Catmull Clark Subdivision and Loop Subdivision

-Aditi Awasthi (aa92578), Asmita Limaye (al57252)
Github repository: https://github.com/aditiawas/354H-FinalProj

Subdivision algorithms take a given mesh and subdivide it by defining new vertices, faces and edges which are derived from the previous ones **via a refinement scheme** to produce a denser mesh. The **limit surface** theoretically is supposed to be the value of the mesh after the subdivision algorithm is applied an infinite number of times. An extraordinary vertex is created when either less than four or more than four edges form a vertex. These are irregular areas of high curvature in the mesh. Edges can be specified to be sharp to ensure that these edges are not 'smoothed over' when subdivision is applied iteratively.

Catmull Clark Subdivision [1]
Catmull Clark subdivision is applied to meshes consisting of quad faces, and the output mesh consists of quad faces. The algorithm computes a) **new face points** for each face, where each face point is the average of the vertices corresponding to each face b) **new edge points** for each edge, where each edge point is the average of the two neighbouring face points and the two endpoints of the edge and c) **new vertex points** for each vertex P: we take the average (F) of all n (recently created) face points for faces touching P, and take the average (R) of all n edge midpoints for original edges touching P, where each edge midpoint is the average of its two endpoint vertices (not to be confused with new edge points above). The point P is moved to the new vertex point (F + 2*R + (n-3)*P)/n, where n is the number of adjacent faces.
To form edges and faces in the new mesh: a) Each new face point is connected to the new edge points of all original edges defining the original face b) Each new vertex point is connected to the new edge points of all original edges incident on the original vertex c) New faces are defined which are enclosed by these edges.

Loop Subdivision [4]
The algorithm works on meshes consisting of trimesh faces.
For each original vertex, a new vertex is created by weighted averaging of the neighboring vertices. The weights are determined by the valence (number of incident edges) of the vertex. For each original edge, a new vertex is created by weighted averaging of the endpoints of the edge and the two opposite vertices of the adjacent triangles. The original mesh is split into smaller triangles by connecting the newly created vertices. The process is repeated for a specified number of iterations to achieve the desired level of smoothness.

Vertex update rule: The position of each new vertex is calculated using a weighted average of its neighbouring vertices. The weights are chosen to ensure a smooth limit surface. For a vertex with valence n, the weight for the vertex itself is (1 - n * beta), and the weight for each neighbouring vertex is beta, where beta = (1 - alpha) / n, and alpha = (4 - 2 * cos(2π / n)) / 9.
Edge split rule: Each original edge is split by introducing a new vertex at its midpoint. The position of the new vertex is calculated using a weighted average of the endpoints of the edge and the two opposite vertices of the adjacent triangles. The weights are 3/8 for the endpoints and 1/8 for the opposite vertices.

Implementation details

GUI: This code is present in the smoothing.cpp file, which is the entry point for our project. I have implemented a basic GUI using the FLTK library to provide a user-friendly interface for selecting input files, choosing which subdivision algorithm to implement and setting number of iterations.

Catmull Clark: All data structures and code for the implementation can be used in the `catmull.cpp` and `catmull.h` files. I stored the mesh vertices and mesh faces for each catmull clark object, and calculated the mesh normals (`quadVertices, quadFaces, quadNormals`), Additionally, to define sharp creases/edges, I defined a vector called vertex_sharpness (`vertex_sharpness`) which would correspond to the boolean value of sharpness for the vertices.
For performing subdivision: `countFacesAdjacent`: to count faces adjacent to each vertex, `countSharpEdgesAdjacent`: to count the sharp edges incident on each vertex, `edgesToFaces`: A map from each edge (constructed using the vertices) to the index of the face it belongs to, `verticesToSharpEdgeVertices`: A map from each vertex to the endpoints of the sharp edges that are incident upon it .
The **extraordinary vertices** are handled by the refinement scheme, since the number of faces adjacent to a vertex are counted and subdivision is performed taking this into account.
To handle **sharp creases**, I used the following rules: [2]

a) Edge points: For each sharp edge (endpoints are both sharp), the new edge point is the average of the old edge's endpoints and is tagged as sharp.
b) Vertex points: If the number of sharp edges is <2, the normal rule is followed for updates. If the number of sharp edges = 2, the new vertex point is a weighted average of the old vertex location (3/4) and of the two other endpoints of the incident sharp edges creases (⅛ each). If the number of sharp edges >2 or the vertex is a sharp vertex, the new vertex point is just the old vertex location (the vertex does not move).

