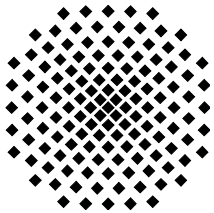
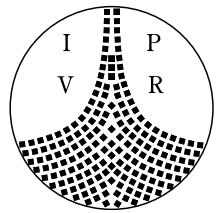




Universität Stuttgart

Institute for Parallel and Distributed
High Performance Systems (IPVR)

Breitwiesenstraße 20 – 22
D-70565 Stuttgart

Data-Parallel Image Processing in Parallaxis

Thomas Bräunl

Dieses Dokument wurde erstellt mit FrameMaker 4.0.2.

Overview

1. Parallaxis-III Language

2. Data-Parallel Problems

3. Parallel Image Processing
   - Point Operators
   - Local Operators
   - Global Operators

1. Parallaxis-III

Data-Parallel Programming Language Parallaxis

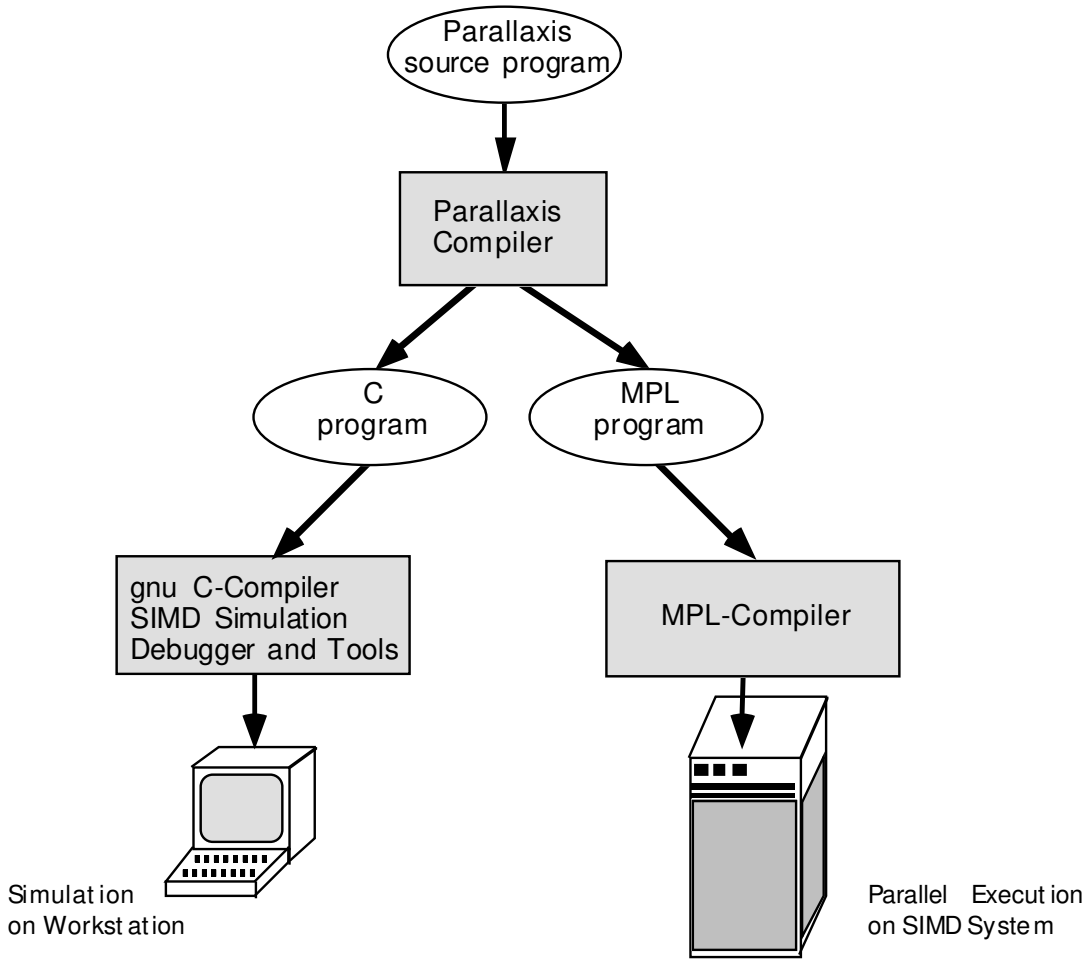
History

- language definition by Thomas Bräunl in 1989
- programming environment is distributed as public domain software
- Parallaxis is widely used at universities for teaching data-parallel programming concepts
- current version Parallaxis-III (1995)

Key Features

- **structured programming** (based on sequential Modula-2)
- **machine independent**
- program comprises parallel algorithm plus PE- and connection-specification
- simulation system for single processor systems
- parallel system on MasPar and Connection Machine
- source level debugger with visualization tools: PE data, PE activity, and PE load

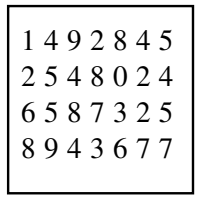
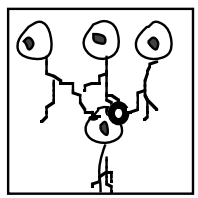
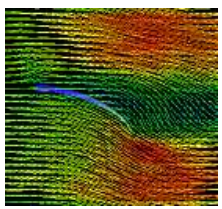
Compiler


Parallaxis
source program
Parallaxis
Compiler
C
program
MPL
program
gnu C-Compiler
SIMD Simulation
Debugger and Tools
MPL-Compiler
Simulation
on Workstation
Parallel Execution
on SIMD System

2. Data Parallel Problems

- **SIMD is suffcient** for most compute-intense problems
- SIMD requires different approach to MIMD programming
