Acceleration Structures for Real-Time Ray Tracing on current Hardware

Projekt SS 2010

VR Systems Group

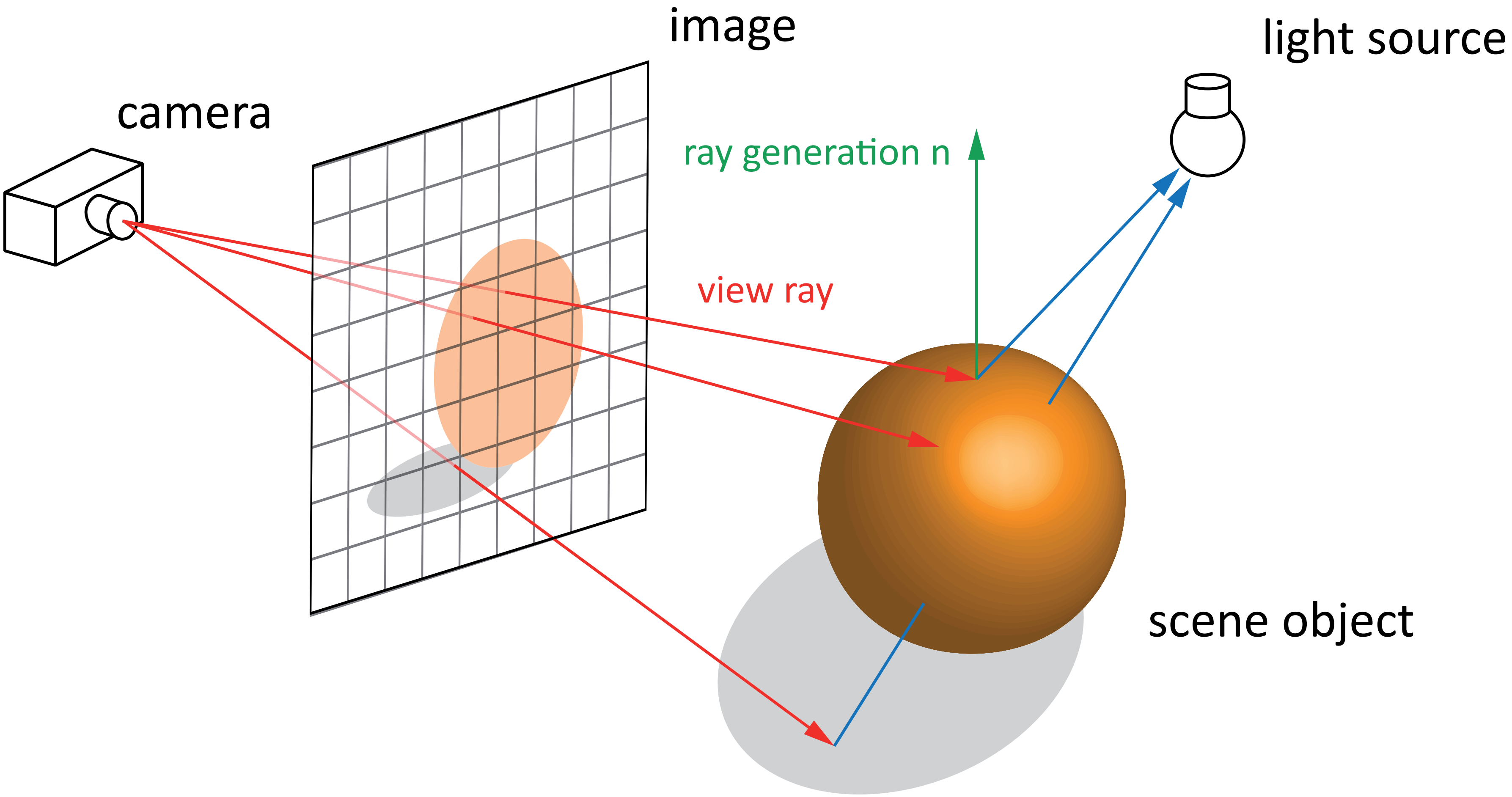
Jan Frederick Eick, Sascha Gärtner, Henning Gründl, Sebastian Thiele

Stephan Beck

Bauhaus-Universität Weimar

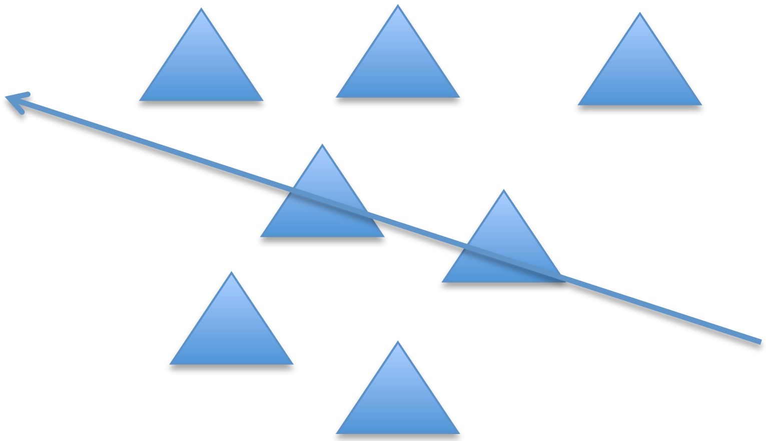
Raytracing

Introduction



Raytracing

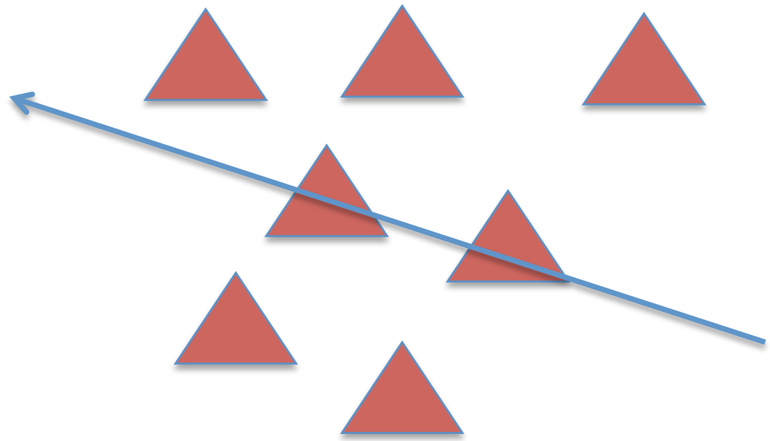
Acceleration Structures



- intersect ray with geometry

Raytracing

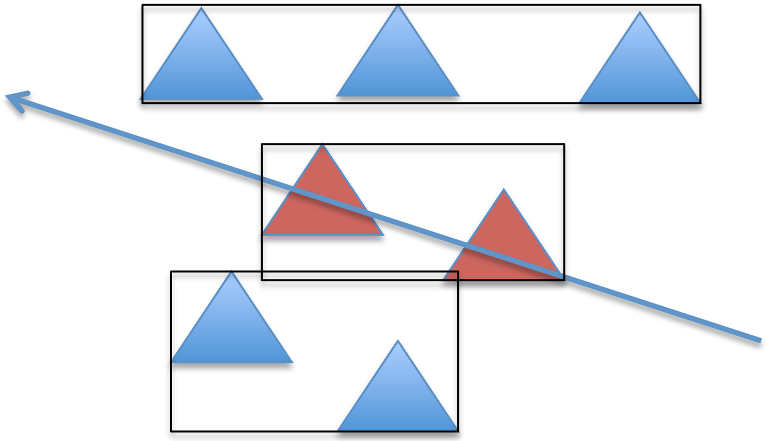
Acceleration Structures



- intersect ray with geometry
- brute force intersects ray with each triangle

Raytracing

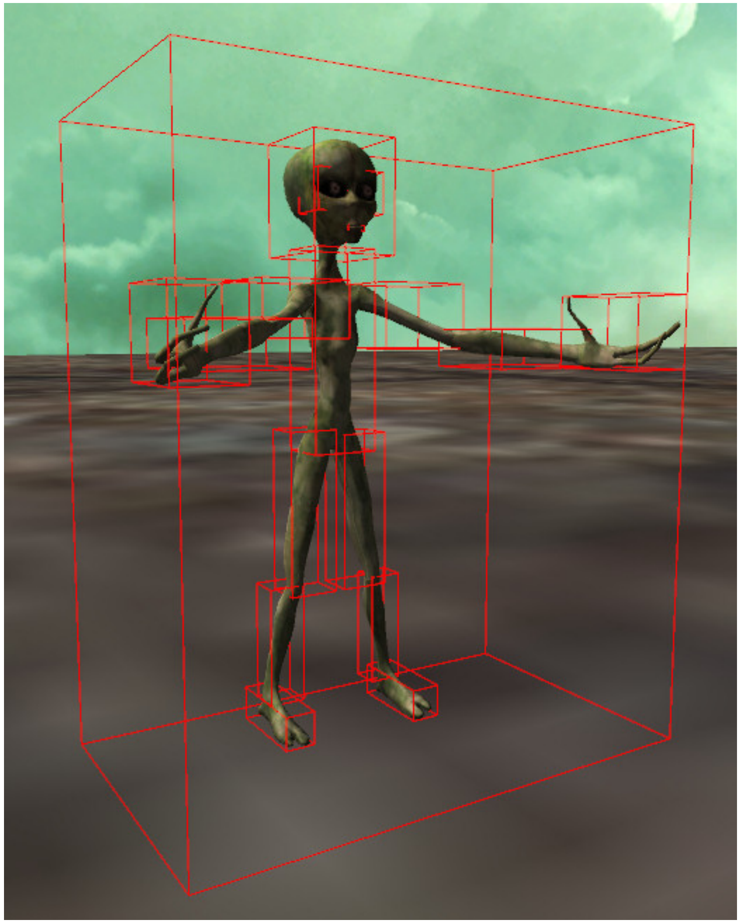
Acceleration Structures



- intersect ray with geometry
- brute force intersects ray with each triangle
- avoid unnecessary triangle intersections with **Acceleration Structures**

Acceleration Structures

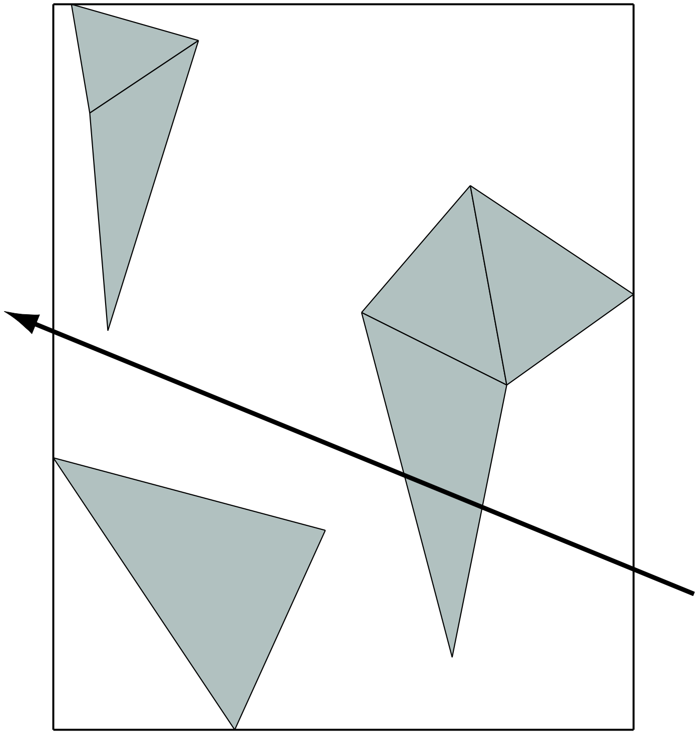
Bounding Volume Hierarchy (BVH)



- geometric objects are wrapped in **bounding volumes**
- bounding volumes are **axis aligned**
- root node keeps the whole geometry

Acceleration Structures

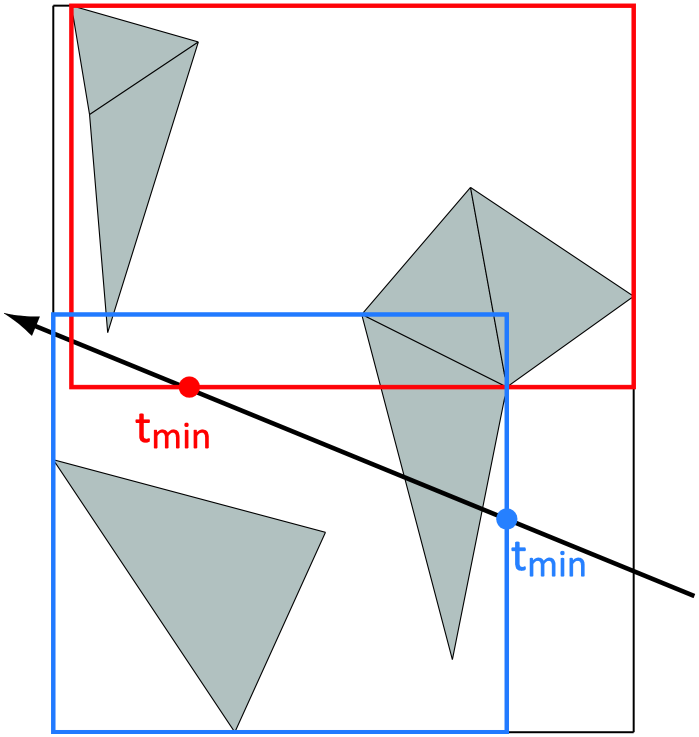
Bounding Volume Hierarchy



- **intersection** with bounding boxes

Acceleration Structures

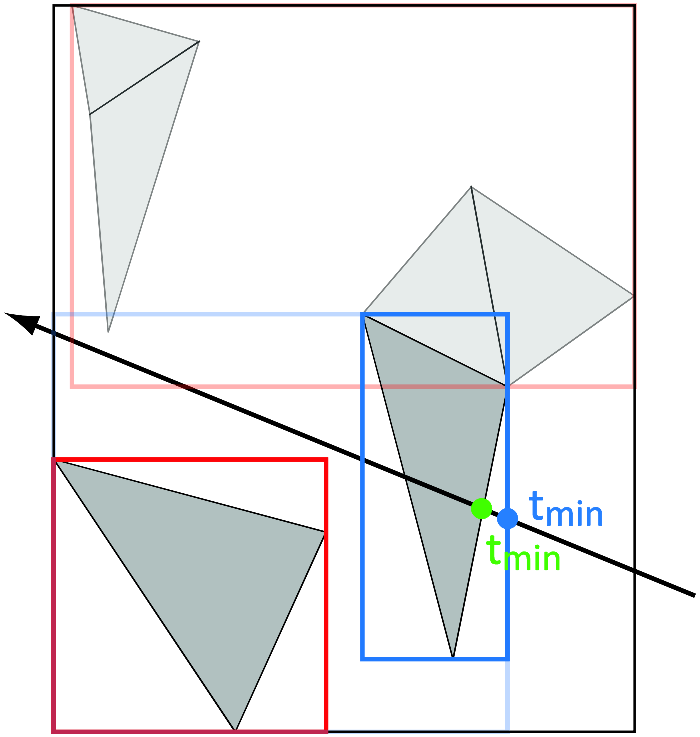
Bounding Volume Hierarchy



- **intersection** with bounding boxes
- handle node near the viewer first

Acceleration Structures

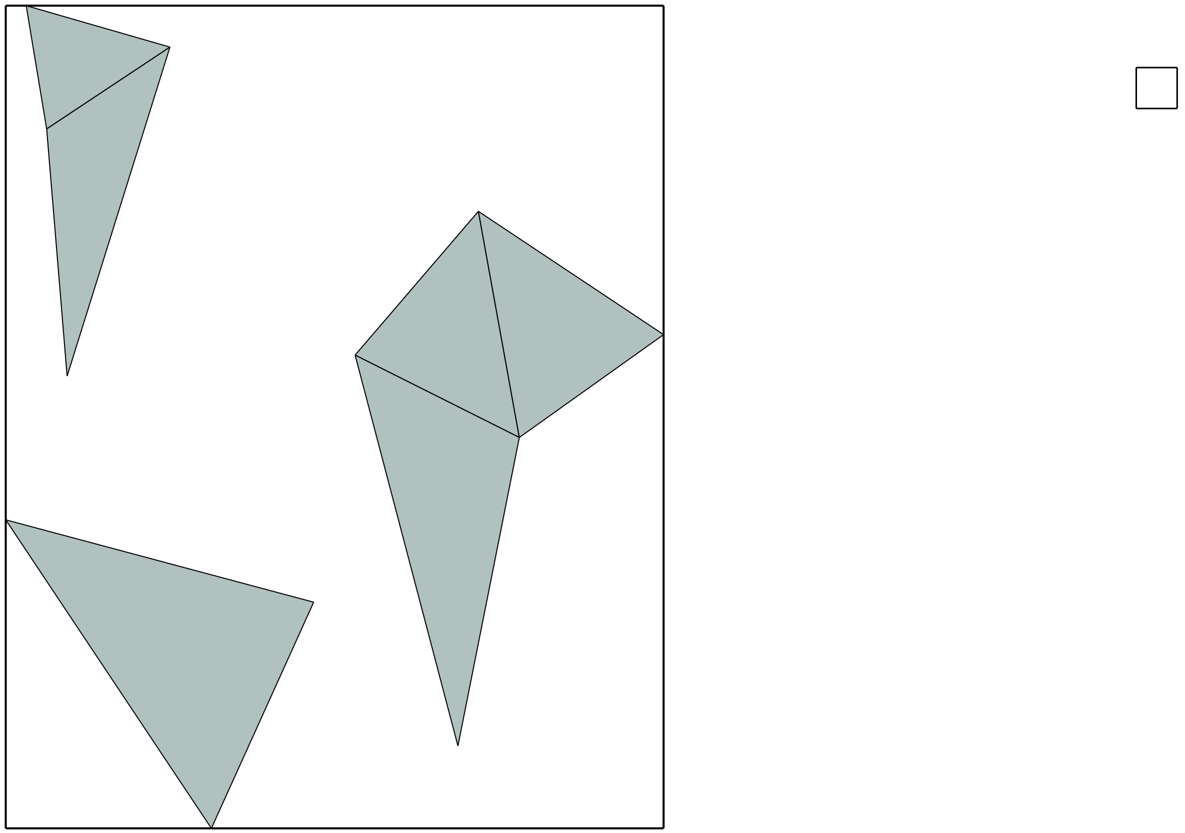
Bounding Volume Hierarchy



- **intersection** with bounding boxes
- handle node near the viewer first
- possible to stop if intersection is found (**early ray termination**)

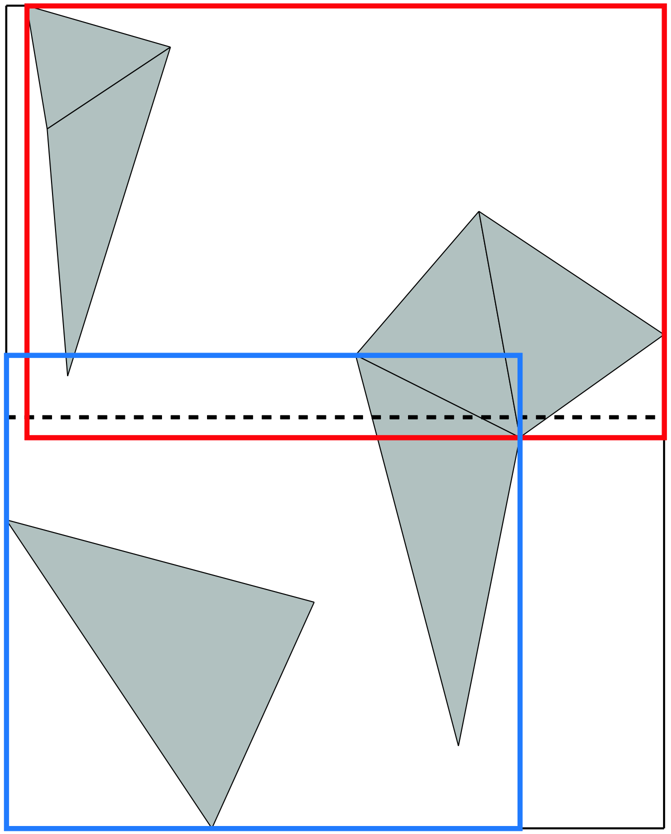
Acceleration Structures

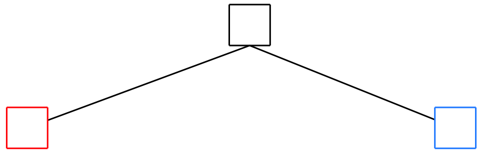
Bounding Volume Hierarchy



Acceleration Structures

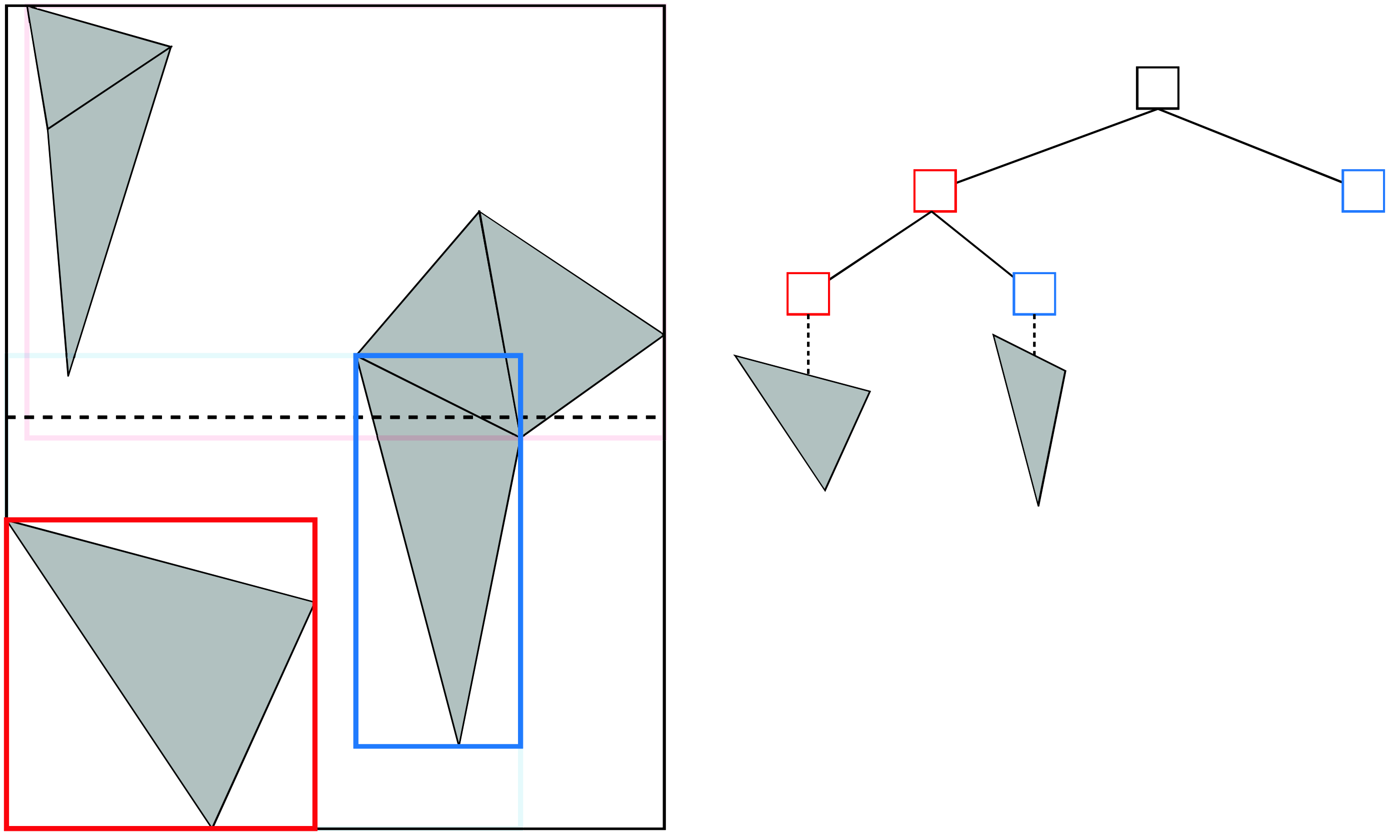
Bounding Volume Hierarchy





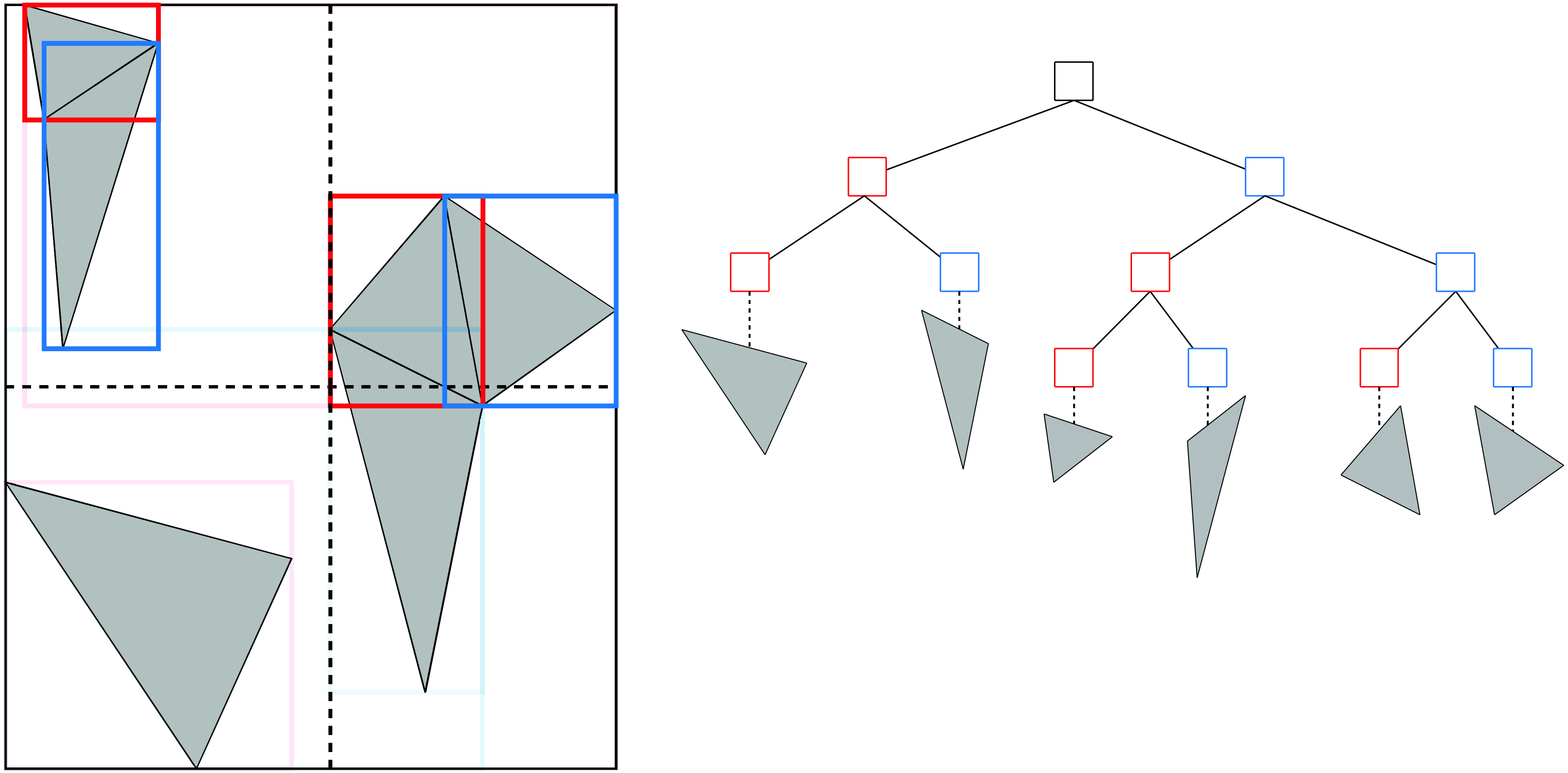
Acceleration Structures

Bounding Volume Hierarchy



Acceleration Structures

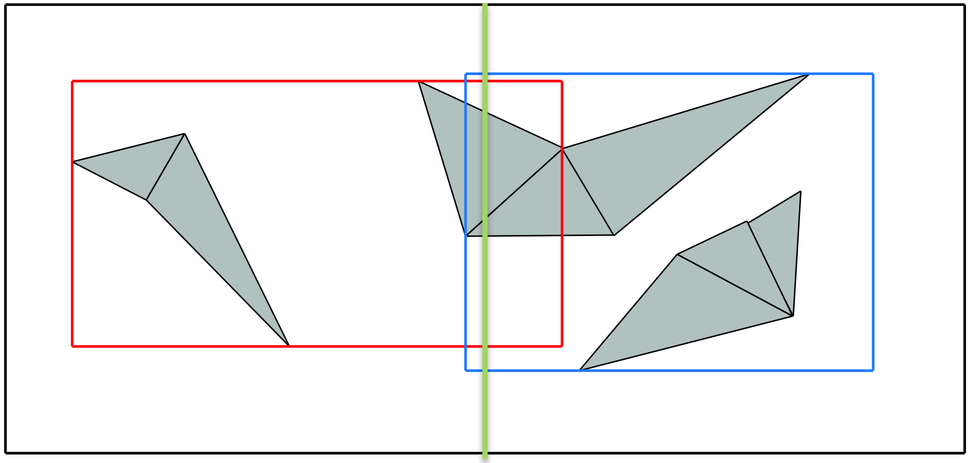
Bounding Volume Hierarchy



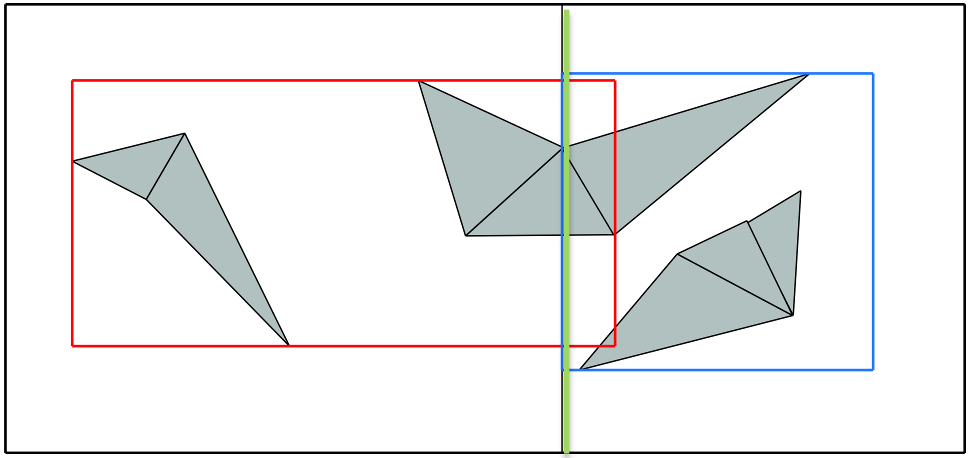
Acceleration Structures

Splittingplane

split in the middle

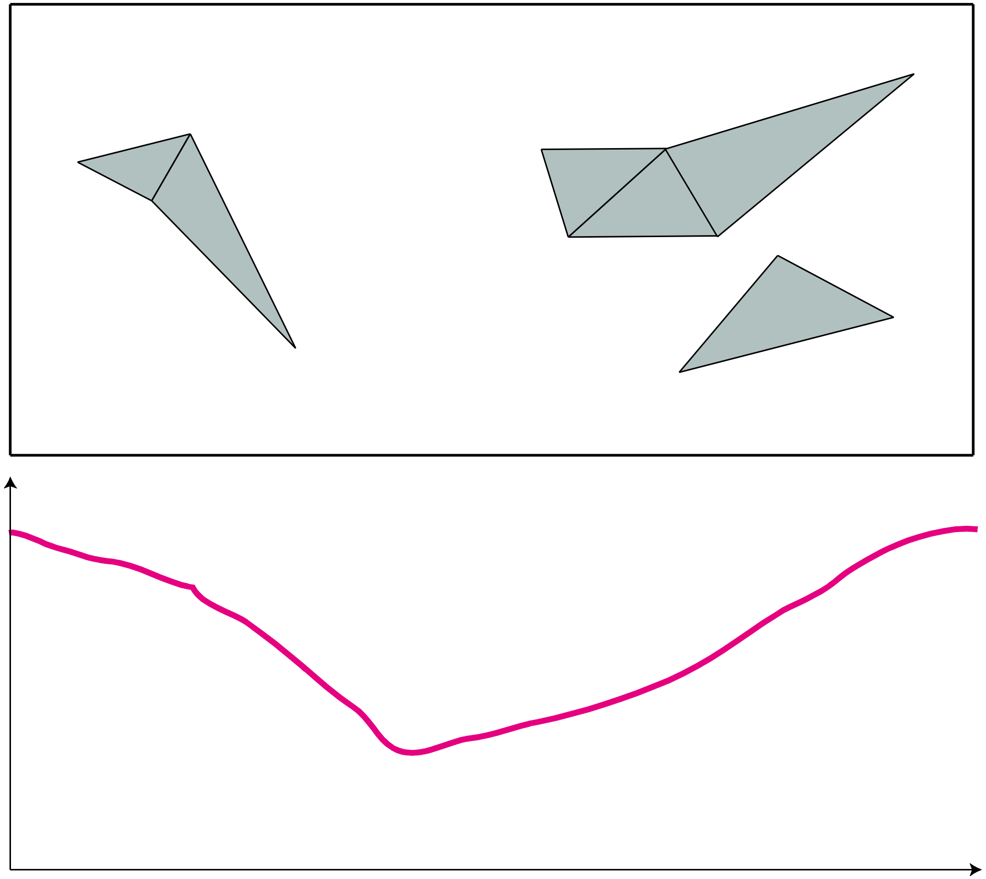


median cut



Acceleration Structures

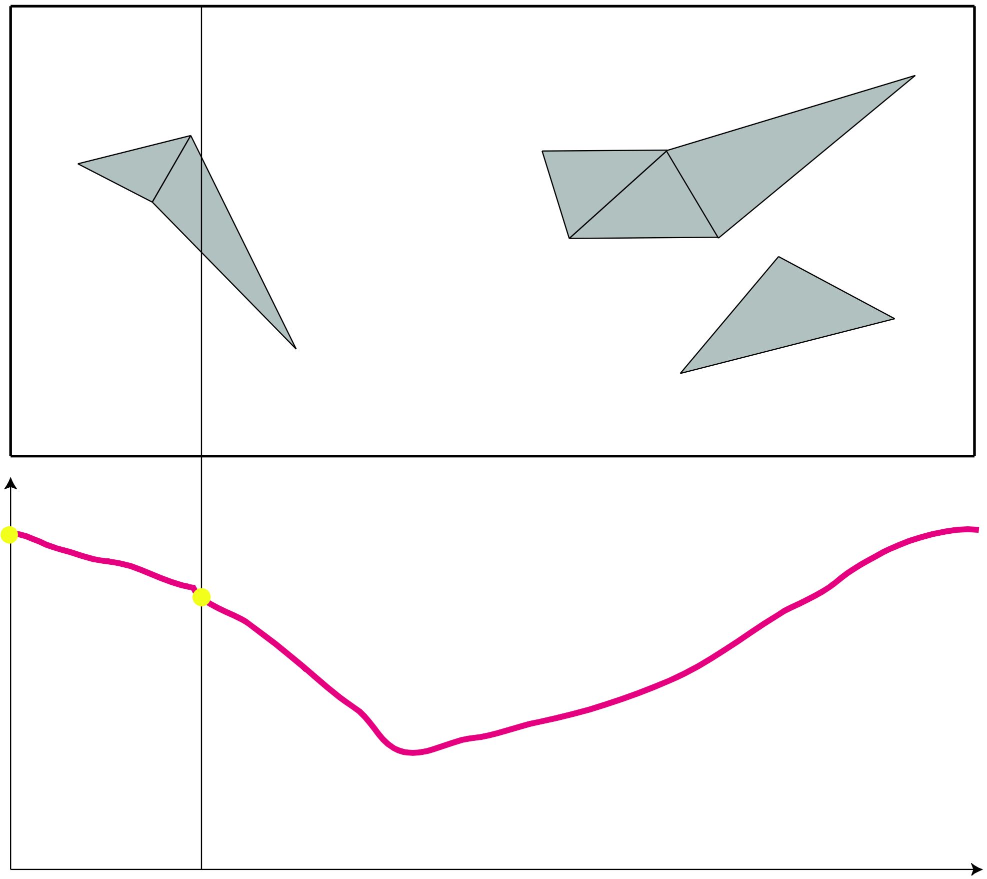
Surface Area Heuristic (SAH)



- each node in the BVH should be of **minimum volume**

Acceleration Structures

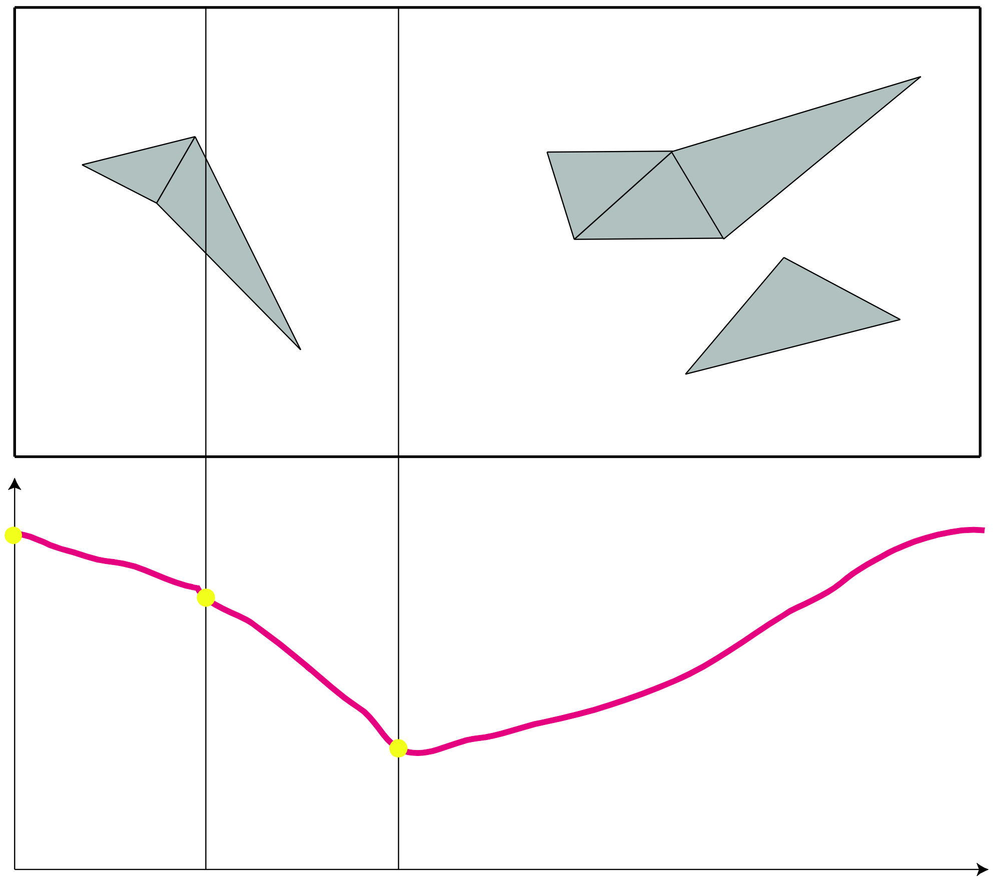
Surface Area Heuristic



- each node in the BVH should be of **minimum volume**
- SAH gives the cost for n candidate planes (**bins**)

Acceleration Structures

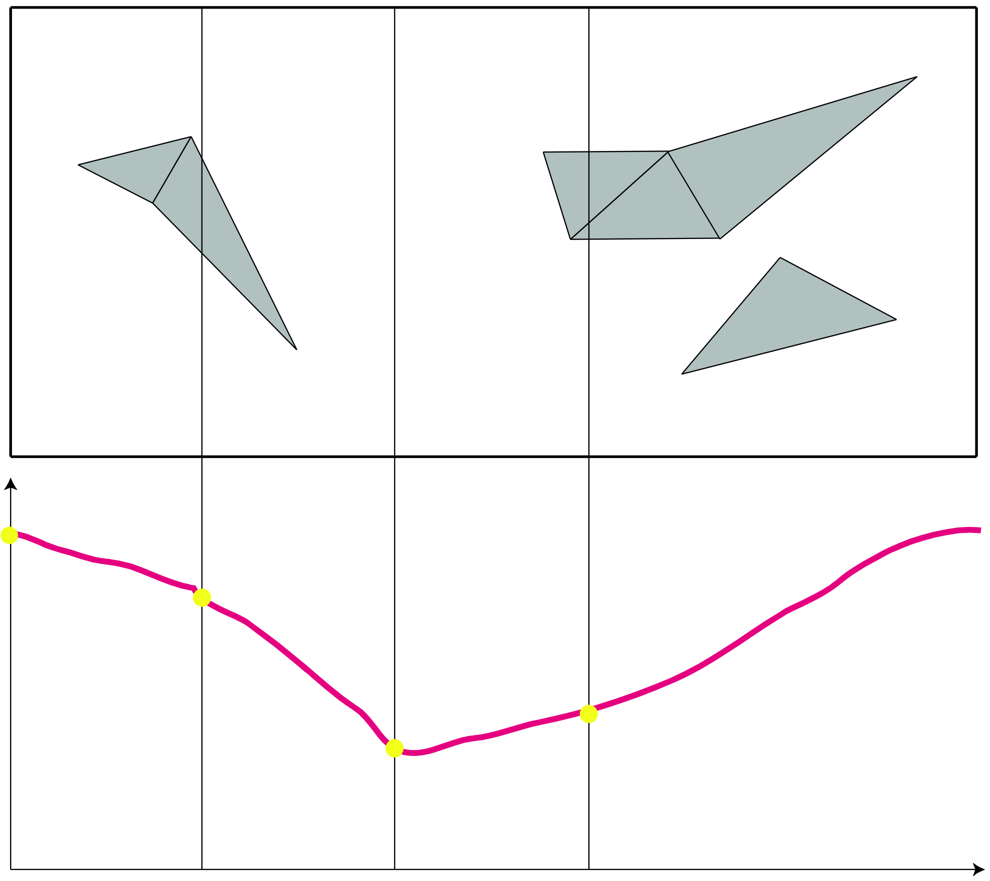
Surface Area Heuristic



- each node in the BVH should be of **minimum volume**
- SAH gives the cost for n candidate planes (**bins**)

Acceleration Structures

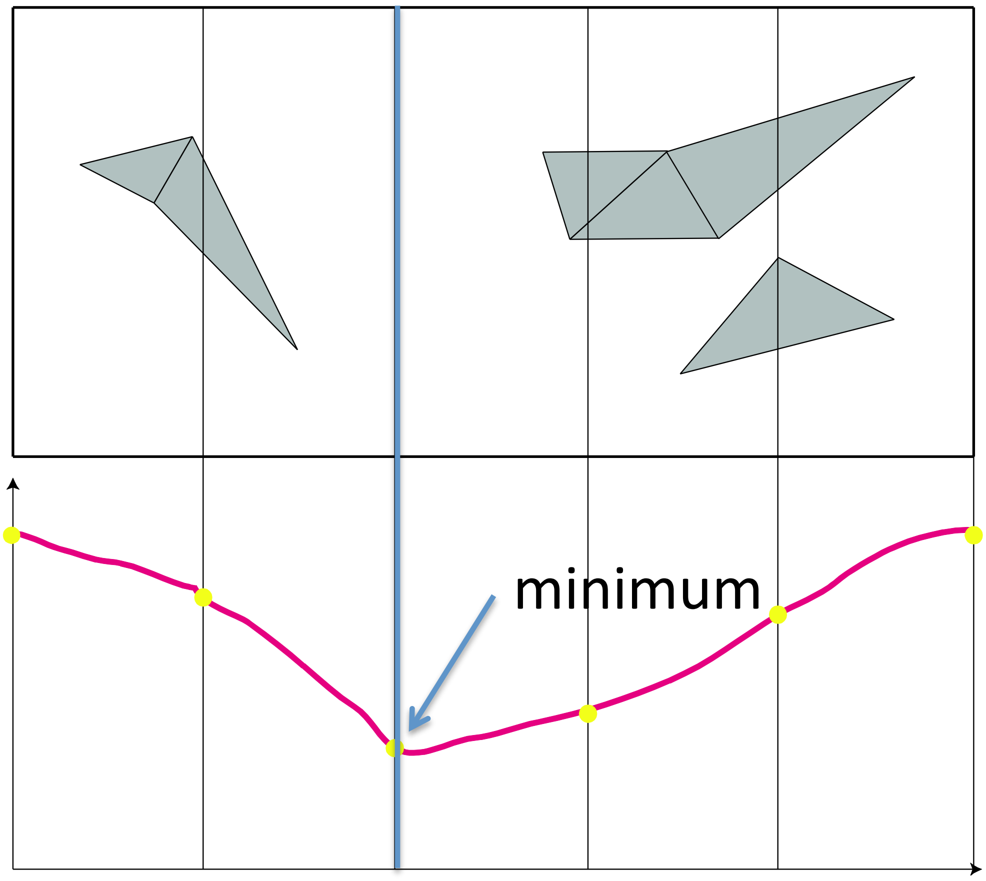
Surface Area Heuristic



- each node in the BVH should be of **minimum volume**
- SAH gives the cost for n candidate planes (**bins**)

