Parallel Implementation of Terrain Erosion Applied to the Surface of Mars

Bedřich Beneš Rafael Forsbach

ITESM Campus Ciudad de México
beda@campus.ccm.itesm.mx

Abstract

We present a parallel version of the algorithm that simulates thermal erosion [7]. We split the input data into the strips that are assigned to different processes and run the erosion in parallel. When this task is finished the processes exchange information about the material transported through their boundaries, update their data, and run new erosion step. We use message passing for process synchronization. First, the information about the material transported through the boundary is saved into files and then messages are sent to the corresponding processes informing that their data is ready. Using files for process synchronization makes the implementation platform independent.

The parallel version is stable and runs very well on large data. We have achieved speedup 8.4 on ten CPUs. With small data the method is devoting high effort to communication and the speedup decreases. We have tested this algorithm on the 3D map of martial surface obtained from Mars Global Surveyor [8].

CR Categories: I.3.1 [Computing Methodologies]: Computer Graphics—Parallel processing I.3.5 [Computing Methodologies]: Computer Graphics—Physically based modeling

Keywords: Terrain erosion, Parallel algorithms, Volume graphics

1 Introduction

Nowadays, we are receiving data from spacecrafts visiting distant parts of the solar system. Processing and analyzing the data presents a great problem. Mars Global Surveyor spacecraft (MGS) has approached Mars in September 1997 and one of its tasks is to measure altitude of the martial surface, providing a 3D topographical map of Mars. The MGS uses Mars Orbiter Laser Altimeter (MOLA) to achive this (see the MOLA web page [8] for details).

An important problem is modeling of the geological behavior of planets. Visual simulation stresses visual plausibility as a proof of correctness of a simulation. That is why many algorithms for generating visually plausible models for purposes of 3D graphics are getting closer to physical simulations. Unfortunately time and space demand of these techniques leaves the majority of them practically unusable for modeling.

We are simulating erosion of the data obtained by the MGS spacecraft. Their amount, and the time and space demand of the algorithms, has led us to parallel implementation on a multiprocessor platform. We have implemented the algorithm of thermal erosion [7]. The purpose was not to implement precise and complex algorithm, but to run an existing one in parallel. Our approach can be thought as a framework for implementation of similar erosion algorithms [3, 5].

After the section introducing the previous work, we describe the thermal erosion algorithm and make its analysis. The Section 4 describes the concept of the parallelization of the algorithms and in the next section we describe the implementation. The Section 7 describes results of our simulation and limits of the parallel version of our algorithm. The last section of the paper presents opened questions and the future work.

2 Previous Work

Musgrave in 1989 [7] has introduced probably one of the first approaches to erosion of artificial terrains. The purpose of this erosion model is to improve visual plausibility of fractal surfaces. The algorithm is described in Section 4.

The time demand of erosion algorithms has been focused in [2]. The authors are using semi-adaptive algorithm that leaves the non-important parts and is eroding only the areas with a high importance.

Another physically based model has been introduced by Chiba et al. [3]. The running water defines a velocity field that is used for terrain erosion. The energy of the running water is evaluated and is applied to change the shape of the terrain.

The latest works of Musgrave [4, 6, 7] do not primarily simulate erosion-based models but generate landscapes by blending noise functions (Perlin, fBm, etc.). These approaches give visually plausible results. On the other hand the author is aware that the erosion is a much stronger tool but the number of parameters and the time required to run the simulation is making these techniques practically unusable. The second argument leads to the idea of parallel implementation of these algorithms. An analysis of the erosion algorithms shows that the process has some good properties that help us to implement the algorithm in parallel.

Many soil erosion algorithms have been also published in the journals focusing remote sensing (see for example [5]). The algorithms are mostly based on moving some material in a discrete matrix from cell to cell according to more or less elaborated physically based model. We can see the same schema here and it could be implemented easily using approach presented in this paper.

3 Erosion Algorithms

Algorithms for soil erosion simulation use similar data representation. The terrain consists of a two-dimensional elevation grid composed of so called cells (see Fig. 1).

