Modeling Lattice Gases Using Cellular Automaton

AiS Challenge

Final Report

April 3rd, 2002

076

Santa Fe High School

Team Members:

Jason Lowry

TD Gonzales

Michael Gomez

Elizabeth Sobel

Teacher:

Anita Gerlach

Project Mentor:

Chris Moore

Bob Walsh

Executive Summary:

Our program simulates the movement of lattice gases using cellular automaton. The basic format is as follows. An array of identical cells to which rules can be applied, a cellular automaton, is set up. These cells interact with each other to model many scenarios: in our case, fluid movement. Cellular automaton consist of three basic parts:

State: a variable for each cell (In our case, cells have a molecule variable placed on them to represent a real water molecule. So each cell will be a 0 or a 1 depending on the presence of a molecule)

Neighborhood: the set of cells that a given cell interacts with. Usually the adjacent cells. In our case, the other “molecules” around a given molecule.

Program: the set of rules that govern each cells movement and interaction with other cells. (In our case, we have set up rules such as which way the molecules move if a collision occurs with another molecule.

This program outputs variables to a text file, which is then inputted into a Visual Basic program. This program creates a visual representation of these variables.

Problem Statement:

The problem that we investigated was modeling the movement of lattice gas over a two dimensional surface.

Description of method used to solve the problem:

The problem was finding an effective model of lattice gases that was simple, feasible, and accurate at the same time. There were many different ways to approach the problem; including traditional modeling methods, completely graphics-based simulators, etc. We chose cellular automata because they provided a relatively easy-to-program and accurate interface. The problem with cellular automata is the sheer amount of time it takes to calculate and go through the frames, especially when the lattice is as big as ours will be (1000 x 1000). Much processing power is needed to run these simulations.
Another problem with the program in general is the difficulty of writing graphics in C++; we overcame this by using Visual Basic, writing a separate program that used output from the C++ program. This meant, however, that two separate programs would have to be used in order to create a complete simulation.
Although the entire simulation is not complete (additional work must be completed on the graphical sector), we have already seen that water can be successfully modeled using lattices. This structure and order is not often attributed to water, but we found out that we could simulate it successfully using that structure.
This is the C++ program, complete with notes and comments on specific functions of the program.

```
#include <iostream.h>
#include <cstdlib>      // Allows for the rand and srand commands to be used
```

```
void main()
{

	int arr[10][10];  // The inital array is a 10X10 for testing purposes
	int test[10][10]; // Tests to see if the bit has moved
	int tc, te, y=0, x=0;   // Total Cycles, y cord, x cord, alternate X

	cout << "How many cycles do you want to run?";
	cin >> tc;                          // inputs the number of frames

	for(int t=0; t<tc; t++) // controls the number of frames
	{
		for(int k=0; k<=99; k++)
		{
			x=k%10;
			y=k/10;
			test[x][y]=0;
		}

		if(t==0)
		{
			for(int z=0; z<=99; z++)	// Generates the frame
			{
				x=z%10;			// These two numbers should
reflect the array size if its 10 they are 10 100 = 100 etc.
				y=z/10;

				if(x<1)
				{
					arr[x][y]=rand()%2;	// This assigns the random
variable
				}

				else
				{
					arr[x][y]=0;
				}

				cout << arr[x][y] << "  ; // Prints out the location and
value
}

}
cout << endl << \nThis is the end of the frame with out movement\n\n";

		for(int p=0; p<=99; p++)
		{
			x=p%10;
			y=p/10;

			if((arr[x][y]==1)&&(test[x][y]==0))
			{
				te=1;
			}
```

```
                    else
                    {
                          te=0;
                    }

                    switch(te)
                    {
                    case 1:

                          if(y==0)
                          {
                                arr[x][y]=0;
                          }

                          else
                          {
                                arr[x][y]=0;
                                y=y+1;
                                arr[x][y]=1;
                                test[x][y]=1;
                                y=y-1;


                          }

                    break;
                    }

                    cout << arr[x][y] << "  ;
}
cout << endl << \nThis is the end of the frame " << t << "\n\n";
      }
}
```

The results of our study:

So far we have managed to get the program to out put the values and the actual simulation. The conclusions that we have come to is that the molecule will continue to move in a strait line until disturbed by an obstacle of some sort, such as another molecule.

The picture below is a screen shot of the output in text format. Before it is written into a file and changed into an image that the human mind can easily comprehend. In the picture below you will see an array with 0’s and 1’s. The 0 represents the absence of a molecule, or in other words the 0 is where there is no molecule present in that particular space. The 1 represents the presence of a molecule in that space.

10 10

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

This is the end of the frame without movement

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

frame 1

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

frame 2

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

frame 3

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

frame 4

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

frame 5

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

frame 6

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

frame 7

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

frame 8

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

0 0 0 0 0 0 0 0 0 0

frame 9

0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0

The picture that you see right here is a screen shot of the output after it was translated and read by the visual basic program and outputted onto the screen and changed into a form that the human mind can easily comprehend. The visual basic program will read the output from the c++ program and put dot where there is a 1, and nothing where there is a 0.

The software, references, tables, and other products of your work:

The software that we are using is visual c++, and visual basic. As you know these two things are what creates, and generates the output. Other than those two pieces of software, we have not used anything else.

Our most significant original achievement on the project:

What we think is the most significant and original achievement on our project that we have accomplished, and what we are very proud of is the fact that the program is written in two different programming languages (visual basic, and c++). Another thing that is original is the code, it was all written by our very own T.D. Gonzales with no references to other programs, he wrote the C++ section of the code. The visual basic part of the program was written by Michael Gomez, and he also designed an original program for our visualization part of the program.

Acknowledgment of the people and organizations that helped us:

We are grateful for the help, and support of our mentors, Chris Moore and Bob Walsh. With out them we would still have absolutely no project and would be lost in the dark. We also thank all of our family members for the support needed to finish this project.