Speeding up Ray-tracing

CGG MFF UK Praha

pepca@cgg.mff.cuni.cz

http://cgg.mff.cuni.cz/~pepca/

Ray–scene intersection

- takes **most of the CPU time** (Whitted: up to 95%)
- scene composed of **elementary solids**
  – sphere, box, cylinder, cone, triangle, polyhedron, ..
  – primitive solids in CSG
  – number of elementary solids .. **N**
- naïve algorithm tests **every ray** (up to the proper recursion depth **H**) against **every elementary solid**
  – **O(N)** tests for one ray

Classification

❶ **faster "ray × scene"**

➡ **faster "ray × solid"** test

» bounding solids with efficient intersection algorithms

➡ **less "ray × solid"** tests

» bounding volume hierarchy, space subdivision (spatial data structures), directional techniques (+2D data structures)

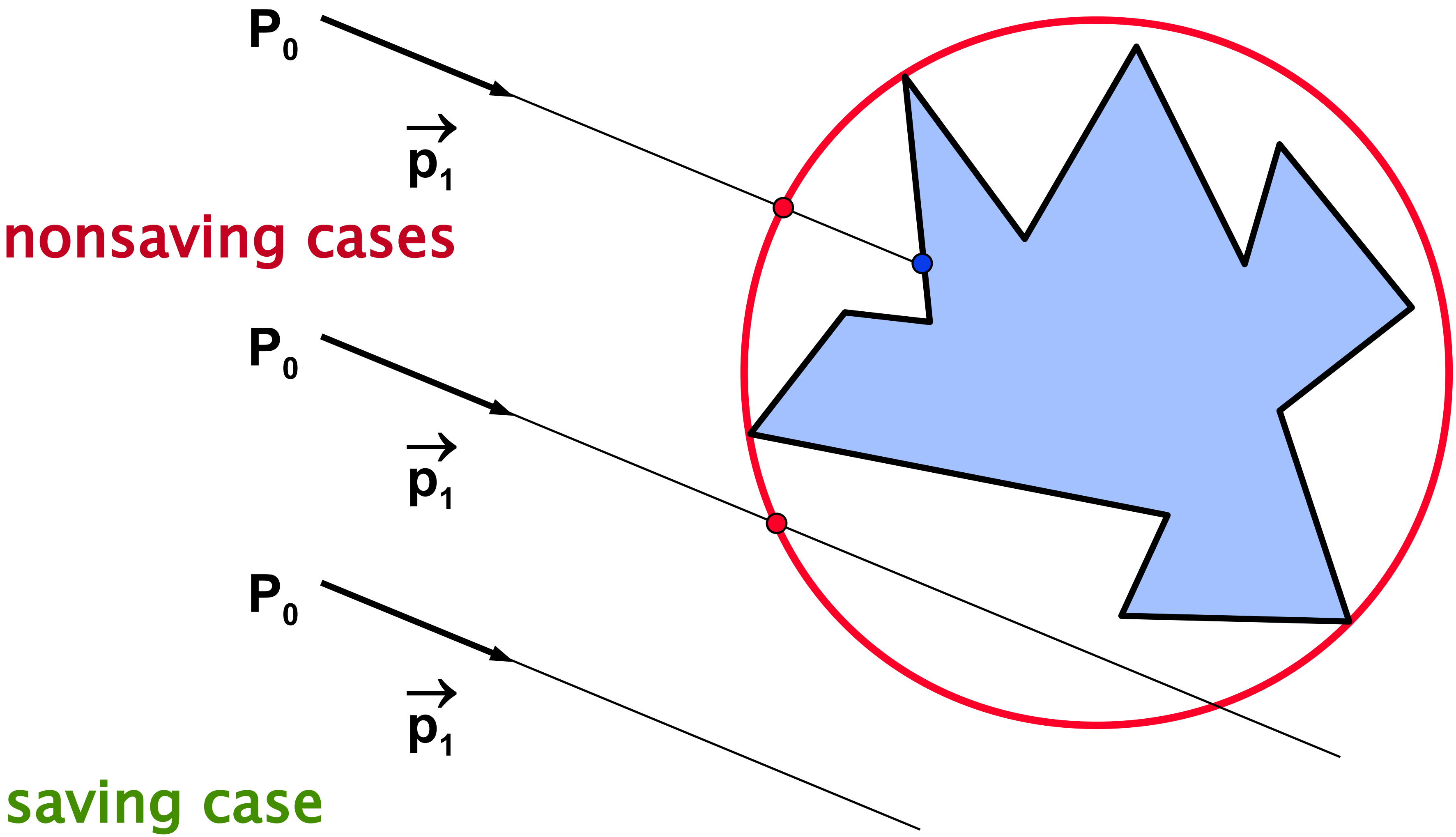
❷ **less rays**

» dynamic recursion control, adaptive anti-aliasing

❸ **generalized rays** (carrying more information)

» polygonal ray bundle, ray cone, ..

Bounding solid



Bounding solid

- ❶ **intersection is** [much] **faster** than with an original object
  – sphere, box (axis-aligned “AABB” or arbitrary orientation “OBB”), intersection of strips, ..
- ❷ a bounding solid should enclose an original object **as tight as possible**
- **eficiency** of a bounding solid .. middle ground between ❶ and ❷
  – total asymptotic complexity is still **O(N)**

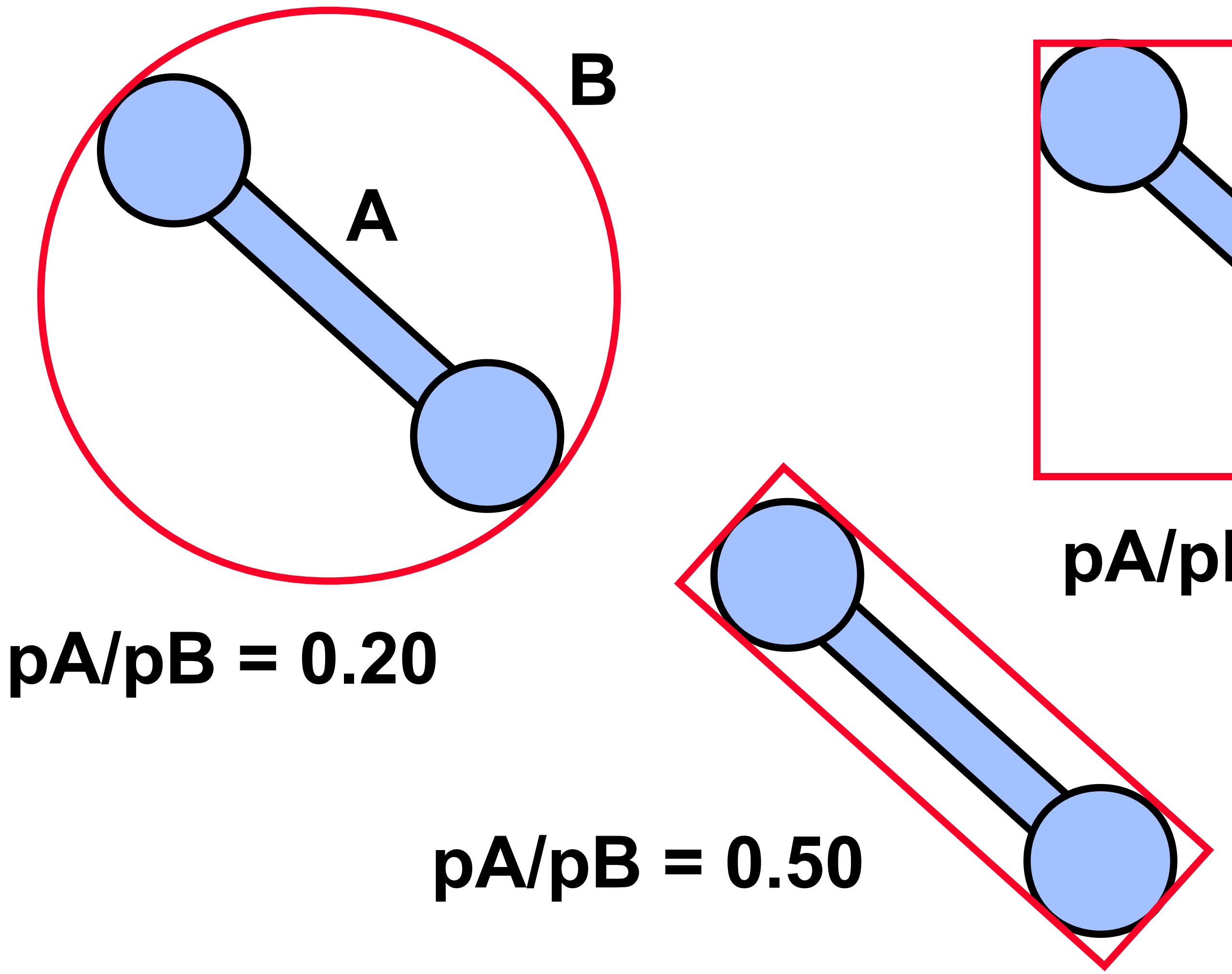
Bounding solid efficiency

Expected **intersection time** ray vs. object:

$$\underline{\mathbf{B + p \cdot I}} \quad < \quad \mathbf{I}$$

- **I** .. intersection time with an **original object**
- **B** .. intersection time with a **bouding solid**
- **p** .. probability of **hitting a bounding solid** (how many rays hit a bounding solid in total)

Bounding solid efficiency



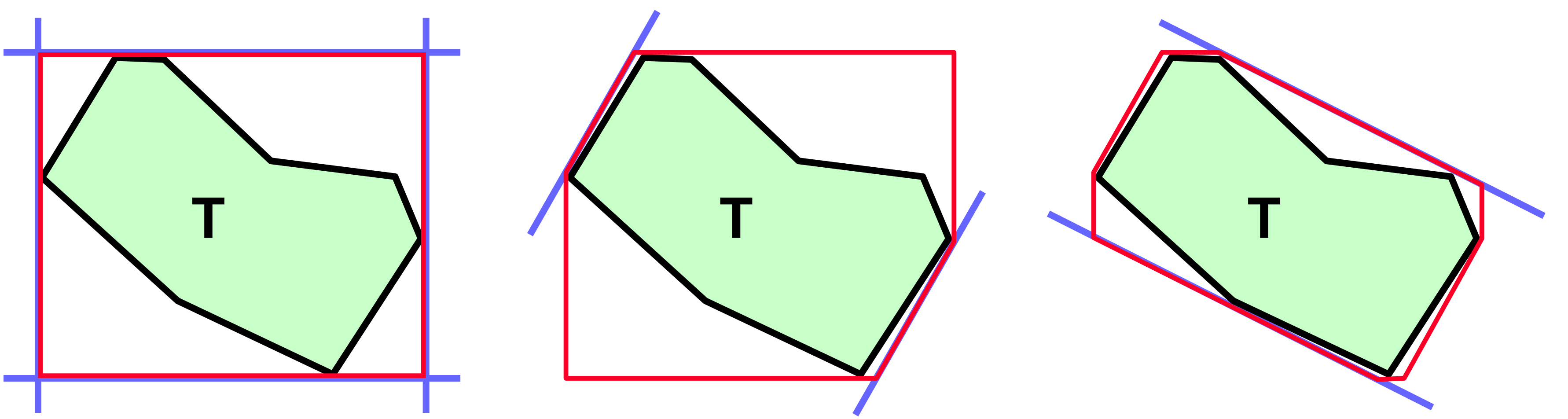
Combined bounding solids

- **better approximation** of an original shape
- **unions and intersections** of simple bounding shapes:

Convex shapes

- bounding solid for **convex shapes**
- ➡ intersection of strips (**"k-dops" system**)
  - strip = space between two parallel planes
  - efficient computation of **d** and **D** constants is necessary:

$$\mathbf{d} = \min_{[x,y,z] \in T} \{ ax + by + cz \}, \quad \mathbf{D} = \max_{[x,y,z] \in T} \{ ax + by + cz \}$$



Bounding solids – an efficient algorithm

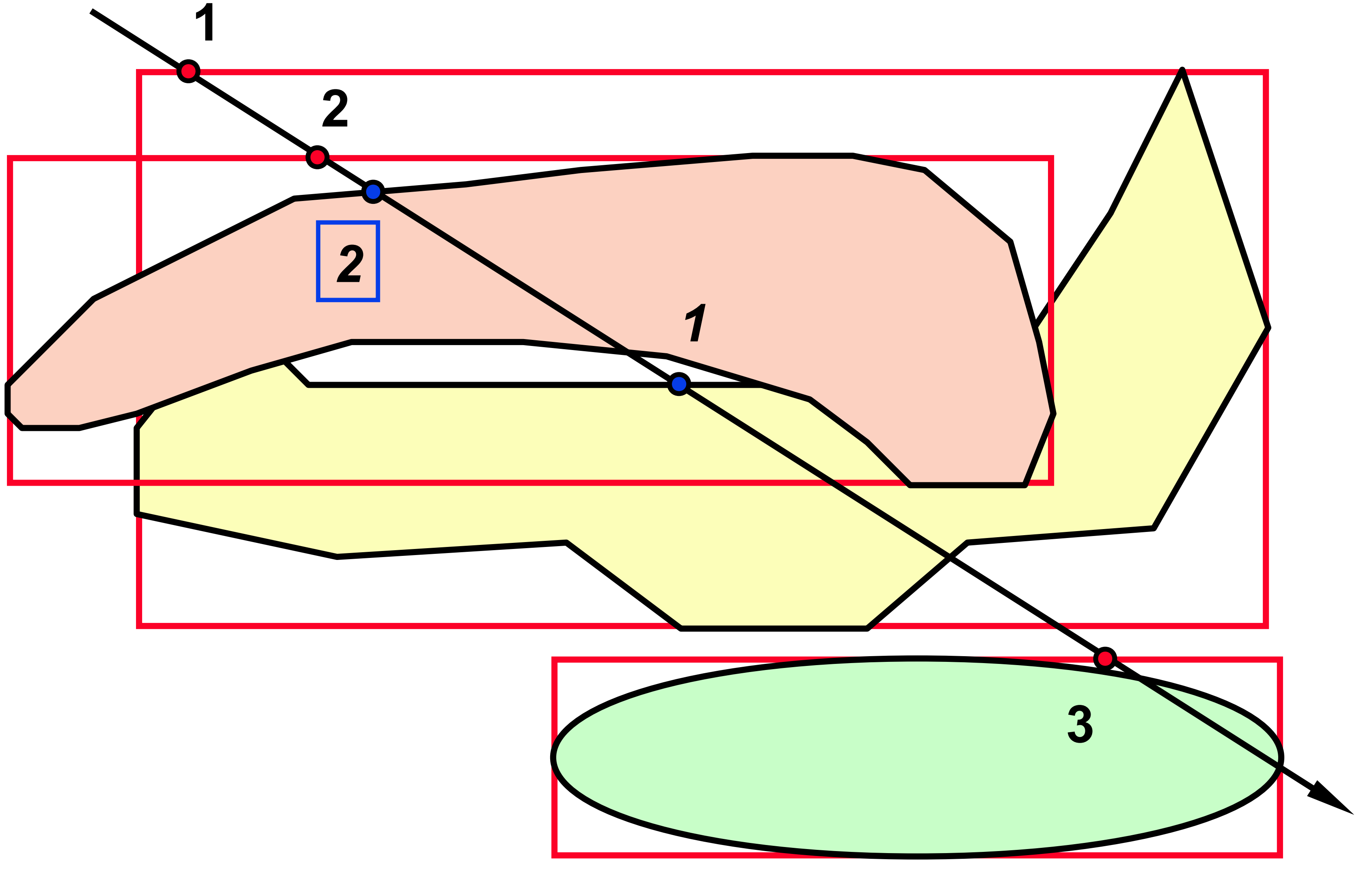
❶ intersections with all **bounding solids**

❷ intersected **bounding solids** are sorted in ascending order from the ray origin

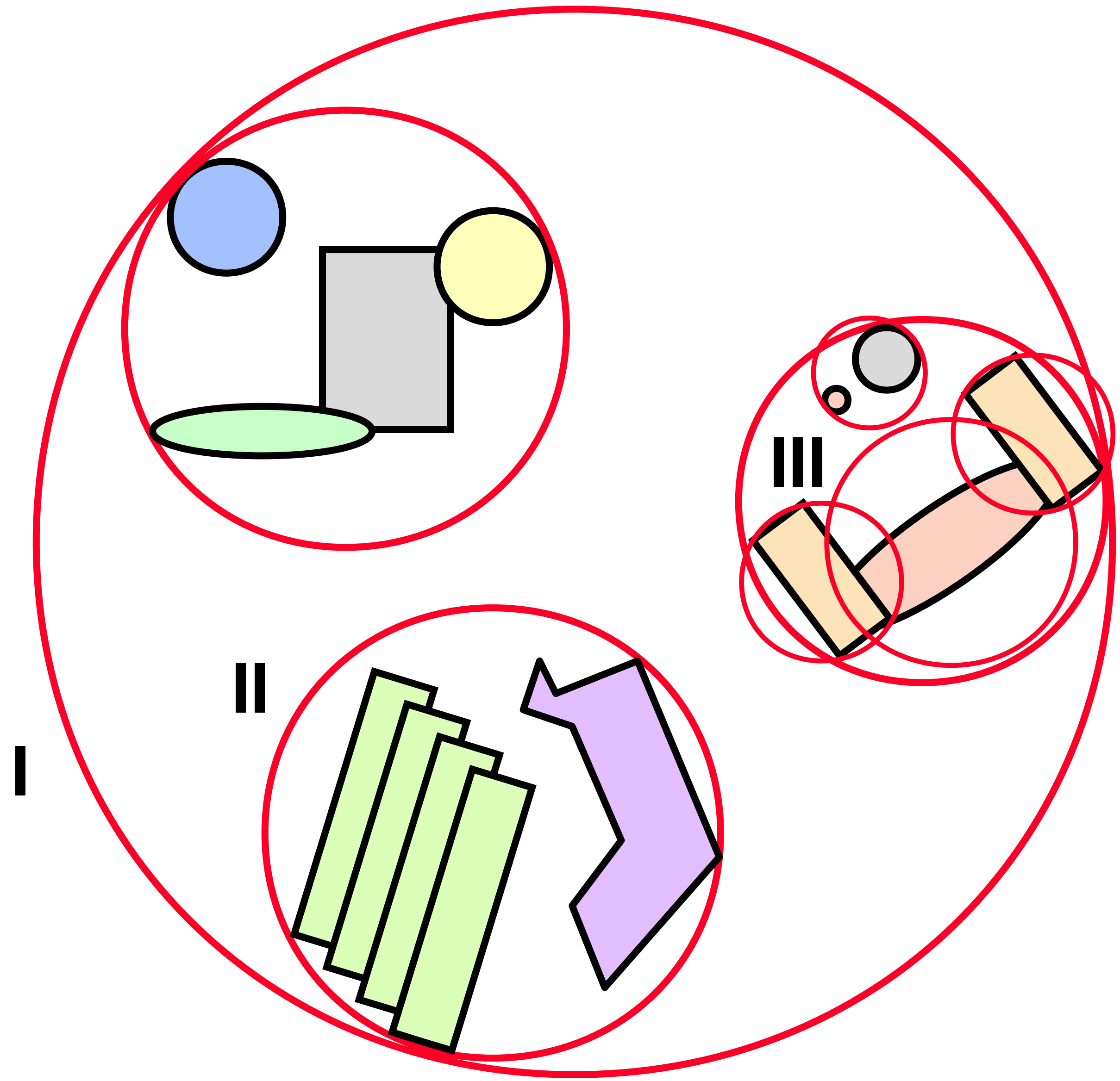
❸ **original objects** will be checked (intersected with the ray) in the same order

➜ if there is an intersection and all **bounding solids with closer intersection** were already tested, the intersection is the closest one

An efficient algorithm



Bounding Volume Hierarchy (BVH)



Hierarchy

- **ideal asymptotic complexity** is **O(log N)**
- efficient for **well structured scenes**
  – many well separated small objects / clusters
  – natural in CSG representation (cutting a CSG tree)
- **automatic construction** is possible
  – very complex optimal methods
  – suboptimal incremental algorithm
- in case of “AABB“ it is called **R-tree** (Guttman, 1984)
  – see: database spatial query technology

