LUND UNIVERSITY | LTH FACULTY OF ENGINEERING

Rendering Small Things: Hardware Micromaps and Particles

Gustaf Waldemarson
Arm & Lund University



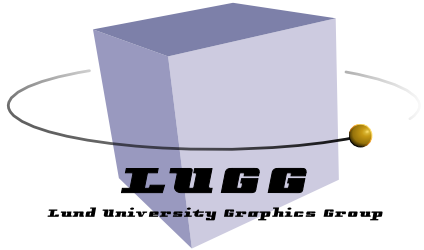
arm



Thank you very much, Jeppe for introducing everyone here to the field of computer graphics, now, after the mandatory struggle to connect my machine to the projectors I will present the contributions I have made to this field.

Who am I?

- Industrial PhD Student at the Lund University Graphics Group
- Software Engineer at Arm





https://gustafwaldemarson.com

Who am I?
- Industrial PhD Student at the Lund University Graphics Group
- Software Engineer at Arm
arm
https://gustafwaldemarson.com

As you all know, I'm Gustaf Waldemarson, the aforementioned PhD student under scrutiny.

But, It is worth mentioning that I am also a software engineer at Arm here in Lund.

Now though, today we will not be talking about my work at Arm, but rather my work done for this thesis, which I have conveniently summarized in the overview of the contributions….

Rendering Small Things

Contributions Overview

Graphics Tooling:

BCON24 PBRT: Create your own Importers and Exporters

BCON23 Handling Custom Data in `glTF` Files with Exporter/Importer Plugins

Included Papers:

I Superluminal Photon Mapping

II Parallel Axis Split Tasks for Bounding Volume Construction with OpenMP

III Succinct Opacity Micromaps

IV Color and Attribute Micromaps

Rendering Small Things
Contributions Overview
Graphics Tooling:
BCON24 PBRT: Create your own Importers and Exporters
BCON23 Handling Custom Data in glTF Files with Exporter/Importer Plugins
Included Papers:
I Superluminal Photon Mapping
II Parallel Axis Split Tasks for Bounding Volume Construction with OpenMP
III Succinct Opacity Micromaps
IV Color and Attribute Micromaps

First, I want to highlight my contributions to the tooling that support the graphics field at large when manipulating scenes, or generating and storing rendering data, which I also happen to have presented at two important conferences.

Then, as you can see on title of this thesis, we will discuss my contributions to how various *small* things are handled during the rendering process. As an example:

- In the Superluminal Photon Mapping paper, we use aspects about the atomic particles to render something akin to Cherenkov radiation.
- Then, in our Bounding Volume Construction work, we investigate another small but important detail: Namely how to build the data-structures necessary for rendering as quickly as possible with readily available tools.
- After that, we will combine both of these topics by rendering small things *inside* the acceleration structure with something known as *opacity micromaps*.
- And in the final work, we further extend that structure, generalizing it with colors, or even *arbitrary* attributes.

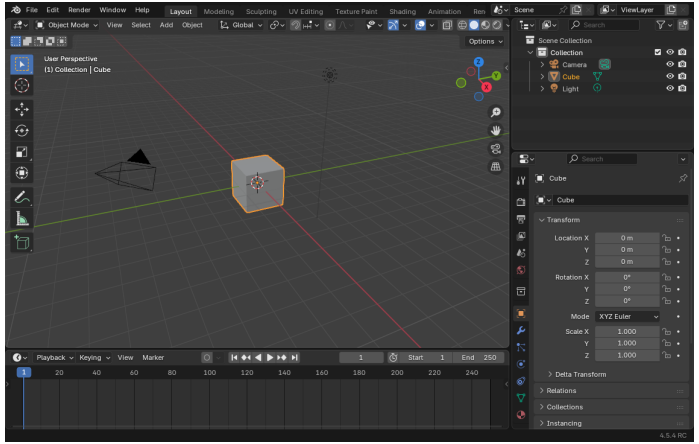
Graphics Tooling:
Blender, Test Scenes, and Rendering Data



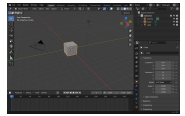
Graphics Tooling:
Blender, Test Scenes, and Rendering Data



Test Scenes and Rendering Data



Test Scenes and Rendering Data



In this field, we handle a lot of 3D assets and a substantial amount of our time is spent wrestling with them to turn them into something usable.

To that end, we often use very specialized tools, and one particularly useful one, is *Blender*. A very complete framework for building and rendering 3D scenes. However, even with a good tool like this, as a researcher you often need to customize the scenes beyond what the tool natively supports, for instance, to add the specific rendering data that you need for your specific application.

This is something I did extensively in at least two papers, and, something we thought could be useful to share with others. Thus, I summarized some of these findings in presentations that I shared at the official Blender Conference in Amsterdam in 2023 and 2024, as well as a pair of specialized Blender sub-tools to perform these operations.

BCON24
PBRT: Create your own Importers and Exporters

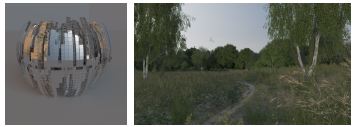


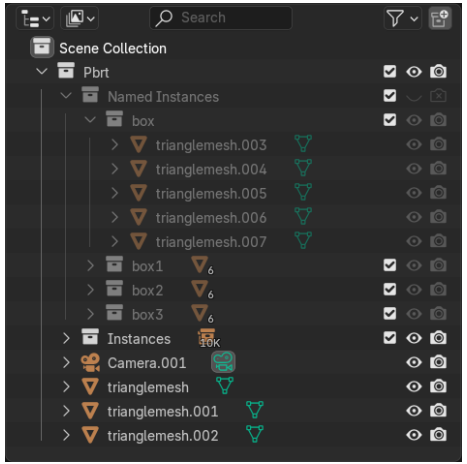
BCON24
PBRT: Create your own
Importers and Exporters

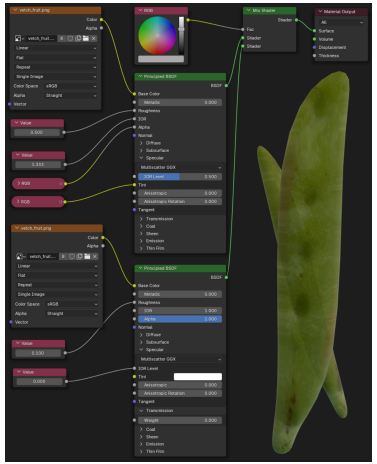


Contributions

PBRT: Importers and Exporters — Scene Layout and Materials









The first, and arguably more impactful of these is a tool for importing scenes from the `PBRT` ray-tracing framework directly into Blender, maintaining as much of the scene layout and material graphs as possible.

And, as an example, this allows us to take a PBRT scene file and import all of its objects and materials, something that has only been done partially before, and not at all for Blender directly.

This tool gives graphics researchers access to a host of new scenes, and thanks to Blender, allows them convert and export them into whatever format they need. Similarly, artists can similarly benefit from this as it makes the PBRT rendering framework more accessible to them.

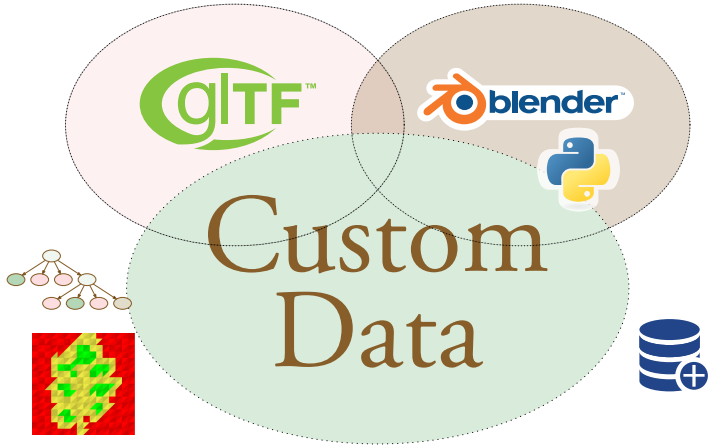
BCON23
Handling Custom Data in `glTF` Files with Exporter/Importer Plugins



BCON23
Handling Custom Data in glTF
Files with Exporter/Importer Plugins



Custom Rendering Data



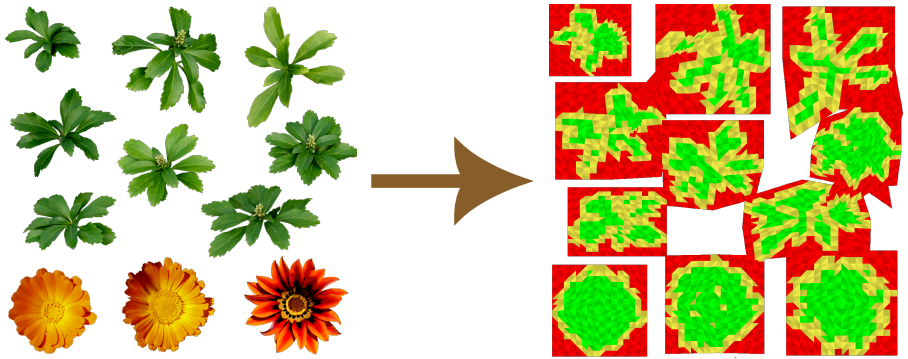
Custom Rendering Data

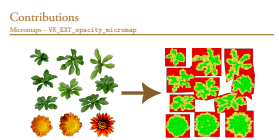


The second piece of tooling that was originally presented at the Blender conference in 2023 tackles a slightly different problem: Storing and associating model and algorithm specific rendering data together with the 3D assets.

Contributions

Micromaps – `VK_EXT_opacity_micromap`





In this work, I presented the tooling in Blender that is used to create addons for storing this data in the `glTF` scene format in a structured way. As well as a small framework for creating such addons.

I also included a number of example use-cases and implementations, such as generating and storing *micromaps* from the associated textures in a 3D model that can then be saved in a `glTF` file.

(And keep these micromaps in mind, as we will return to them later.)

Superluminal Photon Mapping



Superluminal Photon Mapping

HBO Miniseries: Chernobyl

Cherenkov Radiation





– Images from "Chernobyl" Episode 1-2 © HBO

HBO Miniseries: Chernobyl
Cherenkov Radiation



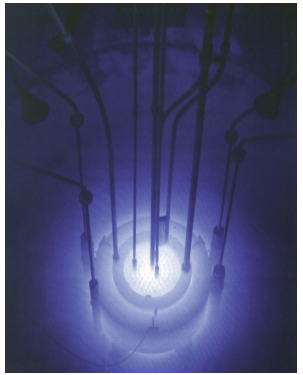


– Images from "Chernobyl" Episode 1-2 © HBO

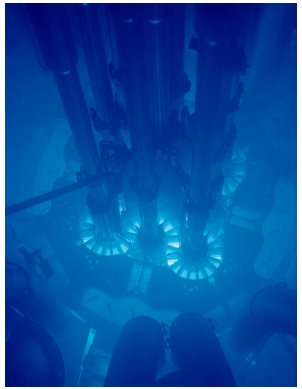
Moving on to the included papers: Here, we will start by talking about the actual particles in the thesis title.

