



RubyConf 2009

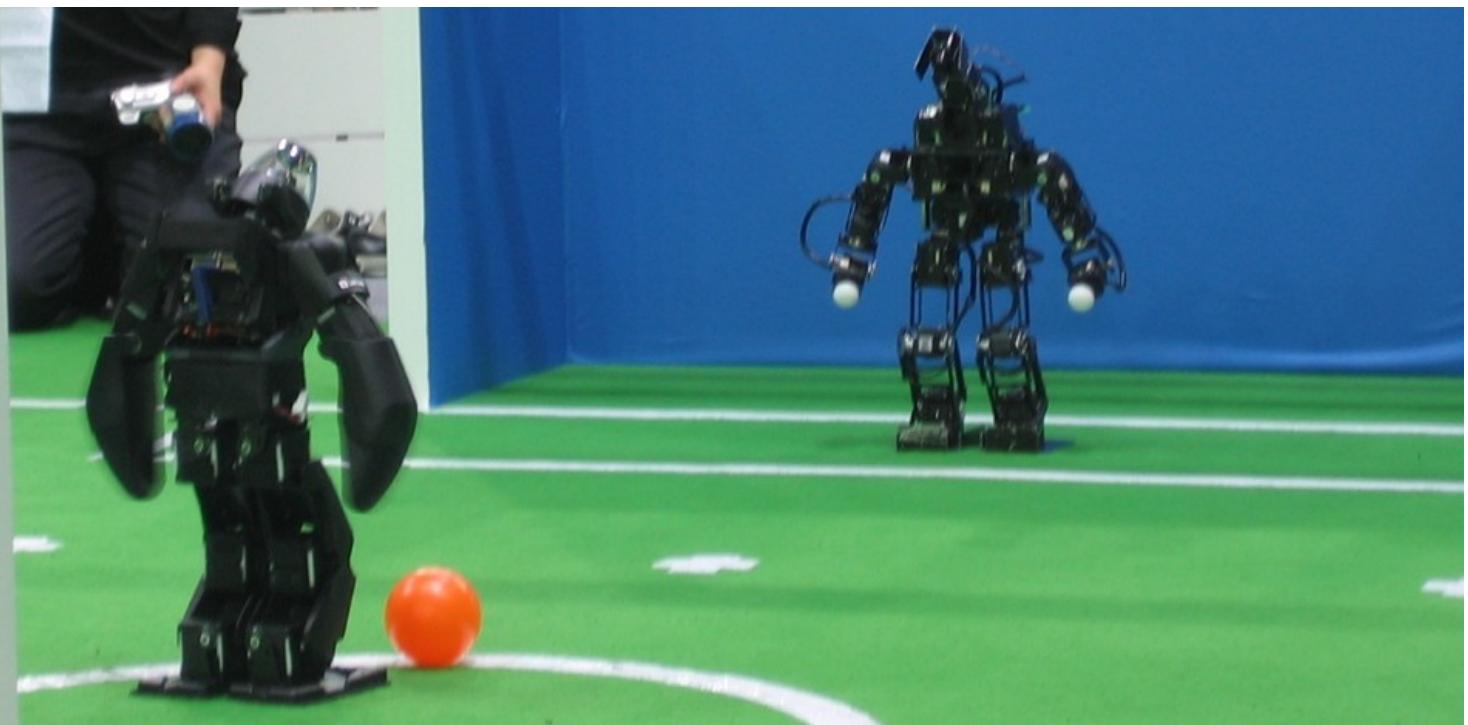
Computer Vision using Ruby and libJIT



Jan Wedekind

Thu Nov 19 13:15:00 PST 2009

Embassy Suites San Francisco Airport, Room 1

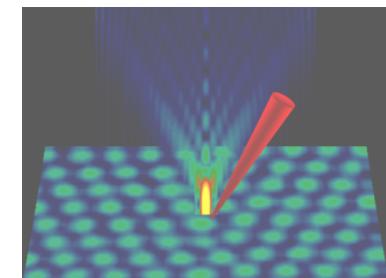
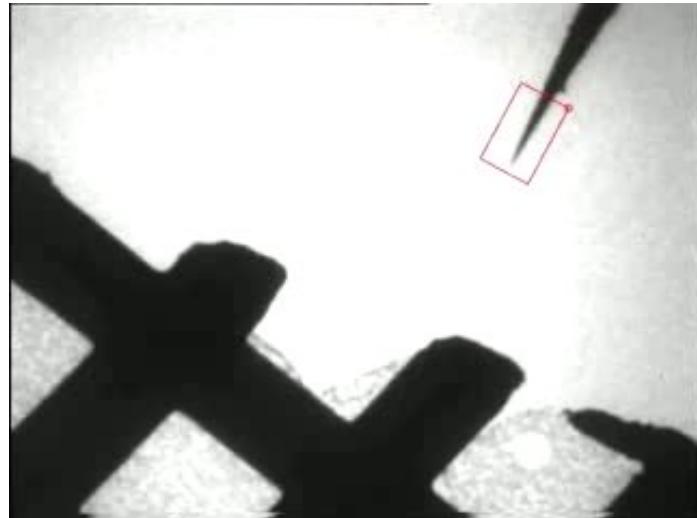


<http://www.flickr.com/photos/schoschie/171544759/>



Background

EPSRC Nanorobotics Project



EPSRC

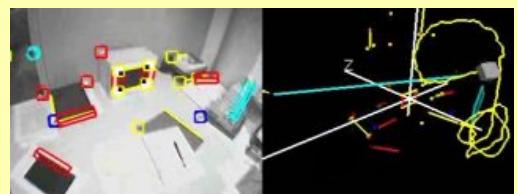


<http://vision.eng.shu.ac.uk/mmv1wiki/index.php/Nanorobotics>

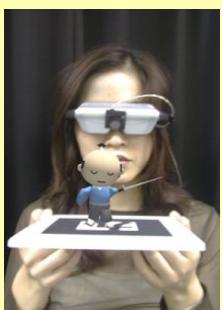
<http://nano.group.shef.ac.uk/>



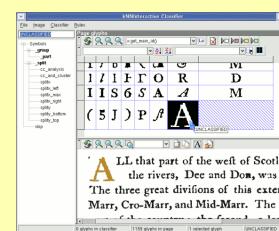
Applications of Computer Vision



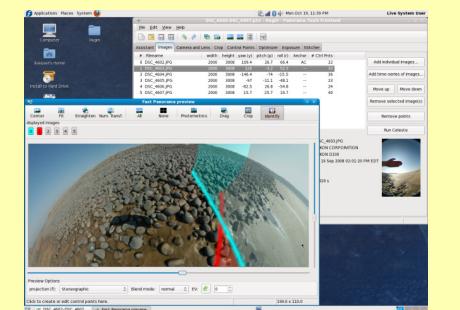
Visual Navigation



Augmented Reality



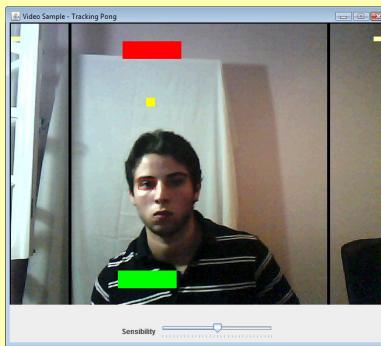
Optical Character
Recognition



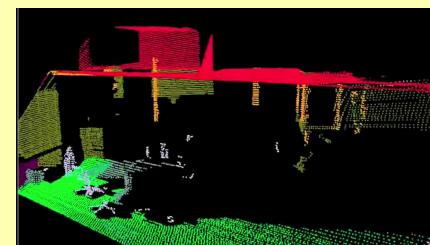
Panorama Stitching



3D Reconstruction



Games



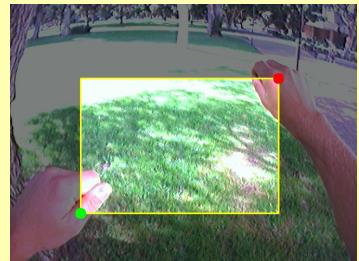
3D Object
Recognition



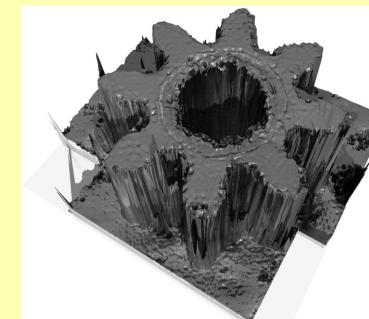
Surveillance



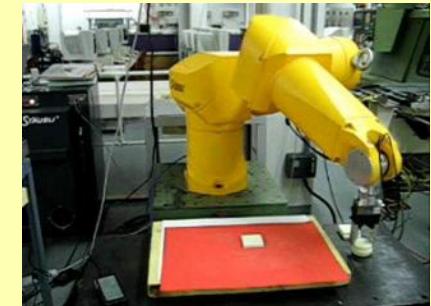
Segmentation



Human Computer
Interfaces



Inspection



Automation



The Array Challenge

Computer Vision means Array Operations

C/C++

```
#include <stdlib.h>
#define SIZE 10000000
int main(void)
{
    int i, *arr = (int *)
        malloc( SIZE *
                sizeof(int) );
    for ( i = 0; i < SIZE; i++ )
        arr[ i ] = i;
    free( arr );
    return 0;
}
```

gcc 4.2.4 0.06s

Ruby

```
SIZE = 10000000
arr = ( 0 ... SIZE ).  
      collect { |i| i }
```

ruby 1.8.6 76.3s

ruby 1.9.1 1.8s

Intel® Core™2 CPU T5600 @ 1.83GHz

Linux 2.6.24-24-generic SMP i686 GNU/Linux

Maybe we shouldn't use Ruby here?



