KIT308 Multicore Architecture and Programming

Assignment 3 — OpenCL

程序代写代做 CS编程辅导

Aims of the assignment

The purpose of this assignment is to give you experience at writing a program using OpenCL programming techniques. This assignment will give you an opportunity to demonstrate your understanding of:

- setting up OpenCL structures
- passing memory from the CP
- creating memory formats tha
- the translation of C/C++ cod

Due Date

11:55pm Friday 15th of Octo ster)

- Late assignments will only be rcumstances and provided that the proper procedures have been followed (see the School Office or this link for details) assignments which are submitted late without good reason will be subject to mark penalties if they are accepted at all (see the School office or this link for details on this as well).
- Forms to request extensions of time to submit assignments are available from the Discipline of ICT office. Requests must be accompanied by suitable documentation and should be submitted **before the assignment due date**.

WeChat: cstutorcs

Assignment Submission

Your assignment is to be submitted electronically via MyLO and should contain:

- An assignment cover sheet;
- A .zip (or .rar) containing a Visual Studio Solution containing a project for each attempted stage of the assignment (in the format provided in the downloadable materials provided below).
- A document containing:
  - A table of timing information comparing the original provided base code times against the Stage 4 OpenCL implementation on each scene file.
  - An analysis of the above timing data.
- **You do not need to (and shouldn't) submit executables, temporary object files, or images.** In particular, you **must** delete the ".vs" directory before submission as it just Visual Studio temporary files and 100s of MBs. Do not however delete the "Scenes" folder or the "Outputs" folder (but do delete the images within this one).
- **NOTE: you MUST ensure to compile your *.cl* code with `clBuildProgram` using the *-cl-std=CL1.2* compiler option. Failure to do so may means that your code does not compile on the marking machine and you will receive penalties as a result.**

Assignment Project Exam Help
Email: tutorcs@163.com
QQ: 749389476
https://tutorcs.com

Marking

This assignment will be marked out of 100. The following is the breakdown of marks:

Task / Topic	Marks
1. OpenCL Setup and Data Transfer	**50%**
CPU-side	
Correct OpenCL setup and kernel creation	5%
Well thought out and correct Open ments for representing scene data	5%
Correct creation of two-dimensiona	5%
Correct memory management (i.e	5%
Inclusion of (helpful) error handlin	5%
Correct alignment of datastructure	5%
GPU-side	
Correct declaration of equivalent OpenCL types and datastructures (for primitives, sceneobjects, etc.)	10%
Output (via printf) that verifies that the scene has been correctly transferred (for full marks, this information should only print once per execution)	5%
Correctly fills output buffer with colour	5%
2. Basic Rendering	**10%**
OpenCL implementation of renderer that colours spheres white and everything else black (i.e. no lighting, material application, reflection, refraction, or shadowing, etc.) NOTE: this stage should ignore the *-samples* command-line argument, i.e. no sure-sampling, each pixel should only be sampled once	5%
Correctly handles all scenes	5%
3. Rendering with Lighting	**10%**
OpenCL implementation of renderer with correct lighting for spheres, planes, and cylinders but no shadows or complex materials (and also no supersampling)	5%
Correctly handles all scenes	5%
4. Full Rendering	**10%**
OpenCL implementation of renderer with correct lighting, shadowing, complex materials, reflection, refraction, and supersampling	5%
Correctly handles all scenes	5%
5. Rendering Large / Complex Images	**10%**
OpenCL implementation (with all features of Stage 4) that renders all the Stage 5 tests by splitting up the render job into multiple smaller parts (i.e. like Stage 2 or 3 of assignment 1)	5%
Added a command-line option to be able to control the size of the individual OpenCL jobs / parts (e.g. `-blockSize` or `-chunkSize`)	5%
Documentation	**10%**
Outputs showing timing information for Stage 4 on all applicable scene files	5%
Analysis of data (comparison between Stage 4 execution times and single-threaded, multi-threaded, and SIMD versions of the code)	5%
Penalties	
Failure to comply with submission instructions (eg. no cover sheet, incorrect submission of files, abnormal solution/project structure, **using incorrect OpenCL version**, etc.)	-10%
Poor programming style (eg. insufficient / poor comments, poor variable names, poor indenting, obfuscated code without documentation, compiler warnings, etc.)	up to -20%
Lateness (-20% for up to 24 hours, -50% for up to 7 days, -100% after 7 days)	up to -100%

