How to turbocharge your productivity with Racket

Dominik Pantůček <dominik.pantucek@trustica.cz>

Trustica, Prague College

27. 2. 2020

Agenda

- Who am I
- Computer Vision in Racket
- JSON
- GUI
- FFI
- Futures
- Procrastination

- Dominik Pantůček
- MSc in Computing
- Lecturing in the BSc and MSc programmes:
  - Digital Forensics
  - Algorithms and Data Structures
  - Cryptography
- About cryptography ...
  - Since 1999
  - Elliptic Curves Cryptography (ECC) research
  - Applied cryptography
- About algorithms ...
  - software 3D renderers in the 90's

- OpenCV
  - No, thanks, but no.
- Acquiring images:
  - Class `bitmap%` – can load from common bitmap image file formats
  - On RaspberryPi: `raspistill` binary and load temporary image file
- Accessing image pixels:
  - Image width, height and ARGB data ...

```
(define w (send bitmap get-width))
(define h (send bitmap get-height))
(define argb-bytes (make-bytes (* w h 4)))
(define offset (* 4 (+ (* w y) x)))
(for/list ((i (in-range 1 4)))
  (bytes-ref argb-bytes (+ off i)))
```

  - Result: `(list red-value green-value blue-value)`

- Closures to the rescue, bitmap pixel reader:

```
(define (make-bitmap-pixel-reader bitmap)
  (define w (send bitmap get-width))
  (define h (send bitmap get-height))
  (define argb-bytes (make-bytes (* w h 4)))
  (send bitmap get-argb-pixels 0 0 w h argb-bytes)
  (λ (x y)
    (define offset (* 4 (+ (* w y) x)))
    (for/list ((i (in-range 1 4)))
      (bytes-ref argb-bytes (+ off i)))))
```

- CV algorithms we use operate in HSV colorspace

- RGB to HSV conversion:

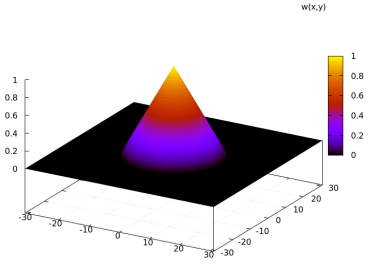
```
(define (rgb->hsv r g b)
  (define minimum (min r g b))
  (define v (max r g b))
  (define delta (- v minimum))
  (define s (/ delta v))
  (define h (* 60 (cond
        ((= r v) (/ (- g b) delta))
        ((= g v) (+ (/ (- b r) delta) 2))
        (else (+ (/ (- r g) delta) 4)))))
  (values h s v))
```

- Productivity boost:

```
(define-values (h s v)
  (apply rgb->hsv (get-pixel x y)))
```

- Although no support in Racket ...
- ... implemented quickly.
- Clean interface for others!

- Finding spotlights:
  - Tresholding pixels based on HSV `value`
  - Averaging pixel positions, HSV `value` is the weight
- Blurred color picker:
  - Weighted average of HSV components from given area
  - Weight decreases with the distance from center
  - $w_{x,y} = 1 - \frac{\sqrt{(x-x_0)^2+(y-y_0)^2}}{r}$
- Matching colors:
  - Is trivial in 2D HS part of HSV space



- Really easy to implement:

```
(define (find-spot-light get-pixel treshold x y w h)
  (define-values (total-weight total-x total-y)
    (for*/fold
        ((acc-weight 0) (acc-x 0) (acc-y 0))
        ((x (in-range x (+ x w)))
         (y (in-range y (+ y h))))
      (define-values (h s v)
        (apply rgb->hsv (get-pixel x y)))
      (if (> v treshold)
        (values (+ acc-weight v)
                (+ acc-x (* v x)) (+ acc-y (* v y)))
        (values acc-weight acc-x acc-y))))
  (values (exact->inexact (/ total-x total-weight))
          (exact->inexact (/ total-y total-height))))
```

- Affine transform in the form of $P' = T \times P$, where:
  - $P = [P_x, P_y]$ – point in original coordinates
  - $P' = [P'_x, P'_y]$ – point in transformed coordinates
  - $T = \begin{bmatrix} T_{(1,1)} & T_{(1,2)} & T_{(1,3)} \\ T_{(2,1)} & T_{(2,2)} & T_{(2,3)} \end{bmatrix}$ – transformation matrix
- M can be computed from two points A and B in original coordinates with known A' and B' in transformed coordinates
- Trivial in Racket

