Subdivision
Surface Subdivision

- Start with Polygon Mesh
- Refine mesh by creating new faces and vertices
- Repeat
Project Overview

Surface Subdivision

- Start with Polygon Mesh
- Refine mesh by creating new faces and vertices
- Repeat
Subdivision

• Many different algorithms
  o Approximating v. Interpolating
  o Face Splitting v. Vertex Splitting
  o Continuity properties of final surface
Subdivision

- Face split vs. Vertex split

Figure 4.1: Different refinement rules.
Loop Subdivision

- Approximating
- Face Splitting
- C2 continuity on regular meshes
Loop Subdivision

- Approximating
- Face Splitting
- C2 continuity on regular meshes
Loop Subdivision

- Approximating
- Face Splitting
- C2 continuity on regular meshes
Loop Subdivision

- Newly created vertices are called **odd vertices**

![Odd Vertices](image)
Loop Subdivision

- Newly created vertices are called **odd vertices**
- Original vertices are called **even vertices**
• But "Approximating" means we recompute positions of all vertices (**even** and **odd**)
Loop Subdivision

• But "Approximating" means we recompute positions of all vertices (even and odd)

http://ezekiel.vancouver.wsu.edu/~cs442/lectures/winged-edge/subdivision/subdivision.html
http://www.cs.dartmouth.edu/~fabio/teaching/graphics08/lectures/10_SubdivisionSurfaces_Web.pdf
Loop Subdivision

• Computing odd vertices

Interior:
\[ v = \frac{3.0}{8.0} (a + b) + \frac{1.0}{8.0} (c + d) \]

Boundary:
\[ v = \frac{1.0}{2.0} (a + b) \]

Notice that to compute \( v \) we need some to know the nearby vertices.
Loop Subdivision

• Computing **even** vertices

**Interior:**
\[ v = v \cdot (1 - k \cdot \beta) + \text{(sum of all } k \text{ neighboring vertices)} \cdot \beta \]

**Boundary:**
\[ v = \frac{1.0}{8.0} \cdot (a + b) + \frac{3.0}{4.0} \cdot v \]

Notice that to compute \( v \) we need to know all neighboring vertices.
Loop Subdivision - Picking Beta

Figure 4.3: Loop subdivision: in the picture above, $\beta$ can be chosen to be either $\frac{1}{n}(5/8 - (\frac{3}{8} + \frac{1}{4} \cos \frac{2\pi}{n})^2)$ (original choice of Loop [16]), or, for $n > 3$, $\beta = \frac{3}{8n}$ as proposed by Warren [33]. For $n = 3$, $\beta = 3/16$ can be used.
Loop Subdivision

• Computing odd vertices
• Computing even vertices

Important:
1. We need to be able to query adjacency information about the mesh.
2. We need to be able to tell if a vertex is a boundary or interior vertex.
Loop Subdivision

Algorithm (one iteration)

1. Build adjacency data structure
   *Tricky*

2. Compute odd vertices
   *Straightforward once you finish step 1.*

3. Compute even vertices
   *Straightforward once you finish step 1.*

4. Rebuild mesh / Connect vertices to create new faces
   *Similar to Project 1 (when you created a mesh from a heightmap)*
Adjacent Data Structure

What properties do you want?

• Efficient traversal and lookup
  - `get_adjacent_faces(&mesh, &edge)`
  - `get_neighbor_vertices(&mesh, &vertex)`

• Efficient memory usage

• Efficient creation and update
Adjacency Data Structure

What data do you need in the structure?

Mesh Data
• Some combination of Vertices, Faces, Edges
• Adjacency information

Loop Subdivision Metadata
• implicit
  o all edges of index < i have been subdivided
• explicit
  o if (!mesh.edge[i].is_subdivided) ...
Adjacency Data Structure

Useful Mesh Attributes

• Every triangle has 3 vertices
• Every triangle is adjacent to up to 3 other triangles
Adjacency Data Structure

Useful Mesh Attributes

• Every triangle has 3 vertices
• Every triangle is adjacent to up to 3 other triangles
• A given vertex has N neighbor vertices
• The same vertex is part of either N-1 or N triangles
  o Why?
  o There is a useful implication of this for Loop Subdivision
Adjacency Data Structure

Useful Adjacency Attributes

• Triangle -> Vertex
• Triangle -> Triangle
• Vertex -> Vertex
• Vertex -> Triangle

This is a simple representation that can handle the queries you need.
Adjacency Data Structure

Implementation

• How you implement (storing and building) the adjacency data structure can be more important than what you represent.

• Stick to C data structures (arrays and structs) for the best speed

• Be mindful that `malloc/new` and `free/delete` are slow
Other Subdivision Algorithms

• Modified Butterfly: interpolating algorithm

Figure 4.5: Modified Butterfly subdivision. The coefficients $s_i$ are $\frac{1}{k} \left( \frac{1}{4} + \cos \frac{2\pi}{k} + \frac{1}{2} \cos \frac{4\pi}{k} \right)$ for $k > 5$. For $k = 3$, $s_0 = \frac{5}{12}$, $s_{1,2} = -\frac{1}{12}$; for $k = 4$, $s_0 = \frac{3}{8}$, $s_2 = -\frac{1}{8}$, $s_{1,3} = 0$. 
Other Subdivision Algorithms

- Modified Butterfly: interpolating algorithm

Figure 4.5: Modified Butterfly subdivision. The coefficients $s_i$ are $\frac{1}{k} \left( \frac{1}{4} + \cos \frac{2\pi}{k} + \frac{1}{2} \cos \frac{4\pi}{k} \right)$ for $k > 5$.
For $k = 3$, $s_0 = \frac{5}{12}$, $s_{1,2} = -\frac{1}{12}$; for $k = 4$, $s_0 = \frac{3}{8}$, $s_2 = -\frac{1}{8}$, $s_{1,3} = 0$. 
Other Subdivision Algorithms

- **Catmull-Clark**: approximating

![Diagram showing masks for face, edge, and boundary vertices]

**Figure 4.8**: Catmull-Clark subdivision. Catmull and Clark [4] suggest the following coefficients for rules at extraordinary vertices: $\beta = \frac{3}{8}$ and $\gamma = \frac{1}{8}$.
Other Subdivision Algorithms

- Kobbelt: approximating

Figure 4.11: Kobbelt subdivision.
Figure 4.20: Different subdivision schemes produce similar results for smooth meshes.