Acceleration Structures

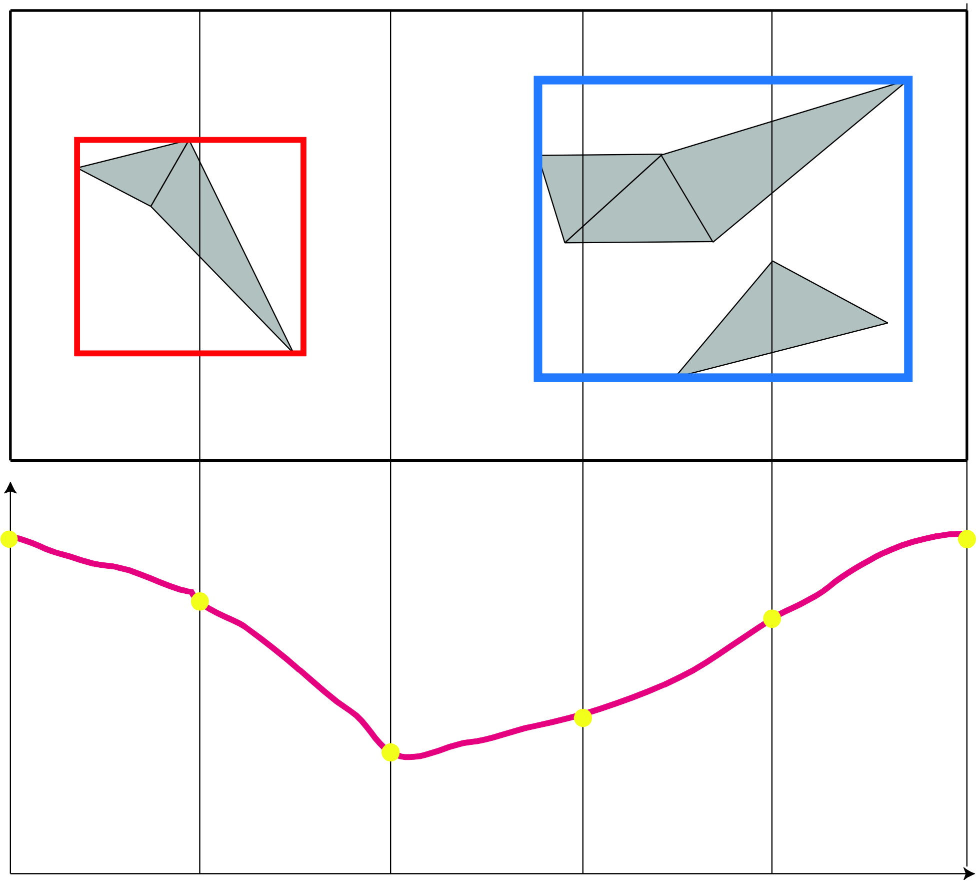
Surface Area Heuristic



- each node in the BVH should be of **minimum volume**
- SAH gives the cost for n candidate planes (**bins**)
- global minima is used as splitting plane

Acceleration Structures

Surface Area Heuristic



- each node in the BVH should be of **minimum volume**
- SAH gives the cost for n candidate planes (**bins**)
- global minima is used as splitting plane

Acceleration Structures

Surface Area Heuristic

$$C(B) = \text{Area}(B_1) \cdot N(B_1) + \text{Area}(B_2) \cdot N(B_2)$$

C(B)	cost for candidate plane
Area(B)	area of BV including objects in bin
N(B)	number of objects in bin

- each node in the BVH should be of **minimum volume**
- SAH gives the cost for n candidate planes (**bins**)
- global minima is used as splitting plane

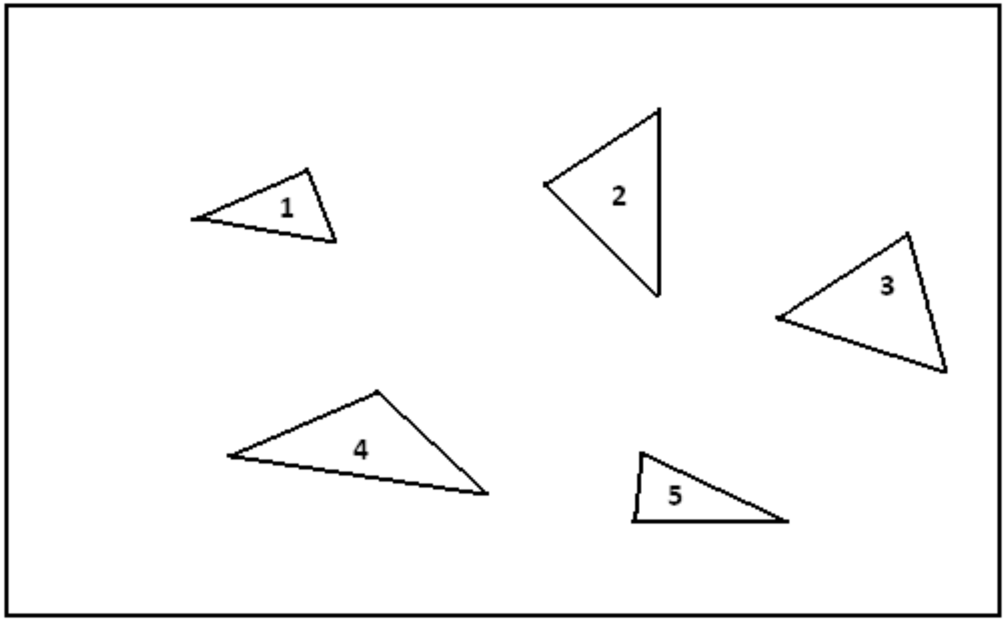
Acceleration Structures

KD-Tree

- binary tree
- devides k-dimensional space recursively through splitting planes
- interior nodes represent planes
- leafes store references to triangles

KD-Tree

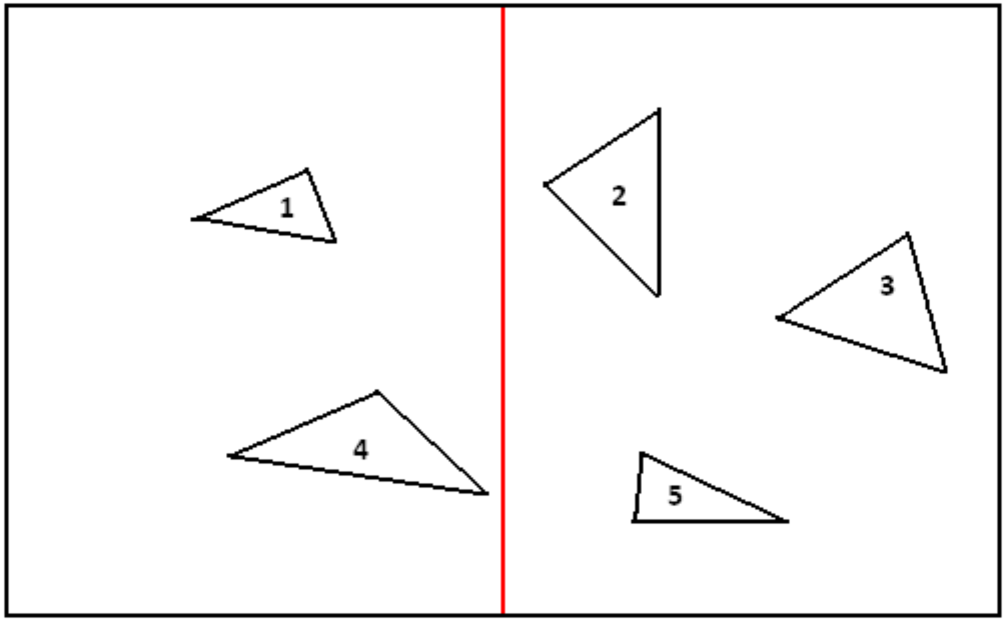
naive „spatial-median“ algorithm

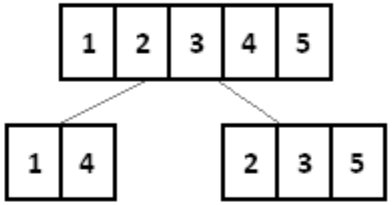


1	2	3	4	5

KD-Tree

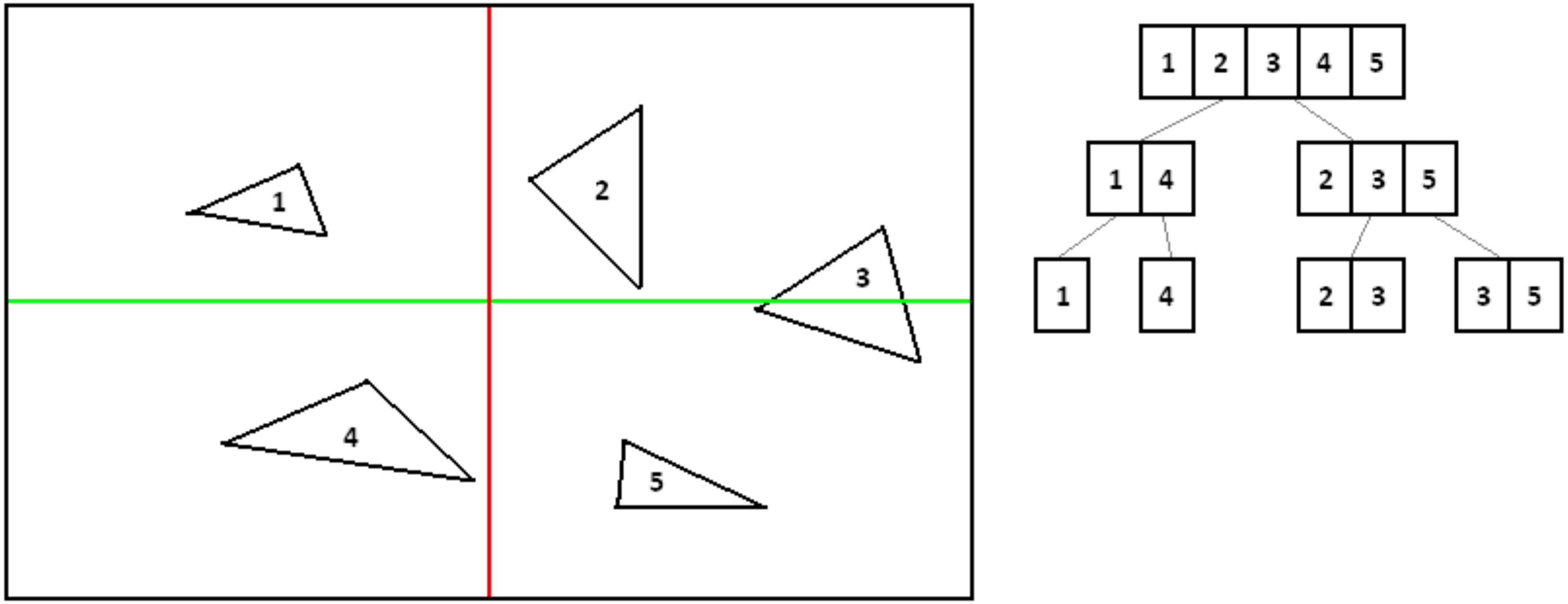
naive „spatial-median“ algorithm





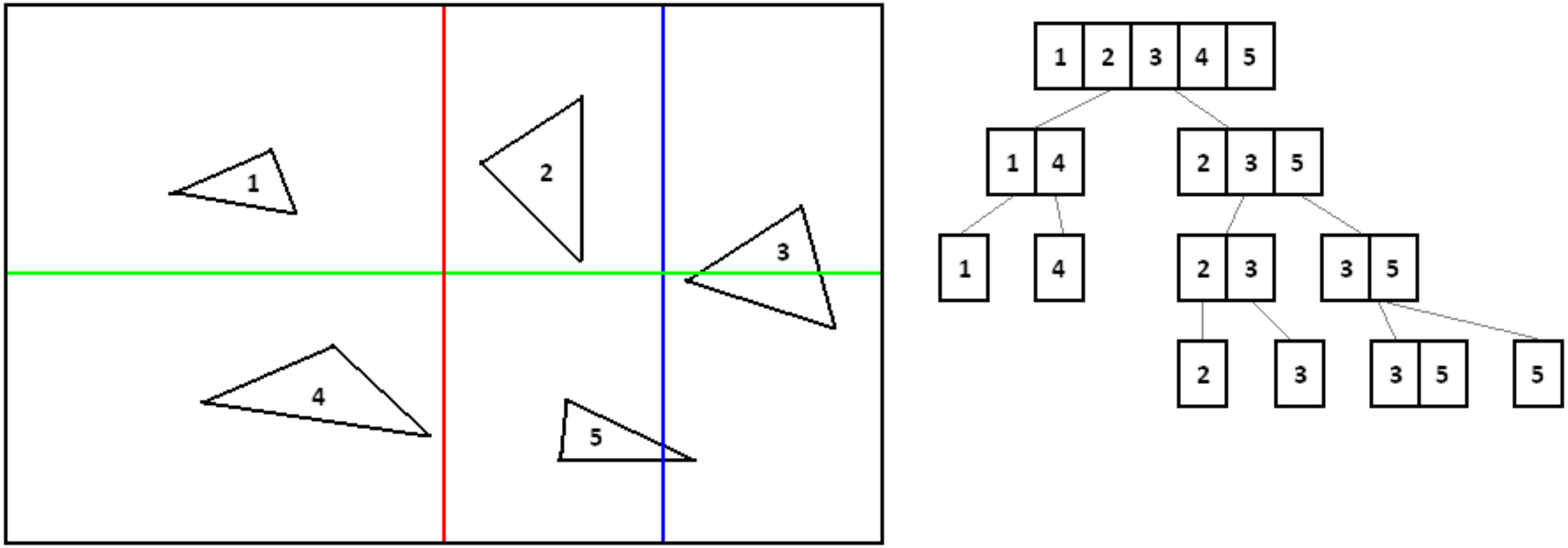
KD-Tree

naive „spatial-median“ algorithm



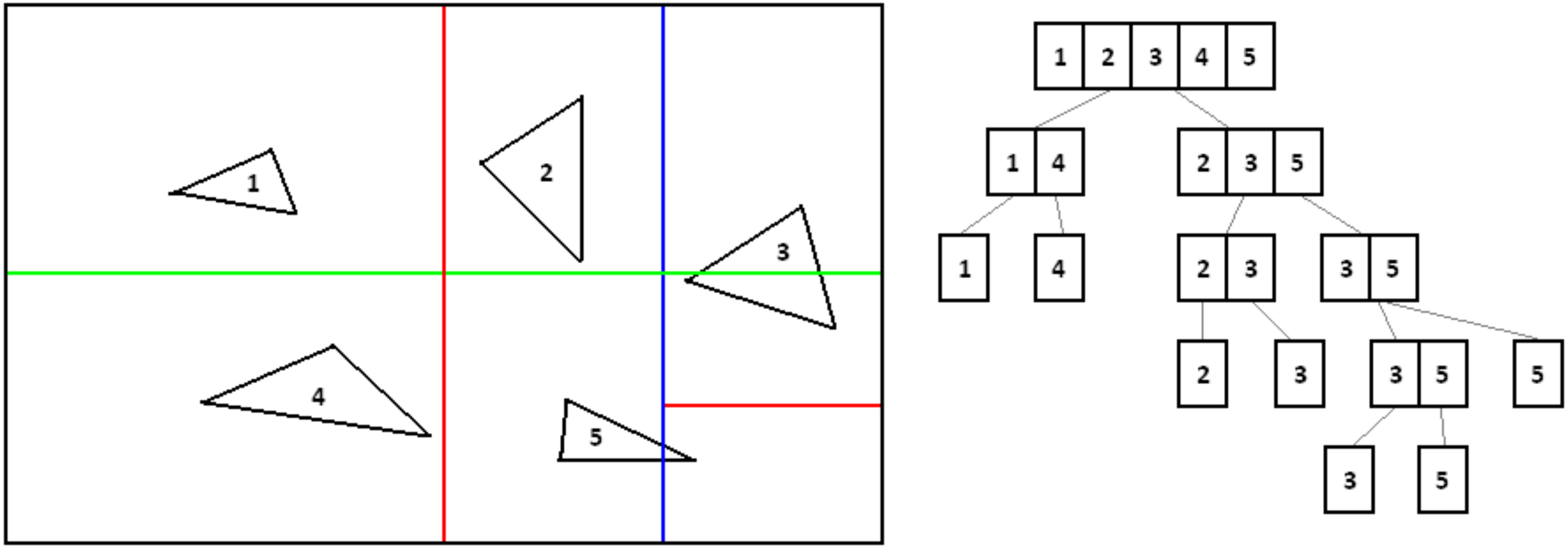
KD-Tree

naive „spatial-median“ algorithm



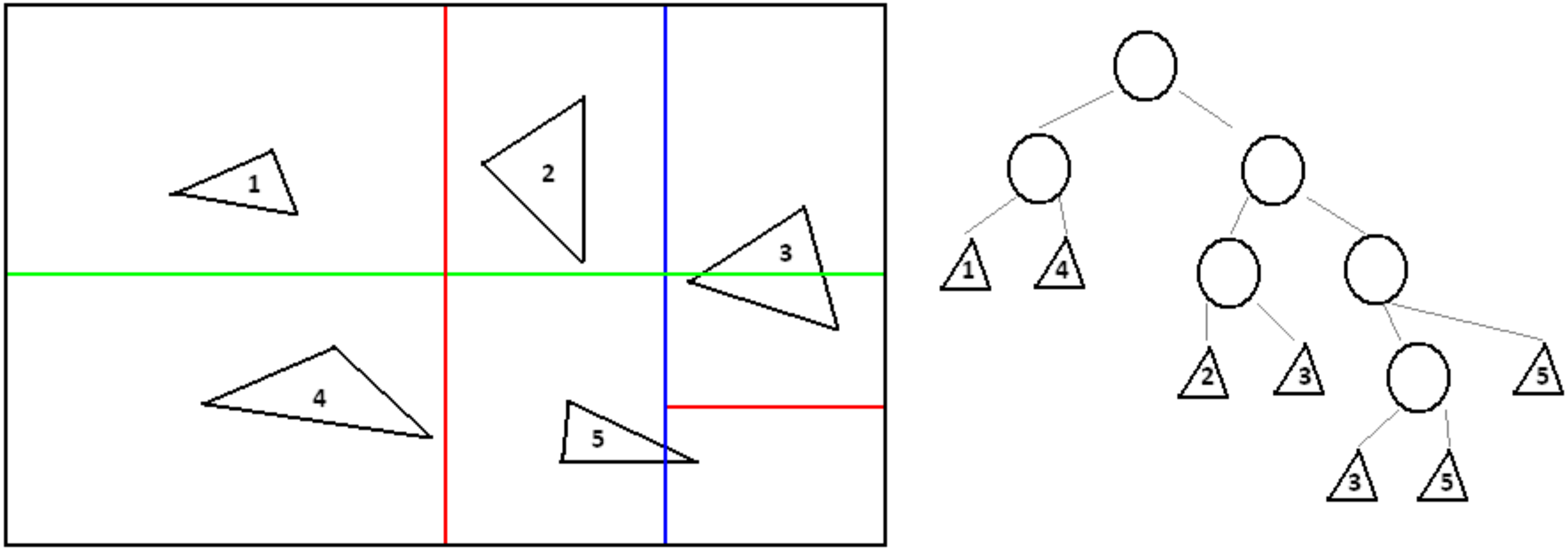
KD-Tree

naive „spatial-median“ algorithm



KD-Tree

naive „spatial-median“ algorithm



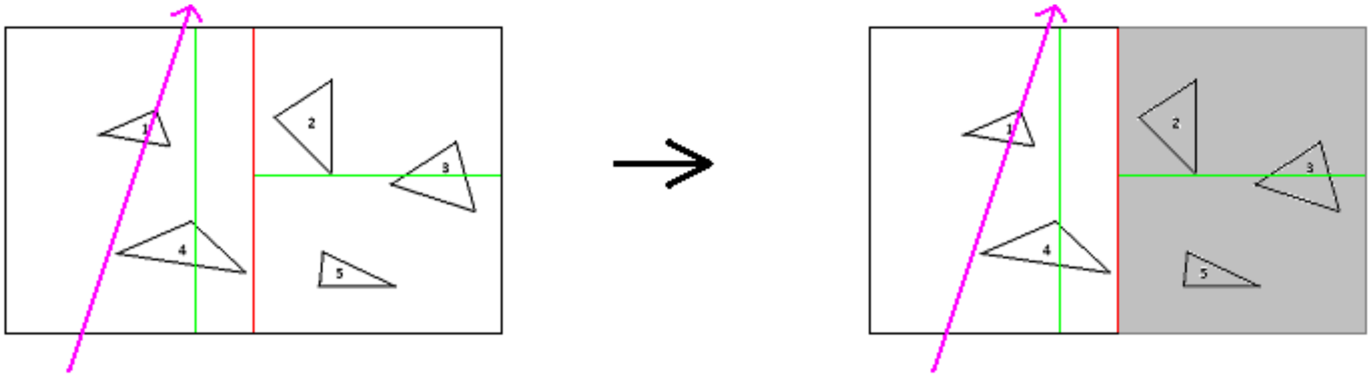
KD-Tree

Traversal

- leaf
  - intesect with triangles in list
- inner node
  - intersect with splitting plane
  - choose child(ren) to continue (three cases)

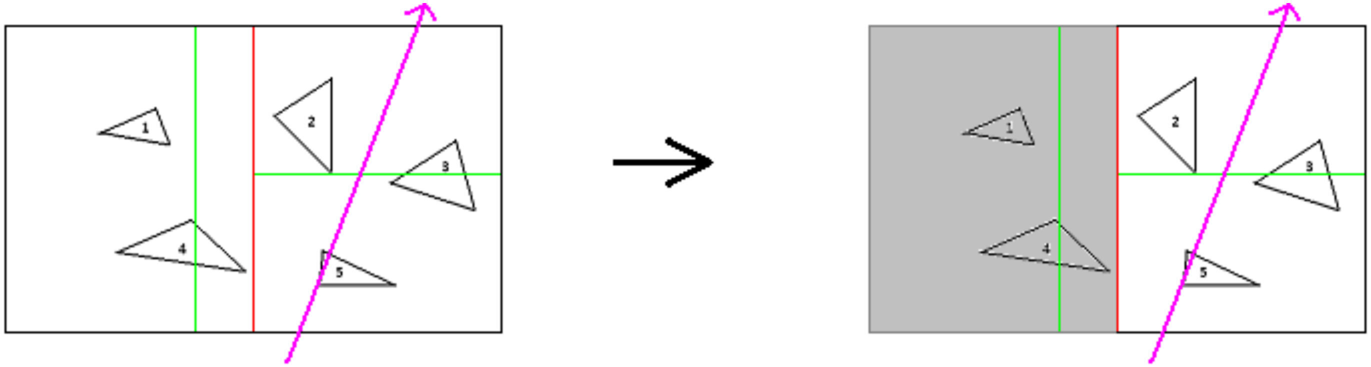
KD-Tree

Case 1



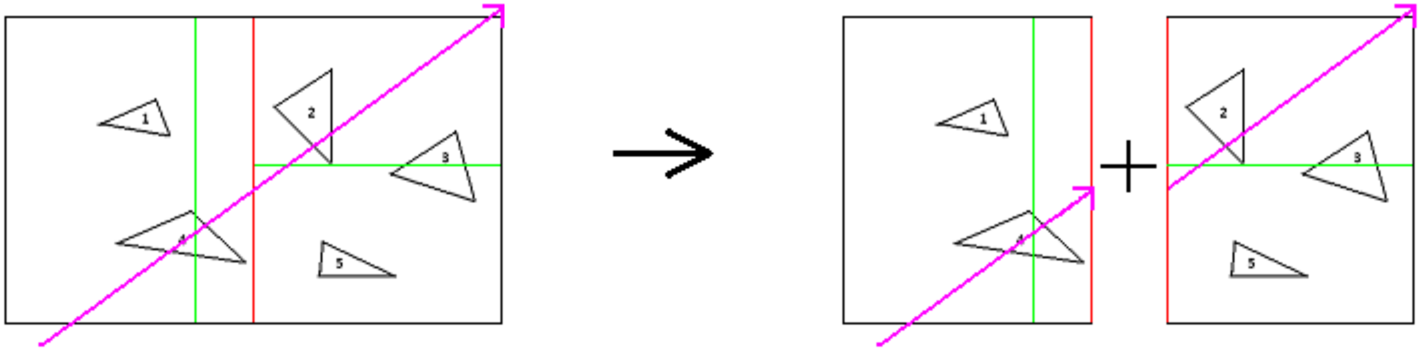
KD-Tree

Case 2



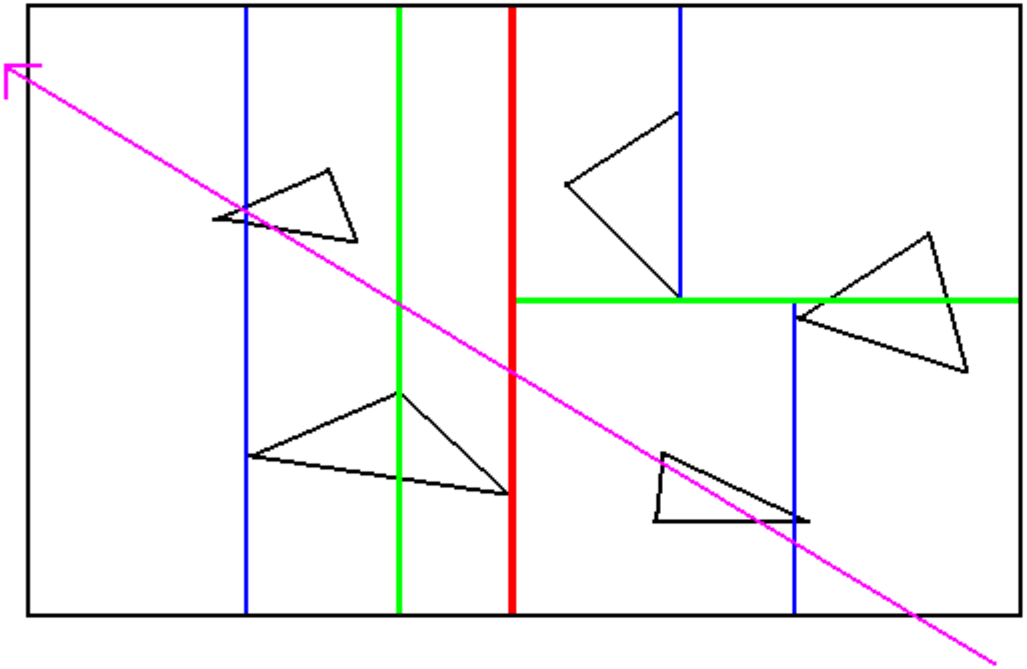
KD-Tree

Case 3



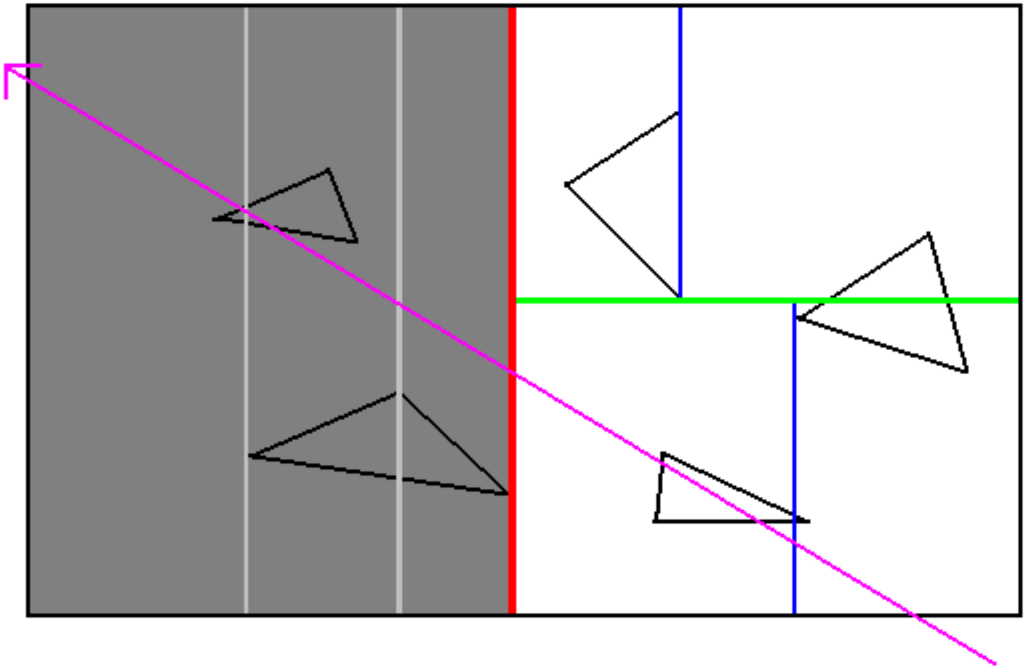
KD-Tree

Traversal



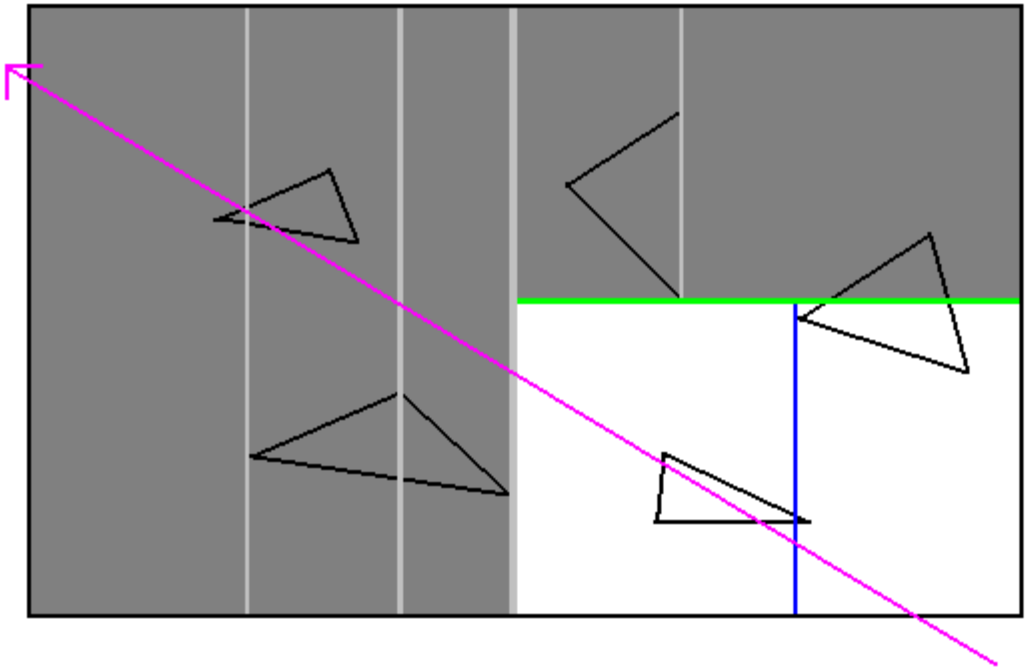
KD-Tree

Traversal



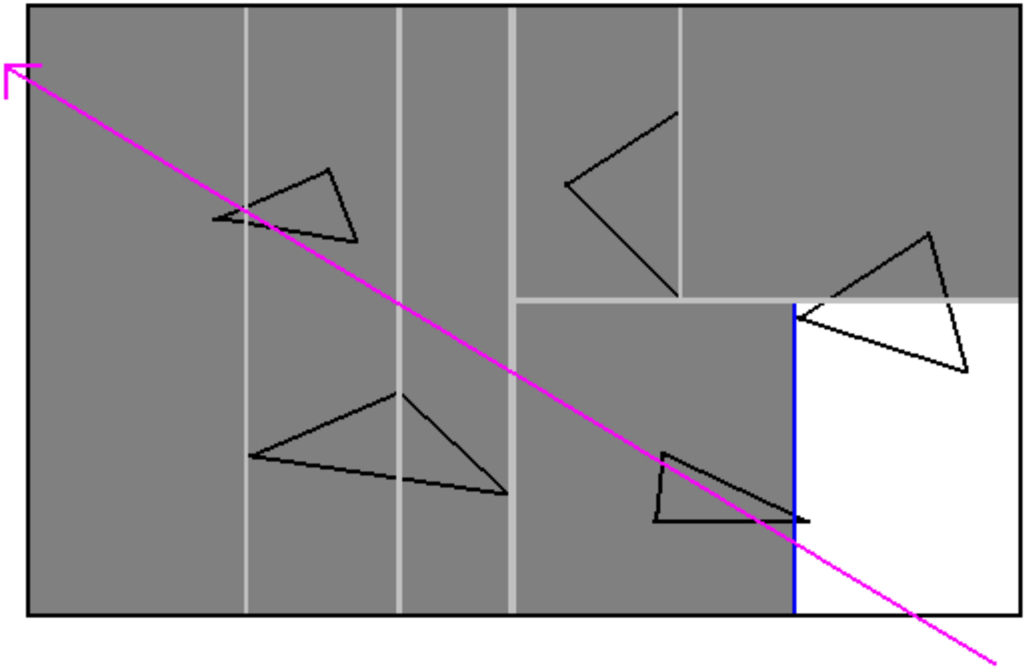
KD-Tree

Traversal



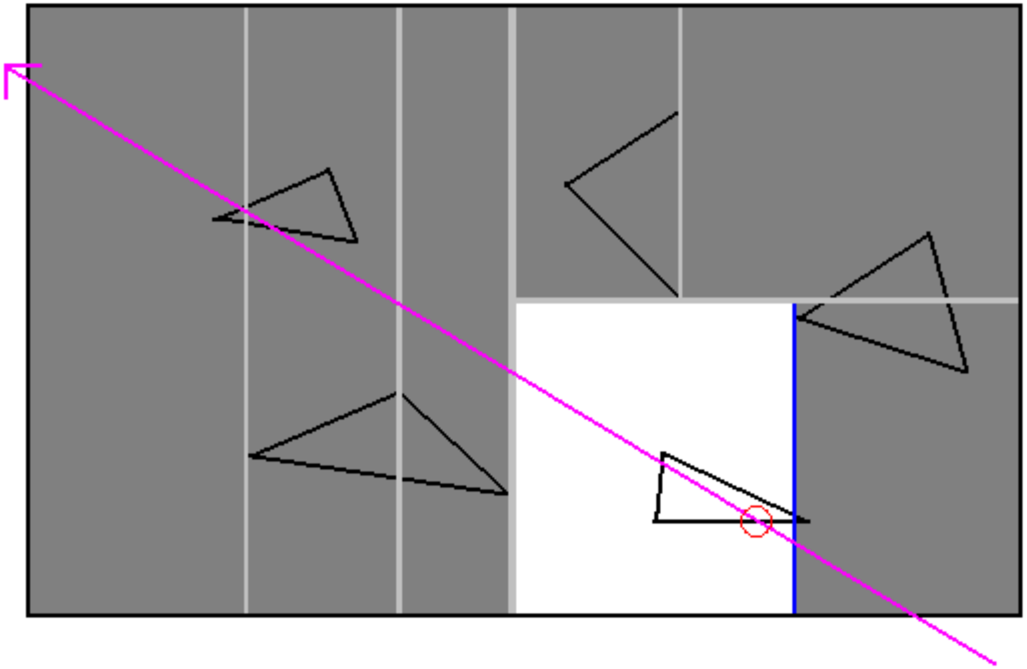
KD-Tree

Traversal



KD-Tree

Traversal



KD-Tree

Pros and Cons

- Pros:
  - front to back traversal
  - early ray termination
  - simple traversal

- Cons:
  - unknown memory usage
    - triangles are referenced multiple times
  - possible numeric problems
  - long building times

KD-Tree

Building Steps

- for each node:
  - check triangles
  - split node
    - clip empty space or
    - find splitting plane
  - stop if:
    - minimum number of triangles per leaf reached or
    - costs of possible child >= costs of current node
- serialize structure

KD-Tree

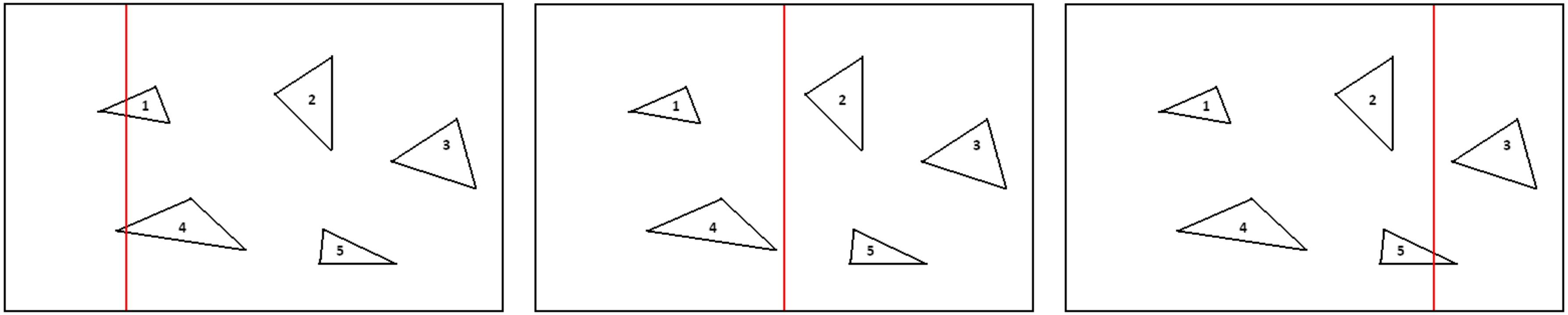
Modes

- bin based SAH
- vertex based SAH
- SAH based Spatial Median

KD-Tree

Modes

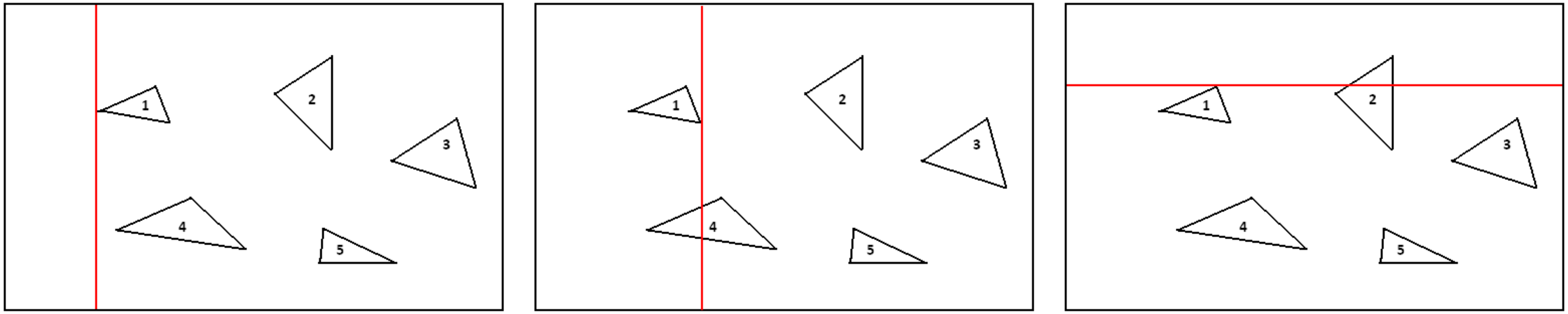
- bin based SAH



KD-Tree

Modes

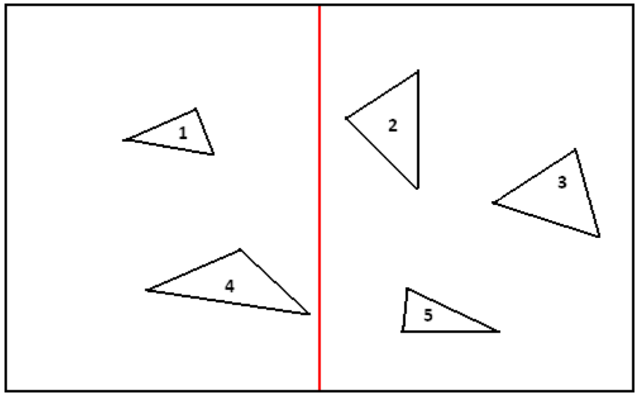
- vertex based SAH
  - reader/writer

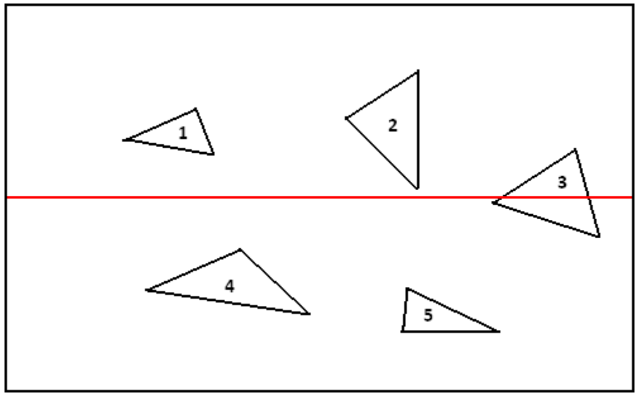


KD-Tree

Modes

- SAH based Spatial Median





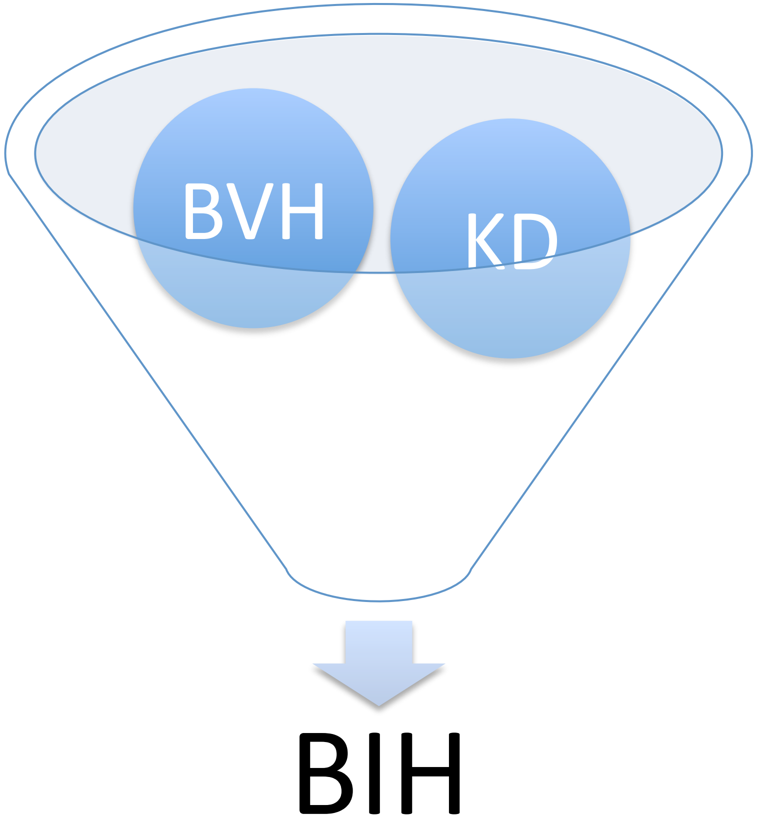
KD-Tree

Fields of Research

- find mode that results in best raytracing performance
- find best combination of build-options
  - SAH cost calculation
  - empty space ratio
  - number of bins

