24-783 Problem Set 7

Deadline: 04/04 (Wed) 23:59

Preparation: Set up CMake projects and utility libraries

You first create projects for the two problem sets.

In the command line window, change directory to:

~/24783/src/*yourAndrewId*

1. Update the course_files and public repositories.
2. Use "svn copy" to copy glutil.cpp, glutil.h, lattice.h, meshlattice.cpp, and meshlattice.h to your utility directory by typing:

```
svn copy ~/24783/src/course_files/utility/glutil.* utility
svn copy ~/24783/src/course_files/utility/lattice.h utility
svn copy ~/24783/src/course_files/utility/meshlattice.* utility
```

3. Then, add these three source files to the CMakeLists.txt file of your utility library.
4. Also add "geblkernel" in target_link_libraries for the utility library of CMakeLists.txt of the utility library. This is needed to use YsShellExt class.
5. Copy the base code for ps7:

```
svn copy ~/24783/src/course_files/ps7 .
```

6. Add this ps7 sub-directory to your top-level CMakeLists.txt
7. Write CMakeLists.txt for ps7 sub-directories. The project is a graphical application. Therefore use MACOSX_BUNDLE keyword. The project name must be "ps7". Case sensitive. It must link "fslazywindow", "geblkernel", and "utility" libraries.
8. Run CMake, compile, and run ps7 executable.
9. Add all the files you created to the control of svn. Svn-copied files are already under SVN's control, and you don't have to add them. Sub-directories created by mkdir (not "svn mkdir") and files copied or moved by copy, cp, move, or mv commands (not "svn move" and "svn copy") must be added by "svn add" command.
10. Commit to the SVN server.

Optional: Check out your directory in a different location and see if all the files are in the server.

In this assignment, use YsShellExt class instead of PolygonalMesh class. YsShellExt has all public functions that PolygonalMesh has. (Actually, I made PolygonalMesh class compatible with YsShellExt.) Only one thing you need to do is call EnableSearch() function in Initialize function.

PS7 Mesh Painting, Virtual Laser Range Scanner, and Slices

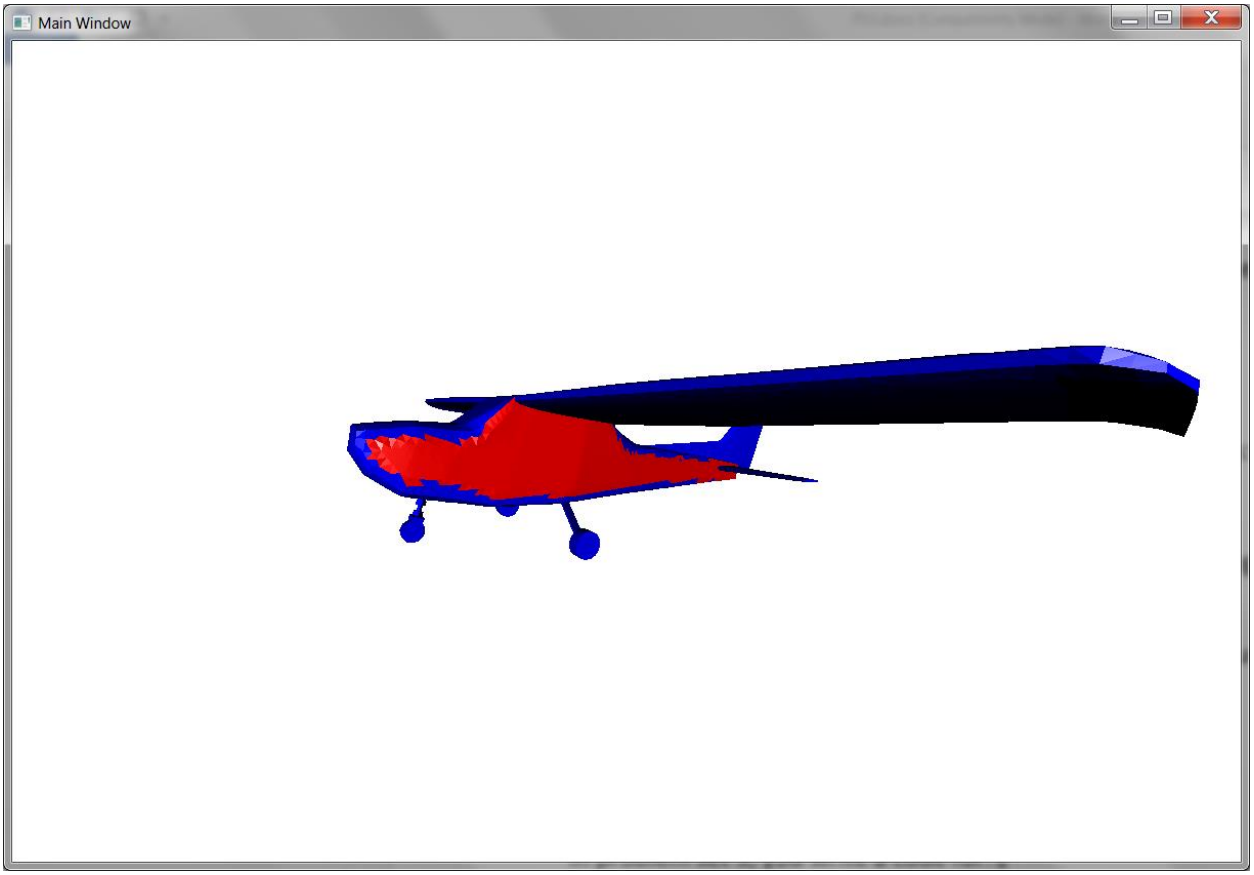
In problem set 6, you write a code for:

(1) Traversing and painting neighbor polygons with similar normal vectors.
(2) Visualizing slices of an STL geometry with constant-y planes.
(3) Simulating and visualizing a laser-range scanner.

Your program takes three states: painting mode, slice mode, and laser-range scanner mode. Therefore, you need to add a member variable of the application class to remember the program state, and your program must behave differently depending on the state.

You start your program with a STL file name as a command-line parameter. Initially, the program is in the painting mode, and the initial colors of the STL model must be all blue.

In the painting mode, when the user clicks on a blue polygon, your program must paint red neighboring polygons whose normal vector make within 20 degrees of the clicked polygon. Your program must start traversal from the clicked polygon, and must not step into a polygon whose normal vector makes greater than 20 degrees of the clicked polygon.



In the painting mode, after clicking a polygon on a side of the Cessna 172.

When the user presses the S key, the program must calculate the slices (intersection between the triangles and a plane) of the STL model, and change the state to the slice mode. Slicing planes are constant-y planes, which evenly divides the range between minimum-y and maximum-y of the STL model into 100 slices. Since an STL is all-triangular mesh, you only need to consider triangle-plane intersections, each of which can give up to 1 line segment. (i.e., Your program must calculate a vertex array for GL_LINES.) In the slice mode the program must only show the slices in black lines. No triangles.

