 Computer Graphics – Assignment 4

# Objective

Today’s lab will work on the assumption you have completed Assignment 3. We will introduce lighting concepts and model loading. The types of lights you will implement are directional, ambient, and point light with attenuation. Model loading will be given to you through a header file, you need to turn the data given to you from the header file into your own working structures. Camera movement will also need to be adjusted to fit more to what the demo is expecting of you. The camera needs to always focus on the center of the model and orbit around that focal point.

Before we start let’s make new functions that will help us for today. You will write functions for the following:

* dot\_product
  + dot product between two vectors with x,y,z components
* cross\_product
  + cross product between two vectors with x,y,z components
* vec3\_normalize
  + normalize a vector with x,y,z components
* combine\_colors
  + additively combine two colors
* modulate\_colors
  + multiplicatively combine two colors
* saturate
  + clamps a value between 0 and 1
* vec3\_length
  + computes the vectors length

Let’s start today’s lab with the star field. You will be rasterizing 3000 points around the origin. Create a random x, y, and z direction between -1 and 1 then scale this position by 50. Store this data in some container. Once you have this data stored, before drawing them make sure to place the vertex into projection space by multiplying the view and projection matrix by this position. Homogenize after the multiplications and plot said pixel. As a safety check, make sure that the new homogenized coordinate is clipped properly before drawing the star.

Next we will work on the model loading and drawing. You will be given StoneHenge.h and StoneHenge\_Texture.h in order to load information from. The data type used to store the mesh data in the header file will be incompatible with our current vertex type so we need to load the data into our own vertex type array. First start off with the vertex information by looping through the number of vertices. Store the positions and scale them by 0.1f. (The model is enormous, so we will scale the model down). Store the texture coordinates and normal as well (DO NOT SCALE THE NORMALS AND TEXTURECOORDINATES). You can hard code the w component of the position vector to 1.0f. Once all of this data is properly stored, we will continue on and render this model. Loop through the number of indicies and draw a triangle at a time. Ensure you are using the index buffer to get the correct triangle information. You will be indexing into the vertex buffer using the index buffer. Your loop will increment by 3 indicies at a time. If you have not implemented near-plane clipping (Mastery 3) then you will need to keep the 3D model far away from you. Otherwise, it will lock-up due to negative W divide issues.

Here are some important values for your lights before we begin the lighting routine.

Directional Light: direction = -0.577, -0.577, 0.577; color = 0xFFC0C0F0

Point Light: pos = -1, 0.5, 1; color = 0xFFFFFF00

In order to do lighting we will accomplish this in the vertex shader. We will do per-vertex lighting and store the current lighting color in the vertex’s color data member. In the vertex shader you will transform the vertex’s position and normal into world space by multiplying them by the world matrix. Next we will find out how much our directional light affects our current vertex by calculating a ratio using the dot product between the vertex normal and the directional light’s direction. Saturate this value so it does not exceed 1 or drop below 0. Next we will lerp two colors (black, and the current directional light’s color) the ratio will be the ratio from our dot product added by some ambient value. Ensure to saturate so this value does not exceed 1 or drop below 0. At this moment we will also calculate the color value for the point light. You will run the point light algorithm here. Lerp two colors (black and the current point light color) the ratio for lerping will be the dot product between the vertex normal and the vector from the vertex position to the point light position. Once you have the two colors stored, we will combine them both. Afterwards, make sure to multiply the position of the current vertex by the view matrix and the projection matrix.

# Grading Breakdown

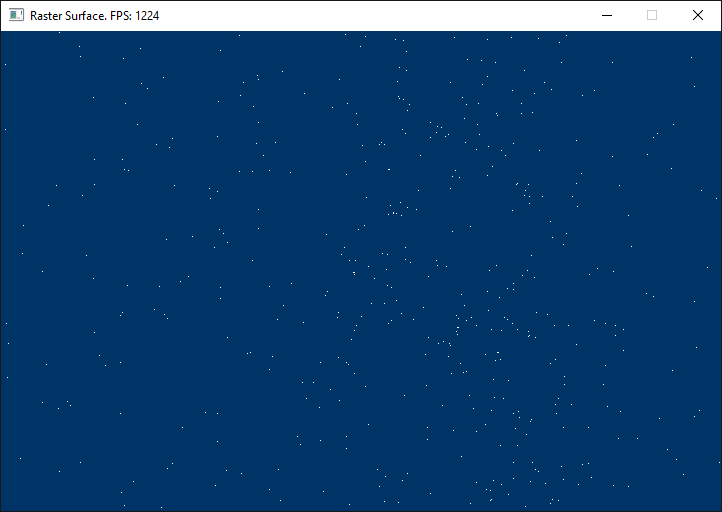
The grading categories are meant to assess the implementation details listed above.

