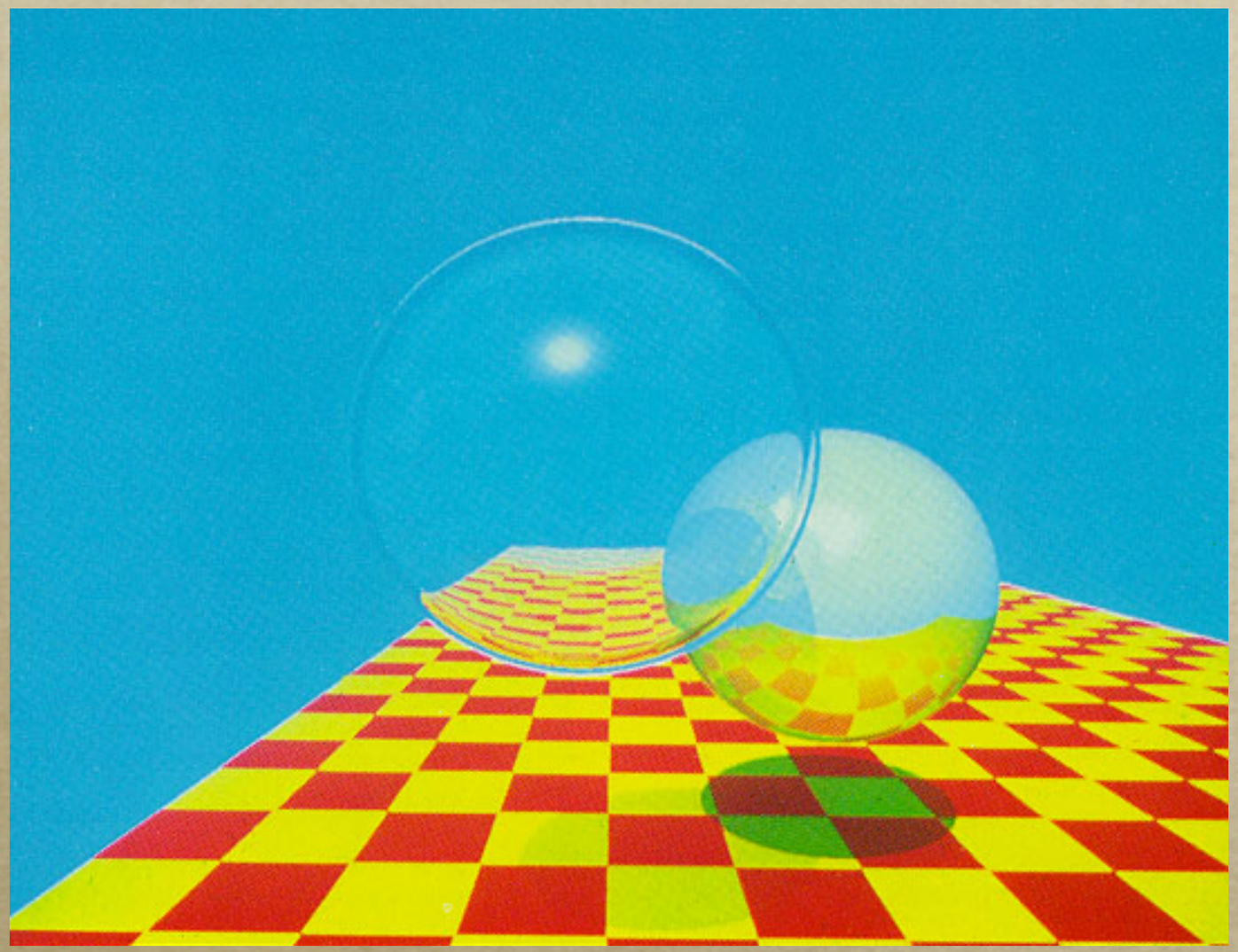
Ambient, Lambert, Phong, reflection, refraction, point light sources.

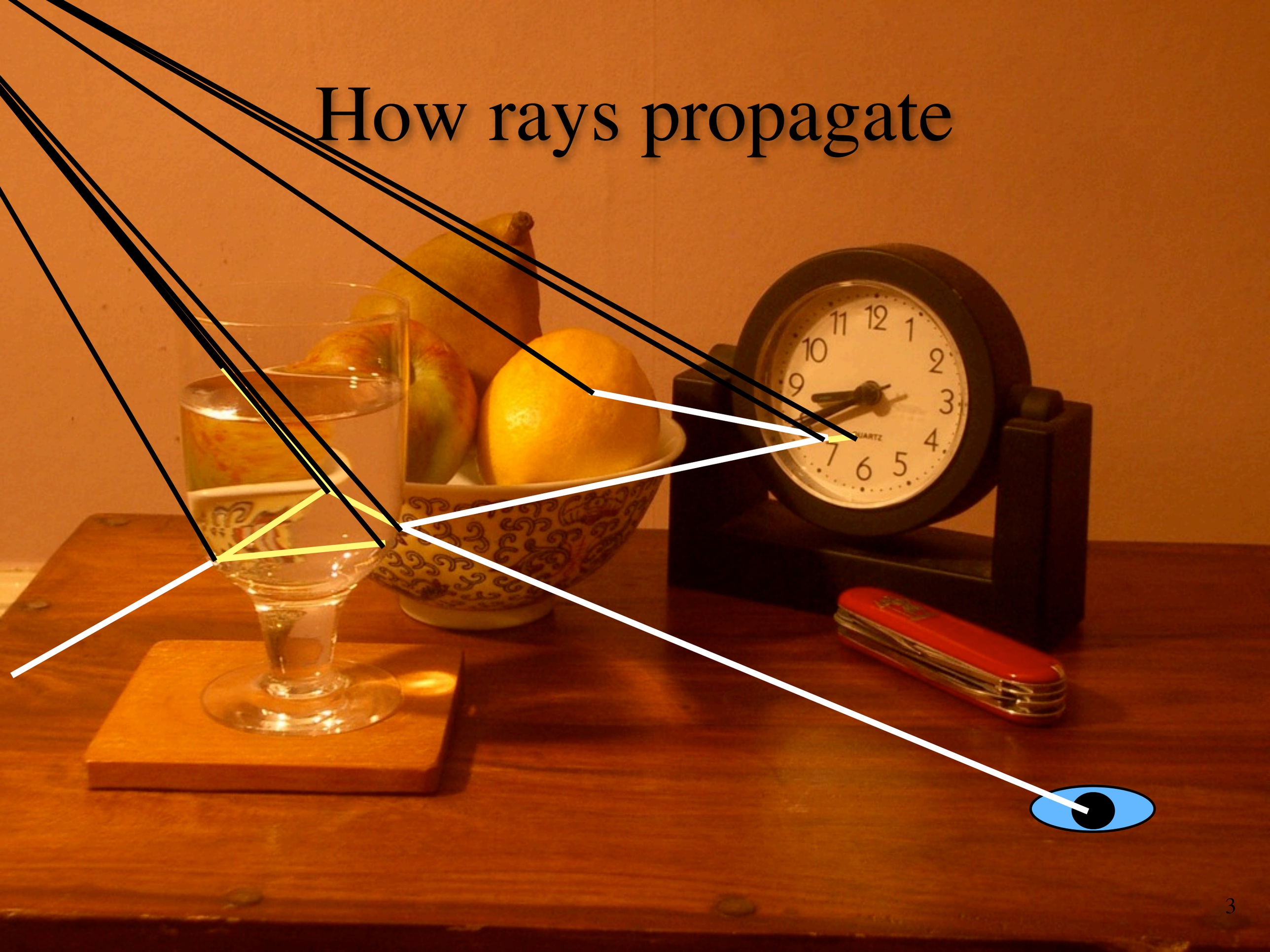


(Slides on spatial content thanks to James Arvo and David Kirk)

Just the beginning…

- *Aliasing artefacts*
- *No surface/surface illumination*
- *No caustics*
- *Real shadows are soft*
- *Colour problems*
- ***Very slow***

How rays propagate



Where are we spending the time?

