An Image Algebra Library for SAC

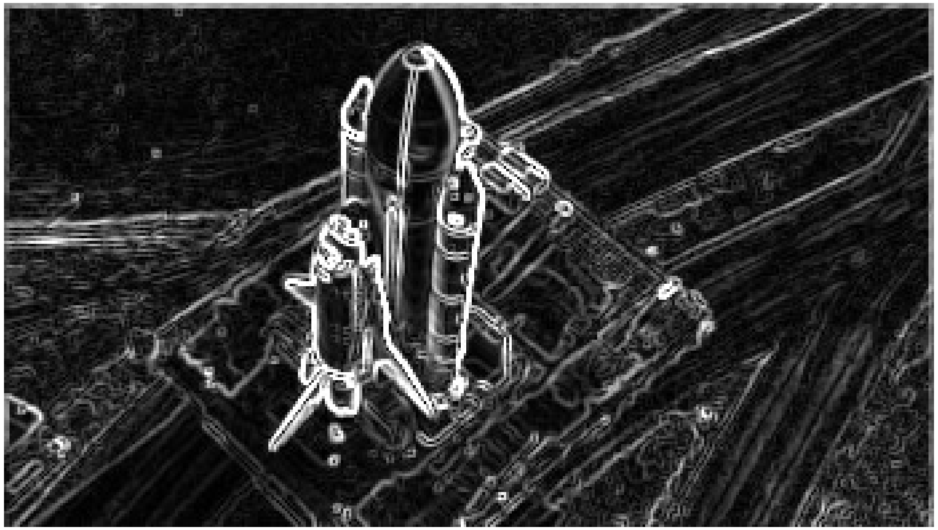
Rumyana Rumenova

Supervisor: Sven-Bodo Scholz

Image Processing

- Transformation and analysis of images.
- Applications in medical research and security.
- Common operations for a variety of tasks.
- Formally described by Ritter's Image Algebra

Example: Edge Detection



High Productivity + High Performance

- Image Algebra describes algorithms concisely

- It can be adapted into domain-specific languages:
  - Simpler, more organised implementation
  - Can be used with highly specialised hardware

High Productivity + High Performance

- The idea behind the Rathlin project's RIPL
  - High-level language with domain specific notation + High performance data-parallel backend
- The philosophy of SAC.
- Better hardware demands smarter software...

Smarter Software?

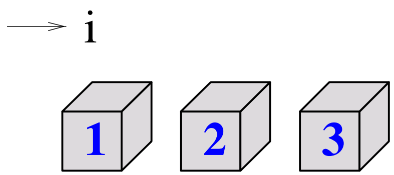
- Can be achieved with a smarter compiler.
- Powerful compiler optimisations.
  - Me: I'll trick you and put this in a loop

```
a = read( “image.bmp”)
b = transpose( a);
c = transpose( b);
write( c);
```

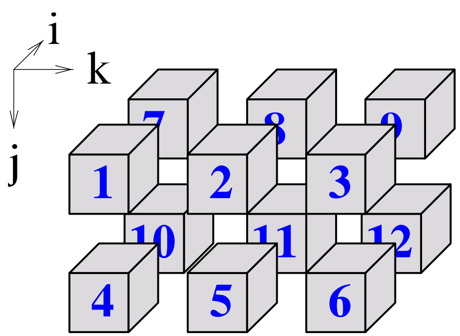
  - Compiler: No I will not do this over and over again

Array programming

- Evertyhing in SAC is an array. Even scalars.



shape vector: [3]
data vector: [1, 2, 3]



shape vector: [2, 2, 3]
data vector: [1, 2, 3, ..., 11, 12]

42

shape vector: [ ]
data vector: [42]

Array programming

- Inherent array structure of images
- Efficient with n-dimensional arrays

- Simple to program
  - Shape-invariant programming:
    - A 10x10 rgb image is really a 10x10x3 array...
    - But the same code for rgb and greyscale images