Loop Subdivision: The code for this section can be found in the `loop.cpp` and `loop.h` files. The mesh vertices and trimesh faces are member variables of the `LoopSubDiv` class `(vertices, trimeshFaces, cv, ce)` (along with crease vertices and edges)
Important data structures used here include:

- `TrimeshFace`: Represents a triangular face using indices into the vertex array.
- `HalfEdge`: Represents a half-edge in the half-edge data structure, storing vertex positions and connectivity information.

The function `subdivLoop` contains the following important components:

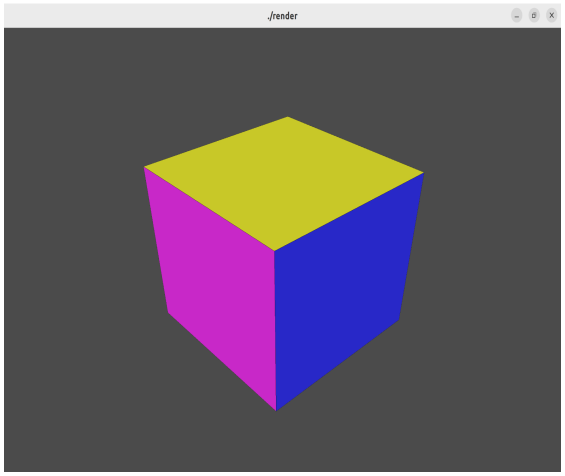
- `halfEdgeDS`: Represents the half-edge data structure. It maps a string key (concatenation of vertex indices) to a `HalfEdge` object.
- `vertexLocToIdx`: Maps a string key (concatenation of vertex indices) to the corresponding vertex index in the `newVertices` vector.
- `oldVertexNeighbors`: Maps a vertex index to a vector representing the neighboring vertices of the corresponding vertex in the previous iteration.
- `creaseVertexSet` and `sharpEdgeSet`: Store crease vertices and sharp edges.
- `beta`: Represents the weight used in the vertex update formula, calculated based on the valence and whether the vertex is a crease vertex.

Rendering: The code for this is present in render.cpp and render.h. It implements an OpenGL-based 3D viewer (using OpenGLWindow class) for displaying triangular and quad meshes and also handles user interactions. It supports file parsing, wireframe and solid rendering in pleasing colors, camera navigation, and basic lighting.

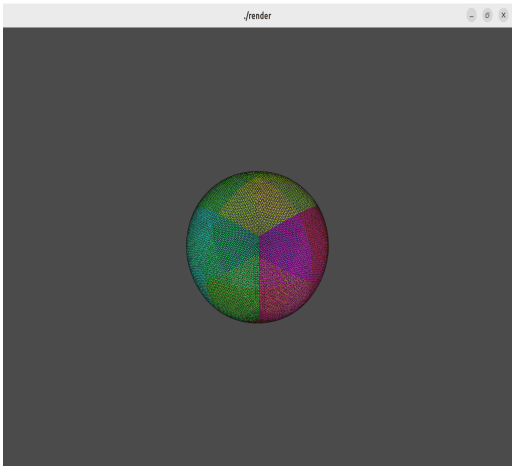
Catmull Clark:

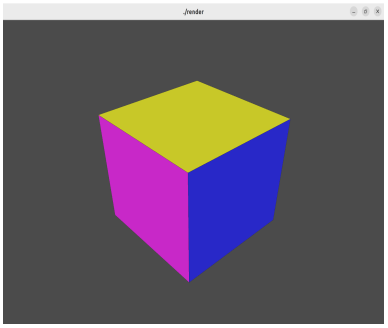
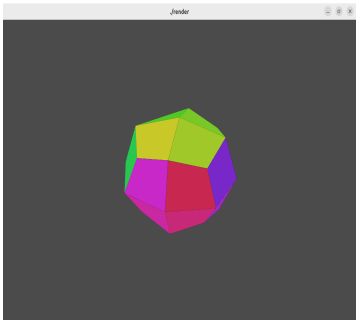
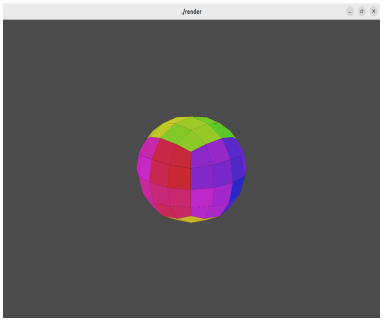
Sample 1

