Ray Tracing in Diablo IV

Kevin Todisco
Principal Software Engineer, Blizzard Entertainment

RENDERING
ENGINE
ARCHITECTURE
CONFERENCE
REAC 24

Agenda

- Phases of Ray Tracing Development
- Constraints and Strategy
- Implementation Details and How Challenges Shape Architecture
- Fitting to the Content
- Summing Up

Chronology

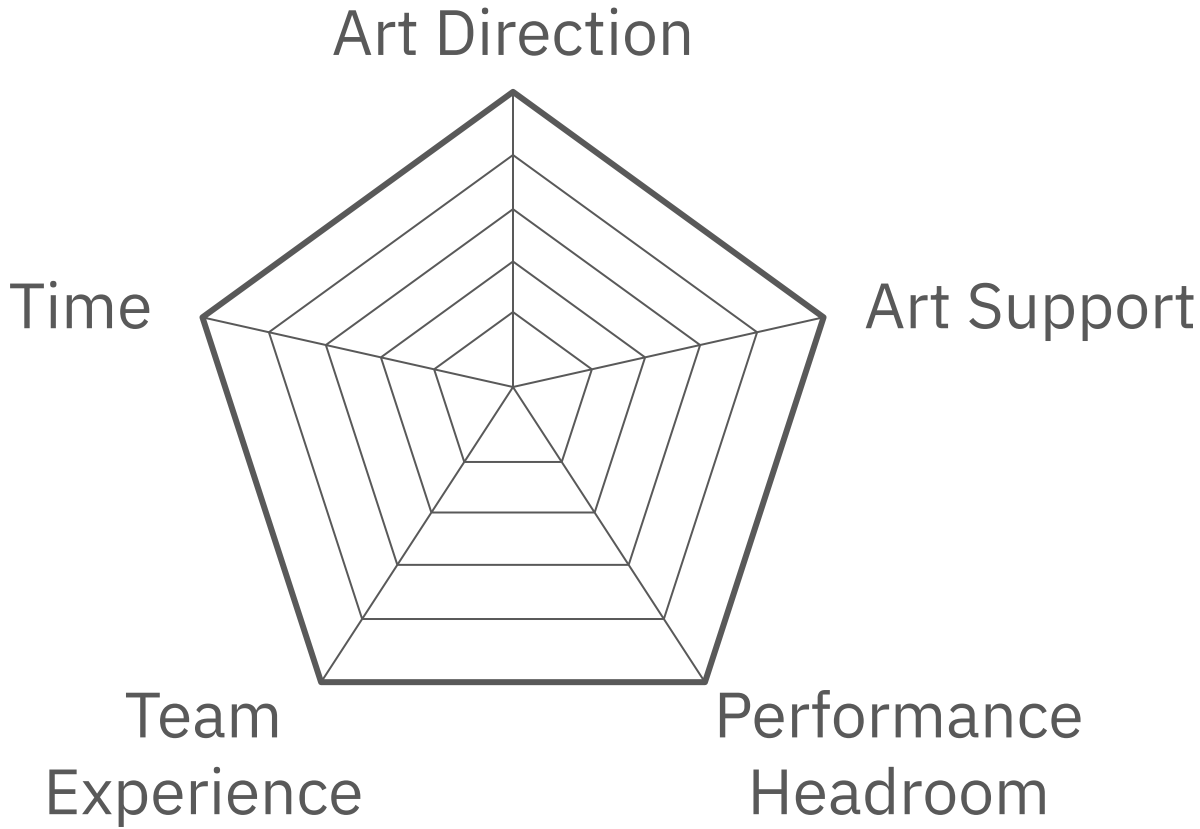
- Began as engineering R&D with no release timeline
- Sat latent for a while
- Picked up once release window came into focus
  - Plus, partnership with NVIDIA

- Goes from side project to "how do we ship this?"

- Nearly all of what's discussed here is from that second stage.

Decision Making

What to ray trace?



Art Direction and Support

- Not directed with ray tracing in mind
- Diablo IV ships on a variety of hardware
  - PC min spec is a GTX 660 or R9 280
  - Xbox One and PS4
- Want to enhance visuals, but not have a new look
- Not wanting to change content
  - Good from the standpoint of asset management and maintenance
  - Challenge from the standpoint of implementation

Time and Experience

- No prior ray tracing technology in the stack
  - Good references, but no implementation
- No prior ray tracing API experience on the team
- Not just one API to consider
  - DXR (PC, Xbox) and PSR (PS5)
- Defer release until after initial launch

Performance Headroom

	Xbox One	PS4	Xbox Series X	PS5	PC
Framerate	30	30	60	60	30-60+
Output Resolution	1080p	1080p	2160p	2160p	720p - 2160p+
Quality	Low	Low	Medium	Medium	Low - Ultra

Performance Headroom

	Xbox One	PS4	Xbox Series X		PS5		PC
Framerate	30	30	60	30	60	30	30-60+
Output Resolution	1080p	1080p	2160p	2160p	2160p	2160p	720p - 2160p+
Quality	Low	Low	Medium	High	Medium	High	Low - Ultra
Ray Tracing	No	No	No	Yes	No	Yes	No - Yes

🟦 Performance 🟨 Quality

Impact

- Solves the most of Art's problems
  - Without changing too much
- Enhances the visuals of the game
  - Without changing too much
- Maximizes cost-to-benefit ratio
  - While changing enough to be an upgrade

Feature Set

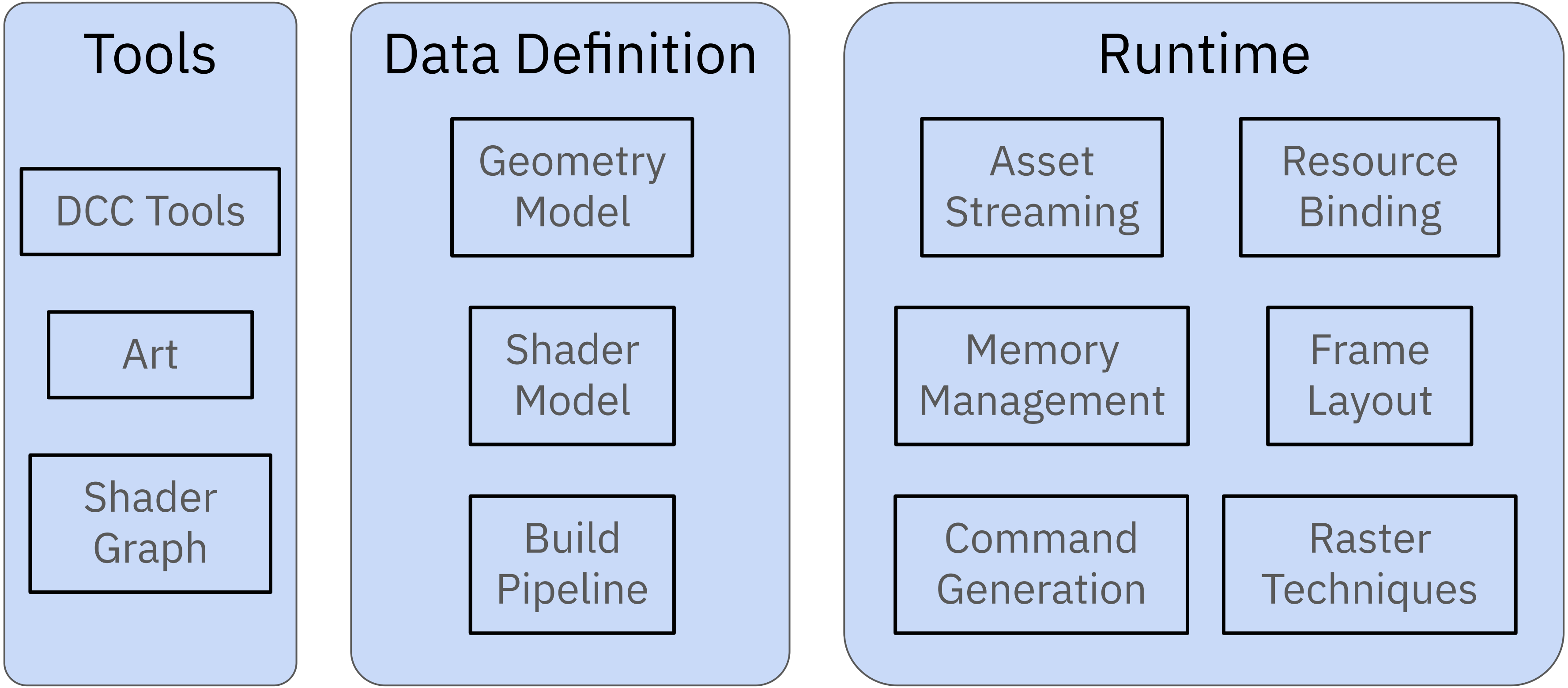
- Tried-and-true techniques: shadows and reflections
- Lowers R&D cost
- Shadows
  - Thematically relevant
- Reflections
  - Straightforward to implement
- Still need to account for variety of specs

Feature Set

Shadows	Low	Medium	High
Directional Lights	✔	✔	✔
Player Light	✖	✔	✔
Local Lights	✖	✖	✔

Reflections	Low	High
Roughness-based multi-ray	✖	✔
Simple blur	✔	✖
High Quality Denoise	✖	✔

Implementing a Foundation



Tools

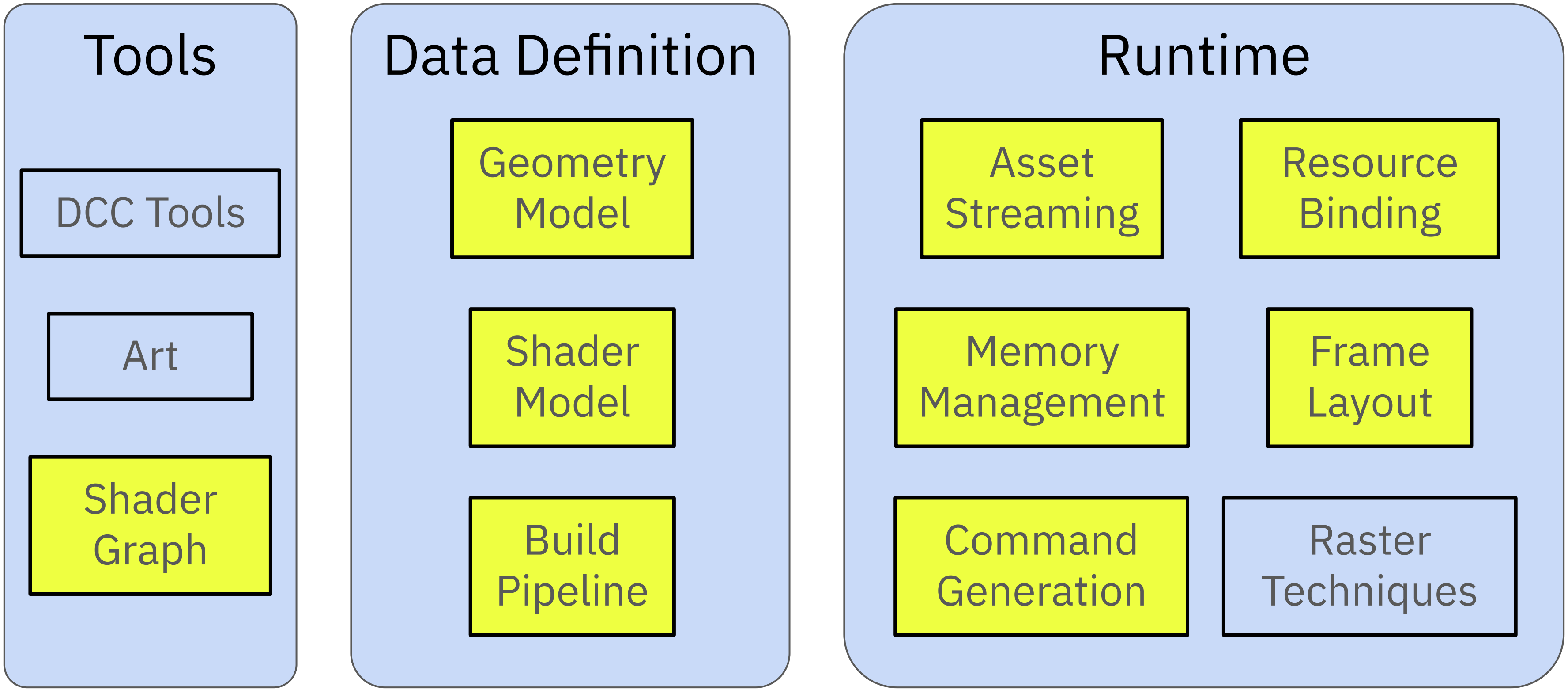
- DCC Tools
- Art
- Shader Graph

Data Definition

- Geometry Model
- Shader Model
- Build Pipeline

Runtime

- Asset Streaming
- Resource Binding
- Memory Management
- Frame Layout
- Command Generation
- Raster Techniques



Tools

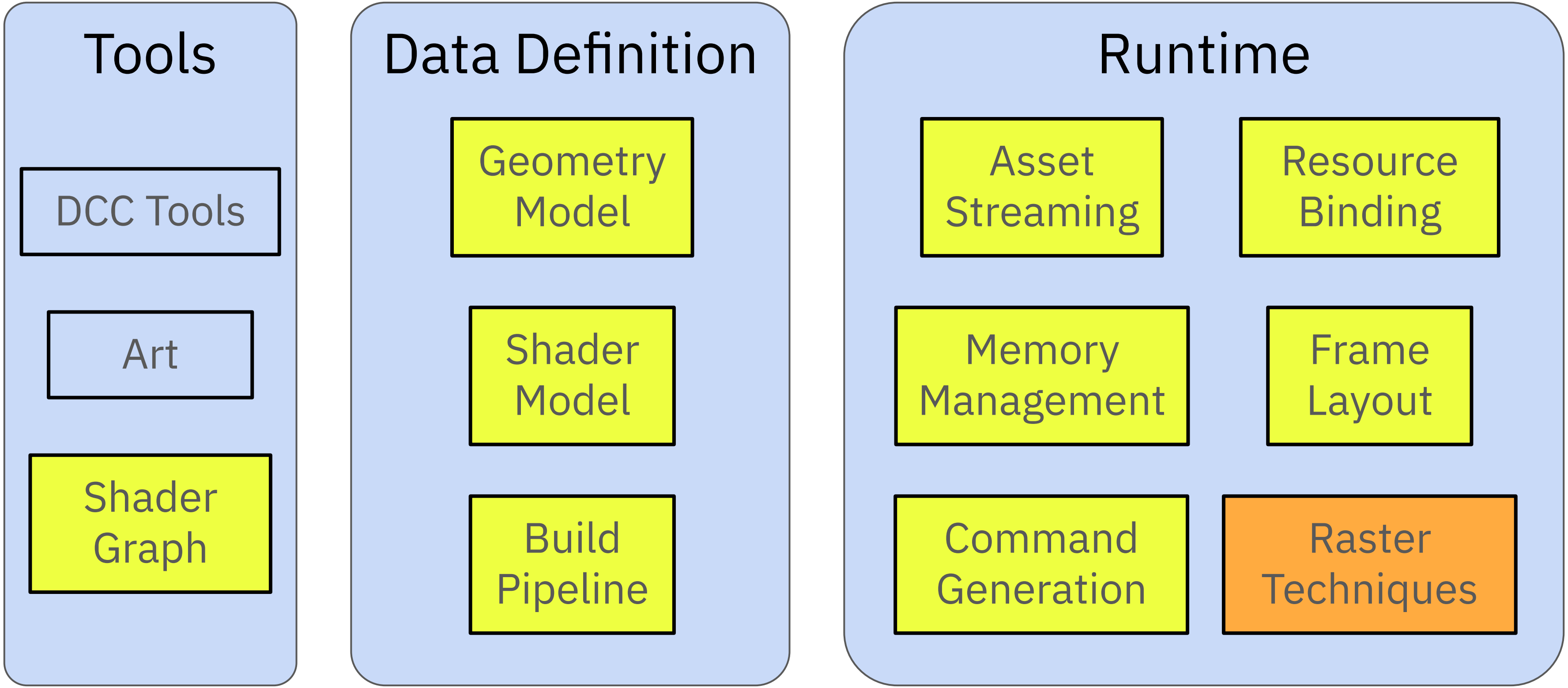
- DCC Tools
- Art
- Shader Graph

Data Definition

- Geometry Model
- Shader Model
- Build Pipeline

Runtime

- Asset Streaming
- Resource Binding
- Memory Management
- Frame Layout
- Command Generation
- Raster Techniques



