Demo: A Functional EDSL for Mathematics Visualization That Compiles to JavaScript

Allister Beharry
ab796@student.london.ac.uk
University of London
United Kingdom

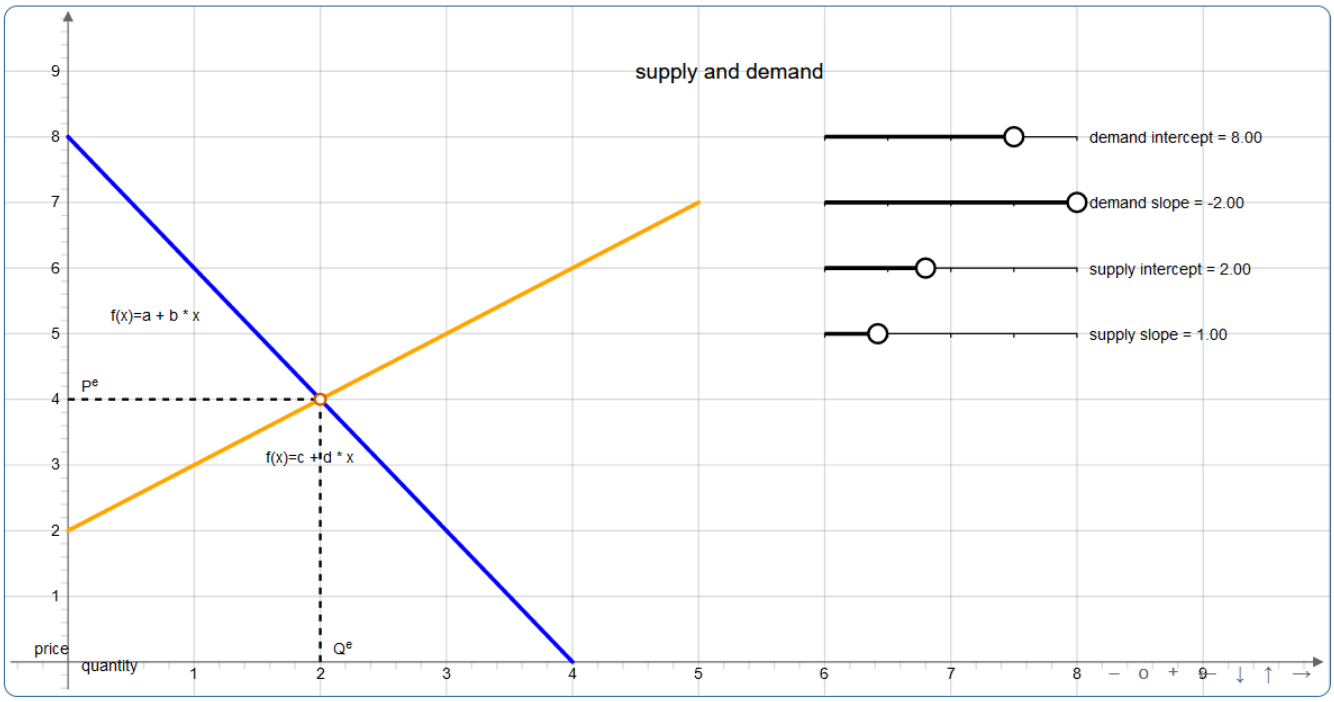

Figure 1. Visualization of basic supply and demand functions in economics

Abstract

Visualizations are a critical part of mathematics practice and education, and computers and open-source web technologies provide accessible ways to create high-quality mathematics visualizations at virtually no cost. However libraries and languages to create visualizations for mathematics are typically fine-grained, low-level, and targeted to vector graphics domain experts or web developers, not mathematics students or teachers or end-users. We present demos of Sylvester: a functional domain-specific language interface to the JSXGraph visualization library embedded in F# that emphasizes readability, composability, and the ability of end-users to easily create and manipulate elements of high-quality interactive mathematics visualizations without needing vector graphics or web development domain knowledge.

CCS Concepts: • **Human-centered computing** → **Scientific visualization**; • **Computing methodologies** → *Computer graphics*; • **Software and its engineering** → **Domain specific languages**.

