Eurographics 2014 Tutorial

Efficient Sorting and Searching in Rendering Algorithms

Organizers and Presenters

Vlastimil Havran and Jiří Bittner

Czech Technical University in Prague

In the tutorial we show the connection between rendering algorithms and sorting and searching as classical problems studied in computer science. We provide both theoretical and empirical evidence that for many rendering techniques most time is spent by sorting and searching. In particular we discuss problems and solutions for visibility computation, density estimation, and importance sampling. For each problem we mention its specific issues such as dimensionality of the search domain or online versus offline searching. We will present the underlying data structures and their enhancements in the context of specific rendering algorithms such as ray tracing, photon mapping, and hidden surface removal.

EUROGRAPHICS 2014 TUTORIAL

EFFICIENT SORTING AND SEARCHING IN RENDERING ALGORITHMS

VLASTIMIL HAVRAN & JIŘÍ BITTNER

Czech Technical University in Prague

DCGI

TUTORIAL OVERVIEW

Introduction (5 min)	VH
Sorting and Searching Techniques (30 min)	JB
Hierarchical Data Structures (30 min)	JB/VH
Ray Tracing (20 min)	VH
Q & A (5 min)	
Coffee break	
Rasterization and Culling (25 min)	JB
Photon Maps and Ray Maps (20 min)	VH
Irradiance Caching (5 min)	VH
BRDF and BTF (10 min)	VH
Sorting and searching on GPU (20 min)	JB
Q & A (10 min)	

Introduction

- Recall that we mostly use sorting and searching in rendering
- Highlight connections between different problems in rendering
- Show standard efficient approaches
- Show non-standard approaches
- New trends, GPUs, mobile devices

Issues not Covered in our Tutorial

- Collision detection algorithms
- Volumetric rendering
- Image based rendering
- Non-photo realistic rendering
- General clustering techniques
- Graph theory and other related problems

- Updated tutorial slides available at

  http://dcgi.felk.cvut.cz/~havran/eg2014tut/

Tutorial Organization and Level

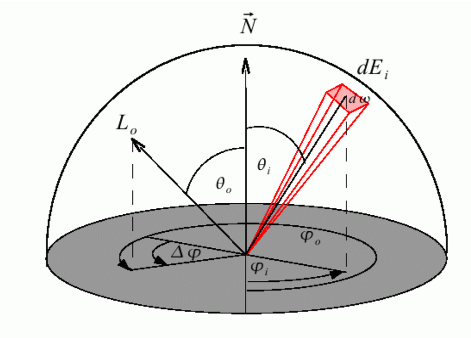
- Intermediate level – basic knowledge is required
- Any questions can be asked during the presentation
- The details are not given due to the lack of time
- Detailed bibliography provided in the supplementary material

Introduction to Rendering

- Rendering Equation

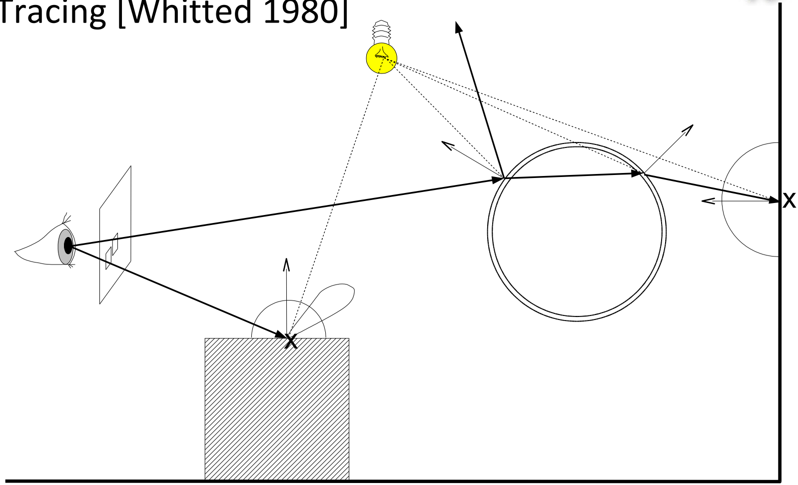
$$L(\vec{\omega_o}, x) = L^e(\vec{\omega_o}, x) + \int_\Omega L_i(x, \vec{\omega_i}) \cdot f_r(x, \vec{\omega_i}, \vec{\omega_o}) \cdot (\vec{n} \cdot \vec{\omega_i}) d\omega_i,$$

- Convolving incoming light with surface reflectance properties



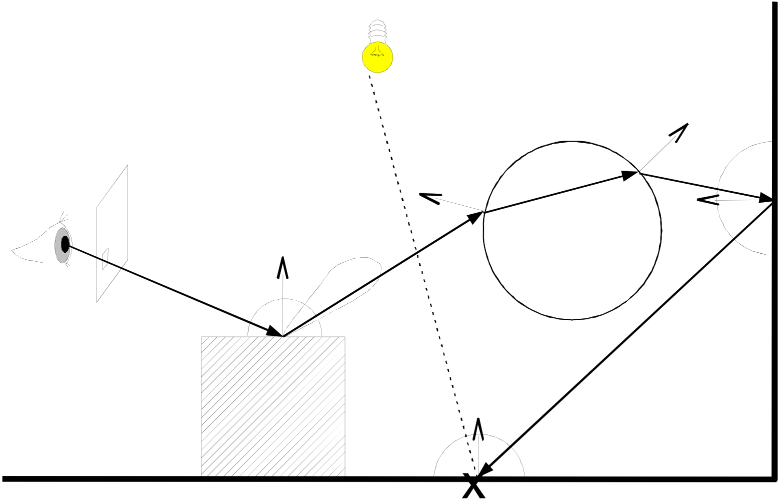
Ray Tracing

- Ray Tracing [Whitted 1980]



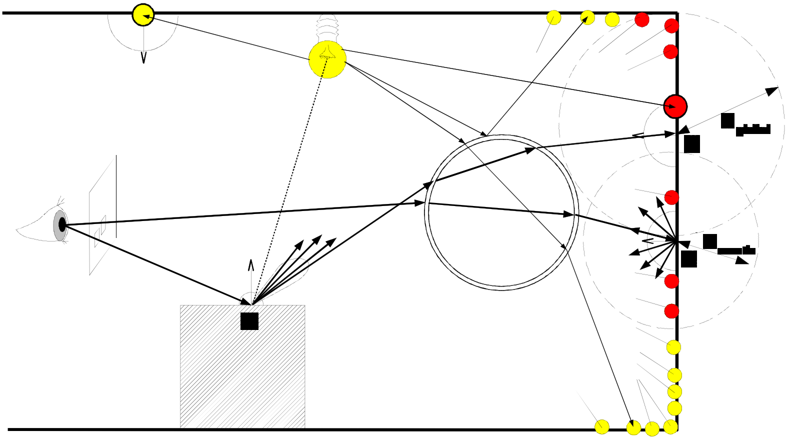
Path Tracing

- Rendering Equation [Kayija 1986]

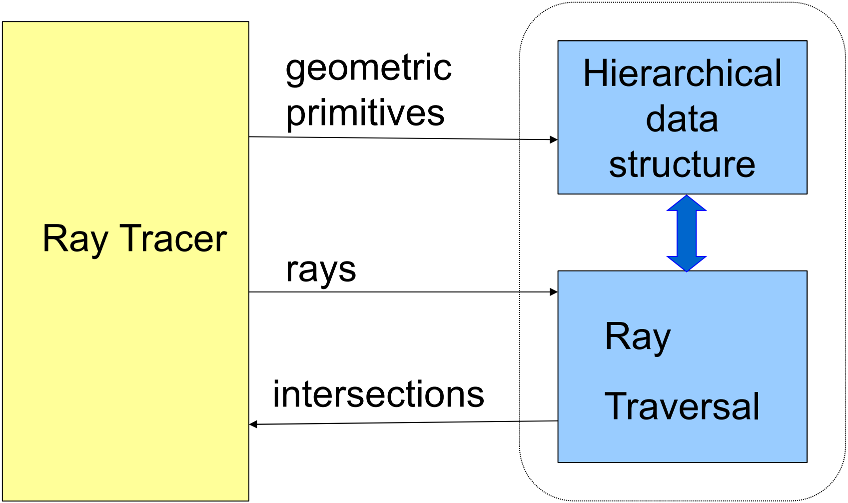


Photon Mapping Algorithm

- Photon mapping [Jensen 1993]
- Vertex-connection-merging [Georgiev 2012,Bekaert2003]

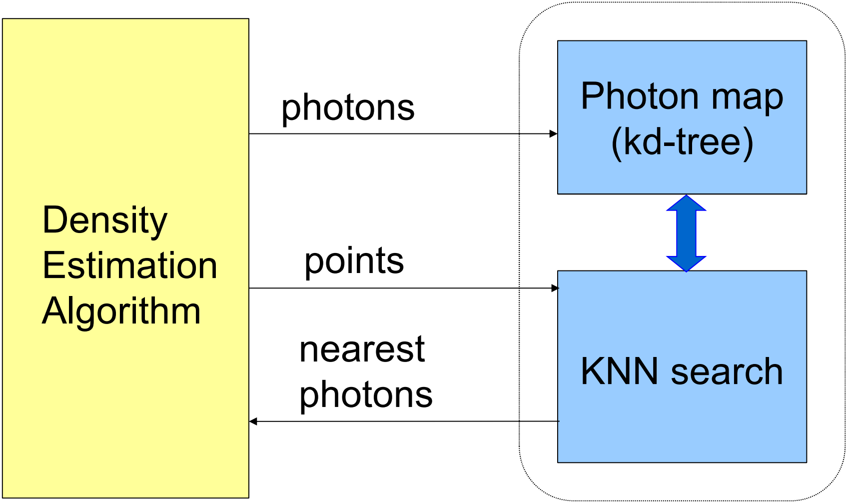


Ray Tracing

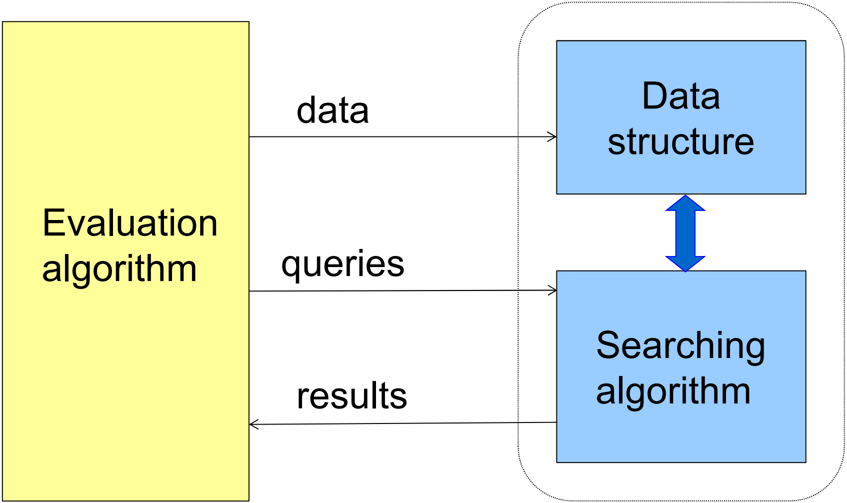


Ray Tracer
geometric primitives
rays
intersections
Hierarchical data structure
Ray Traversal

Photon Density Estimation



Sorting and Searching in Rendering



Take Home Message

- 90 percent of the time in most rendering algorithms is taken by sorting and searching
- The rest of computation is pure evaluation of math formulas, random number generation etc.
- It is therefore necessary to understand well the particular instances of sorting and searching for rendering
- Tremendous research and development effort was spent since 1980 in the algorithms for both software and hardware

Thank you!

EUROGRAPHICS 2014 TUTORIAL

INTRODUCTION TO SORTING AND SEARCHING

JIŘÍ BITTNER

Czech Technical University in Prague

TUTORIAL OVERVIEW

Introduction (5 min)	VH
Sorting and Searching Techniques (30 min)	**JB**
Hierarchical Data Structures (30 min)	JB/VH
Ray Tracing (20 min)	VH
Q & A (5 min)	
Coffee break	
Rasterization and Culling (25 min)	JB
Photon Maps and Ray Maps (20 min)	VH
Irradiance Caching (5 min)	VH
BRDF and BTF (10 min)	VH
Sorting and searching on GPU (20 min)	JB
Q & A (10 min)	

Search Problem

Search Space

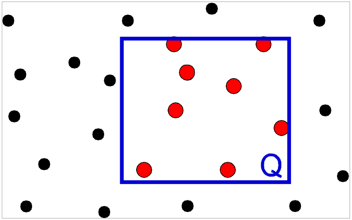
$$Q \times S \rightarrow A$$

Query Domain

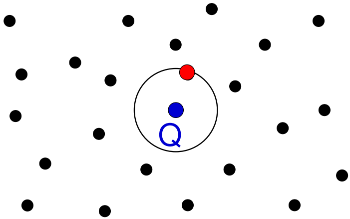
Domain of Answers

Geometric Search Problems

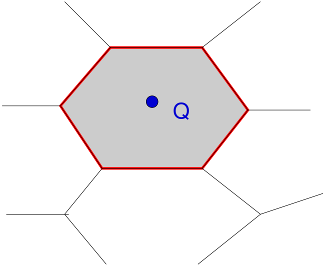
Range search



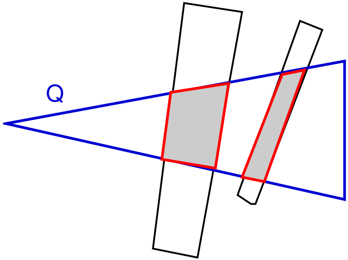
Nearest Neighbors



Point location



Intersection detection



Search Problems in Rendering

Problem	Q	S	A
Ray shooting	ray	{objects}	point
Hidden Surface Removal	{rays}	{objects}	{points}
Visibility culling	{rays}	{objects}	{objects}
Photon maps	point	{points}	{points}
Ray maps	point	{rays}	{rays}
Irradiance caching	point	{spheres}	{spheres}

Searching Algorithms

- Exact vs. Approximate
  - Approximate: finds solution close to exact one
  - E.g. ε-nearest neighbor
- Online vs. offline
  - Offline: applied for entire sequence of queries
  - E.g. single ray query vs "all" rays queries
- Static vs. dynamic
  - Dynamic: input may change

Sorting

- Organizing data
- Improve searching performance
- Naïve search: O(n) time
- With sorting: O(log n)
- In special cases even O(1)

Basic Sorting Algorithms

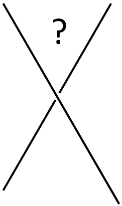
Algorithm	Method	Best	Average	Worst
Heapsort	Selection	O(n log n)	O(n log n)	O(n log n)
Selection sort	Selection	$O(n^2)$	$O(n^2)$	$O(n^2)$
Quicksort	Partitioning	O(n log n)	O(n log n)	$O(n^2)$
Bucket sort	Distribution	O(n)	O(n)	$O(n^2)$
Merge sort	Merging	O(n log n)	O(n log n)	O(n log n)
Bubble sort	Exchanging	O(n)	$O(n^2)$	$O(n^2)$
Insertion sort	Insertion	O(n)	$O(n^2)$	$O(n^2)$

Space complexity: O(n)

Multidimensional Sorting

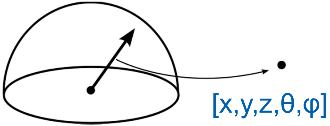
- We deal with multidimensional data!
  - Objects, points, rays, normals, …
- Define relations among elements of S

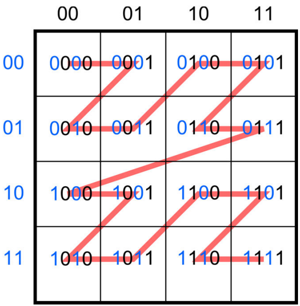
Numbers (1D)	Points (3D)	Lines (4D)
5 < 9	[5,3,2] ? [9,6,7]	
10 > 2	[10,1,1] ? [2,8,6]	



Problem Dimensionality

- Spatial sorting: 3D domain
  - Surfaces: 2D, height fields: 2.5D
- Spatio-temporal sorting: 4D domain
- Ray space sorting: 5D domain
  - 4D for lines
- Space filling curves: nD → 1D
  - Morton codes
- Feature vectors: nD





Comparison-based Sorting

- Evaluating A < B
  - Quicksort
  - Selection sort
  - Heap sort
  - Merge sort
  - Shell sort
  - Insertion sort
  - …
- Ω(n log n)

Quicksort example

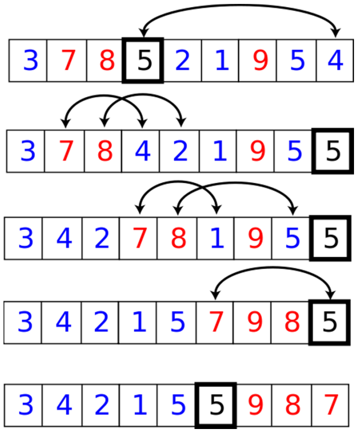
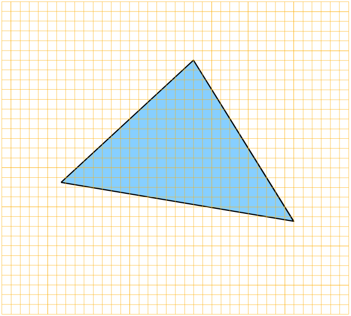


figure courtesy of D. Coetzee

Bucketing

- Distributing input data into buckets / bins
- Buckets
  - Regular grids
  - 1D bins
  - 2D image (A-buffer)
  - 3D voxel grid
- Not a comparison-based sort
- O(n)
  - Assuming discretized data
- Radix sort a special case



Bucketing Example

Data range 0 – 9

Input: 7 8 3 2 3 6 9 5 8

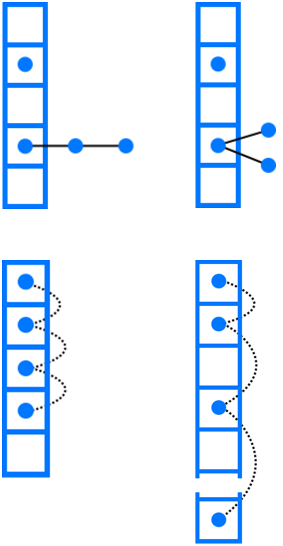
Buckets:

		2	3		5	6	7	8	9
			3					8	

Output: 2 3 3 5 6 7 8 8 9

Hashing

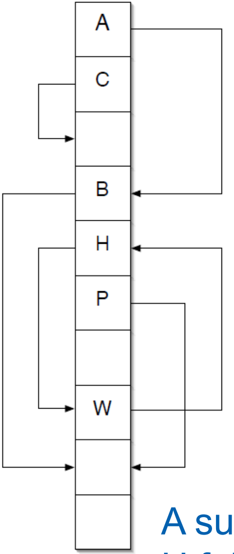
- Sparse data in higher dimensions
- Hashing function
- Resolving collisions
- Chaining
  - Linked list, balanced tree
- Open addressing
  - Linear/quadratic probing
  - Double hashing
- Perfect hashing
  - No collisions
  - Memory to store hash function
  - Useful for static data



Cuckoo Hashing

- Cuckoo hashing
  - Two hash functions
  - Inserted entry pushes away the old entry
  - Longer build times
  - Fast retrieval

```
procedure insert(x)
   if T[h1(x)] = x or T[h2(x)] = x then return;
   pos ← h1(x);
   loop n times {
      if T[pos] = NULL then { T[pos] ← x; return};
      x ↔ T[pos];
      if pos= h1(x) then pos← h2(x) else pos← h1(x);}
   rehash(); insert(x)
end
```



A success
H fail

pseudocode and figure courtesy of R. Pagh

Sorting in Rendering

- Sort by partitioning (Quicksort like)
  - Top-down construction of spatial hierarchies
- Sort by selection (Heapsort like)
  - Bottom-up construction of spatial hierarchies
  - k-NN search
- Sort by insertion (Insertion sort like)
  - Incremental construction of hierarchies
- Sort by distribution (Bucket sort like)
  - Rasterization (z-buffer, A-buffer)
  - GPU sorting (radix sort)
- Sort by exchanging (Bubble sort like)
  - Incremental priority orders

EUROGRAPHICS 2014 TUTORIAL

HIERARCHICAL DATA STRUCTURES

JIŘÍ BITTNER, VLASTIMIL HAVRAN

Czech Technical University in Prague

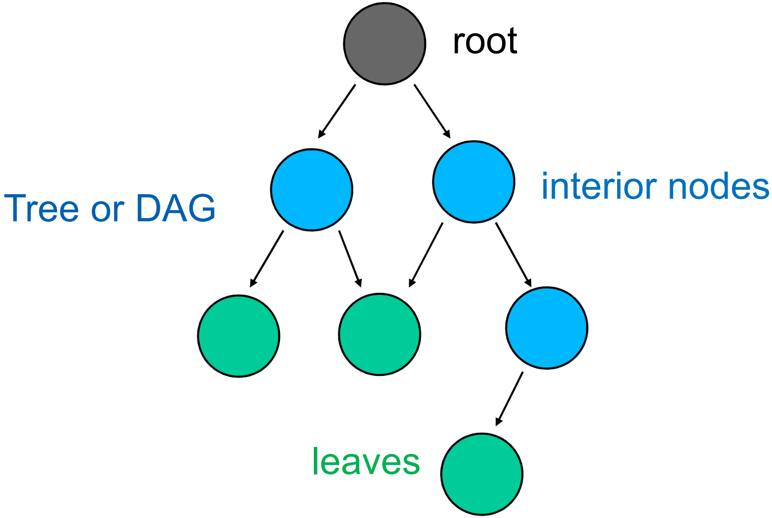
TUTORIAL OVERVIEW

Introduction (5 min)	VH
Sorting and Searching Techniques (30 min)	JB
Hierarchical Data Structures (30 min)	JB/VH
Ray Tracing (20 min)	VH
Q & A (5 min)	
Coffee break	
Rasterization and Culling (25 min)	JB
Photon Maps and Ray Maps (20 min)	VH
Irradiance Caching (5 min)	VH
BRDF and BTF (10 min)	VH
Sorting and searching on GPU (20 min)	JB
Q & A (10 min)	

Hierarchical Data Structures (HDS)

- Connection to sorting
- Classification
- Bounding volume hierarchies
- Spatial subdivisions
- Hybrid data structures
- Searching using HDS
- Special techniques on hierarchies

Hierarchical Data Structure



Connection to Sorting

- Hierarchical Data Structures =
  implementation of (spatial) sorting

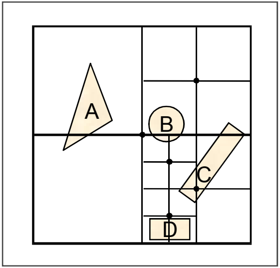
- Why ?
- Top-down construction of HDS equivalent to quicksort
- Time complexity O(N log N)

Recall Quicksort

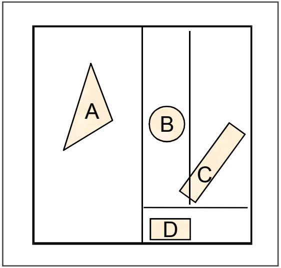
- Pick up a pivot Q
- Organize the data into two subarrays
  - Smaller than Q
  - Larger or equal Q
- Recurse in both subarrays
- In 3D Pivot = Plane
  - Smaller / larger ~ back / front

Examples of HDS in 2D

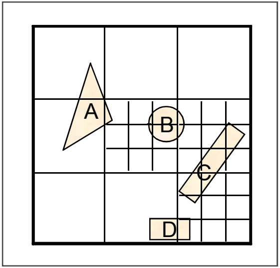
octree



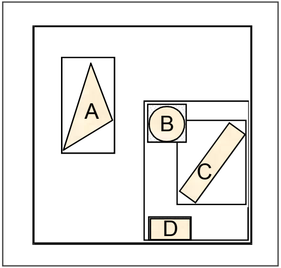
kd-tree



hierarchy of grids



bounding volume hierarchy



HDS Classification

- Data domain organization
- Dimensionality
- Data layout

HDS - Data Domain Organization

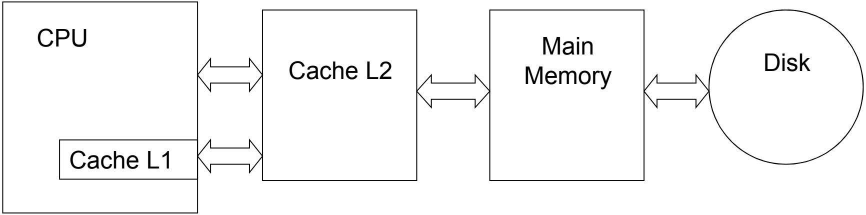
- Spatial subdivisions
  - Organizing space (non-overlapping regions)
- Object hierarchies
  - Organizing objects (possibly overlapping regions)
- Hybrid data structures
  - Spatial subdivision mixed with object hierarchies
- Transformations and mappings

HDS - Dimensionality

- Necessary to represent data entities
  - 1D, 2D, 3D, 4D, or 5D
- Data entities
  - Points, lines, oriented half-lines, disks, oriented hemispheres, etc.
- Possibility to extend many problems to time domain
  - Plus one dimension

HDS – Data Layout

- Internal data structures
- External data structures (out of core)
- Cache-aware data structures
- Cache oblivious data structures

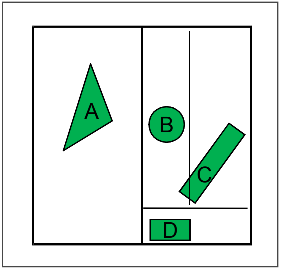


Types of Nodes in HDS

- Interior node
  - Eepresents a “pivot”, data entities sorted according to pivot
  - E.g. subdivision plane or a set of planes plus references to child nodes
  - Efficient representation crucial for searching performance
- Leaf node
  - Contains data
  - E.g. list of objects, points
  - Entities themselves or references
- Implementation concerns
  - Discriminating interior/leaf node
  - Implicit pointers to child node(s)

Spatial Subdivisions

- Non-overlapping regions of child nodes
- Space is organized by subdivision entities (planes)
- Constructed top-down
- Fully covering original spatial region
  - Point location always possible: empty or non-empty leaf



kd-tree

Spatial Subdivision Examples

- Kd-trees – axis aligned planes
- BSP-trees – arbitrary planes
- Octrees – three axis aligned planes in a node
- Uniform grids (uniform subdivision)
- Recursive grids

Object Hierarchies

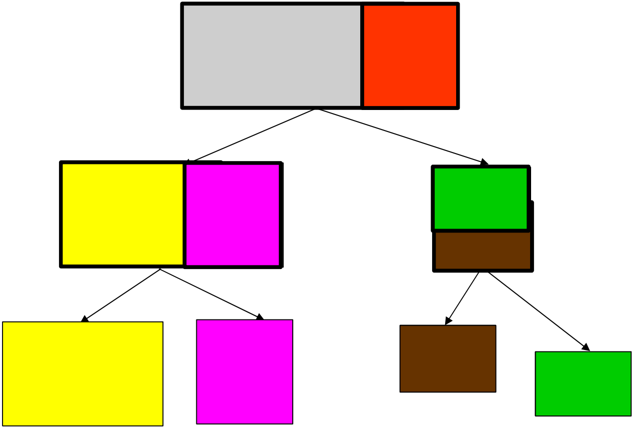
- Possibly overlapping regions of child nodes
- Possibly some spatial regions are not covered
  - Point location impossible
- Construction methods
  - Top-down (sorting)
  - Bottom-up (clustering)
  - Incrementally (by insertion)
- Bounding Volume Hierarchies (BVH)
  - Bounding volume = AABB, sphere, OBB, …

Examples of Object Hierarchies

- Bounding Volume Hierarchies (BVHs)
- R-trees and their many variants
- Box-trees
- Several others
  - Special sort of bounding volumes... sphere trees etc.

Bounding Volume Hierarchies

Constructed Top-Down



Hybrid Data Structures

- Combining various interior nodes
- Combining spatial subdivisions and object hierarchies
- Sharing pros and cons of both types
- Can be tuned to compromise of some properties
  - E.g. efficiency and memory

Other HDS

- Content of the node
  - Single splitting plane, more splitting planes, box, additional information
- Arity of a node
  - Also called branching factor, fanout factor
- Way of constructing a tree
  - Height, weight balancing, postprocessing
- Data only in leaves or also in interior nodes
- Augmenting data

Example of Other HDS

- Cell trees (polyhedral shapes for splitting)
- SKD-trees (two splitting planes at once)
- hB-trees (holey brick B-trees)
- LSD-tree (height balanced kd-tree)
- P-trees (polytope trees)
- BBD-trees (bounding box decomposition trees)
- And many others

(see surveys listed in tutorial notes, in particular encyclopedia [Samet06])

Transformation Approach

- Transform the problem domain
- Transformation examples
  - Box in 3D -> point in 6D
  - Sphere in 3D -> point in 4D
- Transformation can completely change searching algorithm

HDS Construction Algorithm (Top-Down)

Initial phase

Create a node with all elements

Put the node in the auxiliary structure AS (stack or priority queue)

Divide & Conquer phase

```
While AS not empty do {
    Get node N from AS
    If (should be subdivided(N)) {
        decide splitting
        create new nodes and put them to AS
    }  else
      create leaf
}
```

Search Algorithms using HDS

- Down traversal phase (location) + some other phase
- Start from the root node
- Visiting an interior node
  - Use stack (LIFO) or priority queue to record nodes to be visited
- Visiting a leaf
  - Compute incidence (such as ray-object intersection)
- Note: auxiliary structure implements another sorting phase during searching

Search Algorithms using HDS

- Range queries
  - Given range X, find all incidences of X with data
- Nearest neighbour
  - Find the nearest neighbour
  - K-nearest neighbours
- Intersection search
  - Given point Q, find all objects that contain Q
- Ranking
  - Given query object Q, report on all objects in order of distance from Q
- Reverse nearest neighbours
  - Given point Q, find all points to which Q is the nearest neighbour

Search Performance Model

- Result = the cost of computation ... C
- Performance is inverse proportional to the quality of the data structures for given problem
- Two uses of performance model
  - **A posteriori**: documenting and testing performance
  - **A priori**: constructing data structures with higher expected performance

Search Performance Model

$$C = C_T + C_L + C_R$$

$$C = C_{TS} \cdot N_{TS} + C_{LO} \cdot N_{LO} + C_{ACCESS} \cdot N_{ACCESS}$$

- C_T … cost of traversing the nodes of HDS
- C_L … cost of incidence operation in leaves
- C_R … cost of accessing the data from internal or external memory

Performance Model

- C_T … cost of traversing the nodes of HDS
  - N_{TS} … number of traversal steps per query
  - C_{TS} … average cost of a single traversal step
- C_L … cost of incidence operation in leaves
  - N_{LO} … number of incidence operation per query
  - C_{LO} … average cost of incidence operation
- C_R … cost of accessing the data from internal or external memory
  - N_{ACCESS} … number of read operations from internal/external memory per query
  - C_{ACCESS} … average cost of read operation

HDS for Dynamic Data

- Two major options:
  - Rebuild HDS after the data changes from scratch
  - Update only necessary part of HDS
- Design considerations:
  - How much data are changed (M from N entities)
  - How efficient would be the updated data structures now and in the longer run?
  - How much time is required in both methods?

Rebuild from Scratch

- Construction time is typically O(N log N)
- The constants behind big-O notation are important in practice !
- Suitable if most objects are moving (M » N)
- Quality of hierarchy is high!
- Hint (top-down HDS):
  - Save exchange operations by keeping order given by previous hierarchy

HDS Updates

- Only update data structures to reflect the changes
- Assumed searching performance remains acceptable
  - No guarantees
- Additional bookkeeping data to monitor HDS cost
- Techniques for 1D trees (rotation, balancing) often not applicable
- Updating larger amount of data at once (bulk updating)

HDS Updates

- Insertion method – delete and reinsert the data in the tree (also deferred insertion)
  - Suitable if the number of changed objects is small
  - Each insertion/deletion requires O(log N)
  - Necessary delete and update some interior nodes
- Postorder processing (only for object hierarchies)
  - Suitable if number of changed objects is high
  - First update all leaves (data itself)
  - Traverse the whole tree in O(N) and reconstruct interior nodes of object hierarchy knowing both children

EUROGRAPHICS 2014 TUTORIAL

RAY TRACING

VLASTIMIL HAVRAN

Czech Technical University in Prague

TUTORIAL OVERVIEW

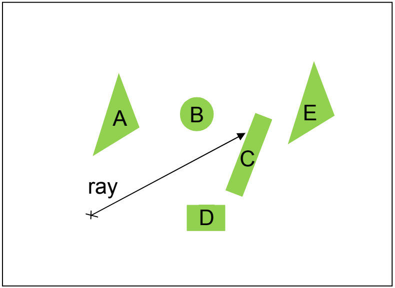
Introduction (5 min)	VH
Sorting and Searching Techniques (30 min)	JB
Hierarchical Data Structures (30 min)	JB/VH
Ray Tracing (20 min)	VH
Q & A (5 min)	
Coffee break	
Rasterization and Culling (25 min)	JB
Photon Maps and Ray Maps (20 min)	VH
Irradiance Caching (5 min)	VH
BRDF and BTF (10 min)	VH
Sorting and searching on GPU (20 min)	JB
Q & A (10 min)	

Ray Tracing

- Ray shooting versus ray tracing
- Connection to sorting and searching
- Performance model/studies
- Uniform Grids
- Hierarchical Data Structures
- Special techniques on hierarchies

Ray Tracing

- Task: given a ray, find out the first object intersected, if any
- Input: a scene and a ray
- Output: the intersected object



Ray Tracing *versus* Ray Shooting

- Ray shooting
  - Only a single ray
- Ray tracing in computer graphics
  - Ray shooting (only a single ray)
  - Ray casting – only primary rays from camera
  - Recursive ray tracing
  - Distribution ray tracing and others

Time and Space Complexity

- Computational Geometry
  - Aiming at worst-case complexity
  - Restriction to certain class of object shape (triangles, spheres)
  - Unacceptable memory requirements O(log N) query time induces $\Omega(N^4)$ space [Szirmay-Kalos and Marton, 1997/8]
- Computer Graphics
  - Aiming at average-case time complexity of search O(log N), space complexity O(N), time complexity of build O(N.log N)
  - Practicality and robustness
  - Ease of implementations
  - Acceptable performance on particular computer hardware

    (CPU versus GPU)

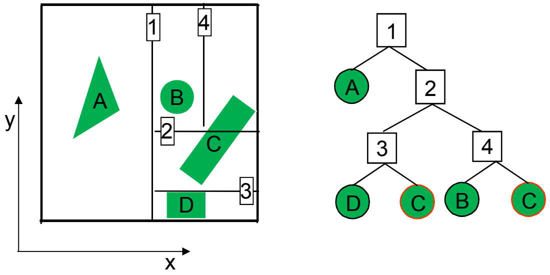
Computer Graphics Ray Tracing Algorithms

- Techniques developed: aimed at practical applications, no complexity guarantees, many “tricks”, the analysis difficult or infeasible
- Basic techniques
  - Bounding volumes, spatial subdivision, ray classification
- Augmented techniques
  - Macro regions, pyramid clipping, proximity clouds, directed safe zones
- Special tricks
  - Ray boxing, mailbox, handling CSG primitives, other types of coherence

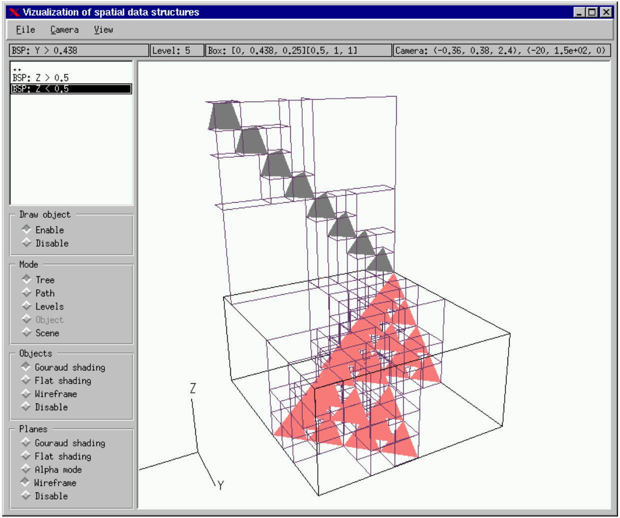
Ray Tracing Data Structures Build Algorithm Classification

- Subdivision techniques (top down)
  - Binary space partitioning (e.g. kd-trees, octrees)
  - Uniform and hierarchical grids
  - Bounding volume hierarchies
- Clustering (bottom up)
  - Bounding volume hierarchies
- Insertion based algorithm
  - Bounding volume hierarchies
- Hybrid algorithms – part of tree can be created differently

Example: Kd-tree Construction

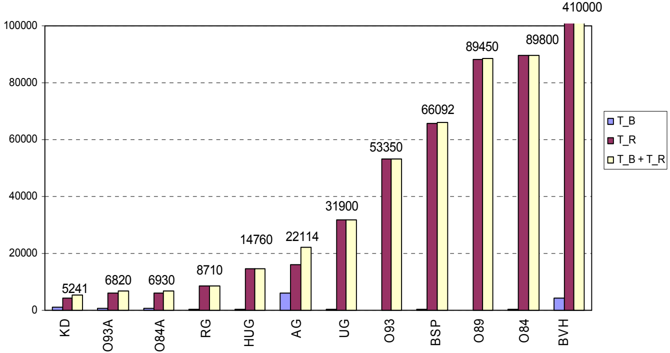


Kd-tree Visualization



Data Structures Comparison, year 2000

- 30 scenes **x** 12 data structures **x** 4 ray distributions
- 1440 measurements
- Timings (build time, search time, total time) for ray tracing



Note: BVH tested constructed by insertion [Goldsmith+Salmon 87].