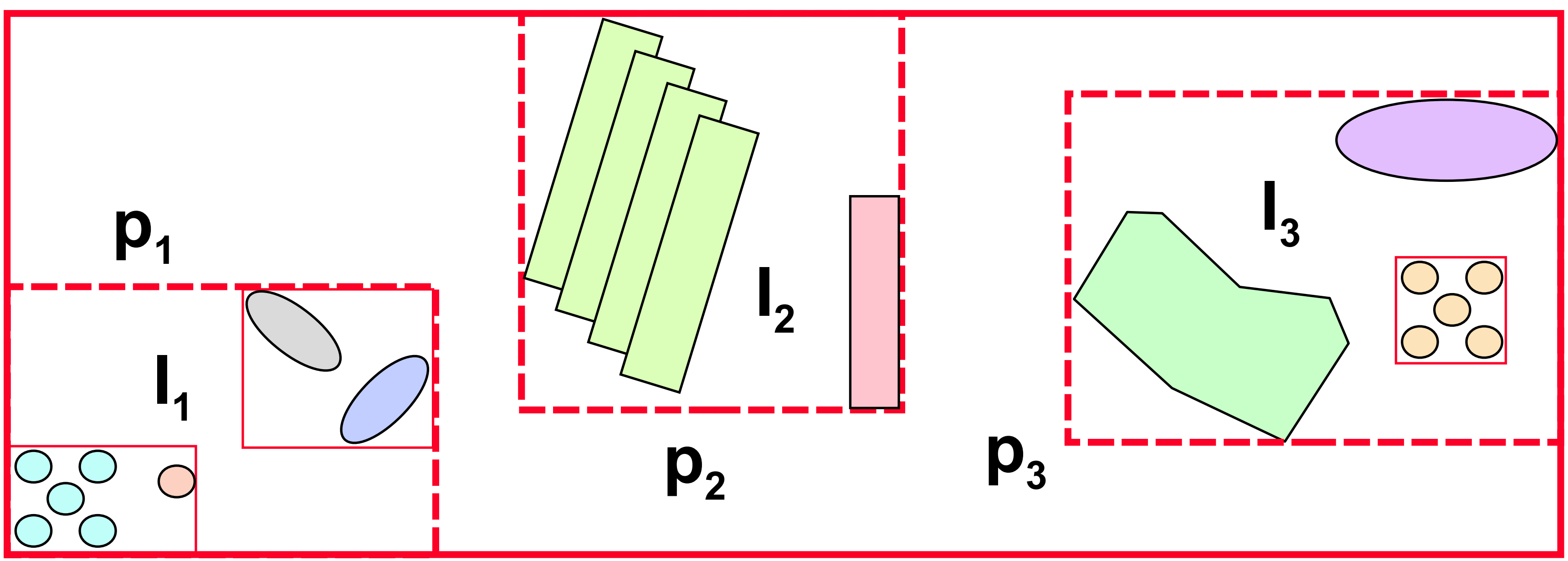
Efficiency of a hierarchy

$$K \cdot B + \sum_{i=1}^{K} p_i\, I_i \overset{?}{<} \sum_{i=1}^{K} I_i$$

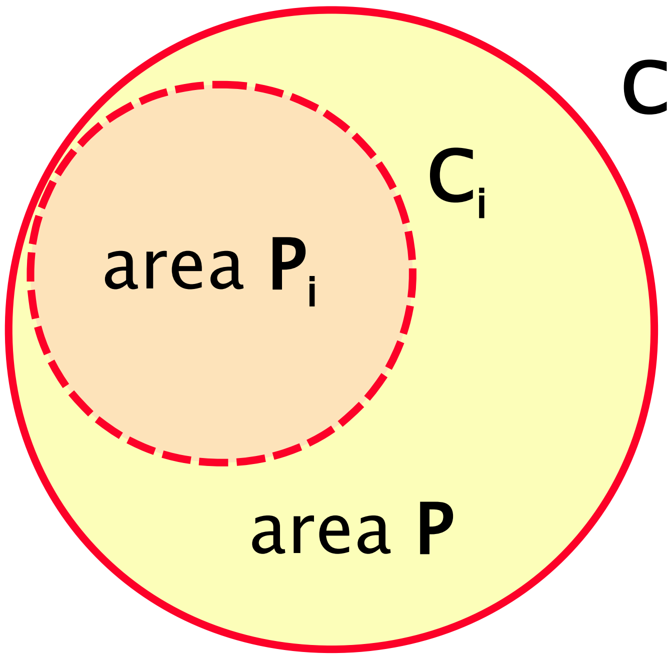
B .. intersection time with the bounding solid

p_i .. probability of hitting the i-th bounding solid

I_i .. time for objects inside of the i-th bounding solid



Efficiency of a hierarchy



P(d), $\mathbf{P_i(d)}$ .. area projected from the direction **d**

S, $\mathbf{S_i}$ .. surface area of a shape

For single direction **d**:

$$p_i = \Pr(\text{ hit } C_i \mid \text{ hit } C) = \frac{P_i(d)}{P(d)}$$

For every direction and **convex objects**:

$$p_i = \frac{\int P_i(d)\, dd}{\int P(d)\, dd} = \frac{S_i}{S}$$

Incremental construction ideas

1. create an **empty hierarchy** (tree root)
2. take the 1^{st} object and **insert it into the root**
   – root bounding solid must be updated
3. for the n^{th} object there are **options** (in one node):
   – object will be stand-alone (w/o any bounding solid)
   – object will have new bounding subsolid
   – object will go inside an existing bounding solid

➨ **order of insertion** objects does matter !
   – some defined 3D order and random shuffle

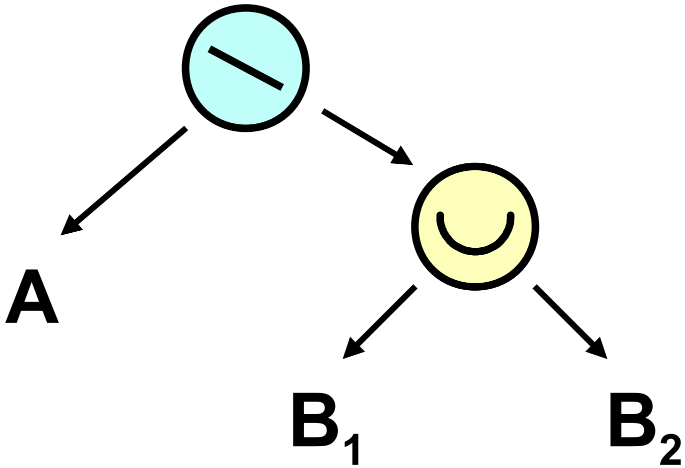
Bounding volume hierarchies

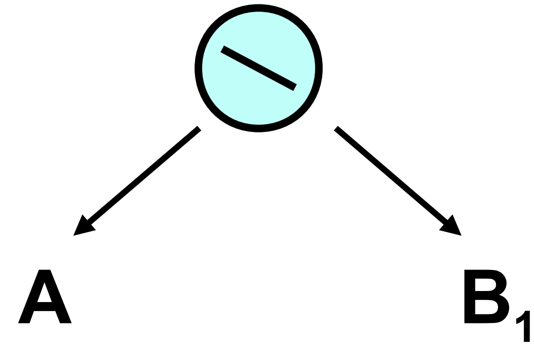
- “**Sphere tree**” (Palmer, Grimsdale, 1995)
  – simple test and transformation, worse approximation
- “**AABB tree**”, “**R-tree**“ (Held, Klosowski, Mitchell, '95)
  – simple test, comples transformation
- “**OBB tree**” (Gottschalk, Lin, Manocha, 1996)
  – simple transformation, more complex test, good approx.
- “**K-dop tree**” (Klosowski, Held, Mitchell, 1998)
  – more complex transformation and test, excellent approx.

“Cutting” CSG tree

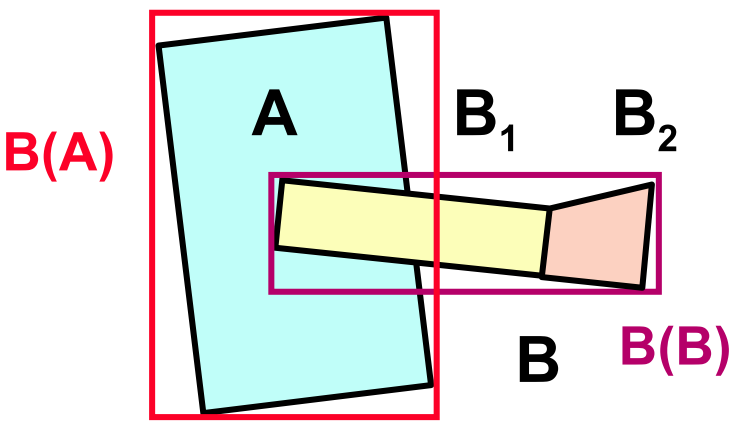
- efficient for **subtractive set operations** (intersection, difference)
- primary bounding solids are assigned to (finite) **elementary solids**
  - analytic computation
- bounding solids are propagated **from leaves to the root node**
- **subtractive operations** can reduce bounding solids in ancestors (arguments)

