School of Electronics and Computer Science
Faculty of Engineering, Science and Mathematics
University of Southampton

Joshua England
09/12/10

3D Graphical Output of Voxel Data from Gait Experiments

Project supervisor: John Carter
Second examiner: Dr N.G Green

A project progress report submitted for the award of Computer Science

Contents

1 Project Description

1.1 Aims

Researchers at Southampton University have been analysing the biometric information about how people walk to see if it is possible to identify a person by their Gait. By triangulating data from 12 cameras they are able to produce voxel data to describe in 3D how a person is walking. The aim of this project is to use this data to produce a graphical output as part of the analysis pipeline. This must be done in real time to allow for visual analysis of the data and demonstrations. Ray Tracing will be used to produced the graphical output with processing taking place on the GPU like the rest of the pipeline.

1.2 Technical Challenge

The main technical challenges of this project centre on how to effectively use the GPU for Ray Tracing. Traditionally GPUs have been developed for fast pixel-based rendering for APIs such as OpenGL and DirectX which requires different optimisations than for Ray Tracing. However, in recent years there has been an increase of general purpose GPU programming through APIs such as Nvidia CUDA[1]. This has allowed for the parallelism of the GPU to be utilised for non-traditional tasks such as financial computations and Ray Tracing. Indeed the existing Gait research pipeline uses CUDA and so will this project. Nevertheless, there may need to be some optimisations needed to take account for the different hardware being used.

1.3 Requirements

1.3.1 Functional

1. Read Voxel Data From Experiment

The system must be able to read in the voxel output of the previous stages of the pipeline. This comes in the form of a flat byte array which represents the 3D space where a byte has the values 1 if an object is present in it. The data could already be present on the GPU but could need to be transferred between locations or read in from a file.

2. Allow interaction with the user

This interaction is to allow the user to change the virtual camera and view the virtual scene from different angles or perspectives.

3. Display a Graphical Representation

Using a windowing system display a graphical representation of the scene to the user from a point of view which they can control.

1.3.2 Non-Functional

1. Use Nvidia CUDA

CUDA is used for other parts of the pipeline and offers parallelism of execution and interesting interfaces for display the graphical output. Therefore this new component must be designed to use Nvidia CUDA.

2. Be able to execute on Linux

The rest of the pipeline uses a linux-based cluster of computers, this new component must easily integrate with the existing system.

3. Run in real-time

The aim of this project is to have a real-time graphical representation of the 3D voxel data. Lag is acceptable but it must operate in real/constant time.

4. Minimum frame rate of 30 FPS

To achieve smooth animation an interaction, a minimum frame rate of 30fps must be kept to.

1.4 Scope

This project is an extension to an existing research so no data collection will be needed. The process of generating the voxel data has already been established. Instead this project will produce a 3D graphical output near the end of the pipeline. Because of the existing system, nVidia graphics cards and CUDA technology will be used to produce the image on linux operated computers.

Depending on time constraints possible extensions would produce better quality images or provide optimisations in other parts of the pipeline.

2 Research

2.1 Ray Tracing

Unfortunately most of the existing research did not apply to this problem. Most of the papers talked about Ray Tracing for traditional graphics objects such as polygons or triangles whereas this problem involves individual points in 3D space. Others suggested complex data-structures such as a “sparse voxel octree”[2] or complex methods such as “BVH-based Packet Traversal” [3] whereas the data comes in the form of a flat-array. To use these techniques an extra stage would need to be added to the pipeline to create and update the data-structure. Other papers suggested techniques which were only suited to static scenes or were based on CPU rendering[4] rather than processing on a GPU.

Nevertheless there was more promising research. The main technique for quickly/simply traversing the voxel space came from a paper[5] suggesting a 3D version of Bresenham's line drawing algorithm. This technique has been used in CPU experimentation and is planned to be used in the GPU implementation. Other research also helped with the mathematical background of ray tracing and how to control the virtual camera[6].

2.2 CUDA

GPU programming is a different paradigm to traditional CPU programming. However, the CUDA By Example book[7] gave many helpful examples (including one using ray tracing) which made the task of programming for a GPU seem easier.