This was an idea I originally came up with after watching the HBO miniseries about Chernobyl that ran back in 2019, where some lighting effects where attributed to *Cherenkov Radiation*, which I ended up doing a bit of research upon:

Nuclear Reactors



Reed Research Reactor



Advanced Test Reactor

©Argonne National Laboratory CC-BY-SA-2.0

Nuclear Reactors

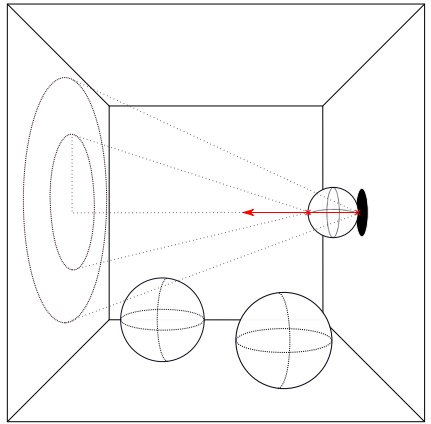
Reed Research Reactor

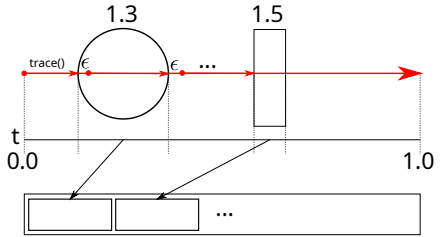
Advanced Test Reactor

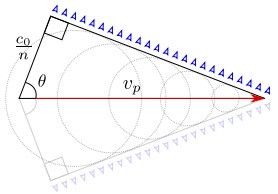
Typically, it is a phenomenon that can be seen in the proximity of nuclear reactors, as you can see from these photographs.

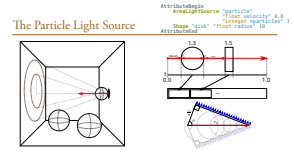
The Particle Light Source

```
AttributeBegin
    AreaLightSource "particle"
                    "float velocity" 0.8
                    "integer nparticles" 1
    Shape "disk" "float radius" 10
AttributeEnd
```



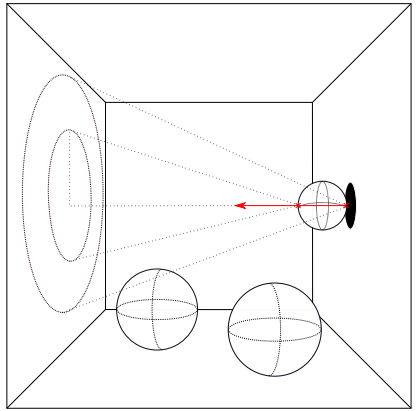


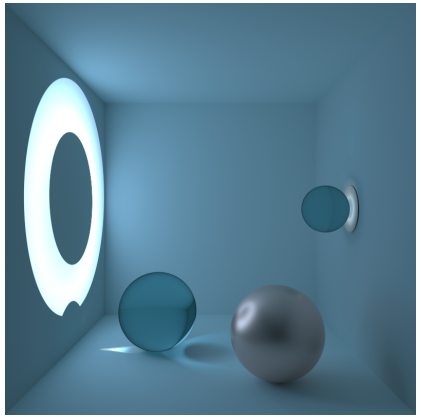




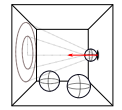
The way we render this, is by first performing a bit of pre-processing: We use ray-tracing to follow a number of virtual charged particles in the scenes, then pick out the segments where the particle is actually superluminal, i.e., the regions where these highly energetic photons will be emitted, and treat these as light sources.

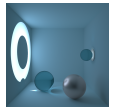
Superluminal Photon Mapping





Superluminal Photon Mapping

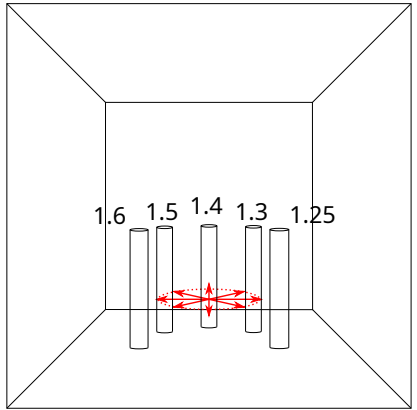


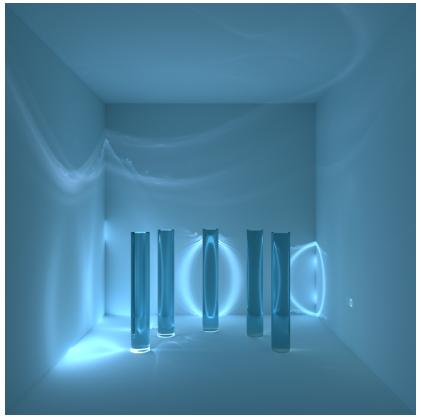


And once all that is done, and we trace a particle like the one visualized with this red-arrow, you can get a rendering that looks something like *this*.

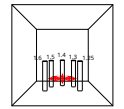
But, as this phenomenon depends on the material density that the particle passes through, we wanted to see how changing that would affect the render.

Superluminal Photon Mapping





Superluminal Photon Mapping

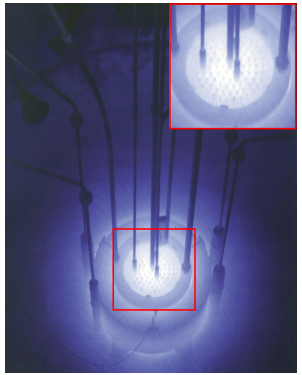


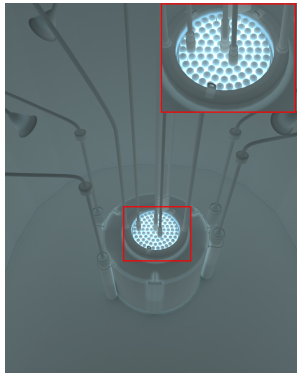


To that end, we created another scene with rods of various index-of-refraction and multiple particles emitted horizontally, which gave us a rendering like *this* one.

Finally, we also wanted to see if we could create a scene similar to where you would actually see this effect.

Superluminal Reactor Rendering





Gustaf Waldemarson and Michael Doggett. "Photon Mapping Superluminal Particles". In: *Eurographics 2020 - Short Papers*. Ed. by Alexander Wilkie and Francesco Banterle. The Eurographics Association, 2020. ISBN: 978-3-03868-101-4. DOI: 10.2312/egs.20201004

Superluminal Reactor Rendering

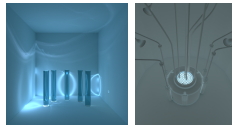


To that end, we created a simple virtual reactor based on photographs of the "Reed research reactor" and used our algorithm on it, giving us this picture here to the right.

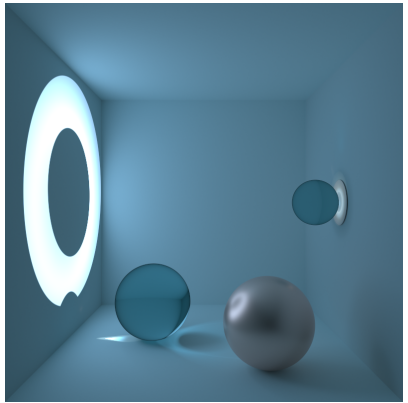
This was then eventually published as a short-paper for the Eurographics conference in 2020.

Contributions

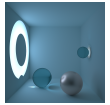
Superluminal Photon Mapping



- First paper to address particles plasma like phenomena
- Novel treatment of particle segments as light-sources



Contributions
Superluminal Photon Mapping
• First paper to address particles plasma like phenomena
• Novel treatment of particle segments as light-sources



- This is then, at least to our knowledge, the first paper to address particles and plasma-like phenomena in a physically based rendering setting.
  - Demonstrating that ray-tracing is quite suitable to simulate charged particles, as well as the photons that they generate.
- And it is consequently a completely novel method for rendering light similar to that of the Cherenkov effect, and is particularly well suited for scientific communication of this phenomenon, even if it might be too simple of an approach for critically accurate atomic simulations.

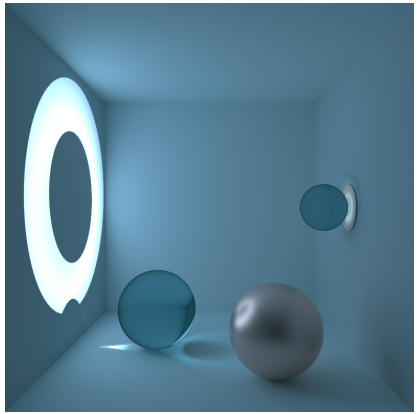
Parallel Axis Split Tasks for Bounding Volume Construction with OpenMP



Parallel Axis Split Tasks for Bounding Volume Construction with OpenMP



Performance Concerns

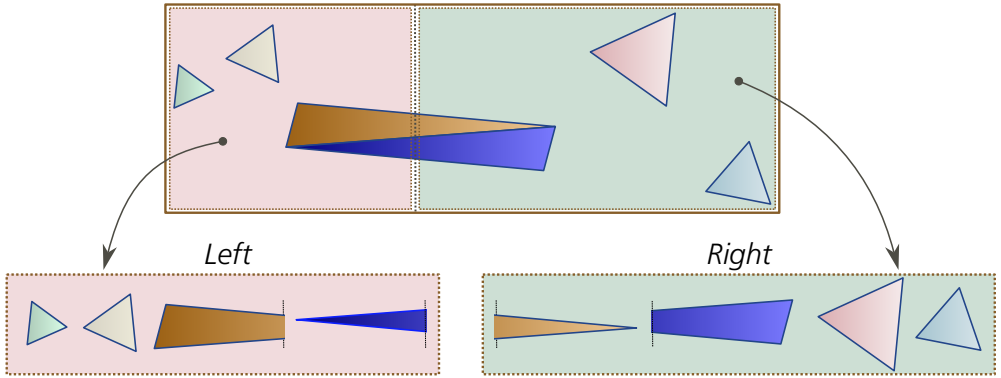


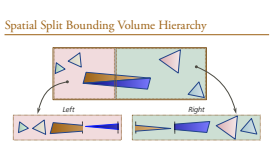
- $>$ 24 hours to render

As we followed up on this work, one very obvious drawback to our algorithm is that it takes a long time to render a frame with acceptable quality: Around 24 hours on one particular machine. Something that we would clearly like to improve.

And as this is a type of ray-tracing problem, the primary way to improve performance is to somehow improve the so-called acceleration structure.

