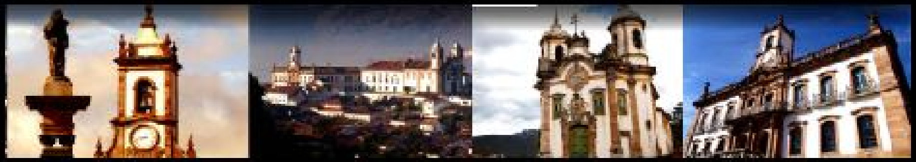
XXV SIBGRAPI - Conference on Graphics, Patterns and Images
SIBGRAPI 2012
August 22-25, Ouro Preto, Brazil



ESQ: editable SQuad representation for triangle meshes

Luca Castelli Aleardi
(Ecole Polytechnique, France)

ÉCOLE POLYTECHNIQUE

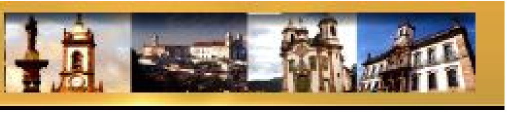
Olivier Devillers
Geometrica - INRIA Sophia

INRIA SOPHIA ANTIPOLIS

GEOMETRICA

Jarek Rossignac
(Georgia Institute of Technology)

Georgia Institute of Technology

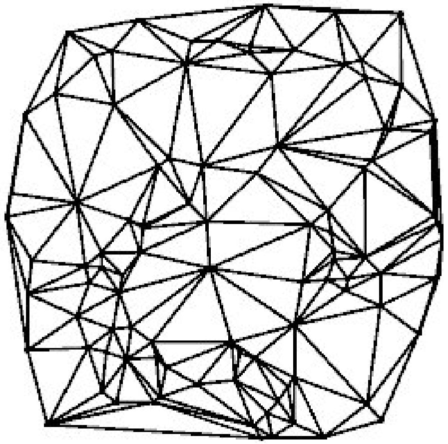


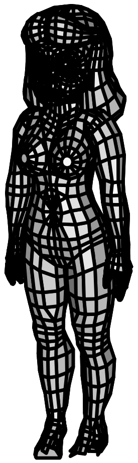
(Triangle) mesh encoding: compression and compact data structures

Before we start... Geometric data: (triangle) meshes

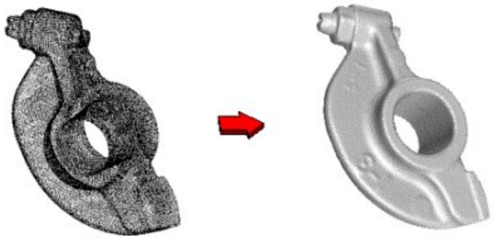
Among data structures for geometric data, I pick meshes...

(commonly used in Computational geometry and Geometry processing)

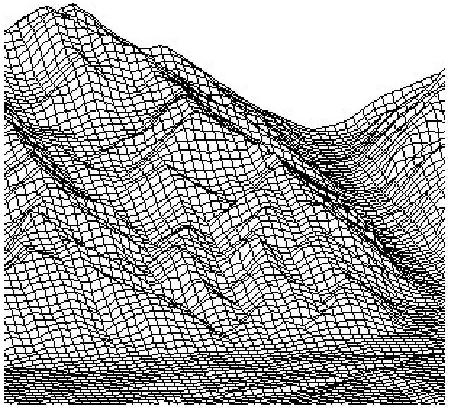




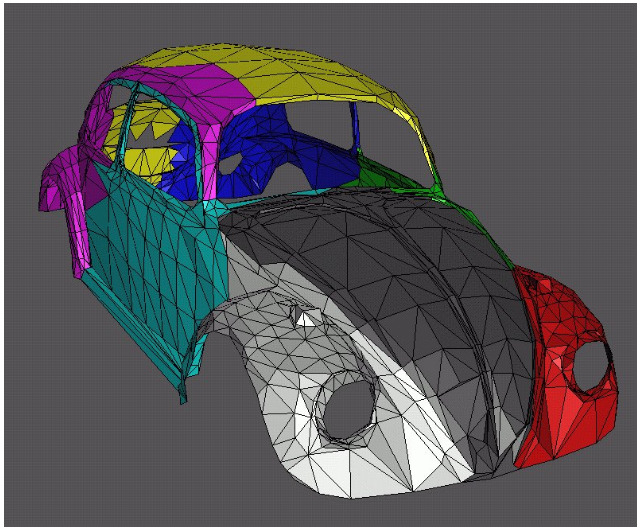
Surface recontruction from sampling



Geographic information systems

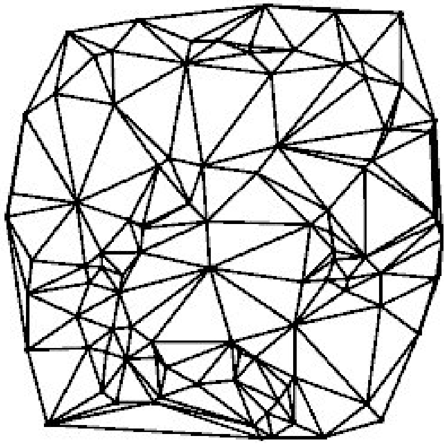


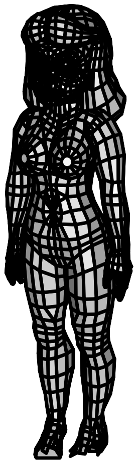
Surface modelling

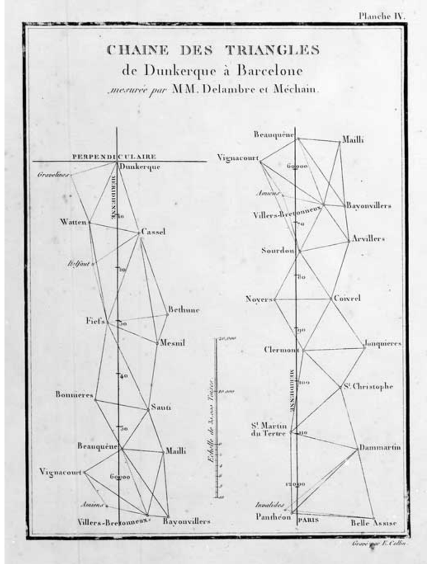


Before we start... Geometric data: (triangle) meshes

Among data structures for geometric data, I pick meshes...



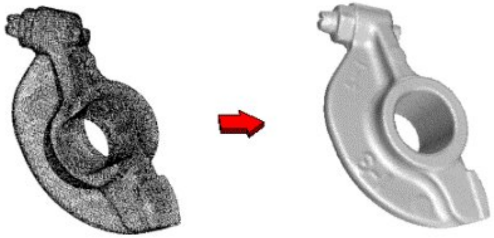




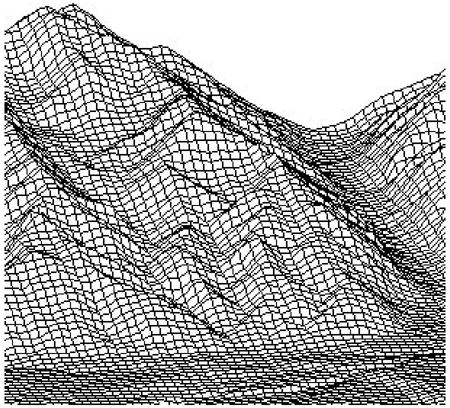
triangles meshes already used in early 19th century

(Delambre et Mchain)

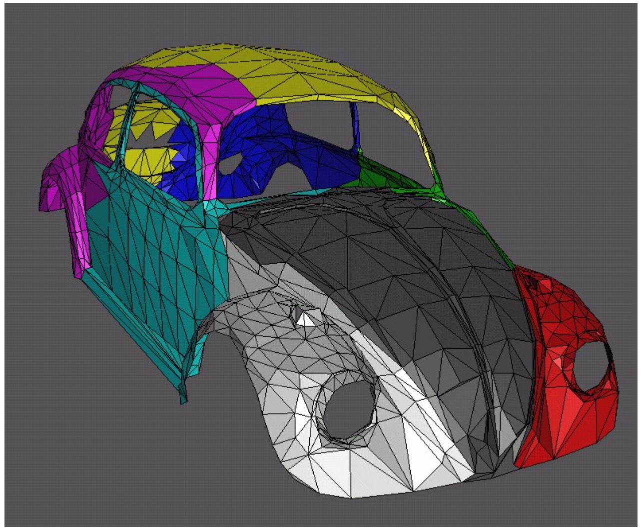
Surface recontruction from sampling



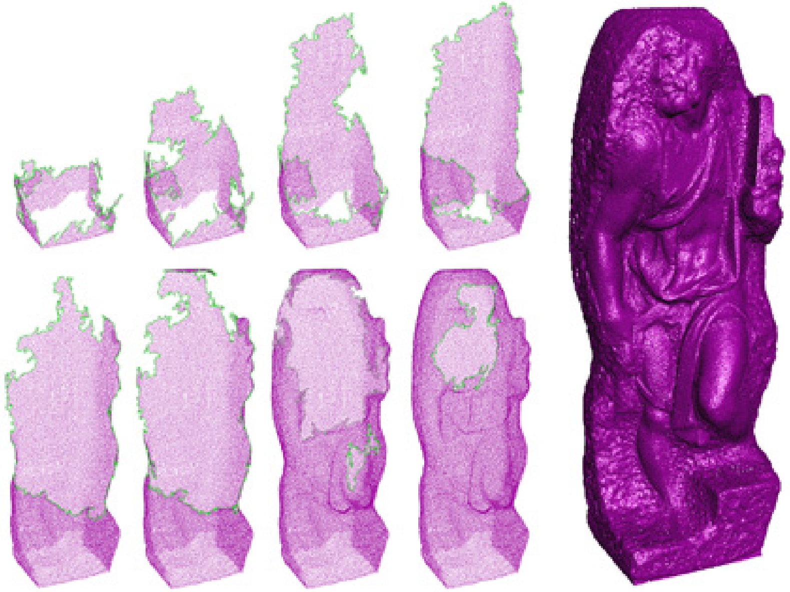
Geographic information systems



Surface modelling



Before we start... ∃ very large geometric data



St. Matthew (Stanford's Digital Michelangelo Project, 2000)
186 millions vertices
6 Giga bytes (for storing on disk)

several minutes for loading the model from disk

David statue (Stanford's Digital Michelangelo Project, 2000)

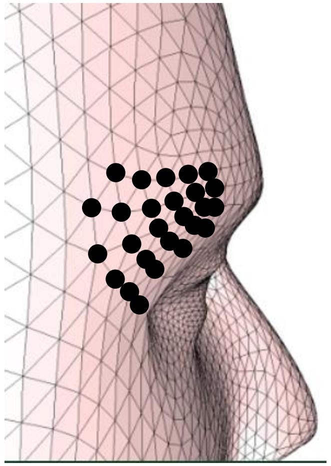
2 billions polygons
32 Giga bytes (without compression)

No existing algorithm nor data structure for dealing with the entire model

Geometric information vs Combinatorial information

Connectivity is by far the most expensive information

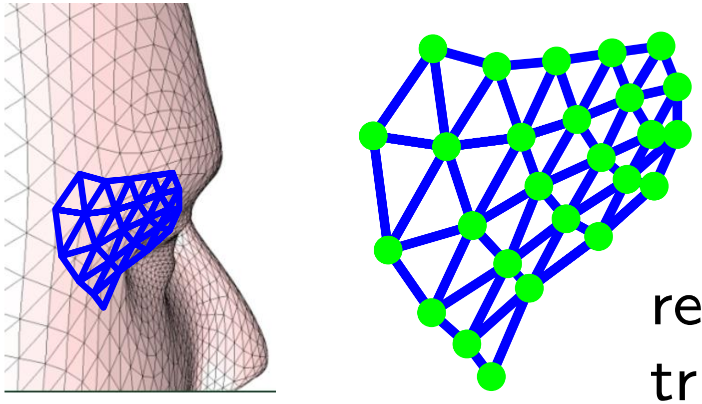
Geometry



vertex coordinates

between 30 et 96 bits/vertex

"Connectivity": the underlying triangulation



adjacency relations between triangles, vertices

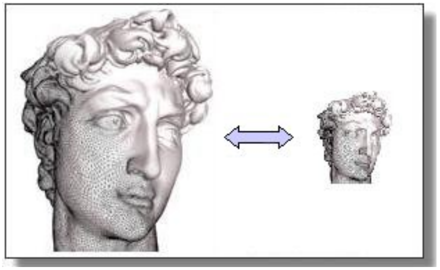
vertex	1 reference to a triangle
triangle	3 references to vertices 3 references to triangles

$13n \log n$ or $416n$ bits

1 reference: pointer or integer value (32 bits)

