Graphics 2008/2009, period 1

Lecture 9

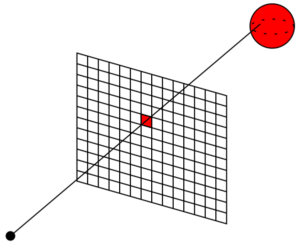
Ray tracing

Outline

Ray tracing / ray casting

Idea: for every pixel

- Compute ray from viewpoint through pixel center
- Determine first object hit by ray (including intersection point)
- Calculate shading for the pixel (possibly with recursion)



Triangles: barycentric coordinates

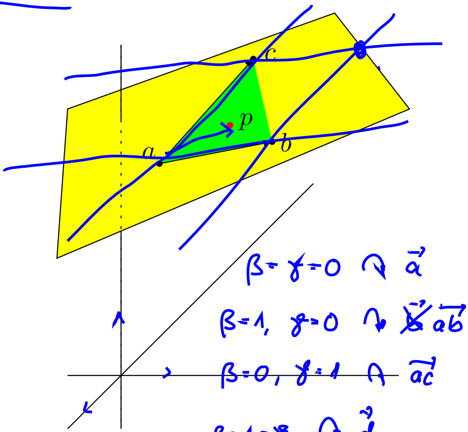
Recall that the plane V through the points a, b, and c can be written as

$$p = a + \beta(b - a) + \gamma(c - a).$$

Q: When does a point p in V lie in the triangle formed by a, b, and c? (incl. edges)

$\beta, \gamma \geq 0$

$\beta + \gamma \leq 1$



$\beta = \gamma = 0 \curvearrowright \vec{a}$

$\beta = 1,\ \gamma = 0 \curvearrowright \vec{ab}$

$\beta = 0,\ \gamma = 1 \curvearrowright \vec{ac}$

$\beta = 1 = \gamma \curvearrowright \vec{d}$

Rays: parametric representation

A ray starting in the point e with direction d can be written as $p = e + td$.

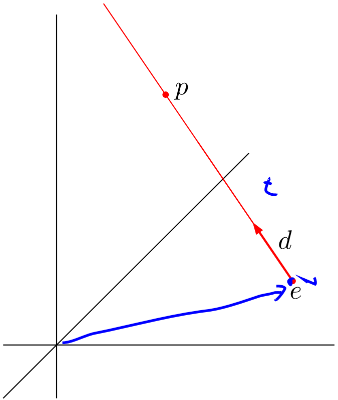
Q: are there any conditions on t?

$\vec{e}$

$t=0 \rightsquigarrow \vec{p}(0)=\vec{e}$

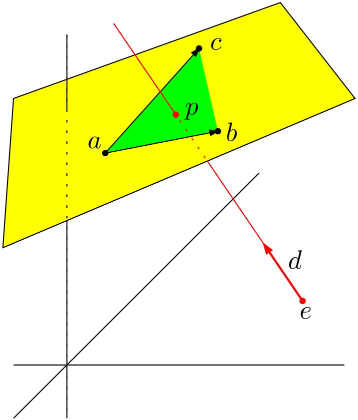
(cf. persp. proj.: far plane f)

$t \in [0,\infty)$ / $[0,f]$



Intersecting a ray and a triangle

If there is a unique intersection between a ray and a triangle, then the intersection point p satisfies both the plane equation and the ray equation, as well as the conditions on β, γ and t.



Intersecting a ray and a triangle

So we can write:

$$
\begin{aligned}
\overbrace{x_e + t x_d}^{\text{ray}} &= \overbrace{x_a + \beta(x_b - x_a) + \gamma(x_c - x_a)}^{\text{plane}} \\
y_e + t y_d &= y_a + \beta(y_b - y_a) + \gamma(y_c - y_a) \\
z_e + t z_d &= z_a + \beta(z_b - z_a) + \gamma(z_c - z_a)
\end{aligned}
$$

which can be rewritten as

$$-\beta(x_b - x_a) = (x_a - x_b)\cdot\beta$$

$$
\begin{aligned}
(x_a - x_b)\beta + (x_a - x_c)\gamma + x_d t &= x_a - x_e \\
(y_a - y_b)\beta + (y_a - y_c)\gamma + y_d t &= y_a - y_e \\
(z_a - z_b)\beta + (z_a - z_c)\gamma + z_d t &= z_a - z_e
\end{aligned}
$$

or as

$$
\begin{bmatrix} x_a - x_b & x_a - x_c & x_d \\ y_a - y_b & y_a - y_c & y_d \\ z_a - z_b & z_a - z_c & z_d \end{bmatrix}
\begin{bmatrix} \beta \\ \gamma \\ t \end{bmatrix}
=
\begin{bmatrix} x_a - x_e \\ y_a - y_e \\ z_a - z_e \end{bmatrix}
$$