Tools

DCC Tools

Art

Shader Graph

Data Definition

Geometry Model

Shader Model

Build Pipeline

Runtime

Asset Streaming

Resource Binding

Memory Management

Frame Layout

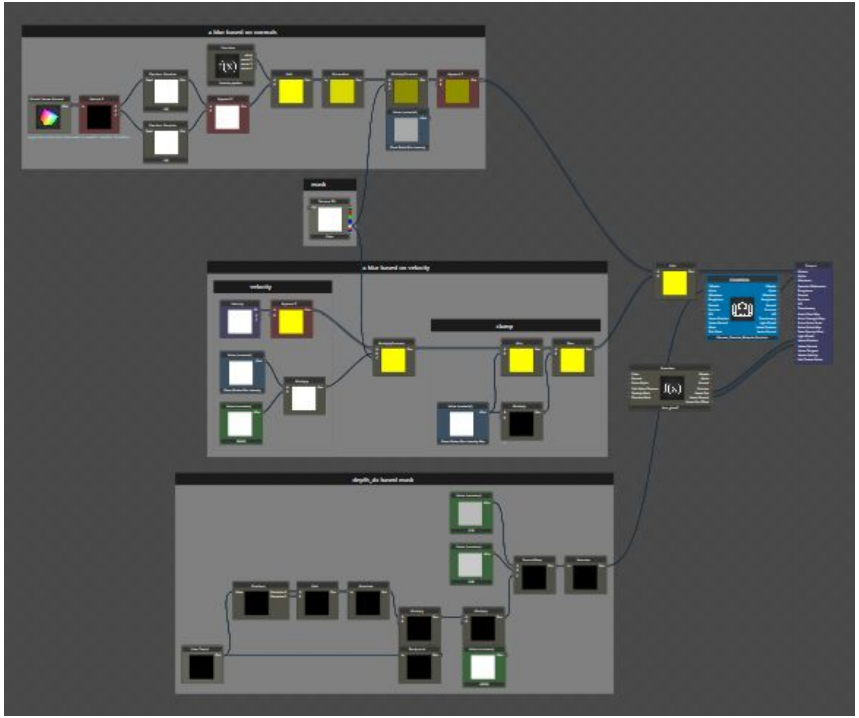
Command Generation

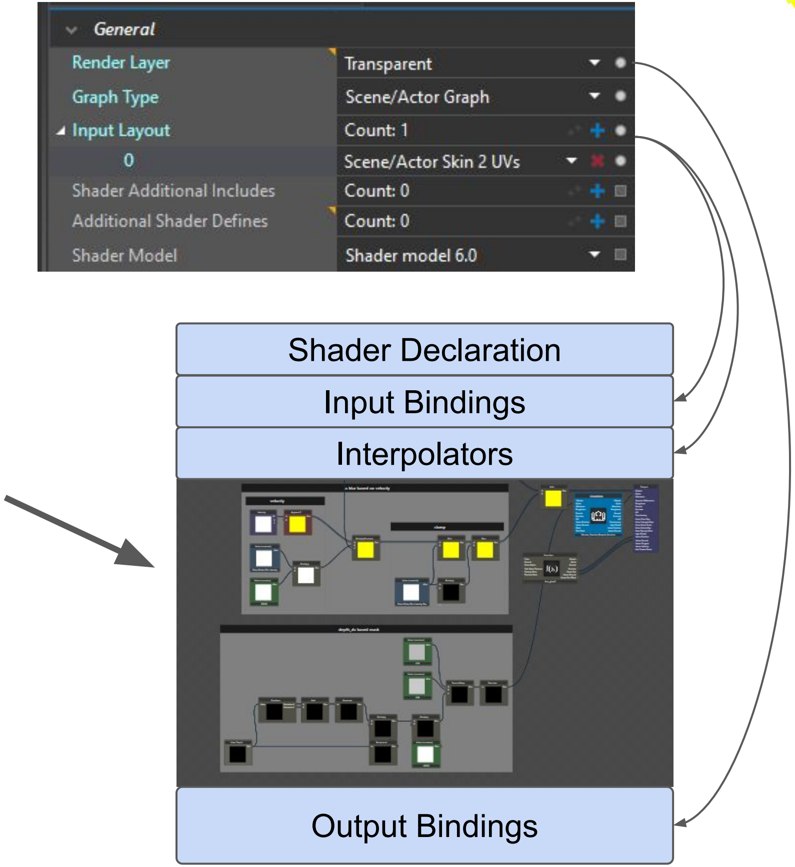
Raster Techniques

Shader Implementation Details

- No bindless paradigm.
  - Hit group textures use register space 1
  - Vertex and index buffers start at slot 52, after raster bindings
- Vertex processing is done on async compute
  - Results are cached
  - Two classes of assets: skinned actors and SpeedTree
  - This was (conveniently) done independent of ray tracing
  - SpeedTree precompute is only on when ray tracing is on
  - Increases required memory

Shader Graphs

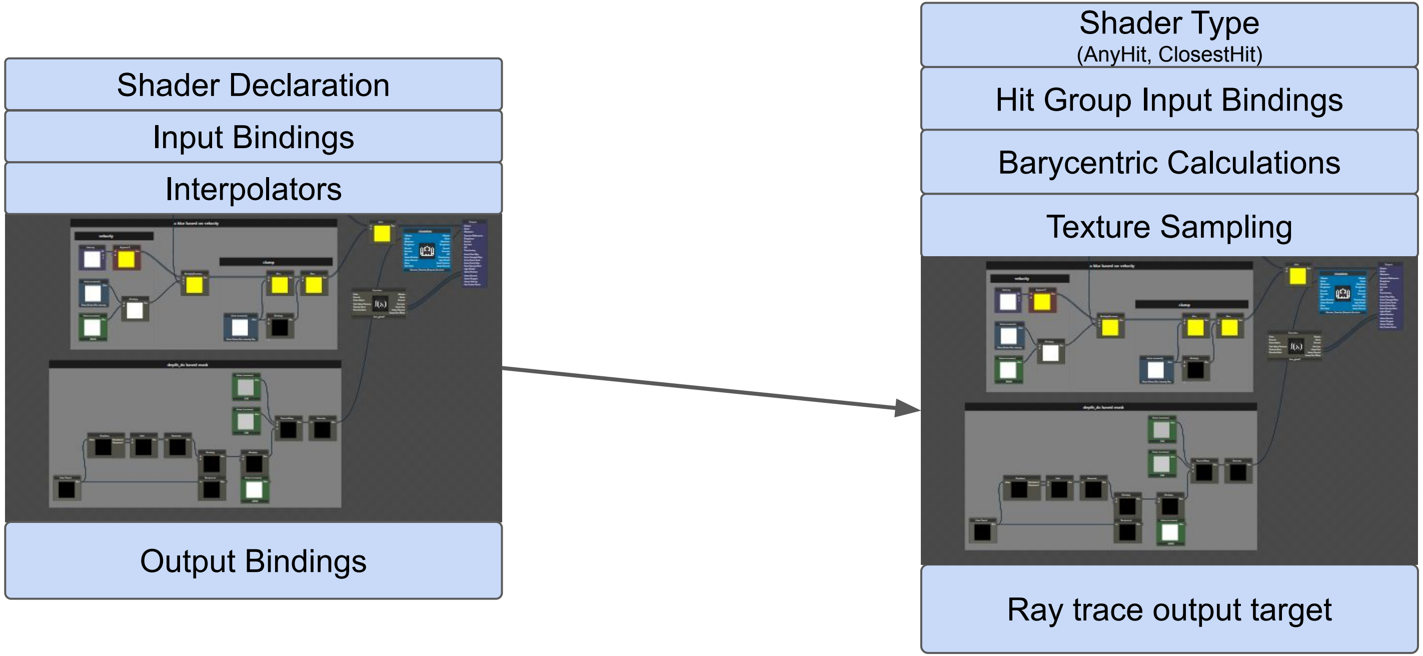




General

Render Layer	Transparent
Graph Type	Scene/Actor Graph
Input Layout	Count: 1
0	Scene/Actor Skin 2 UVs
Shader Additional Includes	Count: 0
Additional Shader Defines	Count: 0
Shader Model	Shader model 6.0

Shader Graphs



Shader Declaration

Input Bindings

Interpolators

Output Bindings

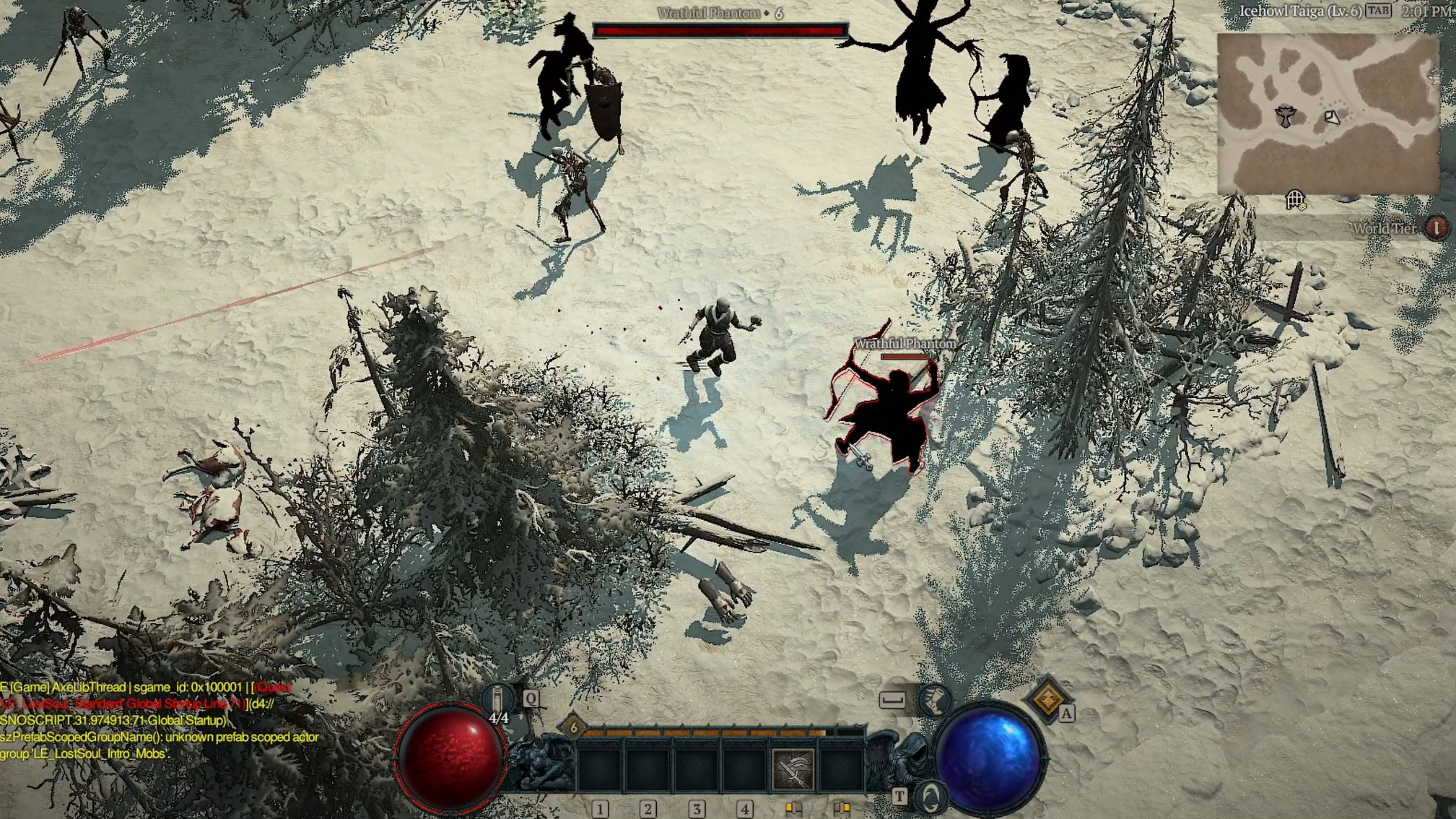
Shader Type
(AnyHit, ClosestHit)

Hit Group Input Bindings

Barycentric Calculations

Texture Sampling

Ray trace output target



```
E [Game] AxeLibThread | sgame_id: 0x100001 | [(Quest
'LE_LostSoul_Standard' Global Startup Line 71)](d4://
SNOSCRIPT.31.974913.71.Global Startup)
szPrefabScopedGroupName(): unknown prefab scoped actor
group 'LE_LostSoul_Intro_Mobs'.
```

Render Work Generation

- 1 render thread, 4 render workers
  - Work is separated manually among workers
  - 1 worker for early-frame work like shadows, 1 for gbuffer, 1 for post, etc.
- Each worker has multiple state machines
  - Global state describing high-level state of the graphics pipe
  - Thread local state describing low-level details like active command lists and bound resources
- Scene traversal modifies high-level and low-level state
- Issuing a draw translates state into command list ops
- State modification is a very hot path!

- Ray tracing work is constructed similarly

Building RT Shader Tables

- New low-level thread-local RT state is added per worker
  - Tracks bound hit-group resources, PSO build, shader binding tables, and the active top-level acceleration structure
- A top-level acceleration structure (TLAS) is built from beginning to end on one render worker.
  - One TLAS can not be built by multiple workers
- PSO creation is deferred, and incremental build is used on supported platforms

Overview of Scene Traversal

- Visibility tests bucket objects into different display lists
  - Display lists are enumerated, named lists, max of 64
  - Examples include Gbuffer, Transparent, Shadows, Reflections
- Display lists are iterated over to issue pipeline state and draw commands
- For ray tracing, each technique is executed in similar steps
  - GatherShaderLibraries - Assemble the pipeline object
  - GatherInstances - Assemble the top level acceleration structure
  - TraceRays
- Gathering libraries and gathering instances must behave identically

Example Traversal

Traverse Object0

AddInstance()

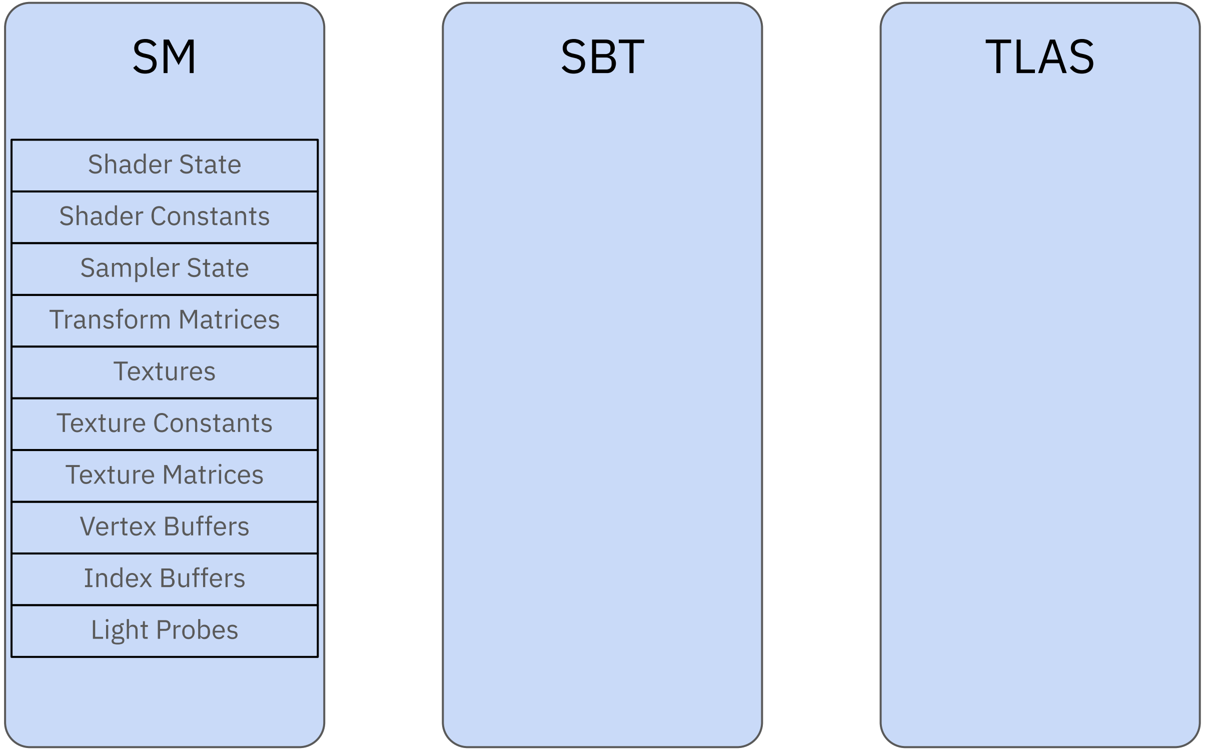
Traverse Object1

AddInstance()

Traverse Object2

AddInstance()

...