2.3 Experimentation

CPU-Based ASCI Art Ray Tracer

The main area for experimentation came from developing a CPU-based ray tracer which printed to stdout to simulate what would be printed on the screen if there was a graphical output. This gave an opportunity to test the fast voxel traversal algorithm[5] and other mathematical or implementation considerations. A screenshot of the output can be seen in Illustration 1 below.

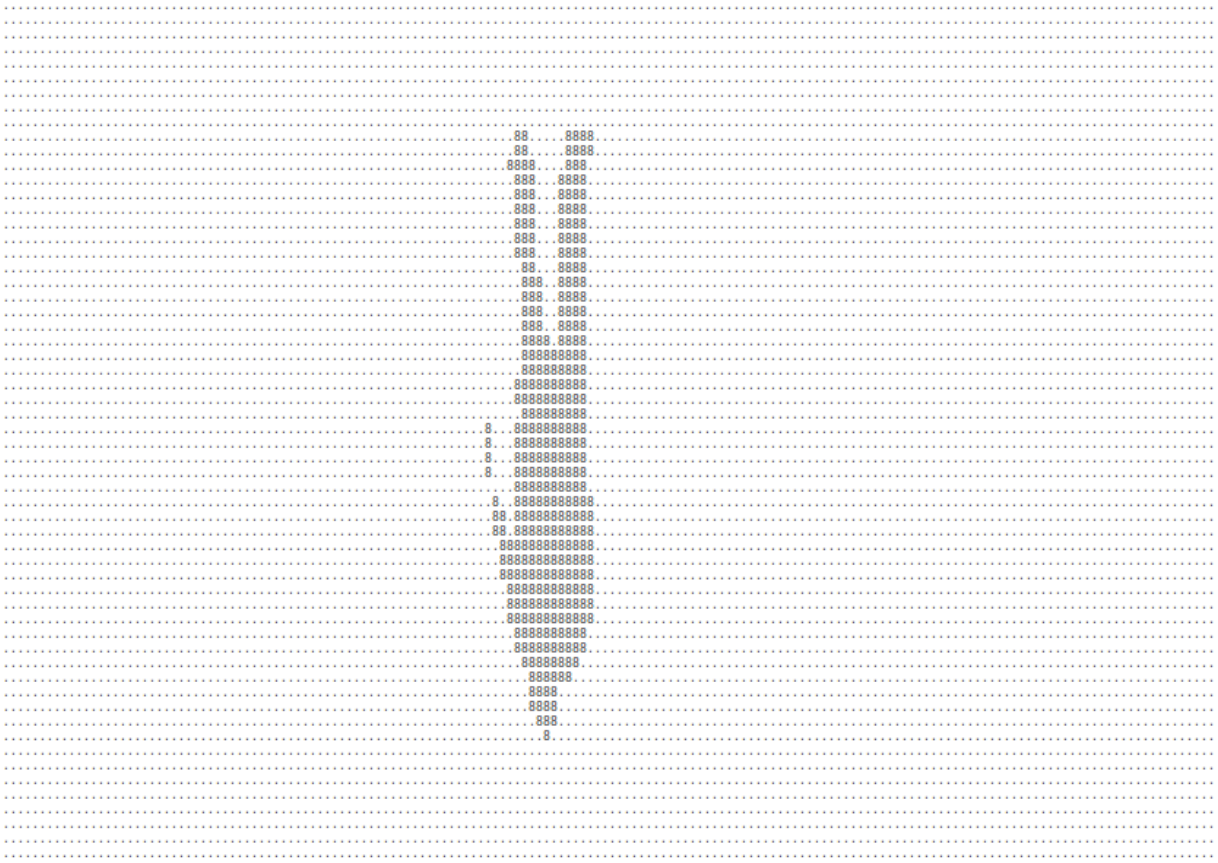

Illustration 1: Screenshot of the output of the ASCII-Art Ray Tracer for the CPU. '.' represents empty voxels, '8' represents the presence of an object.

