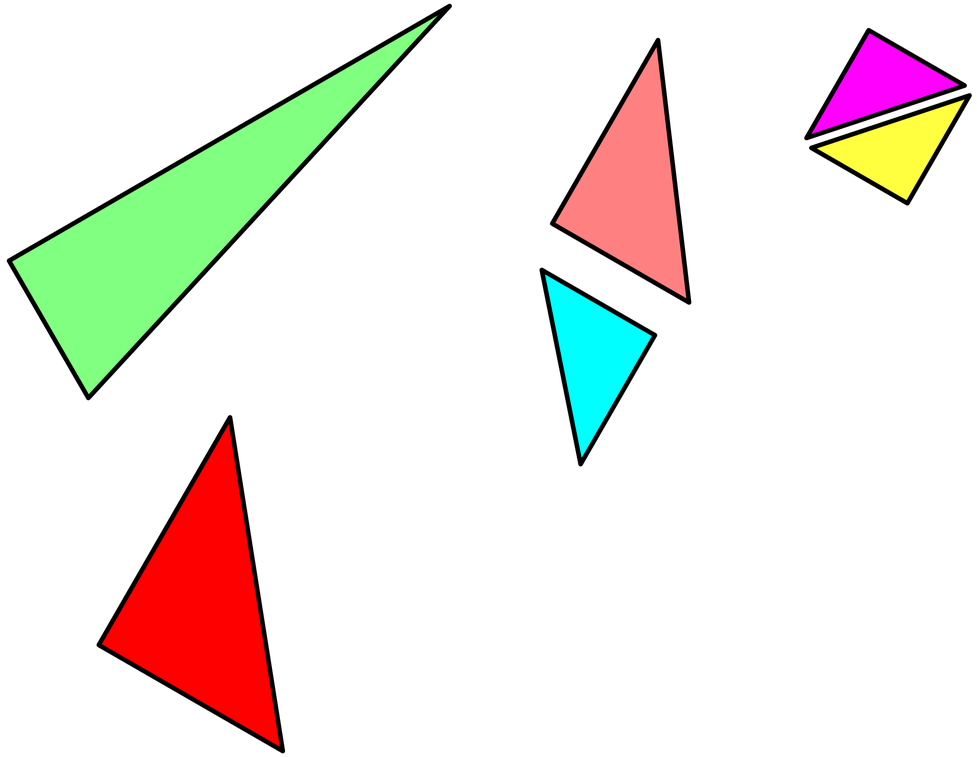
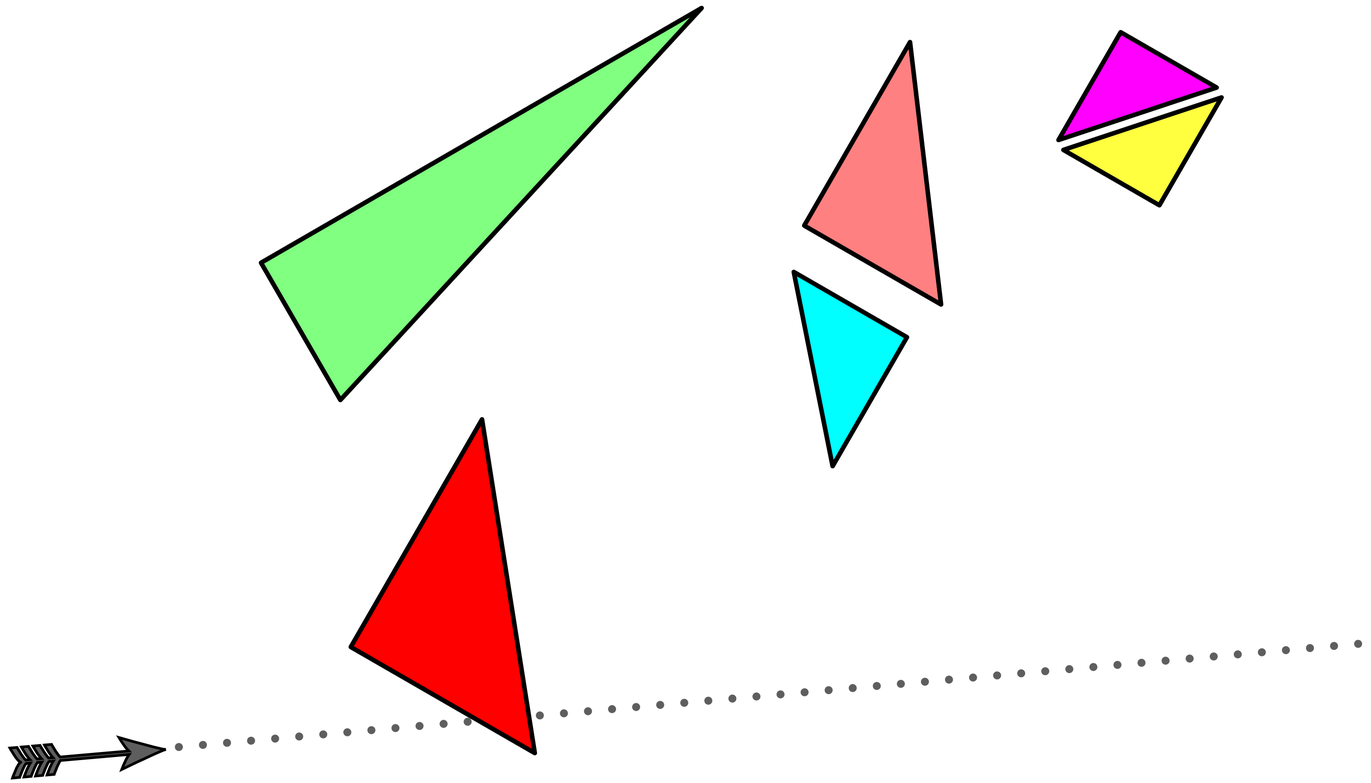
Decoupling Data Layouts from Bounding Volume Hierarchies

Christophe Gyurgyik, Alexander J Root, Fredrik Kjolstad

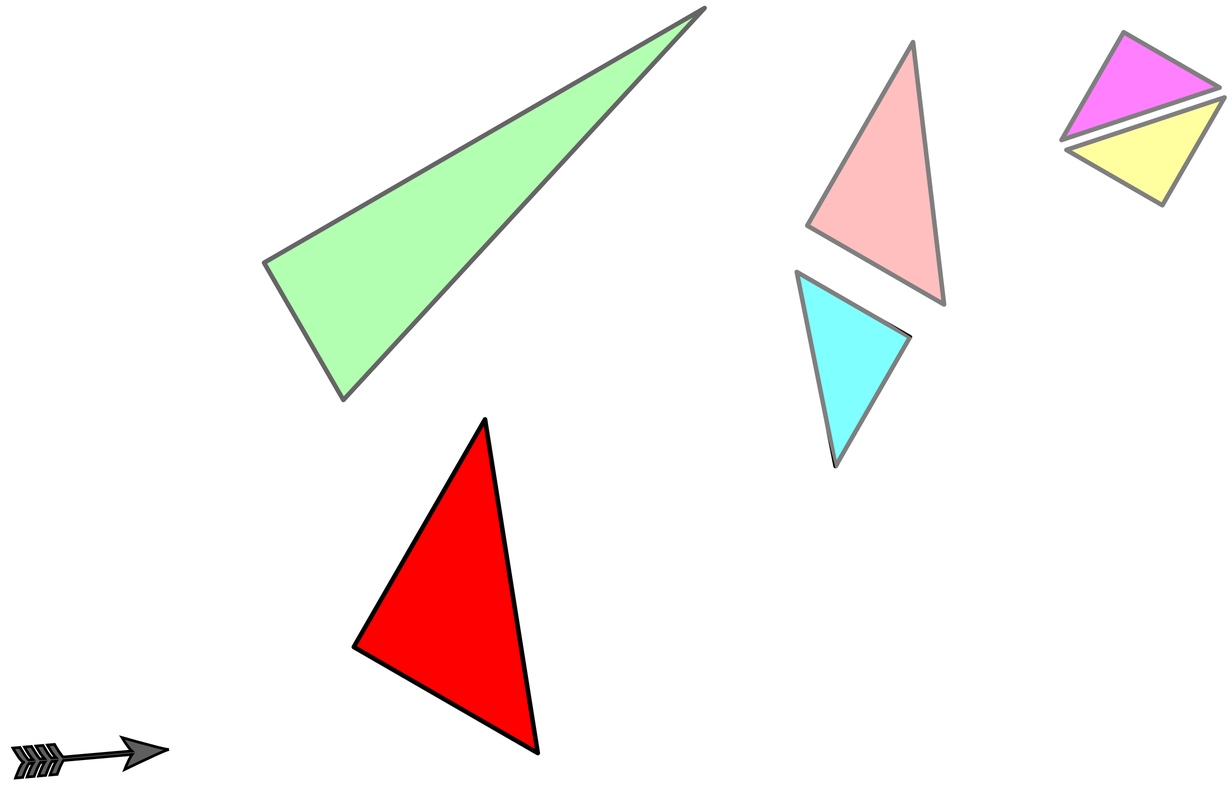
BVHs are indexes for speeding up spatial queries



Which triangles does the → intersect?



Which triangles does the → intersect?



