

Packet Ray Tracing

with the
ARM NEON Architecture

Gustaf Waldemarson

Faculty of Engineering (LTH), Lund University

ARM Sweden AB

May 17, 2026



◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ≡ | ≡ ↺ 🔍 ↻

↻ ↺

2



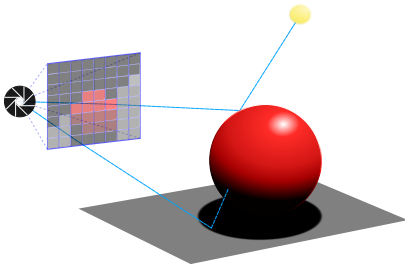
100
 100
 100
 100
 100

◀ ▶

◀

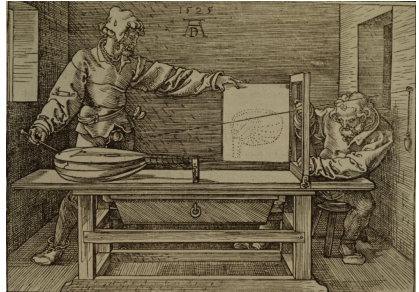
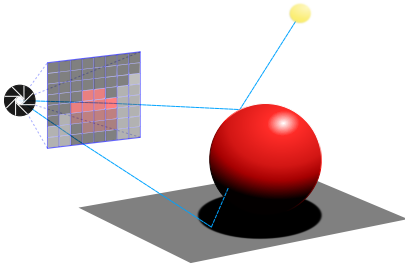
Ray Tracing

Some history — Ray Casting



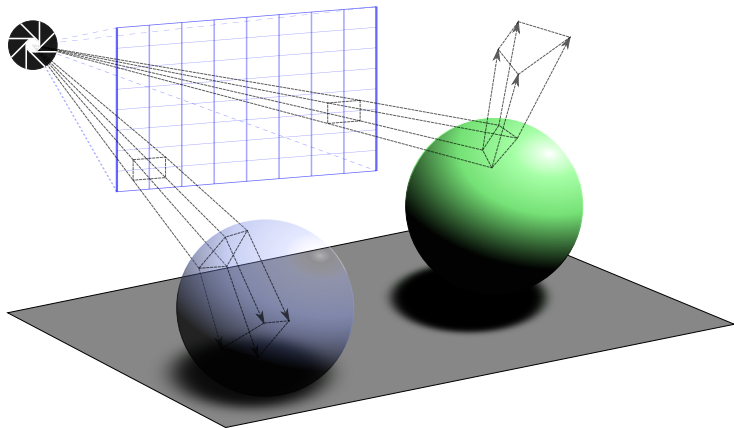
Ray Tracing

Some history — Ray Casting

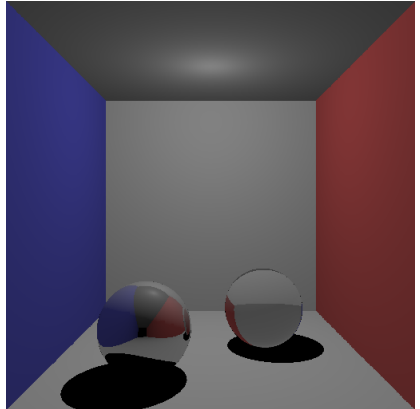
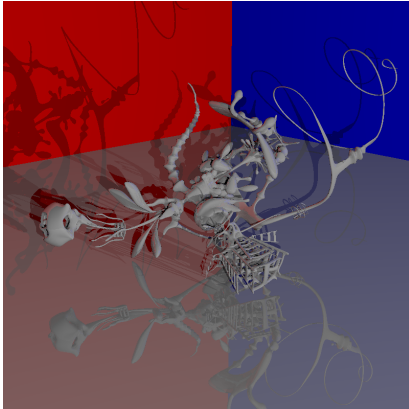


Ray casting is a very old method!