Acceleration Structures

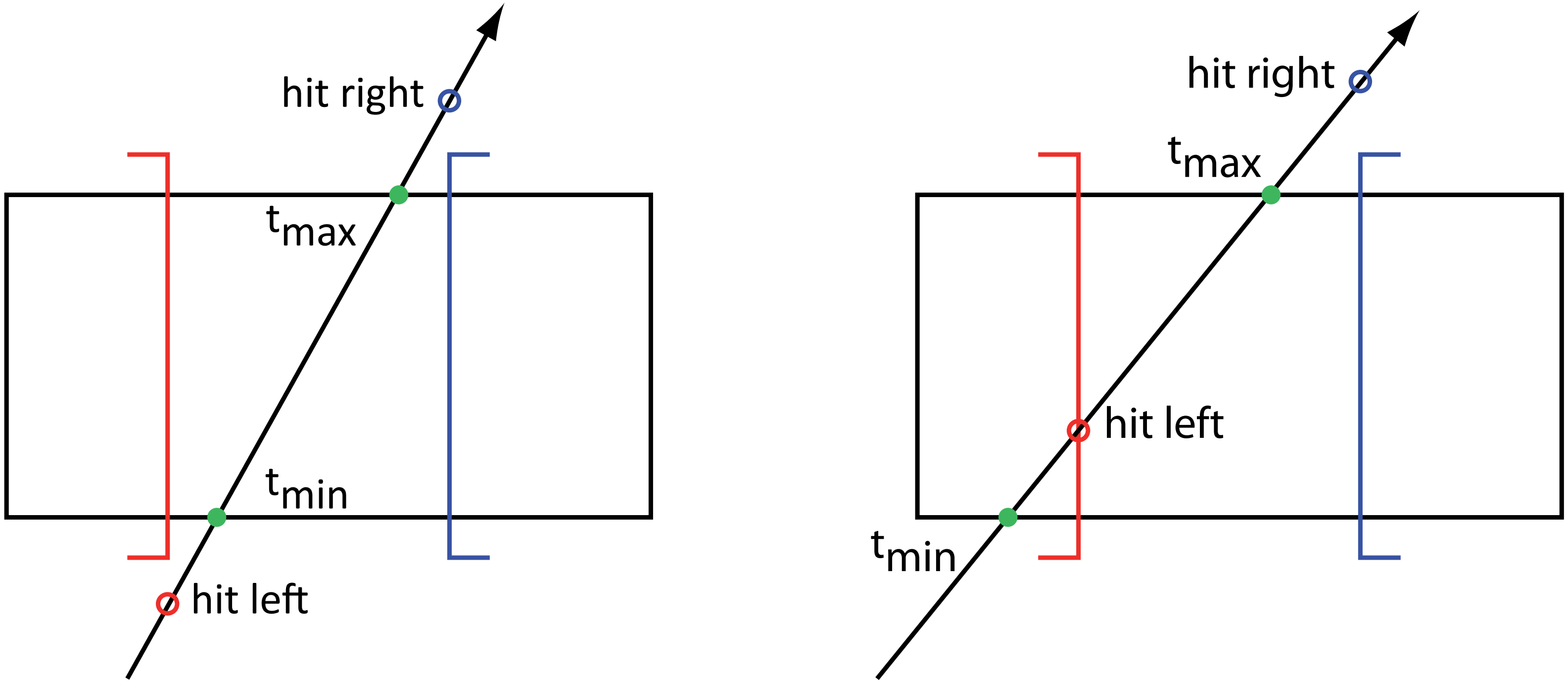
Bounding Interval Hierarchy (BIH)



- cross-over of **partitioning object lists** and **traversing spatial partitions**
- uses **two splitting planes**
- SAH-based
- hierarchically subdividing scene AABB

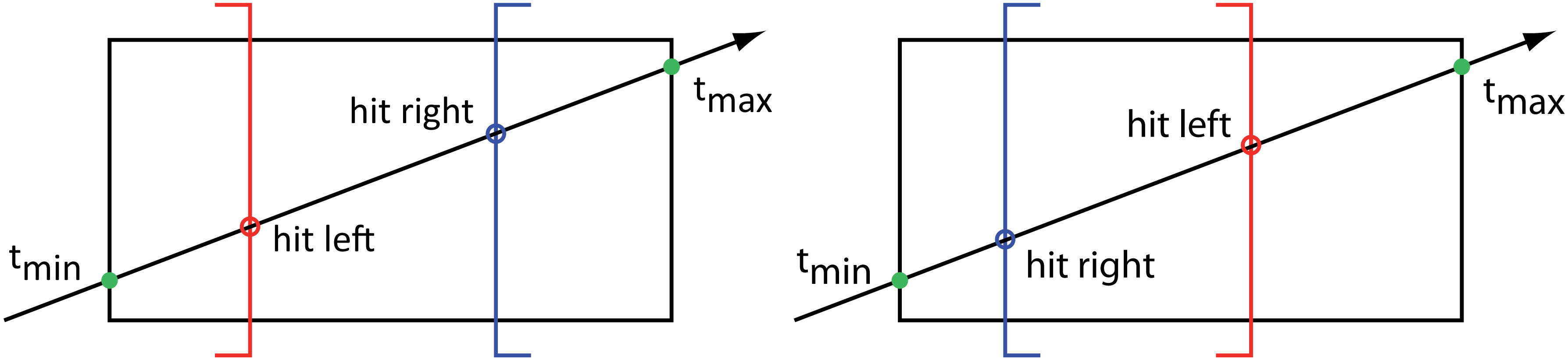
Acceleration Structures

Bounding Interval Hierarchy – Intersection



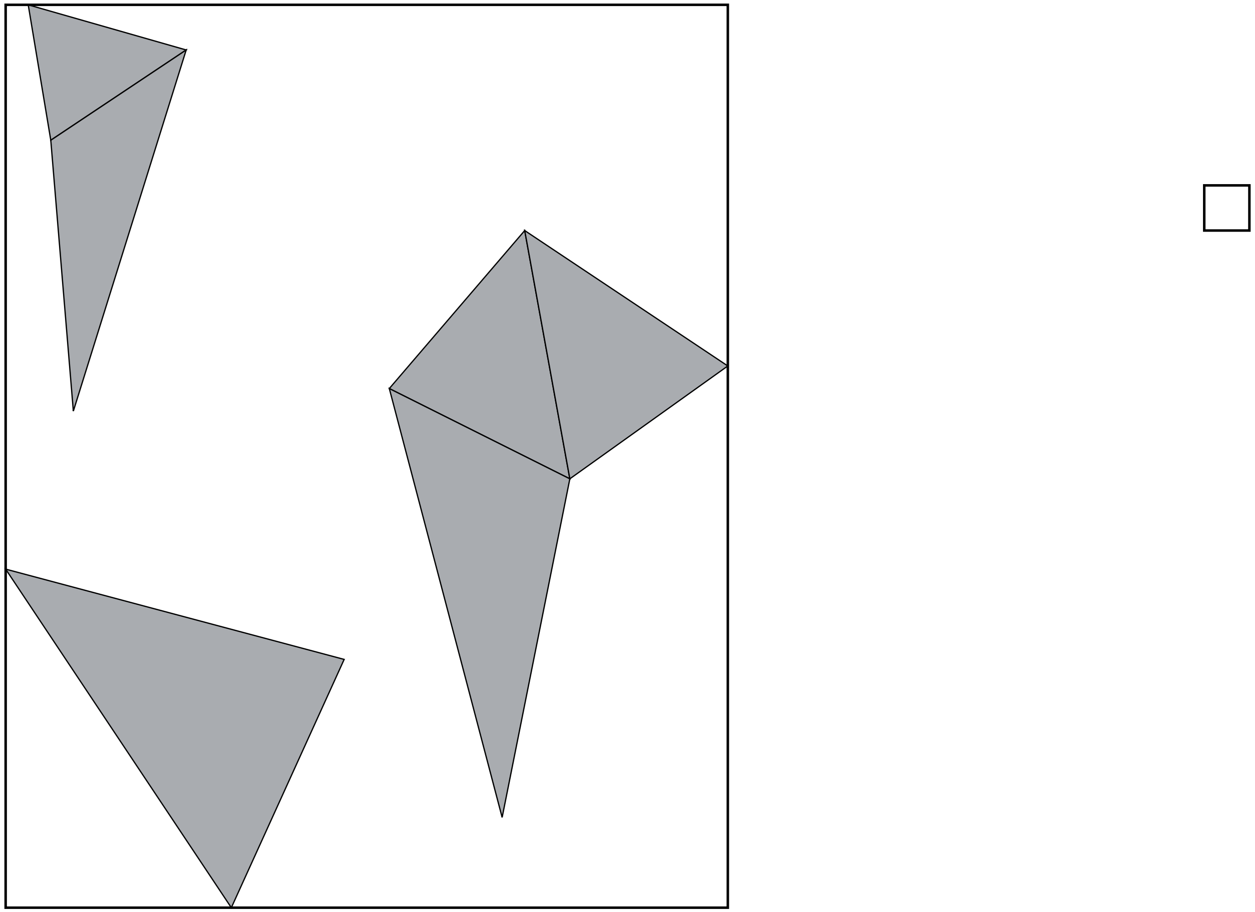
Acceleration Structures

Bounding Interval Hierarchy – Intersection



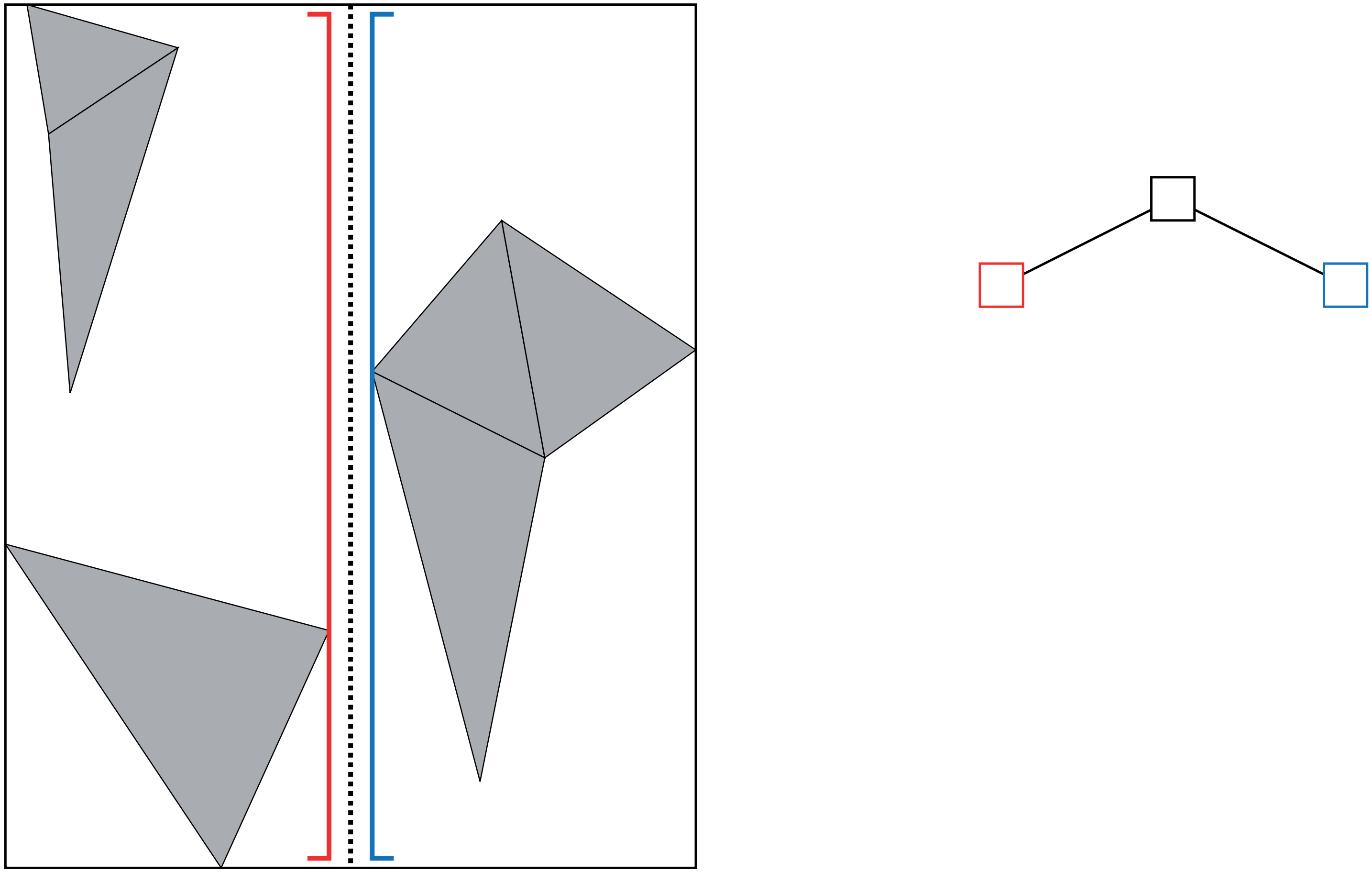
Acceleration Structures

Bounding Interval Hierarchy – Construction



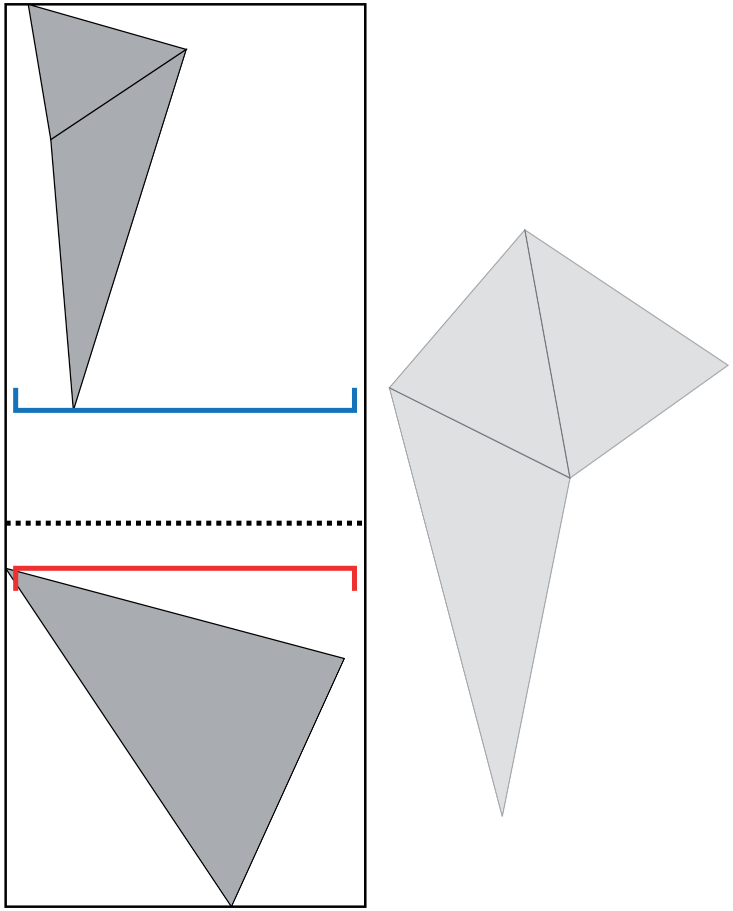
Acceleration Structures

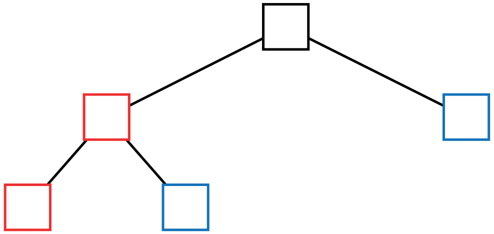
Bounding Interval Hierarchy – Construction



Acceleration Structures

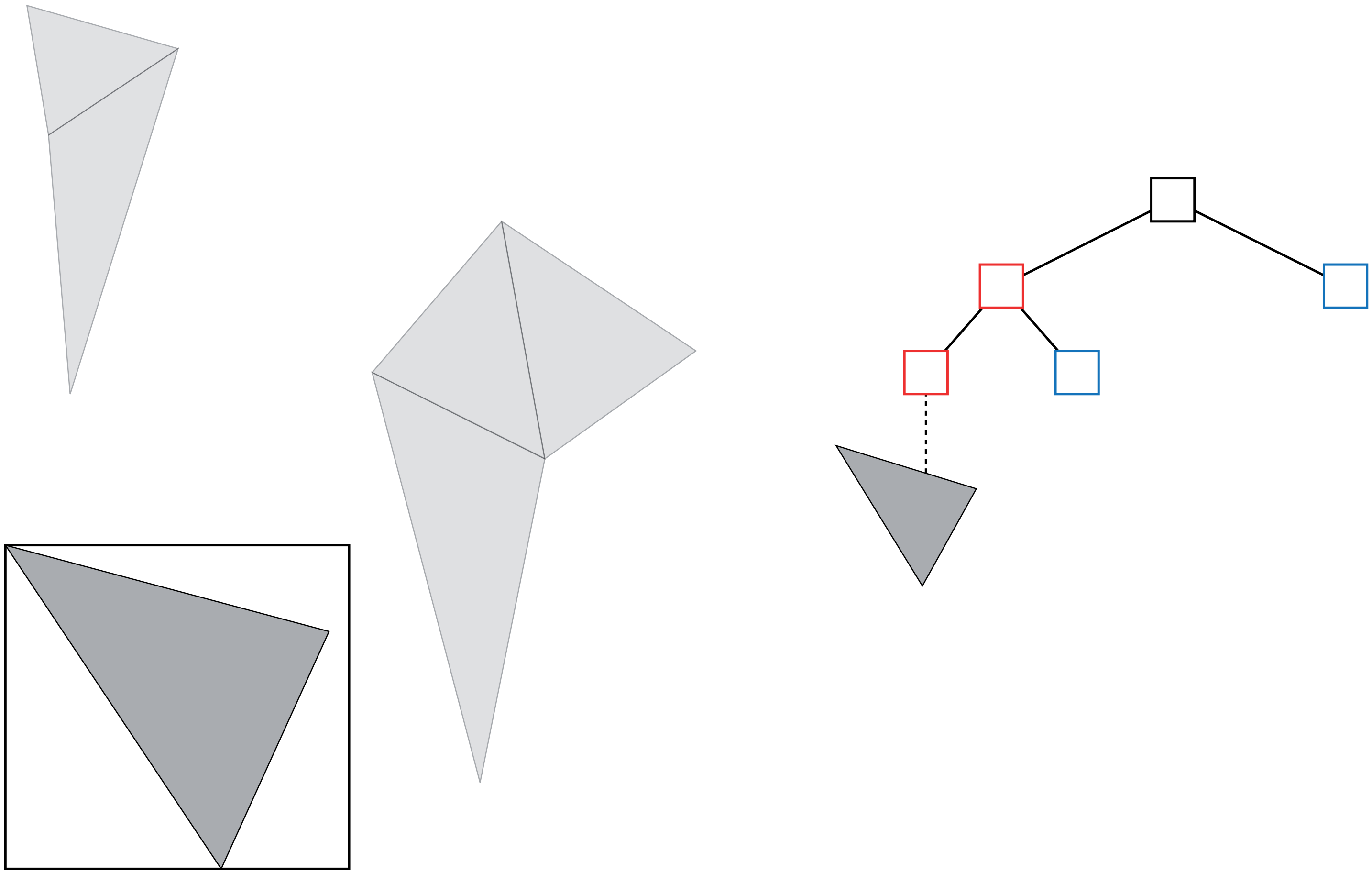
Bounding Interval Hierarchy – Construction





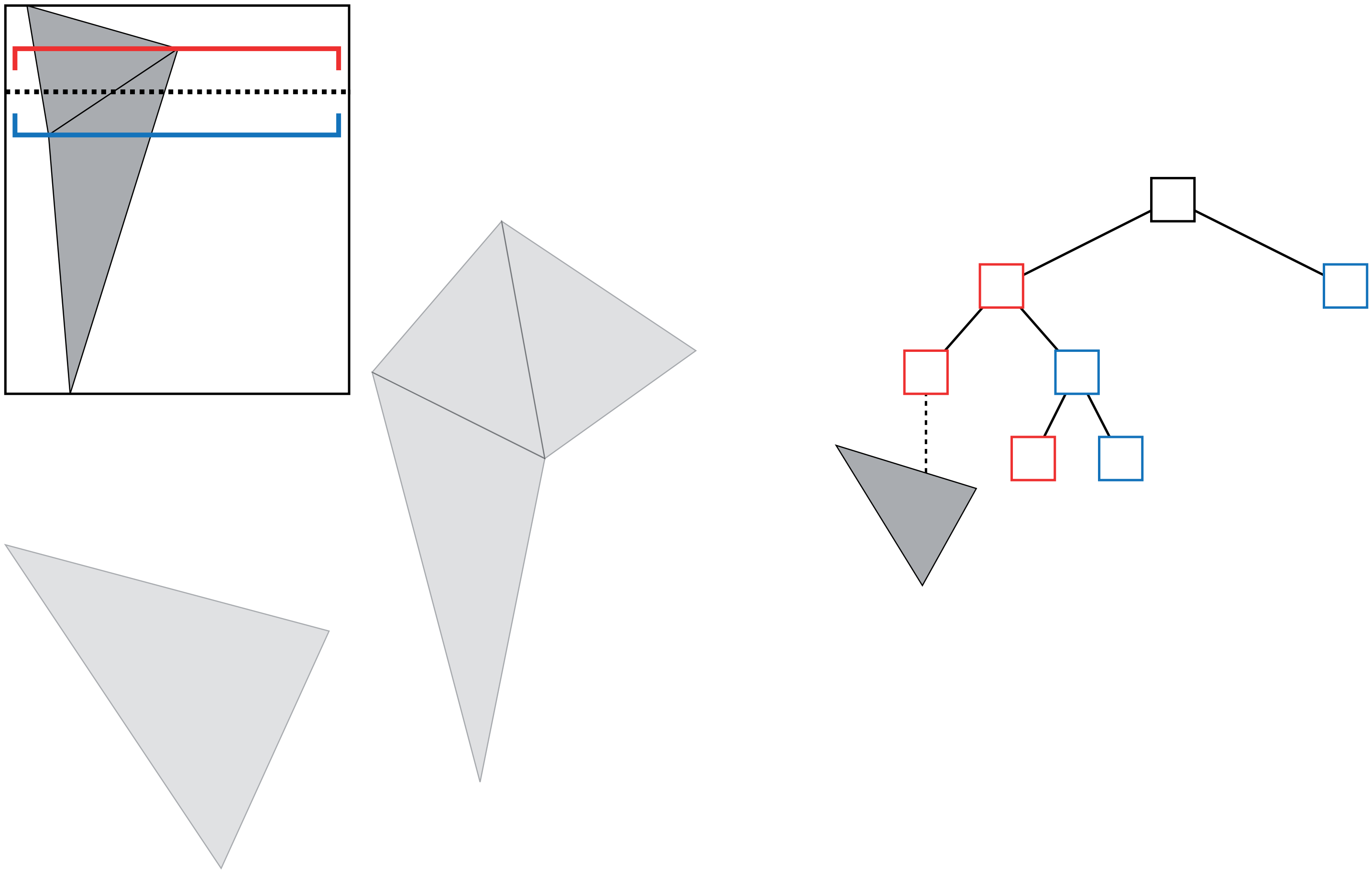
Acceleration Structures

Bounding Interval Hierarchy – Construction



Acceleration Structures

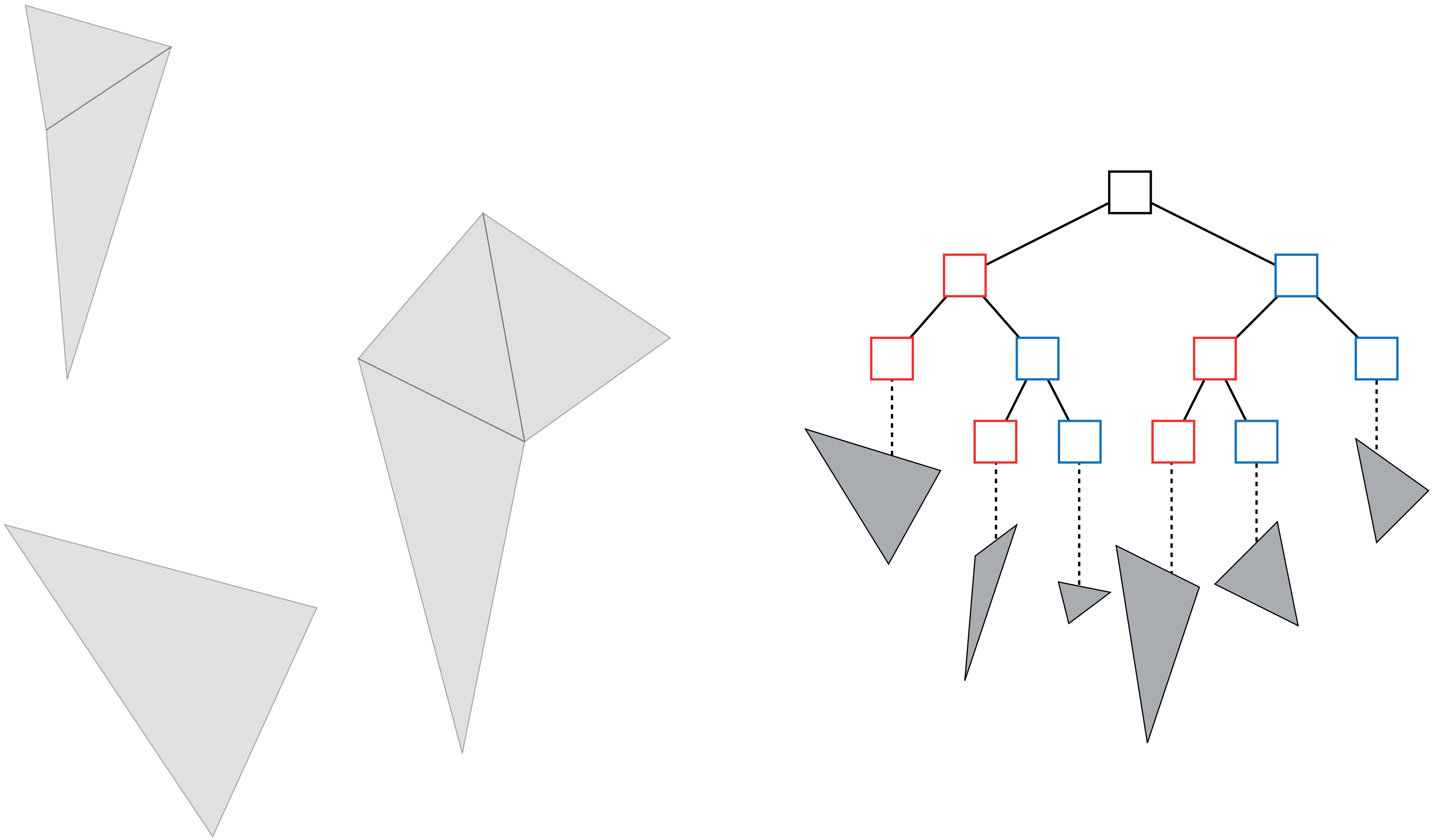
Bounding Interval Hierarchy – Construction



Bauhaus-Universität Weimar

Acceleration Structures

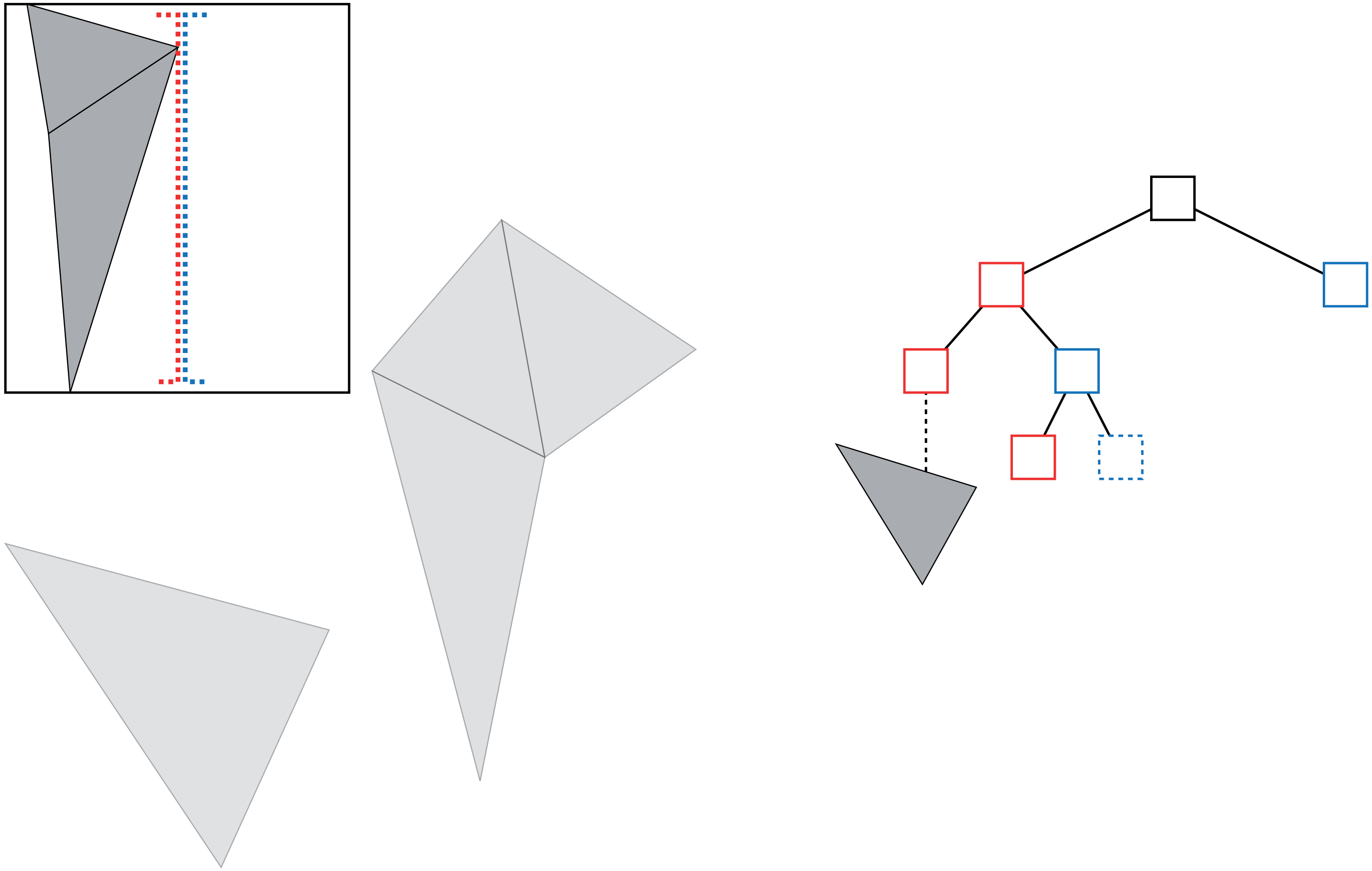
Bounding Interval Hierarchy – Construction



Bauhaus-Universität Weimar

Acceleration Structures

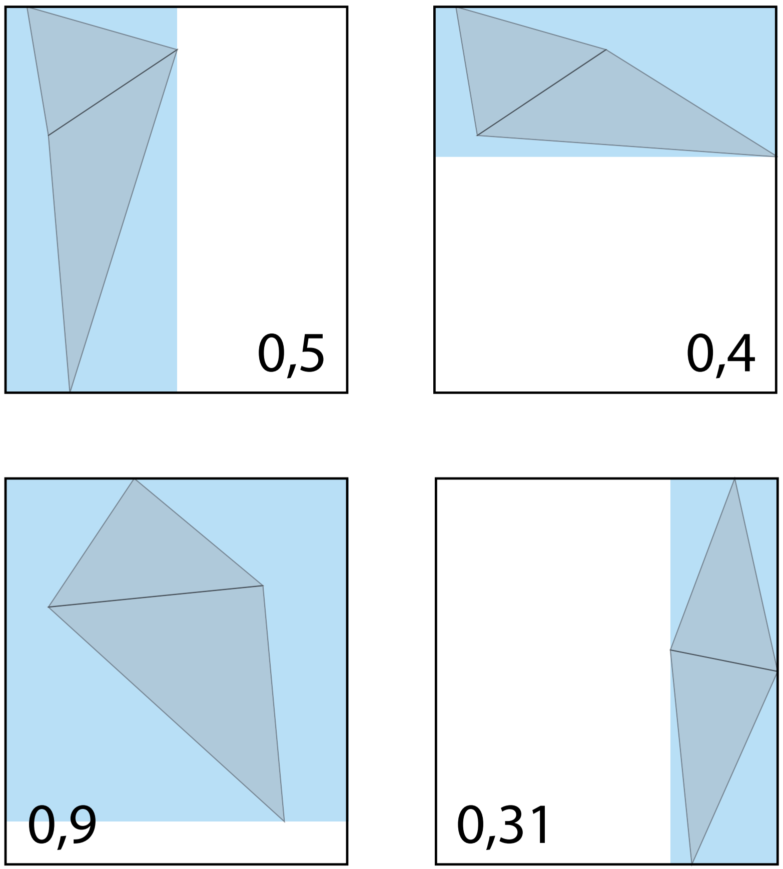
Detect Empty Space



Acceleration Structures

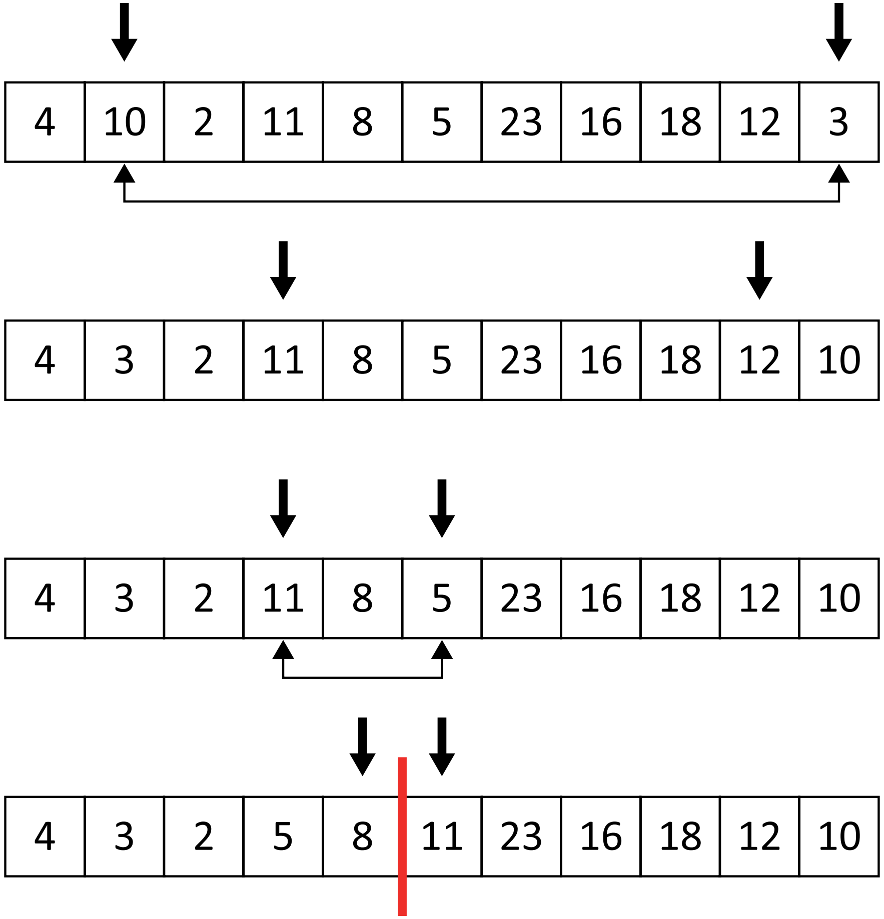
Detect Empty Space

- surface ratio
- higher threshold
  - more empty nodes
  - tighter fitted BB's
- lower threshold
  - less empty nodes
  - loose BB's



Acceleration Structures

Sorting Triangles



4	10	2	11	8	5	23	16	18	12	3

4	3	2	11	8	5	23	16	18	12	10

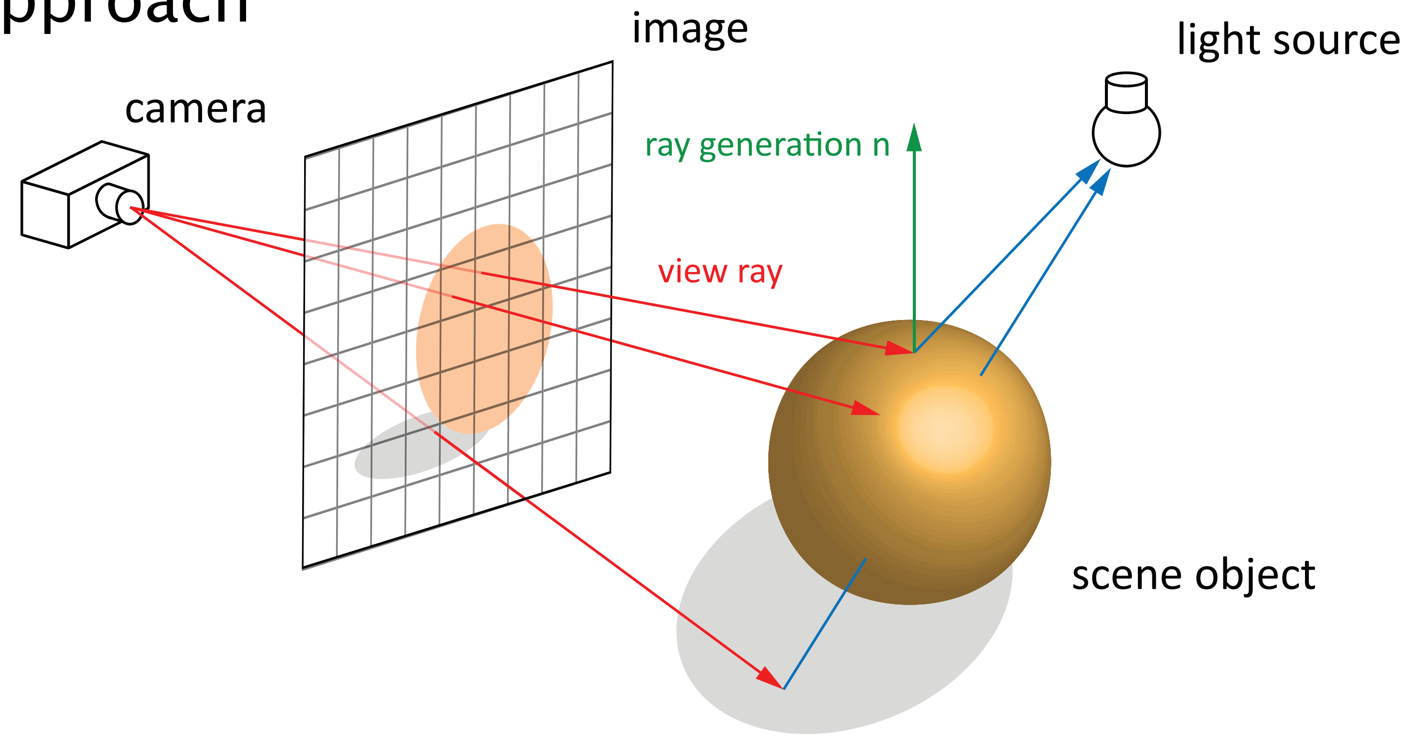
4	3	2	11	8	5	23	16	18	12	10

4	3	2	5	8	11	23	16	18	12	10

Raytracing

Recursive

Classic approach



- generation of primary ray
- tracing of one ray at a time in succession

Raytracing

Recursive

Pros:

- simple algorithm
- small memory usage

Cons:

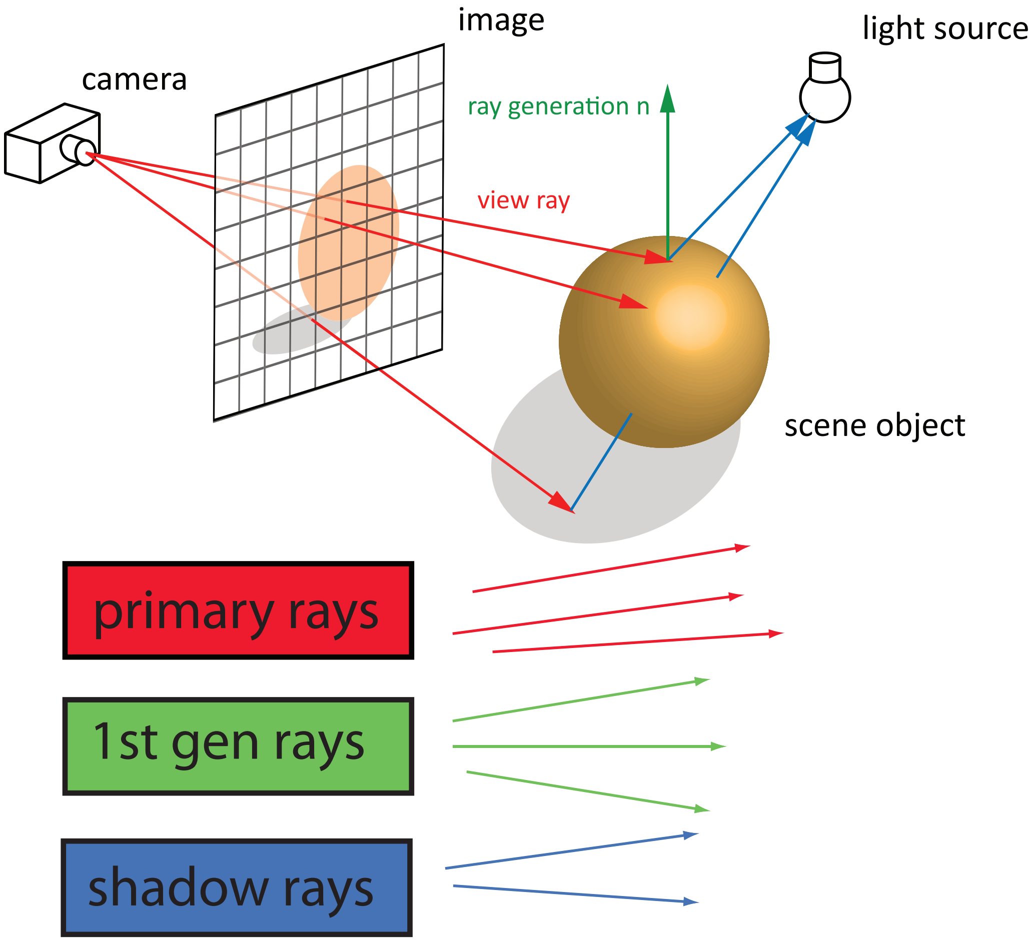
- lots of CPU cache penalty, little to no coherence between rays – ergo slow

Raytracing

Iterative

Our approach:

- rays are generated and shaded in bulks – each generation

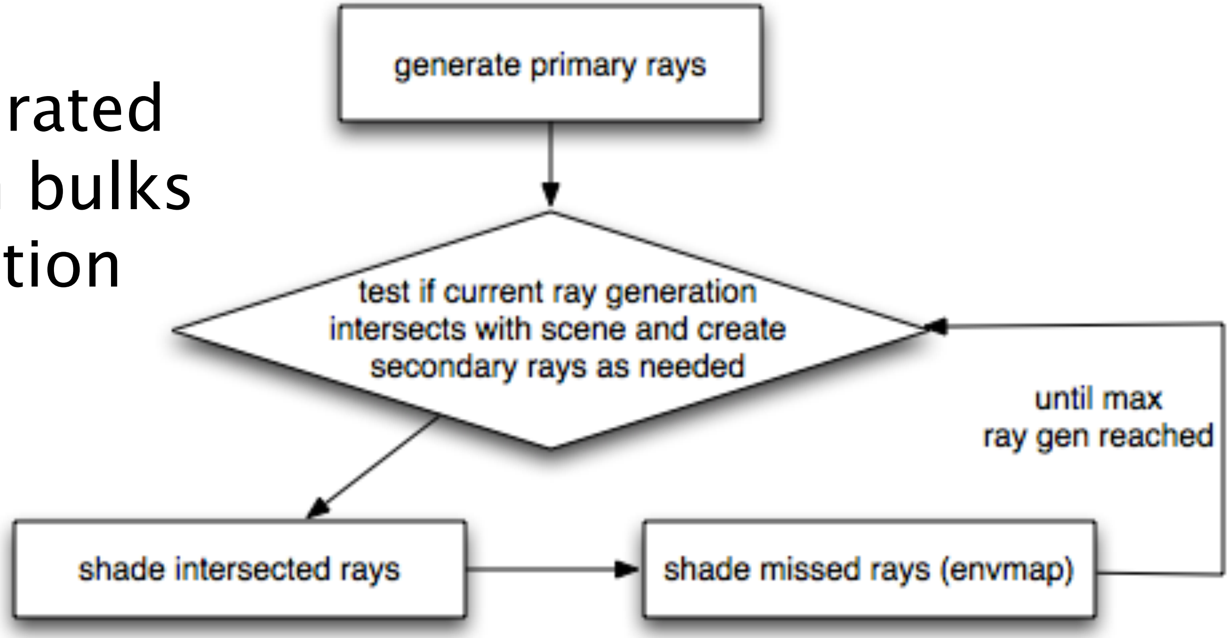


Raytracing

Iterative

Our approach:

- rays are generated and shaded in bulks – each generation


generate primary rays
test if current ray generation intersects with scene and create secondary rays as needed
until max ray gen reached
shade intersected rays
shade missed rays (envmap)

Raytracing

Iterative

Pros:

- CPU cache coherence between rays
- longer hot cache stages for acceleration structures
- no allocation or destruction of rays during rendering

Cons:

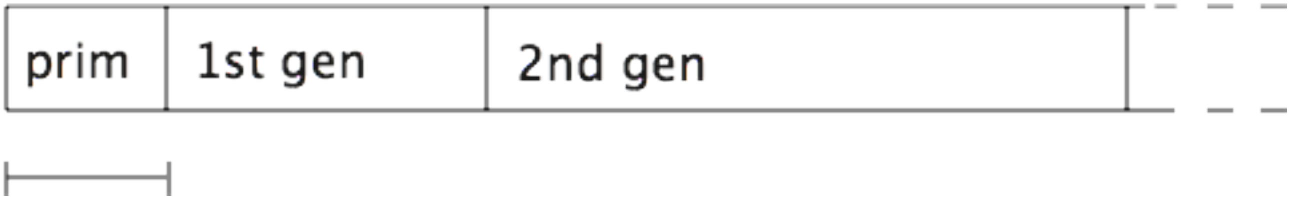
- huge memory footprint due to initial allocation of ray generations (exponential memory usage)

