# Task 24 - Spike Summary Report

---

**Spike:** Task_24
**Title:** Profiling, Performance & Optimization
**Author:** Thomas Horsley, 103071494

## Goals & Deliverables

**Aim:** Given a codebase, analyze performance data and identify performance improvements.

**Deliverables:**

- Optimized code and performance data
- Spike Summary Report
- Git Commit History

---

## Technology, Tools and Resources

### Tech and Tools

The project was tested using C++ 14 in Visual Studio Community 2022.
Data was formatted in MS Excel 2022.
Source control is handled using Git.

---

## Tasks Undertaken

## Planning

### Given Function Definitions

| | |
|---|---|
| crash_test_all_A1 | Loops through the set of boxes twice checking if the two boxes are the same, if not access the via index element. Re-allocs all the member variables rvalue’s to new locals before running comparisons on these values. Uses a set of if statements for AABB collision detection |
| crash_test_all_A2 | Loops through the set of boxes twice accessing each box the via index. Re-allocs all the member variables rvalue’s to new names before running comparisons on these values. Uses a set of if statements for AABB collision detection |
| crash_test_all_B | Passed a struct temporary which will be shallow copied and exist locally within the function. All of the structs member variables are then again copied into local scope data before being checked against AABB collision detection algorithm |
| crash_test_all_C | Passed a reference to a struct. Still copies all of the member variables locally and runs AABB collision detection on the local data using a set of if statements. |
| crash_test_all_D | Passed a reference to a struct, uses a set of if statements to run AABB comparison using on each of the two structs member variables. |

### Custom Function Definitions

| | |
|---|---|
| crash_test_all_E | Passed two references to box structs, checks AABB collision using a single or'd if statement. |
| crash_test_all_F | Passed two references to box structs, checks AABB collision using a if else statement set instead of an if statement set |
| crash_test_all_G | Caches the array of boxes locally and accesses the struct refs using that local data. Runs collisions detection the same as crash_test_F( ). |

# Results & Discussion

> 💡 All tests were ran in visual studio's release environment with optimizations off (see **Code**). Tests were ran at a length of 15 seconds over 3 iterations. Frame pointers were enabled for debugging.

### Control Data & Window / Rendering Flag Optimizations

The control set was conducted on the given code with no modifications rendering was enabled as to understand how the collision functions effect the UX as a whole. Unless specified otherwise, all other tests were conducted with rendering disabled.

