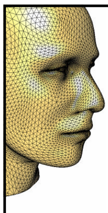
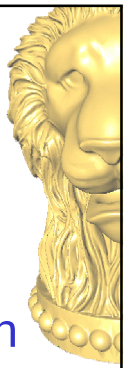




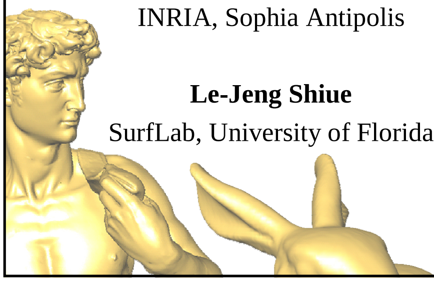
Tutorial: Algorithms on Meshes Based on the on CGAL Polyhedron

Pierre Alliez
INRIA, Sophia Antipolis

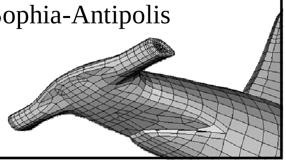
Lutz Kettner
MPI für Informatik, Saarbrücken

Le-Jeng Shiue
SurfLab, University of Florida

Radu Ursu
GeometryFactory, Sophia-Antipolis



July 19, 2004



Contents

- History and Overview of CGAL
- Robustness and the Exact-Geometric-Computing Paradigm
- Tutorial on Polyhedral Surfaces (Meshes)
- Demo of the Viewer

Contents

- History and Overview of CGAL
- Robustness and the Exact-Geometric-Computing Paradigm
- Tutorial on Polyhedral Surfaces (Meshes)
- Demo of the Viewer

Motivation

Computational Geometry Impact Task Force Report 1996
"*Application Challenges to Computational Geometry*" had four key recommendations:

1. Production and distr. of usable (and useful) geometric codes
2. Interdisciplinary forums
3. Experimentation
4. Reward structure for implementations in academia

Computational Geometry Algorithms Library

www.cgal.org

- Project Goal

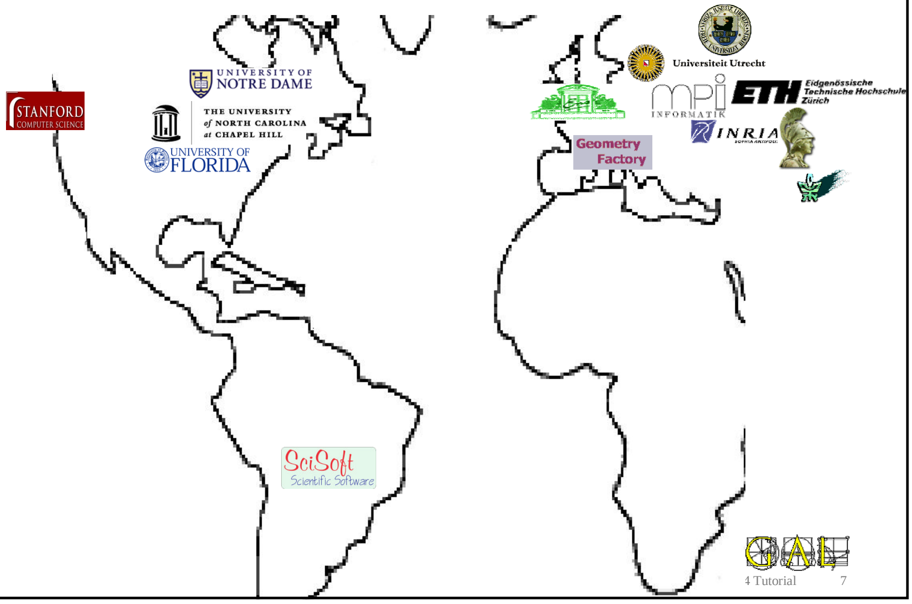
  "make the large body of geometric algorithms developed in the field of computational geometry available for industrial applications."

- C++ Library: Release 3.0.1 (Jan. 2004)

Development Started 1995

- **ETH Zurich (Switzerland),**
- **Freie Universität Berlin (Germany),**
- **INRIA Sophia-Antipolis (France),**
- **Martin-Luther-Universität Halle-Wittenberg (Germany),**
- **Max-Planck-Institut für Informatik (Germany),**
- **RISC Linz (Austria),**
- **Tel Aviv University (Israel),**
- **Utrecht University (The Netherlands).**

CGAL Developers 2004



CGAL in Numbers

- 1200 C++ classes, 300 KLOC, 1100 page manual
- ~40 developer years
- supported platforms
  - Linux, Irix, Solaris, Windows (OS X)
  - g++, SGI CC, SunPro CC, VC7, Intel
- Release cycle of ~12 months
- 6000 downloads per year
- 795 Users registered on user list
- 43 Developers registered on developer list

Development Process

- Editorial board reviews submissions
- Developer manual
- Own manual tools; LaTeX source ➔PS, PDF, HTML
- 1-2 developer meetings per year, 1 week long
- CVS server for version management
- Bug tracking system
- Three internal releases per week
- Automatic test suites for different compilers/platforms

Open Source License since 3.0 (2003)

- A guarantee that CGAL remains free
- Promote CGAL as a standard for users
- Opens CGAL for new contributions

- Different licenses for different parts
  - **LGPL** for Kernel and Support Library
  - **QPL** for Basic Library

  - **Commercial Licenses** from CGAL Start-up **GeometryFactory**, founded by Andreas Fabri in 2003

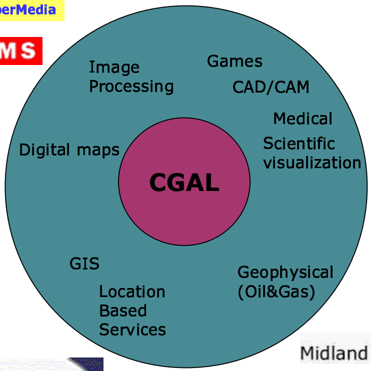
Commercial Customers

TOSHIBA

HyperMedia

BAE SYSTEMS

QinetiQ



Leica Geosystems

TruePosition

Agip

TGS

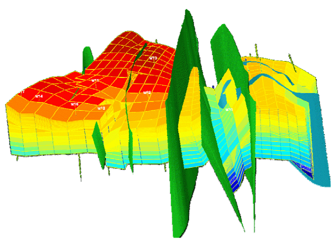
Midland Valley

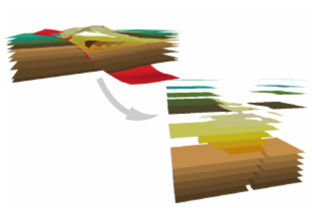
Riesen&Stettler

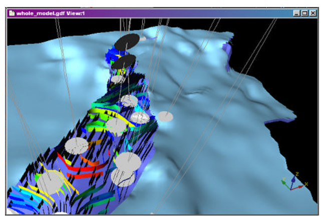
Midland Valley

Midland Valley

- 4DVista
  for modelling, visualization, and simulation of geological structures.