程序代写代做 CS编程辅导
WeChat: cstutorcs
Assignment Project Exam Help
Email: tutorcs@163.com
QQ: 749389476
https://tutorcs.com

Programming Style

This assignment is not focussed on programming style (although it is concerned with efficient code), but you should endeavour to follow good programming practices. You should, for example:

- comment your code;
- use sensible variables names;
- use correct and consistent indenting; and
- internally document (with comments) any notable design decisions.

[NOTE: any examples in the provided assignment materials that don't live up to the above criteria, should be considered to be deliberate examples of what not to do and are provided to aid your learning ;P]

The Assignment Task

You are to heavily modify a slightly simplified implementation of our original single-threaded "simple" raytracer so that in runs as an OpenCL program. To fully complete this assignment (to Stage 4), you will need to convert everything from `render` onwards (i.e. all the functionality of the `render` function, every function that is called from `render`, and everything that is called from there, and so on) to be written in OpenCL.

From the provided single-threaded raytracer implementation, you will create multiple subsequent versions that convert various parts of the program to be implemented in OpenCL:

1. Code to start an OpenCL Kernel and transfer the `scene` datastructure (no easy task).
2. Basic rendering that just sho erything else as black (i.e. `traceRay` working to to the point of `objectIntersection` working etc.).
3. Rendering with support for p c lighting complete.
4. Full rendering with all feature CL.
5. Rendering of large / complex

Implementation

1. OpenCL Initialisation and

This stage involves initialising OpenCL (getting the platform, context, etc. etc.) as we've done in the tutorials. However, it requires careful thought when deciding how to pass through data to the OpenCL kernel as all the parameters to render (e.g. `width`, `height`, `aaLevel`) need to be available in the kernel and this includes the entire scene object.

You can't simply pass a pointer to the scene, as any objects more complicated than scalars need to be allocated to a buffer (so that they can be available to the GPU). Even if you allocate the scene object to a buffer, the containers for the scene objects won't transfer over automatically, so you'll need to think about how to transfer them as well.

Additionally, to be able to use the scene in OpenCL, you'll need an equivalent definition of the `Scene` struct which in turn means you'll need definitions of scene objects and primitives. Some of these structs (e.g. Point, Vector, Colour, etc.) can be represented by built-in OpenCL types to greatly simplify your implementation (and gain access to a host of built-in functions).

Memory layout for structs on the CPU and GPU is not likely to be the same, and so you need to take great care to ensure that each struct has the same size on both devices. This means that on the CPU-side, in order to get the memory alignment you want, you may need to either manually pad structs, use the pre-defined OpenCL types (e.g. `cl_float3`), or make use of the `_declspec(align(X))` command.

You also need to be explicit about what kind of memory different variables are. Many of your datastructures and pointers to datastructures will need to be declared `__global`.

In order to complete this step you will need to:

- Have completely transferred a copy of the scene (including the scene object containers) to the GPU and verified this via outputting the whole structure.
- Filled the entire output buffer with colour to ensure that your kernel is working over the correct range.
    - Each pixel should be coloured as *RGB(yCoordinate % 256, 0, xCoordinate % 256)*
      (i.e. when working with 8 bits per channel RGB pixel values, the blue-channel of the pixel will be the x-coordinate of the pixel modulo 256, and the red channel of the pixel will be the y-coordinate of the pixel modulo 256).