SM

Shader State

Shader Constants

Sampler State

Transform Matrices

Textures

Texture Constants

Texture Matrices

Vertex Buffers

Index Buffers

Light Probes

SBT

TLAS

Example Traversal

➡ Traverse Object0

AddInstance()

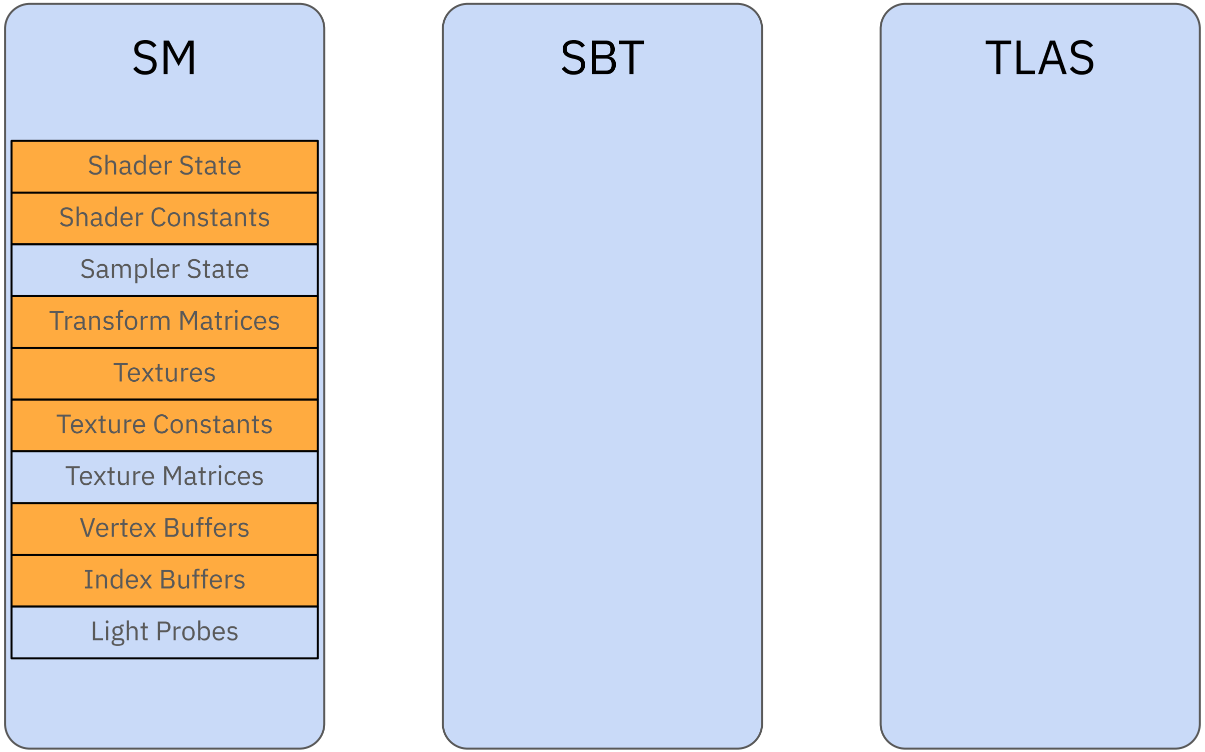
Traverse Object1

AddInstance()

Traverse Object2

AddInstance()

…



SM

Shader State

Shader Constants

Sampler State

Transform Matrices

Textures

Texture Constants

Texture Matrices

Vertex Buffers

Index Buffers

Light Probes

SBT

TLAS

Example Traversal

Traverse Object0

AddInstance()

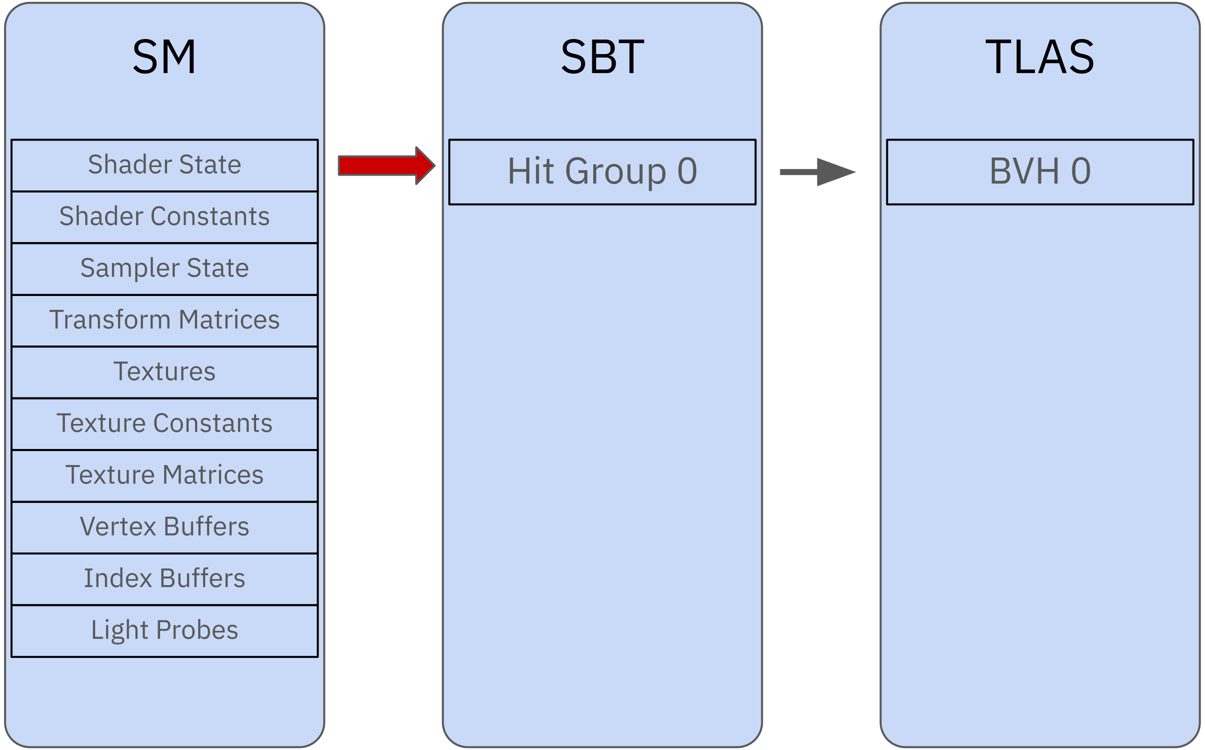
Traverse Object1

AddInstance()

Traverse Object2

AddInstance()

...



SM

Shader State

Shader Constants

Sampler State

Transform Matrices

Textures

Texture Constants

Texture Matrices

Vertex Buffers

Index Buffers

Light Probes

SBT

Hit Group 0

TLAS

BVH 0

Example Traversal

Traverse Object0

AddInstance()

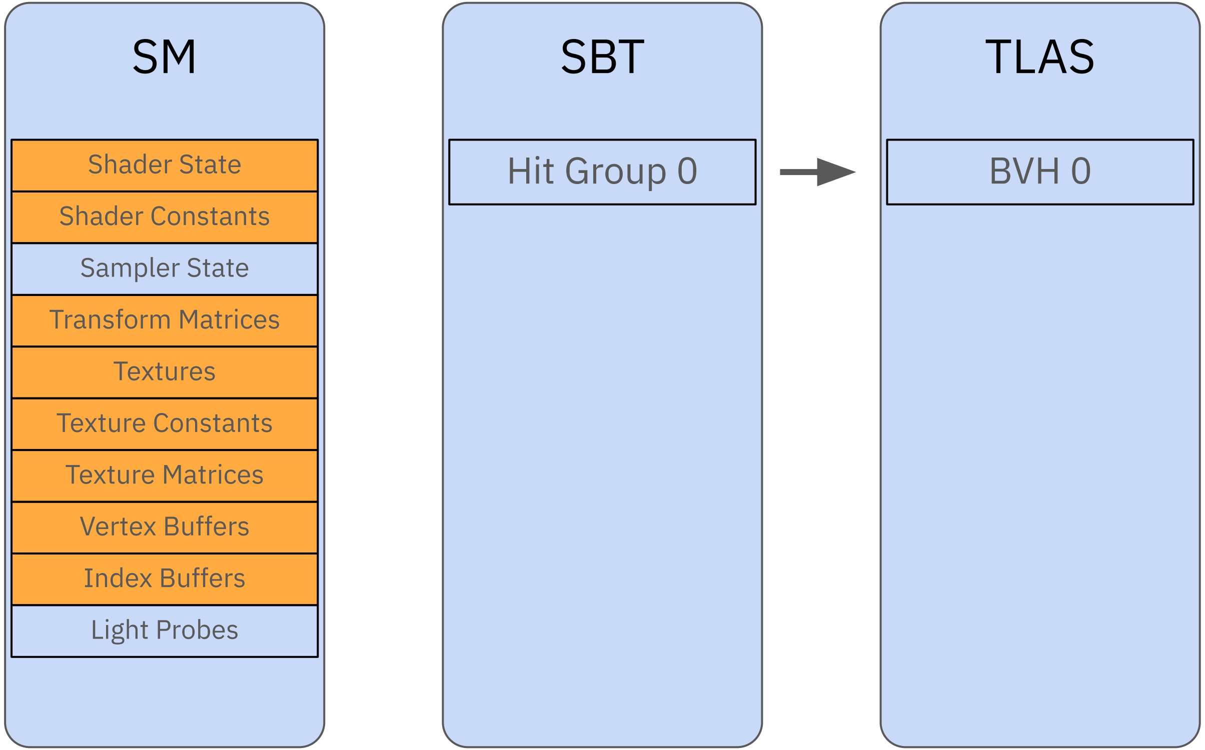
→ Traverse Object1

AddInstance()

Traverse Object2

AddInstance()

...



Example Traversal

Traverse Object0

AddInstance()

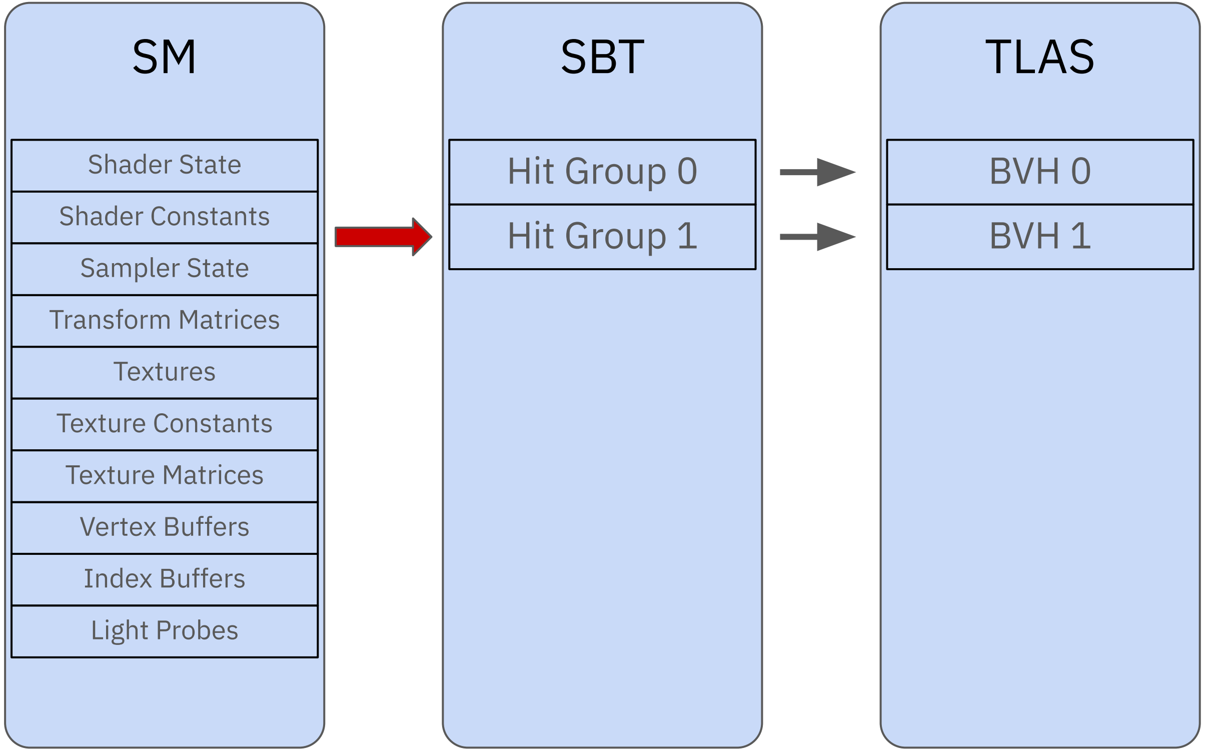
Traverse Object1

AddInstance()

Traverse Object2

AddInstance()

...



SM

Shader State

Shader Constants

Sampler State

Transform Matrices

Textures

Texture Constants

Texture Matrices

Vertex Buffers

Index Buffers

Light Probes

SBT

Hit Group 0

Hit Group 1

TLAS

BVH 0

BVH 1

Example Traversal

Traverse Object0

AddInstance()

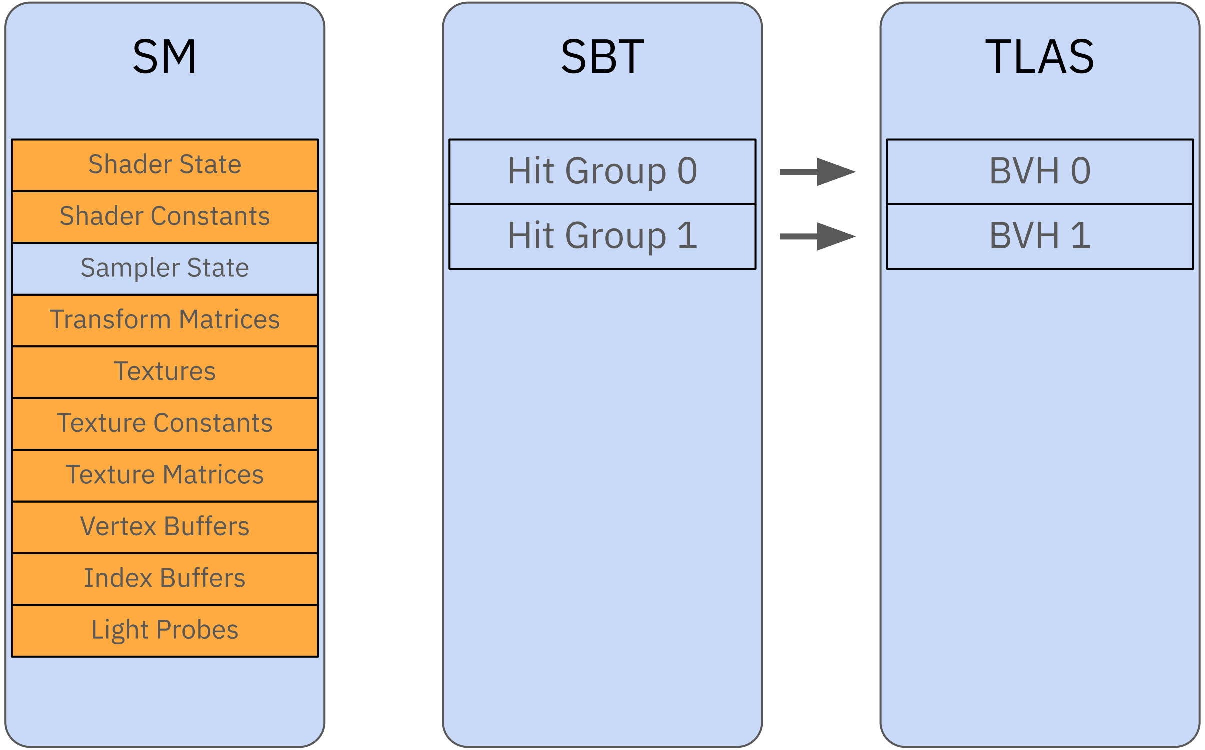
Traverse Object1

AddInstance()

➡ Traverse Object2

AddInstance()

...