- **Data exchange is more efficient for SIMD**
- efficiency is important, *but*
- PE utilization is not outmost goal
- natural design of algorithms
- **simple programs**
- readable programs

Which application areas qualify ?

	Teacher's view	Students' view
 Numeric Algorithms sorting, searching, linear equations	*Too theoretical, boring!*	*Too theoretical, boring!*
 Neural Networks and Genetic Algorithms	*Challenging, interesting!*	*Interesting, but requires much work!*
 Simulation	*Detailed knowledge required for each simulation!*	*Interesting, different subjects!*
Image Processing	*Interesting, rewarding, reasonable chunks*	*Interesting, single subject, early results, cool!*

3. Parallel Image Processing

- Point Operators
- Local Operators
- Global Operatos
- Histograms
- Edge Detection
- Edge Thinning
- Corner Detecion
- Regions
- Hough Transform
- Fast Fourier Transform
- Textures with Cooccurance
- Stereo
- Motion

Point operators use data of their own pixel only, no data exchange is required.

Image Data Types

```
 1  TYPE binary = BOOLEAN;
 2       gray   = [0..255];
 3       color  = RECORD
 4                  red, green, blue: gray
 5                END;
 6
 7  CONST b_black =  TRUE;
 8        b_white = FALSE;
 9        g_black =     0;
10        g_white =   255;
11        c_black = color(  0,  0,  0);
12        c_white = color(255,255,255);
```

Typical Procedure Head:

```
PROCEDURE xyz(img: VECTOR OF gray): VECTOR OF gray;
```

Example Routines for File-I/O and X-Window Output:

`read_c_image, read_g_image, read_b_image, write_c_image, write_g_image, write_b_image, OpenWindow, SetArea, SetLine, SetColumn, CloseWindow`

