CELIP: A cellular language for image processing

Wilhelm HASSELBRING
Uniuersität-GH-Essen, Fachbereich 6, Informatik/Software Engineering. Schützenbahn 70, 4300 Essen 1, FRG

Abstract. Cellular automata might be implemented as programmable special-purpose processors controlled by universal computer systems. This configuration is appropriate, because in general only special tasks of complex problems are efficiently parallelizable. Many tasks for image processing like window operators for filtering, smoothing or edge detection may be presented in a way particularly suitable for cellular automata. In this paper we present CELIP, a CEllular Language for Image Processing. It has been developed for prototyping cellular algorithms for image processing.

Keywords. Parallel computer, SIMD, cellular automaton, cellular algorithm, programming language, image processing.

1. lntroduction

The use of faster logical circuits and their minimization increased the performance of computers by magnitudes in the last decades. For a long time the classical von Neumann architecture and sequential processing of programmes has been kept. This kind of processing seems to come to an upper limit for further essential advance in programme efficiency.

One way for an increase in the performance of computer systems is to parallelize computing. But only by the advancements in VLSI-technology and a dramatic drop of hardware costs this way became feasible. Two well-known types of parallel computers are the array processor and the multiprocessor. According to the classification of Flynn [2], the first belongs to the *sing/e instruction stream-multiple data stream* (SIMD) computers and the latter to the *multiple instruction stream-multiple data stream* (MIMD) computers.

Another technique for increasing the performance is pipelining, where a problem is divided into functional units and pumped through corresponding hardware components. The input is processed according to the production-line principle.

A theoretical model of a parallel computer is the cellular automaton. The reader is referred to [13] for a full account of cellular automata. A cellular automaton consists of a finite number of Moore-type automata being fixed to the gridpoints of a finite-dimensional space. The automata are connected by a homogeneous pattern. The states of the automata are changed synchronously according to a given function.

2. Cellular image processing

The construction of parallel computer architectures on higher levels was launched only recently, because it was-with respect to the costs for software engineering-more economical to increase the performance by improvements of the construction elements.

Parallel programming and in particular cellular programming requires a new approach to problem solving compared with conventional sequential programming. It is possible to obtain relatively complex behavior of the entire system by relatively simple local instructions. A well-known example for this is the *game of life,* introduced by Conway [3]. A human being can emulate this global operations only in a sequential manner (cell by cell). That is remarkable, just because the human brain itself works in a parallel (neuronal) way.

The use of cellular automata, however, presents some problems. First, barely practical useable cellular algorithms are available. Another shortcoming is the requirement of too large and too many cells for a hardware realization. Hence it is necessary using simulators to develop cellular algorithms. Examples for this approach are CELLAS (cellular space simulation language) [8] and BVCP (the Brunswick Virtual Cellular Processor) [5].

The first cellular algorithms were constructive proofs for important theoretical questions like the computational universality of cellular automata. But these algorithms provide no practical support for cellular programming, because they mostly do not take advantage of the parallelism. Synchronization problems like the Firing Squad Synchronization Problem [13] have greater practical benefit. This are partial tasks to operate cellular automata.

lt is well known that not all tasks are efficiently parallelizable. A measure for the degree of parallelizability is the amount of information (data), that is simultaneously moved around. Examples for good parallelizable numerical tasks are matrix operations [9].

On the other band, many tasks for image processing are tailor-made for cellular automata. A digital grey-scale image is represented in a computer as a two-dimensional rectangular array of discrete grey-scale values (pixels). Typical methods are window operators for filtering, smoothing or edge detection. Parallelization is attractive here since the same operation is applied to every image point, at which the resulting value depends on the grey-scale values of the respective neighbouring points (in the window).

But also in image processing only partial tasks are efficiently parallelizable. Thus a cellular automaton for it will not be operated as a stand-alone computer, but as a special-purpose processor in an universal computer system [7]. This mode of operation is quite appropriate, because in this way the parallelized tasks may be operated on the cellular automaton and all the services offered by the host system may be used.

Programming of cellular automata is done essentially by constructing transition tables. This method is oriented towards the internal representation of cellular algorithms by tables and not towards the requirements of their app1ications. An additional difficulty for handling cellular programmes derives from this distance to the applications. It is hard to see which function such a table computes.

