SU2-6: Video Tracking

SU2-6(4): A Machine Vision Extension for the Ruby Programming Language

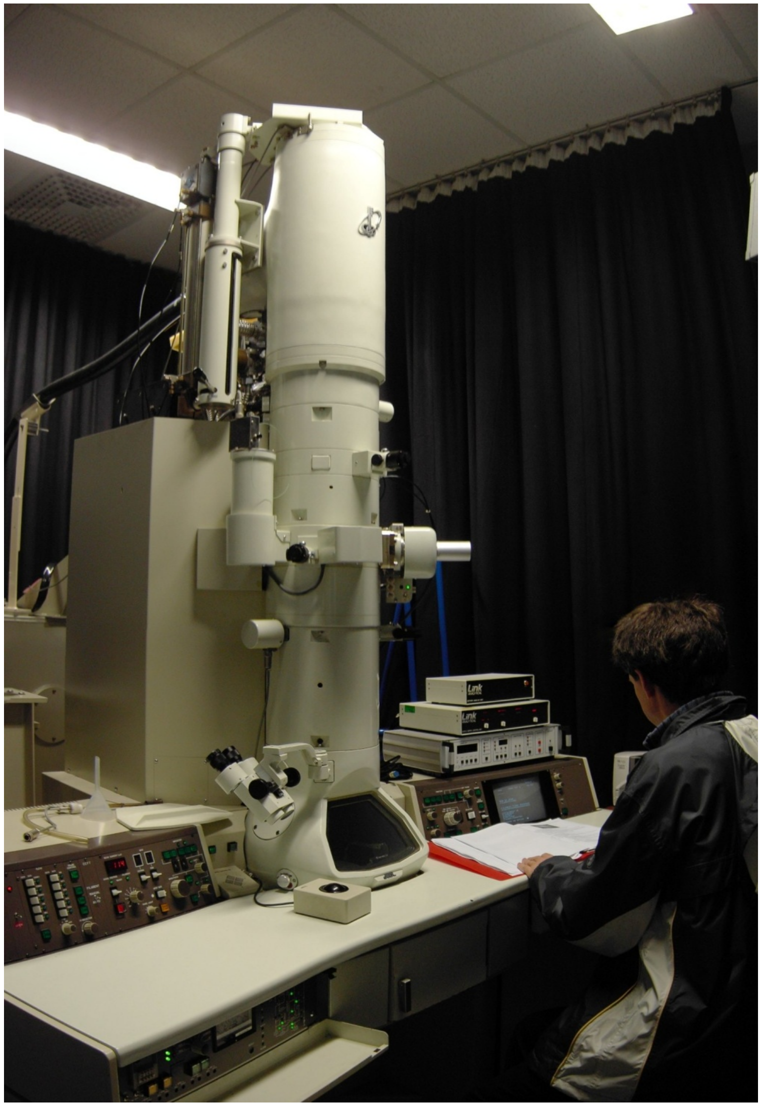
J. Wedekind, B. P. Amavasai, K. Dutton, M. Boissenin

