Fast Interactive Functional Computer Vision with Racket

A Demonstration using Racket and the Kinect Sensor

Benjamin Seppke
University of Hamburg
Dept. Informatics
Vogt-Kölln-Str. 30
22527 Hamburg, Germany
seppke@informatik.uni-hamburg.de

Leonie Dreschler-Fischer
University of Hamburg
Dept. Informatics
Vogt-Kölln-Str. 30
22527 Hamburg, Germany
dreschler@informatik.uni-hamburg.de

ABSTRACT

Functional programming languages, like Lisp or Racket are known to be general purpose languages with a steep learning curve and wide range of applications. They can be used interactively to solve problems and have inspired other comparably new languages with respect to functional extensions (e.g. Python or Swift). In this work, we will demonstrate the use of the Racket programming language with respect to fast interactive computer vision. Based on the VIGRACKET module, which combines the best of the compiled and interactive worlds with respect to common tasks in computer vision, Racket and the VIGRA C++ library (see [4]), we will present the a (near-) realtime computer vision demo. For this demo we have selected the Microsoft Kinect Sensor as the continuous the image source. We present the connection to the sensor by means of image transfer to Racket and a case study: a natural pointer interface for human computer interaction.

Categories and Subject Descriptors

D.1.1 [**Programming Techniques**]: Applicative (Functional) Programming—*Allegro Common Lisp, SBCL, Racket*;
D.2.2 [**Software Engineering**]: Design Tools and Techniques—*modules and interfaces, software libraries*;
I.4.8 [**Image Processing and computer vision**]: Scene Analysis

General Terms

Functional Programming, Racket, Language Interoperability, Computer Vision, Image Processing, Human-Computer-Interaction, Realtime processing

1. INTRODUCTION

In [4] we have presented how well functional programming and computer vision approaches may be combined by means of the VIGRACKET library. Like other state-of-the-art interactive development environments, functional language interpreters natively offer an interactive development cycle, generic modeling and even powerful garbage collectors. So, instead of using a new language with functional extensions, like Swift, Python or others, why not simply use well-known functional languages like Lisp or Racket?

The main aim of this paper is to demonstrate the use of the extended VIGRACKET module with respect to the Kinect sensor as a data source. Since the VIGRACKET module has been developed to introduce computer vision to Racket, neither to support interactive realtime processing nor to support fast functional development interfaces or fast visualizations, some optimizations were additionally needed. For sake of clarity of this demo, these optimization steps can be found elsewhere (see [3]). Herein, we present how to connect with the Kinect sensor to Racket, acquire images and to use the acquired images by means of an interactive natural pointing device interface.

2. PRELIMINARIES

To run the demonstration, some preliminaries need to be fulfilled. Besides Racket, the VIGRACKET module needs to be downloaded and installed from the author's GitHub account: `https://github.com/bseppke`. Depending on the target's operating system, this module may require additional dependencies, which are described in [4].

Since the VIGRACKET module is not an acquisition library, it does not support real-time acquisition devices. For this demonstration, we have selected the Microsoft Kinect sensor as the acquisition device, and the OpenKinect libfreenect library for accessing the sensor. Unfortunately, this library does provide interaction layers for Python and Java, but none for Racket (see [5]). Based on a low level USB-interface (via the libusb), the libfreenect library allows access to all the necessary data. In order to avoid concurrent asynchronous calls and callbacks, we use the synchronous grabbing access API. Finally, it is necessary to download and install the "rackinect" Module from the same GitHub account as mentioned above.

3. CONNECTION TO THE KINECT

Since the image formats of the raw data, which are used by the libfreenect library, are not compatible with the image representation of the VIGRACKET module, we need to introduce another small Racket/C-wrapper, to which we refer to as "rackinect". On the Racket side of this wrapper we define grabbing functions for the acquisition of the raw data. The function `(grabdepth)` grabs the depth data (in mm) and stores it inside a newly allocated one channel VIGRACKET image. The function `(grabvideo)` grabs the current RGB image and stores it by means of a new three channel VIGRACKET image. Finally the `(grabdepth+video)` grabs both data and stores it by means of new a four channel VIGRACKET image, where the first channel contains the depth data (in mm) and the last most three channels contain the RGB data.

