F21 Research: Marching Cube and Intersections

-Finding a method to make a clean cut on a metaball-

Department: Mechanical Engineering (CERLAB)

Advisor: Dr. Soji Yamakawa

Researcher: Jae seok Oh

Implementation of Marching Cube in 2D and 3D

Before getting to the problem, I implemented the Marching Cube in 2D and 3D to better understand the source of the issue we are dealing with. Marching Cube is used to extract the Iso surface (boundary in 2D) of a certain volume (area in 2D). They are particularly useful in merging multiple solid models as they do not suffer from numerical instabilities but since the algorithm depends on linear interpolation, sometimes it could be troublesome to extract some features such as sharp edges.

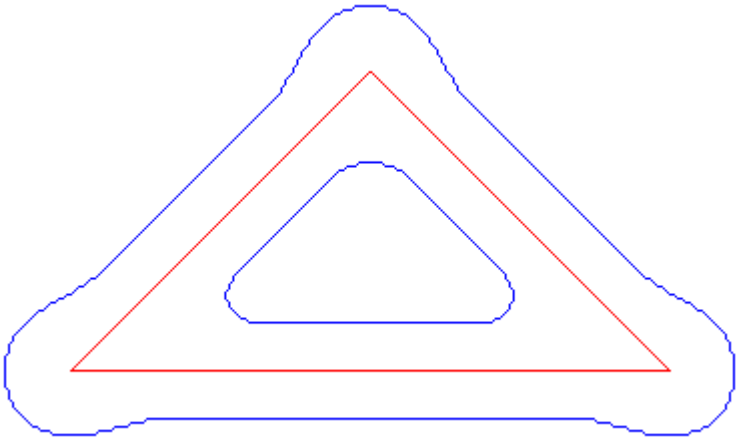

Figure 1. Sample in 2-D: Function defined as set distance from a triangle. Blue line represents the extracted iso boundary.

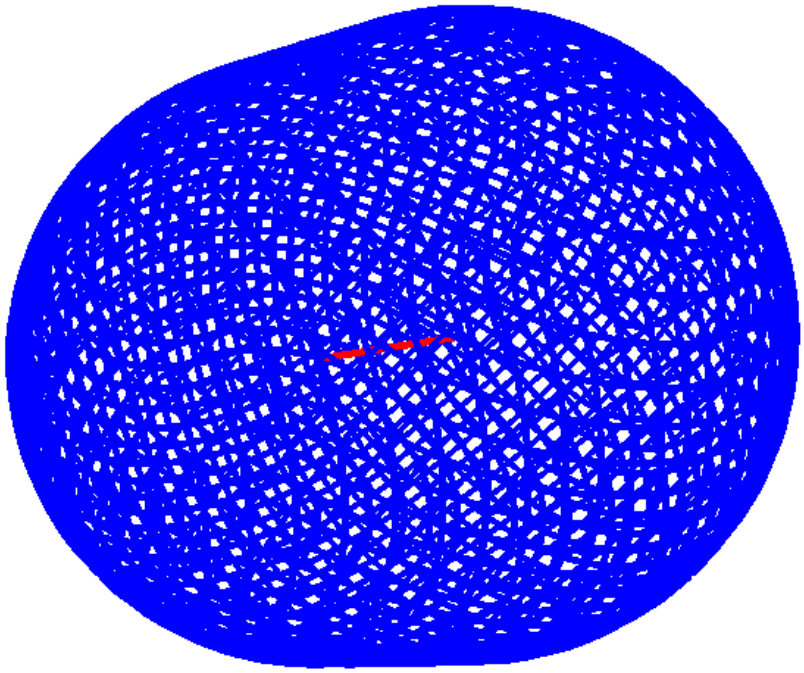

Figure 2. Sample in 3-D: Function defined as set distance from a line (red). Blue line represents the extracted iso surface.

