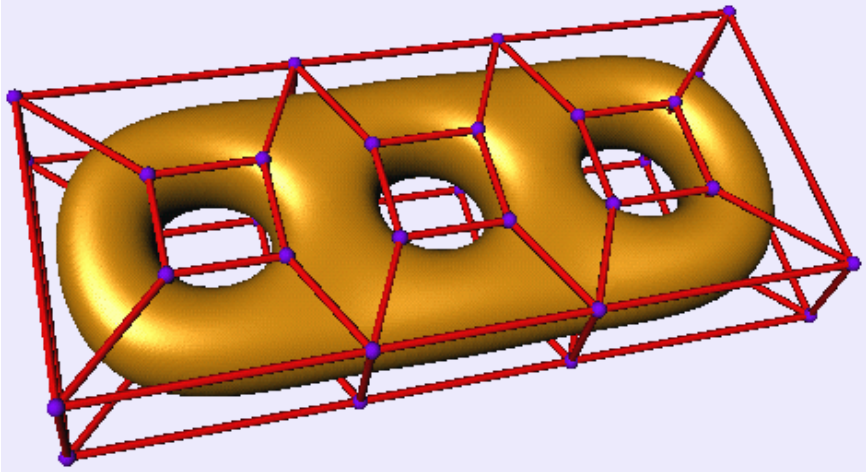
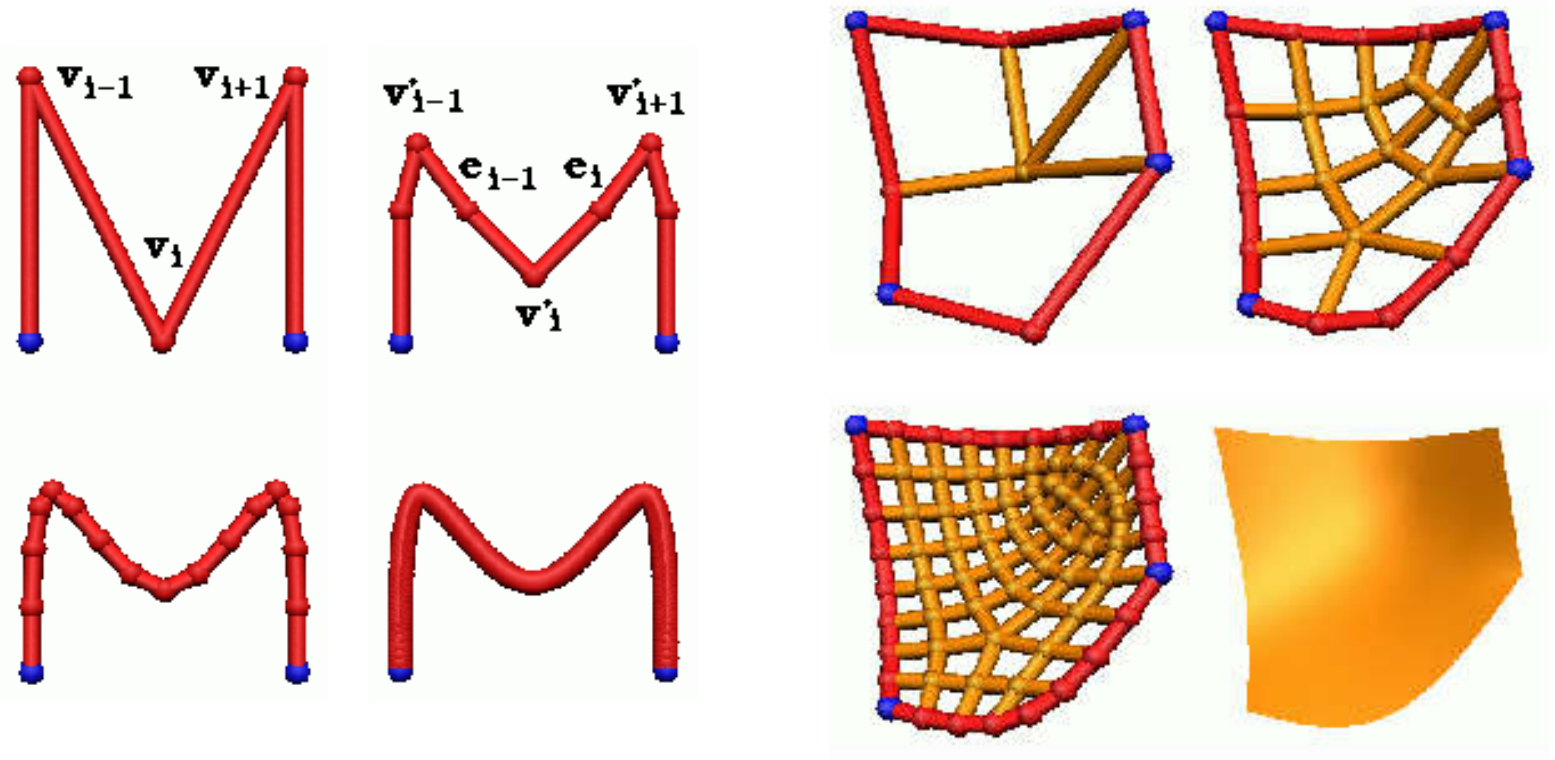
1. Subdivision Curves and Surfaces

Recursively generated curves and surfaces are quite popular in Computer Graphics. They are easy to construct from a polygon / polyhedral mesh, and provide line segments / polygons at multiple levels of resolution. In contrast to splines, Subdivision Surfaces can represent objects of arbitrary topology.



Example: Catmull-Clark Surface representing a genus-3 object.

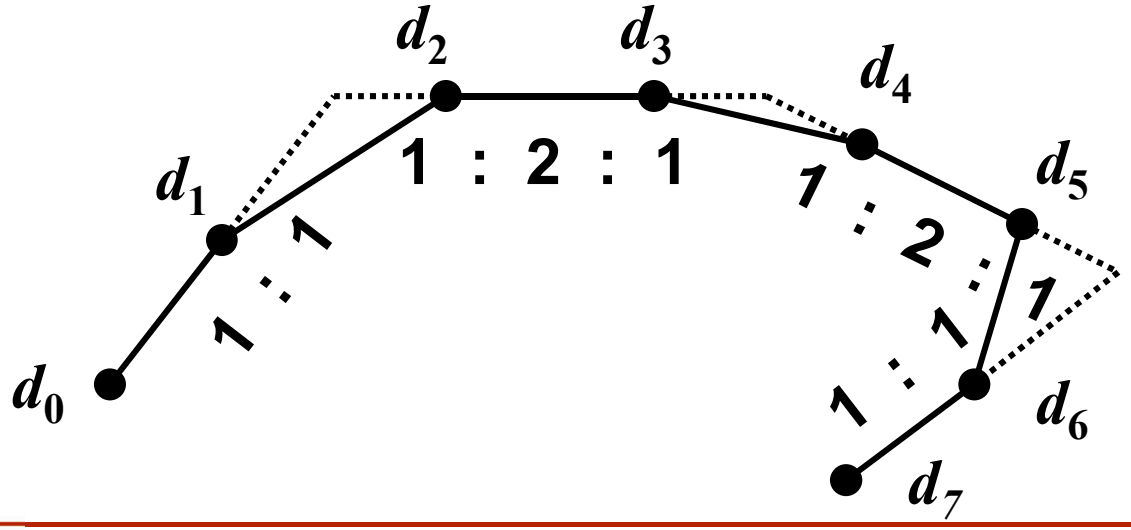
1. Subdivision Curves and Surfaces



Simple subdivision masks are applied multiple times, resulting in a smooth limit shape. The mask can be implemented as a linear map (matrix multiplication), mapping a polygon to its next finer resolution.

