A Tutorial on CGAL Polyhedron for Subdivision Algorithms*

Le-Jeng Shiue† Pierre Alliez‡ Radu Ursu§ Lutz Kettner¶

Abstract

We give an overview of the tutorial for the `CGAL::Polyhedron_3` and its use in subdivision algorithms. The full tutorial and the accompanying source code are available at `http://www.cgal.org/Tutorials/Polyhedron/`.

Introduction

Polyhedron data structures based on the concept of halfedges have been very successful for the design of general algorithms on meshes. Common practice is to develop such data structure from scratch, since clearly a first implementation is at the level of a students homework assignment. But then, these data structures consist almost entirely of pointers for all sort of incidence informations. Maintaining them consistently during mesh operations is not anymore a trivial linked-list update operation. So, moving from a students exercise to a reliable research implementation, including maintaining and optimizing it, is a respectable software task.

What is common practice for simple data structures, such as linked lists, should be common practice even more so for mesh data structures, namely, to use a good, flexible, and efficient library implementation. In C++ the *Standard Template Library*, STL, is an excellent address for our analog example of the linked lists [Aus99], and we argue in the full tutorial that the Polyhedron data structure in CGAL is such a flexible mesh data structure [Ket99], and it comes with a rich and versatile infrastructure for mesh algorithms. CGAL, the *Computational Geometry Algorithms Library*, is a C++ library available from `www.cgal.org` [FGK+00].

We strongly believe that such a tutorial with its wealth of information will give a head start to new researches and implementations of mesh algorithms. We also believe that it will raise the quality of implementations. Firstly, it encourages the use of well tested and over time matured implementations, e.g., `CGAL::Polyhedron_3` in its current design is about five years publicly released and used. Secondly, it documents good implementation choices, e.g., the example programs can be used as starting points for evolutionary software development. Thirdly, it offers easy access to additional functionality, such as the efficient self intersection test, that otherwise could be expandable in a research prototype.

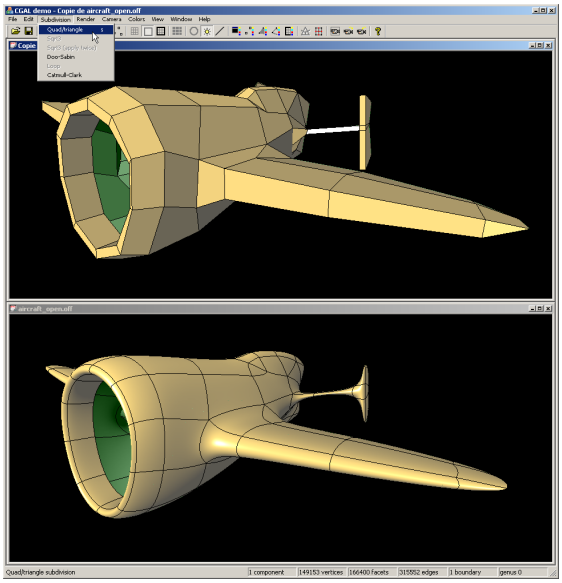

Figure 1 – *The polyhedron viewer running on Windows. A coarse polygon mesh is subdivided using the quad-triangle subdivision scheme.*

The tutorial is organized around subdivision surfaces in a polyhedron viewer. The polyhedron viewer (Figure 1) demonstrates the basic functionalities of the `CGAL::Polyhedron_3` and some extended functionalities such as file I/O, mesh superimposition, and trackball manipulation. Several subdivision surfaces are supported in the polyhedron viewer, including Catmull-Clark, Loop, Doo-Sabin, $\sqrt{3}$ and Quad-Triangle subdivisions. The tutorial shows how to implement subdivision surfaces in two different mechanisms provided by `CGAL::Polyhedron_3`: *Euler operators* and *modifier callback mechanism*. A $\sqrt{3}$ subdivision implementation is designed based on the Euler operators and a Quad-Triangle subdivision implemen-

*Partially supported by the grant NSF 9457806-CCR and IST Programme of the EU as a Shared-cost RTD (FET Open) Project under Contract No IST-2000-26473 (ECG - Effective Computational Geometry for Curves and Surfaces).

†SurfLab, University of Florida
‡GEOMETRICA, INRIA Sophia-Antipolis
§Geometry Factory, Sophia-Antipolis
¶Max-Planck-Institut für Informatik, Saarbrücken, Germany

tation is designed based on overloading the modifier. Extended from the previous design, a *combinatorial subdivision library* (CSL) is then proposed with increased sophistication and abstraction. CSL abstracts the geometry operations from the refinements. Subdivisions in CSL are build from refinement host with a template geometry policy. Several fundamental refinement schemes are provided within CSL. They are instantiated with a geometry policy that can be user defined.

