Thread-Pooling Architecture in a Custom-Made Game Engine

COMP 8045 – Major Project 1

JJ – A00000000
8-28-2018

Table of Contents

1. Student Background

A Junior programmer with 4 years' experience at British Columbia Institute of Technology. I possess a number of skills that I will utilize with various programs and languages. I am a hard-worker and coupled with a positive attitude, can work well independently or cooperatively within a team environment.

1.1 Skills

- Languages: C++, C, Objective C, C#, Java, HTML, JavaScript
- Tools: Visual Studio, Net Beans, Git, Android Studio, XCode
- Editors: Unity3D, Maya, Blender
- Created multiple Video Game projects during enrollment
- Skills with developing Mobile apps on Android and Apple devices
- Considerable team-working experience

1.2. Education

British Columbia institute of Technology (BCIT) - Burnaby

- **Bachelor of Technology** **Graduating January 2019**
  - Games Development Option
- **Diploma of Technology** **Graduated 2016**
  - Computer System Digital Processing

1.3. Work Experience

- **Gentek Building Supplies** **May – November 2013, May – Sept 2014,**
  - Delivery Man / Supply Associate **April – September 2015, April – Sept 2016**
  - Dealt with customers and orders **April – September 2017, April - Present**

2. Project Description

The purpose of this project is to explore one of the many concepts that are commonly used within the gaming industry today. Specifically, I refer to the concept of the Thread-Pool design pattern. My objective is to use it as the base architecture to build a game engine from inception. With this design pattern, my engine will be taking advantage of modern computers multiple CPU cores and provide greater performance than currently on a basic game engine. Along-side the engine, I will be making a demonstration by using a plethora of 3D game objects to be interacting with each other in the game world.

3. Problem Statement and Background

3.1. Problem

The problem the Thread-Pool concept alleviates are *the limitations of having a single threaded game engine that has a lot of objects to process in the short span of a single frame that causes slow downs to occur.* Games nowadays need to be using all of the power a desktop or game console can provide to keep up with all of the objects in the game world. If not, there will be a noticeable drop in the frames due to the numerous numbers of objects the computer has to calculate. Players can and will notice when the game is stuttering, and if it gets too bad they will just quit playing the game altogether. Therefore, keeping a steady frame rate is quintessential in building a game and keeping the player happy. ***Using the Thread-Pool Concept, the game engine will be able to utilize more of the computer's hardware to give the game engine greater computing power alleviating slow processing.***

3.2. Background

In the project for the game's development course, our groups were tasked with creating a game engine that utilized multi-threading. Our engines had many different components (a rendering, physics, A.I and many more components) that needed to work together. Most of the engines we made used a different form of threading. These forms comprised of assigning threads to each single component. When the game demanded a certain function requiring physics, it would send a message to the appropriate thread that runs the function. This proved not to be the best way of using the threads on the computer as there was a lot of wait time in between all of the functions. However, it did provide a good introduction of how to deal with threads in an engine. The best way of handling the threads would be to create all the threads that are needed on the start of the engine. Next would be to run the threads continuously on idle and wait for a "Job" to be handed to the thread. With multiple threads working on these Jobs, the system will utilize more of the computer's hardware and increase the number of calculations that it can do.

Another way of dealing with the abundance of calculations that games need to do is to put most of them on the GPU. The GPU can do a lot of calculations rapidly and can easily surpass the CPU cores abilities. However, one of the major draw backs is that we would want to keep the calculations off the graphics card because it needs to handle all the rendering when it needs to display on the screen. If this was a simple demonstration of physics only then I could see the need to do GPU calculations. Nonetheless, I want this to be a full-fledged game engine by the end of the project time that can do a myriad of different systems, like physics, AI, networking, rendering and much more.

4. Game Engine

I will be creating a game engine from inception and will encapsulate the following features: Thread-Pool architecture, Job System, Rendering and Physics.