Intersecting a ray and a triangle

Now, if we write

$$\begin{bmatrix} x_a - x_b & x_a - x_c & x_d \\ y_a - y_b & y_a - y_c & y_d \\ z_a - z_b & z_a - z_c & z_d \end{bmatrix} \begin{bmatrix} \beta \\ \gamma \\ t \end{bmatrix} = \begin{bmatrix} x_a - x_e \\ y_a - y_e \\ z_a - z_e \end{bmatrix}$$

as

$$A \begin{bmatrix} \beta \\ \gamma \\ t \end{bmatrix} = \begin{bmatrix} x_a - x_e \\ y_a - y_e \\ z_a - z_e \end{bmatrix}$$

then we see that

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{bmatrix} \beta \\ \gamma \\ t \end{bmatrix} = A^{-1} \begin{bmatrix} x_a - x_e \\ y_a - y_e \\ z_a - z_e \end{bmatrix}$$

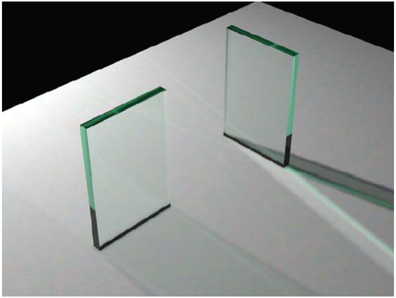
Outline

✓ 1. Ray/triangle intersection revisited
2. Refraction
3. Instancing
4. Constructive Solid Geometry
5. Faster ray tracing

Refraction

Light traveling from one transparent medium into another one is refracted.

glass, water, air, ...

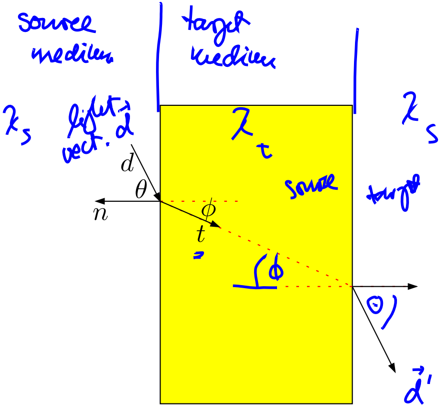


Snell's law

Angles before and after refraction are related as follows:

$$\boxed{\lambda_s \sin\theta = \lambda_t \sin\phi.}$$

where λ_s and λ_t are the refractive indices of the source and target media, respectively, and θ and ϕ the angles indicated in the image.



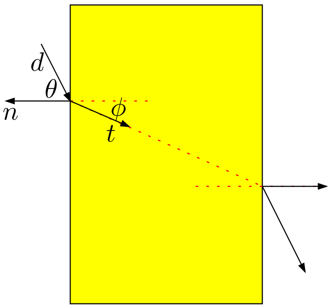
Getting rid of sines

$v \cdot w = \|v\| \cdot \|w\| \cdot \cos\Theta = v_1 w_1 + v_2 \cdot w_2 + v_3 w_3$

An equation that relates sines of the angles θ and ϕ is not as convenient as an equation that relates the cosines of the angles.

With the identity $\sin^2\phi + \cos^2\phi = 1$ we derive the following equation from Snell's law:

$$\cos^2\phi = 1 - \frac{\lambda_s^2(1-\cos^2\theta)}{\lambda_t^2}$$



Getting rid of sines

Snell's law:

$\lambda_s \sin\theta = \lambda_t \sin\phi$

$\curvearrowright \sin^2\phi = \left(\frac{\lambda_s}{\lambda_t} \cdot \sin\theta\right)^2$ [A]

$\sin^2\phi + \cos^2\phi = 1$

trigonometric identity

$\curvearrowright \sin^2\phi = 1 - \cos^2\phi$ [B]

$\cos^2\phi = 1 - \sin^2\phi$ [C]

[A] in [C]: $\cos^2\phi = 1 - \left(\frac{\lambda_s}{\lambda_t}\right)^2 \cdot \sin^2\theta$ (D)

(B) in (D): $\boxed{\cos^2\phi = 1 - \frac{\lambda_s^2(1-\cos^2\theta)}{\lambda_t^2}}$

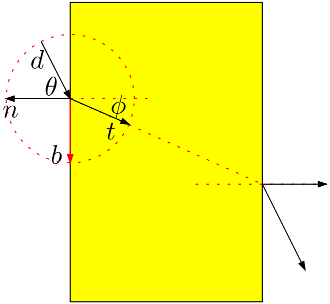
Constructing an orthonormal basis

How do we find the refracted vector t?

