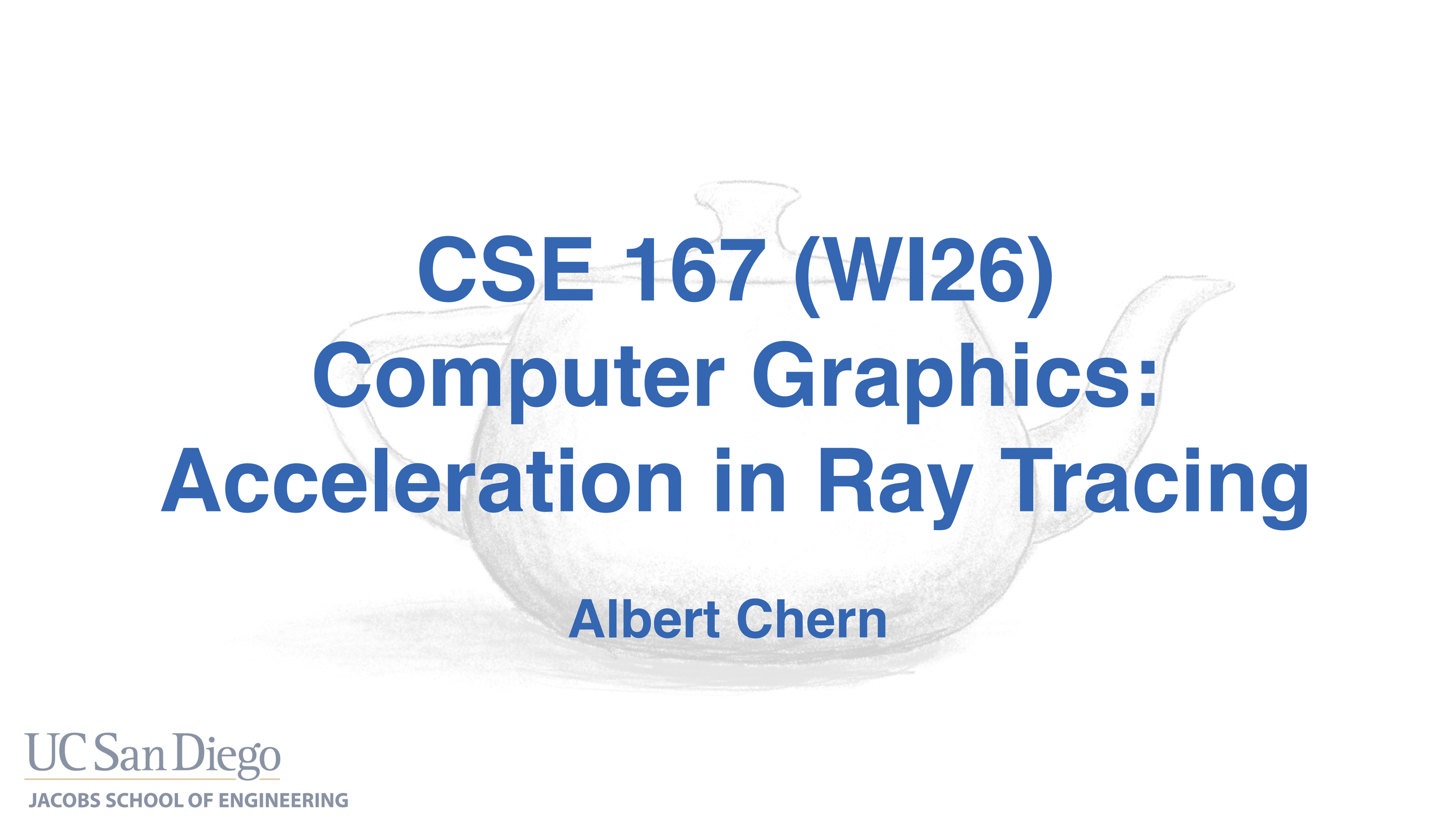


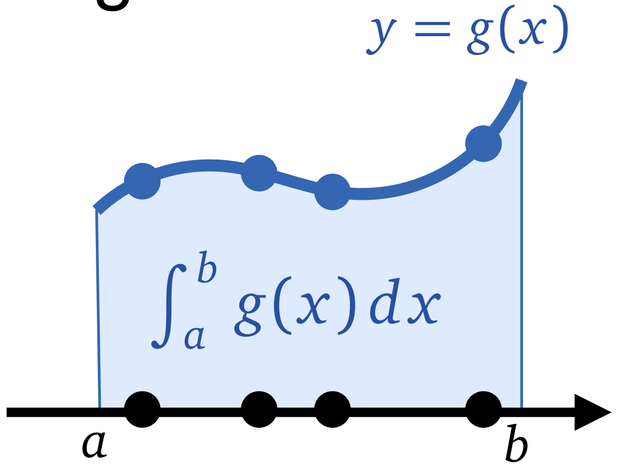
CSE 167 (WI26)
Computer Graphics: Acceleration in Ray Tracing

Albert Chern

UC San Diego
JACOBS SCHOOL OF ENGINEERING

Monte Carlo method

- The **Monte Carlo method** is a way to estimate an integral.
- Suppose we are given a function $g : [a, b] \to \mathbb{R}$
- **Task:** compute $I = \int_a^b g(x)\,dx$



- Monte Carlo method: Sample *N* random points $x_1, x_2, \ldots, x_N$ independently and uniformly from the domain [a,b].
- For each sample, the integral is estimated as $I_i = (b-a)g(x_i)$
- The average would converge to the true solution

$$\frac{1}{N}\sum_{i=1}^{N} I_i \xrightarrow{N\to\infty} I$$

Monte Carlo method

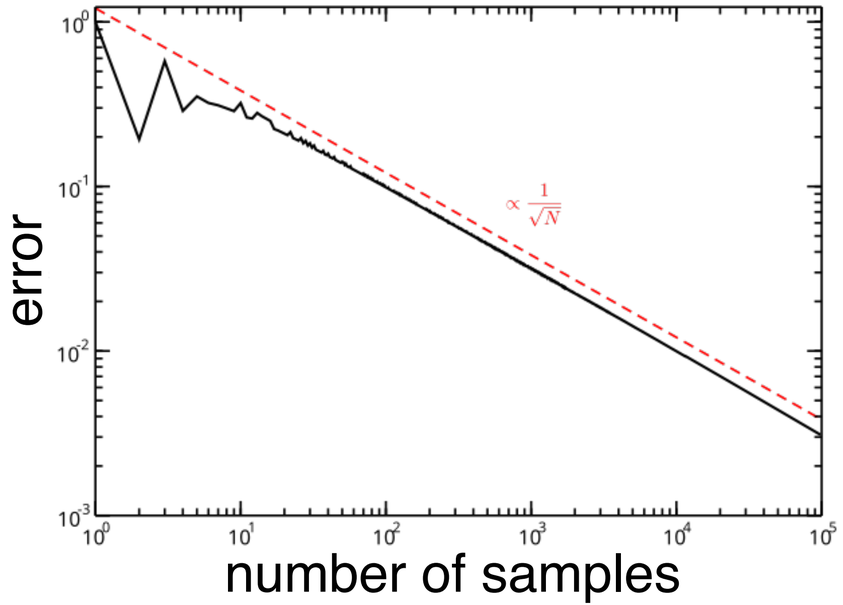
- The average would converge to the true solution

$$\frac{1}{N}\sum_{i=1}^{N} I_i \xrightarrow{N\to\infty} I$$

- More precisely, $\left(\frac{1}{N}\sum_{i=1}^{N} I_i\right) - I \approx \frac{\sqrt{\mathrm{Var}(I_i)}}{\sqrt{N}}$

- We don't need to know anything about g

- But the convergence is slow.



Ray tracing

- Recall rendering equation

$$\begin{cases} L_{\text{out}}(\mathbf{x}, \mathbf{v}_{\text{out}}) = L_{\text{e}}(\mathbf{x}, \mathbf{v}_{\text{out}}) + \iint_{H_{\mathbf{x}}} f(\mathbf{x}, \mathbf{v}_{\text{out}}, \mathbf{v}_{\text{in}}) L_{\text{in}}(\mathbf{x}, \mathbf{v}_{\text{in}}) \, d\sigma^{\perp}_{\mathbf{v}_{\text{in}}} \\ L_{\text{in}}(\mathbf{x}, \mathbf{v}_{\text{in}}) = L_{\text{out}}\big(\text{Cast}(\mathbf{x}, \mathbf{v}_{\text{in}})\big) \end{cases}$$

  - L_{out} is the radiance of outgoing light (at a point and direction)
  - L_{in} is the radiance of incoming light
  - L_{e} is the radiance of emitting light
  - f is the BRDF

Ray tracing

- Recall rendering equation

$$\begin{cases} L_{\text{out}}(\mathbf{x}, \mathbf{v}_{\text{out}}) = L_{\text{e}}(\mathbf{x}, \mathbf{v}_{\text{out}}) + \iint_{H_{\mathbf{x}}} f(\mathbf{x}, \mathbf{v}_{\text{out}}, \mathbf{v}_{\text{in}}) L_{\text{in}}(\mathbf{x}, \mathbf{v}_{\text{in}}) \, d\sigma^{\perp}_{\mathbf{v}_{\text{in}}} \\ L_{\text{in}}(\mathbf{x}, \mathbf{v}_{\text{in}}) = L_{\text{out}}\big(\text{Cast}(\mathbf{x}, \mathbf{v}_{\text{in}})\big) \end{cases}$$

- The final pixel color is the integral of incoming radiance over the pixel.
- Apply the Monte Carlo method.
- Integrals over pixel and integrals over the hemisphere $\iint_{H_{\mathbf{x}}}$ are replaced by one random ray sampling.

Bottlenecks of ray tracing

- Ray-geometry intersection (geometry query)

$$L_{\text{in}}(\mathbf{x}, \mathbf{v}_{\text{in}}) = L_{\text{out}}\big(\text{Cast}(\mathbf{x}, \mathbf{v}_{\text{in}})\big)$$

- Slow convergence of averaging random samples (slow convergence of Monte Carlo method)

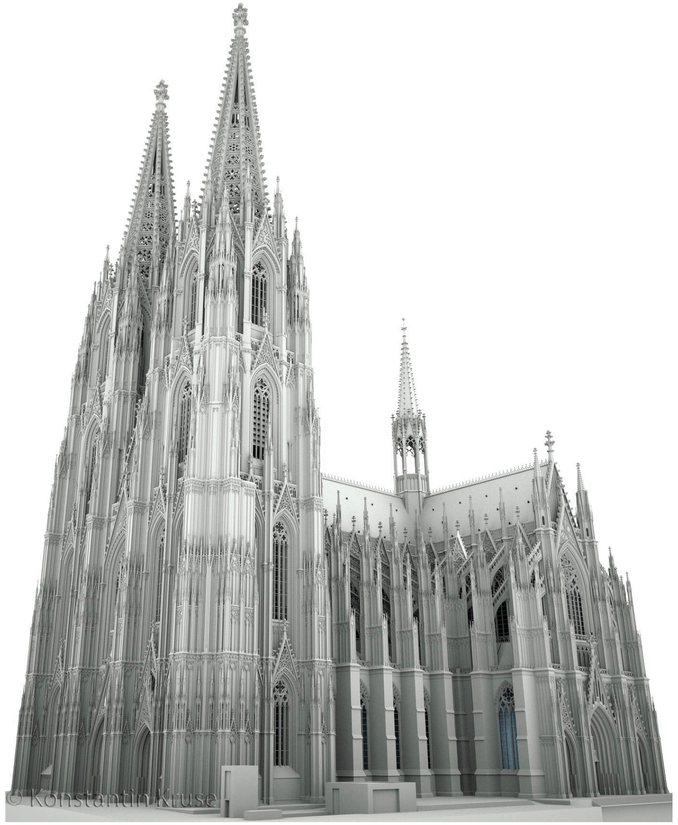
$$\left(\frac{1}{N}\sum_{i=1}^{N} I_i\right) - I \approx \frac{\sqrt{\text{Var}(I_i)}}{\sqrt{N}}$$

