# Overview

This is a documentation for building an all-hex mesh from a given triangle mesh corresponding to the work [1]. It uses Centroidal Voronoi Tessellation (CVT) to segment the input surface into patches based on the surface normal information, and then build a polycube with each face mapping to a segmented surface patch. This work is suitable for surface models whose surface normal are roughly perpendicular to the six axis directions, i.e., , , and .

# Compiling Codes

Compiler: Visual Studio (any version should work)

External Libraries:

* Eigen (<http://eigen.tuxfamily.org/index.php?title=Main_Page>)

Compiling tool: CMake

* Under the project folder, build three subfolders named “build”, “io” and “src”.
* In “src”, build a subfolder “cpp” and put all the source files in it. The file “CMakeList.txt” and the folder “cmake” are also in “src”.
* Open CMake, and put the path of “src” as the source and that of “build” as the target. CMake will automatically generate a VS project (.sln) in the “build” folder. Make sure all the paths of the external libraries are correct, which can be checked in the files of the folder “cmake”.

# Structure of Codes

Steps of running this code:

1. In “Main.cpp”, give the path and file name to the variable “inputmeshName”;

2. Run the code (the 1st time) and it will output an initial segmentation result in the file “XXX\_output\_initial.vtk”, where “XXX” is a user-assigned prefix. This process assigns a so-called “Cluster ID” to each triangle according to how close its normal aligns with the six axis directions (, , and ). A “Cluster ID” takes integer value from 0 to 5, corresponding to , ,, , , , respectively. Check the initial result and manual adjustment is needed if certain elements intuitively should be assigned with a different “Cluster ID”. If so, create a text file named “XXX\_Output\_SpecialElements.txt” with two columns, where the first lists element IDs and the second is desired Cluster IDs. One example is provided.

3. In “Main.cpp”, go to the definition of the function *polycube.PostProcessing()*, set the global variable “HAS\_SPECIAL\_ELEMENT” to be “1”. Go to the definition of the function *ModifySpecialElement()*, and one will see how the file “XXX\_\_Output\_SpecialElements.txt” will be used.

4. Run the code (the 2nd time) and check the output file “XXX\_output\_CleanUp.vtk” if the desired segmented result is achieved. If so, check the output file “XXX\_paraMapping.vtk” and use it to manually construct polycube. A polycube is a coarse unstructured hex mesh with corner points from “XXX\_paraMapping.vtk”. A point is a corner point if it is shared by three patches. One may also need to select other corners from “XXX\_paraMapping.vtk” to create a valid polycube. Users need to create the connectivity relationship for each cube in the polycube. One example of a manually constructed polycube is provided; see “navair\_paraHex\_torus\_hex.raw”.

5. In “Main.cpp”, go to the definition of the function *polycube.HexMeshParametricDomain()*, and set the global variable “READ\_IN\_PARAHEX\_TORUS” to be “1”. Run the code again (the 3rd time) and the all-hex mesh will be generated in “XXX\_realHex\_hex.vtk”.

Note that the quality of the resulting hex mesh may depend on where the corner points are located. If the quality is not as good as expected, one may change certain corner points to try again.

# Reference

[1] K. Hu, Y. Zhang. Centroidal Voronoi Tessellation Based Polycube Construction for Adaptive All-Hexahedral Mesh Generation. Computer Methods in Applied Mechanics and Engineering, 305:405-421, 2016.