Eurographics Workshop on Natural Phenomena (2005)
E. Galin, P. Poulin (Editors)

Interactive physically based Fluid and Erosion Simulation

B. Neidhold,[1†] M. Wacker[2] and O. Deussen[3]

[1]Lehrstuhl für Computergraphik und Visualisierung, TU Dresden, Germany
[2]Computergraphik, HTW Dresden, Germany
[3]Lehrstuhl für Computergrafik und Medieninformatik, Universität Konstanz, Germany

Abstract

Realistically eroded terrain is a base of almost every outdoor visualization for simulators or computer games. In order to achieve convincing results physically based erosion algorithms are necessary. We present a new method that combines a non-expensive fluid simulation with an erosion algorithm. Both parts are running at interactive rates so the artist is able to influence the erosion process in real-time by changing simulation parameters or applying additional water to the scene. In this way, we support realism as well as design aspects during the terrain creation process. To simplify the three dimensional fluid simulation we use a newtonian physics approach that works on a two dimensional grid storing acceleration, velocity and mass. The method provides all features that are important for simulation of erosion e.g. moving, non-moving water (rivers, lakes) and evaporation. This allows us to support effects like dissolving, transportation and sedimentation of material in the erosion process.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Physically based modeling

1. Introduction

Erosion techniques for terrain generation have been addressed in computer graphics for more than 15 years. Most of the introduced algorithms are using a physically based approach for simulating the erosion process itself as well as for the underlying dynamic processes e.g. water transportation or wind. On the other hand some algorithms generate the eroded terrain from scratch. They use modifications of the midpoint displacement approach introduced by Mandelbrot [MH82] or perlin noise. One of the first fractal based algorithms for hydraulic erosion was introduced by Kelley *et al.* [KMN88]. This generates fractal terrain around a previously generated *river network*. Prusinkiewicz [PH93] proposed a fractal algorithm to generate hydraulic eroded mountains that do not need pregenerated input data. Instead, inspired by midpoint displacement, he included the generation of rivers into the fractal algorithm. A completely procedural approach for manipulating solid objects with tools (e.g. erosion tool) was introduced by [CDM*02].

The first physically based approach to generate realistic terrains was described by Musgrave *et al.* [MKM89]. This paper describes two algorithms - thermal weathering and hydraulic erosion. The first algorithm simulates the tearing down process and distribution of sediment caused by thermal shocks. Some parts of the material are simply displaced around the actual point if the local inclination is greater than a specified material constant. The other technique is based on some characteristics of water in the erosion process. Soil can be dissolved, transported and deposited. Now in the simulation, depending on the local inclination and water amount, some underlying material is dissolved into the water. After some movement caused by a simple water transportation algorithm the material is deposited at another location. Each part of the simulation can be influenced by some material parameters. This method has been extended by Roudier *et al.* [RPP93] to use different materials. Chiba *et al.* [CMF98] introduced the first algorithm for hydraulic erosion based on velocity fields. This includes the simulation of a water, elevation, collision energy and velocity field. The water flow is simulated by applying forces caused by the local gradient. While moving, the water flows over the ground surface and dissolves some amount of material. Depositing of sediment terminates when the amount of dis-

† benjamin.neidhold@inf.tu-dresden.de

solved material reaches a certain threshold. A technique that tries to mix hydraulic and thermal erosion was published by Nagashima [Nag98]. It needs a pregenerated *river network* that is created with a two dimensional fractal function. Then, the river banks are eroded by some physically inspired rules. In 1997, Benes *et al.* [BMS97] published an hierarchical thermal erosion algorithm that simply distributes some amount of material to its eight neighbors of a simulation grid. They optimized the distribution step by downscaling the simulation grid before one erosion step and upscaling the grid afterward to speed up the simulation. In 2002, Benes *et al.* [BF02] presented an extension of his former paper to simulate hydraulic erosion by also distributing material to its eight neighbors. They added two additional layers for the amount of water and the dissolved soil. As an extension to all prior algorithms they introduced an evaporation step to simulate drying pools of water. A special feature of his implementation is the separation of the simulation steps for erosion, transportation, evaporation and deposition. So, the frequency of the independent steps can be changed to speed up the simulation, of course losing some physical exactness.

All presented techniques for the underlying water simulation have one or more of the following disadvantages: They cannot handle hollows in a way that they are filled with lakes, the water begins to oscillate because of numerical instability and huge time steps, or the simulation does not run in real-time.

In this paper, we introduce a new method for fluid simulation that runs in real-time. We describe how to solve a *3D water simulation* in a special *2D fluid solver* that can handle moving water and lakes (still water) natively. On the basis of the water simulation a special data structure incorporating several layers enables us to implement fluvial erosion processes efficiently using a set of discrete grids. The efficiency of the process in combination with a set of tools enables the user to create eroded terrain interactively for moderately complex terrains.

