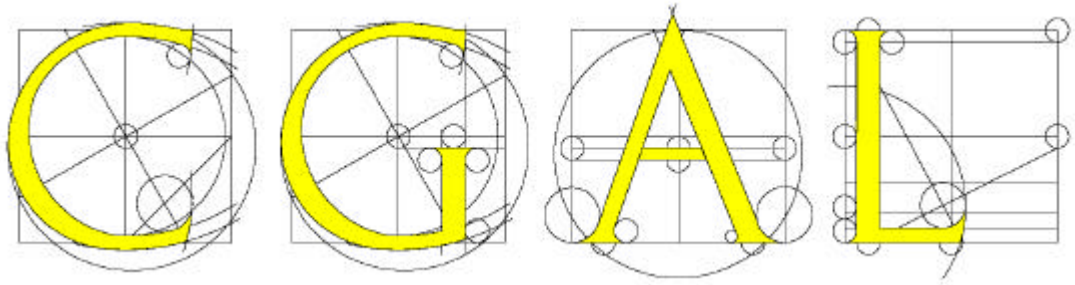
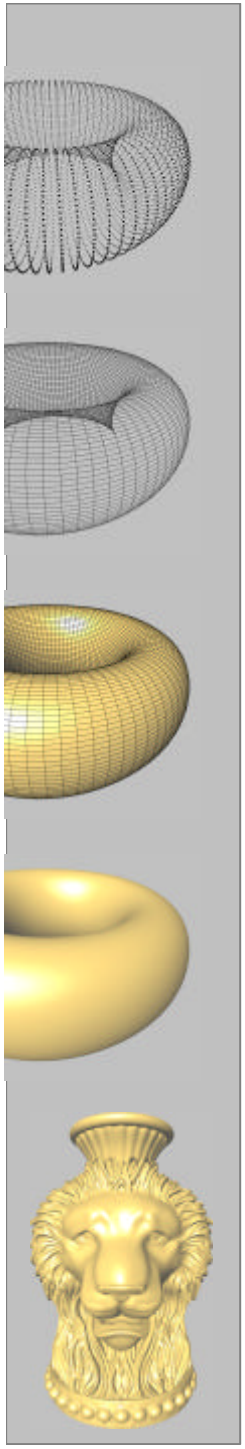
3D Polyhedral Surfaces in



Pierre Alliez

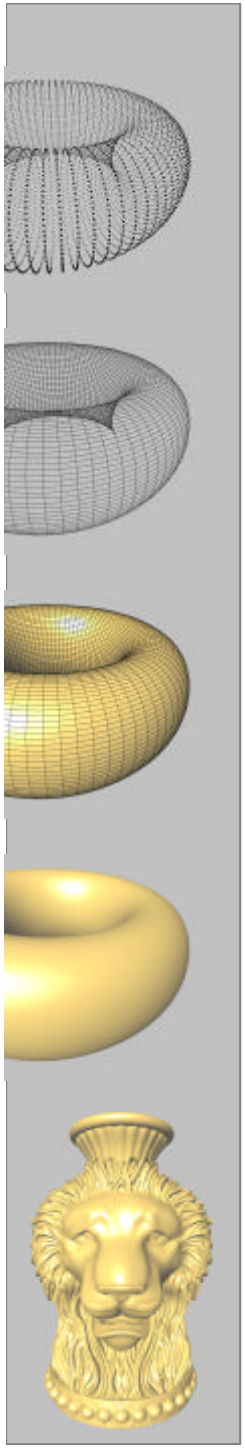
INRIA
SOPHIA ANTIPOLIS

http://www.cgal.org



Outline

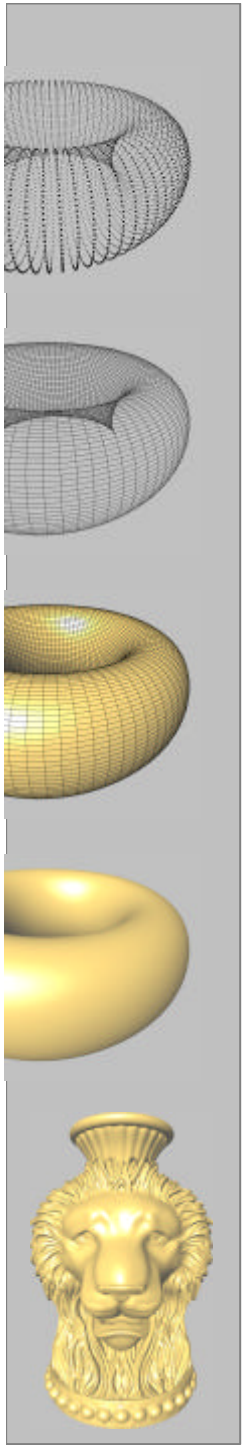
- Motivations
- Definition
- Halfedge Data Structure
- Traversal
- Euler Operators
- Customization
- Incremental Builder
- File I/O
- Examples
- Applications
- Exercises



Motivations

From **rendering**...

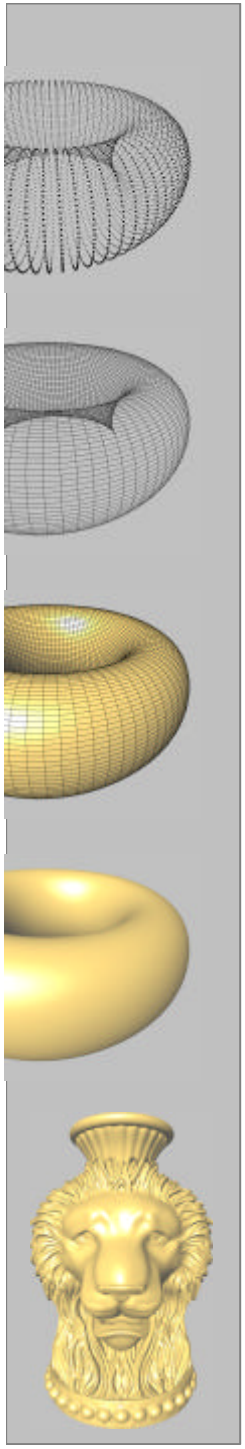
- Static
- Compact storage in array
- Traversal over all facets
- Attributes per facet/vertex



