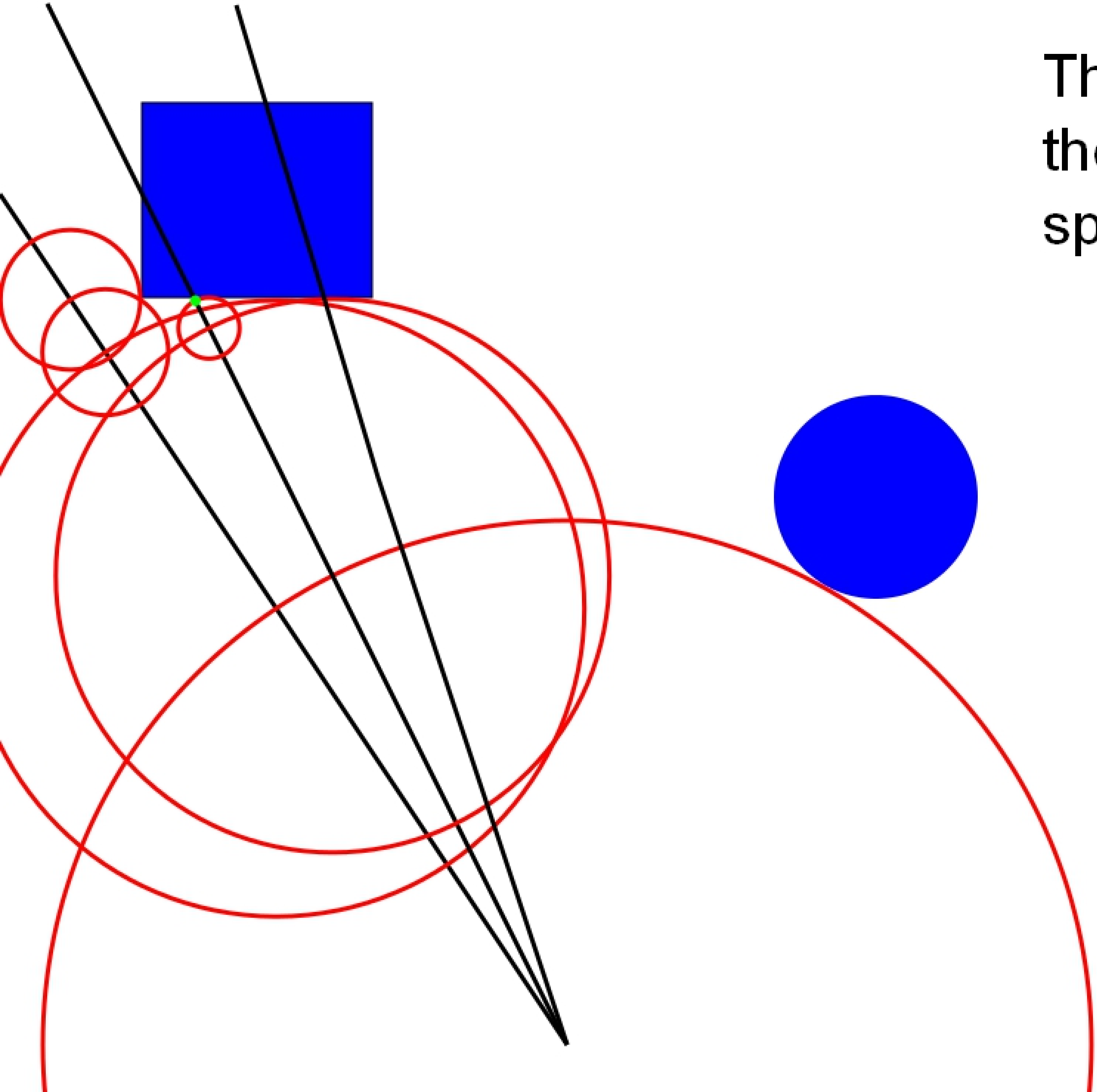
The second sphere overlaps a lot with the 2nd sphere of the 1st ray. If we could share those between rays, the 2nd ray could start with a large headstart.



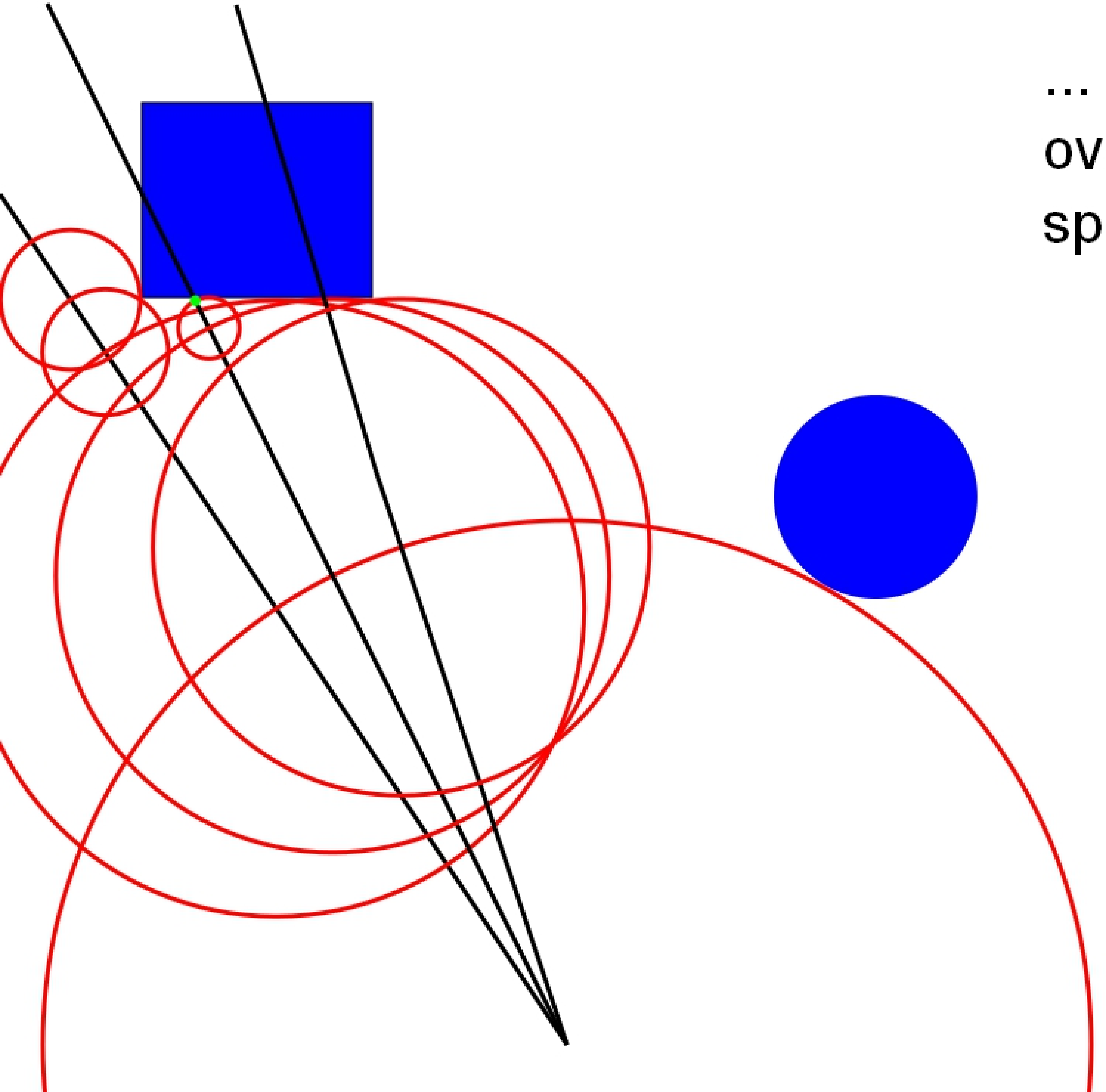
The 3rd sphere
has no overlap



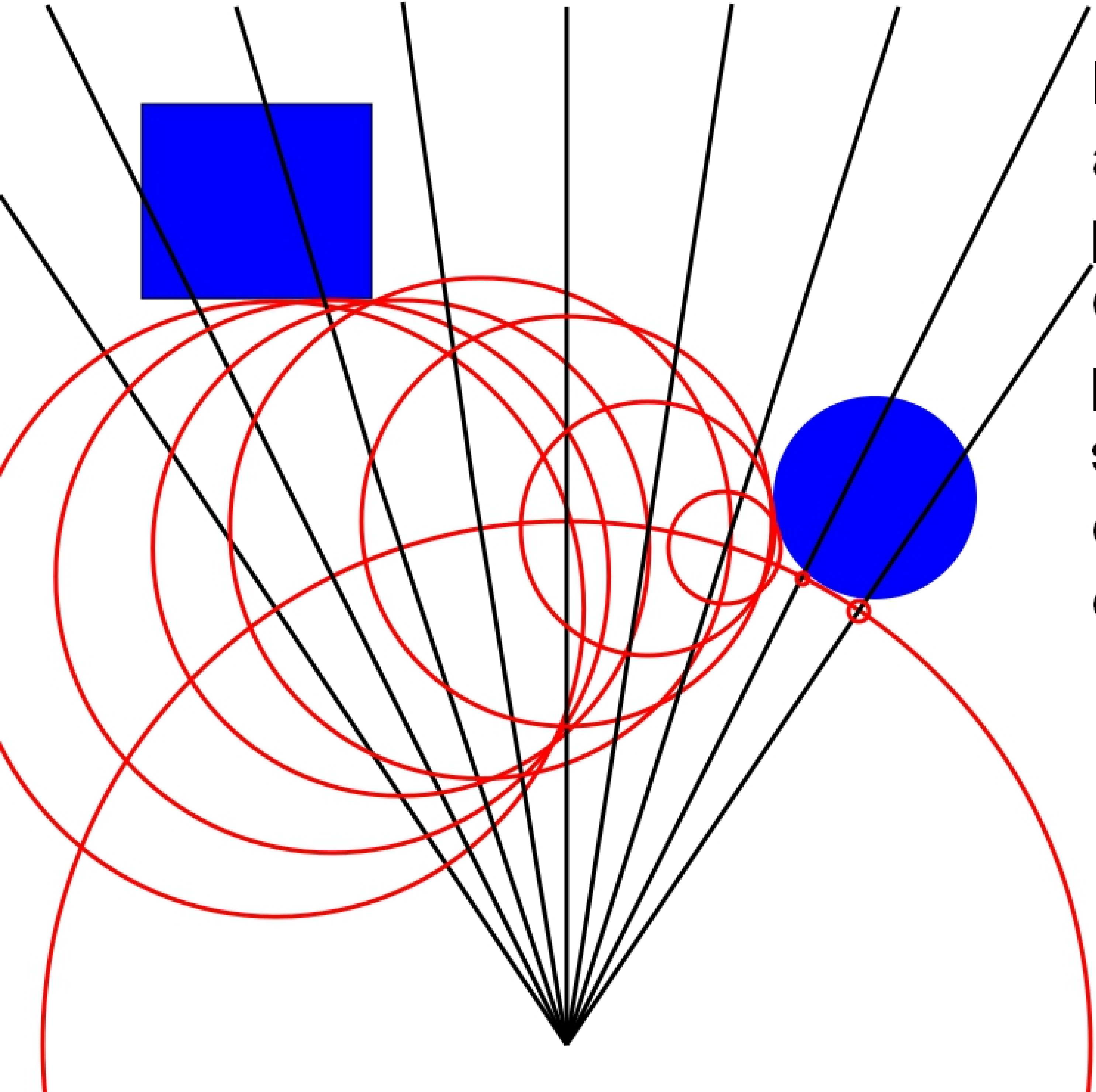
The 2nd ray hits an object.



The 3th ray has the same initial sphere again.