2. System setup

Before we go into detail for our algorithms, we describe some basic data structures in this section. A two dimensional regular height field is the simplest and most commonly used data structure for terrain visualization and environmental simulations (e.g. erosion). This is also used by 2D/3D fluid simulations [KM90] [OH95] [FM97]. The low memory usage and a fast ray test for ray-tracing are the advantages of this representation in which each grid cell stores the height value of the underlying terrain. However, not all terrain types can be converted into a height field data structure (e.g. terrain with caves).

A voxel grid is a three dimensional data structure where each grid cell stores an id of the local material type (e.g. air, water, sand or granite), so that the terrain data and the fluid data can be stored together in one grid. On the one hand, a voxel grid allows the representation of caves, but on the other hand, it requires much more storage space [CMT04] [Sta03]. Therefore, voxel-based erosion simulations are not suitable for running in real-time, yet.

Benes *et al.* [BF01] introduces a layered data structure as a trade-off between a height field and a voxel grid. For each position (x,y) a static array of attributes (material-id, layer-width) is stored. The approach is shown to be a good choice whenever only a few different layers lie on top of each other. This data structure can also be used for fluid simulation. Another approach for three dimensional fluid simulations is using particle systems which turn out to be very efficient data structures for storing the state of a fluid [MCG03]. A particle system for fluid simulation and a height field for the terrain is also a very efficient combination of two different approaches [HW04]. Sets of (irregular) triangles are a fast representation for terrain and fluid in real-time rendering for graphical information systems GIS (e.g. with OpenGL). But they are not suitable for erosion simulation in combination with a fluid because the re-tessellation that is necessary after an erosion step can be quite expensive.

In our implementation, we are inspired by the layered data structure of [BF01]. We use five layers onto our two dimensional height field H to store all relevant data for the fluid and the erosion simulation (see table 1). Each layer is represented as a two dimensional grid. For our fluid simulation we use the actual fluid amount F, velocity $\vec{v}$ and acceleration $\vec{a}$ in each discrete grid cell. The two vectors are stored in 3D because we need to obtain the correct velocity amount used by the erosion algorithm. Our erosion model uses the dissolved material layer S which stores the amount of sediment in each cell.

acceleration (3D)	$\vec{a}$
velocity (3D)	$\vec{v}$
fluid amount	F
dissolved material amount	S
terrain level	H

Table 1: *Data grids used by our simulation.*

The actual simulation is performed by applying finite differences algorithms on these layers. The layers can interact, e.g. dissolved material can elevate the terrain at a certain position. In the next section, our basic water model is presented before erosion models are introduced.

3. Water simulation

In a water simulation, we want to describe the amount of water at a given place at a given time. A large amount of research has been done over the past 50 years to solve this problem with approximations of the full 3D Navier-Stokes

equations that describe realistic animation of fluids. A semi-Lagrangian approximation for smoke-like fluids called "stable fluids" was introduced by Stam [Sta99]. Because of the fact that we do not need all of the 3D-features like multiple water layers, vertical vortices or waves these equations are simplified and then discretized to a two dimensional version. The result compared with real Navier-Stokes solvers is less physically correct but much faster and fits very good into the requirements of our erosion algorithm. The base for our approach is a system of first order differential equations

$$\dot{\vec{v}} = \vec{a} - K_A \cdot \vec{v} = \frac{\vec{F}}{m} - K_A \cdot \vec{v} \quad (1)$$

$$\dot{\vec{x}} = \vec{v} \quad (2)$$

that describes the movement of *material* depending on velocity $\vec{v}$ and acceleration $\vec{a}$. Note that in a simulation specifically for fluids material stands for the amount of water at a discrete grid cell. For complex erosion simulations we define a multidimensional *material vector* which stores for example the amount of water and dissolved sediment. Especially for erosion simulation the underlying terrain can be very rough and influences (damps) the movement of the fluid. The additional term $K_A \cdot \vec{v}$ gives us the option to control the sliding friction between the fluid and the terrain with the parameter $K_A \in [0..1]$. With the value $K_A = 0.3$ used by our simulation we emulate a very rough underlying terrain that damps the velocity of the fluid about 30 percent in one time unit. For time-discretization of the differential equations we use the explicit Euler method.

$$\vec{v}_{t+\Delta t} = \vec{v}_t + \vec{a}_t \cdot \Delta t - K_A \cdot \vec{v}_t \cdot \Delta t \quad (3)$$

$$\vec{x}_{t+\Delta t} = \vec{x}_t + \vec{v}_{t+\Delta t} \cdot \Delta t \quad (4)$$

When the actual acceleration is obtained from all internal and external forces (see section 3.1), the new velocity after the time step Δt $\vec{v}_{t+\Delta t}$ can be computed at each discrete grid cell. The new destination of the transported material $\vec{x}_{t+\Delta t}$ may not be a discrete grid cell. In order to solve this we distribute the material vector with a bilinear interpolation to the nearest four neighbors.

