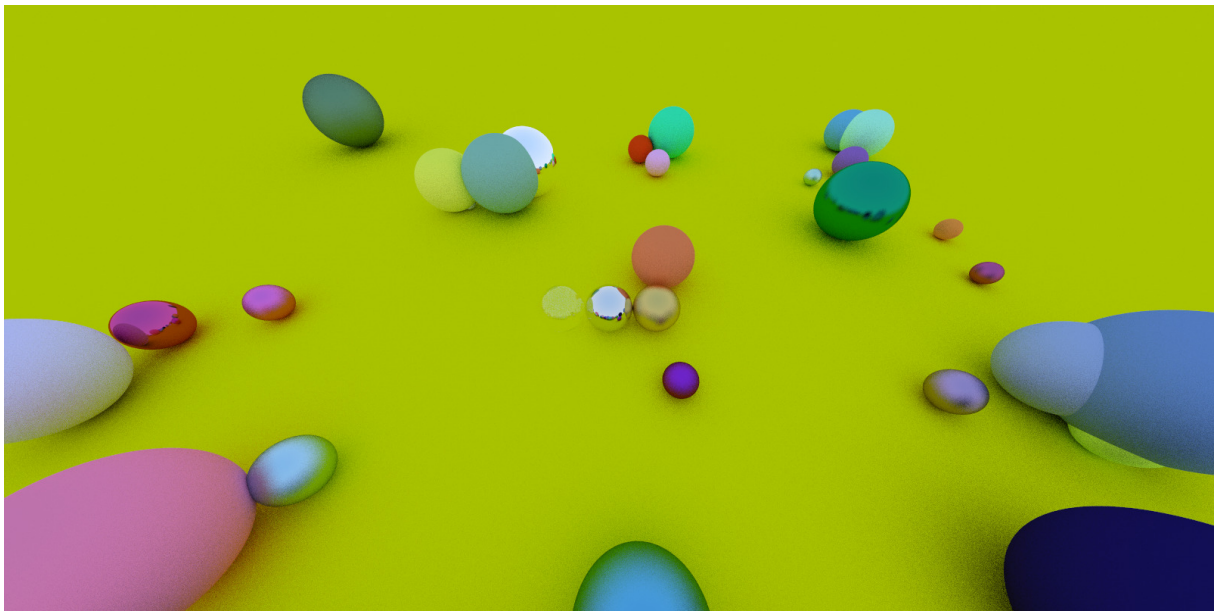

LULEÅ UNIVERSITY OF TECHNOLOGY (LTU)

Optimization of raytracer

Student name: *Benjamin Vesterlund*

Course: *S0017D Spelkonsoler och system*
Professor: *Fredrik Lindahl*
Date: *September 21, 2020*

Contents

1 Before Optimization **2**
1.0.1 Test . . . 2
1.1 Testing with Valgrind . . . 2
1.2 Testing with VS . . . 3
1.3 Testing with Lint . . . 3
1.4 Identifying the critical code path . . . 3

2 Optimization **4**
2.1 Cleaning memory allocation and cleanup . . . 4

3 After optimization **4**
3.1 Memory allocation . . . 4
3.1.1 How is memory allocated . . . 4
3.1.2 My own memory allocator . . . 4
3.1.3 Improvement that should work . . . 4
3.2 Running the tests again . . . 4
3.2.1 Test . . . 4
3.3 Testing with Valgrind . . . 5
3.4 Callgrind result . . . 5
3.5 Cachegrind result . . . 5

1. Before Optimization

1.0.1. Test.

Variables:

- Size: 600x300
- ray/pixel: 10
- bounces: 5
- spheres: 27 (25 + world and center mirror sphere)

Result with no test:

- 1800000 rays in 5.07005s
- 355026 rays/s

1.1. Testing with Valgrind.

- 1704 (24 direct, 1680 indirect)
- definitely lost: 88 bytes in 3 blocks
- indirectly lost: 1,728 bytes in 57 blocks
- possibly lost: 0 bytes in 0 blocks

- still reachable: 0 bytes in 0 blocks
- suppressed: 0 bytes in 0 blocks

1.2. Testing with VS.

- 23 Possible loss of data warnings
- 4 Variable initialize warnings
- 2 Arithmetic overflow warnings

1.3. Testing with Lint.

- 1 Performance warning
- 19 Style warnings

1.4. Identifying the critical code path.

Callgrind resulted in:

- 46,443,661,218 Program totals
- 11,926,057,235 sphere::hit()
- 7,224,084,582 vector::dot()
- 7,214,692,233 ray::direction()

Cachegrind resulted in:

- I refs: 46,443,656,243
- I1 misses: 3,303
- I1 miss rate: 0.00%
- D refs: 30,980,314,440
- D1 misses: 41,144
- D1 miss rate: 0.0%
- LL refs: 44,447
- LL misses: 13,447
- LL miss rate: 0.0%

What takes time in this program is calculating the bounces, stepping through all spheres in one thread is a lot of work. The main improvement will be to multi thread this process. Work on the old mathLib might also help.

2. Optimization

2.1. Cleaning memory allocation and cleanup.

1. Added a delete loop to remove all the allocated pointers.
2. Cleaned all "errors" - made sure all type casts happen as intended.
3. Fixed up arithmetic's warnings.
4. Added threading.

3. After optimization

3.1. Memory allocation.

3.1.1. How is memory allocated.

currently there are multiple memory allocations, I have one sphere array, one material array and a *(thread)* argument array. then spheres are put into even more arrays.

3.1.2. My own memory allocator.

Tried many ways to reduce the amount of allocations and amount of lookups but found no easy or more effective.

3.1.3. Improvement that should work.

A system that would be interesting to implement (if I had nothing else to do.. which I do.) would be to make a quad-tree based lookup to not have to step through all spheres for every ray, when all that is needed is the spheres in a area around the ray with a radius of the largest sphere radius.

(might not be what was wanted in this section..)

3.2. Running the tests again.

3.2.1. Test.

Variables:

- Size: 600x300
- ray/pixel: 10
- bounces: 5
- spheres: 27 (25 + world and center mirror sphere)

Result with no test:

- 1800000 rays in 3.0214s
- 595750 rays/s

3.3. Testing with Valgrind.

- total heap usage: 70 allocs, 70 frees, 3,685,589 bytes allocated.
- all heap blocks were freed.
- 0 errors.

3.4. Callgrind result.

- 50,081,260,694 Program totals
- 12,132,669,920 sphere::hit()
- 7,722,587,376 vector::dot()
- 7,389,662,870 ray::direction()

3.5. Cachegrind result.

- I refs: 50,066,984,028
- I1 misses: 8,480,251
- I1 miss rate: 0.02%
- D refs: 30,962,611,358
- D1 misses: 205,601
- D1 miss rate: 0.0%
- LL refs: 8,685,852
- LL misses: 70,546
- LL miss rate: 0.0%