Project 3P98: Game

Working Title: *Canyon Commander*

Tennyson Demchuk - td16qg@brocku.ca - 6190532

Daniel Sokic - ds16sz@brocku.ca - 6164545

Aditya Rajyaguru - ar18xp@brocku.ca - 6582282

COSC 3P98

May 3rd, 2021

Game Objective

Achieve the highest possible score by flying close to the ground for as long as possible! Be careful not to crash into the canyon!

System Functionality

The core gameplay functionality can be described as user manipulation of a three dimensional plane object that navigates a procedurally generated three dimensional space. In this space, terrain is generated and divided into four categories: Mountain, Ground, Sand, and Water. Mountain terrain generates at terrain peaks and it has a grey coloured texture. Ground terrain generates immediately below Mountain terrain and has a green coloured texture. Sand terrain is the lowest part of the terrain and is generated below Ground terrain, it has a yellowish texture. Finally, Water terrain is generated below a predefined plane and is set to appear on top of Sand terrain. At no point does Water generate on top of Ground or Mountain terrain. Before the user can begin the game, they are presented the controls, along with a prompt to press the left shift key in order to begin the game. When running the game in Visual Studio, controls and message prompts are printed in the debug console window. If the window is disabled the game will function normally however the user will not receive these notifications in the main game window. Once the user is ready, they are able to make use of the controls to move about the space. When the "Forward Thrust" key is held, the plane builds up momentum to move faster, which allows the user to move faster forward while fighting against gravity. When the "Forward Thrust" key is released, momentum is slowly lost as no thrust is applied and the plane object falls slowly as it is subjected to gravity. Throughout the game, gravity is constantly applied to the plane object, affecting both height and pitch of the plane, if left alone the plane object will fall to the ground. Also, the plane is not able to climb above a predefined height, as the play area is restricted to between the top of the mountains and the water plane. Upon collision with the ground, water, or any part of the terrain, the plane is considered destroyed and the user has lost the game. As the terrain is randomly generated the user must be careful and attentive in order not to crash into an unexpected mountain or hill. In order to achieve a higher score, the user must fly close to the ground but not crash for as long as possible, when flying close to the ground the score increases steadily. At any given time the user can interact with the game through specified keys on the keyboard, and is not permitted to interact with the game world in any other way. The game loads chunks randomly upon each start of the game, and will rarely have two complete terrain maps loaded in the exact same manner. The user may decide to pause the game at any point, in which case the screen will remain visible and running, however control is taken away from the user as they remain suspended in midair. Also, when paused chunks will continue to load. If through some means the user manages to go below a certain height level of the game world, they will be assumed to be out of bounds and will lose the game on the next frame. The user is not permitted to suddenly stop in midair, as the intention of gameplay involves constant flight simulation, with the user focusing on managing their speed, height, and position of

themself in relation to the terrain. When manipulating the yaw of the plane, the user is able to continuously rotate 360 degrees, allowing freedom of movement. However, the user is only permitted 180 degrees of pitch movement, as the plane is restricted from performing loops in such a manner. This has been purposefully restricted to prevent users from flying upside down, disorienting the user and mixing up the controls in the process. Also, the user is able to roll as well, however by a marginal amount. Full plane rotations could also cause disorientation.

Architecture

The system architecture is based around the Model-View-Controller (MVC) structure. In this architectural system, the Model represents all data-related logic that the user would interact with. The View relates to the UI logic for the game itself. While the Controller acts as the interface between the Model and View to take data from the Model and render it properly on the View. The following list of components indicate which aspects of our project relate to the corresponding component of the architecture:

Model:

- Camera
    - Position
    - Momentum
    - Yaw, Pitch, Roll
- World
    - Chunk
    - Terrain
- Plane
    - Position

View:

- Game Window
- Cursor Binding
- Text Display and Message Prompts

Controllers:

- Keyboard Input
- Mouse Offset Tracking
- Render Loop
    - World Updating
    - Gravity
    - Time keeping
    - Pause Logic