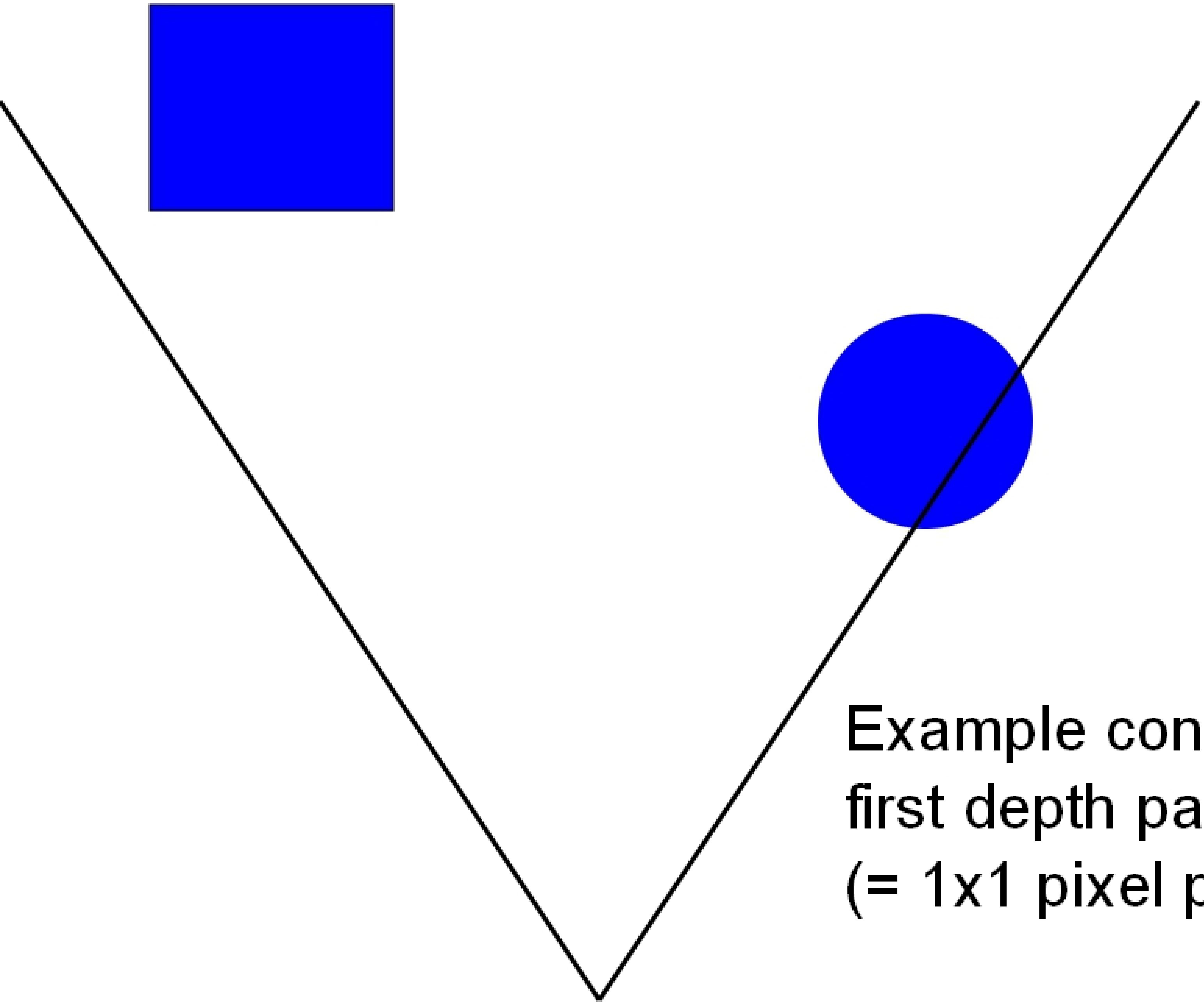


... and an overlapping 2nd sphere

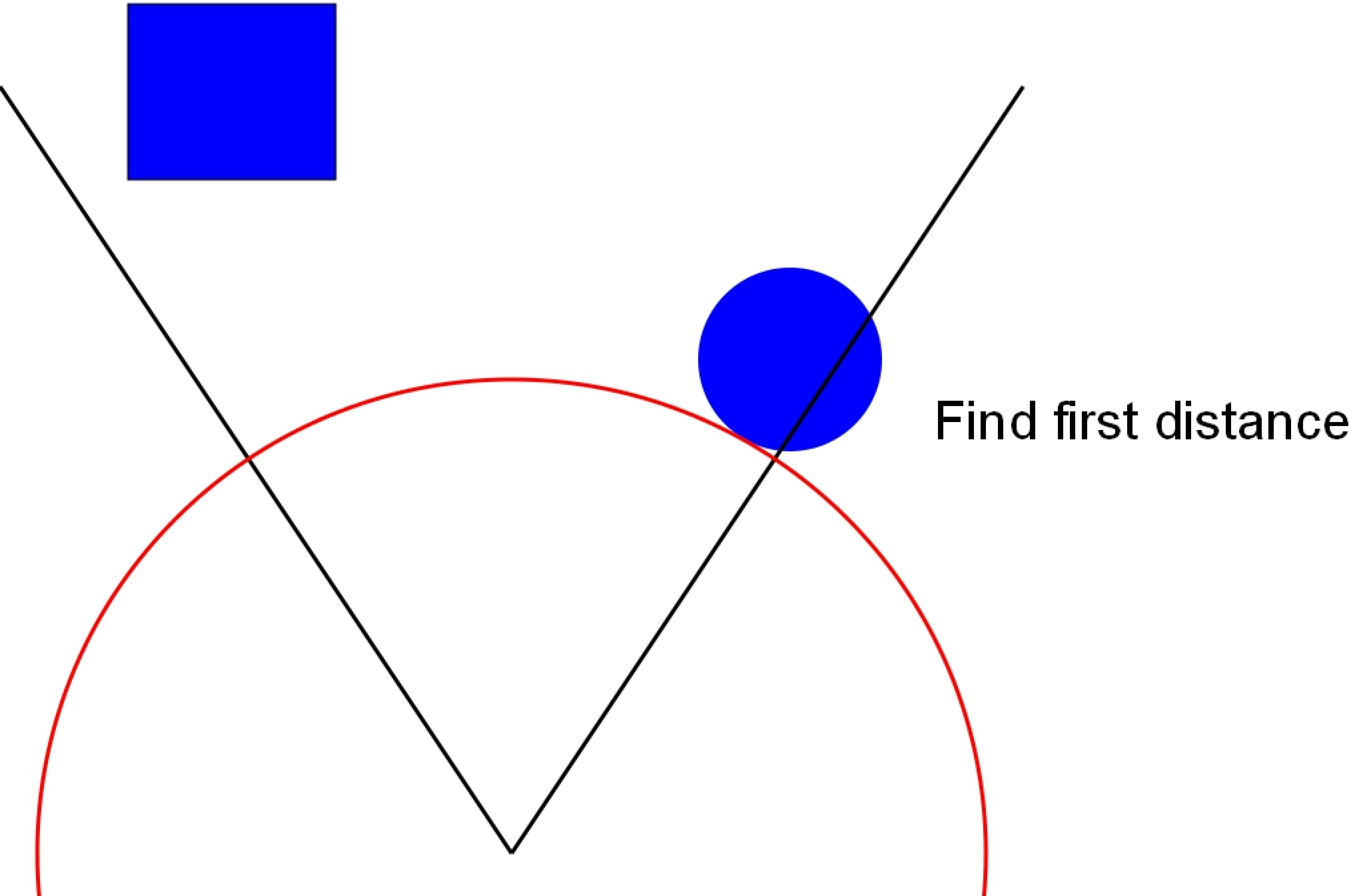


Because rays are marched in parallel on the GPU, it's not possible to share the overlapping distances

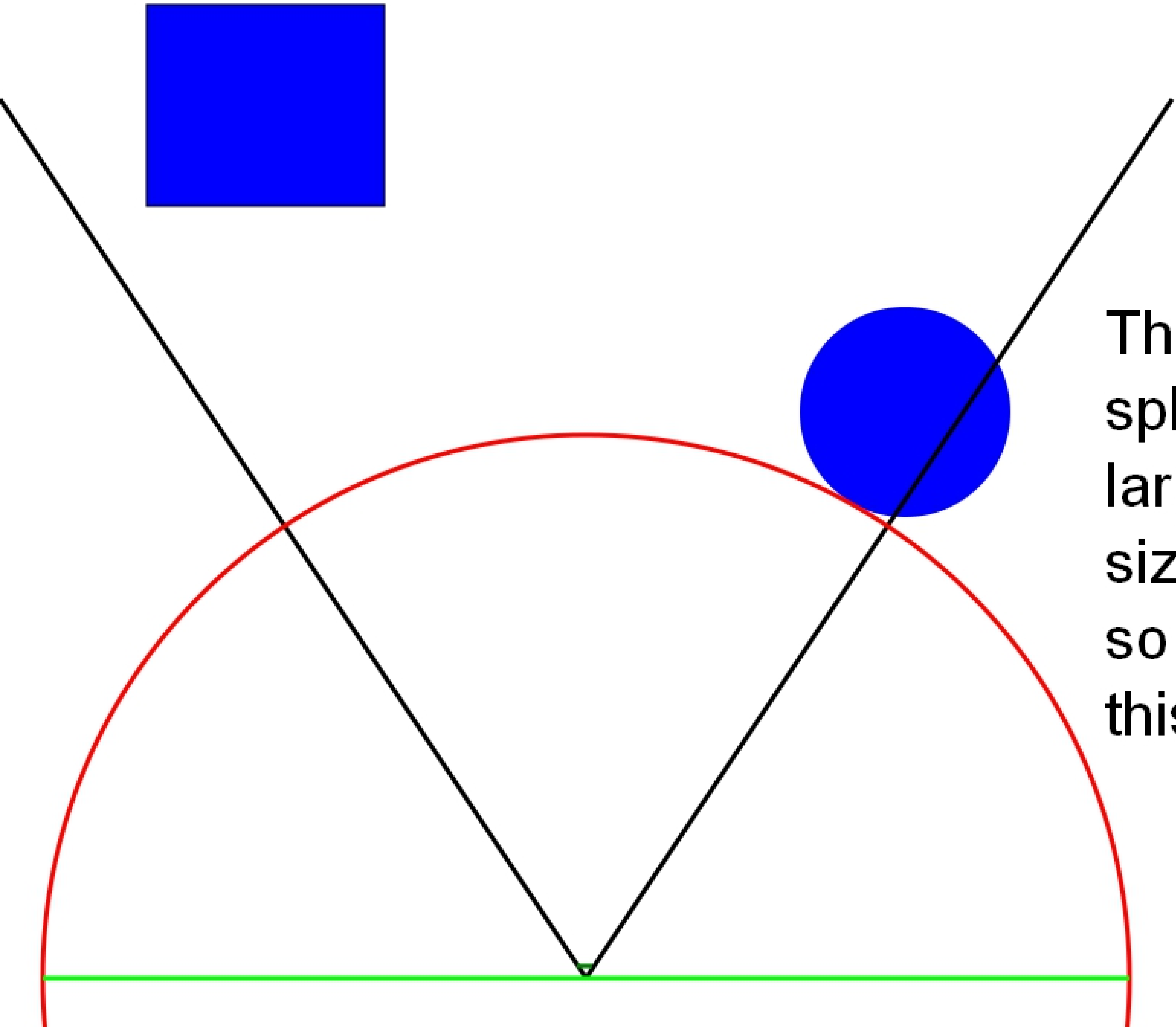
- What is cone marching: a way to share initial distance estimations between neighbouring pixels -> speedup!
- Split your shader in 2 parts, 1 for depth, 1 for color.
- The depth is calculates in multiple passes.
- Each pass doubles the resolution, and takes the result of the previous pass as input.
- Instead of marching along a ray until something is hit, the depth pass marches along the center of a cone, until an object is close enough to intersect the cone.
- Since cones get thinner when the resolution doubles, each pass gets progressively closer to the true depth.



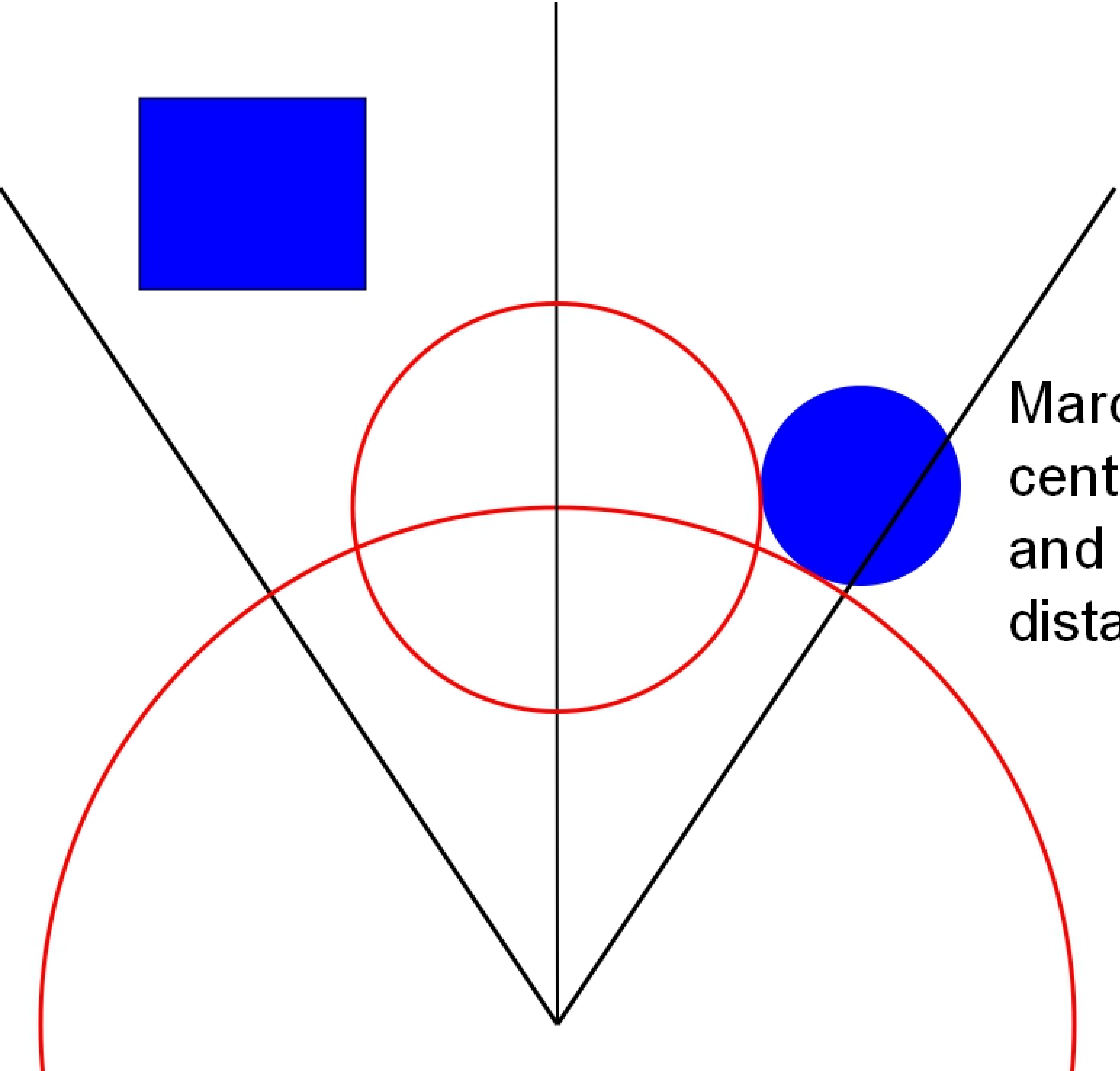
Example cone march:
first depth pass: 1 cone
(= 1x1 pixel preview)



