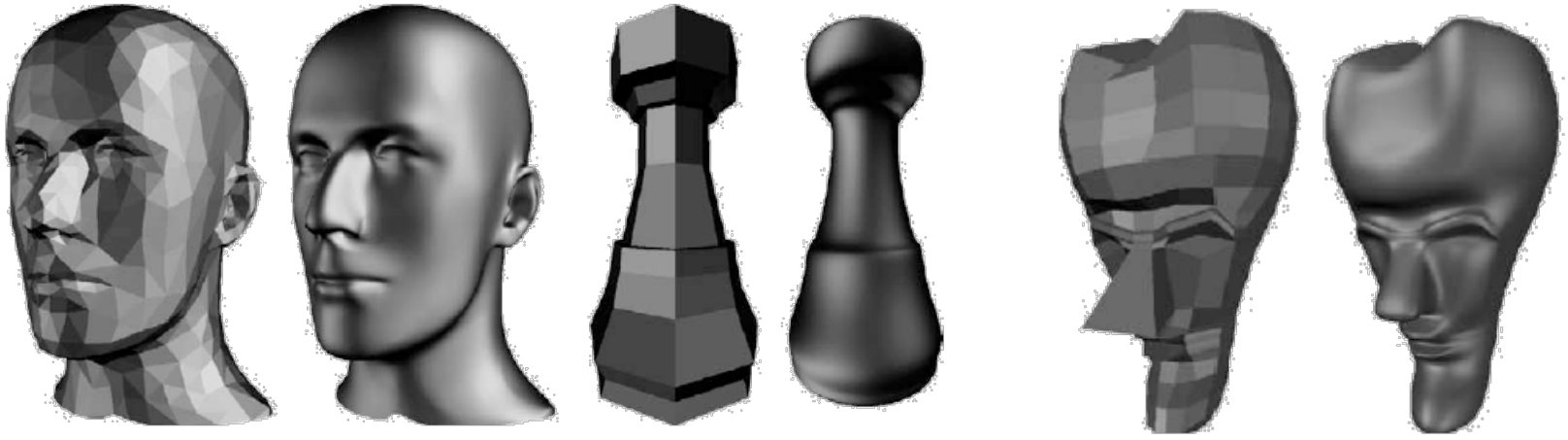
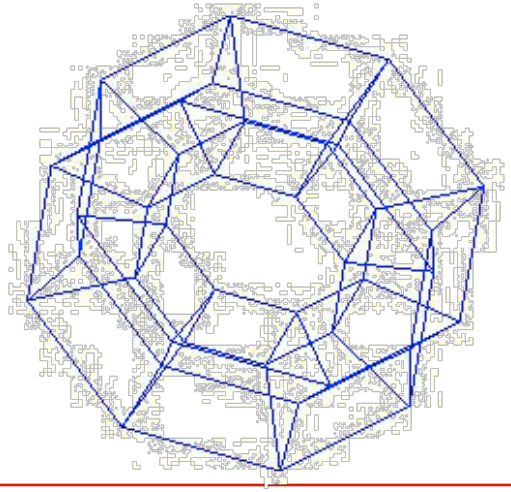
Regular subdivision

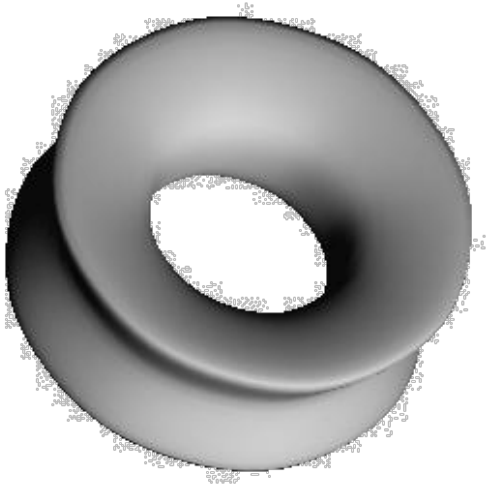


Loop 87 Catmull&Clark 78 Zorin&Schröder 01

- Use split&tweak curve subdivision to create smooth surfaces



Presti



T-mesh subdivision

Georgia Tech College of Computing
Interactive & Intelligent Computing

- Assume Triangle mesh with arbitrary genus

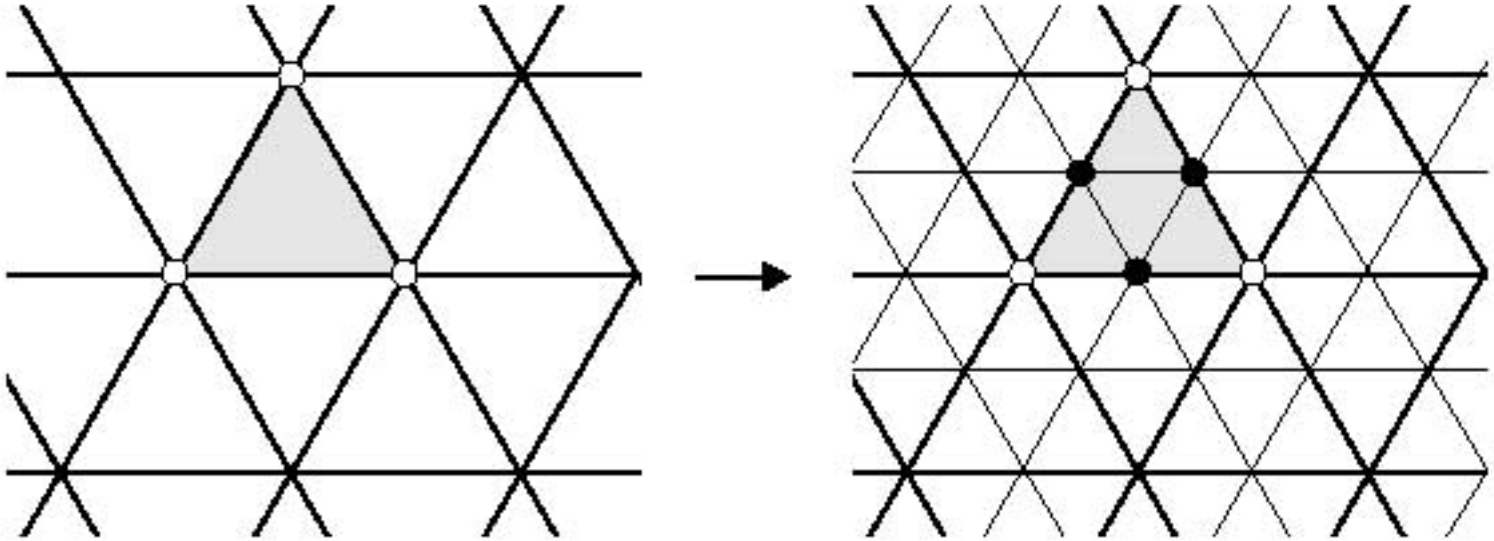
Lecture Objectives

- Learn how to implement a regular subdivision for manifold triangle meshes using a butterfly scheme to compute the new vertices at each stage.

- Reading:
  - Lear how to extend the above to an adaptive subdivision by studying the research paper: "A Sub-Atomic Subdivision Approach" by **S. Seeger, K. Hormann G. Hausler and G. Greiner**

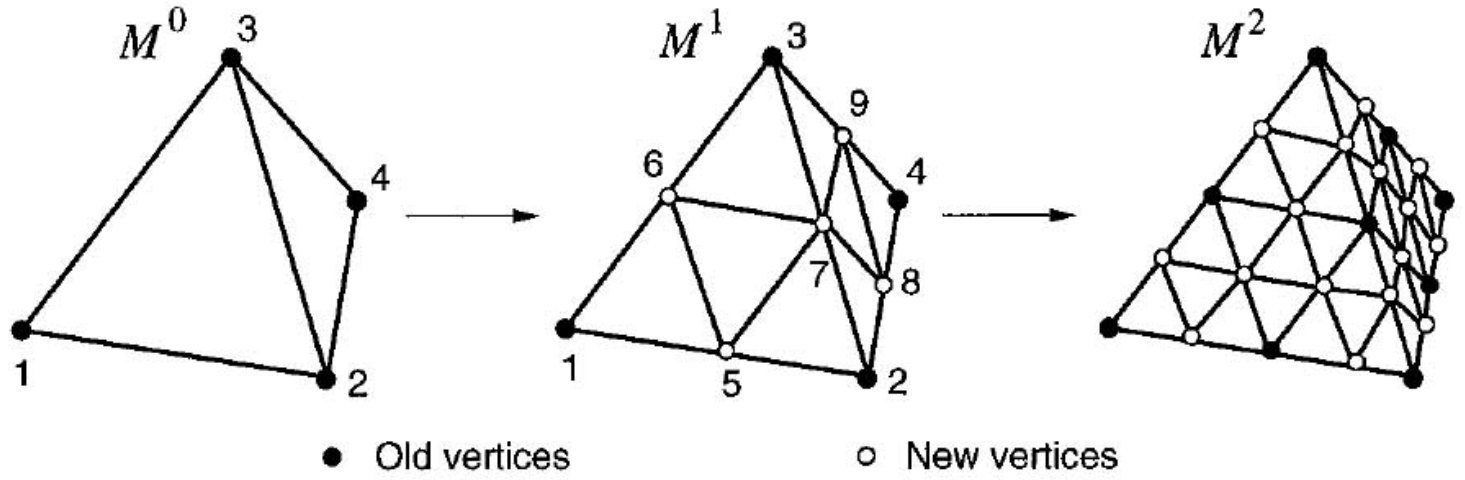
Subdivision

- Insert a new vertex in the middle of each edge
- Split each triangle into 4 triangles