1. Subdivision Curves and Surfaces

Example: Chaikin's Algorithm [G. Chaikin, An algorithm for high speed curve generation. CG and Image Processing 3 (1974), pp. 346–349] simply cuts the corners of a polygon, thus replacing every inner vertex by two new ones. It was proven later [R. Riesenfeld, On Chaikin's algorithm. IEEE CG and Applications 4, 3 (1975), pp. 304–310] that this scheme generates piecewise quadratic polynomials.



1. Subdivision Curves and Surfaces

Subdivision mask for Chaikin's Algorithm (quadratic splines)

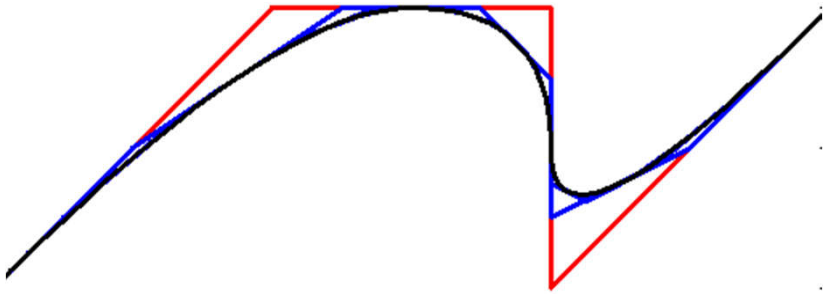
$$\begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ \\ \\ \\ \\ p_{2n-2} \end{bmatrix} := \begin{bmatrix} 1 & & & & & & \\ 1/2 & 1/2 & & & & & \\ & 3/4 & 1/4 & & & & \\ & 1/4 & 3/4 & & & & \\ & & 3/4 & 1/4 & & & \\ & & 1/4 & 3/4 & & & \\ & & & & \ddots & & \\ & & & & & & 1 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ \\ \\ \\ \\ p_n \end{bmatrix}$$

1. Subdivision Curves and Surfaces

```
% Chaikin's Algorithmus in Matlab. Author: MHB 2017
P = [ 0 1 2 2 3; 0 1 1 0 1]'; % control polygon
plot( P(:,1), P(:,2), 'r', 'linewidth', 2);
axis equal
hold on
for k = 1:4 % number of subdivisions
    n = size( P, 1); % number of points
    M = zeros( 2*n-2, n); % subdiv. matrix

    M(1,1) = 1; % upper part
    M(2,1) = .5;
    M(2,2) = .5;

    M(2*n-2, n) = 1; % lower part
    M(2*n-3, n) = .5;
    M(2*n-3, n-1) = .5;
```

1. Subdivision Curves and Surfaces

```
    for i = 2:n-2 % inner part of M
        M( 2*i-1, i) = .75;
        M( 2*i-1, i+1) = .25;
        M( 2*i, i) = .25;
        M( 2*i, i+1) = .75;
    end % i-loop for inner part

    P = M*P; % apply matrix and draw polygon
    plot( P(:,1), P(:,2), 'linewidth', 2);
end % k-loop for subdivision levels

plot( P(:,1), P(:,2), 'k', 'linewidth', 2);
hold off
```



1. Subdivision Curves and Surfaces

Cubic Spline Mask (simplified at boundaries)

$$\begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ \\ \\ \\ \\ p_{2n-1} \end{bmatrix} := \begin{bmatrix} 1 & & & & & & \\ 1/2 & 1/2 & & & & & \\ 1/8 & 3/4 & 1/8 & & & & \\ & 1/2 & 1/2 & & & & \\ & 1/8 & 3/4 & 1/8 & & & \\ & & 1/2 & 1/2 & & & \\ & & & & \ddots & & \\ & & & & & 1 & \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ \\ \\ \\ \\ p_n \end{bmatrix}$$

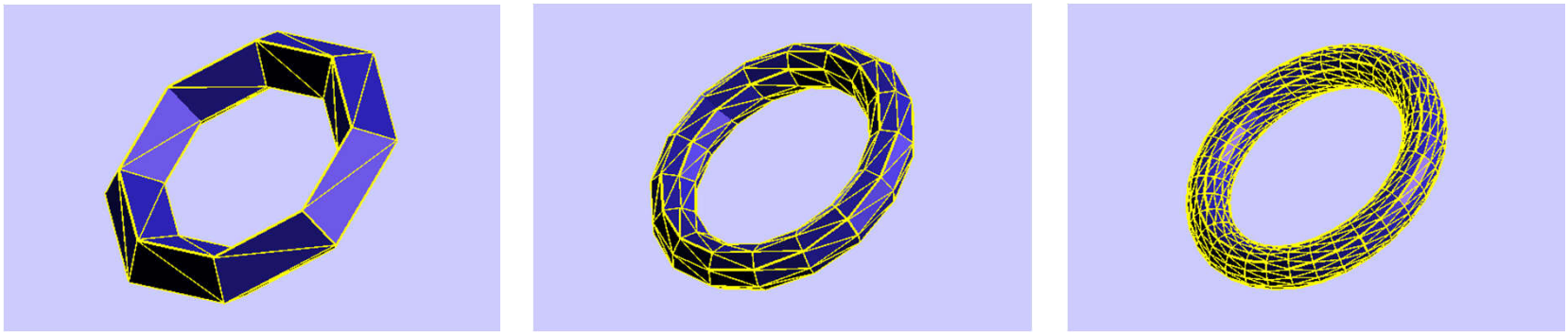
1. Subdivision Curves and Surfaces

Subdivision mask for piecewise cubic interpolating curve

$$\begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ \\ \\ \\ \\ p_{2n-1} \end{bmatrix} := \begin{bmatrix} 1 & & & & & & \\ 3/8 & 3/4 & -1/8 & & & & \\ & 1 & & & & & \\ -1/16 & 9/16 & 9/16 & -1/16 & & & \\ & & 1 & & & & \\ & -1/16 & 9/16 & 9/16 & -1/16 & & \\ & & & & \ddots & & \\ & & & & & & 1 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ \\ \\ \\ \\ p_n \end{bmatrix}$$