Each cell of the grid is an approximation of the underlying area, i.e., one cell can represent area of $n \times n$ meters. The value of the cell is a statistical representative (typically the average, or one sampled value) of the underlying area. The cell has also a set of properties

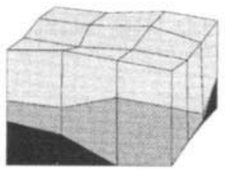


Figure 1: Example of one cell (in the middle) and its neighbors

necessary for the erosion, namely amount of water and soils that it contains, altitude, average gradient plane that corresponds to the slope of the surface, etc.

The cell communicates with its eight neighbors by exchanging information about the transferred material. Depending on the amount of water and the gradient of the cell, some material is entering the cell and some is leaving. The material is either dissolved in water, or it can be transported in a form of dust or mud etc. depending on the particular simulation. The flow of one cell can be characterized by the differential equation:

$$Q_{in} - Q_{out} = \frac{dS}{dt},$$

where Q_{in} is the incoming flow, Q_{out} is the outlet, S is the volume of transferable material stored in the cell, and t is the time.

The algorithm of [7] manipulates the material in the volume directly (see next Section) whereas the algorithm of Chiba et al. [3] uses water as a medium that transfers the material. Algorithms published in [5] transfer both, water and the material.

Benes [1] has extended the data representation by a layered data structure. Instead of representing the volume as a set of voxels the terrain is supposed to be consisting of layers of equal material as shows Fig. 1. This representation has the same properties as voxels but has smaler demand on the memory. We use this data representation in our simulations.

4 Thermal Erosion Algorithm Analysis

The algorithm works on a 2D array of cells. Each element of the array keeps complete information about the underlying layers. Example of one element and its neighbors is given in Fig. 1. The two leftmost elements in the front row keep three underlying layers, whereas the rightmost element in the front has just two layers.

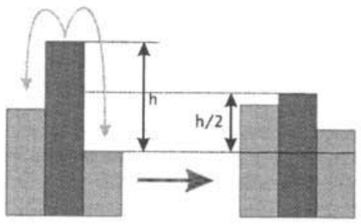


Figure 2: Principle of the thermal erosion. One half of the highest gradient of the mid-element is proportionally distributed to its meighbors

The thermal erosion algorithm [7] distributes portions of material depending on the height of the element (Fig. 2). The material is broken-up due to thermal shocks, i.e., due to changes of the temperature during the day and night. Part of the material from one cell simply falls in the direction of the highest gradient i.e., it is distributed to the neighbors. To simulate inner friction of the material, so called talus angle is introduced. If the gradient is smaller than the certain predefined value, the material is not transported.

The total volume outlet S of a cell is given by the equation:

$$S = \frac{rha}{2}$$

where h is the highest difference to the neighbors, $0 < r < 1$ is the resistance coefficient and a is the surface area of the cell.

The algorithm processes each cell of the terrain and measures highest gradient to its neighbors. Half of this value gives the portion of the material that can be moved. This is the maximum that can be moved and this corresponds to completely weak material with very low inner friction. If the material is harder, this portion is multiplied by coefficient r that corresponds to the resistance. The volume to be removed is $ah/2$. The volume that is moved to each neighbor is determined by the gradient of the cell i.e., the neighbor with highest gradient will receive the greatest portion of the material.

Physical meaning of the previously mentioned talus angle is following. If the gradient given by the explored cell and its neighbor is smaller than the corresponding talus angle, the material is not transported to this position. If the potential energy of the material cannot overcome the inner friction, the material is not removed.

Since the represented terrain consists of different layers, we can easily simulate erosion of different materials. Figure 3 from [1] shows an example of the terrain erosion. The terrain consists of two different materials. One, on the first frame on the top, is very weak and is eroded easily. The underlying material cannot be eroded and remains fixed. The time demand of the algorithm is very high.