<http://www.imdb.com/title/tt0343818/>



<http://vision.eng.shu.ac.uk/jan/rubyconf09-foils.pdf>

Expressiveness (multi-dimensional '+')

C++ and Boost::MultiArray

```
#include <boost/multi_array.hpp> // 3726 lines of code
#include <iostream>
using namespace boost;
template< typename T >
T &multi_plus( T &a, const T &b, const T &c )
{
    a = b + c;
    return a;
}
template< template< typename, size_t, typename > class Arr,
          typename Alloc, typename T, size_t N >
detail::multi_array::sub_array< T, N > multi_plus
( detail::multi_array::sub_array< T, N > a,
  const Arr< T, N, Alloc > &b, const Arr< T, N, Alloc > &c )
{
    typename Arr< T, N, Alloc >::const_iterator j = b.begin();
    typename Arr< T, N, Alloc >::const_iterator k = c.begin();
    for ( typename detail::multi_array::sub_array< T, N >::
          iterator i = a.begin(); i != a.end(); i++, j++, k++ )
        multi_plus( *i, *j, *k );
    return a;
}
template< template< typename, size_t, typename > class Arr,
          typename Alloc, typename T, size_t N >
Arr< T, N, Alloc > &multi_plus
( Arr< T, N, Alloc > &a,
  const Arr< T, N, Alloc > &b, const Arr< T, N, Alloc > &c )
{
    typename Arr< T, N, Alloc >::const_iterator j = b.begin();
    typename Arr< T, N, Alloc >::const_iterator k = c.begin();
    for ( typename Arr< T, N, Alloc >::
          iterator i = a.begin(); i != a.end(); i++, j++, k++ )
        multi_plus( *i, *j, *k );
    return a;
}
```

```
template < template< typename, size_t, typename > class Arr,
          typename Alloc, typename T, size_t N >
multi_array< T, N > operator+
( const Arr< T, N, Alloc > &a,
  const Arr< T, N, Alloc > &b )
{
    array< size_t, N > shape;
    std::copy( a.shape(), a.shape() + N, shape.begin() );
    multi_array< T, N > retVal( shape );
    multi_plus( retVal, a, b );
    return retVal;
}
int main(void)
{
    multi_array< int, 2 > a( extents[ 2 ][ 2 ] );
    a[ 0 ][ 0 ] = 1; a[ 0 ][ 1 ] = 2; a[ 1 ][ 0 ] = 3; a[ 1 ][ 1 ] = 4;
    multi_array< int, 2 > b( extents[ 2 ][ 2 ] );
    b[ 0 ][ 0 ] = 5; b[ 0 ][ 1 ] = 4; b[ 1 ][ 0 ] = 3; b[ 1 ][ 1 ] = 2;
    multi_array< int, 2 > r( a + b );
    std::cout << "[ [ " << r[ 0 ][ 0 ] << ", " << r[ 0 ][ 1 ]
                << " ], [ " << r[ 1 ][ 0 ] << ", " << r[ 1 ][ 1 ]
                << " ] ] " << std::endl;
    // [ [ 6, 6 ], [ 6, 6 ] ]
    return 0;
}
```



Expressiveness (multi-dimensional '+')

Ruby and Standard Library

```
class Array

  def +( other )

    zip( other ).collect { |x,y| x + y }

  end

end

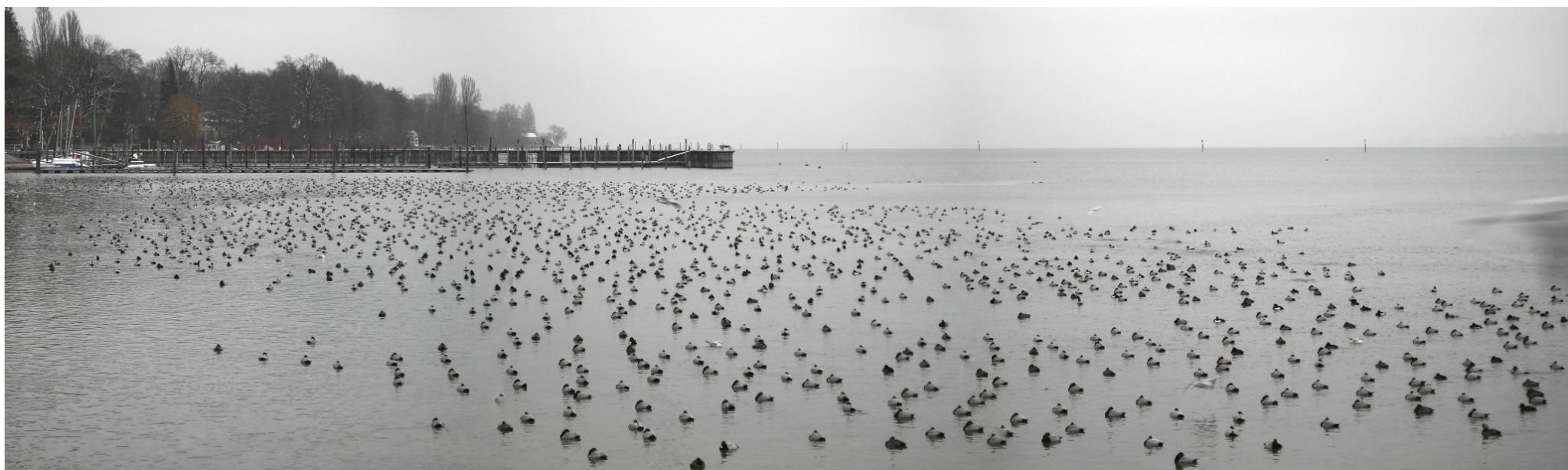
a = [ [ 1, 2 ], [ 3, 4 ] ]
b = [ [ 5, 4 ], [ 3, 2 ] ]
puts ( a + b ).inspect
# [ [ 6, 6 ], [ 6, 6 ] ]
```

straightforward!



Implementation

Uniform Arrays



[ ,  ,  ,  ,  ,  ,  , ...]

Also see: [NArray](#)



Implementation

Malloc Objects

```
m = Malloc.new 10
```

```
m.write '0123456789'
```

```
# "0123456789"
```

```
m.read 5
```

```
# "01234"
```

```
( m + 2 ).read 5
```

```
# "23456"
```



Implementation

Using 'Array#pack' and 'String#unpack'

```
class INT
  def initialize( v = 0 )
    @malloc = Malloc.new 4
    set v
  end
  def inspect
    "#{ self.class.inspect }(#{ get })"
  end
  def set( v = 0 )
    @malloc.write [ v ].pack( 'i' )
    v
  end
  def get
    @malloc.read( 4 ).unpack( 'i' )[ 0 ]
  end
end
```

```
i = INT.new 3
# INT(3)
i.set 5
# 5
i
# INT(5)
i.get
# 5
```



Implementation

Type System: Generic Native Scalar

```
class INT_
  class << self
    attr_accessor :bits
    attr_accessor :signed
  end
  # ...
end
SIGNED = true
UNSIGNED = false
def INT( bits, signed )
  retval = Class.new INT_
  retval.bits = bits
  retval.signed = signed
  retval
end
```

```
USINT = INT 16, UNSIGNED
# USINT
u = USINT.new 3
# USINT(3)
v = USINT.new u.malloc
# USINT(3)
```



Implementation

Type System: Native Arrays

```
class Sequence< Object
  class << self
    attr_accessor :elem, :num,
                  :stride
  end
  # ...
  def sel( i )
    elem.new @malloc + i *
              elem.bytes * stride
  end
  def []( i )
    sel( i ).get
  end
  def []=( i, v )
    sel( i ).set v
  end
end
```

```
SINT = INT 16, SIGNED
# SINT
s = Sequence( SINT, 8 ).new.fill!
# Sequence(SINT,8):
# [ 0, 0, 0, 0, 0, 0, 0, 0 ]
s[ 3 ] = 5
# 5
s
# Sequence(SINT,8):
# [ 0, 0, 0, 5, 0, 0, 0, 0 ]
s[ 3 ]
# 5
s.sel 3
# SINT(5)
```



Implementation

Type System: Recursive Arrays

```
M = Sequence( Sequence( INT( 32, SIGNED ), 3 ), 2 )

# MultiArray.int(3,2)
m = M.new.fill! 1

# MultiArray.int(3,2):
# [ [ 1, 1, 1 ],
#   [ 1, 1, 1 ] ]

m[ 1 ]
# Sequence.int(3):
# [ 1, 1, 1 ]

m[ 2, 1 ]
# 1

m[ 1 ][ 2 ] = 0
# 0

m[ 1 .. 2, 0 .. 1 ]
# MultiArray.int(2,2):
# [ [ 1, 1 ],
#   [ 1, 0 ] ]
```

Array Operations





Implementation

Array Operations: Scalar Operations

```
module RubyScalar

  def op( *args, &action )
    instance_exec *args, &action
    self
  end

  def -@
    self.class.new.op( get ) { |x| set -x }
  end

  # ...

end

class INT_
  include RubyScalar
end
```

```
i = SINT.new 3
# SINT(3)
-i
# SINT(-3)
```



Implementation

Array Operations: Element-Wise Array Operations

```
module RubySequence

  def op( *args, &action )
    for i in 0 ... num
      subargs = args.collect do |v|
        v.is_a?( Sequence_ ) ? v[ i ] : v
      end
      sel( i ).op *subargs, &action
    end
    self
  end

  def -@
    self.class.new.op( get ) { |x| set -x }
  end
end

class Sequence_
  include RubySequence
end
```

```
S = Sequence( INT, 3 )
# Sequence.int(3)
s = S.new.indgen!
# Sequence.int(3):
# [ 0, 1, 2 ]
-s
# Sequence.int(3):
# [ 0, -1, -2 ]
```

How to speed-up things?





Implementation

Reflection in Ruby: Example

```
class Const
  attr_accessor :inspect
  alias_method :to_s, :inspect
  def initialize( s )
    @inspect = s.to_s
  end
  def method_missing( name, *args )
    str = "#{ self }.#{ name }"
    unless args.empty?
      str += "( #{args.join ', '} )"
    end
    Const.new str
  end
  def coerce( y )
    return Const.new( y ), self
  end
end
```

```
a = Const.new 'a'
# a
b = Const.new 'b'
# b
-a
# a.-@
a + b
# a.+( b )
a[ 2 ]
# a[]( 2 )
2 * a
# 2.*( a )
2 * a + b
# 2.*( a ).+( b )
2 * ( a + b )
# 2.*( a.+( b ) )
```



Implementation

Reflection in Ruby: Limitations

a or b	b and a
# a	# a
a < b ? a : b	a > b ? a : b
# a	# a
b = a	x = a
# a	# a
if a > b	a - b
a -= b	#
end	#
# a.-(b)	# a.-(b)
begin	b = a + 1
a += 1	#
end until a > b	#
a	#
# a.+(1)	# a.+(1)



Implementation

JIT-Compilation: Scalar Operation I/II

```
module JITScalar
  def op(*args, &action)
    if ([self] + args).all? { |arg| arg.jit_support? }
      JITFunction.jit(self, *args) do |f, _self, *_args|
        _self.op *_args, &action
      end
    self
  else
    super *args, &action
  end
end
class INT_
  include JITScalar
end
```

'JITFunction.jit': cache, compile, and execute



Implementation

JIT-Compilation: Scalar Operation II/II

```
i = INT.new 3
# INT(3)
i.class.new.op( i.get ) do |x|
  puts "self = #{self}"
  puts "x      = #{x}"
  puts
  set -x
end
# self = INT(*i1)
# x      = i2
# INT(-3)
```



```
function -(@( i1: ptr,
              i2: int ): void
  i3 = -i2
  *i1 = i3
```

machine code generated by libJIT/ DotGNU

<http://freshmeat.net/projects/libjit/>

Also see: [ruby-libjit](#)



Implementation

JIT-Compilation: Array Operation I/V

```
module JITSequence
  def op(*args, &action)
    if ([self] + args).all? { |arg| arg.jit_support? }
      JITFunction.jit(self, *args) do |f,_self,*_args|
        _self.op *_args, &action
      end
    self
  elsif is_jit?
    f = malloc.f
    r = JITTerm.value f, JITType::PTR, malloc
    s = stride * elem.bytes
    rend = r + s * num
  end
end
```



Implementation

JIT-Compilation: Array Operation II/V

```
extract, increment = [], []
args.each do |arg|
  if arg.is_a? Sequence_
    ptr = JITTerm.value f, JITType::PTR, arg.malloc
    astride = arg.stride * arg.elem.bytes
    extract.push proc { arg.elem.new( ptr ).get }
    increment.push proc { ptr.set ptr + astride }
  else
    extract.push proc { arg }
  end
end
```



Implementation

JIT-Compilation: Array Operation III/V

```
f.until( proc { r == rend } ) do
    _subargs = extract.collect { |i| i.call }
    elem.new( r ).op *_subargs, &action
    increment.each { |i| i.call }
    r.set r + s
end
else
    super *args, &action
end
end
end
class Sequence_
    include JITSequence
end
```

Also cumbersome but much more generic!



