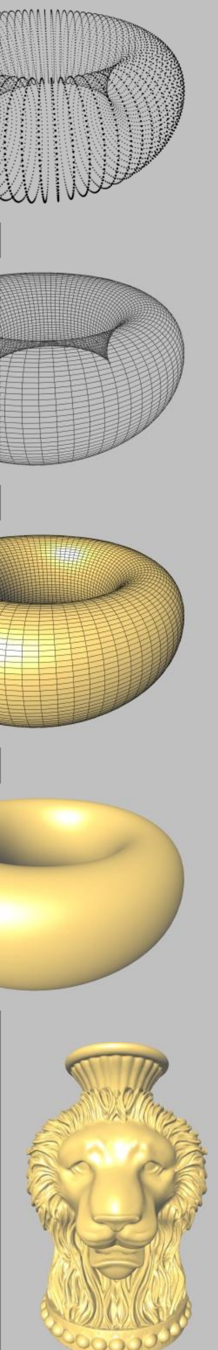
3D Polyhedral Surfaces in CGAL

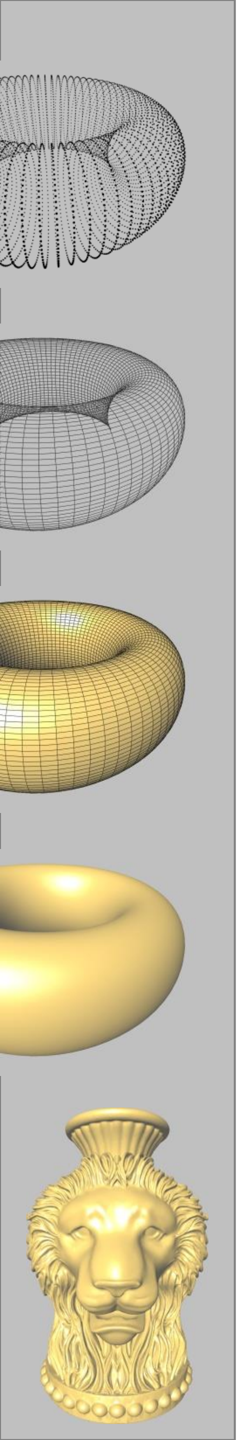
Pierre Alliez

http://www.cgal.org



Outline

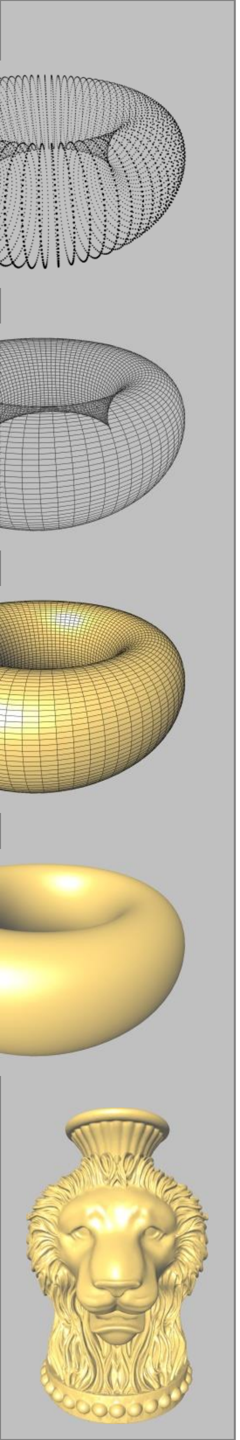
- Motivations
- Definition
- Halfedge Data Structure
- Traversal
- Euler Operators
- Customization
- Incremental Builder
- File I/O
- Examples
- Applications
- Exercises



Motivations

From **rendering**...

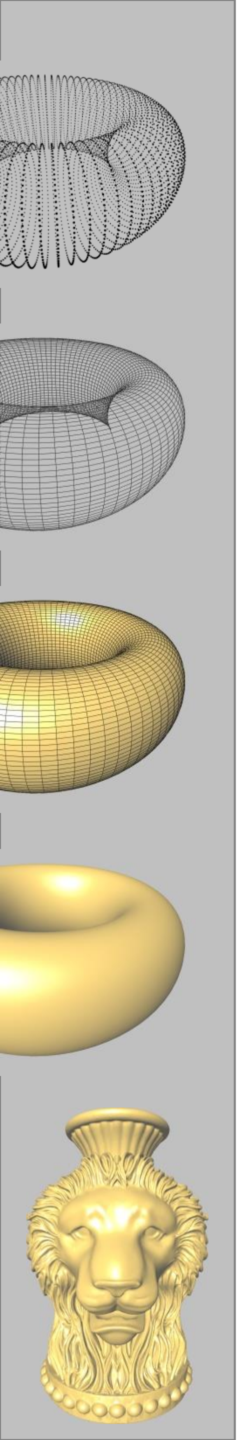
- Static
- Compact storage in array
- Traversal over all facets
- Attributes per facet/vertex



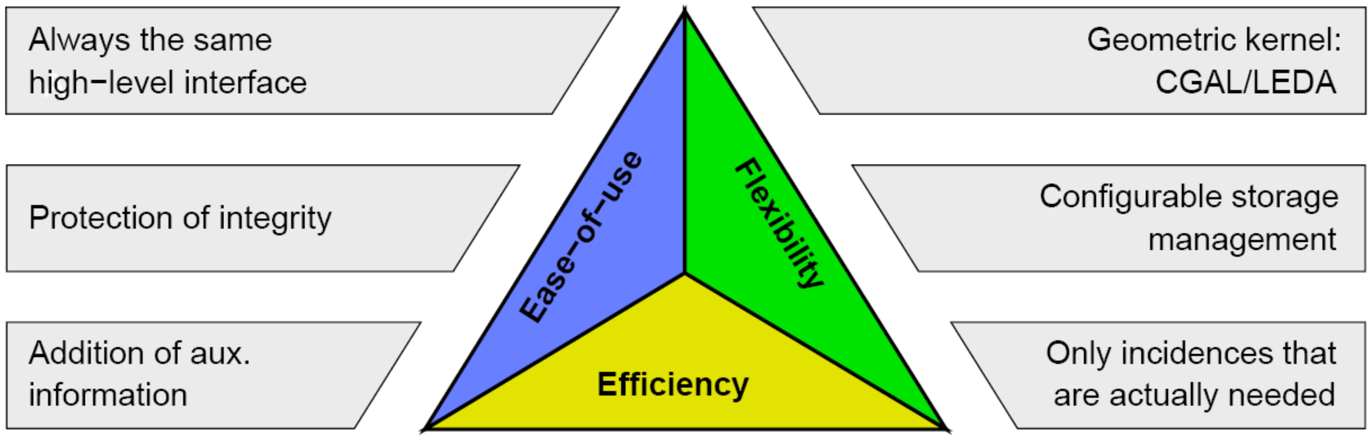
Motivations

...to **algorithms** on meshes

- Dynamic pointer updates
- Dynamic storage in lists
- Traversal over incidences

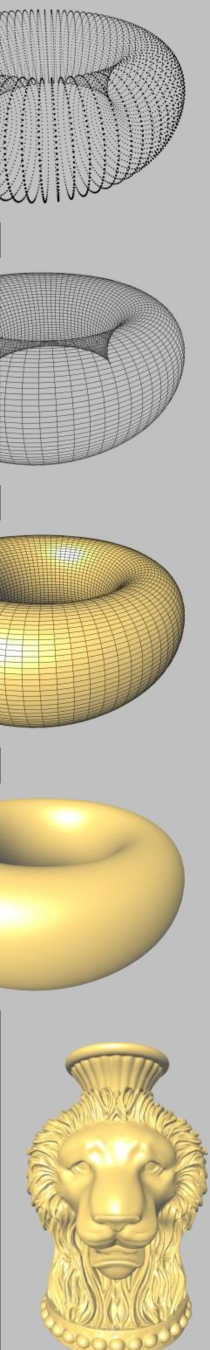


Design Goal



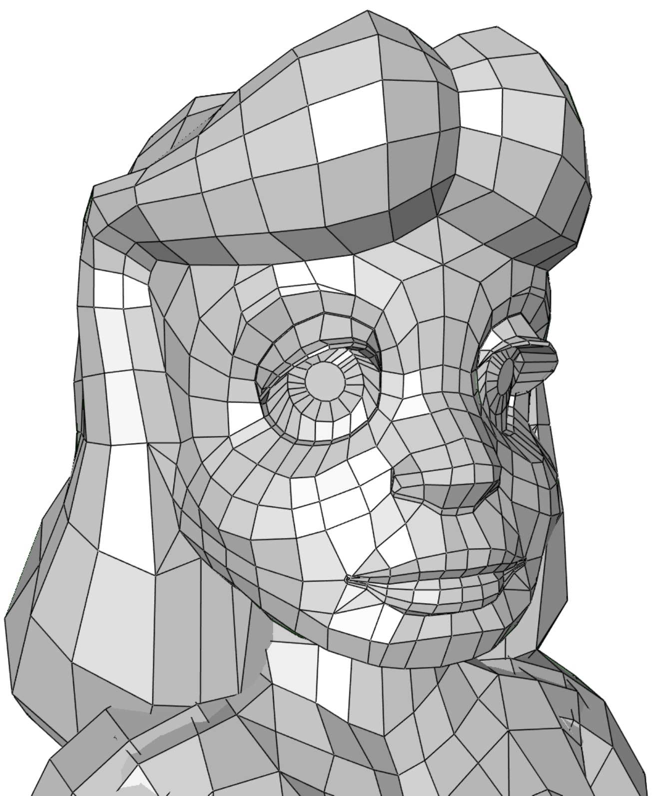
Method: paradigm of *Generic Programming*

Example: STL

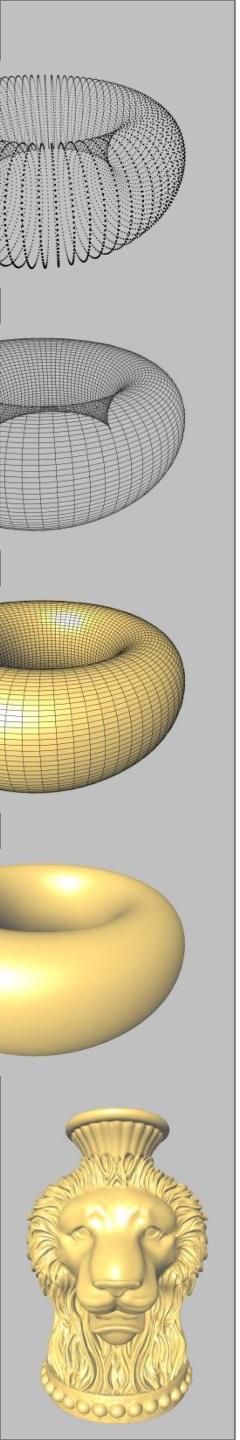


Definition

Polyhedral Surface: boundary representation of a polyhedron in IR^3.



ttp://www.cgal.org



Polyhedral Surface

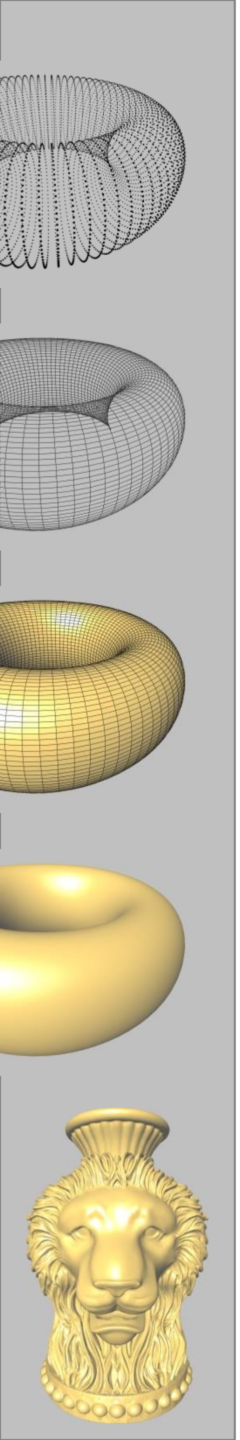
Represented by three sets **V**,**E**,**F** and an incidence relation on them, restricted to orientable 2-manifolds with boundary.

V = Vertices in IR^3

E = Edges, straight line segments.

F = Facets, simple, planar polygons without holes.

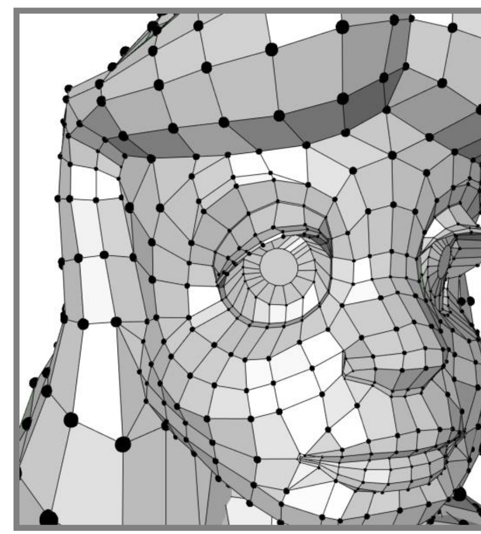
Can be extended: edges to curves, facets to curved surfaces.



Polyhedral Surface

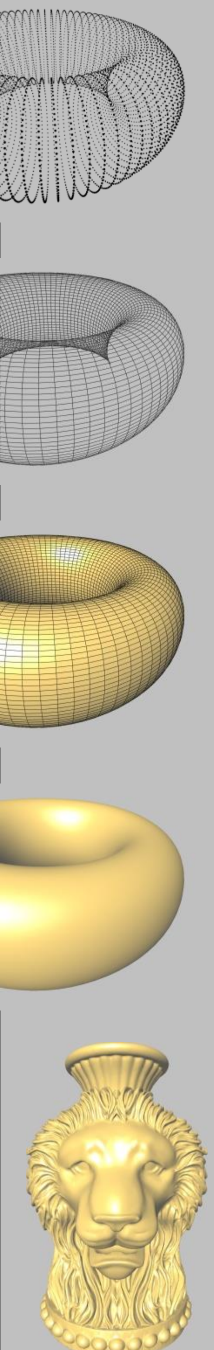
Represented by three sets **V**,**E**,**F** and an *incidence relation* on them, restricted to orientable 2-manifolds with boundary.

⇒ Edge-based data structure ?