Whitted Style Ray Tracing



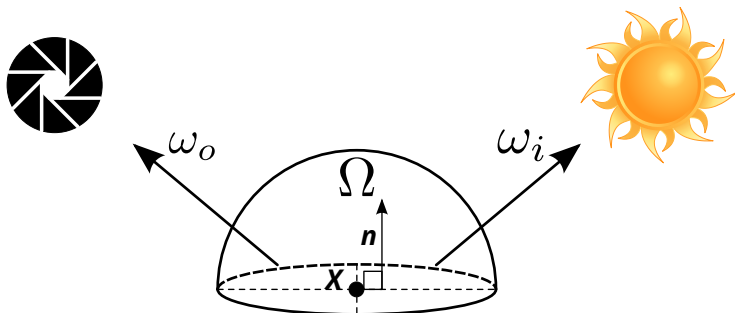
Whitted Style Ray Tracing



The Rendering Equation

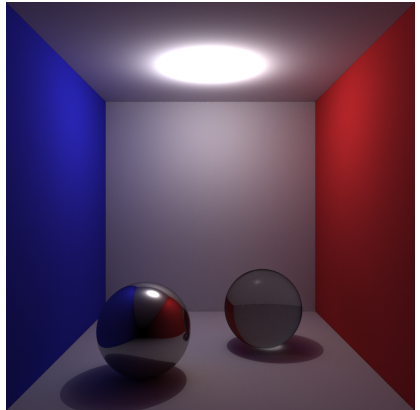
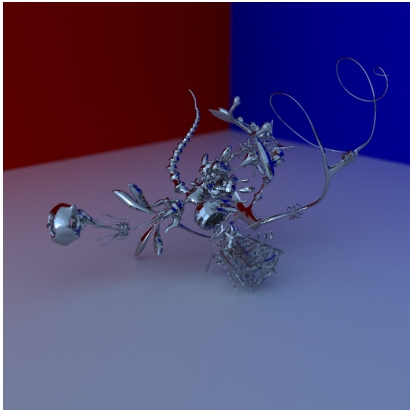
Path Tracing and the Rendering Equation

$$L_o(\mathbf{x}, \omega_o) = L_e(\mathbf{x}, \omega_o) + \int_{\Omega} f_r(\mathbf{x}, \omega_i, \omega_o) L_i(\mathbf{x}, \omega_i) (\omega_i \cdot \mathbf{n}) d\omega_i$$



Path Tracing

Path Tracing and the Rendering Equation



Ray Tracing Complexity

Acceleration Data Structures

$$\mathcal{O}(\#Rays \cdot \#Primitives)$$

Ray Tracing Complexity

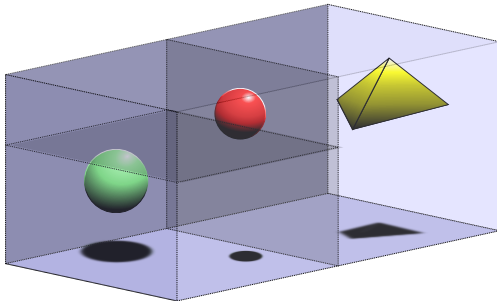
Acceleration Data Structures

$$\mathcal{O}(\#Rays \cdot \#Primitives) \implies \mathcal{O}(\#Rays \cdot \log(\#Primitives))$$

Bounding Volume Hierarchy

Acceleration Data Structures

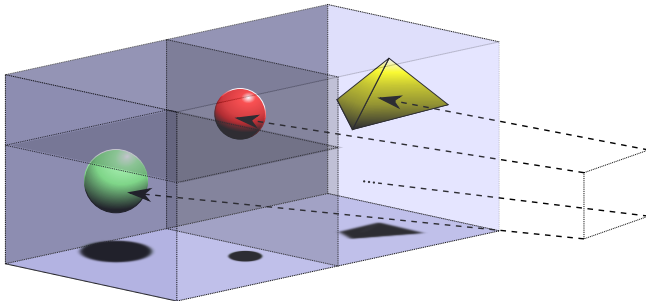
$$\mathcal{O}(\#Rays \cdot \#Primitives) \implies \mathcal{O}(\#Rays \cdot \log(\#Primitives))$$