Accelerating geometry query

- Accelerate geometry query
- Accelerate Monte Carlo

Geometric query problem

- Complex geometry or large environment
- How can we perform geometry query efficiently? (e.g. for ray tracing)



Cologne Cathedral 3D model



Minecraft video game

In ray tracing

Given a ray 

expensive

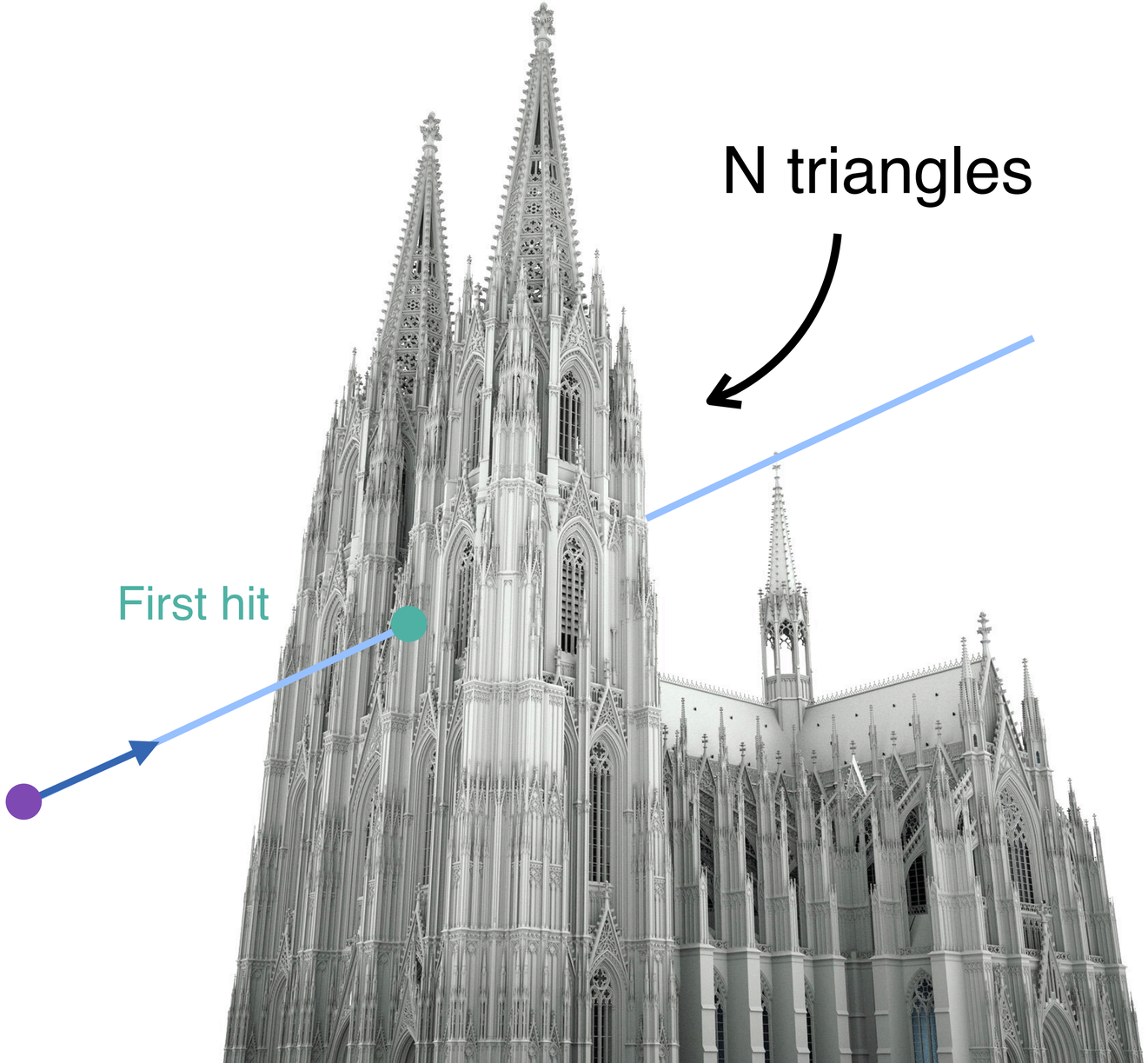
For each  in the scene

Intersect( , )

Keep the closest hit

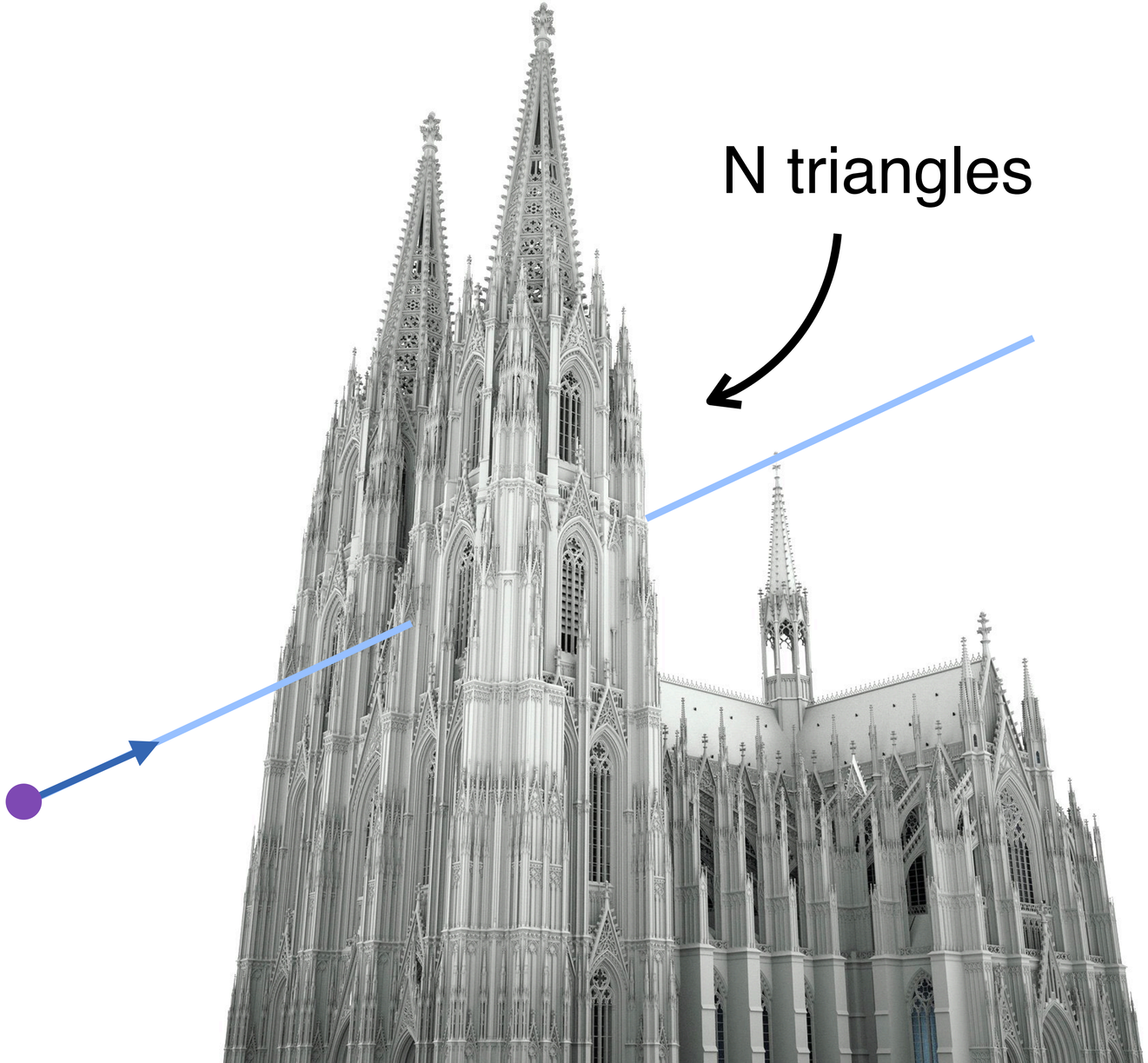
EndFor

- Complexity: O(N)
- Can we do better?



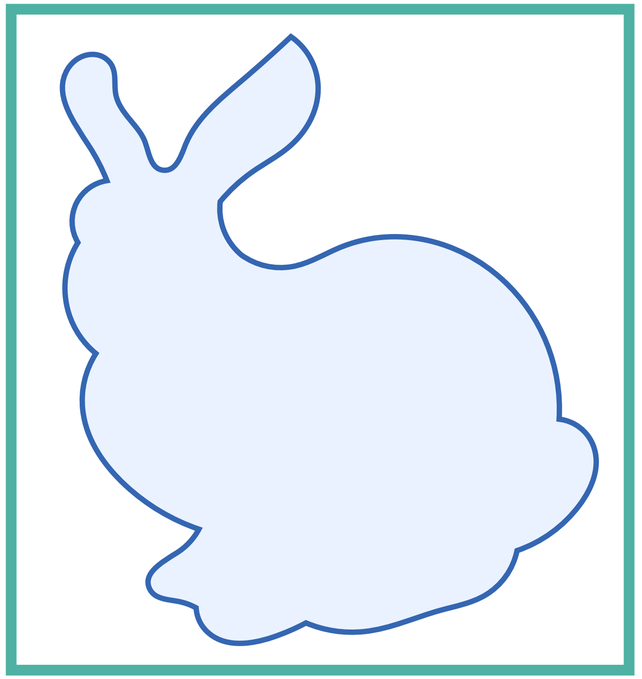
In ray tracing

- Ray only occupies a narrow beam
- We don’t need to loop over all triangles
- Need good data structure to organize triangles

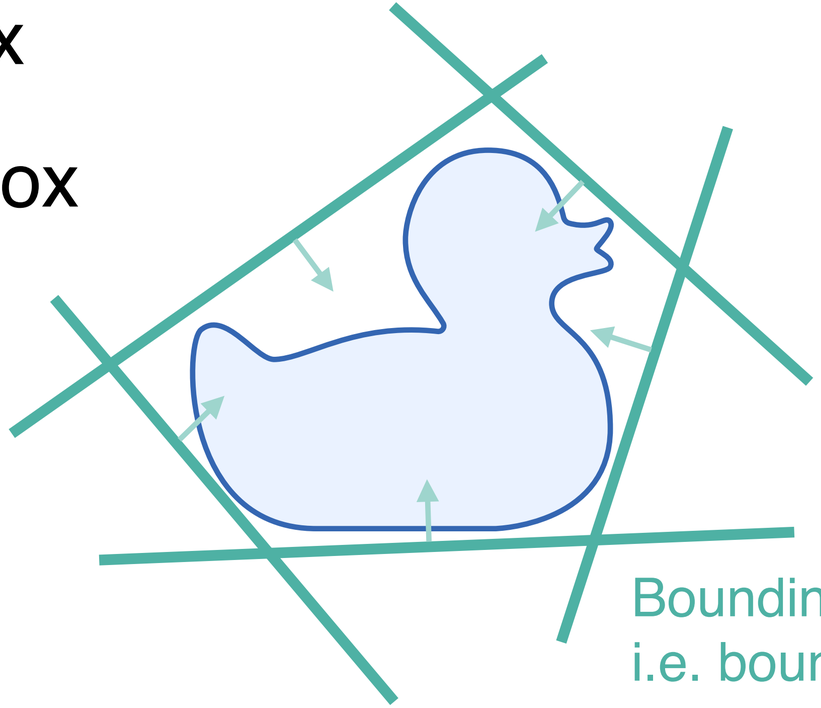


Idea: Simpler bounding geometry

- Bound geometries with simpler shapes such as box, sphere etc.
- Test if ray intersects with bounding box (cheap)
- If the ray does hit the bounding box, query triangles in the box
- Skip all triangles in the box if the ray misses the box
- Worst case scenario is still O(N)…



Bounding box



Bounding convex shape
i.e. bounding half-spaces

How do we speed it up?