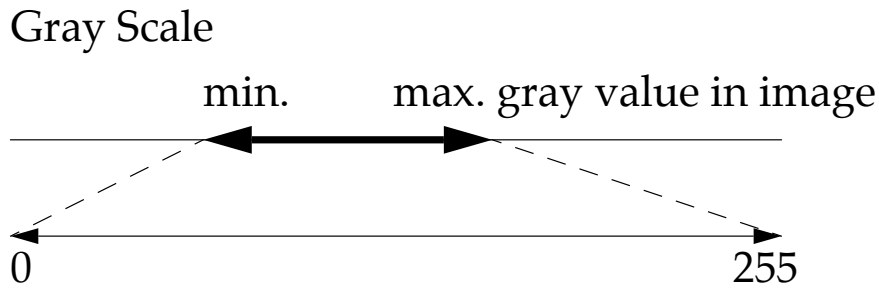
3.1.1 Type Conversions

Example: Convert gray [0..255] to binary [TRUE, FALSE] using a threshold

```
1  PROCEDURE gray2binary(img:VECTOR OF gray; threshold: gray): VECTOR OF binary;
2  BEGIN  (* remember: b_black = TRUE *)
3    RETURN img < threshold
4  END gray2binary;
```

3.1.2 Stretch Gray Value Range

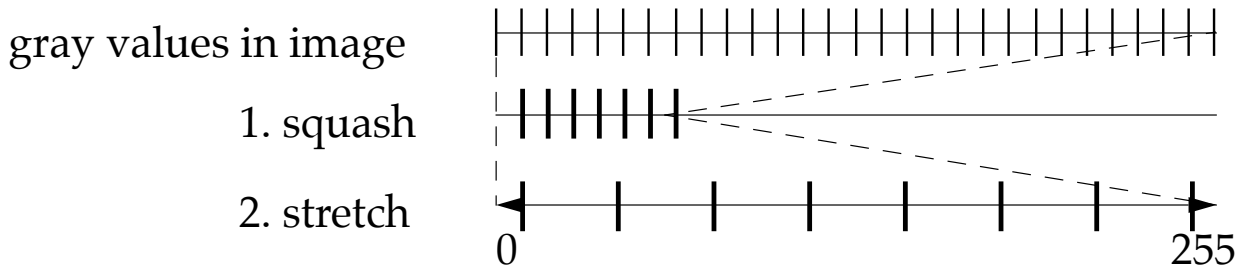
Extend the range of gray values in an image, in order to make details visible
e.g. in very dark (or very bright) medical images or satellite images




```
1  PROCEDURE gray_stretch(img: VECTOR OF gray; g_min,g_max: gray): VECTOR OF gray;
2  (* stretch gray values into interval [g_min..g_max] *)
3  VAR tmax,tmin: INTEGER;
4  BEGIN
5    tmin := REDUCE.MIN(img);
6    tmax := REDUCE.MAX(img);
7    IF tmin = tmax THEN INC(tmax) END;  (* avoid division by 0 *)
8    RETURN (g_max-g_min) * (img-tmin) DIV (tmax-tmin) + g_min;
9  END gray_stretch;
```

3.1.3 Reduce Number of Gray Levels

Reduce the number of different gray values in an image
e.g. for reducing required memory / disk space



Original with 256 gray levels

16 gray levels

8 gray levels

Reduce Number of Gray Levels

```
 1  PROCEDURE gray_reduce(img: VECTOR OF gray; num_vals: gray): VECTOR OF gray;
 2  VAR tmax,tmin: INTEGER;
 3      res      : VECTOR OF gray;
 4  BEGIN
 5    tmin := REDUCE.MIN(img);
 6    tmax := REDUCE.MAX(img);
 7    IF tmax-tmin+1 > num_vals > 1 THEN
 8      (* stretch to [0..num_vals-1] *)
 9      res := (num_vals-1) * (img - tmin) DIV (tmax-tmin);
10      (* stretch to [tmin ..tmax]   *)
11      res := (tmax-tmin) * res DIV (num_vals-1) + tmin;
12     ELSE res := img  (* unchanged  *)
13    END;
14    RETURN res;
15  END gray_reduce;
```