1.1 The Loop Subdivision Scheme

Recursively Generated Surfaces

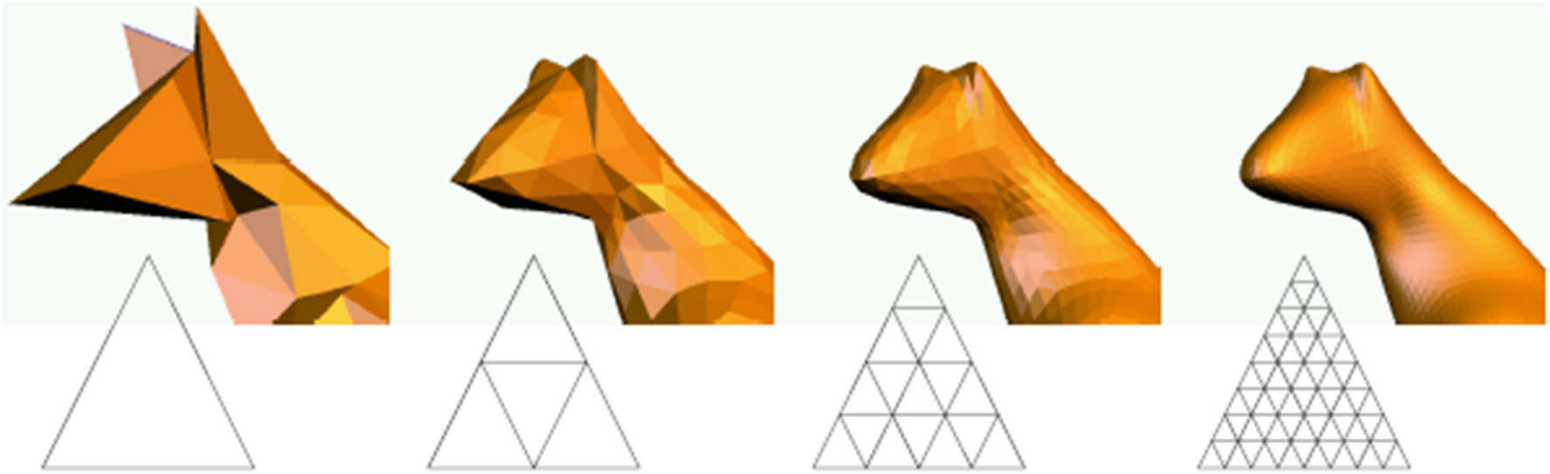


Loop subdivision for triangle meshes: each triangle is split into four

Subdivision surfaces are smooth surfaces generated from polygon meshes. The polygons are repeatedly split into smaller ones based on certain subdivision masks, converging to a smooth limit surface. Thus, complicated objects with arbitrary topology can be modeled in one piece.

1.1 The Loop Subdivision Scheme

Recursively Generated Surfaces

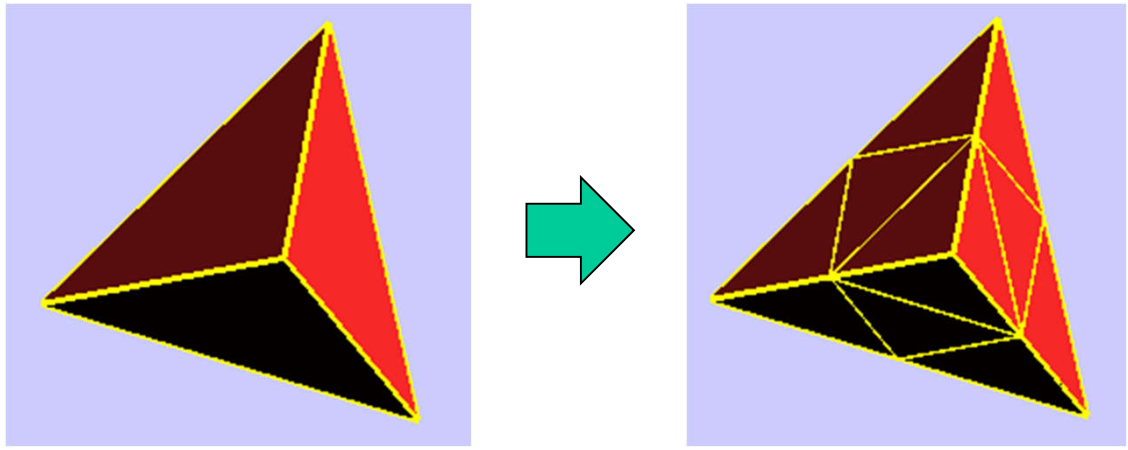


Loop Subdivision Surfaces are used to generate smooth surfaces from triangle meshes. Therefore, every triangle is split into four by applying the same rules over and over again.

[C.T. Loop, Smooth subdivision surfaces based on triangles, M.S. Thesis, Department of Mathematics, University of Utah, 1987]

1.1 The Loop Subdivision Scheme

Triangle Splitting



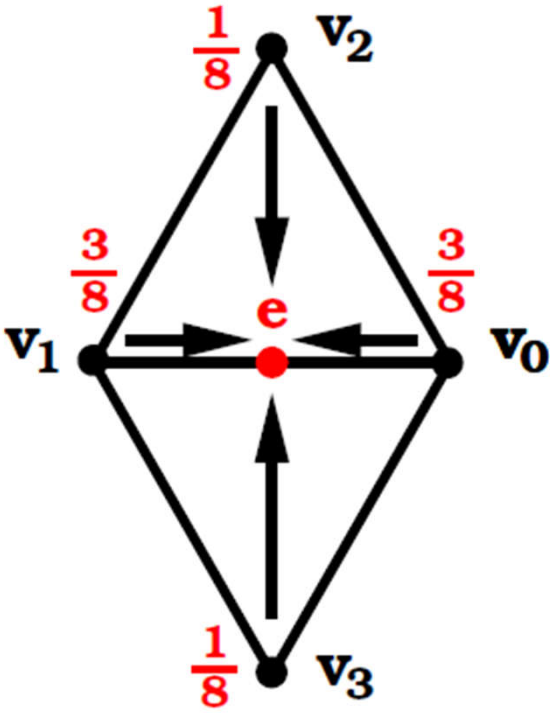
Every triangle is split into four by inserting a new point for every edge. Here, we simply choose the midpoint to illustrate the mesh structure. In contrast, the Loop scheme is a bit more sophisticated:

1.1 The Loop Subdivision Scheme

Edge-Mask

Rather than taking the edge midpoint, Loop suggests a weighted average of the four points making up the two adjacent triangles:

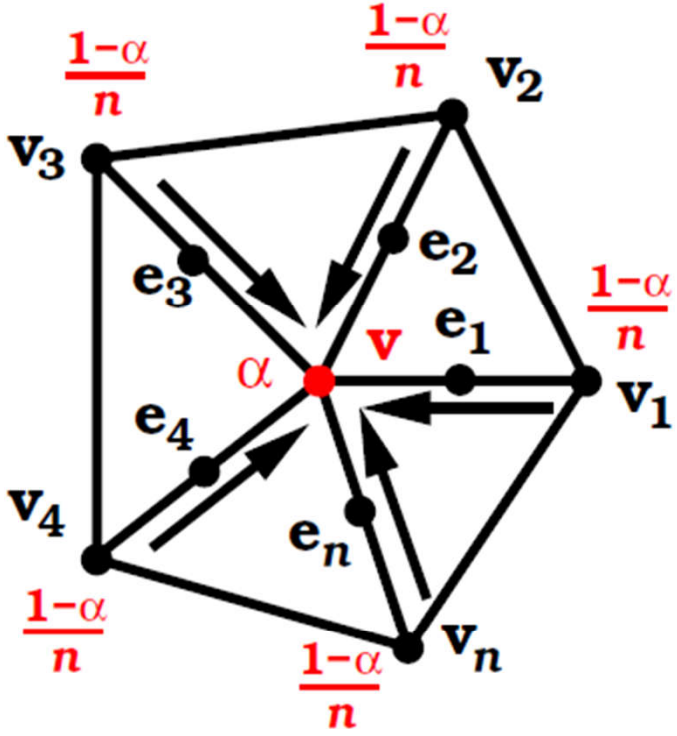
$$\mathbf{e}' = \frac{3}{8}(\mathbf{v}_0 + \mathbf{v}_1) + \frac{1}{8}(\mathbf{v}_2 + \mathbf{v}_3) \qquad (2)$$



1.1 The Loop Subdivision Scheme

Vertex Mask

After computing all edge vertices, the original vertices are re-located according to the following formula, where n is the vertex **valence** (number of edges):



$$\mathbf{v}' = \alpha(n)\mathbf{v} + \frac{1-\alpha(n)}{n}\sum_{i=1}^{n}\mathbf{v}_i,$$

$$\alpha(n) = \frac{3}{8} + \left(\frac{3}{8} + \frac{1}{4}\cos\frac{2\pi}{n}\right)^2 \qquad (3)$$

1.1 The Loop Subdivision Scheme

Alternative Computation

The vertex mask in equation (3) has the drawback, that the vertices need to be duplicated, since the original values of the neighbor vertices are taken into account (even if these have already been re-located). The following scheme is equivalent to (3), but uses only the neighboring edge-vertices, such that the v-coordinates can simply be overwritten by the new ones:

$$\mathbf{v}' = \beta(n)\mathbf{v} + \frac{1-\beta(n)}{n}\sum_{i=0}^{n}\mathbf{e}'_i, \qquad (4)$$

$$\beta(n) = \frac{8}{5}\alpha(n) - \frac{3}{5}.$$

1.1 The Loop Subdivision Scheme

Properties

Like the subdivision masks for cubic spline curves are derived from knot insertion in the **de Boor Algorithm** for B-Splines, see [Gerald Farin: Curves and Surfaces for CAGD, MK 2002], Loop used a quartic spline surface on equilateral triangles, the **quartic Box-Spline**. The loop scheme re-produces these splines, when all valences are equal to 6, according to a uniform triangle mesh.

The masks in (2) and (3) for n=6 are directly derived from the construction of Box-Splines. Loops contribution is mostly the choice of $\alpha(n)$ at **extraordinary vertices** with $n \neq 6$. Here, the mask is chosen such that the surface is nearly C^2-continuous (steady tangent plane and nearly steady curvatures).

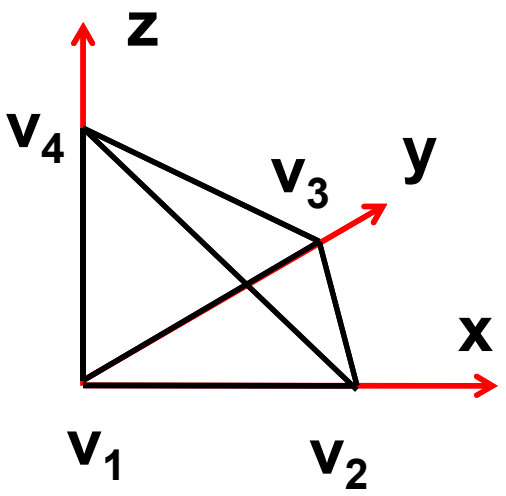
1.2 Implementation

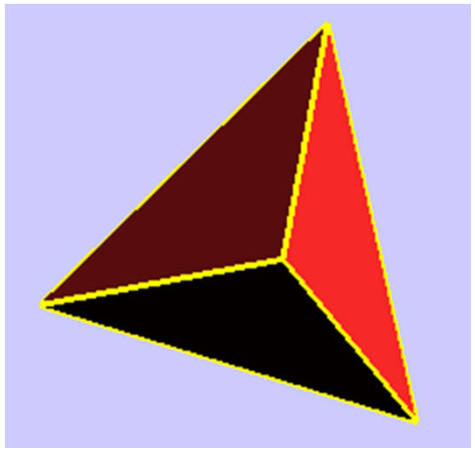
Meshes

A simple representation of meshes is the one in Maya Object Files (*.obj). Here, a list of vertices is followed by a list of faces, which can be triangles composed of three vertex indices.

Example: A Tetrahedron in OBJ

```
v   0   0   0
v   8   0   0
v   0   8   0
v   0   0   8
f   1   2   4
f   2   3   4
f   3   1   4
f   3   2   1
```





1.2 Implementation

Mesh Data Structure

To represent a triangle mesh, we need a vertex class containing the coordinates [x, y, z] and the valence n.

Further, a triangle class needs to be defined, containing the three vertex indices (e.g. denoting which vertices are used). Since in most programming languages indices start with 0, these need to be decremented when reading OBJ.

```
v0 = [0, 0, 0]      f0 = [0 1 3]
v1 = [8, 0, 0]      f1 = [1 2 3]
v2 = [0, 8, 0]      f2 = [2 0 3]
v3 = [0  0  8]      f4 = [2 1 0]
```

denoting, for example that f0 is composed of v0, v1, and v3.