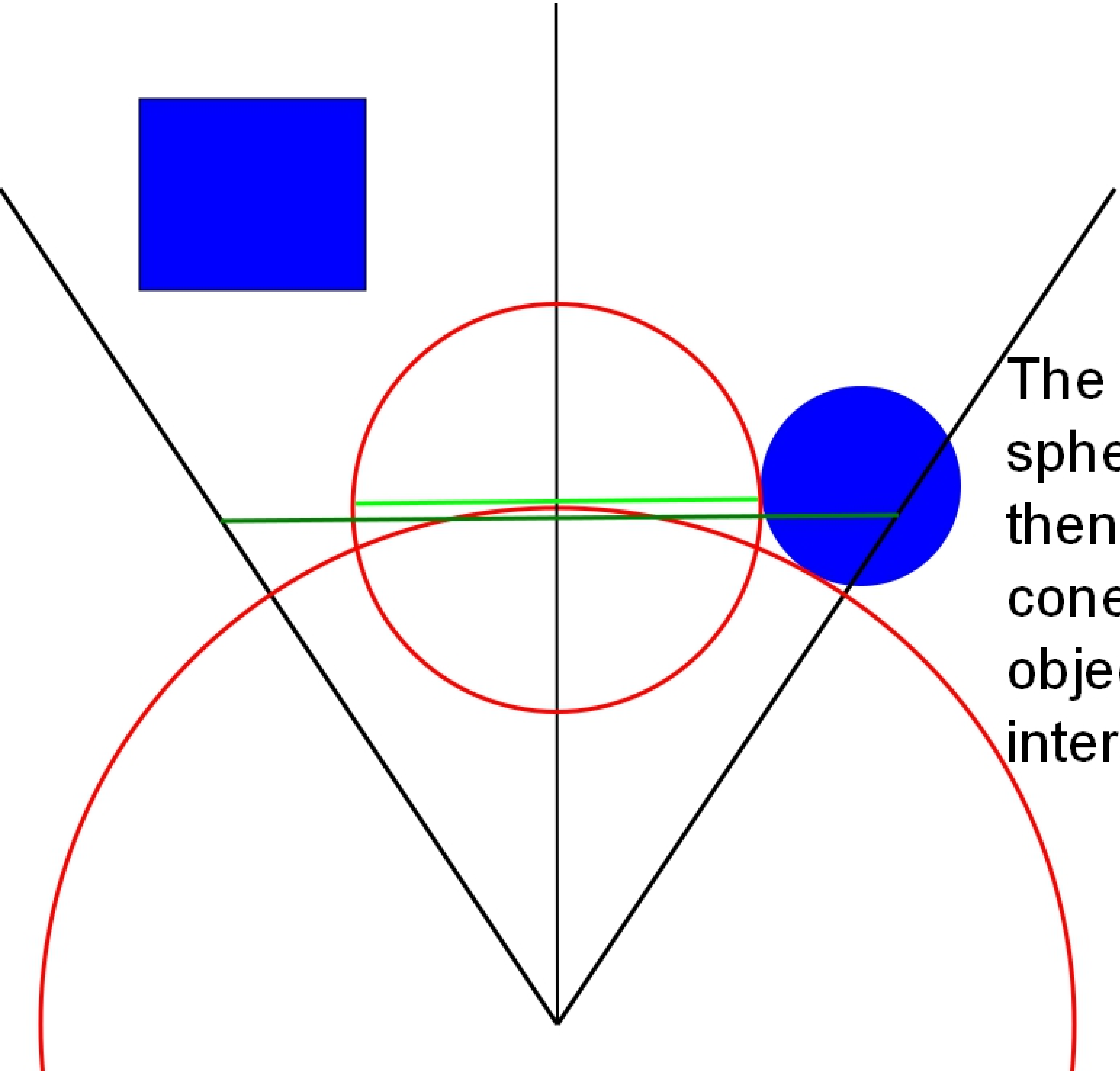
Find first distance



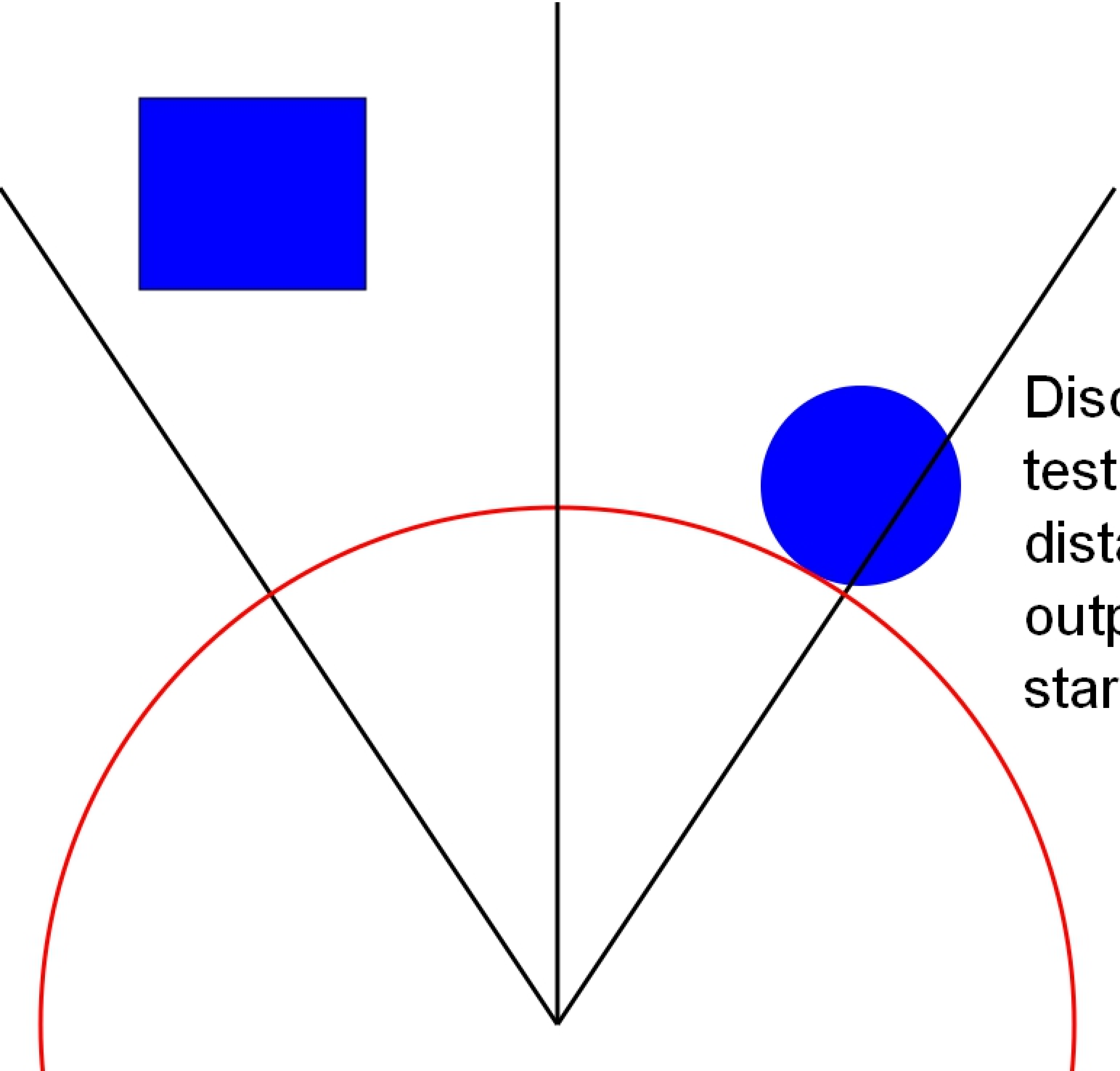
The size of the sphere is much larger then the size of the cone, so continue with this pass.



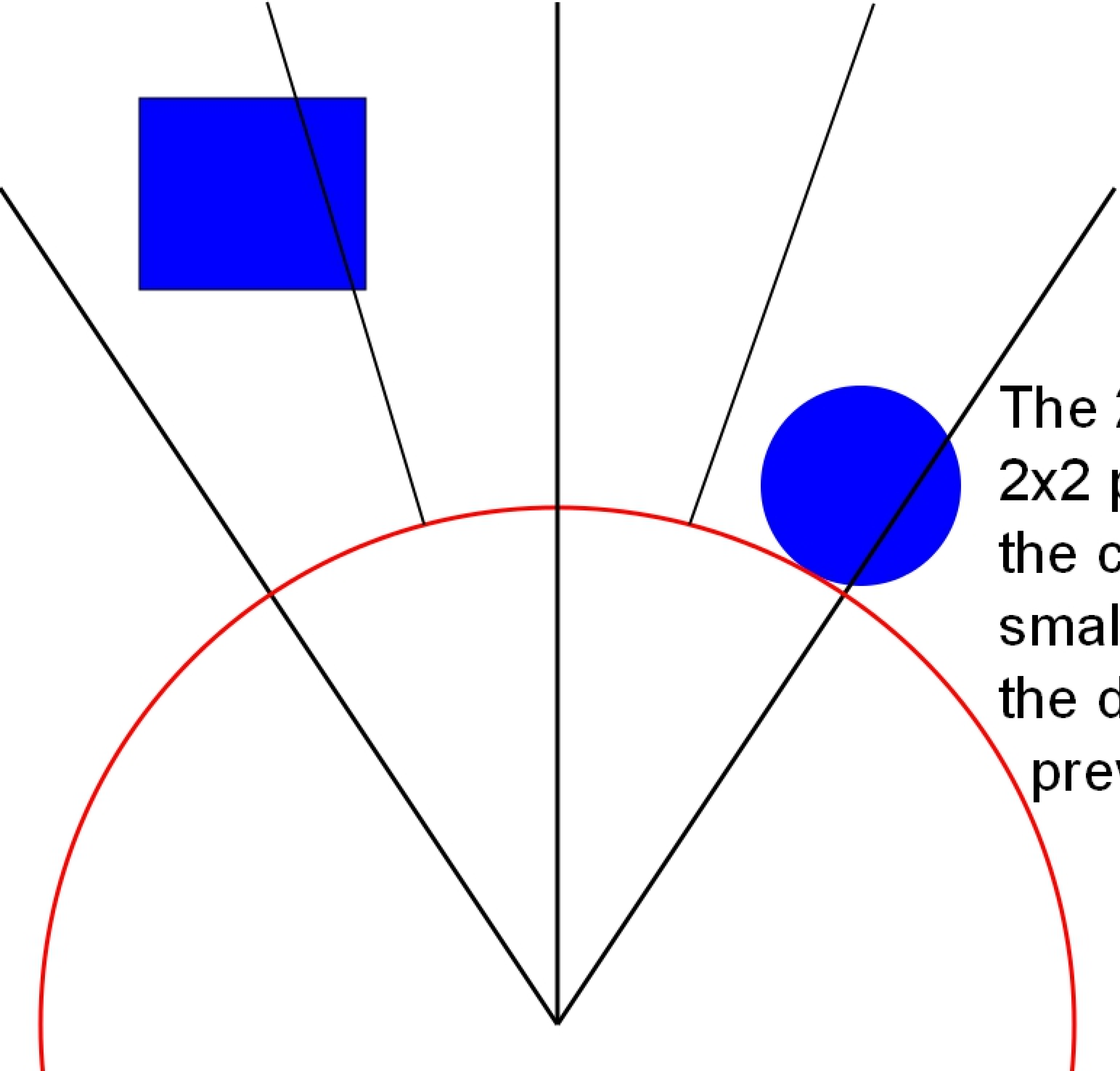
March along the center of the cone and find the next distance



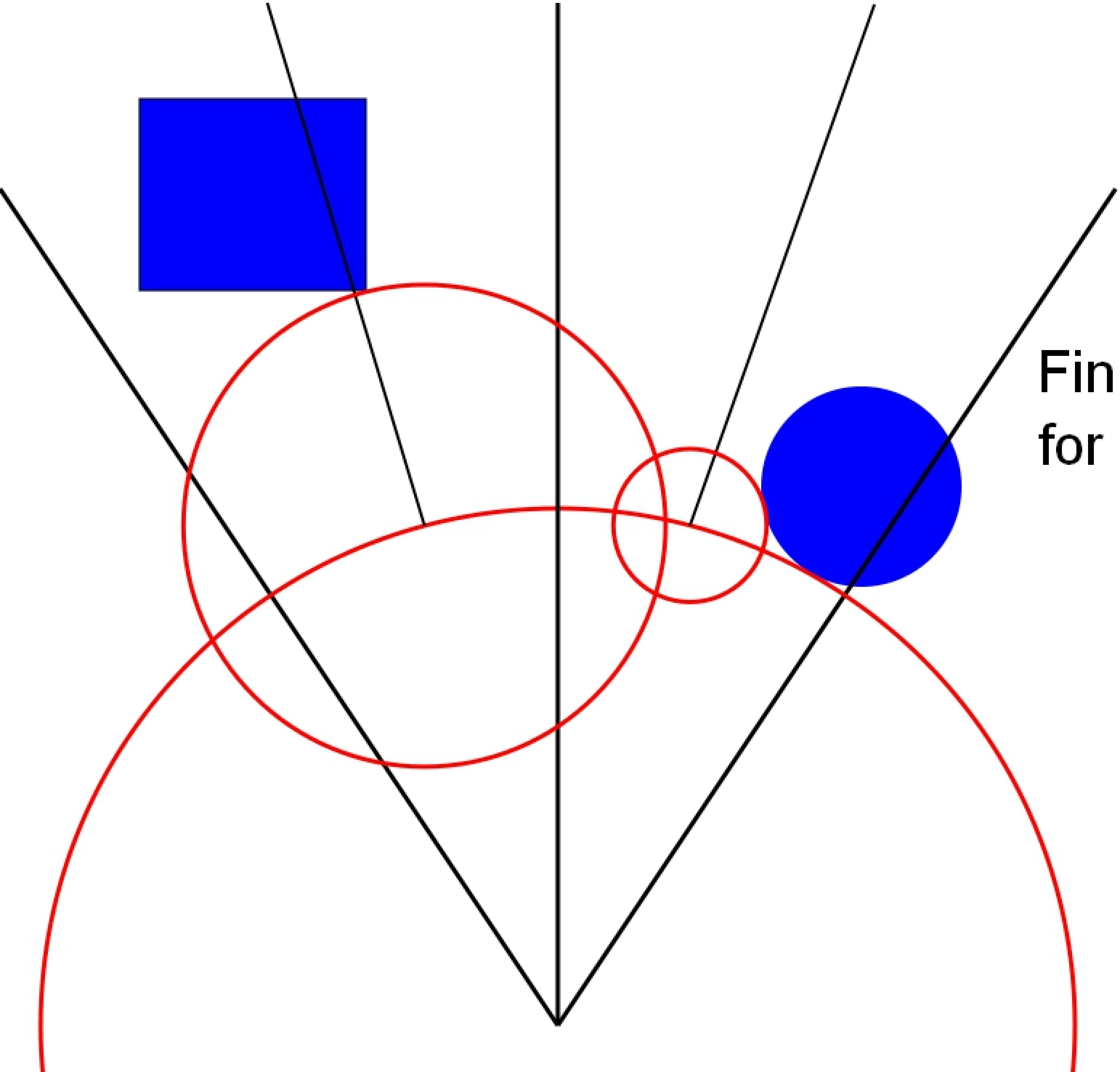
The size of this sphere is smaller then that of the cone, so some object must intersect the cone.



