Hardware-Accelerated Dual-Split Trees

Daqi Lin, Elena Vasiou, Cem Yuksel, Daniel Kopta, Erik Brunvand
University of Utah

HPG 2020

Hi everyone, I'm Daqi Lin from University of Utah. Today I'm going to introduce hardware-accelerated dual-split trees, which are ray tracing acceleration structures that exhibit high performance on ray tracing hardware. This work is a collaboration with Elena Vasiou, Cem Yuksel, Daniel Kopta, and Erik Brunvand.

Background: Acceleration Structures

HPG 2020

- A key factor of high-performance ray tracing

For a background introduction, ray tracing is an essential technique to synthesize photorealistic images. A key factor of high-performance ray tracing is the acceleration structure, which provides fast spatial queries to find hit points along the ray, without checking all primitives in the scene.

Background: Acceleration Structures

HPG 2020

- A key factor of high-performance ray tracing

[Image from PBRT v3]

For a background introduction, ray tracing is an essential technique to synthesize photorealistic images. A key factor of high-performance ray tracing is the acceleration structure, which provides fast spatial queries to find hit points along the ray, without checking all primitives in the scene.
Without any acceleration structure, it is hard to imagine a complex image like this could be rendered in a reasonable amount of time.

Background: Acceleration Structures

- A key factor of high-performance ray tracing
- BVHs are the most popular due to high performance

[Image from PBRT v3]

Currently, bounding volume hierarchies are the most popular acceleration structures, widely used in offline and real-time ray tracing, due to their high construction and traversal performance.

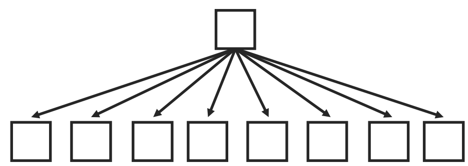
Background: Recent Trend of BVHs

HPG 2020

Noticing that memory cost is the bottleneck of ray tracing performance, recent work have focused on reducing the size of BVHs.

Background: Recent Trend of BVHs

- Wide BVH



Noticing that memory cost is the bottleneck of ray tracing performance, recent work have focused on reducing the size of BVHs.
One approach is to increase the branching factor so that each node has more than 2 child nodes. This is also known as wide BVH. The benefit of using a wide BVH is that it reduces the number of internal levels, resulting in a smaller tree size and fewer traversal steps.

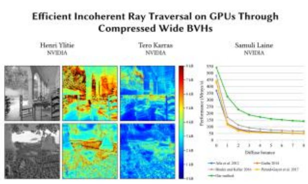
Background: Recent Trend of BVHs

- Wide BVH
- Quantization/Compression of BVH

Noticing that memory cost is the bottleneck of ray tracing performance, recent work have focused on reducing the size of BVHs.
One approach is to increase the branching factor so that each node has more than 2 child nodes. This is also known as wide BVH. The benefit of using a wide BVH is that it reduces the number of internal levels, resulting in a smaller tree size and fewer traversal steps.
Another approach is to quantize the bounding planes and compress other parts of the node structure like child address, which can dramatically reduce the size of BVHs.

Background: Recent Trend of BVHs

- Wide BVH
- Quantization/Compression of BVH



[Ylitie et al. 2017]

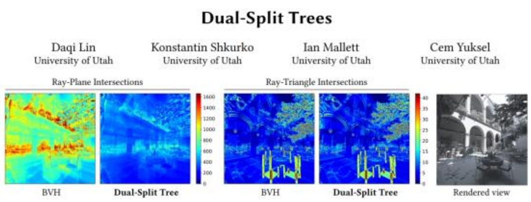
[Benthin et al. 2018]

Noticing that memory cost is the bottleneck of ray tracing performance, recent work have focused on reducing the size of BVHs.
One approach is to increase the branching factor so that each node has more than 2 child nodes. This is also known as wide BVH. The benefit of using a wide BVH is that it reduces the number of internal levels, resulting in a smaller tree size and fewer traversal steps.
Another approach is to quantize the bounding planes and compress other parts of the node structure like child address, which can dramatically reduce the size of BVHs.
These two papers published in HPG 2017/2018 have taken both approaches, to get significant performance improvement compared to prior works.

Background: Dual-Split Trees [Lin et al. 2019]

HPG 2020

- Hybrid acceleration structure between k-d tree and BVH



[Lin et al. 2019]

Recently, we introduced dual-split trees, which is a hybrid acceleration structure between k-d tree and BVH.

Background: Dual-Split Trees [Lin et al. 2019]

HPG 2020

- Uses 31 - 48% less space than BVH2 with identical space partitioning

Recently, we introduced dual-split trees, which is a hybrid acceleration structure between k-d tree and BVH.
The key feature of dual-split trees is that it significantly reduces the space of a binary BVH, while representing the identical space partitioning.

Background: Dual-Split Trees [Lin et al. 2019]

- Uses 31 - 48% less space than BVH2 with identical space partitioning
- Relatively minor performance improvement due to decoding and branching cost

Recently, we introduced dual-split trees, which is a hybrid acceleration structure between k-d tree and BVH.
The key feature of dual-split trees is that it significantly reduces the space of a binary BVH, while representing the identical space partitioning.
However, compared to the space savings, we saw relatively small performance improvement, due to that fact that dual-split trees introduce extra decoding and branching cost. While decoding and branching are complex in software, they could be a simple task on hardware. That’s what motivates us

Our Contribution

- We introduce **hardware-accelerated dual-split trees**

to introduce hardware-accelerated dual-split trees in this work. We show that dual-split tree traversal can be greatly accelerated by a hardware pipeline.

Our Contribution

HPG 2020

- We introduce **hardware-accelerated dual-split trees**
- Up to **31%** less render time & **38%** less energy than HW BVH2

to introduce hardware-accelerated dual-split trees in this work. We show that dual-split tree traversal can be greatly accelerated by a hardware pipeline.

With our implementation on ray tracing hardware, we see up to 31% less render time, and up to 38% less energy consumption in path tracing results, as compared to binary BVHs with identical space partitioning.

Our Contribution

HPG 2020

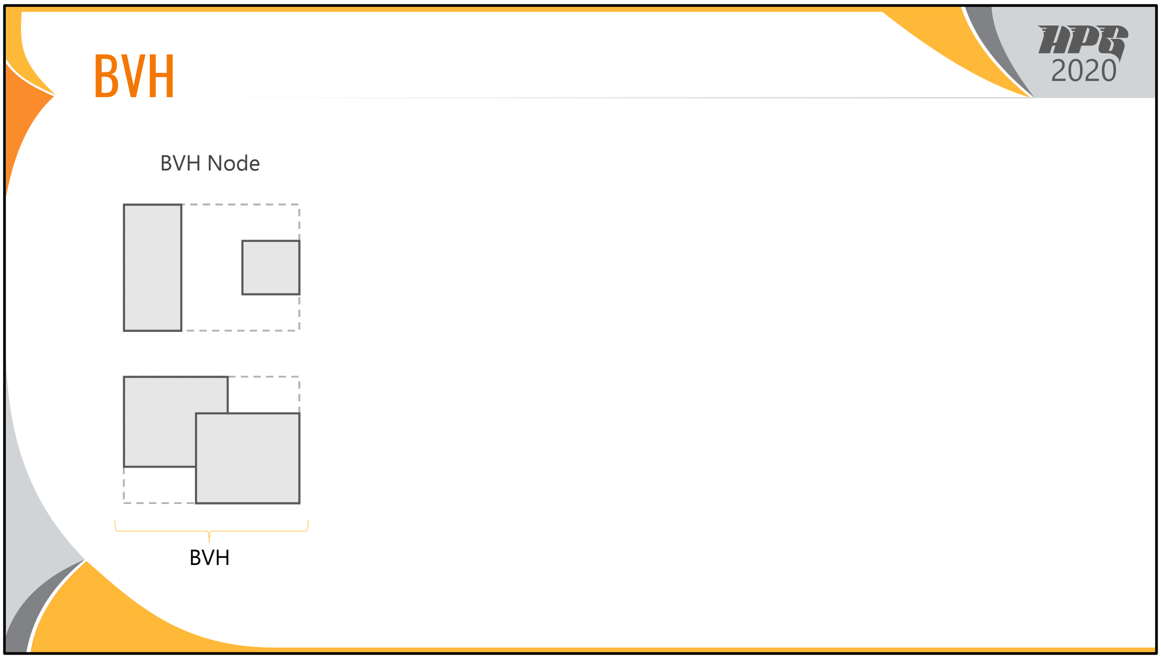
- We introduce **hardware-accelerated dual-split trees**
- Up to **31%** less render time & **38%** less energy than HW BVH2
- Higher performance & less energy than BVH2, BVH4, and BVH8 on ray tracing hardware

to introduce hardware-accelerated dual-split trees in this work. We show that dual-split tree traversal can be greatly accelerated by a hardware pipeline.

With our implementation on ray tracing hardware, we see up to 31% less render time, and up to 38% less energy consumption in path tracing results, as compared to binary BVHs with identical space partitioning.

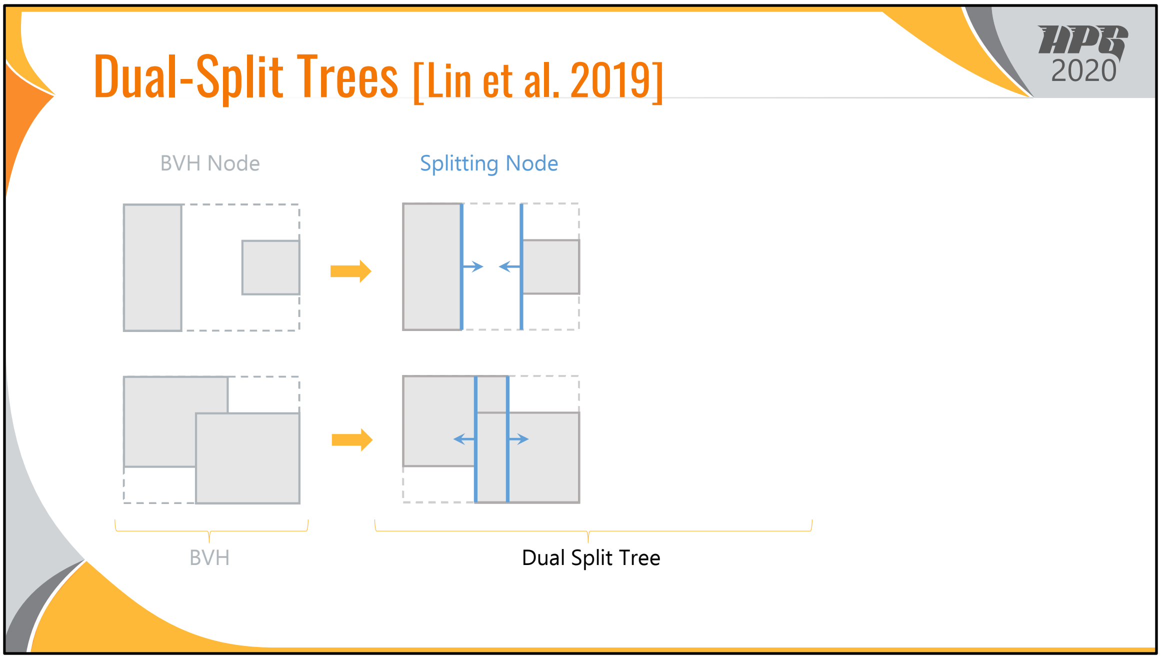
We also observed that hardware-accelerated dual-split trees achieves higher performance and consumes less energy than BVHs with 2, 4, and 8 child nodes.

Before introducing hardware-accelerated dual-split trees, let’s review what are dual-split trees.

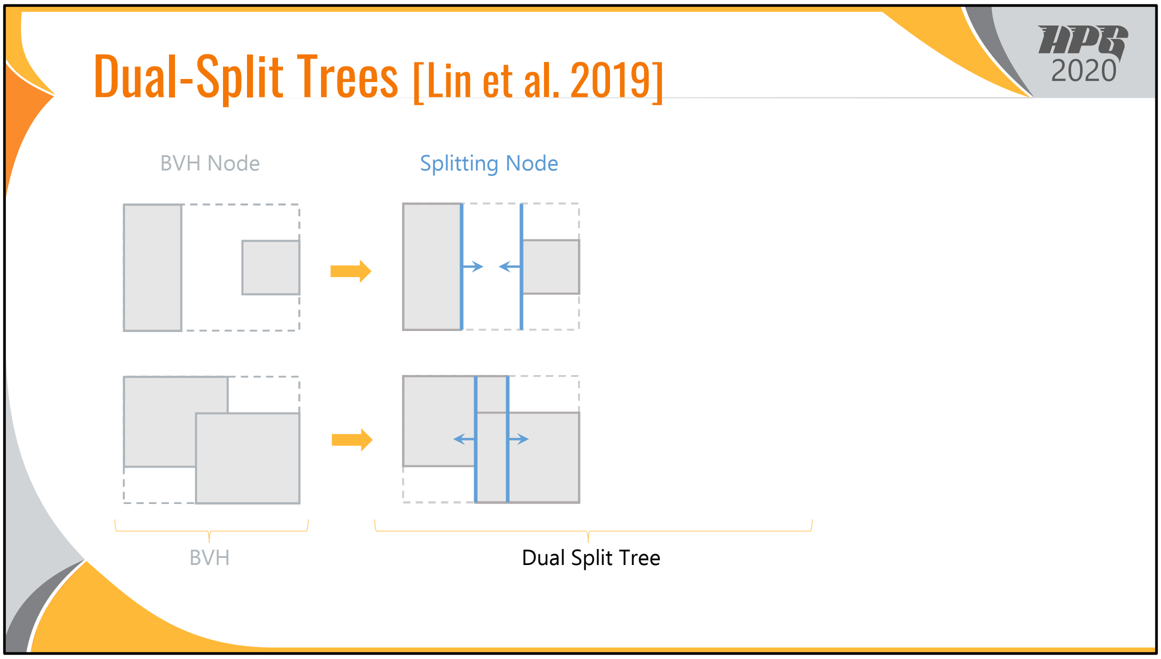


Here are two examples of a BVH node, shown in 2D. Notice how the parent node shown as a box with dashed lines, share planes with the two child nodes. Because the same bounding plane is stored twice in both parent and one child, it is an inherent redundancy of the BVHs.

