GPGPU Acceleration of Memworld, a Raycasting Engine for Direct-Memory Representation of 3D Spaces

DESIGN DOCUMENT

Team 18
Client: Dr. Wymore
Adviser: Dr. Wymore

William Blanchard	Parallelization Lead
Mason DeClercq	Team Lead
Jay Edwards	Documentation Lead
Cristofer Medina Lopez	Integration Lead
Dalton Rederick	Communications Lead
Collin Reeves	Game Development Lead

sddec22-18@iastate.edu
https://sddec22-18.sd.ece.iastate.edu

Revised: 5/1/2022 : Version 3

Executive Summary

Development Standards & Practices Used

730-2014 - IEEE Standard for Software Quality Assurance Processes | IEEE Standard | IEEE Xplore

- We will use the IEEE standard for software quality assurance processes because we are making a software product that needs testing . This standard talks about how we should make a software quality assurance process in order to confirm that our product meets the established requirements that were given.

1219-1998 - IEEE Standard for Software Maintenance | IEEE Standard | IEEE Xplore

- We will adhere to this standard in how we will handle maintenance of Memworld. This standard shows the groundwork for testing and adhering to client given requirements for software that is undergoing development. The standard goes into detail about having a control group of requirements that needs to be maintained as development on the project continues.

24748-5-2017 - ISO/IEC/IEEE International Standard - Systems and Software Engineering--Life Cycle Management--Part 5: Software Development Planning | IEEE Standard | IEEE Xplore

- This standard talks about systems and software engineering with life cycle management. We would be focusing on the software development planning section of this standard. This talks about using peer reviews and testing during development as well as coding standards in order to have a clear understanding that our project is working and how it works.

Summary of Requirements

Requirements/specifications:

- Must use "direct-memory" representation and rendering
- Must parallelize rendering algorithm using GPGPU
- Preferably use a portable GPGPU framework, both in terms of hardware and OS
- Must characterize performance in terms of FPS and determine speedup from GPGPU parallelization
- Performance target is 30 FPS or higher with the following settings:
    - 1024 x 768 resolution or higher
    - Voxel density of 5 or higher
    - Max draw distance of 100 voxels or higher

Test application requirements:

- Must use parallelized engine
- Should be portable
- Should highlight advantages/strengths of engine
- Visually pleasing UI
- May incorporate major new features such as physics, destructible environment, lighting, etc.

Applicable Courses from Iowa State University Curriculum

- COM S 228
- COM S 252
- COM S 309
- COM S 319
- COM S 327
- COM S 336
- CPR E 308
- CPR E 388
- CPR E 557
- E E 285

New Skills/Knowledge acquired that was not taught in courses

- OpenCL
- OpenGL
- Ray casting techniques

Table of Contents

List of figures/tables/symbols/definitions

Definitions:

GPGPU - General-purpose computing on graphics processing units, uses the GPU to perform computation normally done by the CPU.

Voxel - A basic unit of a three-dimensional digital representation of an image or object.

Parallelization - A program or system that processes data in parallel threads.

Octree - A tree data structure where each internal node has eight children.

1 Team

1.1 Team Members

William Blanchard

Mason DeClercq

Jay Edwards

Cristofer Medina Lopez

Dalton Rederick

Collin Reeves

1.2 Required Skill Sets for Your Project

- Know C/C++
    - The program shall be written in C.
- Know how to utilize Git
    - The project shall be managed using Git.
- Experience in game development
    - A test application shall be made in order to demonstrate the use of a "direct-memory" representation of rendering.
- Know how to set up a development environment
    - The developers shall use their own computer to run the application.
- Know how to use a GPGPU framework
    - The developers shall use a GPGPU framework that is portable to parallelize the program.

1.3 Skill Sets covered by the Team

- Know C/C+
    - Wil (Career experience)
    - Mason (Career experience)
    - Jay (Course work experience)
    - Cristofer (Course work experience)
    - Dalton (Some experience)
    - Collin (Course work experience)
- Know how to utilize Git
    - Wil (Career experience)
    - Mason (Career experience)
    - Jay (Career and course experience)
    - Cristofer (Course and project experience)
    - Dalton (Experience from labs in other courses)
    - Collin (Career experience)
- Experience in game development
    - Wil (Semi-career experience)
    - Mason (Personal projects)
    - Jay (Minor personal projects)
    - Dalton (Minor experience in game app design)
    - Collin (Minor personal projects)
- Set up development environment
    - Wil (Minor experience)
    - Mason (Career experience)
    - Cristofer (Minor experience)
    - Collin (Career experience)

- Know how to use a GPGPU
    - Wil (CUDA)
    - Mason (OpenCL experience)

1.4 Project Management Style Adopted by the team

The team will use a mix of Agile and Scrum. We will use Agile for setting milestones and optional objectives. We will use Scrum for regular meetings with each other.

1.5 Initial Project Management Roles

William Blanchard - Parallelization Lead

Mason DeClercq - Team Lead

Jay Edwards - Documentation Lead

Cristofer Medina Lopez - Integration Lead

Dalton Rederick - Communications Lead

Collin Reeves - Game Development Lead

2 Introduction

2.1 Problem Statement

The goal of our Senior design project is to update and improve the direct memory rendering model created by Dr. Wymore, including features by his request or to our interest. These features include, but are not limited to, increased resolution, improved frame rate or run speed, and inclusion of miscellaneous object interactions such as physics and collision. At the end of the project, the team will create a simple game to show off these improvements and additional features.

2.2 Requirements & Constraints

Final deliverables are:

1. GPGPU-parallelized Memworld engine
2. Memworld test/demonstration application

Requirements/specifications:

1. Must use "direct-memory" representation and rendering (**constraint**)
2. Must parallelize rendering algorithm using GPGPU (**constraint**)
3. Preferably use a portable GPGPU framework, both in terms of hardware and OS (**constraint**)
4. Must characterize performance in terms of FPS and determine speedup from GPGPU parallelization (**constraint**)
5. Performance target is 30 FPS or higher with the following settings:
6. 1024 x 768 resolution or higher (**constraint**)
7. Voxel density of 5 or higher (**constraint**)
8. Max draw distance of 100 voxels or higher (**constraint**)

Test application requirements:

1. Must use parallelized engine (**constraint**)
2. Should be portable (**constraint**)
3. Should highlight advantages/strengths of engine
4. Visually pleasing UI
5. May incorporate major new features such as physics, destructible environment, lighting, etc.

2.3 Engineering Standards

730-2014 - IEEE Standard for Software Quality Assurance Processes | IEEE Standard | IEEE Xplore

- We will use the IEEE standard for software quality assurance processes because we are making a software product that needs testing . This standard talks about how we should make a software quality assurance process in order to confirm that our product meets the established requirements that were given.

1219-1998 - IEEE Standard for Software Maintenance | IEEE Standard | IEEE Xplore

- We will adhere to this standard in how we will handle maintenance of Memworld. This standard shows the groundwork for testing and adhering to client given requirements for software that is undergoing development. The standard goes into detail about having a control group of requirements that needs to be maintained as development on the project continues.

24748-5-2017 - ISO/IEC/IEEE International Standard - Systems and Software Engineering--Life Cycle Management--Part 5: Software Development Planning | IEEE Standard | IEEE Xplore

- This standard talks about systems and software engineering with life cycle management. We would be focusing on the software development planning section of this standard. This talks about using peer reviews and testing during development as well as coding standards in order to have a clear understanding that our project is working and how it works.

2.4 Intended Users and Uses

The main beneficiaries for this project would be Game Developers, specifically those that fit the niche of wanting a voxel-based game engine with a focus on performance. They will use it as a Game Developer would use any other game engine, as a tool to develop games. The difference with Memworld is that it will give a direct memory representation of what it is rendering.

- Use Case 1: Render an object/world
- Use Case 2: Physics simulation on voxels
- Use Case 3: Create a visual representation of current memory storage

3 Project Plan

3.1 Project Management/Tracking Procedures

We are using an agile management style. This will help us with our project because we will be making an application that shows off our rendering engine. We will have to keep iterating over our application, designing, developing, and testing it until we meet the requirements of the project. We are able to do this every week with the tasks that we assign ourselves.

We will be using GitLab in order to track our progress throughout the semester. This allows us to assign each other tasks and we can comment/view the work on those tasks in order to keep organized.