Spatial Split Bounding Volume Hierarchy





In our case, we chose to improve the so-called Spatial Split Bounding Volume Hierarchy method, or simply the *SBVH*, which as (Jeppe alluded to,) is still in one sense the *best* acceleration structure out there.

Parallel Axis Splits

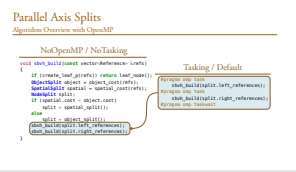
Algorithm Overview with OpenMP

NoOpenMP / NoTasking

```
void sbvh_build(const vector<Reference> &refs)
{
    if (create_leaf_p(refs)) return leaf_node();
    ObjectSplit object = object_cost(refs);
    SpatialSplit spatial = spatial_cost(refs);
    NodeSplit split;
    if (spatial.cost < object.cost)
        split = spatial_split();
    else
        split = object_split();
    sbvh_build(split.left_references);
    sbvh_build(split.right_references);
}
```

Tasking / Default

```
#pragma omp task
    sbvh_build(split.left_references);
#pragma omp task
    sbvh_build(split.right_references);
#pragma omp taskwait
```

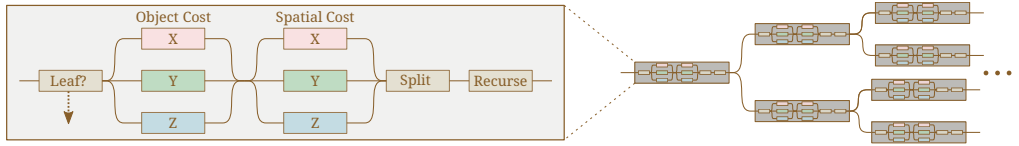
So, if we take a quick look at the typical recursive build algorithm for the *SBVH*, at a very high level, it proceeds as follows:

- If the objects we are currently investigating should be left alone according to some metric we do that and exit. Otherwise:
- We investigate the cost for performing various ways of separating objects into a left and right node candidate…
- And then choose the lowest cost, and continue this process recursively.

And with `OpenMP`, we can even keep this algorithm mostly as-is; simply annotating the recursive calls with so-called *tasking* `#pragmas` will let the compiler parallelize the algorithm, forming a *task-tree* as shown *here* (**next-slide**).

Parallel Axis Splits

The Axis Split Tasks



TaskingOnly

```
Split split_spatial()
{
  for (a = 0; a < 3; ++a)
  {
    // ...
  }
```

ParallelFor

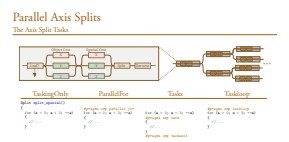
```
#pragma omp parallel for
for (a = 0; a < 3; ++a)
{
  // ...
}
```

Tasks

```
for (a = 0; a < 3; ++a)
#pragma omp task
{
  //...
}
#pragma omp taskwait
```

Taskloop

```
#pragma omp taskloop
for (a = 0; a < 3; ++a)
{
  // ...
}
```

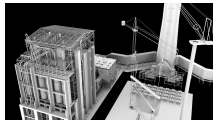
However, if we have many objects, these cost estimations can take a long time, and consequently cause the parallelization to take a while to kick-in.

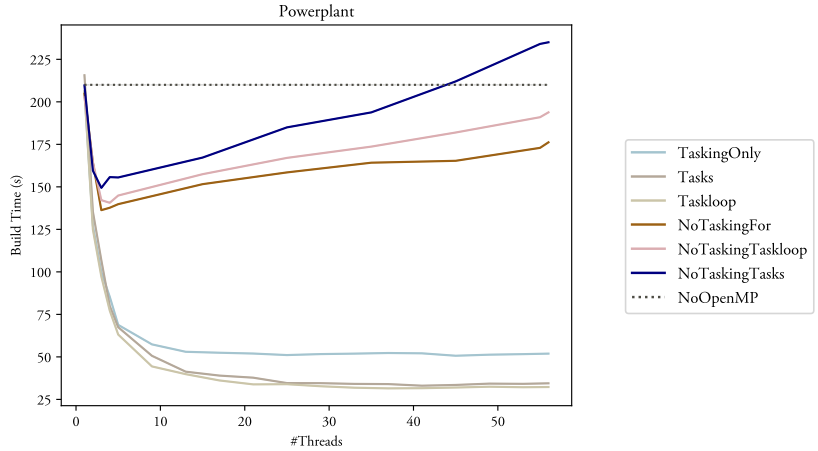
So to that end, we noted that each of these cost estimations can be viewed as separate tasks in-of-themselves that can *also* be parallelized to further improve the overall performance.

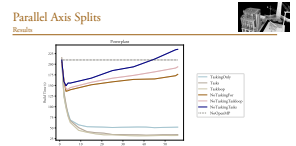
`OpenMP` even has several ways to divide the code for this into tasks, which I briefly outlined here. In theory, these should all do the same thing, but in practice yields some different results…

Parallel Axis Splits

Results







On a quite modern machine with lots of threads, the SBVH build times typically follow the same pattern:

The top three lines represents *only* parallelizing the cost estimations, and since there are only three axes to search, it cannot scale beyond that, and more threads only increase the overhead.

With the recursively spawned tasks, we can instead scale quite a bit further, as you can see from the light-green line *here*.

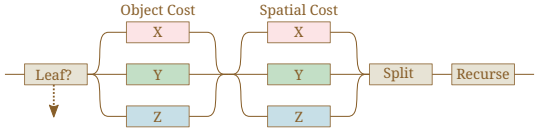
And, when you then also *add* the extra parallelization from the cost estimations, we get the fastest build-times, as shown by our brown and tan lines down here.

Unfortunately, here it is also clear that the parallel axis search will not scale much further than this, as the lines flattens out quite substantially after 20–30 threads.

Contributions

Parallel Axis Splits

- Simple and effective approach to parallelizing the *SBVH* construction.



Gustaf Waldemarson and Michael Doggett. "Parallel Axis Split Tasks for Bounding Volume Construction with OpenMP". In: *Proceedings of the 20th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - GRAPP*. INSTICC. SciTePress, 2025, pp. 347–354. ISBN: 978-989-758-728-3. DOI: 10.5220/0013317100003912



Still, these results were written up in a paper and presented at the *VISIGRAPP* 2025 conference, but in my own words I would say that the primary contribution from this paper is demonstrating that a very simple use of `OpenMP` allows the SBVH algorithm to parallelize quite nicely, while keeping its structure very maintainable for the future.

Something that is arguably still of interest, such as in collision detection, since the modern ray-tracing APIs have turned this particular structure to a mostly inscrutable black-box that we cannot really change anymore.

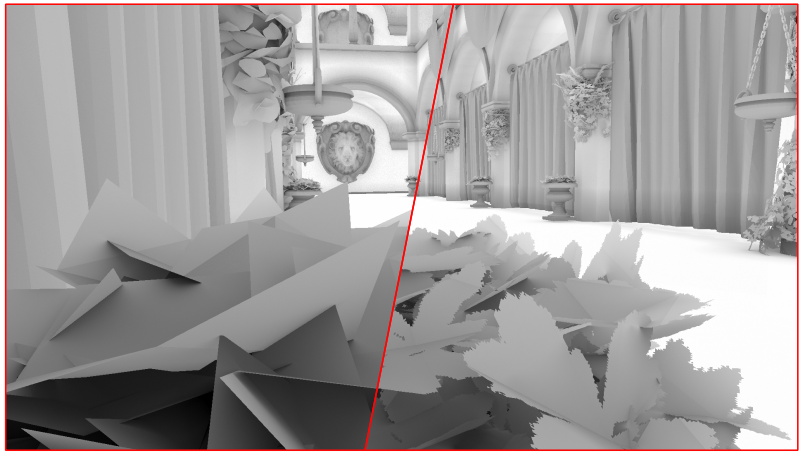
Succinct Opacity Micromaps



Succinct Opacity Micromaps



Opacity Micromaps



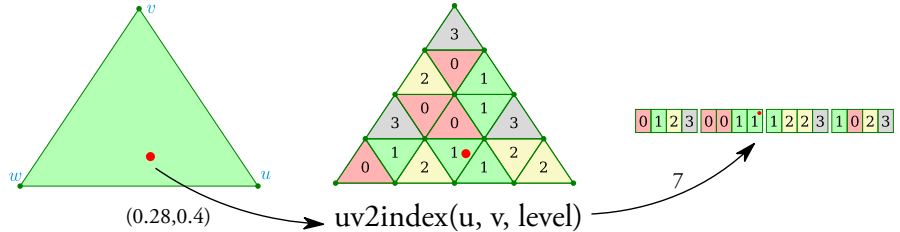


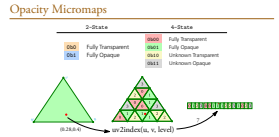
Moving on, but also staying somewhat on the topic of acceleration structures: This next paper is about a specific kind of construct known as an opacity micromap, which you may recall from the second tooling contribution.

It is a kind of helper for the acceleration structure that is intended to simplify the handling of transparent or semi-transparent objects in the real-time ray-tracing pipeline.

Opacity Micromaps

2-State		4-State	
		0b00	Fully Transparent
0b0	Fully Transparent	0b01	Fully Opaque
0b1	Fully Opaque	0b10	Unknown Transparent
		0b11	Unknown Opaque



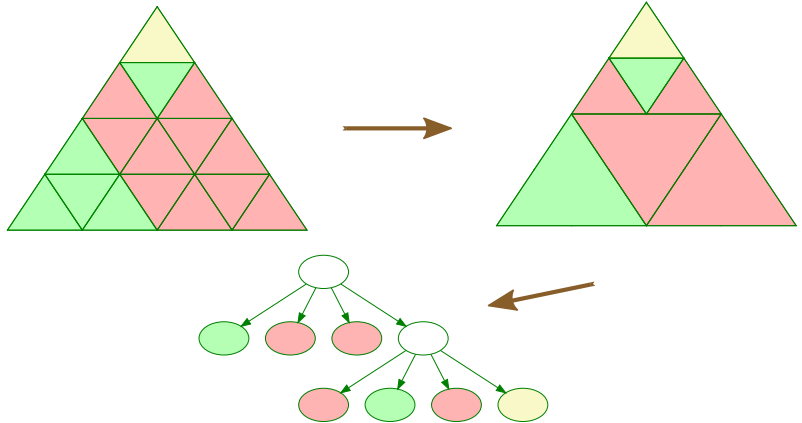


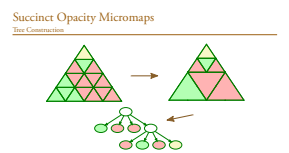
In short; it is an array of values mapped onto fixed subdivided areas of a triangle as designed by a special space-filling curve. In theory, we could assign *any* kind of data to each of these sub-triangle faces, but for opacity micromaps we only care about 2 or 4 values that are mostly self-explanatory:

- Fully transparent and opaque means that that particular sub-triangle is either completely opaque or transparent.
- Unknown values should look up the actual opacity values using some other method, by e.g., calling the `AnyHit`-shader and looking it up in an alpha texture.
  - *(Additionally, these values can be converted to an equivalent `2-state` value, e.g., when you want to use the same micromap, but avoid calling the `AnyHit`-shader, such as for shadow-rays in the ray-tracing pipeline.)*

Succinct Opacity Micromaps

Tree Construction

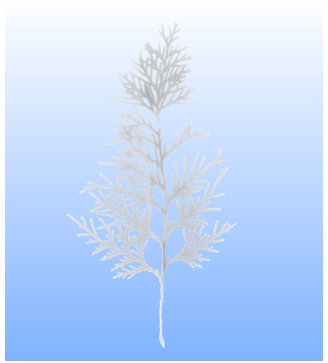


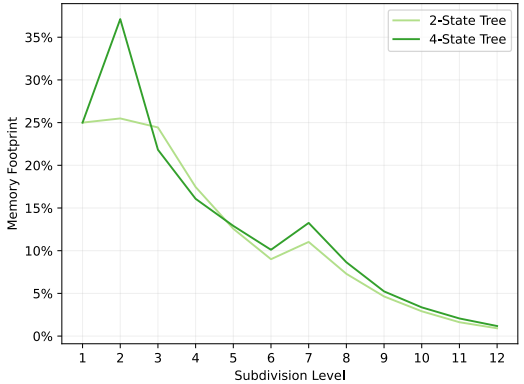


However, this structure grows exponentially with the number of subdivision levels *(effectively by $\mathcal{O}(4^n)$)*, so something needs to be done to handle such large micromaps in practice.

