GMac: The Next Generation (1 of 2)

by Ahmad Eid - Thursday, February 09, 2017

https://gacomputing.info/2017/02/09/gmac-tng-1/

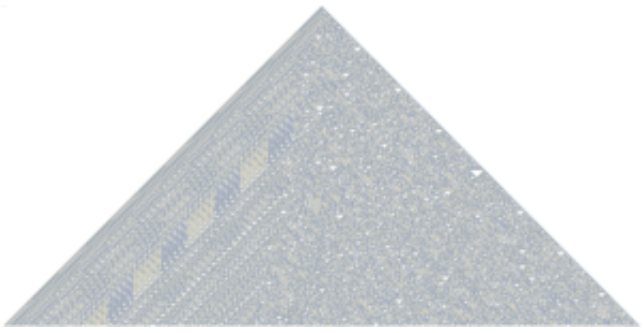
> The belief that complex systems require armies of designers and programmers is wrong. A system that is not understood in its entirety, or at least to a significant degree of detail by a single individual, should probably not be built. -- Niklaus Wirth.

[dropcap]I[/dropcap]n the summer of 2003, I was almost at the end of my M.Sc. thesis. One day I was surfing the web searching for some references to add to the thesis I'd been writing. I ran into Mikael Nilsson's interesting M.Sc. thesis "Geometric Algebra with Conzilla: Building a Conceptual Web of Mathematics". His work contains a nice introduction to Geometric Algebra, with accompanying UML-based conceptual models. It got me very curious about GA. I made some more searches finding a few papers, books, and tutorials. After reading some of them I was suddenly transferred into a new world of algebraic abstractions. For the first time in my life as a student, engineer, and software developer I could hope to express, explore, and manipulate geometric abstractions with such clarity. I then decided that my Ph.D. will be about exploring Geometric Algebra using the best way I know: developing software. It was a natural decision to make given GA's potential, but a hard and lonely journey to make. About 8 years have passed now developing my tool, GMac, into its current state all by myself passing through difficult events I could only have passed by Allah's well and mercy.

In this post and the next, I talk about this journey and some of the lessons I learned along the way. I talk about how the current GMac design came to appear and the joys I found learning the hard way that profound belief [1. Belief in Islam, or "Iman" in Arabic, has 6 pillars: Belief in Allah, Belief in the angels, Belief in the revealed books, Belief in the commissioned Messengers (peace be upon them), Belief in the resurrection and the events of Qiyamah, Belief in the predestination by Allah of all things, both the (seemingly) good and the (seemingly) bad. You can read more details about this central concept in here. You can also watch this beautiful lecture illustrating the profound effects of believing in these 6 pillars on Dr. Jeffrey Lang, an Associate Professor of Mathematics at the University of Kansas.], solid foundation, clear goals, and lots of patience are together the only way to realizing distant dreams.

Discovering The Computational Universe

> I had a very selfish reason for building Mathematica. I wanted to use it myself, a bit like Galileo got to use his telescope four hundred years ago. But I wanted to look, not at the astronomical universe, but at the computational universe. -- Stephen Wolfram.



Rule 30: Rich behavior can emerge from very simple computational rules. (source)

In the Autumn of 2005, I'd just begun my Ph.D. preparatory courses when I had another perceptual shift, this time in how I think about computations. I found Stephen Wolfram's 2002 fascinating book "A New Kind of Science" **(NKS)** after watching one of his lectures about it. After reading a few pages of NKS online, I couldn't stop until I finished reading the whole book, without the endnotes, in just 2 weeks. I

couldn't wait to buy the book because it would take at least a month to arrive back then so I read it online.

Reading this book made me rethink about many concepts of science, mathematics, and computation that I took for granted. Concepts like numbers, chaos, randomness, complexity, intelligence, mathematical models, and the nature of computation. Although I don't agree with everything he says in his book, Wolfram's insights and explorations are fascinating and inspiring for someone like me. The most intriguing of all the concepts he talks about is the Computational Universe. Wolfram used cellular automata and similar computational tools for mapping and mining the computational universe.

I instantly realized that Geometric Algebra can be a uniquely valuable tool in any attempt to explore the computational universe. I just needed the right computational vessel to make such explorations. I searched a lot but found none that suited my goals. I read that Wolfram had created **Mathematica** as his personal lab for doing computational experiments. All this made me decide to create my own GA-based tool for voyaging and exploring the computational universe, and so my journey began.