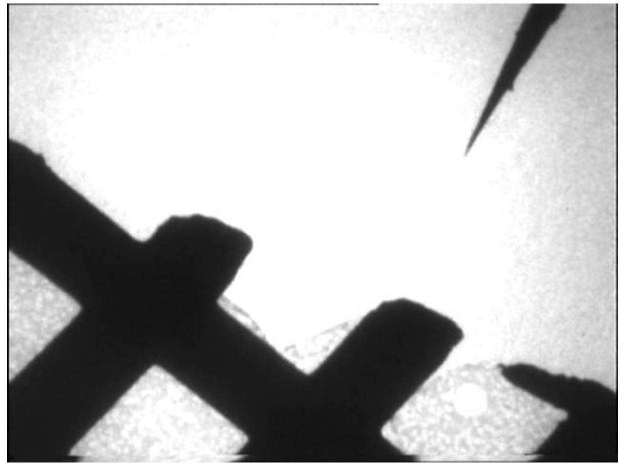
Sunday, June 22th 2008

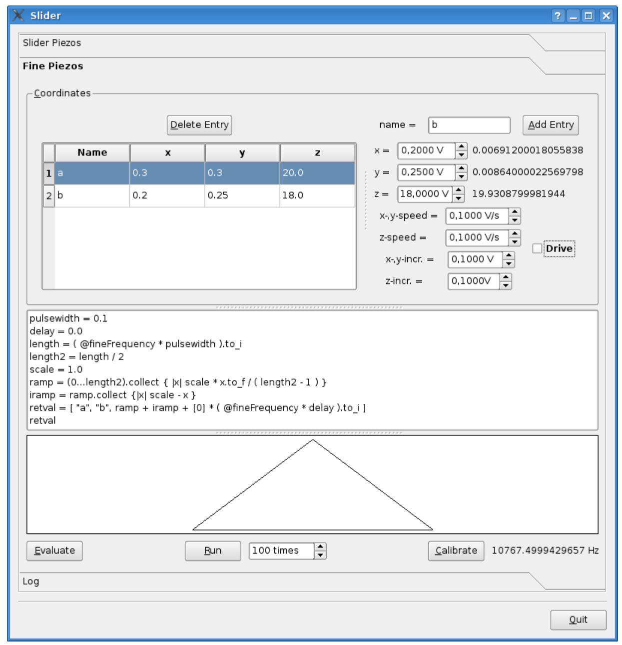
Microsystems and Machine Vision Laboratory
Materials Engineering Research Institute
Sheffield Hallam University
Sheffield
United Kingdom

MATERIALS AND ENGINEERING RESEARCH INSTITUTE

Introduction
EPSRC Nanorobotics grant







EPSRC

electron microscopy

- telemanipulation
- drift-compensation
- closed-loop control

computer vision

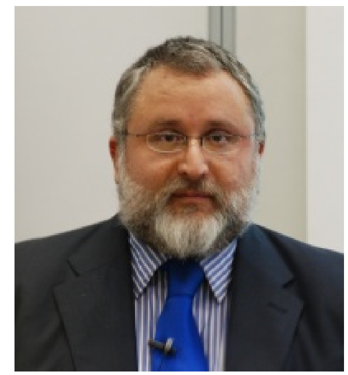
- real-time software
- system integration
- theoretical insights

Sheffield Hallam University

Introduction
GPLv3 free software license

four freedoms (Richard Stallman)

1. The **freedom to run** the program, for any purpose.
2. The **freedom to study** how the program works, and adapt it to your needs.
3. The **freedom to redistribute** copies so you can help your neighbor.
4. The **freedom to** improve the program, and **release your improvements** to the public, so that the whole community benefits.



respect the freedom of downstream users (Richard Stallman)

GPL requires derived works to be available under the same license.

