Functional Specification

Tessellation Sample

Fall 2017

C.Lavelle

Conor Lavelle

Version History

Version 1.0 - Draft

Contents

1. Introduction, Overview, and Purpose

This Tessellation Sample is a 3D graphics demo built in NVIDIA's Rendering Research Framework, Falcor.

1.1 Outcomes and Scope

The planned outcome of this project is to learn the tessellation pipeline

- There will be a series of individual effects that use the tessellation pipeline that the user can switch between with a UI.
- These effects will include displacement mapping, dynamic alteration of geometry by user input, terrain generation, water simulation, and an advanced tessellation effect that will be decided later.

1.2 Project Requirements

- The final project will be installable on any PC with Windows 10, and it will require about 2GB of ram.
- DirectX 12 is needed for this project.
- The installed project's size will be heavily dependent on the model and texture needs of the tessellation effects implemented.
- The TCRS as set out for GAM400 will be met, including the installer.

1.3 Development Requirements

- The main development environment will be visual studio 2017.
- NVIDIA's rendering framework, Falcor, will be the base on which I build the project.
- Visual Studio 2017 community edition will be used.

2. Functional Description

2.1 Install and Uninstall

The installer will install the sample to program files by default or wherever the user specifies. The installer will also handle any data files, like models, textures and shaders, and redistributables and dependencies required for the project to run. Uninstall will fully remove all installed components of the project.

2.2 Demo Features

2.2.1 Introduction to Tessellation

The project will include a mode where the user can view a wireframe screen space quad. Because the quad is wireframe, the user should be able to easily view the effects of changing tessellation types and factors.

2.2.2 Displacement Mapping

The project will include a mode where the user can load an arbitrary model and load an arbitrary displacement map onto it. Ideally, the user will be able to switch between no mapping, normal mapping, and displacement mapping. However, this might be limited by ability to generate normal maps that go

along with each displacement map. It may be the case that the user can only switch between no mapping and displacement mapping.

2.2.3 Dynamic Geometry Modification

The project will include a mode where the user can use their mouse to alter the geometry of a plane. Ideally, it will only tessellate the minimum amount at the point of the user's click, increasing the or decreasing the height of newly tessellated polygons based on the duration the user holds either left or right click.

2.2.4 Terrain Generation and Water Simulation

The project will include a mode where the user can enter a seed that generates an island surrounded by simulated water. The user will be able to alter the generated terrain on the island in a similar fashion to the plane altering demo described in 2.2.3. In addition, the user will be able to change the lighting in this scene, though the lighting and shading of the world is not really the primary focus of this mode.

2.2.5 Advanced Tessellation Effect

The goal of this project is to learn the tessellation pipeline. I currently only have a surface level understanding of this pipeline. Because of this, I feel it would be unrealistic to make concrete plans for the implementation of advanced tessellation effects. Thus, I outline here a few different possible effects that could be implemented towards the end of the project, most likely after milestone 2.

2.2.5.1 Improved Terrain Generation and Water Simulation

One potential final feature would be the improving of the terrain generation and water simulation. The water simulation has a very high ceiling for rendering and simulation quality. I could probably work on a water simulation all semester and not get close to reaching the level of water quality currently being worked on in the graphics world. I will aim to have the water react well to the user manually altering the terrain, make the world more dynamic by spawning things like rivers and waterfalls, and make the water simulation more robust and nicer looking in general. If this turns out to be easier than expected and there's a lot of time to add stuff, I could also potentially add simulated grass to the generated terrain.

2.2.5.2 Model Altering Tessellation Effects

Another potential final feature is to use the tessellation pipeline to apply interesting effects on models. One such effect would be to clip the geometry of a model to a height. By varying the height over the time, it would create the appearance of the model being built or warping in. Another such effect would be to target a volume for the geometry. By increasing or decreasing this volume over time, it would create the appearance of the model inflating or deflating. More specific control of the volume over time could create the appearance of a model pulsing like a heartbeat. This mode would allow the user to switch between the shaders and change properties of those shaders.

2.2.5.3 Fur Rendering

Another potential final tessellation-based feature is to render and simulate fur. This probably won't be available for arbitrary models/fur, but specific models will be prepared that showcase the functionality. In this mode, the user will be able to load specific models and apply forces like wind to the scene to view the fur properly simulating on the model.

2.2.5.4 Level of Detail

A final potential feature is level of detail. This is a very relevant problem to AAA game development, and might currently be the most generally relevant application of the tessellation pipeline. This mode will allow the user to load an arbitrary model and vary its polygonal detail. It will also allow the user to enable a setting where the polygonal detail of the model is automatically scaled by distance to the camera.

2.3 User Functionality

Much of the user functionality is described in the previous section. However, it will be expanded on here. Switching between modes, loading models and textures, and changing properties of modes will all be handled by Falcor's built in GUI system, which is just a wrapper around ImGUI.

2.4 Metrics

The only metric of interest will be framerate. I won't be playtesting and collecting user data to try and iterate on the design because there's very little design involved with this project. Instead, I will compare the framerate of different modes described above to get a better understanding of the relative costs of operations within the tessellation pipeline. In addition, I'll ensure that nothing I does completely tanks the framerate to an unacceptable (< 60fps on 1080ti) rate.

3. Specific Requirements

3.1 User Interfaces

3.1.1 Hardware Interfaces

The project will support keyboard and mouse input to interact with the GUI.

3.1.2 Software Interfaces

The software interface will be a GUI created from Falcor's ImGUI wrapper. The ability to pause and change the time scale is already built in to Falcor.

3.2 Networking or Communication Protocols

There will be no networking in this project.

3.3 Memory Constraints

There are no memory constraints. Though memory is not going to be wasted carelessly, I'm not shooting for any specific memory footprint. All of Falcor's objects have a Create() method that returns std::shared_ptr or std::unique_ptr, which should be sufficient for my use.

3.4 Framerate

As stated above, the project will need to be able to maintain a framerate of 60 fps or higher on a 1080ti or equivalent card.

3.5 Assumption and Dependency

I assume the project will be run on a 64-bit machine with Windows 10 with a graphics card that supports DirectX 12 and Shader Model 5.0.