A Research Proposal on Mathematically Rigorous and Computationally Efficient Representations of Geometry

Stephen Kelly

October 16, 2015

1 Introduction

Geometry is one of the earliest academic studies in mathematics. Research is constantly leading to new patterns and constructions. Many of these find highly practical applications in engineering. In this paper we will seek to study geometry by developing patterns that are sensible to computers and people. To obtain sensiblile patterns we address two different audiences. One of which is the computer which requires execution effiency. The other is people, who require lucid representation of code.

This project will try to refine existing implementations of geometry for computers and explore new procedures. As such, we will review the existing systems, their applications, and areas for improvement. Later we will discuss some of the areas for improvement in more detail, and establish a proposed scope for this project.

1.1 The Julia Programming Language

Programming languages are the grammar and syntax a computer presents to a user. This project is fundamentally exploratory in nature and seeks to generate understanding of geometric relationships. I plan to use Julia as a programming language for exploration. There are many reasons this language is ideally suited for such mathematical exploration.

Functions and Data Julia is a descendent of the Lisp family of programming languages. Similar to Lisp, it is an experiment in language design. Much of this advancement revolves around the representation of data and the execution of functions. The language optionally typed, which means function specialization on types is inferred. See below:

```
julia> increment(x) = x + 1
increment (generic function with 1 method)

julia> increment(1)
2

julia> increment(1.0)
2.0
```

the ‘increment‘ function was defined for any ‘x‘ value. When the ‘1‘, an integer type was passed as an argument, an integer was returned. Likewise when a floating point, ‘1.0‘ was passed, the floating point ‘2.0‘ was returned.

Let’s see what happens when we try a string:

```
julia> increment("a")
ERROR: MethodError: ‘+‘ has no method matching +(::ASCIIString, ::Int64)
Closest candidates are:
  +(::Any, ::Any, ::Any, ::Any...)
  +(::Int64, ::Int64)
  +(::Complex{Bool}, ::Real)
  ...
 in increment at none:1
```

The problem is that the ‘+‘ function is not implemented between the ‘ASCIIString‘ and ‘Int64‘ types. We need to either implement a ‘+‘ function which might be ambiguous, or specialize the function for ‘ASCIIString‘. A specific implementation is preferrable in this case:

```
julia> function increment(x::ASCIIString)
           ASCIIString([increment(c) for c in x])
       end
increment (generic function with 2 methods)
```

Since ‘ASCIIString‘ is a series of 8 bit characters, we can iterate over the string and increment each character individually. The ‘[]‘ indicates we are constructing an array of characters to pass to be passed to the ‘ASCIIString‘ type constructor. Now we see our example works:

```
julia> increment("abc")
"bcd"
```

What was demonstrated here is the concepts of specialization and multiple dispatch, both are highly coupled topics. Each function call in Julia is specialized for types if possible. This means the author only has to write a few sufficently abstract implementations of functions. If special cases occur multiple functions with different arity or type signatures can be implmented. Explicitly this is called multiple dispatch. In practice by the user this looks like abstracted code. To the computer, this means choosing the most specific, and thus performant method. Let’s go back to the integer and floating point example. Below is the LLVM assembly generated for each method:

```
julia> @code_llvm increment(1)

define i64 @julia_increment_21458(i64) { // <return type> <function name>(<arg type>)
top:
  %1 = add i64 %0, 1
  ret i64 %1 // return <return type> <return id>
}

julia> @code_llvm increment(1.0)

define double @julia_increment_21466(double) {
top:
  %1 = fadd double %0, 1.000000e+00
  ret double %1
}
```

The only real similarity is the line count. Note I have annotated the LLVM code so this is understandable. Each one of these functions are generated by the Julia compiler at run time. The REPL (Read-Eval-Print-Loop) allows interactive evaluation of Julia code. It is highly useful for exploration and testing of ideas.

Many of the concepts used for performance also serve as methods for expressability. In this case, multiple dispatch used by the compiler for specialization of functions reveals it self as a way for the user to specialize over many types. Revealing the role in which this paradigm allows Julia to achieve high performance is a matter to be developed in further sections.

Types, Immutables, and Parameters

Types and immutables are containers of data. The primary difference between the two is the notion of "mutability". Types are mutabile, immutables are immutable. What does this mean? Let's break something first:

```
julia> type FooIsMutable
           a
       end

julia> f = FooIsMutable(1)
FooIsMutable(1)

julia> f.a
1

julia> f.a = 2
2

julia> f.a
2

julia> immutable FooIsImmutable
           a
       end

julia> f = FooIsImmutable(1)
FooIsImmutable(1)

julia> f.a
1

julia> f.a = 2
ERROR: type FooIsImmutable is immutable
```

