Lab Book

Daniel Bates

Contents

[Rust Labs 3](#_Toc104404691)

[Week 1 – Lab A 3](#_Toc104404692)

[Q1. Installs 3](#_Toc104404693)

[Q2. Hello World 4](#_Toc104404694)

[Week 2 – Lab B 5](#_Toc104404695)

[Q1. First threads 5](#_Toc104404696)

[Q2. Joining threads 6](#_Toc104404697)

[Q3. Experimentation 7](#_Toc104404698)

[Week 3 – Lab C 13](#_Toc104404699)

[Q1. Multiple Rust Files 13](#_Toc104404700)

[Q2. Ownership 14](#_Toc104404701)

[Q3. Classes 18](#_Toc104404702)

[Week 4 – Lab D 20](#_Toc104404703)

[Q1. Ownership Limitations 20](#_Toc104404704)

[Q2. Reference Counters 21](#_Toc104404705)

[Week 5 – Lab E 24](#_Toc104404706)

[Q1. Thread safe printing 24](#_Toc104404707)

[Q2. Triangles and OpenGL 26](#_Toc104404708)

[Week 6 – Lab F 27](#_Toc104404709)

[Q1. Particles 27](#_Toc104404710)

[Q2. Threaded Particles 31](#_Toc104404711)

[Week 7 – Lab G 33](#_Toc104404712)

[Q1. Colliding particles 33](#_Toc104404713)

[Q2. Recording collisions using an Atomic 35](#_Toc104404714)

[Week 8 – Lab H 36](#_Toc104404715)

[Q1. Condition variables 36](#_Toc104404716)

[Q2. Striped arrays, sequential access 38](#_Toc104404717)

[Q3. Striped arrays, random access 39](#_Toc104404718)

[CUDA Labs 40](#_Toc104404719)

[Week 1 – Lab 1 40](#_Toc104404720)

[Q1. Setting up CUDA 40](#_Toc104404721)

[Q2. CUDA error checking 41](#_Toc104404722)

[Q3. Check time range 42](#_Toc104404723)

[Week 2 – Lab 2 43](#_Toc104404724)

[Q1. Understand the block and thread indices 43](#_Toc104404725)

[Q2. Find vector addition using multiple 1D thread blocks 43](#_Toc104404726)

[Q3. Understand the thread indices for 2D blocks 44](#_Toc104404727)

[Q4. Find vector addition using one 2D thread block 45](#_Toc104404728)

[Q5. Find vector addition using multiple 2D thread blocks 45](#_Toc104404729)

[Q6. Matrix addition 46](#_Toc104404730)

[Week 3 – Lab 3 48](#_Toc104404731)

[Q1. Vector dot-product 48](#_Toc104404732)

[Q2. Vector dot-product using unified memory 49](#_Toc104404733)

[Q3. Vector dot-product using shared Memory 51](#_Toc104404734)

[Week 4 – Lab 4 53](#_Toc104404735)

[Q1. Create an OpenGL-CUDA program based on a CUDA SDK sample 53](#_Toc104404736)

[Q2. Understand pixel colour 54](#_Toc104404737)

[Q3. Image transformation 55](#_Toc104404738)

[Q4. Image smoothing 58](#_Toc104404739)

[Week 5 – Lab 5 60](#_Toc104404740)

[Q1. A simple matrix multiplication program in CUDA using one thread block 60](#_Toc104404741)

[Q2. Compare the performance of the CUDA solution against the CPU solution 61](#_Toc104404742)

[Week 6 – Lab 6 63](#_Toc104404743)

[Q1. Set up a virtual canvas and draw on it an image in CUDA 63](#_Toc104404744)

[Q2. Drawing a checkboard in CUDA 64](#_Toc104404745)

[Q3. Drawing the Mandelbrot and Julia Sets 66](#_Toc104404746)

[Week 7 – Lab 7 68](#_Toc104404747)

[Q1. Drawing based on a canvas of size [-1, 1]x[-1, 1] 68](#_Toc104404748)

[Q2. Write a simple ray caster 69](#_Toc104404749)

[Week 8 – Lab 8 70](#_Toc104404750)

[Q1. Draw a box without front wall 70](#_Toc104404751)

[Q2. Free motion animation 70](#_Toc104404752)

[Q3. Ball-box walls collision animation 72](#_Toc104404753)

[Final Lab 73](#_Toc104404754)

[GPU Design 73](#_Toc104404755)

[CPU Design 73](#_Toc104404756)

[Performance comparison 73](#_Toc104404757)

[Reflection 73](#_Toc104404758)

# Rust Labs

## Week 1 – Lab A

Date: 2nd Feb 2022

### Q1. Installs

#### Question:

Install Rust and the stated extensions for visual studio code

#### Solution:

N/A

#### Test Data:

N/A

#### Sample Output:

N/A

#### Reflection:

Just getting things ready for future labs

#### Metadata:

Set up

#### Further Information:

N/A

### Q2. Hello World

#### Question:

Create a new project folder. Use the visual studio code terminal to first change the location to the new folder. Use “cargo init” to setup a new project then use “cargo run” to run the project.

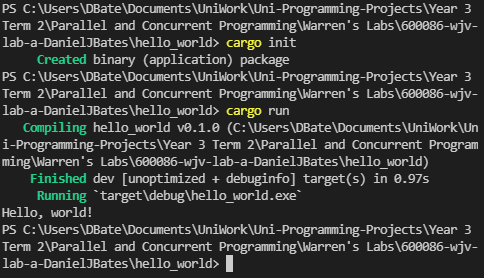
#### Solution:



#### Test Data:

N/A

#### Sample Output:



### 

#### Reflection:

Just the basic stuff

#### *Metadata*:

Hello World

#### Further Information:

Useful resources for learning Rust:

<https://www.youtube.com/watch?v=zF34dRivLOw>

<https://doc.rust-lang.org/stable/book/>

## Week 2 – Lab B

Date: 9th Feb 2022

### Q1. First threads

#### Question:

Replace the synchronous call to your function with an asynchronous call.

#### Solution:

#### 

#### Test data:

n/a

#### Sample output:

#### Reflection:

This is threading 101

#### Metadata:

Threads

#### Further information:

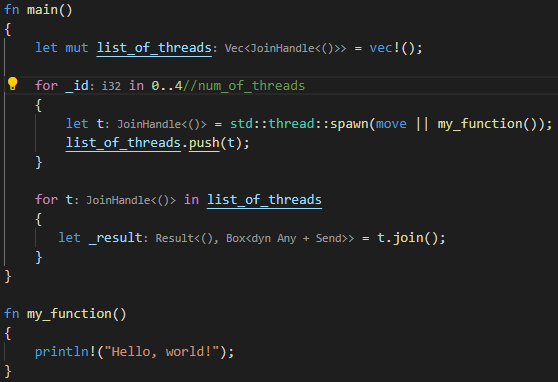
N/A

### Q2. Joining threads

#### Question:

Add code to create an arbitrary number of threads and then join them.

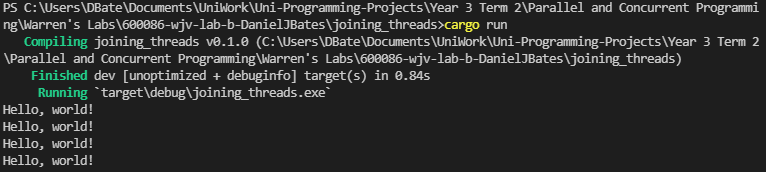
#### *Solution*:



#### *Test Data*:

N/A

#### Sample Output:



#### Reflection:

Rust is a bit complicated to wrap my head around. Need to do some reading on the language a bit for next lab

#### Metadata:

Joining threads

#### Further Information:

N/A

### Q3. Experimentation

#### Question:

experiment with giving the threads items of work, as well as altering the number of threads used

#### Solution:

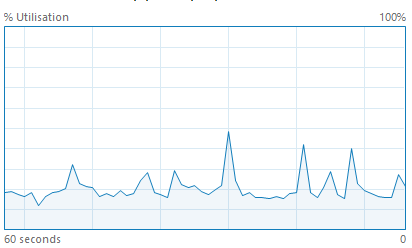
N/A

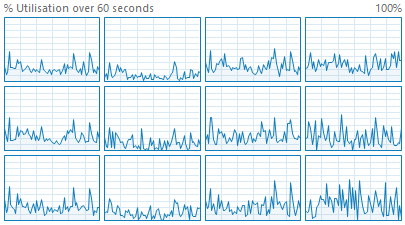
#### Test data:

N/A

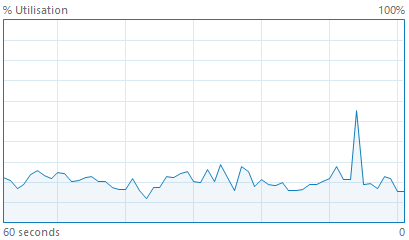
#### Sample output:

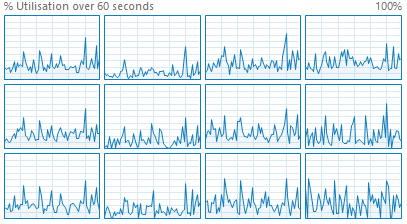
**512 threads (Hello World) –**



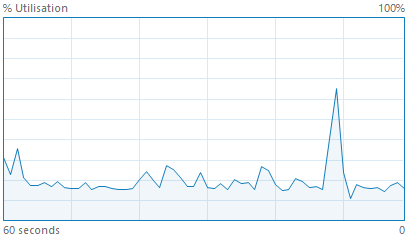


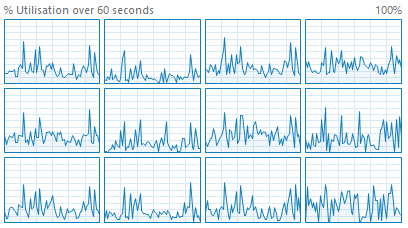
**1024 threads (Hello World) –**



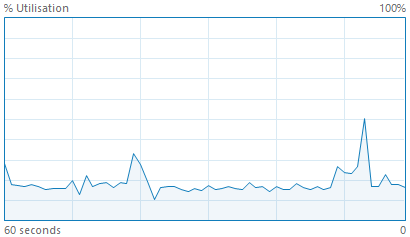


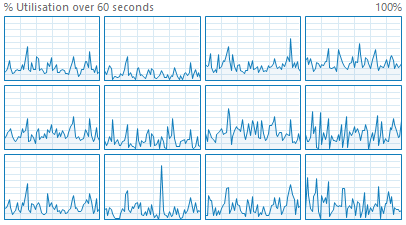
**2048 threads (Hello World) –**

****

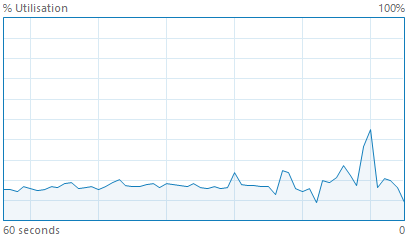
****

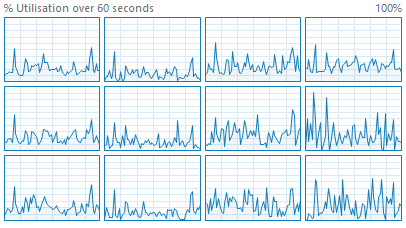
**512 threads (1 + 2 + 5 \* 25) –**



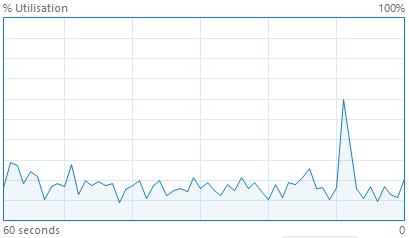


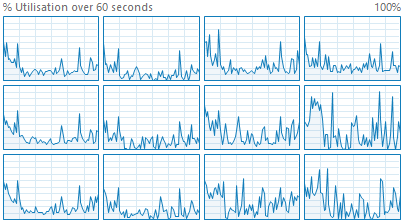
**1024 threads (1 + 2 + 5 \* 25) –**

****

****

**2048 threads (1 + 2 + 5 \* 25) –**

****

****

#### Reflection:

Graphs are a bit messy need redoing.

