Rain Scene Animation Through Particle Systems and Surface Flow Simulation by SPH

Bruno Barcellos Coutinho Antonio A. F. Oliveira
Yalmar Ponce Atencio
Computer Graphics Laboratory - COPPE - UFRJ
Rio de Janeiro, Brazil
(barcellos, oliveira, yalmar)@cos.ufrj.br

Gilson Antonio Giraldi
National Laboratory for Scientific Computing
Rio de Janeiro, Brazil
gilson@lncc.br

***Abstract*—Realistic rain scenes animation is a complex task due to both the rendering and fluid dynamics simulation issues. The associated phenomena, like rain strokes rendering, splashing of raindrops and the simulation of the generated surface flow are known difficulties in this area. This paper describes a computational framework to incorporate these elements in a scene containing a digital terrain model (DTM). In the rain model, the raindrops are modeled by a particle system implemented in GPU. Each particle represents a drop and the Precomputed Radiance Transfer (PRT), expressed in a spherical harmonics bases, is used to incorporate environment lightning to the rendering engine. The Smoothed Particle Hydrodynamics (SPH) method is employed for simulating the superficial flow over the terrain and lake formations. The rendering of the fluid free surface is performed by applying an environment mapping technique plus Fresnel effects to a regular geometric representation known as "carpet". The experimental results show the potential of the proposed pipeline for computer graphics and highlight the fact that it is a promising framework for real time applications.**

***Index Terms*—rain animation; SPH; particle system; PRT; terrain model;**

I. Introduction

Physically-based techniques for the animation of natural elements like fluids (gas or liquids), elastic, plastic and melting objects, among others, have taken the attention of the computer graphics community in the last decades [1], [2]. In particular, a fair amount of work has been done on realistic animation of rain and associated phenomena [3]. The motivation for such interest rely in the potential applications of these methods in film post-production, games, simulators and virtual reality [4].

Raining effects have been rendered in four ways: using image-based, textures-based, particle system or hybrid approaches [3]. In image-based methods the rain is extracted from a video and then applied to different background. The key idea of these approaches is to get, from the source video, the complex brightness patterns observed in rain streaks and then to apply these patterns to the target one. These patterns are produced by elements like lighting, viewpoint effects, reflection and refraction which are hard to properly control in a simulation. In this area we shall mention the methods that add or remove rain patterns from videos [5]–[7] and the image-based method presented in [4] which includes the development and utilization of a streak database.

The texture-based techniques use a texture which is continuously scrolled following the direction of the falling rain [8]. The animation of rain with virtual particles applies the combination of a particle system with a physically-based motion model of the rain and light interaction techniques [6], [9], [10]. Methods in this category includes collision detection methods for raindrops [9], GPU implementations for rendering rain streaks [11], depth of field (DOF) effects to take into account the distance from the viewer [9], as well as level-of-details (LOD) approaches [12]. The last category is composed by hybrid approaches combining particle systems and scrolling textures [10], image-based methods to transfers details from video of real rain to virtual particles and rendering techniques [6].

In this paper we propose an hybrid framework for realistic rain scenes animation which is composed by a particle-based rainfall model, surface flow simulation based on Smoothed Particle Hydrodynamics (SPH), and a method (carpet) for extracting the fluid free surface. The Precomputed Radiance Transfer (PRT) is used to incorporate environment lightning to the rain strokes rendering. The rendering of the free surface of the fluid is performed through environment mapping technique plus Fresnel effects.

Our framework is inspired in the hybrid technique described in [6] which final goal is to add a rainfall in a target video sequence. However, unlikely that reference, our aim is to render a whole virtual scene. Besides, the reference [6] did not simulate surface flows.

The PRT is a global illumination method used as an alternative to the expensive ray tracing based techniques [13]. Basically, the incoming light, as well as the transfer function of objects, are expressed in a spherical harmonics base. That means, the light integration is reduced to a simple dot product between the vector of light coefficients and the vector of the transfer function coefficients. Through PRT, we can compute the appearance of the rain, based on the environment map, in real-time [14].

The SPH is a mesh free, lagrangian, method used for the numerical simulation of mechanical systems [15]. Mesh free methods have been a hot topic of research in the computer graphics community mainly due to their efficiency and sim-

plicity to deal with boundary changes [16], [17]. A common characteristic of SPH models is the need of a method to simulate and extract the free surface of the fluid [18]. In this paper, the simulation is addressed by the method proposed in [18]. The free surface extraction is implemented by building a representation of the surface based on a regular structure (carpet) following [19].

