Sparse Voxel Octree

Tristano Munini

June 27, 2021

Topics and Objectives

The topics we will talk about are:

- Ray Tracing
- Signed Distance Functions
- Constructive Solid Geometry
- Sparse Voxel Octree (SVO)
- SVO Traversal Algorithm
- Implementation and Results

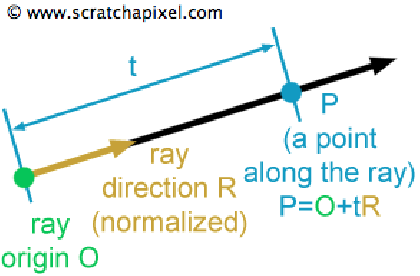
Objectives:

- use an SVO data structure to speed up the rendering process of my `C++` implicit surface renderer
- get a better understanding of 3D spatial data structures

Ray Tracing: as simple as possible

- Can be seen as a root finding algorithm between rays and objects' descriptions
- Comprehends always a camera and a scene
- The rendering process generates an image out of a (3D) scene from the camera's perspective
- At each pixel corresponds at least one ray (usally several)
- The rays can bounce on surfaces, split in different rays and collect various information on their path
- Properly done Ray Tracing is extremely complex and computationally heavy

Ray Tracing: as simple as possible

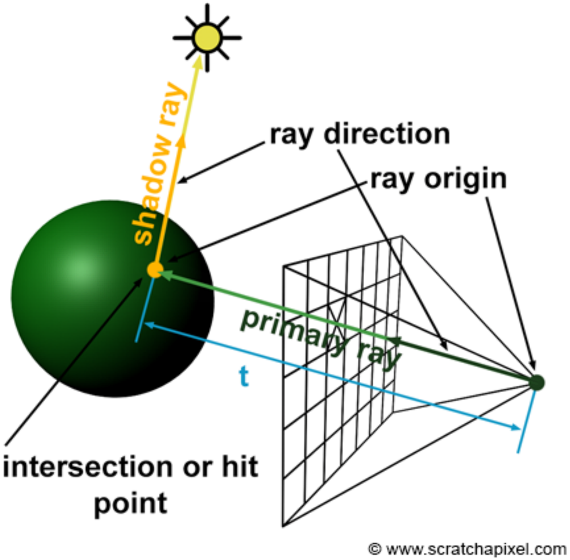


In formulas:

$$r(t) = o + t * d$$

where t is a timestep, o is the ray's origin, d is the ray's direction, and $r(t)$ is a point along the ray.

Ray Tracing: as simple as possible



Signed Distance Functions

A distance surface is implicitly defined by a function $f : \mathbb{R}^3 \to \mathbb{R}$ that characterizes $A \subset \mathbb{R}^3$, set of points that are on or inside the implicit surface:

$$A = \{x : f(x) \leq 0\}$$

The surface can be also defined with $f^{-1}(0)$, which gives exactly the points on the surface.

Signed Distance Functions

The surface can be defined from the outside using a *point-to-set* distance:

$$d(x, A) = min_{y \in A} ||x - y||$$

Thus $d(x, A)$, given a point $x \in \mathbb{R}^3$, returns the shortest distance to the surface defined by A.

Definition (Signed Distance Function)

