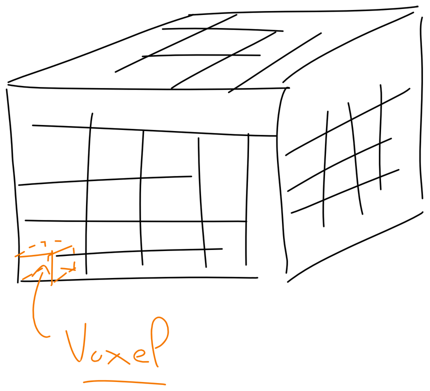
02/01 Modelization

① Voxels
② Implicit surfaces
③ Meshes

Voxels

↳ 3D Pixels



Voxel stores attributes
- binary
  ↳ inside / outside
- density
  ↳ grayscale
- colors
- . . . .

Implicit surfaces

Equation $f(x,y,z) = 0$ ← Isosurface

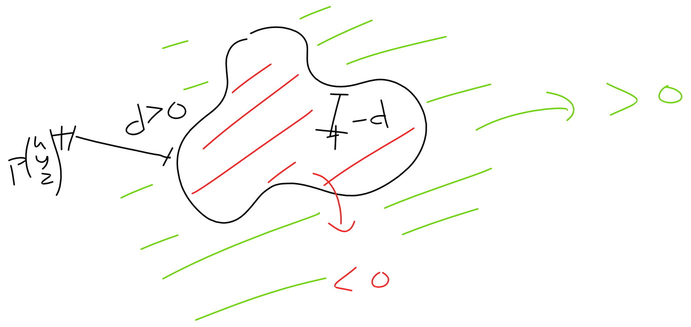
Example

Sphere center $C = \begin{pmatrix} c_x \\ c_y \\ c_z \end{pmatrix}$, radius r

$$\underbrace{(x-c_x)^2 + (y-c_y)^2 + (z-c_z)^2 - r^2}_{f(x,y,z)} = 0$$

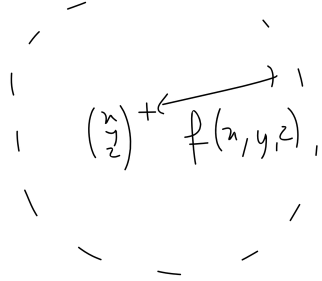
Distance functions

$f(x,y,z)$ signed distance to the object


$p\begin{pmatrix} x \\ y \\ z \end{pmatrix}$
$d>0$
$-d$
>0
<0

Often a bound is sufficient.

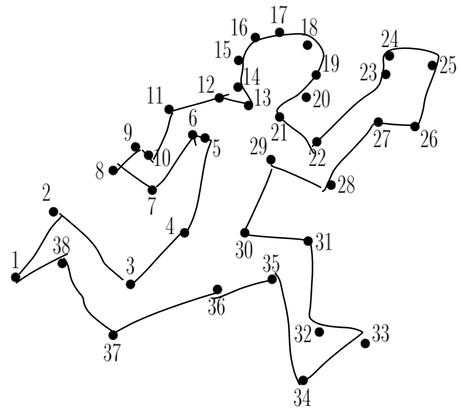
$$f(x,y,z) < d \quad \text{if } d > 0$$
$$f(x,y,z) > d \quad \text{if } d < 0$$

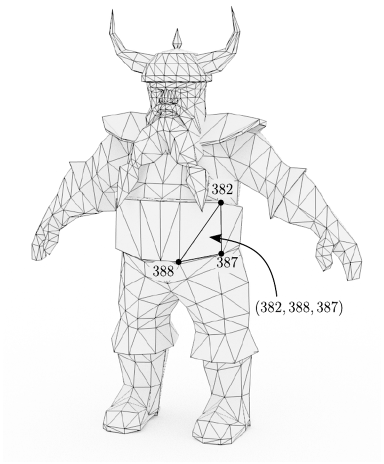

$\begin{pmatrix}x\\y\\z\end{pmatrix} + t$
$f(x,y,z)$

No intersection with the isosurface

Meshes

Here, the segments are implicit.


