Variational Methods for Discrete Surface Parameterization. Applications and Implementation.

vorgelegt von
Dipl.-Math. techn. Stefan Sechelmann

von der Fakultät II - Mathematik und Naturwissenschaften
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften
– Dr. rer. nat. –

Promotionsausschuss

Vorsitzender:	NN
Gutachter/Berichter:	Prof. Dr. Alexander I. Bobenko
	NN

Tag der wissenschaftlichen Aussprache: NN

Berlin, den 07. Januar 2013

Contents

List of Figures

Chapter 1

Discrete Differential Geometry - Software Packages

1.1 Introduction

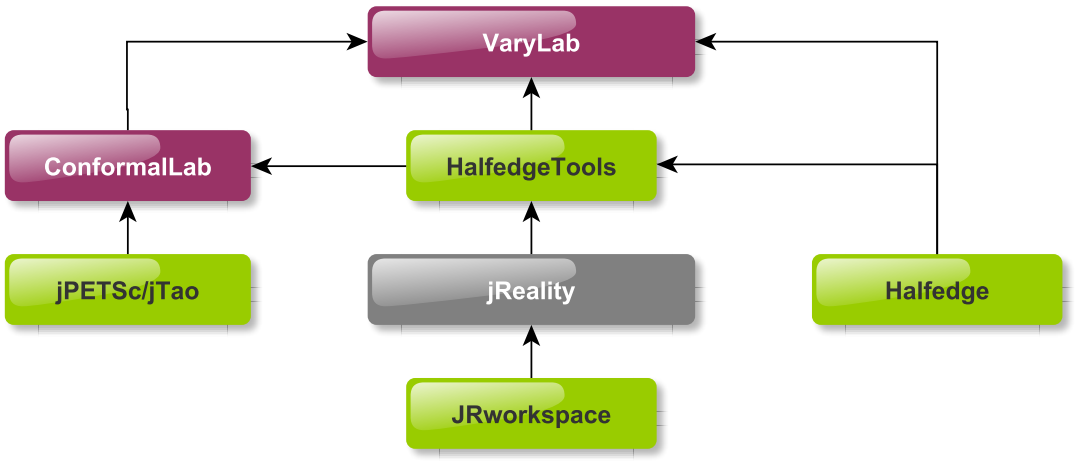

Figure 1.1: Software architecture and dependencies of the DDG Framework. JTEM library packages (green), mathematical software packages (red).

In the field of Discrete Differential Geometry (DDG) there is a special need for experiments conducted with the help of computer software. Especially if the methods of DDG are applied to problems in computer graphics, geometry processing, or architecture, algorithms have to be implemented and convincing examples have to be presented. Additionally a suitable visualization of the results has to be included in a state-of-the-art publication.

There is a growing knowledge of software development in the mathematical community. This is due to the curricula of universities which started to include programming courses for undergraduate students with a visualization emphasis, e.g., [Geo, Cal]. It enables students to extend

their abilities of creating visualizations and mathematical software, where former generations of students solely used the visualization abilities of standard computer algebra packages like Mathematica or MatLab.

The audience of this chapter is two-fold. On the one hand it is students creating their visualizations of surfaces and develop algorithms. On the other hand it is researchers in the field of discrete differential geometry who need a stable data structure and programming infrastructure to get the job done.

This Chapter is the description and getting-started manual of a set of software packages (Berlin DDG Framework) written in Java. They are specifically designed for the creation of custom interactive software for experiments with algorithms and geometries treated within DDG. It is currently beeing used for teaching a mathematical visualization course at TU-Berlin [Geo] and for research projects within the geometry group.

Section 1.2 introduces the JRworkspace library of the Jtem project [dt13b]. It is the foundation of any application created with the DDG Framework. It is also the user interface basis of Jreality, a mathematical visualization library that uses JRworkspace as plug-in and user interface tool [dt13a]. Section 1.3 introduces the Halfedge and HalfedgeTools package. They implement half-edge data structure and various user interface tools and algorithms for interaction and editing. In Section 1.4 we describe the software ConformalLab. This package implements the methods of the publications [BPS10, Sec12, SRB12, BSS]. Section 1.5 introduces VaryLab, the software implementation of the methods described in the publications [LGSR11, LSRG12, SRB12]. This package is also released to partners of the development group as VaryLab[Gridshells], VaryLab[Ultimate], or even online as VaryLab[Service].

