COMP 429/529 Parallel Programming: Project 2

Due: Midnight, December 22, 2023

Notes: You may discuss the problems with your peers but the submitted work must be your own work. Late assignment will be accepted with a penalty of -10 points per day up to three days. Please submit your source code through blackboard. This assignment is worth 20% of your total grade. **You have the option to do this project either individually or with a partner. However, team sizes are limited to two only.**

Corresponding TA: Ilyas Turimbetov (iturimbetov18@ku.edu.tr)

TA office hours: Tuesdays 17:30-19:00, Thursdays 13:00-14:30 or by appointment if preferred, Location Eng230

Description

In this assignment you will implement a parallel version of the Marching Cubes algorithm in CUDA. Additionally, the algorithm transforms (twists) the initial shape and saves the output at every step. You don't have to fully understand all the details about the algorithm and the data structures it uses, but some level of understanding is useful.

Marching Cubes

Marching Cubes is a simple algorithm for creating a triangle mesh from a uniform grid of cubes superimposed over a region of the function. Each vertex in the mesh defines whether it's the inside or the outside of a shape that is being represented. If all 8 vertices of the cube are positive, or all 8 vertices are negative, the cube is entirely above or entirely below the surface and no triangles are emitted. Otherwise, the cube straddles the function and some triangles and vertices are generated. Since each vertex can either be positive or negative, there are technically 2^8 possible configurations, but many of these are equivalent to one another. There are only 15 unique cases (other cases are rotated version of these), shown here:

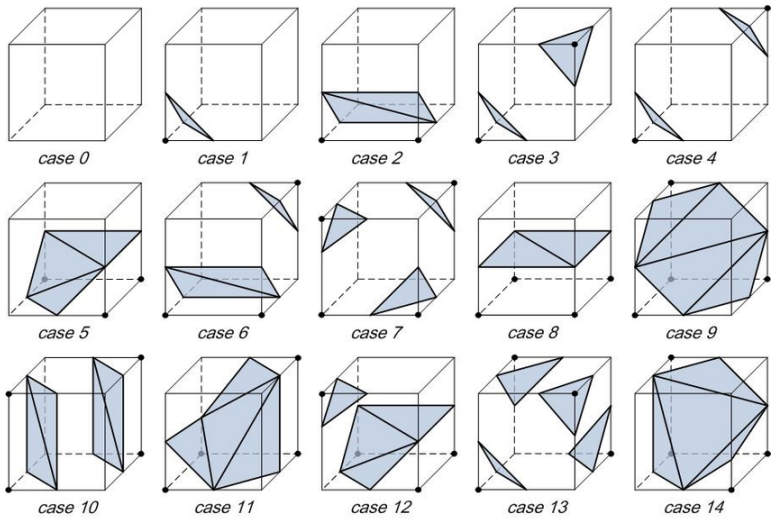

Figure 1: 15 unique triangle meshing configurations.

You can see an example of meshing a sphere on a figure below. (a) shows a binary (voxel) grid. It can serve as an input to the Marching Cubes algorithm to generate smoother surfaces at arbitrary angles, to generate a shape show on Figure 2 (b).

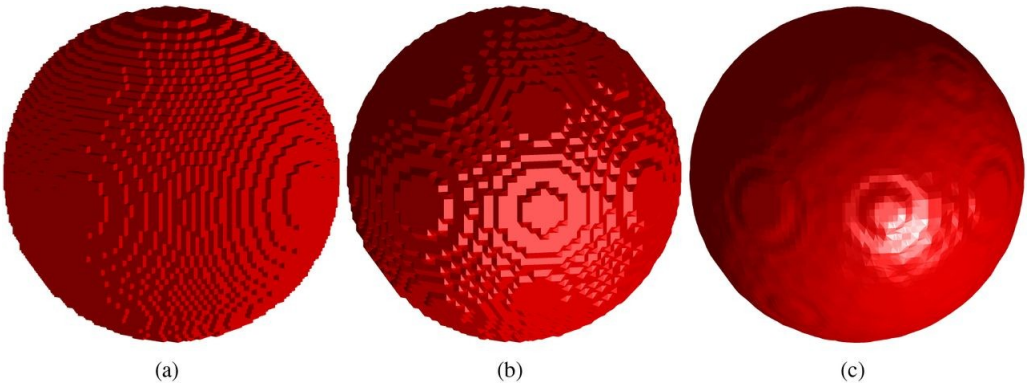

Figure 2: (a) voxel grid. (b) marching cubes. (c) smoothing + marching cubes

However, instead of working on a voxel representation of a shape, the code provided to you uses Signed Distance Function (SDF) representation of the shapes. When passed the coordinates of a point in space, SDF returns the shortest distance between that point and some surface. The transformation of the shape is also done by using the SDF representation. You won't need to modify the shape given to you.
You can find additional details in the sources given below (clickable)

