Faculty of Cognitive Science

Bachelor Thesis

Romeo: Scripting Environment with interactive Visualizations

Author: Simon Danisch
sdanisch@email.de

Supervisor: Prof. Dr.-Ing. Elke Pulvermüller

Co-Reader: Apl. Prof. Dr. Kai-Christoph Hamborg

Filing Date: 01.02.2014

I Abstract

This bachelor thesis is about writing a simple scripting environment for scientific computing, with focus on visualizations and interaction. Focus on visualization means, that every variable can be inspected and visualized at runtime, ranging from a textual representation to complex 3D scenes. Interaction is achieved by offering simple GUI elements for all parts of the program and the visualizations. All libraries are implemented in Julia and modern OpenGL, to offer high performance, opening the world to scientists who have to work with large datasets. Julia is a novel high-level programming language for scientific computing, promising to match C speed, making it the optimal match for this project. -This section needs more work, and should probably be written in the end

II Table of Contents

I Abstract **I**

II Table of Contents **II**

III List of Figures **IV**

IV List of Tables **V**

V Listing-Verzeichnis **V**

VI List of Abbreviations **VI**

1 Introduction **1**
- 1.1 Field of Research and Problem . . . 1
- 1.2 Problem Solutions and Measurements of Success . . . 3
  - 1.2.1 Extensibility . . . 3
  - 1.2.2 Speed . . . 4
  - 1.2.3 Visualizations and Interaction . . . 4
- 1.3 Outlook . . . 4
- 1.4 Used Technologies . . . 4
  - 1.4.1 Julia . . . 4
  - 1.4.2 OpenGL . . . 4
- 1.5 Similar Work . . . 4
  - 1.5.1 Other Languages . . . 4
  - 1.5.2 Ipython Notebook . . . 4
  - 1.5.3 Matlab . . . 4
  - 1.5.4 Mathematica . . . 4
  - 1.5.5 Other Graphic acceleration APIs . . . 4

2 Background **5**

3 Architecture **6**

4 Results and Analysis **6**
- 4.1 Performance Analysis . . . 6
- 4.2 Extendability Analysis . . . 6
- 4.3 Usability Analysis . . . 6

5 Conclusion **6**
- 5.1 Discussion . . . 6
  - 5.1.1 Performance . . . 6
  - 5.1.2 Extensability . . . 6
  - 5.1.3 Usability . . . 6
- 5.2 Future Work . . . 6

Appendix **I**

A GUI **I**

III List of Figures

Abb. 1 Volume Visualization 1
Abb. 2 Architecture 6
Abb. 3 Prototype I

IV List of Tables

Tab. 1 FE Implementation comparison 1

V Listing-Verzeichnis

VI List of Abbreviations

1 Introduction

This Bachelor Thesis is about writing a fast and interactive visualization environment for scientific computing. As GUI elements and editable text fields are supplied, one can also write and execute scripts, and immediately visualize all bound variables of the script and edit them via simple GUI elements like sliders. The focus is on creating a modular library, that is written in a fast high-level language, making the library easy to extend. The introduction is structured in the following way. First, an introduction to the general field of research and its challenges is given. From these challenges, the problems relevant to this thesis will be extracted. Finally this chapter will conclude with a solution to the problem, how to measure the success and give an outlook on the structure of the entire Bachelor Thesis.

1.1 Field of Research and Problem

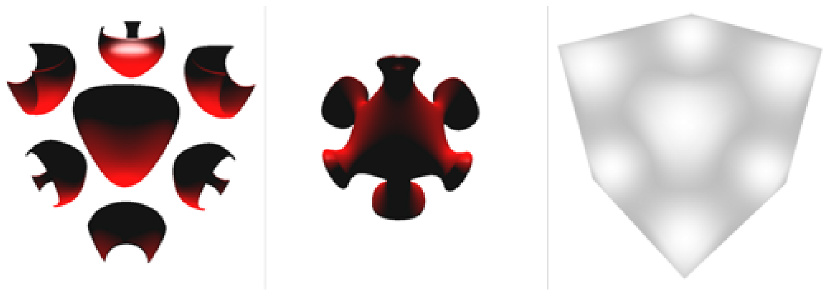


Figure 1: different visualizations of f(x,y,z)=sin(x/15)+sin(y/15)+sin(z/15), visualized with Romeo. From left to right: Isosurface, isovalue=0.76, Isosurface, isovalue=0.37, maximum value projection

