Load Balancing in Parallel Implementation of Hydraulic Erosion Visual Simulations

Bedřich Beneš

Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Ciudad de México,
Calle del Puente 222, 14380 México D.F
beda@campus.ccm.itesm.mx
http://paginas.ccm.itesm.mx/˜ beda

Abstract. A load balancing in a parallel implementation of an algorithm visually simulating hydraulic erosion is introduced. Water flowing over the terrain surface captures particles of material and changes their location that has a great influence to the terrain morphology. The algorithms attempting to solve this processes require a high computational effort. A data parallel approach is used. Data representing the terrain is divided into strips and the work is assigned to different computational units. The work is asymmetrical, the parts of the terrain keeping water require higher computational effort than the dry ones. The chunks of data are evaluated and assigned to each computational unit in such a way to maintain equilibrium of the work distribution. At the end of every parallel step the data is collected, load balancing evaluated, and the work is redistributed again. The computational units communicate by message passing. The surface of Mars's data obtained from NASA Mars Orbital Laser Altimeter is used and we simulate water running over the terrain. We run the implementation on IBM R6000 with 12 CPUs and the runtime of the computational units with the load balancing differs approximately 10 %.

1 Introduction

Simulations of natural phenomena take an important place among scientific tasks. There are dozens of approaches aiming to solve different problems that could be classified according to many criteria. An interesting way is to simulate the visual appearance of natural phenomena. There are two basic goals. The first is to visually prove the underlying theory, the second aims to provide visually plausible models that can be used in computer graphics, computer animation, film industry, etc.

This paper situates itself to parallel computer graphics. We are primarily interested in visually plausible results and the physical accuracy does not take the first place for us. Anyhow, to obtain visually plausible results we must obey physical laws and rules. The solution of the underlying equations is usually simplified in such a way to provide visually plausible results fast but without visual degradation. This is mainly achieved by omitting details that does not influence the resulting models more than is their visual resolution. The computer graphics community coins the terms *physics based* or *physics inspired* modeling in such cases.

2 Previous Work and Motivation

One of the important objects that the computer graphics aims to simulate is terrain. The algorithms providing these models can be divided into two important groups, the fractal-based approaches and the simulations.

The fractal-based techniques use elaborated noise functions, such as the fractional Brownian motion (fBm) [9, 10] or the Perlin's function [16], to simulate objects that have the same fractal dimension as the terrains in reality. Surprisingly, the same fractal dimension results in visual similarity. The usage of multifractals is based on using more complicated bases, but the principle is the same as in the case of fractals. Fractal-based algorithms are out of the scope of this paper. We refer the reader to the tutorial [5] or to book [16] for more details about these techniques

There are number of physics-based simulations. Musgrave *et al* [10] have described one of the first algorithms visually simulating terrain erosions. Two algorithms have been introduced in this paper – the thermal weathering and the hydraulic erosion. The first algorithm simulates the sediment runoff caused by thermal shocks of the terrain. Part of the material is deposited to different locations depending on the terrain's local gradient. The later technique is based on the fact that water can dissolve, transport, and deposit certain amount of soil. The terrain gradient controls this transport as well.

Chiba *et al.* [4] have introduced another physics-based algorithm that simulates ridges and valleys by applying forces caused by the running water to the terrain surface. Water is approximated by particles and a simple collision detection algorithm is used to simulate the erosion.

A similar technique has been described by Nagashima [11]. The main difference is that the terrain is eroded adaptively. The shape of the river is generated independently to the surface using two-dimensional fractal interpolation. Then the banks of the river are generated using physics inspired rules.

Beneš *et al.* [3] have introduced a hydraulic erosion algorithm simulating erosion caused by the sources of water saturated by soils and drying splashes. The important contribution of this paper is the simulation of the motion of material inside the water.

Simulation of natural phenomena is usually a time demanding task, so the need for the parallel implementation of these algorithms is obvious. Surprisingly, we have not found many parallel implementations of these algorithms. Some of these techniques belong, or are similar to, cellular automata (for example [3, 4, 6–8, 10, 11]) so the algorithms for parallel implementation of cellular automata [14] can be applied here.

In the paper [2] a parallel version of the thermal erosion algorithm [10] is described. Entire surface is divided into regular areas that are assigned to different computational units. Each computational unit runs the erosion step independently and when the task is done, the units exchange the amount of material that passes through the boundary.

