C A S H
Cellular Automata in Simulated Hardware

Rob J. de Boer & Alex D. Staritsky,
Theoretical Biology,
Utrecht University,
Email: R.J.DeBoer@bio.uu.nl

This document accompanies CASH version 2003

Abstract

CASH is a system for the simulation of 2-dimensional Cellular Automata (CAs) in a rapid and conceptually elegant way. The system is fast because of clever neighborhood retrieval algorithms, and a fast X11-based graphical display. The system is elegant because it has a natural parallelism, defining all of its operations in terms of the whole array of automata.

1 Overview

The CASH system is inspired on the Cellular Automaton Machine (CAM) of Toffoli and Margolus (MIT Press, 1987). CASH simulates part of the CAM hardware and provides a similar, slower, but more flexible, environment. The basic idea of CASH is that the CA rules are defined in terms of the entire array of cells of the CA. In CASH such an array is called a "plane". Any CASH plane can be displayed on any full color X11 computer screen.

Most of the functions in the CASH library operate on planes and return a plane. An example is the logical function `And(a,b,c)`, where `a`,`b`, and `c` are planes. Calling `And(a,b,c)` means that plane `b` and `c` are compared and that the result is stored in plane `a`. Plane `a` is thus "on" at every position where both `plane b` and `plane c` are "on". A CASH function typically also returns a pointer to the result. Thus, writing `a = And(a,b,c);` is the same as writing `And(a,b,c);`. The former notation is helpful if you want to write something like `a = Or(a, And(b,b1,b2), And(c,c1,c2));`.

The best way to learn CASH is to look at the various examples and to browse through this reference manual. For more complicated applications we advise you through browse through the C-code of the library. The CASH environment is written in the C programming language. Using CASH required little knowledge of C. CASH has special algorithms for fast neighborhood retrieval, boundary conditions, and is designed to allow for vectorization and/or parallization.

2 Some CASH examples

2.1 Game of Life

The following is a listing of the `life.c` program in the `examples` directory:

```
#include <stdio.h>
#include <cash.h>

extern int nrow,ncol,scale;

main()
{
  int time;
  TYPE **state, **input;

  nrow = 256;
  ncol = 256;

  state = New(); input = New();

  ColorTable(0,1,BLACK,WHITE);
  OpenDisplay("Game of Life",nrow,ncol);
  Init(state);

  for (time = 1; time <= 10000; time++) {
    Moore8(input,state);
    PLANE(
      state[i][j] = (input[i][j] == 3 ||
      (input[i][j] == 2 && state[i][j]));
    );
    PlaneDisplay(state,0,0,0);
```

```
    if (Mouse()) Init(state);
  }
  CloseDisplay();
}
```

Notice in this program that

- The `extern int nrow, ...;` line makes some of the CASH options local to this file so that you can change them.
- The `TYPE **state ..` line declares two planes that are allocated and initialized by the `state = New();` line.
- The `ColorTable();` statement defines zero to be black and one to be white.
- The `OpenDisplay();` statement opens an X11-window with the title "Game of Life", and with a size of nrow lines and ncol columns.
- The `state` plane is initialized by the `Init()` function which prompts the user for further instructions.
- The time is defined by conventional C loop.
- The Moore neighborhood of each cell in the `state` plane is stored in the `input` plane.
- The `PLANE()` macro defines a loop over the whole field, i.e., `i=1, 2, .., nrow` and `j=1, 2, .., ncol`. For each `i,j` pair we define the Game of Life rule by a conventional C statement `a = (b)`, where `b` is a logical expression yielding "0" or "1".
- The logical expression employs the state and the Moore neighborhood input of each cell.
- The `state` plane is plotted by `PlaneDisplay();`
- If the user clicks any mouse button, the user is prompted to re-initialize the `state` plane.
- When time is up, or when the user quits from `Init()`, the window is closed, and the program terminates.

In the Unix environment this program is compiled and excuted by the following commands:
`make life`
`life`
In the examples directory we provides some interesting initial configurations for the Game of Life. Try to read the `glider1, glider2, boat` or `boats` files, and place them just somewhere in the plane.