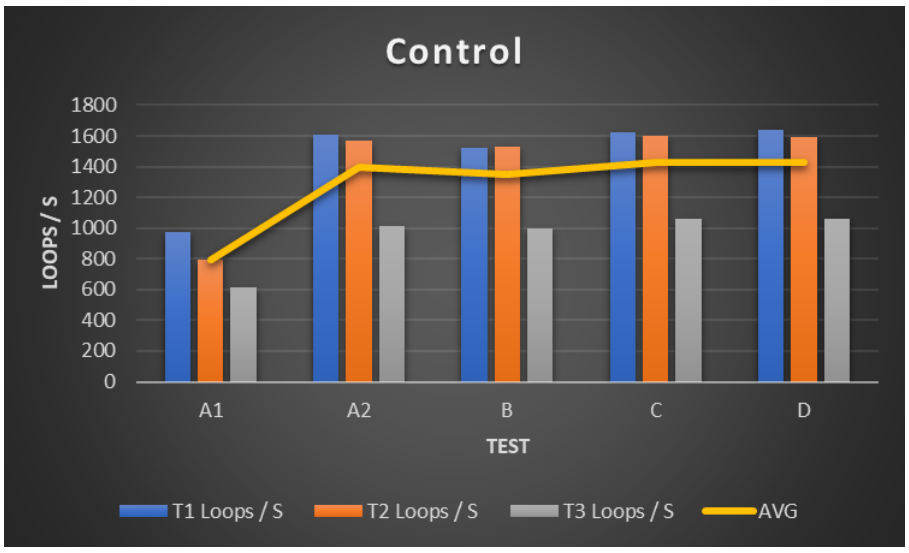



Fig 1.1 Control group with rendering overhead

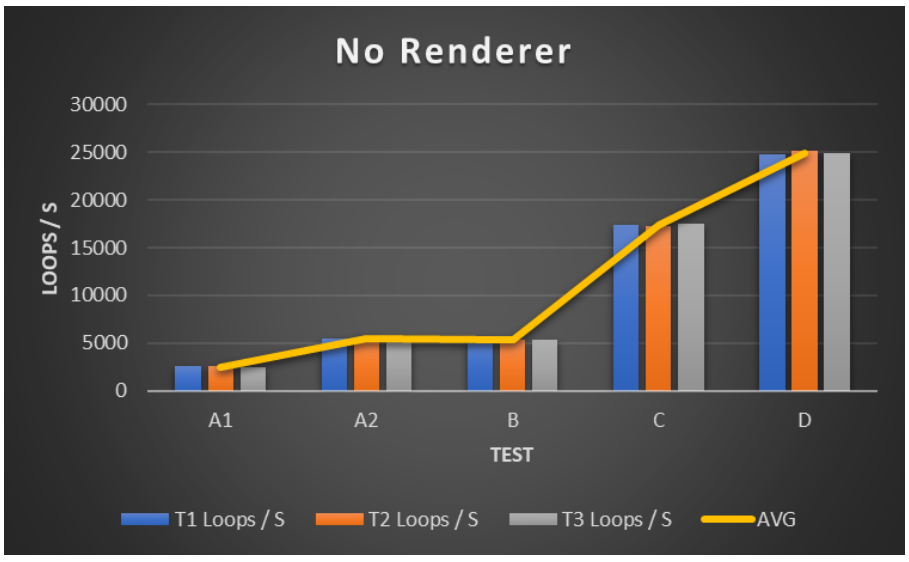



Fig 1.2 Control group senza rendering

The control group showed that only one of the collision detection methods had a significant impact on the performance when rendering overhead was included. Therefore, this will be considered the only 'unacceptable' function by performance (though significant improvements can still be made to future approaches). Once rendering is turned off, it's clear that crash_test_all_D( ) was the most performant. Future tests support this;

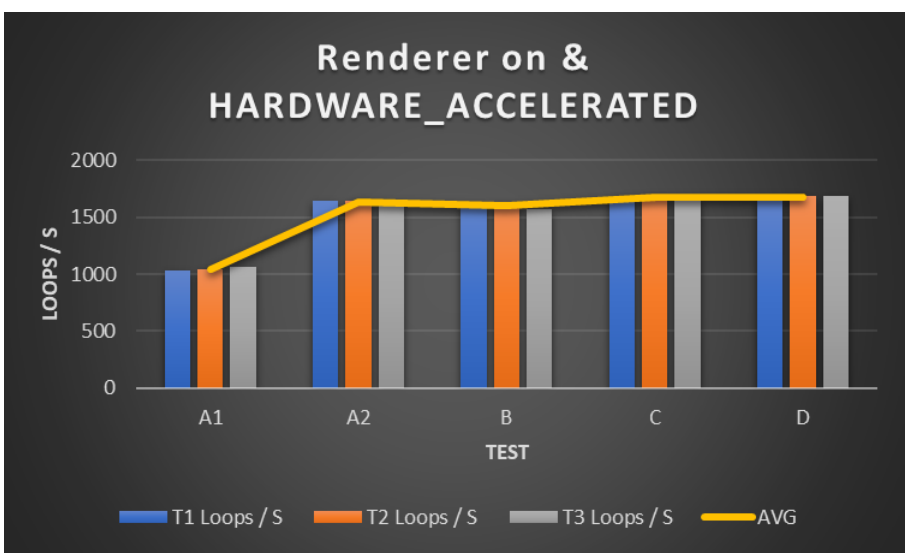



Fig 1.3 Performance impact of collision detection with hardware accelerated rendering enabled.

