Light Path Guided Culling for Hybrid Real-Time Path Tracing

Jan Kelling Daniel Ströter Arjan Kuijper

Motivation

- Photorealistic rendering requires accurate lighting
- Difficult with rasterization
- Path Tracing to the rescue!

[Sponza]

To produce photorealistic images (like the one on the right), we require accurate (indirect) lighting computation.
Generic indirect lighting solutions are pretty much impossible with just rasterization.
Path tracing can solve the rendering equation exactly => produce photorealistic results.
Path tracing is very expensive but with GPU-accelerated ray tracing and denoising and sampling advancements it is becoming a viable option for real-time.

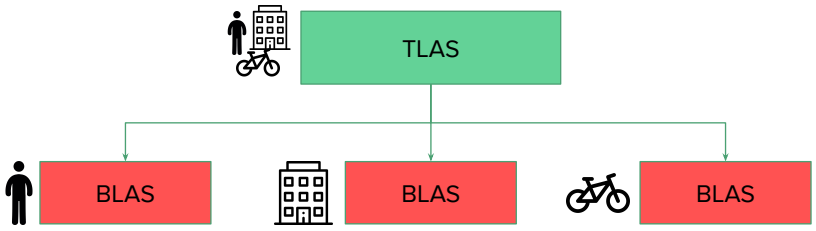
Otherwise, real-time ray tracing technique are a cheaper alternative for real-time.
They can solve at least parts of the rendering equations (capture specific light paths), for instance just diffuse global illumination.
Our novel culling method does apply for other real-time ray tracing techniques as well.

Denoising advancements: [Schied17]
GPU advancements: [Wyman18]

Acceleration structures

GPUs: Two-level acceleration structure scheme

- Bottom level acceleration structure (BLAS): single mesh
- Top level acceleration structure (TLAS): entire scene



To find ray intersections quickly, GPUs need acceleration structures.
GPUs use two-level acceleration structures:
Bottom level acceleration structures (BLAS) encapsulate single meshes.
Top level acceleration structures (TLAS) reference bottom level acceleration structures and contain the entire scene used for ray tracing.

BLASes are self-contained, i.e. they contain all vertex data. That means they can get quite large.

Real-time Ray Tracing In Large Scenes

- Memory consumption of acceleration structures
- Instances with unique vertex transform require own BLAS
  - Animations!
- Build time of animated BLASes
- TLAS build time

[Anno]

While path tracing works well in research, applying it to bigger scenes can produce problems. City scene from the game Anno.

Acceleration structures can grow quite large in graphics memory.
TLAS gets large for many instances.
BLASes also a problem:
Instanced rendering (rasterization) allows to render huge city scenes with many units.
A building mesh can be rendered for a hundred instances, all sharing the same index/vertex buffer.
BLASes can be shared as well, However, for uniquely transformed (animated or terrain adapted) meshes, this is not possibly anymore for raytracing.
When mesh is uniquely transformed (think of vertex transformations in the vertex/mesh shader), BLASes cannot be shared between instances.
-> Huge memory overhead, BLAS per instance.

For running animations, we have to refit/rebuild their BLASes in each frame. That's not a cheap operation either way.

A high number of instances (more than a hundred thousands) will result in large TLAS memory consumption and high TLAS build timings.

How to render large scenes?

What do we usually do in computer graphics when we want to render a large scene?
Culling!

Culling for Ray Tracing

- Frustum/Occlusion culling does not work for ray tracing
- State of the art [DS19]: heuristic based on distance/size
  - Iteration over all instances on CPU?
- Want to cull objects that do not influence lighting
  - Difficult to predict

For instance, frustum/occlusion culling is widely used.
But do not work well for ray tracing.
Even objects far outside the frustum and occluded on screen might be important for the lighting.
Think of surfaces behind the camera throwing indirect light onto the scene. Or objects visible in mirror!

What games have been doing for the last couple of years: Culling heuristic based on (objectSize / distance(object, cam))
That usually requires iteration over all instances on the CPU.
And is not optimal: either a lot of unnecessary objects included in acceleration structure or objects further away already culled.

Ideally, only include objects contributing to lighting. But that is hard to predict.

Idea: let’s count during tracing how often objects are encountered by rays (light paths)
And cull objects that are not encountered often.