Bounding Volume Hierarchy

Acceleration Data Structures

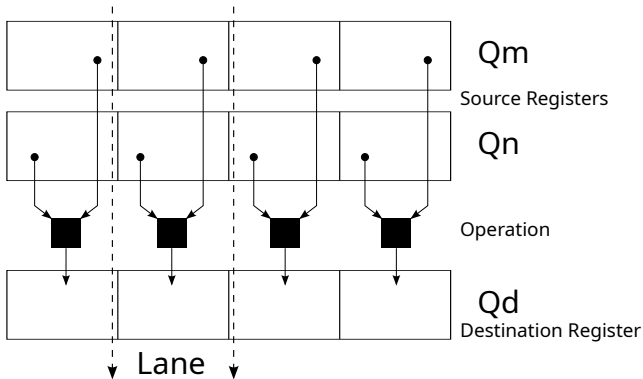
$$\mathcal{O}(\#Rays \cdot \#Primitives) \implies \mathcal{O}(\#Rays \cdot \log(\#Primitives))$$



Vector Processing

Also a very old idea...

A typical SIMD operation



Different SIMD Architectures

Single Instruction Multiple Data

- | | |
|-------------|---------------------------------------|
| VIS | – Sparc (Visual Instruction Set) |
| MDMX | – Mips (Mips Digital Media eXtension) |
| MMX,SSE,AVX | – x86 |
| AltiVec,VSX | – PowerPC |

Limitations

Single Instruction Multiple Data

- Conversion to vector/batch form.
- Alignment constraint.
- Not all applications can utilize it.

ARM NEON

Implementation

NEON™

The SIMD Library

Implementation

- Vector operations library.
- Uses NEON compiler intrinsics.

Custom Vector Types

The SIMD Library

neon4f

4 floats

neon4i

4 integers

neon4b

4 “booleans”

Structure of Arrays

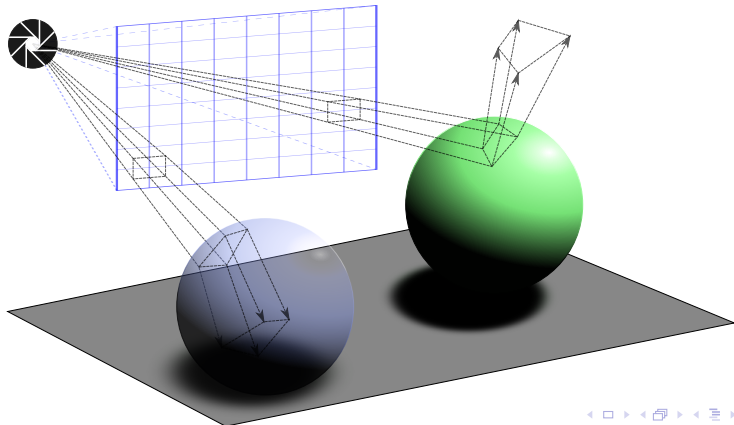
Implementation

```
struct Vec4x3 {  
    neon4f x;  
    neon4f y;  
    neon4f z;  
};
```

x	x_0	x_1	x_2	x_3
y	y_0	y_1	y_2	y_3
z	z_0	z_1	z_2	z_3

Packet Tracing

Implementation

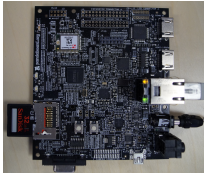


The Ray Tracer(s) – Not one, but four of them

Implementation

- Scalar Single-Ray Tracer
- Scalar Packet Tracer
- Neon Single-Ray Tracer
- Neon Packet Tracer

