Polygon Meshes

Peter Palfrader

University of Salzburg

July 2011

Outline

1. Introduction
2. Data Structures
3. Model Repair
4. Mesh quality and smoothness
5. Subdivision Surfaces
6. Further Reading

Outline

1 Introduction

2 Data Structures

3 Model Repair

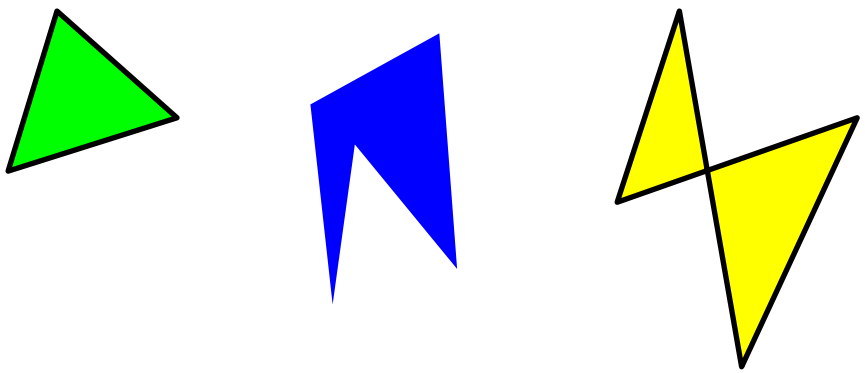
4 Mesh quality and smoothness

5 Subdivision Surfaces

6 Further Reading

Polygon

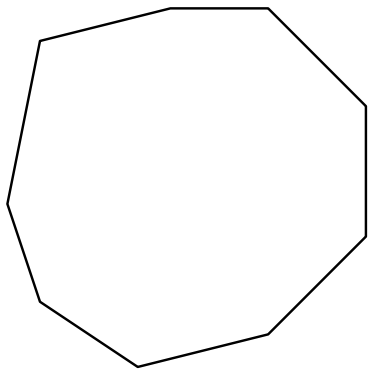
- polygon: a closed figure in the plane, bounded by straight line segments.



- Defined by vertices and edges.

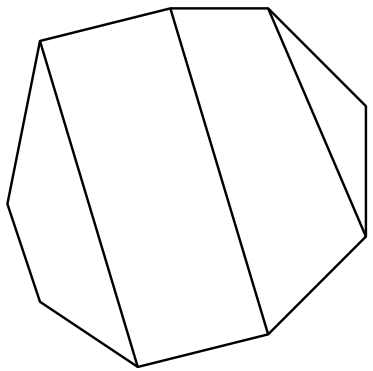
Mesh

- polygon mesh: a set of polygons
- triangle mesh: mesh whose polygons are all triangles



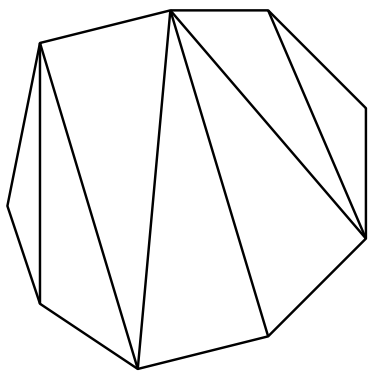
Mesh

- polygon mesh: a set of polygons
- triangle mesh: mesh whose polygons are all triangles



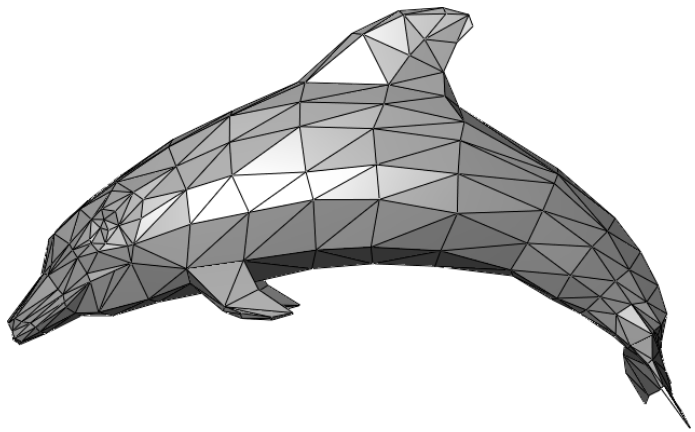
Mesh

- polygon mesh: a set of polygons
- triangle mesh: mesh whose polygons are all triangles



Mesh in 3D

- If not all the vertices are confined to a single plane, meshes can be used to represent 3D-objects (polyhedra).



[image: wikipedia/Chrschn; PD]

Outline

1 Introduction

2 Data Structures

3 Model Repair

4 Mesh quality and smoothness

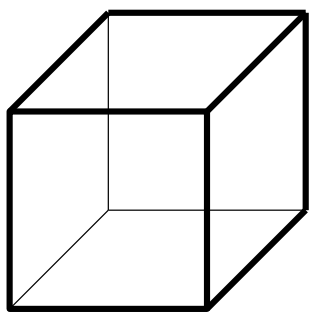
5 Subdivision Surfaces

6 Further Reading

Elements

Polygon meshes consist of:

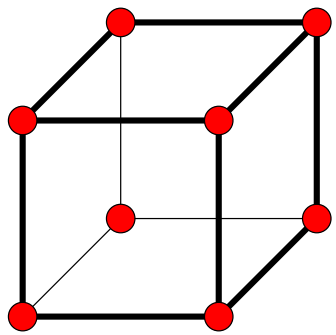
- vertices
- edges
- faces



Elements

Polygon meshes consist of:

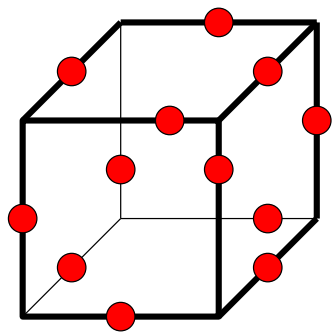
- vertices
- edges
- faces



Elements

Polygon meshes consist of:

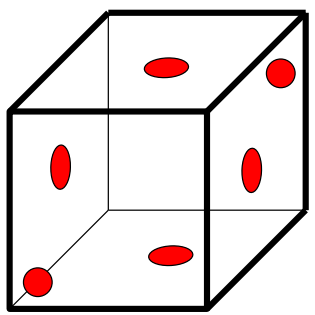
- vertices
- edges
- faces



Elements

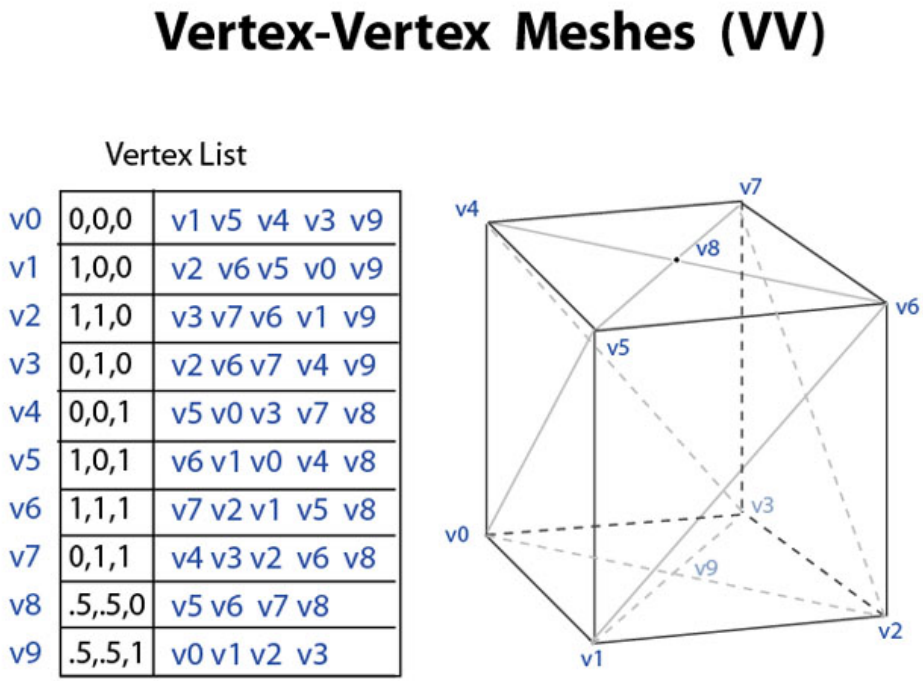
Polygon meshes consist of:

- vertices
- edges
- faces



Vertex-Vertex mesh, or VV mesh

- For each vertex store an ordered list of its neighbors.
- Very simple.
- Unfortunately not all that convenient either.



[image: wikipedia/Rchoetzlein; CC BY-SA 3.0]

Operations on meshes

- Access to individual faces, edges, vertices.
- Enumerate faces, edges, vertices.
- Oriented traversal of the edges/vertices of a face.
- Access to incident faces of an edge.
- Given an edge, access to its endpoint vertices.
- Given a vertex, find one or all incident faces/edges.
- Find the *one-ring neighborhood* of a vertex.

[PMP]

Face-Based Data Structure

Triangle Soup:

- Basic Concept: Store just an unstructured set of faces.

Triangles		
$x_{1,1}, y_{1,1}, z_{1,1}$	$x_{1,2}, y_{1,2}, z_{1,2}$	$x_{1,3}, y_{1,3}, z_{1,3}$
$x_{2,1}, y_{2,1}, z_{2,1}$	$x_{2,2}, y_{2,2}, z_{2,2}$	$x_{2,3}, y_{2,3}, z_{2,3}$
...		
$x_{n,1}, y_{n,1}, z_{n,1}$	$x_{n,2}, y_{n,2}, z_{n,2}$	$x_{n,3}, y_{n,3}, z_{n,3}$

- slightly better: indexed face-set:

Triangles		
$v_{1,1}$	$v_{1,1}$	$v_{1,1}$
$v_{2,1}$	$v_{2,1}$	$v_{2,1}$
...		
$v_{F,1}$	$v_{F,1}$	$v_{F,1}$

Vertices		
x_1	y_1	z_1
x_2	y_2	z_2
...		
x_V	y_V	z_V

Face-Based Data Structure

Triangle Soup:

- Basic Concept: Store just an unstructured set of faces.

Triangles		
$x_{1,1}, y_{1,1}, z_{1,1}$	$x_{1,2}, y_{1,2}, z_{1,2}$	$x_{1,3}, y_{1,3}, z_{1,3}$
$x_{2,1}, y_{2,1}, z_{2,1}$	$x_{2,2}, y_{2,2}, z_{2,2}$	$x_{2,3}, y_{2,3}, z_{2,3}$
...		
$x_{n,1}, y_{n,1}, z_{n,1}$	$x_{n,2}, y_{n,2}, z_{n,2}$	$x_{n,3}, y_{n,3}, z_{n,3}$

- slightly better: indexed face-set:

Triangles		
$v_{1,1}$	$v_{1,1}$	$v_{1,1}$
$v_{2,1}$	$v_{2,1}$	$v_{2,1}$
...		
$v_{F,1}$	$v_{F,1}$	$v_{F,1}$

Vertices		
x_1	y_1	z_1
x_2	y_2	z_2
...		
x_V	y_V	z_V

Triangle Soup, cont'd

- Simple.
- Storage/memory efficient.
- Several queries/actions are quite cumbersome:
  - Iteration over all vertices (in the basic variant).
  - Iteration over all edges.
  - What are the faces incident at a vertex?
  - What are the faces incident at an edge?

Triangle Soup, cont'd

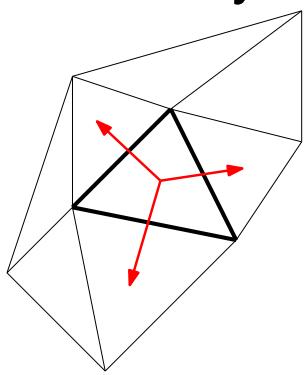
- Simple.
- Storage/memory efficient.
- Several queries/actions are quite cumbersome:
  - Iteration over all vertices (in the basic variant).
  - Iteration over all edges.
  - What are the faces incident at a vertex?
  - What are the faces incident at an edge?

