Mesh Subdivision

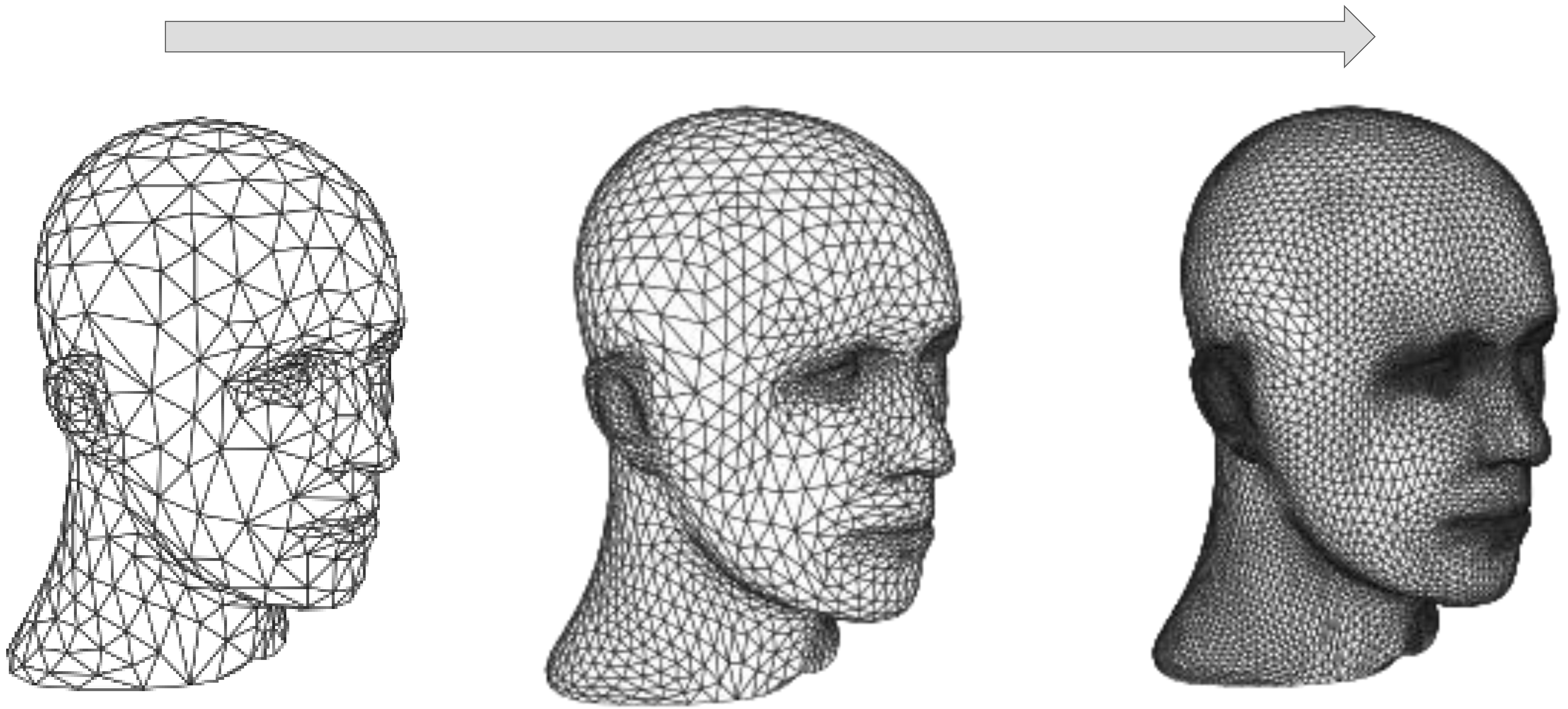
Bo Zhu

School of Interactive Computing

Georgia Institute of Technology

Subdivision surfaces

- Start with a coarse triangle mesh, and recursively subdivide on it

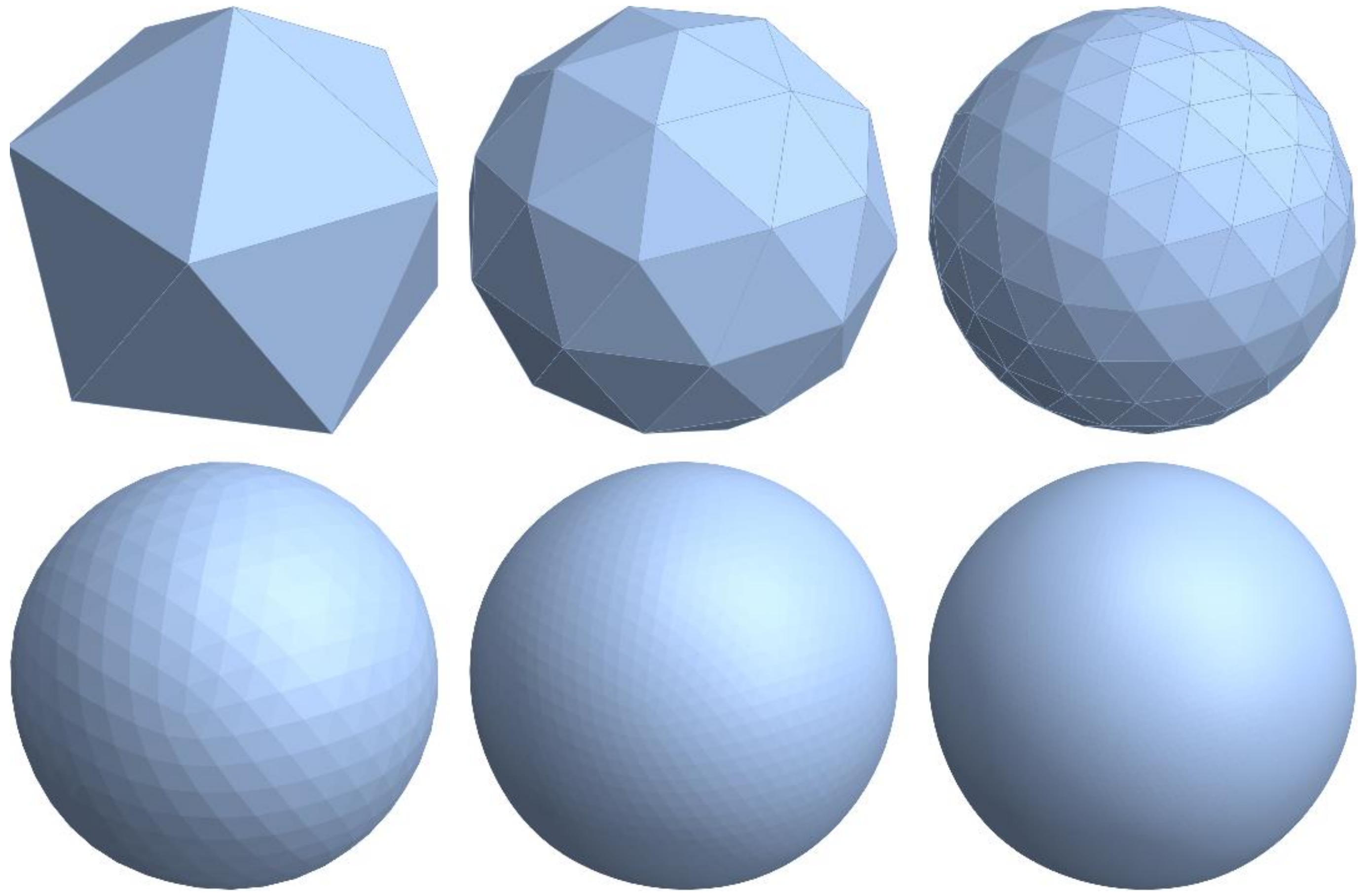


Algorithm Overview

- Two stages: (1) update **topology** by refining mesh (2) update **geometry** by moving vertices

1. Mesh subdivision
   - new topology
   - create new vertices and faces
2. Vertex placement
   - new geometry
   - compute vertex position