The main contribution of this work is the development of a particle based framework to simulate a rainfall scene as well as associated effects, like the superficial flow and lake formation. The paper is organized as follows. Section II gives background for paper understanding. The proposed integrated model is presented in section III. The experimental results are shown on section IV. Finally, we present the conclusions and future works on Section V.

II. Background

The proposed methodology relies on the Navier-Stokes equations plus SPH for fluid simulation, a regular representation for the free surface and a particle-based framework for realistic rain scenes animation. In this section we review these approaches.

A. Navier-Stokes for Fluid Animation

The majority fluid models in computer graphics follow the Eulerian formulation of fluid mechanics that is based on a top down viewpoint of the nature: the fluid is considered as a continuous system subjected to Newton's and conservation Laws as well as state equations connecting the macroscopic variables of pressure P, density ρ, and velocity $\vec{v}$. In this theory, the mass conservation, also called continuity equation, is given by [20]:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{v}) = 0 \tag{1}$$

The linear momentum conservation equation, also called Navier-Stokes, can be obtained by applying the third Newton's Law to a volume element dV of fluid. For incompressible flows ($\nabla \cdot \vec{v} = 0$) it can be written as [20]:

$$\rho \frac{d\vec{v}}{dt} = -\nabla P + \mathbf{F} + \mu \nabla^2 \vec{v}, \tag{2}$$

where $\mathbf{F}$ is an external force field and μ is the viscosity of the fluid. Also, we may need an additional equation for the pressure field. This is a state equation which ties together all of the conservation equations for continuum fluid dynamics and must be chosen to model the appropriate fluid (*i.e.* compressible or incompressible). In the case of liquids, the pressure P is temperature insensitive and can be approximated by $P = P(\rho)$. In [18], it is applied the expression: $P = c^2\rho$, where c is the speed of sound in this fluid. Equation (2) needs initial conditions $\vec{u}(t = 0, x, y, z)$ as well as boundary conditions, like the usual no-slip one: $\vec{u}|_S = 0$.

B. Smooth Particle Hydrodynamics

The two fundamental elements in the Smoothed Particle Hydrodynamics (SPH) method are an interpolation kernel W and a particle system that represents a discrete version (sample) of the fluid [15]. The kernel estimation of a scalar quantity $A(\mathbf{r})$ is given by:

$$\langle A(\mathbf{r}) \rangle = \int_{Space} A(\mathbf{r}') W(\mathbf{r} - \mathbf{r}', h)\, d\mathbf{r}', \tag{3}$$

where the function $W(\mathbf{r} - \mathbf{r}', h)$ is an interpolation kernel which must satisfies properties like Volume conservation and Dirac delta function limit [15]. So, if we take a sampling of A then the $A(\mathbf{r}')$ in equation (3) will be known only at a discrete set of N points $\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_N$. Hence, through the mentioned properties it is possible to show that the scalar field A and its gradient $\langle \nabla_{\mathbf{r}} A(\mathbf{r}) \rangle$ can be estimated by [15]:

$$\langle A(\mathbf{r}) \rangle = \sum_{j=1}^{N} \frac{m_j}{\rho(\mathbf{r}_j)} A(\mathbf{r}_j) W(\mathbf{r} - \mathbf{r}_j, h). \tag{4}$$

$$\langle \nabla_{\mathbf{r}} A(\mathbf{r}) \rangle = \sum_{j=1}^{N} \frac{m_j}{\rho(\mathbf{r}_j)} A(\mathbf{r}_j)\ \nabla_{\mathbf{r}} W(\mathbf{r} - \mathbf{r}_j, h). \tag{5}$$

where the smoothing length h is the width of the kernel and defines the distance at which a particle interacts with other particles. An analogous expression can be obtained by the Laplacian. In this work, the smoothing kernels are usual ones applied in the literature and described in [18].