Triangle Soup with connectivity information on the side

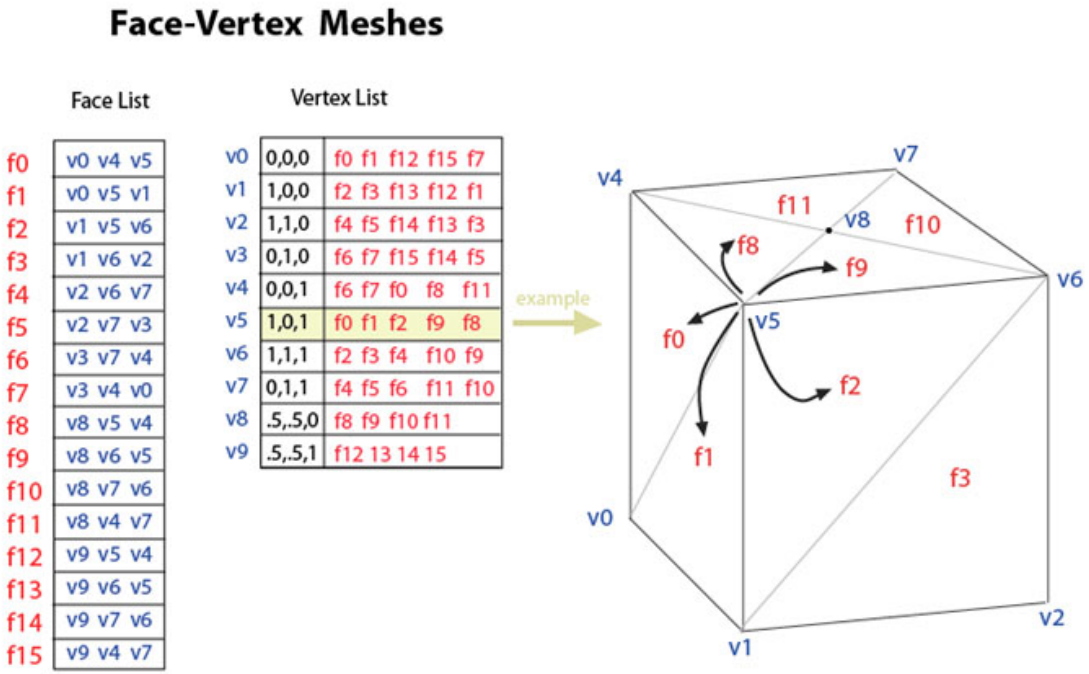
- Augment indexed face-set with connectivity information:

Face	
`Vertex*`	`v[3]`
`Face*`	`neighbor[3]`

Vertex	
`Point`	`pos`
`Face*`	`f`



Face Vertex Mesh



[image: wikipedia/Rchoetzlein; CC BY-SA 3.0]

Face Based, cont'd

- No explicit edge representation.
- Does not work so well when we have not only triangles.

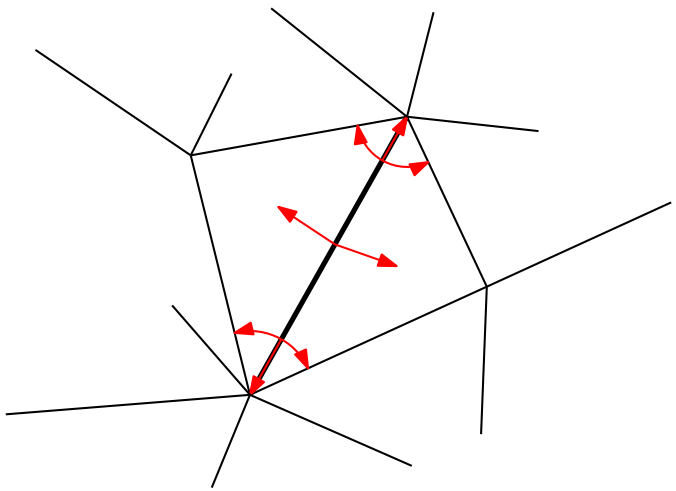
Edge-based Data Structure: Winged Edge

- Edges are first class citizens:

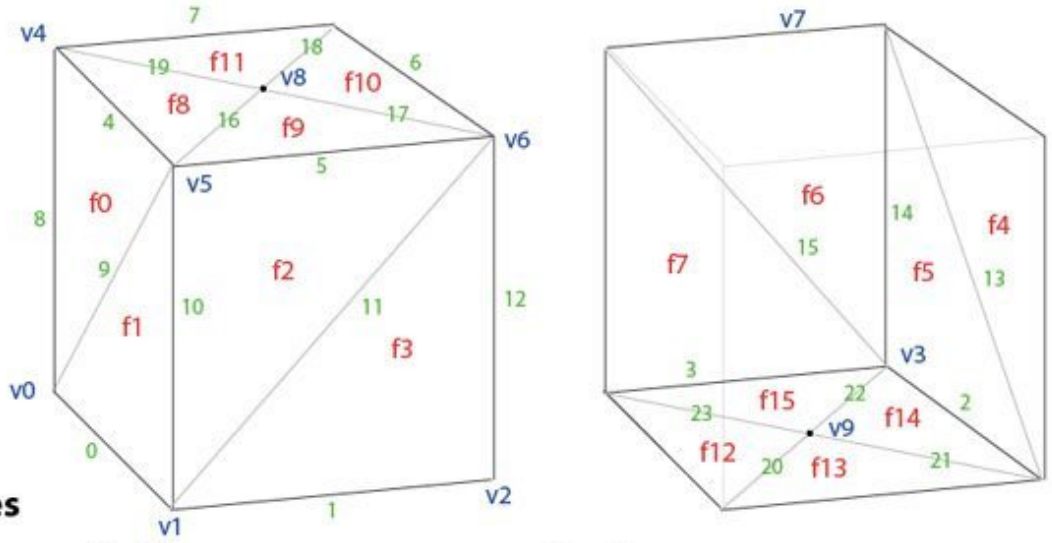
Edge	
`Vertex*`	`v[2]`
`Face*`	`f[2]`
`Edge*`	`next[2]`
`Edge*`	`prev[2]`

Vertex	
`Point`	`pos`
`Edge*`	`e`

Face	
`Edge*`	`e`



Winged Edge Mesh



Winged-Edge Meshes

Face List

f0	4
f1	0
f2	5
f3	1
f4	6
f5	2
f6	7
f7	3
f8	4
f9	5

Edge List

e0	v0 v1	f1 f12	9 23 10 20
e1	v1 v2	f3 f13	11 20 12 21
e2	v2 v3	f5 f14	13 21 14 22
e3	v3 v0	f7 f15	15 22 8 23
e4	v4 v5	f0 f8	19 8 16 9
e5	v5 v6	f2 f9	16 10 17 11
e6	v6 v7	f4 f10	17 12 18 13
e7	v7 v4	f6 f11	18 14 19 15
e8	v0 v4	f7 f0	3 9 7 4
e9	v0 v5	f0 f1	8 0 4 10

Vertex List

v0	0,0,0	8
v1	1,0,0	10
v2	1,1,0	12
v3	0,1,0	14
v4	0,0,1	8
v5	1,0,1	10
v6	1,1,1	12
v7	0,1,1	14
v8	.5,.5,0	16
v9	.5,.5,1	20

[image: wikipedia/Rchoetzlein; CC BY-SA 3.0]

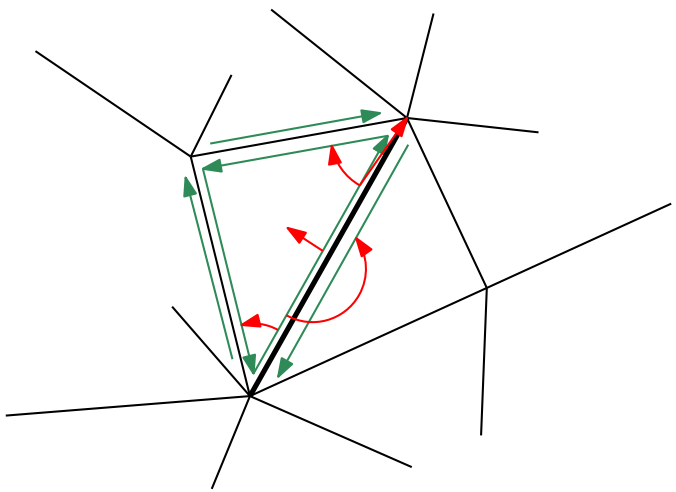
Half Edges

- Walking around in any of the previous data structures still requires distinguishing between different cases. A “half-edge“ approach can avoid this necessity.
- Every (unoriented) edge gets split into two (oriented) half edges.

HalfEdge	
Vertex*	v
Face	f
Edge*	next
Edge*	prev
Edge*	buddy

Vertex	
Point	pos
HalfEdge*	e

Face	
HalfEdge*	e



Outline

1 Introduction

2 Data Structures

3 Model Repair

4 Mesh quality and smoothness

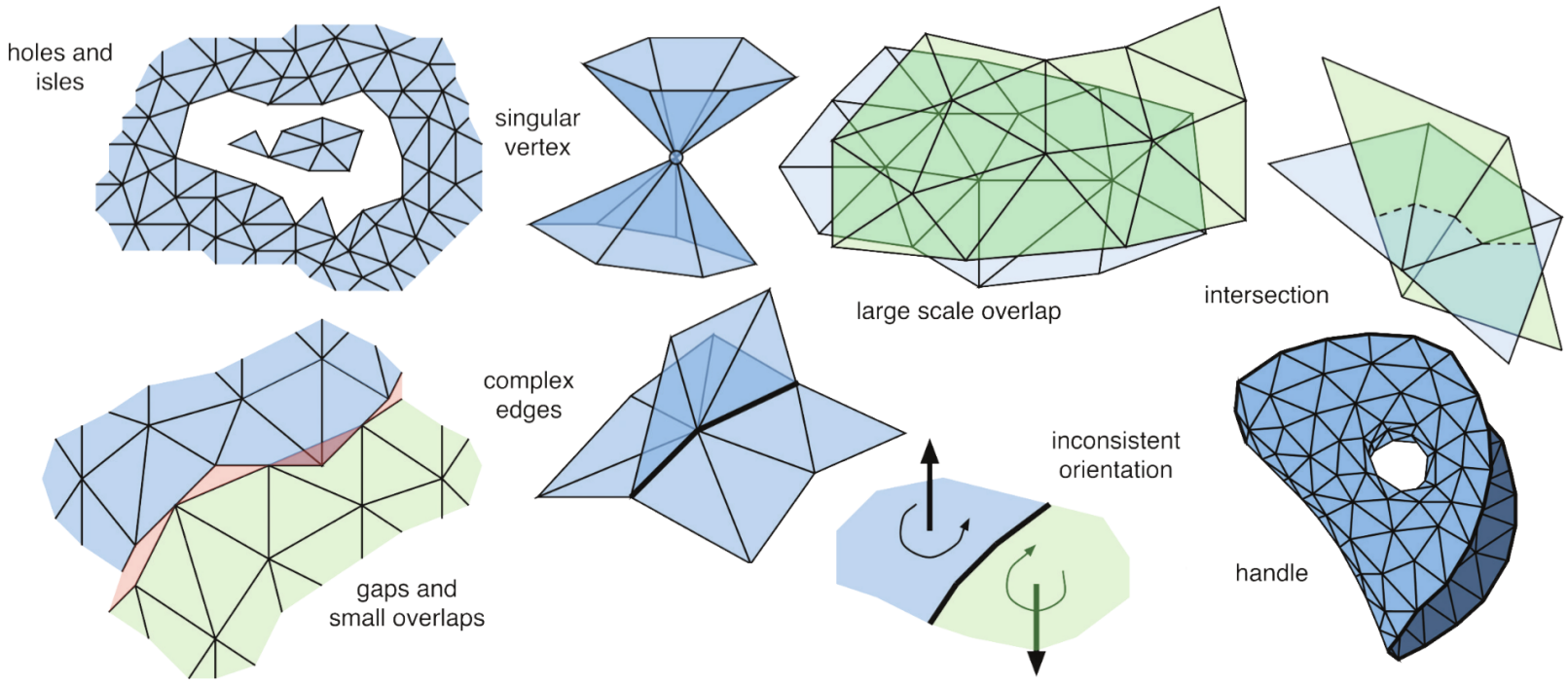
5 Subdivision Surfaces

6 Further Reading

Model Repair

Model repair: the process of removing artifacts from a geometric model in order to generate an output model suitable for further processing by downstream applications that require certain quality guarantees for their input.[PMP]