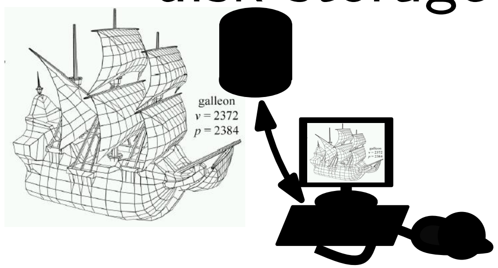
Before we start... What we are aiming at

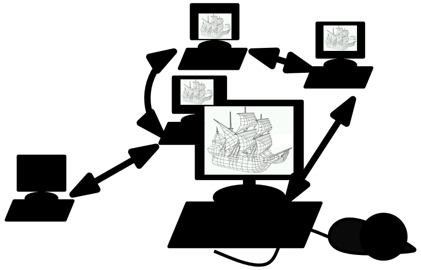
Mesh compression



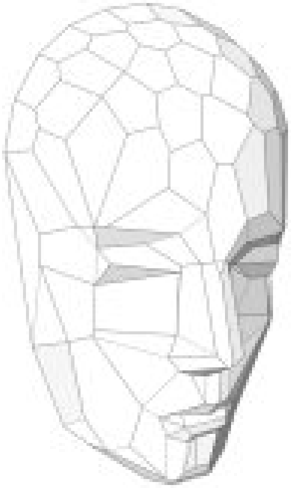
Transmission

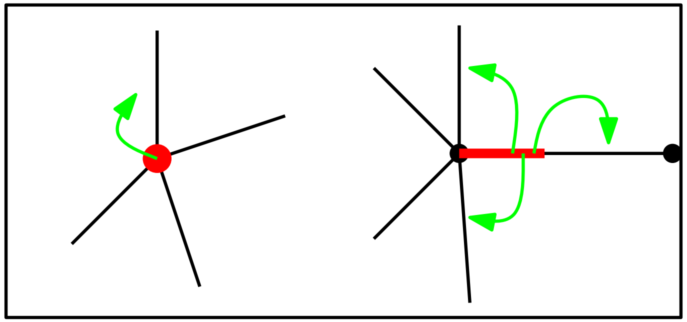
disk storage

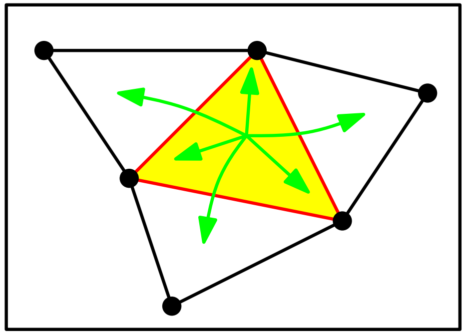




Geometric data structures

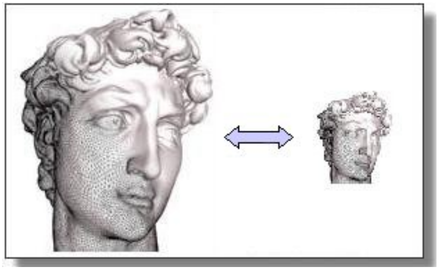




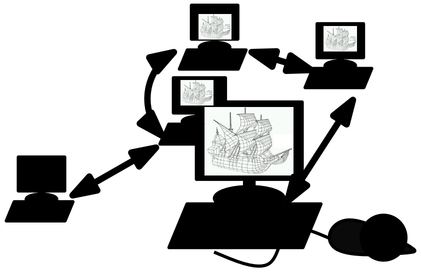


Before we start... What we are aiming at

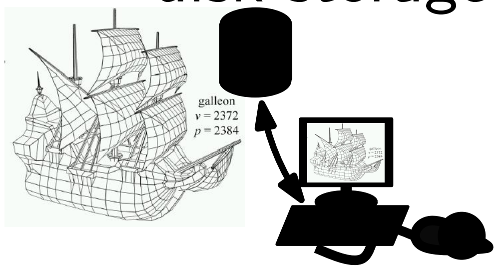
Mesh compression



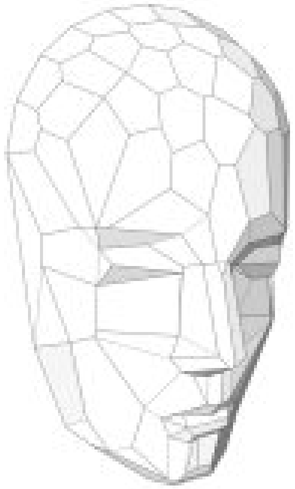
Transmission

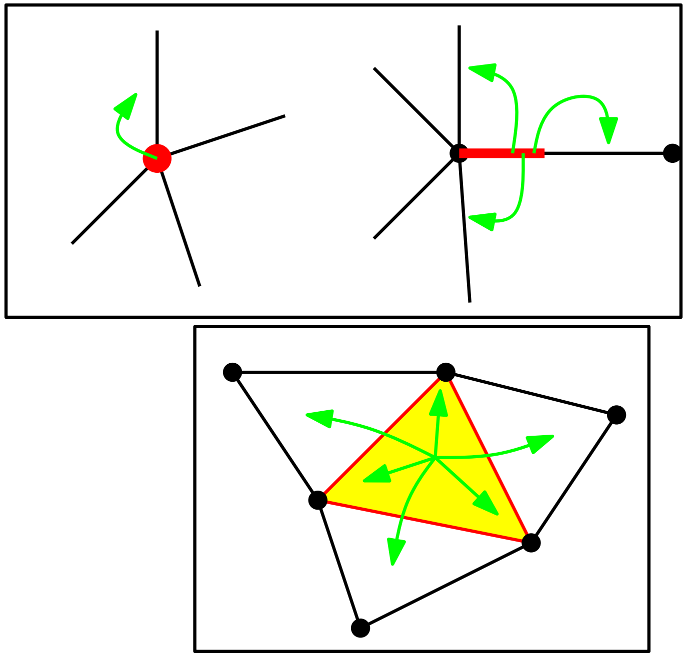


disk storage



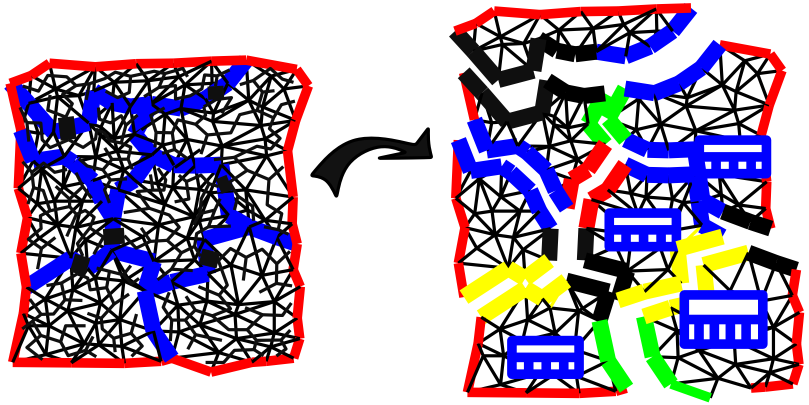
Geometric data structures

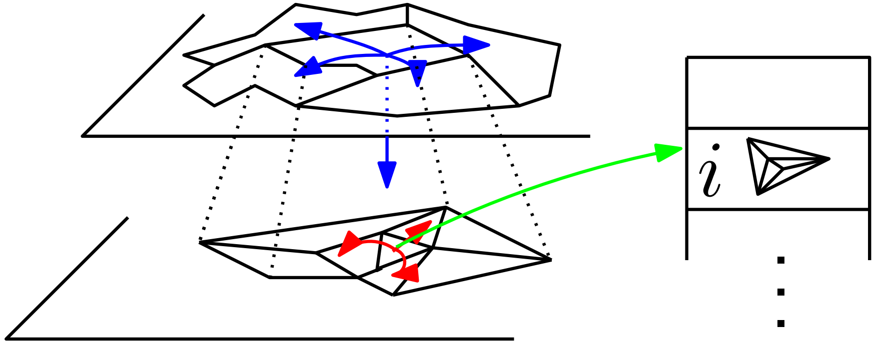




MERGE INTO: *Compact representations* of geometric data structures

(space-efficient data structures)

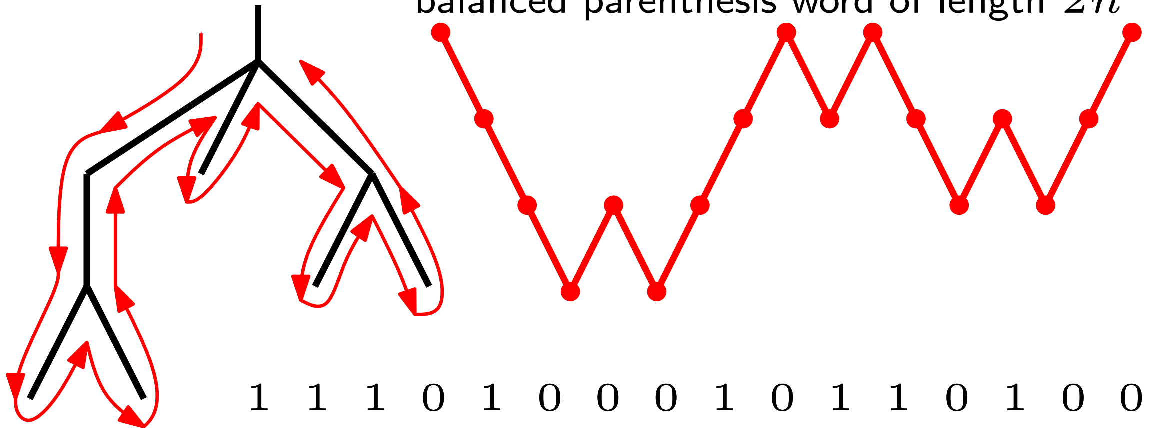




Starter: the encoding of plane trees

ordered tree with n edges

balanced parenthesis word of length $2n$

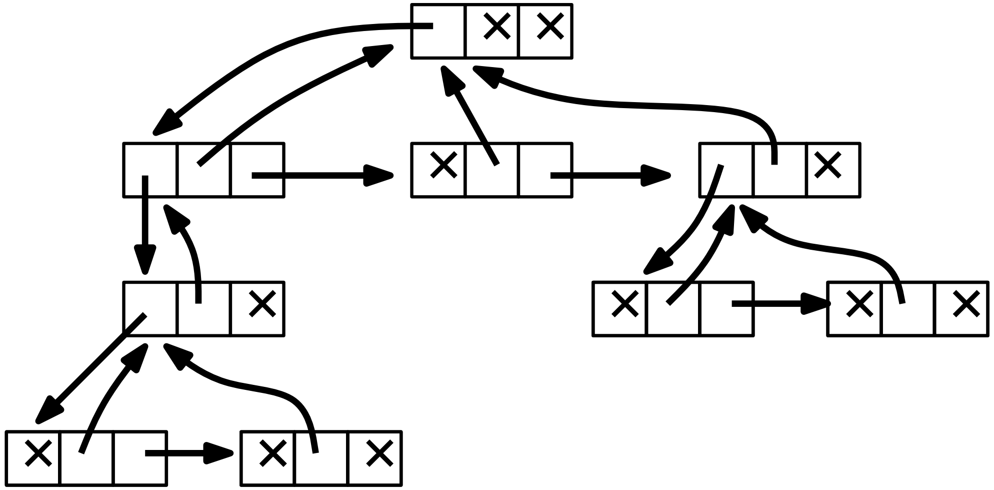


$\Rightarrow$ $2n$ bits for encoding an ordered tree with n edges

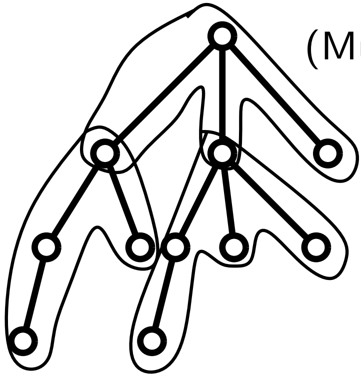
$$\|\mathcal{B}_n\| = \frac{1}{n+1}\binom{2n}{n} \approx 2^{2n} n^{-\frac{3}{2}}$$

This is an optimal encoding!