SM

Shader State

Shader Constants

Sampler State

Transform Matrices

Textures

Texture Constants

Texture Matrices

Vertex Buffers

Index Buffers

Light Probes

SBT

Hit Group 0

Hit Group 1

TLAS

BVH 0

BVH 1

Example Traversal

Traverse Object0

AddInstance()

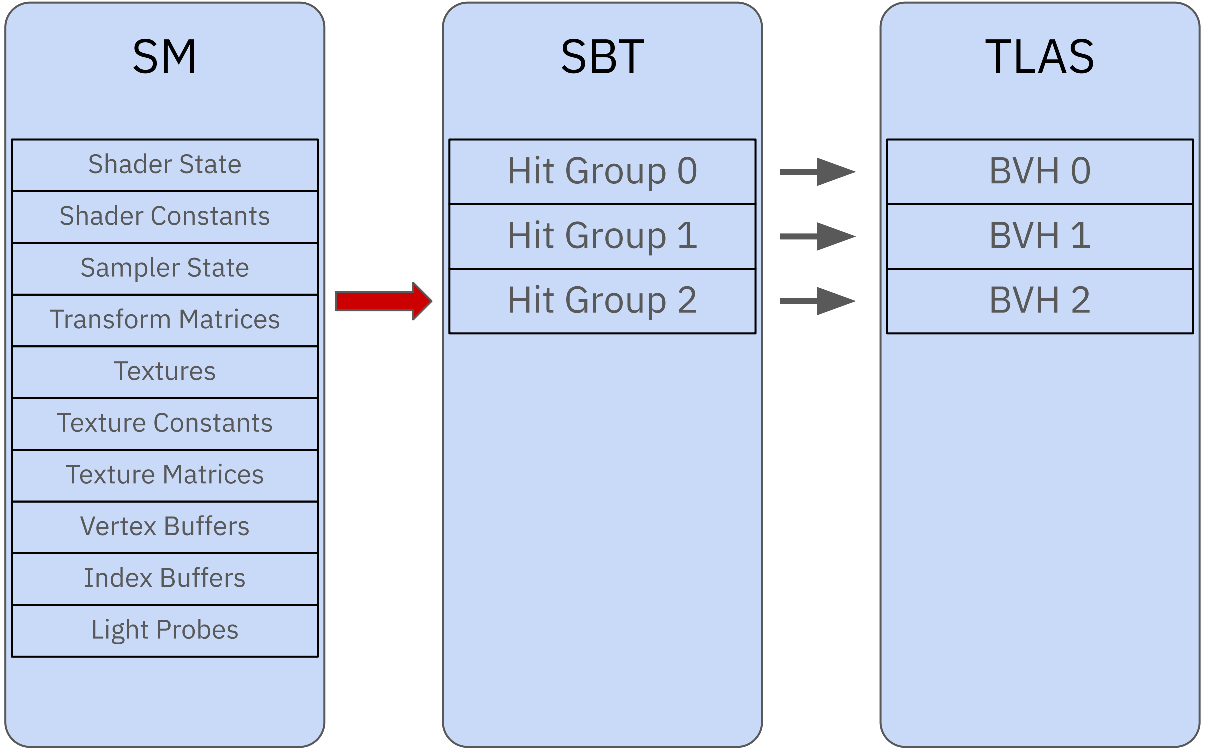
Traverse Object1

AddInstance()

Traverse Object2

→ AddInstance()

...



Example Traversal

Traverse Object0

AddInstance()

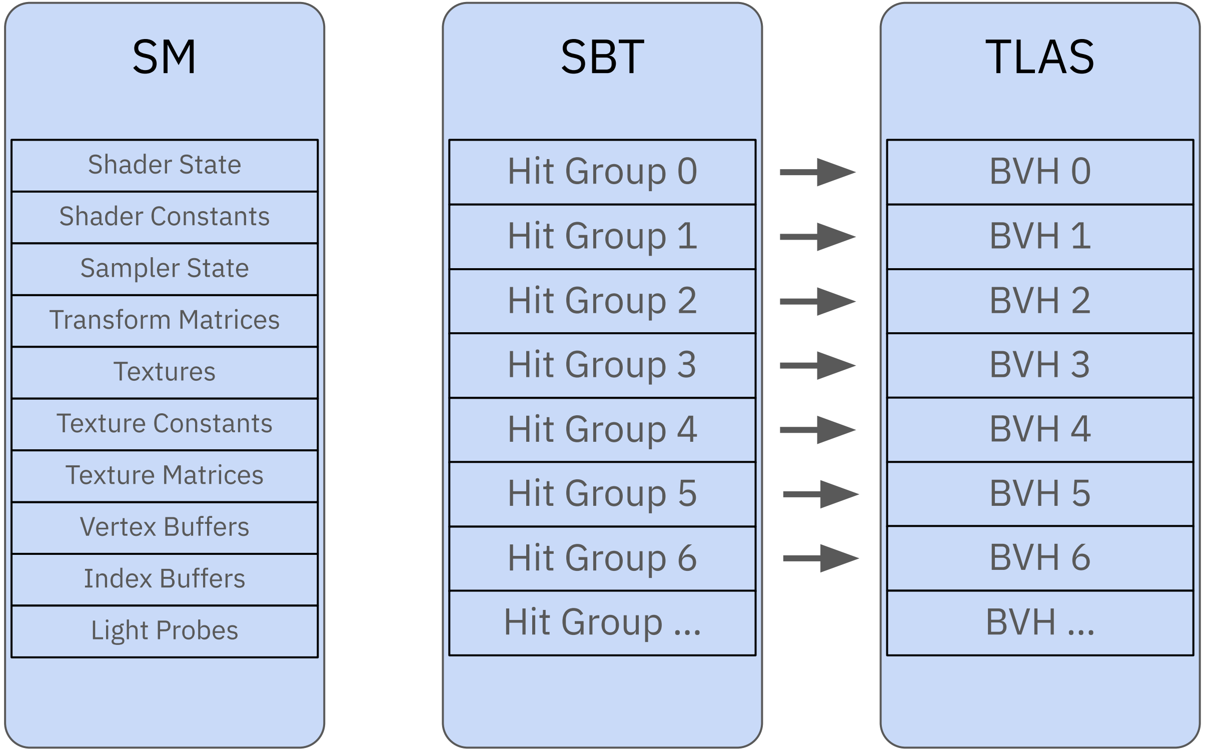
Traverse Object1

AddInstance()

Traverse Object2

AddInstance()

→ ...



SM

Shader State

Shader Constants

Sampler State

Transform Matrices

Textures

Texture Constants

Texture Matrices

Vertex Buffers

Index Buffers

Light Probes

SBT

Hit Group 0 → BVH 0

Hit Group 1 → BVH 1

Hit Group 2 → BVH 2

Hit Group 3 → BVH 3

Hit Group 4 → BVH 4

Hit Group 5 → BVH 5

Hit Group 6 → BVH 6

Hit Group ...

TLAS

BVH 0

BVH 1

BVH 2

BVH 3

BVH 4

BVH 5

BVH 6

BVH ...

Improving Performance

- Raytracing involves many more objects than primary game camera visibility
    - Objects behind the camera, outside the main frustum
- This puts a strain on our existing architecture
    - Our hot path on the CPU becomes even hotter.
- Changing architecture would be... massive

- Don't change the architecture, change the hit count.

Improving Performance

Traverse Object0

AddInstance()

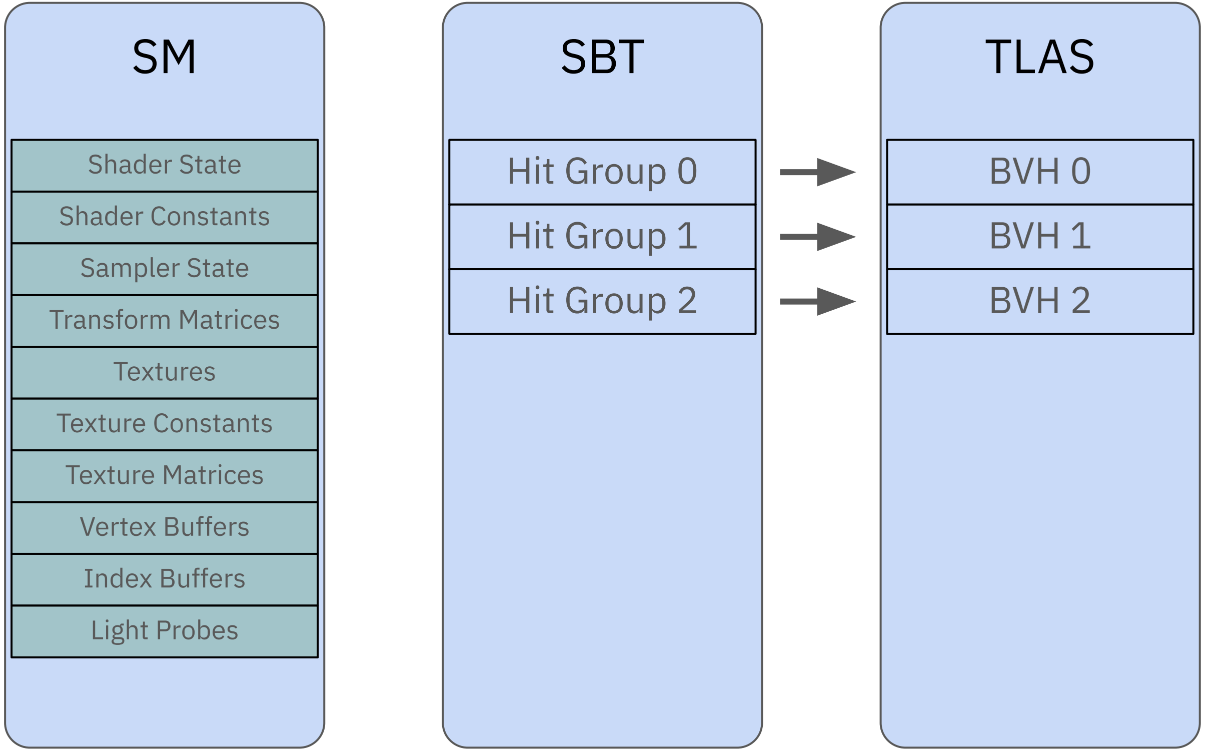
Traverse Object1

AddInstance()

Traverse Object2

AddInstance()

...



SM

- Shader State
- Shader Constants
- Sampler State
- Transform Matrices
- Textures
- Texture Constants
- Texture Matrices
- Vertex Buffers
- Index Buffers
- Light Probes

SBT

- Hit Group 0 → BVH 0
- Hit Group 1 → BVH 1
- Hit Group 2 → BVH 2

TLAS

- BVH 0
- BVH 1
- BVH 2

Improving Performance

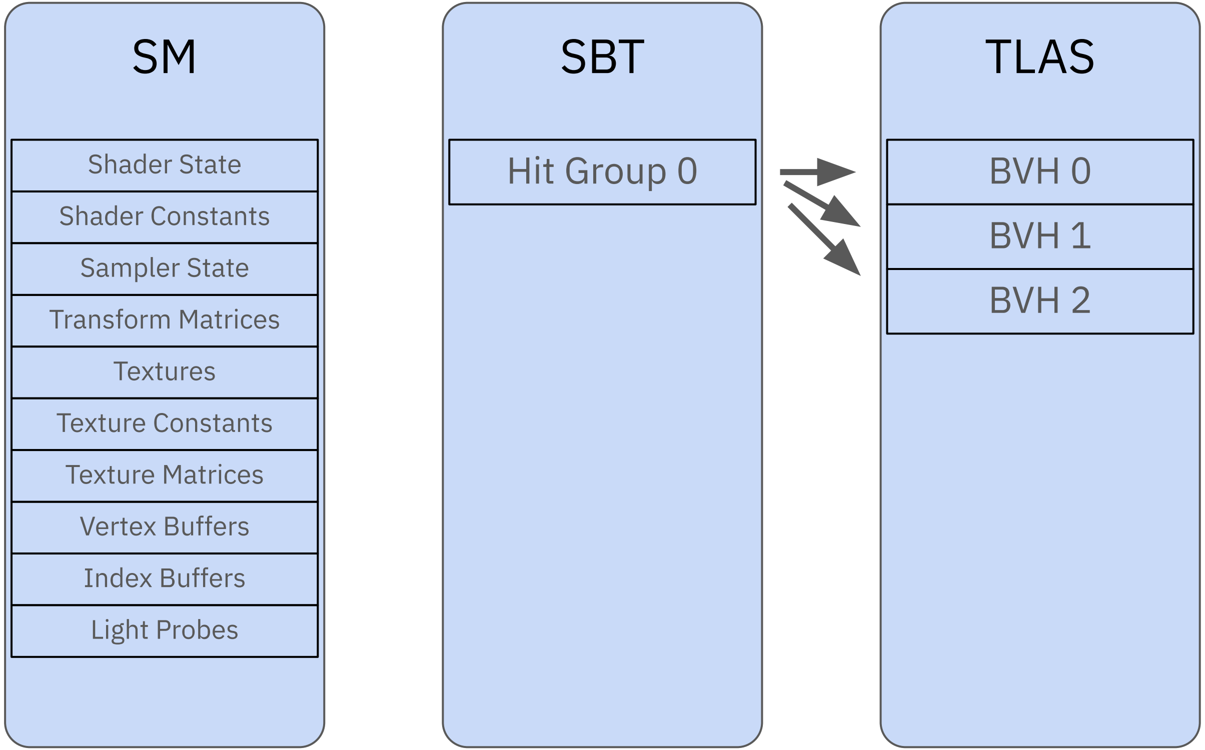
Traverse Object0

Count Object1

Count Object2

AddInstances()

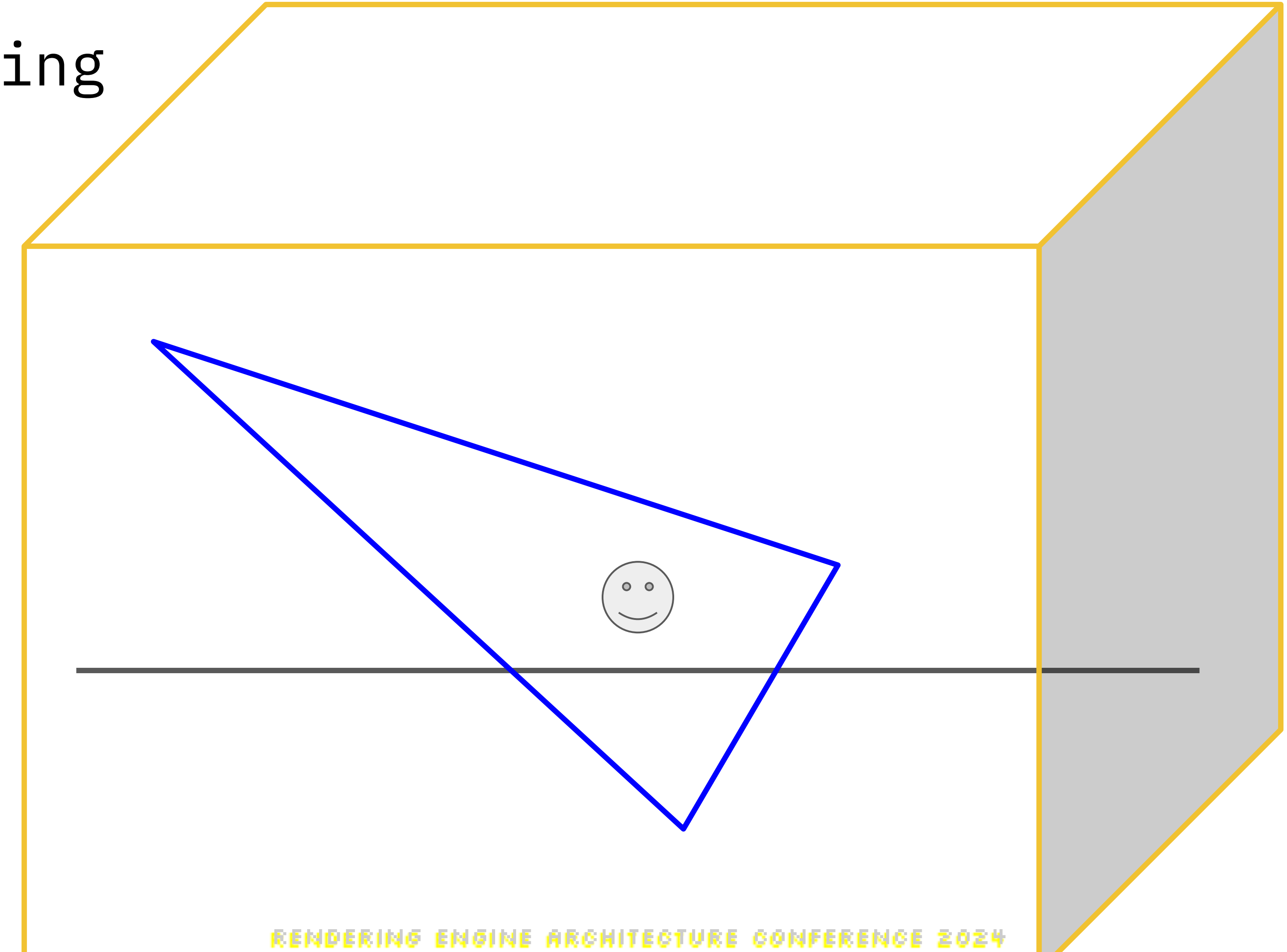
...