At the end of this stage the program should successfully transfer all scene information to the OpenCL kernel and be able to create a full image (filled with coloured pixels). Be thorough in your testing (i.e. test all the scenes) to ensure that everything works correctly before progressing.

Hints / Tips

- There is a commented-out function `outputInfo` (and call to the function in `main`) that you can use to display helpful values about the scene for testing. This code can be duplicated in OpenCL and called from your kernel to confirm that everything has transferred correctly.
- Just make a basic kernel to begin with and transfer only the scene object. Verify that it has transferred correctly by using printf (this is available in OpenCL (huzzah!) at the cost of greatly reduced performance) on the CPU and on the GPU (use the work offset values to ensure you only print stuff once).
- In OpenCL, rather than defining structs for many of the renderer types you can simply typedef OpenCL data types. e.g. use "`typedef well_chosen_type Point;`" instead of a struct.
- Memory alignment is a big issue. Never assume that something has transferred correctly to the OpenCL program — verify that it has via `printf`.
- Pay attention to where in memory your datastructures are. You will need to use `__global` and/or `__constant` to avoid compiler errors/warnings.
- Your kernel function will be a conversion of the render function, but at this stage it doesn't need to do anything except get all of the same parameters, output the scene (via `printf`), and fill the image in with a colour. Refer to the introductory OpenCL tutorials for help making a basic kernel.
- There are a number of possible problems you may encounter when loading OpenCL files:
    - Due to how the assignment solution/projects are set up to use shared folders for scenes, outputs, and reference images, you'll need to specify the path of your OpenCL file when using `clLoadSource` (e.g. "Stage1/Raytrace.cl"). This also applies for any further files you `#include` within OpenCL as below.
    - Rather than create one giant OpenCL source file, you would be wise to split your code across multiple files and `#include` them from within the OpenCL file you load using the `clLoadSource` function. You'll need to ensure that you define things in the correct order (sometimes function prototypes can help here).
    - Some OpenCL implementations have a problem with caching old versions of included files, you may wish to manually clear this cache if you have trouble, or set up a pre-build event to do it for you (see this stackoverflow thread for more info).

NOTE: this is perhaps the hardest stage of the assignment.

NOTE: you MUST ensure to compile your code with `clBuildProgram` using the *-cl-std=CL1.2* compiler option. Failure to do so may mean that your code does not compile on the marking machine and you will receive penalties as a result. To do this, use code similar to the following:

```
clBuildProgram(program, 0, NULL, "-cl-std=CL1.2", NULL, NULL);
```

程序代写代做 CS编程辅导
WeChat: cstutorcs
Assignment Project Exam Help
Email: tutorcs@163.com
QQ: 749389476
https://tutorcs.com

2. Basic Rendering

This stage involves implementing some of the functionality of the traceRay function to at least detect whether a collision with an object has occured or not. As you only need to add colour when a collision with a sphere has been detected you can either check for collisions with all kinds of objects and be selective, or just not check for collisions with boxes at this stage.

Hints / Tips

程序代写代做 CS编程辅导

- Be really careful when converting * operations. Much of the original code overloads * to be dot-product and this will not automatically translate in the OpenCL program (it may compile, but instead just do a SIMD like multiplication) — there is a `dot(X, Y)` function that you can use.
- All of the *stdlib* math functio[illegible] OpenCL functions. Most of the time this is as simple as removing "f" from the end of the function name.

3. Rendering with Boxes and [illegible]

This stage involves including the int[illegible] and cylinders and adding in the lighting calculations.

For this stage you should treat all n[illegible]e gourard shading and you don't need to implement checkerboards, circles, or wood textures (i.e. when applying c[illegible] material's `diffuse` value).

4. Full Rendering

This stage involves including the materials, shadowing, reflection, refraction, etc.

5. Full Rendering of Large / Complex Images

WeChat: cstutorcs

This stage is a fairly artificial task as modern GPUs can handle large jobs without locking up the rest of the OS / causing crashes. However, the intent is for you show that you understand how to schedule multiple OpenCL jobs to complete one large task.