3. The programming language CELlP

The programming Language CELIP [4) is defined as an extension to Standard-C [6]. A language for cellular automata should be made available, since such an automaton attached to a computer system is to be programmed and controlled by the host.

In the configuration described above there exists the problem of transferring the input (digital image) between the cellular automaton and the host. To obtain a well-balanced relation between the time for loading and for processing, in most existing systems the input is pumped

```
cell
{
     byte reg1,
          reg2;
     byte regarr [9];
}
```
Fig. 1.A definition of the automata cell.

A	B	C
D	E	F
G	H	I

Fig.2. The '8-Neighbourhood' in image processing.

through a pipeline (column for column). Hence for window operators the grey-scale values of the neighbour points have to be held. To avoid this difficulty, in CELIP each operation is applied to all points simultaneously. This concept belongs to the above-mentioned SIMD organization. CELIP makes no operations for pipelining .available.

In most existing systems it is necessary to define the cellular net (retina). In CELIP the cellular net is fixed in size and shape. Every image point corresponds exactly to one cell.

The expanded instruction set allows defining the cell memory, the neighbourhood connections and the transition functions for the cellular automaton. In principle it is possible to define an arbitrary neighbourhood.

These assumptions are not realistic with a view towards a possible hardware realization, but they are well suited to prototyping cellular algorithms for image processing.

3.1. The definition of the automata cell

The user defines the local memory which is divided into several 8-bit-registers for the cells (identical in structure for every image point). The declarations take place in a ce ll-block to set them apart from the declarations for memory on the host. An example is given in Fig. 1.

The reserved word byte indicates an 8-bit-register (one byte) which can hold the integer values from 0 to 255 (2^8-1). lt is planned to add further register types. Especially 16-bit-register to increase the range for intermediate results. Such supplements won't change the basic concepts of CELIP.

3.2. Intensities

Sometimes it is necessary to operate in several registers in all cells with certain intensities (grey-scale values). To meet this requirement the data type intensity is introduced. intensity-variables can hold the same values as a cell register, hence integer values from 0 to 255. The memory for variables of this data type is located in the host's memory, not in the cellular automaton.

3.3. The definition of the neighbourhood

In theory the neighbourhood connections are specified with a neighbourhood index [13]. In a two-dimensional space the neighbourhood index consists of a finite number of pairs of natural numbers. All pairs are different. The cellular automaton is an euclidean space, where each cell has an unique pair of *x*- and y-coordinates. The *x*- and y-coordinates of the neighbour of a cell are obtained by adding the corresponding elements of the neighbourhood index (offsets) to the coordinates of this cell.

Figure 2 shows an example for the so-called 'Moore-Neighbourhood'. Especially in irnage processing it is also called '8-Neighbourhood'.

In image processing it is usual to represent the neighbourhood connections in this way. In this example the name E is implicitly assumed to mark the centre cell. The pair (0, 0)

```
(-1,  1)  (0,  1)  (1,  1)
(-1,  0)  (0,  0)  (1,  0)
(-1,-1)   (0,-1)   (1,-1)
```

Fig. 3. Neighbourhood index for the '8·Neighbourhood'.

```
neighbourhood
{
      A      B      C
      D     $E      F
      G      H      I
}
```

Fig. 4. The '8-Neighbourhood' in CELIP.

corresponds in the neighbourhood index to the centre cell. The neighbourhood index in Fig. 3 corresponds with the above definition. Strictly spoken the name '9-Neighbourhood' would be more suitable, because each cell is a neighbour of itself.

In CELIP the definition of the neighbourhood takes place with (modest) graphical means. The implemented CELIP-compiler converts this representation into an internal representation, that is in accordance with the neighbourhood index. An example for the '8-Neighbourhood' is given in Fig. 4.

The preceding reserved word neighbourhood could be replaced by the synonymous word neighbourhood. The declaration of the neighbours takes place inside the braces. The neighbourhood index is assigned according to the positions of the names with respect to spaces or tabs and line feeds. The offsets are related to the position of the centre cell, which is marked with the character $. The first neighbour in the row with the centre cell obtains the lowest x-offset in the neighbourbood index. The line feed decreases the y-offset by 1. It is obligatory to declare exactly one such centre cell. The optional name behind S has to follow directly without preceding whitespace characters.

lt is possible to define an indexed neighbourhood array. This is done by declaring the array indices explicitly to obtain the relation to their respective positions. These array indices should not be confused with the neighbourhood indices, which have geometrical means.