Figure 1.1 shows the dependencies of the packages. Every application depends on JRworkspace which implements plug-in functionality. It is the basis of the Jreality plug-in system. HalfedgeTools is using Jreality for visualization and is build on top of the Jtem project Halfedge. ConformalLab and VaryLab use Jpetsc/Jtao to perform numerical optimization. Their algorithms are implemented as JRworkspace plug-ins.

The development of the described software is joint work with Thilo Rörig (HalfedgeTools, VaryLab), the Jreality members [dt13a], Hannes Sommer (Jpetsc/Jtao) [Som10], Ulrich Pinkall and Paul Peters (JRworkspace), and Boris Springborn (Halfedge).

1.2 JRworkspace - Java API for modular applications

JRworkspace is part of the Jtem family of software projects [dt13b]. It defines a simple API to create modular Java applications. It consists of three basic classes (Listings 1.2, 1.3, and 1.4). The project contains a reference implementation that supports the creation of Java Swing applications using the JRworkspace API. This implementation is used in all applications described in this work.

1.2.1 Plug-ins and the controller

In JRworkspace a module is called plug-in and extends the abstract class Plugin (Listing 1.2). The idea is that a plug-in can be installed by the controller calling its `install` method or uninstalled via the `uninstall` method. You can think of it as a feature added to your program. In particular there is no more than one instance of a plug-in class in a program.

A plug-in has a life-cycle during the runtime of the program which includes these basic steps:

instantiation	1	set default plug-in state
`restoreStates`	2	load plug-in state from `Controller`
`install`	3	calls `getPlugin` to obtain dependent plug-ins
-	4	program execution
`storeStates`	5	stores state values in the `Controller`
(`uninstall`	6	clean up)

In step 1 before installation of a plug-in the controller calles the `restoreStates` method. During runtime of the application the plug-in can interact with possible user interface it created during installation or offer services to other plug-ins. Before program termination or before uninstall the `storeStates` method is called. The plug-in is supposed to store its state values by calling the `storeProperty` method of the controller. Inter-plug-in-communication is done via the `getPlugin` method of the controller. A plug-in should call `getPlugin` from within the `install` method to obtain an instance of a dependent plug-in. The `getPlugin` method always returns the same instance of a plug-in so its result can be stored by the `install` method for later reference, see for example Listing 1.1.

Step 6 `uninstall` is only used with dynamic plug-ins that support this operation.

```
1  public class MyPlugin extends Plugin {
2    private DependentPlugin dependency = null;
3    private double doubleState = 0.0;

5    public void helloPlugin() {
6      String depName = dependency.getPluginInfo().name;
7      System.out.println("I am a plug-in. I depend on " + depName);
8    }
9    @Override
10   public void storeStates(Controller c) throws Exception {
11     c.storeProperty(MyPlugin.class, "doubleState", doubleState);
12   }
13   @Override
14   public void restoreStates(Controller c) throws Exception {
15     doubleState = c.getProperty(MyPlugin.class, "doubleState", 1.0);
16   }
17   @Override
18   public void install(Controller c) throws Exception {
19     dependency = c.getPlugin(DependentPlugin.class);
20   }
21 }
```

Listing 1.1: A simple plug-in class. It depends on a plug-in called `DependentPlugin` and has the property `doubleState`. It provides the method `doWork` that prints some message.

```
1  public abstract class Plugin {
2
3    public PluginInfo getPluginInfo() {
4      return PluginInfo.create(getClass());
5    }
6
7    public void install(Controller c) throws Exception{}
8
9    public void uninstall(Controller c) throws Exception {}
10
11   public void restoreStates(Controller c) throws Exception {}
12
13   public void storeStates(Controller c) throws Exception {}
14
15   @Override
16   public String toString() {
17     if (getPluginInfo().name == null) {
18       return "No Name";
19     } else {
20       return getPluginInfo().name;
21     }
22   }
23
24   @Override
25   public boolean equals(Object obj) {
26     if (obj == null) {
27       return false;
28     } else {
29       return getClass().equals(obj.getClass());
30     }
31   }
32
33   @Override
34   public int hashCode() {
35     return getClass().hashCode();
36   }
37
38 }
```