Assume the incoming vector d and the normal n are normalized. First, t lies in the plane spanned by d and n.

Next, we can set up an orthonormal basis in this plane by picking an appropriate vector b.

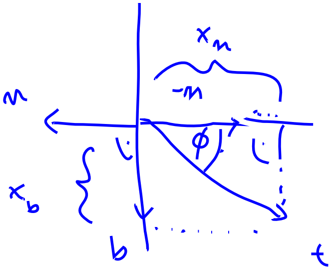
$\|\cdot\| = 1$

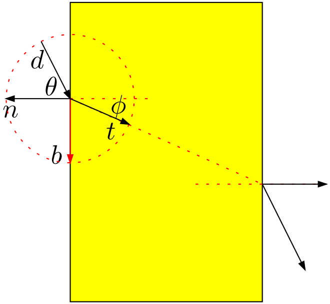


Finding the refraction vector

We have

$$\begin{aligned} t &= b \sin\phi - n \cos\phi \\ d &= b \sin\theta - n \cos\theta \end{aligned}$$





Finding the refraction vector

We have

$$\begin{aligned} t &= b \sin\phi - n\cos\phi \quad (A) \\ d &= b\sin\theta - n\cos\theta \end{aligned}$$

So we can solve for b:

$$b = \frac{d + n\cos\theta}{\sin\theta} \quad (B)$$

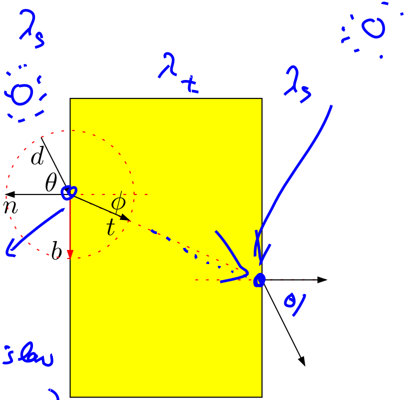
and for t: (B) in (A)

$$\begin{aligned} t &= \frac{\sin\phi(d+n\cos\theta)}{\sin\theta} - n\cos\phi \\ &= \frac{\lambda_s(d+n\cos\theta)}{\lambda_t} - n\cos\phi \\ &= \frac{\lambda_s(d-n(d\cdot n))}{\lambda_t} - n\sqrt{1-\frac{\lambda_s^2(1-(d\cdot n)^2)}{\lambda_t^2}} \end{aligned}$$

Snell's law

$\cos^2\phi$

≤ 0 total internal reflection



Outline

✓ 1. Ray/triangle intersection revisited

✓ 2. Refraction

3. Instancing

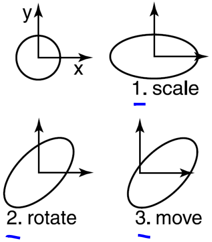
4. Constructive Solid Geometry

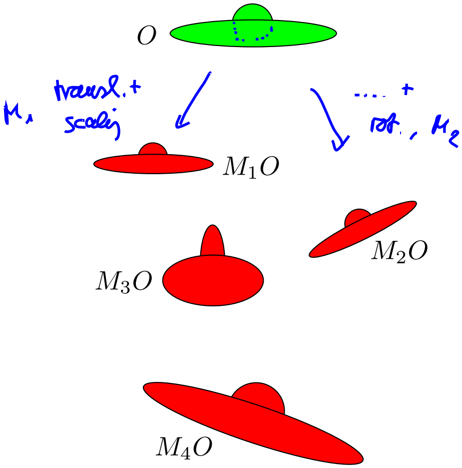
5. Faster ray tracing

Copying and transforming objects

- Instancing is an elegant technique to place various transformed copies of an object in a scene.

