OpenFlipper: An Open Source Geometry Processing and Rendering Framework

Jan Möbius and Leif Kobbelt

Department of Computer Science 8,
RWTH Aachen University,
Ahornstrasse 55, 52074 Aachen,
http://www.graphics.rwth-aachen.de

Abstract. In this paper we present OpenFlipper, an extensible open source geometry processing and rendering framework. OpenFlipper is a free software toolkit and software development platform for geometry processing algorithms. It is mainly developed in the context of various academic research projects. Nevertheless some companies are already using it as a toolkit for commercial applications. This article presents the design goals for OpenFlipper, the central usability considerations and the important steps that were taken to achieve them. We give some examples of commercial applications which illustrate the flexibility of OpenFlipper. Besides software developers, end users also benefit from this common framework since all applications built on top of it share the same basic functionality and interaction metaphors.

Keywords: Geometry Processing, Software Framework, Open Source

1 Introduction

Currently a lot of work is being done to simplify the research and development in the field of geometry processing algorithms. Results of this work are for example the software frameworks Meshlab [2] or Graphite [3]. Most of these frameworks have two major limitations.

On the one hand, they focus on one special type of geometric primitives, namely triangle or polygonal meshes. However there are a lot of applications where multiple types of data are needed, e.g., isosurfaces get extracted from volumetric data or meshes are generated from point clouds. Therefore these applications only focus on parts of the overall geometry processing pipeline. Moreover types like subdivision or spline curves and surfaces are often not supported by these frameworks and thus reduce their versatility. On the other hand, the applications are usually published under the GPL [4] and are therefore not usable in industrial projects leaving out another large group of potential users and contributors.

Our main goal is to provide a common software development platform for the majority of the geometry processing community. Researchers should be able to use the framework to significantly speed up development and focus on the

actual research project as most basic functionality is readily available. Industry should obtain a common platform to build their software upon while end users benefit from a unified look and feel for all algorithms using the framework. This large user community heavily improves development as existing code from other people can be re-used and combined to new algorithms.

In the next section we give an overview of our design goals for the OpenFlipper framework. Section 3 briefly describes its central API. In Section 4 the currently available features are presented, and in Section 5 OpenFlipper's scripting system is explained. Finally, some examples on how OpenFlipper simplifies the development process of two projects are given in Section 6.

2 Design Goals

The ultimate motivation for the implementation of a general application framework is that in most research groups various projects are being worked on in parallel and that a considerable percentage of the functionality is shared between those projects. Hence the primary goal of a common framework is to avoid implementation and code redundancy by providing a central repository of modules, e.g., for user interfaces, data file handling, rendering and others.

Another observation is that throughout the life cycle of a project (or of an algorithm) different aspects of software engineering become relevant. In the initial basic research phase, the software framework should allow the programmer to focus on the algorithm itself and it should provide a test bed in which different variants of the algorithms can be evaluated.

At a later stage of the development, the implemented functionality needs to be tested extensively. While the major mode of usage for our framework is that of an interactive application, extensive testing and repetitive execution of similar procedures is more effective in some kind of batch mode. Hence our system comes with a scripting interface that enables the flexible meta-implementation of control procedures that build on top of the available functionality in the C++ implemented modules.

Finally, we want our framework to also support the dissemination and commercialization of geometry processing functionality. This imposes two additional requirements at the technical as well as at the legal level. First, the design of ergonomic and intuitive user interfaces needs to be supported effectively. This is made possible through the option to implement custom designed user interfaces for groups of users with different levels of technical expertise, e.g., software developer or client users. On the legal level we have to resolve licensing issues (this is why we have put our framework under the LGPL license) and we have to offer a mechanism to exchange and license functionality without exchanging source code.

Our solution to the latter issue is that individual modules are implemented as plugins that can be loaded (as pre-compiled code) at runtime. The plugins can access the central functionality through a flexible API that also supports the communication between plugins. In addition we included a license checker

mechanism into the framework which enables the protection of individual plugins by identifying the computer on which the program is running.

Our vision is that the OpenFlipper framework will provide a software ecosystem within which an open source community can develop and share their implementations and where there is a seamless transition to and integration with commercial modules. A research group or company could provide some of their plugins as open source while for others a user license has to be obtained. Users can collect their personalized set of plugins satisfying their particular application requirements which can consist of a combination of free and commercial modules.