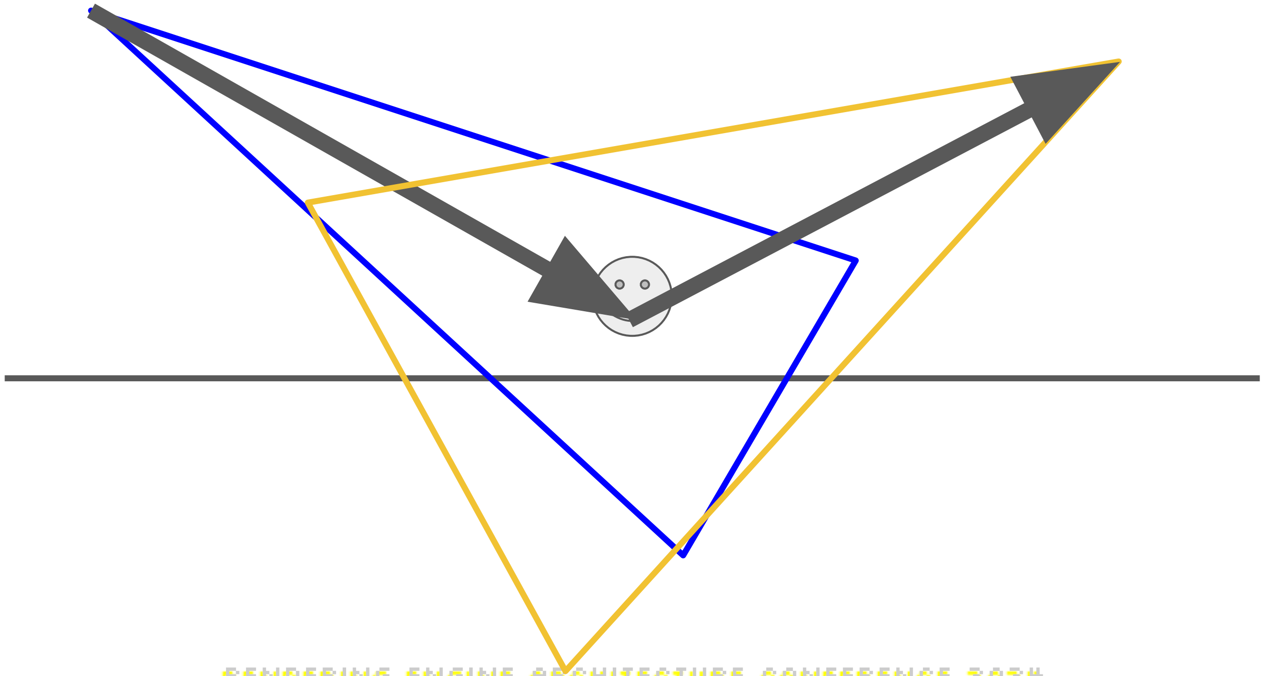
Culling

- Both techniques started with naive area-based culling
- Game camera is fixed
  - Can take advantage of this for reflections
  - But...
- Still need to consider in-game cutscenes
- We forgo a specialized solution and instead choose a generic one

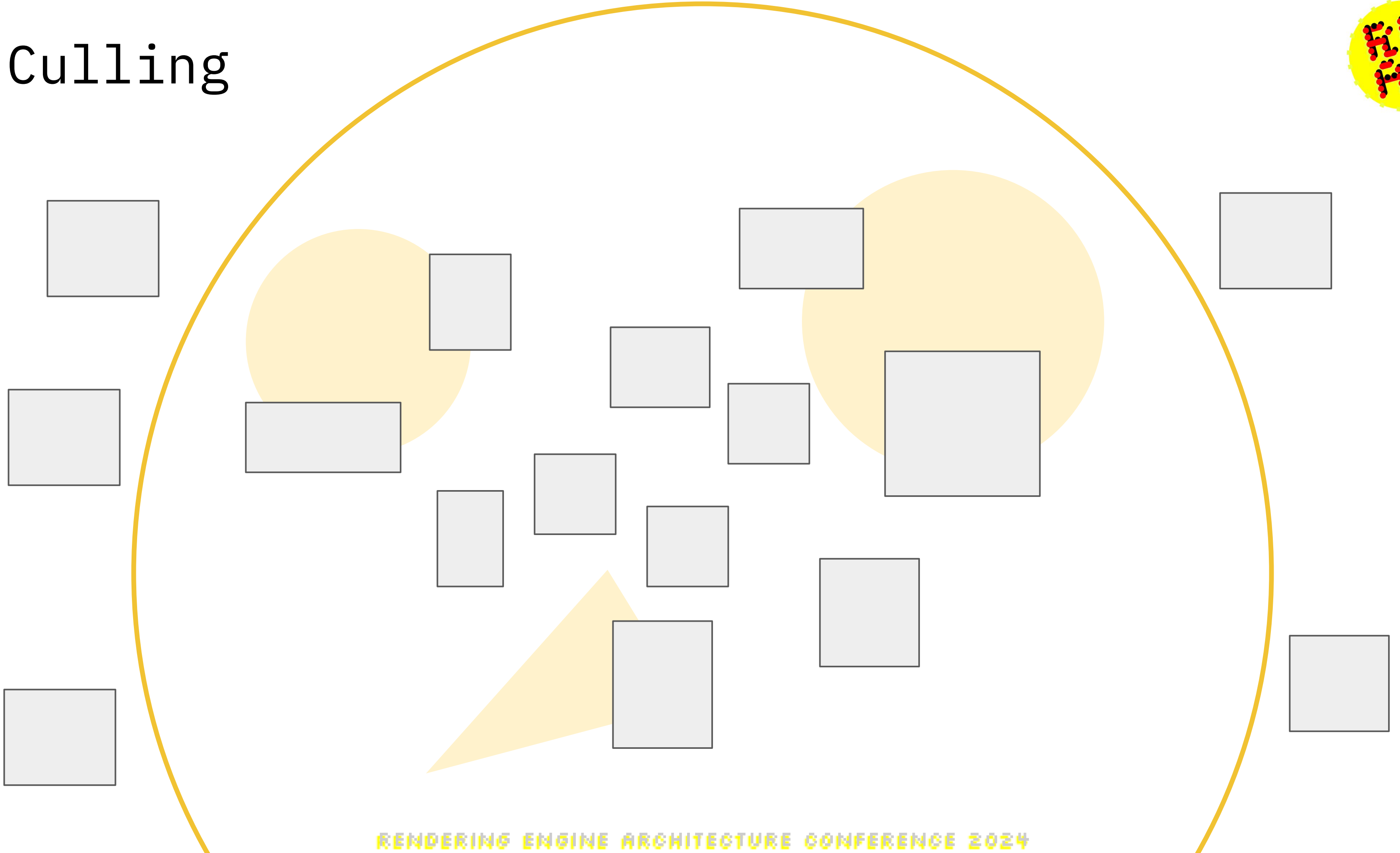
Culling



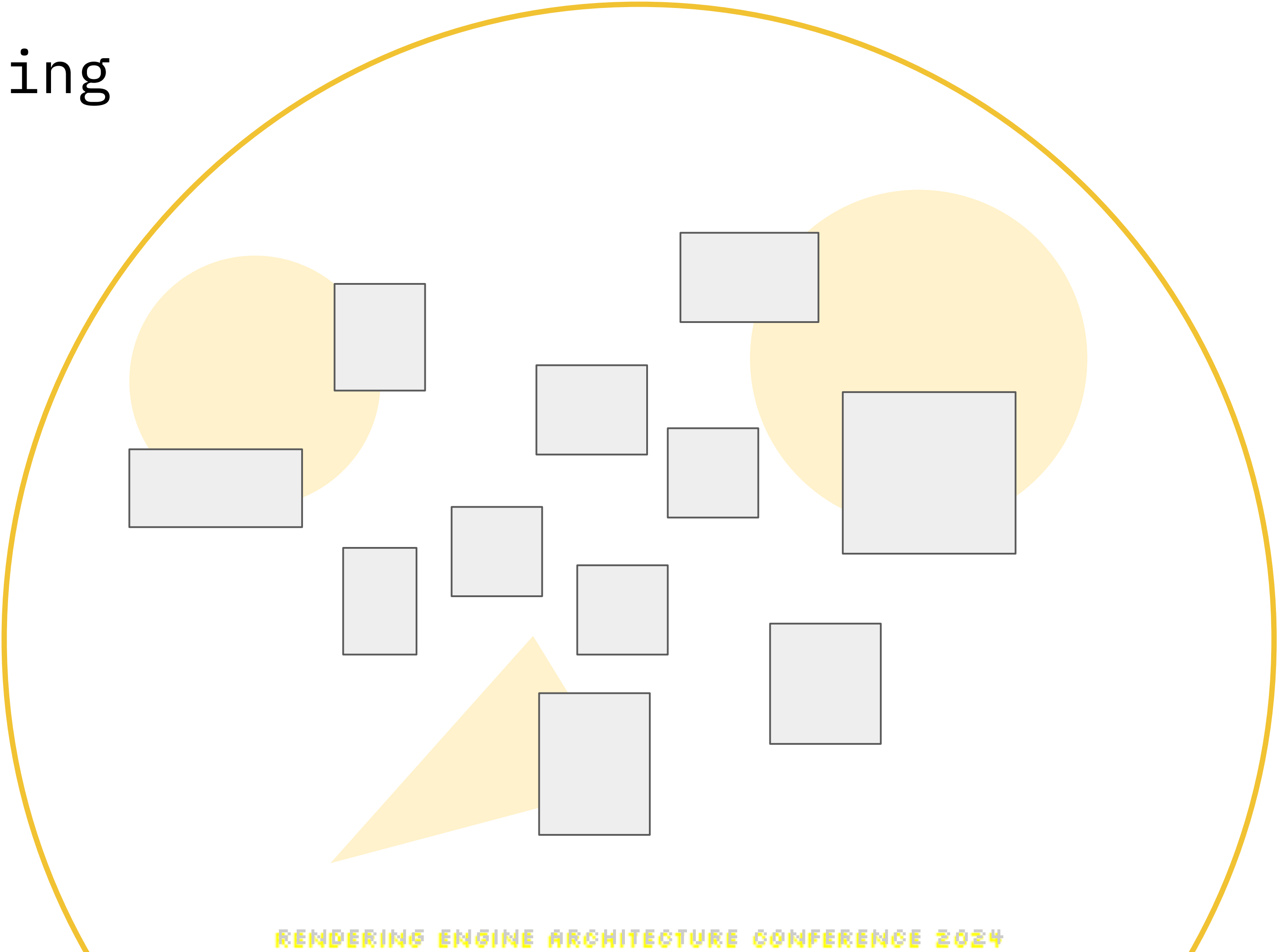
Culling



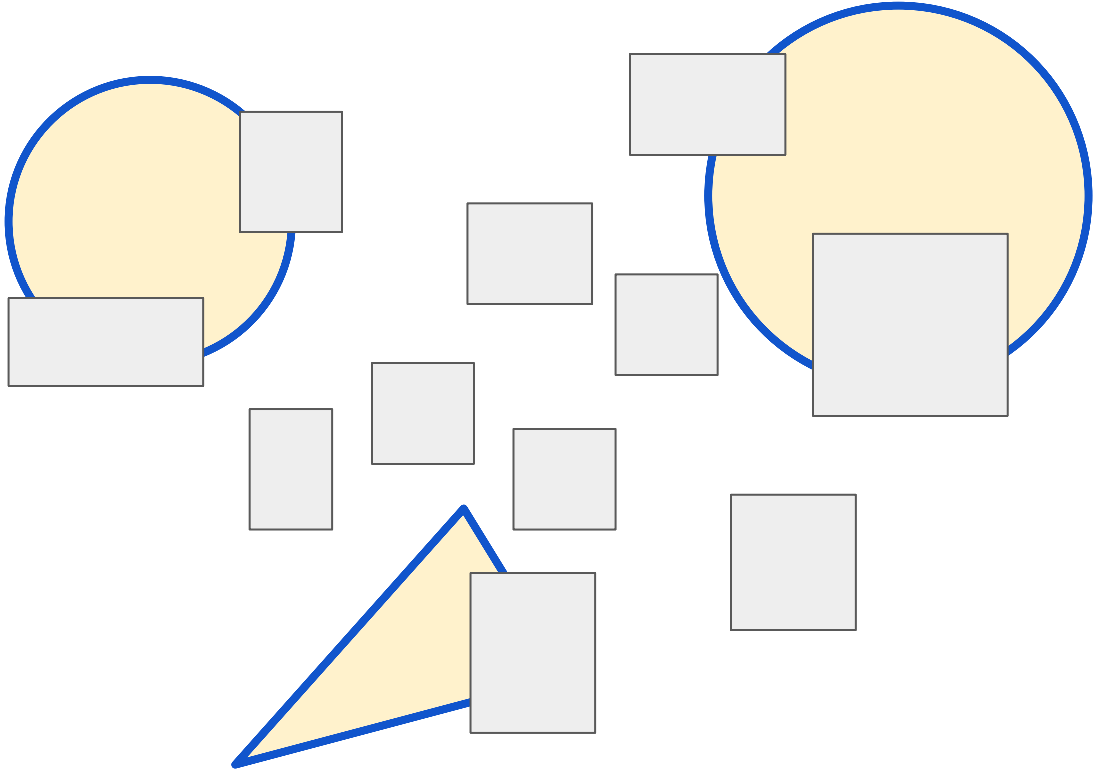
Culling



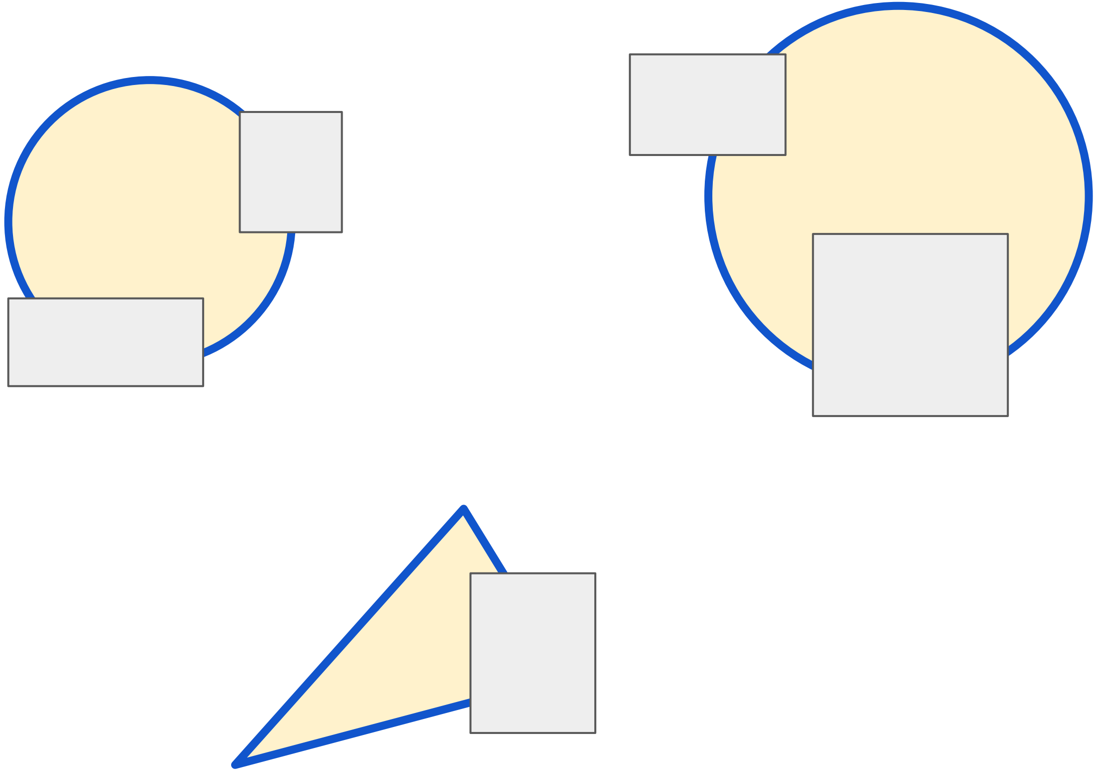
Culling



Culling



Culling



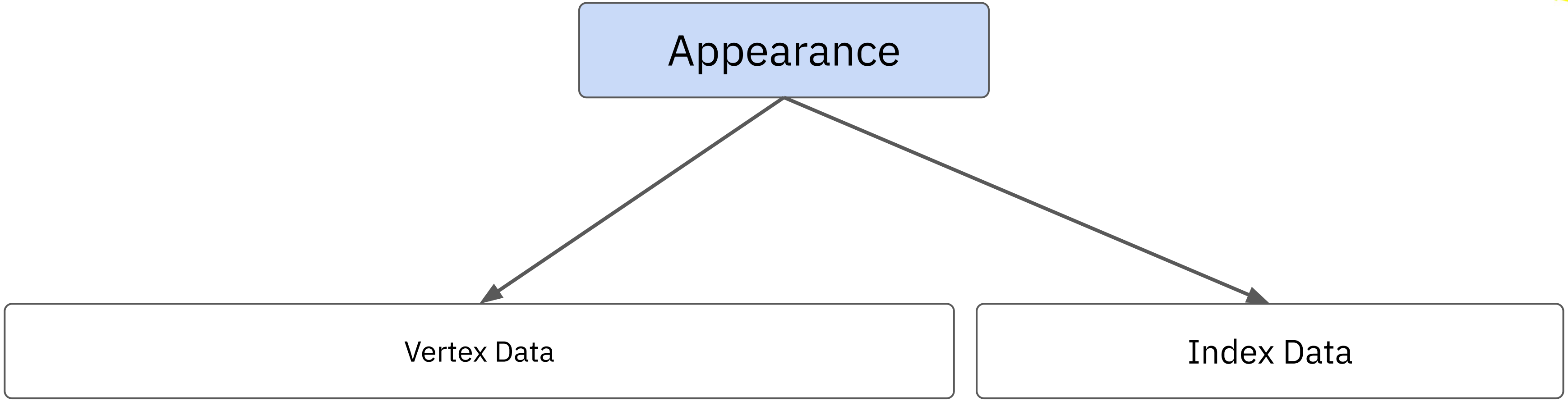
Improving Performance

- Savings depend heavily on scene construction, but...
- Reflections
  - Saved an average of 2000 objects from TLAS
  - Anywhere from 20-33% reduction
- Shadows
  - Saved anywhere from 14-20ms of CPU time (i9 9900KF)
  - Most saving in outdoor daytime scenes with no actual positional lights
    - Gracefully handles this content-specific scenario
  - Still significant in high density areas like cities