Expl.: circle → elipse

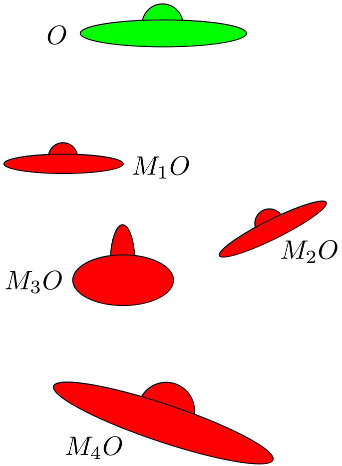




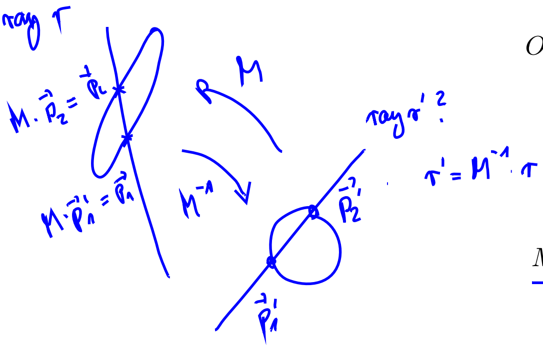
Copying and transforming objects

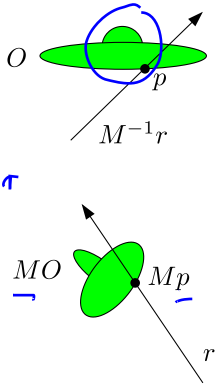
Instead of making actual copies, we simply store a reference to a base object, together with a transformation matrix.

intersections are preserved under basic transformations



Ray/instance intersection



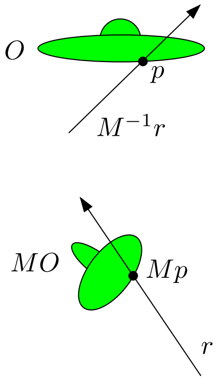


Ray/instance intersection

To determine the intersection q of a ray r with an instance MO, we first compute the intersection p of the inverse transformed ray $M^{-1}r$ and the original object O.

The point q is then simply Mp.

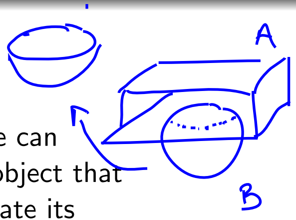
This way, complicated intersection tests (e.g. ray/ellipsoid) can often be replaced by simpler tests (ray/sphere).



Outline

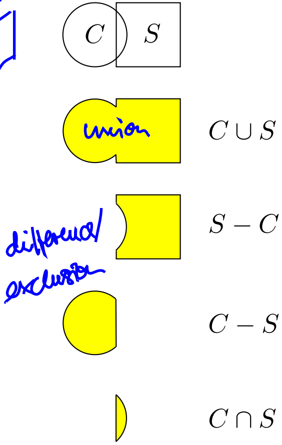
Constructive Solid Geometry

B - A



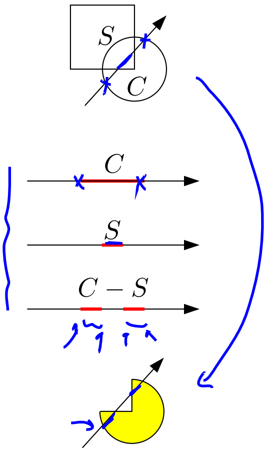
For ray tracing, we can basically use any object that allows us to calculate its intersection with a 3D line.

Using Constructive Solid Geometry (CSG) we can build complex objects from simple ones with set operations.



Intersections and CSG

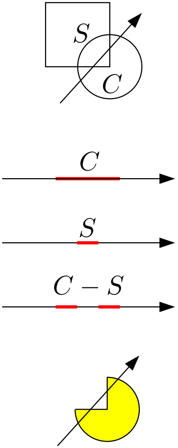
Big advantage: instead of actually constructing the objects, we can calculate ray-object intersections with the original objects and perform set operations on the resulting intervals.



Intersections and CSG

For every base object, we maintain an interval (or set of intervals) representing the part of the ray inside the object.

The intervals for combined objects are computed with the same set operations that are applied to the base objects.



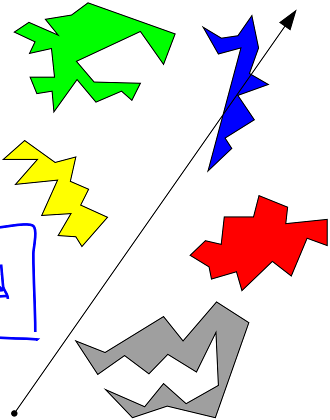
Outline

✓ 1. Ray/triangle intersection revisited
✓ 2. Refraction
✓ 3. Instancing
✓ 4. Constructive Solid Geometry
→ 5. Faster ray tracing

The bottleneck in ray tracing

Our algorithm for testing ray/object intersections is more or less the same as that of [Whitted, 1980]: test all objects, and report the nearest one.