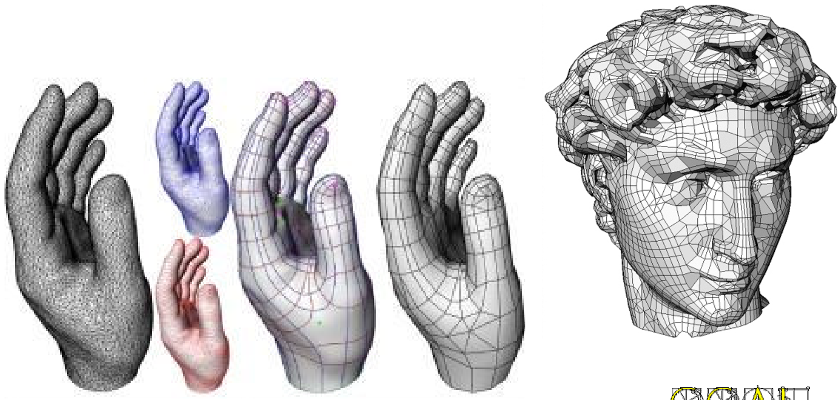






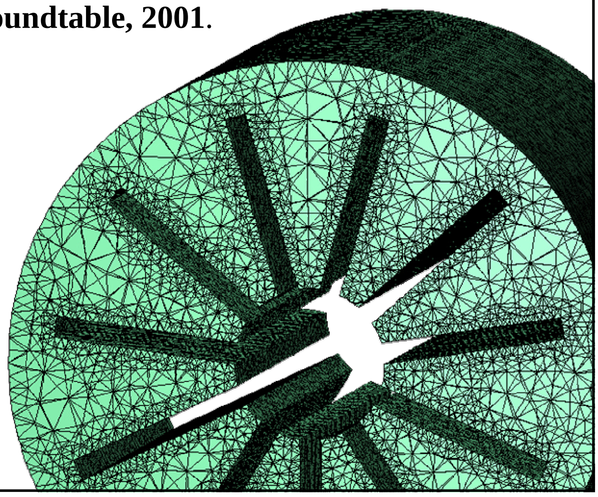
Academic Users of the Polyhedron

- *Anisotropic Polygonal Remeshing.*
  P. Alliez, D.Cohen-Steiner, O.Devillers, B.Levy, and M.Desbrun.
  SIGGRAPH 2003.



Academic Users of the Polyhedron

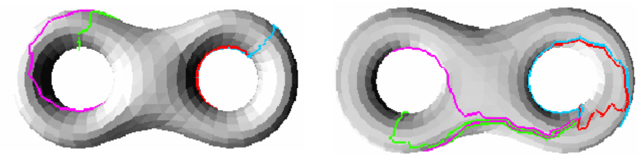
- *Efficient and Robust Algorithms for Overlaying Surface Meshes*.
  X. Jiao and M. T. Heath.
  Intern. Meshing Roundtable, 2001.



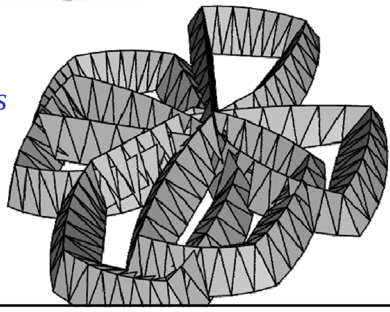
Academic Users of the Polyhedron

Computing a Canonical Polygonal Schema of an Orientable Triangulated Surface. F. Lazarus, M. Pocchiola, G. Vegter and A. Verroust.

Annu. ACM Sympos. Comput. Geom., 2001.



Controlled perturbation for arrangements of polyhedral surfaces with application to swept volumes.
S. Raab.
Annu. ACM Sympos. Comput. Geom., 1999.



Structure of CGAL

Basic Library
Algorithms and data structures

Geometric Kernel
Geom. primitives, predicates, operations

Core Library
Configuration, assertions, ...

Support Library
Visualization,
File I/O,
Numbertypes,
Generators,
...

Geometric Kernel

Primitives

2D, 3D, dD

- Point, Vector
- Line, Ray, Segment
- Triangle
- Iso_rectangle
- Bbox
- Circle

- Affine transformation

Predicates

- Order predicates
- Orientation test
- Incircle test

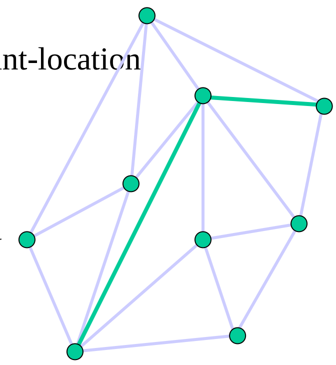
Constructions

- Center point
- Intersection
- Squared distance

SGP'04 Tutorial 17

Basic Library: Triangulation

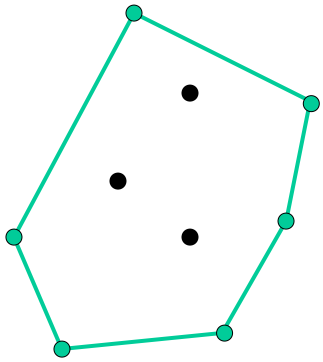
- Triangle-based data-structure
  - compact, fast, walk for point location
- Delaunay/Voronoi
  - Delaunay hierarchy for fast point-location
- Constrained Delaunay
  - => terrain triangulations
- Regular triangulations
  - Weighted points, bio-geometry
- Tetrahedrization in 3D



SGP'04 Tutorial 18

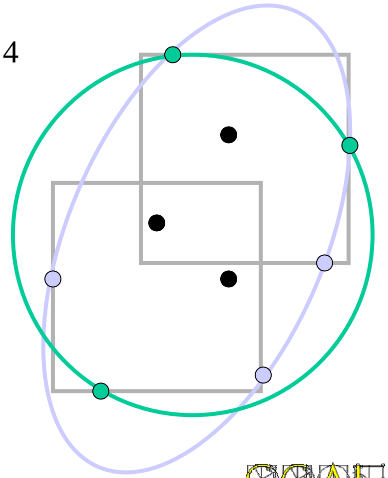
Basic Library: Convex Hull

- 5 different algorithms for 2D
- 3 different algorithms for 3D
  - Static (quickhull)
  - Randomized incremental
  - Dynamic (tetrahedrization)



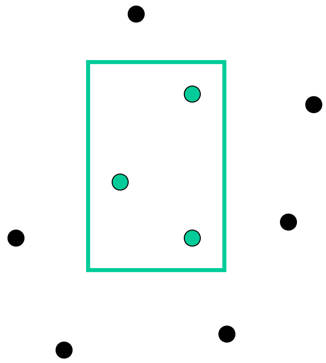
Basic Library: Geometric Optimization

- Smallest enclosing circle and ellipse in 2D
- Smallest enclosing sphere in dD
- Rectangular p center, 2 <= p <= 4
- Width in 2D and 3D



Basic Library: Search Structures

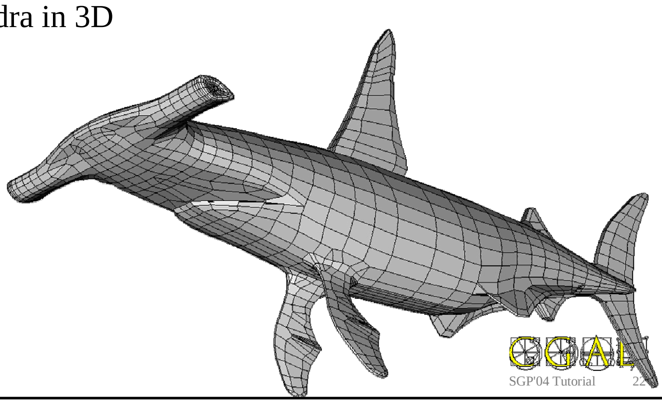
- Range-, segment-, KD-tree
- Arbitrary dimension
- Mixed segment-range-trees
- Static
- Window query, enclosing query
- Nearest neighbors
- Approximate nearest neighbors



SGP'04 Tutorial 21

Basic Library: Halfedge Data-Structure

- Polyhedral surface: orientable 2-manifolds with boundary
- Planar map and arrangements
- Nef polygons; closed under Boolean operations
- Nef polygons embedded on the sphere
- Nef polyhedra in 3D