We say that f is a signed distance function (SDF) when holds

$$|f(x)| = d(x, f^{-1}(0)) \tag{1}$$

SDF Examples

Given a point $P = (x, y, z)$

- Sphere

$$\sqrt{x^2 + y^2 + z^2} - r$$

- Torus

$$\sqrt{\left(\sqrt{x^2 + z^2} - r0\right)^2 + y^2} - r1$$

- Cube

$$\sqrt{max(|x| - l, 0)^2 + max(|y| - l, 0)^2 + max(|z| - l, 0)^2}$$

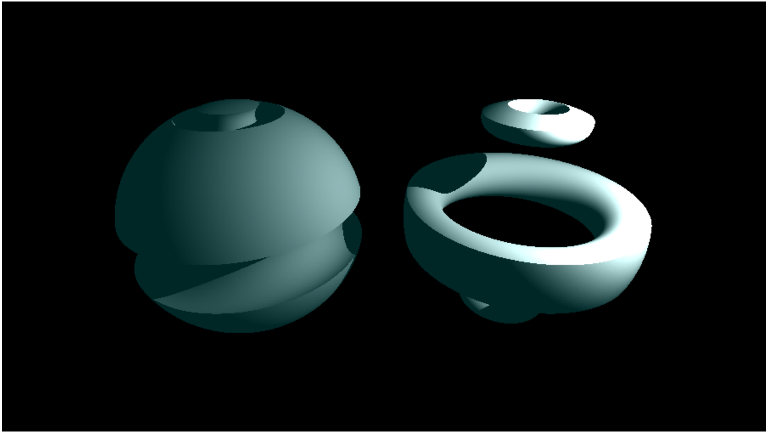
- Look the bibliography for more

Constructive Solid Geometry

SDFs make easy to create complex shapes from few simple primitives. This technique it known as Constructive Solid Geometry (CSG).

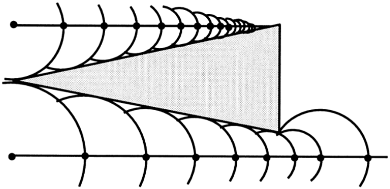
- union $min(f_1, f_2)$
- intersection $max(f_1, f_2)$
- subtraction $max(f_1, -f_2)$
- mixing $k * f_1 + (1 - k) * f_2$ with $k \in [0, 1]$

CSG Example



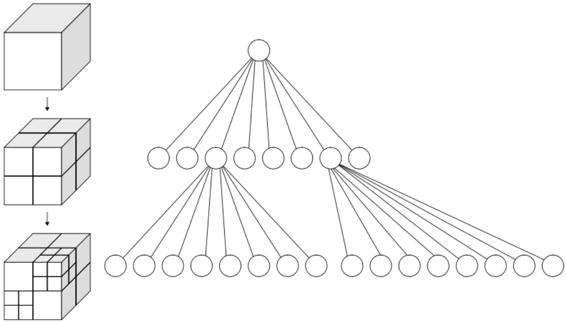
Sphere Tracing

- Studying ray's path proceeding with a fixed step may lead to errors, with Sphere Tracing an adaptive step is used
- The step size is given by an SDF
- Given $x \in \mathbb{R}^3$ and a surface S, $d(x, S)$ means we can move x in every direction by $d(x, S)$ being sure at worst to just hit the surface
- In other words we are drawing a safe sphere around the point, in which there is no surface



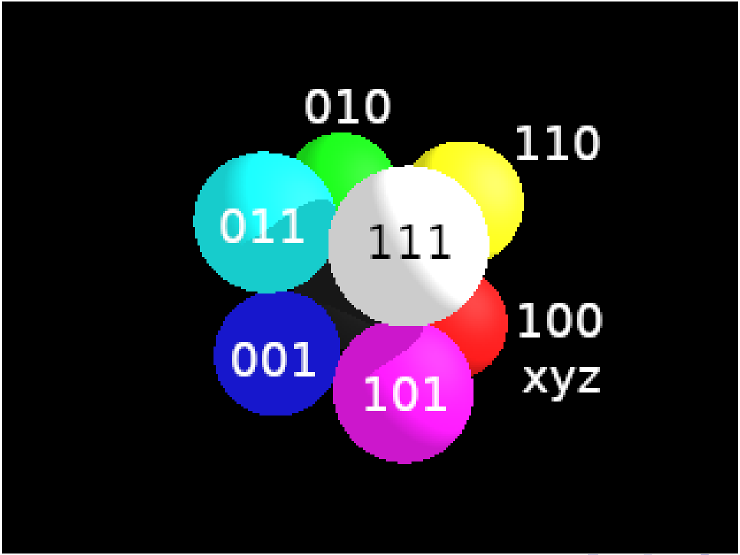
Octree

An Octree is a space partitioning data structure in which each node has exactly 8 children