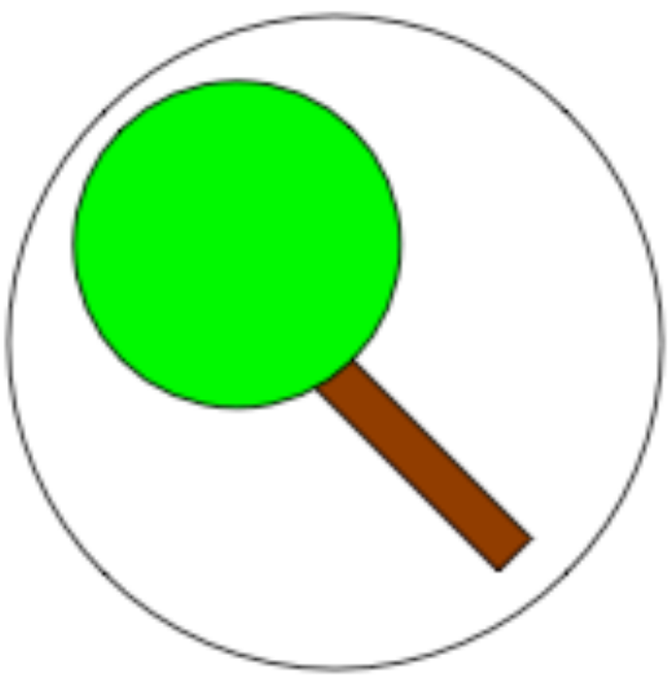
- *1000 x 1000 pixels*
- *Say 6 secondary rays per pixel*
- *100k objects; 10 ops per intersection*
- *2GHz processor; maybe 5 cycles per op*
- *So how long to render?*
- *And is that really all?*
- *And: where is the time really spent?*

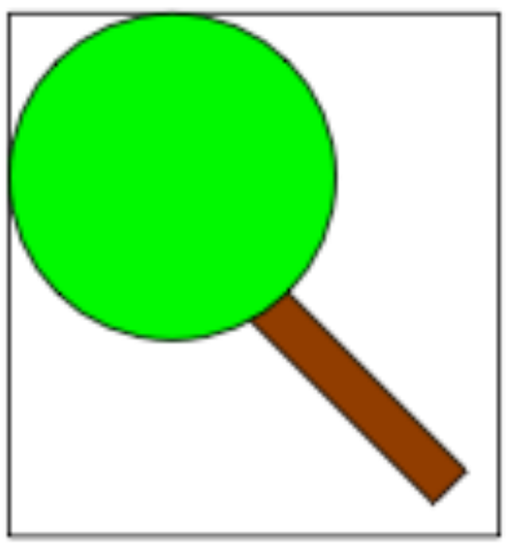
Bounding Volumes

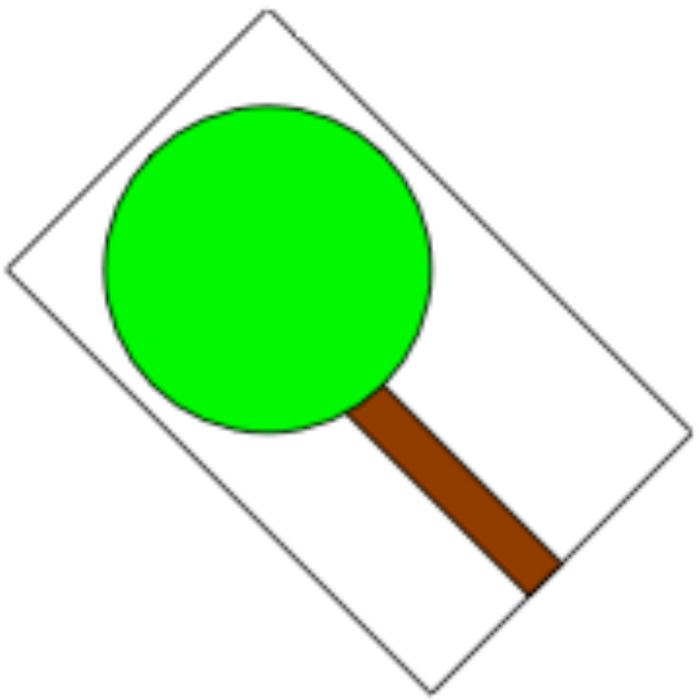
- *Enclose objects inside a volume with a simple intersection test (e.g. a sphere)*
- *You only need to know IF the ray hits the volume, not where*
- *Does this decrease or increase computation? It depends…*
- *Cost: n * B + m * I*
  - *n rays, B cost of intersection with bounding volume, m rays intersect bounding volume, I cost of intersecting with objects.*

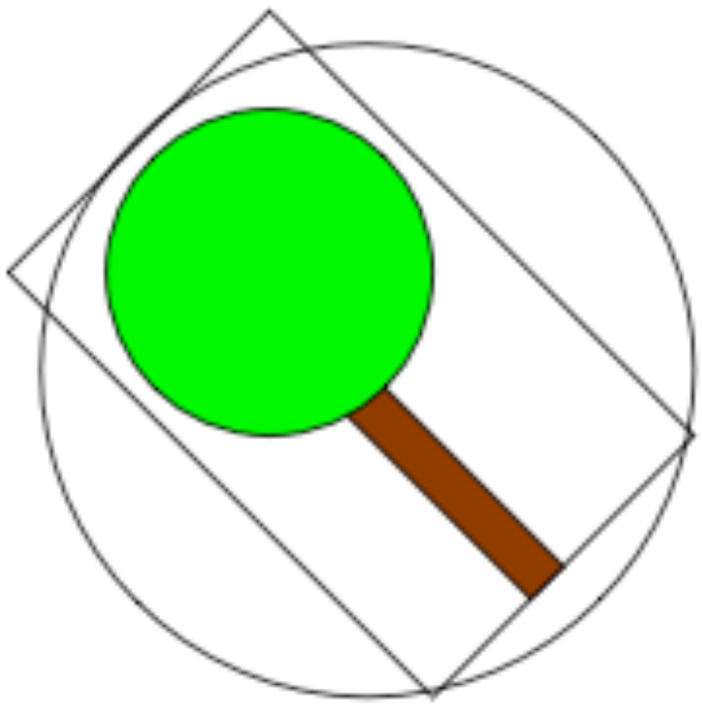
Bounding Volumes

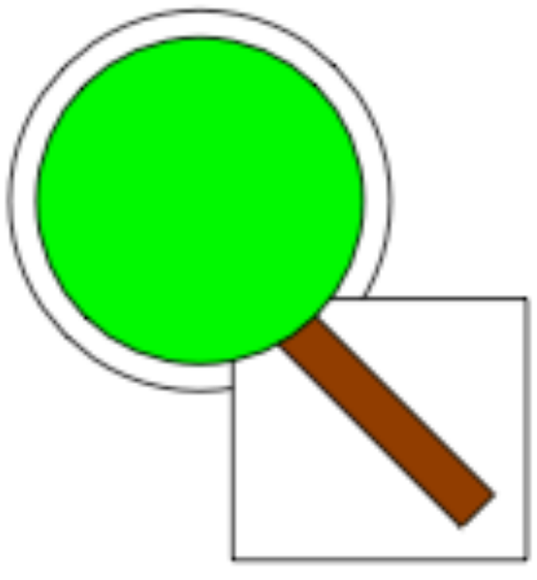
Bounding Volumes











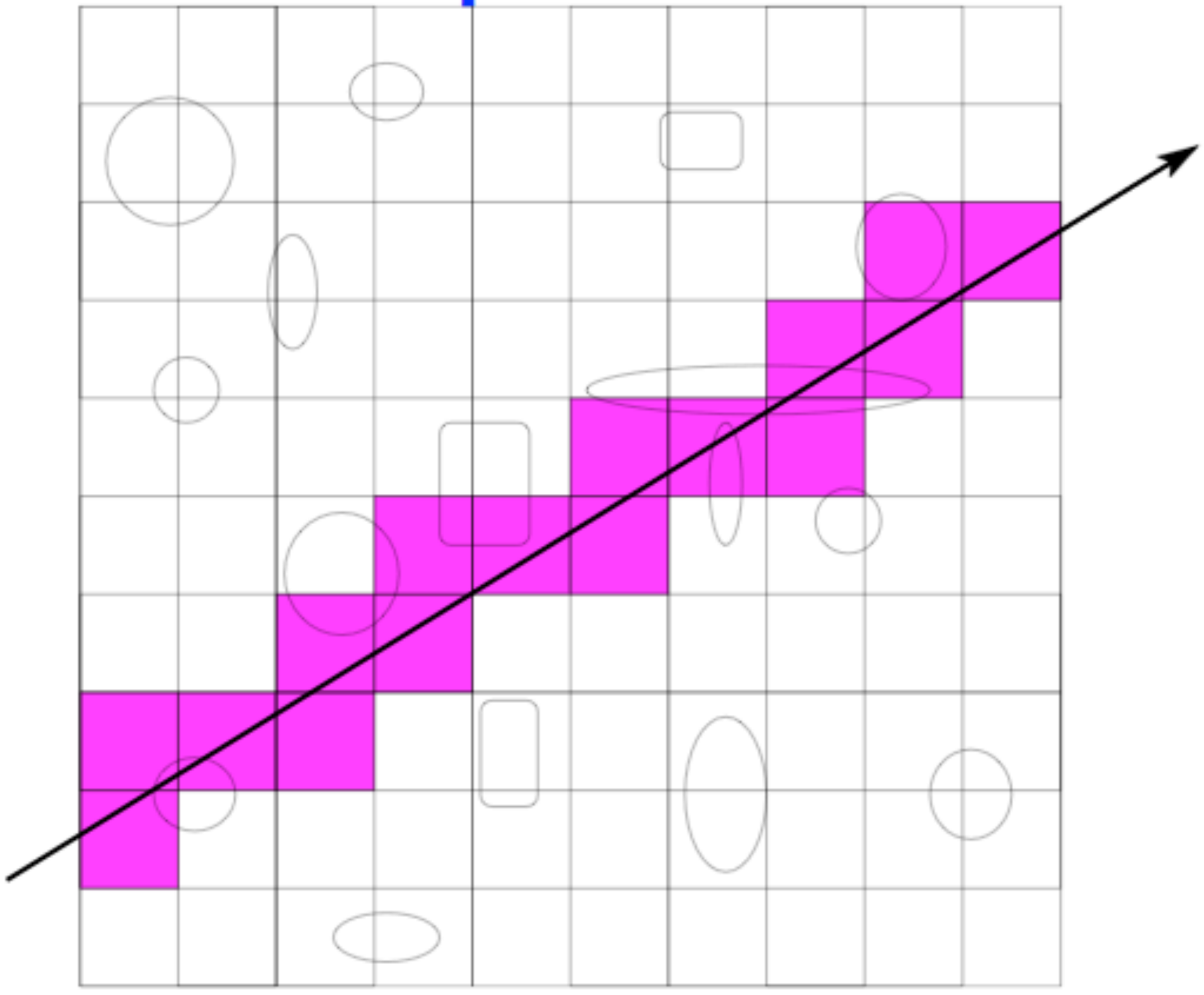
Hierarchical Volumes

- *Put volumes within volumes*
- *I.e., we form a **tree** of bounding volumes*
- *If the volumes are placed **really well**, then we get O(log n) intersection tests*
- *Unfortunately, it isn't automatic*
- *Non-spherical volumes produce tighter bounds, but aren't automatic either.*

