Design and Implementation of a Parallel Graphics Rendering Engine

Kyuhyeon Nam , Ke Wang
EE/CSCI 451: Project Presentation
University of Southern California

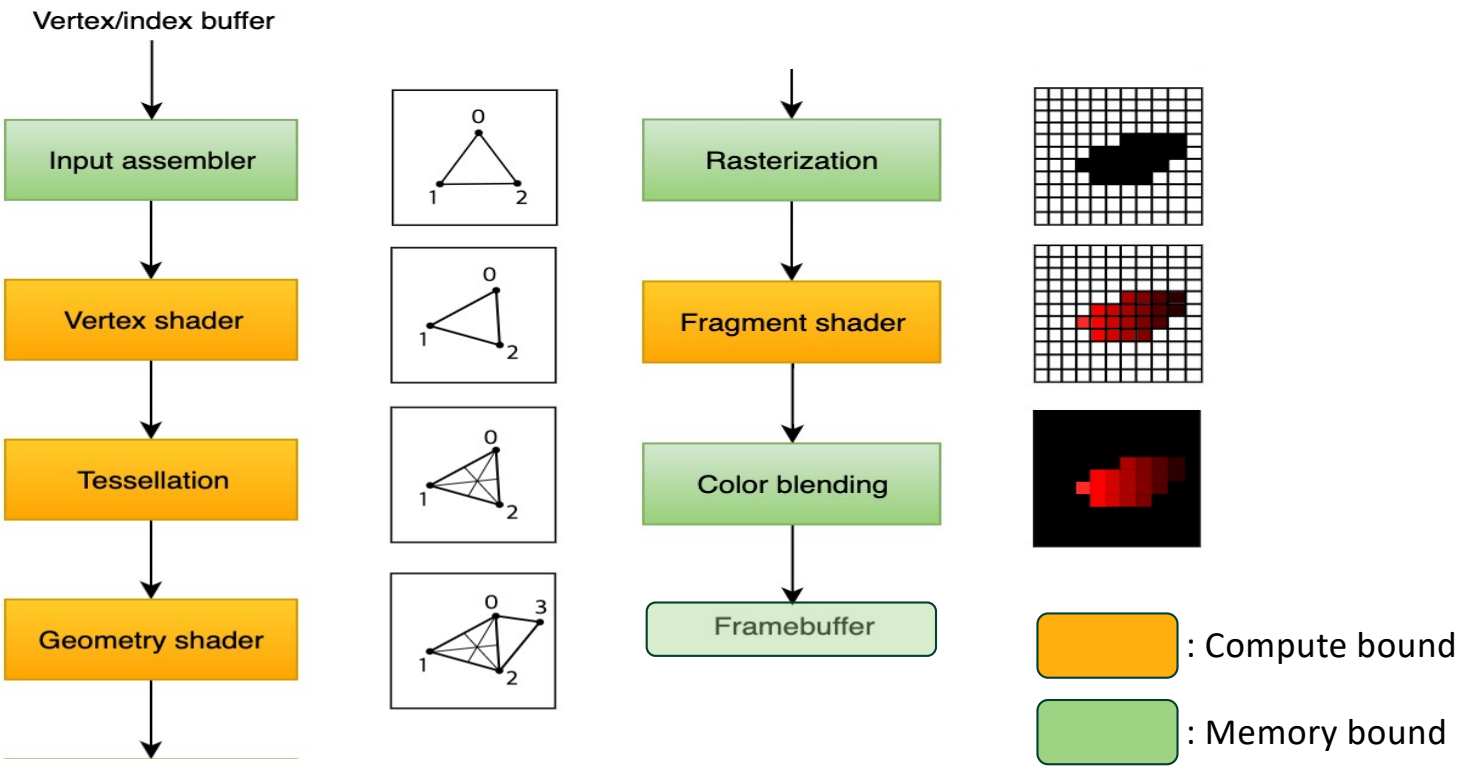
Introduction

<Early graphics>

<Modern graphics>

Context

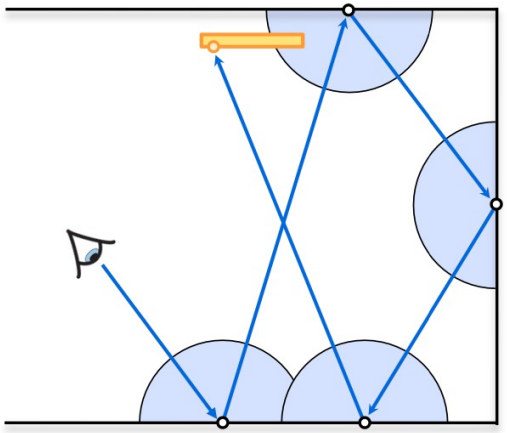
- Rendering pipeline



Context

- “Path Tracing” for realistic

<Rendering Equation>



$$L(\mathbf{x},\vec{\omega}) = L_e(\mathbf{x},\vec{\omega}) + \int_{H^2} f_r(\vec{\mathbf{x}},\vec{\omega}',\vec{\omega}) L(r(\mathbf{x},\vec{\omega}'),-\vec{\omega}') \cos\theta' \mathrm{d}\vec{\omega}'$$

$$L(\mathbf{x},\vec{\omega}) \approx L_e(\mathbf{x},\vec{\omega}) + \frac{f_r(\vec{\mathbf{x}},\vec{\omega}',\vec{\omega}) L(r(\mathbf{x},\vec{\omega}'),-\vec{\omega}') \cos\theta'}{p(\vec{\omega}')}$$

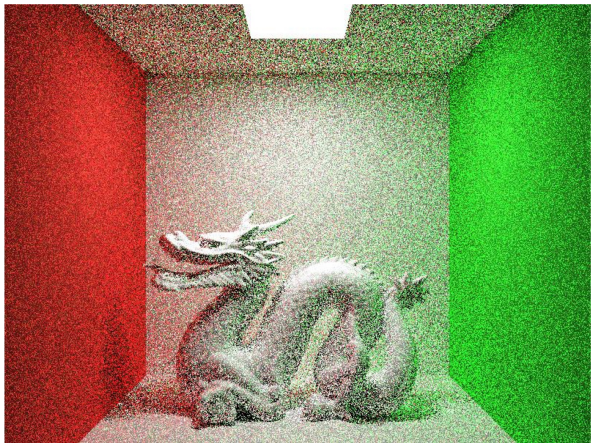
<Path Tracing Algorithm>

$$L(\mathbf{x},\vec{\omega}) = L_e(\mathbf{x},\vec{\omega}) + L_r(\mathbf{x},\vec{\omega})$$

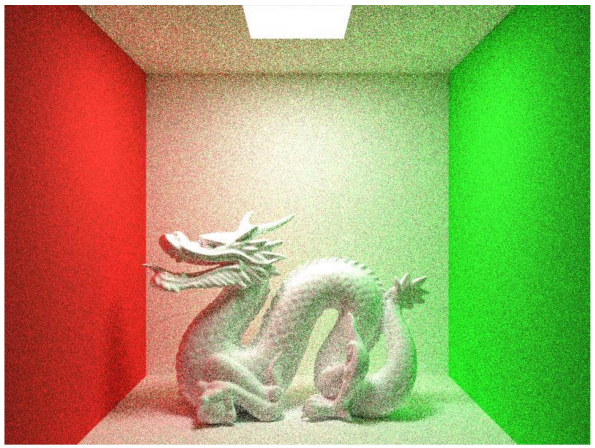
```
Color color(Point x, Direction ω, int moreBounces):

    if not moreBounces:
        return Le(x,-ω)

    // sample recursive integral
    ω' = sample from BRDF
    return Le(x,-ω) + BRDF * color(trace(x, ω'), moreBounces-1) * dot(n, ω') / pdf(ω')
```