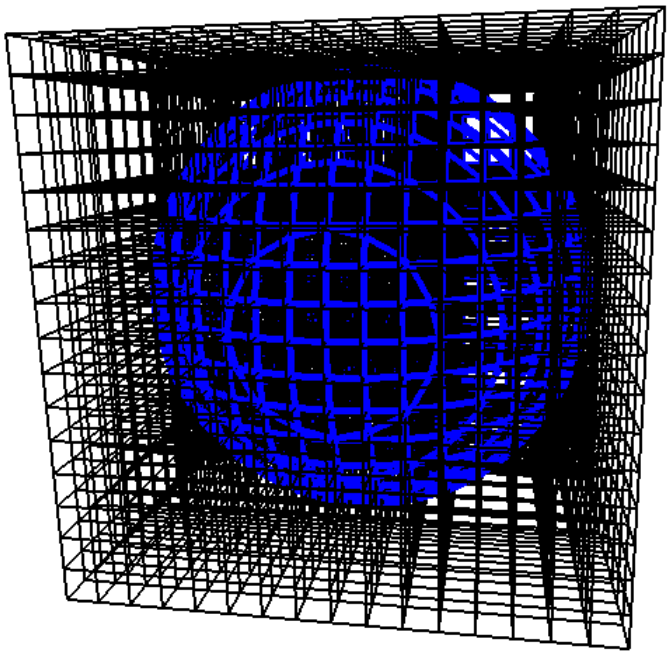

Figure 3. Visualization of marching cubes and the resulting iso surface in 3-D.

Saving and loading metaball information

The program can extract and render a surface of a 3d volume from given points (lines), but it can also save the vertex information into multiple polygons in obj and be rendered in a mesh viewer program [*Fig. 4*]. Conversely, it can also run marching cubes on a given information of function values in 3-D grid and cube dimensions, and render the iso surface [*Fig. 5*].