The next section describes the erosion algorithm itself and its parallel implementation is given in the Section 4. The Section 5 deals with the results of our work and the last Section 6 concludes the paper.

3 The Hydraulic Erosion Algorithm Description and Analysis

3.1 Data Structures

The terrains used in computer graphics are usually represented either in regular two-dimensional elevation grid called a *height field* or as a three-dimensional matrix called a *voxel space*. Both representations can be compressed [1] and the simulation can be run on the compressed data. These data structures are usually converted to sets of triangles for the purpose of displaying. The sets of triangles can be processed easily using OpenGL, VRML [15], or ray tracing. Since the subsurface structures are not very interesting for the visualization of the erosion processes, we deal with the regular height fields here.

The surface is represented as a regular grid, where every cell communicates with its eight neighbors (see in the Figure 1). A cell contains information about the underlying area of the terrain, the amount of water, soils dissolved in the water, speed of the water, etc. It is supposed that these values are constant over the cell and this value usually corresponds to the average value. The exception is the elevation. We use data from NASA Mars Orbital Laser Altimeter (MOLA) [12]. In this case the elevation corresponds to one sample of this spacecraft. From these parameters, and from the fact that we know the volume of the water, we can evaluate the forces applied to the terrain.

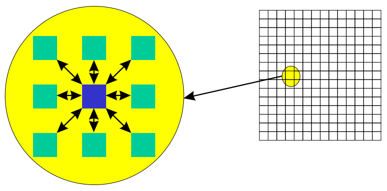
Fig. 1. One cell communicates with its neighbors by exchanging material (left)

The sequential version of the algorithm runs the erosion steps for every element and, based on the physical laws and the infbw and outlet of the grid cell, determines the material exchange. Grouping more elements into one logical group and running the erosion algorithm on one computational unit is a straightforward idea that leads to good speedup of the algorithm (see in [2]) in the case of a homogeneous distribution of the material. On the other hand, since the water is not distributed equally, the runtime of the different computational units can differ a lot. Some kind of a load balancing should be applied to approach constant runtime of each computational unit. This is the main aim of this paper.

There are many different erosion algorithms simulating different factors. The factor that has been considered as the most important to the geomorphology is water [1, 3, 4, 6, 10, 11]

3.2 Hydraulic Erosion

The hydraulic erosion process can be described by the following steps [3, 13]:

1. new water appears,
2. the water moves, erodes the underlying terrain, and captures the material,
3. the material travels into the water,

4. water and the suspended material are transported, and
5. water deposits the material at another locations.

First, water appears at some place or places. This can be due to the presence of rain, water sources, or because of the water flow. For example in the case of a water source new water appears in some cells and is then distributed to other locations depending on the local gradient of the terrain surface.

In the next step, the water absorbs some material. This can have two causes. First, the water hits the surface. Since the water has certain volume and speed, it acts with corresponding force to the surface. The material that cannot resist the force is moved [4]. Second case of the material transport is the material dissolving [3, 10].

In the step 4., water, as well as the captured sediment, is transported according to inner and outer forces. The most important factor contributing to the water flow is gravitation, although the internal forces can also play important roles.

The last step, the material deposition, is influenced by the two mayor factors [13]. First, the water slows down so the heavy particles of material cannot be carried anymore. The second factor contributing to the deposition process is the excess of the water sediment capacity. This is caused by the evaporation of the water.

Figure 2 from [3] shows results of a simulation of the above described process. The water appears after heavy rains everywhere. It deposits certain amount of soils corresponding to the sediment capacity of the water. The water travels on the surface (in fact [13] this process is *under the surface*, but this simplification does not bring any significant visual defect of the model). As the water evaporates the material is deposited at the bottom of the splash.

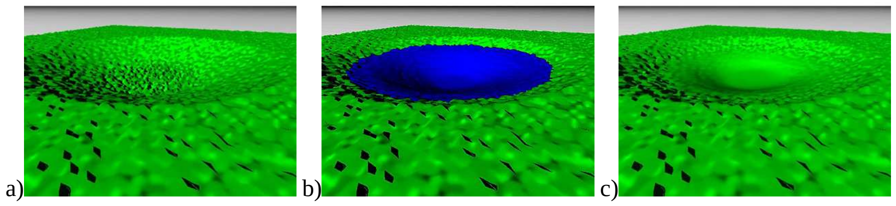

Fig. 2. A simulation of the hydraulic erosion process from [3]. a) The original surface. b) After a heavy rain the water absorbs some material that travels into the water. The water travels down and evaporates. c) As the water sediment capacity exceeds the material is deposited