Our first insight here was noting that this structure lends itself to a become a kind of 4-way tree as you can see here.

Memory Footprint Improvement





Best ≈ 1 % of the original size.



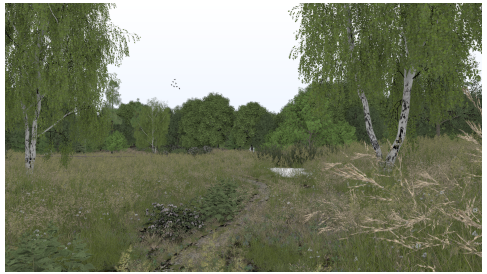
Then, using techniques from so-called *succinct* data-structures, these trees can be greatly compressed.

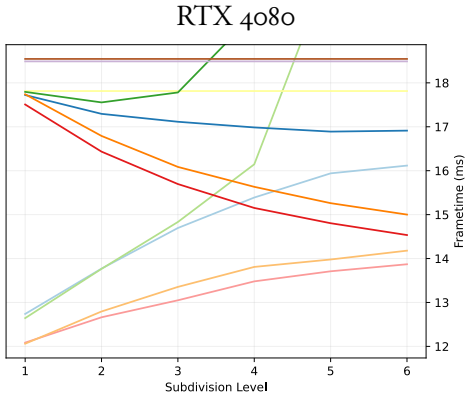
And, looking at the average results over all scenes in this work, this turned out to be very effective at reducing the footprint of the micromaps: Typically reducing it to about 45-25 % of the original size, but in some extreme cases to less than 1 % of the original size.

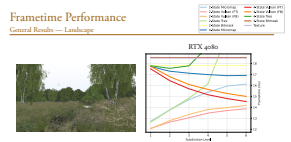
(Or expressed another way: Compressed 110 *times.)*

Frametime Performance

General Results — Landscape







But, compressing the micromaps is not enough though; we also need a method for looking up values again at run-time; which is also included in this particular work.

That method, shown in the green lines here, is then compared with a software emulation of the micromaps (blue), the official API versions of the micromaps (red/orange), as well as a few other variants for performing transparency, such as bitmasks, or textures (brown/purple).

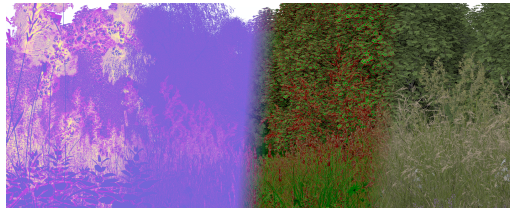
In general terms though: Our look-up algorithm works, but does not really perform very well beyond 3 or 4 subdivision levels, but still provide a decent compression at that point.

Contributions

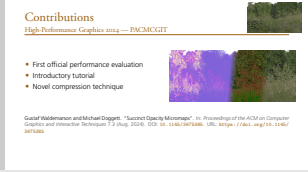
High-Performance Graphics 2024 — PACMCGIT



- First official performance evaluation
- Introductory tutorial
- Novel compression technique



Gustaf Waldemarson and Michael Doggett. "Succinct Opacity Micromaps". In: *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 7.3 (Aug. 2024). DOI: 10.1145/3675385. URL: https://doi.org/10.1145/3675385

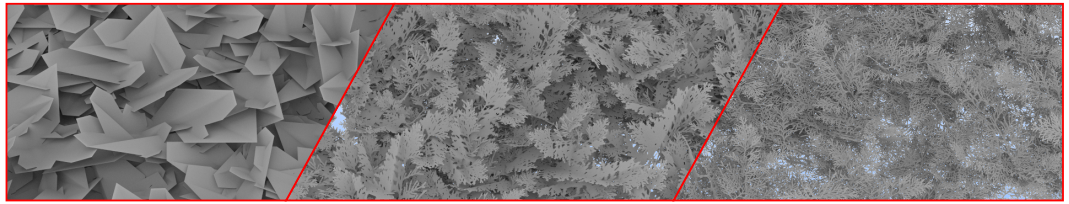


All-in-all, these results were presented at the *High-Performance Graphics Conference in 2024* and published in the conference proceedings and associated journal. In summary however, I would say that the major contributions of this work are:

1. That it is the first open performance evaluation of the official opacity micromap specification.
2. And a bit unintentionally; this work also turned into an excellent introductory tutorial on what micromaps are and how they are used, and is still one of the first results that you will find if you search for "opacity micromaps" in your favorite search engine.
3. And last but not least, the new compression technique that significantly reduce the size of the micromaps, even if some more work is needed for them to be used in real-time at higher subdivision levels.

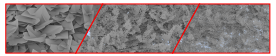
Contributions

US Patent US20250157145A1



Gustaf Waldemarson. "Graphics Processing". Patent Application US20250157145A1. Priority date: 2023-11-15. May 2025. URL: https://patents.google.com/patent/US20250157145A1/en

Contributions
US Patent US20250157145A1



Gustaf Waldemarson. "Graphics Processing". Patent Application US20250157145A1. Priority date: 2023-11-15. May 2025. URL: https://patents.google.com/patent/US20250157145A1/en

Additionally, the compression techniques covered in this paper was also first filed as a US patent, which was eventually published in late 2025, and finally granted a few months ago.

Color and Attribute Micromaps



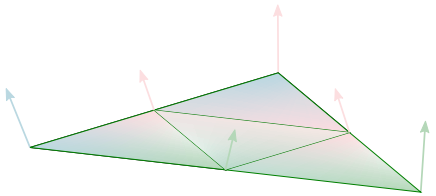
Color and Attribute Micromaps



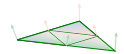
Attribute Micromaps

Going Beyond Opacities

- Base Color
- Metalness and Roughness
- Normals

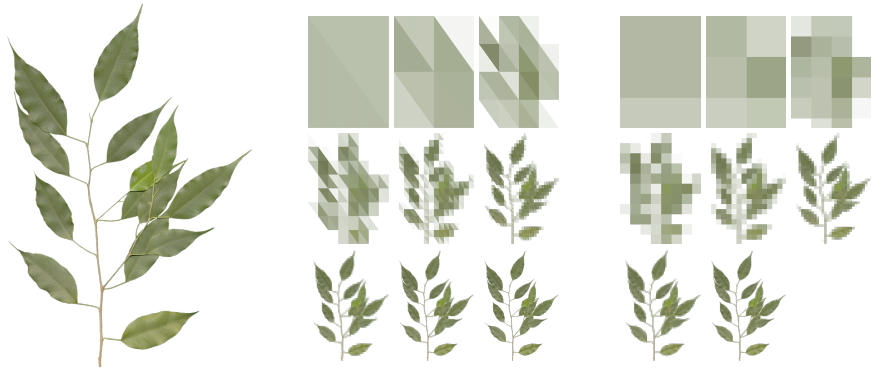


Attribute Micromaps
Going Beyond Opacities
• Base Color
• Metalness and Roughness
• Normals

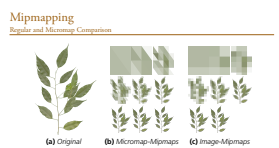


In the earlier paper we handled micromaps with only simple values to mark up opacity, so the natural evolution is of course to see what else we can we fit into that structure: And in this follow-up work we focused on creating a micromap-like structure for colors, and even arbitrary attributes. However, in this work we stuck to common things that are used for shading, such as base-colors, metalness and roughness values, and normals to simplify our evaluation.

Mipmapping
Regular and Micromap Comparison



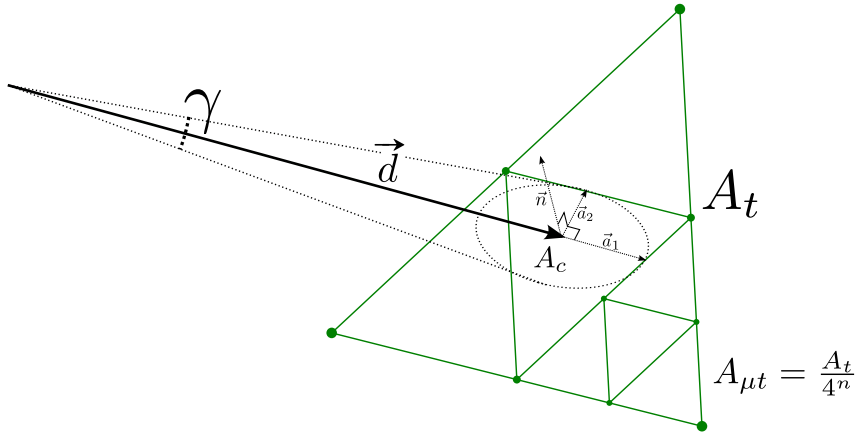
(a) *Original* **(b)** *Micromap-Mipmaps* **(c)** *Image-Mipmaps*

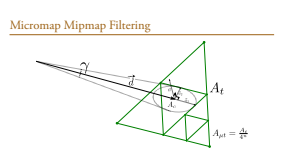


The first new insight came already from the previous paper: Namely that the tree-structure can be re-used to store *intermediate* values, the most natural being an averaged representation of the underlying tree nodes.

This is very similar to the concept of mipmapping, but applied on a per-triangle basis, here shown for a simple quad shown the middle, with a comparison to the regular image-based mipmaps to the right.