3.2 Local Operators

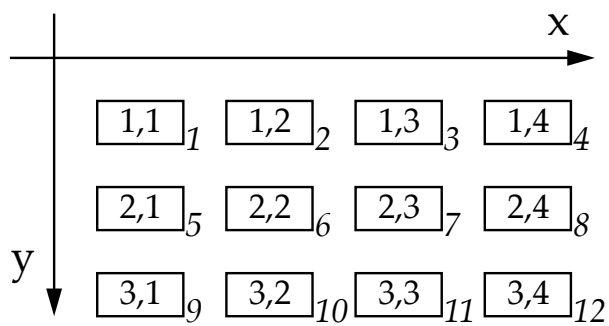
Local operators use their own pixel data and that of local neighbors only.

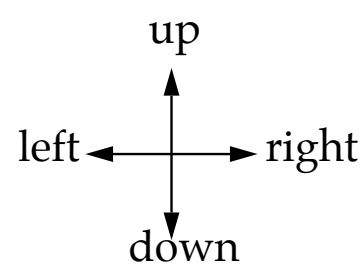
PE Configuration

```
1  CONFIGURATION grid[*],[*];  (* 8-way nearest neighbor grid *)
2  CONNECTION right: grid[i,j] <-> grid[i  ,j+1] :left;
3             up   : grid[i,j] <-> grid[i-1,j  ] :down;
4             up_l : grid[i,j] <-> grid[i-1,j-1] :down_r;
5             up_r : grid[i,j] <-> grid[i-1,j+1] :down_l;
```

Typical Procedure Head:

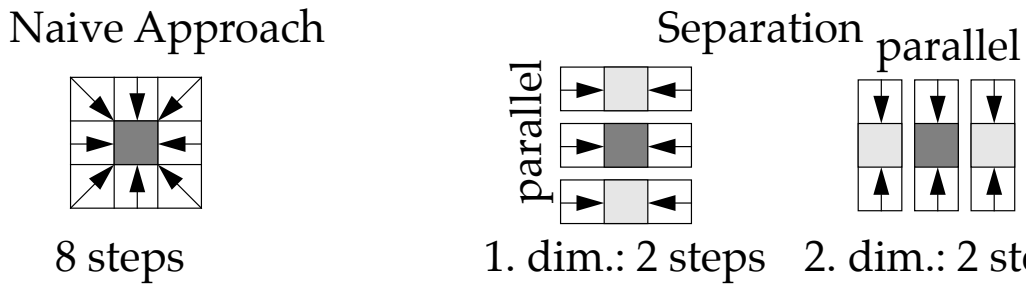
```
PROCEDURE xyz(img: grid OF gray): grid OF gray;
```





3.2.1 Mean

Compute the mean gray value from local neighborhood (smoothing)
naive approach: 8 data exchanges
with separation: 4 data exchanges




```
1  PROCEDURE sum_3x3(img: grid OF gray): grid OF INTEGER;
2  (* returns sum of local 3x3 neighborhood area *)
3  VAR res: grid OF INTEGER;
4  BEGIN
5    res:= img + MOVE.right(img) + MOVE.left(img); (* horizontal *)
6    res:= res + MOVE.down(res)  + MOVE.up(res);   (* vertical   *)
7    RETURN res;
8  END sum_3x3;
```

```
1  PROCEDURE mean_3x3(img: grid OF gray): grid OF gray;
2  (* mean value of 3x3 neighborhood *)
3  BEGIN
4    RETURN sum_3x3(img) DIV 9
5  END mean_3x3;
```