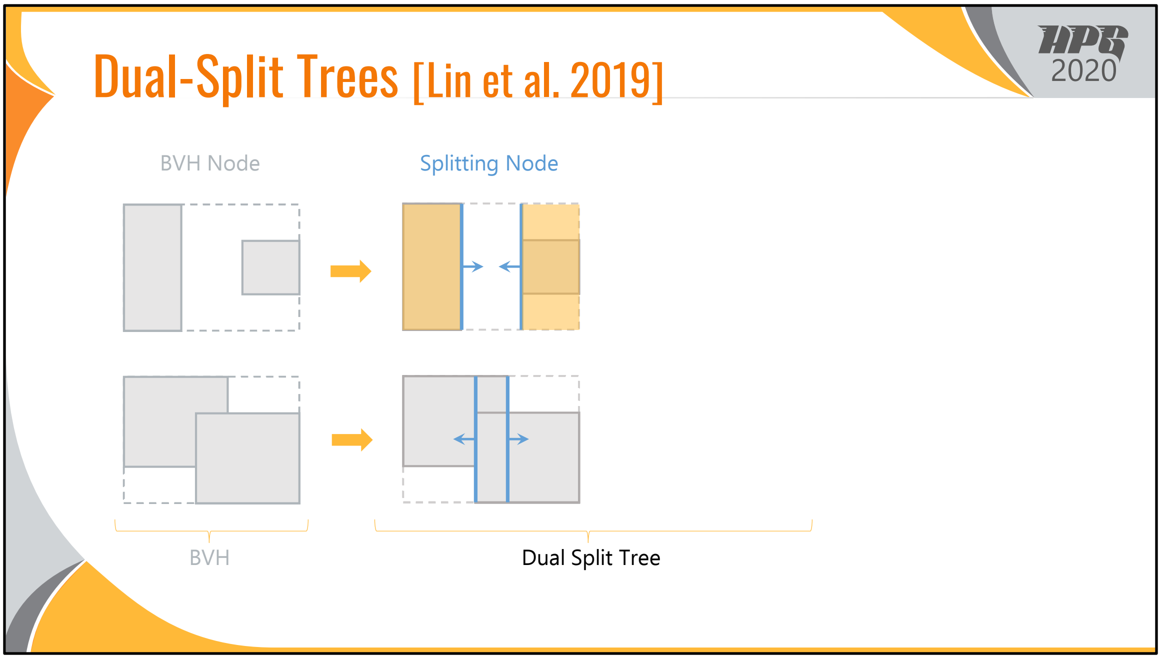
However, we only need to know how the child nodes split the space of the parent node in order to the traverse the tree



So here comes the dual-split tree. In stead of storing two bounding boxes for the two children,

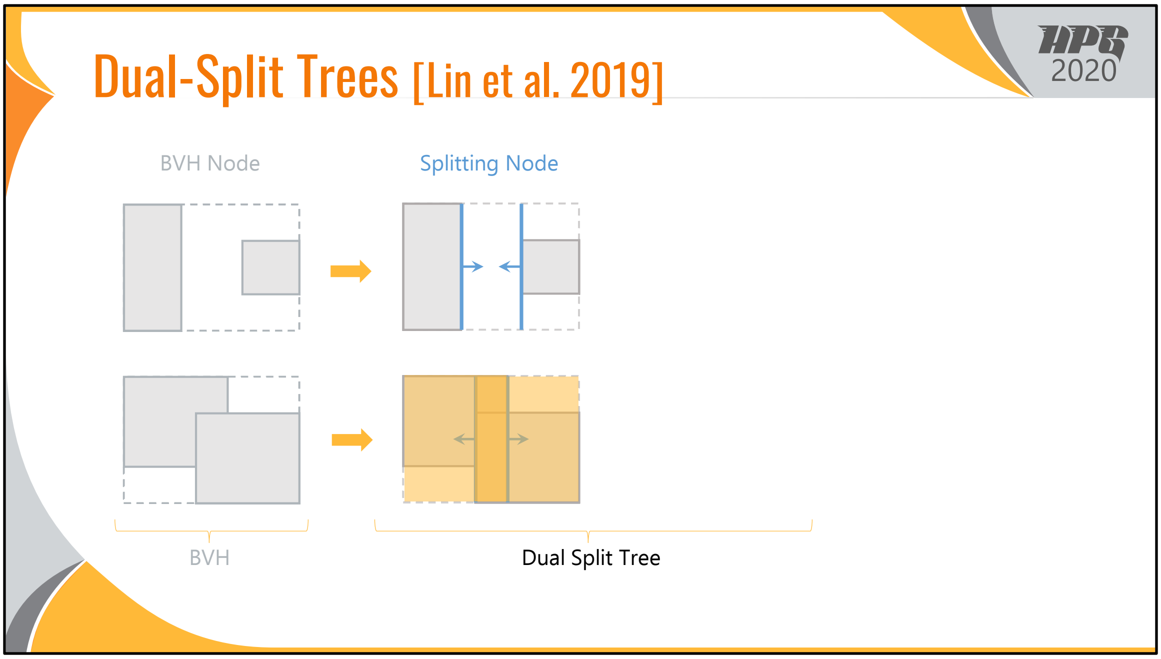


So here comes the dual-split tree. In stead of storing two bounding boxes for the two children, it only stores two axis-aligned splitting planes to separate the child nodes. This is similar to a k-d tree, but with two splitting planes instead of one.



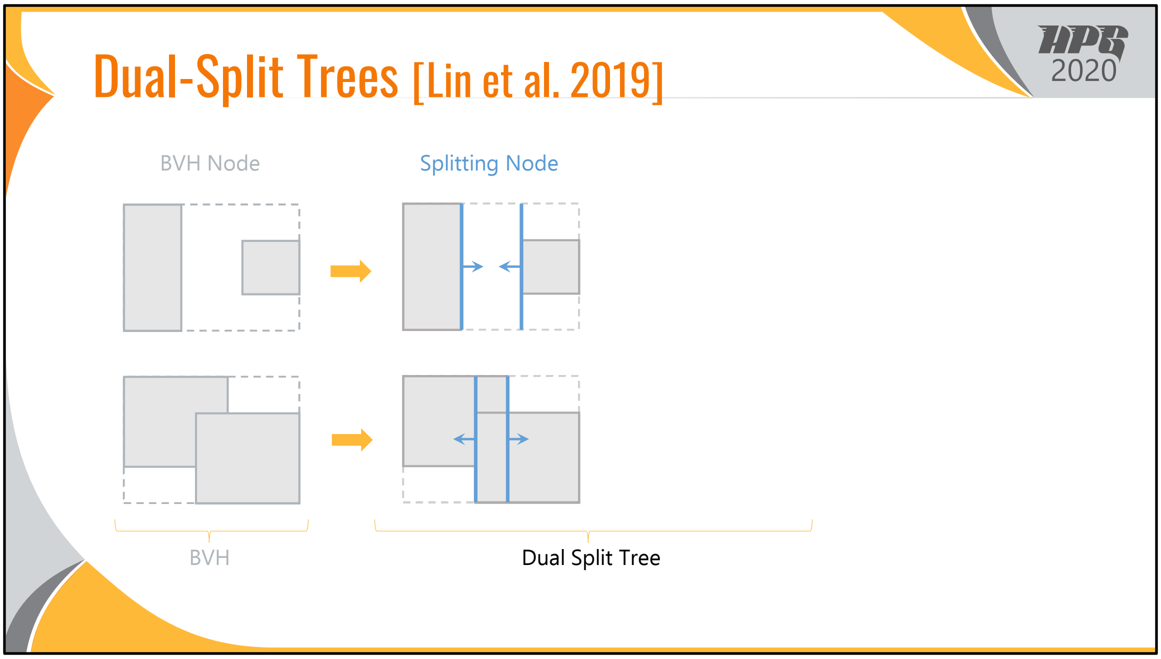
So here comes the dual-split tree. In stead of storing two bounding boxes for the two children, it only stores two axis-aligned splitting planes to separate the child nodes. This is similar to a k-d tree, but with two splitting planes instead of one.

As you can see, the bounding volume of the parent node is separated into two bounding volumes that can leave gap between them,



So here comes the dual-split tree. In stead of storing two bounding boxes for the two children, it only stores two axis-aligned splitting planes to separate the child nodes. This is similar to a k-d tree, but with two splitting planes instead of one.

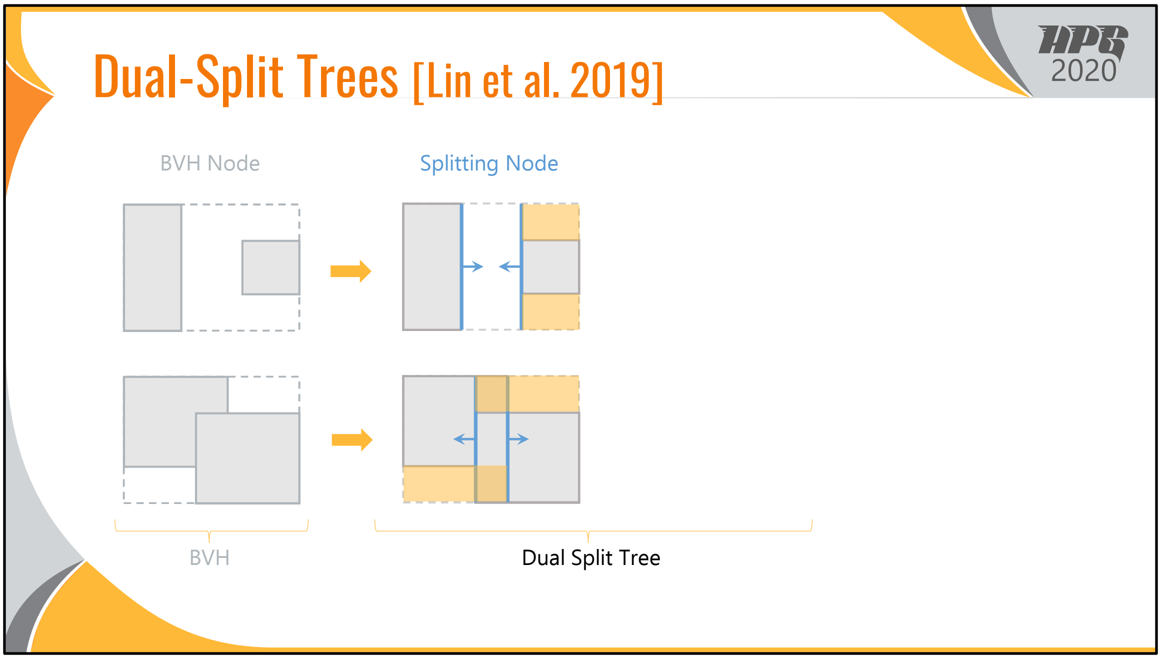
As you can see, the bounding volume of the parent node is separated into two bounding volumes that can leave gap between them, or overlap with each other.



So here comes the dual-split tree. In stead of storing two bounding boxes for the two children, it only stores two axis-aligned splitting planes to separate the child nodes. This is similar to a k-d tree, but with two splitting planes instead of one.

As you can see, the bounding volume of the parent node is separated into two bounding volumes that can leave gap between them, or overlap with each other. The blue arrow on a splitting plane represent the plane normal, which points outwards.

We call a node like this, a splitting node.

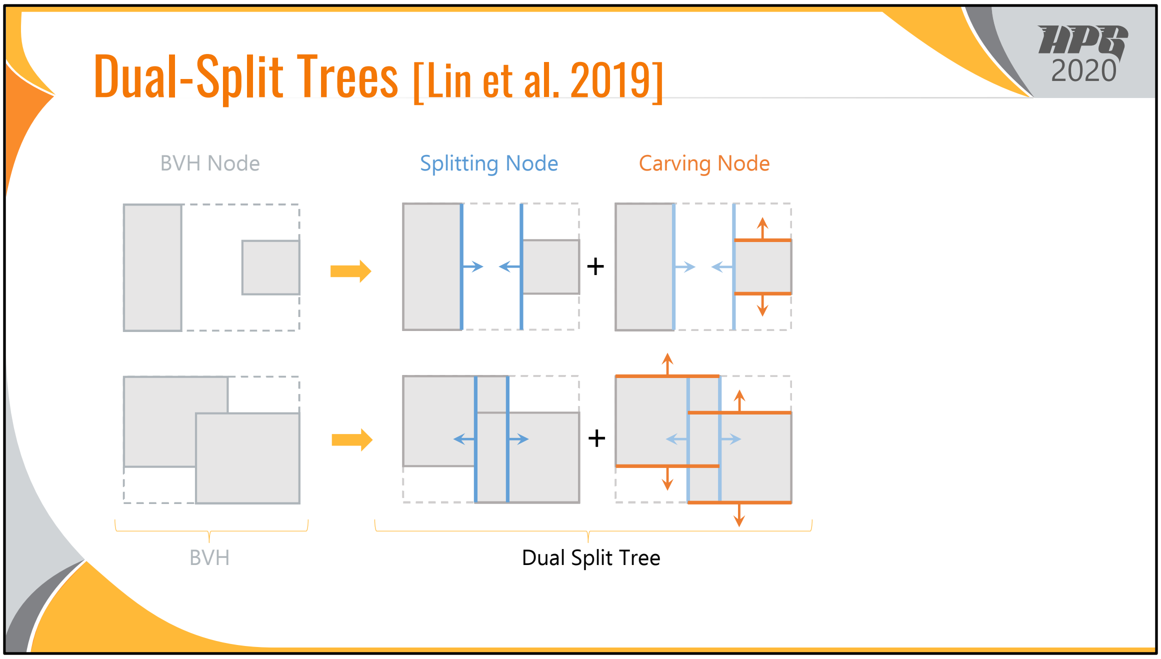


So here comes the dual-split tree. In stead of storing two bounding boxes for the two children, it only stores two axis-aligned splitting planes to separate the child nodes. This is similar to a k-d tree, but with two splitting planes instead of one.

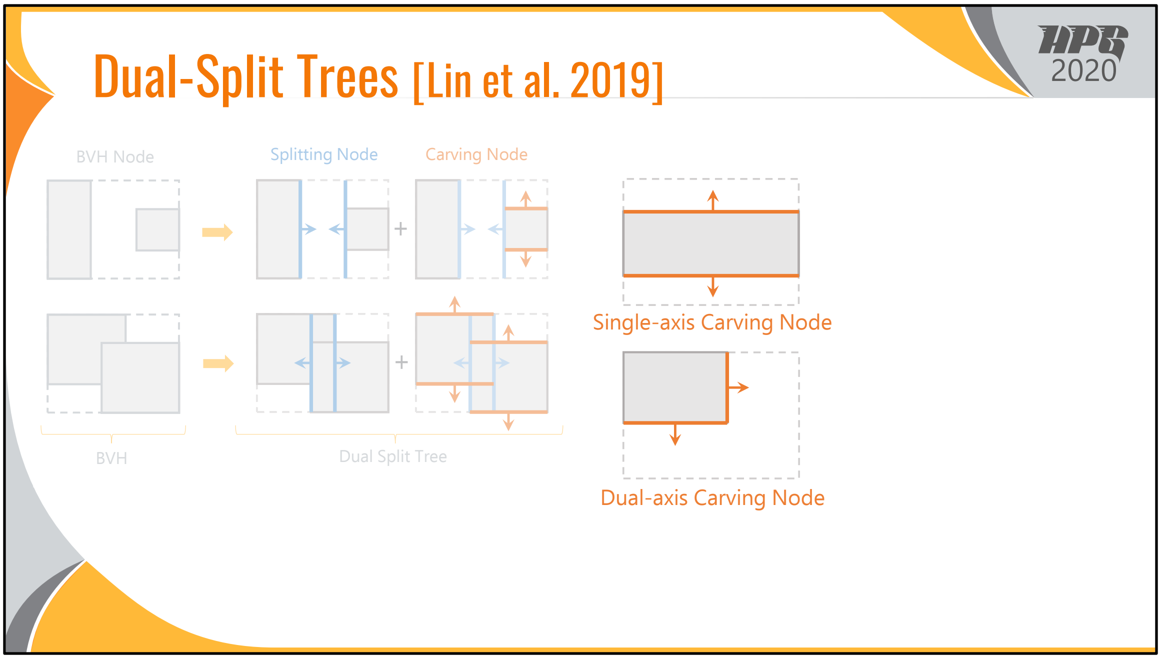
As you can see, the bounding volume of the parent node is separated into two bounding volumes that can leave gap between them, or overlap with each other. The blue arrow on a splitting plane represent the plane normal, which points outwards.