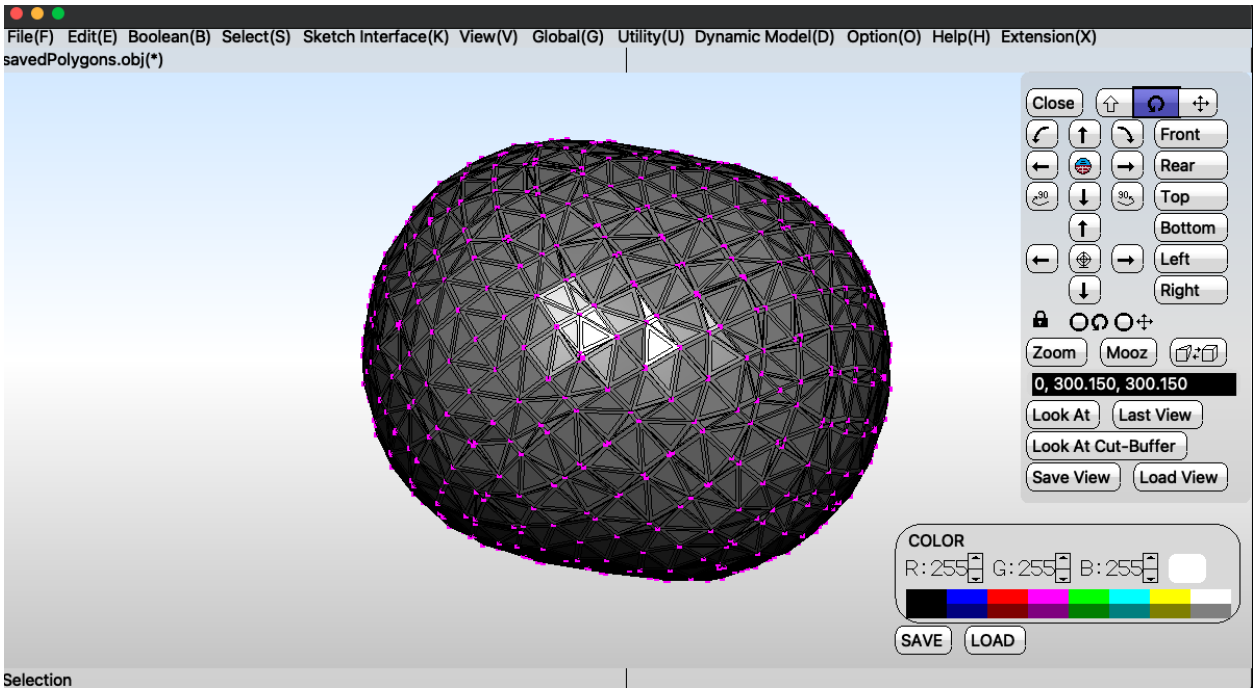

Figure 4. Saved Metaball in obj file and rendered in professor's mesh program.

