Ray-Tracing on GPU: Light Sampling Algorithms

Dmitry Ivanov (divanov@correo.ugr.es)

19 de septiembre de 2024

Universidad de Granada

Dmitry Ivanov

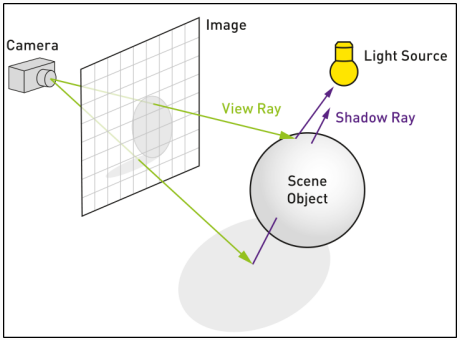
- B.Sc. degree in Information Technology at the **Belgorod Shukhov State Technology University**.
- M.Sc. candidate in Software Development at the **University of Granada**.
- Master's specialization in Visualization, Immersive Environments and Video Games.
- Software Engineer at **Institut Straumann**. CAD (Computer-Aided Design) and CAM (Computer-Aided Manufacturing) technologies in Orthodontics.

To my tutor, **Dr. Carlos Ureña Almagro**, who took me under his wing and whose profound knowledge and success motivated me to start and finish this project.

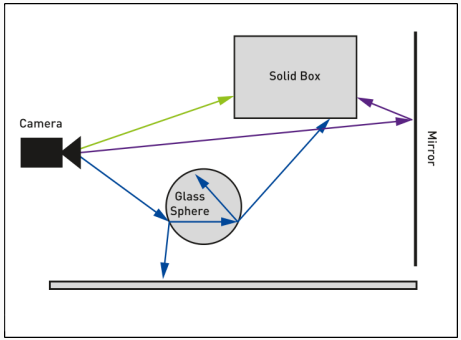
To my professor, **Dr. Alejandro José León Salas**, whose moral support and encouragement allowed me to proceed despite every obstacle I met.

To **all professors** at the **University of Granada** who motivate every student to put in additional efforts to reach perfect results.

1. Ray-Tracing Evolution.
2. Motivation and hardware improvements.
3. Goals.
4. Development of Ray-Tracing frameworks on CPU and GPU.
5. Ray-Tracing comparison. CPU vs. GPU.
6. Basic Light Sampling on GPU.
7. Conclusion.



Ray casting. The process of finding the closest object along a ray. *Source: "Ray tracing Gems I."* [Haines and Shirley, 2019]



Ray-Tracing uses the ray casting mechanism to recursively gather light contributions from reflective and refractive objects. *Source: "Ray tracing Gems I."* [Haines and Shirley, 2019]

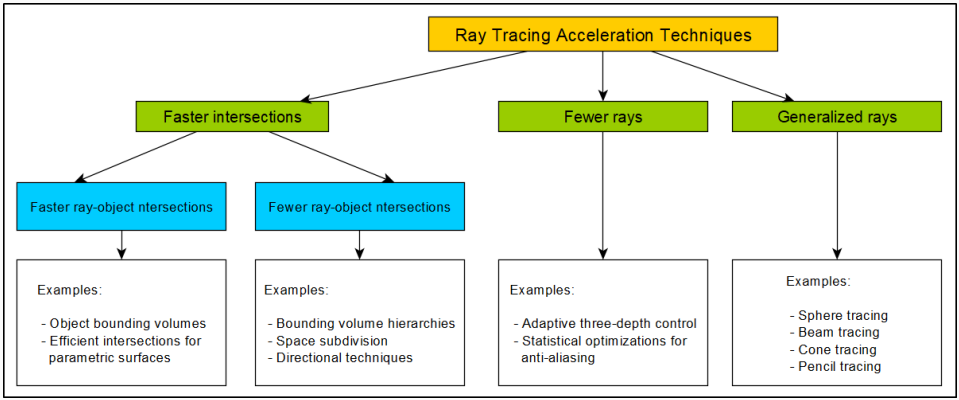
Ray-Tracing isn't a new technology. It has been evolving since the 1960s. Some of the most significant milestones are:

- [Appel, 1968]: Some Techniques for Shading Machine Renderings of Solids.
- [Whitted, 1980]: An Improved Illumination Model for Shaded Display.
- [Cook et al., 1984]: Distributed Ray-Tracing.
- [Goral et al., 1984]: Modeling the Interaction of Light Between Diffuse Surfaces.
- [Kajiya, 1986]: The Rendering Equation.
- [NVIDIA, 2009]: OptiX Ray-Tracing Engine.
- [Laine et al., 2013]: Wavefront Path Tracing on GPU
- [NVIDIA, 2018]: Turing RTX Architecture.
- [NVIDIA, 2022]: Deep learning super sampling (DLSS).

Ray-Tracing Performance

The weakest point of Ray-Tracing is its performance.

The weakest point of Ray-Tracing is its performance.