To this end, we must define how to handle velocities and the material vectors from different locations that are transported to the same discrete grid cell. The components of the material vectors are simply summed up, e.g. the fluid amount at the destination point F_{dest} and the transported fluid F_{add} is added to the new fluid amount F_{new}. The corresponding velocity vectors have to be mixed. The velocity at the destination point $\vec{v}_{dest}$ and the velocity of the added material $\vec{v}_{add}$ are weighted with the factor k to the new velocity $\vec{v}_{new}$ (see formula 6). This weighting factor results as the ratio of the water amount at the destination point F_{dest} and the new summed up water amount F_{new}.

$$F_{new} = F_{dest} + F_{add} \quad (5)$$

$$\vec{v}_{new} = k \cdot \vec{v}_{dest} + (1-k) \cdot \vec{v}_{add} \quad (6)$$

$$k = \frac{F_{dest}}{F_{new}} \quad (7)$$

Every fluid simulation has to include diffusion in the simulation model emulating the interaction of neighboring fluid particles. Without diffusion, the approaches based on differential equations would be instable or oscillating. To this end we insert after every water simulation step we insert a diffusion step that smooths the velocity field and all components of the material vectors. Smoothing is done by distributing the value of the actual grid cell to the four direct neighbors. This method is described in detail by Stam [Sta99].

3.1. Calculation of the acceleration

In common Newtonian Physic Systems the acceleration direction in a landscape at the position (x,y) is the direction of the biggest tilt angle α of the underlying terrain height field $I(x,y)$ calculated by the gradient $\nabla I(x,y)$. Therefore, the acceleration force can be obtained from that angle α.

$$|\vec{a}| = \sin\alpha \cdot g \qquad g \approx 9.81 \frac{m}{s^2} \quad (8)$$

$$\nabla I(x,y) = \left(\frac{\Delta I}{\Delta x}, \frac{\Delta I}{\Delta y} \right)^T \quad (9)$$

The discretization of the gradient at the position (x,y) leads us to formula (9) where ΔI is the altitude difference of two *measuring points* around the position (x,y) and $\Delta x, \Delta y$ are distances between these measuring points. For a simple approximation of the gradient it would be enough to take two measuring points in each direction x and y. To increase the precision and to better represent the contiguous characteristics of the height field we calculate the directional derivatives between the height $I(x,y)$ and all of its eight neighbors $I_1 ... I_8$ that are lower than I(x,y) Eq. 10. Please notice that in this case Δx and Δy are 1. We drop the directional derivative calculation of the higher grid cells because of the fact that water flows only to downhill cells.

$$\nabla I_{1..8} = \left\{ \begin{pmatrix} \Delta I_7 \\ -\Delta I_7 \end{pmatrix}, \begin{pmatrix} 0 \\ -\Delta I_6 \end{pmatrix}, \begin{pmatrix} -\Delta I_5 \\ -\Delta I_5 \end{pmatrix}, \begin{pmatrix} \Delta I_8 \\ 0 \end{pmatrix}, \begin{pmatrix} -\Delta I_4 \\ 0 \end{pmatrix}, \begin{pmatrix} \Delta I_1 \\ -\Delta I_1 \end{pmatrix}, \begin{pmatrix} 0 \\ \Delta I_2 \end{pmatrix}, \begin{pmatrix} -\Delta I_3 \\ \Delta I_3 \end{pmatrix} \right\} \quad (10)$$

$$\Delta I_n = I(x,y) - I_n \quad (11)$$

$$\nabla I(x,y) = avg(\nabla I_{1..8}) \quad (12)$$

To obtain a single gradient later used for calculation of the acceleration vector we use the $avg()$ function that calculates the average of all obtained directional derivatives from lower grid cells. This method (comparable to a filter kernel

in image processing) is very suitable and fast when the fluid at the actual cell has a high velocity and the local gradient has little influence on the fluid. With low or zero velocity the fluid would only be accelerated into the direction of the gradient. For a more accurate but slower simulation, the fluid is split up and accelerated with the directional derivatives of all lower neighboring cells. This is equivalent to temporally subsample the actual grid cell. For a compromise between the fast and an accurate simulation we define a threshold velocity. If the actual velocity of the water is lower than this threshold, we do not use the average function to build a single gradient but we split the fluid proportionally to the corresponding difference ΔI_n and apply the respective directional derivatives from lower cells. With this method the gradient is accurately calculated also at problematic locations. A good example is the bottom of a hollow. At that location a traditionally obtained gradient (acceleration) is not zero but our implementation calculates zero (see figure 1).