intersects(→,) ✗

intersects(→,) ✓

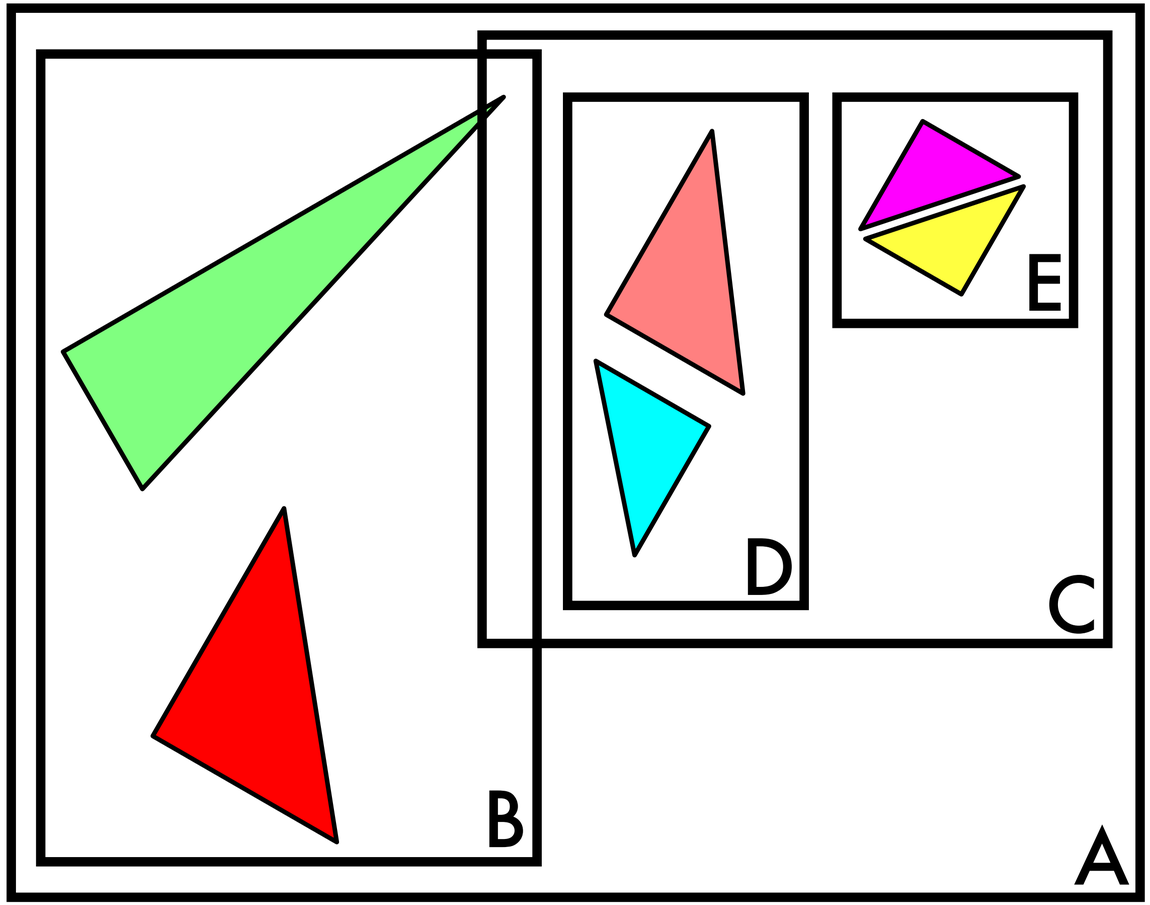
intersects(→,) ✗

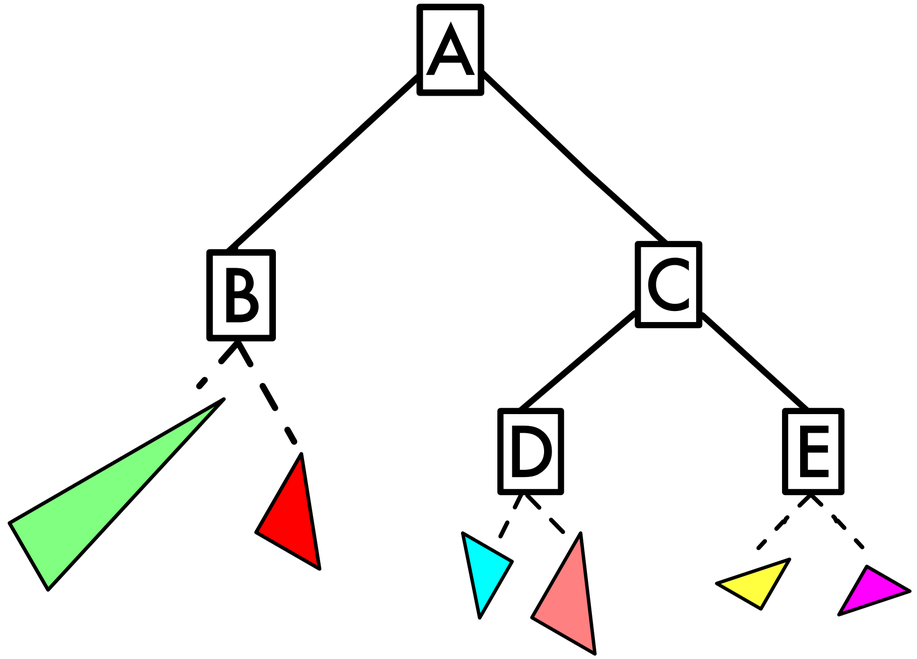
intersects(→,) ✗

intersects(→,) ✗

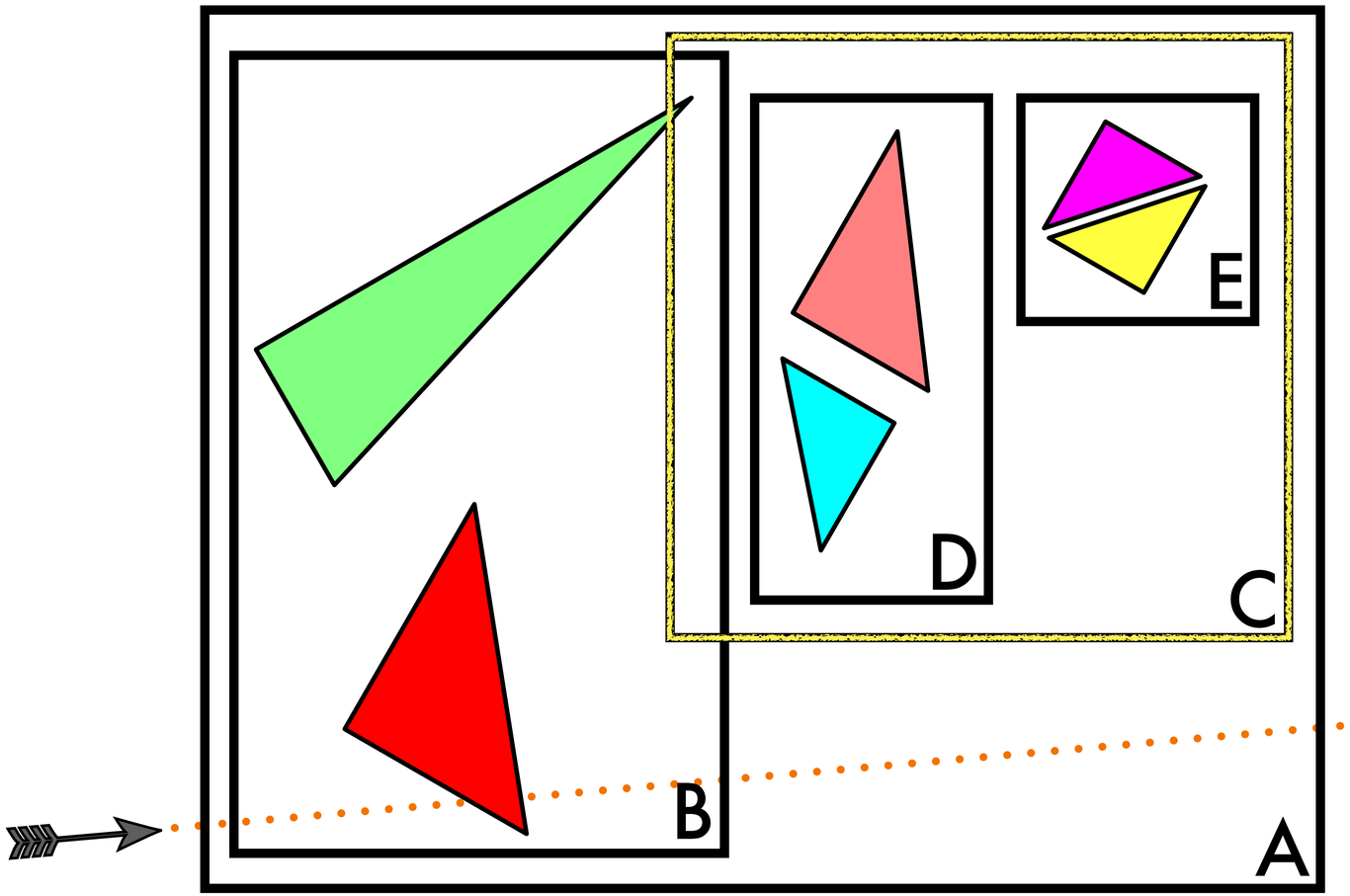
intersects(→,) ✗

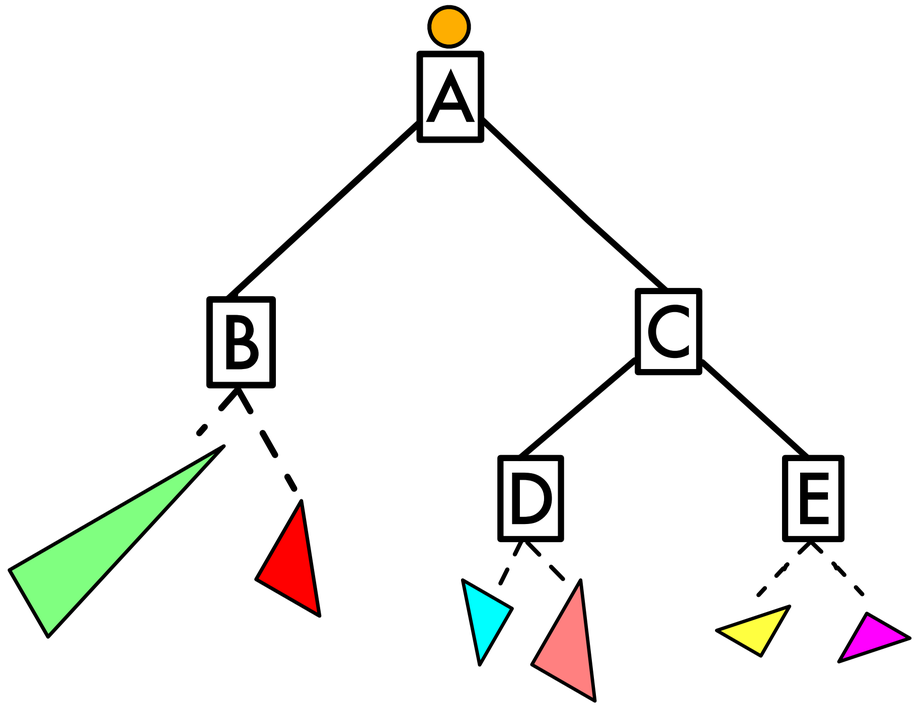
Bounding Volume Hierarchy (BVH)



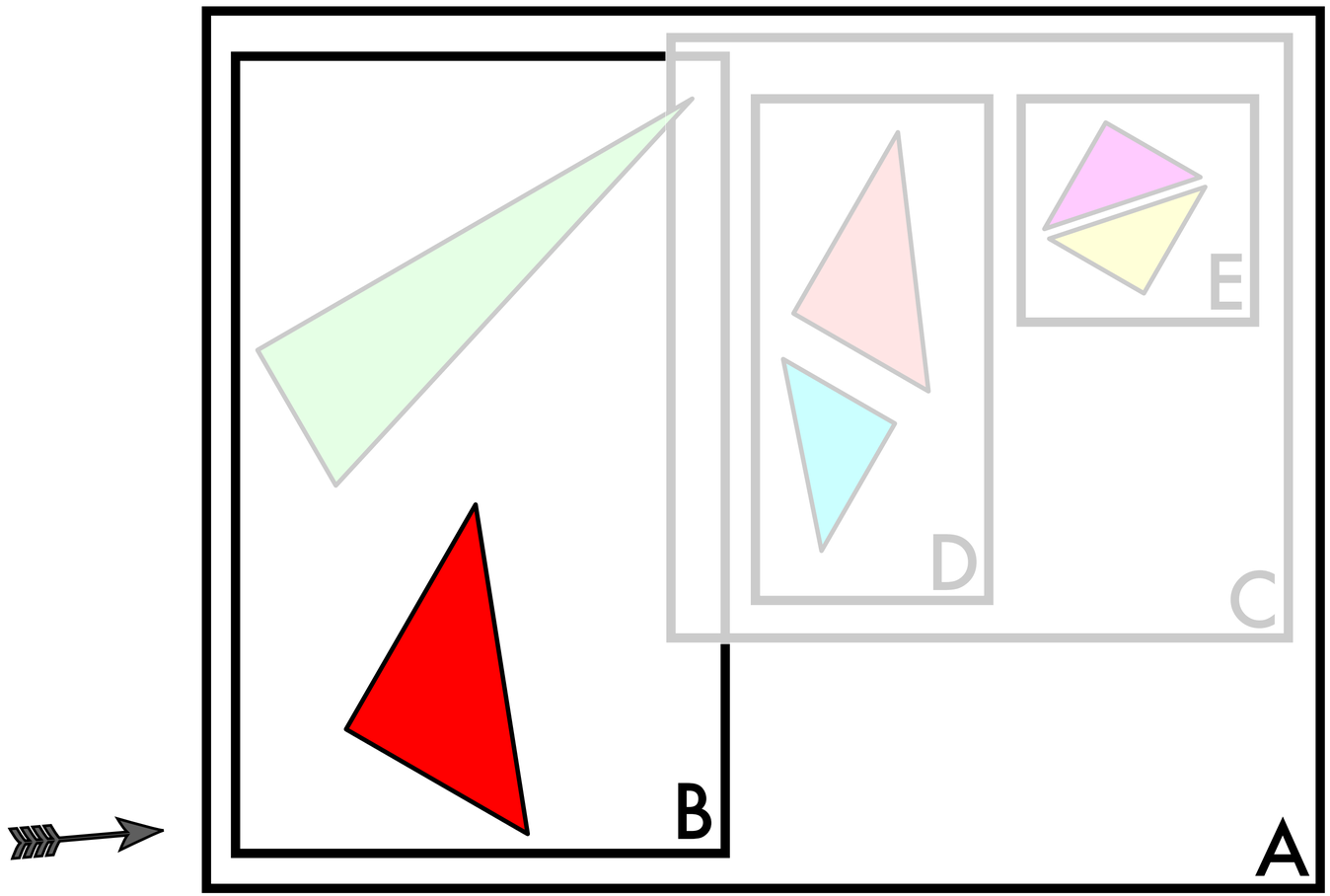


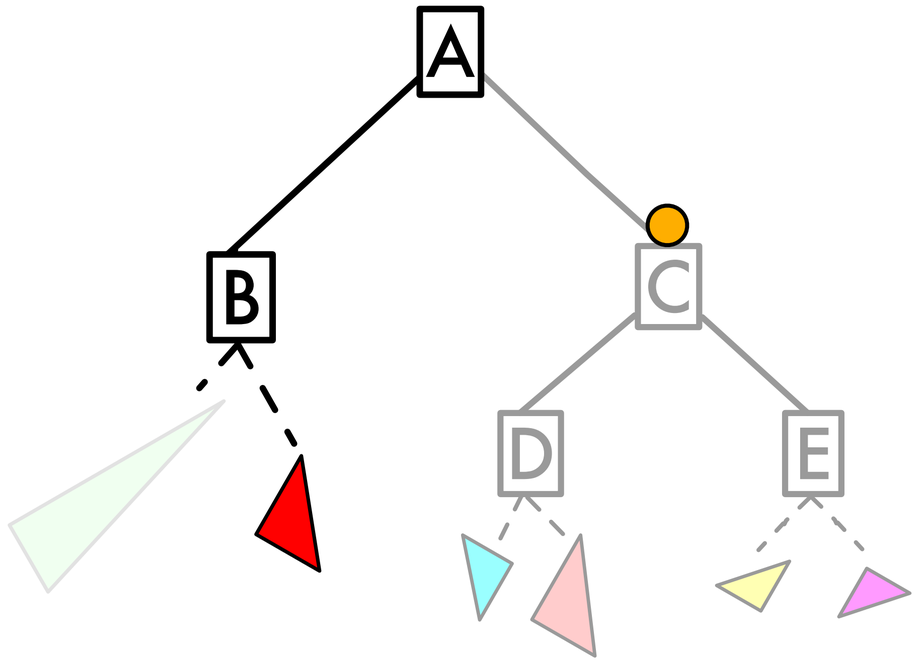
Which triangles does the ⟶ intersect?



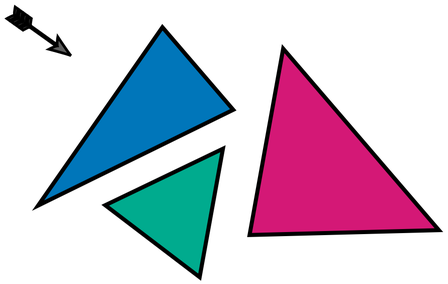


Which triangles does the ⟶ intersect?

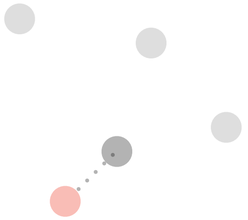




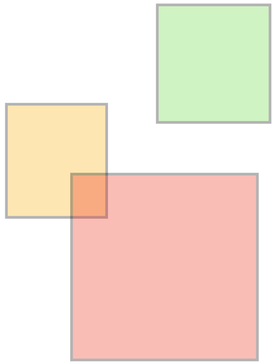
BVHs are used in numerous domains!



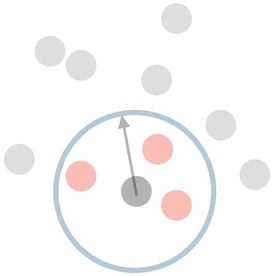
ray tracing



closest point query



collision detection

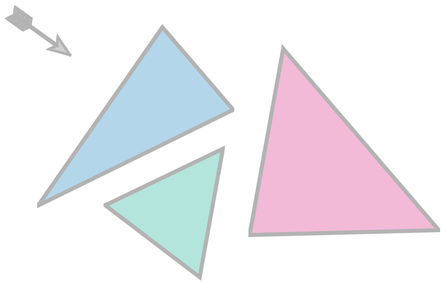


neighbor search

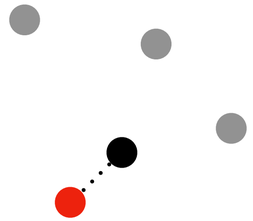
SPHRAY: A Smoothed Particle Hydrodynamics Ray Tracer for Radiative Transfer

Gabriel Altay[1], Rupert A.C. Croft[1], and Inti Pelupessy[1]
[1] *Carnegie Mellon University, Department of Physics, 5000 Forbes Avenue, Pittsburgh PA 15213, USA*

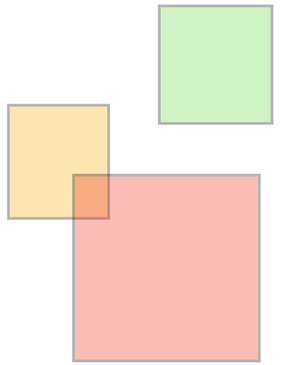
BVHs are used in numerous domains!