As an example serves the so-called 'von Neumann-Neighbourhood' in Fig. 5. Especially in image processing it is also called '4-Neighbourhood'.

The '-'-characters inside the neighbourhood definition increases the x-offset by 1. It can be seen as a neighbour without a name or an index. Both left '-'-characters are necessary in Fig. 5 to obtain the right x-offsets for N[2] (0, 1) and N[4] (0, – 1). The right ones were inserted for the optical symmetry.

An indexed neighbourhood array is in particular useful for processing the neighbours in a loop.

The access to registers of the neighbours is syntactical similar to the selection of structure components in C. For example in the expression 'A.reg1' register reg1 of neighbour A is selected. The selection 'E.reg1' is equivalent to 'reg1', because the own registers (in the centre cell) are the default.

```
neighbourhood
{
   N [
           –     2     –
           3    $0     1
           –     4     –
      ]
}
```

Fig. 5. The '4-Neighbourhood'.

```
where  (reg2 >  100)
{
  there  reg1=255;
  else   reg2=0;
}
```

Fig. 6. Threshold binarization with the where-statement.

```
where (F.reg1 <= 150)
{
  there   reg1= B.reg2;
  else    reg2= F .reg1;
  there   reg2= A.reg1;
  else    regl= D.reg2;
}
```

Fig. 7. The where-statement.

```
main ()
{
    int i, j;     /* suitable index counter */
    cell
    {
        byte reg,        /* for the greyscale image */
             temp;       /* for exchange */
        byte regarr [9]; /* for sorting */
    }
    neighbourhood
    {
        MOORE [
                 4  3  2
                 5 $0  1
                 6  7  8
              ]
    }
    /* Fetch the neighbour values from a file for sorting:  */
    load ("greyscale.dat", &reg);
    for  (i = 0; i <  9; i ++)
        regarr[i] = MOORE(i].reg;
    /* The Bubblesort-Algorithm:  */
    for  (i = 0; i <  5; i ++)
    {
        for (j = 8; j  > 1; j --)
        {
            where (regarr[j] < regarr(j-1))
            {
               /* temp is temporary used to excha.nge the
                * contents (tobubble):
                */
               there  temp =regarr[j];
               there  regarr[j] =regarr[j-1];
               there  regarr[j-1] =temp;
            }
        }
    }
    reg = regarr [4]; /*  The resulting median */
}
```

Fig. 8. Median filtering.

3.4. Cellular statements

The transition functions are specified by assignments to a register of the centre cell. This cellular statements are executed on the cellular automaton. The syntax corresponds with the syntax of plain C-assignments. First an example (the above definitions are valid):

```
          intensity    threshold;
          threshold = 100;
          reg1  =  reg2  >  threshold;
```

Tue first statement is no cellular Statement, because it is executed by the host. Cellular statements like the second one are executed by the cellular automaton. The result of the operation on the right side of the assignment then becomes the subsequent state for the register specified on the left hand side.

The binary operator > performs a binarization: The registers reg1 in all those cells, in which the relation 'reg2 > threshold' is valid, obtain the integer value 255; elsewhere they will hold the value 0. This operation is a so-called *threshold binarization.* It is a *point operation* because the neighbourhood connections are not used. The result of such a binarization is a binary image in the destination register.

There are binary operators for other comparisons and arithmetic and logical operations with the common priorities available. The results of the arithmetic operations are restricted to the values from 0 to 255 (to be able to hold them in 8-bit-registers).

Tue logical operators are represented by the reserved words and, or, xor and the unary operator not resp. to distinguish them from the corresponding C-operators. In C there exist different operators for logical and bit-wise logical operations, but in CELIP only bit-wise logical cellular operators are available.

The following operation is an example for a *window binarization* to present a more complex example:

```
          reg1  =  100 < reg2  and  reg2  <  200;
```

For binary images *erosion* is used for noise reduction, whereby the object areas are reduced by their marginal points. An object area in a binary image is represented by connected object points. Object points are cells with the integer value 255 in the register for the binary image. An object point only remains an object point, if all his neighbours are object points themselves. This may be expressed as follows:

```
          reg1  = A.reg1  and  B.reg1  and  C.reg1  and  D.reg1  and
                  E.reg1  and  F.reg1  and  G.reg1  and  H.reg1  and  I.reg1;
```

Small interferences, but also thin lines, are eliminated. The reverse operation *dilatation* can be realized with the or-operator instead of and.