3 Final Design

3.1 Overview

The key element to using the GPU, rather than the CPU, is the ability to flatten the nested for-loops needed to start the Ray Tracing algorithm for each pixel on the image. Because these tasks can be

started in parallel the colour value for each pixel can be calculated simultaneously rather than sequentially as in a CPU. Other optimisations can also be made using different memory allocations on the GPU and CPU. For example, using constant or texture memory on the GPU can give optimisations for repeatedly access locations. CUDA also allows the programmer to access physical rather than virtual memory controlled by the CPU which could make sending new data to the GPU faster.

The final part of the requirements is to make the system interactive[1], allowing the user the view the scene through a moveable camera. To do this SDL[2] will be used to capture their inputs and using its extensions into OpenGL[3] the final image will be accumulated into a pixel-buffer which processing on the GPU and then shown in the windowing system of the operating system the user is using. An example of using this technique can be found in the CUDA By Example book[8].

Due to the examples from the book and the previous pipeline, the implementation will be programmed using C.

3.2 Flow Diagram

Illustration 2 shows a flow diagram representing how the system will operate.

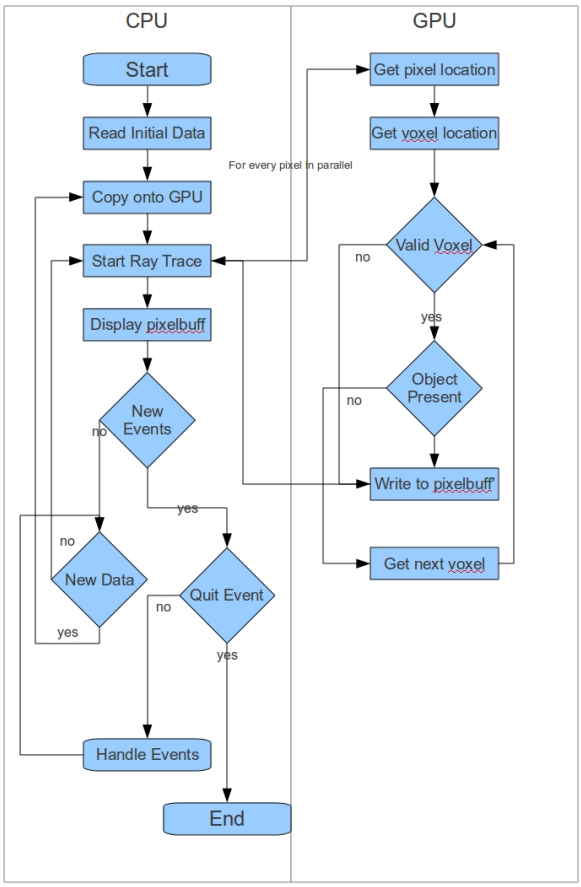

Illustration 2: Flow Diagram for the system

1 1.3.12.Allow interaction with the user
2 The intention is to use SDL version 1.3
3 SDL 1.3 is compatible with OpenGL 3.2 and allows for pixel-buffers

A larger version of this diagram can be found in Appendix B Illustration 6.

3.3 Justification

Just using a CPU to do the computation the time complexity of the ray trace is $\Omega(W \times H)$ where W, H are the width and height of the screen in pixels. By using a GPU to calculate the colour values for the pixels in parallel these outer loops disappear. This means that the ray trace complexity per pixel is at worst is the largest distance across the 3D volume. Since the size of the space is known to be $77 \times 201 \times 367$ voxels the most number of comparisons needed is $\sqrt{77^2 + 201^2 + 367^2}$ meaning the new time complexity for rendering an image is $O(426)$ although time does need to be added on for memory access which could prove to be a bottle neck.

4 Work To Date

4.1 Working Habit

To date Wednesdays have been used to complete project work due to an absence of lectures thus giving a 8 clear hours (working 9:00 – 17:00) to dedicate to project work. Although this is below the recommended 12 hours per week there is plenty of time over the Christmas vacation to catchup on the missed hours. Also, Tuesday mid-day meetings for 30minutes have taken place every week with the project supervisor John Carter.