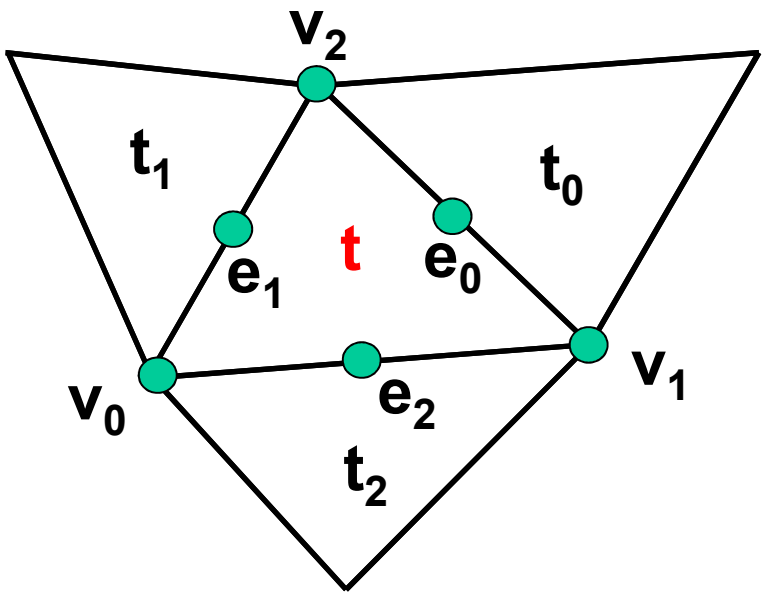
1.2 Implementation

Triangle Class

In order to access adjacency, the indices of neighboring triangles and of the e-vertices are added to the triangle class:

Example:

Every triangle **t** is linked via indices to the elements depicted to the right.
The construction is simplified a lot by setting the edge-indices to the same value as the vertex indices on the opposite side.
(e_0 and t_0 opposite to v_0, etc.)

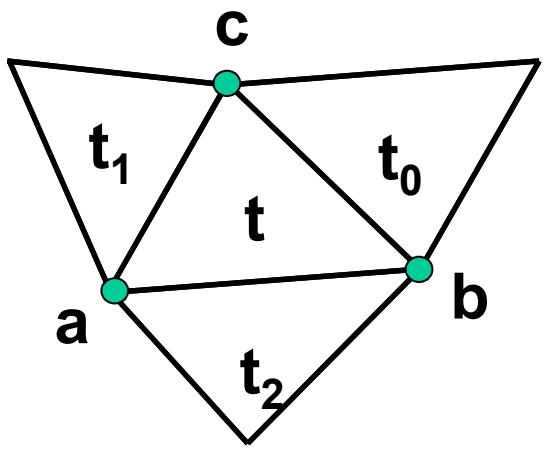


1.2 Implementation

Connectivity Algorithm

Now the following algorithm (presented in pseudo code) is used for constructing the connectivity and counting the valences:

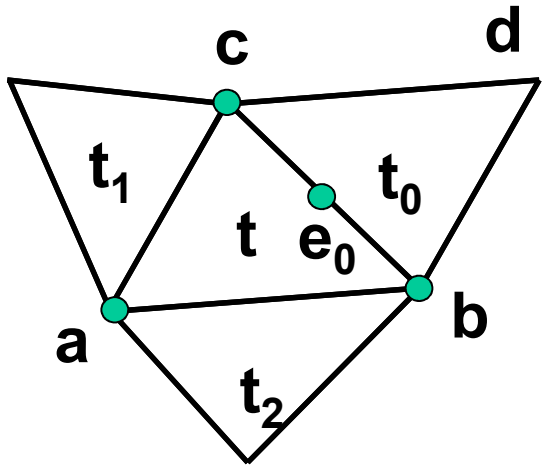
```
Input: sets of vertices V and triangles T

for each v∈V:
    n(v):=0
for each t∈T with vertices a,b,c:
    find t0≠t containing b and c
    find t1≠t containing c and a
    find t2≠t containing a and b
    n(a)++
    n(b)++
    n(c)++
```



1.2 Implementation

Subdivision Algorithm (Edge Mask)

```
for each t∈T with neighbors t0, t1, t2:
    if index(t) < index(t0)
        d = vertex of t0 with b≠d≠c
        e0 = (a + 3*b + 3*c + d)/8
        V := V ∪ {e0}
    else
        get e0 from t0
analogously construct e1 and e2
```

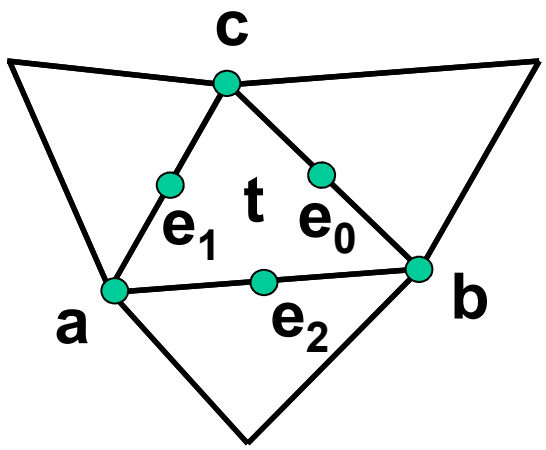


This way, it is guaranteed that
e0 is generated only once.
Finally, the vertex-mask needs to be computed based on equation (4):

1.2 Implementation

Subdivision Algorithm (Vertex Mask)

```
for each v∈V:
    v := β(v)*v
for each t∈T with vertices a,b,c:
    a += (1-β(a))/(n(a))*(e1+e2)/2
    b += ...
    c += ...
```



Finally, the triangles are replaced:

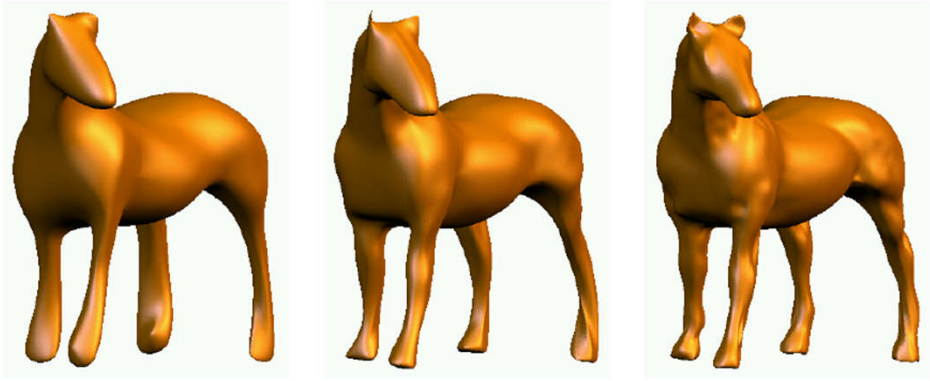
```
for each t∈T with vertices a,b,c:
    replace t by [e1,e0,c]
    T := T ∪ {[e1,e2,e0]}
    T := T ∪ {[a,e2,e1]}
    T := T ∪ {[e2,b,e0]}
```

1.2 Implementation

Subdivision Algorithm

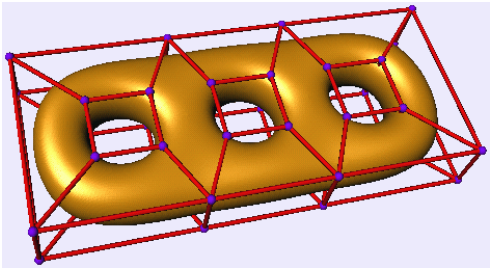
The algorithm computes the vertex masks in parts by traversing the triangles rather than the vertices. This way, the vertex class does not require indexing to all adjacent triangles.

The triangle connectivity can be initialized in the last step, or alternatively, with the connectivity algorithm on the next subdivision.



