

High-Performance Ray Tracing for Physically Based Rendering

Context for Gustaf Waldemarson's PhD Defence

Jeppe Revall Frisvad

April 2026

What is physically based rendering?

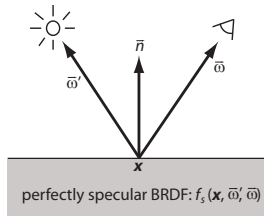
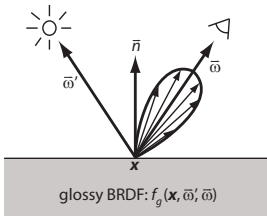
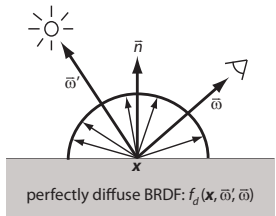
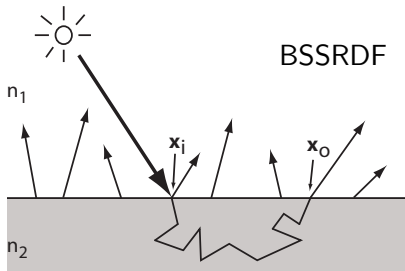
- ▶ **Rendering:** the particular way in which something is performed. (Oxford Advanced Learner's Dictionary)
- ▶ **Rendering an image:** the particular way in which an image is generated.
- ▶ **Photographic rendering:** the particular way in which an image is generated using a camera (including development).
- ▶ **Computer graphics rendering:** the particular way in which an image is generated using a computer.
- ▶ **Physically based rendering:** a physically based way of computing an image.
 - ▶ Think of a photographic rendering as a physical experiment.
 - ▶ Physically based rendering is then an attempt to model photographic rendering mathematically and computationally.
 - ▶ The (unreachable) goal of the models is to predict the outcome of the physical experiment: “taking a picture”.

Models needed for physically based rendering

- ▶ Consider the experiment: “taking a picture”.
- ▶ What do we need to model it?
 - ▶ Camera (light sensor)
 - ▶ Objects (scene geometry)
 - ▶ Luminaires (light sources)
 - ▶ Light (light propagation)
 - ▶ Materials (light scattering and absorption)
- ▶ Mathematical models for these physical phenomena are required as a minimum in order to render an image.
- ▶ We can use very simple models:
 - ▶ Pinhole camera (perspective, lines from a point).
 - ▶ Mathematical primitives (planes, spheres, triangles).
 - ▶ Practical lights (point, directional, spot).
 - ▶ Ray optics (light propagates along straight lines).
 - ▶ Lambertian materials (constant reflectance).
- ▶ **A higher level of realism requires more sophisticated models.**

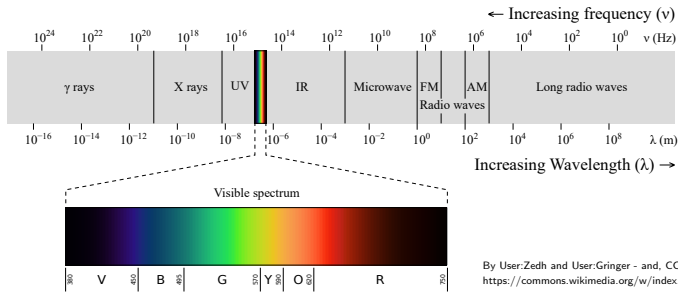
Materials (light scattering and absorption)

- ▶ Optical properties (index of refraction, $n(\lambda) = n'(\lambda) + i n''(\lambda)$).
- ▶ Reflectance distribution functions, $S(\mathbf{x}_i, \vec{\omega}_i; \mathbf{x}_o, \vec{\omega}_o)$.



Light propagation

- ▶ Visible light is electromagnetic waves of wavelengths (λ) from 380 nm to 780 nm.



- ▶ Electromagnetic waves propagate as *rays of light* for $\lambda \rightarrow 0$.
- ▶ Rays of light follow the path of least time (Fermat).
- ▶ How does light propagate in air? In straight lines (almost).
- ▶ The parametrisation of a straight line in 3D is therefore a good, simple model for light propagation: $\mathbf{r}(t) = \mathbf{o} + t\vec{\omega}$, $t \in [t_{\min}, t_{\max}]$, $t_{\max} > t_{\min} > 0$.

Light sources

- ▶ A light source is described by a spectrum of light $L_{e,\lambda}(\mathbf{x}, \vec{\omega}_o)$ which is emitted from each point on the emissive object.
- ▶ A simple model is a light source that from each point emits the same amount of light in all directions and at all wavelengths, $L_{e,\lambda} = \text{const.}$
- ▶ The spectrum of heat-based light sources can be estimated using Planck's law of radiation. Examples:



- ▶ The surface geometry of light sources is (usually) modelled in the same way as other geometry in the scene. **GW: superluminal particles, another type.**

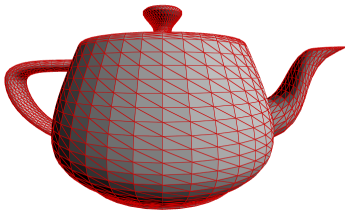
Scene geometry

- ▶ Surface geometry is often modelled by a collection triangles some of which share edges (a triangle mesh).
- ▶ Triangles provide a discrete representation of an arbitrary surface.

