Optimizing in C++ using Assembly Language and using inline Assembly

By

DeVaughn Brown-Battle

Project Goal: The goal of this project was to spot optimize sections of the game loop using DarkGDK development kit and C++ in visual studio to decrease input bottleneck using inline assembly for the 1st time.

Why is using assembly language relevant in today’s age of programming?

All types of code Python, C++, NodeJS at some point has to be translated to be processed by the CPU and usually when code is processed it has to go to the compiler then it is translated into assembly language variables, functions, then into assembly language instructions to be processed by the CPU.

Having the actual knowledge to code in Assembly Language allows you to not only know how the computer runs your code but also allows you to shorten this process as now you are communicating directly with the CPU.

Although coding an entire program in Assembly Language is not always advised due to the high amount of maintenance using inline assembly can be very advantageous.

Benefits of inline assembly:

Good choice for spot optimization as Assembly Language is a direct communication to the CPU.

Assembly Language can also help take advantage of a CPU’s clock processing power to optimize an application due to operating at the instruction level.

Optimizing in assembly can also be easier to read for experienced programmers and could use less lines of actual code.

CPU specific instructions for different CPUs could offer special instructions to complete programming tasks which is very valuable if you want users to be able to take advantage of the processing options they have available for the code.

Why is inline assembly more valuable then maybe intrinsic functions or vectors?

Intrinsic functions might still optimize a section of code however it can become less readable and modifiable if a programmer has less knowledge about the code and at the Assembly instruction level could still translate to more lines of code to run and execute. Vectors can combine instructions well especially for libraries however it can still be large amounts of code as Vectors combine sections of code it’s not with a direct focus of execution time and can still be prone to potential bugs based on vector setup. Assembly Language can put a direct focus on cutting execution time due to direct communication with the CPU and cut down directly on bottleneck due to CPU instruction access and can complete special tasks not modifiable with high level programming. The only challenge is code maintenance and error checking is more in the hands of the programmer as well as variable and function renaming for Assembly Language instructions or inline Assembly.

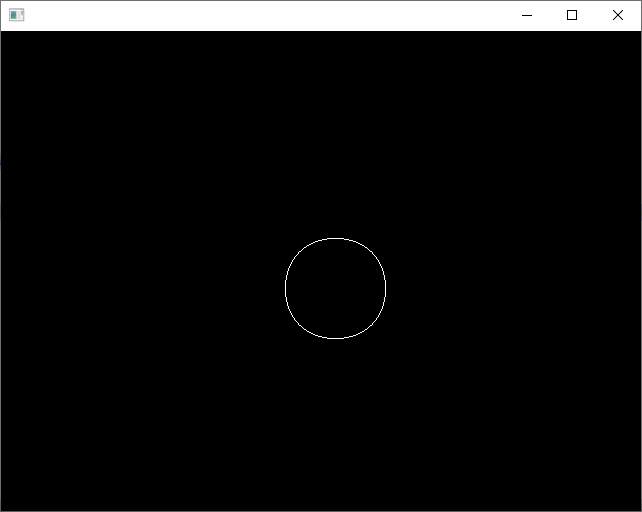
As a programmer why is knowing Assembly Language or inline Assembly Language valuable?

Truth is only about 5% of today’s programmers actually know what assembly language (.ASM) is and even if a programmer or a developer is aware of assembly language even less can actually use it for spot optimization, computer specific instructions, or for special tasks for optimal coding performance so knowing this skill means you can optimize the effectiveness and efficiency of a program in a way most developers can’t.

Programing Title: Moving\_Ball.cpp

Programming Goal: The Moving Ball program essentially performs a basic feature move a ball or circle based on the pressed Up, Down, Left, Right arrow keys however there is initially a small bottleneck delay in all key strokes. So the goal is to decrease this key stroke bottleneck to execute a smooth move control scheme and track the key stroke times.

Below is a screenshot of the program



Doesn’t seem like much however this is an input based project to get the full code you will need to use to below download links.

DarkGDK, Visual Studio 2008, and Direct X 9 download link:

https://drive.google.com/file/d/1l-PbPw9ELlthlM5PREHeH1mT8sBhH1xp/view?usp=sharing