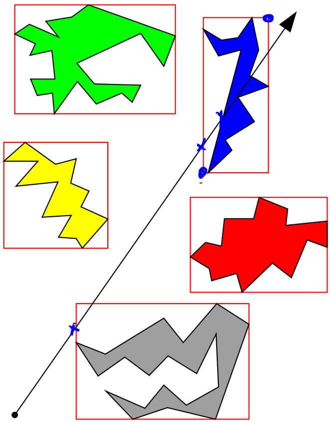
It is estimated that 75% to 95% of the time in ray/tracing is spent on ray/object intersections [Chang, 2001].



Bounding boxes

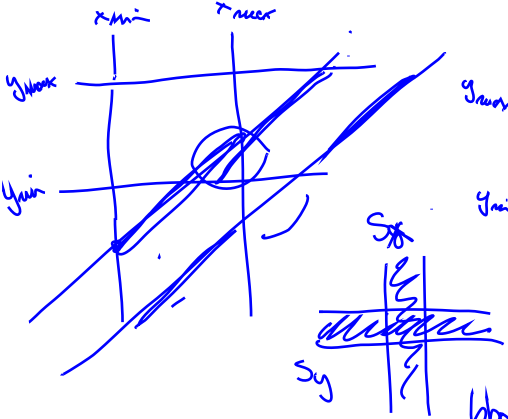
A common technique to improve ray/object intersection query times is the use of bounding boxes.

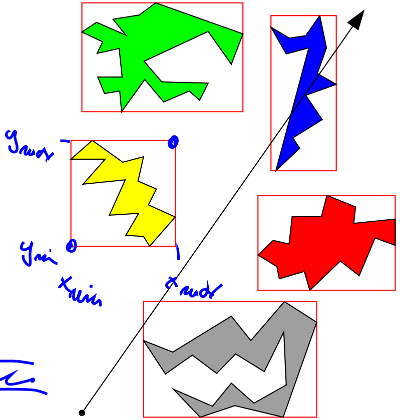
One advantage:
We don't need the actual intersection point but just a yes or no answer to the intersection test.



Bounding boxes

How do we get that?

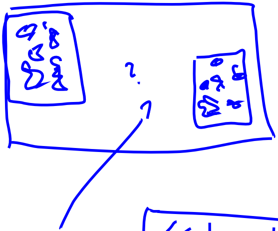


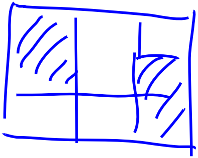


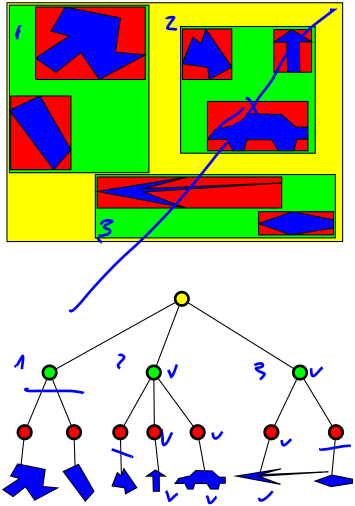
bbox : $S_x \cap S_y$

Hierarchical bounding boxes

But why stop with bounding objects? We can also bound groups of bounding boxes, and build a hierarchy.



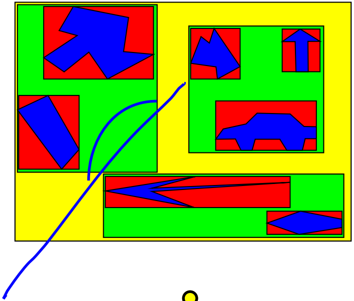


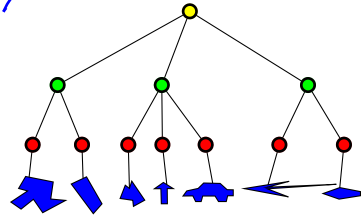


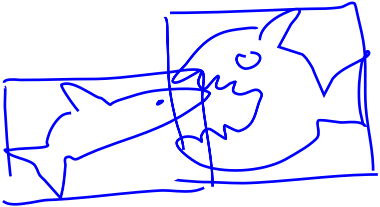
Hierarchical bounding boxes

But why stop with bounding objects? We can also bound groups of bounding boxes, and build a hierarchy.

In practice, the choice of what items to group is a hard problem.