Raytracing

Iterative

Memory usage:

Size of containers chosen for worst case scenario (each ray creates 2 secondary rays each generation)



4 containers used for intersection tests

+ num_lights containers used for shading

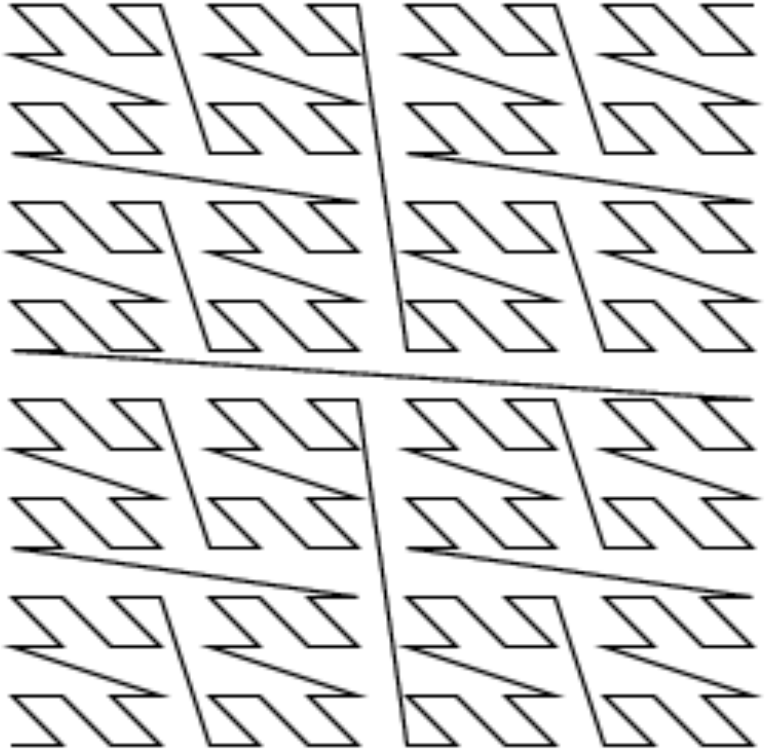
e.g.: sizeof(Ray) = 88 bytes, 800 x 600res, max 6 generations, 2 lights: **15468.75 MB**

Raytracing

Iterative

en detail:

- Z filling curve



Raytracing

Iterative

en detail:

- Z filling curve
- Schlick’s approximation

Raytracing

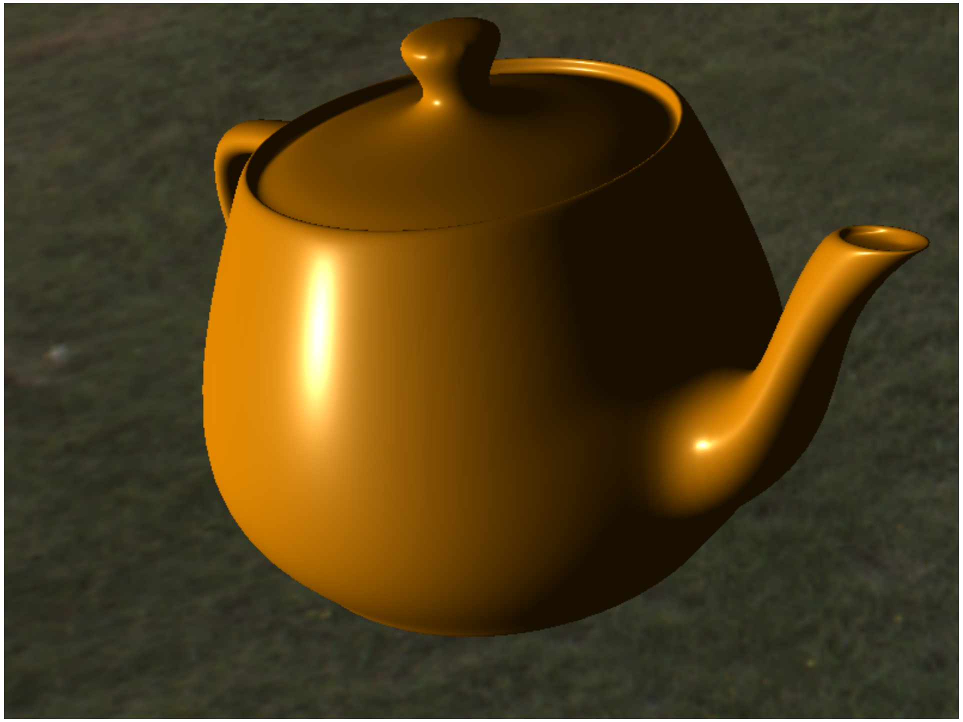
Iterative

en detail:

- Z filling curve
- Schlick’s approximation

Shading:

- basic phong shading



Raytracing

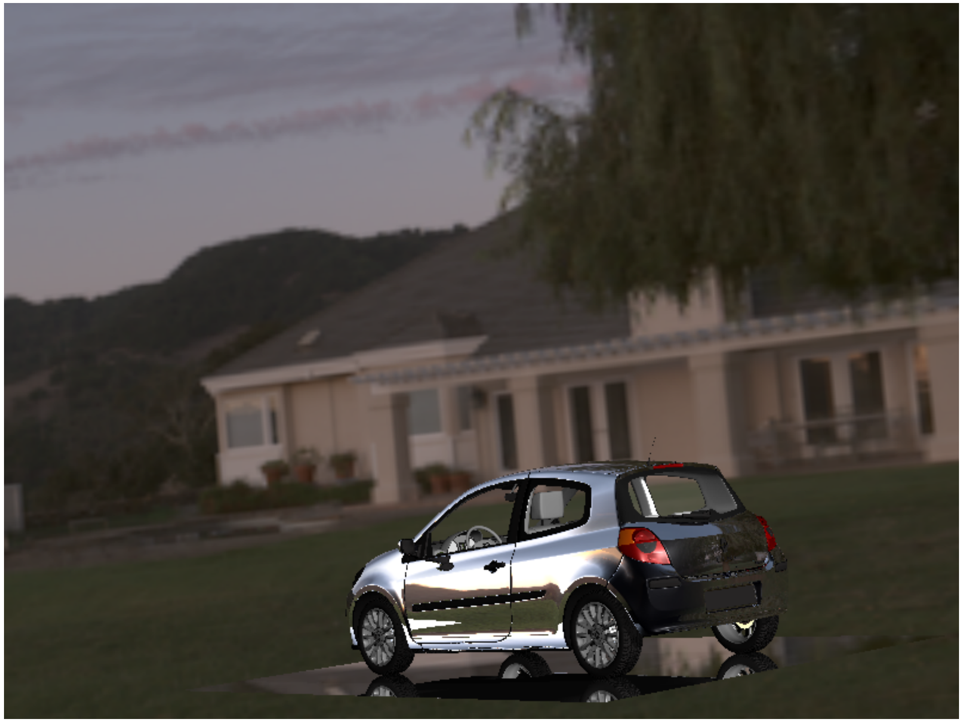
Iterative

en detail:

- Z filling curve
- Schlick’s approximation

Shading:

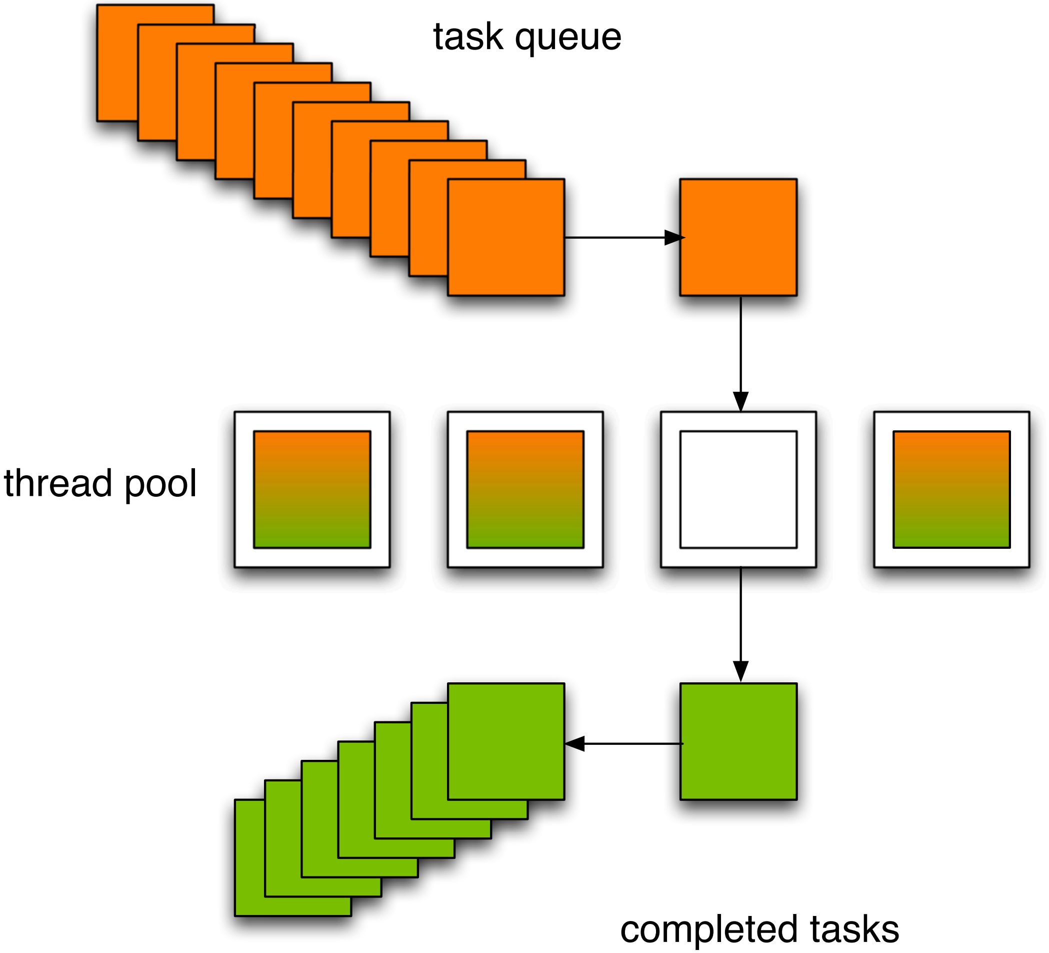
- basic phong shading
- spheric environment map



Parallelization

Boost Threads

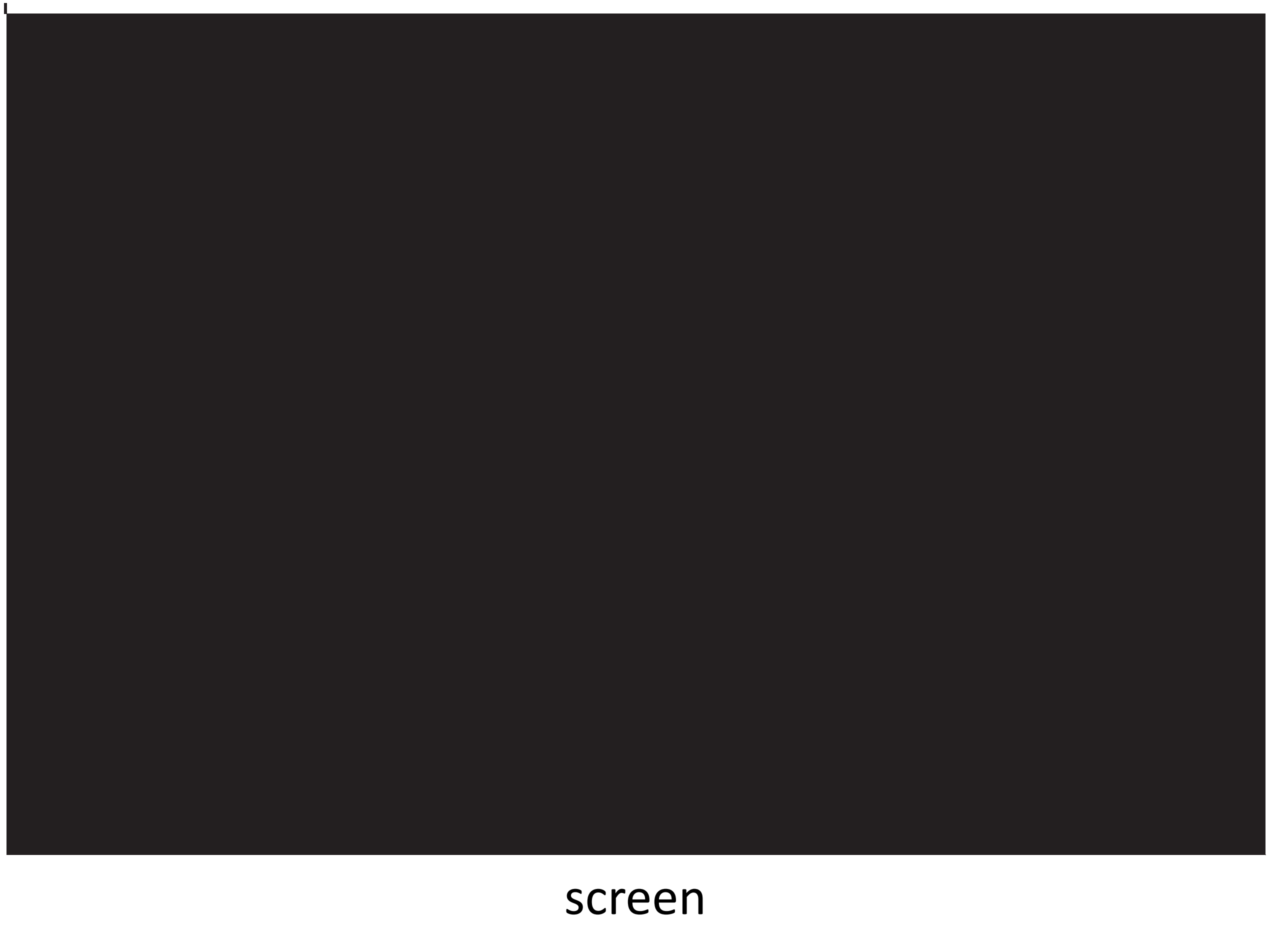
- Boost Threads
- Boost Thread Pool
  - Implementation of the Thread Pool Pattern