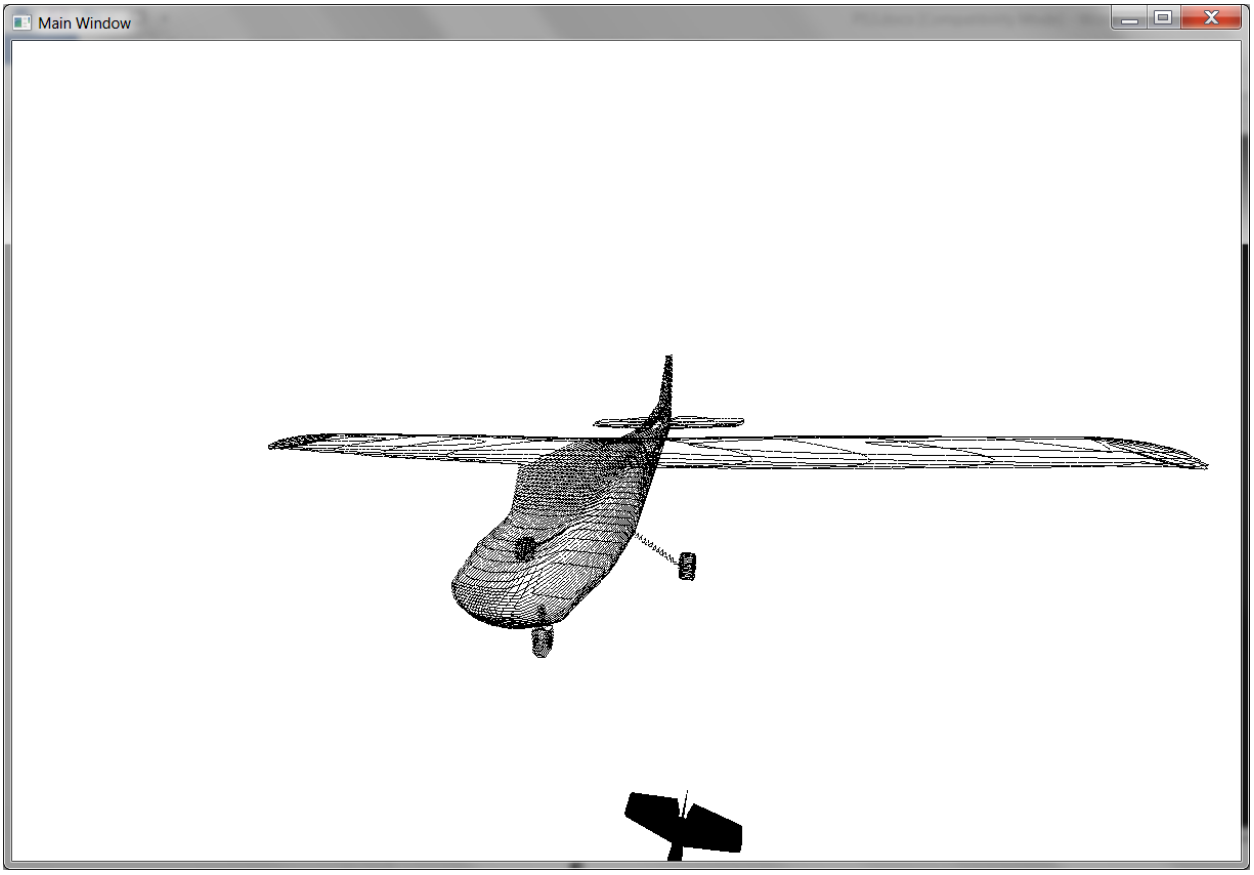
Before starting the calculation of the slices, you should do:

```
auto t0=FsSubSecondTimer();
```

to record the starting time, and as soon as the calculation is done, do:

```
printf("%d milli seconds\n",(int)FsSubSecondTimer()-t0);
```

to show the number of seconds it took to calculate the slices. If your program takes more than 2 seconds for the slice calculation with c172r.stl, you lose 5 points.