Implementation

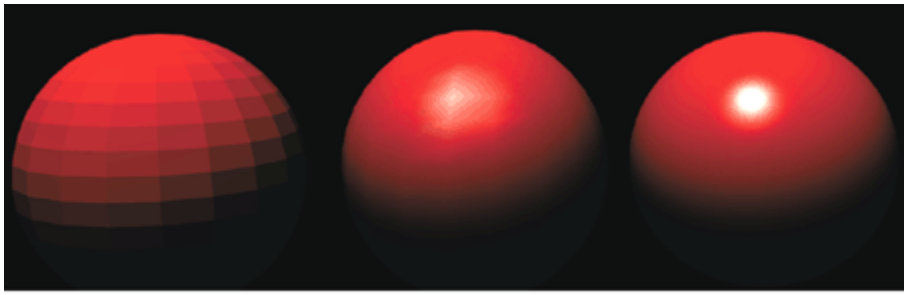
The following libraries and algorithms were used in order achieve our implementation:

GLFW and GLM

The GLFW and GLM libraries contained many necessary functions and variable types to properly render data, receive input, and initialize variables.

Phong Shading

Phong shading is a technique we utilized for improving the way our textures were rendered on screen. We primarily used Phong shading for water and grass textures. The phong shader is basically an interpolation technique that helps create a smoother surface when rendering leading to some better reflections as well. The following image (Retrieved from: https://opengl-notes.readthedocs.io/en/latest/topics/lighting/shading.html) provides a visual representation of the difference between various shader techniques. We can see that the flat shading is not very reflective or smooth, as you can see rectangular polygons shown all over the rendered image. The phong shader is implemented in the following way, the normal vectors are defined in the vertex shader and they are interpolated between the vertices and lighting is calculated by the fragment shader. Based on this it can then calculate the colours of the pixels appropriately showing off any reflections on the surface that may occur, and result in a smoother surface overall.



Procedural generation for terrain

We generate our terrain randomly each time the code is run. Initially we create a 2D vector containing values for noise from 0.0 to 1.0. We can then use this noise to generate terrain or visualize elevation, which is called a height map. The noise is generated using the Simplex method which is already defined in the glm library provided by openGL. We then also decided on a frequency value that determines the frequency at which noise occurs, this means that the higher the frequency the more peaks or mountains you end up getting. We chose a frequency value of 0.003, which establishes a fair number of peaks but also gives plenty of flat lands to

work with. Lastly, we also combined this with varying values of frequency to give us a mix of high peaks and small hills. You can see this code in the chunk.h file, the computeheight function specifically deals with the generation of these elevated peaks.

Caching

Our procedurally generated terrain was generated into chunks of memory, defined in chunk.h. The caching is just a matrix of all the nearby chunks around you and it has a specific set bound for how far a chunk of the terrain it can hold in memory. As you travel across the terrain, the chunks will be drawn on the canvas. The manner in which this is done is as follows: the program will check the cache, to see if the chunk exists in the cache. If the chunk is in the cache then it's simple just read it and draw it onto the canvas. If the chunk is not in the cache, then we do the procedural generation described above for that particular chunk of the terrain.

Camera

The camera class was implemented primarily using the glm library. The glm::perspective method was used to set the field of view for our game and we used the glm::lookat to represent the first person display in the game. The camera's positions are updated with the shift key where the camera gets a thrust vector allowing it to move forward. Gravity is also acting on the camera to pull the player downwards to make them crash into the surfaces. This is all done with vectors that have their weights updated gradually to simulate acceleration in the game.

Textures

We liked the aesthetic of minecraft, so the textures are taken from there to add to our terrain generation.

User Manual

Setup Checklist for Visual Studio:

- **Make sure to set Config to x64

- Using GLFW and GLAD

    - https://www.glfw.org/download.html

    - https://glad.dav1d.de/

- Go into Project Preferences -> VC++ Directories and add:

    - GL Dependencies -> lib folder to Library Directories

    - GL Dependencies -> include folder to Include Directories

- Go into Project Preferences -> Linker -> Input

    - Add "glfw3.lib" to Additional Dependencies

    - Add "opengl32.lib" to Additional Dependencies (this lib comes with the Microsoft SDK and does not need to be downloaded)

    - Add GL Dependencies -> glad.c to project source files

Important Notes:

- Controls, Game State, and Important information is shown in the command line window when playing the game
- If desired, to remove the background command line window in VS on Windows:

    - Project Preferences -> Linker -> System. Set SubSystem to WINDOWS

    - Project Preferences -> Linker -> Advanced. Set EntryPoint to "mainCRTStartup"

