Fondamenti di Grafica Tridimensionale

Paolo Cignoni
p.cignoni@isti.cnr.it
http://vcg.isti.cnr.it/~cignoni

Mesh Simplification

- Rendering somewhat roughly linearly dependent with primitive number.
- How to reduce the complexity of a mesh in term of primitive numbers without sacrificing (too much) the quality

Simplification Algorithms

- Simplification approaches:
  - incremental methods based on local updates
    - mesh decimation [Schroeder et al. 92]
    - energy function optimization [Hoppe et al. 93,96,97]
    - quadric error metrics [Garland et al. '97]
  - coplanar facets merging
    - [Hinker et al. `93, Kalvin et al. `96]
  - Re-tiling
    - [Turk `92]
  - Clustering
    - [Rossignac et al. `93, ... + others]
  - Wavelet-based
    - [Eck et al. `95, + others]

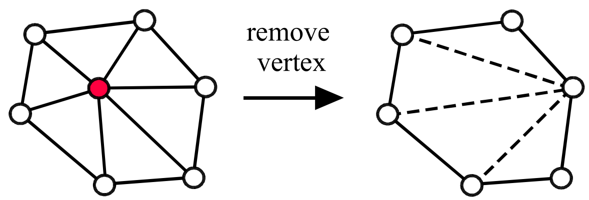
Incremental methods based on ***local updates***

- All of the methods such that :
  - simplification proceeds as a sequence of ***local updates***
  - each update ***reduces mesh size*** and [monotonically] ***decreases*** the ***approximation precision***
- Different approaches:
  - **mesh decimation**
  - **energy function optimization**
  - **quadric error metrics**

- Local update actions:

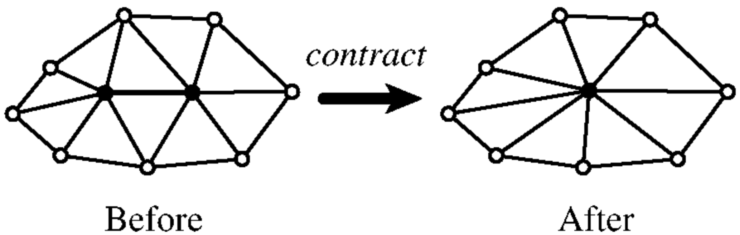
No. Faces

  - **vertex removal**



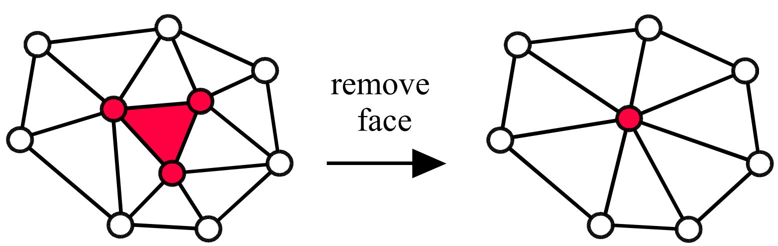
n-2

  - **edge collapse**
    - **preserve location**
    - **new location**



n-2

  - **triangle collapse**
    - **preserve location**
    - **new location**



n-4

The common framework:

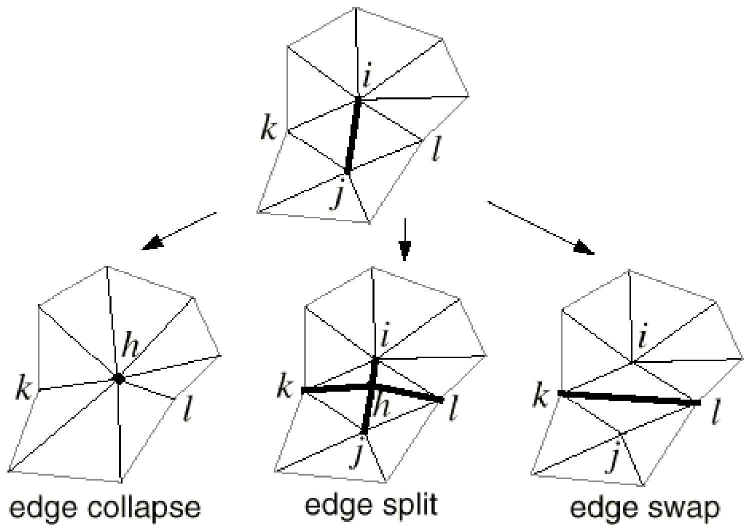
- **loop**
  - ***select*** the element to be deleted/collapsed;
  - ***evaluate approximation*** introduced;
  - ***update*** the mesh after deletion/collapse;

until mesh **size/precision** is satisfactory;

Energy function optimization

Mesh Optimization **[Hoppe et al. `93]**

- Simplification based on the iterative execution of :
  - edge collapsing
  - edge split
  - edge swap



- approximation quality evalued with an **energy function** :

$$E(M) = E_{dist}(M) + E_{rep}(M) + E_{spring}(M)$$

which evaluates geometric **fitness** and repr. **compactness**

$\mathbf{E_{dist}}$: sum of squared distances of the original points from M

$\mathbf{E_{rep}}$: factor proportional to the no. of vertex in M

$\mathbf{E_{spring}}$: sum of the edge lenghts

... Energy function optimization: Mesh Optimization ...

Algorithm structure

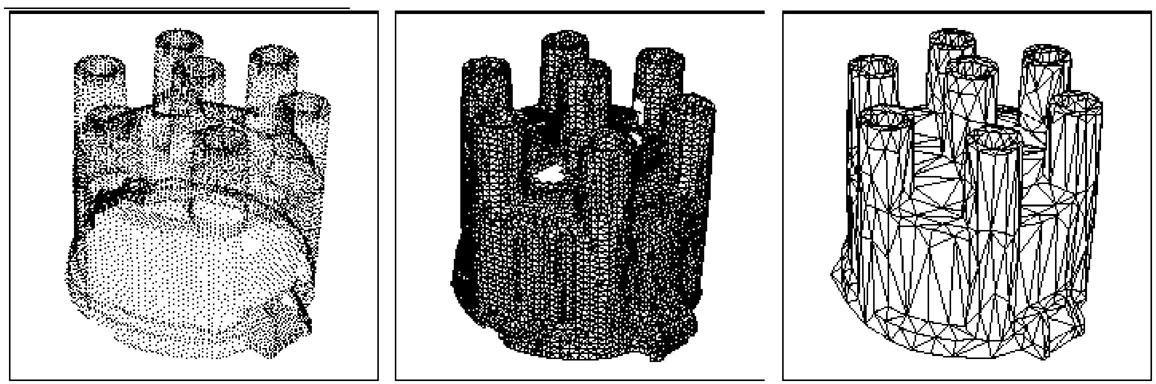
- outer minimization cicle (***discrete*** optimiz. probl.)
  - choose a legal action (edge collapse, swap, split) which reduces the energy function
  - perform the action and update the mesh (M_i -> M_{i+1})

- inner minimization cicle (***continuous*** optimiz. probl.)
  - optimize the vertex positions of M_{i+1} with respect to the initial mesh M_0

but (to reduce complexity)

- legal action selection is random
- inner minimization is solved in a fixed number of iterations

Mesh Optimization - *Examples*



(j) Laser range data ($n = 12,745$) (k) Output of phase one (l) Output of phase two

[Image by Hoppe et al.]

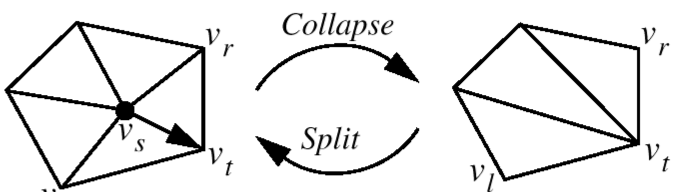
Mesh Optimization - *Evaluation*

- high quality of the results
- preserves topology, re-sample vertices
- high processing times
- not easy to implement
- not easy to use (selection of tuning parameters)
- adopts a global error evaluation, but the resulting approximation is not bounded

... Energy function optimization: **Progressive Meshes** ...

Progressive Meshes
[Hoppe `96]

- execute **edge collapsing *only*** to reduce the *energy function*
- *edge collapsing* can be easily inverted ==> store sequence of inverse *vertex split* trasformations to support:
  - multiresolution
  - progressive transmission
  - selective refinements
  - geomorphs



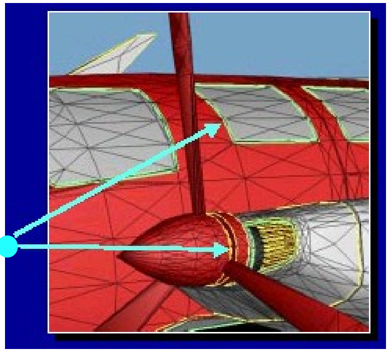
a) An edge collapse and split

- *faster* than MeshOptim.

... Energy function optimization: **Progressive Meshes** ...

Preserving mesh ***appearance***

- shape and crease edges
- scalar fields discontinuities (e.g. color, normals)
- discontinuity curves

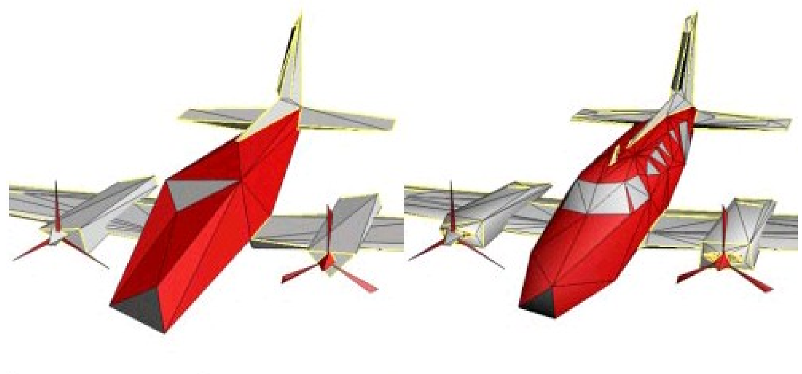


[image by H. Hoppe]

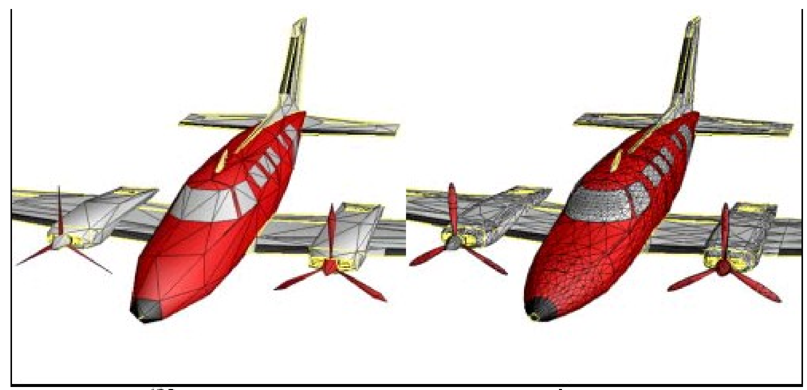
Managed by inserting two new components in the *energy function*:

- E_{scalar}: measures the accuracy of scalar attributes
- E_{disc}: measure the geometric accuracy of discontinuity curves

Progressive Meshes
Examples



(a) Base mesh M^0 (150 faces)   (b) Mesh M^{175} (500 faces)



(c) Mesh M^{425} (1,000 faces)   (d) Original $\hat{M} = M^n$ (13,546 faces)

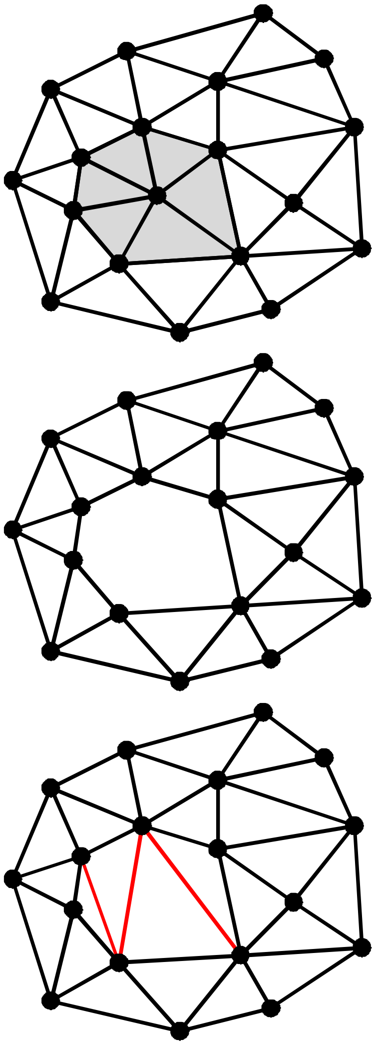
Progressive Meshes - *Evaluation*

- high quality of the results
- preserves topology, re-sample vertices
- not easy to implement
- not easy to use (selection of tuning parameters)
- adopts a global error evaluation, not-bounded approximation
- preserves vect/scalar attributes (e.g. color) **discontinuities**
- supports **multiresolution** output, geometric morphing, **progressive transmission, selective** refinements
- much **faster** than MeshOpt.