Context

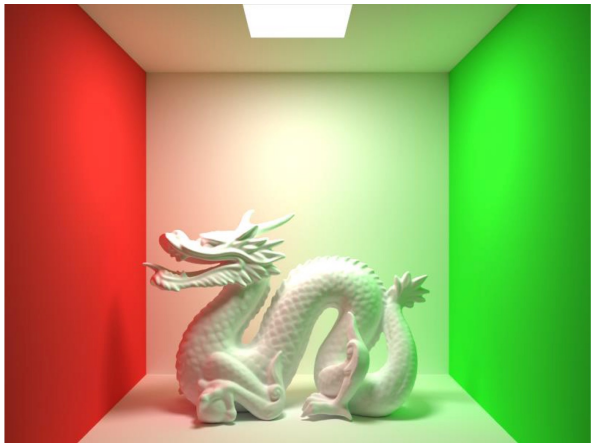
- “Path Tracing” for realistic (cont.)



<1 Path / Pixel>



<10 Path / Pixel>



<10000 Path / Pixel>

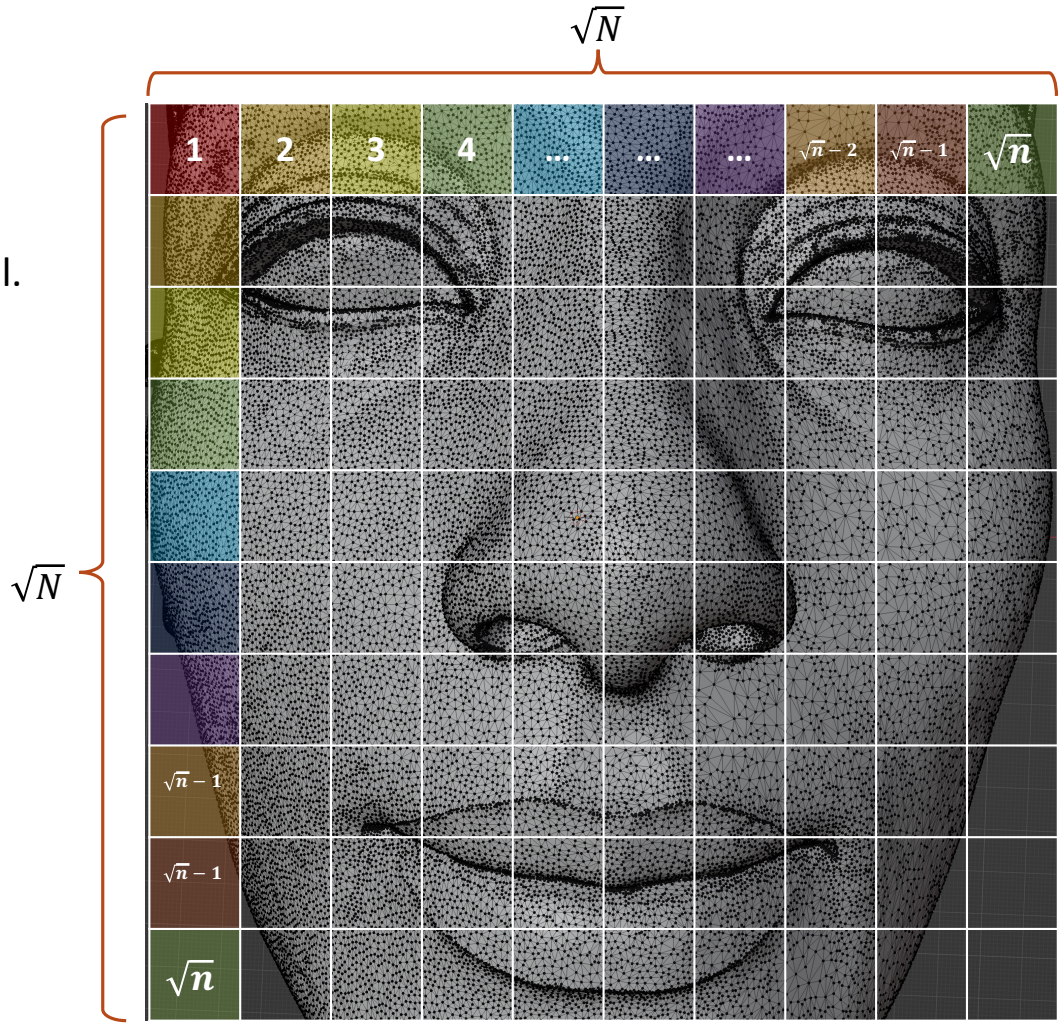
Hypothesis

- "Parallel Computing as a Solution”
- Scalabiblity--Color of each pixel can be calculated independently
  (The interactions between the beams are not considered)
- Each thread is responsible for multiple pixels
- Using GPU(CUDA) more efficiently reduces rendering time-more resources and high speed computation

Key idea

- Changing the algorithm is not our Goal.

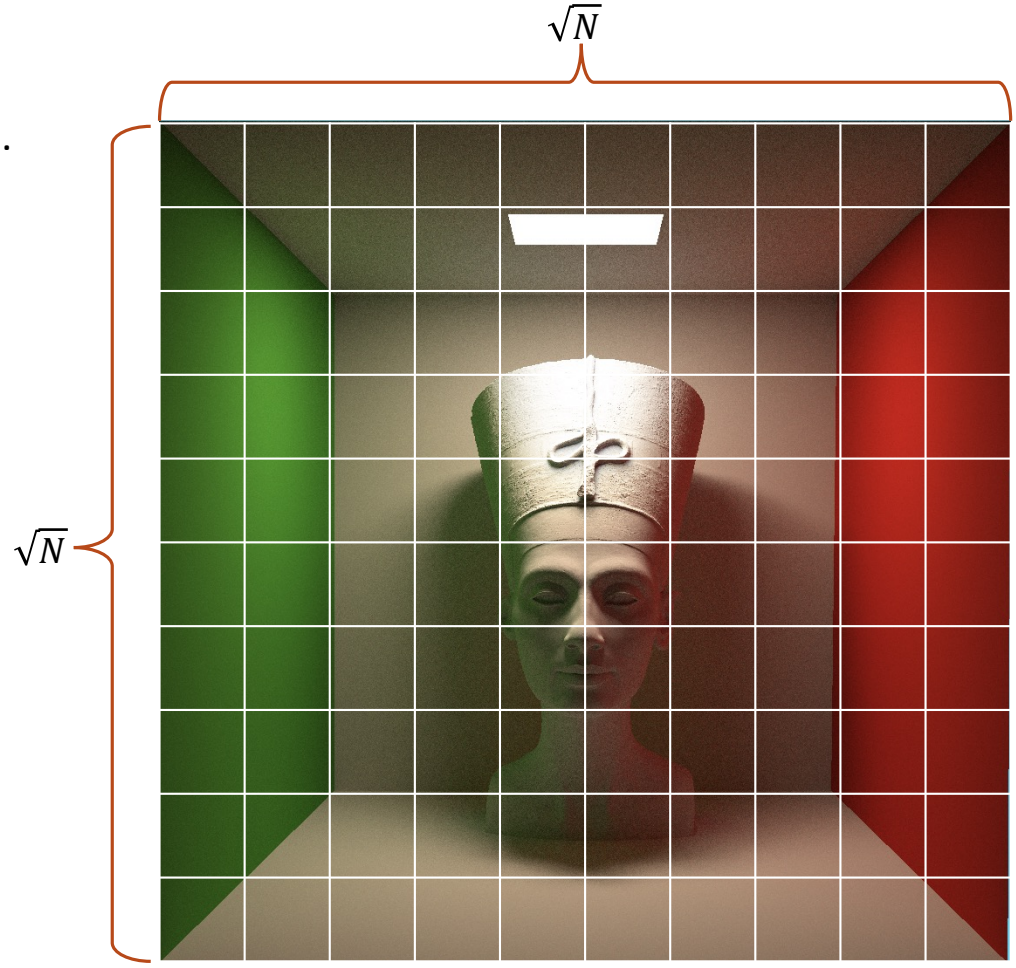
- ***N threads responsible for vertices***
  - **Assembling Triangle (Vertice → triangle)**
  - **Triangle: Move, Scale, Rotate**