BVH Data

Appearance

BVH Data

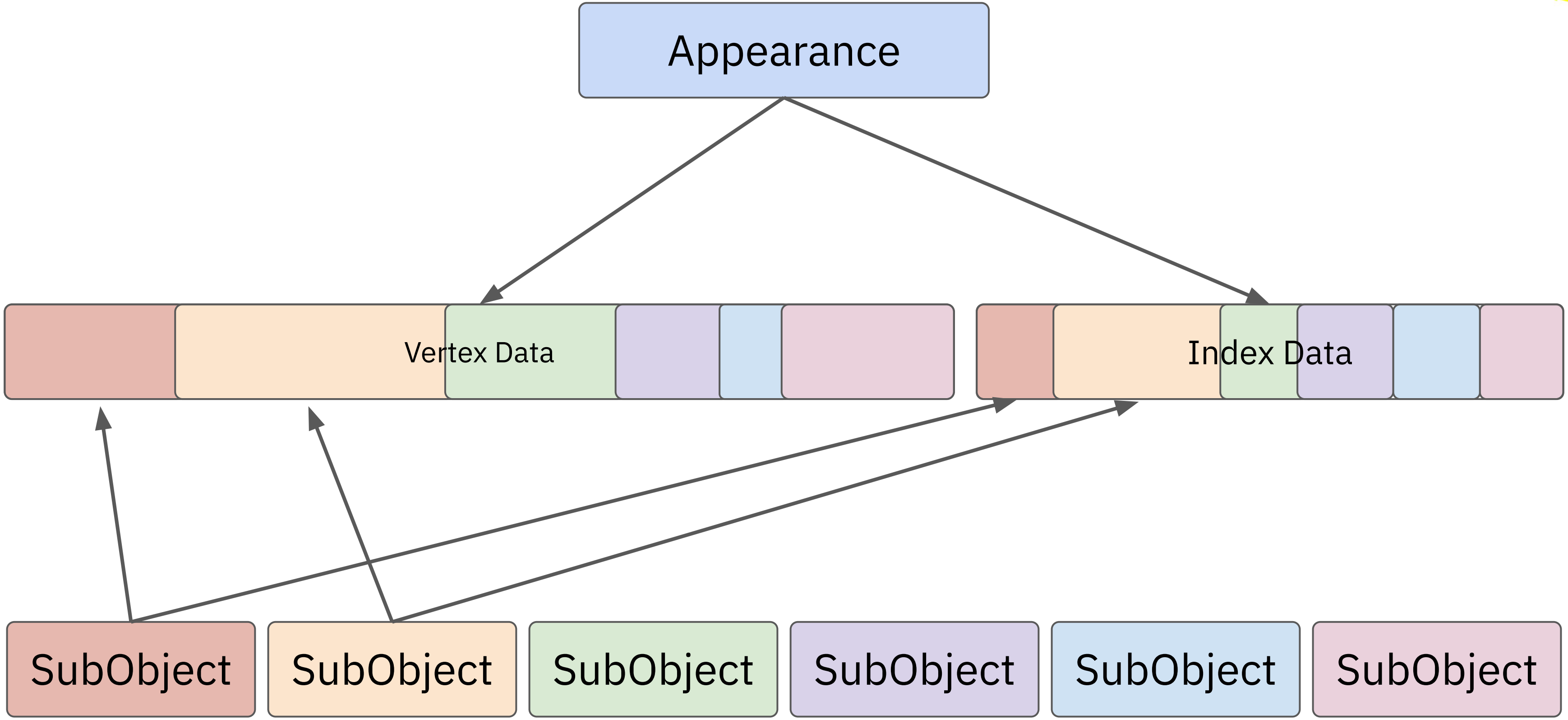


Appearance

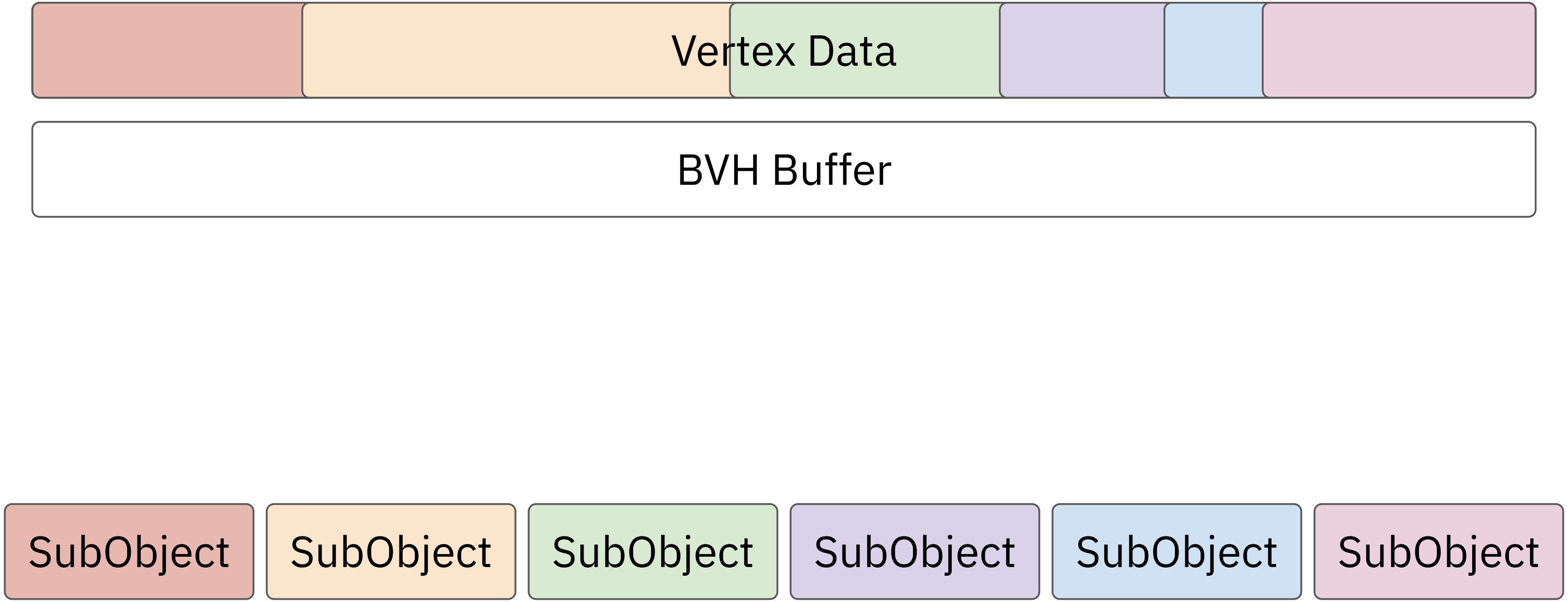
Vertex Data

Index Data

BVH Data


Appearance
Vertex Data
Index Data
SubObject
SubObject
SubObject
SubObject
SubObject
SubObject

BVH Data



Vertex Data

BVH Buffer

SubObject

SubObject

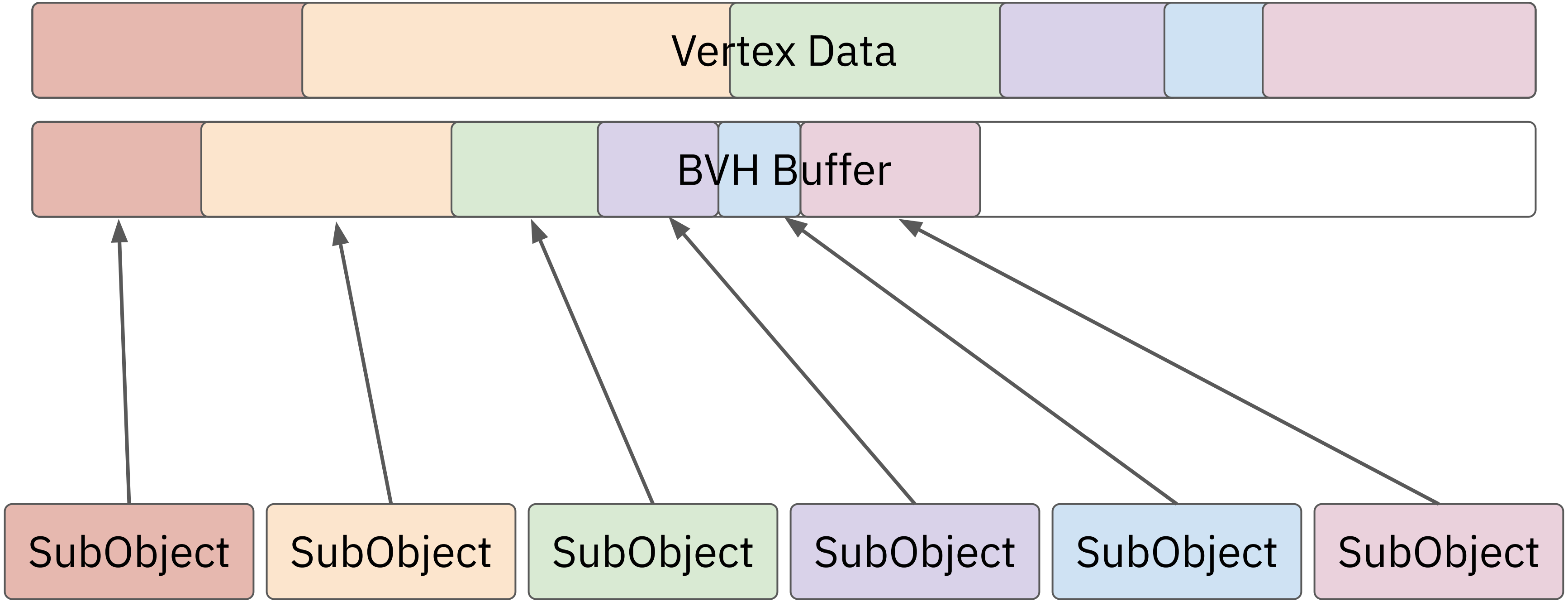
SubObject

SubObject

SubObject

SubObject

BVH Data


Vertex Data
BVH Buffer
SubObject
SubObject
SubObject
SubObject
SubObject
SubObject

BVH Data

- Data augmentation is a size and offset.
- The issue here is this data is immutable.

```
struct SubObject
{
    // Existing fields.
    ...

    // New RT fields.
    uint32 bvhOffset;    // Offset into larger BVH buffer.
    uint32 bvhSize;      // Cached size of the BVH.
};
```

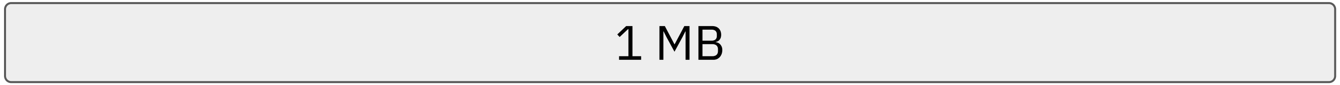
BVH Data

- Parent structure manages monolithic buffer
- Ad-hoc support for compaction
- Optimizing for memory gets difficult

```
struct BottomAcceleration
{
    Buffer* bvhBuffer;

    // New offset data after compaction.
    map<uint32, uint32>* compactionInfo;
    bool compacted;
    bool allowUpdate;
};
```

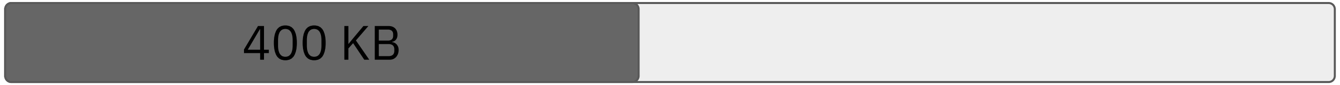
BVH Compaction Review

- Initially, size estimated and upper bound allocation made.



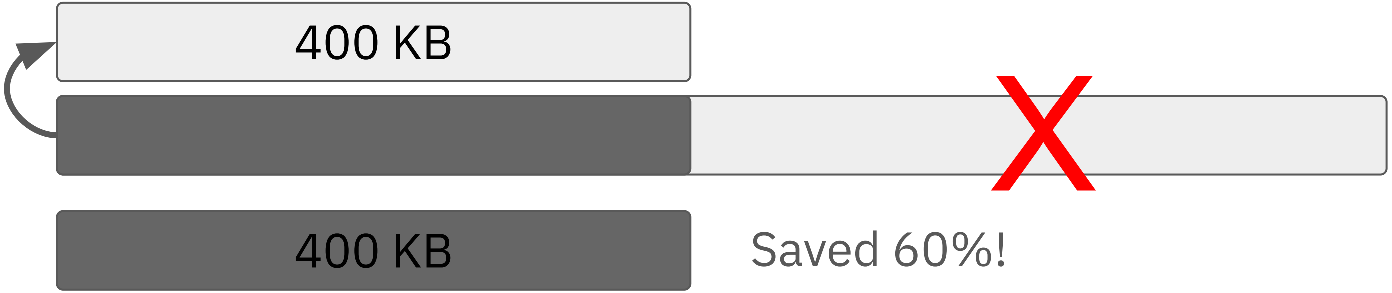
1 MB

- At build completion, query the real build size.



400 KB

- Create a new allocation and copy, discard the original.



400 KB

400 KB

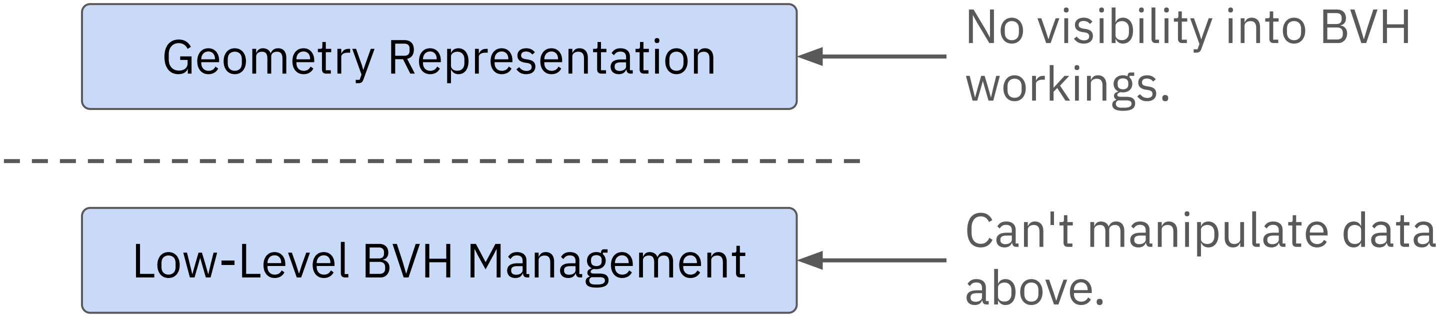
Saved 60%!