3.2 Task Decomposition

- Improve speed of rendering engine
    - Implement GPGPU framework
        - OpenCL research
        - OpenCL/GL interoperation
        - Implementation
    - Octree Implementation of world
        - Octree research
        - Implementation
    - Octree Traversal Algorithm
        - Tile traversal algorithm (DDA) research
        - Implementation
- Create a test application to show off our rendering engine
    - Object importing
        - Determine software that would be easy to create voxel objects with
        - Learn how data is stored in binary
        - Implementation
    - Object movement
        - Object data structure implementation/research
    - Physics
        - Object data structure implementation (co-task)
        - Physics equations / type of physics that we would need research
    - Lighting
        - Research different lighting techniques
        - Implement lighting into OpenCL render kernel
    - GUI
        - Work on implementing a GUI for our application so that it is user friendly

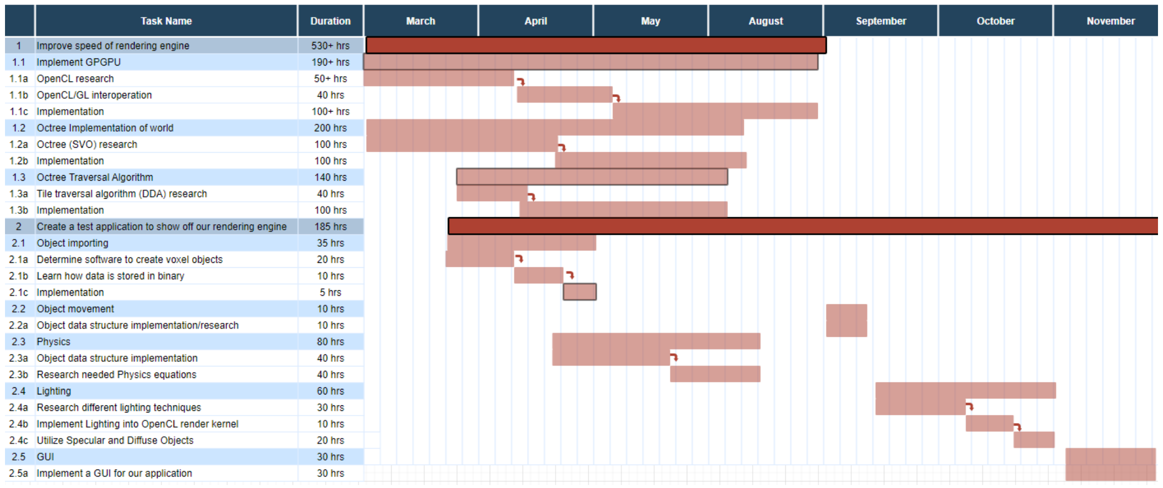
3.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

Milestones:

- Performance:
    - 30 FPS or higher with the following settings
        - Voxel density of 5 or higher
        - 1024 x 768 resolution or higher
        - Render distance of 100 voxels or higher
- Physics (all require under 2 fps slowdown from current implementation):
    - Objects can fall
    - Objects can collide and stop movement
- Simple rigid body physics
    - Lighting:
    - Ray trace from object to light source (1 bounce from object)
    - 2 object reflection patterns (Specular and diffuse object options)
- Object importing:
    - Can import a world of 256 x 256 x 256
    - Can import individual objects up to 64 different objects
- Portability:
    - 2 platforms (Windows and Mac)

3.4 Project Timeline/Schedule

Project Timeline



	Task Name	Duration
1	Improve speed of rendering engine	530+ hrs
1.1	Implement GPGPU	190+ hrs
1.1a	OpenCL research	50+ hrs
1.1b	OpenCL/GL interoperation	40 hrs
1.1c	Implementation	100+ hrs
1.2	Octree Implementation of world	200 hrs
1.2a	Octree (SVO) research	100 hrs
1.2b	Implementation	100 hrs
1.3	Octree Traversal Algorithm	140 hrs
1.3a	Tile traversal algorithm (DDA) research	40 hrs
1.3b	Implementation	100 hrs
2	Create a test application to show off our rendering engine	185 hrs
2.1	Object importing	35 hrs
2.1a	Determine software to create voxel objects	20 hrs
2.1b	Learn how data is stored in binary	10 hrs
2.1c	Implementation	5 hrs
2.2	Object movement	10 hrs
2.2a	Object data structure implementation/research	10 hrs
2.3	Physics	80 hrs
2.3a	Object data structure implementation	40 hrs
2.3b	Research needed Physics equations	40 hrs
2.4	Lighting	60 hrs
2.4a	Research different lighting techniques	30 hrs
2.4b	Implement Lighting into OpenCL render kernel	10 hrs
2.4c	Utilize Specular and Diffuse Objects	20 hrs
2.5	GUI	30 hrs
2.5a	Implement a GUI for our application	30 hrs

3.5 Risks And Risk Management/Mitigation

Agile projects can associate risks and risk mitigation with each sprint.

- Improve speed of rendering engine
  - Implement GPGPU framework
    - OpenCL research (risk = 0.1)
    - OpenCL/GL interoperation (risk = 0.4)
    - OpenCL/GL Implementation (risk = 0.2)

  - Octree Implementation of world
    - Octree research (risk = 0.3)
    - Octree Implementation (risk = 0.3)

  - Octree Traversal Algorithm
    - Tile traversal algorithm (DDA) research (risk = 0.3)
    - Octree Traversal Algorithm Implementation (risk = 0.6)

    If the implementation doesn't go as expected, there are many other different types of traversal algorithms for tile based worlds. There is also a website called shadertoy.com that has a lot of implementations for the algorithm and other algorithms.

- Create a test application to show off our rendering engine
  - Object importing
    - Determine software that would be easy to create voxel objects with (risk = 0.1)
    - Learn how data is stored in binary (risk = 0.1)
    - Implementation (risk = 0.3)
  - Object movement
    - Object data structure implementation/research (risk = 0.4)
  - Physics
    - Object data structure implementation (co-task) (risk = 0.4)
    - Physics equations / type of physics that we would need research (risk = 0.6)

    If implementation doesn't go well, we can look into using external libraries/functions for specific areas of our physics implementation.

  - Lighting
    - Research different lighting techniques (risk = 0.1)
    - Implement lighting into OpenCL render kernel (risk = 0.7)
      - Utilize specular and diffuse objects (risk = 0.7)

    Lighting is known to heavily decrease performance when implemented, so there is a chance that it could affect the model's ability to hit our performance requirements. Some alternatives are static lighting, which doesn't update as the program runs, or getting rid of lighting entirely, as it isn't a requirement for completion of the project.

- GUI
    - Work on implementing a GUI for our application so that it is user friendly (risk = 0.2)

3.6 Personnel Effort Requirements

Task	Estimate	Explanation
OpenCL Research	50+ hours	OpenCL will need ongoing research as we continue to optimize the algorithm and further learn what OpenCL is capable of doing.
OpenCL/GL Interoperation	40 hours	OpenCL and OpenGL have the capability of sharing objects on the GPU, but it is possible that it won't work on all platforms, so we will have to look into how this works in further detail.
OpenCL/GL Implementation	100+ hours	Work with OpenCL and OpenGL will require constant updates and debugging to implement new design features.
Octree Research	100 hours	Octrees are a complicated data structure that can be hard to understand/implement. There aren't too many good resources on it considering there are limited uses for it.
Octree Implementation	80 hours	There are very few resources to base our implementation off of, and given that those who have implemented it already are mainly researchers, their implementation far exceeds the needs for our project. We will have to find a way to simplify their algorithms.
Tile Traversal Algorithm (DDA) research	40 hours	There are quite a few resources for this topic, but most of them are for 2 dimensions and we need one for 3 dimensions, or we will have to figure out how to convert it to 3 dimensions.
Octree Traversal Algorithm Implementation	100 hours	There are many resources available for this on shadertoy.com, but the implementations are difficult to understand or mostly not needed for our project. We will have to parse through the implementations or create our own by using what we learned from the DDA research.
Determine software that would be easy to create voxel objects with	20 hours	Going through the installation and testing process for each program to ensure it works for our uses.
Learn how data is stored in binary	10 hours	Understanding how our memworld's data is being used and perhaps reading documentation for the tools that we are using.
Implementation	5 hours	Once we understand how the data is stored, getting the data and transferring it to our application shouldn't take too much work since the data stored can be directly translated to our application.