4.1. Thread-pool

At the start of the game engine, I will be focusing on getting a series of threads working simultaneously and asynchronously. The biggest challenge will be having all of the threads working separately and keeping all the data within the game correct. For example, two threads can't work on a single objects physics component as they will overwrite each others' data. This would be a major race condition that could happen a lot that I will need to mitigate. As each of the threads are running within the game, they will be handed a Job on the Job queue to complete. Once the thread is done with its Job, it discards the Job and either check to see if there is another Job on the queue or go to sleep for a set amount of time or a condition variable in order to let other threads to work on their Jobs in the meantime. *Figure 1* depicts what a thread will do upon initialization to finish. Once the thread initializes, it then moves to the idle location to wait for a Job. If there is no Job for the Thread to do, then the thread goes to sleep to allow other threads to run. If the re is a Job, the thread takes it, executes and returns to the idle position to wait for a new Job. This diagram is what the threads will be doing the whole time the game is running.

Next, to keep a solid frame-rate the game engine will need to limit the amount time the Jobs can work on the Jobs in the queue. Let's take an example of 30 frames-per-second and calculate the amount of time the threads will need to complete all the available Jobs in about 0.033 seconds (or 33 milliseconds). So, when the game engine runs through a single frame most Jobs need to be complete within the 33 milliseconds. In this project, I plan to build a scheduler that will keep track of the timing of all the functions that the game needs to run. With it I can get Jobs as closely to the time limit as possible and halt the threads when the time comes to draw for rendering or have the threads start work on the next frame. The Scheduler will be built from the ground up using the C++ standard library to keep track of all the functions runtimes.

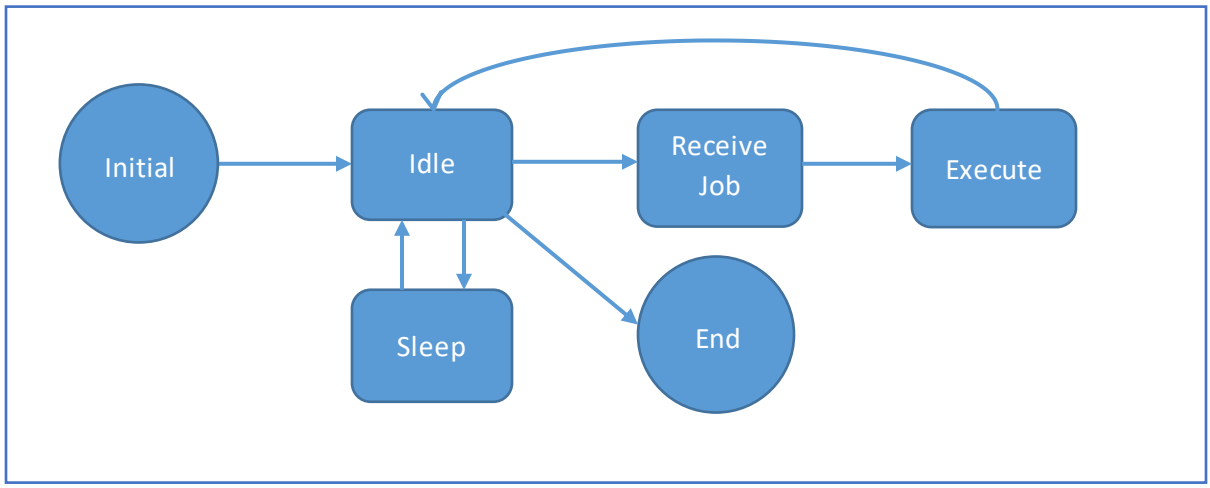


Figure 1

4.2. Job System

The 'Jobs' that the engine will be creating, and managing will be a variety of functions that the game needs to calculate in the duration of a single frame of the game such as collision detection, player input, rendering and much more. When combined with the Thread-pool system, the game engine will be able do a lot more calculations within the span of a single frame. However, some Jobs are going to be much more complex in nature than others. Such as Jobs that require another Job to finish its calculations first before it can finish its own calculations. Most of the Jobs are going to have to be in a particular order. Therefore, some form of sorting is going to have to organize these Jobs to make certain that Jobs that are more important go first. For example, rendering Jobs like fulcrum culling will have to go last because the positions of the objects in the game world need to be updated first before they are displayed on the screen.

In order to do this, I will be building a system where the game and threads can add jobs onto the queue in the manager. Then, the manager takes those jobs and figures out which thread will be able to fit it within the time. Through the Scheduler, the engine can determine the best location for the Job to go to depending on the time and the priority level.

