Assignment P1 – Low Level

Formal assignment description for P1 - INFOMOV
Jacco Bikker, 2022

Universiteit Utrecht

Introduction

This document describes the requirements for the first assignment for the INFOMOV course. For this assignment, you will improve the performance of one part of a larger application, using low level optimizations.

Base Code

Please download the code for this assignment from Github: https://github.com/jbikker/tanks22. Note that the project is designed to run out-of-the-box on Windows machines using Visual Studio 2019 or 2022 (the free Community Edition should do just fine). If you use a different platform, you may need to convert some code to make it work on your machine.

The base code contains several potential targets for low-level optimization:

1. The sprite rendering code in sprite.cpp;
2. The map rendering code in map.cpp;
3. The grid functionality in grid.cpp;
4. The flag simulation and rendering code in flag.cpp.

The goal for this assignment is to chose one of these, and to optimize it using the 'Rules of Engagement' presented in the lecture on low-level optimization.

Game Architecture

The Tank game is a minimalistic real-time strategy game, which currently does not have any means to control the armies: the application serves as a 'tech demo' for a future, more complete product. The performance of the tech demo is however a bit disappointing, which is something we will resolve in several steps during the INFOMOV course.

The game intentionally makes very little use of the GPU. In the end something must be displayed of course, but until then, all rendering happens on the CPU, which plots pixels in a screen-sized buffer. This includes the map, which is drawn with a custom zoom and bilinear interpolation. Likewise, sprites are drawn with straight C++ code on the CPU; alpha blending and (again) bilinear interpolation is all handled on the CPU.

In terms of architecture, the game uses a simple list of *actors*, which all receive a 'Tick' per frame to advance their states. Actors include the tanks, their bullets, as well as some particle effects.

An important high-level optimization has already been implemented, in the form of a grid, which helps tanks find other tanks, to evade or to shoot at. You can find this functionality in grid.cpp. Whether this functionality is optimal on a low level is of course unknown.

Optimization

For this first assignment you are explicitly asked to optimize the code with **low-level optimizations**. That means:

- Refrain from using the GPU – optimize CPU code instead.
- Do not replace *algorithms*; instead, optimize the existing algorithms, even if you feel smarter algorithms would improve matters. In other words: **no high level optimizations**.
- Do not use multithreading (for now).

Additionally, you are asked not to change any interfaces. If you do feel you have to make invasive modifications, make sure this affects the component at hand only:

The plan is to make all (or most) submitted optimizations publicly available, so that we can do later assignments with a 'Frankenstein' application. The only way to make this work is by not changing any interfaces at this stage. Later in the block this restriction will be lifted.

Subtasks

Your grade for this assignment will be based on three aspects of your work:

1. The quality of the applied optimization process (30%).
2. The final performance of the component you chose to optimize (40%).
3. Your documentation (30%).

We will describe each of these in more detail below.

1. Process

In the lectures we explained how the optimization process can be a deliberate sequence of steps, where profiling is pivotal. Since you will be working on one part of a larger code base, you need to find a way to profile just the snippet you will be working on. Once you are able to accurately profile this snippet, apply low-level optimizations, e.g. the ones from the 'Rules of Engagement', until you feel there is nothing to be gained, or until time runs out. Make notes of each iteration.

2. Performance

Since the code is new for this academic year, it is hard to predict exactly how fast it can be made, and which optimizations can be reasonably expected in the first two weeks of the course. As a rule of thumb, you lose points when you miss obvious opportunities for optimization.

Your score for performance also depends on the snippet that you selected. We estimate that the four snippets have been sorted by difficulty, i.e. the (rather compact) sprite rendering code is relatively easy to optimize, while the (more involved) flag simulation code is probably harder. This will reflect in your score. Note that you can work only on one snippet, so chose carefully.

3. Documentation

The final aspect that we assess is your report. Make sure that you document your process clearly and that you present a full overview of performance data, in an accurate and readable way.

Some Technical Details

The tank game is implemented on top of a template, which you can find (for your own use) on Github: https://github.com/jbikker/advgrtmpl8. The template provides some limited but useful functionality:

- Opening a window and providing a linear framebuffer for plotting pixels: for this, GLFW is used which should make the code mostly platform-independent.
- Image loading, storage and editing using a Surface class.
- Vector, matrix and quaternion classes for common float and integer vectors.
- OpenCL functionality to easily integrate GPGPU code in your project (not for this assignment!).
- Highly optimized project files, which compile in a flash.

You can find the core template code in template.cpp. Feel free to modify this at will; template code is a valid optimization target!

The template lets you specify a screen resolution in template/common.h. For the tank game, this defaults to a full-screen 1920x1080. Hint: windowed rendering does complicate debugging.

The colors used by the template are 32-bit RGB. This is used everywhere in the template, so it is useful to understand how this works:

A single 32-bit value consists of 4 bytes. Three of these store red, green and blue; a fourth byte can be used for 'alpha' (i.e. transparency). Since each color component has one byte available, the range for each component is 0..255.

Combining the bytes in a 32-bit value is done using 'bitshifting'. E.g., green is in the second byte, so its value is shifted 8 bits to the left, which is the same as multiplying by 256.

We will talk a bit more about the technical details of the game in the practical sessions.

Team

You may work on this assignment alone, or with one partner. You may team with one partner for all assignments, but it is also allowed to change teams per assignment. You cannot change your team halfway an assignment; if for whatever reason you don't want to finish the project with your partner, both of you will work alone. Both team members may continue working with the code that was produced up till the split.

You may exchange information about the project with other students, online or in real life. Do not share code snippets; limit the exchange to ideas, hints, and concepts. This is in fact encouraged.

Deliverables

Your submission will consist of a **report** plus **project files**. Make sure the code compiles out-of-the-box in VS2019 and/or VS2022. If any other tools are required to produce the intended executable, please add a `readme.txt` that contains build instructions.

Deadline

The deadline for this assignment is **Tuesday May 10, 17:00**. If you fail to meet this deadline, you may submit one day later. One point will be subtracted from your grade in this case. Please submit your work using Teams.

Academic Conduct

The work you hand in must be your own original work, or properly referenced. If you used materials from other sources, please specify this clearly in your report.

Do not store your work in a publicly accessible location (this includes github!). If other students use your work (now or in the future), you may be reported along with the perpetrators.

Purpose

The purpose of this assignment is to practise low-level optimization and the related structured process. A faster application is a by-product of this. In subsequent iterations we will approach this (or another application) in a more holistic way.

The End

Questions and comments:

bikker.j@gmail.com

INFOMOV 2022