This stage involves trying to render images that require a large amount of time to generate. **NOTE: on some system configurations, trying to run such a large OpenCL job may cause stability issues** (in this situation, this step of the assignment isn't an artificial task at all).

Assignment Project Exam Help

This stage involves creating a solution that is capable of rendering these tests via rendering the whole image in parts (i.e. in a similar fashion to Stage 2 (chunks) or Stage 3 (blocks) of Assignment 1. However the image is subdivided, you should provide a command-line argument that allows for control of how big each segment is.

Email: tutorcs@163.com

Hints / Tips

- Techniques we've explored for Assignment 1 should be helpful here.
- You do not need to wait for the longer tests to complete if you are confident you have created a solution that works on all of them (some of them take may take a very long time to produce a result on some systems).

QQ: 749389476

https://tutorcs.com

Documentation

After completing stage 4 of the assignment (anything before a complete stage 4 solution would produce incomparable times) you should provide:

- timing information for each scene file / ...
  - The base single-threaded code (given as the RayTracerAss3 project base);
  - The completed multi-threaded code (i.e. Assignment 1 solution) using a thread count matching the maximum number of logical processors in your system and a block size of 16;
  - The completed SIMD code (i.e. Assignment 2 solution) using a thread count matching the maximum number of logical processors in your sys...
  - The time taken for the ... a Stage 4 implementation); and
  - The average time take... ...he remaining runs over 9 runs.
- an explanation of the results ... ...ence between the performance of Stage 4 compared to the base code, or why a particular implementation wo... ...rticular scene, etc.).

程序代写代做 CS编程辅导

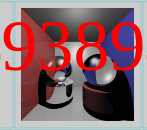
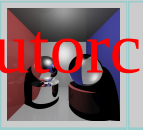
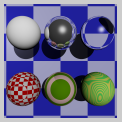
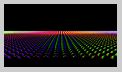
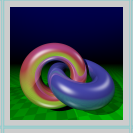
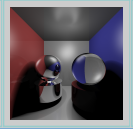
Tests / Timing

The following tables list all the tests ... ...enerate correctly at each stage. They also shows the timing tests that need to be performed in order to fully comp... ...tion of the assignment. Fully completing this tests may take up to an hour (with the 1 run for the previous versions, and 10 required runs for OpenCL) on some hardware, so plan your time accordingly.

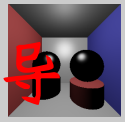
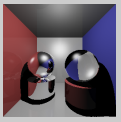
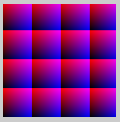
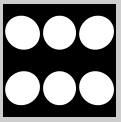
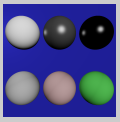
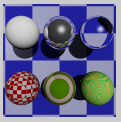
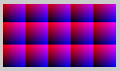
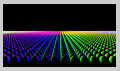
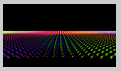
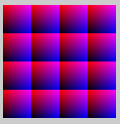
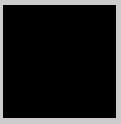
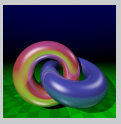
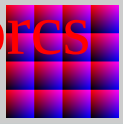
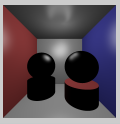
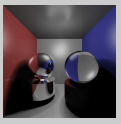
In order to confirm your images match the images created by the base version of the assignment code, it's strongly recommended you use a image comparison tool. For part of the marking for this, Image Magick will be used (as it was in Assignment 1 and 2).

NOTE: all debug `printf` outputs for normal executions (included any output added in order to complete stage 1 of the assignment) should be removed before timing stage 4 and 5 — however, you should NOT remove error `printf`s that occur if something goes wrong in the OpenCL setup process.