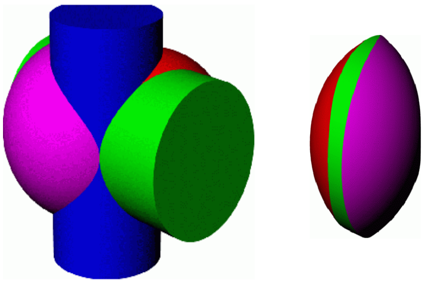
SGP'04 Tutorial 22

Contents

- History and Overview of CGAL
- Robustness and the Exact-Geometric-Computing Paradigm
- Tutorial on Polyhedral Surfaces (Meshes)
- Demo of the Viewer

SGP'04 Tutorial 23

Intersection of Four Simple Solids

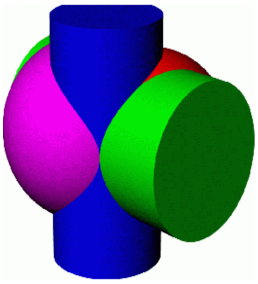


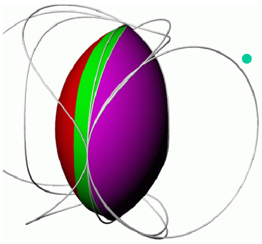
- output is a combinatorial object plus coordinates (not a point set)

- Rhino3D:
  - $(((s_1 \cap s_2) \cap c_2) \cap c_1) \rightarrow$ successful
  - $(((c_1 \cap c_2) \cap s_1) \cap s_2) \rightarrow$ ``Boolean operation failed"
- geometric problems are non-continuous functions from input to output

SGP'04 Tutorial 24

Intersection of Four Simple Solids





- output is a combinatorial object plus coordinates (not a point set)

- Rhino3D:
  - ((($s_1 \cap s_2$) $\cap$ c_2) $\cap$ c_1) → successful
  - ((($c_1 \cap c_2$) $\cap$ s_1) $\cap$ s_2) → ``Boolean operation failed"
- geometric problems are non-continuous functions from input to output

2D-Orientation of Three Points

Orientation(p, q, r) = $\text{sign}((q_x\text{-}p_x)(r_y\text{-}p_y) - (q_y\text{-}p_y)(r_x\text{-}p_x))$

$$p: \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}$$

$$q: \begin{pmatrix} 12 \\ 12 \end{pmatrix}$$

$$r: \begin{pmatrix} 24 \\ 24 \end{pmatrix}$$

2D-Orientation of Three Points

Orientation(p, q, r) = sign((q_x-p_x)(r_y-p_y) – (q_y-p_y)(r_x-p_x))

$$p : \begin{pmatrix} 0.5 + x \cdot u \\ 0.5 + y \cdot u \end{pmatrix}$$

$$q : \begin{pmatrix} 12 \\ 12 \end{pmatrix}$$

$$r : \begin{pmatrix} 24 \\ 24 \end{pmatrix}$$

$$0 \le x, y < 256, u = 2^{-53}$$

256x256 pixel image

red=pos., yellow=0, blue=neg.

orientation evaluated with `double`

SGP'04 Tutorial 27

2D-Orientation of Three Points

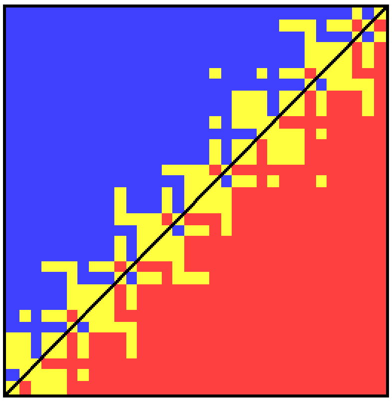
Orientation(p, q, r) = sign((q_x-p_x)(r_y-p_y) – (q_y-p_y)(r_x-p_x))

$$p : \begin{pmatrix} 0.5 + x \cdot u \\ 0.5 + y \cdot u \end{pmatrix}$$

$$q : \begin{pmatrix} 12 \\ 12 \end{pmatrix}$$

$$r : \begin{pmatrix} 24 \\ 24 \end{pmatrix}$$

$$0 \le x, y < 256, u = 2^{-53}$$



256x256 pixel image

red=pos., yellow=0, blue=neg.

orientation evaluated with `double`

SGP'04 Tutorial 28

2D-Orientation of Three Points

Orientation(p, q, r) = sign((q_x-p_x)(r_y-p_y) – (q_y-p_y)(r_x-p_x))

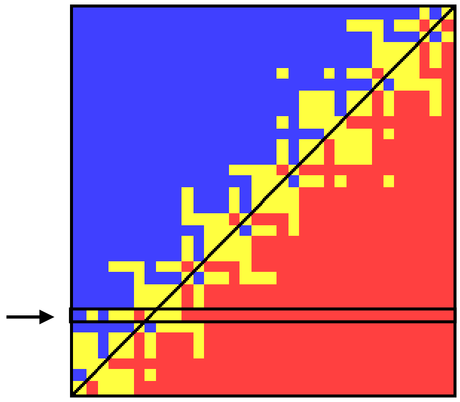
$$p: \begin{pmatrix} 0.5 + x \cdot u \\ 0.5 + y \cdot u \end{pmatrix}$$

$$q: \begin{pmatrix} 12 \\ 12 \end{pmatrix}$$

$$r: \begin{pmatrix} 24 \\ 24 \end{pmatrix}$$

$$0 \le x, y < 256, u = 2^{-53}$$

Keep y-coordinate fix



2D-Orientation of Three Points

Orientation(p, q, r) = sign((q_x-p_x)(r_y-p_y) – (q_y-p_y)(r_x-p_x))

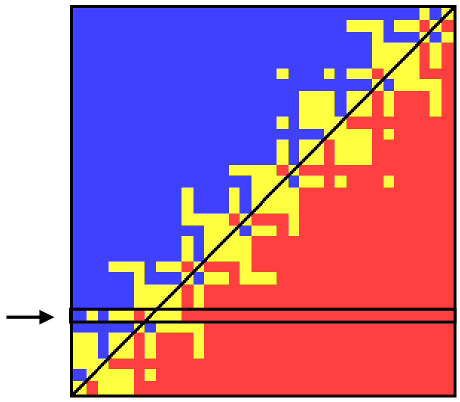
$$p: \begin{pmatrix} 00000.1... + x \cdot u \\ 00000.1... + y \cdot u \end{pmatrix}$$

$$q: \begin{pmatrix} 01100.0... \\ 01100.0... \end{pmatrix}$$

$$r: \begin{pmatrix} 11000.0... \\ 11000.0... \end{pmatrix}$$

$$0 \le x, y < 256, u = 2^{-53}$$

Keep y-coordinate fix:
block sizes of 2^5 and 2^4



2D-Orientation of Three Points

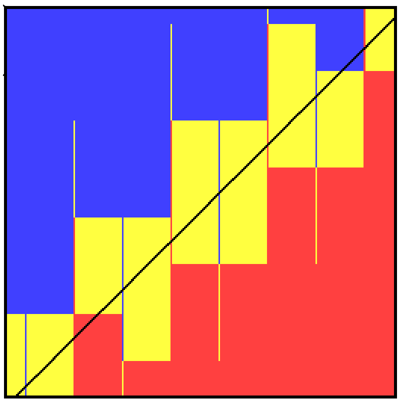
Orientation(p, q, r) = sign((q_x-p_x)(r_y-p_y) – (q_y-p_y)(r_x-p_x))

$$p: \begin{pmatrix} 0.50000000000002531 + x \cdot u \\ 0.50000000000001710 + y \cdot u \end{pmatrix}$$

$$q: \begin{pmatrix} 17.300000000000001 \\ 17.300000000000001 \end{pmatrix}$$

$$r: \begin{pmatrix} 24.00000000000005000000 \\ 24.0000000000000517765 \end{pmatrix}$$

$$0 \le x, y < 256\,, u = 2^{-53}$$



256x256 pixel image

red=pos., yellow=0, blue=neg.

