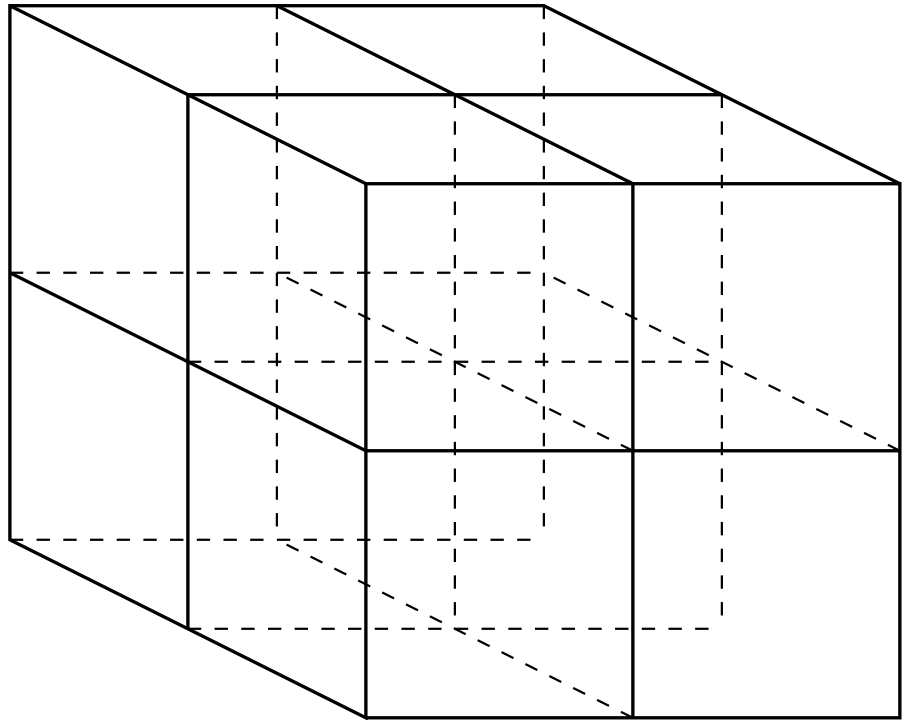
Ray Tracing Efficiency and Quality

COSC342

Lecture 20
16 May 2017

Ray Tracing Efficiency and Quality

- Efficient Ray Tracing
  - Space subdivision
  - Bounding boxes
  - Ray grouping
- Higher Quality
  - More rays
  - Super sampling