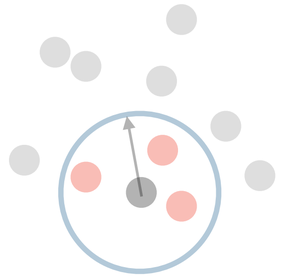
ray tracing



closest point query



collision detection

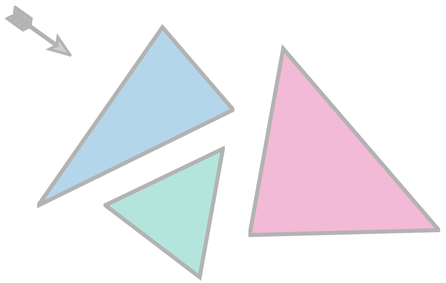


neighbor search

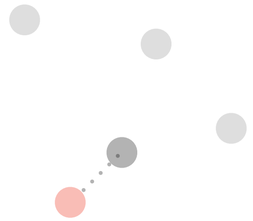
Fast Point to Mesh Distance by Domain Voxelization

Geordan Gutow and Howie Choset

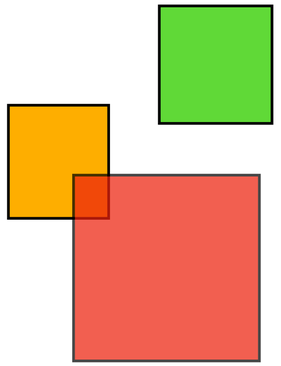
BVHs are used in numerous domains!



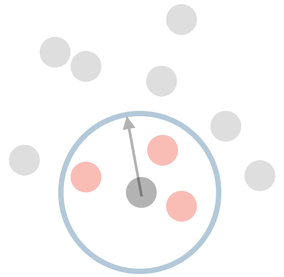
ray tracing



closest point query



collision detection

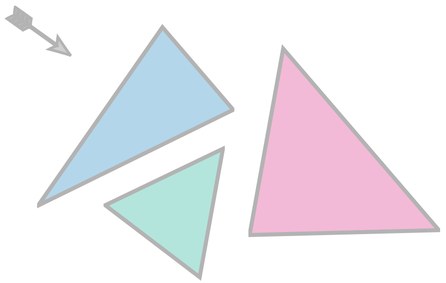


neighbor search

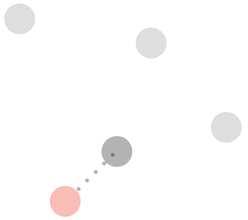
Efficient Configuration Space Construction and Optimization for Motion Planning

Jia Pan[1*] and Dinesh Manocha[2]

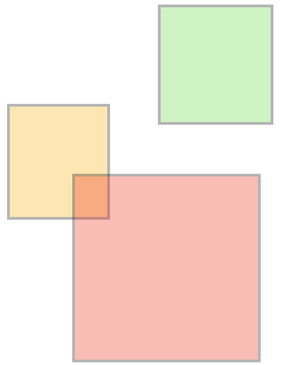
BVHs are used in numerous domains!



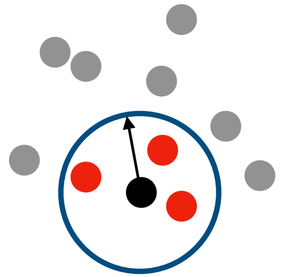
ray tracing



closest point query



collision detection

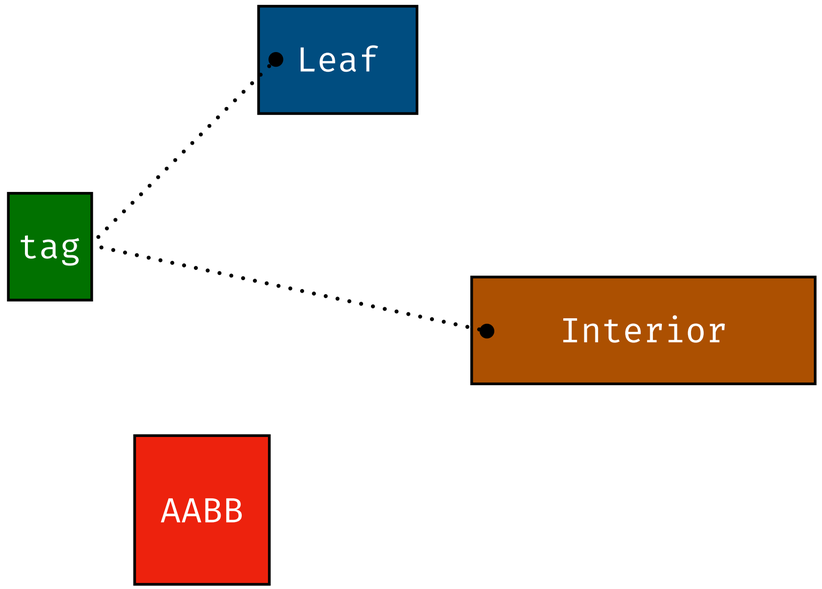


neighbor search

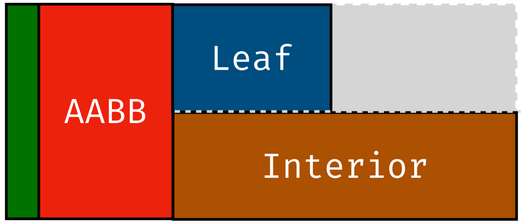
Quantized bounding volume hierarchies for neighbor search in molecular simulations on graphics processing units

Michael P. Howard[a,*], Antonia Statt[b], Felix Madutsa[b], Thomas M. Truskett[a], Athanassios Z. Panagiotopoulos[b]

BVHs are *physically* represented many different ways



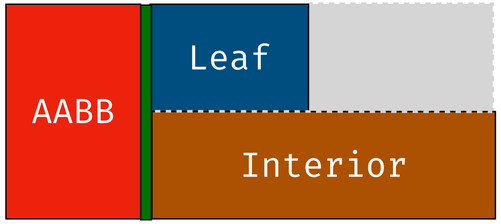
BVHs are *physically* represented many different ways




```
struct Node {
  uint8 tag;
  AABB bbox;
  union {
    Leaf leaf;
    Interior interior;
  };
};
```

BVHs are *physically* represented many different ways

Intra-Node Optimization



Bit-Stealing Made Legal

Compilation for Custom Memory Representations of Algebraic Data Types

THAÏS BAUDON, Univ Lyon, France, EnsL, France, UCBL, France, CNRS, France, Inria, France, and LIP, France
GABRIEL RADANNE, Inria, France, EnsL, France, UCBL, France, CNRS, France, and LIP, France
LAURE GONNORD, UGA, France, Grenoble INP, France, LCIS, France, and LIP, France

```
split .[0, 1] {
| 0 from Node => &<64, 2>(
  {{(_.v as i64),(_.l as RBT),(_.r as RBT)}}
  ) with .[0,1]:(=0) with .[1,2]:(_.c as Color)
| 0 from Empty => i64 with . : (= 0)
```

DARGENT: A Silver Bullet for Verified Data Layout Refinement

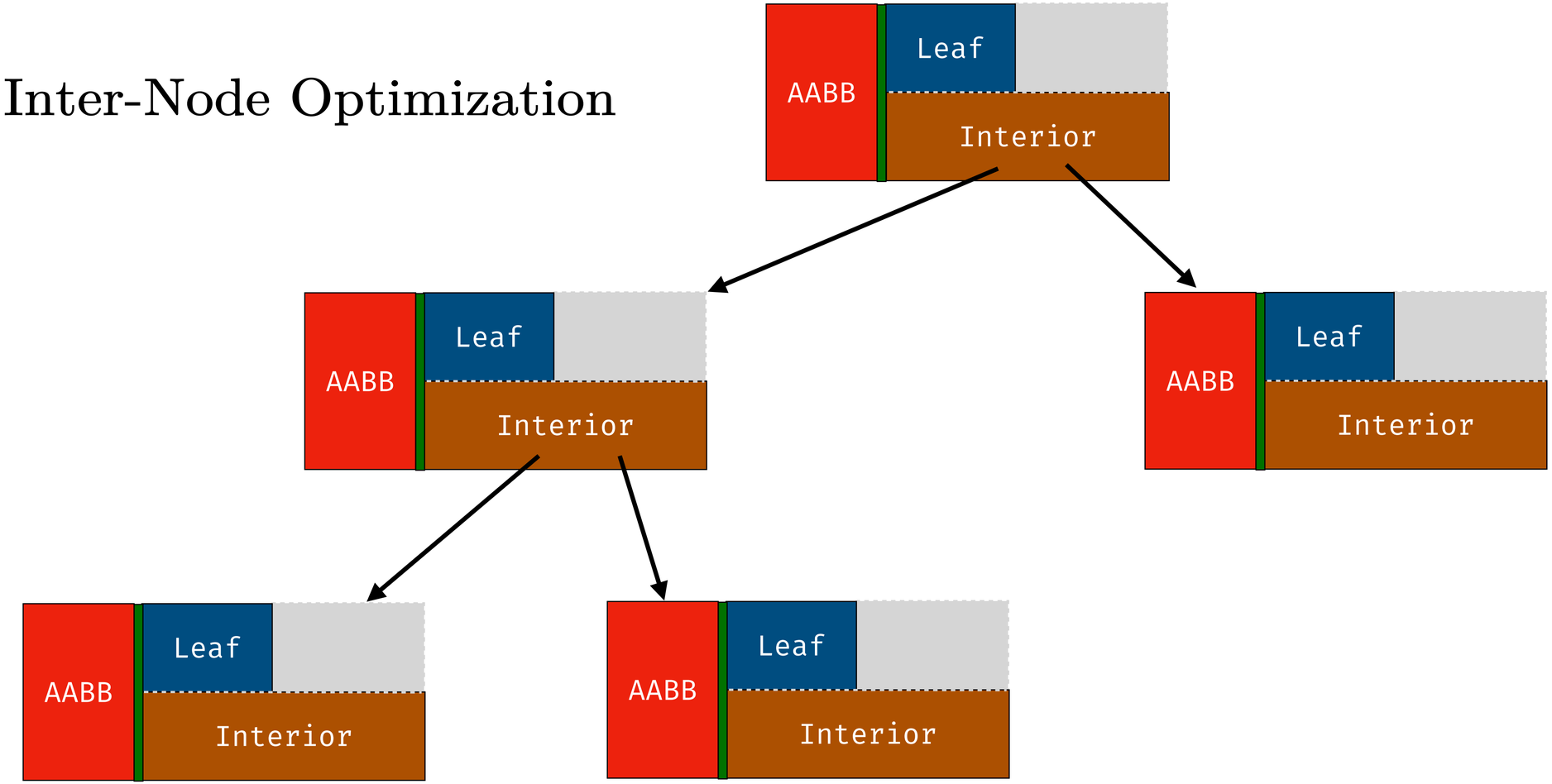
ZILIN CHEN, UNSW Sydney, Australia
AMBROISE LAFONT, University of Cambridge, UK
LIAM O'CONNOR, University of Edinburgh, UK
GABRIELE KELLER, Utrecht University, Netherlands
CRAIG MCLAUGHLIN, UNSW Sydney, Australia
VINCENT JACKSON, University of Melbourne, Australia
CHRISTINE RIZKALLAH, University of Melbourne, Australia

```
struct : record {a : 4B, b : 1b},
ptr : pointer at 8B,
sum : variant (1b)
  {A(0) : 2B at 1B, B(1) : 1B at 1B} at 5B
```

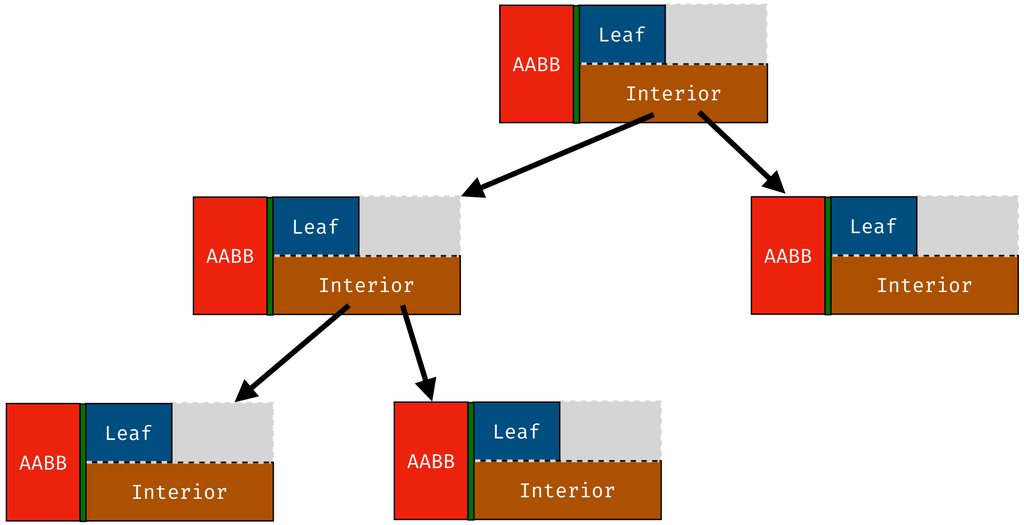
```
struct Node {
  AABB bbox;
  uint1 tag;
  union {
    Leaf leaf;
    Interior interior;
  };
};
```