An example of 2 iterations on a tetrahedron

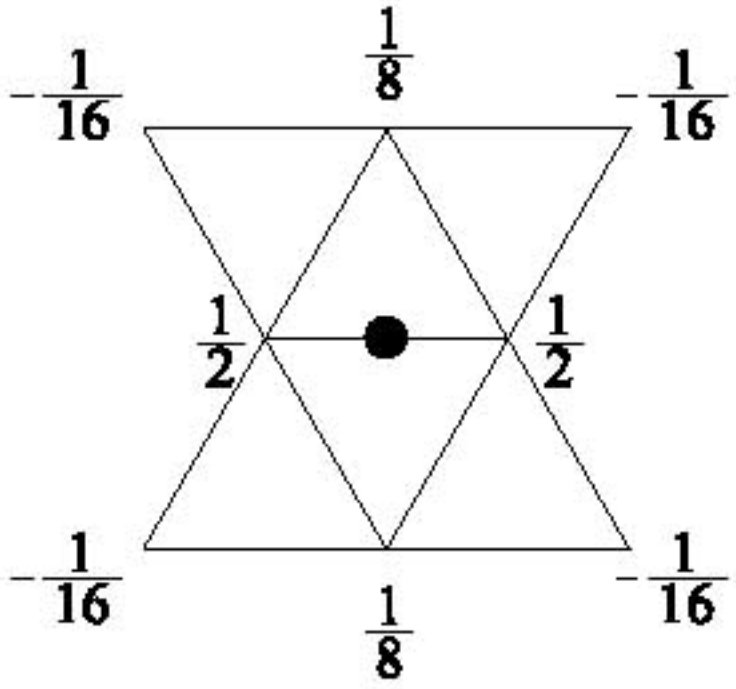
- Repeat the operation several times



- BUT, at each step, you will adjust the position of the new vertices, using the butterfly rule

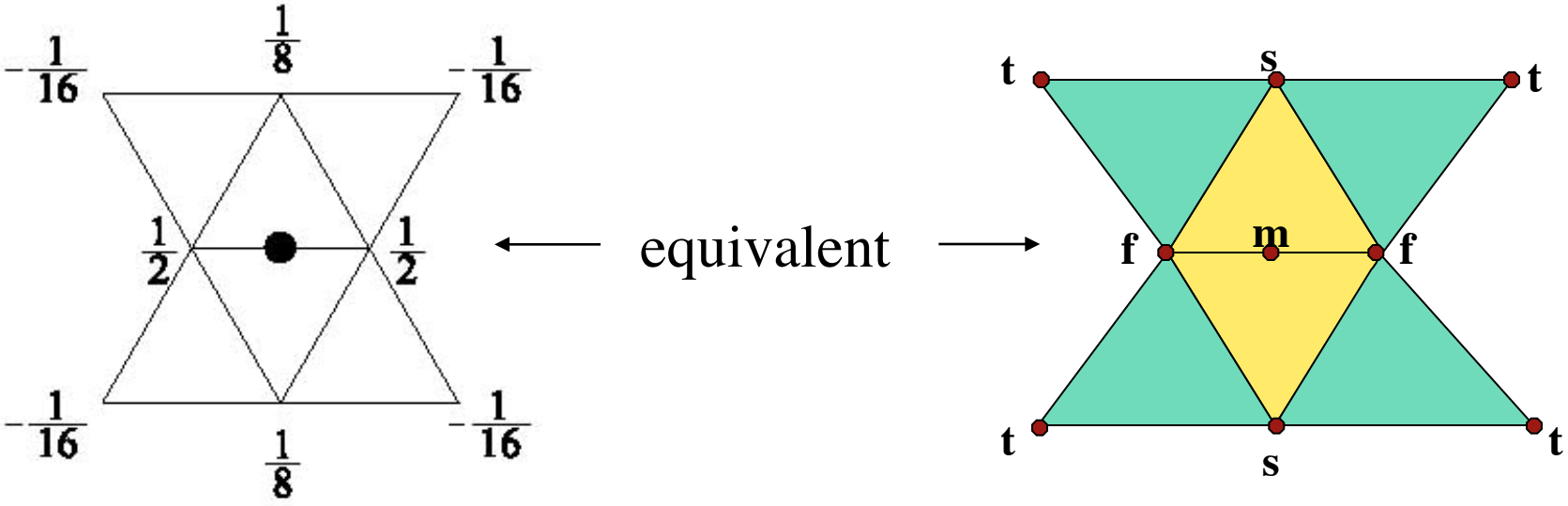
Butterfly mask

- To compute the position of the new (mid-edge) point, multiply its neighbors by the corresponding coefficients and add them up.



A more intuitive formulation

- First insert the new points in the middle of the edges
  - m = **average**(f)
- Then compute adjustment vectors
  - d = (**average**(s) – **average**(t))/4
- Then adjust all the m points by displacing them by the corresponding d
  - m = m + d
- Then compute the new triangles



Subdivision a triangle mesh k times

- Read the input file and construct the X, Y, Z, V tables
  - Allocate X[N4^k], Y[N4^k], and Z[N4^k] and fill the first N entries
- Repeat the following process k times
  - Compute the O table
    - For each corner c, find the opposite corner b, then O[c]:=b; O[b]:=c;
  - Compute mid-edge points
    - For each corner c do
      - n+=1; (X[n],Y[n],Z[n]):=**average**(f)+(**average**(s) – **average**(t))/4;
      - M[c]:=M[c.o]:=n;
  - Compute a new V-table (4 times longer than the previous one)
    - For each corner c do
      - V[n++]=c.v; V[n++]=c.p.m.v; … // 12 entries
- Write the X, Y, Z and the V tables on file

Input/output (file) format

- N # the number of vertices
  - x[0] y[0] z[0]
  - x[1] y[1] z[1]
  - …
  - x[N-1] y[N-1] z[N-1]
- T # the number of triangles
  - V[0] V[1] V[2] # the Integer IDs of vertices for triangle 0
  - V[3] V[4] V[5] # the Integer IDs of vertices for triangle 1
  - V[3] V[4] V[5] # the Integer IDs of vertices for triangle 2
  - V[6] V[7] V[8]
  - …
  - V[3T-3] V[3T-2] V[3T-1] # the Integer IDs of vertices for triangle T-1

How to construct the O table

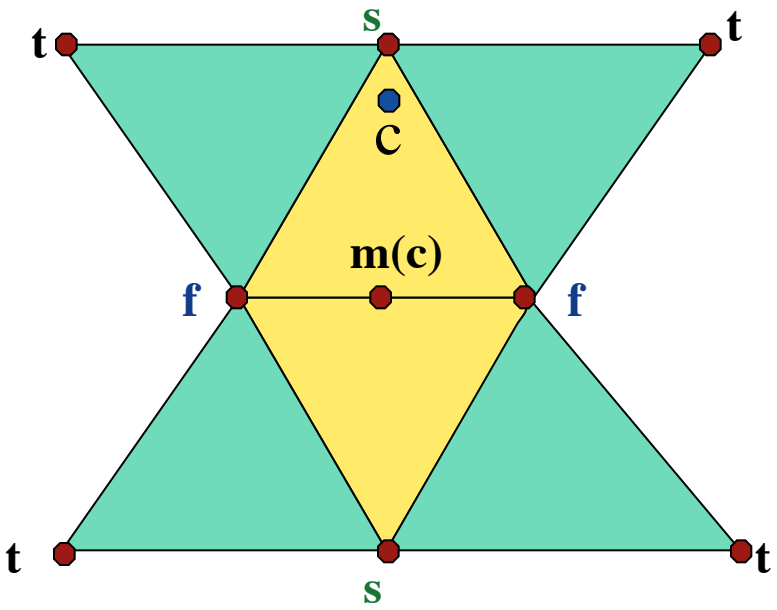
```
FOR c:=0 TO 3T-2 DO
    FOR b:=c+1 TO 3T-1 DO
        IF (c.n.v==b.p.v) && (c.p.v==b.n.v)
        THEN { O[c]:=b; O[b]:=c}
```

