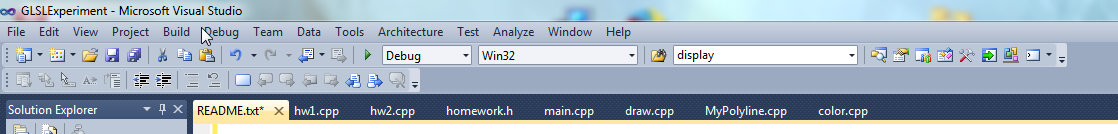
**CS 4731 Homework 2 Robert Dabrowski**

**HOW TO COMPILE AND RUN**

1. Double click on the "HW1" icon to load in visual studio 2010
2. Ensure that you have the settings in circled in red below
3. Press “F5” to build and run



**Mandatory Key Controls**

* **User hits 'W' (Draw your wireframe)**at a suitable initial position from the viewer.
* **User hits 'N' (Draw next wireframe)**Organize the PLY files in a list going from 1-43. Hitting N should load and draw the next wireframe model to the current one in your list of PLY files. You can hardcode filenames if you want. The PLY files may not all be of the same size. So to properly set up the viewing position using LookAt, you may have to calculate the bounding box of the mesh and then set your view distance to a suitable multiple of the bounding box
* **User hits 'P' (Draw previous wireframe)**Organize the PLY files in a list going from 1-43. Hitting P should load and draw the previous wireframe model to the current one in your list of PLY files.
* **User hits 'X' (Translate your wireframe in the +ve X direction)**Continously move your wireframe some small units along the +ve X axis and redraw it. Use the idle function to animate this. The ply file should continue to slide along the +ve X axis till the user hits 'X' again. Essentially, the 'X' key acts as a toggle key. If the ply file is stationary and the user hits the 'X' key, the ply file should continue to slide along the +ve X axis until the user hits 'X' again. Camera position remains fixed for this translation and all other translations below. The exact amount to move the ply file before redrawing will affect how much and how much your translation is apparent depends on how far you positioned your wireframe from the viewer. So, it's left to you as a design choice to pick an appropriate distance to translate the wireframe along the +ve X axis each time the user hits 'X'.
* **User hits 'x' (Translate your wireframe in the -ve X direction)**Use the idle function to continuously move your wireframe some units along the -ve X axis. The number of units to translate your wireframe each time the user hits 'x' is left to you as a design choice.
* **User hits 'Y' (Translate your wireframe in the +ve Y direction)**Use the idle function to continuously move your wireframe some units along the +ve Y axis. The number of units to translate your wireframe each time the user hits 'Y' is left to you as a design choice.
* **User hits 'y' (Translate your wireframe in the -ve y direction)**Use the idle function to continuously move your wireframe some units along the -ve Y axis. The number of units to translate your wireframe each time the user hits 'y' is left to you as a design choice.
* **User hits 'Z' (Translate your wireframe in the +ve Z direction)**Use the idle function to continuously move your wireframe some units along the +ve Z axis. The number of units to translate your wireframe each time the user hits 'Z' is left to you as a design choice.
* **User hits 'z' (Translate your wireframe in the -ve Z direction)**Use the idle function to continuously move your wireframe some units along the -ve Z axis. The number of units to translate your wireframe each time the user hits 'z' is left to you as a design choice.
* **User hits 'R' (Rotate your wireframe about it's CURRENT position)**Just like in a showroom where the wireframe is on a swivel, rotate your wireframe smoothly 360 degrees at a moderate speed about its CURRENT position (not about the center of the scene) This rotation is NOT the same as moving the wireframe in a wide arc. The rotation should be about the Y axis and the wireframe should not translate while rotating. After each 360 degree rotation of the "current" PLY file, load and display the "next" (of the 43 PLY) files. In this way, after 43 cycles, all polyline files should have been drawn one by one. On the 44th cycle, go back and display the first PLY file that was drawn. Finally, alternate between rotating PLY files clockwise and counter-clockwise. For instance, PLY file 1 should rotate 360 degrees clockwise before loading PLY file 2 which rotates counterclockwise before loading PLY file 3 which rotates clockwise, and so on.  
    