From equation (5) we can observe that there is no need for a mesh to compute spatial derivatives. With equations (4) and (5), we can rewrite the terms of the Navier-Stokes equation (2), using this approach, as (see [15] for details):

$$\vec{f}_i^{press} = -\sum_j m_j \frac{p_i + p_j}{2\rho_j} \vec{\nabla} W(\mathbf{r}_i - \mathbf{r}_j, h) \tag{6}$$

$$\vec{f}_i^{visc} = \mu \sum_j m_j \frac{\vec{v_j} - \vec{v_i}}{\rho_j} \vec{\nabla}^2 W(\mathbf{r}_i - \mathbf{r}_j, h) \tag{7}$$

$$\vec{f}_j^{grav} = \rho_j \vec{g_j} \tag{8}$$

where the $\vec{f}_i^{press}$ and $\vec{f}_i^{visc}$ are the pressure and viscosity forces. Only the gravity force $\vec{f}_i^{grav}$ is considered as external force. The density at each particle can be found from the following equation:

$$\rho_i = \sum_{j=1}^{N} \rho_j m_j W(\mathbf{r}_i - \mathbf{r}_j, h). \tag{9}$$

Besides the dynamic forces in expression (6)-(8) we also need a surface tension to be applied in the particles nearby the free surface. Such particles undergo a density gradient, which is an artifact of the SPH framework, pictured on Figure 1.

In [18] this problem is addressed through an additional field, called the color field c_S, which is unity in SPH particles and

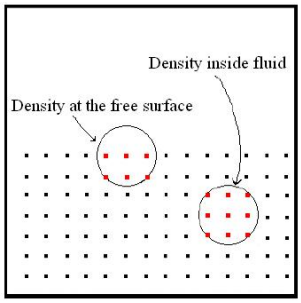


Fig. 1. Density gradient nearby the free surface.

zero otherwise, whose smoothed version $\langle c_S(x)\rangle$ is computed by expression (4). The free surface tension is given by:

$$t^{slivre} = -\sigma\kappa\frac{\vec{n}}{||\vec{n}||}, \tag{10}$$

where σ is a scale factor and $\vec{n}$, κ are the surface normal and curvature, at the free surface particles. They are computed, respectively, by:

$$\vec{n} = \vec{\nabla}\langle c_S(x)\rangle; \quad \kappa = -\frac{\Delta\langle c_S(x)\rangle}{||\vec{n}||}. \tag{11}$$

In the SPH approach, the numerical scheme for fluid simulation is obtained by substituting expressions (6)-(8) into the Navier-Stokes equation, which gives the expression:

$$\frac{d\vec{v}_i}{dt} = \frac{Q_i^t}{\rho_i}, \tag{12}$$

where $Q_i^t = \vec{f}_i^{press} + \vec{f}_i^{visc} + \vec{f}_j^{grav}$.

The Equation (12) computes the particles acceleration. So, the Leapfrog technique [15] allows to update the particles positions as follows:

$$\vec{v}_i^{t+\delta t} = \vec{v}_i^t + \frac{\delta t}{2}Q_i^t, \tag{13}$$

$$r_i^{t+\delta t} = r_i^t + (\delta t)\cdot\vec{v}_i^{t+\delta t}, \tag{14}$$

where δt is a time step.

C. Free Surface Computation

In this paper we apply the carpet structure, described in [19] for free surface representation. The carpet consists of two data structures: a $n \times n$ (virtual) grid and a quadtree. The key idea is to store SPH particles in the quadtree and to traverse this structure to efficiently extract the free surface geometry which will be stored in the grid nodes. Each node (x, y, z) of the grid stores in the z coordinate the local height of the free surface. At the initialization, the grid is flat and located below any terrain height value. Then, a full quadtree is instantiated on the grid quads with each node storing the absolute minimum terrain height in this domain, the current absolute grid height and a velocity value. Then, the SPH particles are inserted into the tree leaves by storing their absolute height in the corresponding spatial bin. The carpet is pushed to the highest level particles high. Next, the inner nodes of the quadtree are updated by propagating the height values from bottom to top.

We can now render the free surface efficiently by traversing the quadtree. The recursion is aborted and nothing is done if a height value below the height of the terrain is found. If we reach a leaf node, we can be sure that it must be rendered. Afterwards, the carpet velocity is accelerated by gravity in order to smoothly update carpet position. At the end of this process, only the visible parts of the free surface are stored in the grid nodes. In this paper, the rendering of the free surface is performed through traditional environment mapping plus Fresnel effects [21].

D. Real-Time Rain Animation with PRT

Following [6], we also use a particle system for the rendering of rain. Using such a system realistic rain simulations controlled by some few parameters can be produced. Figure 1 shows an outline of the whole process till the final rendering is obtained.