Creating The Perfect Library

> Good judgment comes from experience, and experience comes from bad judgment. -- Frederick P. Brooks

Object Oriented Design Principles (source)

When it comes to repetitive tasks, good software developers are the laziest of people; they prefer to create programs to do the repetitive tasks for them [2. Many online articles express this idea in one form or another as a common experience. For some examples, see here, here, and here.]. To be more accurate, their deep creativity drives their apparent laziness. They never want to do the same task twice, and if they have to, they never want to do it in the same way. They almost always enjoy finding the most general framework for doing several tasks at once with minimum effort; I'm one of those people.

I started my actual Ph.D. research work in 2007 by studying GA mathematics through reading the

excellent book "Geometric Algebra for Computer Science". In 2008 I started to investigate several software designs and implementations in hope to implement a single perfect universal GA-based software library.

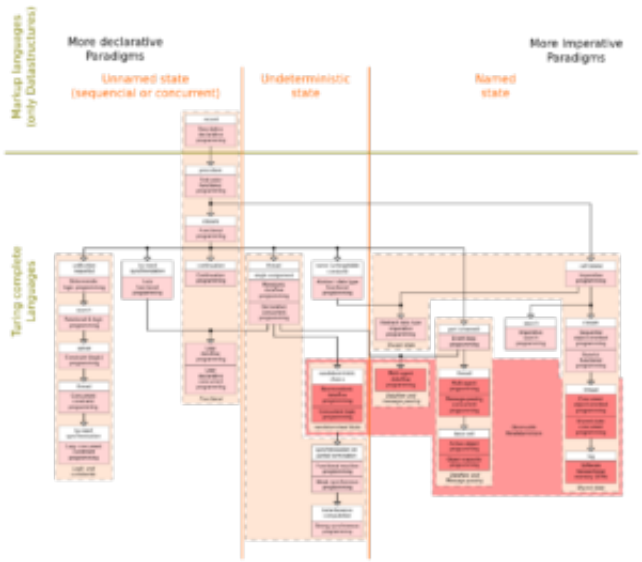
Driven by my creative laziness, I initially thought that the powerful unification GA brings to mathematics can be directly translated into a unified software library suitable for all Geometric Computing purposes. I was naive back then with little experience in the difficult problems that face designing large sophisticated multi-purpose software systems. I gradually realized that this perfect software library is a myth. While working on the development of two big decision support software systems based on relational databases for several years, I gradually gained experience in the true meaning of the phrase "Necessity is the Mother of Invention".

Successful software libraries are mainly driven by necessity not only by the elegance of some underlying abstraction, even a beautiful algebraic abstraction such as GA. Each particular application has its own set of necessities that we, software engineers, call requirements: functional, non-functional, financial, scope related, performance related, etc. There can never be a single perfect software library suitable for all applications in a wide-ranging domain like Geometric Computing.

Failing in my initial attempt I went higher in the level of abstraction. I had a good background in Object-Oriented (OO) Software Design. Why not, I asked myself, create an OO design based on the mathematical concepts of Geometric Algebra and then derive specific concrete implementations for specific applications. After all, this is how I learned about GA originally; using UML to diagram GA's main concepts and relations. I soon found that OO design is a good conceptual tool to study some domain, but there can never be a single OO design suitable for all purposes either; especially when good memory\processing performance is critical [3. I now know from first-hand experience that there are many deep conceptual problems associated with Object-Oriented Programming and the misuse of Object-Oriented Design. You can get a good idea by watching David West's lecture "OOP is Dead! Long Live OODD!"]. This got me to learn about other programming paradigms I've never considered reading about before.

Shifting Paradigms

> One principle problem of educating software engineers is that they will not use a new method until they believe it works and, more importantly, that they will not believe the method will work until they see it for themselves. -- W. S. Humphrey, The Personal Software Process



Programming Paradigms (source)

Back in 2009, I only knew about procedural programming and object oriented programming. I did some reading and discovered many other paradigms like functional, aspect-oriented, concurrent, dynamic, feature-oriented, and more. I realized that GA can be used equally well as an algebraic base for creating software in all of these paradigms. My deep creative laziness led me to the most general of all paradigms: Automatic Meta-Programming.