Key idea

- Changing the algorithm is not our Goal.

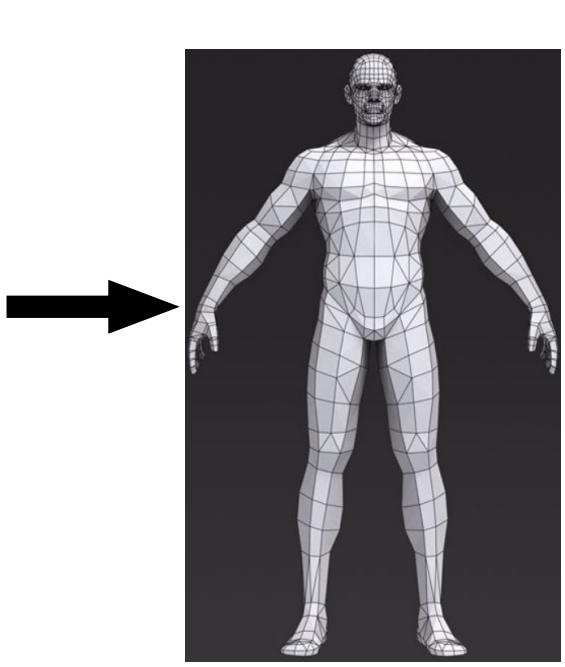
- ***N threads responsible for each block***
  - ***BlockSize = No. of Pixel / No. of Threads***

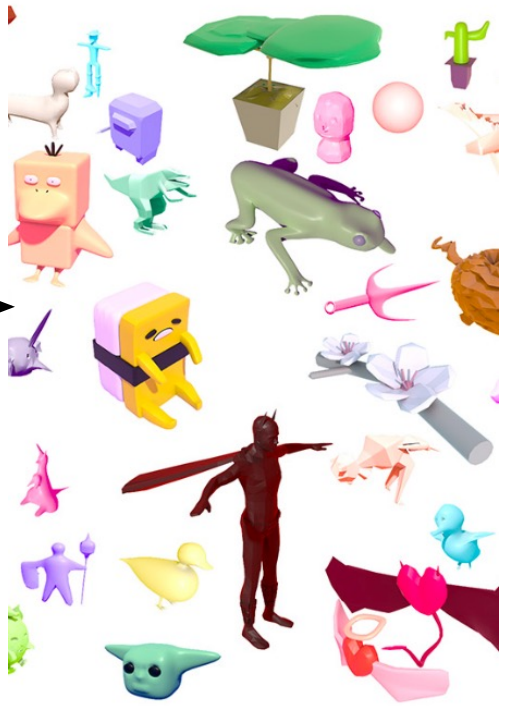


Challenge: Loading the scene

- Vertices
- Meshes
- Triangles
- Materials
- Textures
- Positions
- Rotations
- …

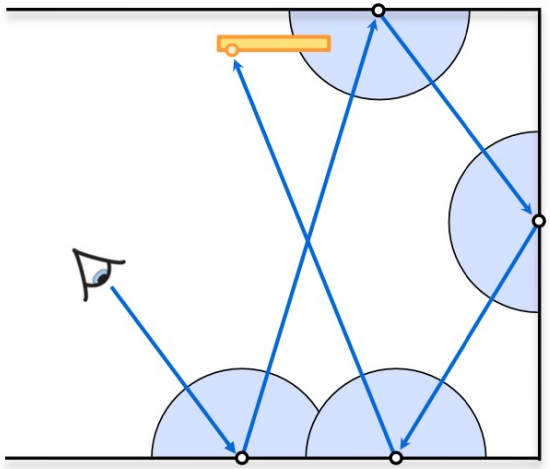
```
o Visor
v 0.320384 14.057541 0.507779
v 0.385196 13.984534 0.445066
v 0.416643 14.114325 0.462461
v 0.629804 14.365792 0.220990
v 0.643727 14.511806 0.201993
v 0.624523 14.536142 0.250058
v 0.626923 14.357680 0.224195
v 0.598598 14.519918 0.300183
v 0.605319 14.333345 0.262418
v 0.516982 14.252226 0.379605
v 0.571713 14.503694 0.351682
v 0.522263 14.487471 0.422635
v 0.445928 14.479359 0.506406
v 0.458891 14.171107 0.431790
v 0.340788 14.203555 0.534330
v 0.335027 14.146772 0.525174
v 0.265893 14.098101 0.546231
v 0.235407 14.187331 0.577817
v 0.195679 14.252226 0.604825
v 0.358791 14.471247 0.573697
v 0.246449 14.463135 0.627713
v 0.121145 14.455023 0.659757
v 0.150431 14.276562 0.621305
v 0.071395 14.309009 0.642362
v 0.009703 14.325233 0.651059
v 0.009703 14.455023 0.673947
v 0.009703 14.325233 0.651059
v 0.009703 14.455023 0.673947
v -0.101738 14.455023 0.659757
v -0.051989 14.309009 0.642362
v -0.131024 14.276562 0.621305
v -0.176273 14.252226 0.604825
v -0.227163 14.463135 0.627713
```





Challenge: Algorithm

- Where should I place the camera?
- How to calculate ray direction?
- How to know if the ray is blocked?
- How to simulate the light path in a physical-based way?
- Radiance?
- How to calculation the integral?
- How many times should a ray be reflected?
- How to speed up the whole process?
- …