The goal of this tutorial is to show how to use `CGAL::Polyhedron_3` on basic graphics functionalities, such as rendering and interactive trackball manipulation, *and* how to design and implement algorithms around meshes. Since connectivity and geometry operations are the primal implementation components in mesh algorithms, subdivisions are chosen to demonstrate both operations on `CGAL::Polyhedron_3`. Readers intended to design and implement mesh algorithms other than subdivisions will also be benefited from the tutorial.

Intended Audience

The intended audience of the tutorial are researchers, developers or students developing algorithms around polyhedron meshes. Knowledge of the halfedge data structure and subdivisions are prerequisites. Short introductions of these two topics are given in the tutorial. The tutorial assumes familiarity with the C++ template mechanism and the key concepts of generic programming [Aus99].

CGAL Polyhedron

CGAL Polyhedron (`CGAL::Polyhedron_3`) is realized as a container class that manages geometry items such as vertices, halfedges, and facets with their incidences. `CGAL::Polyhedron_3` has chosen the halfedge data structure as the underlying connectivity structure. In the halfedge data structure, a halfedge is associated with a facet and stores the adjacency pointers to it previous, next and opposite halfedge (Figure 2). The details of the halfedge data structure and the `CGAL::Polyhedron_3` based on it are described in [Ket99].

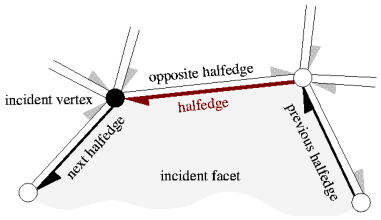

Figure 2 – *One halfedge and its incident primitives. The next halfedge, the opposite halfedge, and the incident vertex are mandatory, the remaining elements are optional.*

What are the potential obstacles in using CGAL and `CGAL::Polyhedron_3`?

1. Is it fast enough? Yes. CGAL, coming from the field of Computational Geometry, might have a reputation of using slow exact arithmetic to be on the safe side, but nonetheless, we know where to apply the right techniques of exact arithmetic to gain robustness and yet not to loose efficiency. In addition, CGAL uses *generic programming* and *compile-time polymorphism* to realize flexibility without affecting optimal runtime.

2. Is it small enough? Yes. `CGAL::Polyhedron_3` can be tailored to store exactly the required incidences and other required data, not more and not less.

3. Is it flexible enough? Yes, certainly within its design space of oriented 2-manifold meshes with boundary that was sufficient for the range of applications illustrated with our example programs.

4. Is it easy enough to use? Yes. The full tutorial with its example programs are exactly the starting point for using `CGAL::Polyhedron_3`. The example programs are short and easy to understand. There is certainly a learning curve for mastering C++ to the level of using templates, but it has to be emphasized that using templates is far easier then developing templated code.

5. What is the license, can I use it? Yes, we hope so. CGAL since release 3.0 and our tutorial programs have open source licenses. Other options are available.

Subdivision Surfaces

A subdivision algorithm recursively applies *refinement* and *geometry smoothing* on the control mesh (Figure 5, 6), and approximates the limit surface of the control mesh. Several refinement schemes in practice are illustrated in Figure 3. The stencils of the geometry smoothing are depending on the refinement schemes, i.e. the reparameterizations. A stencil defines a control submesh that is associated with normalized weights of the nodes. Figure 4 demonstrates the stencils of the PQQ scheme in Catmull-Clark subdivision [CC78] and DQQ scheme in Doo-Sabin subdivision [DS78]. We also demonstrate Loop [Loo94], $\sqrt{3}$ [Kob00] and Quad-Triangle [SL03] subdivisions in this tutorial. For further details about subdivisions, readers should refer to [WW02] and [ZS00].

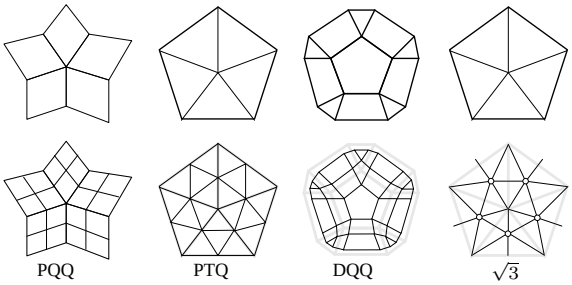

Figure 3 – *Examples of refinement schemes: primal quadrilateral quadrisection (PQQ), primal triangle quadrisection (PTQ), dual quadrilateral quadrisection (DQQ) and $\sqrt{3}$ triangulation. The control meshes are shown in the first row.*