Implementation

JIT-Compilation: Array Operation IV/V

```
s = Sequence( INT, 3 ).new.indgen!
# Sequence(INT,3):
# [ 0, 1, 2 ]
s.class.new.op( s.get ) do |x|
  puts "self = #{self}"
  puts "x      = #{x}"
  puts
  set -x
end
# self = INT(*i7)
# x      = i13
# Sequence(INT,3):
# [ 0, -1, -2 ]
```





Implementation

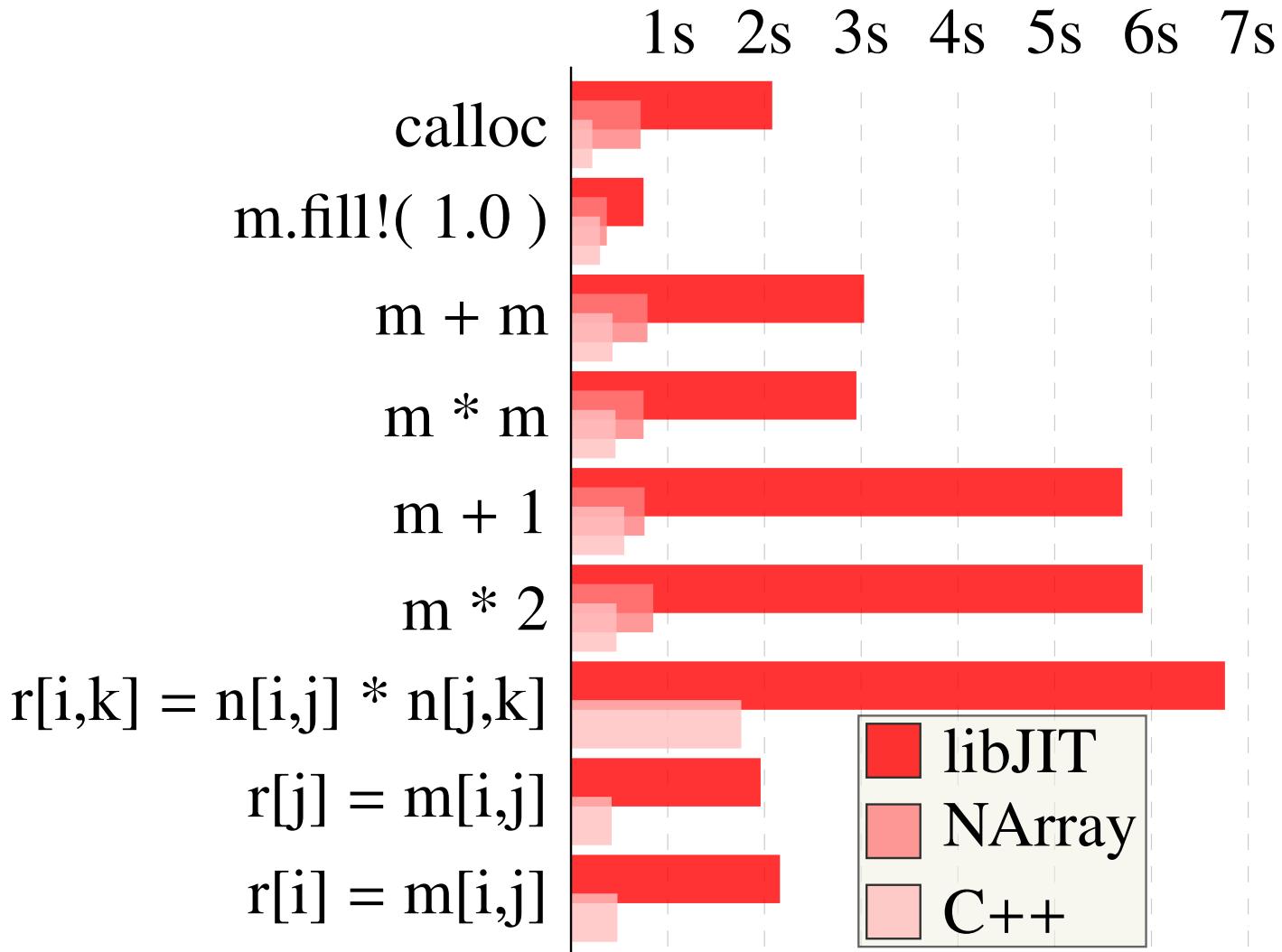
JIT-Compilation: Array Operation V/V

```
function -@( int i1, int i2, int i3: ptr,  
           int i4, int i5: int, int i6: ptr ): void  
  
    i7 = i3  
    i8 = i2 * 4  
    i9 = i8 * i1  
    i10 = i7 + i9  
    i11 = i6  
    i12 = i5 * 4  
  
    L0:  
        if i7 == i10 then goto L1  
        i13 = *i11  
        i14 = -i13  
        *i7 = i14  
        i11 = i11 + i12  
        i7 = i7 + i8  
        goto L0  
  
    L1:
```



Implementation

JIT Results



Intel® Core™2 CPU T5600 @ 1.83GHz

Linux 2.6.24-24-generic SMP i686 GNU/Linux

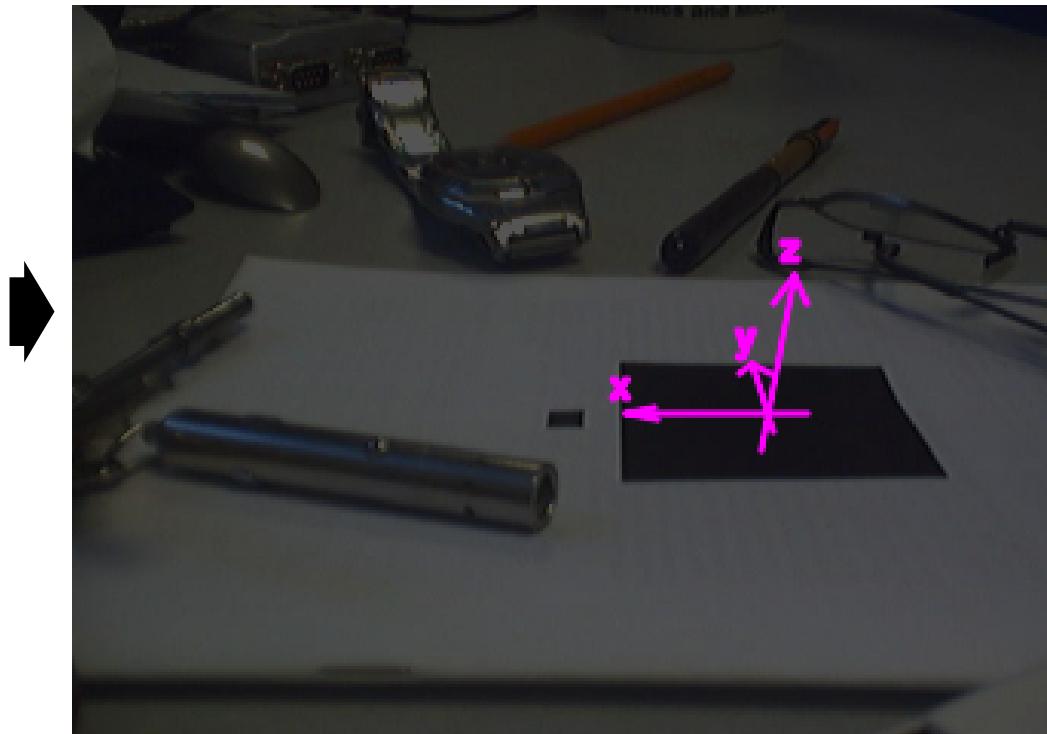
⇒ future work: prefetch cache with GCC code (use [RubyInline](#)?)

Augmented Reality Example

Problem: 3D recognition of planar object

How to get from here ...

... to here?