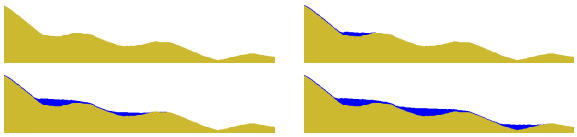

Figure 1: *Cross section through a hilly terrain. During the simulation, water fills hollows and flows downwards.*

Until now, we have not mentioned how the function $I(x,y)$ we use for gradient calculation is defined exactly. If only defined as the actual level of the underlying terrain $H(x,y)$ all fluid would flow to one of the local minima in the grid (hollows) and build pillars. This happens because in the introduced physical model the fluid at adjacent cells does not interact with each other yet. The adjacent cell interaction is obtained by integrating the fluid amount $F(x,y)$ in the definition of the function $I(x,y)$. If we define it as the sum of terrain level $H(x,y)$ and the fluid amount $F(x,y)$ we get excellent lakes, but when water runs down a mountain very fast, it is hindered by water in adjacent cells. A compromise between these two extremes is to use an influence factor $K_F \in [0..1]$ that describes how much of the water at a grid point influences the gradient calculation.

$$I(x,y) = H(x,y) + K_F \cdot F(x,y)$$

We achieved good results by using formula (13) that decreases the influence of the water amount at high velocities.

$$K_F = max(0, 1 - 0.05 \cdot |\vec{v}|) \tag{13}$$

At this point we only know the acceleration direction

$$\vec{m} = -\nabla I(x,y)$$

along the x and y axis that points in the opposite direction of the gradient $\nabla I(x,y)$. To obtain the complete three dimensional direction $\vec{M}$ we also need the component along the z axis.

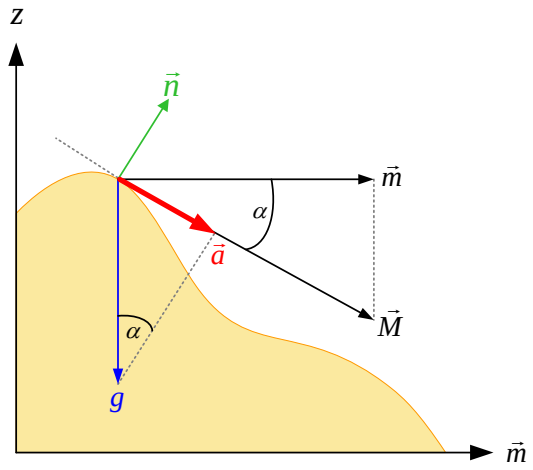

Figure 2: *Calculation of Acceleration*

$$\vec{n} = \left(+\frac{\Delta I}{\Delta x}, +\frac{\Delta I}{\Delta y}, -1\right)^T \tag{14}$$

$$\vec{n} \cdot \vec{M} = 0 \tag{15}$$

$$\vec{M} = \left(-\frac{\Delta I}{\Delta x}, -\frac{\Delta I}{\Delta y}, -\frac{\Delta I^2}{\Delta x^2} - \frac{\Delta I^2}{\Delta y^2}\right)^T \tag{16}$$

As shown in figure 2 vector $\vec{M}$ is the projection of $\vec{m}$ to the plane sloped by the tilt angle of the underlying terrain. With the plane normal $\vec{n}$ also obtained by the gradient we can calculate the vector $\vec{M}$ by equation (16). For the desired acceleration amount $|\vec{a}|$ (equation (8)) we can now use a simple equation for $\sin\alpha$.

$$\sin\alpha = \frac{\left|\vec{M}_z\right|}{\left|\vec{M}\right|} \tag{17}$$

$$\vec{a} = \frac{\left|\vec{M}_z\right|}{\left|\vec{M}\right|} \cdot g \cdot \frac{\vec{M}}{\left|\vec{M}\right|} \tag{18}$$

As you can see the complete three dimensional acceleration vector $\vec{a}$ is the product of the acceleration amount and the normalized acceleration direction vector $\vec{M}$. This result is used by the discretized differential equations in the fluid simulation.

4. Hydraulic erosion function

If water flows over the given terrain, material such as soil and stones is dislocated and transported to lower regions. To simulate this effect we have to define a hydraulic erosion function. This function is used whenever a part of a material vec-

