CS 3920/5920 - Computer Graphics Fall 2017

Program: 3 – More 3-D Viewer

Points: 18

Due Date: November 7th

Grace Date: November 10th - demonstrated no later than 3:00 p.m.

You will continue to work in your groups. You must put the project files in a Prog3 folder in your group S share. Sign up for a 15 minute demo time. You will be modifying program 2.

You will keep the ability to walk around the scene, but will allow multiple figures in the scene. These figures will be undergoing transformations (rotations, scalings, translations) based on a timer. The rotation and scaling must be with respect to the fixed point, which for this program will be the center of the bounding box. Also, you will be doing a perspective transformation.

You will use a FolderBrowserDialog to read in all WRL files (top level only, don't recurse into subdirectories). You will store these figures in a figure list. You will associate "movements" with each figure. The movement will be triggered by a Timer. You will use classes for the movements. These classes will descend from an abstract MovePattern class. That way, you can have state variables to make "clever" movement functions. Also, it decouples the movement pattern from any particular figure.

Again, since we will be making changes to this in the future, no extra credit for this program.

**Required changes to the Figure class:**

1. Add a display matrix, fixed point and translate amount as we discussed in class. The second two will be Vector3. The fixed point must be the center of the bounding box of the figure. Once set, the fixed point doesn't change. Initialize the display matrix to a translation by minus the fixed point and the translate amount by the fixed point, and proceed as discussed in class. These will be set in Figure’s Load method.
2. Add a bounding box. This can be done with just two points, e.g.,

private Vector3 max;

private Vector3 min;

that give the maximum and minimum extents of the object. You can calculate these wherever you feel is appropriate. Since it is only done once per figure, it doesn't need to be that efficient. For example, you could loop through verts in Load.

For this program, you will only use it to specify the fixed point (figure center), which is average of the max and min of each dimension. In program 5, you will probably use it to detect collisions.

1. A read-only property to return the CurrentCenter of the figure (Vector3). Note that this is simply the value stored in translate amount. You will probably use this in some of your movement functions.
2. General Rotate, Scale, and Translate methods as discussed in class. These only change the display matrix or translate amount. These must take parameters that make sense. For Rotate and Scale, you must use Matrix4 methods. For Rotate, you could make 3 separate methods, one for each of the axes or a more general one and then use Matrix4.CreateFromAxisAngle.

Please remember that Matrix4 assumes vertices on the left whereas OpenGL has them on the right. So the order of multiplication is opposite. However, LoadMatrix will take care of it since it assumes column-wise storage whereas Matrix4 assumes row-wise storage. So the "transpose" is "automatic".

1. A Restore method that puts the figure back to the way it started. Simply "reinitialize" the display matrix and translate amount as specified in #1 above.
2. Modify the Show method to use the display matrix and translate amount.
   1. Add a parameter to pass the View (LookAt) Matrix which was set in the Form1 ShowFigs method.
   2. Before GL.DrawArrays, calculate the ModelView matrix as:

DisplayMatrix \* Matrix4d.CreateTranslation from the translateAmount \*

View (LookAt) matrix passed as a parameter

Note this is the reverse of how we normally would think about it.

* 1. GL.LoadMatrix the "show matrix" from step b) just before GL.DrawArrays

**Required FigureList class:**

I am requiring you to make a FigureList class (separate file). All manipulation of the figures (except Movement) must be done via this class. We will be making changes to it in future programs.

You need to comment and put in required “usings”, but it needs to start out as given below (after comment block and usings). The LoadFigures will read in all WRL files in the specified folder and create a FigureMovementPair for each. You can use whatever pattern you want for associating movements (e.g., ordered, random, etc.) with a given figure. You will also need methods to Show, Move, and Restore all the figures in the list. For getting the files in a folder, you can use System.IO.Directory.GetFiles. You then need to handle those that "EndsWith" .wrl

You can only have the single data member as specified below.

class FigureList

{

private struct FigureMovementPair

{

public Figure fig;

public MovePattern movement;

}

private List<FigureMovementPair> figlist =

new List<FigureMovementPair>();

public void LoadFigures(string folderName)

{

**Required MovePattern class:**

Your must have movement classes. These will be in a separate file; however, all movement classes can be in the same file (and in the same file as MovePattern). You need to comment and put in required usings, but it needs to start out as given below (after the comment block and usings), except you will make **good** names for the descendant classes.

You must make four (4) descendant classes, each substantially different. **Each** Move method **must** use Translates, Rotates, and Scalings. For example, moves can have random patterns, sinusoidal patterns, fixed patterns, etc. Figures can spin one way, then another. Figures can get bigger and smaller. You could do something if the CurrentCenter gets to a particular place, etc.

abstract class MovePattern

{

abstract public void Move(Figure fig);

}

class MovePatternWhatever : MovePattern

{

// You can declare whatever state variables you need to define the move

public override void Move(Figure fig)

{

// Want rotates, scales, translates - clever moves!

// As an example of a start - move a little in the X.

fig.Translate(0.1f, 0, 0);

. . .

**GUI Requirements & Changes to Form1.cs.**

1. Keep ability to move around the scene.
2. Replace File Open with a Load Figures menu item or button. It puts up a FolderBrowserDialog and uses the selected folder to read in all WRL figure files in that folder (i.e., passes it to the figList LoadFigures method).
3. A Reset button that puts the view back to the original (5, 5, 5) and puts all figures back to their original positions and shapes.
4. A Timer. When the timer goes off, it calls the Move method and then calls something like ShowFigs, which would be similar to last program: clear, set the Lookat, call FigList's show, call Axes' show, and then swap buffers.
5. Remove the Loading of the ModelView from ShowFigs. Instead, pass the LookAt matrix to figList.Show (which passes it to each Figure in the list) and also pass it to the Axes Show method. **The ModelView matrix will be calculated and loaded in the Figure and Axes class.**
6. **You are NOT allowed to use** GL. Rotate, Scale, Translate, MultMatrix, PushMatrix, PopMatrix, or LoadIdentity. Use Matrix4. This program, I am allowing GL.LoadMatrix
7. A natural way to start, stop, speed up, and slow down the timer. I used a Trackbar, with 0 meaning stop and as you slide to the right, everything moves faster. But you can use other ways. Whatever you use must allow a “stop” moving capability.
8. If your previous program didn't allow small changes in the viewing position (e.g., by 0.1), fix it for this program. Also, fix any errors you had from last program or you will lose points again!
9. You can't have a Figure reference anywhere in Form1.cs. The only state variable I had in Form1.cs was:

private FigureList figList = new FigureList();

If you have any others, you will need to justify why.

**OpenGL startup:**

Experiment with a good setting for the perspective transformation. You can start with something like the following in the Form Load event handler: (Make constants versus the magic numbers I have below):

GL.Enable(EnableCap.DepthTest);

float mult = (float)glControl1.Height / (float)glControl1.Width;

const float WINDOW\_SIZE = 2.0f;

const float WIN\_NEAR = 2.0f;

const float WIN\_FAR = 80.0f;

Matrix4 projMat = Matrix4.CreatePerspectiveOffCenter(

-WINDOW\_SIZE, WINDOW\_SIZE, -WINDOW\_SIZE \* mult,

WINDOW\_SIZE \* mult, WIN\_NEAR, WIN\_FAR);

GL.MatrixMode(MatrixMode.Projection);

GL.LoadMatrix(ref projMat);

GL.MatrixMode(MatrixMode.Modelview); // Set so it can be used in Showing

The mult factor is to maintain the proper proportions based on the size you make your glControl Window. Of course, the better option is to make your glControl square.