Polygon Mesh Processing in CGAL
GeometryFactory

- 7 engineers (6 PhD, produced by INRIA)
- Actively involved in the CGAL Project (reviews, release management, ...)
- License contracts with the academic partners

- Sales of CGAL software components
- Support to increase customer productivity
- Improve components for customers
- Development of new components for customers
Generic Programming
STL Genericity

template <class Key, class Less>
class set {
    Less less;

    insert(Key k) {
        if (less(k, treenode.key))
            insertLeft(k);
        else
            insertRight(k);
    }
};
CGAL Genericity

template < class Geometry >
class Delaunay_triangulation_2 {
    Geometry::Orientation orientation;
    Geometry::In_circle in_circle;

    void insert(Geometry::Point t) {
        ...
        if(in_circle(p,q,r,t)) {...}  
        ...
        if(orientation(p,q,r){...}    
    }   
};
CGAL Genericity

Without explicit conversion to points in the plane
• Triangulate the terrain in an xy-plane
• Triangulate the faces of a Polyhedron

Courtesy: IPF, Vienna University of Technology & Inpho GmbH
CGAL and the Boost Graph Library
Boost Graph Library (BGL)

- Rich collection of graph algorithms: shortest paths, minimum spanning tree, flow, etc.
- A set of concepts to define an abstraction layer for manipulating a graph:
  - algorithms are independant from the data structure

```cpp
boost::dijkstra_shortest_paths(g, source ,
   distance_map(distance_pmap)
   .predecessor_map(predecessor_pmap));
```
Boost Graph Library (BGL)

template <typename Graph>
struct graph_traits {
    typedef ... vertex_descriptor;
    typedef ... edge_descriptor;
    typedef ... vertex_iterator;
    ...
};

vertex_descriptor v, w;
edge_descriptor e;

v = source(e, G);
w = target(e, G);

BOOST_FOREACH(vertex_descriptor v, vertices(G))
{
    ...
}
Boost Graph Library (BGL)

BGL and CGAL

- Glue layer for triangulations, arrangements, HDS, Surface_mesh, Polyhedron_3, OpenMesh, ...
  → We can call dijkstra directly on a CGAL triangulation
- Extension to halfedge and face graphs to implement generic algorithm on polyhedral surfaces
- Most CGAL algorithms are using this API (all in a near future)
- Useful also for having wrappers (Face_filtered_graph, ...)
Boost Graph Library (BGL)

template <typename FaceGraph >
struct boost::graph_traits {
  typedef ... vertex_descriptor;
  typedef ... edge_descriptor;
  typedef ... halfedge_descriptor;
  typedef ... face_descriptor;
};

h_opp = opposite(h,G);
h_next = next(h,G);

h = halfedge(e,G);
e = edge(h,G);

f = face(h,G);
h = halfedge(f,G);
Property Maps:
A generic way to associate properties to an object

template <class PMap>
struct property_traits {
    typename Pmap::key_type key_type;
    typename Pmap::value_type value_type;
    typename Pmap::reference reference;
    typename Pmap::category category;
};

Concepts:

- **ReadablePropertyMap**

  reference get(key_type key, PMap pmap);

- **WritablePropertyMap**

  void put(key_type key, PMap pmap, value_type value);

- **ReadWritePropertyMap** refines **ReadablePropertyMap** and **WritablePropertyMap**

- **Non-mutable LvaluePropertyMap** refines **ReadablePropertyMap**

  const value_type& v = pmap[k]

- **Mutable LvaluePropertyMap** refines **ReadablePropertyMap**

  value_type& v = pmap[k]
Property Maps

// defining a property map with Point_3 as value type and unsigned int as key
std::vector<Point_3> points;
CGAL::Pointer_property_map<Point_3> pmap(points);

// turning a std::map into a property map
std::map<vertex_descriptor, bool> std_map;
boost::associative_property_map< std::map<vertex_descriptor, bool> > pmap(std_map);

// User defined property map creating a CGAL point on the fly
struct MyPoint{ double x,y,z; };
struct My_pmap
{
    typedef MyPoint key_type;
    typedef boost::readable_property_map_tag category;
    typedef K::Point_3 value_type;
    Typedef value_type reference;

    friend reference get(const key_type& k, My_pmap) { return K::Point_3(k.x, k.y, k.z); }
};
Property Tags in BGL API

Tags enable generic code to define and get a property map.

```cpp
typedef boost::property_map<PropertyGraph,
                            boost::vertex_index_t>::type PMap;

boost::property_traits<PMap>::key_type k;
boost::property_traits<PMap>::value_type v;

PropertyGraph graph;
PMap pm = get(boost::vertex_index,graph);

vi = get(pm,v);
put(pm, v, vi);
```
Dynamic Properties using Surface_mesh

typedef CGAL::Surface_mesh<K::Point_3> Mesh;

typedef Mesh::Face_index Face_index;

Mesh m;

Mesh::Property_map<Face_index, std::size_t> cc_ids =
    m.add_property_map<Face_index, std::size_t>("f:cc_ids").first;

// extract the connected component id of each face
PMP::connected_components(m, cc_ids);

for(Face_index f : faces(m))
    std::cout << f << " " << cc_ids[f] << "\n";
Named Parameters

Optional parameters for polygon mesh processing algorithms are provided as named parameters.

```cpp
template<typename PolygonMesh, typename FaceRange , typename NamedParameters >
void CGAL::Polygon_mesh_processing::isotropic_remeshing ( const FaceRange & faces,
                                     const double & target_edge_length,
                                     PolygonMesh & pmesh,
                                     const NamedParameters & np)
```

remeshes a triangulated region of a polygon mesh.