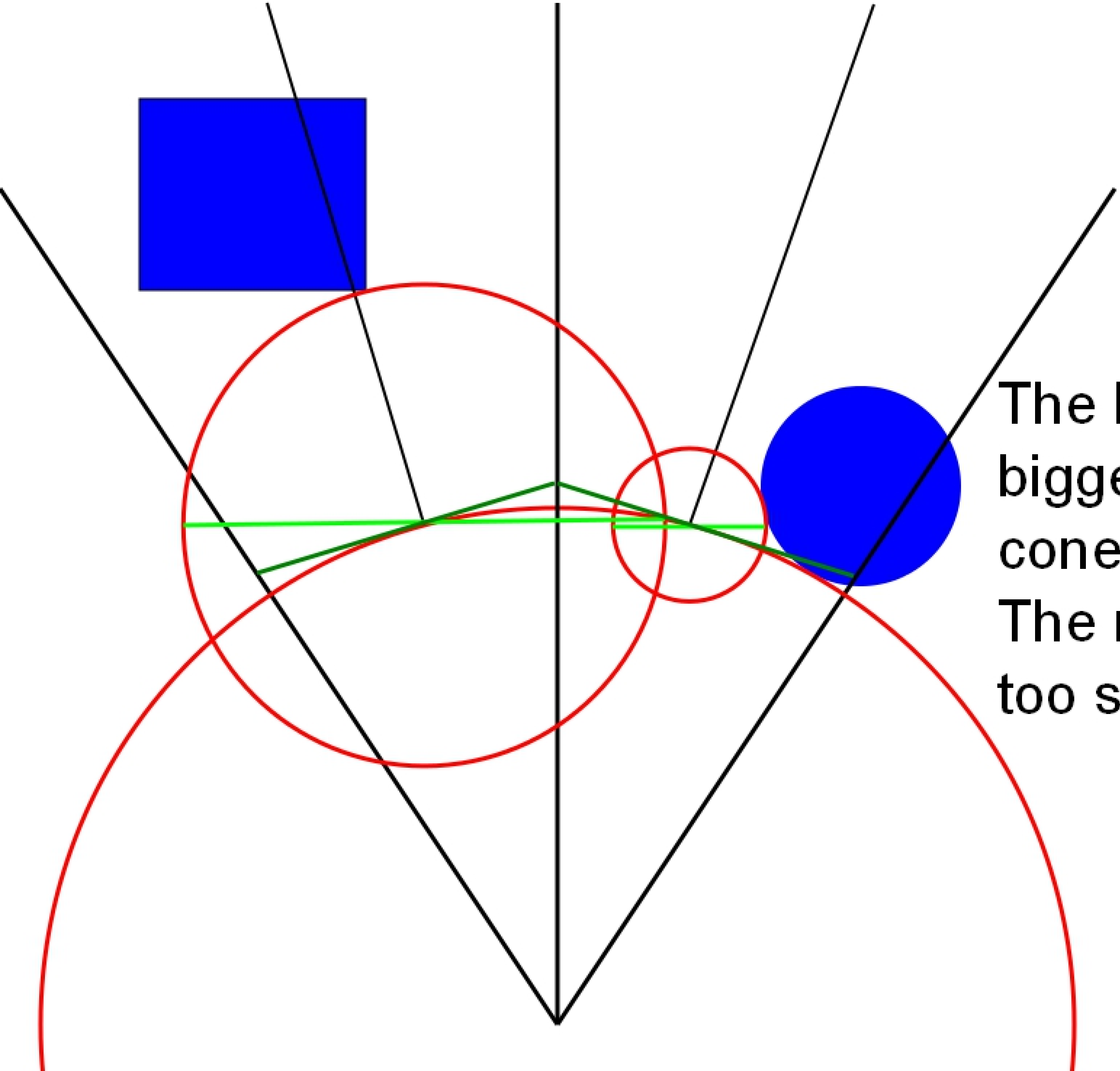
Discard the last test, save the distance in the output buffer and start a new pass.



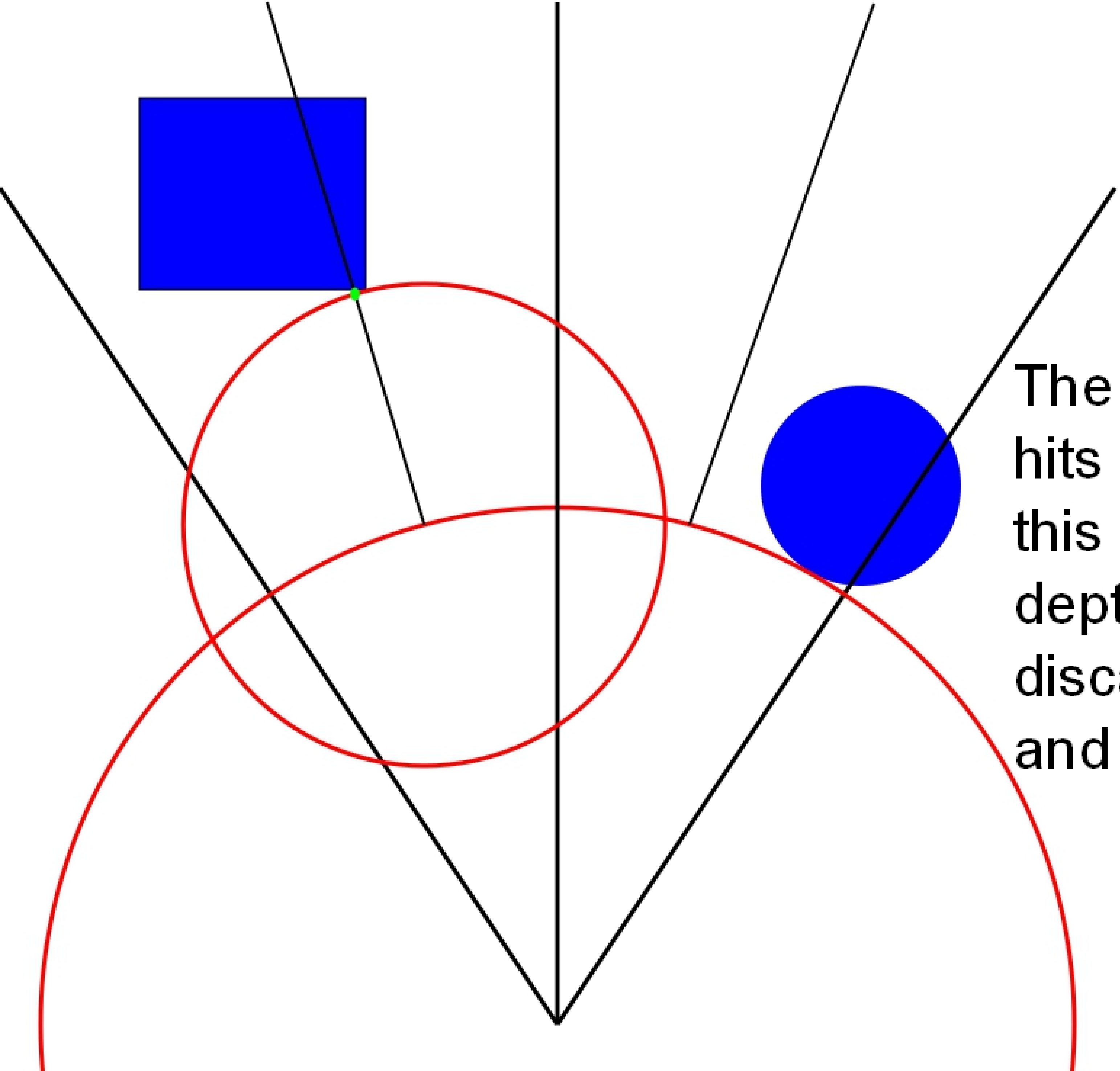
The 2nd pass is 2x2 pixels. Start at the center of each smaller cone. Get the depth from the previous pass.



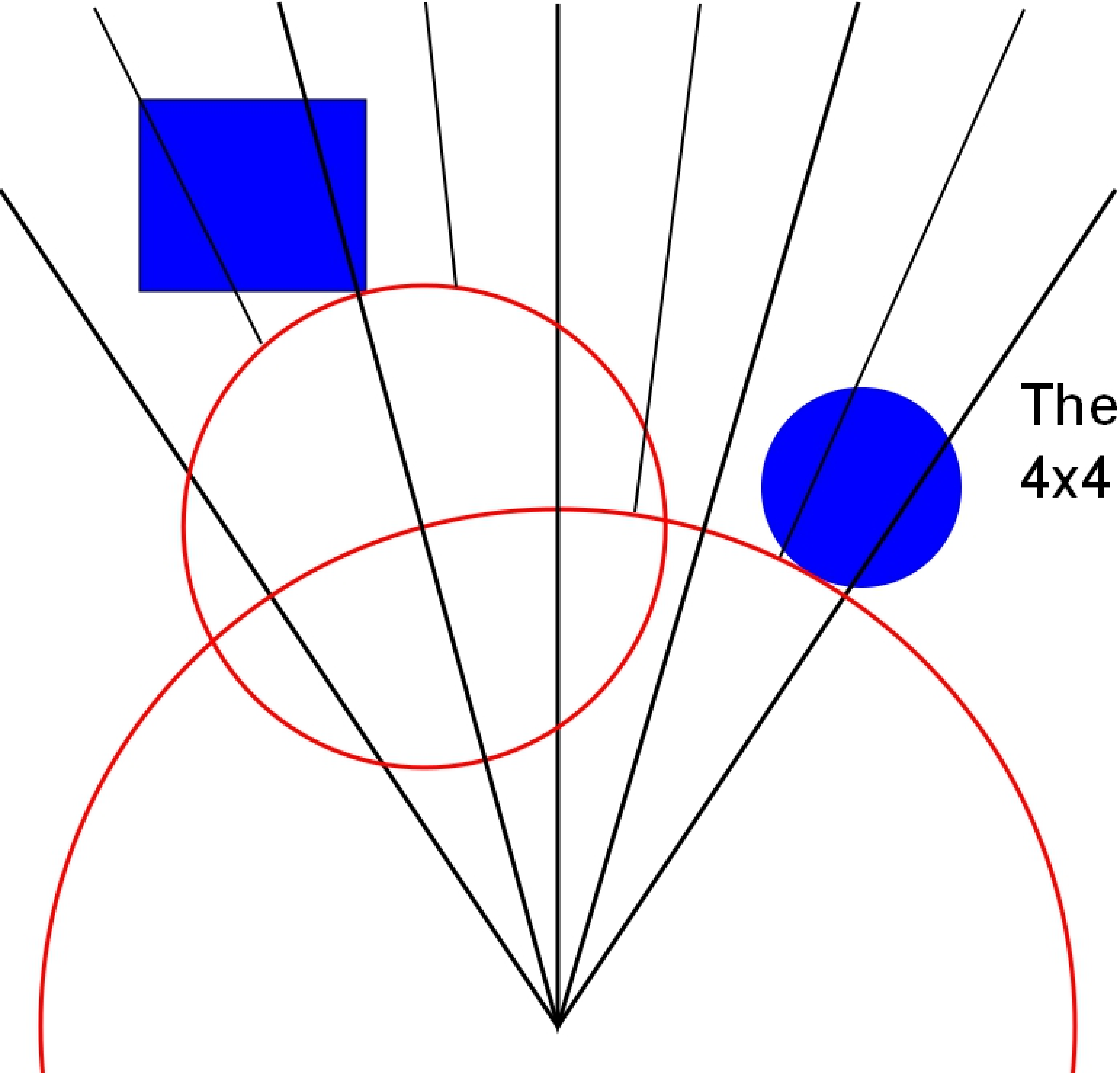
Find the distance
for both cones.



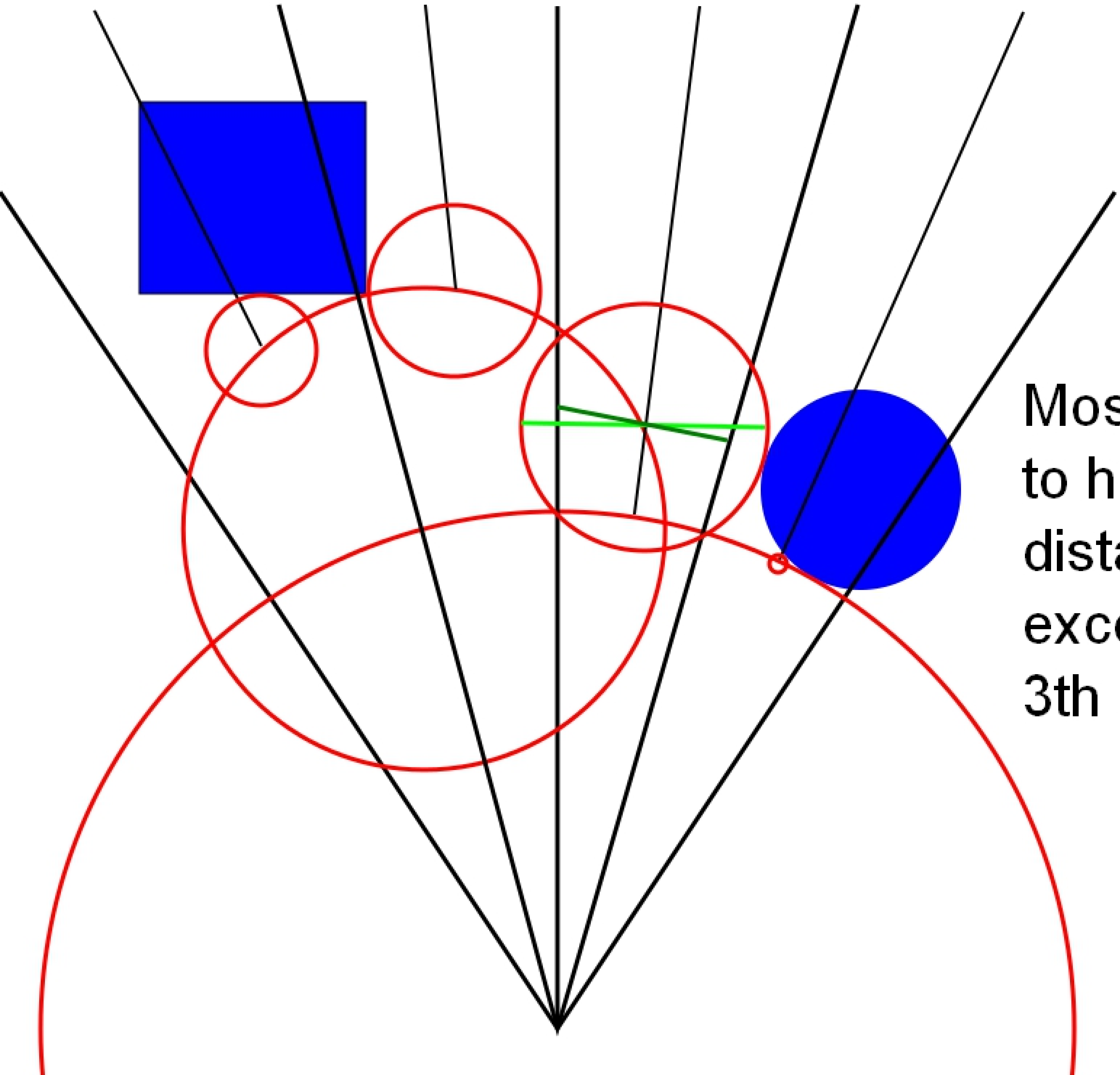
The left sphere is bigger than its cone, so continue. The right sphere is too small, so halt.