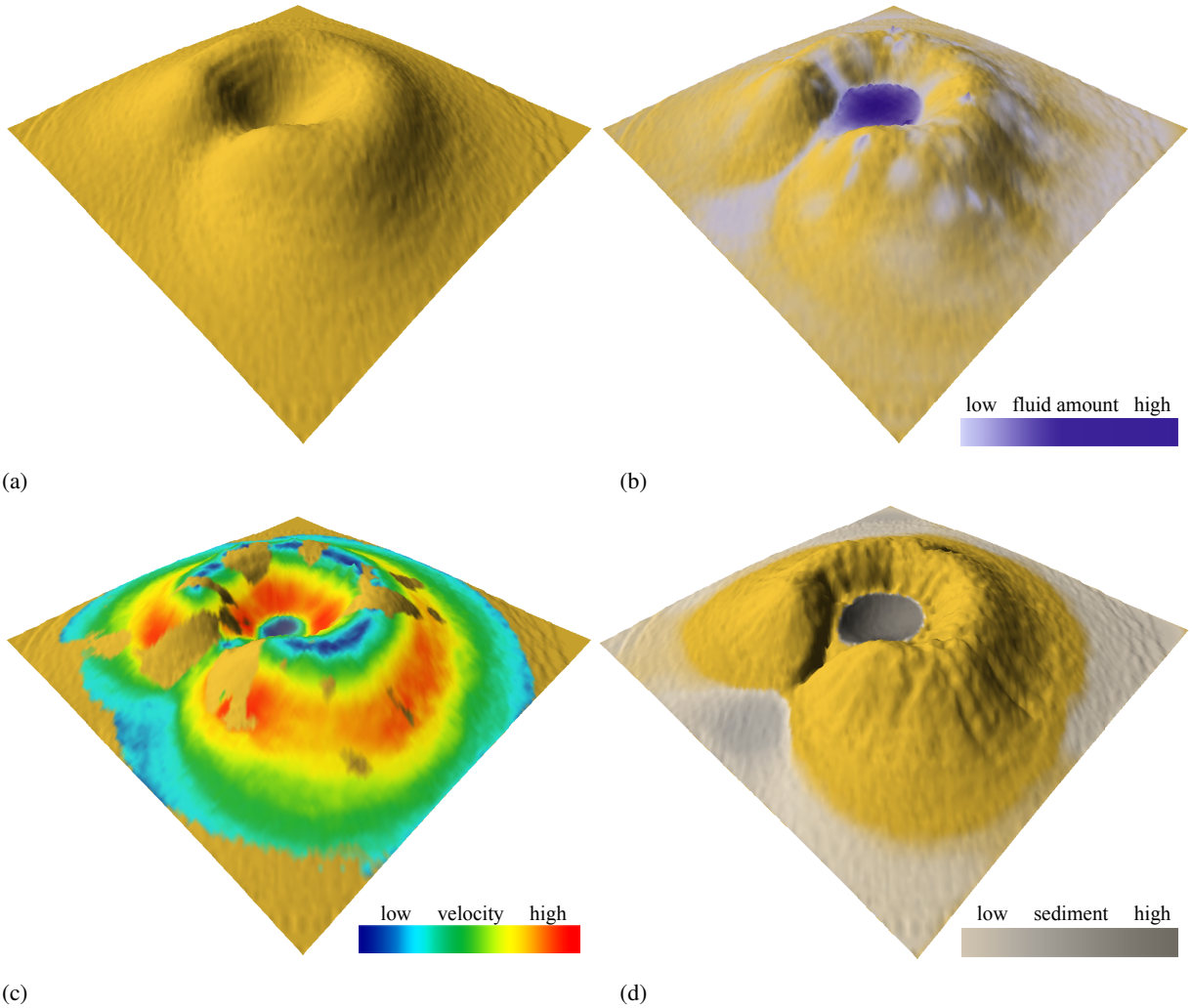

Figure 3: *Hydraulic Erosion: a) original model; b) fluid simulation; c) color-coded velocity amount of the fluid; d) eroded terrain after 300 simulation steps with color-coded amount of depositing.*

tor is moved from one grid cell to another by the fluid simulation. As inputs for the function, there are the amount of transported fluid ΔF and the amount of sediment dissolved in this fluid ΔS. Without erosion these inputs would be simply added to the underlying terrain H respectively the actual sediment amount S at the destination cell. If erosion takes place, the transfer function is modified such that some sediment can be deposited to or dissolved from the underlying terrain at the destination cell.

Our erosion algorithm uses some material specific constants which are influencing the whole process. We introduce the *sediment capacity constant* K_C which specifies how much sediment of the underlying material can be dissolved in one unit of water at a velocity of one. The *deposition constant* $K_D \in [0..1]$ controls the rate at which soil is deposited at the target grid cell. The counterpart, the *dissolving constant* $K_S \in [0..1]$ controls the dissolving rate of the underlying terrain into the fluid per simulation step (see table 2). Please note that the comparatively big default value of $K_C = 15$ can be used to match the different magnitudes of time steps for fluid simulation (normally measured in seconds) and erosion simulation (measured in years).

sediment capacity constant	K_C	15
deposition constant	K_D	0.5
dissolving constant	K_S	0.3

Table 2: *Material constants and their default values.*

To decide whether to deposit or to dissolve some material in the actual step we calculate the *sediment capacity* c_s of the fluid that specifies the maximum amount of sediment that can be transported by the fluid ΔF. Because of the fact that our fluid simulation also provides the actual velocity $\vec{v}$, our model enables us to compute c_s in a more physically accurate way compared to prior presented models. The factor Δt is used to scale the equation correctly at time steps unequal to one. The whole rule for sediment capacity then reads

$$c_s = \frac{K_C}{\Delta t} \cdot \Delta F \cdot |\vec{v}| \tag{19}$$

In the simulation step for depositing or dissolving, we compare the sediment capacity c_s to the actual amount of dissolved sediment ΔS. As a decision formula, we implemented the following.

