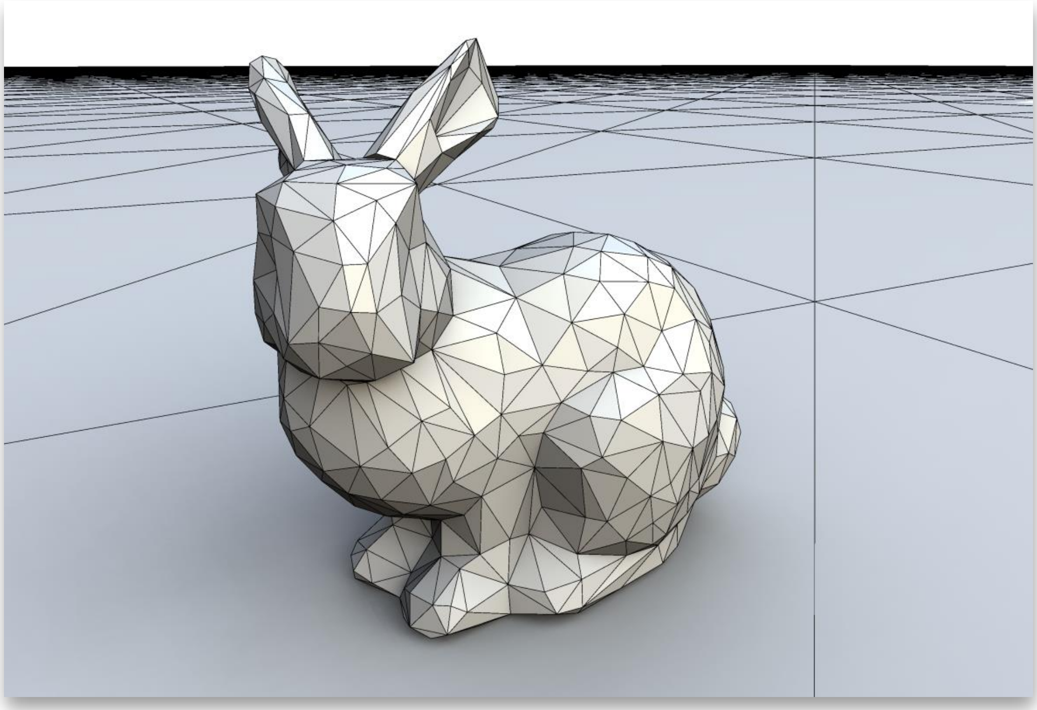
Graphics 2014



Raytracing

Recursive Raytracing · Data Structures

[Faculty of Science]
Information and
Computing Sciences

Universiteit Utrecht

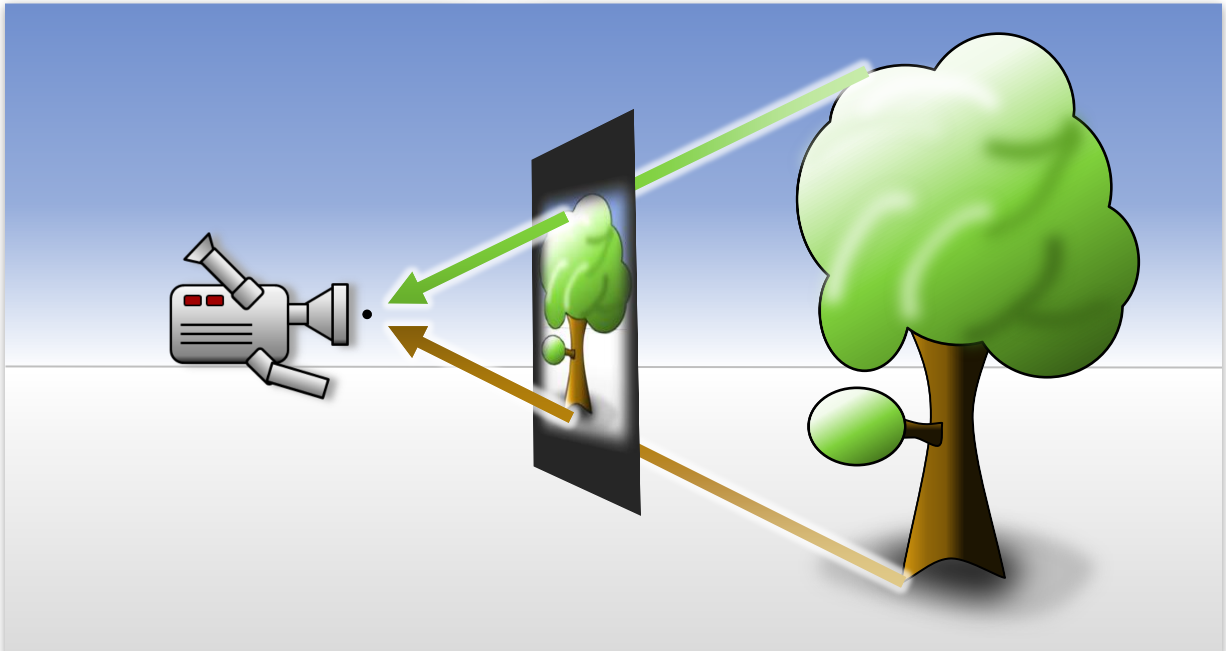