Different schemes available, depend on input/output topology and geometry:

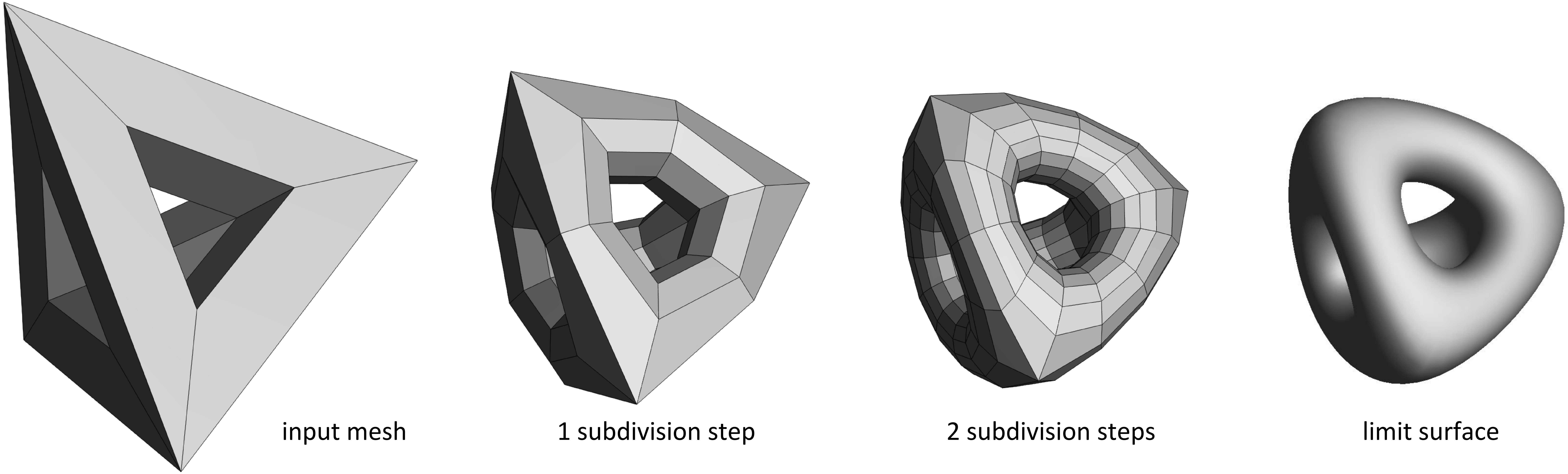


Subdivision Process

1. Start with input mesh $\mathbf{M}_0$
2. Recursively apply subdivision rule f: $\mathbf{M}_{i+1} = f(\mathbf{M}_i)$

- Limit surface:

$$\mathbf{S} = \lim_{i\to\infty} \mathbf{M}_i = \lim_{i\to\infty} f^i(M_0)$$

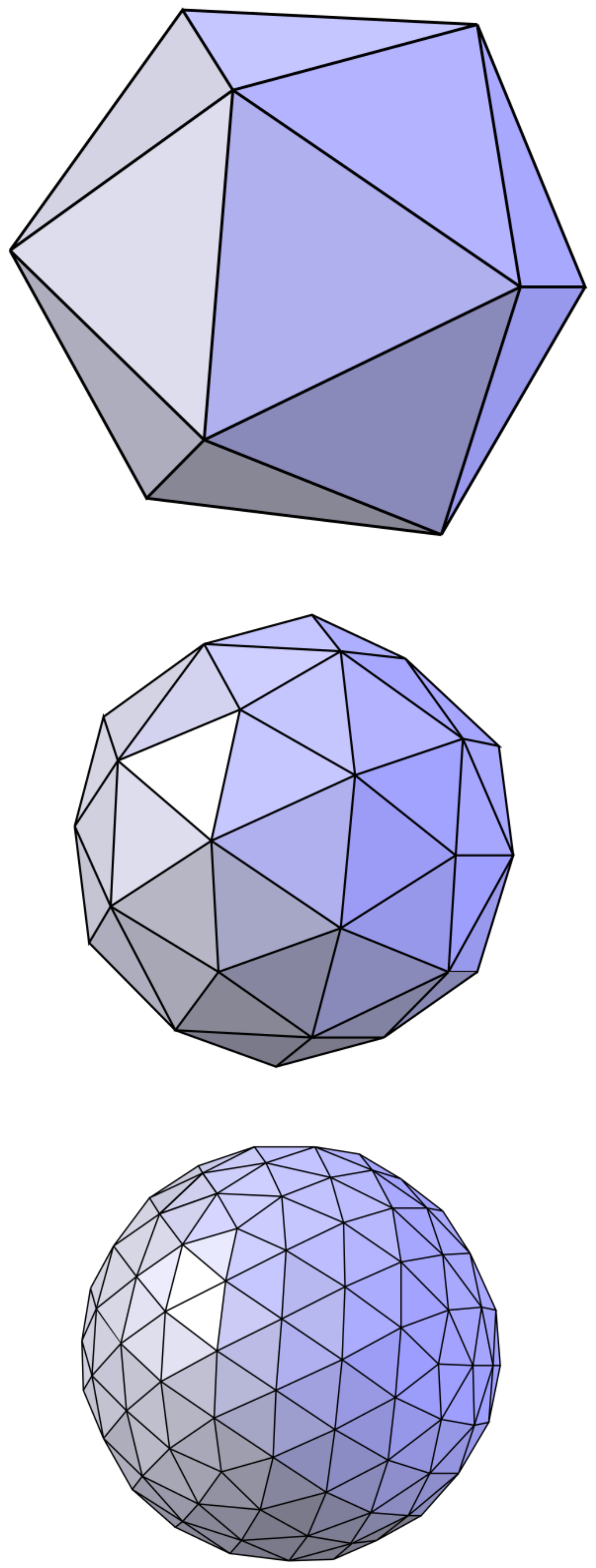


[DeRose et al.1998]

Charles Loop

Charles Loop is a Principal Research Scientist in the Learning & Perception Research group with NVIDIA Research. Charles is the inventor of Loop Subdivision, a geometric modeling algorithm used for creating smooth shapes for use in medical imaging, special effects, and video games. He was recently awarded a Technical Achievement Award from The Academy of Motion Picture Arts and Science for this work. Charles other contributions to Computer Graphics include the GPU font and path rendering algorithm used in popular commercial graphic design tools; to the hardware accelerated subdivision surface rendering algorithm used in Pixar's Open SubDiv library. While at Microsoft Research, he initiated a project called Holoportation, that demonstrated a two-way telepresence system, allowing users wearing Augmented Reality (AR) display devices such as Hololens to interact with each other's photo-realistic 3D holograms in real-time, while physically separated by thousands of miles. Since joining NVIDIA, Charles has worked on applying Deep Learning to 3D reconstruction problems. Charles holds an M.S. in Mathematics from the University of Utah, and a Ph.D. in Computer Science from the University of Washington. Charles is located in Redmond.

Personal webpage: charlesloop.com.



Research Interests:

- Computer Graphics, Geometric Modeling, Surface Reconstruction, RGBD Image-Based Modeling, Subdivision Surface Techniques, Surface Rendering, GPU Algorithms, GPU Ray Tracing.

Education:

- Ph.D. Computer Science, University of Washington. 1992. *Dissertation:* "Generalized B-spline Surfaces of Arbitrary Topological Type".
- M.S. Mathematics, University of Utah. 1987. *Thesis:* "Smooth Subdivision Surfaces Based on Triangles".

Awards:

- Technical Achievement Award, Academy of Motion Picture Arts and Science. 2019. "For his influential research on the fundamental scientific properties of subdivision surfaces as 3D geometric modeling primitives".