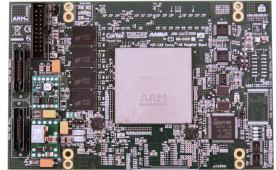
Test Platforms



Pandaboard



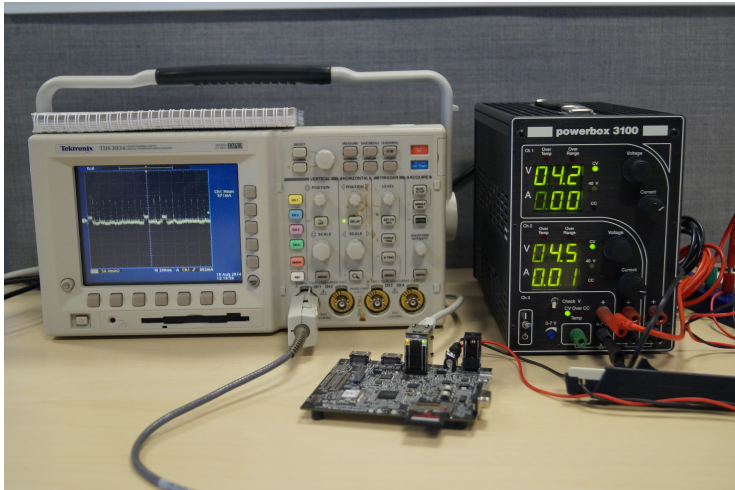
Chromebook



CoreTile Express

Measuring Energy Consumption

Pandaboard



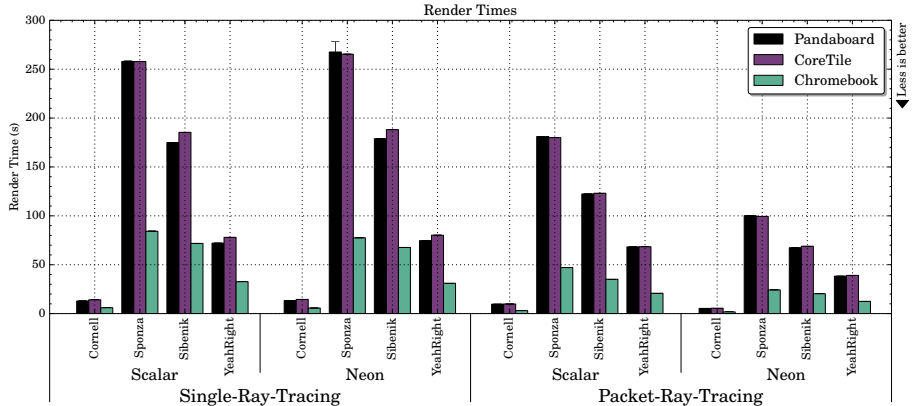
Measuring Energy Consumption

Chromebook

```
/sys/  
|-- class  
|   |-- ata_device  
|   |-- ...  
|   |-- graphics  
|   |-- ...  
|   |-- power_supply  
|       |-- ADP1  
|       |-- BAT1  
|           |-- current_now  
|           |-- ...  
|           |-- charge_now  
|           |-- manufacturer  
|           |-- model_name  
|           |-- ...  
|           |-- voltage_now  
|   |-- ...
```

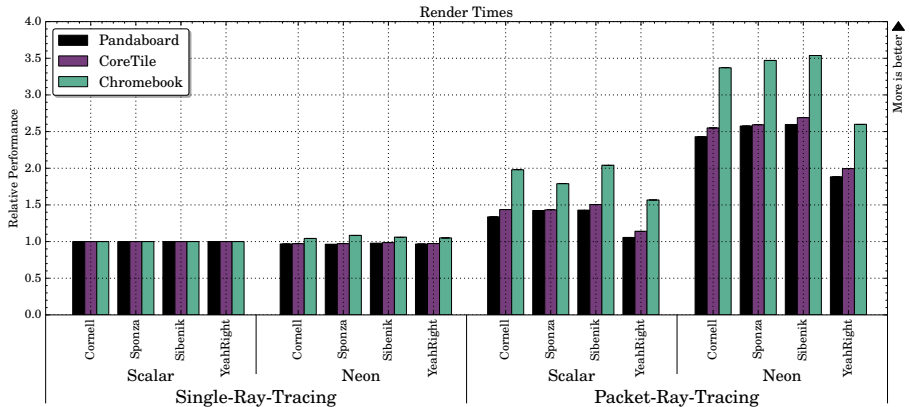
Absolute Render Times

Results



Relative Render Times

Results



Render Times Discussion

Results

- Why is the Chromebook 50-75% better than the Pandaboard?

Render Times Discussion

Results

- Why is the Chromebook 50-75% better than the Pandaboard?
- Because of the SIMD width!