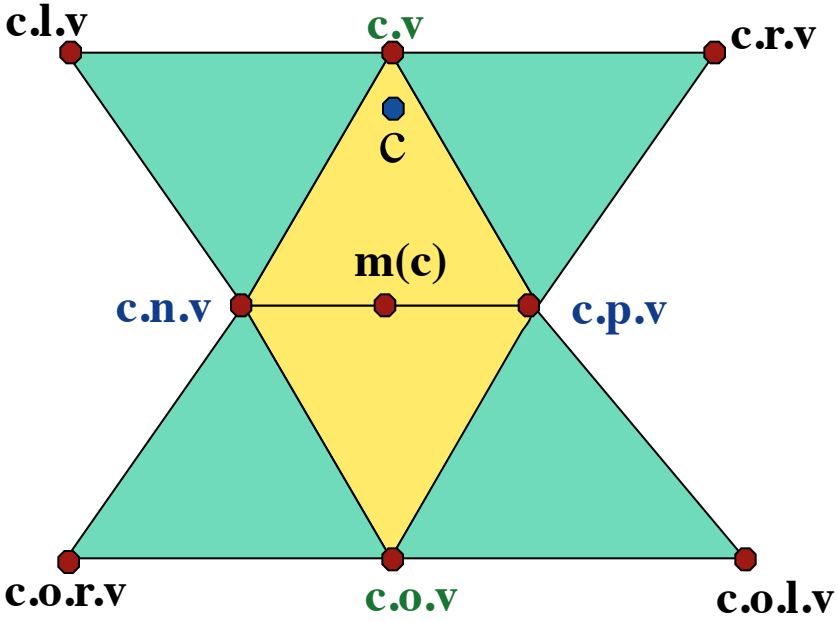
How to access the neighbors from a corner

Given a corner c, write down the complete expression for

– m(c) = **average**(f) + (**average**(s) – **average**(t))/4

using .n, .p, .o, .l, .r, and .v operators

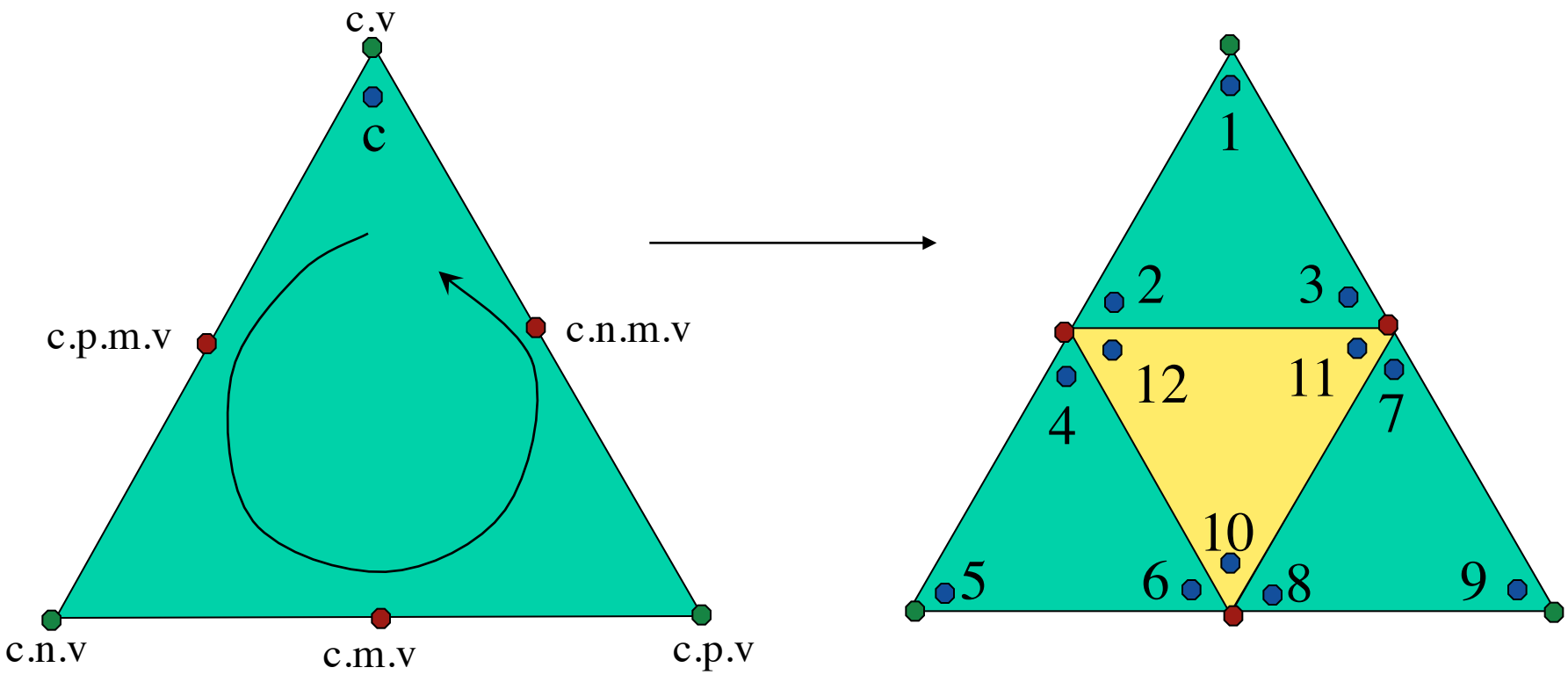




m(c)= (c.n.v+c.p.v)/2+((c.v+c.0.v)/2–(c.l.v+c.r.v+c.o.l.v+c.o.r.v)/4)/4

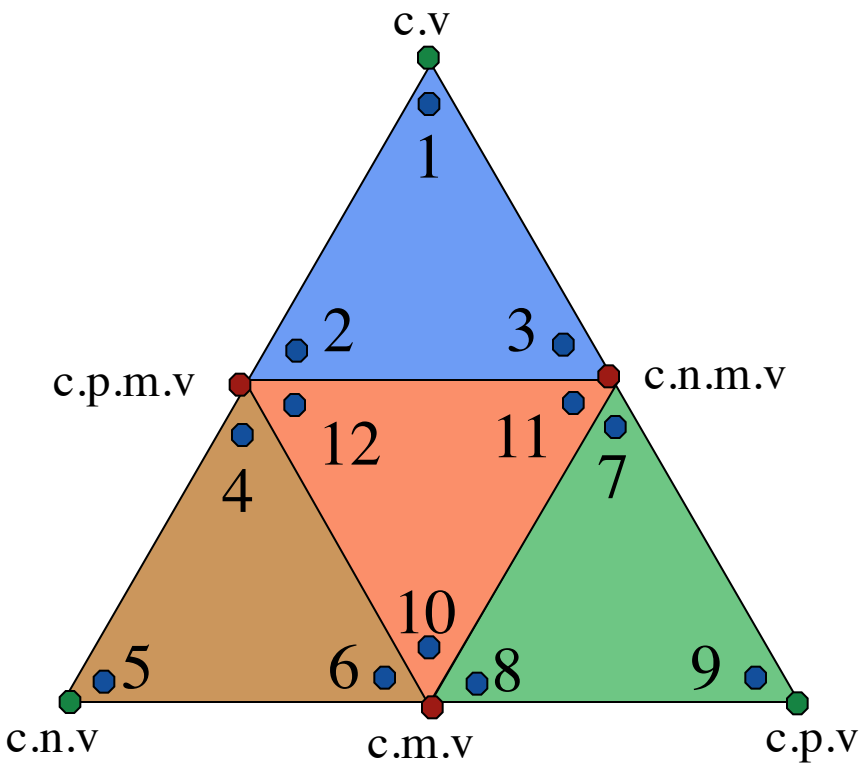
How to split a triangle

Given a corner c, write down the complete expression for the next 12 entries of V as shown using .n, .p, .o, .l, .r, and .v operators



Generating the V entries for 4 new triangles

- Write the 12 V-table entries in terms of c



c.v

c.p.m.v

c.n.m.v

c.p.m.v

c.n.v

c.m.v

c.n.m.v

c.m.v

c.p.v

c.m.v

c..n.m.v

c.p.m.v

Complete T-mesh subdivision algorithm

```
read k; allocate(X[4k], Y[4k], Z[4k]); for i:=0 to k–1 do read(X[i],Y[i],Z[i]);
read t, allocate(V[3t]); for i:=0 to 3t–1 do read(V[i]);
Allocate(O[3t],F[3t],M[3t],W[12t]);
for c:=0 to 3t–2 do for b:=c+1 to 3t-1 do
        if (c.n.v==b.p.v) && (c.p.v ==b.n.v) then {O[c]:=b; O[b]:=c};
i:=k–1; for c:=0 to 3t–1 do {M[c]:=M[c.o]:=++i;
    (X[i],Y[i],Z[i]):=(c.n.v+c.p.v)/2+((c.v+c.o.v)/2–(c.l.v+c.r.v+c.o.l.v+c.o.r.v)/4)/4};
i:=0; for c:=0 to 3t–1 do if c.f then {
    F[c]:=F[c.n]:=F[c.p]:=false;
    W[i++]:= c.v; W[i++]:= c.p.m.v; W[i++]:= c.n.m.v;
    W[i++]:= c.p.m.v; W[i++]:= c.n.v; W[i++]:= c.m.v;
    W[i++]:= c.n.m.v; W[i++]:= c.m.v; W[i++]:= c.p.v;
    W[i++]:=  c.m.v; W[i++]:= c.n.m.v; W[i++]:= c.p.m.v};
write (4k, X, Y, Z, 4t, W);
```

