24783 Advanced Engineering Computation: Problem Set 8

(*) In the following instruction (and in all of the course materials), substitute your Andrew ID for where you see *yourAndrewId*.

START EARLY!

1 Check Out or Update Base Code and Libraries

Please make sure you have up-to-date libraries and course files before starting an assignment.

If you have not done working-directory set up as described in the first assignment (like in case you need to work from a different computer), please see Problem Set 1 and set up the working directory.

I assume you created the working directory called *24783* under you home directory and you checked out your Git repository in there.

Home directory is typically *C:\Users\username* in Windows, */Users/username* in macOS, and */home/username* in Linux, where *username* is the user name in your local computer.

First, open command-line (Developer PowerShell or Terminal), and move to your working directory by typing:

```
cd ~/24783
```

You need to check out (or clone) Git repositories once. If you have not checked out yet, do the following:

```
git clone https://yourAndrewId@ramennoodle.me.cmu.edu/Bonobo.Git.Server/course_files.git
git clone https://yourAndrewId@ramennoodle.me.cmu.edu/Bonobo.Git.Server/yourAndrewId.git
```

You need to replace "yourAndrewId" with your Andrew ID. You'll be asked to type in credentials.

Also we are going to use two additional repositories:

```
git clone https://github.com/captainys/MMLPlayer.git
git clone https://github.com/captainys/public.git
```

If you are successful, you should have the following directory structure under your home directory.

```
Your User Directory
├── (Other files and directories)
└── 24783
    └── course_files
```

```
└── yourAndrewID
```

If you already have checked out these repositories (most likely you did for Problem Set 1), you need to update (or git pull) in those repositories. By change directory to the location where you checked out repositries and then type:

```
git pull
```

To update all four repositories, you can type the following commands in a sequence:

```
cd ~/24783/course_files
git pull
cd ~/24783/yourAndrewID
git pull
cd ~/24783/public
git pull
cd ~/24783/MMLPlayer
git pull
```

2 Copy Base Code and Add to Git's Control

Copy ps8 subdirectory from course_files to your directory. The directory structure must look like:

```
Your User Directory
├── (Other files and directories)
└── 24783
    ├── public
    ├── course_files
    └── yourAndrewID
        └── ps8
```

3 Make a CMake Project

In this problem set, you write three CMakeLists.txt scripts. One is the top-level CMakeLists.txt, which needs to enable C++11 features, include public libraries, and include ps8_1 and ps8_1 sub-directories.

The other two are for two sub-directories. CMakeLists.txt in ps8_1 sub-directory needs to have a library called dha which uses dha.h and dha.cpp and links geblkernel library, and an executable target called ps8_1 which uses main.cpp, glutil.cpp, and glutil.h, and links dha, geblkernel, and fssimplewindow libraries.

CMakeLists.txt in ps8_2 sub-directory needs to have a library called astar which uses astar.h and astar.cpp and links geblkernel library, and an executable called ps8_2 which uses main.cpp, glutil.cpp, and glutil.h, and links astar, geblkernel, and fssimplewindow libraries.

4 Dihedral Angle Based Segmentation

In this assignment, you write a code that creates groups of polygons separated by low dihedral-angle (sharp) edges. Your program must take two parameters from the command line for example:

```
ps8_1 c172r.stl 30
```

The first parameter is the STL file to be loaded, and the second parameter is the angle threshold in degrees.

Your program must then open the STL file and:

- Create a segmentation (groups of polygons) separated by edges that the dihedral angle of the two polygons sharing the edge is greater than the user-specified angle threshold. The polygons must be drawn white (unless you do 10 points bonus).
- Draw group boundaries in blue lines. Use glLineWidth(4).
- Optionally, paint polygons so that polygons within the same group have the same color, and the polygons of the neighboring group always have a (visibly) different color. (10 points bonus).

Dihedral angle can be calculated between two polygons. You use GetNormal function to get two normal vectors, and use operator* to calculate the cosine of the dihedral angle.