Absolute Energy Consumption

Pandaboard

Scene	Single-Ray-Tracing		Packet-Ray-Tracing		
	Scalar	Neon	Scalar	Neon	
Cornell	17	13	14	7	J
Sponza	266	238	182	105	J
Sibenik	164	163	128	75	J
YeahRight	75	74	75	46	J

Relative Energy Consumption

Pandaboard

Scene	Single-Ray-Tracing		Packet-Ray-Tracing	
	Scalar	Neon	Scalar	Neon
Cornell	1.00	1.27	1.23	2.44
Sponza	1.00	1.12	1.46	2.52
Sibenik	1.00	1.00	1.28	2.19
YeahRight	1.00	1.02	1.00	1.64

Absolute Energy Consumption

Samsung Chromebook

Scene	Single-Ray-Tracing		Packet-Ray-Tracing		
	Scalar	Neon	Scalar	Neon	
Cornell	57	53	39	27	J
Sponza	696	656	444	258	J
Sibenik	349	334	218	138	J
YeahRight	275	261	217	137	J

Relative Energy Consumption

Samsung Chromebook

Scene	Single-Ray-Tracing		Packet-Ray-Tracing	
	Scalar	Neon	Scalar	Neon
Cornell	1.00	1.08	1.46	2.10
Sponza	1.00	1.06	1.57	2.69
Sibenik	1.00	1.04	1.60	2.52
YeahRight	1.00	1.05	1.27	2.00

Conclusion

- Packet-Tracing greatly improves the performance for Whitted style ray tracing.

Future Work

- **Path Tracing** – The accurate, but slow and ray-incoherent approach to global illumination.
- **Instant Global Illumination** – A faster and more ray-coherent approach, but less accurate.
- Study the generated **assembly** more thoroughly, or re-write it entirely in assembler.
- ARMv8 Architecture, what may be improved?

Discussion

So what will the machines do?

The End

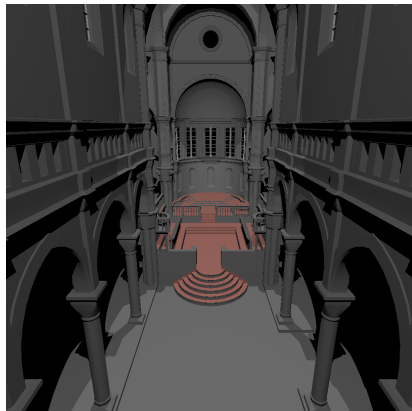
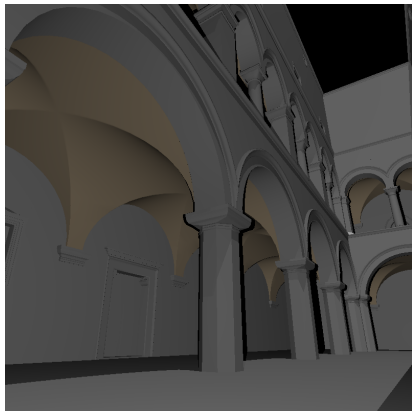
References for the Interested

[smallpt](#) A minimal C++ Path Tracer in 99 lines.

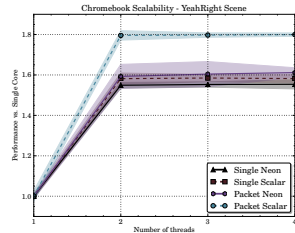
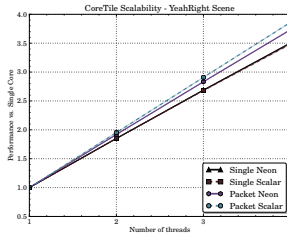
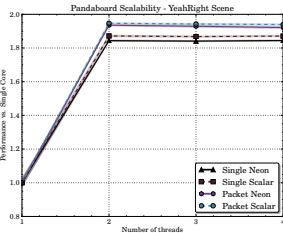
[PBRT](#) Physically Based Rendering – An open source (MIT) and extremely flexible ray tracer.

[Ingo Wald](#) PhD Thesis from 2004, but still rather up-to-date.

Sibenik and Sponza



Scalability



The Ray Tracing Image Algorithm

```
Initialize scene;
for each pixel do
|   create a ray for the given pixel;
|   for each object in scene do
|   |   if ray intersected an object then
|   |   |   set pixel color to the color of the object;
|   |   else
|   |   |   set pixel color to black
|   |   end
|   end
end
```

Algorithm 1: How to ray trace an image.