If $\Delta S > c_s$ deposit:

$$H = H + \frac{K_D}{\Delta t} \cdot (\Delta S - c_s) \tag{20}$$

$$S = S + \Delta S - \frac{K_D}{\Delta t} \cdot (\Delta S - c_s) \tag{21}$$

If $\Delta S \leq c_s$ dissolve:

$$H = H + \frac{K_S}{\Delta t} \cdot (\Delta S - c_s) \tag{22}$$

$$S = S + \Delta S - \frac{K_S}{\Delta t} \cdot (\Delta S - c_s) \tag{23}$$

In both cases we modify the transfer function for the height H of the terrain ((20) resp. (22)) and the sediment S dissolved in the fluid with an *erosion term* ((21) resp. (23)). This term is additionally scaled with one of the two material constants K_D, resp. K_S that are specific for erosion. They are used since depositing and dissolving are continuous processes and hence the erosion process affects a specified portion of the difference $\Delta S - c_s$.

5. Water distribution

The most important influence factor for a hydraulic erosion system is the distribution of water. Two major types of *fluid-generators* exist: rainfall and water sources. Rainfall in nature is strongly influenced by a complex system of wind and air humidity, called *adiabatics*. To simplify this system, we simulate rainfall by dropping some amount of rain to a grid-cell $R(x,y)$ at constant or random intervals (approximately 2 to 50 time steps). Eq. (24) emulates the precipitation on the slope of a mountain. The constant K_R scales the actual rain amount. We achieved good results with $K_R = 0.001$.

$$R(x,y) = \begin{cases} \frac{K_R}{\Delta t} \cdot H(x,y)^2, & H(x,y) > 0 \\ 0, & H(x,y) \leq 0 \end{cases} \tag{24}$$

Besides the rain frequency and the rain amount, the distribution of rain over the landscape realized with a noise pattern also has a significant effect on the erosion result. Figure 4 shows a blue noise distribution of rain where the positions are obtained from a Hammersley point set [WLH97]. For performance reasons we pre-calculate a very dense point set and control the runtime density of the rain points by subsampling the point set.

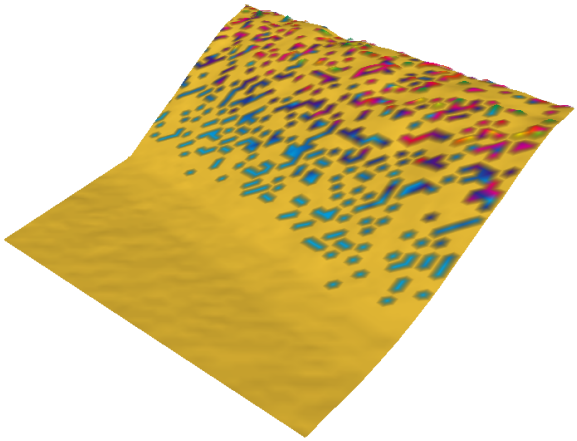

Figure 4: *Blue noise distribution of rainfall with color-coded distribution values.*

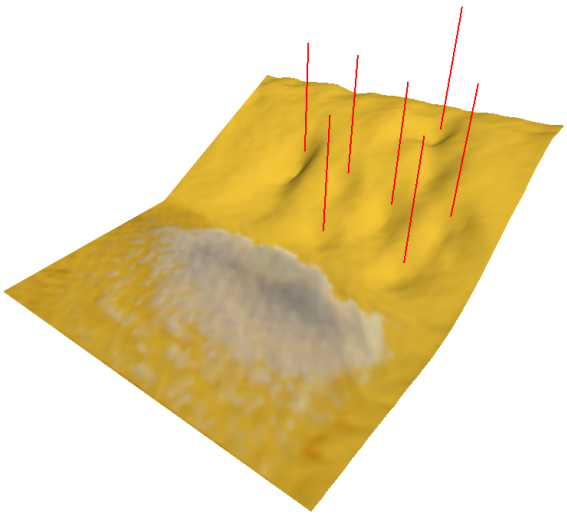

Figure 5: *Erosion with seven fixed river sources with color-coded amount of deposited sediment.*

The rain distribution is influenced globally by some parameters. To improve the interactive terrain creation process we additionally introduce the possibility to define water sources in the simulation environment (see figure 5). At every water source, some water is dropped with a Gaussian distribution in regular intervals. The radius of the Gaussian

distribution can be controlled by the user. Using this method it is possible to generate erosion patterns and valleys where the user needs them. Additionally, the user can *paint* water interactively into the simulation.

In the real world water disappears in many ways from the surface. In our simulation, water can disappear in two ways: either through reaching the border of the simulation area or during an evaporation process. In nature, the evaporation amount depends on the temperature and the amount of water. We use an evaporation function proposed by [BF02] (see Eq. (25)) which decreases the amount of water exponentially over the time. Evaporation can be controlled with the *evaporation constant* K_E. For performance reasons, we define a lower bound ε. When the fluid amount reaches this value it is set to zero.