Ray Tracing Data Structures, year 2014

- Three prevailing data structures:
  - BVHs
  - Kd-trees
  - Hybrid: Spatial BVHs – BVHs and KD-trees
- The implementation often only for triangular scenes
- The other data structures interesting but not widely accepted in practice
- BVH with cost model based on SAH favored for simplicity and fixed memory footprint
- Kd-trees or Spatial BVHs favored for performance guarantees but more complex to build, dynamic allocation needed

Bounding Volume Hierarchies (BVHs)

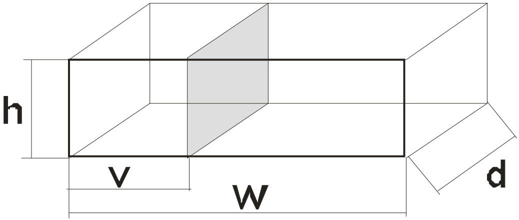
- Known since 1980, automatic build since 1984, efficient and robust implementations however since 2006.
- Easy construction algorithm - subdivision of objects into the groups in top-down fashion using cost model
- Two other building algorithms possible
  - Insertion based algorithm starting from a single leaf
  - Merging like algorithms (agglomerative clustering)
- Light-weight versions of BVH possible
- Build algorithm in O(N log N)
- Optimization algorithms of an existing BVH available

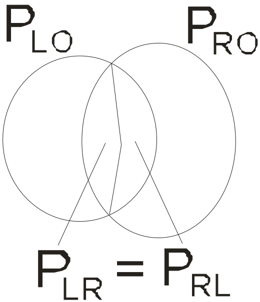
Kd-trees

- The easy spatial subdivision with non-overlapping spatial regions representing leaves
- Empty leaves are needed with zero storage
- One geometric primitive can be referenced in more leaves – unknown number of references
- The performance usually higher than for BVHs
- Well known and tested, robust build and traversal algorithms
- Build algorithm also in O(N log N)
- No optimization algorithms possible

Geometric Probability of Ray Box Intersection (Surface Area Heuristic)

$$P_{LEFT} = P_{LO} + P_{LR} + P_{RL}$$
$$P_{RIGHT} = P_{RO} + P_{LR} + P_{RL}$$

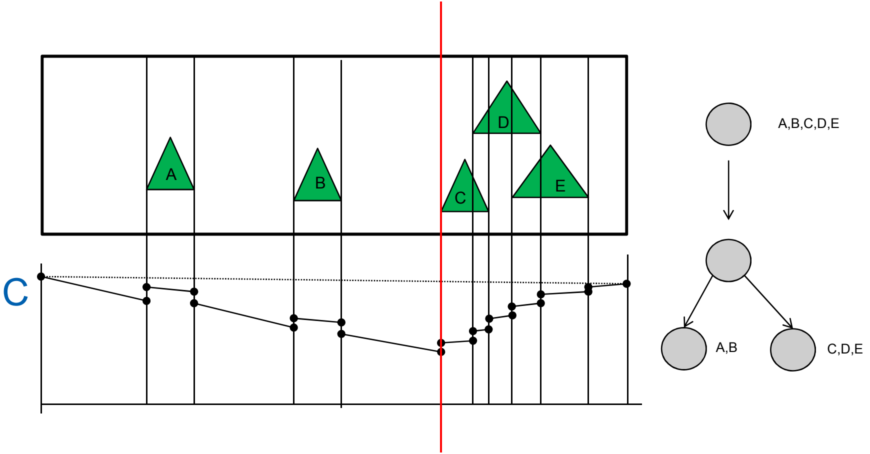




- Probability computed from surface area of the box
- Condition: uniform ray distribution

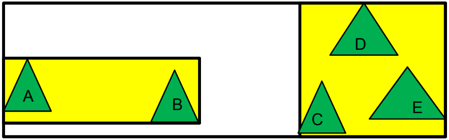
Kd-tree Building with Greedy Cost Model

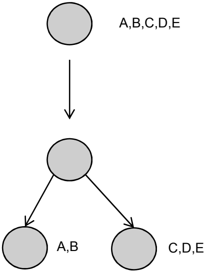
- Cost function $C = P_{LEFT} \cdot N_{LEFT} + P_{RIGHT} \cdot N_{RIGHT}$
- The cost minimization in top-down build for each node



BVH Building with Greedy Cost Model

- Cost function $C = P_{LEFT} \cdot N_{LEFT} + P_{RIGHT} \cdot N_{RIGHT}$
- The cost minimization in top-down build for each node
- Bounding box is tight over all triangles!



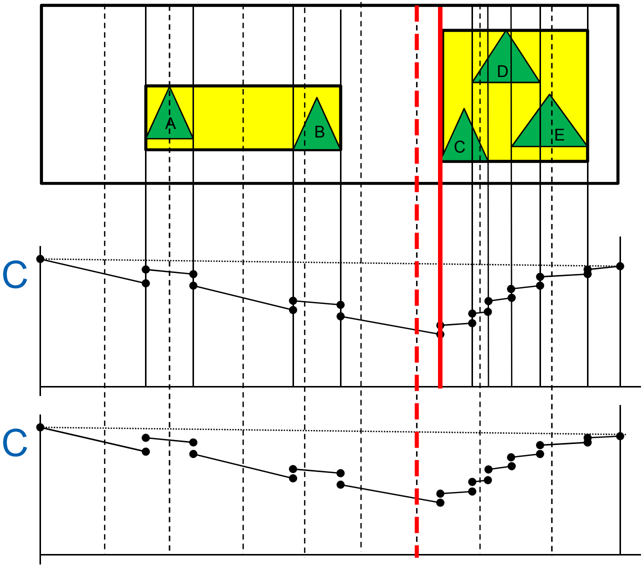


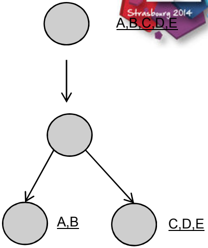
- Only some combinations are explored
- Bounding boxes can overlap
- One bounding box takes more memory

Cost based on SAH Evaluation Modes

- Exact algorithm e.g. using sweeping technique for N primitives maximum 2.N evaluations. For kd-trees [Havran 2001, Wald and Havran 2006], for BVH [Wald 2007]
- Approximate algorithm – some prescribed number of bins either fixed or using some formula. For kd-trees [Hunt et al. 2006, Popov et al. 2006], for BVHs [Havran et al. 2006]
- It can be combined together
  - Upper tree levels (closer to root node) – approximate algorithm.
  - For smaller number of geometric primitives – exact algorithm

Exact versus Approximate Cost Evaluation





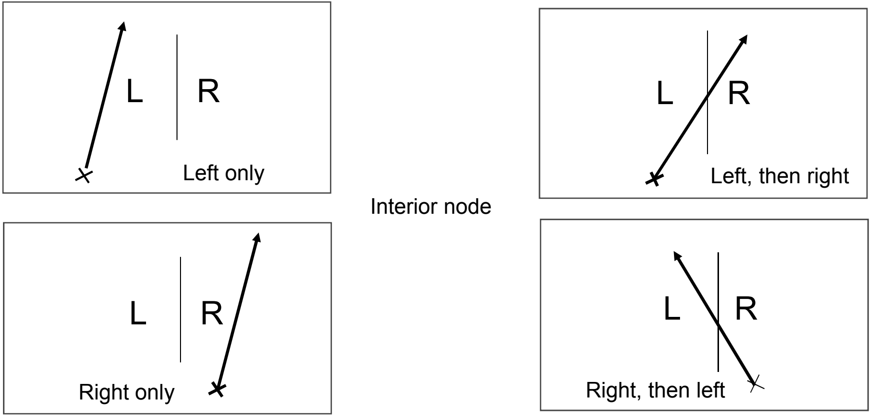
- Exact using boundaries
- Approximate with 8 samples

Top-Down Building Termination Criteria

- Kd-trees local: using a stack
  - *Simple local*: maximum depth (k_1+k_2.log N) + number of geometric primitives is limited (e.g. 2 or 1 primitive)
  - *More complicated local*: a maximum number of cost improvement failures + maximum estimated depth + number of objects
- BVH – not needed, but usually more geometric primitives in a leaf (2 to 8)
- Kd-trees and BVHs: Global using a priority queue
  - Maximum memory used
  - Maximum memory used + maximum leaf cost

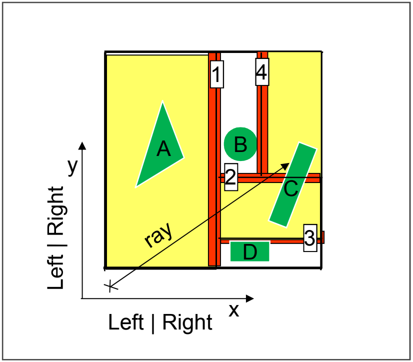
Recursive Ray Traversal Algorithm Cases

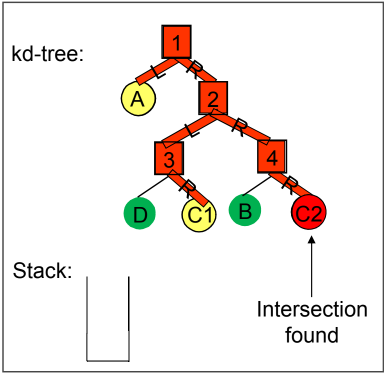
- Assuming binary hierarchy (both BVH and kd-tree)



Traversal Algorithm for Hierarchies

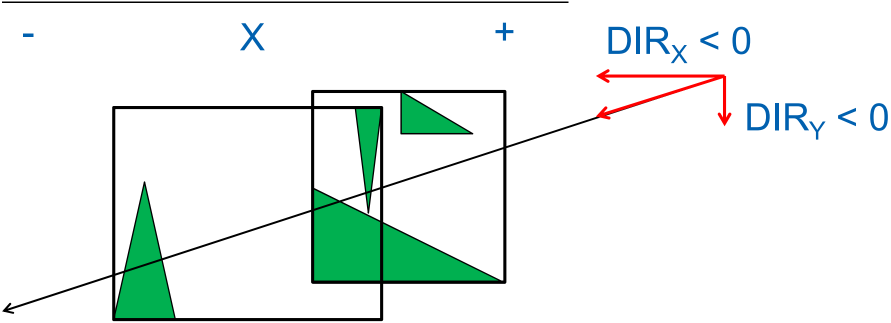
- Kd-tree traversal algorithm with a stack





BVH Traversal Algorithm

- Similar, but the ray has to be checked along its traversed path until the first intersection found
- The bounding boxes in principle arbitrary, in practice a single axis orientation is encoded as for kd-trees in 2 bits

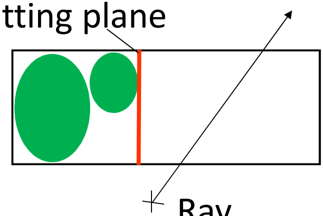


Some Notes on the Cost Model with SAH

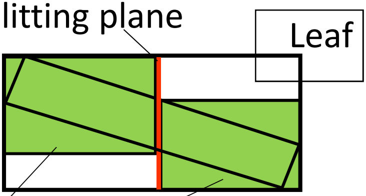
- The data structures with cost model can be several times more efficient than a spatial/object median
- The cost model based on SAH is not ideal as underlying assumptions are not fulfilled
  - Distribution of rays is not uniform
  - Rays can intersect objects so they are of finite length
  - Rays can also have origin inside the scene
- Some nice tricks are possibly to reduce both expected cost and improve the performance

Kd-trees Efficiency Improvements

Cutting off empty space



Reducing objects' axis-aligned bounding boxes



BVH Efficiency Improvements

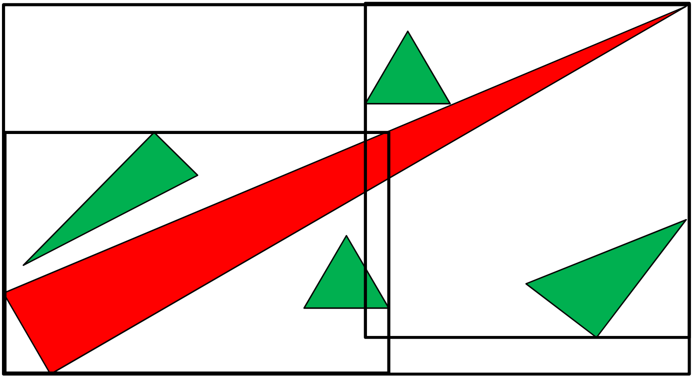
- Shallower BVHs for parallel SIMD traversal [Dammertz 2008], four child nodes by tree compaction
- Optimization algorithms for already existing BVHs
  - Rotation based [Kensler 2008]
  - Rotations with GPUs [Kopta et al. 2012]
  - Insertion/removal based algorithm [Bittner et al. 2013]
  - Treelet based parallel optimization [Karras and Aila 2013]
- Fast algorithm for building HLBVHs (hierarchical linear BVH) in parallel using Morton codes
- Zero storage proposed variant of BVH based on heap by [Eiseman et al. 2012], but in general inefficient

SBVH – Hybrid between Kd-tree and BVH

- Split BVH proposed originally by [Havran 2007], particular algorithms by [Ernst and Greiner 2008], [Popov et al. 2009], and [Stitch et al. 2009]
- Idea – base is BVH, but problematic primitives are allowed to be referenced several times as in kd-trees
- The idea of more references corresponds to split clipping proposed for kd-trees
- Build algorithms
  - [Year 2008] – subdivide in advance, early split clipping
  - [Year 2009] – late split clipping - local greedy algorithm decision for top-down construction

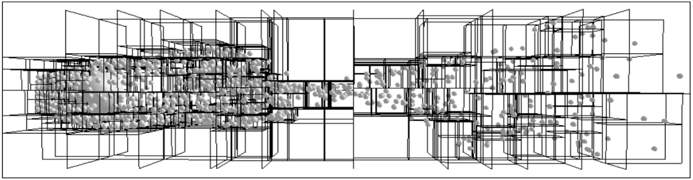
SBVH – Solved Geometrical Situation

- The typical situation solved by more references – elongated triangles not aligned to coordinate axes

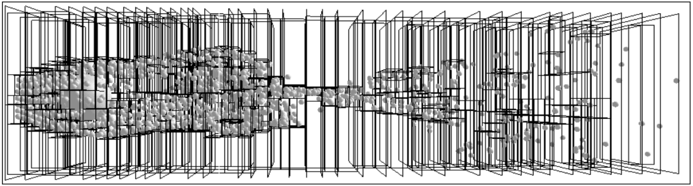


Construction for Preferred Ray Sets

- Possible for both BVHs and Kd-trees
- Idea: non-uniform distribution of rays, gain 100-200% maximum



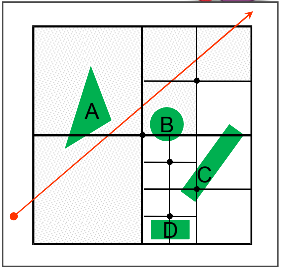
Uniform



Parallel projection

Ray Tracing with Octrees

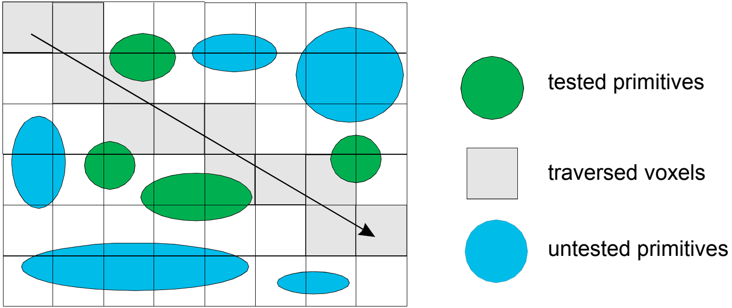
- Interior node branching factor is eight
- Up to four child nodes can be traversed in an interior node
- Traversal algorithm necessarily more complicated than for kd-tree
- Octrees are less adaptive to the scene object distributions than kd-trees
- Geometric probability can be used in the same way as for kd-trees [Octree-R, 1995]
- Octrees are less efficient than kd-trees due to the implementation constants
- Most recently Loose Octrees [Ulrich 1999]
- Most efficient traversal algorithm [Revelles et al. 2000]



Note: octrees can be simulated by kd-trees

Ray Tracing with Uniform Grids

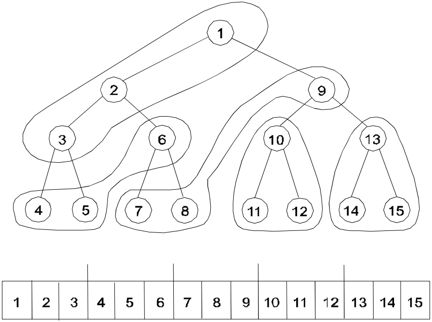
- Arity of a node proportional to the number of objects
- Traversal method based on 3D differential analyzer (3DDA)
- For skewed distributions of objects in the scene it is inefficient
- For highly and moderately uniform distributions of objects it is slightly more efficient than kd-trees



Data Layout in Memory

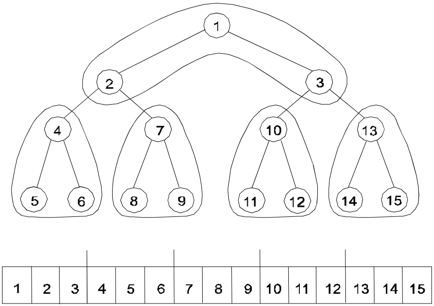
- Minimizing memory footprint
- Minimizing latency by treelets