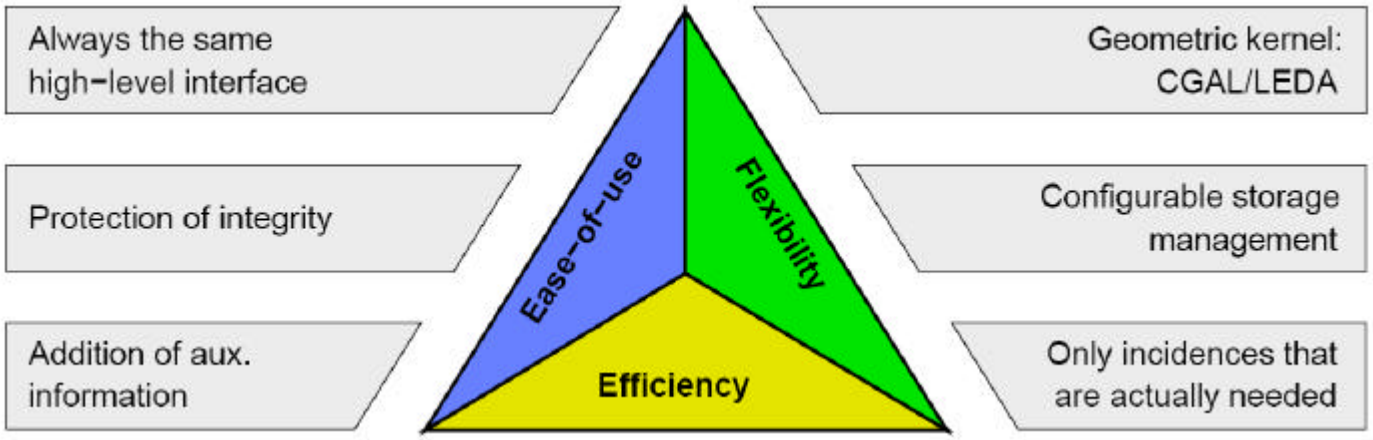
Motivations

...to **algorithms** on meshes

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

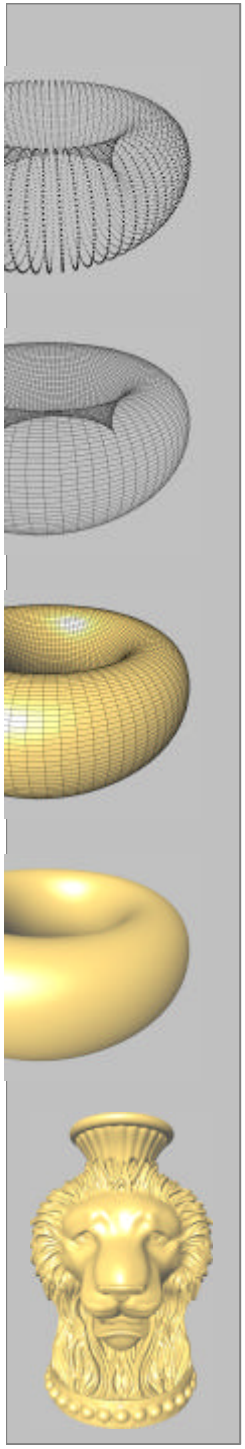


Design Goal



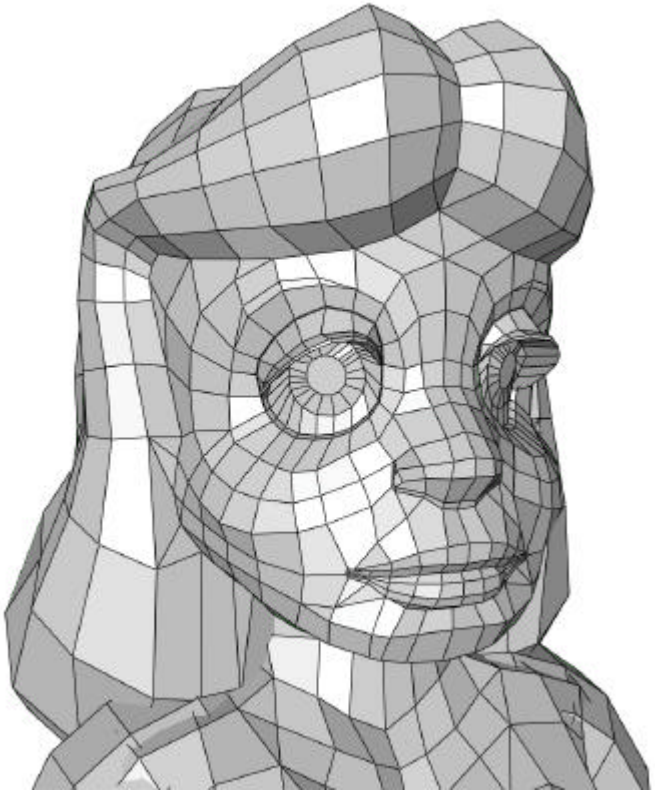
Method: paradigm of *Generic Programming*

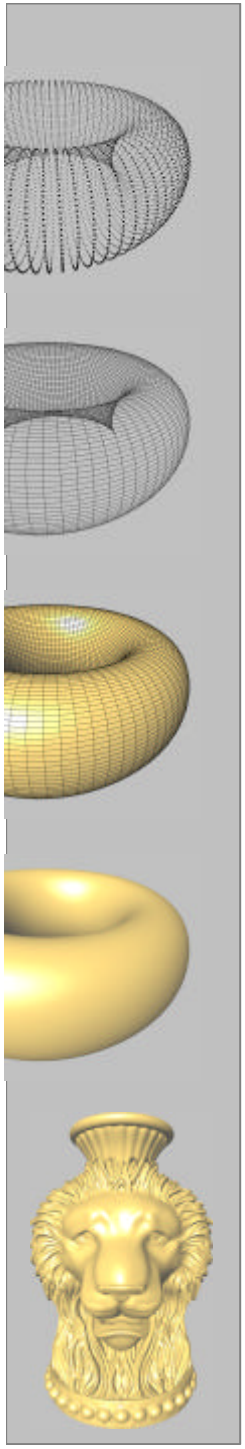
Example: STL



Definition

Polyhedral Surface: boundary representation of a polyhedron in $\mathbb{R}^3$.





Polyhedral Surface

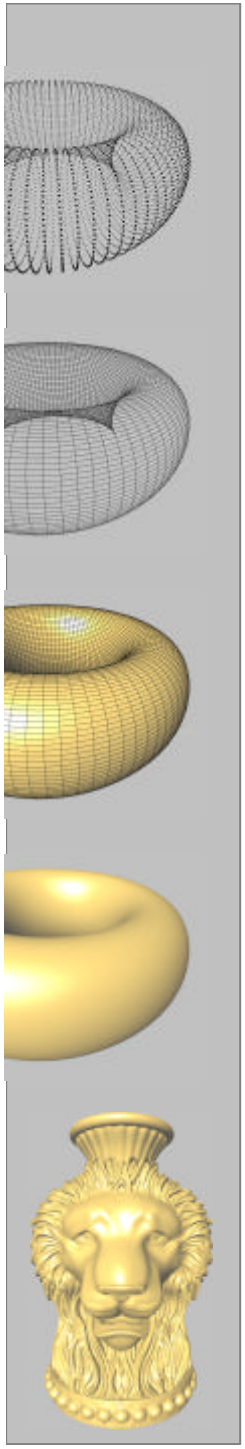
Represented by three sets **V**,**E**,**F** and an incidence relation on them, restricted to orientable 2-manifolds with boundary.

V = Vertices in $\mathbb{R}^3$

E = Edges, straight line segments.

F = Facets, simple, planar polygons without holes.

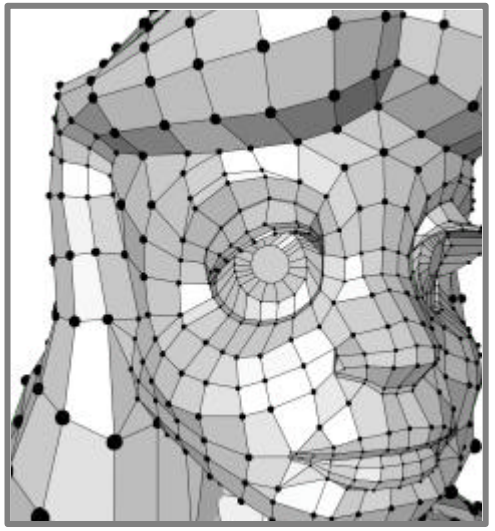
Can be extended: edges to curves, facets to curved surfaces.

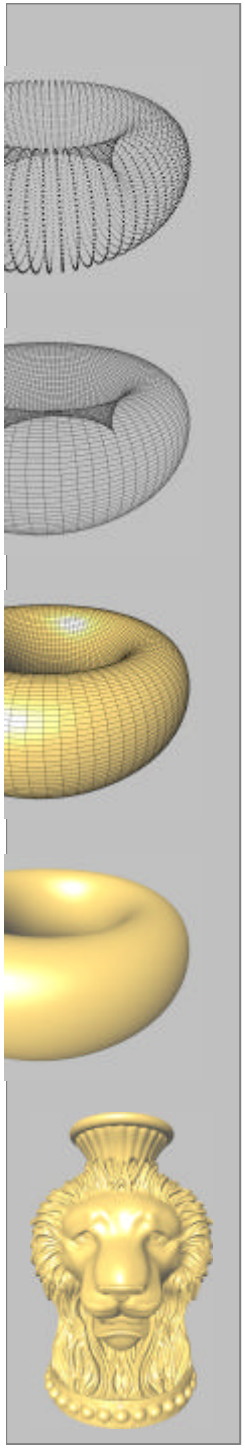


Polyhedral Surface

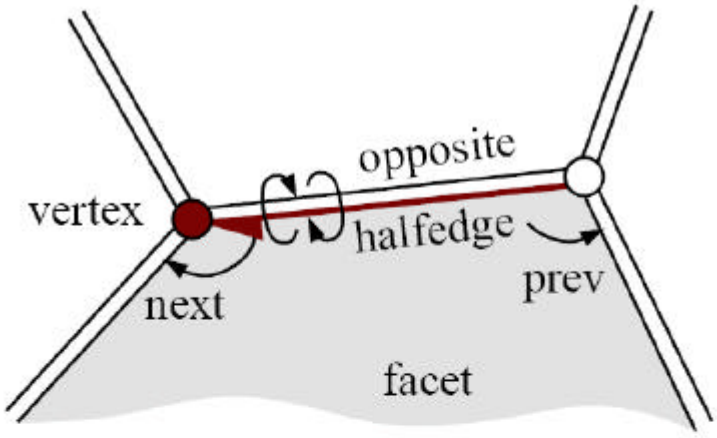
Represented by three sets **V**,**E**,**F** and an *incidence relation* on them, restricted to orientable 2-manifolds with boundary.

⇒ Edge-based data structure ?





Halfedge Data Structure



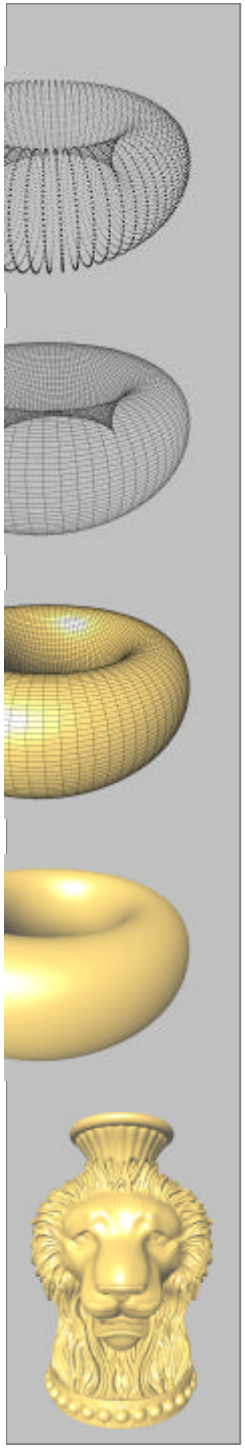
[Weiler 85], [Mäntylä 88], DCEL [de Berg, van Krefeld, Overmars, Schwarzkopf 97]

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.