Freak Show



[image: [PMP]]

Surface Oriented Algorithms

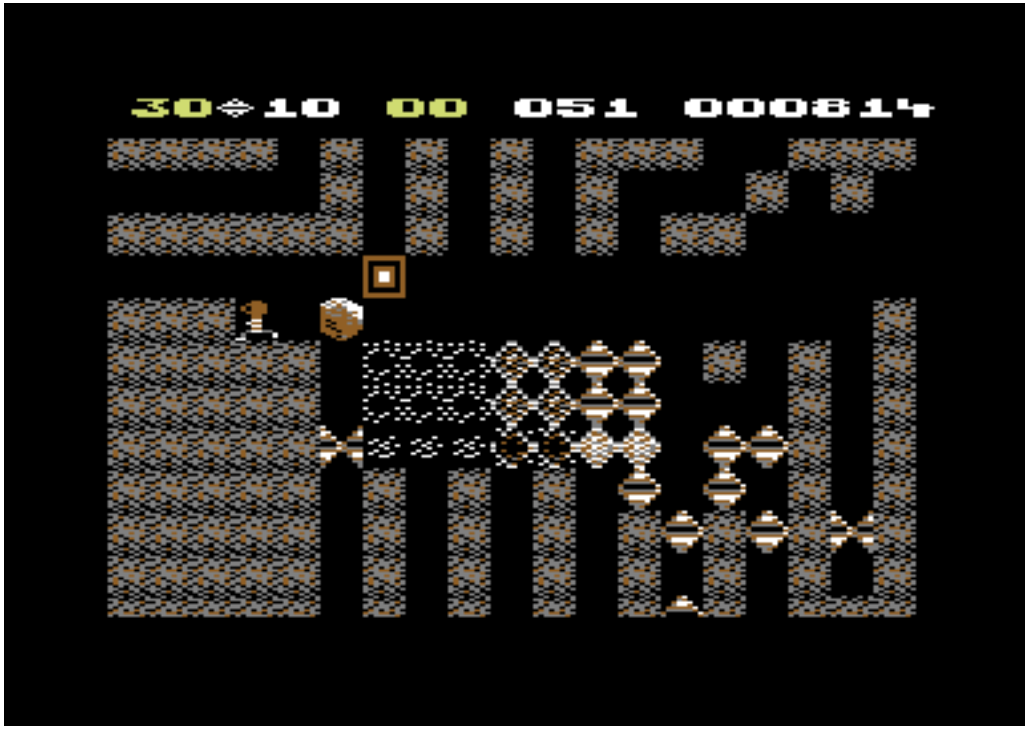
- Identify artefacts explicitly and resolve them.
- Work directly on the input mesh.
- Areas without artefacts are not changed.
- Low number of additional triangles.
- Easier to understand, smaller memory requirements, fast.

Patching Holes: Liepa

Example algorithm: Filling Holes in Meshes [Liepa 2003]

1. Identify holes.
2. Hole Triangulation.
3. Refinement and Fairing.

Boulder Dash



[image: Wikipedia]

Patching Holes: Liepa

Example algorithm: Filling Holes in Meshes [Liepa 2003]

1. Identify holes.
2. Hole Triangulation.
3. Refinement and Fairing.

Liepa: Identifying Holes

- Holes can be identified automatically by searching for edges with only one incident face.
- Or they can be pointed out by a human by identifying border edges or vertices.
- Once a border edge is found, a series of boundary edges/vertices can be traced until they identify a closed loop of boundary edges.

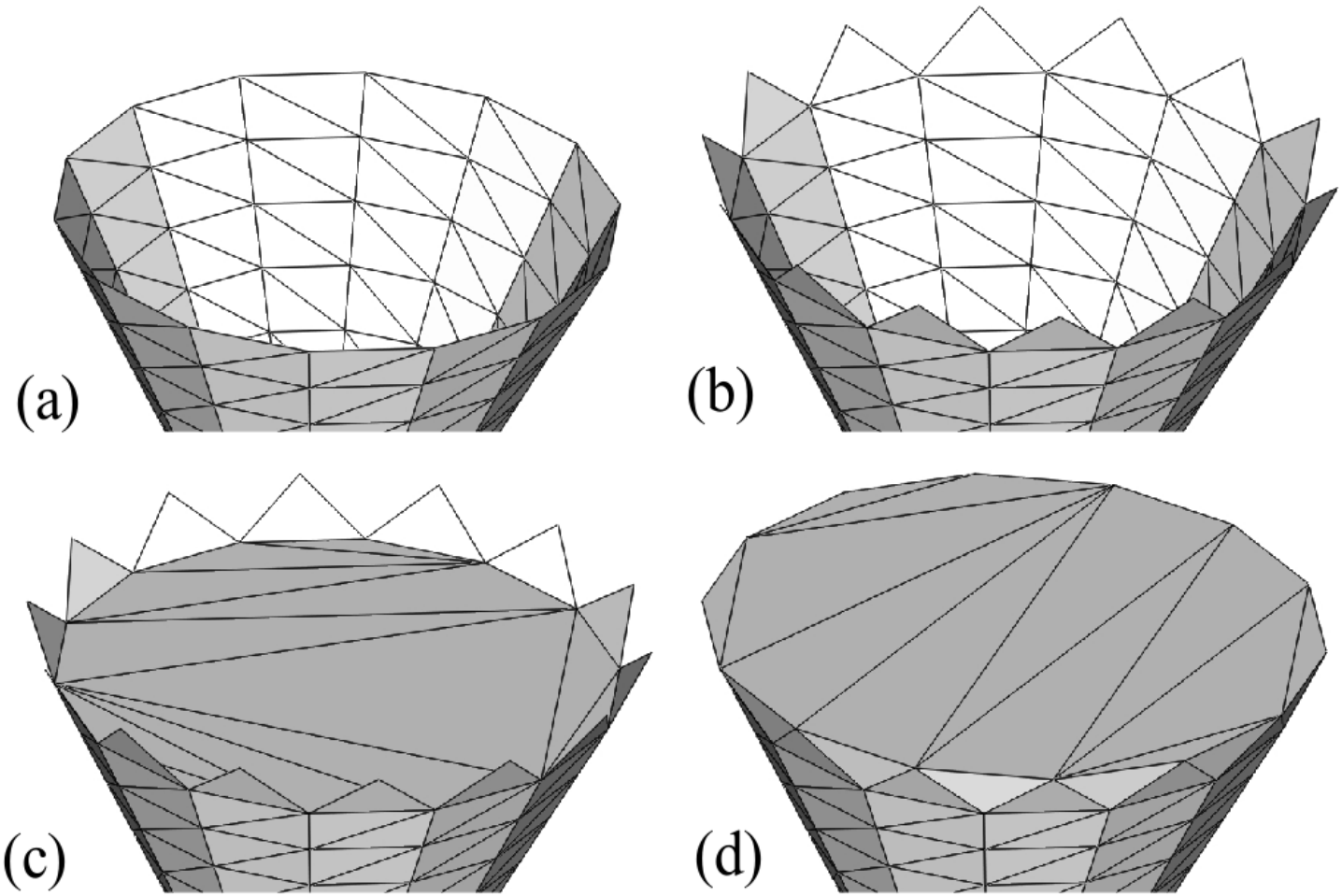
Liepa: Hole Triangulation

- Triangulating three-dimensional polygons is Hard.[Barequet et al.]
- Therefore we will compromise for possibly self-intersecting triangulations.
- Dynamic programming approach to find minimum-weight triangulation:
  - Weight function $\Omega : V^3 \rightarrow L$
  - $W_{i,j}$: weight of the minimum-weight triangulation of the subpolygon from v_i to v_j.
  - Initialization: $W_{i,i+1} = 0$; $W_{i,i+2} = \Omega(v_i, v_{i+1}, v_{i+2})$; $j = 2$.
  - Step: j++; For all i: $k = i + j$;
    $W_{i,k} = \min_{i<m<k}[W_{i,m} + W_{m,k} + \Omega(v_i, v_m, v_k)]$.
    Remember m as $\lambda_{i,k}$.
  - $W_{0,n-1}$ is the minimum-weight triangulation.

Liepa: Hole Triangulation

- Triangulating three-dimensional polygons is Hard.[Barequet et al.]
- Therefore we will compromise for possibly self-intersecting triangulations.
- Dynamic programming approach to find minimum-weight triangulation:
  - Weight function $\Omega : V^3 \rightarrow L$
  - $W_{i,j}$: weight of the minimum-weight triangulation of the subpolygon from v_i to v_j.
  - Initialization: $W_{i,i+1} = 0$; $W_{i,i+2} = \Omega(v_i, v_{i+1}, v_{i+2})$; $j = 2$.
  - Step: $j{+}{+}$; For all i: $k = i + j$;
    $W_{i,k} = \min_{i<m<k}[W_{i,m} + W_{m,k} + \Omega(v_i, v_m, v_k)]$.
    Remember m as $\lambda_{i,k}$.
  - $W_{0,n-1}$ is the minimum-weight triangulation.

Liepa



[image: [Liepa 2003]]

Liepa: Mesh Refinement

- Compute edge length data for the vertices on the hole boundary, then diffuse these values into the interior of the patching mesh, subdividing triangles to reduce edge lengths. Relax interior edges to maintain Delaunay properties.
  1. For every vertex, compute $\sigma(v_i)$.
  2. For each triangle, compute centroid v_c, and its $\sigma(v_c)$. For every corner m: if $\alpha||v_c - v_m|| > max(\sigma(v_c), \sigma(v_m))$, subdivide the triangle at v_c.
  3. If no triangles were created, refinement ends.
  4. Relax patching mesh edges.
  5. Goto 2.

Liepa: Mesh Refinement

- Compute edge length data for the vertices on the hole boundary, then diffuse these values into the interior of the patching mesh, subdividing triangles to reduce edge lengths. Relax interior edges to maintain Delaunay properties.
  1. For every vertex, compute $\sigma(v_i)$.
  2. For each triangle, compute centroid v_c, and its $\sigma(v_c)$. For every corner m: if $\alpha||v_c - v_m|| > max(\sigma(v_c), \sigma(v_m))$, subdivide the triangle at v_c.
  3. If no triangles were created, refinement ends.
  4. Relax patching mesh edges.
  5. Goto 2.