Object data structure implementation/research (movement)	10 hours	Initial implementation of data structure movement will be relatively simple given the world coordinate system is inherent in the raycaster's design. Extra time is given for future development and bug fixing of quirky behavior.
Object data structure implementation (co-task) (physics)	40 hours	The design of objects and the implementation of physics into those objects will take nearly as much time as the research for this particular concept.
Physics equations / type of physics that we would need research	40 hours	Many concepts are involved in physics, including dynamic velocity, collisions, etc. This would mean there will be an extensive amount of time dedicated to finding related texts.
Research different lighting techniques	30 hours	There are many different implementations out there for lighting in rendering. We need time to find these implementations and determine which would be the most appropriate to use in the raycasting engine.
Implement lighting into OpenCL render kernel	10 hours	Once the appropriate implementation is found, all we would have to do is effectively convert the pseudocode and math for the implementation into C code. Extra time given for debugging.
Utilize specular and diffuse objects	20 hours	After we can get basic lighting we will implement specular and diffuse objects. These are essentially equations used to make a surface look matte or reflective. Extra time allotted for debugging as this can impact performance heavily
GUI development	30 hours	Doing research and design planning on the implementation of the GUI and how we can maximize functionality while maintaining user friendliness.

3.7 Other Resource Requirements

- Research Time
- Internet access for research
    - Google Scholar
- Access to a computer
    - Windows
    - Mac
    - Linux (if time permits)
- Required programs
    - MinGW
    - GLFW
    - an IDE (member's choice)
    - Cmake
    - Vulcan
    - Any other programs used in the future

4 Design

4.1 Design Context

4.1.1 Broader Context

Public health, safety, and welfare:

- Don't crash the computer when running the program.
- Don't use all of the memory on the computer.

Global, cultural, and social:

- Ensure that the usage and formatting of the program adheres to the standards set by similar programs to ensure usability for the user.

Environmental:

- Make sure the program doesn't take too much processing power to avoid high electricity usage.

Economic:

- Memworld will be using OpenCL, which will allow for execution for different types of chipsets. Memworld will be expected to run on a number of different platforms and devices allowing for a range of different users to use it, whether the device is more economical or is more computationally intensive.

4.1.2 User Needs

Developers need a way to import objects into the scene because it is difficult to make voxel objects by using coordinates in a world.

Developers need a way to move through the scene because they need to add character movement to their game/application.

Developers need a way to add lighting to a scene because it can be difficult to tell the difference between objects without it.

Developers need a way to implement physics on objects because that can add dynamic movement to an otherwise static environment.

Clients need a way to modify settings to the program because they might want to change the FOV or what GPU is being used.

Clients need to have an intuitive UI because they will need to use it without any help.

Clients need a way to know the product is satisfactory to their expectations because it determines the usefulness of the product for their needs (I.E. a demo to test the product).

4.1.3 Prior Work/Solutions

This is a paper from NVIDIA demonstrating how they implemented a sparse voxel octree. This is more complicated than we need ours to be. It is an efficient way to traverse a world to find voxels. We also are restricted by not being able to use a sparse version of the world (i.e. we have to represent empty space as a zero in memory).

Laine, Samuli, and Tero Karras. "Efficient sparse voxel octrees–analysis, extensions, and implementation." *NVIDIA Corporation* 2.6 (2010).

We found a thesis paper done on a similar idea. While they are able to get high fidelity static models working with their design, real time animation appears to be difficult to accomplish with their work. This means that for our purposes, game development, it is likely not a feasible strategy to follow.

Crassin, C., Neyret, F., & Sillion François X. (n.d.). *Gigavoxels: Un pipeline de Rendu Basé Voxel pour l'exploration efficace de scènes larges et détaillées* (thesis).

Here is a 3D voxel engine,

This is Ken Silverman's personal project which is a similar 3D voxel engine. He explains part of his process on the website and in the readme for the project, mainly just having tips for certain parts of the implementation. The issues with this source is that the notes and codes are not outsider friendly. The notes read like personal memos and the code is hard to understand for someone other than the dev.

Silverman, K. (2018). PND3D demo and source code. Retrieved 2022, from http://advsys.net/ken/voxlap/pnd3d.htm

4.1.4 Technical Complexity

1. The raycasting engine consists of advanced concepts involving high-level programming and in-depth math. This includes the use of libraries such as OpenGL, as well as the use of concepts like octrees and general raycasting themes.

2. Raycasting is an existing concept. The purpose of this project is to take the problem of raycasting and optimize it using the physical resources available to the computer. In this way, the scope of our problem, while challenging, does aim to improve upon currently existing standards if it doesn't at least try to find a different way of implementing them.

4.2 Design Exploration

4.2.1 Design Decisions

1. OpenCL integration (GPGPU parallelization framework)
2. Voxel Octree implementation (rendering algorithm)
3. Physics: Object, Movement
4. Lighting implementation

4.2.2 Ideation

Design Decision - GPGPU Parallelization

1. OpenCL
2. Direct-Compute
3. AMD App SDK
4. CUDA
5. Vulkan

We used a lotus blossom to find different options and their benefits and downsides. We found that there were a decent number of options to pick from for the GPGPU frameworks.

4.2.3 Decision-Making and Trade-Off

1. OpenCL
    a. Pros: Works on most devices. Well established support and documentation. Works well with OpenGL
    b. Cons: It is harder to use than some other options
2. Direct-Compute
    a. Pros: Caters toward gaming, Long history of development
    b. Cons: Platform limitation - May not support all desired devices. Uses HLSL style language for computing
3. AMD App SDK
    a. Pros: Targets heterogeneous system architecture
    b. Cons: It was discontinued
4. CUDA
    a. Pros: Fast on Nvidia systems
    b. Cons: Only works on machines that have an Nvidia card
5. Vulkan
    a. Pros: It is easier to use. Works on most devices. Decent documentation
    b. Cons: It is newer, so it is not as stable

Decision Matrix for GPGPU

	Cross Platform Compatibility (0-10)	Stability (0-10)	Existing documentation and support (0-5)	Ease of use (0-5)	Total (0-30)
OpenCL	10	10	5	2	27
Direct-Compute	5	9	5	1	20
AMD App SDK	5	0	0	0	5
CUDA	4	9	5	2	20
Vulkan	10	8	4	4	26

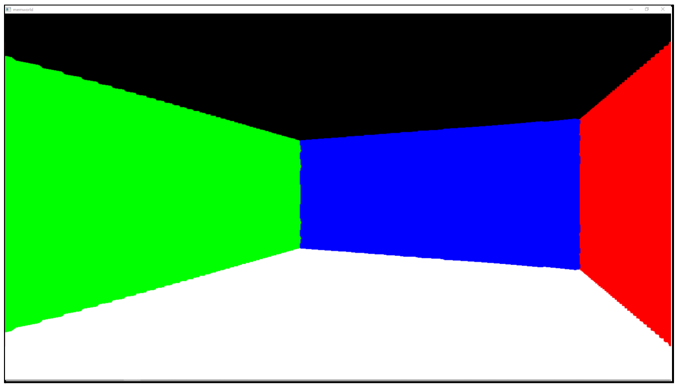
We chose to use OpenCL because it is a cross-platform framework. It has been used for many years and is very stable. There is a lot of documentation on it because it has been around for a long time. The one downside is that it can be more difficult to use because there is a lot of setting up that needs to be done in order to get it to work, but it integrates well with the project that we were given.

4.3 Proposed Design

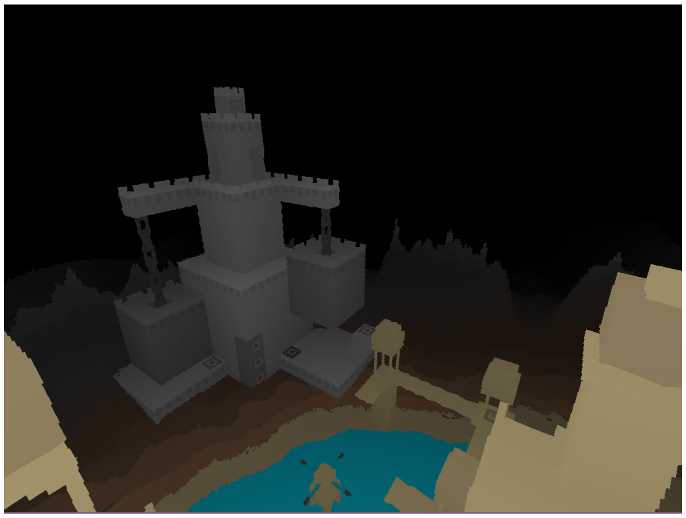
- We decided upon OpenCL, and have fully implemented that into the Memworld engine. This made the engine work much more efficiently, and it is able to render a high voxel-density smoothly.
- We are currently implementing a naive version of a voxel octree traversal algorithm to improve rendering speeds.
- We are currently implementing physics and testing it as well.