An implementation is present as parto of DirectX 6.0 tools

Mesh Decimation

[Schroeder et al'92]



- Based on controlled removal of ***vertices***
- Classify vertices as ***removable*** or ***not*** (based on local topology / geometry and required precision)

Loop

- choose a *removable* vertex $\boldsymbol{v_i}$
- delete $\boldsymbol{v_i}$ and the incident faces
- re-triangulate the hole

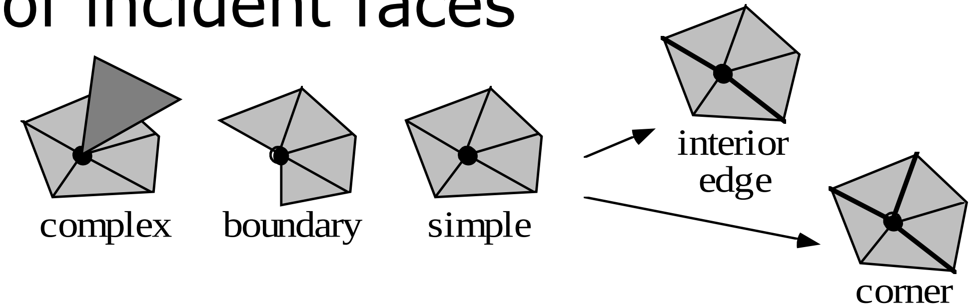
until

no more removable vertex **or** reduction rate fulfilled

- General method (manifold/non-manifold *input*)
- Algorithm phases:
  - topologic classification of vertices
  - evaluation of the decimation criterion (error evaluation)
  - re-triangulation of the removed triangles patch

Topologic classification of vertices

- for each vertex: find and characterize the loop of incident faces



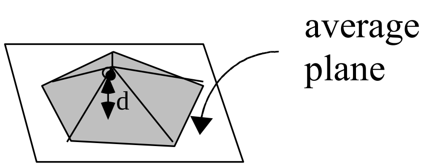
- *interior edge*: if dihedral angle between faces $< k_{angle}$
  (k_{angle} : user driven parameter)

- *not-removable vertices*: complex, [corner]

Decimation criterion -- a vertex is *removable* if:

- **simple** vertex:

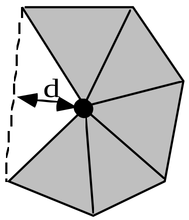
  if distance ***vertex - face loop average plane*** is lower than n_{max}



d: distance to plane

- **boundary / interior / corner** vertices:

  if distance ***vertex - new boundary/interior edge*** is lower than n_{max}

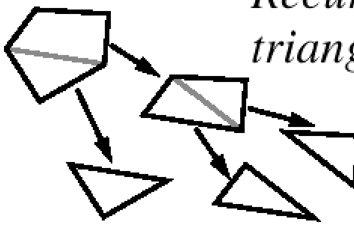


d: distance to edge

- adopts *local evaluation* of the approximation!!
- n_{max} : value selected by the user

Re-triangulation

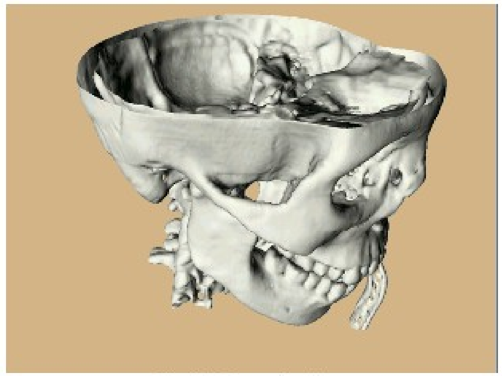
- face loops in general non planar ! (but star-shaped)
- adopts ***recursive loop splitting*** re-triangulation



control *aspect ratio* to ensure simplified mesh quality

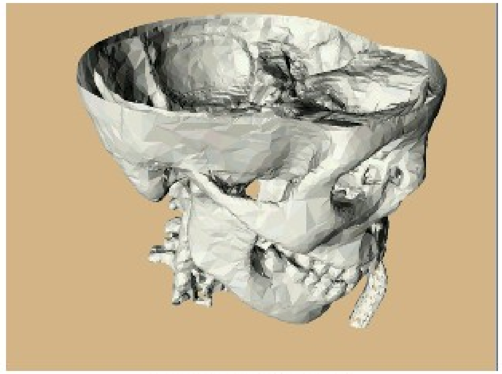
- for each vertex removed:
  - *if* simple or boundary vertex ==> 1 loop
  - *if* interior edge vertex ==> 2 loops
  - *if* boundary vertex ==> - 1 face
  - *otherwise* ==> - 2 faces

Decimation - *Examples*



Full Resolution
(569K Gouraud shaded triangles)

75% decimated
(142K Gouraud shaded triangles)



75% decimated
(142K flat shaded triangles)

90% decimated
(57K flat shaded triangles)

(images by W. Lorensen)

Original Mesh Decimation - *Evaluation*

- good efficiency (speed & reduction rate)
- simple implementation and use
- good approximation
- preserves topology; vertices are a subset of the original ones
- error is ***not*** bounded (local evaluation ==> accumulation of error!!)

Approximation Error Evaluation

Classification of simplification methods based on ***approximation error*** evaluation euristics:

User' viewpoint:
- simple to grasp
- simple to drive

- **locally-bounded** error, based on mesh distances
  [ex. standard Mesh Decimation]
- **globally bounded** error, based on mesh distances
  [ex. Envelopes + enhanced Decimation + others]

→ *very handy*

- control based on **mesh characteristics**
  [ex. vertex proximity, mesh curvature]

→ *may be misleading*

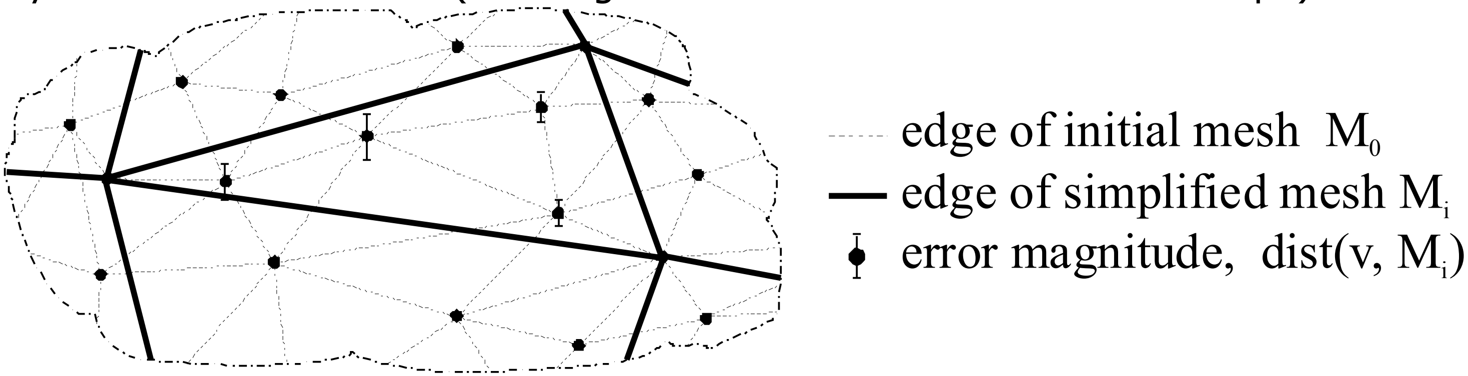
- **energy function** evaluation
  [ex. Mesh Optim. , Progr. Meshes]

→ *not easy, many parameters to be selected*

Heuristics proposed for ***global error evaluation***:

- ***accumulation*** *of local errors*
  [Ciampalini97]
  fast, **but** approximate

- ***vertex--to--simplified mesh*** **distance**
  [Soucy96]
  requires storing which of the original vertices maps to each simplified face;
  very near to exact value (but large under-estimation in the first steps)



... Heuristics proposed for ***global error evaluation***:

- ***input mesh -- to -- simplified mesh <u>edges</u>*** **distance**
  [Ciampalini97]
  - for each internal edge:
    - select sampling points $\mathbf{p_i}$ (regularly/random)
    - evaluate distance $d(M_0, \mathbf{p_i})$

  sufficiently precise and efficient in time

- ***input mesh -- to -- simplified mesh*** **distance**
  [Klein96]

  precise, **but** more complex in time

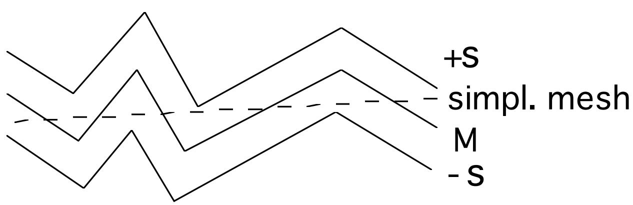
- **use *envelopes*** [Cohen et al.'96]

  precise, no self-intersections **but** complex in time and to be implemented

Simplification Envelopes [Cohen et al.'96]

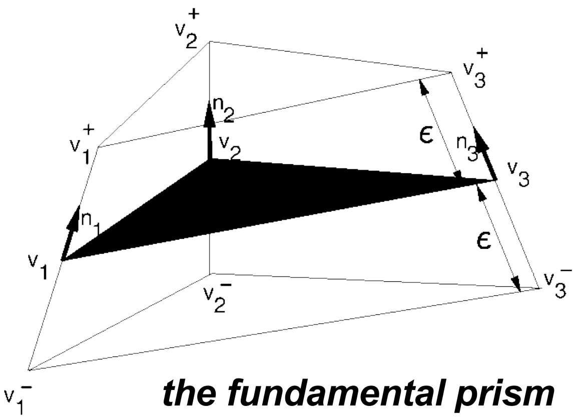
- given the input mesh ***M***
  - build two envelope meshes $\boldsymbol{M_-}$ and $\boldsymbol{M_+}$ at distance $\boldsymbol{-\varepsilon}$ and $\boldsymbol{+\varepsilon}$ from ***M*** ;
  - simplify M (following a decimation approach) by enforcing the decimation criterion:

    a candidate vertex may be removed **only if** the new triangle patch does not intersect neither $\boldsymbol{M_-}$ or $\boldsymbol{M_+}$



- by construction, envelopes do not self-intersect

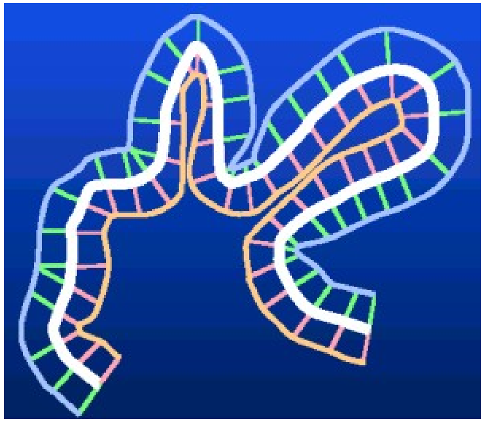
  ==> simplified mesh is **not self-intersecting** !!



the fundamental prism

- distance between envelopes becomes smaller near the bending sections, and simplification harder

- **border tubes** are used to manage open boundaries



(drawing by A. Varshney)