Currently, the OpenFlipper framework is used intensively in the context of our computer graphics and geometry processing teaching curriculum since it is perfectly suited for student projects.

Compared to other mesh processing software frameworks (e.g., Meshlab [2]) the two major distinctive features are (1) the scripting module which allows users to run OpenFlipper in batch mode and eases the definition of custom tailored interfaces and (2) the scenegraph structure that can handle multiple objects (even with different representations) simultaneously.

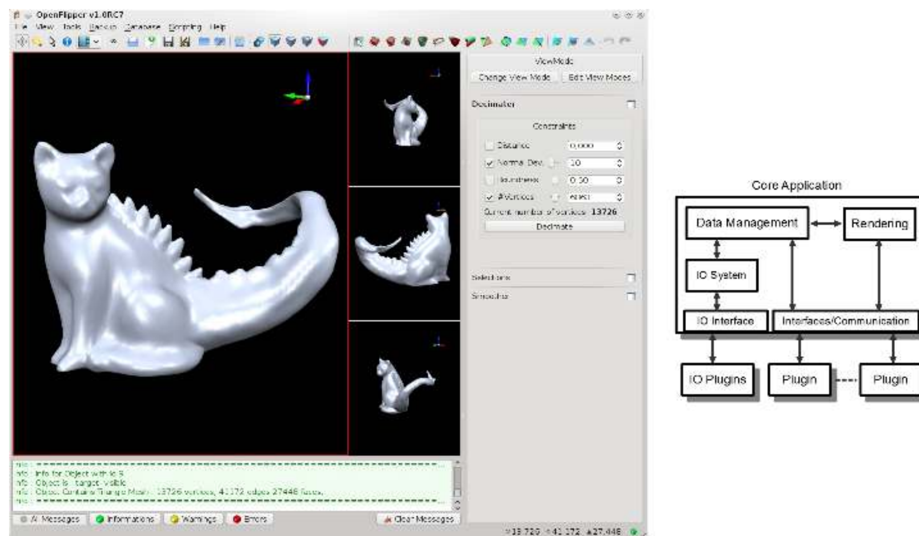

Fig. 1. Left: View of the OpenFlipper interface. It consists of a menu bar and several toolbars at the top, a configurable set of toolboxes on the right, and a large area for the viewers on the left. The viewer area can be split into multiple views rendering the scene with different visualization settings. Every part of the user interface can be extended and re-designed by plugins. Right: Framework scheme.

3 OpenFlipper API

A common design goal for today's software is cross-platform compatibility. As a consequence, a user interface library has to be used which must enable platform independent software development. We chose the *Qt cross-platform application and User Interface framework* [11] and adopted our API to some of the metaphors and concepts used by it.

Qt is based on a signal/slot architecture. Every event, e.g., a mouse click, results in a signal. Every signal can be connected to an arbitrary number of slots which are functions to be executed at each occurrence of the event. Events occuring in parallel are stored in a global event queue and get executed by the main thread in a serialized order.

OpenFlipper uses this signal/slot metaphor as its internal communication paradigm. OpenFlipper's core application communicates with the plugins by emitting signals and in turn activate functions at their counterpart. The functions provided by plugins or by the core application are grouped into separate interfaces which are subdivided into user interaction (keyboard or mouse events, picking), user interface specification (toolboxes, menus), file input/output, object modification, texturing, view control and many others. It is not necessary to implement all functions of an interface, allowing developers to quickly develop little plugins focusing on certain algorithms. The core application will automatically detect what has been implemented and only the relevant signals will be passed to the plugins avoiding function call overhead.

OpenFlipper is not bound to a specific type of geometric data. The core application is independent of the data representation as types are handled via container objects. These objects provide the rendering code and the actual data structure as well as functions for setting and retrieving the data. Due to this representation, arbitrary types can be added by the programmer with little additional implementation effort. For common data types like triangle and polygonal meshes OpenFlipper uses the OpenMesh library [1] which is a highly efficient halfedge based data structure. Additionally, we plan to integrate other commonly used data types like the tetrahedron meshes implemented in CGAL [13] or the volume data acquired with CT or MRI scanners.