The aggregate operators max and min are used as function calls with a variable number of operands. Tue resulting values are the maximal resp. minimal values of the cellular operands. Thus another expression for erosion is the following:

```
          reg1  =  max  (A.reg1,  B.reg1,  C.reg1,  D.reg1,  E.reg1,  F.reg1,
                         G.reg1,  H.reg1,  I.reg1);
```

The 'Roberts-Gradient' is used for edge enhancement [14]. It is realized in CELIP as follows:

```
          reg1  =  max  (|B.reg1  -  A.reg1|,  |D.reg1  -  A.reg1|);
```

The character | can be used as parentheses. For addition and subtraction the resulting values are the absolute values of the result given by the enclosed expression.

A more complex statement is the conditional where-statement. Figure 6 shows another realization of the above-mentioned threshold binarization.

An example to explain the reasons for the choosen syntax is given in Fig. 7.

In every there- and else-branch we allow only one assignment or an interlocked where-statement. This restriction is necessary to synchronize the statements. In general in some cells the given condition is valid (the there-statements are executed) and in others it is not valid (the else-statements are executed). Thus the there- and else-statements can influence each other via the neighbourhood connections. Consequently the necessary synchronization is done in a syntactical way. The else-branches are optional.

4. Examples

We discuss now some more sophisticated examples.

4.1. Median filtering

Median filtering is a smoothing operation, working with various neighbourhood connections (windows) [11,14).

4.1.1. The algorithm

The values in the neighbourhood of each cell are ordered according to their grey-scale values. The middle value is the resulting median. This filter is used to eliminate short-wave fluctuations (among other things the noise) without levelling the edges out.

4.1.2. The implementation in CELIP

In this example an indexed '8-Neighbourhood' is used. The complete programme is given in Fig. 8.

At first the values of the neighbours are stored in the local array regarr to sort them there. The obtained values are sorted by a bubblesort-algorithm. lt is sufficient to sort up to the middle.

4.1.3. Application

Figure 9 shows the application of this median filter. Figure 9(a) is the original image. In Fig. 9(b) the original is overlayed with random noise and in Fig. 9(c) the obtained noisy image is median filtered.

4.2. Thinning

We now present a parallel thinning algorithm.

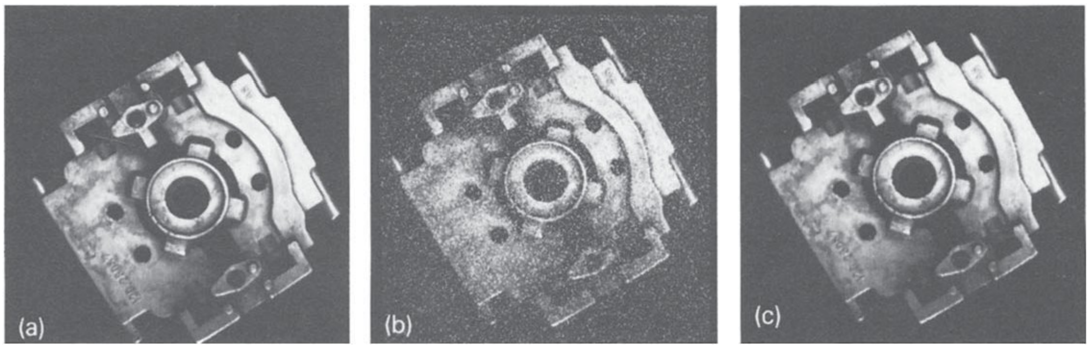

Fig. 9. Application of the median filter.

```
0 0 *    * 0 0    * 1 *    * 1 *
0 1 1    1 1 0    1 1 0    0 1 1
* 1 *    * 1 *    * 0 0    0 0 *
  A1       A2       A3       A4

0 0 0    1 * 0    * 1 1    0 * *
* 1 *    1 1 0    * 1 *    0 1 1
1 1 *    * * 0    0 0 0    0 * 1
  B1       B2       B3       B4
```