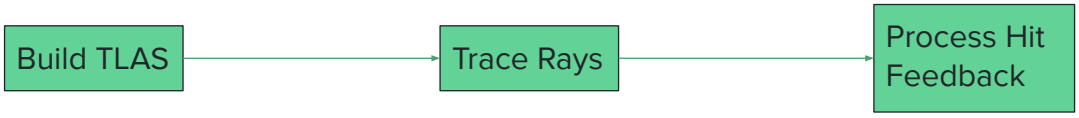
Hit feedback

- Previously proposed for animation prioritization [Mak23]
- Each instance gets an activation state
- Problem: How to know a culled object is needed again?
  - Replace with static bounding box in TLAS (**Hitbox**)
  - To reduce number of instances: group into bigger boxes

We call this hit feedback, similar to sampler feedback.
Utilizing hit feedback was proposed before, but just for prioritizing animated BLAS rebuilds.
We built a generic culling strategy on top of it for all instances.
For that, each model instance gets a state and based on the number of enounctered light paths. State decides whether to cull it in future frames.

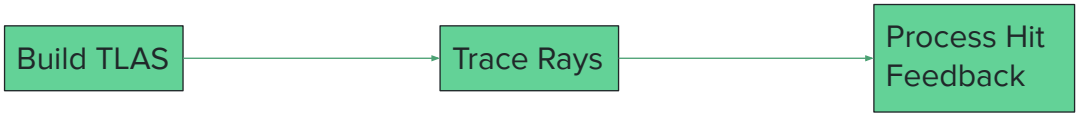
But when an object is culled, it won't be encountered by light paths again.
We replace it with a static bounding box in the TLAS that still counts the number of light paths and is ignored otherwise.
We do not need to build (or maintain/animate) the BLASes for culled instances.
Culled instances are grouped together to also reduce the number of instances in the TLAS, effectively culling entire regions of the scene.

Light Path Guided Culling (LiPaC)



Three important parts of every frame for our culling pipeline. This all happens on the GPU.

Light Path Guided Culling (LiPaC)

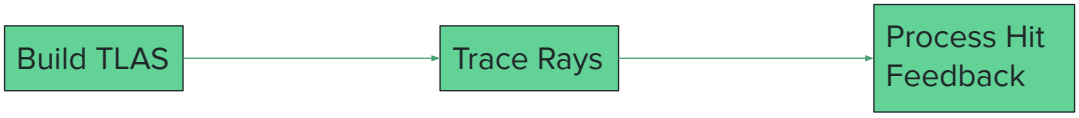


On object/hitbox hit, increase respective counter.

Hitbox intersection: Just continue the ray

When an object or hitbox gets hit (in closest hit shader), we atomically increase the respective counter in the GPU hit feedback buffer.
When a ray encounters a hitbox, we afterwards just continue tracing the ray, effectively ignoring the hitbox for all lighting calculations.

Light Path Guided Culling (LiPaC)



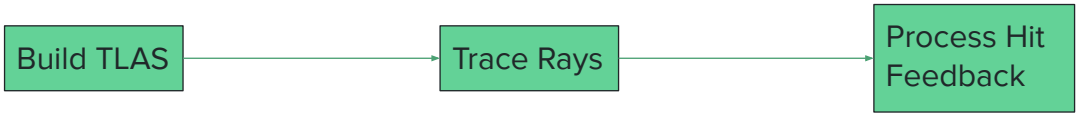
On object/hitbox hit, increase respective counter.

Hitbox intersection: Just continue the ray

Update instance/hitbox culling states based on encountered light paths

Afterwards, we iterate over all instances and hitboxes and re-evaluate whether they should be active.

Light Path Guided Culling (LiPaC)



Add active models and hitboxes to the TLAS instance buffer

On object/hitbox hit, increase respective counter.

Hitbox intersection:
Just continue the ray

Update instance/hitbox culling states based on encountered light paths

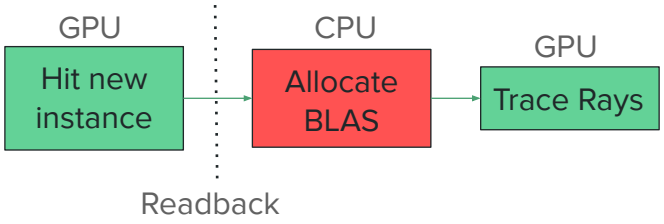
And then only in the next frame, we build the TLAS from the active instances.
We do so by iterating over the instances on GPU side and assembling the TLAS instance buffer.
No iteration over instances on CPU side is needed! GPU-driven BLAS management

Let’s look at the city scene, freeze and visualize the acceleration structure.
We were looking into a mirror all along! :)

And, as desired, objects visible in the mirror are not culled. While objects occluded by the mirror are culled.