Problems

- result might be self-intersecting
- islands are ignored

Volumetric Approach

- Converts input to some intermediary representation that represents volume.
- e.g.: voxel grid, adaptive octree, BSP tree
- drawbacks: loss of information
- advantage: robust and reliable

Volumetric Approach: Nooruddin, Turk, I

Example algorithm: Nooruddin, Turk: *Simplification and Repair of Polygonal Models Using Volumetric Techniques* [Nooruddin, Turk 2003]

- Mesh to Voxels:
  - Parity Count
  - Ray Stabbing

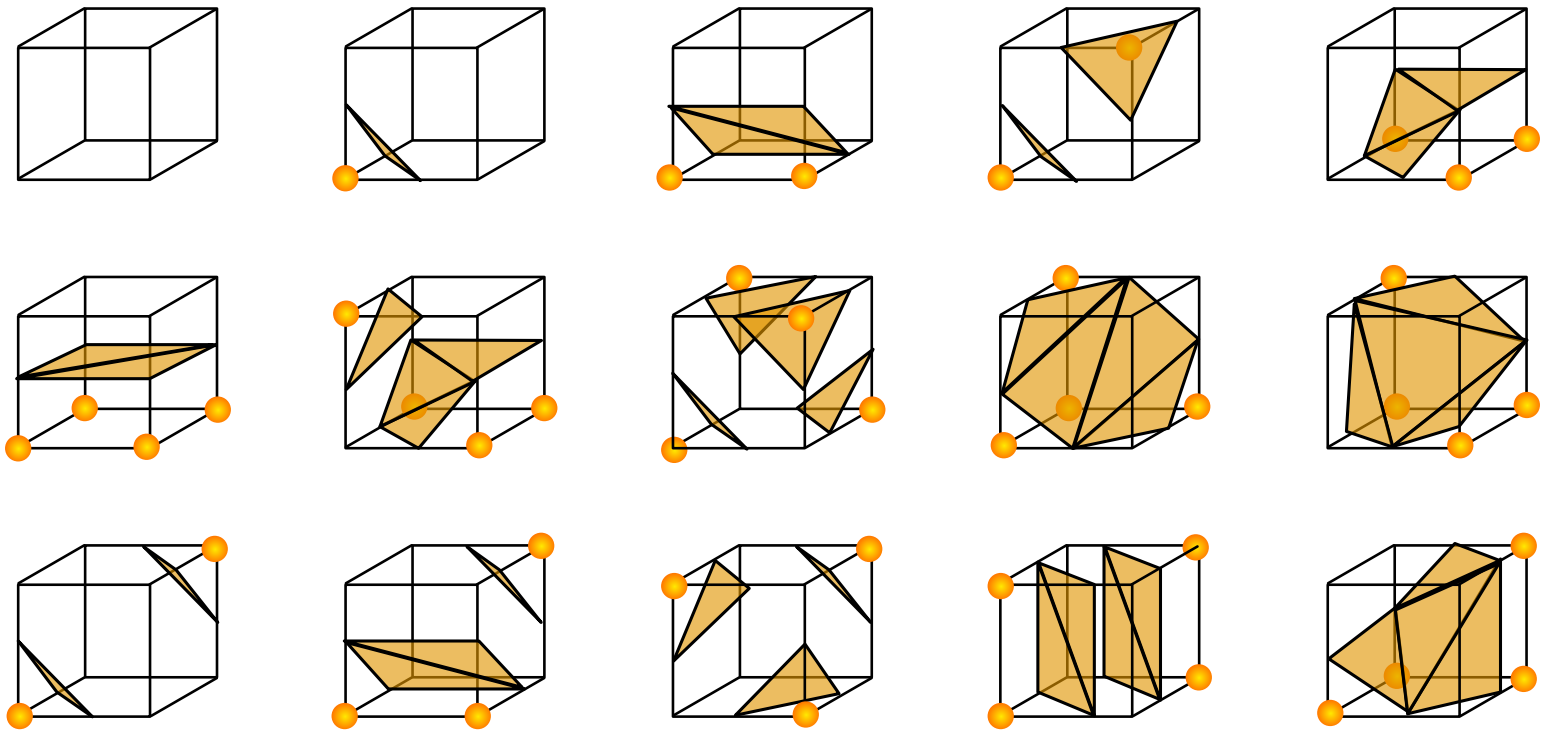
Volumetric Approach: Nooruddin, Turk, II

- Morphological Operations:
  - Classification: *feature*, *non-feature* voxels.
  - Compute distance map.
  - Primitives: *erosion*, *dilution*.
  - Compound Operations: *opening*, *closing*.

Volumetric Approach: Nooruddin, Turk, III

- Voxels to Mesh:
  - Surface extraction: Marching Cubes.
  - Smoothing and triangle count reduction.

Marching Cubes



[image: wikipedia/Jmtrivial; GPL]

Outline

1 Introduction

2 Data Structures

3 Model Repair

4 Mesh quality and smoothness

5 Subdivision Surfaces

6 Further Reading

Mesh quality

- Mesh quality refers to non-topological properties, such as sampling density, regularity, size, orientation, alignment, and shape of mesh elements.
- Local properties:
  - Element type
  - Element shape
  - Element density
- Global properties: degree of vertices.
  Irregular/*Semiregular*/*Highly Regular*/*Regular* meshes.

[PMP]

Smoothing

- Remove undesirable noise and uneven edges while retaining desirable geometric features. [Desbrun et al.]
- Compute shapes that are as smooth as possible.

Fairing

- Idea: Define an energy function that penalizes “ugly” behavior and try to minimize that function.
- Fair surfaces should follow the principle of simplest shape: the surface should be free of any unnecessary details or oscillations [PMP].

Fairing

- $\omega : V^2 \to \mathbb{R}$ weight function for edges.
- Weighted Umbrella Operator:

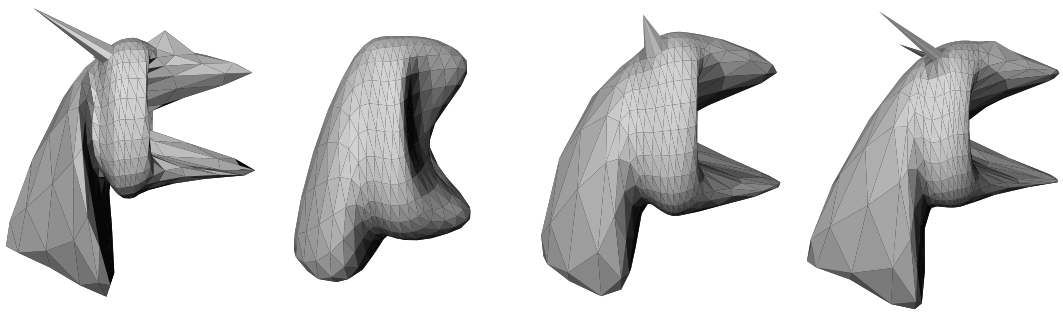
$$U_\omega(v) = -v + \frac{1}{\sum_{v_i \in N(v)} \omega(v, v_i)} \cdot \sum_{v_i \in N(v)} \omega(v, v_i) v_i$$

- To smooth a mesh, replace each vertex v with $v + U_\omega(v)$.

Fairing

- Different weights produce different results:
  - *uniform umbrella operator* with $\omega(v_i, v_j) = 1$.
  - *scale-dependent umbrella operator* with $\omega(v_i, v_j) = ||v_i - v_j||^{-1}$.
  - *harmonic umbrella operator* with $\omega(v_i, v_j) = \cot(\angle(v_i, v_{k_1}, v_j)) + \cot(\angle(v_i, v_{k_2}, v_j))$

    where $\triangle(v_i, v_{k_1}, v_j))$ and $\triangle(v_i, v_{k_2}, v_j)$ are the two faces incident to (v_i, v_j).



[image: [Desbrun et al.]]

Fairing

- Second Order Umbrella Operator:

$$U_\omega^2(v) = -U_\omega(v) + \frac{1}{\sum_{v_i \in N(v)} \omega(v, v_i)} \cdot \sum_{v_i \in N(v)} \omega(v, v_i) U_\omega(v_i)$$

- If $U_\omega(v)$ measures the deviation of a vertex v from a taunt surface bounded by its neighbors, then $U_\omega^2(v) = 0$ implies that the deviation from tautness at a vertex v is equal to the average deviation from tautness of its neighbors.[Liepa 2003]
- Relocating the inner vertices of a patching mesh in Liepa to establish this property amounts to solving a linear system.

Spectral Analysis

- Signal processing approach: transform the “signal” from the mesh domain to a frequency domain.
- i.e. something like Fourier/Laplace Transform.
- There the signal can be enhanced.
- Then transform back.

Outline

1 Introduction

2 Data Structures

3 Model Repair

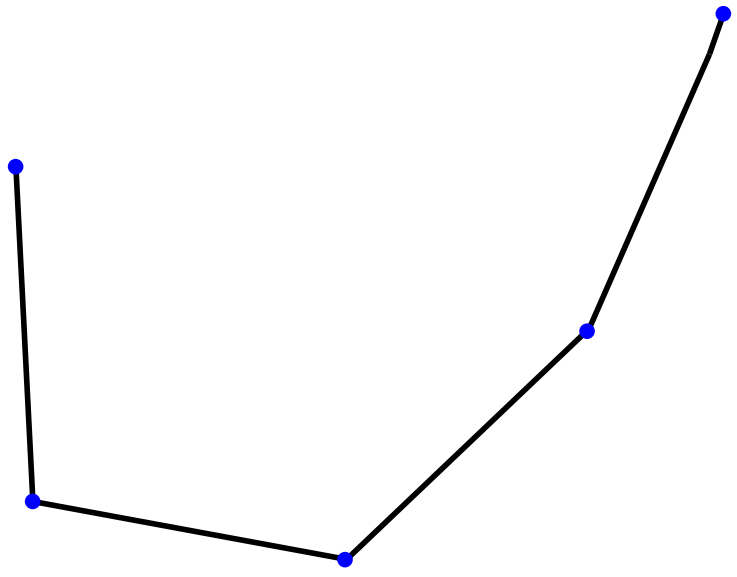
4 Mesh quality and smoothness

5 Subdivision Surfaces

6 Further Reading

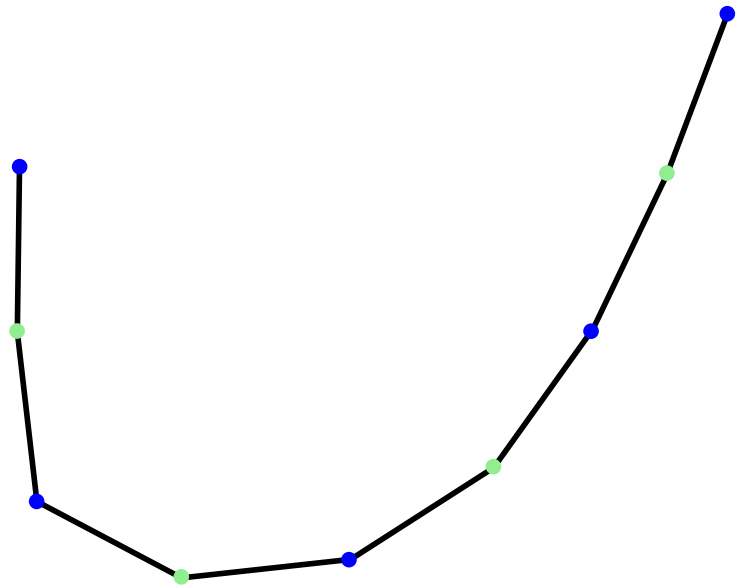
Subdivision Surfaces

- *Subdivision defines a smooth curve or surface as the limit of a sequence of successive refinements.* [Zorin et al. 2000]



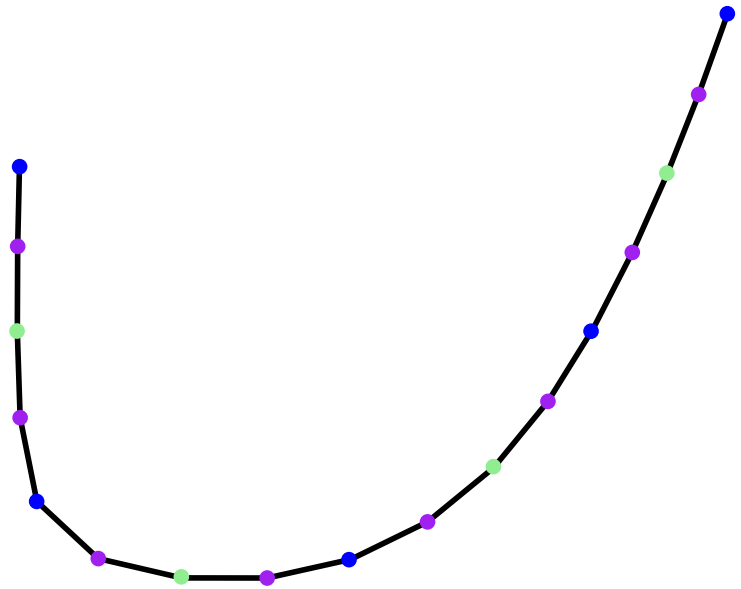
Subdivision Surfaces

- *Subdivision defines a smooth curve or surface as the limit of a sequence of successive refinements.* [Zorin et al. 2000]