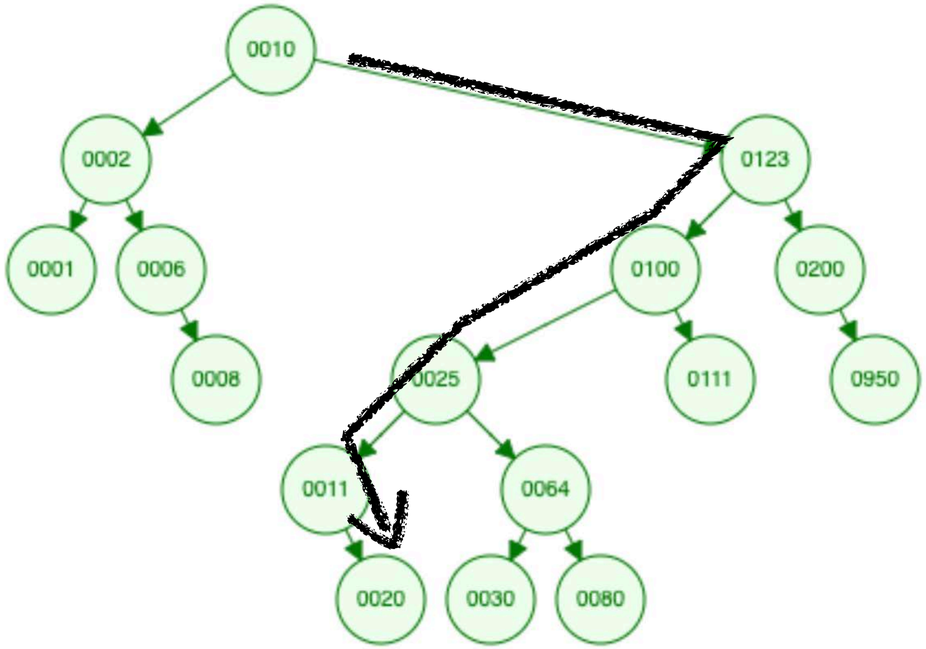
$$O(N) \dashrightarrow O(\log N)$$

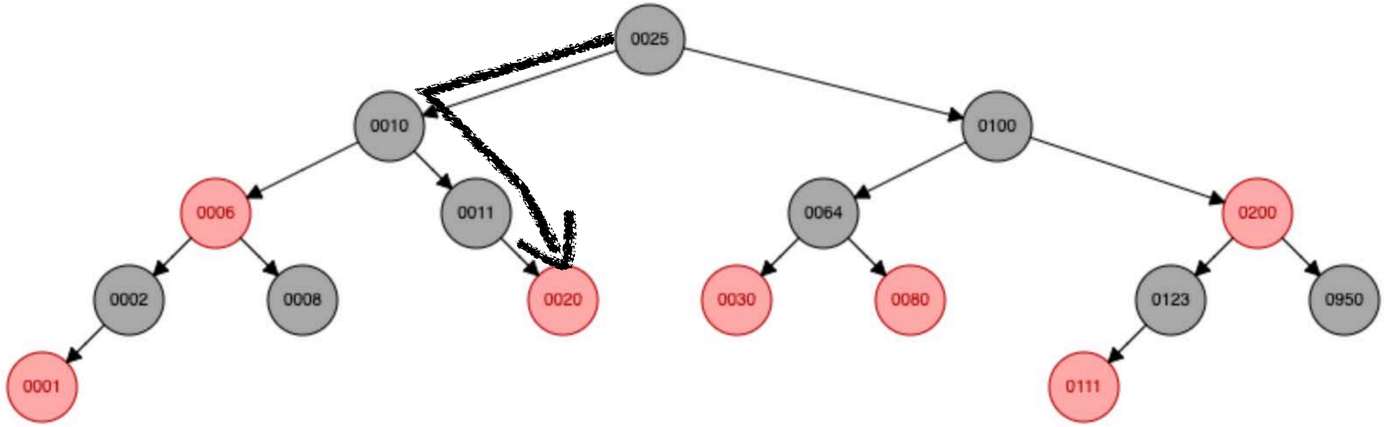
A simpler but similar problem

- Suppose we have set of integers **S**

10 123 2 100 6 25 64 11 200 30 950 111 20 8 1 80

- Given an integer, say $k = 18$, find the element in **S** closest to k
- First, sort the set into a binary search tree O(N log N)





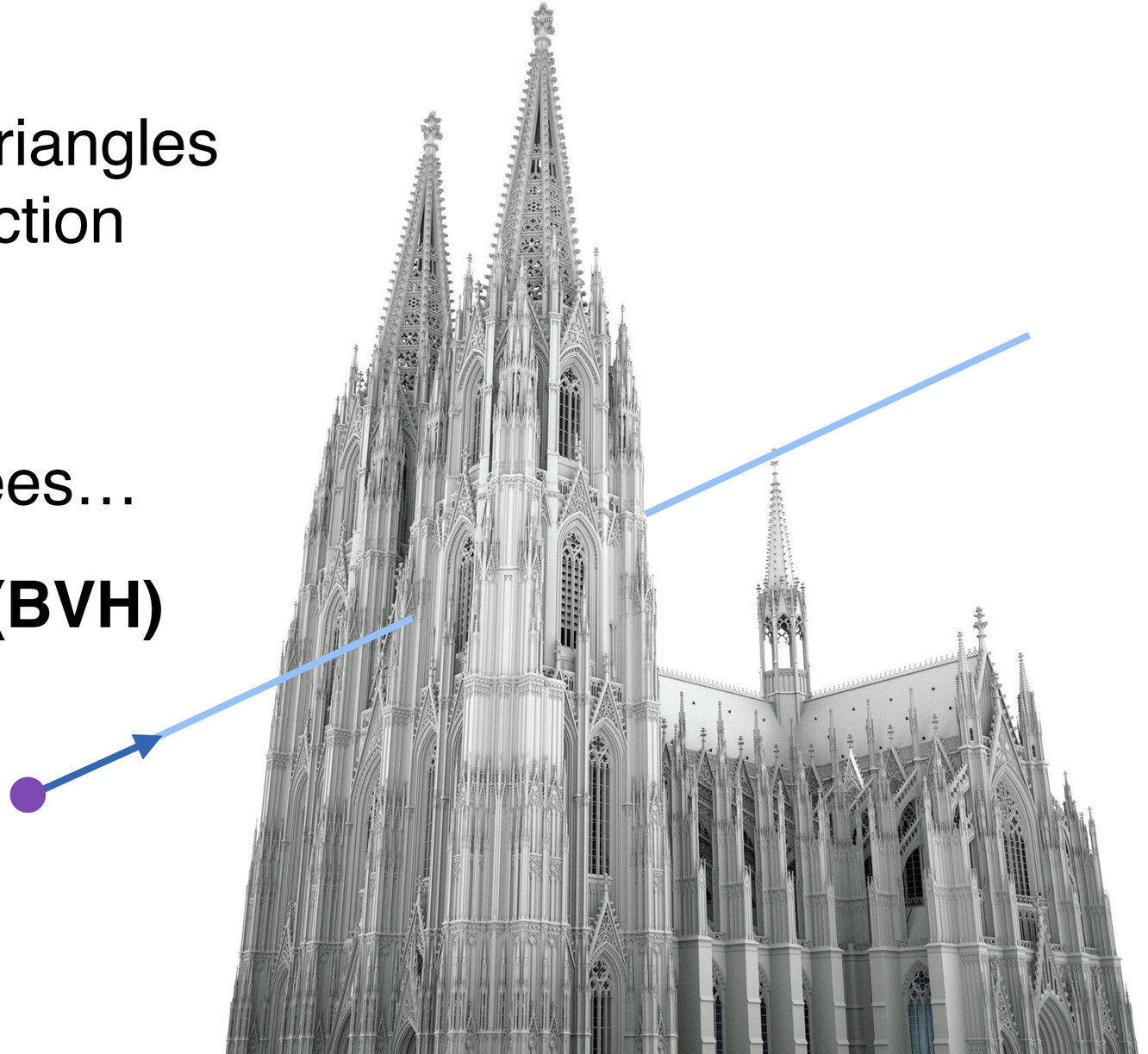
Key: *Hierarchy*

- Then any query would take O(log N)

Fast geometry query

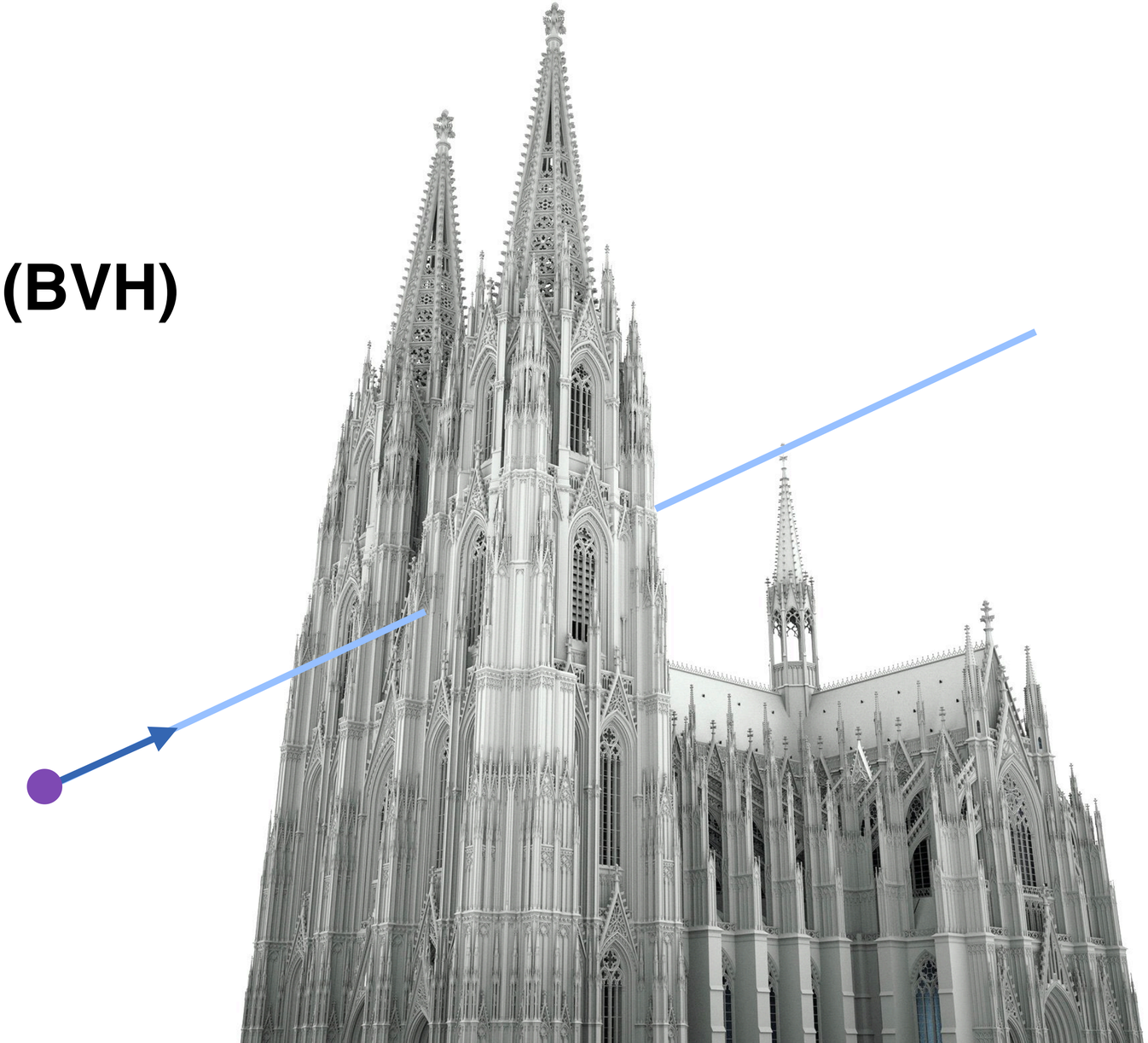
- Can we also reorganize scene triangles to enable fast ray-scene intersection queries?