Along with the Job system, I will include some way to visualize it with the SDL2 library as it is being used to build the game anyways. This way I can get an accurate representation of what is happening behind the scenes for the threads in the game. This will require taking a snapshot of a single frame and allowing the threads to run the Jobs they are getting until the end of the frame. Next, the game will gather all the information from the threads and display it using in the game window. The graph will include when the threads will start and stop a Job, what time they start sleeping and length of time in total for a single frame to complete. A complete graph of what the system and threads are doing at specific times will be a great addition.

4.3. Render, Input and Physics

In the game engine, I will need to utilize the architecture that I have build and display it for players to see. In order to do that I will be developing three different aspects to the game to have as a proof of concept. Rendering will handle all of the drawing to the screen and shader portion of the game. This will mainly use the SDL2 library to display the game. Furthermore, SDL2 provides an input reader as the player wants to take actions in the game as well. So, I will be using it for the input reading as the proof of concept is being shown. However, just having objects displaying won't provide enough of the necessary calculations in order to test the system. Using the Bullet library will provide a lot of functionality to generate the needed Jobs for the system to take on.

4.4. Proof of Concept

The final product of the game will be a fully functional engine that is multithreaded. From those threads, I will make a system that will allow those threads to handle many different types of Jobs under the time limit constraint. Furthermore, a visible graph that depicts a snapshot frame of what the threads have been doing and the Jobs they have performed.

As a final proof of concept for proving that the engine is working and fully functional. There will be a demonstration of many objects within the game world that will collide with each other and respond

appropriately. The number of objects that the system will need to handle will have to be worked on while I am building the project. However, the minimum number of objects that the system will have to handle will be the what the computer can handle when there is no multithreading. This will allow for an adequate test of the systems limitations.

5. Innovation

The Thread-Pool concept has been done many times in the past when trying to utilize a computers hardware to get the most out of the CPU's. Which means that the concept is more of an exploration that is commonly used throughout the industry rather than an innovation. However, what I have read up on thread-pooling is that Jobs are handed one after the other. I am proposing a different idea on how to handle the execution Jobs in the system.

This idea is to have each thread have its own list or queue of Jobs that needs to be completed. As the threads are finishing their current Job and remove it from the system. It will be able to grab a Job right away from the queue or list instead of waiting for the main thread to finish whatever Job it was doing. This can decrease the time waiting for new Jobs and allow the threads to work much more frequently. Furthermore, I can adjust the lists to take Jobs away from it or insert new higher priority Jobs that need to be completed. Some of the downsides will be the memory cost of having multiple lists and the main thread will have to do more computation as to allocating what Jobs can go on what thread's list.

6. Complexity

The Thread-Pool architecture in the game engine is going to be the most complex component. To make sure this design works I will need to make sure that all the threads process the Jobs correctly and are able to work on the appropriate information. The challenge will come with certain fallbacks of multi-threading programming in which race condition are most likely to occur in the development of the engine. I will have to ensure that the information provided to each of the Jobs are kept separate so that there is no overwriting of data.

A lot of companies have used some form of threading in their games in order to get the most out of the CPU cores. There is a video on Youtube from the Company Bungie, makers of the game Destiny, giving lectures about their multi-threading design patterns and the complexity of what it really takes for it to work. [1]

7. Scope

The Scope of this project is to build a fully realized and working game engine with the features and functionalities below. Each section is sorted by the respective core features of both the Game Engine and the Thread-Pool design.

Game Engine -

Main Features	**Sub-Features**	**Description**
Rendering	3D	3D rendering of objects in the game world to display on screen.
	Vertex + Fragment Shaders	Shader used to render the object on screen with OpenGL's shader code GLSL.
Input	General	The Input for the engine is going to be basic movement for the player which includes the WASD keys for directional movements and Mouse Control which will control the camera.
Physics	General	I will be using the Bullet library to do the physics calculations which include Collision Detection, Rigid Body, Acceleration and Velocity.