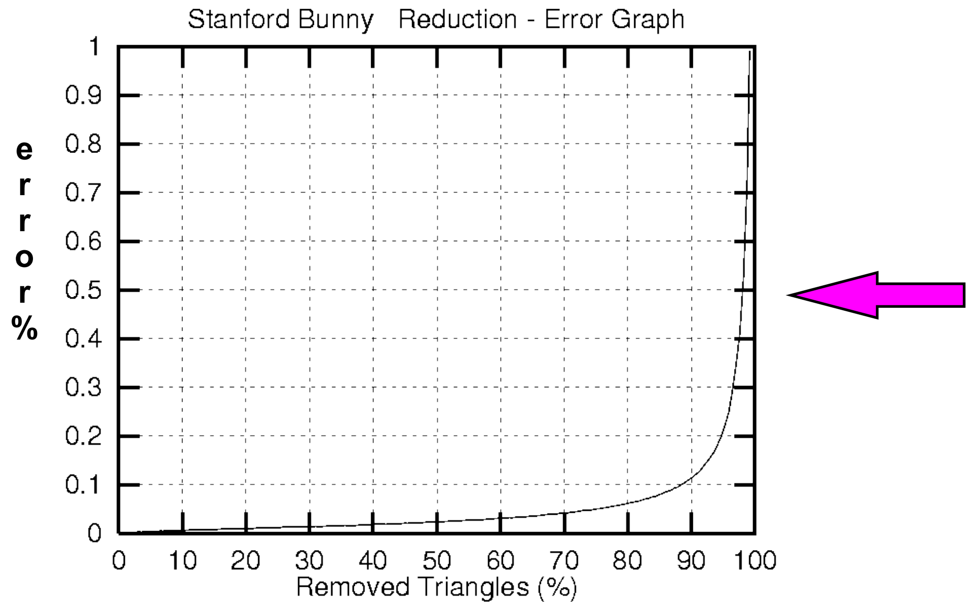
Simplification Envelopes - *Evaluation*

- works on manifold surface **only**
- bounded approximation
- construction of envelopes and intersection tests are not cheap
- **>** three times more RAM (input mesh + envelopes + border tubes)
- preserve topology, vertices are a subset of the original, prevents self-intersection

available in public domain

Results

- Simplification times ~= linear with mesh size



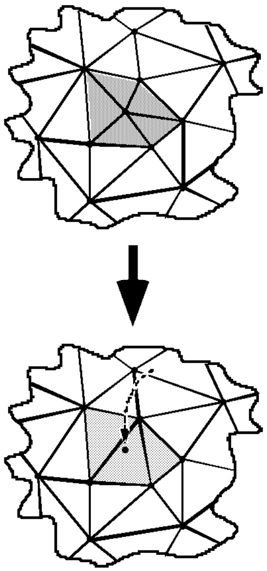
no staircase abrupt error increase
(fundamental for the quality of the multiresolution output)

Construction of a multiresolution model

Keep the ***history*** of the simplification process :

- when we remove a vertex we have **dead** and **newborn** triangles

- assign to each triangle *t* a ***birth error*** t_b and a ***death error*** t_d equal to the error of the simplified mesh just before the removal of the vertex that caused the birth/death of *t*



By storing the ***simplification history*** (faces+errors) we can

simply extract ***any approximation level*** in real time

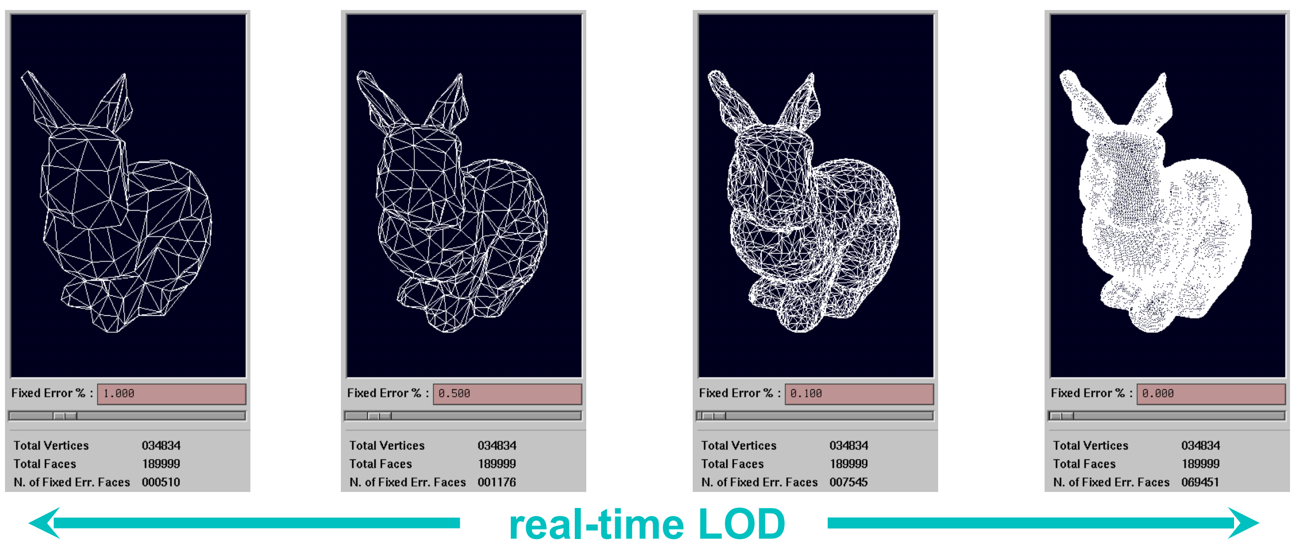
Real-time resolution management

- by extracting from the ***history*** all the triangles t_i with

  t_b <= □< t_d

  we obtain a model $\mathbf{M}_{\varepsilon}$ which satisfies the approximation error ε

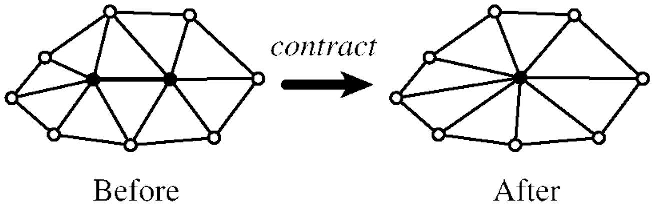
- mantaining the whole *history* data structure costs approximately 2.5x - 3x the full resolution model



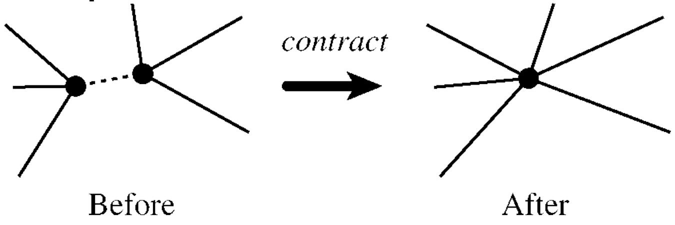
Simplification using Quadric Error Metrics

[Garland et al. Sig'97]

- Based on incremental **edge-collapsing**



- **but** can also collapse vertex couples which are **not connected** (topology is not preserved)



Geometric error approximation is managed by simplifying an approach based on **plane set distance** **[Ronfard,Rossignac96]**

- INIT: store for each vertex the set of incident planes
- Vertex_Collapsing $(v_1, v_2) \Rightarrow v_{new}$
  - plane_set (v_{new}) = union of the two **plane sets** of v_1, v_2
  - collapse only if v_{new} is not “farther” from its plane set than the selected target error **ε**

criticism:

- storing plane sets and computing distances is not cheap !

Quadric Error Metrics solution:

- quadratic distances to planes represented with **matrices**
  - plane sets merge *via* matrix sums
  - very efficient evaluation of error *via* **matrix operations**

  but
  - triangle size is taken into account only in an approximate manner (orientation only in Quadrics + weights)

Algorithm structure:

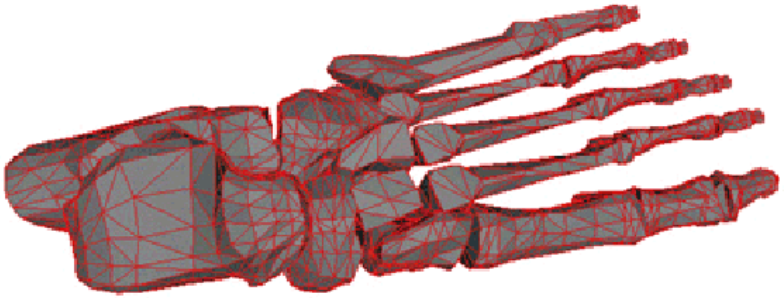
- select valid vertex pairs (upon their distance),
  insert them in an heap sorted upon minimum cost;
- **repeat**
  - extract a valid pair v_1, v_2 from heap and contract into v_{new};
  - re-compute the cost for all pairs which contain v_1 or v_2 and update the heap;

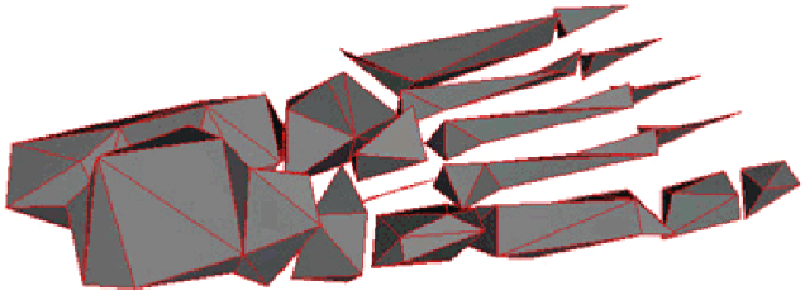
until sufficient reduction/approximation **or** heap empty

An example

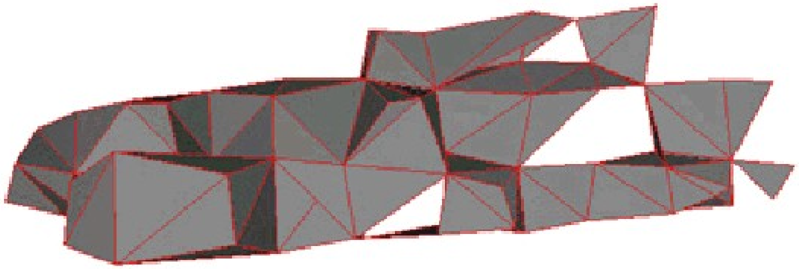
- **Original.** Bones of a human's left foot (4,204 faces).
- Note the many separate bone segments.

- **Edge Contractions.** 250 face approximation.
- Bone seg-ments at the ends of the toes have disappeared; the toes appear to be receding back into the foot.





„ **Clustering.** 262 face approximation.



