CS 3451 - Computer Graphics **Fall 2025**

Assignment 2: Mesh Subdivision

We will practice implementing Loop's subdivision algorithm to subdivide a triangle mesh. You will be tasked with developing the three subdivision steps to refine a triangle mesh and enhance its visual quality.

1 Reading

Your implementation will be based on C++11 and the Eigen vector math library. Before commencing the programming tasks, you should familiarize yourself with the algorithm details by reviewing the relevant course slides. For additional context and a deeper understanding, we recommend referring to Charles Loop's Master's thesis, "Smooth Subdivision Surfaces Based on Triangles," as well as Matt's blog on Loop subdivision, both accessible via the provided link below.

- Course slides (on Canvas)
- Matt's blog on Loop subdivision
- Charles Loop's M.S. thesis

2 Starter Code

Make sure to execute 'git pull' from our git repository to ensure you have the most recent updates, particularly as this assignment has been recently updated from its previous version. After pulling the latest code, make sure to run the setup script to recreate the project files (for Windows: run `.\scripts\setup.bat`; for Mac or Linux: run `./scripts/setup.sh`). The starter code for this assignment, found under `assignments/a2/`, includes two C++ files: `main.cpp` and `LoopSubdivision.h`. Your implementation should be focused on `LoopSubdivision.h`.

The outcomes of this assignment will be dynamically displayed in an OpenGL window. By pressing the **"s"** key, you can initiate a single iteration of the subdivision process. Conversely, pressing the **"r"** key reverts the mesh to its original state. Press **"z,"** **"x,"** **"c,"** **"v"** to switch to different models. In particular, **"z"** for the default icosahedron mesh, "x" for the cap mesh, **"c"** for the jet fighter mesh, and **"v"** for your customized mesh (which is also the icosahedron mesh before you put your own mesh file named "my_mesh.obj" in a2 folder). **The mesh will disappear after pressing** "s" **due to the empty array of triangles before you finish implementing the first step.** Use the left and right keys of your mouse to rotate and zoom in on the camera view. This interactive setup allows you to observe the progressive subdivision and smoothing of the triangle surfaces, achieved through successively pressing **"s."**

Upon successful compilation and execution, an OpenGL window appears with a default icosahedron mesh object (see Figure 1a), which will be the mesh for our default test example. If you are interested in the method used to create the initial icosahedron shape for the sphere, you can find the relevant information at the following link: Icosahedron Sphere Tutorial.

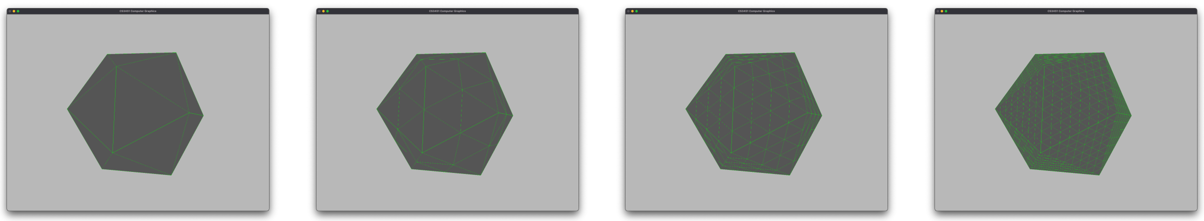
3 Algorithm

The fundamental principle of Loop's subdivision algorithm revolves around the alternate refinement of topology and smoothing of geometry. In each iteration, the algorithm initially subdivides each triangle in the input mesh into four smaller triangles. This is achieved by introducing a new vertex on each edge. Subsequently, it smooths the positions of all vertices. This smoothing process encompasses both the newly added vertices (we call them **odd vertices**) and the pre-existing vertices (we call them **even vertices**), each treated in distinct ways.

Regarding the algorithm's input, it accepts a coarse triangle mesh and aims to produce a refined output. Specifically, the inputs are two arrays (both represented as `std::vector` in C++): `old_vtx` and `old_tri`, which store the vertices and triangles of the coarse mesh, respectively. The outputs are `new_vtx` and `new_tri`, representing the vertices and triangles of the refined mesh. The arrays `new_vtx` and `new_tri` are initially empty at the start. Considering that all vertices from the coarse mesh also form part of the refined mesh, we copied all the vertices from the `old_vtx` into `new_vtx` before executing the subdivision algorithm.