Loop Subdivision by Charles Loop in 1987

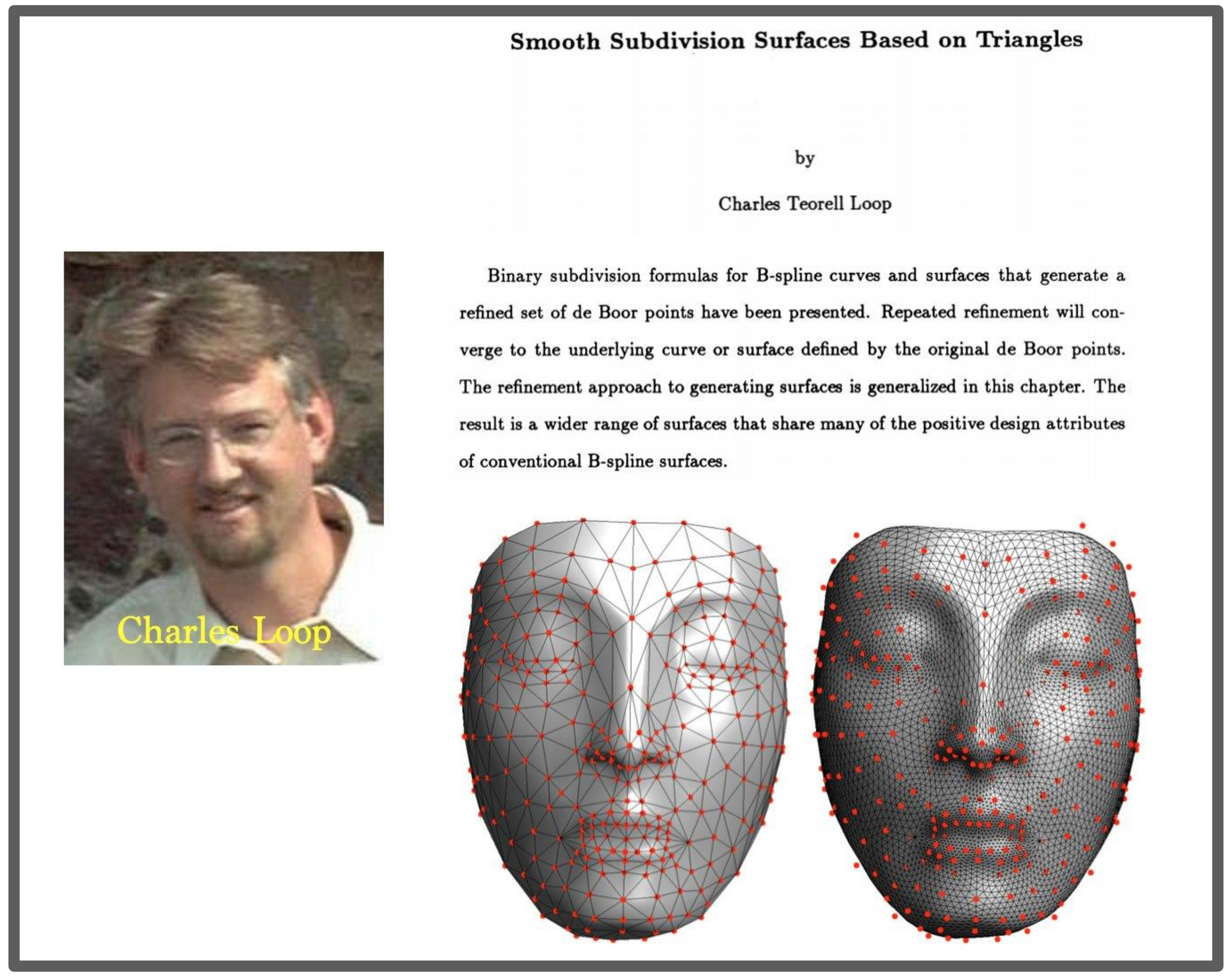
Reference

Charles Loop:

Smooth Subdivision Surfaces Based on Triangles**,**

M.S. Mathematics thesis,

University of Utah, 1987

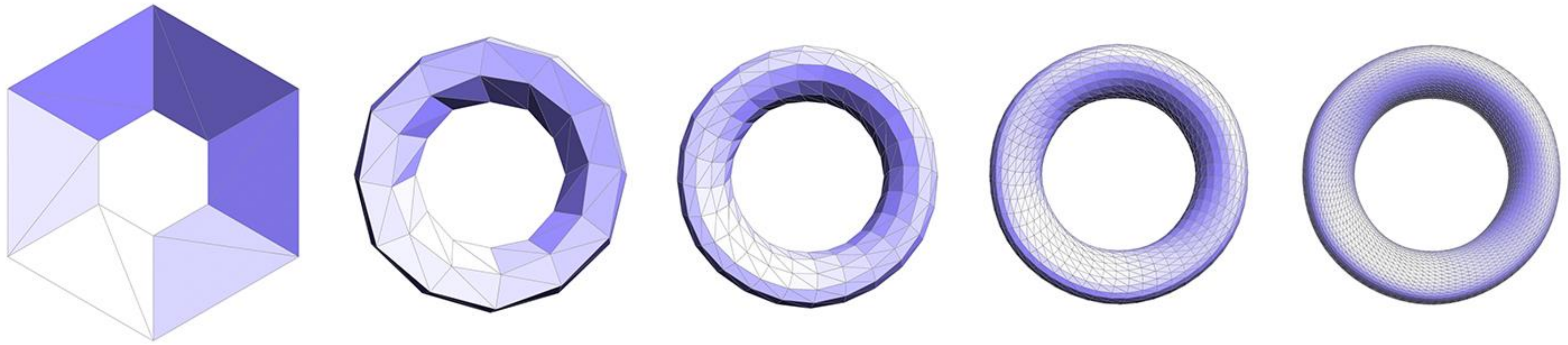


https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/thesis-10.pdf

https://graphics.stanford.edu/~mdfisher/subdivision.html

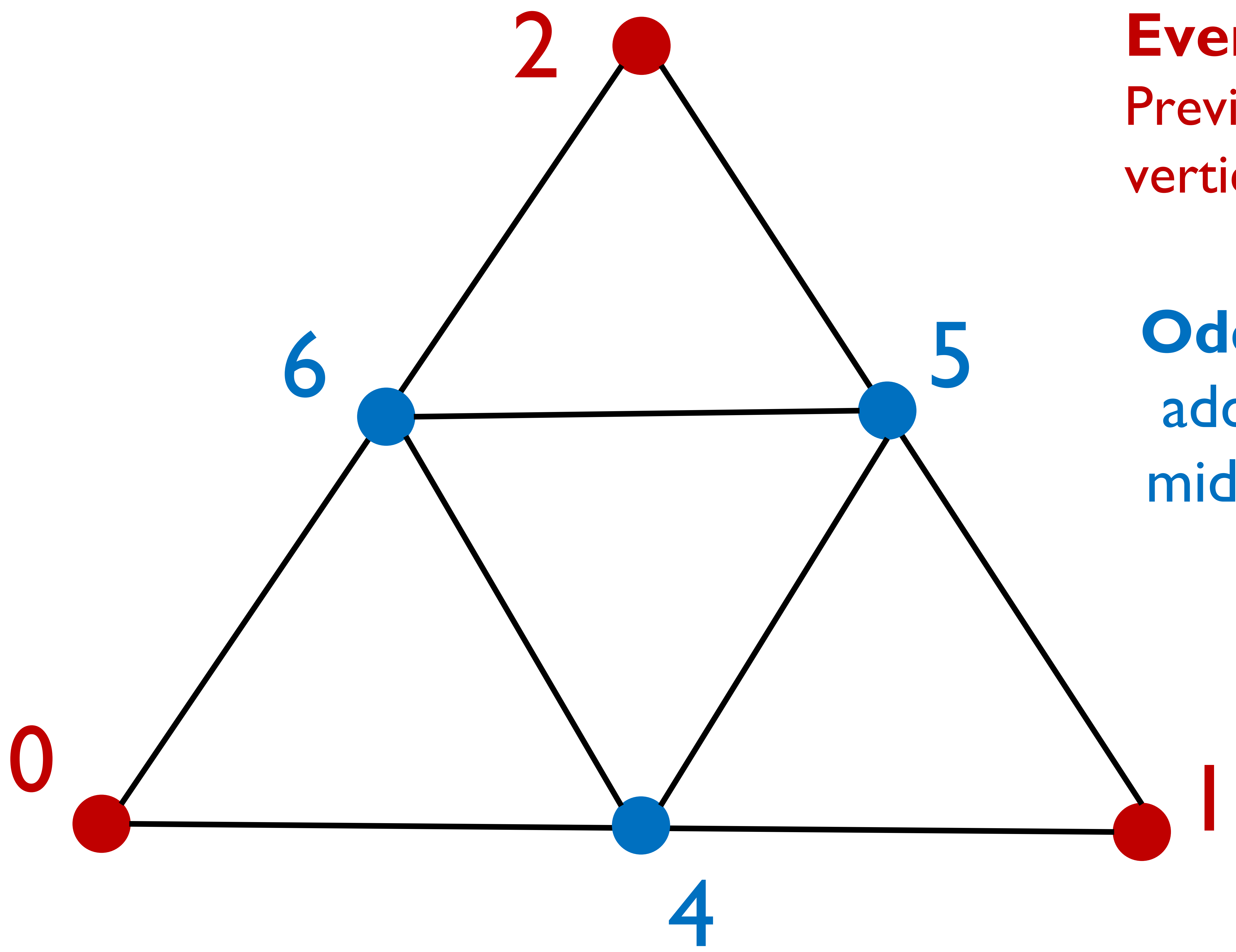
https://en.wikipedia.org/wiki/Loop_subdivision_surface