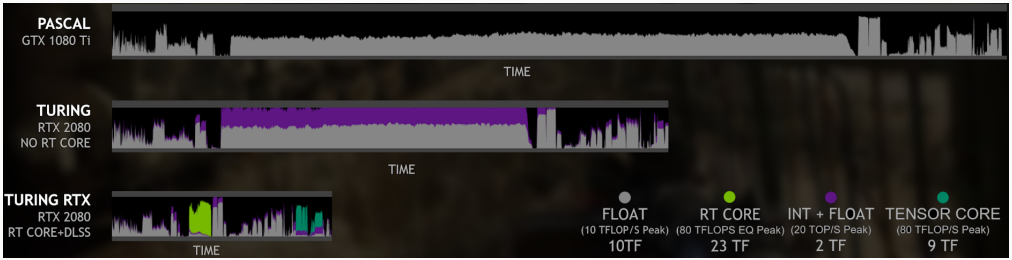
A broad classification of acceleration techniques. [Arvo and Kirk, 1989]

New innovations from **NVIDIA**:

- Dedicated Ray-Tracing hardware, **RT Cores**.
- High-performance mixed-precision computing hardware, **Tensor Cores**.

New innovations from **NVIDIA**:

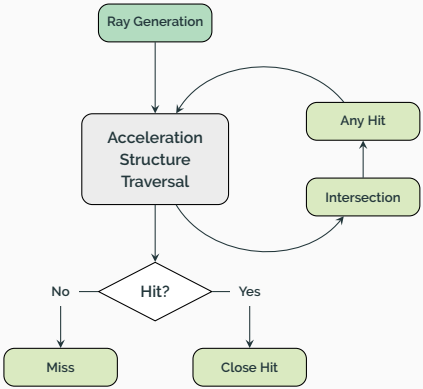
- Dedicated Ray-Tracing hardware, **RT Cores**.
- High-performance mixed-precision computing hardware, **Tensor Cores**.



GeForce RTX 2080 vs. GeForce GTX 1080 performance improvement. *Source: [NVIDIA]*:

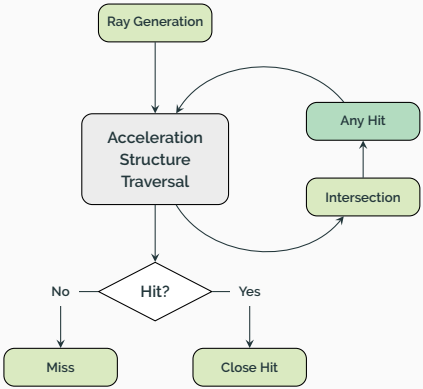
- **Ray Generation** shader. «Main loop» and entry point to the ray tracer.

FIVE NEW SHADER TYPES:



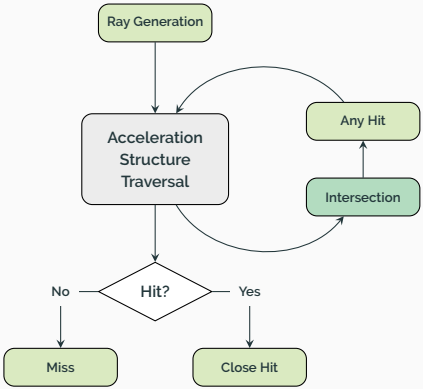
- **Ray Generation** shader. «Main loop» and entry point to the ray tracer.
- **Any Hit** shader. Runs once per any intersection found between a ray and an object. Typically used to determine transparency.

FIVE NEW SHADER TYPES:



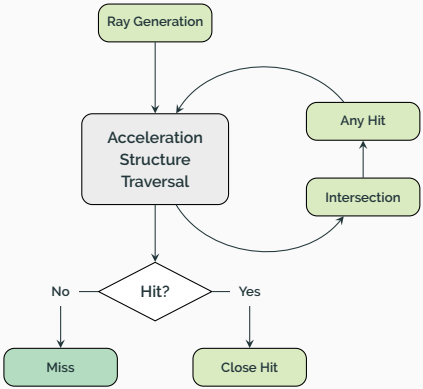
- **Ray Generation** shader. «Main loop» and entry point to the ray tracer.
- **Any Hit** shader. Runs once per any intersection found between a ray and an object. Typically used to determine transparency.
- **Intersection** shader. Defines how rays intersect geometry.

FIVE NEW SHADER TYPES:



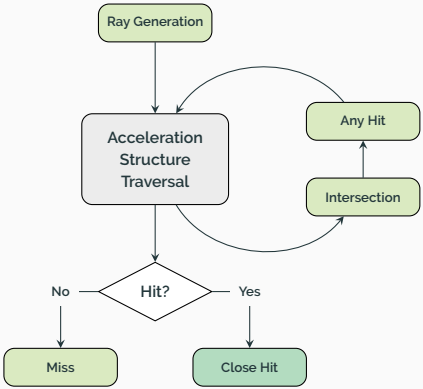
- **Ray Generation** shader. «Main loop» and entry point to the ray tracer.
- **Any Hit** shader. Runs once per any intersection found between a ray and an object. Typically used to determine transparency.
- **Intersection** shader. Defines how rays intersect geometry.
- **Miss** shader. Defines behavior when rays miss geometry.

FIVE NEW SHADER TYPES:



- **Ray Generation** shader. «Main loop» and entry point to the ray tracer.
- **Any Hit** shader. Runs once per any intersection found between a ray and an object. Typically used to determine transparency.
- **Intersection** shader. Defines how rays intersect geometry.
- **Miss** shader. Defines behavior when rays miss geometry.
- **Close Hit** shader. Runs once per ray to shade the final (closest) hit.