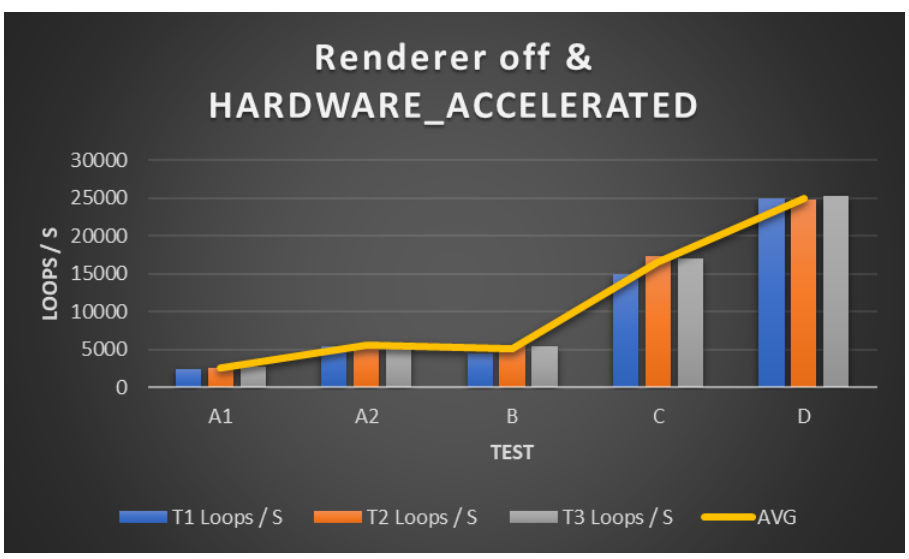



Fig 1.4 Hardware acceleration without the rendering overhead.

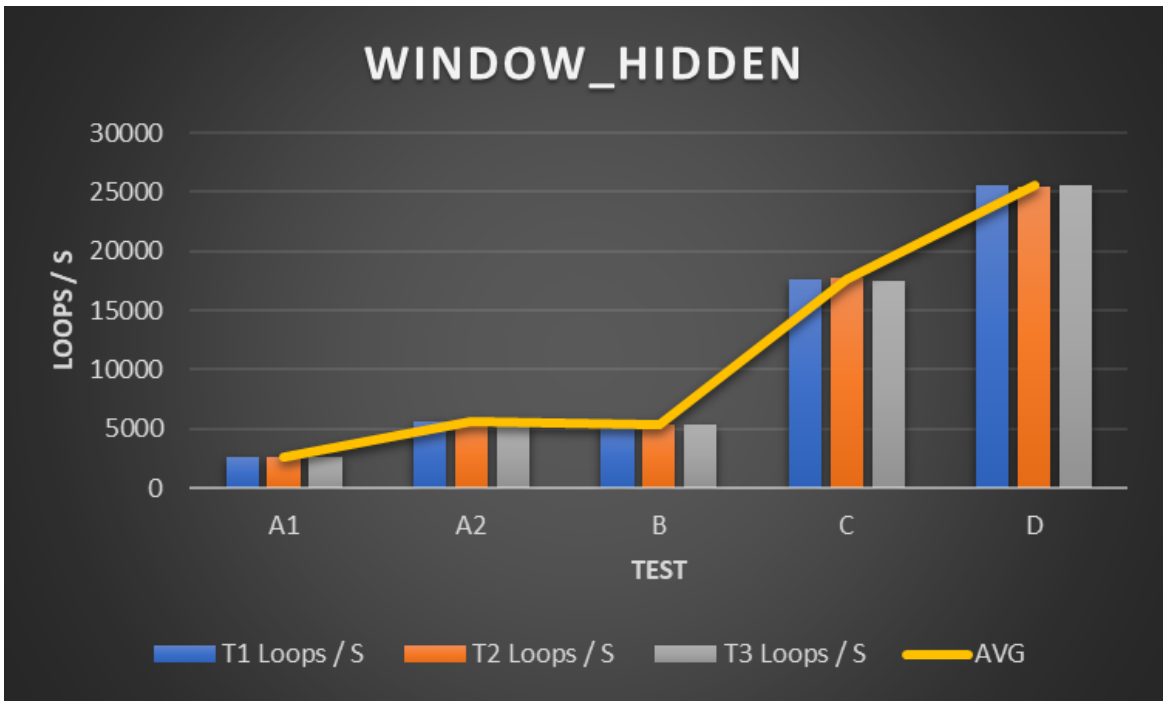



Fig 1.5 Window hidden and rendering disabled

Modifying window and rendering flags as seen in the figures above only yielded an average of 246 more $Loops/S$ when the renderer was active and had no significant performance benefits when the render wasn't being used. This was as expected.

## Collision Optimization

> 💡 If I were optimizing this for a 2D game / engine, I would implement a binary space partitioning system to localize all of the collisions to a node of a traversable binary tree. This would see significant performance gains for much larger entity counts. However due to time restraints, this isn't going to happen and smaller optimizations will be attempted and documented.