3.2.2 Median

1	7	4
3	1	99
6	0	2

Mean: (1+7+4+3+1+99+6+0+2) / 9 ≈ **14**

Median: middle-el. (0, 1, 1, 2, 3, 4, 6, 7, 99) = **3**

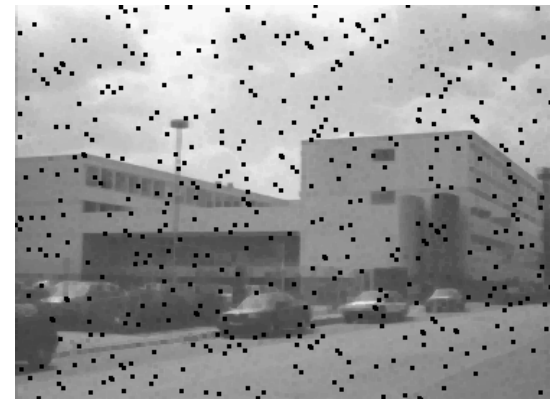
Fast-Median: middle-el. (middle-el.(1,7,4), middl-el.(3,1,99), middle-el.(6,0,2))
= middle-el. (4, 3, 2) = **3**

```
 1  PROCEDURE median_3x3fast(img: grid OF gray): grid OF gray;
 2  (* approximation: median in y-direction of median in x-direction *)
 3  VAR a: grid OF ARRAY[1..3] OF gray;
 4  BEGIN
 5    a[1] := MOVE.left (img);
 6    a[2] := img;
 7    a[3] := MOVE.right(img);
 8    (* sort 3 elems. with 3 comparisons/swaps *)
 9    IF a[1] > a[2] THEN swap(a[1],a[2]) END;
10    IF a[2] > a[3] THEN swap(a[2],a[3]) END;
11    IF a[1] > a[2] THEN swap(a[1],a[2]) END;
12    (* send median in x-direction up and down *)
13    SEND.up  (a[2],a[1]);
14    SEND.down(a[2],a[3]);
15    (* sort 3 elems. with 3 comparisons/swaps *)
16    IF a[1] > a[2] THEN swap(a[1],a[2]) END;
17    IF a[2] > a[3] THEN swap(a[2],a[3]) END;
18    IF a[1] > a[2] THEN swap(a[1],a[2]) END;
19    RETURN a[2];
20  END median_3x3fast;
```

3.2.3 Noise Reduction

Original

Noise added



Minimum filter

Maximum filter

Mean filter

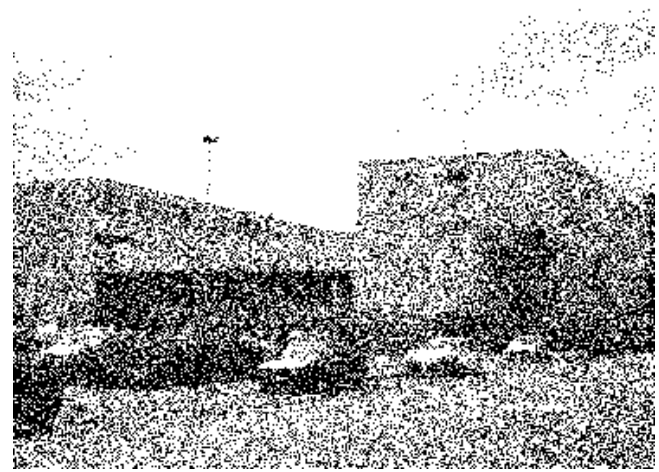
Median filter

3.2.4 Dithering

Transform a gray image to a binary image, simulating more intensity levels

a) Random Dithering
Use an individual random threshold to convert each pixel from gray to binary