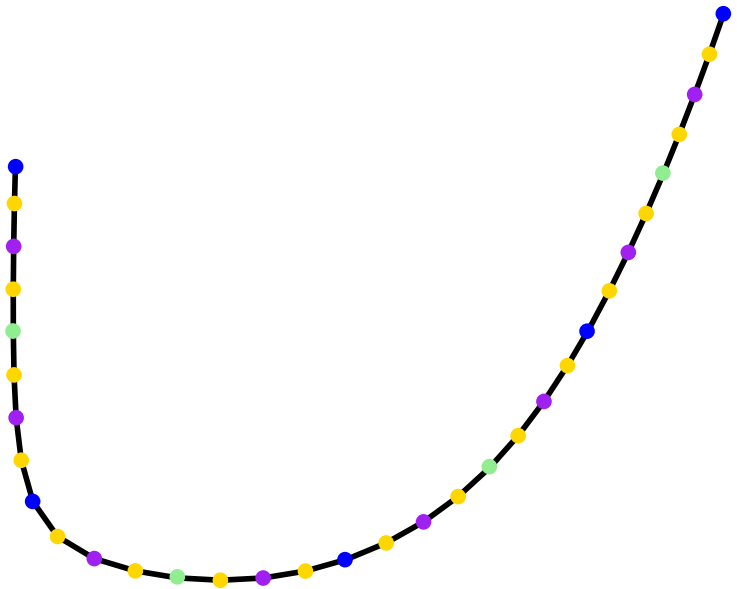
Subdivision Surfaces

- *Subdivision defines a smooth curve or surface as the limit of a sequence of successive refinements.* [Zorin et al. 2000]

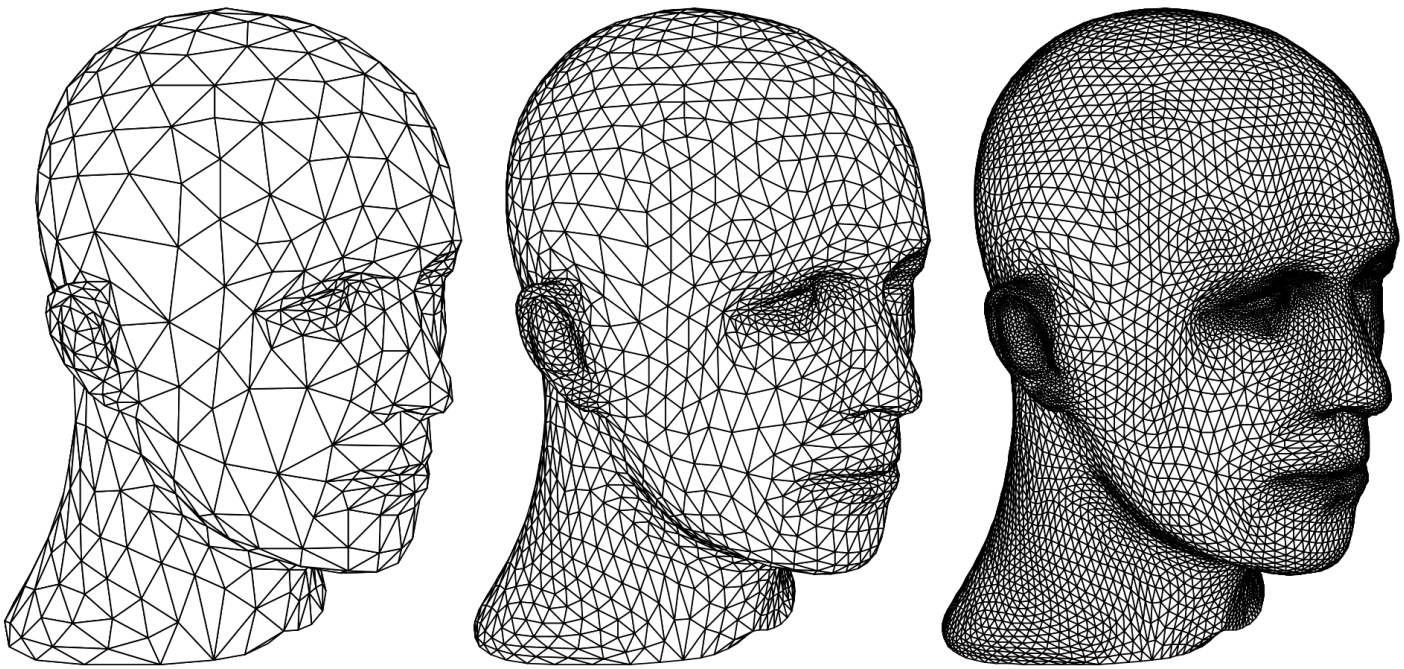


Subdivision Surfaces

- *Subdivision defines a smooth curve or surface as the limit of a sequence of successive refinements.* [Zorin et al. 2000]



Also in 3D



[image from [Zorin et al. 2000]]

Classification

Classification criteria of refinement schemes:

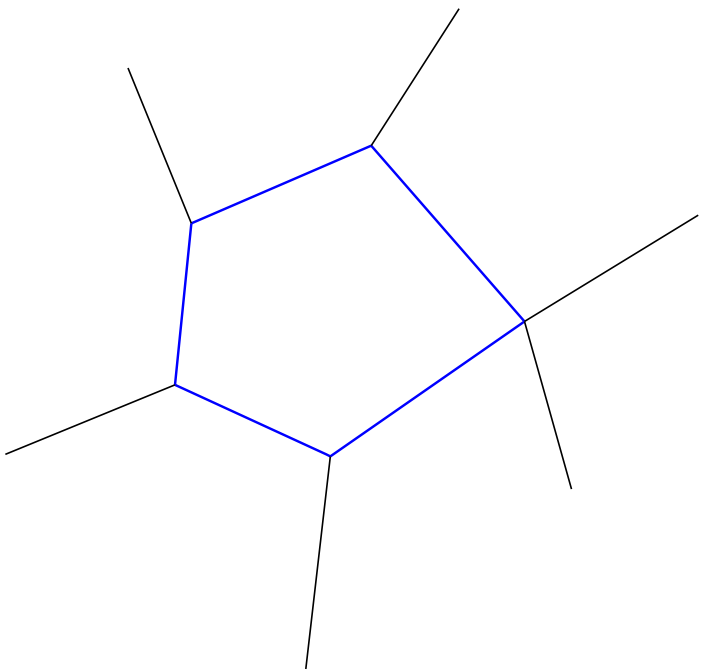
- type of refinement rule,
- type of mesh that is generated,
- and more ...

Catmull Clark

Edwin Catmull and Jim Clark in 1978 [Catmull-Clark]

Face-split refinement:

1. For each face, add a *facepoint*.
2. For each edge, add an *edgepoint*.
3. Add edges from each new facepoint to its new edgepoints.
4. Move original points around.

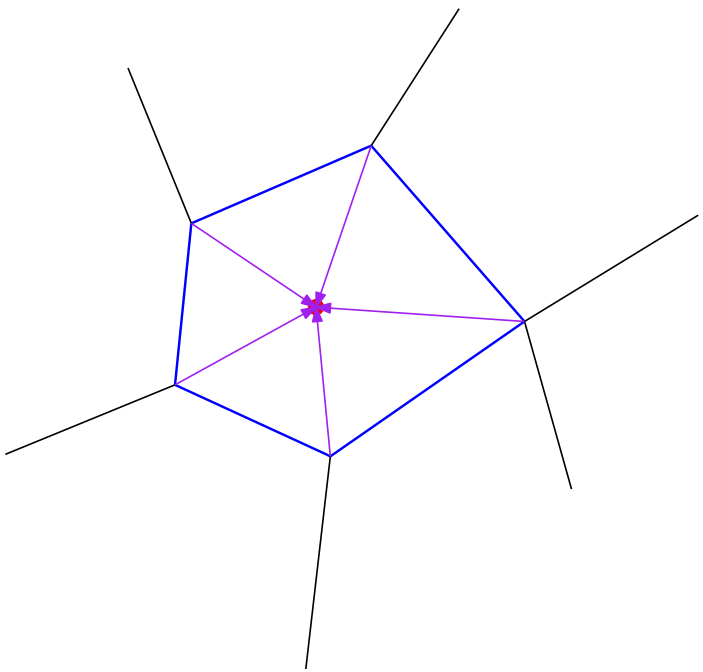


Catmull Clark

Edwin Catmull and Jim Clark in 1978 [Catmull-Clark]

Face-split refinement:

1. For each face, add a *facepoint*.
2. For each edge, add an *edgepoint*.
3. Add edges from each new facepoint to its new edgepoints.
4. Move original points around.

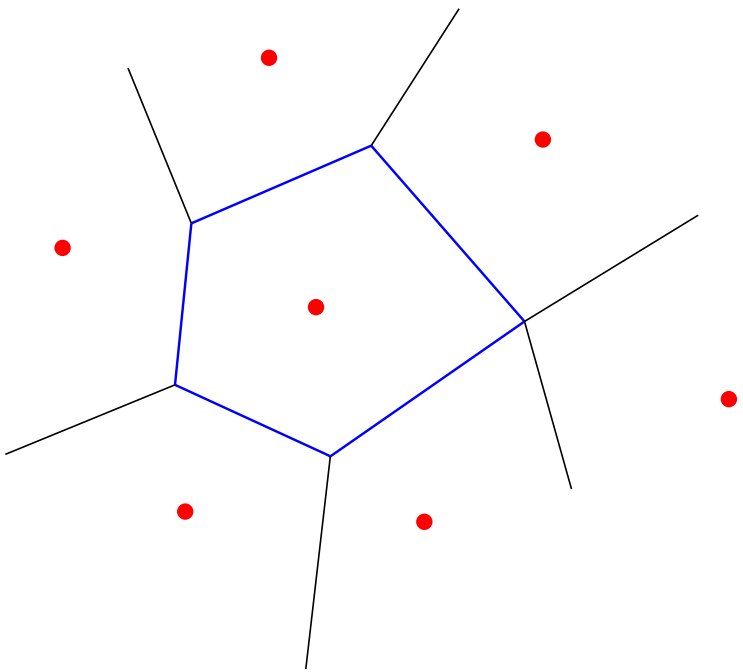


Catmull Clark

Edwin Catmull and Jim Clark in 1978 [Catmull-Clark]

Face-split refinement:

1. For each face, add a *facepoint*.
2. For each edge, add an *edgepoint*.
3. Add edges from each new facepoint to its new edgepoints.
4. Move original points around.

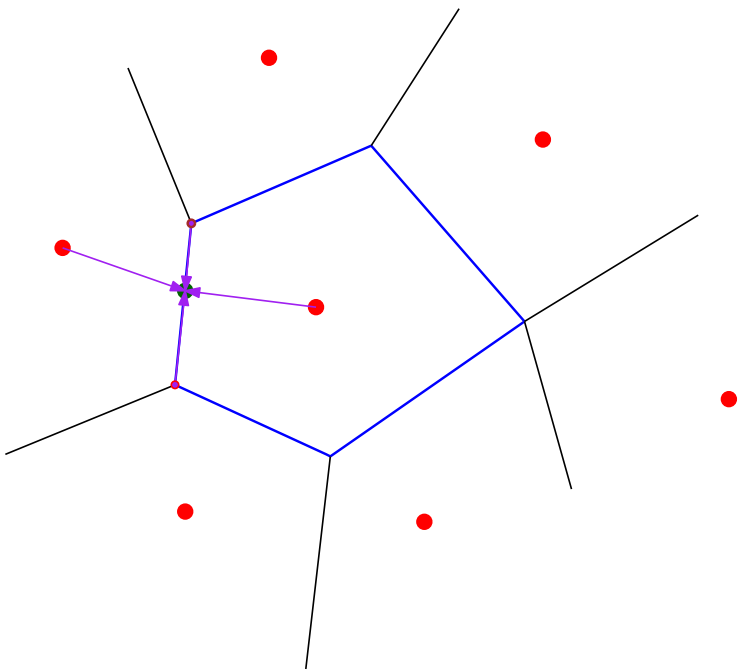


Catmull Clark

Edwin Catmull and Jim Clark in 1978 [Catmull-Clark]

Face-split refinement:

1. For each face, add a *facepoint*.
2. For each edge, add an *edgepoint*.
3. Add edges from each new facepoint to its new edgepoints.
4. Move original points around.

