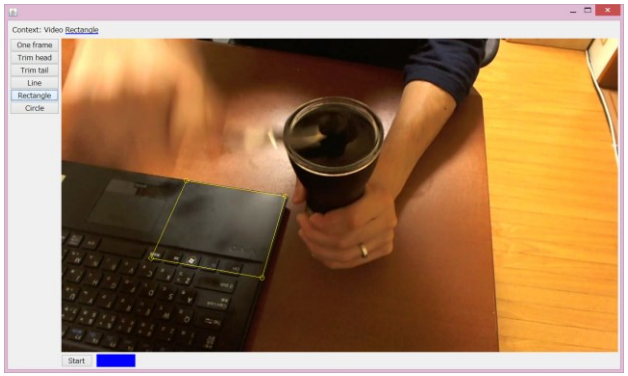


Figure 1.
Overview of *Visionsketch* IDE

Visionsketch: Gesture-based Language for End-user Computer Vision Programming

Jun Kato <jun.kato@acm.org>

Affiliation: The University of Tokyo
ACM Member No.: 7960513
Address: Science bldg. 7, Room 302, The University of Tokyo, 7-3-1 Hongo, Bunkyo, Tokyo, 113-0033, Japan
Supervisor: Prof. Takeo Igarashi
PLDI 2013 SRC category: Graduate

Problem and Motivation

An interactive touch surface is getting more and more popular as the primary input source for computers, including smartphones and tablet devices. Such devices rarely come with a set of a mouse and keyboard, making it difficult for the programmer to write code in a traditional text-based development environment. Meanwhile, there is an increasing demand on the use of complex data types e.g. images, reflecting the increase in computational power. Current programming languages and development environments usually do not have built-in support for such complex data types, resulting in poor programming experience. In this paper, we propose Visionsketch, a domain-specific language for computer vision programming along with its gesture-based development environment optimized for a touchscreen.

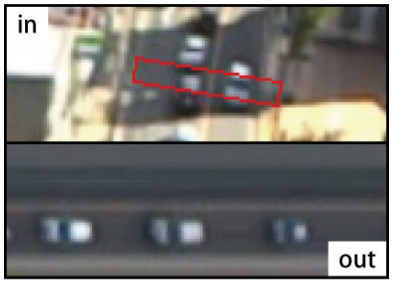
Background and Related Work

TouchDevelop [1] enbraces the idea of using a touchscreen as the primary input device to author code. It introduces a new language and a context-dependent software keyboard with large keytops. This is straight-forward evolution of the text-based integrated development environment (IDE). Such text-based environment has difficulty in visualizing complex data types. To address this issue, Active code completion [2] proposes type-dependent interfaces to specify parameters for the API (e.g. color palette for choosing desired color). Sikuli IDE [3] provides a special text editor that is capable of visualizing image data in the code, which is passed as an argument of the built-in API. While these IDEs still provide text-based editors, our development environment puts more focus on image data visualization and direct manipulation. It actually does not need any text input.

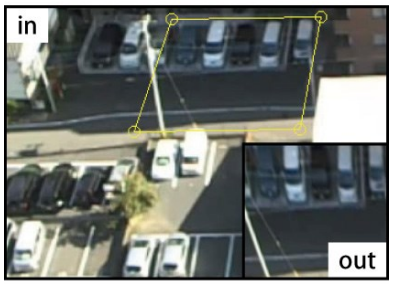
Approach and Uniqueness

General programming languages are text-based, where method names and arguments are represented by text. On the other hand, our programming language deals with computer vision algorithms, which take an image or video (time series images) as input. Such data cannot nicely be represented by text. Optional arguments usually have visual meaning, such as four

Linear-polar conversion
in: rectangle in video or image
out: video or image



Time-lapse conversion
in: narrow line in video
out: time-lapse image



Perspective transform
in: rectangle in video or image
out: video or image

Point objects denoting a rectangular area in the image. Output from the algorithms are also images, videos, or a group of regions in the image. To better reflect the visual nature of this style of coding, we propose Visionsketch, a "visual" programming language where each code element is represented by an image or video rather than text. Please note that it is different from traditional visual programming languages whose program structure is visual but data is still referenced by text, including file names and constants.

In our development environment, the programmer first needs an image or video to start programming. He can also use live video data from a web camera. Then, he does some gestures on the image or video to choose possible operations and specify parameters in the context instead of typing method names and parameters. Once the details are fixed, the result image or video from the operation is immediately visualized. He can continue drawing gestures on the result to author another code element.

Examples of the supported operations in the current prototype implementation are listed in the left figure. All features are implemented as Java classes and can be easily extended by the Java programmer.

Results and Contributions

We created two example applications to see the effectiveness of the Visionsketch language.

Use case 1 (one command): On a given table, I would like to know the number of coins without having to count them by myself. I just use my tablet device to take a photo of the table, draw a circle containing one coin. Then, other coins are highlighted in the photo with the total number shown on the right side of the window.

Use case 2 (sequential commands): I usually grind coffee beans, drink a cup of espresso, and start my work. I do not know the right amount of coffee powder for one cup, but I think I can estimate it by counting how many times I rotate the grinder's handle. To count the number of grinds, first, I record a video of my hand and draw a line on the video where my hand crosses once per one rotation. Next, a time-lapse image appears on the right of the video. I open it and highlight some timings when my hand crosses the line. Then, other crossing points are also highlighted in the image with the total number.

We asked two test users to implement these applications. While one user was familiar with basic programming concepts, the other did not know about programming except for HTML.

In both cases, they succeeded in the implementation in an hour, after fifteen minutes of introduction to the basic usage of the development environment. The programmer pointed out the similarity of the Visionsketch environment and Instagram, a photo editing tool. Indeed, we think the difference between a programming environment and an end-user interface is in its level of freedom. Designing user interface can be even thought of as designing a domain-specific programming language. The non-programmer appreciated that she did not need to remember any difficult words but just choose and test multiple tools to find the proper solution. This preliminary result highlights the benefit of direct manipulation with intuitive gestures.

We are currently preparing the open-source distribution of Visionsketch. Our future work includes support for "if" statement which might be able to borrow some ideas from a decision table of Subtext [4] or conditionals of Substroke [5]. They both investigate the possibility of rendering a programming language in the two-dimensional space.

References

[1] Nikolai Tillmann, Michal Moskal, Jonathan de Halleux, and Manuel Fahndrich. 2011. TouchDevelop: programming cloud-connected mobile devices via touchscreen. In *Proc. of ONWARD '11*. ACM, New York, NY, USA, 49-60.

[2] Cyrus Omar, YoungSeok Yoon, Thomas D. LaToza, Brad A. Myers. 2012. Active code completion. In *Proc. of ICSE '12*. IEEE Press, Piscataway, NJ, USA, 859-869.

[3] Tom Yeh, Tsung-Hsiang Chang, and Robert C. Miller. 2009. Sikuli: using GUI screenshots for search and automation. In *Proc. of UIST '09*. ACM, New York, NY, USA, 183-192.

[4] Jonathan Edwards. 2005. Subtext: uncovering the simplicity of programming. In *Proc. of OOPSLA '05*. ACM, New York, NY, USA, 505-518.

[5] Bret Victor. 2007. Substroke Design Dump. *http://worrydream.com/substroke/*