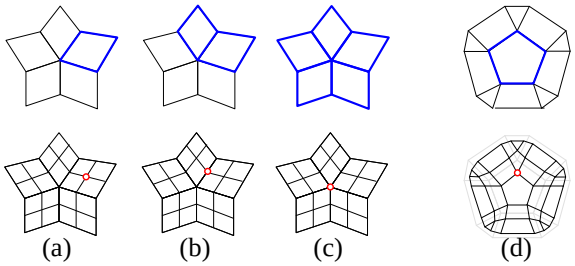

Figure 4 – *The stencil (top blue) and its vertex (bottom red) in Catmull-Clark subdivision (a-c) and Doo-Sabin subdivision (d). Catmull-Clark subdivision has three stencils: facet-stencil (a), edge-stencil (b) and vertex-stencil (c). Doo-Sabin subdivision has only corner-stencil (d). The stencil weights are not shown.*

Tutorial Outlines

Polyhedron Viewer

The tutorial starts with an implementation of a basic polyhedron viewer based on the `CGAL::Polyhedron_3` with the default configuration. This basic viewer demonstrates basic functionalities of a `CGAL::Polyhedron_3`. We describe how to import a polyhedron file in the OFF format based on the *modifier callback mechanism* and the *incremental builder*. We also show the mesh traversal based on the *iterators* and the *circulators* for rendering and the OFF file exporting.

An extended polyhedron viewer is then introduced by customizing the `Polyhedron_3` with extra attributes and functionalities. This enriched polyhedron supports facet and vertex normals for rendering, supports the axis-aligned bounding box of the polyhedron, and provides geometry items specialized with algorithmic flags. The superimposition of the control mesh on the subdivision surfaces are implemented with the flags of the halfedge items (Figure 6).

The tutorial also features a trackball to interactively manipulate the polyhedron, a snapshot function of the camera viewpoint and the transformation states, a raster output to the clipboard, and the vectorial output to a postscript file.

Subdivision Algorithms

The second part of the tutorial focuses on the design and the implementation of $\sqrt{3}$ subdivision (Figure 5) and Quad-Triangle subdivision (Figure 6).

In addition to its importance in the surface modeling, we choose subdivision algorithms to demonstrate both the *connectivity operation* (refinement) and the *geometry operation* (smoothing) of a `CGAL::Polyhedron_3`. These two operations are the primary implementation components required by algorithms on polyhedron meshes. Readers intended to design and implement mesh algorithms other than subdivisions will also be benefited from the techniques we proposed here.

The key to implement a subdivision algorithm is to efficiently support the refinement, i.e. the connectivity modifications. Two approaches are introduced to support the refinement: the *Euler operators* (operator scheme) and the *modifier callback mechanism* (modifier scheme). The operator scheme reconfigures the connectivity with a combination of Euler operators. $\sqrt{3}$ subdivision [Kob00] is used to demonstrate this scheme. We also compare our implementation with the $\sqrt{3}$ subdivision provided in OpenMesh library.

Though simple and efficient in some refinements, e.g. $\sqrt{3}$ subdivision, the correct combination of the operators is hard to find for some refinements, e.g. Doo-Sabin subdivision [DS78]. The modifier scheme solves the problem by letting the programmers create their own combinatorial operators using the polyhedron incremental builder. Quad-Triangle subdivision [SL03, Lev03] is used to demonstrate this scheme.

Combinatorial Subdivision Library

The *C*ombinatorial *S*ubdivision *L*ibrary (CSL) is designed based on the policy-based design [Ale01]. The policy-based design assembles a class (called *host*) with complex behavior out of many small behaviors (called *policies*). Each policy defines an interface for a specific behavior. CSL proposes a generic subdivision solution as a *refinement function* parameterized with the *geometry smoothing rules*. Subdivisions in CSL are build as proper combinations of the refinement functions and the geometry policy classes. The refinement function refines the control mesh, maintains the correspondence between the control mesh and refined mesh, and applies the smoothing stencils provided by the policy class. For example, Catmull-Clark subdivision [CC78] is structured as a quadralization function parameter-

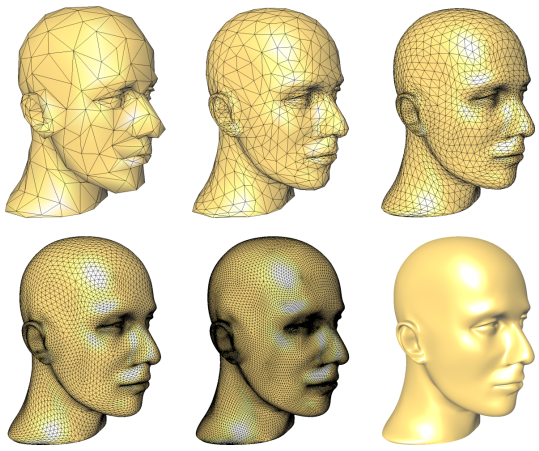

Figure 5 – $\sqrt{3}$ *subdivision of the mannequin mesh.*

ized with the Catmull-Clark smoothing rules.

```
void CatmullClark_subdivision(Polyhedron& p) {
  quadralize_polyhedron
              <CatmullClark_rule<Polyhedron>>(p);
}
class CatmullClark_rule {
public:
  void facet_rule( Facet_handle facet, Point& pt);
  void edge_rule(Halfedge_handle edge, Point& pt);
  void vertex_rule(Vertex_handle vertex, Point& pt);
};
```