3.3 Why in Parallel?

The number of operations to perform is huge. The sequential algorithm runs 100 erosion steps of an array 150×150 cells on a PC PentiumIII/500MHz for more than two minutes. This terrain resolution is too small and cannot be used for animations because the results suffer geometric aliasing a lot. The real data of the Mars surface has resolution 5000×5000 cells and the same number of steps runs for more than three hours on the same computer. The runtime vary, depending on the input parameters namely on the amount of water, but the need for some speedup is evident.

A parallel algorithm for the thermal erosion si described in the Section 1. This algorithm runs almost the same number of CPU instructions for any kind of cell. In this case the regular data subdivision is a clear and efficient solution for the work distribution leading to a good speedup. In the case of hydraulic erosion the task is different. We deal with an algorithm that is sensitive to the input data. More water slows down the runtime and the irregular distribution of water causes some areas to need higher computational effort. Apparently, some kind of data analysis and load balancing must be performed.

We do not describe details about the algorithm of the hydraulic erosion supposing the description given in the Section 3.2 is sufficient for understanding the following text. We refer the reader to [3] for details.

4 Parallel Algorithm

We use a distributed memory approach allowing us to run the program in heterogeneous environments. It would not be difficult to implememnt this algorithm on shared memory computer, but in our implementation every computational unit has its local memory and the computational units communicate using message passing. We suppose an each-to-each connection network allowing for an efficient broadcasting but a bi-directional circular connection could be sufficient.

Let's make a brief description of the parallel algorithm that is described in depth in the next section. The first step is to determine the distribution of the cells keeping the non-zero amount of water. The data is divided into parts requiring the same computational effort. This is done by one processing unit at the beginning. Then the data is distributed to all processing units. When the computation of one erosion step is done, every unit exchanges the data describing the amount of material crossing the boundary with the corresponding neighbors. This is achieved by the message passing. When the erosion steps for all computational units is evaluated and the data is updated, the data is redistributed and the new erosion step runs.

The input data of the algorithm are: the number of the computational units p, input data a in the form of matrix of dimension $x \times y$, and the parameters of the erosion. The output data is the matrix b of the same resolution. The algorithm works in the following way:

Sequential data preprocessing

1. Load the complete data into one computational unit
2. Evaluate the occupancy of the cells by water
3. *Load balancing*
   Split the data into strips maintaining constant amount of work to erode each strip

Run in parallel for each computational unit

4. Run the erosion
5. *Synchronization:*
   a) Send the boundary data to the neighbors

    b) Wait for the data from the neighbors

6. Erode the boundary data
7. Rendez vous
8. If this is not the end of simulation continue with step 3.
9. Save b

4.1 The Load Balancing

The key question is the way the data is split. The important thing is to avoid communication among the computational units, since every communication slows down the entire computational speed. The simplest division having complete coverage of a two-dimensional matrix with smallest communication overhead is into strips. Let's follow the example in the Figure 3. There is a matrix of 10×10 elements and $p = 3$. The input data is divided into three strips, the first and the second keeping 3×10 cells and the last one having 4×10.

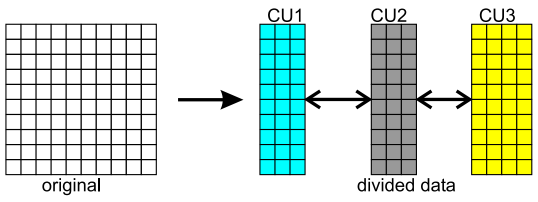

Fig. 3. Terrain surface represented in 10×10 cells is divided for a work on three computational units

As mentioned above the computational effort for a cell containing water is different than for the cells that are dry. A dry cell requires eight comparisons to detect that no neighbor keeps water. The computation of a wet cell needs the same, but also 16 more comparisons for the water distribution. The amount of work required for the cell keeping water is therefore $3\times$ higher.