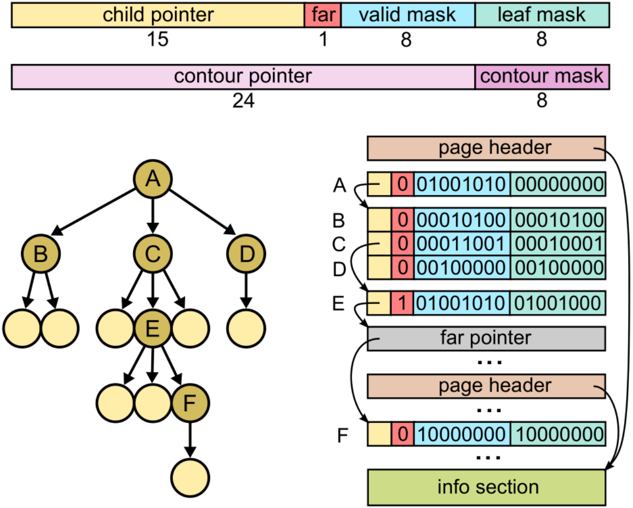
Child Encoding

In a right-handed coordinate system the child numbering respects the quadrants' sing as if the cube was centered in the origin



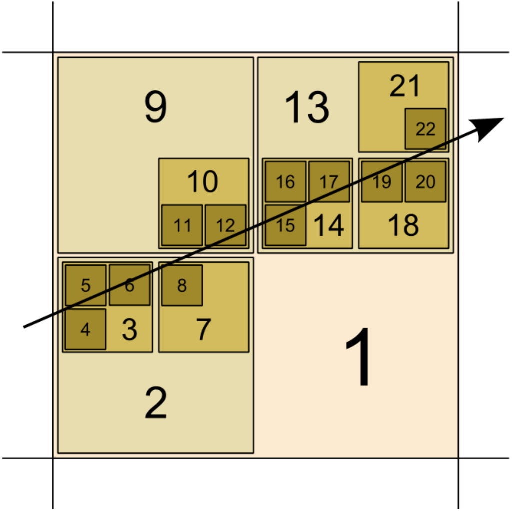
Sparse Voxel Octree

- Complete information only in the leaves
- Don't waste space encoding empty nodes



SVO Traversal

- Hierarchy traversal in depth-first order
- PUSH, ADVANCE, POP



SVO Traversal

- Each node is defined using its *parent* and an *idx* from 0 to 7
- The entire octree is contained within a cube of scale s_{max}
- Each children have dimensions that are half the parent's ones and $scale - 1$
- Each cube is axis-aligned

SVO Traversal

- We want to know the timestamp at which the ray hits the cube's faces
- Solving the ray equation $r(t) = o + t * d$ for the x-axis gives

$$t_x(x) = \frac{1}{d_x}x + \frac{-p_x}{d_x}$$

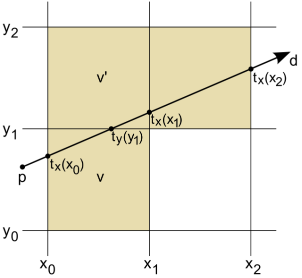
  we can do the same for t_y and t_z and precompute the coefficients

ADVANCE

- Proceed to the next sibling voxel of same dimensions
- We just need to change *idx*
- Each cube is axis-aligned and defined by two opposite vertices (x_0, y_0, z_0) and (x_1, y_1, z_1) such that
  $t_x(x_0) \leq t_x(x_1)$
  $t_y(y_0) \leq t_y(y_1)$
  $t_z(z_0) \leq t_z(z_1)$
- The t-values span intersected by the cube is given by
  $tc_{min} = max(t_x(x_0), t_y(y_0), t_z(z_0))$
  $tc_{max} = min(t_x(x_1), t_y(y_1), t_z(z_1))$

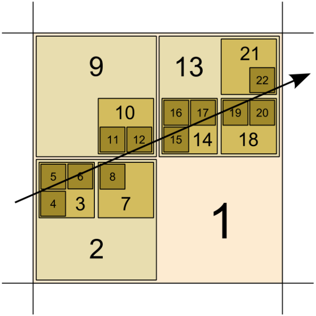
ADVANCE

- In other words there are 6 axis-aligned planes and when the ray enters the first 3 it is inside the cube, when it exits at least one of the last 3 it exits the cube
- We can determine the next voxel of same scale by comparing $t_x(x_1)$, $t_y(y_1)$, $t_z(z_1)$ with tc_{max} and for each equality we need to flip the corresponding bit in *idx*