There is a spike whenever the application is run

#### Metadata:

N/A

#### Further information:

REDO Graphs

## Week 3 – Lab C

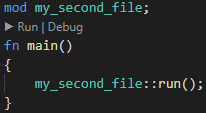
Date: 16/02/2022

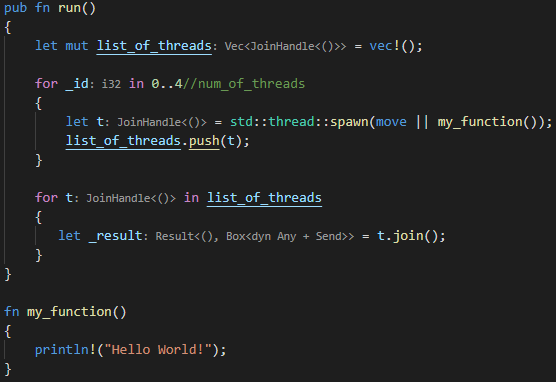
### Q1. Multiple Rust Files

#### Question:

Move your thread main function, from the previous lab to its own Rust file

#### Solution:





#### Test data:

N/A

#### Sample output:

Hello World! X4

#### Reflection:

Functions need to be public to be used by a file outside of the original file

#### Metadata:

N/A

#### Further information:

N/A

### Q2. Ownership

#### Question:

**Part 1 –** Try to print p1 twice

**Part 2 –** Alter the code so that the print\_person returns the Person object back to the main

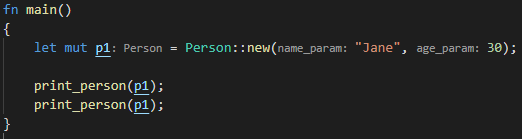
**Part 3 –** Modify print\_person to use a reference

**Part 4 –** Add the function increment\_age which takes a mutable reference as a parameter

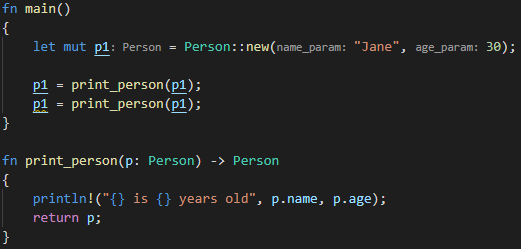
**Part 5 –** Cause borrowing to fail

#### Solution:

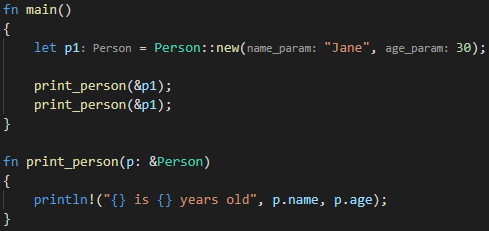
**Part 1 –**



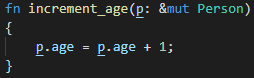
**Part 2 –**

****

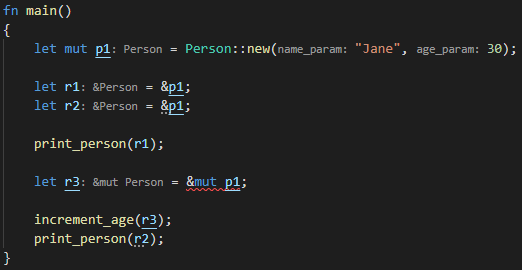
**Part 3 –**

****

**Part 4 –**

****

**Part 5 –**

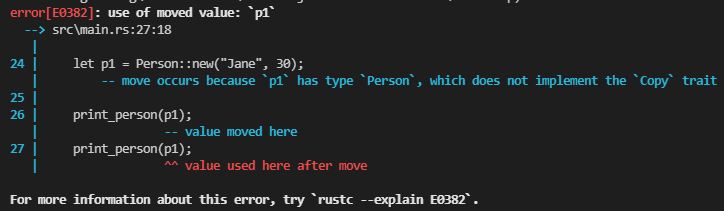
****

#### Test data:

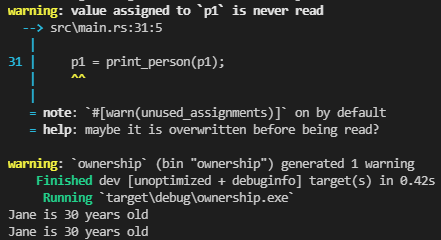
**All parts are N/A**

#### Sample output:

**Part 1 –**

****

**Part 2 –**

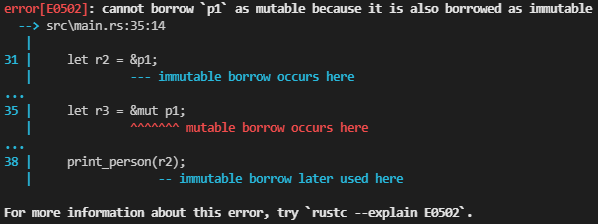


**Part 3 –**



**Part 4 –** N/A

**Part 5 –**

****

#### Reflection:

Not sure if you would ever use explicit mutable and immutable references

#### Metadata:

Ownership

#### Further information:

Rust does not allow overriding of function names

<https://doc.rust-lang.org/book/ch04-02-references-and-borrowing.html>

### Q3. Classes

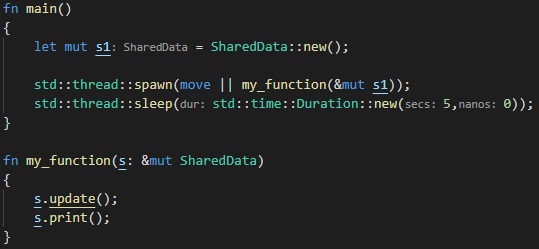
#### Question:

**Part 1 –** Create a new thread function which takes SharedData as a parameter and then calls the update and print functions

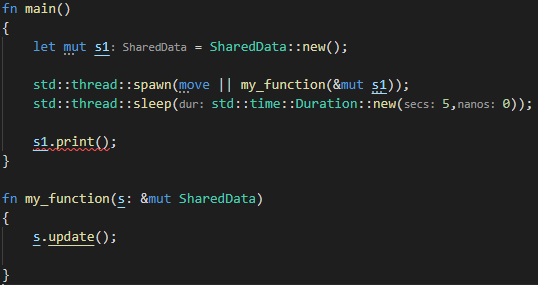
**Part 2 –** move the print function from your thread function to the main program

#### Solution:

**Part 1 –**

****

**Part 2 –**

****

#### Test data:

**Part 1 –** N/A

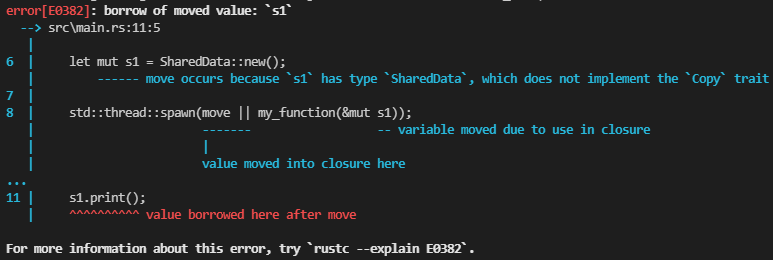
**Part 2 –** N/A

#### Sample output:

**Part 1 –**



**Part 2 –**

****

#### Reflection:

Couldn’t get part 1 to work with the template I used from the joining threads lab

#### Metadata:

Classes

#### Further information:

The fix to the problem addressed in part 2 will be covered later

## Week 4 – Lab D

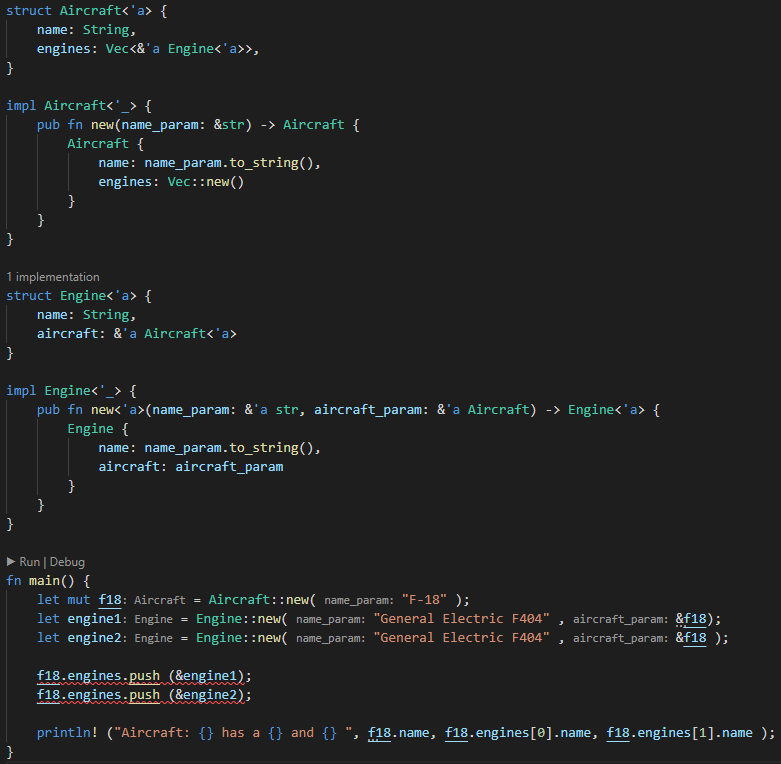
Date: 23/02/2022

### Q1. Ownership Limitations

#### Question:

Read and understand the syntax of the provided code. Try to expand the code to include a data member in Engine that links to the Aircraft

