

Profiling, Performance, Optimization

Name: Shiva Gupta

Student ID: 102262514

Unit Code: COS30031

Task Number: 24

Date: 31/10/2021

Repo link: https://bitbucket.org/ShivaGupta_/cos30031-102262514/src/master/24%20-%20Spik%20-%20Profiling%2C%20Performance%20and%20Optimsation/

Environment

This section contains information about the factors that may affect the performance.

- Battery: Connected
 - Programs Running: VS Studio
 - Compiler Optimizations: Off
 - IDE Configuration: Debug 32-bit
-

Goals/Deliverables

[CODE] + [SPIKE REPORT] + [SHORT REPORT]

Tasks Undertaken

Compiler Optimizations

[Resource: <https://docs.microsoft.com/en-us/cpp/build/reference/compiler-options-listed-by-category?view=msvc-160>]

Compiler optimizations can affect the performance of the code by a great extent, Hence, to analyze the true compilation time for the code, these can be turned off as follows.

1. We can do this by **right clicking on the project -> Configuration properties -> C/C++ -> Optimization -> Optimization = Disabled.**

There are different compiler flags that can affect the performance and the most useful for us are as follows.

Compiler Flag	Purpose
<code>/o1</code>	Creates small code.
<code>/o2</code>	Creates fast code.
<code>/od</code>	Disables Optimization.
<code>/os</code>	Favors small code.
<code>/ot</code>	Favors fast code.

`small code` here refers to space optimization.

IDE Configurations

[Resource: <https://docs.microsoft.com/en-us/visualstudio/debugger/how-to-set-debug-and-release-configurations?view=vs-2019>]

IDE configurations can also severely affect the performance of the code. There is a significant overhead for the `Debug` configurations. The code runs faster in `Release configurations` but for the purpose of this task we will use the debug config for all the tests performed and disable the compiler optimizations.

Profiling Approach

Collecting Data

The profiling approach used for this task is to store the test times in a `.csv` file. This approach has been selected because of the developer's familiarity with this approach in task 7. This will help us to draw conclusions and chart them appropriately using MS Excel functions. Accuracy and quality analysis are guaranteed through this approach with minimal overhead in the performance.

2. We can use the `fstream` library to our advantage and simply output the results of the tests to the `.csv` file which can then be opened and charted in MS Excel.

```
//Sample program to show the profiling approach used
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
    ofstream myFile;
    myFile.open("test1.csv");

    myFile << "Col1, " << "Col2, " << "Col3, " << endl;
    myFile << "v1, " << "v2, " << "v3, " << endl;

    myFile.close();

    return 0;
}
```

```
}
```

File checking and error handling are deliberately left out.

	A	B	C	D	E
1	Col1	Col2	Col3		
2	v1	v2	v3		
3					
4					
5					

Function to measure time for tests

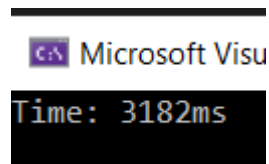
[Resource: https://wiki.libsdl.org/SDL_GetTicks]

To collect the data we need to use a function that can measure the time spent for a method execution with minimal overhead. An option to undertake the task was to use the `std::chrono` library and use its functions to measure time as we did in task 7. But the code provided for testing uses a function within the `SDL` library called `SDL_GetTicks()`.

It measures the time when the `SDL` library was initialized. Since, we need the time difference and not the actual time so we can use this function to collect time data for the tests.

A sample approach to show the functioning of `SDL_GetTicks()` is as follows.

```
start_time = SDL_GetTicks();  
cout << "Time: " << start_time << "ms" << endl;
```