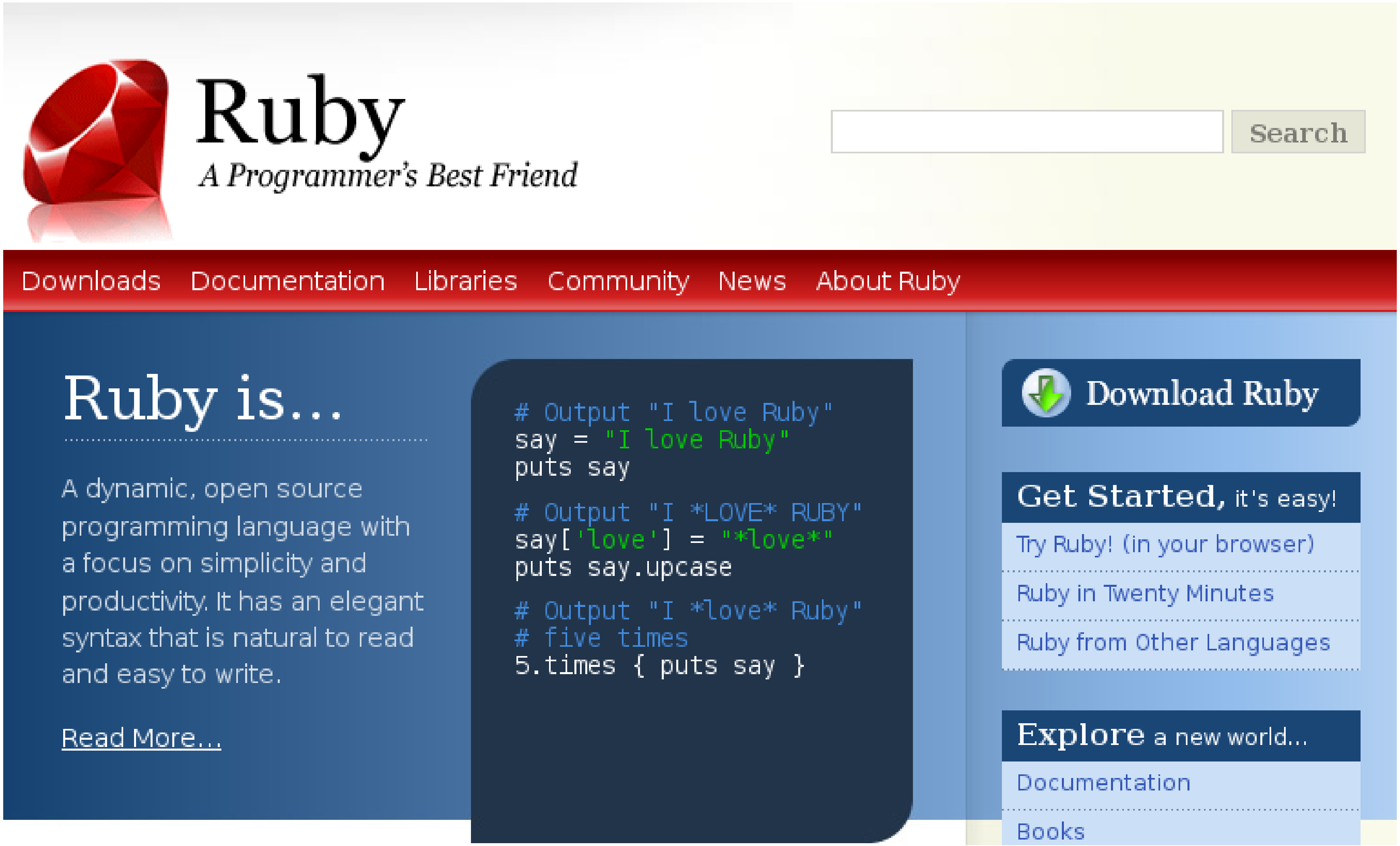
covenant not to assert patent claims (Eben Moglen)

GPLv3 deters users of the program from instituting patent ligitation by the threat