2.2 Majority voting

This example first displays a random plane, it then does majority voting employing the CASH primitives `Copy(), Moore9()` and `GE()`. Finally it takes the "voted" plane into a voting-rule with "annealing" (Toffoli & Margolus, 1987). The latter is implemented as loop over the plane. According to the majority-voting rule each cell adopts the state the majority of its neigbors is in. The annealing just introduces errors at the critical values of the majority function (i.e., at 4 and 5 neighbors).

The following is a listing of the `vote.c` program in the `examples` directory:

```
#include <stdio.h>
#include <cash.h>

extern int nrow,ncol;

main(argc, argv)
int argc; char *argv[];
{
  int time, margin;
  TYPE **a, **b;

  if (argc>1) ReadOptions(argv[1]);
  a = New(); b = New();
  margin = nrow/10;
  OpenDisplay("Majority Vote",nrow+2*margin,3*ncol+4*margin);

  Random(a,0.5);
  PlaneDisplay(a,margin,margin,0);
  printf("Number of bits: %ld\n",Total(a));

  for (time = 1; !Equal(a,b) && !Mouse() && time<100; time++) {
    Copy(b,a);
    Moore9(a,b);
    GE(a,a,5);
    PlaneDisplay(a,margin,2*margin+ncol,0);
  }
  printf("Number of bits: %ld\n",Total(a));

  for (time = 1; !Mouse() && time<1000; time++) {
    Moore9(b,a);
    PLANE(a[i][j] = (b[i][j] == 4 || b[i][j] >= 6););
```

```
    PlaneDisplay(a,margin,3*margin+2*ncol,0);
  }
  printf("Number of bits: %ld\n",Total(a));

  CloseDisplay();
}
```

Notice in this program that

- The main program now has the conventional C arguments, and `ReadOptions()` uses this to read parameters from the filename given when the program was called.
- The `OpenDisplay()` function now opens a window which will fit three planes next to each other, plus margins around them.
- We initialize plane `a` with 50% bits "on"; this plane is displayed, and the number of bits that are "on" is printed.
- The first time loop stops when the previous and the current plane are identical, or when the user clicks the mouse, or after one hundred time steps.
- The three CASH functions save a copy of plane `a` in `b`, they store the Moore neighborhood of `b` in `a`, and check for each point in `a` if it has five or more neighbors. The result of this is stored in `a` and is displayed on the screen.
- At the end of the first loop the total is printed again.
- In the next time loop the previous result is modified further by annealing. The Moore neighborhood of `a` is stored in plane `b`. Subsequently we loop over the plane and set plane `a` "on" whenever it has four or more than five neighbors.

Note that the three CASH functions in the first loop could be replaced by
`Moore9(b,a);`
`PLANE(a[i][j] = (b[i][j] >= 5));`
where we however no longer check whether `a` and `b` are equal.

2.3 Diffussion limited aggregation

This example models "free" particles moving around by Brownian motion which freeze when they find a "frozen" particle in their immediate (Moore) neigborhood. The free and the frozen particles are stored in two separate planes. The program further illustrates how user parameters can be read from a parameter file, and how one can save Planes as .png for making movies.

The following is a listing of the `dla.c` program in the `examples` directory:

```
#include <stdio.h>
#include <cash.h>

extern int scale,nrow,ncol;

main(argc, argv)
int argc; char *argv[];
{
  int i, j, time, margin, movie = 1;
  TYPE **frozen, **free, **moore, **tmp;
  float dens = 0.1;

  if (argc>1) {
    ReadOptions(argv[1]);
    InDat("%f","dens",&dens);
    InDat("%d","movie",&movie);
  }

  margin = nrow/10;
  ColorTable(16,19,BLACK,WHITE,RED,GREEN);
  OpenDisplay("Diffussion limited aggregation",nrow+2*margin,3*ncol+4*margin);
  if (movie) OpenPNG("Frames",nrow,ncol);

  frozen = New(); free = New(); moore = New(); tmp = New();

  frozen[nrow/2][ncol/2] = 1;
  Random(free,dens);

  printf("Particles: %ld\n",Total(frozen)+Total(free));

  for (time = 1; !Mouse() && time <= 10000; time++) {
    Motion(free,free,1.0,time);
    Moore8(moore,frozen);
    PLANE(
      if (free[i][j] && moore[i][j] && frozen[i][j] == 0) {
         frozen[i][j] = free[i][j];
         free[i][j] = 0;
      }
    );
    PlaneDisplay(frozen,margin,margin,16);
    PlaneDisplay(free,margin,2*margin+ncol,16);
```

```
    BinSum(tmp,2,frozen,free);
    PlaneDisplay(tmp,margin,3*margin+2*ncol,16);
    if (movie) PlanePNG(tmp,16);
  }
  printf("Particles: %ld\n",Total(frozen)+Total(free));
  CloseDisplay();
  if (movie) ClosePNG();
}
```