Listing 1.2: The `Plugin` base class. Note that plug-ins are equal if their classes are. It is not supported to have multiple instances of the same plug-in class installed.

```
1  public interface Controller {

3    public <T extends Plugin> T getPlugin(Class<T> clazz);

5    public <T> List<T> getPlugins(Class<T> pClass);

7    public Object storeProperty(Class<?> context, String key, Object property);

9    public <T> T getProperty(Class<?> context, String key, T defaultValue);

11   public <T> T deleteProperty(Class<?> context, String key);

13   public boolean isActive(Plugin p);

15 }
```

Listing 1.3: The `Controller` interface

```
1  public class PluginInfo {

3    /** The plug-in name */
4    public String name = "unnamed";
5    /** The vendor name of the plug-in */
6    public String vendorName = "unknown";
7    /** An email address of this plug-ins vendor */
8    public String email = "unknown";
9    /** An icon which will be the plug-ins icon
10    *  in the application */
11   public Icon icon = null;
12   /** An URL to the Documentation of the plug-in. */
13   public URL documentationURL = null;
14   /** Indicates if this plug-in is meant to be dynamically
15    *  installed or uninstalled */
16   public boolean isDynamic = true;

18   public PluginInfo() {
19   }
```

Listing 1.4: The `PluginInfo` plug-in meta data class

Bibliography

[BPS10] Alexander I. Bobenko, Ulrich Pinkall, and Boris Springborn. Discrete conformal maps and ideal hyperbolic polyhedra. Preprint; `http://arxiv.org/abs/1005.2698`, 2010.

[BSS] Alexander I. Bobenko, Stefan Sechelmann, and Boris Springborn. Uniformization of discrete Riemann surfaces. in preparation.

[Cal] Caltch Discretization Center. DDG lecture notes and assignments. `http://brickisland.net/cs177/`.

[dt13a] JREALITY development team. JREALITY website, 2013. `http://www.jreality.de`.

[dt13b] JTEM development team. JTEM website, 2013. `http://www.jtem.de`.

[Geo] Geometry Group@TU-Berlin. Mathematical visualization undergraduate course. `http://www3.math.tu-berlin.de/geometrie/Lehre/{WS08-SS13}/MathVis/`.

[LGSR11] Elisa Lafuente Hernández, Christoph Gengnagel, Stefan Sechelmann, and Thilo Rörig. On the materiality and structural behaviour of highly-elastic gridshell structures. In C. Gengnagel, A. Kilian, N. Palz, and F. Scheurer, editors, *Computational Design Modeling: Proceedings of the Design Modeling Symposium Berlin 2011*, pages 123–135. Springer, 2011.

[LSRG12] Elisa Lafuente Hernández, Stefan Sechelmann, Thilo Rörig, and Christoph Gengnagel. Topology optimisation of regular and irregular elastic gridshells by means of a non-linear variational method. In L. Hesselgren, S. Sharma, J. Wallner, N. Baldassini, P. Bompas, and J. Raynaud, editors, *Advances in Architectural Geometry 2012*, pages 147–160. Springer, 2012.

[Sec12] Stefan Sechelmann. Uniformization of discrete Riemann surfaces. In *Oberwolfach Reports*, 2012.

[Som10] Hannes Sommer. jPETScTao JNI library web page, 2010. `http://jpetsctao.zwoggel.net/`.

[SRB12] Stefan Sechelmann, Thilo Rörig, and Alexander I. Bobenko. Quasiisothermic mesh layout. In L. Hesselgren, S. Sharma, J. Wallner, N. Baldassini, P. Bompas, and J. Raynaud, editors, *Advances in Architectural Geometry 2012*, pages 243–258. Springer, 2012.

Acknowledgements