Basic Raytracing

BASIC

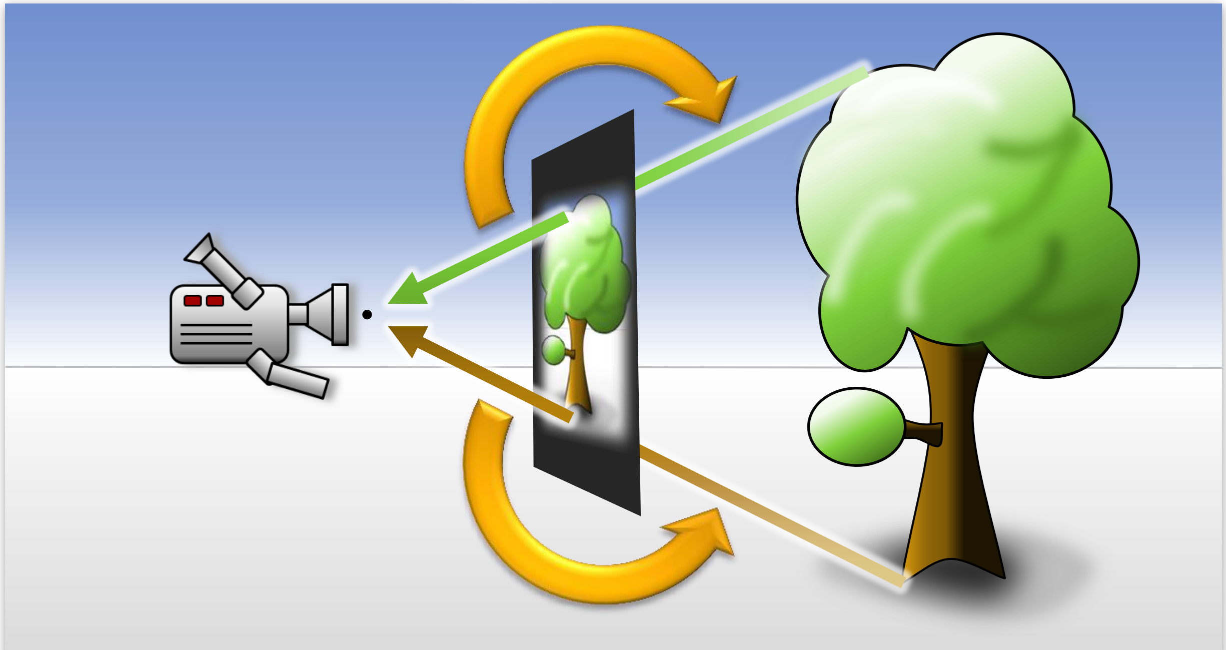
basic topics

study completely

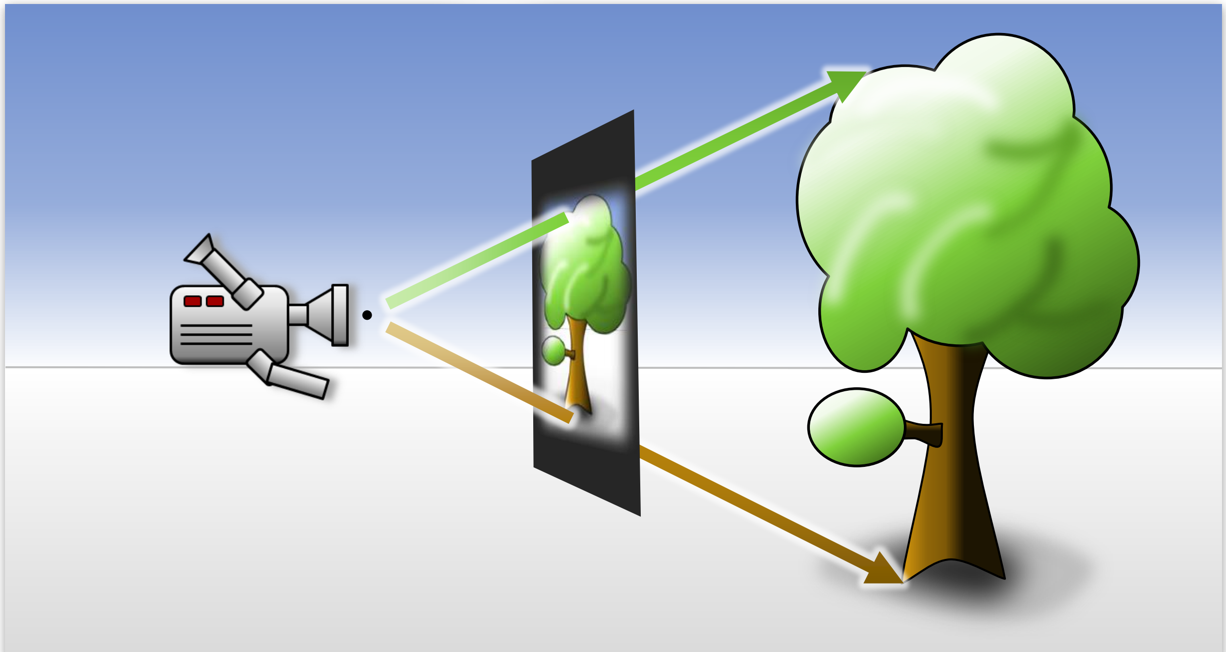
Central Projection



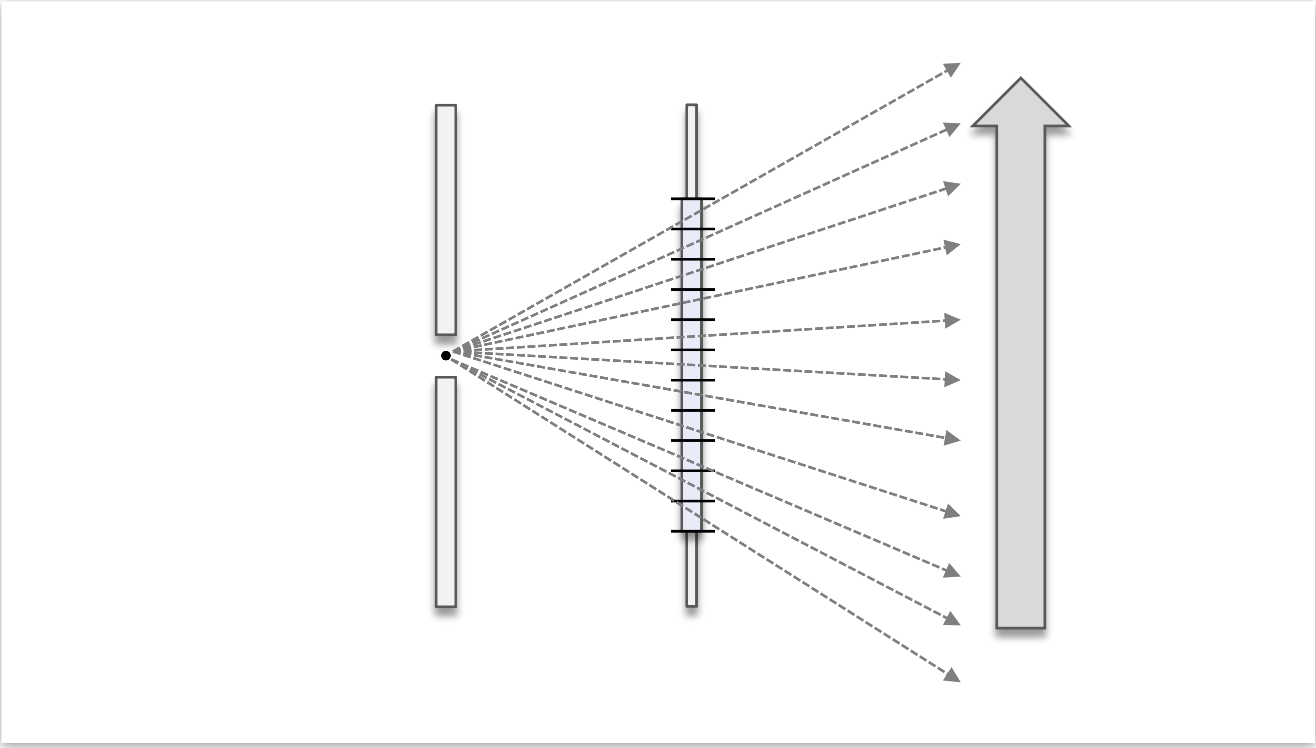
Central Projection



Ray Tracing