What just happened demonstrates the contract defined by mutability. Mutable objects, which is an instance of a type (i.e. 'f'), can have their fields (i.e. 'a') changed. Immutables cannot. The immutable contract helps develop a notion of functional purity. Practically this can be of great benefit to the compiler. For example:

```
julia> a = (1,2,3)
(1,2,3)

julia> b = typeof(a)
Tuple{Int64,Int64,Int64}

julia> isbits(b)
true

julia> a = ([1],[2],[3])
([1],[2],[3])

julia> b = typeof(a)
Tuple{Array{Int64,1},Array{Int64,1},Array{Int64,1}}

julia> isbits(b)
false
```

'isbits' ask the question "will this type be tightly packed in memory"? A 'Tuple' is a fixed-length set of linear, ordered, data. It has syntax for construction with '()'. In computations we want our data be close together for fast access. In modern times we call such data "cache friendly", or "cache localized".

Immutability helps us achieve this. Let's look that the types inside the 3-tuples and see their ‘isbits‘ status:

```
julia> isbits(Array{Int64,1})
false

julia> isbits(Int64)
true
```

Why is this the case? We see that ‘Int64‘ is bits, because it is literally 64 bits. ‘ArrayInt,64‘ is a data type

Solid Modeling Paradigms

The expression of solid bodies is fundamental in the development of any natural problem statement. For example, in diffusion we model the transfer of energy throughout a domain. An engineer might define such a domain with a model, say of an injection molding nozzle. Such a domain is difficult to describe in terms of a functional boundary. The development of modern computational tools for solid modeling have vastly different paradigms. Many

Functional Representation

Functional representation in computation centers around a signed, real-value function where the boundary is defined as ‘f(...) = 0‘. In ‘R3‘ this looks like ‘f(x,y,z) = 0‘. For modelling purposes we must add the additional constraint that the function evaluates to a negative inside the boundary. Further more the magnitude of the return value must correspond to the minimum distance between the point and the boundary.

Mesh

BRep

CSG

Exploration

Geometry Types

GeometryTypes.jl is a package for Julia that provides geometric strutures and relations. It was started early 2015 as the integration of Meshes.jl, ImmutableArrays.jl, HyperRectangles.jl, and FixedSizeArrays.jl. This package was able to resolve the relations between geometric structures. With the release of Julia version 0.4 is became possible to build the appropriate abstractions. For example ImmutableArrays represented a 3 dimensional vector with the concrete type ‘Vector3Int64‘. FixedSizeArrays introduced the dimensionality as a parameter as ‘Vector3,Int64‘. This means the notion of a fixed length vector can be abstracted over arbitrary dimensionality.

Simplices

Recently a ‘Simplex‘ type was added to ‘GeometryTypes‘. A ‘Simplex‘ is defined as the minimum convex set containing the specified points. The initial prototype

Distance Fields

Dual Numbers

Rvachev Functions

In the 1960's Vladimir Rvachev produced a method for handling the "inverse problem of analytic geometry". His theory consists of functions which provide a link between logical and set operations in geometric modeling and analytic geometry.[**?**] I believe the following anecdote helps elucidate the theory. While attempting to solve boundary value problems, Rvachev formulated an equation of a square as

$$a^2 + b^2 - x^2 - y^2 + \sqrt{(a^2 - x^2)^2 + (b^2 - y^2)^2} = 0$$

Implicitly, the sides of a square can be defined as $x = +/-a$ and $y = +/-b$. The union of these two is a square. By reducing the formulation of the square we can generalize an expression for the union between two functions.

$$\cup : f_1 + f_2 + \sqrt{f_1^2 + f_2^2} = 0$$

Likewise we can see that intersections and negations can be formed for logical completion.

$$\cap : f_1 + f_2 - \sqrt{f_1^2 + f_2^2} = 0$$

$$\neg : -f_1$$

These formulations can be modified for C^m continuity for any m. [**?**] In addition Pasko, et. al. have shown that Rvachev functions can serve to replace a geometry kernel by creating logical predicates. [**?**] Their research also establishes the grounds for user interfaces and environment description. For this work a practical implementation will most likely leverage their insights. Rvachev and Shapiro have also shown that using the POLE-PLAST and SAGE systems a user can generate complex semi-analytic geometry as well.[**?**]

1.2 Numerical Analysis

While a functional representation for geometry is mathematically enticing on its own, the power it gives for numerical analysis might be its greatest virtue. Numerical analysis justified the initial investigation by Rvachev early on. A boundary value problem on a R-Function-predicate domain allows for analysis without construction of a discrete mesh.[**?**]

One of the most general expositions in the English language of R-Functions applied to BVPs is Vadim Shapiro's"Semi-Analytic Geometry with R-Functions". [**?**] Unfortunately, no monographs about R-Functions exist in the English literature. Most literature is in Russian, however many articles presenting applied problems using the R-Function Method. [**?**] This is the topic of this project I stand to gain the most insight.