The framework does not introduce an abstraction layer between algorithms and data types. Therefore, developers can directly access the underlying data structure (e.g., OpenMesh or CGAL) in their plugins. This results in low effort when porting existing algorithms to OpenFlipper as no code change has to be made for the algorithms themselves. OpenFlipper provides several iterators to retrieve objects in the framework. The iterators can be restricted to fetch objects of a specific data type, e.g., triangle meshes, or to fetch only user selections when working on multiple objects. After retrieval and modification of the data, a plugin has to send a signal and the core application will pass the information to all other plugins and updates the viewers accordingly. It is not the main purpose of the OpenFlipper framework to provide a library with geometry processing algorithms. It only provides the framework to use existing algorithms (such as the excellent CGAL [13] library) and easily test and develop new algorithms in

one system. Nevertheless, the free branch contains already a significant number of public domain implementations of state-of-the-art algorithms (see Section 4).

An integral part of the OpenFlipper framework is the rendering. The scene is represented by an OpenGL-based scenegraph and already provides rendering functions for the integrated data types. As almost all graphic cards support vertex buffers we use them for efficient rendering (with cache optimization to increase speed). OpenFlipper uses shaders for advanced rendering techniques but also provides fallbacks, if shaders are not supported by the used hardware. For presentations, the system can render images off-screen with very high resolution. Additionally we will integrate scene export for ray tracers and Sketch 3D (Latex code) in a future release. As the system is completely modular, developers can exchange the rendering for different kinds of data types with their own OpenGL code.

We also integrated a solution for tracing mouse clicks through the scene. The scenegraph implements color picking such that mouse events can be directly mapped to the clicked object or parts of the object, e.g., its faces or vertices.

4 Current Functionality

In this section we present a short overview of the currently available functionality of OpenFlipper and give some preview on functionality that will be added in future releases. The framework runs on Windows, Linux and Mac OS X. We provide IO plugins for OBJ (Wavefront), OFF (Object File Format), STL (stereolithography CAD), PLY (Polygon File Format),BVH (Biovision) and OM (OpenMesh). Furthermore we plan to integrate DAE (Collada), VTK (Visualization Toolkit) and NETGEN (automatic mesh generator, implemented along with tetrahedron meshes) support in one of the next releases.

Currently, OpenFlipper has built-in support for several data types. Triangle and polygonal meshes are supported via the integrated OpenMesh [1] data structure. B-Spline curves and surfaces are represented by proprietary data structures (IO supported through OBJ). Furthermore we provide a skeleton datastructure for skeleton based animations of polygon meshes and some simple primitives like planes, spheres and light sources. For future releases, we plan to integrate tetrahedron meshes (CGAL [13]), point clouds, polygonal lines and volumetric data (e.g., from CT or MRI).

For the current release a set of standard geometry processing algorithms is already included. The standard algorithms are implementations from recent research papers and constitute a basic repository for other software developers and users. The OpenFlipper framework only provides the interface to use them. Some of the algorithms have been developed in well maintained libraries such as CGAL and OpenMesh and are only imported here to show how simple it is to integrate new functionality by using existing code. The following list gives a short overview of the available functionality:

Selections: OpenFlipper comes with a large set of selection metaphors for all supported data types. The build in selections are: single click , surface and volume lasso, paint ball, boundary, connected component, and a flood filling based on normal deviation between adjacent primitives. The selection metaphors are implemented for triangle and polygon meshes and, where applicable, for other data types. For polygonal meshes we also distinguish between standard selections and special feature selections (e.g., important edges). For area based algorithms it is possible to select a handle and a modeling area.

Isotropic Remesher: The isotropic remesher implements the remeshing algorithm by Botsch and Kobbelt [6] and tries to establish an average target edge length specified by the user on an isotropic triangle mesh.

Decimater: The decimater based on [15, 16] decimates triangular meshes with respect to different constraints for the resulting meshes. It currently supports approximate distance constraints(error quadrics) to the original surface, upper bounds for the normal deviation, aspect ratio of triangles and a target complexity.

Smoother: This plugin implements a Laplacian smoother for triangular meshes [14]. It can smooth in both tangential and normal direction with C^0 or C^1 continuity while keeping the result within a prescribed distance to the original surface. OpenFlipper's feature selection can be used to specify edges or areas which are kept fixed.