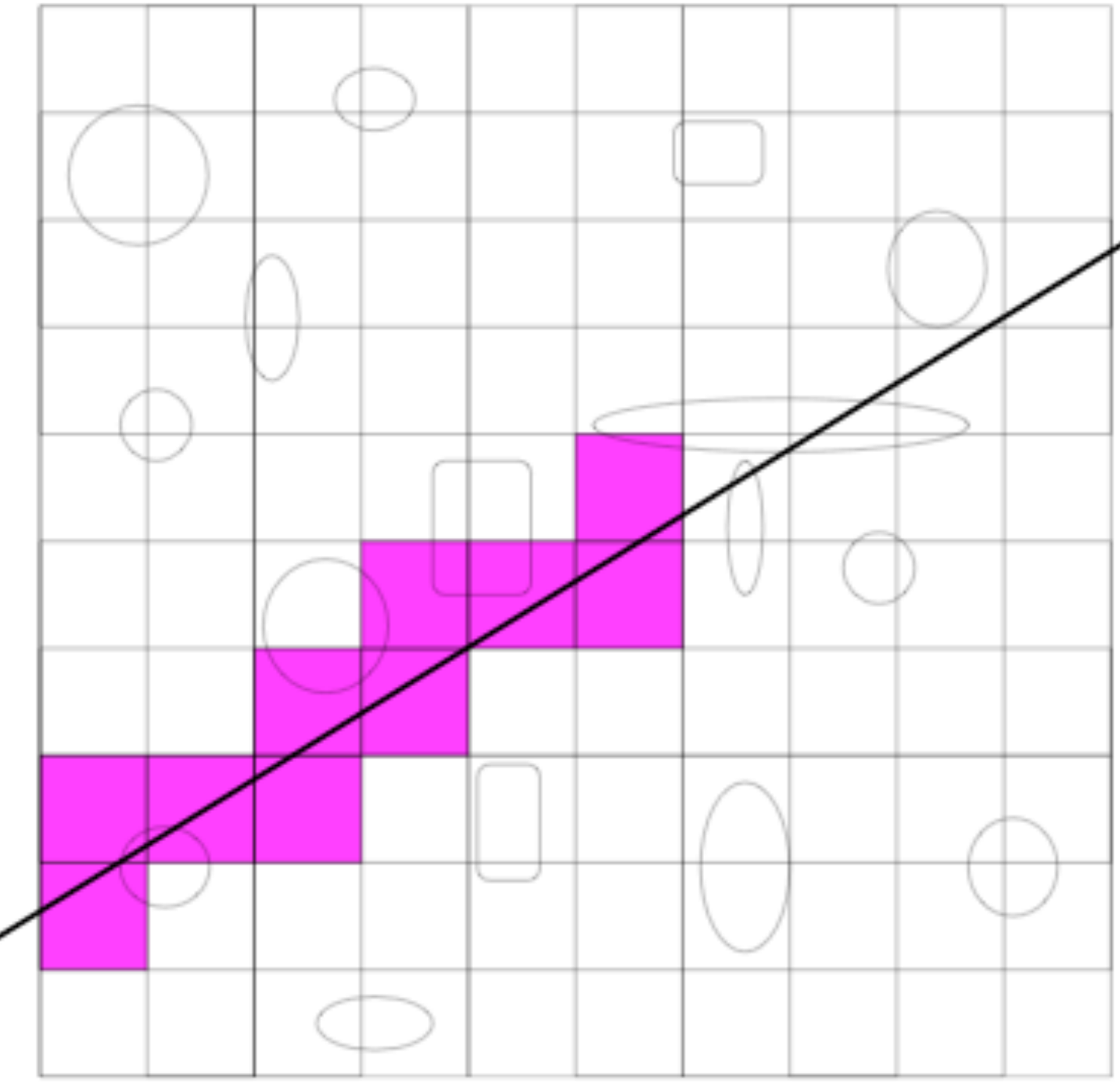
Spatial Subdivision

- *Rather than adding new (invisible) objects as boundaries…*
- *Let's just divide the space.*
- *If a **picture element** is called a **pixel**…*
- *Then a **volume element** must be called?*
  - *A **voxel***