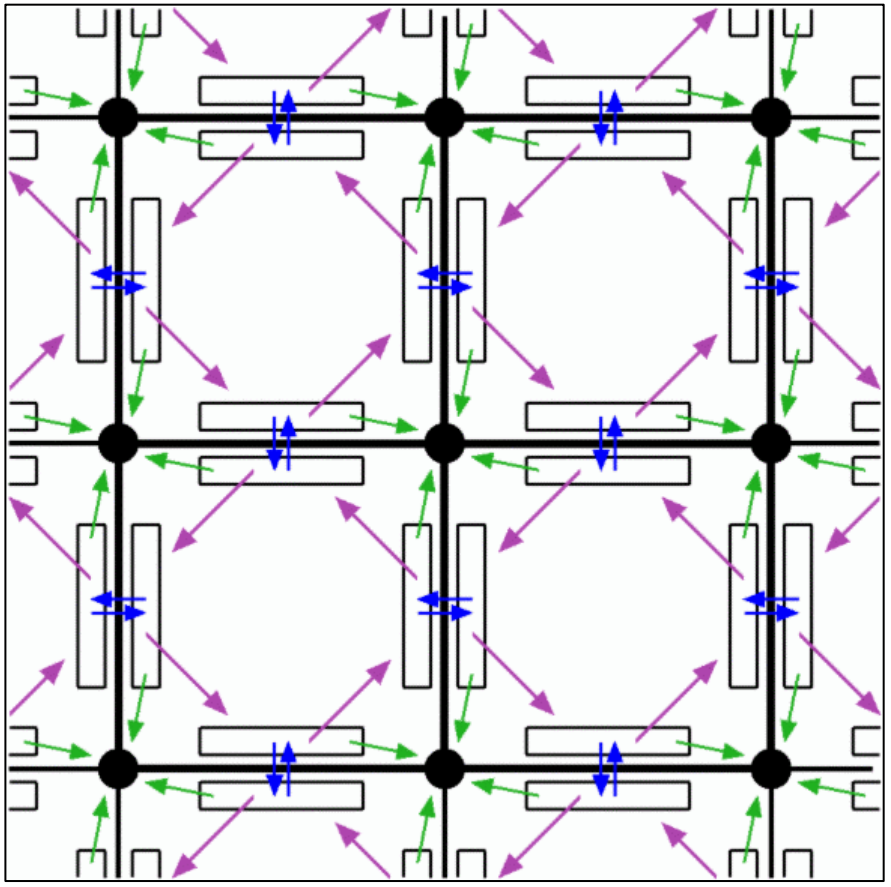
Halfedges

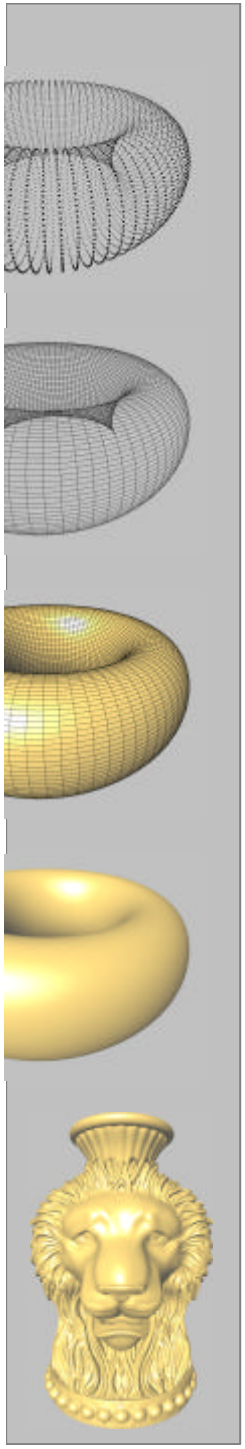
Require:

Oriented surface

Idea:

Consider 2/4 ways of accessing an edge

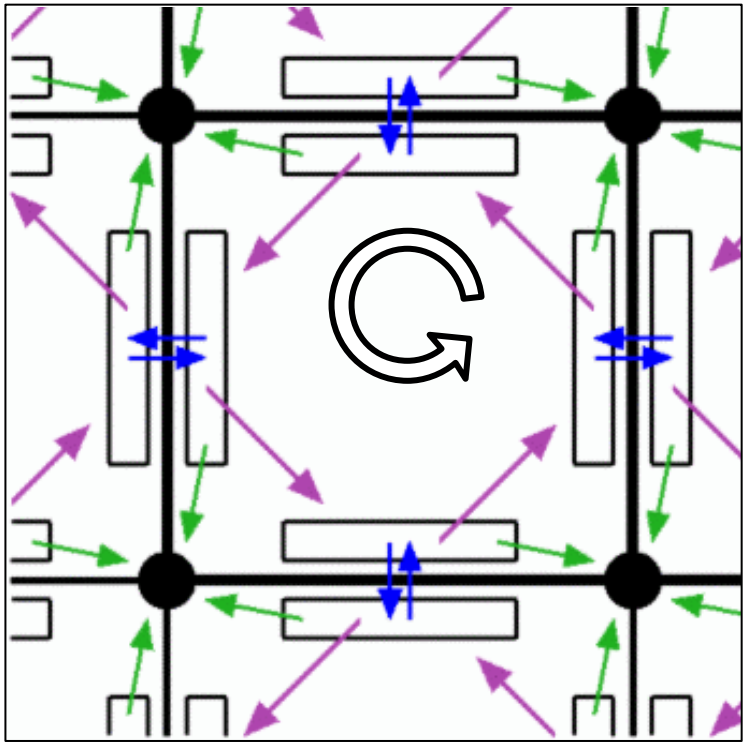




Halfedge

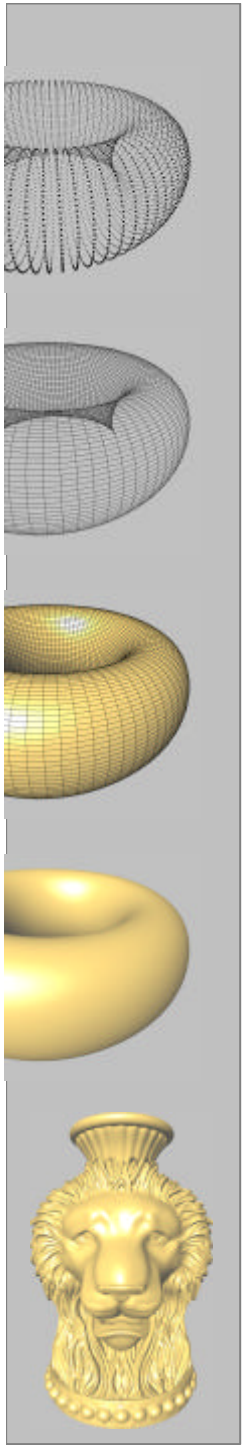
Associated with:

- 1 vertex
- 1 edge
- 1 facet



3 references:

- vertex
- opposite halfedge
- facet



Halfedges

Geometry:

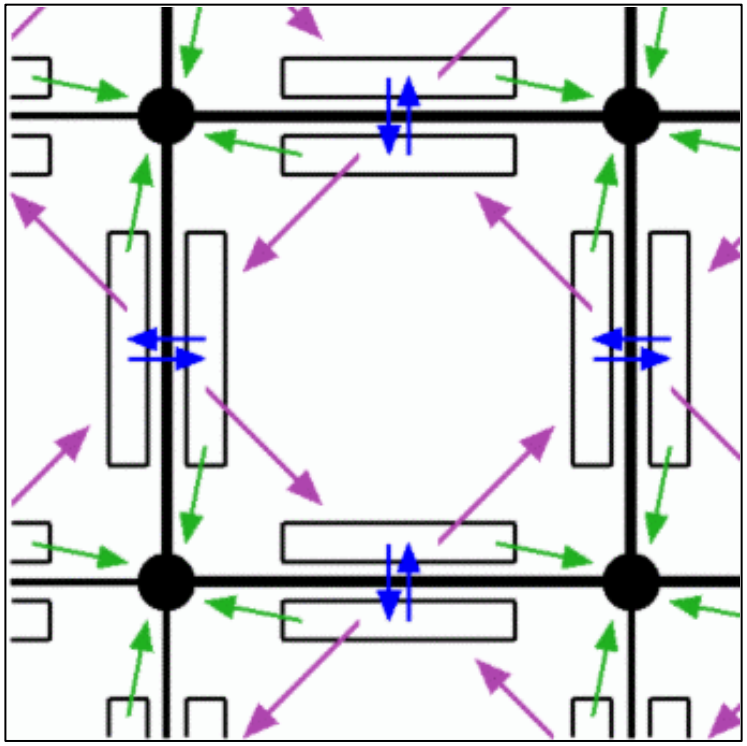
- vertices

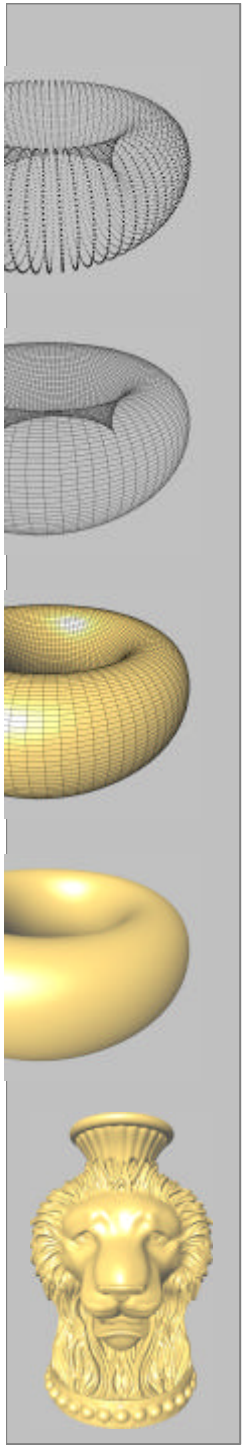
Attributes:

- on vertices
- on halfedges
- on facets

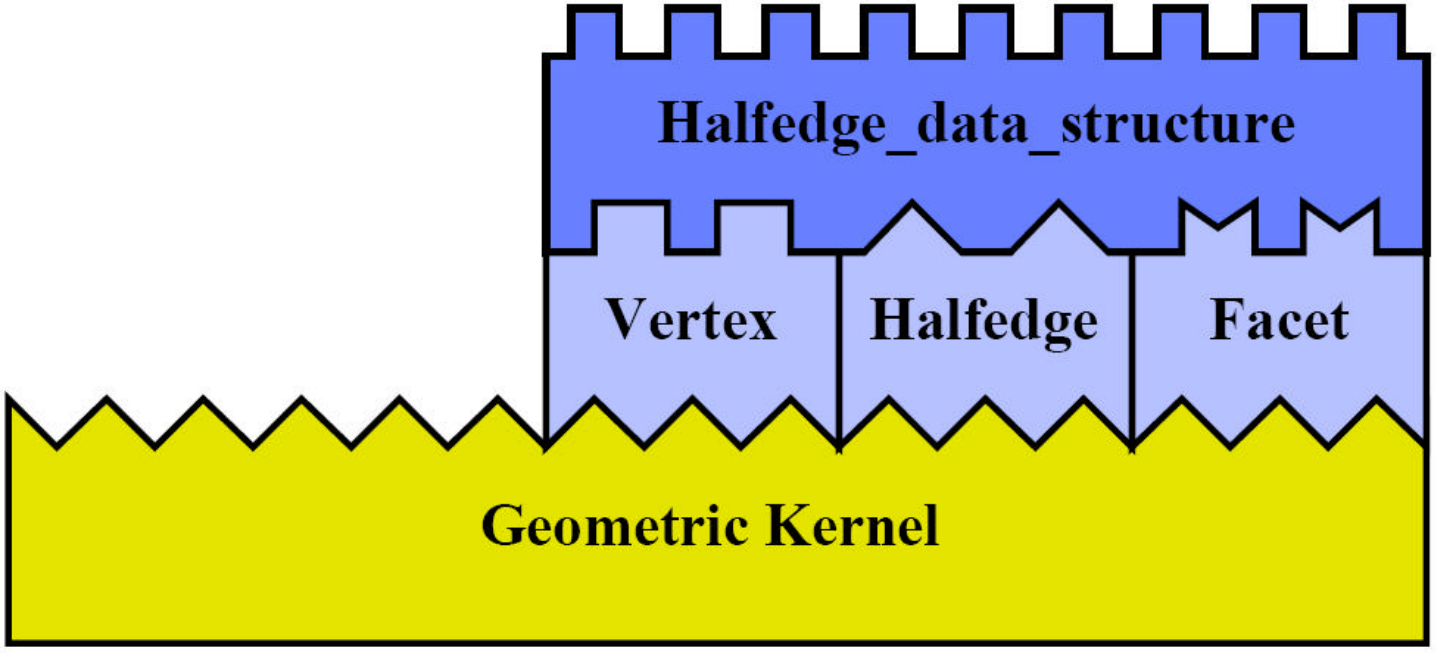
Connectivity:

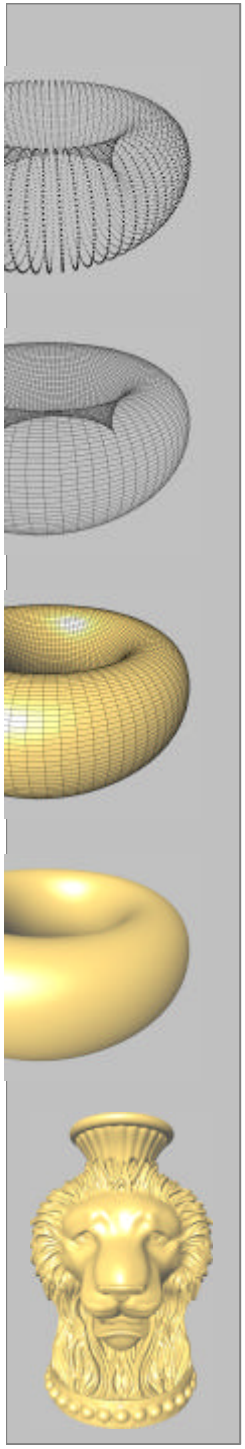
- halfedges only



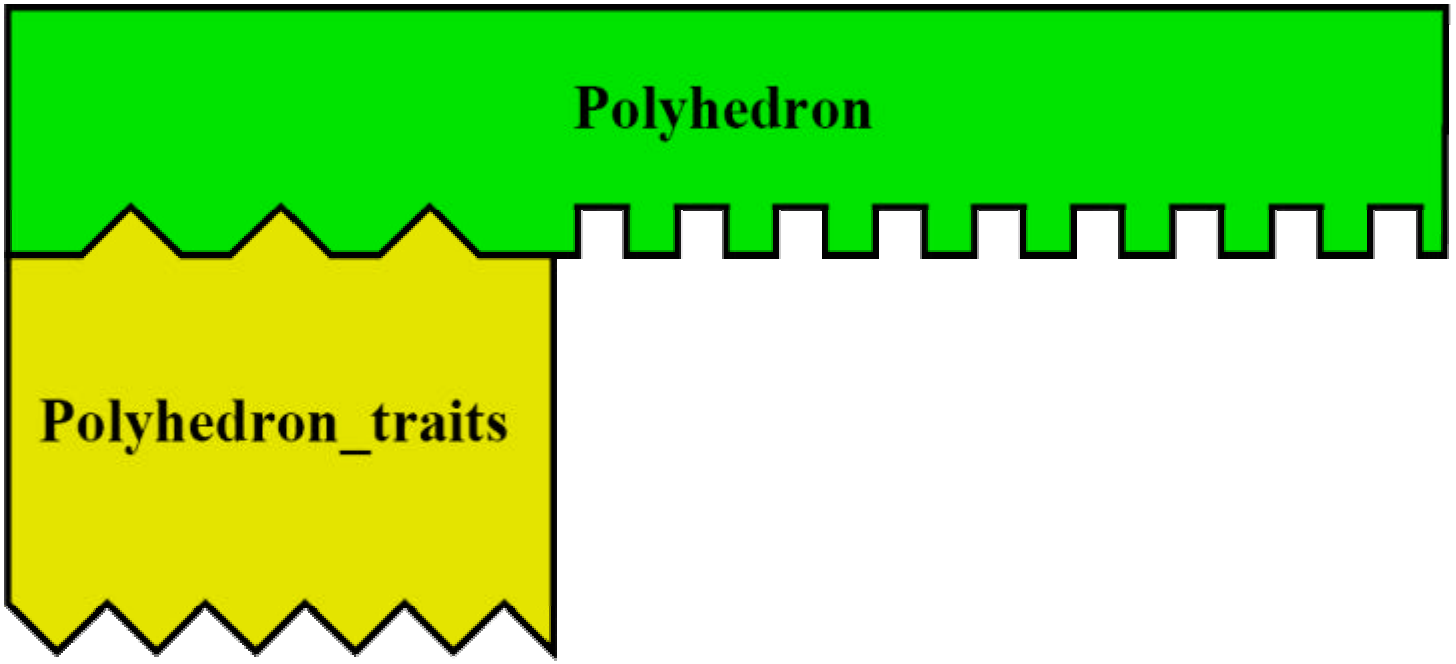


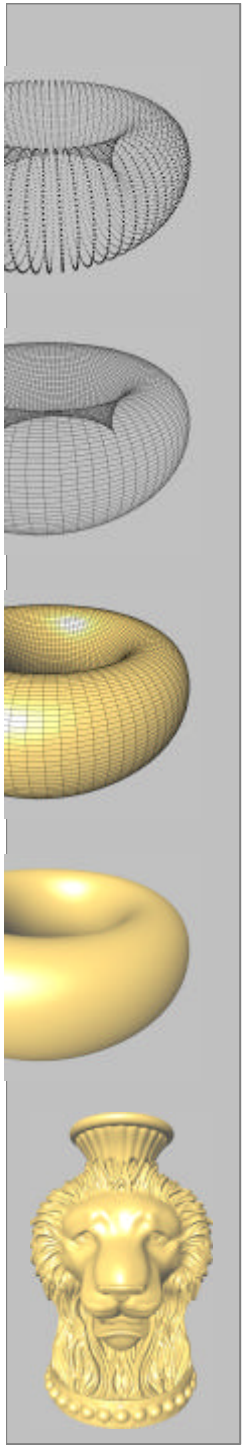
Building Blocks





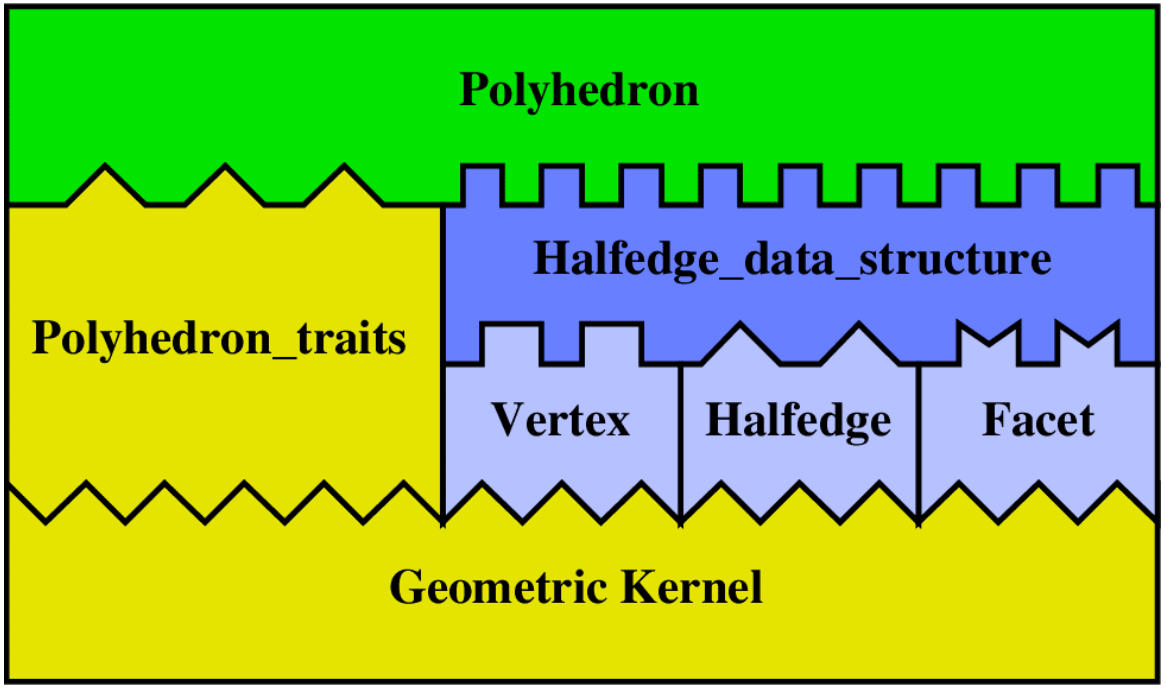
Building Blocks

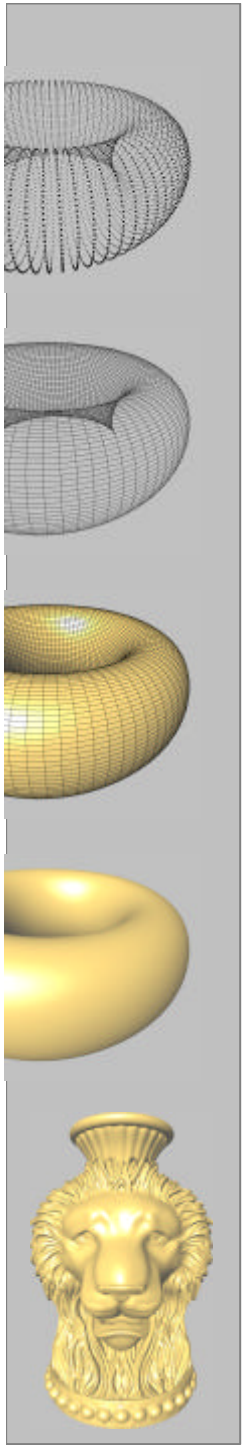




Polyhedral Surfaces

Building blocks assembled with C++ templates





Polyhedral Surface

Polyhedron

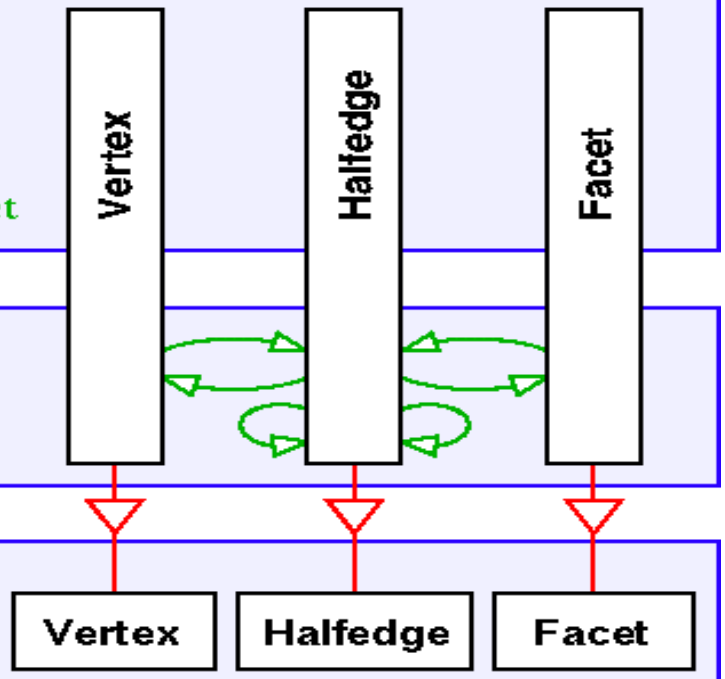
- provides ease- of- use
- protects combinatorial integrity
- defines circulators
- defines extended vertex, halfedge, facet