FIVE NEW SHADER TYPES:



- Investigate the state of the art in Ray-Tracing on the GPU.

- Investigate the state of the art in Ray-Tracing on the GPU.
- Design, implement and test the CPU Ray-Tracing framework.

- Investigate the state of the art in Ray-Tracing on the GPU.
- Design, implement and test the CPU Ray-Tracing framework.
- Design, implement and test the GPU Ray-Tracing framework.

- Investigate the state of the art in Ray-Tracing on the GPU.
- Design, implement and test the CPU Ray-Tracing framework.
- Design, implement and test the GPU Ray-Tracing framework.
- Evaluate the performance of the Ray-Tracing framework on different hardware (CPU or GPU).

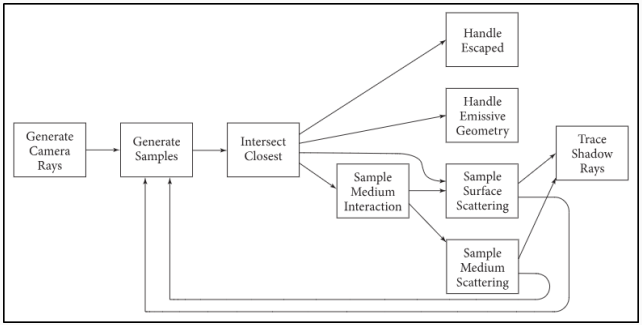
- Investigate the state of the art in Ray-Tracing on the GPU.
- Design, implement and test the CPU Ray-Tracing framework.
- Design, implement and test the GPU Ray-Tracing framework.
- Evaluate the performance of the Ray-Tracing framework on different hardware (CPU or GPU).
- Evaluate light sampling algorithms on the GPU.

- Investigate the state of the art in Ray-Tracing on the GPU.
- Design, implement and test the CPU Ray-Tracing framework.
- Design, implement and test the GPU Ray-Tracing framework.
- Evaluate the performance of the Ray-Tracing framework on different hardware (CPU or GPU).
- Evaluate light sampling algorithms on the GPU.
- Prepare basis for future research in the field of Ray-Tracing and Simulation.

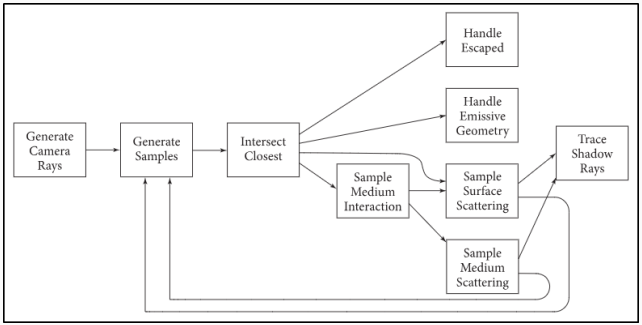
- [Christen, 2005]: attempted to use OpenGL and Direct3D shaders.

- [Christen, 2005]: attempted to use OpenGL and Direct3D shaders.
- [Parker et al., 2013]: NVIDIA OptiX engine with a programmable Ray-Tracing pipeline.

- [Christen, 2005]: attempted to use OpenGL and Direct3D shaders.
- [Parker et al., 2013]: NVIDIA OptiX engine with a programmable Ray-Tracing pipeline.
- [Laine et al., 2013]: **Wavefront Path Tracing** (In the picture below) on GPU.



- [Christen, 2005]: attempted to use OpenGL and Direct3D shaders.
- [Parker et al., 2013]: NVIDIA OptiX engine with a programmable Ray-Tracing pipeline.
- [Laine et al., 2013]: **Wavefront Path Tracing** (In the picture below) on GPU.



- [Knoll et al., 2019] and [Meister et al., 2020]: RTX-based Ray-Tracing improvements.

To develop the Ray-Tracing using Vulkan, we need to have an RTX-compatible GPU. The following hardware and software were used:

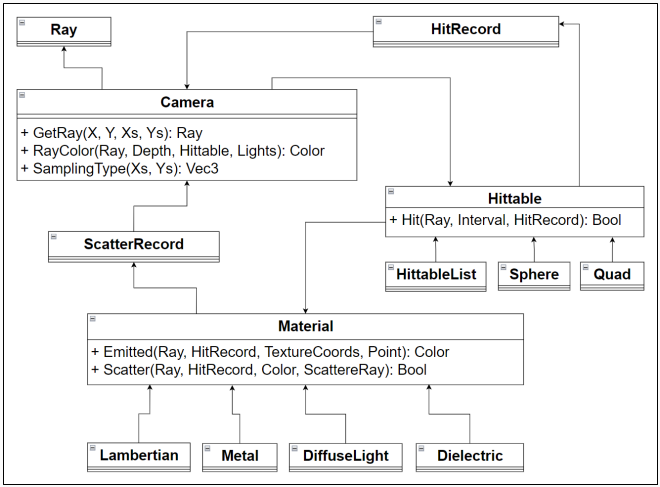
- **System**: XMG Neo 17 (Laptop)
- **OS**: Windows 10
- **CPU**: Intel i9-14900 HX
- **RAM**: 64 GB
- **GPU**: NVIDIA GeForce RTX 4070 (with Ray-Tracing cores)
- **GPU Ram**: 8 GB
- **Compiler**: MSVC 19.41 (Visual Studio 2022)
- **CMake**: 3.30.2
- **DirectX Support**: 12.1

