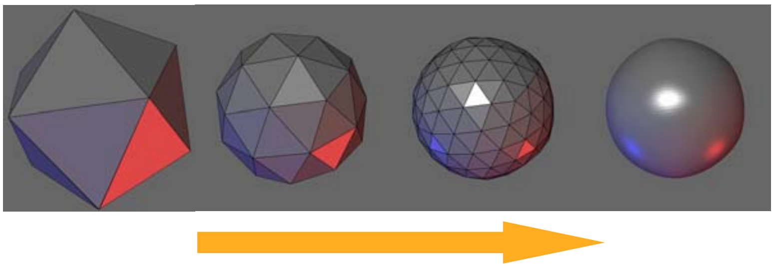
Example: Loop Scheme

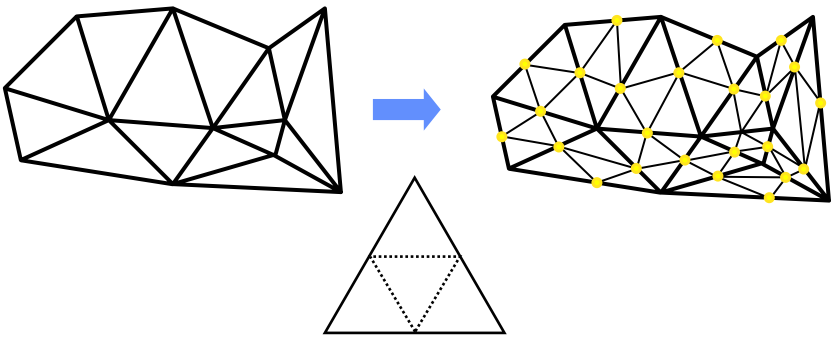
What makes a good scheme?



- **recursive application leads to a smooth surface**

Example: Loop Scheme

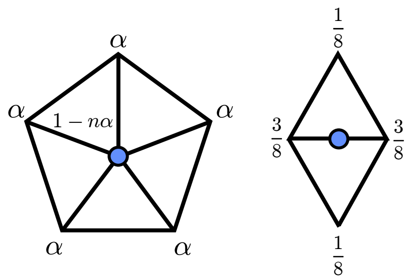
Refinement rule

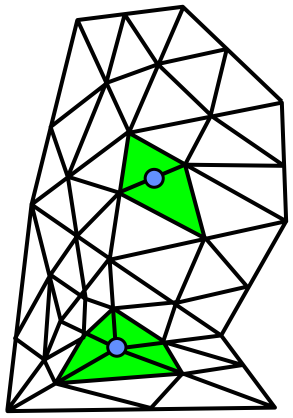


Example: Loop Scheme

Two geometric rules:

- even (update old points)
- odd (insert new)



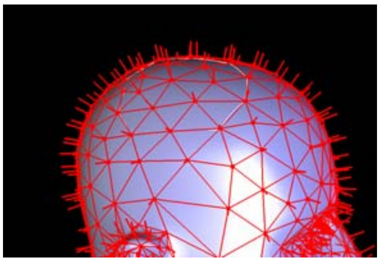


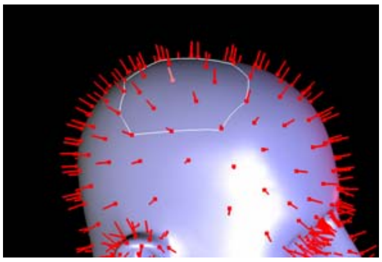
$\alpha = 3/8n,\ n > 3,\ \ \alpha = 3/16,$ if $n=3$