- Inspired by hierarchal search trees…
  - **Bounding volume hierarchy (BVH)**
  - **k-D tree**
  - **Octree**



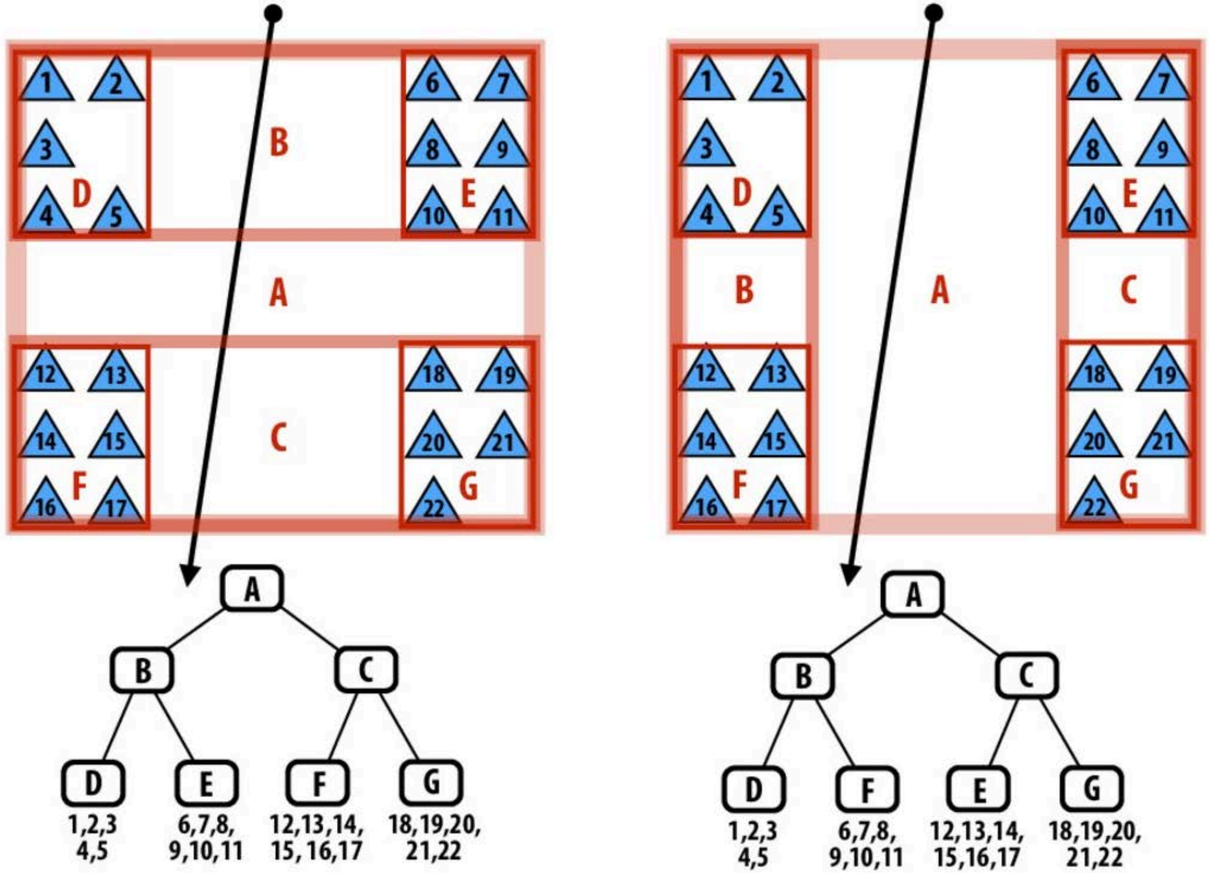
Fast geometry query

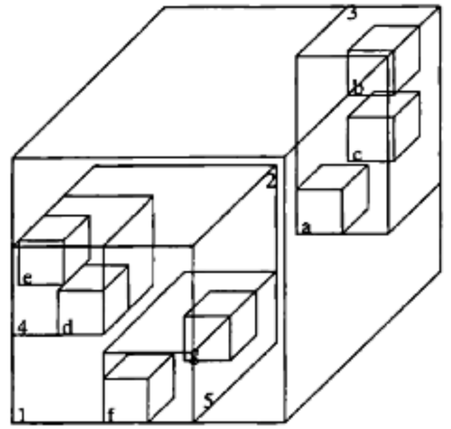
- **Bounding volume hierarchy (BVH)**
- **k-D tree**
- **Octree**

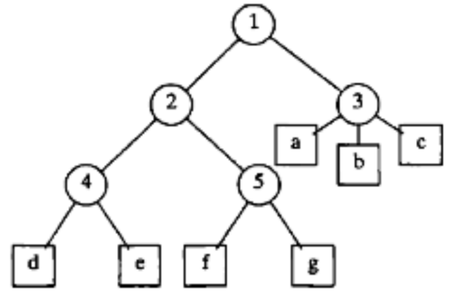


Bounding Volume Hierarchy (BVH)

- Leaf nodes: small number of primitives
- Interior nodes: store sub-bounding boxes

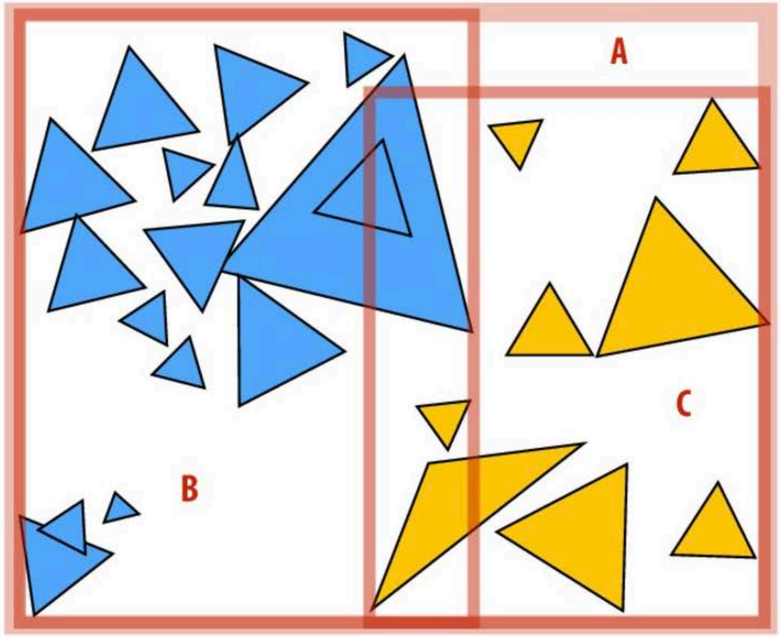


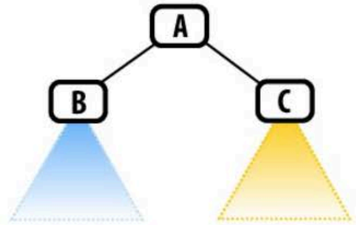




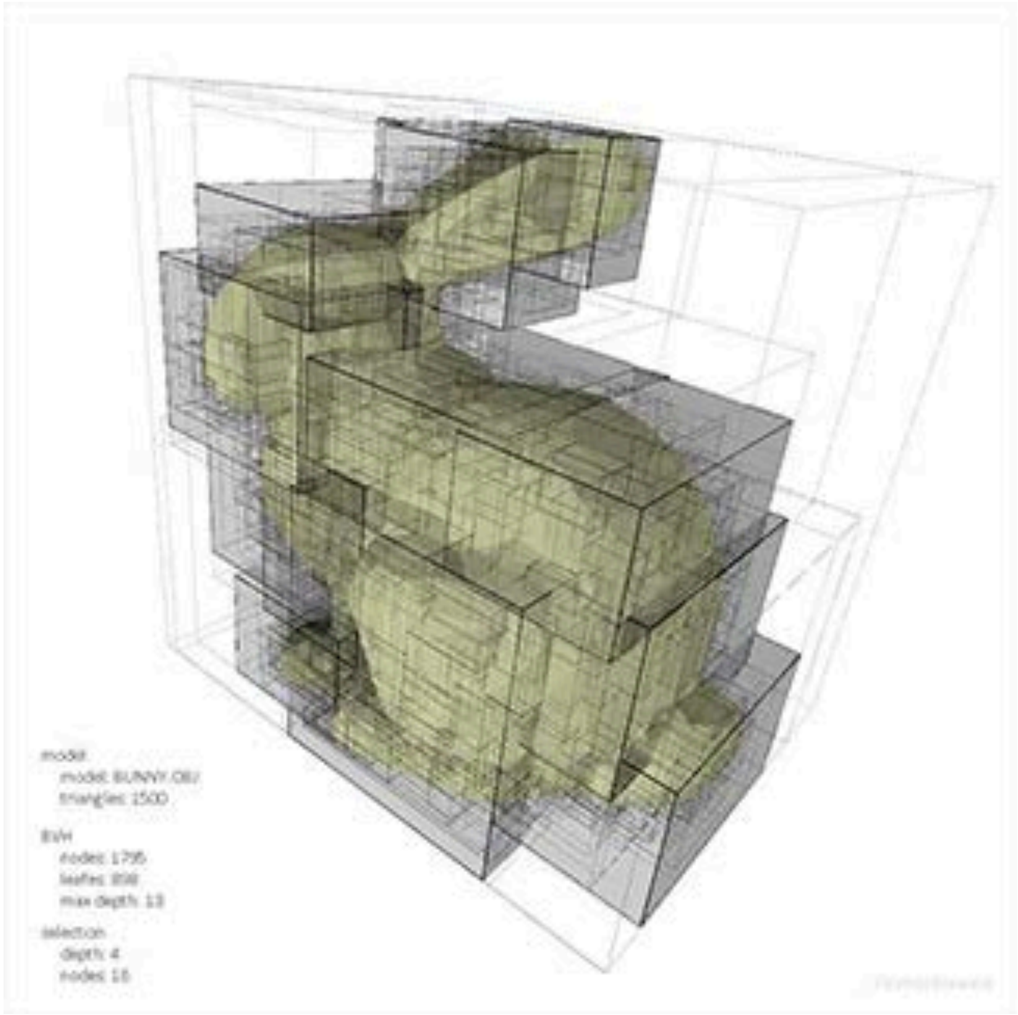
Bounding Volume Hierarchy (BVH)