of withdrawing further rights to use the program.

other (Eben Moglen)

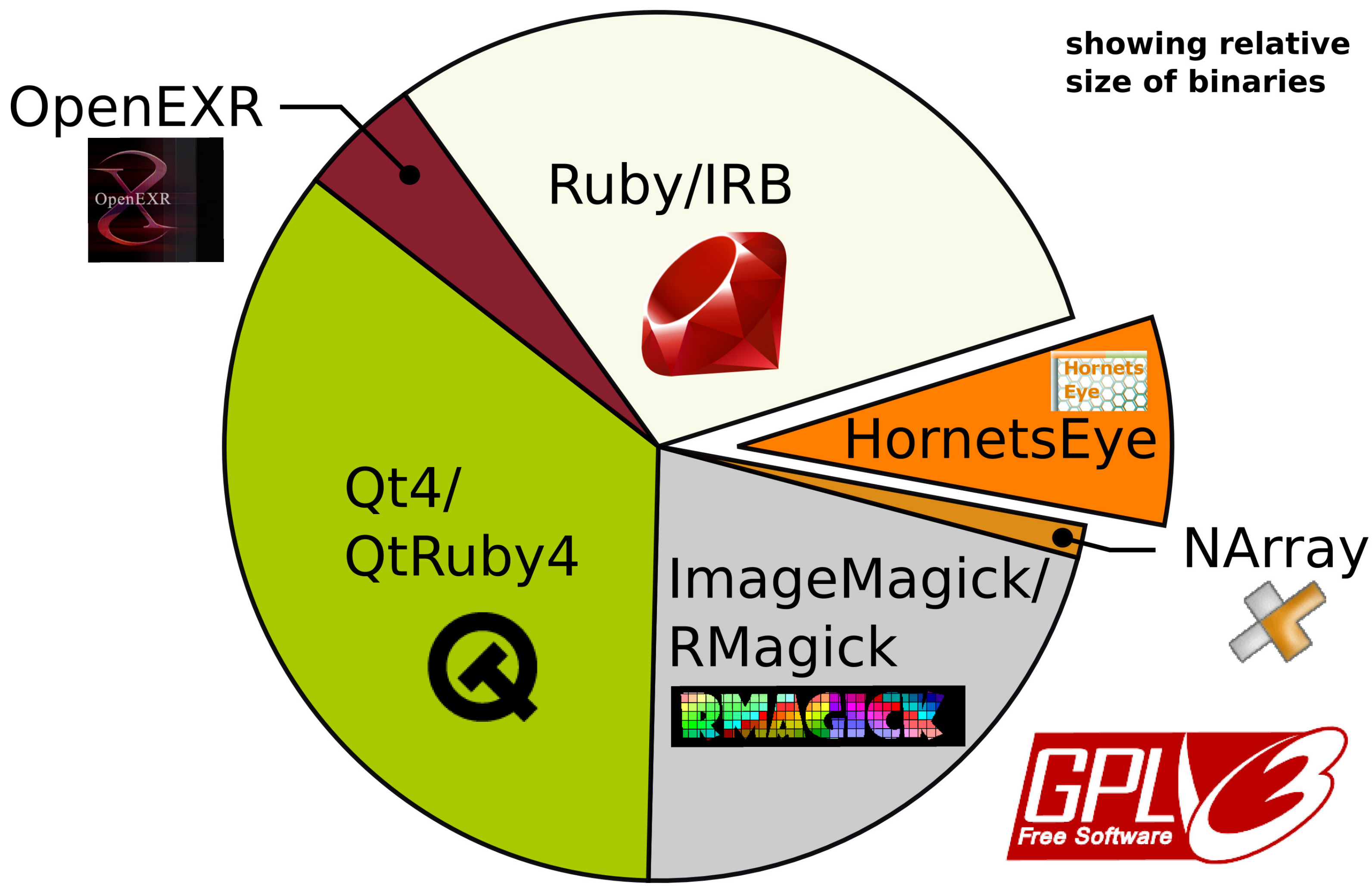
GPLv3 has regulations against DMCA restrictions and tivoization.