- Video animation of how a mesh is produced
- Reading with some code descriptions
- Presentation with multiple example images

- Information on SDF (you don't need the ray marching part)
- Sample SDF functions and transformations

Transformation

The transformation that you are going to use takes the SDF of a shape and twists it by a specific amount of degrees defined by a variable *twist*. The program generates a number of frames specified by a variable *frameNum*. In each subsequent frame *twist* grows until it reaches *maxTwist*, resulting in *frameNum* objects generated. An example with 5 such frames is shown on Figure 3. Note that increasing the number of frames would generate more objects that represent the intermediate steps, instead of further twisting it.

Figure 3: Transforming (or twisting) a shape shown on 5 frames

Serial Code

We are providing you with a working serial program that implements the Marching Cubes algorithm. The provided code creates a smooth triangle mesh of a shape provided, given the desired resolution.

```
The program may be run from the command prompt as follows:
./mcubes [-n <int>] [-f <int>] [-p <int>] [-b <int>] [-t <int>] [-c] [-o]

With the arguments interpreted as follows:
-n <int> Number of mesh cubes in one dimension (resolution)
-f <int> Number of frames
-p <int> Part to run (-1 for all)
-b <int> Number of thread blocks
-t <int> Number of threads in each block
-c enables testing (comparing against the serial implementation)
   // !!! You do not need to pass it if your shape looks the same !!!
-o enables saving output .obj files

NOTE:
-c option should be DISABLED in ALL the reported experiments
-o option should be enabled in SOME experiments (when mentioned)

You can add additional arguments (such as thread and block count)
```

The workflow of the serial algorithm given to you is the following:

- Starting in the *main* function (from "main.cu" file), read the command line arguments and initialize the global variables.

- Initialize the SDF of the shape and the transformation operation.
- For every frame in range $[0, frameNum)$ run *MarchCubes*.
- For every cube in the mesh, map the vertices to corresponding edges that will form triangles for meshing.
- Save the vertices and normals (needed for a smooth appearance) of the mesh.

Saving:

- If -o option is enabled, save the mesh in .obj format. The code is given in the *IO.h* file.

Other files

- Helper files are there for simpler interfacing with CUDA API. For example, you can use *checkCudaErrors(...)* call with all your CUDA calls to see at least some error reports.

Requirements

To run the program, you will only need an NVCC compiler that uses gcc and comes with CUDA toolkit. On KUACC machines, NVCC is available as a loadable CUDA module.

```
To compile , type
       module load gcc/9.3.0
       module load cuda/11.8.0
       nvcc main.cu -o mcubes
       // in case there is an issue with the cudart library try
       nvcc main.cu -o mcubes -lcudart
Example run:
       ./mcubes -n 100 -f 20 -p -1 -o
```

Output files

The output uses wavefront .obj file format. The easiest way to view the output (.obj) files is to use online tools, like 3dviewer.net or creators3d.com. In case you want to see the files from the KUACC cluster machine, there are 2 ways to do so: 1) to *scp* the files to your machine first, then use any viewer you want. 2) ssh to the cluster with -X option (it will enable opening a window with a graphical interface) and use tools like obj-viewer.

Assignment

Part 1: Single thread with CUDA

In this part of the assignment you will have to use the provided code and run it on the GPU, making sure you are properly allocating the data and making CUDA API calls. The code for this part will automatically get full points if you implement Part 2. For this part you need to provide the performance comparison of running a single GPU thread and a single CPU thread. Explain why are you seeing these results in your report.