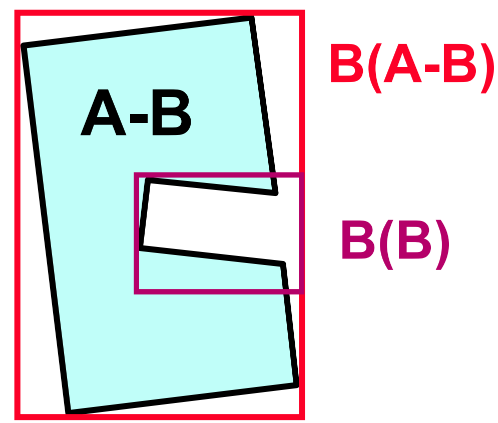
“Cutting” CSG tree





$B(A\text{-}B) \cap B(B_2) = 0$

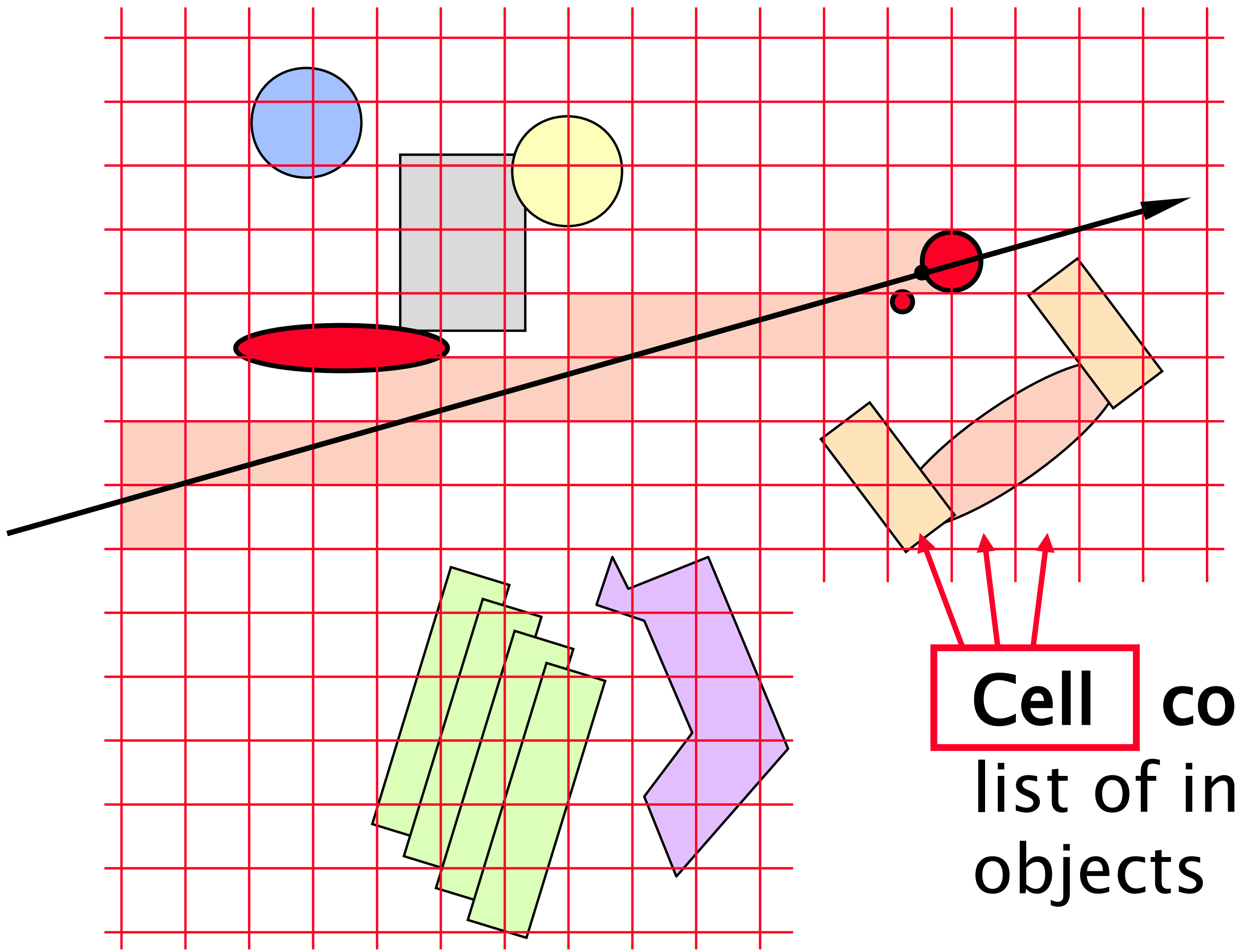




Space subdivision (spatial directories)

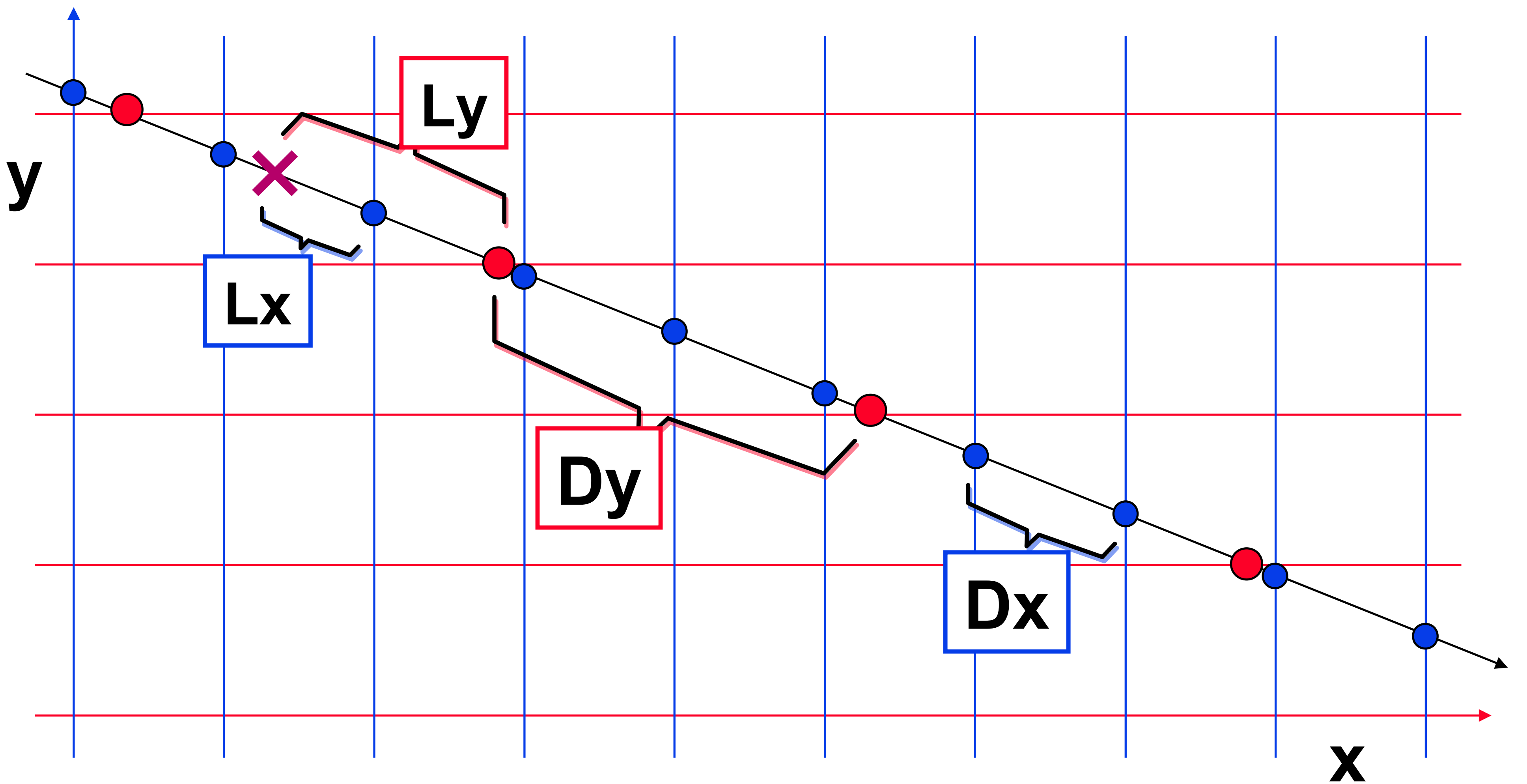
- **uniform subdivision** (equal cells)
  - \+ simple traversal & addressign
  - – many traversal steps
  - – big data volume
- **nonuniform subdivision** (mostly adaptive)
  - \+ less traversal steps
  - \+ less data
  - – more complex implementation (data struture & traversal)

Uniform subdivision (grid)



$\boxed{\text{Cell}}$ **contains** list of intersected objects

Grid traversal (3D DDA)



Grid traversal (3D DDA)

- **ray**: $\mathbf{P_0} + \mathbf{t} \cdot \vec{\mathbf{p_1}}$ for $\mathbf{t > 0}$

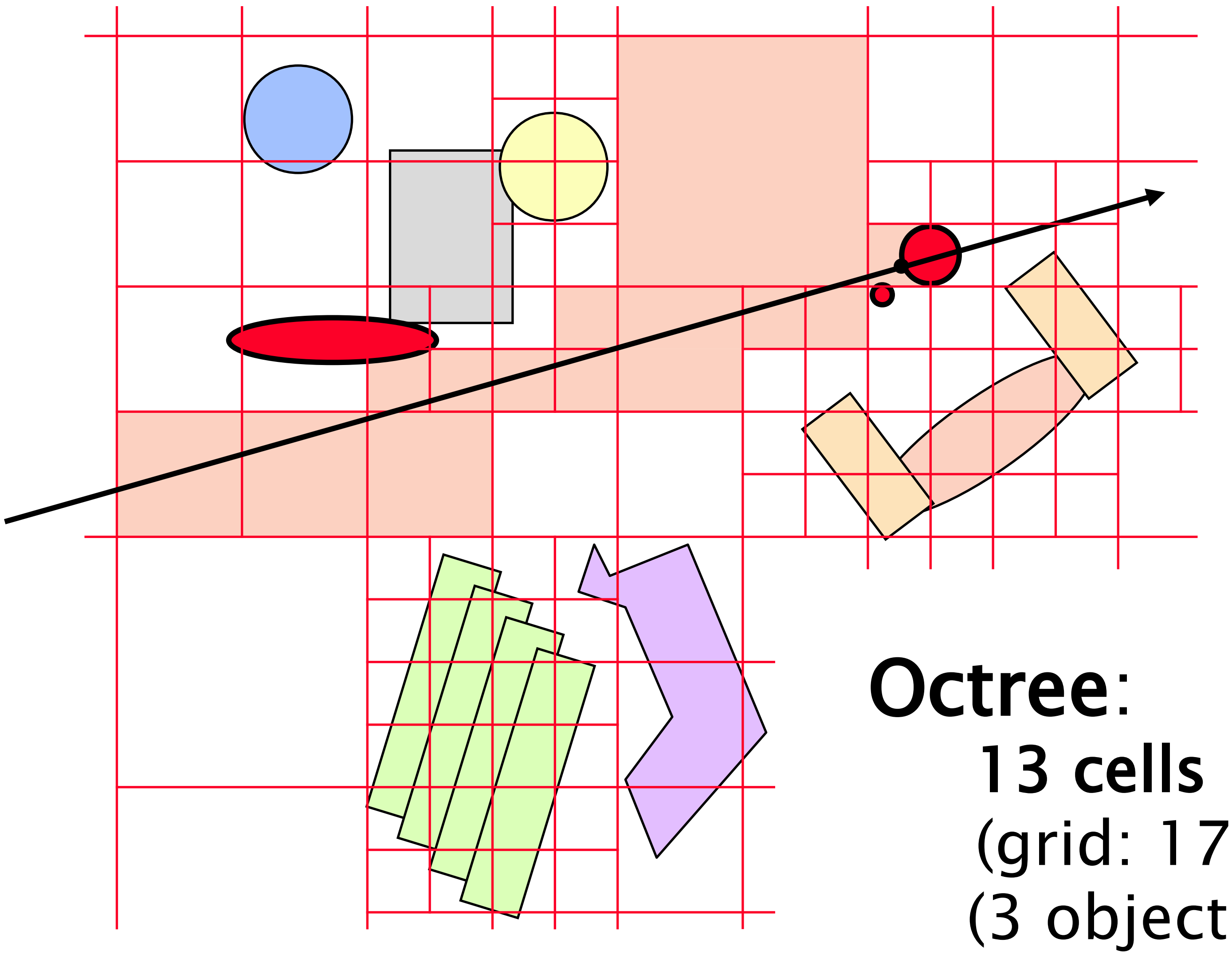
- for the given direction $\vec{\mathbf{p_1}}$ there are precomputed **constants** **Dx**, **Dy**, **Dz**:
  - distance between subsequent intersections of the ray and the parallel wall system (perpendicular to **x**, **y**, **z**)