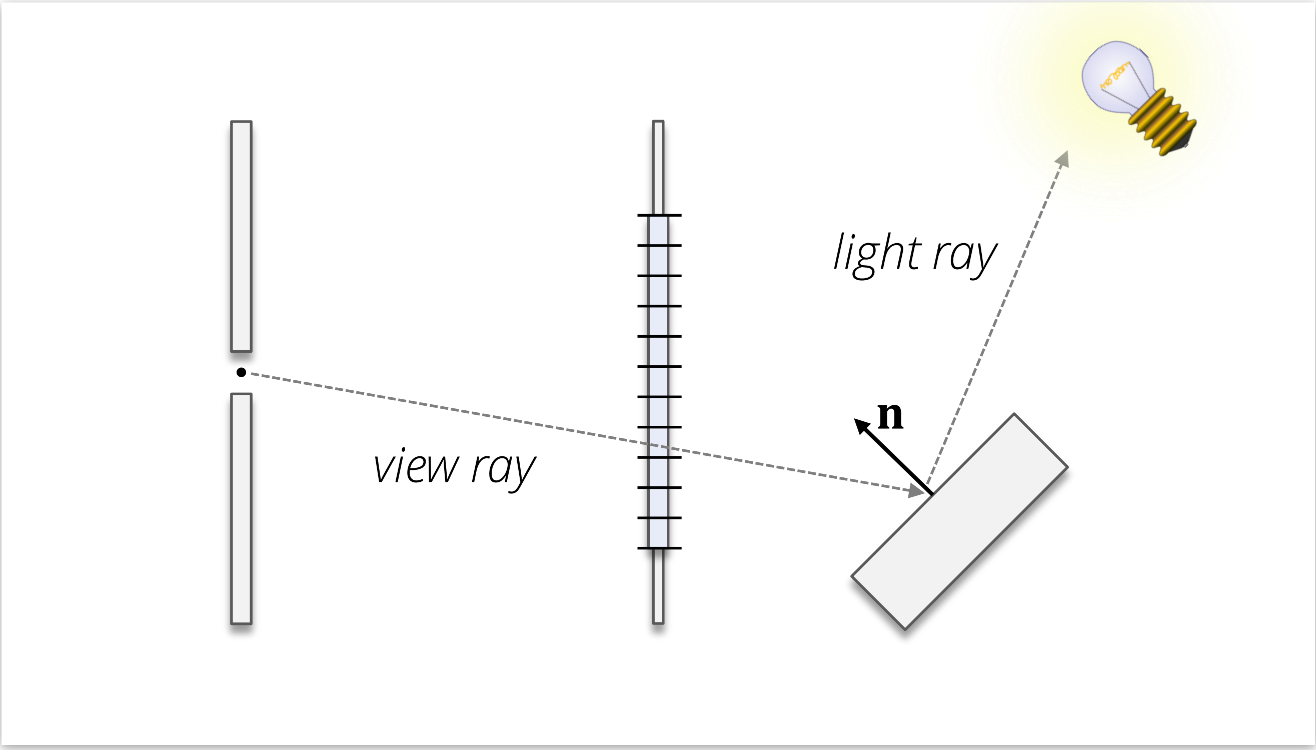
Ray Tracing



Primary Rays

- Rays through each pixel
- Details: Tutorial #6

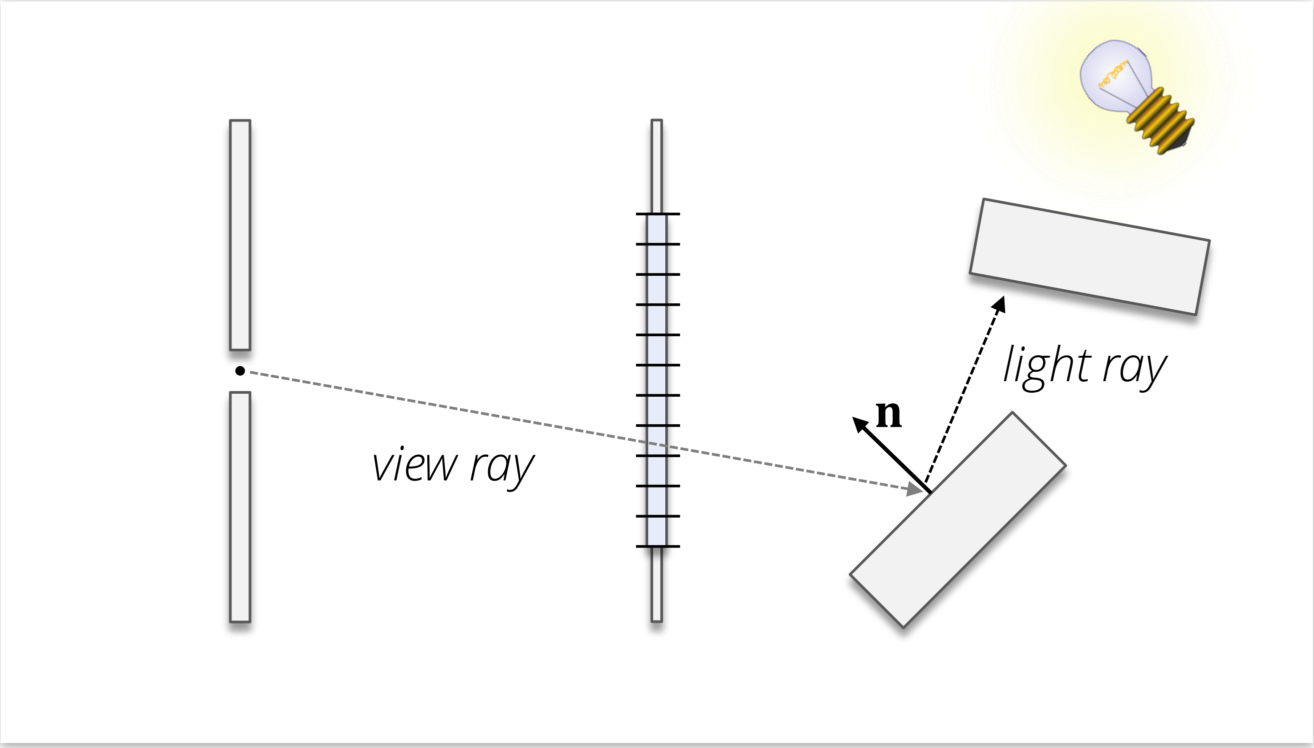
Local Illumination



Primary Rays

- Rays through each pixel
- (Basic trigonometry)