Subdivider: Implementation of Loop [8], sqrt(3) [7], interpolatory sqrt(3) [9] and modified butterfly[10] subdivision for triangular meshes.

Mesh information: This plugin provides various information about arbitrary polygonal meshes. It computes statistics about aspect ratios, dihedral angles, bounding boxes, average edge length, genus, number of elements and many more.

Laplace, mean curvature, Gaussian curvature: The three plugins compute the laplace vector, mean curvature and Gaussian curvature for every vertex on a triangular mesh. The computed values are attached to the mesh and can be used by other plugins.

Texture control: This plugin provides a user interface to change texture files and to select an arbitrary property on a mesh to be used as texture coordinates. For example the Gaussian curvature plugin computes the curvature at each vertex and stores it as a property. Texture control reads these precomputed values and uses them as texture coordinates for a 1D texture. It also computes a histogram of the values and provides the user with additional functions like clamping values or taking the absolute value of them.

Slicing: Clipping planes can be created by this plugin, enabling the user to look inside of objects. This is especially useful when analyzing tetrahedron meshes like the ones generated by CGAL [13].

Topology control: This plugin implements elementary topological operations for meshes like flipping or splitting edges and adding or removing faces.

Scripting: This scripting plugin controls the integrated scripting environment. It collects functions available to scripting and provides a script editor. Section 5.1 gives a more detailed description of OpenFlipper's scripting capabilities.

Visual scripting: The visual scripting plugin provides a higher level interface to the scripting module. This plugin is further described in Section 5.2.

5 Scripting

OpenFlipper provides a variety of modules to ease the development of an interactive application. At a later stage of software development extensive testing of algorithms is required. OpenFlipper provides a scripting environment integrated into the framework to automate such processes in a batch mode. Qt already ships with an excellent scripting system that is used as the basis for OpenFlipper's scripting environment. As the language follows the ECMA-262 standard [12] which is also used for JavaScript, the syntax is familiar to a large number of developers.

The scripting system of OpenFlipper can be divided into two major parts. The first one is the general *text based scripting* system for experienced developers while the second one is a high level *Visual Scripting Interface* built on top of the general scripting.

5.1 General Scripting

OpenFlipper includes a text based scripting editor and interpreter. The editor collects and shows all available functions exported by the plugins, a description of their functionality and parameters. Scripts are visualized with syntax highlighting and can be directly executed without any compilation. Each function provided by a plugin can be made available to the scripting system.

Basic types like vectors or matrices are known to the system and can be used or manipulated directly. Like JavaScript, the language provides loops, conditionals, input/output and many other standard operators. As scripts are evaluated at runtime, all existing algorithms in OpenFlipper can be used and controlled via the scripting system without having to recompile code. This is especially useful when testing and evaluating algorithms or trying to find optimal parameters for a set of algorithms.

The scripting system is also capable of modifying and extending the user interface. Qt includes the Qt Designer tool which can be used to generate user interfaces and toolboxes. The user interface specification files generated by this graphical designer tool can be loaded at run time and connected to all existing algorithms via the scripting language. Consequently no change to a plugin is necessary for creating a new interface, a simple script is sufficient. The quadrilateral remeshing application described in Section 6.1 uses this option.

5.2 Visual Scripting Interface

Scripting is a powerful tool to combine simple algorithmic blocks to more complex algorithms. However, a programming or scripting language is usually too complex for end users. To support less technically-experienced users in generating scripts, we implemented the more abstract Visual Scripting Interface [21]. The Visual Scripting Interface is build directly on top of OpenFlipper's text based scripting system. The visual script editor is a data flow based block editor inspired by the block or filter based processing in audio or video processing applications [20]. The algorithms in OpenFlipper are represented as blocks with separate inputs and outputs. A simple example is an isotropic remesher. Input and output for this algorithm are surface triangle meshes. Additionally there can be other input parameters like, e.g., average edge length for the remeshed output. Figure 2 shows a simple example for such a visual script. This script consists of only three blocks. The first block computes the average edge length of an input mesh. Afterwards the computed length is passed to a math block and divided by a user specified number. The result is then passed to the isotropic remesher that uses the input value as the target edge length for its output mesh. The execution order for the different blocks can get fairly complicated, so the user has to define an order in which the algorithms are called. This is visualized by the data flow connections. For blocks that don't change objects (e.g., math blocks) the execution order is computed automatically.