Control Points

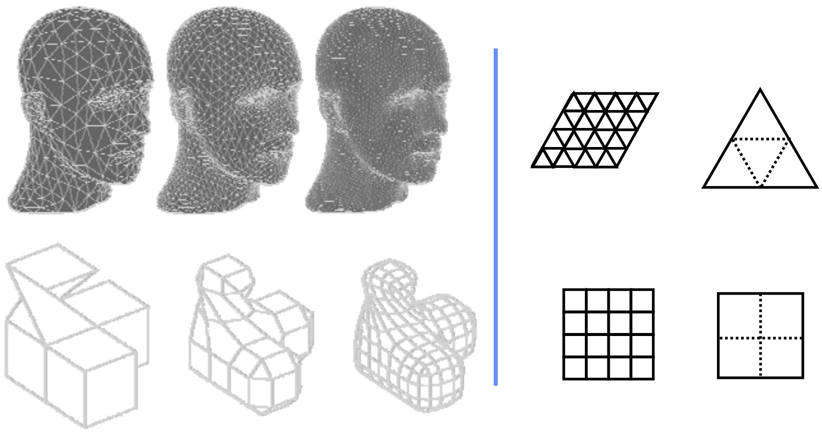
Vertices of initial mesh

- define the surface
- each influences finite part of surface





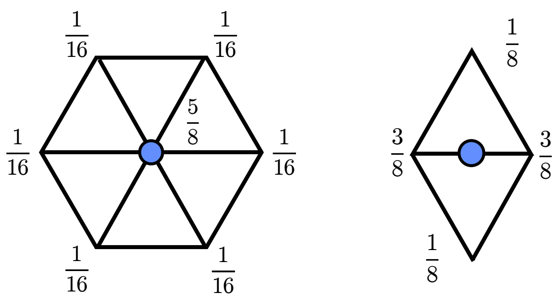
Triangles and Quads



Subdivision and Splines

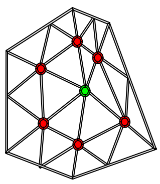
Uniform splines

- **can be computed using subdivision**
- **quartic box spline rules:**

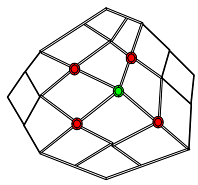


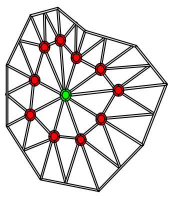
Extraordinary Vertices

Triangle meshes **Quad meshes**

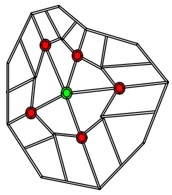


regular





extraordinary

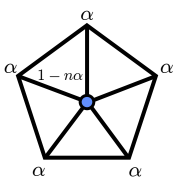


Constructing the Rules

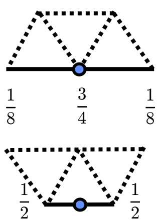
Start with spline rules

- **define rules for:**

Extraordinary vertices



Boundaries

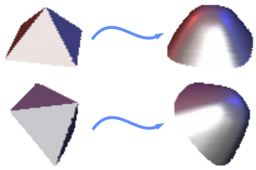


Creases etc.

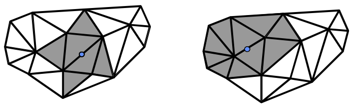
• • •

Constructing the Rules

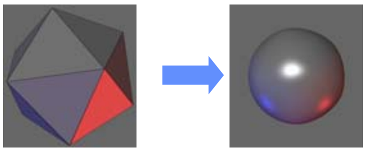
- invariance under rotations and translations



