SRGGE: Lab Project Report

Albert Gil Saavedra

June 2019

1 Project

1.1 Contents:

The laboratory project has been developed using Visual Studio 2019 in Windows. The main folder contains the Visual Studio solution along other folders containing the files needed for the program. The main folders and files are the following:

Lab Project:

- Dependencies (folder): Contains all the dependencies of our program. Includes FT, GLEW, GLUT and SOIL.
- Lab_64(folder): Contains all the codes files and the resources files. It also contains the .dll files necessary for the execution.
- Res(folder): Contains all the resources needed for the program. The main folders are fonts, Models, shaders, Textures, Tilemap and Visibility.
- Src(folder): Contains all the code files and the glm folder.

Lab Session 6:

- Lab_6(folder): Contains all the codes files and the resources files.
- Output(folder): Folder where the output of the visibility is created in a .txt file.
- Tilemap(folder): Contains the tilemap used for the visibility calculations.

Application:

- This folder contains the executable application (*Lab_64.exe*), also contains all the .dll and the resources needed.

Application_Lab6:

- This folder contains the executable application for the Lab 6(*Lab_6.exe*), also contains all the resources needed.

In order to guarantee the correct behavior of the application all these files must remain in the same folder that they are.

1.2 Visual Studio Compatibility:

The application has been developed in the 2019 version of Visual Studio, so it may be not compatible with older versions. In order to execute the application using an older version of Visual Studio you will need to change the **Windows SDK Version** and the **Platform Toolset**. These can be found on **Properties** → **Configuration Properties** → **General**. Changing to the appropriate SDK Version and Platform Toolset the application will work. The Figure 1 show the Properties window with a changed values for a Visual Studio 2017.

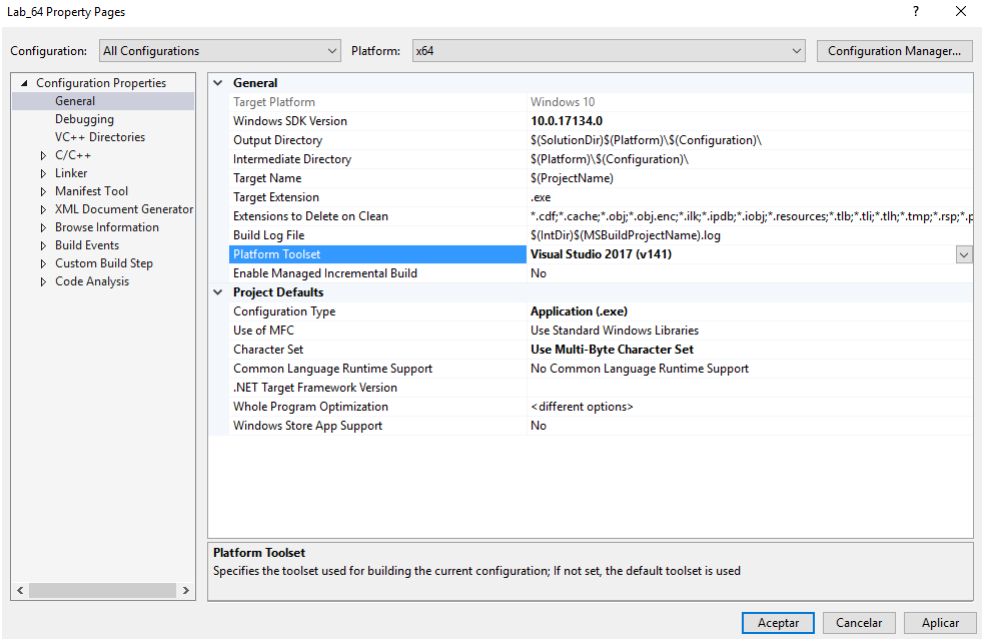

Figure 1: Properties window.

1.3 User Instructions:

The project is implemented in Visual Studio 2019, and configured in 64 bytes, so if it is executed in 32 bytes it is not going to work. It can be executed in "Release" or "Debug" mode. In order to compile the main program you need to:

1. Open the solution file "Lab_64.sln" with Visual Studio.
2. Select the mode "Debug/Release" and 64 bytes (Release gives better performance).
3. Click the run button also named "Local Windows Debuger".