Notice in this program that

- That we call `InDat()` to read the floating point parameter `dens` and the integer parameter `movie`.
- `ColorTable()` here defines four colors. The color table entries 16,17,18, and 19 are BLACK, WHITE, RED, and GREEN respectively.
- The X11 window allows for three planes next to each other.
- If `movie==1` a directory is created for saving .png files.
- Four planes are allocated, we start with one frozen particle in the middle, and with a random distribution of free particles. The density of the latter can be changed by the `dens` parameter.
- The algorithm in the time loop first moves the free particles around.
- It subsequently stores the neighborhood of the frozen partciles in the plane called `moore`.
- Looping over the CA, it checks whether a free particle has a frozen particle in its neighborhood but not below itself.
- If this is true the free particle freezes.
- Both the frozen and the free planes are displayed.
- Both planes are combined in the `tmp` plane by a binary sum, i.e., `tmp = 2*frozen + free`. This combined plane is displayed and written as a .png file.

3 Advanced topics

3.1 Programming

Although the CASH library allows you to define CAs in term of operations on planes it is often much faster to write your own next state function as a loop over the entire plane. CASH provides a macro `PLANE()` which expands to such a loop, i.e., to

```
for (i=1; i <= nrow; i++)
for (j=1; j <= ncol; j++)
```

and then executes its argument for every point in the plane. This illustrates that a CASH plane is indexed by `1..nrow` and `1..ncol`. Instead of using the `PLANE()` macro one can of course also write a C function incorporating such a loop over its `plane` arguments. This is how the CASH library is made.

CASH planes use rows `0` and `nrow+1` and columns `0` and `ncol+1` for the boundaries. The standard neighborhood functions like `Moore9()` and `RandomNeighbor()` all make transparent use of the boundaries. In case one needs to program one's own neighborhood retrieval, one should first set the boundary condition by the `Boundaries()` function, e.g.,

```
Boundaries(b);
PLANE(a[i][j] = b[i-1][j+1]);
```

Note that this loop will access `b[0][ncol+1]`. This is safe because these boundaries are allocated by `New()` and set correctly by the `Boundaries()` function.

We advise you to make use of the standard neighborhood functions (like `Moore8()`) whenever possible. They are very fast and they take care of the boundary conditions. In the Game of Life example we first store the `Moore8()` neighborhood in a plane and define our own loop on the basis of the state-plane and the neighborhood-plane. This is definitely faster and saver than including the neighborhood retrieval in your own loop.

3.2 Type definition

Functions, and arguments to functions, can be of the type *plane*, *row*, *value*, *function*, *int*, or *float*. A *plane* is a double array of values (e.g., `plane a`, a *row* is a single array of values (e.g., `plane a[nrow]`), and a *value* is a single value (e.g., `plane a[nrow][ncol]`). The types *int* and *function* are standard C types.

The actual types of the values in the plane can be changed by the user in the `cash.h` file. The default type is `int`. For instance, one can define the type to be `unsigned char`, which would allow for integer values from $0 \ldots 255$. This nicely corresponds to the 255 colors that are available. In fact, the `unsigned char` suffices for most CA applications. Note however that CASH does not check for overflow or underflow. This is particularly tricky when one is

summing over large neighborhoods. For this reason the default value is `int`.