Ray Tracing is Expensive

One ray, one object is not too bad

- Compute intersections
- Determine colour

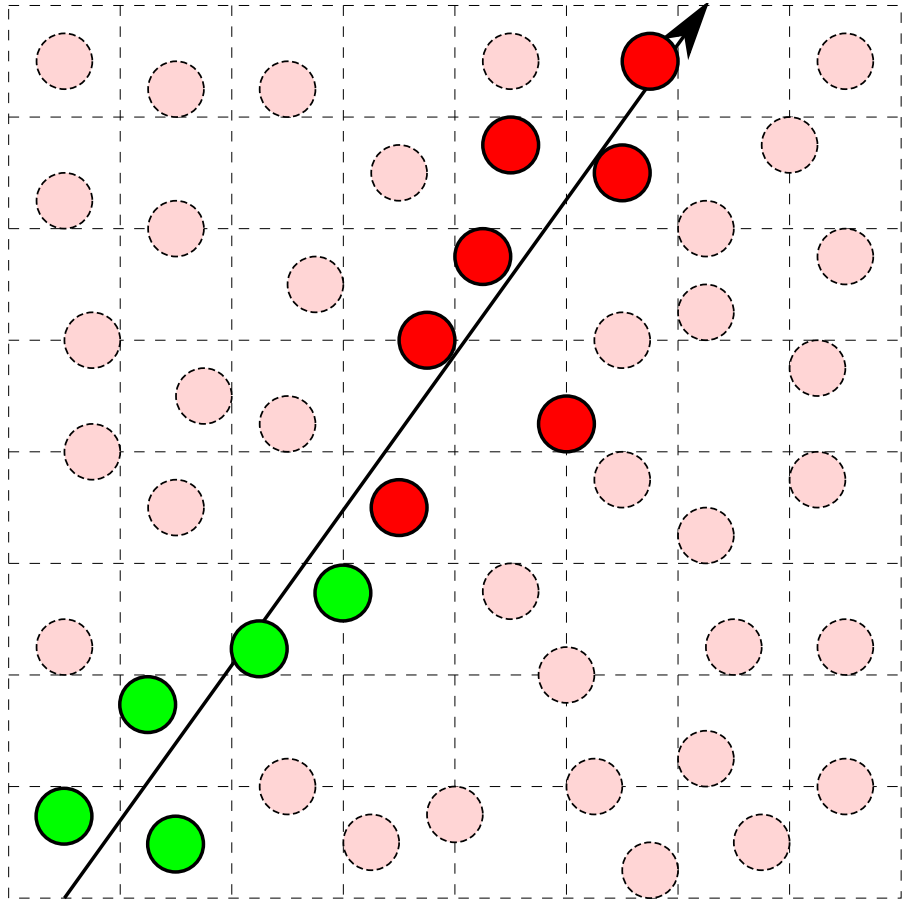
But we don't just do one ray

- One ray per pixel
- HD image: $\approx$ 2 million rays
- Intersections per ray per object
- 1000 objects: $\approx$ 2 billion hits
- Lighting, shadows, reflection, refraction, ...

Uniform Space Subdivision

Most rays miss most objects

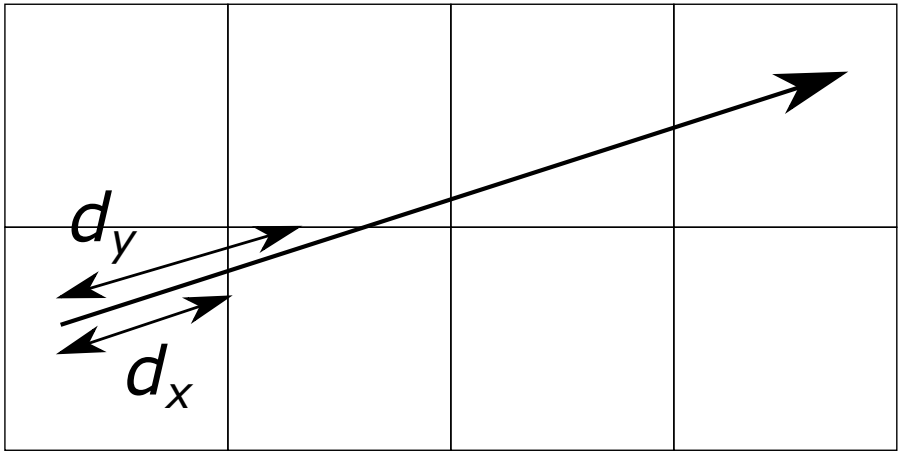
- Can we predict this?
- Divide space into a grid
- Objects lie in some cell(s)
- Cast ray through the grid
- Only check objects in the cells the ray passes through
- Roughly order objects on ray
- Stop at first hit.

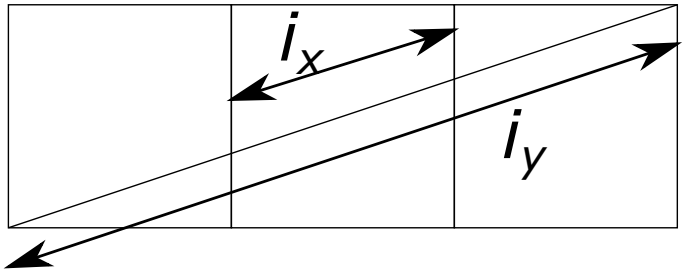


Cleary & Wyvill's Algorithm (1988)

What cells to check?