Rebuilding the O table directly (Stephen Ingram)

```
# this program takes a T-mesh and  computes a subdivided T-mesh using the  butterfly subdivision
read k; allocate(X[4k], Y[4k], Z[4k]); for i:=0 to k-1 do read(X[i],Y[i],Z[i]); # k = number of vertices
read t, allocate(V[3t], O[3t]); for i:=0 to 3t-1 do read(V[i]); # t = number of triangles
for i:=0 to 3t-1 do read(O[i]); #read in opposites from file
allocate(nO[12t], F[3t], M[3t], W[12t]);
# F = true/false it has been fourthed, M = midpoints, nO is new Opposite array
# compute each point's midpoint and its opposite point's midpoint (the same)
 i:=k-1; for c:=0 to 3t-1 do {M[c]:=M[c.o]:=++i;
        (X[i],Y[i],Z[i]):=(c.n.v+c.p.v)/2+((c.v+c.o.v)/2-(c.l.v+c.r.v+c.o.l.v+c.o.r.v)/4)/4};
# compute triangles for new surface
 i:=0; for c:=0 to 3t-1 do if c.f then {
        F[c]:=F[c.n]:=F[c.p]:=false;    # to avoid recalculating a new triangle
        W[i]:= c.v; nO[i++]:=c.m.v;
       W[i]:= c.p.m.v; nO[i++]:=c.n.o.p.m.v;
       W[i]:= c.n.m.v; nO[i++]:=c.p.o.n.m.v;# W = array of new triangles
        W[i]:= c.p.m.v; nO[i++]:=c.p.v;
       W[i]:= c.n.v; nO[i++]:=c.n.m.v;
       W[i]:= c.m.v; nO[i++]:=c.v;
        W[i]:= c.n.m.v; nO[i++]:=c.n.v;
       W[i]:= c.m.v; nO[i++]:=c.n.o.n.m.v;
       W[i]:= c.p.v; nO[i++]:=c.p.m.v;
        W[i]:=  c.m.v; nO[i++]:=c.v;
       W[i]:= c.n.m.v; nO[i++]:=c.n.v;
       W[i]:= c.p.m.v; nO[i++]:=c.p.v;   };
 write (4k, X, Y, Z, 4t, W, nO);
```

Subdivision in Processing: Corners

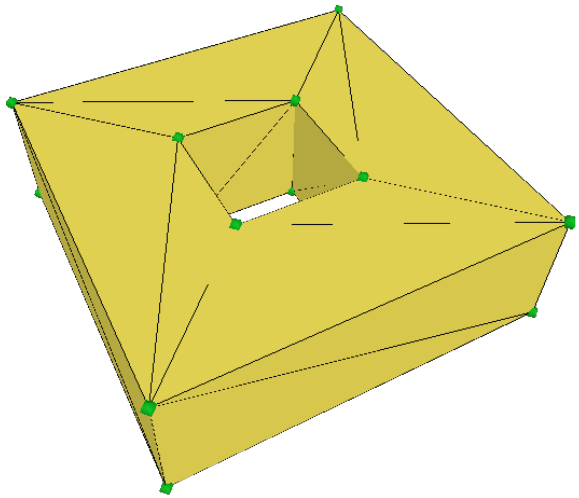
```
int t (int c) {int r=int(c/3); return(r);};
int n (int c) {int r=3*int(c/3)+(c+1)%3; return(r);};
int p (int c) {int r=3*int(c/3)+(c+2)%3; return(r);};
int v (int c) {return(V[c]);};
int o (int c) {return(O[c]);};
int l (int c) {return(o(n(c)));};
int r (int c) {return(o(p(c)));};
// indices to mid-edge vertices associated with corners
int w (int c) {return(W[c]);};
boolean border (int c) {return(O[c]==-1);};  // if has no opposite
// shortcut to get the point of the vertex v(c) of corner c
pt g (int c) {return(G[V[c]]);};
```

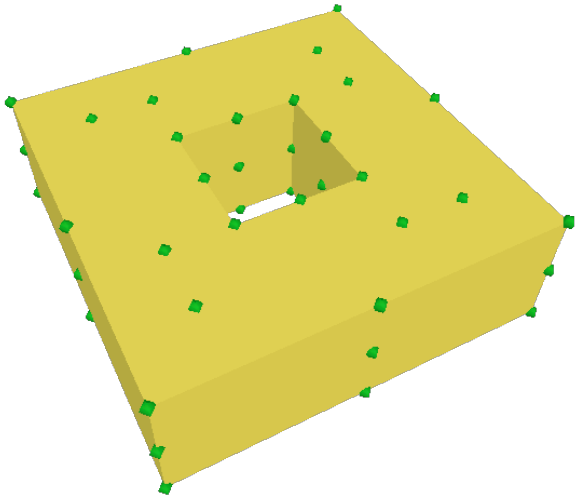
Subdivision in Processing: O table

```
// sets O table from V, assumes consistent orientation of triangles
void computeO() {
// init O table to -1: has no opposite (i.e. is a border corner)
for (int i=0; i<3*nt; i++) {O[i]=-1;};
// for each corner i, for each other corner j
for (int i=0; i<3*nt; i++) {  for (int j=i+1; j<3*nt; j++) {
     if( (v(n(i))==v(p(j))) && (v(p(i))==v(n(j))) ) {
          O[i]=j; O[j]=i;};};}; // make i and j opposite if they match
 };
```

Subdivision in Processing: Split edges

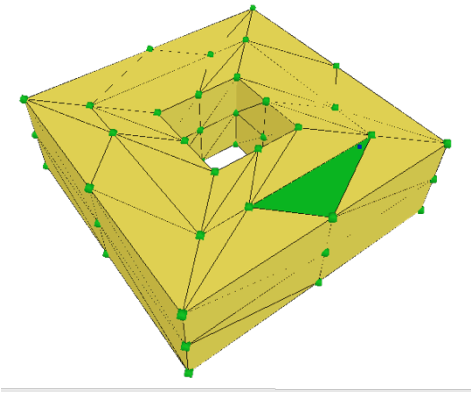
```
// creates a new vertex for each edge and stores its ID
// in the W of the corner (and of its opposite if any)
void splitEdges() {for (int i=0; i<3*nt; i++) {  // for each corner i
    if(border(i)) {G[nv]=midPt(g(n(i)),g(p(i))); W[i]=nv++;}
    else {if(i<o(i)) {G[nv]=midPt(g(n(i)),g(p(i))); W[o(i)]=nv;
     W[i]=nv++; }; };  // if this corner is the first to see the edge
    };
  };
```

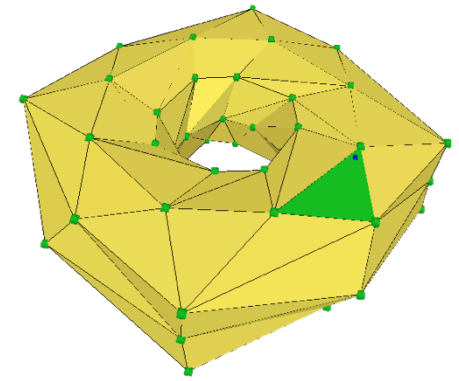




Subdivision in Processing: Bulge edges

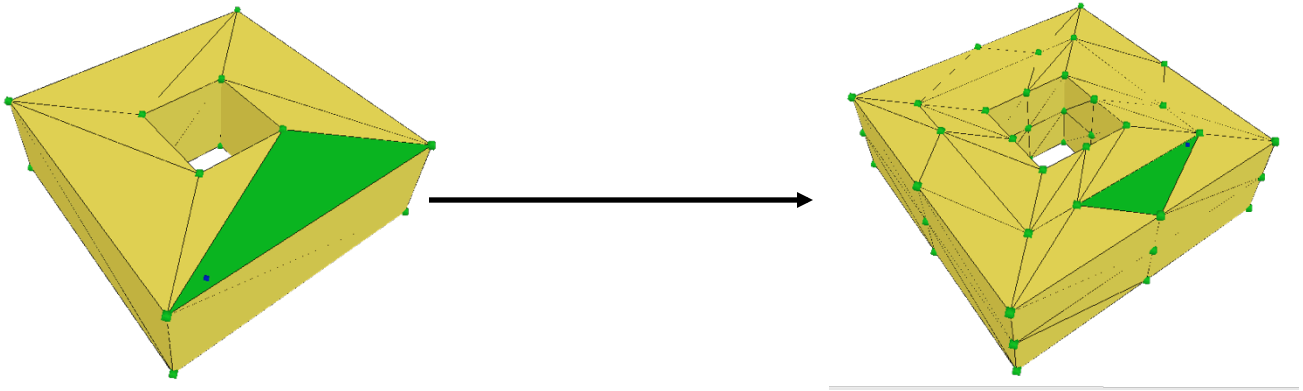
```
// tweaks new mid-edge vertices according to the Butterfly mask
    void bulge() {
  for (int i=0; i<3*nt; i++) {
 // no tweak for mid-vertices of border edges
   if((!border(i))&&(i<o(i))) {
G[W[i]].addScaledVec(0.25,
          midPt(midPt(g(l(i)),g(r(i))),
                midPt(g(l(o(i))),g(r(o(i))))
               ).vecTo(midPt(g(i),g(o(i)))));
     };
    };
  };
```