Parallelization

Boost Threads

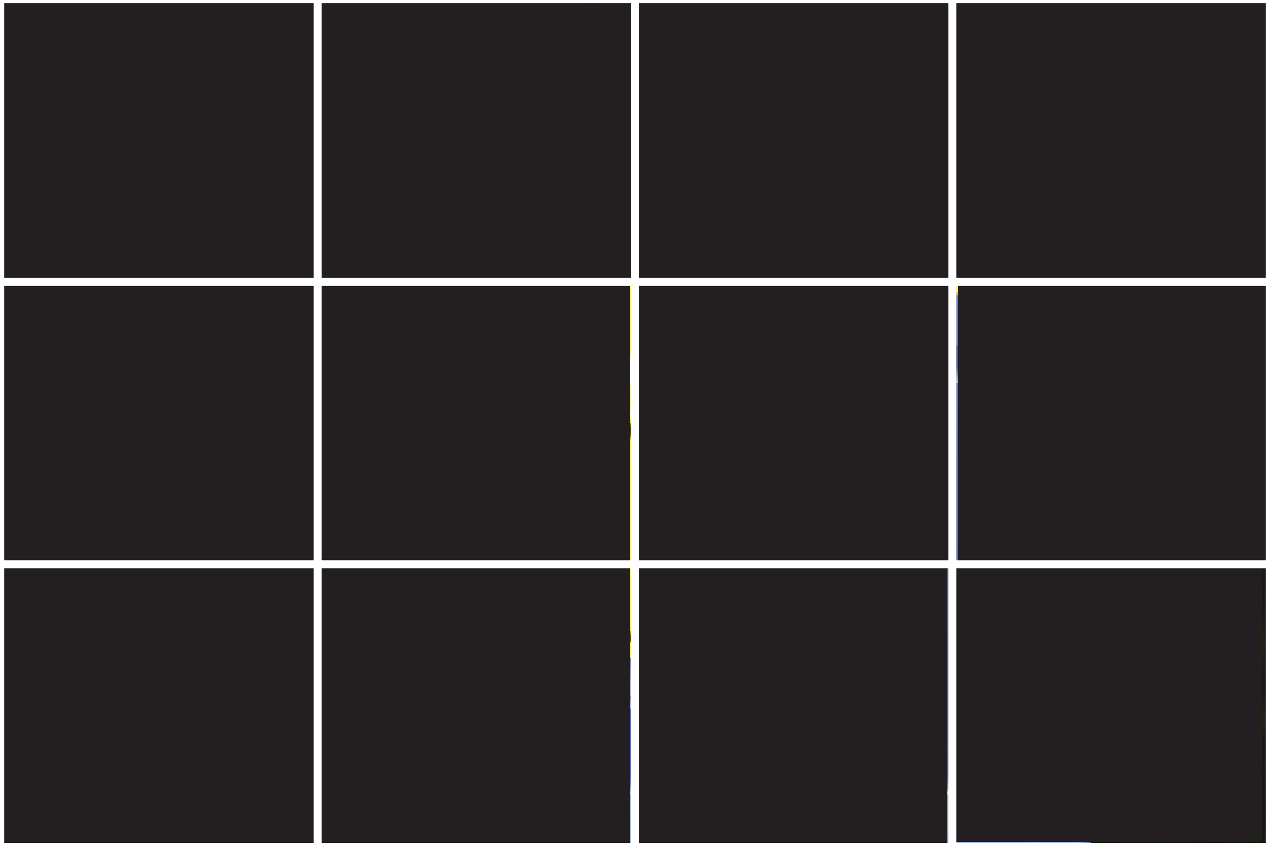
Implementation



Parallelization

Boost Threads

Implementation

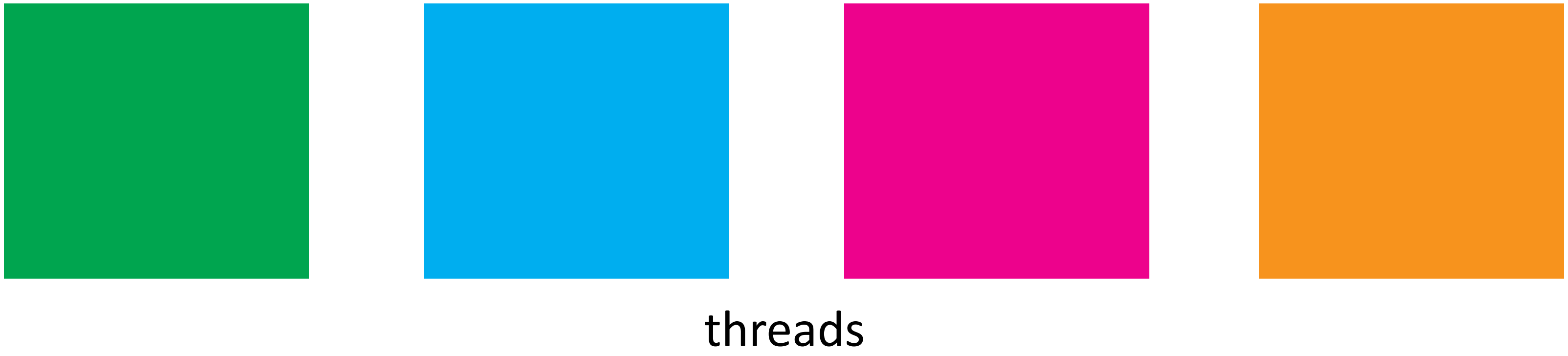


screen

Parallelization

Boost Threads

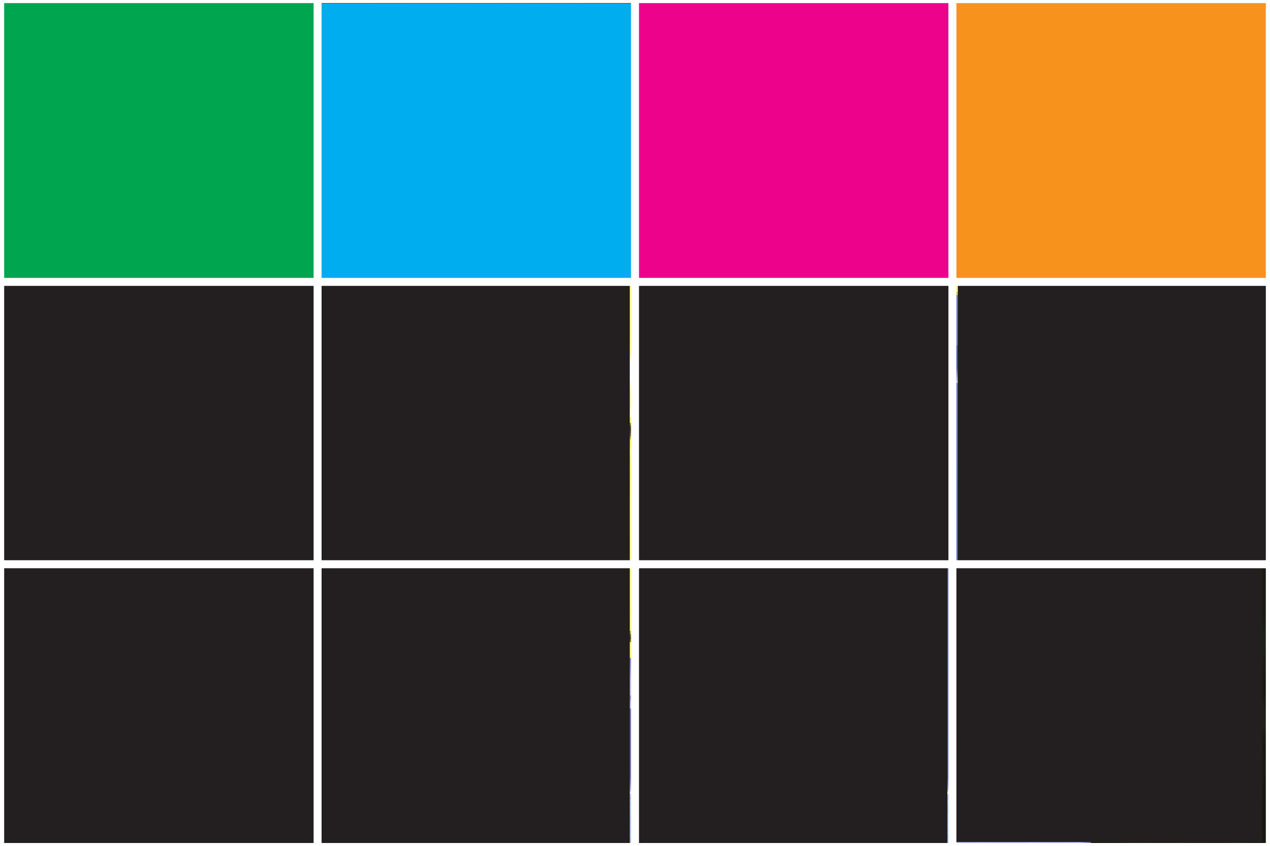
Implementation



Parallelization

Boost Threads

Implementation

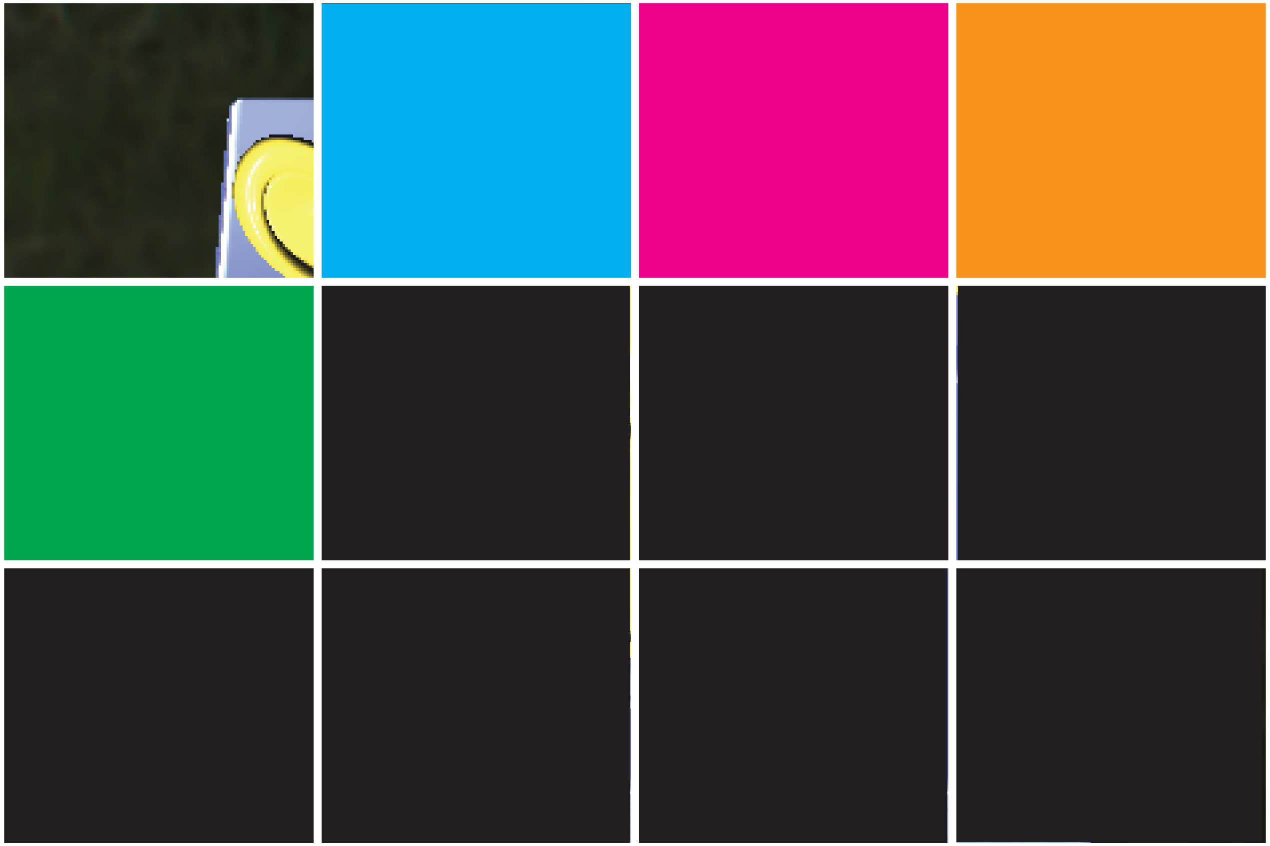


screen

Parallelization

Boost Threads

Implementation

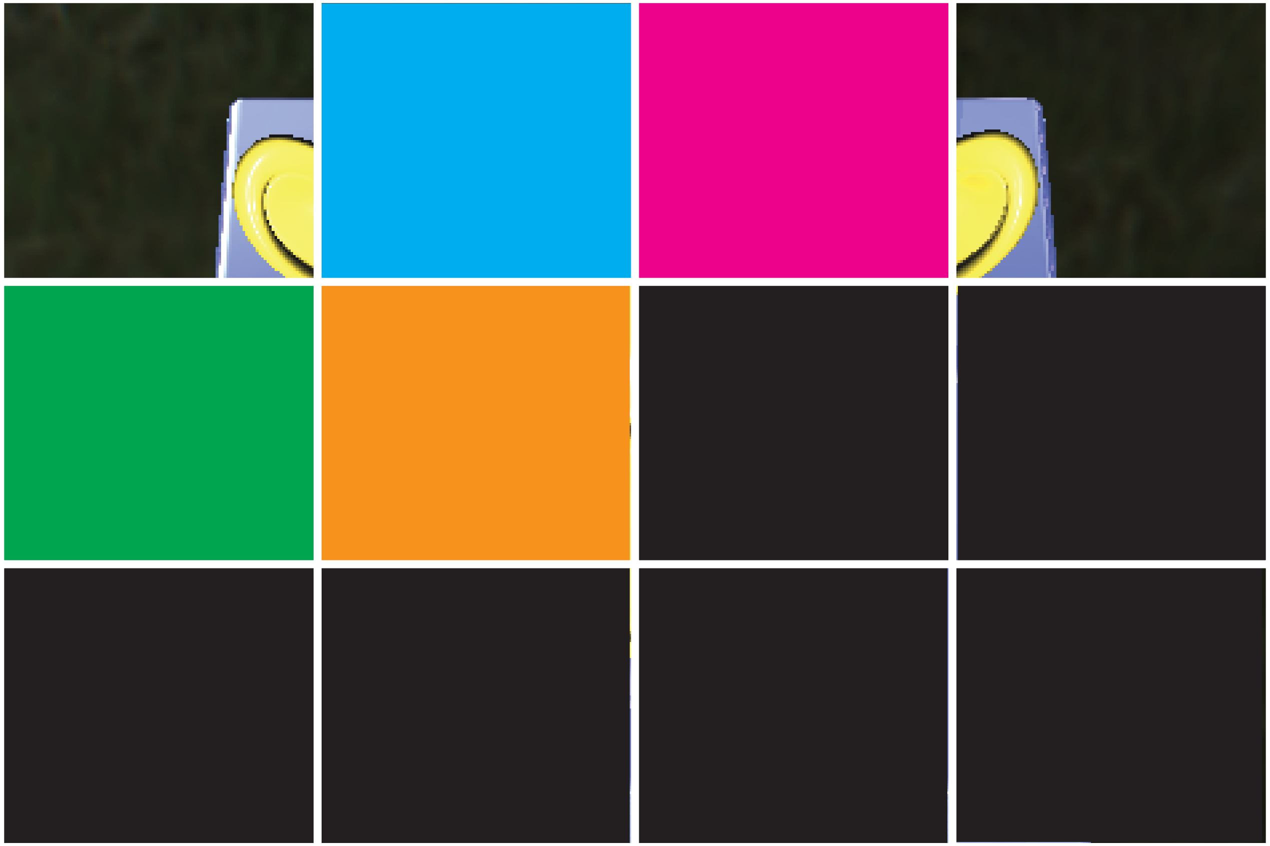


screen

Parallelization

Boost Threads

Implementation

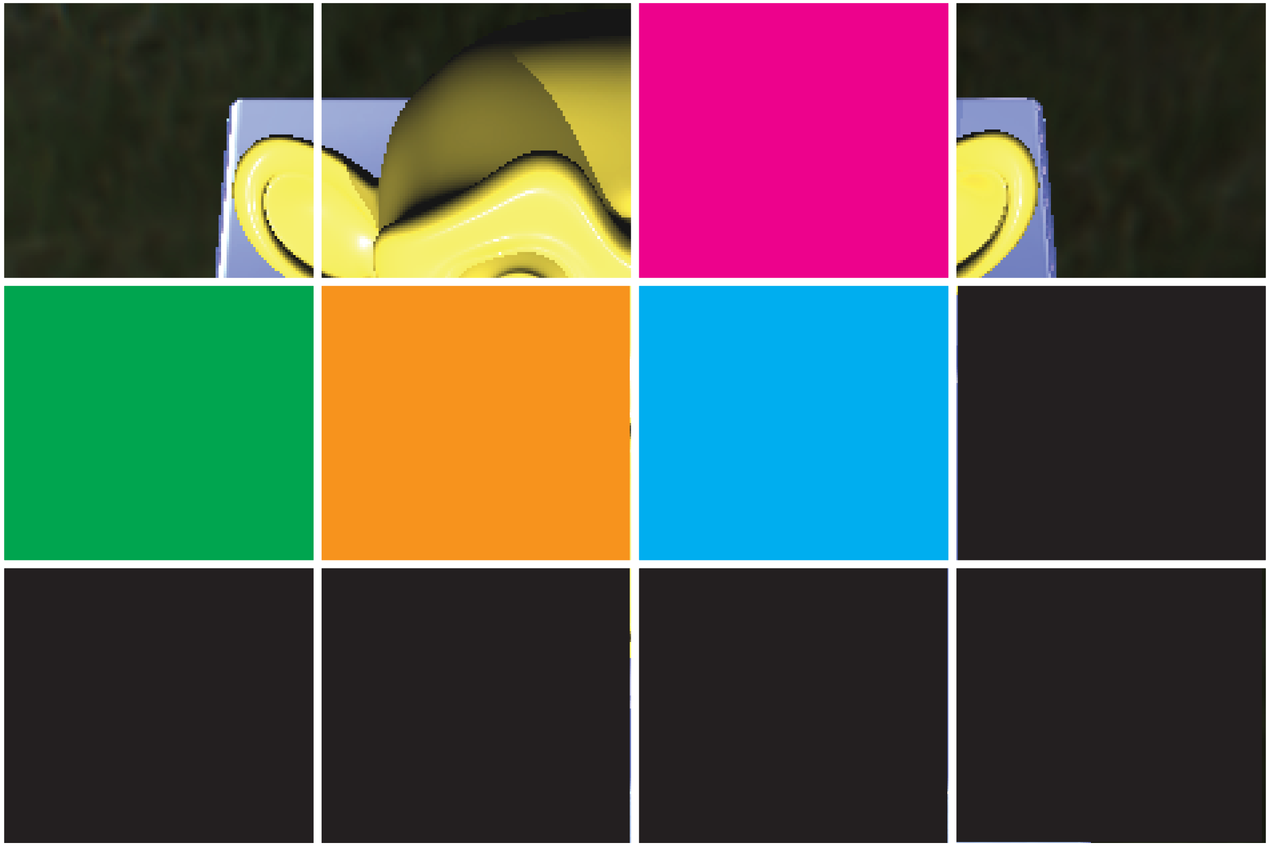


screen

Parallelization

Boost Threads

Implementation

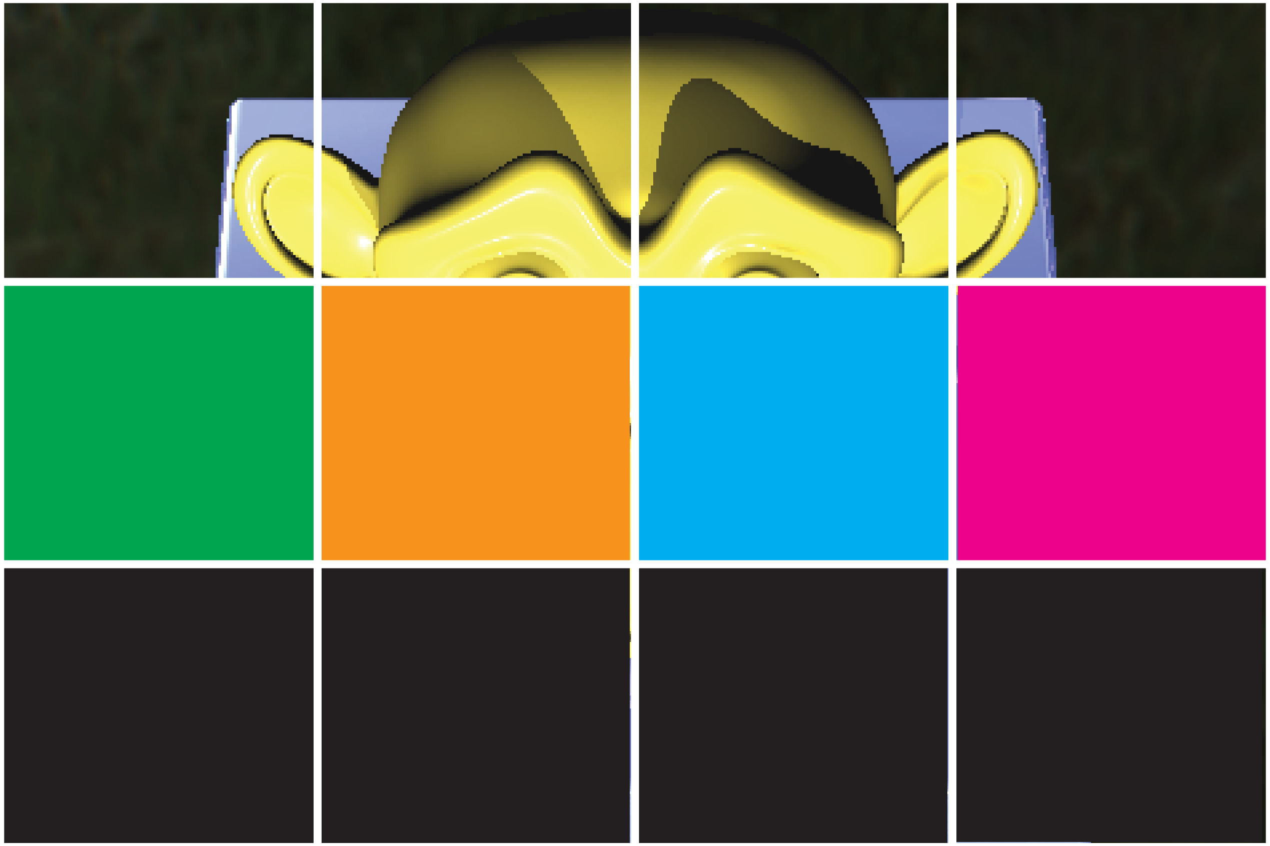


screen

Parallelization

Boost Threads

Implementation

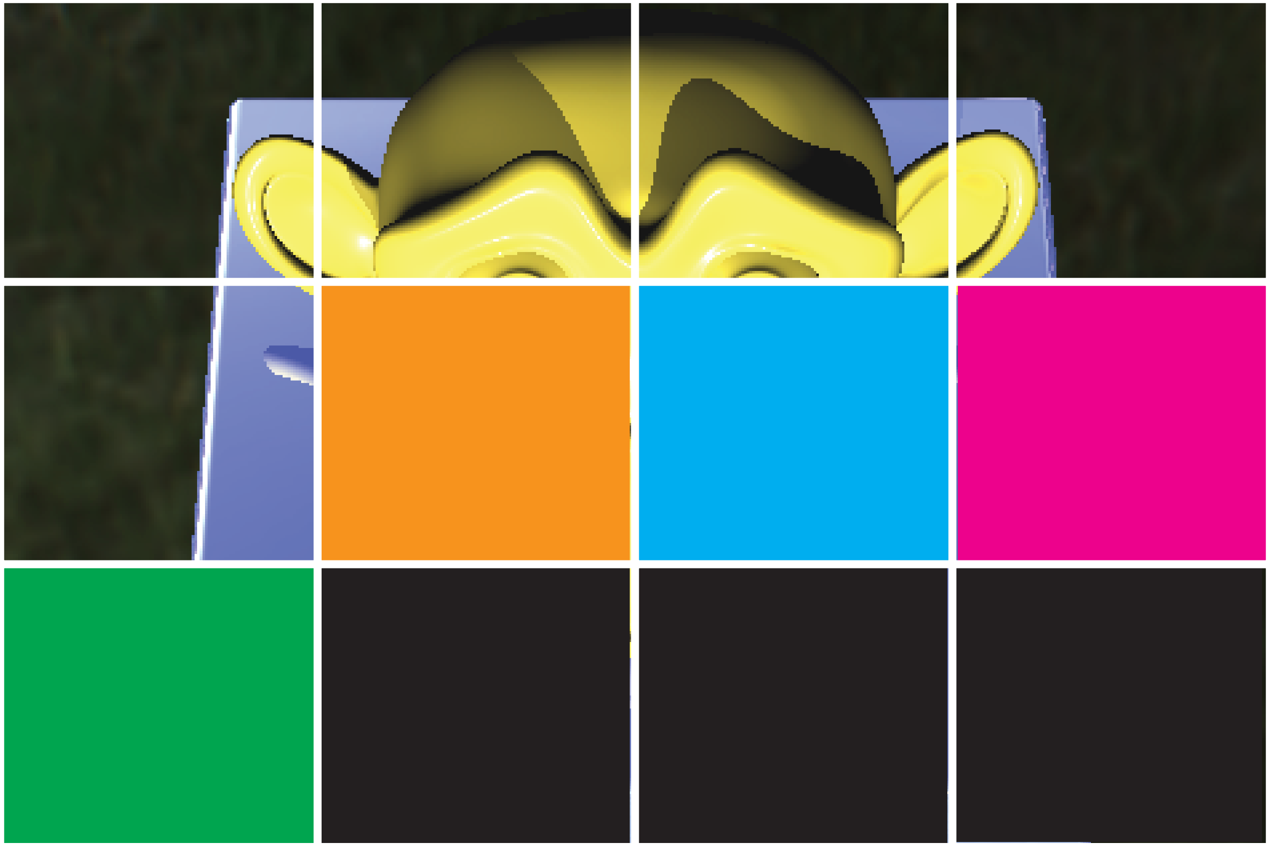


screen

Parallelization

Boost Threads

Implementation

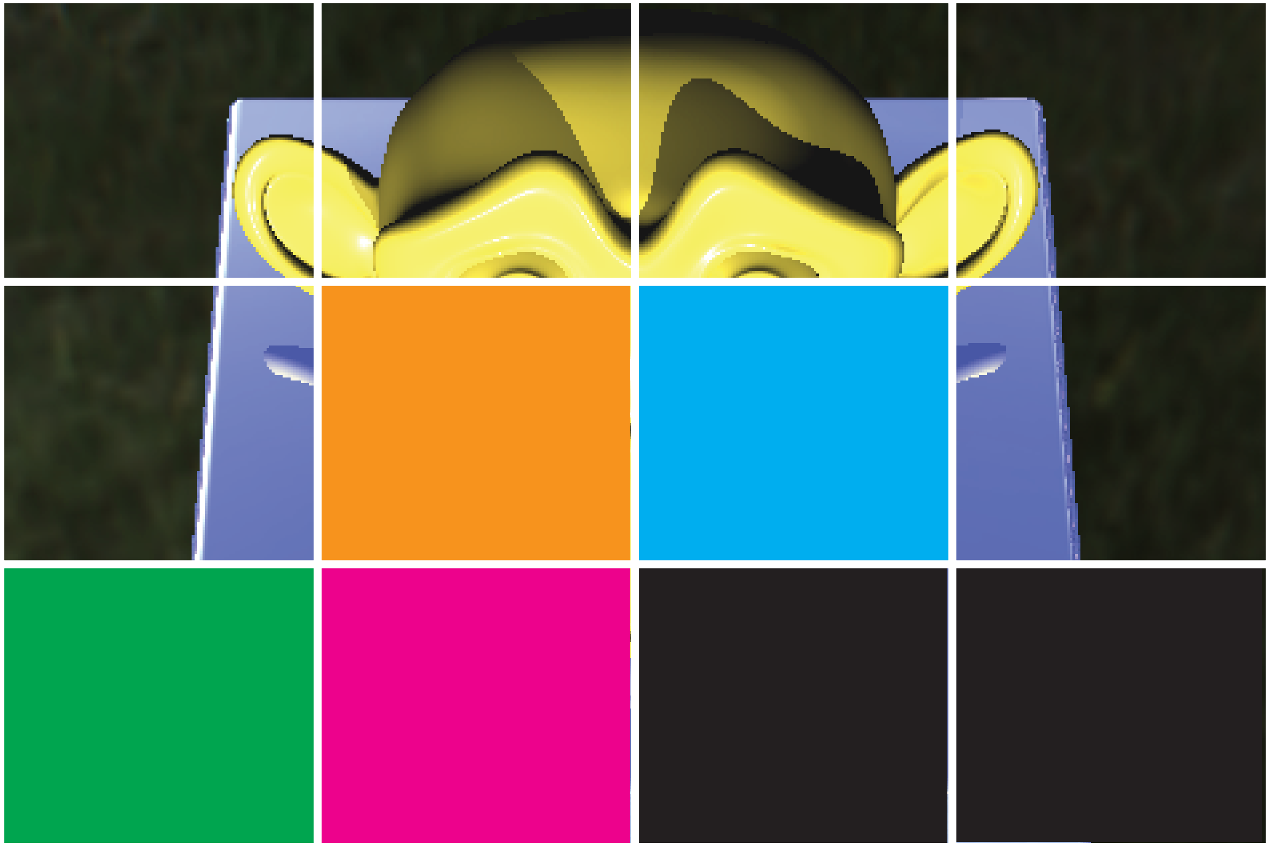


screen

Parallelization

Boost Threads

Implementation

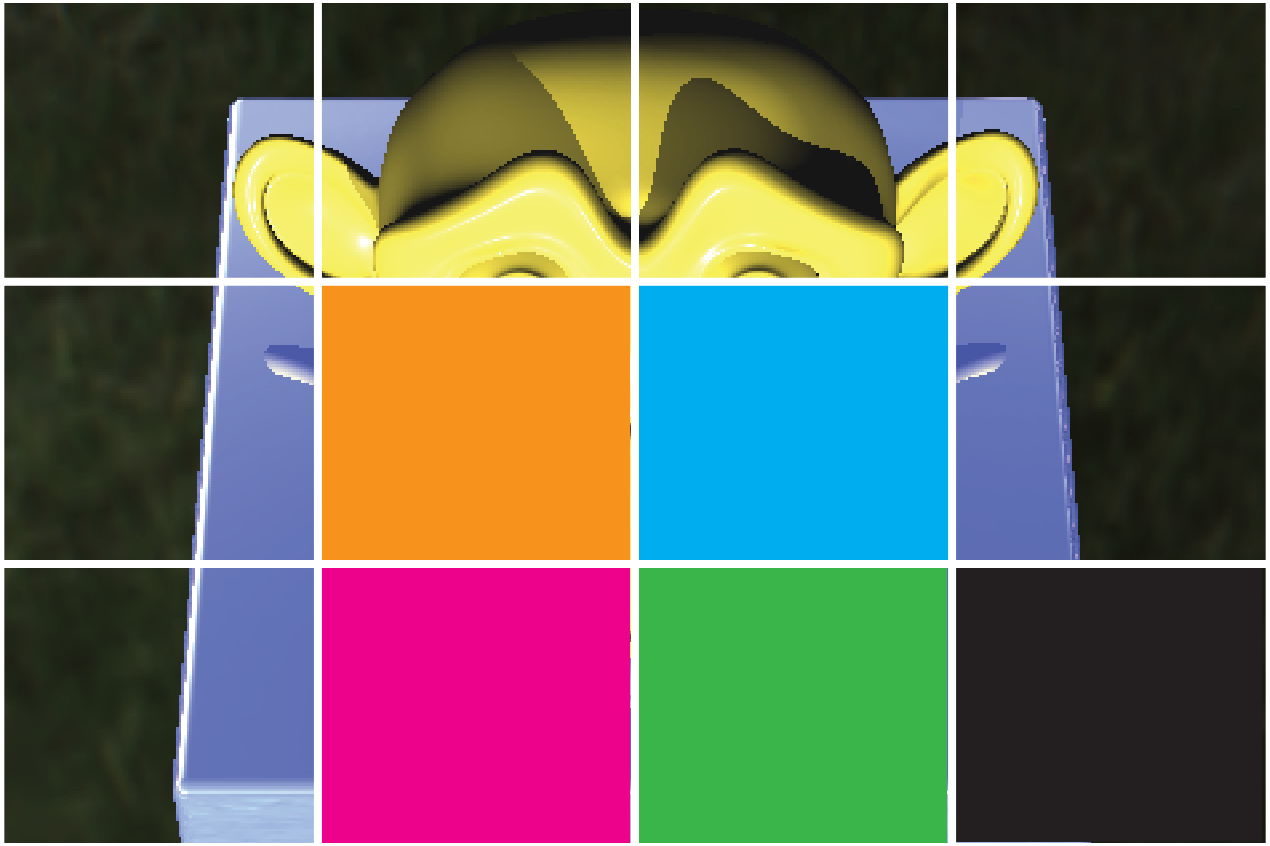


screen

Parallelization

Boost Threads

Implementation

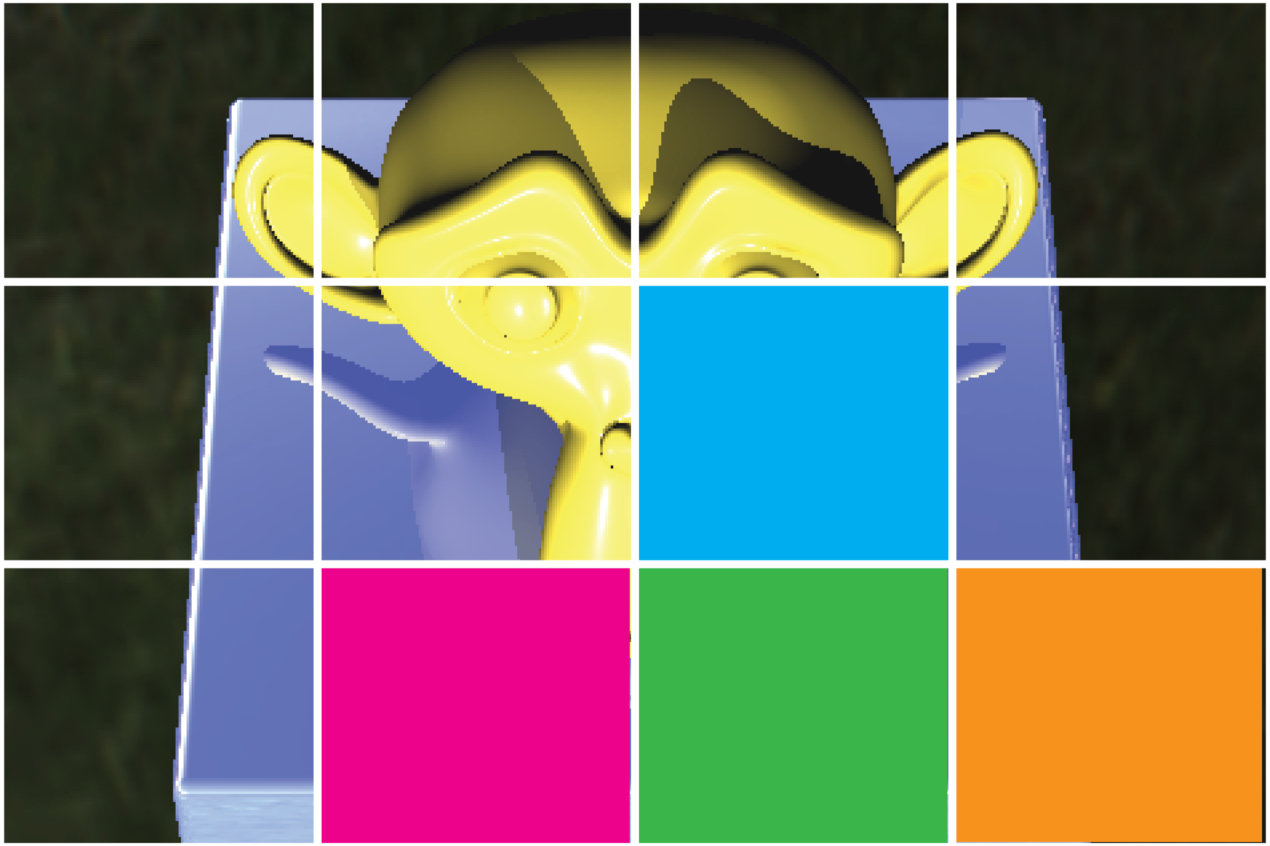


screen

Parallelization

Boost Threads

Implementation

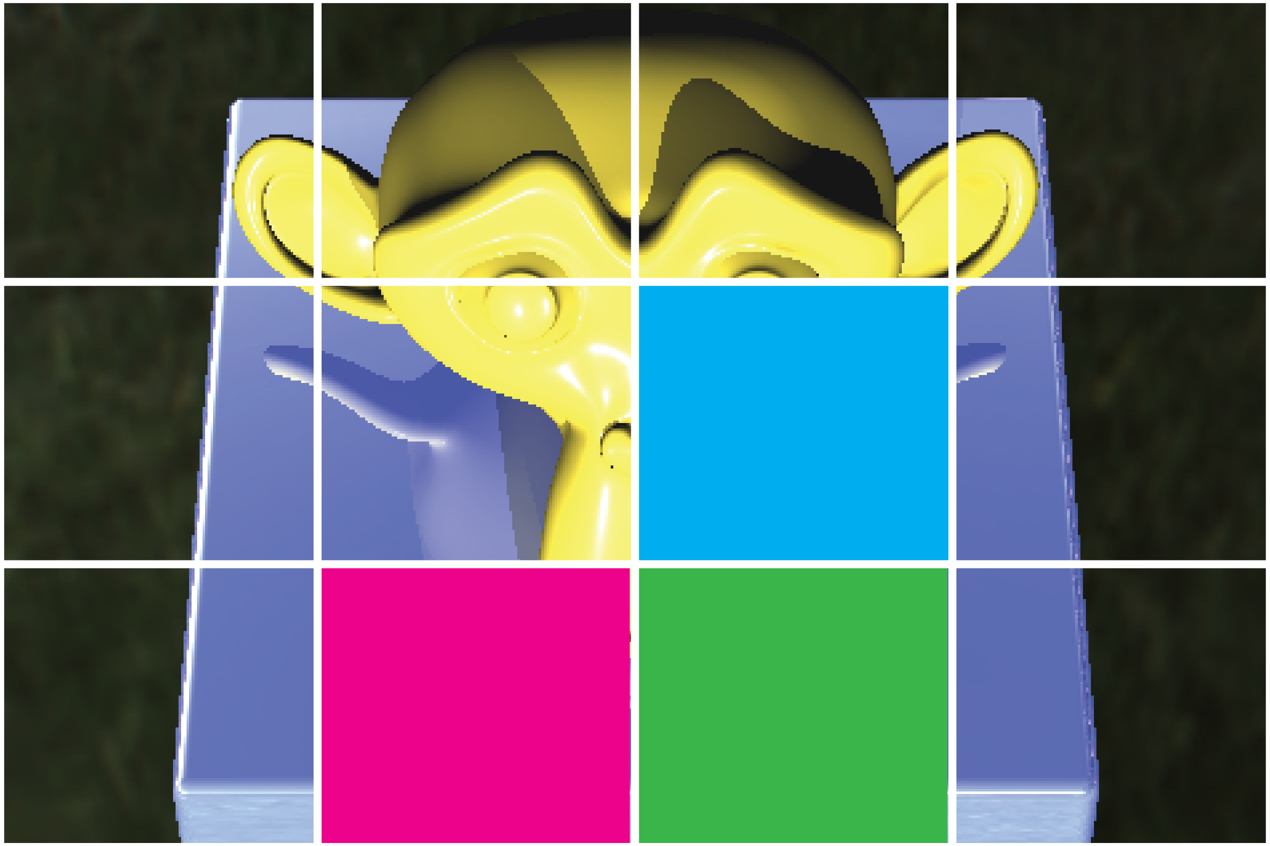


screen

Parallelization

Boost Threads

Implementation

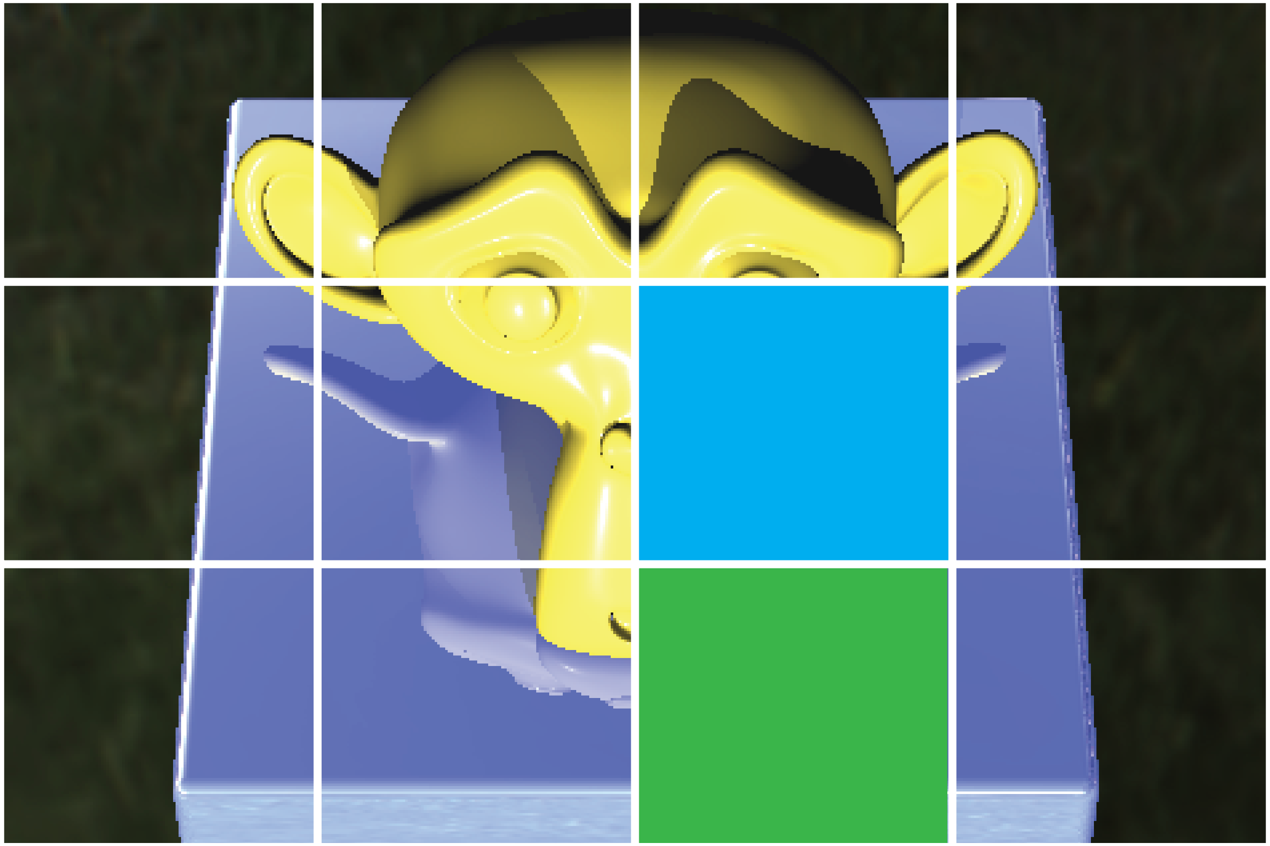


screen

Parallelization

Boost Threads

Implementation

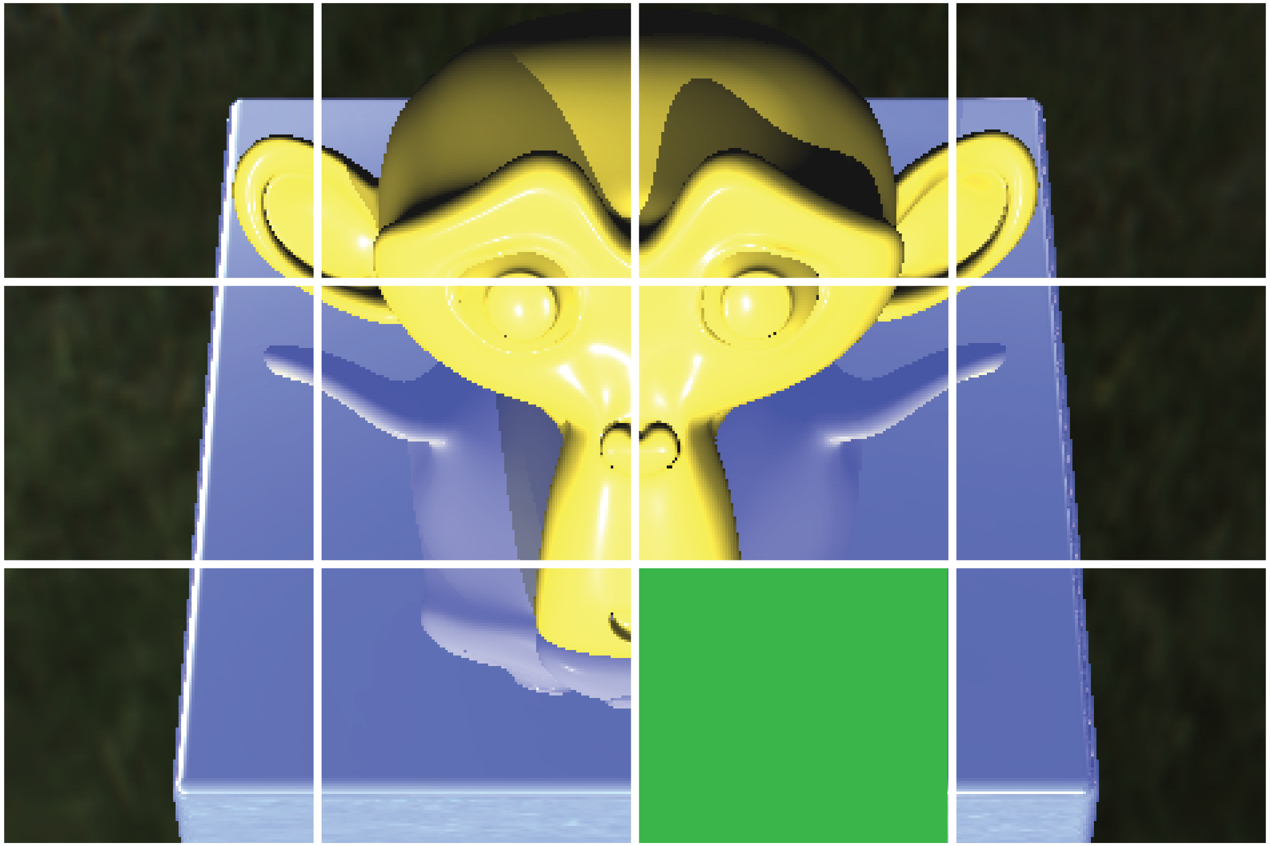


screen

Parallelization

Boost Threads

Implementation

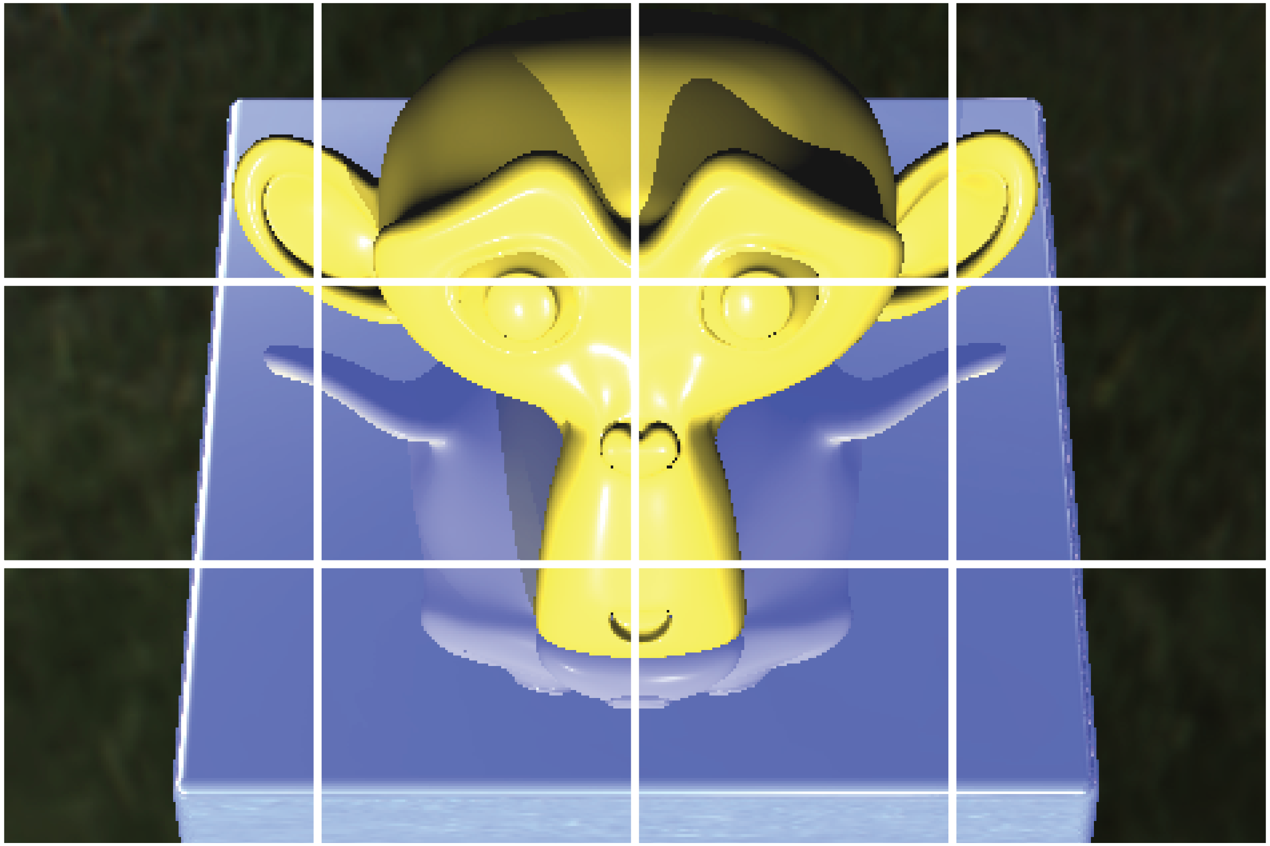


screen

Parallelization

Boost Threads

Implementation



screen

Parallelization

Boost Threads

Pros:

- full control over creation, synchronization and termination of threads
- less memory usage – only N (# of threads) RaytracerIterative objects are allocated

Parallelization

Intel Threading Building Blocks

ITBB – a high level threading library consisting of data structures and algorithms for task-based parallelism

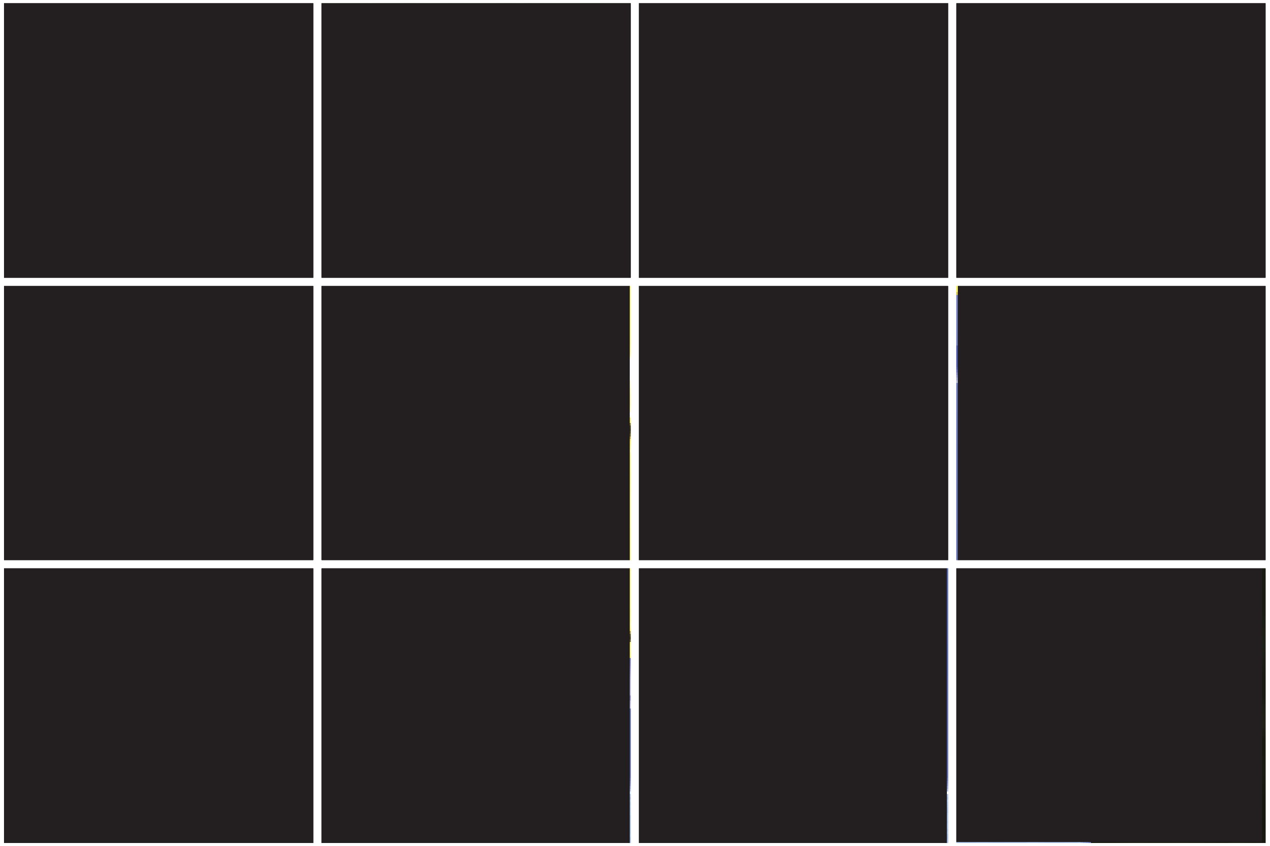
- abstracts platform details and threading mechanisms
- emphasizes data parallel programming
- not perfectly suited for our tile rendering approach
  - a typical ITBB application: Quicksort on big array

Parallelization

Intel Threading Building Blocks

Implementation

- partition screen space



screen

Parallelization

Intel Threading Building Blocks

Implementation

- partition screen space
- create root task

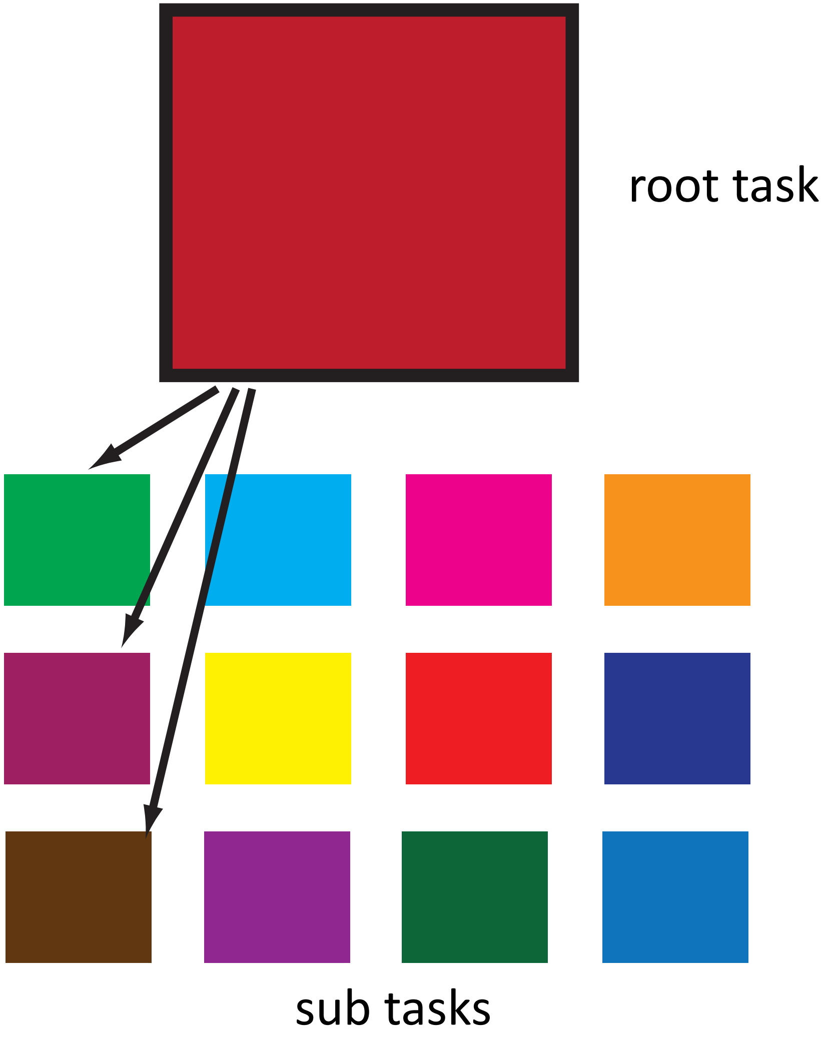
root task

Parallelization

Intel Threading Building Blocks

Implementation

- partition screen space
- create root task
- spawn sub tasks

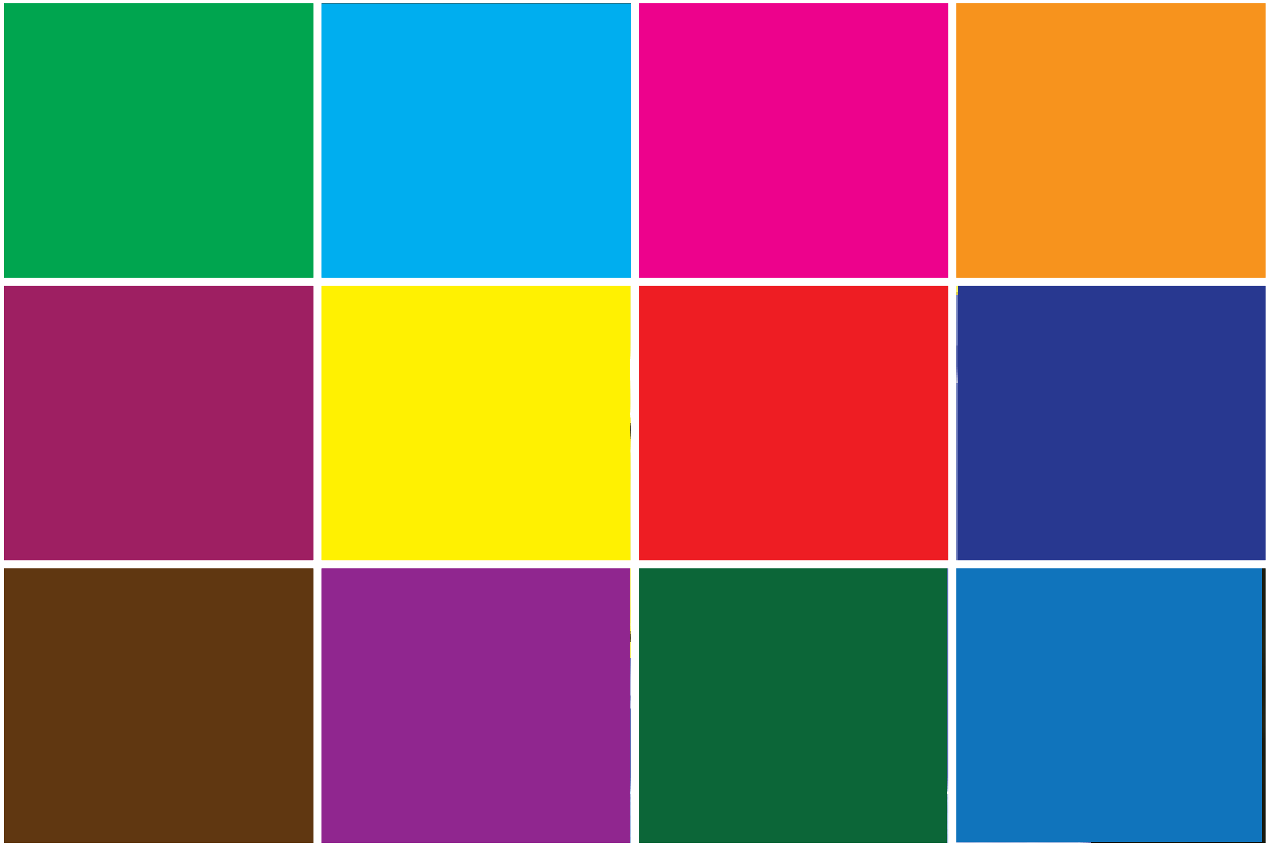


Parallelization

Intel Threading Building Blocks

Implementation

- partition screen space
- create root task
- spawn sub tasks
- root task waits for completion of sub tasks (scheduling is handled by ITTB)



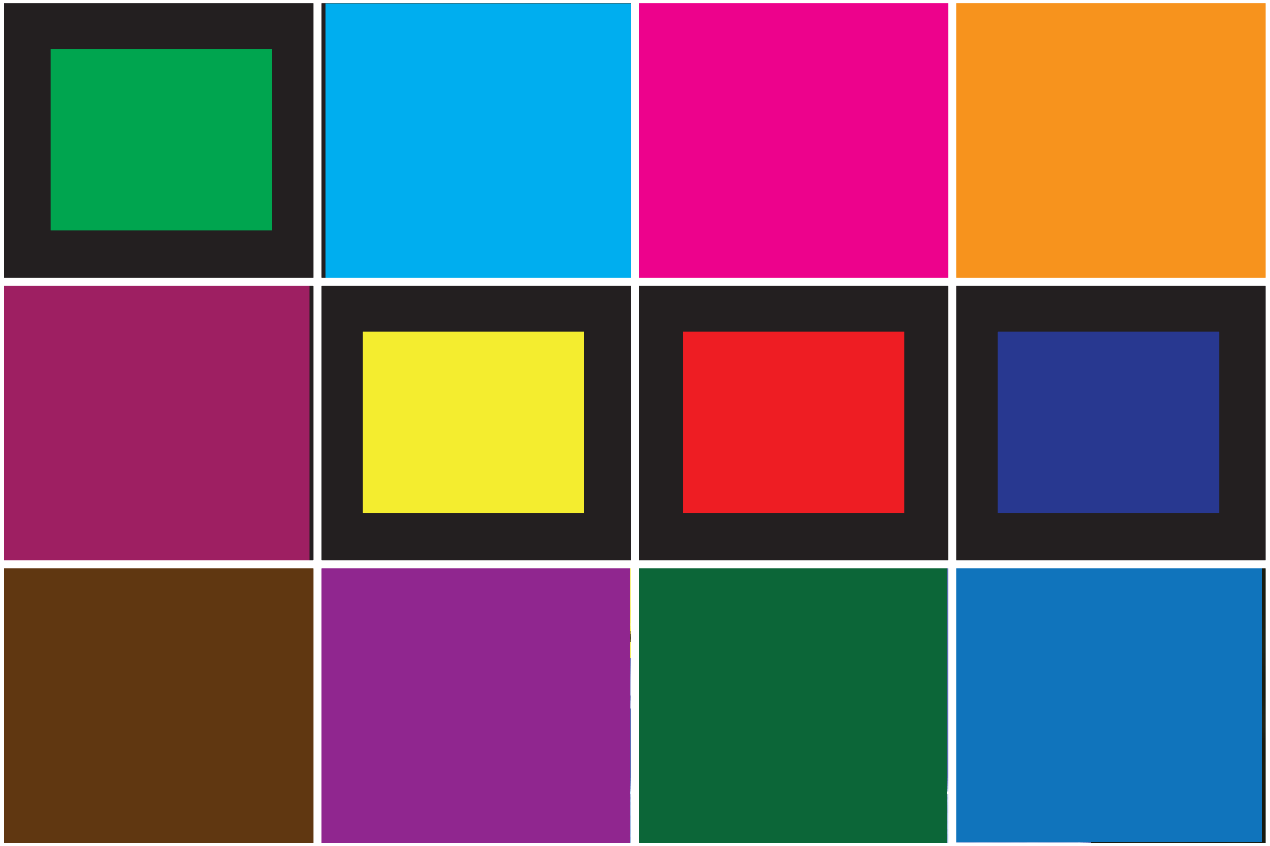
screen

Parallelization

Intel Threading Building Blocks

Implementation

- partition screen space
- create root task
- spawn sub tasks
- root task waits for completion of sub tasks (scheduling is handled by ITTB)



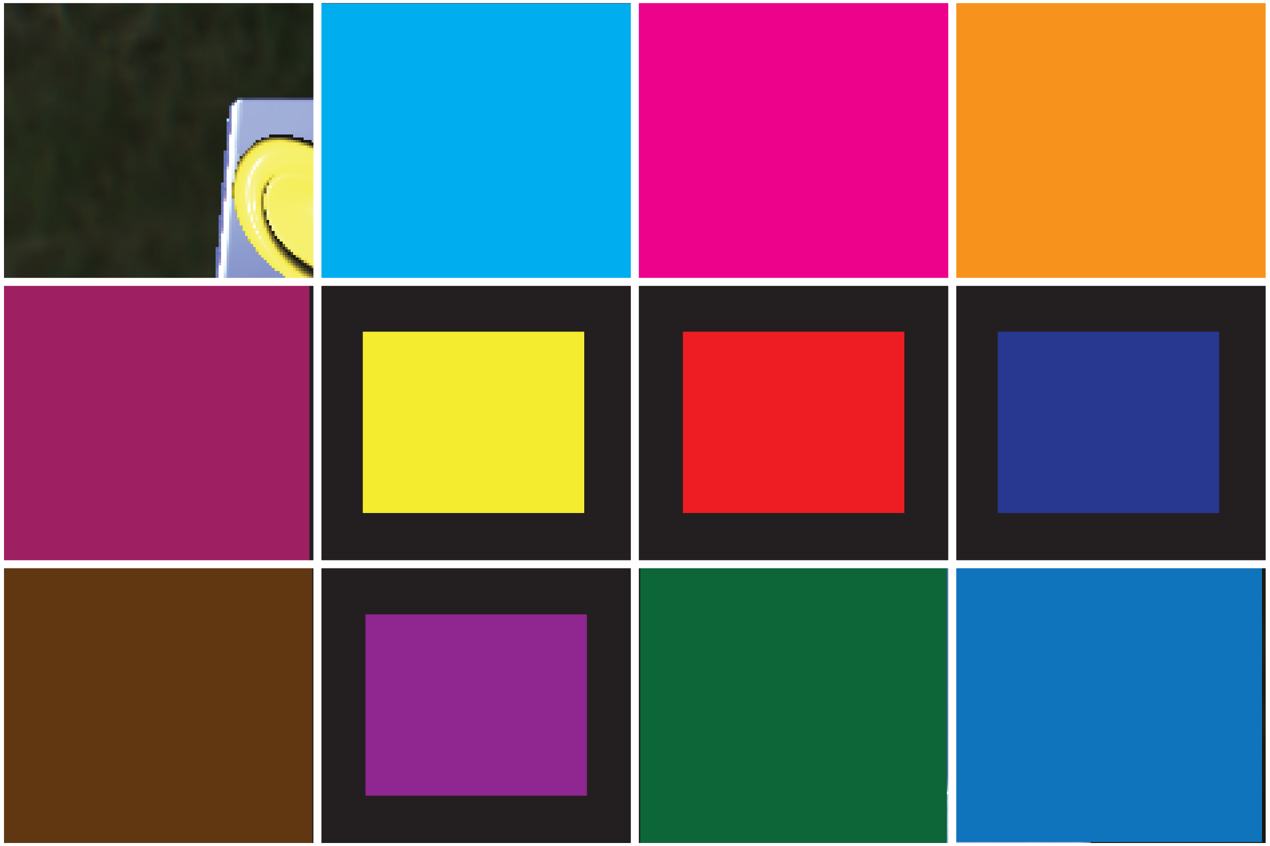
screen

Parallelization

Intel Threading Building Blocks

Implementation

- partition screen space
- create root task
- spawn sub tasks
- root task waits for completion of sub tasks (scheduling is handled by ITTB)



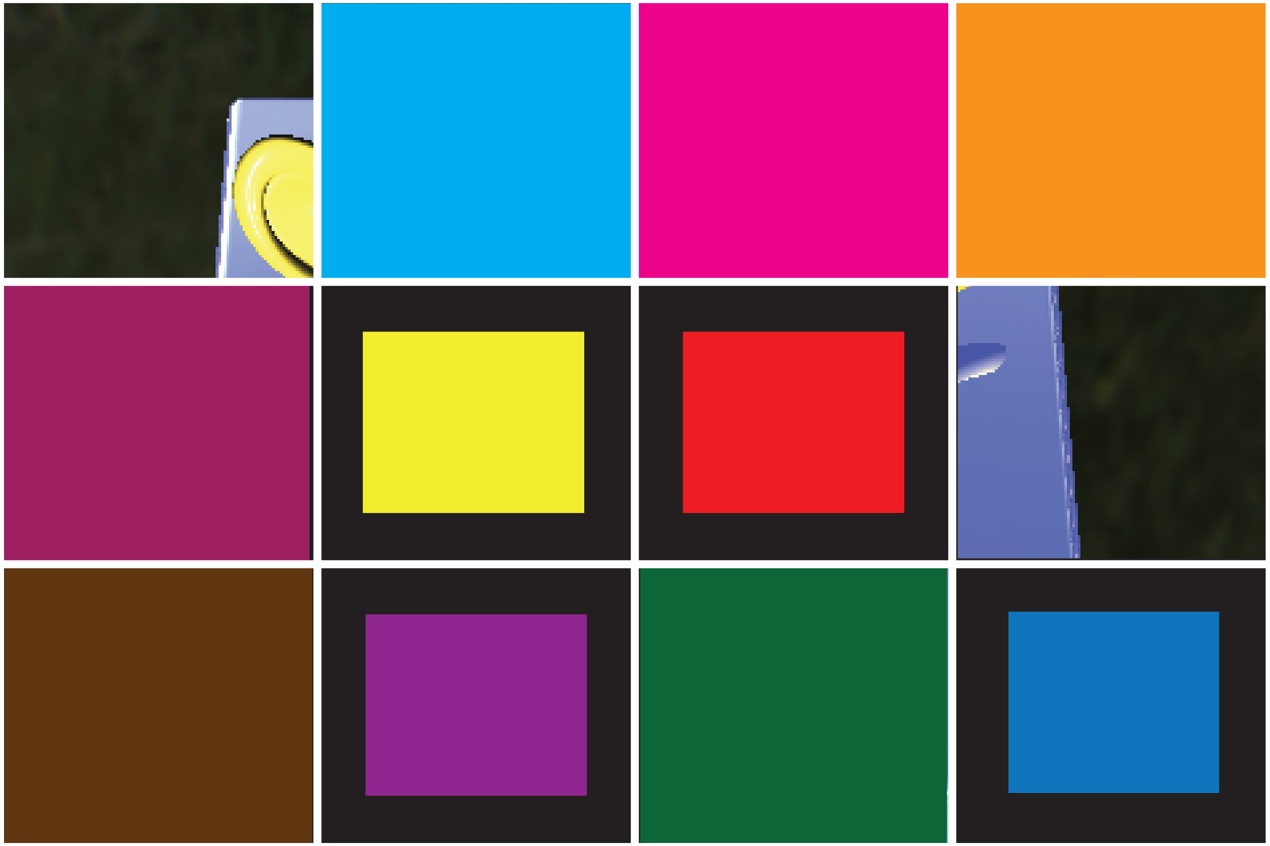
screen

Parallelization

Intelligent Threading Building Blocks

Implementation

- partition screen space
- create root task
- spawn sub tasks
- root task waits for completion of sub tasks (scheduling is handled by ITTB)

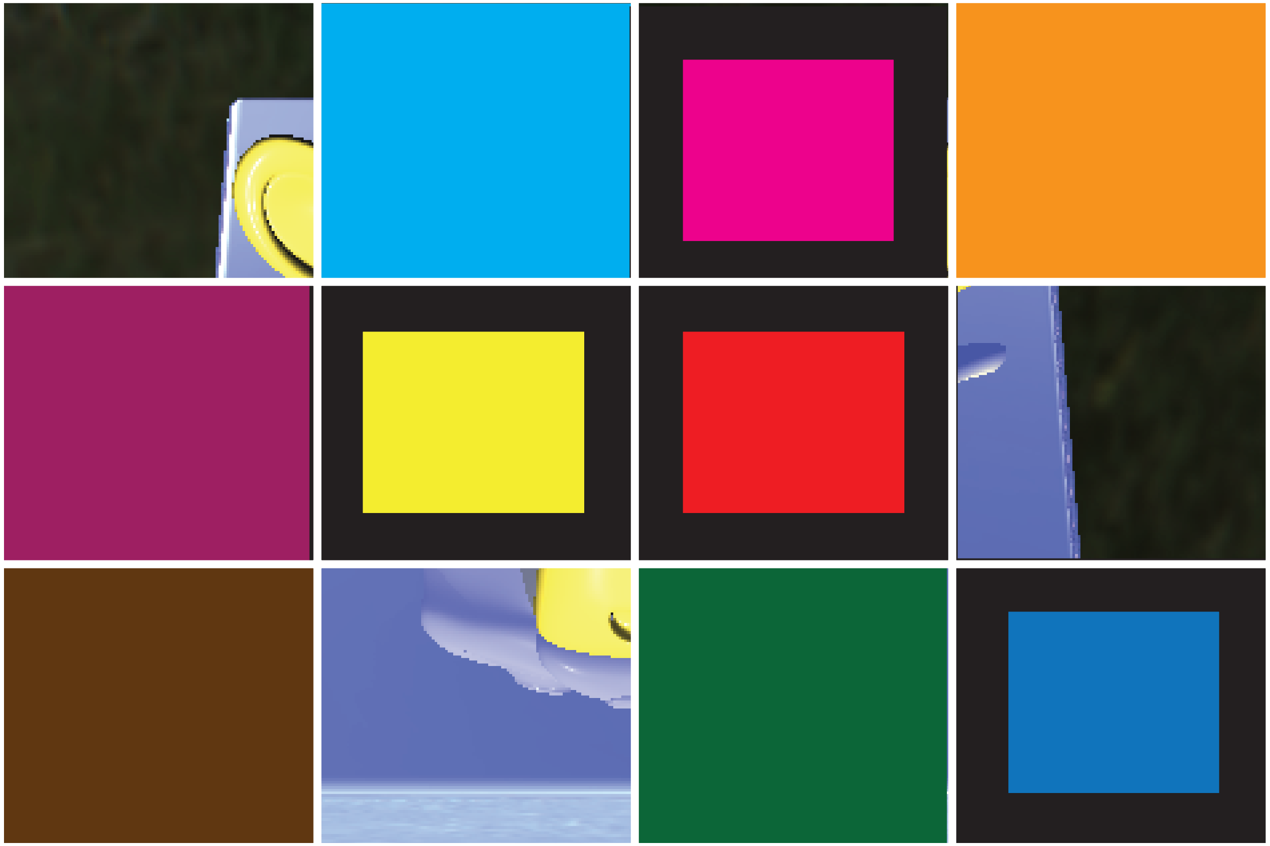


Parallelization

Intelligent Threading Building Blocks

Implementation

- partition screen space
- create root task
- spawn sub tasks
- root task waits for completion of sub tasks (scheduling is handled by ITTB)