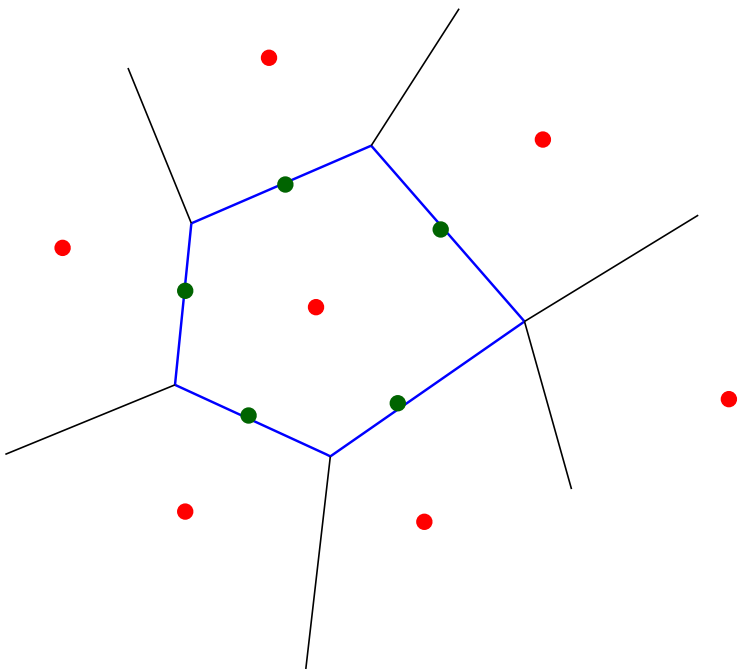


Catmull Clark

Edwin Catmull and Jim Clark in 1978 [Catmull-Clark]

Face-split refinement:

1. For each face, add a *facepoint*.
2. For each edge, add an *edgepoint*.
3. Add edges from each new facepoint to its new edgepoints.
4. Move original points around.

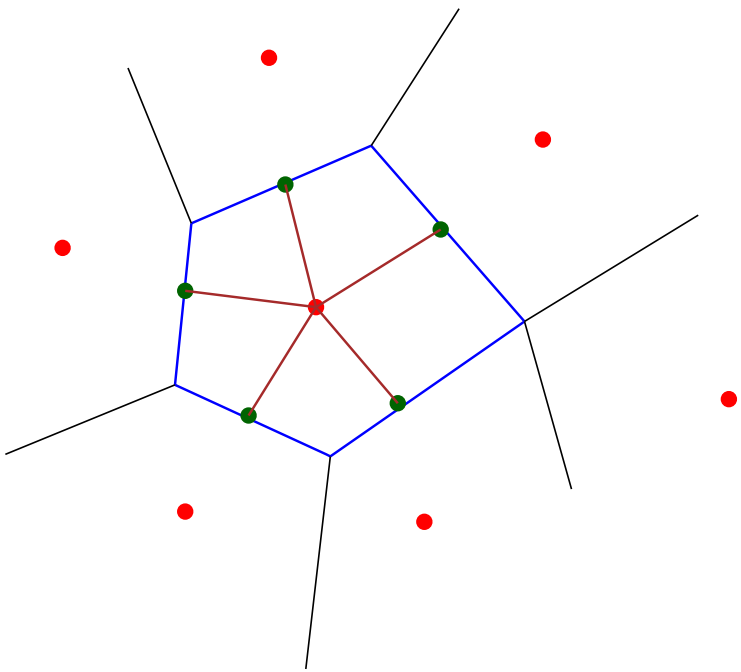


Catmull Clark

Edwin Catmull and Jim Clark in 1978 [Catmull-Clark]

Face-split refinement:

1. For each face, add a *facepoint*.
2. For each edge, add an *edgepoint*.
3. Add edges from each new facepoint to its new edgepoints.
4. Move original points around.

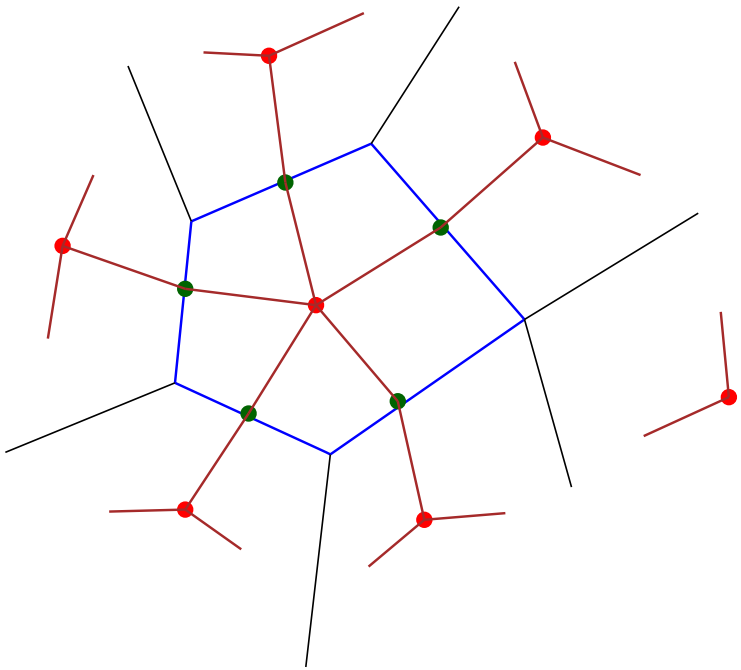


Catmull Clark

Edwin Catmull and Jim Clark in 1978 [Catmull-Clark]

Face-split refinement:

1. For each face, add a *facepoint*.
2. For each edge, add an *edgepoint*.
3. Add edges from each new facepoint to its new edgepoints.
4. Move original points around.

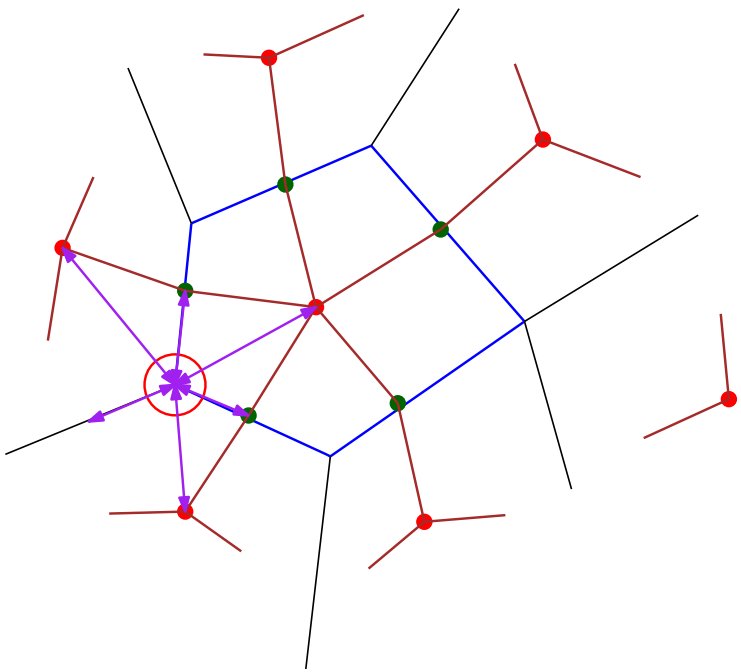


Catmull Clark

Edwin Catmull and Jim Clark in 1978 [Catmull-Clark]

Face-split refinement:

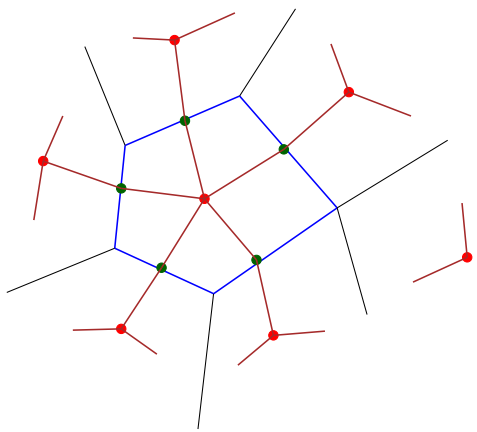
1. For each face, add a *facepoint*.
2. For each edge, add an *edgepoint*.
3. Add edges from each new facepoint to its new edgepoints.
4. Move original points around.



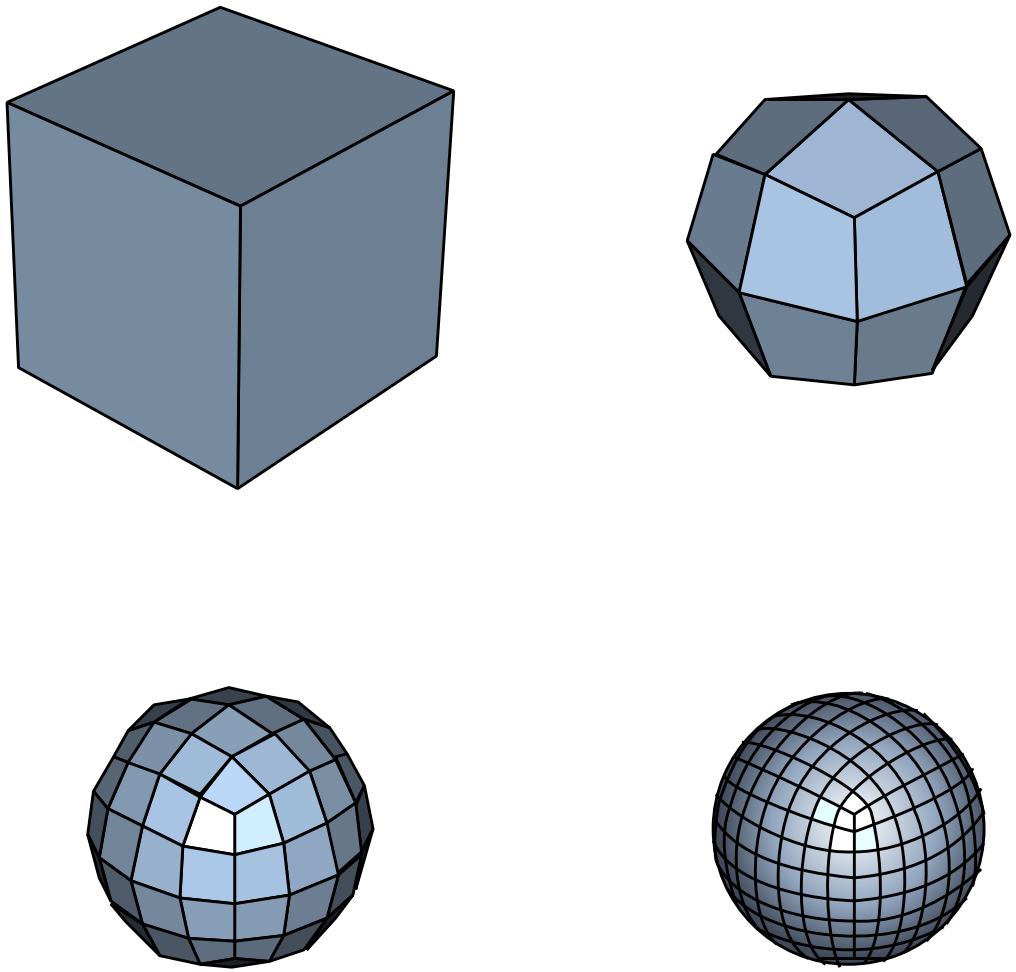
Catmull Clark

Original points get moved to $\frac{Q+2R+(n-3)S}{n}$, where n is the number of incident faces, Q is the barycenter of these faces' facepoints, R is the average over all incident edges' edgepoints, and S is the vertex's old coordinates.

- All new faces are quadrilaterals.
- New faces are not necessarily planar.