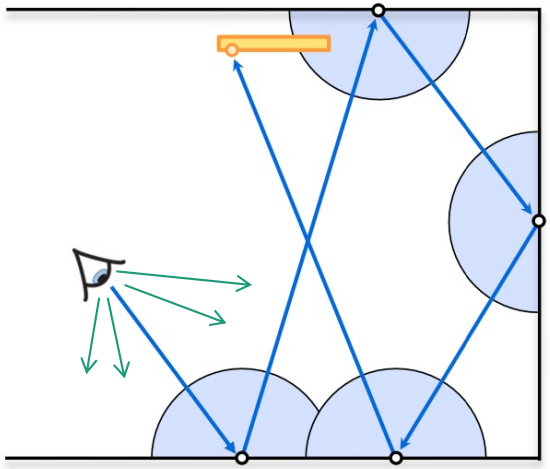


$$L(\mathbf{x},\vec{\omega}) = L_e(\mathbf{x},\vec{\omega}) + \int_{H^2} f_r(\vec{\mathbf{x}},\vec{\omega}',\vec{\omega}) L(r(\mathbf{x},\vec{\omega}'),-\vec{\omega}') \cos\theta' \mathrm{d}\vec{\omega}'$$

$$L(\mathbf{x},\vec{\omega}) \approx L_e(\mathbf{x},\vec{\omega}) + \frac{f_r(\vec{\mathbf{x}},\vec{\omega}',\vec{\omega}) L(r(\mathbf{x},\vec{\omega}'),-\vec{\omega}') \cos\theta'}{p(\vec{\omega}')}$$

Challenge: Parallelization

- Can we parallize the loading process?
- Can we parallize the algorithm for each pixel? (Multiple sample points)
- How to balance the calcutaion time and the image quality?
- Threads number
- How to write codes of CUDA programs within cpp project?
- Where and when to malloc and memcpy to CUDA memory?
- Using shared memory
- Memory alignment for coalesced reads from GPU memory
- Thread divergence
- ...



$$L(\mathbf{x},\vec{\omega}) = L_e(\mathbf{x},\vec{\omega}) + \int_{H^2} f_r(\vec{\mathbf{x}},\vec{\omega}',\vec{\omega}) L(r(\mathbf{x},\vec{\omega}'),-\vec{\omega}')\cos\theta' \mathrm{d}\vec{\omega}'$$

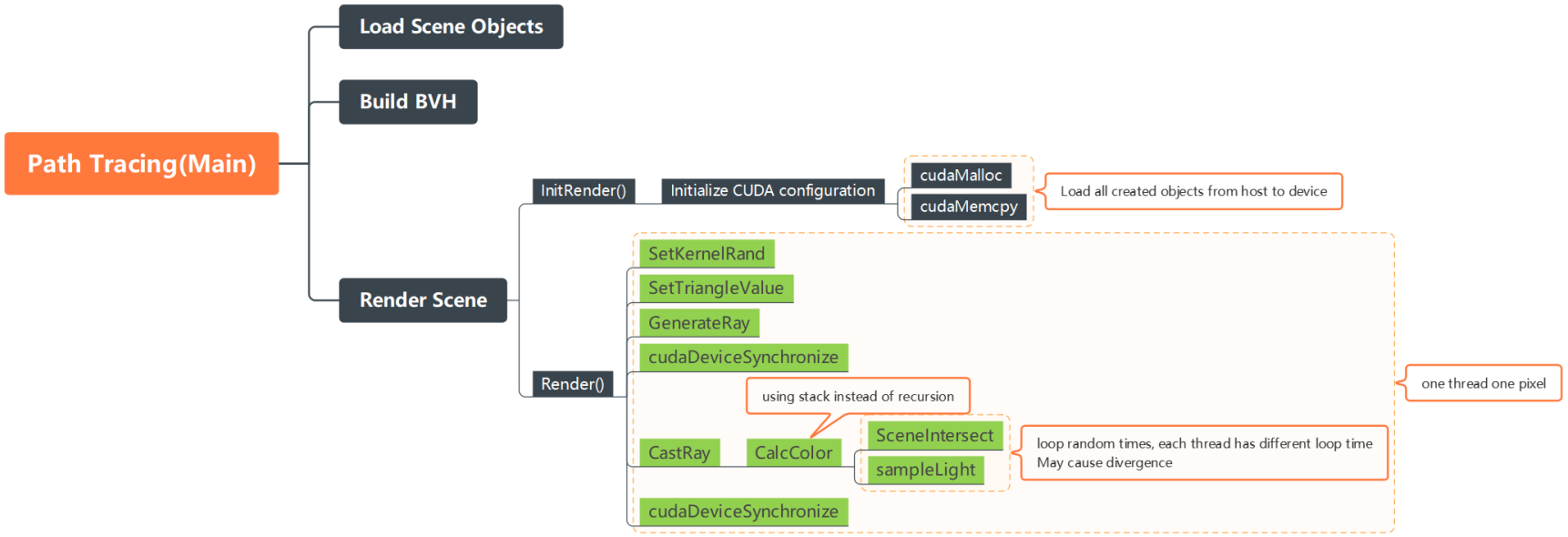
$$L(\mathbf{x},\vec{\omega}) \approx L_e(\mathbf{x},\vec{\omega}) + \frac{f_r(\vec{\mathbf{x}},\vec{\omega}',\vec{\omega}) L(r(\mathbf{x},\vec{\omega}'),-\vec{\omega}')\cos\theta'}{p(\vec{\omega}')}$$

Implementation (Serial)

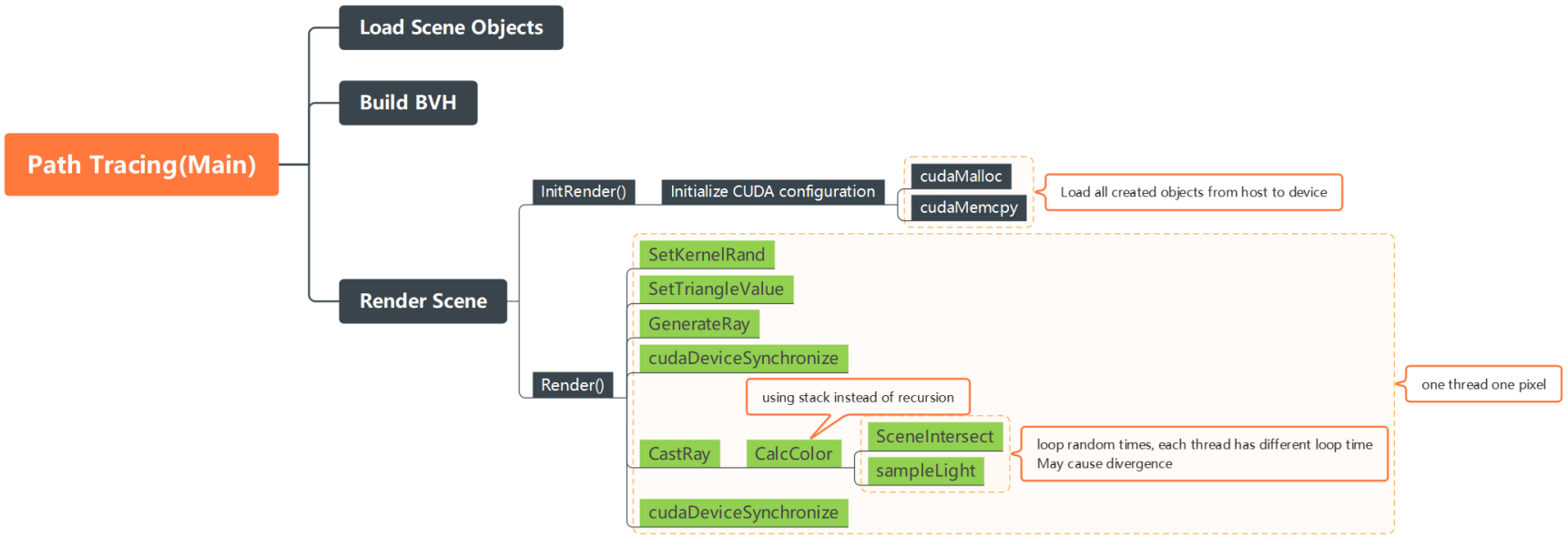
```
shade(p, wo)
    # Contribution from the light source.
    Uniformly sample the light at x' (pdf_light = 1 / A)
    L_dir = L_i * f_r * cos θ * cos θ' / |x' - p|^2 / pdf_light

    # Contribution from other reflectors.
    L_indir = 0.0
    Test Russian Roulette with probability P_RR
    Uniformly sample the hemisphere toward wi (pdf_hemi = 1 / 2pi)
    Trace a ray r(p, wi)
    If ray r hit a non-emitting object at q
        L_indir = shade(q, -wi) * f_r * cos θ / pdf_hemi / P_RR

    Return L_dir + L_indir
```