- Subtrees can overlap



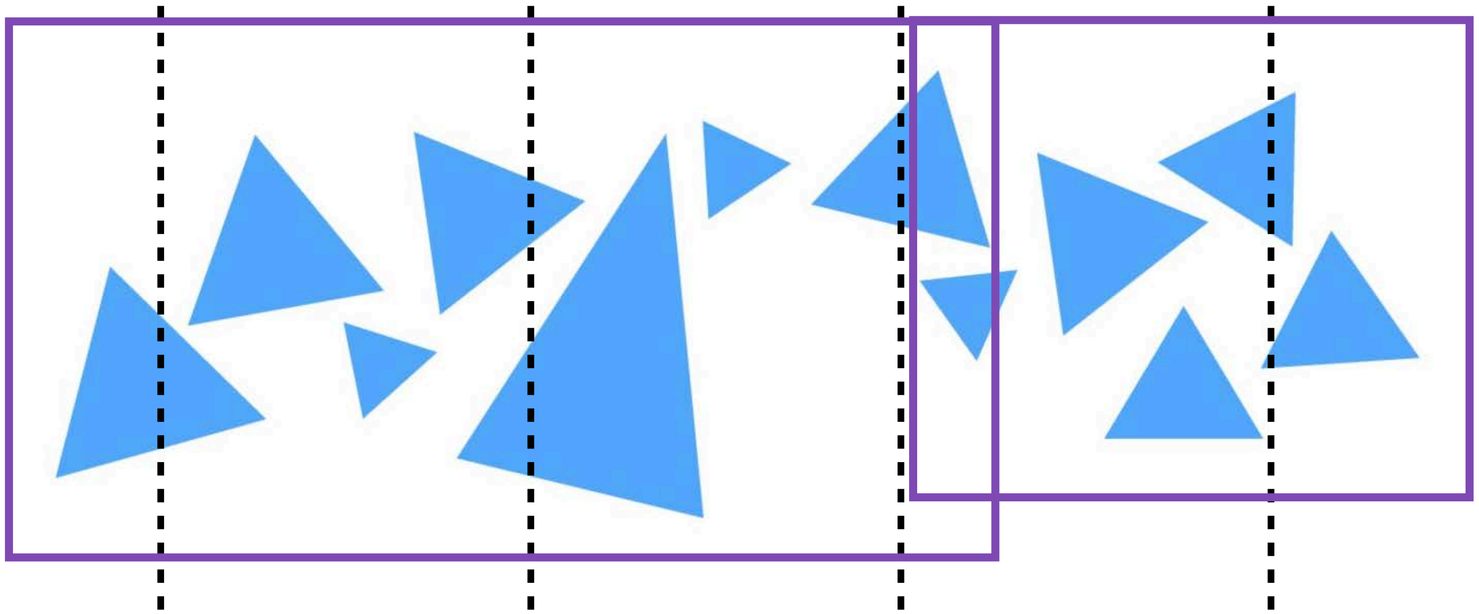


Bounding Volume Hierarchy (BVH)



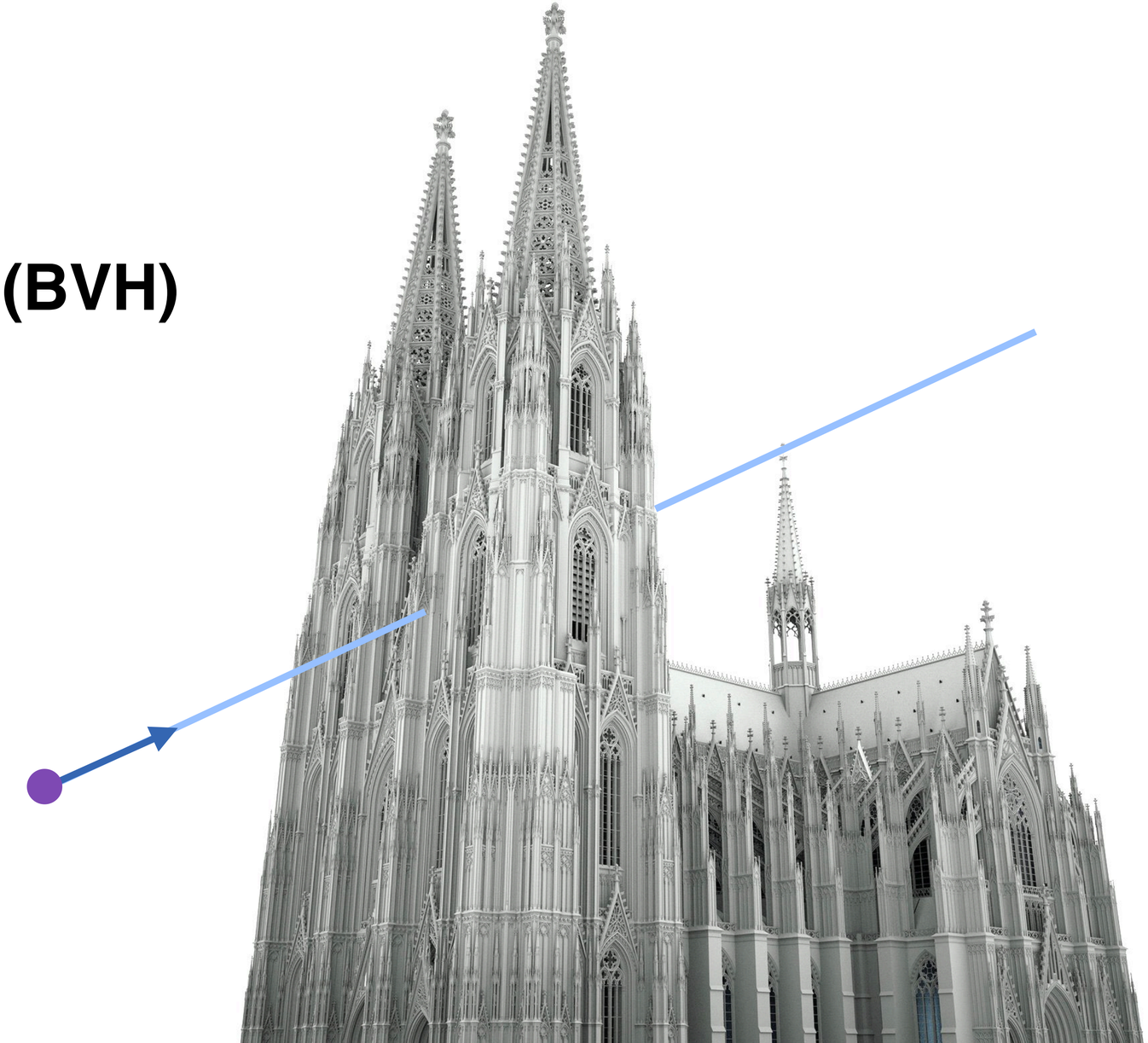
Bounding Volume Hierarchy (BVH)

- To build BVH
- Repeatedly partition each group into two sub-groups
  - To fairly partition group, set up some score based on position
  - Based on histogram, evenly partition elements into two groups



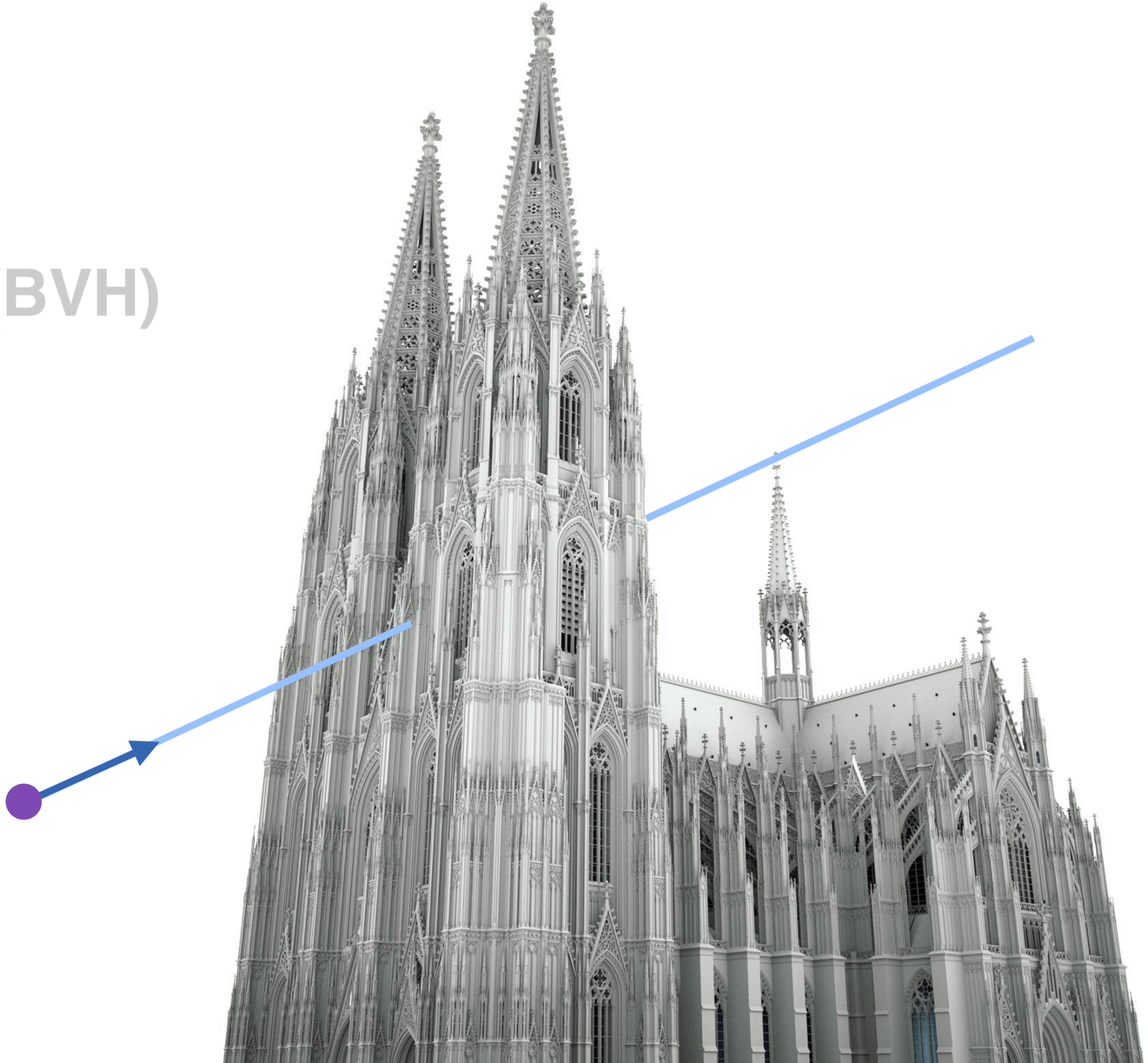
Fast geometry query

- **Bounding volume hierarchy (BVH)**
- k-D tree
- Octree

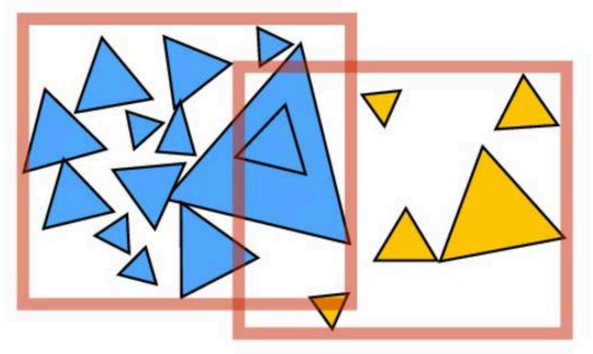


Fast geometry query

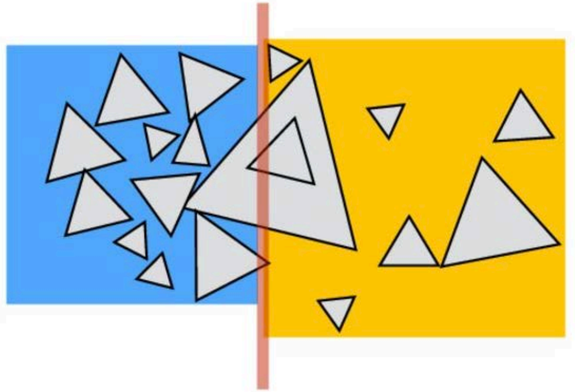
- Bounding volume hierarchy (BVH)
- **k-D tree**
- Octree