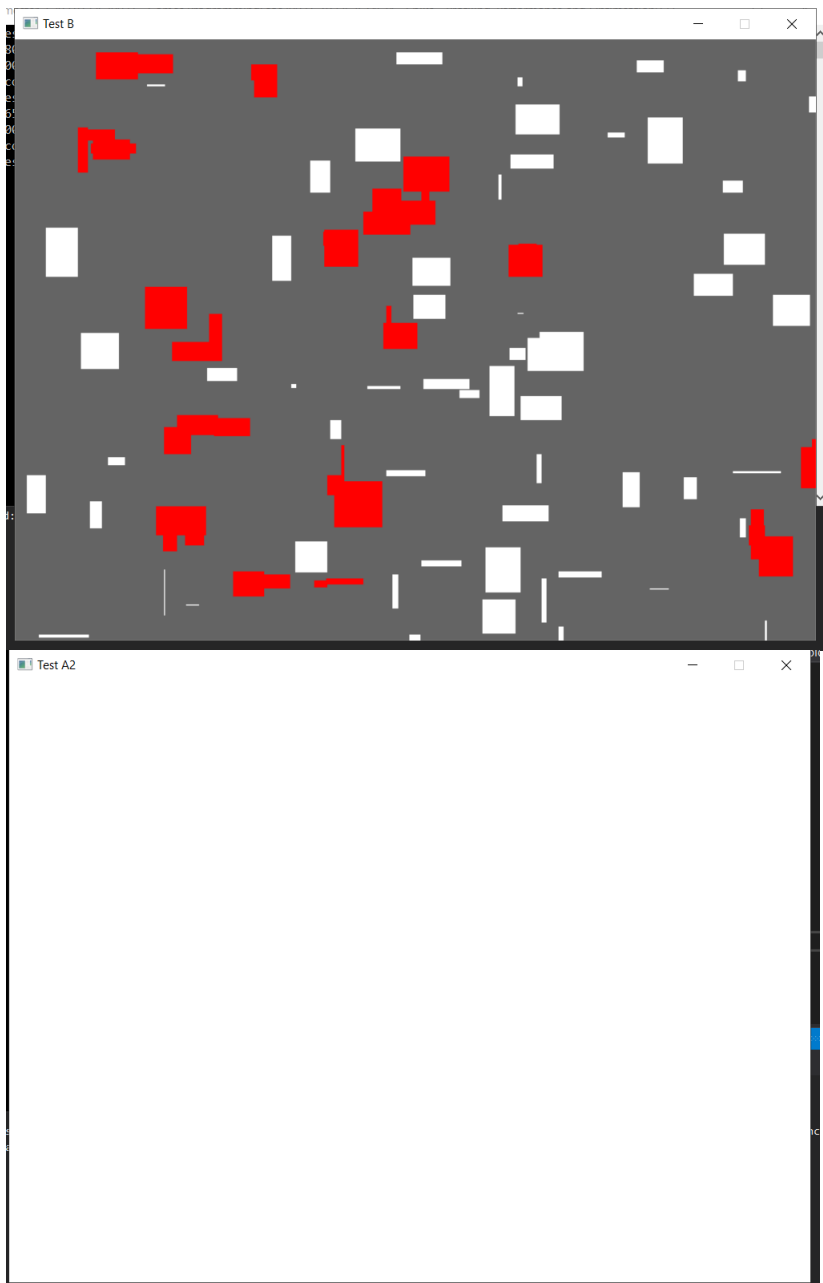
Rendering Overhead

The task requirements state that the rendering should be turned OFF while collecting test results which seems apt because we want to decrease the overhead as much as possible.

3. A simple approach to turn off rendering is to set the render loop to `false`.

```
// RENDER  
if (false) {  
    // 1. clear the background  
    SDL_SetRenderDrawColor(renderer, 100, 100, 100, 255);  
    SDL_RenderClear(renderer);  
    // 2. render all boxes  
    render_boxes(renderer);  
    // 3. show it  
    SDL_RenderPresent(renderer);  
}
```

Difference between rendering ON and OFF.



style="zoom:60%;" />

Function Pointers

The code in this task uses function pointers to test different ways of box collisions. The main advantages of function pointers are that they provide a way by which we can assign functions to a variable and we can pass functions into another functions as parameters.

A simple example of function pointers is as follows:

```
void PrintStatus(const string& status) {
    cout << "The current status of the machine is: " << status << endl;
}

int main() {
    void(*function)(const string& status);
    function = PrintStatus;

    function("running");
}
```