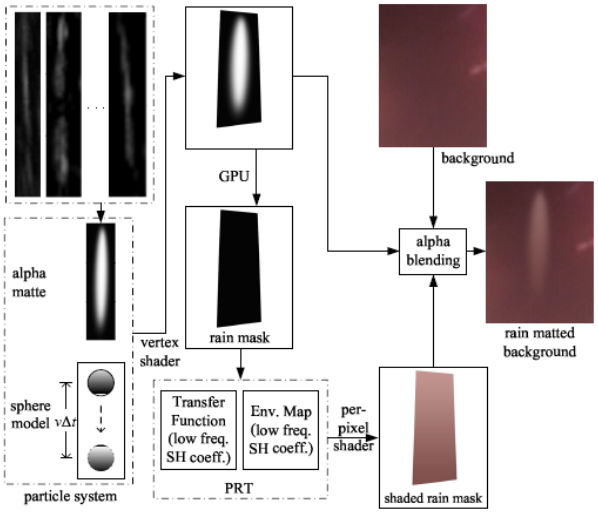


Fig. 2. Rendering rain pipeline (Reprinted from [6]).

Each rain drop is represented by a particle consisting of a spherical model. Besides simple features, like velocity and refraction index, it has an associated oppacity map refereed here as the drop alpha matte (see Figure 2), left-middle. The alpha matte is defined on a rectangular strip whose width is related to the sphere diameter and the length to the product of the particle velocity by the exposure time of the simulation virtual camera (Figure 2, left-bottom).

To obtain the image of a drop, we start by making the alpha matte and particle centers coincident. Then, the alpha matte is oriented so that two sides become parallel to the velocity and the other two orthogonal to the viewing direction. Positioned in that way, it is, then, projected onto the image plane. See Figure 2, top middle. Every pixel on that projection (the alpha-mask) will take part in the "drop image". To compute the color/brightness attributes of those pixels two components are considered. One comes from an environment-map (E-M)

while the other from the background image. That environment map can be generated based on an image or video of a real scene like in [6] and serves for different purposes but, in the case, mainly to make the environment lightness influence the results. The E-M component is modified, due to the light-drop interaction in the following way. Each one of the alpha-mask pixels defines a viewing direction (VD) - that of its projecting ray. The spherical form of the drop model allows that the radiance transfer function (RTF) for any outgoing direction has a closed form expression. We remark that this makes its evaluation much faster since there is no need of tracing the light inside the drop. In spite of that, evaluating the flux of light leaving the drop in a certain direction, considering the contributions of all incoming directions can be computationally heavy for real time applications. To address this problem, both the E-M and the RTF for VD are replaced by compact Spherical Harmonics (SH) based representations of them - we have used only the 16 lowest frequency SH functions. The scalar product of these representations gives the E-M contribution to the pixel value(s). We observe that these representations are both obtained in a pre-processing phase constituting the so called Pre-Computed Radiance Transfer (PRT) engine which makes it possible, through a GPU based computation process, to obtain the E-M contribution to the drop image in real-time (Figure 2, middle-bottom).

In a last step the background contribution is blended with the E-M one to produce the final rain drop image. Blending weights of each component are obtained from the projected alpha-matte (see alpha blending block in Figure 2).

We follow Wang et al.'s approach [6] to obtain a closed form for a spherical drop RTF. Here, however, we treat the problem in a discretized context and indicate a more robust computation process.The discretization approach used resumes to subdividing the range $[0, \pi]$ into intervals (J_m) of length Δ and take the middle points of them (α_m).

The reversibility of light implies that the RTF for VD must be identical to the distribution function of the outgoing light intensity if the incoming direction is VD. So, let us focus on that last function. Now, let I_{in} be the intensity of the incident light and L_n and I_n be, respectively, the direction and intensity of the outgoing light after being either reflected without entering the drop ($n = 1$) or reflected inside the raindrop $(n-2)$ times ($n \geq 2$).

Incident rays parallel to VD can hit the raindrop at different positions, resulting in different incident angles γ. On the other hand, as the whole light path inside the drop must lie in the same plane containing its center, the azimuth angle ϕ is always the same. So, to express the outgoing direction L_n, we must only be concerned on obtaining the polar angle θ_n. Then, consider the triangle wave $TW_n : x \rightarrow |2.\arcsin(\sin(\frac{x+mod(n,2).\pi}{2}))|$. Since the drop is spherical, it can be proved that: $\theta_n = \Theta_n(\gamma) \triangleq TW_n(2((n-1)\beta - \gamma))$, where β, is the angle between the refracted light and the surface normal at the entrance point.