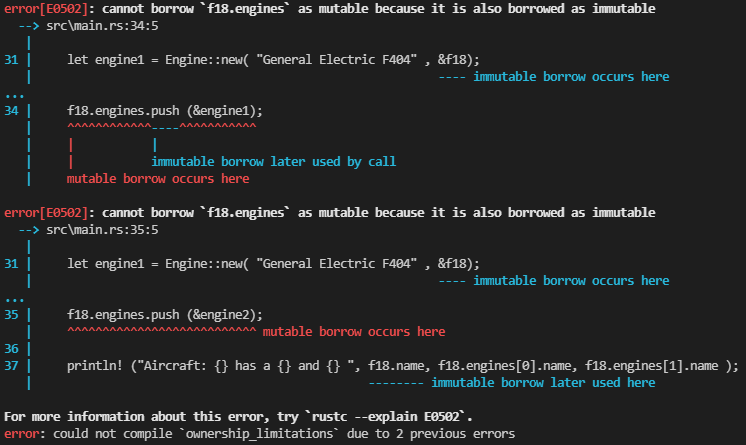
#### Solution:



#### Test data:

N/A

#### Sample output:



#### Reflection:

The limitation with the current code is that due to ownership restrictions it is not possible to link the Aircraft to an Engine

#### Metadata:

Limitations

#### Further information:

'a notation attached to the reference is called a lifetime parameter. It allows the compiler to determine whether all references are going to stay "alive" at least as long as the "parent"

### Q2. Reference Counters

#### Question:

**Part 1 –** Examine the provided code. Explain what is happening with the reference counters and why we do not need to pass them as references

**Part 2 –** Remove the clone() method. Explain why this program now fails to build

**Part 3 –** Add a new boolean data member requires\_service to Engine. Add a new method service(&mut self) to Engine. This method will just set the requires\_service data member to false. Try to call the service method on engine1. Explain why the error is occurring

#### Solution:

**Part 1 –**

****

A vector of reference counters for Engine objects is created



For each engine a new reference counter is made for a new engine object



Each engine is cloned creating another pointer to the same allocation associated with reference counter. I think we don’t need to pass as a reference here because a new reference is being created by clone and the reference counter for that object then increases the count

**Part 2 –** The program does not build now because the value for engine1 gets moved when it gets put into the list of engines for the aircraft. So, when the program tries to print that engines name later it can’t because the engine object has been borrowed and not returned

**Part 3 –** I think the error is occurring because the references created by the reference counter are immutable

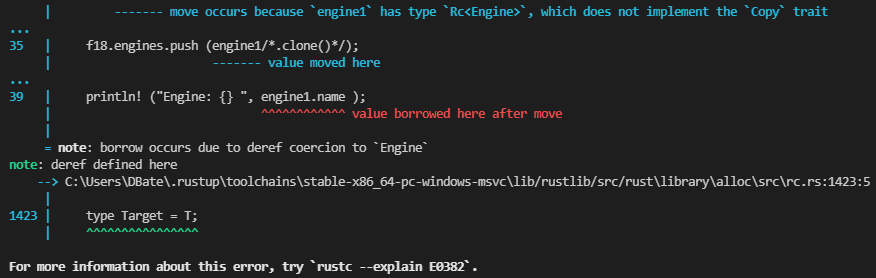
#### Test data:

**Part 1 –** N/A

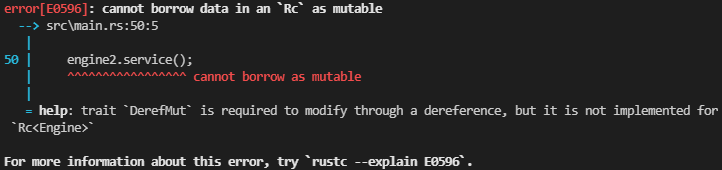
#### Sample output:

**Part 1 –** N/A

**Part 2 –**



**Part 3 –**



#### Reflection:

**Part 1 –** I think I understand how RC works but not well enough to properly explain it

**Part 2 –** This part was just the same as the ownership issues previously covered

**Part 3 –** The solution to this will be covered in a later lab

#### Metadata:

Understanding RC a bit more

#### Further information:

Ask about my library enallage to see if I’m understanding it correctly

## Week 5 – Lab E

Date: 02/02/2022

### Q1. Thread safe printing

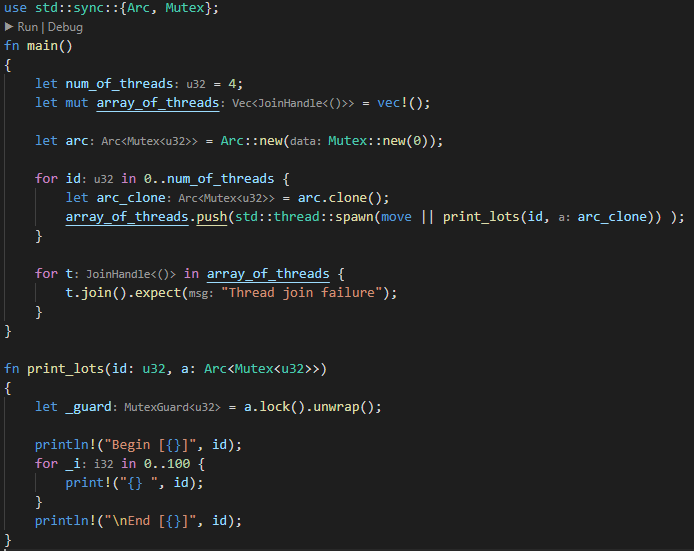
#### Question:

**Part 1 –** Implement the thread safe printing

**Part 2 – Q1.**What happens to your code if you fail to release the mutex? **Q2.**Are you able to verify this in your code? **Q3.**What happens if you raise an exception within the critical section? **Q4.**Extend your code to verify your answer

#### Solution:

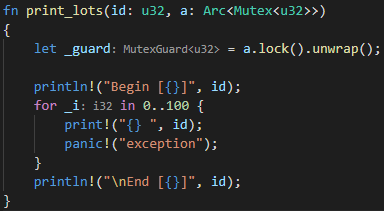
**Part 1 –**



**Part 4 –**

**Q2.**

**Q4.**

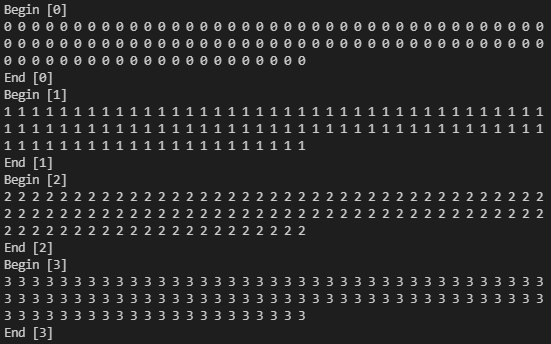
****

#### Test data:

**All parts –** N/A

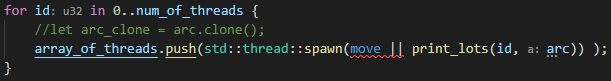
#### Sample output:

**Part 1 –**

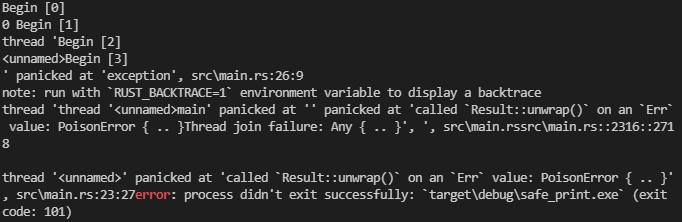


**Part 2 –**

**Q2.**

****

**Q4.**



#### Reflection:

**Part 1 –** I’m not really sure what an ARC does differently to an RC

**Part 2 –**

I wasn’t 100% sure what question 1 meant but I’ve answer what I think it means

**Q1.** If the mutex isn’t released then in the first iteration of the thread loop the mutex is moved so in all iterations after the mutex and be used. In short it creates an ownership error.

After Tuesday live lecture I understand what is meant. The program reaches a deadlock

**Q3.** When I raised an exception in the critical section I get a “PoisonError”

#### Metadata:

Errors

#### Further information:

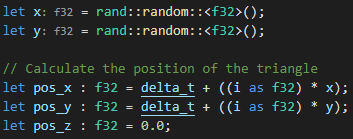
N/A

### Q2. Triangles and OpenGL

#### Question:

Update the code to make the triangles move more chaotically

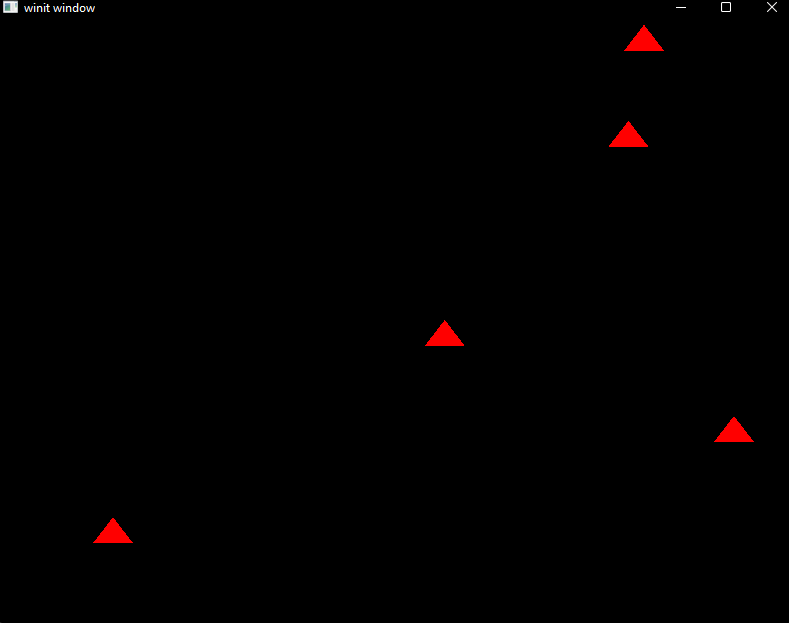
#### Solution:



#### Test data:

N/A

#### Sample output:



#### Reflection:

Rnad was acting weird to import the crate

#### Metadata:

Chaos

#### Further information:

N/A

## Week 6 – Lab F

Date: 09/03/2022

### Q1. Particles

#### Question:

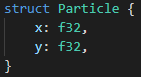
**Part 1 –** Implement structs described in the lab sheet

