October 2020

Automatic Simulation and Display of Functions Within Developer Tools

Collin Irwin

Rachel Hausmann

Automatic Simulation and Display of Functions Within Developer Tools

ABSTRACT

Mathematical functions are frequently implemented in a variety of code, e.g., video games, web publishing (CSS/HTML), scientific programming, etc. Developers of such code often benefit from visualizations of the output of in-code functions for various inputs. At present, to obtain visualizations, developers need to utilize alternative tools, e.g., graphing or mathematical software/websites. Using such alternative tools outside the code-development environment breaks the development flow. This disclosure describes techniques that find functions within code that a developer is currently viewing and automatically simulate and display a visualization of the results. Such instant visualization of functions present in code enables developers to ensure that the function and its parameters, as entered by the developers, are indeed accurate and as intended. The introduction of bugs is forestalled, and developer time is optimized.

KEYWORDS

- Numeric simulation
- Mathematical function
- Integrated development environment (IDE)
- Function visualization
- Developer tool

BACKGROUND

Mathematical functions are frequently implemented in a variety of code, e.g., video games, web publishing (CSS/HTML), scientific programming, etc. Developers of such code often benefit from visualizations of the output of in-code functions for various inputs. At present,

to obtain visualizations, developers need to utilize alternative tools, e.g., graphing or mathematical software/websites. Using such alternative tools outside the code-development environment breaks the development flow. The lack of immediate visualization of mathematical functions within a code-development environment can lead to inadvertent introduction of bugs in the code.

Some code-development environments automatically display the color that appears in code as a small thumbnail. For example, if the code-phrase `rgb(255, 0, 0)`, representing a fully red color, appears in a CSS file, a small thumbnail of the color is provided adjacent to the code-phrase. However, such color visualizations are restricted to functions that control the display and are not generally available for numeric and non-numeric functions.