Catmull Clark

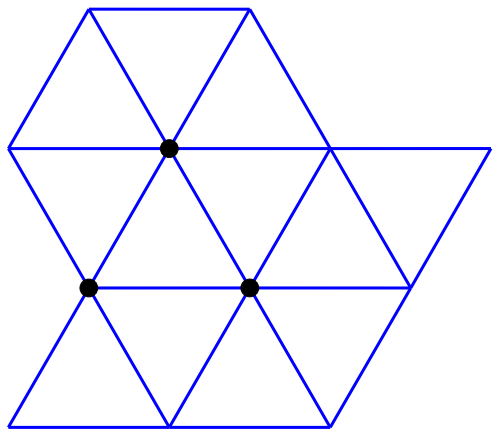


[image: wikipedia/Romainbehar; CC BY-SA 3.0]

Loop

Charles Loop [Loop 1987]

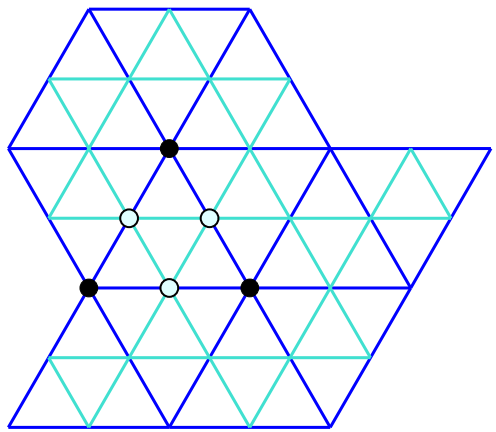
- Also a face-split scheme.
- Only works on triangular meshes.
- Refine each triangle into four new triangles.



Loop

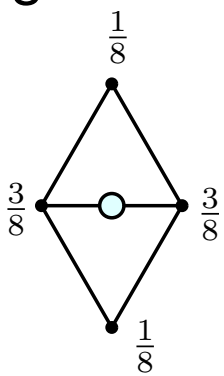
Charles Loop [Loop 1987]

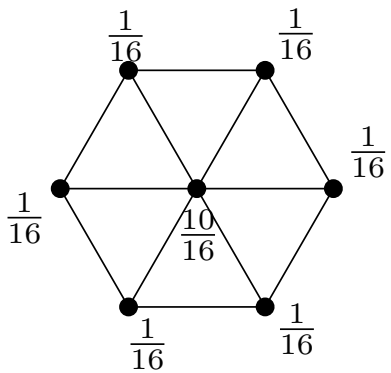
- Also a face-split scheme.
- Only works on triangular meshes.
- Refine each triangle into four new triangles.



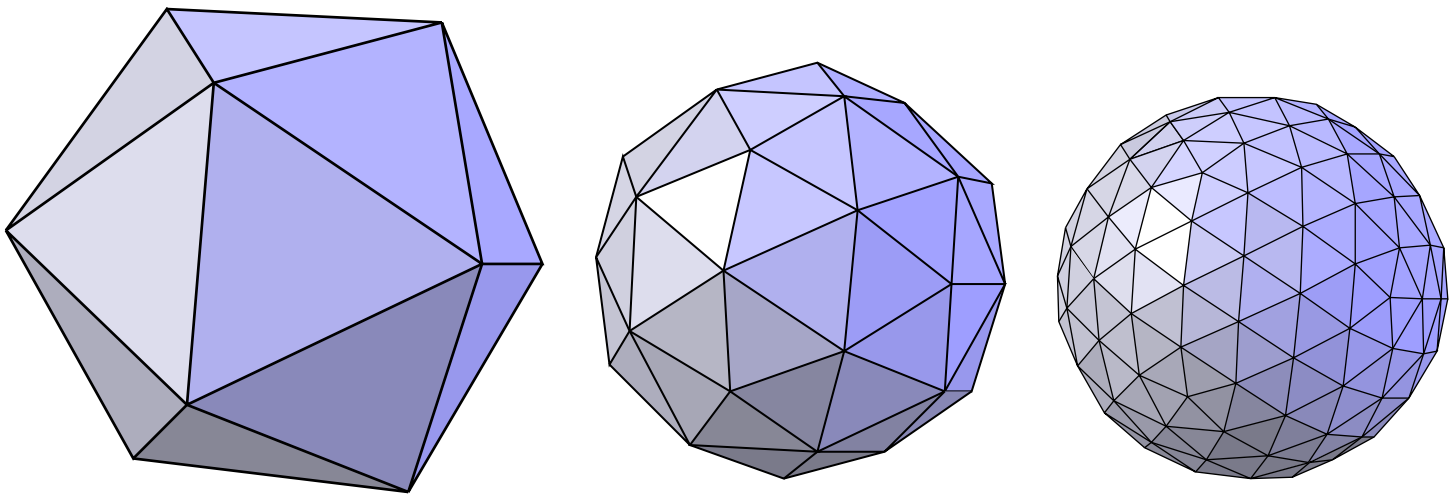
Loop

- New vertices are chosen as (weighted) average of vertices in the neighborhood.





Loop



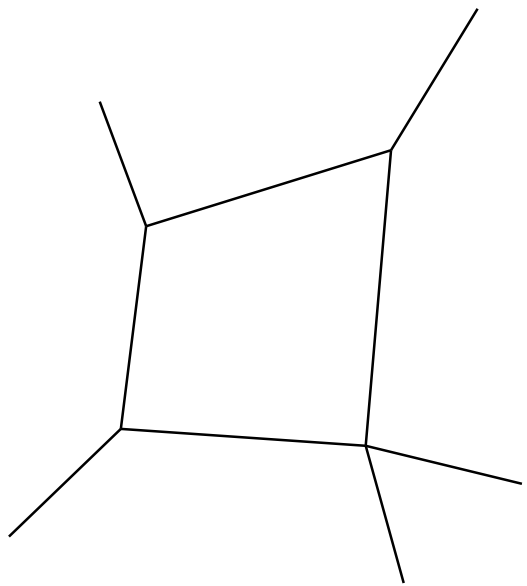
[image: wikipedia/Simon Fuhrmann; CC BY-SA 3.0]

Doo Sabin

Daniel Doo, Malcolm Sabin [Doo 1978], [Doo-Sabin 1978]

Vertex-split refinement:

1. Generate facepoints and edgepoints.
2. Generate one point for each corner of all faces.
3. Create new faces:
   1. in every existing face,
   2. in every existing vertex,
   3. for every existing edge.

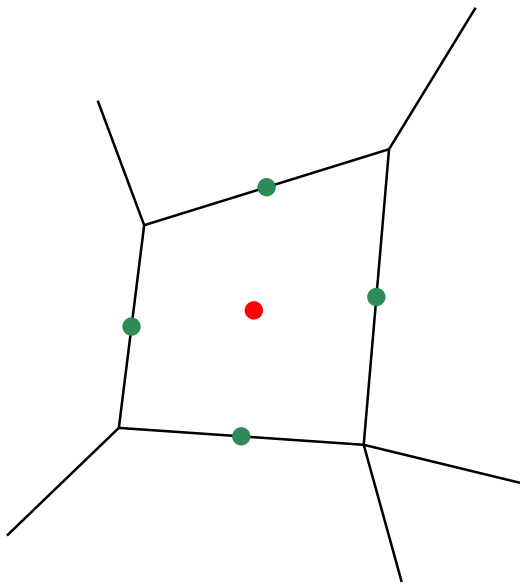


Doo Sabin

Daniel Doo, Malcolm Sabin [Doo 1978], [Doo-Sabin 1978]

Vertex-split refinement:

1. Generate facepoints and edgepoints.
2. Generate one point for each corner of all faces.
3. Create new faces:
   1. in every existing face,
   2. in every existing vertex,
   3. for every existing edge.

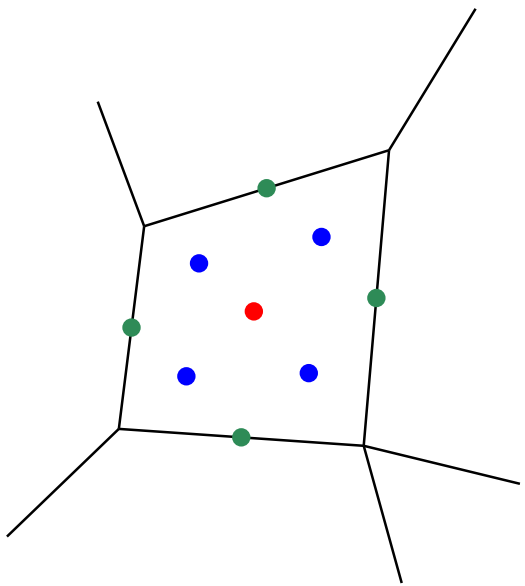


Doo Sabin

Daniel Doo, Malcolm Sabin [Doo 1978], [Doo-Sabin 1978]

Vertex-split refinement:

1. Generate facepoints and edgepoints.
2. Generate one point for each corner of all faces.
3. Create new faces:
   1. in every existing face,
   2. in every existing vertex,
   3. for every existing edge.

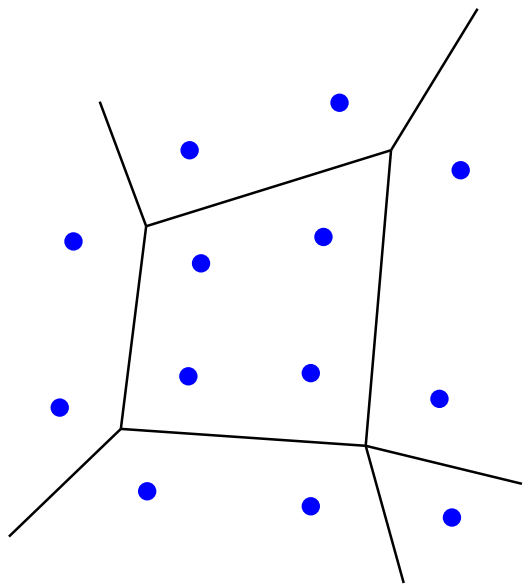


Doo Sabin

Daniel Doo, Malcolm Sabin [Doo 1978], [Doo-Sabin 1978]

Vertex-split refinement:

1. Generate facepoints and edgepoints.
2. Generate one point for each corner of all faces.
3. Create new faces:
   1. in every existing face,
   2. in every existing vertex,
   3. for every existing edge.

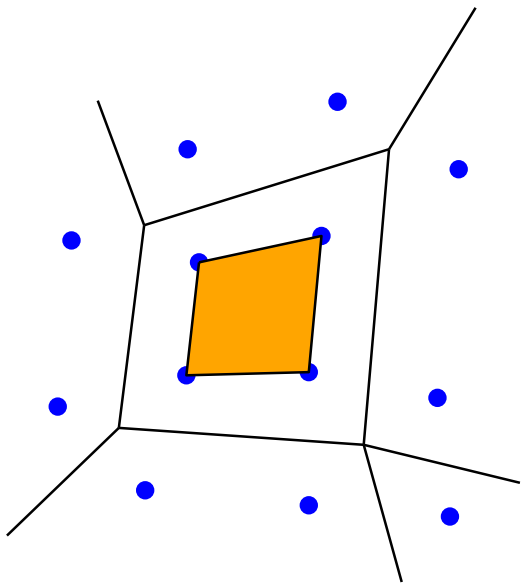


Doo Sabin

Daniel Doo, Malcolm Sabin [Doo 1978], [Doo-Sabin 1978]

Vertex-split refinement:

1. Generate facepoints and edgepoints.
2. Generate one point for each corner of all faces.
3. Create new faces:
   1. in every existing face,
   2. in every existing vertex,
   3. for every existing edge.

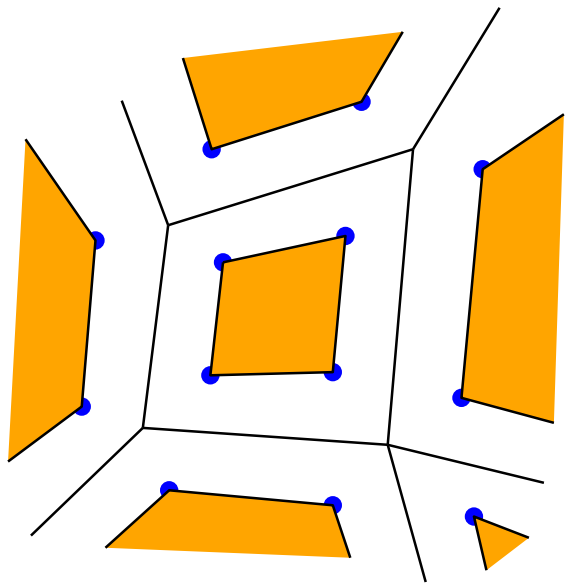


Doo Sabin

Daniel Doo, Malcolm Sabin [Doo 1978], [Doo-Sabin 1978]

Vertex-split refinement:

1. Generate facepoints and edgepoints.
2. Generate one point for each corner of all faces.
3. Create new faces:
   1. in every existing face,
   2. in every existing vertex,
   3. for every existing edge.