Loop subdivision surfaces from different resolution meshes of the Caltec Horse Model (data from http://www.multires.caltech.edu)

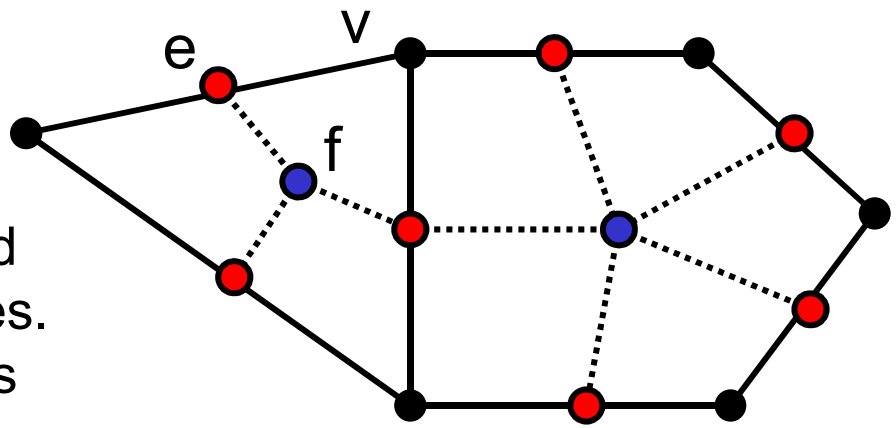
1.3 Catmull-Clark Subdivision



Recursively Generated Surfaces

The second subdivision scheme we introduce is the one by Catmull / Clark [E. Catmull and J. Clark, Recursively generated B-spline surfaces on arbitrary topological meshes, Computer Aided Design, Vol. 10, No. 6, 1978, pp. 350-355].

The scheme works for arbitrary polygons. It generates only quadrilaterals by inserting a new vertex for every face, another one for every edge, and by modifying the original vertices. Therefore, three different masks are necessary.



1.3 Catmull-Clark Subdivision

Cubic Splines

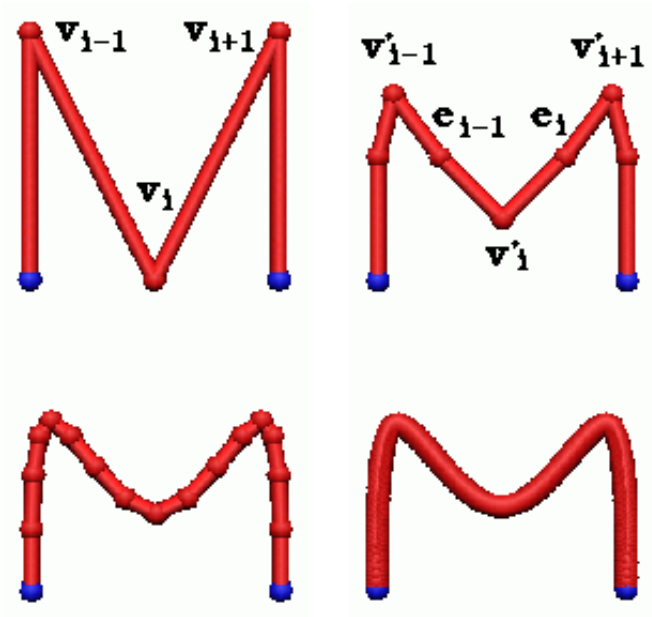
We illustrate the principle of subdivision by the example of cubic spline curves. Starting with a polygon $[v_1\, v_2, \ldots, v_n]^T$, in a first pass, the midpoints are inserted:

$$e_i := 0.5(\, v_i + v_{i+1})$$

Second, the original vertices are re-located:

$$v_i := 0.25(\, e_{i-1} + 2v_i + e_i \,)$$

This scheme is applied repeatedly, providing a piecewise cubic curve.



The scheme is equivalent to the **subdivision mask** on the next slide. Prove equivalence as exercise!

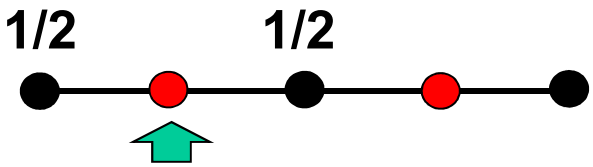
1.3 Catmull-Clark Subdivision

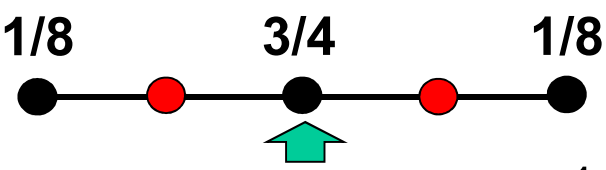
Cubic Spline Subdivision Mask

$$\begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ \\ \\ \\ \\ \\ v_{2n-1} \end{bmatrix} := \begin{bmatrix} 1 & & & & & & \\ 1/2 & 1/2 & & & & & \\ 1/8 & 3/4 & 1/8 & & & & \\ & 1/2 & 1/2 & & & & \\ & 1/8 & 3/4 & 1/8 & & & \\ & & 1/2 & 1/2 & & & \\ & & & & \ddots & & \\ & & & & & 1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ \\ \\ \\ \\ \\ v_n \end{bmatrix} \qquad (5)$$

1.3 Catmull-Clark Subdivision

From Curves to Surfaces



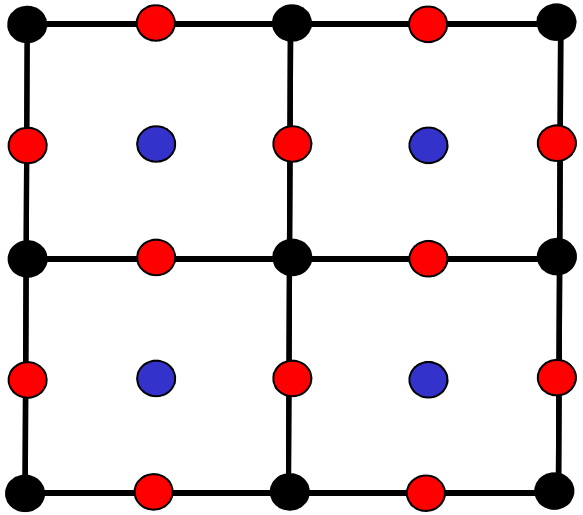


edge mask: $e = \overline{v}_e$,

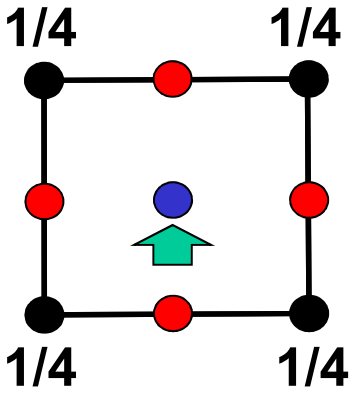
where $\overline{a}_b$ denotes the average of all type-a vertices adjacent to b.

vertex mask: $v_{new} = \frac{1}{2}(v + \overline{e}_v)$.