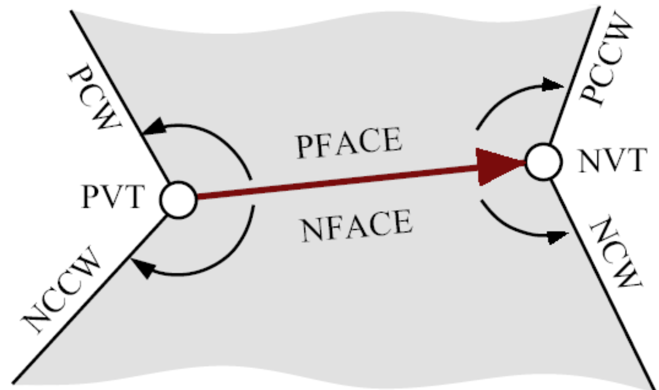


CGAL

Edge-centered Data Structures



Winged-Edge Data Structure



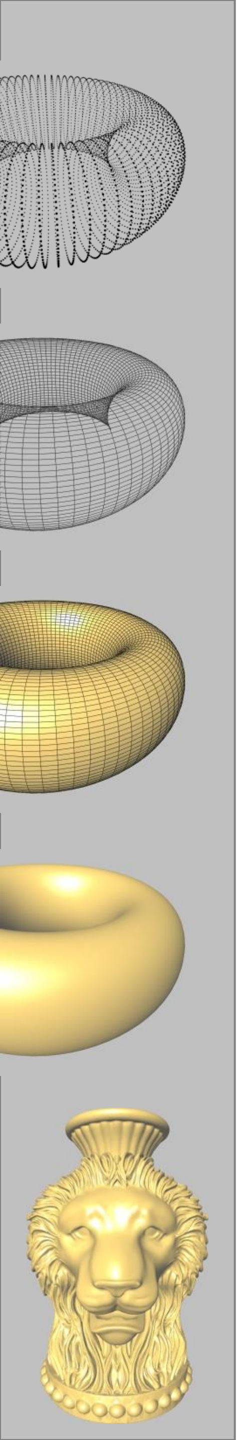
[Baumgart 75], DCEL [Muller & Preparata 78]

Edge size: 4-8 pointers per edge

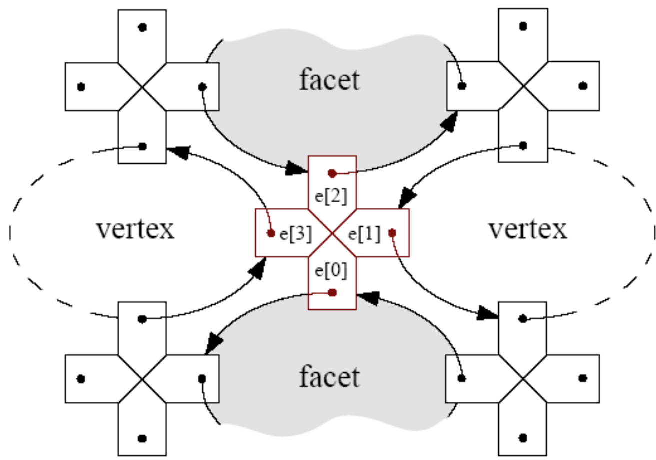
Ref. size: 2 pointers per reference

Efficiency:
```
succ_vertex( Edge e, Vertex v) :=
    if e.PVT = v then (e.PCW,v)
                 else (e.NCW,v)
```

Note: Branching!



Quad-Edge Data Structure



[Guibas & Stolfi 85]

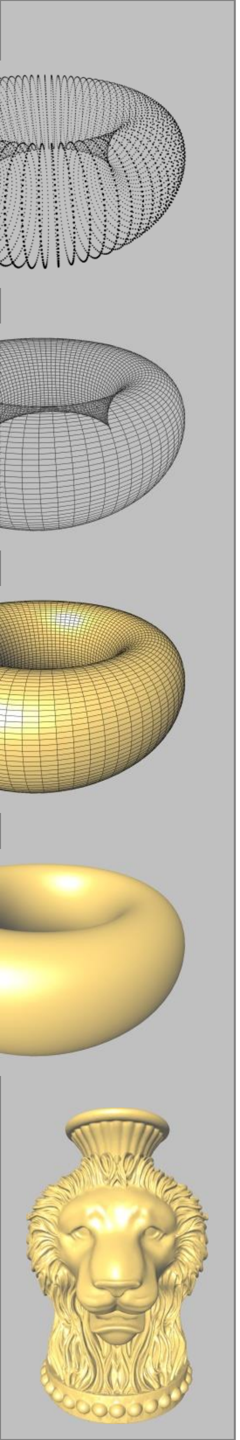
Edge size: 2-8 ptrs + 2-8 bits per edge

Ref. size: 1 pointer + 1-2 bits per reference

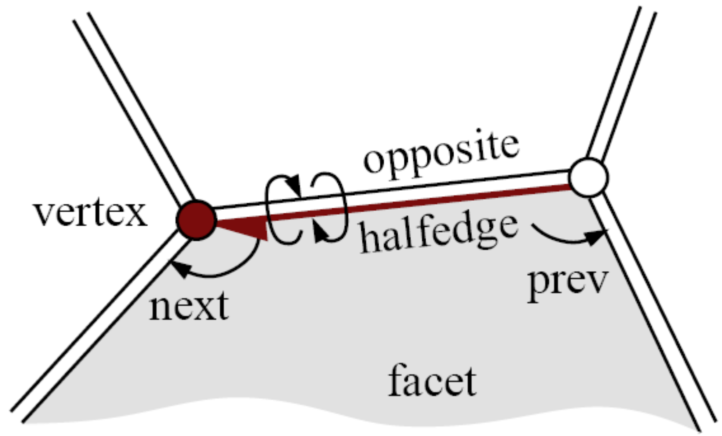
Efficiency:

```
succ_vertex(Edge e, int r) := e[r]
succ_facet( Edge e, int r) :=
            e[r+1 mod 4] - 1 mod 4
```

Note: Easy duality, but no type safety.
mod-op, bit fiddling, array access.



Halfedge Data Structure



[Weiler 85], [Mäntylä 88], DCEL [de Berg, van Krefeld, Overmars, Schwarzkopf 97]

Edge size: 4–10 pointers per edge

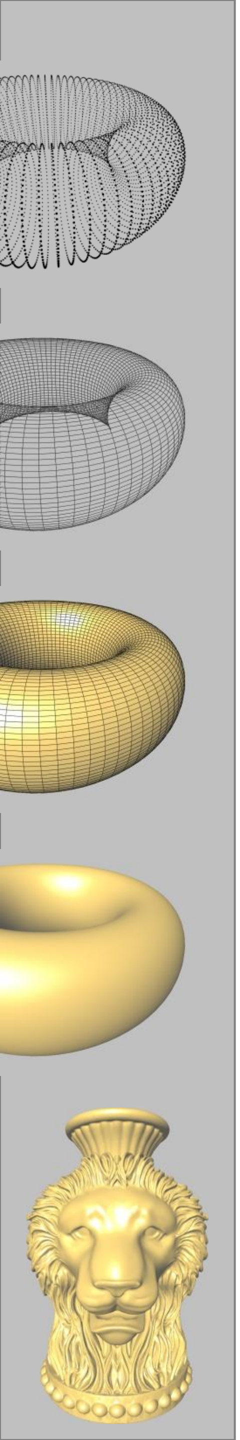
Ref. size: 1 pointer per reference

Efficiency:

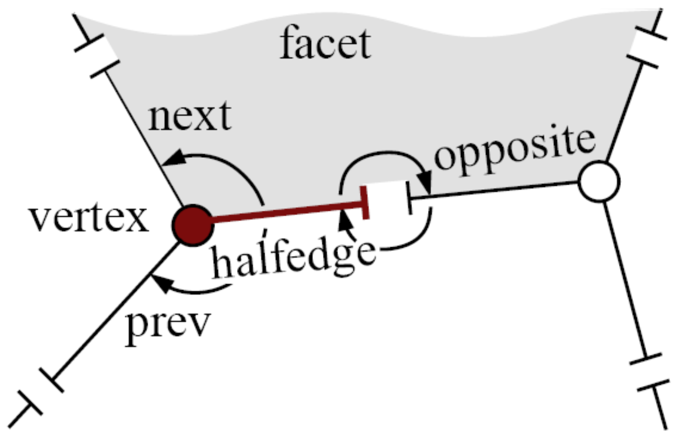
```
succ_vertex( Edge e) := e.opp.next
succ_facet(  Edge e) := e.next
```

Note: Encodes oriented facets.

http://www.cgal.org



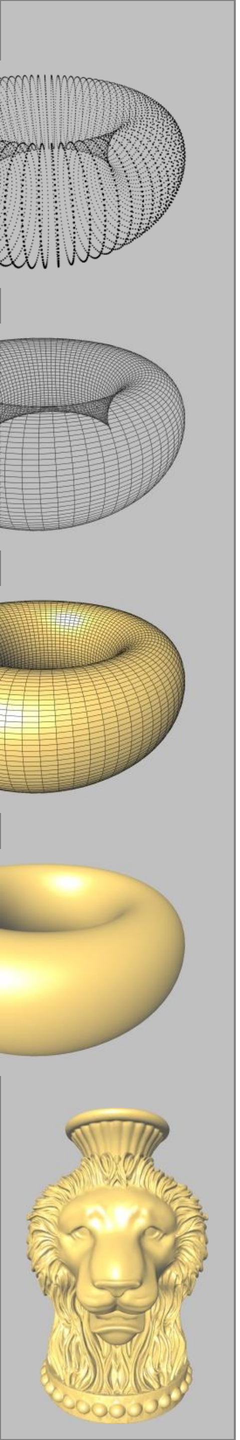
VE-Structure



[Weiler 85]

Edge size:	4-10 pointers per edge
Ref. size:	1 pointer per reference
Efficiency:	`succ_vertex( Edge e) := e.next` `succ_facet( Edge e) := e.next.opp`
Note:	Dual to halfedge.

http://www.cgal.org



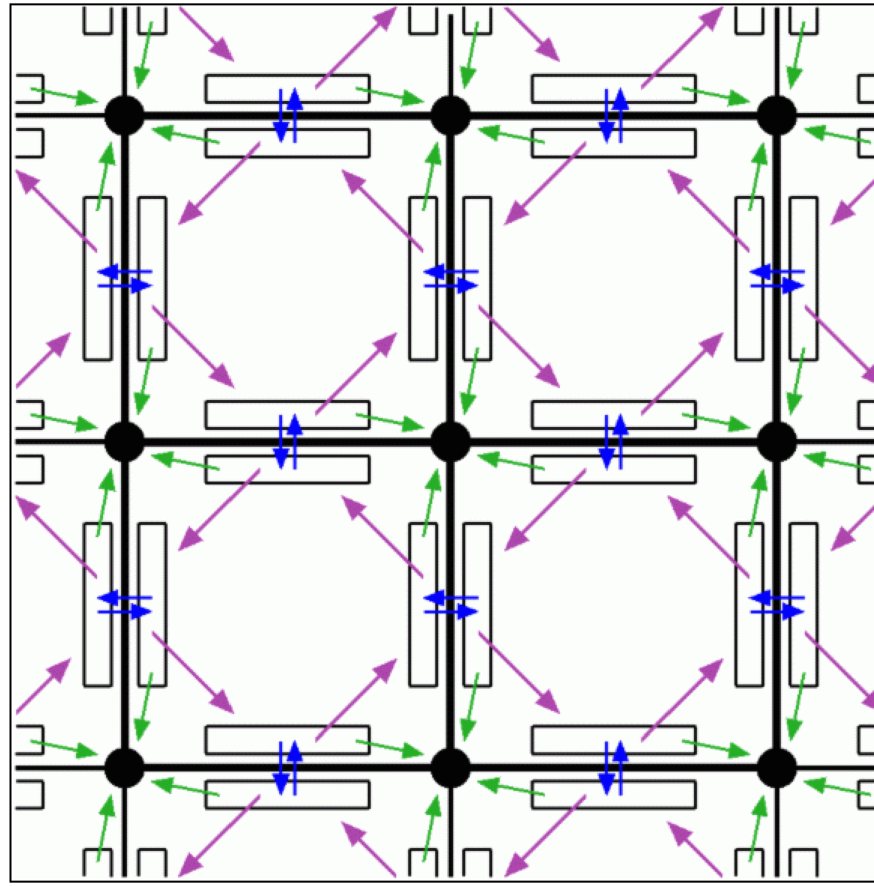
Halfedges

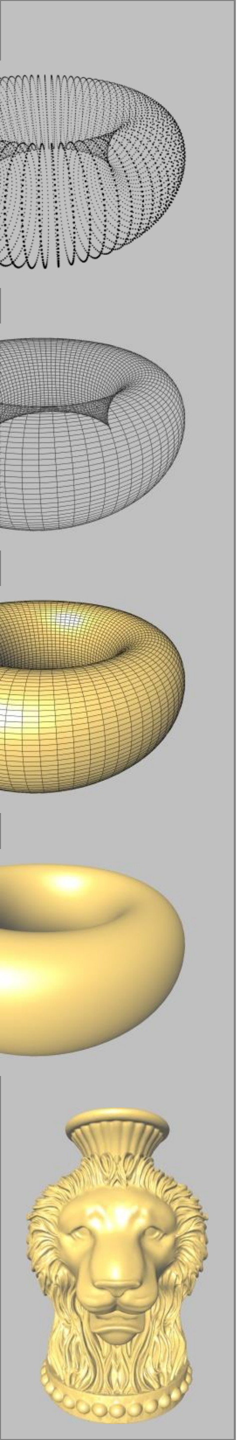
Require:

Oriented surface

Idea:

Consider 2/4 ways of accessing an edge





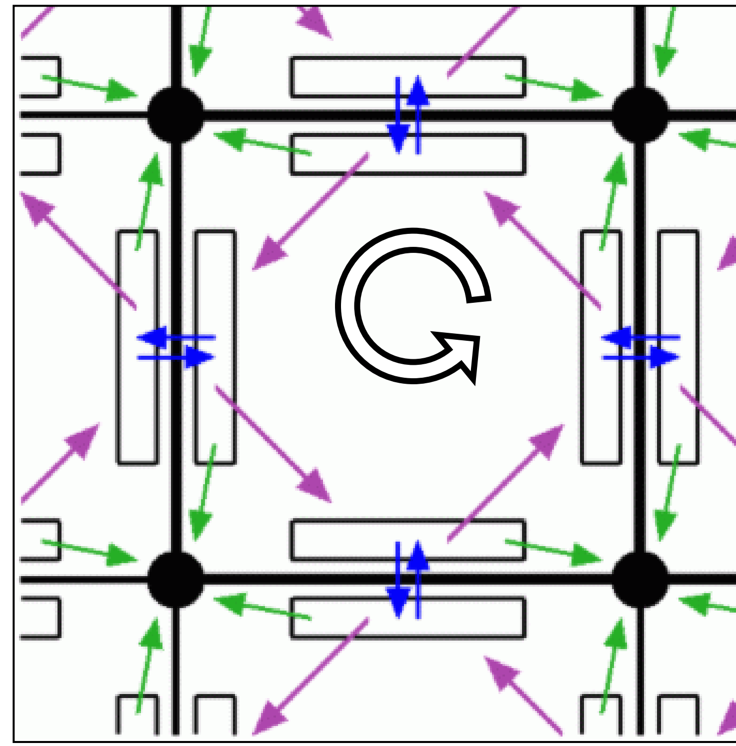
Halfedge

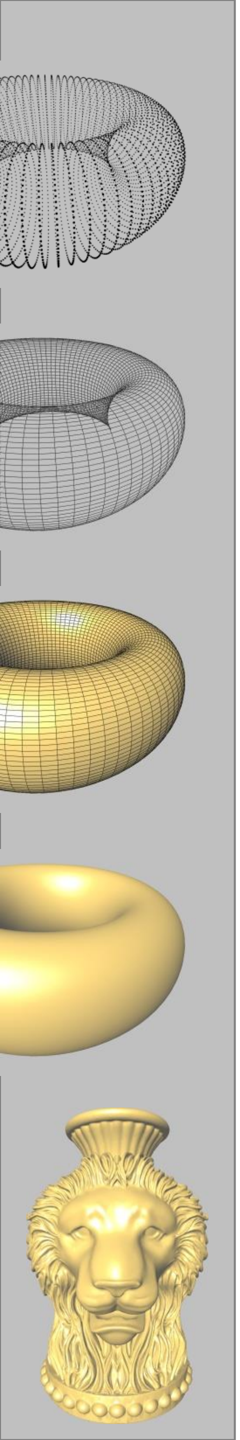
Associated with:

- 1 vertex
- 1 edge
- 1 facet

References:

- vertex
- opposite halfedge
- next halfedge
- facet





Halfedges

Geometry:

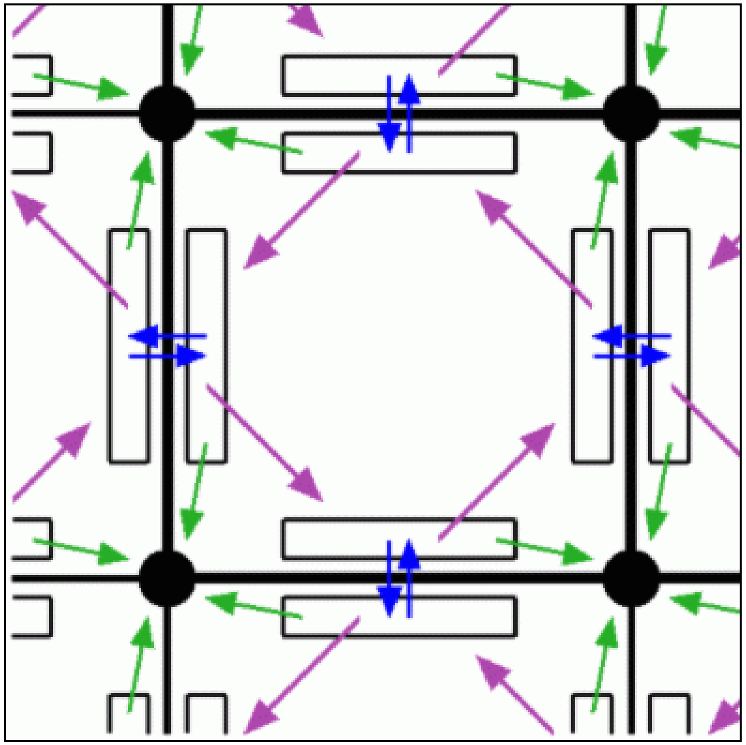
- vertices

Attributes:

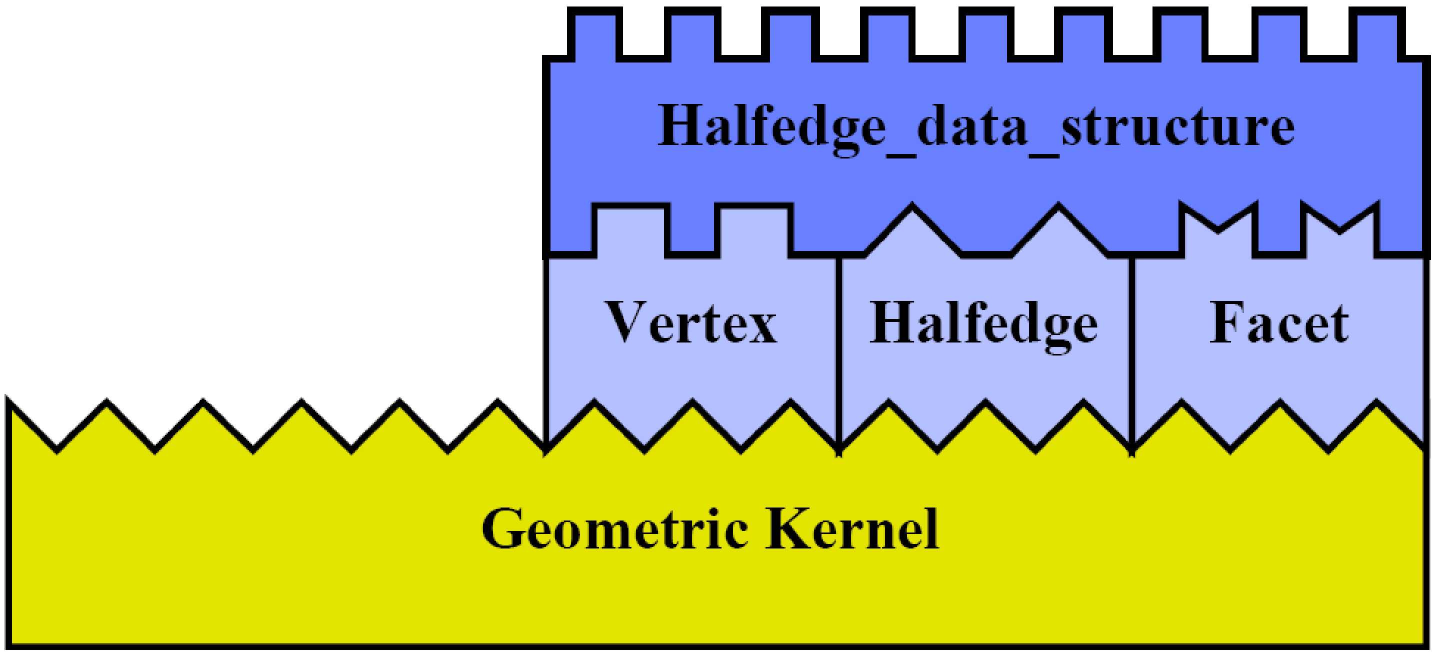
- on vertices
- on halfedges
- on facets

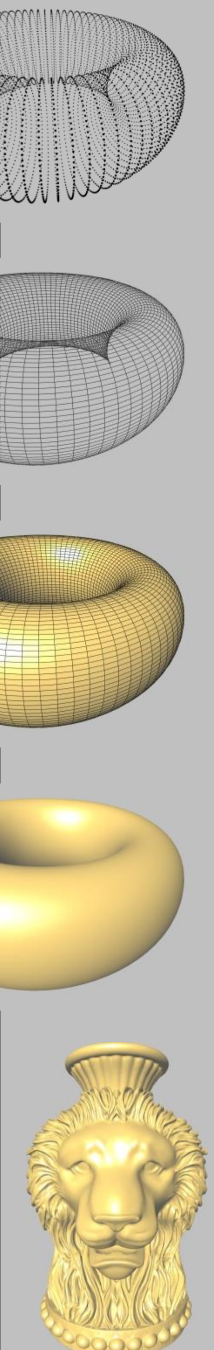
Connectivity:

- halfedges only

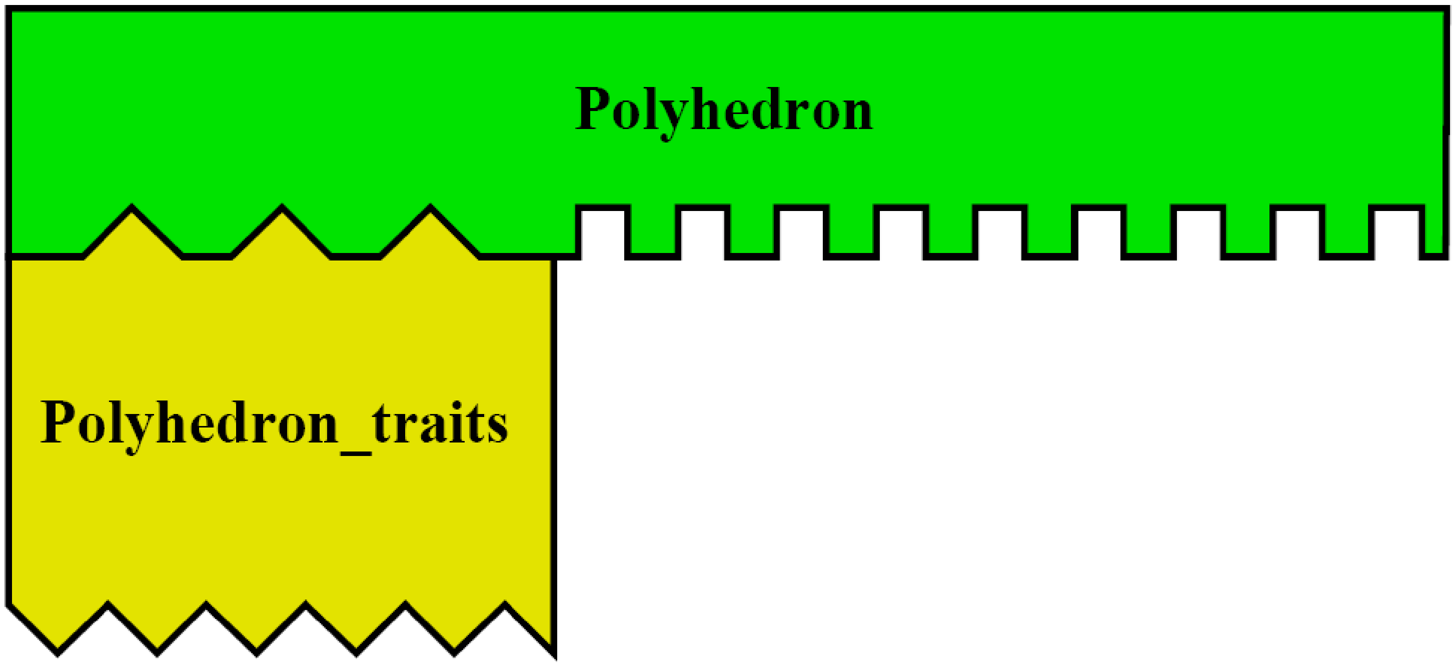


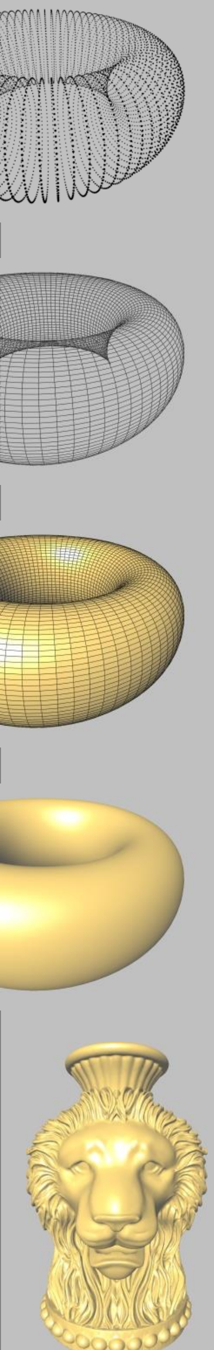
Building Blocks





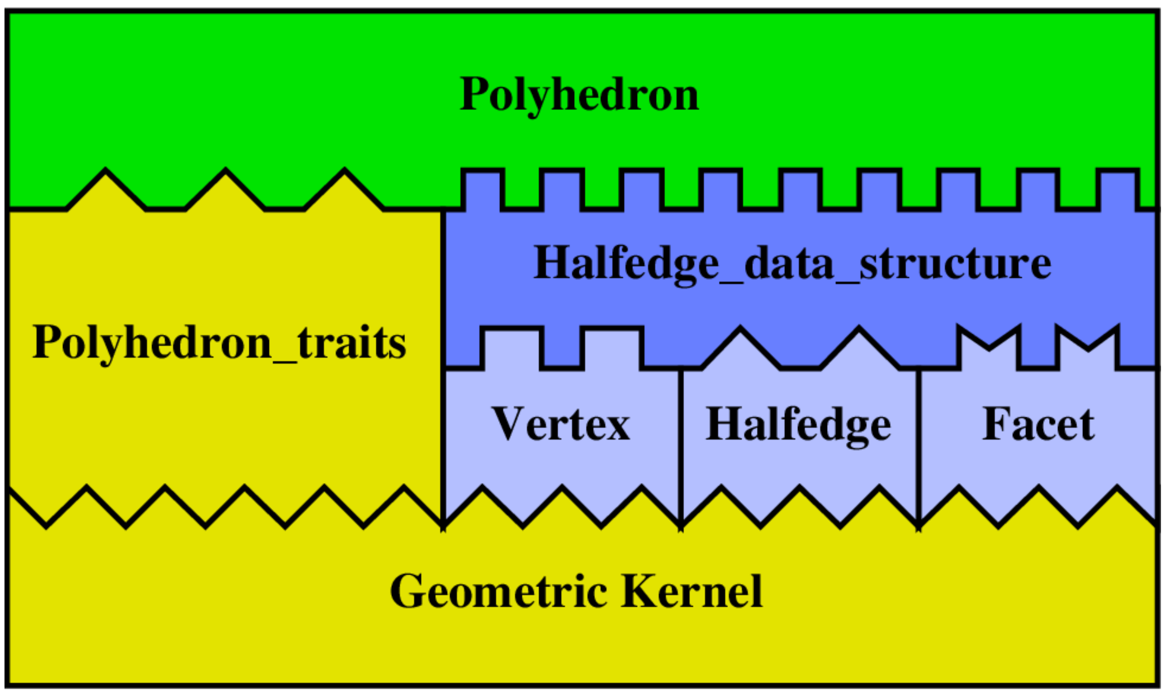
Building Blocks



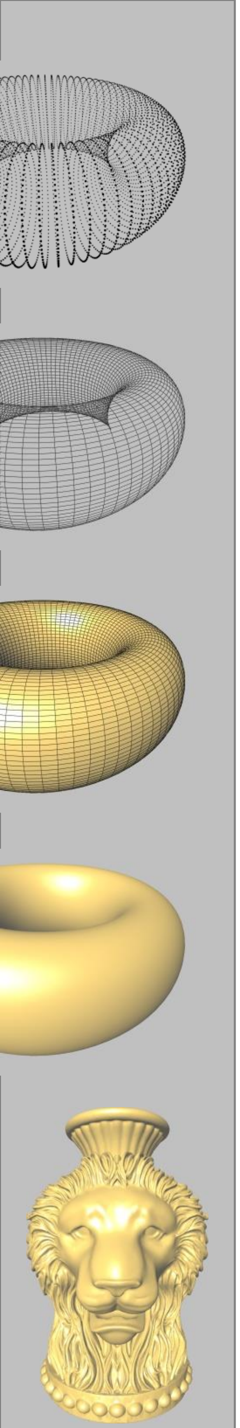


Polyhedral Surfaces

Building blocks assembled with C++ templates



http://www.cgal.org



Polyhedral Surface

Polyhedron

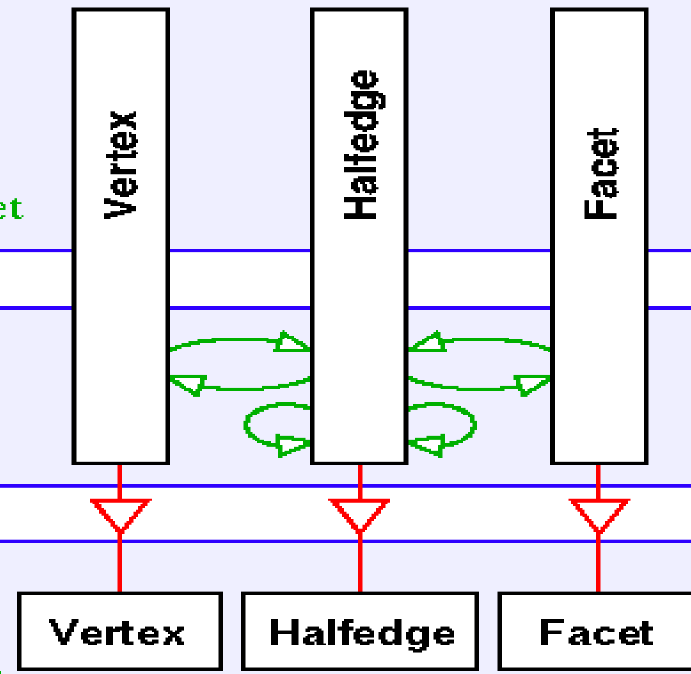
- provides ease-of-use
- protects combinatorial integrity
- defines circulators
- defines extended vertex, halfedge, facet

Halfedge_data_structure

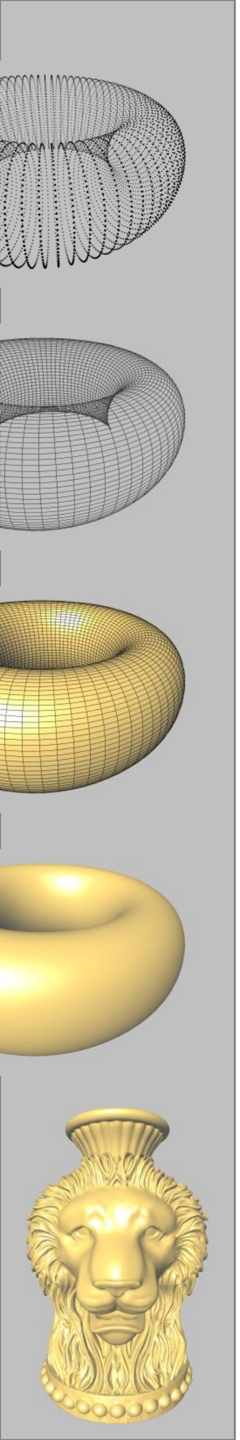
- manages storage (container class)
- defines iterators