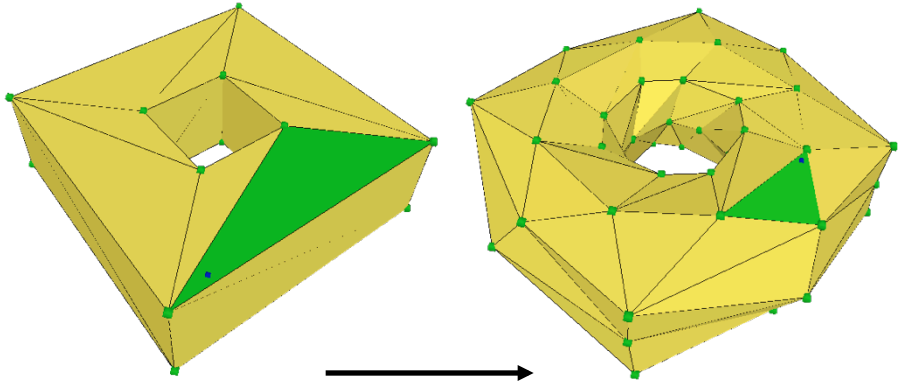
Subdivision in Processing: Split triangles

```
void splitTriangles() {    // splits each triangle into 4
  for (int i=0; i<3*nt; i=i+3) {
    V[3*nt+i]=v(i); V[n(3*nt+i)]=w(p(i)); V[p(3*nt+i)]=w(n(i));
    V[6*nt+i]=v(n(i)); V[n(6*nt+i)]=w(i); V[p(6*nt+i)]=w(p(i));
    V[9*nt+i]=v(p(i)); V[n(9*nt+i)]=w(n(i)); V[p(9*nt+i)]=w(i);
    V[i]=w(i); V[n(i)]=w(n(i)); V[p(i)]=w(p(i));
    };
  nt=4*nt;
  };
```

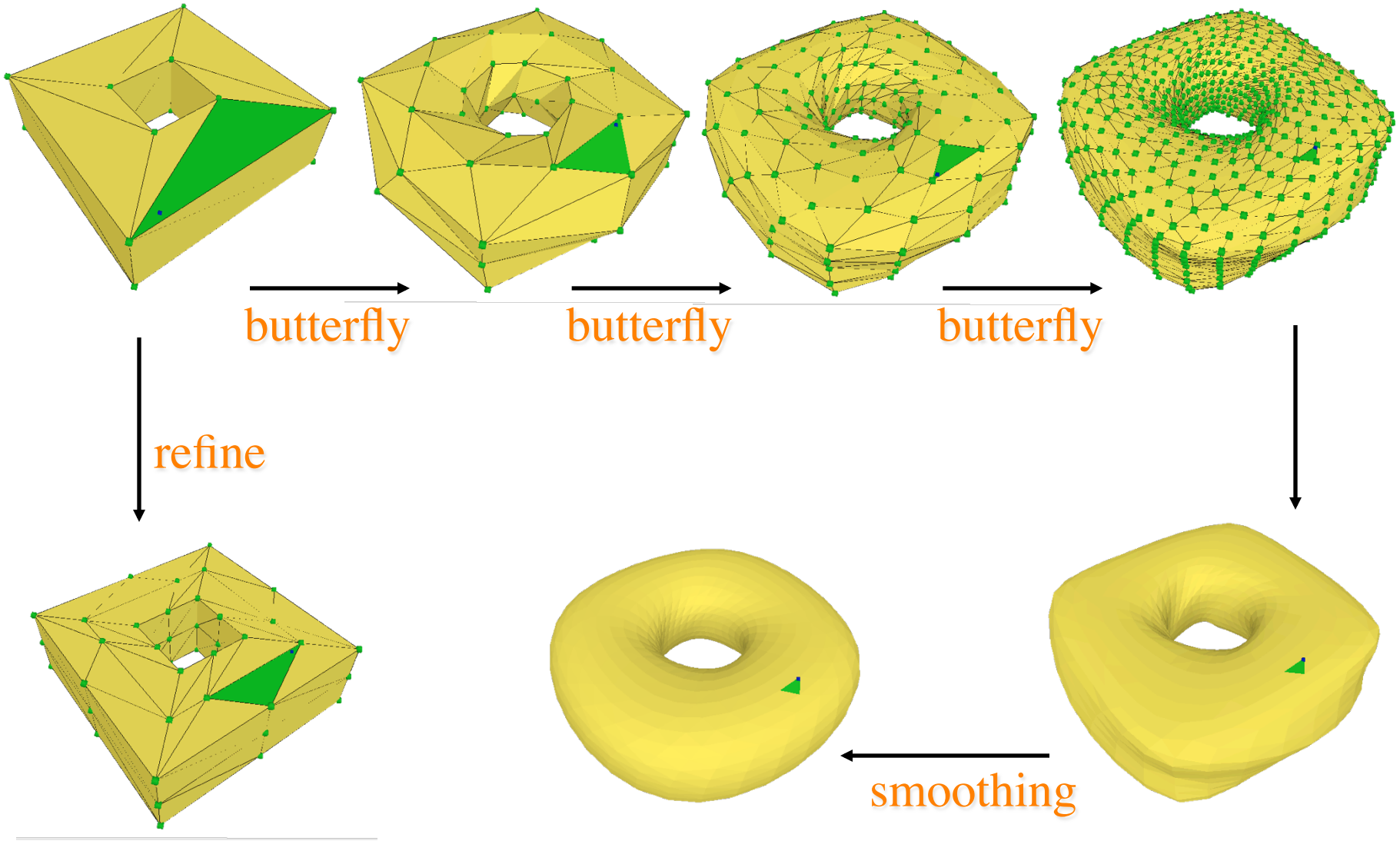


Subdivision in Processing: Butterfly

```
void keyPressed() {
    …
 if (key=='B') {
    splitEdges();
    bulge();
    splitTriangles();
    computeO();
    };
```



Butterfly subdivision & smoothing



Smoothing in Processing: Valences

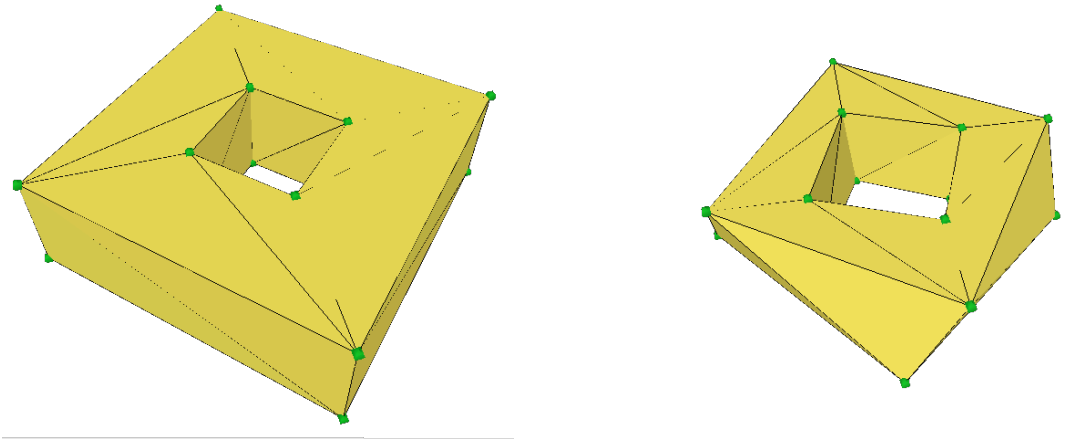
```
void computeValence() {      // caches valence of each vertex
  // resets the valences to 0
for (int i=0; i<nv; i++) {Nv[i].setTo(0,0,0); valence[i]=0;};
for (int i=0; i<3*nt; i++) {valence[v(i)]++; };
  };
```

Smoothing in Processing: Normals

```
// computes the vertex normals as sums of the normal vectors
// of incident tirangles scaled by area/2
void computeVertexNormals() {
  computeValence();
  for (int i=0; i<3*nt; i++)
    {Nv[v(p(i))].add(g(p(i)).vecTo(g(n(i))));};
  for (int i=0; i<nv; i++) {Nv[i].div(valence[i]);};
  };
```

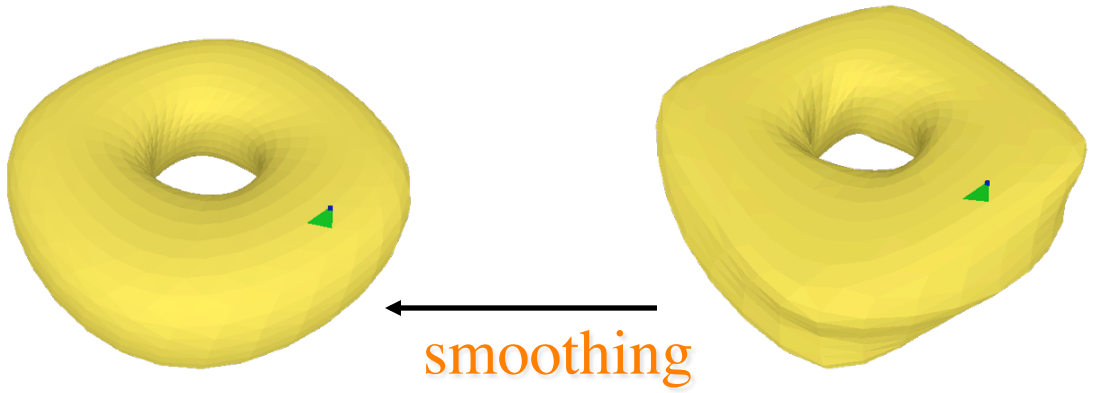
Smoothing in Processing: Tuck

```
// displaces each vertex by a fraction s of its normal
void tuck(float s) {
   for (int i=0; i<nv; i++) {G[i].addScaledVec(s,Nv[i]);}; };
```