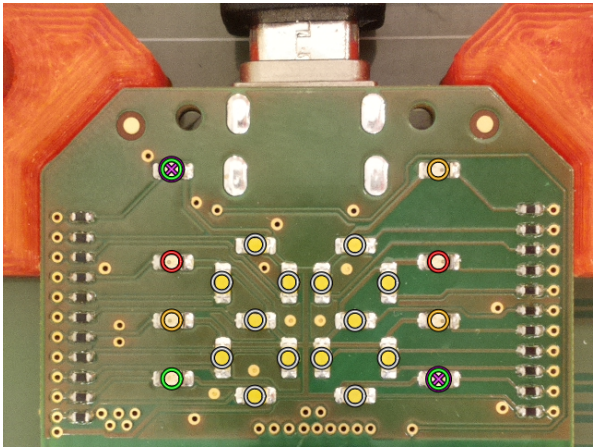
- In Racket:

```
(define (make-point-transformer A A/ B B/)
  (define-values (Dx Dy) (apply values (map - B A)))
  (define-values (D/x D/y) (apply values (map - B/ A/)))
  (define-values (Ax Ay) (apply values A))
  (define-values (A/x A/y) (apply values A/))
  (define D^2 (+ (* Dx Dx) (* Dy Dy)))
  (define t11 (/ (+ (* D/x Dx) (* D/y Dy)) D^2))
  (define t12 (/ (- (* D/x Dy) (* D/y Dx)) D^2))
  (define t21 (- t12))
  (define t13 (+ (- (* t11 Ax) (* t21 Ay) A/x)))
  (define t22 t11)
  (define t23 (- A/y (* t21 Ax) (* t22 Ay)))
  (λ (x y)
    (values (+ (* t11 x) (* t12 y) t13)
            (+ (* t21 x) (* t22 y) t23))))
```

h = 353.53
s = 0.46
v = 227.45
n = 0.43
r = 255
g = 0
b = 27
LED: red

h = 30.61
s = 0.66
v = 233.20
n = 0.34
r = 255
g = 130
b = 0
LED: orange

h = 142.03
s = 0.33
v = 231.30
n = 0.44
r = 0
g = 255
b = 94
LED: green



JSON

- The world is changed.
- Not everyone is using Racket.
- Javascript, PHP, Python, Ruby, QBasic ...
- JSON - Exporting results for other systems to process
- Usage:

```
(require json)
(with-output-to-file "output.json" #:exists 'replace
  (λ ()
    (displayln (jsexpr->string data))))
```

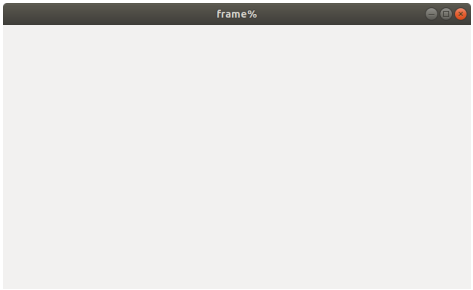
- Productivity boost: no work left for the programmer!

GUI in Racket

- Remember Turbo Vision?
- Widgets:
  - `frame%`
  - `vertical-pane%`
  - `horizontal-panel%`
  - `message%`
  - `button%`
  - `gauge%`

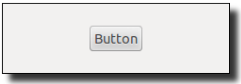
- Top-level window – a `frame%`:

```
#lang racket/gui
(define win
  (new frame%
       (label "frame%")
       (width 640)
       (height 360)))
(send win show #t)
```



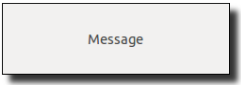
- Clickable button – a `button%`:

```
(define button
  (new button%
       (parent panel)
       (label "Button")))
```



- Informative label – a `message%`:

```
(define message
  (new message%
       (parent panel)
       (label "Message")))
```

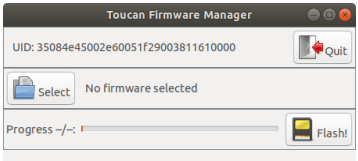


- Stack widgets vertically in `vertical-pane%`:

```
(define top-vbox
  (new vertical-pane%
       (parent this)))
```

- Stack widgets horizontally in `horizontal-pane%`:

```
(define toolbar
  (new horizontal-panel%
       (parent top-vbox)
       (border 1)
       (style '(border))
       (stretchable-height #f)))
```



- Productivity boost: Ever tried Python+Qt?
  - + QML
  - + CSS
  - + Javascript
  - ...