2019 Sci-Tech Awards- Charles Teorell Loop



Loop Subdivision: Algorithm Overview

- Key idea: split each triangle into four triangles



Even vertices: Previously existing vertices on the mesh

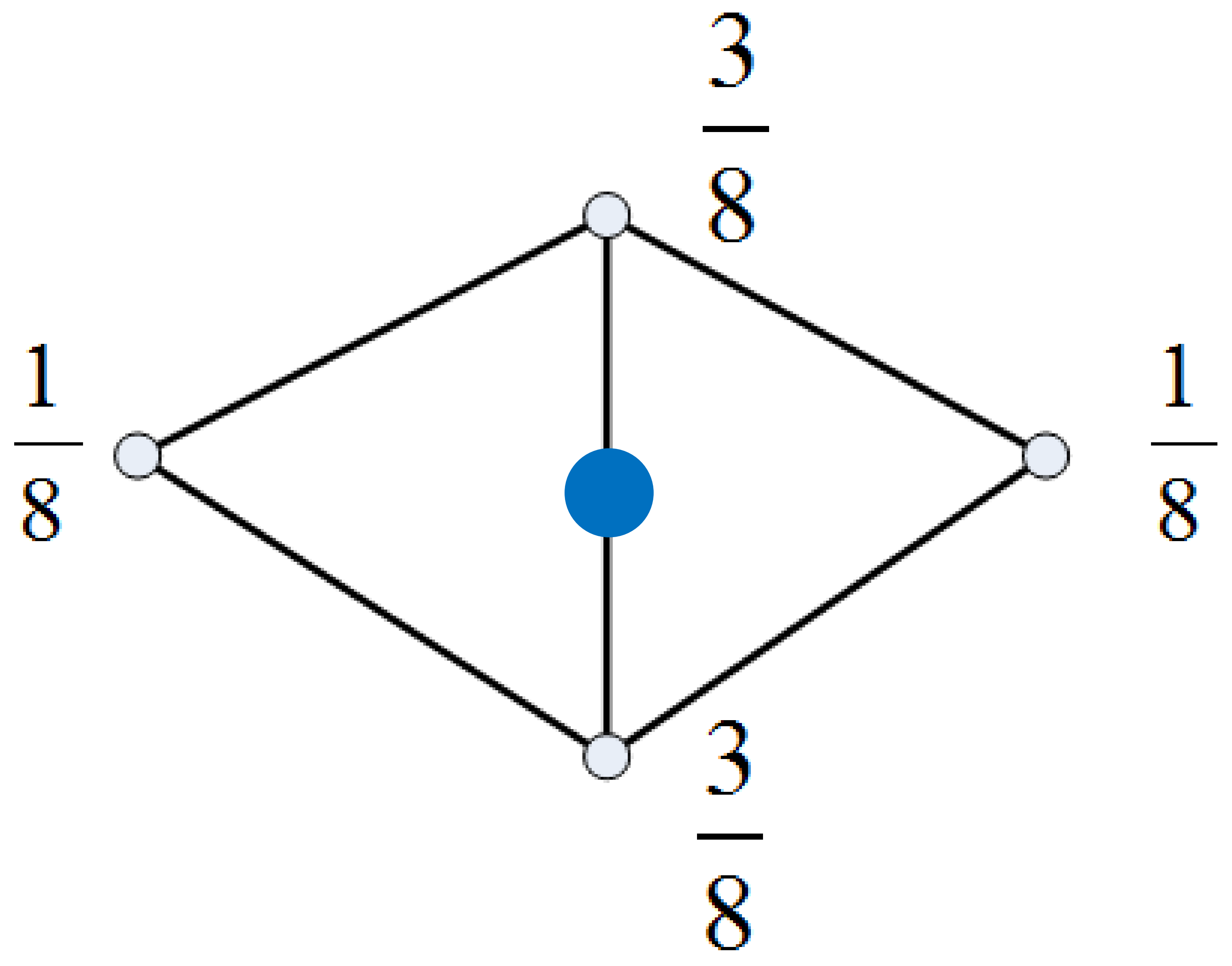
Odd vertices: Newly added vertices in the midpoint of each edge