Introduction
Ruby programming language

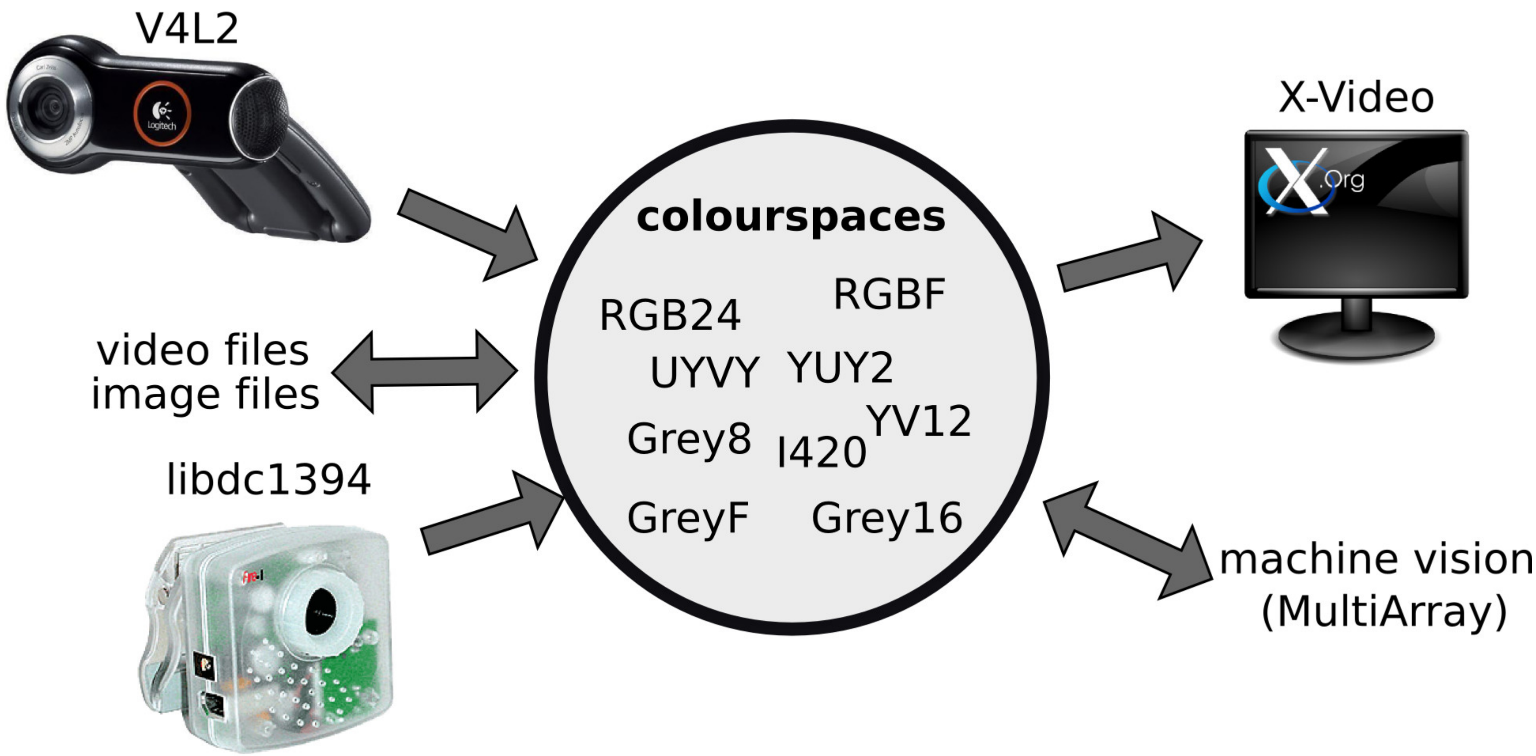


http://www.ruby-lang.org/

HornetsEye
Free software code base



HornetsEye
I/O integration



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

also see: `http://fourcc.org/`

HornetsEye
Multi-dimensional arrays

$$g, h \in \{0, 1, \ldots, w-1\} \times \{0, 1, \ldots, h-1\} \to \mathbb{R}$$

$$h\left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}\right) = g\left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}\right)/2$$

Ruby

MultiArray.dfloat
Fixnum

h = g / 2

MultiArray.respond_to?("binary_div_lint_dfloat")

no

h = g.collect { |x| x / 2 }

yes

MultiArray.dfloat(320, 240):
[[245.0, 244.0, 197.0, ...],
[245.0, 247.0, 197.0, ...],
[247.0, 248.0, 187.0, ...]
...

[3.141]

Array.pack("D")

String.unpack("D")

C++

```
for ( int i=0; i<n; i++ )
  *r++ = *p++ / q;
```

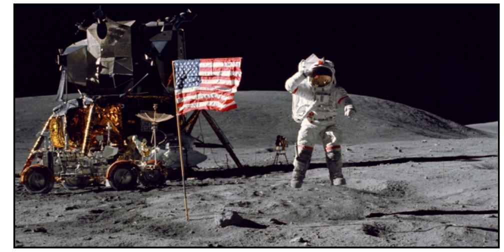
MultiArray.binary_div_byte_byte
MultiArray.binary_div_byte_bytergb
MultiArray.binary_div_byte_dcomplex
MultiArray.binary_div_byte_dfloat
MultiArray.binary_div_byte_dfloatrgb
...

"\x54\xE3\xA5\x9B\xC4\x20\x09\x40"

Generic operations
Linear shift-invariant filters

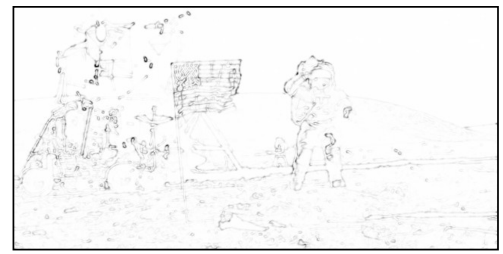
Common code

```
grad_sigma, cov_sigma = 1.0, 1.0
x, y = img.gauss_gradient_x( grad_sigma ), img.gaussgradient_y( grad_sigma )
a = ( x ** 2 ).gauss_blur( cov_sigma )
b = ( y ** 2 ).gauss_blur( cov_sigma )
c = ( x * y ).gauss_blur( cov_sigma )
tr = a + b
det = a * b - c * c
```



Yang et al.

```
noise = 1.0
g = ( ( a - b ) ** 2 + ( 2 * c ) ** 2 ) / ( a + b + noise ** 2 ) ** 2
result = ( g.normalise( 1.0..0.0 ) ** m * ( x ** 2 + y ** 2 ) )
```