By standard memory allocator



Depth-first-search (DFS)

Treelets



Van Emde Boas

Performance Model for Ray Tracing

- Cost = cost for intersections +

  cost for traversal +

  cost for reading data

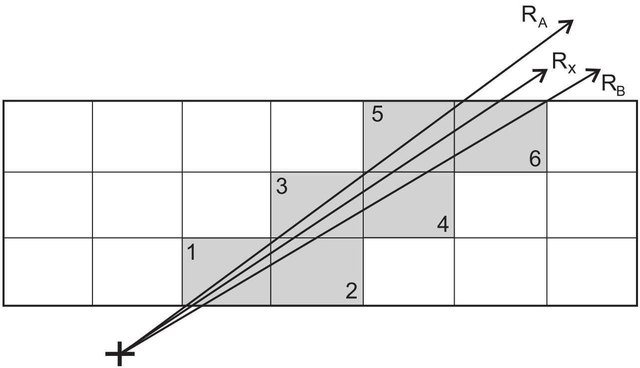
- Faster ray-object intersection tests
- Decreasing number of ray-object intersection tests
- Faster traversal step
- Decreasing number of traversal steps
- Reducing memory throughput and latency

Offline Ray Shooting

- Shooting several rays at once
- Rays are formed by camera, by viewing frustum or by point light sources
- Rays are **coherent** = similar in direction and origin
- Problem can be formulated as offline setting of searching
- We can amortize the cost of traversal operations though the data structure … the number of traversal steps is decreased typically by 60 to 70%

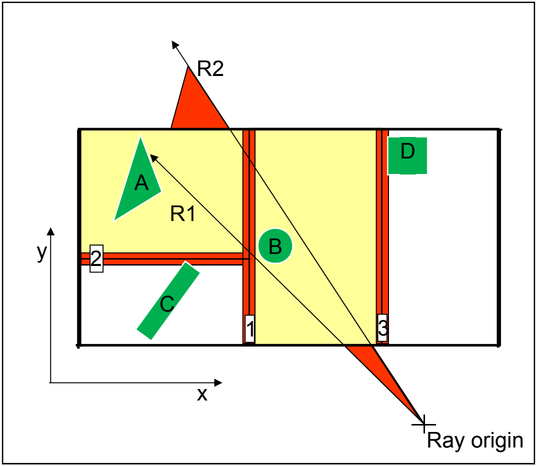
Offline Ray Shooting: Coherence

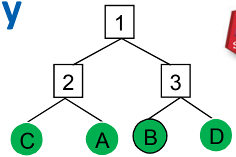
- If boundary rays traverse the same sequence S of leaves, then all rays in between also traverse the same sequence.
- Proof by convexity (convex leaves, convex shaft)

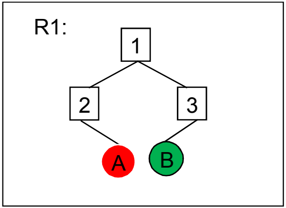


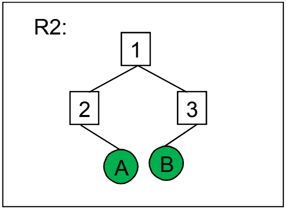
Offline Ray Tracing with Hierarchy

- Pruning the search in the tree



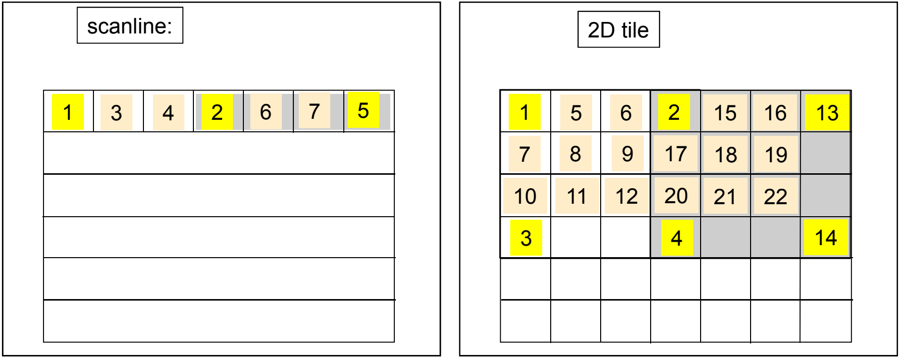






Offline Ray Tracing for Primary Rays

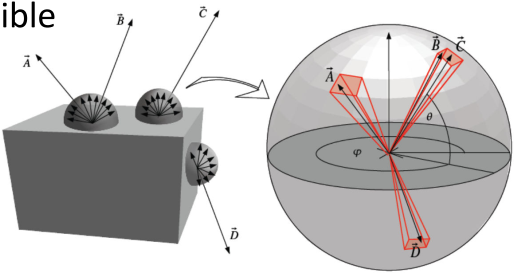
- Hidden surface removal based on offline algorithm
- Rays have common origin, viewing frustum by image



- Other applications: hierarchical image sampling

Ray Sorting to Improve Coherence

- Store the rays into cache according to direction
- When a bucket is filled in, shoot all of them at once
- Improves access pattern for incoherent queries
- Speedup up to 30% (CPU [EG, Havran et al. 2005]) and 300% (GPU [EG, Garanzha, 2010])
- Other schemes possible for offline setting
- Not for primary rays – already coherent



Shadow Rays Tricks

- For shadow rays we can get ANY INTERSECTION or NO intersection, we do not need the first intersection
- Typical task in many light methods (virtual points lights) in global illumination algorithms.
- We can relax the traversal order in BVHs and kd-trees
- The good traversal order
  - Precomputed and stored in one bit in each interior node [Ize+Hansen, EG 2011]
  - Taking also ray distribution into account [Feltman et al. 2012]
  - Computed on the fly [Nah+Manocha, CGF 2014] from SA
- Time reduction to approximately half in best scenes
- Cannot help if the shadow rays are unoccluded

What Was not Presented

- Ray packets – for coherent (primary) rays, the use of SIMD (SSE, AVX etc. instructions)
- Ray tracing on GPUs (2nd part of the tutorial)
- Ray tracing on mobile devices (smartphones) and other special architectures as game consoles.
- Hardware for ray tracing
- Ray-geometric primitive intersection algorithms for example NURBS
- Application scenarios in computer graphics (rendering, collision detection, games) and the other ones

PhD Theses on Ray Tracing in Last 20 years

- V. Lu: Multicore Construction of K-D Trees with Applications in Graphics and Vision, 2014.
- S. Popov: Algorithm and Data Structures for Interactive Ray Tracing on Commodity Hardware, 2012.
- J. Bikker: Ray Tracing in Real-time Games, 2012.
- T. Ize: Efficient Acceleration Structures for Ray Tracing Static and Dynamic Scenes, 2009.
- W. Hunt: Data Structures and Algorithms for Real Time Ray Tracing at the university of Texas at Austin, 2009.
- C. Benthin: Realtime Ray Tracing on Current CPU Architectures, 2006.
- A. Y-H. Chang: Theoretical and Experimental Aspects of Ray Shooting, 2005.
- I. Wald: Real Time Ray Tracing and Global Illumination, 2004.
- V. Havran: Heuristic Ray Shooting Algorithms, 2001.
- G. Simiakakis: Accelerating Ray Tracing with Directional Subdivision and Parallel Processing, 1995.

Thank you!

TUTORIAL OVERVIEW

Introduction (5 min)	VH
Sorting and Searching Techniques (30 min)	JB
Hierarchical Data Structures (30 min)	JB/VH
Ray Tracing (20 min)	VH
Q & A (5 min)	
Coffee break	
Rasterization and Culling (25 min)	JB
Photon Maps and Ray Maps (20 min)	VH
Irradiance Caching (5 min)	VH
BRDF and BTF (10 min)	VH
Sorting and searching on GPU (20 min)	JB
Q & A (10 min)	

EUROGRAPHICS 2014 TUTORIAL

RASTERIZATION AND CULLING

JIŘÍ BITTNER

Czech Technical University in Prague

DCGI

TUTORIAL OVERVIEW

Introduction (5 min)	VH
Sorting and Searching Techniques (30 min)	JB
Hierarchical Data Structures (30 min)	JB/VH
Ray Tracing (20 min)	VH
Q & A (5 min)	
Coffee break	
Rasterization and Culling (25 min)	JB
Photon Maps and Ray Maps (20 min)	VH
Irradiance Caching (5 min)	VH
BRDF and BTF (10 min)	VH
Sorting and searching on GPU (20 min)	JB
Q & A (10 min)	

Rasterization - Hidden Surface Removal

- Find visible surface for every pixel (ray)

Problem	Q	S	A
Ray shooting	ray	{objects}	point
Hidden Surface Removal	**{rays}**	**{objects}**	**{points}**
Visibility culling	{rays}	{objects}	{objects}
Photon maps	point	{points}	{points}
Ray maps	point	{rays}	{rays}
Irradiance caching	point	{spheres}	{spheres}

Hidden Surface Removal

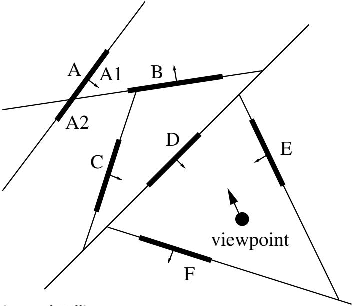
- List priority algorithms
- Area subdivision algorithms
- Scan-line algorithms
- Z-buffer
- Ray casting

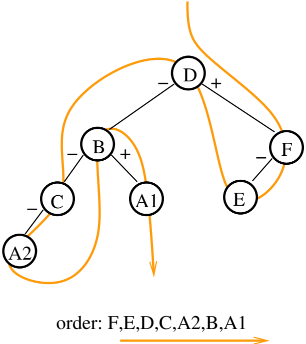
Depth Sort

- Draw faces back to front [Newell72]
- Overwrite the farther ones (painter's alg.)
- Determine strict depth order
  - Resolve cycles of overlaping polygons
- Step 1: depth sort (Z)
  - Quick sort, bubble-sort (temporal coherence)
- Step 2: rasterization (YX)
  - Bucket sort to pixels

Depth Sort with BSP Tree

- BSP built in preprocess
  - Select a plane
  - Partition the polygons in front/back fragments
  - If >1 polygon → recurse
- Quick-sort like, heuristics for splitting-plane selection





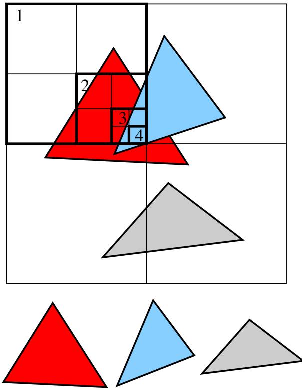
Depth Sort with BSP Tree

- Tree size: O(n2)
- Run-time: simple traversal

- Improvements
  - BSP need not be autopartition!
  - For manifolds depth order can be predetermined → coarser BSP
  - Generalization to all BSP nodes
    'Feudal priority tree' [Chen96]

Area Subdivision

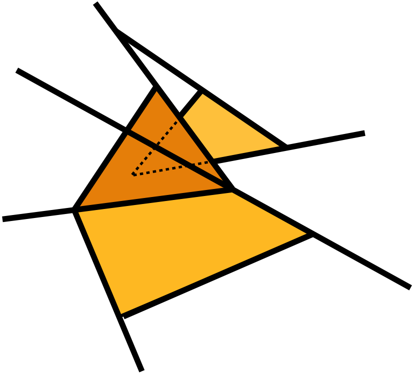
- Subdivide screen space [Warnock69]
- Classify polygons with respect to the area
- Terminate if trivial solution
- Step 1: octree subdivision (XY)
  - Quick sort like
- Step 2: list for octree nodes (Z)
  - Insertion sort



1	I	I	D
2	I	I	
3	S	I	
4	S	S	

Naylor's BSP projection

- Draw polygons front to back
- Clip polygons by 2D BSP of projected polygons
- Step 1: depth sort (Z)
  - 3D BSP built in preprocess
- Step 2: 2D BSP (XY)
  - Quick sort like subdivision of the projection plane

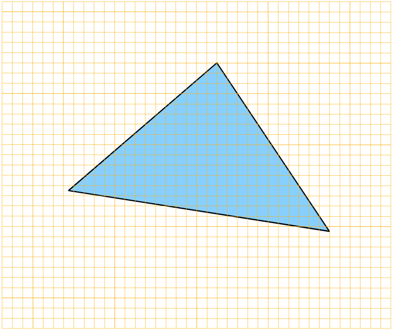


Scan-Line

- Sort by scan-lines (Y)
- Sort spans within a scanline (X)
- Search for closest span (Z)
- [Watkins70]
  - Bubble sort in X and Y
  - O(log n) search in Z

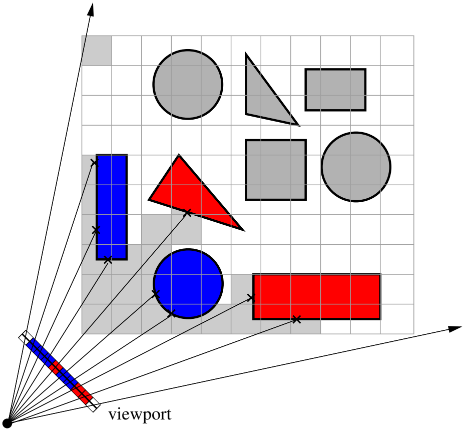
Z-buffer

- Rasterize polygons in arbitrary order
- Maintain per pixel depths
- Step 1: rasterization (YX)
  - Bucket sort like
- Step 2: per pixel depth comparison (Z)
  - Min selection



Ray Casting

- Cast ray for each pixel
- Step 1: spatial data structure (XYZ)
  - Preprocess
  - Trees ~ quick sort
  - Grid ~ distribution sort
- Step 2: search for nearest intersection
  - Min selection with early termination



Z-buffer vs. Ray Casting

	scan-line coherence	presorting	output sensitive
Z-buffer	yes +	no +	no -
Ray casting	no -	yes -	yes +

- Z-buffer better in simple sparsely occluded dynamic scenes
- Ray casting better in complex densely occluded static scenes

HSR - Summary

- Search for closest object for every pixel (ray)
- HSR algorithms sort in
  - Directions (XY)
  - Depth (Z)
  - Differ in sorting order and methods [Suth74]
- Current winners: z-buffer, ray casting

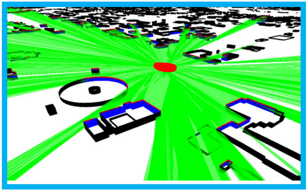
Visibility Culling

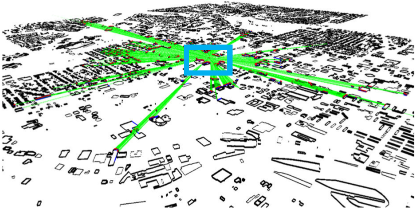
Find visible objects for a given view point or view cell

Problem	Q	S	A
Ray shooting	ray	{objects}	point
Hidden Surface Removal	{rays}	{objects}	{points}
Visibility culling	**{rays}**	**{objects}**	**{objects}**
Photon maps	point	{points}	{points}
Ray maps	point	{rays}	{rays}
Irradiance caching	point	{spheres}	{spheres}

Visibility Culling – Motivation

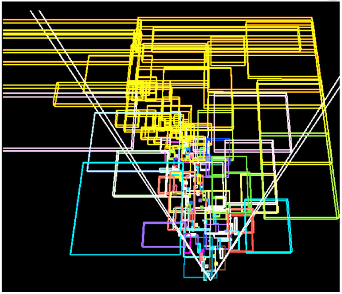
- Q: Why visibility culling?
  - Object outside screen culled by HW clipping
  - Occluded objects culled by z-buffer in O(n) time
- A: Linear complexity not sufficient!
  - Processing too many invisible polygons
- Goal
  - Render only what can be seen!
  - Make z-buffer output sensitive



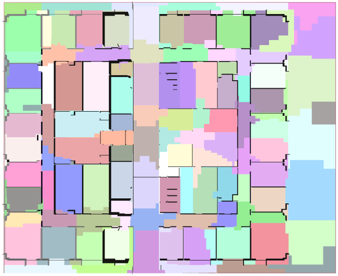


Visibility Culling - Introduction

- Online
  - Applied for every view point at runtime



- Offline
  - Partition view space into view cells
  - Compute Potentially Visible Sets (PVS)

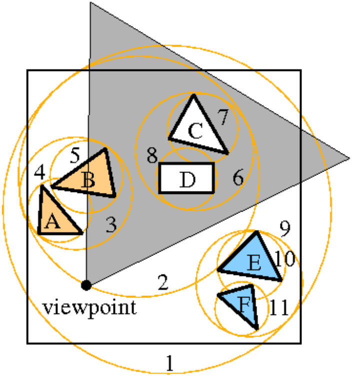


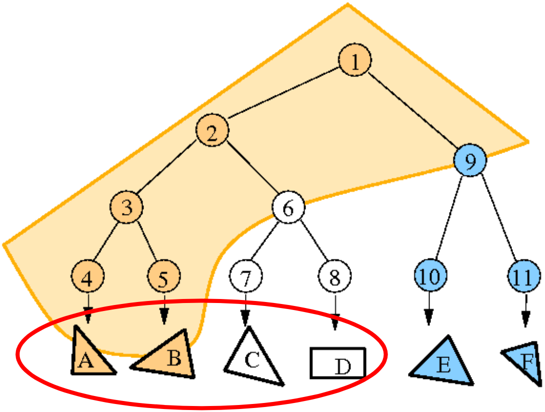
Online Visibility Culling

- For every frame cull whole groups of invisible polygons
- Conservative solution
  - Determine a superset of visible objects
  - Precise visibility solved by z-buffer
- Hierarchical data structures
  - kD-tree, octree, BVH
- View-frustum culling
- Occlusion culling
  - CPU techniques
  - GPU based (HW occlussion queries)

View Frustum Culling

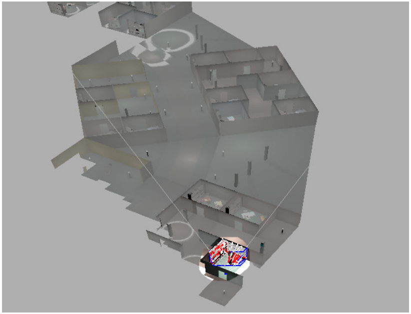
- Objects intersecting the view frustum
- Hierarchical VFC
  - Spatial hierarchy: kD-tree, BSP tree, octree, BVH





Occlusion Culling

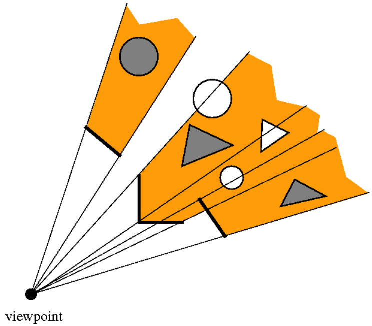
- VFC disregards occlusion
- 99% of scene can be occluded!