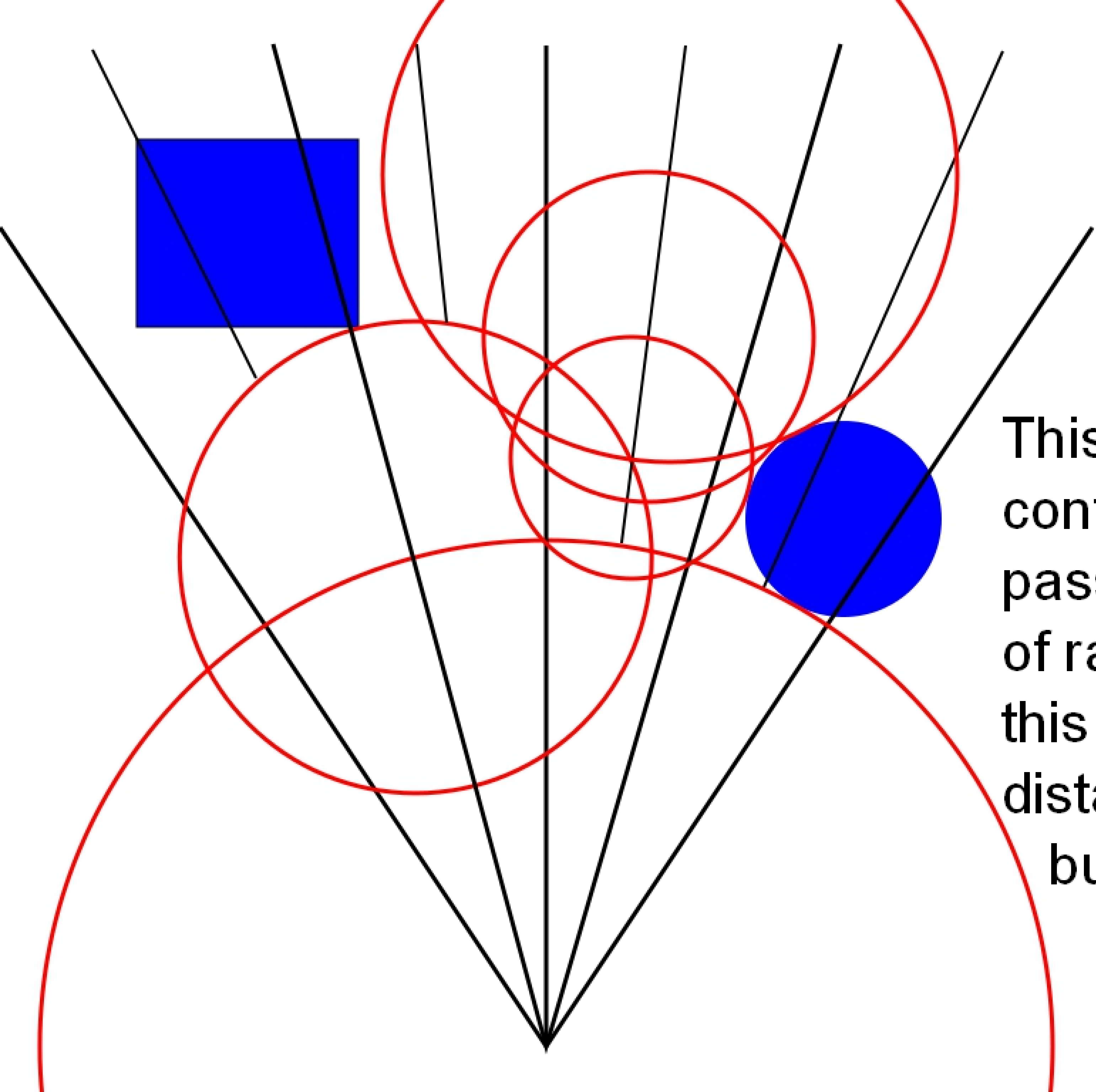
The next left step hits an object, but this is not the final depth pass, so discard the result and halt.



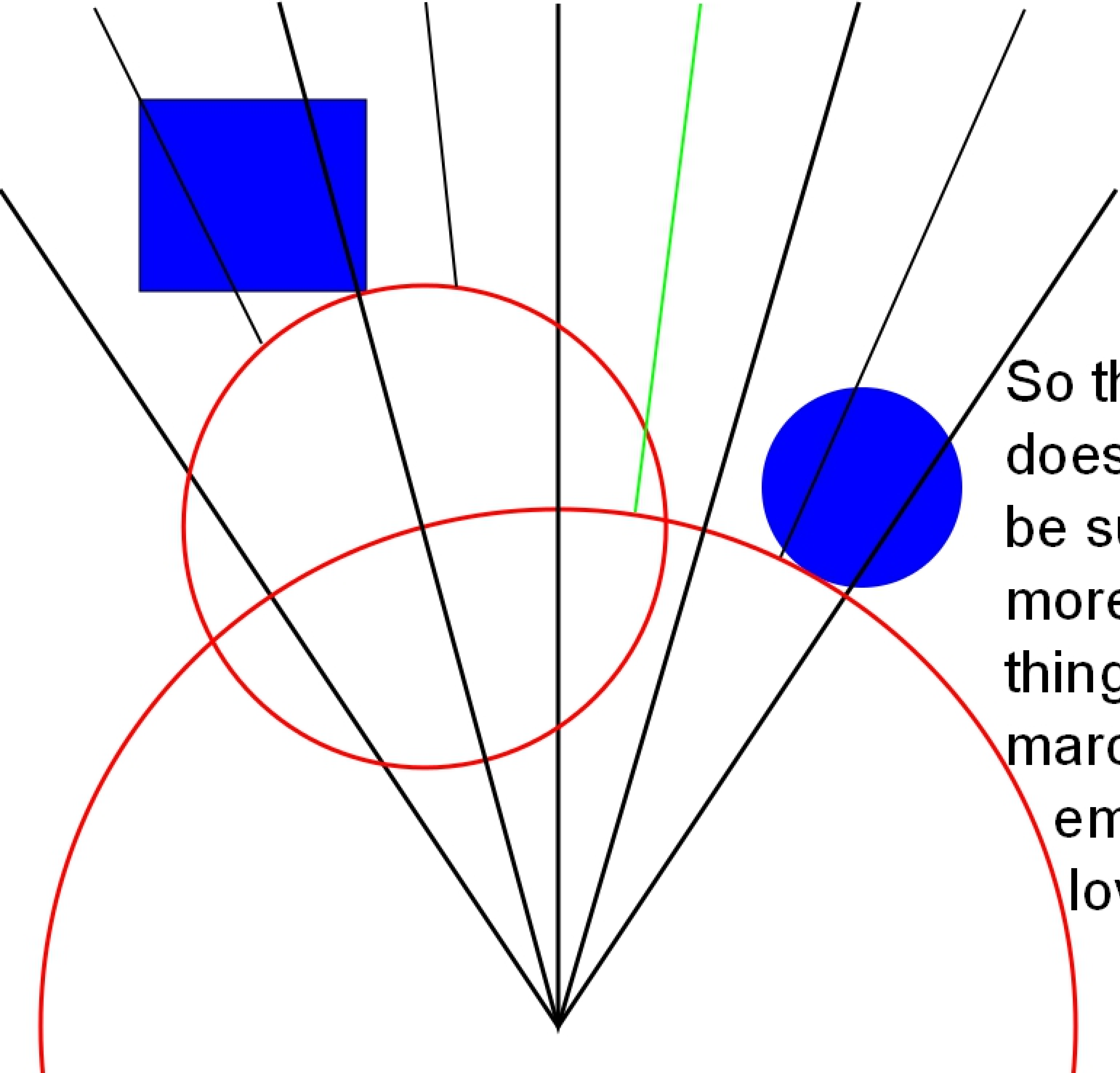
The 3th pass is
4x4 pixels.



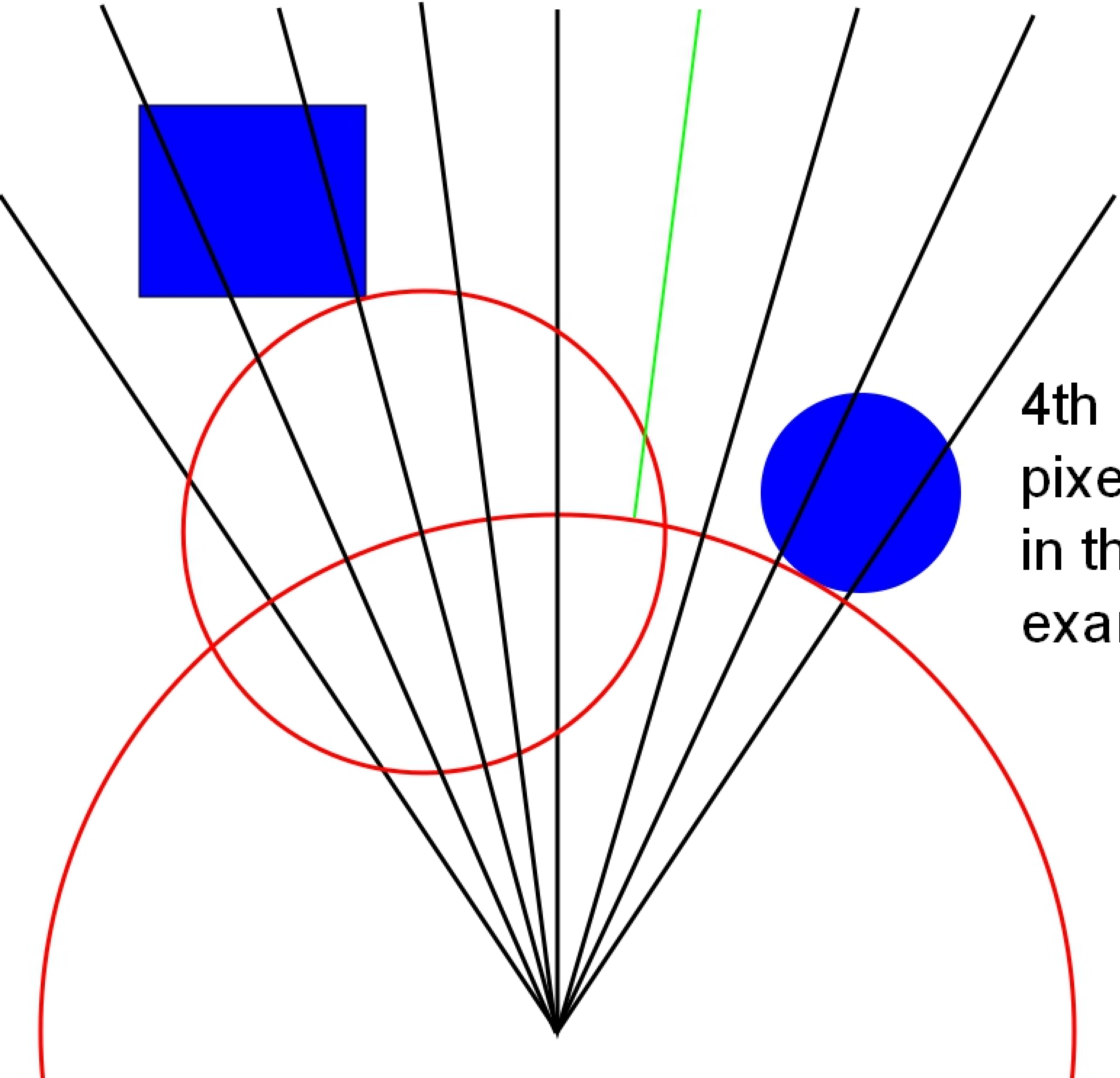
Most cones have to halt at the first distance test, except for the 3th cone.



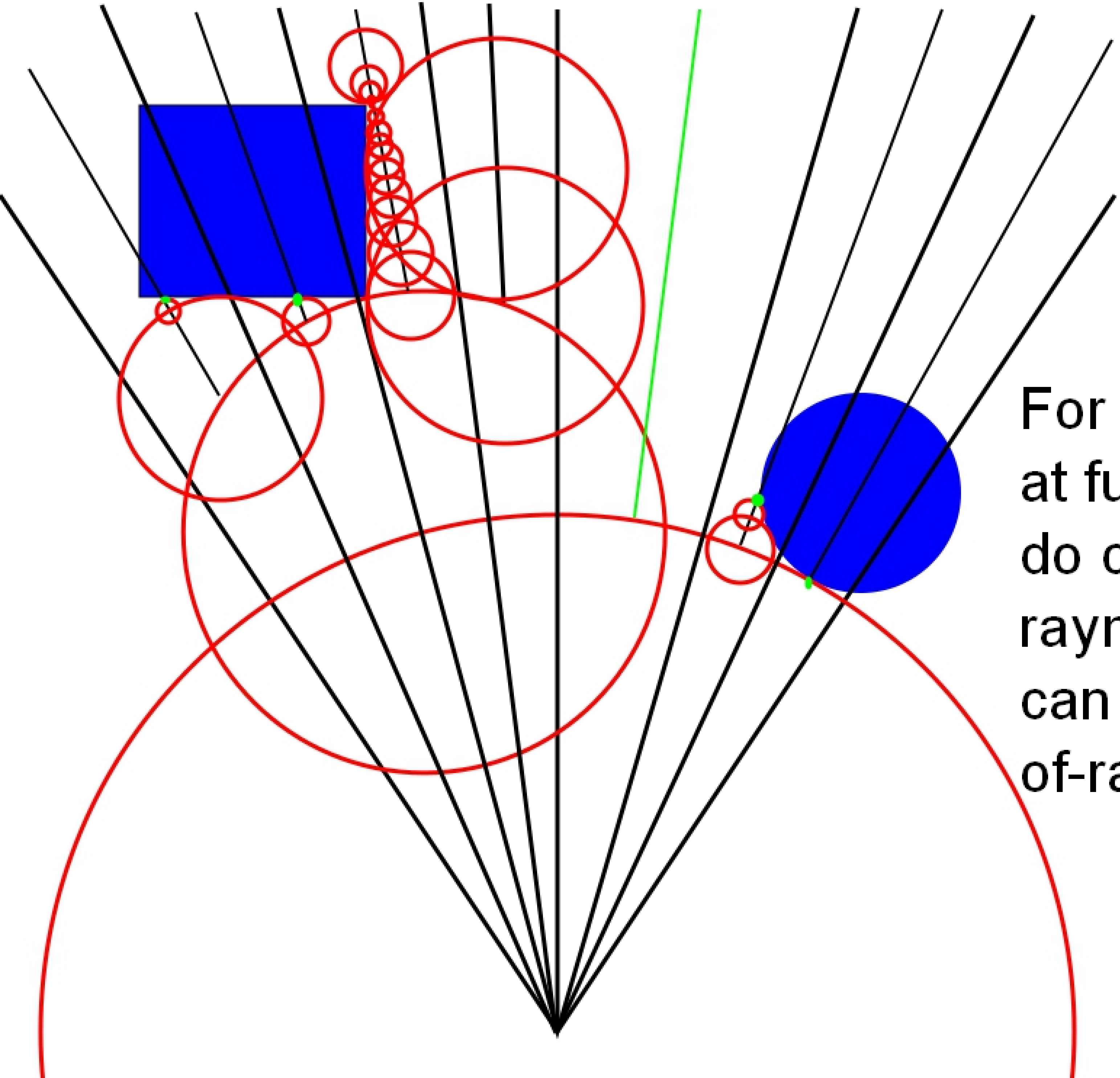
This 3th cone can continue the 3th pass until it is out of range. We put this out-of-range distance in the buffer.



So the 3th cone does not need to be subdivided any more. This speeds things up by marching wide empty spaces at lower resolution.