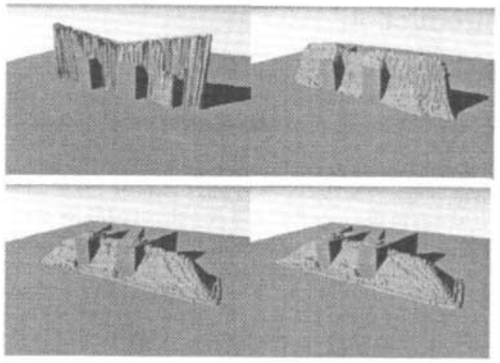


Figure 3: An example of layered erosion [1]

We have to process each cell, evaluate gradients, and move some volume of the material. Moreover the terrains are usually huge. On the other hand the algorithm itself has some good properties for example the computational time needed for evaluation of one cell is almost constant. The two extreme cases that can happen are completely flat area, where the algorithm performs no material transport, and very noisy terrain, where the material is transported from each point to each side. In very unruly terrains, like the ones obtained from real data, we can assume that both limiting cases of the cells are represented with equal probability. This means that the computational time needed for evaluation of certain amout of cells is constant.

5 Parallel Implementation

5.1 Data Subdivision

One important characteristic of the erosion algorithm is that it can be also described as a process communication. Cells can be thought of independently and can be treated as communicating computational units. Figure 4 schematically shows this view. The cell communicates with its neighbors using duplex channel that transfers the material for each layer.

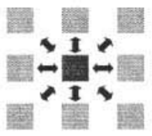


Figure 4: Material transfer among one cell and its neighbors

Moving on a higher level, we can divide the underlying surface to some smaller areas that communicate with their neighborhood. We can group some cells, evaluate their input and output and let them communicate. This, together with the above-mentioned facts about the almost constant computation time needed for one cell, has led us to the following parallel implementation.

We divide entire extent into equally long and wide strips. Each strip is hosted by one process (in the best case also by one CPU) and the erosion algorithm is run in parallel. When the erosion of each strip is computed, the processes exchange data, recompute the boundaries, and then processes continue with the next erosion step.

5.2 Algorithm

This parallel implementation can be described as follows:

1. Split the terrain into n strips.
2. Evaluate one erosion step for each strip in parallel.
3. Exchange information about the material passed through the boundary areas in parallel.
4. Update the data in parallel.
5. Go to step 2.

The advantage of this algorithm is that each computational unit keeps the portion of the data and we exchange just the boundary information. We do not need shared memory and each computational unit uses independent memory. This allows us to implement our algorithm in a heterogeneous environment. Dividing the data into the strips is done just once at the beginning.

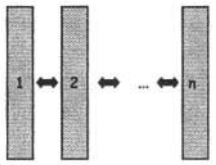


Figure 5: Data subdivision

Second reason for division the data into the strips is the communication demand. In this case each process communicates at most with the two neighbors. Because there is certain time needed for synchronization, i.e., when one process finishes before another one it cannot continue without updating the boundary and has to wait for data, we want to diminish the communication as much as possible. Figure 5 shows the data subdivision, each strip is hosted by one process and the arrows show their communication. The first and the last process communicate only with one neighbor, whereas the processes keeping inner strips have to exchange some data to both sides.

5.3 Process Synchronization

Each process runs the erosion independently to the neighbors. When all the cells processed by one computational unit are eroded, the process waits for the data from the neighbors and sends its data to them. We are using message passing for this purpose. This is implemented as a file generation that assures the system to be running on any platform (even Windows98).

After finishing the computation of the strip the process saves the boundary data on disk (in our implementation the largest file had 1M) and sends message to the neighbor. Schematically, for the process keeping the inner strip, this can be written as follows (this is detailed description of the steps 2-4 from the previous section.):

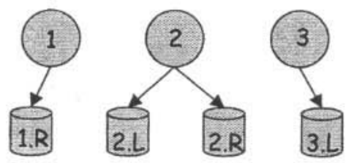
1. Erode the strip.
2. Save the left boundary.
3. Send message that the data is ready to the left neighbor.
4. Save the right boundary.
5. Send message that the data is ready to the right neighbor.
6. Wait for message from the left neighbor.
7. Read the data generated by the left neighbor.
8. Erode the left boundary.
9. Wait for message from the right neighbor.
10. Read the data generated by the right neighbor.
11. Erode the right boundary
12. Go to the step 1.