CSDN @Q_pril

Implementation (OpenMP)



Implementation (CUDA)

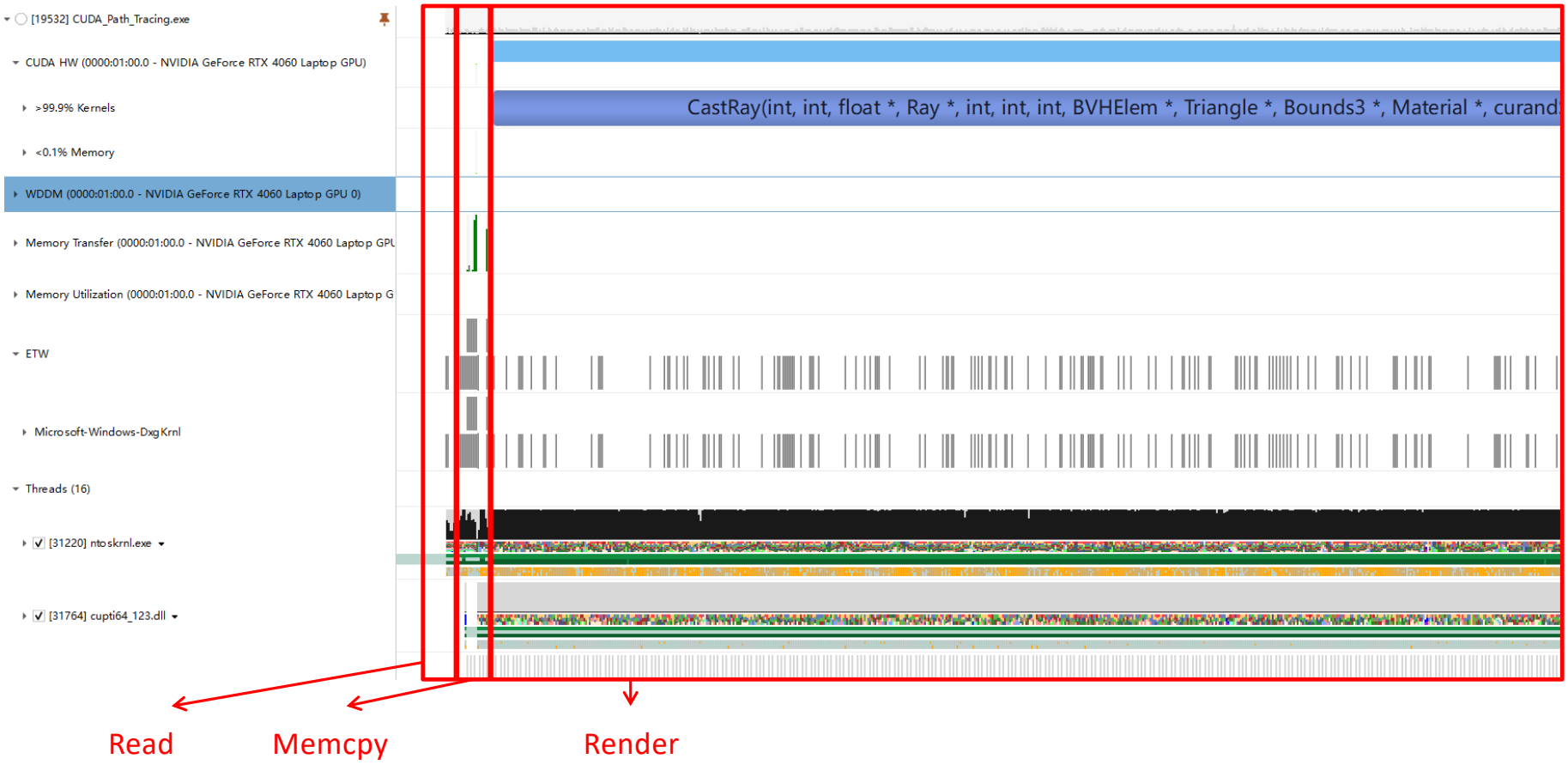


Experimental Setup

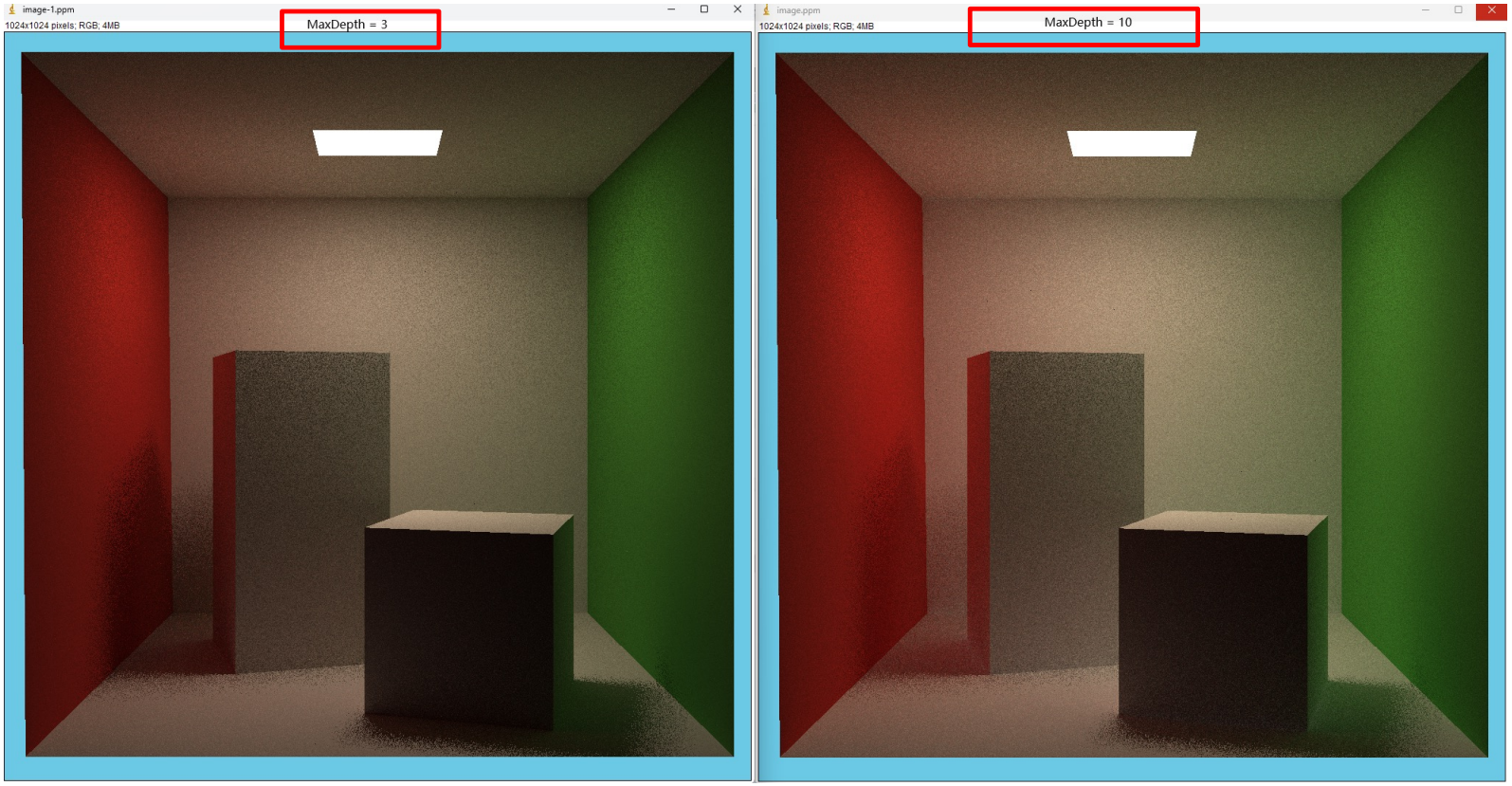
```
PS C:\Program Files\NVIDIA GPU Computing Toolkit
[CUDA Bandwidth Test] - Starting...
Running on...

 Device 0: NVIDIA GeForce RTX 4060 Laptop GPU
 Quick Mode

 Host to Device Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(MB/s)
   33554432                     12542.7

 Device to Host Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(MB/s)
   33554432                     12680.8

 Device to Device Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(MB/s)
   33554432                     206088.0

Result = PASS

NOTE: The CUDA Samples are not meant for perform
```

```
Device 0: "NVIDIA GeForce RTX 4060 Laptop GPU"
  CUDA Driver Version / Runtime Version          12.2 / 12.3
  CUDA Capability Major/Minor version number:    8.9
  Total amount of global memory:                 8188 MBytes (8585216000 bytes)
MapSMtoCores for SM 8.9 is undefined.  Default to use 128 Cores/SM
MapSMtoCores for SM 8.9 is undefined.  Default to use 128 Cores/SM
  (24) Multiprocessors, (128) CUDA Cores/MP:     3072 CUDA Cores
  GPU Max Clock rate:                            2370 MHz (2.37 GHz)
  Memory Clock rate:                             8001 Mhz
  Memory Bus Width:                              128-bit