Let R be the reflectance of the drop and $T = 1 - R$, its transmittance. If $n = 1$, we have a straightforward formulation: $\frac{I_1(\theta)}{I_{in}} = 0.25R(\frac{\theta}{2})$. For $n \geq 2$, define $\mu(\gamma) = T(\gamma)^2 R(\gamma)^{n-2} \sin(\gamma)\cos(\gamma)$. For a small Δ, a very reasonable approximation of $\frac{I_n(\alpha_m)}{I_{in}}$ is given by:

$$\zeta_m^n \triangleq \frac{\sum_k \mu(\alpha_k).|(J_k \cap \Theta_n^{-1}(J_m))|}{\Delta \sin(\alpha_m)} \tag{15}$$

where $|J|$ is the length of the 1D set J. In consequence,

$$RTF(\alpha_m, \phi) = \frac{I_{out}(\alpha_m)}{I_{in}} = \sum_{n=1}^{\infty} \frac{I_n(\alpha_m)}{I_{in}} \cong \sum_{n=1}^{n_{max}} \zeta_m^n \tag{16}$$

According to [6] with $n_{max} = 9$, the percentual error resulting of cutting off the contributions from the terms with a higher n is reduced to 0.2%. During the simulation process we obtain $RTF(\theta, \phi)$ by simply interpolating the values pre-computed for the two α_m closest to θ.

When applying PRT, both the discretized outgoing light intensity distribution and the environment map are projected onto a set of Spherical Harmonics subspace. The SH functions define an orthonormal basis for the set of scalar functions defined on the unitary sphere S. They are defined by:

$$y_l^m(\theta, \phi) = \sqrt{2} K_l^m \cos(m\varphi) P_l^m(\cos\theta), m > 0, \tag{17}$$

$$y_l^m(\theta, \phi) = \sqrt{2} K_l^m \sin(-m\varphi) P_l^{-m}(\cos\theta), m < 0, \tag{18}$$

$$y_l^0(\theta, \phi) = K_l^0 P_l^0(\cos\theta), m = 0. \tag{19}$$

where P_l^m are the associated Legendre polynomials and K_l^m is a normalization coefficient.

Because the SH basis is orthonormal, the coordinate (f_l^m) of a scalar function f in relation to the harmonic y_l^m is given by the integral over S of the product $f.y_l^m$.

With these coordinates, we can compute the n^{th} order reconstruction function to approximate f:

$$\widetilde{f}(\theta, \varphi) = \sum_{l=0}^{n-1} \sum_{m=-l}^{l} f_l^m y_l^m(\theta, \varphi). \tag{20}$$

For Computer Graphics purposes it is well accepted that making $n = 4$, which means having 16 SH coefficients, is a good solution for the trade-off between computational cost and accuracy. Thus the vectors in our PRT engine have that dimension.

III. The Proposed Integrated Framework

As it has already been indicated, the main purpose of this work is to integrate in a single framework the following elements: (a) A particle-based rainfall model implemented in CUDA; (b) a SPH based surface flow simulation model; and (c) A method (carpet) for extracting the fluid free surface. This integrated scheme is applied to a wholly synthesized scene containing a digital terrain model (DTM) represented by a regular mesh that keeps the height for each point.

The rain simulation method takes alpha mattes from a database which is generated from rain videos through the technique described in [4]. Then, the raindrops are modeled by a particle system implemented in GPU. Each particle represents a drop and it is rendered using the alpha matte

and the Precomputed Radiance Transfer (PRT) to incorporate environment lightning, following the technique described in section II-D. The surface flow simulation follows the Navier-Stokes equation (2), and its discretization is given by expressions (12)-(14). It is important to emphasize that our SPH model has non-traditional elements (particles generation due to precipitation and evolution on a curved surface) that requires specific considerations for a GPU implementation. That is way we decided for a CPU implementation of the surface flow in this work.

The carpet structure (section II-C) stores the free surface geometry and its physical integrity is kept by the surface tension computed by expression (10).