Keywords: dsl, f#, mathematics, visualization, javascript

ACM Reference Format:
Allister Beharry. 2023. Demo: A Functional EDSL for Mathematics Visualization That Compiles to JavaScript. In *Proceedings of the 11th ACM SIGPLAN International Workshop on Functional Art, Music,*

FARM '23, September 8, 2023, Seattle, WA, USA

ACM ISBN 979-8-4007-0295-2/23/09…$15.00
https://doi.org/10.1145/3609023.3609808

Modelling, and Design (FARM '23), September 8, 2023, Seattle, WA, USA. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3609023.3609808

1 Introduction

1.1 Motivation

There are several ways to create computer-based mathematics visualizations today including interactive point-and-click applications like GeoGebra[7]. However programming languages provide a powerful abstraction that allows the user to instantly move between different levels of detail and aggregation in a visualization, from zooming in to modify the specifics of one element or attribute of the visualization, to grouping several related elements and modifying them together, to zooming out to modify the high-level composition and design and properties of the entire visualization. Following Hudak[4] if the ideal abstraction for a particular computer application is a programming language designed for that application, we'd expect domain-specific programming languages to offer a more effective and usable interface to the complex tasks of creating computer-based visualizations for mathematics.

The target audience for these visualization programming languages should be kept in mind however. Existing programming languages and libraries like PGF/TikZ[8] and MetaPost[3] are powerful tools for creating sophisticated vector-graphics based mathematical visualizations, but inherit syntactic and semantic language features which may make them hard to understand and use by people who are not experienced programmers. For instance drawing a line in TikZ looks like:

Listing 1. Example TikZ code

```
\coordinate [label=left:$A$] (A) at
    (0,0);
\coordinate [label=right:$B$] (B) at
    (1.25,0.25);
\draw (A) -- (B);
```

The syntax of these languages can be intimidating and the possibilities for composite data types which makes it easier for end-users to simplify and reuse both their own code and existing code in libraries can be limited. The contrast between the power and level of detail afforded by programming languages for diagramming and their difficulty of use has been observed e.g. [5, pg. 5]. In addition, interactivity is an important component of mathematics visualizations which can enhance both diagramming and learning processes as it gives the diagram programmer a constrained way to dynamically control and adapt their diagrams to different parameters without having to change the code.

JSXGraph [9] is a client-side JavaScript library for constructing advanced vector graphics visualizations in a web browser. JSXGraph has evolved from a library for displaying geometric constructions in different file formats to a powerful and comprehensive set of primitives for creating interactive visualizations in the browser over several areas of mathematics from calculus to statistics, using Scalable Vector Graphics (SVG). The JSXGraph library contains more than 60 high-level elements of mathematics visualizations from primitives and simple shapes like angles, lines, polygons, to much more complex constructions like Riemann integration diagrams. JSXGraph has certainly raised the bar for beautiful and detailed mathematics visualizations that are open-source and accessible to anyone.

But JSXGraph is a JavaScript library targeted at web developers and designed to be embedded in web applications, not directly used by mathematics educators or students. JavaScript is a language that inherits from the C syntax family and was intended to be used inside editors and IDEs to build applications, not used interactively or in an ad-hoc way.

1.2 About

Sylvester[1] is an F# embedded DSL for mathematical computing that supports an integrated approach to applications like computer algebra, theorem proving, and visualization. Sylvester contains a functional interface to the JSXGraph library that is intended to expose the power of JSXGraph to end-users for interactive iterative development of mathematical visualizations in web-based interactive environments for mathematical computing like Jupyter notebooks.

The visualization component of the Sylvester language tries to provide the ideal abstraction level for a programming language tool for mathematics visualization and diagramming. It supports high-level manipulation of vector graphics primitives without burdening the user with unfamiliar syntax and providing all the power and ease-of-use of a modern functional programming language.