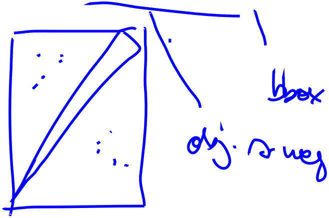


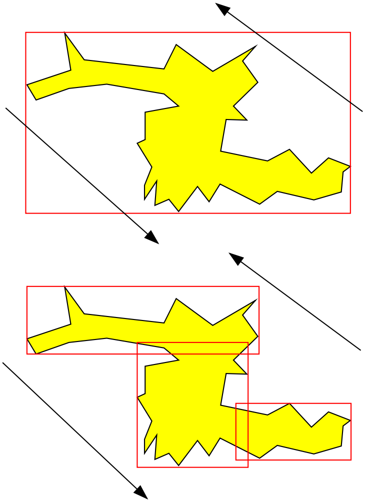




Current research

Packing an object in more than one box makes the ray/object test more expensive if there is a hit, but may drastically reduce false positives.



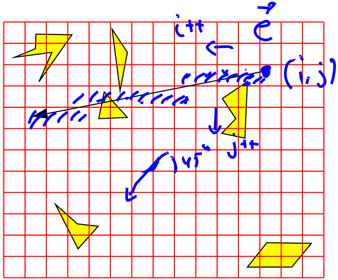


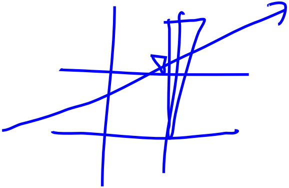
Uniform spatial subdivision

An alternative method for intersection test speed-up is to put a regular grid over the object space, and to traverse from cell to cell.

Q: Can we stop if we encounter a cell that contains a hit object?

Q: What's the best grid size? And do we need a regular grid?



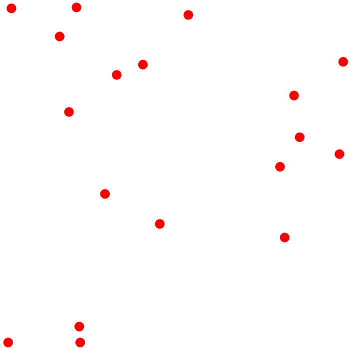


Octrees

One way to get a grid that somehow resembles the distribution of objects: Octrees

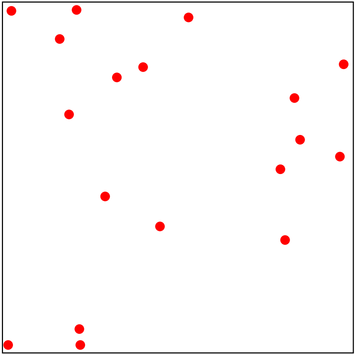
An octree is the 3D version of the Quadtree.

2D



Octrees

The idea is as follows: given a set of objects, we first compute an axis-parallel bounding box that contains all of them.

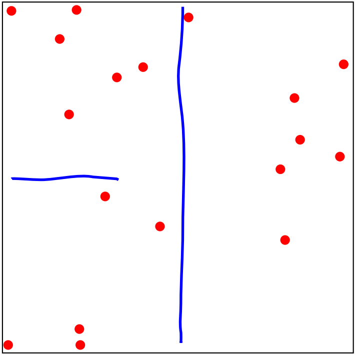


Octrees

2

Next, if the box contains more than a predetermined number of objects, we split it evenly along all dimensions.

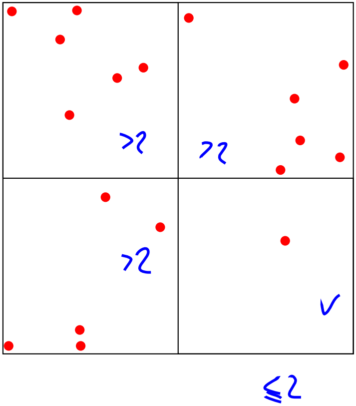
We continue until the condition on the maximal number of objects in a node is satisfied



Octrees

Next, if the box contains more than a predetermined number of objects, we split it evenly along all dimensions.

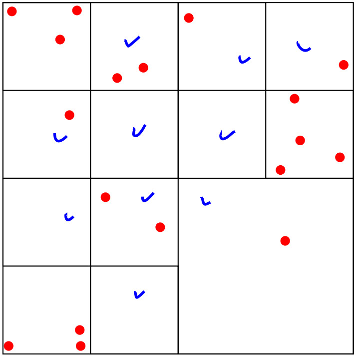
We continue until the condition on the maximal number of objects in a node is satisfied



Octrees

Next, if the box contains more than a predetermined number of objects, we split it evenly along all dimensions.