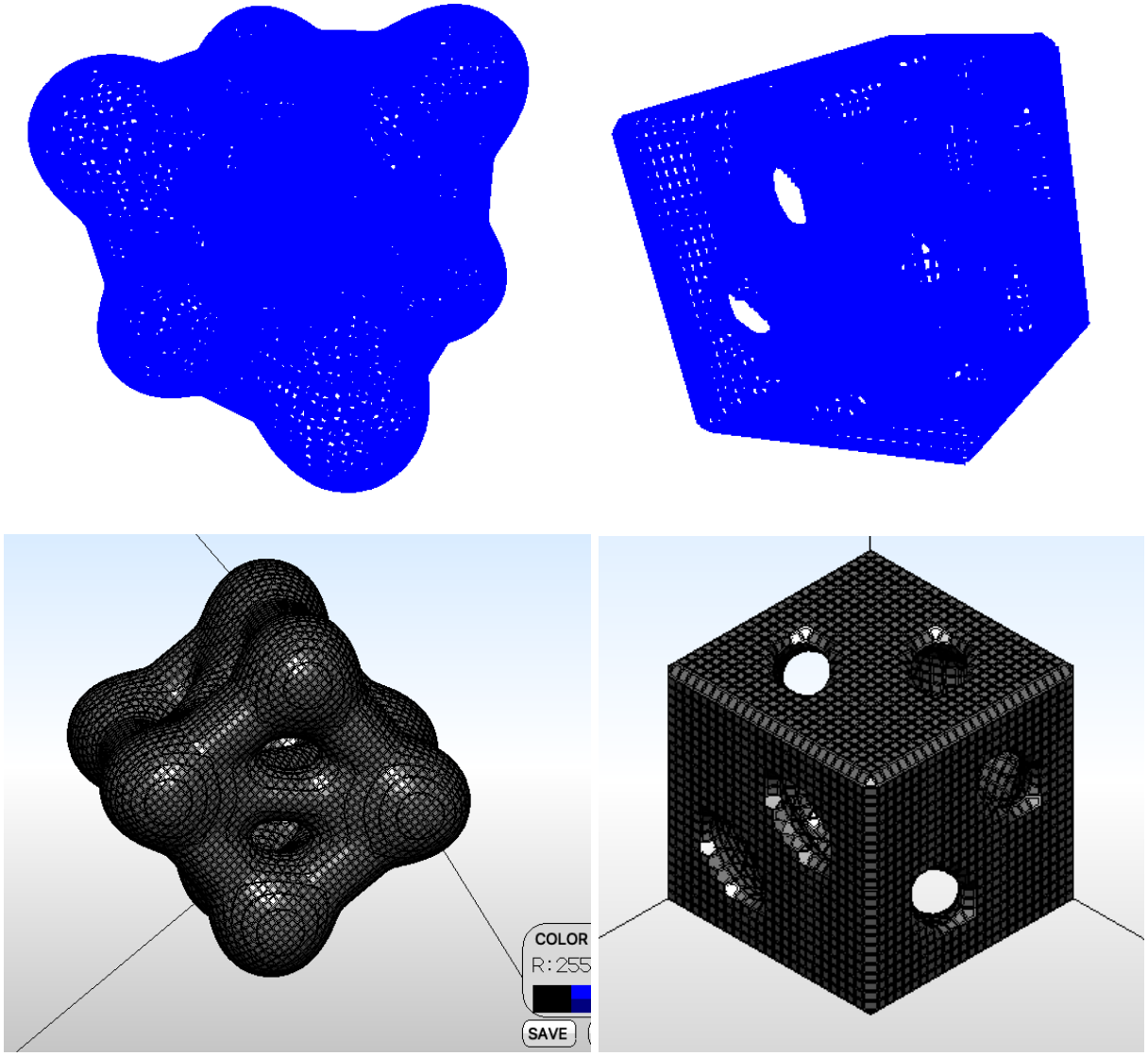

Figure 5. Loaded Metaball iso surface extracted from given input file in txt. Blue lines rendered in my program (above). Gray mesh rendered in professor's mesh program from saved obj file.

Giving edge information to retrieve lost boundaries

As in the figure below, because the marching cube linearly interpolates along the edges of each cube to find the points on a surface, it loses the sharp boundaries or edge information [*Fig. 6*]. This problem is very close to the original goal of the research: finding a method to save sharp features (edges). Explicitly giving inputs of these

boundaries and additionally adding intersection points between the boundary lines and the cubes' planes can partially retrieve the information in simple cases [*Fig. 7*].

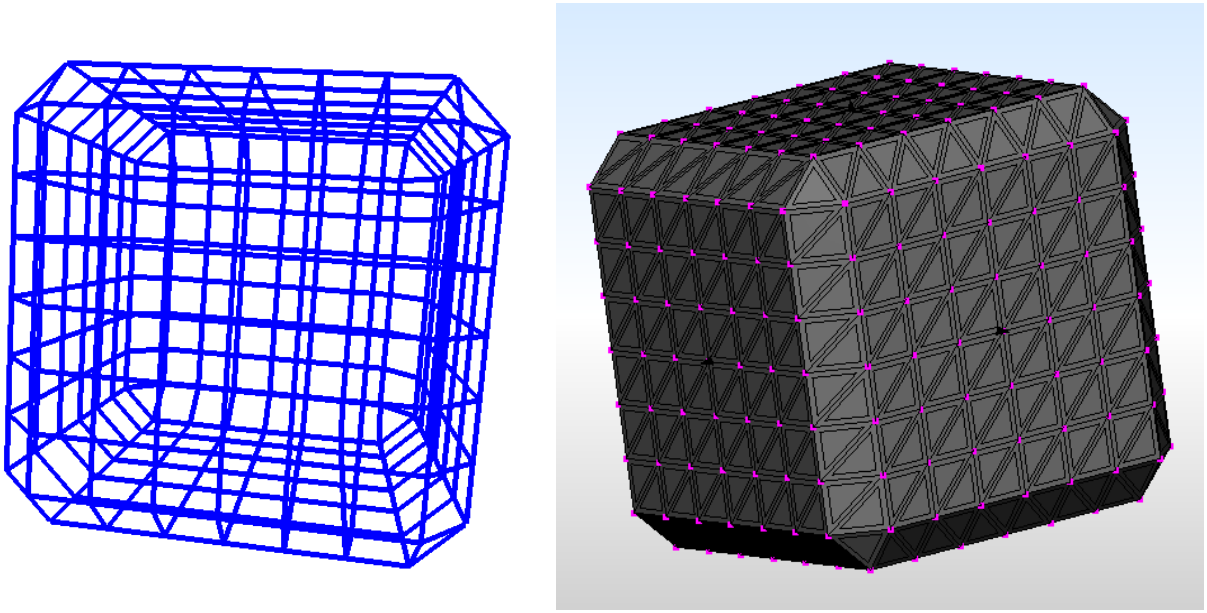

Figure 6. The surface should represent 3-D cube but the boundaries are cut-off and chamfered.