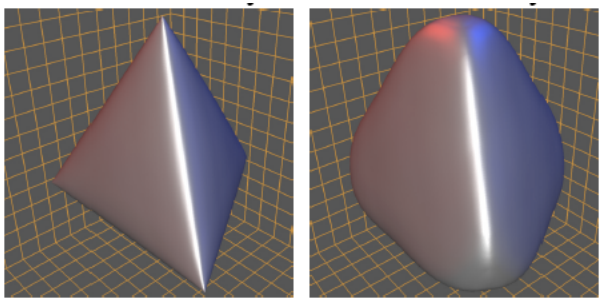
Smoothing in Processing: Smooth

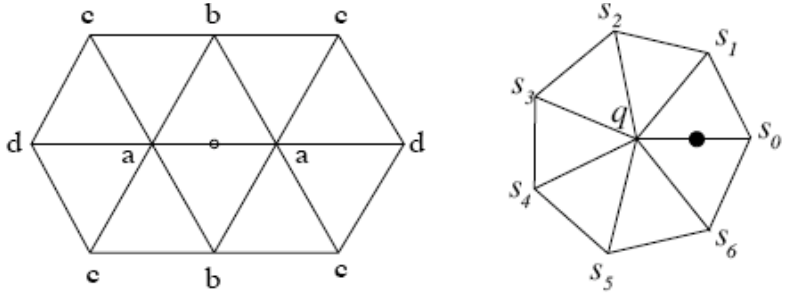
```
void keyPressed() {
  …
  // bi-laplace smoothing
  if (key=='S') {
    computeVertexNormals();
    tuck(0.6);
    computeVertexNormals();
    tuck(-0.6);
    };
```



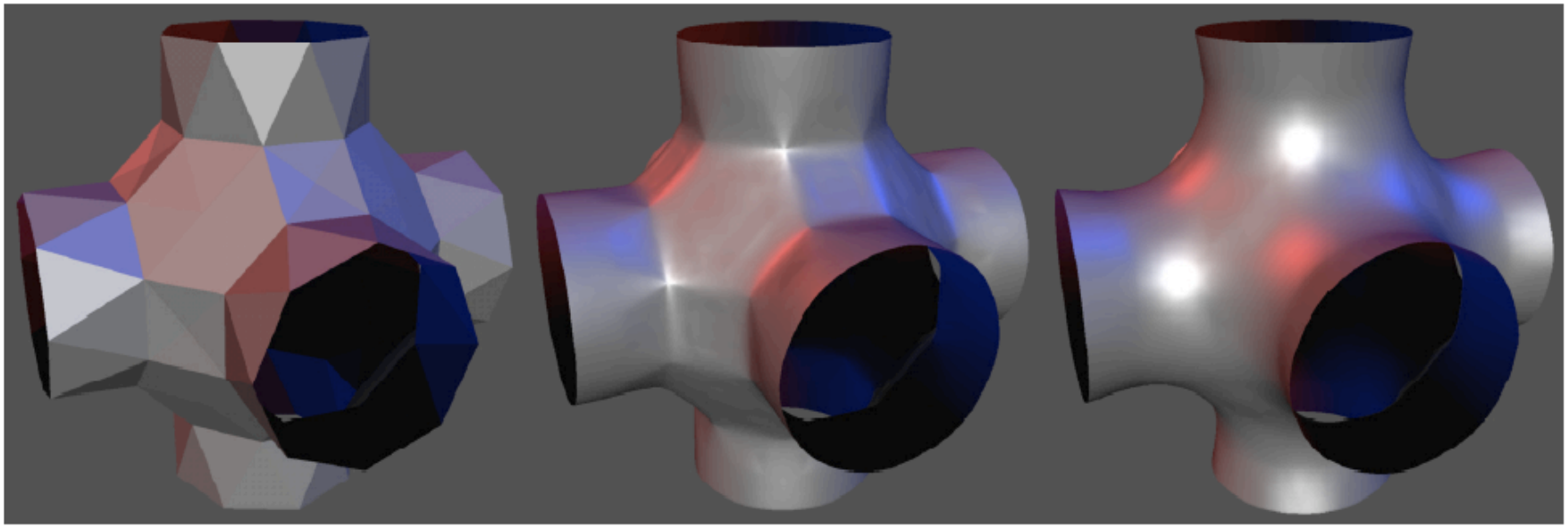
Modified Butterfly scheme

"**Interpolating** Subdivision for Meshes with Arbitrary Topology",
Zorin, Schröder, Sweldens





$$s_j = (1/4 + \cos(2\pi j/K) + 1/2\cos(4\pi j/K))/K$$



Other subdivision schemes

- Loop subdivision
  - Move mid-edge points 1/4 towards the average of their second neighbors (s)
  - Move the original vertices towards the average of their new neighbors.

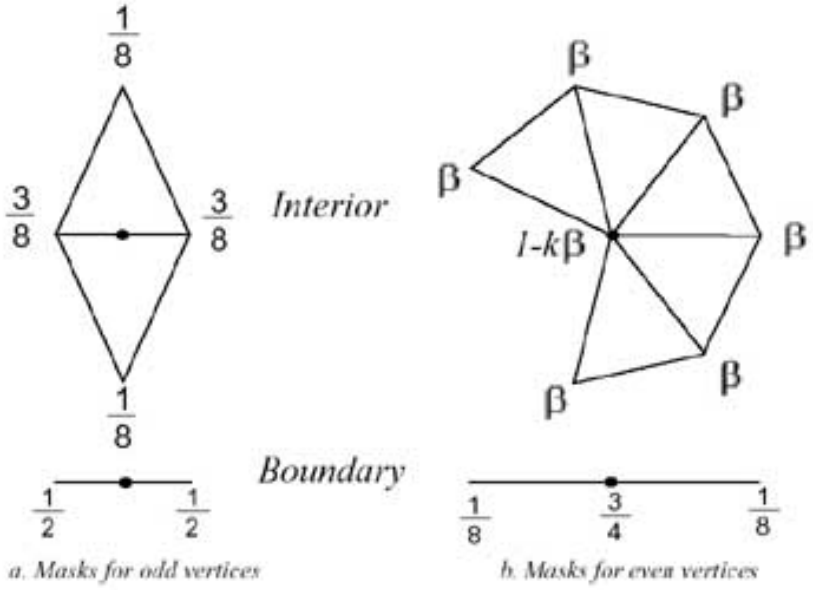


Figure 1: Masks for Loop scheme

Loop suggests to use

$\beta=(5/8-(3/8+(\cos(2\Pi/k))/4))/k$

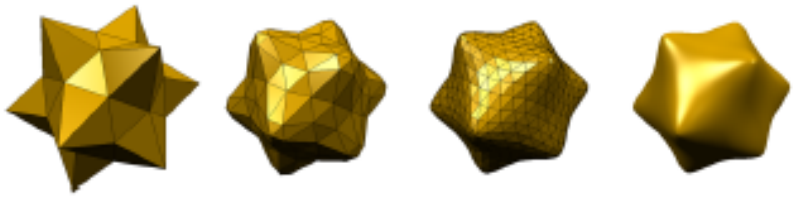
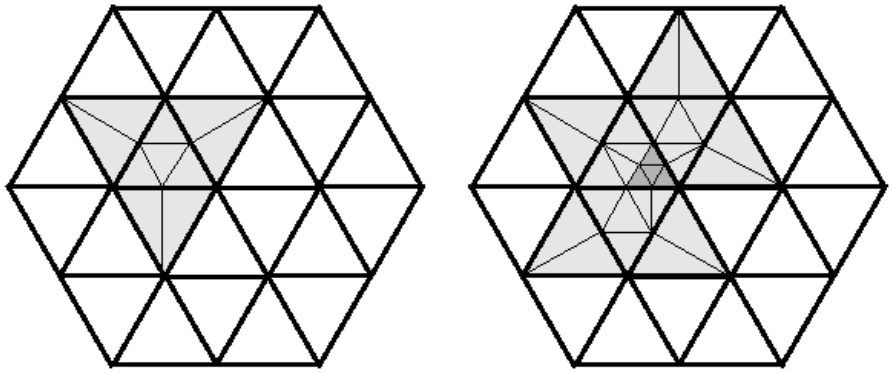
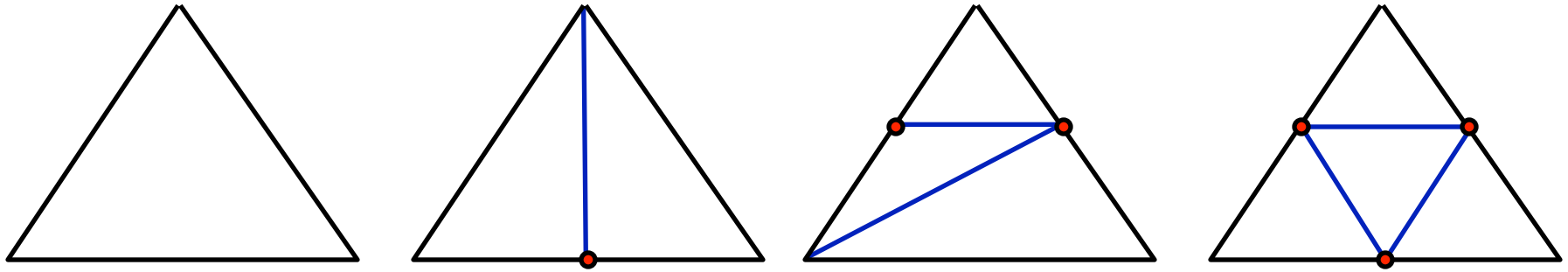


Figure 1: Loop subdivision surface.