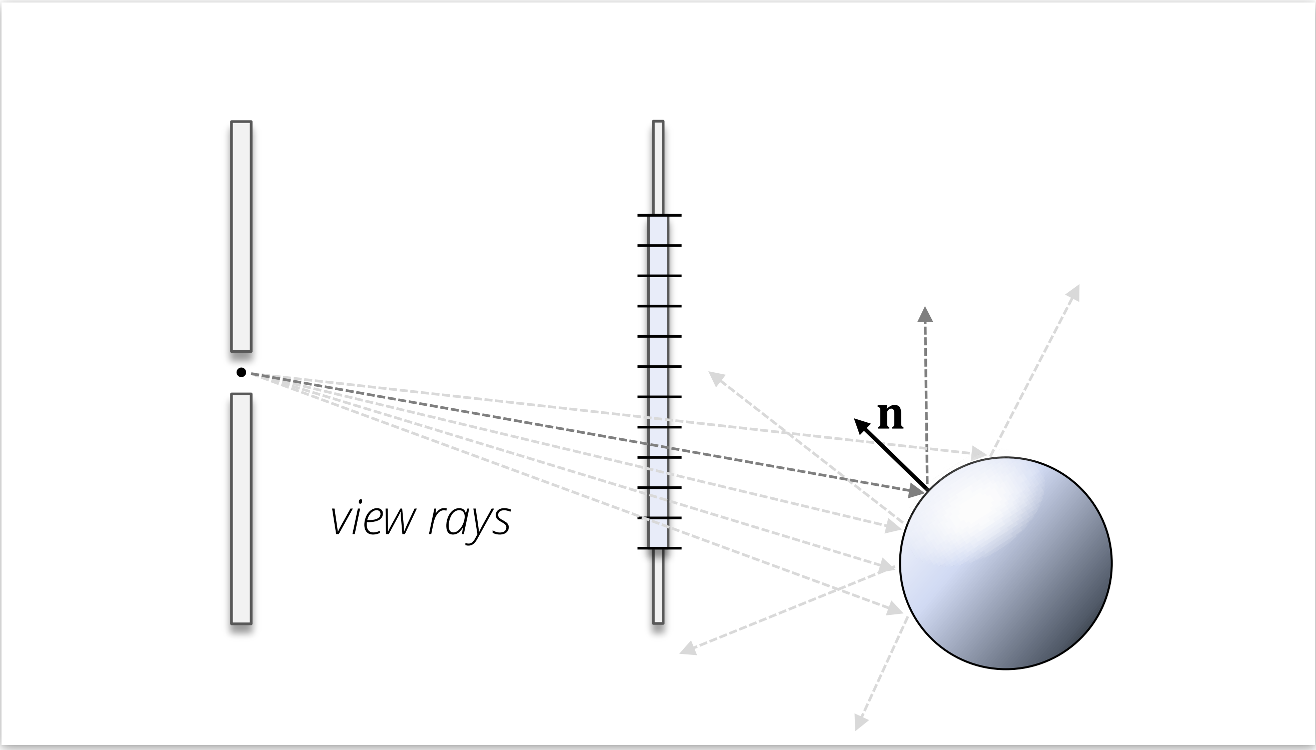
Shadows



Shadow rays

- Blocked by occluders (hard shadows)

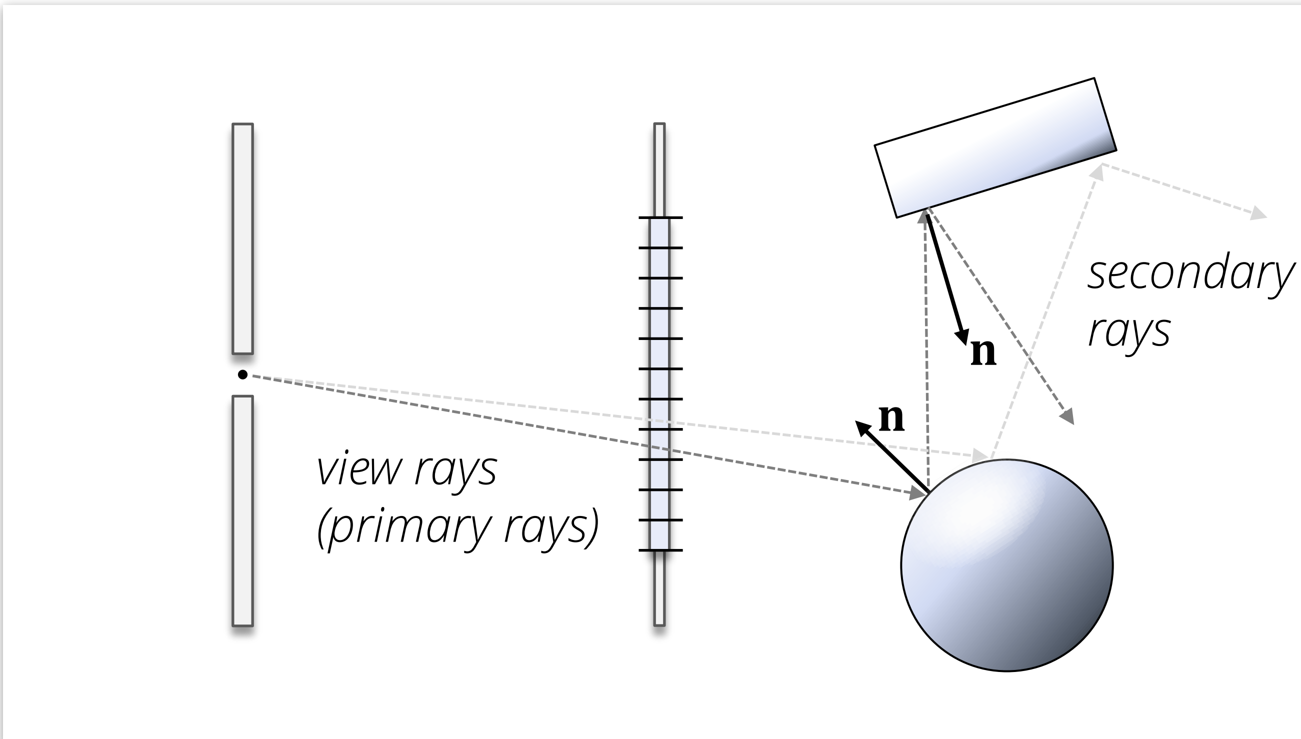
Reflection



Reflection

- Reflect ray across normal at intersection point
- (Basic linear algebra)

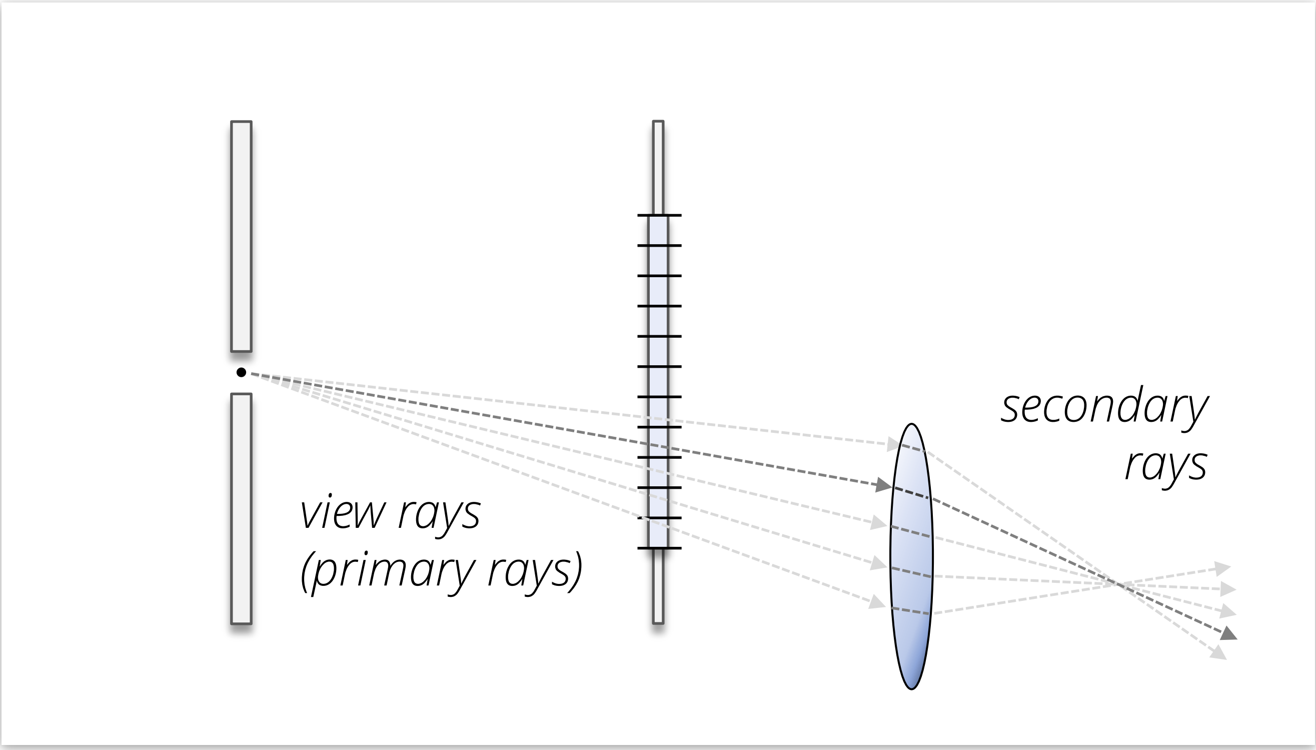
Multiple Reflections: Recursion



Multiple Reflections

- Call algorithm recursively for secondary rays
- (Terminate after *n* levels, for safety)

