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

III List of Figures

IV List of Tables

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 the interactivity involves GUI elements and editable text fields, 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, Finally this chapter will conclude with how a solution to the problem can look like, how to measure the success and give an outlook on the structure of the entire Bachelor Thesis.

1.1 Field of Research

The general research field is developing convenient infrastructure for scientific computing. This Bachelor Thesis will deal with the subfield, which involves writing little scripts, while visualizing the results. An example would be a material researcher, investigating reactions of different materials to pressure, via simulating the pressure waves with the finite element algorithm. The researcher would need to read in the 3D meshes he wants to analyze, have an easy way to tweak the material parameters and it would be preferable to get instant feedback of how the pressure waves propagate. Also, in the end the researcher might want to interactively tweak the visualization parameters, to create a good looking view of the results in order to publish them.

This kind of work flow introduces several demands on the software. The researcher might have a good mathematical understanding of the physics behind the involved processes, but probably a rather poor computational science background. His research might involve tweaking the simulation algorithm itself, as it is possible that it doesn't meet his accuracy demands. At the same time the used packages should be easy to work with, to get the highest possible productivity. If the researcher wastes most of his time reading documentations, recompiling and setting up libraries, finding segfaults and the like and waiting for the computations to finish, the time spend on solving his research problem will be decreased dramatically. From this we can conclude, that three features are important to the researcher: 1. The code must be accessible and easy to extend 2. It must be really fast 3. The scripting language should abstract away memory management and should make it easy to implement algorithms.

This does not look difficult, but these demands together are actually very hard to solve and are an active research area. To clarify this point, take for example an algorithm to count the occurrence of a character in a string. Here are different implementations:

The last one actually has an overflow error, when there are too many occurrences of the character that one searches. The author actually still hasn't supplied a fix for this, simply because the resulting code was too hard to debug.

A common problem is, that there is always a trade off between usability and performance. High performant code is usually very hard to read, with inlined functions, unrolled loops, type specific optimizations without generalizity. Leveraging this problem is an ongoing effort addressed by a lot of researchers. This research is especially important for scientific computing, as the people working in that field usually are not experts in writing high performant code. A data scientist who needs to analyze terrabytes of twitter feeds, isn't usually someone who has the expertise and time to write highly optimized code, even though he really needs high performance. Otherwise the computation of his analysis might take several years.

So, in addition to offering a lot of extensible libraries and being easy to use, infrastructure for scientific computing must be really fast. The mentioned tools all have their own way of dealing with this demand, which I will describe in more detail in the chapter Background.

Introducing infrastructure meeting this demand would increase developing cycles immensely though. Updating a code base of 2000 lines of concise code, is much easier than rewriting 10.000 lines of obscure, highly optimized code. As a result, beginners won't even look at the code base, and even professionals won't like touching the code base.

In summary, there are two main demands. First, high usability and readily available libraries, while offering high performance, as otherwise working with large datasets becomes infeasible.

1.2 Problem

If you introduce the demand of having everything written in a high level, fast language, you will not find much software meeting this demand. The reason for this is, that there are just a few, recently released languages, that come close to satisfying this demand (see Figure **??**). Due to their novelty, they lack of good editors and a rich set of libraries. The goal of this bachelor thesis is to offer a scripting environment meeting the demand of being written in one of these new languages, while meeting the demands for scientific computing. In a Bachelor Thesis, one can certainly not meet all the demands, so this thesis will concentrate on being fast and flexible

1.3 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

1.4 Outlook

1.5 Motivation and Problem Description

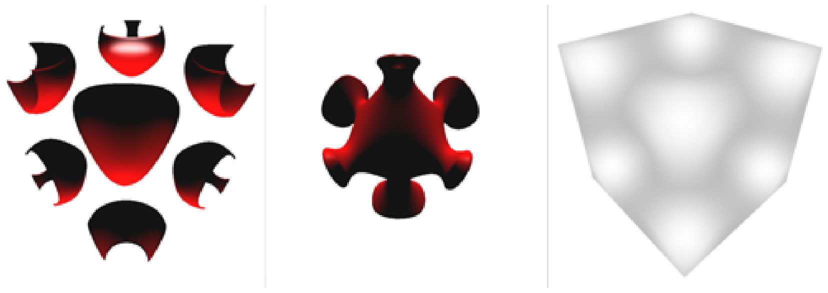


Figure 1: different visualizations of f(x,y,z)=sin(x/15)+sin(y/15)+sin(z/15), visualized with Romeo. From left to right: Isosurface, isovalue=0.76, Isosurface, isovalue=0.37, maximum value projection

When one tries to visualize a volume described by f(x,y,z)=sin(x/15f0)+sin(y/15f0)+sin(z/15f0), the limitations of the human mind becomes clear. While a computer doesn't have the creativity of humans yet, the task of visualizing this volume is a no-brainer for a computer (see Figure 1). In science, there are many comparable problems, which are unsolvable without the aid of a computer. For example making sense of one petabyte of data, predicting the folding of a novel protein, simulating black holes and many more. This naturally leads to computers being an essential part of state of the art research. Researchers have unique demands for their tools, which makes writing software for scientific computing an interesting research field. Consider, that researchers mostly do not have a background in computer science, some problems need a lot of computational power and due to the novelty of some problems hand tailored solutions are needed. In addition, high interactivity is a crucial feature, since a lot of research is a creative process which involves exploring many alternative solutions. This bachelor thesis is about writing a scripting environment with advanced visualization capabilities, while keeping the previous mentioned demands of scientific computing in mind. The resulting demands for the scripting environment are as follows.