WeChat: cstutorcs
Assignment Project Exam Help
Email: tutorcs@163.com
QQ: 749389476
https://tutorcs.com

	Timing Test		Base Single-Threaded	Base Multi-Threaded (i.e. Ass1 Solution)	Multi-Threaded SIMD (i.e. Ass2 Solution)	Stage 4 OpenCL First Run	Stage 4 OpenCL Average of 9 Runs
1.	-input Scenes/cornell.txt -size 1024 1024 -samples 1						
2.	-input Scenes/cornell.txt -size 1024 1024 -samples 4						
3.	-input Scenes/cornell.txt -size 1024 1024 -samples 16						
4.	-input Scenes/allmaterials.txt -size 1000 1000 -samples 4						
5.	-input Scenes/5000spheres.txt -size 1280 720 -samples 1						
6.	-input Scenes/donuts.txt -size 1024 1024 -samples 1						
7.	-input Scenes/cornell-199lights.txt -size 1024 1024 -samples 1						

The following tests will be run on your code for each scene file and compared against the reference output shown:

Test	Stage 1	Stage 2	Stage 3	Stage 4
1. -input Scenes/cornell.txt -size 256 256 -samples 1				
2. -input Scenes/allmaterials.txt -size 1000 1000 -samples 4				
3. -input Scenes/5000spheres.txt				
4. -input Scenes/donuts.txt -size				
5. -input Scenes/cornell-199lights.txt -size 1024 1024 -samples 1				

程序代写代做 CS编程辅导
WeChat: cstutorcs
Assignment Project Exam Help

The following tests will be run on your code for Stage 5:

Test	Stage 5
1. -input Scenes/donuts.txt -size 2048 2048 -samples 1	
2. -input Scenes/donuts.txt -size 2048 2048 -samples 2	
3. -input Scenes/donuts.txt -size 2048 2048 -samples 4	
4. -input Scenes/donuts.txt -size 2048 2048 -samples 8	
5. -input Scenes/donuts.txt -size 2048 2048 -samples 16	
6. -input Scenes/donuts.txt -size 2048 2048 -samples 32	

Email: tutorcs@163.com
QQ: 749389476
https://tutorcs.com

Provided Materials

The materials provided with this assignment contain:

- The source code of the base version of the raytracer (i.e. the original starting point of Assignment 1).
- A set of scene files to be supplied to the program.
- A set of reference images for testing.
- Some batch files for testing purposes.

Download the materials as a ZIP file.

程序代写代做 CS编程辅导

Source Code

The provided MSVC solution contain

RayTracerAss3

The provided code consists of 21 sc

- **Raytracing logic:**
  - *Raytrace.cpp*: this file contains the `main` function which reads the supplied scene file, begins the raytracing, and writes the output BMP file. The main render loop, ray trace function, and handling of reflection and refraction is also in this file.
  - *Intersection.h* and *Intersection.cpp*: these files define a datastructure for capturing relevant information at the point of intersection between a ray and a scene object and functions for testing for individual ray-object collisions and ray-scene collisions.
  - *Lighting.h* and *Lighting.cpp*: these files provide functions to apply a lighting calculation at a single intersection point.
  - *Texturing.h* and *Texturing.cpp*: these files provide functions for the reading points from 3D procedural textures.
  - *Constants.h*: this header provide constant definitions used in the raytracing.
- **Basic types**:
  - *Primitives.h*: this header contains definitions for points, vectors, and rays. It also provides functions and overloaded operators for performing calculations with vectors and points.
  - *SceneObjects.h*: this header file provides definitions for scene objects (ie. materials, lights, spheres, and boxes).
  - *Colour.h*: this header defines a datastructure for representing colours (with each colour component represented as a `float`) and simple operations on colours, including conversions to/from the standard BGR pixel format.
- **Scene definition and I/O**:
  - *Scene.h* and *Scene.cpp*: the header file contains the datastructure to represent a scene and a single function that initialises this datastructure from a file. The scene datastructure itself consists of properties of the scene and lists of the various scene objects as described above. The implementation file contains many functions to aide in the scene loading process. Scene loading relies upon the functionality provided by the `Config` class.
  - *Config.h* and *Config.cpp*: this class provide facilities for parsing the scene file.
  - *SimpleString.h*: this is helper string class used by the `Config` class.
- **OpenCL I/O**:
  - *LoadCL.h* and *LoadCL.cpp*: these files contain a helper function for loading OpenCL files. These are not required for the base code, but are useful for all assignment Stages.
  - **NOTE: *LoadCL.h* contains the line to include the OpenCL header files, and also ensures you will use version 1.2 of the OpenCL APIs.**
- **Image I/O**:
  - *ImageIO.h* and *ImageIO.cpp*: these files contain the definitions of functions to read and write BMP files.
- **Miscellaneous**:
  - *Timer.h*: this class provides a simple timer that makes use of different system functions depending on whether `TARGET_WINDOWS`, `TARGET_PPU`, or `TARGET_SPU` is defined (we don't use the latter two, but I left this file unchanged in case anyone wanted to see how such cross-platform stuff can be handled).