4.3.1 Design Visual and Description

Initial Memworld Image



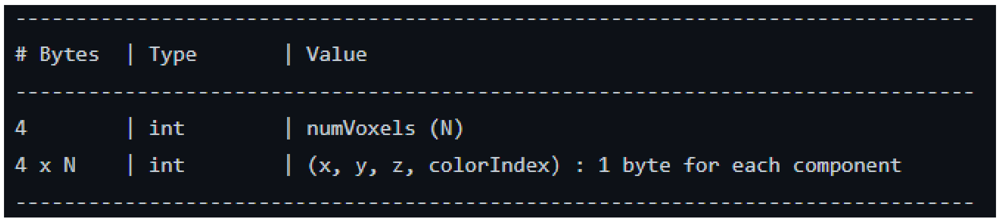
Current Memworld Image



List of Features:

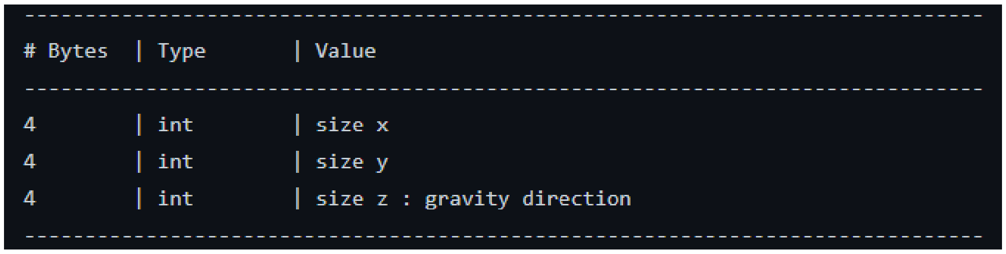
- **File Importing**
    - **Type of Files**
        - The files that our group is importing to our application are .vox files. These are in the format given by MagicaVoxel (see “Voxel Model Github Repository” by Ephtracy in the appendices).
        - They contain color and location information for each voxel in the model along with the total length, width and height of the model.

Memory Representation of .vox Object



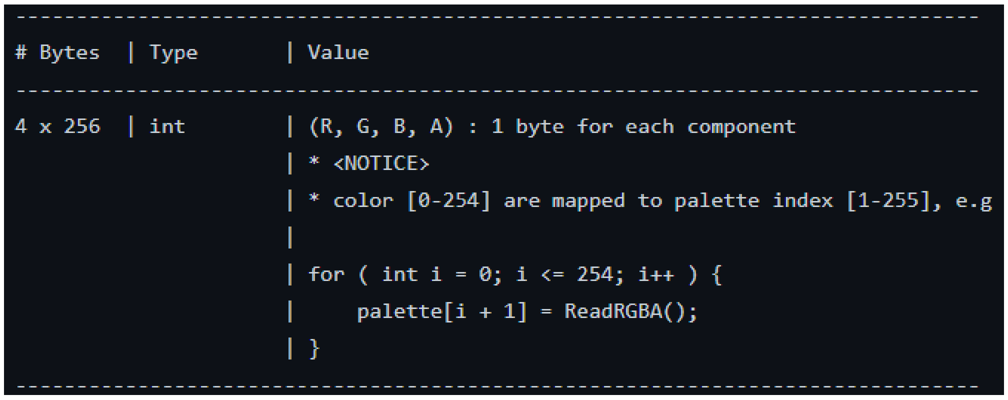
```
-------------------------------------------------------------------------------
# Bytes  | Type       | Value
-------------------------------------------------------------------------------
4        | int        | numVoxels (N)
4 x N    | int        | (x, y, z, colorIndex) : 1 byte for each component
-------------------------------------------------------------------------------
```

Memory Representation of .vox Object Size



```
-------------------------------------------------------------------------------
# Bytes  | Type       | Value
-------------------------------------------------------------------------------
4        | int        | size x
4        | int        | size y
4        | int        | size z : gravity direction
-------------------------------------------------------------------------------
```

Memory Representation of .vox Color Palettes



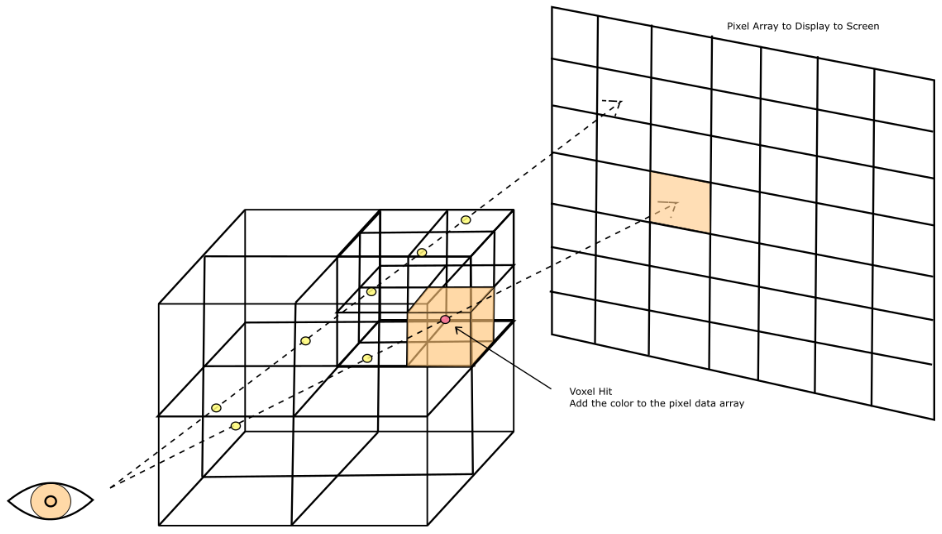
```
-------------------------------------------------------------------------------
# Bytes  | Type       | Value
-------------------------------------------------------------------------------
4 x 256  | int        | (R, G, B, A) : 1 byte for each component
                      | * <NOTICE>
                      | * color [0-254] are mapped to palette index [1-255], e.g
                      |
                      | for ( int i = 0; i <= 254; i++ ) {
                      |     palette[i + 1] = ReadRGBA();
                      | }
-------------------------------------------------------------------------------
```

- **How it Works**
    - Our program parses the .vox file.
    - It finds the amount of voxels in the file, provided towards the top of the .vox file.
    - With the information above, the program finds the RGBA color pallet used by the file.
        - It is located on a non-fixed line in the file.
        - It is found by finding the line R G B A 0 4, located around num_vox + 450 lines into the file.
        - The program reads through and assigns the color palette for the object there.
    - With the information above, the program can read through each voxel.
        - Each voxel provides an X Y Z and color value, with color being the palette color used.
        - The program assigns that color to the provided X Y Z position in the Memworld world array.

- **Renderer**
    - **Voxel Octree**

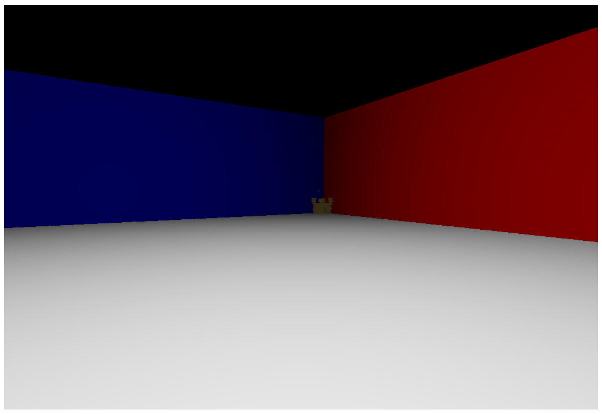
In the diagram listed below, there is a demonstration of how the octree works and how it translates the rays to the output screen. The eye is the camera of the character that moves around the world. Rays are cast from the eye to an imaginary grid in the distance. For each ray, there is a work item that is being run on the GPU. This work item computes the magnitude of the ray and travels along the ray until it reaches a voxel that it hits. There is meta-data that keeps track of whether a voxel is contained within a larger cube. If there isn't a voxel within the larger cube, the ray is traversed until it reaches the next larger cube in order to skip unneeded memory accesses. Once it hits a voxel, the color stored in that position of the world array is transferred to the pixel array to be displayed to the screen.