PUSH

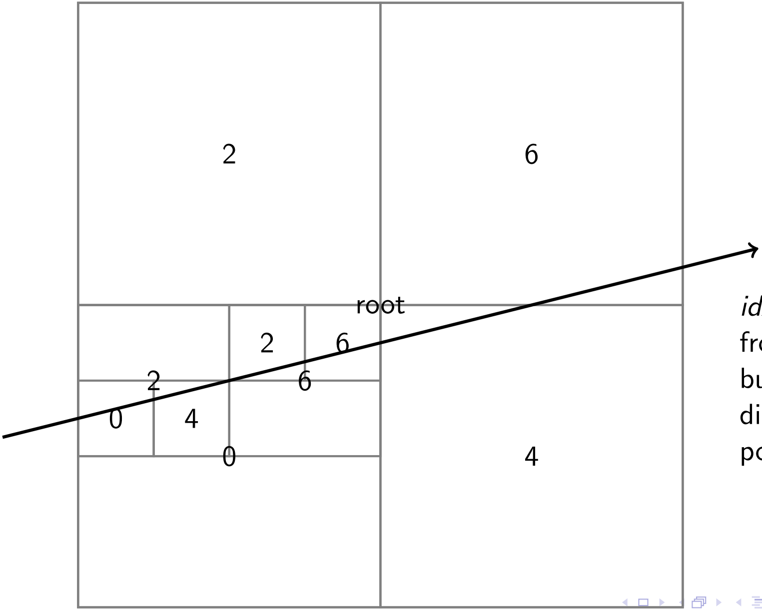
- Proceed to the child voxel that the ray enters first
- We evaluate t_x, t_y, t_z at the voxel's center (just 3 planes) and compare them against tc_{min}
- The comparison gives us which bits set in *idx*



POP

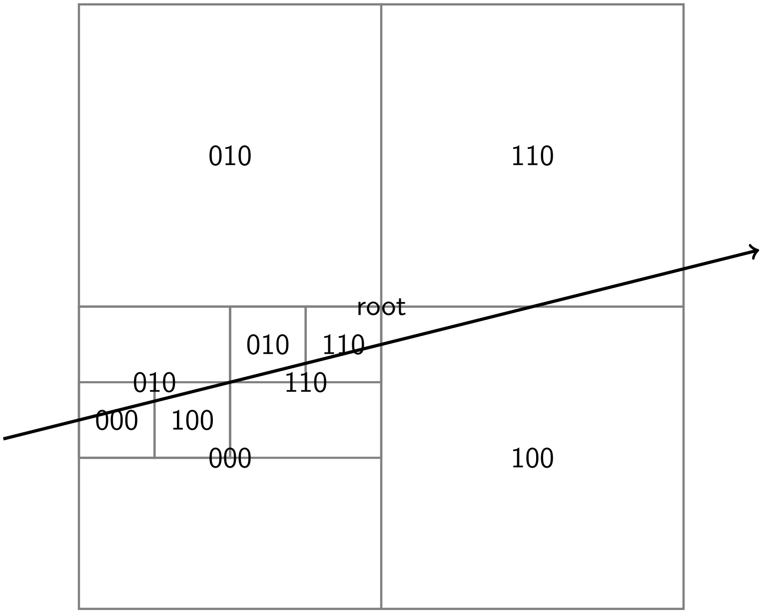
- Proceed to the next sibling of the highest ancestor that the ray exits
- We need a POP when an ADVANCE creates an *idx* that disagree with ray's direction
- This corresponds with the ray exiting not only from the current node, but also from its parent

POP: Common Ancestor

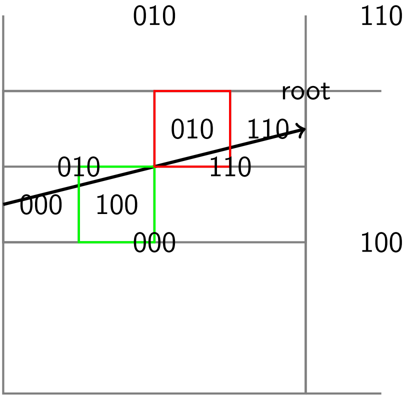


idx changes from 4 to 6 but the ray direction is positive!