- d_x – distance to X boundary
- d_y – distance to Y boundary
- d_z – distance to Z boundary
- i_x – ray length across cell in X
- i_y – ray length across cell in X
- i_z – ray length across cell in X





Cleary & Wyvill's Algorithm

```
cell = (X, Y, Z)    // Starting point of the ray
while (inside the scene and no hit):
    dMin = min(dx, dy, dz)
    if (dMIn == dx)
        cell = (X+1, Y, Z)
        dx += ix
    else if (dMin == dy)
        cell = (X, Y+1, Z)
        dy += iy
    else
        cell = (X, Y, Z+1)
        dz += iz
    hit = intersect(ray, objects[cell])
```

Quadtrees and Octrees

Objects are not spread evenly

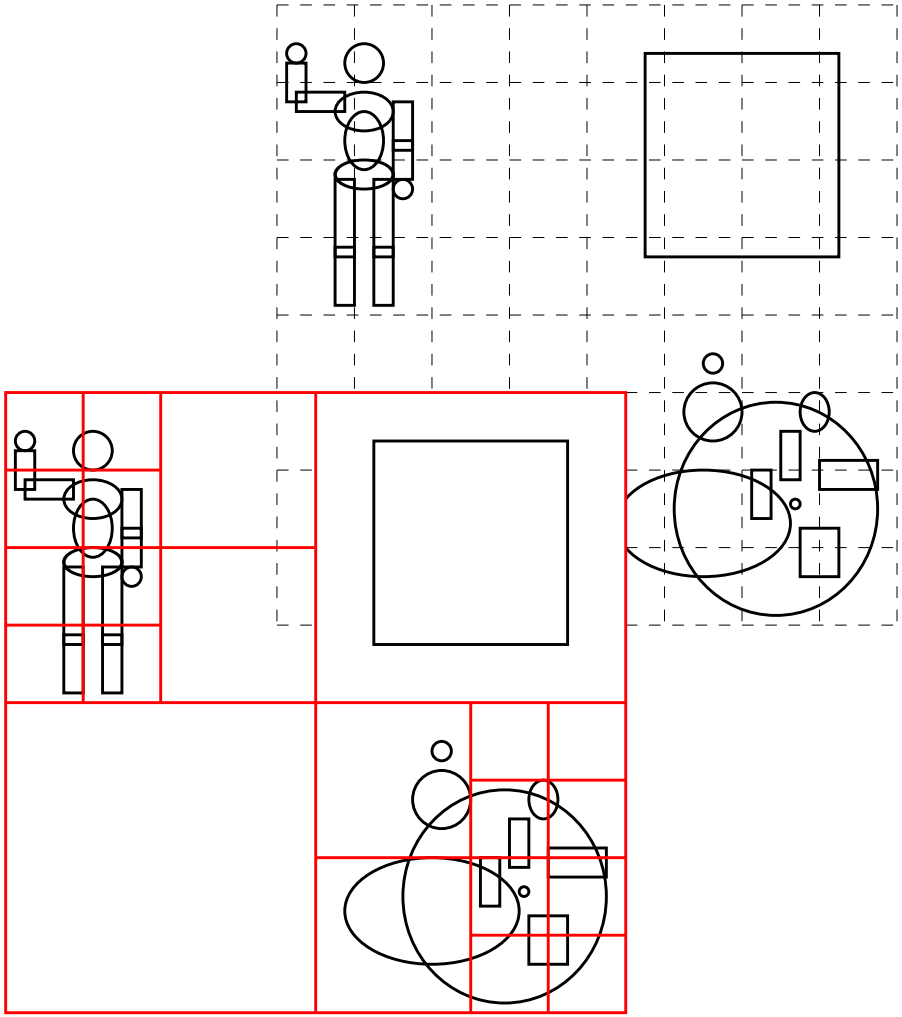
- Tend to cluster together
- Large objects cross many cells