  Hint: Use double buffering (glutSwapBuffers( )) to make the rotation smooth. You can continously update the new wireframe positions and redisplay the meshes in the glutIdleFunc function.
* **User hits Key 'c':**Toggle between 1 color and randomly selected colors for edges: When toggled ON, the mesh edges are drawn using randomly assigned colors. When OFF, the mesh edges are drawn using red.
* **User hits Key 'h':**Increment the amount of shearing of the wireframe along the X axis by a small amount. Repeatedly hitting the 'h' key should shear the wireframe by a bit more and more. Note that after you shear the mesh, performing a transform (e.g. rotation, scale or translate) should transform the sheared mesh.
* **User hits Key 'H':**Decrease the amount of shearing of the wireframe along the X axis by a small amount. Repeatedly hitting the 'H' key should shear the wireframe by a bit less and less. Note that after you shear the mesh, performing a transform (e.g. rotation, scale or translate) should transform the sheared mesh.
* **User hits Key 't':**Increment the amount of twisting of the wireframe around the Y axis by a small amount. Repeatedly hitting the 't' key should twist the wireframe by a bit more and more. Note that after you twist the mesh, performing a transform (e.g. rotation, scale or translate) should transform the twisted mesh.
* **User hits Key 'T':**Decrease the amount of twisting of the wireframe around the Y axis by a small amount. Repeatedly hitting the 'T' key should twist the wireframe by a bit less and less. Note that after you twist the mesh, performing a transform (e.g. rotation, scale or translate) should transform the twisted mesh.

**Additional Key Controls**

* **User hits Key ‘D’:** rotate about X axis in the positive direction
* **User hits Key ‘d’:** rotate about X axis in the negative direction
* **User hits Key ‘E’:** rotate about Y axis in the positive direction
* **User hits Key ‘e’:** rotate about Y axis in the negative direction
* **User hits Key ‘F’:** rotate about Z axis in the positive direction
* **User hits Key ‘f’:** rotate about Z axis in the negative direction
* **User hits Key ‘[’:** draws color cube
* **User hits Key ‘q’:** quits program
* **User hits Key ‘L’:** scales larger
* **User hits Key ‘l’:**  scales smaller
* **User hits Key ‘S’ or ‘s’:** stops any idle motion
* **User hits Key ‘b’:** prints CTM and reshape matrices to console
* **User hits Key ‘U’:** increases idle speed
* **User hits Key ‘u’:** decreases idle speed to no less than 0.5
* **User hits Key ‘L’:** turns on idle mode – objects will rotate or translate continuously
* **User hits Key ‘l’:** turns off idle mode – objects will rotate or translate in increments per button press
* **Default:** message noting “not implemented yet” printed to console

**Program Structure**

1. Generic initialization functions are called
2. HW2 specific keyboard and display functions are registered as call backs
3. Glut main loop
4. Many key presses trigger events like loading a new PLY
5. Idle function runs in background for continuous motion (once started)

**New Files**

* Hw2.cpp – primary implementation of homework2
* PLYreader.cpp – utilities for reading PLY files
* PLYPicture.cpp - abstraction of PLY pictures

**Old Files (some have minor additions)**

* **utils.h** – Holds useful functions, macros and struct definitions that will hopefully be useful in later homeworks as well as this one
* **homework.h** – Holds function headers useful for only one homework at a time
* **textfile.cpp** – Given functions for reading ANY text file
* **init.cpp** – generalized initialization functions
* **MyPolyline.cpp** – functions relating to the creation and usage of the MyPolyline struct – which holds the number of points in the polyline as well as a pointer to the array of points in the line
* **MyPicture.cpp -** functions relating to the creation and usage of MyPicture structs, which holds the number of polylines in the picture as well as the array of polylines
* **geometries.cpp** – functions to generate single polyline shapes
* **keys.cpp –** functions used with the keyboard call backs
* **color.cpp –** functions related to coloring
* **frame.cpp –** Functions related to the “Frame” abstraction. A frame is defined as the four qualities used to define the world frame: Left, Right, Bottom, Top. This is an abstraction in process to help with world frame and viewport abstraction
* **GRS.cpp –** Functions related to reading GRS formatted files
* **draw.cpp –** Fully functional drawing functions like Fern. Functions capable of being in a callback function followed by flush.
* **hw1.cpp -**  functions specific to hw1
* **main.cpp –** the main!
* **vshader1.glsl** – My first vertex shader!
* **fshader1.glsl** – my first fragment shader!