BVHs are *physically* represented many different ways

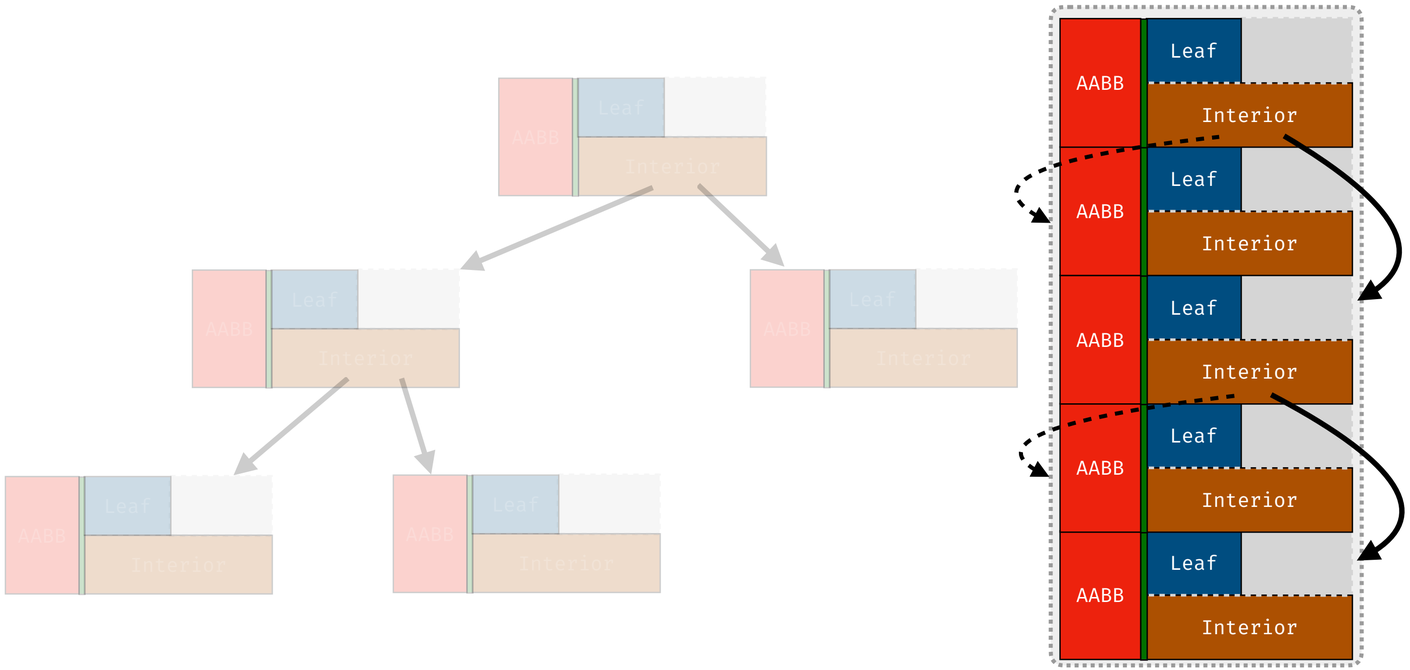
Inter-Node Optimization



BVHs are *physically* represented many different ways

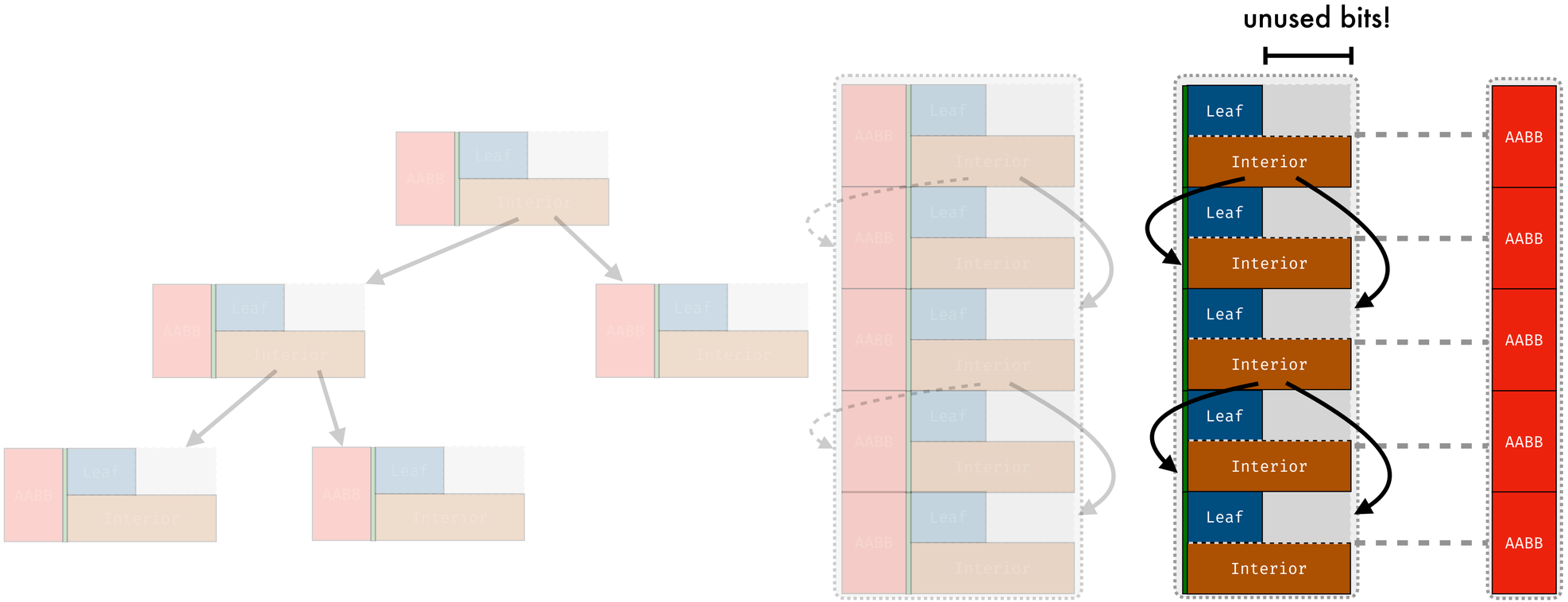


BVHs are *physically* represented many different ways



Locality

BVHs are *physically* represented many different ways

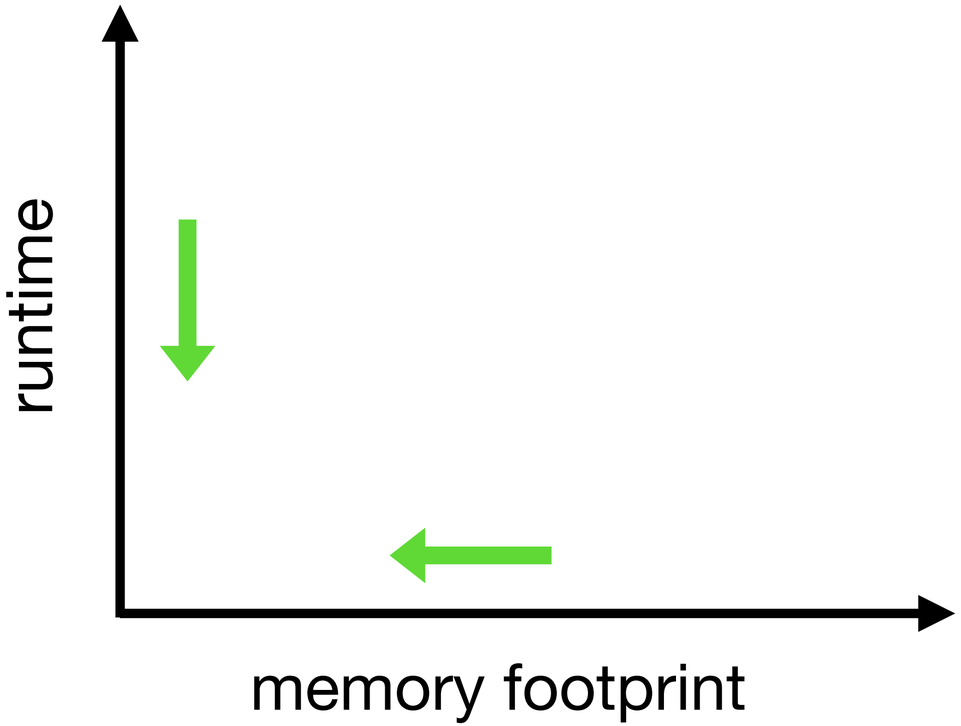


Cache-friendly

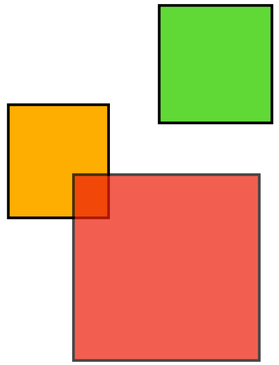
BVHs are *physically* represented many different ways

Space-efficient

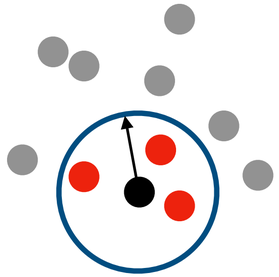
Implementation of BVHs should minimize two objectives



The objectives are dependent on three factors

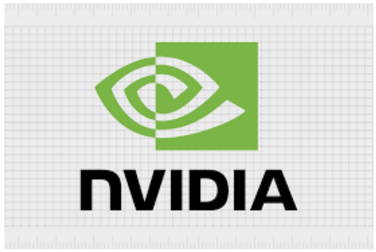


collision detection

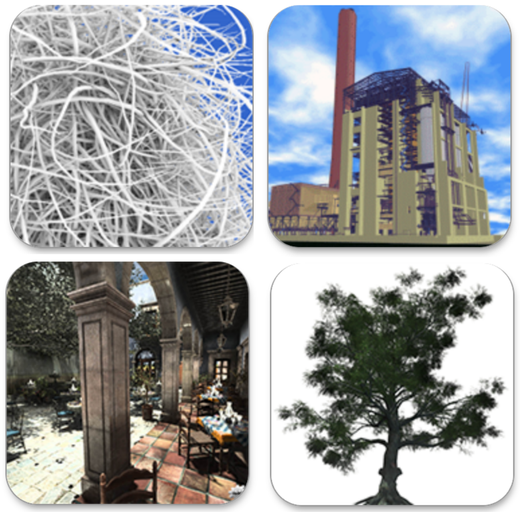


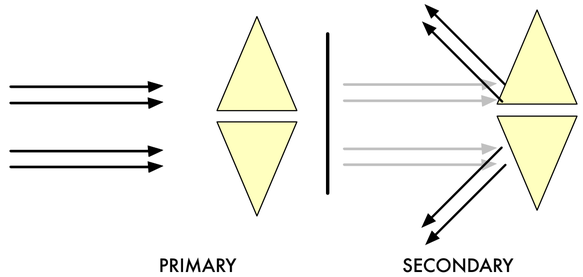
neighbor search

Algorithm



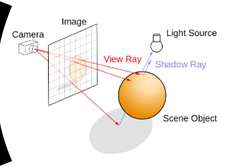
Machine





Data

Each prior work is *one point* in this design space

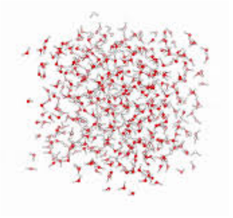
(

, x86 , )

Exploiting Local Orientation Similarity for Efficient Ray Traversal of Hair and Fur

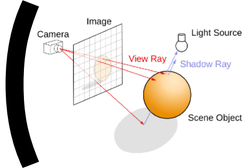
Sven Woop†, Carsten Benthin†, Ingo Wald†, Gregory S. Johnson†, and Eric Tabellion‡

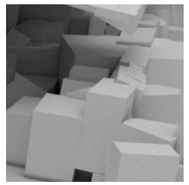
(

, NVIDIA CUDA , )

Quantized bounding volume hierarchies for neighbor search in molecular simulations on graphics processing units

Michael P. Howard[a,*], Antonia Statt[b], Felix Madutsa[b], Thomas M. Truskett[a], Athanassios Z. Panagiotopoulos[b]

(

, NVIDIA CUDA , )

SOBB: Skewed Oriented Bounding Boxes for Ray Tracing

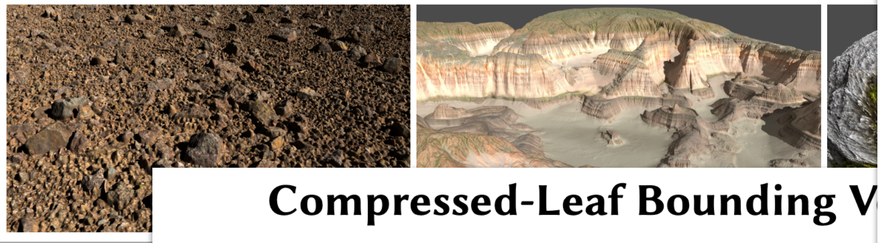
M. Káčerik and J. Bittner

Each prior work is *one point* in this design space

Ray Tracing Lossy Compressed Grid Primitives

Carsten Benthin Karthik Vaidyanathan Sven Wo

Intel Corporation



Compressed-Leaf Bounding V

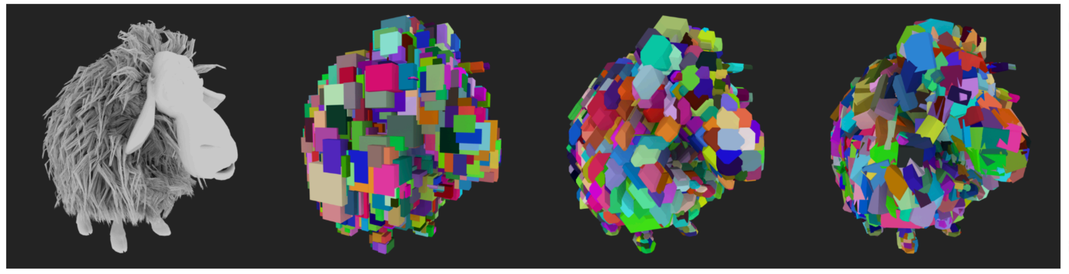
Carsten Benthin
Intel Corporation

Ingo Wald
Intel Corporation

Sv
Intel Corporation

SAH-Optimized *k*-DOP Hierarchies for Ray Tracing

MARTIN KÁČERIK, Czech Technical University in Prague, Czechia
JIŘÍ BITTNER, Czech Technical University in Prague, Czechia



Intel Corporation

Memory-Conserving Bounding Volume Hierarchies
aytracing

Wyvill

Alberta, Canada
ucalgary.ca

Reduced Precision

and M. Salvi

[1]Intel Corporation

Bandwidth-Efficient BVH Layout
for Incremental Hardwa

G. Liktor and K. Vaidyanath

AQB8: Energy-Efficient Ray Tracing Accelerator through Multi-Level Quantization

Yen-Chieh Huang
National Yang Ming Chiao Tung University
Hsinchu, Taiwan
lashhw.cs09@nycu.edu.tw

Chen-Pin Yang
National Yang Ming Chiao Tung University
Hsinchu, Taiwan
ycpin.cs12@nycu.edu.tw

Tsung Tai Yeh
National Yang Ming Chiao Tung University
Hsinchu, Taiwan
ttyeh@cs.nycu.edu.tw

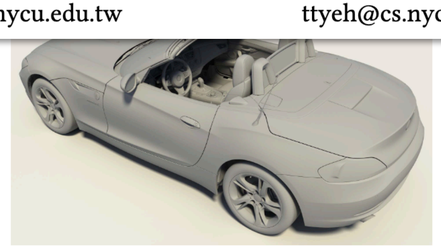
with Hierarchical Mesh Quantization

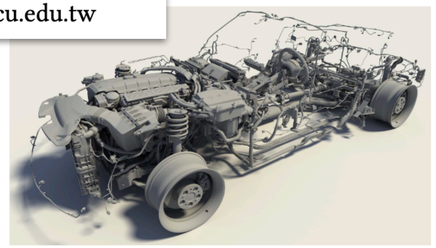
a* Manfred Ernst†
Intel Labs

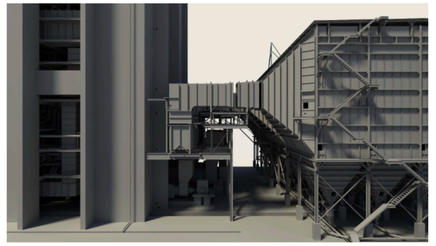
Bounding Volume Hierarchies of Slab Cut Balls