The main program will start with the default mode(QEM). The camera can be rotated using the mouse and moved with the keyboard. The main actions are the following:

- Mouse Movement: Rotate the camera.
- Keys W/A/S/D: Move the camera position.
- Press key M : Change LOD mode. Available Modes: QED and Mean Vertex.

It is recommended to use the full screen to get a better performance of the camera implementation. While the application is running the mouse is not visible, in order to visualize the mouse outside the application when this is not in full screen mode, move the mouse fast to one side of the screen and the mouse will pop out of it. Also to get a better performance and faster initialization of the application it is better to use Release Mode.

The camera has no vertical movement. It can be changed by comment a line of code inside the Camera class in the *ProcessKeyboard()* function. It is indicated in the code the line which must be commented. This can be useful in order to check the visibility functionality.

It is also important to stay inside the map, if the camera gets out of the map you will get an error and the application will no longer work. The map is surrounded by walls so it is easy to identify the borders.

The FPS and the mode used are showed at the upside left corner.

The *lab6* application can run in both modes and configurations. In order to compile it, it is recommended to set the variable *numberOfRays*, located at the start of the main function, to a low integer. Otherwise it can be running for more than one hour.

2 Implementation

2.1 Session 1: FPS

As said before the FPS are showed at the upside left corner. The application is capable of render simultaneously multiple copies of models and different models.

2.2 Session 2: Tile map

The Tile map file(*Tilemap.txt*) is contained in the *Tilemap* folder inside the *Res* folder. It contains a rectangular map with 12 rooms delimited by walls, mapped as 1. The tiles mapped as 0 are the floor tiles and the others correspond to the models.

The reading implementation is done in the Tilemap class, while is loaded in this class, it is used in the Scene class.

2.3 Session 3 & 4: LOD

The calculations of the simplified versions of the loaded models are done in the LOD class. This class includes the implementation for the mean vertex calculations and the Quadric Error metrics. When applying the LOD to a model the mode used is determined by a boolean variable. To make things easier, all the LOD with both modes are loaded for each model at the start of the application, thing that makes the starting time larger, but allows to change the mode interactively while running the application. (M key.)

We can see that some parts of the models in low LOD using QEM have not mantained exactly the same shape. One case could be the bunny ears, which present sharp shapes. This could be an effect of using the bounding box from $(-1.0, -1, 0, -1.0)$ to $(1.0, 1.0, 1.0)$.

Also as a limitation of our implementation each time that the application is started all the LOD are calculated. A better implementation would be doing it just one time and saving it in a file in order to just load the files each time instead of calculating the LOD each time.

The advanced functionalities are not implemented.

2.4 Session 5: Time-critical Rendering

The time-critical rendering is implemented in the Scene class in the *selectLOD()* function. It calculates the LOD for each model every frame starting with the lowest LOD for all the models. The TPS, and FPS variables can be found in the *setParameters()* function in Scene class.

In order to represent the change of LOD more clearly, each LOD have a different color. The four colors used are:

1. White: LOD with max triangles. Grid made of 256 divisions for each component.
2. Blue: Grid made of 128 divisions for each component.
3. Red: Grid made of 64 divisions for each component.
4. Green: LOD with minimum triangles. Grid made of 32 divisions for each component.

The advanced functionality(hysteresis) is not implemented.

2.5 Session 6: Visibility

The visibility calculations are precomputed in the *Lab6* program. This program takes as input the tile map used, and generates a visibility file which is used in the main application. The visibility file can be found in the Visibility folder under the name: *visibility.txt*

The algorithm that I have implemented uses the Supercover Bresenham. My algorithm is based on the article *A Fast Voxel Traversal Algorithm for Ray Tracing.* [1]

The walls visibility have been computed as you could see inside the museum from a wall. So if the camera steps on a wall, the visibility that you get is only the front one, meaning that, you will only see the things at the front of the wall and the walls of each side. This is done in order to not lose visibility when you step in a wall by error. It is important to maintain the camera inside the museum, otherwise the application would return an error.

As said before, if you want to check the visibility, you can comment a line of code to unlock the vertical movement.

[1] A Fast Voxel Traversal Algorithm for Ray Tracing `http://www.cse.yorku.ca/~amana/research/grid.pdf`.