Teapot example:



wireframe



faces

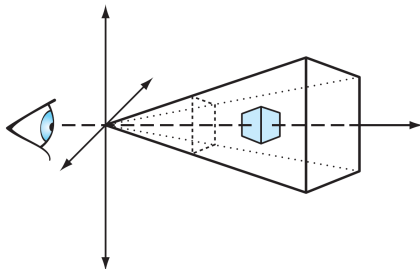


shaded

- ▶ Triangles are useful as they are defined by only three vertices.
And ray-triangle intersection is simple.

Camera

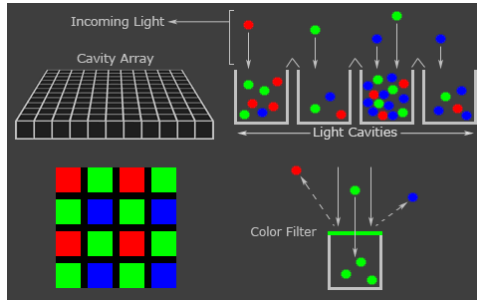
- ▶ A camera consists of a light sensitive area, a processing unit, and a storage for saving the captured images.
- ▶ The simplest model of a camera is a rectangle, which models the light sensitive area (the chip/film), placed in front of an eye point where light is gathered.



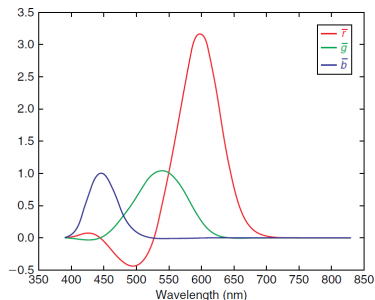
- ▶ We can use this model in two different ways:
 - ▶ Follow rays from the eye point through the rectangle and onwards (ray casting).
 - ▶ Project the geometry on the image plane and find the geometry that ends up in the rectangle (rasterization).

The light sensitive Charge-Coupled Device (CCD) chip

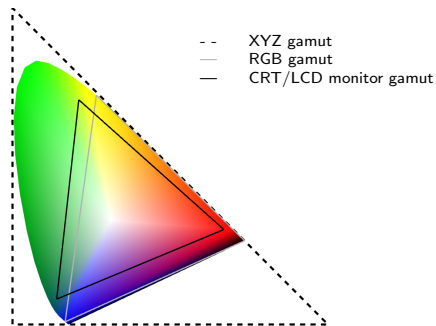
- ▶ A CCD chip is an array of light sensitive cavities.
- ▶ A digital camera therefore has a resolution $W \times H$ measured in number of pixels.
- ▶ A pixel corresponds to a small area on the chip.
- ▶ Several light sensitive cavities contribute to each pixel because the light measurement is divided into red, green, and blue.
- ▶ Conversion from this colour pattern to an RGB image is called demosaicing.



Colorimetry (spectrum to RGB)



CIE color matching functions



The chromaticity diagram

$$R = \int_{\mathcal{V}} C(\lambda) \bar{r}(\lambda) d\lambda \quad , \quad G = \int_{\mathcal{V}} C(\lambda) \bar{g}(\lambda) d\lambda \quad , \quad B = \int_{\mathcal{V}} C(\lambda) \bar{b}(\lambda) d\lambda \quad ,$$

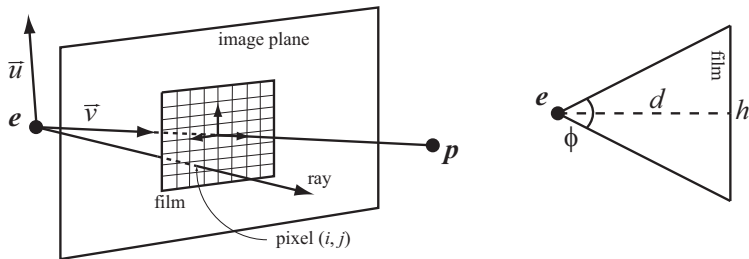
where $\mathcal{V}$ is the interval of visible wavelengths and $C(\lambda)$ is the spectrum that we want to transform to RGB. Spatially varying material properties are oftentimes specified as color variation. **GW: color micromaps, a method for acceleration.**

Ray generation

- Camera description:

Extrinsic parameters		Intrinsic parameters	
$\mathbf{e}$	Eye point	ϕ	Vertical field of view
$\mathbf{p}$	View point	d	Camera constant
$\vec{u}$	Up direction	W, H	Camera resolution

- Sketch of ray generation:



- Given pixel index (i, j) , we find the direction $\vec{w}$ of a ray through that pixel.

Ray-triangle intersection

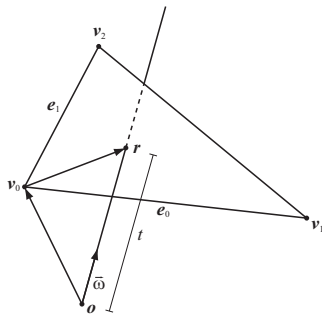
- ▶ Ray: $\mathbf{r}(t) = \mathbf{o} + t\vec{\omega}$, $t \in [t_{\min}, t_{\max}]$.
- ▶ Triangle: $\mathbf{v}_0$, $\mathbf{v}_1$, $\mathbf{v}_2$.