No background in computer science puts a strong constraint on the used programming language. It should be high level and easy to learn as researchers should not be forced to waste their time with fixing memory corruptions or obscure language constructs. The entire suite should be open source and written in one language, to allow the researchers to modify every part easily, enabling them to create a solutions which is hand tailored to their particular problem. The two constraints of writing even the core libraries in one language and having also research areas with high demand for computational power means, that in addition to being high-level the chosen language must also be fast. As an example, the high level language Python can be substantially slower than a high-performance language like Fortran. In extreme cases, Python is up to 1155 times slower than Fortran [?], which means if a computation needs 1 day with Fortran, it could take 3 years within Python, rendering the problem intractable within Python.

Finally, the scripting environment should offer simple GUI elements like sliders, to enable the researcher a simple way to interact with the parameters in his script.

These considerations lead to the 3 main design choices: Julia, a novel scientific computing language, is chosen as the main language, Modern OpenGL for high performance graphic rendering and writing generic,Reusable open source code for a high amount of customizability. The feature set consists of parametrized scientific visualizations, simple GUIs for the parameters and a simple script editor.

-This section needs more quotation, to integrate it into an ongoing research problem. Also, some formulations have to be rewritten.

1.6 Used Technologies

1.6.1 Julia

1.6.2 OpenGL

1.7 Similar Work

1.7.1 Other Languages

1.7.2 Ipython Notebook

1.7.3 Matlab

1.7.4 Mathematica

1.7.5 Other Graphic acceleration APIs

2 Background

3 Design

3.1 Architecture Overview

3.2 OpenGL Wrapper

3.3 Window System

3.4 Event System

3.5 Visualization System

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

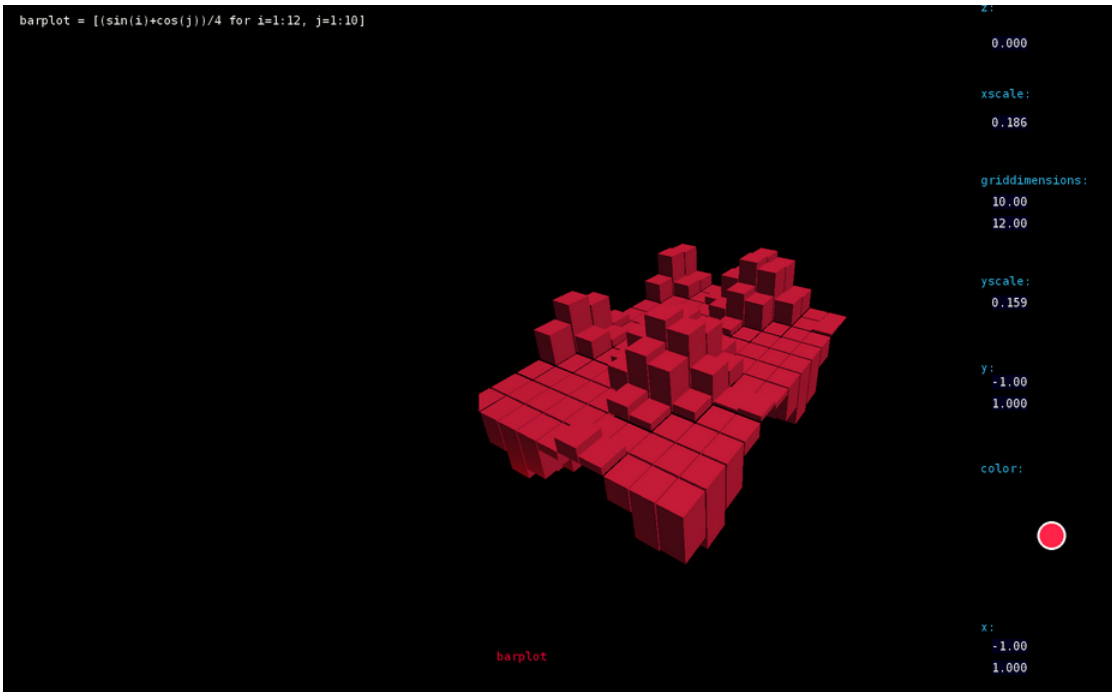
5.2 Future Work

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 2: 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)