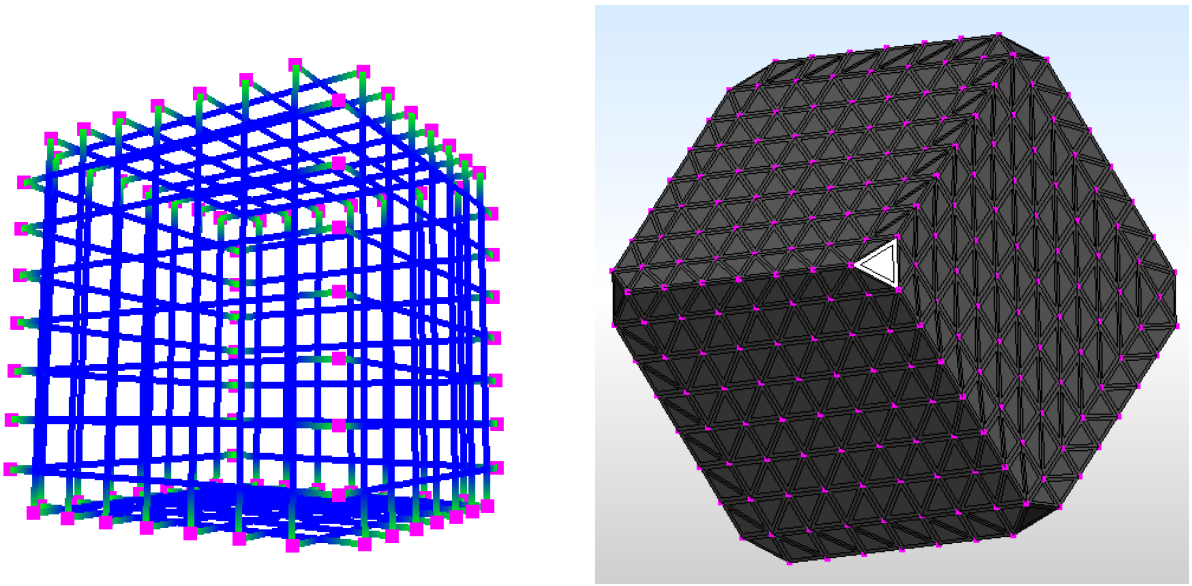

Figure 7. Boundary information partially retrieved from the input edges. Pink dots on the left represent the intersection points between marching cubes' planes and input edges.

Problem in running marching cube on rotated model (axis not aligned)

The above mentioned loss of boundary features issue raised even more issues when I tried to run the ‘axis-not-aligned’ cube model through the marching cube. As in figure 8 (left), even if we retrieve the intersection points, those points will not be utilized if the corresponding cube does not have interpolation points. This results in still rough or chopped off boundaries as in figure 8 (right).

