COM00003H

MSc/BEng/MEng Examinations, 2017–18

DEPARTMENT OF COMPUTER SCIENCE

Embedded Systems Design and Implementation (EMBS)

Open Examination

Issued at:
April 16th 2018

Submission due:
12:00 Midday, May 9th 2018

Your attention is drawn to the Guidelines on Mutual Assistance and Collaboration in the Student's Handbook and the Department's 'Guide to Assessment Policies and Procedures' (http://www.cs.york.ac.uk/exams/statementonassessment/).

All queries on this assessment should be addressed to **Ian Gray**.

Your examination number must be written on the front of your submission, and included with your submitted source code. You must not identify yourself in any other way.

See https://wiki.york.ac.uk/display/RTS/2018+Assessment+Information for all course information, current notices and any further documentation referred to within this paper.

Electronic submission of your report, developed code, project files, and FPGA bitfiles should be submitted via http://www.cs.york.ac.uk/student/assessment/submit/
Demonstration of your solution(s) will occur after hand-in at a specific time to be posted on the module web site.

Feedback and marks due: Wednesday 6th June 2018.

Should you wish to request an extension see the appropriate section of the Department's 'Guide to Assessment Policies and Procedures'.

Overview

In this assessment you will implement and accelerate a particle simulation engine. Your system will communicate with both the user and a network server to receive scenes to simulate. You will visualise the scene using VGA graphics and will attempt to use the parallelism inherent in the FPGA to accelerate your design. You will describe your approach and design in a report.

For marking, the assessment is split into:

(a) [35 marks] Written report describing your key design, implementation, evaluation and testing activities.

(b) [65 marks] Demonstration and evaluation of your developed system against test scenarios.

Further details of the report and evaluation are given later in this exam paper.

Particles

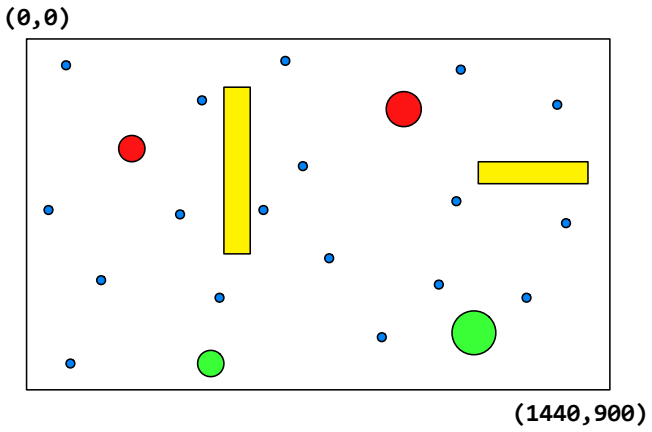

Figure 1: Example Particle Scene

Your system will simulate a 2D particle system composed of particles and attractors. An extension will also implement walls which block the movement of particles. Attractors and walls are stationary, particles will move over time according to how they are attracted to the attractors.

The purpose of the assessment is to accelerate the particle system such that your implementation can achieve a high frame rate with a large number of

simulation objects. You must make use of hardware resources on the FPGA by porting elements of the engine to Vivado HLS. It is up to you how to best split this problem between hardware and software.

The scene is 1440 units wide by 900 units high, with the origin coordinate $(0, 0)$ in the top left corner.

A particle p is defined by the following:

- An X position: $p.x$
- A Y position: $p.y$
- An X velocity: $p.vx$
- A Y velocity: $p.vy$

An attractor a is defined by the following:

- An X position: $a.x$
- A Y position: $a.y$
- A gravitational force: $a.g$. Positive values of g draw particles towards, whilst negative values repel particles.

All particles start with 0 velocity. Assume standard rules of geometry.