4.2 Time Break Down

The following sections give more detail to how time was spent relating to the Gantt chart in Appendix A Illustration 4.

4.2.1 Background Reading

The first 3 weeks of the academic year were spent on background reading. This did not include the CUDA By Example book. There was a change of project title and aims between the end of the last summer term and the beginning of the new academic year meaning that no relevant research could take place over the summer months. However, a work placement helped to sharpen programming and problem solving skills.

4.2.2 CPU Ray Tracer

The development of the CPU Ray Tracer took a further 4 weeks. See Illustration 1 for a screenshot of the output.

4.2.3 CUDA Research

Extra time was spent reading and experimenting with Nvidia CUDA. Further experimentation will continue in the Christmas vacation.

4.2.4 Progress Report

The last few weeks of autumn term were spent writing this document and developing the final design for this document. Much of this work can be reused in the final project report.

4.3 Gantt Chart

The Gantt chart for the actual time spend until 15/12/2010 can be found in Appendix A Illustration 4. Compared to the initial chart in Illustration 3 (found in Appendix A) it can be seen that the experimentation has taken the form of an ASCII Art Ray Tracer which has also replaced the CPU development steps for future work. This is because there is little value in developing a full CPU solution since the only difference between this and the GPU equivalent is generation method. Therefore the introduction of SDL (for user input) and OpenGL (for display) will be developed at the same time as the GPU implementation. Another change is an extra week where CUDA research took place.

5 Future Work

5.1 Time-line

The following sections show a rough break down of the time-line for finishing the project. This relates to the predicted time spend Gantt chart in Appendix A Illustration 5.

5.1.1 CUDA Experimentation

Due to the major differences between programming for a CPU and a GPU, the next step is to experiment and practice with the CUDA API to gain confidence.

5.1.2 GPU Solution Development

As part of the experimentation the final GPU solution can begin to be developed. This includes the integration of OpenGL and SDL so that the final image can be displayed and the user can take control of the virtual camera in the scene.

5.1.3 Final Project Report

After the GPU solution has been developed the next focus is documentation as this is very important in the project.

5.1.4 Integration

Previous student projects which added features to the existing pipeline have often struggled to successfully integrate with the existing pipeline. The avoid this affecting other key parts of the project this has been delayed until after the final documentation and before the final demonstration.

5.2 Gantt Chart

The Gantt chart for the future time spend can be seen in Appendix A Illustration 5. This gives more time to the GPU implementation than for the initial Gantt chart (Illustration 3). This is to take account for the reduced CPU experiments and implementations as mentioned previously[4].

4 4.2.2 CPU Ray Tracer

References

[1] Anon., What is CUDA, 2010, http://www.nvidia.com/object/what_is_cuda_new.html
[2] R.Shrout, PC Perspective - John Carmack on id Tech 6, Ray Tracing, Consoles, Physics and more, 2010, http://www.pcper.com/article.php?aid=532
[3] J.Gunther et al, Realtime Ray Tracing on GPU with BVH-based Packet Traversal, 2007
[4] I.Wald and P.Slusallek, State of the Art in Interactive Ray Tracing, 2001
[5] J.Amanatides and A.Woo, A Fast Voxel Traversal Algorithm for Ray Tracing,
[6] S.Borman, Raytracing and the camera matrix – a connection, 2003
[7] E.Kandrot and J.Sanders, CUDA by Example: An Introduction to General-Purpose GPU Programming, 2010
[8] E.Kandrot and J.Sanders, CUDA by Example: An Introduction to General-Purpose GPU Programming, 2010,8,140-147