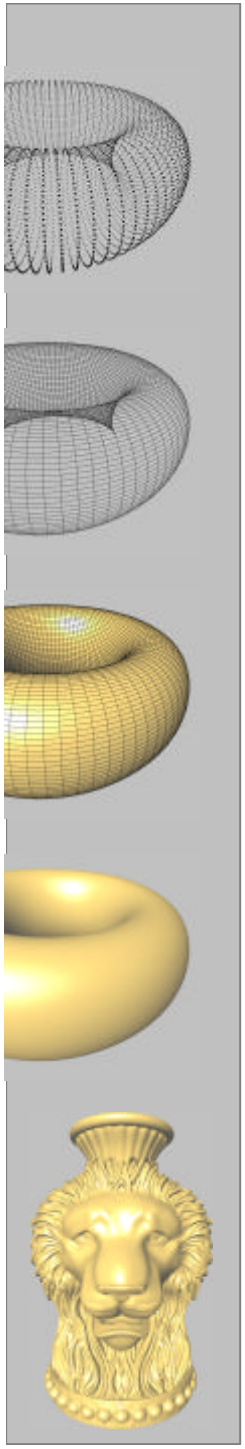
Halfedge_data_structure

- manages storage (container class)
- defines iterators

Items

- stores actual information
- contains user added data and functions



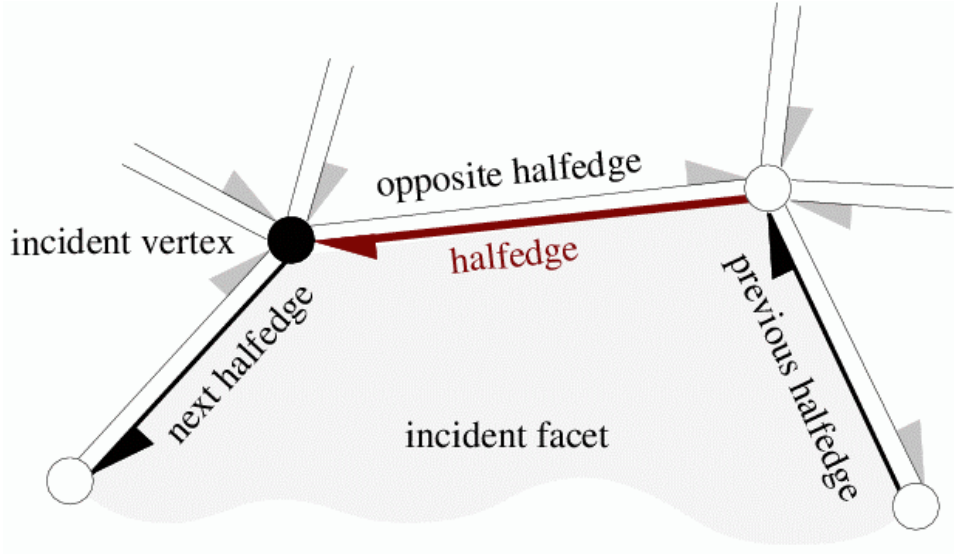


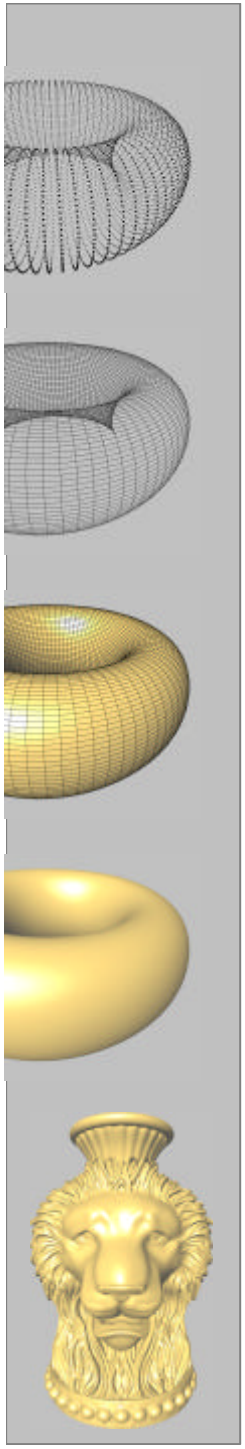
Default Polyhedron

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

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

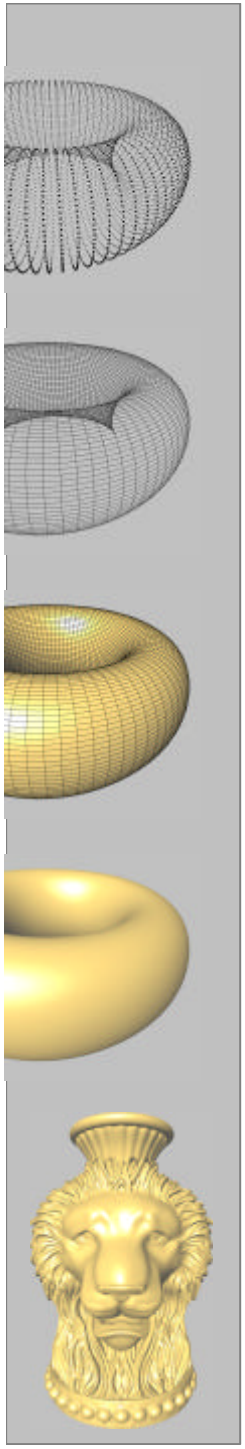
Facet	
Halfedge_handle	halfedge()
Plane&	plane()
Normal&	normal()
Color&	color()
.....	...





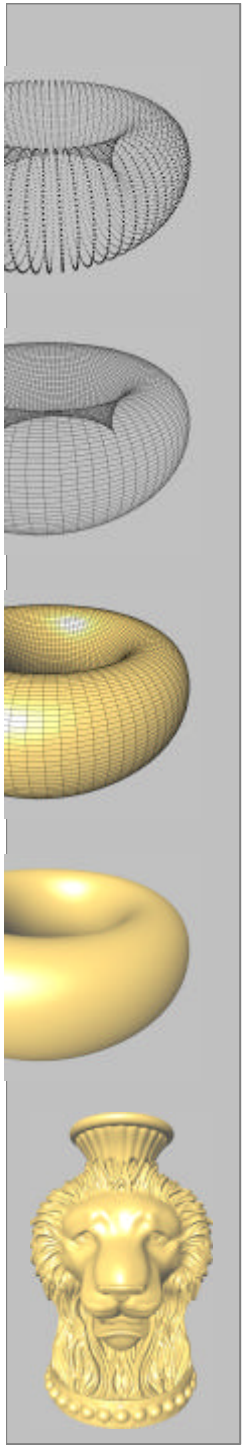
Default Polyhedron

```
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;
}
```



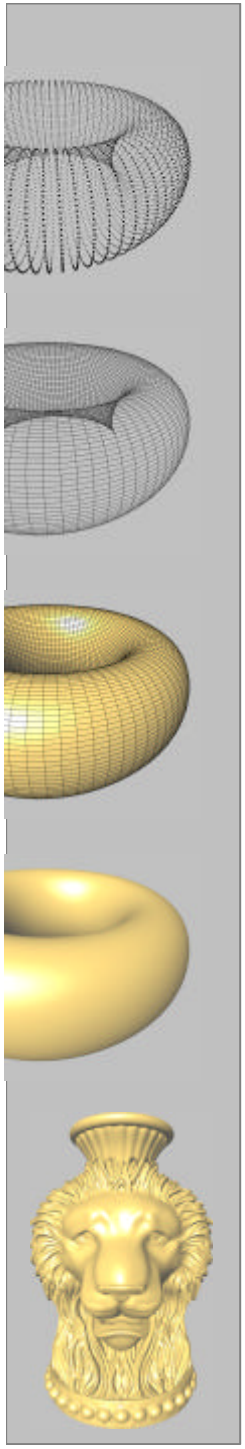
Extending Primitives

```
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;
      };
};
```



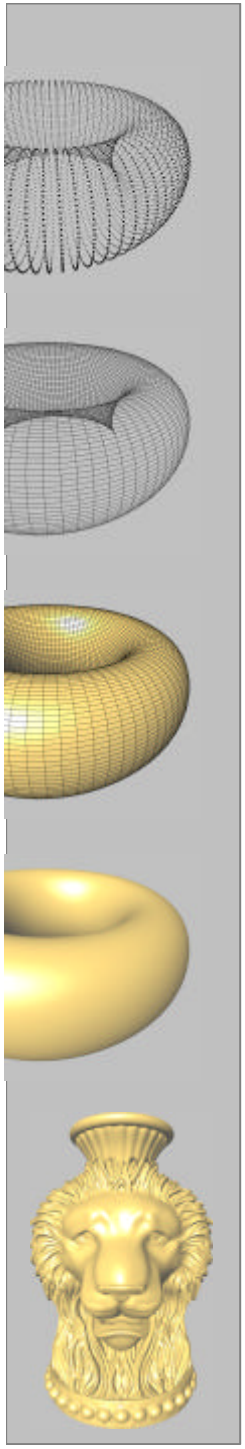
Add Color to Facets

```
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;
}
```

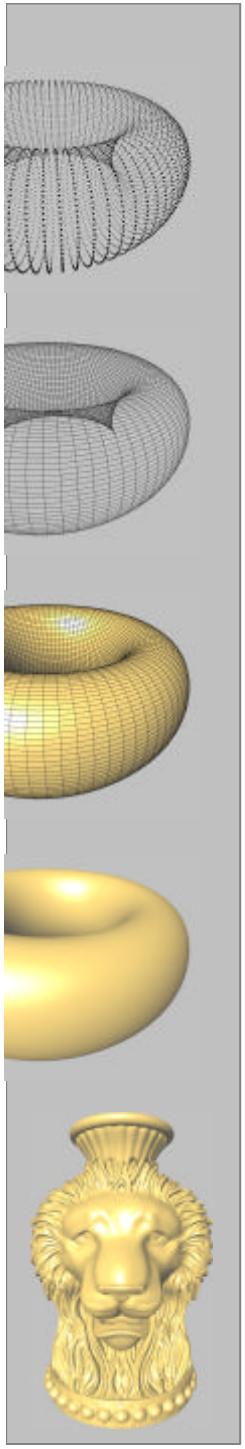


Add Color to Facets

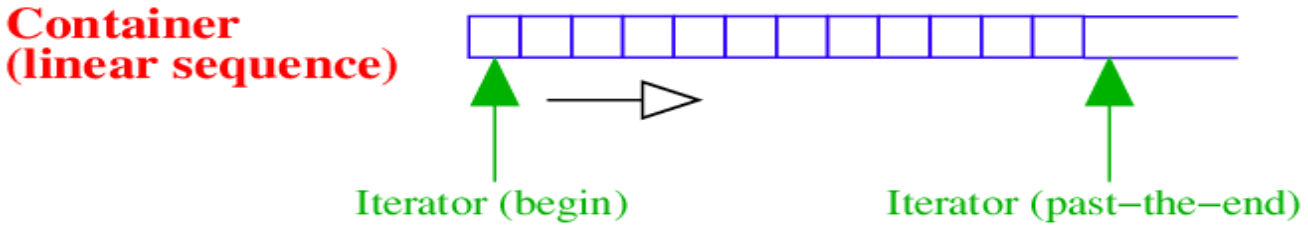
```
template <class Refs>
struct CFace : public CGAL::HalfedgeDS_face_base<Refs>{
    CGAL::Color color;
};

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;
```