Input object



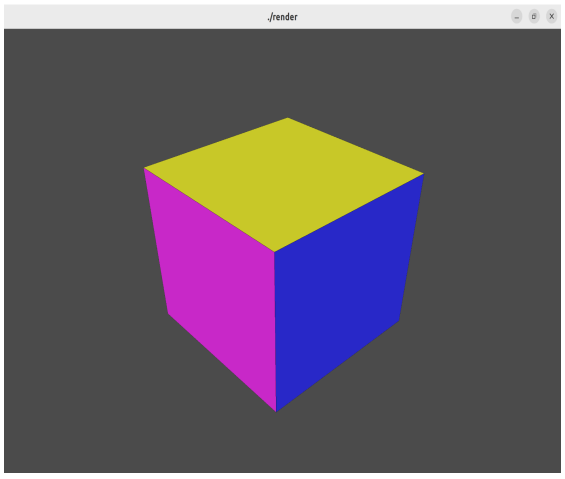
Limit surface



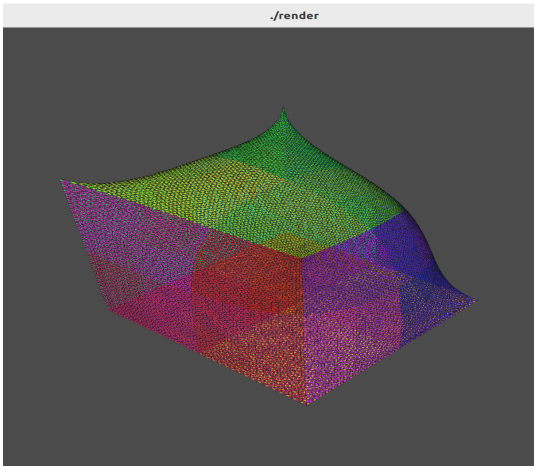
Input	Iteration 1	Iteration 2
		

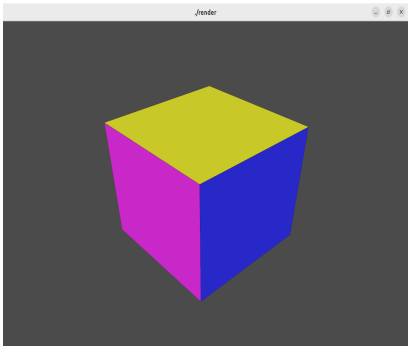
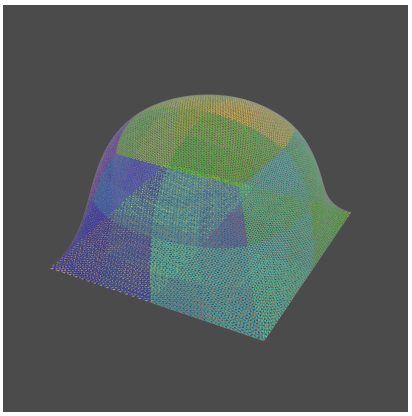
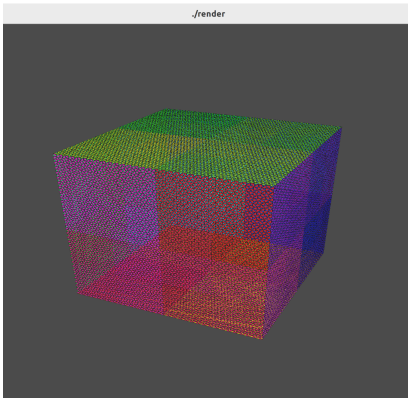
Sample 2 (Sharp edges)

Input object



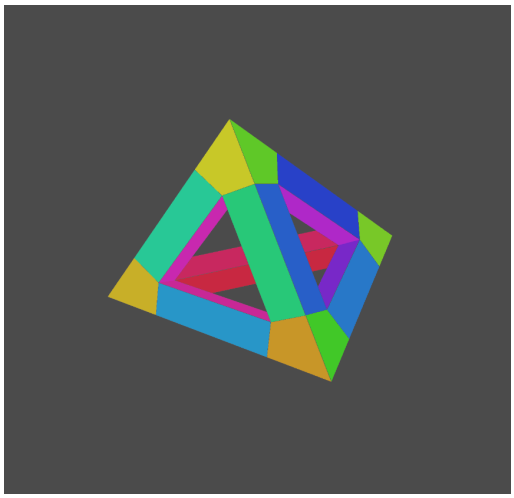
Limit surface



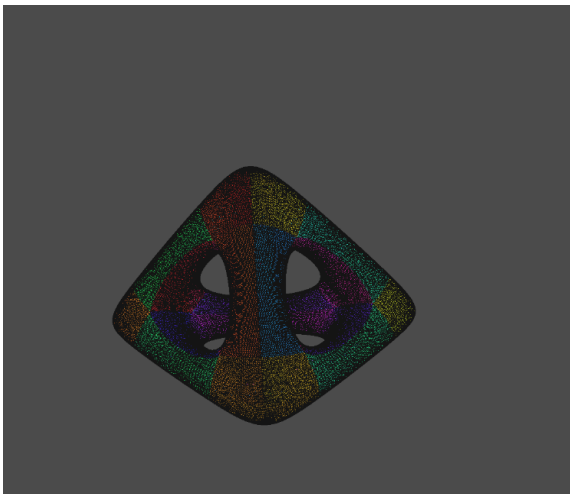
Input	One face sharp	All faces remain sharp
		