To create groups, you start traversal from each polygon that is not yet in a group. Create a temporary group that consists of only one polygon, and then grow the group by visiting edge-connected polygons while growing a group. However, if the dihedral angle between the current polygon and a neighboring polygon is greater than the user-specified threshold, do not grow the group into the neighboring polygon. Also you must not grow the group into a polygon that is already in a group. The traversal must stop when the group can no longer grow.

For this assignment, you implement two functions (if you go for the bonus question, three functions) in dha.cpp.

```
std::unordered_map <YSHASHKEY,int> MakeDihedralAngleBasedSegmentation(
  const YsShellExt &mesh,const double dhaThr);

std::vector <float> MakeGroupBoundaryVertexArray(
  const YsShellExt &mesh,const std::unordered_map <YSHASHKEY,int> &faceGroupInfo);

void MakeFaceGroupColorMap(
  YsShellExt &mesh,const std::unordered_map <YSHASHKEY,int> &faceGroupInfo)
```

Data type YSHASHKEY is an unsigned integer. It is possible to specialize std::hash for YsShellExt::PolygonHandle, however, because std::hash already has an implementation for unsigned integer, it is easier to use a vertex search key. Use GetSearchKey to get a search key from a polygon handle, and FindPolygon to get a polygon handle from a search key. You can add your own functions in dha.cpp if you feel necessary. However, these three functions must be in the prototypes as defined in dha.h. Also the library "dha" may be tested by a different test program. If your own functions are used from these three functions, write it in dha.cpp.

The first function, MakeDihedralAngleBasedSegmentation, takes a mesh and dihedral-angle threshold as input parameters. The return value is an unordered_map from a polygon search key to a segment (face-group) identifier. Segmentation is equivalent to giving labels to the polygons. If two polygons have an identical label, the two polygons belong to the same group. The second function, MakeGroupBoundaryVertexArray, takes a mesh and the segmentation created by MakeDihedralAngleBasedSegmentation as input, and returns a vertex array of the face-group boundaries that can be drawn as GL_LINES.

If you go for the 10-point extra credit, you implement the third function, MakeFaceGroupColorMap. In this function, polygons must be painted by SetPolygonColor, so that (1) all polygons in the same face group has an identical color, and (2) no two neighboring face-groups (two face-groups that shares at least one edge piece) are painted in the same color.