[Images by Garland and Heckbert]

Quadric Error for Surfaces

- Let $\mathbf{n}^{\mathrm{T}}\mathbf{v} + d = 0$ be the equation representing a plane
- The squared distance of a point $\boldsymbol{x}$ from the plane is

$$D(\mathbf{x}) = \mathbf{x}(\mathbf{n}\mathbf{n}^{\mathrm{T}})\mathbf{x} + 2d\mathbf{n}^{\mathrm{T}}\mathbf{x} + d^2$$

- This distance can be represented as a quadric

$$Q = (\mathrm{A}, \mathbf{b}, c) = (\mathbf{n}\mathbf{n}^{\mathrm{T}}, d\mathbf{n}, d^2)$$

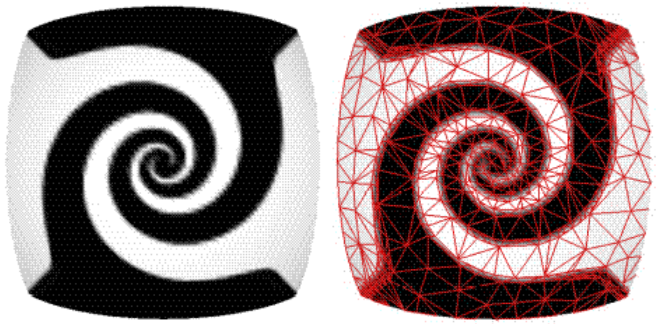
$$Q(\mathbf{x}) = \mathbf{x}\mathrm{A}\mathbf{x} + 2\mathbf{b}^{\mathrm{T}}\mathbf{x} + c$$

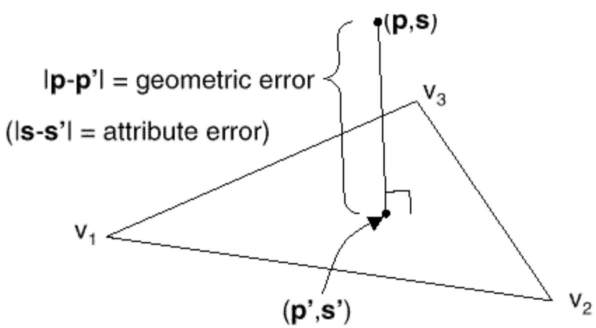
Quadric

- The boundary error is estimated by providing for each boundary vertex *v* a quadric Q_v representing the sum of the all the squared distances from the faces incident in *v*
  - The error of collapsing an edge e=(v,w) can be evaluated as $Q_w(v)$.
  - After the collapse the quadric of *v* is updated as follow $Q_v = Q_v + Q_w$

Domain Error

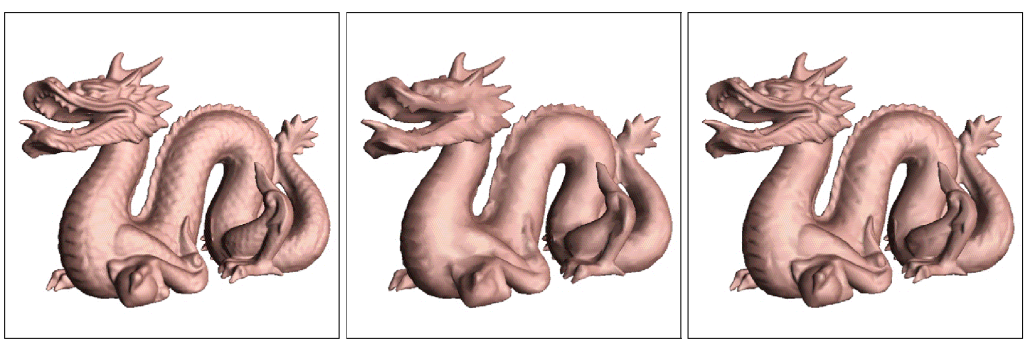
- The two dataset D and D′ span different domains n, n′
- Same problem of classical surface simplification
- Measure the Hausdorff distance between the boundary surfaces of the two datasets D and D′
  - $e^a_f(D, D') = \max_{x \in n} (\min_{y \in n'}(dist(x,y)))$
  - $e_f(D, D') = \max(e^a_f(D, D'), e^a_f(D', D))$
- Various techniques to approximate this distance between two surfaces [Ciampalini et al. 97]





Quadric can be exteneded to take into account:

- color and texture attributes error are computed by projecting them in R^{3+m} [Garland 98]
- by computing attribute error as the squared deviation between original value and the value interpolated [Hoppe 99]



(a) Original mesh (b) Q is just geometric error (c) Q also includes normals

Quadric Error Metrics -- *Evaluation*

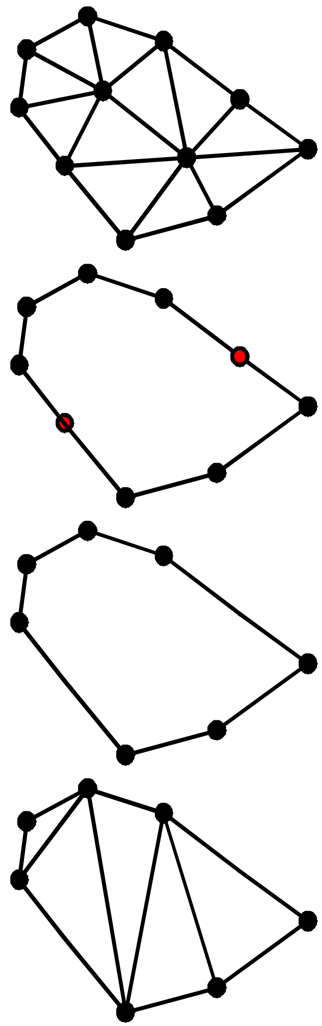
- iterative, incremental method
- error is bounded
- allows topology simplification (aggregation of disconnected components)
- results are very high quality and ***times incredibly short***
- Various commercial packages use this technique (or variations)

Not-incremental methods:

- coplanar facets merging
  [Hinker et al. `93,  Kalvin et al. `96]
- re-tiling
  [Turk `92]
- clustering
  [Rossignac et al. `93, ... + others]
- wavelet-based
  [Eck et al. `95]

Coplanar Facets Merging

Geometric Optimization
[Hinker '93]



- Construct nearly co-planar sets (comparing normals)
- Create edge list and remove duplicate edges
- Remove colinear vertices
- Triangulate resultant polygons

Geometric Optimization - Evaluation

simple and efficient heuristic

- evaluation of approximation error is highly inaccurate and not bounded

  (error depends on relative size of merged faces)

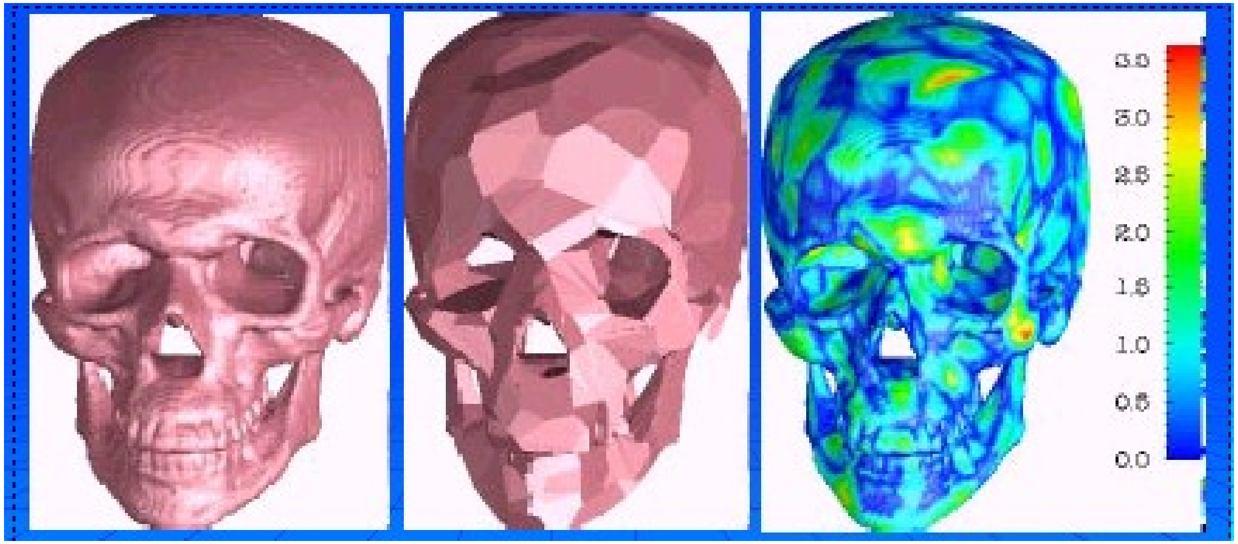
- vertices are a subset of the original
- preserves geometric discontinuities (e.g. sharp edges) and topology

Superfaces [Kalvin, Taylor '96]

- group mesh faces in a set of *superfaces*:
  - iteratively choose a seed face f_i as the current *superface* Sf_j
  - find by propagation all faces adjacent to f_i whose vertices are at distance $\varepsilon/2$ from the mean plane to Sf_j and insert them in Sf_j
  - moreover, to be merged each face must have orientation similar to those of others in Sf_j
- straighten the *superfaces* border
- re-triangulate each *superface*

Superfaces - an example

- Simplification of a human skull (fitted isosurface), *images courtesy of IBM*



Superfaces - Evaluation

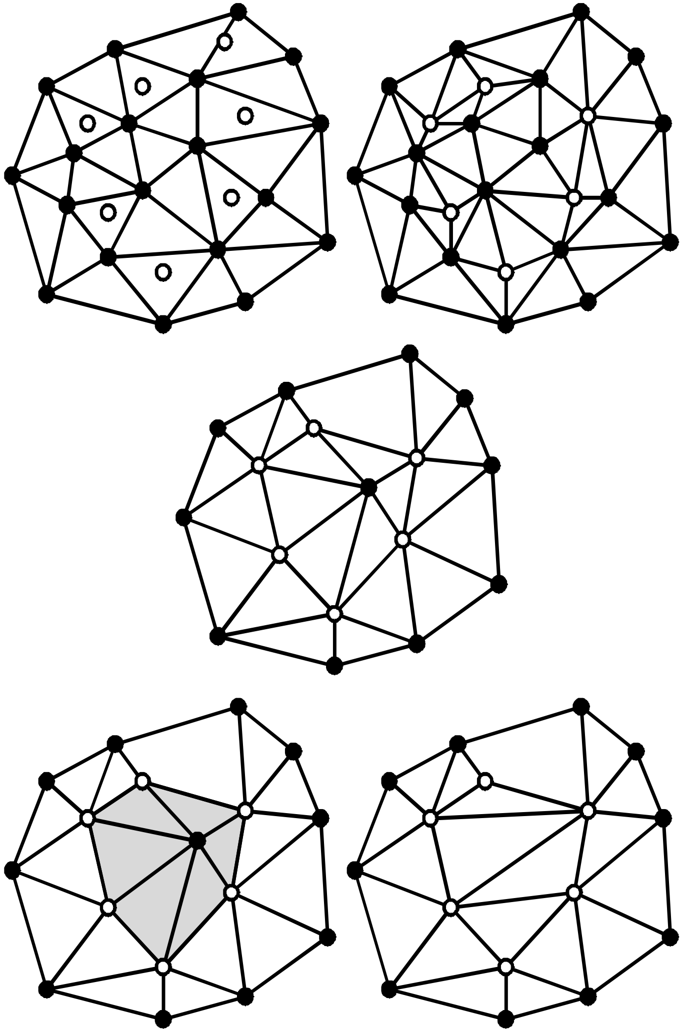
- slightly more complex heuristics
- evaluation of approximation error is more accurate and bounded
- vertices are a subset of the original ones
- preserves geometric discontinuities (e.g. sharp edges) and topology

Re-tiling

Re-Tiling **[Turk `92]**

- Distribute a new set of vertices into the original triangular mesh (points positioned using repulsion/relaxation to allow optimal surface curvature representation)
- Remove (part of) the original vertices
- Use local re-triangulation

no info in the paper on time complexity!