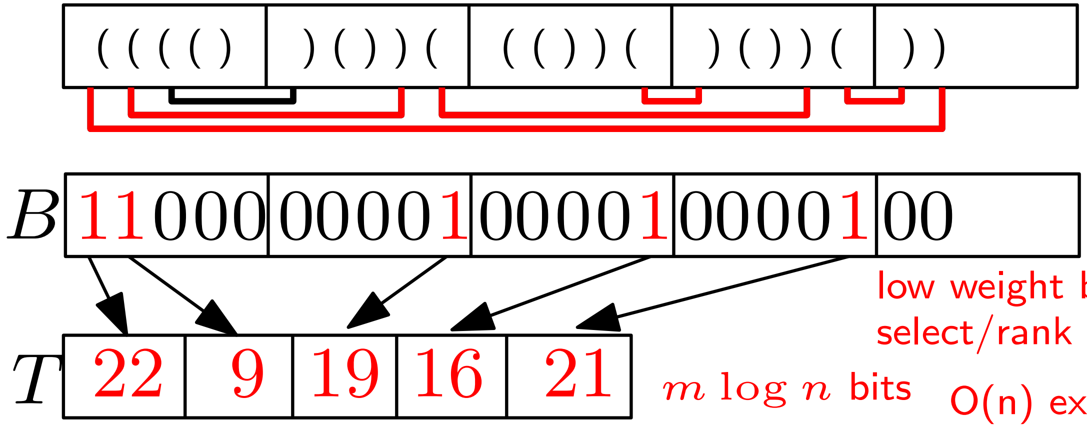
Compare to the standard explicit represention:



$3n$ pointers ≈ 96 bits



(Munro and Raman, Focs97)
(Jacobson, Focs89)



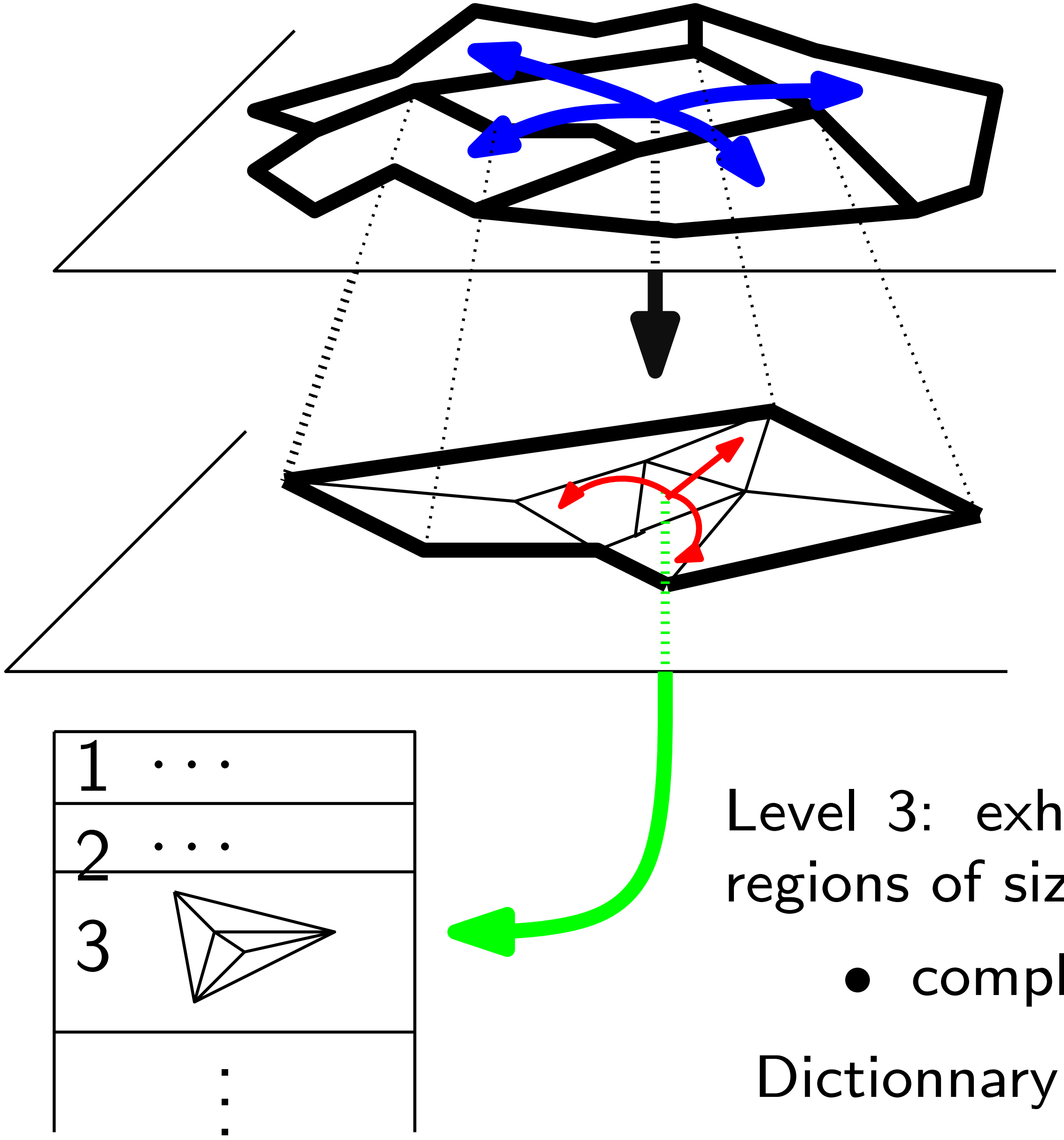
$2n$ bits

low weight bit vectors
select/rank queries

$m \log n$ bits

O(n) extra bits

A hierarchical approach, with a dictionary at bottom.



Level 1:

- $\Theta(\frac{n}{\log^2 n})$ regions of size $\Theta(\log^2 n)$, represented by pointers to level 2
- global pointers of size $\log n$

space $O(\frac{n}{\log^2 n} \cdot \log n) = o(n)$

Level 2:

in each of the $\frac{n}{\log^2 n}$ regions

- $\Theta(\log n)$ regions of size $C \log n$, represented by pointers to level 3
- local pointers of size $\log \log n$

space $O(\frac{n}{\log n} \cdot \log \log n) = o(n)$

Level 3: exhaustive catalog of all different regions of size $i < C \log n$:

- complete explicit representation.

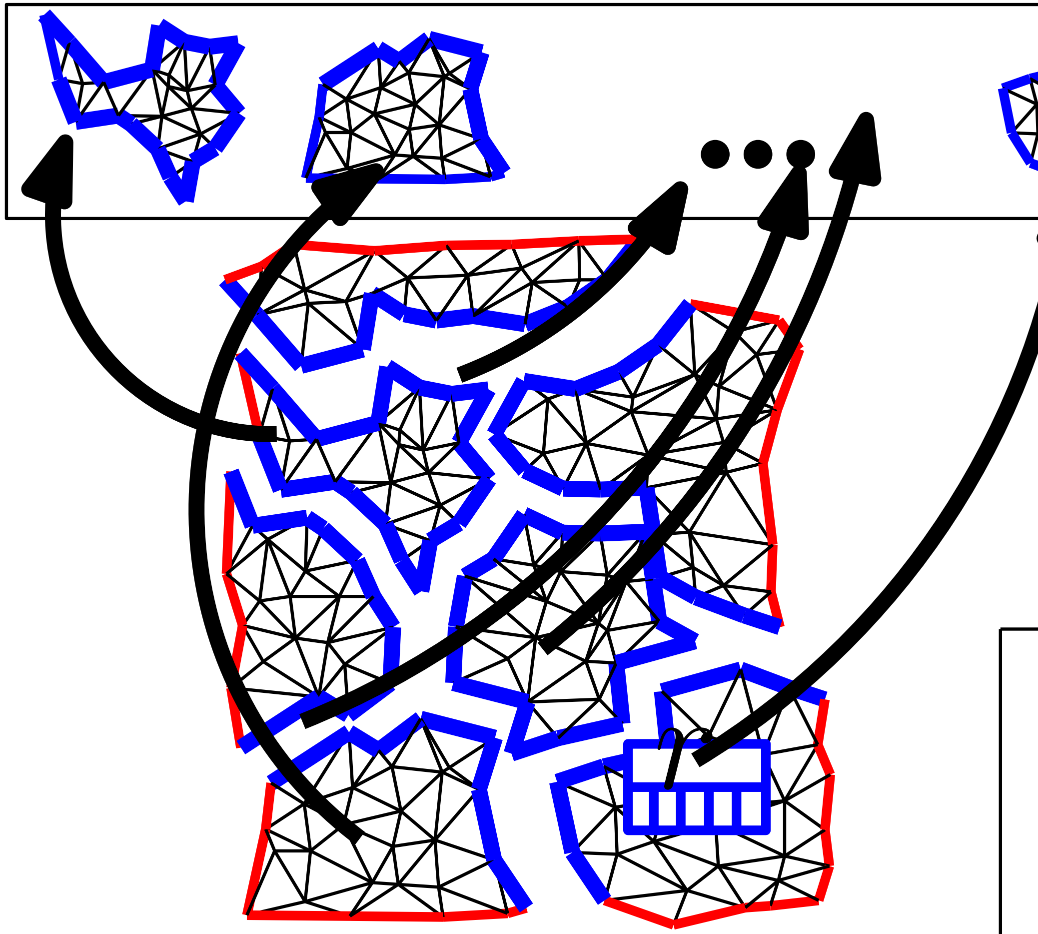
Dictionnary space is $o(n)$ if C small enough.

A hierarchical approach, with a dictionary at bottom.

Dominant term?

The dominant term is given by the sum of references to the dictionary

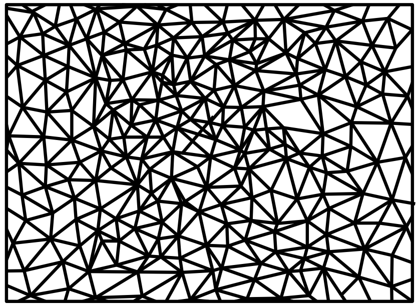
references on objects of $\mathcal{T}_k$ have size $\log_2 \mathcal{T}_k \sim 2.175k$ if $k \to \infty$



we should take all k s.t.
$\frac{1}{12} \log n < k < \frac{1}{2} \log n$

Adaptative to "reasonable" entropy reduction

$\sum_j 2.175k_j = 2.175m$ bits



2.175bpt is entropy of triangulations **with a boundary** larger than previous $\frac{1}{2} \cdot 3.24$bpt

General idea (literary digression)

one-act theatre play
(La leçon, Eugène Ionesco, 1951)

During a private lesson, a very young student, preparing herself for the total doctorate, talks about arithmetics with her teacher

(the young student cannot understand how to subtract integers)

Teacher Listen to me, If you cannot deeply understand these principles, these arithmetic archetypes, you will never perform correctly a "polytechnicien" job... you will never obtain a teaching position at "Ecole Polytechnique". For example, what is $3.755.918.261$ multiplied by $5.162.303.508$?

Student (very quickly) the result is 193891900145...

Teacher (very astonished) yes ... the product is really... But, how have you computed it, if you do not know the principles of arithmetic reasoning?

Student: it is simple: I have learned by heart all possible results of all possible different multiplications.

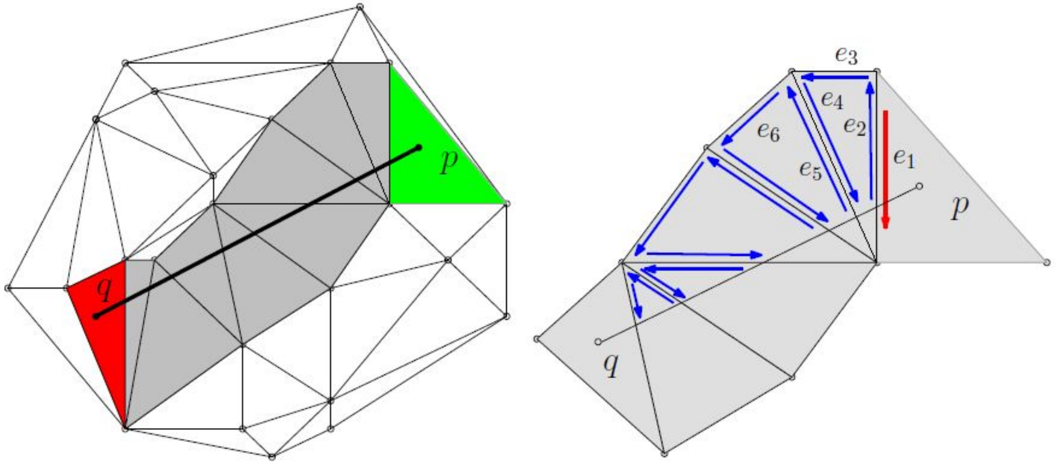
General idea (literary digression)

one-act theatre play
(La leçon, Eugène Ionesco, 1951)

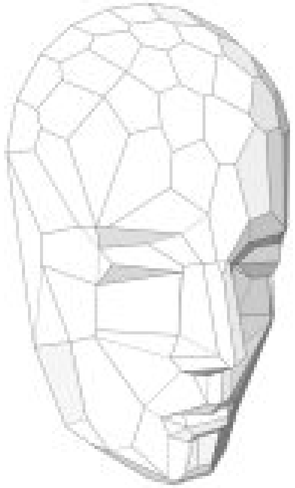
"La leçon" is played every night (since 1957) in Paris