orientation evaluated with `double`

SGP'04 Tutorial 31

2D-Orientation of Three Points

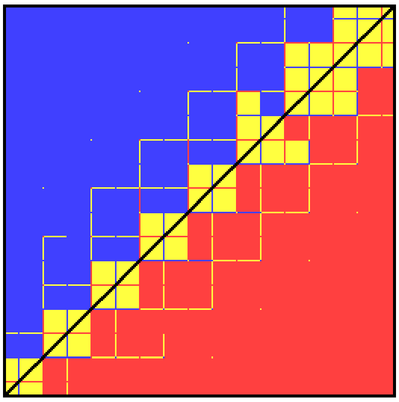
Orientation(p, q, r) = sign((q_x-p_x)(r_y-p_y) – (q_y-p_y)(r_x-p_x))

$$p: \begin{pmatrix} 0.5 + x \cdot u \\ 0.5 + y \cdot u \end{pmatrix}$$

$$q: \begin{pmatrix} 8.8000000000000007 \\ 8.8000000000000007 \end{pmatrix}$$

$$r: \begin{pmatrix} 12.1 \\ 12.1 \end{pmatrix}$$

$$0 \le x, y < 256\,, u = 2^{-53}$$



256x256 pixel image

red=pos., yellow=0, blue=neg.

orientation evaluated with `double`

SGP'04 Tutorial 32

2D-Orientation of Three Points

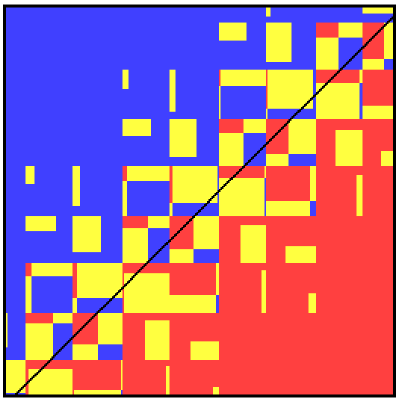
Orientation(p, q, r) = sign((q_x-p_x)(r_y-p_y) – (q_y-p_y)(r_x-p_x))

$$p: \begin{pmatrix} 0.5 + x \cdot u \\ 0.5 + y \cdot u \end{pmatrix}$$

$$q: \begin{pmatrix} 8.8000000000000007 \\ 8.8000000000000007 \end{pmatrix}$$

$$r: \begin{pmatrix} 12.1 \\ 12.1 \end{pmatrix}$$

$$0 \le x, y < 256, u = 2^{-53}$$

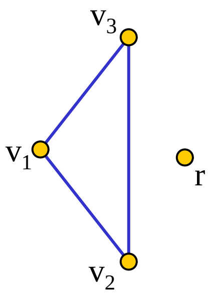


256x256 pixel image

red=pos., yellow=0, blue=neg.

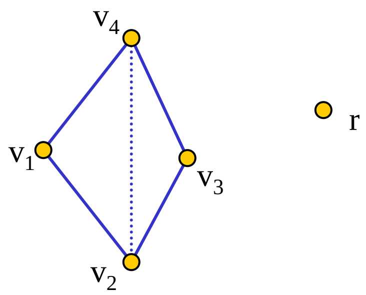
orientation evaluated with `ext double`

Convex Hulls in the Plane



- maintain current hull as a circular list L=(v_0,v_1,…,v_{k-1}) of its extreme points in counter-clockwise order
- start with three non-collinear points in S.
- consider the remaining points r one by one.

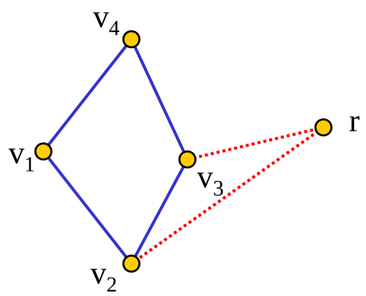
Convex Hulls in the Plane



- maintain current hull as a circular list $L=(v_0,v_1,\ldots,v_{k-1})$ of its extreme points in counter-clockwise order
- start with three non-collinear points in *S*.
- consider the remaining points *r* one by one.

SGP'04 Tutorial 35

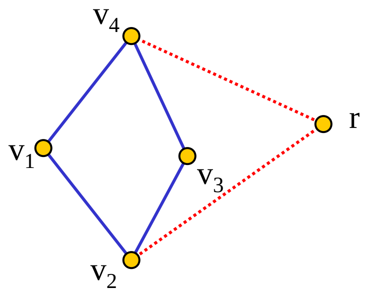
Convex Hulls in the Plane



- maintain current hull as a circular list $L=(v_0,v_1,\ldots,v_{k-1})$ of its extreme points in counter-clockwise order
- start with three non-collinear points in *S*.
- consider the remaining points *r* one by one.

SGP'04 Tutorial 36

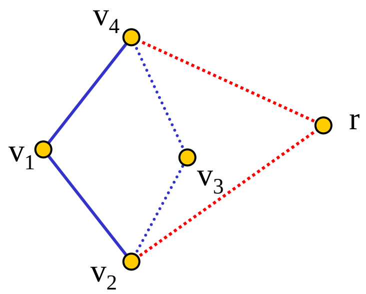
Convex Hulls in the Plane



- maintain current hull as a circular list $L=(v_0,v_1,\ldots,v_{k-1})$ of its extreme points in counter-clockwise order
- start with three non-collinear points in *S*.
- consider the remaining points *r* one by one.

SGP'04 Tutorial 37

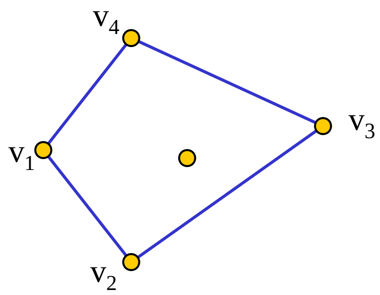
Convex Hulls in the Plane



- maintain current hull as a circular list $L=(v_0,v_1,\ldots,v_{k-1})$ of its extreme points in counter-clockwise order
- start with three non-collinear points in *S*.
- consider the remaining points *r* one by one.

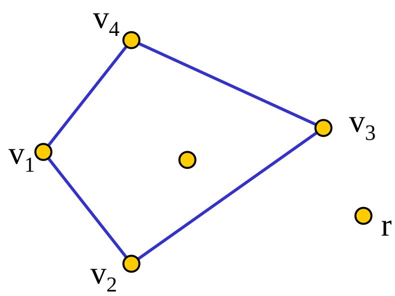
SGP'04 Tutorial 38

Convex Hulls in the Plane



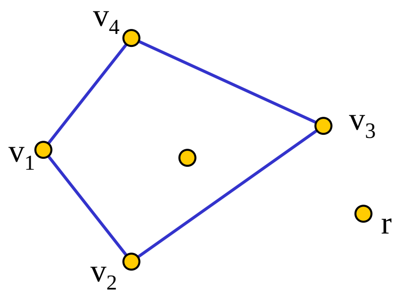
- maintain current hull as a circular list $L=(v_0,v_1,\ldots,v_{k-1})$ of its extreme points in counter-clockwise order
- start with three non-collinear points in S.
- consider the remaining points r one by one.