We call a node like this, a splitting node.

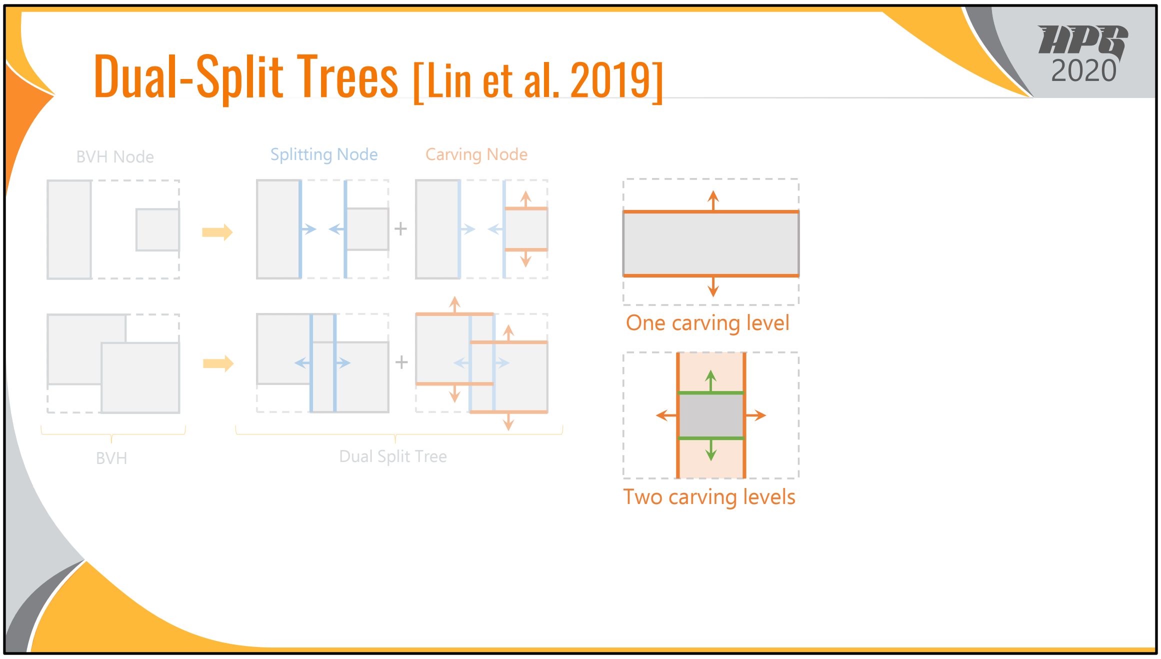
After splitting the child nodes, we noticed that inside the child bounding volume we just created, there are still empty spaces around the original child bounding boxes.



Therefore, we create a carving node that also uses two planes, but to carve out the empty spaces. Notice that in the first case, we completely avoid storing any shared plane between the parent and the child nodes.



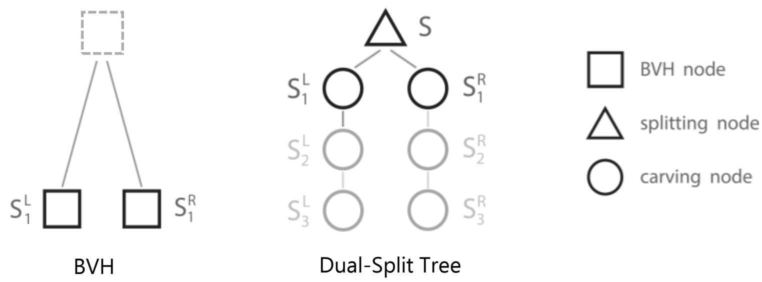
It is also possible to make two carving planes perpendicular to each other. This is known as the dual-axis carving node, as opposed to the single-axis carving node. In 3D, it is very common that after splitting in one axis, there are empty spaces on the two other axes. In this case, one dual-axis carving node can carve out the empty spaces that would have required two single-axis carving nodes, saving a lot of space.



In addition, if one carving level is not enough to carve out all empty spaces, we can create more than one successive carving nodes, making multiple carving levels. Notice that the relationship between the bounding box and carving planes guarantees that there are at most three carving levels.

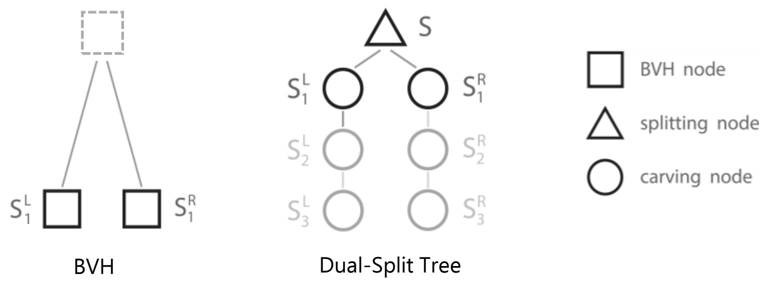
Dual-Split Trees [Lin et al. 2019]

HPG 2020



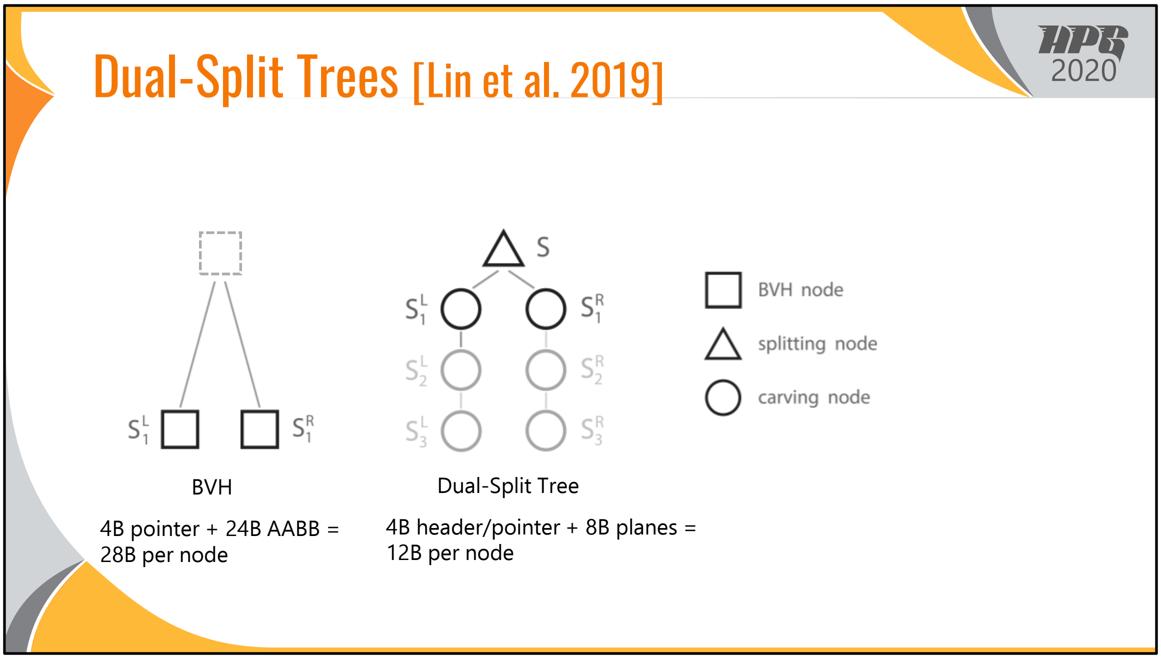
To see how dual-split tree reduces the space of BVHs, here we compare an ordinary binary BVH to a dual-split tree.

Dual-Split Trees [Lin et al. 2019]



4B pointer + 24B AABB =
28B per node

To see how dual-split tree reduces the space of BVHs, here we compare an ordinary binary BVH to a dual-split tree.
An ordinary BVH node uses a 4-byte pointer and a 24-byte AABB, summing up to 28 bytes per node.

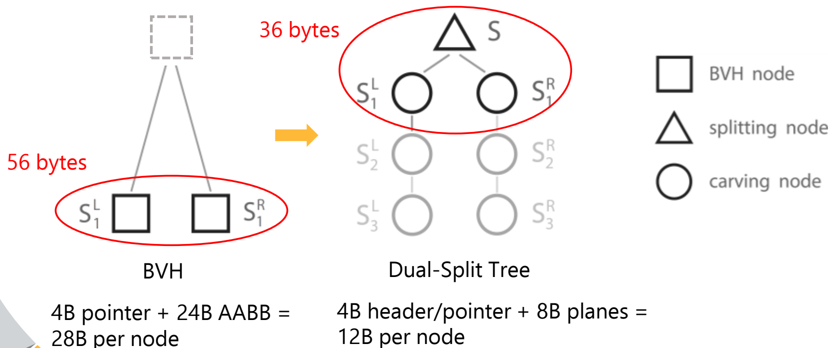


To see how dual-split tree reduces the space of BVHs, here we compare an ordinary binary BVH to a dual-split tree.
An ordinary BVH node uses a 4-byte pointer and a 24-byte AABB, summing up to 28 bytes per node.
In comparison, a dual-split tree splitting or carving node only uses 12 bytes.

Dual-Split Trees [Lin et al. 2019]

HPG 2020

- More nodes, less storage!



To see how dual-split tree reduces the space of BVHs, here we compare an ordinary binary BVH to a dual-split tree.
An ordinary BVH node uses a 4-byte pointer and a 24-byte AABB, summing up to 28 bytes per node.
In comparison, a dual-split tree splitting or carving node only uses 12 bytes.
Since on average only two carving nodes are created after a splitting node, as shown in our prior work. A pair of BVH child nodes can be converted to one splitting node and two carving nodes on average, reducing the size from 56 bytes to 36 bytes.

Dual-Split Trees [Lin et al. 2019]

HPG 2020

- Can be converted from BVH easily

A dual-split tree can be converted from any given binary BVH very easily.

Dual-Split Trees [Lin et al. 2019]

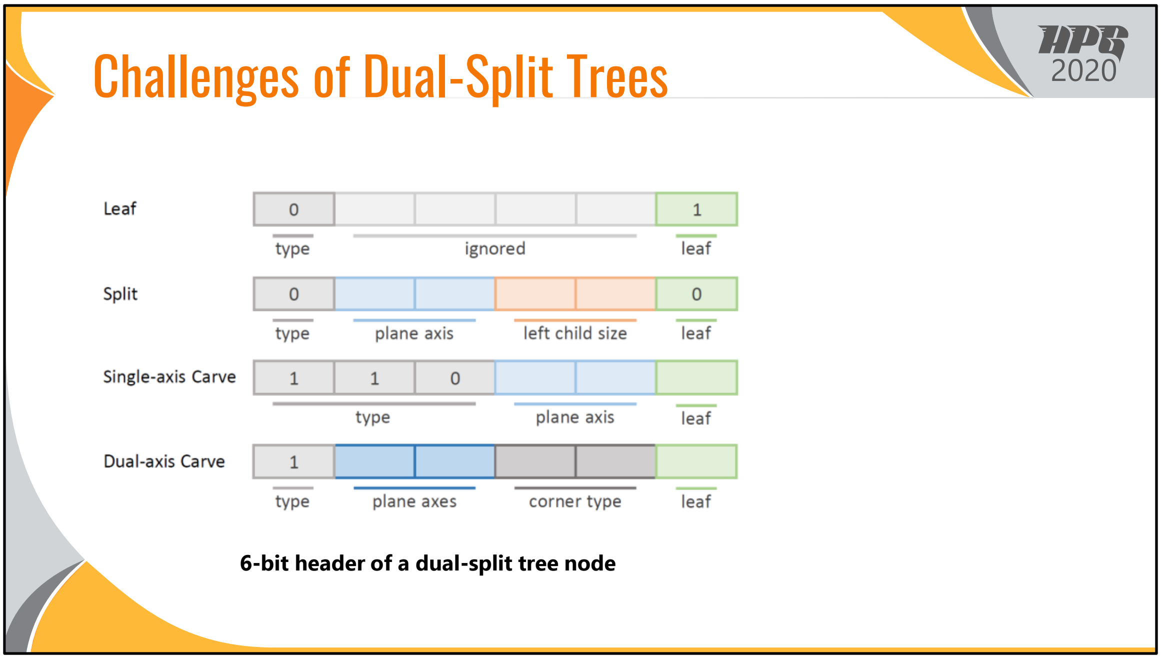
- Can be converted from BVH easily / Can be built from scratch

A dual-split tree can be converted from any given binary BVH very easily. But since it's a different acceleration structure on its own, it can also be built from scratch.

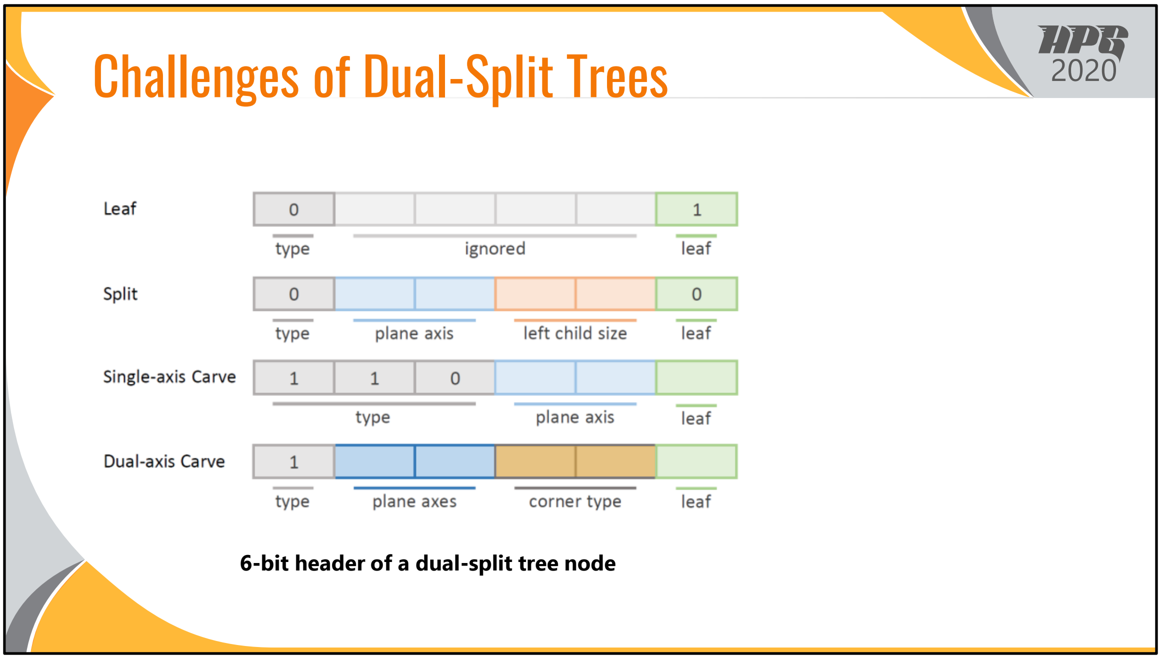
Dual-Split Trees [Lin et al. 2019]

- Can be converted from BVH easily / Can be built from scratch
- Traversal resembles k-d tree (maintains $[t_{min}, t_{max}]$)

A dual-split tree can be converted from any given binary BVH very easily. But since it’s a different acceleration structure on its own, it can also be built from scratch.
In terms of traversal, it resembles a k-d tree, in the sense that it maintains a t_min, t_max interval and trim this interval when intersecting with child nodes.