In April 2009 I bought and read the book "Generative Programming: Methods, Tools, and Applications" in which the authors present a grand tour of Generative Programming that is bound to become a classic. The book focuses on the generally unappreciated connection, back then, between Domain Specific Languages **(DSLs)** and Generative Programming as a motivation for future developments. I immediately started designing and implementing the first prototype of my dream GA-based software vessel for exploring the computational universe; I called it GMac. The prototype took a couple of months to be functional for the purpose of completing my Ph.D. [4. All the details of this GMac prototype can be found in my Ph.D. You can download a softcopy from here, or buy a hard copy from Amazon.].

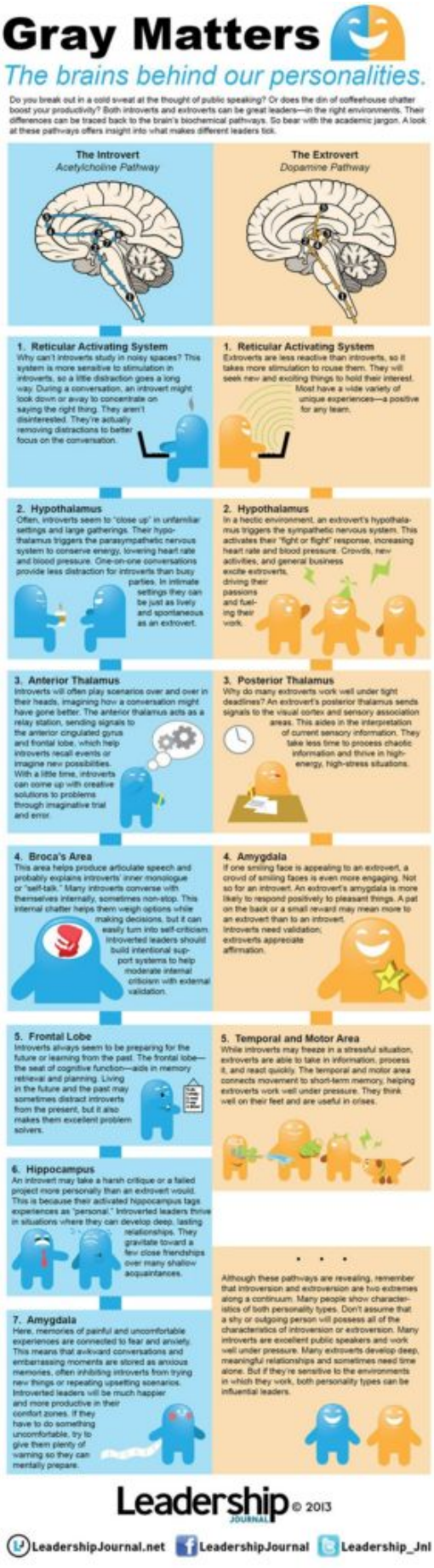
I tried to apply some of what I'd learned from the Generative Programming book to implement simple GA-based algorithms, which I call ***G***eometric ***Mac***ros, through code generation. A geometric macro is a set of standard GA multivector expressions assigned to local variables inside the macro. There are no control flow statements inside this simple computational concept. All the computational power mainly comes from the mathematics of GA itself. I used the GMac prototype to implement parts of a ray tracer that I implemented based on Kevin Suffern's nice book "Ray Tracing from the Ground Up", in addition to some other small computational problems.

Using Automatic Programming and Code Generation is a fundamental design decision I made for GMac. My experience with OO design can be expected to extend to all other programming paradigms. There can be no universal design methodology suitable for all applications. Even more, in a single large application, we might need several paradigms to be used [5. This occasional necessity affected the design of some powerful modern multi-paradigm languages like F# and Scala, for example.]. The only common activity between all paradigms is the creative composition of structured textual code that is both human-readable and machine-compilable. Instead of trying to force some kind of fixed design method upon GA-based

application developers, we can just provide a set of powerful tools to help them in composing their application in whatever programming paradigm, software system design, or programming language they choose. I still believe this is the best, most effective, and most general way to use GA in software development.

Swimming Upstream

> I had to resign from the University to continue my intellectual research project. -- Professor Abdel-Wahab Elmessiri [6. Professor Abdel-Wahab Elmessiri is a famous Egyptian thinker, one of the critics of western modernity, and a pro of founding an 'Islamic' version of modernism, author of the Encyclopedia "Jews, Judaism and Zionism".]



I'm an Introvert. The brains of Introverts and Extroverts work

differently.