Fig. 10. Configurations in which the central element is dele1ed.

```
main ()
{
    cell
    {
        byte r,         /* the binary image */
             oldr,      /* the previous state */
             mark;      /* marked for deletion */
    }
    neighbourhood
    {
        A   B   c
        D   $E  F
        G   H   I
    }
    load ("binary.dat",&r);
    oldr = 0;
    vhile (difference (&r,&oldr))
    {
        oldr = r;
        /* Sequently application of the masks A1 to B4 ! */
        mark = r and F.r and H.r and not(D.r or A.r or
        B.r); r = r and not (mark);
        mark = r and G.r and H.r and not(A.r or B.r or
        C.r); r = r and not (mark);
        mark = r and D.r and H.r and not(B.r or C.r or F.r);
        r = r and not (mark);
        mark = r and A.r and D.r and not(C.r or F.r or I.r);
        r = r and not (mark);
        mark = r and B.r and D.r and not(F.r or H.r or
        I.r); r = r and not (mark);
        mark = r and B.r and C.r and not(G.r or H.r or I.r);
        r = r and not (mark);
        mark'"' r and B.r and F.r and not(D.r or G.r or H.r);
        r = r and not (mark);
        mark = r and F.r and I.r and not(A.r or D.r or G.r);
        r = r and not (mark);
    }
}
```

Fig. 11. Thinning.

4.2.1. *The algorithm*

A characterization of a wide class of parallel thinning algorithms was presented by Rosenfeld [10]. We will present an algorithm of this family as described in [l].

Thinning is a basic transformation for binary images which associates with any object area (connected set of object points) a connected subset consisting only of simple digital arcs and curves (the skeleton). Thinning is usually achieved by considering local operations assigning state 0 to all object points which are not used either for identifying locally elongated regions or for preserving the connectivity both of the figure and of the background. The skeleton is considered as 8-connected (by the 'Moore-Neighbourhood') and the background is considered as 4-connected (by the 'von Neumann-Neighbourhood').

The skeleton is obtained after a finite number of steps as a stick-like figure having the same connectivity as the original objects. Hence all the elements of the skeleton have only two neighbours, except branch points, which have more than two, and end points, which have on1y one neighbour.

A practical application of thinning is the preprocessing of handdrawn symbols or other line-pattems, that generally have first to be enhanced and then transformed into a more suitable representation [12].

The algorithm suggested in [1] involved erasing the central elements lying in each configuration shown in Fig. 10.

The state of the *-elements can be either 0 or 1. Tue masks have to be applied in the sequence $A_1, B_1, A_2, B_2, A_3, B_3, A_4, B_4$ until all deletable points have been removed. In [1] it was shown that the algorithm works correctly.

4.2.2. *The implementation in CELIP*

The logical 1-elements are cells with the intensity value 255 in the corresponding registers (we are working with binary images).

The '8-Neighbourhood' given in Fig. 4 is used. For the application of mask A_1 for example at first all cells in the according configuration are marked:

```
          mark = r and F.r and H.r and not(D.r or A.r or B.r);
```

All marked cells are deleted:

```
          r = r and not (mark);
```

The other mask operations are applied accordingly until no more changes will occur. To check this condition the previous states are remembered in register `oldr`. The complete programme is given in Fig. 11.

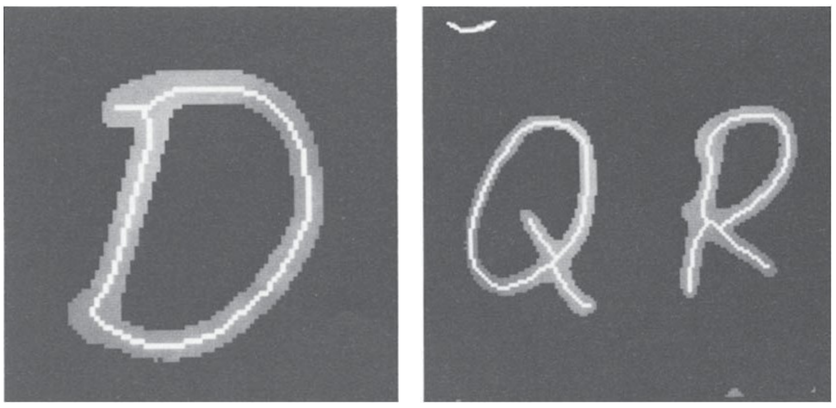

Fig. 12. Application of thinning.

4.2.3. Application

Figure 12 shows the application of this thinning algorithm to handdrawn letters. The background is black. The original letters are displayed with an average intensity and the skeleton is white.