- ▶ Edges and normal:

$$\mathbf{e}_0 = \mathbf{v}_1 - \mathbf{v}_0, \quad \mathbf{e}_1 = \mathbf{v}_0 - \mathbf{v}_2, \quad \mathbf{n} = \mathbf{e}_0 \times \mathbf{e}_1.$$

- ▶ Barycentric coordinates:

$$\begin{aligned} \mathbf{r}(u, v, w) &= w\mathbf{v}_0 + u\mathbf{v}_1 + v\mathbf{v}_2 = (1 - u - v)\mathbf{v}_0 + u\mathbf{v}_1 + v\mathbf{v}_2 \\ &= \mathbf{v}_0 + u\mathbf{e}_0 - v\mathbf{e}_1. \end{aligned}$$

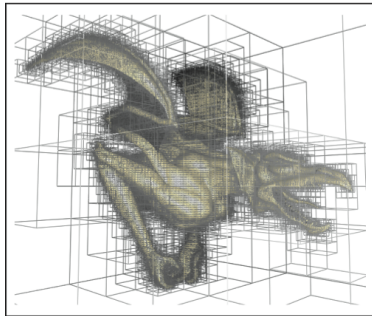


- ▶ The ray intersects the triangle's plane at $t' = \frac{(\mathbf{v}_0 - \mathbf{o}) \cdot \mathbf{n}}{\vec{\omega} \cdot \mathbf{n}}$.
- ▶ Find $\mathbf{r}(t') - \mathbf{v}_0$ and decompose it into portions along the edges $\mathbf{e}_0$ and $\mathbf{e}_1$ to get u and v . Then check

$$u \geq 0 \quad , \quad v \geq 0 \quad , \quad u + v \leq 1 \quad .$$

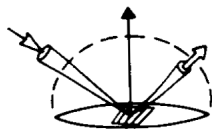
Spatial subdivision

- ▶ To model arbitrary geometry with triangles, we need many triangles.
- ▶ A million triangles and a million pixels are common numbers.
- ▶ Testing all triangles for all pixels requires 10^{12} ray-triangle intersection tests.
- ▶ If we do a million tests per millisecond, it will still take more than 15 minutes.
- ▶ This is prohibitive. We need to find the relevant triangles.
- ▶ Spatial data structures offer logarithmic complexity instead of linear.
- ▶ A million tests become twenty operations ($\log_2 10^6 \approx 20$).
- ▶ 15 minutes become 20 milliseconds.
- ▶ **GW: fast construction of high quality bounding volume hierarchy (BVH).**



Shading a triangle mesh

- ▶ The energy travelling along a ray is measured in radiance $L = \frac{d^2\Phi}{\cos\theta dA d\omega}$ (radiant flux per projected area per solid angle).
- ▶ The outgoing radiance L_o at a surface point $\mathbf{x}$ in the direction $\vec{\omega}_o$ is the sum of emitted radiance L_e and reflected radiance L_r :

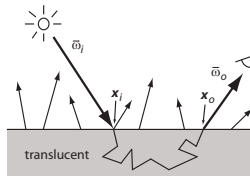
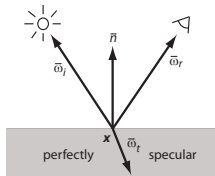
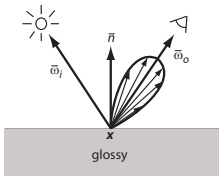
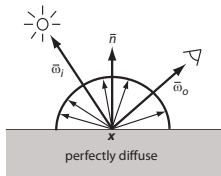


$$L_o(\mathbf{x}, \vec{\omega}_o) = L_e(\mathbf{x}, \vec{\omega}_o) + L_r(\mathbf{x}, \vec{\omega}_o) .$$

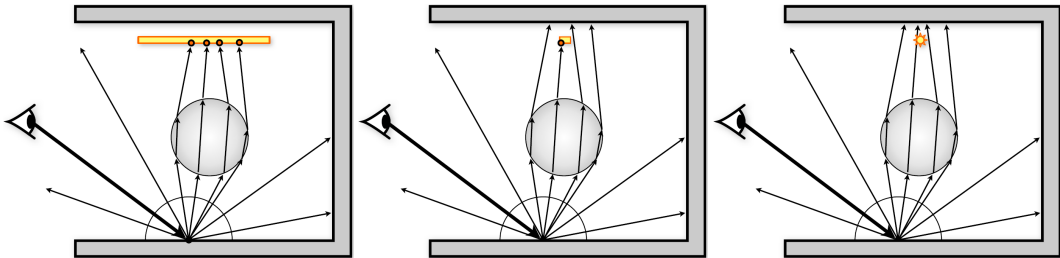
- ▶ Reflected radiance is computed using a bidirectional reflectance distribution function (BRDF) f_r and an estimate of incident radiance L_i at the surface point.
- ▶ Algorithm:
 - ▶ Find all visible points in a scene (ray casting).
 - ▶ At every point in the scene (with BRDF f_r), gather the contribution from all directions of incidence (shader and light source sampler) and add emission.
 - ▶ Evaluate direct and indirect illumination independently using Monte Carlo integration (sampling).