Sample 3

Input object

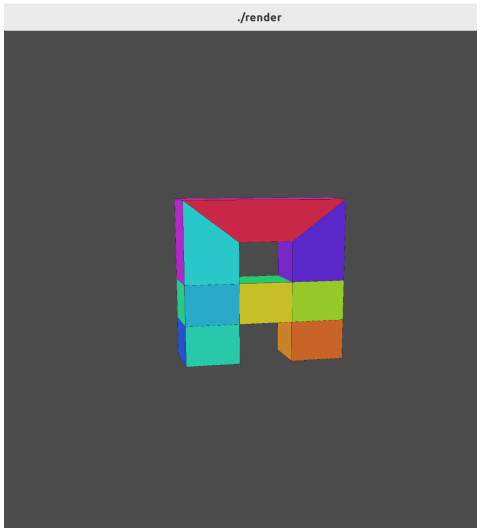


Output



Sample 4

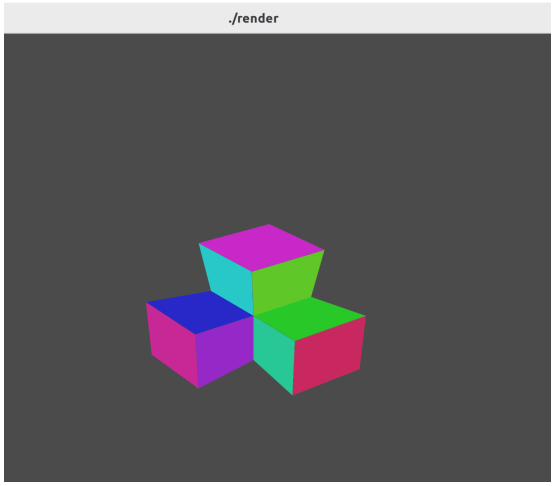
Input object



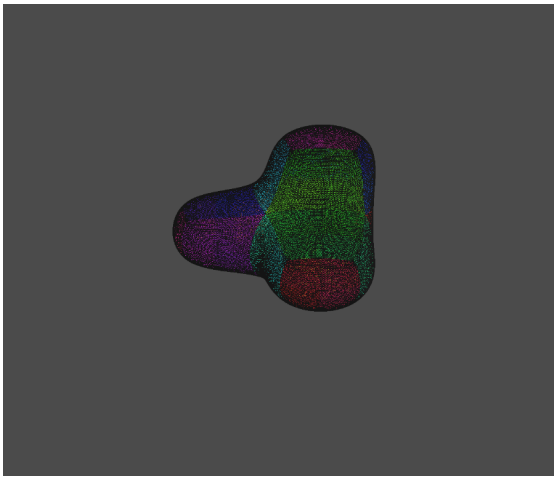
Output

Sample 5 (Closed stacked cubes)

