Gavin Franklin

11/4/16

Project Proposal

Summary:

Title: The Shape Animator

Description:

The program will allow the user to click buttons to allow specific three dimensional shapes to appear in a plane. Then allow them to manipulate the shapes size and location through certain key commands then they will be able to make a new plane with no shapes in it, create more shapes and scroll back through their previously made shapes.

Intended user:

This would be a useful program for anyone who wants to make basic objects and manipulate them on a very basic level. It will be simple to use but lacks a lot of functionality.

Problem:

Mainstream 3d renderers such as 3ds max or Maya are great tools but can be somewhat overwhelming to a first time user. This program would provide something that’s very in between that allows for shape rendering but lacks functionality.

Technologies:

I’ve been looking into opengl for C++ which should help me with the rendering process. To do this I need to use freeglut a C++ library that is an extension of a much older library called Glut. Since creating a window and linking it to a GUI is quite troublesome I’ve decided to go with and existing function in opengl that allows assigning operations to specific key commands. This program will not run or compile without the glut library. In order to get that library I would suggest following this YouTube video <https://www.youtube.com/watch?v=_OAYRV5fjuE> for the first few minutes.

Data:

The data in this project is simple. The position of the shapes, the color and the size of the shapes are all stored within a double linked list of nodes. When the display function creates shapes it goes to the current node to check to see what shape as well as its properties such as their color, size, rotation and even position in the plane. Whenever the user clicks buttons to alter the shape the only thing that is being altered is the information in the node. The user can then flip through the shapes because the nodes are part of a double linked list. There will be data that will not change such as the z coordinates because you can get the same effect by manipulating the size, or the fineness of the shapes. To make sure this data does not change it will be an automatic input into the shapes parameters. The lights in the function are also another piece of data that is vital to the program. Global variables or lists are created in order to map out the lights. Later in the main function these lists will be applied to an aspect of a light and will serve as sort of mapping of it parameters. One other very important piece of data is the amount of nodes which will be useful when saving and loading. This is just a counter that increments every time a new node is created.

GUI

There will be absolutely no GUI other than the three dimensional space. The reason for this is really complication. Making both a three dimensional plane as well as a GUI would require multiple libraries but it would be possible. The big problem is actually linking the GUI to the grid since the grid has a location function but its parameters only alter where on the screen it’s located so it would be a headache to try and incorporate both into one program. Instead what I plan on doing is creating a readme document that instructs the user on how to use the program. Instead of clicking buttons with the mouse the user will rely on a fixed set of hotkeys to manipulate the shape courtesy of another Glut function.

Window:

a = rotate left p = prev window

d= rotate right r = add or sub red

s = rotate down g = add or sub green

w = rotate up b=add or sub blue

j = move object left q =quit

l = move object right -=minimize

k = move object down +=Maximize

I = move object up z=save

1 = sphere x=load

2=cube m=animate

3=cone space = center

0=clear

N=next window

Algorithm:

Before my main function there are pointers to instances of nodes created globally on the heap so the rest of my program can use them. Other global variables are declared that will be useful in save in load such as string stream as well as creating variables that will be important for describing the lights. The first thing my main function will do is create a window by initializing it, giving its dimensions, and naming it. In the initialize function it will set the default window color and tell the program to use a depth check that will help with 3d feel. Then back in the main it will give the window each of the functions it needs to run display, idle, reshape, and for my purposes the key down. The last thing the main will do is create the lights based on pre-declared variable that give the coordinates and brightness of the lights. The second set of lights will be dedicated to the shapes themselves. Anything in the window will be affected by these lights. The reshape function will take in two variables that it gets from the system which are the dimensions of the new window. The reshape function then uses these dimensions to recreate the window and everything in it allowing the user to manipulate the window at will without affecting the shape in the window.

The node class which has instances in the main will be used only to hold data. Each node is a part of a double linked list with its fellow nodes and has a previous as well as a next node. A nodes properties include shape, colorb, colorr, colorg, posx, posy, rotx, roty, and size all stored as either integers or floats. Each of the properties is used when creating the shape and can be manipulated by the user. Because of this each property has a getter and setter.

Both the idle and the display functions in main.cpp are the same. This function creates a shape in the window based on the information held within the shape property of the current node. Once a shape is determined display then pulls more properties in order to make up the color, position and rotation of that shape. The key function allows the user to change information in the node therefore changing the information of the window or shape by certain key bindings. Certain key functions that manipulate basic node data such as size color and position will be handled directly from the key function where as some functionality such as save and load will be handled in functions of their own. One key that is worth mentioning is the m key which is handled in the key function. This creates an animation loop that goes through exactly one time and has a delay property that is created by the sleep function. This function is usable because of the importation of the time library. The save function allows the user to save all the nodes information to a text document while the load function clears all the existing nodes and creates new ones using the information in the file. The way this works is the save function will first output the counter variable which keeps track of the amount of nodes while the load function takes that number and performs a for loop that creates as many nodes and fills them with information as counter specifies. Finally when q is pressed the key function redirects the program to the function exitcommand which removes all of the nodes on the heap and closes the program.