Array programming

- Simple to program
  - Flexible index ranges:

```
a = with {
    ( . < iv < .): 1;
    } : genarray( [3,4], 0);
```

0	0	0	0
0	1	1	0
0	0	0	0

```
a = with {
   ( . < iv < .): [1,1,1];
   } : genarray( [3,4], [0,0,0]);
```

[0,0,0]	[0,0,0]	[0,0,0]	[0,0,0]
[0,0,0]	[1,1,1]	[1,1,1]	[0,0,0]
[0,0,0]	[0,0,0]	[0,0,0]	[0,0,0]

Image Algebra Concepts

- Formal mathematical description of concepts and operations

points	[int, int] to represent (x,y)
values	greyscale – [int] rgb – [int, int, int]
images/templates	arrays of greyscale/rgb/int

Left Morphological Max Operator

$$\mathbf{a} \boxed{\vee} \mathbf{t} = \left\{ (\mathbf{y}, \mathbf{b}(\mathbf{y})) \,:\, \mathbf{b}(\mathbf{y}) = \bigvee_{\mathbf{x} \in \mathbf{X}} [\mathbf{a}(\mathbf{x}) + \mathbf{t}_{\mathbf{y}}(\mathbf{x})],\ \mathbf{y} \in \mathbf{Y} \right\}$$

Definition: Let $\mathbb{F}$ be a value set and $\mathbf{X}$ a point set. An $\mathbb{F}$-*valued image on* $\mathbf{X}$ is any element of $\mathbb{F}^{\mathbf{X}}$. Given an $\mathbb{F}$–valued image $\mathbf{a} \in \mathbb{F}^{\mathbf{X}}$

Left Morphological Max Operator

$$\mathbf{a} \boxed{\vee} \mathbf{t} = \left\{ (\mathbf{y}, \mathbf{b}(\mathbf{y})) \;:\; \mathbf{b}(\mathbf{y}) = \bigvee_{\mathbf{x} \in \mathbf{X}} [\mathbf{a}(\mathbf{x}) + \mathbf{t}_{\mathbf{y}}(\mathbf{x})],\; \mathbf{y} \in \mathbf{Y} \right\}$$

Definition. A *template* is an image whose pixel values are images (functions). In particular, an $\mathbb{F}$-*valued template from* $\mathbf{Y}$ *to* $\mathbf{X}$ is a function $\mathbf{t} : \mathbf{Y} \to \mathbb{F}^{\mathbf{X}}$. Thus, $\mathbf{t} \in \left(\mathbb{F}^{\mathbf{X}}\right)^{\mathbf{Y}}$ and $\mathbf{t}$ is an $\mathbb{F}^{\mathbf{X}}$-valued image on $\mathbf{Y}$.

For notational convenience we define $\mathbf{t}_{\mathbf{y}} \equiv \mathbf{t}(\mathbf{y})\ \forall \mathbf{y} \in \mathbf{Y}$. The image $\mathbf{t}_{\mathbf{y}}$ has representation

$$\mathbf{t}_{\mathbf{y}} = \{(\mathbf{x}, \mathbf{t}_{\mathbf{y}}(\mathbf{x})) \;:\; \mathbf{x} \in \mathbf{X}\}\,.$$

Left Morphological Max Operator

$$\mathbf{a} \boxed{\vee} \mathbf{t} = \left\{ (\mathbf{y}, \mathbf{b}(\mathbf{y})) \,:\, \mathbf{b}(\mathbf{y}) = \bigvee_{\mathbf{x} \in \mathbf{X}} [\mathbf{a}(\mathbf{x}) + \mathbf{t}_{\mathbf{y}}(\mathbf{x})],\ \mathbf{y} \in \mathbf{Y} \right\}$$

```
value[+] max_convolution( value[+] img, value[+] t)
{
 b = with {
    ( . <= iv <= .): max( img[iv] + t);
     } : genarray( shape( img), default_el( img));
 return b;
}
```