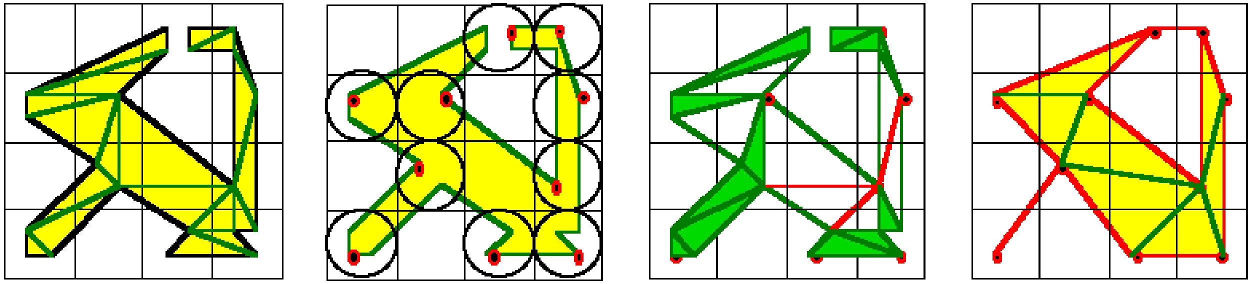


Clustering

Vertex Clustering **[Rossignac, Borrel `93]**

- detect and unify *clusters* of nearby vertices

  (discrete gridding and coordinates truncation)
- all faces with two or three vertices in a cluster are removed
- does not preserve topology (faces may degenerate to edges, genus may change)
- approximation depends on grid resolution



(figure by Rossignac)

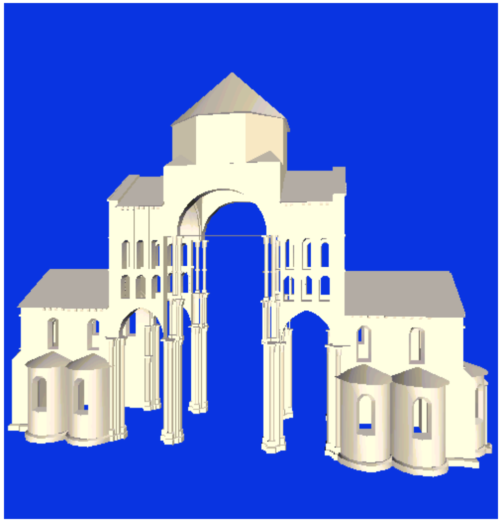
- Simplification of a table lamp,
  IBM 3D Interaction Accelerator,

  courtesy IBM

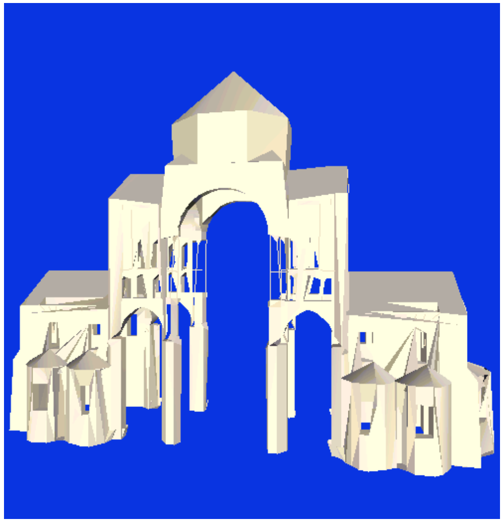
10,108 facets 1,383 facets 474 facets 46 facets

Clustering -- Examples (2)

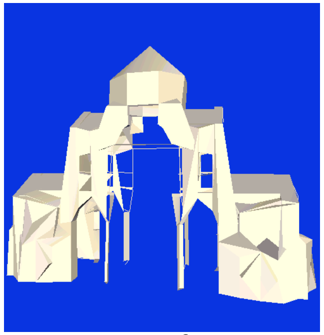
- Simplification of a portion of Cluny Abbey, IBM 3D Interaction Accelerator, courtesy IBM France.



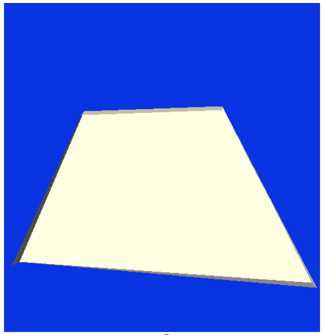
46,918 facets



6,181 facets



1,790 facets



16 facets

Clustering - *Evaluation*

- high efficiency (but timings are not reported in the paper)
- very simple implementation and use
- low quality approximations
- does not preserve topology
- error is bounded by the grid cell size

Multiresolution Analysis [Eck et al. '95, Lounsbery'97]

- Based on the ***wavelet*** approach
  - simple base mesh
  - + local correction terms (wavelet coefficients)

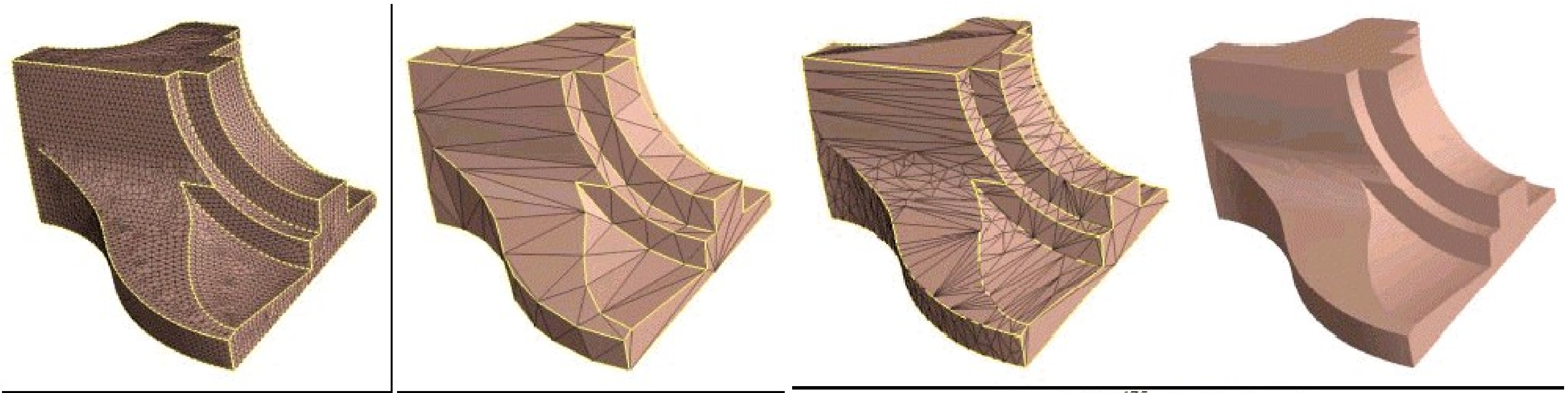
- Given input mesh M:
  - ***partition*** : build a low resolution base mesh K_0 with tolerance η_1
  - ***parametrization*** : for each face of K_0 build a parametrization on the corresponding faces of M
  - ***resampling*** : apply ***j*** recursive quaternary subdivision on K_0 to build by parametrization different approximations $\boldsymbol{K_j}$

- Supports:

  bounded error, compact multiresolution repr., mesh editing at multiple scales

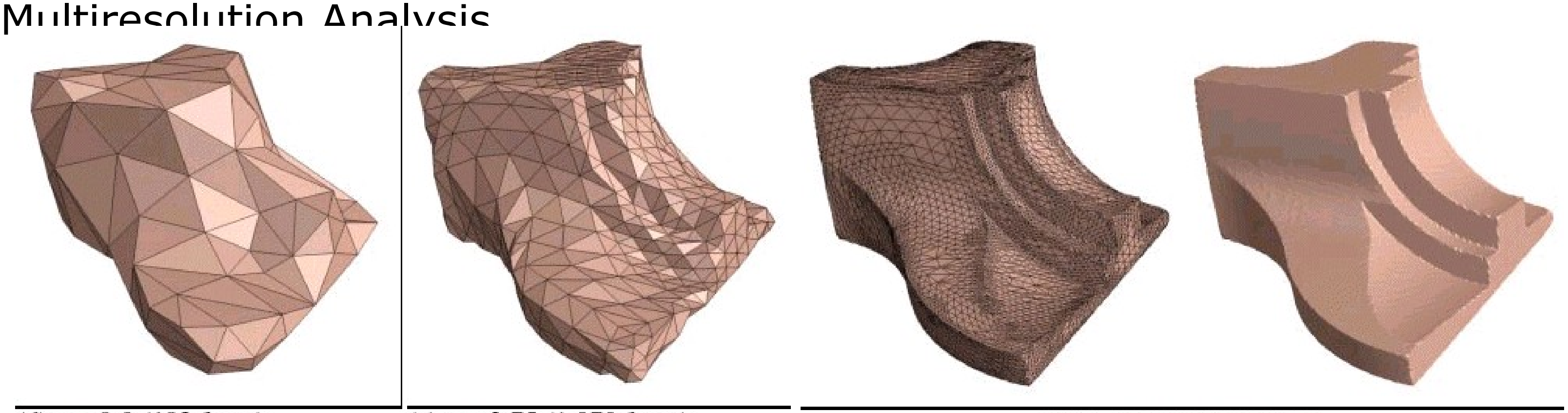
Hoppe's experiment: comparative eval. of quality of multiresolution representation

- Progressive Meshes



(a) $\hat{M}$ (12,946 faces) (b) M^{75} (200 faces) (c) M^{475} (1,000 faces)

- Multiresolution Analysis

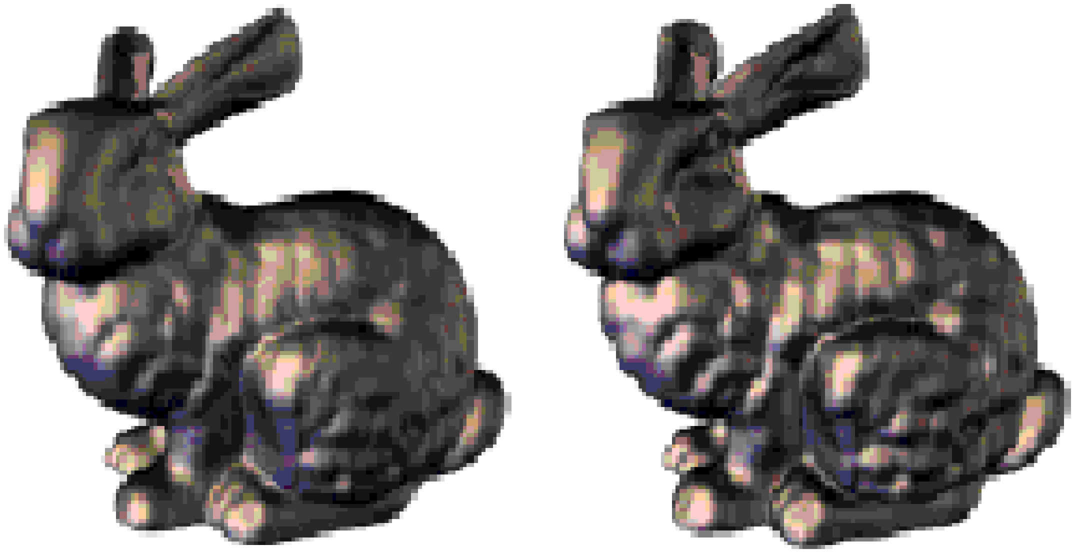


(d) $\epsilon = 9.0$ (192 faces) (e) $\epsilon = 2.75$ (1,070 faces) (f) $\epsilon = 0.1$ (15,842 faces)

Multires Signal Processing for Meshes

[Guskov, Swelden,Schroeder 99]

- Still the ***Partition, Parmetrization and Resampling*** approach but the original mesh connectivity is retained:
  - partition is done on the simplified mesh
  - use of a ***non-uniform relaxation procedure*** (instead of standard triangle quadrisection) that mimics the inverse simplification process
  - Possibility of using signal processing techniques on mesh (eg. Smoothing, detail enhancement …)



[image by courtesy of Guskov et al. 99]

Preserving detail on simplified meshes

- **Problem Statement :**

how can we preserve in a *simplified* surface
the **detail** (or **attribute value**)
defined on the *original* surface ??