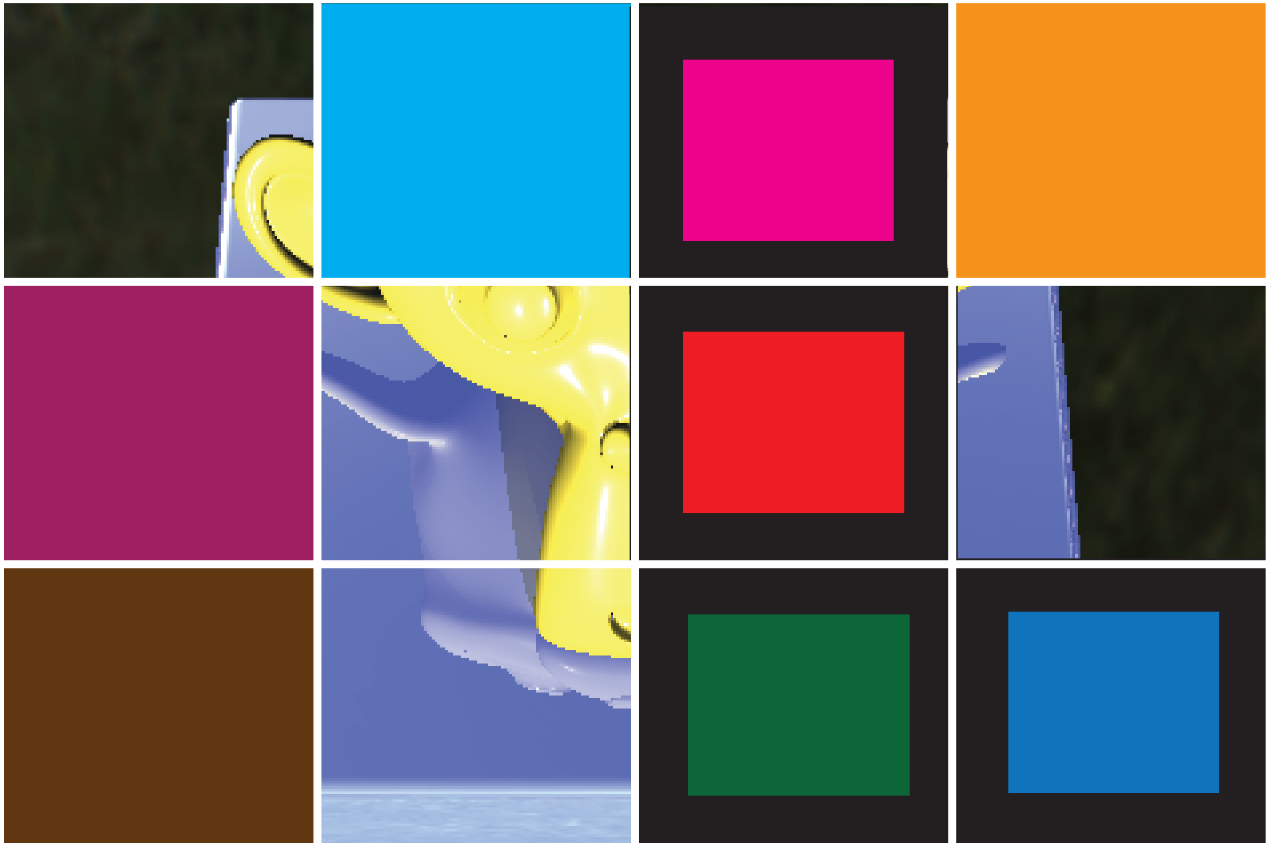
screen

Parallelization

Intel Threading Building Blocks

Implementation

- partition screen space
- create root task
- spawn sub tasks
- root task waits for completion of sub tasks (scheduling is handled by ITTB)



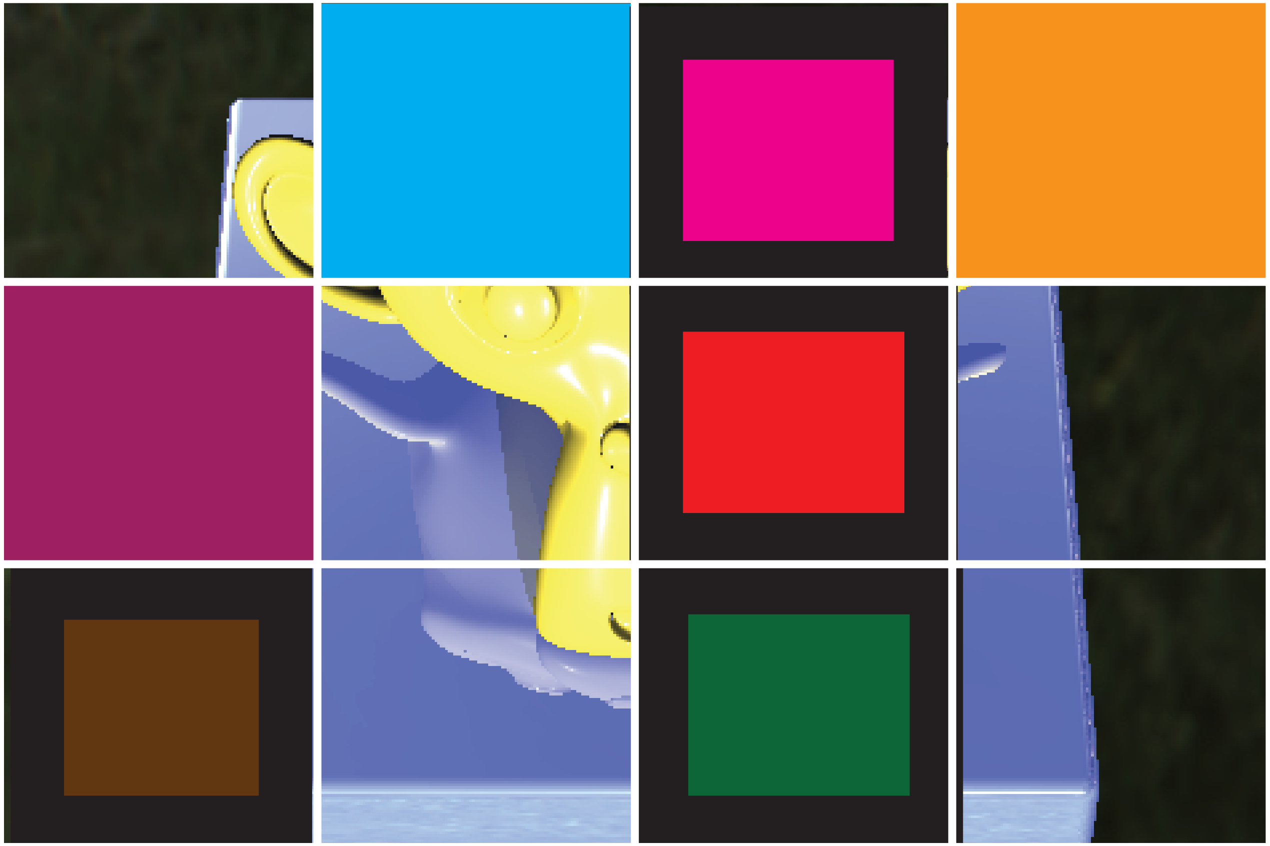
screen

Parallelization

Intel Threading Building Blocks

Implementation

- partition screen space
- create root task
- spawn sub tasks
- root task waits for completion of sub tasks (scheduling is handled by ITTB)



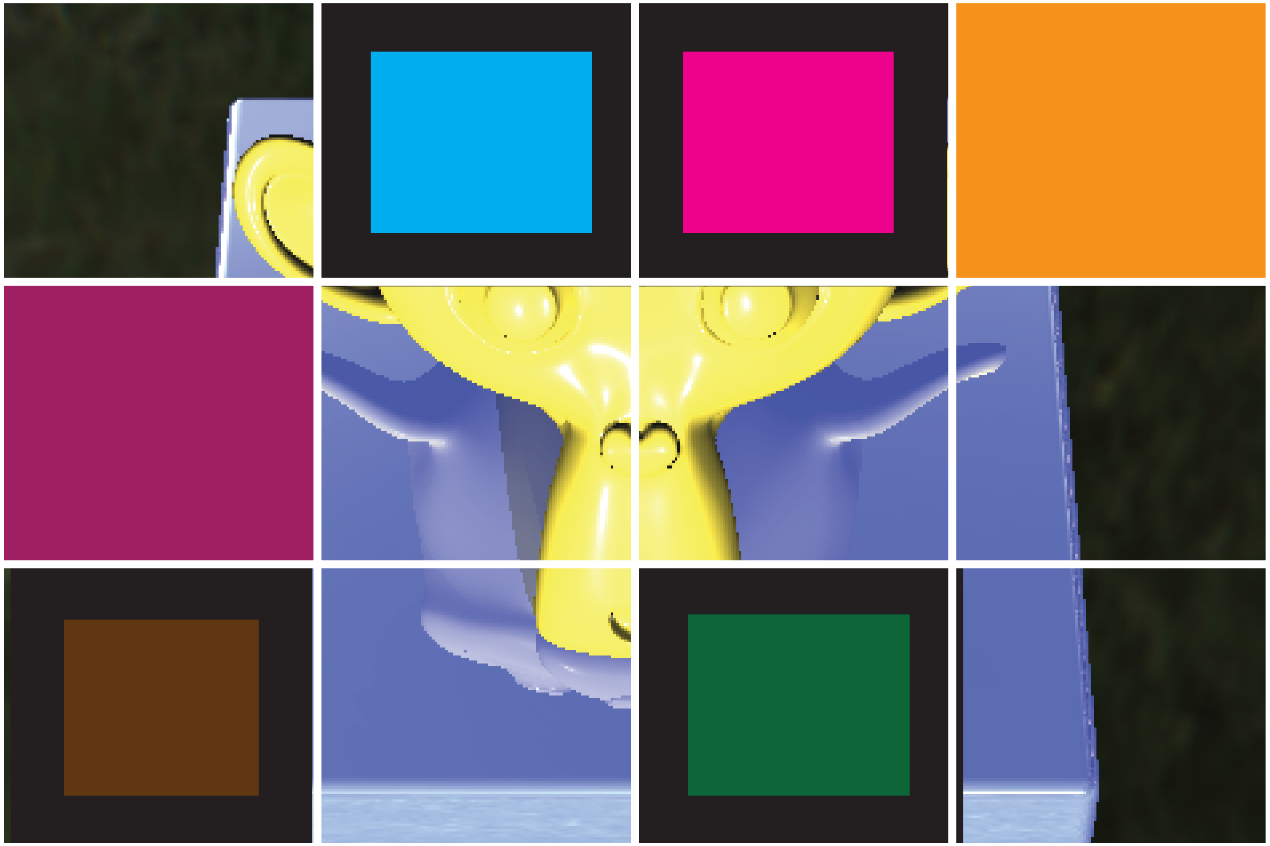
screen

Parallelization

Intel Threading Building Blocks

Implementation

- partition screen space
- create root task
- spawn sub tasks
- root task waits for completion of sub tasks (scheduling is handled by ITTB)



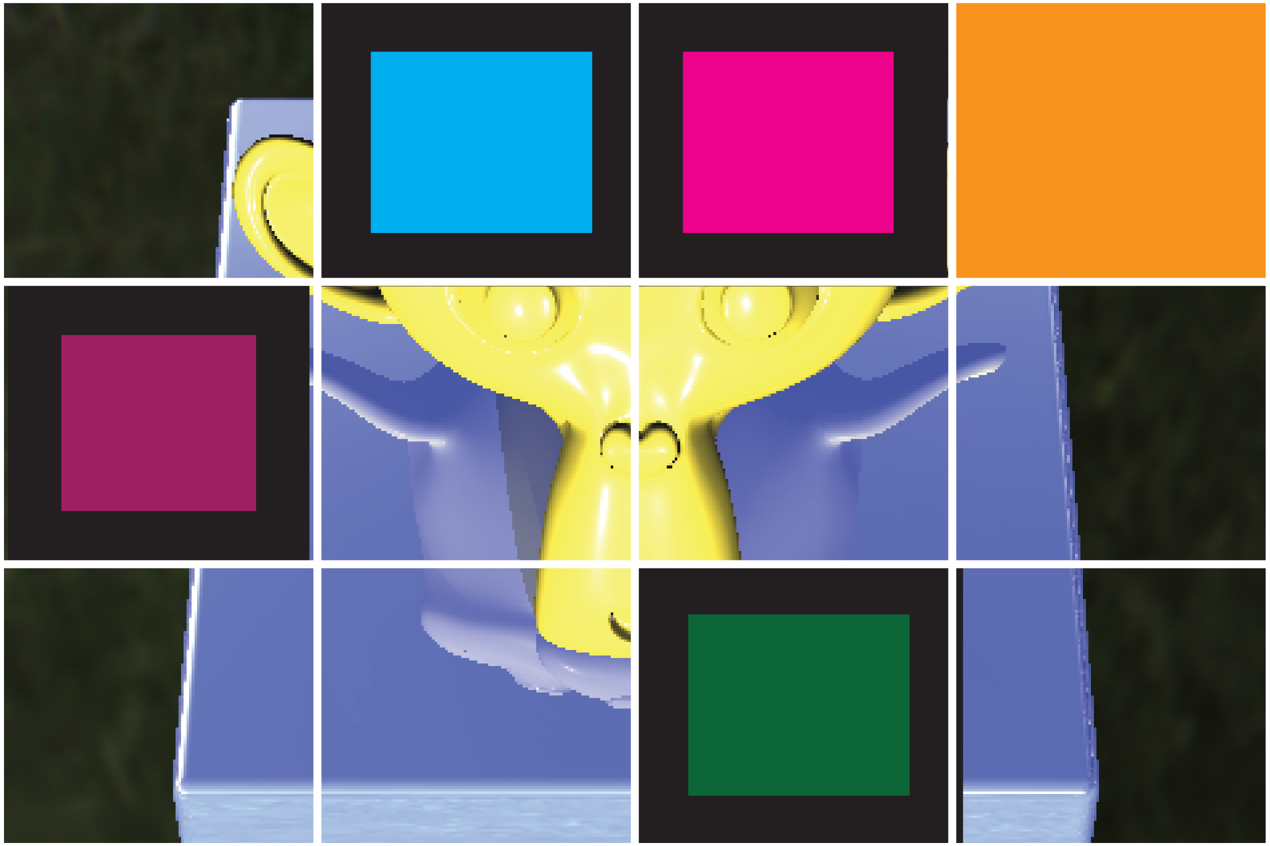
screen

Parallelization

Intel Threading Building Blocks

Implementation

- partition screen space
- create root task
- spawn sub tasks
- root task waits for completion of sub tasks (scheduling is handled by ITTB)



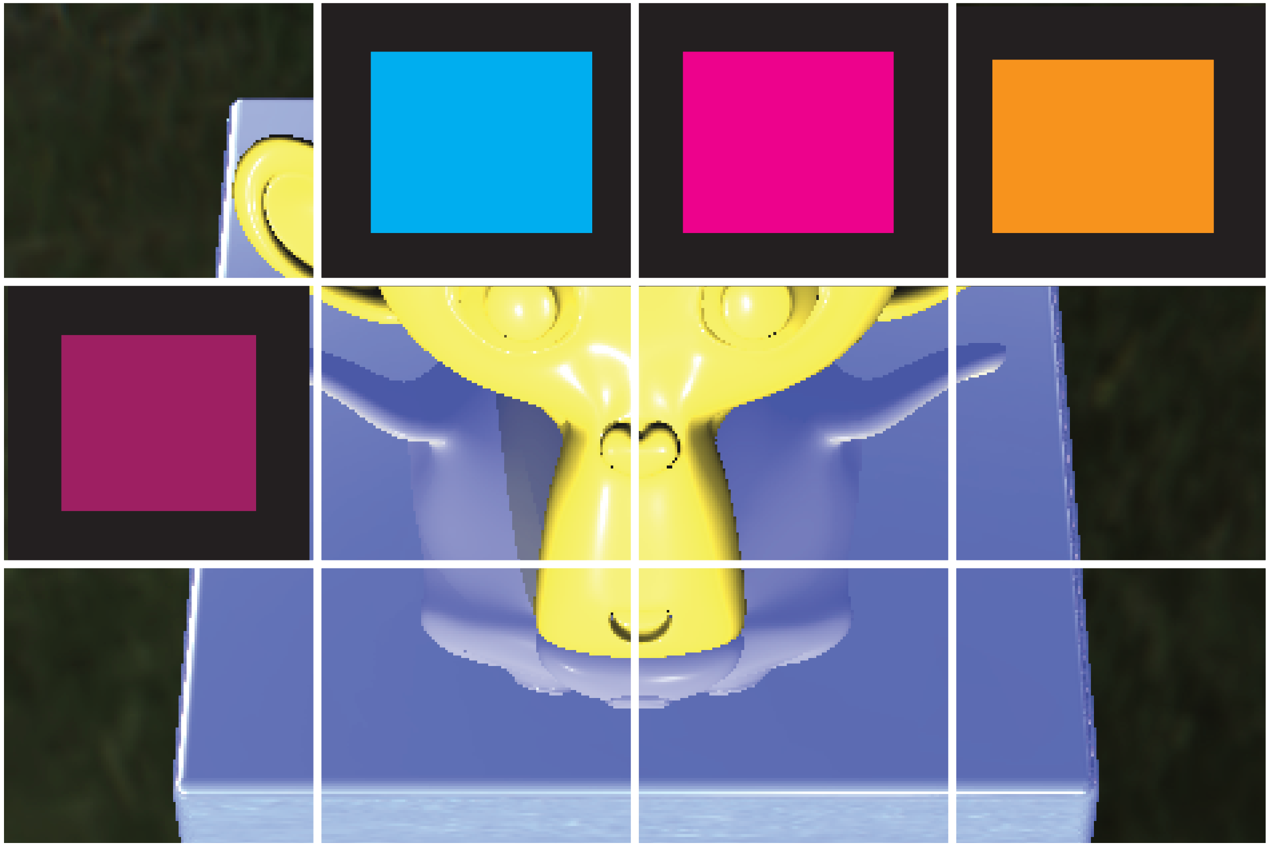
screen

Parallelization

Intel Threading Building Blocks

Implementation

- partition screen space
- create root task
- spawn sub tasks
- root task waits for completion of sub tasks (scheduling is handled by ITTB)



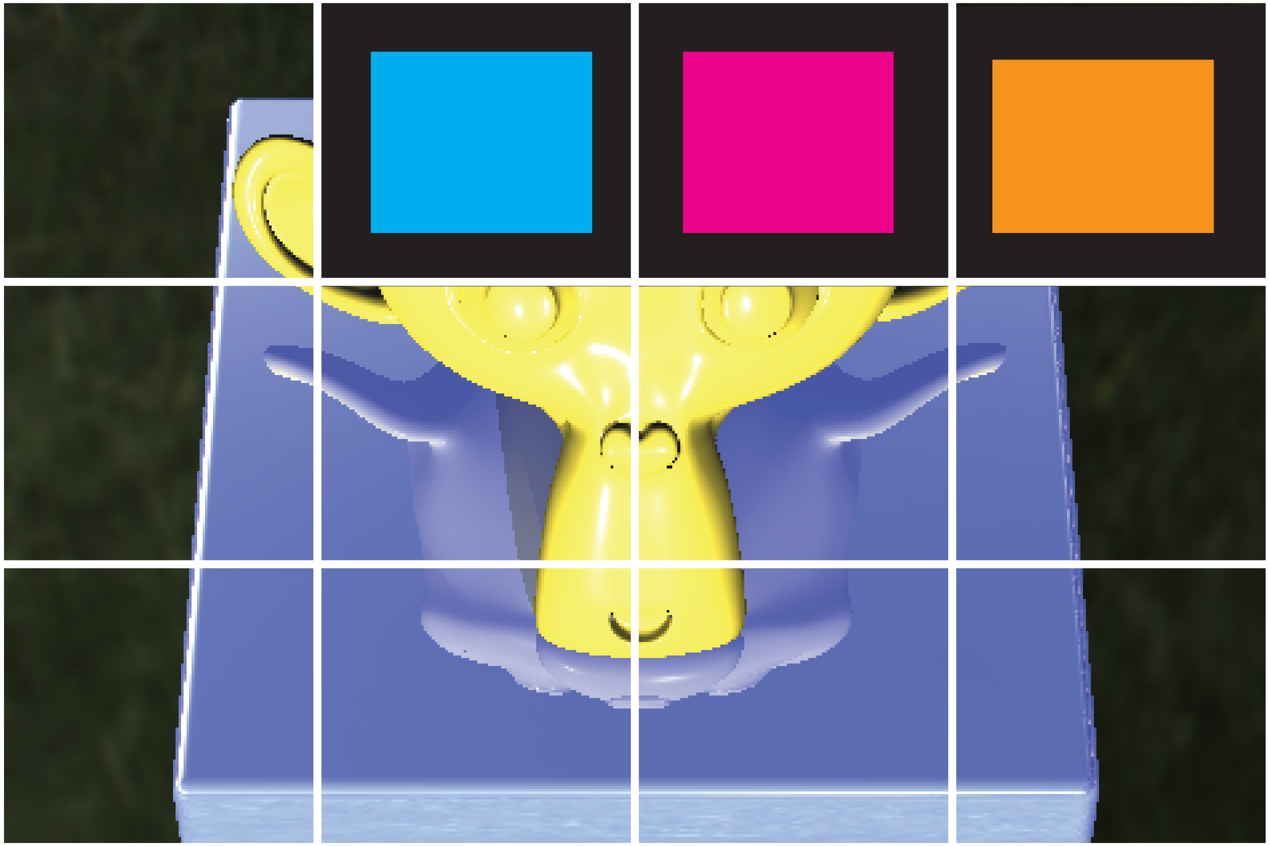
screen

Parallelization

Intelligent Threading Building Blocks

Implementation

- partition screen space
- create root task
- spawn sub tasks
- root task waits for completion of sub tasks (scheduling is handled by ITTB)



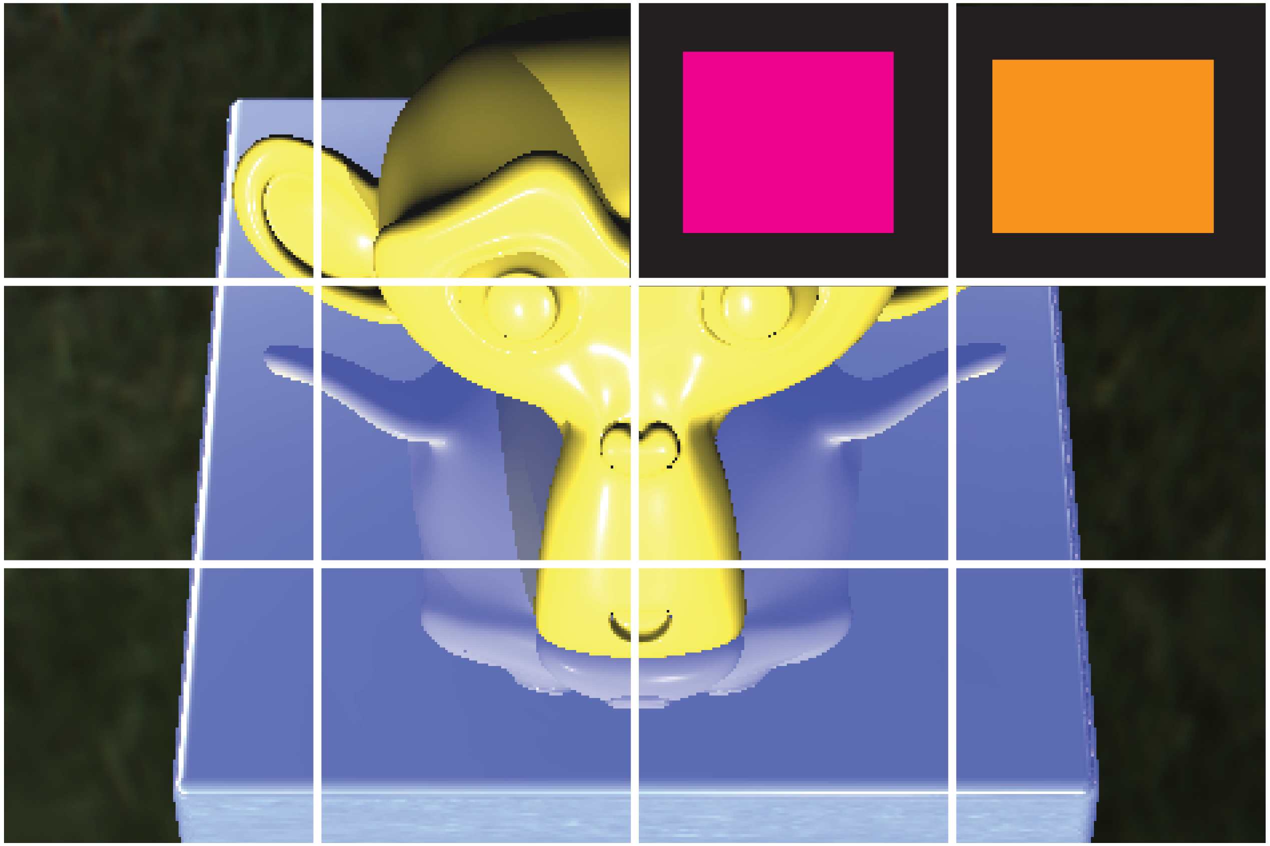
screen

Parallelization

Intel Threading Building Blocks

Implementation

- partition screen space
- create root task
- spawn sub tasks
- root task waits for completion of sub tasks (scheduling is handled by ITTB)



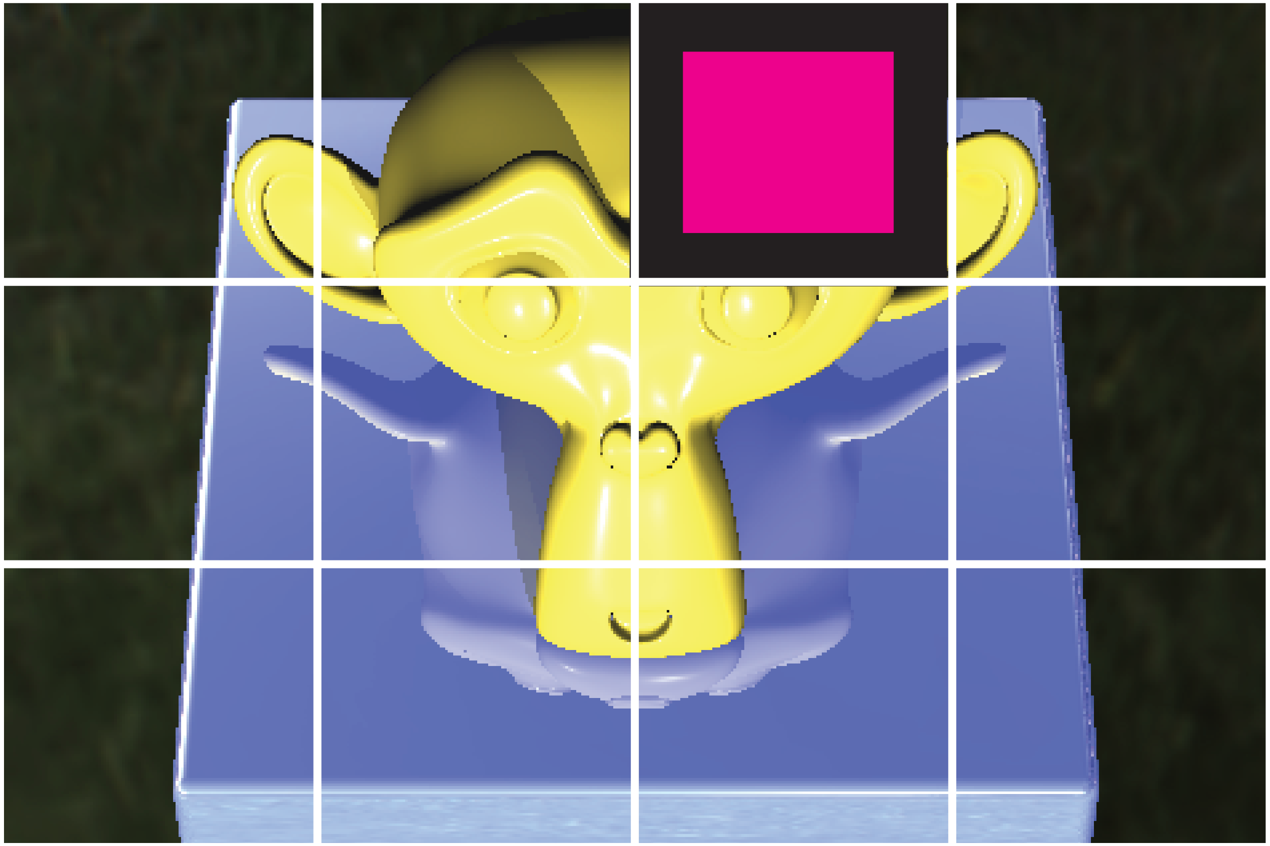
screen

Parallelization

Intel Threading Building Blocks

Implementation

- partition screen space
- create root task
- spawn sub tasks
- root task waits for completion of sub tasks (scheduling is handled by ITTB)



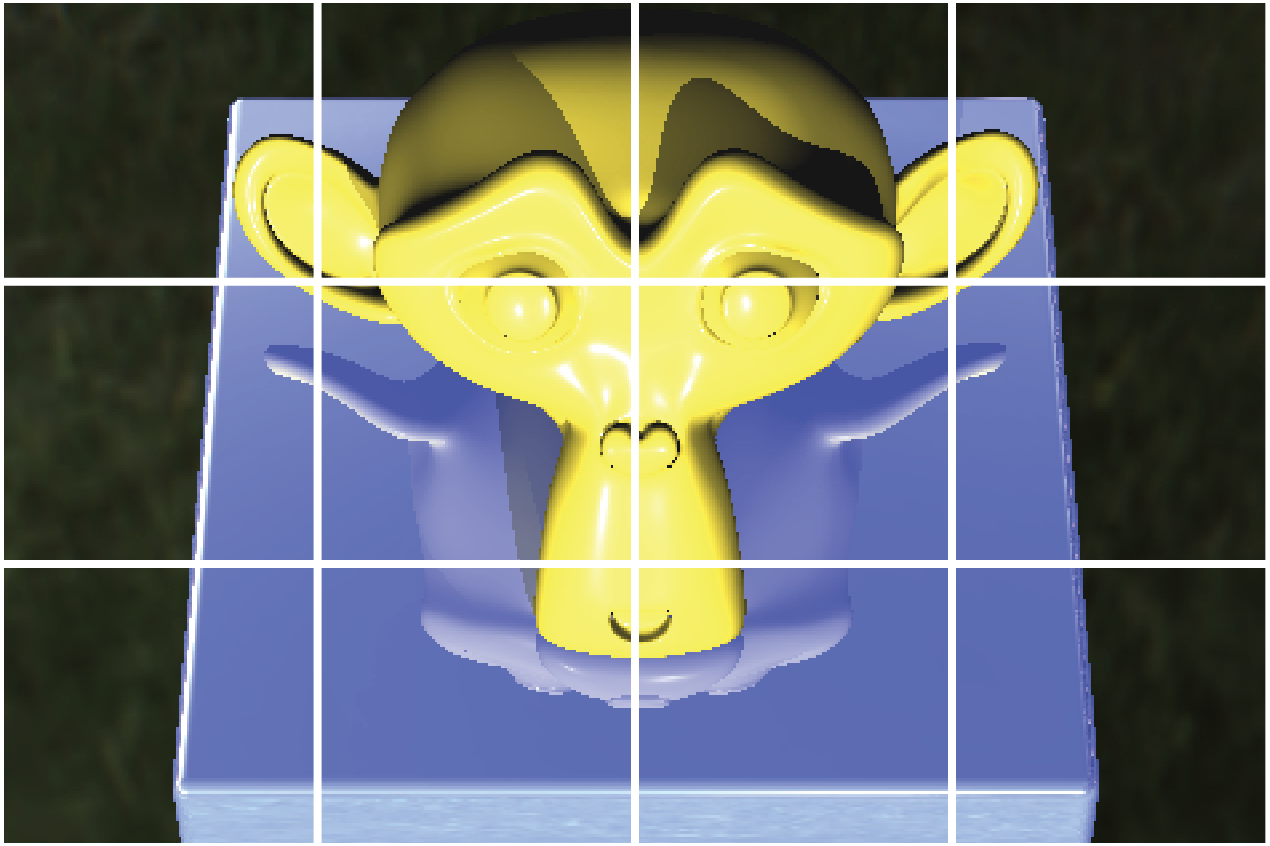
screen

Parallelization

Intel Threading Building Blocks

Implementation

- partition screen space
- create root task
- spawn sub tasks
- root task waits for completion of sub tasks (scheduling is handled by ITTB)



screen

Parallelization

Intel Threading Building Blocks

Pros:

- hides thread handling details
- tasks are allocated to individual cores dynamically by ITBB' s runtime-engine
- ITBB claims to automate the efficient use of the cache

Cons:

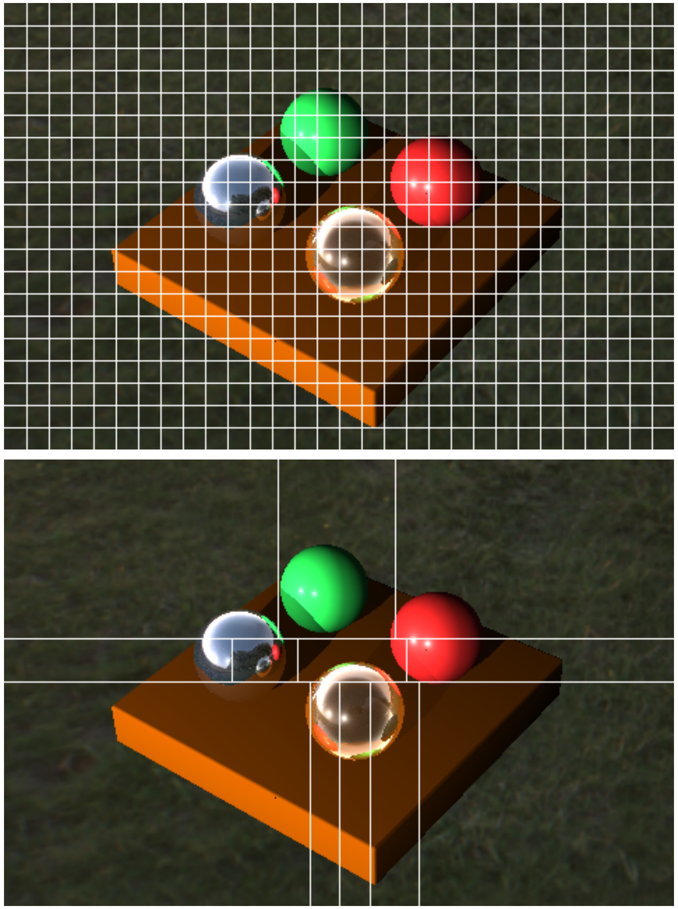
- back to the huge memory footprint –

  we allocate M (# of tiles) RaytracerIterative objects

In this approach there is not much difference regarding performance to the Boost Threadpool version.

Parallelization

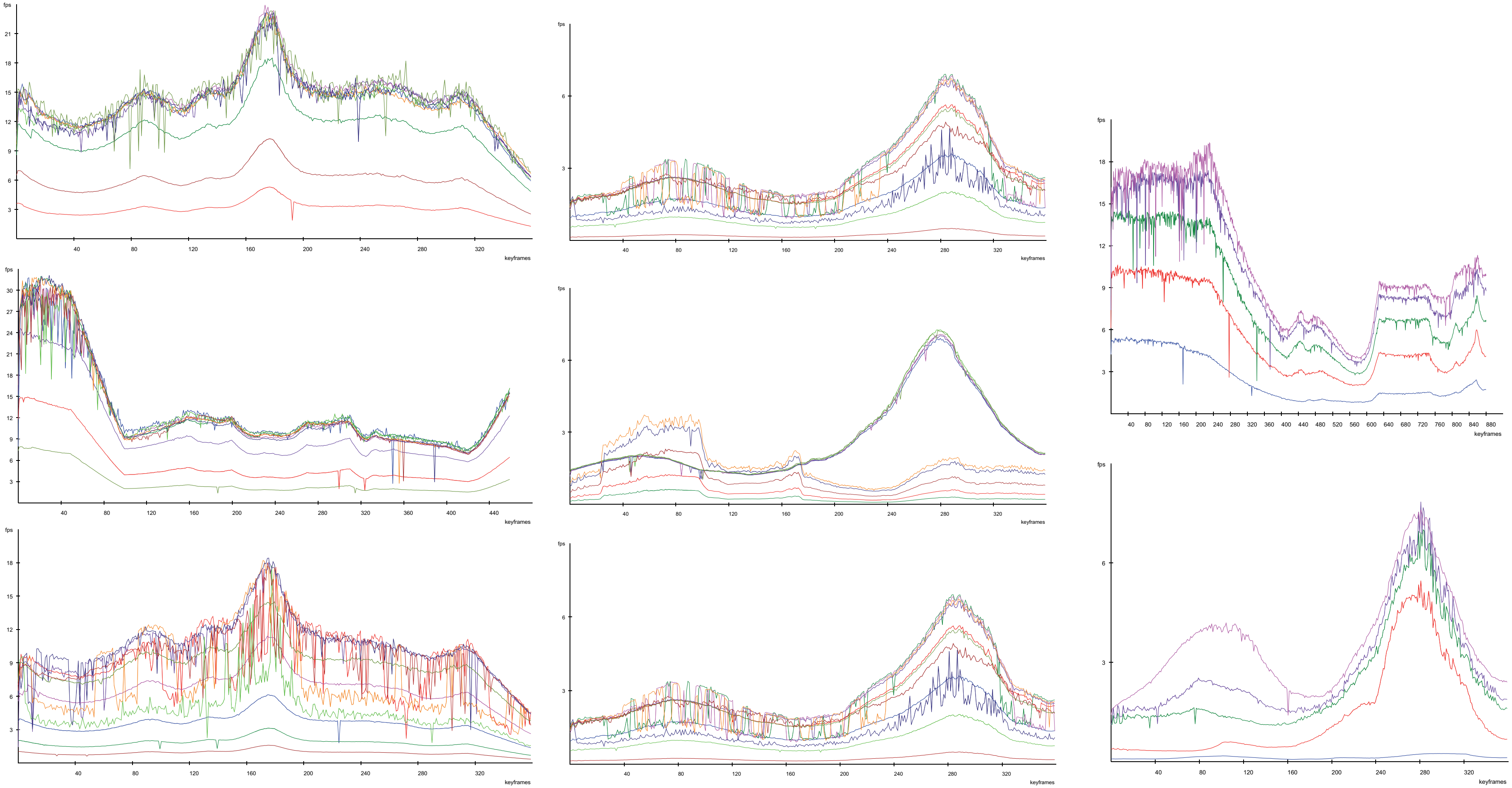
Dynamic Boost Threads



- load balancing
  - balancing the load among the tiles
- compute each tile load (rendering time)
- move horizontal/vertical edges
- GOAL: each tile load equal to the overall frame time over the number of threads

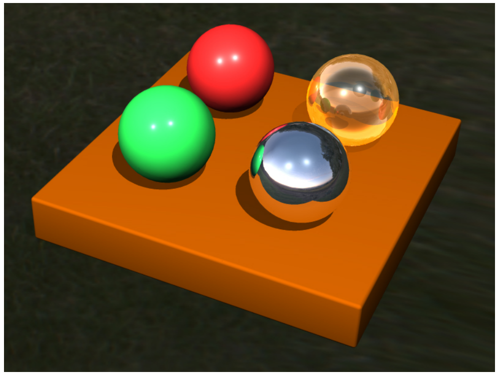
Results

Logging and Plotting



Results

Objects



name	four spheres	teapot high	lucy
triangles	5.452	16.896	78.870
vertices	2.736	8.448	39.437
normals	2.728	8.448	39.437

Results

Objects

name	**conference**	**clio all**
triangles	282.094	348.397
vertices	166.817	185.132
normals	108.181	155.829

Results

Configuration

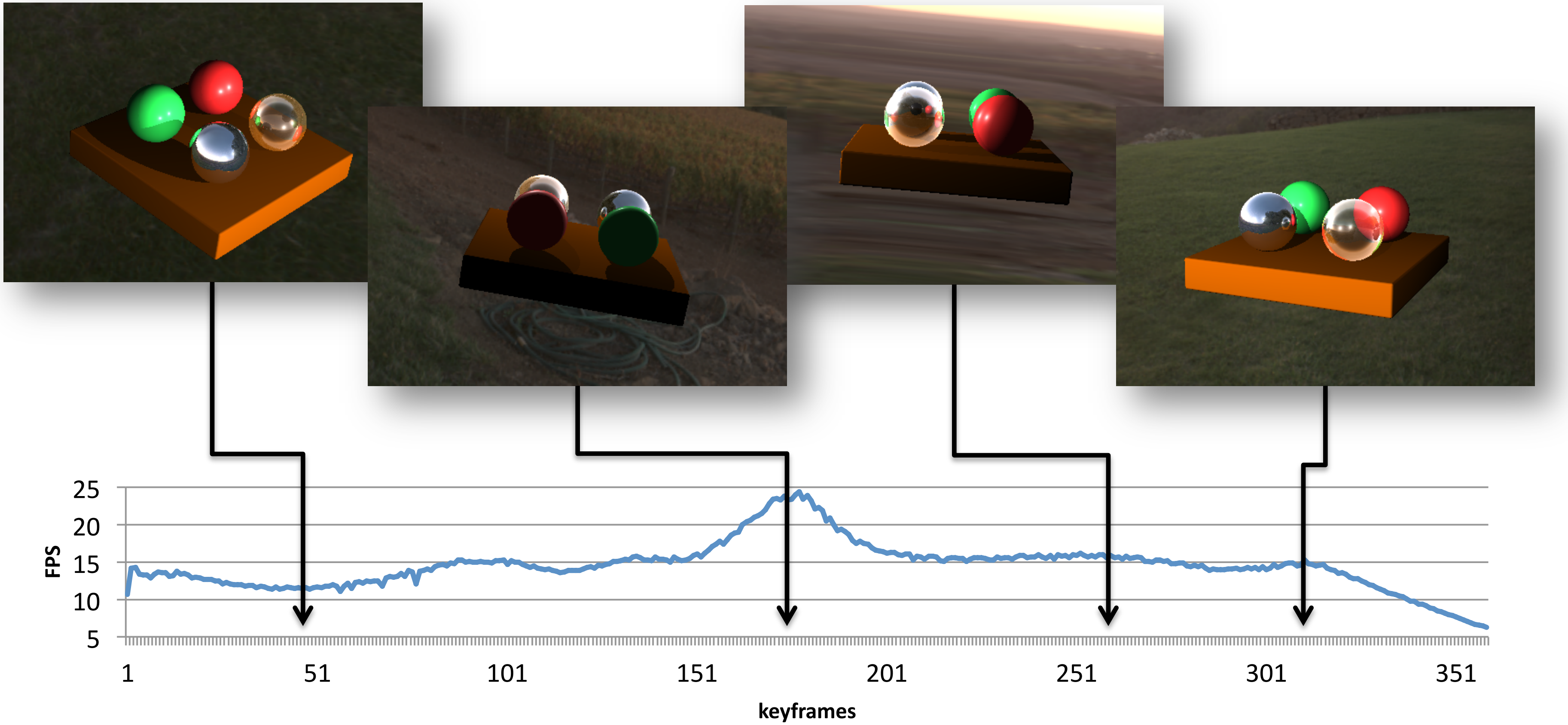
- Resolution: 800 x 600
- Tilesize: 16 x 16

- Ubuntu 9.04
- Memory: 46,7 GB
- Processor: 2 x Intel® Xeon® X5680, 3.33 GHz, 6 Cores