1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38


382
387
388
(382, 388, 387)

① Collection of vertices → triplets of coordinates

$$\underbrace{x\ y\ z}_{v_0}\ \Big|\ \underbrace{x\ y\ z}_{v_1}\ \Big|\ - - - - -$$

② Collection of triangles → triplets of vertex indices

$$\underbrace{v_0\ v_1\ v_2}_{t_0}\ \Big|\ \underbrace{v_0\ v_1\ v_2}_{t_1}\ \Big|$$

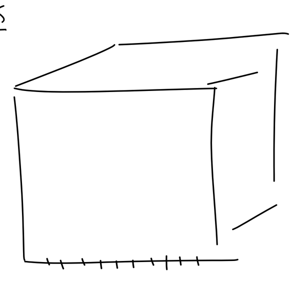
Goal

↳ advantages / drawbacks for every representation

↳ examples of things you would / would not represent

	Voxels	Implicit surfaces	Meshes
Pros	Simple Procedural Editing Acquisition	Smoothness Light Animate t parameter (finding right function) Editing (union, intersection)	Light Rendering Edit (free) Animation
Cons	Heavy Animation Smoothness	Render ? Edit (grab, pull)	Smoothness No "interior" data
Examples	Minecraft PDE RMI	Design Mechanical parts Simple shapes	Characters