Recall that the input matrix is denoted by a and has resolution $x \times y$. We compute an additional data structure – one-dimensional array h_w (the histogram) of dimension x. The h_w keeps the number of the wet cells in every column. It is clear that the number of the dry cells denoted by $h_d[i]$ in the $i-$th column is a complementary to the wet ones:

$$h_d[i] = y - h_w[i]. \tag{1}$$

It is also clear that

$$x \times y = \sum_{i=1}^{x}(h_w[i] + h_d[i]) = \sum_{i=1}^{x}(h_w[i] + (y - h_w[i])).$$

To determine the boundary of the strips we should compute the amount of work required for each strip. Let's say that the amount of work required for a dry cell is equal to one. As described above, the wet cell requires approximately three times more work to be tested. To evaluate the amount of work needed to compute erosion of the entire terrain denoted by *total* we must sum the amount of work required for each column:

$$total = \sum_{i=1}^{x}(h_d[i] + 3h_w[i]) = \sum_{i=1}^{x}(y + 2h_w[i]). \tag{2}$$

We use p computational units so the work s DC that should be done by each one is

$$s = \frac{total}{p}.$$

Once the number s is determined we just scan the histogram h_w from the left to the right and, using the equations (1) and (2), determine the boundary of the strips. It is achieved by the following piece of code

```
$acc = 0;$
 for  $(i = 0; i < x; i++)$
{
  $acc+ = (y + 2 * h\_w[i]);$
  if (acc≥s) {WriteOutBoundaryIndex(i);acc=0;}
}
```

An additional counter acc accumulates the number of DC for the $i-$th column. When the acc if greater or equal to the limit value s the boundary index is saved, indicating that here ends the data belonging to one computational unit and starts the data that owns the another one.

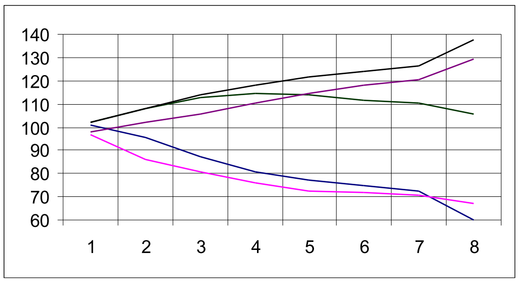

Fig. 4. Graph showing the percentage of the coverage of the terrain by different computational units during the eight steps of the simulation

It is interesting to see how the area assigned to different computational unit changes with the water distribution. This is demonstrated in the Figure 5. This image shows an OpenGL preview of an area of the Martial surface. We have selected the area of Elysium Mons that is close to the highest volcanoes of Mars and has an interesting diversity. The area is intentionally quite small, occupying 444×346 cells. At the beginning the amount of wet cells was small and as the water was distributed this number has increased significantly. We have used five computational units so the area is divided into five strips. The images show the subdivision that corresponds to the water distribution. The percentage of the are covering by the strips shows the graph in Figure 4. The images in the Figure 9 show the same scene ray-traced.

There is one special case that could cause very intensive communication. It is not worth to divide the data with very high local concentration of water. This can lead to the case when every computational unit holds just one column and this increments the communication among the units. To avoid this case, we do not divide the areas having less than an *ad hoc* constant. We use 5% of the total width of the area.

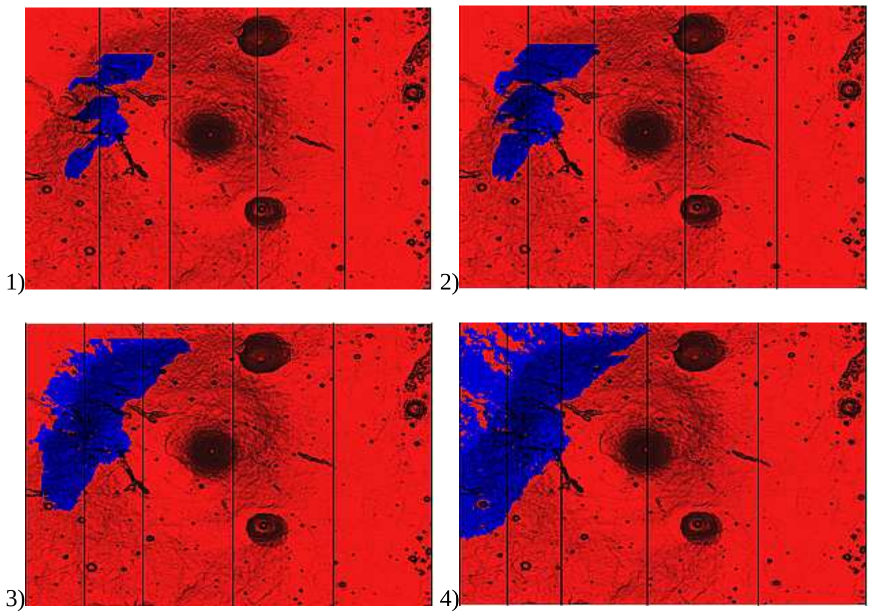

Fig. 5. OpenGL preview of a dynamic loadbalancing. Martial surface is assigned by four springs of water and the water travels on the surface eroding the terrain. The entire process is computed by five computational units so the surface is divided into five strips