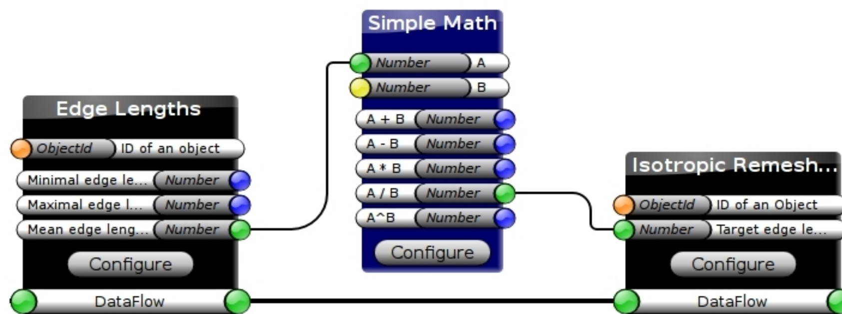

Fig. 2. Remeshing algorithm in the Visual Scripting Editor

Many algorithms require user interaction to select an object to work on. The visual scripting system provides several interactive blocks, allowing to select objects or asking for user input.

From the implementational point of view every block in the editor is associated with an xml file containing in- and output specifications as well as small code snippets which represent the blocks in the final script. Every visual script is therefore parsed, the blocks from the xml files and all variables are connected to a documented OpenFlipper script which optionally can be viewed and modified

by the user in the text based script editor. We have observed that these generated scripts provide an excellent foundation to learn the OpenFlipper scripting language. Users can read the code and use it as a starting point for creating more complex algorithms. The scripts are documented and changes made can be directly executed to get new results.

As the scripting blocks are defined and composed from simple xml files, the visual scripting system is not restricted to OpenFlipper's language. Therefore the editor and its components can be used to create script code for arbitrary languages.

6 Industrial Use Cases

A major design goal for OpenFlipper was to provide a toolkit allowing us to reduce the time and implementational overhead when converting research code to an end user application. This requires a solid base of working code that can be legally used in commercial and open source projects (LGPL). Researchers are provided with a stable toolkit, enabling them to focus on implementation and testing of new algorithms while visualization, selection or analytic tools are readily available.

Additionally, the user interface in research and end user applications is usually quite different. As OpenFlipper allows us to create an additional interface on top of the existing functionality, only little effort has to be spend to abstract the interface while keeping the original interface available for expert users.

There are already several commercial projects using OpenFlipper as their platform. The projects provide continuous improvements and new features to OpenFlipper's freely available parts and algorithms. A lot of basic algorithms are implemented for these projects which will also be published as open source and are therefore usable and valuable for the community. In the following sections, we present two of these projects where OpenFlipper significantly reduced the coding effort during development.

6.1 Quadrilateral Remeshing

Based on the OpenFlipper toolkit, a software for generating high quality quadrangular meshes from unstructured triangle meshes [17] has been developed in the context of a commercial project. The user can control the algorithm's output by providing additional constraints for the output structures and therefore modify the resulting quadrangulation.

In the development process of this project the interaction metaphors already defined in OpenFlipper have been used to define these constraints. One of the constraint controls is OpenFlipper's selection system. The user can select important edges and the algorithm uses the selection as a guidance for the final edge directions. Additionally, OpenFlipper provides freely drawable polygonal lines on surfaces which are also used by the system to control the final output. Furthermore the selection system is used to specify where singular vertices

should be positioned. All interaction metaphors, visualization, data types and input/output functions for this algorithm were already provided by the toolkit.

At its frontend the algorithm makes extensive use of multiple interfaces. The implementation consists of several parts which are implemented in independent plugins. The first plugin computes the principal curvature directions on the input mesh. The second plugin computes, based on the principal directions and possible user hints, a direction guiding field which is used to control the edge flow in the resulting quadrangular mesh. The third plugin generates a parametrization and extracts a quadrangular mesh. The interfaces to all plugins are available to the professional user while a simple unified interface exists showing only the relevant steps and settings while hiding the remaining parameters (with empirically derived defaults) from the user. This additional interface is purely defined as an OpenFlipper script and can be loaded and even modified at run time.