Appendix A

Gantt Charts

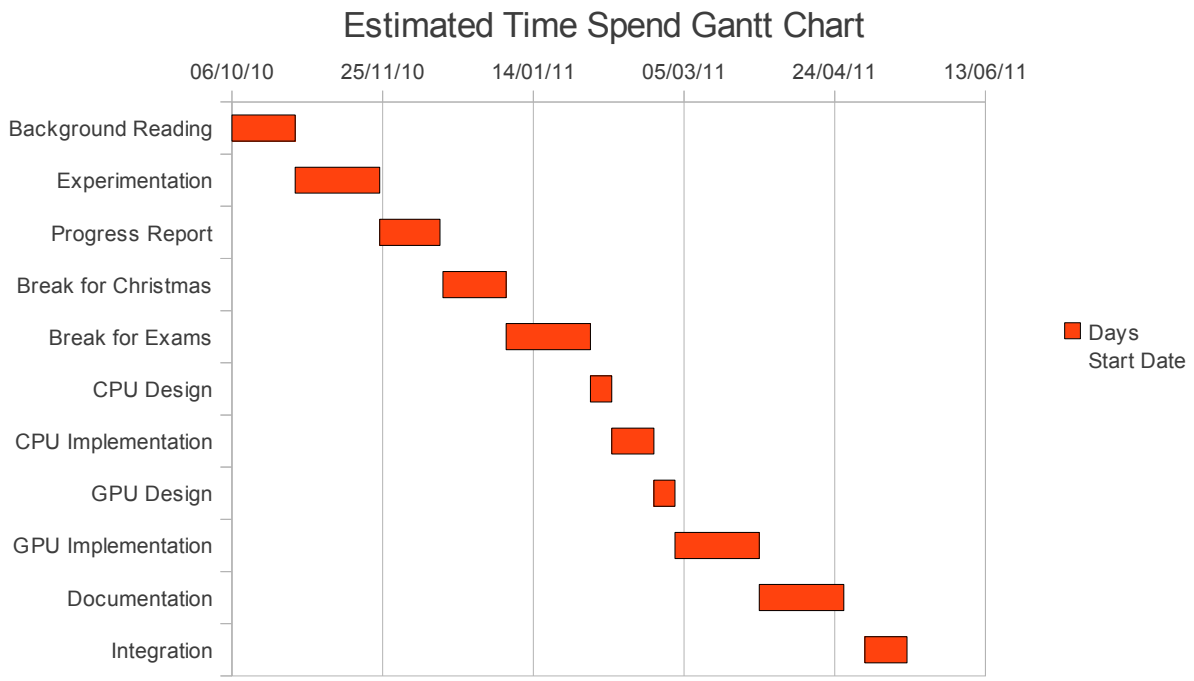

Illustration 3: Estimated time spend at start of project