Items

- stores actual information
- contains user added data and functions



CGAL

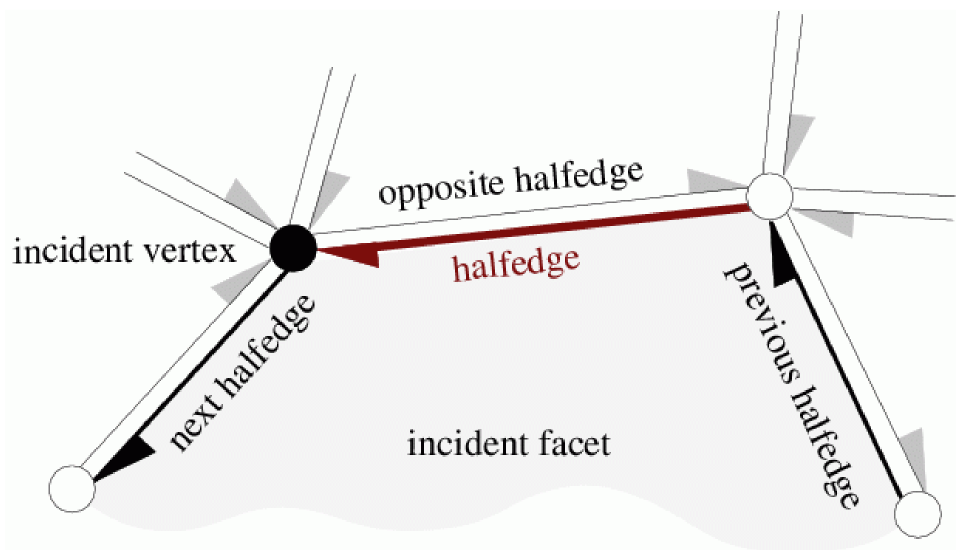


Default Polyhedron

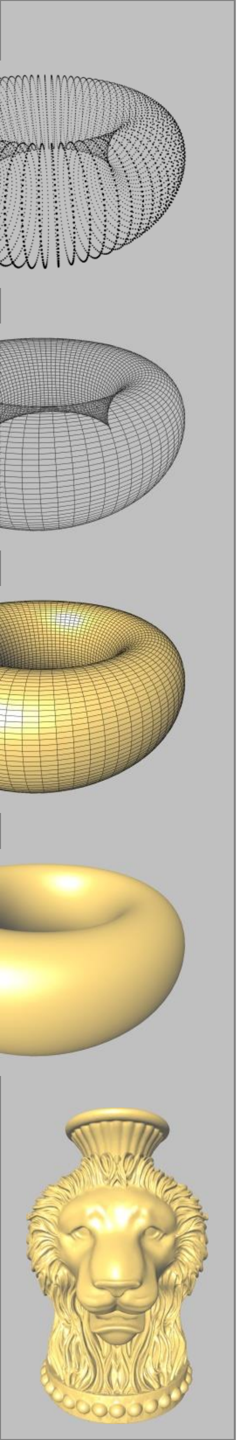
Vertex	
Halfedge_handle	halfedge()
Point&	point()
......	...

Halfedge	
Halfedge_handle	opposite()
Halfedge_handle	next()
Halfedge_handle	prev()
Vertex_handle	vertex()
Facet_handle	facet()
......	...

Facet	
Halfedge_handle	halfedge()
Plane&	plane()
Normal&	normal()
Color&	color()
......	...

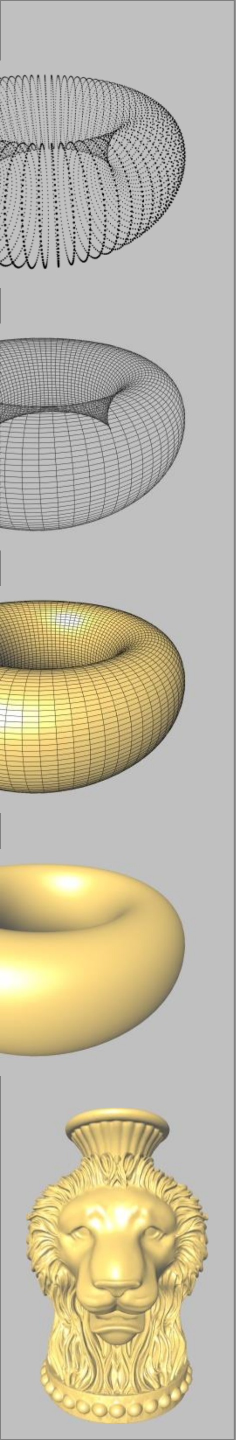


http://www.cgal.org



Default Polyhedron

```
typedef CGAL::Simple_cartesian<double> Kernel;
typedef Kernel::Point_3                Point_3;
typedef CGAL::Polyhedron_3<Kernel>     Polyhedron;
typedef Polyhedron::Vertex_iterator    Vertex_iterator;

int main() {
    Point_3 p( 1.0, 0.0, 0.0);
    Point_3 q( 0.0, 1.0, 0.0);
    Point_3 r( 0.0, 0.0, 1.0);
    Point_3 s( 0.0, 0.0, 0.0);

    Polyhedron P;
    P.make_tetrahedron( p, q, r, s);
    for (Vertex_iterator v = P.vertices_begin();
         v != P.vertices_end(); ++v)
        std::cout << v->point() << std::endl;
}
```

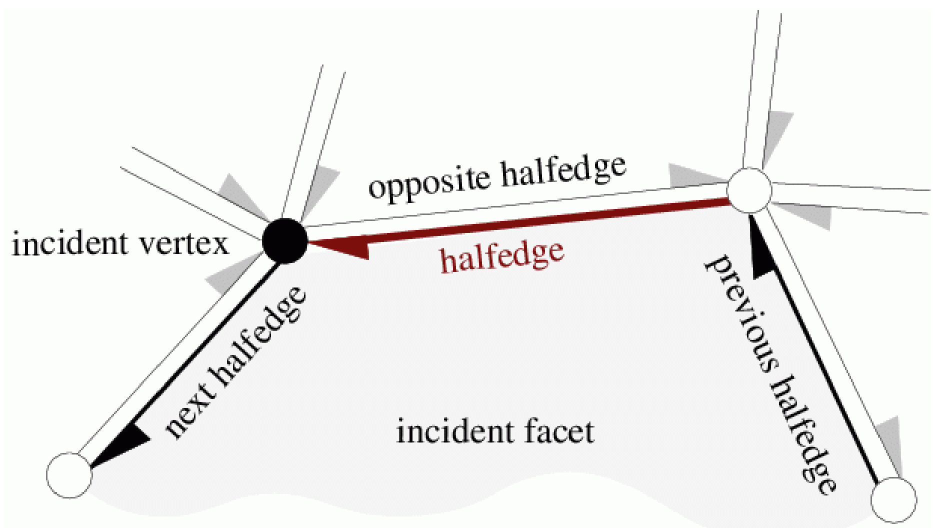


Flexible Data Structure

Vertex	
Halfedge_handle	halfedge()
Point&	point()
......	...

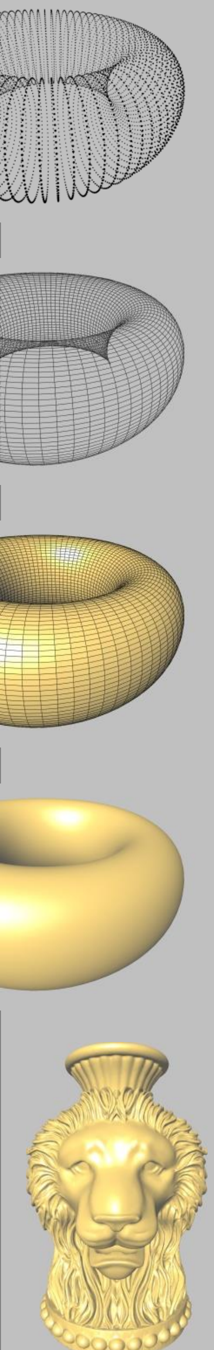
Halfedge	
Halfedge_handle	opposite()
Halfedge_handle	next()
Halfedge_handle	prev()
Vertex_handle	vertex()
Facet_handle	facet()
......	...

Facet	
Halfedge_handle	halfedge()
Plane&	plane()
Normal&	normal()
Color&	color()
......	...



http://www.cgal.org

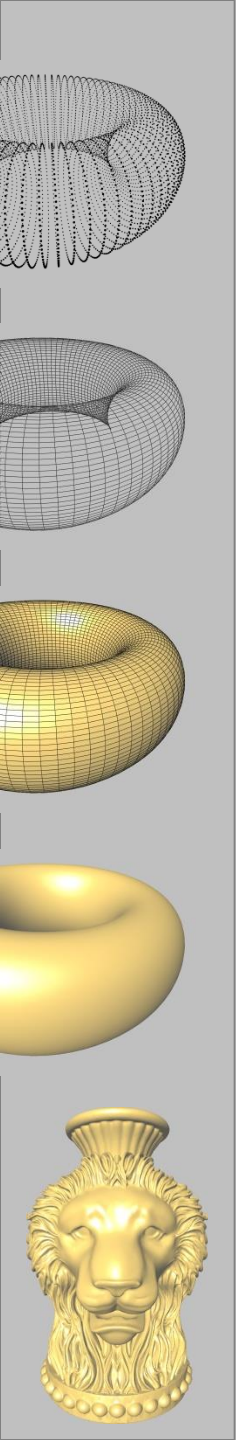
Flexible Polyhedral Surfaces



```
template <
     class  PolyhedronTraits_3,
     class  PolyhedronItems_3 = CGAL::Polyhedron_items_3,
     template < class T, class I>
     class  HalfedgeDS        = CGAL::HalfedgeDS_default,
     class  Alloc             = CGAL_ALLOCATOR(int)>
class Polyhedron_3;
```

http://www.cgal.org

CGAL

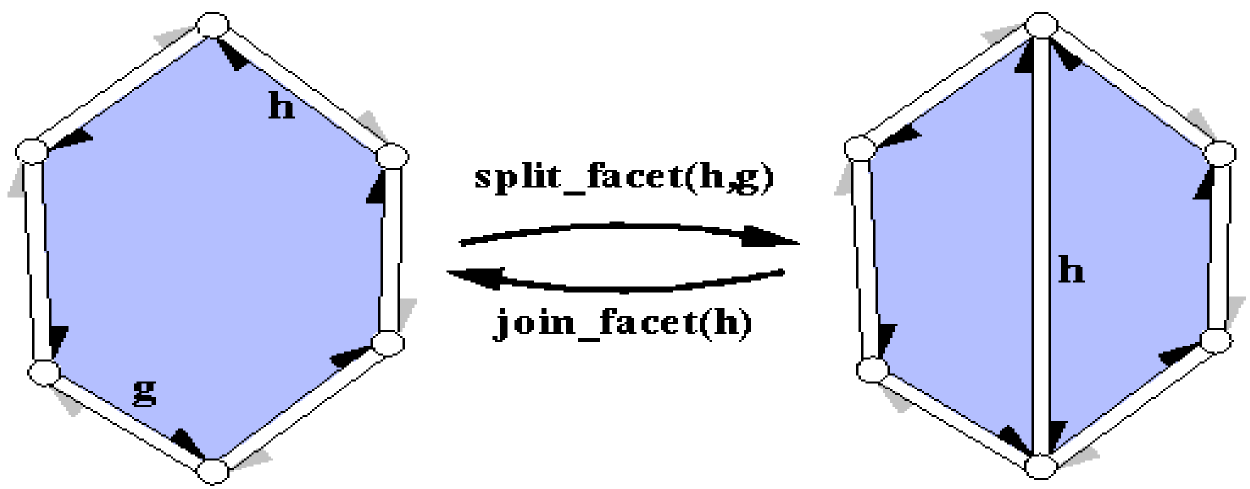


Default Polyhedron

```
typedef CGAL::Simple_cartesian<double> Kernel;
typedef Kernel::Point_3                Point_3;
typedef CGAL::Polyhedron_3<Kernel>     Polyhedron;
typedef Polyhedron::Vertex_iterator    Vertex_iterator;

int main() {
    Point_3 p( 1.0, 0.0, 0.0);
    Point_3 q( 0.0, 1.0, 0.0);
    Point_3 r( 0.0, 0.0, 1.0);
    Point_3 s( 0.0, 0.0, 0.0);

    Polyhedron P;
    P.make_tetrahedron( p, q, r, s);
    for ( Vertex_iterator v = P.vertices_begin();
          v != P.vertices_end(); ++v)
        std::cout << v->point() << std::endl;
}
```

Euler Operators

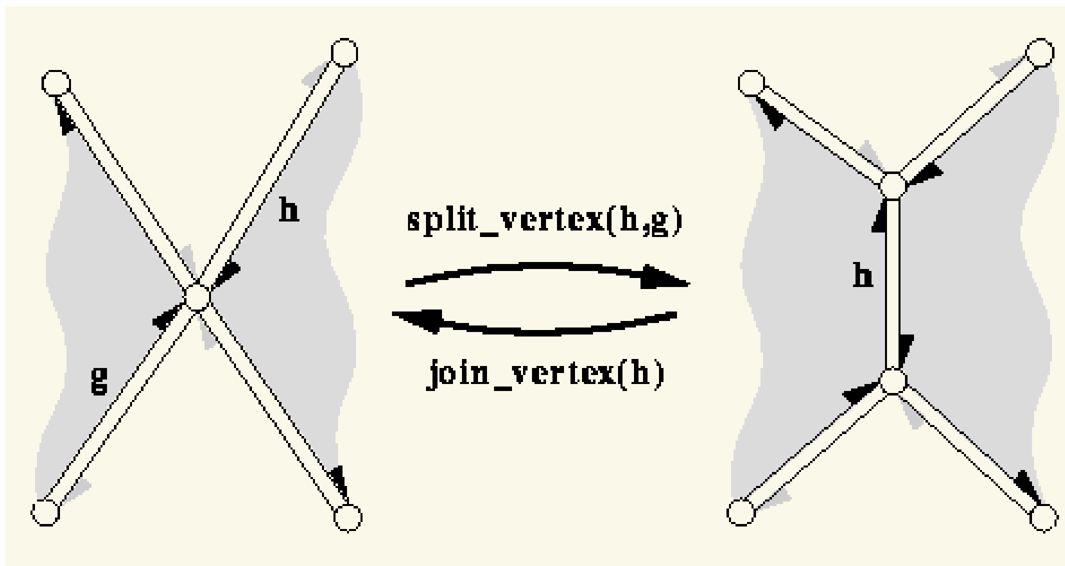
- Preserve the Euler-Poincaré equation
- Abstract from direct pointer manipulations