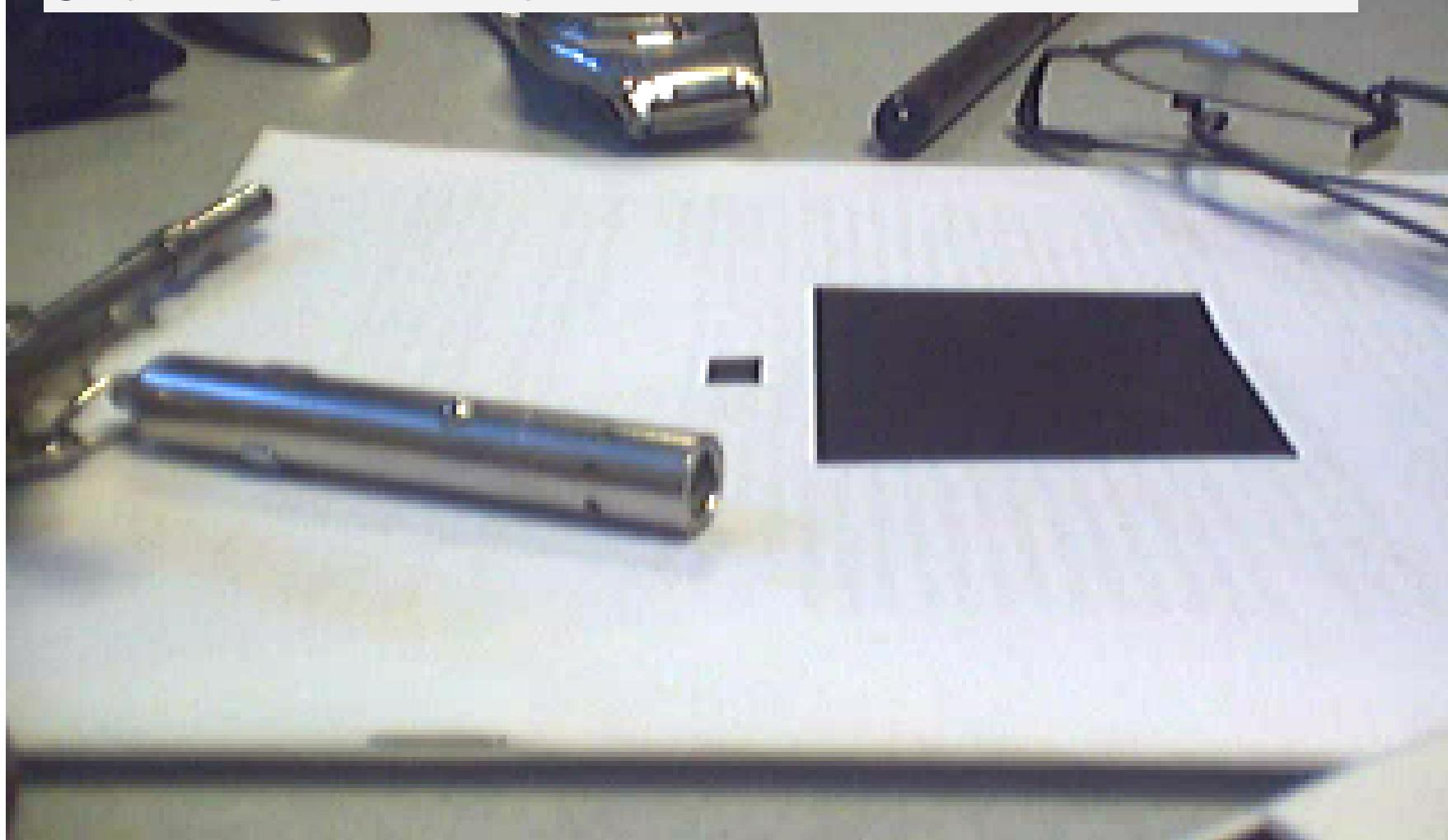




Augmented Reality Example

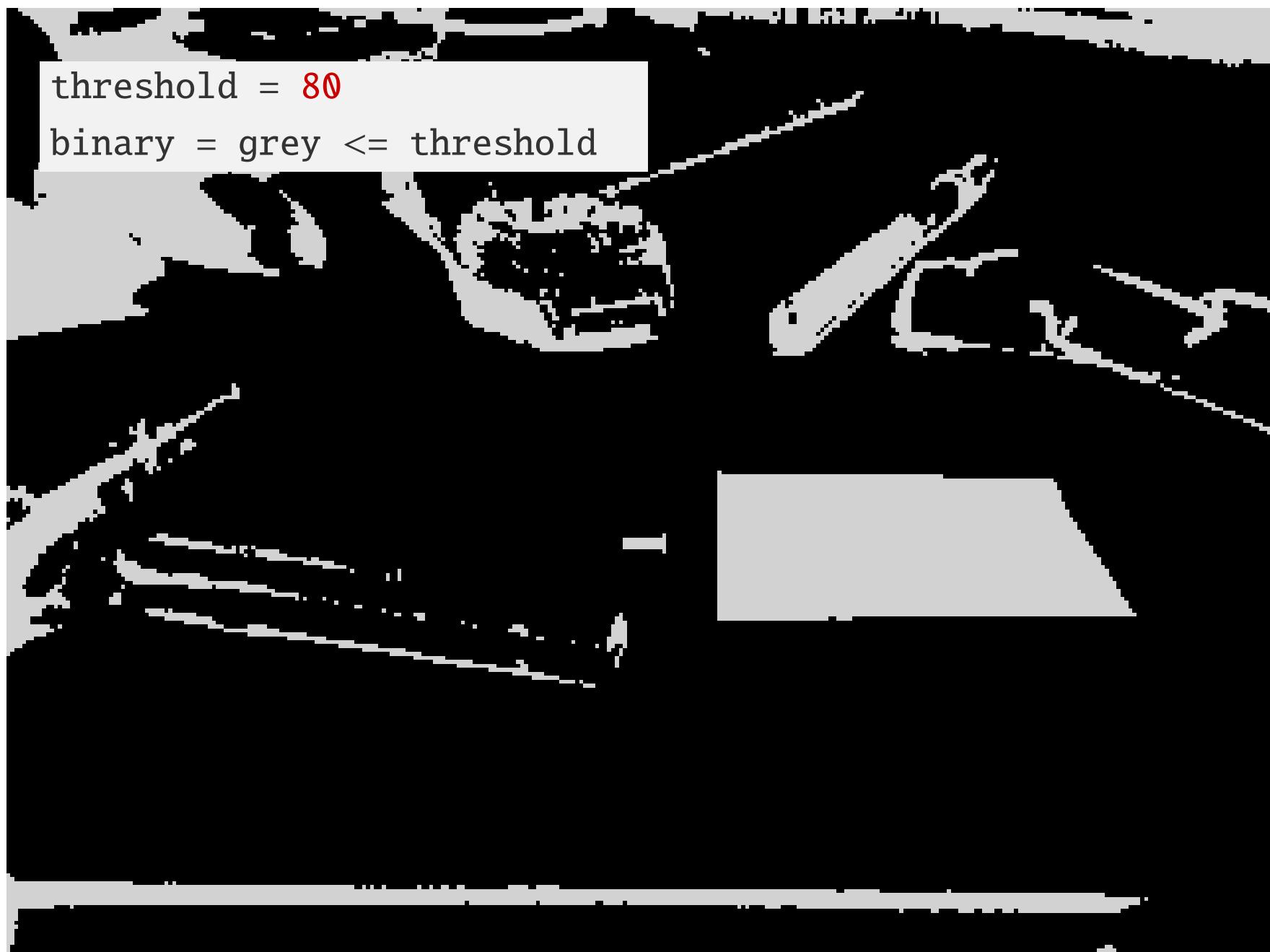
Capture Image

```
# Ommitting initial monkey-patching and 'requires' ...
input = V4L2Input.new '/dev/video0', 320, 240
grey = input.read_ubyte
```



Augmented Reality Example

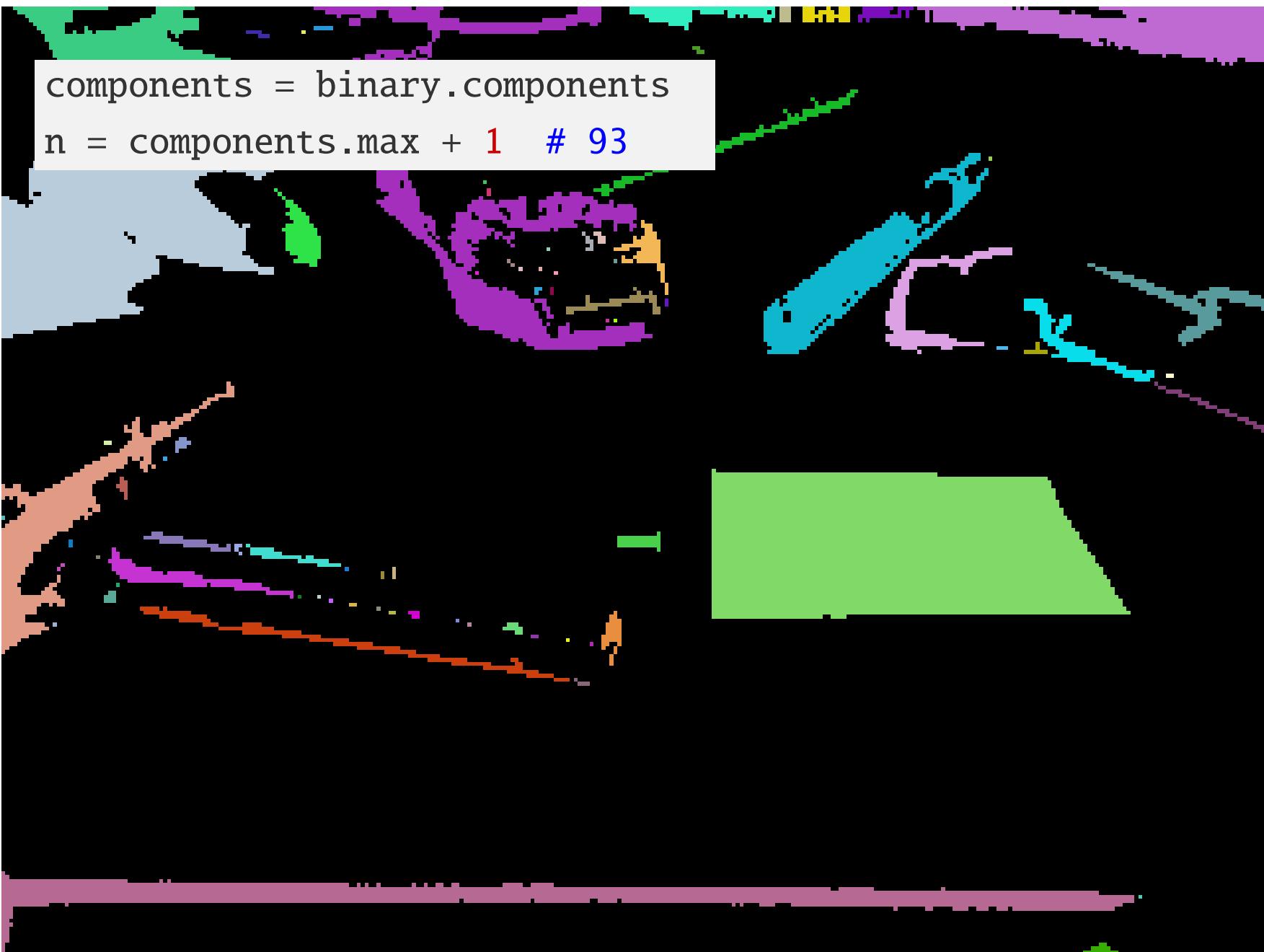
Threshold Image



Augmented Reality Example

Connected Component Analysis

```
components = binary.components  
n = components.max + 1 # 93
```