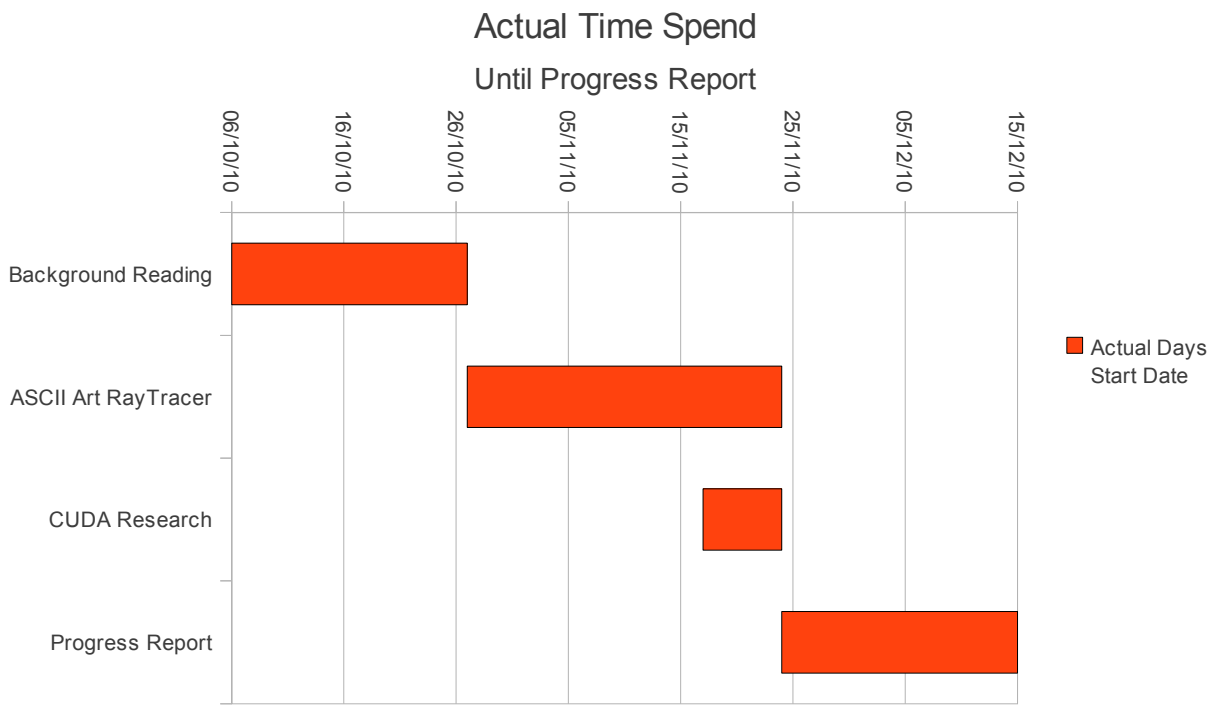

Illustration 4: Actual time spend until 15/12/2010

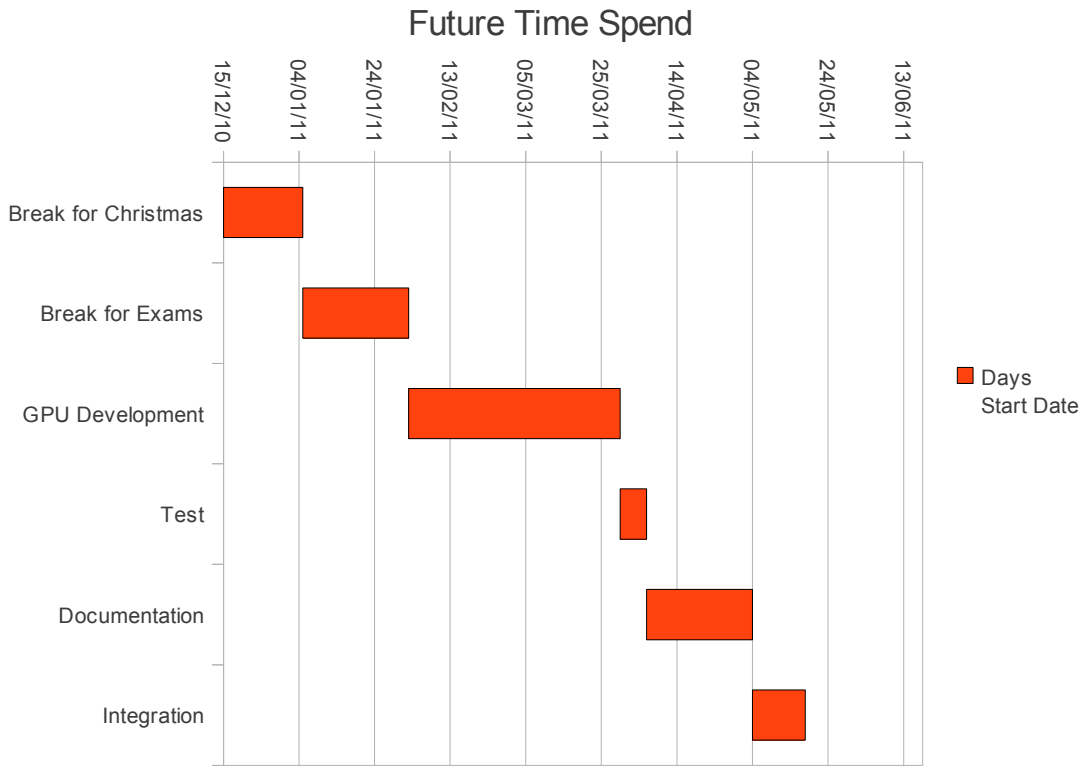