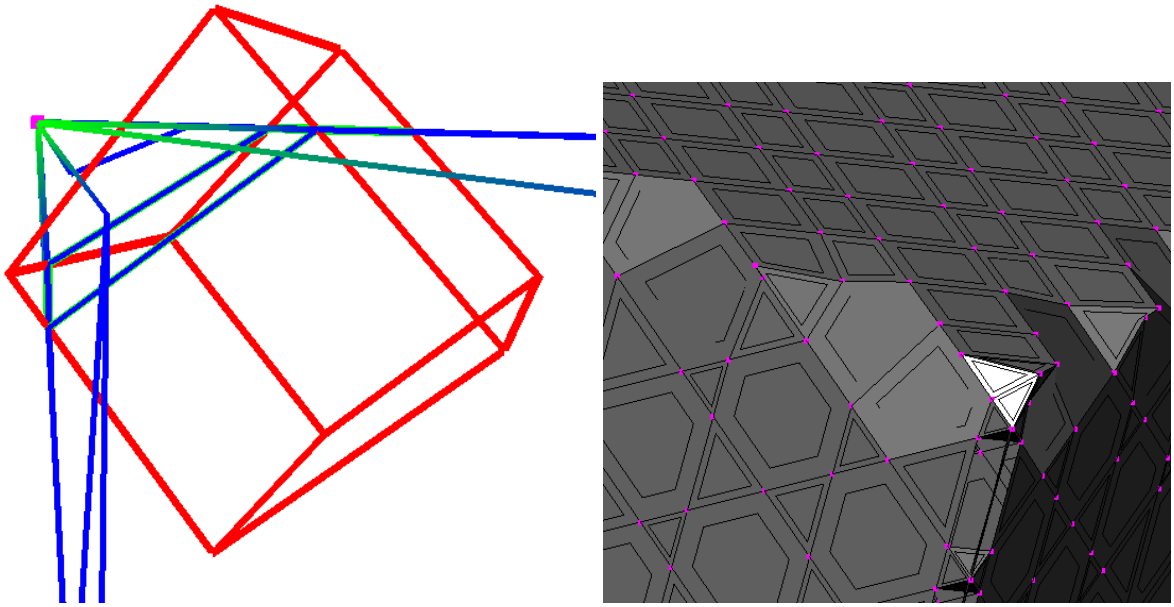

Figure 8. Pink dot in the left figure represents the intersection point but it is not utilized since the cube containing the point has no interpolation points.

Options to resolve the aforementioned issue

The 2 simple options so far are as follows:

1. Modify the function value of one corner of the corresponding cube’s plane that has the intersection point.
    a. Difficult to systematically find which corner’s value to change.
    b. Ambiguous to decide what new value to assign.
    c. Needs post-processing to fix the boundaries.
2. Move the intersection point to the adjacent cube. (I.e if the intersection point was on plane 1 of the cube 1, move it to plane 1 of one of the neighboring cubes.)
    a. Physically moving intersection point yields more post-work to restore the original boundary.

