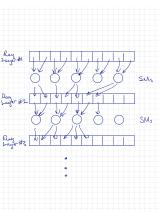
Parallel Ray Tracing

CUDA



Idea:

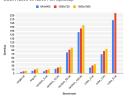
- 1. Execute ray simulation in iterations
- Reassign new reflected and refracted rays each iteration
- Compute surface colors from last layer and iterate backwards

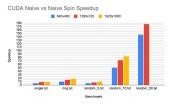
Constraints:

- Each layer depends on previous layer
- Extremely memory intensive
- Dynamically scaling workload per layer
- Diffuse vs bounced rays not data parallel

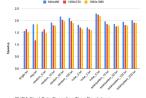
Benefits

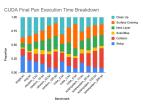
- Collision checking is data parallel
- Tracing each ray is independent



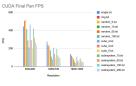


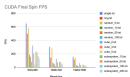








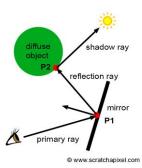




Conclusions:

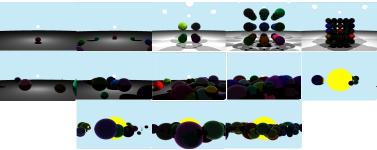
- More memory constrained the compute constrained
- Increasing object count less impactful than expected
 Ideal platform for ray tracing implementation

Background



The ray tracing algorithm is a way to render a 3D scene to a 2D image by simulating the mathematical properties of light rays.

Benchmarks



Three Load Factors:

- 1. Resolution: 640x460, 1280x720, 1920x1080
- 2. Number of objects: 1 to 201
- 3. Number of non-diffusing rays: PAN vs SPIN

Naive Implementation



Idea:

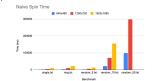
For each ray, recursively trace it and all its scattered rays until ray depth or diffuse, then recursively backtrack to compute surface color.

Benefits:

- Simple to understand
- Create a complex and accurate simulation of real light and shadows

Drawbacks:

- Must check every object against every ray
- Recursion is expensive in practice
- · Slow compared to rasterization



OpenMP/AVX

Idea:

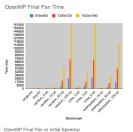
- 1. Execute multiple ray simulations in parallel
- 2. Distribute workload evenly between threads
- 3. Optimize arithmetic computations with vector intrinsics

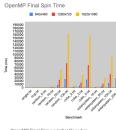
Constraints

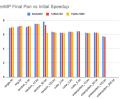
- Each layer depends on previous layer
- Nested and recursive structure harder to parallelize with OpenMP
- Using AVX intrinsics with complex data types

Benefits

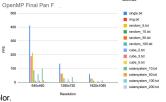
- Tracing each ray is independent
- Dynamic scheduling prevents large load



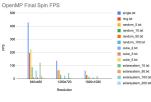




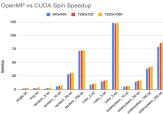




OpenMP vs CUDA Pan Speedup



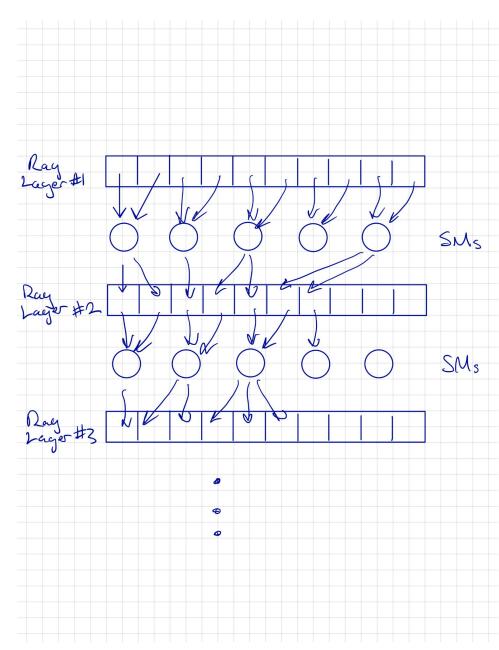




Conclusions:

- Increasing object count highly impactful versus CUDA
- Provides general speedup benefits over naive, but at a completely lower scale than CUDA

CUDA



Idea:

- 1. Execute ray simulation in iterations
- 2. Reassign new reflected and refracted rays each iteration
- 3. Compute surface colors from last layer and iterate backwards

Constraints:

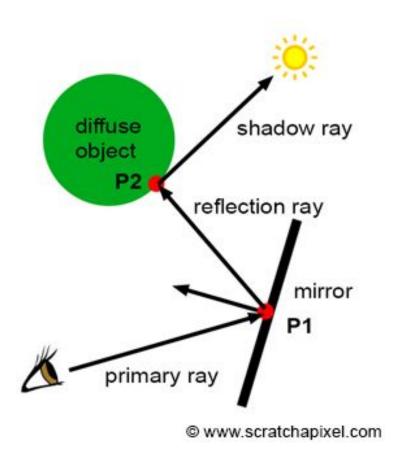
- Each layer depends on previous layer
- Extremely memory intensive
- Dynamically scaling workload per layer
- Diffuse vs bounced rays not data parallel

Benefits

- Collision checking is data parallel
- Tracing each ray is independent

Parallel Ray Tracing

Background



The ray tracing algorithm is a way to render a 3D scene to a 2D image by simulating the mathematical properties of light rays.

OpenMP/AVX

Idea:

- 1. Execute multiple ray simulations in parallel
- 2. Distribute workload evenly between threads
- 3. Optimize arithmetic computations with vector intrinsics

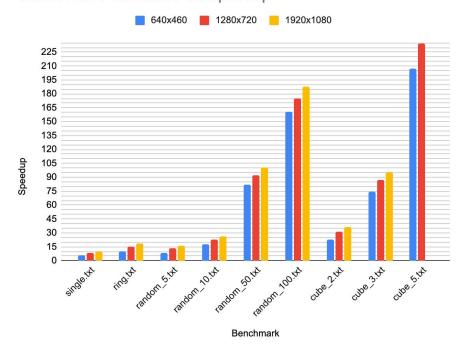
Constraints:

- Each layer depends on previous layer
- Nested and recursive structure harder to parallelize with OpenMP
- Using AVX intrinsics with complex data types

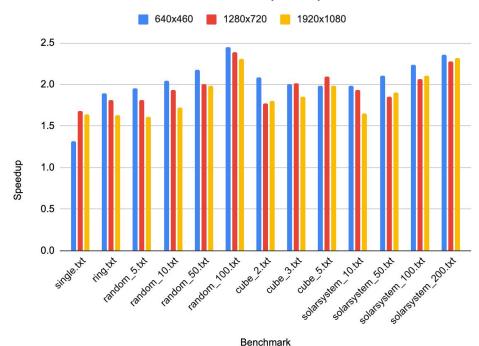
Benefits

- Tracing each ray is independent
- Dynamic scheduling prevents large load imbalances