Augmented Reality Example

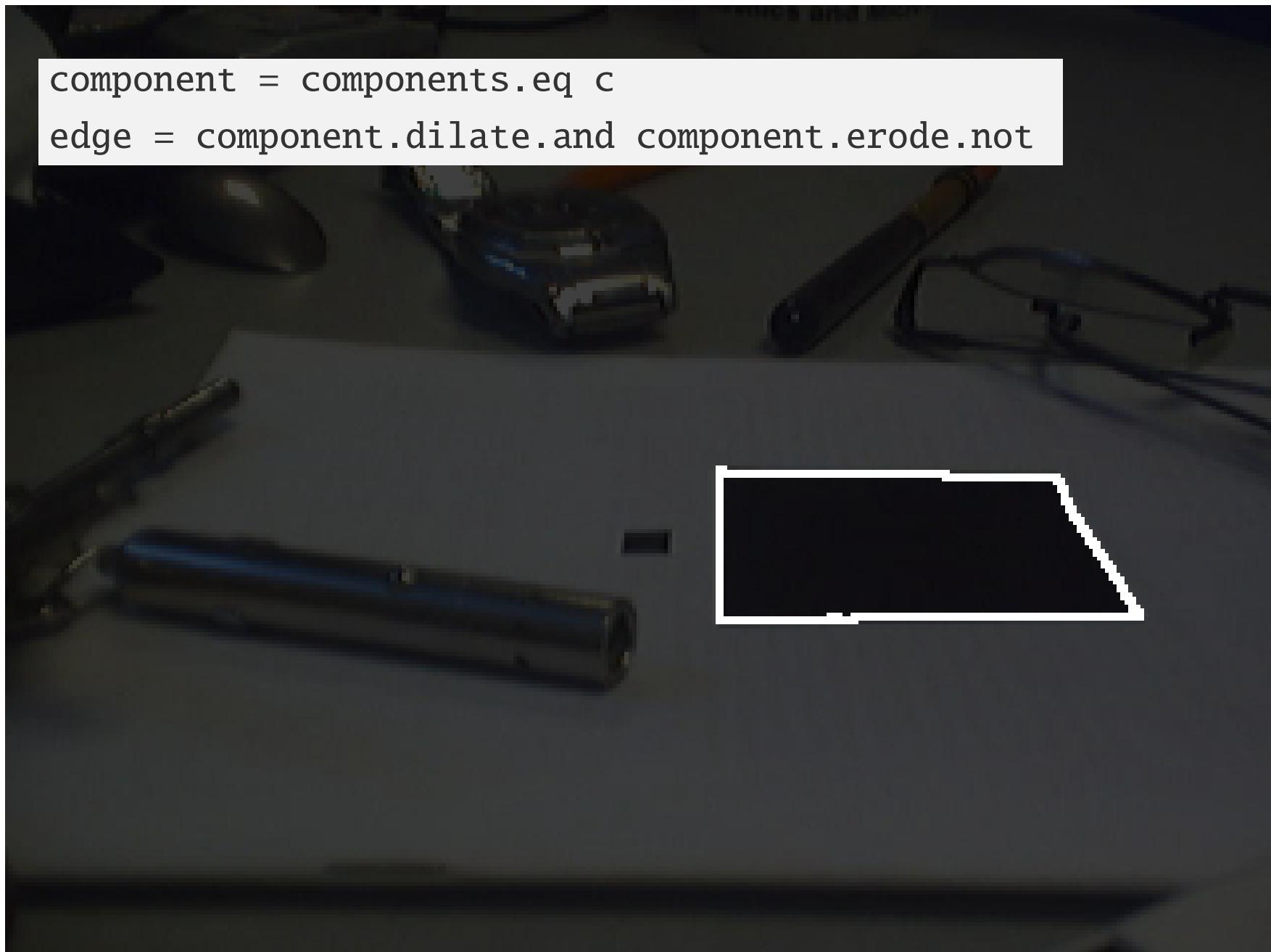
Impose Size Constraints

```
range = 30 ** 2 .. 100 ** 2
hist = components.hist n
mask = hist.between? range.min, range.max
Sequence.int( n ).indgen!.mask( mask ) .
  to_a.each do |c|
    # c is 27, 47, 56, or 90
    # ...
end
```

Augmented Reality Example

Extract Edge of Component

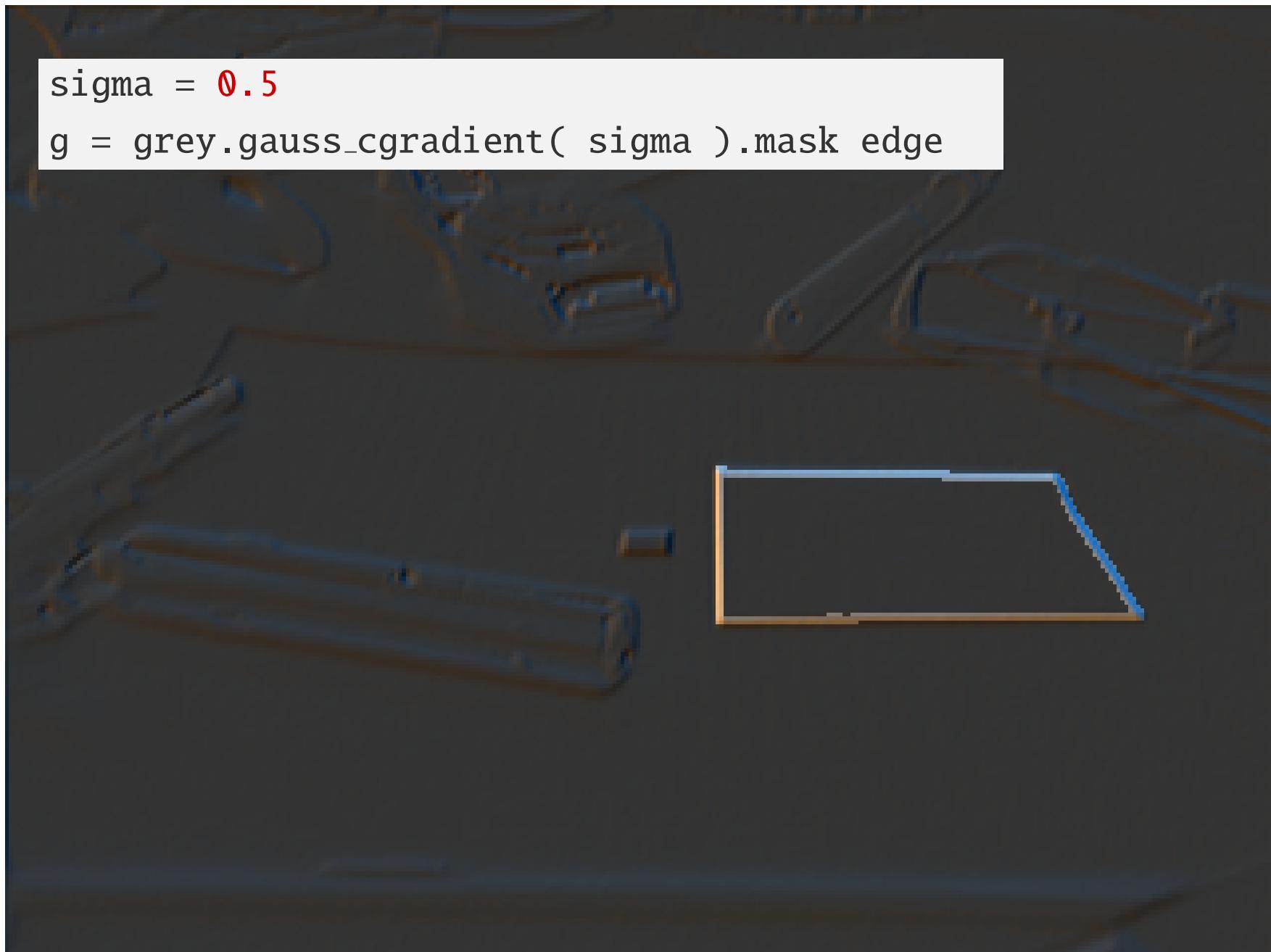
```
component = components.eq c  
edge = component.dilate.and component.erode.not
```



Augmented Reality Example

Compute Gradients

```
sigma = 0.5  
g = grey.gauss_cgradient( sigma ).mask edge
```



Augmented Reality Example

Group Dominant Orientations

```
q = 36
d_q = ( ( g.arg / Math::PI + 1 ) *
         q / 2 ).to_int % q
d_hist = d_q.hist_weighted q, g.abs
msk = d_hist >= d_hist.max / 4
segments = msk.components
partitions = d_q.map segments
```





Augmented Reality Example

Centre of each Line

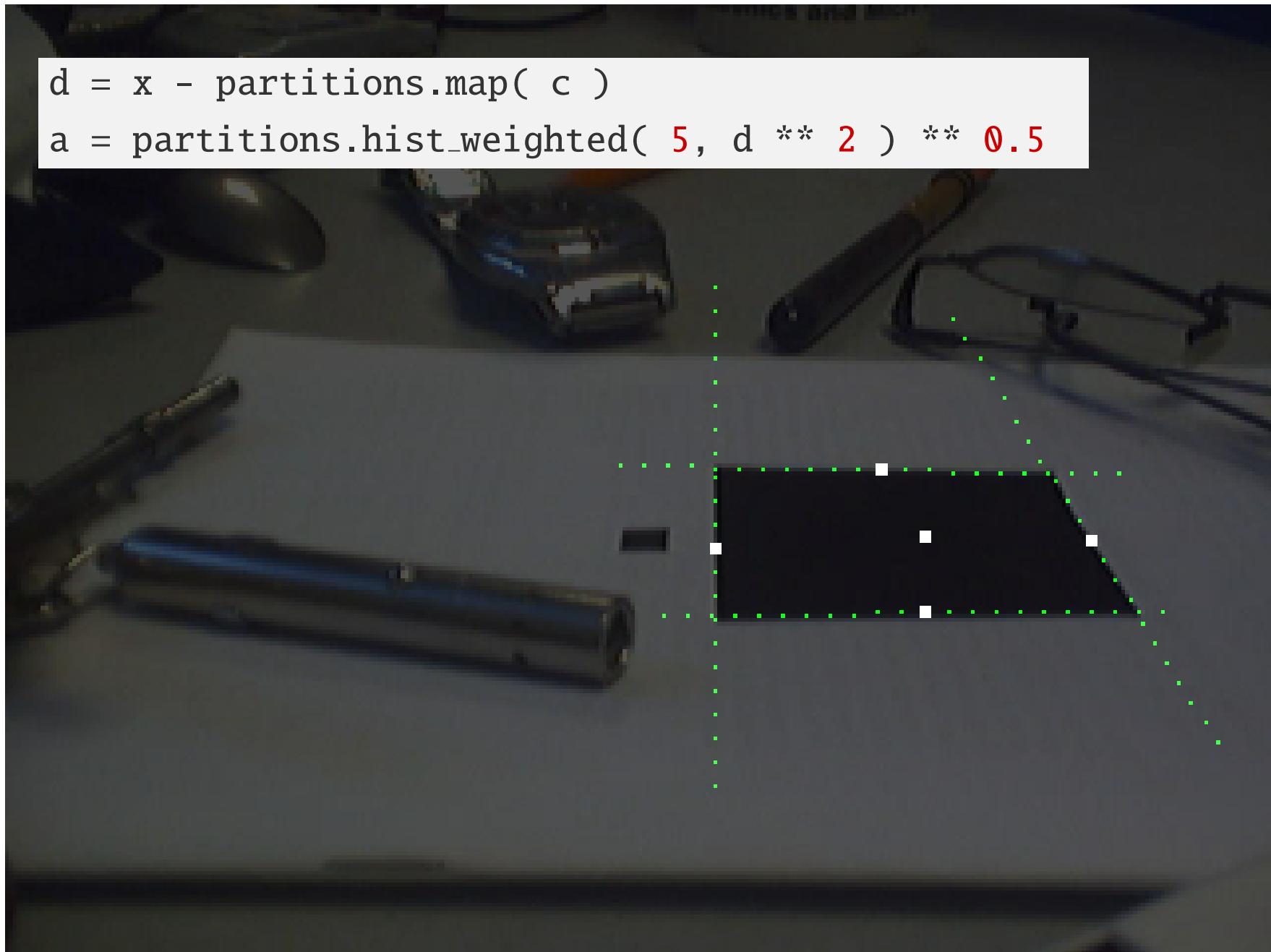
```
if segments.max == 4
  cramp = MultiArray.complex_ramp *grey.shape
  x = cramp.mask edge
  weights = partitions.hist( 5 ).major 1.0
  c = partitions.hist_weighted( 5, x ) / weights
  # ...
end
```