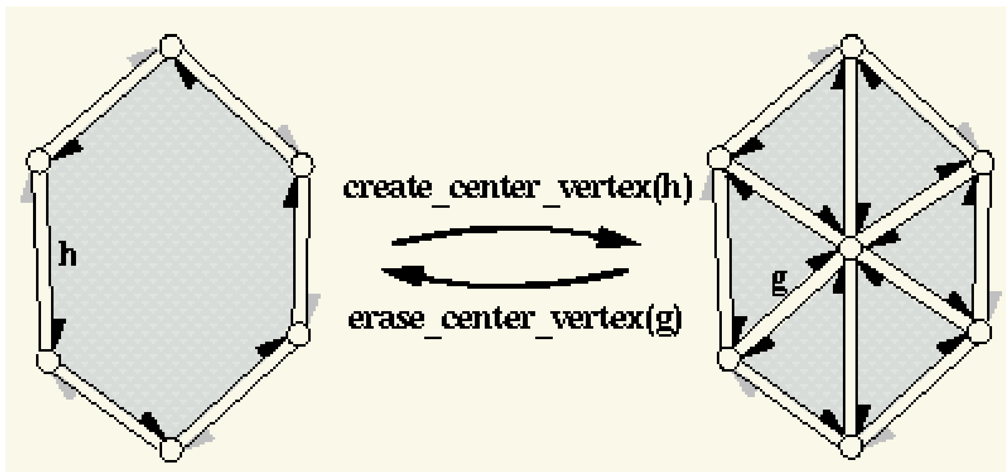
`Halfedge_handle P.split_facet(Halfedge_handle h,Halfedge_handle g)`

Euler Operators

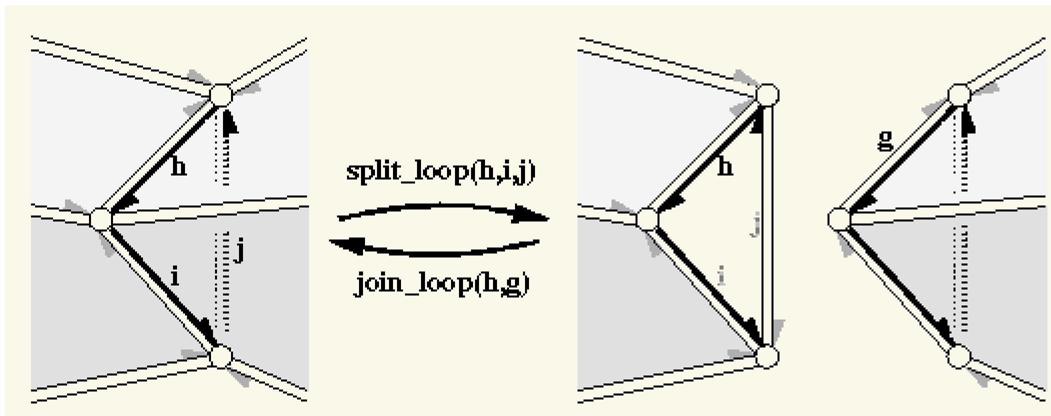
```
Halfedge_handle P.split_vertex ( Halfedge_handle h, Halfedge_handle g
Halfedge_handle P.join_vertex ( Halfedge_handle h)
```



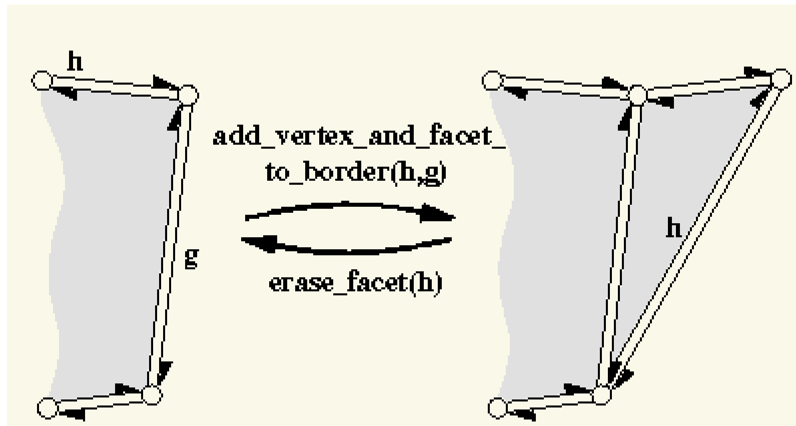
Euler Operators

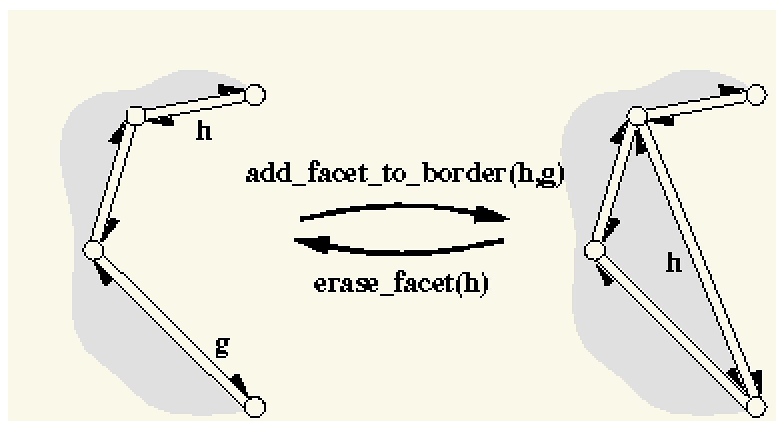


Modifying the Genus

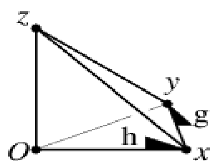


Modifying Facets & Holes



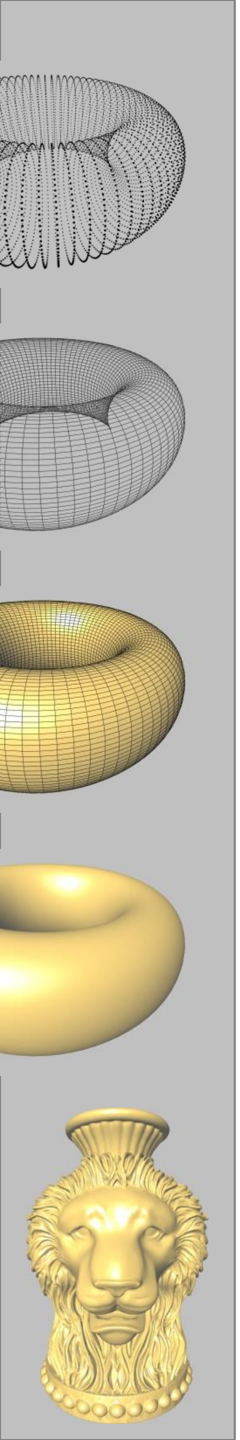


Create a Cube with Euler Operator

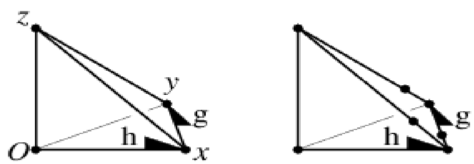


(a)

```
Halfedge_handle h = P.make_tetrahedron(
    Point(1,0,0), Point(0,0,1),
    Point(0,0,0), Point(0,1,0));
Halfedge_handle g = h->next()->opposite()->next();  (a)
```

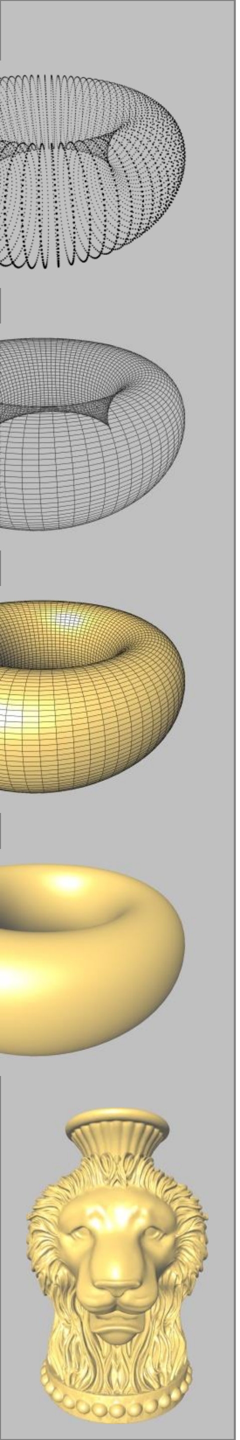


Create a Cube with Euler Operator

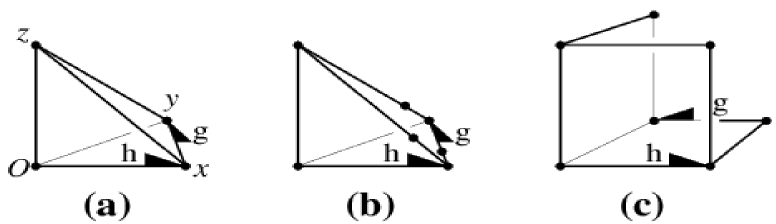


(a) (b)

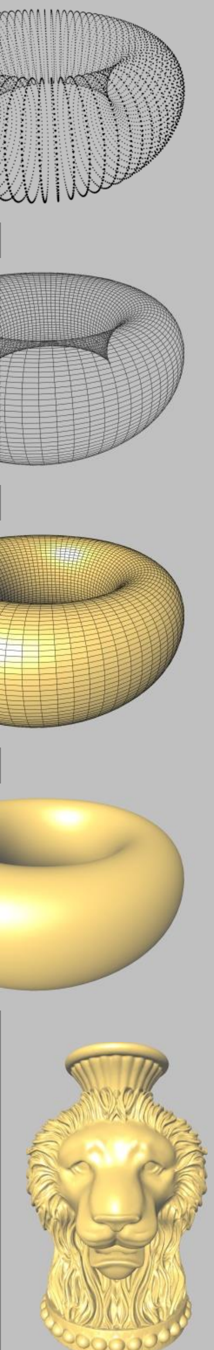
```
Halfedge_handle h = P.make_tetrahedron(
    Point(1,0,0), Point(0,0,1),
    Point(0,0,0), Point(0,1,0));
Halfedge_handle g = h->next()->opposite()->next();  (a)

P.split_edge( h->next());
P.split_edge( g->next());
P.split_edge( g);                                   (b)
```



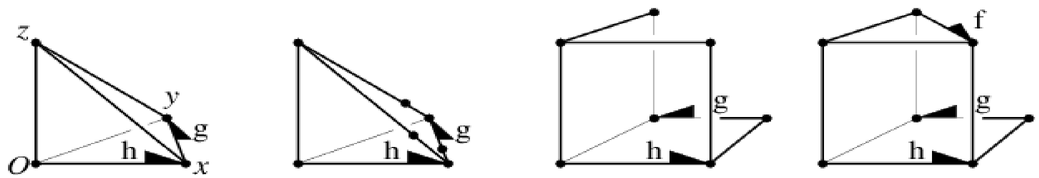
Create a Cube with Euler Operator




```
h->next()->vertex()->point()      = Point( 1, 0, 1);
g->next()->vertex()->point()      = Point( 0, 1, 1);
g->opposite()->vertex()->point() = Point( 1, 1, 0); (c)
```



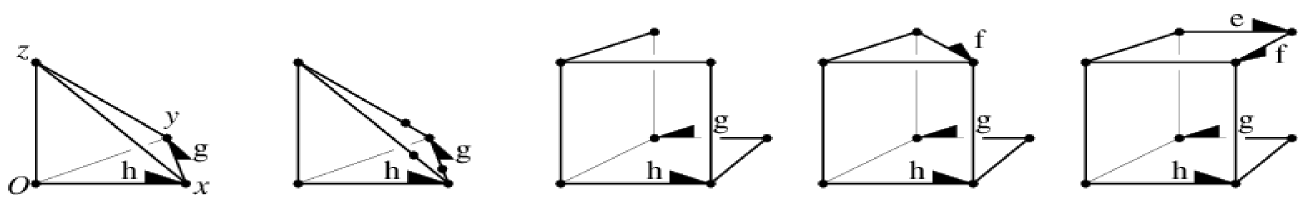
Create a Cube with Euler Operator



(a) (b) (c) (d)

```
h->next()->vertex()->point()     = Point( 1, 0, 1);
g->next()->vertex()->point()     = Point( 0, 1, 1);
g->opposite()->vertex()->point() = Point( 1, 1, 0); (c)
Halfedge_handle f = P.split_facet( g->next(),
                                   g->next()->next()->next()); (d)
```

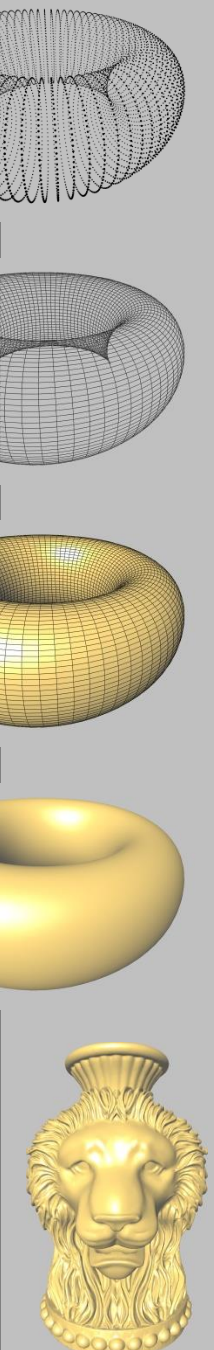
Create a Cube with Euler Operator

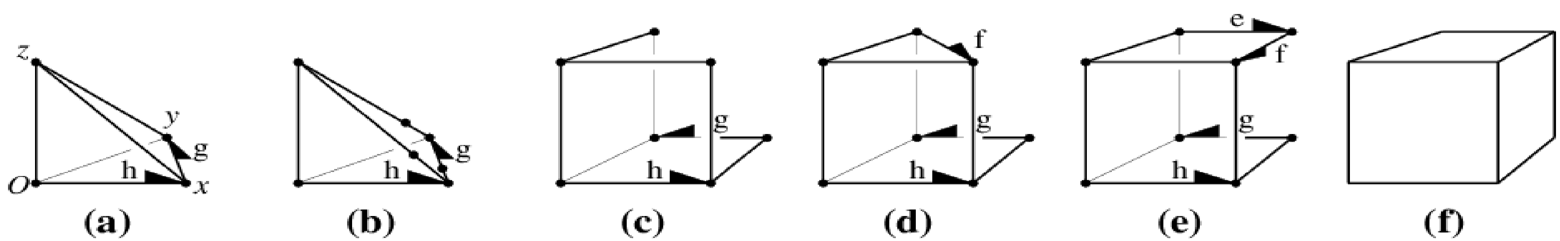


(a) (b) (c) (d) (e)

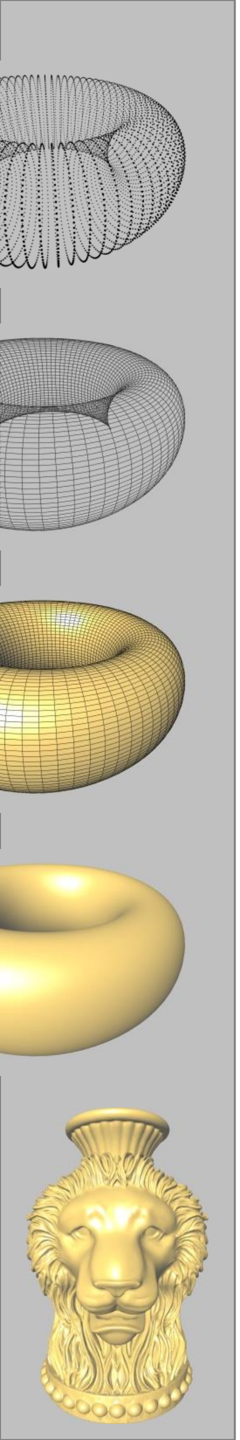
```
h->next()->vertex()->point()     = Point( 1, 0, 1);
g->next()->vertex()->point()     = Point( 0, 1, 1);
g->opposite()->vertex()->point() = Point( 1, 1, 0); (c)
Halfedge_handle f = P.split_facet( g->next(),
                        g->next()->next()->next()); (d)
Halfedge_handle e = P.split_edge( f);
e->vertex()->point() = Point( 1, 1, 1);             (e)
```