The algorithm you will implement comprises three steps:

Step 1: Add Odd vertices and Sub-triangles In this step, you will begin by adding new vertices and updating the triangles of the mesh. Specifically, there are two tasks: (1) For each edge on the old mesh, add a new vertex to the `new_vtx` list. This new vertex, also known as an odd vertex, should be positioned at the midpoint of the edge. The index of this vertex in the `new_vtx` array will correspond to its position in the list. (2) For each triangle on the old mesh, add four sub-triangles to the `new_tri` list. When doing so, ensure that the orientation of each sub-triangle is preserved, with the three vertices following a counterclockwise order to maintain consistency in the mesh's structure. After your implementation, by clicking **"s,"**, you should be able to find an icosahedron mesh with refined triangles and vertices, as shown in Figure 1. The positions of all the newly added vertices (i.e., odd vertices) will remain on the edges of the old mesh, and the positions of all old vertices (i.e., even vertices) will remain unchanged, which will be smoothed in the next two steps.



(a) Original (b) 1 iteration (c) 2 iteration (d) 3 iteration

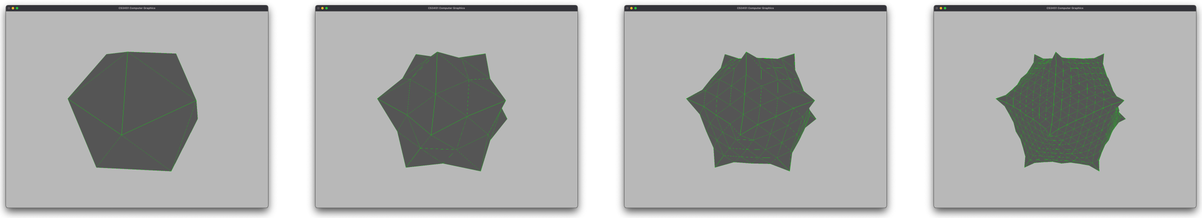
Figure 1: Illustration of subdivision results after step 1

Step 2: Update Odd Vertex Position In this step, you will focus on updating the position of odd vertices that were introduced in Step 1. This involves three sub-tasks: (1) Implement the `FindTheOtherVertex()` function, which will locate the vertex opposite a given edge within a triangle. You will need this function to determine the correct set of vertices involved in the subsequent smoothing operation. (2) Implement the `SmoothOddVtx()` function, which computes the smoothed position of an odd vertex. The smoothed position should be calculated using the formula

$$\text{Smoothed odd vertex} = 0.375 \times (V_0 + V_1) + 0.125 \times (V_2 + V_3),$$

where V_0 and V_1 are the two vertices incident to the edge, and V_2 and V_3 are the two vertices opposite the edge (see our course slides for details). (3) Finally, within the for-loop, call the `SmoothOddVtx()` function to update the position of each odd vertex accordingly so that all odd vertices are repositioned based on their surrounding geometry.

After your implementation, by clicking **"s,"**, you will see a spiky shape of the icosahedron mesh with refined triangles and vertices, as shown in Figure 2. You see this intermediate result because only the odd vertices are smoothed while all the even vertices remain unchanged.



(a) Original (b) 1 iteration (c) 2 iteration (d) 3 iteration

Figure 2: Illustration of subdivision results after step 2

Step 3: Update Even Vertex Position In this step, you will update the positions of even vertices. This involves two primary tasks: (1) Implement the `SmoothEvenVtx()` function, which calculates the smoothed position of each even vertex. The function should take a given vertex's index and neighboring vertices' indices as input. The smoothed position should be computed following the expression:

$$\text{Smoothed even vertex} = (1 - n\beta)V_{center} + \beta V_{nb1} + \beta V_{nb2} + ... + \beta V_{nbn},$$

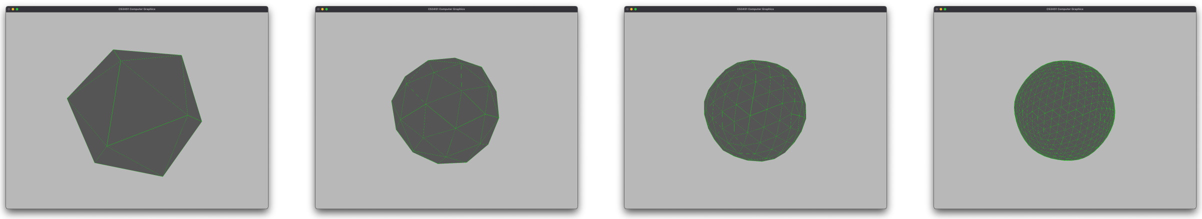
where V_{center} indicates the position of the even vertex itself to be smoothed on the old mesh, and $V_{nb1}, V_{nb2}, ..., V_{nbn}$ are the positions of the neighboring vertices of V_{center} on the old mesh, with n indicating the number of neighbors (see course slides for details).

The value of β depends on the value of n, which is calculated as:

$$\beta = \begin{cases} \frac{3}{8n}, & n > 3 \\ \frac{3}{16}, & n = 3. \end{cases} \qquad (1)$$

(2) Within the for-loop, call the `SmoothEvenVtx()` function to update the position of each even vertex stored in `new_vtx`, ensuring that all even vertices are repositioned based on their surrounding neighbors.