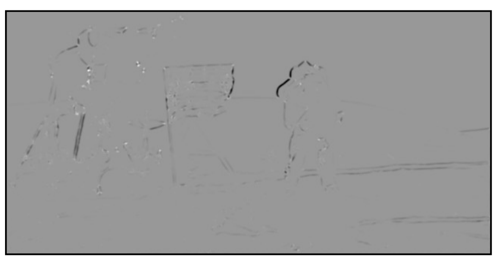


Kanade-Lucas-Tomasi

```
dissqrt = ( tr ** 2 - det * 4 ).major( 0.0 ).sqrt
result = 0.5 * ( tr - dissqrt )
```

Harris-Stephens

```
k = 0.1
result = det - tr * tr * k
```



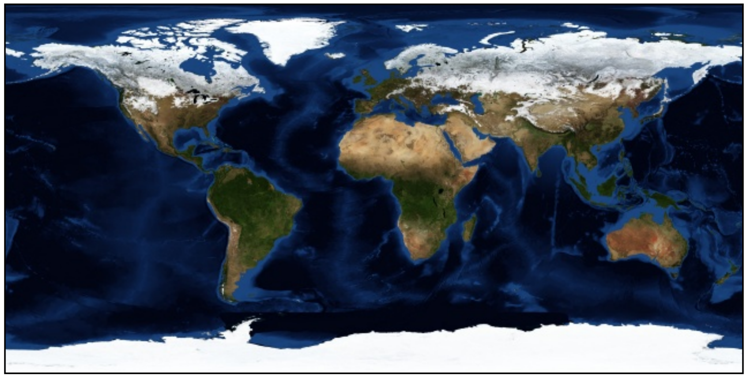
MultiArray.correlate_byte_byte
MultiArray.correlate_byte_bytergb
MultiArray.correlate_byte_dcomplex
MultiArray.correlate_byte_dfloat
MultiArray.correlate_byte_dfloatrb
...

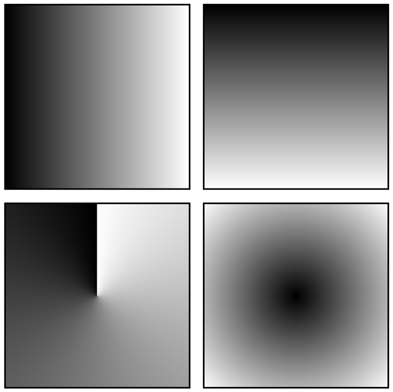
Generic operations
Warps

$$g \in \{0,1,\ldots,w-1\} \times \{0,1,\ldots,h-1\} \rightarrow \mathbb{R}^3$$
$$h \in \{0,1,\ldots,w'-1\} \times \{0,1,\ldots,h'-1\} \rightarrow \mathbb{R}^3$$
$$W \in \{0,1,\ldots,w'-1\} \times \{0,1,\ldots,h'-1\} \rightarrow \mathbb{Z}^2$$

$$h(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}) = \begin{cases} g(W(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix})) & \text{if } W(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}) \in \{0,1,\ldots,w-1\} \times \{0,1,\ldots,h-1\} \\ 0 & \text{otherwise} \end{cases}$$

```
class MultiArray
  def MultiArray.ramp1( *shape )
    retval = MultiArray.new( MultiArray::LINT, *shape )
    for x in 0...shape[0]
      retval[ x, 0...shape[1] ] = x
    end
    retval
  end
  # ...
end
img = MultiArray.load_rgb24( ”test.jpg” )
w, h = *img.shape; c = 0.5 * h
x, y = MultiArray.ramp1( h, h ), MultiArray.ramp2( h, h )
warp = MultiArray.new( MultiArray::LINT, h, h, 2 )
warp[ 0...h, 0...h, 0 ], warp[ 0...h, 0...h, 1 ] =
    ( ( ( x - c ).atan2( y - c ) / Math::PI + 1 ) * w / 2 - 0.5 ),
    ( ( x - c ) ** 2 + ( y - c ) ** 2 ).sqrt
img.warp_clipped( warp ).display
```





