Assignment 4 Software Design Document

CS2300 Section 3 Spring 2023

Cameron Johnson

# Project Description

There are three parts to this assignment. Part A has three subparts. Subpart 1 takes values from a file, creates triangles using them, and then using an eye location from the file, determines whether or not each triangle is culled based on which way they’re facing. Then, using a light direction given from the file, the light intensity on each triangle is calculated, regardless of it being culled or not. Then, these processes are combined, and the light intensity calculation is only done on the rendered triangles.

Part B has 2 subparts. Subpart 1 does parallel projection given a point on the plane, the plane’s normal, the point to be projected, and the projection direction. This part finds the coordinates where the point is projected on the plane. Subpart 2 similarly does perspective projection, though this time finding where the point falls on the plane when being projected with respect to the origin.

Part C has two subparts. Subpart 1 takes a bounded plane and a line and finds the distance between the two. Subpart 2 takes a bounded plane and a line and finds if they intersect, and if they do, where the line and bounded plane intersect.

# Approach

Across all parts, I used methods to read the files and place the values from the file into the arrays they needed to be placed. I then created methods to carry out each step of the process. Some methods, like in my part B program, carry out the whole process alone, though occasionally if there was a process that was done multiple times in the process (such as a dot product), I created a separate method for that process. For all parts, I also created a method for printing to a file.

# Detailed Design

## Programming Language

I used Java for this assignment. Java became really useful in part A when it came to the multiple triangles. I was able to create an object for each triangle that could hold it’s normal vector and if it was culled or not, so those values could be called wherever needed using getters.

## Modules

**Modules either shared or work similarly between parts**

**readFileValues(File inputFile, arrays to be updated from file)**

This method simply reads the file and places values from the file into the given arrays

The process is formatted dependent on the requirements of the part

Input: file to be read, any array to be updated with values

Output: no return, input arrays updated with values from file

**readLines(File inputFile)**

This method reads how many lines are in the file

Some subparts will modify this method to skip the first line of the file

This method is used typically as to offer a size for an array holding values produced from a given line, for as many lines in the file

Input: file to be read

Output: number of lines in the file

**Part A**

**cullTriangles(inputFile, eyeLocation)**

This method reads each line from the file, using the values as points for the triangle. Then, using the eye direction, finds if the triangle is culled or not.

Input: file to be read, eye location

Output: an array of triangle **objects**, with each object holding a value determining if it’s culled or not

**calculateLightIntensity(triangle, lightDirection)**

This method calculates the light intensity on a given triangle, using the previous array of triangle objects

Input: triangle from array of triangles, light direction

Output: an array of light intensities corresponding to the given triangles

**cullThenIntensity(listOfTriangles, lightDirection)**

This method reads through the list of triangles and finds which ones are rendered. Rendered triangles are then put into a list of their own, which the light intensity on each of these triangles will be calculated.

Input: array of triangles, light direction

Output: array of light intensities on ONLY the rendered triangles

**Part B**

**parallelProjection(inputFile, pointOnPlane, planeNormal, projectionDirection)**

This method calculates the position of the projected points on the plane

Input: file to be read, point on plane, the plane’s normal, projection direction

Output: matrix of parallel projections, each row of the matrix corresponds to the given points in each line of the file

**perspectiveProjection(inputFile, pointOnPlane, planeNormal)**

This method calculates the position of the projected points on the plane with respect to the origin

Input: file to be read, point on plane, plane’s normal

Output: matrix of perspective projections, each row of the matrix corresponds to the given points in each line of the file

**dotProduct(vector1, vector2)**

This method calculates the dot product between two vectors given they’re of same length

Input: two vectors

Output: dot product between two vectors

**Part C**

**partCsubPart1(inputFile)**

This method carries out the first part of part C

Input: file to be read

Output: no return, prints to file the distances between points and their plane

**partCsubPart2(inputFile)**

This method carries out the second part of part C

Input: file to be read

Output: no return, prints to file the intersect point/”Does not intersect”

**dotProduct(vector1, vector2)**

This method calculates the dot product between two vectors given they’re of same length

Input: two vectors

Output: dot product between two vectors

## Flowcharts

My flowchart was prioritized least of all my tasks for this assignment. Due to this and the near approaching deadline, I was unable to make a flowchart for this assignment.

## Key Data Structures

Objects were a huge help in part A, given that I could keep the info of each triangle, such as its normal and if it’s culled or not, in an object of it’s kind. Helps when updating multiple values for each triangle.

The gauss elim algorithm in part C is integral for that subpart to work. I’m not sure I would’ve been able to figure out that algorithm myself easily.

While it’s really only used in part B, the ability to print to two separate files with one method is helpful. Using a method, passing in the values to be output and the name of the file to be written to made printing for those two parts easy.

## Test Description

The only test file I used was the given test input file. Part C subpart 2 requires my own values, as every output for that subpart is the same, but for parts A and B, the one test file provides many values for testing.

For part A, the test file provides 9 tests, as each triangle and the process of culling and finding light intensity can be seen as a test. I tested 4 of these triangles and their outcomes: three rendered triangles and one culled triangle. One thing I found interesting was that there were triangles that were rendered, however had a light intensity of 0. I had initially thought this was a bug in my program, but my handwritten work shown that the light intensity on those triangles were negative, in which case the light intensity would be 0. All of my handwritten work matched the output in my program.

In part B, same applies. Test file provided many separate tests. I tested 3 points in parallel projection, and 3 points in perspective projection. My handwritten work matched my program output.

In part C, same applies for subpart 1. I tested three of the distance outcomes with handwritten work, and they were all matching. I was told that for subpart 2, the test file outputs “Does not intersect.” for every input, and while my program did initially have this same output, it does not show the output of an intersecting line and bounded plane. I needed to make my own test for this.

[Update] For part C, I do not have enough time to test if an intersecting plane/line provides the expected output. However, given that the gauss elimination algorithm is coded correctly, and assuming that the checks on the U vector work properly, it should output accurate numbers given a linear system that has an intersecting plane/line.