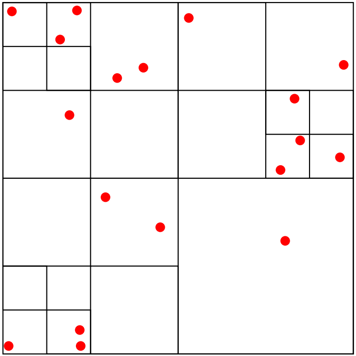
We continue until the condition on the maximal number of objects in a node is satisfied



Octrees

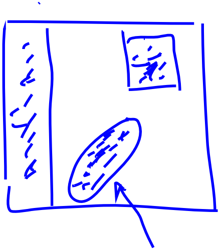
Next, if the box contains more than a predetermined number of objects, we split it evenly along all dimensions.

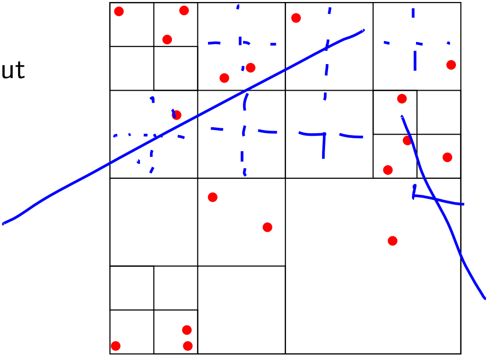
We continue until the condition on the maximal number of objects in a node is satisfied



Octrees

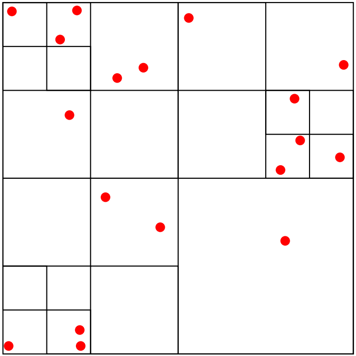
Traversal of the nodes is similar to the traversal in uniform spatial subdivision–but somewhat more complicated.





Octrees

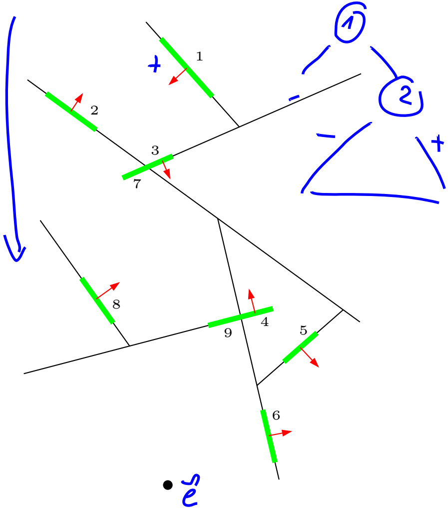
Instead of splitting evenly, we could also do balanced splits, based on the object distribution.



BSP trees

We have seen BSP trees before. Apart from speeding up projective rendering, they can also be used for ray tracing.

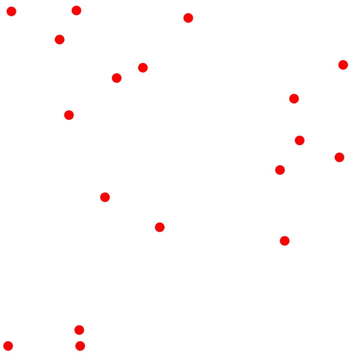
However, in ray tracing, we do not only deal with triangles, so finding splitting planes is a bit more complicated.



BSP trees

We try to find a splitting plane that splits the objects into two groups of more or less equal size

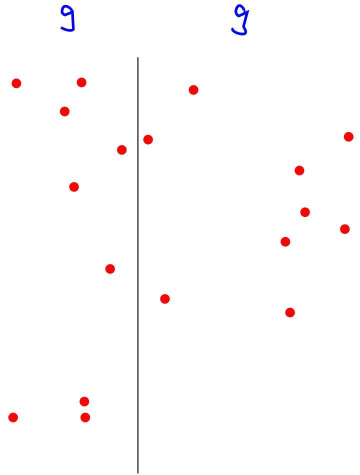
In practice, we limit ourselves to axis-parallel splitting planes.



BSP trees

We try to find a splitting plane that splits the objects into two groups of more or less equal size

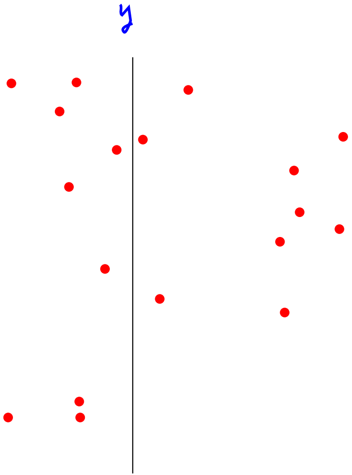
In practice, we limit ourselves to axis-parallel splitting planes.



BSP trees

We go into recursion on the two groups, continuing until every group has at most a predetermined number of objects.