- **What one would preserve:**
  - **color** (per-vertex or texture-based)
  - **small variations of shape curvature** (bumps)
  - **scalar fields**
  - **procedural textures** mapped on the mesh

Approaches proposed in literature are:

- **integrated** in the simplification process

  (ad hoc solutions **embedded** in the simplification codes)

- **independent** from the simplification process

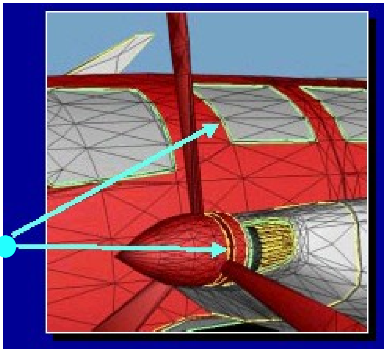
  (post-processing phase to restore attributes detail)

Integrated approaches:

- attribute-aware simplification
  - do not simplify an element ***e*** **IF** ***e*** is on the boundary of two regions with different attribute values

or

  - use an enhanced multi-variate approximation evaluation metrics (shape+color+...)
    [Hoppe96,GarHeck98,Frank etal98, Cohen etal98]



(image by H. Hoppe)

- store removed detail in textures
  - *vertex-based* [Maruka95 , Soucyetal96]
  - *texture-based* [Krisn.etal96]

- preserve **topology** of the attribute field [Bajaj et al.98]

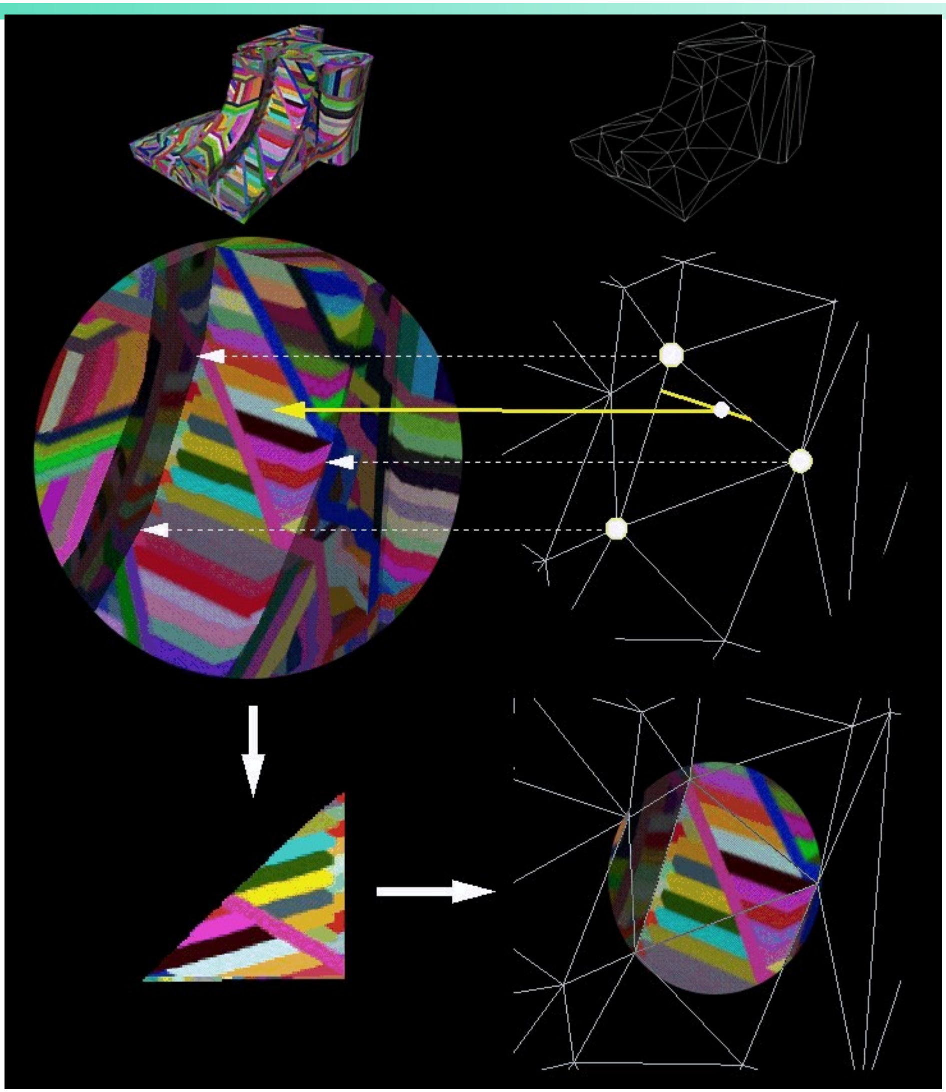
Simplification-Independent approach:

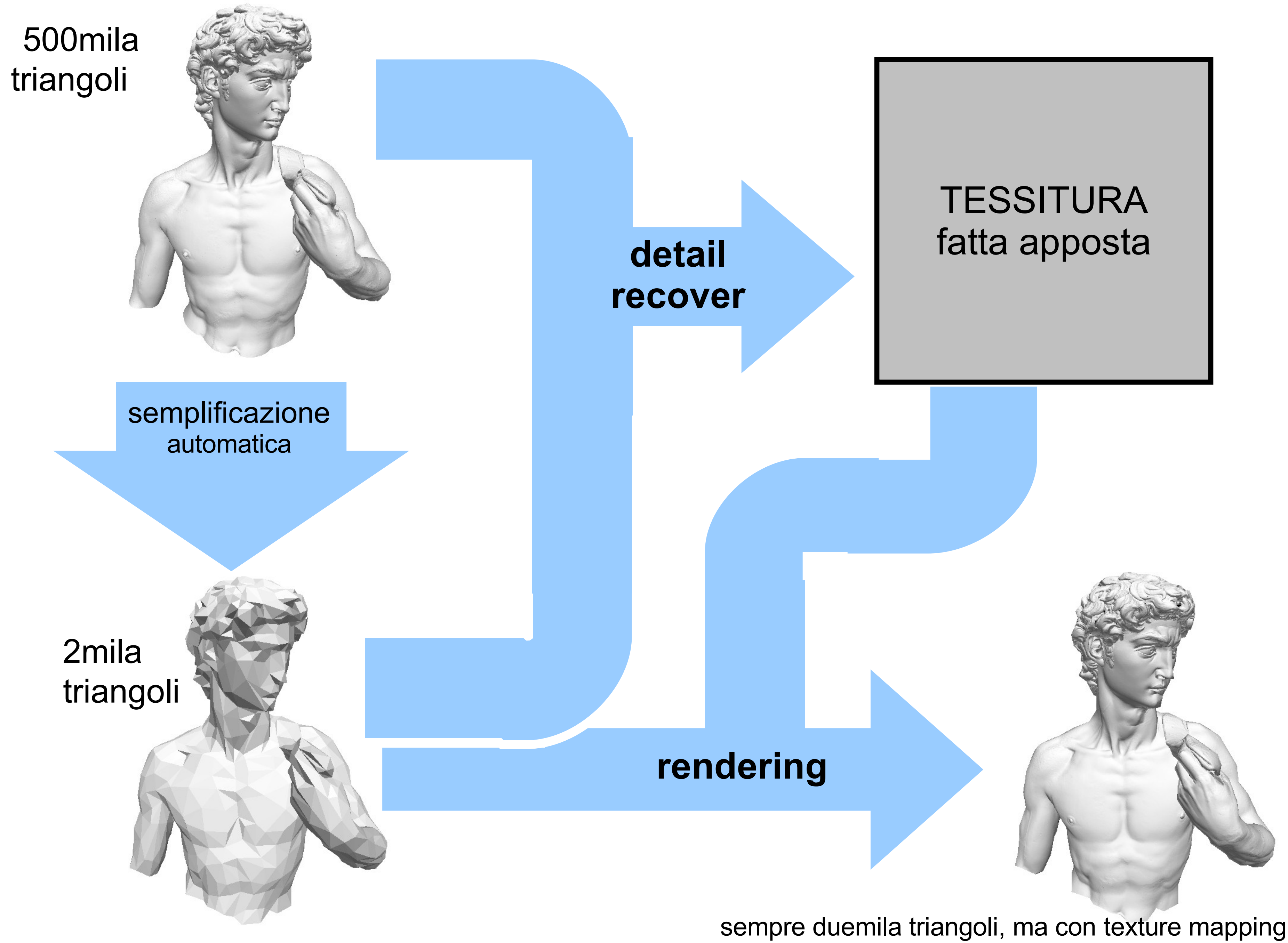
our Vis'98 paper [Cignoni etal 98]

- ***higher generality:*** attribute/detail preservation is not part of the simplification process
- performed as a ***post-processing*** phase (after simplification)
- any attribute can be preserved, by constructing an ad-hoc ***texture map***
- Used today in most games...

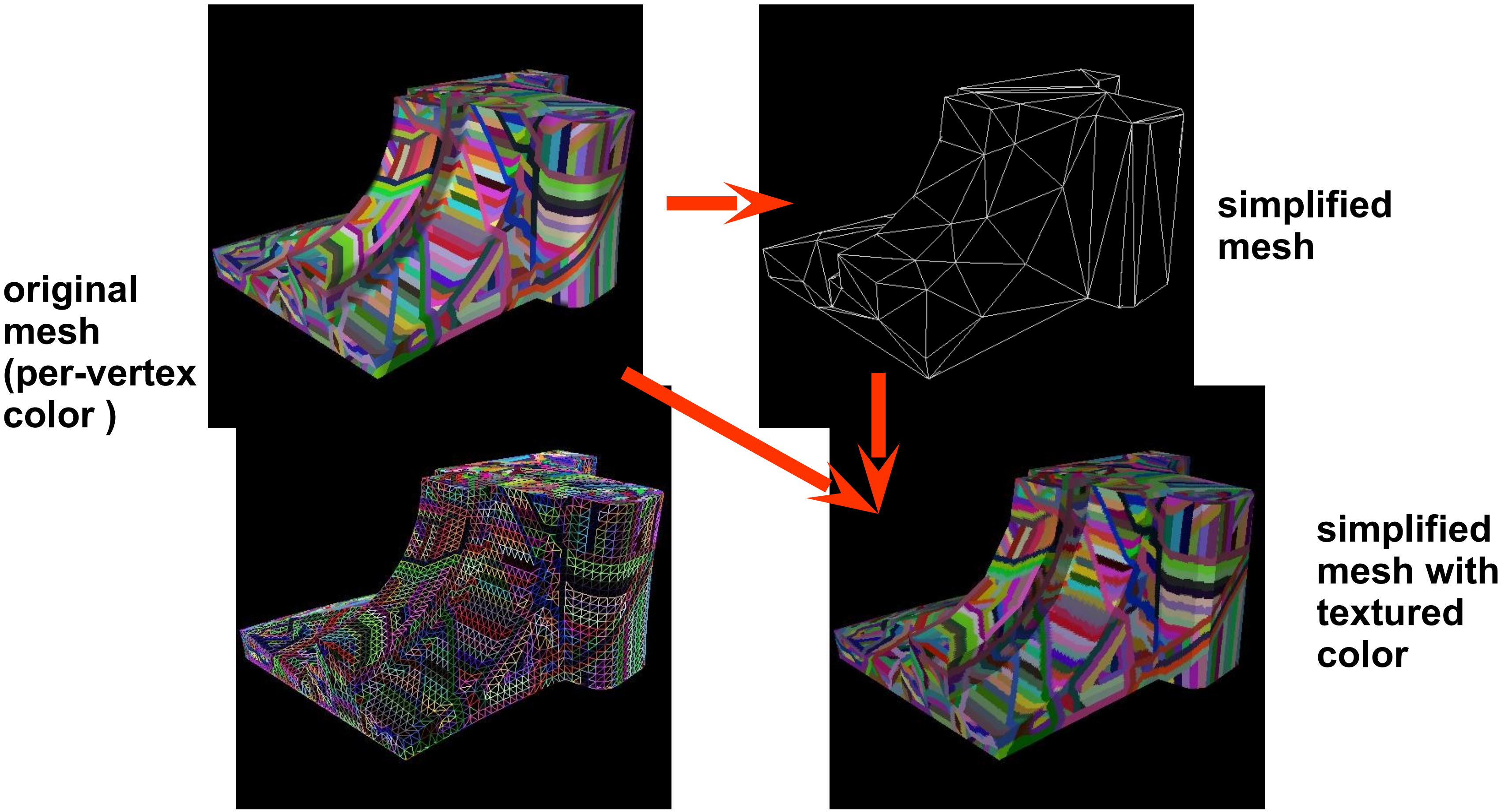
A simple idea: ... Preserving detail: Simplif.-Independent...

- for each texel simplified face:
  - detect the original detail by choosing either the closest point or along the normal.