In case you want to change the type, from e.g., `int` to `unsigned char`, just change the first line of the `cash.h` file in the `cash/lib` directory from `typedef int TYPE;` into `typedef unsigned char TYPE;`. Subsequently you have to recompile the library by typing `make libcash.a` or `make install`.

3.3 Vectorization/Parallization

One of the objectives for the development of CASH was vectorization on a supercomputer. Thus all algorithms in the CASH library, including those for the graphical display, have been written such that they can be vectorized by the C-compiler. This was tested on a Convex and on a parallel SiliconGraphics machine. In fact, you will find Convex `/*$dir force_vector*/` directives in the C-code which tell the C-compiler that it is safe to do the subsequent loop in parallel. You may have to adapt these statements for your own machine (e.g., change the `/*$dir force_vector*/` into a `#pragma concurrent`); In the CASH directory you will find a script `convert` and two arguments `force.sed` and `pragma.sed` that will change the whole library for you.

To allow for vectorization/parallization avoid the use of global variables (like `nrow, ncol`) as controls in loops. In the CASH library we assign `nrow` and `ncol` to two local variables `nr` and `nc`. This tells the C-compiler that the loop conditions cannot change during the loop.

On some machines single loops are much faster (or easier to vectorize/parallize) than double loops. For this reason CASH allows you to write loops over the entire plane as a single loop. The format is:

```
int i,l;
for (i=first,l=last; i <= l; i++) a[0][i] = b[0][i];
```

which copies `plane b` to `plane a`. The global variables `first` and `last` are predefined by CASH. Note that we again copy the global variable `last` to a local variable `l` so that the C-compiler knows that the loop control cannot change due to the `a[i] = b[i]` assigment (i.e., `last` and `a[][]` could overlap).

4 Reference Manual

4.1 Options/parameters and their default settings

Options and parameters can be set using normal C assignments. Alternatively they can be read from a file using the `ReadOptions()` function. The following is a listing of all options

with their default value.
`int nrow = 100;`
`int ncol = 100;`
Sets the number of rows/columns in the planes.
`int boundary = 0;`
Sets the boundary condition. There are three possibilities: wrapped or periodic (`boundary=0;`), fixed (`boundary=1;`), or echo (`boundary=2;`). For the "echo" boundary condition the boundary echoes the value of the cell checking the boundary.
`int boundaryvalue = 0;`
Sets value of the boundary for fixed boundary conditions.
`graphics = 1;`
Switches the X11 graphics on.
`int scale = 1;`
Scales the graphical output.
`int seed = 55;`
Sets the seed of the psuedo random number series.
`int psborder = 1;`
Switch for plotting borders around planes in PostScript files.
`int psreverse = 1;`
Reverses the ColorRamp from white to black in PostScript files.
`int ranmargolus = 1;`
The `ranmargolus` option allows you to speed up the `Margolus()` function if you don't use the random plane. The Margolus function internally calls `Random()` only if `ranmargolus = 1`.

4.2 Basic operations

`plane New();`
Initializes and allocates a plane.
`int PlaneFree(plane a);`
Terminates and de-allocates a plane.
`plane NewP();`
Allocates a pointer plane (as returned by the Shift and NoiseBox functions). For de-allocating a pointer plane just use the C's standard `free();`.
`plane Fill(plane a, value c);`
Fills a plane with a constant value, e.g., `Fill(a,3);`.
`long Total(plane a);`
Returns the number of non-zero cells in a plane.
`int Equal(plane a,plane b);`
Tests whether two planes are equal.
`int Max(plane a); int Min(plane a);`
Returns the maximum/minimum value of a plane.

`plane Copy(plane a, plane b);`
Copies plane `b` to `a`.
`row CopyRow(row a, row b);`
Copies one row to another.
`plane Motion(plane a, plane b, float r, int time);`
Particle motion where `r` is the probability of each particle moving North, South, East or West. The `time` argument should take even and odd values alternatingly. On sequential machines `Motion()` can be used with `a=b`, and a trick to give alternate even and odd arguments is the following: `Motion(a,a,1.0,odd=!odd);` where odd is an integer.

