24-783 Problem Set 5

Deadline: 03/20 (Mon) 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 Polygonal-Mesh class files (polygonalmesh.cpp, polygonalmeshio.cpp, and polygonalmesh.h) to your utility directory by typing:
   ```
   svn copy ~/24783/src/course_files/utility/polygonalmesh* utility
   ```
3. Then, add these three source files to the CMakeLists.txt file of your utility library.
4. Also add "ysclass" in target_link_libraries for the utility library of CMakeLists.txt of the utility library.
5. Copy the skeleton code for the NACA 4-digit air foil by typing:
   ```
   svn copy ~/24783/src/course_files/ps5/naca .
   ```
6. Add this naca sub-directory to your top-level CMakeLists.txt
7. Create a sub-directory called:

   ps5

   and then, inside ps4 create sub-directories:

   ps5_1
   ps5_2

   File/Directory names are case sensitive. Use underscore. NOT hypen.
   The directory structure under your SVN directory is important. The grading script expects that the directory structure under your SVN directory is:
   ```
   ps5
       ps5_1
       ps5_2
   utility
   ```
   (By the way, don't delete other sub-directories you created for the earlier assignments.)
8. Copy main.cpp of the binary-STL visualizer:

   ~/24783/src/course_files/binaryStlViewer/main.cpp

   to ps5_1 and ps5_2 directories. (I suggest to start from the binaryStlViewer example, but if you are more comfortable to start with another base code, that's up to you.)
9. Write CMakeLists.txt for ps5_1 and ps5_2 sub-directories. The project is a graphical application. Therefore use MACOSX_BUNDLE keyword. The project name must be "ps5_2". Case sensitive and use underscore. Do not use hyphen. It must link "fslazywindow", "naca4digit", and "utility" libraries.

10. Modify top-level CMakeLists.txt so that your build tree includes utility, naca, and ps5/ps5_1 and ps5/ps5_2 sub-directories.
11. Run CMake, compile, and run ps5_1 and ps5_2 executables.
12. 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.
13. Commit to the SVN server.
14. Optional: Check out your directory in a different location and see if all the files are in the server.

PS5-1 Generating a NACA 4-digit airfoil (50 points)

In this problem, you write a function to generate a NACA 4-digit airfoil cross-section and visualize it. Write a function in naca4digit.cpp so that it takes 4-digit number that defines the NACA 4-digit air foil and the resolution in Z direction. The function must return a polygon of the airfoil lying on the plane x=0.

Use the equation in:

https://en.wikipedia.org/wiki/NACA_airfoil

It is not difficult to calculate the airfoil at all. For this assignment, you can assume that the curvature of the camber is small. Since the airfoil must lie on the ZY plane, replace x in the equation with z. You can just apply the formula in the web site: $y(z)=y_c(z) \pm y_t(z)$. If the sampling interval dz=0.02, sample y(z) at z=0, 0.02, 0.04, …, 1.0.

If you strictly interpret the airfoil definition, y_t is measured perpendicular to the camber line, but if we assume that the curvature of the camber line is small, we can just add y_t and y_c to get a reasonably good approximation.

The function must return a std::vector <YsVec3> that represents a polygon.

Fill the function:

```
std::vector <YsVec3> MakeNACA4DigitAirfoil(int naca,const double dz)
```

written in naca4digit.cpp

Then, modify main.cpp so that the program takes two parameters as:

```
ps5_1 aaaa dz
```

The first parameter "aaaa" must be the four digit that defines the airfoil. The second parameter "dz" should be the resolution that must be passed to MakeNACA4DigitAirfoil function.

In Initialize function, when the user gives these two parameters, create and store the airfoil (add a member variable for this purpose), remove call to the LoadBinarySTL. The dimension of the airfoil is about 1.0 (because 0<z<1.0), and the center is near (0,0,0.5). Set up target-camera distance d appropriately.

In Draw function, draw a wireframe of the airfoil.

For this part, you do not have to use mesh, nom, col, and bbx. (You can use to use some of them if it is easier for you). You can remove unused variable (which is recommended), but leaving them will not be a basis for taking points off.

PS5-2 Extruding a NACA 4-digit airfoil cross-section to create a wing, and save as a binary STL. (50 points)

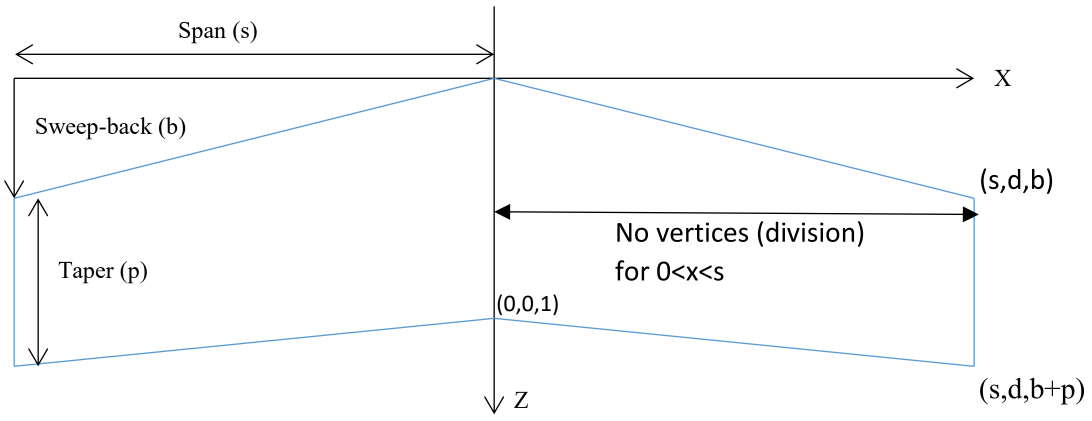
In PS5-2, you write a function that creates a wing geometry by extruding the cross section of a NACA 4-digit airfoil, and then saves it as a binary-STL file.