Part 2. Parallel CUDA execution

In this part of the assignment, you will have to split the individual cubes between the threads and blocks. For that, you need to properly calculate the indices of the cubes that belong to each thread and make sure you are not overloading GPU shared memory. Make sure the state of the device memory before each kernel launch is all zeroes and the result is copied to the host afterwards. Allocate just enough global memory for a single frame. Experiment with different block and thread counts and report your performance scaling results with -c option and -o options *disabled* and answer the following questions in the report:

- Have you observed any issues with high thread counts while working on this part? If so, why do you think it happened?
- What is the thread and block count that provides the best performance?
- How does the memcopy overhead scale with increasing problem size? How about kernel overheads? Compare the two, try to find the setting where they are equal.

Part 3. Inter-Frame + Intra-Frame Parallelization

In Part 2 you have implemented a so-called discrete kernel approach, with a separate kernel launch per iteration (frame). Now, try to allocate more memory and calculate the result for each frame in a single kernel launch. This approach with the time loop moved inside the kernel is called persistent kernel. Make sure you are using all the available threads. Again, you need to disable correctness tests, run the experiments and answer these questions:

- Have the kernel execution overheads changed? How about memory copies?
- How does the kernel overhead compare to Part 2 with high number of frames and small problem size? What if the problem size is large and the number of frames is small? Explain

Part 4. Double buffer

In this part you will try to improve the overall running time of a specific problem you have found in Part 2, while keeping our memory usage at minimum. Assuming that you have found the problem size and block/thread count which results in approximately comparable overheads of kernel execution and data movement, try to overlap the two. While with a large buffer allocated for Part 3 we can overlap memory copies without concern, in this task you need to use enough memory for only 2 frames. This way, you can send the data for frame *i-1* while computing the frame *i*. You might want to use CUDA Events and Streams in this task. Report the obtained results and discuss the following topics:

- What is the obtained speedup compared to Part 2?
- What is the observed memory usage compared to previous approaches?

Experiments

You are going to conduct an experimental performance study on KUACC. The aim of this experimental study is to get performance data across different numbers of threads with different mesh sizes (resolution) and number of frames. In these experiments, **you will disable the testing features once you confirm the shape is correct** and report the observed numbers. Keep in mind that you are need to make sure that kernel execution timing is done properly in Parts 1-3 by doing a cudaDeviceSynchronize() call after the kernel launch and before cudaMemcpy calls. Since in Part 4 requires overlap between the kernel and memcpy, only the overall running time will be collected.

- The first experiment will be a strong scaling study such that you will run your parallelized code multiple times under different thread counts and a single thread block. Please use the command line arguments below for this study.

```
bash$ ./mcubes -n 256 -f 1 -b 1 -t <threadsNum>
bash$ ./mcubes -n 64 -f 64 -b 1 -t <threadsNum>
bash$ ./mcubes -n 8 -f 32768 -b 1 -t <threadsNum>
```

  <threadsNum>will be 1, 4, 16, 32, 64, 128, 256, 512, 1024. Plot the speedup and execution time figures as a function of thread count and include them in your report. Include the CPU timing as well.

- In the second experiment, you will evaluate your code's performance under different block counts, but with fixed thread count. Use the thread count that worked best for you.

```
bash$ ./mcubes -n 256 -f 1 -b <blockNum> -t <bestThreadNum>
bash$ ./mcubes -n 8 -f 32768 -b <blockNum> -t <bestThreadNum>
```

  <blockNum>will be 1, 10, 20, 40, 80, 160. Plot the speedup and execution time figures as a function of block count.

  Now use the obtained results to answer the questions for each part. Feel free to add more data points to your figures and experiment with varying inputs and parameters.

Important Remarks

In this assignment, you will only have to work on *main.cu*. You don't need to do any changes to other files, although you are not restricted. you are provided correctness test code, but you are not expected to pass it. The important thing is to make sure that you shape actually looks the way it should, with no lacking triangles.
Important! To ensure that all of your performance data is taken from the same machine on the cluster. All performance data for experiments can be submitted as a single job. To

collect data for both gcc and clang compilers in one job you can first compile them separately and provide separate path to the executables.

More details on how to run your code in KUACC cluster can be found in Section Environment below.

Environment

Even if you develop and test your implementations on your local machine or on the computer labs on campus, you must collect performance data on the KUACC cluster.