Convex Hulls in the Plane



- **[Property A]** A point r is outside CH iff there is an i such that the edge (v_i, v_{i+1}) is visible for r. (orientation$(v_i,v_{i+1},r) > 0$)

Convex Hulls in the Plane



- **[Property A]** A point r is outside CH iff there is an i such that the edge (v_i, v_{i+1}) is visible for r. (orientation$(v_i,v_{i+1},r) > 0$)
- **[Property B]** If r is outside CH, then the set of edges that are weakly visible (= orientation is non-negative) from r forms a non-empty consecutive subchain; so does the set of edges that are not weakly visible from r.

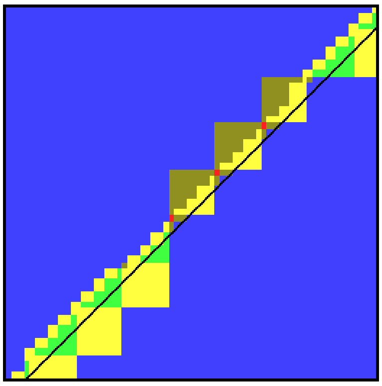
SGP'04 Tutorial 41

Single Step Failures

- Systematic construction of instances leading to violations of properties **(A)** and **(B)** when executed with `double`'s
- and in all possible ways
  - a point outside sees no edge
  - a point inside sees an edge
  - a point outside sees all edges
  - a point outside sees a non-contiguous set of edges
- examples involve nearly collinear points, of course
- examples are realistic as many real-life instances contain collinear points (which become nearly collinear by conversion to `double`'s)

SGP'04 Tutorial 42

Systematic Search

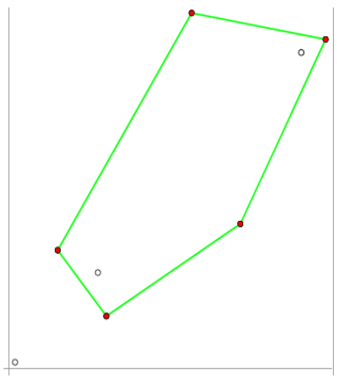


- a point outside sees no edge
  - p_1 = (0.5, 0.5)
  - p_2 = (7.3000000000000194, 7.3000000000000167)
  - p_3 = (24.000000000000068, 24.000000000000071)
  - p_4 = (24.00000000000005, 24.000000000000053)
- (p_2, p_3, p_4) form a counter-clockwise triangle

- Classification of $(x(p_1) + x{\cdot}u, y(p_1) + y{\cdot}u)$ with respect to the edges (p_2, p_3) and (p_4, p_2).

red = sees no edge, ocher = collinear with one, yellow = collinear with both, blue = sees one but not the other, green = sees both

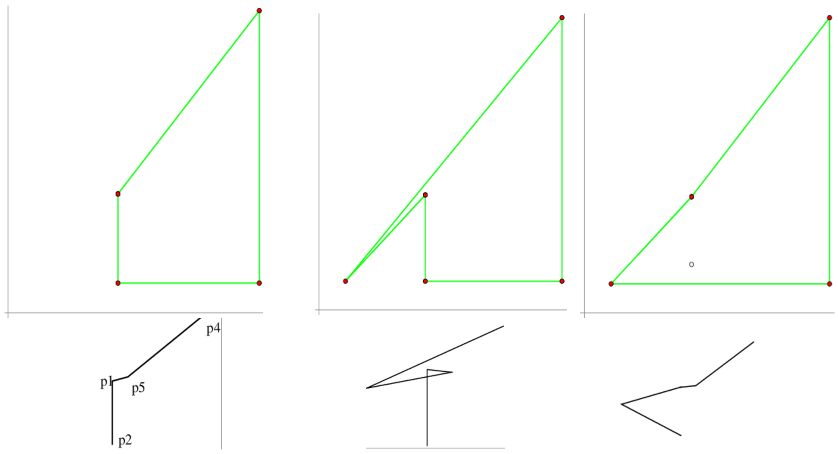
Global Consequences I



[this run did not terminate!]

- a point outside sees no edge of the current hull
- a point outside sees all edges of the current hull

Global Consequences II



- a point inside sees an edge of the hull
- a point outside sees a non-contiguous set of edges

Antibes

- Arrangements of circular arcs

Antibes

- Arrangements of circular arcs, with double arithmetic

Antibes

- Arrangements of circular arcs, with float arithmetic

What can be done?

- Redesign algorithms for FP arithmetic
  - works for some problems, no general theory
- Exact arithmetic
  - long integers, rationals, *k*-th roots, algebraic numbers
- FP filter for efficiency (interval arithmetic)
  - error bounds: static, semi-static, dynamic

- Type of arithmetic and filter depends on the application
  ➔Flexibility

Geometric Kernels

- Primitives, Predicates, Constructions
  - `Kernel::Point_2`
  - `Kernel::Left_turn_2`

- Convenient typedefs for common kernel
  - `Exact_predicates_inexact_constructions_kernel`
  - `Exact_predicates_exact_constructions_kernel`
  - `Exact_predicates_exact_constructions_kernel_with_sqrt`

- In principle choice of:
  - Cartesian and homogeneous representation
  - Reference counting and no reference counting
  - Floating-point filters as kernel adaptors or number types
  - **`CGAL::Simple_cartesian<double> // non-robust!!!`**

Contents

- History and Overview of CGAL
- Robustness and the Exact-Geometric-Computing Paradigm
- Tutorial on Polyhedral Surfaces (Meshes)
- Demo of the Viewer

Contents

- History and Overview of CGAL
- Robustness and the Exact-Geometric-Computing Paradigm
- Tutorial on Polyhedral Surfaces (Meshes)
  - Halfedge data-structure and the default polyhedron
  - Customizing the polyhedron
  - $\sqrt{3}$ -subdivision using Euler-operators
  - Qt-subdivision using incremental builder
  - Combinatorial subdivision library (CSL) using policy-based design
  - Other useful geometric algorithms for polyhedra
- Demo of the Viewer

Why Use a Library Solution?

- Easy to get started with a first implementation
- Typical students homework assignment
- "I have already one and its good enough"

- Many pointers and pointer types
- Easy to get started, but not so easy to make it compact and fast ("It's my 4th implementation")
- Hard to debug
- Hard to maintain and adapt over time
- Many goodies can be re-used
  - File IO
  - Rendering
  - Self-intersection test

SGP'04 Tutorial 53

Why Use the CGAL Solution?

- Is it fast enough?
  Yes; compile-time polymorphism, templates

- Is it small enough?
  Yes; can be tailored.

- Is it flexible enough?
  Yes; within the modeling space

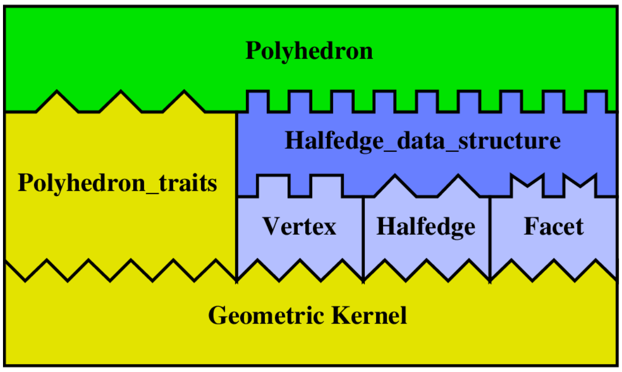
- Is it easy enough to use?
  Yes; see this tutorial

- What is the license, can I use it?
  Yes, we hope so.