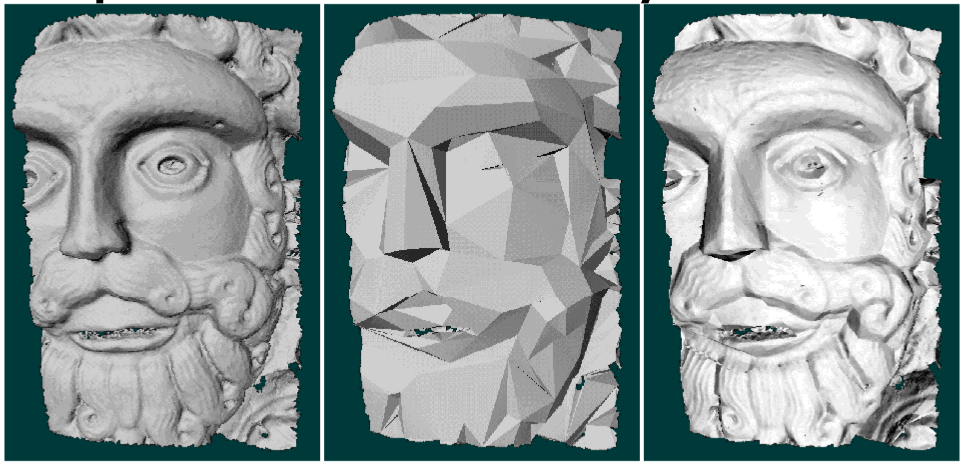



500mila
triangoli
semplificazione
automatica
2mila
triangoli
detail
recover
TESSITURA
fatta apposta
rendering
sempre duemila triangoli, ma con texture mapping

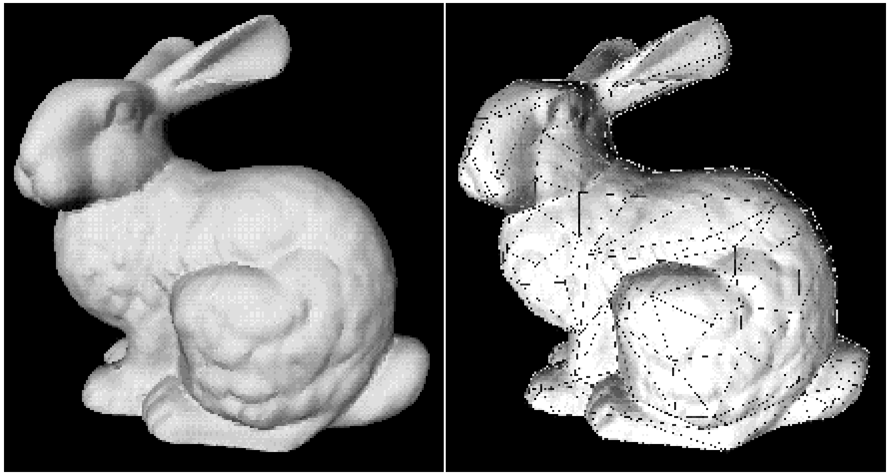
- an example of ***color*** preservation



- example of ***geometric detail*** preservation by **normal *mapping***



Original 20k face
simplified 500 face



Original 60k faces
simplified 250 faces

originale
500K triangles

semplificato ma con tessitura
2K triangles

Incremental Simp Method

The common framework:

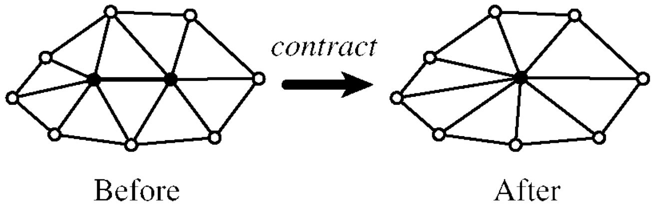
- **loop**
  - ***select*** the element to be deleted/collapsed;
  - ***evaluate approximation*** introduced;
  - ***update*** the mesh after deletion/collapse;

until mesh **size/precision** is satisfactory;

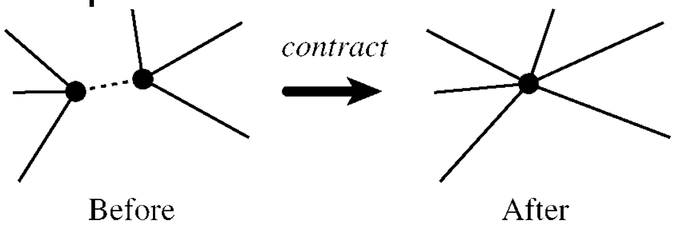
Simplification using Quadric Error Metrics

[Garland et al. Sig'97]

- Based on incremental **edge-collapsing**



- **but** can also collapse vertex couples which are **not connected** (topology is not preserved)

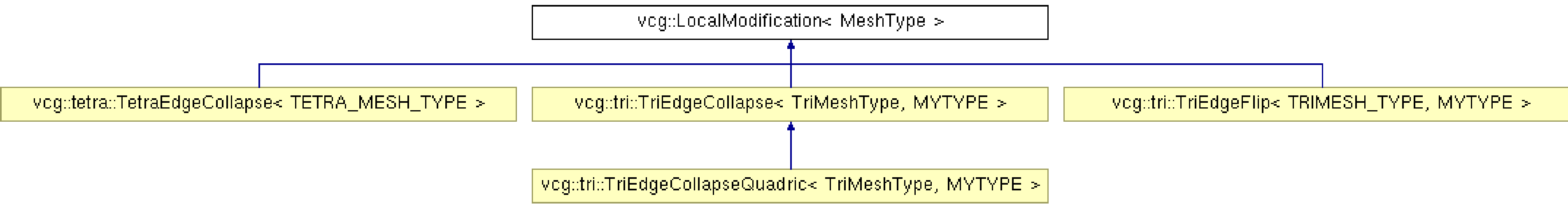


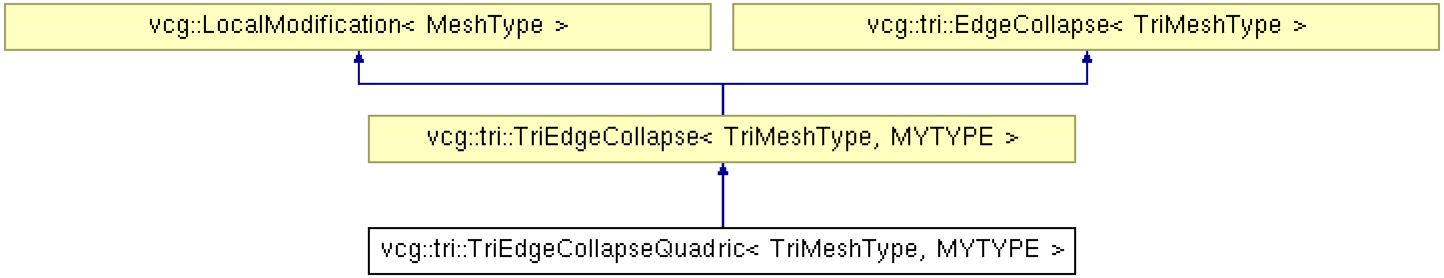
The main simplification loop

```
vcg::LocalOptimization<MyMesh> DeciSession(mesh);

DeciSession.Init<MyTriEdgeCollapse >();

DeciSession.SetTargetSimplices(FinalSize);
DeciSession.SetTimeBudget(0.5f);

while(DeciSession.DoOptimization() && mesh.fn>FinalSize)
    printf("Current Mesh size %7i heap sz %9i err %9g \r",

   mesh.fn,DeciSession.h.size(),DeciSession.currMetric);

printf("mesh  %d %d Error %g \n",
                   mesh.vn,mesh.fn,DeciSession.currMetric);
```

Classi in gioco

- LocalOptimization
  - Classe astratta per il loop di ottimizzazione
- LocalModification
  - Classe astratta per una generica operazione che modifica la mesh localmente con un certo costo
- EdgeCollapse
- TriEdgeCollapse
  - Particolare local modification
- TriEdgeCollapse Quadric


vcg::LocalModification< MeshType >
vcg::tetra::TetraEdgeCollapse< TETRA_MESH_TYPE >
vcg::tri::TriEdgeCollapse< TriMeshType, MYTYPE >
vcg::tri::TriEdgeFlip< TRIMESH_TYPE, MYTYPE >
vcg::tri::TriEdgeCollapseQuadric< TriMeshType, MYTYPE >


vcg::LocalModification< MeshType >
vcg::tri::EdgeCollapse< TriMeshType >
vcg::tri::TriEdgeCollapse< TriMeshType, MYTYPE >
vcg::tri::TriEdgeCollapseQuadric< TriMeshType, MYTYPE >

Local Modification

```
template <class MeshType> class LocalModification
{
 public:
        typedef typename LocalOptimization<MeshType>::HeapType HeapType;
        typedef typename MeshType::ScalarType ScalarType;

inline LocalModification(){};
virtual ~LocalModification(){};

virtual ModifierType IsOfType() = 0 ; /// return the type of operation
/// return true if the data have not changed since it was created
virtual bool IsUpToDate() = 0 ;
/// return true if no constraint disallow this operation to be performed (ex:
   change of topology in edge collapses)
virtual bool IsFeasible() = 0;
/// Compute the priority to be used in the heap
virtual ScalarType ComputePriority()=0;
/// Return the priority to be used in the heap (implement static priority)
virtual ScalarType Priority() const =0;
/// Perform the operation and return the variation in the number of simplicies
    (>0 is refinement, <0 is simplification)
virtual void Execute(MeshType &m)=0;
/// perform initialization
static void Init(MeshType &m, HeapType&);
  virtual const char *Info(MeshType &) {return 0;}
/// Update the heap as a consequence of this operation
virtual void UpdateHeap(HeapType&)=0;
};       //end class local modification
```

Local Modification

- Classe astratta generica
  - Potrebbe essere un edge collapse
  - Uno swap
  - Un vertex deletion ecc.
- Astrarre una generica operazione di modifica locale alla mesh
  - Adatta ad essere prioritizzata
    - Deve saper dare una priorita'
  - Sapersi applicare alla mesh
  - Sapere se e' sempre valida

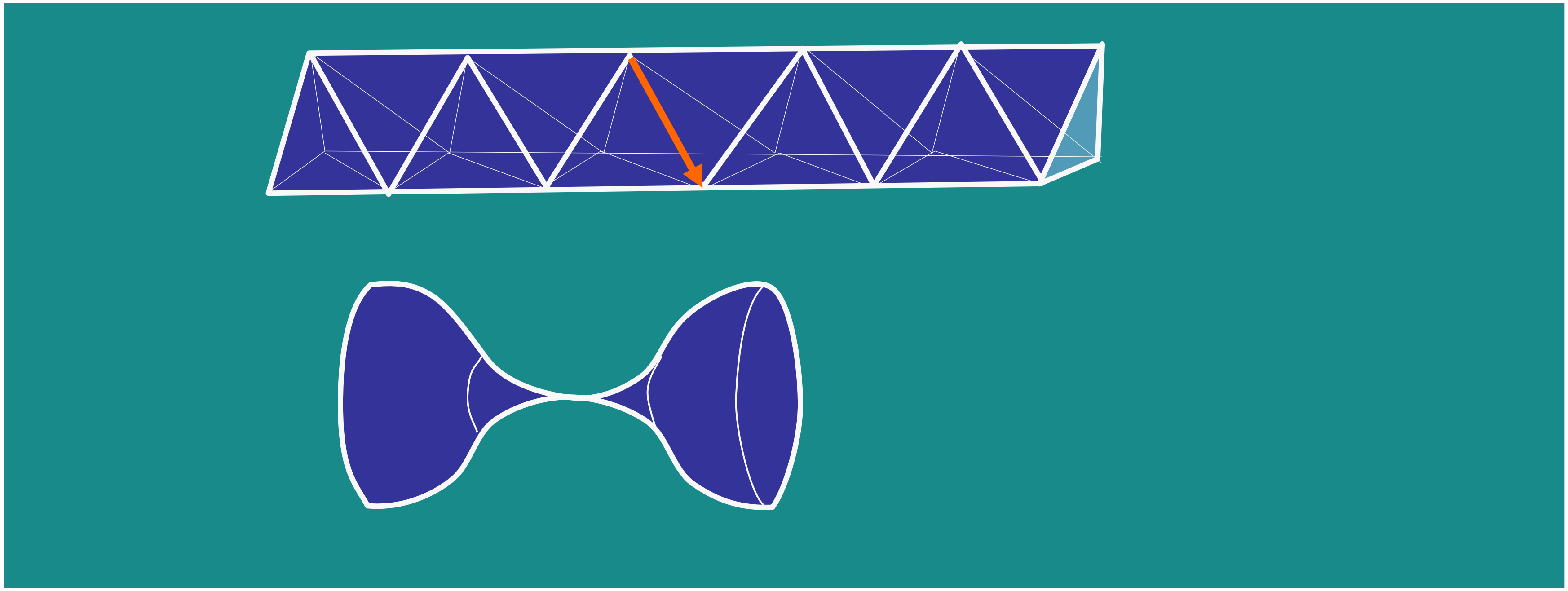
EdgeCollapse e TriEdgeCollapse

- EdgeCollapse
  - Classe astratta per rappresentare un collasso di un edge su una generica mesh
  - Non sa nulla di priorita' quadriche ecc
- TriEdgeCollapse
  - Generica local op basata su collasso
    - Sa aggiornare lo heap
    - Eseguirsi, sapere se e' valida ecc.
  - Da questa si deriva quella con le quadriche

```
template <class TRI_MESH_TYPE>
class EdgeCollapse
{
typedef typename vcg::face::VFIterator<FaceType>  VFI;
typedef typename std::vector<vcg::face::VFIterator<FaceType> >
   VFIVec;

static VFIVec & AV0(){static VFIVec av0; return av0;}
static VFIVec & AV1(){static VFIVec av1; return av1;}
static VFIVec & AV01(){static VFIVec av01; return av01;}

bool LinkConditions(EdgeType  pos);
void FindSets(EdgeType &p)bool LinkConditions(EdgeType  pos);
int DoCollapse(EdgeType & c, const Point3<ScalarType> &p);
}
```