Latency

- GPU-driven acceleration structure management
- On instance activation: BLAS build needed
  - Roundtrip to CPU via command readback buffer
- Latency of multiple frames

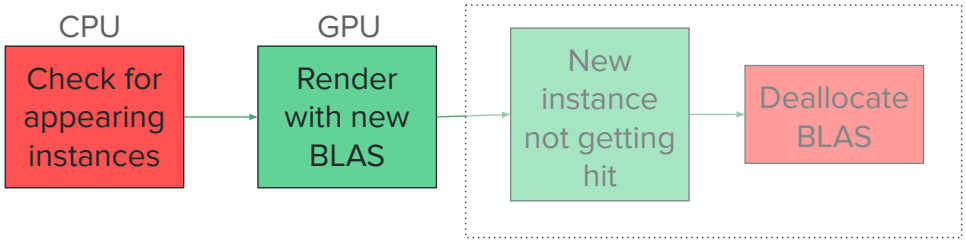


BLAS allocation and build recording needs CPU.
Latency is problematic: when a new instance appears on screen, it will only be considered by ray tracing after a couple of frames.
Nasty artefacts like shadow and occlusion pop-in.

Latency

- Mitigation: activate appearing instances each frame
  - Will get deactivated by LiPaC again if not important
  - Latency still visible through mirror

In practice: not noticeable with mitigation in place



We propose mitigation: check in each frame on CPU (before building the TLAS) whether instances have newly appeared in the frustum

E.g. because of camera move or newly created models

We still use a heuristic here, only do this for large, “important” objects

Instances activated by this that are not encountered by light paths are then quickly de-activated again.

There are cases not caught by this, but it gets rid of the most noticeable problems.

Hybrid Path Tracing

- Path Tracing: can we really cull any visible objects?
- Combine with rasterization!
  - Primary hits: culled instances still visible. +Performance!
  - Rasterize culled instances into shadow map
  - Guarantee baseline quality for culled instances
- Allows aggressive culling
- “80% of the visual quality for 20% of the overhead”

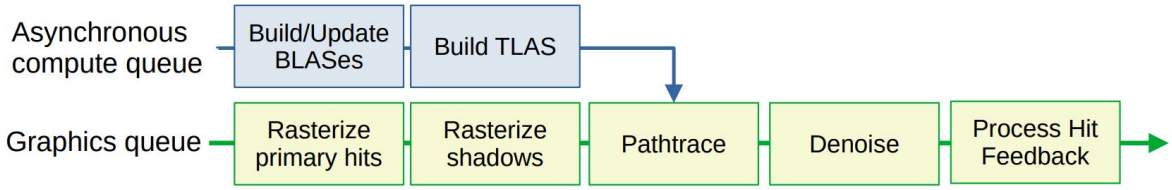
But when we consider path tracing, we have to ask: can we even cull objects at all? As long as they are encountered by just a single light path, we will get noticeable artefacts, objects/shadows popping in etc

This culling strategy is especially interesting for hybrid techniques.
Rasterizing primary hits does not change visual quality. But improves performance and causes instances culled from ray tracing to still be visible.
By rasterizing culled instances into shadow maps, we guarantee that shadows are not completely popping in.
And we guarantee to never fall below a visual baseline (of a renderer using just rasterization techniques)

Then you can only activate the “most important” instances for lighting of the scene without major artefacts.