SGP'04 Tutorial 54

Polyhedral Surfaces

Building blocks assembled with C++ templates

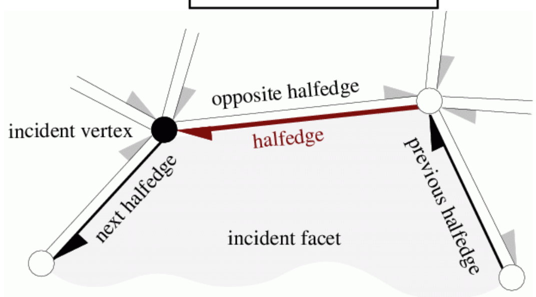


Default Polyhedron

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

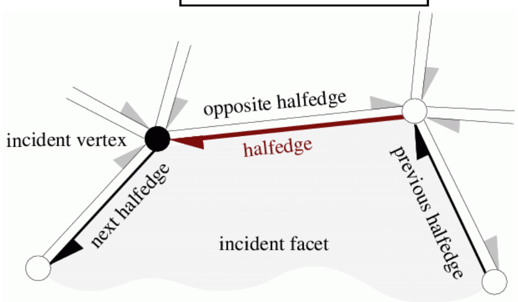
```
typedef CGAL::Simple_cartesian<double> Kernel;
typedef Kernel::Point_3                Point_3;
typedef CGAL::Polyhedron_3<Kernel>     Polyhedron;
typedef Polyhedron::Vertex_iterator    Vertex_iterator;

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;
}
```


Flexible Halfedge 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()
……	…



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;
};
```

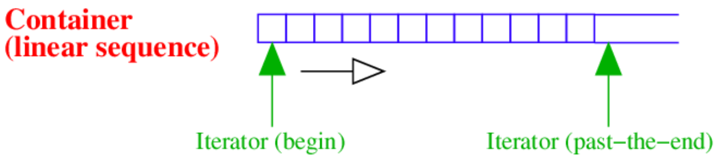

Flexible Polyhedral Surfaces

```
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;
```

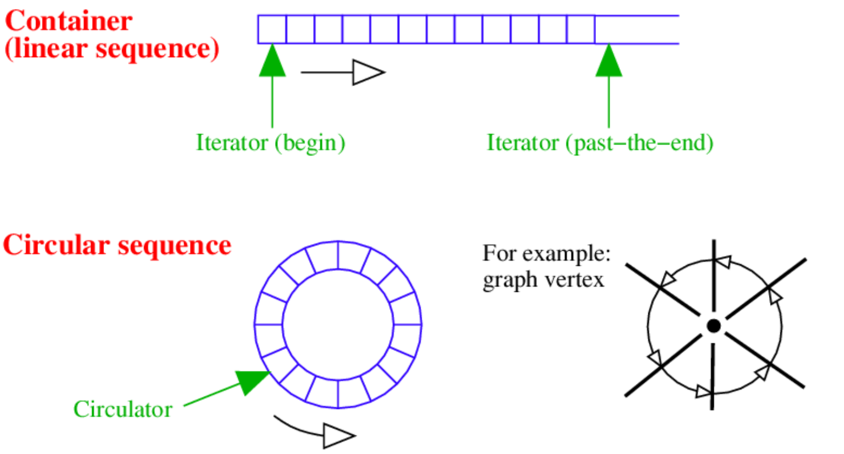
Default Polyhedron

```
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;
}
```


Iterators



Iterators and Circulators



SGP'04 Tutorial 65

Render all Facets

```
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();
    CGAL_assertion( CGAL::circulator_size(j) >= 3);
    glBegin( GL_POLYGON);
    do {
        glVertex3dv( &(j->vertex()->point().x()));
    } while ( ++j != i->facet_begin());
    glEnd();
}
```

SGP'04 Tutorial 66

Polyhedral Surface

Polyhedron

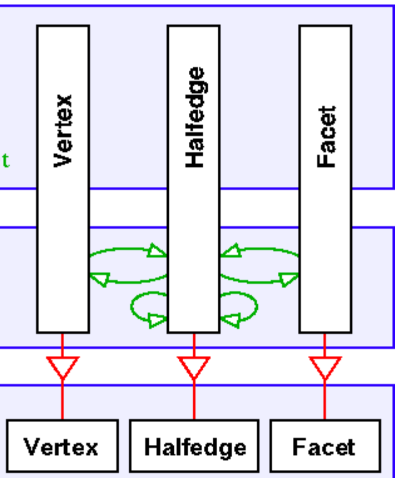
- provides ease- of- use
- protects combinatorial integrity
- defines circulators
- defines extended vertex, halfedge, facet

Halfedge_data_structure

- manages storage (container class)
- defines iterators

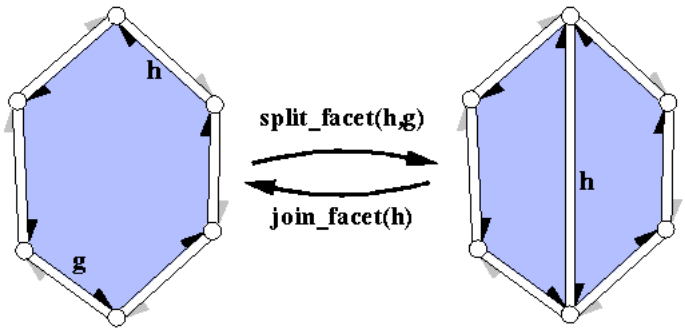
Items

- stores actual information
- contains user added data and functions

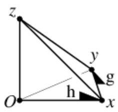


Euler Operators

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



Create a Cube with Euler Operator

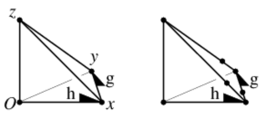


(a)

```
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)
```

SGP'04 Tutorial 69

Create a Cube with Euler Operator

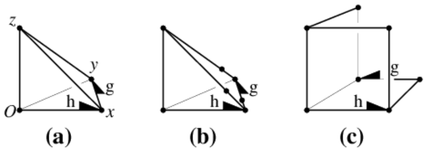


(a) (b)

```
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)
```

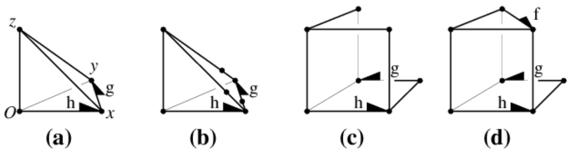
SGP'04 Tutorial 70

Create a Cube with Euler Operator



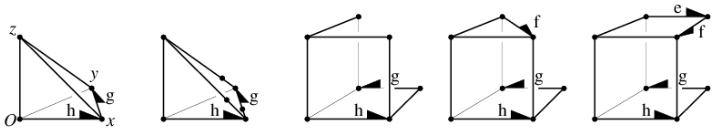

```
h->next()->vertex()->point()     = Point( 1, 0, 1);
g->next()->vertex()->point()     = Point( 0, 1, 1);
g->opposite()->vertex()->point() = Point( 1, 1, 0); (c)
```


Create a Cube with Euler Operator




```
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)
```

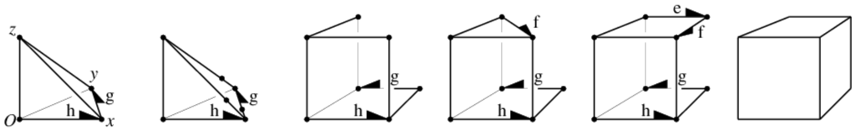
Create a Cube with Euler Operator



(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)
```