Generic operations
Affine transforms

```
class MultiArray
  def MultiArray.ramp1( *shape )
    retval = MultiArray.new( MultiArray::LINT, *shape )
    for x in 0...shape[0]
      retval[ x, 0...shape[1] ] = x
    end
    retval
  end
  # ...
end
img = MultiArray.load_rgb24( "test.jpg" )
w, h = *img.shape
v = Vector[ MultiArray.ramp1( w, h ) - w / 2,
            MultiArray.ramp2( w, h ) - h / 2 ]
angle = 30.0 * Math::PI / 180.0
m = Matrix[ [ Math::cos( angle ), -Math::sin( angle ) ],
            [ Math::sin( angle ), Math::cos( angle ) ] ]
warp = MultiArray.new( MultiArray::LINT, w, h, 2 )
warp[ 0...w, 0...h, 0 ], warp[ 0...w, 0...h, 1 ] =
  ( m * v )[0] + w / 2, ( m * v )[1] + h / 2
img.warp_clipped( warp ).display
```

$$W_\alpha\left(\begin{pmatrix}x_1\\x_2\end{pmatrix}\right)=\begin{pmatrix}\cos(\alpha) & -\sin(\alpha)\\ \sin(\alpha) & \cos(\alpha)\end{pmatrix}\begin{pmatrix}x_1\\x_2\end{pmatrix}$$

Application
Inverse compositional Lucas-Kanade

given: template T, image I, previous pose $\vec{p}$

sought: pose-change $\Delta\vec{p}$

$$\underset{\Delta\vec{p}}{\operatorname{argmin}} \int_{\vec{x}\in T} \|T(\vec{x}) - I(W_{\vec{p}}^{-1}(W_{\Delta\vec{p}}^{-1}(\vec{x})))\|^2 \mathrm{d}\vec{x} = (*)$$

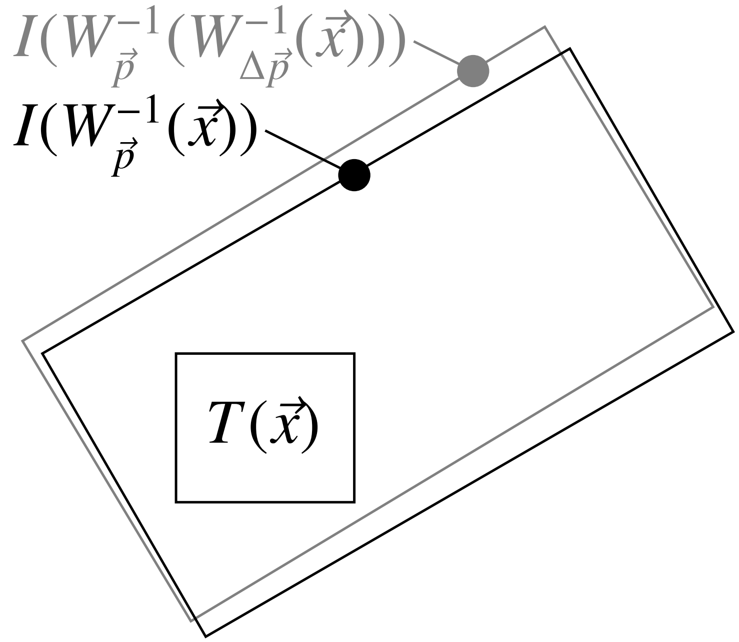
(1) $T(\vec{x}) - I(W_{\vec{p}}^{-1}(W_{\Delta\vec{p}}^{-1}(\vec{x}))) = T(W_{\Delta\vec{p}}(\vec{x})) - I(W_{\vec{p}}^{-1}(\vec{x}))$

(2) $T(W_{\Delta\vec{p}}(\vec{x})) \approx T(\vec{x}) + \left(\frac{\delta T}{\delta\vec{x}}(\vec{x})\right)^T \cdot \left(\frac{\delta W_{\vec{p}}}{\delta\vec{p}}(\vec{x})\right) \cdot \Delta\vec{p}$

$$(*) \overset{(1,2)}{=} \underset{\vec{p}}{\operatorname{argmin}}(\|\mathcal{H}\,\vec{p} + \vec{b}\|^2) = (\mathcal{H}^T\,\mathcal{H})^{-1}\,\mathcal{H}^T\,\vec{b}$$

where $\mathcal{H} = \begin{pmatrix} h_{1,1} & h_{1,2} & \cdots \\ h_{2,1} & h_{2,2} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix}$ and $\vec{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \end{pmatrix}$

$$h_{i,j} = \left(\frac{\delta T}{\delta\vec{x}}(\vec{x}_i)\right)^T \cdot \left(\frac{\delta W_{\vec{p}}}{\delta p_j}(\vec{x}_i)\right),\ b_i = T(\vec{x}_i) - I(W_{\vec{p}}^{-1}(\vec{x}_i))$$