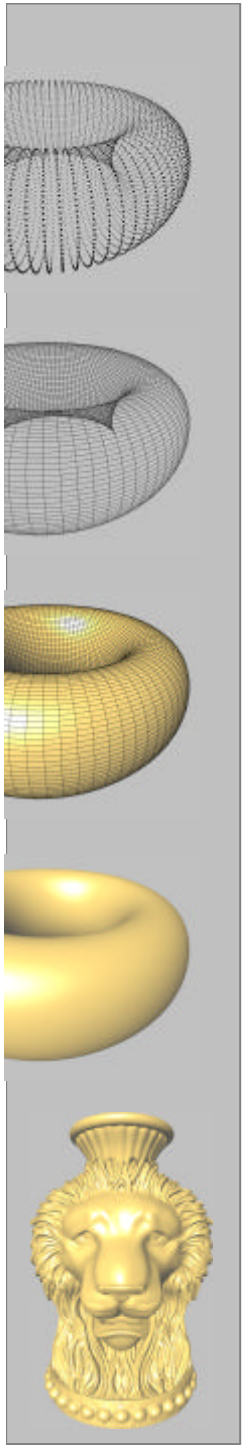


Traversal



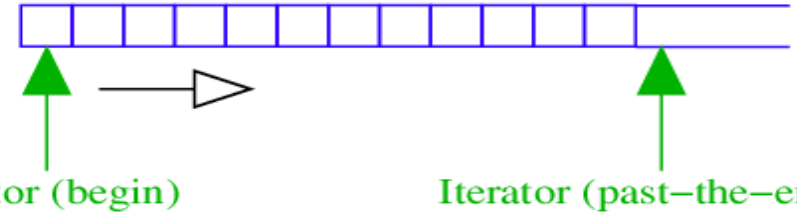
Iterators


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

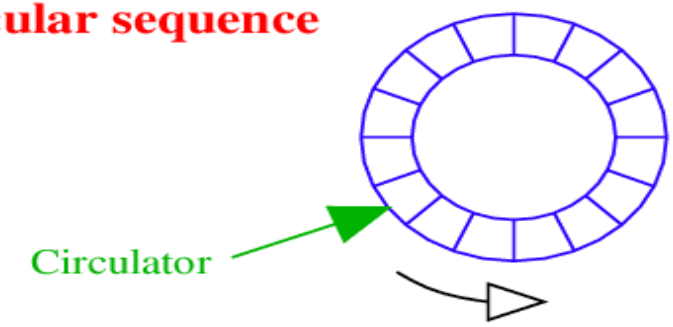


Iterators and Circulators

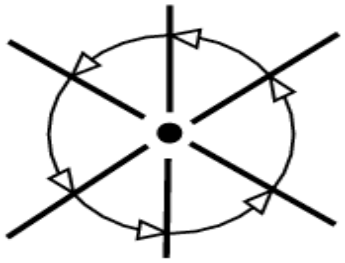
Container (linear sequence)

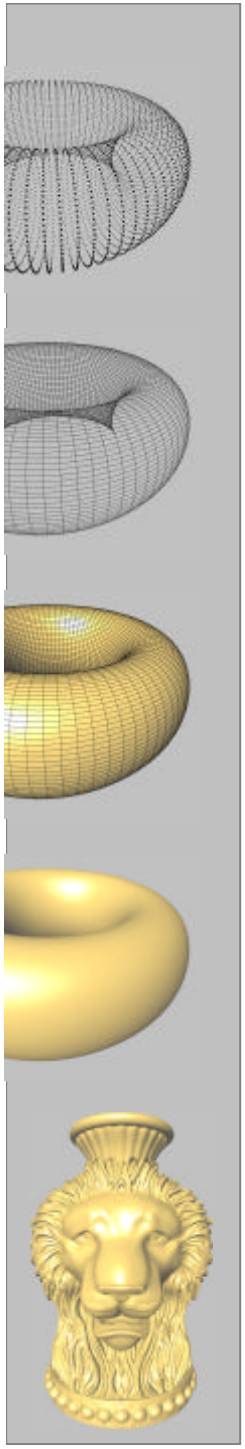


Circular sequence



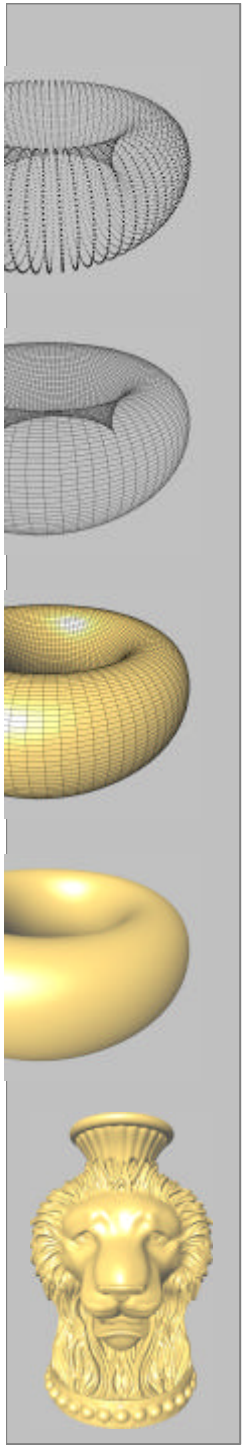
For example:
graph vertex





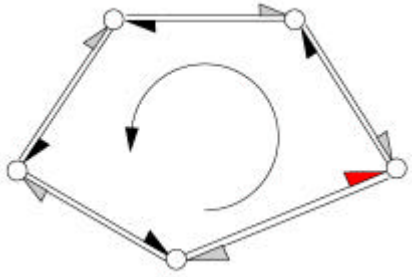
Iteration

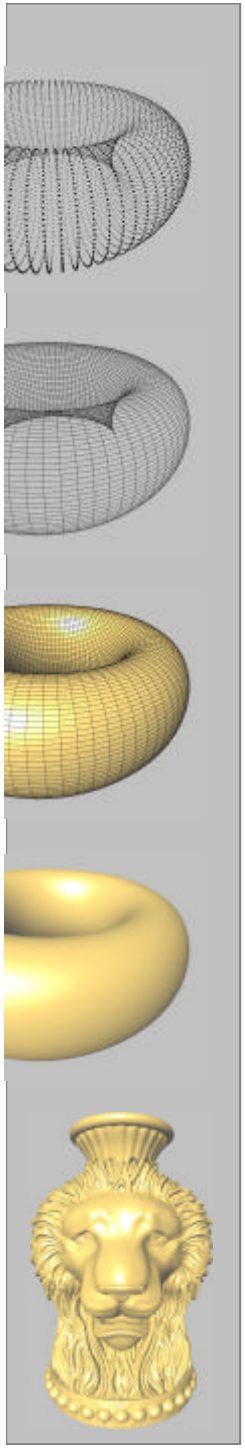
```
Vertex_iterator iter;
for(  iter = polyhedron.vertices_begin();
      iter != polyhedron.vertices_end();
      iter++)
{
  Vertex_handle hVertex = iter;
  // do something with hVertex
}
```



Circulation

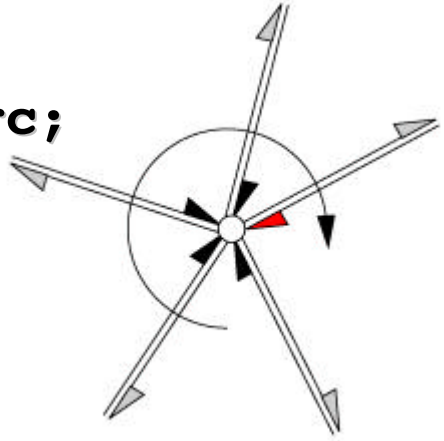
```
// circulate around hFacet
Halfedge_around_facet_circulator circ =
   hFacet->facet_begin();
Halfedge_around_facet_circulator end = circ;

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

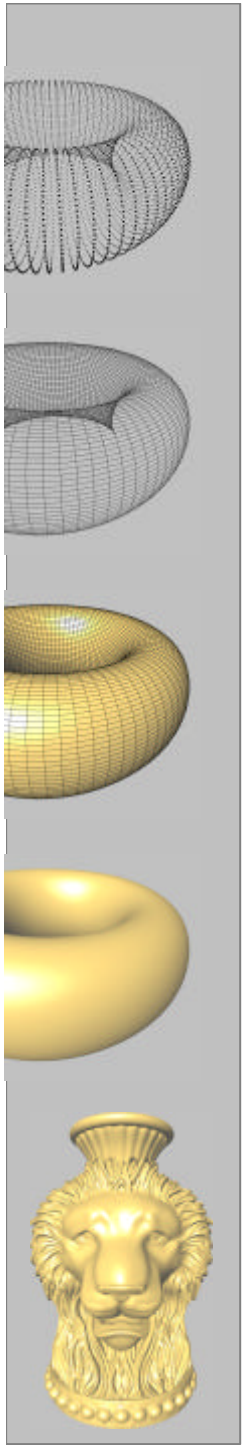




Circulation