Mesh Slicing

Today the dominant paradigm for solid modeling has remained unchanged since the 1970's, relying primarily on Boundary Representation (B-Rep). It relies primarily on the manipulation and representation of edges, vertices, and faces to build a model. The primary mechanism for the representation is a "feature tree". While B-Rep is intuitive for users of a graphical environment, it is unwieldy as a textual and functional representation. This methods is natural for engineers and designers, but sacrifices parametric design. In addition, B-Rep requires the use of a geometry kernel to handle the interpretation of constraints and geometric construction. [**?**]

Geometry kernels often decouple functional representations from a user's design hierarchy which complicates numerical analysis.[**?**] This middle step of Computer Aided Engineering (CAE) is known as pre-processing. For example in the Finite Element Analysis (FEA) process the requires establishing proper aspect ratio, area, and connectivity of nodes. Research has shown that functional representations can simplify or eliminate these steps and algorithms.

In addition, designers targeting parametric design have turned to the methods of Constructive Solid Geometry (CSG), which works using manipulation of geometric primitives (half-spaces) as a level of abstraction. This enables parametric solids to be represented using operations and relations on primitive solids. CSG has been growing in popularity due to programs such as OpenSCAD[1], CoffeeSCAD[2], POVRay[3] and Thingiverse Customizer[4]. These programs are particularly popular for collaboration in conjunction with version control systems such as Git.

1.3 Closed Loop Optimization

Optimization could be studied using mathematical solid modeling. I have not completely researched existing literature on this topic, so I include this idea as a "stretch goal" for the project. Given a parametrized solid and numerical analysis, optimization might prove useful. Possible ideas include the minimization of mechanical stress on a solid or reduction of material volume.

2 Technologies

2.1 Julia

Julia is a programming language first released in early 2012 by a group of developers from MIT. The language targets technical computing by providing a dynamic type system with near-native code performance. This is accomplished by using three concepts: a Just-In-Time (JIT) compiler to target the LLVM framework, a multiple dispatch system, and code specialization.[**?**] The syntactical style is similar to MATLAB and Python. The language implementation and many libraries are available under the permissive MIT license.[5]

Benchmarks have shown the language can consistently perform within a factor of two of native C and FORTRAN code.[6] This is enticing for a solid modeling application and for numerical analysis, as the code abstraction can grow organically without performance penalty. In fact, the authors of Julia call this balance a solution to the "two language problem". The problem is encountered when abstraction in a high-level language will disproportionately affect performance unless implemented in a low-level language.

Since February 2014 I have been actively contributing to the Julia community through contributions to the base compiler and library, package maintenance and development, and general community support through forums and mailing lists. From September 2014 to May 2015 I developed a multiple material path planner in Julia for 3D printers at my co-op at Voxel8. There I learned to architect and optimize Julia code. When I first discovered Julia I realized it

[1]`http://www.openscad.org`
[2]`http://coffeescad.net/`
[3]`http://www.povray.org/`
[4]`http://www.thingiverse.com`
[5]`http://opensource.org/licenses/MIT`
[6]`http://julialang.org/benchmarks`

had great potential for CAD, and the I would like to use Julia as a media of exploration for the mathematical concepts and techniques during this project. While the language is relatively new and in development, sufficient documentation and an active community can help support software development. Many advanced and mature packages for fields such as operations research exist.[**?**]

2.2 Open Source

Along with Julia, I believe it is important to publish software frequently and openly. In the past the open source community has been very receptive to new ideas. I expect that over the course of the project we can attract external contributions to the software developed. This will help strengthen and quicken developments.

As such, I believe it will be important to improve the accessibility to the mathematical theory. Along with a report, I would like to improve and write Wikipedia pages to enable a greater audience. The existing Wikipedia page for Rvachev-Functions[7] contains simple concepts compared to the richness of the academic literature existing on the topic.

3 Conclusion

The motivation of this project is social. 3D printing has promise to be disruptive to economics and technological application. However it is only the physical media in which collaboration between people can change. The underlying mathematics of the *digital* media for collective action will be the focus of this project.

The intellectual challenge for this project will be mathematical. In particular, the question is *how can techniques for functional solid modeling and numerical analysis be applied to improve the state of 3D modeling?* My previous insight into the importance of parametric design and engineering analysis will drive the development of a solid mathematical understanding.

The application of this project will be through distillation of mathematical theory into software and documentation. In particular, libraries and scripts will provide means for others to experiment and improve our work. Documentation and explanations of theory will enrich the open source 3D modeling community.

I first encountered Rvachev functions in October 2013 while researching mathematical techniques to implement a CAD program. The existing literature is vast, however few practical implementations exist for the maker community to use. I expect this project to take 4/3 units of work during the 2014-2015 academic year. My hope is this will permit me to review the literature and develop applications of utility that might inspire more research. I appreciate your time in reviewing this proposal.

[7]`http://en.wikipedia.org/wiki/Rvachev_function`