  L2 Cache Size:                                 33554432 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(131072), 2D=(131072, 65536), 3D=(16384, 16384, 16384)
  Maximum Layered 1D Texture Size, (num) layers  1D=(32768), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(32768, 32768), 2048 layers
  Total amount of constant memory:               zu bytes
  Total amount of shared memory per block:       zu bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  1536
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          zu bytes
  Texture alignment:                             zu bytes
  Concurrent copy and kernel execution:          Yes with 1 copy engine(s)
  Run time limit on kernels:                     Yes
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Disabled
  CUDA Device Driver Mode (TCC or WDDM):         WDDM (Windows Display Driver Model)
  Device supports Unified Addressing (UVA):      Yes
  Device supports Compute Preemption:            Yes
  Supports Cooperative Kernel Launch:            Yes
  Supports MultiDevice Co-op Kernel Launch:      No
  Device PCI Domain ID / Bus ID / location ID:   0 / 1 / 0
```

Performance analysis:

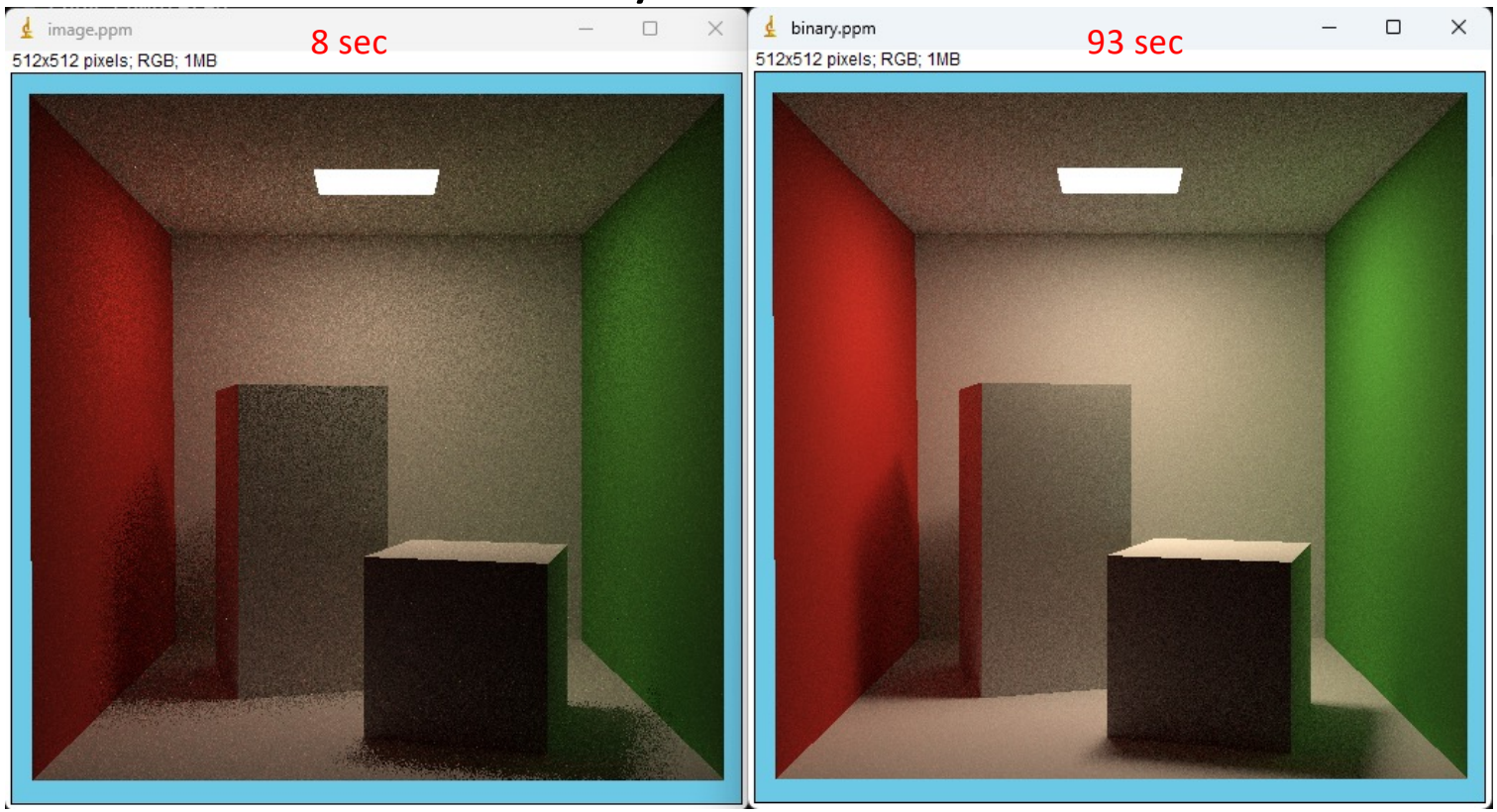


Performance analysis:



Changing MaxDepth will increase calculation time, but have little effect on the result.