Voxels

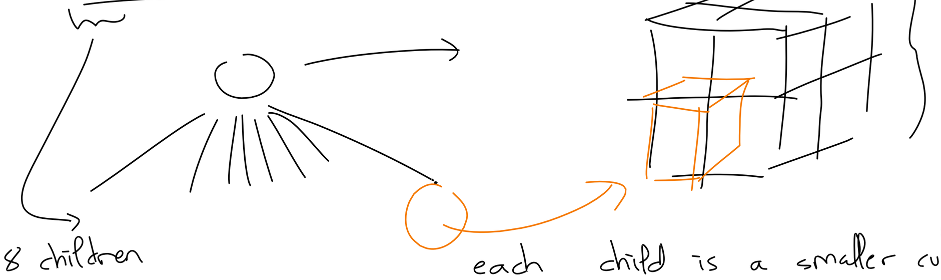


→ 10^9 voxels

⇒ very costly in memory if stored naively

1000

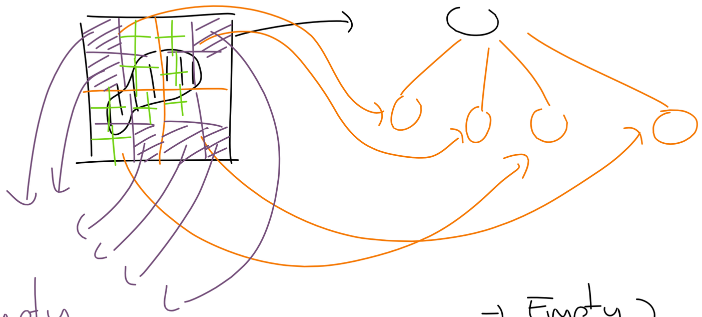
↳ octrees


8 children
each child is a smaller cube

Dividing by 2 along each axis
⇒ 8 children

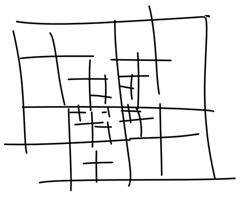
binary function

2D quadtree



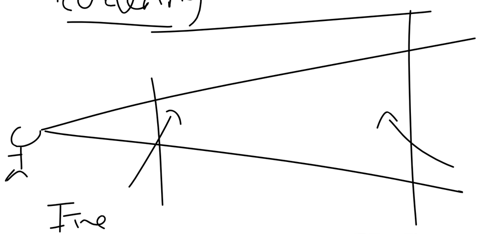
⇒ no more splitting

→ Empty } no more splitting
→ Full



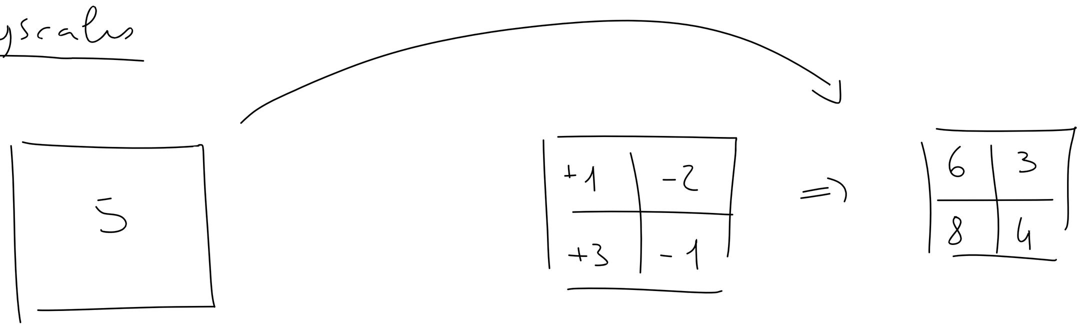
⇒ Local resolution, depending on how much details are locally necessary

Rendering


Coarse resolution
Fine resolution

⇒ Adapt resolution to view point
⇒ Load / unload data based on pixel size.

Grayscales



Average value of a node

Difference in its children
↳ reduces entropy ⇒ better compression
↳ threshold (loss) ⇒ neglect small differences
⇒ sparse data

Gigavoxels (Cyril Crassin)

Implicit surfaces ~ Vector images
↳ svg for 3D

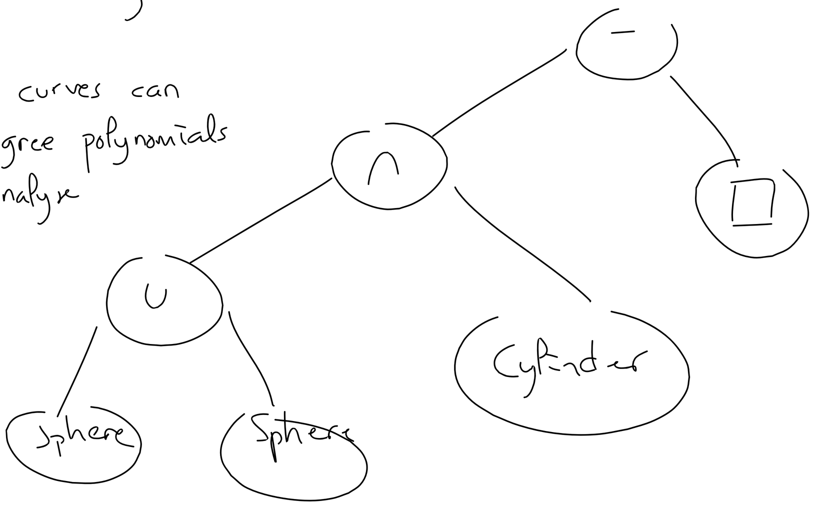
Storage

↳ A set of basic primitives (spheres, parallelepipeds, cylinders, cones) with parameters

↳ Union, Difference, Intersection operations

⇒ CAD modelling.