The leftmost and the rightmost processes do not perform step 2, 3, 6, 7, 8 and 4, 5, 9, 10, 11 respectively, because the left, resp. the right neighbor does not exist. It is important to save the data first and then to wait for the message otherwise the processes will deadlock. There is no need for a message delivery confirmation. The process just saves the data and sends the message that the data is ready.

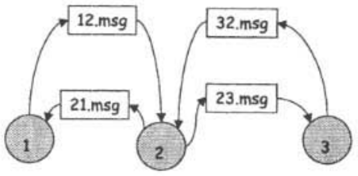
The processes are identified by the number of the strip that host (i.e., $1, 2, \ldots, n$), where n is the number of the running process. After erosion of the entire data every inner process saves two files, called $i.L$ and $i.R$, where i is the number of the process and L resp. R signifies the left and the right boundary. This corresponds to the steps two and four from the algorithm. Then the messages informing the neighbors are sent (steps three and five).

We have implemented the message passing in the form of files as well. We could use IBM POE library, but we want to have system that is platform independent and simple. Each file (that has zero size) is called $ij.msg$, where number i describes the sender and j the receiver of the message. So for example the seventh strip will be expecting data from the left neighbor in the file $6.R$ and it will be waiting for existence of the file $67.msg$. At the same time, if this is the process hosting the inner strip, it will generate two files $7.L$ and $7.R$. Then it sends two messages, $76.msg$ for the process six informing that the data from the left boundary is ready, and $78.msg$ for the process eight. It is important to note that the files with data cannot be used as messages, because the waiting process could try to read to the file into which the other process is just writing. So our message passing can be thought as a form of locking the files.

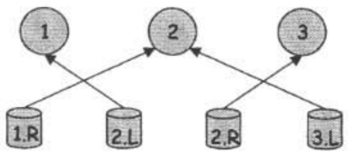
The following example with three processes should explain better the synchronization. The image shows the situation when all processes have finished their work and have saved the material from



their boundaries. The leftmost process (denoted by one) has generated file $1.R$, the inner process (two) has saved files $2.L$ and $2.R$ and the process three has saved file $3.L$. Then, as displayed in the next figure, all processes generate messages informing the neighbors about the existence of the files with the material from the boundaries.



Here the process one has sent message $12.msg$ to the process two. This process has sent message $21.msg$ to the first process and message $23.msg$ to the process three. Finally the last process has sent message $32.msg$ to the second process. Sending the messages



about the availability of the data causes reading the files with data by the corresponding processes as displayed in the last figure. This kind of synchronization has an important advantage. Since all processes will finish at finite time no deadlock can occur.

The synchronization by files is simple to implement and has also one important practical advantage. Since the waiting for the message is active, the operating system will probably not dealocate the process from the CPU that can occur in the case of the passive waiting. This means that the process will remain with high probability at one CPU and this will not delay the synchronization of the other processes and the entire system.

6 Implementation

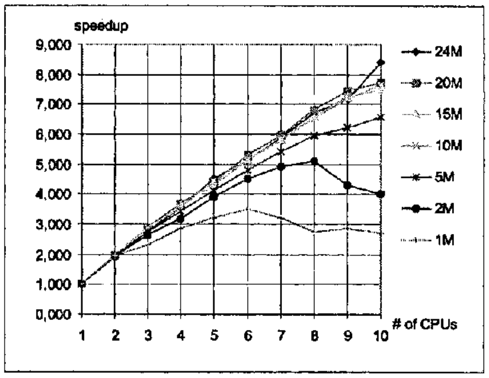
The program was implemented in C under IBM AIX version 4.3. We have run it on IBM S80 R6000 with 12 CPUs RS64-III equipped by 18GB of the operating memory.

The entire system is consisting of three groups of algorithms. First is the basic set of routines manipulating the data. We have programs for manipulating the strips - one for splitting the original data and another one for gluing the strips back for further processing. We have also implemented set of routines for converting the data from the NASA EGDR data format [8] into our inner representation. We have programs converting this into the data suitable for displaying (we use OpenGL for previews and POVRay for ray-traced images). The second group of programs includes the UNIX scripts running all the processes. The third is the erosion algorithm itself. As mentioned above, the program reads the strip just once, at the startup, and keeps it in the memory.