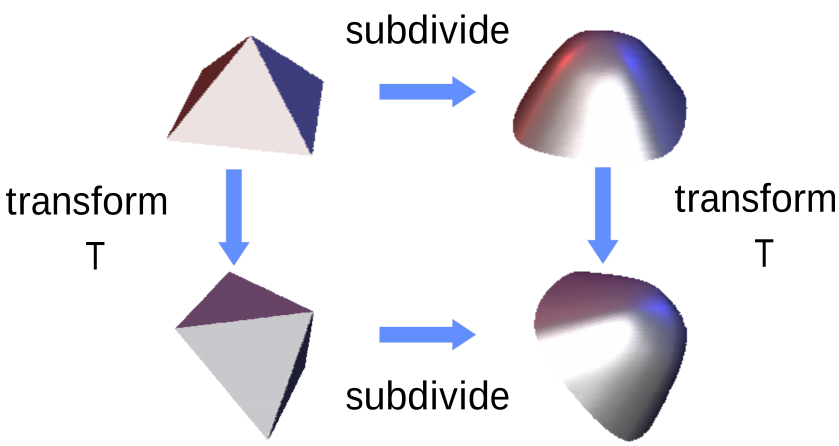
- small support



- smoothness and Fairness

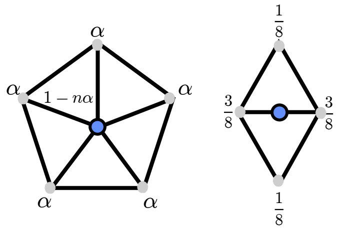


Invariance w.r.t rigid transforms



Invariance

Coefficients of masks must sum to 1

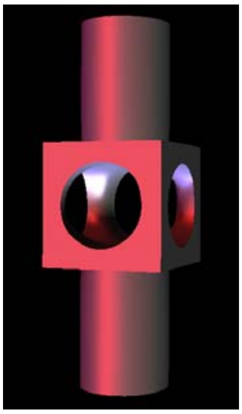


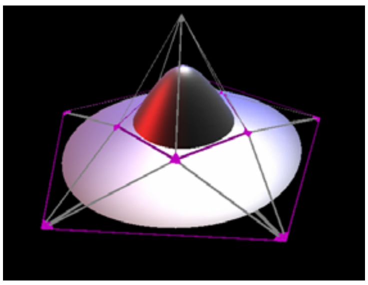
$$p = \sum a_i p_i$$

displacement

$$\sum a_i(p_i + t) = \underbrace{\left(\sum a_i\right)}_{1} t + p$$

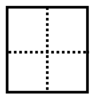
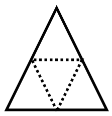
Crease Examples





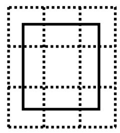
Subdivision Schemes

Primal

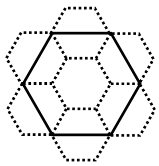
	Approx.	Interp.
	Catmull-Clark	Kobbelt
	Loop	Butterfly

Dual

- **(no interpolation)**



Doo-Sabin, Midedge

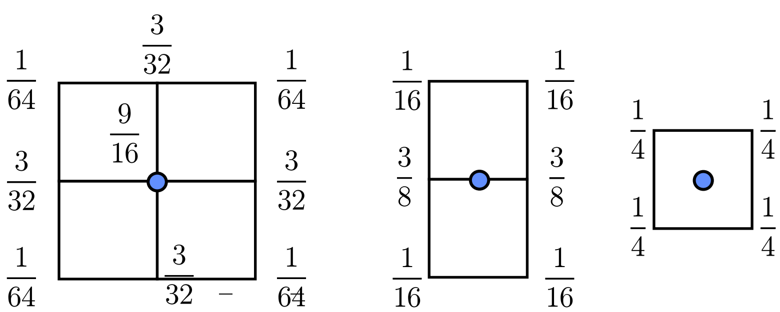


Dyn-Levin-Liu (non-linear)