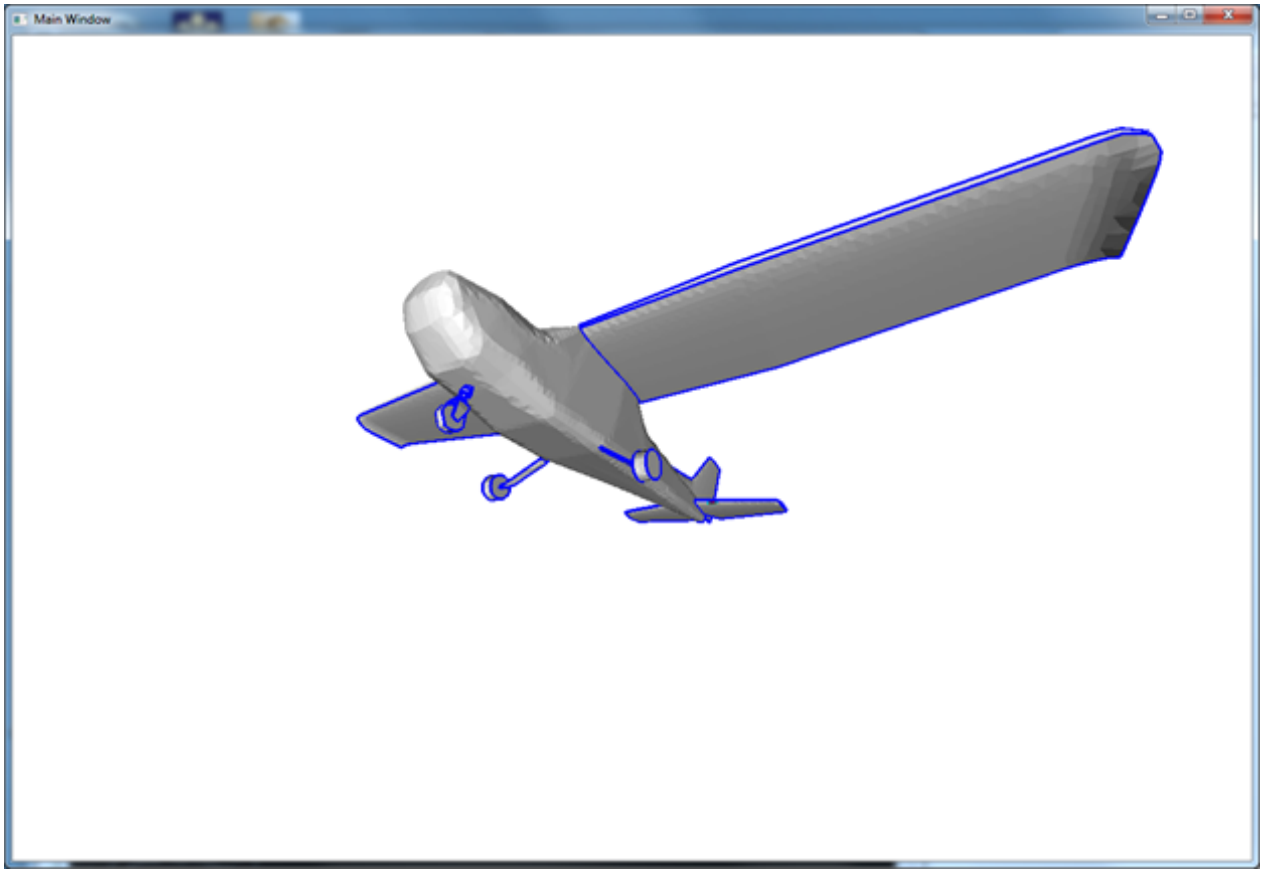

Fig. 1: Segmentation Example

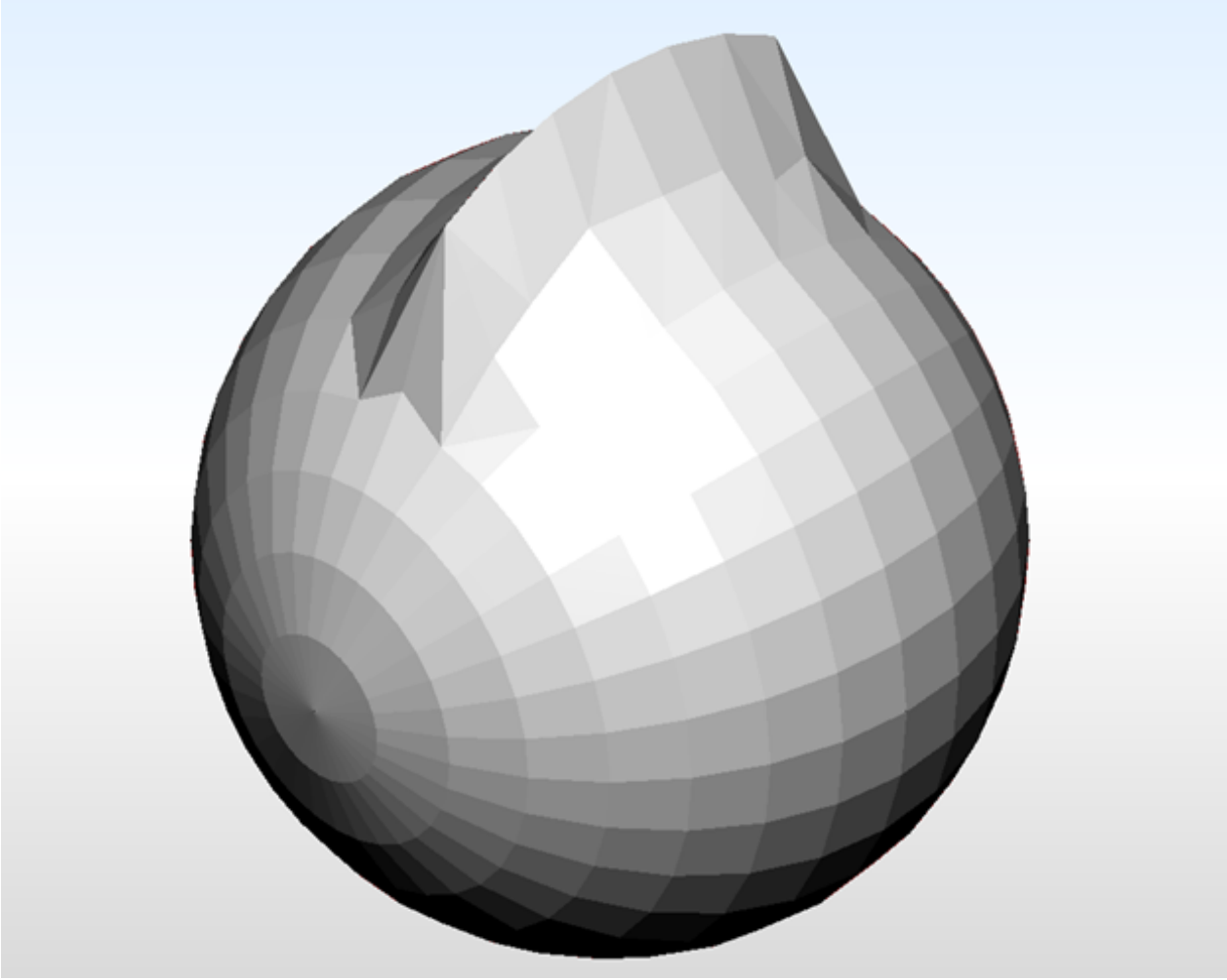

Fig. 2: In this case, low-dihedral angle edges does not separate polygons into two groups.

5 A-Star Path Finding

A* (A-star) is a very popular path-finding altorighm. You can find a very detailed explanation of the algorithm on Wikipedia. https://en.wikipedia.org/wiki/A*_search_algorithm.

You write a function (actually two functions) to find a shortest path connecting between two vertices on YsShellExt.

You implement two functions in astar.cpp:

```
std::vector <YsShellExt::VertexHandle> A_Star(
  const YsShellExt &shl,
  YsShellExt::VertexHandle startVtHd,
  YsShellExt::VertexHandle goalVtHd);

std::vector <YsShellExt::VertexHandle> A_Star_ReconstructPath(
  const YsShellExt &shl,
  const std::unordered_map <YSHASHKEY,YsShellExt::VertexHandle> &cameFrom,
  YsShellExt::VertexHandle current);
```

You can add your own functions in astar.cpp if you feel necessary. However, these two functions must be in the prototypes as defined in astar.h. Also the library "astar" may be tested by a different test program. If your own functions are used from these three functions, write it in astar.cpp.

Data type YSHASHKEY is an unsigned integer. std::unordered_map, std::unordered_set can be used for closedSet, g_score, and f_score. You can use YsAVLTree class, which is compatible with the AVL tree class you wrote for the assignment for keep vertices sorted in the order of f_score. You also need a map from a vertex to the AVL-tree node because in each iteration f_score of a node may change, and then the order needs to be updated. (I believe std::priority_queue is not designed for re-ordering based on the updated cost.) It is possible to specialize std::hash for YsShellExt::VertexHandle, however, because std::hash already has an implementation for unsigned integer, it is easier to use a vertex search key. Use GetSearchKey to get a search key from a vertex handle, and FindVertex to get a vertex handle from a search key.

The pseudocode from Wikipedia suggest that you should add neighbor to openSet if neighbor is not in openSet during the iteration. However, if we use an AVL tree as openSet, you need to wait until you have fScore for the vertex until you can add it to openSet. Also, when you update the score of the neighbor, if the neighbor is already in the AVL tree, the tree node for the neighbor must first be deleted, and then re-inserted. Therefore, you need to re-organize the pseudocode to translate into the working program.