4.3 Input/output operations

`int ReadOptions(char filename[]);`
Opens a file and reads options and parameters from it. An option or parameter file has to have each parameter on a different line. The name of the parameter and its value should be separated by a space or a tab. An example of a parameter file is
`nrow 256`
`ncol 256`
which, when read by `ReadOptions("filename");` sets the size of the planes.
`int InDat(char format[], char name[], &par);`
Reads a parameter from the file opened by `ReadOptions()`. For example, `InDat("%d", "kill", &kill);` sets the integer parameter `kill`.
`int iPrint(*FILE file, char format[], plane a);`
`int cPrint(*FILE file, plane a);`
`int bPrint(*FILE file, plane a);`
Prints a plane in integer/character/binary values into a file.
`int ReadPat(plane a, int row, int col, char filename[]);`
Reads integer data from a file and copies it into a plane at the position `(row,col)`. (See the `gilder` examples files.)
`Init(plane a);`
Prompts the user to type some instructions how to initialize `plane a`.

4.4 Random operations

CASH implements two functions for getting random numbers, and several functions allowing for randomness in planes.

For planes we have
`plane Random(plane a, float r);`
to fill a plane with random bits, where $0 \leq$ `r` ≤ 1 determines the coverage.

`plane Shake(plane a, plane b);`
to give every point in the plane a randomly chosen new position.
`plane Normalize(plane a, plane b);`
to normalize a plane to a coverage of 50% (by adding or deleting randomly chosen bits).
`plane NoiseBox(plane a);`
to get a *pointer* to a plane with 50% random bits. This uses the very fast NoiseBox algorithm of Toffoli & Margolus (1987). Please be **warned** that `NoiseBox` returns a pointer to an internal plane of CASH. If you intend to keep the random plane for further use make a copy of it, e.g., `Copy(b,NoiseBox(a));`

For general random operations use
`int SEED(int i);`
to set the seed of the pseudo random series,
`double RANDOM();`
to obtain uniform values between zero and one, and,
`double normal(double mean; double stdev);`
to obtain values from a normal distribution. The functions `RANDOM()` and `SEED()` are defined in the `cash.h` file. The default is an algorithm for random numbers developed by Knuth (see Numerical Recipes). You are free to change it in the header file `cash.h` to the UNIX `rand()` which is worse but faster.

4.5 Logical operations

`plane And(plane a, plane b, plane c);`
`plane Or(plane a, plane b,plane c);`
Returns the AND/OR of two planes.
`plane AndNot(plane a, plane b, plane c);`
Returns the And of one plane and the complementary of another, i.e., `a = (b && !c)`.
`plane Xor(plane a, plane b, plane c);`
Returns the exclusive Or of two planes.
`plane Not(plane a, plane b);`
Returns the complementary plane, i.e., `a = (!b)`.
`plane AndCopy(plane a, plane b,plane c);`
Copies if equal, i.e., `if (b && c) a = c; else a = 0;` .

4.6 Filter operations

`EQ(plane a, plane b, value c);`
Performs `a = (b == c)`.
`plane NE(plane a, plane b, value c);`

Performs `a = (b != c)`.
`plane GE(plane a, plane b, value c);`
Performs `a = (b >= c)`.
`plane LE(plane a, plane b, value c);`
Performs `a = (b <= c)`.
`plane GT(plane a, plane b, value c);`
Performs `a = (b > c)`.
`plane LT(plane a, plane b, value c);`
Performs `a = (b < c)`.
`plane IN(plane a,plane b,value c,value d);`
Performs `a = (c <= b <= d)`, e.g., a = IN(a,b,1,10);.