Hybrid Path Tracing

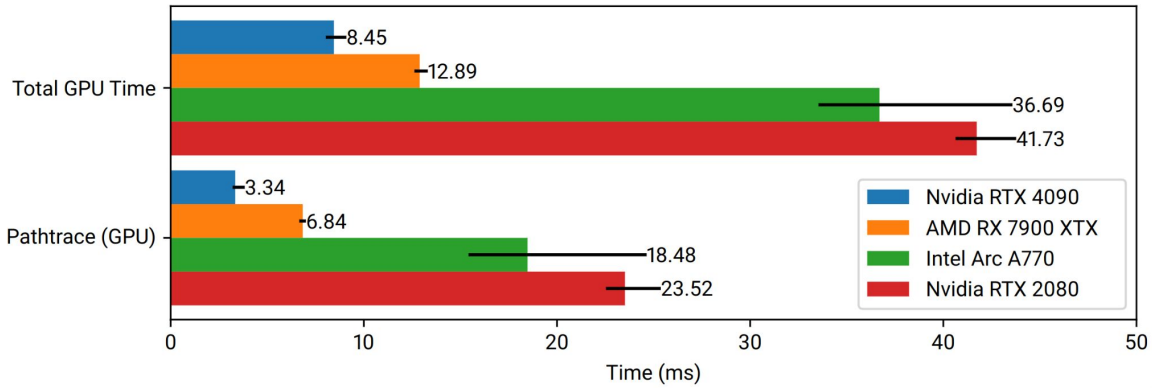
Optimized pipeline: overlap acceleration structure building with rasterization via asynchronous compute



Can use GPU capabilities of rasterization and async compute to the max: interleave acceleration structure building with tasks that mainly require rasterization.

Evaluation

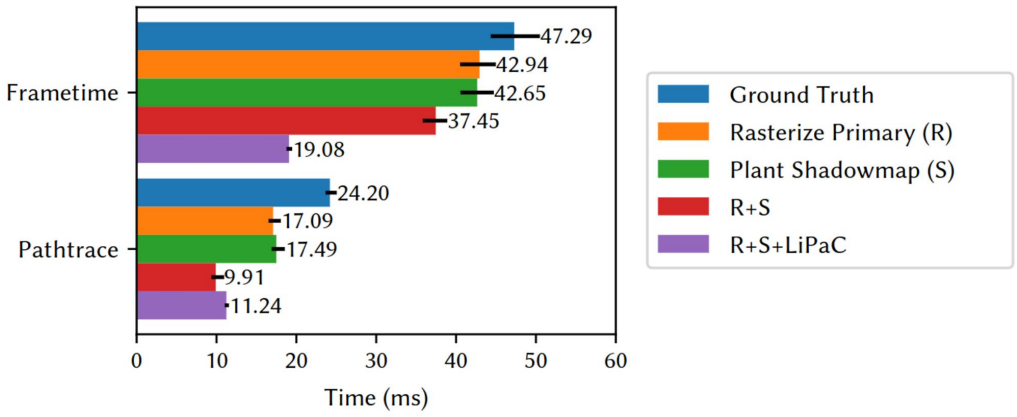
Real-Time Path Tracing in the city scene with 10’000 animated units (1080p)



Path tracing is still challenging. Only possible in real-time on the most powerful current-gen GPUs.
But other GPUs can fall back to other real-time ray tracing techniques (single bounce diffuse GI or reflections).
Our culling method does apply for other ray tracing techniques as well.

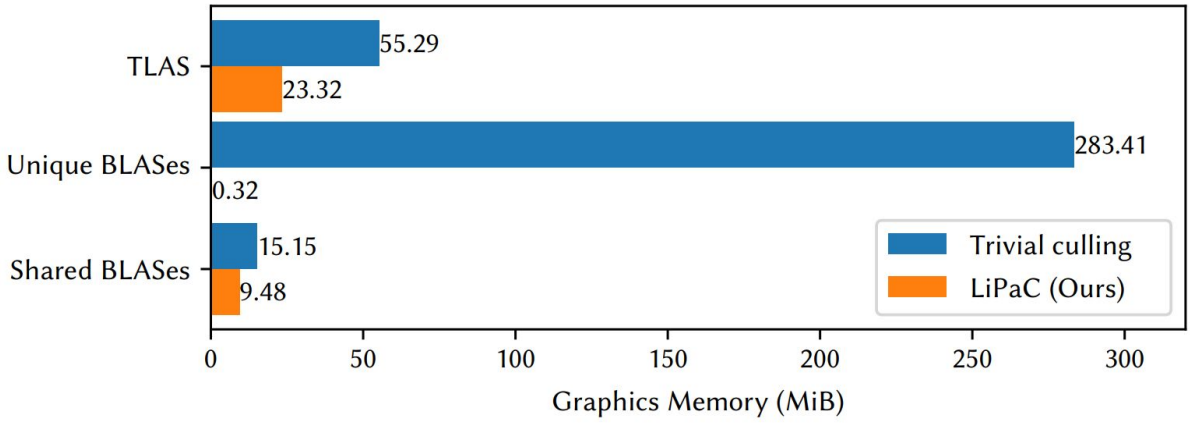
Evaluation

Impact of the various optimizations in our method (AMD RX 7900 XTX, 1080p)