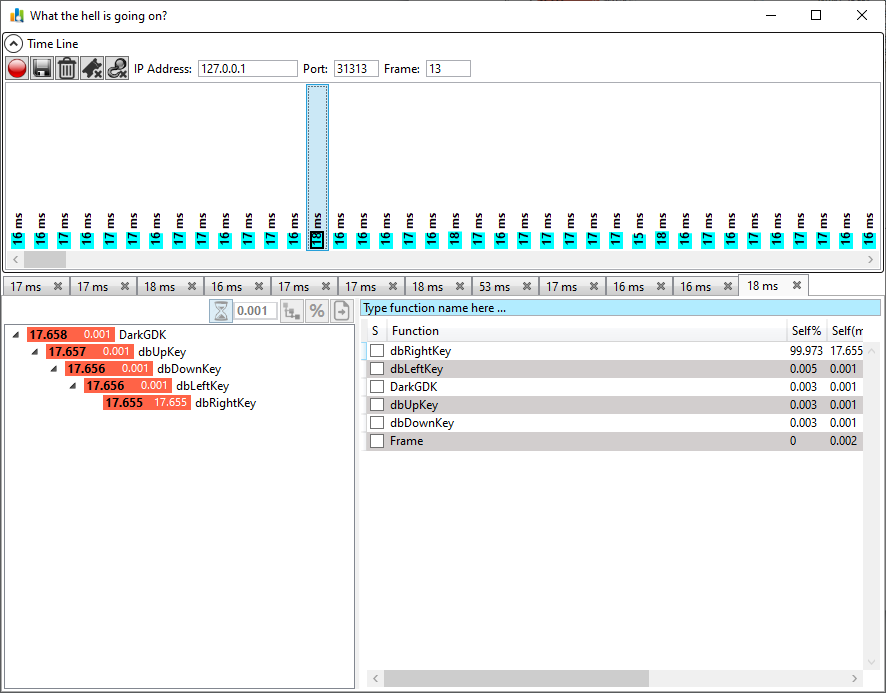
Original Source Code Download link:

https://drive.google.com/file/d/1NrC5xqnmYdioL8maeobedT9eKBGUgoOs/view?usp=sharing

NOTE: All software is free software and Windows 10 tested

The original Moving BallCPP source code can be copied into an empty C++ project in visual studio 2008 and with the DarkGDK download instructions and files you should be able to run the full code.

Below is the original performance profile of the program:



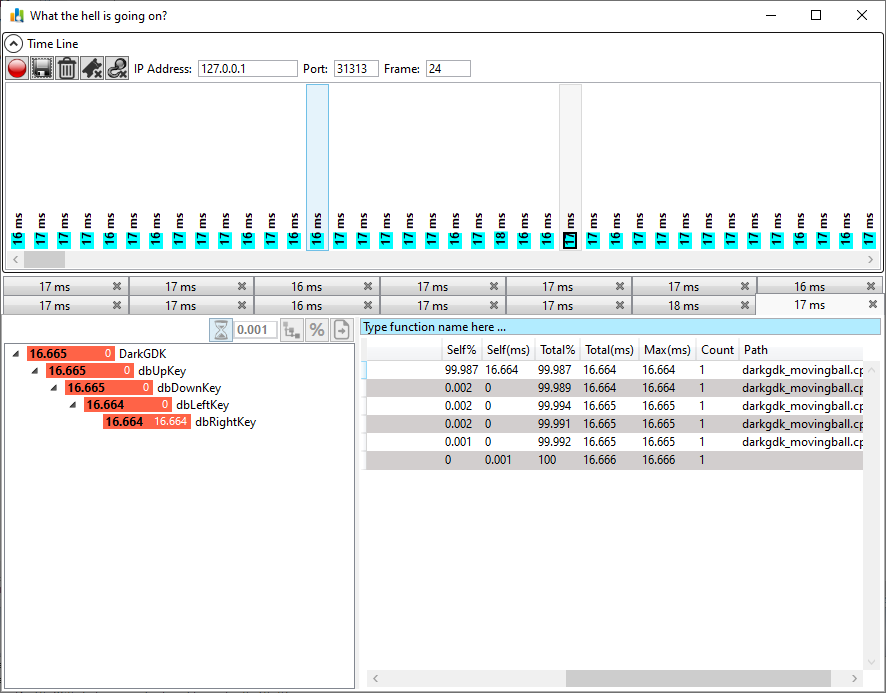
Here my program reaches the top bottleneck of 18ms after 14 frames with a total reach of the highest screen captured frames of 18ms 3 times for all Up, Down, Left, and Right key inputs not bad but sluggish for the user key input bottleneck.

Below is the optimized version of the same program using inline assembly in my C++ code.

Download Link:

https://drive.google.com/file/d/12ujZfDmqaJxIVfO8rDbIBecheal8a7VN/view?usp=sharing

Below is a screenshot of the profile of the optimized version of my program.



Here my program now displays an input of 16ms after 14 frames with a total reach of the highest screen captured frames of 18ms only 1 time for all Up, Down, Left, and Right key inputs roughly a 2ms difference and more steady input response times from a bird’s eye view this doesn’t seem like much however for a user especially for game programs this 2ms difference with a more smoother response time can be night and day for the success of a program or game this could also be more if I optimize for clocks on a CPU but that would be better for graphics and level specific optimizations.

Summary:

Inline Assembly and Assembly language is an extremely valuable concept and rare in today’s age of programming as it’s considered a lost art but if used offers rare performance and special task options not available to many types of programs and developers. Having this knowledge can put you in a rare section of developers and can help nearly any program uniquely achieve success as opposed to other programs that don’t use these concepts.