**Part 2 –** Implement functions for these structs as described in the lab sheet

**Part 3 –** Test release mode and add the macro described in the lab sheet

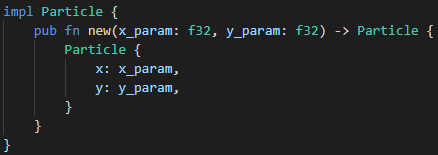
#### Solution:

**Part 1 –**





**Part 2 –**

**Text

Description automatically generated**

**Part 3 –**

****

****

****

#### Test data:

**Part 1 –** N/A

**Part 2 –**

**Text

Description automatically generated**

**Part 3 –** N/A

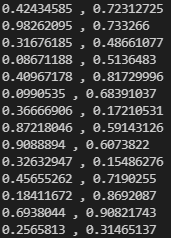
#### Sample output:

**Part 1 –** N/A

**Part 2 –**

**Text

Description automatically generatedETC.**

**ETC.**

**Part 3 –** N/A

#### Reflection:

The timer took me a bit to figure out. I was using a timer crate at first, but I think it only works with a threaded solution, so I just went with using delta time. Not sure what #[derive(Debug, Copy, Clone)] does

#### Metadata:

Particles

#### Further information:

N/A

### Q2. Threaded Particles

#### Question:

Implement a threaded version of the pervious question using chunks and scoped thread pools

#### Solution:

Text

Description automatically generated

Text

Description automatically generated

#### Test data:

N/A

#### Sample output:

Text

Description automatically generatedA picture containing text

Description automatically generatedText

Description automatically generated with medium confidenceA picture containing text, keyboard

Description automatically generated

#### Reflection:

This was a bit confusing at first although I had similar code the first time I did it. I redid the code after I understood it more thanks to the Tuesday lecture

#### Metadata:

Chunks

#### Further information:

I changed the randomness of the movement for more predictable outcomes for testing

## Week 7 – Lab G

Date: 16/03/2022

### Q1. Colliding particles

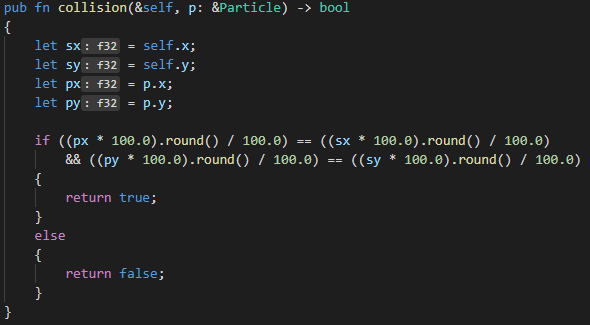
#### Question:

**Part 1 –** Create a new function that checks if a particle collides with or is very close to (within 2 d.p.) another particle

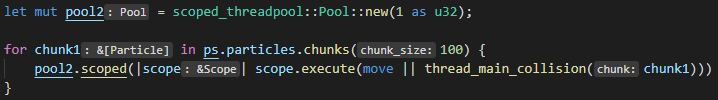
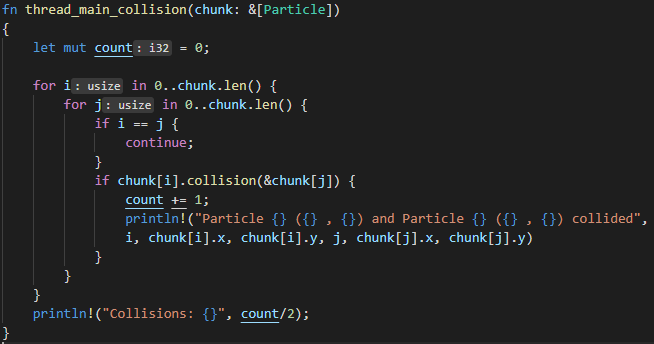
**Part 2 –** Add a counter to count the number of collisions

#### Solution:

**Part 1 –**



**Part 2 –**

****

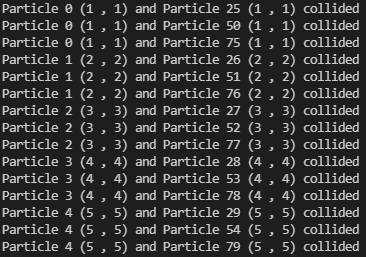
#### Test data:

**All Parts –** The same tester code was used as last lab

#### Sample output:

**Part 1 –** N/A

**Part 2 –**

**ETC.**



#### Reflection:

**Q1.** Is locking required in your solution to prevent race conditions?

**A1.** I didn’t use any locking in my solution

**Q2.** Are there any other race conditions that can occur in your code?

**A2.** I don’t think there are because I used 2 different thread pools. If I had just used 1 I think I would have and then I would also need to use locking

**Q3.** Are there any optimisations you can make to your code?

**A3.** I could have the collisions stored in a list so the program is counting the same collision twice

#### Metadata:

Collisions

#### Further information:

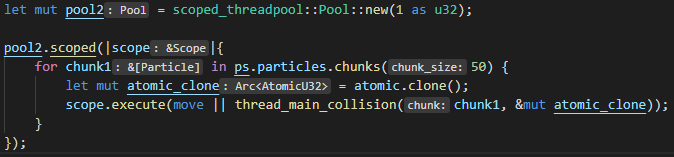
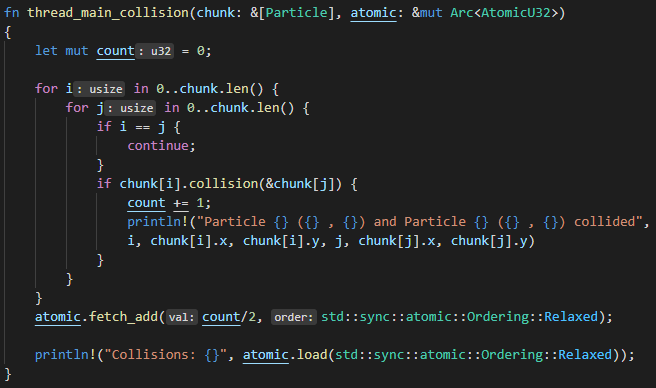
I’m not sure how using chunks for the collisions would work. How would you be able to check for collisions between chunks

### Q2. Recording collisions using an Atomic

#### Question:

Replace the local counter with an atomic counter to measure the number of collisions across all threads

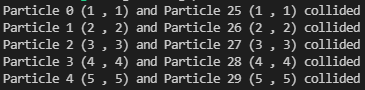
#### Solution:



#### Test data:

N/A

#### Sample output:

**ETC.**



#### Reflection:

I don’t understand the Ordering variable for atomic functions. And for some reason even though for collisions I only gave it 1 thread it executes twice and does the same thing again.

#### Metadata:

Atomics

#### Further information:

N/A

## Week 8 – Lab H

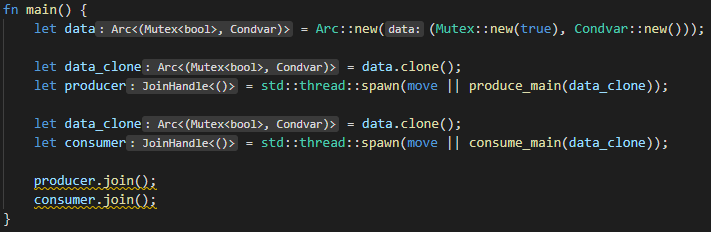
Date: 23/03/2022

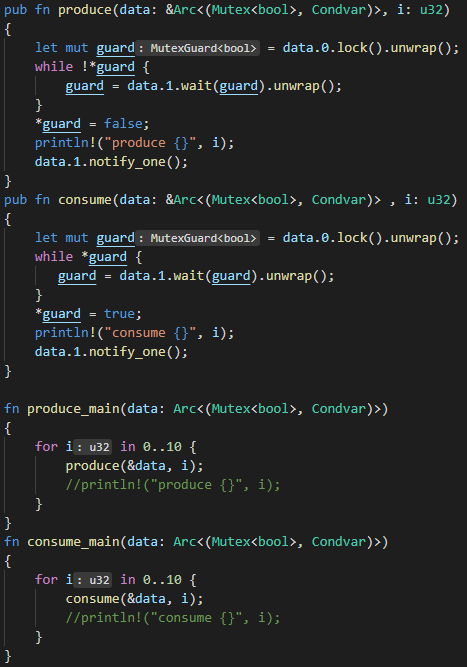
### Q1. Condition variables

#### Question:

Create a producer / consumer model using condition variables

#### Solution:

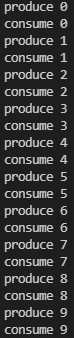




#### Test data:

N/A

#### Sample output:



#### Reflection:

I got myself very confused on this, with a lot of help from Warren I got the task done. But I think I defiantly need to practice this more to better understand it in practice

#### Metadata:

Condition Variables

#### Further information:

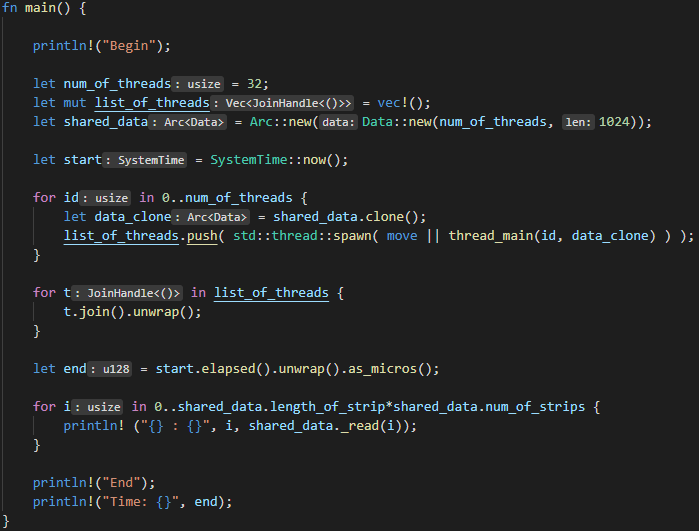
N/A

### Q2. Striped arrays, sequential access

#### Question:

Add timing code to the example to measure the duration of array access. Alter the program to use 2, 4, 8, 16, 32 threads

#### Solution:



#### Test data:

2, 4, 8, 16, 32 threads

#### Sample output:

|  |  |  |
| --- | --- | --- |
| **Threads** | **Strips** | **Time** |
| 2 | 16384 |  |
| 4 | 8192 |  |
| 8 | 4096 |  |
| 16 | 2048 |  |
| 32 | 1024 |  |