⚠ Intersection curves can be high degree polynomials difficult to analyze


−
∩
□
∪
Cylinder
Sphere
Sphere

⇒ Constructive representation → command CNC
→ 3D printing

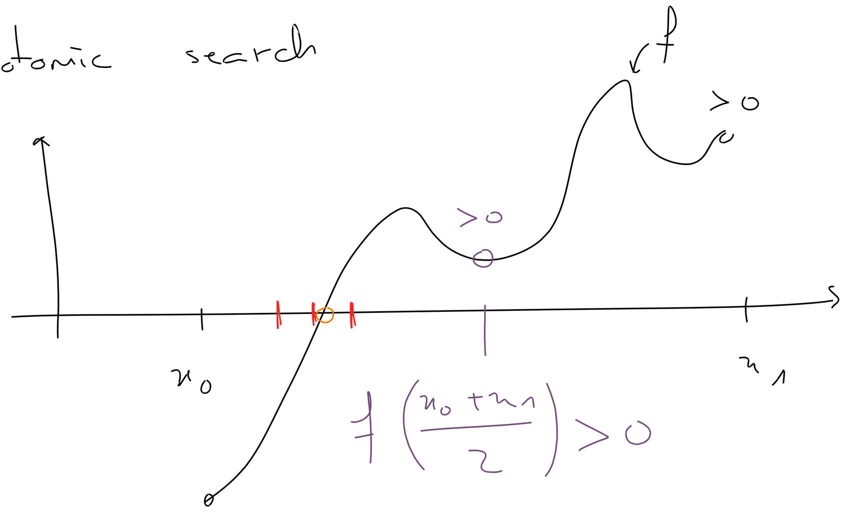
Rendering

- $f(x,y,z) = 0$

- f continuous, $f(x_0) < 0$, $f(x_1) > 0$

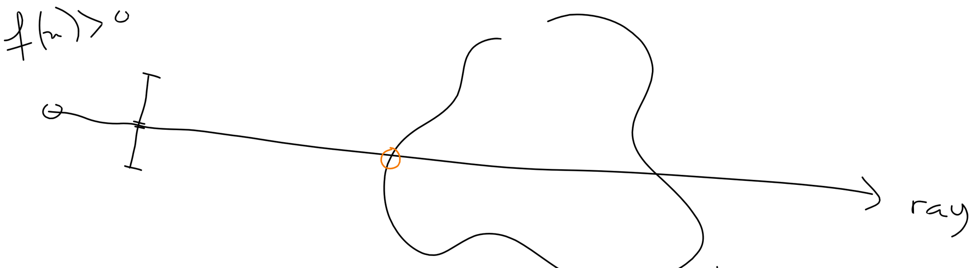
⇒ $\exists x \in [x_0, x_1]$ $f(x) = 0$

Dichotomic search


f
>0
>0
x_0
x_1
$f\left(\frac{x_0+x_1}{2}\right) > 0$
<0

⇒ $x \in \left[x_0, \frac{x_0+x_1}{2}\right]$, $f(x) = 0$

⇒ Converges towards the x, $f(x) = 0$

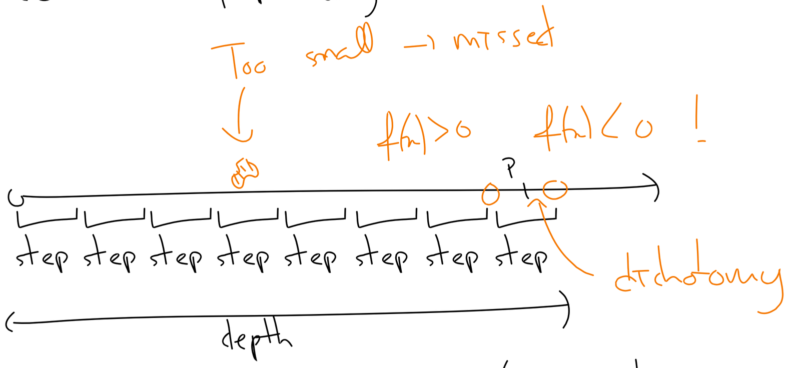


Rendering a pixel ⇒ ray / object intersection

If $f(x) > 0$ at the view point, find x, $f(x) < 0$?

⇒ Walk on the ray (ray marching)

→ Constant step length


Too small → missed
f(x)>0
f(x)<0 !
p
step step step step step step step step
dichotomy
depth

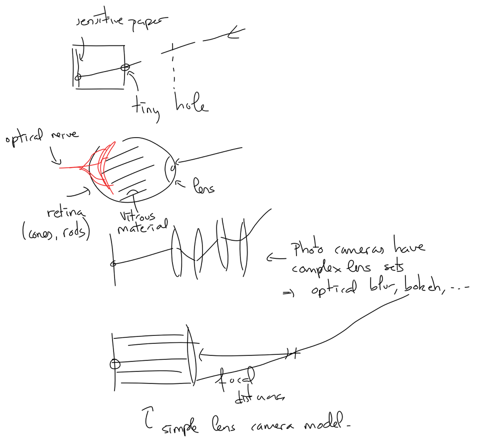
↳ point → depth

↳ normal → $\frac{\partial f}{\partial x}(p)$

$\frac{\partial f}{\partial y}(p)$

$\frac{\partial f}{\partial z}(p)$

Digression on camera models


sensitive paper
tiny hole
optical nerve
retina (cones, rods)
vitrous material
lens
focal distance