Step 1 (Topology Update): Subdivide Each Triangle into 4 Triangles

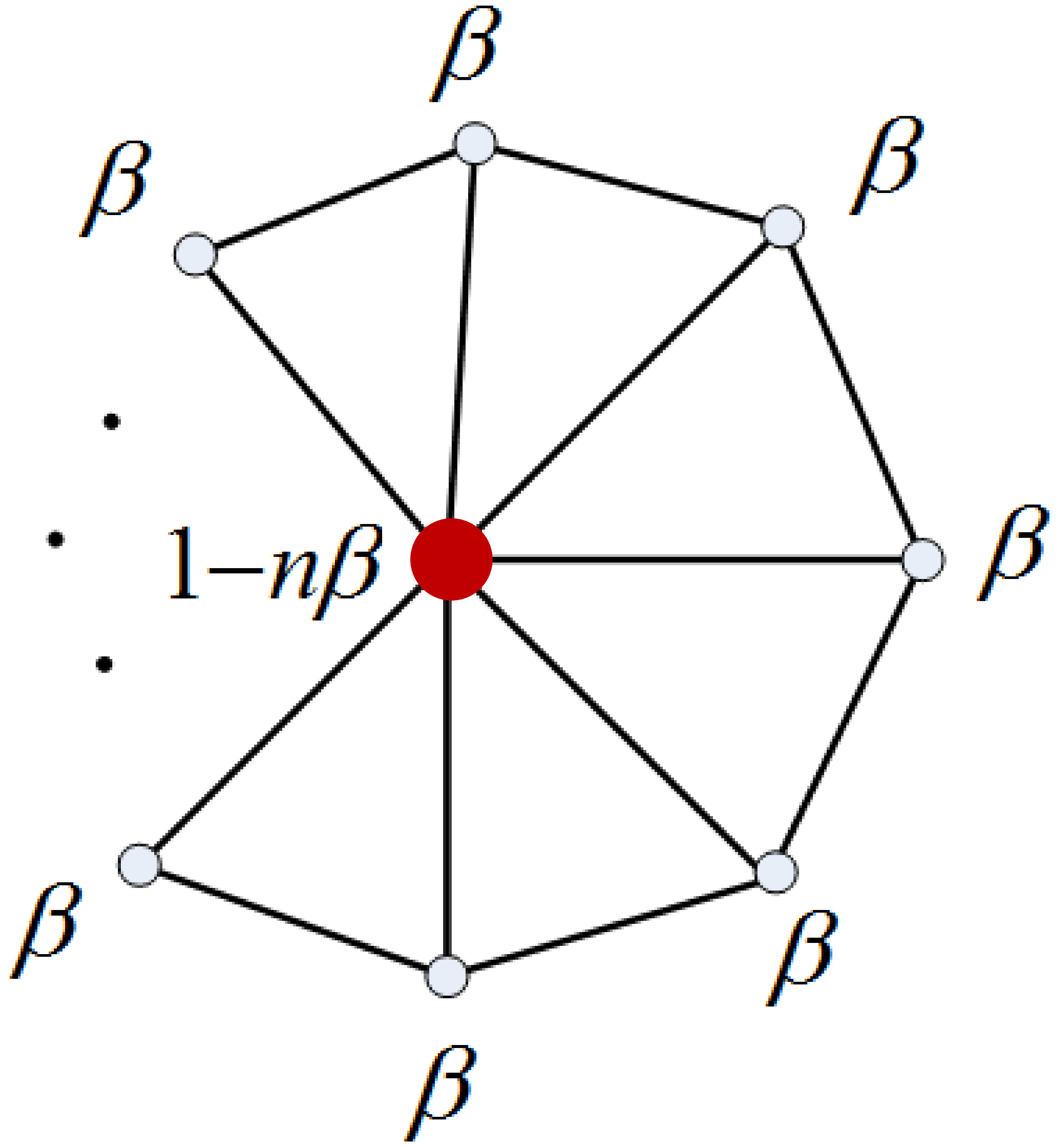
Even vertices **Odd vertices**

Step 2 (Geometry Smoothing): Smooth the Vertex Positions

- Key idea: calculate vertex position based on where it is originated

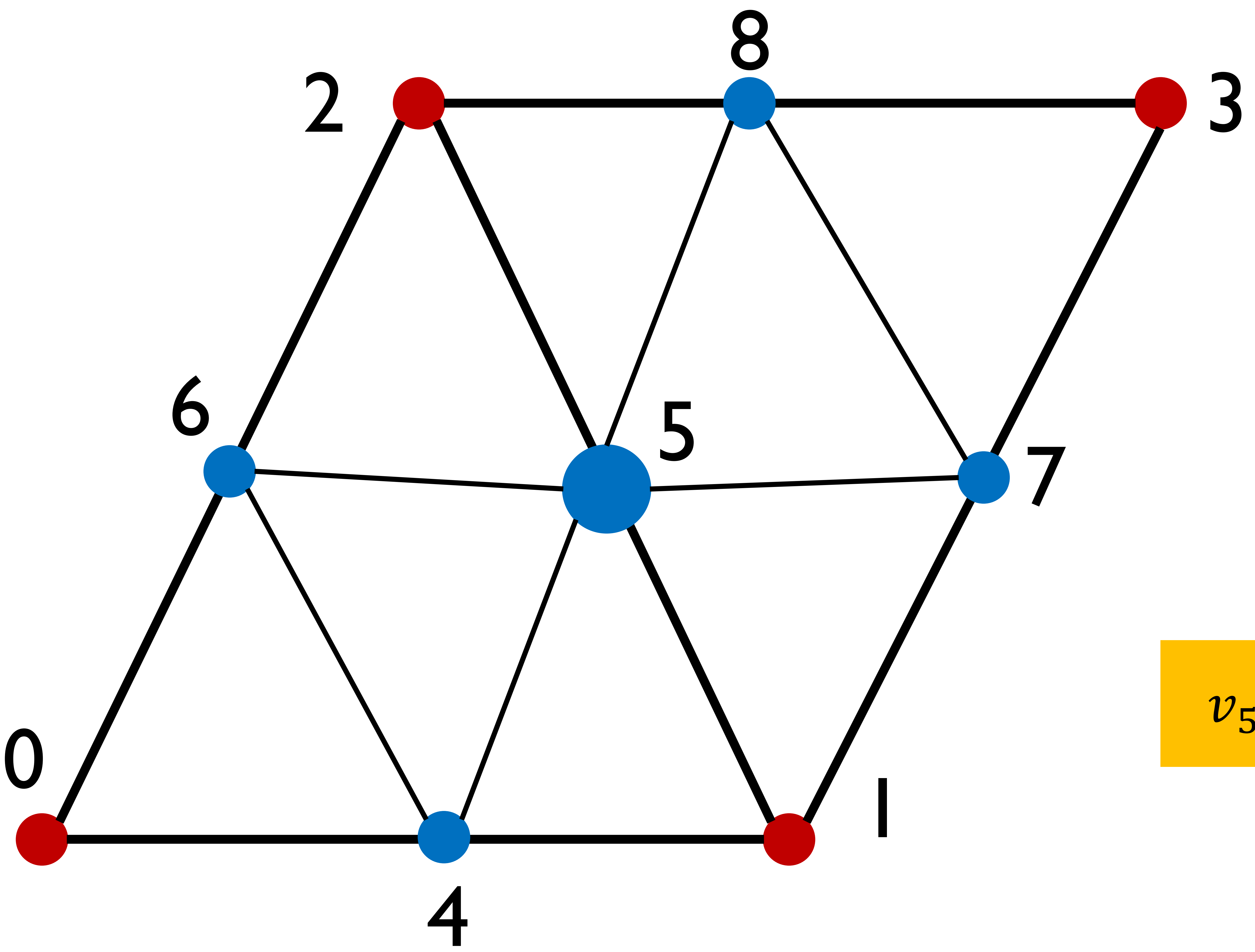


Case I: new (“**odd**”) vertices



Case II: inherited (“**even**”) vertices

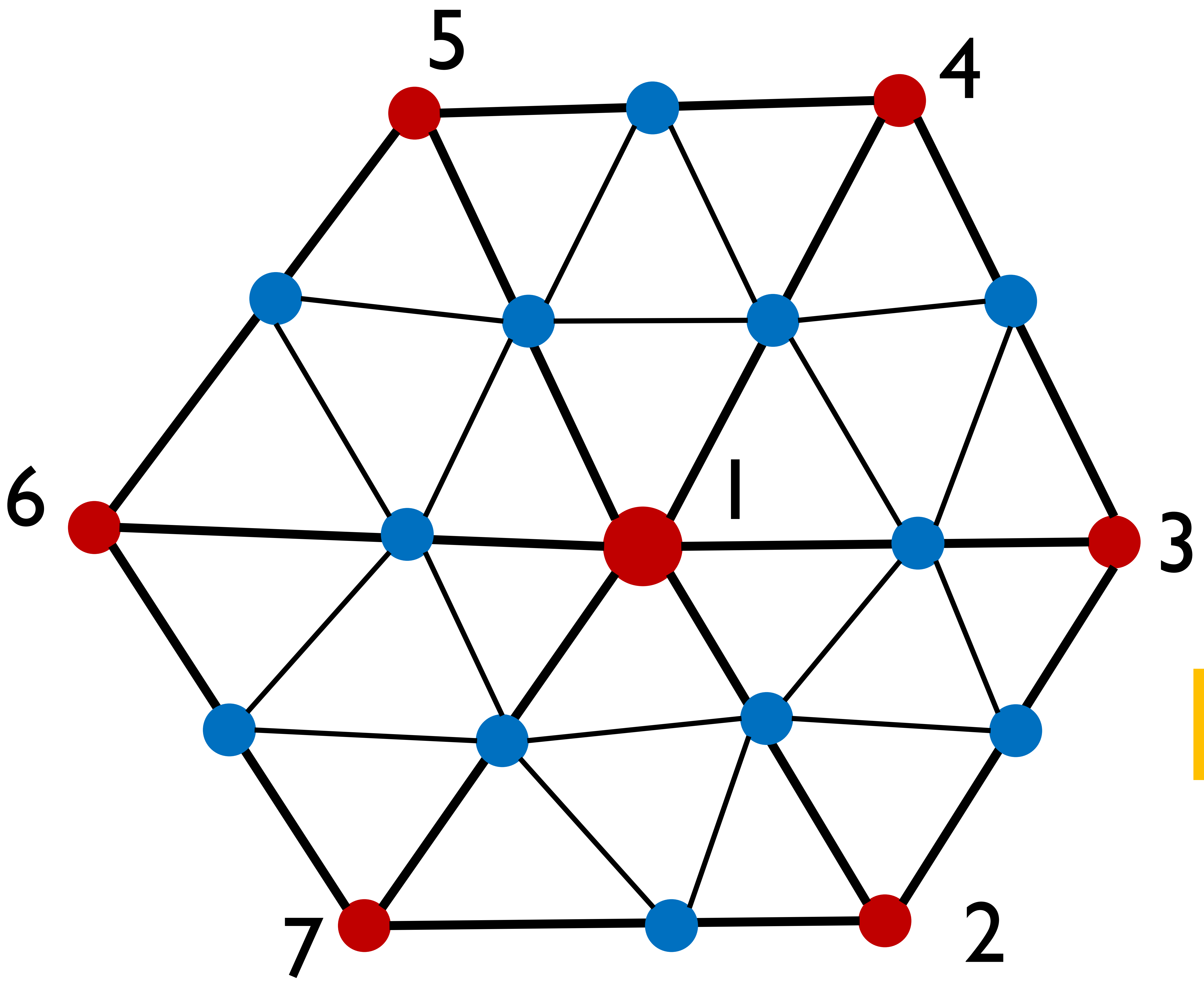
Case I: Update Odd Vertex Position



$$v_5 = \frac{1}{8}v_0 + \frac{1}{8}v_3 + \frac{3}{8}v_2 + \frac{3}{8}v_1$$

Notice: Odd vertices are calculated based on **even** vertices only

Case II: Update Even Vertex Position



$$\beta = \frac{1}{8}, n = 6$$

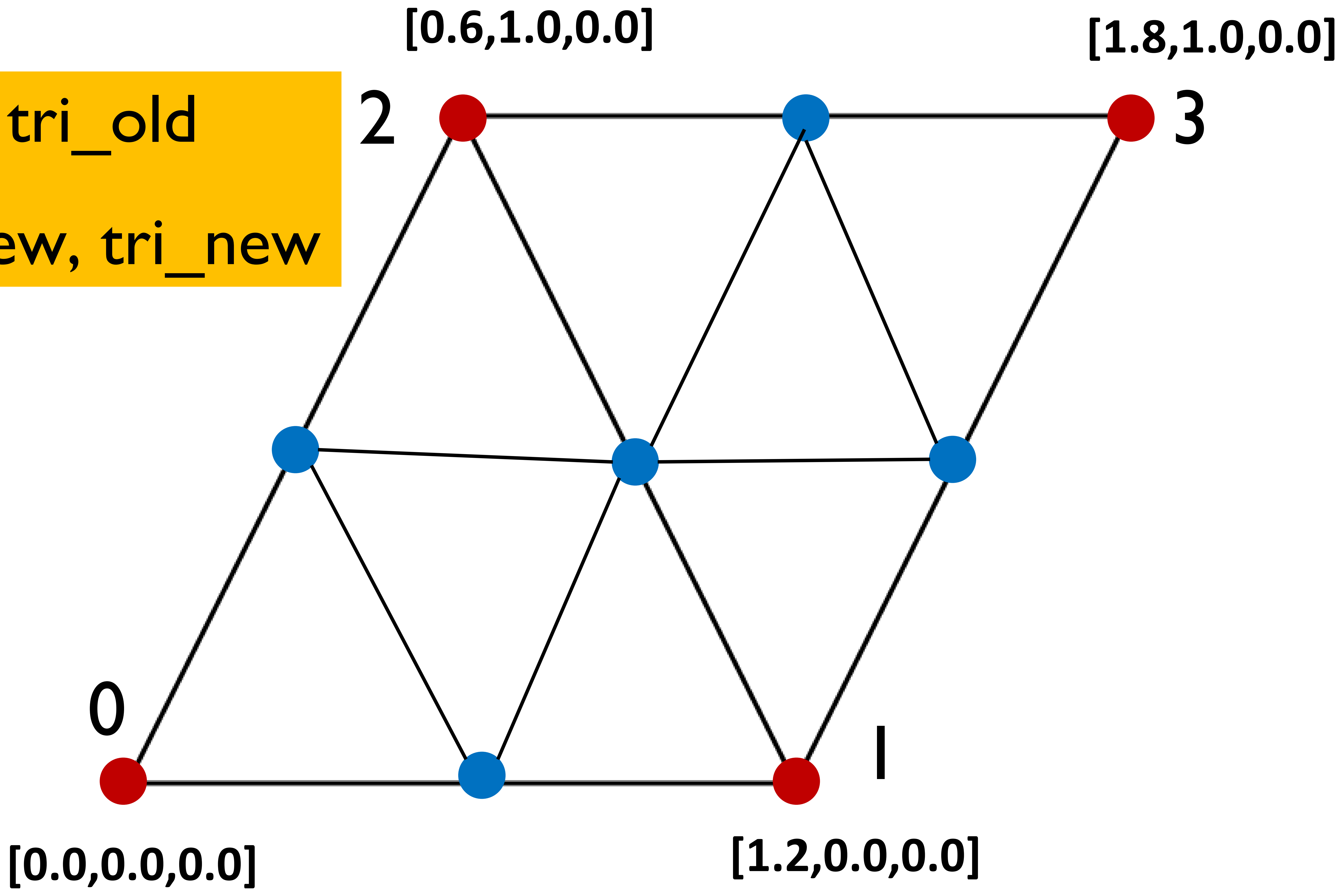
$$v_1 = \frac{2}{8}v_1 + \frac{1}{8}v_2 + \frac{1}{8}v_3 + \frac{1}{8}v_4 + \frac{1}{8}v_5 + \frac{1}{8}v_6$$

Notice: Even vertices are calculated based on neighboring **even** vertices only (including itself)

An Example

Input: vtx_old, tri_old

Output: vtx_new, tri_new



Programming Tips

- Make a **copy** of old vertices before updating any vertex positions
- All the vertex positions used on the right-hand side need to be **read from the copy**, rather than the currently updating vertex array
  - By doing this, we can avoid order-dependent results
- Create **auxiliary data structures** to avoid duplicating vertices
  - For each triangle edge on the old mesh, you can add one odd vertex only!

Auxiliary data structures

```
std::unordered_map<Vector2i,int> edge_odd_vtx_map;
```

```
std::unordered_map<Vector2i,std::vector<int> > edge_tri_map;
```

```
std::unordered_map<int,std::vector<int> > vtx_vtx_map;
```