2 Examples

2.1 Basics

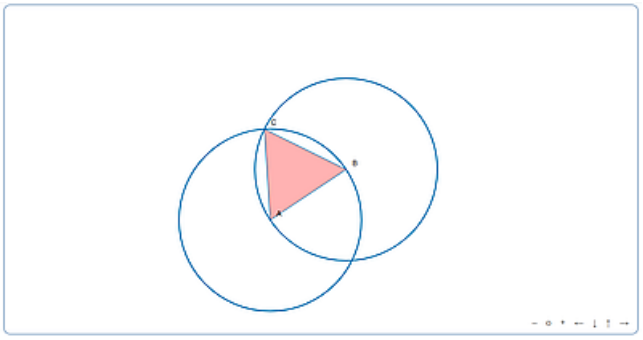

Figure 2. Visualization of construction of an equilateral triangle using 2 points in Sylvester.

The best way to introduce visualization in Sylvester is to look at some examples. The code in listing 2 creates the simple geometric visualization above:

Listing 2. Visualization of construction of an equilateral triangle using two points

```
let layout = {|
    boundingbox = area 6. 6. 0.
    showNavigation = true
    showCopyright = false
    keepAspectRatio = true
|}
let board = board layout
let A = point -2. -2. noface board
let B = point 1. 0. noface board
let C1 = circle A B defaults board
let C2 = circle B A defaults board
let C = intersection C1 C2 1 {|size =
    0 |} board
let ABC = polygon [|A; B; C|] defaults
    board |> withFillColor red
board
```

Some characteristics of the Sylvester visualization language syntax should be apparent from this short example, especially in contrast to TikZ and JavaScript code:

- The code is mostly plain English with a minimum of block, statement, and variable delimiters.
- Diagrams consist of a single board on which are drawn primitives or elements like points, lines, and circles.
- Element properties can be set through attributes on construction, like setting the size of the circles' intersection points to 0.
- Elements are drawn and properties set using regular F# functions and function results can be piped backwards and forwards as normal e.g. creating a polygon and then piping it to a function that sets the fill color to red.
- Element dependencies are specified by simply using variables pointing to reference elements e.g. the circle `C1` is drawn using the point `A` as its radius and the point `B` as a circumference point.

2.2 Economics

Economics is a mathematical discipline that is naturally oriented around functions of one or more real variables and consequently relies heavily on graphs and other mathematical visualizations for teaching and exposition of concepts, and for modelling and interpretations of empirical data. Mathematical computing environments like Wolfram Mathematica [10] and and Maple[6] and wxMaxima[11] that support visualizations have therefore been heavily utilized for exposition and visualization of both economic concepts and models of economic data, using the DSLs for mathematics visualizations provided by these environments.

The figure below is a simple economics visualization [2] created using Wolfram Mathematica.

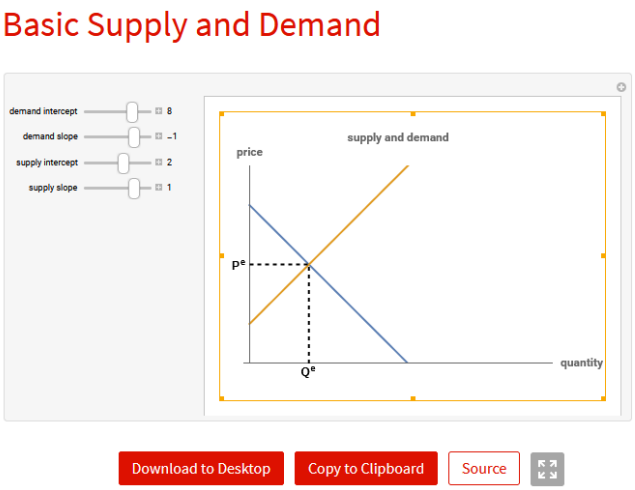

Figure 3. Wolfram Demonstrations Project - Basic Supply and Demand.

For the major part of our demo we'll see how the different aspects of the language are used to create the different parts of the economics visualization on the first page and compare to the Mathematica code. Sylvester allows us to create this visualization with a lot less verbosity and complexity and with more features and reusability, compared to using JavaScript or Mathematica.