- for the $\mathbf{P_0}$ there is an **initial cell [i, j, k]** and quantities **t**, **Lx**, **Ly**, **Lz**:
  - ray parameter **t**, distances to the closest walls in the **x**, **y**, **z** system

Grid traversal (3D DDA)

- ❷ processing in the **cell [i, j, k]** (intersections)
- ❸ stepping to the **next cell**:
  - **D = min {Lx,Ly,Lz};** /* assumption: **D = Lx** */
  - **Lx = Dx; Ly = Ly - D; Lz = Lz - D;**
  - **i = i ± 1;** /* according to the sign of $\mathbf{P_{1x}}$ */
- ❹ **end conditions**:
  - an actual (the closest) intersection was found
    - » the intersection is <u>in the current cell</u>
  - no intersection was found and the next cell is outside of the grid domain

Nonuniform subdivision of space



Octree:

13 cells

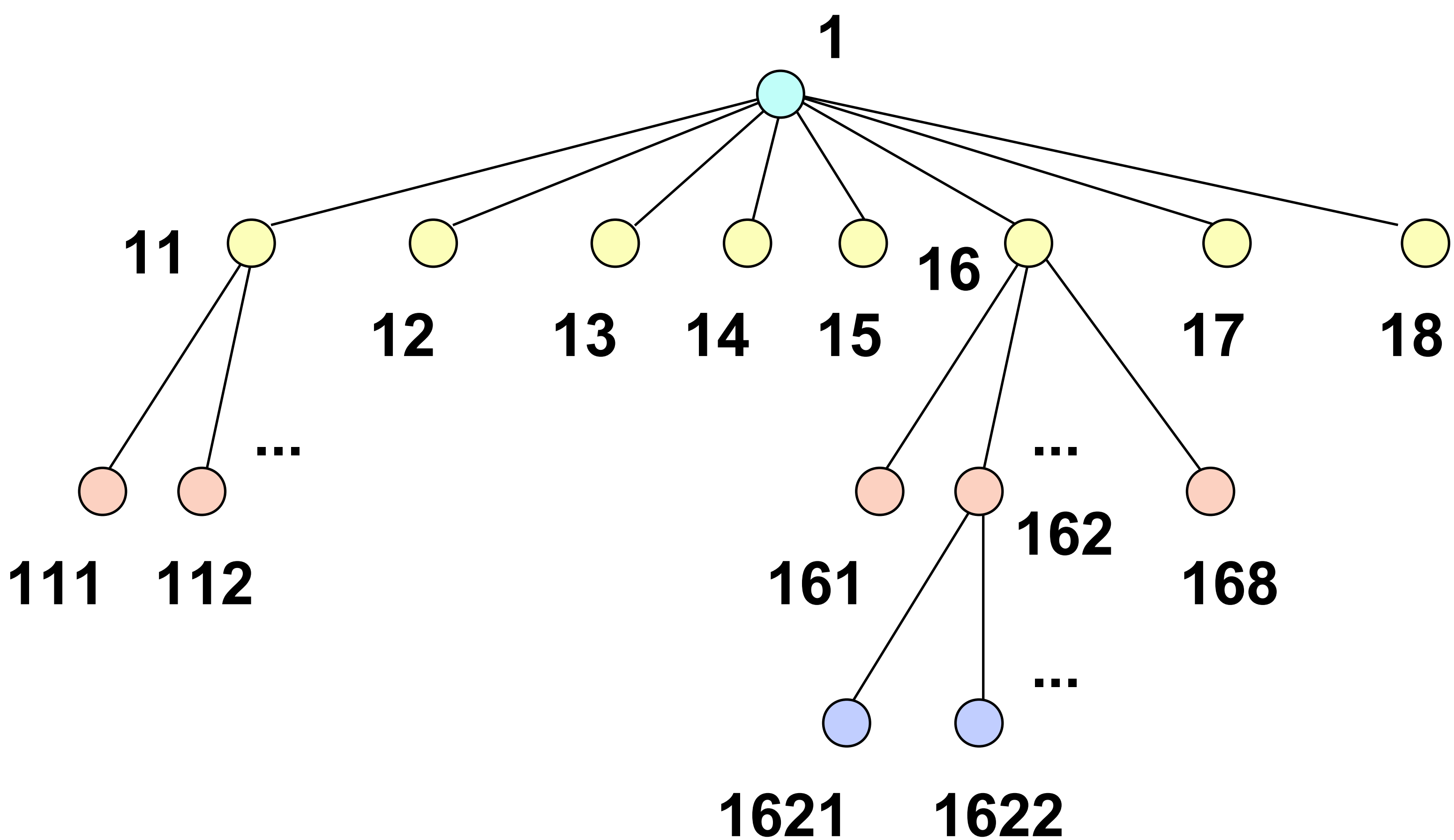
(grid: 17 cells)

(3 objects tested)

Adaptive subdivision systems

- **octree** (division in the middle)
  - representation – pointers, implicit representation or hash table (Glassner)
- **KD-tree** (Bentley, 1975)
  - static division: in the middle, cyclic coordinate component
  - adaptive: both components and bounds are dynamic
- [general **BSP-tree**]
  - dividing planes have arbitrary orientation

Octree storage by Glassner



Octree storage by Glassner

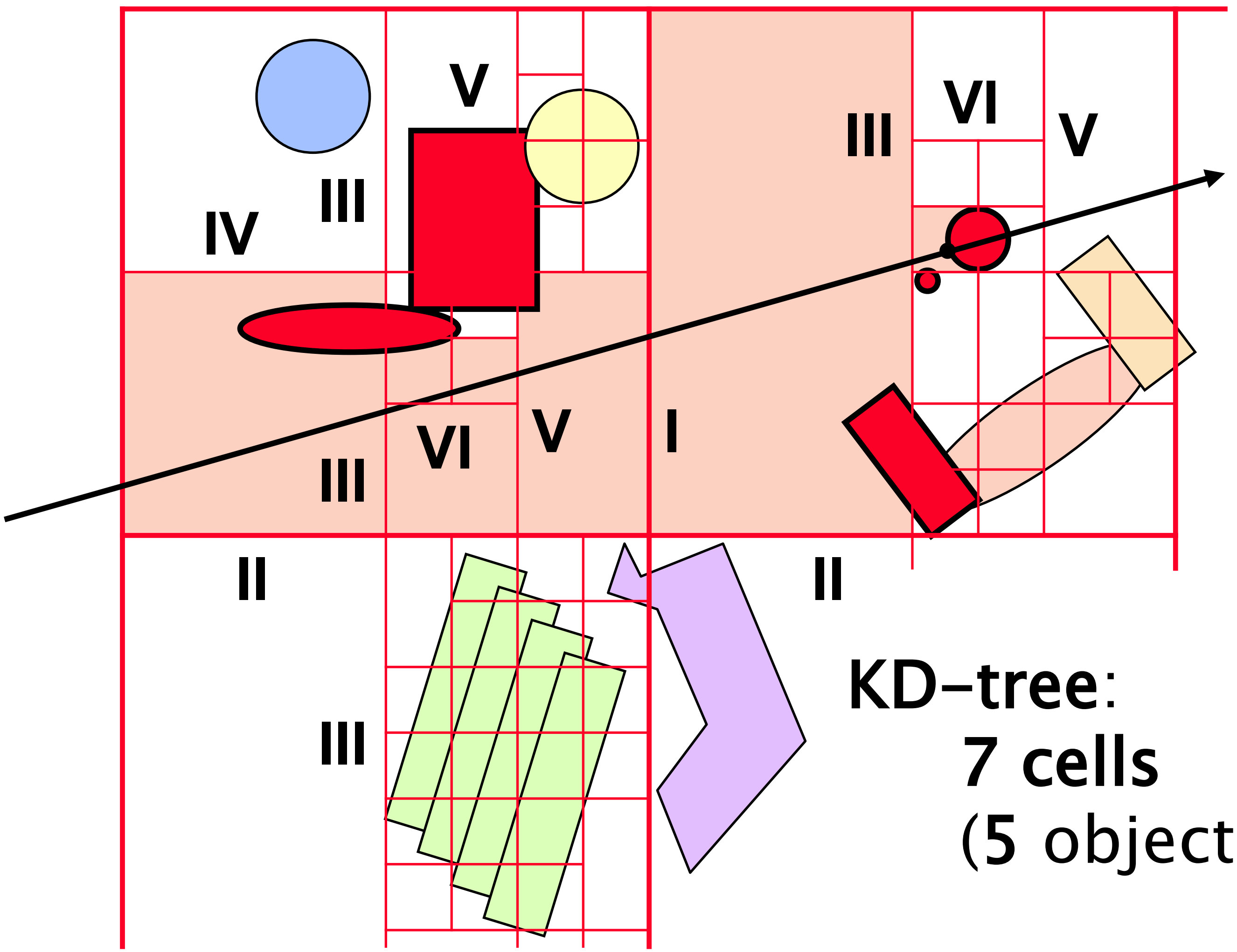
- each individual cell has its **signature**
  – root .. **1**
  – ancestors of the root .. **11** až **18**, .. etc.
  – <u>each voxel</u> (potential cell) has its specific signature

- actual **tree nodes** are stored in **sparse hash table**
  – hash-function example: **Signature <u>mod</u> TableSize**

Tree traversal (Glassner)

- point on the ray .. **[x, y, z]**
  - associated voxel's signature .. **[1 ÷ 8]** ^{k}
- look for **all prefixes** of the code in the hash table
  - the 1st (shortest) found prefix defines the current cell
- after **cell processing** the point **[x, y, z]** is moved in the direction of the ray ($\mathbf{p}_1$)
  - the new point is localized, ...