Catmull-Clark Scheme

Primal, quadrilateral, approximating

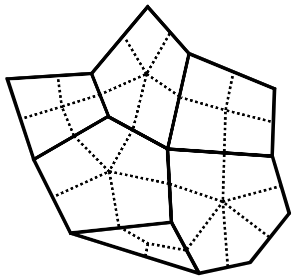
- **tensor-product bicubic splines**



Catmull-Clark Scheme

Reduction to a quadrilateral mesh

- **do one step of subdivision with special rules: only quads remain**

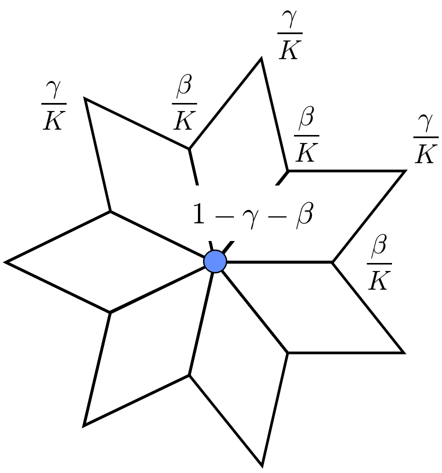


Catmull-Clark Scheme

Extraordinary vertices

$$\gamma = \frac{1}{4K}$$

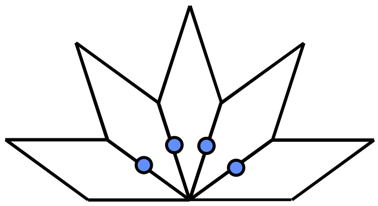
$$\beta = \frac{3}{2K}$$

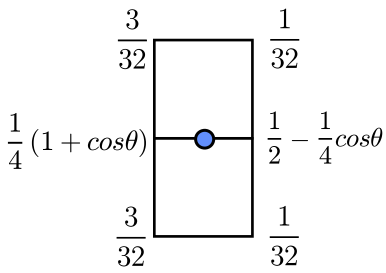


Catmull-Clark Scheme

Boundaries, creases, corners

- cubic spline (same as Loop!)
- need to fix rules for C^1-continuity





Implementing subdivision

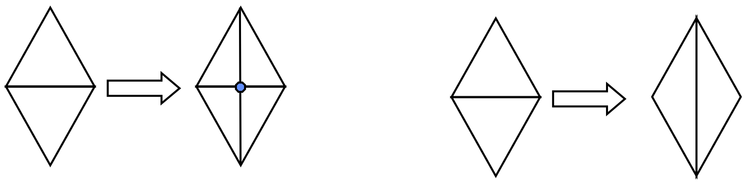
Operations needed:

- create a copy of the mesh maintaining vertex correspondence with the old mesh
- refine a mesh
- collect all neighbors of a vertex (for updating positions of old vertices, discussed at the last lecture)
- find vertices of two triangles sharing an edge (for computing positions of new vertices)

Implementing subdivision

Uniform refinement

- can be achieved using two simple operations

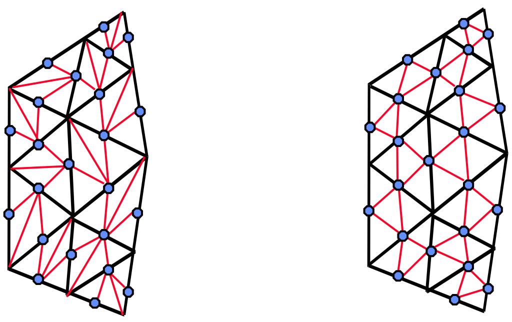


split two triangles adding a vertex

edge flip

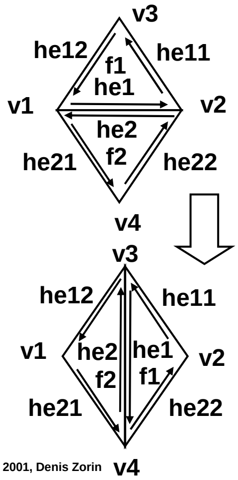
Implementing subdivision

Step 1 (left): split all edges in any order, adding vertices for every edge and spit adjacent triangles in to two
Step 2 (right): flip all edges connecting an old vertex with a newly inserted one



Implementing an edge flip

Example: given a pair of half-edges he1,he2 flip the corresponding edge




```
he1.next = he22; he1.vertex = v4;
he2.next = he12; he2.vertex = v3;
he11.next = he1;
h12.next =  he21; h12.face = f2;
h21.next = he2;
he22.next = he11; he22.face = f1;
if (f2.halfedge == he22)
   f2.halfedge = he12;
if (f1.halfedge == he12)
   f1. halfedge = he22;
if(v1.halfedge == he1)
   v1.halfedge = he21;
if(v2.halfedge == he2)
   v2.halfedge = he11;
```


Building a half-edge data structure

Similar to building face-based triangular mesh

Input: a list of vertices, a list of faces, each face is a list of vertex indices enumerated CCW

1. Create arrays of vertices, faces and halfedges, one half-edge for every seq. pair of vertices of every face; initialize all pointers to zero.

2. For each face `f`**, with n vertices**

assign `f.halfedge` **to its first half-edge;**
for each vertex `v` **of a face, assign** `v->halfedge` **to the halfedge starting at it if nothing is assigned to it yet;**
for each half-fedge `he` **of a face, assign**
`he.face =f, he->next =`**next half-edge in the face,**
`he->vertex =` **next vertex in the face;**
record half-edge pointer he in the edge map:

```
edgemap(v[i],v[i+1]) = he
```

3. Go over all entries of the edge map, assign for half-edges `edgemap(i,j) edgemap(i,j)`

links to each other if both exist

Dealing with boundaries

To minimize implementation effort it is useful to create two halfedges for boundary edges, one of which has zero face pointer;

A boundary vertex `v` should always have `v.halfedge`

pointing to a boundary halfedge.

Then it is easy e.g. to find two boundary neighbors of a vertex.