Recursive subdivision:

- Divide in half on each axis
- Divide cells with many objects
- Repeat until some limit
- 2D gives quadtrees
- 3D gives octrees

Octree vs. uniform subdivision:

- Adapts to the scene
- Usually more efficient



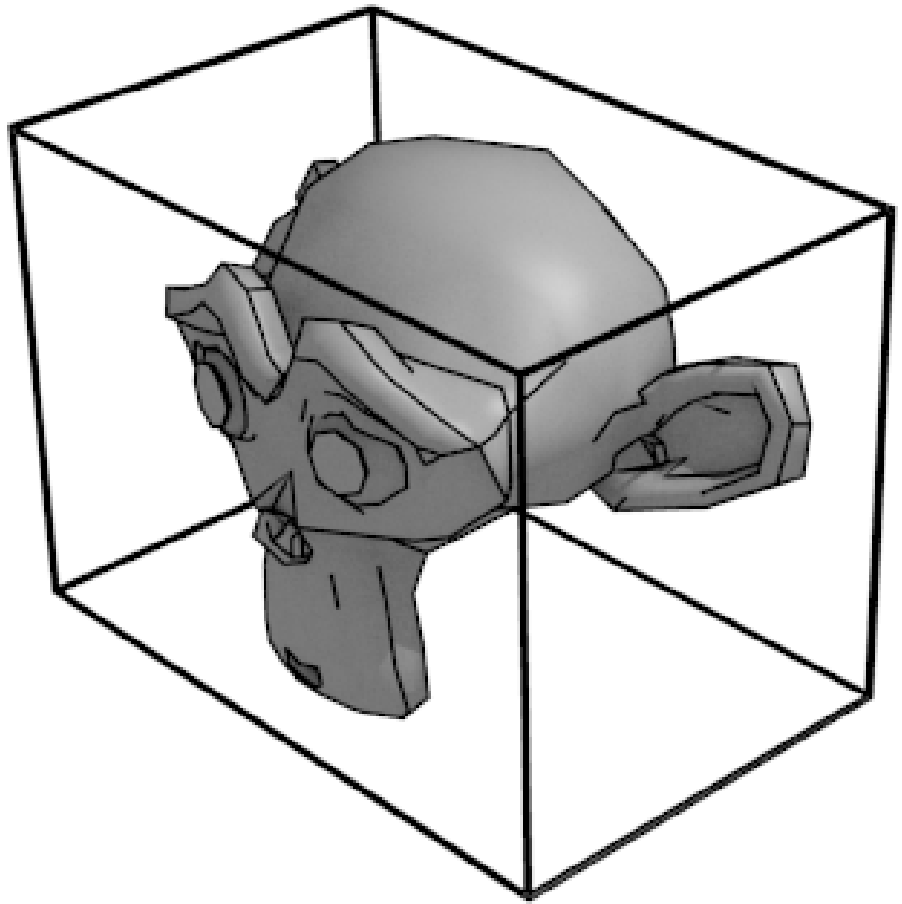
Bounding Boxes

Some ray-object checks are easy

- Spheres are simple
- Cubes, cylinders etc. are fine
- Triangle meshes etc. get hard

Most rays miss most objects

- Approximate complex objects
- Wrap them in a sphere/cube
- Intersect ray with wrapper
- If it misses, ignore the object
- Otherwise do full computation