WeChat: cstutorcs
Assignment Project Exam Help
Email: tutorcs@163.com
QQ: 749389476
https://tutorcs.com

Stage1 – Stage5

These projects are empty.

To begin work on the assignment you should (in Windows Explorer) copy all of the 21 *.h* and *.cpp* files from *RaytracerAss3* into the *Stage1* folder and then right-click on the *Stage 1* in Visual Studio and choose "Add / Exiting Item..." and add those 21 files.

Executing

The program has the following functionality:

- By default it will attempt to load the scene "Scenes/cornell.txt" and render it at 1024x1024 with 1x1 samples.
- By default it will output a file named "Outputs/[*scenefile-name*]_[*width*]x[*height*]x[*sample-level*]_[*executable-filename*].bmp" (e.g. with all the default options, "Outputs/cornell.txt_1024x1024x1_RayTracerAss2.exe.bmp")
- It takes command line arguments that allow the user to specify the width and height, the anti-aliasing level (must be a power of two), the name of the source scene file, the name of the destination BMP file, and the number of times to perform the render (to improve the timing information).
- Additionally it accepts an argument for whether each thread will instead colour the area rendered by the thread as a solid tint based on the x,y coordinates of each pixel.
- It loads the specified scene.
- It renders the scene (as many times as requested).
- It produces timing information for the first run, and the average of the time taken for all subsequent renders produced, ignoring all file IO.
- It outputs the rendered scene as a BMP file.

For example, running the program at the command line with no arguments would produce output similar to the following (as well as writing the resultant BMP file to *Outputs/cornell.txt_1024x1024x1_RayTracerAss3.exe.bmp*):

```
first run time: 3847ms, subsequent average time taken (0 run(s)): N/A
```

Adding *-runs 10* to the arguments would produce output similar to the following:

```
first run time: 3891ms, subsequent average time taken (9 run(s)): 3814.2ms
```

Testing Batch Files

A number of batch files are provided that are intended to be executed from the command line, e.g.

- For timing:
  - `baseTiming.bat` will perform all the timing tests required for the base single-threaded code (you'll have to make your own to do the multi-threaded and SIMD tests, or just run those manually).
  - `stage4Timing.bat` will perform all the timing tests required for Stage 4.
- For testing (requires Image Magick installation), e.g.:
  - `stage1Tests.bat` will p… required for Stage 1 Tests.
  - `stage2Tests.bat` will p… required for Stage 2 Tests.
  - `stage3Tests.bat` will p… required for Stage 3 Tests.
  - `stage4Tests.bat` will p… required for Stage 4 Tests.
  - `stage5Tests.bat` will p… required for Stage 5 Tests (but you'll need to edit it to add your custom command-line options
  - (`baseTests.bat` will co… aded code running on your machine against the reference images — this is just to confirm nothing str…

程序代写代做 CS编程辅导

WeChat: cstutorcs

Assignment Project Exam Help

Email: tutorcs@163.com

QQ: 749389476

https://tutorcs.com