Cross-Platform FFI

- Mostly two libraries are needed:

```
(require ffi/unsafe
         ffi/unsafe/define)
```

- Loading dynamic library:

```
(define hidapi-lib
  (ffi-lib "libhidapi-libusb.so"
           #:fail (λ ()
                    (ffi-lib "libhidapi-0.dll"))))
```

- Defining C-style structure:

```
(define-cstruct _hid_device_info
  ([path                _bytes]
   [vendor-id           _uint16]
    ...))
```

- Defining a definer:

```
(define-ffi-definer define-hidapi
  hidapi-lib
  #:make-c-id convention:hyphen->underscore
  #:default-make-fail make-not-available)
```

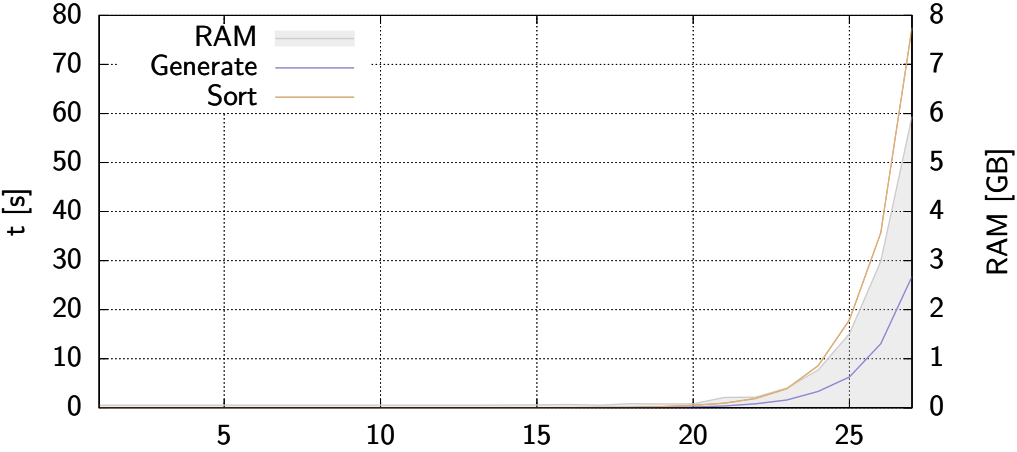
- Binding a function using previously defined definer:

```
(define-hidapi hid-enumerate
  (_fun _uint16 _uint16
   -> _hid_device_info-pointer/null))
```

- Productivity boost: Try THIS in Python ...

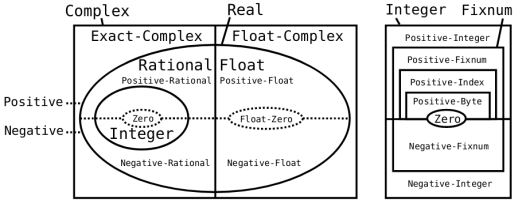
- Sort 2^{27} numbers
  - $134217728 \approx 134M$
- List
- Fixnums
- Futures

Parallelism with Futures: Sorting a List



Fixnums

- A `fixnum?` is a number $n \in \langle -2^{62}, 2^{62} - 1 \rangle$ on 64bit platforms
- Stored in 64 bits $= 8$ bytes
- Arithmetic operations are faster with fixnums than with other numbers types
- `+` $\rightarrow$ `fx+`, `-` $\rightarrow$ `fx-`, $<$ $\rightarrow$ `fx<`, ...
- Numeric tower[1]:



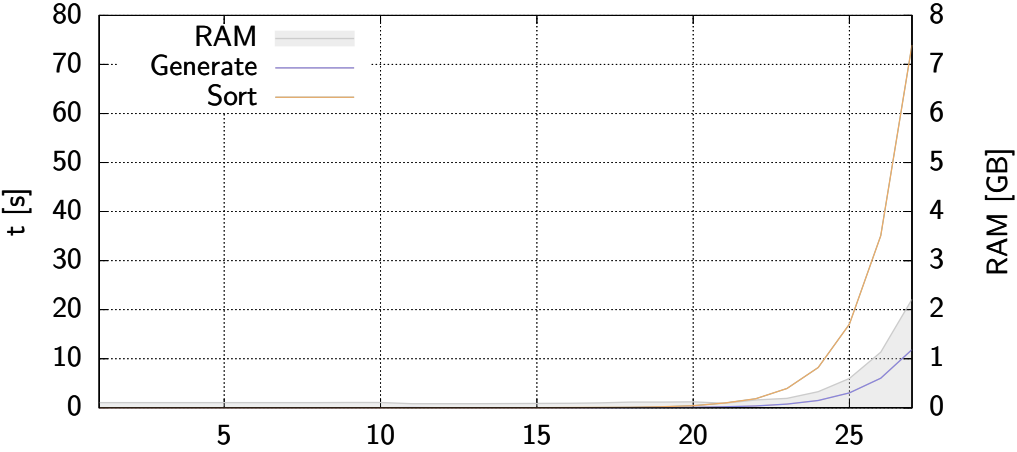
Fixnums – Unsafe Ops

- Assuming the value is a fixnum means we can bypass type checking
- These operations are deemed "unsafe" and can be pulled-in by:

```
(require racket/unsafe/ops)
```

- All fixnum operations get their unsafe versions:
  - `unsafe-fx+`
  - `unsafe-fx-`
  - `unsafe-fx<`
  - ...
- Merge-sort implementation using fixnums