The `quadralize_polyhedron<>()` is the host function refining the input mesh and the `CatmullClark_rule` is the policy class applying the Catmull-Clark stencils. The refinement functions are implemented based on the Euler operations or the modifier callback mechanism. The refinement functions also maintain the correspondence with the stencil, i.e., the submesh centered around the given facet, edge, or vertex, and the smoothing point. The smoothing point is calculated by calling the policies, e.g., the `facet_rule()`, the `edge_rule()`, and the `vertex_rule()` respectively. Inside a policy, applying the stencil is simplified to the mesh traversal of a 1-ring neighborhood which can be done with the circulators. Following example illustrates the policy of the facet-stencil in Catmull-Clark subdivision.

```
void facet_rule(Facet_handle facet, Point& point) {
  Halfedge_around_facet_circulator hcir
                          = facet->facet_begin();
  Vector vec = hcir->vertex()->point() - ORIGIN;
  ++hcir;
  do {
    vec = vec + hcir->vertex()->point();
  } while (++hcir != facet->facet_begin());
  point = ORIGIN + vec/circulator_size(hcir);
}
```

This policy-based approach offers a convenient way to specialize a subdivision with the template smoothing rules. CSL currently supports Catmull-Clark, Loop, Doo-Sabin, $\sqrt{3}$ and Quad-Triangle subdivisions. Though demonstrated with a specific enriched `Polyhedron_3` in our polyhedron viewer, CSL accepts any polyhedron mesh specialized from the `Polyhedron_3` with the `Point` type defined in the vertex.

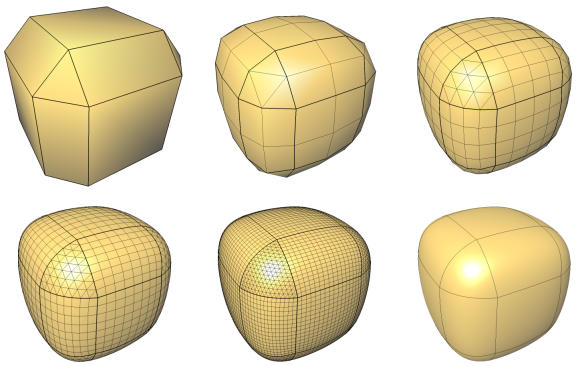

Figure 6 – *Quad-Triangle subdivision of the rhombicuboctahedron mesh.*

References

[Ale01] Andrei Alexandrescu. *Modern C++ Design: Generic Programming and Design Patterns Applied*. Addison-Wesley, 2001.

[Aus99] Matthew H. Austern. *Generic programming and the STL: using and extending the C++ Standard Template Library*. Addison-Wessley, 1999.

[CC78] E. Catmull and J. Clark. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer-Aided Design*, 10:350–355, September 1978.

[DS78] D. Doo and M. Sabin. Behaviour of recursive division surfaces near extraordinary points. *Computer-Aided Design*, 10:356–360, September 1978.

[FGK+00] A. Fabri, G.-J. Giezeman, L. Kettner, S. Schirra, and S. Schönherr. On the Design of CGAL, a Computational Geometry Algorithms Library. *Softw. – Pract. Exp.*, 30(11):1167–1202, 2000.

[Ket99] L. Kettner. Using generic programming for designing a data structure for polyhedral surfaces. *Comput. Geom. Theory Appl.*, 13:65–90, 1999.

[Kob00] L. Kobbelt. $\sqrt{3}$-Subdivision. In *Proceedings of SIGGRAPH*, pages 103–112, 2000.

[Lev03] A. Levin. Polynomial generation and quasi-interpolation in stationary non-uniform subdivision. *Compu. Aided Geom. Des.*, 20(1):41–60, 2003.

[Loo94] C. Loop. Smooth spline surfaces over irregular meshes. In *Proceedings of SIGGRAPH*, pages 303–310, 1994.

[SL03] J. Stam and C. Loop. Quad/triangle subdivision. *Computer Graphics Forum*, 22(1):79–85, March 2003.

[WW02] J. Warren and H. Weimer. *Subdivision Methods for Geometric Design*. Morgan Kaufmann, 2002.

[ZS00] D. Zorin and P. Schröder, editors. *Subdivision for Modeling and Animation*, Course Notes. ACM SIGGRAPH, 2000.