Uniform Space Subdivision

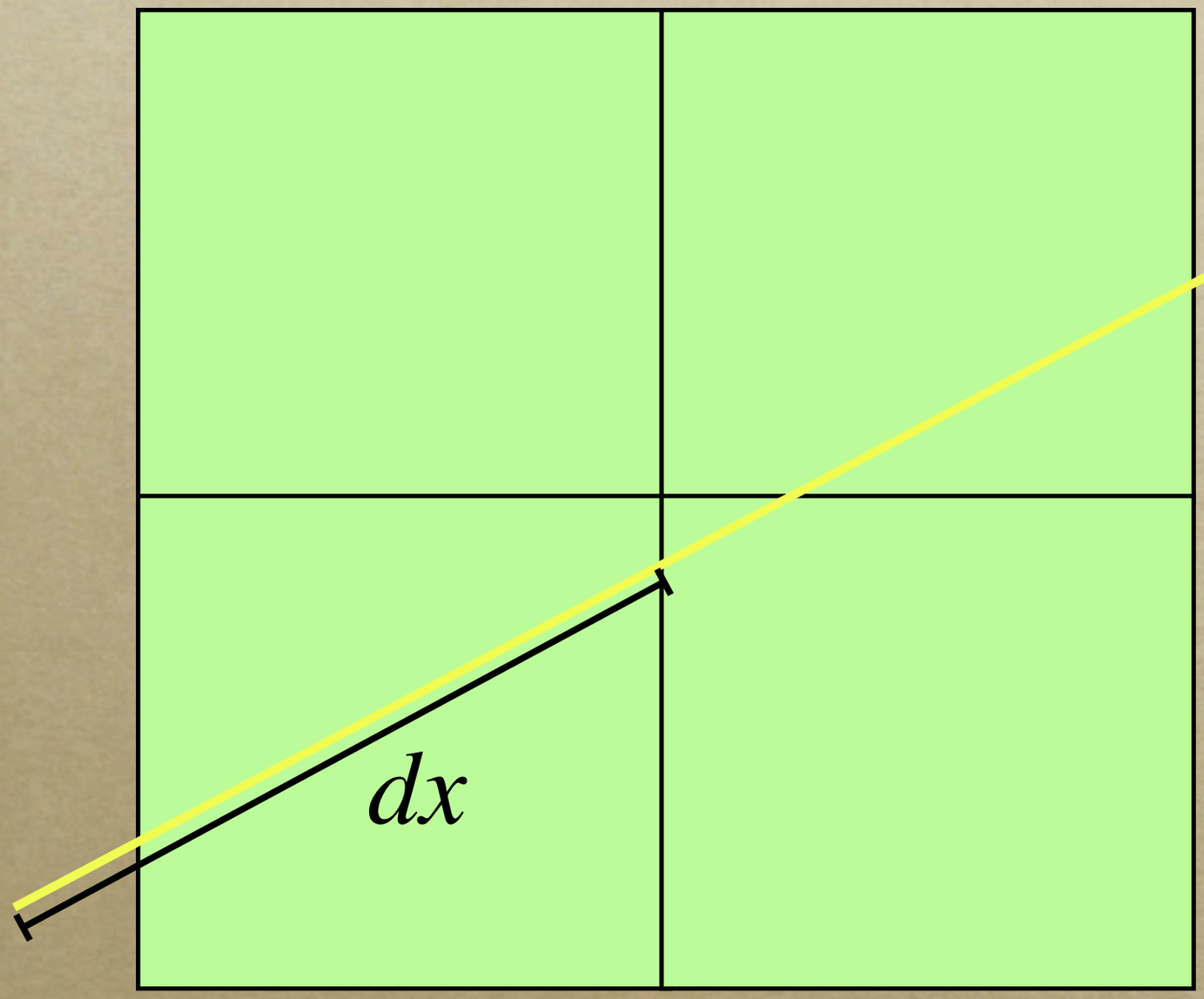


Uniform Space Subdivision



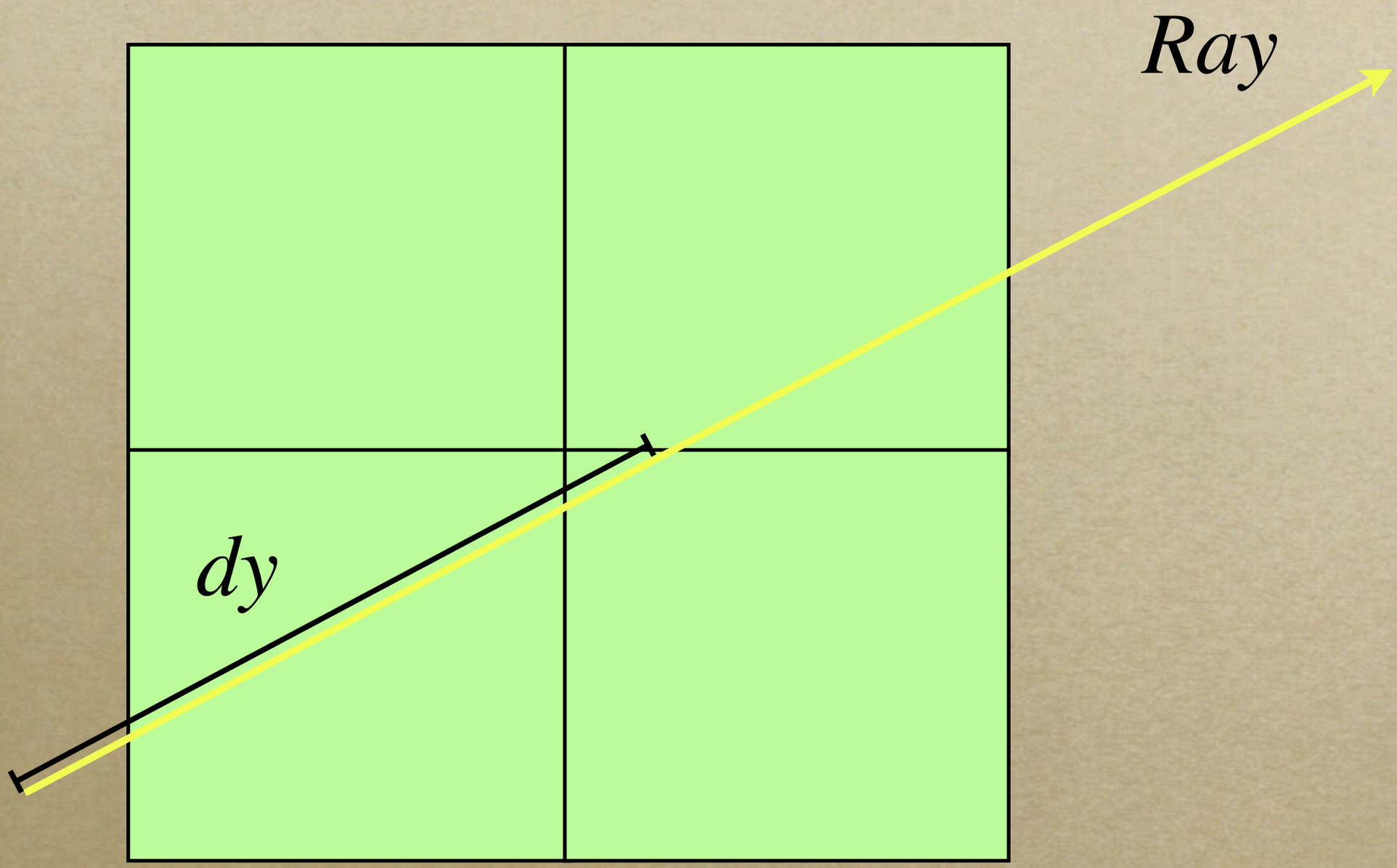
How do we determine the next voxel to test?

Cleary's Algorithm

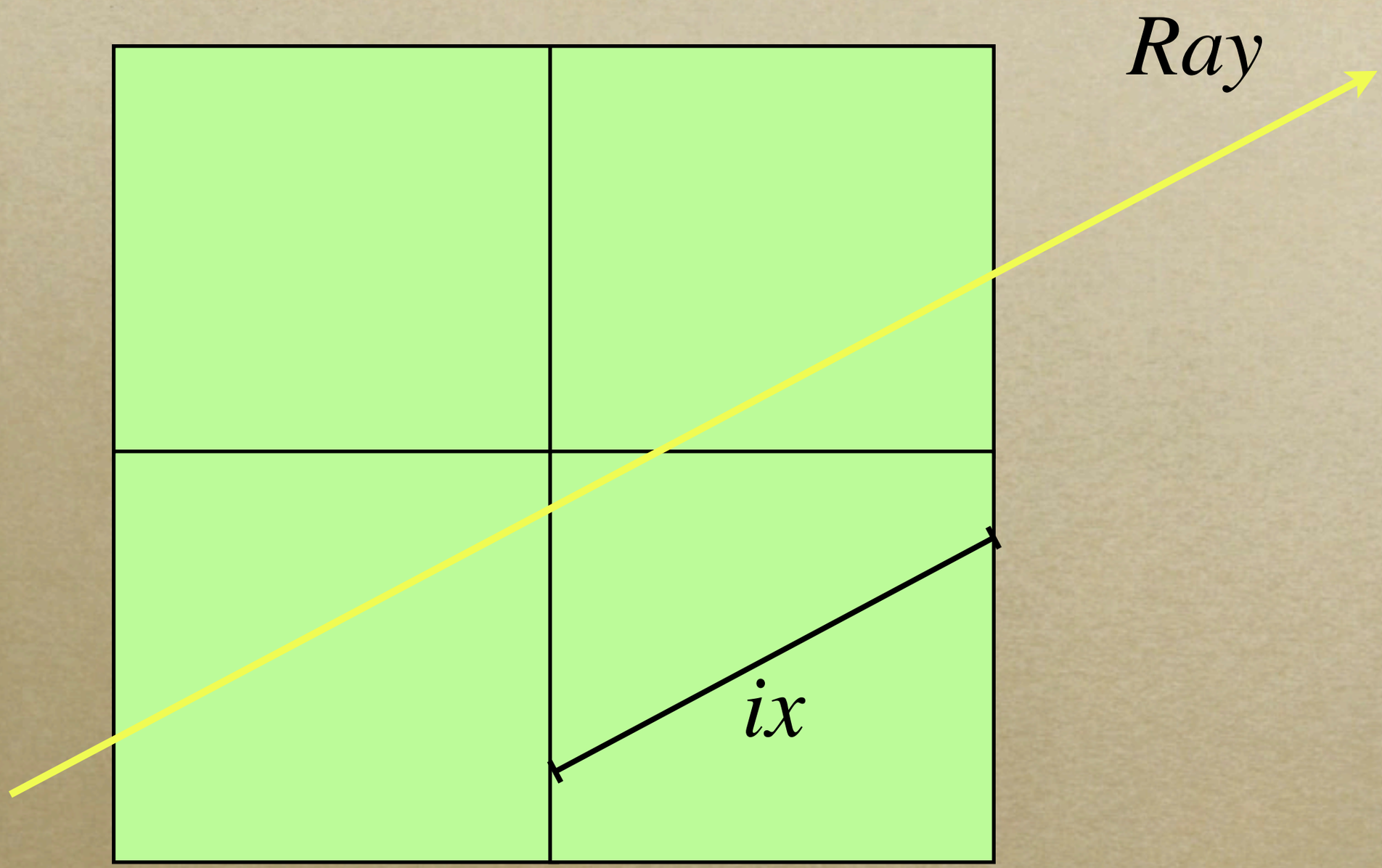


dx is the distance travelled along the ray from its start to when the ray crosses a cell boundary in x (i.e. a vertical grid line in this case)