Slice mode

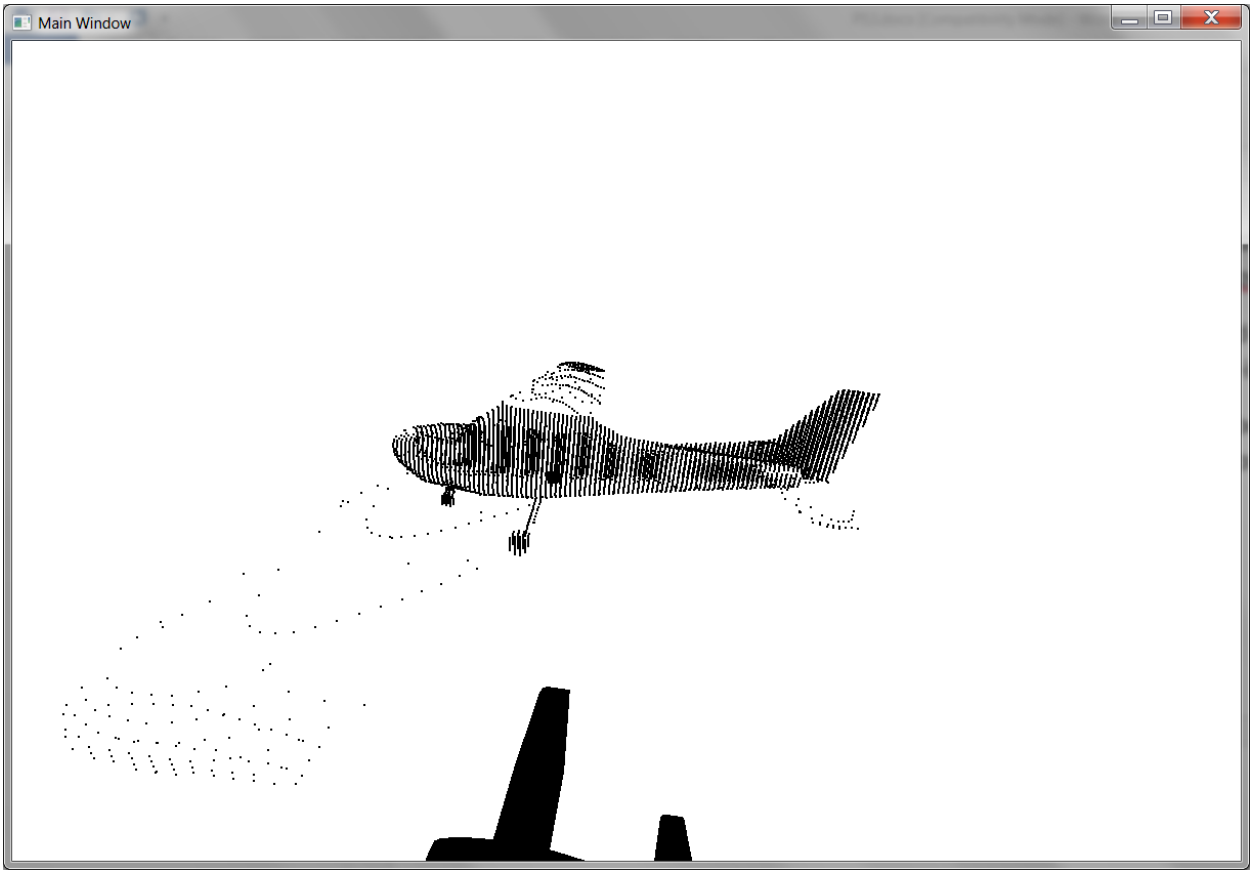
When the user presses the L key, your program must calculate intersections between the lines that are parallel to the Z axis, evenly-spaced 100 lines in each X- and Y-directions lying between minimum-XY and maximum-XY of the STL model (10000 lines in total).

Then your program must go into the laser-range scanner mode and show the intersecting points as black dots on the graphics window. Use GL_POINTS with glPointSize(4).

Actual laser-range scanner shoots a laser from one point, and will not be able to see occluded points, but for this assignment, you shoot parallel rays, and show all intersecting points.

Since all rays are parallel to the Z-axis, you can use a lattice to accelerate the calculation. In this case, you only need 1 box in the Z-direction.

Similar to the slice mode, your program must show the number of seconds needed to calculate all intersections. You will lose 5 points if your program takes more than 2 seconds to calculate intersections with c172r.stl.



Range-Scanner mode

IMPORTANT

There are two Cessna models in the data subdirectory. One is smaller (in terms of triangle count), one is larger. You can use smaller one for testing purpose, but measure the performance of your program with the larger one c172r.stl. Not cessna.stl.

Have you added MACOSX_BUNDLE keyword for your graphical projects?

Test your .CPP and .H files with the compiler server

Your final version code must pass the compiler server. See instructions on the Canvas.

Commit your files to the server.

Also, **check out** your submitted files in **a different location** and make sure all files are submitted, and the file contents are the ones you wanted to be graded.