- Accessing KUACC outside of campus requires VPN. You can install VPN through this link: https://my.ku.edu.tr/faydali-linkler/
- A detailed explanation is provided in http://login.kuacc.ku.edu.tr/ to run programs in the KUACC cluster. In this document, we briefly explain it for the Unix-based systems. For other platforms you can refer to the above link.
- In order to log in to the KUACC cluster, you can use ssh (Secure Shell) in a command line as follows: The user name and passwords are the same as your email account.

```
bash$ ssh $<$username$>$@login.kuacc.ku.edu.tr
bash$ ssh dunat@login.kuacc.ku.edu.tr //example
```

- The machine you logged into is called login node or front-end node. **You are not supposed to run jobs in the login node** but only compile them at the login node. The jobs run on the compute nodes by submitting job scripts.
- To run jobs in the cluster, you have to change your directory to your scratch folder and work from there. The path to your scratch folder is

```
bash$ cd  /scratch/users/username/
```

- To submit a job to the cluster, you can create and run a shell script with the following command:

```
bash$ sbatch <scriptname>.sh
```

- To check the status of your currently running job, you can run the following command:

```
bash$ squeue -u <your-user-name>
```

  A sample of the shell script is provided in Blackboard along with the assignment folder. In the website of the KUACC cluster, a lot more details are provided.

- To copy any file from your local machine to the cluster, you can use the scp (secure copy) command on your local machine as follows:

```
scp -r <filename> <username>@login.kuacc.ku.edu.tr:/kuacc/users/<username>/
scp -r src_folder dunat@login.kuacc.ku.edu.tr:/kuacc/users/dunat/  //example
```

-r stands for recursive, so it will copy the src_folder with its subfolders.

- Likewise, in order to copy files from the cluster into the current directory in your local machine, you can use the following command on your local machine:

```
scp -r <username>@login.kuacc.ku.edu.tr:/kuacc/users/<username>/fileToBeCopied  ./
scp -r dunat@login.kuacc.ku.edu.tr:/kuacc/users/dunat/src_code ./  //example
```

- To compile the assignment on the cluster, you will use NVCC. Before using the compilers, you firstly need to load the CUDA module if is is not already loaded as follows:

```
bash$ module avail //shows all available modules in KUACC
bash$ module list   //list currently loaded modules.
bash$ module load gcc/9.3.0  //loads compiler
bash$ module load cuda/11.8  //loads CUDA 11
```

- If you have problems compiling or running jobs on KUACC, first check the website provided by the KU IT. If you cannot find the solution there, you can always post a question on Blackboard.
- Don't leave the experiments on KUACC to the last minutes of the deadline as the machine gets busy time to time. Note that there are many other people on campus using the cluster.

Submission

- Document your work in a well-written report which discusses your findings.
- Your report should present a clear evaluation of the design of your code, including bottlenecks of the implementation, and describe any special coding or tuned parameters.
- We have provided a timer to allow you to measure the running time of your code.
- Submit both the report and source code electronically through blackboard.
- Please create a parent folder named after your username(s). Your parent parent folder should include a report in pdf and a subdirectory for source code. Include all the necessary files to compile your code. Be sure to delete all object and executable files before creating a zip file.
- GOOD LUCK.

Grading

- 10% Part 1 and corresponding results
- 30% Part 2 and corresponding results
- 20% Part 3 and corresponding results
- 20% Part 4 and corresponding results
- 10% Report clarity: quality of figures, answers and discussions. Note that experiment results are part of the corresponding tasks. Do not expect 90% without the report.
- Finally the fastest and correct top three implementations will be acknowledged in the class.

References

https://github.com/Magnus2/MeshReconstruction
https://youtu.be/_xk71YopsA
http://paulbourke.net/geometry/polygonise/
https://bit.ly/3Dynbse
http://jamie-wong.com/2016/07/15/ray-marching-signed-distance-functions/
https://www.iquilezles.org/www/articles/distfunctions/distfunctions.htm
https://www.creators3d.com/online-viewer
https://github.com/justint/obj-viewer
https://3dviewer.net/

Figures

Long, Zhongjie and Nagamune, Kouki. (2015). "A Marching Cubes Algorithm: Application for Three-dimensional Surface Reconstruction Based on Endoscope and Optical Fiber". Information (Japan). 18, 1425-1437.
Shijie Yan and Qianqian Fang. (2020). "Hybrid mesh and voxel based Monte Carlo algorithm for accurate and efficient photon transport modeling in complex bio-tissues". Biomed. Opt. Express 11, 6262-6270.