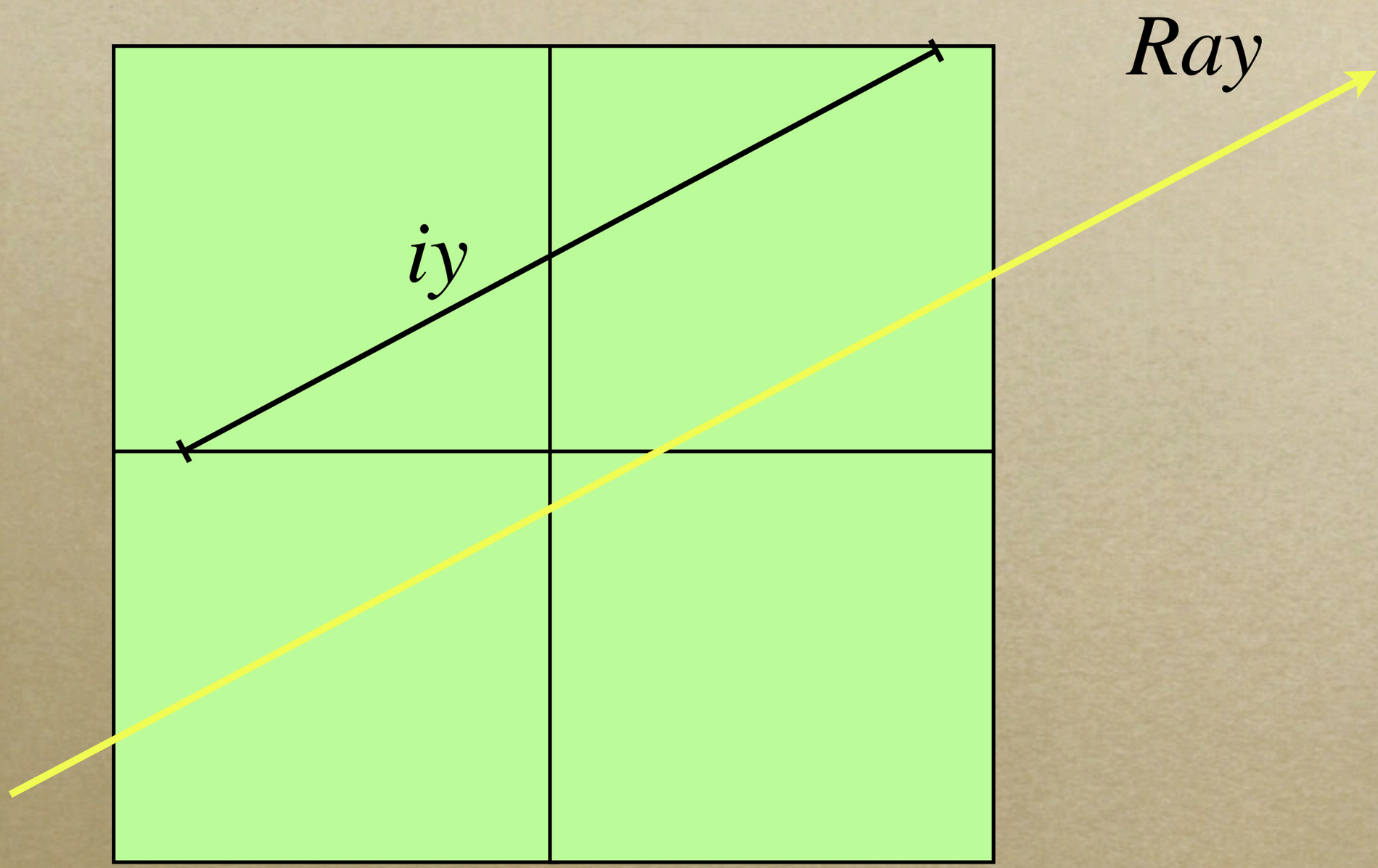
Cleary's Algorithm



Cleary's Algorithm



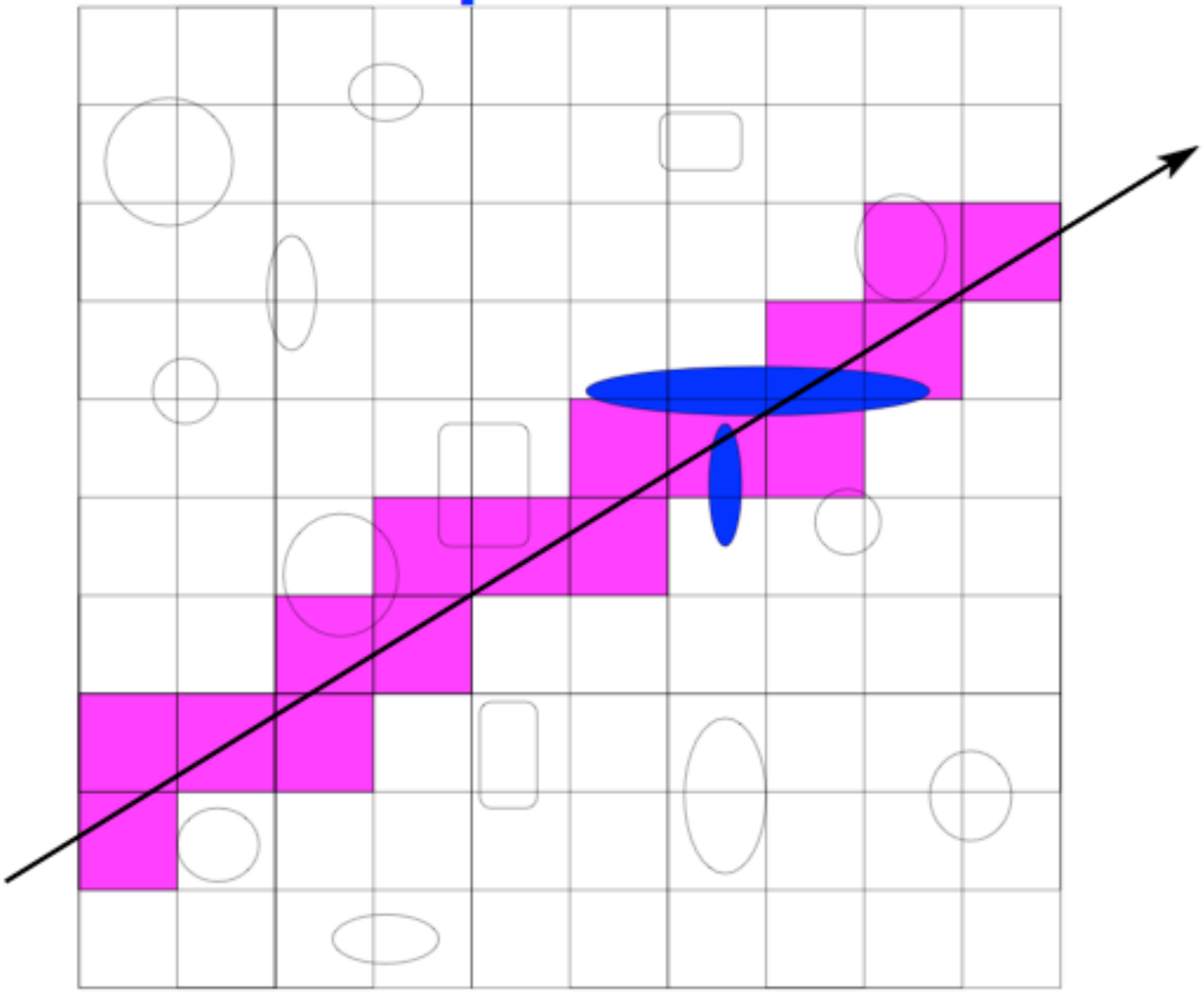
Cleary's Algorithm



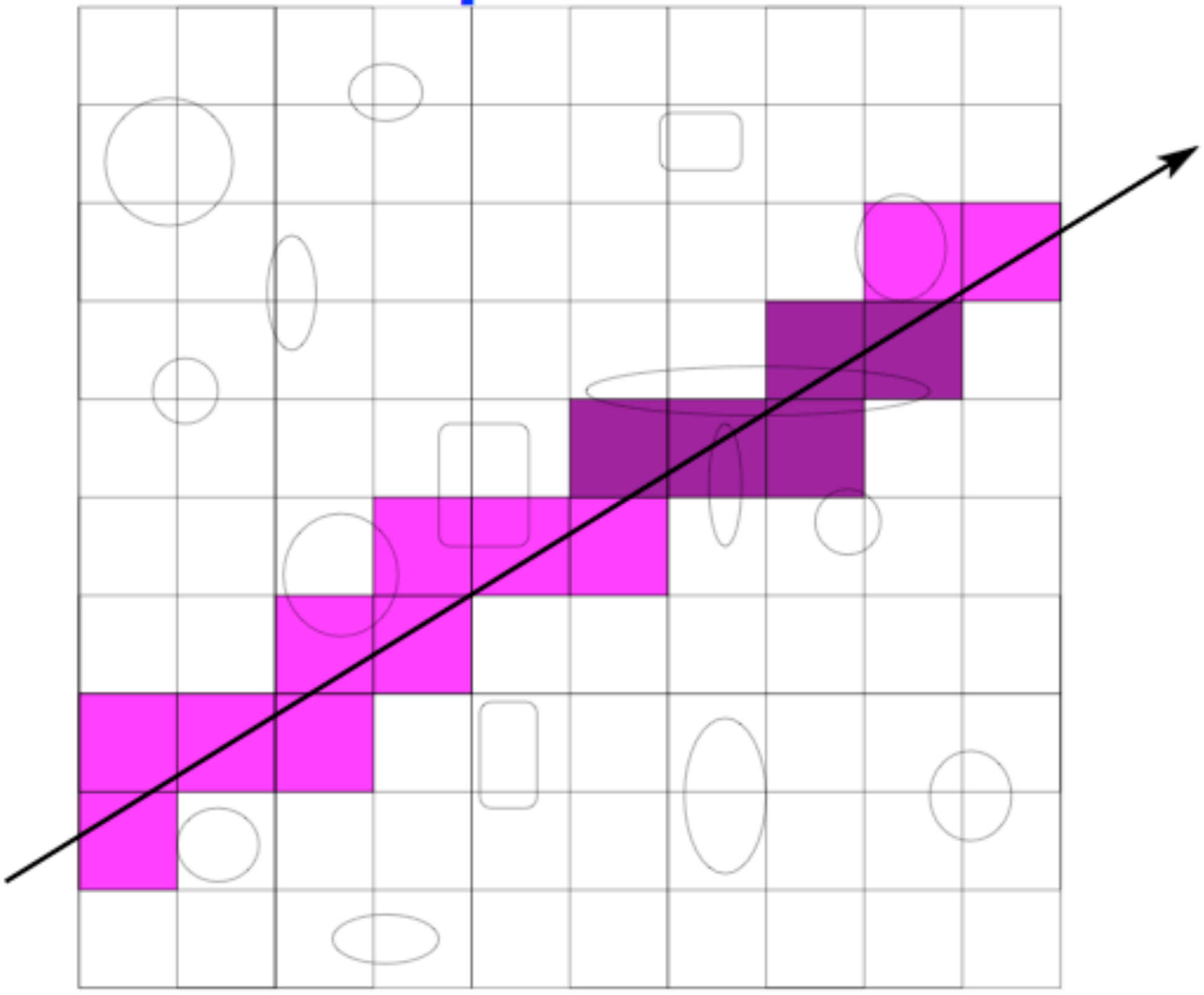
Process

- *Find smallest of dx, dy, dz*
- *increment that axis e.g: for dx, x:=x+1*
- *update value, e.g: dx := dx + ix*
- *check voxel x,y,z*
- *check for end of world*