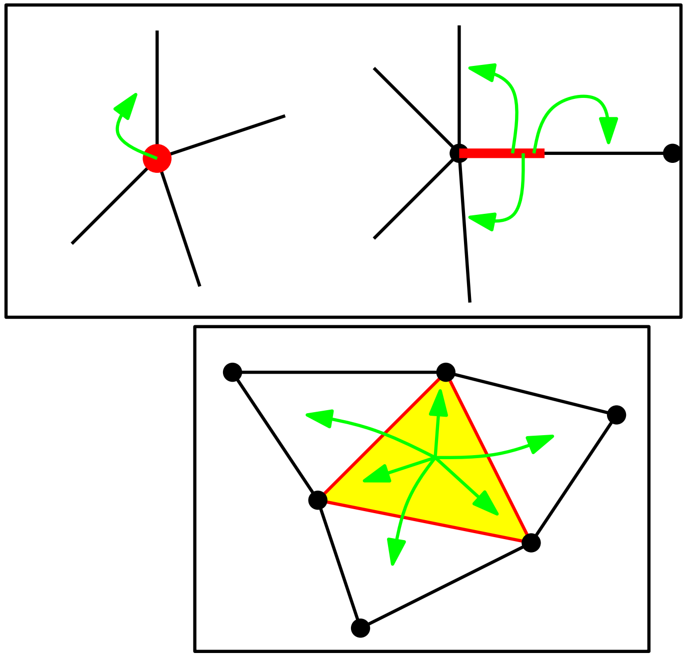
at the "Theatre de la Huchette" (8pm)

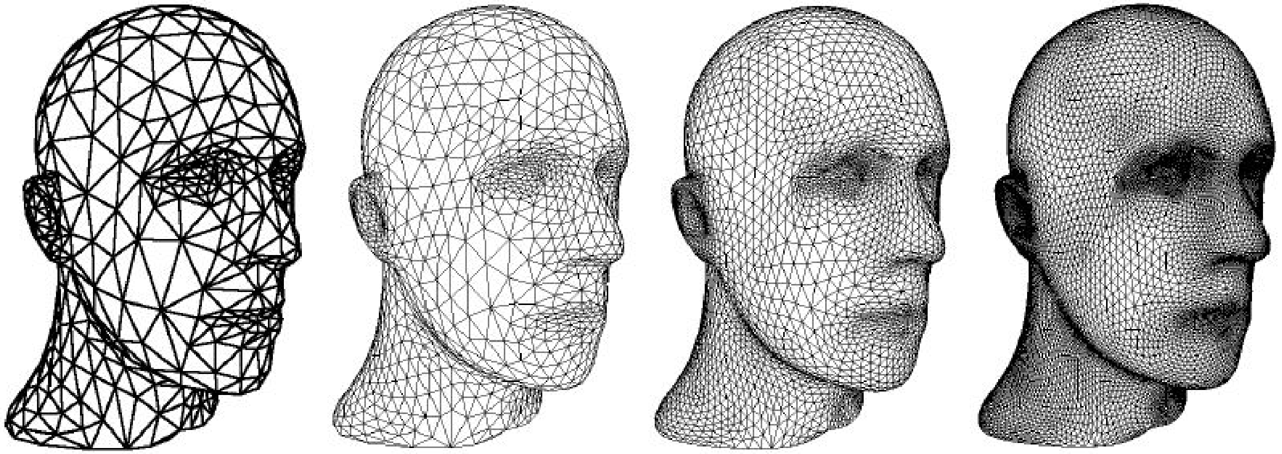
Data structures: navigation queries and dynamic updates



Geometric data structures

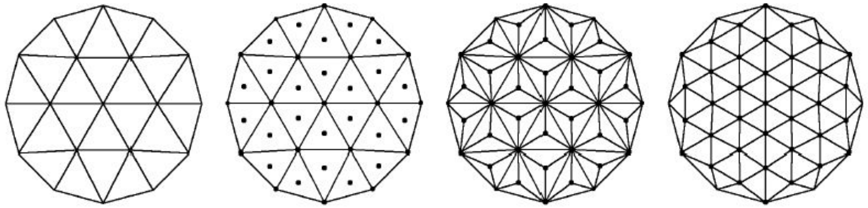






$\sqrt{3}$-subdivision (L. Kobbelt)

vertex insertions
+ edge flips



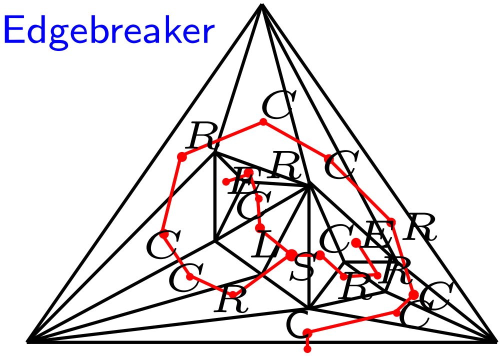
Mesh compression/Graph encoding

Mesh compression

Taubin et al. ('98)

Rossignac ('99)
Lopes et al. ('03)
Lewiner et al. ('04)
. . . . (many many others)

Edgebreaker



Valence (degree)

Touma and Gotsman ('98)
Alliez and Debrun
Isenburg
Khodakovsky
. . . . (many others)

Cut-border machine

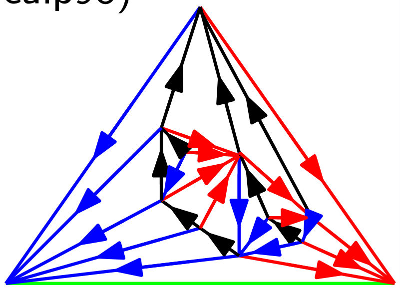
Gumhold et al. (Siggraph '98)
Gumhold (Soda '05)

Graph theory / combinatorics

Spanning tree-based schemes

Turan ('84)

Keeler Westbrook ('95)
He et al. ('99)
Chuang et al. (Icalp98)



Optimal encodings

Poulalhon Schaeffer (Icalp03)
planar triangle meshes

Fusy Poulalhon Schaeffer (Soda05)
planar polygonal meshes
Fusy (GD05)
4-connected triangulations

Castelli-Aleardi Fusy Lewiner (SoCG08)

Castelli-Aleardi Fusy Lewiner (CCCG10)

genus g meshes, with boundaries
triangular and quadrangular meshes

Compact data structures

Succinct representations (theoretical results)

Jacobson (Focs89)

Munro Raman (Focs97)

Chiang et al. (Soda01)
Blandford Blelloch (Soda03)
Blandford et al. (Alenex'04, IMR'03)
Nakano et al. (2008)

Farzan Munro (ESA 2008)

Blelloch Farzan (CPM 2010)

Castelli-Aleardi Devillers Schaeffer
(Wads05, CCCG05, SoCG06)

Barbay Castelli-Aleardi He Munro (Isaac07)

Practical compact data structures (with efficient implementations)

Directed Edges (Campagna et al. (1999))
Star Vertices (Kalmann et al. (2002))

SOT (Gurung Rossignac (SPM 2009))
SQUAD (Gurung Laney Lindstrom Rossignac, EG'11)
LR (Gurung et al. (Siggraph'11))

Castelli Aleardi Devillers Mebarki (CCCG06)
Castelli Aleardi Devillers (Isaac 2011)
Castelli Aleardi Devillers Rossignac (Sibgrapi 2012)

Existing works and our new results

Popular mesh data structures space requirements

Triangle-based data structure (CGAL)

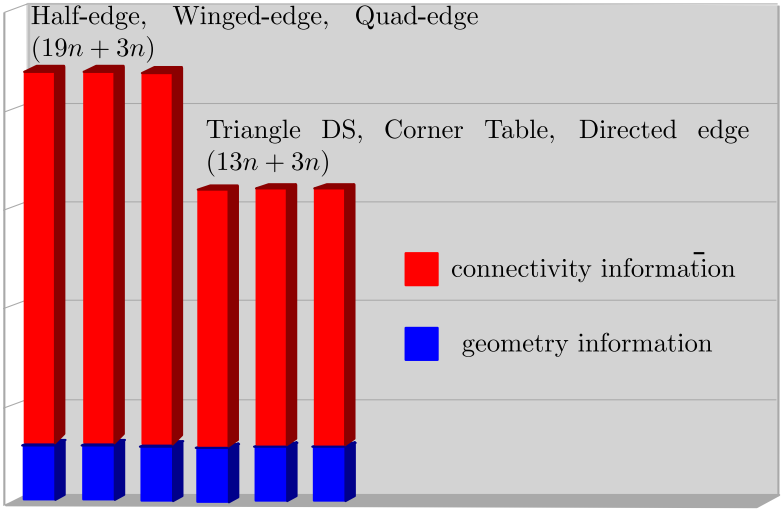
$$(3+3)\times f + n = 6\times 2n + n = 13n$$

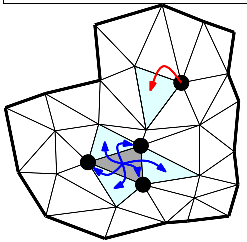
```
class Triangle{
  Triangle t1, t2, t3;
  Vertex v1, v2, v3;
}
 class Vertex{
  Triangle  root;
  Point p;
}
```

connectivity

Size (number of references)

```
class Point{
  float x;
  float y;
  float z;
}
```





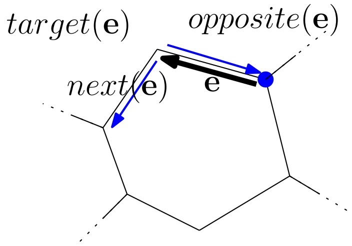
Half-edge

$$3\times 2e + n = 18n + n$$

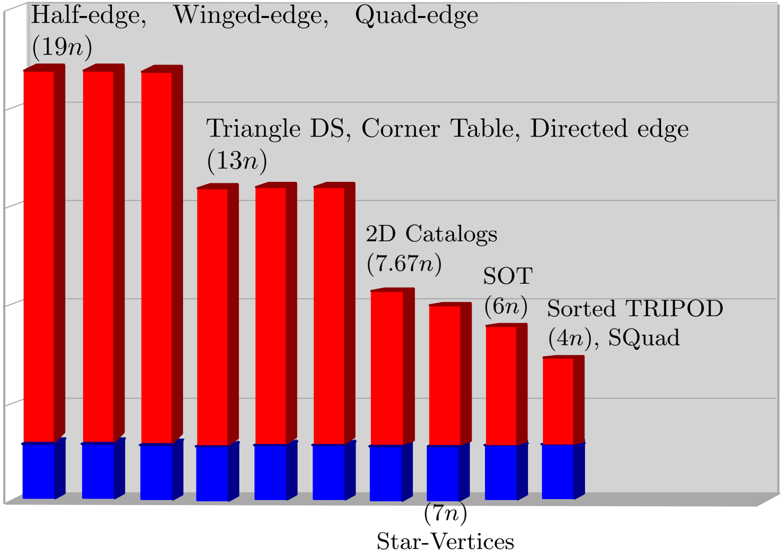
```
class Point{
  float x;
  float y;
  float z;
}
```

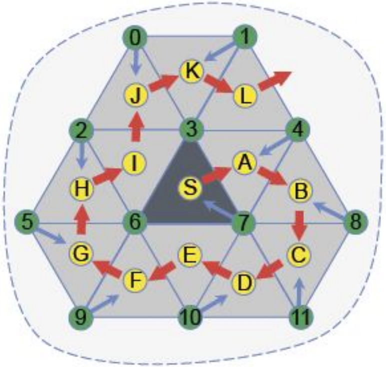
```
class Halfedge{
  Halfedge next, opposite;
  Vertex source;
}
 class Vertex{
  Halfedge  e;
  Point p;
}
```

connectivity



Compact representations: existing solutions space requirements



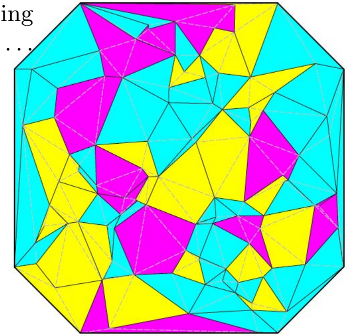


perform face re-ordering

(as in SOT, SQuad, LR and Sorted TRIPOD)

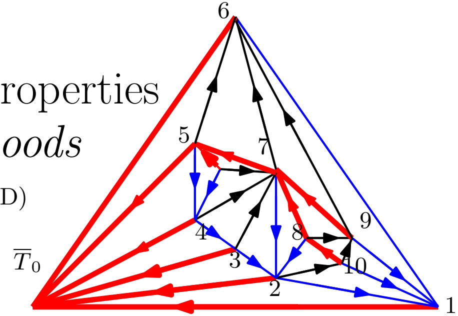
perform regrouping of neighboring triangles into quads, pentagons , . . .