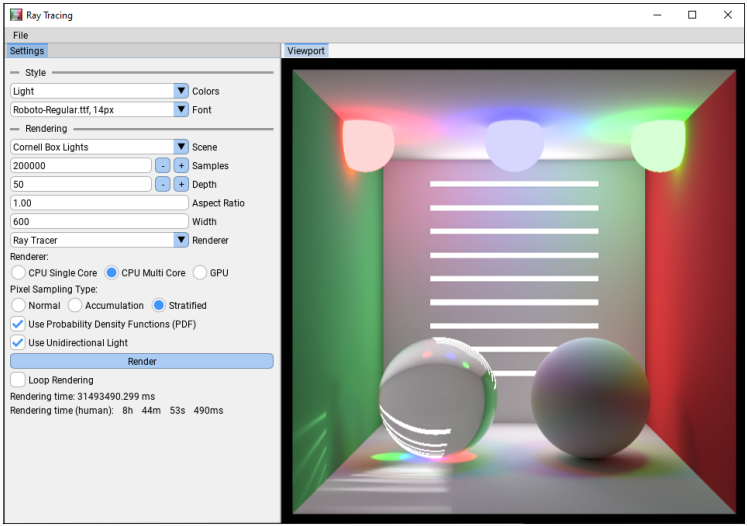
We use VCPKG to manage the third-party libraries in the project and support all popular platforms: Windows, Linux, and macOS. The libraries used in the project are:

- **C++**: main programming language.
- **GLSL**: shading language.
- **Vulkan SDK**: official Vulkan SDK.
- **GLFW**: window creation and input handling.
- **GLM**: mathematics library for graphics software.
- **Dear ImGui with Docking**: graphical user interface library.
- **Spdlog**: fast C++ logging library.
- **Stb**: single-file public domain image loader for C/C++.
- **Tinyobjloader**: tiny but powerful single-file object loader.

Ray-Tracing framework by Peter Shirley (In One Weekend series).

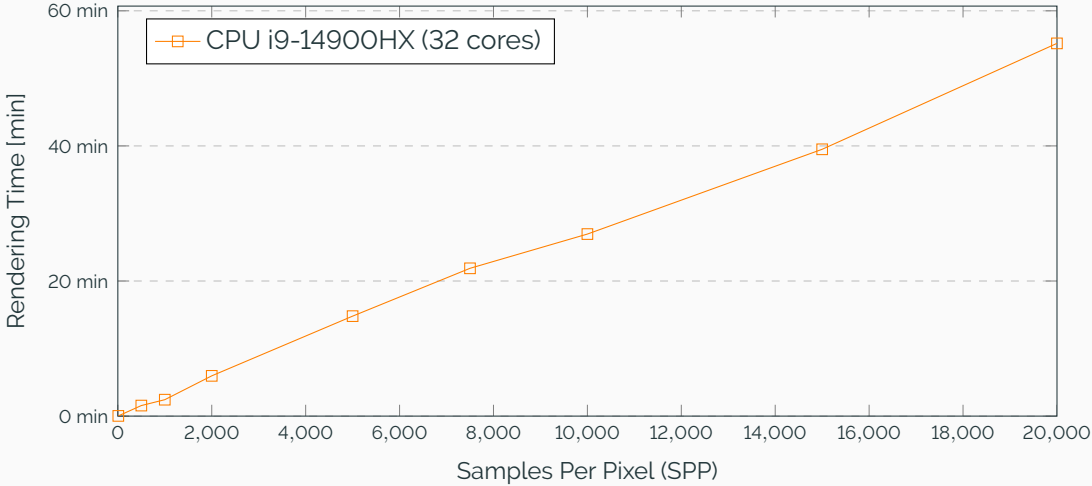
- Camera.
- Hittable - object.
- Hittable List - group of objects.
- Material.
- Ray.
- Hit Record - hit properties.
- Scatter Record - ray behavior.




Ray Tracing
File
Settings
Viewport
Style
Light
Colors
Roboto-Regular.ttf, 14px
Font
Rendering
Cornell Box Lights
Scene
200000
Samples
50
Depth
1.00
Aspect Ratio
600
Width
Ray Tracer
Renderer
Renderer:
CPU Single Core
CPU Multi Core
GPU
Pixel Sampling Type:
Normal
Accumulation
Stratified
Use Probability Density Functions (PDF)
Use Unidirectional Light
Render
Loop Rendering
Rendering time: 31493490.299 ms
Rendering time (human): 8h 44m 53s 490ms

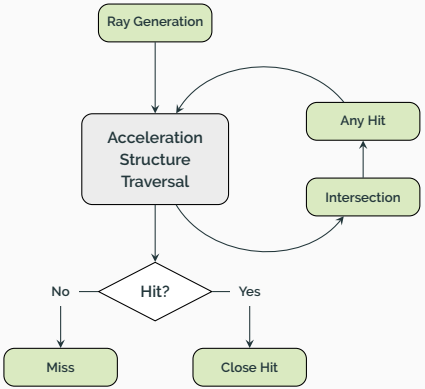
Samples per pixel	Rendering Time CPU (ms)	Rendering Time CPU (human)
10	2'000 ms	2s
50	9'000 ms	9s
100	19'000 ms	19s
250	42'000 ms	42s
500	94'000 ms	1m 34s
1000	146'000 ms	2m 26s
2000	358'000 ms	5m 58s
5000	888'000 ms	14m 48s
7500	1'313'000 ms	21m 53s
10000	1'617'000 ms	26m 57s
15000	2'370'000 ms	39m 30s
20000	3'311'000 ms	55m 11s

Ray Tracing performance on CPU.