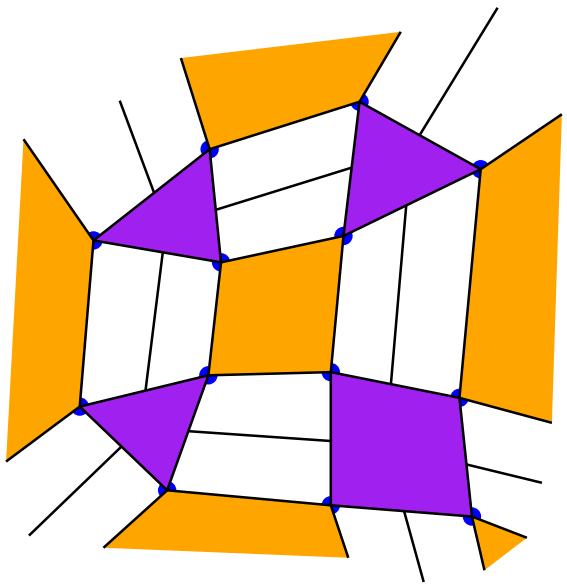


Doo Sabin

Daniel Doo, Malcolm Sabin [Doo 1978], [Doo-Sabin 1978]

Vertex-split refinement:

1. Generate facepoints and edgepoints.
2. Generate one point for each corner of all faces.
3. Create new faces:
   1. in every existing face,
   2. in every existing vertex,
   3. for every existing edge.

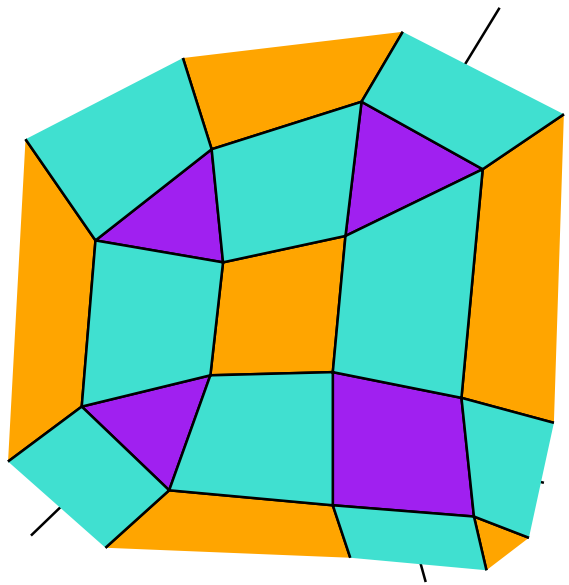


Doo Sabin

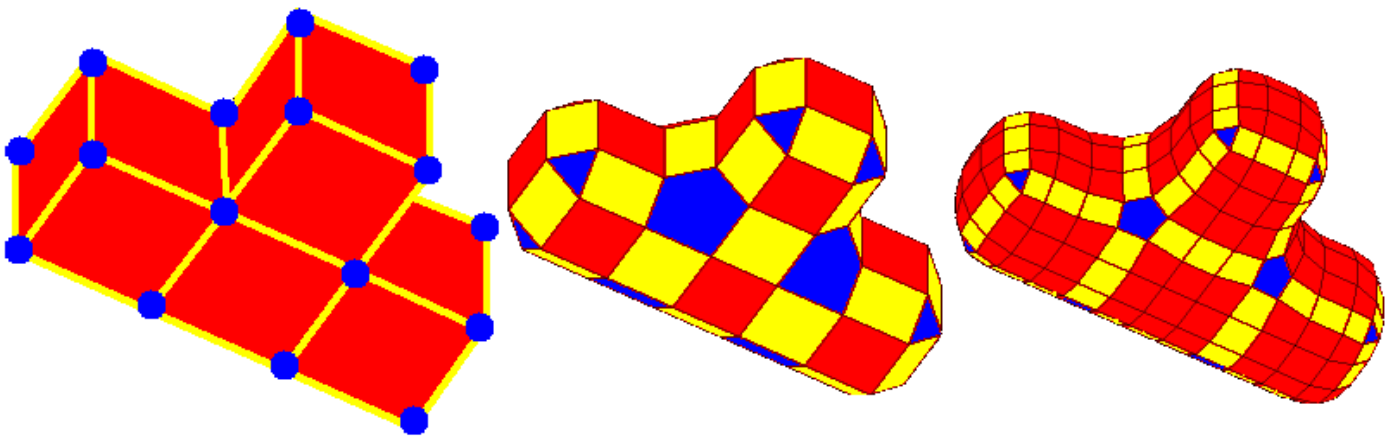
Daniel Doo, Malcolm Sabin [Doo 1978], [Doo-Sabin 1978]

Vertex-split refinement:

1. Generate facepoints and edgepoints.
2. Generate one point for each corner of all faces.
3. Create new faces:
   1. in every existing face,
   2. in every existing vertex,
   3. for every existing edge.

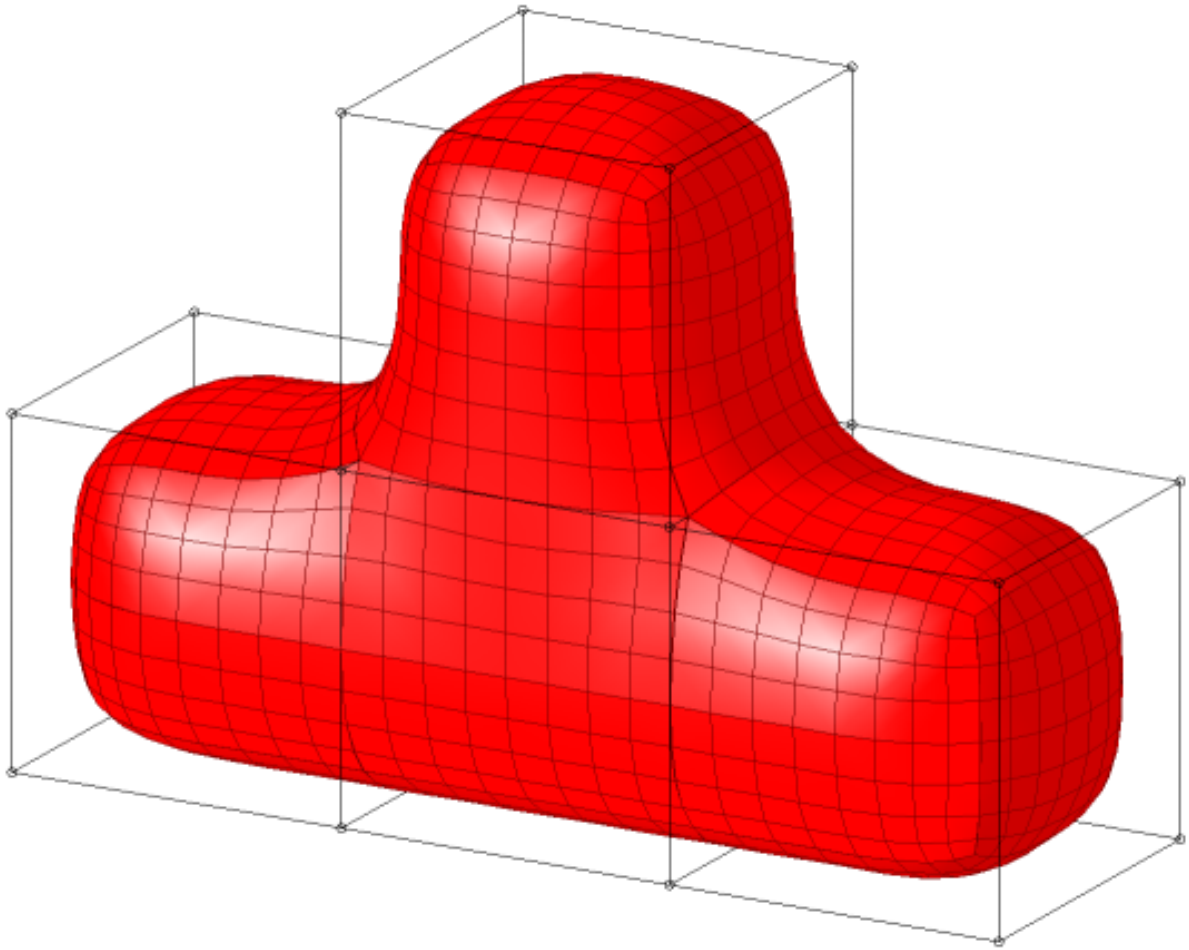


Doo Sabin



[image: wikipedia/Orderud; PD]

Doo Sabin



[image: wikipedia/Orderud; PD]

Summary

Subdvision Surfaces - Properties:

- Scalable – Level of Detail.
- Efficient – only a small number of neighbors influence new points.
- Numerical Stability - many nice properties for e.g. FEM; Also guaranteed continuity, affine invariant.
- Simple.

Outline

1 Introduction

2 Data Structures

3 Model Repair

4 Mesh quality and smoothness

5 Subdivision Surfaces

6 Further Reading

References I


Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, Bruno Levy: ”Polygon Mesh Processing”, A K Peters, 2010.


Wikipedia, ”Polygon mesh — Wikipedia, The Free Encyclopedia”, https://secure.wikimedia.org/wikipedia/en/wiki/Polygon_mesh


Leif Kobbelt, ”Model Repair - Polygon Mesh Processing Book Website”, http://www.pmp-book.org/download/slides/Model_Repair.pdf


Peter Liepa: ”Filling holes in meshes”, SGP ’03 (Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing).


Gill Barequet, Matthew Dickerson, David Eppstein: ”On Triangulating Three-Dimensional Polygons”, SCG ’96 Proceedings of the twelfth annual symposium on Computational geometry.


F.S. Nooruddin and Greg Turk: ”Simplification and Repair of Polygonal Models Using Volumetric Techniques”, IEEE Transactions on Visualization and Computer Graphics, Vol. 9, No. 2, April-June 2003, pp. 191-205.


Mathieu Desbrun, Mark Meyer, Peter Schröder, Peter and Barr, Alan H.: ”Implicit fairing of irregular meshes using diffusion and curvature flow”, SIGGRAPH ’99 Proceedings of the 26th annual conference on Computer graphics and interactive techniques.


D. Zorin, P. Schröder, T. DeRose, L. Kobbelt, A. Levin, W. Sweldens: ”Subdivision for Modeling and Animation.”, ACM SIGGRAPH 2000 Courses. New York: ACM, 2000.

References II


Edwin Catmull and Jim Clark: ”Recursively generated B-spline surfaces on arbitrary topological meshes”, Computer-Aided Design 10(6):350-355 (November 1978).


Wikipedia, ”Catmull-Clark subdivision surface — Wikipedia, The Free Encyclopedia”, https://secure.wikimedia.org/wikipedia/en/wiki/Catmull-Clark_subdivision_surface


Charles Loop: ”Smooth Subdivision Surfaces Based on Triangles”, M.S. Mathematics thesis , University of Utah, 1987.


Daniel Doo: ”A subdivision algorithm for smoothing down irregularly shaped polyhedrons”, Proceedings on Interactive Techniques in Computer Aided Design, pp. 157 - 165, 1978.


Daniel Doo and Malcolm Sabin: ”Behavior of recursive division surfaces near extraordinary points”, Computer-Aided Design, 10 (6) 356?360 (1978).


Ken Joy: ”On-Line Geometric Modeling Notes: Doo-Sabin Surfaces”, http://graphics.cs.ucdavis.edu/education/CAGDNotes/Doo-Sabin/Doo-Sabin.html


Wikipedia, ”Doo-Sabin subdivision surface — Wikipedia, The Free Encyclopedia”, https://secure.wikimedia.org/wikipedia/en/wiki/Doo-Sabin_subdivision_surface

Thank you for your attention.

Questions?