T. Larsson[1] and T. Akenine-Möller[2]

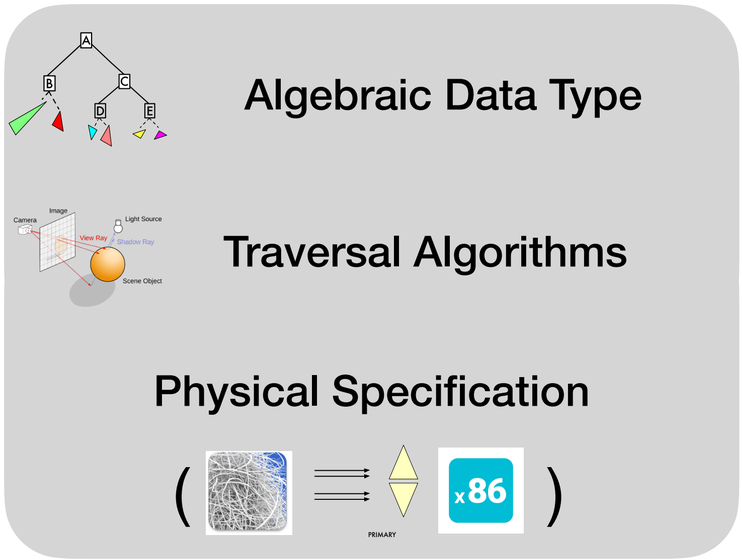
[1]Mälardalen University, Sweden
[2]Lund University, Sweden

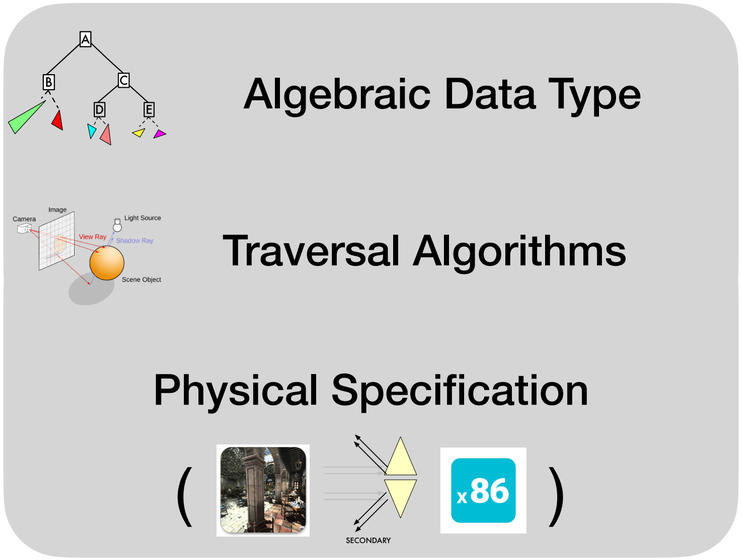


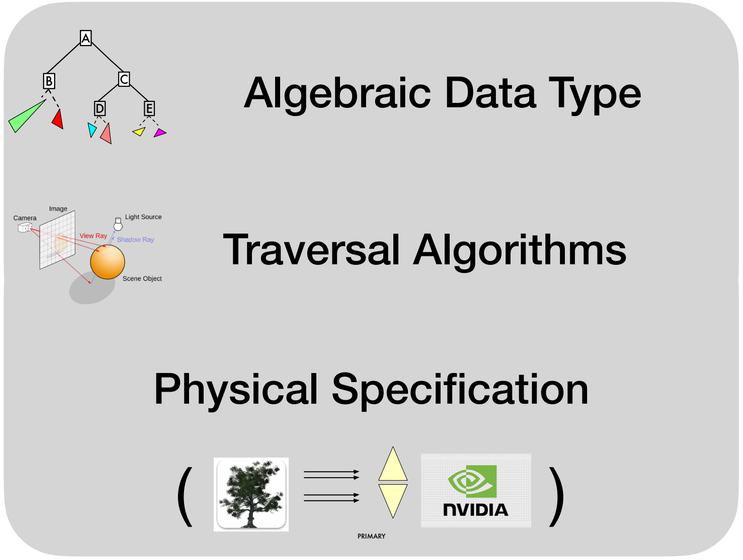




Goal: design a system that aids this exploration

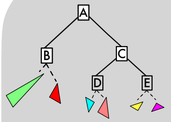




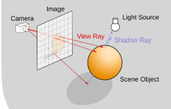


Goal: design a system that aids this exploration

Domain Expert



Algebraic Data Type



Traversal Algorithms

Physical Specification

( PRIMARY x86)

Physical Specification

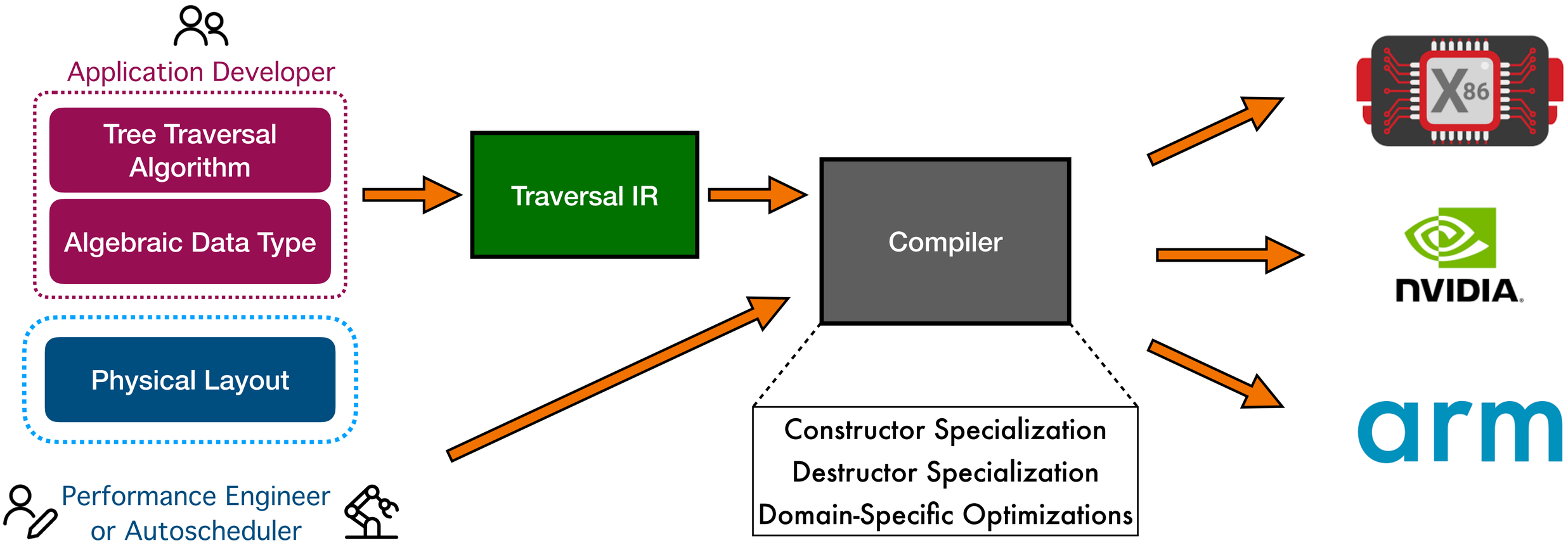
( SECONDARY x86)

Physical Specification

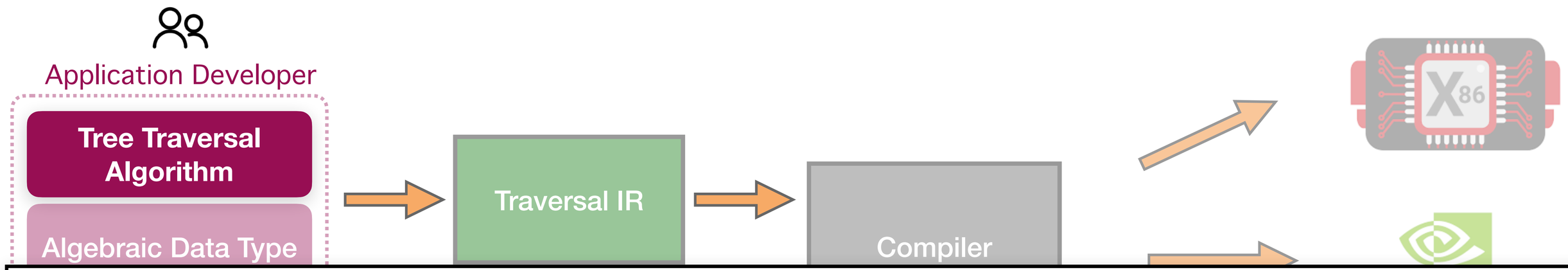
( PRIMARY NVIDIA)

Performance Engineer

System Overview



System Overview



Efficient tree traversal algorithms are **automatically** generated from `Bonsai`

16:10 - 17:30 Optimization of Data Pipelines at Flatirons 2 — PLDI Research Papers

16:10 20m Talk ☆ **[SIGPLAN OOPSLA'25] Homomorphism Calculus for User-Defined Aggregations**
Ziteng Wang University of Texas at Austin, Ruijie Fang University of Texas at Austin, Linus Zheng University of Texas at Austin, Dixin Tang University of Texas at Austin, Işıl Dillig University of Texas at Austin

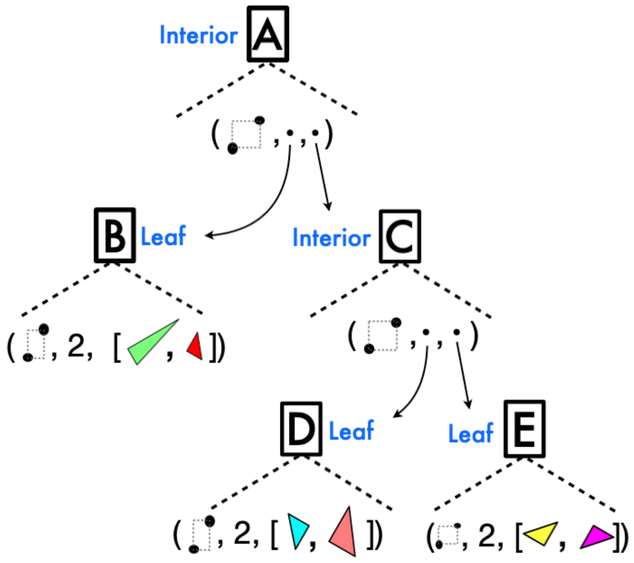
16:30 20m Talk ★ **Bonsai: Compiling Queries to Pruned Tree Traversals** DISTINGUISHED PAPER
Alexander J Root Stanford University, Christophe Gyurgyik Stanford University, Purvi Goel Stanford University, Kayvon Fatahalian Stanford University, Jonathan Ragan-Kelley Massachusetts Institute of Technology, Andrew Adams Adobe Research, Fredrik Kjolstad Stanford University
DOI Pre-print

16:50 20m Talk ☆ **A Compiler for Fused Relational Operations on Multisets**
James Dong Stanford University, Fredrik Kjolstad Stanford University
DOI

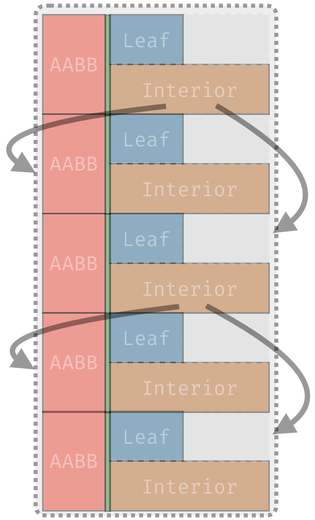
17:10 20m Talk ☆ **Optimal Predicate Pushdown Synthesis**
Robert Zhang University of Texas at Austin, Eric Hayden Campbell University of Texas at Austin, Dixin Tang University of Texas at Austin, Işıl Dillig University of Texas at Austin
DOI

Specifying the logical interface

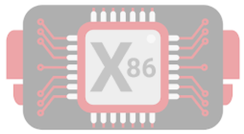
`type BVH`



`layout BVH`

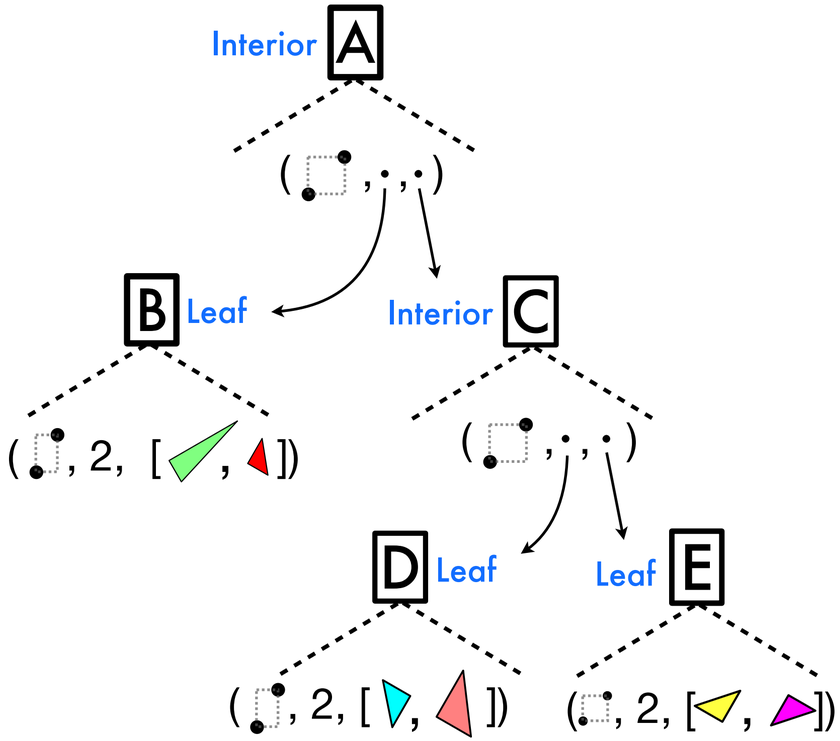


`machine code`



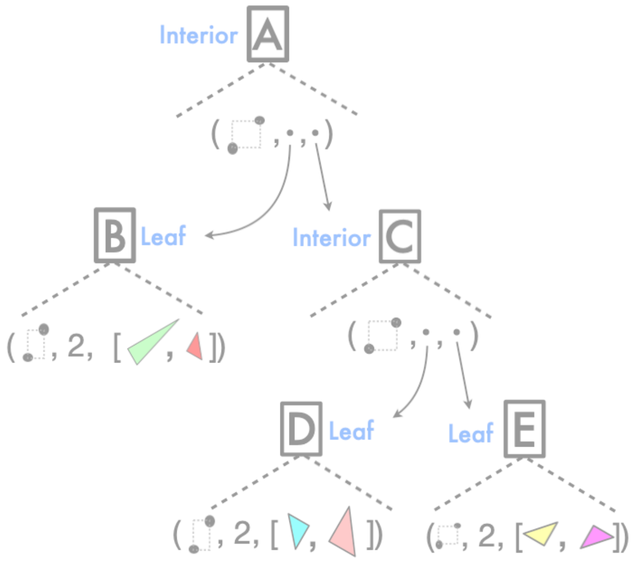
BVH applications can be written agnostic to physical representation

```
type AABB(low: f32×3, high: f32×3);
type BVH = Interior(bbox: AABB, left: BVH, right: BVH)
         | Leaf(bbox: AABB, n: u16, data: Triangle[n]);
-----------------------------------------------------------------
func closest_hit(ray: Ray, bvh: BVH, best: mut (f32, Triangle)) =
  match bvh:
  | Interior(bbox, L, R) →
    if worth_exploring(ray, bbox, best):
      closest_hit(ray, L, best); closest_hit(ray, R, best)
  | Leaf(bbox, n, data) →
    if worth_exploring(ray, bbox, best):
      closest_triangle(ray, n, data, best)
```