4.2 Synchronization

When the erosion of one strip is done every communication unit exchanges the data passing through the boundary and erodes the boundary columns. The strips of data must be sewed together. This corresponds to the steps 5-8 from the algorithm in the introduction of the Section 4.

Let's suppose we have two strips of data assigned to two computational units, the left one is denoted by l and the other by r. To complete the erosion task we must run the erosion step for the last column of the strip l using the data from the first column of the strip r. In the next step we must do similar step for the strip r; we run the erosion of the first column of the r using the data from the last column of the strip l. This is done in parallel by sending the missing data and collecting them when the erosion step is done.

This technique is implemented by message passing. When the computational unit is done with its work it sends messages to the left and to the right and waits for the messages from the same sides as shown in Figure 6.

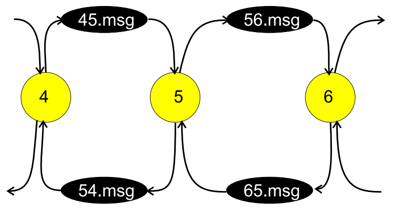

Fig. 6. Computational units synchronization at the end of an erosion step

After this step all the computational units wait for the slowest one. Then the erosion step is over and the data is completed. The next step is the sequential load balancing, corresponding to the step 8) from the algorithm in the introduction of the Section 4, and then the erosion runs again.

The synchronisation could be also done in parllel. The units inform their neighbors about the tranferred material so it is possible to evaluate the changes in the boundaries. The problem is that this could not be done in one pass, since two units can have the same requirement. In this case we should broadcast the information to each unit and they could decide what should be passed to the neighbors.

5 Implementation and Results

The entire system is implemented in C and runs under Windows and UNIX. The load-balancing part was developed and tested on IBM PC under Windows 2000 and the entire system runs under IBM AIX version 4.3.on IBM SP80 R6000 equipped with 12 CPUs RS64-III with 18GB of the operating memory.

The most important for us was the difference of the computational time of different computational units according to the assigned work. We have tested different areas of the Martial surface that were differently inundated by water. We have performed 200 erosion steps watching how the computational time required by the physical CPU of the computer was changing. The test was performed on five and ten CPUs. The entire test was repeated five times and the runtime average was used. Values of the same tests differing by more than 10% were discarded. The supercomputer was completely devoted to these tests.

The graphs in the Figure 7 show the difference of the runtime of the five and ten CPUs and the following table shows the average of the computational time during the 200 erosion steps per CPU and the difference from the optimal 100%:

CPU		1	2	3	4	5	6	7	8	8	10	**total average [%]**
5 CPUS	**average[%]**	86	93	96	94	131						100
	difference[%]	14	7	4	6	31						12.4
10 CPUs	**average[%]**	139	126	93	79	92	95	96	96	95	89	100
	difference[%]	39	26	7	21	8	5	4	4	5	11	13.1

The maximum difference for five CPUs was +31% and -14% and +39% and -21% for ten CPUs. The average of the difference was 12.4% for five CPUs and 13.1% for ten. As we can see from the numbers and from the graphs the runtimes do not vary very much confirming the analysis and the implementation of the load balancing.