Octree Ray Casting Diagram

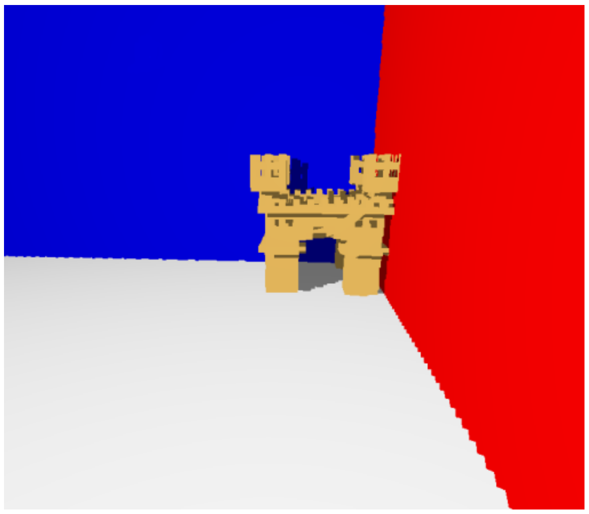

Pixel Array to Display to Screen
Voxel Hit
Add the color to the pixel data array

- **Lighting**
    - **Distance Based**
        - The color of the voxel is darkened based on the distance traveled along the ray. At the time of writing, the color is darkened by reducing the r, g, and b values by distance divided by 512. After 512 is reached, the color put into the pixel array is black. This gives a nice transition to the max view distance and a default color being black without having an abrupt change from a color of a voxel to black.

          Distance Based Lighting

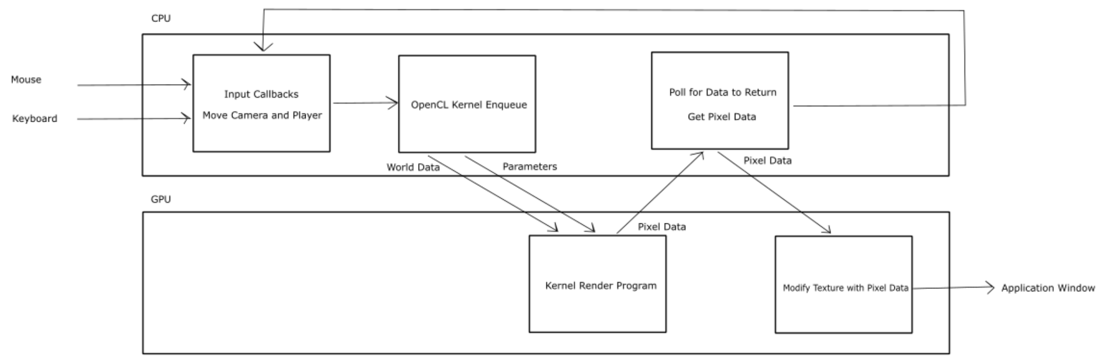
          
    - **Ray Based**
        - This will work the same way as described by the voxel octree section. The only difference is that once a voxel (voxel 1) is reached, another ray is cast towards a point light. This ray is traversed until the light or another voxel (voxel 2) has been reached. If another voxel (voxel 2) has been reached, the color received from the initial voxel (voxel 1) is then darkened and put into the pixel array to create a shading effect. This is done by reducing the r, g, and b values by 50%.

          Ray Based Lighting

- **Physics**
    - The overall idea for physics is to have a specific thread running the physics related functions at a set interval (a few times a second, as an example). All interactable things in the world will be defined as objects, and from there they will have a flag for whether they will be affected by the physics engine. From there, our physics functions will, when called by our thread, simulate one moment of physics interactions. This will include things like objects falling by gravity, objects colliding with one another and being pushed back, etc.
- **OpenCL**
    - The general idea of using OpenCL is to allow us to run multiple tasks at the same time in order to parallelize the work of casting multiple rays across the world. We are using 2 dimensional work items in order to have a task running for each pixel. Using OpenCL, we are able to choose a GPU to use in case there are multiple on the system. We use OpenCL to create a command queue in order to enqueue kernel programs onto the GPU. We are also able to send the world data to the GPU for the kernel programs to use, and we are able to read the pixel array data from the GPU kernel to output to the screen.
    - In the diagram listed below, there is an overarching view of how our code interacts with the computer. The mouse and keyboard are used for inputs to the program. This modifies the user's position and rotation, which are parameters to the renderer kernel. The renderer kernel outputs pixel data and then OpenGL sends the pixel data back to its texture memory on the GPU to render to the application window.

System Interaction Diagram



- **Threads**
    - The demo application utilizes multithreading in order to optimize the program that uses the engine as much as possible. There will be a thread for input, one for physics, and one for rendering the current frame, as well as any others that we feel would benefit the program if they ran in parallel to everything else. These are all tasks that are able to be run in parallel with each other, and would speed up runtime by a considerable amount if parallelized correctly.
    - Any memory access conflicts will be taken care of by mutexes. These will occur in places such as the assignment and reading camera position in the input and frame processing threads respectively. This is to ensure that the security and integrity of the program is not compromised by memory reads/writes.
- **OpenCL/OpenGL interoperability**
    - The purpose of interoperability of these two frameworks is to allow the ability to share data across the program more efficiently. Using OpenGL would rely on the CPU, whereas OpenCL puts more computing power onto the GPU. The integration of OpenCL will help boost performance, however, it adds complexity to the code structure.
    - Interoperability of the two frameworks would allow the OpenCL memory objects to share data with OpenGL data structures which include vertex buffer objects, textures and render buffer objects. This would allow for these OpenGL data structures to be processed as memory objects on a kernel. This can help with performance by reducing the effort required to send data back and forth between the CPU and GPU.
- **Settings File**
    - The settings file is a way for a user to change how they want the engine to render, specifically in regards to the world size, window size, draw distance, and voxel density. Changing these values can speed up or slow down frame rate depending on whether you increase or decrease the values. Meaning that a user with a slower computer can change the settings and get a better performing render at the expense of size/clarity. A user with a faster computer will be able to change the settings to get a higher quality render.

Memworld Test Application Design:

Gameplay Overview:

- The type of test application that we are planning to make will be a 3D platformer.
- There will be 4 different levels that increase in size as you progress through the levels up to a max size of 256 x 256 x 256.
- The overarching goal in these levels will be to acquire various collectables as well as permanent power ups that allow you to go back to previous levels and get more collectables.
- There will be an area that will be the hub in which you will be able to choose which level you want to enter.
- The user will be able to press the escape key to open up a settings menu and also be able to choose if they want to go back to the hub area.
- The application will attempt to utilize the full range of our 256/256/256 voxel worlds by incorporating verticality in the level design.

- The application will provide hazards or failure states in the level design (lava, pits, enemies, etc…) to provide some kind of challenge.
- The application will put a major focus on displaying the strengths of the engine by highlighting working features (pillars/windows to show off shadows, falling objects/moving objects for physics, etc).

4.3.2 Functionality

The design is intended to operate in the context of more complete programs that require the use of a 3D voxel-based viewport in their end products. This includes applications such as video games and environment analysis. In addition, because of the nature of the design, the product could potentially be used to visually represent what is stored in memory at any given moment, which would be of use to people such as data scientists.

The current design satisfies most of the requirements given to us by our client, including the utilization of the computer's memory to store the world data and the implementation of a ray casting algorithm to use that data to render the world.

We have not yet implemented a demo application for our design. As an additional requirement, we were told to create an application that demonstrates the capabilities of the engine. This is currently in production.

4.3.3 Areas of Concern and Development

Concerns:

- New features added for the demo could harm performance.
- Performance of the application on different platforms and making sure computation of the application is predictable across platforms.

Development solutions:

- We need to further optimize the program so that it runs faster on lower end hardware.
- We need to develop new features with performance in mind so that they don't cause issues.
- We need to be cognizant of the stability of the application on other platforms when making changes.

4.4 Technology Considerations

GPGPU tools:

1. OpenCL
    a. Pros: Works on most devices. Well established support and documentation. Works well with OpenGL
    b. Cons: It is harder to use than some other options
2. Direct-Compute
    a. Pros: Caters toward gaming, Long history of development
    b. Cons: Platform limitation - May not support all desired devices. Uses HLSL style language for computing
3. AMD App SDK
    a. Pros: Targets heterogeneous system architecture
    b. Cons: It was discontinued
4. CUDA
    a. Pros: Fast on Nvidia systems
    b. Cons: Only works on machines that have an Nvidia card
5. Vulkan
    a. Pros: It is easier to use. Works on most devices. Decent documentation
    b. Cons: It is newer, so it is not as stable

Operating systems:

- Windows 10:
    - Strengths: Widely available
    - Weaknesses: Additional software required to run the application
- Mac
    - Strengths: Widely available, Accessible libraries
    - Weaknesses: Lack of support for necessary tools

Possible solutions and Design alternatives:

We made a decision to use OpenCL in the project as it would work best in being able to meet the requirements of the project. OpenCL provides years of documentation to assist in accomplishing parallelization of the application. OpenCL is also cross-platform which will allow for this application to meet portability requirements.