Status quo on the two methods

From implementing (1), modifying the values,

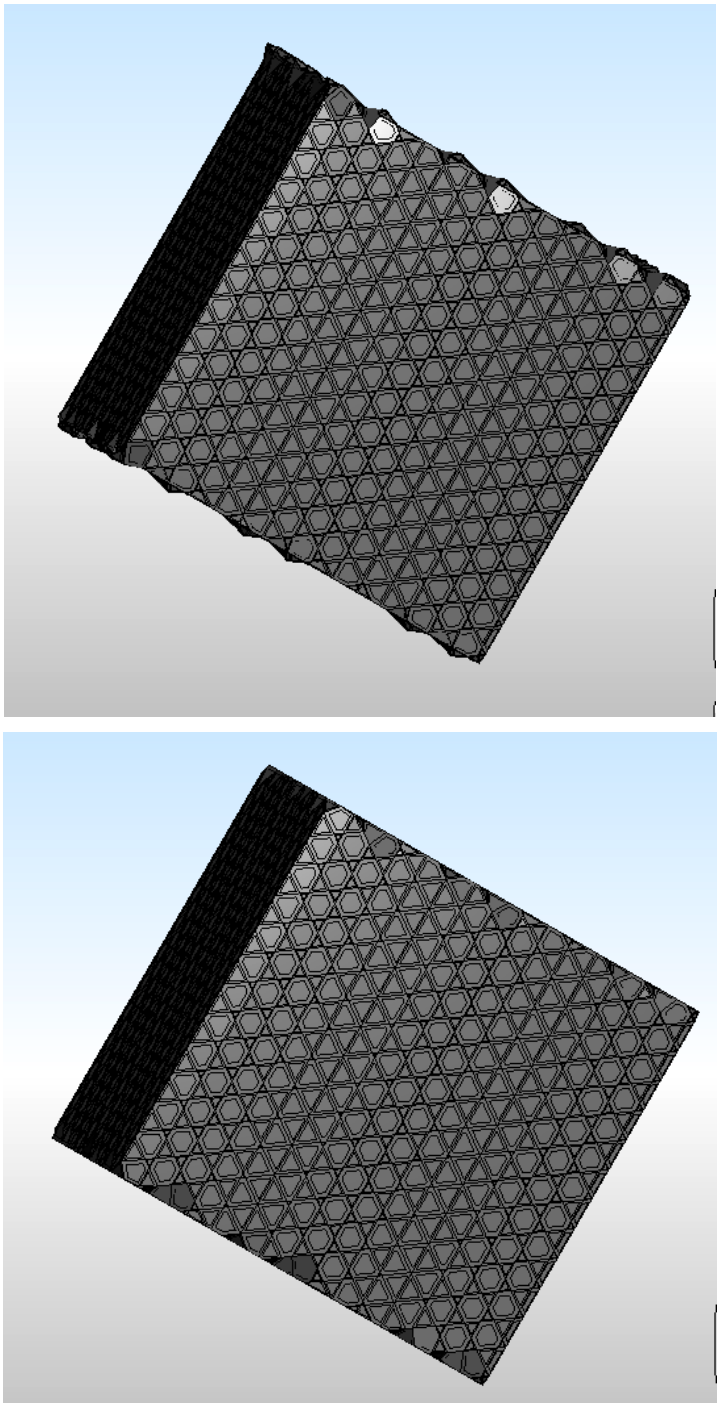

Figure 9. Above: Modified function values with input edge intersections. Below: Original with only input edge intersections.

Modifying the values to include the unused intersection points make the model boundaries protruded and strange [*Fig. 9 (above)*], which is expected, but currently some bug exists in the code where when implemented, the mesh becomes non-manifold, which shouldn't happen [*Fig. 10*].
Function values were assigned as follows:

When a neighboring corner with a different sign of function value is found, assign the average of current corner value and neighboring value if that average is not close to zero. If close to zero, assign a very small positive value.