(as in 2D Catalogs, SQuad)



use Combinatorial properties such as *Schnyder woods*

(as in TRIPOD and Sorted TRIPOD)



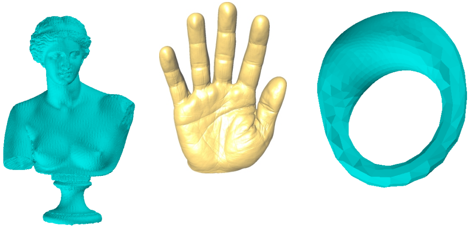
Mesh data structures: existing works

	Data Structure	memory size	navigation time	vertex access	dynamic
Traversable and modifiable (not space-efficient)	Edge-based data structures (Half-edge, Quad-edge, Winged-edge)	$18n + n$	$O(1)$	$O(1)$	*yes*
	Triangle based DS / Corner Table	$12n + n$	$O(1)$	$O(1)$	*yes*
	Directed edge (Campagna et al. '99)	$12n + n$	$O(1)$	$O(1)$	*yes*
Compact, traversable and modifiable	2D Catalogs (Castelli Aleardi et al., '06)	$7.67n$	$O(1)$	$O(1)$	*yes*
Compact and traversable (not modifiable)	Star vertices (Kallmann et al. '02)	$7n$	$O(d)$	$O(1)$	*no*
	TRIPOD (Snoeyink, Speckmann, '99)	$6n$	$O(1)$	$O(d)$	*no*
	SOT (Gurung et al. 2010)	$6n$	$O(1)$	$O(d)$	*no*
	Castelli-Aleardi and Devillers (2011)	$4n$	$O(1)$	$O(d)$	*no*
	SQUAD (Gurung et al. 2011)	$\approx (4+\varepsilon)n$	$O(1)$	$O(d)$	*no*
	LR (Gurung et al. 2011)	$\approx (2+\delta)n$	$O(1)$	$O(d)$	*no*
	no theoretical guarantees (experimental benchmark)	$\varepsilon \approx 0.09$ $\delta \approx 0.08$			

memory requirements: we count the number of references per vertex

Mesh data structures: our new results

	Data Structure	memory size	navigation time	vertex access	dynamic
Traversable and modifiable (not space-efficient)	Edge-based data structures (Half-edge, Quad-edge, Winged-edge)	$18n + n$	$O(1)$	$O(1)$	*yes*
	Triangle based DS / Corner Table	$12n + n$	$O(1)$	$O(1)$	*yes*
	Directed edge (Campagna et al. '99)	$12n + n$	$O(1)$	$O(1)$	*yes*
Compact, traversable and modifiable	2D Catalogs (Castelli Aleardi et al., '06)	$7.67n$	$O(1)$	$O(1)$	*yes*
Compact and traversable (not modifiable)	Star vertices (Kallmann et al. '02)	$7n$	$O(d)$	$O(1)$	*no*
	TRIPOD (Snoeyink, Speckmann, '99)	$6n$	$O(1)$	$O(d)$	*no*
	SOT (Gurung et al. 2010)	$6n$	$O(1)$	$O(d)$	*no*
	Castelli-Aleardi and Devillers (2011)	$4n$	$O(1)$	$O(d)$	*no*
	SQUAD (Gurung et al. 2011)	$\approx (4+\varepsilon)n$	$O(1)$	$O(d)$	*no*
	LR (Gurung et al. 2011) no theoretical guarantees (experimental benchmark)	$\approx (2+\delta)n$ $\varepsilon \approx 0.09$ $\delta \approx 0.08$	$O(1)$	$O(d)$	*no*
Compact, traversable and modifiable	Castelli-Aleardi Devillers Rossignac (Sibgrapi 2012)	$4.8n$	$O(1)$	$O(d)$	YES



Our results are provided with theoretical guarantees and experimental evaluation

We have tested our implementations on 3D models
we are only about 1.6 times slower than non compact data structures

Let’s start by revising a popular data structure for triangle meshes

Triangle based DS (used in CGAL): description

```
class Point{
  float x;
  float y;
  float z;
}
```

```
class Triangle{
  Triangle t1, t2, t3;
  Vertex v1, v2, v3;
}
 class Vertex{
  Triangle  root;
  Point p;
}
```

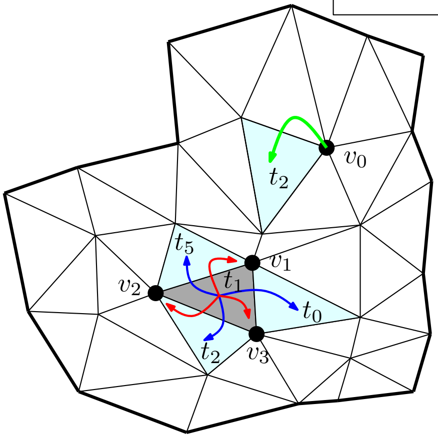
connectivity

for each triangle, store:

- 3 references to neighboring faces
- 3 references to incident vertices

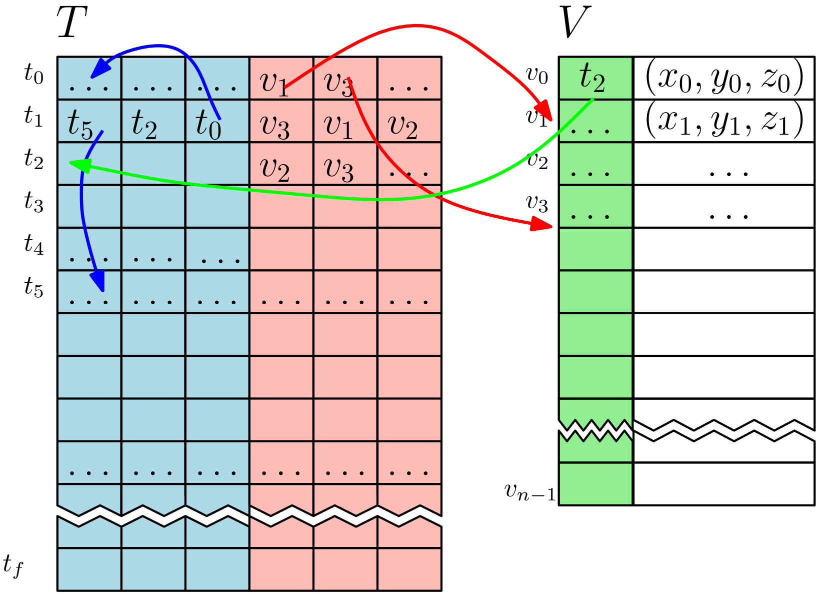
for each vertex, store:

- 1 reference to an incident face



$$(3+3)\times f + n = 6\times 2n + n = 13n$$

Size (number of references)



Triangle based DS (used in CGAL): mesh traversal

```
class Point{
  float x;
  float y;
  float z;
}
```

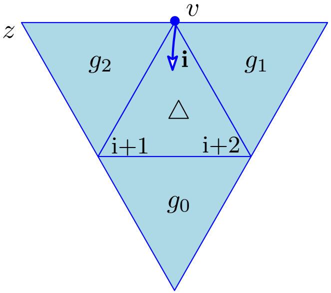
```
class Triangle{
  Triangle t1, t2, t3;
  Vertex v1, v2, v3;
}
 class Vertex{
  Triangle  root;
  Point p;
}
```

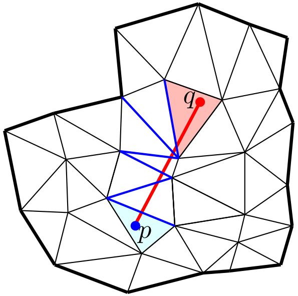
connectivity

the data structure supports the following operators

$v = \texttt{vertex}(\triangle, i)$
$\triangle = \texttt{face}(v)$
$i = \texttt{vertexIndex}(v, \triangle)$
$g_0 = \texttt{neighbor}(\triangle, i)$
$g_1 = \texttt{neighbor}(\triangle, ccw(i))$
$g_2 = \texttt{neighbor}(\triangle, cw(i))$
$z = \texttt{vertex}(g_2, \texttt{faceIndex}(g_2, \triangle))$

```
int cw(int i) {return (i + 2)%3; }
int ccw(int i) {return (i + 1)%3; }
```





we can locate a point, by performing a walk in the triangulation

```
int degree(int v) {
  int d = 1;
  int f = face(v);
  int g = neighbor(f, cw(vertexIndex(v, f)));
  while (g != f) {
    int next = neighbor(g, cw(faceIndex(f, g)));
    int i = faceIndex(g, next);
    g = next;
    d++;
  }
  return d;
}
```

we can turn around a vertex, by combining the operators above

Triangle based DS (used in CGAL): mesh traversal

```
class Point{
  float x;
  float y;
  float z;
}
```

```
class Triangle{
  Triangle t1, t2, t3;
  Vertex v1, v2, v3;
}
 class Vertex{
  Triangle  root;
  Point p;
}
```

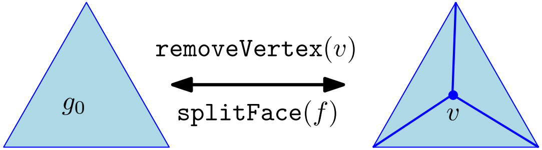
connectivity

the data structure supports the following operators

$\texttt{removeVertex}(v)$

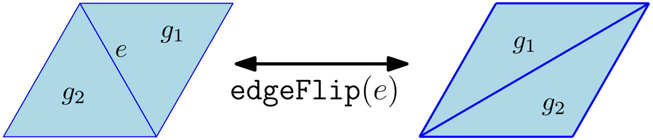
$\texttt{splitFace}(f)$

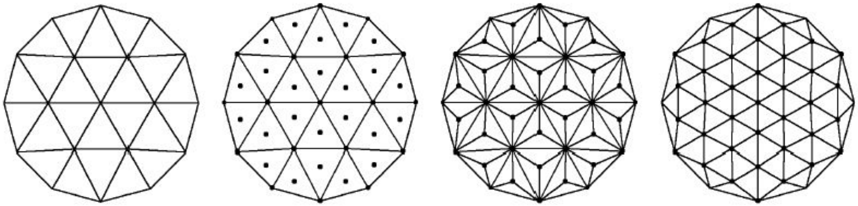
$\texttt{edgeFlip}(e)$

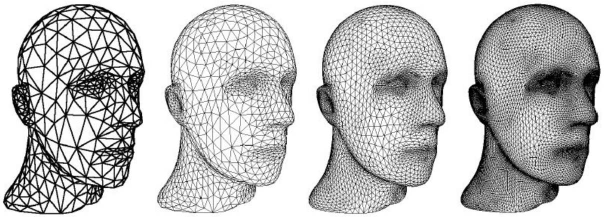


the data structure is **modifiable**

all these operators can be performed in $O(1)$ time





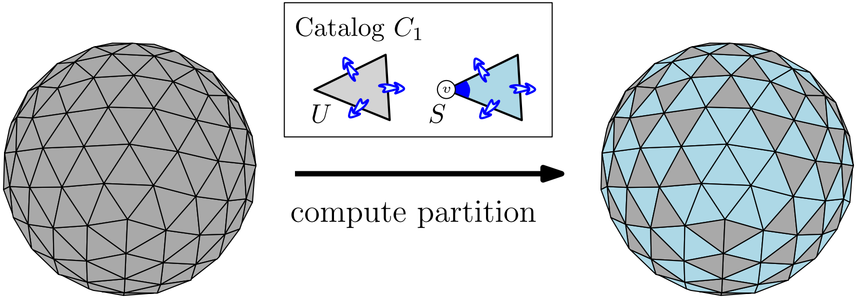


Construction and description of the ESQ data structure

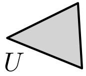
ESQ construction (preprocessing phase)

define a **patch catalog**

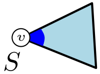
partition triangle faces into patches



Catalog C_1: the smallest catalog
only 2 patches



one triangle, with **no matched** vertex



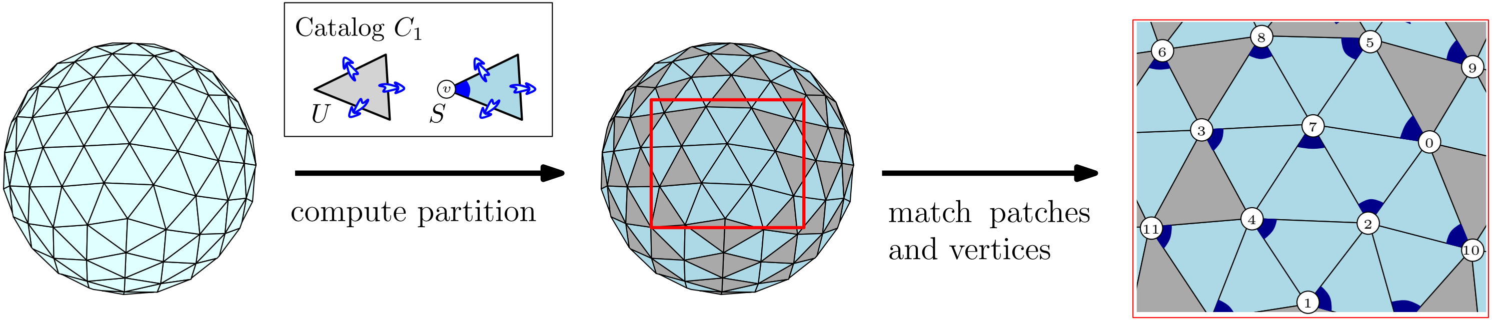
one triangle, with **one** matched vertex