POP: Common Ancestor



POP: Common Ancestor



Position at 024

	0	1	2	3
x	0	0	0	1
y	0	0	1	0
z	0	0	0	0
		0	2	4

Ray

x	1
y	1
z	0

Position at 062

	0	1	2	3
x	0	0	1	0
y	0	0	1	1
z	0	0	0	0
		0	6	2

POP: Common Ancestor

My implementation differs from the paper's one because I don't need to descend from the highest common ancestor to collect contours' information

Pseudocode

```
INITIALIZE
(t_min, t_max) ← (0, 1)
t′ ← project cube(root, ray)
t ← intersect(t, t′)
h ← t′_max
parent ← root
idx ← select child(root, ray, t_min)
(pos, scale) ← child cube(root, idx)
while not terminated do
  tc ← project cube(pos, scale, ray)
  if voxel exists and t_min ≤ t_max then
    if voxel is small enough then return t_min
    INTERSECT
    tv ← intersect(tc, t)
    if voxel has a contour then
      t′ ← project contour(pos, scale, ray)
      tv ← intersect(tv, t′)
    end if
    if tv_min ≤ tv_max then
      if voxel is a leaf then return tv_min
      PUSH
      if tc_max < h then stack[scale] ← (parent, t_max)
      h ← tc_max
      parent ← find child descriptor(parent, idx)
      idx ← select child(pos, scale, ray, tv_min)
      t ← tv
      (pos, scale) ← child cube(pos, scale, idx)
      continue
    end if
  end if
  ADVANCE
  oldpos ← pos
  (pos, idx) ← step along ray(pos, scale, ray)
  t_min ← tc_max
  if idx update disagrees with ray then
    POP
    scale ← highest differing bit(pos, oldpos)
    if scale ≥ s_max then return miss
    (parent, t_max) ← stack[scale]
    pos ← round position(pos, scale)
    idx ← extract child slot index(pos, scale)
    h ← 0
  end if
end while
```

My implementation differs in the INTERSECT and *pos*-related parts

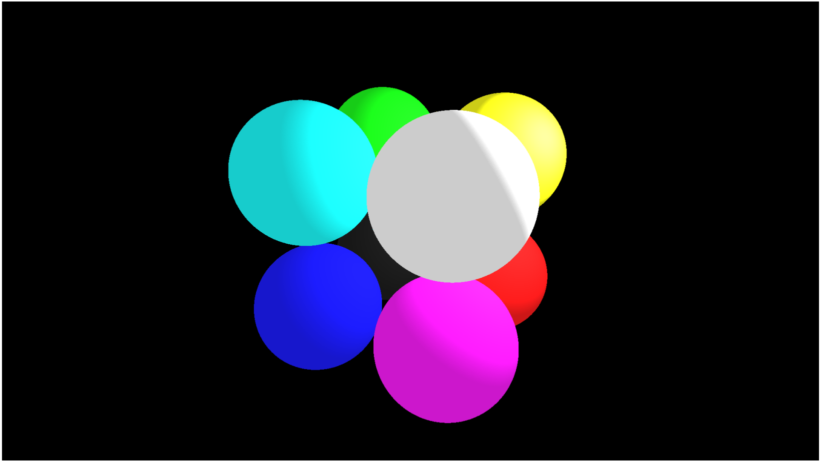
Implementation: Difficulties

- Difficulties? A lot...
- Several implementation specific problems that I had to overcome on my own
- Debugging was really time consuming because there were several factors to take into consideration
- Even when things (seem to) work, you're afraid that it could be only a lucky case
- Hofstadter's Law: It always takes longer than you expect, even when you take into account Hofstadter's Law.