Input object

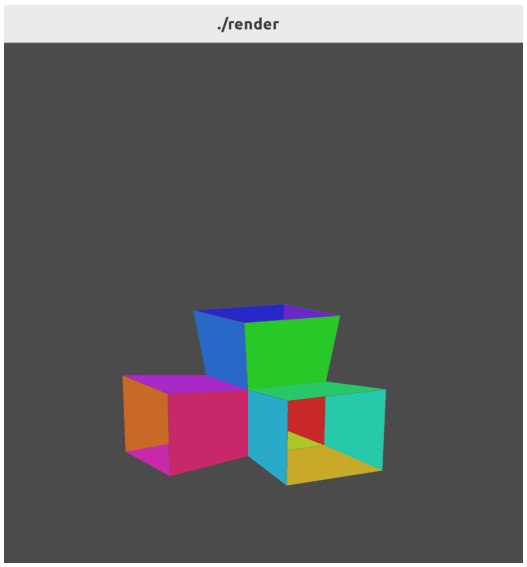


Output

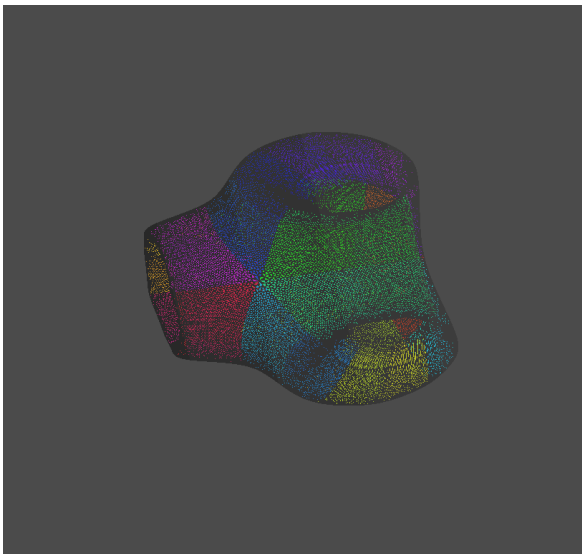


Sample 6 (Open stacked cubes)

Input object



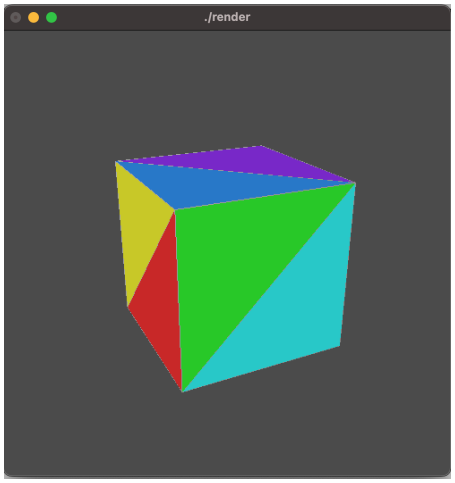
Output



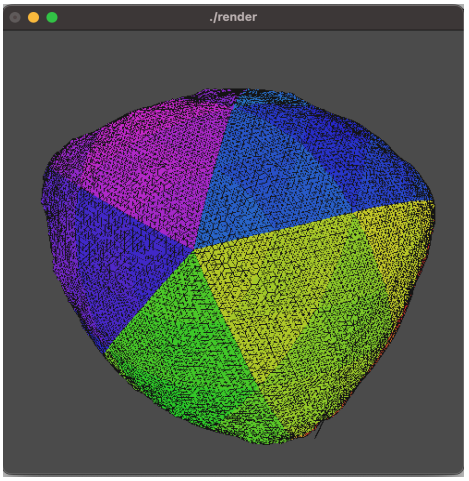
Loop Subdivision:

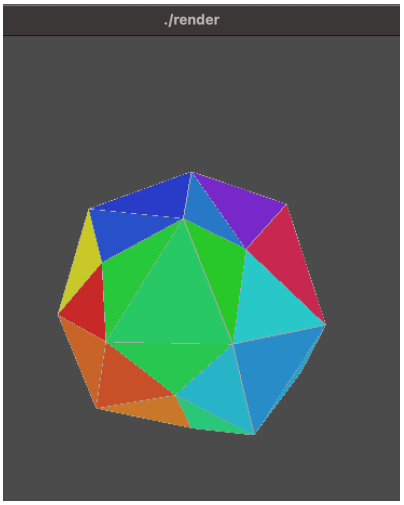
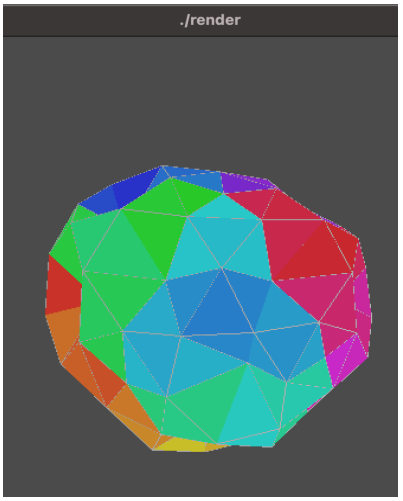
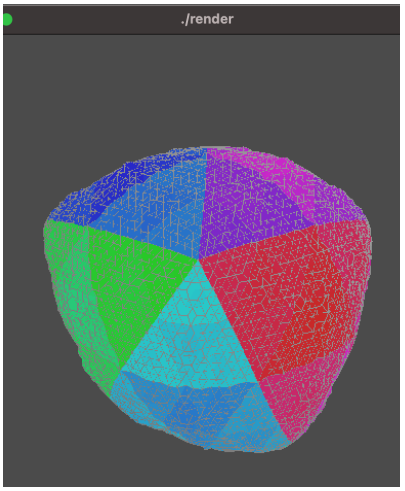
Sample 1