DESCRIPTION

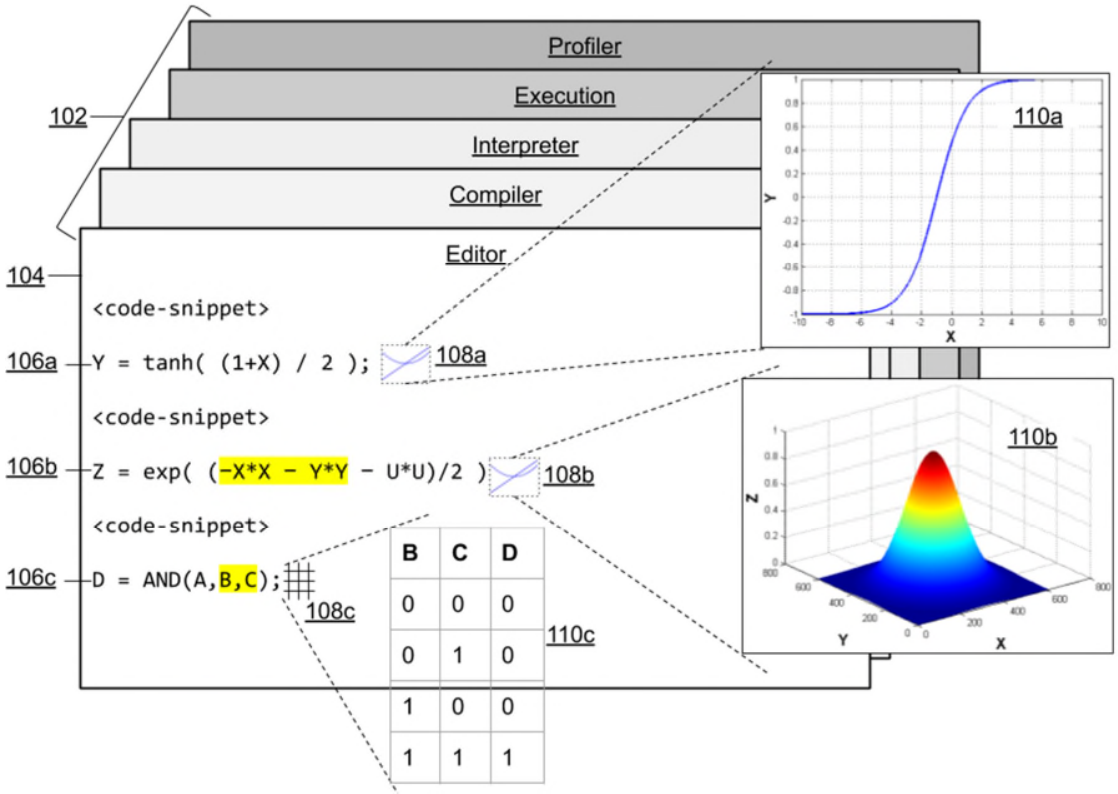

Fig. 1: Automatic simulation and display of functions within developer tools

Fig. 1 illustrates automatic simulation and display of functions within developer tools, per the techniques of this disclosure. An integrated development environment (IDE, 102) includes a coordinated collection of developer tools (toolchain), e.g., an editor, a compiler, an interpreter, an execution window, a profiler, a static analyzer, etc., also referred to collectively as execution infrastructure. The editor (104), or customizable text editor, and associated utilities such as code-search utility, etc., are referred to as the front-end tool.

Within the IDE, e.g., in the editor (104), a developer writes code that includes a mathematical function. In the example of Fig. 1, the code includes three mathematical functions (106a-c). The functions can have one or more inputs and one or more outputs. The inputs and the outputs of the functions can each be discrete or continuous variables.

Per the techniques of this disclosure, the presence of a mathematical function is recognized and the function is automatically graphed. For example, inclusion of the function 106a, which has one continuous input and one continuous output, results in the automatic generation of the graph 110a. The function 106b has three continuous inputs and one continuous output. The developer selects two of the three inputs, e.g., by highlighting the inputs or by selecting them using sliders, selection boxes, or another input mechanism. A three-dimensional graph (110b) representing the function 106b is auto-generated. The function 106c has three discrete inputs and one discrete output. A table of function outputs versus developer-selected function inputs is auto-generated (110c). Upon detection of a mathematical formula, an icon (108a-c) can optionally be placed next to it, such that the graph is displayed upon selection of the icon by the developer.

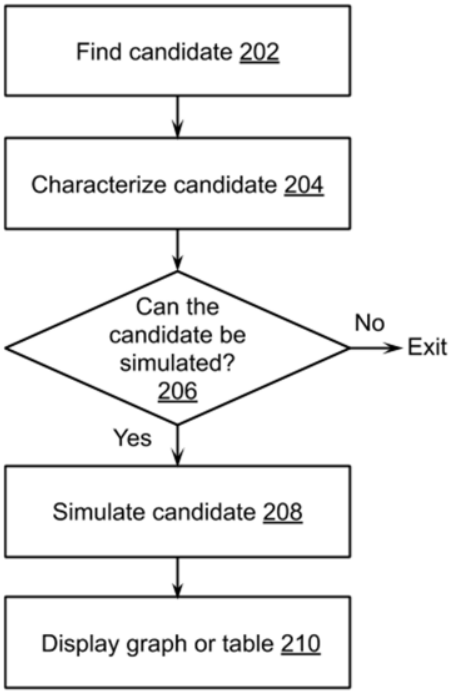

Fig. 2: Finding functions within an editor amenable to simulation and graphing them

Fig. 2 illustrates an example process to find functions within an editor (or front-end) amenable to simulation and graphing the identified functions.

Finding a candidate function (202)

Per the techniques, the content of the code presently being displayed in the front-end is analyzed to identify candidate computational functions or methods. Such analysis makes use of the execution infrastructure.

Characterizing a candidate function (204)

Once a candidate function is found, it is characterized by attributes such as:

- Input types, e.g., numbers, strings, arrays, objects, enums, booleans, etc.
- Output types, e.g., numbers, strings, arrays, objects, enums, booleans, etc.

- Whether the method relies on a non-input state, e.g. whether it is non-static, or a class method, or whether it relies on global state, etc.
- The types of the non-input states, if any.

The characterization of the candidate function is based on a type system, either intrinsic to the programming language (such as with C++, TypeScript, Java, etc.) or integrated with execution infrastructure independent of the programming language. Example types include numbers, strings, arrays, objects, enums, booleans, etc.

