Brock University
Faculty of Mathematics & Science
Department of Computer Science

COSC 3P93: Parallel Computing

4P78 Project Step 2

Prepared By:

Parker TenBroeck 7376726
pt21zs@brocku.ca

Instructor:
Robson De Grande

October 9, 2025

1 Intro

2 Algorithm Overview

2.1 Projection

$fov =$ the field of view for our camera
$far =$ the farthest point our camera can see
$near =$ the closest point our camera can see

$$M_{projection} = \begin{bmatrix} \frac{1}{aspect * \tan(\frac{fov}{2})} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan(\frac{fov}{2})} & 0 & 0 \\ 0 & 0 & -\frac{far+near}{far-near} & -\frac{2 \times far \times near}{far-near} \\ 0 & 0 & -1 & 0 \end{bmatrix} \quad (1)$$

2.2 View

$P_{target} =$ Where the camera is looking
$P_{eye} =$ Where the camera is
$\overrightarrow{V_{world\ up}} =$ the up direction of the world $\langle 0 \quad 1 \quad 0 \rangle$
$\overrightarrow{V_{cam\ forward}} = \langle P_{target} - P_{eye} \rangle$
$\overrightarrow{V_{cam\ right}} = \left\langle \overrightarrow{V_{cam\ forward}} \times \overrightarrow{V_{world\ up}} \right\rangle$
$\overrightarrow{V_{cam\ up}} = \left\langle \overrightarrow{V_{cam\ right}} \times \overrightarrow{V_{cam\ forward}} \right\rangle$

$$M_{view} = \begin{bmatrix} \overrightarrow{V_{cam\ rightx}} & \overrightarrow{V_{cam\ righty}} & \overrightarrow{V_{cam\ rightz}} & \overrightarrow{V_{cam\ right}} \cdot P_{eye} \\ \overrightarrow{V_{cam\ upx}} & \overrightarrow{V_{cam\ upy}} & \overrightarrow{V_{cam\ upz}} & \overrightarrow{V_{cam\ up}} \cdot P_{eye} \\ -\overrightarrow{V_{cam\ forwardx}} & -\overrightarrow{V_{cam\ forwardy}} & -\overrightarrow{V_{cam\ forwardz}} & \overrightarrow{V_{cam\ forward}} \cdot P_{eye} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2)$$

2.3 Model

P_{pos} = world space location of the object
$\overrightarrow{V_{scale}}$ = x, y, z scale of object
$A_{rotation}$ = euler angles of objects rotation, raw,pitch,roll

$$M_{yaw} = \begin{bmatrix} \cos(yaw) & -\sin(yaw) & 0 \\ \sin(yaw) & \cos(yaw) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$M_{pitch} = \begin{bmatrix} \cos pitch & 0 & \sin(pitch) \\ 0 & 1 & 0 \\ -\sin pitch & 0 & \cos pitch \end{bmatrix}$$

$$M_{roll} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(roll) & -\sin(roll) \\ 0 & \sin(roll) & \cos(roll) \end{bmatrix}$$

$$M_{rot} = M_{yaw} \cdot M_{pitch} \cdot M_{roll}$$

$$M_{scale} = \begin{bmatrix} \overrightarrow{V_{scalex}} & 0 & 0 \\ 0 & \overrightarrow{V_{scaley}} & 0 \\ 0 & 0 & \overrightarrow{V_{scalez}} \end{bmatrix}$$

$$M_{rot\ scale} = M_{rot} \cdot M_{scale}$$

$$M_{model} = \begin{bmatrix} M_{rot\ scale11} & M_{rot\ scale12} & M_{rot\ scale13} & P_{pos_x} \\ M_{rot\ scale21} & M_{rot\ scale22} & M_{rot\ scale23} & P_{pos_y} \\ M_{rot\ scale31} & M_{rot\ scale32} & M_{rot\ scale33} & P_{pos_z} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

2.4 Clip

$$\begin{aligned} V_{world} &= M_{model} \cdot V_{local} \\ V_{clip} &= M_{projection} \cdot M_{view} \cdot M_{model} \cdot V_{local} \\ M_{normal} &= {M_{model}}^{-1^T} \\ \langle N_{world} \rangle &= M_{normal} \cdot \langle N_{local} \rangle \end{aligned}$$

$$\begin{aligned} -V_{clip_w} \leq & V_{clip_x} \leq V_{clip_w} \\ -V_{clip_w} \leq & V_{clip_y} \leq V_{clip_w} \\ -V_{clip_w} \leq & V_{clip_z} \leq V_{clip_w} \end{aligned}$$