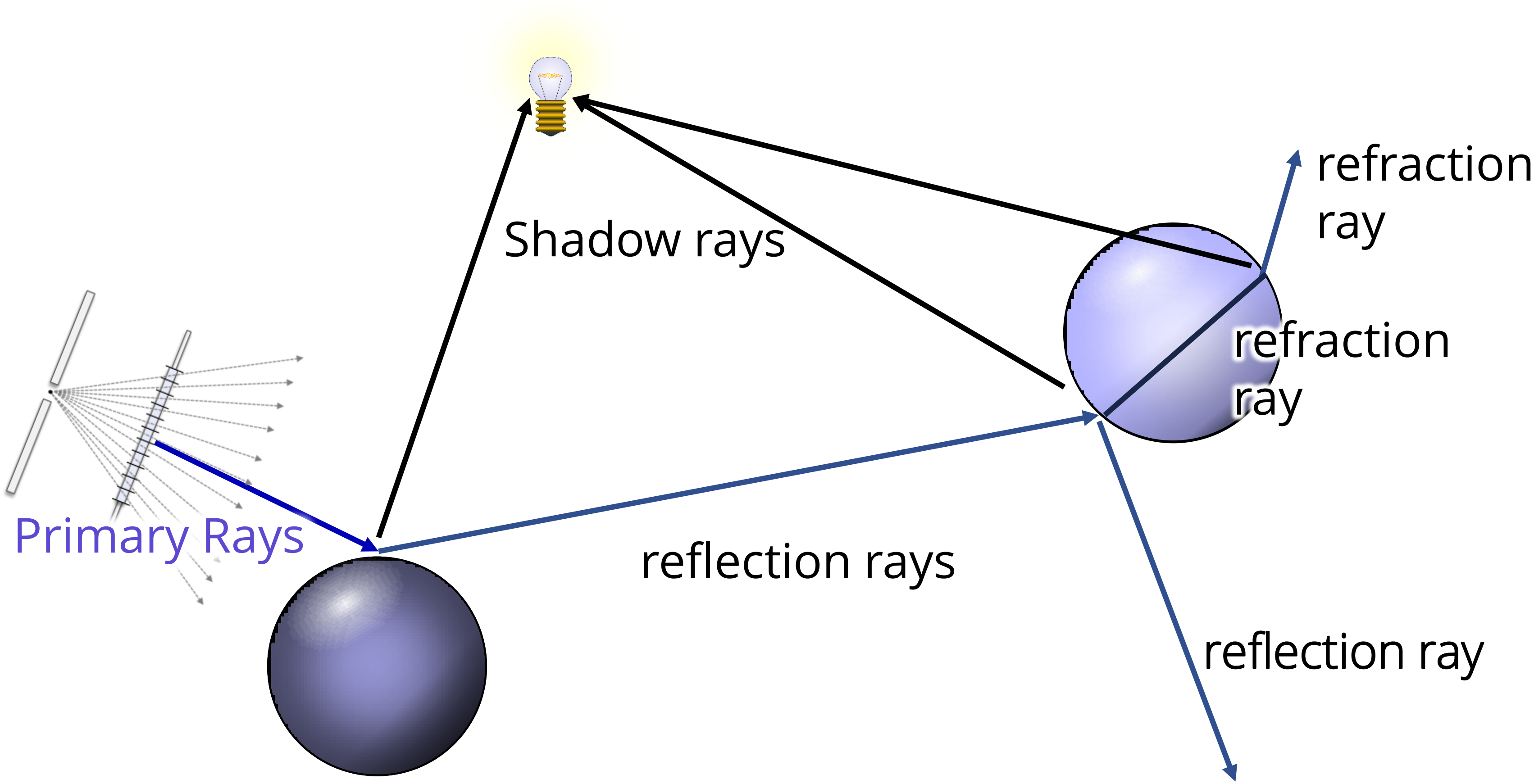
Refraction



Refraction

- Same story
- New rays: Snellius' law

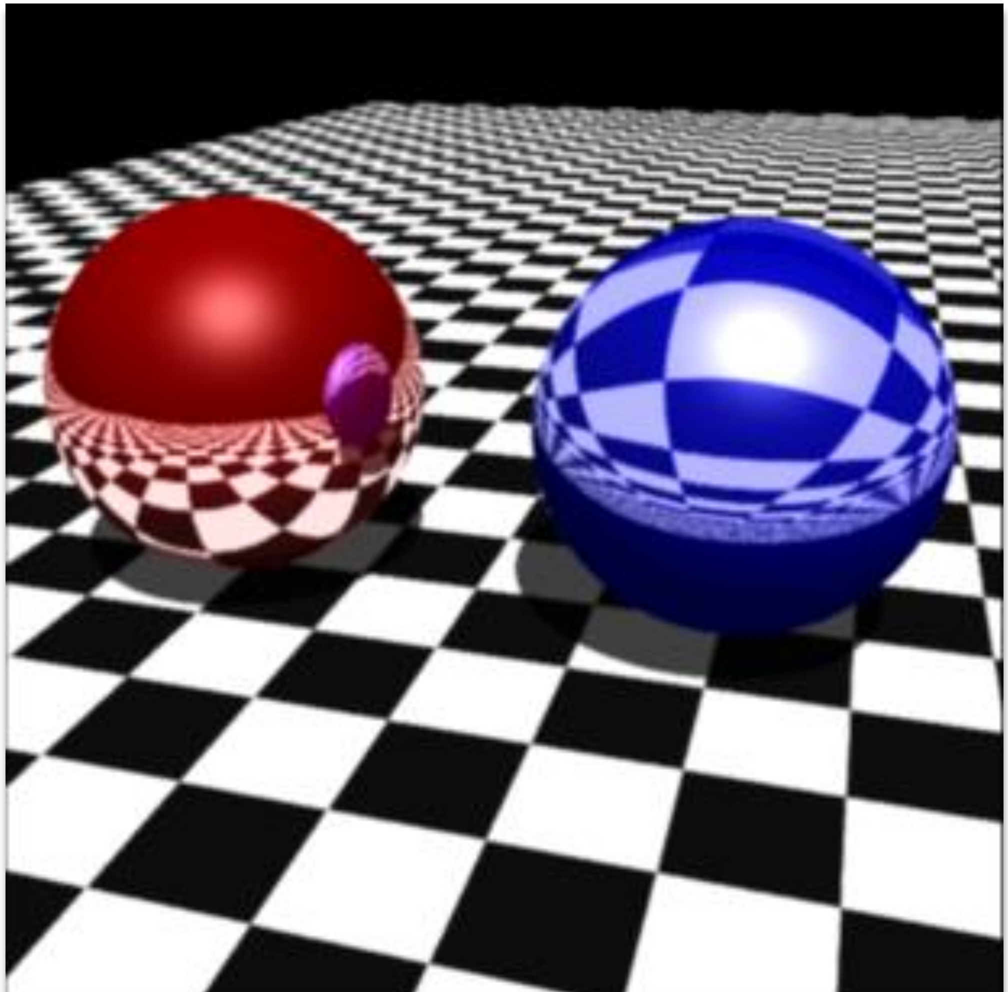
Recursive Raytracing



Worst-case complexity

- $\mathcal{O}(n \cdot m \cdot 2^{r})$
- n = Triangles, m = Pixels, r = maximum recursion depth