The importance of specular paths

- ▶ Light through windows.
 - ▶ Light in water.
 - ▶ Glossy materials.
 - ▶ Translucent materials.
-
- ▶ But occluded specular paths are hard to find.



The caustic challenge



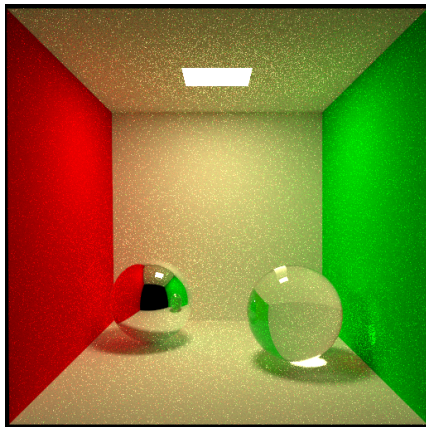
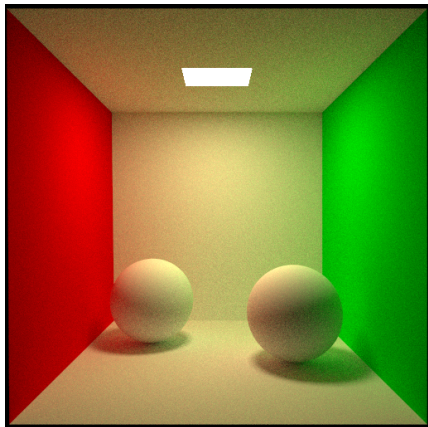
[Hachisuka et al. 2013]

- ▶ How to find a small light source occluded by a transparent specular object (glass, water)?
- ▶ Translucent objects carry the same challenge.
- ▶ **GW: superluminal particles carry the same challenge.**

References

- Hachisuka, T., Jarosz, W., Bouchard, G., Christensen, P., Frisvad, J. R., Jakob, W., Jensen, H. W., Kaschalk, M., Knaus, C., Selle, A., and Spencer, B. State of the art in photon density estimation. In *SIGGRAPH 2012 Courses*, Article 6. ACM, 2012.
- Hachisuka, T., Jarosz, W., Georgiev, I., Kaplanyan, A. S., and Nowrouzezahrai, D. State of the art in photon density estimation. In *SIGGRAPH Asia 2013 Courses*, Article 15. ACM, 2013.

Path-traced caustics are noisy

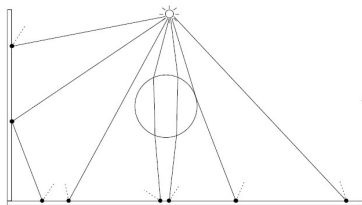


Regular path tracing, 100 paths per pixel.

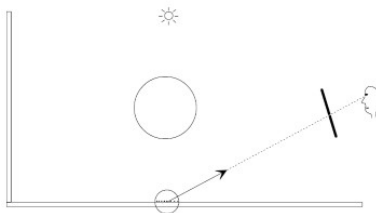
Reusing computations (photon mapping)

- ▶ Store light path vertices (photon mapping)
 - Path reuse vs. memory consumption
- ▶ Algorithm: emit, trace, store, and do flux density estimation when rendering.

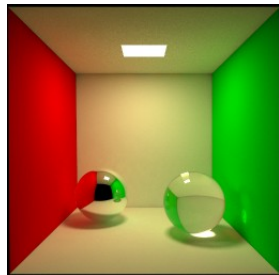
[Jensen and Christensen 2000]



trace and store



flux density estimation



photon mapping

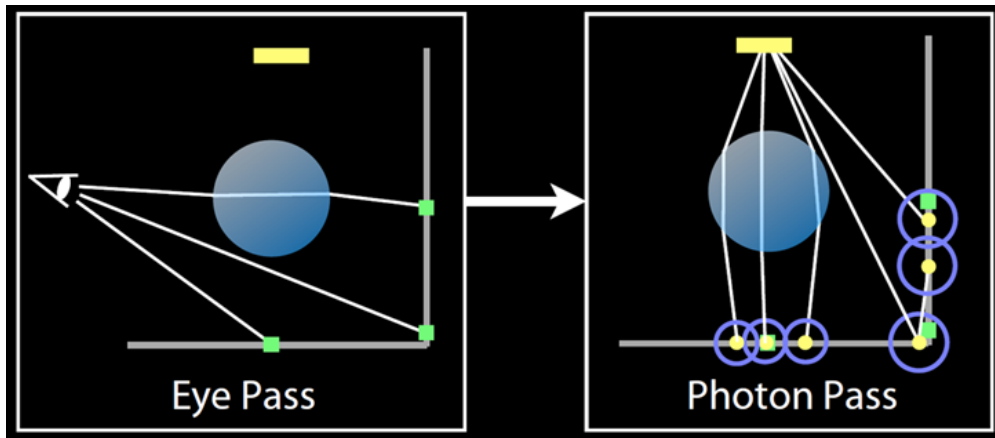
- ▶ “Photons” in photon mapping are *flux packets*.
- ▶ Flux carried by a ray changes like radiance upon diffuse and specular reflection.