Illustration 5: Future time spend

Appendix B

Design diagrams

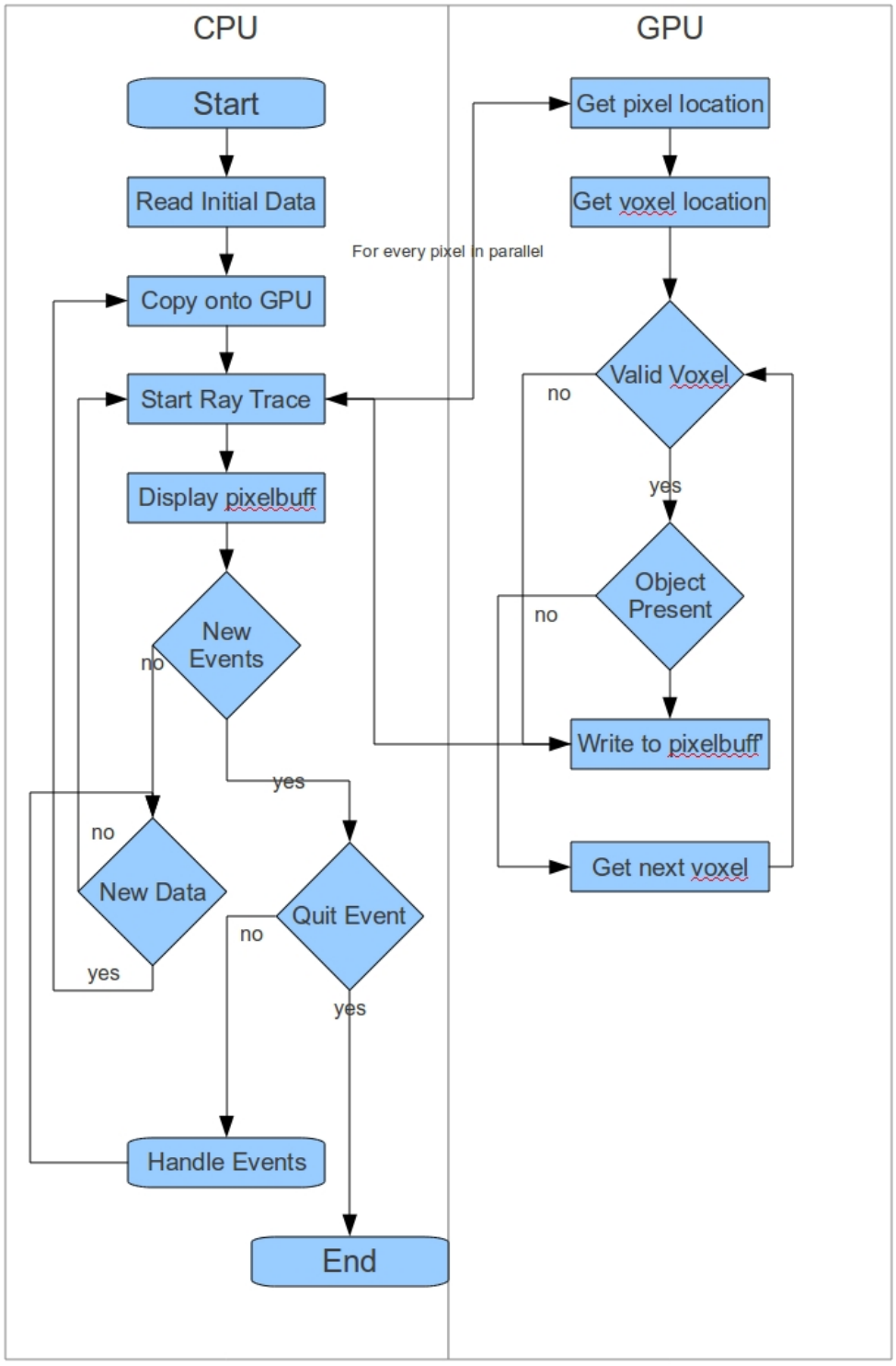

Illustration 6: Flow diagram of the system