The issue has come up with running the application on Mac due to OpenCL being deprecated. A solution to this problem was to make a decision to change the data types of variables that are used to run the rendering kernel.

4.5 Design Analysis

Our proposed design in 3.3 is still in development, though as it sits, we have confidence that it will work with some more development. Our implementation of OpenCL into the code is working smoothly, and we are able to change which device OpenCL is using for GPU parallelization. In addition, physics development has seen some improvements in the time leading up to the writing of this document, and we suspect that we will have the code in by the end of the project timeline.

We are aware that though we may have a working product, that does not necessarily mean that it will be bug-free. Part of our plan to improve upon our design is to remove as many of those bugs as we can. In addition to this, if everything looks to be adequately implemented to meet our goals, we plan on considering other features to add to the demo application to continue pushing just how much processing power the renderer can handle.

4.6 Design Plan

Having been given the base rendering engine at the start of the project, our first semester of work leading into the summer involves the improvement of its speed. This would include the implementation of OpenCL into the engine for GPGPU utilization as well as the octree world implementation and the octree traversal algorithm to further optimize the processes we are using to render.

Starting in the fall semester and spanning the remainder of our project time, we plan on developing a test application to showcase the capabilities of our rendering engine. This demo application will include the implementation of object importing via .vox files, object movement and physics, along with world lighting and GUI. If we have all of these things implemented and bug-free, we will be able to see how efficiently the engine runs, and then be able to determine how feasible a memory-based rendering engine would be.

5 Testing

Testing is an **extremely** important component of most projects, whether it involves a circuit, a process, power system, or software.

5.1 Unit Testing

- File importer: Load .vox files made in MagicaVoxel into Memworld to test the bounds of the program and ensure consistent results with what was created in MagicaVoxel.

- World importing: Ensure boundary wall remains initialized to avoid issues with the kernel. We also want to make sure that any predefined empty space is set to zero.

- Create OpenCL program: Make sure all necessary files are working on the system. Have all API calls for OpenCL defined for the kernel, work groups and enqueueing. Make sure the application can be built with the setup by running the makefile and watching for errors.

- Run OpenCL program: Have defined inputs with expected outputs for the application. Establish these inputs/outputs for the serial application and compare inputs/outputs for the OpenCL implementation.

- Kernel Renderer: Compare the rgb values that are being set to predefined expected values. Verify that the lighting is modifying the rgb values correctly based on a specific light source.

- Physics: Have a predefined input of objects in the world, have an expected outcome of how they will interact with the world, and check their coordinates to make sure the results match up with our expectations. This will be done with our own unit testing functions that we can write to validate outputs.

Tools:

- We will write our own unit tests in a separate c file that we create.
- MagicaVoxel for quick and testable world and object files

5.2 Interface Testing

- Internal API Calls: As our project is effectively a rendering API, we want to ensure that it is easy for developers to use it to render scenes defined in memory. In order to do this, we have a test program with a main function that shows the uses of the functions in the API. If anything seems counter to how it should be used or difficult to set up, we will be able to reorganize the API calls and functions to ensure the usage pipeline is smooth and simple.

- Startup CLI: We are creating a demonstration application to show the capabilities of the rendering engine. At the beginning of this demo, we have a sequence of questions that are asked before the rendering window is created. In order to test whether this sequence of questions and the interface they are inside of are effective, we plan on handing the project to our client as it progresses in order to get feedback on how good they feel using it. If necessary, we can also involve other people outside of the development of the project to run the same tests with.

Tools:

- AutoIt: A tool that lets us send keystrokes and move the mouse. It would be used for testing user interactions with the demo application

5.3 Integration Testing

- File and world importing:
    - File importing and world importing are critical to the project as they make sure that the world bounds and voxel array are properly generated. If they have any issues then the program will quickly crash when running, so it's vital that these are tested together to confirm they work.
    - These will be tested with our own created functions to confirm they work together without causing any issues in the generation.
- OpenCL and renderer:
    - OpenCL is the framework that is tasked with making this application more efficient. The renderer will work on generating the world that we define. The renderer will be parallelized using the OpenCL framework, so it will be critical that these two are integrated correctly. Possible issues that can arise from poor integration would be poor efficiency or the world not rendering correctly.
    - These will be tested by checking inputs and outputs for the renderer and verifying correct definitions for the OpenCL api calls/ setup for what we want to run for on the kernel.
- Physics and renderer:
    - Physics will involve many objects moving around the world and interacting with one another. With this, we need to make sure that the renderer is properly displaying the objects as they move and won't have any overlapping or graphical bugs.
    - This will be tested with our own function calls and visual testing of the world to confirm that no graphical errors are happening.
- Lighting and renderer:
    - Lighting will allow the application to produce light sources to introduce shading. The renderer is tasked with displaying the world which will include lighting components. Integration will be important to ensure that there are no abnormalities with how lighting is functioning in the world and having the application run efficiently.
    - This will be tested using function calls for our lighting definitions and verify that the visuals on the application are correct.

5.4 System Testing

- For our system level testing strategy, we will be taking all of the components from integration testing and making sure that each of the components is working properly in accordance with our requirements.

- We will focus on changing the voxel density, world size, window size, and render distance to make sure that we are meeting the input and output requirements that are given.

- System level testing is to go through full use cases and check that each system is behaving as expected. Interface testing would involve going through normal inputs and verifying that the system reacts appropriately for expected and unexpected inputs, examples would include inserting characters into the command line and simulating keystrokes in the demo. Unit testing would involve verifying results of specific modules during a use case, an example would be checking framerate. Integration tests would be for use cases that touch multiple components of the system. An example would be checking the lighting of an area after an object moved.

- Tools:
    - Dr. Memory for memory leaks and maybe fuzzing if we deem it necessary in the future.

5.5 Regression Testing

- We are ensuring that new additions do not break the old functionality by running our unit, interface, integration, and system tests before submitting a merge request for the code that is being created for the task assigned. We will also be creating the necessary tests needed for future regression testing of the new code.
- Some critical features that we do not want to break in this project are:
    - The kernel rendering function because that is what we use for displaying the scene to the user. If this is broken, the user wouldn't be able to use the program properly.
    - The file importer function because that is what we use for loading our scene with objects. If this is broken, the user wouldn't be able to see the world correctly.
    - Creating the OpenCL program pipe as well as running the OpenCL program needs to work. If OpenCL doesn't work, we won't be able to meet the FPS requirements as well as the world won't be rendered correctly.
    - Physics needs to be close to be correct. There is not a necessity for accurate physics, but we would like objects to fall and jumping to work for our demo application.
- The regression testing is driven by our requirements, but there will also be extra testing to make sure that modules needed to meet requirements are implemented correctly.

5.6 Acceptance Testing

- We have a list of requirements from the client such as a target frame rate, target voxel density, and target world size. Voxel density and world size are configurable variables inside our project, so we can simply set those variables to our requirements and run the program to make sure that those requirements are met. For frame rate, we are using a formula to calculate the frame rate of the project, and it is output in a command line interface, so we can take the average frame rate and compare that to our requirements to ensure that it is complete.
- We involve our client by showing memworld running to different specifications and ensuring that he is satisfied with the progress we are making. We run our ideas for new modules that we want to add by him to make sure that we are meeting what he invisions.

5.7 Security Testing

- We must ensure that buffer overflows are not possible when using input files. It is possible for users to write files in such a way that when a buffer overflow occurs in a given program, they can gain root or admin privileges. In order to avoid this, we must show that our program will not accept any inputs or input files that will cause a buffer overflow error or any other error that could allow the user to gain root privileges.

5.8 Results

The results of our testing will show how fast our project runs with the different options below as well as if the world is being rendered to the user correctly. This helps ensure compliance with our requirements of the application because these are the most important inputs for the frame rate of our rendering engine.