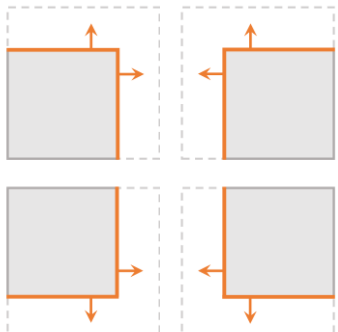


However, a challenge of using dual-split trees is that they have different types of nodes with different functions. Each node type also has its own information stored in different bit positions.



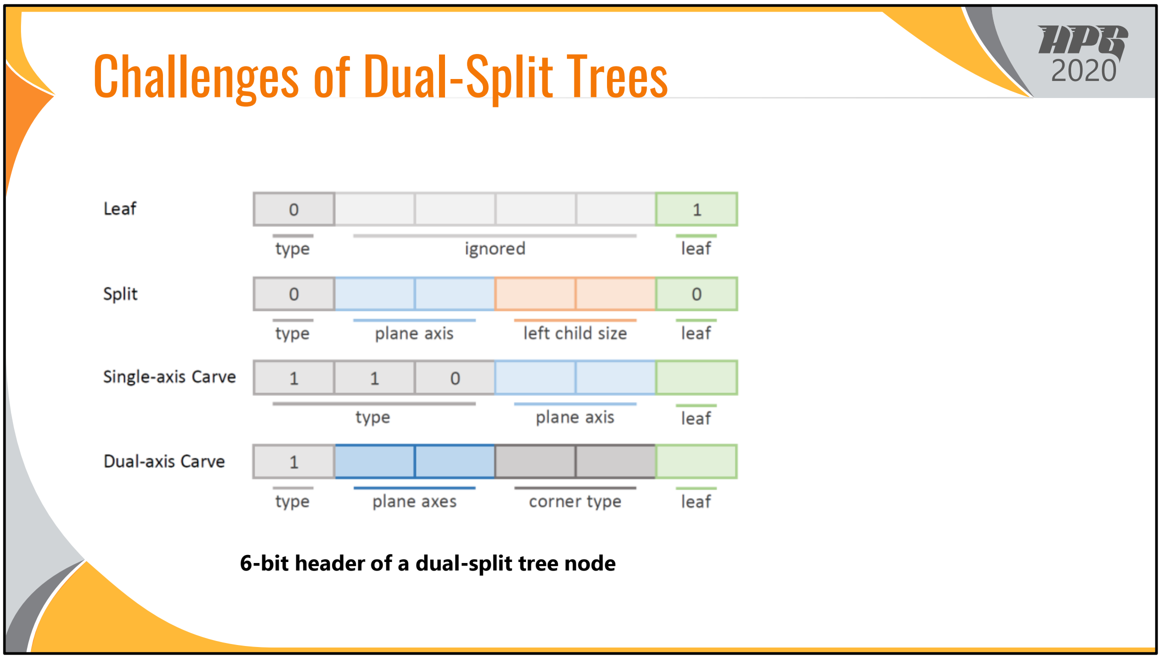
A challenge of using dual-split trees is that they have different types of nodes with different functions. Each node type also has its own information stored in different bit positions. For example, a dual-axis carving node stores two bits to represent four possible corner types,

Challenges of Dual-Split Trees

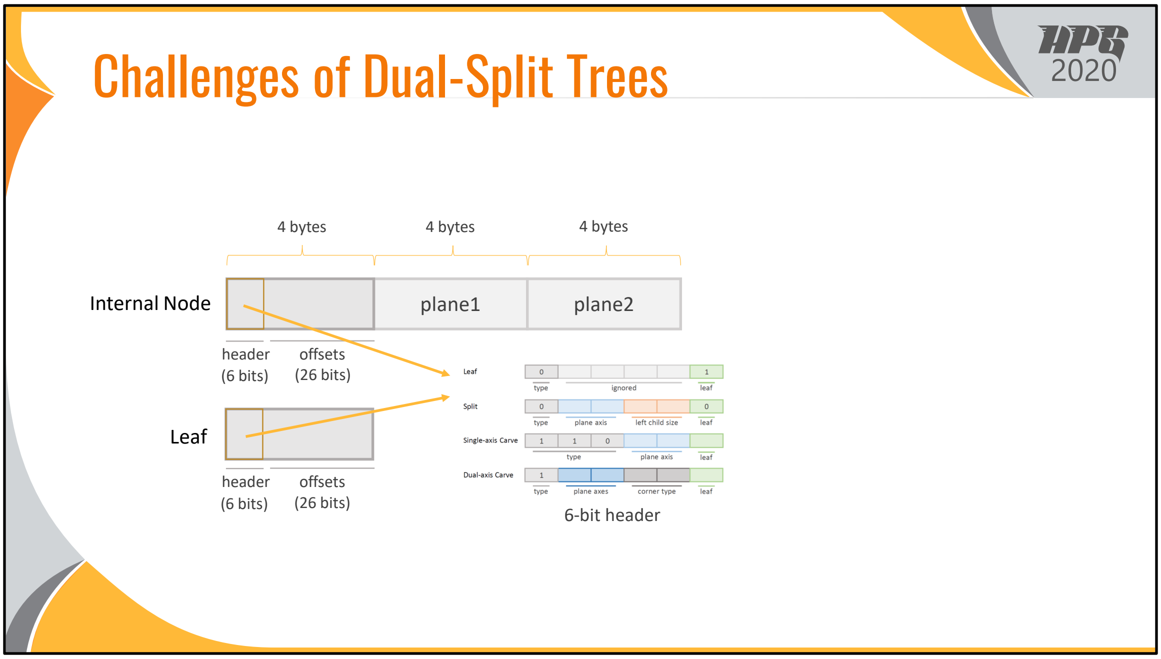


4 "corner types" of dual-axis carving nodes

A challenge of using dual-split trees is that they have different types of nodes with different functions. Each node type also has its own information stored in different bit positions. For example, a dual-axis carving node stores two bits to represent four possible corner types, which corresponds to four different cases of plane normal signs in two axes that look like this.



All this information is packed in a 6-bit header,

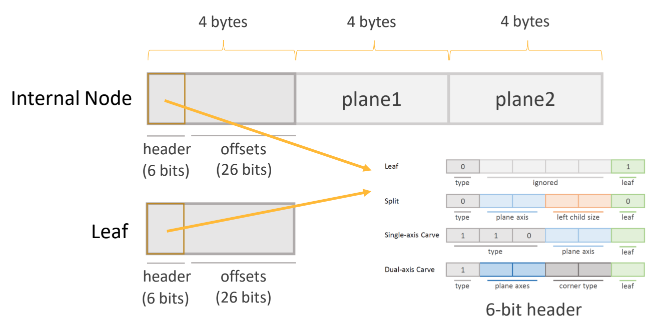


All this information is packed in a 6-bit header, which is a part of the first 4-byte word of the node.

Challenges of Dual-Split Trees

HPG 2020

- Decoding is expensive



All this information is packed in a 6-bit header, which is a part of the first 4-byte word of the node.
Although a compact representation reduces the node size, it requires bit manipulation to decode the information, which can be expensive in software.

Challenges of Dual-Split Trees

HPG 2020

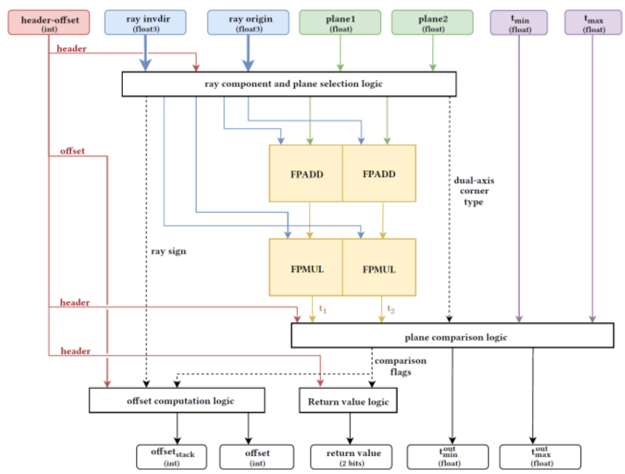
- Branching is expensive

Red box circles all if statements

Furthermore, the different node types creates lots of branching. Here we show the software traversal kernel of dual-split trees, with all if statements circled with red boxes. As can be seen, the branching misprediction penalty could easy undermine the traversal performance.

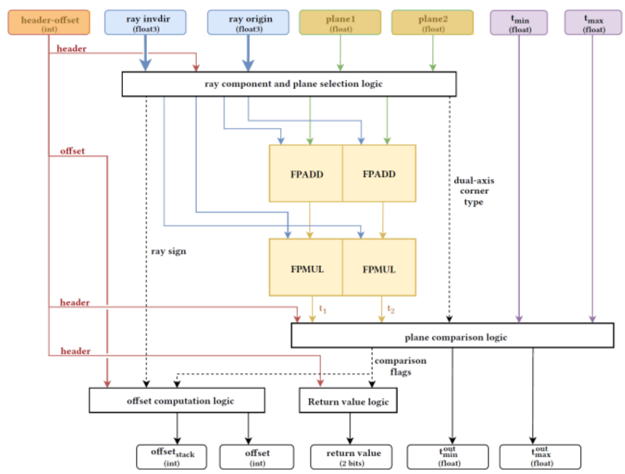
To solve these issues that limit the traversal performance, we introduce hardware acceleration for dual-split trees which largely eliminates the branching and decoding cost, since these tasks are simple in hardware.

Hardware Dual-Split Intersection Pipeline



The hardware acceleration is provided by a hardware dual-split intersection pipeline. Notice that this pipeline is independent of the underlying hardware architecture. One could imagine implementing this as a part of a GPU or a custom ray tracing hardware.

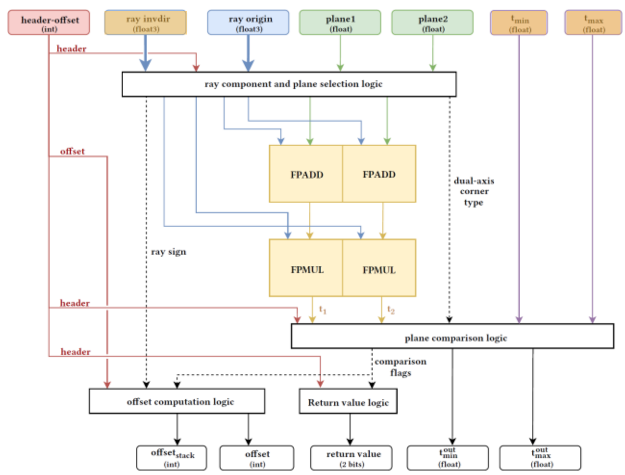
Hardware Dual-Split Intersection Pipeline



Given a dual-split tree node,

Hardware Dual-Split Intersection Pipeline

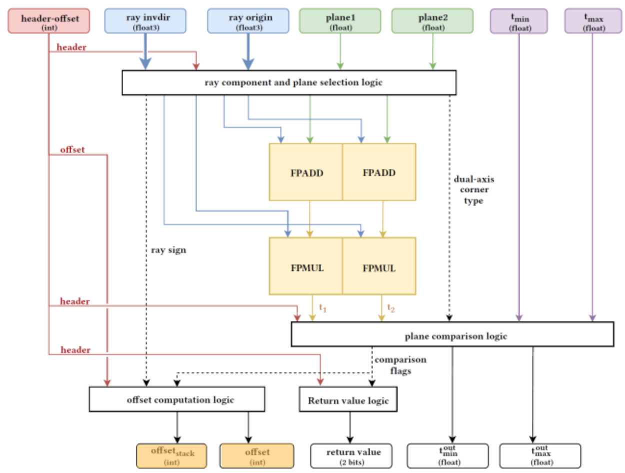
HPG 2020



and the current ray information as inputs

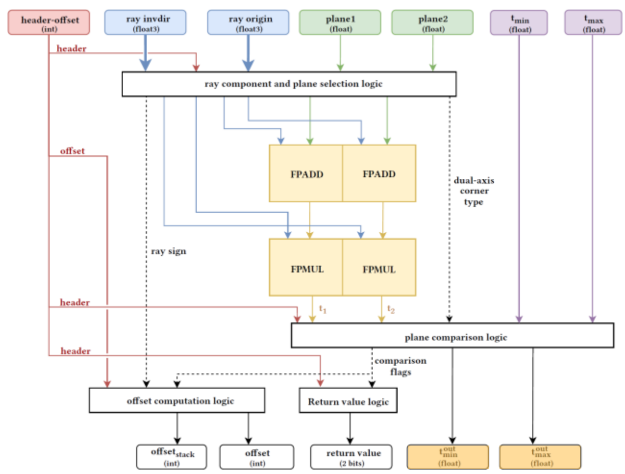
Hardware Dual-Split Intersection Pipeline

HPG 2020



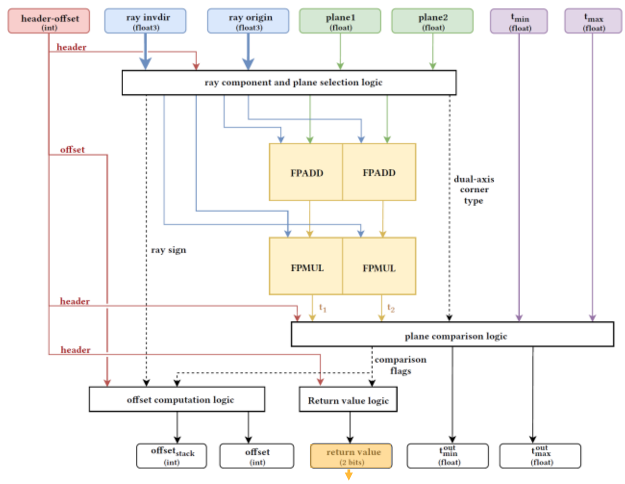
the pipeline produces a pair of child offsets for up to 2 intersected children

Hardware Dual-Split Intersection Pipeline



and the new valid hit distance range of the ray.