After setting the initial conditions, the process of simulation starts. Figure 3 shows the interaction between the models at each time step.

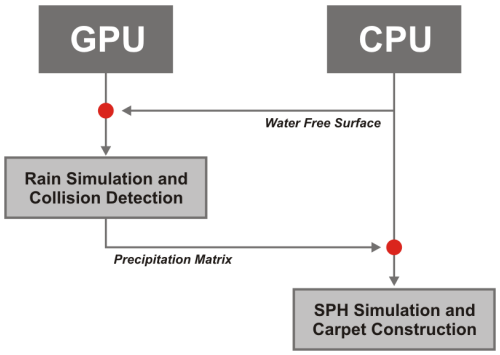


Fig. 3. Overview of the proposed framework.

Basically, the rainfall has to evolve until it reaches the fluid free surface (or the DTM surface), and then the rain particles die and feed the SPH model. The collisions between the rain particles and the fluid surface (carpet) are computed in GPU. For this computation, at each time step the CPU has to send to GPU a matrix with the heights of the fluid free surface.

With this data, the system can check all the new positions of each rain particle: if the rain particle has reached the free surface or the DTM it dies and updates a precipitation matrix for water flow simulation. That precipitation matrix is a particle counter field $counter(i, j, t)$, which mimics the water held over the DTM. For a high resolution carpet with height field $\varphi(i, j, t)$ we can perform the collision detection test for a rain particle at position (x, y, z, t) simply by verifying if $z < \varphi(\lfloor x \rfloor, \lfloor y \rfloor, t)$. If yes, the precipitation matrix is updated as:

$$counter(\lfloor x \rfloor, \lfloor y \rfloor, t) = counter(\lfloor x \rfloor, \lfloor y \rfloor, t) + 1. \quad (21)$$

After the computation of the collisions and update of $counter(i, j, t)$, the GPU has to send back to the flow simulation model, in CPU, a matrix with the precipitation information. With this data, the method inserts SPH particles in the correspondent collision position, according to the expression:

$$N_{new}(i, j, t) = \left\lfloor \frac{counter(i, j, t)}{K} \right\rfloor, \quad (22)$$

where $N_{new}(i, j, t)$ is the number of new SPH particles inserted at time t, in the terrain point $(i, j, z(i, j))$, and K is a precipitation scale factor.

Then, following the dataflow of Figure 3, the SPH simulation starts. SPH particles positions are updated through expressions (12)-(14). If a particle is closer the DTM surface, it may happen that the new position obtained by expression (14) is bellow the DTM surface. In this case, we correct the particle position through a reflection respect to the local DTM tangent plane. Besides, we reflect also the particle velocity and re-scale its intensity using a damping coefficient.

Then, the new carpet height field is computed with the maximum SPH height in each carpet cell. The precipitation matrix is also used to render the splashes of the drops in the fluid surface. This process is based on the work [22], where the splash evolution is represented with a sequence of six textures pictured on Figure 4.

Fig. 4. Pictures sequence for drop splashes rendering.

These textures are rendered with the point sprite technique and rotated according to the angle between z-axis and the free surface normal at the collision point. In order to apply different series of textures for different kind of sprays, sprays of raindrop are classified into three categories, following the different range of angle between the z-axis and the collision point normal.

Rain particles born at the top of the rainfall area. Their velocities are randomly set and the equation of motion is simply the uniform one:

$$r_i^{t+\Delta t} = r_i^t + (\Delta t) \cdot \vec{v}_i, . \quad (23)$$

For the vertical direction we add the gravity term $(-4.9t^2)$. If a rain particle dies at time t then a new rain particle enters the system at the next simulation time in the same position where the died particle had born. In the actual implementation, we subdivide the top of the rainfall area into a egular mesh and use the grid nodes as birth sites for rain particles. The GPU is the host for all these rain particles processing.

In Figure 3 it is supposed that the time scale of the rain and SPH simulation are equal. However, we can change this scheme; for instance, we can update the SPH system only after a number $m > 1$ of rain simulation steps. Such choice improves performance and it may not reduce the quality of surface flow visualization.