4.7 Arithmetic operations

The arithmetic functions come in two (or more) flavors. For instance, one can sum two planes using the `Sum(a,b,c)` function, and one can add a value (e.g., an integer) to a plane by using the `SumV(a,b,c)` function.
`plane Sum(plane a, plane b, plane c);`
`plane SumV(plane a, plane b, value c);`
`plane Minus(plane a, plane b, plane c);`
`plane MinusV(plane a, plane b, value c);`
perform $a = b + c$ or $a = b - c$.
`plane Mult(plane a, plane b, plane c);`
`plane MultV(plane a, plane b, value c);`
`plane MultF(plane a, plane b, float c);`
`plane Div(plane a, plane b, plane c);`
`plane DivV(plane a, plane b, value c);`
perform $a = b \times c$ or $a = b/c$.
`plane Mod(plane a, plane b, plane c);`
`plane ModV(plane a, plane b, value c);`
perform $a = b\%c$.
`plane BinSum(plane a, int n, plane b0, b1, .., bn);`
performs $a = b_0 + 2b_1 + 4b_2 + \ldots + 2^{n-1}b_{n-1}$. Thus the argument `n` tells `BinSum()` how many planes should be summed into `plane a`.

4.8 Bitwise operations

`plane RollRight(plane a, plane b, int c);`
`plane RollLeft(plane a, plane b, int c);`
Bitwise shift right/left.

`plane GetBits(plane a, plane b, int f, int l);`
Get bits from position f to l.
`plane PutBits(plane a, int p, int v);`
Set bits starting at position p to value v.
`plane Hamming(plane a, plane b, plane c);`
Returns the bitwise Hamming distance between two planes.

5 Neighborhood retrieval

CASH implements several Neighborhood functions. The basic functions, like the Moore neighborhood, are easy to use and are very fast. In case you perform the neighborhood retrieval yourself you can employ the function
`plane Boundaries(plane a);`
to set the boundaries according to the boundary-condition.

5.1 Basic neighborhood functions

Denoting a local 3×3 neighborhood as follows

$$\begin{array}{ccc} NW & N & NE \\ W & C & E \\ SW & S & SE \end{array},$$

`plane Moore8(plane a, plane b);`
returns $NW + N + NE + W + E + SW + S + SE$,
`plane Moore9(plane a, plane b);`
returns $NW + N + NE + W + C + E + SW + S + SE$,
`plane VonNeumann4(plane a, plane b);`
returns $N + W + E + S$,
`plane VonNeumann5(plane a, plane b);`
returns $N + W + C + E + S$,
`plane Diagonal4(plane a, plane b);`
returns $NW + NE + SW + SE$,
`plane Diagonal5(plane a, plane b);`
returns $NW + NE + C + SW + SE$, and,
`plane RandomNeighbor(plane a, plane b);`
returns the value of a randomly choosen neighbor in the 3×3 neighborhood.

5.2 The Margolus neighborhood

The Margolus neighborhood is based upon 2×2 blocks of the following form

$$\begin{array}{cc} CW & OPP \\ C & CCW \end{array} .$$

For a full discussion we refer to Toffoli & Margolus (1987). Margolus neighborhoods can be used for having "particle conservation". (An alternative being provided in CASH by the `Motion()` function). In CASH you define a function in terms of the four cells of the Margolus neighborhood (i.e., center (C), opposite (OPP), clockwise (CW), and counter clockwise (CCW)), and a random plane. You may suppress the actual call to `Random()` with the `ranmargolus` option.

The general syntax of the Margolus utility is
```
plane Margolus(plane a, plane b, function f, int time);
```
where the function `f` should look like
```
plane f(plane cnew, plane c, plane cw, plane ccw,
plane opp, plane ran);
```
Within such a function you may use normal CASH functions:
```
cnew = opp;
cnew = And(cnew,ccw,ran);
```
Please see the example on Brownian motion in the `examples` directory. Note that `time` should increase by one every time step. At odd and even times CASH selects different 2×2 blocks.