The Light of Mies van der Rohe

Photon mapping excels at caustics and is thus an excellent choice for architectural lighting simulation.

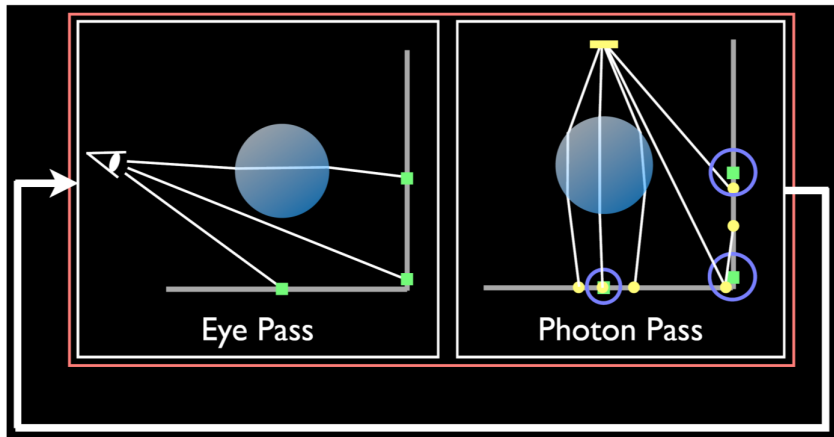
Reusing computations (photon splatting)

- ▶ Storing eye path vertices (photon splatting).
 - Path reuse vs. accuracy in density estimation.

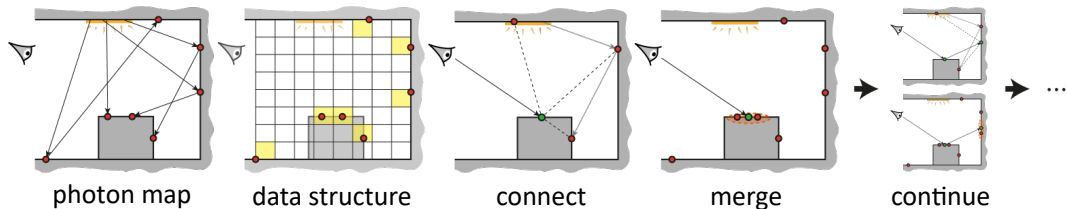


Reusing computations (stochastic progressive photon mapping)

- Store all path vertices and refine progressively (stochastic progressive photon mapping).
 - Path reuse, little memory, but slow convergence



Unified framework and a path tracing revolution



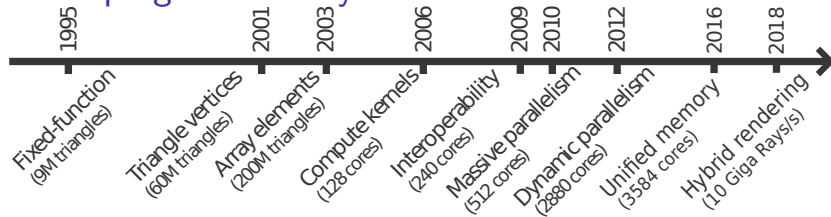
[Hachisuka et al. 2013]

- ▶ Bidirectional path tracing (vertex connection).
- ▶ Progressive photon mapping (vertex merging).
- ▶ Same framework, different integrators.
- ▶ Combine this architectural unification with developments in denoising and GPU:
 - ▶ Consensus on path tracing in the movie industry [Keller et al. 2015].
 - ▶ The GPU can no longer be ignored in production rendering [Christensen et al. 2025].

References

- Keller, A., Fascione, L., Fajardo, M., Georgiev, I., Christensen, P., Hanika, J., Eisenacher, C., and Nichols, G. The path tracing revolution in the movie industry. In *SIGGRAPH 2015 Courses*, Article 24. ACM, 2015.
- Christensen, P., Fong, J., Kilpatrick, C., Gonzalez, F., Ravichandran, S., Shah, A., Jaszewski, E., Friedman, S., Burgess, J., Roy, T. M., Nettleship, T., Seshadri, M., and Salituro, S. RenderMan XPU: a hybrid CPU+GPU renderer for interactive and final-frame rendering. *Computer Graphics Forum* 44(8), e70218. 2025.

Timeline on the programmability of the GPU



1995 Fixed-function rasterization pipeline in hardware.

2001 Vertex shaders (first programmable part of the pipeline).

2003 Fragment/pixel shaders (GPGPU).

2006 Unified shaders (CUDA) and geometry shaders.

2008 Double precision arithmetics.

2009 Compute shaders (interoperability) and tessellation shaders.

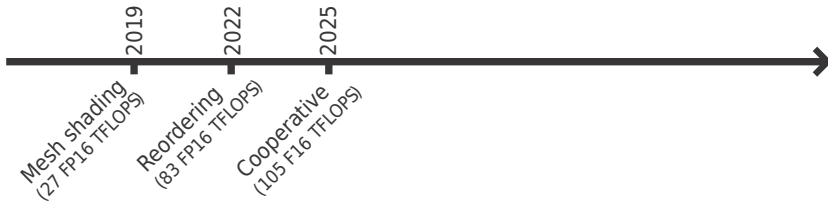
2010 Streaming multiprocessor architecture. **Programmable ray tracing pipeline on the GPU.**

2012 Dynamic parallelism (threads spawn threads).

2016 Unified memory (on demand data migration and dynamic memory allocation).

2018 **Hybrid rendering** (CUDA cores, **RT cores**, tensor cores).