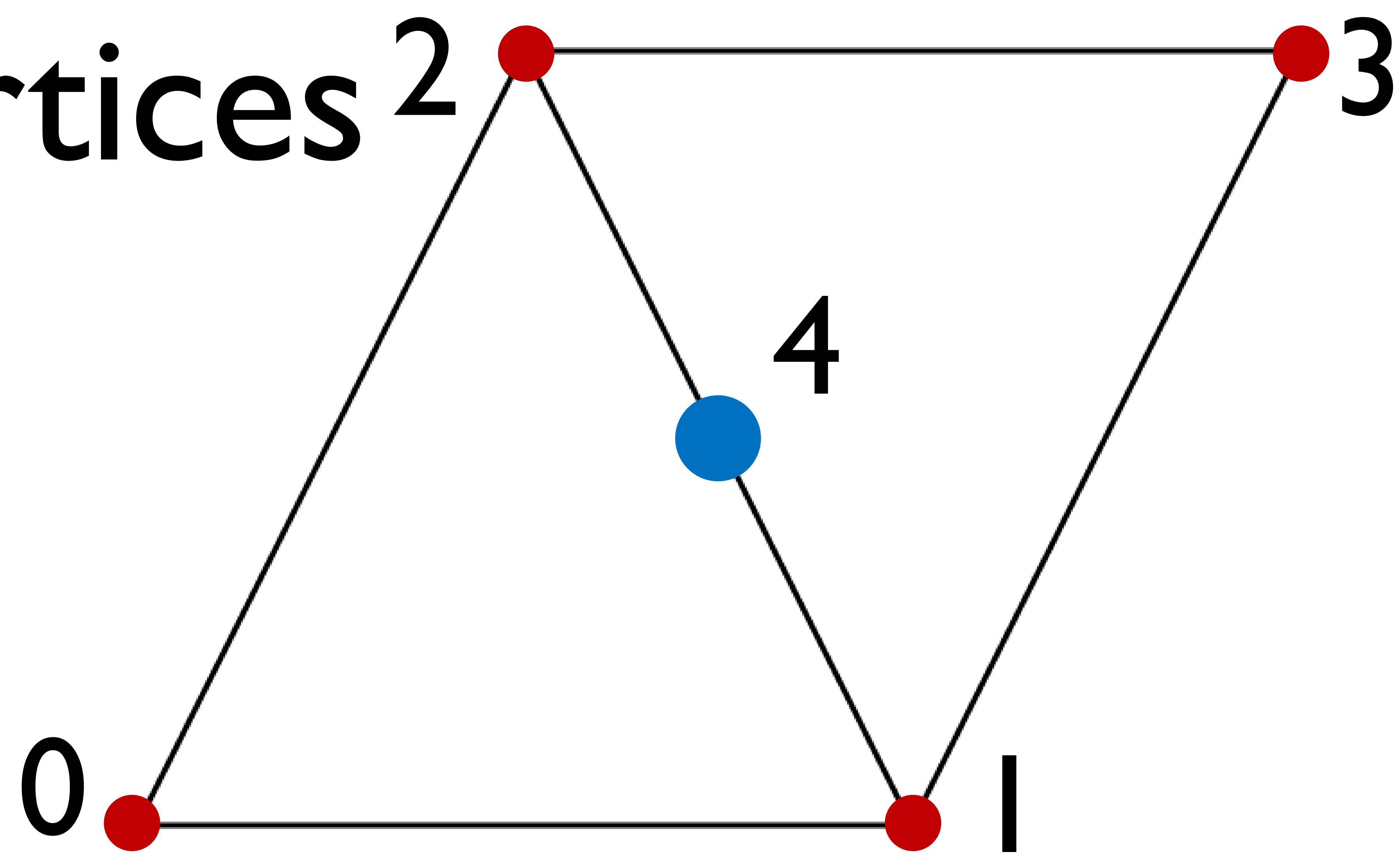
We need to build three auxiliary data structures to avoid duplicating vertices

Pseudocode

- Input: old_vtx, old_tri
- Output: new_vtx, new_tri
- Initialize: new_vtx=vtx; new_tri={empty}

  //// all copy operations are by value (instead of by reference)
- Algorithm:
  1. Update auxiliary data structures (edge_tri_map, vtx_vtx_map)
  2. Add all odd vertices to new_vtx and update auxiliary data structure (edge_odd_vtx_map)
  3. For each triangle in old_tri, add four triangles to new_tri
  4. Update odd vertex positions in new_vtx
  5. Update even vertex positions in new_vtx

Potential Pitfall: Adding Odd Vertices



vtx_new=vtx_old

For each triangle t

  Vector3i vtx=triangles[t]

  For each edge $e \in \{(vtx[0], vtx[1]), (vtx[1], vtx[2]), (vtx[2], vtx[0])\}$

    pos=.5*(vertices[e[0]]+vertices[e[1]])

    vtx_new.push_back(pos)

    ~~if(edge_mid_map.has(e))continue~~

    edge_odd_map[es]=vtx_new.size()-1

Vector2i es=Sorted(Vector2i(e[0],e[1]))
if(edge_mid_map.has(es))continue

How to implement Sorted()?

Potential Pitfall: Build edge_tri_map

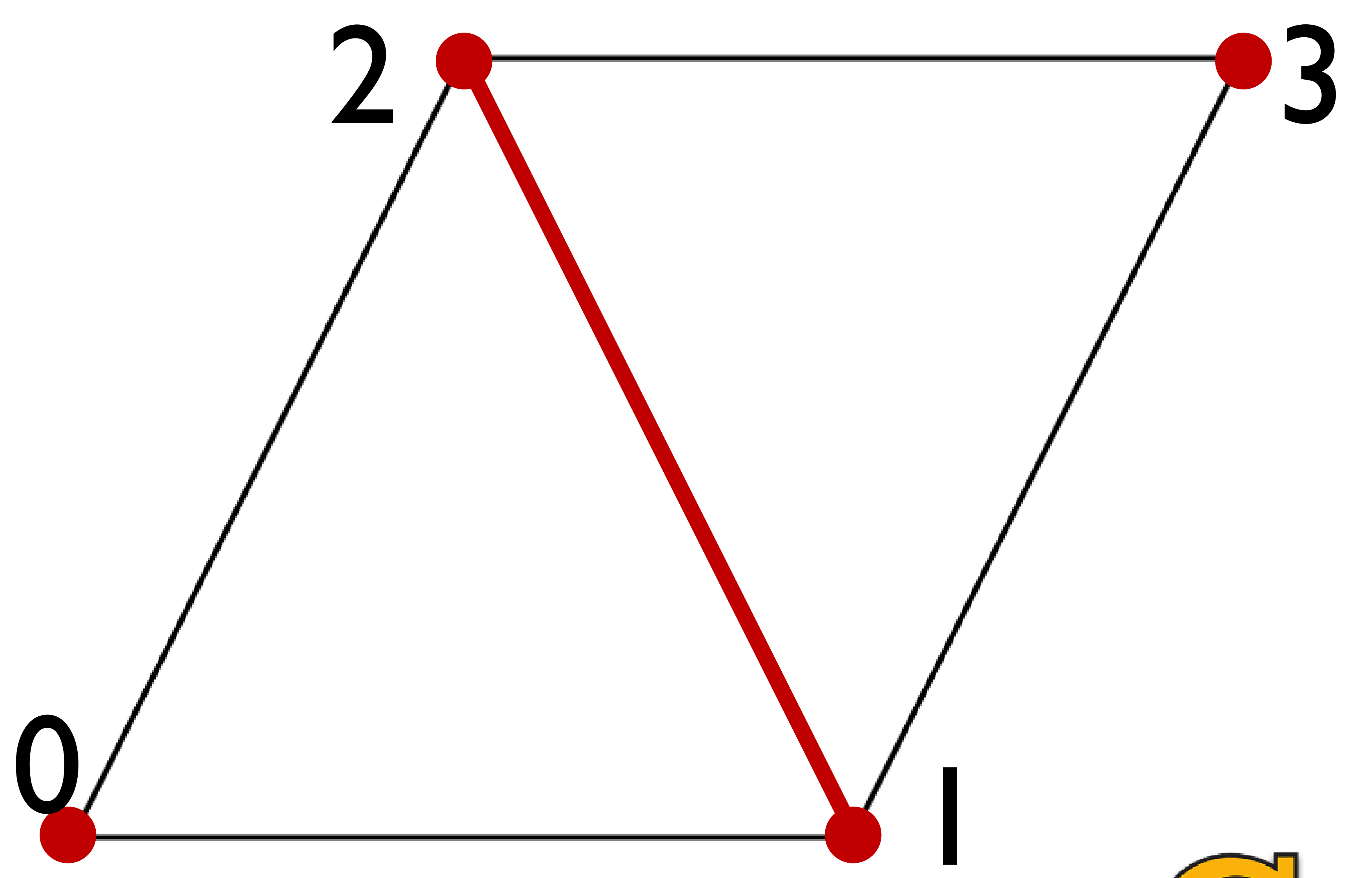
For each triangle t

Vector3i vtx=triangles[t]