Performance analysis:



Implementation on CUDA is slightly different to achieve maximized speedup. (next step is to a

Experimental Setup

- (Model object) Nefetiti (about 2M triangles, file size: 183MB)

- (Setting) Server with CARC (Intel® Xeon® Silver 4126 CPU @ 2.10Ghz)
  - cpus-per-task = 8
  - mem=32G
  - time=2:00:00

- Measure single threaded performance and multi threaded performance (using openMP and CUDA)
  - Resolution(3): 1024x1024, 512x512, 256x256
  - Path tracing sample space(3): 32, 64, 128
  - No. of threads(9): 1, 2, 4, 8, 16, 32, 64, 128, 256
  - Total test set: 81 case (3 x 3 x 9)

Experimental Results

256 x 256

512 x 512

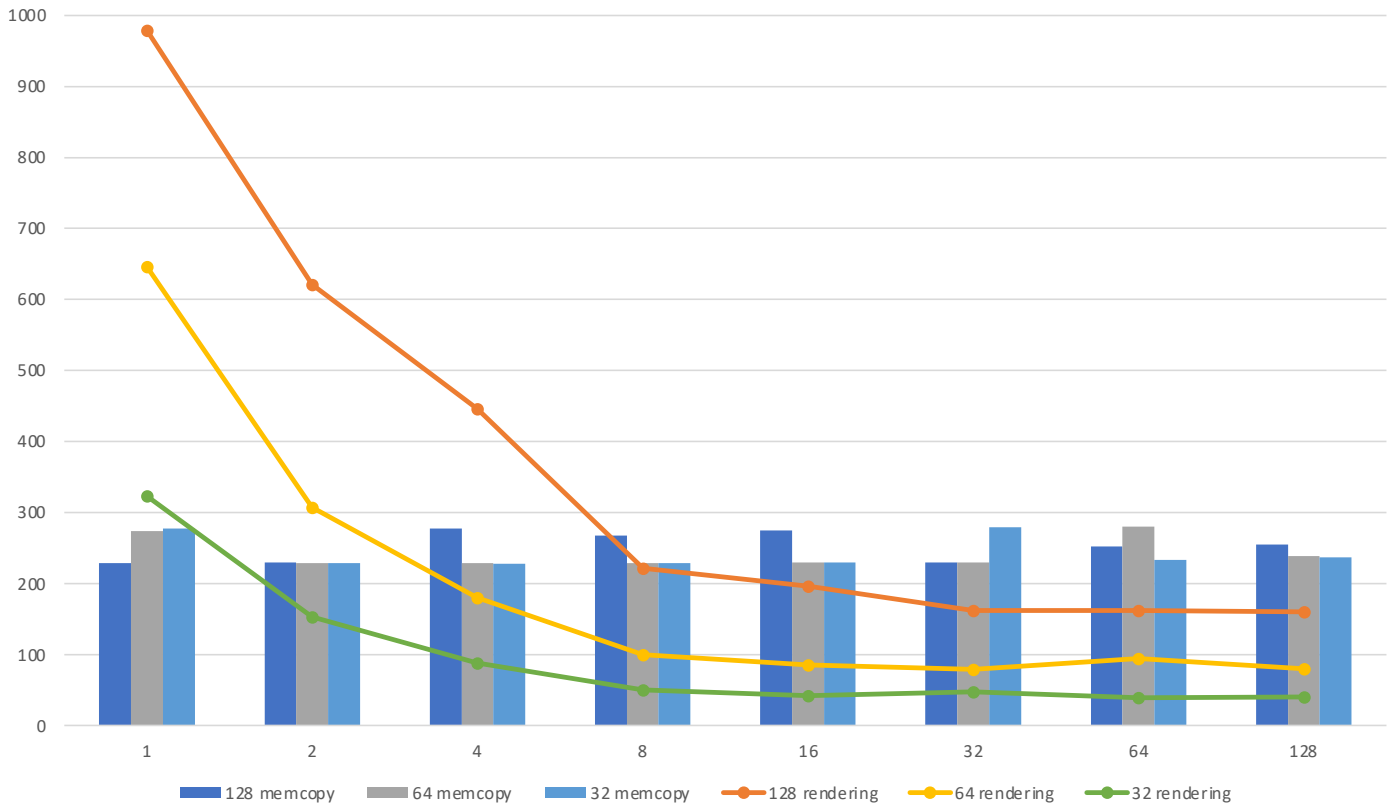
1024 x 1024

Experimental Results (256 x 256)