At each frame of the simulation, you must calculate the following:

foreach *particle* p **do**
    /* Update the particle's position according to its velocity */
    $p.x := p.x + p.vx$;
    $p.y := p.y + p.vy$;
    /* Update the particle's velocity according to the attractors */
    foreach *attractor* a **do**
        /* Calculate the distance (d) from p to a */
        $d := \sqrt{(a.x - p.x)^2 + (a.y - p.y)^2}$;
        /* Ignore attractors too far away */
        if $d < 500$ **then**
            /* Normalise the attraction vector */
            $x_{norm} := \frac{a.x - p.x}{d}$;
            $y_{norm} := \frac{a.y - p.y}{d}$;
            /* Apply attraction */
            if $d < 1.0$ **then**
                $x_{norm} := x_{norm} \times a.g$;
                $y_{norm} := y_{norm} \times a.g$;
            else
                $x_{norm} := x_{norm} \times \frac{1}{d} \times a.g$;
                $y_{norm} := y_{norm} \times \frac{1}{d} \times a.g$;
            end
            /* Update velocity */
            $p.vx := p.vx + x_{norm}$;
            $p.vy := p.vy + y_{norm}$;
        end
    end
end

Your system does not have to properly account for time passing. Assume that each frame is a consistent "tick". This means that at higher framerates the simulation will move faster.

Collisions

Particles must not be able to leave the confines of the scene. You can implement collision with the edges of the scene any way you wish, but rather than 'bouncing' the particle, zero the particle's Y velocity when it hits the top or bottom of the screen, and its X velocity when it hits the left or right sides. This removes energy from the system and keeps it stable. Particles should not get 'stuck' on the sides after a collision.

Particles may pass though each other and do not have to collide with other particles.

If you have implemented walls, adopt a similar scheme for when a particle hits a wall. In rare cases it might be possible for a very fast moving particle to be going so fast that it can move straight through a thin wall. This situation can be ignored.

For all collisions, your goal is only to create a simulation which looks realistic in most cases. It does not have to be perfectly accurate.

Interface

The user must be able to instruct your system via the UART to randomise the scene. When randomising, the system should request how many particles, attractors, and walls (if implemented) to create. Then use the following ranges:

- Particle positions - anywhere in the scene
- Particle velocities - zero
- Attractor positions - anywhere in the scene
- Attractor values - a floating point value. Approximately half of the attractors between 1.0 – 2.0, the other half between -0.5 – -1.0
- Wall positions - anywhere in the scene
- Walls have a thickness of between 20 – 70 units, and a length between 80 – 300 units. They may be oriented either horizontally or vertically.

Scenario Server

The user must be able to instruct your system via the UART to fetch a specific scenario from a server located on the Ethernet network. The scenarios are

numbered and listed on the assessment information page of the wiki:
https://wiki.york.ac.uk/display/RTS/2018+Assessment+Information

Your system will send a message to the server requesting a specific scenario, and the server will respond with a set of Ethernet packets that contain the particles, attractors, and walls, that describe that scenario. Your system should replace any current simulation with these objects, and begin simulating the new scenario.

Details of the UDP packet formats, the protocol, and the IP address and port of the server, are on the assessment information page.

Frames Per Second display

You are aiming to achieve as high a framerate as possible. Your solution must be able to output the current frame rate on the UART (or VGA), expressed as frames per second (FPS). The assessment information page has example code which you can use to calculate framerate.

VGA

You should output the rendered scene on the VGA, at the default 1440×900 resolution. Information and example code for this can be found at the link:
https://wiki.york.ac.uk/display/RTS/Zybo+VGA+Output

Feel free to experiment with other resolutions during testing, but use 1440×900 when demonstrating your solution.

You can render the scene any way you wish. However it must be possible to clearly differentiate between particles, walls, attractors with a positive g value, and attractors with a negative g value.

You will get more marks if your scene is rendered with circles for particles and attractors (see the mark scheme later).

Tips

The most accurate way to simulate the system is to use floating point numbers for particle position and velocity, and to round these to integer pixel coordinates before rendering to the VGA. However, perfect accuracy is not a requirement of the system. The aim is to create a simulation that still looks like it is behaving correctly to the observer. You will not be penalised for small numerical inaccuracies if they can be justified in terms of speed or efficiency. If in doubt, speak to the lecturer.