Target Specifications

Voxel Density	Window Size	Draw Distance	World Size	Average Frame Rate
5	1024 x 768	100	Height: 16 * Voxel Density Width: 25 * Voxel Density Depth: 40 * Voxel Density	Result of testing for FPS

Systems of Each Team Member

	GPU used	Operating System	CPU	RAM
Collin	NVIDIA GeForce RTX 2060	Windows 11	Ryzen 5 3600 6 core	16 GB
Cristofer	Radeon RX 580 Apple M1	Windows 10 MacOS	Ryzen 5 2600 Apple M1	16 GB 8 GB

Dalton	NVIDIA GeForce GTX 980	Windows 10	Intel i5-4690k	16 GB
Jay	NVIDIA GeForce GTX 1070	Windows 10	Intel i7-6700k	16 GB
Mason	NVIDIA GeForce GTX 1060 with Max-Q Design, Intel(R) UHD Graphics 630	Windows 10	Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz	16 GB
Wil	NVIDIA GeForce RTX 3070 NVIDIA GeForce GTX 970M	Windows 10 Manjaro Linux	Intel 10th Gen i7 Intel 6th Gen i7	16 GB 12 GB

Our design for testing is useful because we get to have our code tested on a variety of platforms and see what the performance is based on the different hardware that is available. This will help us determine if we are moving in the right direction or if we need to change how we are rendering and/or taking inputs from the user. The one downside of testing an application like ours is that we have to manually run systems tests which is harder to do because humans are not perfect at replicating actions. If everyone is testing, we should be able to detect what is wrong and what needs to be changed with most of the bugs that may occur.

6 Implementation

The progress made in this semester by our team would allow us to complete some of the major requirements of the original rendering application. We have been able to meet the requirements involving portability, performance and object importing and some physics. We will continue to implement more functionality regarding physics and lighting.

An implementation plan for next semester is to be able to implement a test application that will be able to demonstrate the capabilities of the design plan that we implemented for the rendering engine that we were given from our client.

Source File Descriptions:

In order to import .vox files, we take in a filename, a pointer to a world where we store the information, we can get the size of the object and put the values in the rest of the given parameters.

```
void file_importer_read_vox(char* filename, unsigned int *world, int
*object_x_size, int *object_y_size, int *object_z_size);
```

For our main program, we have an initialize_world function that sets up the objects and walls within our world (colors the voxels). The initialize_shader_and_texture function creates a plane and a texture that we modify in order to render colors to the screen. The mouse_callback changes the camera view based on the new x and y positions. The process input function checks if you are pressing buttons on the keyboard and moves the character/camera accordingly. The frambuffer_size_callback just sets the size of the OpenGL viewport (ex. 1920 x 1080).

```
void initialize_world();

void initialize_shader_and_texture(unsigned int quadVAO, unsigned
int quadVBO, unsigned int texture);

void mouse_callback(GLFWwindow* window, double xpos, double ypos);

void process_input(GLFWwindow *window, const uint32_t* world);

void framebuffer_size_callback(GLFWwindow *window, int width, int
height);
```

For our OpenCL implementation, we have a read_kernel_source function that reads a file and puts that into a string in order to compile it by the program later. The initialize_opencl_memworld function initializes OpenCL by getting the platform to run on (what GPU), building the kernel program, and creating buffers to use to transfer memory back and forth between the CPU and GPU. The compute_kernel puts the compiled kernel program into a queue to be executed by the GPU as well as writes the arguments required by the kernel to the function. The get_data_back function reads the data back from the GPU once it has finished running. The opencl_impl_free function frees all of the OpenCL buffers created by the program.

```
void read_kernel_source(char *filename);

void initialize_opencl_memworld();

void compute_kernel(uint32_t* world, uint32_t* pixels, int
window_width, int window_height, int world_height, int world_depth,
int voxel_density, float FOCAL_LENGTH, int max_draw_distance, camera
cam);

void get_data_back(uint32_t* pixels);

void opencl_impl_free();
```

In order to render the image above we use a kernel program to create rays and traverse the ray until we intersect a voxel with a color to render. The rays are rotated based on the camera's altitude and azimuth values. We scale down the color based on the distance of the ray intersection.

```
__kernel void render(__global unsigned int* world, __global unsigned
int* buffer, int window_width, int window_height, int world_height,
int world_depth, int voxel_density, float focal_length, int
max_draw_distance, camera cam)
```

7 Professionalism

This discussion is with respect to the paper titled “Contextualizing Professionalism in Capstone Projects Using the IDEALS Professional Responsibility Assessment”, *International Journal of Engineering Education* Vol. 28, No. 2, pp. 416–424, 2012

7.1 Areas of Responsibility

Area of Responsibility	**Definition**	**NSPE Canon**	**SE code of Ethics tie in**	**Differences between SE ethics and NSPE ethics**
Work Competence	Perform work of high quality, integrity, timeliness, and professional competence.	Perform services only in areas of their competence; Avoid deceptive acts.	Section 3: Strive for high quality, acceptable cost, and reasonable schedule and ensure significant tradeoffs are clear. Section 8 also promotes self growth, which leads to work competence	We can make software without having competence in the subject as long as we ask for the opinions of those who do have competence in the subject.
Financial Responsibility	Deliver products and services of realizable value and at reasonable costs.	Act for each employer or client as faithful agents or trustees.	Section 3 (Product) and Section 5 (Management) both discuss the cost of the product and ensure value for the client/employer	The SE code of ethics also includes the public as well as employers and clients in this section.
Communication Honesty	Report work truthfully, without deception, and understandable to stakeholders.	Issue public statements only in an objective and truthful manner; Avoid deceptive acts.	All sections, especially 1, 2, and 7, promote communication honesty between the engineer and the party in question.	The SE code of ethics has more information on communication honesty between multiple groups of people, such as clients, employers, colleagues, and the public.
Health, Safety, Well-Being	Minimize risks to safety, health, and well-being of stakeholders.	Hold paramount the safety, health, and welfare of the public	Section 1 promotes safety in regard to the public in terms of privacy, quality of life, and the environment.	The SE code of ethics provides additional remarks about protecting the environment and the privacy of the people.

Property Ownership	Respect property, ideas, and information of clients and others.	Act for each employer or client as faithful agents or trustees.	Section 2 states to act in the best interests of the client, their employer, and the public, showing similar traits to that in this area.	The SE code of ethics also includes the public as well as employers and clients in this section.
Sustainability	Protect the environment and natural resources locally and globally.		Section 1 states that engineers should disclose potential harm to the environment and only approve software that doesn't harm the environment.	The NSPE canon doesn't include anything about sustainability, but the SE code of ethics does.
Social Responsibility	Produce products and services that benefit society and communities.	Conduct themselves honorably, responsibly, ethically, and lawfully so as to enhance the honor, reputation, and usefulness of the profession.	Section 1 touches on the importance of only approving software whose ultimate effect is "to the public good."	The NSPE canon puts emphasis on conducting yourself properly to enhance the reputation of the profession, while the SE code of conduct does not touch on this aspect of social responsibility.

7.2 Project Specific Professional Responsibility Areas

Area of Responsibility	**Definition**	**NSPE Canon**	**Project Application**	**Group Performance**
Work Competence	Perform work of high quality, integrity, timeliness, and professional competence.	Perform services only in areas of their competence; Avoid deceptive acts.	Highly applies. Our project is based on enhancing performance of a pre-existing software, so knowledge of the product is necessary	Medium. We have had struggles with the project but have been preceding at a steady pace

Financial Responsibility	Deliver products and services of realizable value and at reasonable costs.	Act for each employer or client as faithful agents or trustees.	Potentially applicable. No current plans to spend funds but game design may include purchasing licenses/models later	N/A
Communication Honesty	Report work truthfully, without deception, and understandable to stakeholders.	Issue public statements only in an objective and truthful manner; Avoid deceptive acts.	Applies. Weekly reports are necessary, code should be understandable, and the client needs to be updated properly on work done.	High. Honest communication in meetings, reports, and with clients has been constant throughout the project
Health, Safety, Well-Being	Minimize risks to safety, health, and well-being of stakeholders.	Hold paramount the safety, health, and welfare of the public	Not very applicable. As long as we don't break a computer we shouldn't have to worry about this.	N/A
Property Ownership	Respect property, ideas, and information of clients and others.	Act for each employer or client as faithful agents or trustees.	Applicable. The project is based on Dr. Wymore's work and it should be acknowledged that he started the project.	High. We haven't betrayed Dr.Wymore's trust and haven't tried to change his original idea
Sustainability	Protect the environment and natural resources locally and globally.		Not applicable. Our project doesn't have a physical aspect to the project	N/A
Social Responsibility	Produce products and services that benefit society and communities.	Conduct themselves honorably, responsibly, ethically, and lawfully so as to enhance the honor, reputation, and usefulness of the profession.	Applicable. We carry ourselves with dignity when interacting with others to ensure the public image of engineers remain positive. For the project, can promote voxel and alternative rendering processes.	Medium. While the group hasn't been entirely professional, it has been able to match the overall tone of conversation with clients/ group members.

7.3 Most Applicable Professional Responsibility Area

Applicable Responsibility: Communication Honesty

What it means to the project: With the project being entirely code based, great communication is required to ensure everyone is up to date with changes being made live to the project.