Timeline on the programmability of the GPU in the 2020s

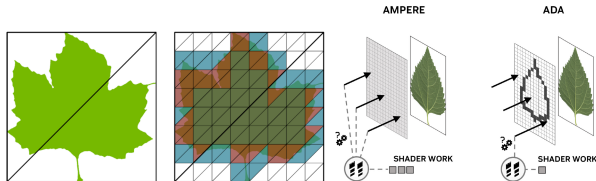


2019 Mesh shading pipeline (programmable geometry processing).

2022 Shader execution reordering (dynamic scheduling for ray tracing).

2025 Neural techniques in shaders (cooperative vectors enable tensor-core execution).

2022 Hardware support for **opacity micromaps** to improve performance for alpha-tested geometry.

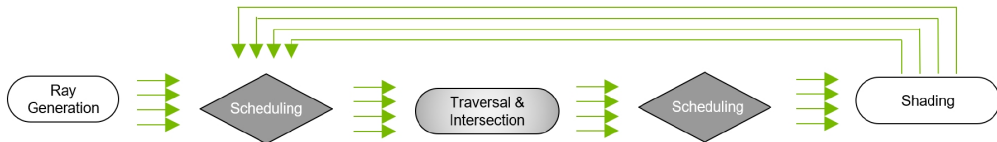


NVIDIA Ada GPU Architecture.

<https://images.nvidia.com/aem-dam/Solutions/geforce/ada/nvidia-ada-gpu-architecture.pdf>

Hybrid rendering

RAY TRACING



RASTER

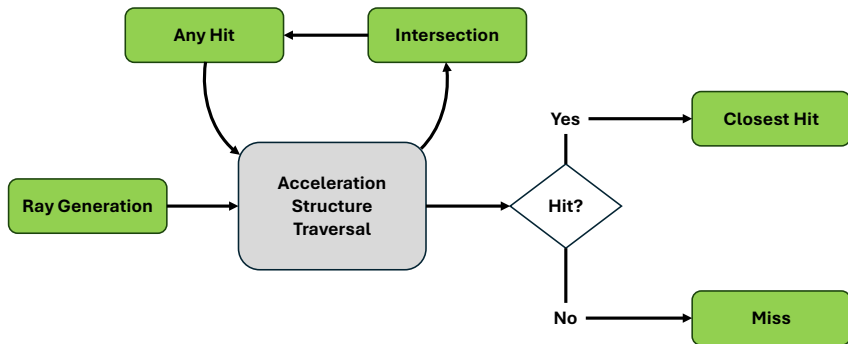


- Ray tracing and rasterization pipelines operate simultaneously and cooperatively.

References

- Emmett Kilgarriff, Henry Moreton, Nick Stam, and Brandon Bell. NVIDIA Turing GPU Architecture In-Depth. September 2018. <https://devblogs.nvidia.com/nvidia-turing-architecture-in-depth/>
- NVIDIA Turing GPU Architecture: Graphics Reinvented. NVIDIA whitepaper, WP-09183-001_v01. September 2018.

GPU ray tracing



- ▶ The “Ray Generation” program uses the camera model.
- ▶ The “Intersection” program is for ray-object intersection (hardwired for triangles).
- ▶ The “Closest Hit” program is the shader.
- ▶ The “Any Hit” program evaluates coverage and transparency.
- ▶ **GW: succinct opacity micromaps, fast compressed avoidance of Any Hit.**

Ray tracing is now performant

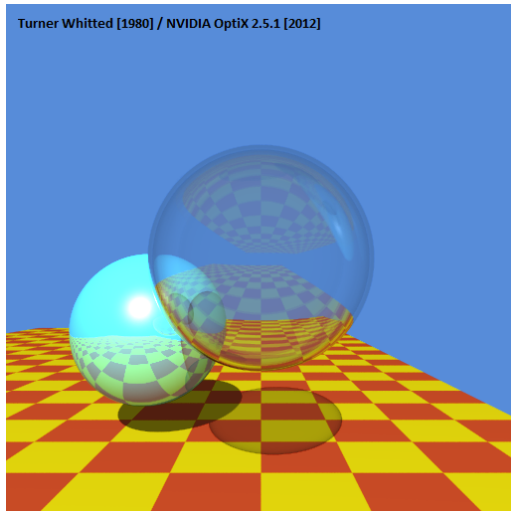
- ▶ 74 minutes in the original ray tracing paper of Turner Whitted [1980].
- ▶ More than 300 frames per second using GPU ray tracing in 2012.
- ▶ That's a 6 orders of magnitude speed-up! (Beats Moore's law.)

$$74 \cdot 60 / (1/300) = 1,332,000$$

$$2^{(2012-1980)/2} = 2^{16} = 65,536$$

But 950 fps on RTX in 2022.