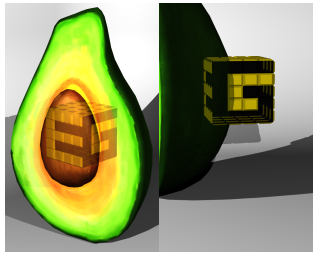
Micromap Mipmap Filtering



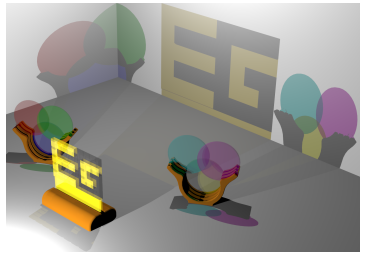


This of course also demands the math and algorithms needed to perform suitable filtering as you go up or down in the mip-map hierarchy depending on the pixel foot-print of the ray. Accordingly, we included a novel algorithm to perform this filtering for a pixel foot-print approximated with a ray-cone.

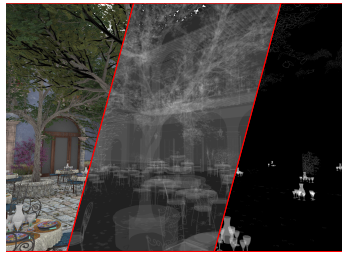
Fixed-Function Operations



Intensity Projection



Colored Shadows



Complexity Counts

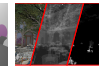
Fixed-Function Operations



Intensity Projection



Colored Shadows



Complexity Counts

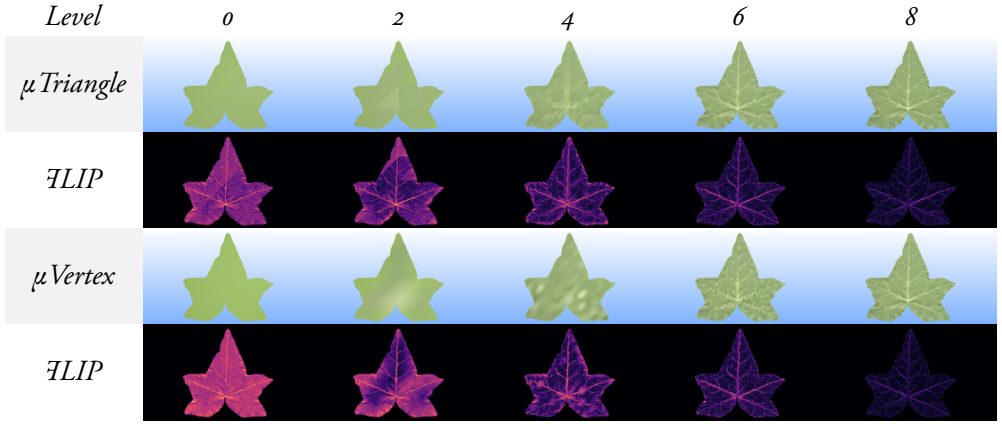
Another novelty in this work is the realization that opacity micromaps are, in one sense, used to let the ray-tracing hardware run more autonomously.

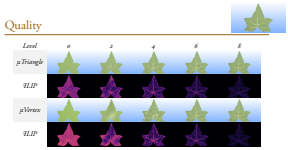
And, since these official micromaps are already prepared for extensions, we explored a number of ways for letting the hardware run even more autonomously, such as letting it perform various operations directly on the objects it intersects based on the available micromap data, all *without* having to call the expensive `AnyHit`-shader.

Some good candidates for this include alpha-blending along a ray, which would, for instance, yield colored shadows, from e.g., stained glass windows, but it could also project some maximum intensity onto the screen or count specific things along the rays.

Quality



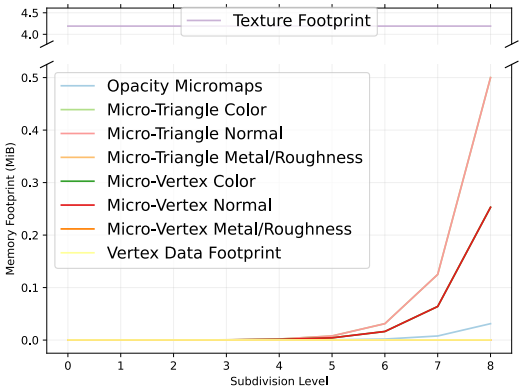


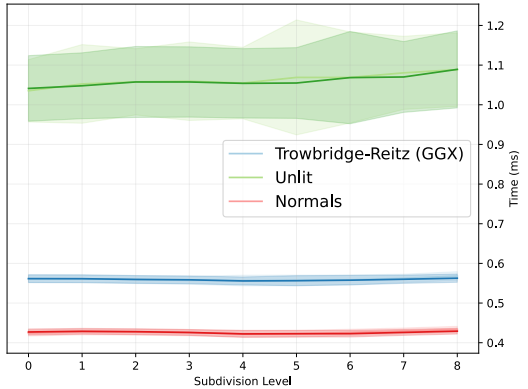


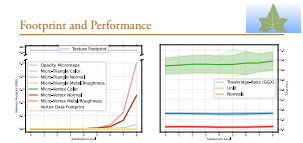
Ideally, assets would need to be made from scratch to take full advantage of this concept, but for this work we instead created micromaps from existing textures so that we could easily see how the rendering *quality* changes as more and more data is available in the micromaps through an increase in subdivision levels.

As an example, for this leaf object you need around 8 subdivision levels for the quality to be acceptable.

Footprint and Performance







And the main benefit from this is that you *can* substantially reduce the memory footprint for these attributes with this changed representation. As you can see here, the sum of the various micromaps are barely a quarter of the full texture footprint here.

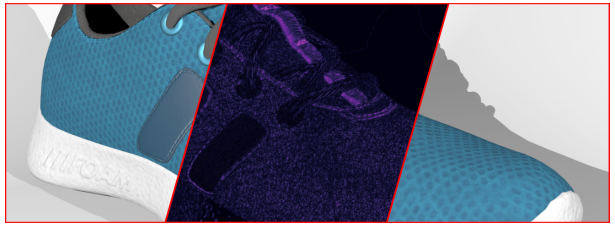
Out of interest, we also measured if the runtime was impacted by the subdivision levels in a few different rendering algorithms, but as you can see, it is basically flat for all reasonable micromap levels.

(And I would also like to emphasize that this is before any kind of compression is applied on the micromap data, although the same applies for textures in this case.)

Contributions

Color and Attribute Micromaps

- Colors and Attributes
  - Quality
  - Performance
- Mipmapping
- Fixed-Function Operations



Currently in submission

Contributions
Color and Attribute Micromaps
• Colors and Attributes
– Quality
– Performance
• Mipmapping
• Fixed-Function Operations
Currently in submission

These results are currently considered complete, and the manuscript for this work is currently in submission. That said, in my own words the major contributions for this work is:

- The extension of micromaps to arbitrary attributes, with an associated quality and performance evaluation the most common ones.
- A novel algorithm for creating something akin to mipmaps with the micromap data, along with a corresponding algorithm for filtering this data during ray-tracing.
- And finally, the initial idea for allowing the ray-tracing hardware to perform more operations autonomously, that *could* be used to avoid calling the `AnyHit` shader in some situations.

Conclusions

Conclusions

Contributions

Overview

Graphics Tooling:

BCON24 PBRT: Create your own Importers and Exporters

BCON23 Handling Custom Data in `glTF` Files with Exporter/Importer Plugins

Included Papers:

I Superluminal Photon Mapping

II Parallel Axis Split Tasks for Bounding Volume Construction with OpenMP

III Succinct Opacity Micromaps

IV Color and Attribute Micromaps

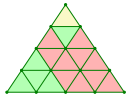
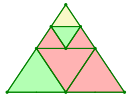
Contributions
Overview
Graphics Tooling:
BCON24 PBRT: Create your own Importers and Exporters
BCON23 Handling Custom Data in glTF Files with Exporter/Importer Plugins
Included Papers:
I Superluminal Photon Mapping
II Parallel Axis Split Tasks for Bounding Volume Construction with OpenMP
III Succinct Opacity Micromaps
IV Color and Attribute Micromaps

Which brings me to the end of my contributions to the field. And just to quickly summarize things; during my work…

- I've created tooling for importing `PBRT` scenes directly into Blender, and added support for including custom rendering data when exporting them again; as well as educated people on these processes at the Blender Conferences in 2023 and 2024.
- I've introduced a new way of representing and handling virtual charged particles during rendering in my *Superluminal Photon Mapping* paper, allowing us to render images of *Cherenkov radiation* in a setting that has not been done before.
- Simplified the parallelization of the construction of spatial split bounding volume hierarchies in the *Parallel Axis Split Tasks* paper.
- Improved the knowledge base about opacity micromaps and their compress-ability in our *Succinct Opacity Micromaps* paper, and,
- Extended aspects about these micromaps to cover arbitrary attributes in my *Color and Attribute Micromaps* paper.

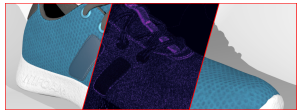
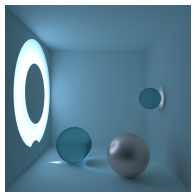
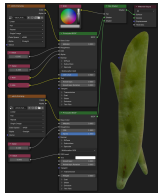
Thanks for Listening

Links / Contact


Rendering Small Things:
Hardware Micromaps and Particles

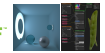
- https://gustafwaldemarson.com/pages/publications/
- gustaf.waldemarson@cs.lth.se
- gustaf.waldemarson@arm.com
- gustaf.waldemarson@gmail.com

arm

LUND UNIVERSITY

Thanks for Listening
Links / Contact
Rendering Small Things:
Hardware Micromaps and Particles
https://gustafwaldemarson.com/pages/publications/
gustaf.waldemarson@cs.lth.se
gustaf.waldemarson@arm.com
gustaf.waldemarson@gmail.com

And with that, I think it's about time to conclude my part of this presentation. For the interested, all details for these publications and tools can, or at least, will *eventually* be found on my personal homepage, linked *here*.

But now, we will take a 5–10 minute break before the actual disputation will begin.