ESQ construction (preprocessing phase)

define a patch catalog

partition triangle faces into patches

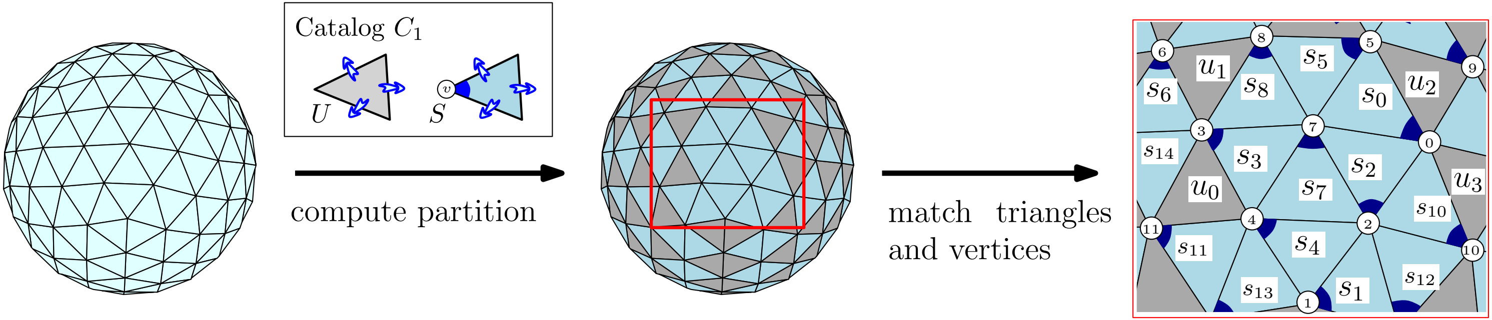
compute a **matching** triangles/vertices



ESQ construction (preprocessing phase)

define a patch catalog
partition triangle faces into patches
compute a matching triangles/vertices

re-order triangles according to the matching



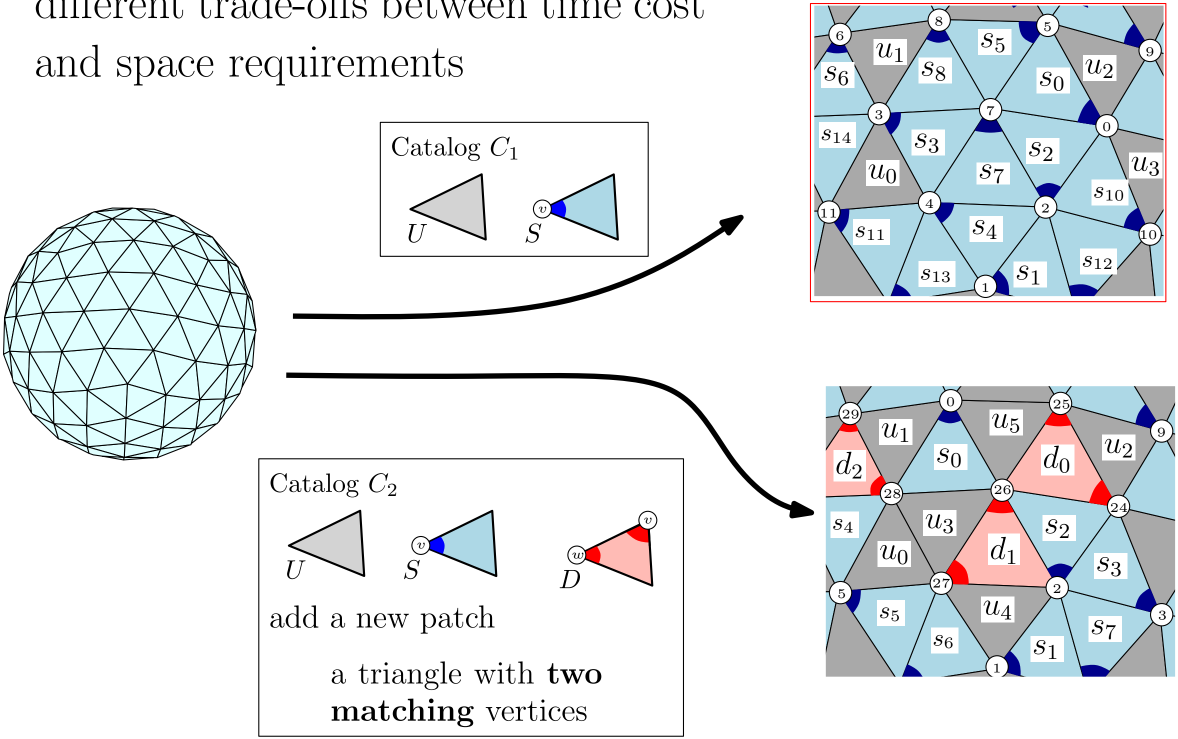
compute partition

match triangles and vertices

patches (and thus triangles) are re-ordered according to the matched vertex number

ESQ construction (preprocessing phase)

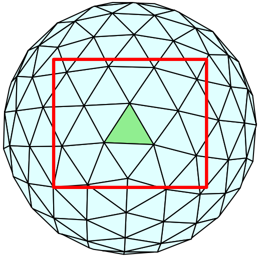
choosing a different catalog provides different trade-offs between time cost and space requirements

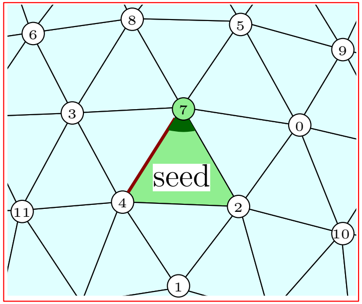


add a new patch

a triangle with **two matching** vertices

Matching phase: perform a DFS traversal

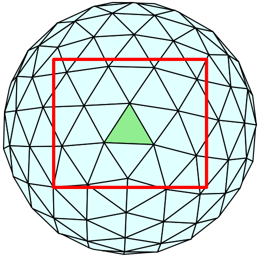


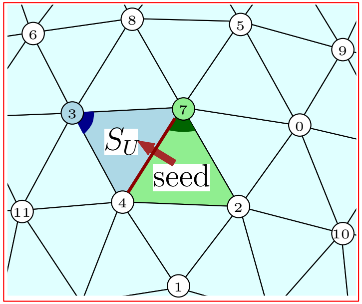


start the traversal choosing

a **seed** (green) face
a **gate** edge (red)

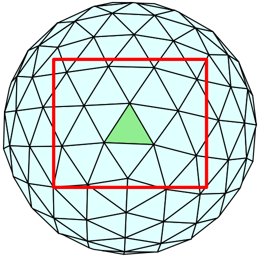
Matching phase: perform a DFS traversal

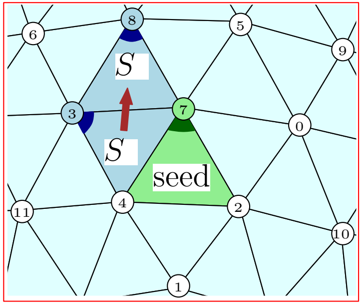




```
traverse unvisited triangles, rightmost
  if the opposite vertex in the visited triangle is un matched
      set the triangle as patch of type $S$
      match the vertex and the visited triangle
  otherwise
      set the triangle as patch of type $U$
```

Matching phase: perform a DFS traversal

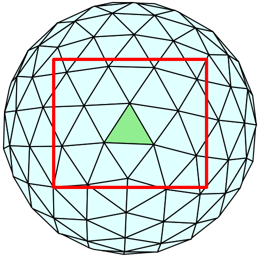


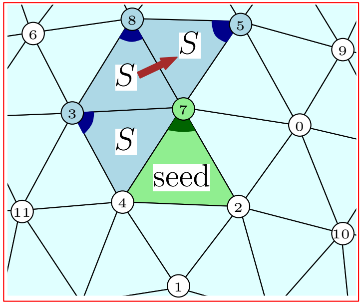


traverse unvisited triangles, rightmost

```
if the opposite vertex in the visited triangle is un matched
        set the triangle as patch of type $S$
        match the vertex and the visited triangle
otherwise
        set the triangle as patch of type $U$
```

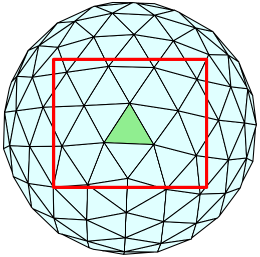
Matching phase: perform a DFS traversal

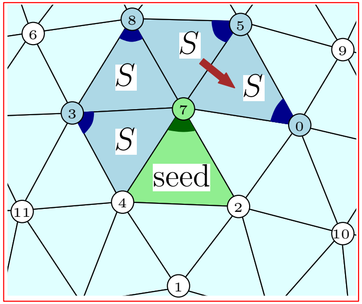




```
traverse unvisited triangles, rightmost
    if the opposite vertex in the visited triangle is un matched
            set the triangle as patch of type $S$
            match the vertex and the visited triangle
    otherwise
            set the triangle as patch of type $U$
```

Matching phase: perform a DFS traversal

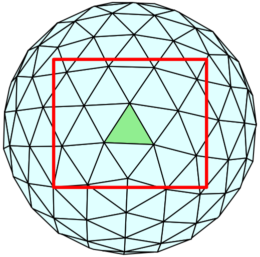


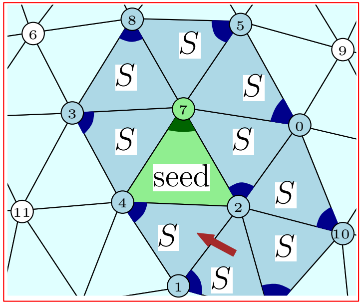


traverse unvisited triangles, rightmost

```
if the opposite vertex in the visited triangle is un matched
        set the triangle as patch of type $S$
        match the vertex and the visited triangle
otherwise
        set the triangle as patch of type $U$
```

Matching phase: perform a DFS traversal

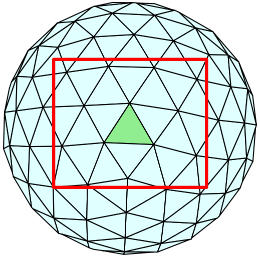


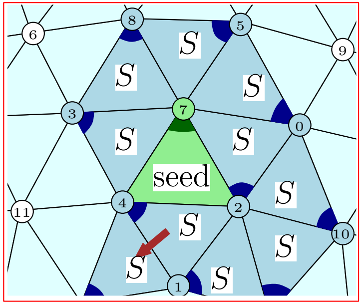


traverse unvisited triangles, rightmost

```
if the opposite vertex in the visited triangle is un matched
        set the triangle as patch of type $S$
        match the vertex and the visited triangle
otherwise
        set the triangle as patch of type $U$
```

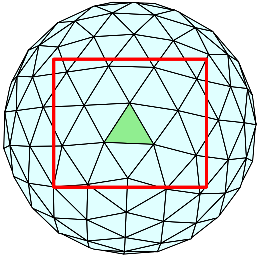
Matching phase: perform a DFS traversal

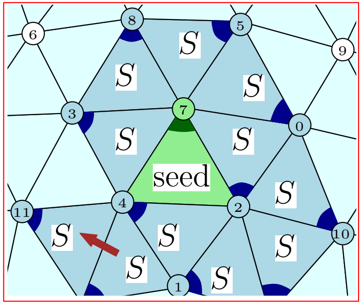




```
traverse unvisited triangles, rightmost
  if the opposite vertex in the visited triangle is un matched
        set the triangle as patch of type $S$
        match the vertex and the visited triangle
  otherwise
        set the triangle as patch of type $U$
```

Matching phase: perform a DFS traversal

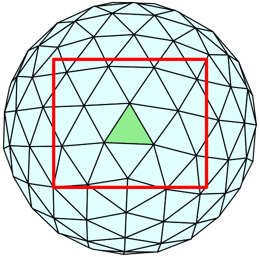


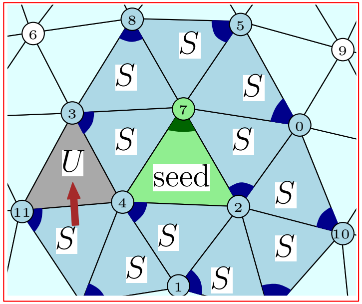


traverse unvisited triangles, rightmost

- if the opposite vertex in the visited triangle is un matched
    - set the triangle as patch of type S
    - match the vertex and the visited triangle
- otherwise
    - set the triangle as patch of type U