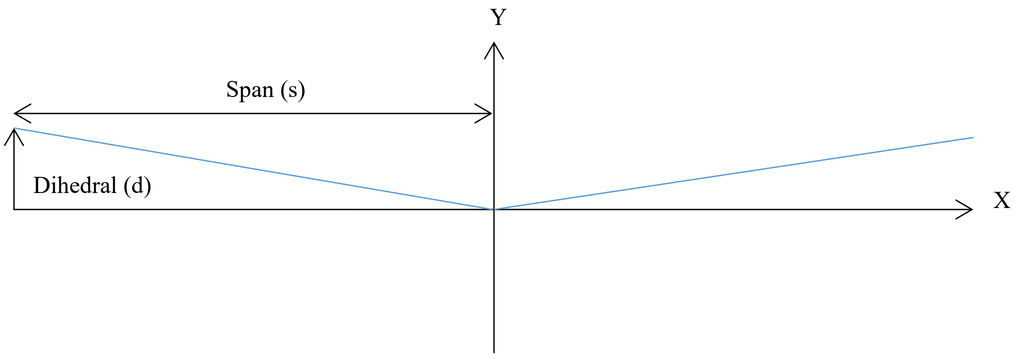
Your program takes 6 parameters as:

```
    ps2_YourAndrewID aaaa s b d p dz
```

The first parameter (argv[1]), aaaa is a number that defines the NACA 4-digit airfoil. The second parameter (argv[2]) s is the wing span, and the third parameter (argv[3]) b is sweep-back by which the wing cross section at x=±s is displaced in z direction. The fourth parameter (argv[4]) d is dihedral by which the wing cross section at x=±s is displaced in y direction. The fifth parameter (argv[5]) p is taper at which the wing cross section at x=±s is uniformly scaled from the cross section at x=0. The last parameter (argv[6]) dz is the sampling interval in the z coordinate. The airfoil section must be sampled at 0, dz, 2*dz, 3*dz, …., 1.0. The last z should be rounded off to 1.0. (Use MakeNACA4DigitAirfoil function written in PS5-1 to calculate the cross-section).

Use the chord length of one, therefore the leading edge is at (0,0,0), and the trailing edge is at (0,0,1) at x=0. The leading edge at x=s will be at (s,d,b), and because the cross section is uniformly scaled by p, the chord length at x=s will be p, and therefore the trailing edge will be at (s,d,b+p) at x=s.





To create the wing surface, first calculate the cross section at x=0. Then, scale and translate this cross section to get cross sections at x=±s. After doing this, you have three cross sections at x=0, x=-s, and x=+s. Then you can create polygons by connecting the three cross sections. For each pair of the edge segment, you can make a quadrilateral. To later save it as an STL, you need to split it into two triangles.

You do not have to divide polygons between x=0 and x=-s and between x=0 and x=+s. (Some students last year thought there must be reasonably good resolution in X direction and divided it into smaller pieces, but you don't have to.)

The wing geometry needs to be geometrically closed except at the wing tip (x=±s). In other words, you can leave the wing geometry open at x=±s. The normal vectors of the triangles must point outward. The order of the vertices of each triangle must appear counter-clockwise when looked from the outside of the wing. (Think how you can verify this. You can add some extra functionalities in your program for this purpose.)

By the way, it is not difficult at all to write your own cross-product function, but you can also calculate a cross-product of two vectors in YS-Class library as:

```
YsVec3 v1,v2;

YsVec3 crs=YsUnitVector(v1^v2);
```

Operator ^ for YsVec3 class calculates cross-product.

Write this process in function

```
void MakeNACA4DigitAirfoilWing(class PolygonalMesh &mesh,int naca,float
span,float sweepBack,float dihedral,float taper,float dz);
```

in naca4digit.cpp. This function must create a wing in a PolygonalMesh object that is given as the first parameter by reference. You can choose a color, but avoid too dark or too bright color.

Initialize function in main.cpp of binaryStlViewer calls LoadBinStl function to load the geometry from a binary STL file. In this assignment, you call MakeNACA4DigitWing function instead. Then the rest should be the same, i.e., the view target and target-view distance must be calculated based on the dimension of the geometry.

In Initialize function, immediately after creating the wing geometry, save the geometry in the binary-STL format. The file name must be "wing.stl", and it must be in the current working directory. Write PolygonalMesh::SaveBinStl function in polygonalmeshio.cpp, and use it from main.cpp.

For this assignment, you can assume the byte-order of the CPU is little-endian, that is, you can directly give a pointer to the variables to fwrite function without reversing the order of the bytes. Add a member function write the function

```
bool PolygonalMesh::SaveBinStl(const char fn[]) const
```

in polygonalmeshio.cpp, and use this function to save a file.

Sample image (Generated with parameters: `2312 3 1 1 0.2 0.1`)