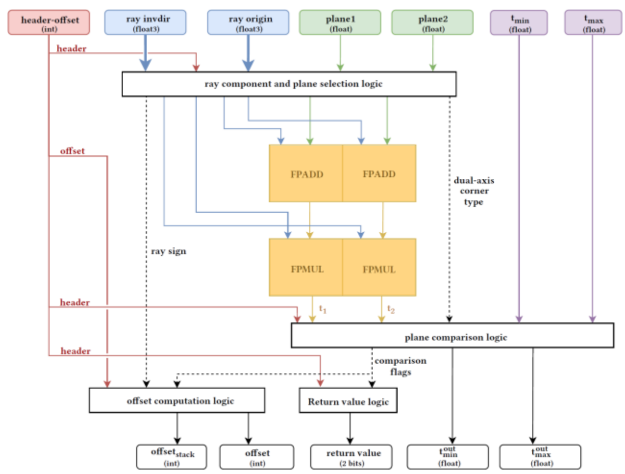
Hardware Dual-Split Intersection Pipeline



0 – no intersection, 1 – single intersection, 2 – double intersections, 3 – leaf node

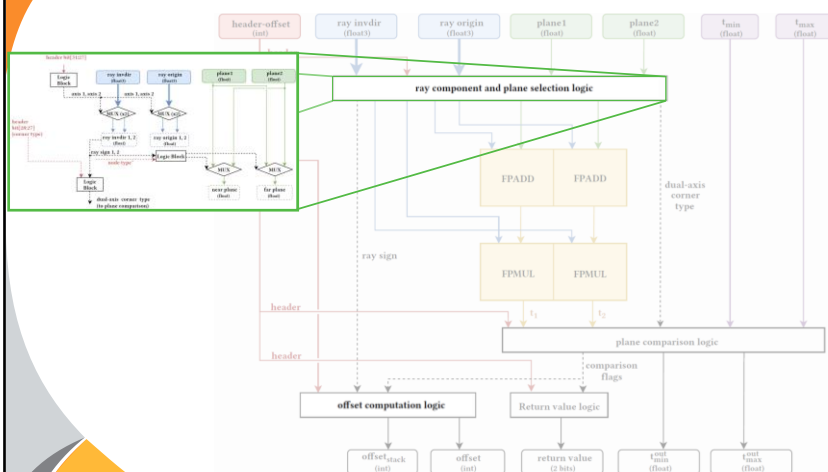
It also produces a 2-bit return value to guide the later code path.

Hardware Dual-Split Intersection Pipeline



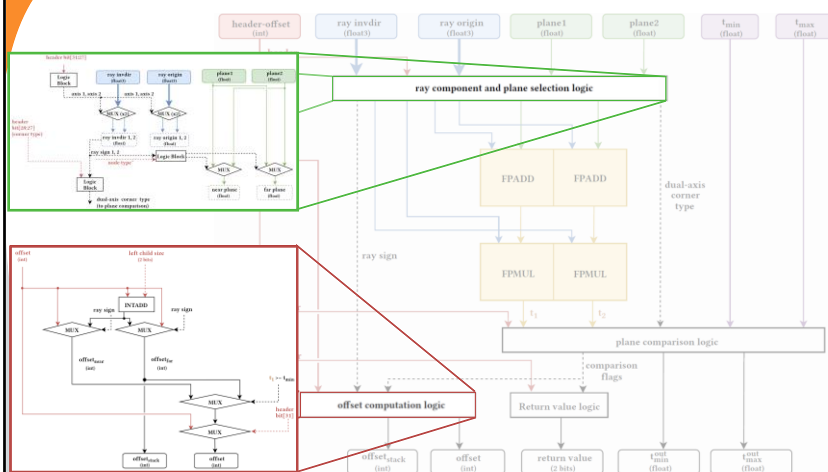
Notice that the dual-split intersection pipeline consists 2 pairs of floating point adders and floating point multipliers, responsible for the ray-plane testing. They occupy the majority of energy and area of the pipeline.

Hardware Dual-Split Intersection Pipeline

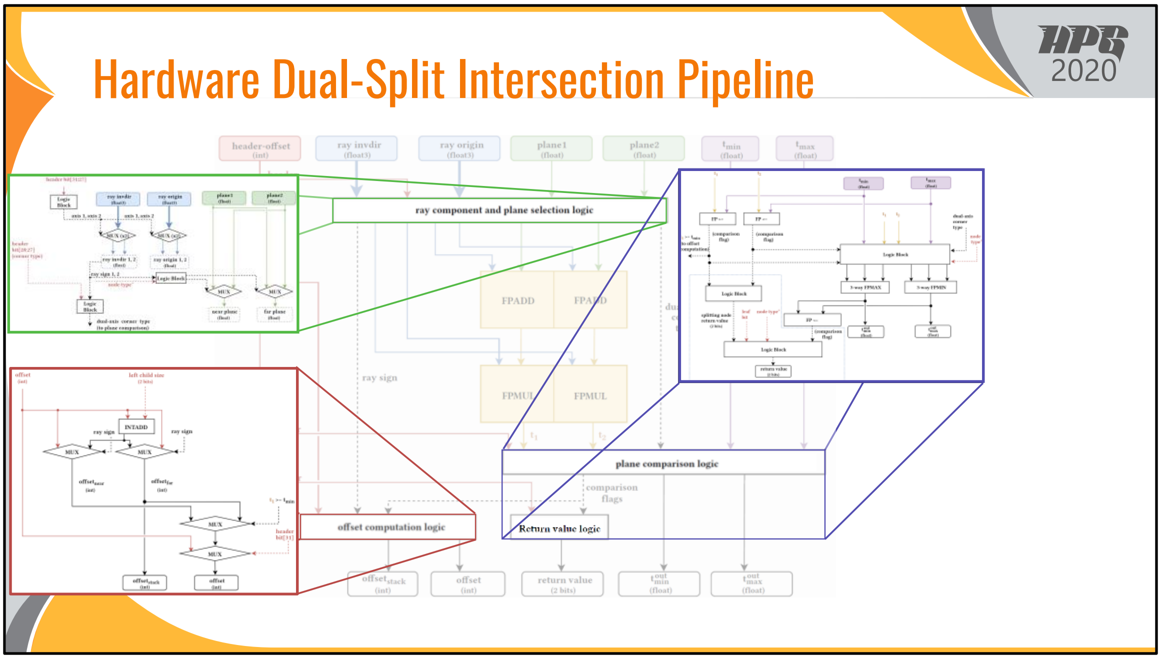


In addition, the pipeline has a ray component and plane selection logic,

Hardware Dual-Split Intersection Pipeline



an offset computation logic,



a plane comparison logic, and a return value logic.

With these logic blocks, decoding can be handled automatically, and branching is completely eliminated, since we can treat all node types uniformly and select the result according to the node type.

Compared to the floating point adders and floating point multiplers, these logic blocks take up a relatively small area in the pipeline since they mainly consist of simple combinational logic and multiplexers.

Hardware Dual-Split Intersection Pipeline

- 3x less FPADD/FPMUL units than HW ray-box pipeline for BVH [Kopta et al. 2013]

		Ray-box	Dual-split
Integer Adder	(INTADD)	–	1 unit
Floating-point Adder	(FPADD)	6 units	2 units
Floating-point Multiplier	(FPMUL)	6 units	2 units
Floating-point Min/Max	(FPMIN/MAX)	12 units	3 units
Floating-point Comparison	(FPCMP)	2 units	4 units
	Total Area	0.1278 mm^2	0.0498 mm^2
	Energy per Operation	0.1377 nJ	0.0610 nJ
	Latency	8 cycles	8 cycles

Here we compare our dual-split intersection pipeline to the ray-box intersection pipeline proposed by Kopta et al. Note that the ray-box pipeline uses 3 times more FPADD and FPMUL.

Hardware Dual-Split Intersection Pipeline

- 3x less FPADD/FPMUL units than HW ray-box pipeline for BVH [Kopta et al. 2013]

		Ray-box	Dual-split	
Integer Adder	(INTADD)	–	1 unit	
Floating-point Adder	(FPADD)	6 units	2 units	
Floating-point Multiplier	(FPMUL)	6 units	2 units	
Floating-point Min/Max	(FPMIN/MAX)	12 units	3 units	
Floating-point Comparison	(FPCMP)	2 units	4 units	
	Total Area	0.1278 mm^2	0.0498 mm^2	2.6x less area
	Energy per Operation	0.1377 nJ	0.0610 nJ	2.3x less energy
	Latency	8 cycles	8 cycles	

Numbers are calculated with units synthesized by 65nm CMOS library, assuming a 1GHz processor.

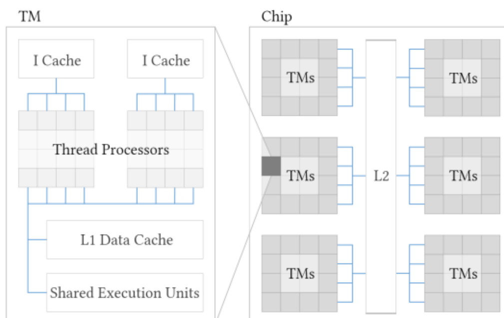
Here we compare our dual-split intersection pipeline to the ray-box intersection pipeline proposed by Kopta et al. Note that the ray-box pipeline uses 3 times more FPADD and FPMUL.
The result of reducing these expensive units is that dual-split pipeline has 2.6x less area and 2.3x less energy compared to ray-box pipeline. Both pipelines have the same latency.

Evaluation

Now, we evaluate the performance of the hardware-accelerated dual-split trees. This include comparison with dual-split trees without acceleration, and BVHs with different branching factors.

Test Platform: TRaX Architecture

- TRaX: a highly parallel and general MIMD architecture for high-performance ray tracing [Spjut et al. 2009]

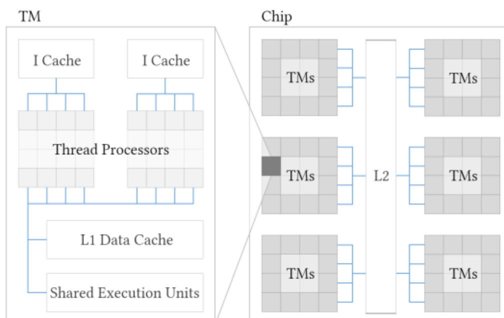


Our test platform is a highly parallel and general MIMD architecture called TRaX, that allows high performance ray tracing using thousands of threads.

We use TRaX because it is a general MIMD architecture, where the ray tracing task is completely software-controlled. This gives us confidence that the results we get can be extrapolated across a wide variety of architectures.

Test Platform: TRaX Architecture

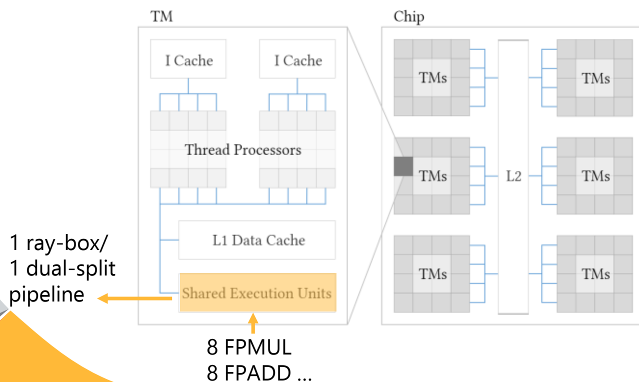
- We use a cycle-accurate simulator [Shkurko et al. 2018] for TRaX



We write our ray tracing program in C++, and use a cycle-accurate simulator to run our program on TRaX.

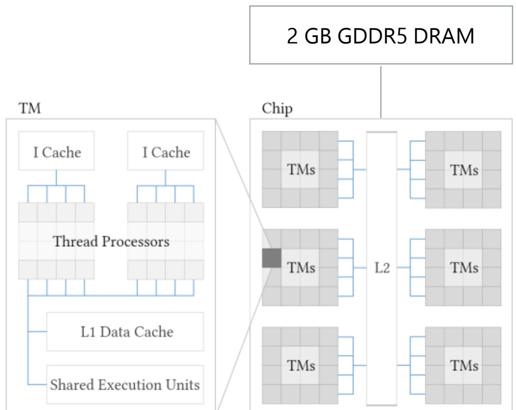
Test Setup

- Shared units on the thread multiprocessor (TM) can be reconfigured to pipelines



We implemented the hardware pipelines for BVHs and dual-split trees using the shared execution units on each thread multiprocessor, or TM. The shared execution units include 8 floating point multipliers and adders.
During runtime, a subset of the shared units can be reconfigured to a ray-box intersection pipeline for a BVH, or a dual-split intersection pipeline for dual-split trees.

Test Setup



The simulated chip is connected to a 2GB GGDR5 DRAM, simulated by a complex memory simulation system.

Acceleration Structures

HPG 2020

- **DST** (Dual Split Tree with hardware acceleration)
- **DST*** (Dual Split Tree with software decoding)
- **BVH2**
- **BVH4**
- **BVH8**

We compare the following acceleration structures in our test. The hardware accelerated dual-split tree. The dual-split tree with only software decoding.
And BVHs with branching factors of 2, 4, and 8. we call them by BVH2, BVH4, and BVH8.
All these acceleration structures are converted from a high quality binary BVH, with their nodes stored in depth first order with sibling nodes stored consecutively in memory. And no plane quantization is used.

Acceleration Structures

HPG 2020

- **DST** (Dual Split Tree with hardware acceleration)
- **DST*** (Dual Split Tree with software decoding)
- **BVH2**
- **BVH4**
- **BVH8**

BVHs with single
HW ray-box intersection pipeline
(test one box at a time)

Notice that the BVHs use a single hardware ray-box intersection pipeline, which means the bounding boxes in a node need to be tested sequentially,

Acceleration Structures

HPG 2020

- **DST** (Dual Split Tree with hardware acceleration)
- **DST*** (Dual Split Tree with software decoding)
- **BVH2**
  3x more units
- **BVH4**
  3x more units
- **BVH8**
  3x more units

BVHs with single
HW ray-box intersection pipeline
(test one box at a time)

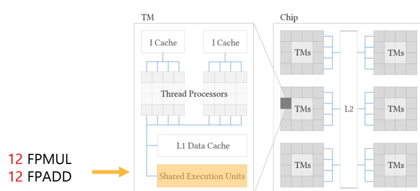
Notice that the BVHs use a single hardware ray-box intersection pipeline, which means the bounding boxes in a node need to be tested sequentially. This ray-box intersection pipeline already uses 3 times more floating point adder and multipler units than the dual-split pipeline.

Acceleration Structures

HPG 2020

- **DST** (Dual Split Tree with hardware acceleration)
- **DST*** (Dual Split Tree with software decoding)
- **BVH2**, **BVH2+**
  6x more units
- **BVH4**
- **BVH8**

BVHs with extended parallel ray-box intersection pipelines
(test all boxes at the same time)



Additionally, we compare with BVHs with extended parallel ray-box intersection pipelines, allowing testing all boxes in the node at the same time.
The binary BVH with the enhanced pipeline is denoted as BVH2+. Note that it uses 6 times more units than dual-split pipeline, and that requires adding more shared execution units to the TM.