- Solution: Detect and cull also occluded objects

Shadow Frusta

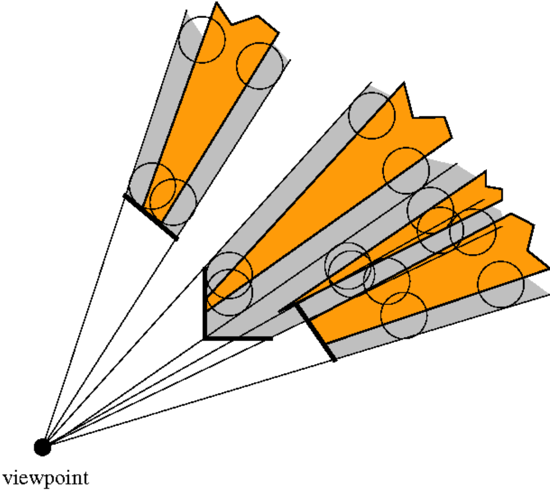
- Construct shadow frusta for several occluders [Hudson97]

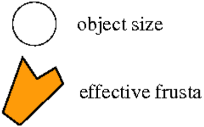


- Object is invisible if inside a shadow frustum
- Queries on the spatial hierarchy

Shadow Frusta - Properties

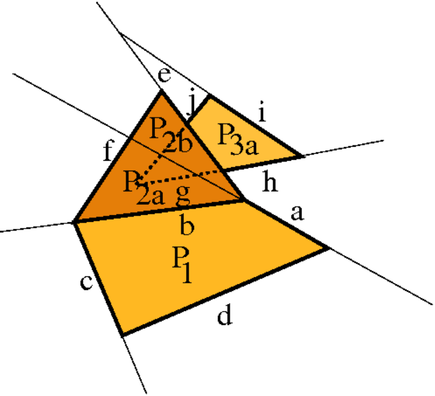
- Properties
  - + Easy implementation
  - No occluder sorting
    - No occluder fusion!
    - O(n) query time
  - Small number of occluders

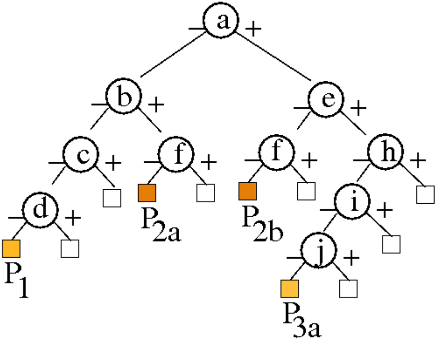




Occlusion Trees

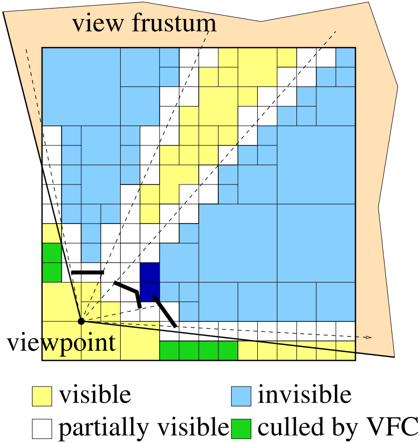
- Occluders sorted into a 2D BSP tree [Bitt98]
- Occlusion tree represents fused occlusion
- Example: occlusion tree for 3 occluders





Occlusion Tree - Traversal

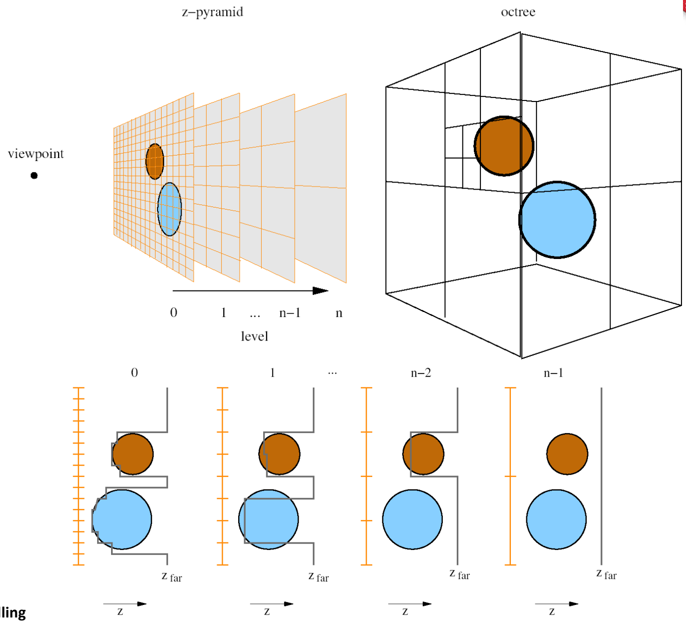
- Visibility test of a node
  - Depth-first-search
  - Found empty leaf $\rightarrow$ tested object is visible
  - Depth test in filled leaves
- Presorting occluders
  - Tree size: worst case O(n2),
  - O(log n) visibility test
- Allows to use more occluders
- Not usable for scenes with small polygons



Hierarchical Z-buffer

- Extension of z-buffer to quickly cull larger objects [Greene 96]
- Ideas
  - octree for spatial scene sorting
  - z-pyramid for accelerated depth test

Hierarchical Z-buffer - Example

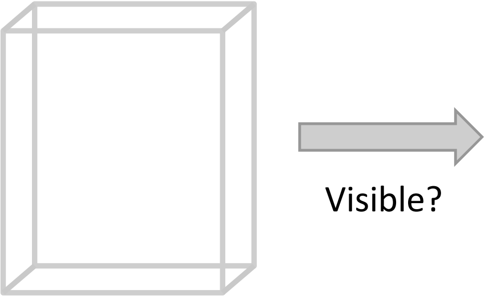


Hierarchical Z-buffer - Usage

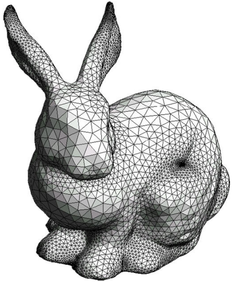
- Hierarchical test for octree nodes
- Find smallest node of z-pyramid, which contains the tested box
- Box depth > node depth $\rightarrow$ cull
- Otherwise: recurse to lower z-pyramid level
- Optimization: use temporal coherence
  - z-pyramid constructed from polygons visible in the last frame

HW Occlusion Queries

- Query visibility from view point
- No preprocessing
- Dynamic Scenes
- Hardware occlusion queries → # visible pixels



Visible?

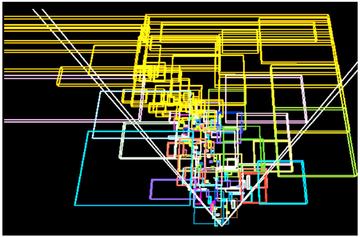


Query bounding box

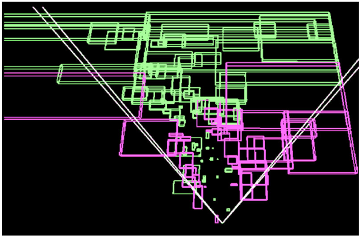
Render geometry

HW Occlusion Queries

- Issues
  - Latency – the result not readily available, the query costs time
- CHC/CHC++ [Bitt04,Matt08]
  - Adapting hierarchy levels to be queried
  - Interleaving querying and rendering
  - Minimizing state changes, Query batching



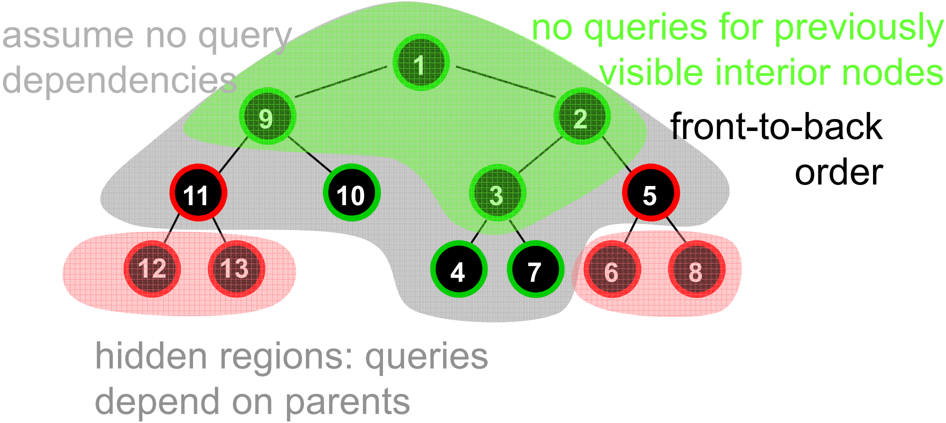
CHC: ~100 state changes



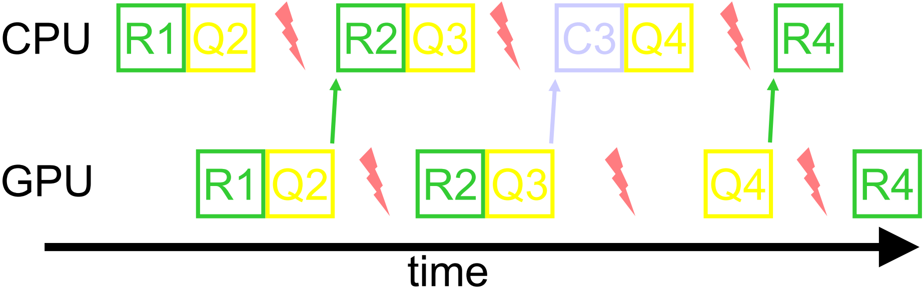
CHC++: 2 state changes

CHC / CHC++

Heavy use of temporal & spatial coherence`



CPU Stalls & GPU Starvation



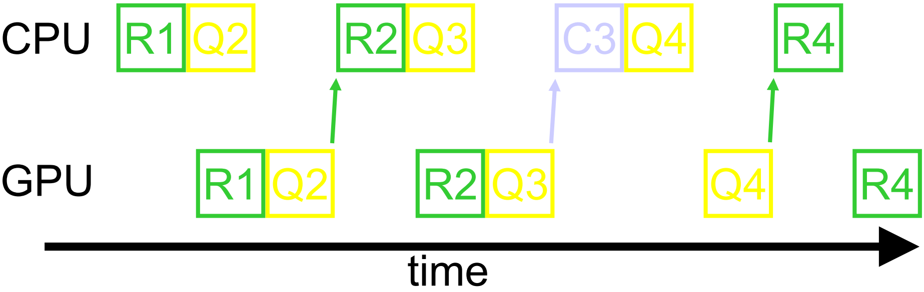
Rx Render object x

Qx Query object x

Cx Cull object x

Waiting time

CHC



Rx Render object x

Qx Query object x

Cx Cull object x

Cells and Portals

- Partition the scene in cells and portals
  - Cells ~rooms
  - Portals ~ doors&windows
- Cell adjacency graph
- Constrained search
  - Portal visibility test [Luebke 96]

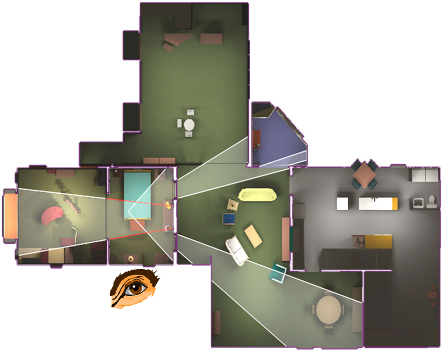
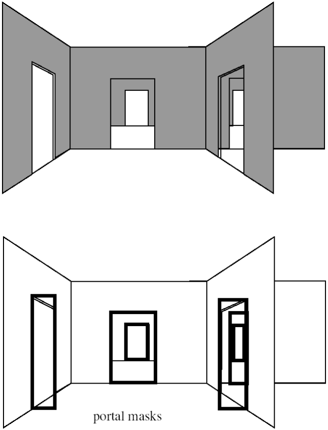
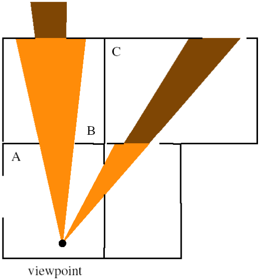


Image courtesy of D. Luebke

Portal Visibility Test

- Intersection of bounding rectangles of portals





Cells and Portals Example

- Viewpoint in cell E

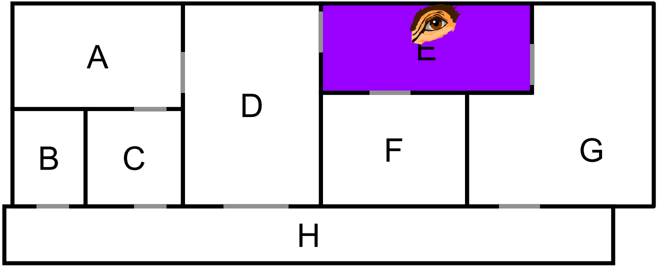
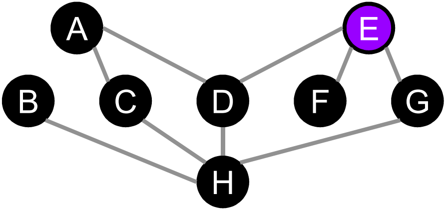
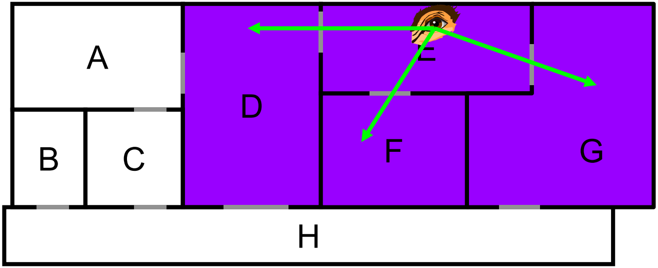


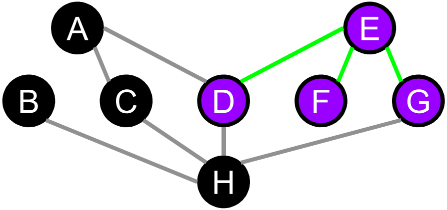
Image courtesy of D. Luebke



Cells and Portals - Example

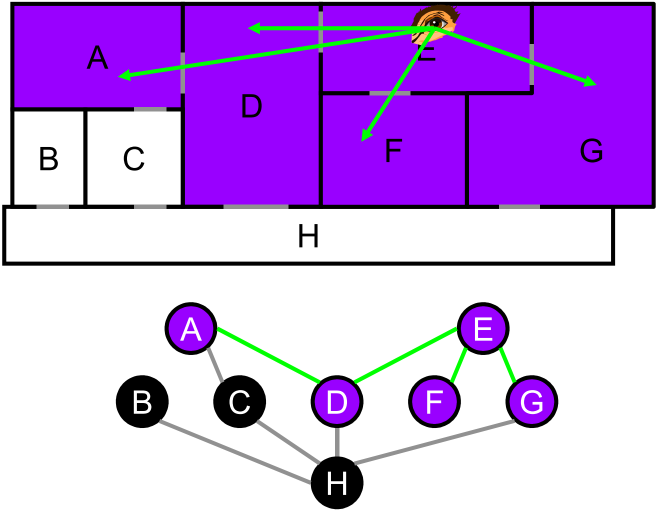
- Adjacent cells DFG





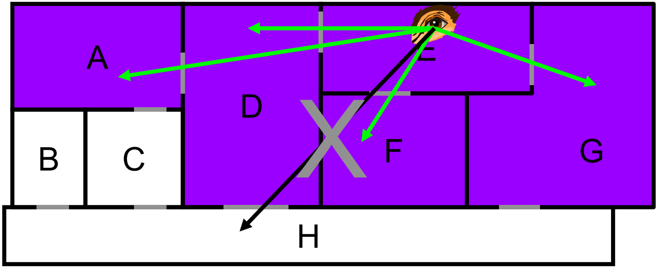
Cells and Portals - Example

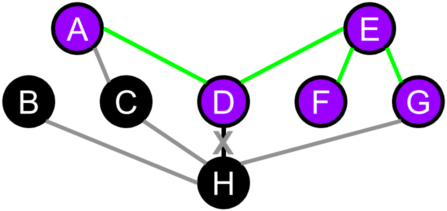
- Cell A visible through portals E/D+D/A



Cells and Portals - Example

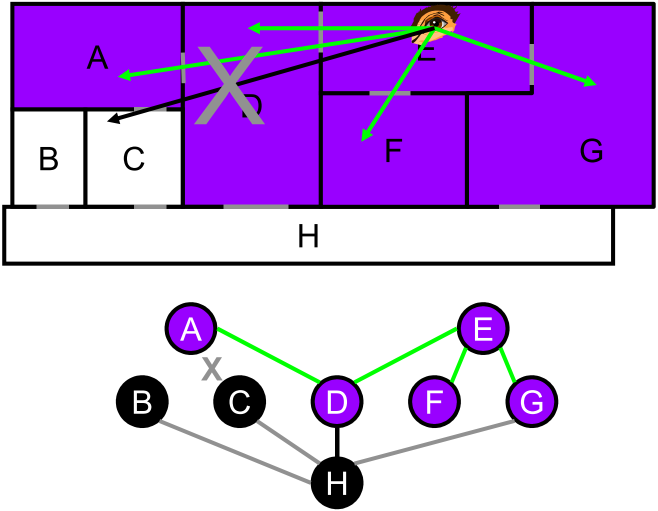
- Cell H not visible through portals E/D+D/H