Sorting Fixnums – Performance



Futures

- Threads in Racket are (almost) thread safe
- Explicit parallel execution is not simple
- Future – an expression for which we will want evaluated value later:

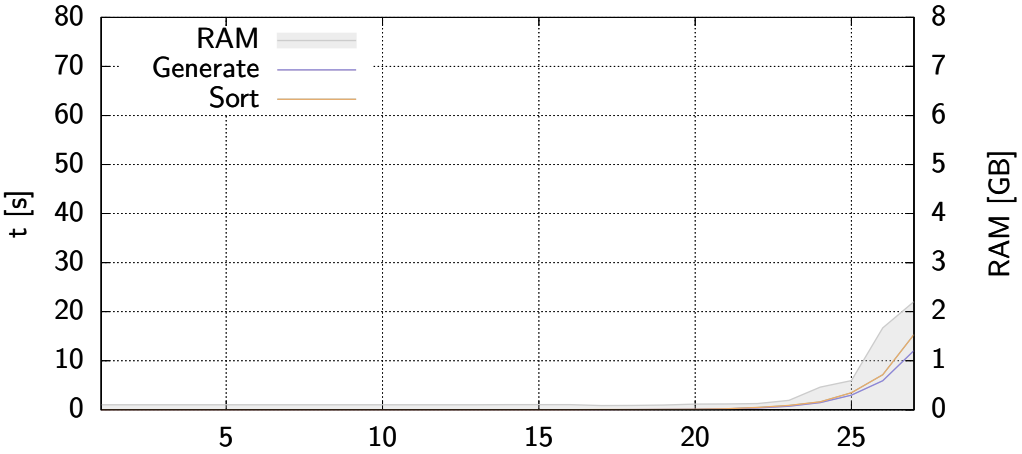
```
(define f (future (λ () (+ 1 2))))
```

- Touching the future yields its value

```
(displayln (touch f))
```

- Racket will try its best to have the value ready *before* it is touched

Sorting Fixnums with Futures – Performance



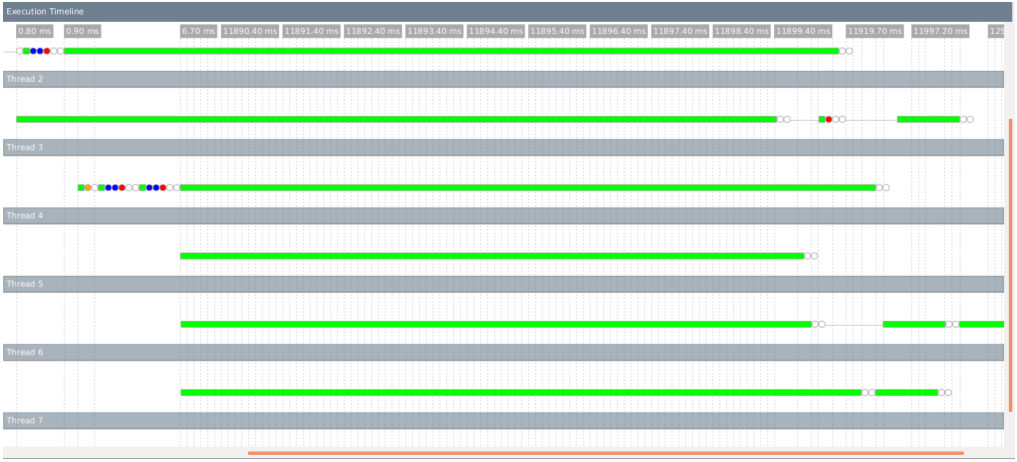
- Useful GUI library: future-visualizer:

```
(require future-visualizer)
```

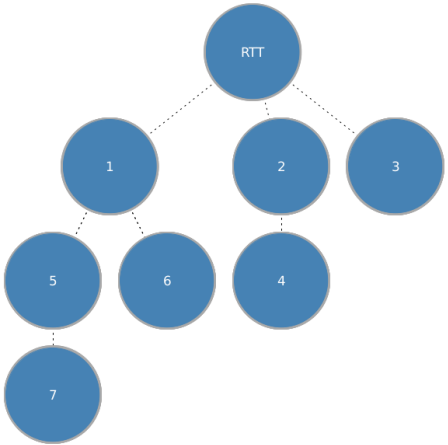
- Wrap your code in `visualize-futures` form:

```
(visualize-futures
  (fxvector-futures-sort! fxvec))
```

Futures Analysis – Timeline



Futures Analysis – Creation Tree



- Boost: Listen to the money talk!

- Programming 3D games:

- Productivity boost:
  - Negative.

Questions...

... and answers.

Thank you.
See you in the afternoon!