Spatial partitioning



Primitive-based partitioning (e.g. Bounding volume hierarchy)

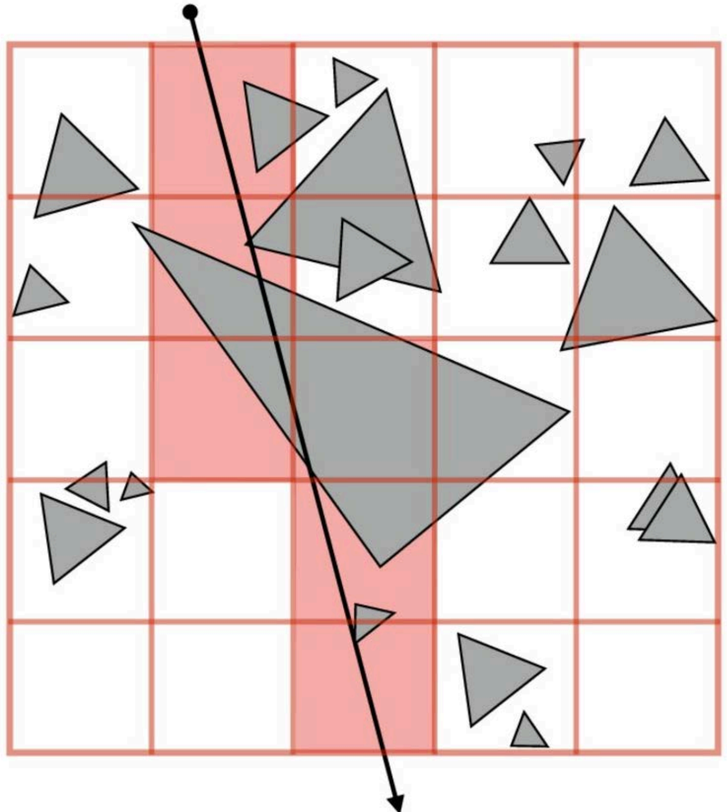


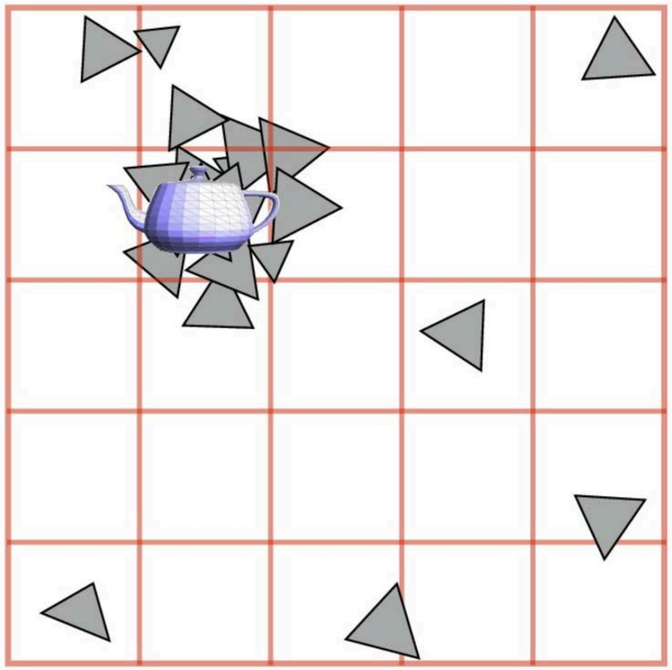
Space-based partitioning

- Partition space into disjoint regions

Spatial partitioning

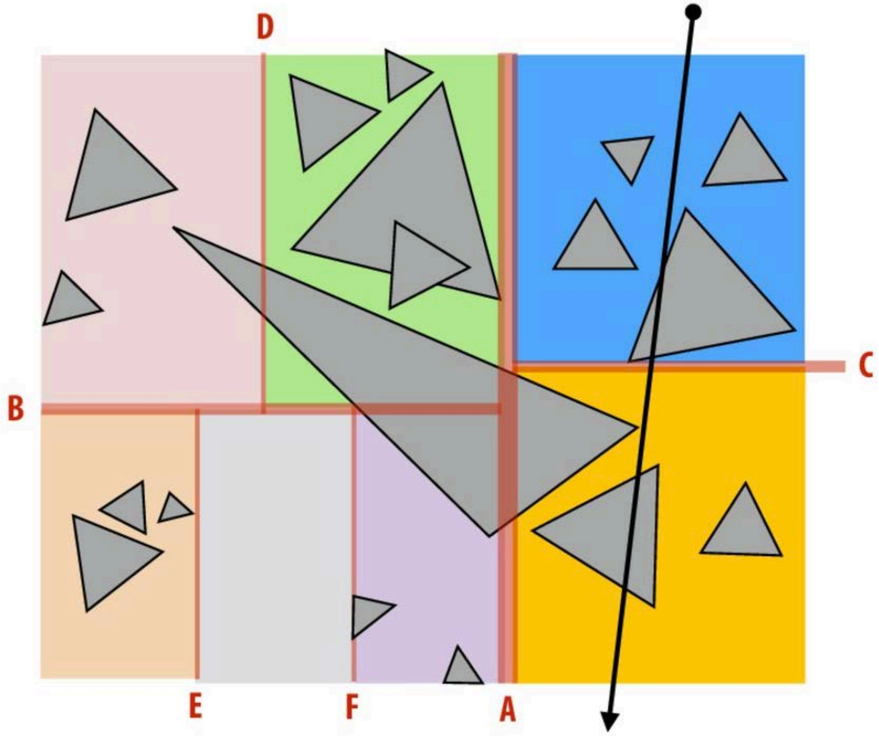
- For example, partition space into uniform grid
- We can march the ray cell-by-cell and stop whenever we hit a triangle
  - For BVH there is no obvious ordering among bounding volumes
  - But each cell may contain much more nodes than BVH

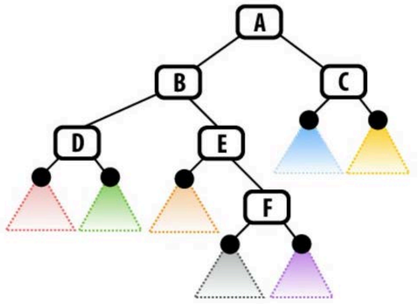




k-D tree

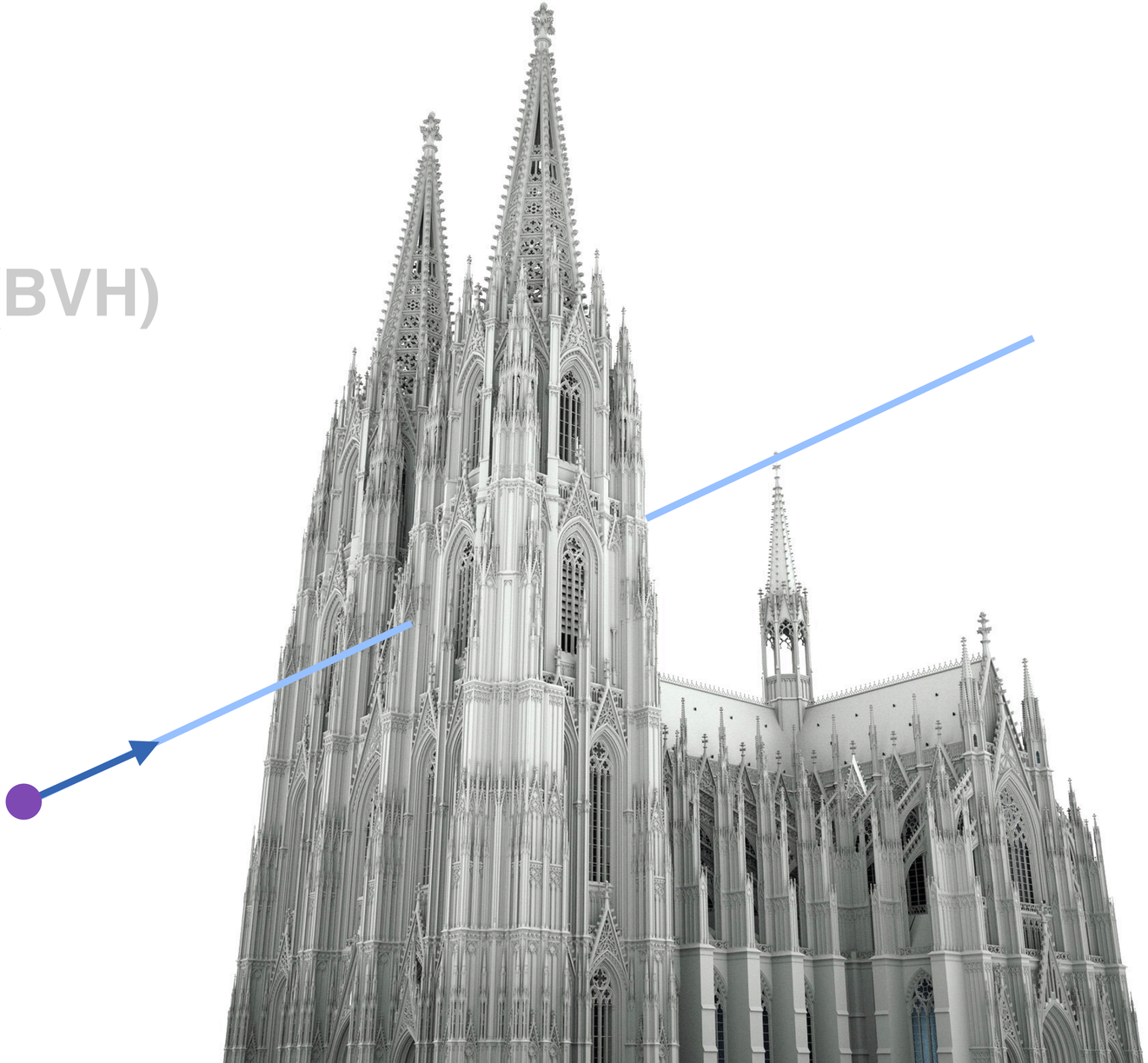
- k-D tree partition space non-uniformly
  - Repeatedly separate space by axis-aligned planes
  - E.g. when partitioning normal to y axis, score elements by y axis and use the medium value for the position of the partition plane
- Search intersection similar to BVH
- Unlike BVH, we can terminate early when a hit is found (easy to figure out the order of boxes along the ray)