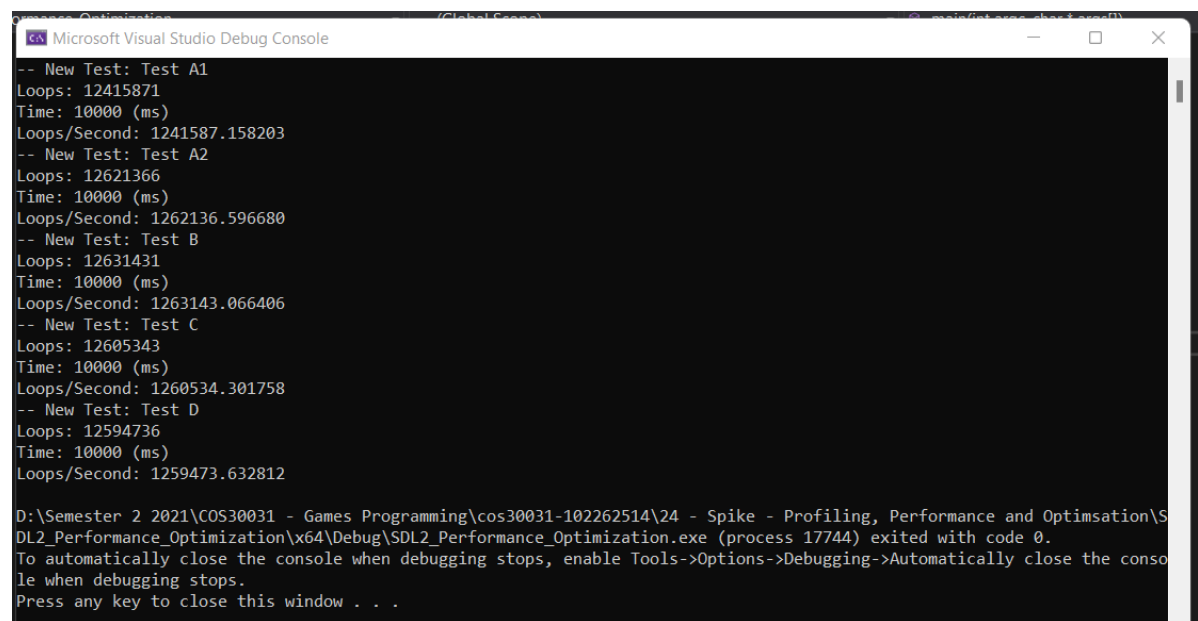
```
function("paused");  
function("stopped");  
  
return 0;  
}
```

Microsoft Visual Studio Debug Console


```
The current status of the machine is: running  
The current status of the machine is: paused  
The current status of the machine is: stopped
```

What we found out


During this task, we learnt how to measure performance of different collision testing approaches. The approach we adopted helped us to collect data efficiently and with low overhead.





```
Microsoft Visual Studio Debug Console  
-- New Test: Test A1  
Loops: 12415871  
Time: 10000 (ms)  
Loops/Second: 1241587.158203  
-- New Test: Test A2  
Loops: 12621366  
Time: 10000 (ms)  
Loops/Second: 1262136.596680  
-- New Test: Test B  
Loops: 12631431  
Time: 10000 (ms)  
Loops/Second: 1263143.066406  
-- New Test: Test C  
Loops: 12605343  
Time: 10000 (ms)  
Loops/Second: 1260534.301758  
-- New Test: Test D  
Loops: 12594736  
Time: 10000 (ms)  
Loops/Second: 1259473.632812  
  
D:\Semester 2 2021\COS30031 - Games Programming\cos30031-102262514\24 - Spike - Profiling, Performance and Optimisation\SDL2_Performance_Optimization\x64\Debug\SDL2_Performance_Optimization.exe (process 17744) exited with code 0.  
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.  
Press any key to close this window . . .
```


 COS30031 - 102262514


<> Source


 Commits


 Branches


 Pull requests


 Pipelines

 Deployments

 Jira issues

















 Security

 Downloads

 Repository settings

Search commits

All branches

Author	Commit	Message	Date
 Shiva Gupta	bbc0e35	[MODIFY]: task 24 code and results	38 seconds ago
 Shiva Gupta	bab539f	[ADD]: task 19 spike report	2 days ago
 Shiva Gupta	5a7c83a	[MODIFY]: task 19 exception fixed	2 days ago
 Shiva Gupta	d7393e9	[ADD]: task 13 report complete task	3 days ago
 Shiva Gupta	4ead830	[ADD]: task 12 complete with reports	3 days ago
 Shiva Gupta	a337ab6	[ADD]: task 12 UML and code	3 days ago
 Shiva Gupta	9475f3a	[MODIFY]: task 13 code complete	3 days ago
 Shiva Gupta	ef4c31e	[MODIFY]: task 19 dispatcher code	4 days ago
 Shiva Gupta	da67906	[ADD]: task 19 blackboard complete	4 days ago
 Shiva Gupta	555e9ea	[ADD]: task 19 setup	4 days ago
 Shiva Gupta	543fd8d	[MODIFY]: task 13 command take, put, look, etc. ...	4 days ago
 Shiva Gupta	631a566	[MODIFY]: Take commands for task 13	4 days ago
 Shiva Gupta	6a8c99e	[ADD]: code has items that contain other items	5 days ago
 Shiva Gupta	f805d5b	[ADD]: code command processer complete	6 days ago
 Shiva Gupta	3985ef8	[MODIFY]: move command in task 12	7 days ago
 Shiva Gupta	93abe74	[ADD]: task 22 video and spike report	2021-10-28