**Spike:** 25

**Title:** Measuring Performance & Optimisations

**Author:** Ben Holmes, 103024841

**Goals / deliverables:**

* Collect base data from collision methods (make sure its at a useful scale)
* Write out info gained from that data, include:
  + A description of each collision test (method) approach (and the differences between them)
  + The method you used to collect your data and your reasons for choosing it
  + The raw data you collect to support your results.
  + A summary table of the results
* Using the measurements, figure about best method and modify to only use that, then modify to attempt to improve.
* Test the modifications.
* Include results of these modifications and rational.

**Technologies, Tools, and Resources used:**

* Excel
* Visual Studio
* Sample code
* Word

**Tasks undertaken:**

* Did first tests (after removing render)
* Made conversion program to convert results.txt to a new txt with csv format
* Grabbed the csv file into excel and made a worksheet,
* Formatted the data and made the graph
* Wrote first half of spike report
* Adjusted code to run best test,
* Created an improved version of the best
* Test both and compare/contrast
* Finish report

**What we found out:**

After doing starting first tests (which will results will be shown below) I created a conversion program to change the results.txt to a csv file as to not spend forever copying across the data one item at a time.

Results of base tests (5 seconds each, ran 40-50 boxes, 5 times for each)

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Average Loops Per Second | | |  |  |  |  |  |  |  |  |  |
|  | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 |
| A | 14072.8 | 13439.4 | 12825.2 | 12268 | 11707.2 | 11100.2 | 10677.6 | 10020.4 | 9762 | 9410.8 | 9053 |
| AA | 28759.8 | 27431.8 | 26139.6 | 24945.4 | 23719.4 | 22377.8 | 21798.8 | 20748 | 19830.8 | 19184.4 | 18422.8 |
| B | 56366.2 | 58740.8 | 55989 | 53442.2 | 50947 | 48575.2 | 46697.8 | 44454.4 | 42599.4 | 41141.2 | 39562.4 |
| C | 170562.8 | 163684.6 | 156574.8 | 149546 | 143288 | 136177.2 | 130784 | 124698.2 | 120547.4 | 115856.2 | 111847 |
| D | 212879.8 | 204838 | 195614 | 186994.6 | 179978 | 171672.8 | 164276.8 | 153556.4 | 151983.4 | 145969.2 | 140641.2 |

(raw data is in ResultsAll5MethodsExcel)

As is clearly shown in the graph D is the best method, although C does come close at the higher end. So I will be modifying D.

Method Explanation:

A: it grabs the boxes from the boxes[] via index then creates each of each box as an int, then compares the corresponding sides for overlap.

A1/2 difference: A1 compares if the indexes are not equal after the collision test.

B: it does the same as A but rather than grabbing the boxes via index, the boxes are passed in via struct copy.

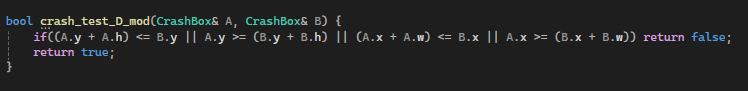
C:is B but using references to the boxes rather than copy

D: doesn’t create the separate edges, just does the same comparison directly in the if statements (also uses struct references)

Improvements attempted:

First:

I tried collating all of the if statements onto one line with OR thinking it may help as it would only have to process one if rather than 4.



It is a small decrease in performance.

Second:

I added the bottom and right edges to the struct, to use in the mod comparison, thinking that having it pre assigned would help.

A computer screen with white text

Description automatically generated

A screen shot of a computer code

Description automatically generated

A computer code on a black background

Description automatically generated

Which was close to the D ran at the same time (more consistent, but would need more data to be sure), but also decreased D’s speed, so didn’t work either. This was because the sides had to be updated each time it moved so it caused performance to drop.

Third:

I spotted a potential minor inefficiency in the update boxes code, so decided to test that (with 10 runs on each).

From:

A computer screen with green text

Description automatically generated

To:

A computer screen with green text

Description automatically generated

Made a difference to consistency and seems to be a minor improvement in performance as expected (no need for a second for loop). Going to keep this change for now.

Fourth:

I made some inline functions for the structures thinking that it would improve performance as it does for many other things, I was very wrong.

A black background with white text

Description automatically generated  
A screen shot of a computer code

Description automatically generated

It halved the performance. I assume because either I implemented them wrong or used them incorrectly.

Summary of attempted changes:

|  |  |  |
| --- | --- | --- |
|  | Average Loops per second | percentage change speed (lower is better) |
| Original D | 140641.2 | 100% |
| Attempt 1 | 133539.6 | 105.32% |
| Attempt 2 | 138328.2 | 101.67% |
| Attempt 3 | 145109.7 | 96.92% |
| Attempt 4 | 75037.5 | 187.43% |

**Risks:**

I decided to use the timing code provided rather than the Visual studio profiler, as I felt that at scale, the best method for testing would be to use the prebuilt one but this will have resulted in some minor inaccuracies on the timing.