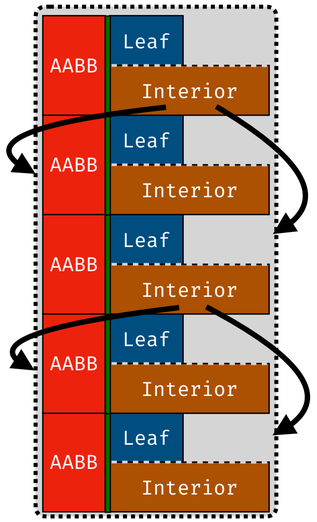


Specifying the physical interface

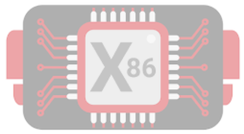
`type BVH`



`layout BVH`



`machine code`



Defining a layout

```
type AABB(low: f32×3, high: f32×3);

type BVH = Interior(bbox: AABB, left: BVH, right: BVH)

         | Leaf(bbox: AABB, n: u16, data: Triangle[n]);
```

```
layout LinearBVH(i: u32) {
 M: u32;
 △s: Triangle[M];
}
```

Defining a layout

```
layout LinearBVH(i: u32) {
 N: u32; M: u32;
 △s: Triangle[M];
 group Nodes[N] by i {

 }
}
```

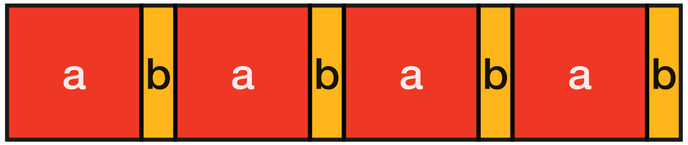
```
type AABB(low: f32×3, high: f32×3);

type BVH = Interior(bbox: AABB, left: BVH, right: BVH)

         | Leaf(bbox: AABB, n: u16, data: Triangle[n]);
```

Defining a layout

```
layout LinearBVH(i: u32) {
 N: u32; M: u32;
 △s: Triangle[M];
 group Nodes[N] by i {

 }
}
```

```
type AABB(low: f32×3, high: f32×3);

type BVH = Interior(bbox: AABB, left: BVH, right: BVH)

         | Leaf(bbox: AABB, n: u16, data: Triangle[n]);
```

```
group G1[4] by i { a: f32; b: i8 }
```

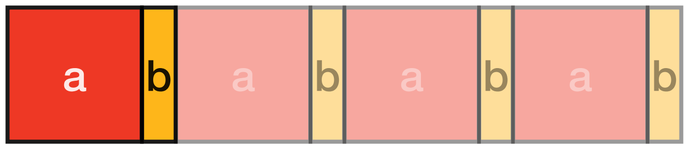


Defining a layout

```
layout LinearBVH(i: u32) {
 N: u32; M: u32;
 △s: Triangle[M];
 group Nodes[N] by i {

 }
}
```

```
type AABB(low: f32×3, high: f32×3);

type BVH = Interior(bbox: AABB, left: BVH, right: BVH)

         | Leaf(bbox: AABB, n: u16, data: Triangle[n]);
```

```
group G1[4] by i { a: f32; b: i8 }
```



i = 0

Defining a layout

```
type AABB(low: f32×3, high: f32×3);

type BVH = Interior(bbox: AABB, left: BVH, right: BVH)
         | Leaf(bbox: AABB, n: u16, data: Triangle[n]);
```

```
layout LinearBVH(i: u32) {
 N: u32; M: u32;
 △s: Triangle[M];
 group Nodes[N] by i {

 }
}
```

```
group G1[4] by i { a: f32; b: i8 }
```

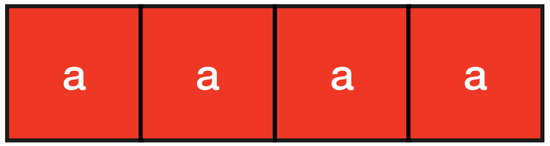
i = 1

Defining a layout

```
type AABB(low: f32×3, high: f32×3);

type BVH = Interior(bbox: AABB, left: BVH, right: BVH)
         | Leaf(bbox: AABB, n: u16, data: Triangle[n]);
```

```
layout LinearBVH(i: u32) {
 N: u32; M: u32;
 △s: Triangle[M];
 group Nodes[N] by i {

 }
}
```

```
group G1[4] by i { a: f32 }
group G2[4] by i { b: i8  }
```



Defining a layout

```
type AABB(low: f32×3, high: f32×3);

type BVH = Interior(bbox: AABB, left: BVH, right: BVH)
         | Leaf(bbox: AABB, n: u16, data: Triangle[n]);
```

```
layout LinearBVH(i: u32) {
 N: u32; M: u32;
 △s: Triangle[M];
 group Nodes[N] by i {

 }
}
```

```
group Go[4] by i {
  group Gi1[2] by j { a: f32 }
  group Gi2[2] by j { b: i8  }
}
```

Defining a layout

```
layout LinearBVH(i: u32) {
 N: u32; M: u32;
 △s: Triangle[M];
 group Nodes[N] by i {
   bbox: AABB;
   n: u16; // # triangles
 }
}
```

```
type AABB(low: f32×3, high: f32×3);

type BVH = Interior(bbox: AABB, left: BVH, right: BVH)

         | Leaf(bbox: AABB, n: u16, data: Triangle[n]);
```

Defining a layout

```
layout LinearBVH(i: u32) {
 N: u32; M: u32;
 △s: Triangle[M];
 group Nodes[N] by i {
   bbox: AABB;
   n: u16; // # triangles
   split n {
     > 0 → Leaf     { }
       0 → Interior { }
   }
 }
}
```

```
type AABB(low: f32×3, high: f32×3);

type BVH = Interior(bbox: AABB, left: BVH, right: BVH)

         | Leaf(bbox: AABB, n: u16, data: Triangle[n]);
```

Defining a layout

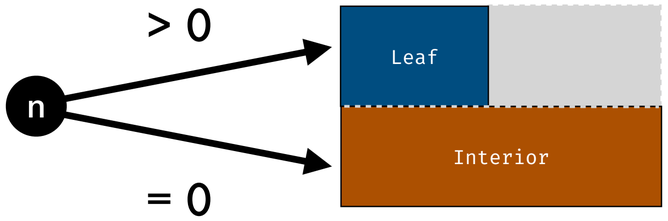
```
type AABB(low: f32×3, high: f32×3);

type BVH = Interior(bbox: AABB, left: BVH, right: BVH)
         | Leaf(bbox: AABB, n: u16, data: Triangle[n]);
```

```
layout LinearBVH(i: u32) {
 N: u32; M: u32;
 △s: Triangle[M];
 group Nodes[N] by i {
   bbox: AABB;
   n: u16; // # triangles
   split n {
     > 0 → Leaf     { }
       0 → Interior { }
   }
 }
}
```

Bit-Stealing Made Legal

Compilation for Custom Memory Representations of Algebraic Data Types

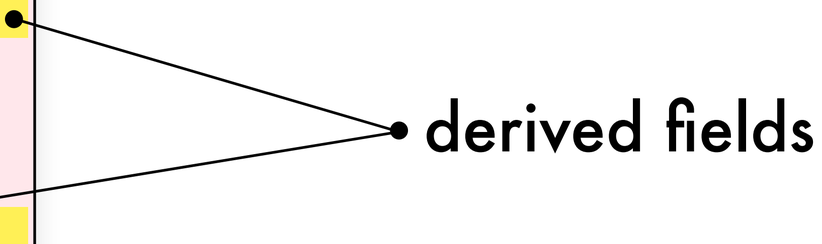
THAÏS BAUDON, Univ Lyon, France, EnsL, France, UCBL, France, CNRS, France, Inria, France, and LIP, France
GABRIEL RADANNE, Inria, France, EnsL, France, UCBL, France, CNRS, France, and LIP, France
LAURE GONNORD, UGA, France, Grenoble INP, France, LCIS, France, and LIP, France



Defining a layout

```
type AABB(low: f32×3, high: f32×3);

type BVH = Interior(bbox: AABB, left: BVH, right: BVH)
         | Leaf(bbox: AABB, n: u16, data: Triangle[n]);
```

```
layout LinearBVH(i: u32) {
 N: u32; M: u32;
 △s: Triangle[M];
 group Nodes[N] by i {
   bbox: AABB;
   n: u16; // # triangles
   split n {
     > 0 → Leaf     {
       idx: u32;
       data = △s[idx : idx + n];
     }
       0 → Interior {
       right: u32;
       left = i + 1;
     }
   }
 }
}
```



Defining a layout

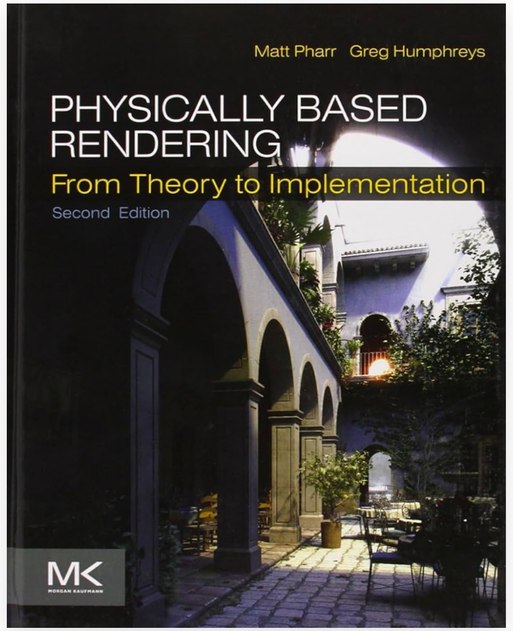
```
layout LinearBVH(i: u32) {
 N: u32; M: u32;
 △s: Triangle[M];
 group Nodes[N] by i {
   bbox: AABB;
   n: u16; // # triangles
   16;
   split n {
     > 0 → Leaf     {
       idx: u32;
       data = △s[idx : idx + n];
     }
       0 → Interior {
       right: u32;
       left = i + 1;
     }
   }
 }
}
```

```
type AABB(low: f32×3, high: f32×3);

type BVH = Interior(bbox: AABB, left: BVH, right: BVH)

         | Leaf(bbox: AABB, n: u16, data: Triangle[n]);
```

Defining a layout

```
layout LinearBVH(i: u32) {
 N: u32; M: u32;
 △s: Triangle[M];
 group Nodes[N] by i {
   bbox: AABB;
   n: u16; // # triangles
   16;
   split n {
     > 0 → Leaf     {
       idx: u32;
       data = △s[idx : idx + n];
     }
       0 → Interior {
       right: u32;
       left = i + 1;
     }
   }
 }
}
```

```
type AABB(low: f32×3, high: f32×3);

type BVH = Interior(bbox: AABB, left: BVH, right: BVH)

         | Leaf(bbox: AABB, n: u16, data: Triangle[n]);
```



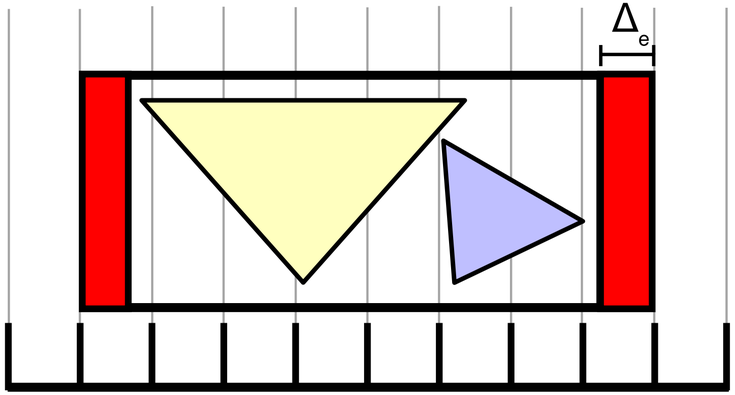
PBRT LinearBVHv4

Sufficiently expressive to realize many layout optimizations

Sufficiently expressive to realize many layout optimizations

Bounding Volume Quantization

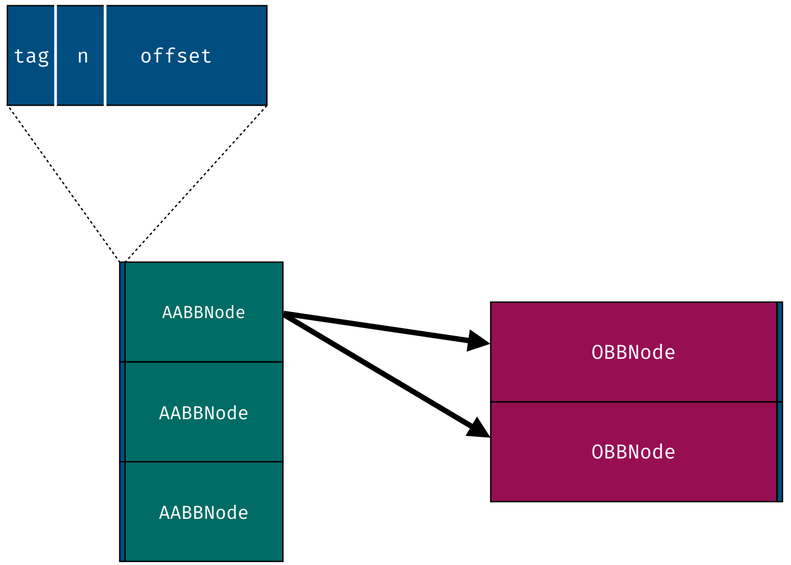
```
wlow: f32×3; whigh: f32×3; bins: f32×3;
group qnodes[N] by i {
 q_min: u30; q_max: u30;
 n: u4;
 low  = fadd_rd(wlow,  dequantize(q_min, bins));
 high = fsub_ru(whigh, dequantize(q_max, bins));
}
```