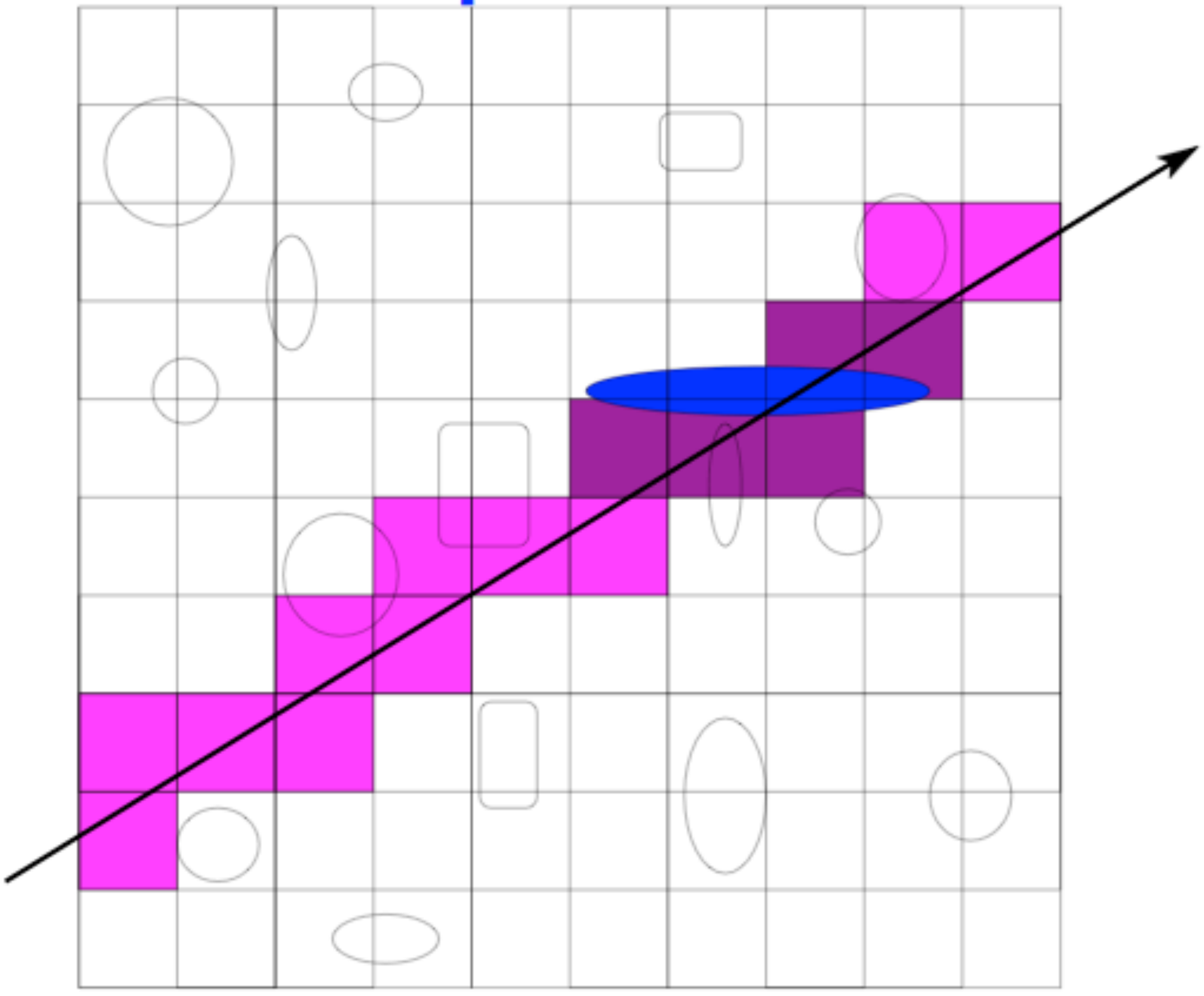
Uniform Space Subdivision



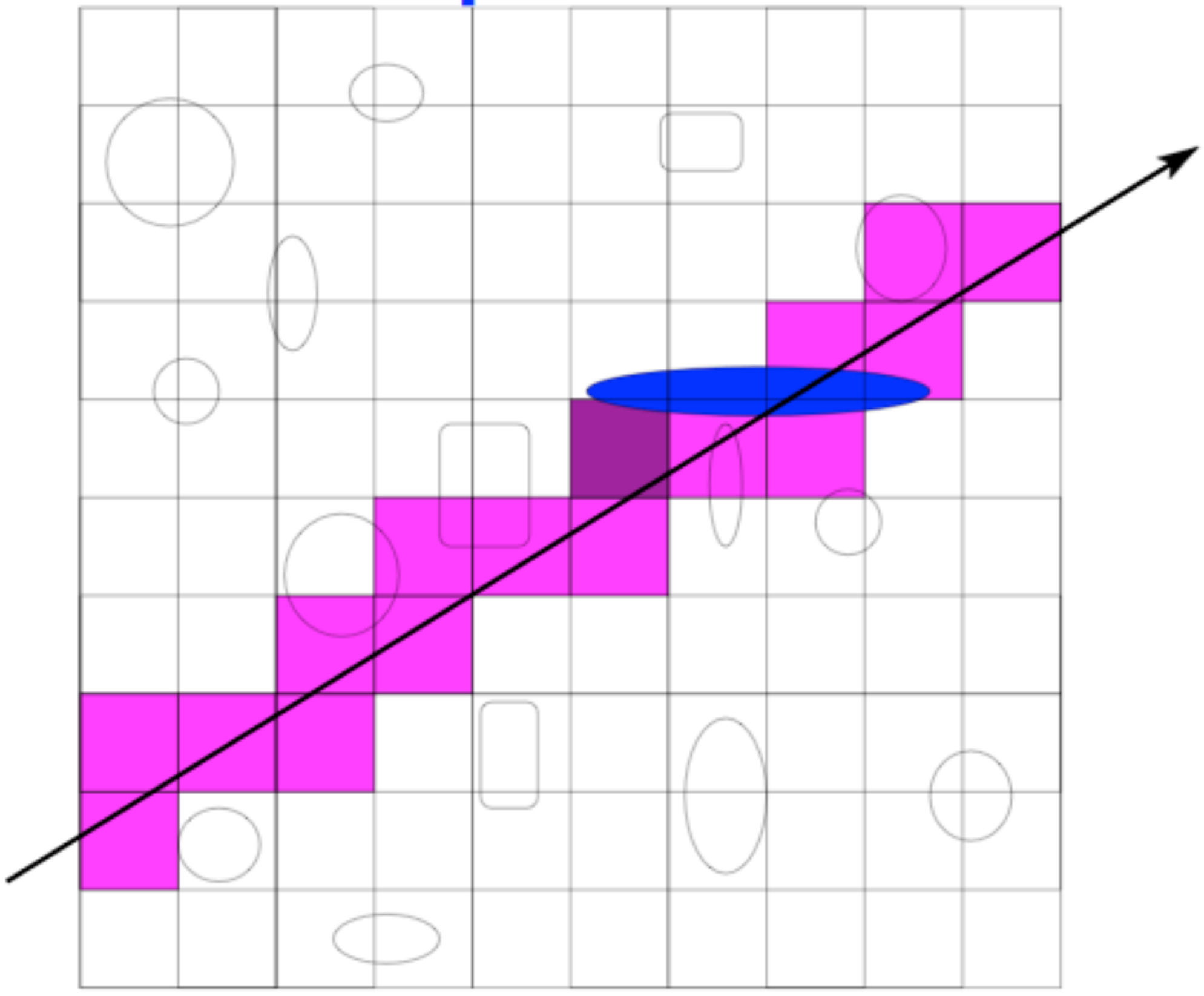
Uniform Space Subdivision



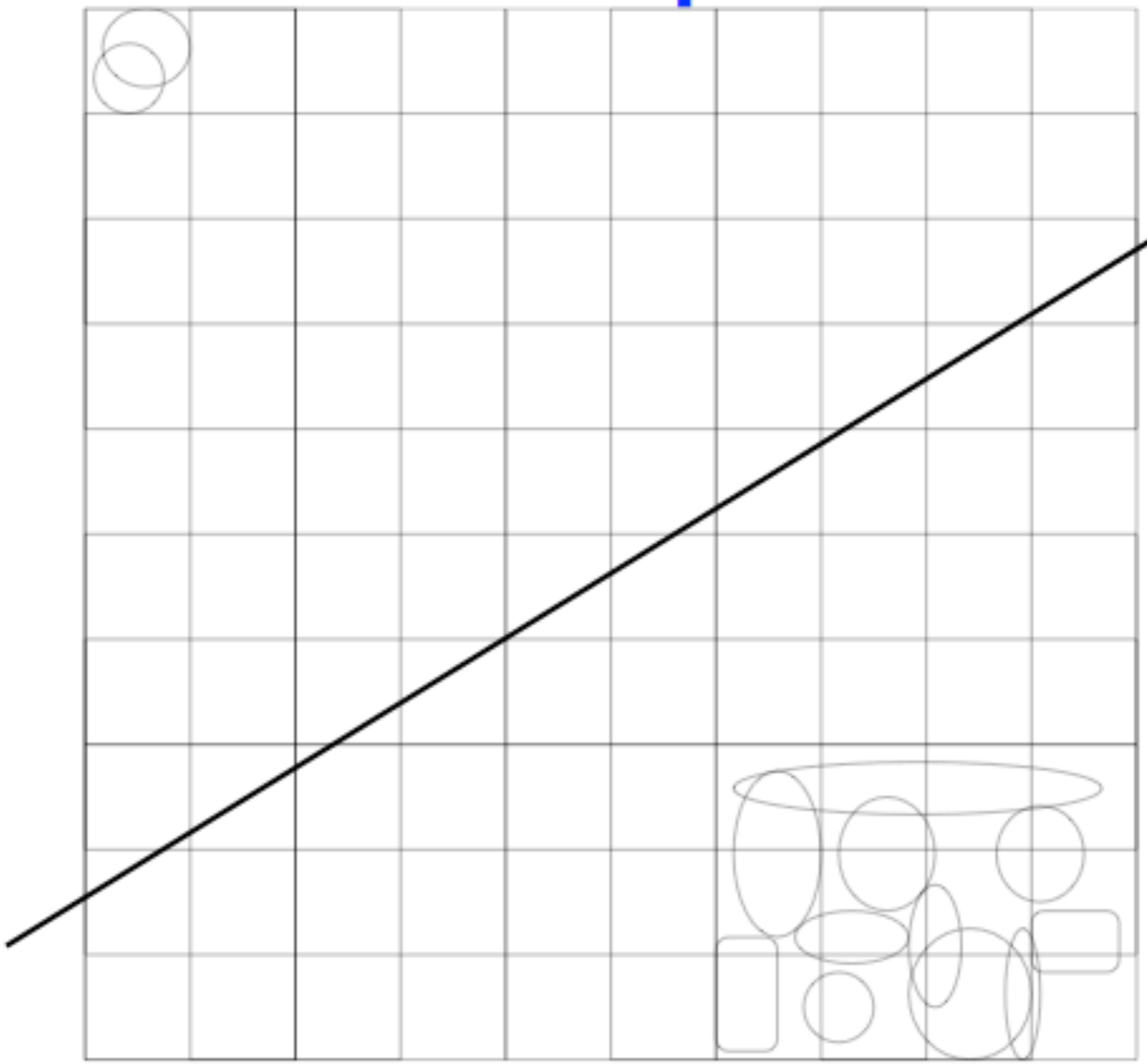
Uniform Space Subdivision



Uniform Space Subdivision



Uniform Space Subdivision

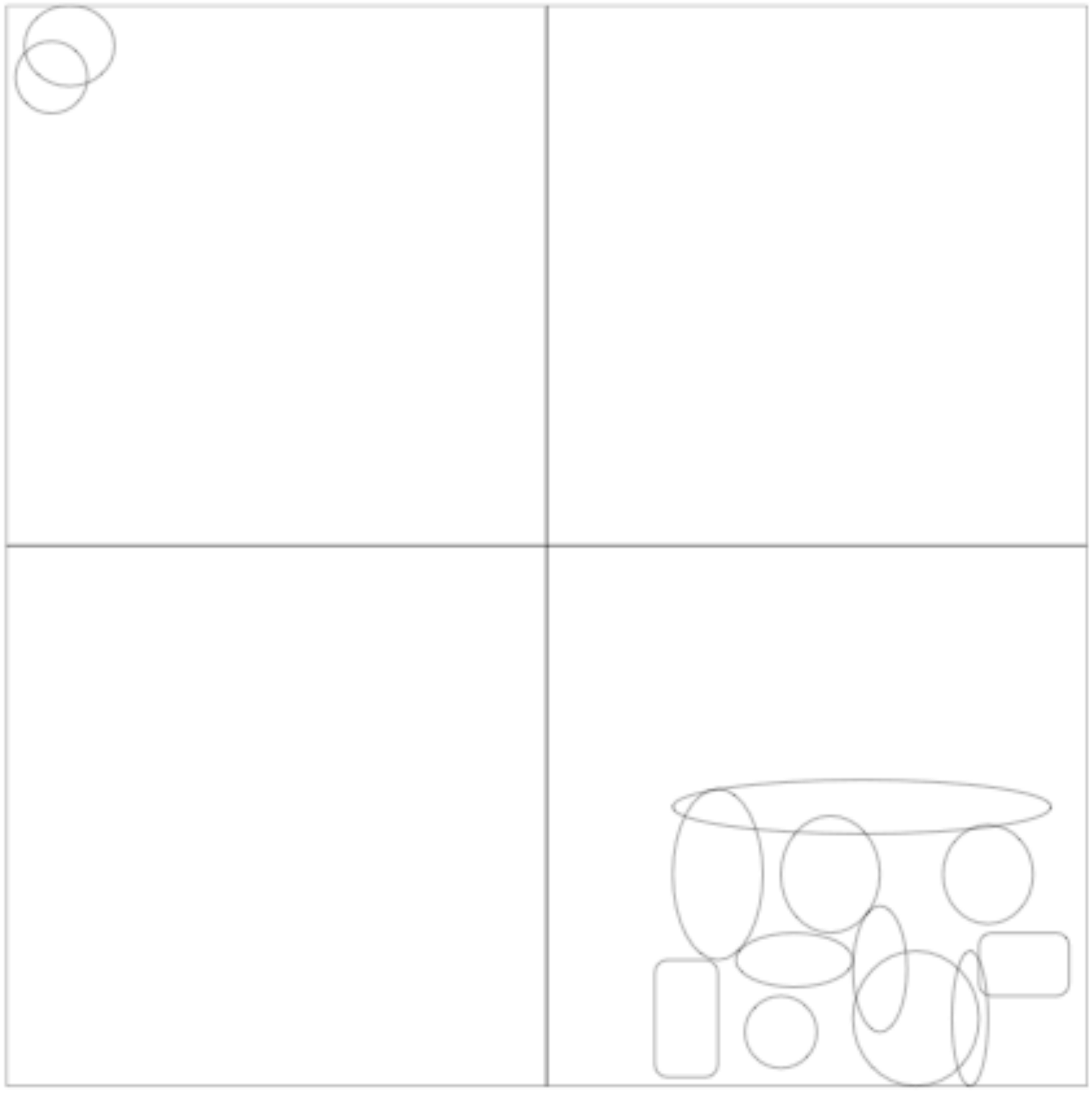