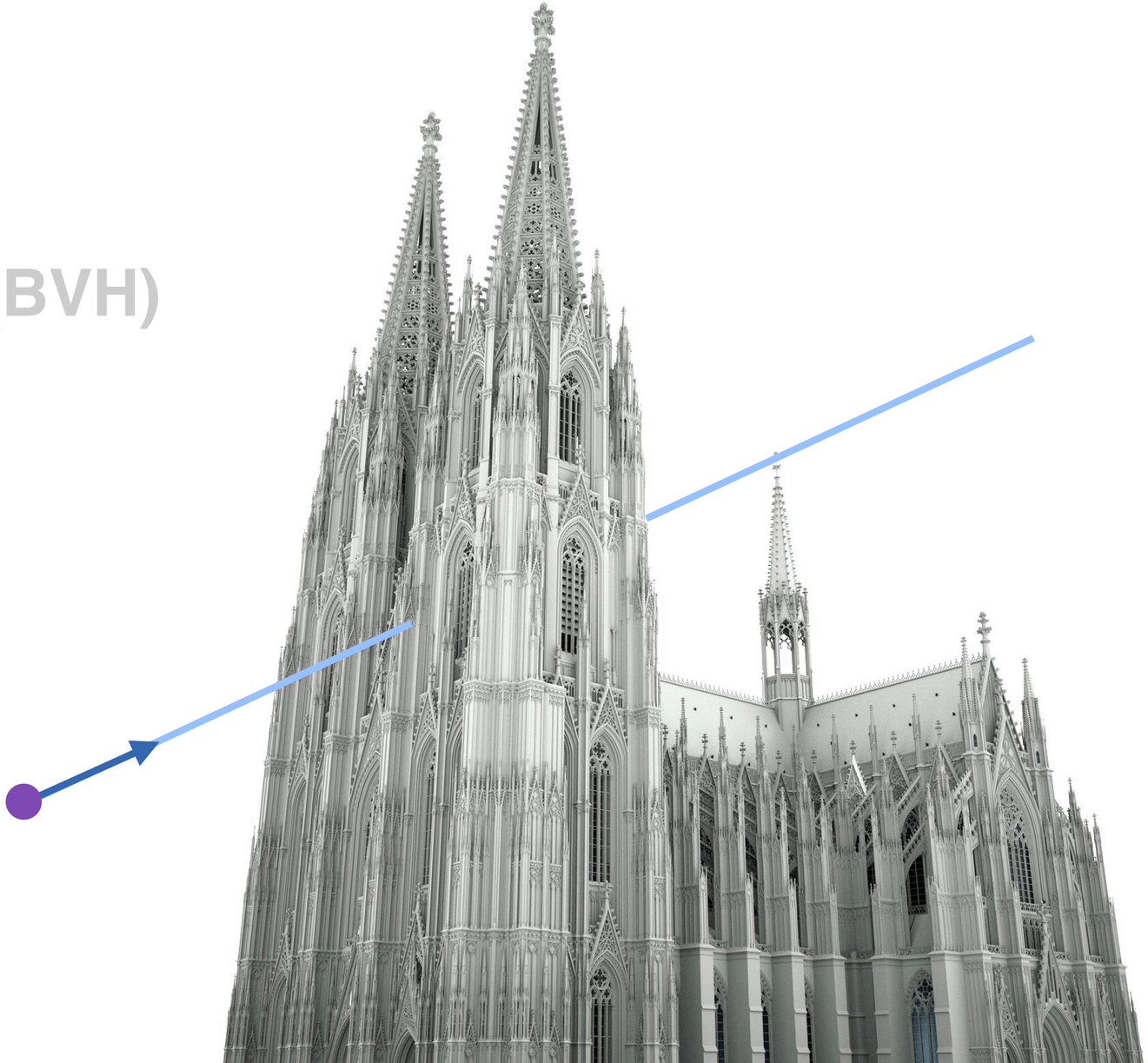
Fast geometry query

- Bounding volume hierarchy (BVH)
- **k-D tree**
- Octree



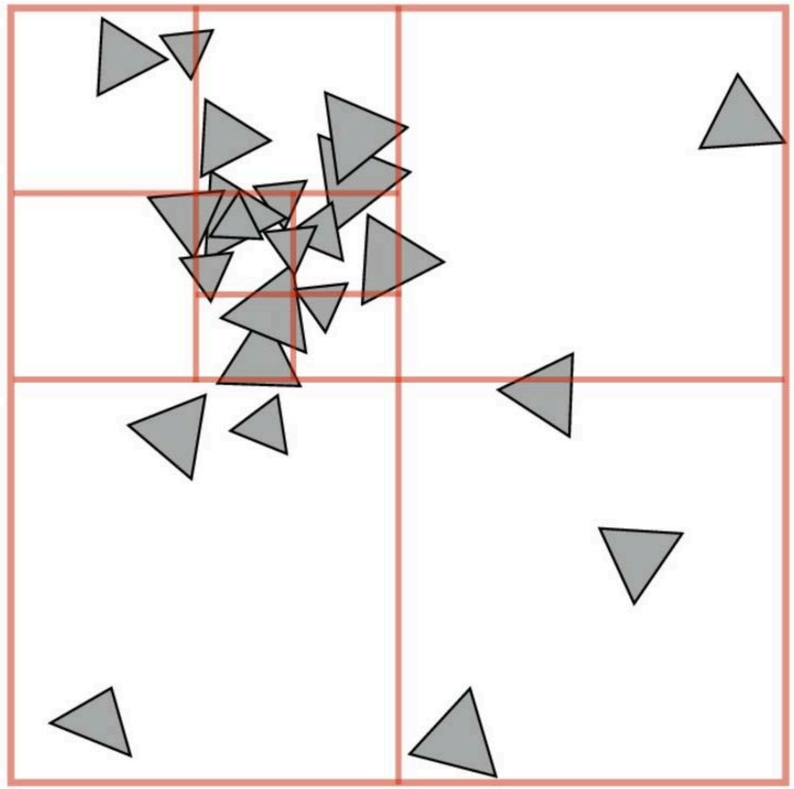
Fast geometry query

- Bounding volume hierarchy (BVH)
- k-D tree
- **Octree**



Quad-tree or Octree

- If needed, repeatedly bisect box into 4 (2D) or 8 (3D) sub-boxes
- Like k-D tree, it is adaptive to scene complexity
- Much easier to build than k-D tree (no need to choose separating planes)
- But lower ray traversal performance than k-D tree

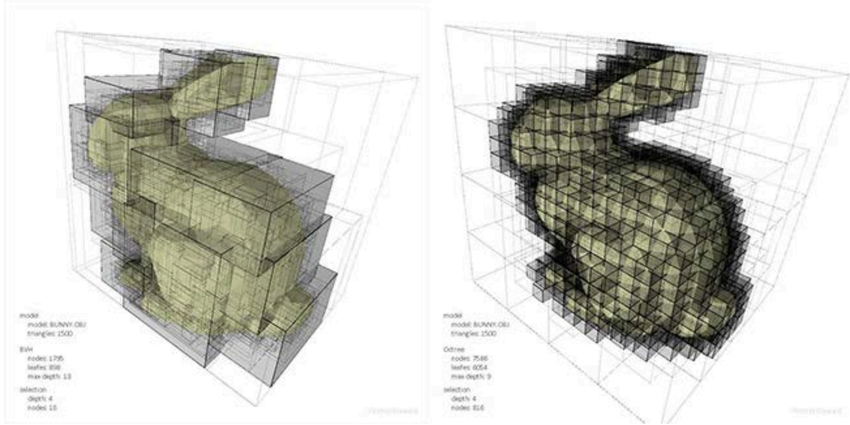


Quad-tree: nodes have 4 children (partitions 2D space)

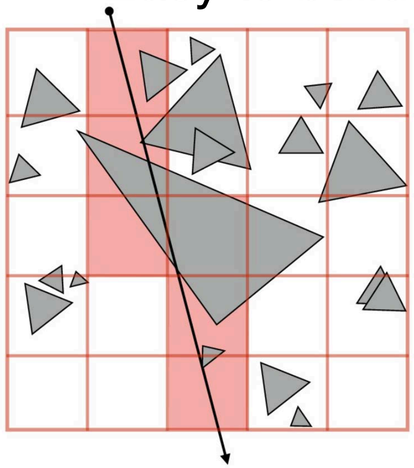
Octree: nodes have 8 children (partitions 3D space)

Summary

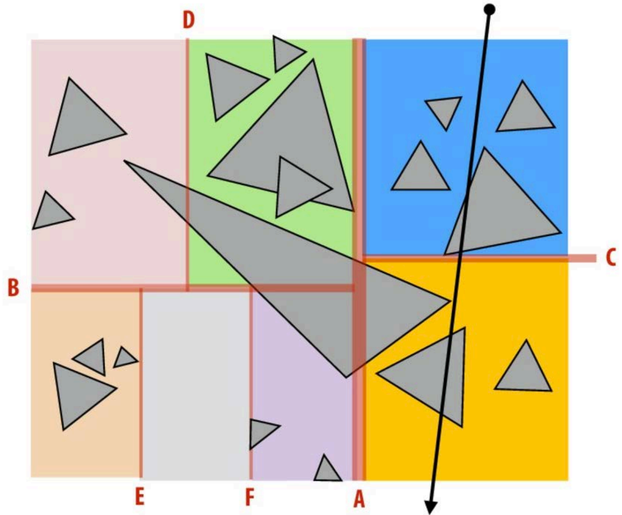
- BVH
  - Adaptive
  - No ordering



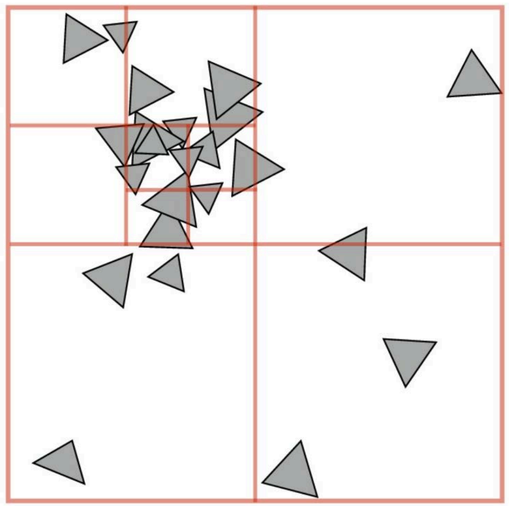
- Uniform grid
  - Non-adaptive
  - Ordered
  - Easy to build



- k-D tree
  - Adaptive
  - Ordered, fast marching
  - More effort to build



- Octree
  - Adaptive
  - Ordered, slower marching
  - Easy to build

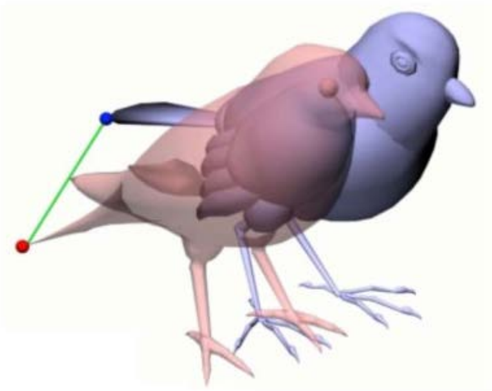


Quad-tree: nodes have 4 children (partitions 2D space)

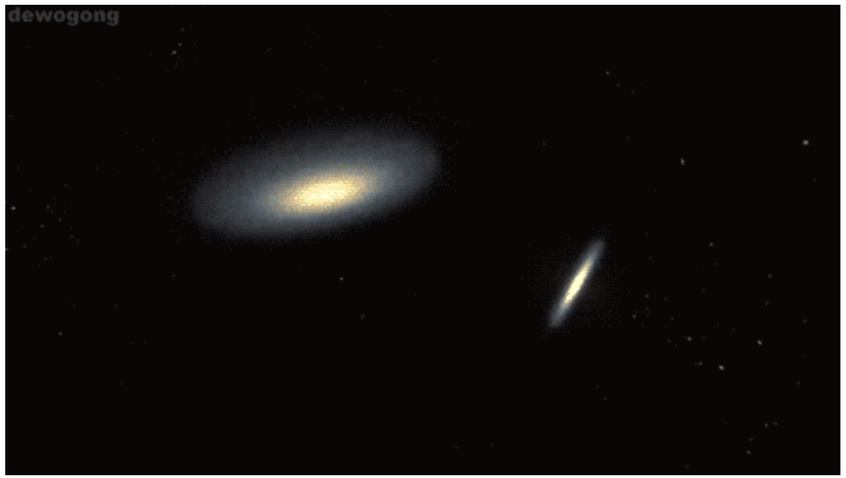
Octree: nodes have 8 children (partitions 3D space)