7 Results

7.1 Speedup

We have run 100 erosion steps of the surface of Mars. The graph and the table in the Fig. 6 show the results. All values are expressed



CPUs\size	24MC	20MC	15MC	10MC	5MC	2MC	1MC
1	1	1	1	1	1	1	1
2	1,91	1,96	1,97	1,97	1,96	1,94	1,94
3	2,79	2,88	2,82	2,90	2,73	2,63	2,30
4	3,53	3,67	3,61	3,55	3,44	3,19	2,87
5	4,51	4,39	4,31	4,44	4,10	3,92	3,20
6	5,15	5,32	5,14	5,17	4,81	4,51	3,52
7	5,88	5,96	5,80	5,81	5,41	4,90	3,22
8	6,71	6,82	6,56	6,55	5,98	5,10	2,76
9	7,15	7,43	7,25	7,20	6,22	4,28	2,85
10	8,40	7,74	7,47	7,66	6,57	4,02	2,72
1 CPU [min]	159	133	98	67	33	17	7

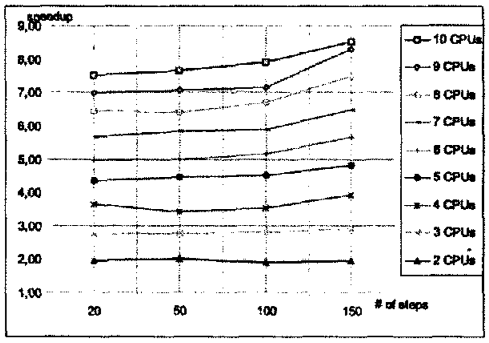
Figure 6: The speedup as a function of the number of CPUs and size of tha data. The last row of the table shows runtimes of the sequential version and the other values are expressed relatively to this value

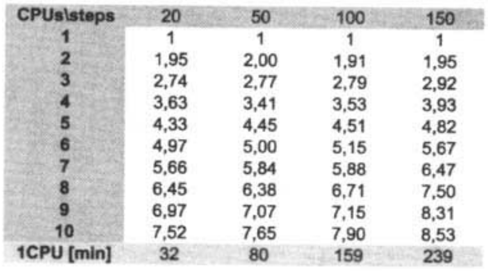
relatively to the speed of the sequential version and the last row shows the real runtimes of the sequential version in minutes. The measured tests were done on 24 million cells (denoted by MC). Corresponding size of the file is 98MB and the sequential version of the erosion runs 2.7 hours. We have also tested the system on 20MC, 15MC, 10MC, 5MC, 2MC and 1MC. All measurements were done three times, values differing more than 20% were excluded, and the average value is displayed. The total runtime of the tests was 8 days. During these tests the computer was totally devoted to the project and no other computations have been done there. This helped clarify the results, because the amount of swapping, cache misses, etc. caused by other running processes was not influencing the computations.

The speedup depends on the communication and it shows that the idea of decreasing the amount of communication with dividing the data into strips is correct. The small amount of data means a little computation and high communication overhead needed for the synchronization of the processes. This is apparent from the graph. For small data, like 1MC, it makes no sense to run the program on more than four or five CPUs. The speedup goes down very fast because of the communication overhead. On the other hand, the speedup is almost linear for huge data. Of course this is true as far as we can measure. Certainly with higher number of CPUs the speedup will go down as well, because it will drown in communication. For 24MC the speedup was 8.4 on 10 CPUs.

7.2 Number of the erosion steps

We were also interested on how the speedup depends on the number of iterations. We have measured the speedup after 20, 50, 100 and 150 iterations for two till ten CPUs. We ran the test on all data we have, i.e., 1MC, 2MC, 5MC, 10MC, 15MC, 20MC and 24MC. We have performed the test three times and computed average of the obtained values. The graph and table in the Fig. 7 show the result.





CPUs\steps	20	50	100	150
1	1	1	1	1
2	1,95	2,00	1,91	1,95
3	2,74	2,77	2,79	2,92
4	3,63	3,41	3,53	3,93
5	4,33	4,45	4,51	4,82
6	4,97	5,00	5,15	5,67
7	5,66	5,84	5,88	6,47
8	6,45	6,38	6,71	7,50
9	6,97	7,07	7,15	8,31
10	7,52	7,65	7,90	8,53
1CPU [min]	32	80	159	239

Figure 7: The speedup as a function of the number of erosion steps and used CPUs. The speedup increase as the surface flattens.