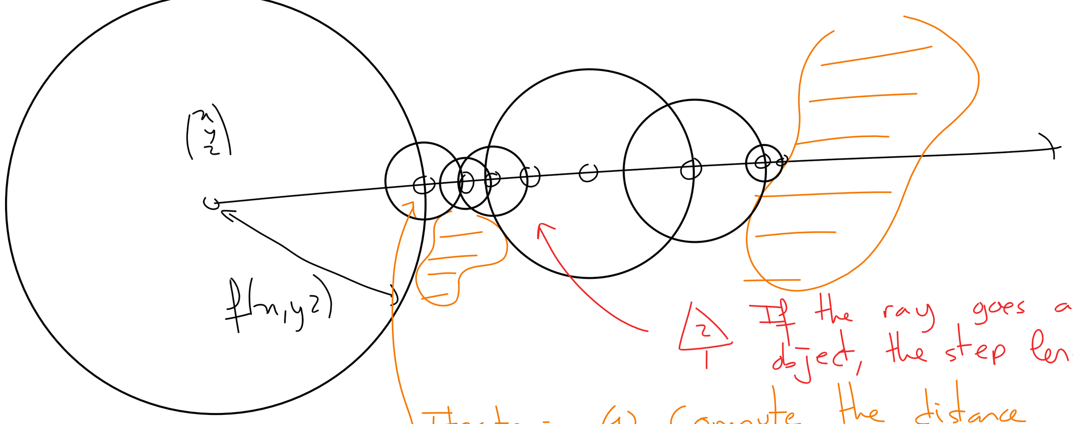
← Photo cameras have complex lens sets
⇒ optical blur, bokeh, ...

↑ simple lens camera model.

- Exponential step → accelerate the further you get
  → missed objects get bigger and bigger

step
2× step
4× step
- - - - -

- Sphere tracing
  ↳ distance function ⇒ empty ball around the query point


$\begin{pmatrix}x\\y\\z\end{pmatrix}$
f(x,y,z)

⚠ If the ray goes arbitrarily close to an object, the step length goes arbitrarily small

Iterate = ① Compute the distance
② Advance on the ray of that length

Meshes

Naive

array of vertices | x y z | x y z | - - -

array of triangles | v0 v1 v2 | v0 v1 v2 | - - - -

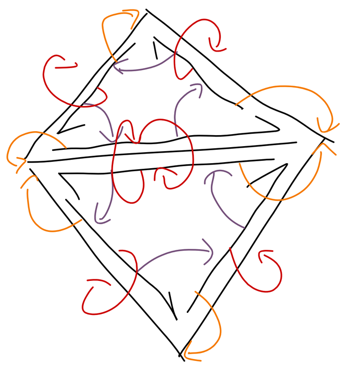
Vertex → neighboring vertices?
Face → neighboring faces?

⇒ A simple array structure is not well suited for traversing the mesh

```
v = vertex
neighbor = [ ]
Θ(#triangles)  for each triangle
                   if v is in the triangle
                       add the other vertices to neighbors
```

```
f face
neighbor = [ ]
Θ(#triangles)  for each triangle t
                   if t has two vertices in common with f
                       add t as a neighbor.
```

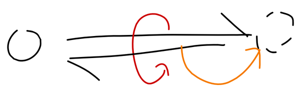
Half-edges (Muller, Preparata, 1977)



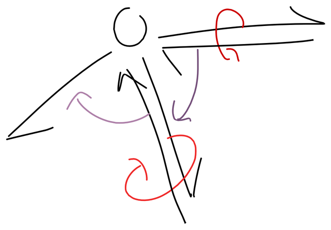
Half-edge
↳ source vertex 🟧
↳ next Half edge 🟪
↳ opposite Half edge 🟥

Vertex
↳ outgoing Half edge

Face
↳ one half edge



v → edge → opposite → source



$\underbrace{\text{edge} \to \text{opposite} \to \text{next}}_{\text{turns around a vertex}}$

Smooth surfaces can be described using the mesh as a control structure for splines.

→ Bézier curves generalized to 3D

(Catmull-Clark for quad meshes, loop for triangles)