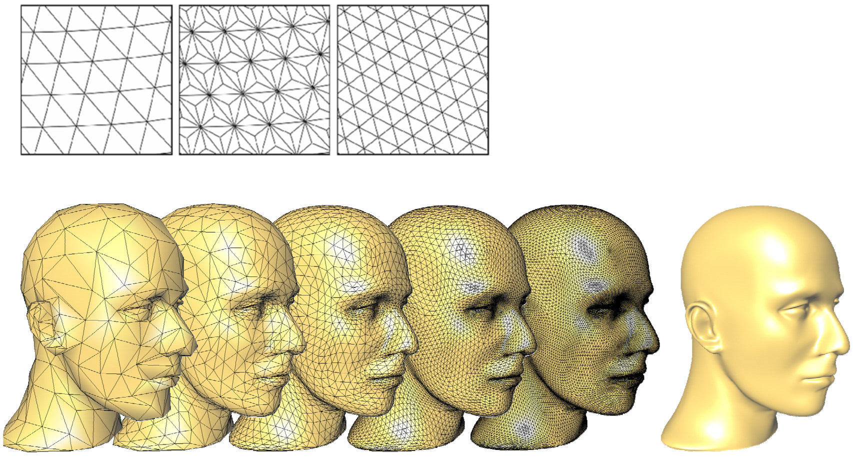

Create a Cube with Euler Operator



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

```
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)
```

√3 -Subdivision [Kobbelt'00]



√3 -Subdivision [Kobbelt'00]

- Barycentric triangulation of a facet exists as Euler-operator
- We need to compute the new point coordinates

```
void create_centroid( Polyhedron& P, Facet_iterator f){
  Halfedge_handle h = f->halfedge();
  Vector vec = h->vertex()->point() - ORIGIN;
  vec = vec + (h->next()->vertex()->point() - ORIGIN);
  vec = vec + (h->next()->next()->vertex()->point()
                                          - ORIGIN);
  Halfedge_handle new_ctr = P.create_center_vertex(h);
  new_ctr->vertex()->point() = ORIGIN + (vec/3.0);
}
```

√3 -Subdivision [Kobbelt'00]

- Edge-flip exists also as Euler-operator
- We need to compute the Smoothing rule for original points

```
struct Smooth_old_vertex {
Point operator()( const Vertex& v) const {
  std::size_t degree = v.vertex_degree()/2;
  double alpha = (4.0-2.0*cos(2.0*CGAL_PI/degree))/9.0;
  Vector vec = (v.point() - ORIGIN) * (1.0-alpha);
  Halfedge_around_vertex_const_circulator h =
                                   v.vertex_begin();
  do {
    vec = vec + (h->opposite()->vertex()->point()
                 - ORIGIN) * alpha/degree;
    ++ h; ++ h;
  } while ( h != v.vertex_begin());
  return (ORIGIN + vec);
}};
```


√3 -Subdivision [Kobbelt'00]

```
void subdiv( Polyhedron& P) {
  std::size_t nv = P.size_of_vertices();
  Vertex_iterator last_v = P.vertices_end();
  --last_v;                          // the last of the old vertices
  Edge_iterator last_e = P.edges_end();
  --last_e;                          // the last of the old edges
  Facet_iterator last_f = P.facets_end();
  --last_f;                          // the last of the old facets
  Facet_iterator f = P.facets_begin();  // centroids
  do {
      create_centroid( P, f);
  } while ( f++ != last_f);
  std::vector<Point> pts;            // smooth old vertices
  pts.reserve( nv);                  // space for the new points
  ++ last_v;                         // move to past-the-end again
  std::transform( P.vertices_begin(), last_v,
                  std::back_inserter( pts), Smooth_old_vertex());
  std::copy( pts.begin(), pts.end(), P.points_begin());
  ++ last_e;                         // move to past-the-end again
  for ( Edge_iterator e = P.edges_begin(); e != last_e; ++e)
      P.flip_edge(e);                // flip the old edges
}
```

$\sqrt{3}$ -Subdivision [Kobbelt'00]

```
#include <CGAL/Simple_cartesian.h>
#include <CGAL/HalfedgeDS_vector.h>
#include <CGAL/Polyhedron_3.h>
#include <CGAL/IO/Polyhedron_iostream.h>
#include <iostream>
#include <algorithm>
#include <vector>

using std::cerr; using std::endl; using std::cout; using std::cin;
using std::exit;

class Polyhedron_min_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, CGAL::Tag_true>
                                                                  Halfedge;
    };
    template < class Refs, class Traits>
    struct Face_wrapper {
        typedef CGAL::HalfedgeDS_face_base< Refs, CGAL::Tag_true>    Face;
    };
};

typedef CGAL::Simple_cartesian<double>                         Kernel;
typedef Kernel::Vector_3                                       Vector;
typedef Kernel::Point_3                                        Point;
typedef CGAL::Polyhedron_3<Kernel, Polyhedron_min_items_3,
                           CGAL::HalfedgeDS_vector>            Polyhedron;

typedef Polyhedron::Vertex                                     Vertex;
typedef Polyhedron::Vertex_iterator                         Vertex_iterator;
typedef Polyhedron::Halfedge_handle                         Halfedge_handle;
typedef Polyhedron::Edge_iterator                             Edge_iterator;
typedef Polyhedron::Facet_iterator                           Facet_iterator;
typedef Polyhedron::Halfedge_around_vertex_const_circulator HV_circulator;
typedef Polyhedron::Halfedge_around_facet_circulator          HF_circulator;

void create_centroid( Polyhedron& P, Facet_iterator f) {
    Halfedge_handle h = f->halfedge();
    Vector vec = h->vertex()->point() - CGAL::ORIGIN;
    vec = vec + (h->next()->vertex()->point() - CGAL::ORIGIN);
    vec = vec + (h->next()->next()->vertex()->point() - CGAL::ORIGIN);
    Halfedge_handle new_center = P.create_center_vertex( h);
    new_center->vertex()->point() = CGAL::ORIGIN + (vec / 3.0);
}
```

```
struct Smooth_old_vertex {
    Point operator()( const Vertex& v) const {
        std::size_t degree = CGAL::circulator_size( v.vertex_begin()) / 2
        double alpha = ( 4.0 - 2.0 * cos( 2.0 * CGAL_PI / degree)) / 9.0;
        Vector vec = (v.point() - CGAL::ORIGIN) * ( 1.0 - alpha);
        HV_circulator h = v.vertex_begin();
        do {
            vec = vec + ( h->opposite()->vertex()->point() - CGAL::ORIGIN
                       * alpha / degree;
            ++ h; ++ h;
        } while ( h != v.vertex_begin());
        return (CGAL::ORIGIN + vec);
    }
};
void subdiv( Polyhedron& P) {
    if ( P.size_of_facets() == 0)
        return;
    // We use that new vertices/halfedges/facets are appended at the end.
    std::size_t nv = P.size_of_vertices();
    Vertex_iterator last_v = P.vertices_end();
    -- last_v;  // the last of the old vertices
    Edge_iterator last_e = P.edges_end();
    -- last_e;  // the last of the old edges
    Facet_iterator last_f = P.facets_end();
    -- last_f;  // the last of the old facets

    Facet_iterator f = P.facets_begin();    // create new center vertices
    do {
        create_centroid( P, f);
    } while ( f++ != last_f);

    std::vector<Point> pts;                    // smooth the old vertices
    pts.reserve( nv);  // get intermediate space for the new points
    ++ last_v; // make it the past-the-end position again
    std::transform( P.vertices_begin(), last_v, std::back_inserter( pts),
                    Smooth_old_vertex());
    std::copy( pts.begin(), pts.end(), P.points_begin());

    ++ last_e; // make it the past-the-end position again
    for ( Edge_iterator e = P.edges_begin(); e != last_e; ++e)
        P.flip_edge(e);    // flip the old edges
}
int main() {
    Polyhedron P;
    cin >> P;
    P.reserve( P.size_of_vertices() + P.size_of_facets(),
               P.size_of_halfedges() + 6 * P.size_of_facets(),
               3 * P.size_of_facets());
    subdiv( P);
    cout << P;
    return 0;
}
```