Maximum convolution

Transpose

Value Recalculation

Prewitt Edge Detection

The edge enhanced image $\mathbf{b} \in \mathbb{R}^{\mathbf{Y}}$ is given by

$$\mathbf{b} := \left(\left[(\mathbf{a} \oplus \mathbf{s})^2 + (\mathbf{a} \oplus \mathbf{t})^2 \right]^{1/2} \right) .$$

$\mathbf{s}$ =

-1	-1	-1
1	1	1

$\mathbf{t}$ =

-1		1
-1		1
-1		1

Prewitt Edge Detection

The edge enhanced image $\mathbf{b} \in \mathbb{R}^{\mathbf{Y}}$ is given by

$$\mathbf{b} := \left(\left[(\mathbf{a} \oplus \mathbf{s})^2 + (\mathbf{a} \oplus \mathbf{t})^2 \right]^{1/2} \right) .$$

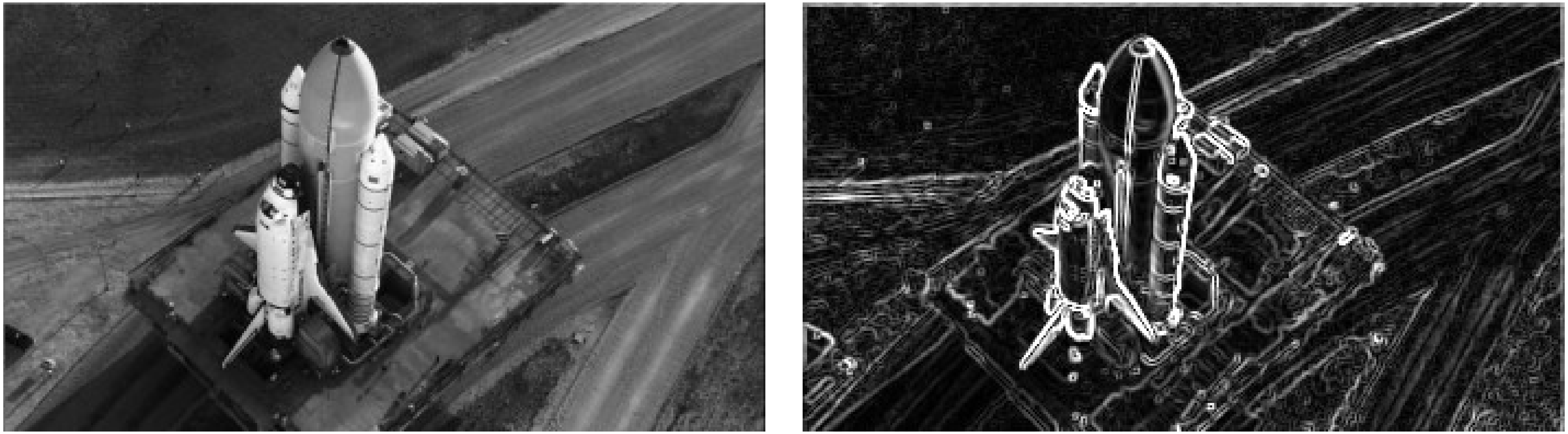
```
s = (greyscale[+]) [ [-1,-1,-1], [0,0,0], [1,1,1] ];
t = (greyscale[+]) [ [-1,0,1], [-1,0,1], [-1,0,1] ];

s_conv = convolution_2d( a, s);
t_conv = convolution_2d( a, t);

as = pow( s_conv, 2);
at = pow( t_conv, 2);

b = sqrt( as + at);
```

Prewitt Edge Detection



Efficiency

- Inline functions:
  - Allow for sophisticated compiler optimisations
  - Reduce memory access

Test results

	Characteristics on 348x196 greyscale image	
	No-inline	Inline (optimised)
1 core	0.42 millis	0.45 millis
10 cores	0.04 millis	0.06 millis
	Prewitt Edge Detection on 1920x1080 greyscale image	
	No-inline	Inline (optimised)
1 core	150 millis	84.8 millis
10 cores	16.5 millis	9.3 millis