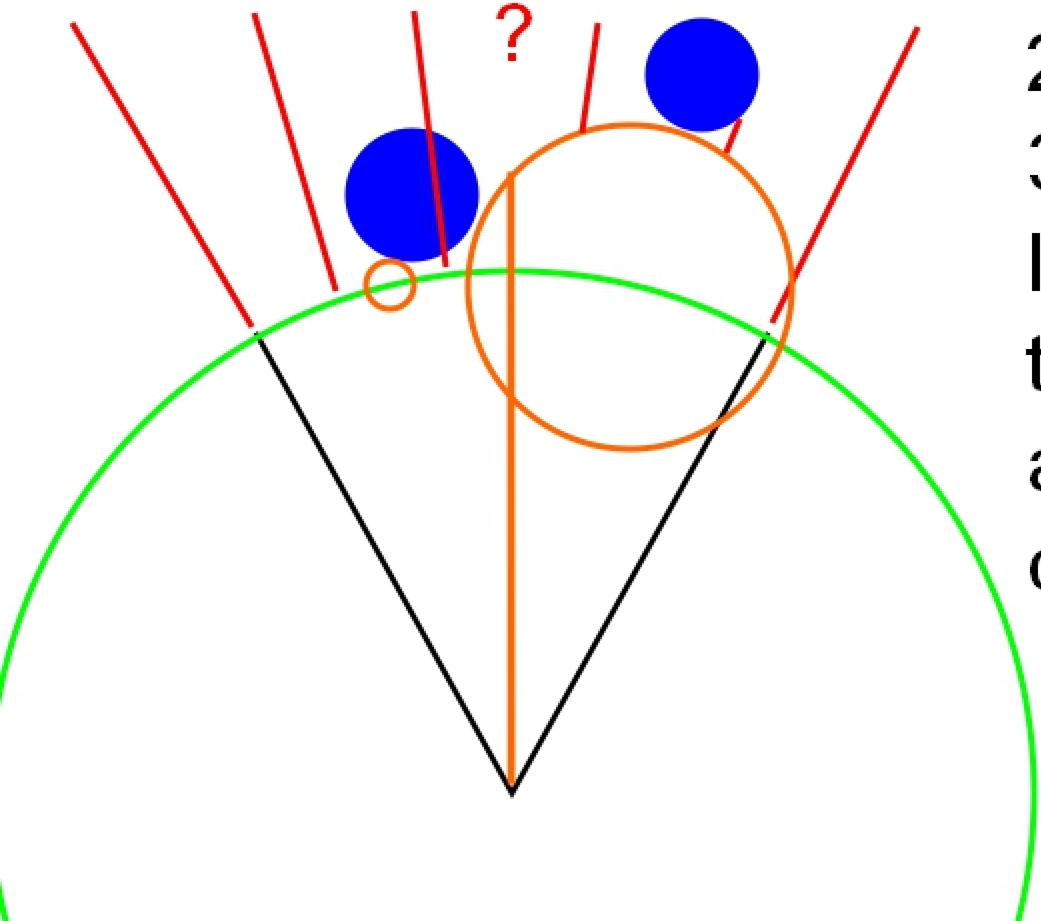
4th pass: 8x8 pixels (final pass in this low-res example)



For the final pass at full resolution, do classic raymarching. You can skip the out-of-range cones.

- After the final depth pass, do the color pass with the color shader, using the buffer as input.
- In real life, the smallest resolutions are not worth the framebuffer/shader switching overhead.
- 4 to 5 depth passes are enough, so starting with your original resolution divided by 16 or 32.
- If your resolution is not nicely divisable by 16 or 32, either limit the cone marching to a smaller area (with a black border around it) or pad your depth framebuffers.

It's very important that each pass doubles the resolution EXACTLY! Otherwise, the cones of a later pass will not be aligned with those of the previous pass, and you will march through the edges of objects



1st pass: 1 pixel

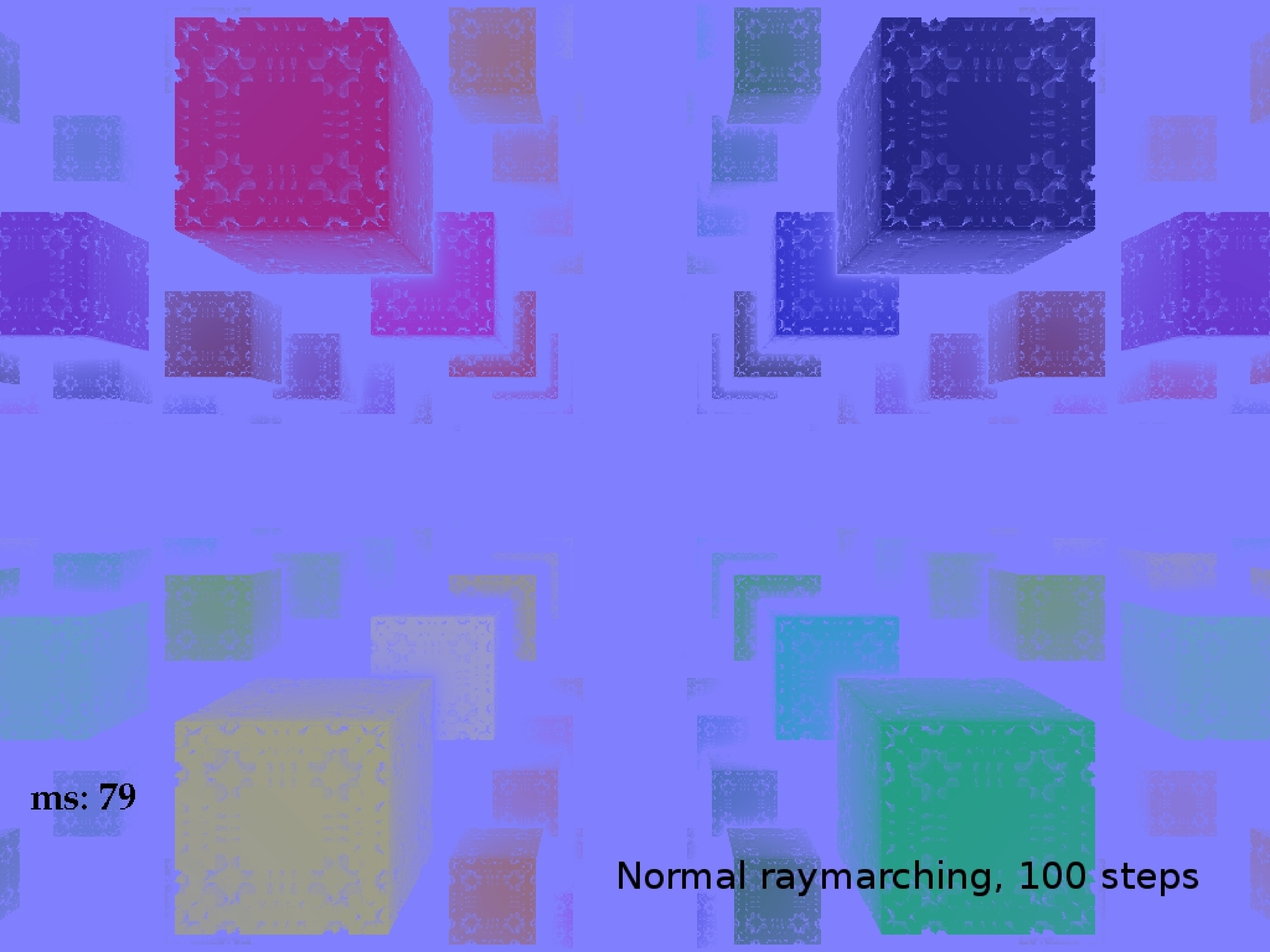
2nd pass: 2 pixels

3th pass: 5 pixels ->BAD!

If the center cone starts from the average depth of the left and right orange cones, or only the right cone, it will skip the left sphere. Later passes will also be wrong.

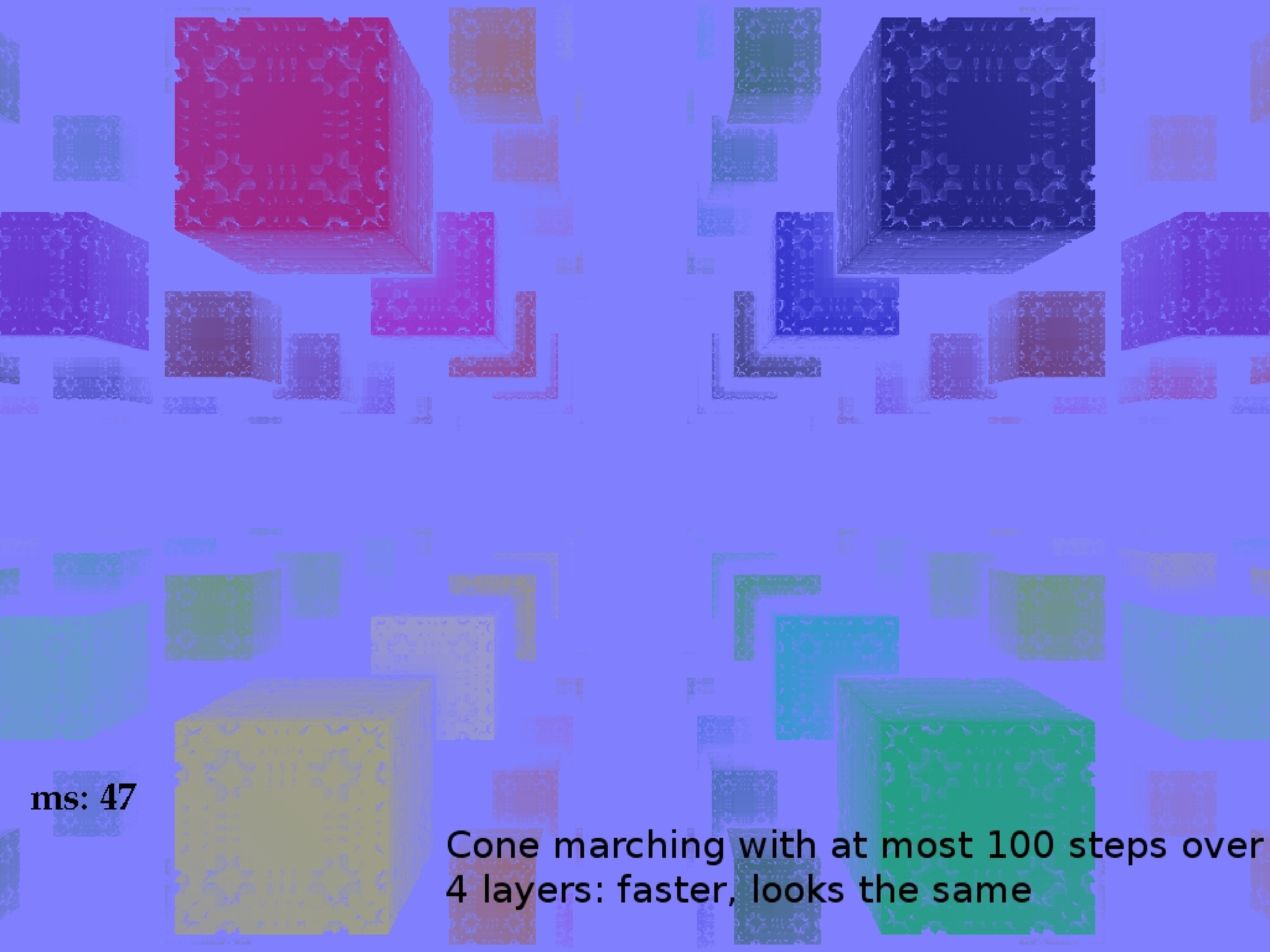
Speedup of finding depth (depends a lot on the scene):

- about 30% if you split the maximum nr of steps over all depth passes. Image looks the same as raymarching.
- About 50% if you give low-resolution passes much more steps. Low-res passes are really cheap, so don't limit yourself there. The image now looks deeper in wide-open parts of the scene.
- About 100% if you give low-res passes many steps, but at the same time lower the amount of steps in the final raymarching depth pass. Compared to classic raymarching, the wide-open parts look deeper, but denser areas lose precision. Depends on what artefacts you find tolerable.