Ray Grouping

Many rays are quite similar

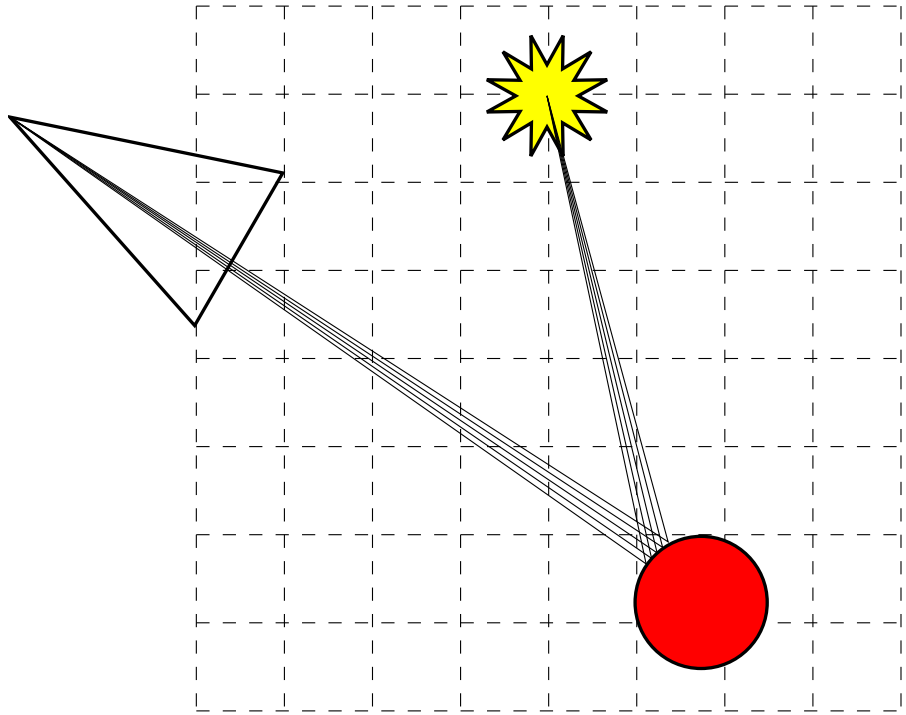
- Rays from neighbouring pixels
- Shadow rays from same area

They will have similar processing

- Traverse similar cells
- Intersect similar objectd

Efficient to do these together

- Better memory/cache usage
- Parallel execution (SIMD)



Antialiasing and Super Sampling

We can now afford more rays

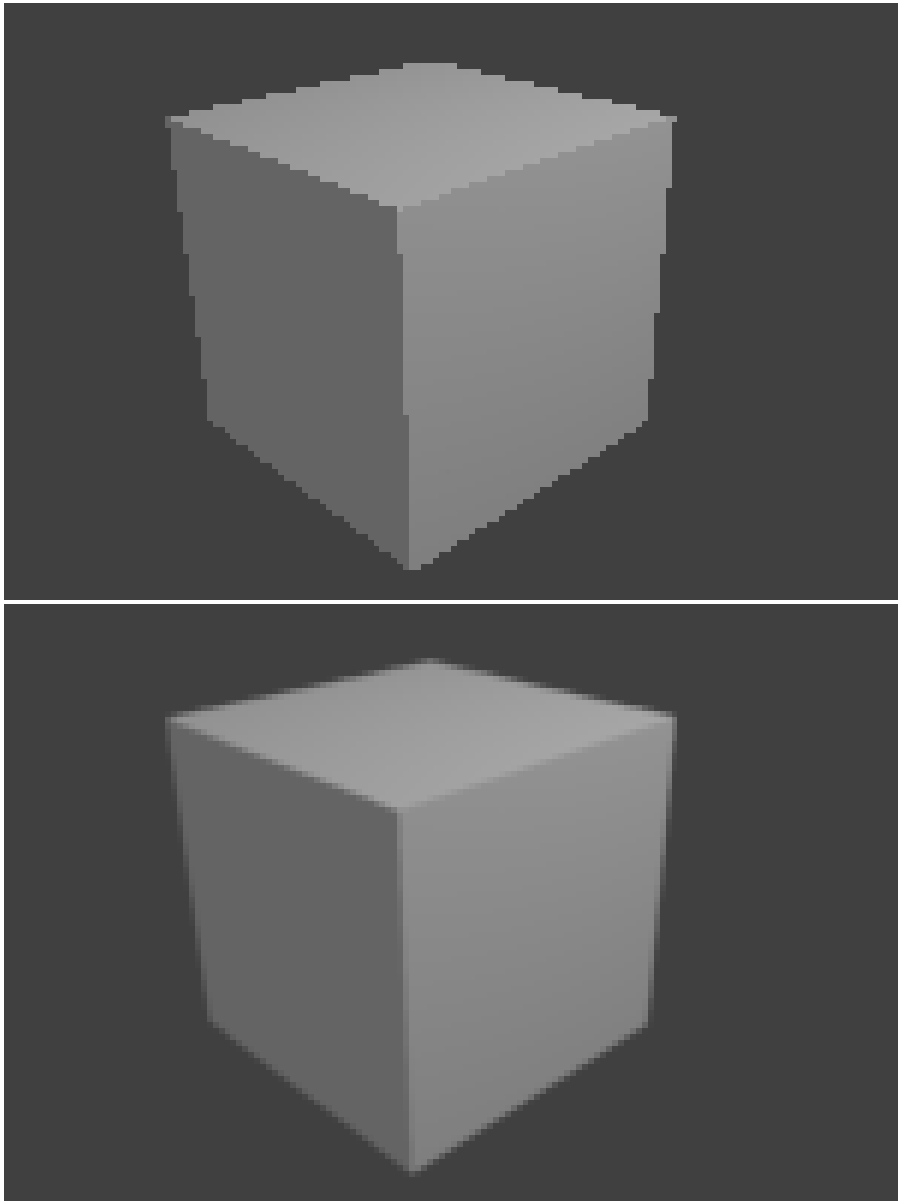
- More reflections, lights, objects, ...
- More rays per pixel
- This overcomes 'jaggies'