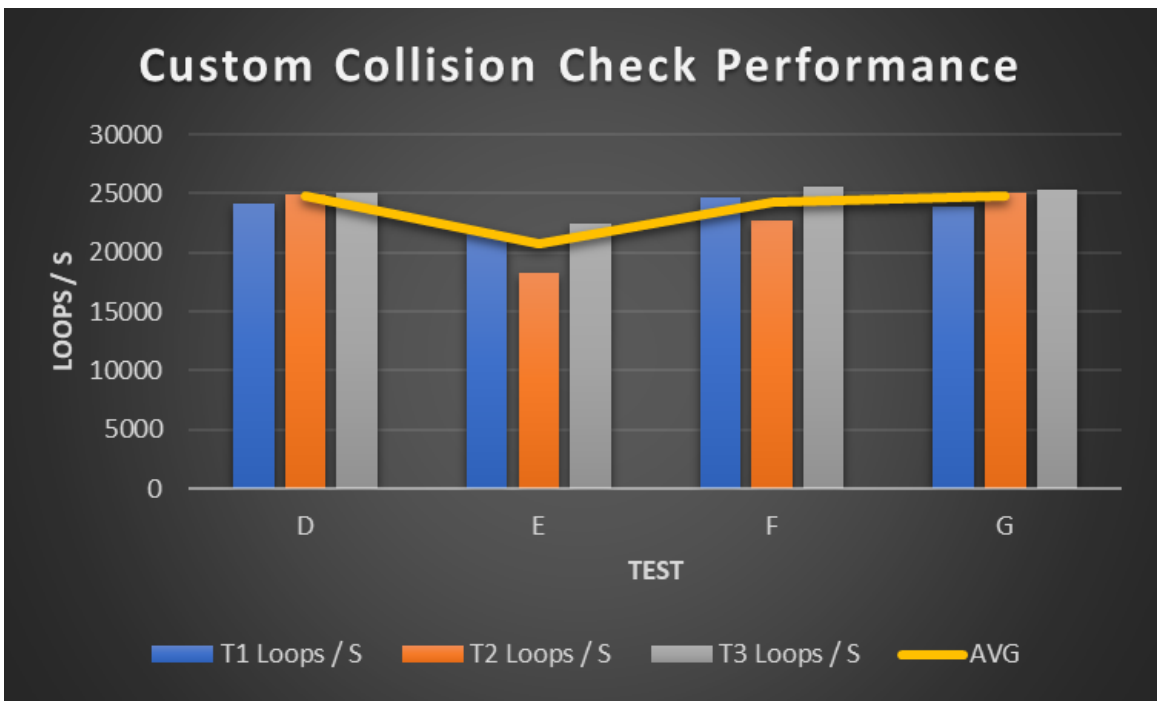



Fig 1.6 Custom collision functions E - F (described above) vs crash_test_all_D( ), the most performant method.

Out of the 3 custom collision functions, the most performant was crash_test_all_G( ) which implemented caching and cut the if block from the detection. Though it was consistently more performant than crash_test_all_D( ), the performance was increased by ~ 1% and can be considered negligible.

Implementations E and F proved to be both slower and less consistent than D, this could be a result of increased cache misses when the prefetcher attempts to guess the outcome of the if-else blocks. It was also noted that crash_test_all_D( ) was the most consistent between it's testing, having a Loops / sec range of 932 (as opposed to it's second test G with 1472).

# Task 24 - Spike Summary Report

## Git Commit History

Commits

History for COS30031-2023-103071494 / 24 - Spike - Profiling, Performance and Optimsation

Commits on Nov 2, 2023

Completed Task 24 - Spike 'Profiling, Performance & Optimization' ...

693e8ab

```
Compiled and ran code
Composed a dataset based of in-code performance metrics
Wrote 3 custom collision methods to compare preformance stats against
Included spike report and code submissions with VC optimization preferences
```

KingSchlock committed now

Commits on Jul 30, 2023

Initial Commit ...