The End

Extra Material

Extra Material

More PBRT Scenes — **PBRT** vs. **Cycles**

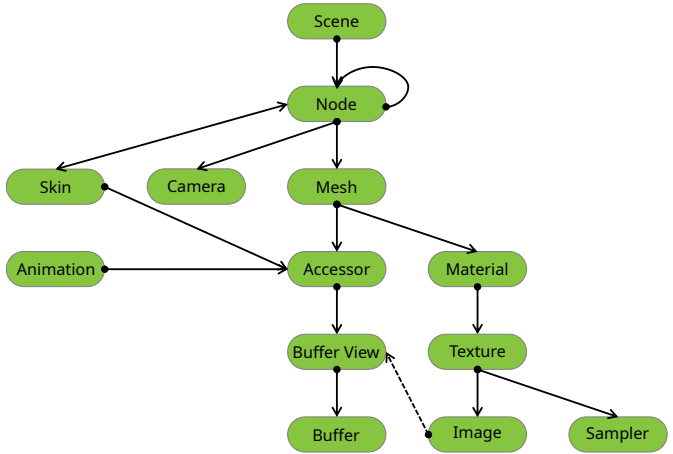


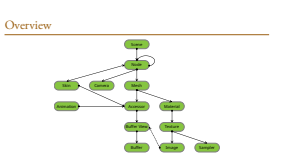


More PBRT Scenes — PBRT vs. Cycles



Overview



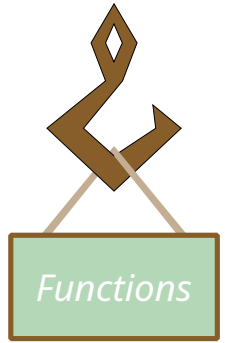


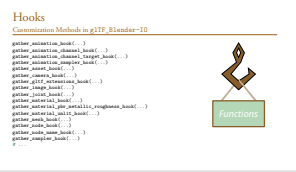
We won't go into the details here, but in essence, a `glTF` file is a container for describing triangle-mesh models, where they are located in 3D space, how they are animated, and what textures they use. But it also go all the way down to how individual bytes are stored.

Hooks

Customization Methods in `glTF_Blender-IO`

```
gather_animation_hook(...)
gather_animation_channel_hook(...)
gather_animation_channel_target_hook(...)
gather_animation_sampler_hook(...)
gather_asset_hook(...)
gather_camera_hook(...)
gather_gltf_extensions_hook(...)
gather_image_hook(...)
gather_joint_hook(...)
gather_material_hook(...)
gather_material_pbr_metallic_roughness_hook(...)
gather_material_unlit_hook(...)
gather_mesh_hook(...)
gather_node_hook(...)
gather_node_name_hook(...)
gather_sampler_hook(...)
# ...
```

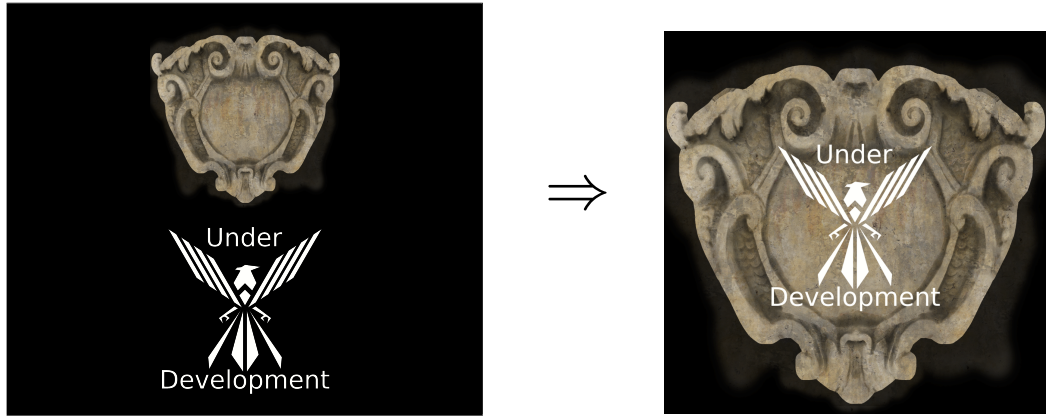




So, what we did was expose and showcase how to use the existing hooks in the Blender `glTF` importer and exporter to do various things…

Texture Watermarking

A Simple Use-Case



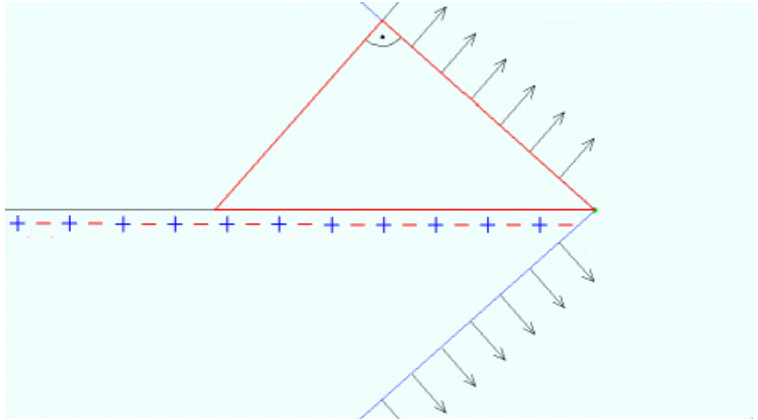
Texture Watermarking
A Simple Use-Case

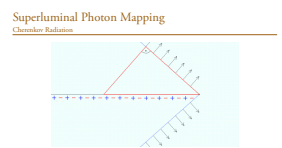


One such thing was a simple watermarking tool that simply adds a watermark on-top of all exported textures.

Superluminal Photon Mapping

Cherenkov Radiation





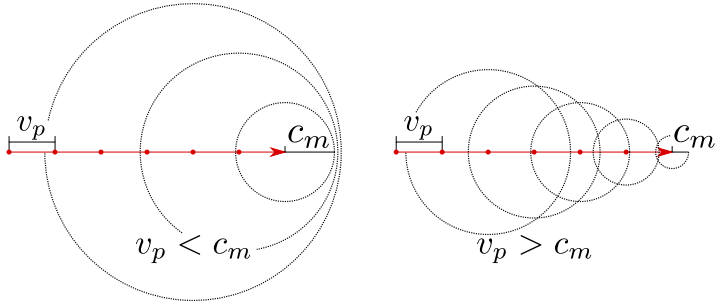
And this video simulating a single particle in 2D was among the few things I found, showing how photons are emitted in a cone from the path of a charged particle.

While it is neat, and describes the phenomenon quite well, it was not exactly the neat looking rendering I was after.

That got me thinking: Could I modify a well known graphics algorithm, namely photon mapping, to generate something similar, but in 3D, and with more visually accurate materials?

The Phenomenon

The Cherenkov Effect

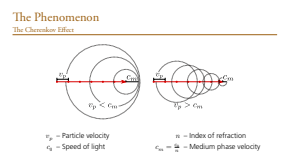


v_p – Particle velocity

c_0 – Speed of light

n – Index of refraction

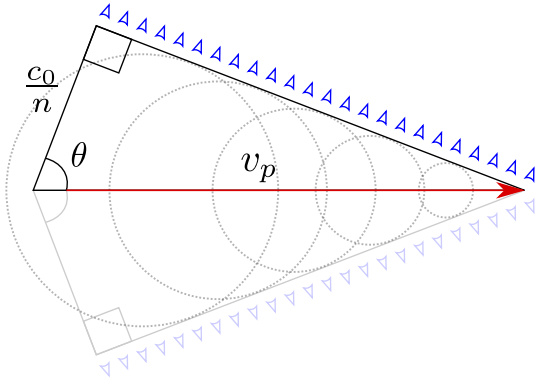
$c_m = \frac{c_0}{n}$ – Medium phase velocity



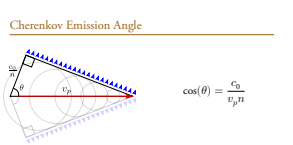
And in short, that is what we did by simulating virtual particles. But if we skip the details: Under normal circumstances, a *slow* particle will interact with matter and send out photons uniformly along its path.

However, when a particle has enough energy such that it moves faster than light does in that medium, these photons begin to constructively interfere: Greatly increasing the light energy output along a certain angle.

Cherenkov Emission Angle



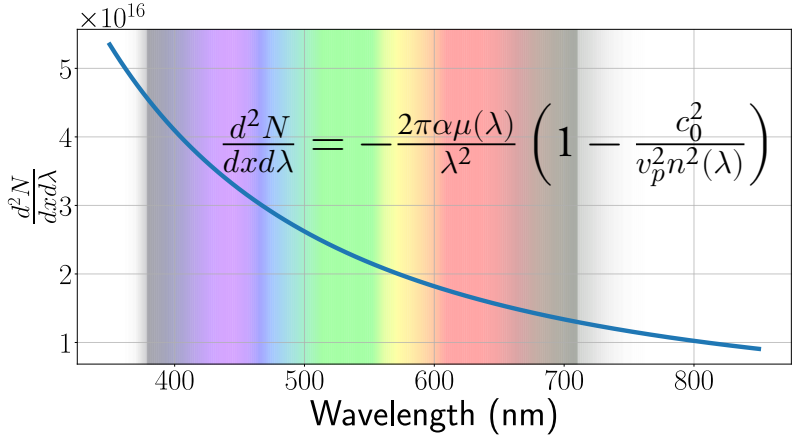
$$\cos(\theta) = \frac{c_0}{v_p n}$$

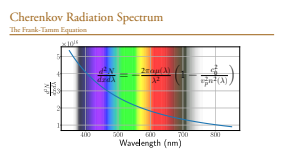


Then, if we know the energy of the particle and some trigonometry, we can find that angle at which photons will generally be emitted.

Cherenkov Radiation Spectrum

The Frank-Tamm Equation

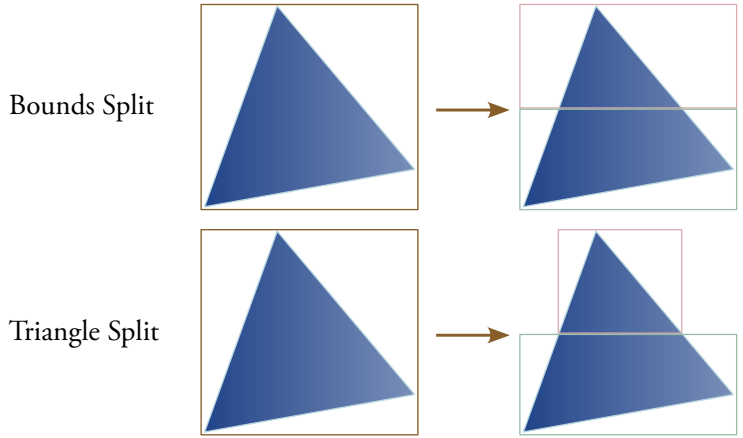


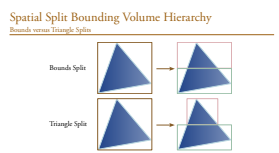


It is also possible to estimate the spectrum for these photons using the so-called Frank-Tamm equation, which shows the general distribution of photons across the electromagnetic spectrum, which, as you can see, distributes more photons towards the ultraviolet end, effectively giving us the brilliant blue color Cherenkov radiation is known for.

Spatial Split Bounding Volume Hierarchy

Bounds versus Triangle Splits





Spatial splits can generate much tighter bounds around all triangles, as when you split the triangle like this, the overall bounds for a split triangle is much smaller than the equivalently split bounding box, as you can see here.