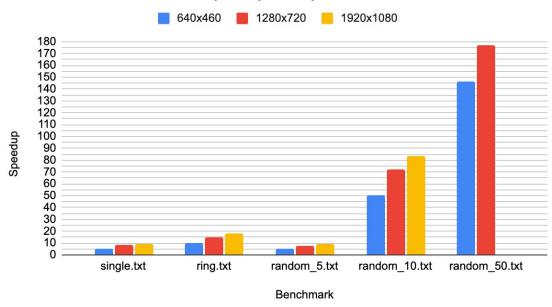
CUDA Naive vs Naive Pan Speedup



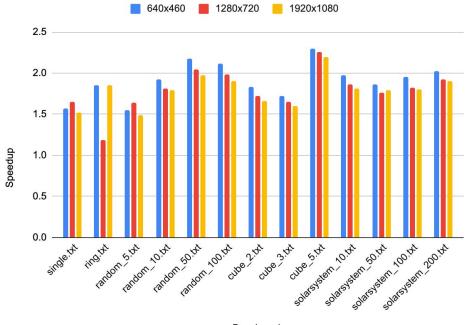
CUDA Final vs CUDA Naive Pan Speedup



CUDA Naive vs Naive Spin Speedup

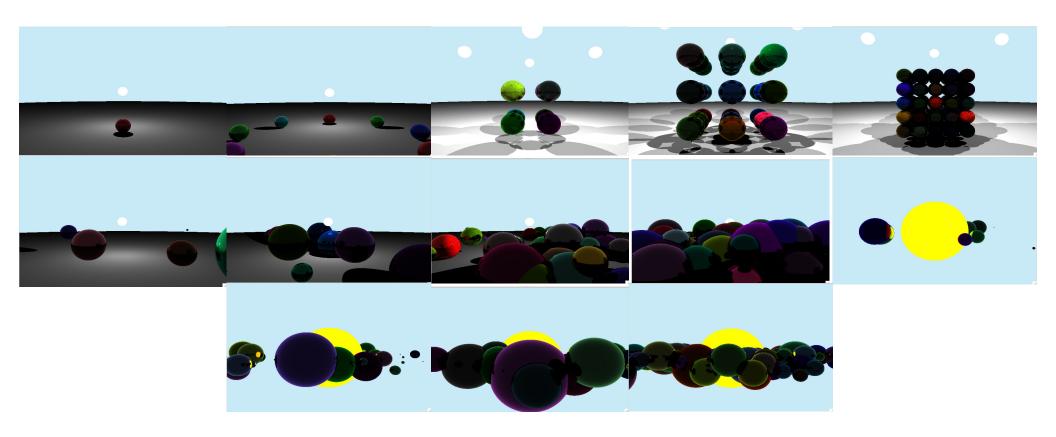


CUDA Final vs CUDA Naive Spin Speedup



Benchmark

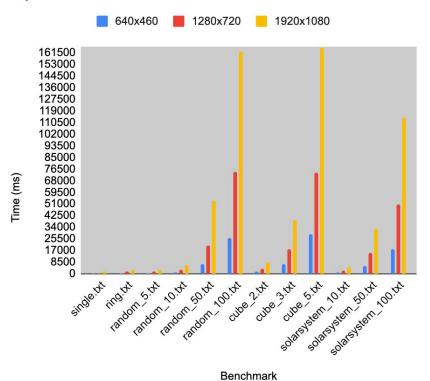
Benchmarks



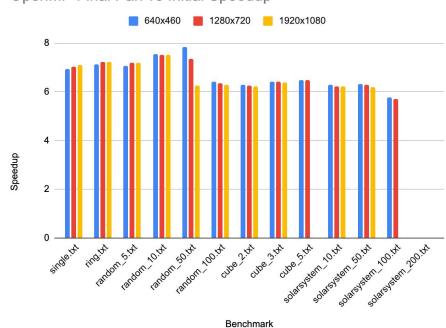
Three Load Factors:

- 1. Resolution: 640x460, 1280x720, 1920x1080
- 2. Number of objects: 1 to 201
- 3. Number of non-diffusing rays: PAN vs SPIN