After finishing the implementation, you should see the final result—a smoothly subdivided icosahedron mesh with both odd and even vertices smoothed (see Figure 3)! The final mesh will exhibit an (imperfect) spherical shape (no worries about its imperfection; that is the result we expected :-).

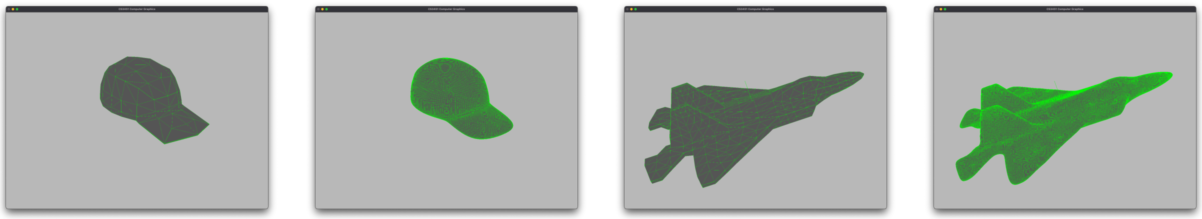


(a) Original (b) 1 iteration (c) 2 iteration (d) 3 iteration

Figure 3: Illustration of subdivision results after step 3

4 Other Mesh Shapes

Our starter code includes two other meshes for you to test your algorithm. Press keys **"x"** and **"c"** to switch to a cap mesh and a jet fighter mesh (see Figure 4). Press **"s"** to check your subdivision algorithm on these new meshes.



(a) Original (b) After 3 iterations (c) Original (d) After 3 iterations

Figure 4: Subdivision results on the cap and the jet fighter

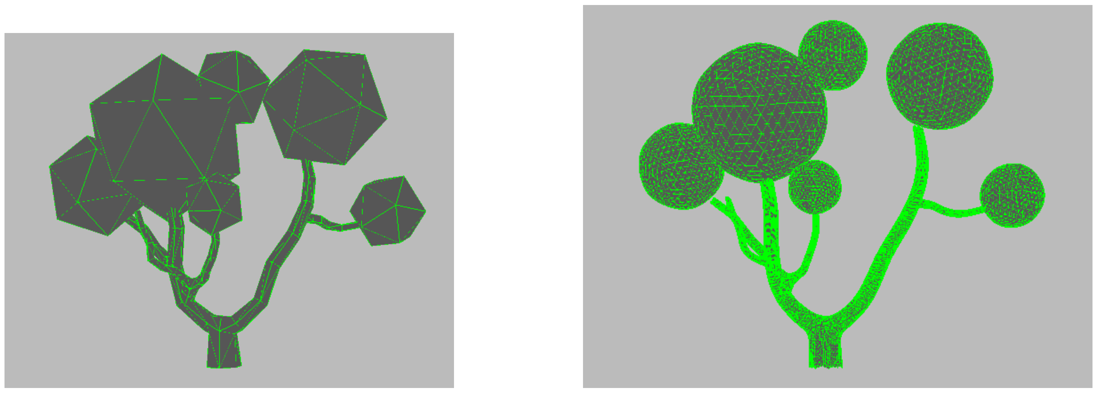
5 Creative Expression: Customized Mesh

For the creative expression part, you are also asked to test your algorithm on a customized mesh object. You can download an existing mesh model as the initial shape from the internet. Your creative work should align with one of the following themes: **Creature and Machine**.

Follow the following steps to test your customized mesh in the starter code:

- Rename your own mesh file as "my_mesh.obj" and put it into the a2 folder
- Run the program and press "v."

We provide an example of customized mesh subdivision in Figure 5.



(a) Original (b) After 3 iterations

Figure 5: An example of running Loop's Subdivision on a customized mesh

6 Submission

Submit the following components for evaluation:

- Your source code `LoopSubdivision.h`;
- Screenshots of subdivision results on an icosahedron mesh, cap mesh, and jet fighter mesh;
- A video demonstrating the functionality of your subdivision algorithm on a customized mesh;
- A concise paragraph that provides a technical explanation of your implementation.

7 Grading

This assignment is worth a total of 8 points, with the grading criteria outlined as follows:

1. Technical contribution (7 pts): The distribution of points is as follows:
   - Adding odd vertices and triangles: 2 points
   - Smoothing odd vertices: 3 points
   - Smoothing even vertices: 2 points
2. Creative expression: 1 point

8 Sharing your Work

You are encouraged to share your graphical work with the class. If you want to do so, please upload your image to Ed **A2 Gallery: Creature and Machine**. This is an excellent opportunity to engage with your peers and gain recognition for your work. We look forward to seeing your beautiful mesh objects!