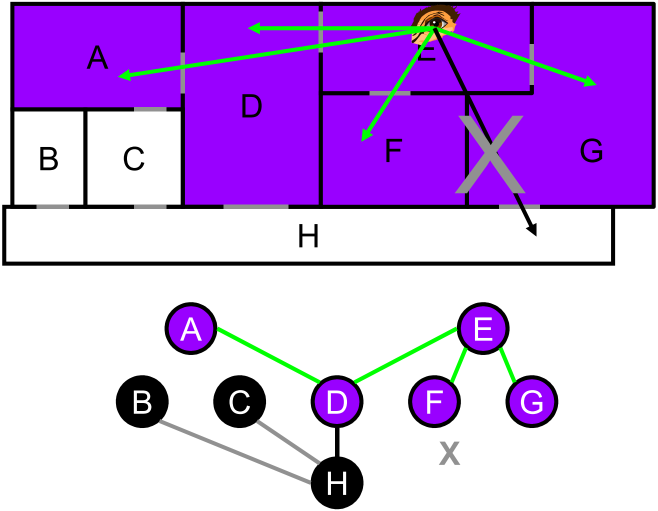
Cells and Portals - Example

- C not visible through portals E/D+D/A+A/C

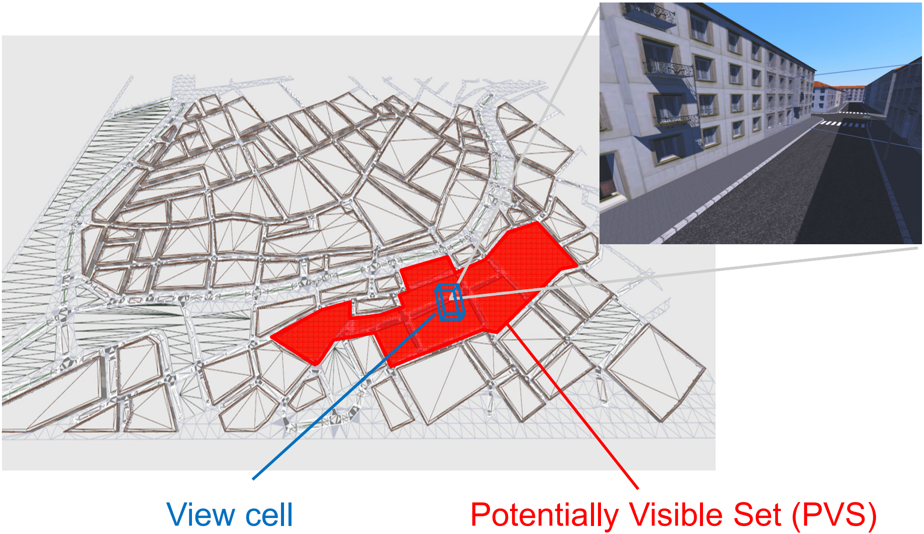


Cells and Portals - Example

- H not visible through portals E/G+G/H



Offline Visibility Culling

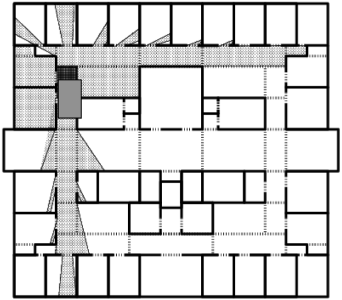


Visibility Preprocessing

- Preprocessing
  - Subdivide view space into view cells
  - Compute Potentially Visible Sets (PVS)
- Usage
  - Find the view cell (point location)
  - Render the associated PVS
- Other benefits
  - Prefetching for out-of-core/network walkthroughs
  - Communication in multi-user environments
- Problems
  - Costly computation (treats all view points and view directions)
  - PVS storage

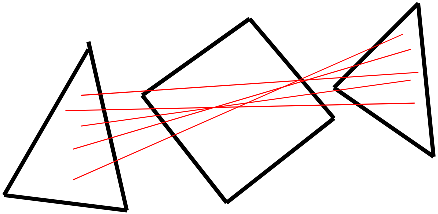
Interiors – Cells and Portals

- Subdivide the scene into cells and portals
- Constrained DFS on the adjacency graph
  - Portal visibility test
- More complex than the online algorithm
  - We do not have a view point!



Interiors – Cells and Portals

- Sampling [Airey90]
  - Random rays
  - Non-occluded ray → terminate

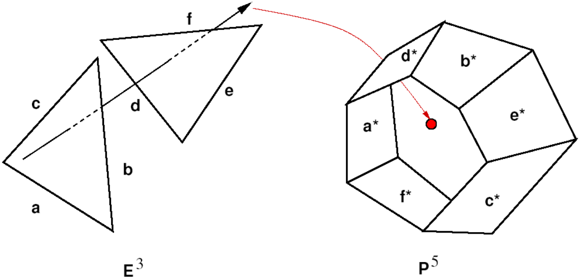


-

+ Simple implementation
- \- Approximate solution

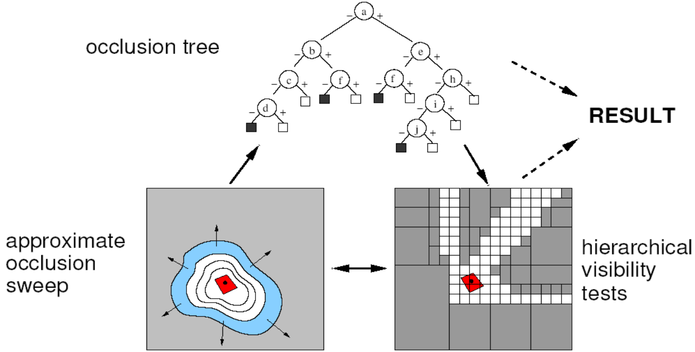
Interiors – Cells and Portals

- Exact computation [Teller 92]
  - Mapping to 5D (Plücker coordinates of lines)
- Portal edges $\rightarrow$ hyperplanes Hi in 5D
- Halfspace intersection in 5D



General Scenes - Occlusion Tree

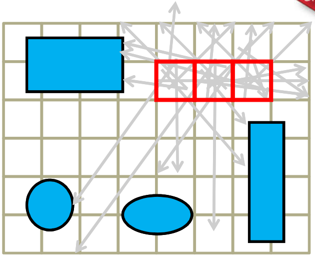
- Extension of the 2D occlusion tree
- 5D BSP tree
  - Plücker coordinates of lines
- The tree represents union of occluded rays



Adaptive Global Visibility Sampling

- Classical from-region visibility

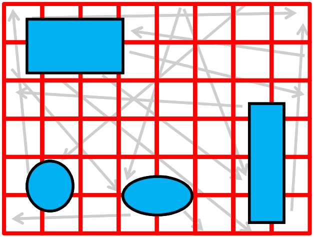
```
for each view cell
  Compute PVS
```



- Global visibility sampling

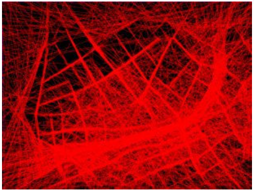
```
while !terminate
  Compute vis. sample
  Add to all PVSs
```

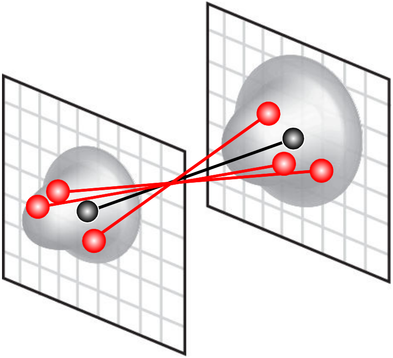
  - Uses visibility coherence
  - And is progressive

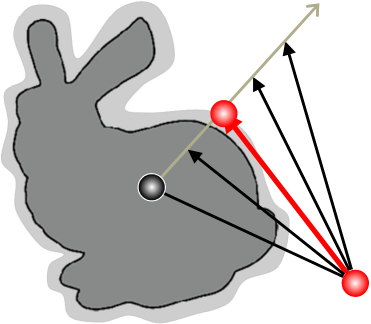


Adaptive Global Visibility Sampling

- Use ray distributions that adapt to visibility changes







Visibility Culling - Summary

- Find visible objects for a view point or view cell
- Heavy use of spatial sorting
  - Common HDS: kD-tree, octree, BVH
- Occlusion culling differs in occluder sorting
  - No sorting, occlussion trees, HOM, cells + portals
- Online vs. offline culling
  - Online: dynamic scenes
  - Offline: very fast at runtime for static scenes

Surveys on Visibility

- C. Dachsbacher: Analyzing Visibility Configurations, 2011.
- J. Bittner and P. Wonka: Visibility in computer graphics, 2003.
- D. Cohen-Or et al.: A survey of visibility for walkthrough applications, 2003.
- F. Durand. 3D Visibility: Analytical Study and Applications, 1999.

EUROGRAPHICS 2014 TUTORIAL

PHOTON MAPS AND RAY MAPS

VLASTIMIL HAVRAN

Czech Technical University in Prague

DCGI

TUTORIAL OVERVIEW

Introduction (5 min)	VH
Sorting and Searching Techniques (30 min)	JB
Hierarchical Data Structures (30 min)	JB/VH
Ray Tracing (20 min)	VH
Q & A (5 min)	
Coffee break	
Rasterization and Culling (25 min)	JB
Photon Maps and Ray Maps (20 min)	VH
Irradiance Caching (5 min)	VH
BRDF and BTF (10 min)	VH
Sorting and searching on GPU (20 min)	JB
Q & A (10 min)	

Density Estimation

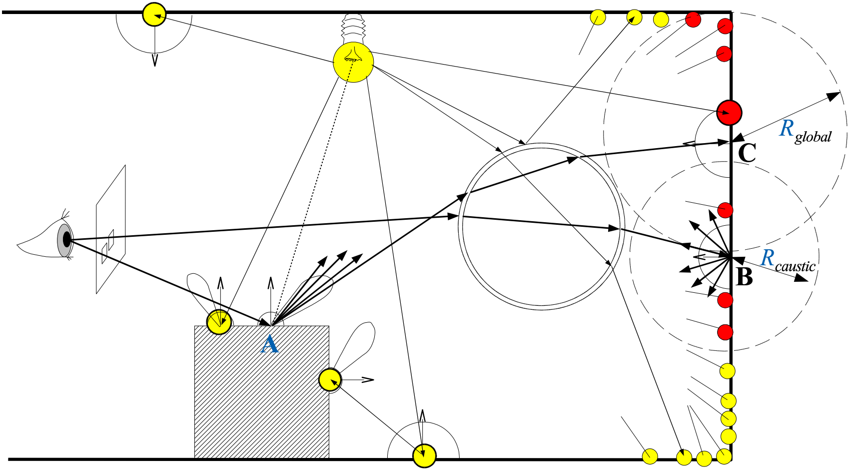
- Photon maps
- Ray maps
- General density estimation
- All of these are applications of
  - Nearest neighbor search (single neighbor)
  - K-nearest neighbor search (K nearest neighbors)
  - Range search (given a range as a sphere)

Photon Maps [Jensen 1993]

- Simple case of density estimation

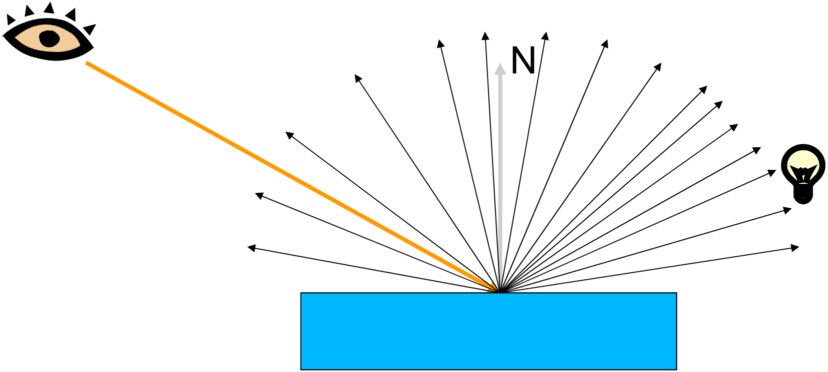
Problem	Q	S	A
Ray shooting	ray	{objects}	point
Hidden Surface Removal	{rays}	{objects}	{points}
Visibility culling	{rays}	{objects}	{objects}
Photon maps	**point**	**{points}**	**{points}**
Ray maps	point	{rays}	{rays}
Irradiance caching	point	{spheres}	{spheres}

Photon Mapping Algorithm



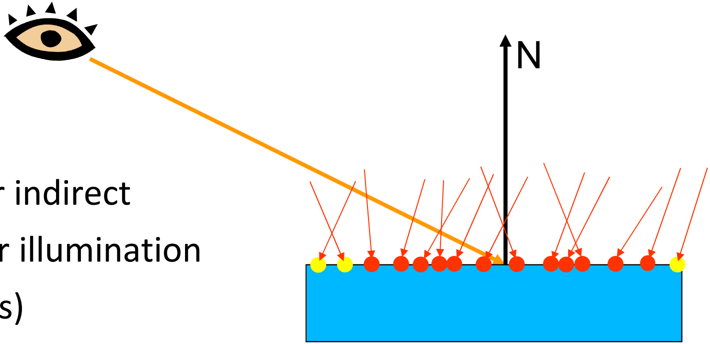
Final Gathering

- Shooting many secondary rays (possibly according to BRDF), gathering radiances from the rays
- Integrating the radiances properly to render image



Direct Visualization of Photon Maps

- Do not shoot final gather rays, use directly visible photons from camera
- It is prone to artifacts on object boundaries referred to as bias

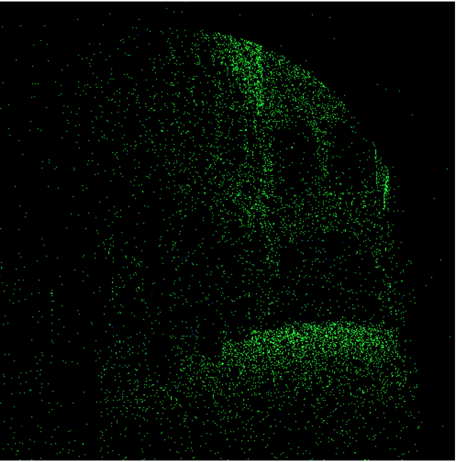


- Used for indirect specular illumination (caustics)

Direct Visualization Example

- Photon Hits

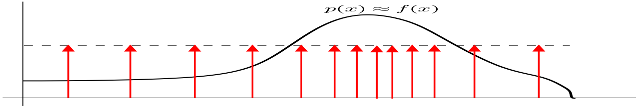
Direct Visualization



Radiance Estimating along Final Gather Rays

- Using the density estimation, from the photon hits estimating *PDF*
- It requires K nearest neighbor searching for each final gather ray
- The number of final gather rays (the number of searches) is enormous
- Typically we shoot 200-4000 final gather rays per pixel
- The number of pixels per image 1-6 x 10^6

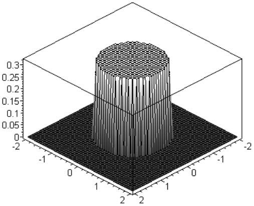
Density Estimation in One Dimension



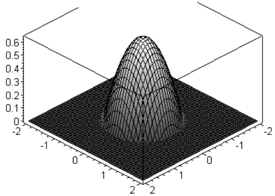
- *Note:* **Importance Sampling**: from given *p(x)* to samples
- **Density Estimation**: from samples reconstruct *p(x)*
- Density estimation requires searching, importance sampling of a function can also use it for tabulated data

Kernels for Density Estimation

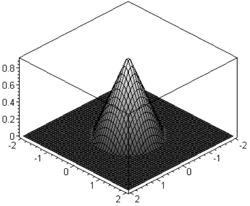
Uniform



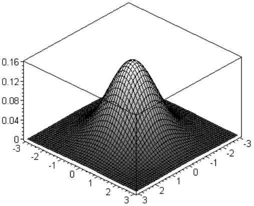
Epanechnikov



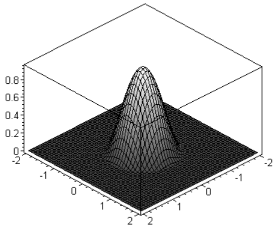
Hat



Gaussian



Biweight

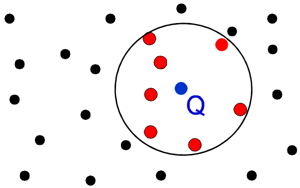


- High efficiency
- Simple formula

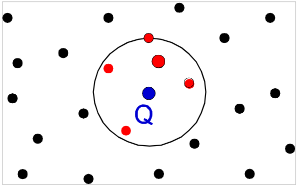
Use of Sorting and Searching

- *Range search* – given a fixed range query (sphere, ellipsoid), find all the photons in the range
- *K nearest neighbor search* – given a center of the expanding shape X (sphere, ellipsoid), find K nearest photons
  - Without considering the direction of incoming photons
  - With considering only valid photons with respect to the normal at point Q

Range Search

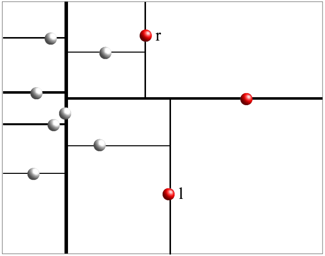


KNN search

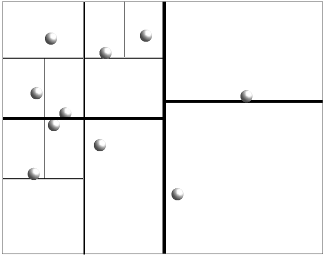


Search Techniques for Points

- Use any data structures described in the section *“Hierarchical Data Structures”*
- Typically kd-trees or kd-B-trees are used



Kd-tree

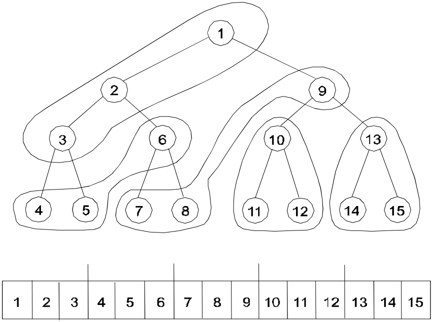


Kd-B-tree

Memory Data Layout

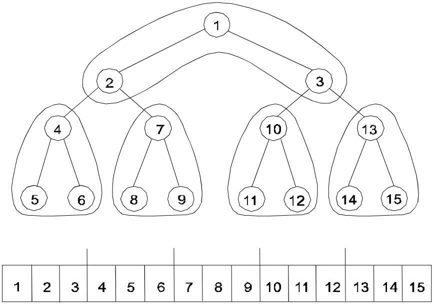
- The same as for ray tracing is possible