But in the end highly rewarding

Repo link:
`https://github.com/Nyriu/RayTracer/tree/octree`

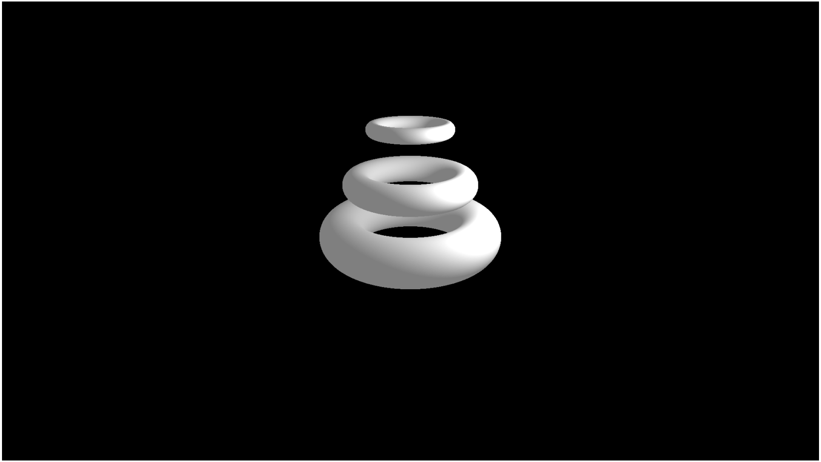
Implementation: Results



1920x1080

SVO	s	μs
No	29	29747704
Yes	15	15876470

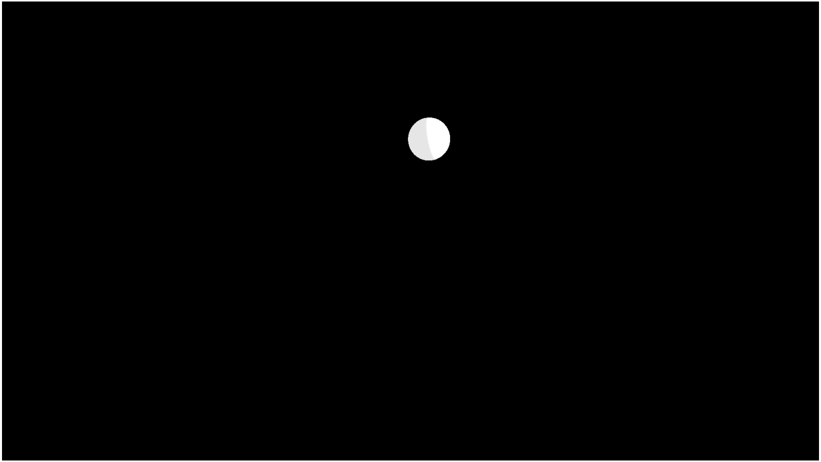
Implementation: Results



1920x1080

SVO	s	µs
No	9	9158535
Yes	5	5330879

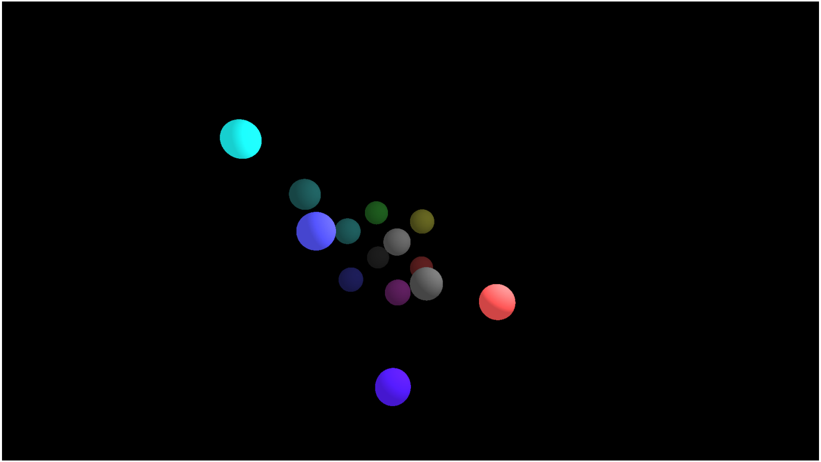
Implementation: Results



1920x1080

SVO	s	μs
No	3	3841319
Yes	2	2262713