(**Cut?**)

The Acceleration Structure

Spatial Splits Bounding Volume Hierarchy

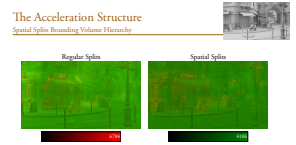


Regular Splits



Spatial Splits





The main benefit from SBVH compared to a regular BVH is that it can reduce the *overall* number of bounding box tests are made, and shift them over to triangle tests, which can be seen in here from the slightly less green tint in the spatial split version, thus improving the "quality" of the structure, or simply the speed at which rays can be tested against it.

The main problem is that building this structure can be surprisingly slow. Thus, we want to be able to create it as quickly, and preferably, easily, as possible.

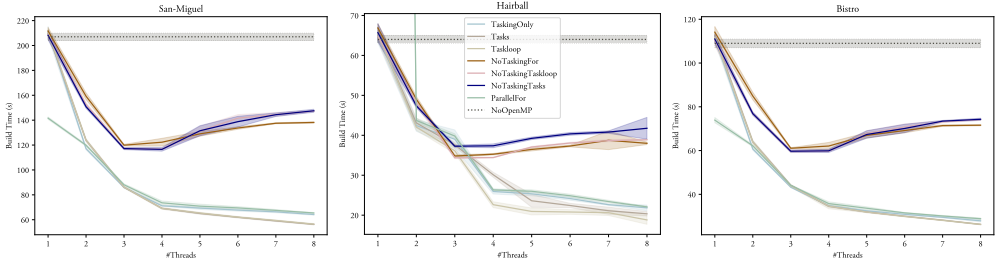
Parallel Axis Splits

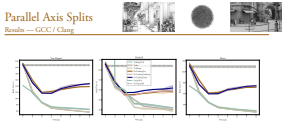
Results — GCC / Clang









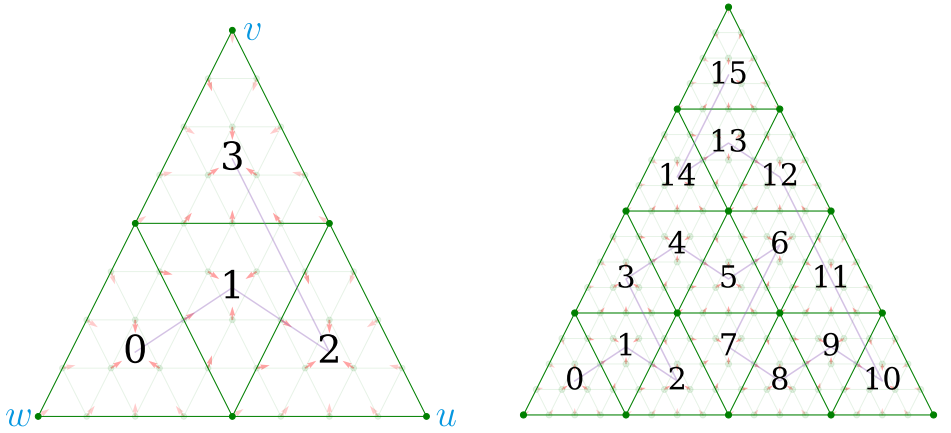


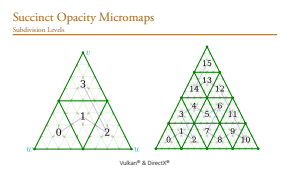
…As shown here for *all* of these methods clustered around the bottom here, which shows an improvement from this kind of task-splittin albeit a relatively minor one for such this particular machine as you can see if you squint and compare the blue line in here to the brown and tan ones a bit further down.

For completeness, we also tried out *only* parallelizing these internal task, without the primary node-level task branching, which as you can see here from the uppermost lines can help, but cannot scale beyond three processors.

Succinct Opacity Micromaps

Subdivision Levels

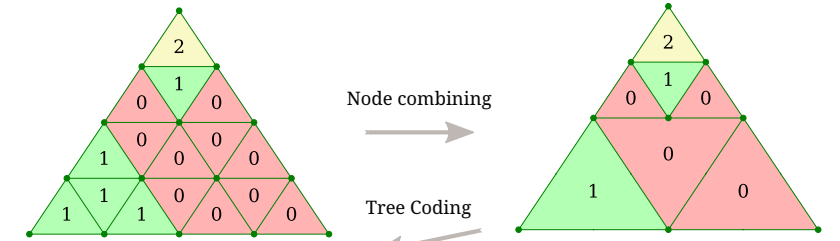




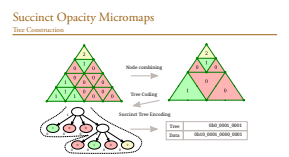
And in order to handle high-resolution features, each triangle can be subdivided a number of times: Here shown for the first two levels.

Succinct Opacity Micromaps

Tree Construction



Tree	0b0_0001_0001
Data	0b10_0001_0000_0001

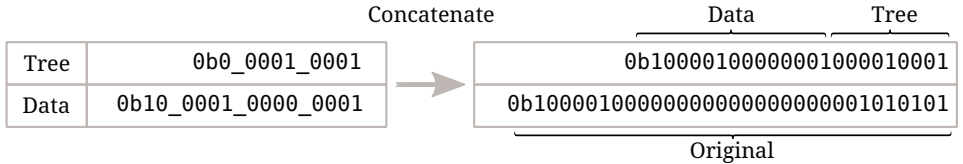


In a more detailed way, micromaps are compressed as follows:

1. In a bottom-up fashion, combine any group of sub-triangles with the same value to a single node.
2. Repeat this for all subdivision levels, forming our 4-way tree, and, then
3. Encode this *succinctly* as a bit-stream.
   - And for clarity, the tree and opacity values, or *data* bits are shown separately here.

Succinct Opacity Micromaps

Tree Construction



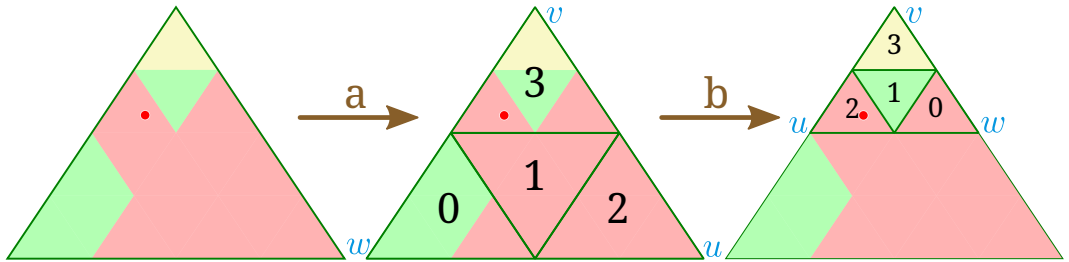
Succinct Opacity Micromaps
Tree Construction

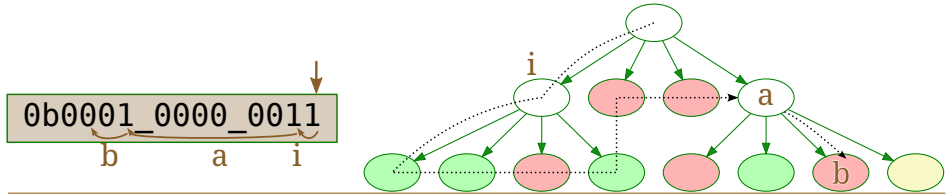


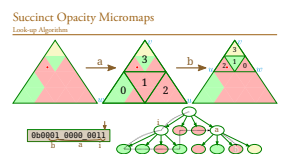
By comparing the final bit-string to the original, it is pretty clear that there are some potential for compression here with the succinct conversion.

Succinct Opacity Micromaps

Look-up Algorithm





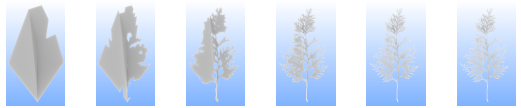


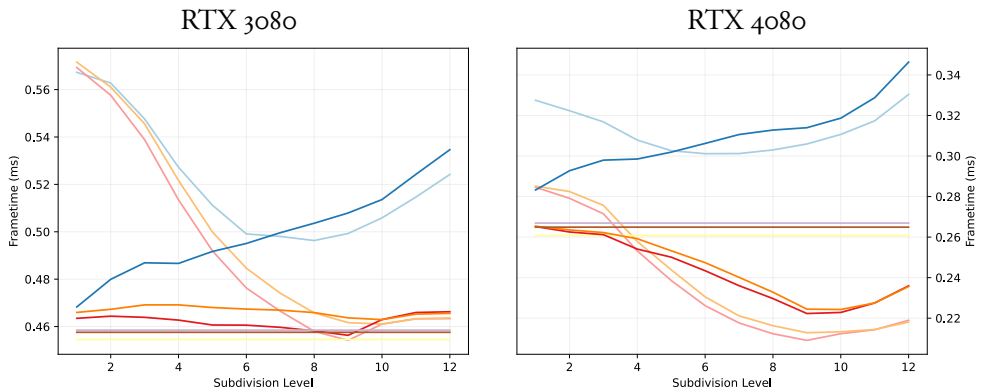
However, we cannot just compress the micromap, we also need a way of looking-up values from this new representation: Which we included in the paper, but in short, you start at the top-node, and depending on which child you should descend to, you step a certain number of bits into the bitstream, and continue until you find a leaf-node.

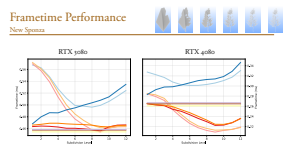
1. After intersecting a triangle, represented by this dot, figure out the index of the child we should visit, and update the barycentric coordinates to that sub-triangle; both of which can be extracted with a single step from the indexing algorithm I presented earlier.
2. This number is now a counter: We need to step at least this number of bits into the bit-string that represents the tree to find the next node to visit.
   - 2.1 However, If we encounter another internal node before this counter is zero, we need to *also* pass through all children owned by it as well, as you can see at the node and bit locations labeled *i*. In other words, we increment our counter by 4.
3. Finally, when this counter is 0, we have found the next node to investigate:
   - 3.1 If the bit value is 1: We repeat the above procedure, going deeper into the tree.
   - 3.2 If the value is 0: We are done and the number of leaf nodes, that is, *zeros* passed until this point is the index into the trailing data bit-stream where we will find our opacity value.

Frametime Performance

New Sponza





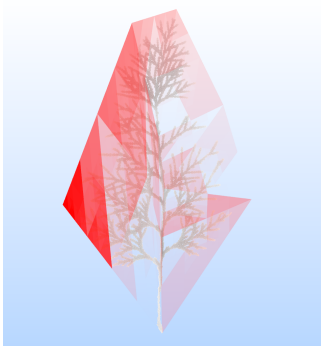


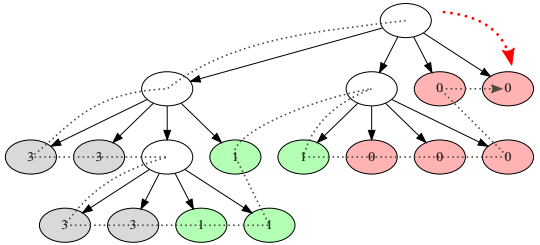
And if we change things around to a somewhat easier scene; a single twig from the New Sponza scene, we can see a bit more interesting performance patterns. Although, note that I have removed the tree results here to allow us to actually see things.