Raytracing in a Nutshell



Intersection Tests

CORE

core topics
important

Ray-Triangle Intersection

Equations:

$$\lambda_1 \mathbf{t}_1 + \lambda_2 \mathbf{t}_2 - \mu \cdot \mathbf{t}_r = \mathbf{p}_r - \mathbf{p}_t$$

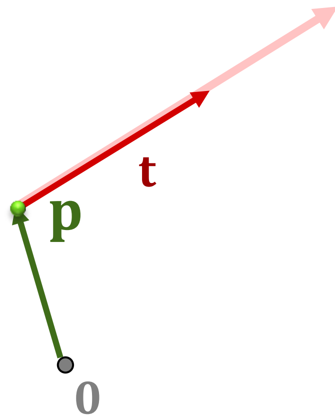
Linear system of equations:

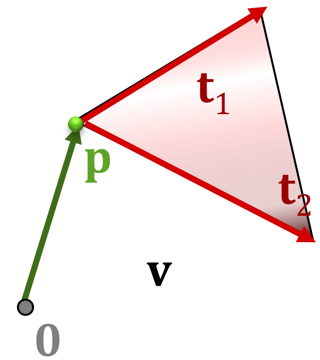
$$\mathbf{M} \cdot \mathbf{x} = \mathbf{b}$$

$$\mathbf{M} = \begin{pmatrix} | & | & | \\ \mathbf{t}_1 & \mathbf{t}_2 & -\mathbf{t}_\mathrm{r} \\ | & | & | \end{pmatrix}, \qquad \mathbf{b} = \mathbf{p}_r - \mathbf{p}_t, \qquad \mathbf{x} = \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \mu \end{pmatrix}$$

→ Gaussian elimination, Kramer's rule

Ray-Triangle Intersection





Parametric line equation:

$$\mathbf{x}(\mu) = \mathbf{p}_r + \mu \cdot \mathbf{t}_r$$
$$\mu \geq 0$$

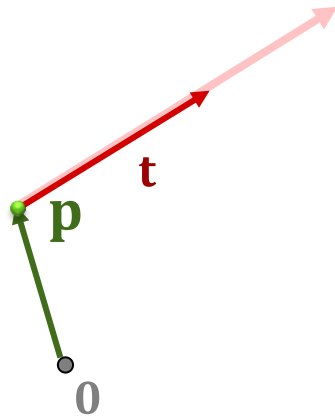
Parametric plane equation:

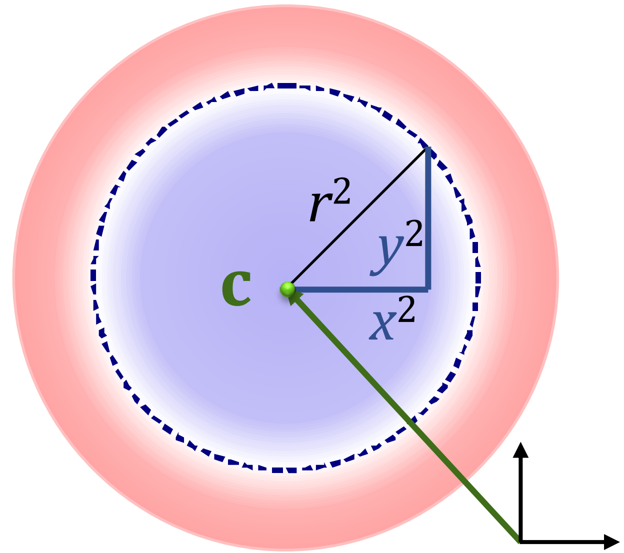
$$\mathbf{x}(\lambda_1, \lambda_2) = \mathbf{p}_t + \lambda_1 \mathbf{t}_1 + \lambda_2 \mathbf{t}_2$$
$$0 \leq \lambda_1 \leq 1,$$
$$0 \leq \lambda_2 \leq 1,$$
$$\lambda_1 + \lambda_2 \leq 1$$

$$\mathbf{x}(\lambda_1, \lambda_2) = \mathbf{x}(\mu)$$
$$\Leftrightarrow \mathbf{p}_r + \mu \cdot \mathbf{t}_r = \mathbf{p}_t + \lambda_1 \mathbf{t}_1 + \lambda_2 \mathbf{t}_2$$

$$\Leftrightarrow \lambda_1 \mathbf{t}_1 + \lambda_2 \mathbf{t}_2 - \mu \cdot \mathbf{t}_r = \mathbf{p}_r - \mathbf{p}_t$$

Ray-Sphere Intersection





Parametric line equation:

$$\mathbf{x}(\mu) = \mathbf{p}_r + \mu \cdot \mathbf{t}_r$$
$$\mu \geq 0$$

Sphere (Implicit!)

$$\langle \mathbf{x} - \mathbf{c}, \mathbf{x} - \mathbf{c} \rangle = r^2$$

$$\langle \mathbf{x}(\mu) - \mathbf{c}, \mathbf{x}(\mu) - \mathbf{c} \rangle - r^2 = 0$$
$$\langle \mathbf{p}_r + \mu \cdot \mathbf{t}_r - \mathbf{c}, \mathbf{p}_r + \mu \cdot \mathbf{t}_r - \mathbf{c} \rangle - r^2 = 0$$

Derivation

Solving the equation:

$$\langle \mathbf{x}(\mu) - \mathbf{c}, \mathbf{x}(\mu) - \mathbf{c} \rangle - r^2 = 0$$

$$(\mathbf{x}(\mu) - \mathbf{c})^2 - r^2 = 0$$

$$(\mathbf{p}_r + \mu \cdot \mathbf{t}_r - \mathbf{c})^2 - r^2 = 0$$

$$\left(\mu \cdot \mathbf{t}_r + (\mathbf{p}_r - \mathbf{c})\right)^2 - r^2 = 0$$

Result: 1D Quadratic equation in μ

$$\mu^2 \cdot \mathbf{t}_r^2 + \mu \cdot 2(\mathbf{t}_r \cdot (\mathbf{p}_r - \mathbf{c})) + (\mathbf{p}_r - \mathbf{c}) - r^2 = 0$$

Spatial Data Structures

Range Queries, Collision Detection

ADV

advanced topics
main ideas

Spatial Data Structures

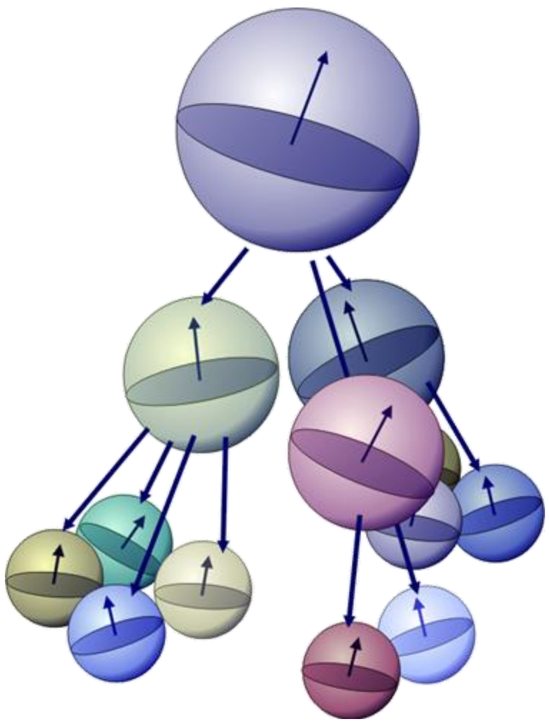
Range Queries

- Common problems
  - Raytracing
  - Select object by mouse click
  - Collision detection
- This should work on large models
  - Scale to billions of primitives
  - Asymptotic complexity

