Brief Instruction to setup the machine

Libraries needed to install: CGAL, Boost and SISL

Attention: SISL needs to be installed in 64 bit. 32 bit SISL will NOT be able to compile the mex function in MATLAB.

CGAL package can be downloaded here: https://github.com/CGAL/cgal

Boost package can be downloaded here: http://www.boost.org/users/download/

SISL package can be downloaded here: https://github.com/SINTEF-Geometry/SISL

It is highly recommended to download SISL package using **git** command, as it is observed by me that the downloaded zip file contain minor errors.

In addition, it is also recommended to compile the above packages from the source, so the directories that contain the header files and lib files can be found easily.

Follow the instructions to install CGAL and Boost.

You need to compile SISL in 64 bit in order to successfully compile the mex function in MATLAB. To do that, follow the steps below:

Step 1: extract the SISL package in any directory that you preferred

Step 2: mkdir build ##create a directory named *build*.

Step 3: cd build ## go to the *build* directory

Step 4: ccmake ../ ## enter CMake GUI

Step 5 (Important): in the terminal window, press **t** ## toggle advanced mode

Add the following contents in the CMake settings:

CMAKE_CXX_FLAGS: -O3 –fPIC

CMAKE_C_FLGAS: -O3 –fPIC

(It is capital letter o, not number zero. To change an item, use arrow keys to navigate to the item you want to change, press **ENTER** to start edit, after you finish editing, press **ENTER** again to finish editing)

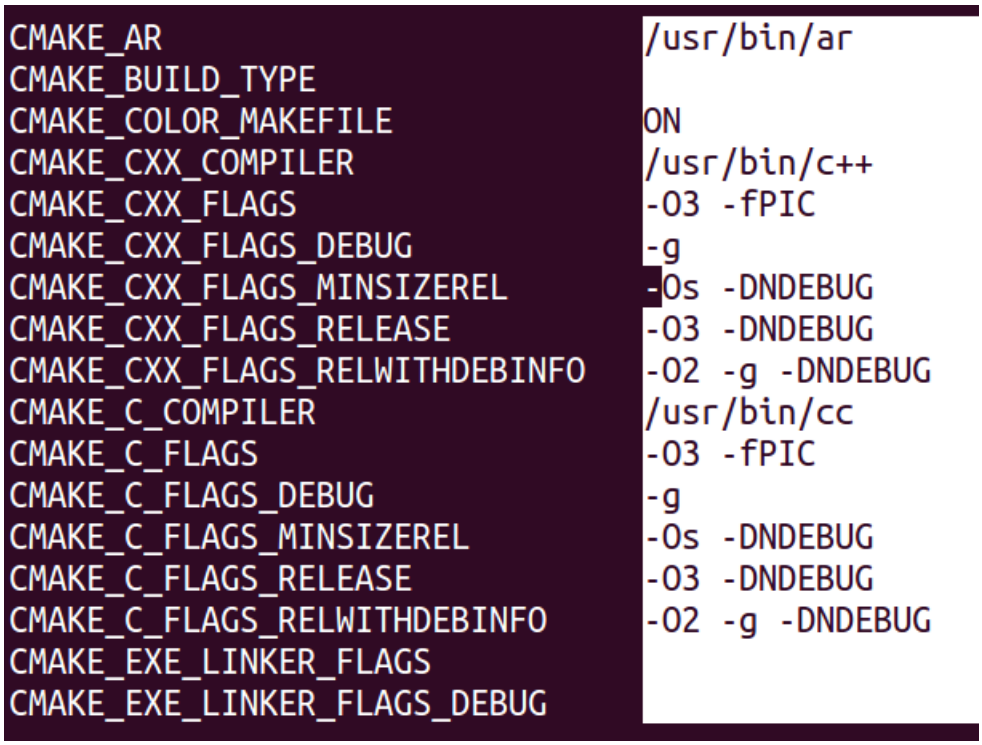

Figure 1. The compiling setting on my machine

Step 6: press **c** ## configure

Press **e** ## to exit if configurations are generated successfully

Press **g** ## to generate CMake files

(if **g** option is not shown, repeat step 6 until you see it. After pressing **g**, you should return to the terminal command mode)

Step 7: make ## to compile

Step 8 (Optional): make install ## checkinstall also works

To compile the mex function in MATLAB

Open MATLAB and past the following command into you MATLAB command window

mex PointLocation.cpp CFLAGS="\$CFLAGS -fopenmp" LDFLAGS="\$LDFLAGS -fopenmp" –I**CGAL_INCLUDE_DIR** –I**BOOST_INCLUDE_DIR** –I**SISL_INCLUDE_DIR** –L**BOOST_LIB_DIR** -lboost_system –L**CGAL_LIB_DIR** –lCGAL –L**SISL_LIB_DIR** –lsisl

Note: SISL_LIB_DIR is the build folder you created while compiling SISL

The syntax to use the Mex function

Output = PointLocation (NURBS_surf, Number_of_particles, A_list_of_patch_size, TestPoints)

There are two inputs you need to call PointLocation function.

NURBS_surf: an array of cells, each cell stores the information of a test particle. To be more specific, each cell of the cell array stores a number of **struct** variables, each of which stores the information of a NURBS patch. Assembling all of the the NURBS patches of a cell of the cell array produces a complete test particle.

Each **struct** variable has a field named "**nurbs**", which is also the **20th** field (starting from 0). The **nurbs** field stores the following information.

- Form ('B-NURBS')
- Dimension (usually 3, the dimension you are working in)
- Number (the dimension of the coefficient matrix)
- Coefficients
- Knots
- Order

(During development, NURBS surfaces were generated by **igesToolBox**. **For your convenience, the zip package is provided along with this instruction. A sample input file can also be found there.**)

Number_of_particles: the amount of particles that needs to be read. Note that this is different from the amount of patches for each particle.

A_list_of_patch_size: this is a $N \times 1$ or $1 \times N$ matrix that tells the MEX function for each test particle how many patches that the particle contains. For example, the sample input has **A_list_of_patch_size** $= [395, 281]$.

TestPoints: a $N \times 3$ matrix that contain a list of points to be tested

$$\begin{bmatrix} \text{x coord} & \text{y coord} & \text{z coord} \\ \dots & \dots & \dots \end{bmatrix}$$

In MATLAB, if you are used to define point in following way,

Point = [xcoord; ycoord; zcoord]

Then you will have to transpose the matrix.

Output: for each point, a number will be returned.

0: the point is not inside any of the input NURBS surface

1: the point is on the surface of one of the input NURBS surface

2: the point is inside one of the input NURBS surface

During development, only 3 dimensional space is tested. The code is not guaranteed to work in dimensions greater than 3. The sample NURBS surfaces can be generated by Octave NURBS package (http://octave.sourceforge.net/nurbs/index.html).

If you have any additional questions, send email to mfeng9@illinois.edu

Document created on 12/15/2015