CPU i9-14900HX (32 cores)
60 min
40 min
20 min
0 min
Rendering Time [min]
0
2,000
4,000
6,000
8,000
10,000
12,000
14,000
16,000
18,000
20,000
Samples Per Pixel (SPP)

- **Ray Generation Group**: Ray Generation shader.
- **Miss Group**: Miss shader (background).
- **Close Hit Group for Triangular Data**: Close Hit shader (meshes).
- **Close Hit Group for Non-Triangular Data**: Close Hit and Intersection shaders (procedural objects).

RAY-TRACING SHADERS

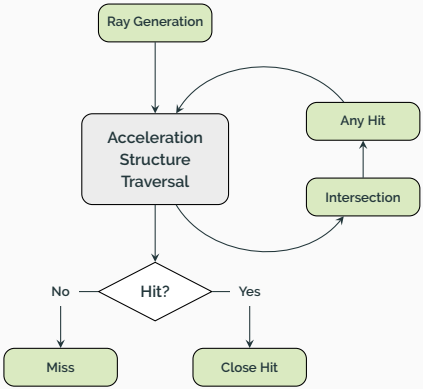


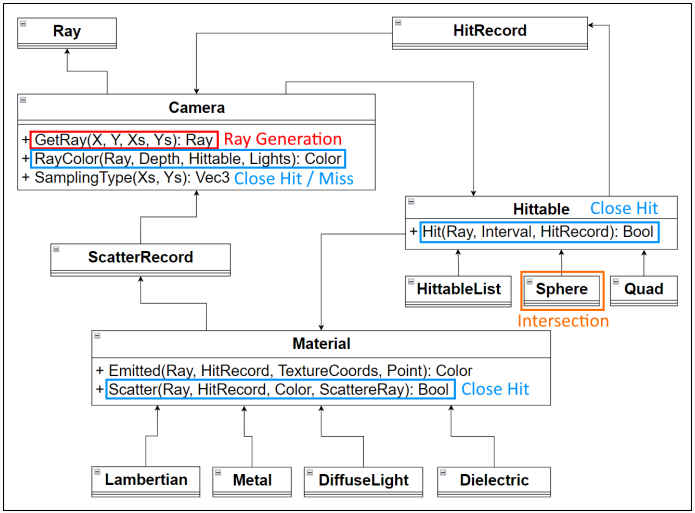
Side Note: Any Hit shader can typically be used to efficiently implement alpha-testing (transparency) and shadow rays. It is not used in this project, so we don't have it in the pipeline.

Mapping CPU classes to Ray-Tracing shaders

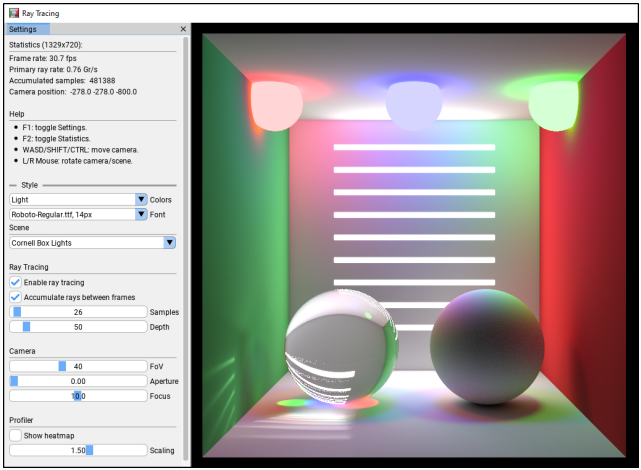
- Camera.GetRay ⇒ Ray Generation
- Hittable.Hit (Sphere.Hit) ⇒ Intersection
- Camera.RayColor ⇒ Close Hit/Miss
- Material.Scatter ⇒ Close Hit
- HitRecord ⇒ Ray Payload
- ScatterRecord ⇒ Ray Payload