Adaptive subdivision

- What if you need to subdivide more in one areas?
  - "A Sub-Atomic Subdivision Approach" by S. Seeger, K. Hormann G. Hausler and G. Greiner





Subdivision of non-triangle meshes

- Insert mid-edge points. Remove old vertices

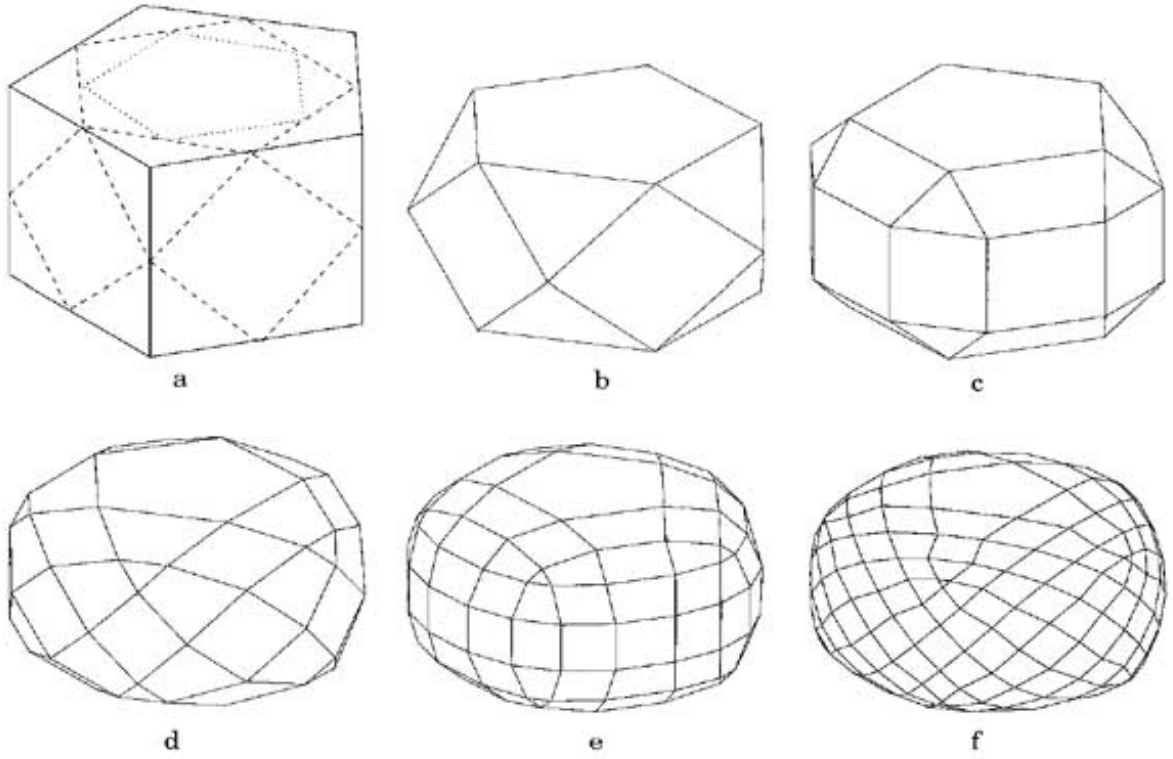
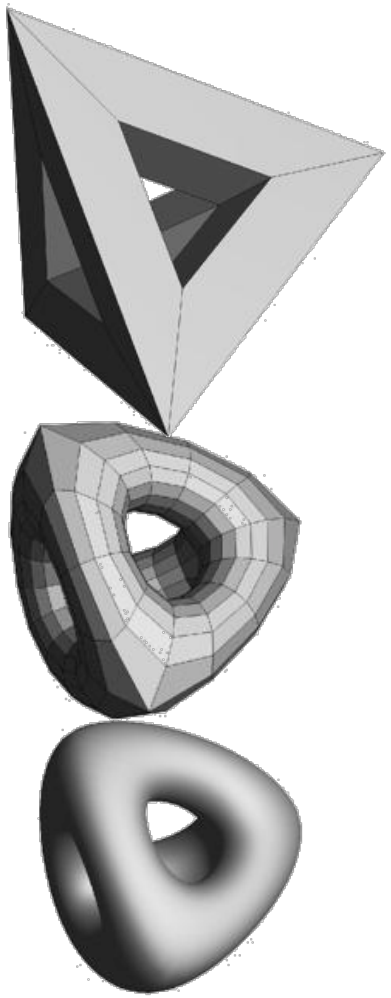


Fig. 1. Midpoint subdivision at work.



J. Peters, U. Reif, The Simplest Subdivision Scheme for smoothing polyhedra. 16(4), ToG 97.

What you must remember

- That regular subdivision multiplies the number of triangles and vertices by 4
- That the butterfly subdivision computes edge-midpoints using the averages of first, second, and third neighbors (f, s, t):
  - $m(c) = \mathbf{average}(f) + (\mathbf{average}(s) - \mathbf{average}(t))/4$
- Our high-level approach algorithm for computing the XYZ' and V' tables for one level of subdivision.
- The fact that the butterfly subdivision interpolates the original vertices
- How to implement an adaptive Butterfly subdivision
- How to perform a Loop subdivision

Examples of questions for exams

- How is the midpoint of edge E computed in the butterfly subdivision scheme given a corner c at the opposite tip of one of the two triangles incident upon E?
- Is the Butterfly subdivision interpolating the original vertices? Justify/prove your answer.
- How many vertices are introduced by three levels of butterfly subdivision to a zero-genus mesh with V vertices?
- What is the influence of one of the vertices of the original triangle mesh on the surface resulting from repeatedly applying the regular Butterfly subdivision? (Hint, each one of the original triangles is subdivided into 4, then 16, and so on. Say that repeated subdivision maps a triangle into a curved patch. Obviously, each vertex of the original mesh influences the patches of the triangles incident upon it. Does it influence more?

A Sketch-Based Interface for Detail-Preserving Mesh Editing

Andrew Nealen
Olga Sorkine
Marc Alexa
Daniel Cohen-Or

TECHNISCHE UNIVERSITÄT DARMSTADT
Discrete Geometric Modeling

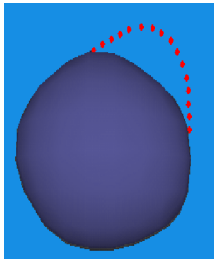
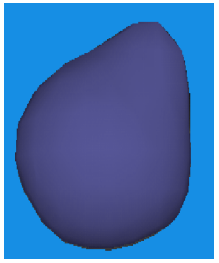
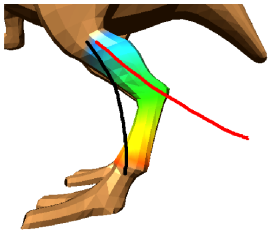
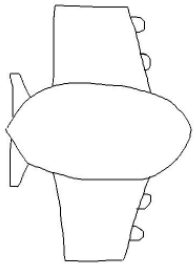
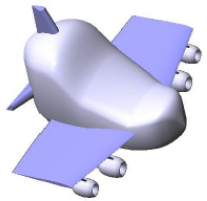
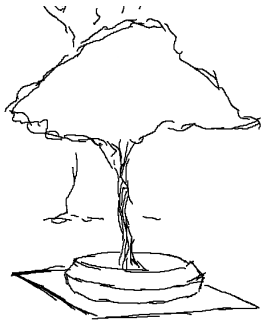
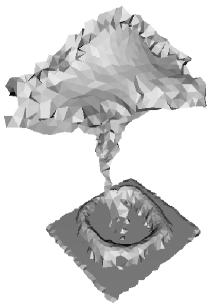
TEL AVIV UNIVERSITY