Create a Cube with Euler Operator



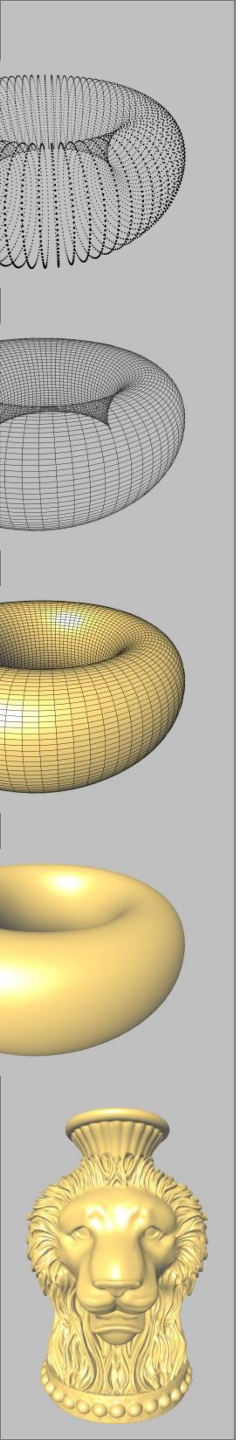



```
h->next()->vertex()->point()     = Point( 1, 0, 1);
g->next()->vertex()->point()     = Point( 0, 1, 1);
g->opposite()->vertex()->point() = Point( 1, 1, 0); (c)
Halfedge_handle f = P.split_facet( g->next(),
                        g->next()->next()->next()); (d)
Halfedge_handle e = P.split_edge( f);
e->vertex()->point() = Point( 1, 1, 1);              (e)
P.split_facet( e, f->next()->next());                (f)
```



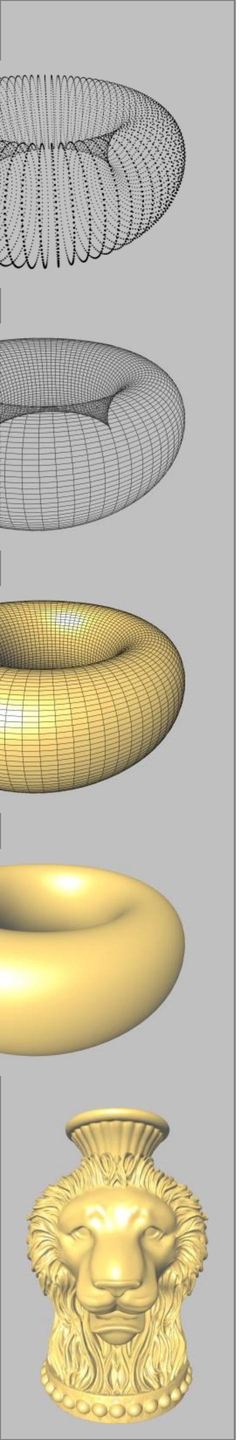
Extending Primitives

```
typedef CGAL::Polyhedron_3<         Traits,
                                    CGAL::Polyhedron_items_3,
                                    CGAL::HalfedgeDS_default> Polyhedron;

class Polyhedron_items_3 {
public:

      template < class Refs, class Traits>
      struct Vertex_wrapper {
      typedef typename Traits::Point_3 Point;
      typedef CGAL::HalfedgeDS_vertex_base<Refs, CGAL::Tag_true, Point> Vertex;
      };

      template < class Refs, class Traits>
      struct Halfedge_wrapper {
      typedef CGAL::HalfedgeDS_halfedge_base<Refs> Halfedge;
      };

      template < class Refs, class Traits>
      struct Face_wrapper {
      typedef typename Traits::Plane_3 Plane;
      typedef CGAL::HalfedgeDS_face_base<Refs, CGAL::Tag_true, Plane> Face;
      };
};
```



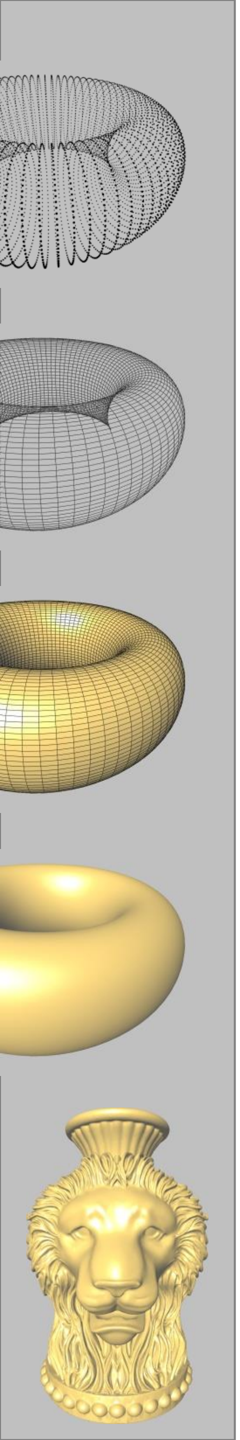
Add Color to Facets

```
template <class Refs>
struct CFace : public CGAL::HalfedgeDS_face_base<Refs>{
    CGAL::Color color;
};

// ...

typedef CGAL::Simple_cartesian<double>            Kernel;
typedef CGAL::Polyhedron_3<Kernel, ...>       Polyhedron;
typedef Polyhedron::Halfedge_handle   Halfedge_handle;

int main() {
    Polyhedron P;
    Halfedge_handle h = P.make_tetrahedron();
    h->facet()->color = CGAL::RED;
    return 0;
}
```



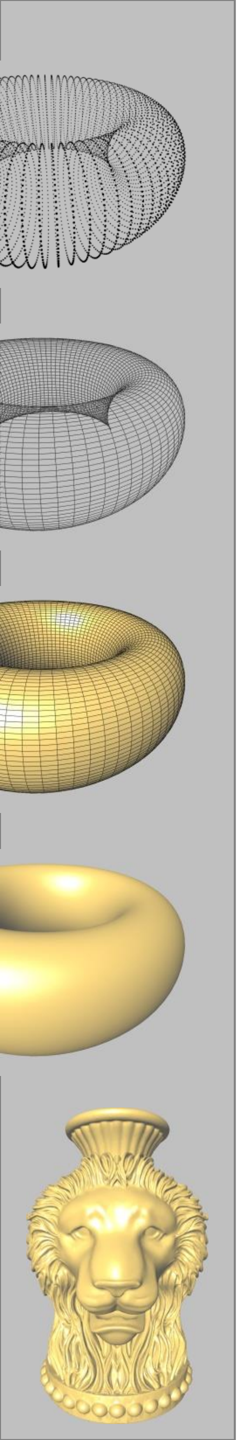
Add Color to Facets

```
template <class Refs>
struct CFace : public CGAL::HalfedgeDS_face_base<Refs>{
    CGAL::Color color;
};

struct CItems : public CGAL::Polyhedron_items_3 {
    template <class Refs, class Traits>
    struct Face_wrapper {
        typedef CFace<Refs> Face;
    };
};

typedef CGAL::Simple_cartesian<double>          Kernel;
typedef CGAL::Polyhedron_3<Kernel, CItems> Polyhedron;
```



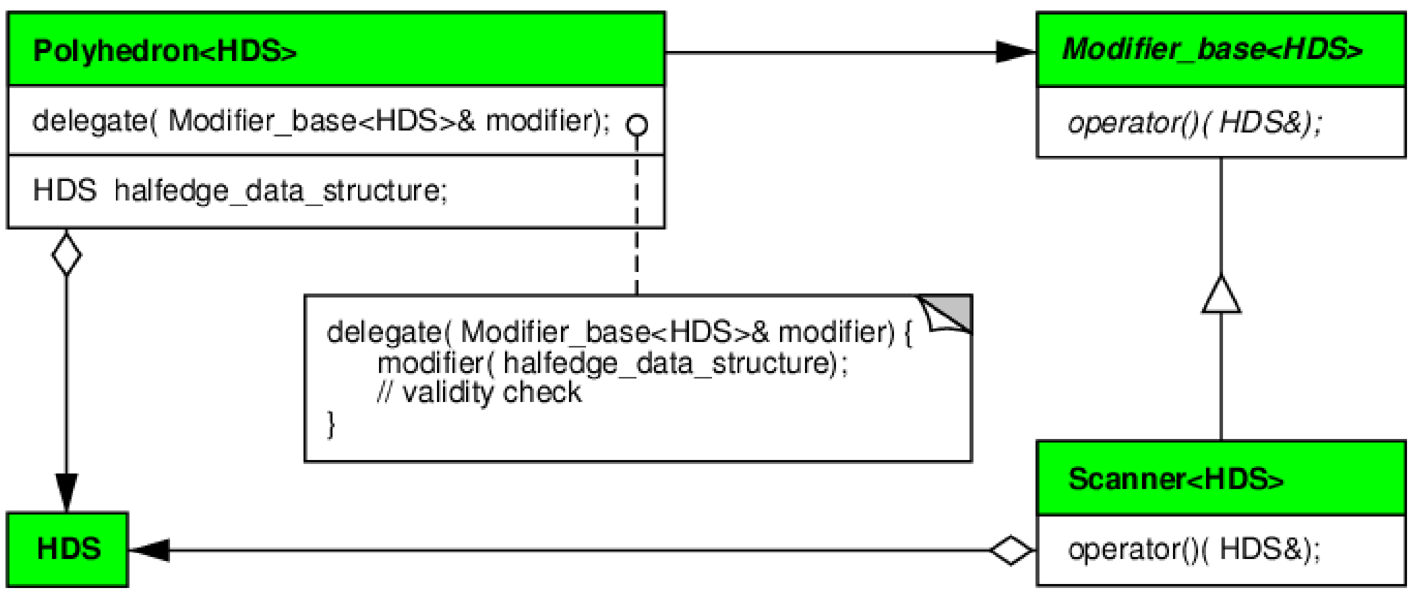
Add Vertex_handle to Facets

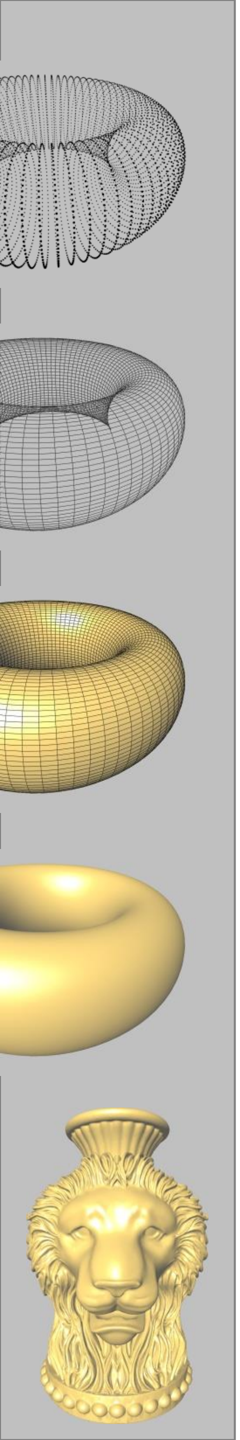
```
template <class Refs>
struct VFace : public CGAL::HalfedgeDS_face_base<Refs>{
    typedef typename Refs::Vertex_handle Vertex_handle;
    Vertex_handle vertex_ref;
};
```



Incremental Builder

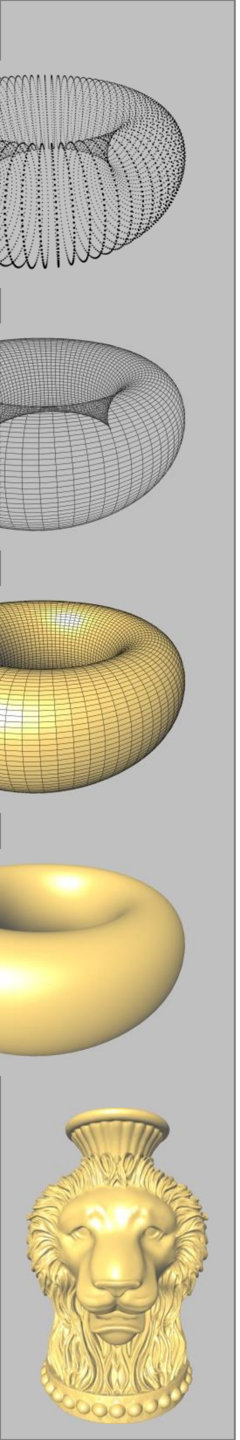
- Uses the modifier design to access the internal HDS





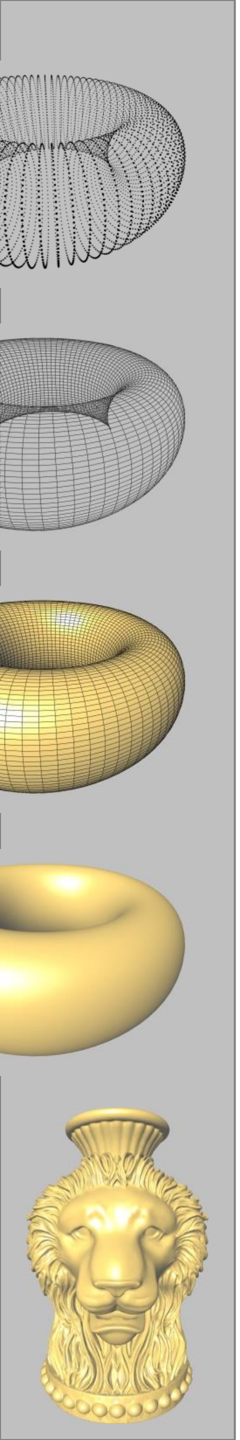
Make Triangle with Incremental Builder

```
template <class HDS>
struct Mk_triangle : public CGAL::Modifier_base<HDS> {

  void operator()( HDS& hds) {// Postcond: 'hds' valid

    CGAL::Polyhedron_incremental_builder_3<HDS> B(hds);
    B.begin_surface( 3, 1, 6);
    typedef typename HDS::Vertex   Vertex;
    typedef typename Vertex::Point Point;
    B.add_vertex( Point( 0, 0, 0));
    B.add_vertex( Point( 1, 0, 0));
    B.add_vertex( Point( 0, 1, 0));
    B.begin_facet();
    B.add_vertex_to_facet( 0);
    B.add_vertex_to_facet( 1);
    B.add_vertex_to_facet( 2);
    B.end_facet();
    B.end_surface();
  }
};
```



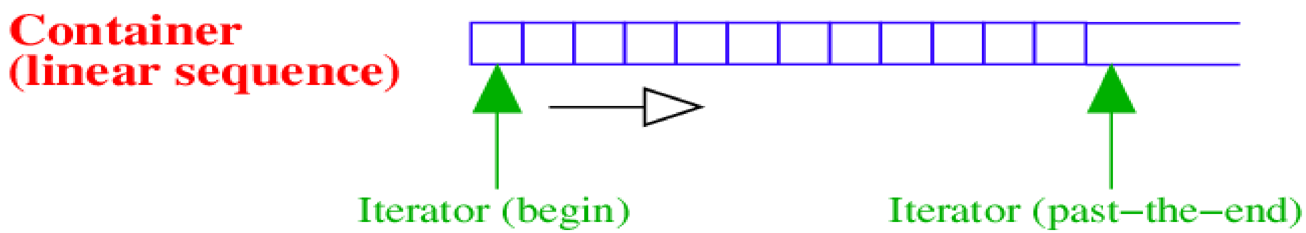
Make Triangle with Incremental Builder

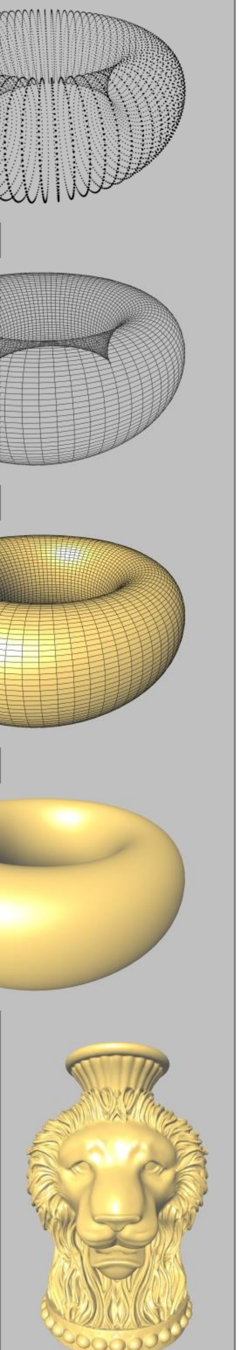
```
main() {
    Polyhedron P;
    Mk_triangle<HalfedgeDS> triangle;
    P.delegate( triangle);
    return 0;
}
```