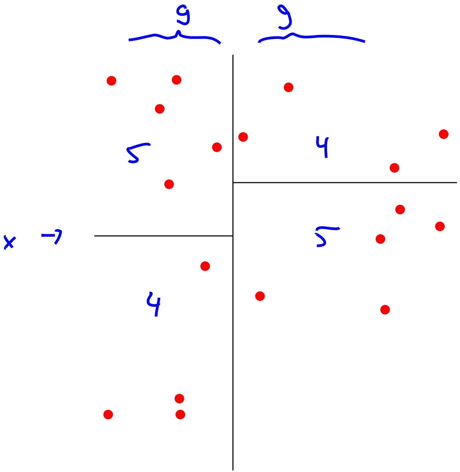
Usually, we split along alternating dimensions.



BSP trees

We go into recursion on the two groups, continuing until every group has at most a predetermined number of objects.

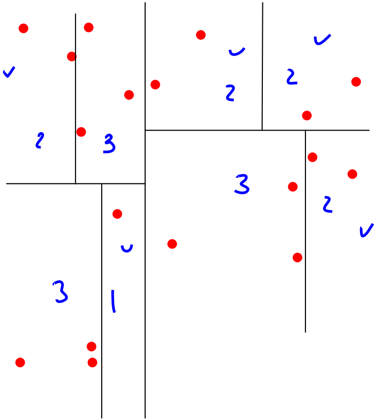
Usually, we split along alternating dimensions.



BSP trees

We go into recursion on the two groups, continuing until every group has at most a predetermined number of objects.

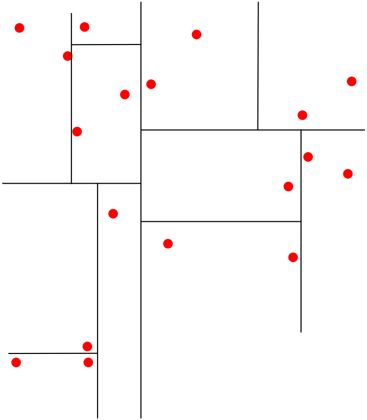
Usually, we split along alternating dimensions.



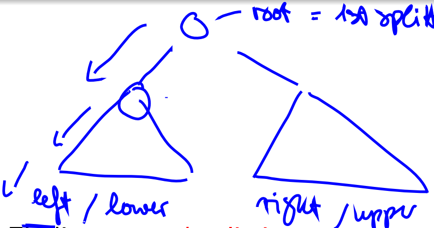
BSP trees

We go into recursion on the two groups, continuing until every group has at most a predetermined number of objects.

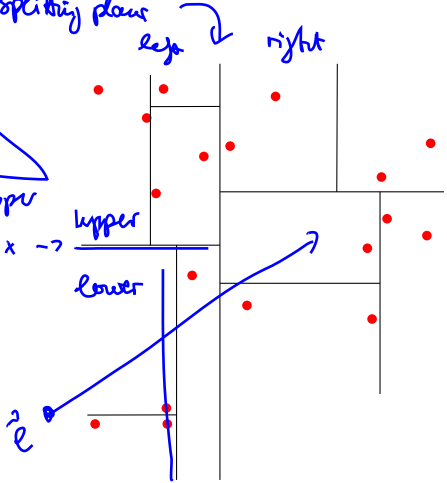
Usually, we split along alternating dimensions.



BSP trees



Finding a good splitting strategy is hard, and topic of ongoing research.

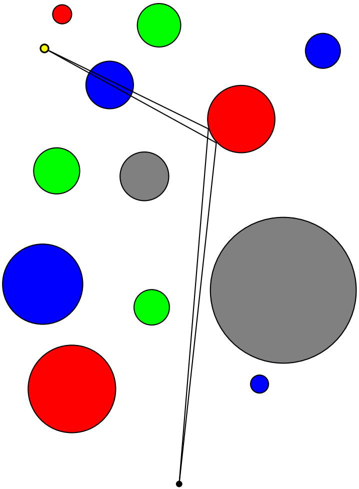


Ray coherence

Q: what distinguishes shadow feelers from other rays?

Can we exploit coherence for shadow feelers? YES

Can we exploit coherence for other rays too? NO



Outline

1. Ray/triangle intersection revisited
2. Refraction
3. Instancing
4. Constructive Solid Geometry
5. Faster ray tracing

More ray tracing ...

... in the 2nd programming assignment

- **1st programming assignment**:
  - due today, Oct 7, 18h
  - results will be online (most likely) by the end of next week
- **2nd programming assignment**:
  - online now
  - due Oct 30st, 18h