Sufficiently expressive to realize many layout optimizations

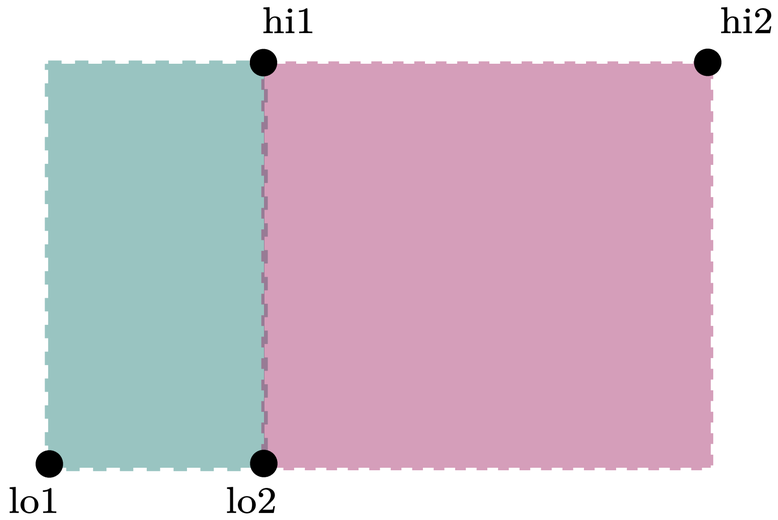
Sum of Arrays

```
offset = index[5:63];
split index[0:1] {
  0b'11 → AABB from AABBs[offset];
  0b'01 → OBB  from OBBs[offset];
  _     → Leaf {
    n = index[1:4] + 1;
    data = △s[offset : offset + n];
  }
}
```

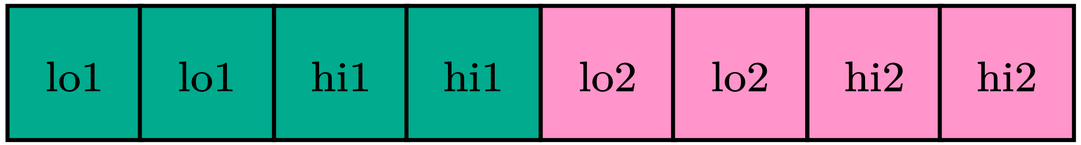


Sufficiently expressive to realize many layout optimizations

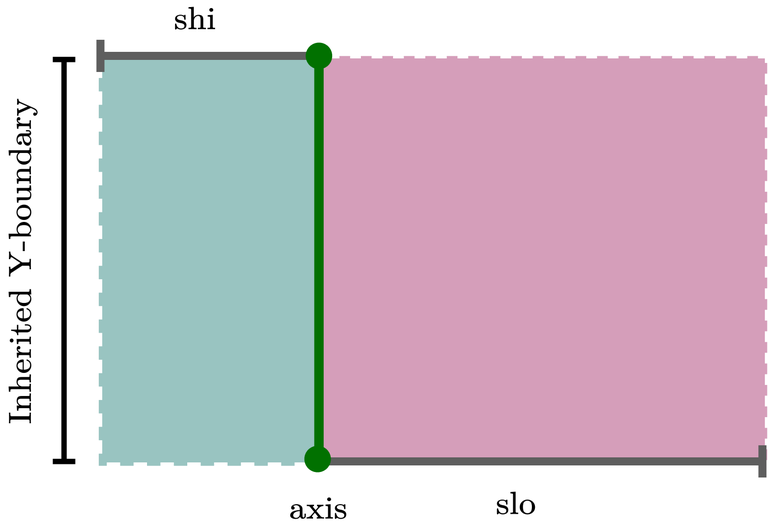
Shared Slab Optimization



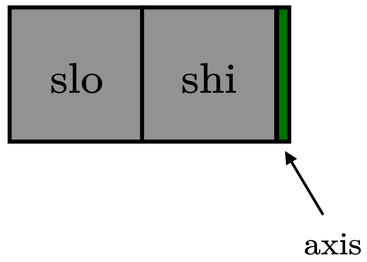
32B




```
lo: f32×2
hi: f32×2
```



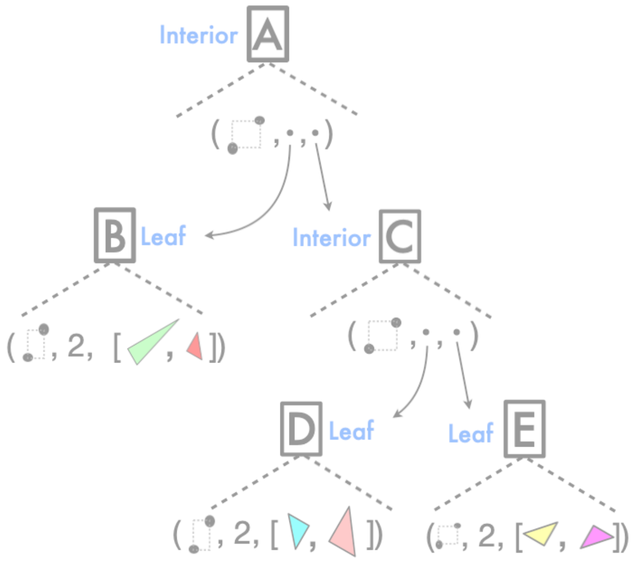
9B



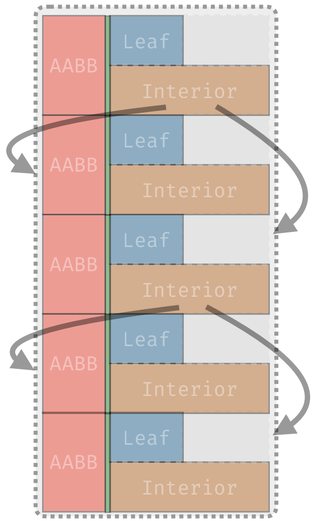

```
axis: u2;
slo: f32; shi: f32;
parent.plo[axis] = slo;
parent.phi[axis] = shi;
```

Automatically specializing the layout

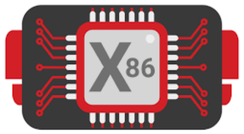
`type BVH`



`layout BVH`



`machine code`



Destructor Specialization

```
| Interior(box, left, right) →
  …
  closest_hit(
    ray,
    ?right?,
    best
  )
…
```

Destructor Specialization

```
| Interior(box, left, right) →
  …
  closest_hit(
    ray,
    ?right?,
    best
  )
…
```

```
layout LinearBVH(i: u32) {
 N: u32; M: u32;
 △s: Triangle[M];
 group Nodes[N] by i {
   bbox: AABB;
   n: u16; // # triangles
   split n {
     > 0 → Leaf     {
       idx: u32;
       data = △s[idx : idx + n];
     }
       0 → Interior {
       right: u32;
       left = i + 1;
     }
   }
 }
}
```

Destructor Specialization

```
| Interior(box, left, right) →
  …
  closest_hit(
    ray,
    LinearBVH,
    best
  )
…
```

```
layout LinearBVH(i: u32) {
 N: u32; M: u32;
 △s: Triangle[M];
 group Nodes[N] by i {
   bbox: AABB;
   n: u16; // # triangles
   split n {
     > 0 → Leaf     {
       idx: u32;
       data = △s[idx : idx + n];
     }
       0 → Interior {
       right: u32;
       left = i + 1;
     }
   }
 }
}
```

Destructor Specialization

```
| Interior(box, left, right) →
  …
  closest_hit(
    ray,
    LinearBVH.Nodes[i],
    best
  )
…
```

```
layout LinearBVH(i: u32) {
 N: u32; M: u32;
 △s: Triangle[M];
 group Nodes[N] by i {
   bbox: AABB;
   n: u16; // # triangles
   split n {
     > 0 → Leaf     {
       idx: u32;
       data = △s[idx : idx + n];
     }
       0 → Interior {
       right: u32;
       left = i + 1;
     }
   }
 }
}
```

Destructor Specialization

```
| Interior(box, left, right) →
  …
  closest_hit(
    ray,
    LinearBVH.Nodes[i].as(Interior),
    best
  )
…
```

```
layout LinearBVH(i: u32) {
 N: u32; M: u32;
 △s: Triangle[M];
 group Nodes[N] by i {
   bbox: AABB;
   n: u16; // # triangles
   split n {
     > 0 → Leaf     {
       idx: u32;
       data = △s[idx : idx + n];
     }
       0 → Interior {
       right: u32;
       left = i + 1;
     }
   }
 }
}
```

Destructor Specialization

```
| Interior(box, left, right) →
  …
  closest_hit(
    ray,
    LinearBVH.Nodes[i].as(Interior).right,
    best
  )
…
```

```
layout LinearBVH(i: u32) {
 N: u32; M: u32;
 △s: Triangle[M];
 group Nodes[N] by i {
   bbox: AABB;
   n: u16; // # triangles
   split n {
     > 0 → Leaf     {
       idx: u32;
       data = △s[idx : idx + n];
     }
       0 → Interior {
       right: u32;
       left = i + 1;
     }
   }
 }
}
```

Destructor Specialization

```
if (!(LinearBVH.Nodes[i].n > 0))
  …
  closest_hit(
    ray,
    LinearBVH.Nodes[i].as(Interior).right,
    best
  )
…
```

```
layout LinearBVH(i: u32) {
 N: u32; M: u32;
 △s: Triangle[M];
 group Nodes[N] by i {
   bbox: AABB;
   n: u16; // # triangles
   split n {
     > 0 → Leaf     {
       idx: u32;
       data = △s[idx : idx + n];
     }
       0 → Interior {
       right: u32;
       left = i + 1;
     }
   }
 }
}
```

Destructor Specialization

```
let _x0 = LinearBVH.Nodes[i] in
if (!(_x0.n > 0))
  …
  closest_hit(
    ray,
    _x0.as(Interior).right,
    best
  )
…
```

```
layout LinearBVH(i: u32) {
 N: u32; M: u32;
 △s: Triangle[M];
 group Nodes[N] by i {
   bbox: AABB;
   n: u16; // # triangles
   split n {
     > 0 → Leaf     {
       idx: u32;
       data = △s[idx : idx + n];
     }
       0 → Interior {
       right: u32;
       left = i + 1;
     }
   }
 }
}
```

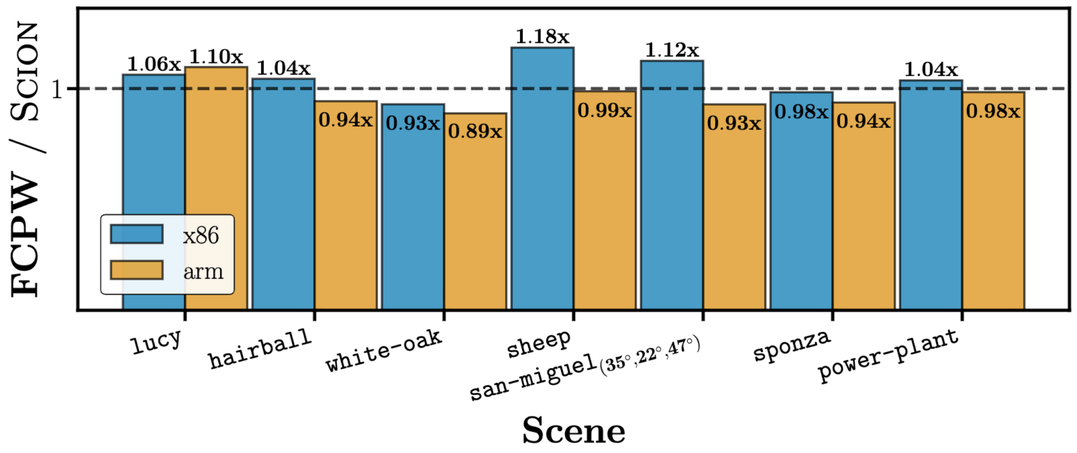
We also perform Constructor Specialization

(see paper for details)

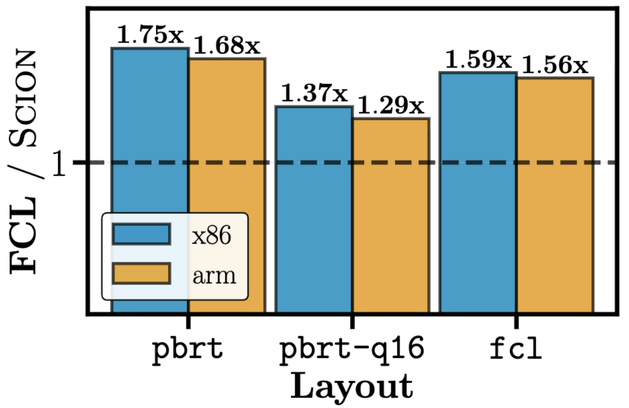
```
build BVH[order=pre] {
  build Interior(bounds: AABB, left: BVH, right: BVH) {
    build bounds;
    build n = 0;
    build left; build right;
    return this;
  }
  build Leaf(bounds: AABB, n: u16, data: Triangle[n]) {
    build bounds;
    build n;
    build o = append(data, n);
    return this;
  }
}
```

These techniques are ~parity with SoTA

(higher is better)