5.3 Shift operations

The basic neighborhood functions are based upon the following four primitives. These primitives can be used directly. This requires a little care however. The routines are very fast because they just return a *pointer* to a location in the argument plane. **The problem is that such a shifted plane no longer has a boundary!** Thus, writing `c = North(c,North(a,b));` is not allowed. This has to be written as `c = North(c,Explode(c,North(a,b)));`, where the explode operation makes a full plane of the pointer. A second complication is that one cannot use single loops (see the section on Vectorization) on shifted planes. Double loops are allowed but one should not assign any new values to a shifted plane. One typically just test the value of a neighbor and one should not alter a neighbor.

```
plane North(plane a, plane b); plane South(plane a, plane b);
plane East(plane a, plane b); plane West(plane a, plane b);
```
Returns a pointer to a particular neighbor.

`plane[9] Neighbors(plane a[9], plane b);`
Returns an array of nine pointers to the 3×3 neighborhood. Note that you do have to call `NewP()` for all nine elements of the array. One can index this array by following predefined indexes: CENTRAL 0, NORTH 1, WEST 2,EAST 3, SOUTH 4, NORTHWEST 5, NORTHEAST 6, SOUTHWEST 7, SOUTHEAST 8.

6 Graphics

6.1 X11 Graphics

`int OpenDisplay(char text[]; int row, int col);`
Opens a window of `row` by `col` pixels (note that `row` is the number of Y-values and `col` is the number of X-values).
`int CloseDisplay();`
Close the X11 window.
`int PlaneDisplay(plane a, int row, int col, int offset);`
`int RowDisplay(row a, int row, int col, int offset);`
Displays a plane/row at the location `(row,col)` in the window offsetting the color table at position offset.
`int BlockDisplay(unsigned char a[nr*nc],`
`int nr, nc, row, col, offset);`
Displays a block of `nr` by `nc` unsigned characters, e.g., any two-dimensional character array, at the location `(row,col)` in the window offsetting the color table at position offset.
`int BlockDisplayFast(unsigned char a[nr*nc],`
`int nr, nc, row, col);`
This is a special call used by the Movie functions for displaying a block of data without scaling or offsetting the colortable. You will probably never need it.
`int Mouse();`
Checks for mouse clicks `Mouse();` returns 1, 2, and 3 for the left, middle, and right mouse button, respectively.

6.2 Color

Simple functions allow the user to change the color table. These functions have to be called before `OpenDisplay()` is called. If you don't change the color table you will inherit your workstation's color table for the X11 graphics, and you will have black and white PostScript graphics. This is usually sufficient. CASH assumes that you have a colorscreen supporting 256 colors. In CASH the following colors are predefined integers: BLACK, WHITE, RED, GREEN, BLUE, YELLOW, MAGENTA, GRAY. Please check the color.c example.

`int ColorTable(int i,j; int k,l,..,m);`
Enumerate the color table where `i` is the first and `j` is the last entry in the color table to be defined. Hence you are expected to supply `j - i + 1` colors by the `k,l,..,m` parameters. For example `ColorTable(16,17,BLACK,WHITE);` sets color 16 to black, and color 17 to white. Using `PlaneDisplay(...,16);` one gets an offset to this part of color table.
`int ColorRGB(int i,r,g,b);`
Sets the color index `i` to the RGB value specificied by the `r,g,b` arguments ($0 \leq$ `r,g,b` ≤ 255).
`int ColorRamp(int i,j,k);`
Defines a color ramp from entry `i` to `j` in the color table, where `k` is the color to be scaled. This parameter `k` may be either: BLACK, WHITE, RED, GREEN, BLUE, or GRAY. For example `ColorRamp(0,16,RED);` makes a color ramp of 17 shades of red.
`int ColorWheel(int i,j);`
Defines a color wheel from entry `i` to `j` in the color table.
`int ColorRandom(int i,j);`
Defines random colors from entry `i` to `j` in the color table.
`int ColorRead(char filename);`
`int ColorDump(char filename);`
Read or write the color table as RGB values to a file.