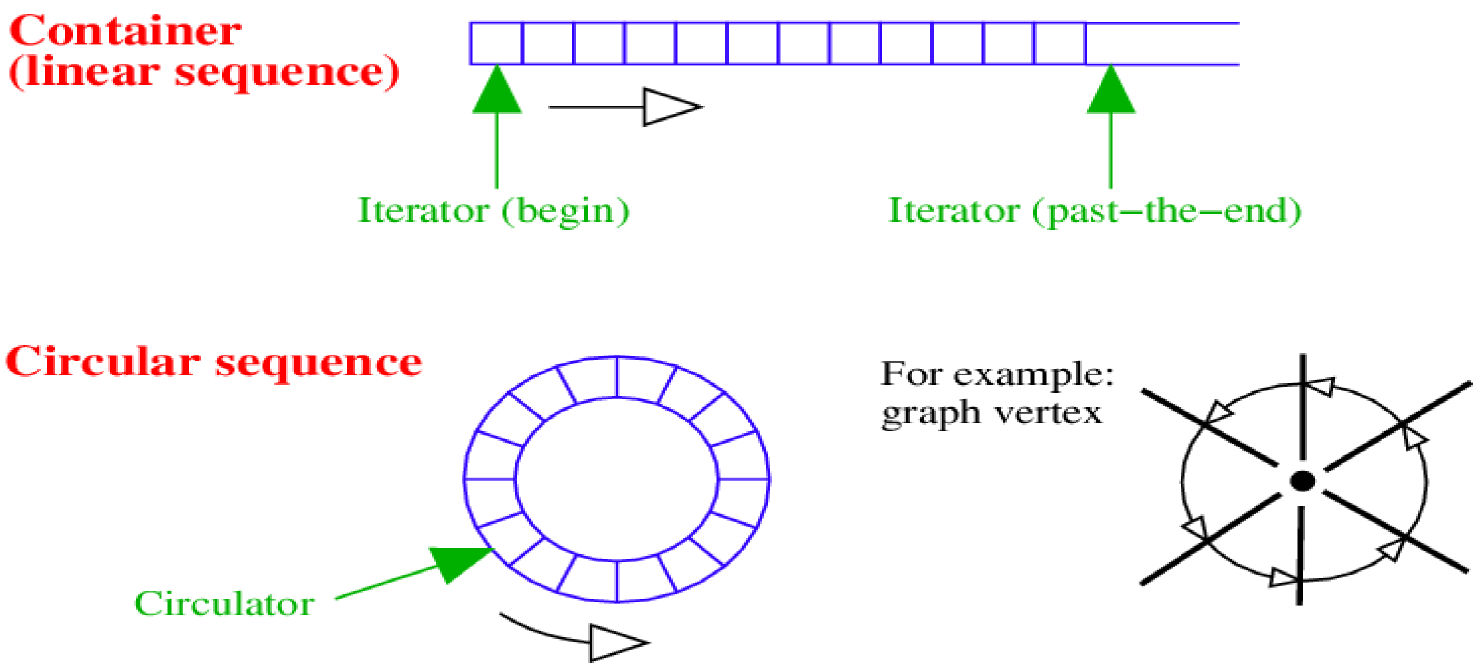
Traversal

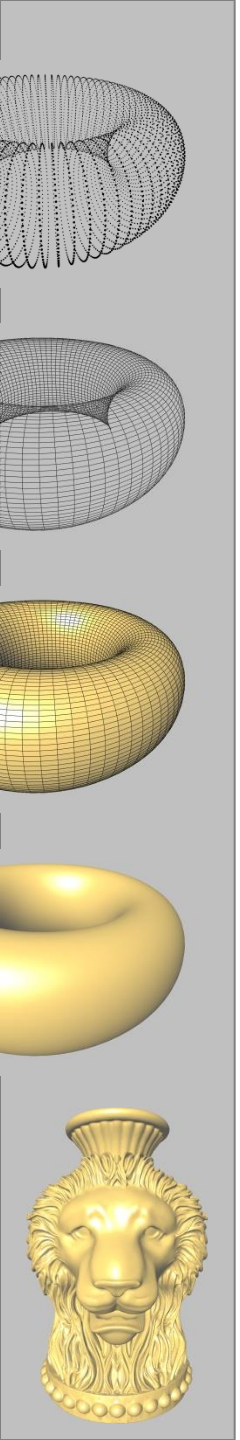
Iterators


Container
(linear sequence)
Iterator (begin)
Iterator (past–the–end)



Iterators and Circulators



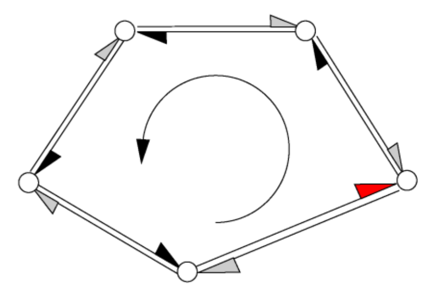


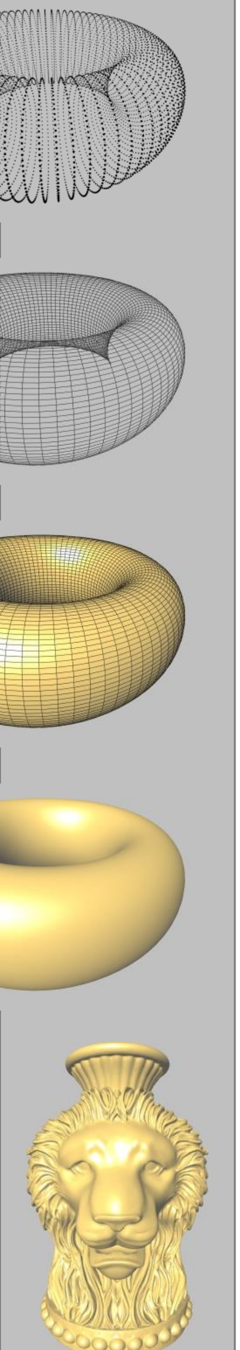
Iteration

```
Vertex_iterator iter;
for(  iter = polyhedron.vertices_begin();
      iter != polyhedron.vertices_end();
      iter++)
{
  Vertex_handle hVertex = iter;
  // do something with hVertex
}
```

Circulation

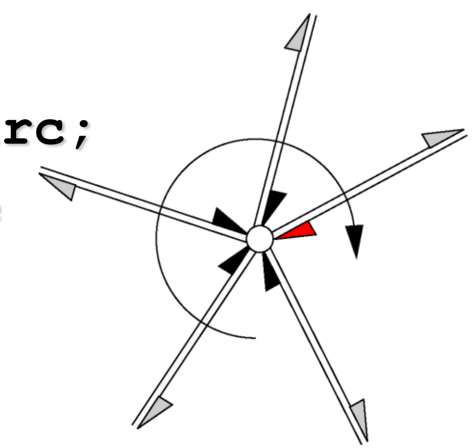
```
// circulate around hFacet
Halfedge_around_facet_circulator circ =
   hFacet->facet_begin();
Halfedge_around_facet_circulator end = circ;

CGAL_For_all(circ,end)
{
   Halfedge_handle hHalfedge = circ;
   // do something with hHalfedge
}
```





Circulation

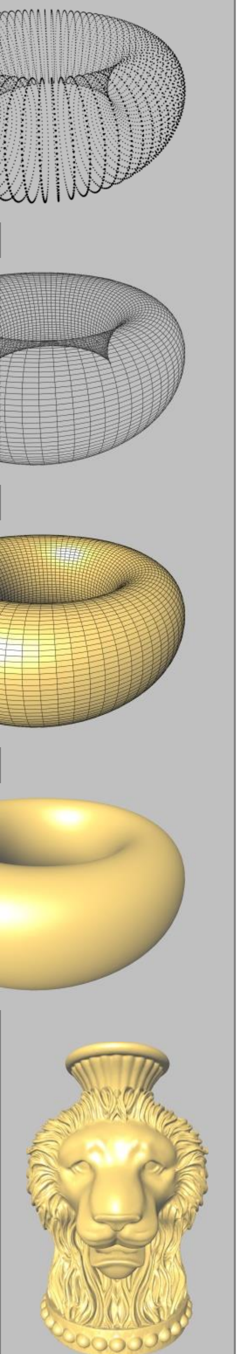
```
// circulate around hVertex
Halfedge_around_vertex_circulator circ =
  hVertex->vertex_begin();
Halfedge_around_vertex_circulator end = circ;

CGAL_For_all(circ,end)
{
  Halfedge_handle hHalfedge = circ;
  // do something with hHalfedge
}
```



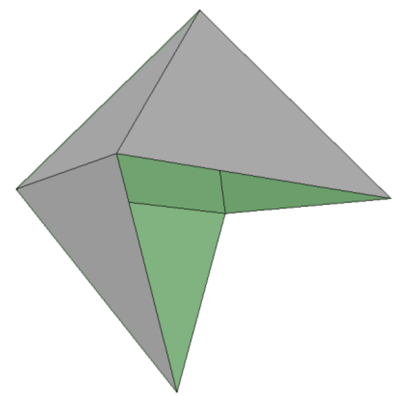
File I/O

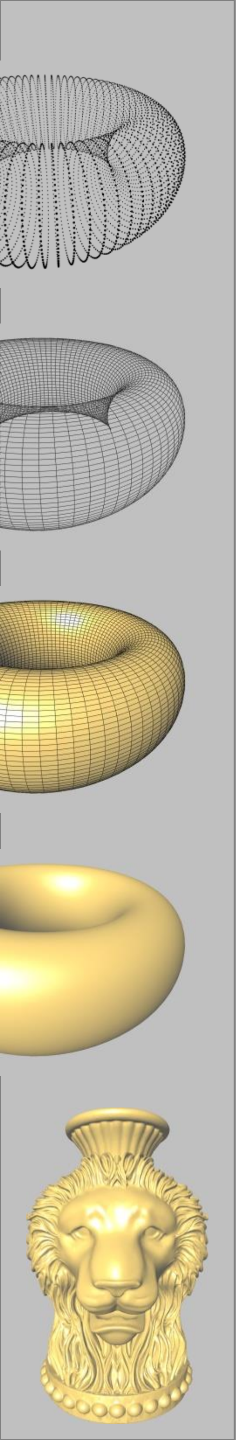
I/O: OFF indexed format

Output: vrml1-2, iv, geomview



```
OFF
6 6 0
0.000000 1.686000 0.000000
1.192000 0.000000 -1.192000
-1.192000 0.000000 -1.192000
-1.192000 0.000000 1.192000
1.192000 0.000000 1.192000
0.000000 -1.68600 0.000000
3 0 4 1
3 1 5 2
3 2 3 0
3 1 2 0
3 3 4 0
3 3 2 5
```

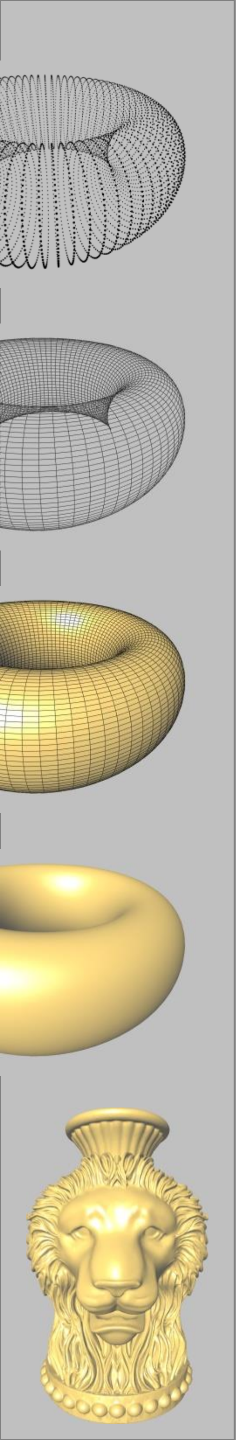




Applications

- Rendering
- Subdivision Surfaces
- Algorithms on Meshes
  - Simplification
  - Approximation
  - Remeshing
  - Smoothing
  - Compression
- Etc.

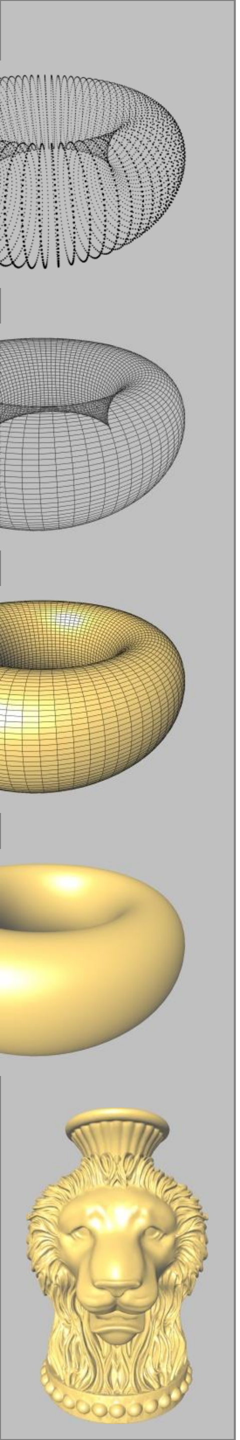
Warm-Up Exercices



Highlight Boundary Edges

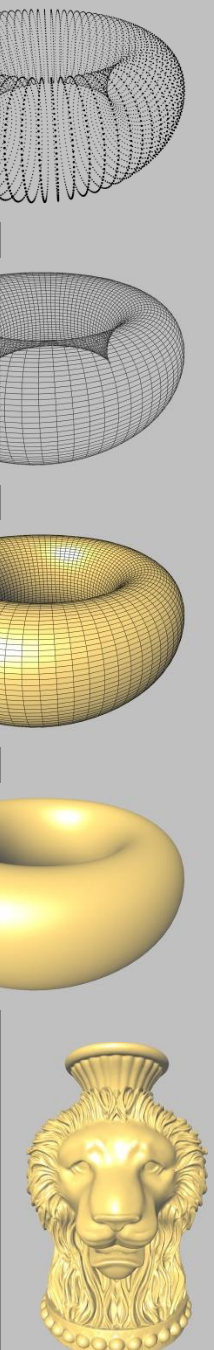
Notes:

```
bool halfedge->is_border();

// change color
::glColor3f(r,g,b); // in [0.0f-1.0f]
// Assemble primitive
```