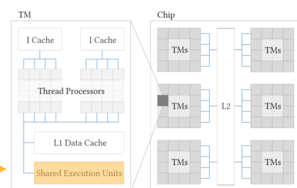
Acceleration Structures

HPG 2020

- **DST** (Dual Split Tree with hardware acceleration)
- **DST*** (Dual Split Tree with software decoding)
- **BVH2**, **BVH2+**
  6x more units
- **BVH4**, **BVH4+**
  12x more units
- **BVH8**

BVHs with extended
parallel ray-box intersection pipelines
(test all boxes at the same time)

24 FPMUL
24 FPADD



Similarly, we have BVH4+ that uses a pipeline with 12x more units than dual-split pipeline, which requires to add even more functional units to the TM.

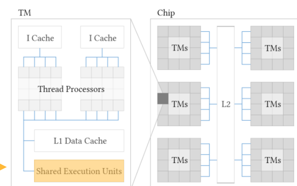
Acceleration Structures

HPG 2020

- **DST** (Dual Split Tree with hardware acceleration)
- **DST*** (Dual Split Tree with software decoding)
- **BVH2**, **BVH2+**
  6x more units
- **BVH4**, **BVH4+**
  12x more units
- **BVH8**, **BVH8+**
  24x more units

BVHs with extended
parallel ray-box intersection pipelines
(test all boxes at the same time)

48 FPMUL
48 FPADD



And BVH8+, with 24 times more units to test 8 bounding boxes simultaneously

Test Cases

HPG 2020

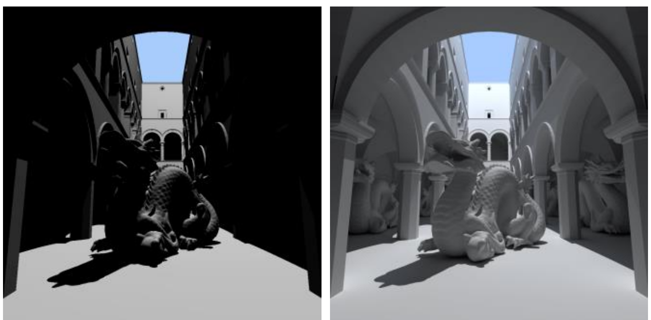
- Ray casting and shadow rays

To evaluate the performance of the acceleration structures, we test them in two different scenarios. The first test renders an image with only direct lighting. So there are only primary rays and shadow rays.
The second test is path tracing with 5 diffuse bounces, which generates highly incoherent rays.

Test Cases

HPG 2020

- Ray casting and shadow rays
- Path tracing with 5 diffuse bounces



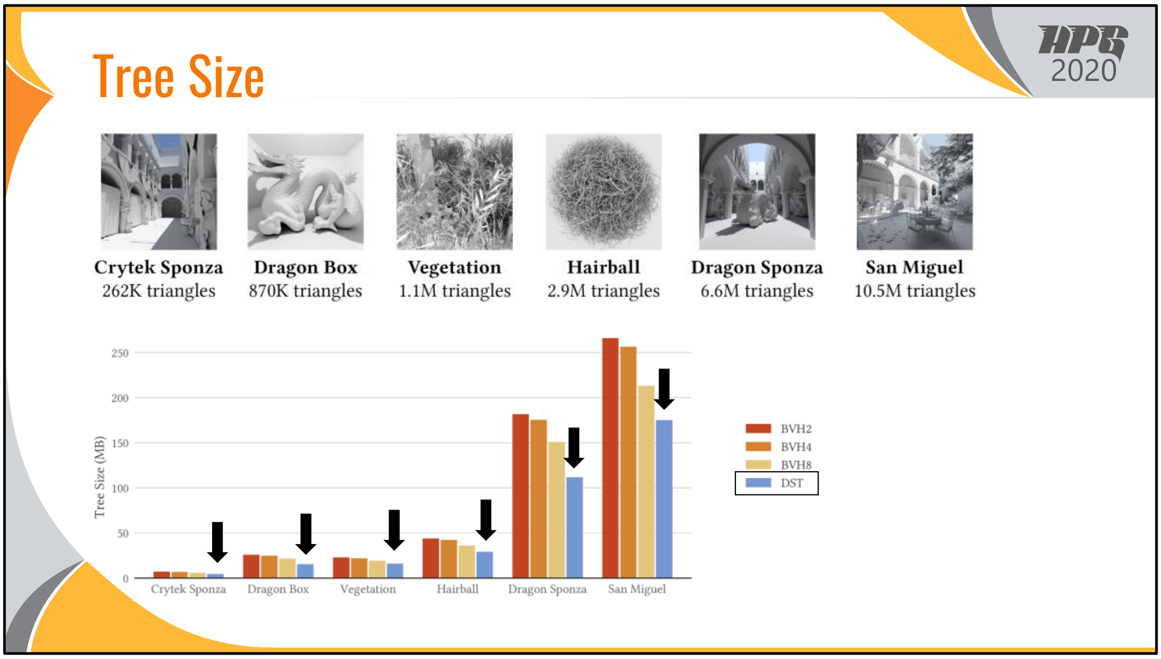
To evaluate the performance of the acceleration structures, we test them in two different scenarios. The first test renders an image with only direct lighting. So there are only primary rays and shadow rays.
The second test is path tracing with 5 diffuse bounces, which generates highly incoherent rays to stress the acceleration structures.

Test Scenes

HPG 2020

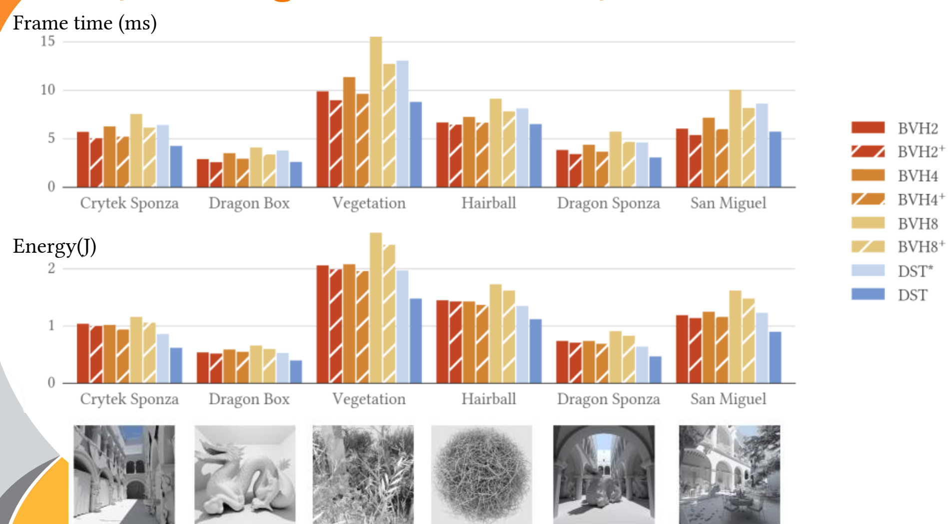
Crytek Sponza	Dragon Box	Vegetation	Hairball	Dragon Sponza	San Miguel
262K triangles	870K triangles	1.1M triangles	2.9M triangles	6.6M triangles	10.5M triangles

We test with these 6 scenes with different levels of geometric complexities.

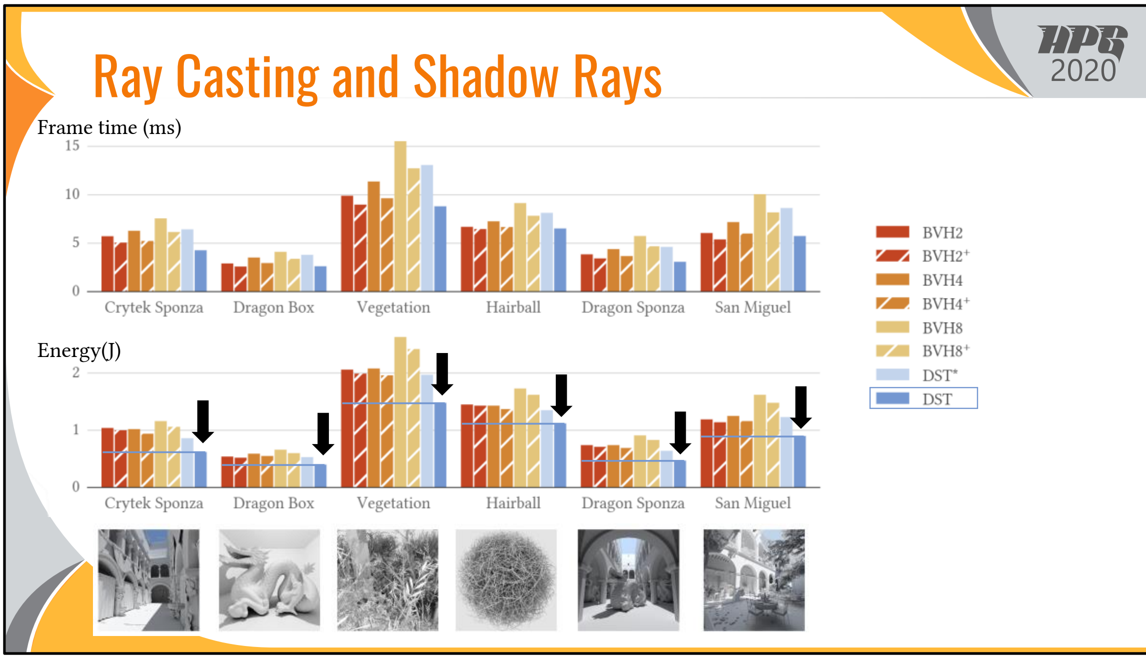


We test with these 6 scenes with different levels of geometric complexities.
Notice that dual-split trees reduce the tree sizes compared to all BVH variants.

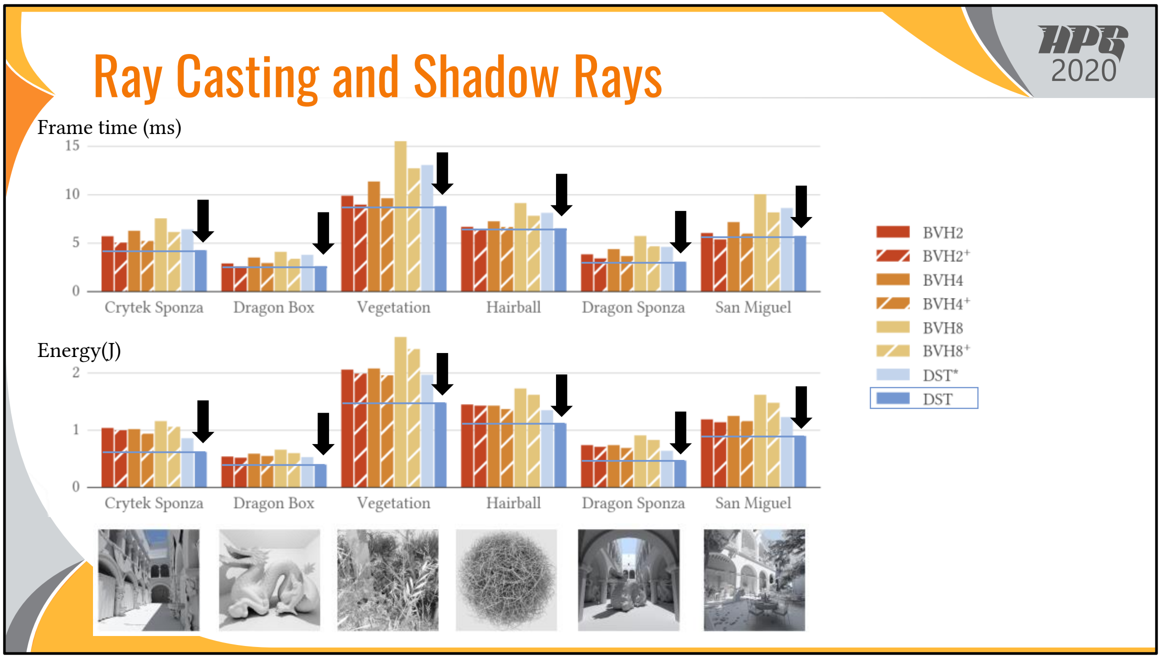
Ray Casting and Shadow Rays



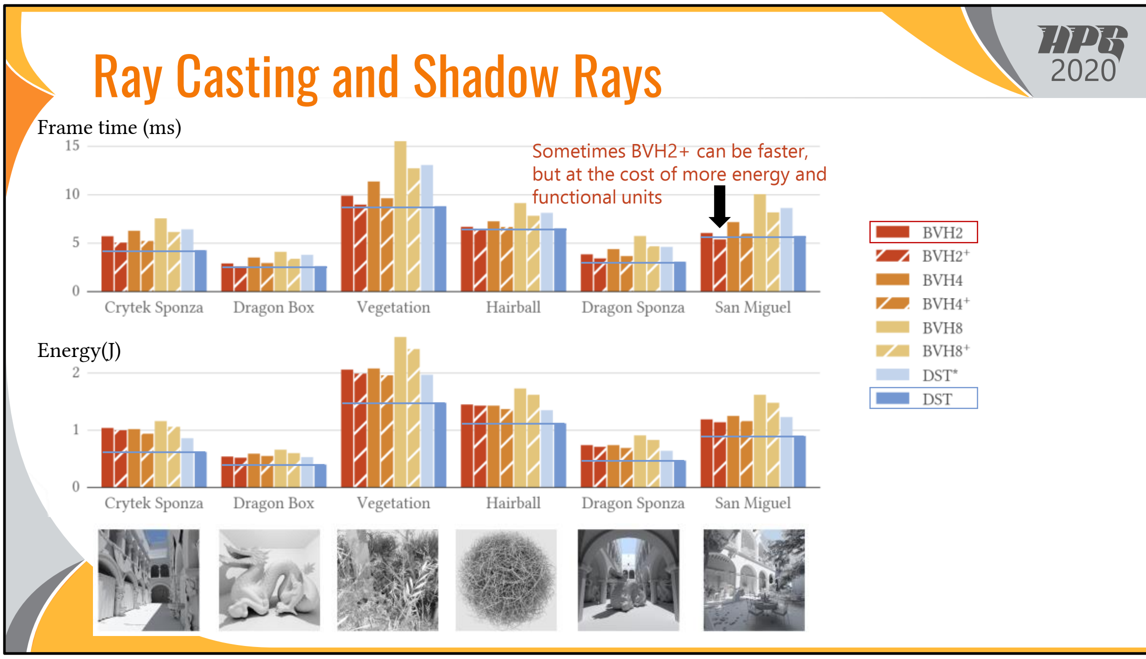
Here we present the performance results. The upper row is the frame time, the lower row is the total energy consumption. Note that these numbers are the lower the better.