Input object



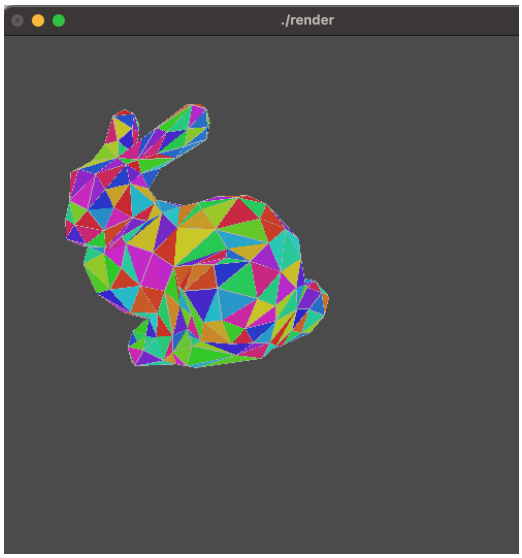
Limit surface



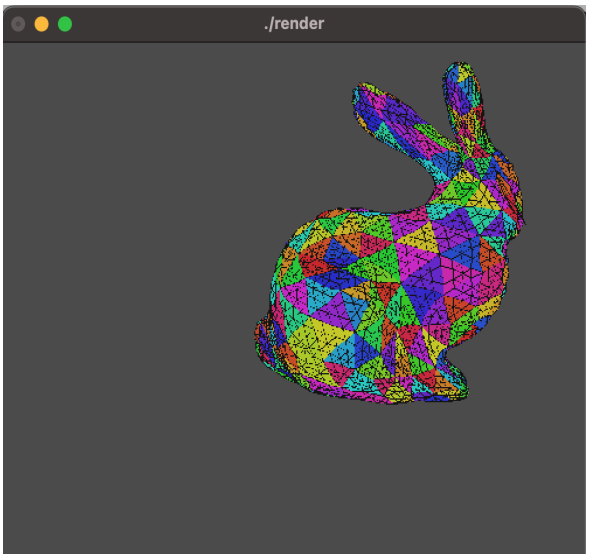
Iteration 1	Iteration 2	Iteration 4
		

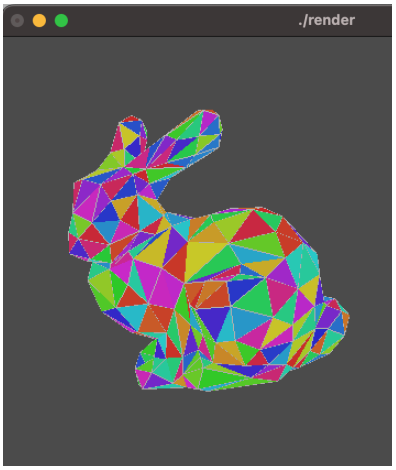
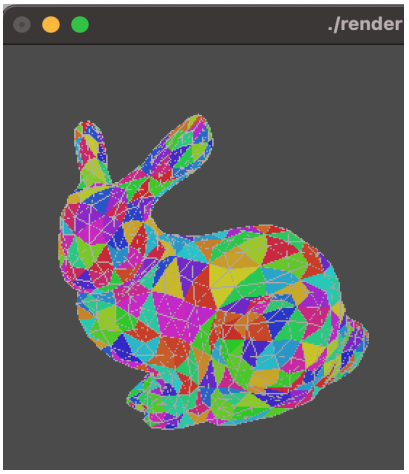
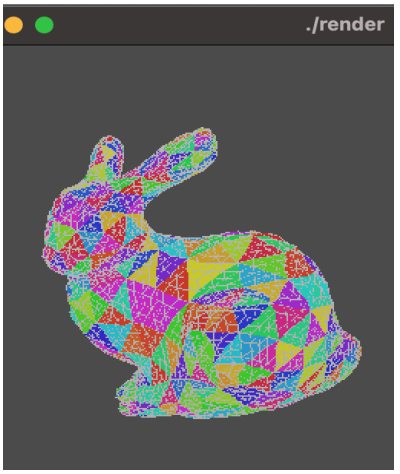
Sample 2