As mentioned above, the communication is higher for higher number of processors as well as for wild terrains. The erosion smoothes the terrain, that leads to the equalization of the runtime of each processing unit. This is clearly visible from the graph. For low number of CPUs (2-4) the speedup is almost constant, regardless to the number of iterations. With higher number of CPUs, the speedup goes up with the number of iterations.

8 Conclusions and Future Work

We have shown that the recently published erosion algorithms simulating soil erosion can be implemented in parallel. We have implemented thermal erosion algorithm as a representative of these techniques. We divide the terrain data into blocks that are run in parallel. After each erosion step the host processes synchronize the data by exchanging the material transported through their boundaries and sending messages informing neighbors that the data is ready.

The speedup of this algorithm is higher for larger data, because for the small one the communication delays the entire computation. For the huge data the communication is not so frequent.

One of the ways to improve the speedup of the algorithm is by elaborating the assumption that the computational time for each cell is constant. This is certainly not true and this could be taken into account. We could make some statistics of the data distribution and run adaptive splitting of the data. More wild terrains would be split into small parts, because here the transport of the material is more intensive.

Another option is to incorporate some kind of dynamical load balancing. The structure of the terrain will change over time, but this should be easy to keep track about the transported material. This information can serve as an input of some critical function. If the value of this function exceeds predefined threshold the load of the host CPU is too high and portion of the data should be moved to the neighbor. The question is how high would be the overhead for measuring the amount of transferred material, evaluating the critical function, and redistributing the data again.

This work is supported by the CONACyT grant No.36172-A.

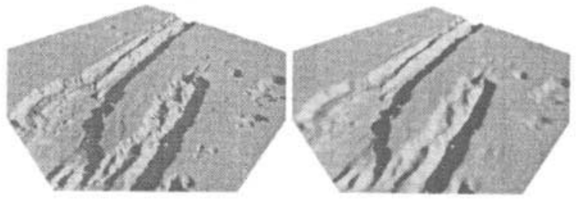


Figure 8: Raytraced images of Valles Marineris before (left) and after 150 erosion steps. The image captures area 1000x1000km. One cell corresponds to 14km^2.

References

[1] B. Beneš and R. Forsbach. Layered Data Structure For Visual Simulation Of Terrain Erosion. In *IEEE Proceedings of the Spring Conference on Computer Graphics - SCCG'01,Bratislava*. IEEE Computer Society, 2001 in print.

[2] B. Beneš, I. Marák, and P. Slavík. Terrain Erosion Model Based On Rewriting of Matrices. In *Proceedings of The Fifth International Conference in Central Europe on Computer Graphics and Visualization*, pages 341–351. University of West Bohemia Press, 1997.

[3] N. Chiba, K. Muraoka, and K. Fujita. An Erosion Model Based On Velocity Fields For The Visual Simulation Of Mountain Scenery. *The Journal of Visualization and Computer Animation*, 9:185–194, 1997.

[4] D.D. Eckbert, F.K. Musgrave, P. Prusinkiewicz, J. Stam, and J. Tessendors. Simulating Nature: From Theory To Applications. *SIGGRAPH 00 Course Notes*, pages 1–213, 2000.

[5] C. Gaillard, F. Zagolski, and F. Bonn. Modelling Of Human Dimension On Soil Erosion Processes For Remote Sensing Applications. In *IEEE Geoescience and Remote Sensing Symposium*, volume 4, pages 2137–2139, 1997.

[6] F.K. Musgrave. Towards The Synthetic Universe. *IEEE Computer Graphics and Applications*, pages 10–13, 1999.

[7] F.K. Musgrave, C.E. Kolb, and R.S Mace. The Synthesis and Rendering Of Eroded Fractal Terrains. *SIGGRAPH 89 Conference Proceedings*, 23(3):11–1–11–9, 1989.

[8] NASA. http://ltpwww.gsfc.nasa.gov/tharsis/mola.html.