- ▶ Ray tracing is now mature and widely spread in production rendering.

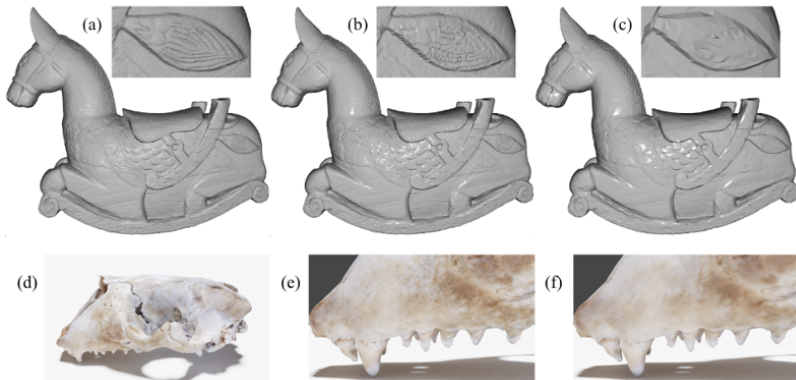


References

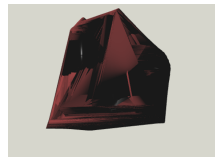
- Whitted, T. An improved illumination model for shaded display. *Communications of the ACM* 23(6), pp. 343–349, June 1980.
- Parker, S. G., Bigler, J., Dietrich, A., Friedrich, H., Hoberock, J., Luebke, D., McAllister, D., McGuire, M., Morley, K., Robison, A., and Stich, M. OptiX: a general purpose ray tracing engine. *ACM Transactions on Graphics* 29(4), pp. 66:1–66:13, July 2010.

Content still grows in complexity

- ▶ The details increase (for scanned objects in particular).
- ▶ The Stanford dragon is $\sim 560\text{k}$ vertices, $\sim 1100\text{k}$ triangles.
- ▶ Mesh simplification removes important details.



photo



16 bits for indices



32 bits for indices

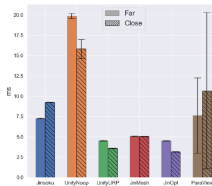
Performance

- ▶ High-poly meshes are expensive to render with conventional pipelines.
- ▶ In real-time rendering, the frametime budget is 16 ms.
- ▶ The budget is lower in VR (12 or 10 ms and two views).

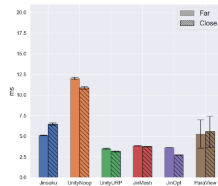
	Seal Skull	Wing	Nobby
no. of triangles	14,504,882	38,629,758	32,905,214
no. of vertices	21,757,335	92,010,363	16,970,666
model size	1.154 GB	3.819 GB	1.723 GB

Reference

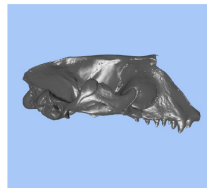
- Jensen, M. B., Jacobsen, E. I., Frisvad, J. R., and Bærentzen, J. A. Tools for virtual reality visualization of highly detailed meshes. In *The Gap between Visualization Research and Visualization Software (VisGap 2021)*, pp. 19-26. The Eurographics Association, 2021.



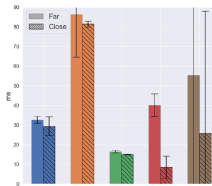
(a) Render times for Seal Skull on Quest.



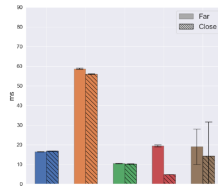
(b) Render times for Seal Skull on Index.



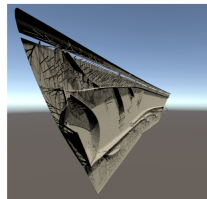
(c) Seal Skull visualized in Jinsoku.



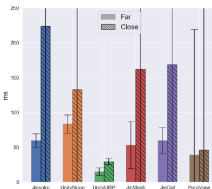
(d) Render times for Wing on Quest.



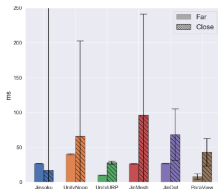
(e) Render times for Wing on Index.



(f) Wing visualized in Unity_noop.



(g) Render times for Nobby on Quest.



(h) Render times for Nobby on Index.

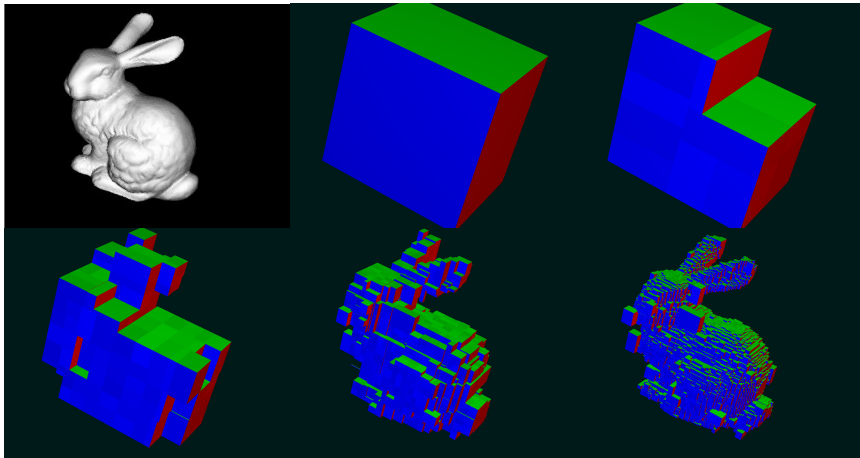


(i) Nobby visualized in ParaView.