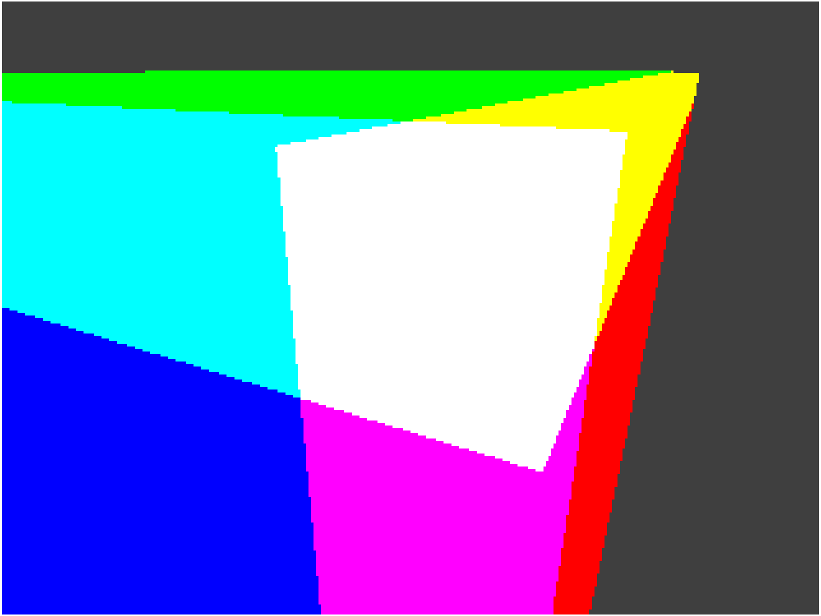
Implementation: Results



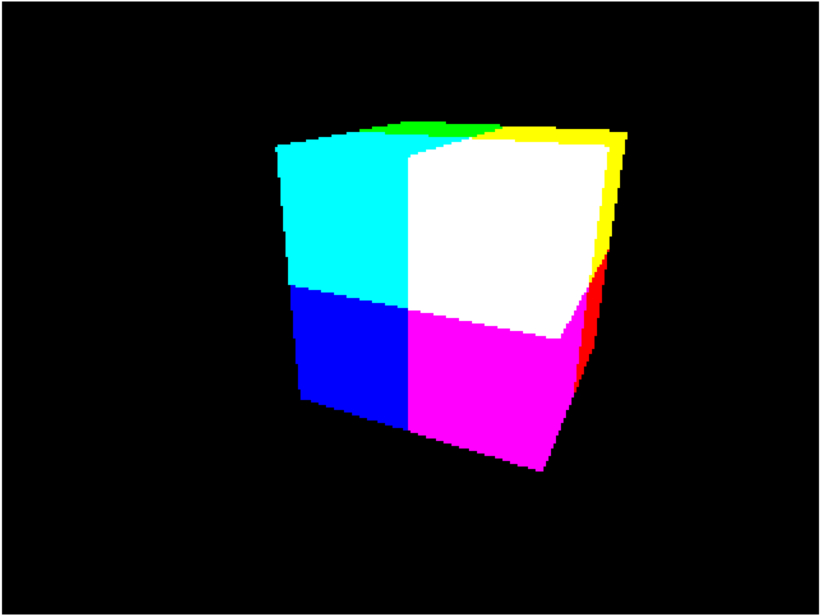
1920x1080

SVO	s	μs
No	31	31424528
Yes	9	9124618

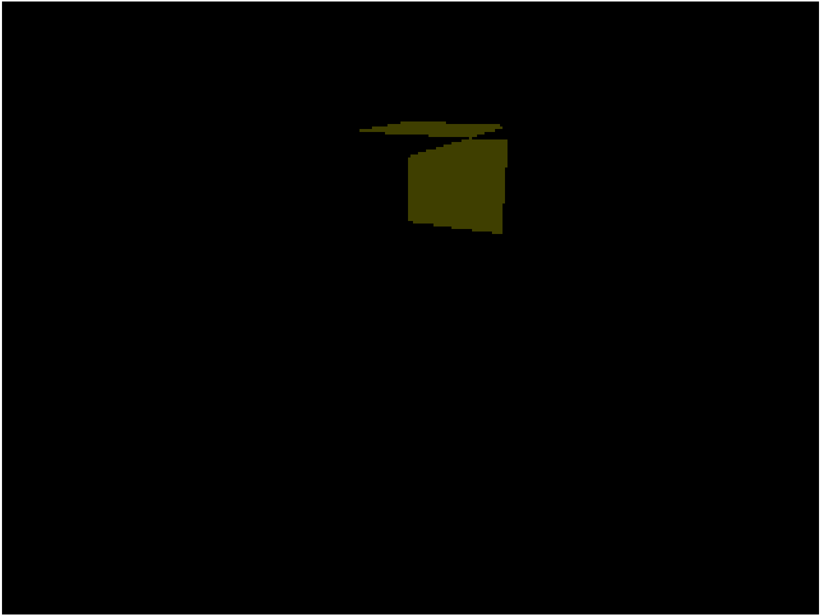
Implementation: Helpers



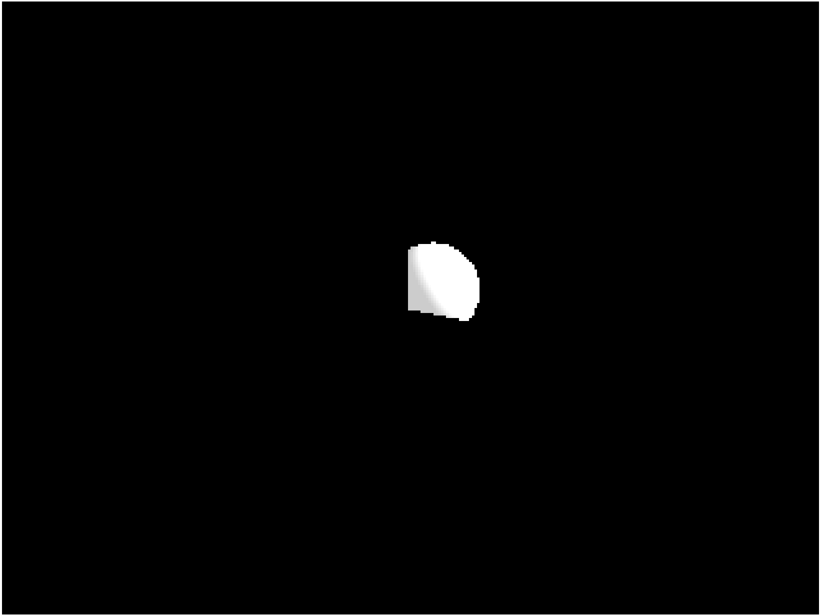
Implementation: Helpers



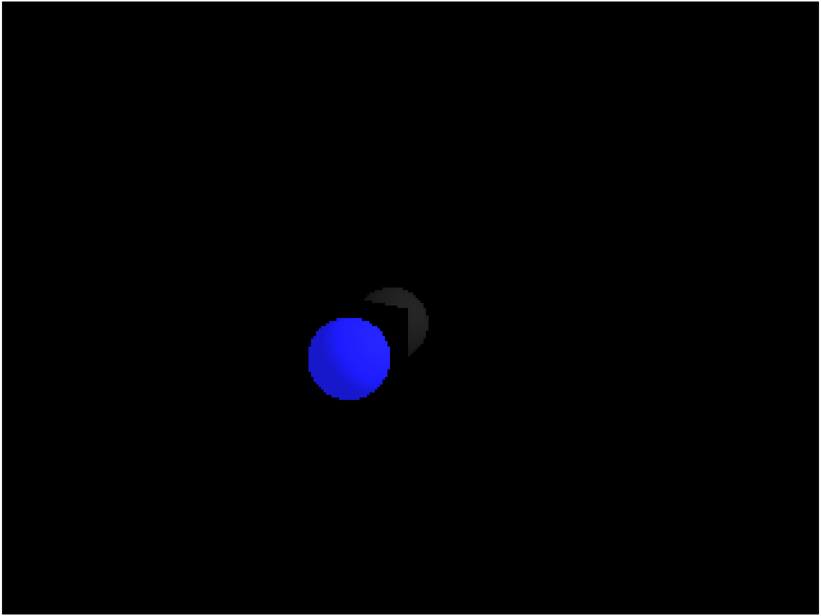
Implementation: Initial Bugs



Implementation: Initial Bugs



Implementation: Initial Bugs



Future

- Cast shadow rays
- Improve the traversal algorithm and add shortcuts
- Add materials system
- Add different sampling patterns for octree generation
- Parallelism

References I

- Jhon C. Hart, *Sphere tracing: a geometric method for antialiased ray tracing of implicit surfaces*, The Visual Computer (1996).
- Inigo Quilez, *Deriving the sdf of a box*.
- ———, *Distance functions*.
- Tero Karras Samuli Laine, *Effcient sparse voxel octrees*, NVIDIA Research (2010).

Thank You