    ** OR: "#pragma comment(linker, "/SUBSYSTEM:windows /ENTRY:mainCRTStartup")"

Controls:

1: Wireframe Terrain

2: Solid Terrain

Left Shift: Thrust Forward

P: Pause

U: Unpause

W: Pitch Up

S: Pitch Down

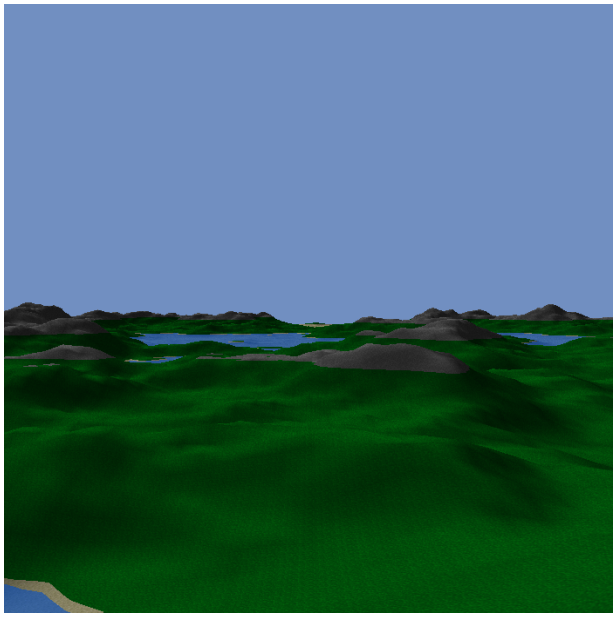
D: Yaw Right

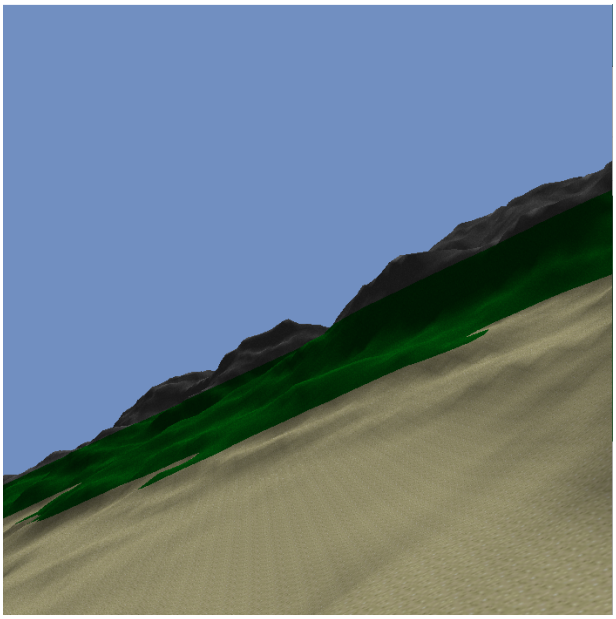
Q: Roll Left

E: Roll Right

Mouse: Pitch Up/Down, Yaw Left/Right

Screenshots:





References:

GLFW: - https://www.glfw.org/

- Graphics library framework

- handles OpenGL context creation, window loading, etc...

GLAD: - https://glad.dav1d.de/

- Initializes and manages platform dependent OpenGL function pointers

GLM: - https://glm.g-truc.net/0.9.9/

- OpenGL Mathematics library

- Simple header-only library used for vector and matrix operations using GLSL (OpenGL Shader Language) like syntax

- Also used to generate simplex noise for terrain generation

STB IMAGE: - https://github.com/nothings/stb/blob/master/stb_image.h

- Simple header-only image loading library

- https://www.redblobgames.com/maps/terrain-from-noise/

- interesting terrain generation using perlin noise

- https://www.minecraft.net/en-us/

- used for grass, sand, rock, and water textures

- https://learnopengl.com/

- used for basic setup instructions of modern OpenGL with GLFW+GLAD, simple shader source code loading library, and modern LookAt cameras technique with mouse controls

-https://stackoverflow.com/questions/13983189/opengl-how-to-calculate-normals-in-a-terrain-height-grid - "finite difference" method for fast terrain poly normal approximation

- https://stackoverflow.com/a/14010215

- stateful iterator for spiral coordinate generation