Determining if the candidate function is amenable to simulation (206)

Once a candidate function is characterized, a determination is made as to whether it is amenable to simulation. Functions with few inputs, e.g., three-to-four inputs, and zero or few non-input states are amenable to simulation. Continuous functions with greater than three inputs are generally not visualizable, as the human eye can perceive at most three dimensions. Therefore functions with four or greater inputs, or with large non-input dependencies, can be disqualified from simulation. However, if the developer selects three inputs from a function that has more than three inputs, then it becomes amenable to simulation.

Functions with inputs or dependencies that have smaller type spaces are more amenable to simulation. If the type of a dependency cannot be logically narrowed by the execution infrastructure to an enumerated type or a number, e.g., if the type is determined to be an unbounded string, an arbitrary object, arbitrarily-sized array, etc., then the function is disqualified from simulation. In general, functions with numeric inputs and outputs are readily visualizable and offer the most benefit to the developer when simulated, and are hence determined as amenable to simulation.

Simulating the function (208)

Candidate functions that are determined to be amenable to simulation are evaluated over its input space, or a subset thereof, and state space. The resulting input-output pairs are stored. Choices are made with regards to the length of the simulation and the sub-space within the space of inputs and states to be simulated based on the context of the function.

For example, if the input and state spaces are relatively small enough and individual simulations are relatively fast, then the simulation can be exhaustive, such as with 3 boolean inputs resulting in 2×2×2 = 8 potential outputs. If the input is known to exist within a finite range, then only that range is simulated. That is, if it is known that the input exists only within the range [0, 1], or is clamped within that range by the function body, then inputs before 0 and after 1 are not simulated. The fidelity (resolution) of the simulation can be decreased in areas of function input space where little change is occurring. The simulation can initially be performed at a coarse level and the granularity can be increased automatically if the developer views the output preview/visualization.

Visualizing the function (210)

Upon completion of the simulation, the input-output pairs are visually displayed within the front-end. This can be within a small pop-up containing a graph, or as a chart in the console (as shown in Fig. 1), or some other visual or non-visual representation. As explained earlier, the pop-up creation may not be immediate; rather, the developer may have to click on an icon or some other portion of the front-end user-interface to initiate the pop-up. Further, the simulation can be resumed at the point of such selection by the developer to improve the resolution of the graph or chart.

The developer can also request more data by zooming in on a section of the chart, zooming in to view more of the chart, or panning the graph/chart to previously unsimulated or unviewed input ranges, all of which cause the production of data points in the requested regions. For charts with three or more inputs, the third, fourth, and further inputs can appear within sliders, selection boxes, or other input mechanisms while the graph itself remains three-dimensional.

Example use-cases

The techniques are applicable to the development of video or virtual reality (VR) games/software, as these types of software make extensive use of mathematical functions relating to the position, rotation, translation, and perspective of objects. Video and VR software also make extensive use of physics formulas, e.g., to simulate collisions, movement in a gravitational field, reflection of light off objects, etc. The techniques are applicable to physics formulas relating to gravity, attraction, springs; mathematical functions such square-root, minimum, maximum, clamp; trigonometric formulas for sine, cosine, tan, arc-tangent; Bezier curves; etc. The techniques are also applicable to webpage creation software, e.g., CSS, HTML, or other markup/styling files, as such code often contains various functions in in-line JavaScript or in styling, e.g., ease-in, ease-out, ease-in-out, etc.

The output of the simulation can be non-visual. Visually-impaired users in particular can benefit from alternative output forms. For example, the output can be physical rather than digital, e.g., it can be fed to a Braille device. As another example, the output can be auditory, e.g., numeric graphs can be mapped to sound frequencies. Non-numeric outputs can be read aloud with text-to-speech tooling.

To prevent security vulnerabilities, simulations can be sandboxed. Network traffic, disk-IO, etc. can be either entirely blocked and disabled, or require explicit approval from the developer.

In this manner, the techniques of this disclosure perform a within-IDE analysis of functions present in code to automatically simulate a function under a variety of inputs and conditions, and output the result of the simulation to the developer to guide the development of code. The analysis and simulation is applicable to numeric functions and other types of functions.

CONCLUSION

This disclosure describes techniques that find functions within code that a developer is currently viewing and automatically simulate and display a visualization of the results. Such instant visualization of functions present in code enables developers to ensure that the function and its parameters, as entered by the developers, are indeed accurate and as intended. The introduction of bugs is forestalled, and developer time is optimized.