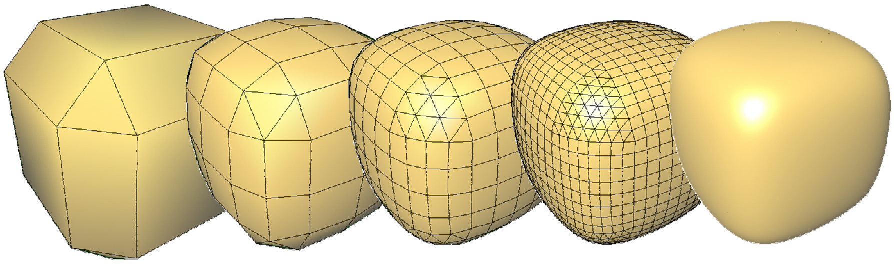

$\sqrt{3}$ -Subdivision [Kobbelt'00]

- Comparison with OpenMesh 1.0.0-beta4
- Lion vase: 400k triangles
- 2 subdivision steps

	CGAL		OPENMESH
$\sqrt{3}$-subdivision	float	double	float
Lion vase: step 1	0.87	1.33	1.33
Lion vase: step 2	3.03	4.68	4.83

Quad-Triangle Subdivision

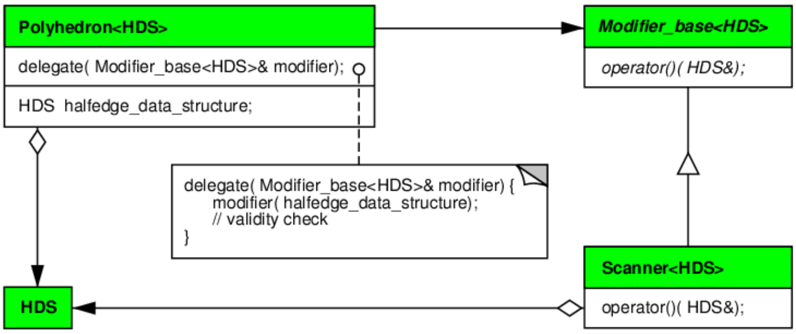
- [Levin'03] [Stam&Loop'03]
- Basically Loop subdivision on triangles and Catmull-Clark subdivision on polygons of the control mesh
- After one iteration the model has only triangles and quads



SGP'04 Tutorial 81

Quad-Triangle Subdivision

- Some schemes are easier to realize with the incremental builder that provides an indexed-facet-set type of construction
- Uses the modifier design to access the internal HDS



SGP'04 Tutorial 82

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;
}
```

CSL: Combinatorial Subdivision Library

Policy-based design [Alexandrescu’01]

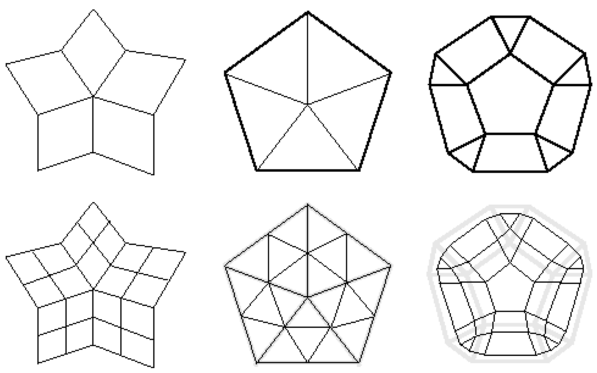
- **Host:** generic function – refinement, stencil correspondence
- **Policy:** template parameter – geometric smoothing

Example:

```
template <class Polyhedron>
void catmull_clark( Polyhedron& P, int step = 1) {
    quad_quadralize_polyhedron(
        P, CatmullClark_rule<Polyhedron>(), step);
}
```


CSL: Combinatorial Subdivision Library

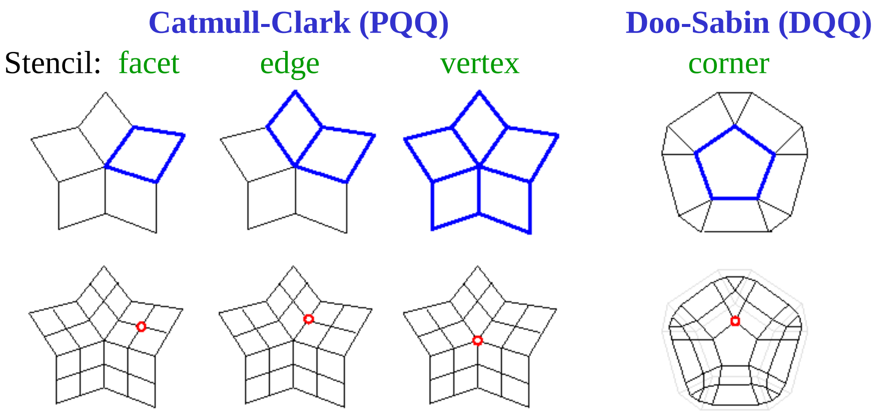
- Available refinement schemes (host functions):



PQQ **PTQ** **DQQ**

CSL: Combinatorial Subdivision Library

- Stencil correspondence (determined by host function):



SGP'04 Tutorial 87

Geometric Algorithms

Self Intersection Test

- Based on fast box intersections [Zomorodian&Edelsbrunner'02]
- Needs exact predicates

Smallest Enclosing Sphere (of Spheres)

- Linear time algorithm (randomized) [Fischer&Gärtner'03]
- Needs exact constructions, but robust with `double`'s

Convex Hull and Width

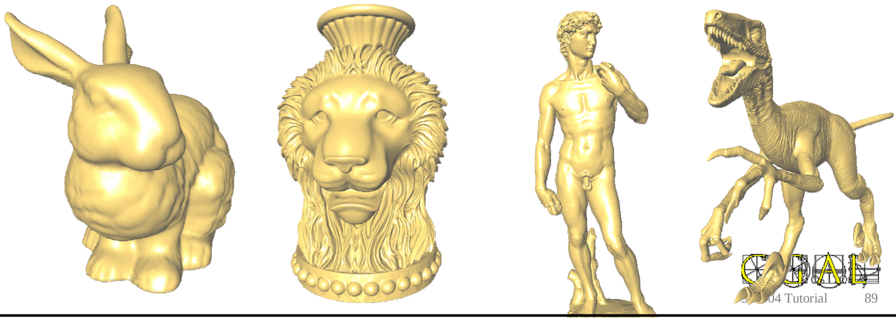
- Quickhull [Barber et al.'96], width can be quadratic
- Convex hull needs exact predicates, width needs exact constr.

SGP'04 Tutorial 88

Geometric Algorithms

time in seconds	min. sphere double	min. sphere gmpq	convex hull	min. width	self inter-section
Bunny	0.02	14	3.5	111	2.3
Lion vase	0.19	396	13.1	276	15.5
David	0.12	215	20.3	112	31.5
Raptor	0.35	589	45.5	123	78.2

#F 70k 400k 700k 2.000k



Demo of the Viewer

Acknowledgements

The Lion-vase and dinosaur models are courtesy of SensAble Technologies, Inc. The bunny and the David model are courtesy of the Stanford graphics group.

http://www.cgal.org/Tutorials/

Contents

- History and Overview of CGAL
- Robustness and the Exact-Geometric-Computing Paradigm
- Tutorial on Polyhedral Surfaces (Meshes)
- Demo of the Viewer