Thread-Pool -

Functionalities	**Sub-Functionalities**	**Description**
General	Scheduling	This will allow the game engine to allot time slots for Jobs to precisely divide up the time from the start of the frame to the end.
	Managing	Allocates Jobs to the threads when they need work to do. The manager can manage the whole set of lists and ensure that the threads are running at the maximum potential.
Threading	Asynchronous	Allow threads to continuously run without interruption and over reliance on other threads.
	Avoid Race Conditions	Make sure that game doesn't override each other components when working on the same data. Also, avoid working on the same data
	Avoid using mutex + semaphores	Mutexes and semaphores lock threads out of being able to do certain functions on an object if another thread is work around that same function. This can lead to a slow down of desperately needed time for calculations. Going to have to mitigate the use of them.
	Timing	Timing is referring to the time that a function takes to execute when given to a thread and the time that a frame needs to be completed. When I have both the aforementioned elements I can create a

		system where the engine can use the most out of the threads with little sleep time.
Jobs	Dependant Jobs	If one Job needs another Job to finish first, then it should be pushed back onto the queue and quit for the first Job to finish. Then once the other Job is finished then the first on can run.
	Priority	When Jobs are created, all of them are equal. With a priority setting the Jobs can be set to a number of importance and given to threads earlier before other ones.
	Sorting	This is only done once the priority numbers are done, then the Jobs can be sorted by order of priority.
	Interruption	If at anytime we need to stop a thread, and have it return to do a more important Job. The engine will also stop it at the end of the frame timing.

8. Test Plan

All on the test plans will be all done with Hands-on testing and will be following a series of test scenarios to make sure that the functionality works.

Test #	Description	Feature / Functionality
Test #1	Verify that the threads successfully run a Job. Test: Have thread print when Job is started and finished Pass: Both the start and finish of the Job are printed Fail: Neither the start nor finish of the Job are printed	Thread-Pool: Managing
Test #2	Verify that threads finish all Jobs within specified time. Test: Have threads finish all Jobs and use timer to print out time Pass: Time is under reasonable amount Fail: Either takes too much time to finish all Jobs or doesn't print anything	Thread-Pool: Timing
Test #3	Verify the number of completed Jobs are the same as the number of Jobs that have gone into the queue. Test: Have system increment counts and print them at the end of a frame. Pass: Counts are the same Fail: Counts are not the same	Thread-Pool: Managing
Test #4	Verify that the most important Job has been taken first in the queue. Test: Have threads print what Job they are on and level of importance Pass: Thread has the highest level of importance Fail: None of the threads have the most important Job	Thread-Pool: Importance + Sorting
Test #5	Verify that the threads pause or stop when the game is paused or minimized. Test: Have user pause + minimize the game when playing Pass: Threads spin-lock or yield Fail: Threads do not spin-lock or yield	Thread-Pool: Pausing + Stopping
Test #6	Verify the objects in game world interact with each other properly. Test: Have game object collide with another game object Pass: Objects collide and react accordingly	Game Engine: Physics

	Fail: Objects do not collide or react accordingly	
Test #7	Verify that the Controls in the game work Pass: W moves forward, A moves left, S moves backwards, D moves right. Furthermore, Mouse controls the camera in the right direction. Fails: Any of the Passing requirements fail.	Game Engine: Input
Test #8	Verify engine display game correctly Pass: Game is displayed on the screen Fail: Game is not displayed, or Game doesn't look right (misalignment, incorrect coloring, etc.)	Game Engine: Render
Test #9	Verify Scheduler correctly analyzes times and sorts jobs in correct order between priority and time Pass: Order of Jobs is listed correctly by Priority and Time of function Fail: Not in the correct order or scheduler doesn't work	Thread-Pool: Scheduling
Test #10	Verify that the game can handle a large number of obstacles on the screen without slowing of frames. Test: Generate large number of objects on the screen to process Pass: Frames are consistent even with a large number of objects Fail: Frames slow down.	Thread-Pool: Managing