5. Actual and prospective implementation

A CELIP-compiler was implemented on a µ-VAX which serves as a host. The cellular Statements are executed on a peripheral image-processing device VTE-PICTURECOM which emulates the cellular automaton. This specia1-purpose computer is capable of storing 24 grey-scale images (8-bit-register) with (512 x 512) image points. It can operate with two registers at a time via arithmetic-logical-units (ALUs). Such a read-write cycle is executed in video-time (40 ms).

It is possible to add the presented cellular extensions of C to another programming language like Pascal or Modula without any essential syntactical or semantical modifications.

For the implementation on different hardware the principle of operating a cellular automaton via a host system should be maintained. The main problem is to implement the variable neighbourhood connections with justifiable expense.

6. Possible extensions

The restriction to 8-bit-registers for the partition of cell memory is a concession to the capabilities of the VTE-PICTURECOM that emulates the cellular automaton. 16-bit-registers are desirable for increasing the range of intermediate results. The corresponding new data types for the implementation on different hardware would not change the basic concepts of CELIP.

The transmission of information between cells should be supported by syntax. For example sending and receiving signals is a usual task of cellular algorithms.

The explicit support of pipeline processing should also be considered.

7. Conclusion

A cellular language has been presented, which allows prototyping cellular algorithms for image processing by operating a cellular automaton attached to an universal computer system as a peripheral device. The operation of a cellular automaton via a host system is appropriate, because most problems are only partially efficiently parallelizable. To meet these requirements CELIP is defined as an extension of C.

Tue cellular extensions are concerned with the definition of the cell memory and the neighbourhood connections as well as for the cellular statements (transition functions). This concept is custom tailored for image processing.

The language in the present form is intended as a proposal for the basic paradigm of a programming language for cellular image processing.

Acknowledgment

I would like to thank R. Vollmar, E. Paulus and U. Döhler for encouraging me to do this work as well as for their helpful discussions and remarks. This thesis was performed at the

Technische Universität Braunschweig (Institut für Nachrichtentechnik, Schleinitzstrasse 23, 3300 Braunschweig, Fed. Rep. Germany).

References

[1] C. Arcelli, L.P. Cordella and S. Levialdi, Parallel thinning of binary pictures, *Electron. Leu.* **11** (1975) 148-149.
[2] M.J. Flynn, Very high-speed computing systems, *Proc. IEEE* 12 (1966) 1901-1909.
[3] M. Gardner, The fantastic combination of John Conway's new solitaire game "life", *Sc. Amer.* 223 (1970) 120-123.
[4] W. Hasselbring, CELIP – Eine zellulare Sprache zur Bildverarbeitung, Diplomarbeit, Institut für Nachrichtentechnik, Technische Universität Braunschweig, 1989.
[5] V. Henkel, BVZP - Braunschweiger virtueller Zellularprozessor – Ein Emulator für zellulare Netze, in: M. Feilmeier, G. Joubert and U. Sehendei, eds., *Parallel Computing 83* (North-Holland, Amsterdam, 1984) 325-333.
[6] B.W. Kernighan and D.M. Ritchie, *Programmieren in C* (Carl Hanser, München, 1983).
[7] T. Legendi, Cellprocessors in computer architecture, *Comput. Linguististics Comput. Languages* 11 (1976) 147-167.
[8] T. Legendi, lntercellas – an intcractive cellular space simulation language, *Acta Cybernet.* 3 (1977) 261-267.
[9] T. Legcndi, Cellular algorithms and their verification, in: R. Vollmar and T. Legendi, eds, *Cellprocessors and Cellgorithms*, Informatik-Skripten **1** (TU Braunschweig, 1981) 24-51.
[10] A. Rosenfeld, A characterization of parallel thinning algorithms, *Inform. Control* 29 (1975) 286-291.
[11] A. Rosenfeld and A.C. Kak, *Digital Picture Processing* (Acadernic Press, New York, 1976).
[12] E.E. Triendl, Skeletonization of noisy handdrawn symbols using parallel operations, *Pattern Recog.* 2 (1970) 215-226.
[13] R. Vollmar, *Algorithmen in Zellularautomaten* (Teubner, Stuttgart, 1979).
[14] F.M. Wahl, *Digitale Bildsignalverarbeitung* (Springer, Berlin, 1984).