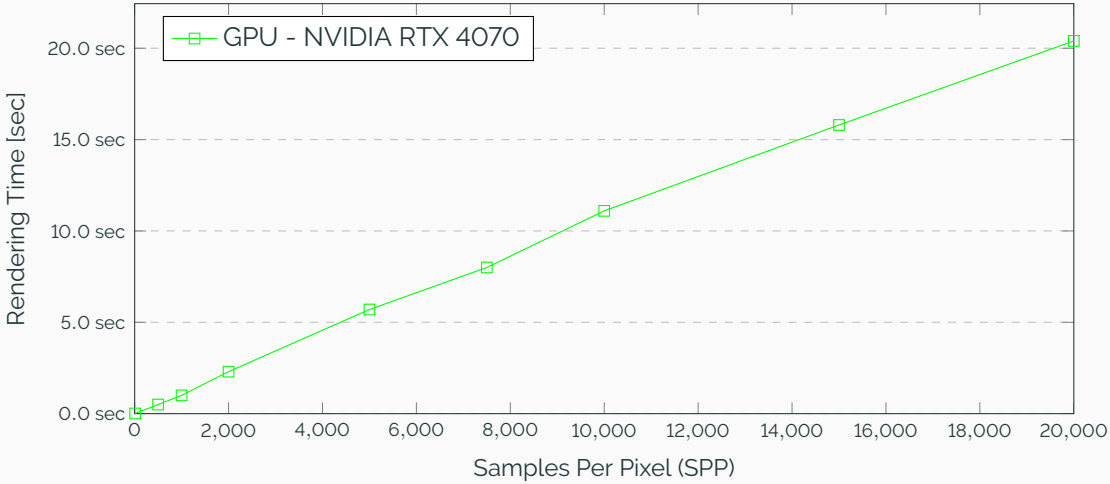
RAY-TRACING SHADERS

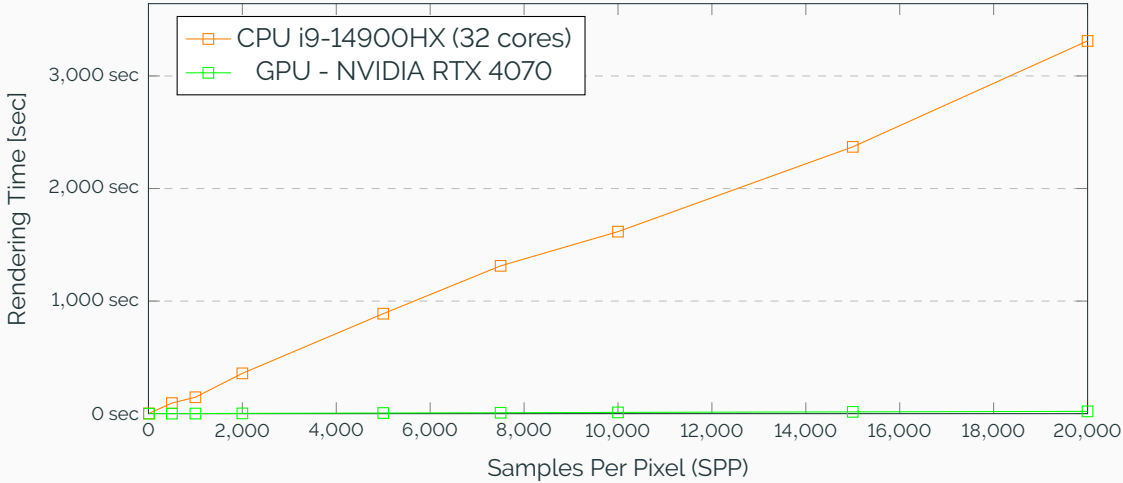



Ray
HitRecord
Camera
+ GetRay(X, Y, Xs, Ys): Ray
Ray Generation
+ RayColor(Ray, Depth, Hittable, Lights): Color
+ SamplingType(Xs, Ys): Vec3
Close Hit / Miss
Hittable
Close Hit
+ Hit(Ray, Interval, HitRecord): Bool
ScatterRecord
HittableList
Sphere
Quad
Intersection
Material
+ Emitted(Ray, HitRecord, TextureCoords, Point): Color
+ Scatter(Ray, HitRecord, Color, ScattereRay): Bool
Close Hit
Lambertian
Metal
DiffuseLight
Dielectric

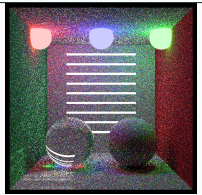
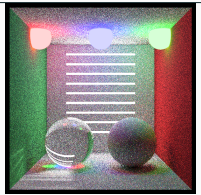
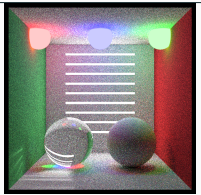
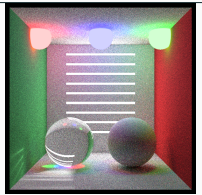
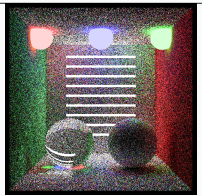
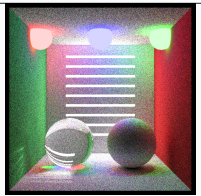
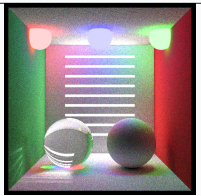
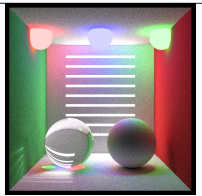
On the test scene (Cornell Box with multiple light sources) we can see Caustics effect, coloured shadows and refraction in the right places. The GPU implementation matches the CPU implementation visually.