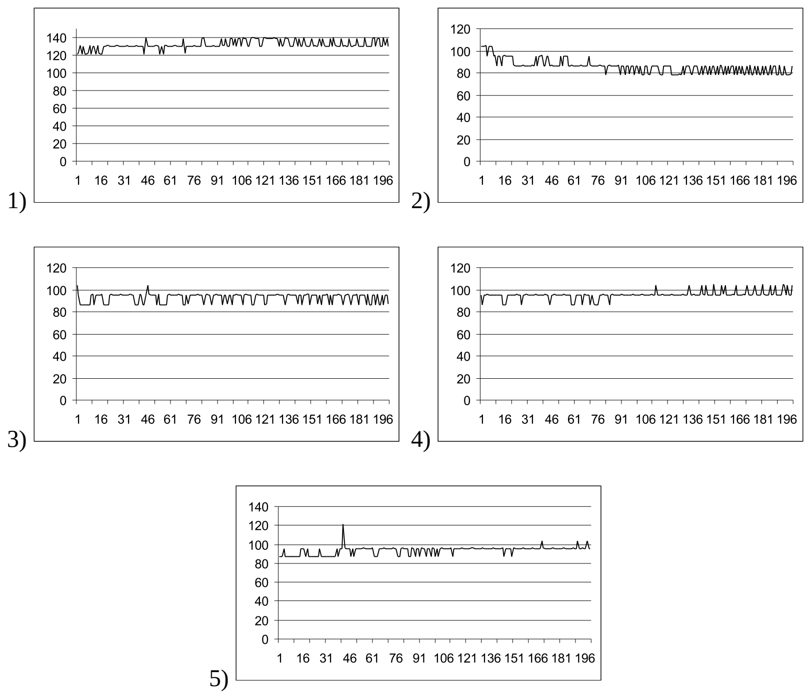

Fig. 7. Graphs displaying the percentage difference of the runtime for different erosion steps on different CPUs. We have performed 200 erosion steps (the axis x) and the percentage of the runtime is on the axis y

The question is if it is worth to implement this load balancing. Maybe the static distribution is sufficient. Answer to this question is in the Figure 4. We have captured all the data and the process of the data redistribution is quite dynamic. We can see (also in the Figure 5) that after eight erosion steps the second strip has changed its size to 67% of its originally assigned size and the last one has grown to 130%.

6 Conclusions

A load balancing in parallel implementation of hydraulic erosion visual simulations was introduced. The principal idea of the algorithm is based on the fact that different cells of the terrain require different effort to be eroded. A precomputation of the work required is done for every column of the terrain matrix and new chunks of data are distributed in such a way to keep the load balanced. When the erosion of each chunk of data is done the computational units synchronize by message passing. The data is collected and the load balancing assigns the data to the computational units again.

We have tested the algorithm on different areas of the Martial surface using the data obtained from the NASA Mars Orbital Laser Altimeter. The program runs on an IBM

R6000 equipped by 12 CPUs. The results depend on the number of CPUs used. With higher number of CPUs the communication slows down the work and the difference among the different units was up to 31% but the average of the runtime difference was just 13.1%. For five CPUs the maximum difference was 14% and the average of the difference is 12.4%

We believe the results have proven that the dynamic load balancing of the data can bring a significant improvement of the parallel implementation of this kind of visual simulations.

This work is supported by the ITESM grant No. 32419.

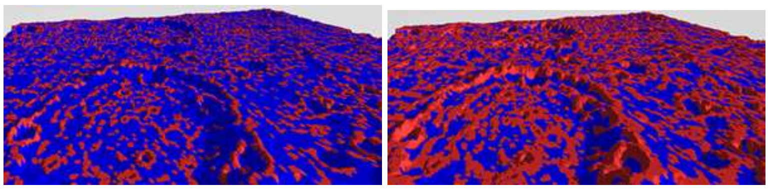

Fig. 8. Another area of Mars drying after a heavy rain

References

1. B. Beneš and R. Forsbach. Layered Data Structure for Visual Simulation of Terrain Erosion. In *IEEE Proceedings of SCCG'01*, volume 25(4) of *IEEE Computer Graphics*, pages 80–86, 2001.
2. B. Beneš and R. Forsbach. Parallel Implementation of Terrain Erosion Applied to the Surface of Mars. In *Proceedings of the Afrigraph'01*, pages 48–55. ACM Siggraph, 2001.
3. B. Beneš and R. Forsbach. Visual Simulation of Hydraulic Erosion. *Journal of WSCG*, 1(Special Issue):79–86, 2002.
4. 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.
5. D.D. Eckbert, F.K. Musgrave, P. Prusinkiewicz, J. Stam, and J. Tessendors. Simulating Nature: From Theory to Applications. *Siggraph 2000 Course Notes*, pages 1–213, 2000.
6. A.D. Kelley, M.C. Malin, and G.M. Nielson. Terrain Simulation Using a Model of Stream Erosion. *Computer Graphics*, 22(4):263–268, 1988.
7. O. Koichi and T. Nishita. A Method for Modeling and Rendering Dunes with Wind-ripples. In *Proceedings of Pacific Graphics'00*, pages 427–428, 2000.
8. I. Marák, B. Beneš, and P. Slavík. Terrain Erosion Model Based on Rewriting of Matrices. In N. Magnenat Thalmann and V. Skala, editors, *Proceedings of The Fifth International Conference in Central Europe on Computer Graphics and Visualization 97 - WSCG*, volume II, pages 341–351. University of West Bohemia Press, February 1997.
9. F.K. Musgrave. Towards the Synthetic Universe. *IEEE Computer Graphics and Applications*, pages 10–13, 1999.
10. F.K. Musgrave, C.E. Kolb, and R.S Mace. The Synthesis and Rendering of Eroded Fractal Terrains. *Computer Graphics*, 23(3):11–1–11–9, 1989.