#### Reflection:

The timing seems to double each time

#### Metadata:

Timings

#### Further information:

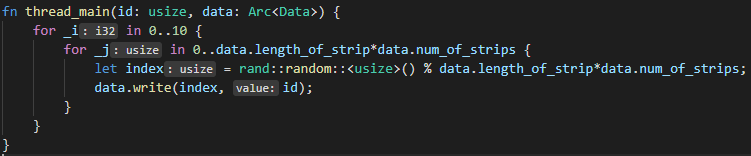
N/A

### Q3. Striped arrays, random access

#### Question:

Modify the code to implement random access

#### Solution:



#### Test data:

2, 4, 8, 16, 32 threads

#### Sample output:

|  |  |  |
| --- | --- | --- |
| **Threads** | **Strips** | **Time** |
| 2 | 16384 |  |
| 4 | 8192 |  |
| 8 | 4096 |  |
| 16 | 2048 |  |
| 32 | 1024 |  |

#### Reflection:

Random access seems to be significantly worse for scaling

#### Metadata:

Timings 2

#### Further information:

N/A

# CUDA Labs

## Week 1 – Lab 1

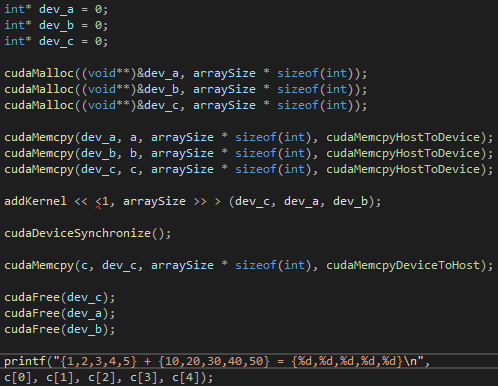
Date: 2nd Feb 2022

### Q1. Setting up CUDA

#### Question:

Set up the default CUDA program. Create a solution that adds 2 arrays together on the GPU

#### Solution:



#### Test data:

a = {1,2,3,4,5} b = {10,20,30,40,50}

#### Sample output:

{1,2,3,4,5} + {10,20,30,40,50} = {11,22,33,44,55}

#### Reflection:

The process can be broken down into 5 steps

1. Allocate memory
2. Copy data to GPU
3. Perform addition
4. Copy results to CPU
5. Release GPU buffers

#### Metadata:

GPU vector addition

#### Further information:

N/A

### Q2. CUDA error checking

#### Question:

Add CUDA error checking

#### Solution:



\*Snippet own code (the whole thing takes up too much space)

#### Test data:

N/A

#### Sample output:

N/A

#### Reflection:

Error checking seems really simple to implement this way but I think using exceptions would be a better way to do it

#### Metadata:

Errors

#### Further information:

N/A

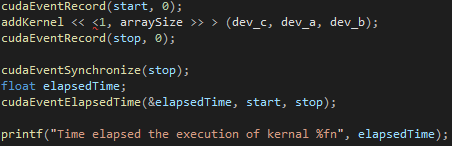
### Q3. Check time range

#### Question:

Create a timer for the kernel execution

#### Solution:





#### Test data:

N/A

#### Sample output:

N/A

#### Reflection:

Pretty simple to create a timer

#### Metadata:

Timer

#### Further information:

N/A

## Week 2 – Lab 2

Date: 9th Feb 2022

### Q1. Understand the block and thread indices

#### Question:

List the values for the built-in variables threadIdx.x and blockIdx.x corresponding to the given thread configurations used for executing the kernel addKernel( ) function on GPU

#### Solution:

N/A

#### Test data:



#### Sample output:



#### Reflection:

I think the first number in “addKernel << <1, 5 >> > (dev\_c, dev\_a, dev\_b);” represents the number of blocks and the second represents the size of the blocks. So, in this case there is 1 block that has 5 threads.

#### Metadata:

Understanding threads and blocks

#### Further information:

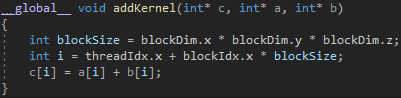
The lab sheet is a bit confusing to read to me, wasn’t really sure what was being asked of me

### Q2. Find vector addition using multiple 1D thread blocks

#### Question:

For the vector addition problem considered in the CUDA template, find the solution based on the given thread configurations.

#### Solution:



#### Test data:



#### Sample output:



#### Reflection:

A 1D vector with multiple blocks has an index equal to threadIdx.x + blockIdx.x \* blockSize

#### Metadata:

Multiple 1D threads

#### Further information:

N/A

### Q3. Understand the thread indices for 2D blocks

#### Question:

List the values for the built-in variables threadIdx.x and threadIdx.y corresponding to given thread configurations used for executing the kernel addKernel( ) function on GPU

#### Solution:

N/A

#### Test data:



#### Sample output:



#### Reflection:

Only the first index of the dim3 is currently used.

#### Metadata:

Indices for 2D blocks

#### Further information:

N/A

### Q4. Find vector addition using one 2D thread block

#### Question:

For the vector addition problem considered in the CUDA template, find the solution based on the given thread configurations

#### Solution:



#### Test data:



#### Sample output:



#### Reflection:

The threads index in a 2D thread block is equal to threadIdx.x + threadIdx.y \* blockDim.x

#### Metadata:

One 2D block

#### Further information:

N/A

### Q5. Find vector addition using multiple 2D thread blocks

#### Question:

For the vector addition problem considered in the CUDA template, find the solution based on the given thread configurations

#### Solution:



#### Test data:



#### Sample output:



#### Reflection:

The thread index for multiple 2D blocks is equal to:

(threadIdx.x + blockIdx.x \* blockDim.x) + (threadIdx.y + blockIdx.y \* blockDim.y)

#### Metadata:

Multiple 2D blocks

#### Further information:

While I have successfully done the 2D block section of the lab I don’t feel as though I thoroughly understand it yet. Will ask if it can be explained to me again next lab

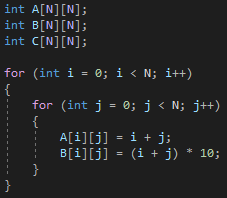
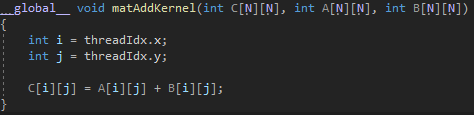
### Q6. Matrix addition

#### Question:

Write a CUDA program to find the addition of two matrices

#### Solution:







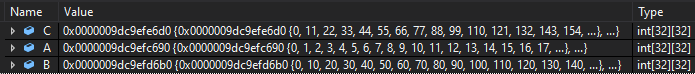


\*I’ve just included key parts that I think are important

#### Test data:

N/A

#### Sample output:



#### Reflection:

For the cudaMalloc, cudaMemcpy etc the size is as follows:



This task was very confusing I don’t really understand the dA,dB and dC variables and stuff like that

#### Metadata:

Matrix addition

#### Further information:

I think this task didn’t have enough guidance on the actual lab sheet, I had to google how to do it eventually as I ran out of time in the lab and even after rewatching the lectures this week multiple times I still couldn’t understand what I was doing wrong

## Week 3 – Lab 3

Date: 16/02/2022

### Q1. Vector dot-product

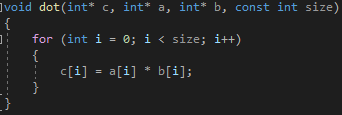
#### Question:

**Part 1 –** Write a C++ program to calculate the dot-product of two vectors

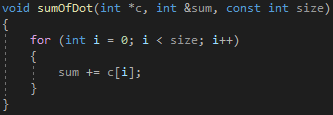
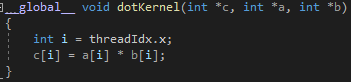
**Part 2 –** Write a C++ program to calculate the dot-product of two vectors on the GPU & calculate the sum of the dot-product of the two vectors on the CPU

#### Solution:

**Part 1 –**

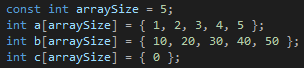
****

**Part 2 –**

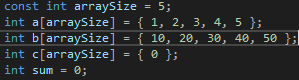
****

#### Test data:

**Part 1 –**

****

**Part 2 –**



#### Sample output:

**Part 1 –**

****

**Part 2 –**

****

#### Reflection:

This is pretty much the same as lab 1

#### Metadata:

Dot-product

#### Further information:

N/A

### Q2. Vector dot-product using unified memory

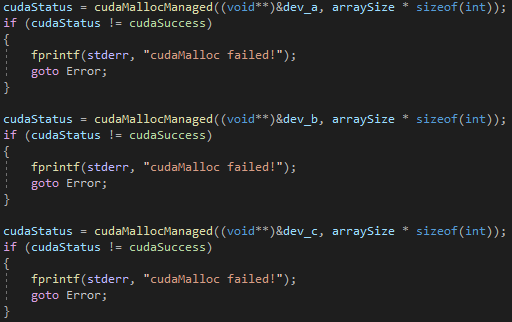
#### Question:

**Part 1 –** Create a CUDA program that calculates the vector dot-product using managed memory

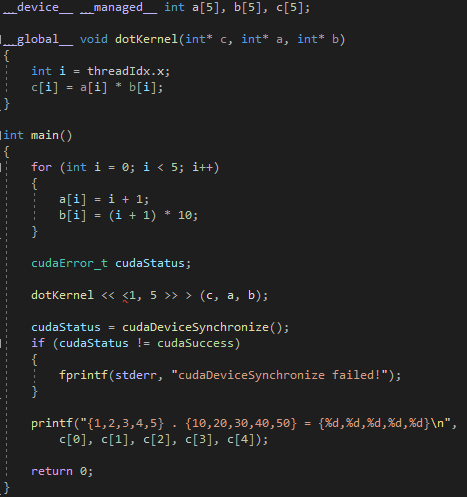
**Part 2 –** Create a CUDA program that calculates the vector dot-product using GPU-declared \_\_managed\_\_ memory

#### Solution:

**Part 1 –**

****

**Part 2 –**

****

#### Test data:

N/A

#### Sample output:

**Part 1 –**

****

**Part 2 –**

****

#### Reflection:

This way of doing things seems to cut out a lot of code compared to the methods that have been previously used in the labs

#### Metadata:

Unified memory

#### Further information:

N/A

### Q3. Vector dot-product using shared Memory

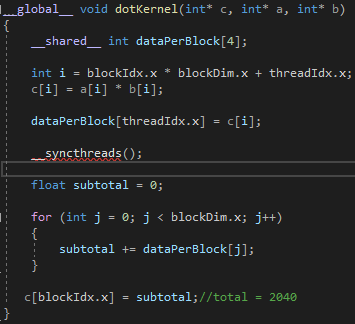
#### Question:

**Part 1 –** Analyse the process given in the lab sheet and identify areas where thread execution needs to be synchronized by calling CUDA function: \_\_syncthreads();

**Part 2 –** Consider different thread configurations, for example, <<>>, <<>>, <<>> and observe if the given program can calculate the vector dot-product correctly. If not, analyse the issues and consider how to fix them.