Topology Preservation

- 2-Manifold
  - A surface S in ***R²*** such that any point on S has an open neighborhood homeomorphic to an open disc or to half an open disc in ***R²***
- A edge collapse can create non manifold situations



Topology Preservation

- Let Σ be a 2 simplicial complex ***without boundary***
- Σ’ is obtained by collapsing the edge $e = (ab)$
- Let $Lk\ (\sigma)$ be the set of all the faces of the co-faces of σ disjoint from σ

Σ and Σ’ are homeomorphic ***iff***

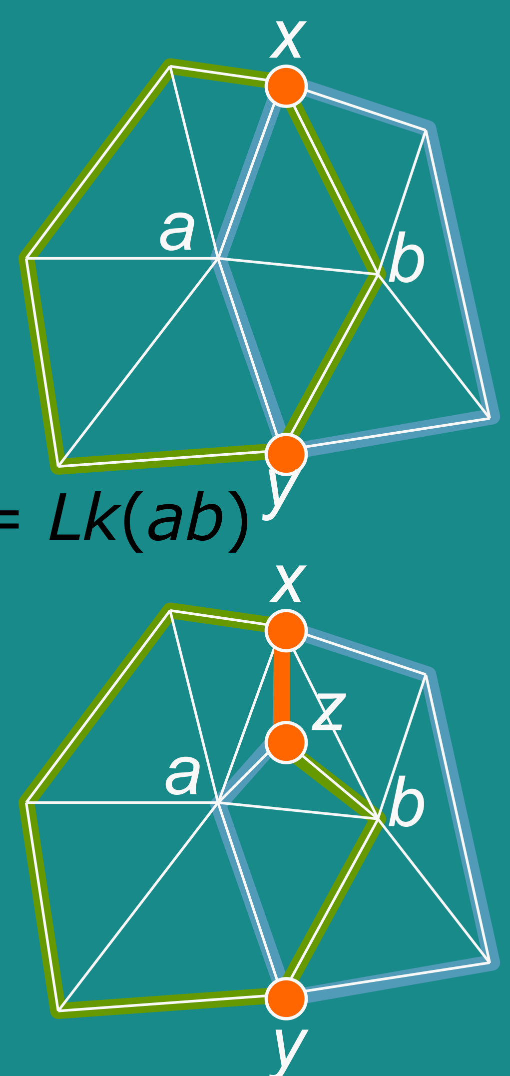
$$Lk\ (a) \cap Lk\ (b) = Lk\ (ab)$$

[Dey 99]

Topology Preservation

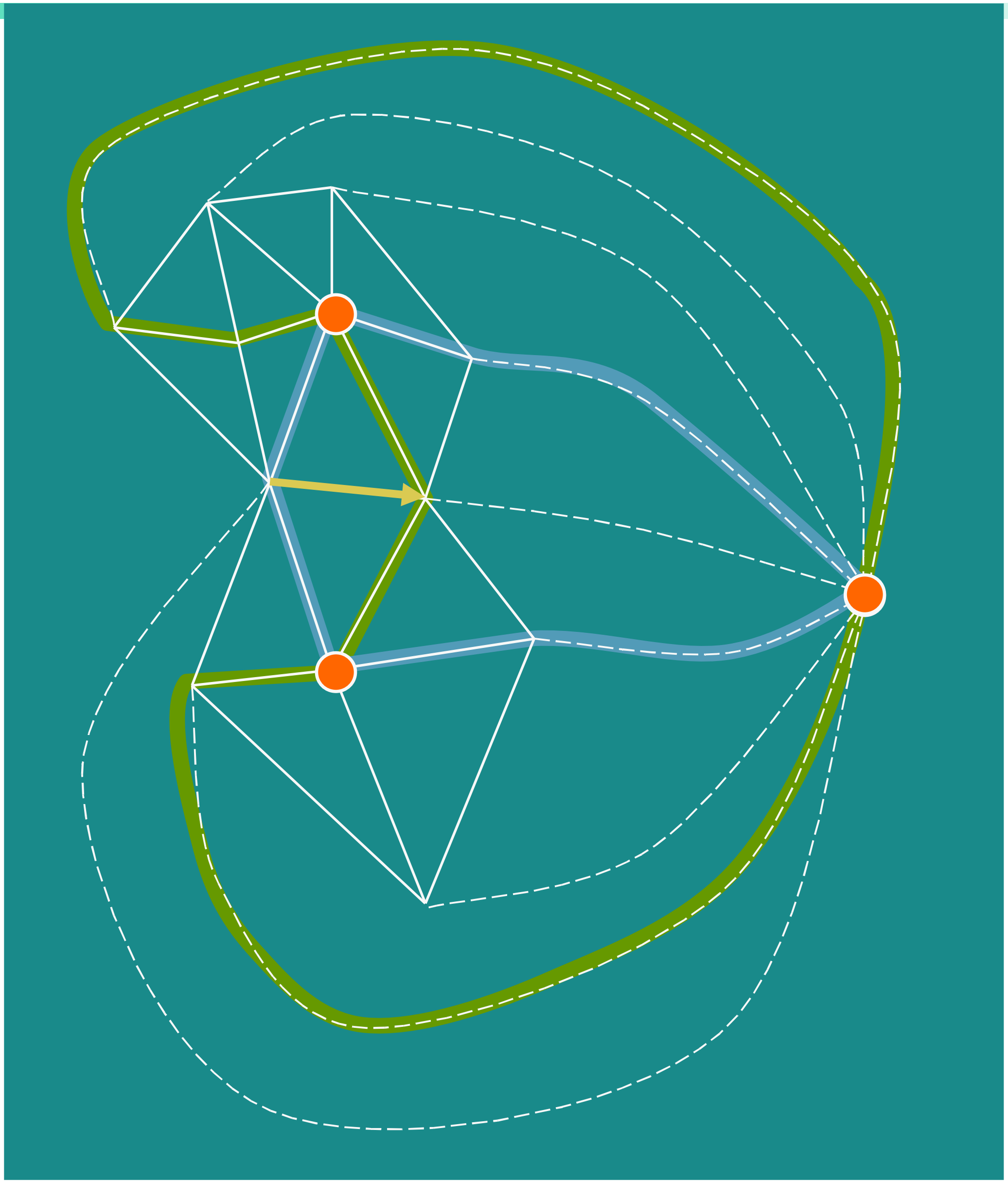
$Lk(a) \cap Lk(b) = \{x, y\} = Lk(ab)$

$Lk(a) \cap Lk(b) = \{x, y, z, zx\} \neq \{y, z\} = Lk(ab)$



Topology Preservation

- Mesh with boundary can be managed by considering a dummy vertex v_d and, for each boundary edge *e* a tetrahedron connecting e with v_d
- Think it wrapped on the surface of a sphere
-

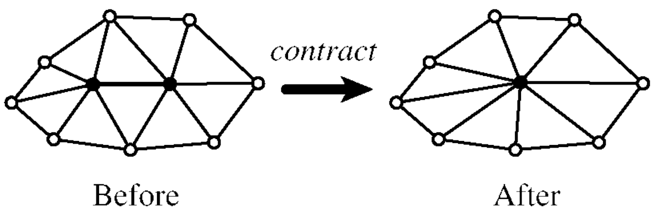


doCollapse

```
for(i=AV01().begin();i!=AV01().end();++i)
{
 FaceType  & f = *((*i).f);
 assert(f.V((*i).z) == c.V(0));
 vcg::face::VFDetach(f,((*i).z+1)%3);
 vcg::face::VFDetach(f,((*i).z+2)%3);
 f.SetD();
 n_face_del++;
}

//set Vertex Face topology
for(i=AV0().begin();i!=AV0().end();++i)
{
 (*i).f->V((*i).z) = c.V(1);
         // In tutte le facce incidenti in v0, si sostituisce v0 con v1
 (*i).f->VFp((*i).z) = (*i).f->V((*i).z)->VFp();
    // e appendo la lista di facce incidenti in v1 a questa faccia
(*i).f->VFi((*i).z) = (*i).f->V((*i).z)->VFi();
(*i).f->V((*i).z)->VFp() = (*i).f;
(*i).f->V((*i).z)->VFi() = (*i).z;
}

c.V(0)->SetD();
c.V(1)->P()=p;
return n_face_del;
```

Lazy heap

- Si suppone di avere uno heap con tutte le operazioni
- Estraggo da heap e aggiorno la mesh
  - tali operazioni invalidano/modificano la mesh e quindi le priorità/validità di parte delle azioni già presenti nello Heap



Lazy Heap

- Due Soluzioni
  - Link espliciti elementi mesh->heap e aggiornamento dello stesso
- Lazy update
  - Si mettono nello heap tutte le nuove operazioni con la nuova priorità
  - Quando si estrae un'op dall heap si controlla che sia sempre valida
  - Di tanto in tanto garbage collection sullo heap

Marche incrementali

- Strumento generico per marcare oggetti in una collezione con
  - C(mark elem) = O(1)
  - C(unmark elem) = O(1)
  - C(unmark All Elem) = O(1)

  - Memorizza per ogni elem un intero *mark* invece di un bit
  - Esiste una marca globale a livello della collezione di elementi

Marche incrementali

- Un oggetto è marcato se
  - elem.mark==global.mark
- Marcatura di un elem
  - elem.mark := global.mark
- Smarcatura globale
  - global.mark++

  - Spesso le marche vengono dette anche marche ***temporali*** per indicare che dicono quando un certo elem è valido

Validità collasso

- Dati
  - Ogni vertice ha una marca temporale:
    - quando e' stato modificato l'ultima volta
  - Ogni collasso (coppia di vertici) ha una marca temporale
    - quando è stato inserito nello heap
- Un collasso è valido se
  - I due vertici non sono stati cancellati
  - Il collasso e' stato messo nello heap piu recentemente della data di ultima modifica dei vertici

Quadric Error for Surfaces

- Let $\mathbf{n}^T\mathbf{v} + d = 0$ be the equation representing a plane
- The squared distance of a point $\boldsymbol{x}$ from the plane is

$$D(\mathbf{x}) = \mathbf{x}(\mathbf{n}\mathbf{n}^T)\mathbf{x} + 2d\mathbf{n}^T\mathbf{x} + d^2$$

- This distance can be represented as a quadric

$$Q = (\mathrm{A},\mathbf{b},c) = (\mathbf{n}\mathbf{n}^T, d\mathbf{n}, d^2)$$

$$Q(\mathbf{x}) = \mathbf{x}\mathrm{A}\mathbf{x} + 2\mathbf{b}^T\mathbf{x} + c$$

Quadric

- The boundary error is estimated by providing for each boundary vertex v a quadric Q_v representing the sum of the all the squared distances from the faces incident in v

• The error of collapsing an edge e=(v,w) can be evaluated as $Q_w(v)$.

• After the collapse the quadric of v is updated as follow $Q_v = Q_v + Q_w$