The simulation can further be speed up if grid cells without fluid are not touched by the whole fluid and erosion simulation. Please note that we assume the fluid temperature to be constant over the time in the evaporation process. For a more correct physical evaporation we can assume low temperatures on high terrain levels and high temperatures on low terrain levels that cause a varying evaporation. As an approximation we propose the additional terrain level factor $H(x,y)$ in the exponent of the evaporation function $(-K_E \cdot \Delta t \cdot H(x,y))$ to simulate such a temperature based evaporation.

$$F(x,y) = \begin{cases} 0, & F(x,y) < \varepsilon \\ F(x,y) \cdot e^{-K_E \cdot \Delta t}, & otherwise \end{cases} \qquad (25)$$

6. Conclusions and Future Work

In this work we demonstrated two new methods concerning erosion simulation. First, a new algorithm for water transportation was introduced. It is able to simulate all features of fluid dynamics that are important for erosion in a more accurate (physically correct) way than prior algorithms without loosing the capability to calculate the results interactively.

Second, interactive methods to influence the terrain shape are introduced that allows the user to control all *global* simulation parameters of several independent simulation steps such as rain fall, water sources, fluid transportation, material dissolving, material depositing and evaporation. Additionally, the user is able to define *local* water sources. The proposed erosion algorithms are mass conserving, so that no material is lost at any simulation step.

All results (see figure 6) where implemented in C# and tested on a 2.4 GHz Pentium IV PC. The Simulation frame rate depends above all on the grid size of the terrain. With 256x256 grid cells the not yet optimized algorithm runs at approximately 4 frames per second.

A weakness of the erosion systems is the disability to simulate erosion in caves and erosion fragments caused by vertical vortices. In future work we plan to enhance our algorithms to handle such phenomena. Also the research in the field of texture generation, object placement (e.g. stones) and object growing (e.g. plants) in the simulation environment would be interesting. Both can be generated in a physically based way with the results of the fluid simulation and the erosion algorithm.

References

[BF01] BENES B., FORSBACH R.: Layered data representation for visual simulation of terrain erosion. In *SCCG '01: Proceedings of the 17th Spring conference on Computer graphics* (2001), IEEE Computer Society, p. 80.

[BF02] BENES B., FORSBACH R.: Visual simulation of hydraulic erosion. In *Journal of WSCG 2002* (2002), vol. 10.

[BMS97] BENES B., MARAK I., SIMEK P.: Hierarchical erosion of synthetical terrains. *In Spring Conference on Computer Graphics, Bratislava: Comenius University* (Jan 1997), 93–100.

[CDM*02] CUTLER B., DORSEY J., MCMILLAN L., MÜLLER M., JAGNOW R.: A procedural approach to authoring solid models. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (2002), ACM Press, pp. 302–311.

[CMF98] CHIBA N., MURAOKA K., FUJITA K.: An erosion model based on velocity fields for the visual simulation of mountain scenery. *Journal of Visualization and Computer Animation 9*, 4 (1998), 185–194.

[CMT04] CARLSON M., MUCHA P. J., TURK G.: Rigid fluid: animating the interplay between rigid bodies and fluid. *ACM Trans. Graph. 23*, 3 (2004), 377–384.

[FM97] FOSTER N., METAXAS D.: Controlling fluid animation. In *CGI '97: Proceedings of the 1997 Conference on Computer Graphics International* (1997), IEEE Computer Society, p. 178.

[HW04] HOLMBERG N., WUENSCHE B. C.: Efficient modeling and rendering of turbulent water over natural terrain. In *GRAPHITE '04: Proceedings of the 2nd international conference on Computer graphics and interactive techniques in Australasia and Southe East Asia* (2004), ACM Press, pp. 15–22.

[KM90] KASS M., MILLER G.: Rapid, stable fluid dynamics for computer graphics. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques* (1990), ACM Press, pp. 49–57.

[KMN88] KELLEY A. D., MALIN M. C., NIELSON G. M.: Terrain simulation using a model of stream erosion. In *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques* (1988), ACM Press, pp. 263–268.

[MCG03] MUELLER M., CHARYPAR D., GROSS M.:

Particle-based fluid simulation for interactive applications. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer animation* (2003), Eurographics Association, pp. 154–159.

[MH82] MANDELBROT B. B., HUDSON R. L.: *The Fractal Geometry of Nature*. W. H. Freeman and Company, New York, 1982.

[MKM89] MUSGRAVE F. K., KOLB C. E., MACE R. S.: The synthesis and rendering of eroded fractal terrains. In *SIGGRAPH '89: Proceedings of the 16th annual conference on Computer graphics and interactive techniques* (1989), ACM Press, pp. 41–50.

[Nag98] NAGASHIMA K.: Computer generation of eroded valley and mountain terrains. *The Visual Computer 13*, 9-10 (Jan 1998), 456–464.

[OH95] O'BRIEN J. F., HODGINS J. K.: Dynamic simulation of splashing fluids. In *CA '95: Proceedings of the Computer Animation* (1995), IEEE Computer Society, p. 198.