Spatial Data Structures

Basic Idea: Hierarchical decomposition

- If number objects too large:
  - Form spatially coherent groups
  - For each group:
    - Simple bounding volume
    - Apply recursively



Result

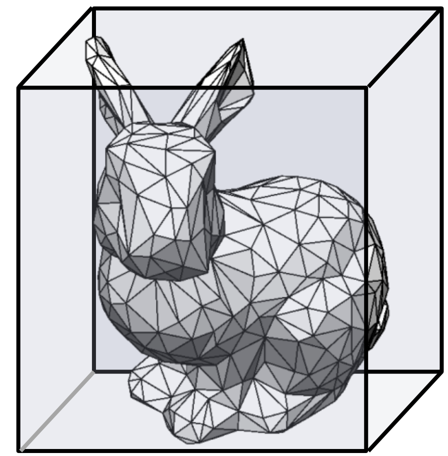
- We obtain a tree of bounding volumes
- “Bounding volume hierarchy”

Bounding Volumes

Axis-Aligned Bounding Box

- Store minimum x,y,z-coord

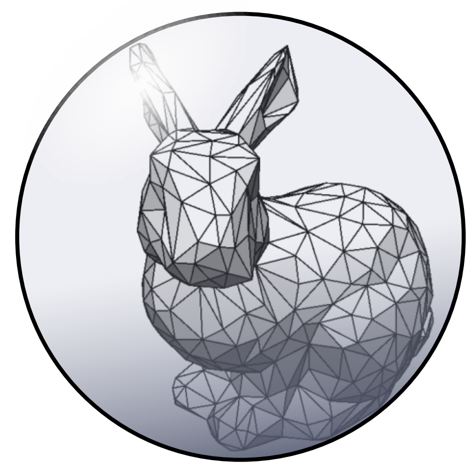
  and
- maximum x,y,z-coord



axis-aligned
bounding box

Bounding Sphere

- Store radius, center
- Such that all geometry
  is contained

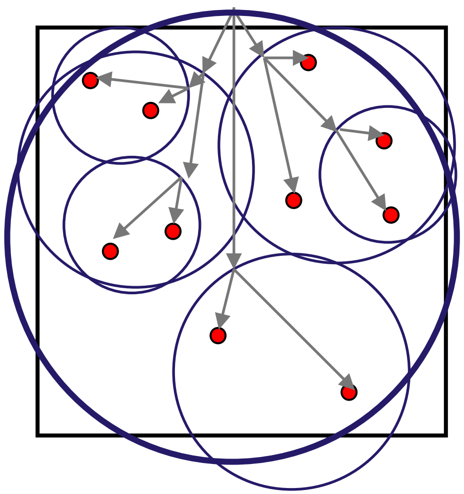


bounding sphere

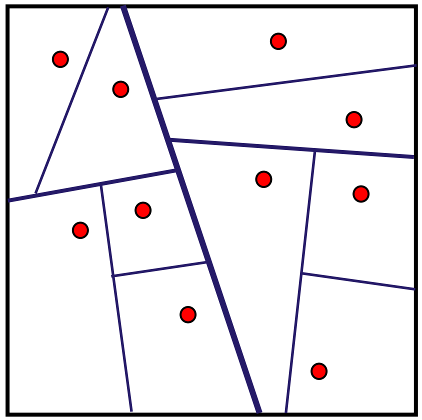
Variants

Variants:

- Bounding volume hierarchy
  - General definition
  - Any bounding volumes
  - Image: spheres



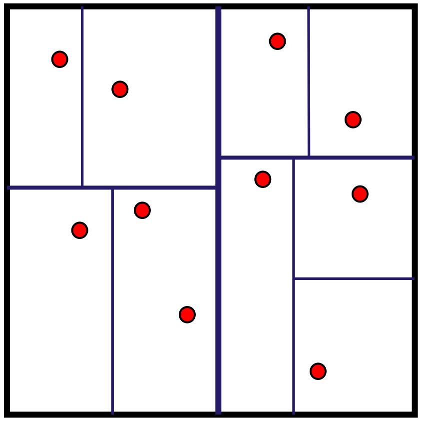
- BSP-tree
  - Split planes (half-spaces)
  - “Binary space partition tree”
  - Arbitrary planes



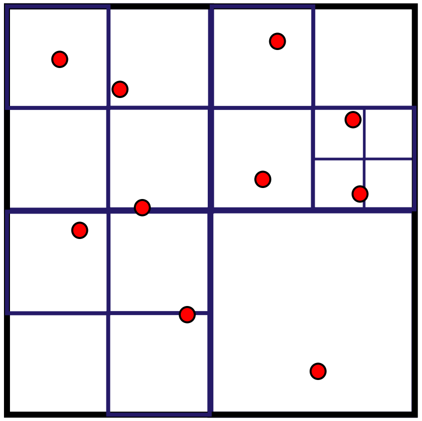
Variants

Variants

- Axis aligned BSP tree / kD-tree
  - Axis-parallel splitting planes
  - Special case: kD-tree
    - Alternating splitting dimensions
    - Median cut:
      split at median coordinate



- Quadtrees / Octrees
  - Divide into 4/8 cubes
  - Special case of the above
    (no binary tree though)



Extended Objects

Extended objects (other than points)

- Extended objects:
  - Triangles
  - Polygons
  - etc…
- Division of space might intersect with object
- Three solutions
  - Split objects (expensive, uncommon)
  - Overlapping nodes (common)
  - Storage multiple times (also common)

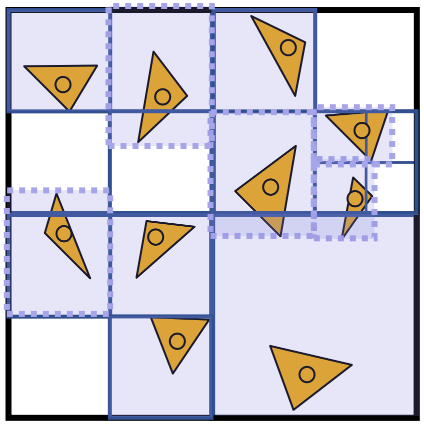
Splitting Objects

First solution: *splitting*

- **Example:** Triangles in BSP tree
  - Split at plane
  - Aim at few splits
- (Rather) easy to see:
  - General BSP tree needs still $\mathcal{O}(n^2)$ fragments (worst case, *n* triangles; practice: $\approx \mathcal{O}(n \log n)$)
  - Lower bound for kD trees, octrees, etc…
- Splitting usually too expensive
  - Used in early low-polygon 3D engines (BSP-visibility)