Input object



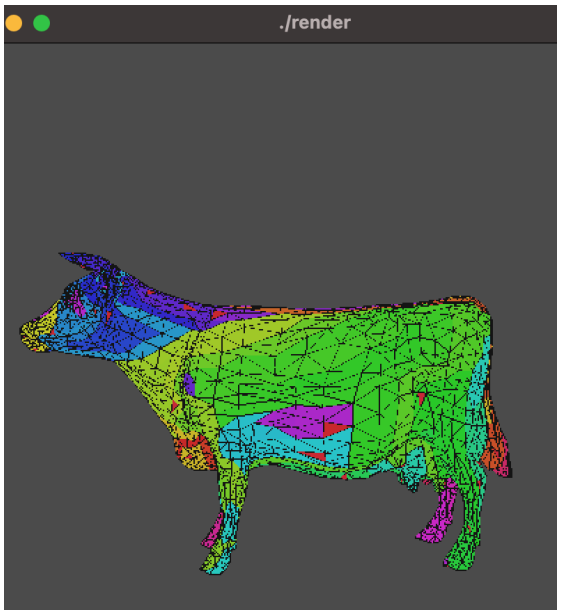
Limit surface



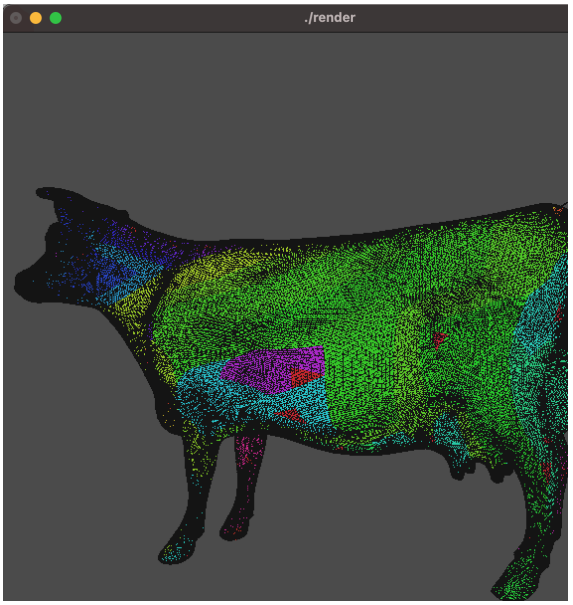
Input	Iteration 1	Iteration 2
		

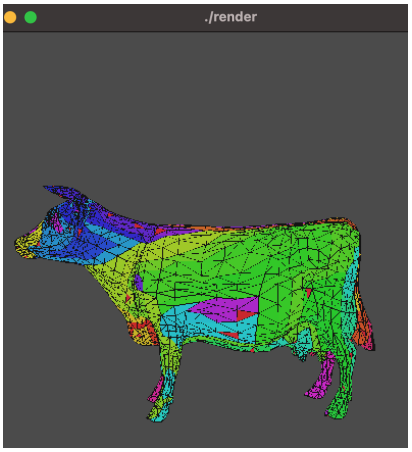
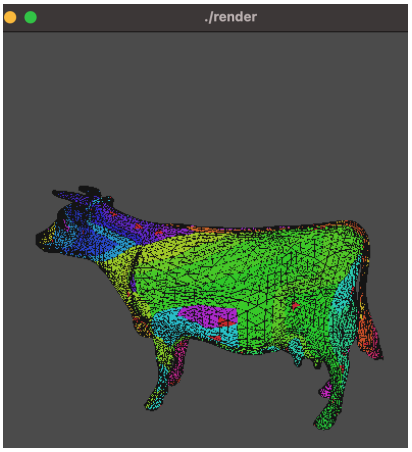
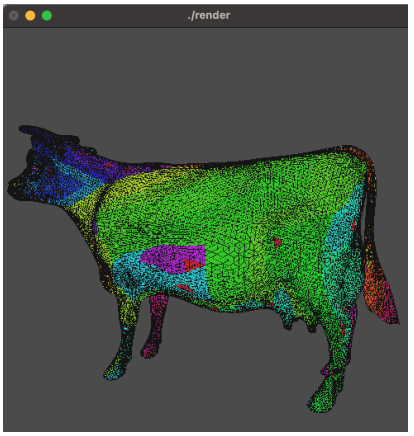
Sample 3