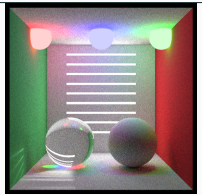
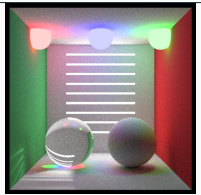
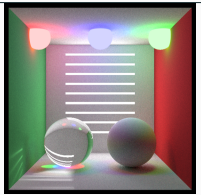
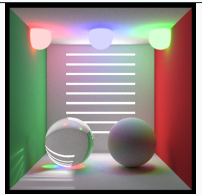
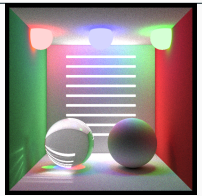
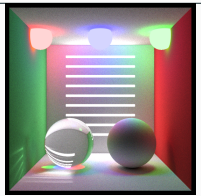
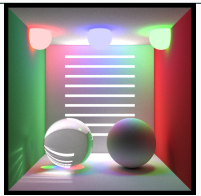
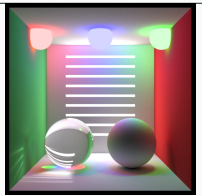

GPU - NVIDIA RTX 4070
Rendering Time [sec]
20.0 sec
15.0 sec
10.0 sec
5.0 sec
0.0 sec
0
2,000
4,000
6,000
8,000
10,000
12,000
14,000
16,000
18,000
20,000
Samples Per Pixel (SPP)



The immediate drop in rendering time on the GPU. The rendering time is still linear.

	10 SPP	100 SPP	250 SPP	500 SPP
CPU				
GPU				

The quality of the ray traced images rendered by CPU and GPU (part 1). SPP - Samples Per Pixel.

	1000 SPP	2000 SPP	5000 SPP	10000 SPP
CPU				
GPU				

The quality of the ray traced images rendered by CPU and GPU (part 2). SPP - Samples Per Pixel.

DEMO

https://www.youtube.com/watch?v=koZc1S7LPkQ

Time to implement light sampling on the GPU. We need to make the following improvements:

- Introduce the probability density functions **(PDF)** for light sampling. The PDF is used to describe the likelihood of a light ray hitting a specific point on the object's surface.

Time to implement light sampling on the GPU. We need to make the following improvements:

- Introduce the probability density functions **(PDF)** for light sampling. The PDF is used to describe the likelihood of a light ray hitting a specific point on the object's surface.
- Add new field to **Ray Payload**. PDF value for geometries, PDF skipping flag for refraction, and scattered PDF values for materials.

Time to implement light sampling on the GPU. We need to make the following improvements:

- Introduce the probability density functions **(PDF)** for light sampling. The PDF is used to describe the likelihood of a light ray hitting a specific point on the object's surface.
- Add new field to **Ray Payload**. PDF value for geometries, PDF skipping flag for refraction, and scattered PDF values for materials.
- Change **Scatter functions** to support the light sampling. In the Lambertian scattering model, we need to calculate the scattering PDF and pass the PDF value from the emitting geometry.

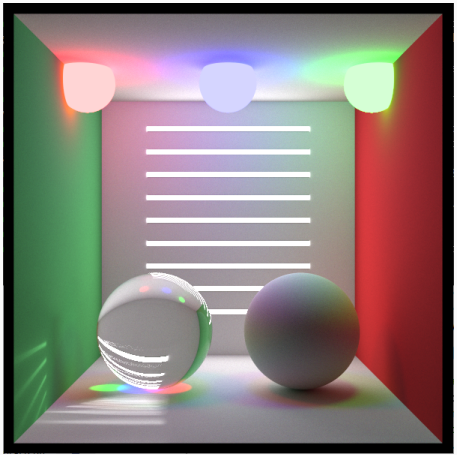
Time to implement light sampling on the GPU. We need to make the following improvements:

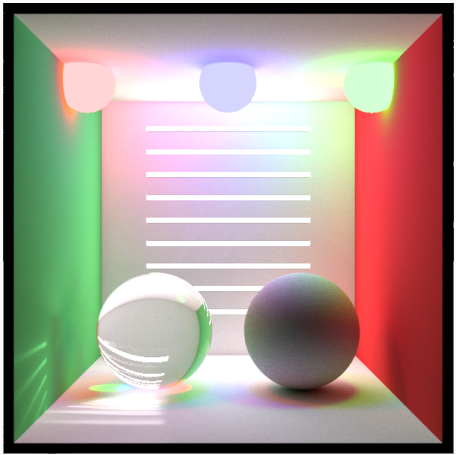
- Introduce the probability density functions **(PDF)** for light sampling. The PDF is used to describe the likelihood of a light ray hitting a specific point on the object's surface.
- Add new field to **Ray Payload**. PDF value for geometries, PDF skipping flag for refraction, and scattered PDF values for materials.
- Change **Scatter functions** to support the light sampling. In the Lambertian scattering model, we need to calculate the scattering PDF and pass the PDF value from the emitting geometry.
- Change **Ray Generation** shader to process PDF values from the scattering functions and geometries.

Time to implement light sampling on the GPU. We need to make the following improvements:

- Introduce the probability density functions **(PDF)** for light sampling. The PDF is used to describe the likelihood of a light ray hitting a specific point on the object's surface.
- Add new field to **Ray Payload**. PDF value for geometries, PDF skipping flag for refraction, and scattered PDF values for materials.
- Change **Scatter functions** to support the light sampling. In the Lambertian scattering model, we need to calculate the scattering PDF and pass the PDF value from the emitting geometry.
- Change **Ray Generation** shader to process PDF values from the scattering functions and geometries.
- Improve **Close Hit** shaders, for procedural and triangle meshes, to calculate the PDF value for a specific emitting light source.