Why is this bad?

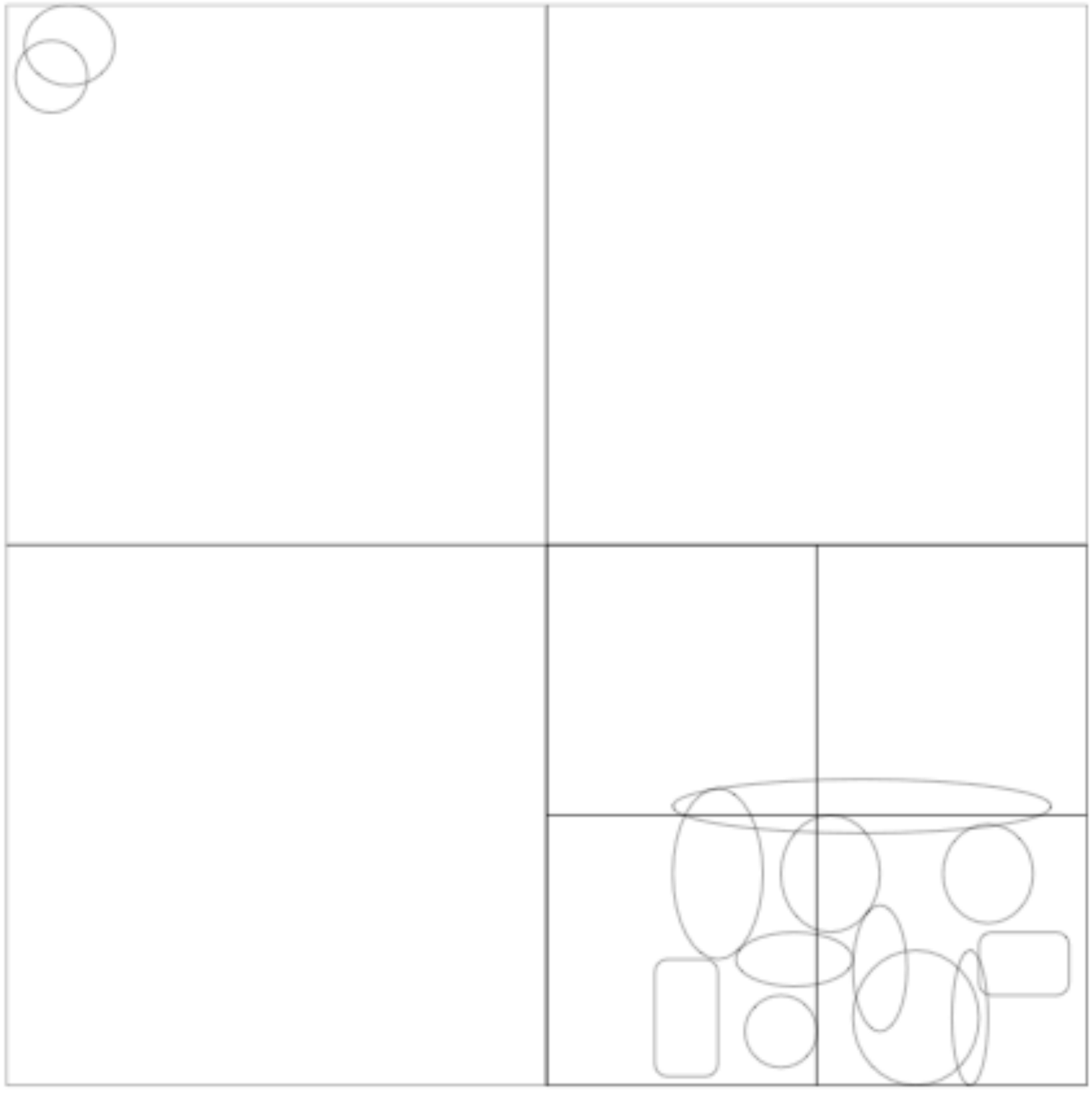
Adaptive Subdivision

- *Instead of lots of little empty cells, make empty cells as big as possible*
- *Use a tree structure to create a hierarchy of bounding cubes*
- *You will get fewer voxels*
- *Is there a down side?*
- *Octrees/BSP trees/kd-trees*