KD–tree (static variant)



KD–tree:
7 cells
(**5** objects tested)

Adaptive subdivision criteria

❶ limited **number of objects** and **subdivision depth**

– if a cell is intersected by more than **M objects** (e.g. **M = 4 .. 32**), subdivide it

– **maximal subdivision level** is **K** (e.g. **K = 5 .. 25**)

❷ limited **number of cells** or **memory occupation**

instead of subdivision depth limit:

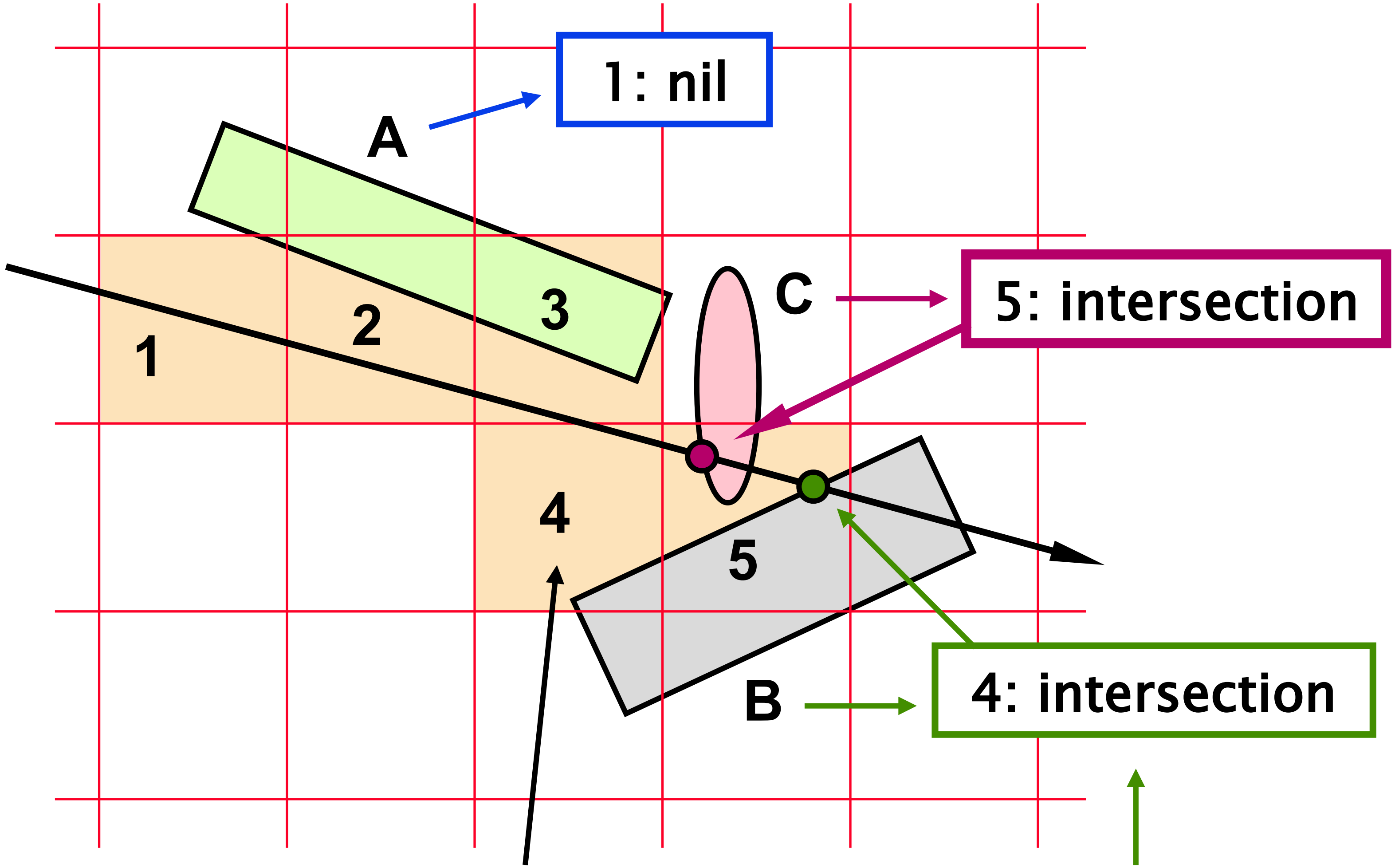
– subdivision is finished after filling the whole **dedicated memory**

– subdivision controlled by a **breadth-first traversal** (FIFO data structure holding candidate cells)

Traversing adaptive subdivision

- **marching the ray**: finding the next cell from the root (see Glassner's method)
- **preprocessing**: tree traversal used for dividing the ray into individual segments (intersections with cells)
  - **t** parameter segments for individual cells
- **additional support data** (à la “finger tree”)
  - pointer to the neighbour cell (on the same tree level)
- recursive **depth-first traversal** with heap
  - heap: list of potentially intersected cells (ordered from the most promising ones)

“Mailbox” technique

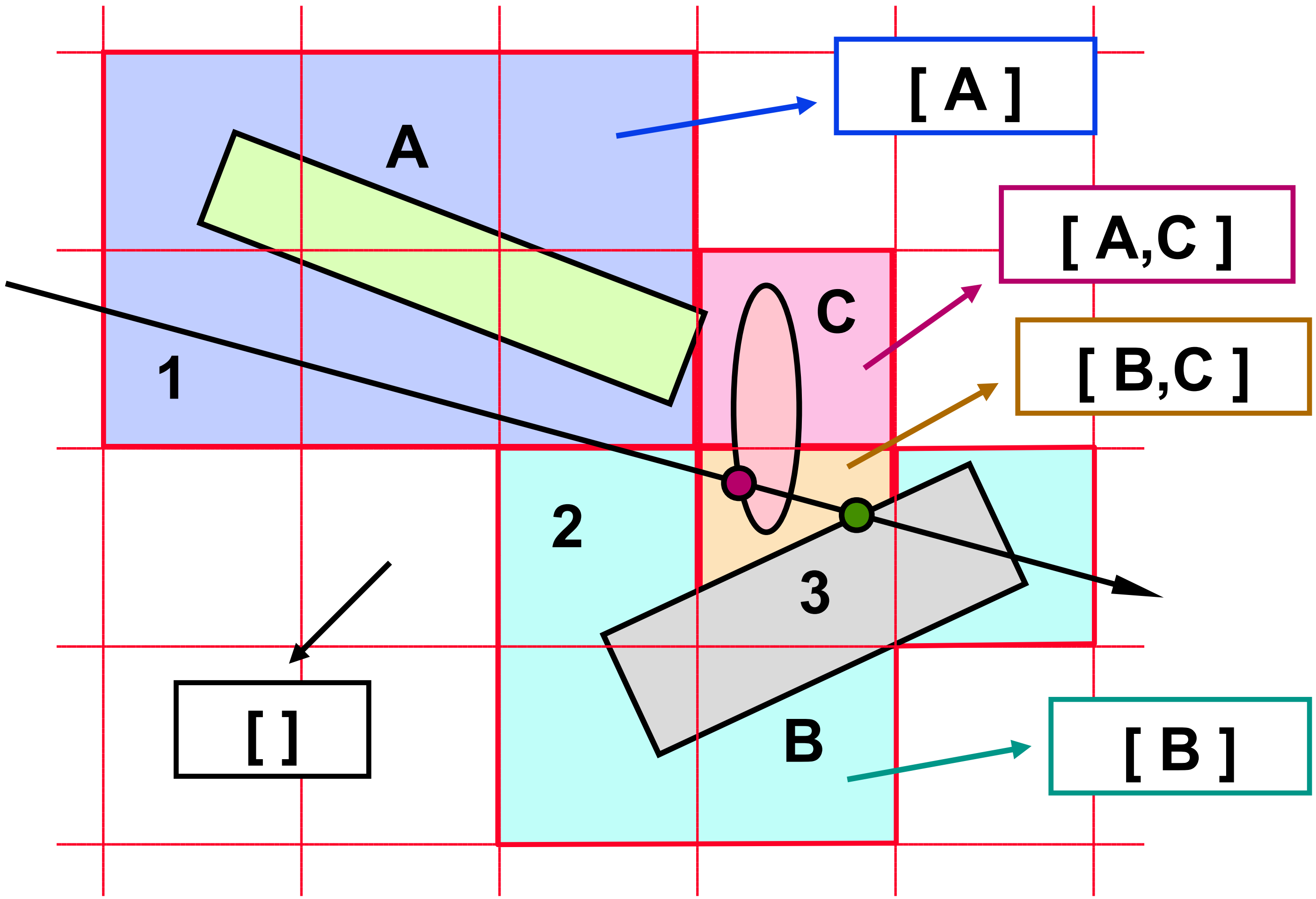


The intersection must be in the current cell (else it is cached)