Spatial data structures

- ▶ Grid.
 - ▶ Mapping from world space position to grid cell.
 - ▶ Key challenge: ray traversal of the grid.
- ▶ Oct tree (octree).
 - ▶ Using subdivision centers that subdivide space into eight octants.
 - ▶ Key challenge: book keeping with 8 child pointers per node.
- ▶ Bounding volume hierarchy (BVH)
 - ▶ Most often using axis-aligned bounding boxes (AABB).
 - ▶ Key challenge: traversal cost optimization.
- ▶ Binary space partitioning tree (BSP tree).
 - ▶ Most often restricted to axis-aligned planes (*kd tree*).
 - ▶ Key challenge: splitting plane selection.

Axis-aligned BSP tree (kd tree) example



The surface area heuristic (SAH) for selecting splitting planes:
$$\text{cost} = \text{SA}_{\text{left}}\Delta_{\text{left}} + \text{SA}_{\text{right}}\Delta_{\text{right}}, \quad \text{where } \Delta \text{ is triangle count.}$$

Reference

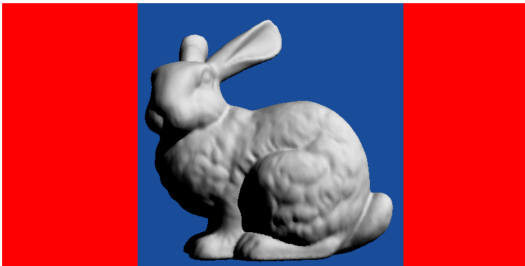
- Kammaje, R. P., and Mora, B. A study of restricted BSP trees for ray tracing. In *Proceedings of the IEEE Symposium on Interactive Ray Tracing (RT '07)*, pp. 55–62. 2007.

Looping



gpu: 345.6 ms

AABB



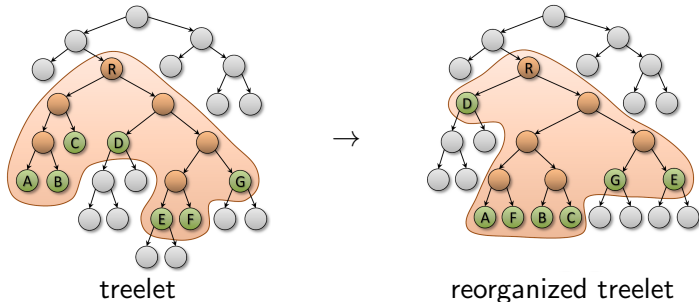
gpu: 234.2 ms

BSP tree



gpu: 8.3 ms

Treelet restructuring bounding volume hierarchy for spatial subdivision



► Practical GPU-based bounding volume hierarchy (BVH) builder.

1. Build a low-quality BVH (parallel linear BVH).
2. Optimize node topology by parallel treelet restructuring (keeping leaves and their subtrees intact).
3. Post-process for fast traversal.

Reference

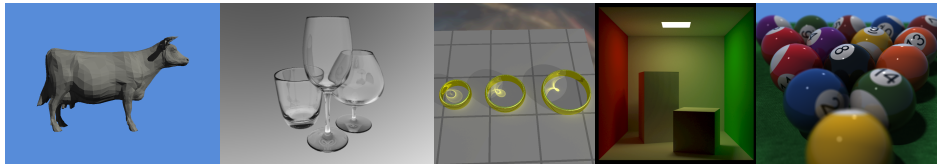
- Karras, T., and Aila, T. Fast parallel construction of high-quality bounding volume hierarchies. In *Proceedings of HPG 2013*, pp. 89–99. ACM, 2013.

Examples of Monte Carlo ray tracing on the GPU



<https://www.nvidia.com/en-us/geforce/community/demos/>

- Interactive ray tracing demos: meshes, glass, PPM, PT, Cook.



Reference

- Parker, S. G., Bigler, J., Dietrich, A., Friedrich, H., Hoberock, J., Luebke, D., McAllister, D., McGuire, M., Morley, K., Robison, A., and Stich, M. OptiX: a general purpose ray tracing engine. *ACM Transactions on Graphics* 29(4), Article 66. 2010.

Conclusion

- ▶ Physically based rendering aims to increase visual realism.
- ▶ Increased realism requires increasingly sophisticated models.
- ▶ This motivates the need for
 - ▶ richer models of light sources,
 - ▶ efficient spatial data structures for complex geometry,
 - ▶ fast-to-evaluate compact representations of high fidelity content,
 - ▶ methods for handling highly detailed spatial variation.
- ▶ **The candidate's thesis addresses aspects of these needs.**