For each edge $e \in \{(vtx[0], vtx[1]), (vtx[1], vtx[2]), (vtx[2], vtx[0])\}$

~~edge_tri_map[e]~~.push_back(t)

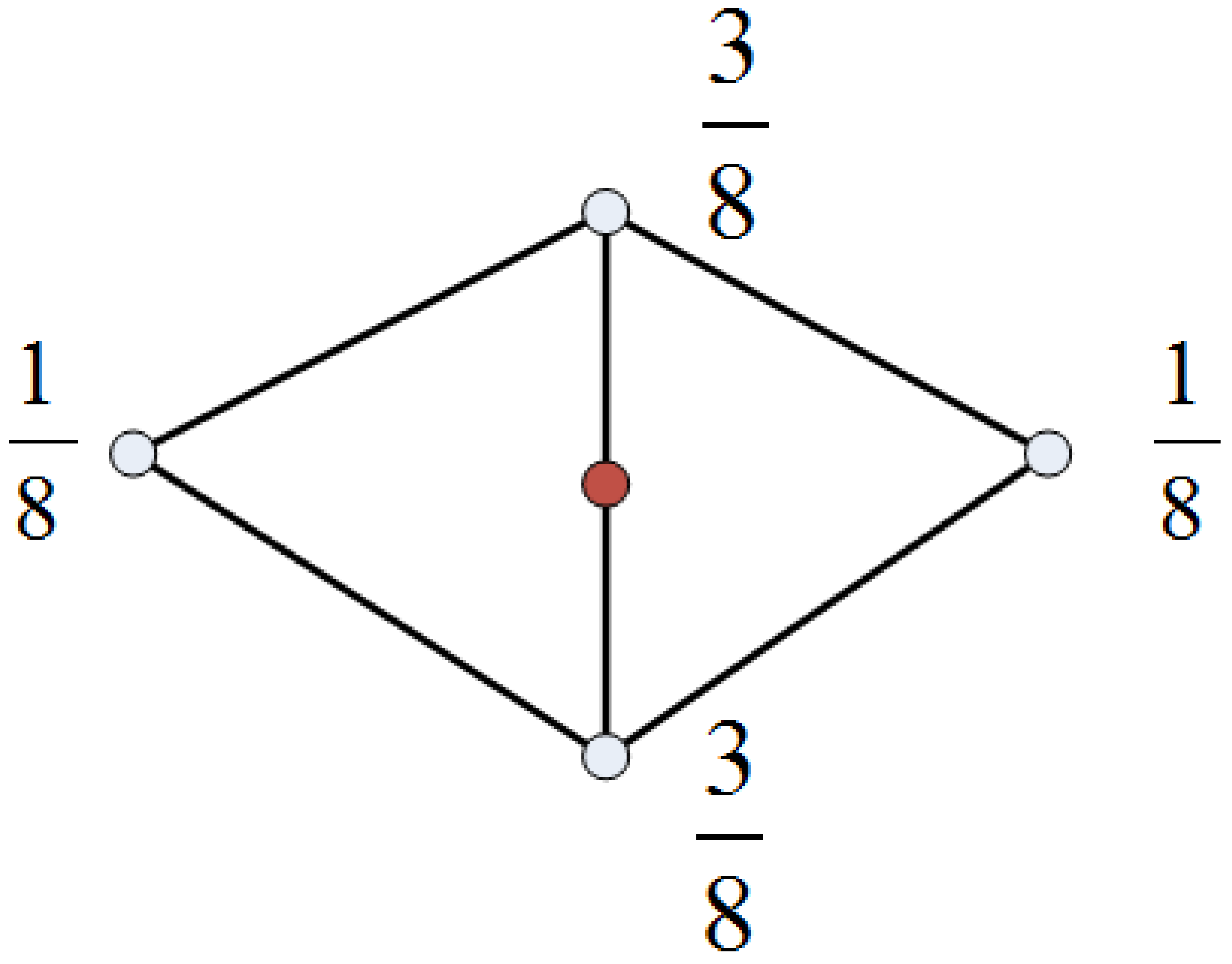
Vector2i es=Sorted(Vector2i(e[0],e[1]))
edge_tri_map[es].push_back(t)



Potential Pitfall: Updating Odd Vertices

For each edge e ← **e=Sorted(Vector2i(e[0],e[1]))**

```
odd_v=edge_odd_map[e]
t0=edge_tri_map[e][0]
t1=edge_tri_map[e][1]
opp_v0=find_opp_vtx(t0,e[0],e[1])
opp_v1=find_opp_vtx(t1,e[0],e[1])
vertices[odd_v]=weighted_average(e[0],e[1],opp_v0,opp_v1)
```



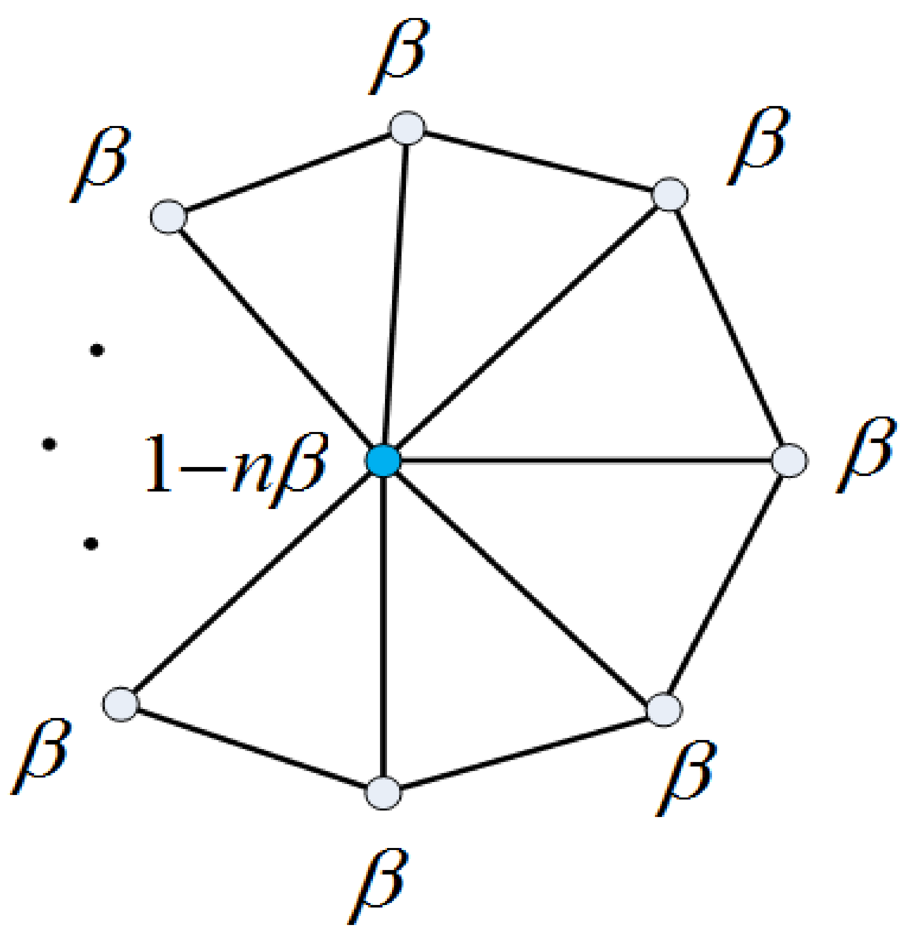
Potential Pitfall: Updating Even Vertices

Build vtx_vtx_map based on the old mesh

For each vertex v

smoothed_pos=weighted_average(pos of v and its nbs)

~~vertices[v]=smoothed_v~~



Record the smoothed positions in a new array and update them together after all the vertices are smoothed!

LIVE DEMO