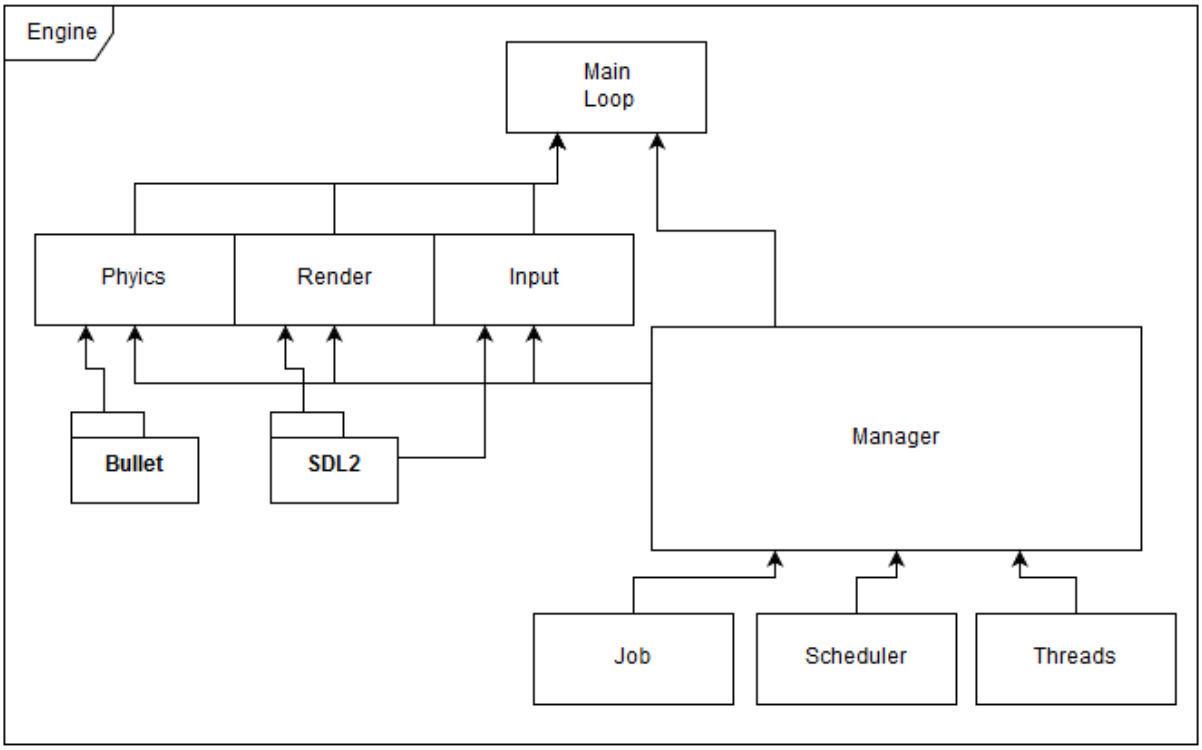
9. Methodology

The methodology I will be using during this project will be the Simplified Agile Methodology.

- Planning of Concept
- Requirements Needed
- Design
- Coding
- Testing
- Documenting
- Version Control

All of these will repeat when I have reached a satisfied state where I believe that what I have finished is good enough to be a completed version. Given that I have a limited amount of time, I will be constraining myself to the development schedule and milestones for the time frame to work with.

10. System/Software Architecture Diagram



11. Technical Challenges

Many technical challenges will be coming from the combination of a game engine with the base architecture of Thread-Pooling. They present as follows:

- **Performance + Stability**: When I get to the point where I can start making a display of the game on the computer, I will have to make sure that the game runs at a smooth frame rate and doesn't stutter. This includes the large number of obstacles that the proof-of-concept will be creating and calculating.

- **Timing**: I will need to figure out how to time all functions in a quick and non-intrusive way. Furthermore, threads will execute these functions within a limited amount of time such as the time it takes for a frame to finish. This is important because if a Job is going to take a long time to complete then they should have a higher priority earlier on in the program than if it was later in the program.

- **Sorting and Priority**: I will need to develop a system in order to assign every Job a level of importance relative to where the engine is at the frame. This will involve sorting through the queue of Jobs to prioritize the correct sequence of Jobs. I foresee this as a great learning opportunity.

- **Debugging Threads**: Debugging just one additional thread is a challenge in of itself. I will have to figure out a way to debug N number of threads where any of them could be anywhere at anytime.