More hierarchical accelerations

- Geometry
  - Closest-point query
  - Distance query



- Particle simulation
  - Fast multipole method (Distant clusters can be simplified as one particle)
  - Fluid simulation, N-body problem



Accelerating Monte Carlo

- Accelerate geometry query
- Accelerate Monte Carlo

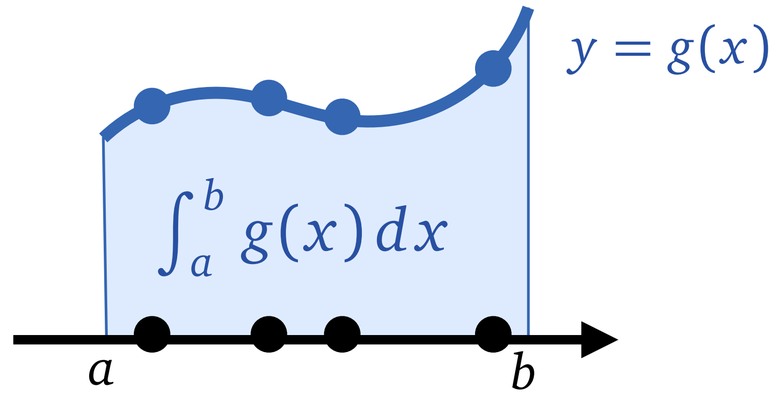
Accelerating Monte Carlo

$$\begin{cases} L_{\text{out}}(\mathbf{x}, \mathbf{v}_{\text{out}}) = L_{\text{e}}(\mathbf{x}, \mathbf{v}_{\text{out}}) + \iint_{H_{\mathbf{x}}} f(\mathbf{x}, \mathbf{v}_{\text{out}}, \mathbf{v}_{\text{in}}) L_{\text{in}}(\mathbf{x}, \mathbf{v}_{\text{in}}) \, d\sigma^{\perp}_{\mathbf{v}_{\text{in}}} \\ L_{\text{in}}(\mathbf{x}, \mathbf{v}_{\text{in}}) = L_{\text{out}}\big(\text{Cast}(\mathbf{x}, \mathbf{v}_{\text{in}})\big) \end{cases}$$

- Evaluating the integral amounts to shooting a ray at a random direction on the hemisphere.
- Can such Monte Carlo integration be accelerated?

Recall Monte Carlo integration

- **Task:** compute $I = \int_a^b g(x)\,dx$



- Sample *N* random points $x_1, x_2, \ldots, x_N$ independently and uniformly.
- For each sample, the integral is estimated as $I_i = (b-a)g(x_i)$
- The average would converge to the true solution

$$\frac{1}{N}\sum_{i=1}^{N} I_i \xrightarrow{N\to\infty} I \qquad \text{with error } \frac{\sqrt{\mathrm{Var}(I_i)}}{\sqrt{N}}$$

- Strategy: Lower the variance of $(b-a)g(x_i)$
- If g is constant, then this variance is zero; we have 1-step convergence!

Importance sampling

- **Task:** compute $I = \int_a^b g(x)\,dx$
- Suppose we don't sample $x_1, x_2, \ldots, x_N$ uniformly, but according to a distribution function $p(x)$
- That is, the probability of x_i lying in a set A is

$$\mathbb{P}\{x_i \in A\} = \int_A p(x)\,dx$$

- Then $\frac{1}{N}\sum_{i=1}^{N}(b-a)g(x_i) \xrightarrow{N\to\infty} \int_a^b g(x)p(x)\,dx$ wrong integral
- Fix it: $\frac{1}{N}\sum_{i=1}^{N}(b-a)\frac{g(x_i)}{p(x_i)} \xrightarrow{N\to\infty} \int_a^b \frac{g(x)}{p(x)}p(x)\,dx = \int_a^b g(x)\,dx$ correct integral

Importance sampling

- **Task:** compute $I = \int_a^b g(x)\,dx$
- Suppose we don't sample $x_1, x_2, \ldots, x_N$ uniformly, but according to a distribution function $p(x)$

$$\frac{1}{N}\sum_{i=1}^{N}(b-a)\frac{g(x_i)}{p(x_i)} \xrightarrow{N\to\infty} \int_a^b \frac{g(x)}{p(x)}p(x)\,dx = \int_a^b g(x)\,dx$$

- The convergence rate is $(b-a)\sqrt{\frac{\mathrm{Var}(g(x_i)/p(x_i))}{N}}$
- Choose $p(x)$ as close to $g(x)$ as possible, so that

$$\frac{g(x)}{p(x)} \approx \text{constant}$$

Application to ray sampling

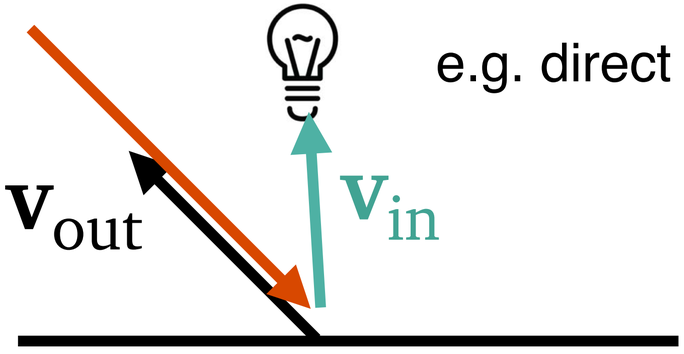
- When evaluating hemisphere integral

$$\begin{cases} L_{\text{out}}(\mathbf{x}, \mathbf{v}_{\text{out}}) = L_{\text{e}}(\mathbf{x}, \mathbf{v}_{\text{out}}) + \iint_{H_{\mathbf{x}}} f(\mathbf{x}, \mathbf{v}_{\text{out}}, \mathbf{v}_{\text{in}}) L_{\text{in}}(\mathbf{x}, \mathbf{v}_{\text{in}}) \, d\sigma^{\perp}_{\mathbf{v}_{\text{in}}} \\ L_{\text{in}}(\mathbf{x}, \mathbf{v}_{\text{in}}) = L_{\text{out}}\big(\text{Cast}(\mathbf{x}, \mathbf{v}_{\text{in}})\big) \end{cases}$$

- Want to sample rays $\mathbf{v}_{\text{in}}$ according to a distribution proportional to

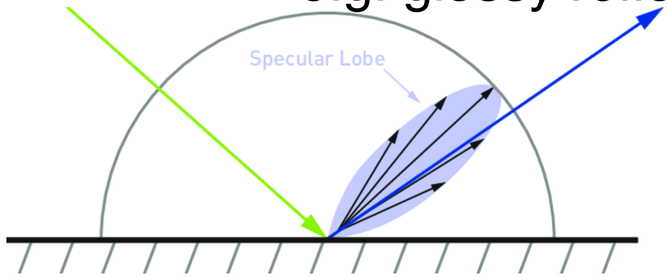
$$f(\mathbf{x}, \mathbf{v}_{\text{out}}, \cdot) L_{\text{in}}(\mathbf{x}, \cdot)$$

- Sample directions where light is large, and more according to BRDF



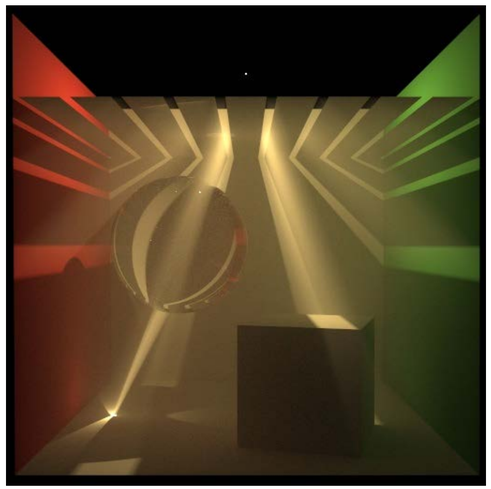
e.g. direct lighting

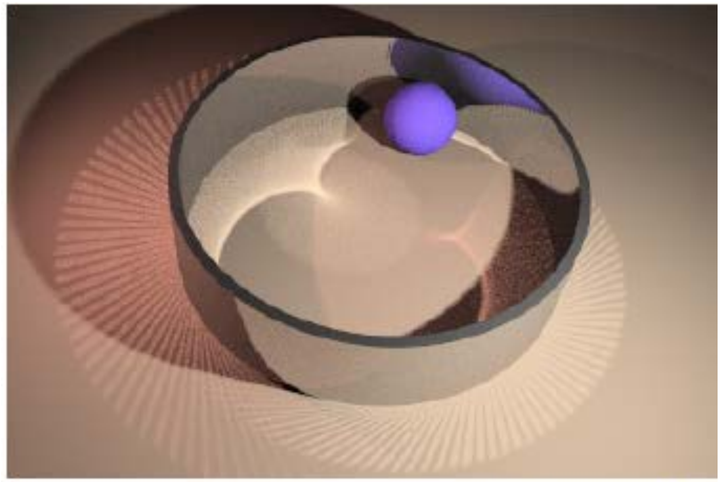
e.g. glossy reflection



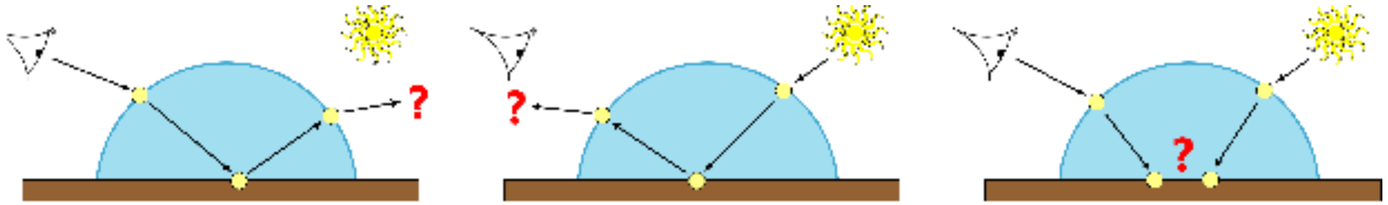
Which direction has more light?

- In more complicated scene, it is harder to know what distribution to sample ray to get more light.
- Challenge: caustics





- A technique is called **bidirectional path tracing**



(a) *Forward* **(b)** *Backward* **(c)** *Bidirectional*

Which direction has more light?

- Another general method in Monte Carlo sampling is the **Metropolis sampling algorithm**.
- Suppose we know how to evaluate a function $g(x)$ this algorithm would produce a random point according to the distribution proportional to g.
- Initialize the point randomly **x**, get a nearby candidate **x'**, move **x** to candidate or stay still according to probability g(**x'**)/g(**x**)
- Idea: Once a rare but important event is found, focus exploration there.
- Once a (rare) path from eye to light is found, explore nearby paths

Increase the rate O(1/sqrt(N))

- Monte Carlo typically have error $(\frac{1}{N}\sum_{i=1}^{N} I_i) - I \approx \frac{\sqrt{\mathrm{Var}(I_i)}}{\sqrt{N}}$
- The order $O(1/\sqrt{N})$ is seemingly unbreakable
- If the samples are independent, error must be $O(1/\sqrt{N})$
- Use correlated sample!
- **Stratified sampling:** If the function we are integrating is smooth in pieces, then stratified sample in each piece increase order to O(1/N) or higher.