S. Baker and I. Matthew: “Lucas-Kanade 20 years on: a unifying framework”

http://www.ri.cmu.edu/projects/project_515.html

Applications
Lucas-Kanade core

initialisation

```
p = Vector[ xshift, yshift, rotation ]
w, h, sigma = tpl.shape[0], tpl.shape[1], 5.0
x, y = xramp( w, h ), yramp( w, h )
gx = tpl.gauss_gradient_x( sigma )
gy = tpl.gauss_gradient_y( sigma )
c = Matrix[ [ 1, 0 ], [ 0, 1 ], [ -y, x ] ] * Vector[ gx, gy ]
hs = ( c * c.covector ).collect { |e| e.sum }
```

tracking

```
field = MultiArray.new( MultiArray::SFLOAT, w, h, 2 )
field[ 0...w, 0...h, 0 ] = x * cos( p[2] ) - y * sin( p[2] ) + p[0]
field[ 0...w, 0...h, 1 ] = x * sin( p[2] ) + y * cos( p[2] ) + p[1]
diff = img.warp_clipped_interpolate( field ) - tpl
s = c.collect { |e| ( e * diff ).sum }
d = hs.inverse * s
p += Matrix[ [  cos(p[2]), -sin(p[2]), 0 ],
             [  sin(p[2]),  cos(p[2]), 0 ],
             [          0,          0, 1 ] ] * d
```

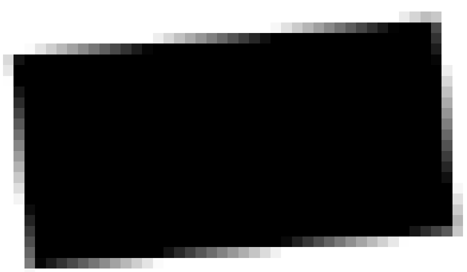
MATERIALS AND ENGINEERING RESEARCH INSTITUTE

Application
Lucas-Kanade implementation details

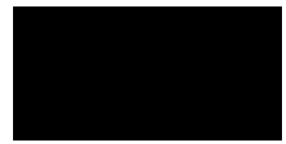
Interpolation

without interpolation

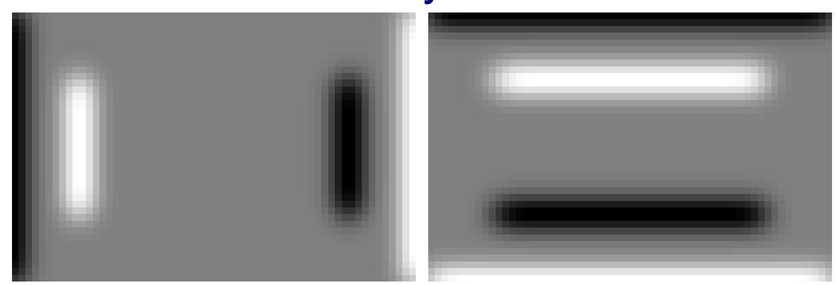
with interpolation



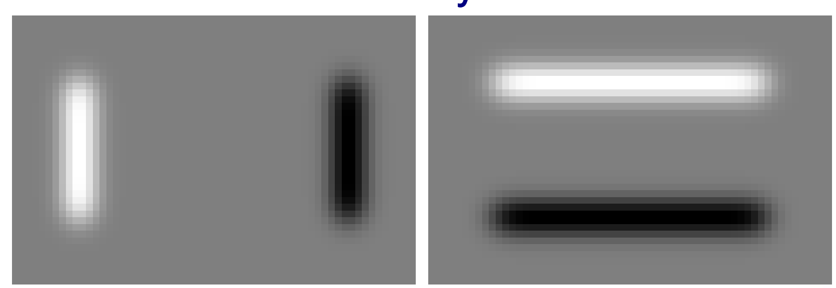
Gradient boundary



boundary effects



no boundary effects



Sheffield Hallam University

conclusion

- concise implementation of Lucas-Kanade
- native blocks of code in C++, Ruby as glue-code
- development platform for general purpose machine vision

future work

- microscopy software
- wavelets, feature extraction
- feature-based object tracking & recognition

free software

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

`http://rubyforge.org/projects/hornetseye/`

`http://sourceforge.net/projects/hornetseye/`