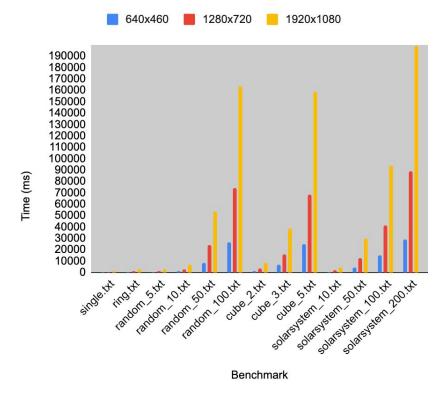
OpenMP Final Pan Time



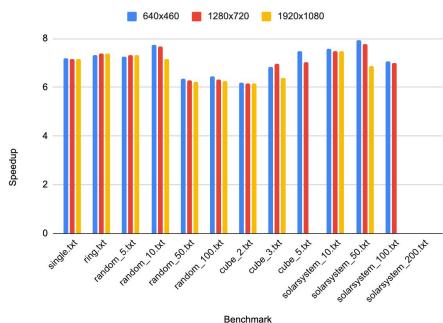
OpenMP Final Pan vs Initial Speedup



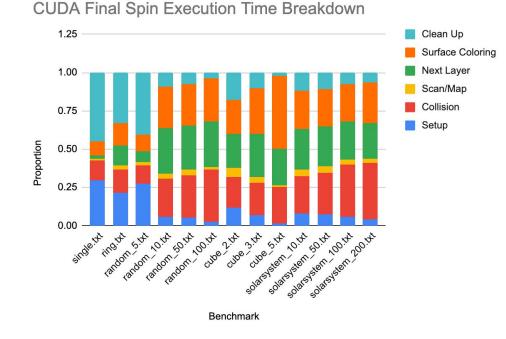
OpenMP Final Spin Time

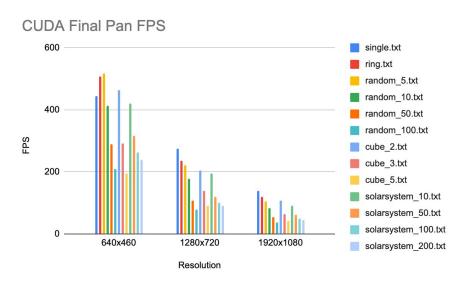


OpenMP Final Spin vs Initial Speedup

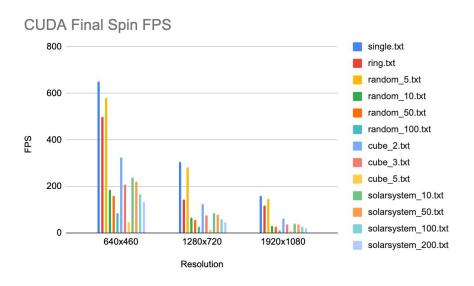


CUDA Final Pan Execution Time Breakdown 1.00 Clean Up Surface Coloring Next Layer 0.75 Scan/Map Collision Setup 0.50 Proportion 0.25 random so st tardon, 10 th solatestern, O.W. solatestern 50.1xt solar steen, now random 5.14 random 10 th oupe 2.1xt cube 3.1xt single, th ingita





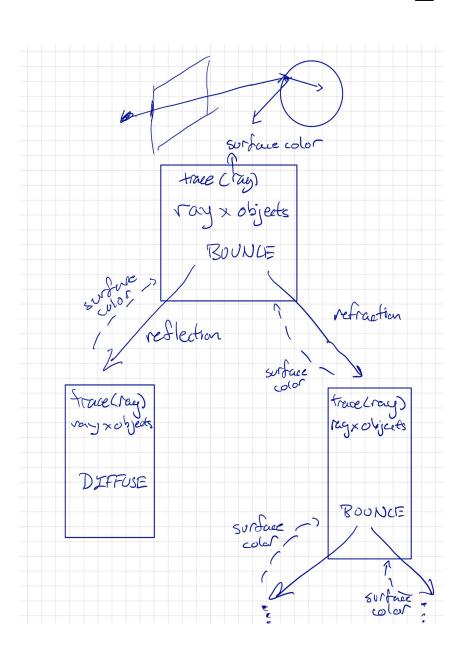
Benchmark



Conclusions:

- More memory constrained the compute constrained
- Increasing object count less impactful than expected
- Ideal platform for ray tracing implementation

Naive Implementation



Idea:

For each ray, recursively trace it and all its scattered rays until ray depth or diffuse, then recursively backtrack to compute surface color.

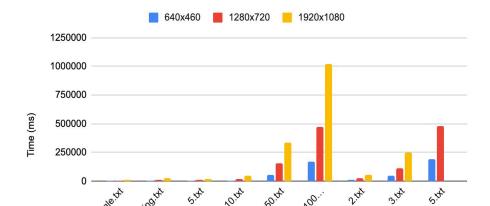
Benefits:

- Simple to understand
- Create a complex and accurate simulation of real light and shadows

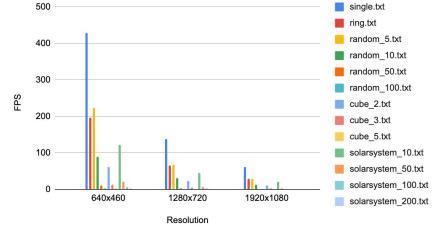
Drawbacks:

Naive Pan Time

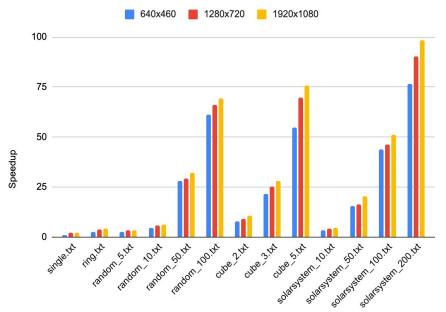
- Must check every object against every ray
- Recursion is expensive in practice
- Slow compared to rasterization



OpenMP Final Pan FPS 500 single.txt ring.txt random_5.txt 400 random_10.txt random 50.txt 300 random 100.txt cube 2.txt 200 cube 3.txt cube 5.txt 100 solarsystem 10.txt solarsystem 50.txt solarsystem_100.txt 640x460 1280x720 1920x1080 solarsystem 200.txt Resolution

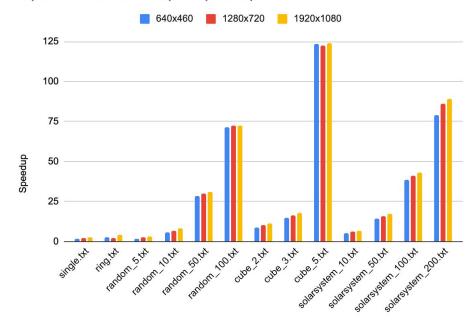








OpenMP Final Spin FPS



Conclusions:

- Increasing object count highly impactful versus CUDA
- Provides general speedup benefits over naive, but at a completely lower scale than CUDA