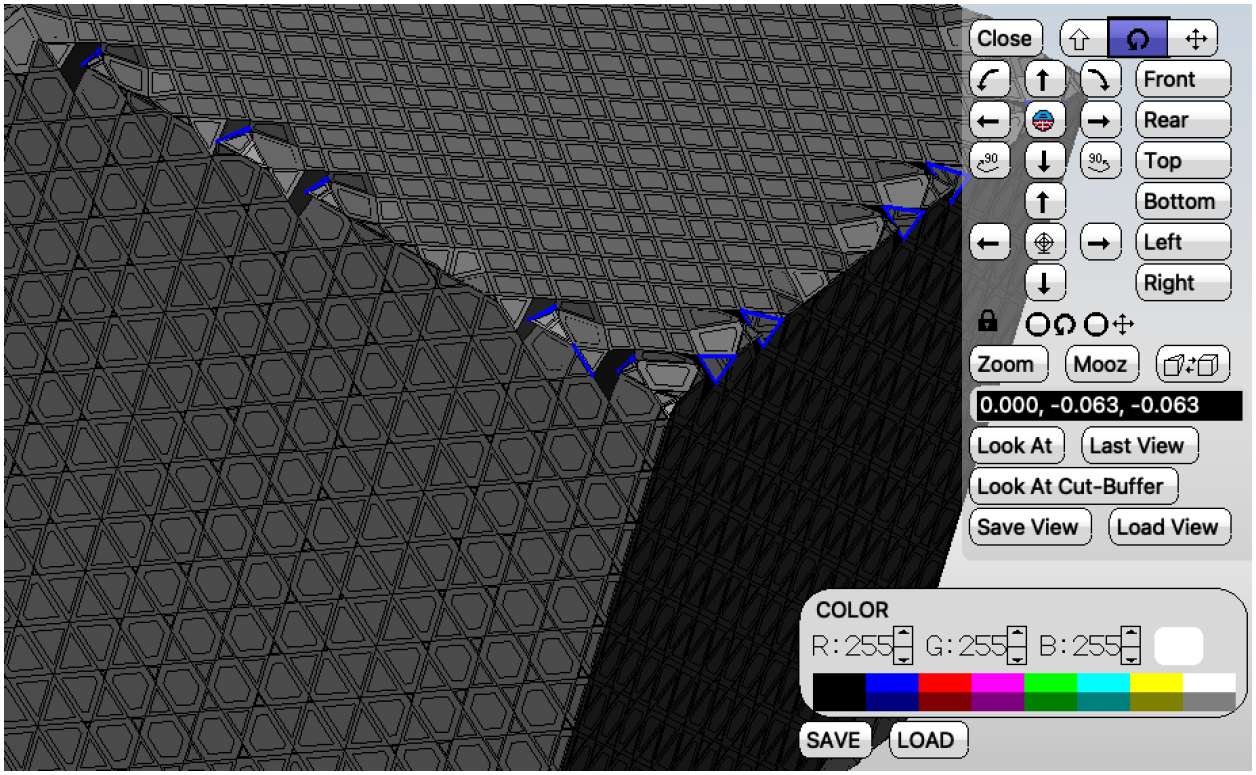

Figure 10. After value modification, some parts (blue in the figure) on the edges of the model became non-continuous.

Method 2 (moving intersection point to another cube that has at least one interpolation point) was also attempted but some bugs also exist that need to be resolved. It is rendered in my program, but cannot be loaded in the professor's mesh viewer. (Most likely the mesh geometry is broken)

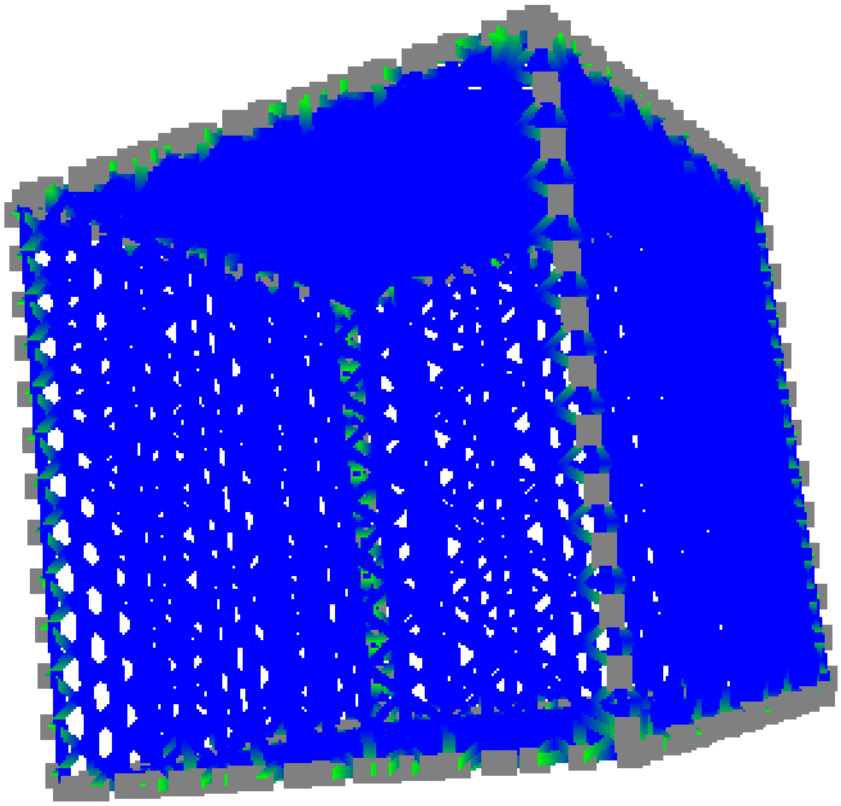

Figure 11. After moving intersections to the neighboring cubes, it is rendered in my program, but there's likely a bug in saving it to polygons, which breaks the geometry.