Listing 3. Visualization of basic supply and demand functions in Sylvester

```
let grid = {|
    boundingbox = bbox -0.5 10. 10.
        -0.5
    showNavigation = true
    showCopyright = false
    keepAspectRatio = false
    axis = true
|}

let xaxis = {|
    name = "price"
    withLabel = true
    offset = [|10; 10|]
|}

let yaxis = {|
    name = "quantity"
    withLabel = true
```

```
    offset = [|10; 10|]
|}

let graph = {|
    strokeWidth = 3
    withLabel = true
    label = autoPosition
|}

let normal = {|
    size = 0
    dash = 2
|}

let board = board grid

setAttrs board.defaultAxes.x xaxis
setAttrs board.defaultAxes.y yaxis

let sliderx, sliderw = 6., 2.
let a = slider sliderx 8. sliderw 2.
    10. 8. {|name = "demand intercept"
    |} board
let b = slider sliderx 7. sliderw -4.
    -2.0 -1. {|name = "demand slope" |}
     board
let c = slider sliderx 6. sliderw 0.
    5. 2. {|name = "supply intercept"
    |} board
let d = slider sliderx 5. sliderw 0.2
    4. 1. {|name = "supply slope" |}
    board

let demand x = a.Value() + b.Value() *
     x
let supply x = c.Value() + d.Value() *
     x

let dg =
    functiongraph demand 0. 4. graph
        board
    |> withName "f(x)=a + b * x"
    |> withStrokeColor blue

let sg =
    functiongraph supply 0. 5. graph
        board
    |> withName "f(x)=c + d * x"
    |> withStrokeColor orange

let eq = intersection dg sg 0 nolabel
    board |> withFillColor white
let ex = perp_segment board.
    defaultAxes.x eq normal board
let ey =  perp_segment board.
    defaultAxes.y eq normal board

draw board [|
    ge.intersection ey board.
        defaultAxes.y 0 {|name = "P^e";
         size = 0|}
    ge.intersection ex board.
        defaultAxes.x 0 {|name = "Q^e";
         size = 0|}
    ge.text 4.5 9. "supply and demand"
         {|fontSize=16|}
|]
```

References

[1] Allister Beharry. 2023. *Sylvester: Unified, typed, notation for symbolic mathematics and proofs (short talk) (ML 2021) - ICFP 2021.* https://icfp21.sigplan.org/details/mlfamilyworkshop-2021-papers/6/Sylvester-Unified-typed-notation-for-symbolic-mathematics-and-proofs-short-talk- [Online; accessed 9. Jun. 2021].

[2] Mark Gillis. 2011. *Basic Supply and Demand.* https://demonstrations.wolfram.com/BasicSupplyAndDemand/ [Online; accessed 18. Jul. 2023].

[3] John Hobby. 2014. *MetaPost.* https://www.tug.org/docs/metapost/mpman.pdf

[4] Paul Hudak. 1996. Building Domain-Specific Embedded Languages. *ACM Comput. Surv.* 28, 4es (dec 1996), 196–es. https://doi.org/10.1145/242224.242477

[5] Dor Ma'ayan, Wode Ni, Katherine Ye, Chinmay Kulkarni, and Joshua Sunshine. 2020. How Domain Experts Create Conceptual Diagrams and Implications for Tool Design. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–14. https://doi.org/10.1145/3313831.3376253

[6] Maplesoft, a division of Waterloo Maple Inc.. [n.d.]. *Maple.* Waterloo, Ontario. https://www.maplesoft.com/products/Maple/

[7] Markus Hohenwarter, GeoGebra Team. 2001. *GeoGebra.* https://www.geogebra.org/

[8] Till Tantau. 2023. *PGF/TikZ.* https://pgf-tikz.github.io/pgf/pgfmanual.pdf

[9] Bianca Valentin and Michael Gerhäuser. 2009. *Interactive SVG with JSXGraph.* Retrieved June 7, 2023 from https://jsxgraph.uni-bayreuth.de/talks/svgopen09/jsxgraph.pdf

[10] Wolfram Research, Inc. 2023. *Mathematica.* https://www.wolfram.com/mathematica

[11] wxMaxima authors. 2023. *wxMaxima.* https://wxmaxima-developers.github.io/wxmaxima/wxmaxima.pdf

Received 2023-06-01; accepted 2023-07-01