The mixed-integer quadrangulation solver used in [17] is also freely available as a separate library (CoMISo [18]). An example for the algorithm's output is shown in Figure 3.

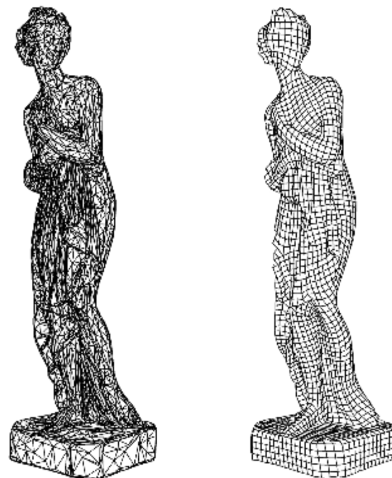

Fig. 3. Model of Iphigenie as an unstructured triangle mesh and the result after the quadrangular remeshing algorithm.

6.2 Car Modeling

In this project [19], a semi-automatic approach to efficiently and robustly recover the characteristic feature curves of a given free-form surface has been developed where the input is not required to be a proper manifold. The technique supports a sketch-based interface implemented in OpenFlipper where the user must roughly sketch the feature location by drawing a stroke on the input mesh. For this type

of interaction OpenFlipper's picking system provides functions that return the 3D position of a mouse click in the scene. The system then snaps the initial sketch curve to the correct position based on a graph-cut optimization scheme that takes various surface properties into account. Additional positional constraints can be placed and modified manually which allows interactive feature curve editing. The feature curves can be used as handles for surface deformation, since they describe the main characteristics of an object. The system allows the user to manipulate a curve while the underlying surface adopts itself to the deformed feature.

During development of this project a lot of the existing functionality of OpenFlipper has been used and therefore significantly reduced the coding effort for this project. No rendering code was required as it was already available for the B-Spline and mesh data types used in this project. For these types the IO and file management already existed in the framework. Figure 4 shows an example for modeling a car using the final application.

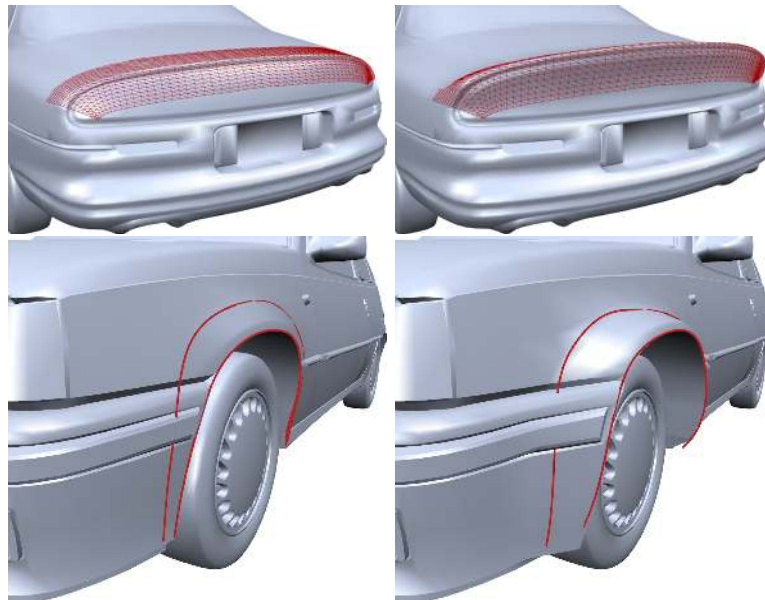

Fig. 4. Car modelling implemented on top of OpenFlipper. Left: Original models, Right: Modified models.

7 Conclusion and Future Improvements

In this paper we presented our OpenFlipper software framework. We developed a system which is should be helpful for the majority of the geometry processing community.

By now OpenFlipper is used for many different purposes. Students use it for learning the basics of computer graphics and especially geometry processing and rendering. Researchers use it to develop and test algorithms while companies benefit by directly using research code in an end user application with only little effort. All of these users provide updates and extensions to the freely available part of the framework. Therefore the functionality of OpenFlipper rapidly increases and it has a growing toolbox of basic algorithms, interaction metaphors and rendering functions which has been given back to the community by the various contributors as free software.