#### Solution:

**Part 1 –**

****

**Part 2 –** N/A

#### Test data:

**Part 1 –** N/A

**Part 2 –**

****

****

****

#### Sample output:

**Part 1 –**



**Part 2 –**







#### Reflection:

**Part 1 –** I don’t really understand what it is doing I get the same results with and without \_\_syncthreads()

**Part 2 –** The program only gets the right sum with the 2,4 layout. This could fixed by making the number of blocks and how many values of the c vector added together as the same.

#### Metadata:

Shared memory

#### Further information:

N/A

## Week 4 – Lab 4

Date: 23/02/2022

### Q1. Create an OpenGL-CUDA program based on a CUDA SDK sample

#### Question:

Set up the project and make sure it compiles

#### Solution:

N/A

#### Test data:

N/A

#### Sample output:

Reflection:

The hidden files need to made visible in the file explorer

#### Metadata:

CUDA SDK

#### Further information:

N/A

### Q2. Understand pixel colour

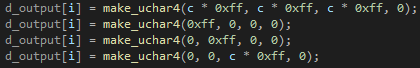
#### Question:

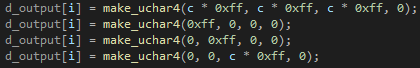
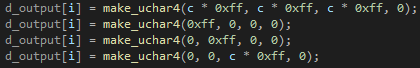
Edit the line “d\_output[i] = make\_uchar4(c \* 0xff, c \* 0xff, c \* 0xff, 0);” in various way to understand pixel colouring works

#### Solution:

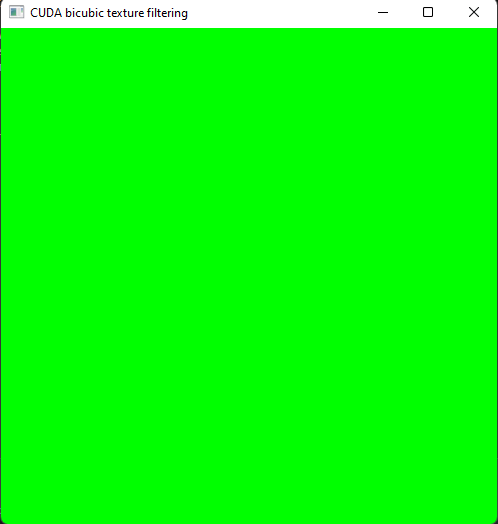
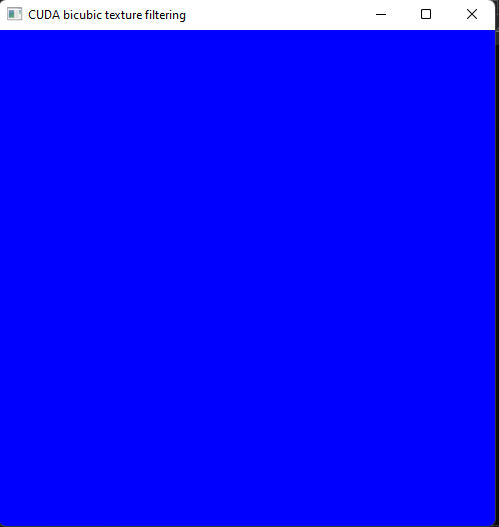
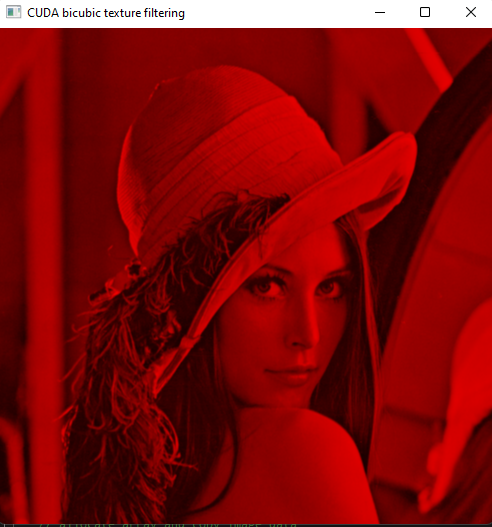
N/A

#### Test data:





#### Sample output:

#### Reflection:

For some reason colours are ordered BGR instead of RGB which is what I’m used to

#### Metadata:

Pixel colours

#### Further information:

N/A

### Q3. Image transformation

#### Question:

**Part 1 –** Translate the image

**Part 2 –** Scale the image

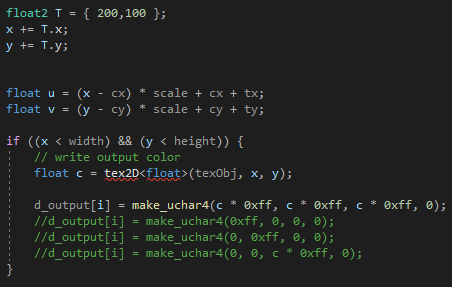
**Part 3 –** Rotate the image

**Part 4 –** Scale by position

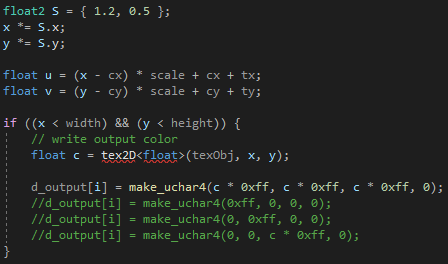
**Part 5 –** Rotate by image centre

#### Solution:

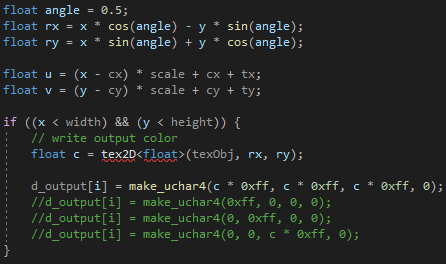
**Part 1 –**

****

**Part 2 –**

****

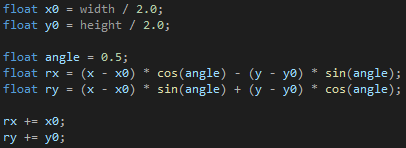
**Part 3 –**



**Part 4 –**



**Part 5 –**

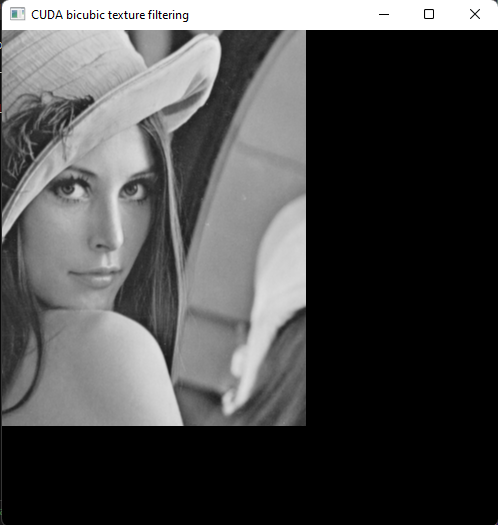


#### Test data:

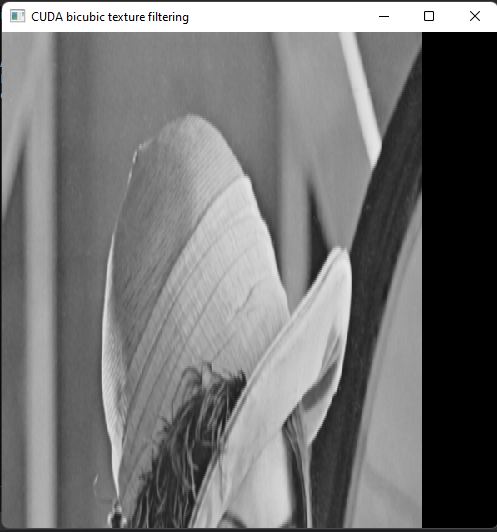
**All parts –** N/A

#### Sample output:

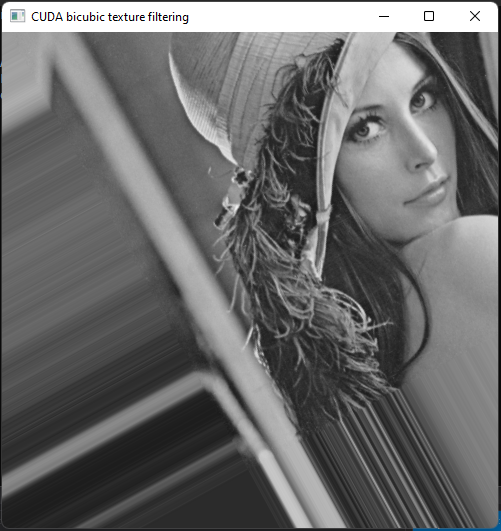
**Part 1 –**

****

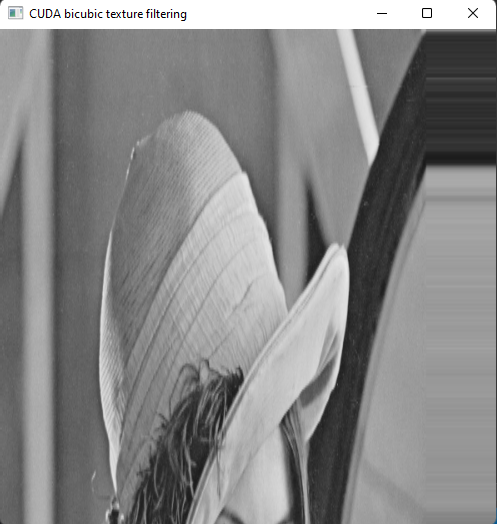
**Part 2 –**

****

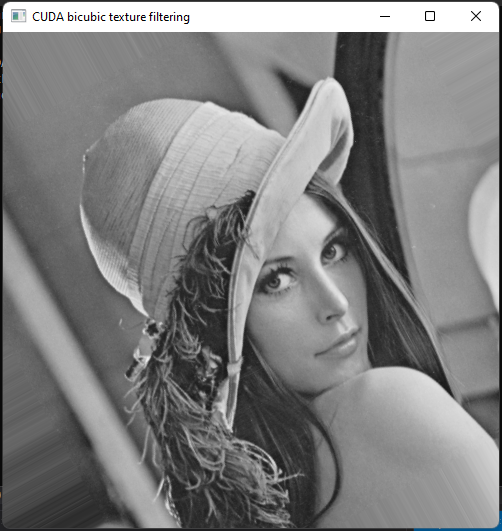
**Part 3 –**

****

**Part 4 –**

****

**Part 5 –**

****

#### Reflection:

These were all pretty straight forward apart for the rotating by image centre

#### Metadata:

Transformations

#### Further information:

N/A

### Q4. Image smoothing

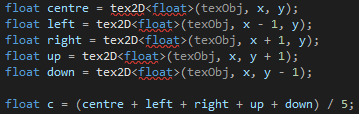
#### Question:

**Part 1 –** Implement image smoothing

**Part 2 –** Modify thread configurations and observe performance

#### Solution:

**Part 1 –**

****

#### Test data:

**Part 1 –** N/A

#### Sample output:

**Part 1 –**



**Part 2 –**

|  |  |
| --- | --- |
| **blocksize** | **Time** |
| 16x16 | 0.365568n |
| 32x32 | 0.437248n |
| 24x24 | 0.390144n |

#### Reflection:

**Part 1 –** This was pretty straight forward

**Part 2 –** The threads failed to execute when I tried 64x64 configuration

#### Metadata:

Smooth

#### Further information:

N/A

## Week 5 – Lab 5

Date: 02/03/2022

### Q1. A simple matrix multiplication program in CUDA using one thread block

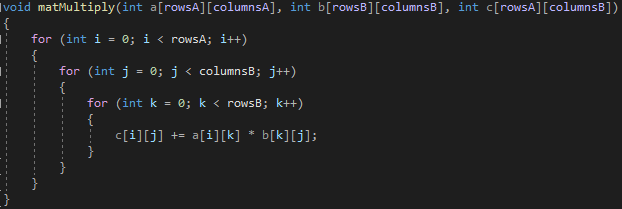
#### Question:

**Part 1 –** Write a matrix multiplication function in c++

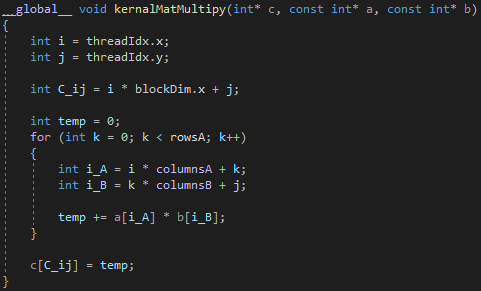
**Part 2 –** Write a matrix multiplication function in CUDA using 1 thread block

#### Solution:

**Part 1 –**

****

**Part 2 –**



#### Test data:

A = (1,2,3,4,5,6,7,8,9) B = (10,11,12,13,14,15,16,17,18)

#### Sample output:

C = (84,90,96,201,216,231,318,342,366)

#### Reflection:

I don’t really understand why you wouldn’t use 2D arrays

#### Metadata:

Matrix Multiply

#### Further information:

N/A

### Q2. Compare the performance of the CUDA solution against the CPU solution

#### Question:

**Part 1 –** 8x8

**Part 2 –** 32x32

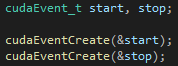
**Part 3 –** 256x256

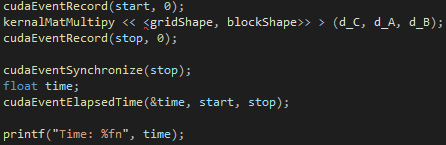
**Part 4 –** 512x512

**Part 5 –** 1024x1024

#### Solution:

**All Parts –**

****

****

#### Test data:

N/A

#### Sample output:

|  |  |
| --- | --- |
| **Part** | **Time** |
| 1 – CPU | 0.007168n |
| 1 – GPU | 0.036864n |
| 2 – CPU | 0.006144n |
| 2 – GPU | 0.059392n |
| 3 – CPU | 0.004192n |
| 3 – GPU | 0.008224n |
| 4 – CPU | Stack overflow |
| 4 – GPU | Stack overflow |
| 5 – CPU | Stack overflow |
| 5 – GPU | Stack overflow |

#### Reflection:

My timings are really weird and inconsistent but I have no clue why. I had similar issues with a rust lab session.

#### Metadata:

Performance

#### Further information:

N/A

## Week 6 – Lab 6

Date: 09/03/2022

### Q1. Set up a virtual canvas and draw on it an image in CUDA

#### Question:

Set up the project then make the image green

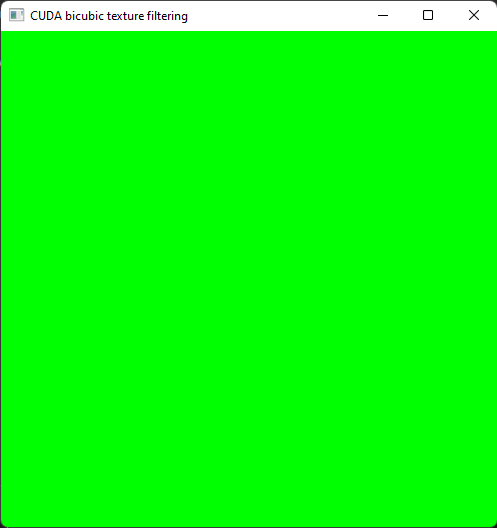
#### Solution:



#### Test data:

N/A

#### Sample output:



#### Reflection:

This was really straight forward

#### Metadata:

Set up

#### Further information:

N/A

### Q2. Drawing a checkboard in CUDA

#### Question:

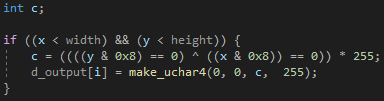
**Part 1 –** Modify the code to produce a checkerboard image

**Part 2 –** Modify the code to produce larger red-blocks in the checkerboard

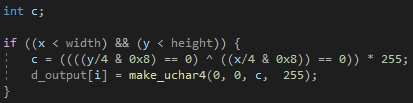
**Part 3 –** Modify the code to produce a red disc

#### Solution:

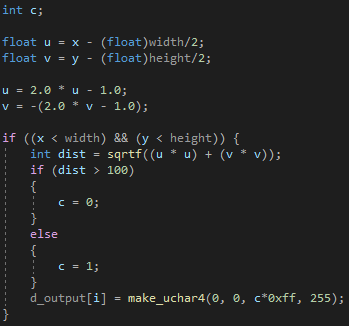
**Part 1 –**

****

**Part 2 –**

****

**Part 3 –**

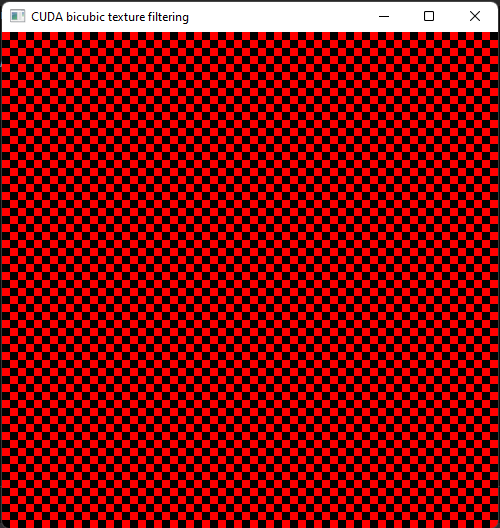
****

#### Test data:

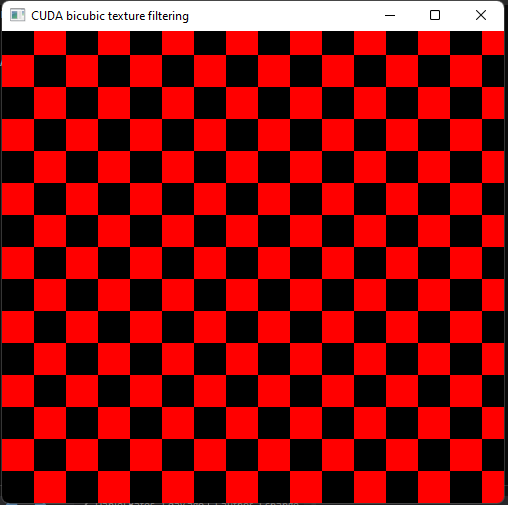
**All Parts –** N/A

#### Sample output:

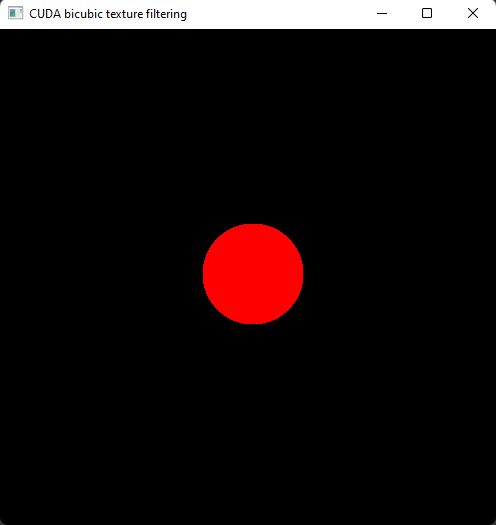
**Part 1 –**



**Part 2 –**

****

**Part 3 –**

****

#### Reflection:

The circle drawing took me a while to do but I think I was just over complicating it. Other than that, everything else was very straight forward

#### Metadata:

Checkers & Discs

#### Further information:

N/A

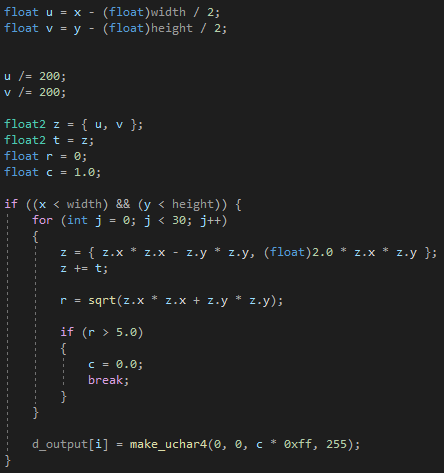
### Q3. Drawing the Mandelbrot and Julia Sets

#### Question:

Implement Mandelbrot and Julia sets

#### Solution:

**Mandelbrot Set –**



**Julia Set –**

Same as Mandelbrot set except the following:

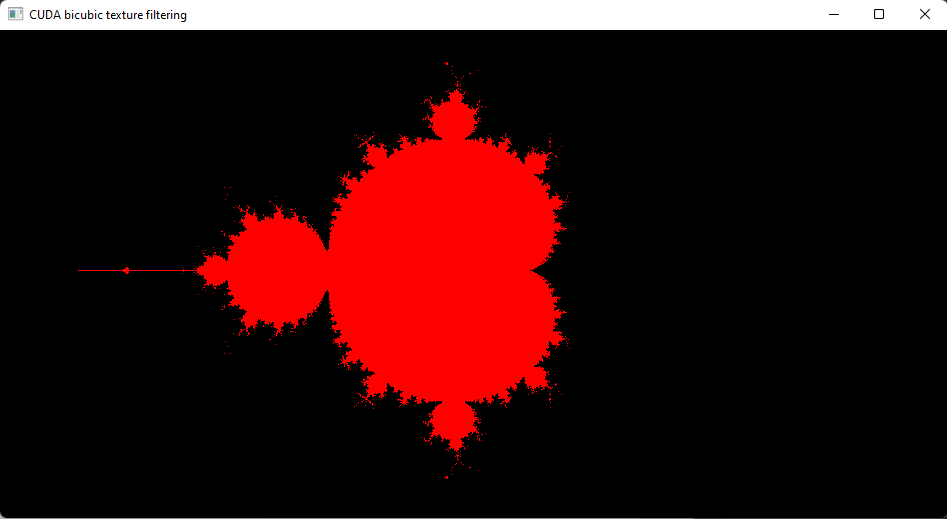


#### Test data:

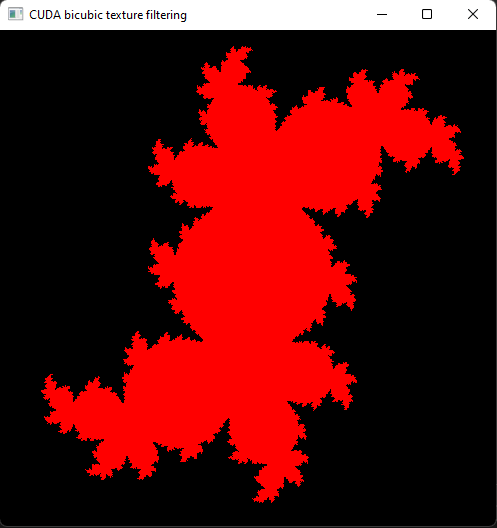
N/A

#### Sample output:

**Mandelbrot Set –**



**Julia Set –**



#### Reflection:

Fairly straightforward but to scale the image size I have to divide the u and v by 200 instead of multiply by 4.0 and I’m not sure why this was the case.

#### Metadata:

Mendelbrot & Julia

#### Further information:

N/A

## Week 7 – Lab 7

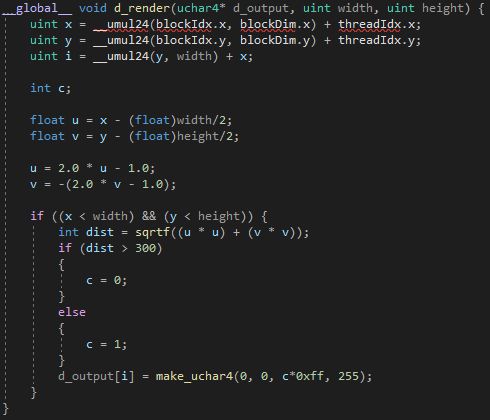
Date: 16/03/2022

### Q1. Drawing based on a canvas of size [-1, 1]x[-1, 1]

#### Question:

Draw a disc based on pixel coordinates defined in float type variables in [-1, 1]x[-1, 1]

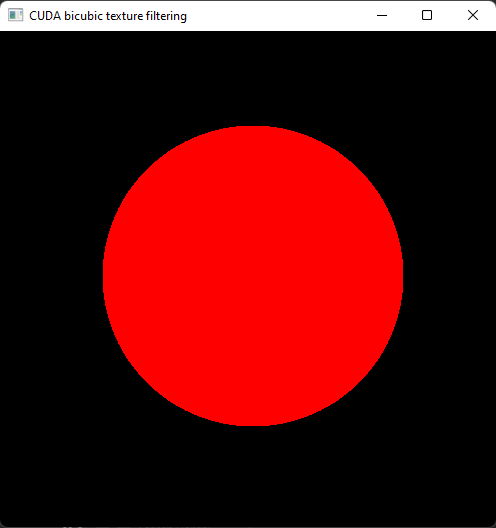
#### Solution:



#### Test data:

N/A

#### Sample output:



#### Reflection:

This was just repeating a task of lab 6 so I just copied the code I used for task 4 of lab 6

#### Metadata:

Drawing Disc

#### Further information:

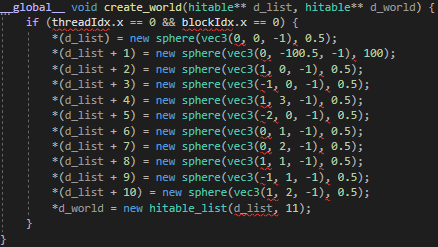
N/A

### Q2. Write a simple ray caster

#### Question:

Write a simple ray caster by implementing the code given in the lab sheet. Then add ten new spheres to the world

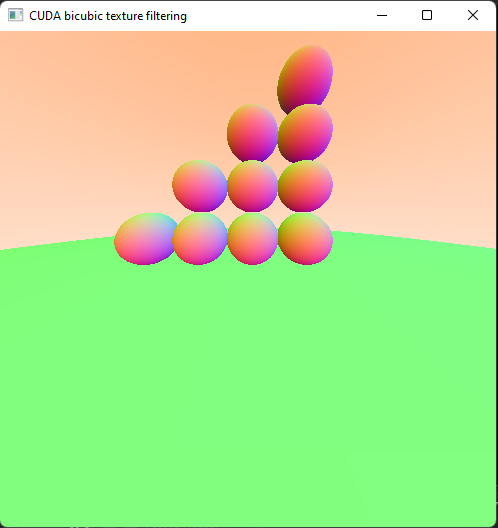
#### Solution:



#### Test data:

N/A

#### Sample output:



#### Reflection:

I’m not sure what the following line is doing:



I think its serving as a skybox but I’m not sure.

#### Metadata:

Spheres

#### Further information:

N/A

## Week 8 – Lab 8

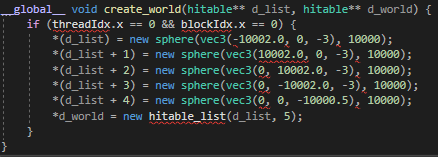
Date: 23/03/2022

### Q1. Draw a box without front wall

#### Question:

Using the spheres create a box without a front wall in the create world method

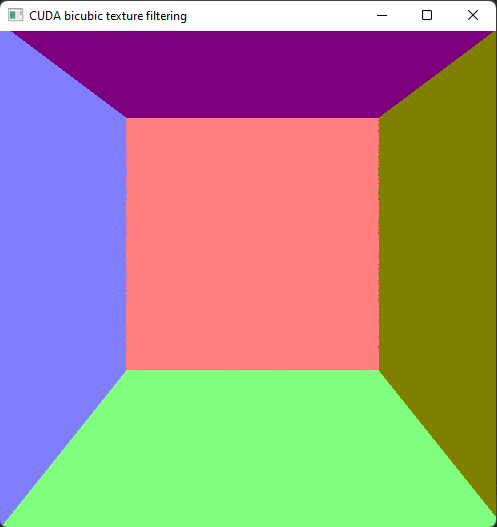
#### Solution:



#### Test data:

N/A

#### Sample output:



#### Reflection:

This was very simple to implement. I’m not sure how to implement the other way the lab sheet mentions though.

#### Metadata:

Box

#### Further information:

N/A

### Q2. Free motion animation

#### Question:

Draw a ball rotating by the centre of the box

#### Solution:

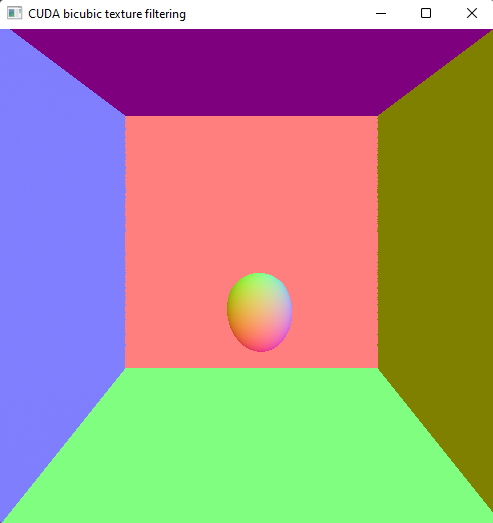




#### Test data:

N/A

#### Sample output:



#### Reflection:

The rotating sphere becomes distorted into and egg shape but I’m not sure why this is

#### Metadata:

Rotation

#### Further information:

N/A

### Q3. Ball-box walls collision animation

#### Question:

Animate a ball in motion with an initial velocity that is reflected when the ball collides with a wall. When a collision takes place the ball should also change colour

#### Solution:

#### Test data:

#### Sample output:

#### Reflection:

I can’t get it to work properly at all and the way it was explained to me to do in the lab sessions causes multiple errors which means the code won’t compile. I’m not sure if this is because I’m on version 11.6 instead of 11.5

#### Metadata:

#### Further information:

# Final Lab

## GPU Design

The plan for the GPU implementations was use 1 block with 1 thread for each particle in the particle system for both the dynamics and the temperature. For the collisions I was planning on splitting each collision type, wall-particle collisions, floor-particle collisions and particle-particle collision, into separate blocks with each block having as many threads as there are particles.

## CPU Design

The particle dynamics and particles temperature were handled over 2 threads each in a 2 separate scoped thread pools, I chose to use the scoped thread pools so I could utilise chunking to split the work between threads and so the ownership of the particle system would be less of an issue. In the particle dynamics thread pool, before the particle system is chunked and the threads run, 5 new particles are created and added to the particle system. This keeps a continues stream of particles being produced.

The collisions were handled slightly differently. Since I didn’t want to chunk the particle system, so I could check to see if any 1 particle is colliding with any other, and I wanted each collision being checked to be done in its own thread. I used 3 threads, 1 for wall-particle collisions, 1 for floor-particle collisions and 1 for particle-particle collision. This method requires me to use an Arc and Mutex for the particle system to avoid any ownership issues.

However, by using these 2 different methods there are issues that happen with glium since the particle system gets moved into closure by putting it in an Arc so the dynamics and temperature threads can’t access the particle system to chunk and use it. So, the dynamics and temperature threads had to be changed to use the Arc and Mutex for the particle system.

## Performance comparison

I left my GPU implementation too late so unfortunately I don’t have enough to do a comparison between implementations.

## Reflection

In hindsight I think I spent too much time focusing on the visualisation aspects, since I didn’t really understand it, when I should have focused on getting the functions and threading done first then added visualisation later on. I also spent too much time on just the CPU implementation as a whole and should have left more time to work on the GPU implementation, especially as that is the side that my knowledge and understanding of is weaker. I also think that the way I approached the tasks was wrong, instead of doing a rough design for everything and then trying to get it all to work together, I should have done one aspect at a time, so focus on dynamics then temperature etc.