Results

Animationpath

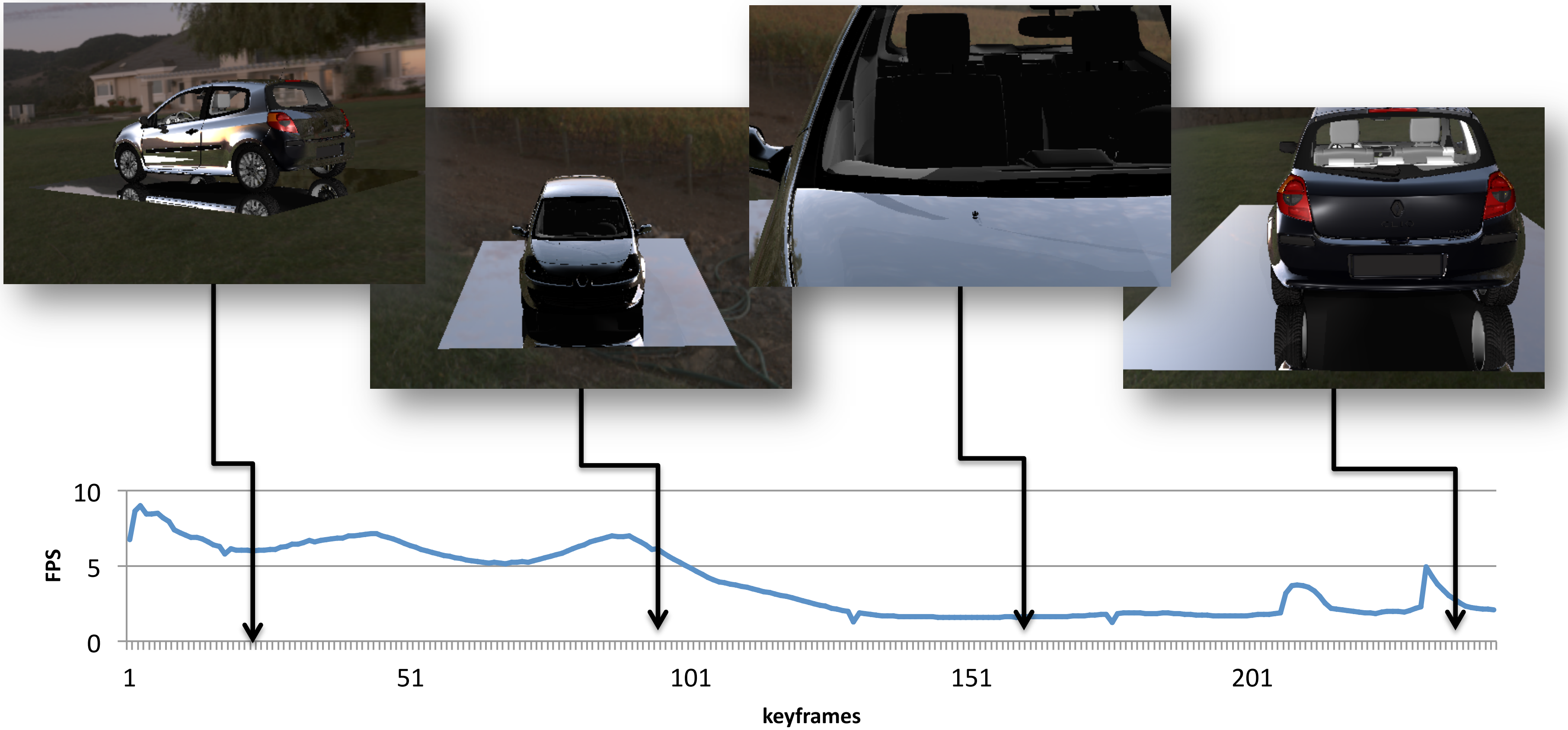
Model: four spheres, AccStruct: BIH



Results

Animationpath

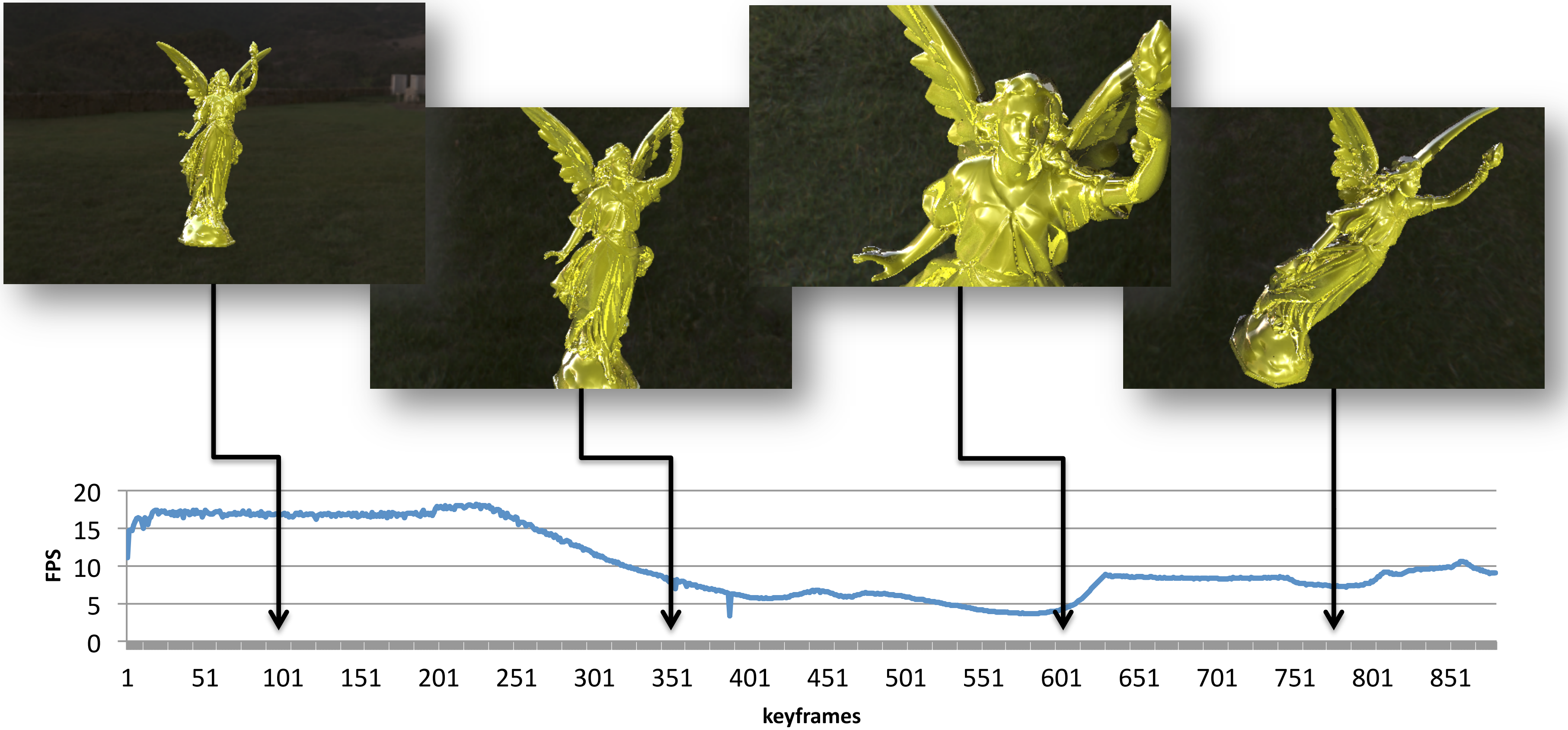
Model: clio all, AccStruct: BIH



Results

Animationpath

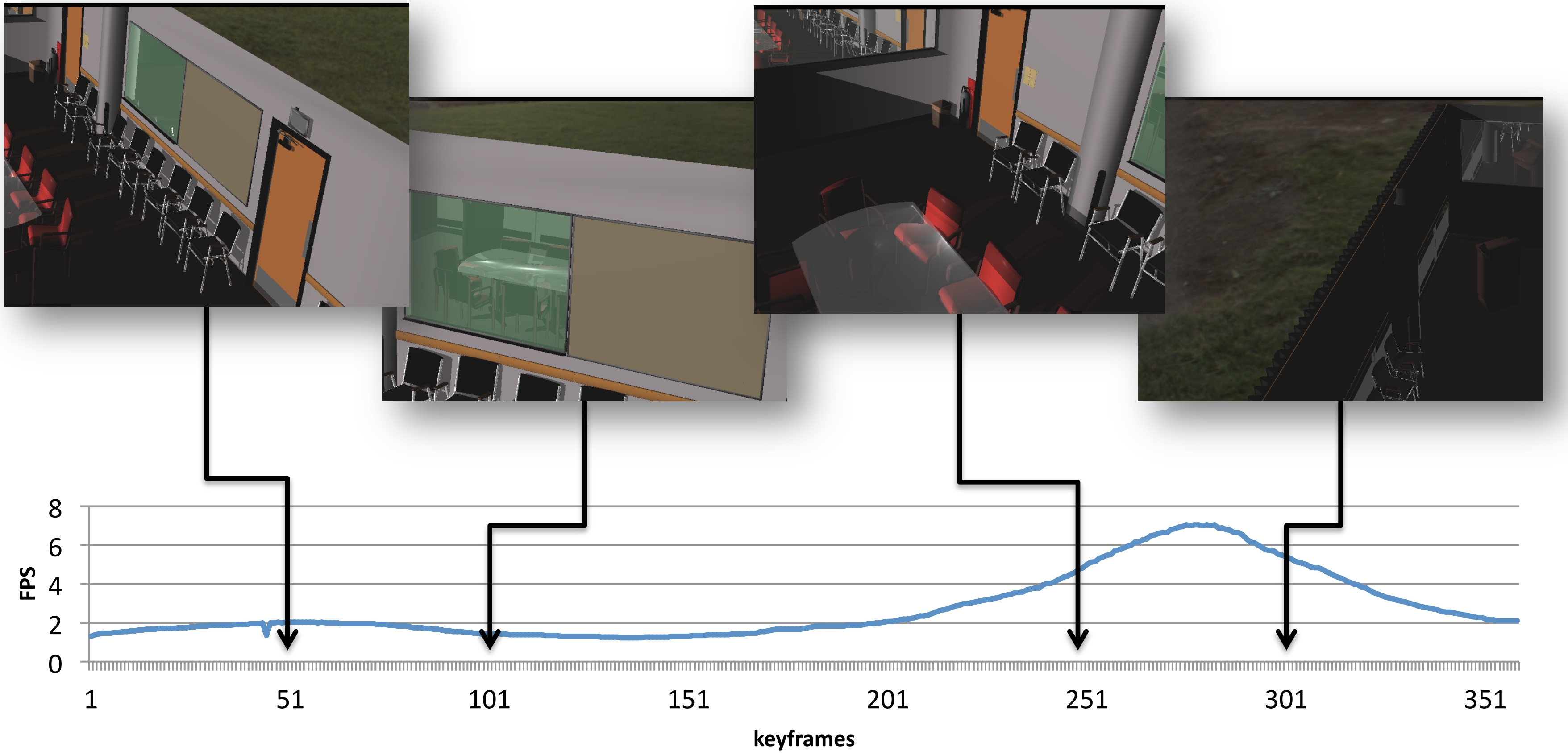
Model: lucy, AccStruct: BIH



Results

Animationpath

Model: conference, AccStruct: BIH



Results

Acceleration Structures

		four spheres	teapot high	lucy	conference	clio all
triangle count		5452	16896	78870	282094	348397
BVH	build time	0,07 sec	0,23 sec	1,27 sec	3,01 sec	5,13 sec
	node count	10.903	33.791	157.739	564.187	696.793
	size	0,374 MB	1,16 MB	5,415 MB	10,32 MB	23,92 MB

Results

Acceleration Structures

		four spheres	**teapot high**	**lucy**	**conference**	**clio all**
triangle count		5452	16896	78870	282094	348397
BVH	build time	0,07 sec	0,23 sec	1,27 sec	3,01 sec	5,13 sec
	node count	10.903	33.791	157.739	564.187	696.793
	size	0,374 MB	1,16 MB	5,415 MB	10,32 MB	23,92 MB
BIH	build time	0,52 sec	1,66 sec	7,38 sec	12,28 sec	25,47 sec
	node count	26.805	71.941	386.781	576.579	1.285.357
	size	0,307 MB	0,823 MB	4,426 MB	6,6 MB	14,71 MB

Results

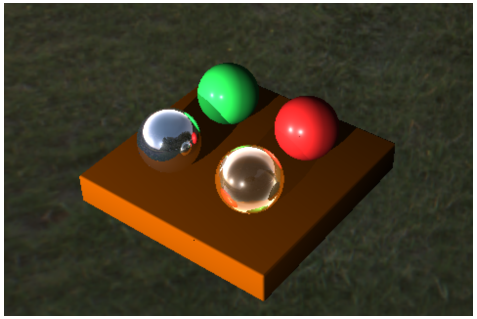
Acceleration Structures

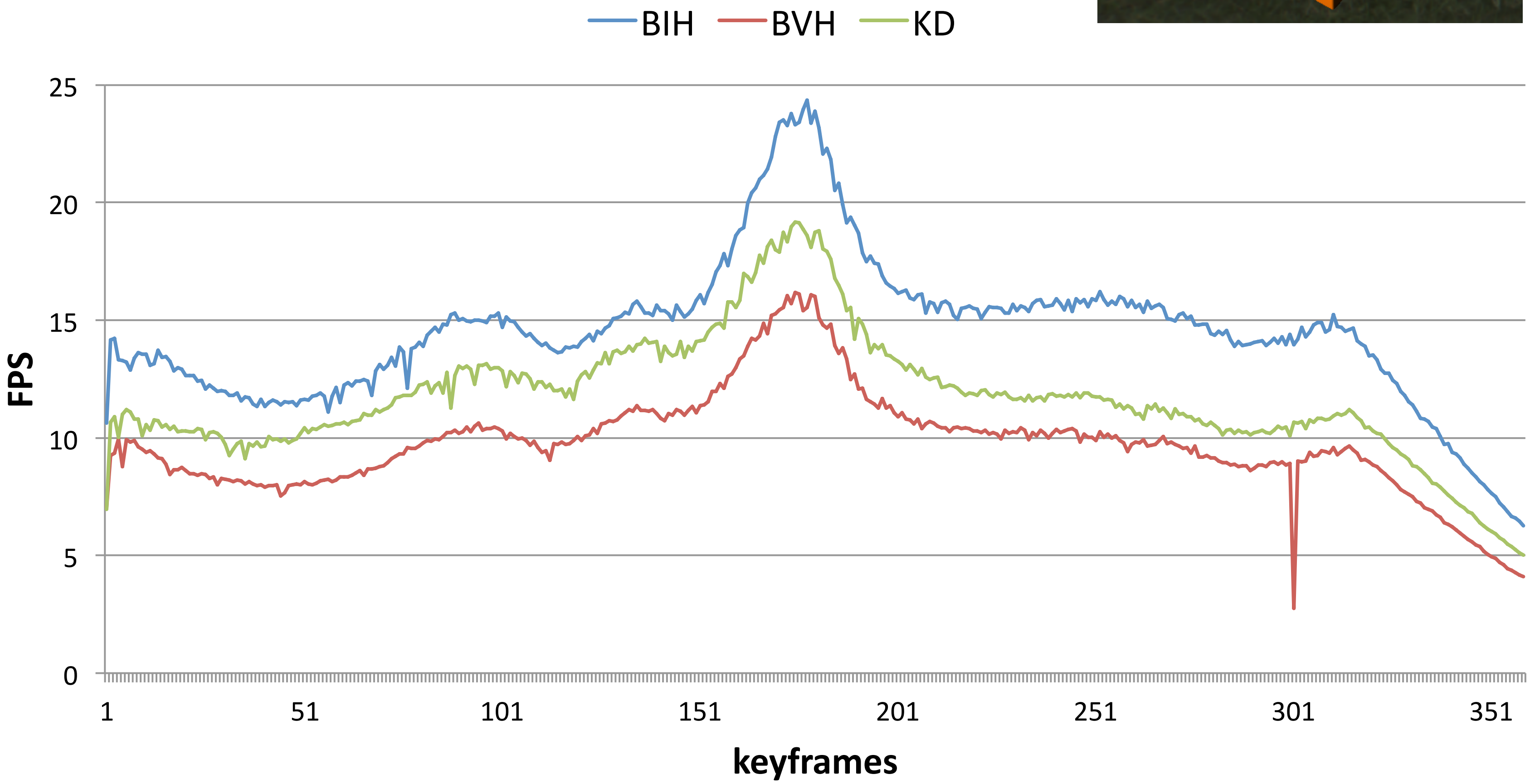
		four spheres	teapot high	lucy	conference	clio all
triangle count		5452	16896	78870	282094	348397
BVH	build time	0,07 sec	0,23 sec	1,27 sec	3,01 sec	5,13 sec
	node count	10.903	33.791	157.739	564.187	696.793
	size	0,374 MB	1,16 MB	5,415 MB	10,32 MB	23,92 MB
BIH	build time	0,52 sec	1,66 sec	7,38 sec	12,28 sec	25,47 sec
	node count	26.805	71.941	386.781	576.579	1.285.357
	size	0,307 MB	0,823 MB	4,426 MB	6,6 MB	14,71 MB
KD	build time	2,17 sec	20,05 sec	592,51 sec	168,74 sec*	741,31 sec
	node count	25.603	82.015	456.693	3.952.379	1.851.887
	size	0,21 MB	0,66 MB	3,65 MB	31,62 MB	14.82 MB