KingSchlock committed on Jul 30

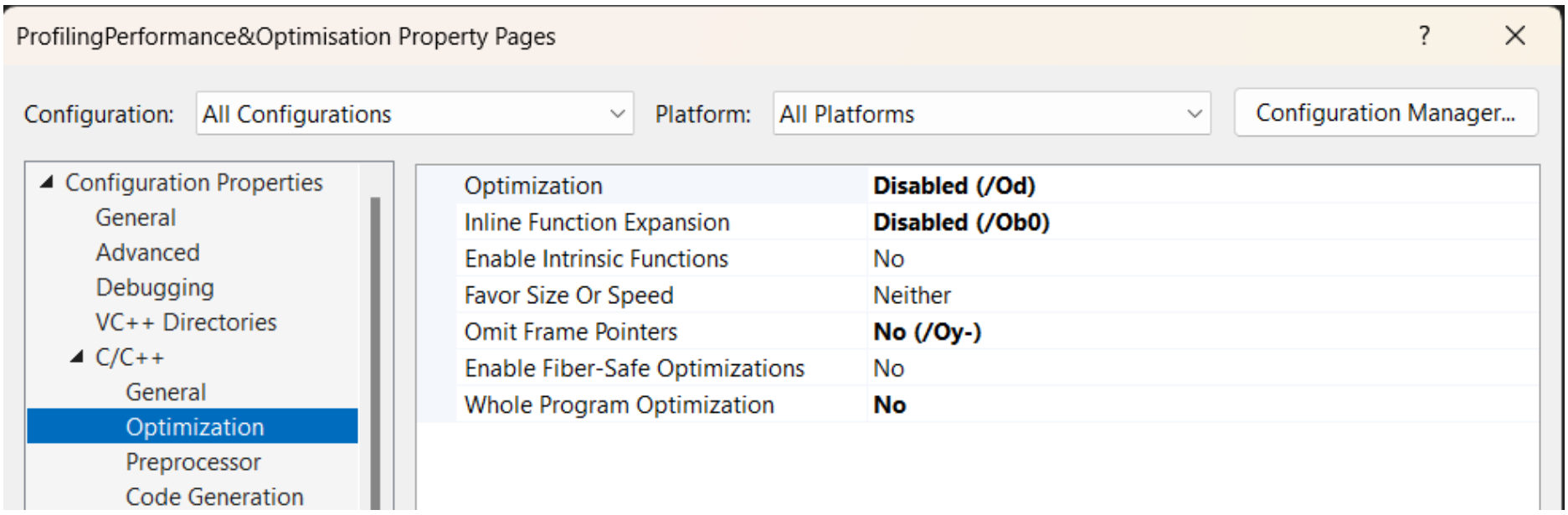
Verified

0d3f501

End of commit history for this file

Newer Older

## Code




Visual Studio optimization configurations

```
208  bool crash_test_E(CrashBox& A, CrashBox& B)
209  {
210      return ((A.y + A.h) <= B.y || A.y >= (B.y + B.h) || (A.x + A.w) <= B.x || A.x >= (B.x + B.w));
211  }
212
213  bool crash_test_F(CrashBox& A, CrashBox& B)
214  {
215      if ((A.y + A.h) <= B.y) return false;
216      else if (A.y >= (B.y + B.h)) return false;
217      else if ((A.x + A.w) <= B.x) return false;
218      else if (A.x >= (B.x + B.w)) return false;
219      else return true;
220  }
```

Fig 2.1 custom individual crash test functions using AABB collision detection

```
310  void crash_test_all_F()
311  {
312      // check i against j
313      for (int i = 0; i < BOX_COUNT; i++) {
314          for (int j = i + 1; j < BOX_COUNT; j++) {
315              if (crash_test_F(boxes[i], boxes[j])) {
316                  boxes[i].state = CONTACT_YES;
317                  boxes[j].state = CONTACT_YES;
318              }
319          }
320      }
321  }
```

Fig 2.2 crash_test_all_F( ) function definition

```
323  void crash_test_all_G()
324  {
325      auto cached_boxes = boxes;
326      int count = BOX_COUNT;
327      // check i against j
328      for (int i = 0; i < count; i++) {
329          for (int j = i + 1; j < count; j++) {
330              if (crash_test_F(cached_boxes[i], cached_boxes[j])) {
331                  cached_boxes[i].state = CONTACT_YES;
332                  cached_boxes[j].state = CONTACT_YES;
333              }
334          }
335      }
336  }
```

Fig 2.3 crash_test_all_G( ) uses 1 realloc to cache a copy of the boxes array. Calls crash_test_F( ).

# What was Learned?

This task involved creating a clean, repeatable testing environment to help increase the accuracy of performance metrics. Additionally, my knowledge of CPU caching and performance analytics increased drastically throughout the duration of this task.