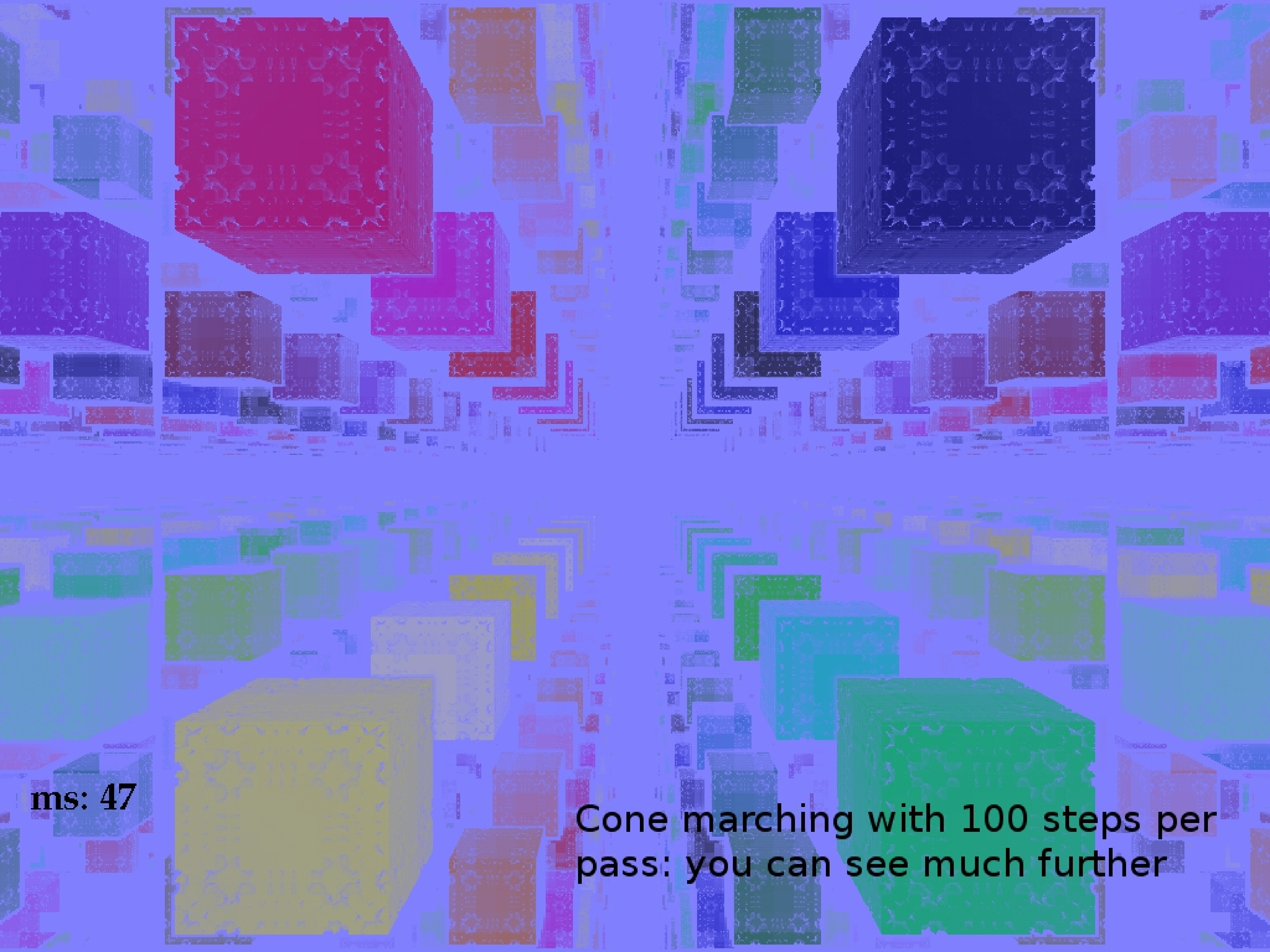


ms: 79

Normal raymarching, 100 steps

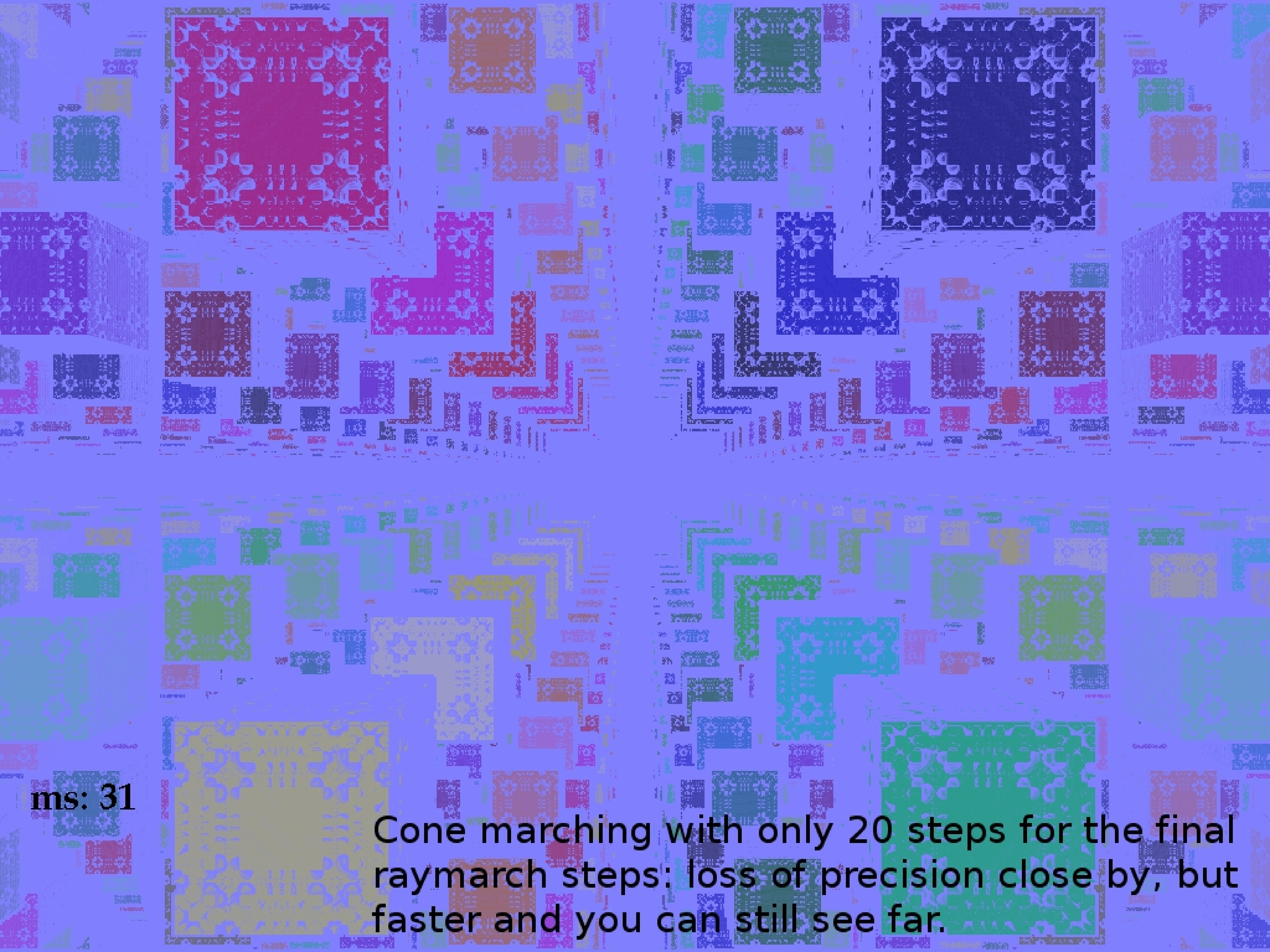


Cone marching with at most 100 steps over 4 layers: faster, looks the same



ms: 47

Cone marching with 100 steps per pass: you can see much further



ms: 31

Cone marching with only 20 steps for the final raymarch steps: loss of precision close by, but faster and you can still see far.

Good points of cone marching:

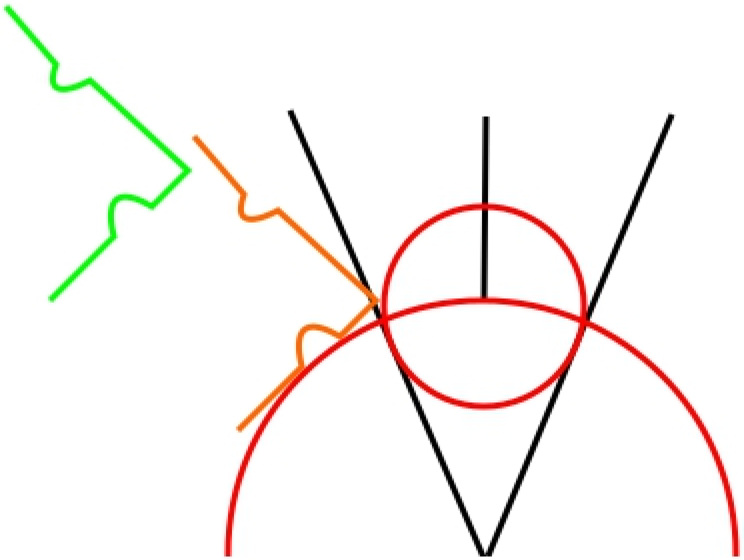
- it works with every distance function.
- no precalculations needed.
- each frame independent, so the distance function is allowed to change (for animation f.e.)
- Small code size: fits in a 4k, even in OpenGL (you need to import the FBO extensions...)

Bad points of cone marching:

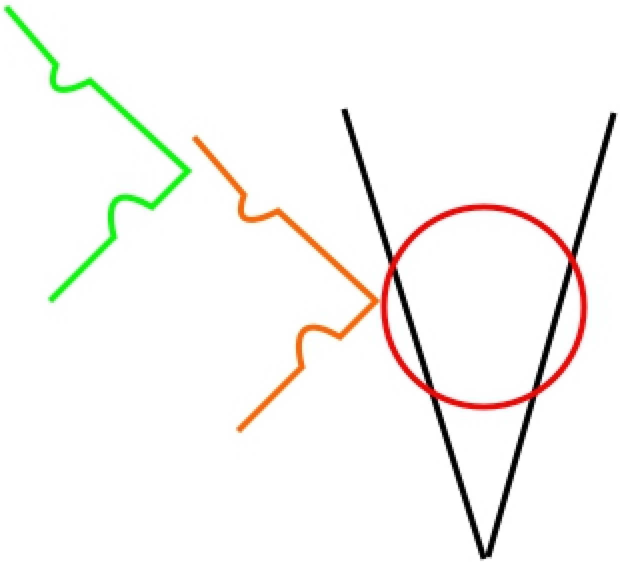
- Only for primary rays (depth). Not for colors (ambient occlusion, reflections, shadows,...)
- The early-out in empty spaces gives visible square artifacts in iteration glow
- Many resolutions are not nicely divisible by powers of 2 (1600*1050, 1366*768), so either pad the FBOs or use a thin black border around scene.

Cone marching and the Mandelbox fractal

- Mandelbox fractal: discovered by Tglad on fractalforums.com . Get the distance formula there, or from Rrrola's Boxplorer (the shader is a readable file)
- The distance formula of the mandelbox is not exact. It's an approximation that errs on the safe side.
- That means that often, when it *seems* the cone hits the mandelbox, it's actually safe to continue.
- We can add a fudge factor to make the cone thinner than it should be.

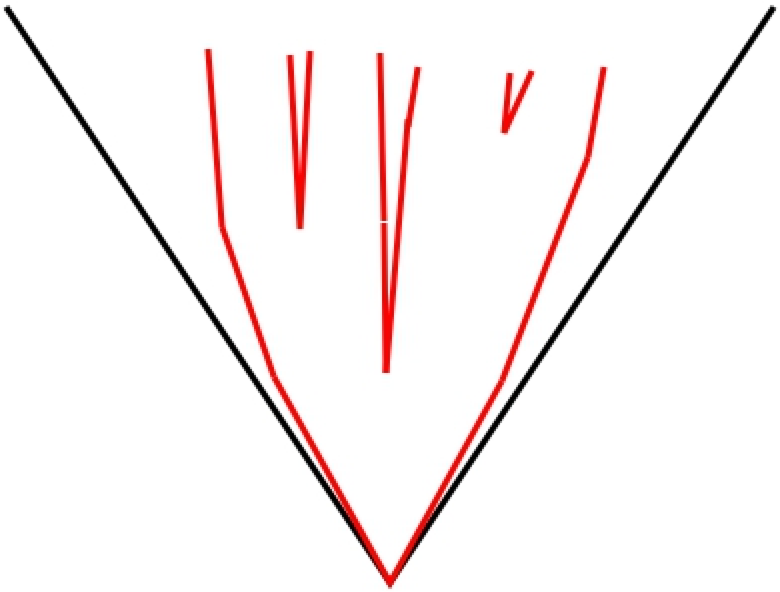


Orange = what the distance function says. In reality, the mandelbox may be anywhere between the orange and green contours. This causes a (probably) unnecessary split.

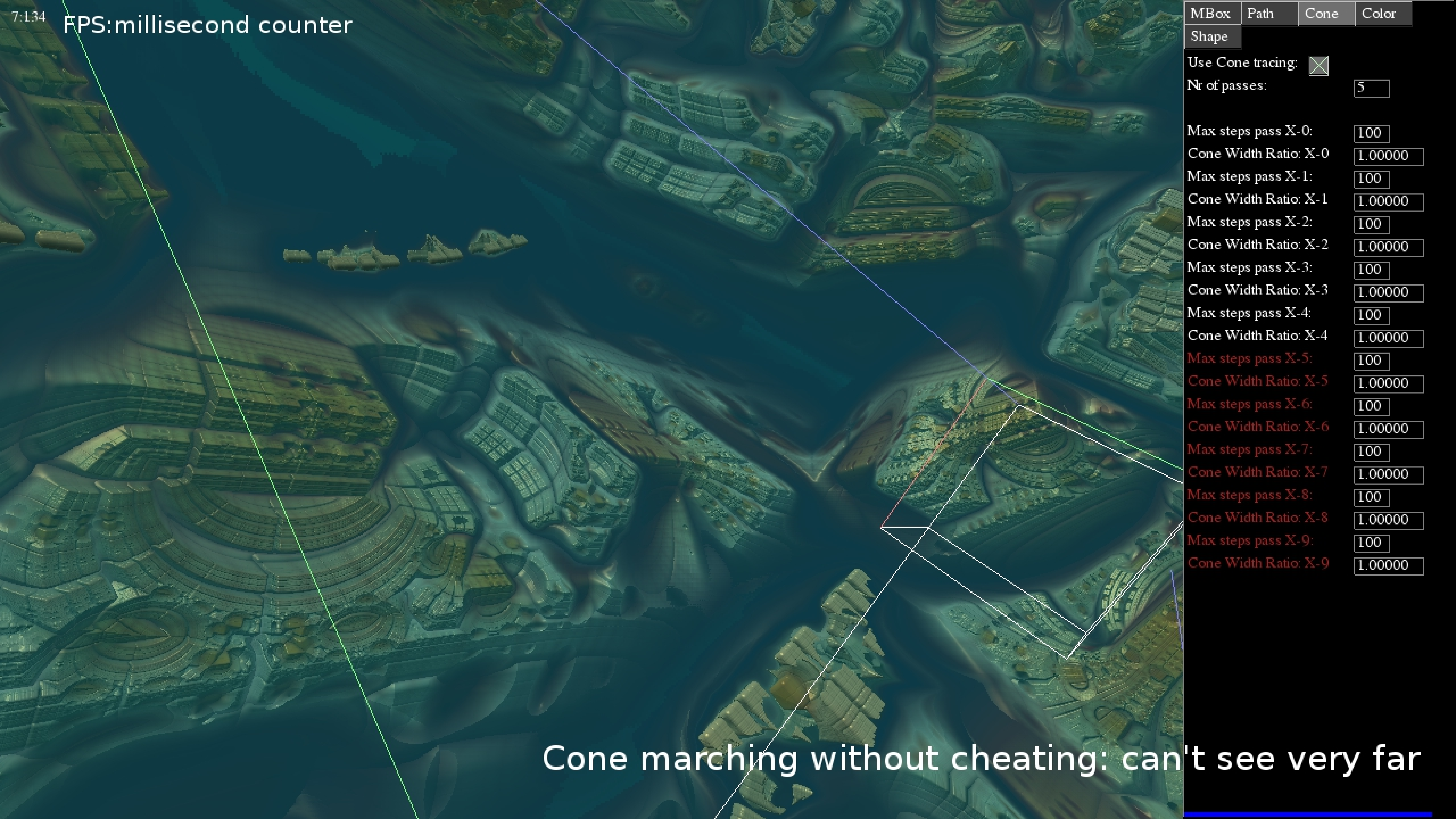


Making the cone thinner avoids some unnecessary splits, so it's faster and can see deeper, but this introduces other artifacts.

- But if the distance function *was* accurate, we punch square holes through edges or thin parts!
- Turns out to be only noticable with lowest passes. (wide cones = large steps & large errors)
- Use different fudge factors for each pass, cheating very little in the first pass and a lot in the last. (makes the cone somewhat bullet-shaped)



- Thin details may fall between the cones, this depends a lot on how "solid" the mandelbox is.