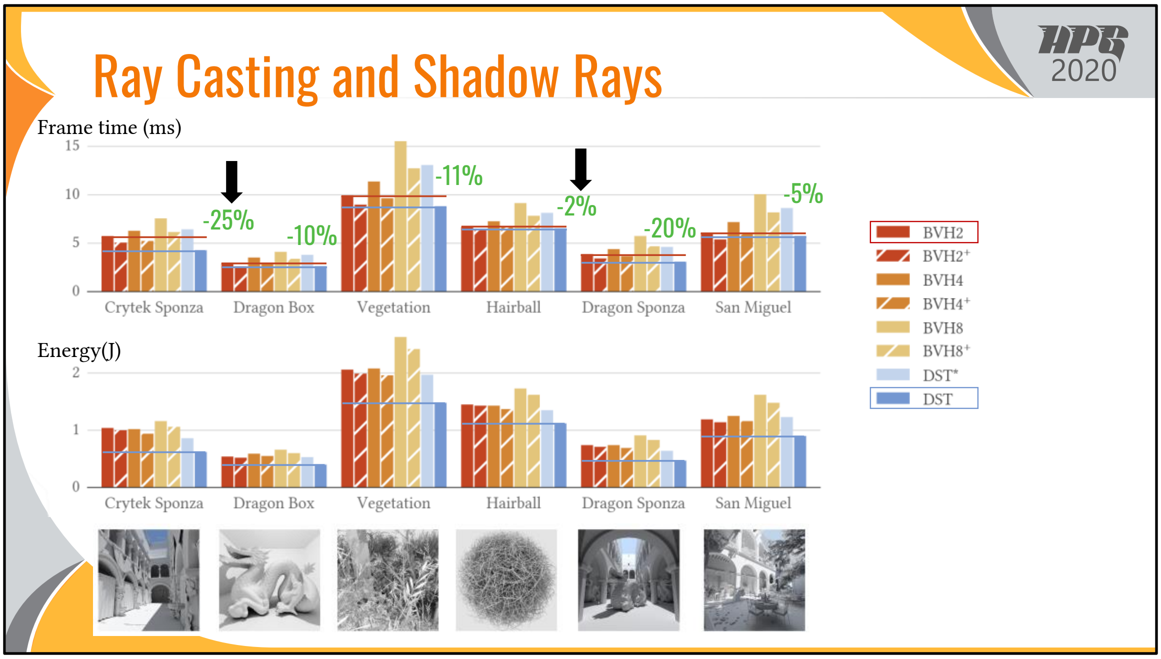
As can be seen, for ray casting test, hardware dual-split trees consistently consumes less energy than all other acceleration structures,



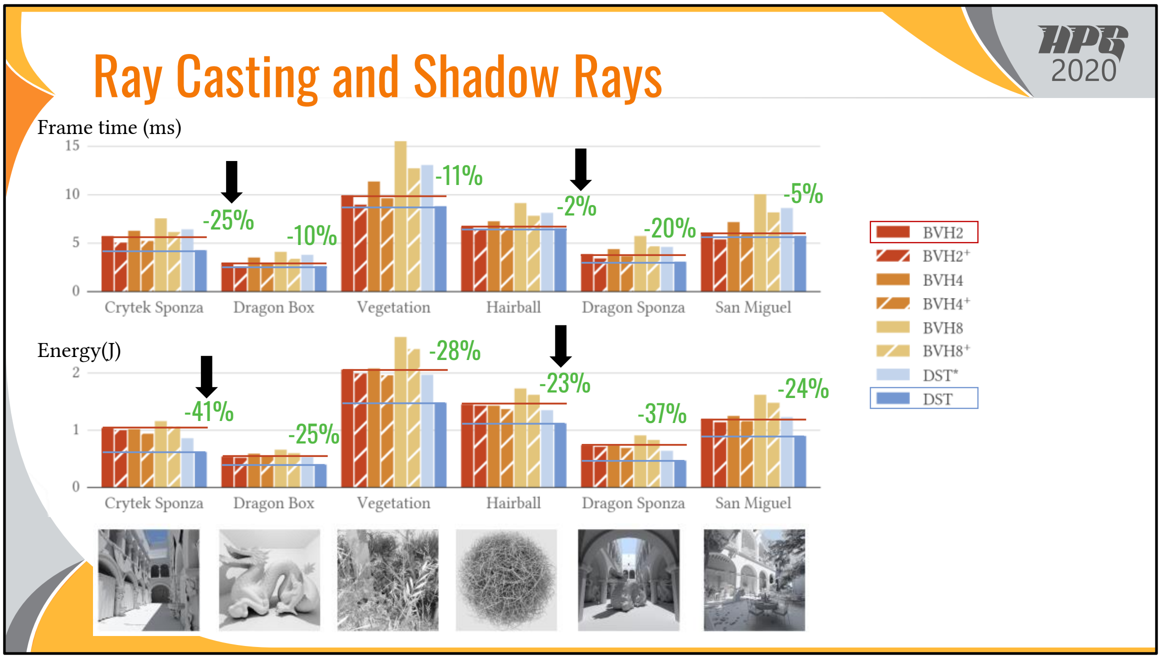
As can be seen, for ray casting test, hardware dual-split trees consistently consumes less energy than all other acceleration structures. It also performs faster than all other acceleration structures,



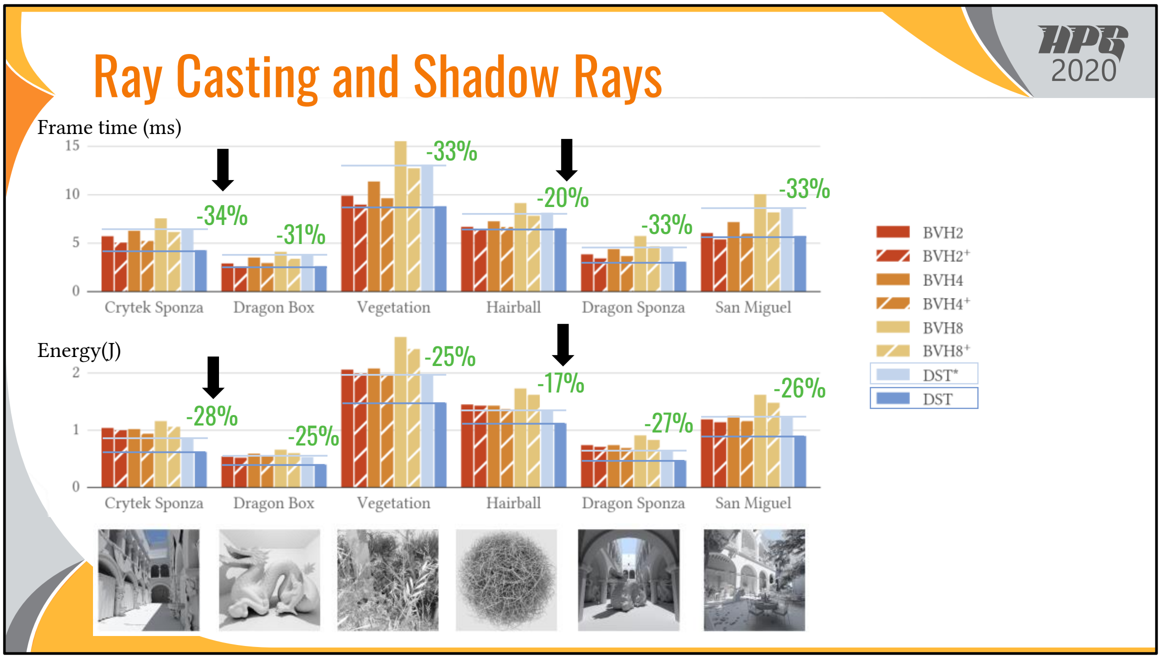
except BVH2+, which sometimes can be faster, but at the cost of using more energy and functional units.



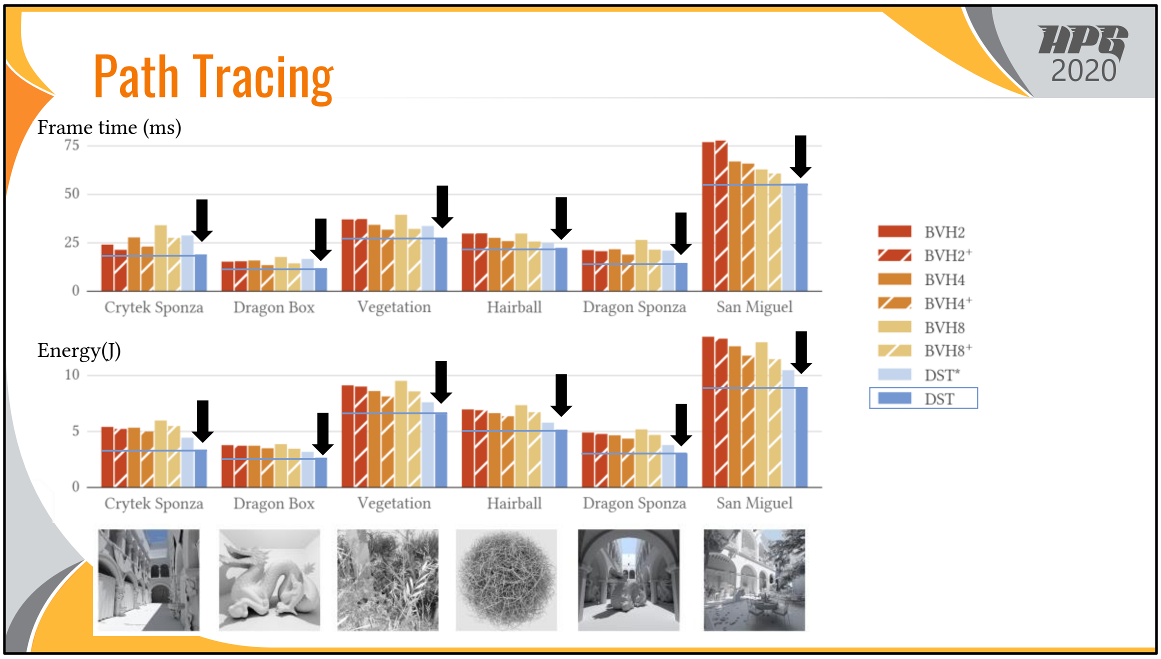
Compared to the baseline BVH2, hardware dual-split trees reduces the render time by 2% - 25%,



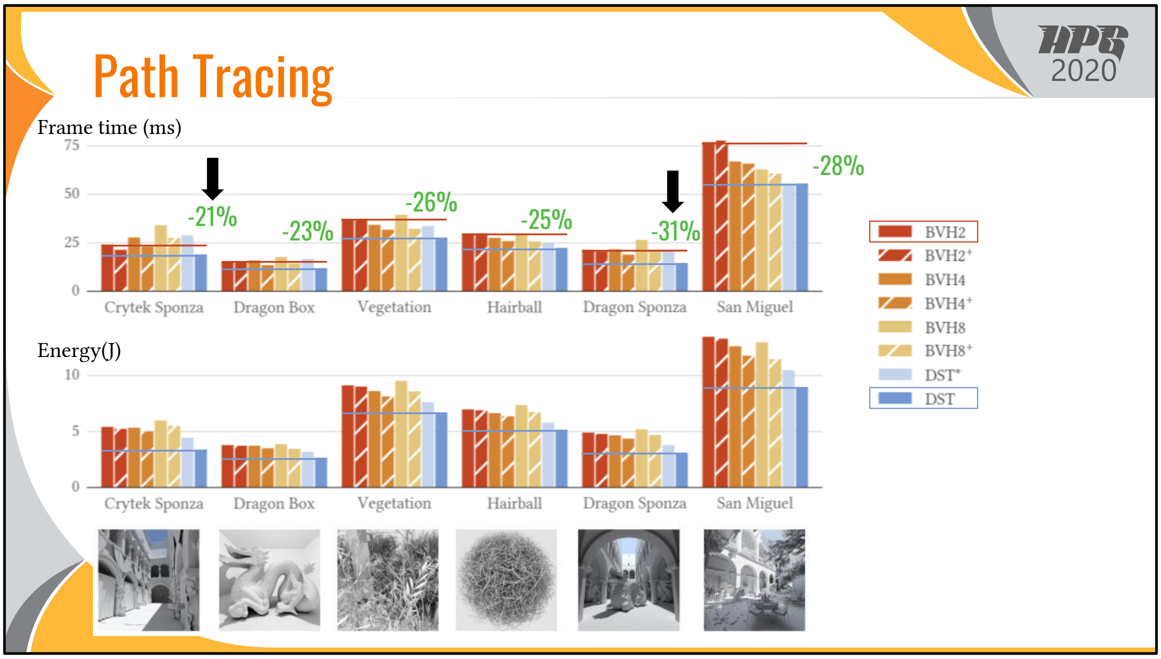
Compared to the baseline BVH2, hardware dual-split trees reduces the render time by 2% - 25%, and reduces the energy consumption by 23% to 41%.



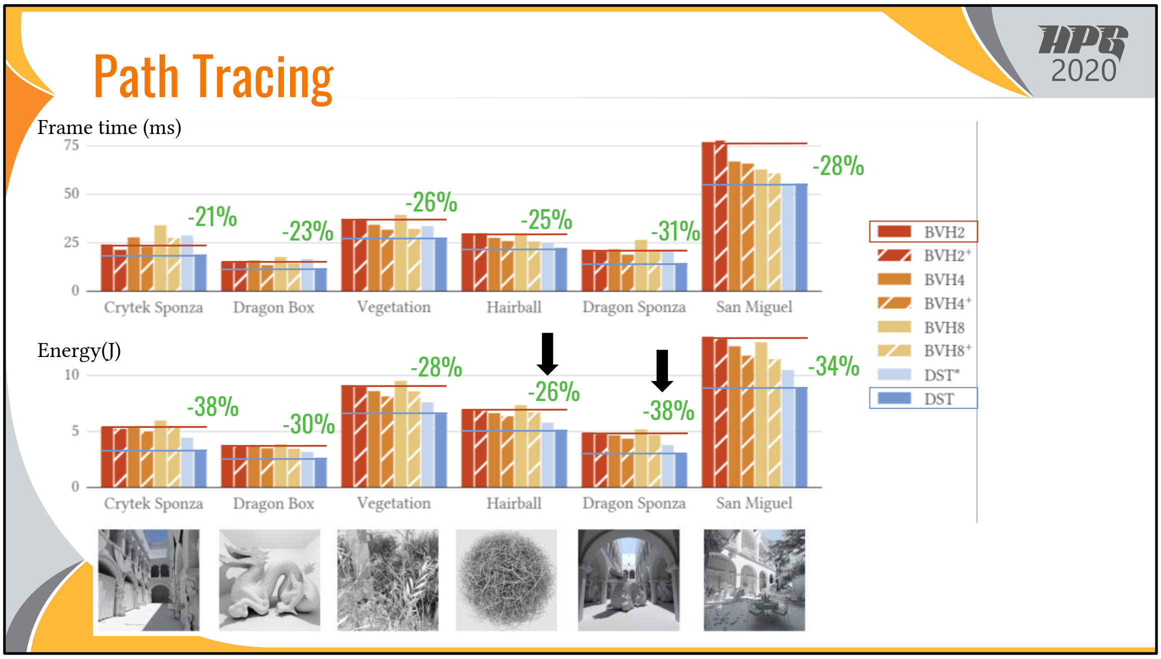
Compared to the software-decoded dual-split trees represented in pale blue, hardware dual-split trees reduces the render time by 20%-34% and reduces the energy by 17%-28%.