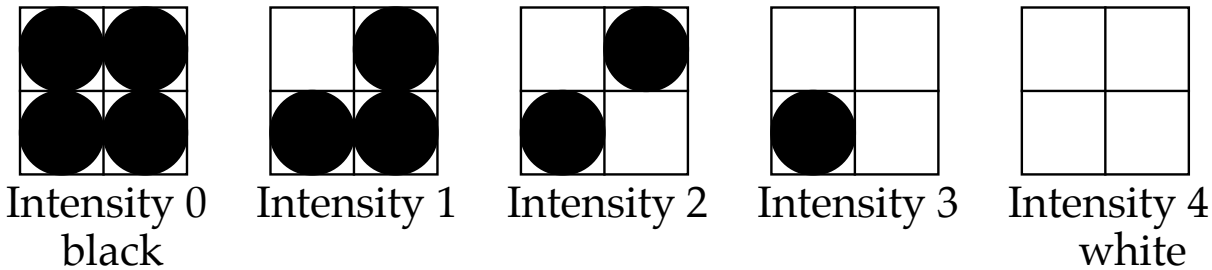
```
1  PROCEDURE dither_random(img: VECTOR OF gray): VECTOR OF binary;
2  CONST low  =  50;  (* all pixels below become black *)
3        high = 200;  (* all pixels above become white *)
4  BEGIN
5    RETURN img < low + RandomInt(img) MOD (high-low)
6  END dither_random;
```



Dithering

b) Ordered Dithering

Examine only one in four pixels and set the four binary values according to its gray value



Dithering

```
 1  PROCEDURE dither_ordered(img: grid OF gray): grid OF binary;
 2  CONST thres = g_white DIV 5;
 3  VAR res: grid OF binary;
 4  BEGIN
 5    IF ODD(DIM(grid,2)) AND ODD(DIM(grid,1)) THEN
 6      res := img < thres;  (* upper left corner *)
 7      SEND.right (img < 3*thres,res);
 8      SEND.down  (img < 4*thres,res);
 9      SEND.down_r(img < 2*thres,res);
10    END;
11    RETURN res;
12  END dither_ordered;
```

3.2.5 Edge Detection

Laplace Operator: edge detection by amplifying gray scale gradient

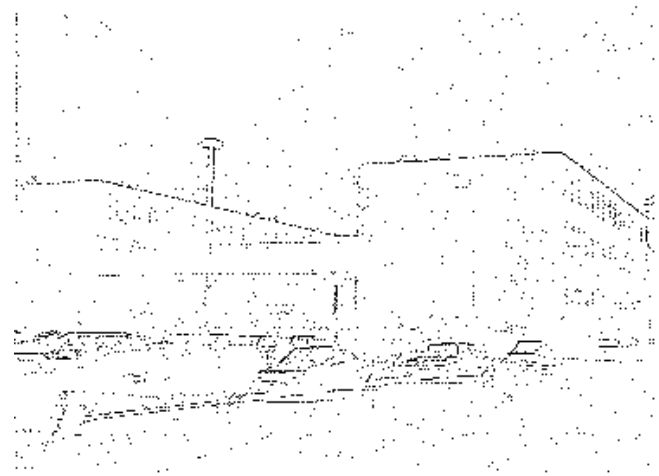
```
1  PROCEDURE laplace_3x3(img: grid OF gray): grid OF INTEGER;
2  BEGIN
3    RETURN 4*img -MOVE.up  (img) -MOVE.down (img)
4                 -MOVE.left(img) -MOVE.right(img);
5  END laplace_3x3;
```

operator for each pixel:

	−1	
−1	4	−1
	−1	

Laplace filter

Laplace filter with threshold



Sobel Operator

Sobel Operator: two templates for gradients in x- and y- direction

–1		1
–2		2
–1		1

Sobel-x

1	2	1
–1	–2	–1

Sobel-y

$$\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} -1 & 0 & 1 \end{pmatrix}$$

Sobel-x separation

```
1  PROCEDURE sobel_x_3x3(img: grid OF gray): grid OF INTEGER;
2  VAR col: grid OF INTEGER;
3  BEGIN
4    col := 2*img + MOVE.up(img) + MOVE.down(img);
5    RETURN MOVE.left(col) - MOVE.right(col);
6  END sobel_x_3x3;
```

```
1  PROCEDURE sobel_y_3x3(img: grid OF gray): grid OF INTEGER;
2  VAR row: grid OF INTEGER;
3  BEGIN
4    row := 2*img + MOVE.left(img) + MOVE.right(img);
5    RETURN MOVE.down(row) - MOVE.up(row);
6  END sobel_y_3x3;
```