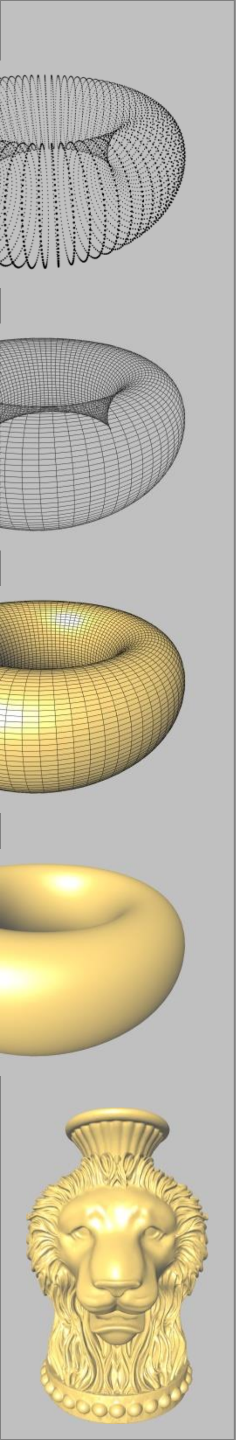


Render all Facets as Polygons

```
typedef Polyhedron::Facet_iterator Facet_iterator;
typedef Polyhedron::Halfedge_around_facet_circulator
                                   HF_circulator;
Facet_iterator f;
for (f = P.facets_begin();
     f != P.facets_end();
     f++)
{
 HF_circulator he = f->facet_begin();
    ::glBegin(GL_POLYGON);
    do
   {
       const Point& p = he->vertex();
        ::glVertex3d(p.x(),p.y(),p.z());
   }
    while( ++he != f->facet_begin());
    ::glEnd();
}
```

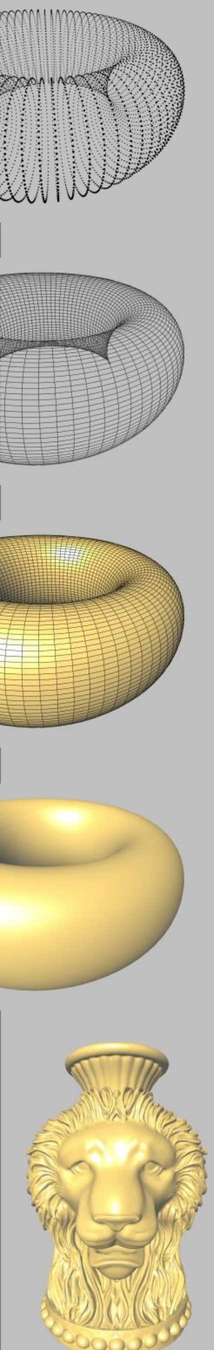


Exercices Around Combinatorics

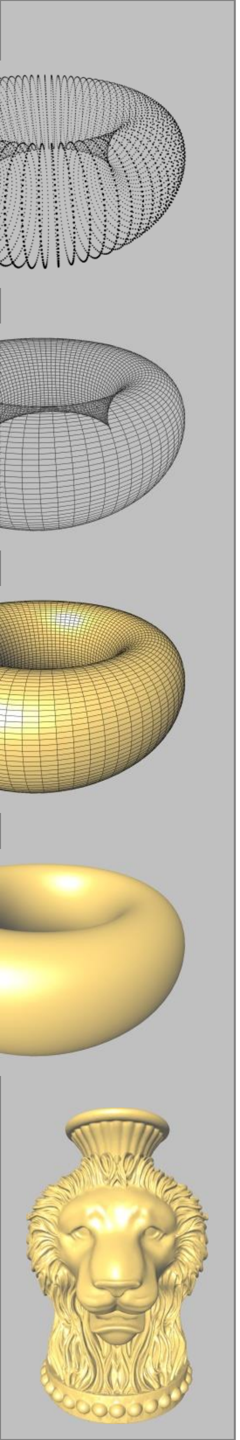


Combinatorial Genus

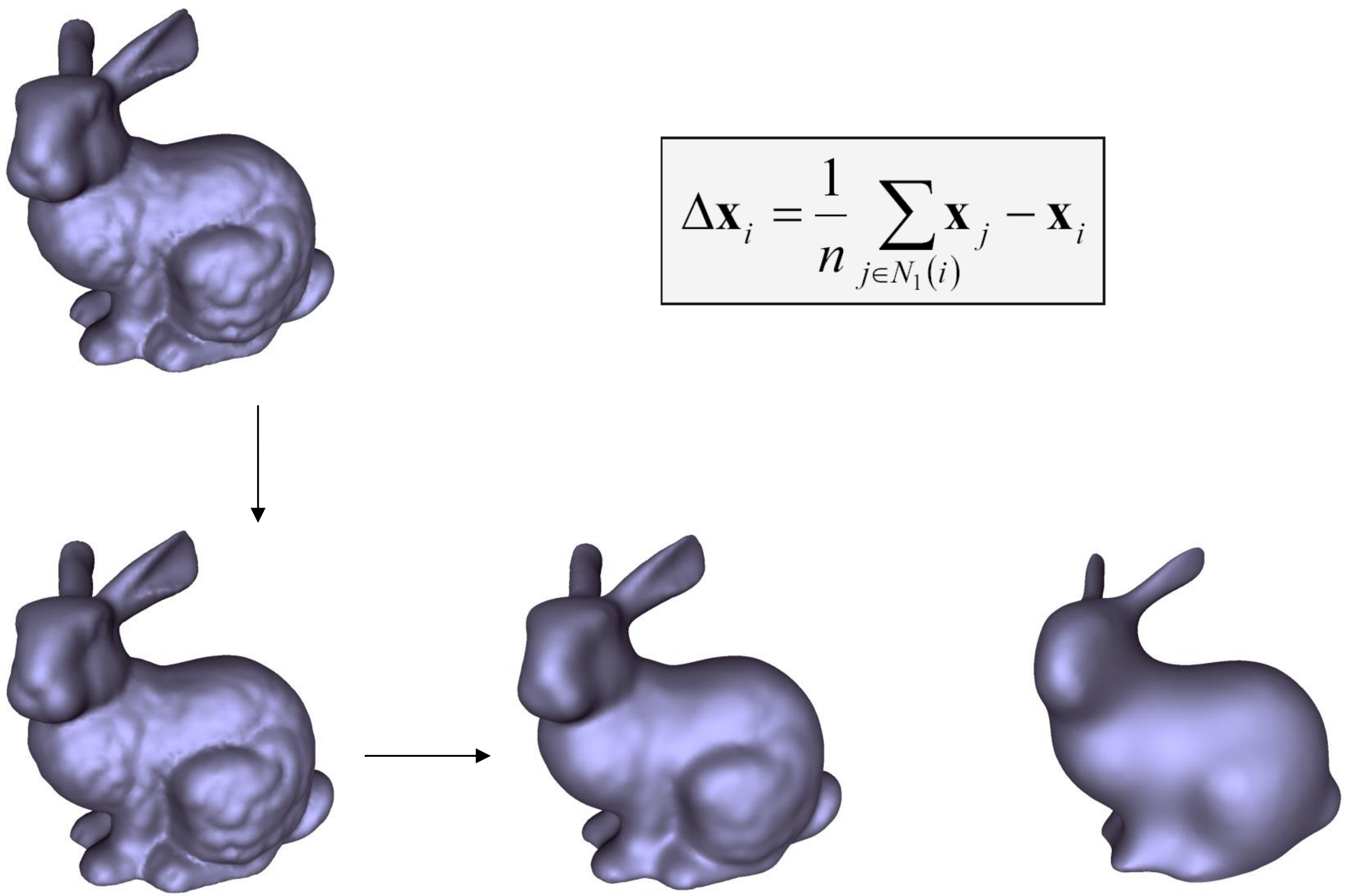
- Enumerate connected components
- Enumerate boundaries
- Deduce genus using Euler formula



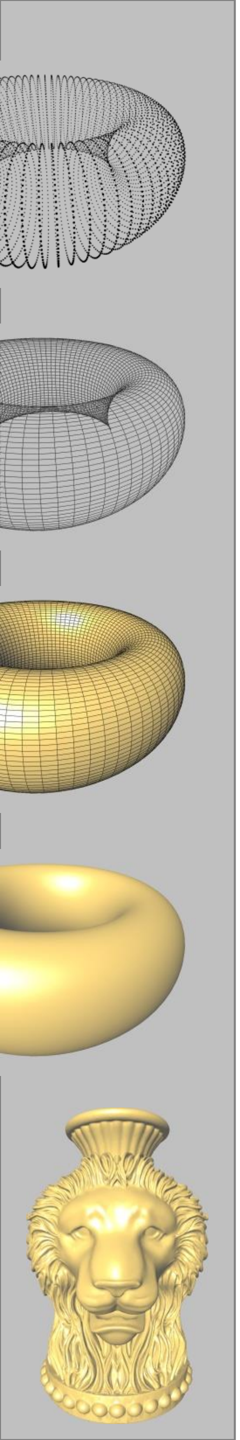
Exercices Around Geometry



Discrete Laplacian

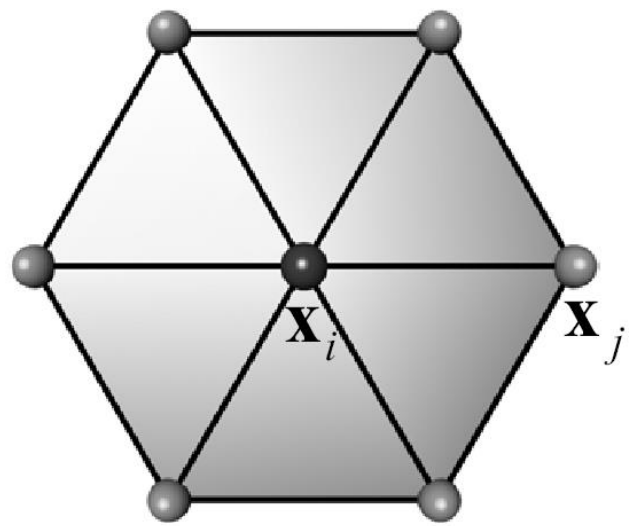


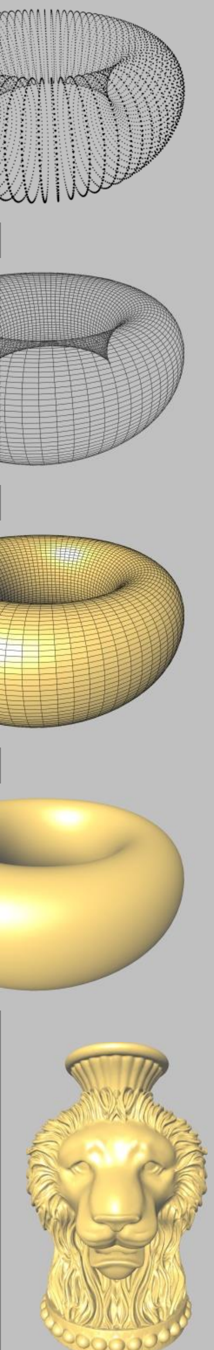
$$\Delta \mathbf{x}_i = \frac{1}{n} \sum_{j \in N_1(i)} \mathbf{x}_j - \mathbf{x}_i$$



$$\Delta \mathbf{x}_i = \frac{1}{n} \sum_{j \in N_1(i)} \mathbf{x}_j - \mathbf{x}_i$$

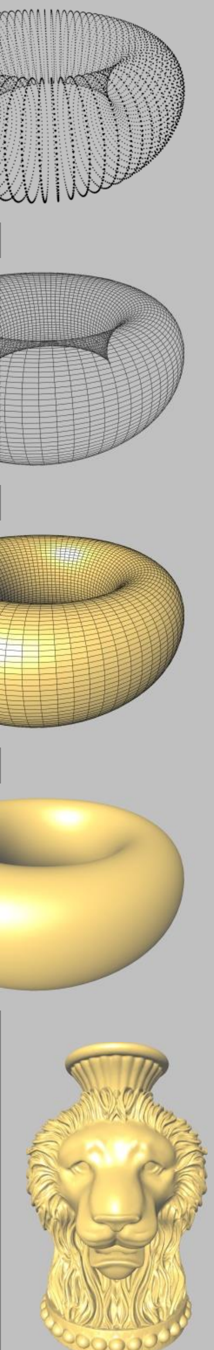
valence(vertex)



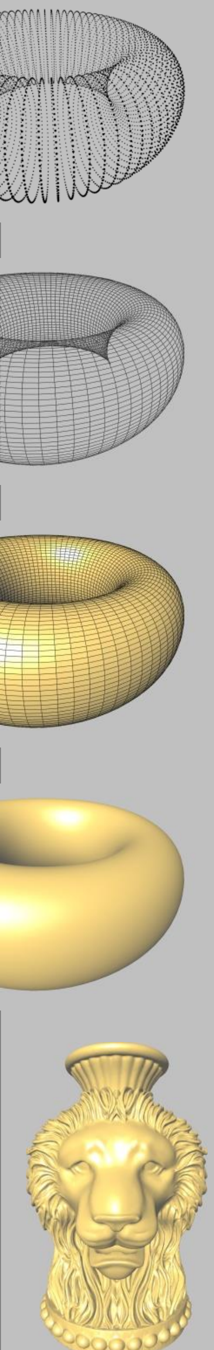


Discrete Laplacian

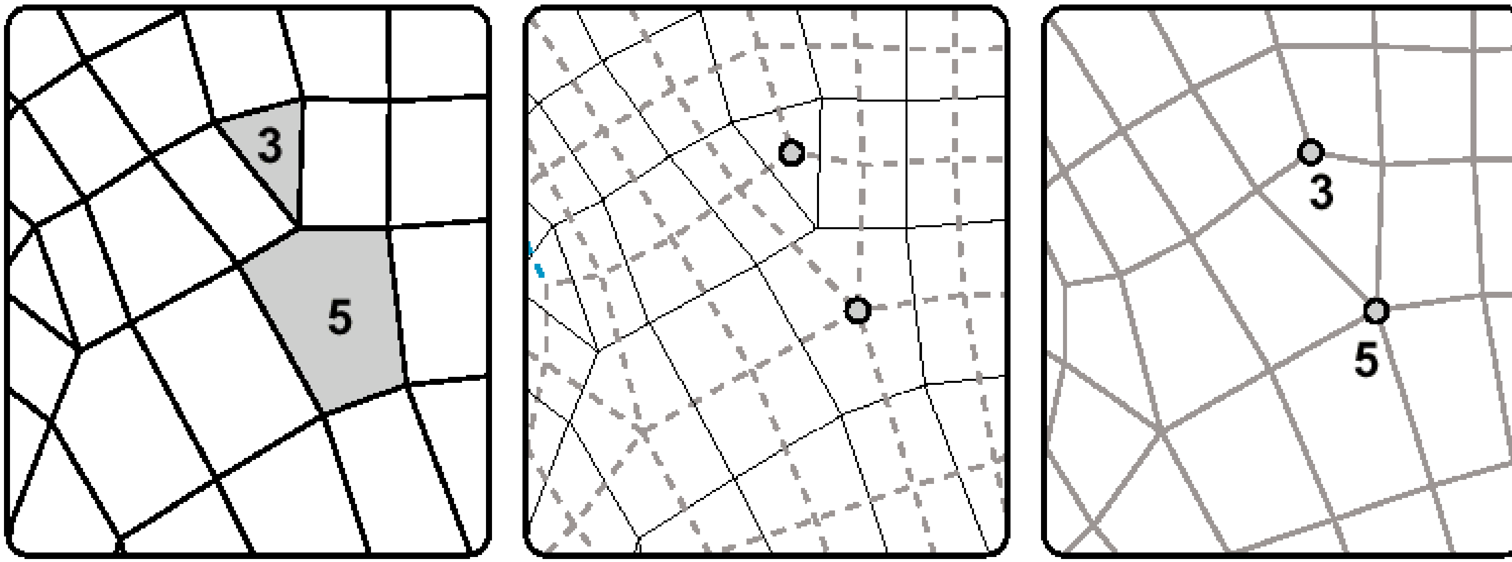
```
int nbv = size_of_vertices();
std::vector<Vector_3> dx(nbv);

iterate on vertices
    dx[i] = Vector_3(0,0,0); // init
    circulate around current vertex
        ... // (compute displacements)

iterate on vertices
    apply displacements
```



Advanced



Dualization



primal → dual

Dualization using Incremental Builder