Overlapping Regions

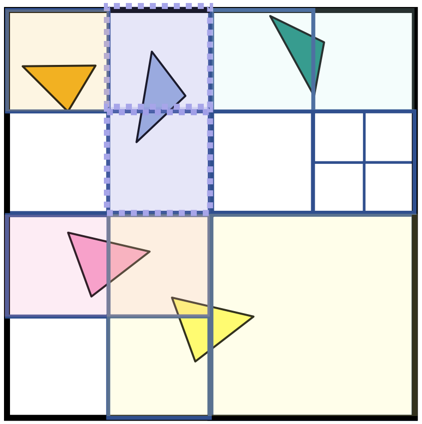
Second Solution: *overlap*



- Permit overlapping bounding volumes
- E.g., second bounding box (octree)
- Possible strategy:
  - Up to 10% oversize (in each direction)
  - No fit into leaf nodes: use an inner node
- Overlap reduces efficiency
  - Multi-coverage of volume
  - 10% in each direction means $1.2^3 \approx 1.7\times$
  - Effect on algorithms might vary

Overlapping Regions

Third Solution: *store multiple times*



- Store primitive multiple times
- Disadvantages
  - Reduced efficiency
  - Additional memory
- Advantages
  - Regular structures
  - No additional bounding boxes
- Common for raytracing

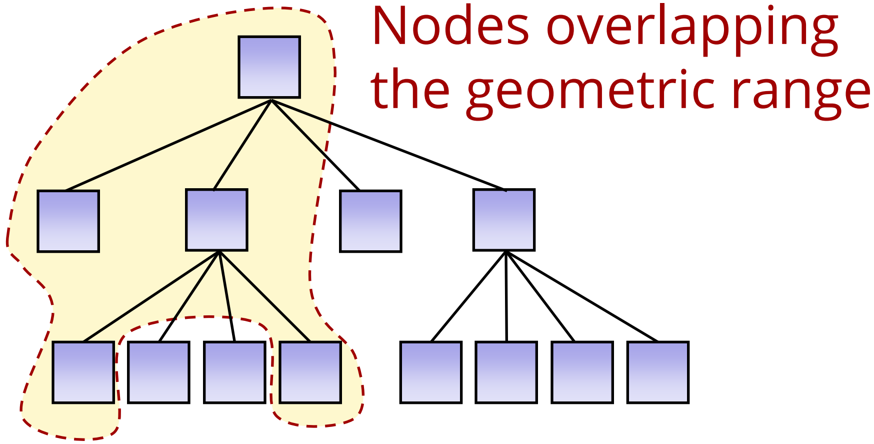
Range Queries

ADV

advanced topics

main ideas

Range Query Algorithm

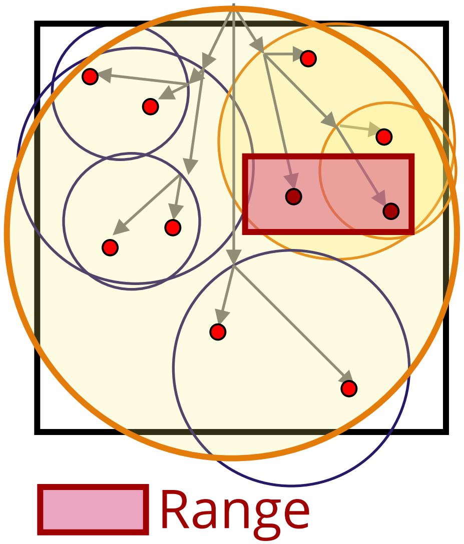


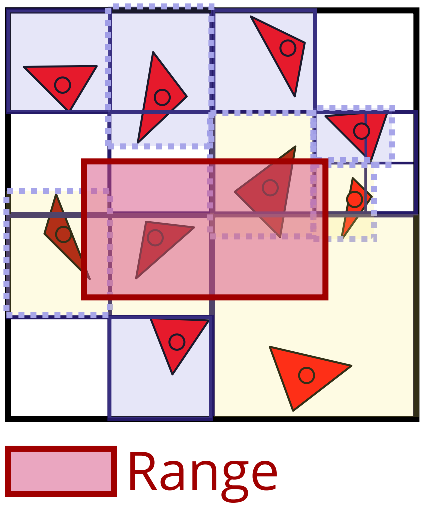
Start at root node: Then, recursively

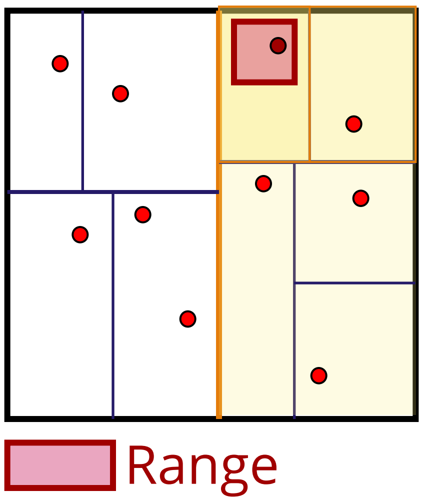
- **If** *range* overlaps *bounding box*
  - Test node primitives
    - Report if within range
  - Call recursively for child nodes
- **If** *range* does not overlap *bounding box*
  - End recursion

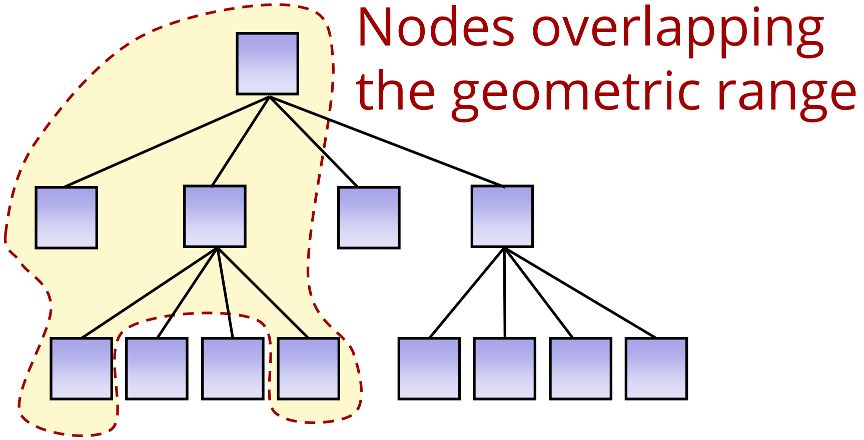
algorithm works for all hierarchy types

Examples

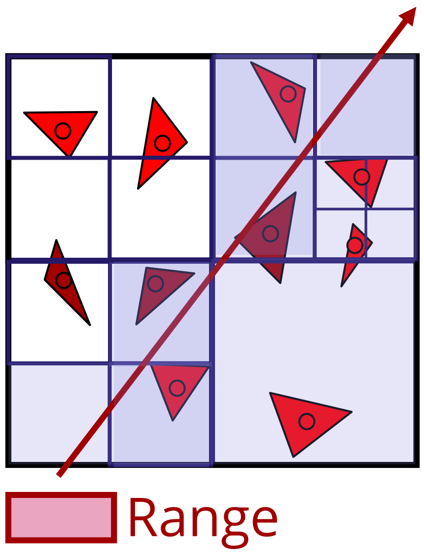








Raytracing



Raytracing: special case

- Ray is the range
- Early ray termination
  - Sorted recursion (child closer to the camera: first)
  - Stop after hit

In Practice

Significant Speedup

- My own, simple implementation
  - Axis-aligned BSP tree
  - Single-core C++
  - 1.000.000 triangle scene
  - ~500.000 triangle-ray intersections per second
- If you work harder…
  - Optimized software ~15M
  - GPU implementations up to 100M
  - Optimized versions:
    Performance also depends on ray coherence