My Ph.D. part of the journey was completed in April 2010 when I defended my thesis. It was an exhausting part of my life with many difficulties resulting from working in an anti-scientific, anti-creative environment by design; especially for introverted people like me [7. In 2014, I read Susan Cain's fascinating book "Quiet: The Power of Introverts in a World That Can't Stop Talking". It profoundly changed how I perceive myself and others. You can get an idea about the importance of her ideas for all humans by watching her TED talk. Also, this nice article illustrates how introverts' minds work internally. I recommend following her ideas as they are important for any activities requiring human interactions, especially teaching and learning.]. In this difficult environment, I met many talented, highly creative, and mostly self-educated people. These people motivated their colleagues and students to make the best research possible and sincerely promoted good education. I'm very grateful for their presence in my life. The system itself is designed to kill the hope in such talented people or drive them out into more scientifically and humanly attractive environments [8. The direct cause of this can be found in this article, where epidemiologist Richard Wilkinson discusses new research showing that, among developed countries, the healthiest and happiest aren't those with the highest incomes but those with the most equality.]. As Barry Schwartz [9. Barry Schwartz studies the link between economics and psychology, offering startling insights into modern life. He is the author of the book "Why We Work" providing an eye-opening, groundbreaking tour of the purpose of work in our lives.] describes it in his beautiful TED talk:

> It is not true that you "just can't get good help anymore." It is true that you "can't get good help anymore" when you give people work to do that is demeaning and soulless. And interestingly enough, Adam Smith — the same guy who gave us this incredible invention of mass production, and division of labor — understood this. He said, of people who worked in assembly lines, of men who worked in assembly lines, he says: "He generally becomes as stupid as it is possible for a human being to become." Now, notice the word here is "become." "He generally becomes as stupid as it is possible for a human being to become." Whether he intended it or not, what Adam Smith was telling us there, is that the very shape of the institution within which people work creates people who are fitted to the demands of that institution and deprives people of the opportunity to derive the kinds of satisfactions from their work that we take for granted.

You can get some idea about the results of this sad situation by reading this 2011 article by Hillel Ofek: "Why the Arabic World Turned Away From Science". I can only pray and hope this situation changes in the near future.

In 2010 I made a mistake trying to publish my Ph.D. work as an article in a high-profile journal not concerned about Geometric Algebra. They rejected the article for good reasons, mostly not related to GA but to the article itself. The editor gave me a piece of advice going around Geometric Algebra having nothing new to offer in terms of applications in computer science. He emphasized that more effort must be made by the GA community to overcome this situation by finding new insights that only GA can provide. I realized then that choosing to do research using GA is a lonely path to take. Recently, I made some interviews with leading researchers in the GA community who stated this experience in similar words. The situation has gotten much better in the last couple of years where GA-related research started to appear at increasing rates in many fields in engineering and computer science.

After completing my Ph.D. back in 2010 I knew I had to design a new version of GMac to realize my dream vessel for exploring the computational universe with Geometric Algebra. I knew it would be a difficult journey to make. As Walter Bright puts it in his article "So You Want To Write Your Own Language?":

> First off, you're in for a lot of work…years of work…most of which will be wandering in the desert. The odds of success are heavily stacked against you. If you are not strongly self-motivated to do this, it isn't going to happen. If you need validation and encouragement from others, it isn't going to happen.

Nevertheless, I was certain that GA will be accepted in the near future as a major language for mathematical modeling in computer science and engineering. I was also certain that I will gain much experience and feel many joys developing the next generation of GMac. I could only work on my project alone against the surrounding environment for two to four months per year because of my many family obligations and non-research work duties. These available months weren't even consecutive; they were spread along the year with no apparent order. During all these years whenever I got an opportunity to make research I faced a difficult question. Should I focus on writing papers in computer science topics not related to Geometric Algebra to be promoted? Or should I invest the little time I had in developing my dream vessel? I always chose the second.

After I finally got promoted to Assistant Professor [10. You can read more about academic ranks in Egyptian universities in here.] in July 2010 and for another full year up to Summer 2011, I had to find additional jobs to support my family, so I had to delay developing the next generation of GMac until August 2011.

In the final part of this series, I will talk about the next phase of my journey developing GMac's next generation. I will explain its main components and how I arrived at their design.

تمت بحمد الله الجمعة ١٣ جماد أول ١٤٣٨هـ