BVH Memory

- Initial implementation, all eligible assets included: 5+ GB BVH data
  - That probably won't work :)
  - Note: vast majority of this is SpeedTree, because instancing
- Compaction is our friend
- But it's not well-supported by the architecture
  - Not every class of asset can be compacted
- How so?

Data Tracking

- Recall: the BVH offset and size on a SubObject is immutable.
  - Immutability is already being violated when populated at load time.
- But, still used to point to BVH location in memory
- Time to refactor.



Geometry Representation

No visibility into BVH workings.

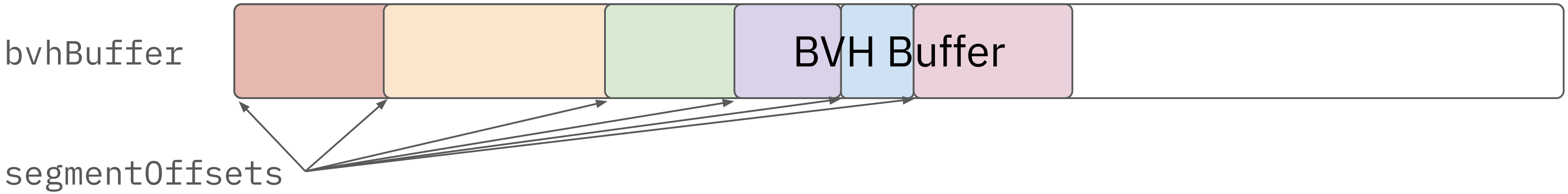
Low-Level BVH Management

Can't manipulate data above.

BVH Data

```
 struct SubObject
 {
     // Existing fields.
     ...

     // New RT fields.
-    uint32 bvhOffset;    // Offset into larger BVH buffer.
-    uint32 bvhSize;      // Cached size of the BVH.
+    uint32 segment;      // Index into array of sub-BVHs.
 };
```

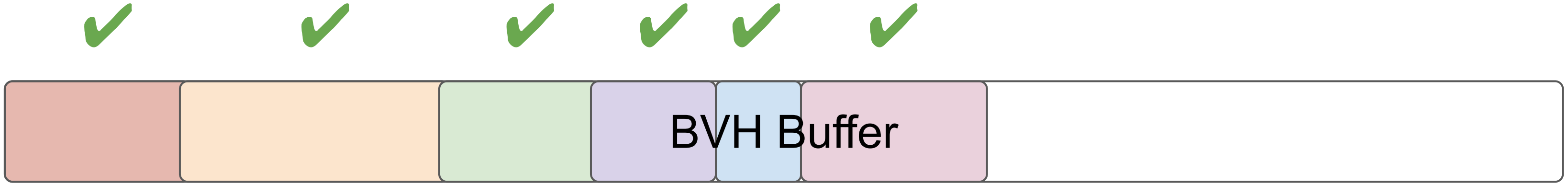
BVH Data



```
 struct BottomAcceleration
 {
     Buffer* bvhBuffer;

-    // New offset data after compaction.
-    map<uint32, uint32>* compactionInfo;
+    // Internal tracking of sub-BVHs.
+    uint32* segmentOffsets;
+    uint32 segmentCount;
     bool compacted;
     bool allowUpdate;
 };
```

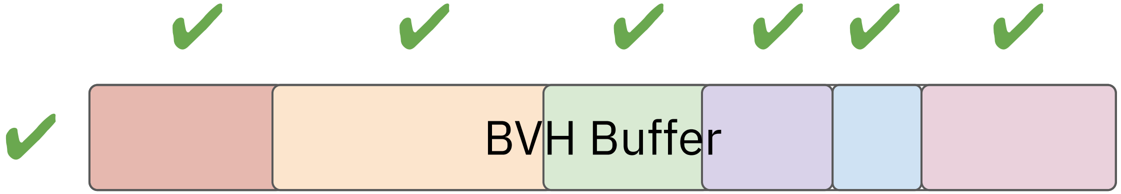
BVH Data

- Every BVH must be built and size queried to compact the larger buffer.



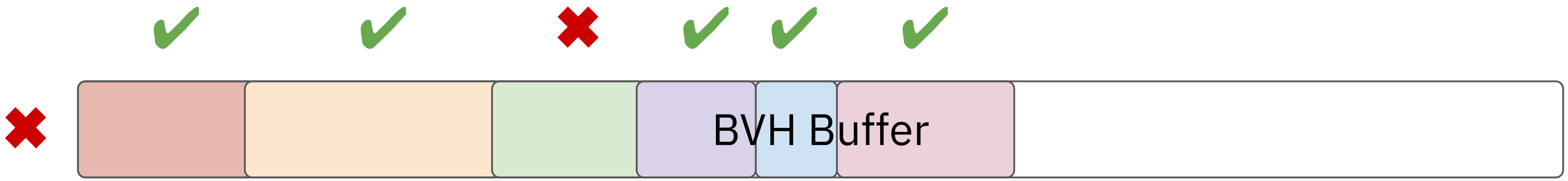
BVH Data

- Every BVH must be built and size queried to compact the larger buffer.



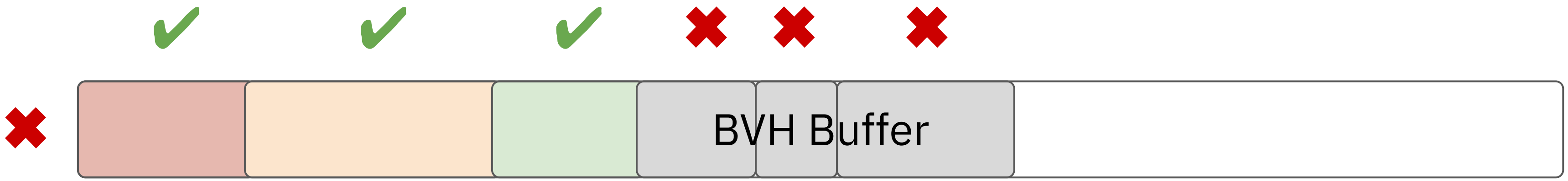
BVH Data

- Every BVH must be built and size queried to compact the larger buffer.
- If only one isn't, compaction can't happen.

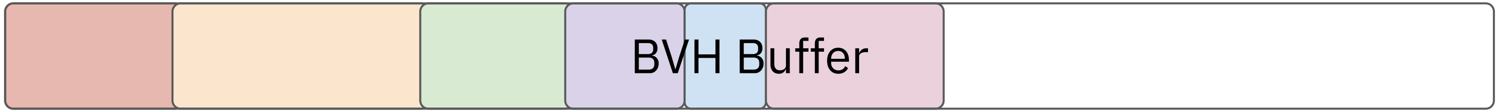


BVH Data

- Every BVH must be built and size queried to compact the larger buffer.
- If only one isn't, compaction can't happen.
- SubObject structure supports variable looks for assets
  - Not all SubObjects will be instantiated!



BVH Data



```
+struct BLAS
+{
+    Buffer* bvh;
+    bool compacted;
+};
```

```
 struct BottomAcceleration
 {
     // Internal tracking of sub-BVHs.
-    uint32* segmentOffsets;
+    BLAS* subBVHs;
     uint32 segmentCount;
-    bool compacted;
     bool allowUpdate;
 };
```

BVH Data

BVH
BVH
BVH
BVH
BVH

```
+struct BLAS
+{
+    Buffer* bvh;
+    bool compacted;
+};

 struct BottomAcceleration
 {
     // Internal tracking of sub-BVHs.
-    uint32* segmentOffsets;
+    BLAS* subBVHs;
     uint32 segmentCount;
-    bool compacted;
     bool allowUpdate;
 };
```

Platform Memory

- PC: buffers have min size of 64K
  - BVHs are typically much smaller than that
  - Pay the full price for each BVH created
- Bad for performance too
- Enter paging.

- NVIDIA RTX Memory Utility
  - https://github.com/NVIDIAGameWorks/RTXMU
  - Easy to integrate
  - Custom backend supports our low-level API abstraction layer

BVH Data

```
 struct BLAS
 {
-    Buffer* bvh;
+    rtxmu::SubAllocation bvh;
     bool compacted;
 };

 struct BottomAcceleration
 {
     // Internal tracking of sub-BVHs.
     uint32* segmentOffsets;
     BLAS* subBVHs;
     uint32 segmentCount;
     bool compacted;
     bool allowUpdate;
 };
```

BVH Memory

5.0+ GB □ 1.5+ GB

- Way better, but still a bit high.
- What other architectural components might be problematic?

Asset Streaming

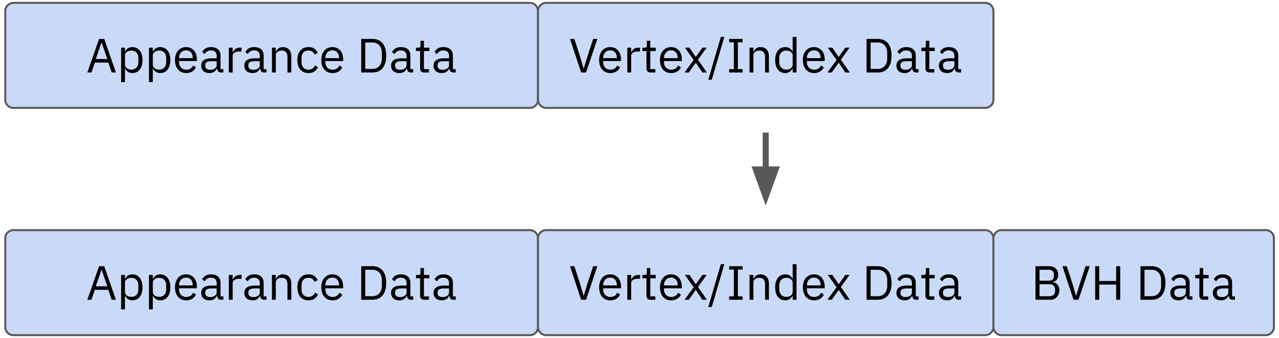
- Streaming distance in the game is large
  - Often larger than TLAS bounds
- Observation: animated objects only update BVH when added to a TLAS
- Solution: defer allocation from `Load()` to `Build()`
  - Deallocate when not used in a TLAS

1.5+ GB □ 250 MB

- More maturity of the design made this a much faster change

Consoles

- PS5 and Xbox both offer BVH builds offline
  - Better size *and* trace efficiency
- We build all static geometry BVH offline
  - Total of about 5GB data when compressed
  - Serialized as a data blob at end of Appearance data file
- Also does not use RTXMU, favors existing memory manager



Appearance Data | Vertex/Index Data

Appearance Data | Vertex/Index Data | BVH Data

Compaction of Dynamic BVH

- Misconception that this isn't possible
  - Update adjusts bounding box extents, while refit rebuilds hierarchy
- However, tradeoff with build quality
  - Quality drifts with each update
- Absorb degraded trace cost or compact more frequently?

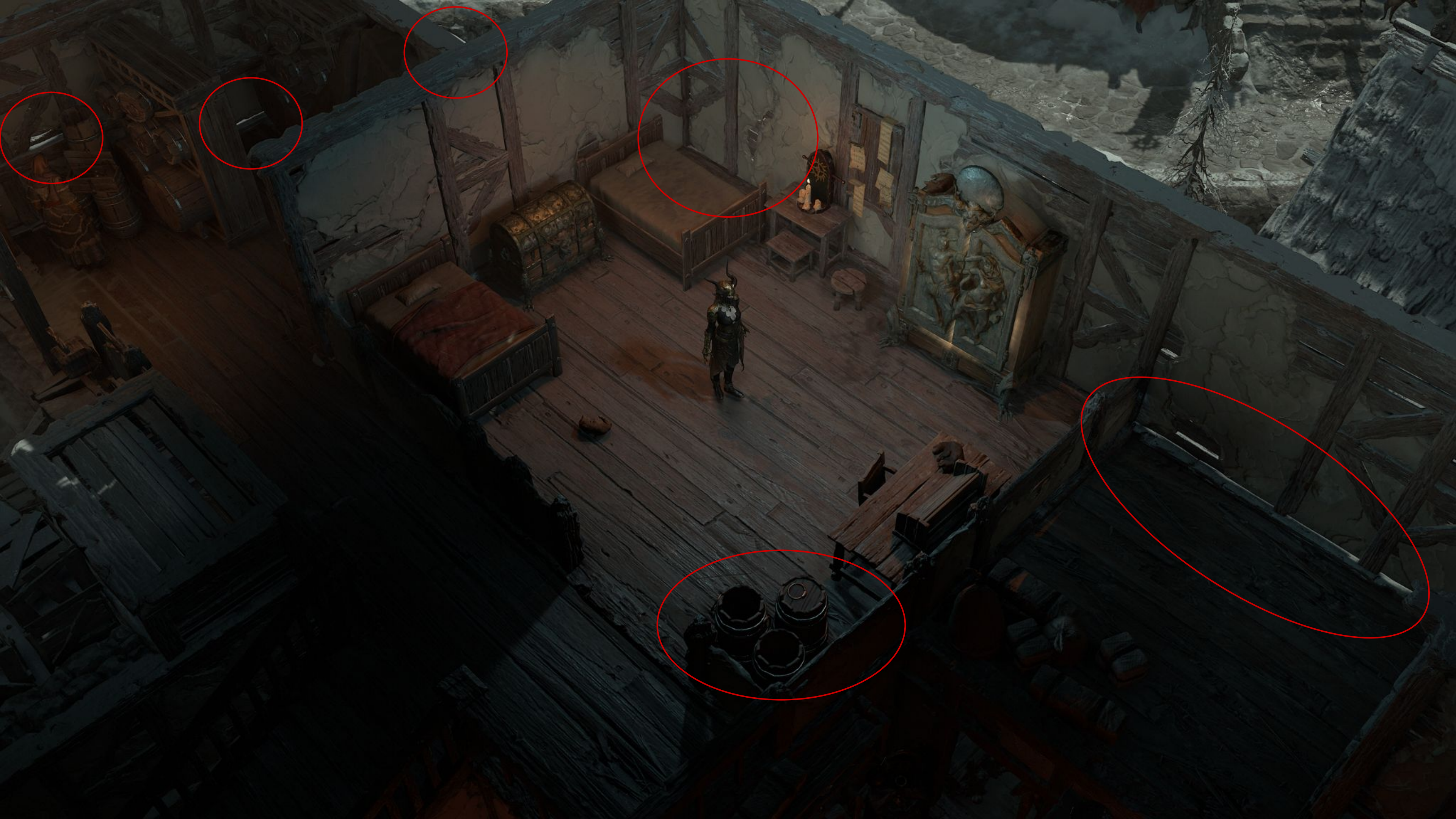
- Ultimately didn't ship dynamic compaction.
  - Good area for future work in our ray tracing implementation