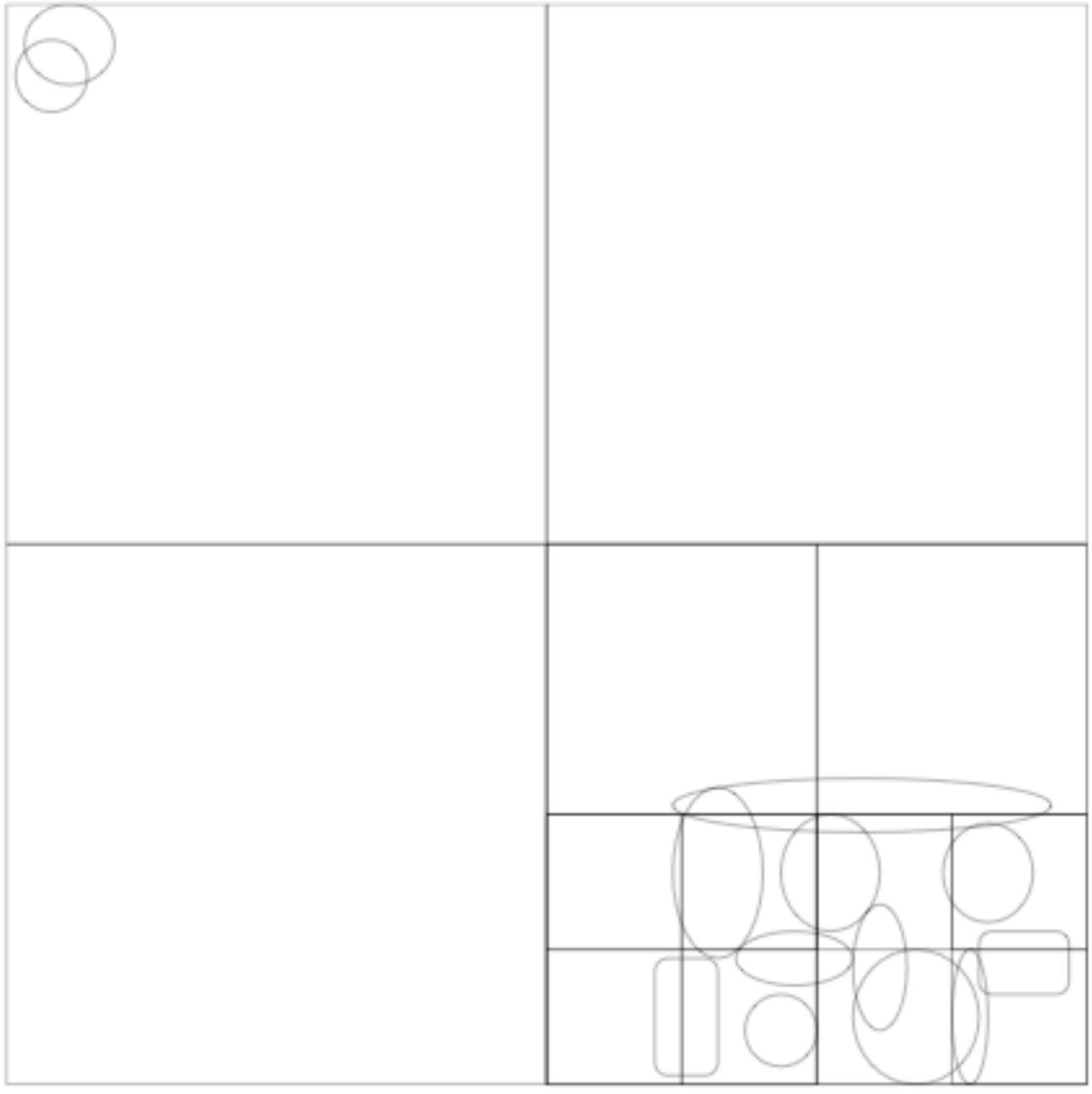
Quadtrees



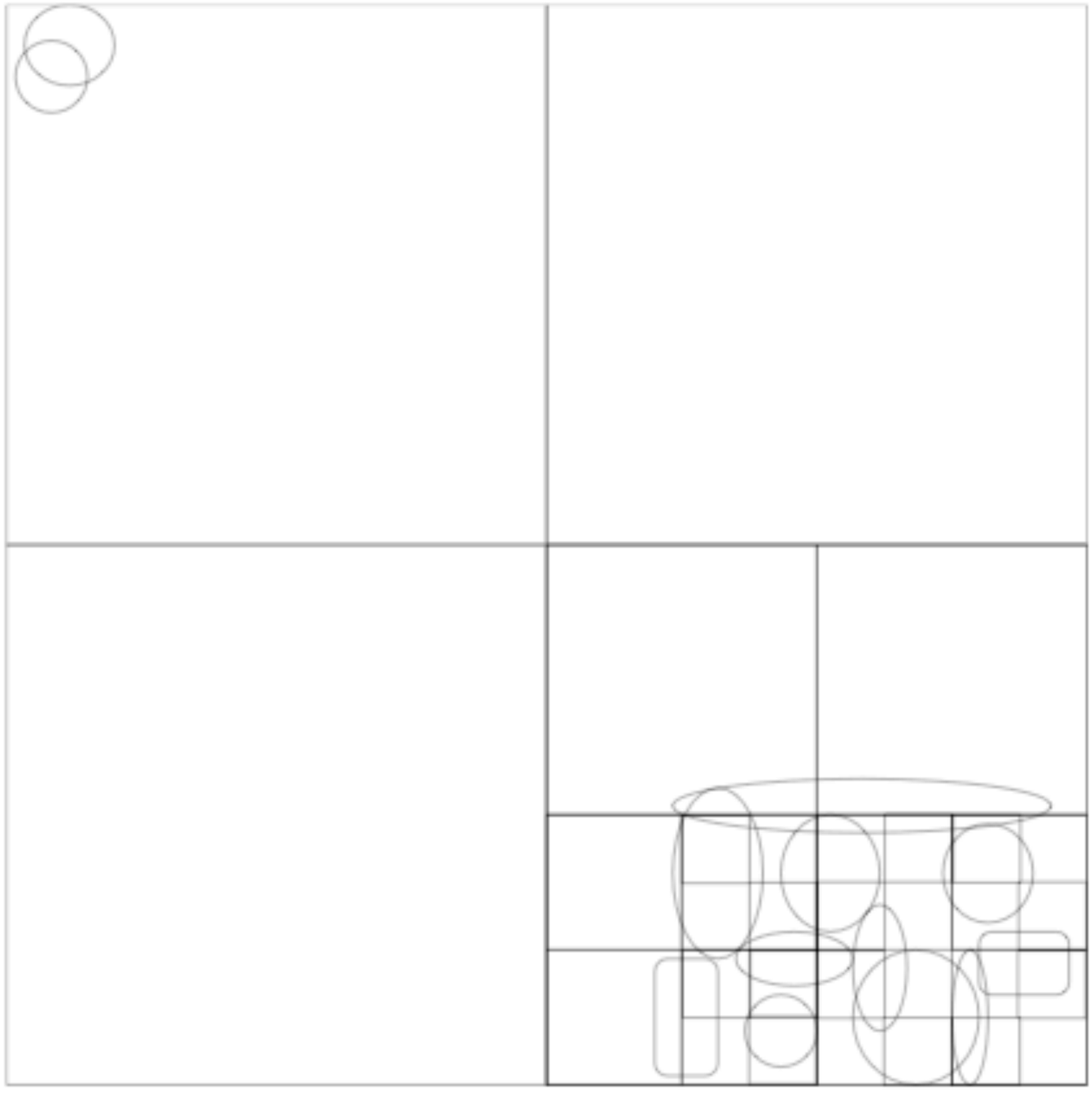
Quadtrees



Quadtrees



Quadtrees



Octrees

- *Divide until a cell has one object or is too small*
- *Facilitates raytracing CSG objects (later)*
- *But the cell-skipping algorithm is NOT obvious*
- *Info in "Ray Tracing News" archives*

Neat Tricks

- *Limit recursion depth by contribution made to pixel*
- *Keep a reference to the last object that caused a shadow*
- *Do inside/outside test on triangles before plane intersection*

Instead of:
(b - a) x (p - a).n, (c - b) x (p - b).n, (a - c) x (p - c).n
do
(b - a) x (u - a).v, (c - b) x (u - b).v, (a - c) x (u - c).v