Input object

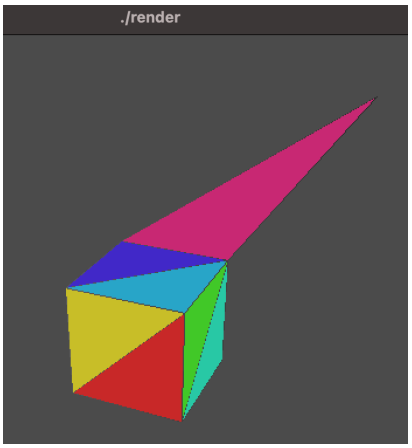
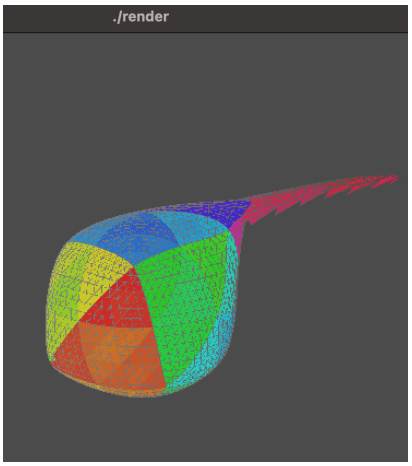
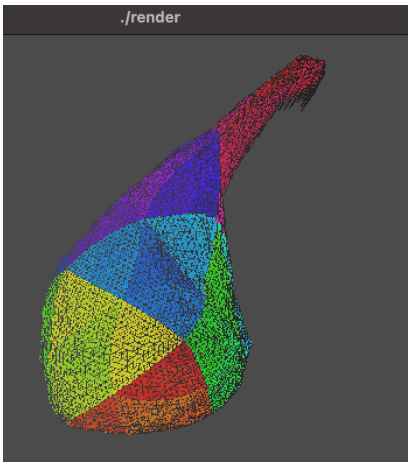


Limit surface

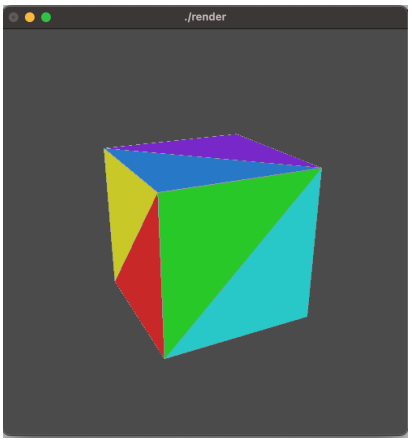
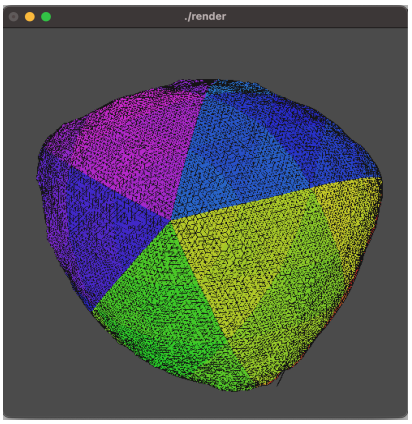
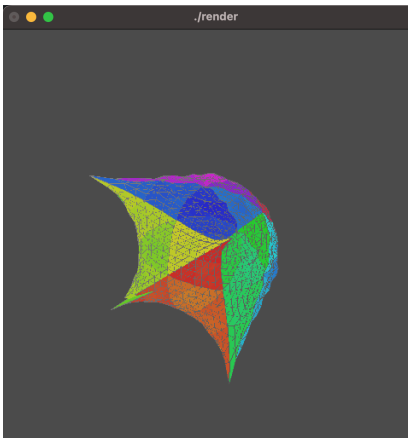


Input	Iteration 1	Iteration 2
		

Sample 4 - Example for extraordinary vertices handling

Input	Without handling	After handling
	Wayward triangles observed through the extension reaching to the extraordinary vertex	The extension to the extraordinary vertex looks very smooth
		

Sample 5 - Example for creases

Input	Without handling	After handling
	The creases get smoothed out	The creases remain sharp
		

Bibliography

1. “Catmull–Clark subdivision surface.” *Wikipedia*, https://en.wikipedia.org/wiki/Catmull%E2%80%93Clark_subdivision_surface. Accessed 2 May 2024.

2. “Catmull-Clark subdivision surfaces: corners and creases - XRT Renderer.” *XRT Renderer*, http://xrt.wikidot.com/blog:31. Accessed 2 May 2024.

3. “Subdivision surface.” *Wikipedia*, https://en.wikipedia.org/wiki/Subdivision_surface. Accessed 2 May 2024.

4. “Loop subdivision surface.” *Wikipedia*, https://en.wikipedia.org/wiki/Subdivision_surface#Loop_subdivision_surface. Accessed 2 May 2024.

5. “Subdivision lecture slides.” *CMU,* https://www.cs.cmu.edu/afs/cs/academic/class/15462-s14/www/lec_slides/Subdivision.pdf. Accessed 2 May 2024.

6. “Subdivision book notes.“, *UW Madison,* https://graphics.cs.wisc.edu/Courses/559-f2010/pubs/pub_RTR-Subdivision.pdf. Accessed 2 May 2024.