Sobel Operator

Strength b and direction r of Sobel operator:

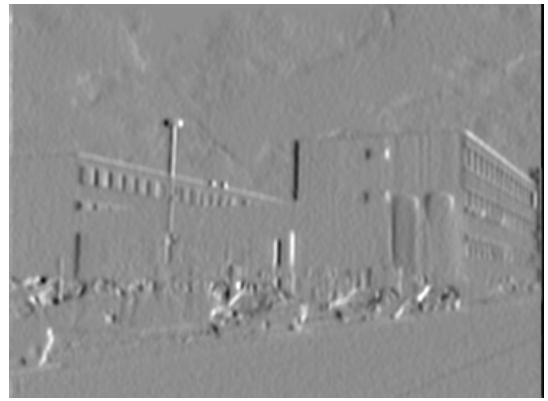
$$b = \sqrt{dx^2 + dy^2} \qquad r = \operatorname{atan}\frac{dy}{dx}$$

```
 1  PROCEDURE edges_sobel_3x3(img: grid OF gray;
 2            VAR strength,direction: grid OF gray);
 3  (* Sobel edge strength and edge direction *)
 4  VAR dx,dy: grid OF INTEGER;
 5  BEGIN
 6    dx := sobel_x_3x3(img);
 7    dy := sobel_y_3x3(img);
 8    strength := limit2gray( ABS(dx) + ABS(dy) );
 9    direction:= round( (arctan2(FLOAT(dy),FLOAT(dx)) + pi) / (2.0*pi) * 255.0 );
10  END edges_sobel_3x3;
```

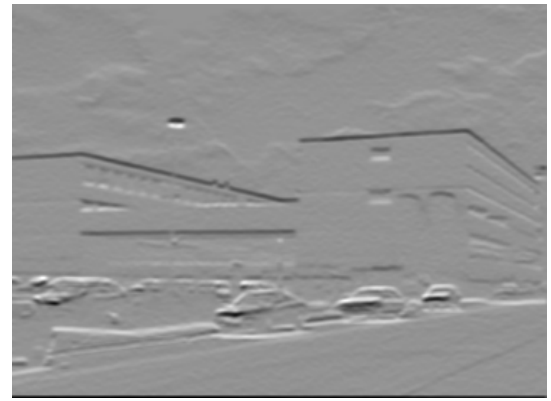
Sobel Operator

Original

Sobel-x

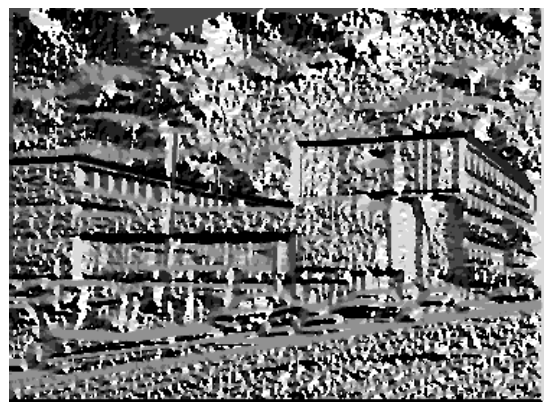


Sobel-y



Sobel strength

Sobel direction



Sobel strength with threshold

Conclusion

Image processing

- is an interesting *application area* for teaching parallel processing
- wide range of operators
- from simple to challenging algorithms
- Parallaxis-III is well suited for implementing parallel image operators

Parallaxis-III

- data-parallel programming (MasPar, CM)
- hardware independent
- structured programming
- simulation system for Unix workstations or PC-linux
- gnu-based source level debugger
- visualization tools
- email: braunl@informatik.uni-stuttgart.de
- ftp: ftp.informatik.uni-stuttgart.de, directory: pub/p3 or pub/parallaxis
- www: http://www.informatik.uni-stuttgart.de/ipvr/bv/p3/p3.html