Augmented Reality Example

Angle of each Line

```
d = x - partitions.map( c )  
a = partitions.hist_weighted( 5, d ** 2 ) ** 0.5
```





Augmented Reality Example

Intersections

```
ms = Sequence[ *( 0 ... 4 ).collect do |i|
  i1, i2 = i + 1, ( i + 1 ) % 4 + 1
  l1, a1, l2, a2 = c[i1], a[i1], c[i2], a[i2]
  ( l1 * a1.conj * a2 - l2 * a1 * a2.conj -
    l1.conj * a1 * a2 + l2.conj * a1 * a2 ) /
  ( a1.conj * a2 - a1 * a2.conj )
end ] - 0.5 * Complex( *grey.shape )
```



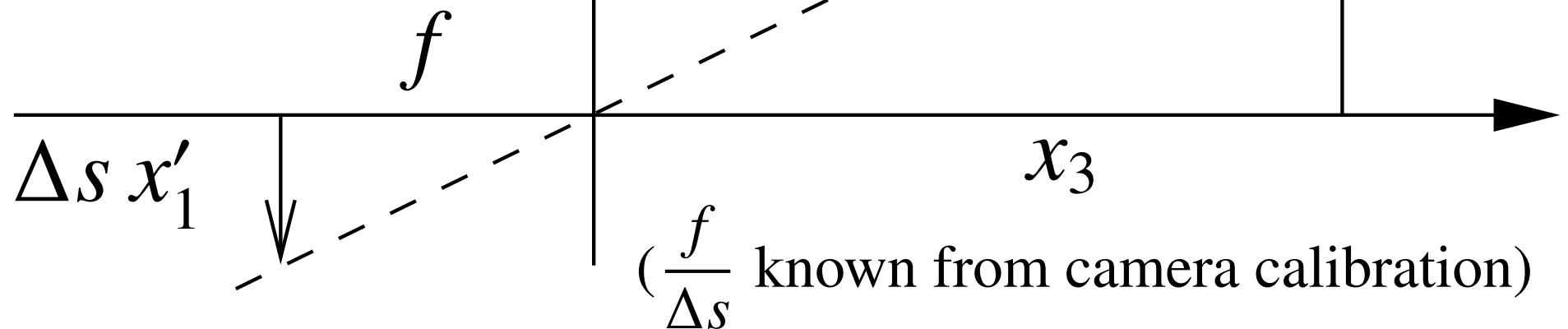
```
s = 0.05 # meter
m = Sequence[ Complex( -1, -1 ), Complex( 1, -1 ),
  Complex( 1, 1 ), Complex( -1, 1 ) ] * s / 2
```

Augmented Reality Example

3D Pose: Pinhole Camera Model

$$\frac{\Delta s x'_1}{f} = \frac{x_1}{x_3}$$

$$\frac{\Delta s x'_2}{f} = \frac{x_2}{x_3}$$





Augmented Reality Example

3D Pose: Homogeneous Coordinates

$$x_3 x'_1 = \frac{f}{\Delta s} x_1$$

$$x_3 x'_2 = \frac{f}{\Delta s} x_2$$

$$\lambda x'_1 = \frac{f}{\Delta s} x_1$$

$$\Leftrightarrow \exists \lambda \in \mathbb{R}/\{0\} : \lambda x'_2 = \frac{f}{\Delta s} x_2$$

$$\lambda = x_3$$

$$\Leftrightarrow \exists \lambda \in \mathbb{R}/\{0\} : \lambda \begin{pmatrix} x'_1 \\ x'_2 \\ 1 \end{pmatrix} = \underbrace{\begin{pmatrix} f/\Delta s & 0 & 0 \\ 0 & f/\Delta s & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{\text{intrinsic camera parameters}} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$



Augmented Reality Example

3D Pose: (Additional) Affine Transform

$$\exists \lambda \in \mathbb{R}/\{0\} : \lambda \begin{pmatrix} x'_1 \\ x'_2 \\ 1 \end{pmatrix} = \begin{pmatrix} f/\Delta s & 0 & 0 \\ 0 & f/\Delta s & 0 \\ 0 & 0 & 1 \end{pmatrix} \left(\begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} t_1 \\ t_2 \\ t_3 \end{pmatrix} \right)$$

$$\Leftrightarrow \exists \lambda \in \mathbb{R}/\{0\} : \lambda \begin{pmatrix} x'_1 \\ x'_2 \\ 1 \end{pmatrix} = \begin{pmatrix} f/\Delta s & 0 & 0 \\ 0 & f/\Delta s & 0 \\ 0 & 0 & 1 \end{pmatrix} \underbrace{\begin{pmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{pmatrix}}_{\text{extrinsic camera parameters}} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} x'_1 \\ x'_2 \end{pmatrix} \text{ is } \begin{pmatrix} m'_{11} \\ m'_{12} \end{pmatrix}, \begin{pmatrix} m'_{21} \\ m'_{22} \end{pmatrix}, \dots, \begin{pmatrix} m'_{41} \\ m'_{42} \end{pmatrix} \text{ and } \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \text{ is } \begin{pmatrix} m_{11} \\ m_{12} \\ 0 \end{pmatrix}, \begin{pmatrix} m_{21} \\ m_{22} \\ 0 \end{pmatrix}, \dots, \begin{pmatrix} m_{41} \\ m_{42} \\ 0 \end{pmatrix}$$



Augmented Reality Example

3D Pose: Planar Homography I/II

Problem: Minimize ϵ_{ij}

$$\exists \lambda_i \in \mathbb{R}/\{0\} : \lambda_i \begin{pmatrix} \begin{pmatrix} m'_{i1} \\ m'_{i2} \\ 1 \end{pmatrix} + \begin{pmatrix} \epsilon_{i1} \\ \epsilon_{i2} \\ 0 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} f/\Delta s & 0 & 0 \\ 0 & f/\Delta s & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{pmatrix} \begin{pmatrix} m_{i1} \\ m_{i2} \\ 0 \\ 1 \end{pmatrix}$$

$$\Leftrightarrow \exists \lambda_i \in \mathbb{R}/\{0\} : \lambda_i \underbrace{\begin{pmatrix} \begin{pmatrix} m'_{i1} \\ m'_{i2} \\ 1 \end{pmatrix} + \begin{pmatrix} \epsilon_{i1} \\ \epsilon_{i2} \\ 0 \end{pmatrix} \end{pmatrix}}_{=: \mathcal{H}} = \begin{pmatrix} f/\Delta s & 0 & 0 \\ 0 & f/\Delta s & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & t_1 \\ r_{21} & r_{22} & t_2 \\ r_{31} & r_{32} & t_3 \end{pmatrix} \begin{pmatrix} m_{i1} \\ m_{i2} \\ 1 \end{pmatrix}$$

$$\Leftrightarrow \exists \lambda_i \in \mathbb{R}/\{0\} : \lambda_i \begin{pmatrix} \begin{pmatrix} m'_{i1} \\ m'_{i2} \\ 1 \end{pmatrix} + \begin{pmatrix} \epsilon_{i1} \\ \epsilon_{i2} \\ 0 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{33} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \begin{pmatrix} m_{i1} \\ m_{i2} \\ 1 \end{pmatrix}, i \in \{1, 2, \dots, 4\}$$



Augmented Reality Example

3D Pose: Planar Homography II/II

$$\Leftrightarrow \underbrace{(h_{31} m_{i1} + h_{32} m_{i2} + h_{33})}_{\lambda_i} \begin{pmatrix} \begin{pmatrix} m'_{i1} \\ m'_{i2} \end{pmatrix} + \begin{pmatrix} \epsilon_{i1} \\ \epsilon_{i2} \end{pmatrix} \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{33} \end{pmatrix} \begin{pmatrix} m_{i1} \\ m_{i2} \\ 1 \end{pmatrix}$$

assuming $\lambda_1 \approx \lambda_2 \approx \dots \approx \lambda_4$

this is approximately the same as

Problem: Minimize $\tilde{\epsilon}_{ij}$

$$(h_{31} m_{i1} + h_{32} m_{i2} + h_{33}) \begin{pmatrix} m'_{i1} \\ m'_{i2} \end{pmatrix} + \begin{pmatrix} \tilde{\epsilon}_{i1} \\ \tilde{\epsilon}_{i2} \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{33} \end{pmatrix} \begin{pmatrix} m_{i1} \\ m_{i2} \\ 1 \end{pmatrix}$$

$$\Leftrightarrow \begin{pmatrix} \tilde{\epsilon}_{i1} \\ \tilde{\epsilon}_{i2} \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{33} \end{pmatrix} \begin{pmatrix} m_{i1} \\ m_{i2} \\ 1 \end{pmatrix} - (h_{31} m_{i1} + h_{32} m_{i2} + h_{33}) \begin{pmatrix} m'_{i1} \\ m'_{i2} \end{pmatrix}$$