Matching phase: perform a DFS traversal

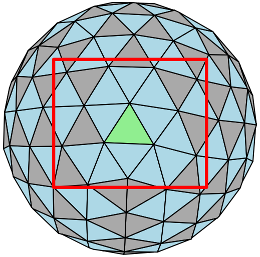


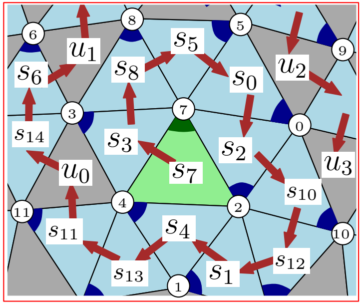


```
traverse unvisited triangles, rightmost
  if the opposite vertex in the visited triangle is un matched
        set the triangle as patch of type $S$
        match the vertex and the visited triangle
  otherwise
        set the triangle as patch of type $U$
```

Matching phase: perform a DFS traversal

recall that in a genus 0 triangulation with n vertices there are $2n-4$ triangles



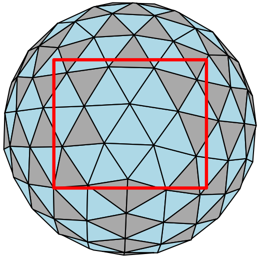


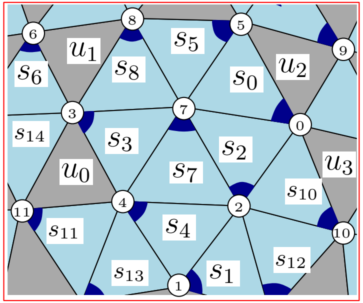
at the end, after $2n-4$ steps we have

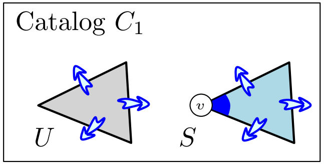
- all triangles are visited, all vertices are matched
- there are n patches of type S
- there are $n-4$ patches of type U

Description of the data structure

Description of the data structure: ESQ (catalog $\mathcal{C}_1$)

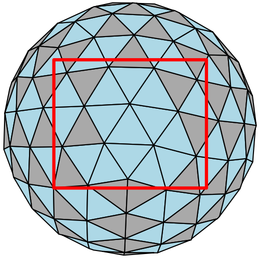


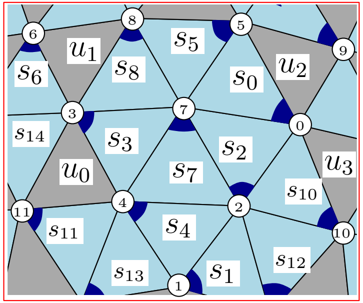


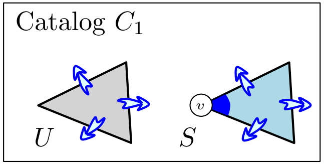


for each triangle, store 3 references to neighboring faces

Description of the data structure: ESQ (catalog $\mathcal{C}_1$)







for each triangle, store 3 references to neighboring faces

Store connectivity 2 tables

table T_S has size $3 \times n$

table T_U has size $3 \times (f - n)$

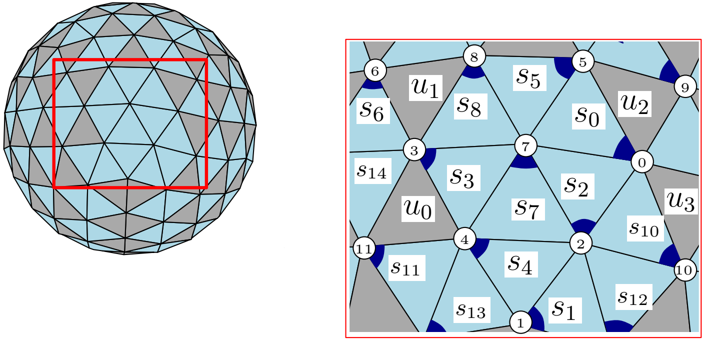
Use one more table for coordinates

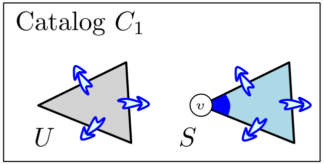
table P_S has size n

	T_S			P_S
s_0	s_5	s_2	u_2	(x_0, y_0, z_0)
s_1	s_{12}	s_4	...	(x_1, y_1, z_1)
s_2	s_0	s_7	s_{10}	(x_2, y_2, z_2)
s_3	s_7	s_8	u_0	...
s_4	s_1	s_{13}	s_7	...
s_5	s_8	s_0	...	...
s_6	s_{14}	u_1	...	...
s_7	s_4	s_2	s_3	...
s_8	s_3	s_5	u_1	...
s_9	...	u_2	...	...
s_{n-1}				...

T_U			
s_3	s_{14}	s_{11}	u_0
s_6	s_8	...	u_1
...	...	s_0	u_2
s_{10}	...	...	u_3

ESQ (catalog $\mathcal{C}_1$): space requirements





for each triangle, store 3 references to neighboring faces

Connectivity cost

$$3 \times f = 3 \cdot (2n - 4)$$

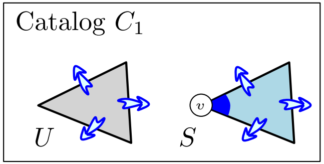
6 rpv (references per vertex)

	T_S			P_S
s_0	s_5	s_2	u_2	(x_0, y_0, z_0)
s_1	s_{12}	s_4	...	(x_1, y_1, z_1)
s_2	s_0	s_7	s_{10}	(x_2, y_2, z_2)
s_3	s_7	s_8	u_0	...
s_4	s_1	s_{13}	s_7	...
s_5	s_8	s_0	...	...
s_6	s_{14}	u_1	...	...
s_7	s_4	s_2	s_3	...
s_8	s_3	s_5	u_1	...
s_9	...	u_2	...	...
s_{n-1}				...

T_U			
s_3	s_{14}	s_{11}	u_0
s_6	s_8	...	u_1
...	...	s_0	u_2
s_{10}	...	...	u_3

ESQ (catalog $\mathcal{C}_1$): traversing the mesh

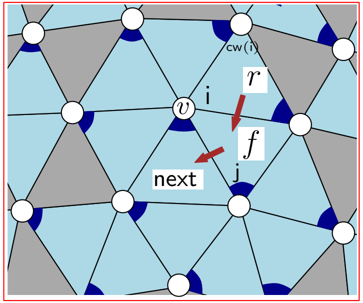
Drawback, as for previous compact representations (SOT, SQUAD, LR, Sorted Tripod)



```
int neighbor(int r, int i) {
    if (patchType(r) == S)
        return tableS[patchIndex(r) * 3 + i];
    else
        return tableU[patchIndex(r) * 3 + i];
}
```

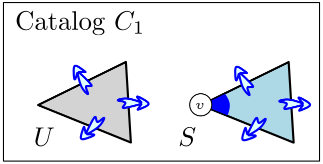
`vertex` operator is slightly slower: to retrieve the vertex incident to a given face r, we turn around until we find the type S patch matching the vertex

it takes $O(d)$ time, as in previous compact representations

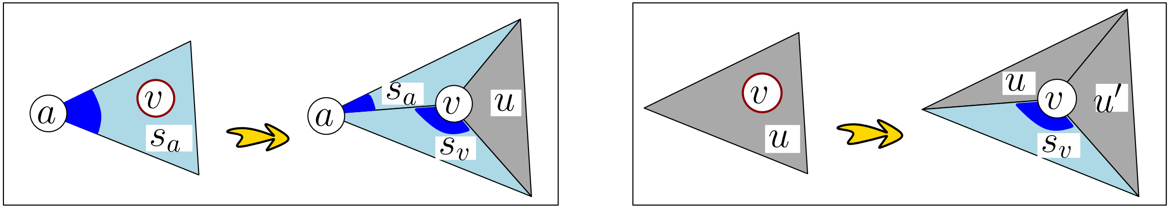


```
int vertex(int r, int i) {

    if (patchType(r) == S && i == 0)
        return patchIndex(r);
    int f = neighbor(r, cw(i));
    int j = faceIndex(r, f);
    while (f != r) {

        if (j == 1 && patchType(f) == S)

            return patchIndex(f);

        int next = neighbor(f, ccw(j));
        j = faceIndex(f, next);
        f = next;

    }
}
```

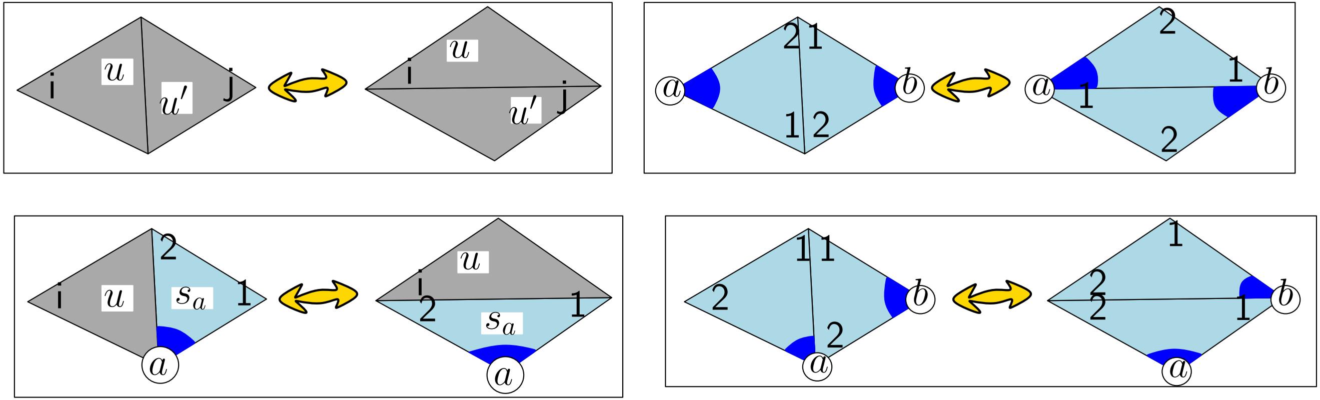
ESQ (catalog $\mathcal{C}_1$): performing updates



`edgeFlip` and `faceSplit` can be performed in $O(1)$ time
(only a constant nummber of references must to be updated)



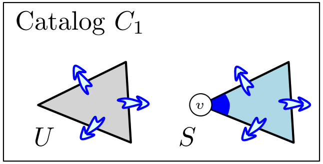
Triangle split



Edge-flip

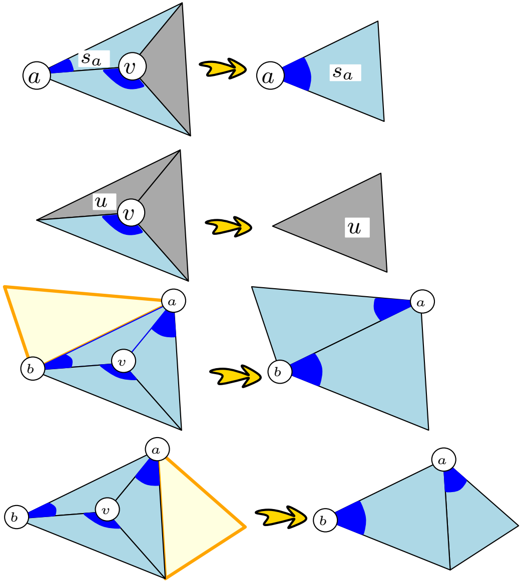
ESQ (catalog $\mathcal{C}_1$): performing updates

Catalog C_1

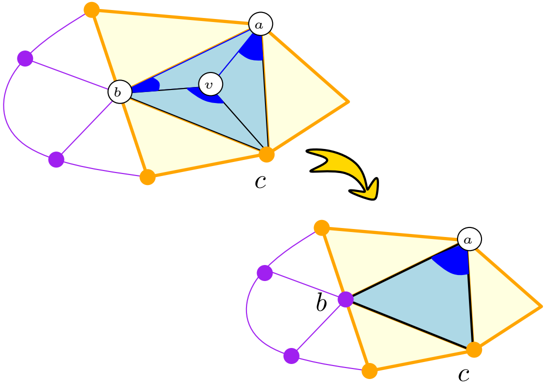


`deleteVertex` can be performed in $O(d)$ time
(we have to turn around a vertex)

4 easy cases: $O(1)$ time



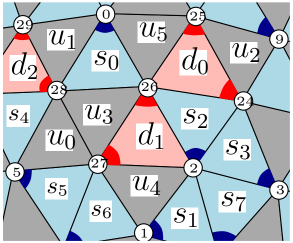
All 3 adjacent triangles are matched (type S): this case is more involved

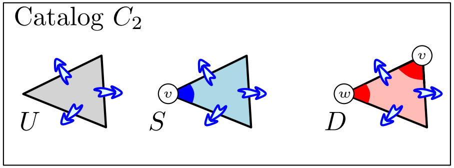


at least one violet triangle has no mark

We spend $O(d)$ time to find the free patch (unmatched)

ESQ (catalog $\mathcal{C}_2$): more efficient data structure





3 types of patches with at most 2 matched vertices

two more tables T_D and P_D

T_S

s_0	u_3	u_5	u_1
s_1	s_7	u_4	...
s_2	d_0	d_1	s_3
s_3	s_2	s_7	...
s_4	u_0	d_2	...
s_5	s_6	u_0	...
s_6	u_4	s_5	...
s_7	s_3	s_1	...
	...	...	...