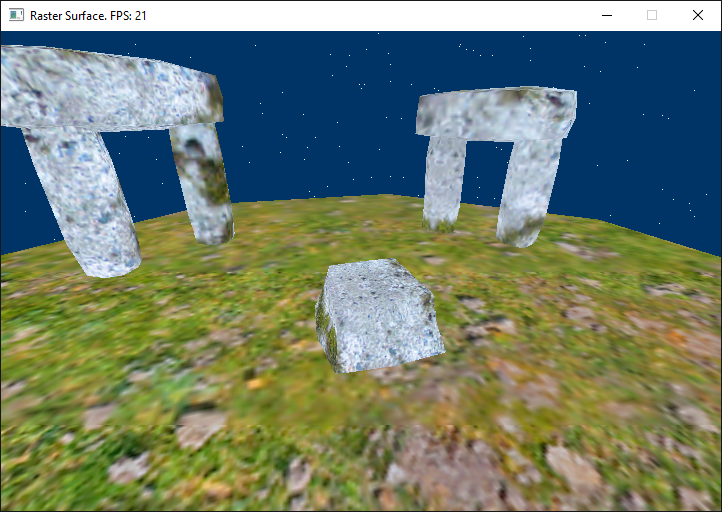
* 25% - Draw star field (must be 3D & support camera rotation)
* 50% - Load and Draw (including texturing) indexed geometry
* 75% - Add Directional and Ambient lighting
* 100% - Add Point light with expanding/shrinking radius (includes attenuation)

# Pre-Requisites

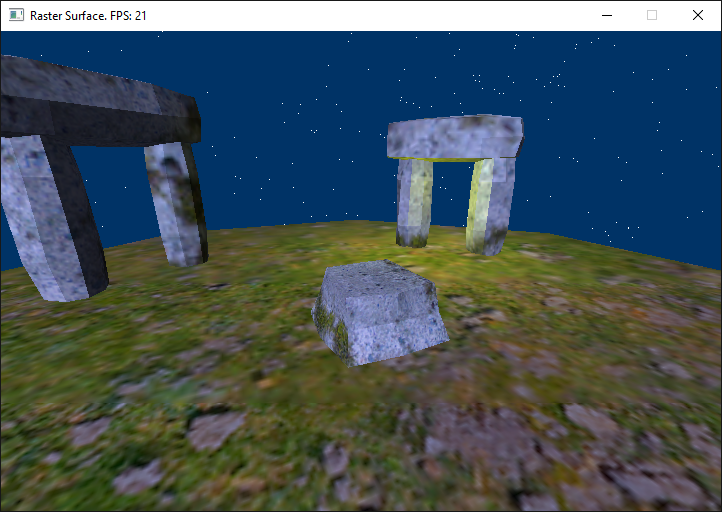
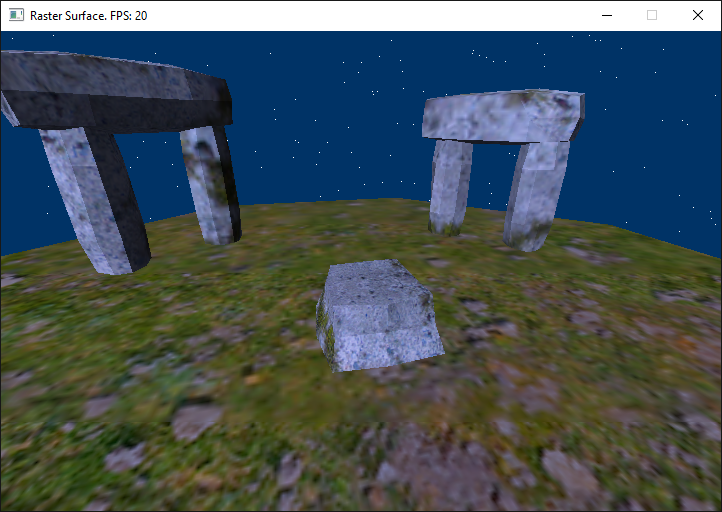
* A version of a Triangle drawing routine that supports 3D & Texturing. (Near plane clipping, only required to mirror screenshots below, not for points. You may stay farther back)

# Example Images

**25% 50%**



**75% 100%**



# Submission

Programming assignments will be turned into FSO. Follow the directions listed there carefully, failure to do so can significantly impact your grade in a negative way. If you did not entirely write your own code for this assignment, don’t turn it in. Weekly MOSS results will be posted to the Discord if available.