The heuristic cost estimate can just be a straight-line distance between two points. If you implement correctly, when you build and run the executable ps8_2 with an STL file as the first command-line parameter, you will see three line strips that connects vertices of minimum x and maximum x, minimum y and maximum y, and minimum z and maximum z. Some models do not have such paths, for example c172r.stl does not have a path connecting max-y and min-y vertices because front wheel is disconnected from the fuselage.

```
  Pseudocode from Wikipedia.org
  (https://en.wikipedia.org/wiki/A*_search_algorithm captured 03/31/2017)
function A*(start, goal)
    // The set of nodes already evaluated.
    closedSet := {}
    // The set of currently discovered nodes that are not evaluated yet.
    // Initially, only the start node is known.
    openSet := {start}
    // For each node, which node it can most efficiently be reached from.
```

```
    // If a node can be reached from many nodes, cameFrom will eventually contain the
    // most efficient previous step.
    cameFrom := the empty map

    // For each node, the cost of getting from the start node to that node.
    gScore := map with default value of Infinity
    // The cost of going from start to start is zero.
    gScore[start] := 0
    // For each node, the total cost of getting from the start node to the goal
    // by passing by that node. That value is partly known, partly heuristic.
    fScore := map with default value of Infinity
    // For the first node, that value is completely heuristic.
    fScore[start] := heuristic_cost_estimate(start, goal)

    while openSet is not empty
        current := the node in openSet having the lowest fScore[] value
        if current = goal
            return reconstruct_path(cameFrom, current)

        openSet.Remove(current)
        closedSet.Add(current)
        for each neighbor of current
            if neighbor in closedSet
                continue // Ignore the neighbor which is already evaluated.
            // The distance from start to a neighbor
            tentative_gScore := gScore[current] + dist_between(current, neighbor)
            if neighbor not in openSet // Discover a new node
                openSet.Add(neighbor)
            else if tentative_gScore >= gScore[neighbor]
                continue // This is not a better path.

            // This path is the best until now. Record it!
            cameFrom[neighbor] := current
            gScore[neighbor] := tentative_gScore
            fScore[neighbor] := gScore[neighbor] + heuristic_cost_estimate(neighbor, goal)

    return failure

function reconstruct_path(cameFrom, current)
    total_path := [current]
    while current in cameFrom.Keys:
        current := cameFrom[current]
        total_path.append(current)
    return total_path
```

6 Test Your Code on the Compiler Server

Test your source files (.cpp and .h files) on the compiler server. Some assignment may not require .h files. You do not have to test files that you don't make modifications. The files you need to test are the ones you write or modify.

We have four compiler servers:

- http://freefood1.andrew.cmu.edu:24780
- http://freefood2.andrew.cmu.edu:24780
- http://freefood3.andrew.cmu.edu:24780
- http://freefood4.andrew.cmu.edu:24780

Make sure you don't see red lines when you select your files and hit "Compile Test" button on the server.

We have multiple servers to make it less likely that all of them need to shut down for maintenance. If do not have to test on all of the servers. You need to make sure that your code passes on one of the servers.

7 Submit

Lastly, you need to submit using git. What you need to do are two things: (1) add files to git's control, and then (2) send to the git server.

7.1 Add Files to git's control

In this case, you want to add all the files under ps8 subdirectory. To do so, type:

```
git add ~/24783/yourAndrewID/ps8
```

This command will add ps8 directory and all files under the subdirectories.

7.2 Send to the Git Server

In Git, sending files to the server is a two-step process. The first step is local commit. You can do it by:

```
git commit -m "Problem Set 8 solution"
```

The message can be anything, but it is recommended to type something meaningful, at least you can see what changes you made to your repository.

Local commit is just local. Git server does not know about any local commit unless the commit is sent (or pushed) to the server. To do so, type:

```
git push
```

Make sure to do it in the CMU network. If you are working from home (probably most likely), use VPN to connect to the CMU network.

You can re-submit (commit and push) your solution as many times as you want with no penalty before the submission due.

8 Verification

It is recommended to clone your repository to a different location and make sure that all of your files have been sent to the Git server.

You can do the following:

```
cd ~
mkdir 24783Verify
cd 24783Verify
git clone https://yourAndrewID@ramennoodle.me.cmu.edu/Bonobo.Git.Server/yourAndrewId.git
```

Once you made sure all the files have been submitted, you can delete files and directories under 24783Verify directory.