Table 1: Performance Bench Brick 1080p

Resolution	1080x1920
Time	186.68 ms/frame
CPU	AMD Ryzen 7 3800X (16) @ 3.900GHz
OS	Debian GNU/Linux 12 (bookworm) x86_64
RAM	64GiB
Compiler	clang version 14.0.6
Triangles	40337

Table 2: Performance Bench Brick 720p

Resolution	480x720
Time	36.47 ms/frame
CPU	AMD Ryzen 7 3800X (16) @ 3.900GHz
OS	Debian GNU/Linux 12 (bookworm) x86_64
RAM	64GiB
Compiler	clang version 14.0.6
Triangles	40337

3 Performance

the average time to render 300 frames for the animation linked here

PUT THE FUCKING YOUTUBE VIDEO LINK HERE

is as follows

4 Paralization Opportunity

The code has been setup in a way such that paralization

Table 3: Performance Bench Halo 1080p

Resolution	1080x1920
Time	186.68 ms/frame
CPU	AMD Ryzen 7 3800X (16) @ 3.900GHz
OS	Debian GNU/Linux 12 (bookworm) x86_64
RAM	64GiB
Compiler	clang version 14.0.6
Triangles	40337

Table 4: Performance Bench Halo 720p

Resolution	480x720
Time	36.47 ms/frame
CPU	AMD Ryzen 7 3800X (16) @ 3.900GHz
OS	Debian GNU/Linux 12 (bookworm) x86_64
RAM	64GiB
Compiler	clang version 14.0.6
Triangles	40337

1. Mesh data is immutable and stored behind a shared pointer allowing access from multiple threads
2. Texture data is immutable and stored behind a shared pointer as well.
3. Scene data is light weight and can be copied with little cost if needed.

   Scene data is the `Camera`, `Light`s, and `Object`s.

   `Object`s only contain a shared pointer to their meshes and material textures so copying them is not costly.
4. When rendering everything aside from the frame buffer and local calculations are immutable and do not depend on eachother for computation.

4.1 Per Frame

The easiest approach is to render each frame in parallel. Because Mesh and Texture data is immutable it can be shared across any number of renderers running in parallel. The only cost would be creating the scene data for each frame running in parallel.

The major downside is each individual frame would still take the same amount of time. This is an issue if we want to use the renderer for real time applications like games.

Additionally each renderer would need its own frame buffer which can be quite large since each fragment(pixel) needs a significant amount of information like ambient, diffuse, specular color and texture data, world position, normal, tangent, bitangent. and more. A 1080x1920 frame buffer is $\sim 250MB$.

4.2 Per Triangle

4.3 Per Fragment (Pixel)

4.4 Combined

Because Triangle

5 Libraries Used

Used `tinyobjloader` [3] for loading and parsing OBJ and MTL files.

Used `stb_image` [1] for loading and parsing texture files into linear RGB.

Used `stb_image_write` [2] for writing rendered frames to disk.

These libraries are only used in the setup of the program when loading the 3D objects and material textures. They have no impact on the rendering performance.

6 Building and Runing

It is Highly reccomended to use Clang when building this project. Clang has much better instruction vectorization optimizations than GCC typically runs 5 times faster (this is only a rough estimate).

Two seperate build files are provided for each respective compiler `build_clang.sh` and `build_gcc.sh`.

Important: to switch between building with gcc or clang you `must` completely delete the build folder if it has been created previously with the other build script.

Once the project has been built you can run it with the `run.sh` script. Which will run the same scene used in 1. 300 frames will be written to `/animation` which can be combined into a animated video with either `make_mp4.sh` or `make_webp.sh` (optional).

You can specify the resolution of the animation by changing the parameters to the program in `run.sh`.

References

[1] S. Barrett, "stb image." [Online]. Available: https://github.com/nothings/stb

[2] ——, "stb image write." [Online]. Available: https://github.com/nothings/stb

[3] S. Fujita, "Tiny obj loader." [Online]. Available: https://github.com/tinyobjloader/tinyobjloader