P_S

(x_0, y_0, z_0)
(x_1, y_1, z_1)
(x_2, y_2, z_2)
...
...
...
...
...
...

T_U

u_3	s_4	s_5	u_0
s_0	...	d_2	u_1
d_0	...	...	u_2
d_1	s_0	u_0	u_3
s_1	d_1	s_6	u_4
d_0	...	s_0	u_5
...	...	...	
...	...	...	
...	...	...	

T_D

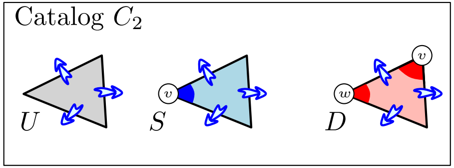
d_0	s_2	u_5	u_2
d_1	u_4	s_2	u_3
d_2	...	s_4	u_1
	...	...	...
	...	...	...
	...	...	...
	...	...	...

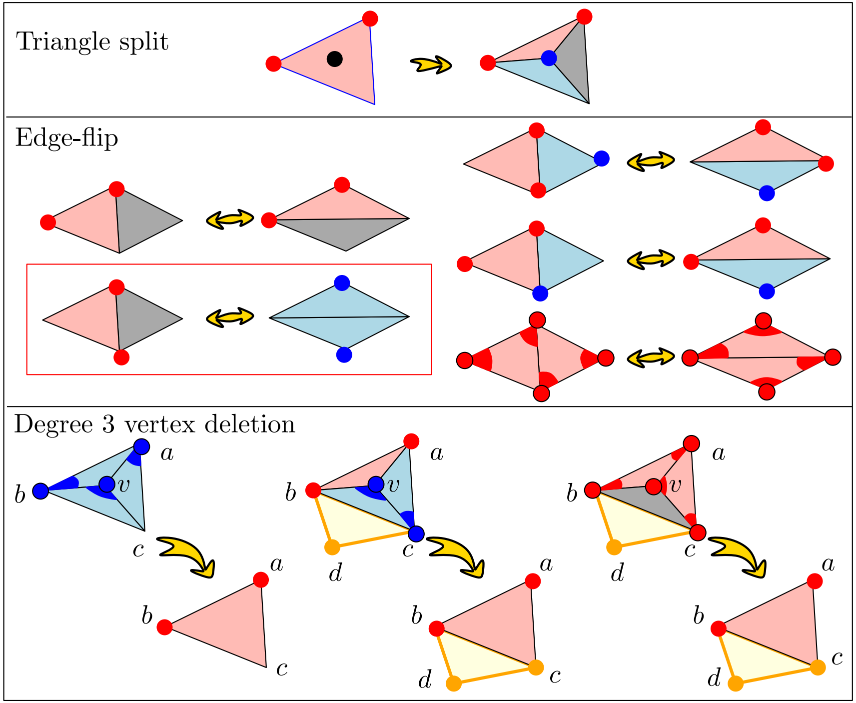
P_D

(x_{24}, y_{24}, z_{24})	(x_{25}, y_{25}, z_{25})
(x_{26}, y_{26}, z_{26})	(x_{27}, y_{27}, z_{27})
(x_{28}, y_{28}, z_{28})	(x_{29}, y_{29}, z_{29})
...	...
...	...
...	...
...	...

ESQ (catalog $\mathcal{C}_2$): all updates in $O(1)$ time

`deleteVertex` can now be performed in $O(1)$ time

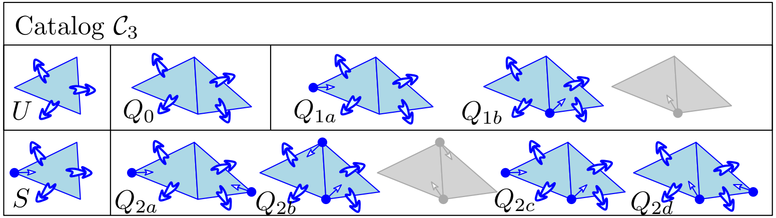




ESQ (catalog $\mathcal{C}_3$, with quads): more compact scheme

use a larger catalog: with quads

regroup pairs of neighboring triangles into **quads**



Two triangles merge

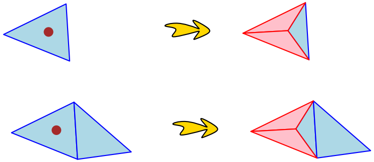
At least $\frac{3}{5}$ quads

At most $\frac{4}{5}$ triangles

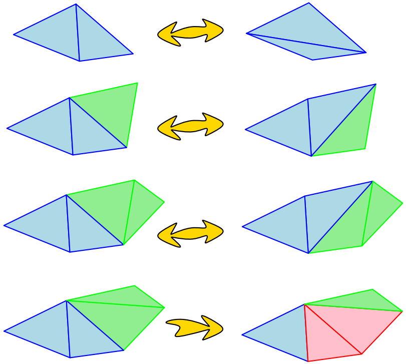
→ $\boxed{\mathbf{4.8n} \times 32 \text{ bits}}$ **Connectivity cost**

(counting argument using Euler's relation)

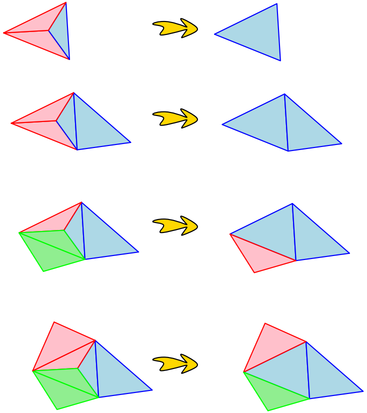
Triangle split



Edge-flip



vertex removal



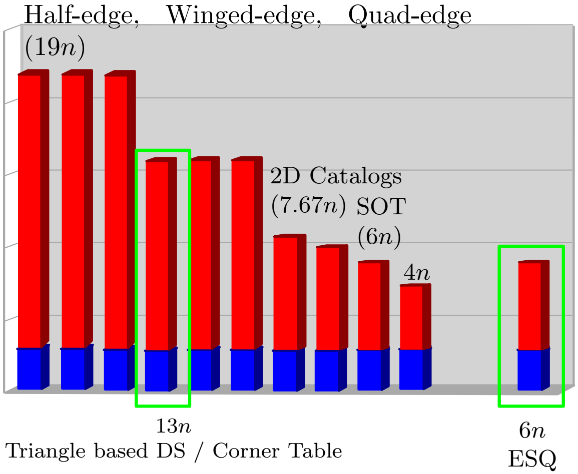
Experimental evaluation: our ESQ vs. Triangle based DS

3D model	statistics vertices	faces	genus
Bague	2652	5.3K	1
Aphrodite	46096	92K	0
Feline	49864	99K	2
Camille's hand	195557	391K	0
Eros	476596	953K	0
Pierre's hand	773465	1.54M	0

$6n \times 32 bits$ $13n \times 32$ bits

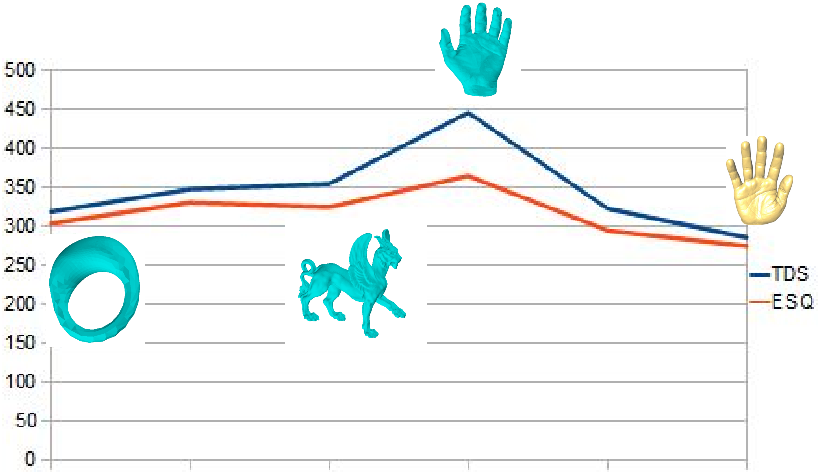
Navigation time

(nanoseconds per operation)



computation of vertex degree

ESQ is slightly faster than *TDS*



Experimental evaluation: our ESQ vs. Triangle based DS

	statistics		
3D model	vertices	faces	genus
Bague	2652	5.3K	1
Aphrodite	46096	92K	0
Feline	49864	99K	2
Camille's hand	195557	391K	0
Eros	476596	953K	0
Pierre's hand	773465	1.54M	0

$6n \times 32bits$

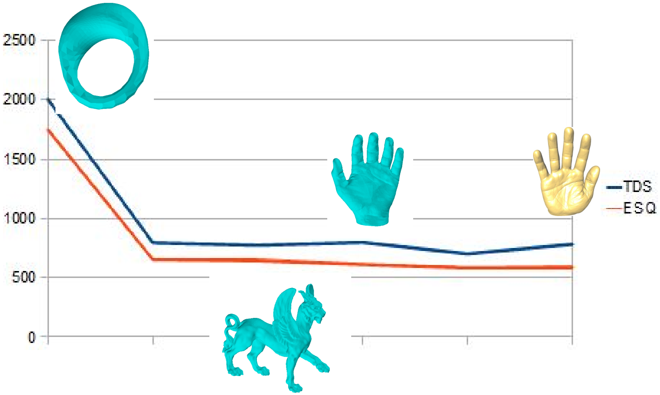
$13n \times 32$ bits

Update time

(nanoseconds per operation)

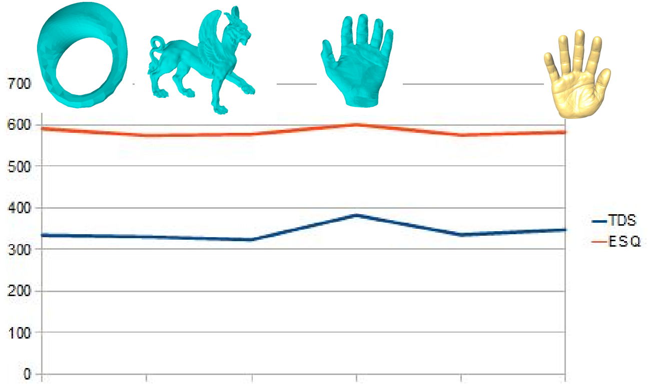
computation of face split

ESQ is slightly faster than *TDS*



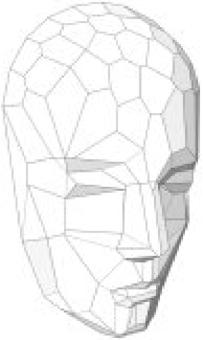
computation of edge flip

TDS is faster than *ESQ*

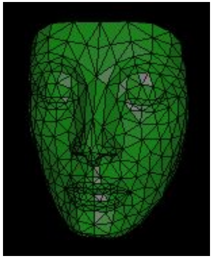


Concluding remarks: extensions and future work

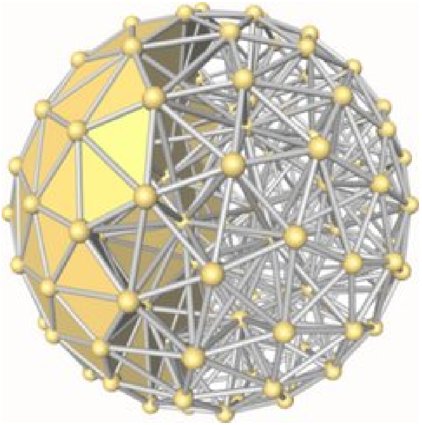
extension to polygonal meshes



Dealing with boundaries



Could our technique apply to higher dimensional complexes?
(3D triangulations)



Thanks