Evaluation

Memory consumption reduction by LiPaC (AMD RX 7900 XTX)



Conclusion

- Hybrid path tracing promising for real-time rendering
- Culling is important for real-time ray tracing methods
  - Possible to achieve significant speedups
- Light path feedback is valuable
  - Consider ray type/differentials?
  - Use it to control mesh level of details?

References

[Schied17]: Spatiotemporal Variance-Guided Filtering: Real-Time Reconstruction for Path-Traced Global Illumination. In Proceedings of High Performance Graphics (Los Angeles, California) (HPG '17). Association for Computing Machinery, New York, NY, USA, Article 2, 12 pages. https://doi.org/10.1145/3105762.3105770

[Wym18]: Chris Wyman, Shawn Hargreaves, Peter Shirley, and Colin Barré-Brisebois. 2018. Introduction to DirectX Raytracing. In ACM SIGGRAPH 2018 Courses (Vancouver, British Columbia, Canada) (SIGGRAPH '18). Association for Computing Machinery, New York, NY, USA, Article 9, 1 pages. https://doi.org/10.1145/3214834.3231814

[Mak23]: Evgeny Makarov. 2023. Practical Tips for Optimizing Ray Tracing. https://developer.nvidia.com/blog/practical-tips-for-optimizing-ray-tracing/

[DS19]: Johannes Deligiannis and Jan Schmid. 2019. "It Just Works": Ray-Traced Reflections in 'Battlefield V. https://www.gdcvault.com/play/1026282/It-Just-Works-Ray-Traced

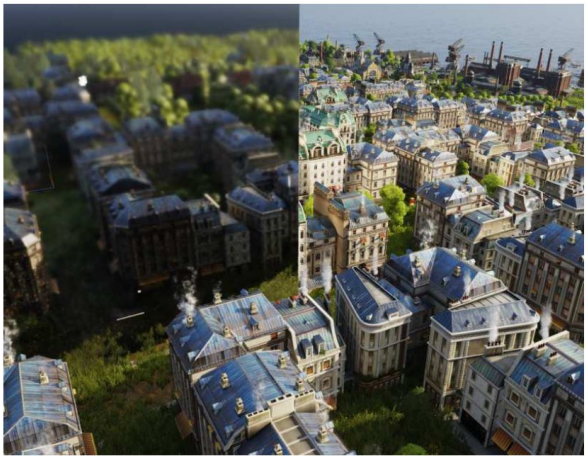
[Sponza]: Intel. GPU Research Samples Library. https://www.intel.com/content/www/us/en/developer/topic-technology/graphics-research/samples.html

[Anno]: Ubisoft Mainz. Anno 1800. https://www.ubisoft.com/en-us/game/anno/1800

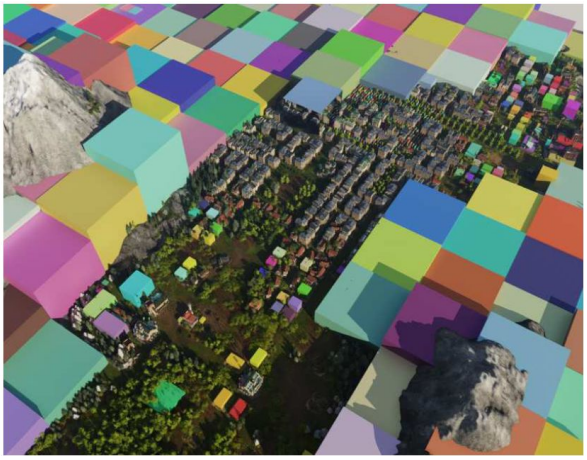
Icons on slide 3 by https://www.flaticon.com/authors/freepik, Flaticon License

Light Path Guided Culling for Hybrid Real-Time Path Tracing

Path traced view



TLAS visualization



Jan Kelling • headscratch.net • jan.kelling@ubisoft.com