Random scattering on a hemisphere without PDF scattering (left) and with PDF scattering (right).





- We created two independent software systems for testing various Ray-Tracing technologies on CPU and GPU.
- The Ray-Tracing framework on the GPU is significantly faster than the CPU version.
- The image quality is comparable between the CPU and GPU versions. The main requirement: implement similar pixel sampling algorithms.
- Ray-Tracing on the GPU is already a mature technology, and it is possible to achieve real-time Ray-Tracing with the algorithmic improvements.

The source code of the Ray-Tracing frameworks on CPU and GPU and all supplementary materials are available on the public GitHub repository and published under the Open Source MIT License at the following URLs:

- `https://github.com/d-k-ivanov/ray-tracing`
- `https://github.com/d-k-ivanov/ray-tracing/blob/main/Docs/tfm.pdf`
- `https://github.com/d-k-ivanov/ray-tracing/blob/main/Docs/tfm-presentation.pdf`

In case of any issues or questions, the author can be contacted via emails **d.k.ivanov (at) live.com** or **divanov (at) correo.ugr.es**.

- Extend the Ray-Tracing framework with more functionality, including UI and better OOP structure.

- Extend the Ray-Tracing framework with more functionality, including UI and better OOP structure.
- Merge the Ray-Tracing on CPU framework with the Ray-Tracing on GPU framework and refactor architecture of shader functions.

- Extend the Ray-Tracing framework with more functionality, including UI and better OOP structure.
- Merge the Ray-Tracing on CPU framework with the Ray-Tracing on GPU framework and refactor architecture of shader functions.
- More shapes, sampling algorithms and scene setups.

- Extend the Ray-Tracing framework with more functionality, including UI and better OOP structure.
- Merge the Ray-Tracing on CPU framework with the Ray-Tracing on GPU framework and refactor architecture of shader functions.
- More shapes, sampling algorithms and scene setups.
- Add real-time denoisers support.

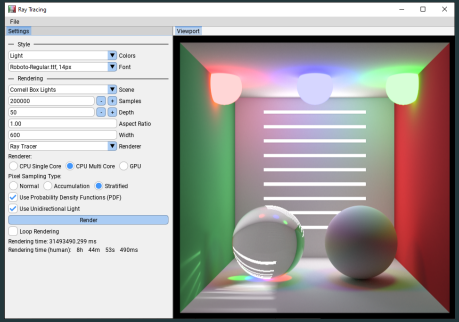
- Extend the Ray-Tracing framework with more functionality, including UI and better OOP structure.
- Merge the Ray-Tracing on CPU framework with the Ray-Tracing on GPU framework and refactor architecture of shader functions.
- More shapes, sampling algorithms and scene setups.
- Add real-time denoisers support.
- Add support for spatio-temporal resampling and Deep Learning Super Sampling (DLSS) for real-time Ray-Tracing.

- Extend the Ray-Tracing framework with more functionality, including UI and better OOP structure.
- Merge the Ray-Tracing on CPU framework with the Ray-Tracing on GPU framework and refactor architecture of shader functions.
- More shapes, sampling algorithms and scene setups.
- Add real-time denoisers support.
- Add support for spatio-temporal resampling and Deep Learning Super Sampling (DLSS) for real-time Ray-Tracing.
- Split the Ray-Tracing evaluation framework into step-by-step modules for partial usage in education.

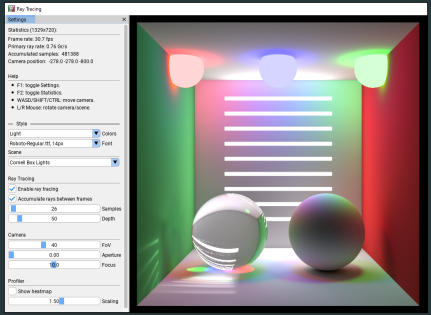
- Extend the Ray-Tracing framework with more functionality, including UI and better OOP structure.
- Merge the Ray-Tracing on CPU framework with the Ray-Tracing on GPU framework and refactor architecture of shader functions.
- More shapes, sampling algorithms and scene setups.
- Add real-time denoisers support.
- Add support for spatio-temporal resampling and Deep Learning Super Sampling (DLSS) for real-time Ray-Tracing.
- Split the Ray-Tracing evaluation framework into step-by-step modules for partial usage in education.
- Add interactive scene editor to add, remove, and modify scene objects.

- Extend the Ray-Tracing framework with more functionality, including UI and better OOP structure.
- Merge the Ray-Tracing on CPU framework with the Ray-Tracing on GPU framework and refactor architecture of shader functions.
- More shapes, sampling algorithms and scene setups.
- Add real-time denoisers support.
- Add support for spatio-temporal resampling and Deep Learning Super Sampling (DLSS) for real-time Ray-Tracing.
- Split the Ray-Tracing evaluation framework into step-by-step modules for partial usage in education.
- Add interactive scene editor to add, remove, and modify scene objects.
- Add Material Definition Language (MDL) support and concepts from Physically Based Rendering (PBR).

Questions?



Ray-Tracing on CPU.



Ray-Tracing on GPU.