The most notable part here is that for a 3080, we will only ever benefit over just using the texture in this very small region around 9 subdivision levels. Similarly, for a 4080, we must have *at least* 4 subdivisions before we will see any benefits.

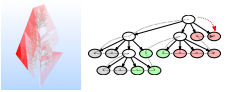
However, please note the axes on these plots: The ranges are relatively small, and the scene is a bit unrealistic.

The Problem(s)





The Problem(s)



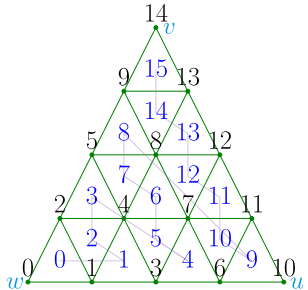
And the reason for that look-up cost is this: When we want to find one of the right-most children, we end up having to scan through all the intervening subtrees, as seen here.

This is fine when the tree is small, but is not acceptable as the tree gets bigger: On average it almost quadruples the number of bits that has to be scanned for each additional tree-level.

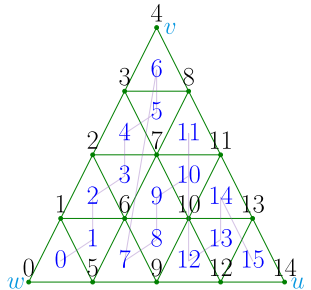
This is obviously problematic for something that ideally should be comparable to an instant $\mathcal{O}(1)$ look-up, similar to a texture fetch.

Attribute Micromaps

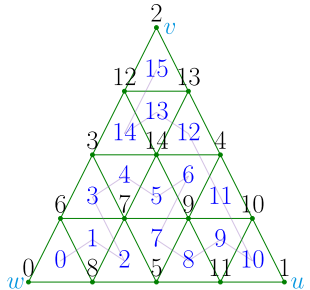
Micro-triangles and Micro-vertices



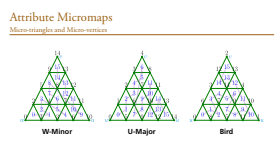
W-Minor



U-Major



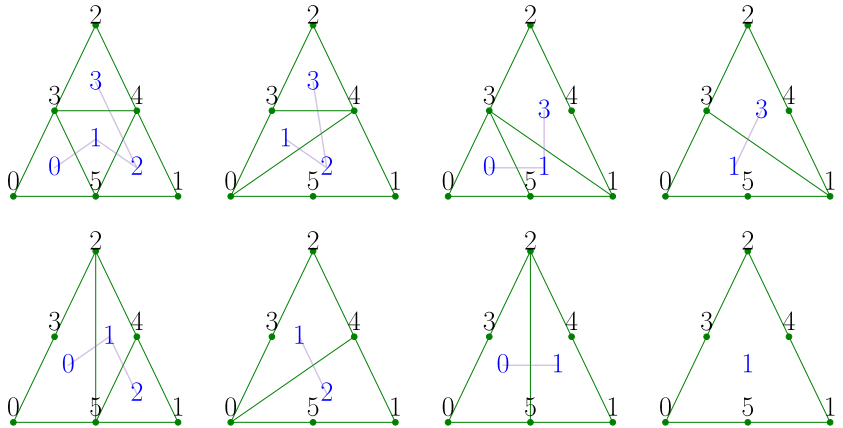
Bird

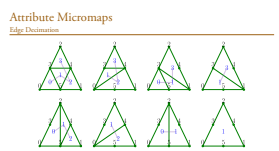


However, one complication that comes up when we start moving beyond opacity values is that we now need to care about micro-*vertices*, and not just micro-*triangles*. And these could be organized in a number of ways, as shown *here*. However, in practice we only used a single of these (namely the *W-Minor* variant), but all three have been used out in the wild.

Attribute Micromaps

Edge Decimation





Another complication is that you typically also want to ensure that the attribute is smoothly interpolated over the surface, thus you need to avoid something known as *cracks*: Basically, discontinuities along the surface.

Shader Access

```
#extension GL_EXT_attribute_micromap : require

layout(binding=0, triangleMicromapEXT) buffer utri
{
    vec4 color;
} uTriAttributes;

layout(binding=1, vertexMicromapEXT) buffer uvtx
{
    vec3 normal;
} uVtxAttributes;
```


Shader Access

#extension GL_EXT_attribute_micromap : require

layout(binding=0, triangleMicromapEXT) buffer utri
{
vec4 color;
} uTriAttributes;

layout(binding=1, vertexMicromapEXT) buffer uvtx
{
vec3 normal;
} uVtxAttributes;

And, we also mocked up a concept that would allow users to use these new micromaps in any kind of shaders.

Attribute Micromaps

Quality

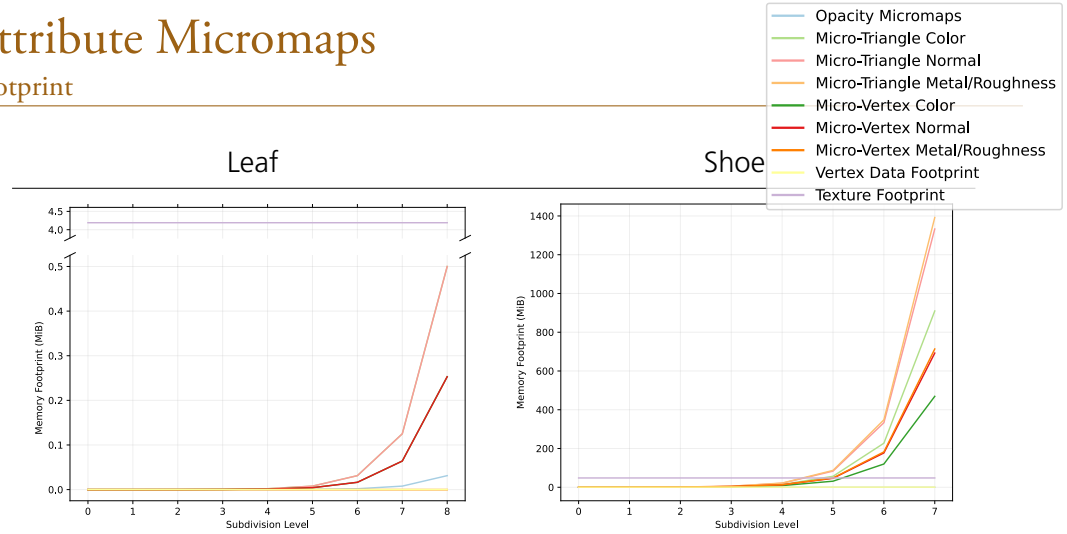


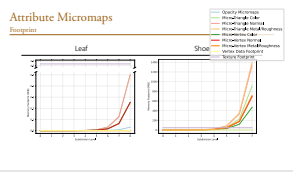


However, for more general objects with a more realistic distribution of vertices, we can do much better, as you can see here.

Attribute Micromaps

Footprint

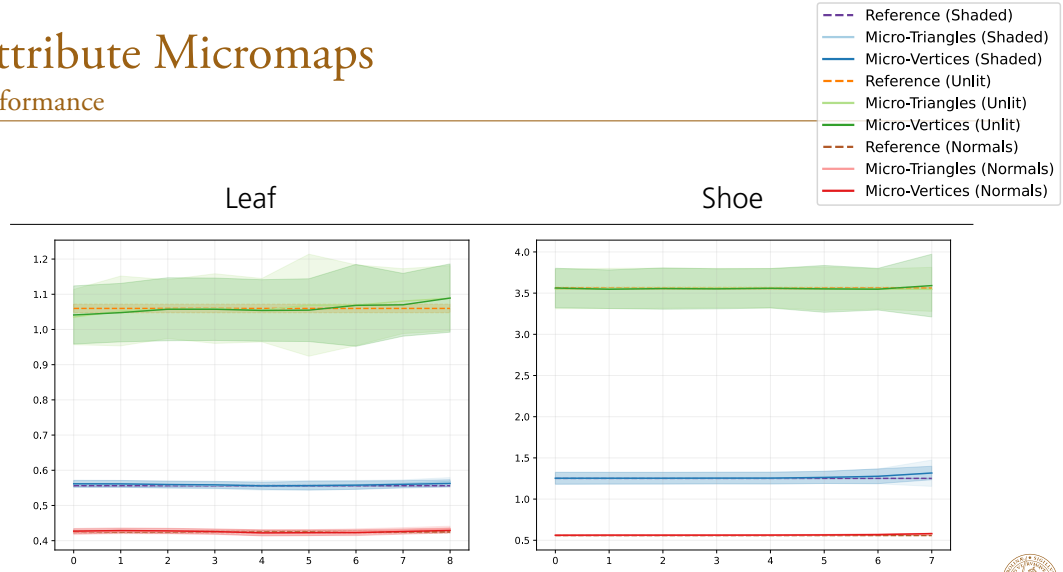


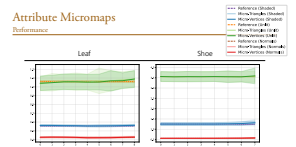


And if you were to look at the memory footprint for each of these, we can see that we are typically below that of textures for more practically useful micromap levels, that is, below the purple line here.

Attribute Micromaps

Performance





Performance-wise however, we were unable to discern any difference from using any of the attribute-micromaps or just using textures, but that is unfortunately expected since it is hard to compete with the hardware accelerated texturing pipeline.

References I

[1] Gustaf Waldemarson. “Graphics Processing”. Patent Application US20250157145A1. Priority date: 2023-11-15. May 2025. URL: https://patents.google.com/patent/US20250157145A1/en.

[2] Gustaf Waldemarson and Michael Doggett. “Color and Attribute Micromaps”. In: *Submission*. 2026.

[3] Gustaf Waldemarson and Michael Doggett. “Parallel Axis Split Tasks for Bounding Volume Construction with OpenMP”. In: *Proceedings of the 20th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - GRAPP*. INSTICC. SciTePress, 2025, pp. 347–354. ISBN: 978-989-758-728-3. DOI: 10.5220/0013317100003912.

[4] Gustaf Waldemarson and Michael Doggett. “Photon Mapping Superluminal Particles”. In: *Eurographics 2020 - Short Papers*. Ed. by Alexander Wilkie and Francesco Banterle. The Eurographics Association, 2020. ISBN: 978-3-03868-101-4. DOI: 10.2312/egs.20201004.

[5] Gustaf Waldemarson and Michael Doggett. “Succinct Opacity Micromaps”. In: *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 7.3 (Aug. 2024). DOI: 10.1145/3675385. URL: https://doi.org/10.1145/3675385.

arm

LUND UNIVERSITY