This operation sequentially performs edge splits, edge collapses, edge flips, tangential relaxation and projection to the initial surface to generate a smooth mesh with a prescribed edge length.

**Template Parameters**
- **PolygonMesh** model of `MutableFaceGraph` that has an internal property map for `CGAL::vertex_point_t`. The descriptor types `boost::graph_traits<PolygonMesh>::face_descriptor` and `boost::graph_traits<PolygonMesh>::halfedge_descriptor` must be models of `Hashable`. If `PolygonMesh` has an internal property map for `CGAL::face_index_t`, then it should be initialized.
- **FaceRange** range of `boost::graph_traits<PolygonMesh>::face_descriptor`.
- **NamedParameters** a sequence of Named Parameters

**Parameters**
- **pmesh** a polygon mesh with triangulated surface patches to be remeshed
- **faces** the range of triangular faces defining one or several surface patches to be remeshed
- **target_edge_length** the edge length that is targetted in the remeshed patch
- **np** optional sequence of Named Parameters among the ones listed below
PMP::isotropic_remeshing(faces(mesh),
  target_edge_length,
  mesh,
  PMP::parameters::number_of_iterations(10)
    .protect_constraints(true)
    .edge_is_constrained_map(constrained_edges_map)
    .vertex_is_constrained_map(constrained_vertices_map)
    .number_of_relaxation_iterations(5)
);
Examples
Surface Mesh Simplification
Surface Mesh Simplification

• Iterative edge collapse preserving the topology of the input model
  – Edges are sorted using a cost function
  – Each edge collapsed becomes a point which position is computed using a placement function.
  – The algorithm stops when no further collapse is possible without changing the topology or when a stop function indicates to.
Surface Mesh Simplification

- The cost, placement, and stop functions can be provided to the main function using named parameters:
  - Several classes are provided in CGAL.
  - User can provide his own custom classes by following the `GetCost`, `GetPlacement`, and `StopPredicate` concepts.
- User can also mark edges as non-collapsible.
- Placement and cost functions can prevent the collapse of an edge (for example when the placement cannot be computed or is invalid).
Surface Mesh Simplification

namespace SMS = CGAL::Surface_mesh_simplification;
namespace params = CGAL::parameters;

SMS::edgeCollapse(surface_mesh, stop,
    params::edge_is_constrained_map(bem)
    .get_placement(Placement(bem))
    .get_cost(Cost())
);

Approximated Hausdorff Distance
(Polygon Mesh Processing – CGAL 4.10)
Approximated Hausdorff Distance
(Polygon Mesh Processing – CGAL 4.10)

- Approximation (lower bound) of the distance from surface A to surface B
  - Points are taken on the surface of A
  - An AABB-tree of the faces of B is used to get the distance of each sample point to B
- A template parameter allows to do the queries in parallel
- The sampling method and the quality of the sampling is controlled using named-parameters
Quick Tour
Distances and Intersections

Based on CGAL::AABB_tree
Distance and Intersection Computation
Box Intersection Detection

- CGAL::Box_intersection_d
  Algorithm for finding all intersecting pairs for large numbers of axis-aligned bounding boxes.
Intersection Test
Hole filling  [Liepa 2003], [Zou et al. 2013]
Isotropic Remeshing [Botsch-Kobbelt 2004]

Feature Preserving
Isotropic Remeshing

Apply on Selection
3D Boolean Operations using Corefinement

• Fast version but restricted to surface meshes
• Various applications: attribute preservation, mesh clipping, ...
Subdivision and Simplification
Parameterization
Skeletonization [Tagliasacchi et al.]

*Mean Curvature Flow* skeletonization
Segmentation [Shapira et al. 2008]

• Segment surface into $k$ patches
• Based on «shape diameter» estimate
Deformation [Sorkine-Alexa 2007]

As Rigid as Possible ("ARAP")
Geodesic Shortest Path
Mesh Repair
Mesh Repair
Mesh Repair

gap

overlapping patches
Mesh Fusion
Self-intersections Removal using Refinement
Self-intersections Removal using Refinement
Self-intersections Removal using Refinement
Self-intersections Removal using Refinement
Polyhedron Demo

Polyhedron *bloppy* (mode: flat+edges, color: #6464ff)
Number of vertices: 2027