Table 1: FE Implementation comparison

implementations	Language	Speed in Seconds
JFinEALE	Julia	9.6
Comsol 4.4 with PARDISO	Java	16
Comsol 4.4 with MUMPS	Java	22
Comsol 4.4 with SPOOLES	Java	37
FinEALE	Matlab	810

The general research field is making computer science more accessible and convenient to use. This is especially relevant for users, that don't have a broad computer science background, or programmers implementing complicated state of the art algorithms. These users can be found especially in scientific computing, which makes this field the focus of this bachelor thesis. More precisely, it focuses on research which involves writing short scripts, while visualizing the results. An example would be a material researcher, who is investigating different shapes and materials and their reaction to pressure. The researcher would need to read in the 3D object he wants to analyze, have an easy way to tweak the material parameters and it would be preferable to get instant feedback on how the pressure waves propagate through the object. Several demands by the researcher makes it challenging, to offer software for this area.

- **Money and Time is constrained**

  This means the research has to conclude quickly and most likely, it is not an option to employ a person or even a company to solve sub problems. From this we can deduce three preferences: If code needs to be written, it should be in an easy to understand high-level language. Computation times and feedback should be without delay to speed up development. The used libraries should be accessible to the researcher, because when something doesn't fit his demands, he most likely needs to resolve it himself due to the money constraint.

- **Visualizations**

  Visualizations are very important for two reasons: In some domains, problems become only managable by visualizing them. One example is a function describing a 3D volume, like f(x,y,z)=sin(x/15)+sin(y/15)+sin(z/15). This is a relatively simple function, which is relatively easy to interpret. But still, beats simply visualizing it 1. Secondly, research is getting published, together with visualizations explaining the results. As they represent the research to the public, they should be as understandable as possible and preferably look good. Offering a creative, interactive work flow can make this challenge considerably easier.

- **Speed**

  Speed can be both seen as a usability or a time/money constraint. While the money/time constraint is obvious, even slightly slower software can reduces usability by a large degree. Speed is also important for immidiate feedback. While scientists are used to slow computing times of their complicated algorithms, it still holds that it is far more satisfying and productive to immidiately see the results of your work. If the simulation of pressure takes 30s compared to 1s, this is not only frustrating, but has an immidiate influence on how many material combinations the researcher can try out.

1.2 Problem Solutions and Measurements of Success

* Write it in one, open source, high level, high performance language (Julia) * Write it in OpenGL * Open Source * Easy ways of creating GUIs * Modularity * Offer a broad variety of visualization

1.2.1 Extensibility

As previously deduced, extensibility is an important factor, which can decide, if a library is fit for scientific computing or not. It's not only that, but also a great factor determining growth of a software, as the more extensible the software is, the higher the probability that someone else contributes to it. In order to write extensible software, we first have to clarify what extensibility is. Extensible needs that the code is accessible. There are different levels of accessibility. The first level is closed source, where people purposely make the code inaccessible. While this is obvious, this is just a special case of not understanding the underlying language. Just shipping binaries without open sourcing the code, means that the source is only accessible in a language which is extremely hard to understand, namely the machine code of the binary. So other examples for inaccessibility are writing in a language that is difficult to understand. Other barriers are obfuscated language constructs, missing documentations and cryptic low level code. Further more the design of the library in the whole is an important factor for extensibility. It's not only important, that all parts are understandable, but also, that they can be reused and rebuild. Important factors here are modularity and short concise code. Modularity is important to see

Besides hiding the code, there are several other possibilities to make access to code hard. One example would be, that the code is written in a language which is not understood by the programmer. Most obviously, software isn't accessible when it's closed source, so being open source is the most crucial condition. But there are other more subtle conditions, which greatly reduce barriers to a software.