How we have demonstrated the responsibility: All changes have been brought up in meetings or through group communication channels before being made. Consistent meetings have ensured communication continues to run smoothly.

Impacts on the project: Communication has caused the project to proceed at a bit slower of a pace, but has ensured that quality and understanding of where the project is has been consistent.

8 Closing Material

8.1 Discussion

This project is a mix of a product and an experiment. Our group was assigned this project to see if having a direct-memory representation of a voxel would increase the speed of different aspects of an application such as collision detection or rendering. The client already had a proof-of-concept ray casting engine and wanted us to improve the speed with certain requirements that needed to be met. As of this moment, the new renderer is roughly 100x faster than the proof-of-concept given to us in a much bigger world with a longer range of vision. We are meeting the FPS requirements of 30 frames per second on all the GPUs that we have available to us including an Intel integrated graphics chip. Collision detection is faster compared to that of modern game engines, but our group's rendering struggles to compete on a high texture density level because of the hardware acceleration for modern rendering on GPUs.

8.2 Conclusion

So far, we have implemented the GPGPU framework and the kernel function that we will be using to parallelize the rendering of the world. We have implemented a voxel object importer to help with world building for our game. We have also implemented some unit tests for our project. We have started working on threading for different tasks, lighting, and physics as well.

The goals for this project are to use a GPGPU framework to parallelize the rendering of a world that has a voxel density of 5, using a render distance of 100, on a window the size of 1024 x 768. We are also tasked with creating an application that will show off what we have completed in our rendering engine.

The best plan of action for this project is to use OpenCL because of its interoperability with OpenGL. We will use a version of an octree to help increase the speed of the renderer. We will create a 3D platformer to demonstrate how our changes to the renderer have been beneficial enough to create a working game.

The main constraint that has been keeping us from achieving the goal is that the more we add to the project, the slower the frame rate will be, so we have to find algorithms to increase the speed of the renderer or optimize certain parts of our code. In the future, we could use Vulkan instead of OpenCL and OpenGL because it has been said that Vulkan is more modern and easier to use. There are also other rendering methods in a grid based environment that could be used to try and maximize efficiency.

8.3 References

C. Crassin, F. Neyret, and Sillion François X., “Gigavoxels: Un pipeline de Rendu Basé Voxel pour l'exploration efficace de scènes larges et détaillées,” thesis.

Ephtracy, H. Castaneda, DimasVoxel, and J. Kirch, “Voxel Model Github Repository,” GitHub, 17-Jan-2022. [Online]. Available: https://github.com/ephtracy/voxel-model.

K. Silverman, “PND3D Demo and Source Code.” [Online]. Available: http://advsys.net/ken/voxlap/pnd3d.htm.

Laine, Samuli, and Tero Karras, "Efficient sparse voxel octrees–analysis, extensions, and implementation." NVIDIA Corporation 2.6 (2010).

Wilhelmsen, Audun, “Efficient Ray Tracing of Sparse Voxel Octrees on an FPGA” Norwegian University of Science and Technology, Trondheim

8.4 Appendices

8.4.1 Team Contract

Team Members:

1) ______Mason DeClercq___________ 2) ________William Blanchard______

3) ______Cristofer Medina Lopez ____ 4) ________Jay Edwards____________

5) ______Dalton Rederick___________ 6) ________Collin Reeves___________

Team Procedures

1. Day, time, and location (face-to-face or virtual) for regular team meetings:

Saturday, 1:00pm, Virtual (Discord) [team]
Tuesday, 4:30pm, VIrtual (Discord) [client]

2. Preferred method of communication updates, reminders, issues, and scheduling (e.g., e-mail, phone, app, face-to-face):

Discord

3. Decision-making policy (e.g., consensus, majority vote):

Consensus vote

4. Procedures for record keeping (i.e., who will keep meeting minutes, how will minutes be shared/archived):

Dalton is taking minutes for meetings. Meeting minutes are placed in the shared Google Drive.

Participation Expectations

1. Expected individual attendance, punctuality, and participation at all team meetings:

Attendance is expected unless notice is given beforehand. Members are expected to be for the most part on time unless notice is given beforehand. Participation in discussion and sharing work done in the previous week is expected for all in attendance.

2. Expected level of responsibility for fulfilling team assignments, timelines, and deadlines:

Follow up on assigned work and stated goals within the time frame decided on. If unable to complete them, provide reasoning why and update the group when it happens

3. Expected level of communication with other team members:

If you are "@"ed respond within 24 hours at least. We meet twice a week and everyone should be talking in the meetings when requested to speak.

4. Expected level of commitment to team decisions and tasks:

If a team commitment is being made that you cannot promise to follow at the time of its making, you should let the team know. If you can no longer follow a team commitment that was already made, you should communicate the issue as soon as you can. Otherwise, you should follow the commitments made by the team.

Leadership

1. Leadership roles for each team member (e.g., team organization, client interaction,

individual component design, testing, etc.):

William Blanchard, Parallelization Lead

Mason DeClercq, Team Lead

Jay Edwards, Documentation Lead

Cristofer Medina Lopex, Integration Lead

Dalton Rederick, Communications Lead

Collin Reeves, Game Development Lead

2. Strategies for supporting and guiding the work of all team members:

Within reason, help when asked for assistance. Provide guidance if you see someone struggling with an issue you know the solution to.

3. Strategies for recognizing the contributions of all team members:

Recognize the effort put into difficult portions of work by thanking members for their contributions.

Member with the least hours put into the project have to pay the tip for a celebration dinner*. Member with most hours put into the project gets to order dessert on the tab of the least member mentioned previously*.

*(only if they are ok with it, we will not force someone to spend money they do not want to spend)

Collaboration and Inclusion

1. Describe the skills, expertise, and unique perspectives each team member brings to the team.

- Jay - I've worked on software testing (junit, using git, and making gradle tasks) in my internship and some experience with making ray tracers in C. I also have done a few personal projects in Unity
- Mason - I have worked on multiple industry projects, so I have experience with managing my work. I have experience with optimizing C code. I have some experience with making games in Unity.
- Dalton - I've done long term design work in other courses, so I have experience with the requirements necessary for those kinds of projects. I am capable of programming in C.
- Collin - I have experience working in many group programming environments, along with a lot of experience in logging my work. Additionally I have experience working with C.
- Cristofer - I have experience working with C. I have experience coordinating with groups and using version control(git) to track work completed. I have some experience working with OpenMP to parallelize code for applications.
- Wil - I have a job as a "Game Engineering Intern," which essentially means I work with the Unreal engine and dig through C++ code to develop user-friendly applications. I had an undergraduate assistantship involving the use and parallelization of a custom graphics engine in VRAC and the use of computer vision libraries for pose estimation.

2. Strategies for encouraging and support contributions and ideas from all team members:

Recognize the effort put into difficult portions of work by thanking members for their contributions.

3. Procedures for identifying and resolving collaboration or inclusion issues (e.g., how will

a team member inform the team that the team environment is obstructing their

opportunity or ability to contribute?)

Have an understanding that messaging or contacting the group outside of team channels (I.E. private messaging) is acceptable for solving issues with group members. Contact the class professor if need be. Once an issue has been brought up, identify a solution per case basis.

Goal-Setting, Planning, and Execution

1. Team goals for this semester:

- Meet client parallelization requirements on the game engine with quality results.
- Have a working idea of what type of game/application that we will make to demonstrate the engine.

2. Strategies for planning and assigning individual and team work:

We will have a meeting every Saturday where we go over what the group has done for the week and see where we are with the project. We can then start planning what we want to accomplish for the next week. There will be overall goals and a schedule that we want to adhere to in order to get the most out of this project.

3. Strategies for keeping on task:

Understand that group meetings are meant to be kept on task, as time available to work together is limited.

Weekly reports for work completed should discourage idleness throughout the week.

Consequences for Not Adhering to Team Contract

1. How will you handle infractions of any of the obligations of this team contract?

Members who do not perform as expected from the team contract will be penalized per individual discretion during the team evaluation process.

2. What will your team do if the infractions continue?

Bring it up with the professor of the course and seek advice from higher-ups.

****

a) I participated in formulating the standards, roles, and procedures as stated in this contract.

b) I understand that I am obligated to abide by these terms and conditions.

c) I understand that if I do not abide by these terms and conditions, I will suffer the consequences as stated in this contract.

1) Mason DeClercq DATE 4/12/2022

2) Jay Edwards DATE 4/12/2022

3) William Blanchard DATE 4/12/2022

4) Collin Reeves DATE 4/12/2022

5) Dalton Rederick DATE 4/12/2022

6) Cristofer Medina Lopez DATE 4/12/2022