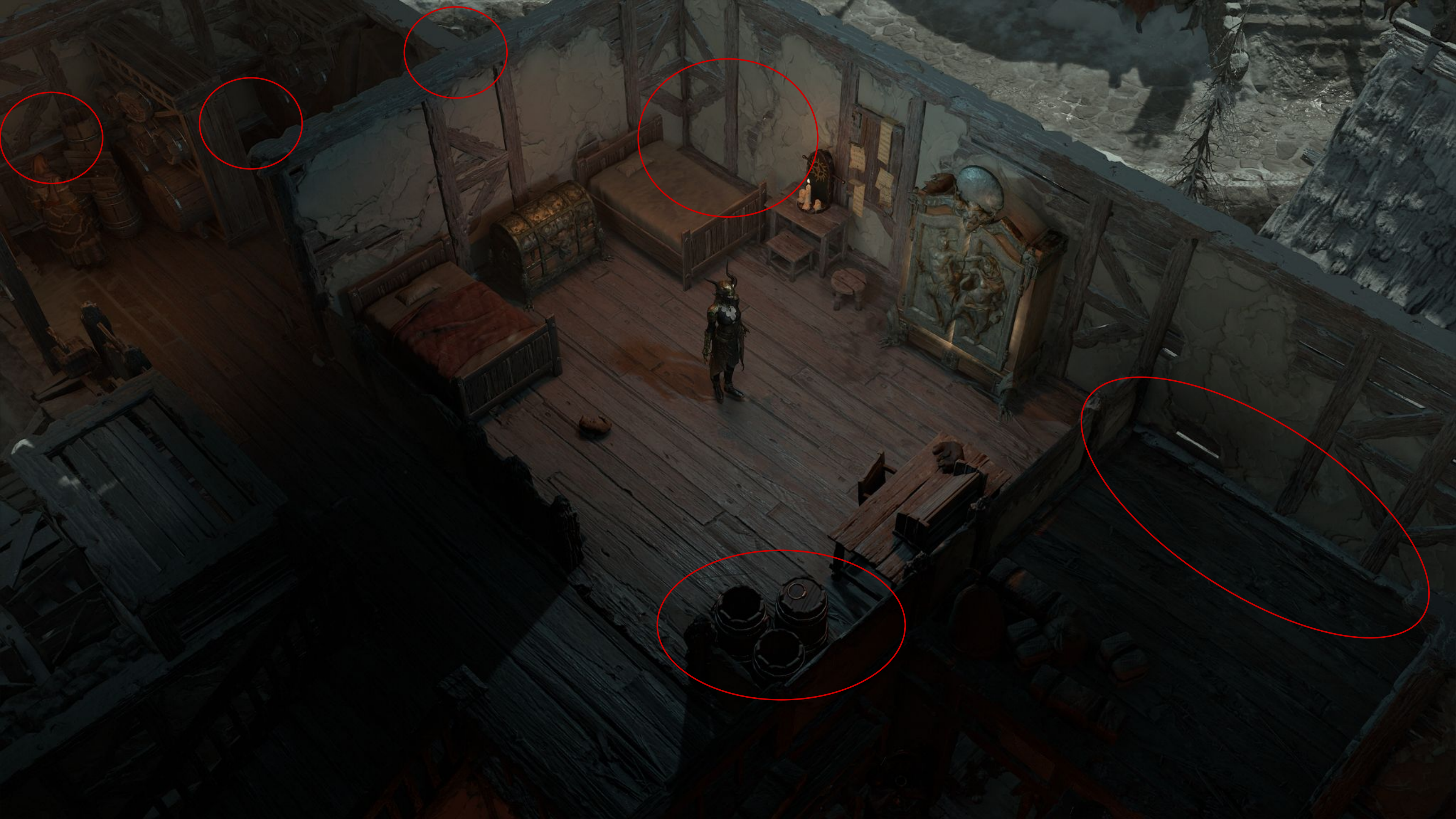
Implementation Recap

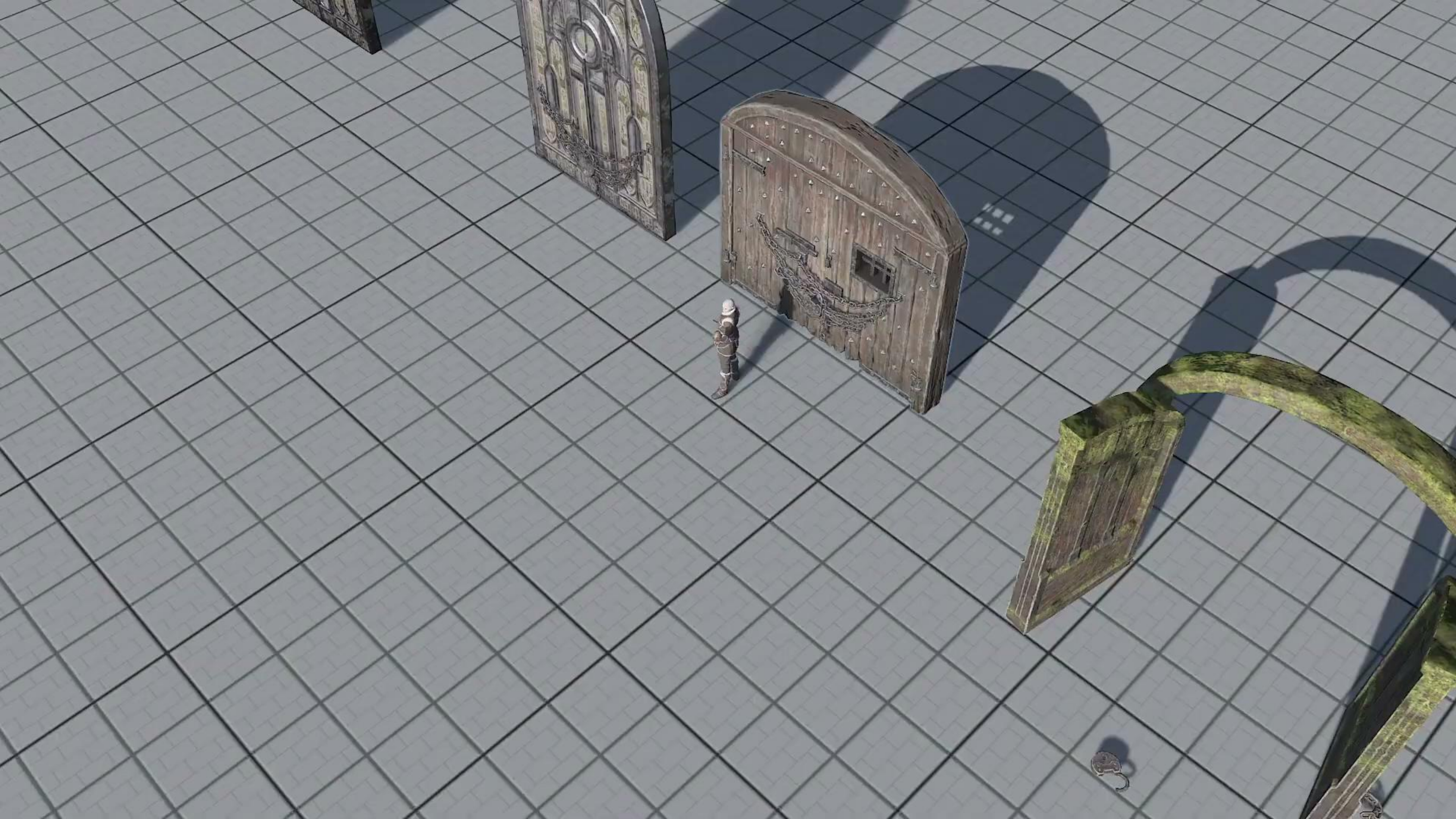
- Design of the engine dictates ray tracing technical design
  - But it may not be the most efficient
- Ray tracing paradigms inform new engine paradigms
- Design maturity made future changes faster and easier
- Pros of BVH architecture
  - Intuitive
  - Encapsulated
  - Memory efficient
  - Flexible

```
struct BLAS
{
    rtxmu::SubAllocation bvh;

    uint32 buildSize;
    uint32 lastUsedFrame;

    bool dynamic : 1;
    bool compacted : 1;
    bool offline : 1;
};
```

Preserving Content



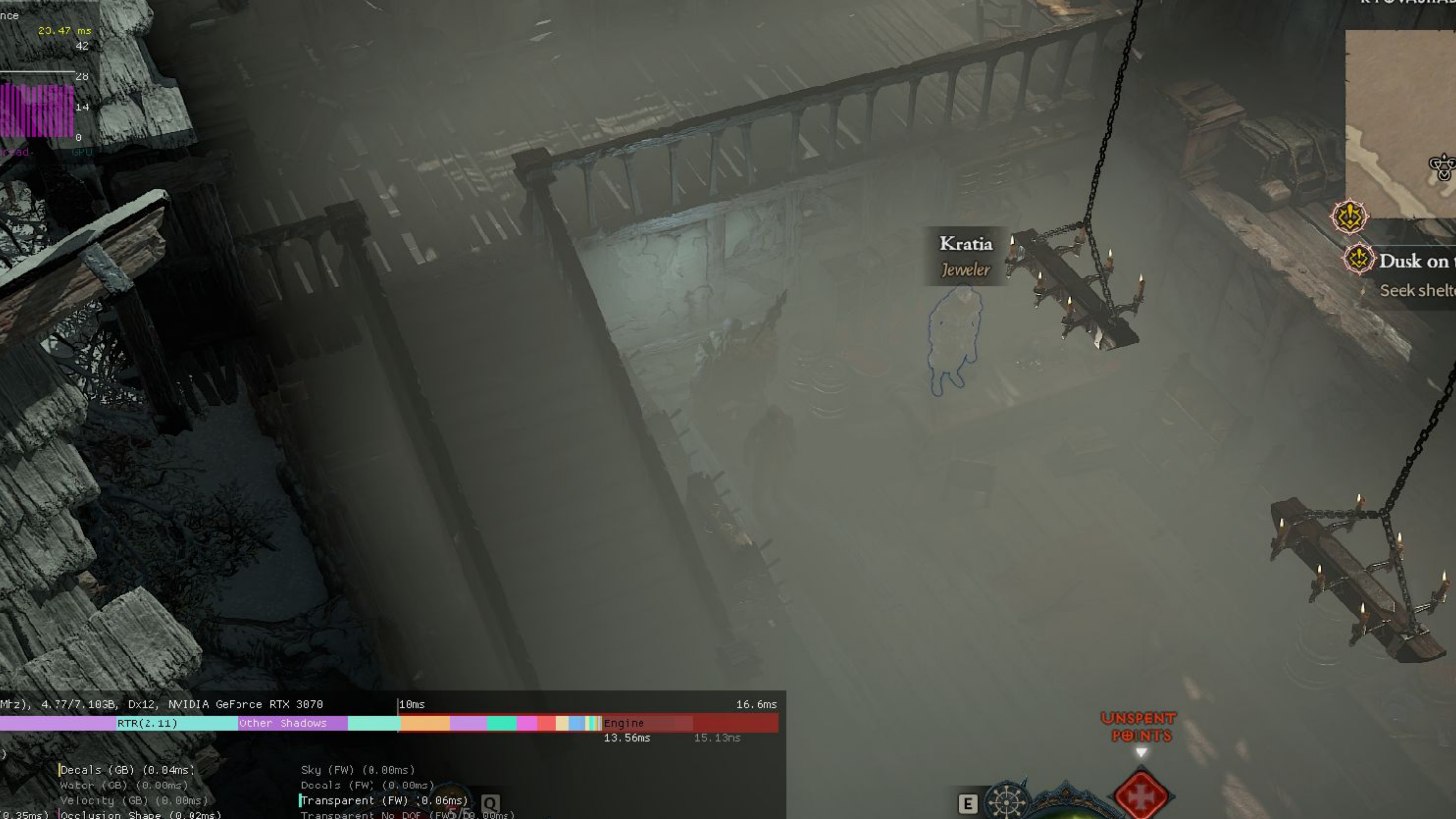




Renderable Types

	Directional Shadows	Local Shadows	Reflections
Opaque Objects	✔	✔	✔
Player	✔	✖	✔
SpeedTree	✔*	✔*	✔*
Particles	✖	✖	✔
Decals	✖	✖	✖
VAT	✖	✖	✖

* Only on PC



```
nce
        20.47 ms
              42
              28
              14
              0
read        GPU
```

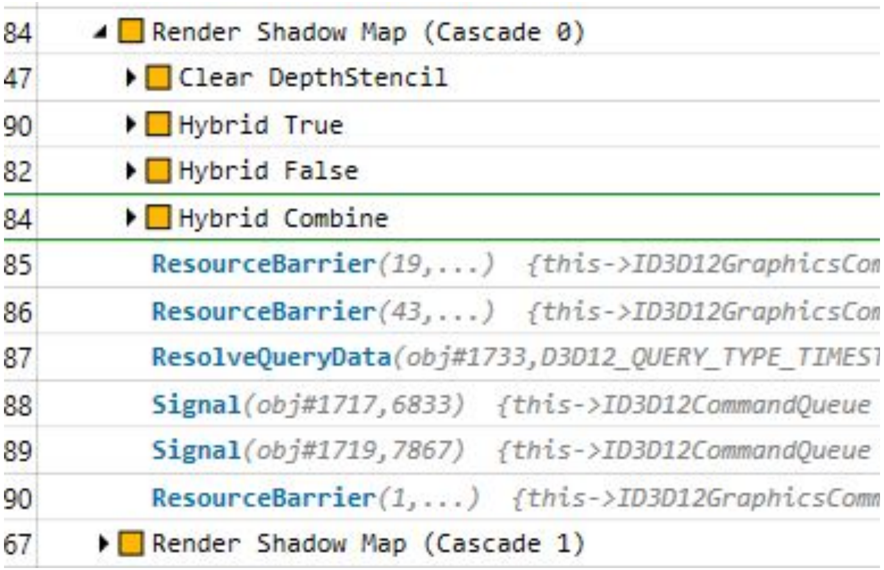
```
MHz), 4.77/7.10GB, Dx12, NVIDIA GeForce RTX 3070        10ms                       16.6ms
RTR(2.11)       Other Shadows                                   Engine
                                                                13.56ms     15.13ms
)
        Decals (GB) (0.04ms)            Sky (FW) (0.00ms)
        Water (GB) (0.00ms)             Decals (FW) (0.00ms)
        Velocity (GB) (0.00ms)          Transparent (FW) (0.06ms)
0.35ms) Occlusion Shape (0.02ms)        Transparent No DOF (FW) (0.00ms)
```

Hybrid Shadows

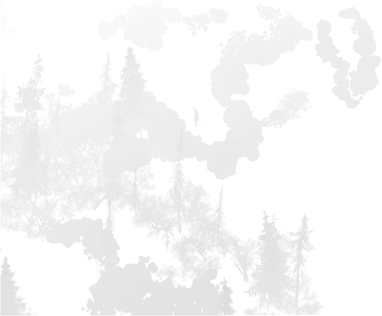
- Require three maps
  - Ray traced shadow map (done in screen space)
  - Raster shadow map of non-raytraced objects
  - Raster shadow map of all objects
- An object is considered hybrid if it can be raytraced.
- An object is considered non-hybrid if it can only be rastered.

- We're not going to consider cached shadow maps and static vs. dynamic objects.

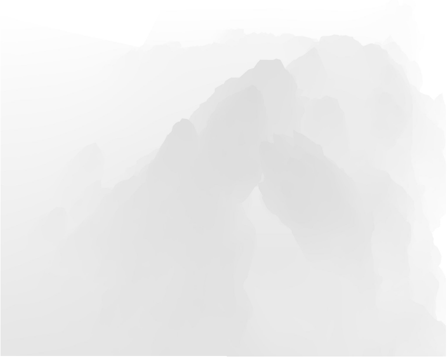
Hybrid Shadows



```
84  ◢ Render Shadow Map (Cascade 0)
47    ▶ Clear DepthStencil
90    ▶ Hybrid True
82    ▶ Hybrid False
84    ▶ Hybrid Combine
85      ResourceBarrier(19,...)  {this->ID3D12GraphicsCom
86      ResourceBarrier(43,...)  {this->ID3D12GraphicsCom
87      ResolveQueryData(obj#1733,D3D12_QUERY_TYPE_TIMEST
88      Signal(obj#1717,6833)  {this->ID3D12CommandQueue
89      Signal(obj#1719,7867)  {this->ID3D12CommandQueue
90      ResourceBarrier(1,...)  {this->ID3D12GraphicsComm
67  ▶ Render Shadow Map (Cascade 1)
```

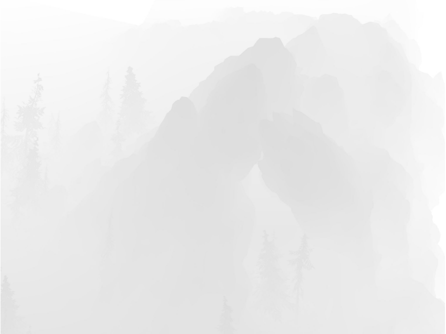


Hybrid



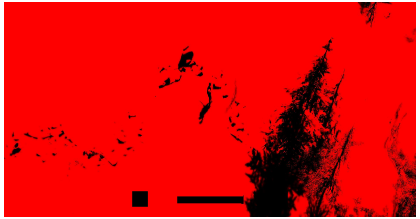
Non-hybrid

Combined



Hybrid Shadows

Ray traced light:

min( ,) and

Non-hybrid

Combined

Non-ray traced light or Volumetrics:

Combined

Summing Up

- Visual Impact
- Areas for Improvement
- Naive design is costly
    - But would that really change if designed for ray tracing?
- Conceptual challenges permeate down to technology
- Foundational work is costly

Thank You

- Keven Cantin
- Michael Bukowski
- John Buckley
- Samuel Delmont
- Jon Lee
- Zach Vinless
- Alexander Demyanenko
- Justin Williams
- Fernando Urquijo
- Ben Hutchings
- Gustavo Samour Lopez
- Charles Zhang
- Zach Schecter
- Joel Peters
- Kevin Bell
- Chad Layton
- Lorenzo Di Spina
- Alex Mueller
- REAC Organizers

Questions?

Questions fielded by Keven Cantin - Thanks Keven!

@kevintodisco