1.2.2 Speed

1.2.3 Visualizations and Interaction

1.3 Outlook

1.4 Used Technologies

1.4.1 Julia

1.4.2 OpenGL

1.5 Similar Work

1.5.1 Other Languages

1.5.2 Ipython Notebook

1.5.3 Matlab

1.5.4 Mathematica

1.5.5 Other Graphic acceleration APIs

2 Background

3 Architecture

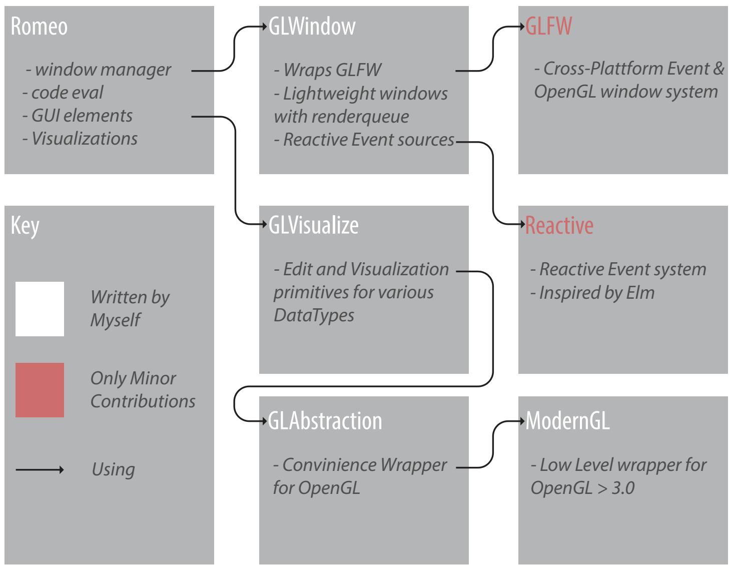


Figure 2: Main modules used in Romeo and their relation (simplified)

4 Results and Analysis

4.1 Performance Analysis

4.2 Extendability Analysis

4.3 Usability Analysis

5 Conclusion

5.1 Discussion

5.1.1 Performance

5.1.2 Extensability

5.1.3 Usability

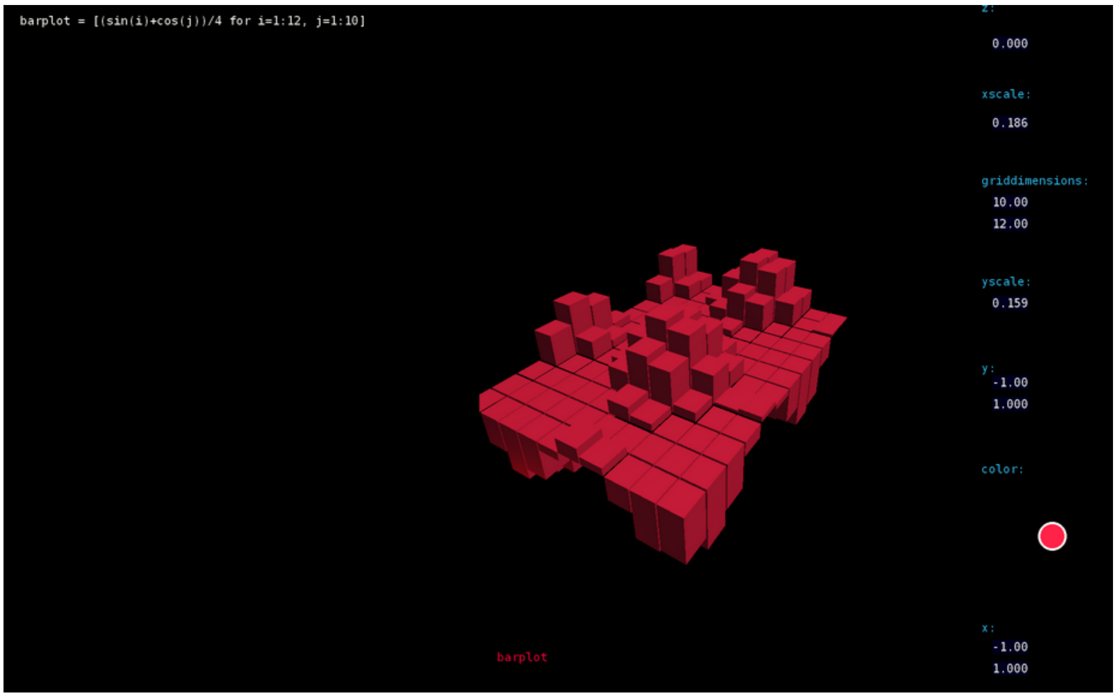
5.2 Future Work

Appendix

A GUI

A nice Appendix.

Screenshot



```
barplot = [(sin(i)+cos(j))/4 for i=1:12, j=1:10]
```

```
z:
 0.000

xscale:
 0.186

griddimensions:
 10.00
 12.00

yscale:
 0.159

y:
 -1.00
 1.000

color:

x:
 -1.00
 1.000
```

Figure 3: Screenshot of the prototype. Left: evaluated script, middle: visualization of the variable barplot, right: GUI for editing the parameters of the visualization

Official Statement

I hereby guarantee, that I wrote this thesis and didn't use any other sources and utilities than mentioned.

Date: ....

(Signature)