- **Avoid using mutexes and semaphores**: As known in usual multi-threading courses mutexes and semaphores are a must when it comes to threading. However, I have learned that game makers should avoid whenever possible to using them because of the waiting that the other threads could be doing in the meanwhile.

12. Technologies

Some of the technologies and libraries that I will be using for my project are:

- C++
- OpenGL
- SDL2
- Bullet
- GitHub.

C++ is widely used within the game industry and is a robust language that is able to use a ton of libraries. As for the graphics of my game engine, I will be using a combination of both OpenGL and SDL2. OpenGL has many funcitionalities and modern tools for game developers to use for their games. It can render both 2D and 3D graphics which makes it a good match to use for my project. SDL2 stands for Simple DirectMedia Player is another robust library that has any functionalities that a game developer needs such as audio, video, keyboard, controller and more. Bullet is a physics library that I will be using for my game to calculate all the physics that I need like collision detection and acceleration. Last, GitHub will serve as my version control in order to keep track of all the changes that I have made to the game engine along the way.

13. Development Schedule and Milestones

All of the milestones can be considered under the game engine category as that is what I have chosen what to focus on during this project. Nonetheless, the milestone names are just to keep it clear and concise for the reader. Times are also going to be adjustable as the project goes on as these are rough estimates. Certain Milestones could take longer, and others could take up less time.

Milestones	Component	Hours (Estimated)
Project Initialization	• New Project (Visual Studio) • GitHub Setup • Libraries imported (SDL2, Bullet)	5
Thread-Pool (1)	• Asynchronous Threads • Threads Run and Gets Single Job • Runs Function + Return Correct Results • Manager that keeps track of Threads	20
Game Engine (1)	• Rendering object in game world • Add Objects to the world • Input allows player to move around • Co-op with Thread-Pool Design • Vertical Sync	45
Timer	• Timer is Constant with Frame-Rate • Consistent Times for Function Run-Times • Get Time of when frame is done	35
Thread-Pool (2)	• Multiple Jobs on Threads List and Sort by highest priority or FIFO • Threads done when Time-Limit is up (Frame ends)	40

	• Remove and/or Insert Jobs that have a Higher Priority	
Game Engine (2)	• Add Physics to Engine • Co-op with Thread-Pool Design	50
Scheduler	• List for Threads will Account for Times of Functions • Organize Which Thread has Room for More Jobs Depending of Length of Time Left for Frame	75
Thread Visuals	• Get Information Based on What Threads are Currently Working on Quickly with Little Interruption • Read and Display Information in Graph Formation	35
Proof-Of-Concept	• Adding all objects in project at start up • Ensuring Everything Runs Smoothly	15
Final Touches	• Bug Fixing • Finishing Touches	20
Testing	• Debugging Threads • Hands-on Testing	50
Documentation	• Function Descriptions • Class Descriptions • Milestones	25
Total		415

14. Deliverables

Thread Pool Architecture

A fully functioning Job system that multiple threads can use to run functions.

Game Engine

Working game engine that renders, responds and plays.

Final Report

A final report on the engine and the development from the project.

15. Conclusion and Expertise Development

While most of the materials I have had some base experience working with, mainly the game engine itself, I will be learning a whole new way of making a game engine. This definitely has a lot to do with my current course that I am taking as I am in the Games Development Course. I will be learning a lot more about threading, timing in the computer and 3D game making. Moreover, I may need to adjust my scope as the project develops and I monitor/evaluate my progress. I do believe and look forward to the challenges this project will provide.

16. References

- Multithreading the Entire Destiny Engine. (2017, January 22). Retrieved from https://www.youtube.com/watch?v=v2Q_zHG3vqg

17. Change Log

1. October 2018 – I have updated section 3 (Page 3),
   - Reduced the scope of the project to just the game engine (Page 3, 7 & 8),
   - Separated the innovation into Game Engine + Innovation (Page 4 & 5),
   - Added more details about how I am going to build the project (Page 4 & 5),
   - Added more details to the Milestones (Page 13 & 14),
   - Clarified on some of the tasks in the scope and updated the software architecture diagram (Page 11),
   - Technologies Used Paragraph is clearer (Page 13),
   - Added Change log (Page 15).