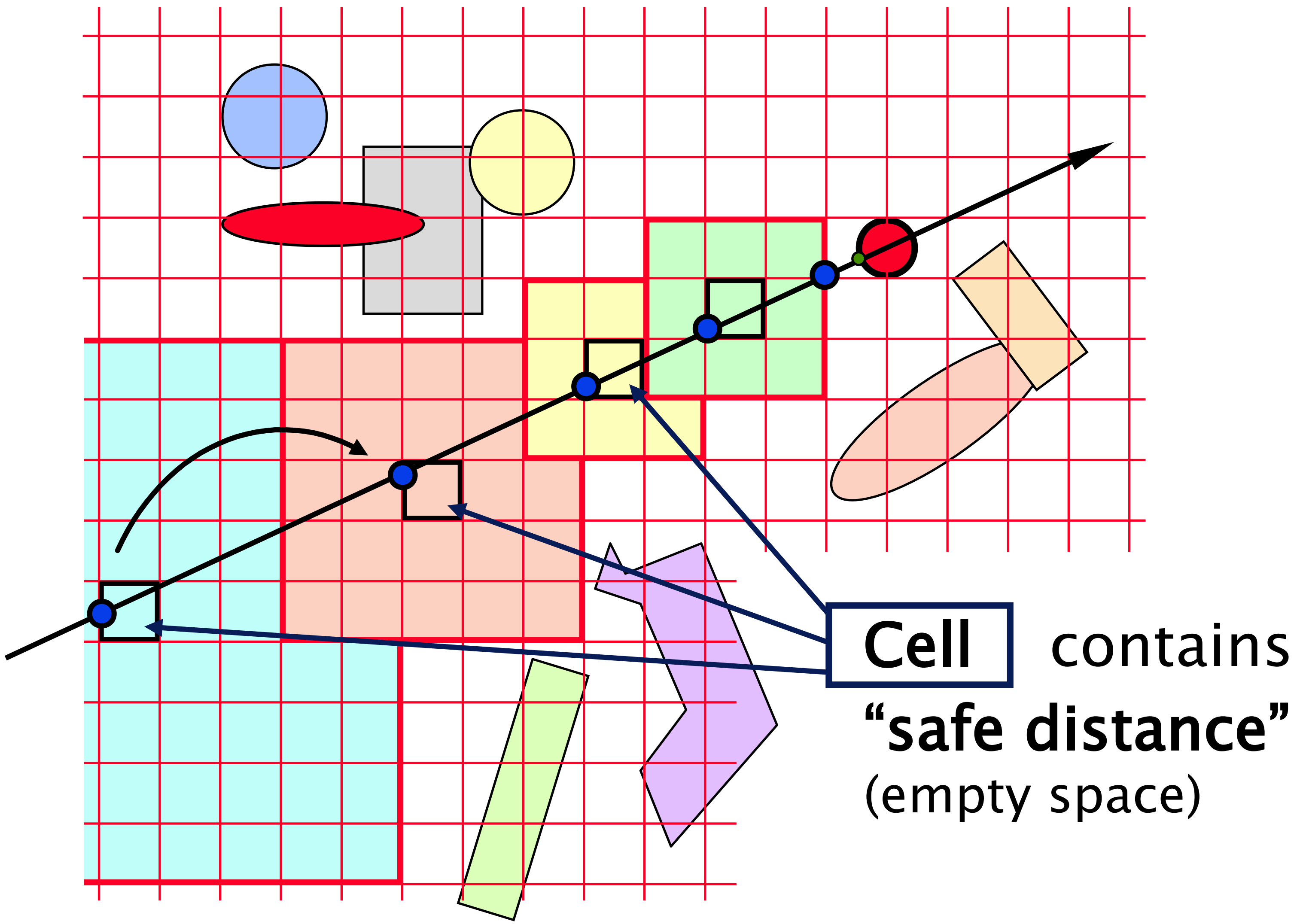
Abstract space division

- **no need to test** (even to access!) lists of objects, which were already tested
- list of objects needs to be processed only in a cell with **different (bigger) set of objects**
- cells can share **equal object lists**
  - tested lists are marked by a special **flag**
  - processing only **nonmarked** lists
  - **mailbox** technique is used on the object level

Abstract space division



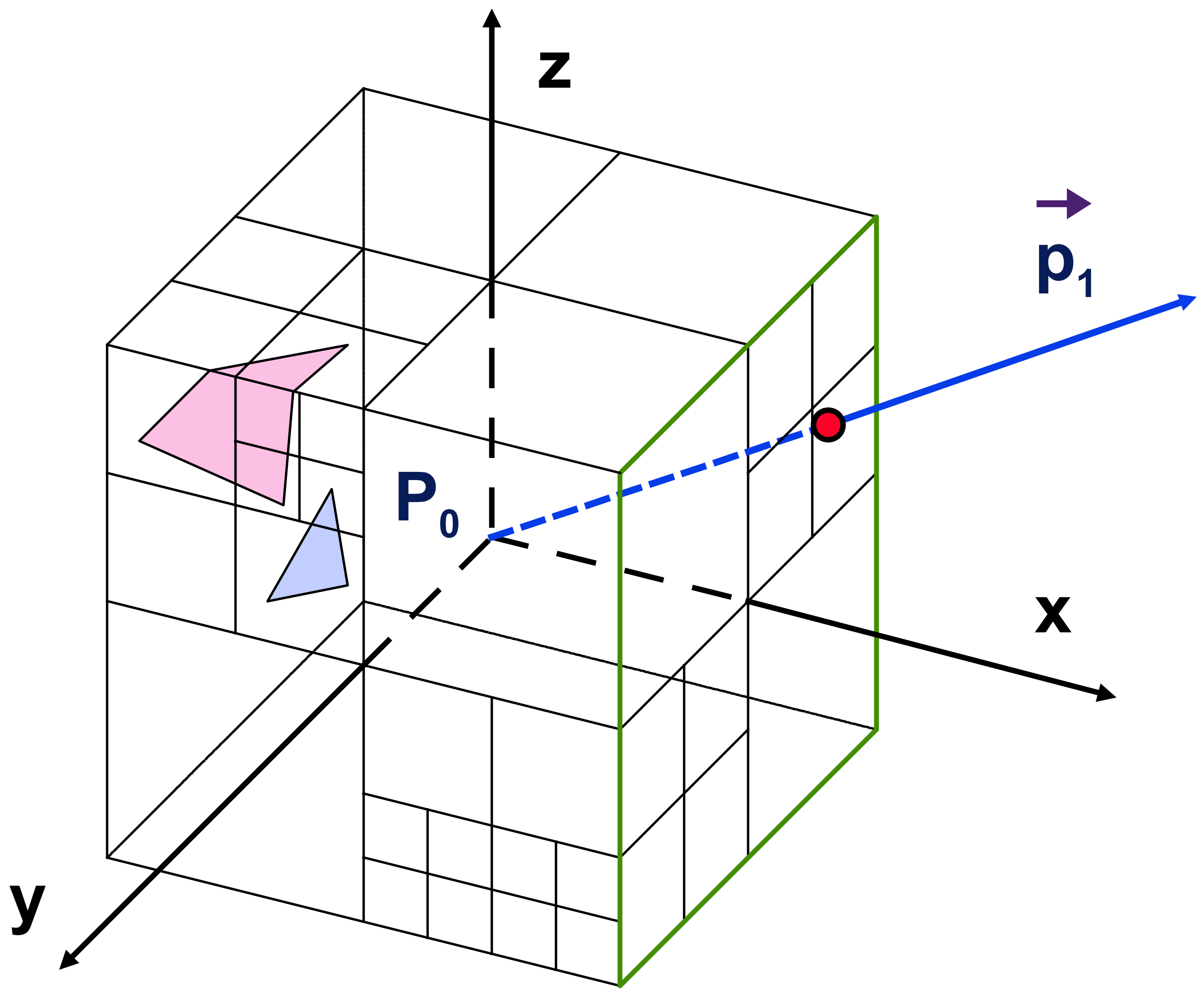
Macro-cells (Miloš Šrámek)



Directional speedup techniques

- utilizing **directional cube**:
- **light buffer**
  – speeding up shadow rays to point light sources
- **ray coherence**
  – for all secondary rays
- 5D **ray classification**
- **image plane directory** (visibility precomputation)
  – only for primary rays

Directional cube (adaptive)



Directional cube

- **axis-oriented**
- cube faces divided into **cells**
  - uniform or adaptive division
  - every cell stores list of relevant objects (can be ordered by the distance from the cube)

➨ **HW rasterization and visibility** (depth-buffer) can be used for uniform division

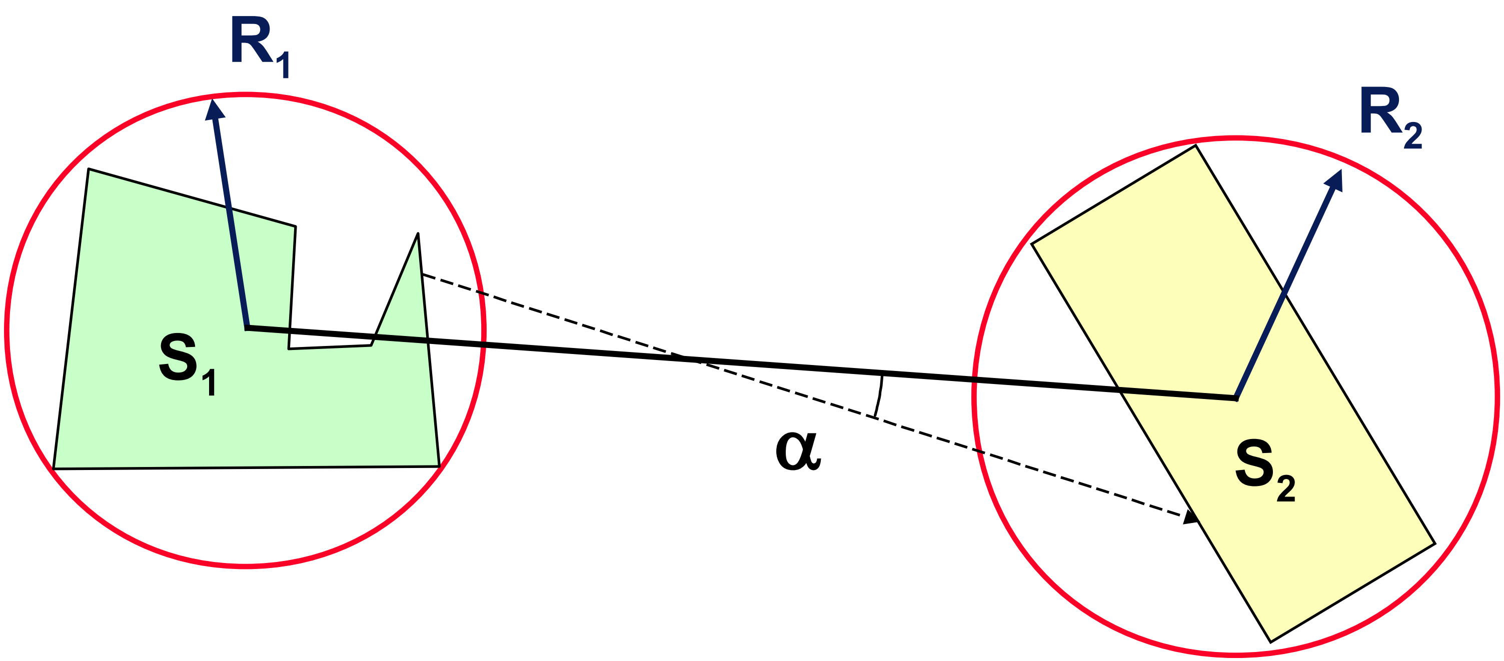
Light buffer

- speeding up **shadow rays**

- **directional cube** in every point light source
  - possible visibility of objects from the light-source point
  - some cells might be covered completely by one object (everything else is in shadow)

- for a **shadow ray** only objects projected in the relevant cell are considered