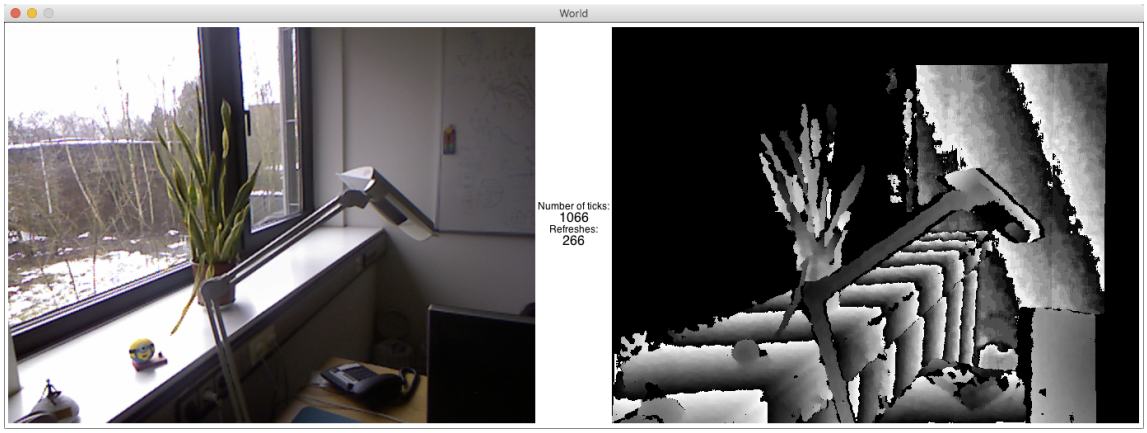

Figure 1: Image taken out of the sequence we get by calling (animate live-view-combined). Left: RGB image, center: current tick and refresh counter, right: depth image. Since the depth values are normalized to millimeters, there is one overflow each 25.5 cm. Black parts of the depth image denote unknown depth.

As a first demonstration, we present the use of both functions for a near real-time depth and video display GUI. Here we use the animate functionality of Racket's `2htdp/universe` module (see [1]). It takes a function with one argument and a Racket bitmap as a result type and calls the given function 28 times per second using an increasing argument `t` and updates the images every fourth frame. An example output is shown in Fig. 1.

```
(define dp (grabdepth+video))
(define d  (image->bitmap (list (car dp))))
(define p  (image->bitmap (cdr dp)))

(define (live-view-combined t [update_each 4])
  (when (= (modulo t update_each) 0)
     (begin
        (set! dp (grabdepth+video))
        (image->bitmap (list (car dp)) d)
        (image->bitmap (cdr dp)        p)))
  (beside p (status t update_each) d))

(animate liveview-combined)
```

4. NATURAL POINTER INTERFACE

The computer mouse has probably been the most common pointing device for human-computer interaction for decades. Due to the electronic mobile revolution with tablets and smartphones, we are now able to use our fingers directly as pointing devices, e.g. by means of (multi-) touch displays. Although this is closer to the "natural pointing" metaphor, it is still artificial and might not be well applicable for general virtual environments. The main limitation is the two-dimensional interface, since virtual worlds are usually not as flat as the displays' touch surfaces.

The Microsoft Kinect sensor with its depth image stream provides another alternative for a more natural pointer interface by tracking your fingers movements (cf. [2]). To simplify the finger detection, we make the following assumptions for this case study:

1. Only a certain range of the depth data is allowed to contain the image of the finger.
2. A finger pointer is defined as the top- and left-most point, which is found within this range.

Under these assumptions, it is quite clear, that we will detect one pointer position if any object is inside the depth interval of interest. Since it is the top- and left-most position, it is not necessary, that the finger is put in front of other objects, but above them. The first necessary step is to threshold the raw depth image accordingly to the range of the depth interval.

```
(define (closerThan depth_img [t 800])
  (image-map!
     (lambda (val) (if (< 0 val t) 255.0 0.0))
     depth_img))
```

In this function we are able to use the in-place method `image-map!` to save allocation costs, because the original depth values are no longer needed in the further processing. To find the top-left part of the thresholded depth image, we may apply the function above while the result is true. The first false coordinate denotes the pointer position. Experiments have shown, that this approach works stable at about 5 frames per second (see [3]).

5. REFERENCES

[1] M. Felleisen. *How to Design Programs: An Introduction to Programming and Computing*. MIT Press, 2001.

[2] P. Premaratne. *Human Computer Interaction Using Hand Gestures*. Cognitive Science and Technology. Springer Singapore, 2014.

[3] B. Seppke. Near-realtime computer vision with racket and the kinect sensor. Technical report, University of Hamburg, Dept. Informatics, 2016.

[4] B. Seppke and L. Dreschler-Fischer. Efficient applicative programming environments for computer vision applications: Integration and use of the vigra library in racket. In *Proceedings of the 8th European Lisp Symposium*, 2015.

[5] The OpenKinect team. The OpenKinect project. Retrieved February 19, 2016 from: `https://openkinect.org`.