Augmented Reality Example

3D Pose: Singular Value Decomposition

$$\underbrace{\begin{pmatrix} m_{11} & m_{12} & 1 & 0 & 0 & 0 & -m'_{11} m_{11} & -m'_{11} m_{12} & -m'_{11} \\ 0 & 0 & 0 & m_{11} & m_{12} & 1 & -m'_{12} m_{11} & -m'_{12} m_{12} & -m'_{12} \\ m_{21} & m_{22} & 1 & 0 & 0 & 0 & -m'_{21} m_{21} & -m'_{21} m_{22} & -m'_{21} \\ 0 & 0 & 0 & m_{21} & m_{22} & 1 & -m'_{22} m_{21} & -m'_{22} m_{22} & -m'_{22} \\ \vdots & \vdots \\ m_{41} & m_{42} & 1 & 0 & 0 & 0 & -m'_{41} m_{41} & -m'_{41} m_{42} & -m'_{41} \\ 0 & 0 & 0 & m_{41} & m_{42} & 1 & -m'_{42} m_{41} & -m'_{42} m_{42} & -m'_{42} \end{pmatrix}}_{=: \mathcal{M}} = \underbrace{\begin{pmatrix} h_{11} \\ h_{12} \\ \vdots \\ h_{33} \end{pmatrix}}_{=: \vec{h}} = \begin{pmatrix} \tilde{\epsilon}_{11} \\ \tilde{\epsilon}_{12} \\ \tilde{\epsilon}_{21} \\ \tilde{\epsilon}_{22} \\ \vdots \\ \tilde{\epsilon}_{41} \\ \tilde{\epsilon}_{42} \end{pmatrix}$$

$\|\vec{h}\| = \mu \neq 0$ to avoid trivial solution

I.e. find $\vec{h} \in \mathbb{R}^9 : \|\mathcal{M}\vec{h}\|$ minimal and $\|\vec{h}\| = \mu$

Solution: Perform **SVD** $\mathcal{M} = \mathcal{U} \Sigma \mathcal{V}^*$ and choose $\vec{h} = \mu \vec{v}_9$ where \vec{v}_9 is right-handed singular vector with the smallest singular value σ_9



Augmented Reality Example

3D Pose: 3D Homography

$$\mathcal{H} = \mu \begin{pmatrix} v_{91} & v_{92} & v_{93} \\ v_{94} & v_{95} & v_{96} \\ v_{97} & v_{98} & v_{99} \end{pmatrix} = \begin{pmatrix} f/\Delta s & 0 & 0 \\ 0 & f/\Delta s & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & t_1 \\ r_{21} & r_{22} & t_2 \\ r_{31} & r_{32} & t_3 \end{pmatrix}$$

$$\|\vec{r}_1\| = \|\vec{r}_2\| = 1$$

$$\vec{r}_3 = \vec{r}_1 \times \vec{r}_2$$

$$t_3 > 0$$

$$\vec{r}_3^\top \vec{t} \leq 0$$

⇒ Enough information to estimate 3D homography!



Augmented Reality Example

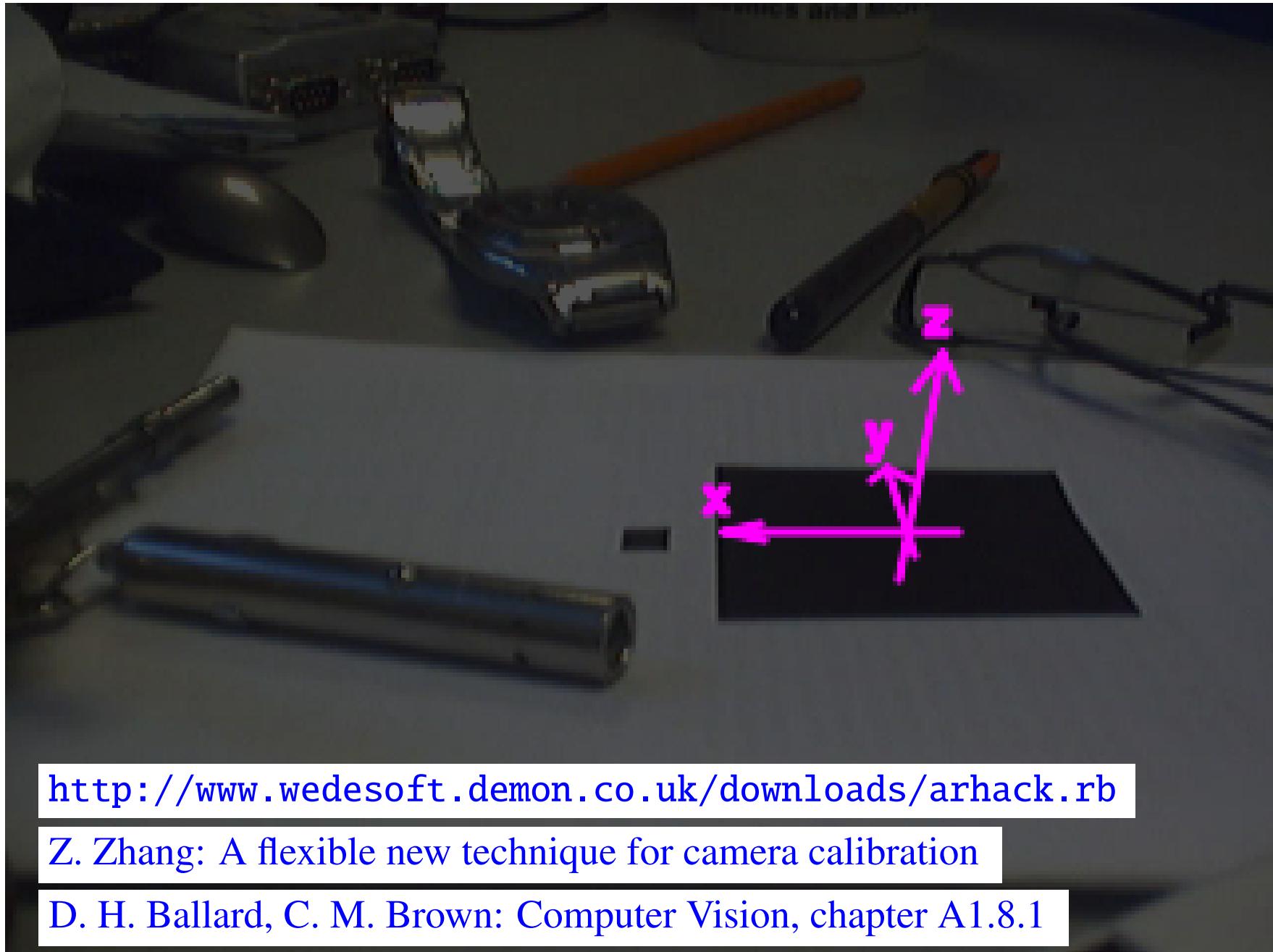
Source Code

```
constraints = []
for i in 0 ... 4 do
  constraints.push [ m[i].real, m[i].imag, 1.0, 0.0, 0.0, 0.0,
    -ms[i].real * m[i].real, -ms[i].real * m[i].imag, -ms[i].real ]
  constraints.push [ 0.0, 0.0, 0.0, m[i].real, m[i].imag, 1.0,
    -ms[i].imag * m[i].real, -ms[i].imag * m[i].imag, -ms[i].imag ]
end

h = Matrix[ *constraints ].svd[ 2 ].row( 8 ).reshape 3, 3
fs = 1.2 * 320 # focal length divided by pixel size
intr = Matrix[ [ fs, 0.0, 0.0 ], [ 0.0, fs, 0.0 ], [ 0.0, 0.0, 1.0 ] ]
rt = intr.inv * h
scale = 0.5 * ( rt.column( 0 ).norm + rt.column( 1 ).norm )
t = rt.column( 2 ) / scale
e1, e2 = rt.column( 0 ).normalise, e2 = rt.column( 1 ).normalise
e3 = e1.x e2
e1, e2, e3 = -e2, e1, -e3 if e3.inner_product( t ) > 0
extr = Matrix[ [ e1[0], e2[0], e3[0], t[0] ], [ e1[1], e2[1], e3[1], t[1] ],
  [ e1[2], e2[2], e3[2], t[2] ], [ 0.0, 0.0, 0.0, 1.0 ] ]
```



Augmented Reality Example Result



<http://www.wedesoft.demon.co.uk/downloads/arhack.rb>

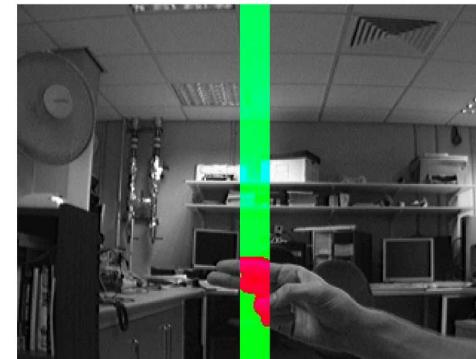
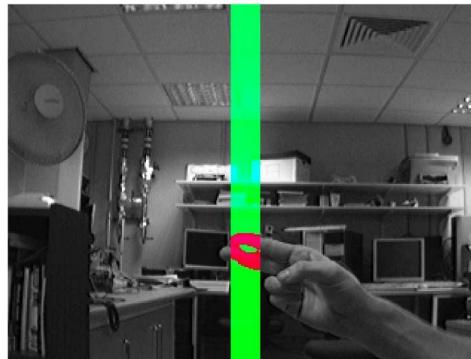
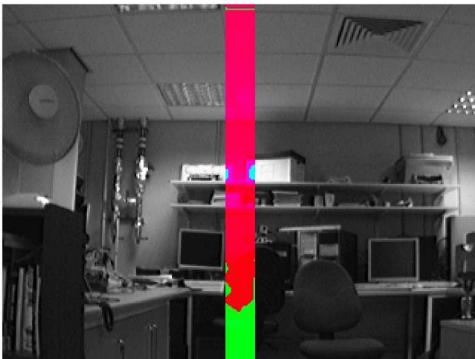
Z. Zhang: A flexible new technique for camera calibration