HLS can create hardware for C floats, but fixed point and integer types can sometimes be much more efficient. There may be data type simplifications and algorithm approximations that you can make that will use less hardware and run faster.

Vivado HLS Fixed point types are described on the wiki:
https://wiki.york.ac.uk/display/RTS/Vivado+HLS+Knowledge+Base

Constraints

Your solution:

- must use the provided EMBS FPGA hardware platform.
- must be created using Xilinx tools with custom hardware peripherals built using Vivado HLS.
- should **NOT** use other IP cores or designs available from the web or elsewhere.

Marks, Notes and Guidance

Written Report [35 marks]:

- Maximum of 1000 words - not including figures, tables, code fragments etc.
- Your report should assume that the reader is knowledgeable about embedded systems, the development toolkits (e.g. Vivado HLS), the course, examination FPGA toolkits, and FPGAs themselves.
- Your report should discuss (all word counts exclude numbers, figures and diagrams):
  (a) [10 marks] Design of system, including justification of the split of functionality between CPU and accelerator (max. 300 words)
  Criteria: clarity and completeness of discussion; quantitative evidence when justifying split of functionality between CPU and accelerator
  (b) [10 marks] Parallelism used within the system (max. 300 words)
  Criteria: clarity and completeness of discussion
  (c) [5 marks] Testing approaches used (max. 200 words)
  Criteria: clarity and completeness of discussion
  (d) [10 marks] Evaluation (max. 200 words)
  Criteria: completeness of evaluation; discussion and justification of hardware resources utilised; discussion of timing performance of the system

Evaluation by Demonstration [65 marks]:
During demonstration the following will be tested (partial marks will be awarded where appropriate). In all cases when a specific number of particles, attractors, or walls is specified, they are to be interpreted as a minimum value. It is acceptable to simulate more.

When reading from the UART how many particles, attractors, and walls to create, the user should be able to type any number from 1 to the maximum supported. You can add 'shortcuts' for common configurations to aid testing if you wish.

(a) [5 marks] Draw a scene with 200 particles and 10 attractors (no particle movement required).

(b) [5 marks] Randomise and redraw the current scene when instructed over the serial. Read from the serial the number of particles and attractors to draw. Support 1000 particles and 50 attractors (no particle movement required).

(c) [8 marks] Simulate a random scene with 200 particles and 10 attractors.

(d) [8 marks] Request and simulate the full set of scenes without walls from the scenario server.

(e) [5 marks] Simulate a random scene with 200 particles, 10 attractors, and 10 walls.

(f) [4 marks] Request and simulate the full set of scenes with walls from the scenario server.

(g) [3 marks] Draw particles and attractors using circles.

(h) [2 marks] Simulate a random scene with 1000 particles and 50 attractors at 30FPS.

(i) [10 marks] Simulate a random scene with 6000 particles and 50 attractors at 30FPS.

(j) [10 marks] Simulate a random scene with 12000 particles and 50 attractors at 30FPS.

(k) [5 marks] Simulate a random scene with 12000 particles, 50 attractors, and 10 walls at 30FPS.

Remember that the use of hardware acceleration and offloading computation to the FPGA logic will be the main subject you will discuss in your report. You should carefully consider what functionality to offload to the FPGA, and which to leave in the ARM core.

Source Code
All hardware designs, Vivado HLS, and C program code should be submitted electronically, together with a bitfile. You should include the following:

- A Zip of your Vivado project and SDK source code.
  - To do this, in Vivado, click "File | Archive Project". Select "Include Configuration Settings" and deselect "Include Run Results". This will produce a zip file which contains your entire Vivado project (including your SDK software). It will not include your HLS project.
  - **Important Note:** If you are low on disk space then you might find that the created Zip file is not complete. Please check it before submission.
- A Zip of your Vivado HLS project.
  - In HLS, click "File | Archive Project". Select "Include Run Results" (they are much smaller than in Vivado so include them).
- Your final bitfile.

If there is anything non-standard about your design build include instructions within a "README" file. Please note that all submitted code should be readable. This means that tabulation should be used to aid readability, and the code should be well commented.