Ray coherence



$$\cos\alpha \geq \sqrt{1 - \frac{R_1 + R_2}{\left\| S_1 - S_2 \right\|}}$$

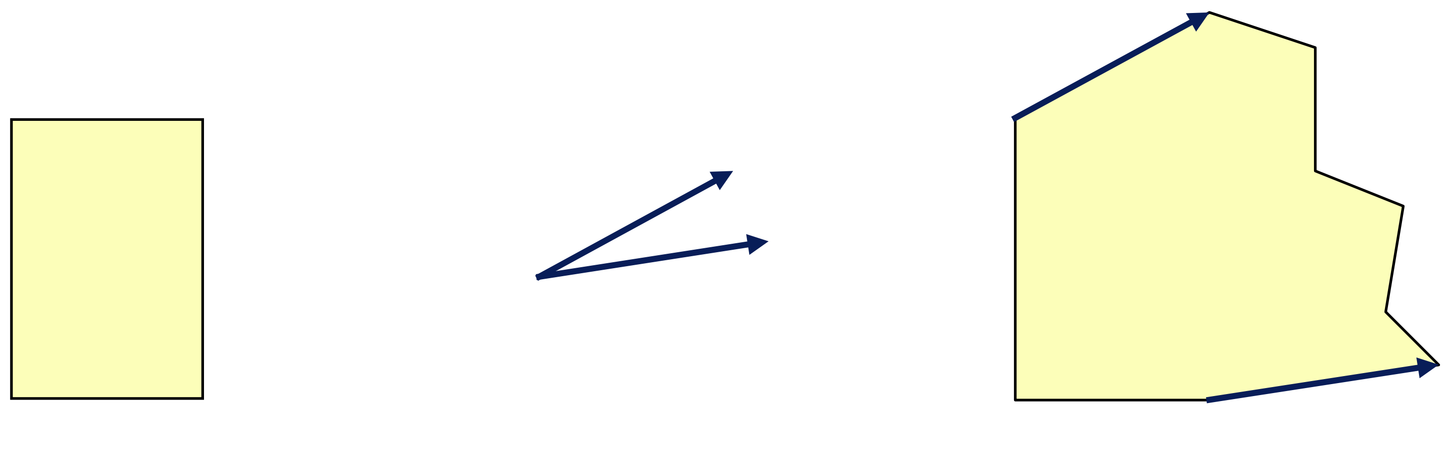
Speedup utilizing coherence

- for every **secondary rays**
  - reflected, refracted, shadow
- assumed bounding solid: **sphere**
- ➡ directional cube placed in every **center of bounding sphere**
  - list of projected objects/light sources in every cell
    - » coherence condition is used
    - » lazy evaluation!
  - lists can be ordered by distance from the cube

5D ray space

- rays in 3D scene:
  - **origin** P_0 **- [x, y, z]**
  - **direction [φ, θ]**
- **5D hypercube** divided into **cells**
  - every cell contains list of possible intersections for the associated ray <u>pencil</u> (“beam”)
  - <u>adaptive subdivision</u> (merging neighbour cells with equal or similar lists)
- **6D variant**: one more quantity (time) for animations

Ray classification



origin (2–3D) + direction (1D, 2D) = bundle / pencil

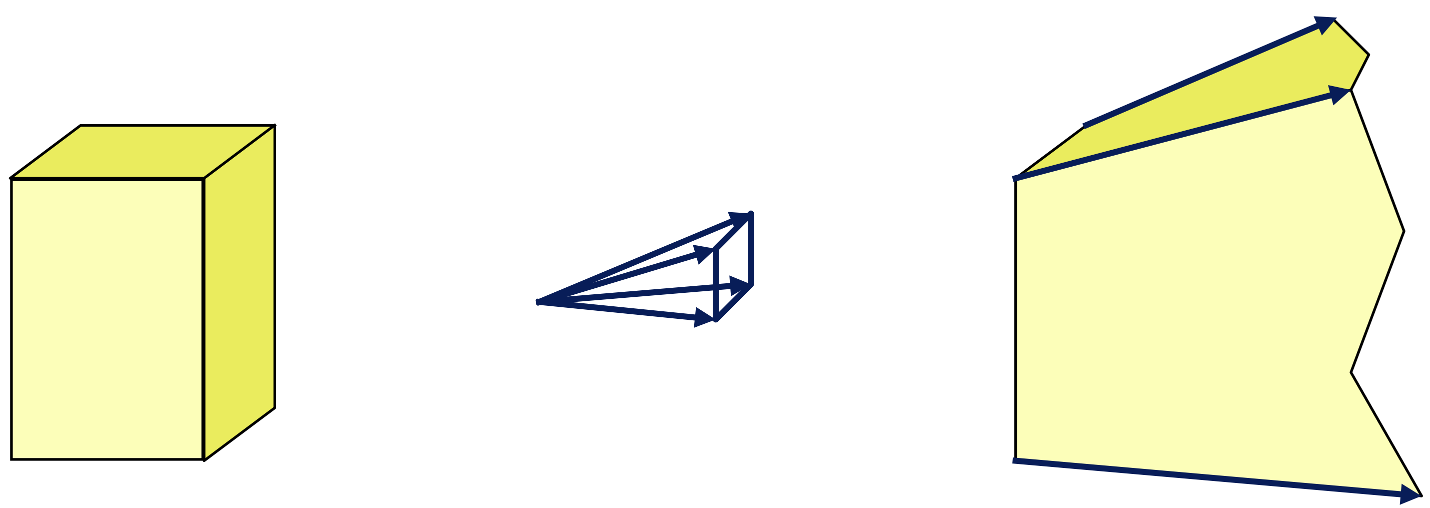


Image plane directory

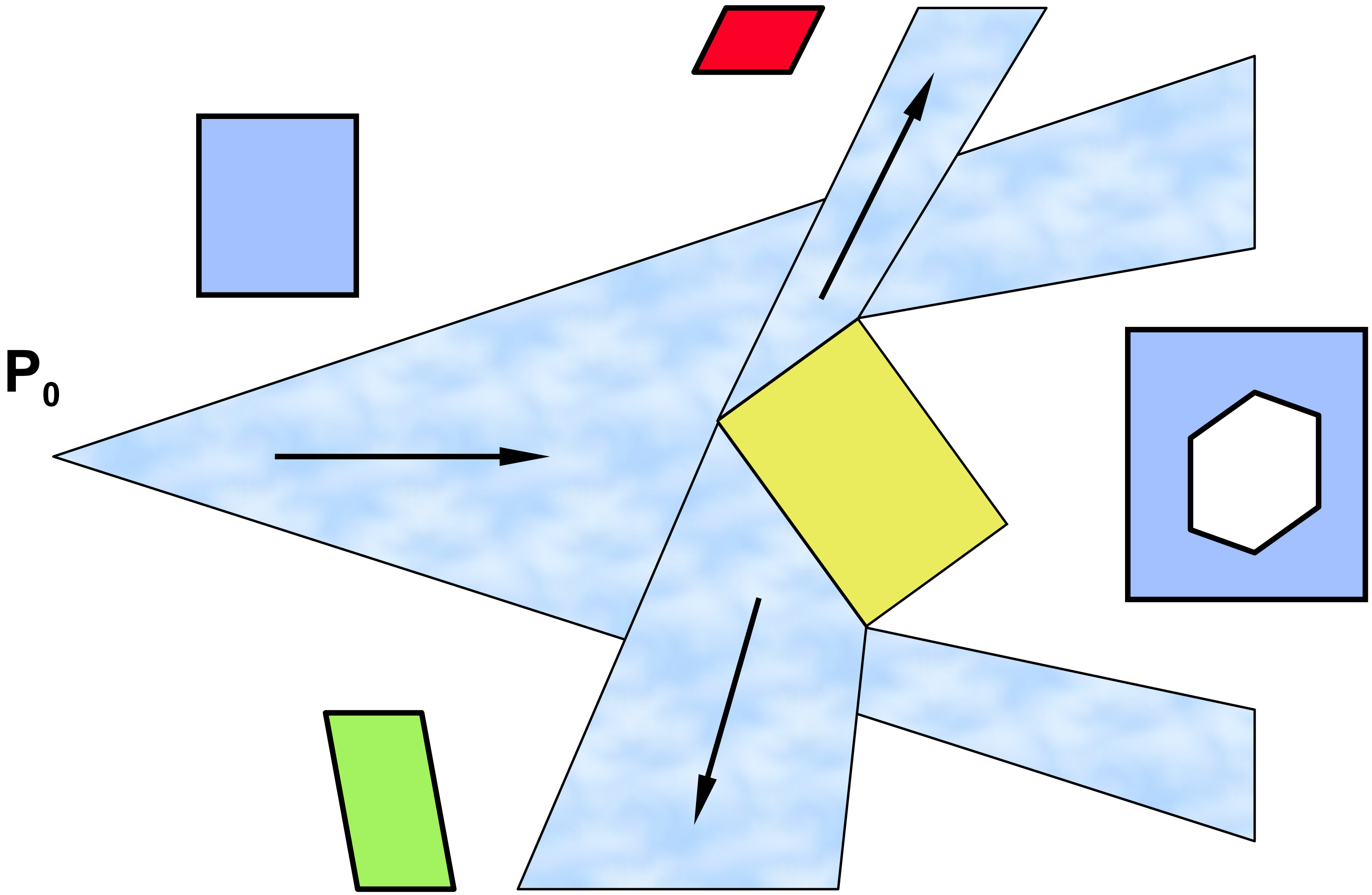
- for **primary rays**
- projection plane is (adaptively) divided into **cells**
  - possible visibility of individual objects in a cell (together with order)
  - complete coverage by one cell by one object is possible (hard to test)

➨ robust variant **of used visibility method**
  - in most pixels it can be done with complete certainty

Generalized rays

- computing more information about **f(x,y)**
  - for <u>anti-aliasing</u> (average color estimation) or <u>soft shadows</u> (shadow ratio)
  - some restrictions to a scene are necessary
- forms of **generalized rays**
  - rotational or elliptical cone, regular pyramid
  - pyramid with <u>polygonal cross section</u> (polygonal scene, see the next slide)

Polygonal scene



References

- **A. Glassner**: *An Introduction to Ray Tracing*, Academic Press, London 1989, 201-262
- **A. Watt, M. Watt**: *Advanced Animation and Rendering Techniques*, Addison-Wesley, Wokingham 1992, 233-248
- **V. Havran:** *Heuristic Ray Shooting Algorithms*, PhD thesis, FEL ČVUT Praha, 2001
- **P. Konečný:** *Obalová tělesa v počítačové grafice*, M.S. thesis, Masaryk University, Brno 1998

References II

- **J. Klosowski, M. Held, J. Mitchell, H. Sowizral, K. Zikan:** *Efficient collision detection using bounding volume hierarchies of k-dops*, IEEE Transactions on VaCG, 21–36, January-March 1998
- **H. Samet:** *Foundations of Multidimensional and Metric Data Structures*, Morgan Kaufmann, 2006
- **H. Samet:** *The Design and Analysis of Spatial Data Structures*, Addison-Wesley, 1990