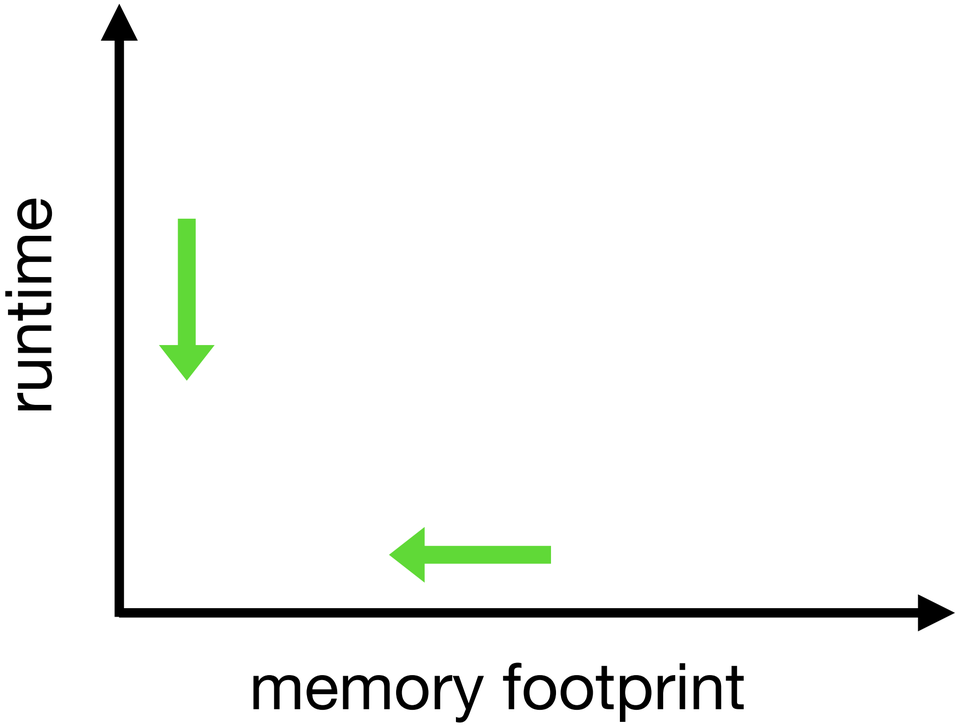


Closest Point Query



Collision Detection

Design Space Exploration: Closest Hit Ray Tracing

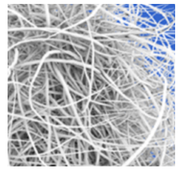


Design Space Exploration: Closest Hit Ray Tracing

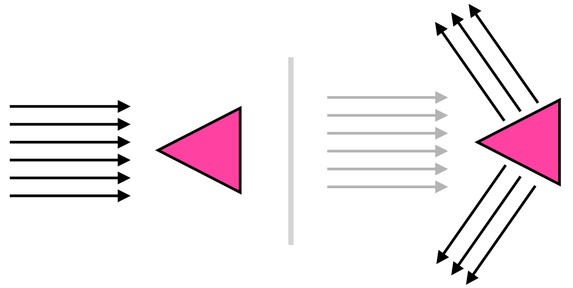
2-ary Layout	Node Size (B)	Description
pbrt	32	PBRTv4 LinearBVH [75]
ptr	48	pbrt + children pointers
pbrt-align16	32	pbrt + 16B align
pbrt-soaos	32	pbrt + SoAoS
pbrt-soaos-align16	32	pbrt-soaos + 16B align
sg-pbrt-q16	16	novel (Section 8.2.5)
sg-pbrt-q16-soaos	16	sg-pbrt-q16 + SoAoS
sg-eq	12	snapped grid quantization [36]
sg-eq-align16	16	sg-eq + 16B align

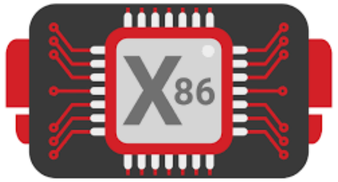
8-ary Layout	Node Size (B)	Description
bvh8	256	unoptimized [103]
bvh8-align16	256	bvh8 + 16B align
bvh8-q8	136	8-bit quantized [10]
bvh8-q8-align16	144	bvh8-q8 + 16B align
bvh8-q8-ci	104	bvh8-q8 + compressed index
bvh8-q8-ci-align16	112	bvh8-q8-ci + 16B align
bvh8-q16	184	16-bit quantized
bvh8-q16-align16	192	bvh8-q16 + 16B align
bvh8-q16-ci	152	bvh8-q16 + compressed index
bvh8-q16-ci-align16	160	bvh8-q16-ci + 16B align



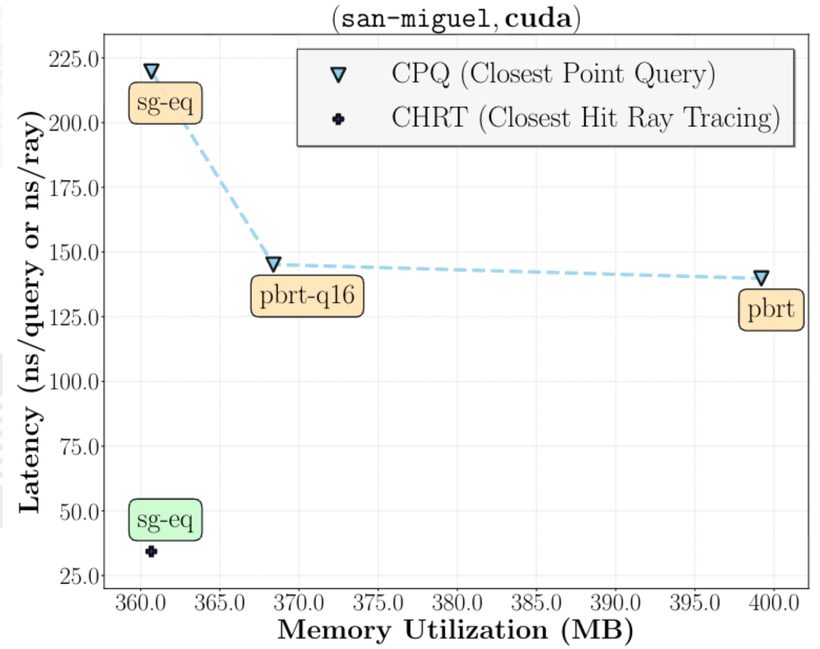




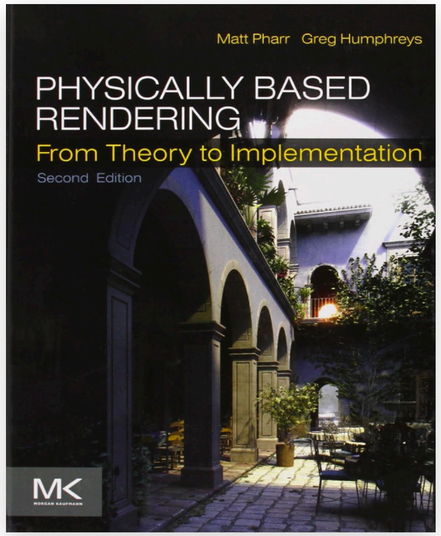


Design Space Exploration: Closest Hit Ray Tracing

The *Pareto-optimal* layout is **algorithm-**, **machine-**, and **data-** specific!



Our work enabled us to rapidly explore *novel* layouts



LinearBVH implicit indexing and bit stealing

+

Quantized bounding volume hierarchies for neighbor search in molecular simulations on graphics processing units

Michael P. Howard[a,*], Antonia Statt[b], Felix Madutsa[b], Thomas M. Truskett[a], Athanassios Z. Panagiotopoulos[b]

Snapped-grid extent quantization

+

Compressed-Leaf Bounding Volume Hierarchies

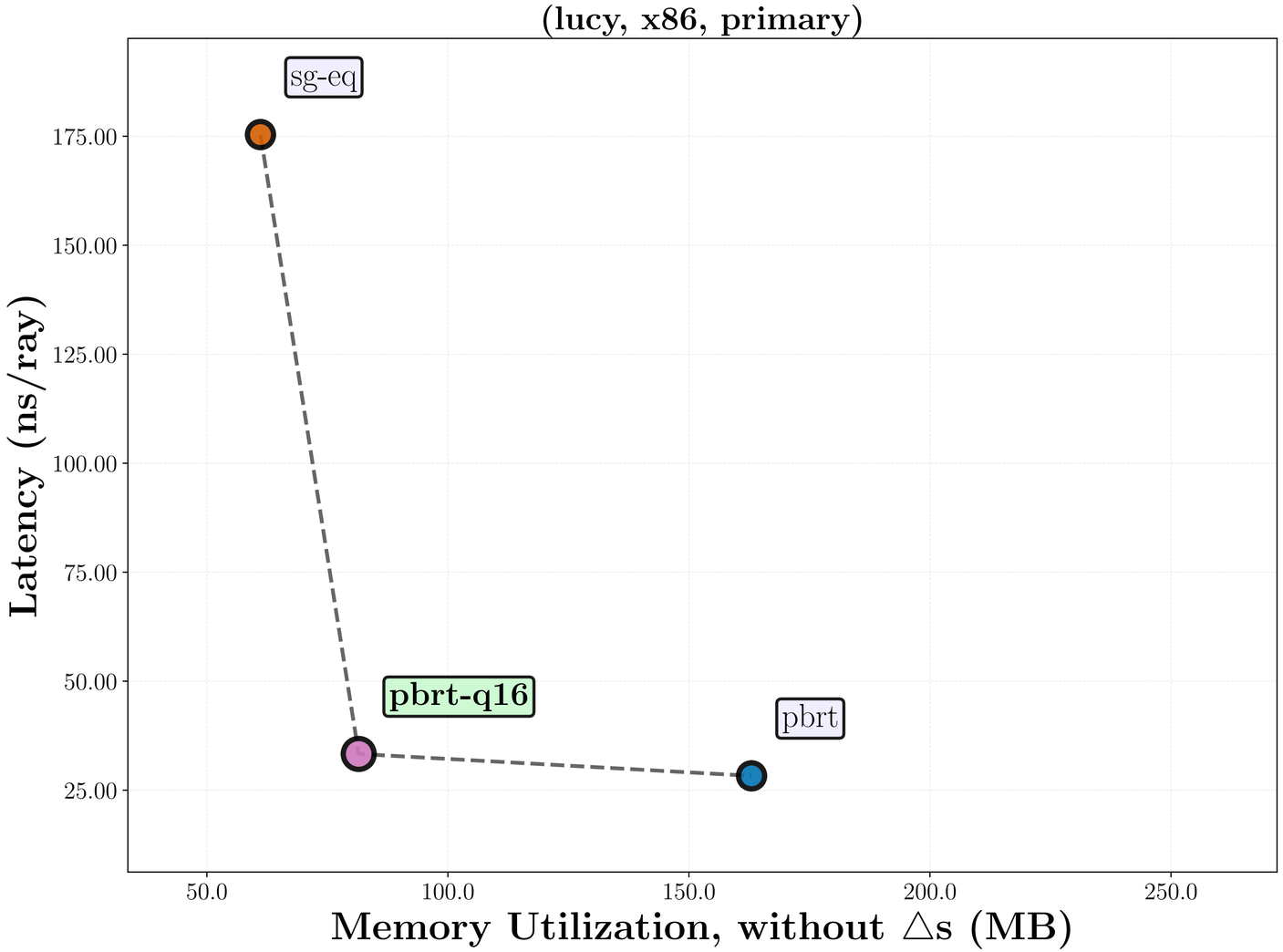
Carsten Benthin, Intel Corporation
Ingo Wald, Intel Corporation
Sven Woop, Intel Corporation
Attila T. Áfra, Intel Corporation

AABB de-quantization scheme

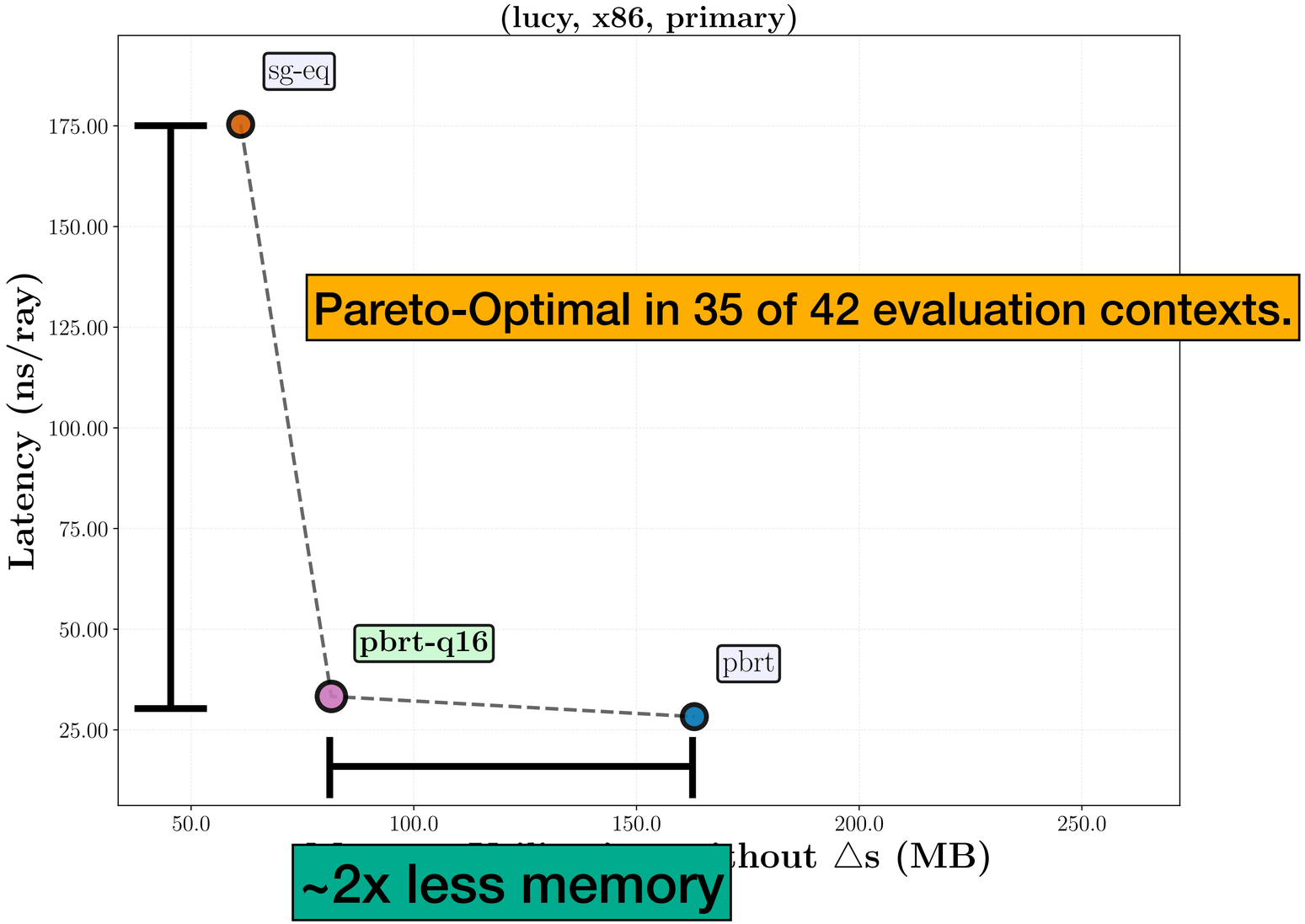
=

PBRT-Q16

Our work enabled us to rapidly explore *novel* layouts



Our work enabled us to rapidly explore *novel* layouts



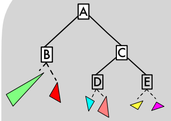
~5x faster

Pareto-Optimal in 35 of 42 evaluation contexts.

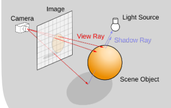
~2x less memory

Conclusion

Domain Expert



Algebraic Data Type



Traversal Algorithms

Physical Specification

( PRIMARY x86)

Physical Specification

( SECONDARY x86)

Physical Specification

( PRIMARY NVIDIA)

Performance Engineer (or AI, I guess)