SPP= 32

SPP= 64

SPP= 128

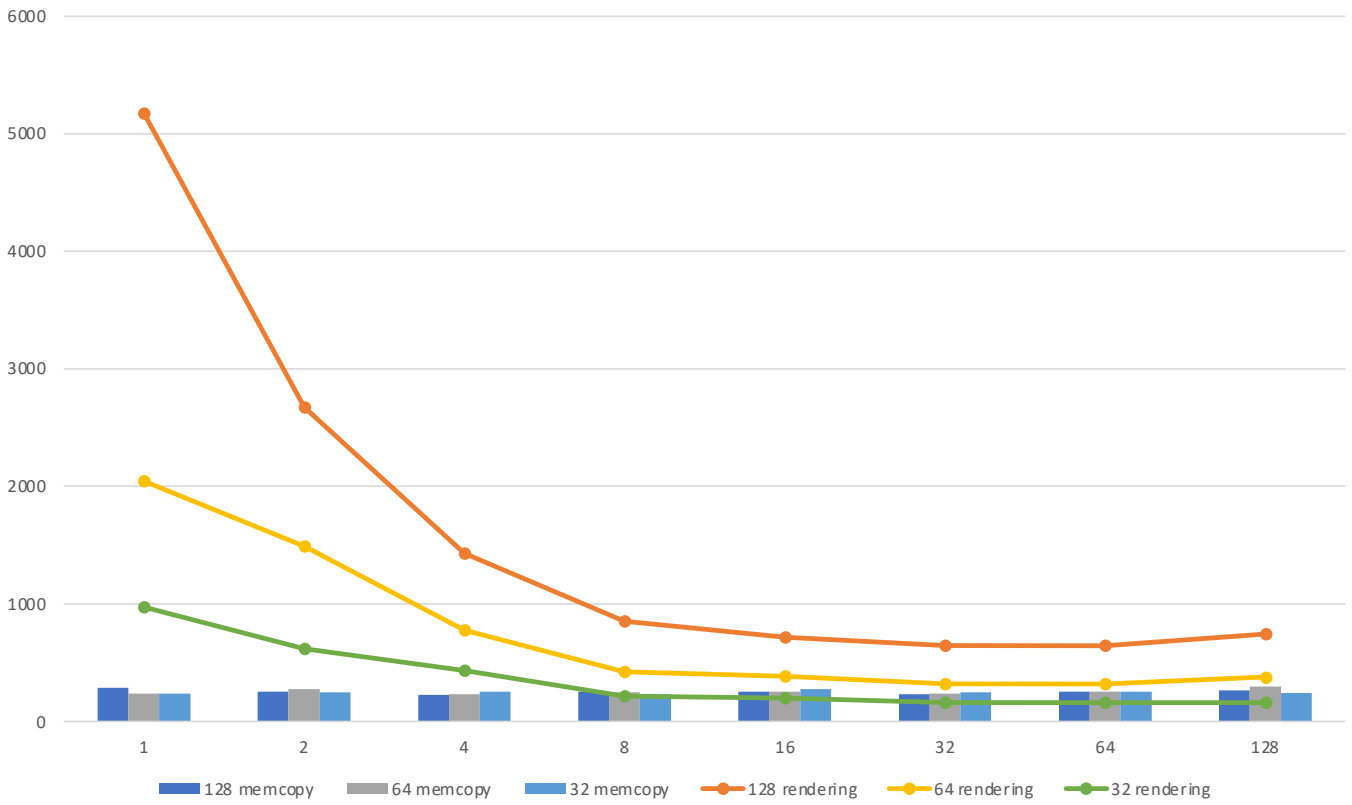


Experimental Results (512 x 512)

SPP= 32

SPP= 64

SPP= 128

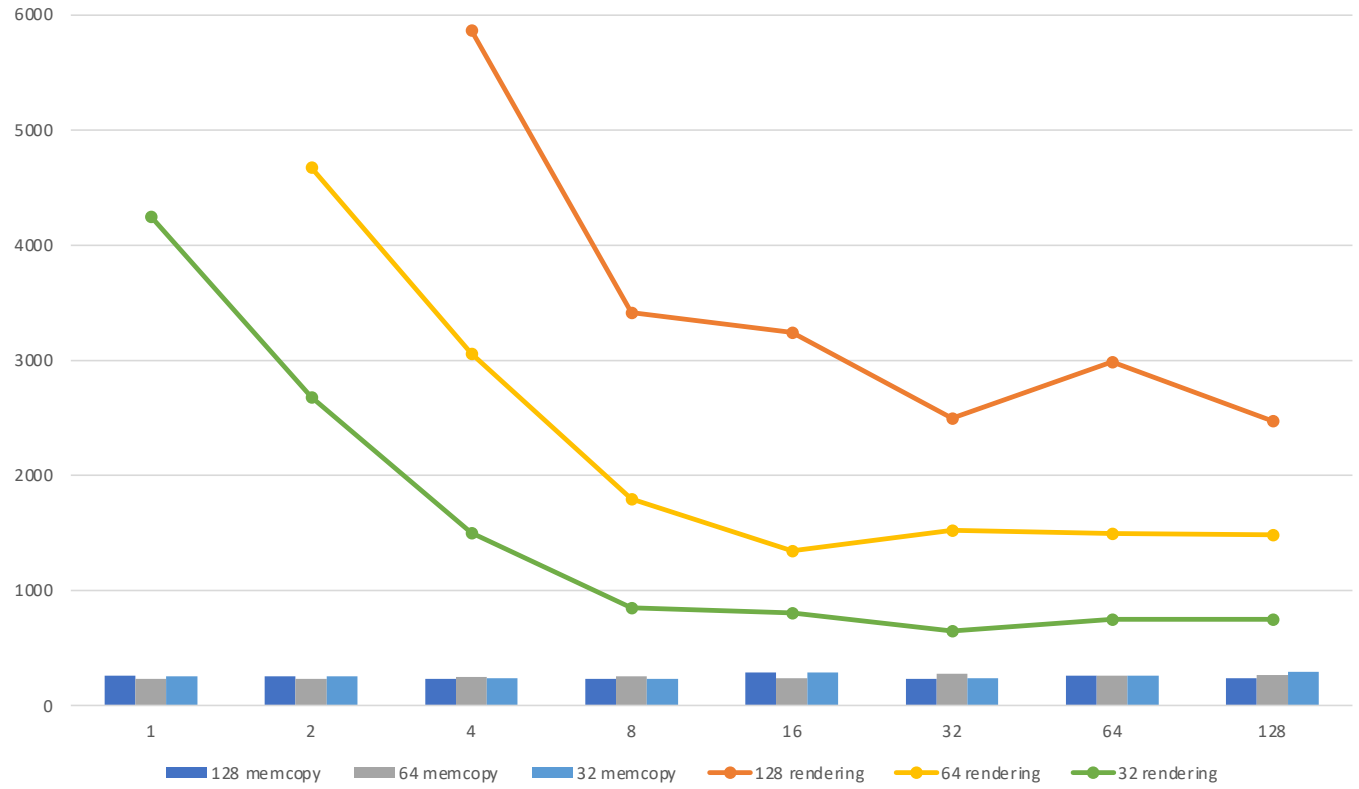


Experimental Results (1024 x 1024)

SPP= 32

SPP= 64

SPP= 128



Experimental Results (Using CUDA)

- Processing…

Conclusion

- Irregular memory accesses and lack of data locality makes graph processing challenging. We observed that for large datasets, cache efficiency is very poor (30-35% cache miss rate for pld, sd1).

- SPP = 256 ➔ Out of Memory

- Resolution=1024x1024, SPP=128, No. of Threads=1,2 ➔ Time out (2hours)

References

1. Yunming Zhang, Vladimir Kiriansky, Charith Mendis, Matei Zaharia, and Saman P. Amarasinghe. *Optimizing cache performance for graph analytics*. CoRR, abs/1608.01362, 2016

- [1] Rost, Randi J ; Licea-Kane, Bill ; Ginsburg, Dan ; Kessenich, John ; Lichtenbelt, Barthold ; Malan, Hugh ; Weiblen, Mike, “OpenGL shading language”, Pearson Addision Wiseley; 2021
-
- [2] Gordon, V.scott, “Computer Graphics Programming in OpenGL with C++”, Bloomfield, New jersey : Mercury Learning & Information; 2020
- [1] OpenGL, https://www.opengl.org, accessed 2023-10-20
-
- [2] KHORONOS group, https://www.khronos.org/opengl, accessed 2023-10-20
- PATH TRACING: A NON-BIASED SOLUTION TO THE RENDERING EQUATION ROBERT CARR AND BYRON HULCHER

- Real-Time Bidirectional Path Tracing via Rasterization Yusuke Tokuyoshi∗ Square Enix Co., Ltd. Shinji Ogaki† Square Enix Co., Ltd.
- Ray Tracing References • “An Improved Illumination Model for Shaded Display”, Turner Whitted, 1980 • “Distributed Ray Tracing”, Cook, Porter, Carpenter, 1984 • “The Rendering Equation”, James Kajiya, 1986 • “Bidirectional Path Tracing”, Eric Lafortune, Yves Willems, 1993 • “Rendering Caustics on Non-Lambertian Surfaces”, Henrik Wann Jensen, 1996 • “Metropolis Light Transport”, Eric Veach, Leonidas Guibas, 1997
- A Reflectance Graphics ROBERT L. COOK Lucasfilm Ltd. and KENNETH E. TORRANCE Cornell University
- Real-Time Rendering Fourth Edition
- Online chapter: Real-Time Ray Tracing version 1.4; November 5, 2018
- Download latest from http://realtimerendering.com
- Tomas Akenine-M¨oller Eric Haines
  Naty Hoffman
  Angelo Pesce
- Michal Iwanicki S´ebastien Hillaire