SIGGRAPH2005

Sketch-Based Interfaces and Modeling

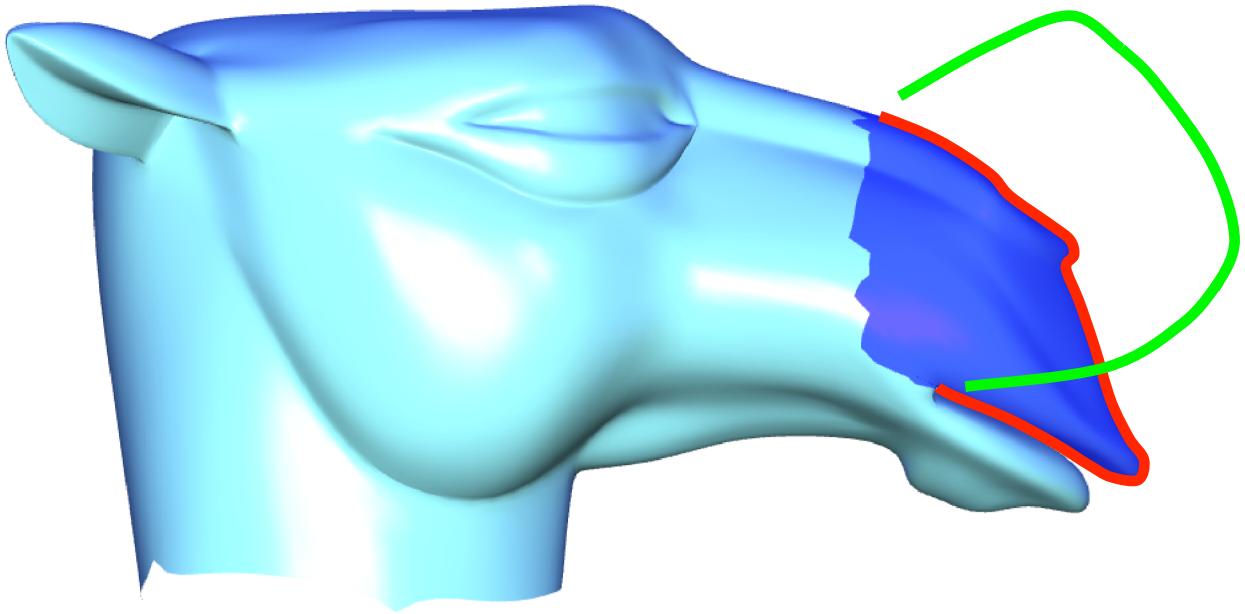
- (Some) previous work
  - SKETCH **[Zeleznik et al. 96]**
  - Teddy **[Igarashi et al. 99 and 03]**
  - Variational implicits **[Karpenko et al. 02]**
  - Relief **[Bourguignon et al. 04]**
  - Sketching mesh deformations **[Kho and Garland 05]**
  - Parametrized objects **[Yang et al. 05]**
  - Many, many more... and (hopefully) more to come!



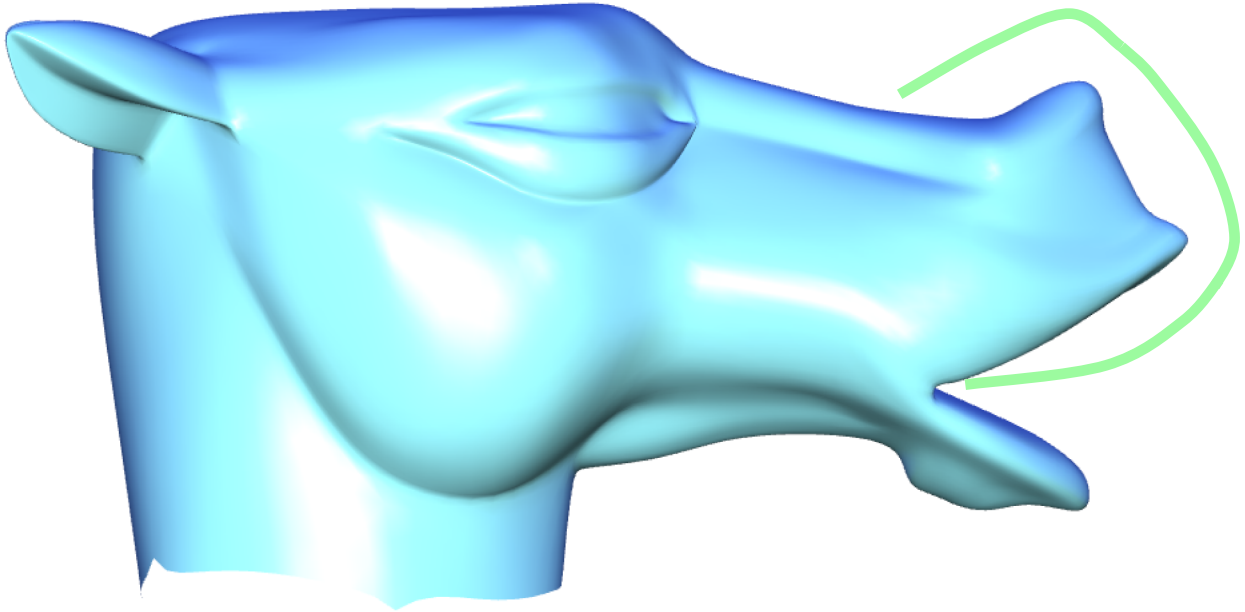

Silhouette Sketching

- Approximate sketching
  - Balance weighting between detail and positional constraints



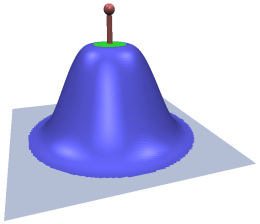
Silhouette Sketching

- Approximate sketching
  - Balance weighting between detail and positional constraints

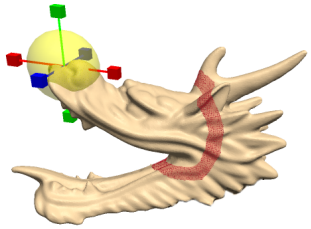


Inspiration and Motivation

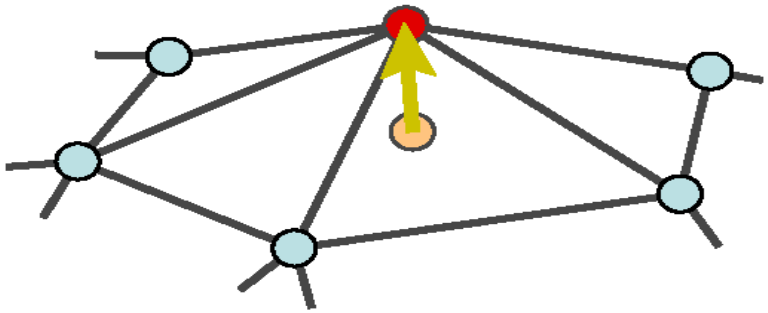
- The (affine) handle metaphor
  - Used in (almost) every editing tool
  - Nice, but can be unintuitive for specific editing tasks
- Laplacian Mesh Editing
  - Preserve local detail **after** imposing editing constraints



[Botsch and Kobbelt 04]



[Sorkine et al. 04]

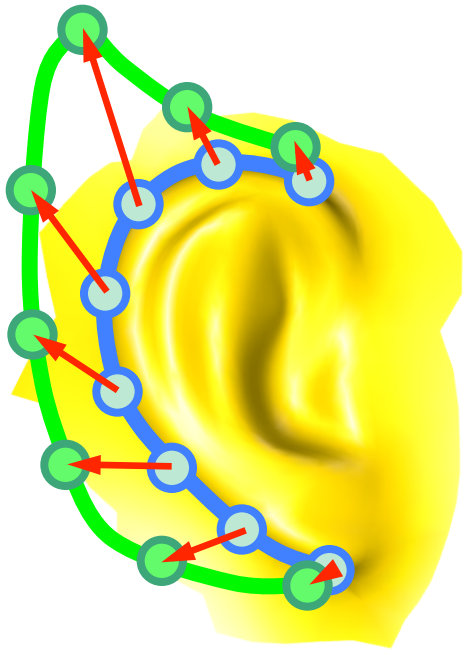


$$\delta_i = \frac{1}{d_i} \sum_{j \in N(i)} (\mathbf{v}_i - \mathbf{v}_j)$$

[Sorkine et al. 04] [Zhou et al. 05]

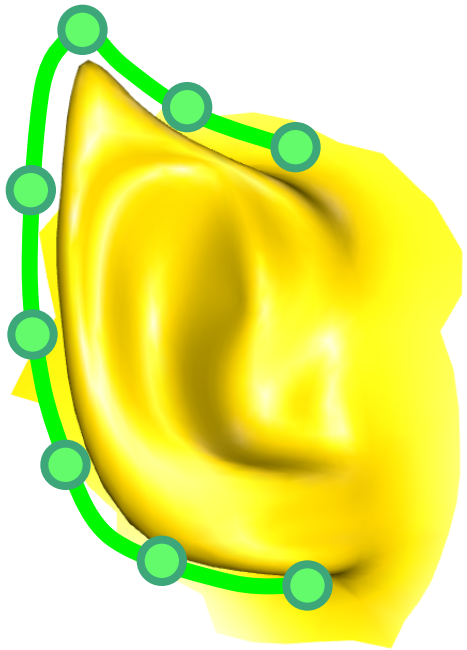
Silhouette Sketching

- Using silhouettes as handles
  - Detect object space silhouette
  - Project to screen space and parametrize [0,1]
  - Parametrize sketch [0,1]
  - Find correspondences



Silhouette Sketching

- Using silhouettes as handles
  - Detect object space silhouette
  - Project to screen space and parametrize [0,1]
  - Parametrize sketch [0,1]
  - Find correspondences
  - Use as positional constraints while retaining depth value



Silhouette Sketching

- What is a good silhouette?