6.3 PostScript

Although CASH has PostScript functions with the same arguments as the X11 graphics functions, you will probably find it more convenient to write .png files for graphical output (see the section on making Movies).
`int OpenPostscript(char filename[]; int row, int col);`
Opens a PostScript file with `row` by `col` pixels.
`int ClosePostscript();`
Closes the file.
`int PlanePostscript(plane a, int row, int col, int offset);`
`int RowPostscript(plane a, int row, int col, int offseet);`
Dump a plane/row a the location `(row,col)` in the page offsetting the color table at position `offset`.
`int BlockPostscript(unsigned char a[nr*nc],`
`int nr,nc,row,col,offset);`
Dumps a block of `nr` by `nc` unsigned characters, e.g., any two-dimensional array, at the location `(row,col)` in the page offsetting the color table at position offset.
`int TextPostscript(char text[]; int row, int col);`
Prints some text at the location `(row,col)` in the page.
`int PagePostscript();`
Starts a new page.

6.4 Movies

CASH is able to save planes as .png files. This enables you to save graphics and to make movies (see the README_MOVIES file). The format is similar to that of the Display routines. Please check the `dla.c` example.
`int OpenPNG(char directoryname[]; int row; int col);`
Creates a directory and allocates the required memory.
`int ClosePNG();`
Closes the PNG-environment and writes an example `mpeg.par` file.
`int PlanePNG(plane a; int offset);`
Writes a plane as a .png file.
`int BlockPNG(unsigned char a[]; int nrow, ncol, offset);`
Writes a unsigned char block as a .png file.

6.5 Old Movie Format

CASH is still able to write planes to a compressed file. Such a file can be read and download to the screen. This allows for fast movies of time consuming simulations. Note that these files easily become extremely large. The format is similar to that of the Display routines.
`int OpenMovie(char filename[]; int row; int col);`
Opens a file for writing.
`int LoadMovie(char filename[], int *row; int *col);`
Open a file for reading.
`int CloseMovie();`
Closes the file.
`int PlaneMovie(plane a; int row; int col; int offset);`
`int RowMovie(row a; int row; int col; int offset);`
Writes a plane/row to the file.
`int BlockMovie(unsigned char a[];`
`int nr, nc, row, col, offset);`
Writes a unsigned char block to the file.
`int PlayMovie(int offset);`
Read an image and displays it on the screen.
`plane MoviePlane(plane a);`
Copies the last image that `PlayMovie()` has plotted onto the screen into the `plane a`. It is your responsibility that `plane a` has the correct dimensions (see `New(), nrow, ncol`).

7 C A S H examples directory

7.1 Usage

Compile CASH programs by employing the UNIX `make` utility. The makefile should be edited such that it matches the installation of CASH on your system. If you have a proper makefile you only need to type
`make program`
`program`
or if you want to supply parameters, type `program par`.

7.2 The examples directory

In the examples directory you will find the following files:

- `life.c` the Game of Life example listed above.
- `vote.c` the Majority vorting example listed above.
- `life.c` the Diffusion limited aggregation listed above.
- `margolus.c` See Toffoli and Margolus (MIT Press, 1987). Two of their examples in one program: with and without a random plane. Which one is chosen depends on the value of the `brownian` parameter.
- `motion.c` shows an alternative way of doing particle motion.
- `movie.c` an example of a program reading an old CASH movie file and writing .png files.
- `color.c` Shows how one can modify the colortable by displaying 256 planes.
- `moore.c` Shows how one achieves a large neighborhood with a roughly Gaussian weighting by repeatedly calling the standard Moore neighborhood function.
- `array.c` Shows how one can use CASH to plot unsigned char matrices.
- `noise.c` compares the speed of the `NoiseBox()` function with that of the `Random(a,0.5)` function.

8 C A S H Installation

Once you have received the compressed `cashXXXX.tar.Z` file you uncompress and untar it, go into the `lib` directory, and make the library. Check the `install` entry in the makefile for setting the `local/lib` and the `local/include` entries.

```
uncompress cashZZZZ.tar.Z
tar xvof cashXXXXX.tar
cd cashXXXX/lib
vi makefile (change the install entry)
make install
cd ../examples
vi makefile (change the local/lib and local/include entries)
make life
life
```

This uncompresses and extracts the tarfile, makes the library and installs the library and the include file at the place you decide by the install entry.

In the CASH directory you will also find a script `cash` that calls the C-compiler with the appropriate libraries. Edit it and copy it to your `local/bin` directory.

Contents