By standard memory allocator



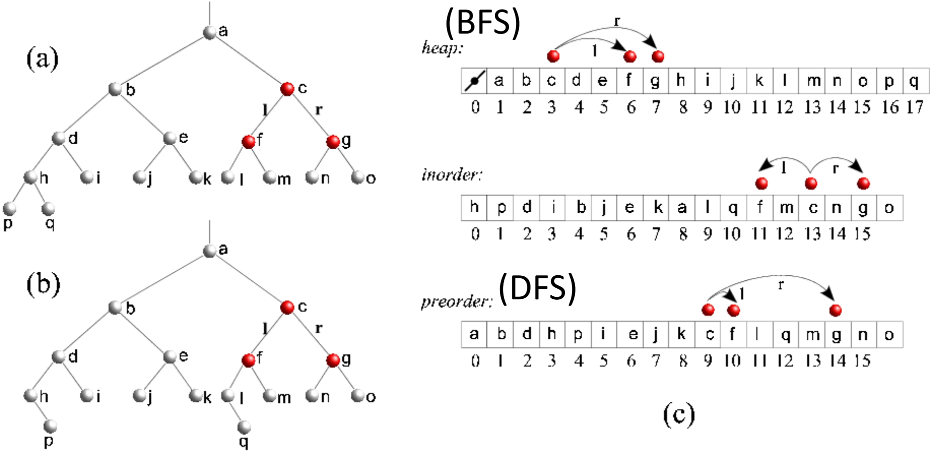
Depth-first-search (DFS)

Treelets



Van Emde Boas

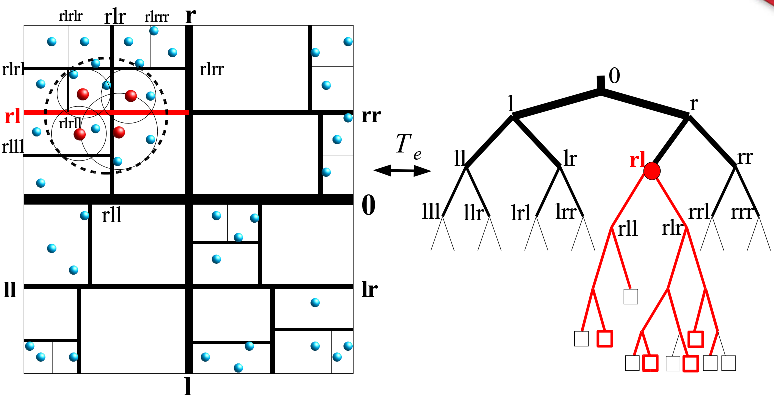
Kd-tree Memory Layout



Practical Yet Efficient Solution

- Use Kd-B-trees
- Construct a tree over an array of photons
- Use 8 Bytes nodes – good packing
- DFS or van Emde Boas Layout
- Sliding mid-point rule = spatial median + shift to nearest photon if one side empty
- One leaf contains a range of 30-70 photons (two indices to photon array)
- Properties:
  - Fast construction time
  - Fast search (complexity proved to be optimal)

Aggregate Searching (Offline Solution)



splitting planes of the spatial kd-tree

first intersected plane <=> start node

density estimation point (ep)

photon hit point

leaf node containing ep

searched leaf

start node for individual queries

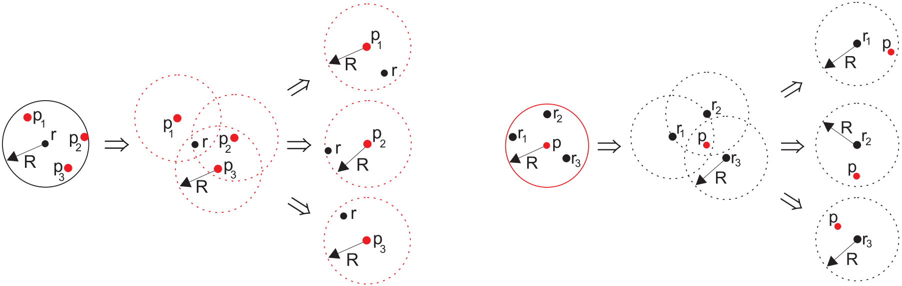
l left child , **r** right child

Searching Tricks for k-NN Search

- Do not use uniform grids, they do not work efficiently for skewed distributions
- Try to avoid a priority queue by using a fixed radius search, where the radius is estimated for given N (from already computed queries or the diagonal of a leaf box)
- Use offline search if possible
- Try to change the role of input data to be queried and queries

Reverse Photon Mapping

- Normal Photon Mapping (gathering energy)
- Reverse Photon Mapping (splatting energy)



- r – ends of final gather rays (in black)
- p – photons (in red)

How To Do Searching Faster?

- Assume that the number of interactions among photons and final gather rays is the same !
- *Traditional Photon Mapping* – a single tree
  - Many searches (~10^9) in a small tree over photons (~10^6)
  - kNN search based on the photon density
- *Reverse Photon Mapping* – more involved *(two trees)*
  - Smaller number of searches (~10^6) in a larger tree over the ends of final gather rays (~up to 10^9)
  - k-NN search is also based on the photon density
- Properties
  - Search in a tree is logarithmic, reverse photon mapping then faster
  - Reverse photon mapping takes more memory

Time Complexity Formulas

- F … number of final gather rays
- K … number of neighbors for kNN search
- V … number of photons
- F.K … number of interactions photon-final gather ray

Traditional Photon Mapping Time:

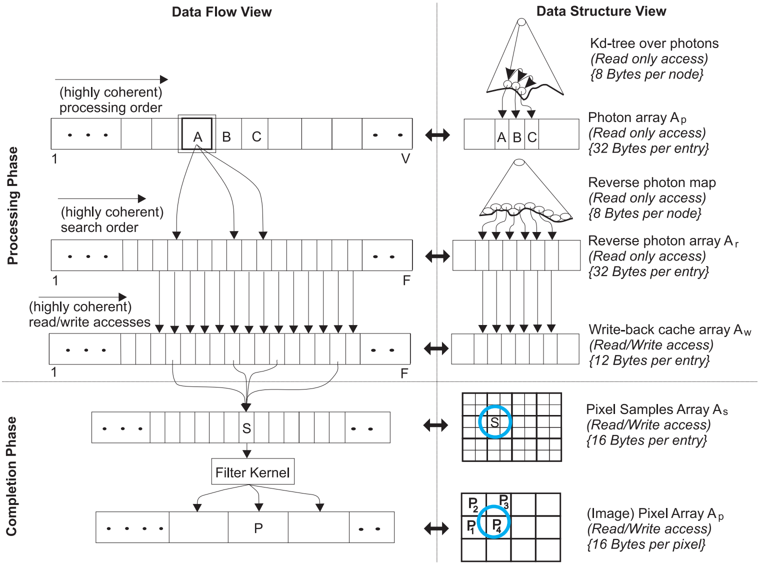
$$C_{PT} = C_1 \,.\, F \,.\, K + C_2 \,.\, F \,.\, \log V$$

Reverse Photon Mapping Time:

$$C_{RPT} = C_1 \,.\, F \,.\, K + C_2 \,.\, V \,.\, \log F$$

For **F >> V** it is easy to show that **F . log V > V . log F**

Data Flow and Data Structures View



Tree Balancing for Searching

- *Balancing Considered Harmful – Faster Photon Mapping using the Voxel Volume Heuristic* [Wald et al. 2005]
- The idea to use voxel volume heuristic for building the tree similarly to SAH as used in ray tracing

$$P_{LEFT} = V_L \ / \ V$$
$$P_{RIGHT} = V_R \ / \ V$$

- The build time 5 to 20 times slower than for spatial median
- The speedup due to the balancing 30% to 400% than for spatial median

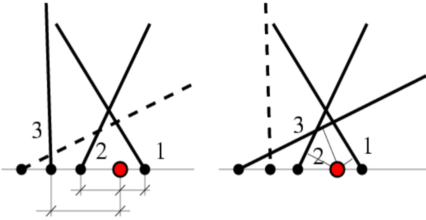
Ray Maps

- [Havran et al. 2005]

Problem	Q	S	A
Ray shooting	ray	{objects}	point
Hidden Surface Removal	{rays}	{objects}	{points}
Visibility culling	{rays}	{objects}	{objects}
Photon maps	point	{points}	{points}
Ray maps	**point**	**{rays}**	**{rays}**
Irradiance caching	point	{spheres}	{spheres}

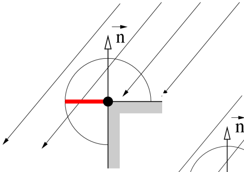
Ray Maps – Extension of Photon Maps

- Ray map: data structure sorting rays not points
- Allows efficient searching for rays
  - Nearest to a point (k-NN)
  - Intersecting a disc/sphere/hemisphere
- Main application: improved density estimation
- Metrics for k-NN search
  1. Distance on the tangent plane
  2. Distance to the ray segment
  3. Distance to the supporting line of the ray

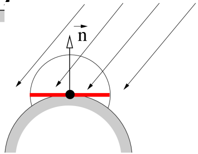


Density Estimation

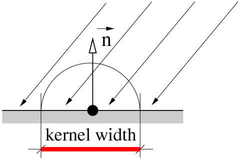
- Problems with photon maps
  - Boundary bias



  - Topological bias



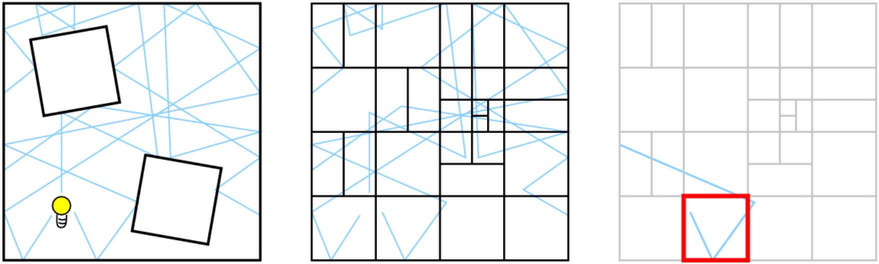
  - Proximity bias



- Ray maps
  - Eliminate boundary bias and topological bias

Ray Map Implementations

- Kd-tree
- Leaves store references to the rays
- Lazy construction driven by the queries
- Support efficient searching and updating



Ray Map Queries

- Queries types
  - Intersection search
  - K-NN search
- Query domains
  - Disc
  - Sphere
  - Hemisphere
  - Axis-Aligned box
  - Possible limitation on ray directions

Ray Map Building

- Spatial median split
- Subdivide if #rays > budget
- Classify rays back, front, both
- Termination criteria
  - #ray references per leaf (~32)
  - Size of the leaf (~0.1% of the scene box)
  - Max tree depth (~30)

Searching Algorithm for Ray Maps

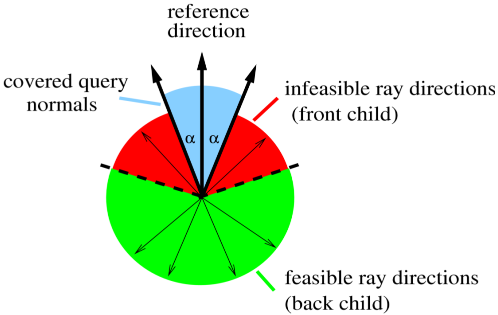
- Intersection search
  - Locate all leaves containing query domain
  - Gather rays
  - Compute intersections
- k-NN search
  - Priority queue
  - Locate the leaf containing the query origin
  - If #rays < N get next node from the queue

Maintenance of Ray Maps

- Deleting a ray
  - Ray cast and remove references
- Adding a ray
  - Ray cast and subdivide if required
- Keeping memory budget
  - Collapsing of unused subtree nodes
  - Least-recently-used strategy

Optimization of Ray Map Search

- Coherence of queries – reducing top down traversal
- Directional splits
  - Queries are oriented
  - Many rays in the opposite direction after reflection
  - Optimization: inserting directional nodes



k-NN Search with Ray Maps

- 1M – 2.5M rays
- Typical memory usage: 16 – 128MB
- Query time (500-NN): 0.2–1.5ms (3.2GHz PC)
- Approx. 2 - 5 times slower than photon maps

Comparison Photon Maps and Ray Maps

Photon maps

Ray maps

Photon maps
+ convex hull

Ray Maps - Summary

- Sorting rays + efficient searching
- Kd-tree implementation
  - Simple implementation
  - Efficient memory usage control
- Density estimation
  - New query domains + new metrics
  - Elimination of boundary bias
  - Reduction of topological bias
  - 2-5x slower than photon maps

Similar Data Structures to Ray Maps

- Ray cache [Lastra02] – hierarchy of spheres
- Volumetric ray density estimation [VanHaevre04]
  - Octree
  - Simulation of plant growth
- Some other concepts do not work
  - Ray → 5D point (Plücker coordinates) and 5D kd-tree
  - Query → 5D polyhedron
  - Really poor performance, culling only at very bottom of the tree

EUROGRAPHICS 2014 TUTORIAL

IRRADIANCE CACHING

VLASTIMIL HAVRAN

Czech Technical University in Prague

DCGI

TUTORIAL OVERVIEW

Introduction (5 min)	VH
Sorting and Searching Techniques (30 min)	JB
Hierarchical Data Structures (30 min)	JB/VH
Ray Tracing (20 min)	VH
Q & A (5 min)	
Coffee break	
Rasterization and Culling (25 min)	JB
Photon Maps and Ray Maps (20 min)	VH
Irradiance Caching (5 min)	VH
BRDF and BTF (10 min)	VH
Sorting and searching on GPU (20 min)	JB
Q & A (10 min)	

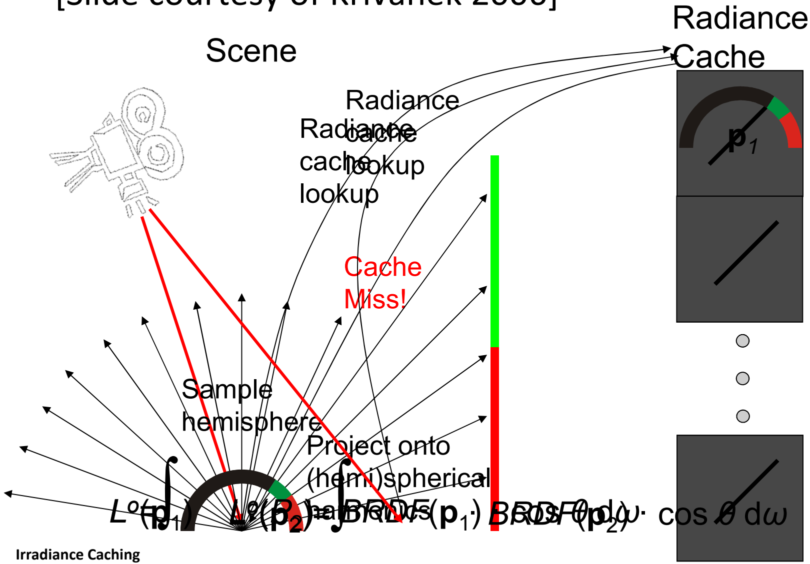
Irradiance Caching [Ward 1988]

- Using interpolation for relatively smooth function of indirect illumination

Problem	Q	S	A
Ray shooting	ray	{objects}	point
Hidden Surface Removal	{rays}	{objects}	{points}
Visibility culling	{rays}	{objects}	{objects}
Photon maps	point	{points}	{points}
Ray maps	point	{rays}	{rays}
Irradiance caching	**point**	**{spheres}**	**{spheres}**

Radiance and Irradiance Caching

- [Slide courtesy of Krivanek 2006]

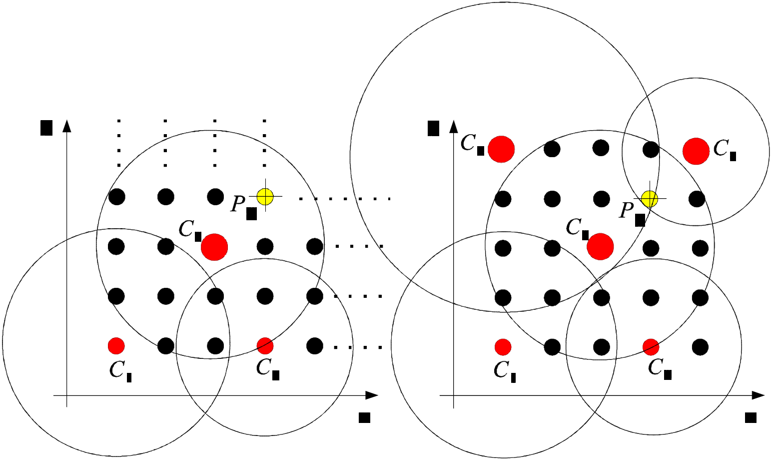


Irradiance Cache Search

- Records – the irradiance specified by spheres (point and radius of influence)
- Query: given a point, find all the sphere in which the point is contained
- Problem is *intersection search*
- Data structures should be dynamic – insertion and deletion is required

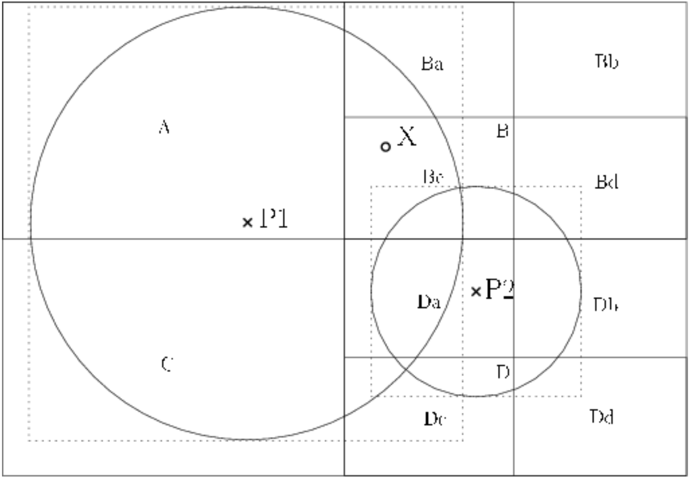
Irradiance Cache Searching

- Data: **spheres** C_1, C_2, C_3 ..... Queries: **points** P_Q
- Ouput of search: set of spheres containing P_Q



Irradiance Cache with Octrees

- [Ward et al. 88]



Data Structures for Irradiance Cache

- Intersection search, spheres are sorted. The search for spheres containing a query point
- Originally octree [Ward 88]
- Survey of other possibilities in the thesis: *Data Structures for Interpolation of Illumination with Radiance and Irradiance Caching*, Karlik [2011]
  - Wards's octree
  - Multiple reference octree
  - Kd-tree
  - Multiple reference kd-tree
  - Bounding volume hierarchies
  - Dual space kd-tree in R^4 (transformation method)

Comparison of Different Data Structures for Irradiance Caching

- Experimental comparison: data structure nodes, nodes visited per query, records visited per query, data structure build time, performance [samples/s]
- Summary:
  - The point kd-tree has low build time and very good search time
  - From the data structures referencing only once each record, BVH is the best (2 to 4 times better than Wards' octree), perhaps the most practical.
  - The fastest search is for multiple reference kd-tree, but its build time is also the highest one
  - The solution via transformation method to R^4 is slow
- The details in the Master Thesis [Karlik 2011]together with data structures for (directional) radiance caching

EUROGRAPHICS 2014 TUTORIAL

SURFACE REFLECTANCE REPRESENTATIONS (BRDF, BTF, ...)