The user community is currently developing a lot of new functions for OpenFlipper. These include support for more file formats (Netgen, VTK, Collada), new data types like point splats or tetrahedron meshes and many algorithms working on them. Figure 5 shows a preview of the tetrahedron meshes.

All this functionality simplifies development and enables the people to focus on their creative work when inventing new algorithms, making OpenFlipper a valuable framework for the community.
The source code and executables are available at our website [22].

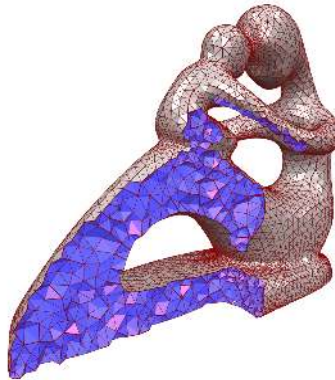

Fig. 5. Preview of a tetrahedron mesh in OpenFlipper.

Acknowledgements. We would like to thank all contributors, users, and developers who support the development of OpenFlipper.

References

1. Botsch, M., Steinberg, S., Bischoff, S., Kobbelt, L. and RWTH Aachen: OpenMesh - A generic and efficient polygon mesh data structure. In: OpenSG Symposium 2002
2. Cignoni, P., Callieri, M., Corsini, M., Dellepiane, M., Ganovelli, F., Ranzuglia, G.: Meshlab: an open-source mesh processing tool. In Sixth Eurographics Italian Chapter Conference 2008, 129–136.
3. Graphite, http://alice.loria.fr/index.php/software.html
4. GNU General Public License, http://www.gnu.org/licenses/gpl.html
5. GNU Lesser General Public License, http://www.gnu.org/licenses/lgpl.html
6. Botsch, M., Kobbelt, L.: A remeshing approach to multiresolution modeling. In: SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing, 185–192
7. Kobbelt, L.: Sqrt(3)-Subdivision. In: Proc. ACM SIGGRAPH '00, 103–112
8. Loop, C. T.: Smooth Subdivision Surfaces Based on Triangles. In M.S. Thesis, Department of Mathematics, University of Utah, 1987
9. Labsik, U., Greiner, G.: Interpolatory sqrt(3)-Subdivision. In: Comput. Graph. Forum, 2000
10. Zorin, D., Schröder, P., Sweldens, W.: Interpolating Subdivision for Meshes with Arbitrary Topology. In: Proceedings of Siggraph 1996, 189–192
11. Qt cross-platform application and UI framework , http://qt.nokia.com
12. Standard ECMA-262, ECMA Script Language Specification, 5th edition (December 2009)
13. CGAL: Computational Geometry Algorithms Library, http://www.cgal.org
14. Botsch, M., Pauly, M. ,Kobbelt, L., Alliez, P., Lévy, B., Bischoff, S., Rössl, C.: Geometric modeling based on polygonal meshes. In: SIGGRAPH '07: ACM SIGGRAPH 2007 courses
15. Kobbelt, L., Campagna, S. , Seidel, H.: A General Framework for Mesh Decimation. In: Proceedings of Graphics Interface 1998, 43–50
16. Garland, M., Heckbert, P.: Surface simplification using quadric error metrics. In: SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques 1997, 209–216
17. Bommes, D., Zimmer, H., Kobbelt, L.: Mixed-integer quadrangulation. In: ACM SIGGRAPH 2009 papers, 77:1–77:10
18. CoMISo: Constrained Mixed-Integer Solver library, http://www.graphics.rwth-aachen.de/comiso
19. Dekkers, E., Kobbelt, L., Pawlicki, R., Smith, R.: A sketching interface for feature curve recovery of free-form surfaces. In: Computer-Aided Design 2011 ,In Press
20. Pavic, D., Schönefeld, V., Krecklau, L., Habbecke, M., Kobbelt, L.: 2D Video Editing for 3D Effects. In: VMV 2008, 389-398
21. Kasprzyk, D.: Diploma Thesis on Optimized User Interface for Geometry-Algorithms. RWTH Aachen University 2009
22. OpenFlipper website, http://www.openflipper.org