Aliasing and artefacts

- Ray either hits or doesn't
- Causes blocky edges
- These are artefacts of the pixel grid

Super sampling

- Cast several rays per pixel
- Average the results



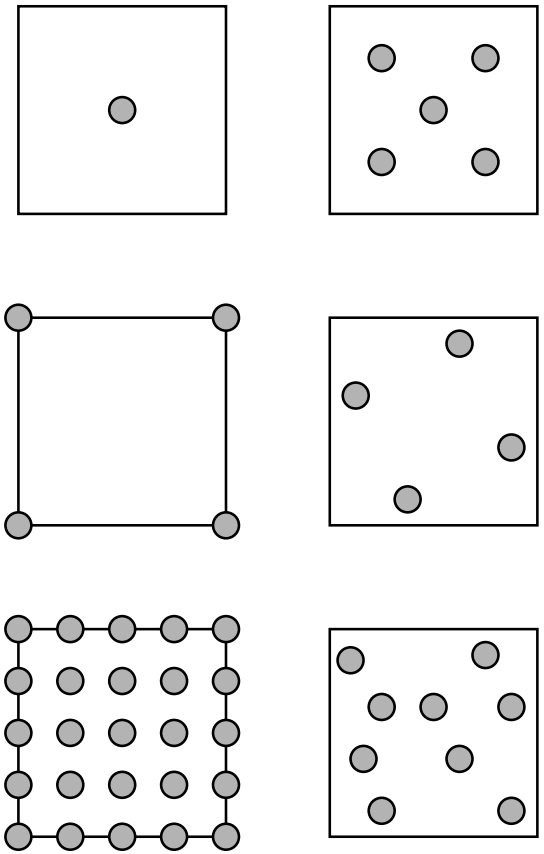
Sampling Methods

Where to cast the rays from?

- Inside pixels?
- At corners?
- Regular grids?
- Random sampling?

Jittering

- Regular sampling can cause artefacts and aliasing
- Random sampling can miss parts of the pixel
- Jittering – Random offsets from a regular grid



Adaptive Sampling

Cast rays where needed

- Cast rays at pixel corners
- If all are the same colour ...
  - Stop
- If not ...
  - Cast more rays
- Can do this recursively
- Weight colours by area