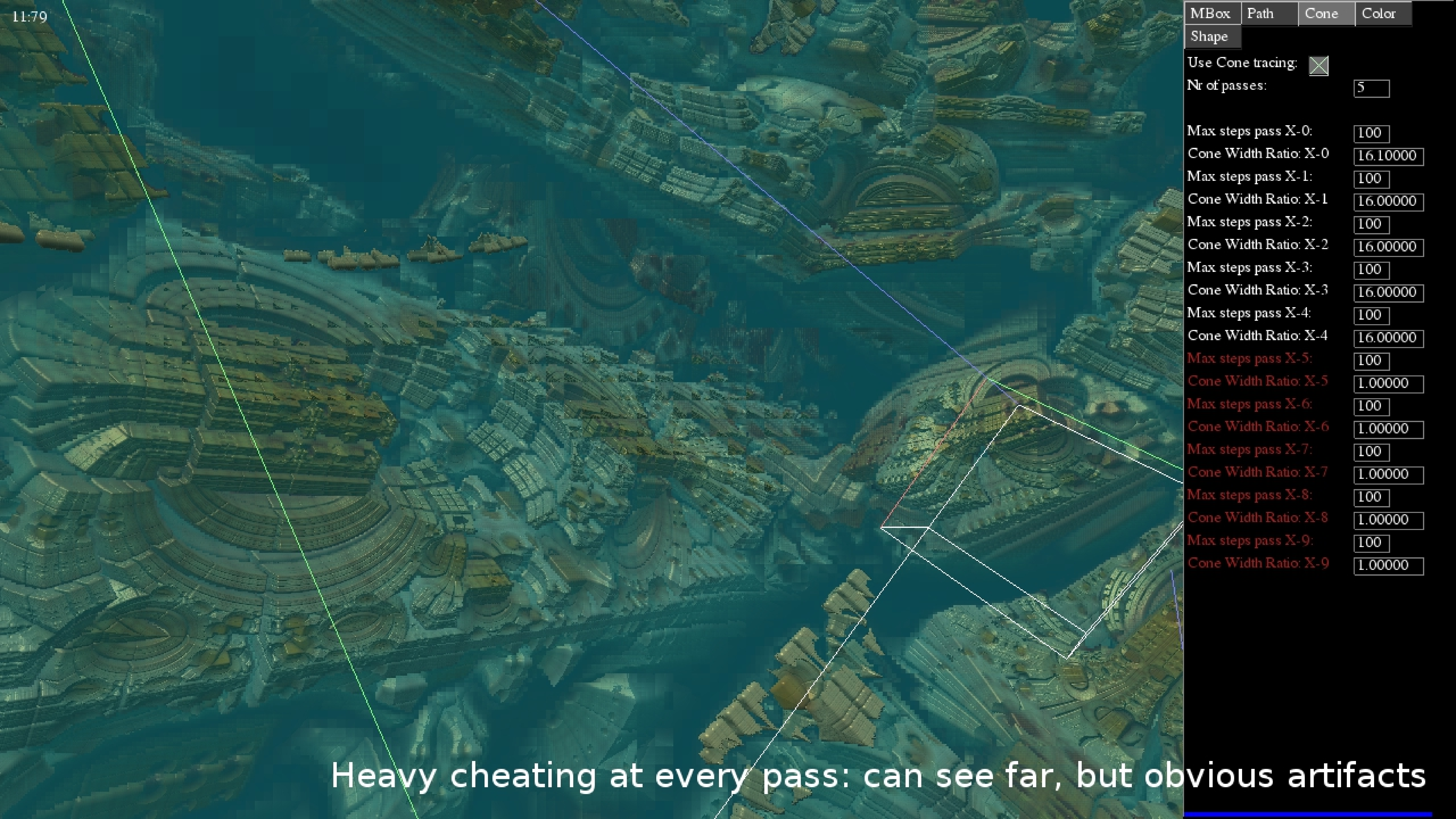


Use Cone tracing: ☒

Nr of passes: 5

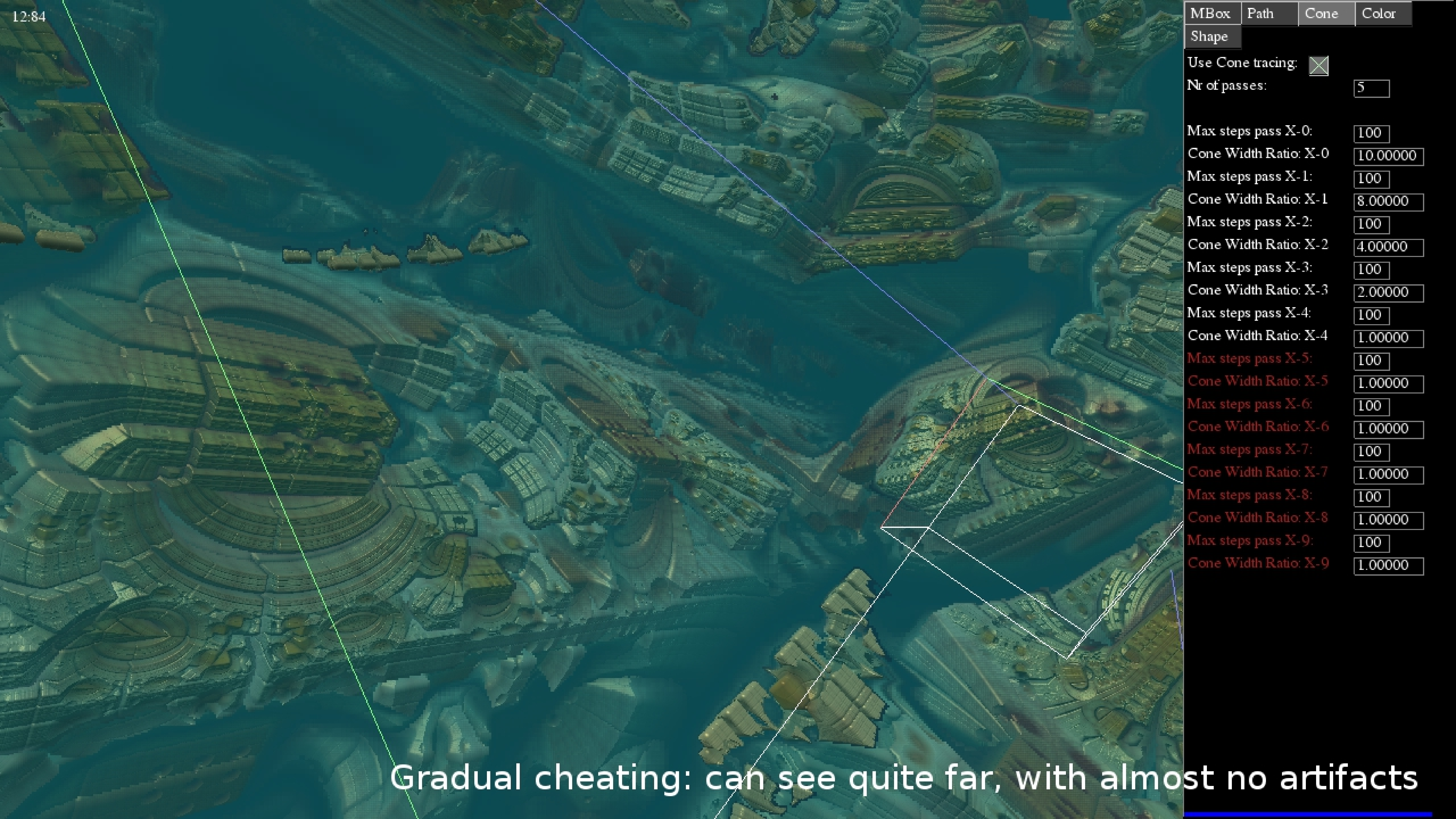
Max steps pass X-0: 100
Cone Width Ratio: X-0 1.00000
Max steps pass X-1: 100
Cone Width Ratio: X-1 1.00000
Max steps pass X-2: 100
Cone Width Ratio: X-2 1.00000
Max steps pass X-3: 100
Cone Width Ratio: X-3 1.00000
Max steps pass X-4: 100
Cone Width Ratio: X-4 1.00000
Max steps pass X-5: 100
Cone Width Ratio: X-5 1.00000
Max steps pass X-6: 100
Cone Width Ratio: X-6 1.00000
Max steps pass X-7: 100
Cone Width Ratio: X-7 1.00000
Max steps pass X-8: 100
Cone Width Ratio: X-8 1.00000
Max steps pass X-9: 100
Cone Width Ratio: X-9 1.00000

Cone marching without cheating: can't see very far



Use Cone tracing:	☒
Nr of passes:	5
Max steps pass X-0:	100
Cone Width Ratio: X-0	16.10000
Max steps pass X-1:	100
Cone Width Ratio: X-1	16.00000
Max steps pass X-2:	100
Cone Width Ratio: X-2	16.00000
Max steps pass X-3:	100
Cone Width Ratio: X-3	16.00000
Max steps pass X-4:	100
Cone Width Ratio: X-4	16.00000
Max steps pass X-5:	100
Cone Width Ratio: X-5	1.00000
Max steps pass X-6:	100
Cone Width Ratio: X-6	1.00000
Max steps pass X-7:	100
Cone Width Ratio: X-7	1.00000
Max steps pass X-8:	100
Cone Width Ratio: X-8	1.00000
Max steps pass X-9:	100
Cone Width Ratio: X-9	1.00000

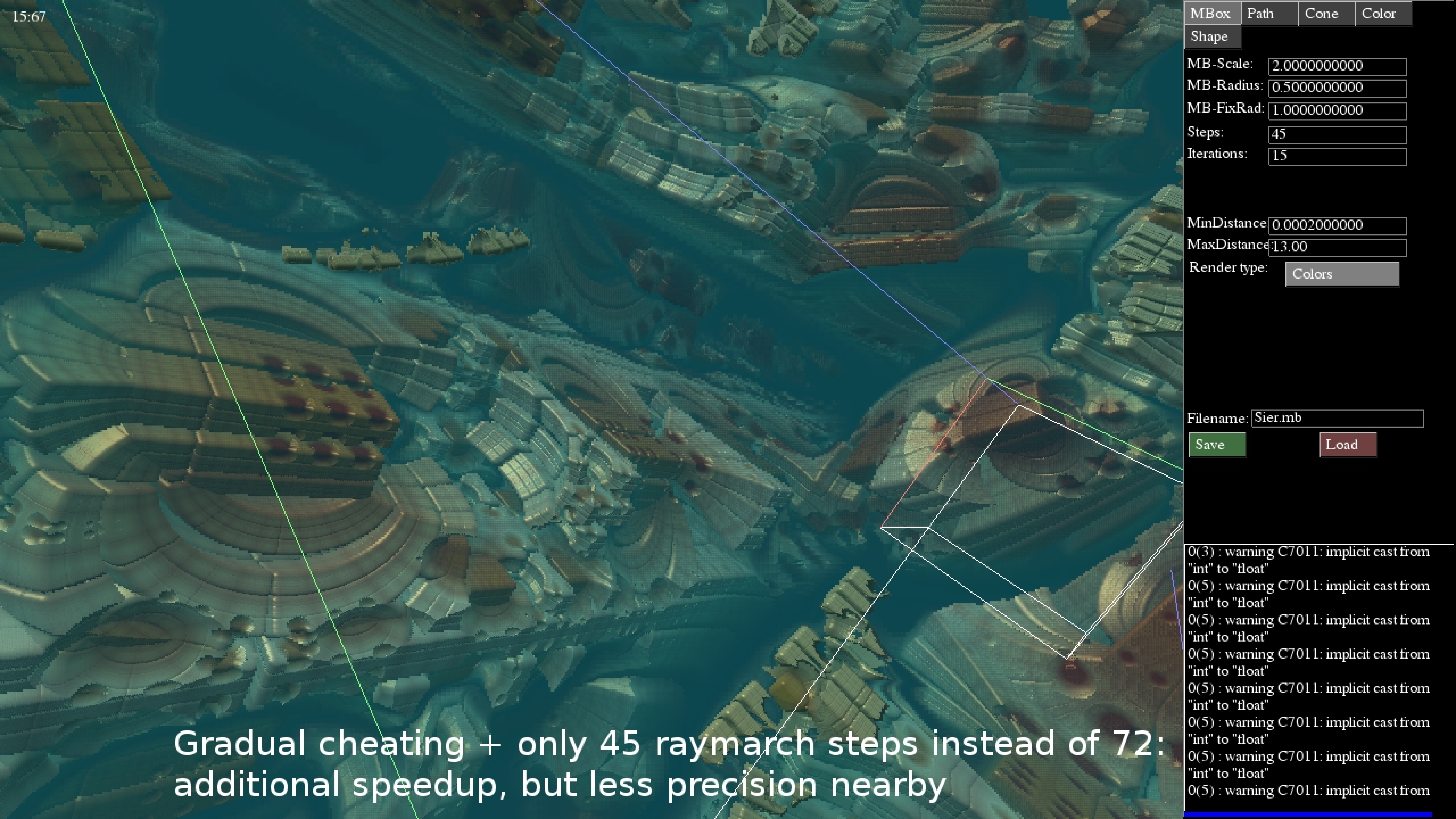
Heavy cheating at every pass: can see far, but obvious artifacts



Use Cone tracing: ☒
Nr of passes: 5

Max steps pass X-0: 100
Cone Width Ratio: X-0 10.00000
Max steps pass X-1: 100
Cone Width Ratio: X-1 8.00000
Max steps pass X-2: 100
Cone Width Ratio: X-2 4.00000
Max steps pass X-3: 100
Cone Width Ratio: X-3 2.00000
Max steps pass X-4: 100
Cone Width Ratio: X-4 1.00000
Max steps pass X-5: 100
Cone Width Ratio: X-5 1.00000
Max steps pass X-6: 100
Cone Width Ratio: X-6 1.00000
Max steps pass X-7: 100
Cone Width Ratio: X-7 1.00000
Max steps pass X-8: 100
Cone Width Ratio: X-8 1.00000
Max steps pass X-9: 100
Cone Width Ratio: X-9 1.00000

Gradual cheating: can see quite far, with almost no artifacts



MB-Scale: 2.0000000000

MB-Radius: 0.5000000000

MB-FixRad: 1.0000000000

Steps: 45

Iterations: 15

MinDistance 0.0002000000

MaxDistance 13.00

Render type: Colors

Filename: Sier.mb

```
0(3) : warning C7011: implicit cast from
"int" to "float"
0(5) : warning C7011: implicit cast from
"int" to "float"
0(5) : warning C7011: implicit cast from
"int" to "float"
0(5) : warning C7011: implicit cast from
"int" to "float"
0(5) : warning C7011: implicit cast from
"int" to "float"
0(5) : warning C7011: implicit cast from
"int" to "float"
0(5) : warning C7011: implicit cast from
"int" to "float"
0(5) : warning C7011: implicit cast from
```

Gradual cheating + only 45 raymarch steps instead of 72:
additional speedup, but less precision nearby

- So cone marching allows you to make tradeoffs between speed, seeing far and render artifacts.
- Thanks to the Revision organisers for allowing me to present a shorter version of this presentation at the Revision 2012 lightning talks.

Bonus: 4K coders, avoid the Windows 7 busy cursor with PeekMessage(0, 0, 0, 0, PM_REMOVE)