VLASTIMIL HAVRAN

Czech Technical University in Prague

TUTORIAL OVERVIEW

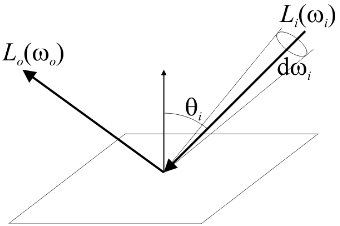
Introduction (5 min)	VH
Sorting and Searching Techniques (30 min)	JB
Hierarchical Data Structures (30 min)	JB/VH
Ray Tracing (20 min)	VH
Q & A (5 min)	
Coffee break	
Rasterization and Culling (25 min)	JB
Photon Maps and Ray Maps (20 min)	VH
Irradiance Caching (5 min)	VH
BRDF and BTF (10 min)	VH
Sorting and searching on GPU (20 min)	JB
Q & A (10 min)	

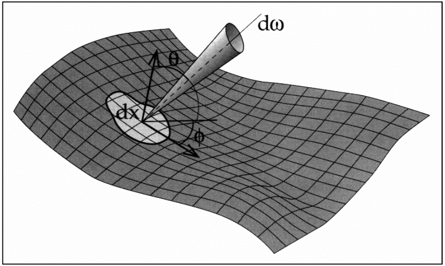
BRDF – Bidirectional Reflectance Distribution Function

- BRDF definition

$$f_r(\omega_i, \mathbf{x}, \omega_o) = \frac{dL_o(\mathbf{x}, \omega_o)}{dE(\mathbf{x}, \omega_i)} = \frac{dL_o(\mathbf{x}, \omega_o)}{L_i(\mathbf{x}, \omega_i)\cos\theta_i d\omega_i}$$

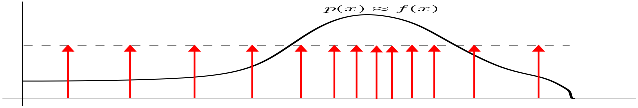
- Unit: [1/sr]





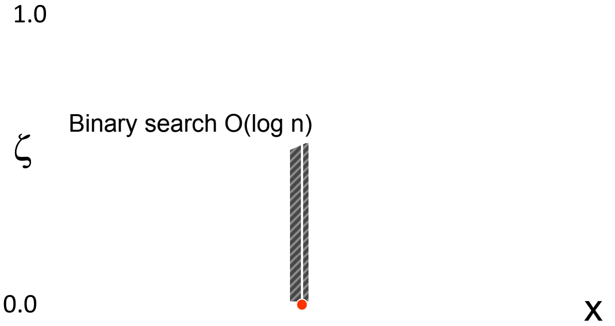
BRDF Overview

- Several possibilities to represent BRDF
  - Analytical models
  - Data tabulated measured models
  - Compression algorithms
- For rendering another operation needed: importance sampling of BRDF.cos(θ)



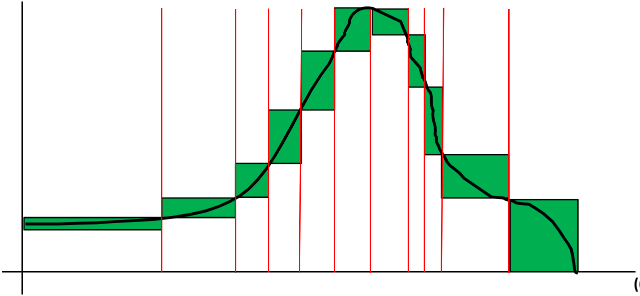
BRDF Importance Sampling [Lawrence et al. 2005]

- Tabulated data – computing importance sampling from BRDF.cos(θ)
  - Set of 1D marginal PDFs in form of cumulative distribution functions (CDF)
  - Compress 1D CDF functions with Douglas-Peucker curve approximation
  - Select first which 1D function using **binary search**
  - Then by **binary search** over 1D function select direction



Generic BRDF Sampling [Montes et al. 2008]

- Rejection sampling is in general slow
  - It pays of to build auxiliary data structures to speed up importance sampling
  - The rejection sampling is limited to regions using quadtrees working over 2D slices of BRDF.cos(θ) and the maximum is stored for each cell of quadtree.
  - Precompute and store for each 2D cell its mean and maximum values, subdivide until these two do not differ much!



BTF Datasets [Dana et al. 1999]

- Extension of BRDF concept by two dimensions for position in space, so 7D function

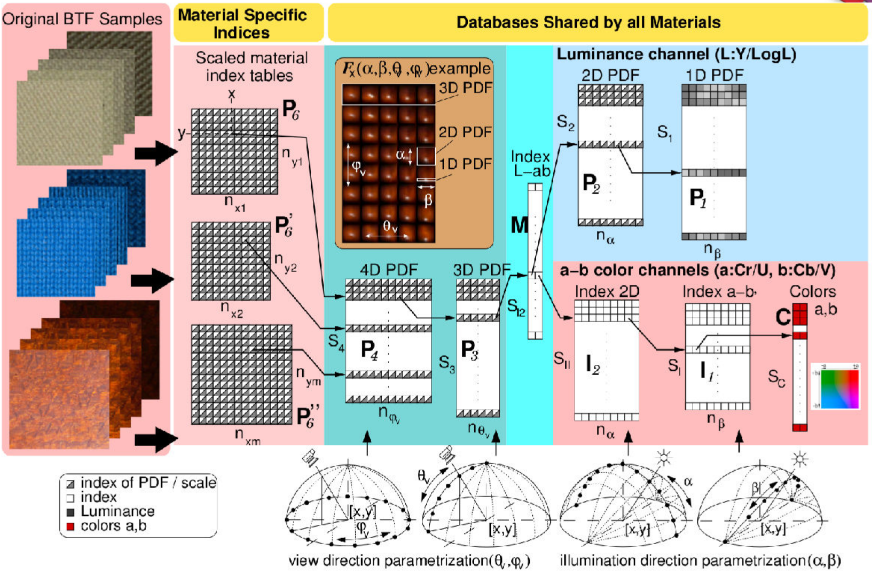
Courtesy of RealReflect project

BTF captures visual richness, anisotropy, visual masking and self-shadowing

BTF Compression - Data Based Driven Approach [Havran et al. 2010]

- BTFBASE - set of codebooks, database like approach, motivated by searching of similar texels in BTF datasets
- For importance sampling we need twofold binary search, cumulative distribution function is computed on the fly for small 1D functions
- For compression the **main operation is only search** – for single threaded application the compression time is from 8 to 20 hours for year 2009.
- Insertion one by one, implements vector quantization
- The multi-level decomposition gives even higher compression ratio

BTF Compression Scheme Overview



BTF Compression Summary

- Algorithm idea motivated by paradigm: rendering is sorting and searching
- Compression ratio from 1:233 to 1:2267 , average 1:764 for most compact representation
- Decompression fast both on CPU and GPU, bottleneck is the number of accesses into the memory, but still 1.5 times faster than [Lafortune 97] model with one lobe
- Fast importance sampling: 310,000 to 1,360,000 evaluations per second on a single core CPU
- Multi-level quantization improves compression ratio 10 times, more details in the paper, demo on the web

EUROGRAPHICS 2014 TUTORIAL

GPU SORTING & SEARCHING FOR RENDERING

JIŘÍ BITTNER

Czech Technical University in Prague

TUTORIAL OVERVIEW

Introduction (5 min)	VH
Sorting and Searching Techniques (30 min)	JB
Hierarchical Data Structures (30 min)	JB/VH
Ray Tracing (20 min)	VH
Q & A (5 min)	
Coffee break	
Rasterization and Culling (25 min)	JB
Photon Maps and Ray Maps (20 min)	VH
Irradiance Caching (5 min)	VH
BRDF and BTF (10 min)	VH
Sorting and searching on GPU (20 min)	JB
Q & A (10 min)	

CPU versus GPU

- CPU
  - Small number of independent cores per CPU (12 cores Xeon E5-2695 V2)
  - Cache based architecture - efficient cache hierarchy (L1, L2, L3)
- GPU
  - Thousands of #cores (2880 cores in GeForce GTX Titan Black), SIMT computation organization (warps)
  - Stream based architecture
  - Limited cache / local memory, high dependency on number of registers
  - Large number of threads, hiding memory latency, minimizing synchronization, on-chip shared buffers

	cores	transistors [M]	L1 kB	L2 kB	L3 MB	TFLOPS
CPU	2-12	4000	384	3072	30	1
GPU	2880	7080	240-720	1536	-	5

NVIDIA GPU Architecture Overview

- Kepler (GK 110)
- 7.1 billion transistors
- Dynamic parallelism
- 3x more efficient than Fermi (same power)
- Hyper-Q sharing one GPU among more CPUs
- Maximum 255 registers per thread, configurable shared memory/cache size etc.

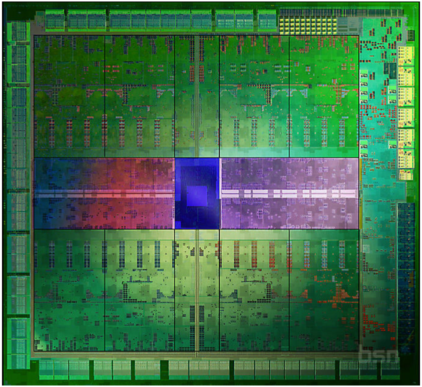
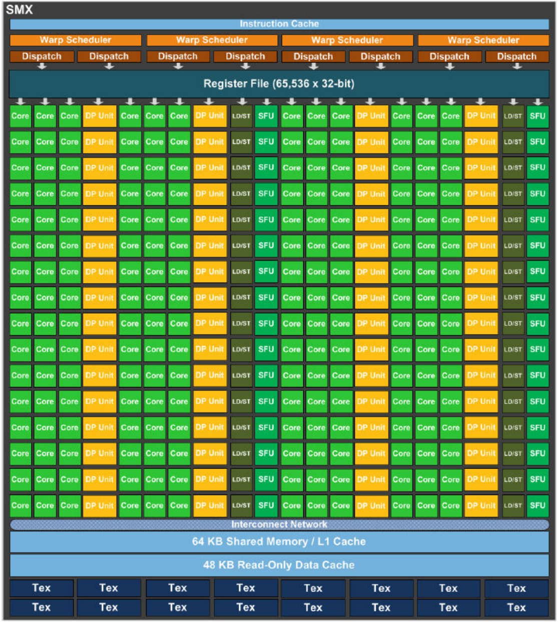


Image courtesy of NVIDIA

NVIDIA Kepler Architecture (GK110)

NVIDIA Kepler SMX



NVIDIA Kepler / Maxwell Comparison

GPU	GK107 (Kepler)	GM107 (Maxwell)
CUDA Cores	384	640
Base Clock	1058 MHz	1020 MHz
GPU Boost Clock	N/A	1085 MHz
GFLOPs	812.5	1305.6
Texture Units	32	40
Texel fill-rate	33.9 Gigatexels/sec	40.8 Gigatexels/sec
Memory Clock	5000 MHz	5400 MHz
Memory Bandwidth	80 GB/sec	86.4 GB/sec
ROPs	16	16
L2 Cache Size	256KB	2048KB
TDP	64W	60W
Transistors	1.3 Billion	1.87 Billion
Die Size	118 mm^2	148 mm^2
Manufacturing Process	28-nm	28-nm

GPU Programming Models

- Fixed OpenGL
  - Difficult mapping of algorithms and to data structures to GPU
  - [Purcell et al. 2002, Ray Tracing on Programmable Graphics Hardware]
- Programmable OpenGL
  - GLSL, tesseletation, vertex, geometry, fragment shaders
  - Compute shaders
- CUDA
  - GPU programming language
  - Flexible + controllable, performance oriented
- OpenCL
  - General parallel oriented programming language (also for CPUs)

Sorting & Searching on the GPU

- Searching parallelization only
  - Build data structure on the CPU, transfer it to the memory
  - GPU is used for **searching only** by many parallel threads
  - Relatively easy implementation
  - Used for NVIDIA OptiX and Karras/Aila framework for ray tracing
- Full parallelization (both build and search)
  - Papers already from 2006 - still relatively cumbersome to implement
  - Common use of parallel prefix sum (general reductions), gather, scatter
  - Dynamic allocation needed for some data structures (kd-tree)
  - Hot research topic - potential for future applications

Searching on the GPU

- Independent queries
  - Large number of threads
  - Minimizing synchronization
  - On-chip shared buffers
- Using uniform grid
  - Efficient parallelization (regular, predictable)
  - Lower algorithmic efficiency for irregular data

Searching on the GPU

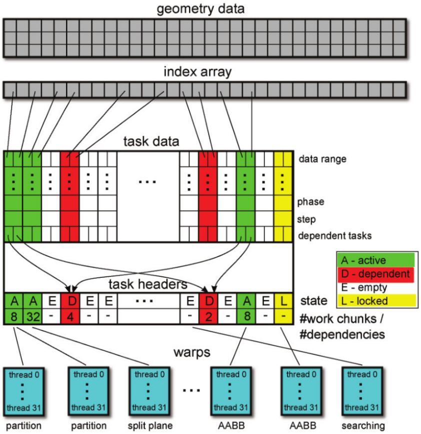
- Using hierarchies
  - Parallel traversal without stack: restart, neighbor links
  - Using stack of limited size
  - Stack spilling
- Scheduling traversal
  - Minimizing memory bandwidth and latency
  - Maximizing coherency of traversal (interior nodes vs leaves)
  - Postpone leaf processing [Aila&Laine]

Sorting on the GPU

- Sorting methods
  - Radix sort / distribution sort [Merrill2010]
  - Merge sort
- Mapping higher dimensional problems to 1D sort
  - Morton codes (LBVH, HLBVH)
- Hashing
  - Linear / quadratic probing
  - Cuckoo hashing [Alcantara09]
- Spatial hierarchies
  - Low amount of “vertical” parallelism near root node [Karras12]
  - Horizontal parallelization: parallel node subdivision step

General Parallelization Framework

- Task pool using persistent threads [Vinkler2013]
  - Organizing unsolved tasks
  - Searching for available work
  - Seamlessly merging vertical / horizontal parallelization



- Dynamic parallelism (NVIDIA Kepler)
  - Ability to spawn new work in the kernel, since year 2013

GPU’s Weak Points for Hierarchies

- Difficult horizontal parallelization
  - Needed for top-level parts of hierarchical data structures
- Inefficient implementation of dynamic allocation
  - As of 2013/2014
  - Difficult to implement kd-trees SBVHs, low performance
- Performance limitations
  - Number of registers and small local cache
- Much more difficult to implement than on a CPU

New Trends – GPUs for Mobile Devices

Two options for mobile devices

- **Light weight programmable GPUs** for mobile phones
  - Mostly useful for rasterization
- **Special hardware** proposed for traversal in ray tracing
  - SIGGRAPH ASIA 2013 [Real Time Ray Tracing on Future Mobile Computing Platform] uses BVHs (*SGRT – Samsung Reconfigurable GPU based on Ray Tracing*)
- Hardware proposal for GPU building
  - SIGGRAPH 2013 [Doyle: A Hardware Unit for Fast SAH-optimized BVH Construction]
- BVHs most perspective for ray tracing in mobile devices?

Summary

- Practice – build on a CPU, traverse on a GPU
- Uniform grids
  - Option for some problems with regular distribution of data
- Building data structures on GPUs is still difficult
  - General sorting algorithm do not suffice
- Open research problems
  - Efficient parallelization of building hierarchical data structures
  - Real time performance needed
  - Advanced global illumination algorithms fully running on GPUs
- With newer GPU architectures things will change
  - More flexibility
  - Rays/W on mobile devices

EUROGRAPHICS 2014 TUTORIAL

EFFICIENT SORTING AND SEARCHING IN RENDERING ALGORITHMS

VLASTIMIL HAVRAN & JIŘÍ BITTNER

Czech Technical University in Prague

DCGI

What Is Left Unpresented in Our Tutorial

- Beam tracing approaches for photon ray splatting [Herzog et al. 2007] with kd-trees, stochastic progressive photon mapping [2009], progressive photon mapping [2008],
- Vertex connection and merging – [Georgiev et al. 2012], [Hachisuka et al. 2012] and similar [Bekaert et al. 2003]
- Importance sampling of environment maps – various papers such as Q2-tree [Wan et al. 2005] and [Havran et al. 2005]
- Light-cuts and light hierarchies [Walter et al. 1998], [Paquette et al. 1998] etc.
- Product importance sampling with quadtrees [Clarberg 2006], [Rouselle 2008], [Clarberg and Akenine-Moller 2008]
- Numerous uses of simple binary search in rendering algorithms and algorithms we have overlooked or not had time for!

Tutorial Conclusion

- Sorting and searching is a must in rendering algorithms even of fast advances of hardware
- The selection of right algorithm and data structure is hardware dependent, getting the best algorithm is difficult, no worst case guarantees
- In general hierarchies work well, for special cases uniform grids, hashing etc.
- Implementation on GPUs and other special architectures possible but cumbersome
- Future trend might be to put searching and also sorting to hardware for fully specified algorithms keeping programmability by shaders etc.

Acknowledgements

- Robert Herzog, Michael Wimmer, Peter Wonka, Tommer Leyvand, David Luebke, and Hansong Zhang and other colleagues for providing us various materials used in the tutorial
- This tutorial was partially supported by Czech Science Foundation agency under the research program number P202/12/2413 (OPALIS)

Thank you!

TUTORIAL OVERVIEW

Introduction (5 min)	VH
Sorting and Searching Techniques (30 min)	JB
Hierarchical Data Structures (30 min)	JB/VH
Ray Tracing (20 min)	VH
Q & A (5 min)	
Coffee break	
Rasterization and Culling (25 min)	JB
Photon Maps and Ray Maps (20 min)	VH
Irradiance Caching (5 min)	VH
BRDF and BTF (10 min)	VH
Sorting and searching on GPU (20 min)	JB
Q & A (10 min)	