11. K. Nagashima. Computer Generation of Eroded Valley and Mountain Terrains. *The Visual Computer*, 13:456–464, 1997.
12. NASA. http://ltpwww.gsfc.nasa.gov/tharsis/mola.html.
13. M. Newson. *Land, Water and Development*. Routledge, 1997.
14. M.J. Quinn. *Parallel computing*. McGraw-Hill, Inc., 1994.
15. M. Reedy, Y. Leclerc, L. Iverson, and N. Bletter. TerraVision II: Visualizing Massive Terrain Databases in VRML. *IEEE Computer Graphics and Applications*, pages 30–38, 1999.
16. A. Watt and M. Watt. *Advanced Animation and Rendering Techniques: Theory and Practice*. Addison-Wesley, Reading, 1992.

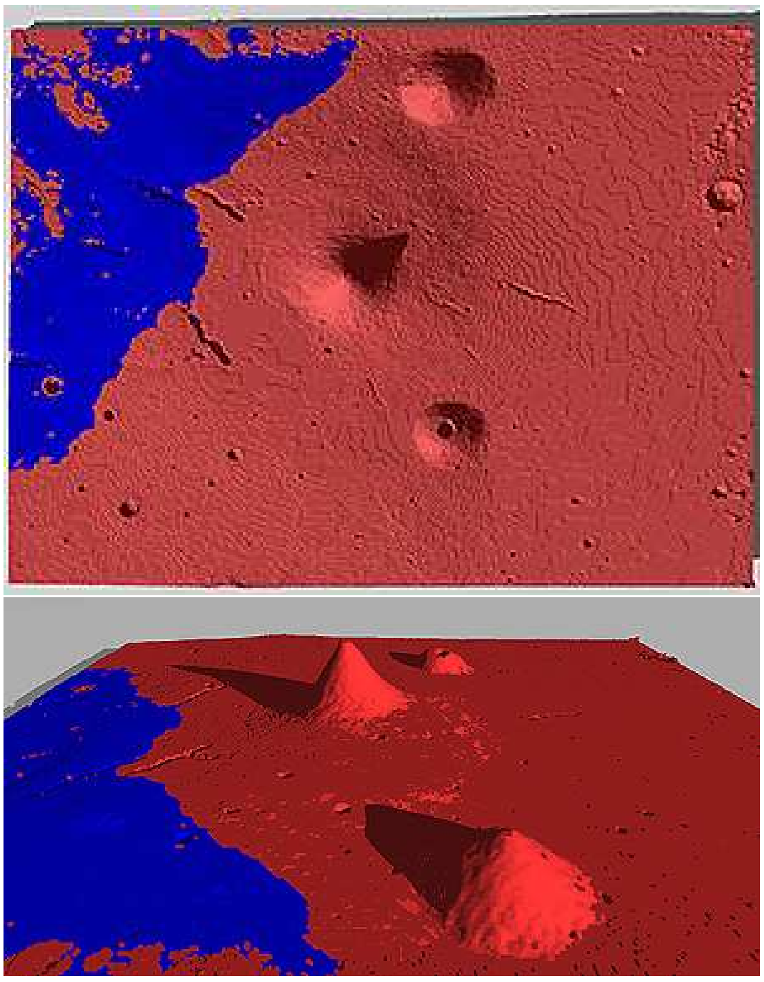

Fig. 9. The heavily inundated area of the three highest volcanoes in the Elysium Mons from the Figure 5 displayed using ray-tracing