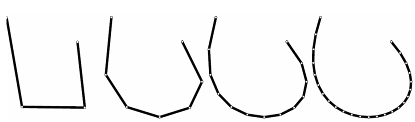
Subdivision surfaces

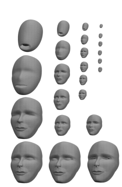
Subdivision surfaces

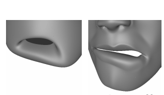
- The goal is to create smooth surfaces out of arbitary meshes.
- Polygons are great, but it is hard to handle many polygons for a modeller.
- A simple Line case, after 3 subdivisions, the curve is smooth.



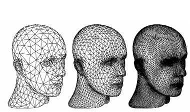
Subdivision surfaces

- Can be used to create models with different resolutions: Level Of Detail (LOD)
- When doing animation, controlling a courser mesh can be easier:

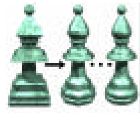




Low-res effect High-res effect



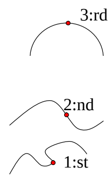
Subdivision surfaces



- Applying a subdivision scheme on a mesh makes the mesh smoother just out of the connectivity of the original mesh.
- The general process of subdivision has the form:

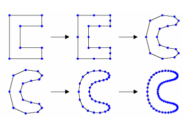
$$p^k = Sp^{k-1}$$

- Where P^{k-1} is the original mesh, S is the *smooth operator* and P^k is the resulting mesh from applying the smooth operator S onto P^{k-1}.
- Subdivision is a sort of SPLINES.
- They can be proven to have C^1 continuity



Subdivision for lines

- Line example:
- Given a polygonal curve p^k the i:th vertex of the curve is p^k_i an edge is given between two consecutive vertices p^k_i and p^k_{i-1}



- The subdivision (averaging) rules are then:

$$\begin{aligned} p^k_{2i} &= \tfrac{1}{8} p^{k-1}_{i-1} + \tfrac{3}{4} p^{k-1}_i + \tfrac{1}{8} p^{k-1}_{i+1}, \\ p^k_{2i+1} &= \tfrac{1}{2} p^{k-1}_i + \tfrac{1}{2} p^{k-1}_{i+1} \end{aligned}$$

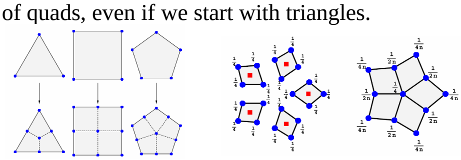
Subdivision

- Subdivision can be divided into two passes:
  - Factorization pass
    - A linear subdivision pass
    - Adding new midpoint vertices
  - Averaging pass
    - Smoothing pass
    - Moving the vertices according to the weighting rules

Subdivision for surfaces

Subdivision on surfaces

- n - Valence
  - How many neighbour vertices a given vertex p^k_i has.
- Catmul-Clark Scheme (Ed Catmul, John Clark) 1978
- Operates on quads. The result will alway be a set of quads, even if we start with triangles.



Efficient datastructures

- Common questions on a mesh during execution of a subdivision surface scheme is:
  - Which faces use this vertex?
  - Which edges use this vertex?
  - Which faces border this edge?
  - Which edges border this face?
  - Which faces are adjacent to this face?
- These questions can be very time consuming if the underlying datastructure doesnt support them efficiently.
- Fortunately there are a few solutions:
  - Half-Edge
  - Winged-Edge
- I recomend to do a google search on Half-Edge (flipcode has one with code, linked from the project web page)

Half Edge

```
class Vertex {
    public:

    Vector3D m_coord;
    Vector3D m_normal;
    Edge* m_edge;  // one of the half-edges emantating from the vertex


};

class Edge {
    public:

    Face* m_face;   // face the half-edge borders
    Vertex* m_start;   // vertex at the start of the half-edge
    Vertex* m_end;   // vertex at the start of the half-edge

    Edge* m_prev;   // Previous half-edge
    Edge* m_opp;   // oppositely oriented adjacent half-edge
    Edge* m_next;   // next half-edge around the face


    bool isOpposite(const Edge &edge) {

          return (m_start == edge.m_end &&
          m_end == edge.m_start);
    }

};

class Face {
    Edge* m_edge;  // one of the half-edges bordering the face

    Vector3D m_normal;

};
```