[PH93] PRUSINKIEWICZ P., HAMMEL M.: A fractal model of mountains with rivers. *Proceeding of Graphics Interface* (May 1993), 174–180.

[RPP93] ROUDIER P., PEROCHE B., PERRIN M.: Landscapes synthesis achieved through erosion and deposition process simulation. *Computer Graphics Forum 12*, 3 (1993), 375–383.

[Sta99] STAM J.: Stable fluids. *SIGGRAPH '99: Conference Proceedings, Annual Conference Series* (August 1999).

[Sta03] STAM J.: Real-time fluid dynamics for games. *Proceedings of the Game Developer Conference* (March 2003).

[WLH97] WONG T., LUK W., HENG P.: Sampling with hammersley and halton points. *Journal of Graphics Tools 2*, 2 (1997), 9–24.

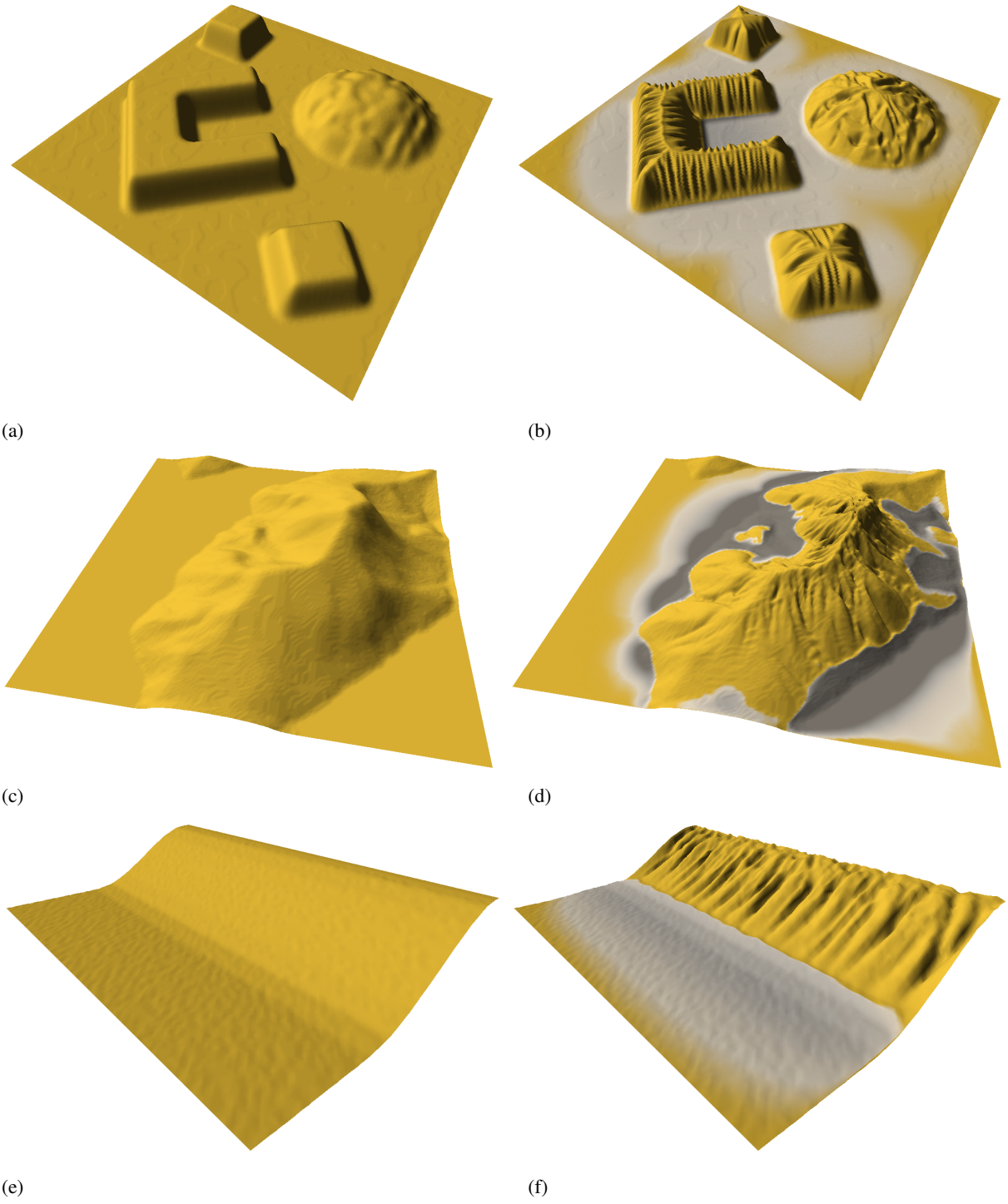

Figure 6: *Hydraulic Erosion: a) original artificial terrain; b) 300 steps eroded artificial terrain; c) original mountain chain; d) 300 steps eroded mountain chain; e) synthetic inclined plane; f) 300 steps eroded inclined plane. The amount of deposited sediment is color-coded in the eroded images.*