In the surface case, the two masks are applied to the rows and columns of a regular rectilinear grid, resulting in three new masks for faces, edges, and vertices.

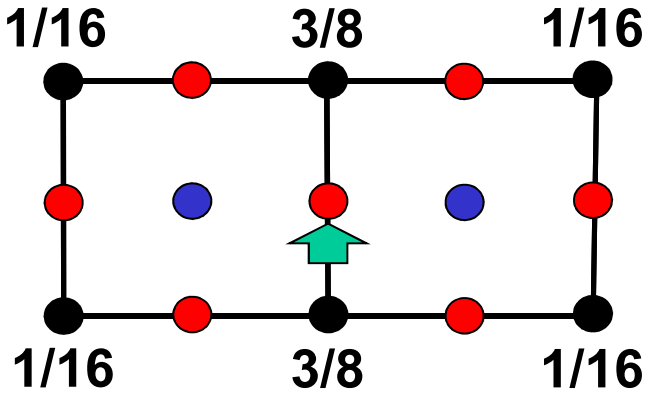


1.3 Catmull-Clark Subdivision

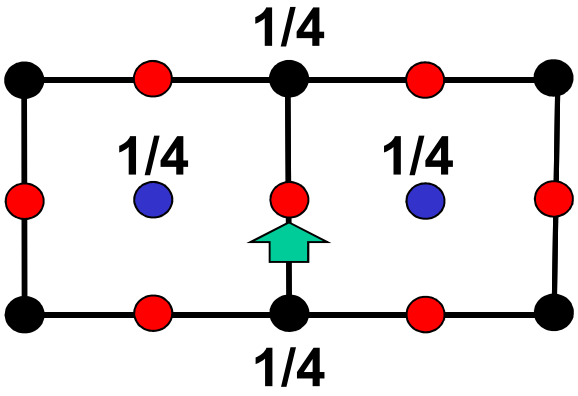


face mask

$$f = \bar{v}_f$$



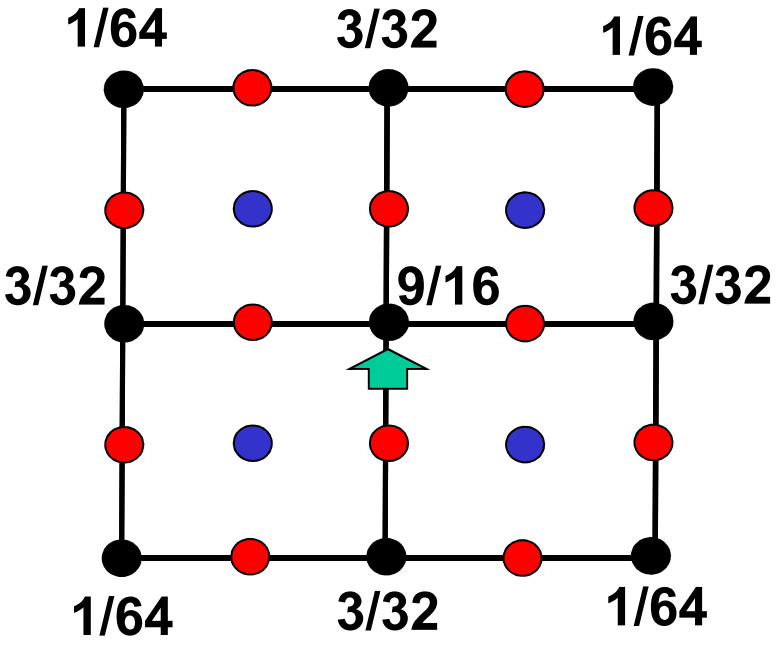
edge mask



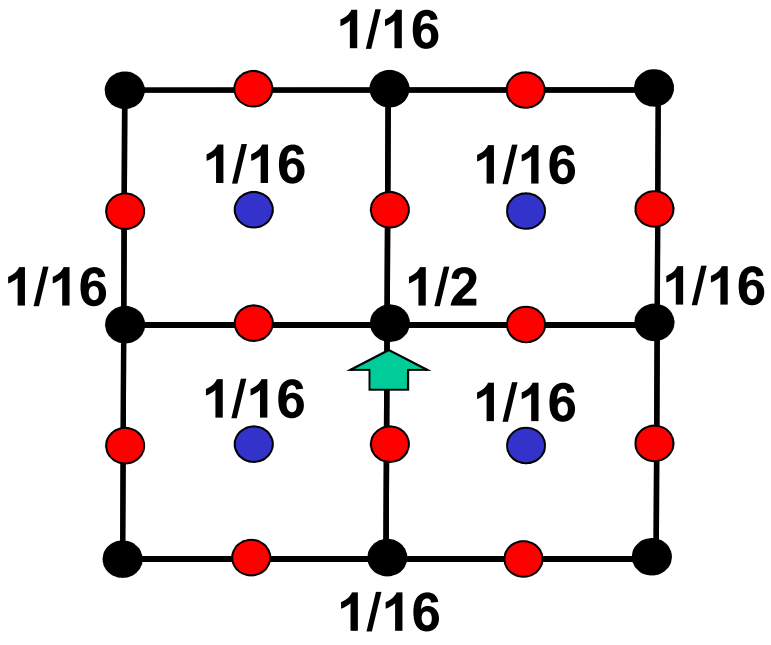
$$e = \frac{1}{2}(\bar{v}_e + \bar{f}_e)$$

alternative edge mask

1.3 Catmull-Clark Subdivision



vertex mask



alternative vertex mask

$$v_{new} = \frac{1}{4}\left(\bar{f}_v + \bar{v}_v + 2v\right)$$

1.3 Catmull-Clark Subdivision

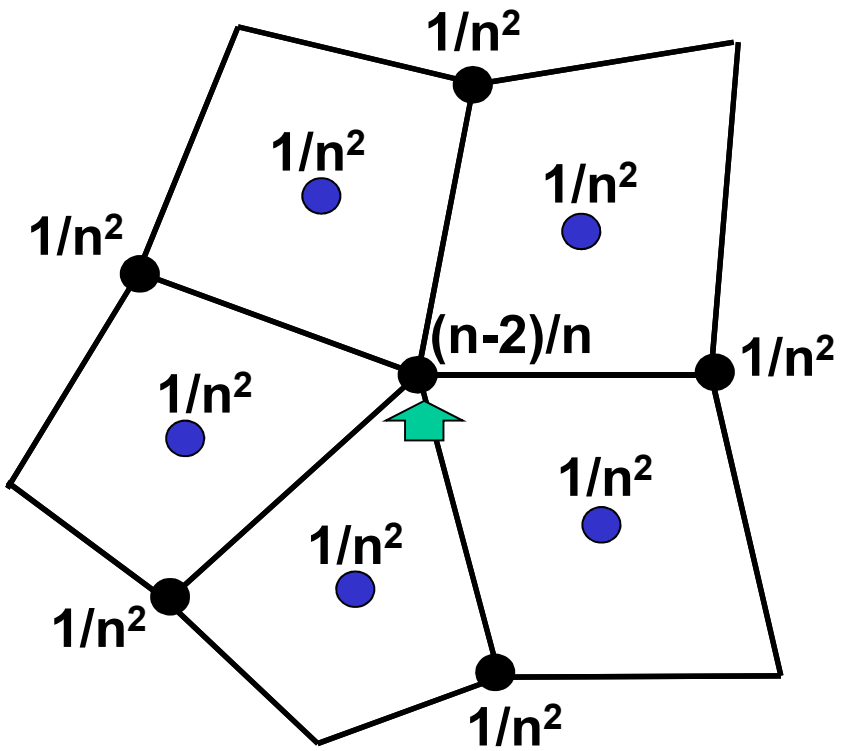
CC Subdivision Masks

The key idea of Catmull / Clark is to generalize the vertex mask to extraordinary vertices, where the valence n_v (number of edges) is different from four:

$$f = \bar{v}_f$$

$$e = \frac{1}{2}(\bar{v}_e + \bar{f}_e) \qquad (6)$$

$$v_{new} = \frac{1}{n_v}\left(\bar{f}_v + \bar{v}_v + (n_v - 2)v\right)$$



CC vertex mask
(note that averaging also divides by n)

1.3 Catmull-Clark Subdivision

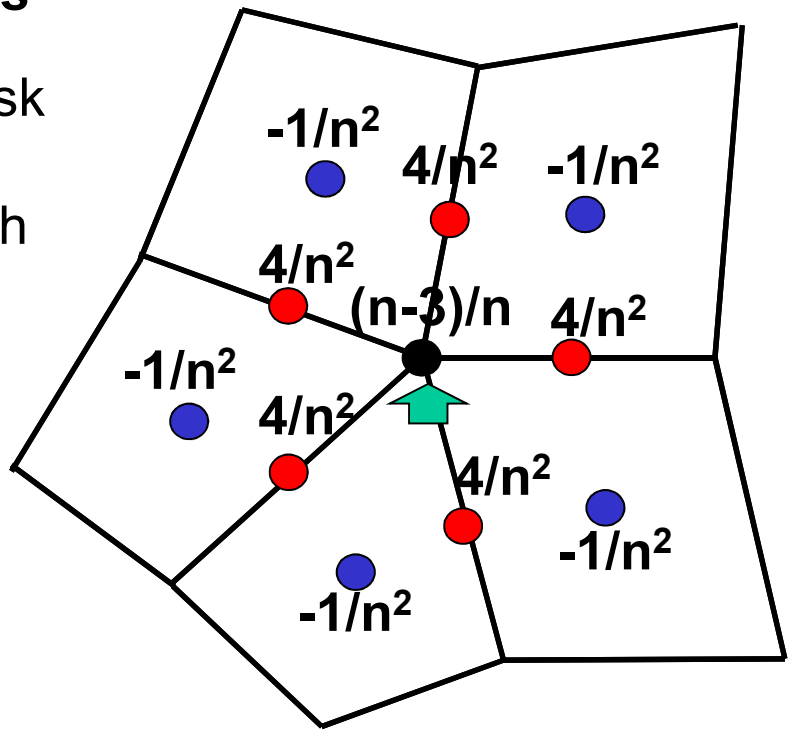
Alternative CC Subdivision Masks

Note that for the original vertex mask all vertices need to be duplicated, since the v-vertices depend on each other. To avoid this, an equivalent local computation is possible:

$$f = \bar{v}_f$$

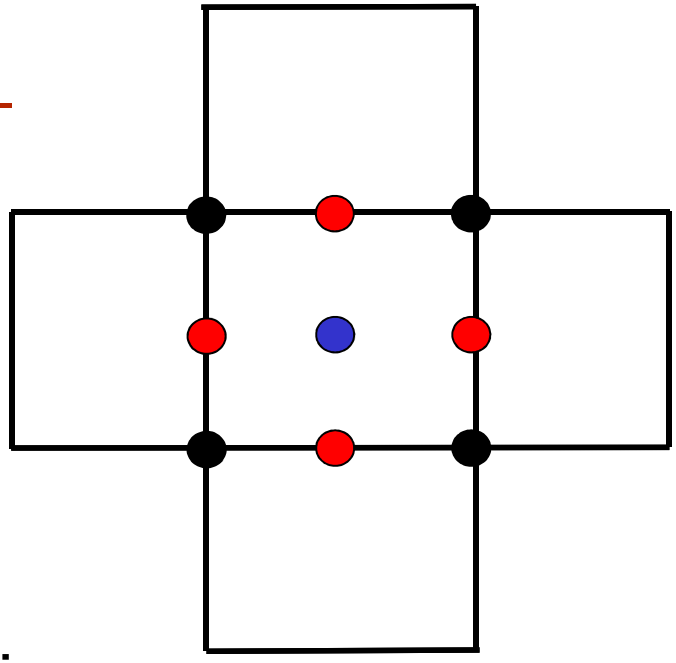
$$e = \frac{1}{2}(\bar{v}_e + \bar{f}_e)$$

$$v_{new} = \frac{1}{n_v}\left(4\bar{e}_v - \bar{f}_v + (n_v - 3)v\right) \quad (7)$$



alternative CC vertex mask (using negative weights for f)

1.3 Catmull-Clark Subdivision



Implementation

The implementation for the Catmull-Clark scheme requires the following data structure for quadrilaterals:

```
quad{
    int v[4] // vertex indices
    int e[4] // edge vertex indices
    int f    // face vertex index
    int q[4] // adjacent quad indices
}
```

Every vertex needs to store its valence n and the three coordinates. The mesh is composed of a vector of quads and a vector of vertices.

1.3 Catmull-Clark Subdivision

Implementation

Connectivity: After reading the mesh, the quad adjacencies and the vertex valences are determined by traversing the quads.

Subdivision:

1. For each quad: compute the face vertex, add it to the vertex vector and record its index in in the quad instance.
2. For each quad, for each edge: if the adjacent quad has a greater index than the current one, compute the edge vertex, add it to the vertex vector and record its index. Otherwise, the edge vertex already exists and its index can be copied from the adjacent quad.
3. Multiply each v-vertex by n(n-3), where n is its valence.
4. For each quad, for each vertex: add each adjacent e-vertex twice and subtract f, according to equation (7).
5. Divide each v-vertex by n^2.

1.3 Catmull-Clark Subdivision

Implementation

6. Replace each quad by four smaller ones, connected to the corresponding vertices. Connect each quad to its four neighbors.

Remark: Steps 3-5 correspond to the alternative vertex mask (7). The e-vertices are multiplied with 2 (rather than 4), since the adjacent quads will also add the same vertex.

To obtain piecewise smooth surfaces with sharp creases and to allow boundary curves, the univariate masks can be applied along specific edges marked in the polygon mesh (in red).