```
// 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
}
```



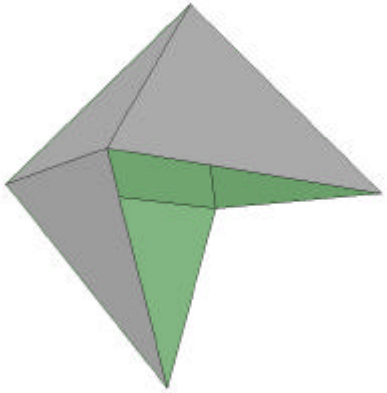


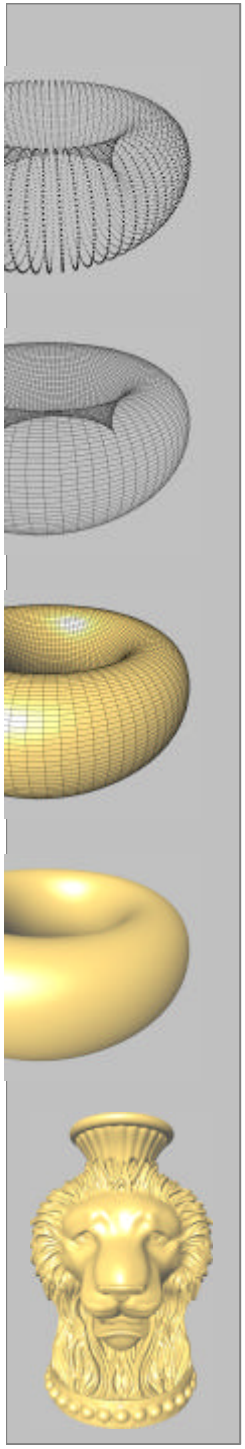
File I/O

I/O: OFF indexed format

Output: vrml1-2, iv, geomview

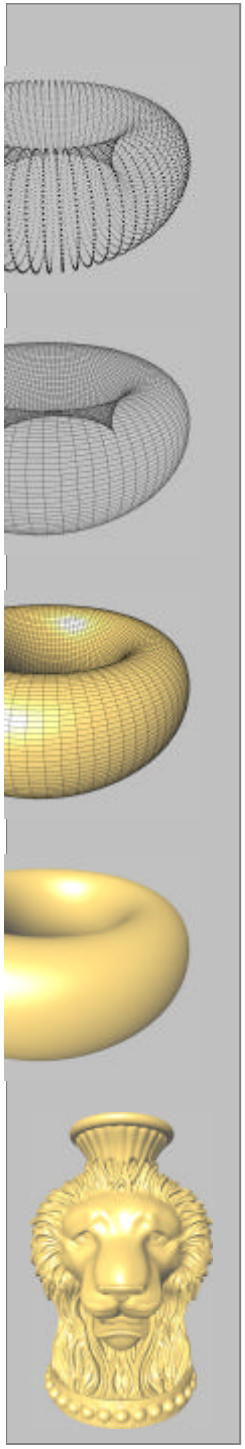
```
OFF
6 6 0
0.000000 1.686000 0.000000
1.192000 0.000000 -1.192000
-1.192000 0.000000 -1.192000
-1.192000 0.000000 1.192000
1.192000 0.000000 1.192000
0.000000 -1.68600 0.000000
3 0 4 1
3 1 5 2
3 2 3 0
3 1 2 0
3 3 4 0
3 3 2 5
```



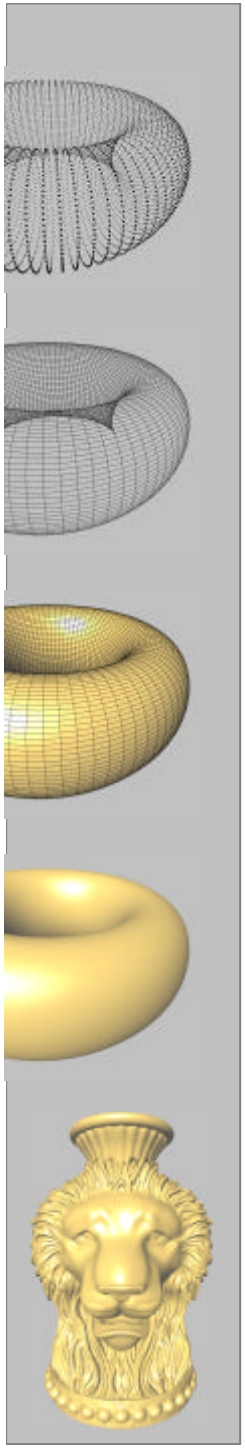


Applications

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



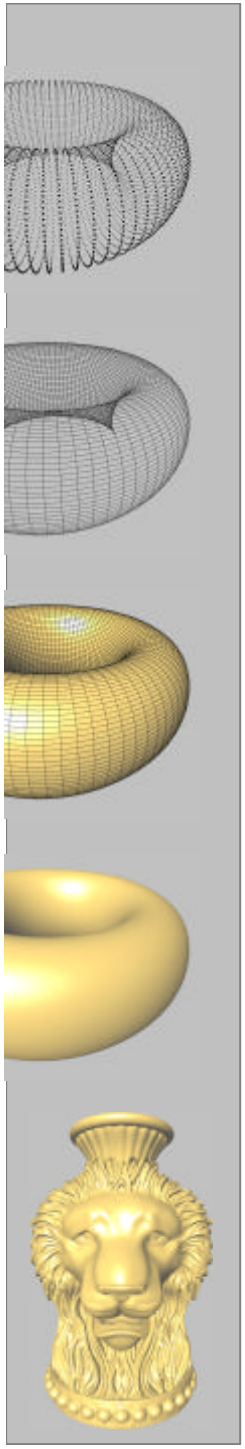
Warm-Up Exercices



Highlight Boundary Edges

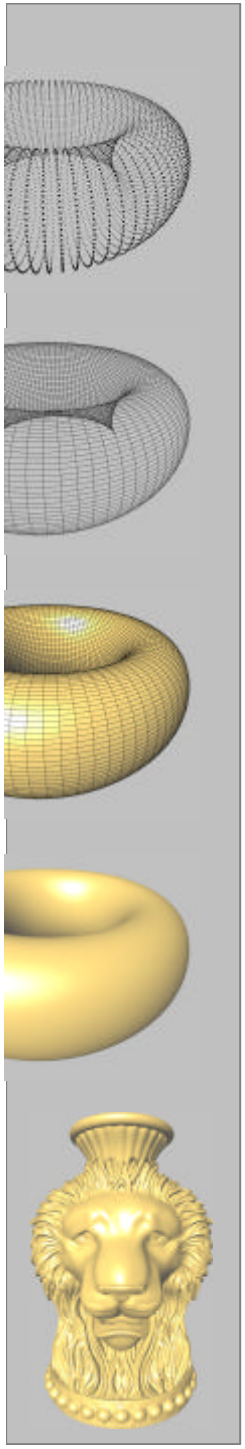
Notes:

```
bool halfedge->is_border();

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

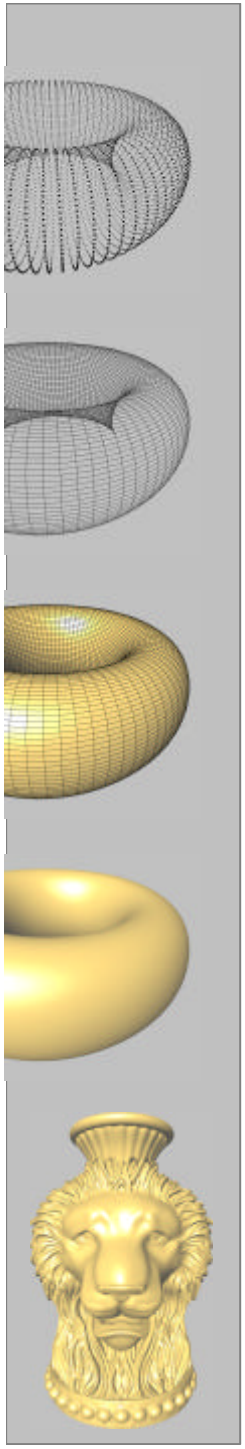


Render all Facets as Polygons

```
typedef Polyhedron::Facet_iterator Facet_iterator;
typedef Polyhedron::Halfedge_around_facet_circulator
                                    Halfedge_facet_circulator;

for (Facet_iterator i = P.facets_begin();
     i != P.facets_end();
     ++i)
{
    Halfedge_facet_circulator j = i->facet_begin();

    ::glBegin(GL_POLYGON);
    do
        ::glVertex3d(j->vertex()->point().x(),
                     j->vertex()->point().y(),
                     j->vertex()->point().z());
    while( ++j != i->facet_begin());
    ::glEnd();
}
```



Exercices Around Combinatorics



Combinatorial Genus

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