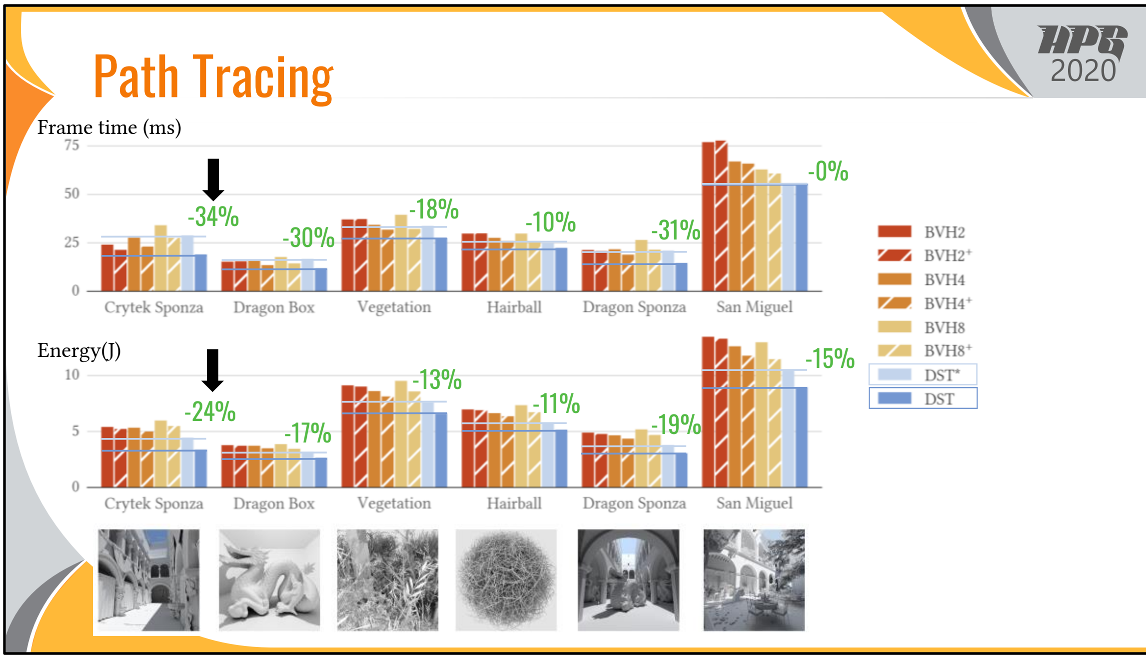
Now we look at the path tracing results. In this case, hardware dual-split trees perform consistently faster and use less energy than all other acceleration structures we tested.



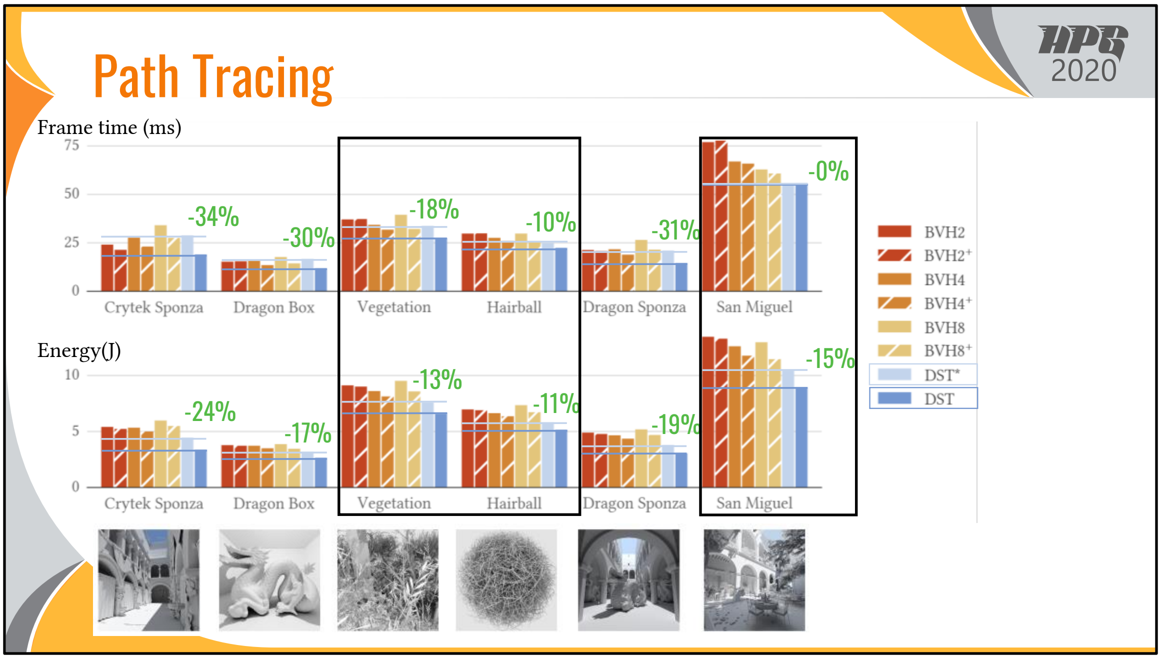
hardware dual-split trees achieve 21%-31% render time reduction compared to the baseline. Notice that the speedup roughly grows with the scene size, since dual-split trees are less memory bound with their reduced memory footprint



hardware dual-split trees also reduce the energy consumption by 26%-38%, far better than other acceleration structures.



Here we show how hardware dual-split trees improves performance over software dual-split trees. hardware dual-split trees reduce the frame time by up to 34% and energy by up to 24%.



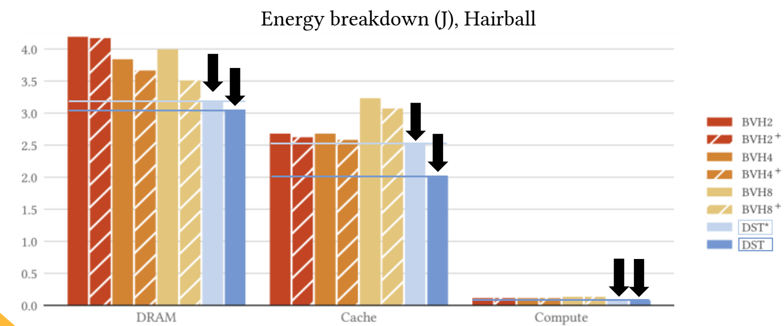
Here we show how hardware dual-split trees improves performance over software dual-split trees. hardware dual-split trees reduce the frame time by up to 34% and energy by up to 24%.

But when the scenes becomes large or have high depth complexity, the performance of software dual-split trees tends to catch up with the hardware dual-split trees as the ray tracing cost is dominated by memory transaction.

Energy Breakdown (path tracing)

HPG 2020

- Both DST and DST* reduce all three kinds of energy consumption as compared to all BVH variants

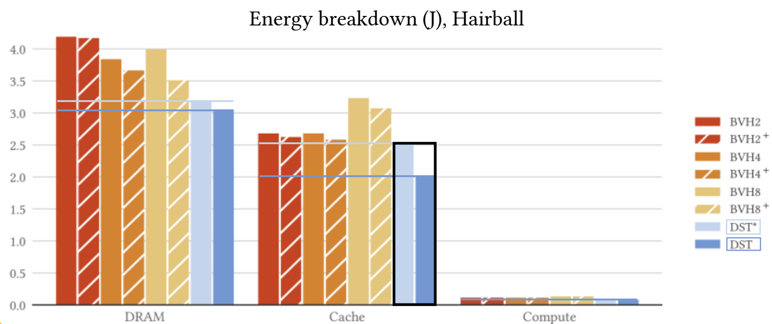


Here, we break down the energy consumption into 3 parts, DRAM, cache and compute in the hairball scene. We can see that that both software and hardware dual-split trees reduce all types of energy as compared to the BVH variants.

Energy Breakdown (path tracing)

HPG 2020

- Both DST and DST* reduce all three kinds of energy consumption as compared to all BVH variants



Here, we break down the energy consumption into 3 parts, DRAM, cache and compute in the hairball scene. We can see that that both software and hardware dual-split trees reduce all types of energy as compared to the BVH variants.
In addition, hardware dual-split trees consume much less cache energy than software dual-split trees, since it reduces the data movement within on-chip storage, which happens in software decoding.

Scalability

HPG 2020

- We increase/decrease the TMs to test scalability
- Dual-split trees and BVH8 scales better than BVH2/4

In terms of scalability, the results in our paper show that dual-split trees scale better than BVH2 and 4 as the number of TMs grows, which is at a similar level as BVH8.

Conclusion

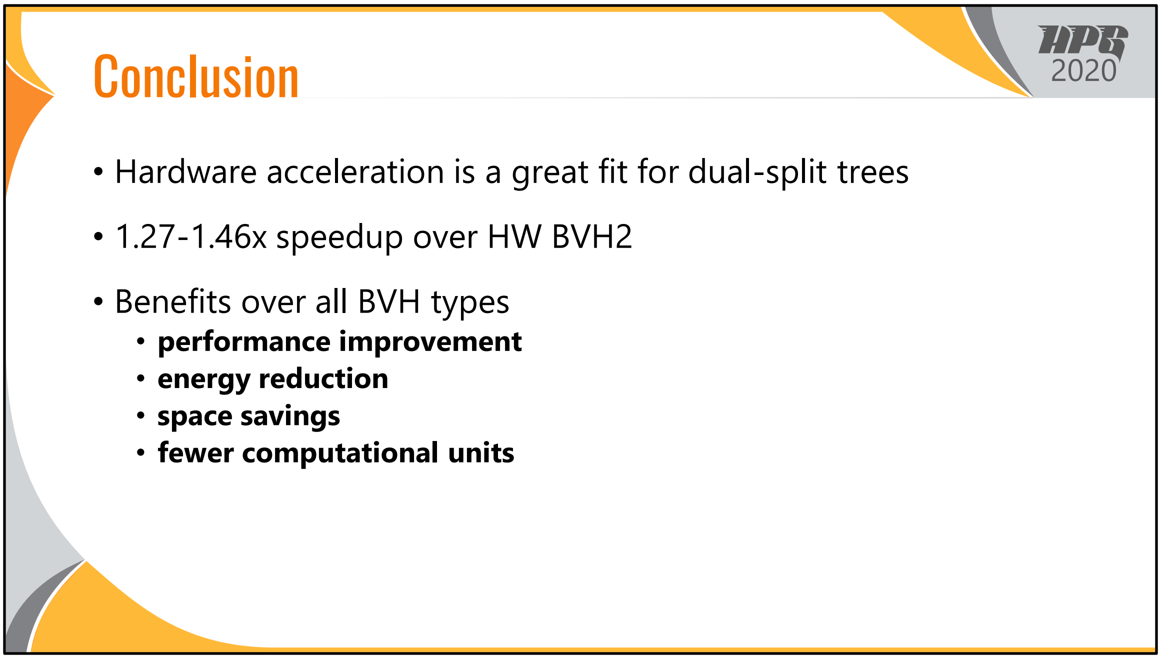
- Hardware acceleration is a great fit for dual-split trees

To sum up, we have introduced hardware-accelerated dual-split trees and demonstrated their advantages using detailed simulations on a general parallel hardware ray tracing architecture.
We found that hardware acceleration is extremely effective in unleashing the performance of dual-split trees,

Conclusion

- Hardware acceleration is a great fit for dual-split trees
- 1.27-1.46x speedup over HW BVH2

To sum up, we have introduced hardware-accelerated dual-split trees and demonstrated their advantages using detailed simulations on a general parallel hardware ray tracing architecture.
We found that hardware acceleration is extremely effective in unleashing the performance of dual-split trees. More specifically, they achieve 1.27-1.46x speedup over hardware-accelerated binary BVHs.

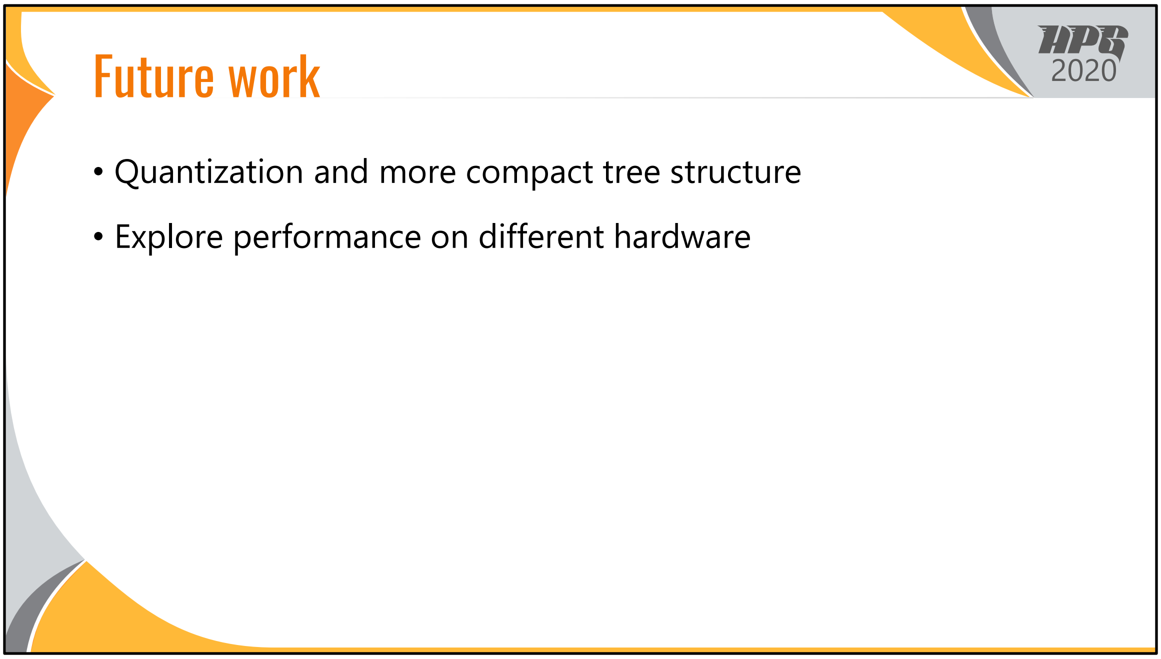


To sum up, we have introduced hardware-accelerated dual-split trees and demonstrated their advantages using detailed simulations on a general parallel hardware ray tracing architecture.
We found that hardware acceleration is extremely effective in unleashing the performance of dual-split trees. More specifically, they achieve 1.27-1.46x speedup over hardware-accelerated binary BVHs.
Compared to hardware-accelerated BVHs with all different branching factors, hardware-accelerated DST can substantially improve the performance and reduce the energy consumption on top of substantial space savings, and the fact that they use significantly fewer computational units.

Future work

- Quantization and more compact tree structure

For future work, one can explore quantization and more compact tree structure for dual-split trees. For example, future work can find out whether dual-split trees with collapsed tree levels and plane quantization provides the same amount of speedup as compared with the case for BVHs.



For future work, one can explore quantization and more compact tree structure for dual-split trees. For example, future work can find out whether dual-split trees with collapsed tree levels and plane quantization provides the same amount of speedup as compared with the case for BVHs.

Another interesting future work is to explore the performance on different hardware architecture.

Hardware-Accelerated Dual-Split Trees

Daqi Lin, Elena Vasiou, Cem Yuksel, Daniel Kopta, Erik Brunvand

University of Utah

HPG 2020