D. H. Ballard, C. M. Brown: Computer Vision, chapter A1.8.1

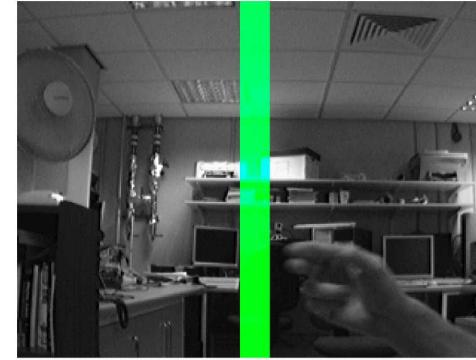


Presenter States

active



idle



timeout

↑
set
reference

choose
slide

<http://www.wedesoft.demon.co.uk/downloads/sensor.rb>



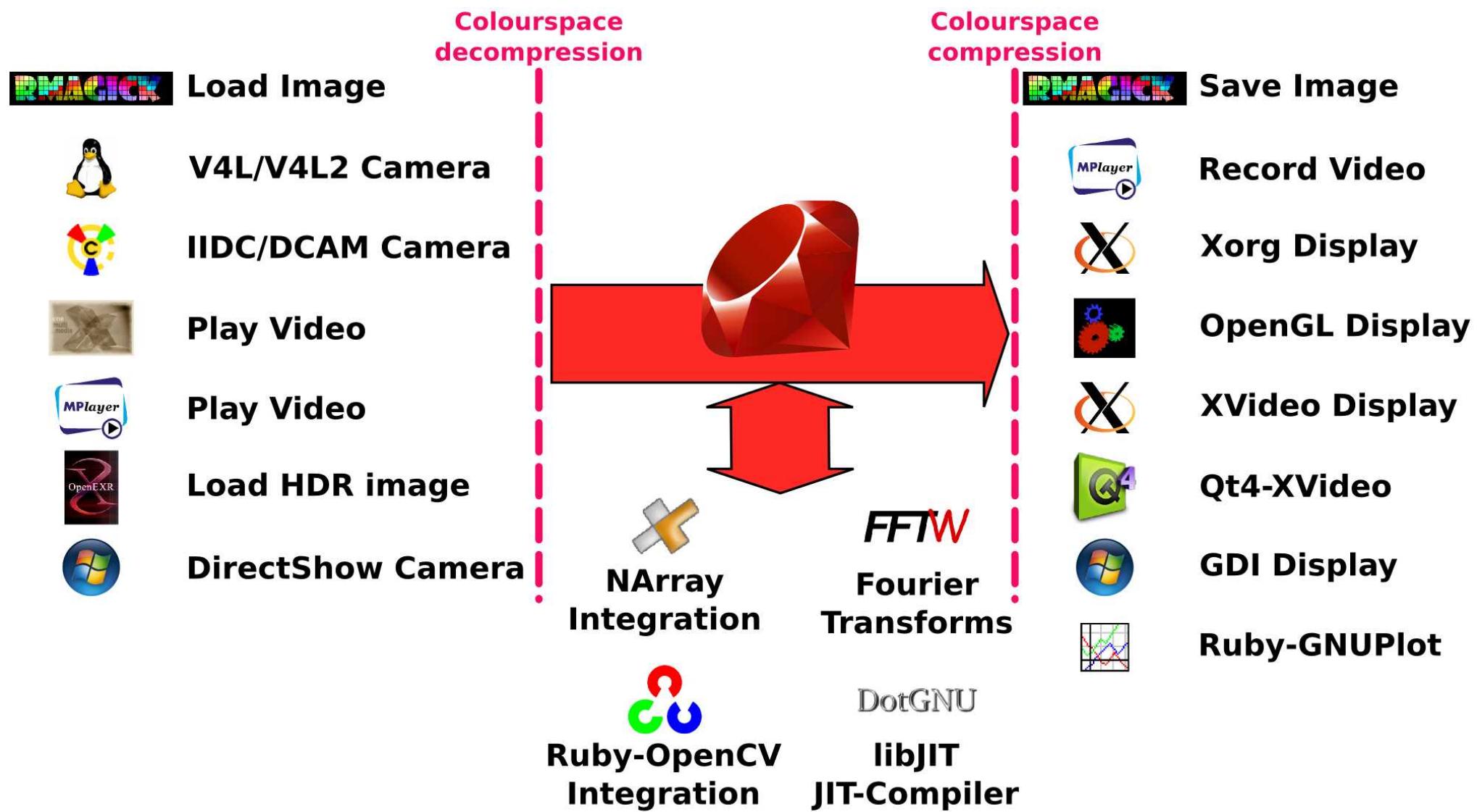
Presenter

Source Code (simplified)

```
input = DC1394Input.new
w, h, o = 20, input.height, input.width / 2 - 10
box = [ o ... o + w, 0 ... h ]
15.times { input.read }; bg = input.read_ubyte[ *box ]
ramp = Sequence.int( h ).indgen!.repmat( w ).roll
X11Display.show do
  img = input.read_ubytergb
  slice = ( img[ *box ].to_sint - bg ).gauss_blur( 2 ).abs >= 12
  n = slice.to_ubyte.sum
  if n > 20
    y = ramp.mask( slice ).sum / n
    print "#{ "%4d" % y }\\r" ; STDOUT.flush
  end
  img[ *box ].r = slice.to_ubyte * 255
  img[ *box ].g = slice.not.to_ubyte * 255
  img
end
```



Input/Output





Closures

Python + OpenCV

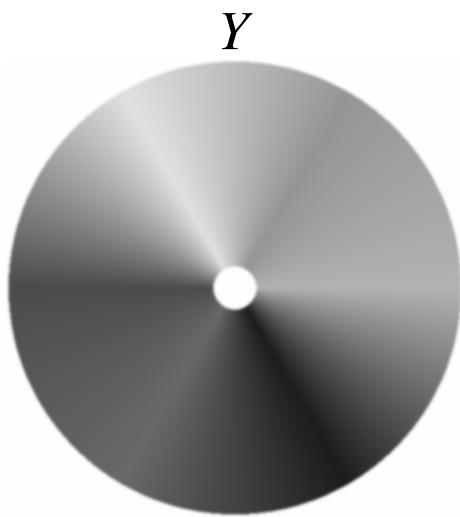
```
import sys
from opencv import cv
from opencv import highgui
highgui.cvNamedWindow( 'Camera' )
capture = highgui.cvCreateCameraCapture( -1 )
while 1:
    frame = highgui.cvQueryFrame( capture )
    gray = cv.cvCreateImage( cv.cvSize( frame.width, frame.height ), 8, 1 )
    cv.cvCvtColor( frame, gray, cv.CV_BGR2GRAY )
    highgui.cvShowImage( 'Camera', gray )
    if highgui.cvWaitKey( 5 ) > 0:
        break
```

Ruby + HornetsEye

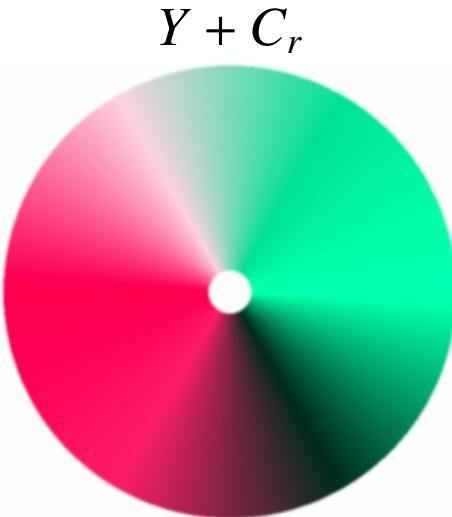
```
require 'hornetseye'
include Hornetseye
capture = V4L2Input.new
X11Display.show( :title => 'Camera' ) { capture.read.to_ubyte }
```



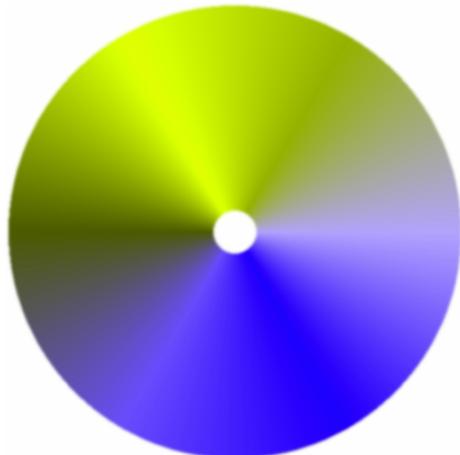
Colourspace Conversions



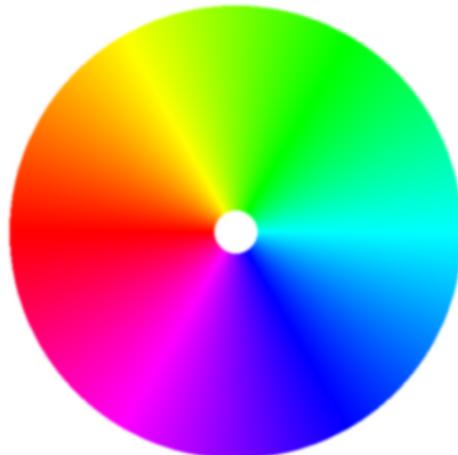
Y



$Y + C_b$



$Y + C_r + C_b$



Channel	Resolution
Y	Luminance
C_r	Chroma red
C_b	Chroma blue

See also: <http://fourcc.org/>

$$\begin{pmatrix} Y \\ C_b \\ C_r \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ -0.168736 & -0.331264 & 0.500 \\ 0.500 & -0.418688 & -0.081312 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} + \begin{pmatrix} 0 \\ 128 \\ 128 \end{pmatrix}$$



Lazy Computation

```
class Seq
  def Seq.new( *arr )
    unless Thread.current[ :lazy ]
      super *arr
    else
      Lazy.new( *arr ) { |x| x }
    end
  end
  def initialize( *arr )
    @arr = arr
  end
  def -@
    Seq.new *@arr.collect { |x| -x }
  end
end
```

```
class Lazy
  def initialize( *arr, &p )
    @arr, @p = arr, p
  end
  def demand
    Seq.new *@arr.collect { |i| @p.call i }
  end
  def -@
    Lazy.new( *@arr ) { |i| -@p.call( i ) }
  end
  def lazy
    previous = Thread.current[ :lazy ]
    Thread.current[ :lazy ] = true
    retval = yield
    Thread.current[ :lazy ] = previous
    retval.demand
  end
```



Credits

Credits

Aiden Lockwood, Aleksey Demakov, Annemie Wedekind, Balasundram Amavasai, Beverly Inkson, Chinwe Lucy Ozoegwu, Damien Douxchamps, Daniel Martín Marín, Géraud De La Mensbruge, Gerhard Wedekind, Hussein Abdul-Rahman, Jing Jing Wang, Jon Travis, Jong Peng, Juan Roldan, Julien Demarest, Julien Faucher, Julien Lacheray, Ken Dutton, Kim Chuan Lim, Kirill Kononenko, Klaus Treichel, Manuel Boissenin, Martin Howarth, Matthias Stumpf, Michael Doronin, Ralph Gay, Richard Dale, Sonia Fernández Rodríguez, Tan Kang Song, Ushakiran Soutapalli, Volkan Karaca, Warren Jasper, Zineb Saghi, ...

<http://www.wedesoft.demon.co.uk/hornetseye-api/>

<http://rubyforge.org/projects/hornetseye/>

<http://sourceforge.net/projects/hornetseye/>

<http://launchpad.net/hornetseye/>

<http://raa.ruby-lang.org/project/hornetseye/>

<http://www.ohloh.net/p/hornetseye/>