Results

Acceleration Structures

Model: four spheres



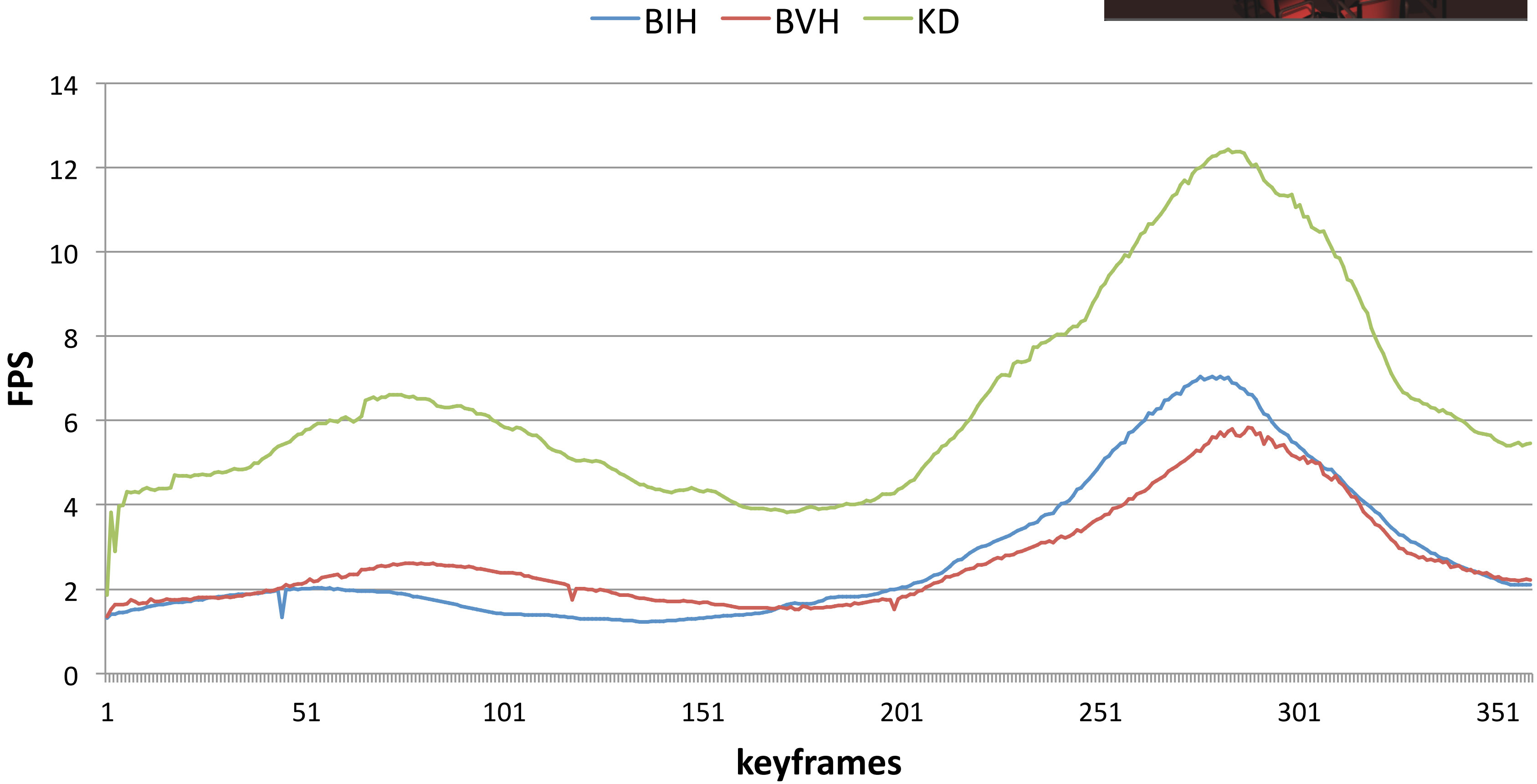


Bauhaus-Universität Weimar

Results

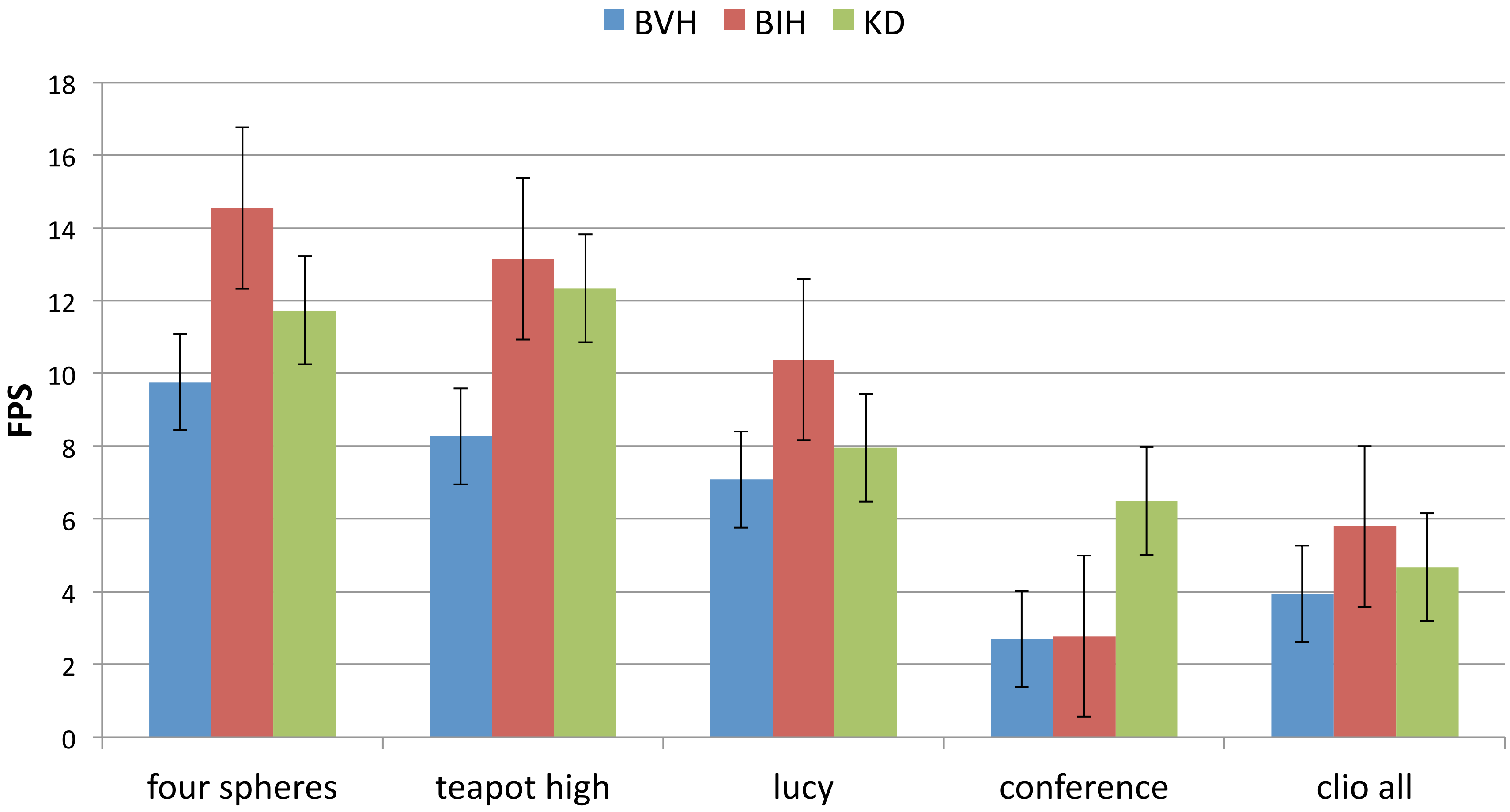
Acceleration Structures

Model: conference



Results

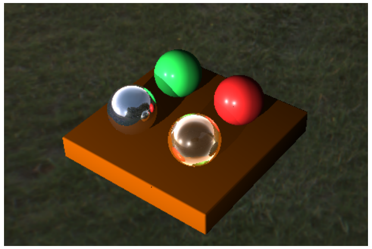
Acceleration Structures

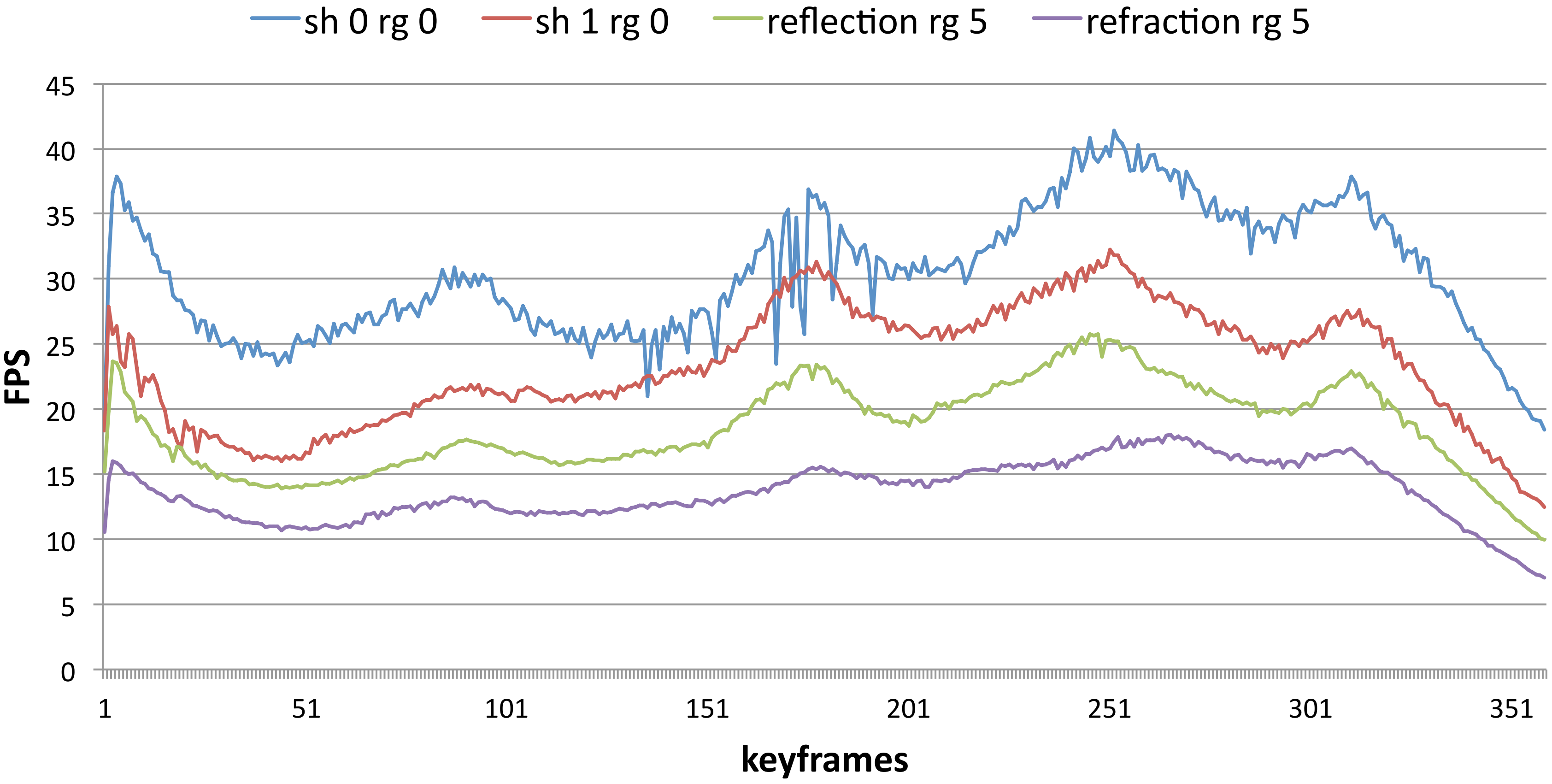


Results

Raygeneration, Reflection and Refraction

Model: four spheres, AccStruct: BIH



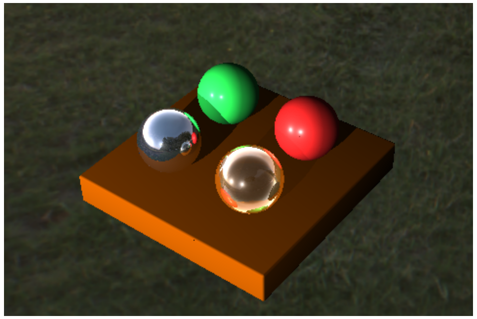


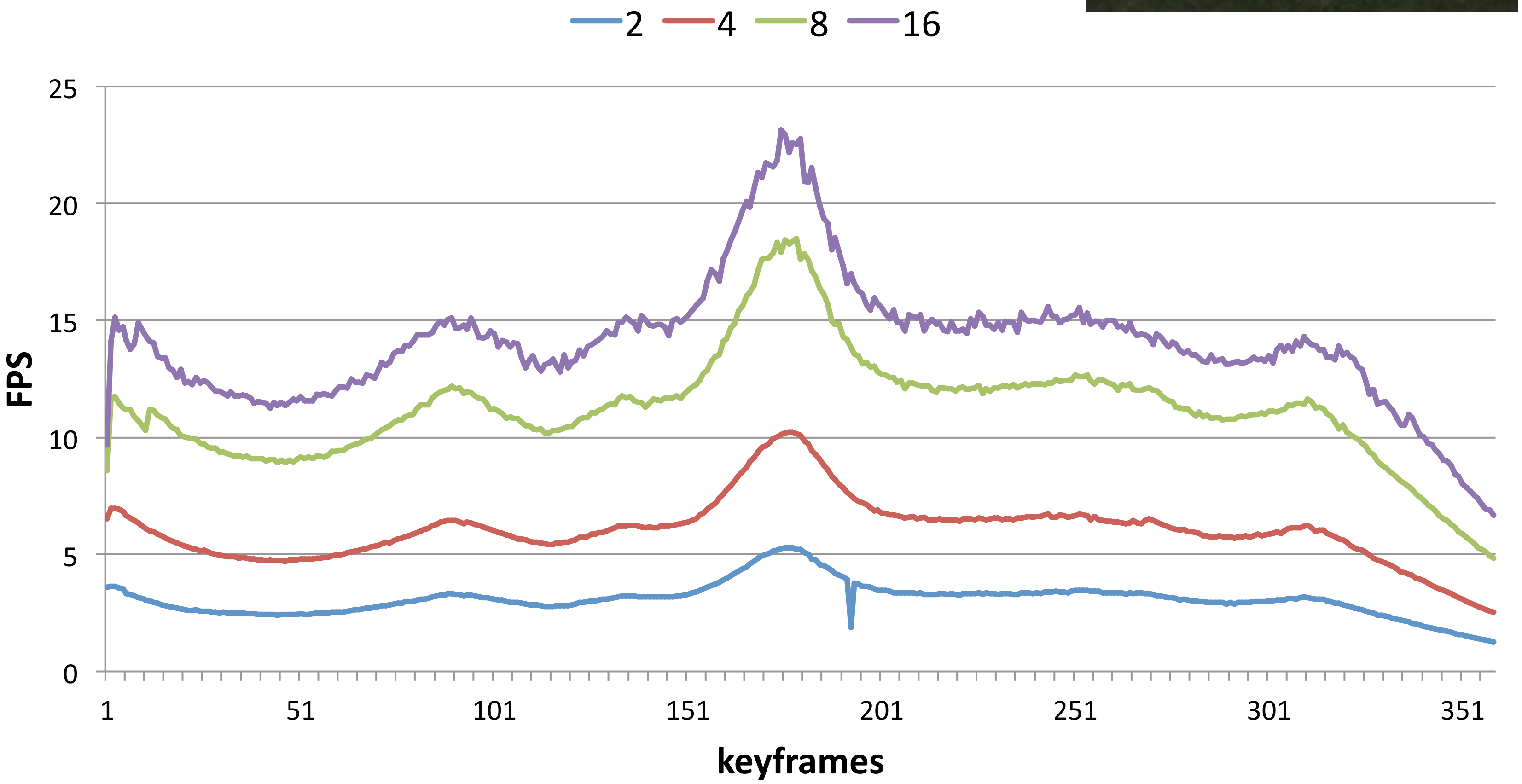
Bauhaus-Universität Weimar

Results

Number of threads

Model: four spheres, AccStruct: BIH

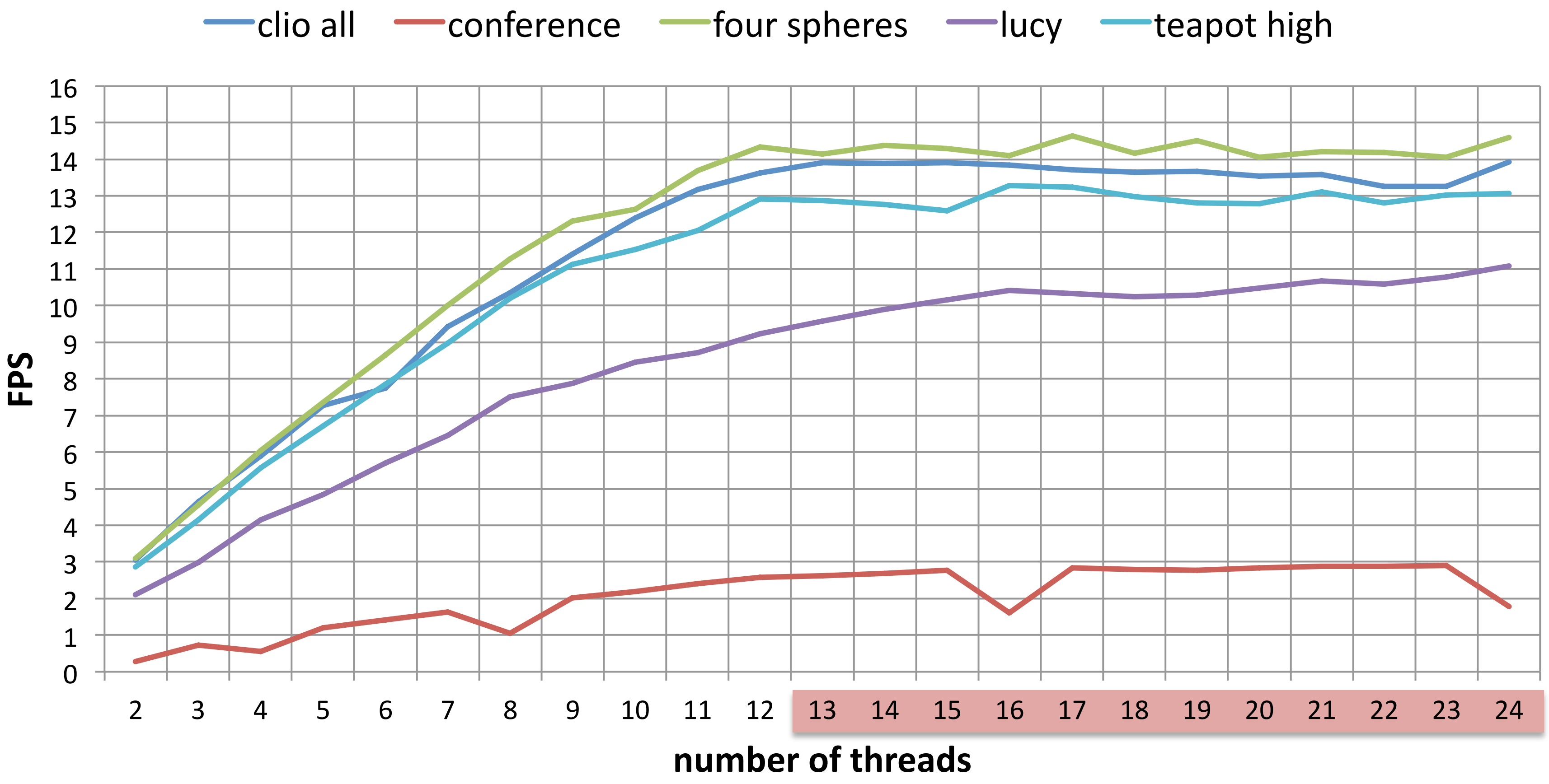




Results

Number of threads

AccStruct: BIH

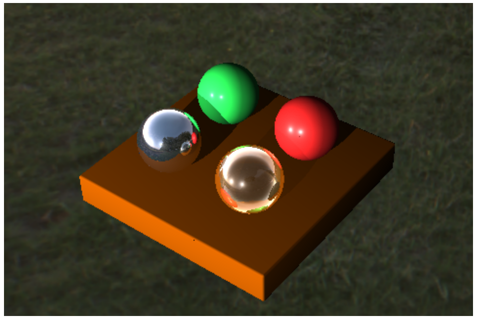


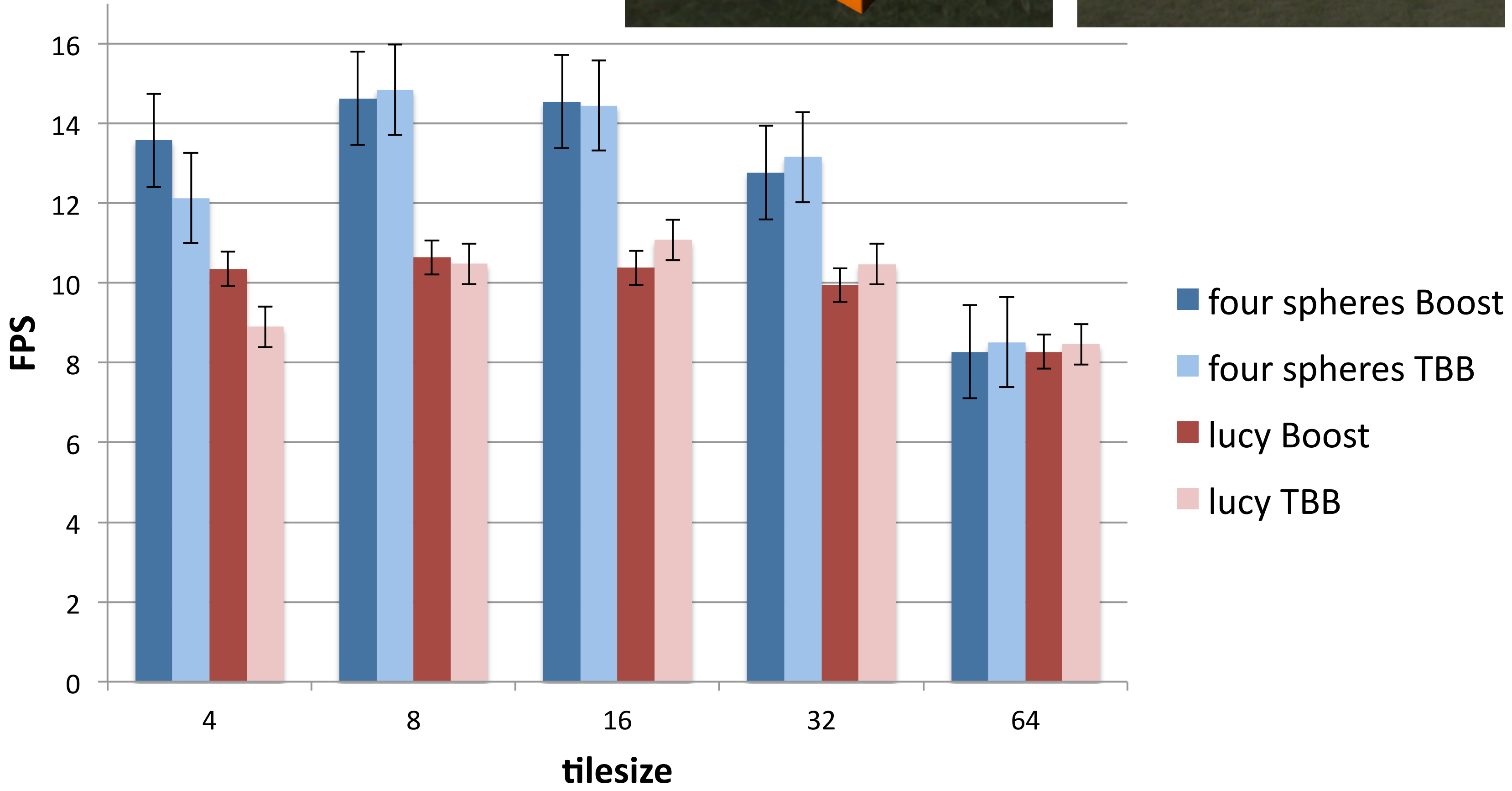
Bauhaus-Universität Weimar

Results

Boost (19) vs. TBB

AccStruct: BIH

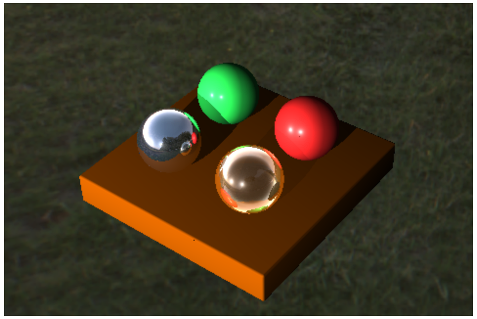


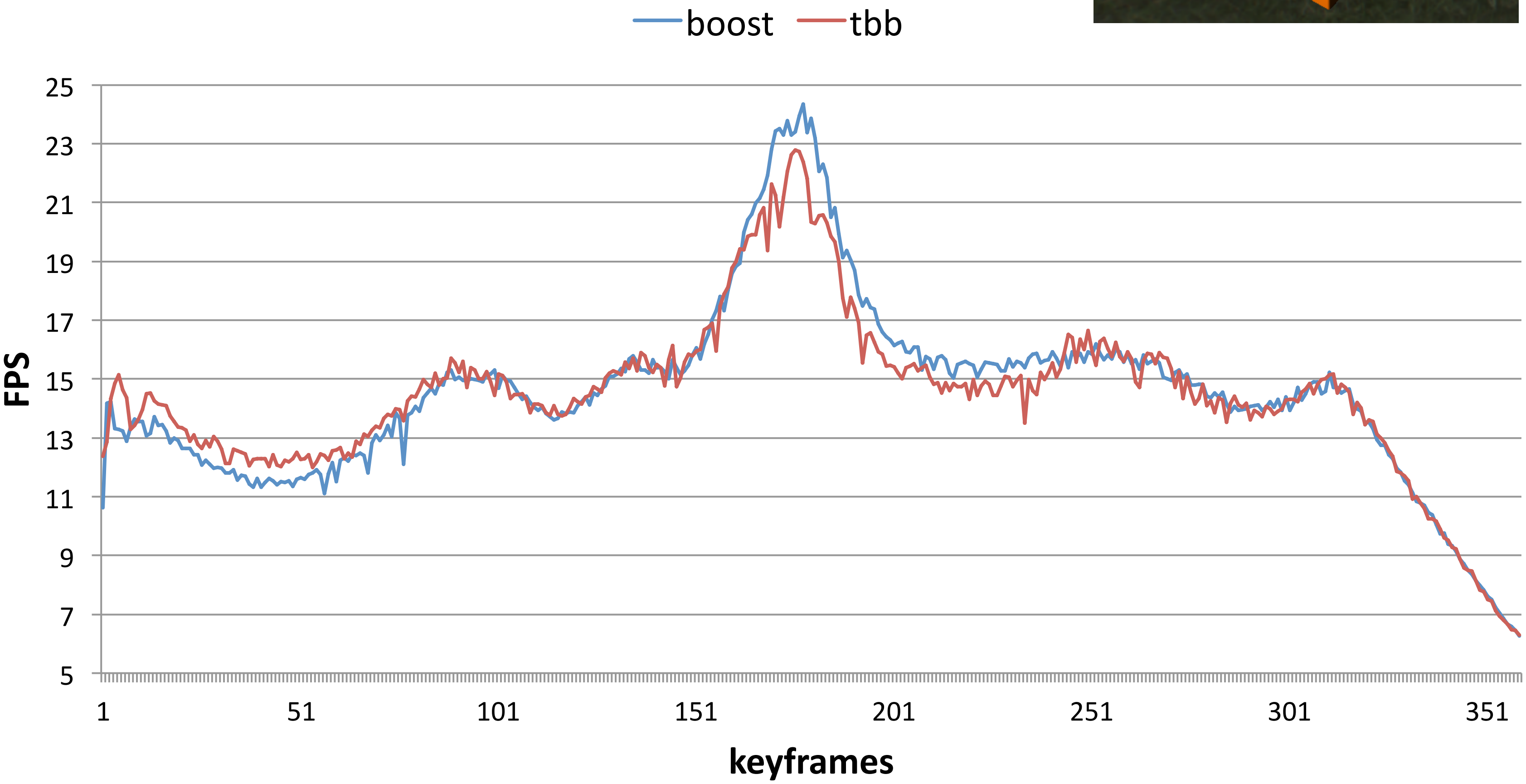


Results

Boost (19) vs. TBB

Model: four spheres, AccStruct: BIH

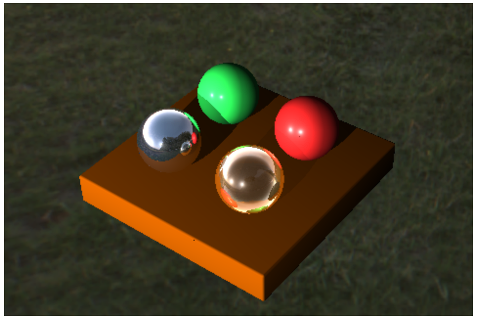


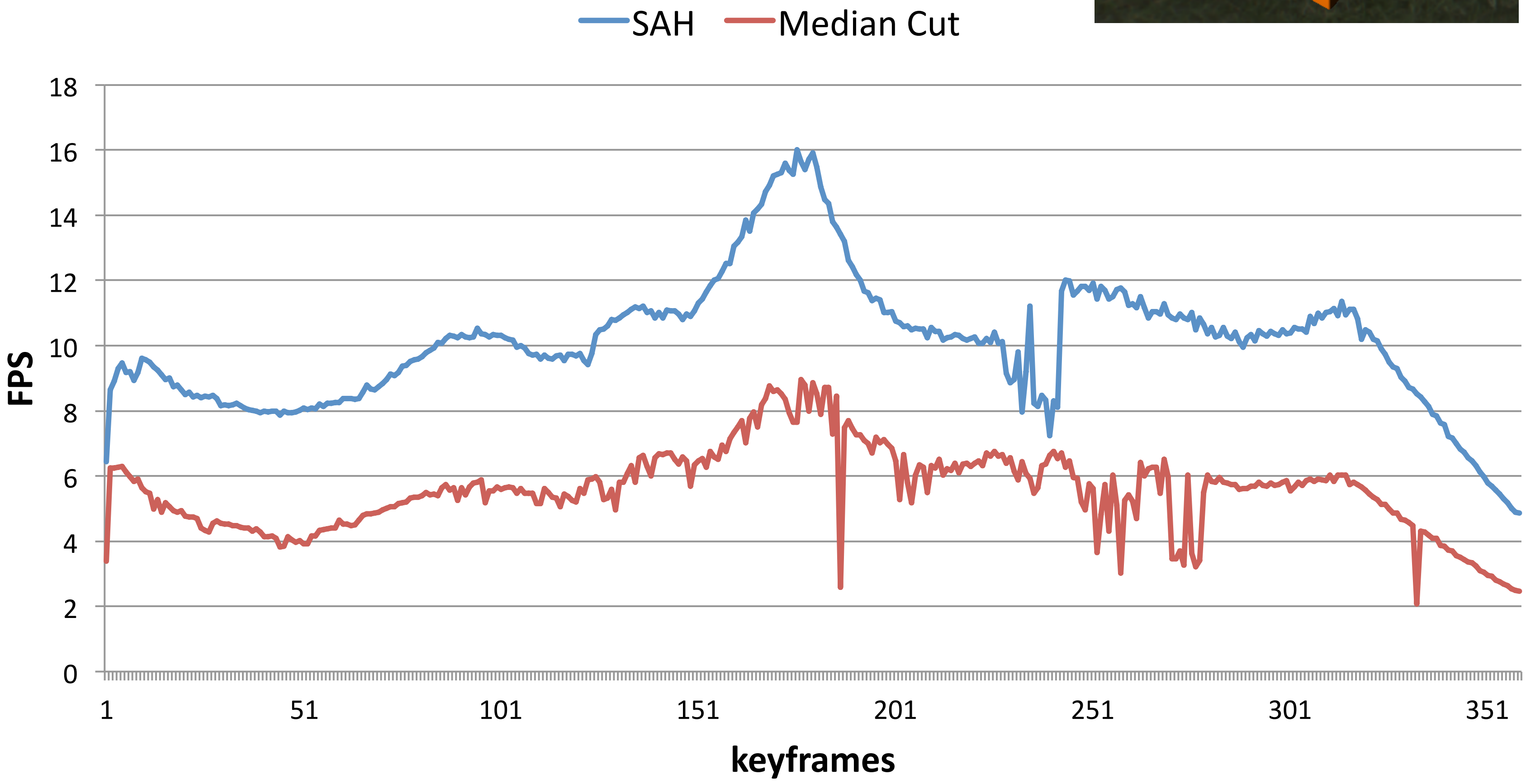


Results

SAH vs. Median Cut

Model: four spheres, AccStruct: BVH



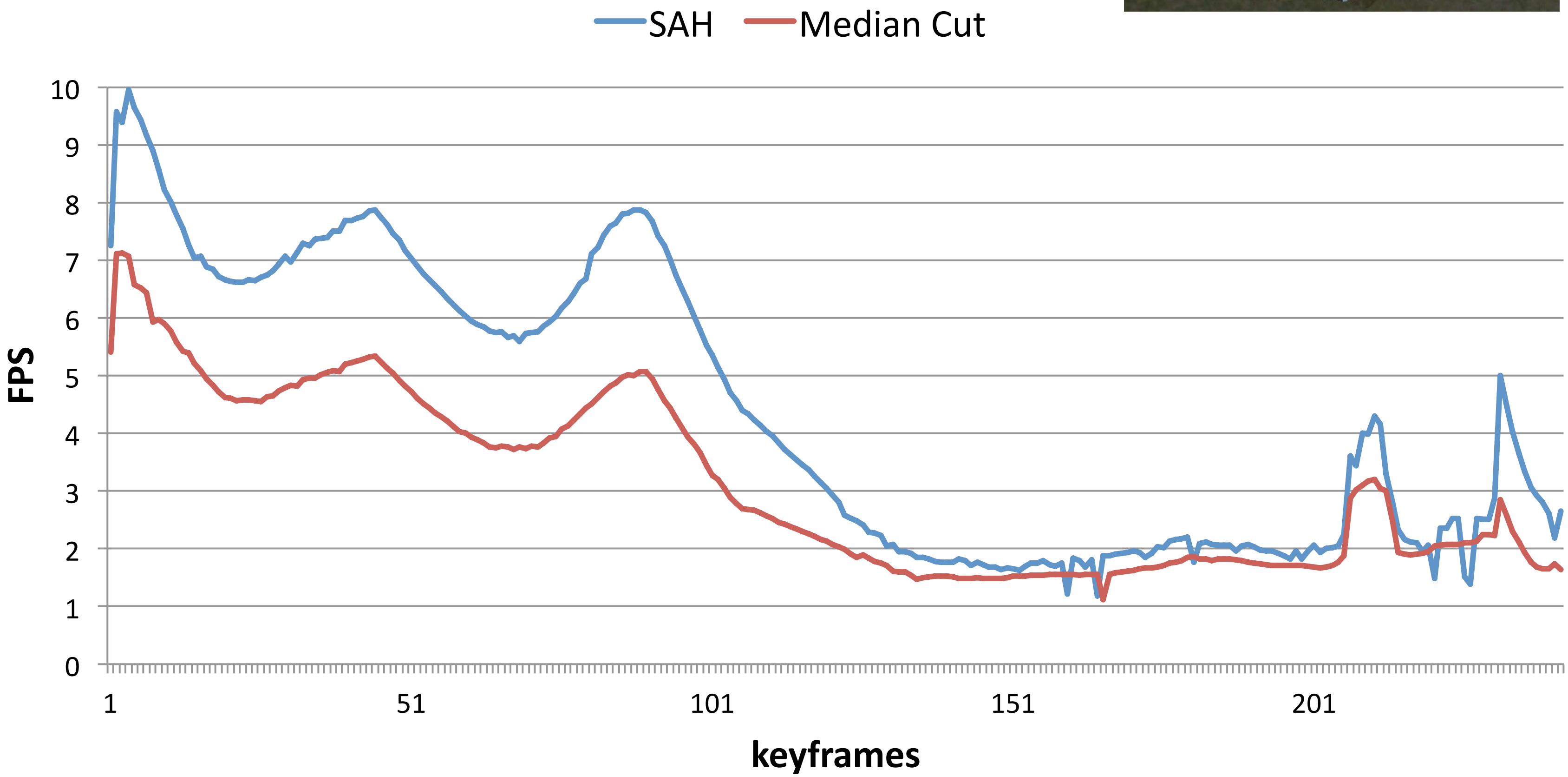


Bauhaus-Universität Weimar

Results

SAH vs. Median Cut

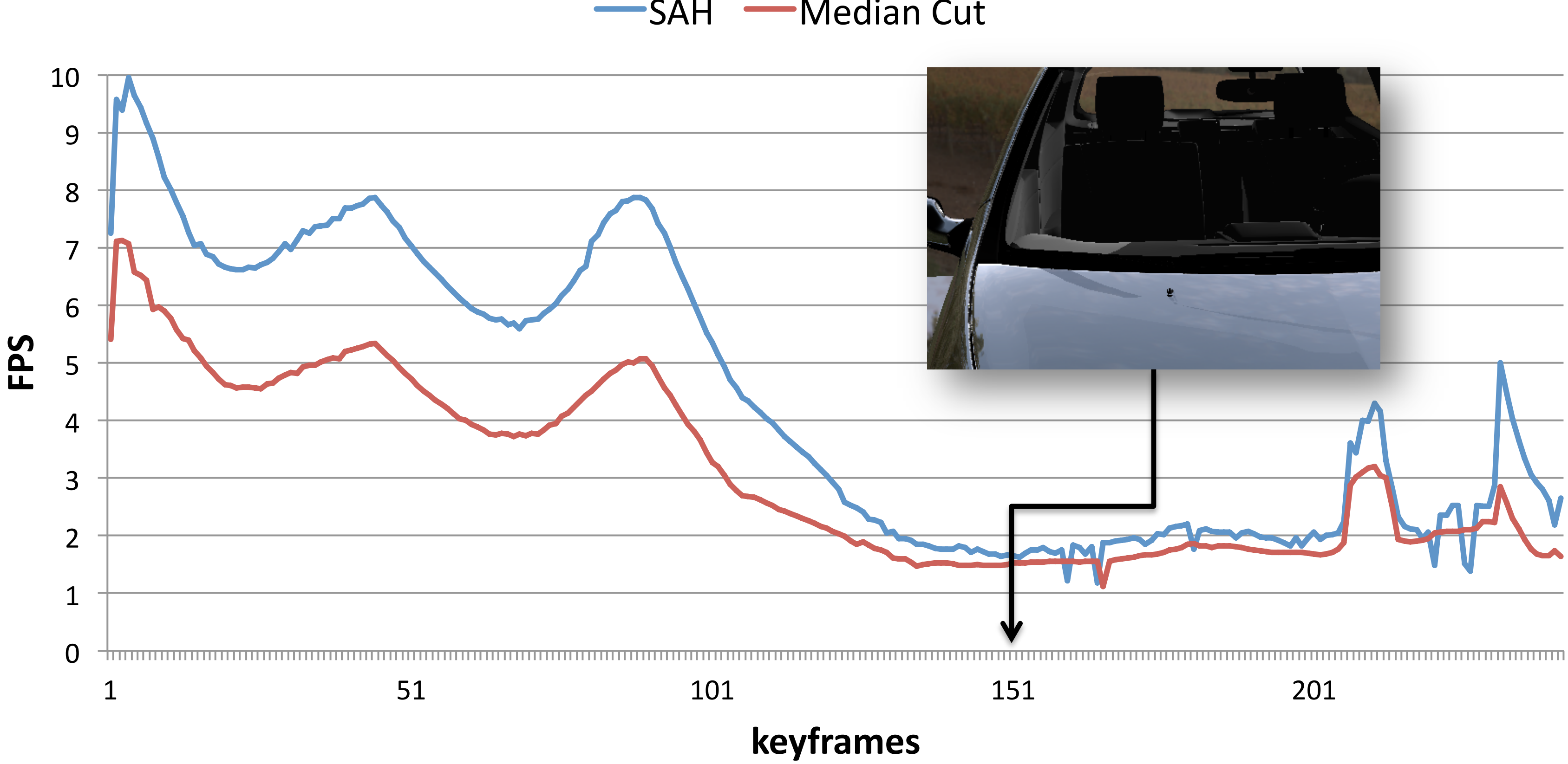
Model: clio all, AccStruct: BVH



Results

SAH vs. Median Cut

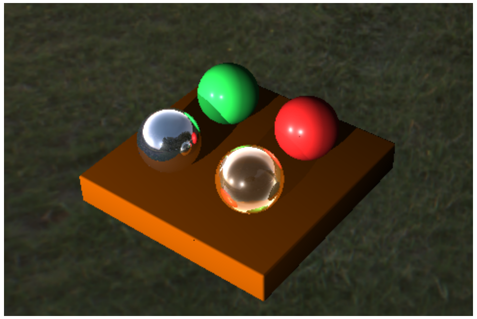
Model: clio all, AccStruct: BVH

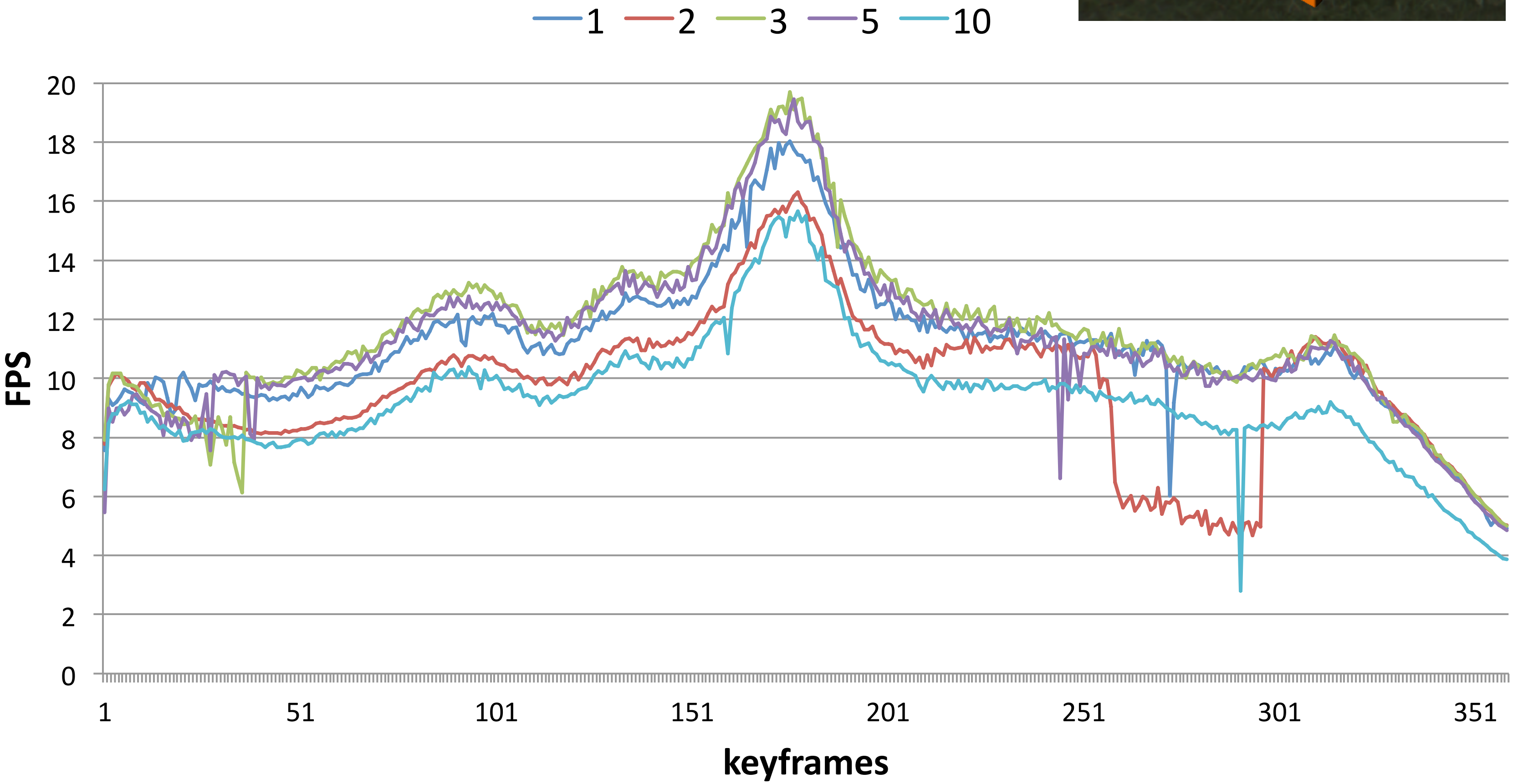


Results

Triangle Leaf Count

Model: four spheres, AccStruct: BVH

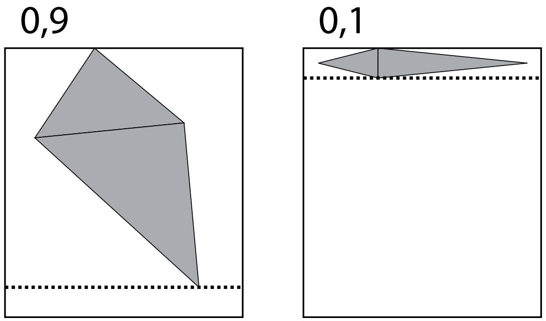


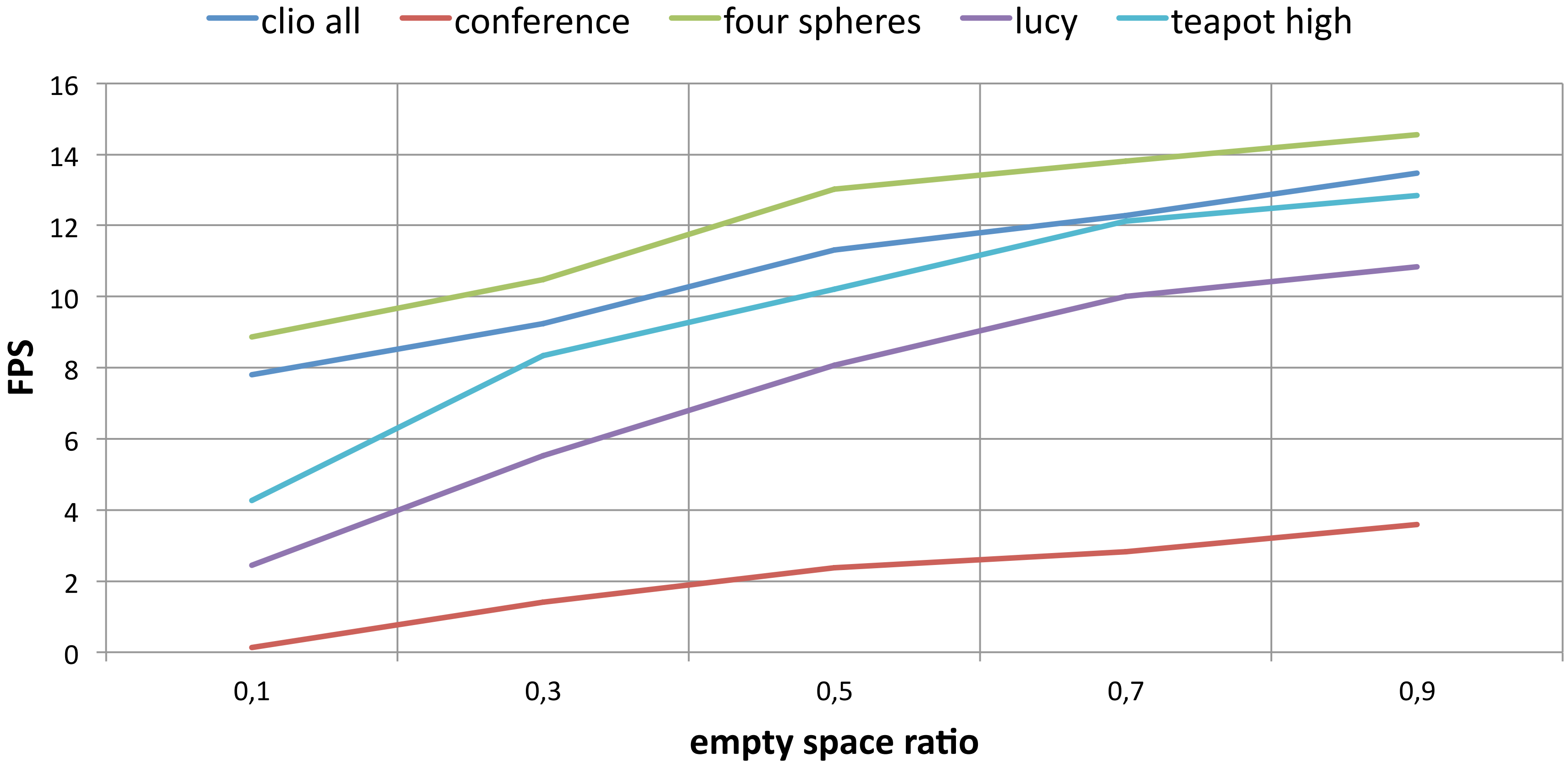


Results

Empty Space Ratio

AccStruct: BIH

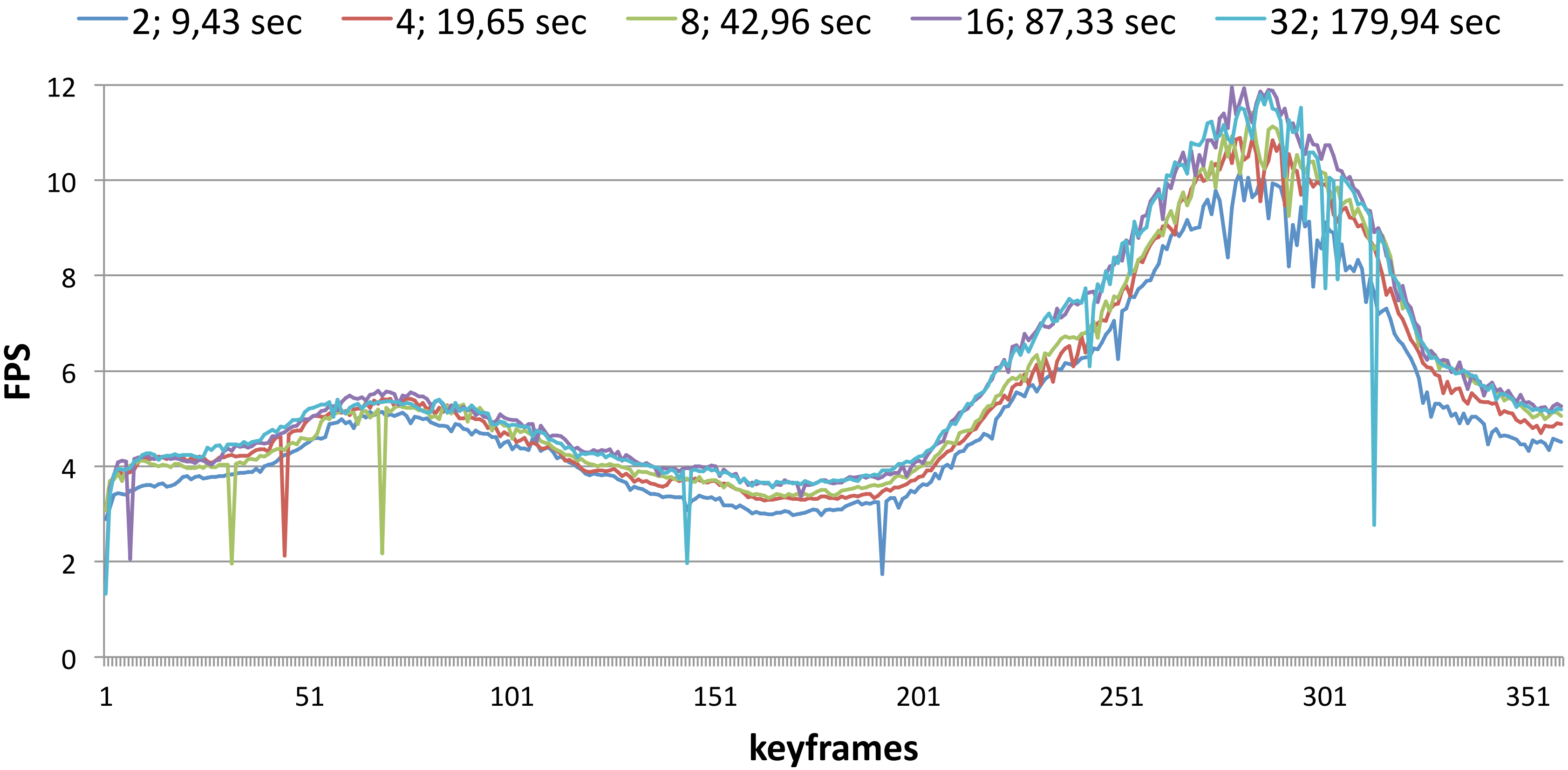




Results

Number of Bins

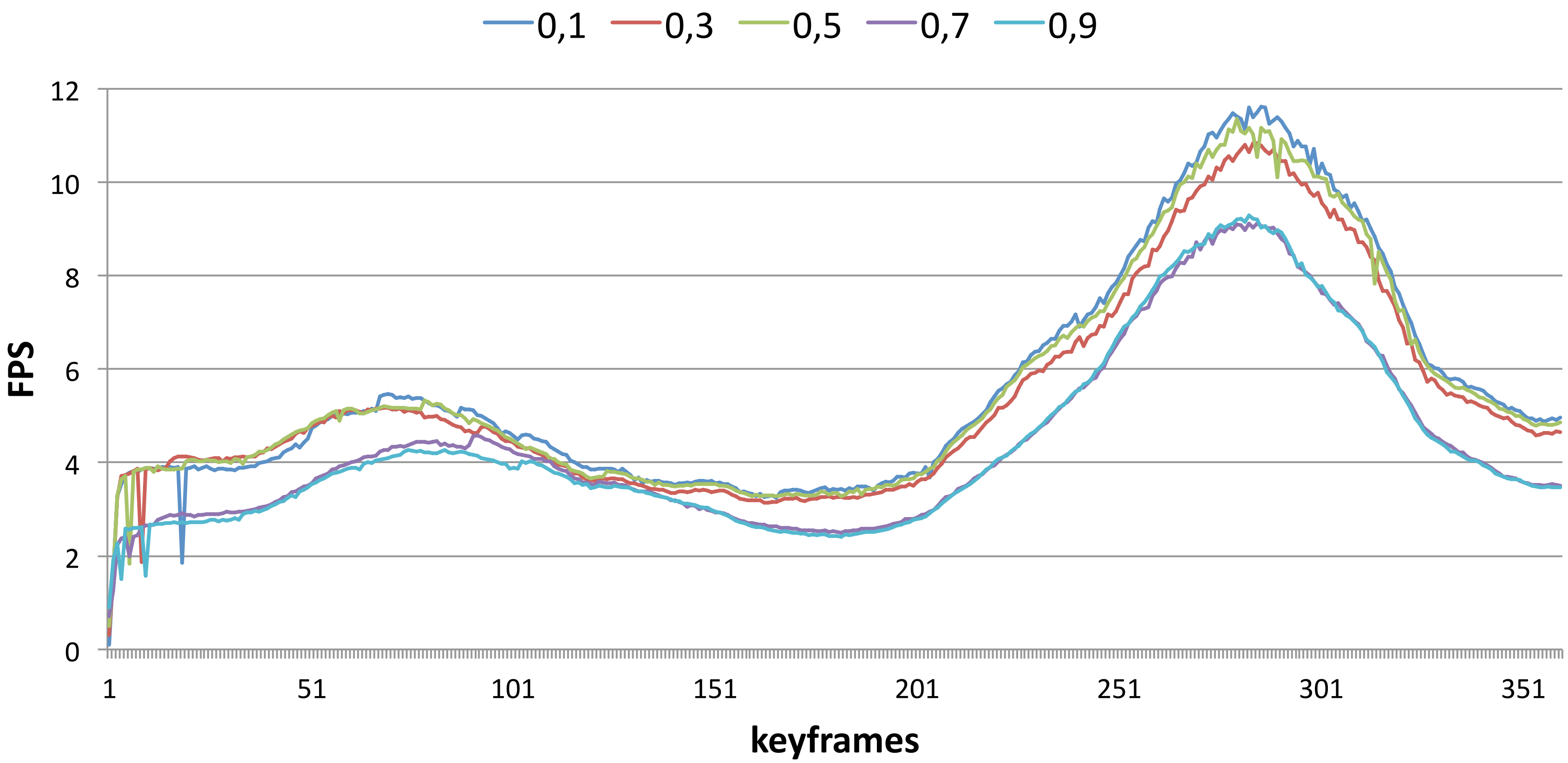
Model: conference, AccStruct: KD, Mode: bin



Results

Empty Space Ratio

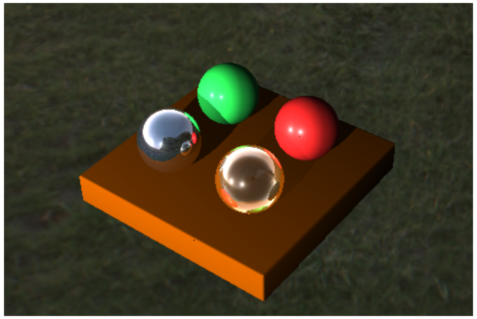
Model: conference, AccStruct: KD, Mode: vertex

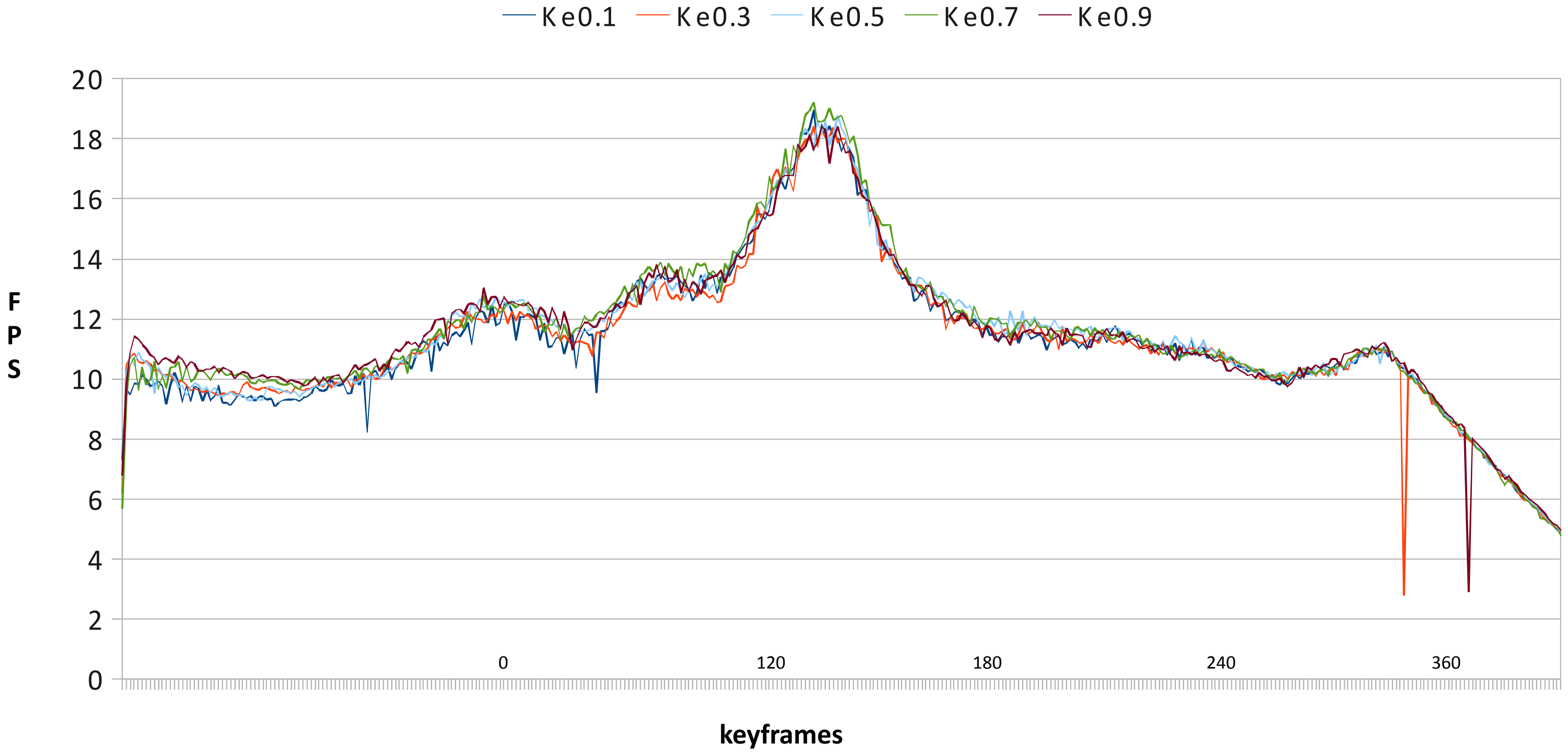


Results

Empty Space Ratio

Model: four spheres, AccStruct: KD, Mode: vertex



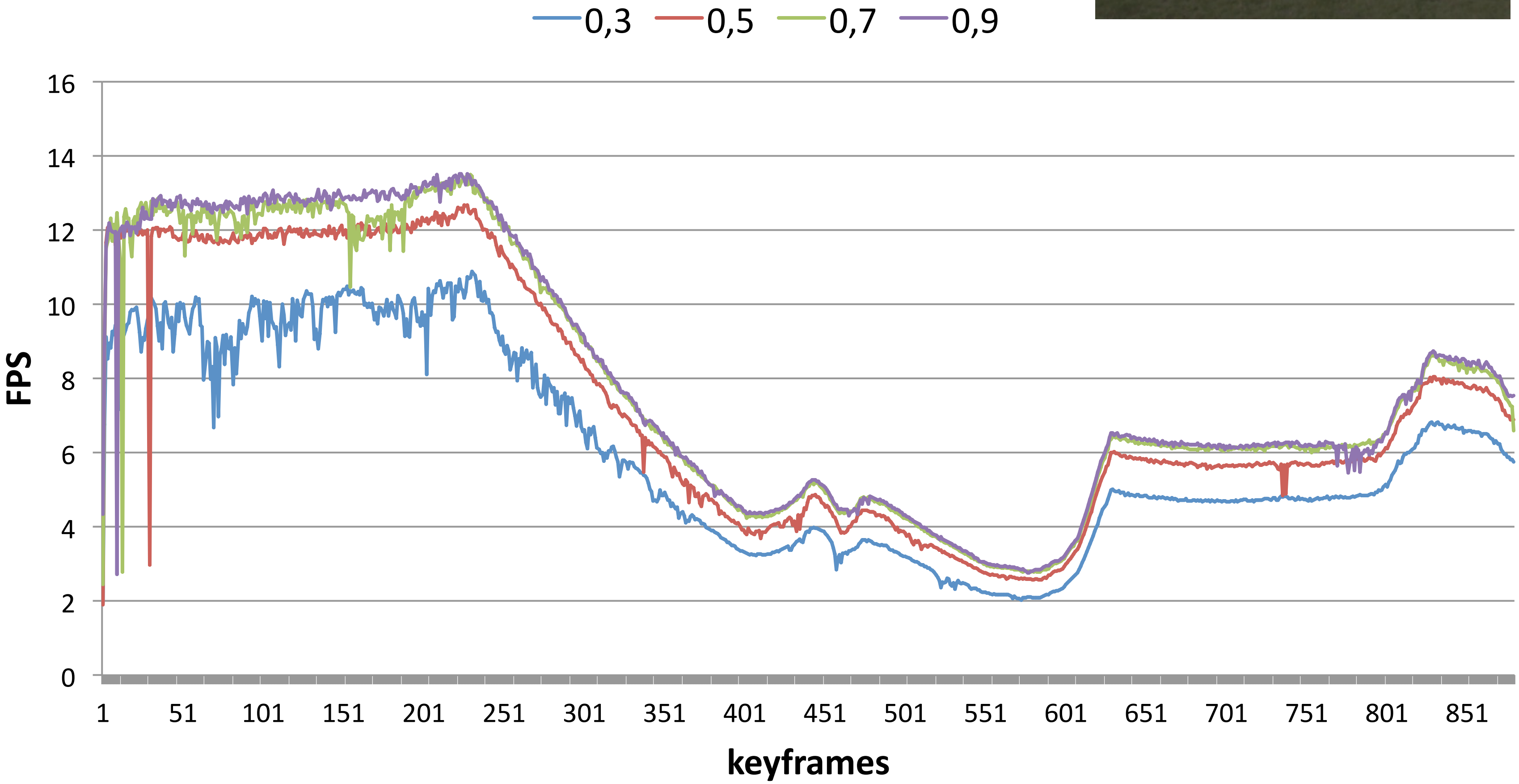


Bauhaus-Universität Weimar

Results

SAH Costs

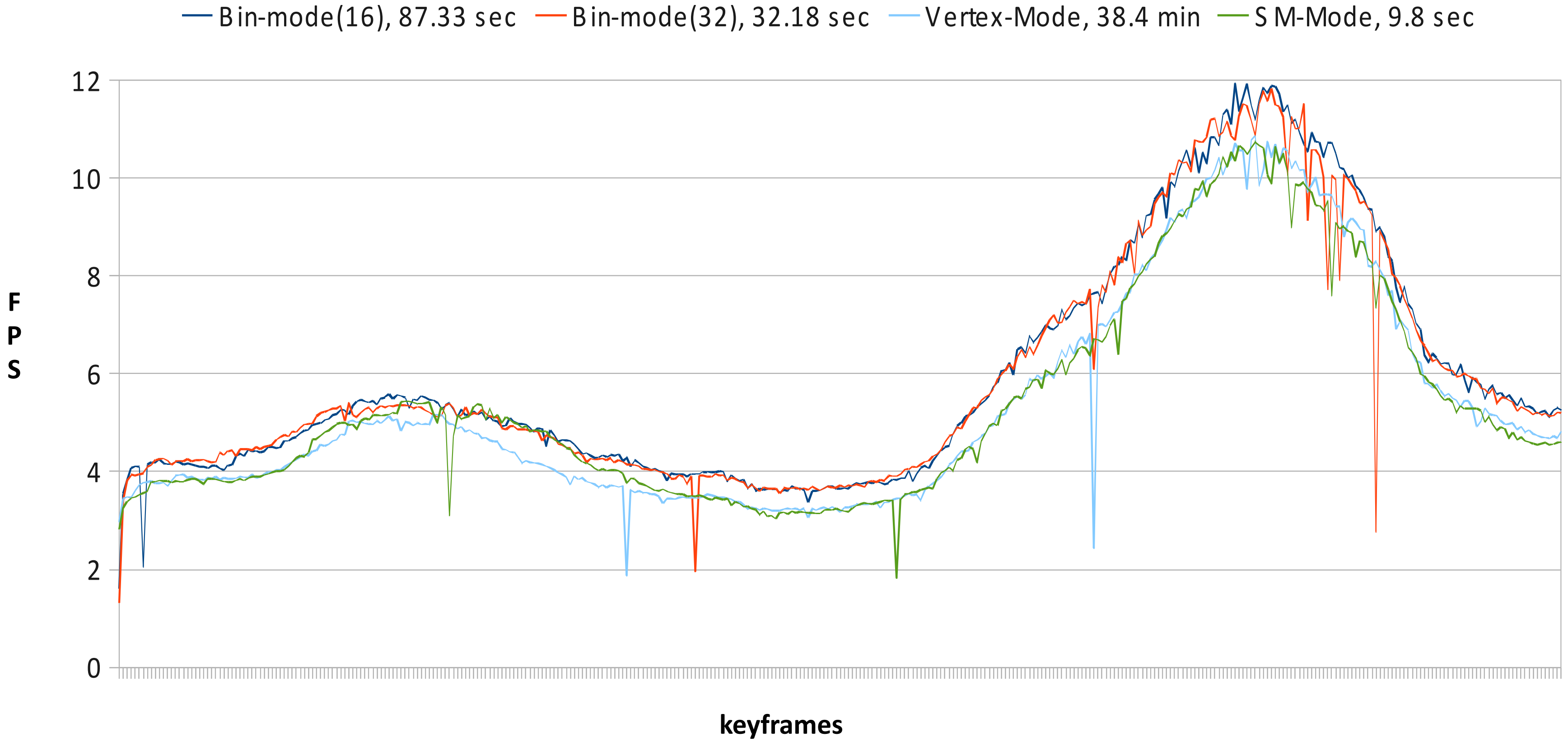
Model: lucy, AccStruct: KD, Mode: vertex



Results

Different Modes

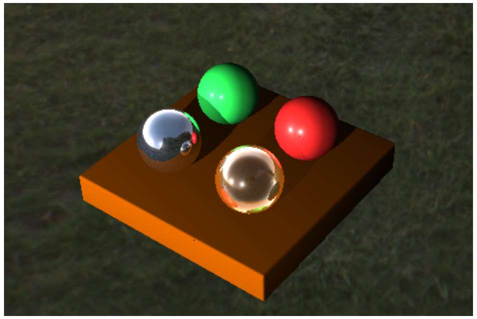
Model: conference, AccStruct: KD

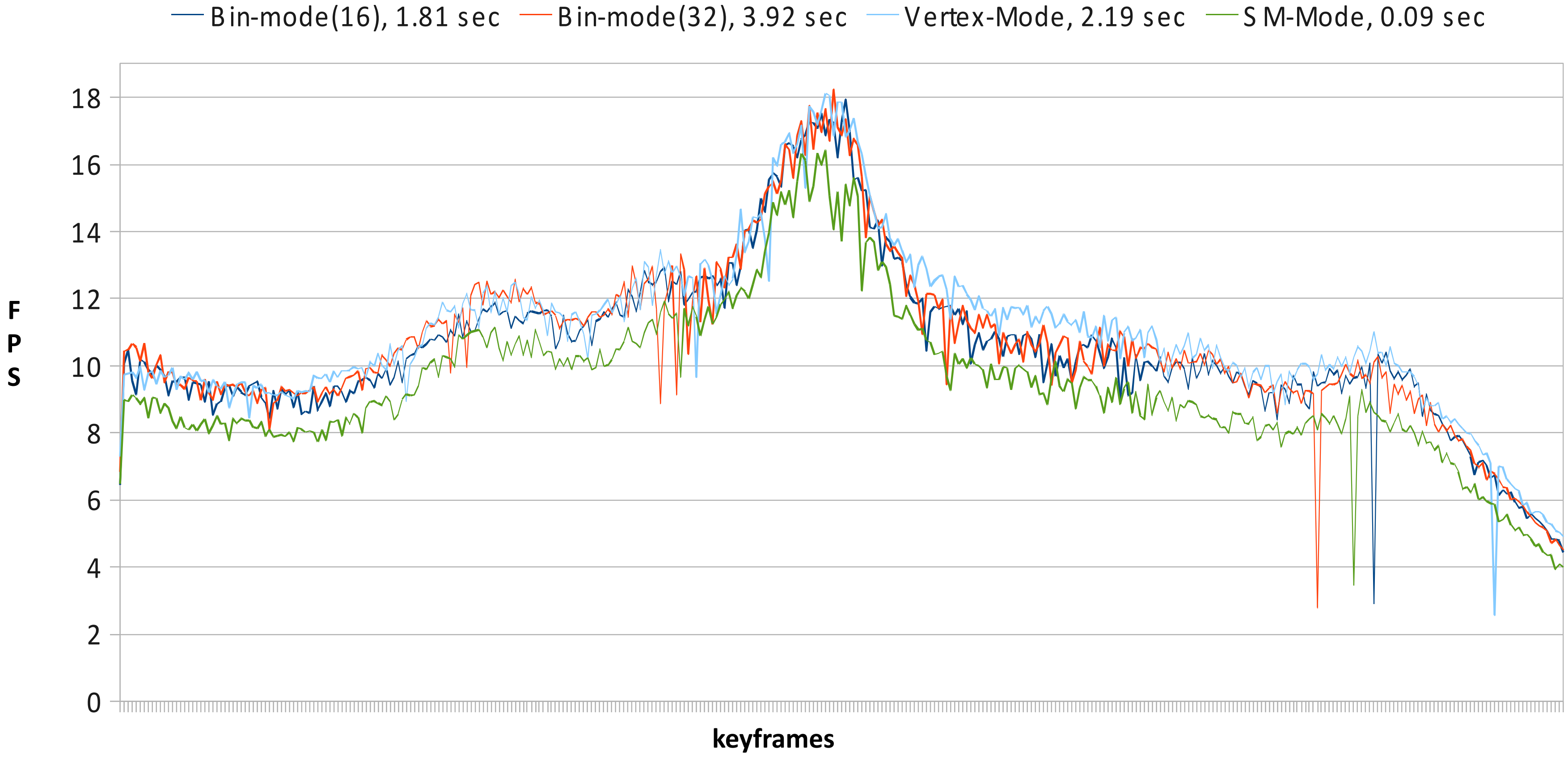


Results

Different Modes

Model: four spheres, AccStruct: KD





Conclusion

Raytracer

- high quality pictures using up to nine ray generations
- including shadows, reflection and refraction
- for scenes of medium complexity we gain 10 Hz at 800 x 600 (BIH)

- no big differences between boost and tbb
- sublinear scaling with increasing number of threads

Conclusion

Acceleration Structures

- BVH has fastest build time
- BIH has best performance on most of the scenes
- KD has a better use of empty space in conference model

Future Work

- textures
- extend to dynamic scenes
  - modular scenes
  - instancing
  - combining different Acceleration Structures
- GPU-Raytracing

DEMO

Thank you.