IV. Results

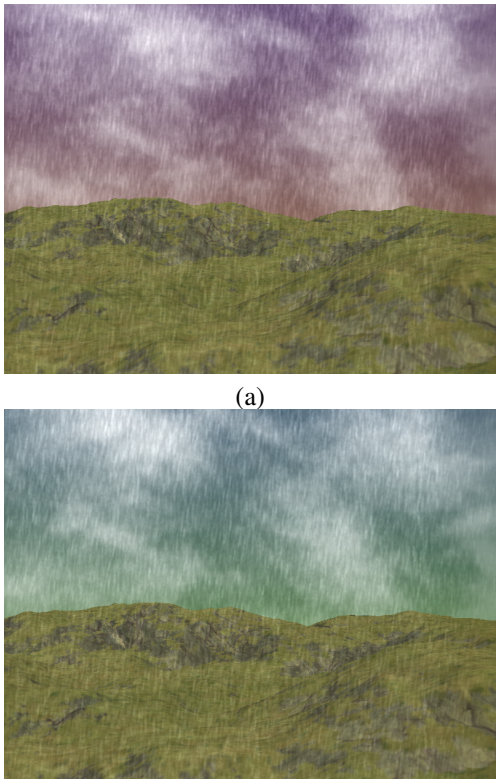
In this section we describe some experiments with the proposed framework. We show the behavior of the fluid in different digital terrain models and highlight aspects that can be useful for computer graphics applications. The framework

was developed in C/C ++ language, and the rain animation in CUDA. The visualization was implemented in OpenGL and the shaders in OpenGL Shading Language. The experiments were performed in a Intel Quad Core 3.0 GHz, with 4 GB of RAM and a Video Card NVidia GeForce 9800 GTX, running in a OpenSuse Linux platform. The pictures included in this paper are snapshots obtained from the framework.

A. Rendering Results

In the presented experiments, the carpet resolution is always 2048×2048, and the DTMs are represented by digital elevation models, with resolution 512×512. The update of SPH particles and the rain drops positions follow expressions (13)-(14) and (23) and respectively, with $\delta t = 0.15$ and $\Delta t = 0.025$.

1) Rain Rendering: The Figure 5.(a) shows the rain rendering results obtained using the method described on section II-D. We can observe realistic effects due to the environment lightning and rendering engine on the synthetic rain. This example shows the effect of changing the environment map color. Even though the same rainfall setup and scene parameters are used in both the pictures, the final rain appearance looks significantly different due to the change of the environment map color.

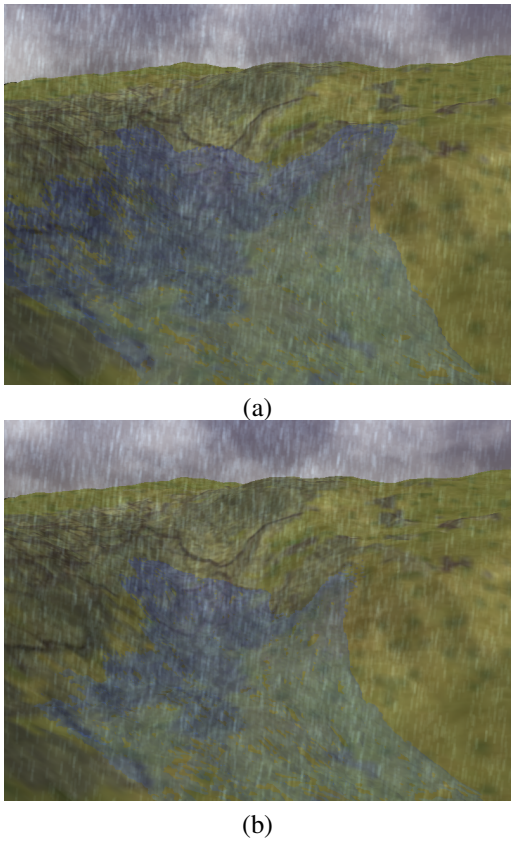


(a)

(b)

Fig. 5. Changing environment map color from red to green.

2) Precipitation Scale: The lake formation can be observed in Figure 6.(a) that shows a concave region in the DTM and the water accumulation along time evolution. In this figure, we can see the enlargement of the flooded area with the downhill path over the terrain surface and affected regions. We explore this example to study the effect of the precipitation scale parameter K in expression (22). The Figure 6.(a) shows the flow configuration in the step 2880 of the simulation when the precipitation scale factor is $K = 2$, while the Figure 6.(b) illustrate the same simulation step, but with precipitation scale $K = 4$. As expected, we get a lower rate of water accumulation in Figure 6.(b).



(a)

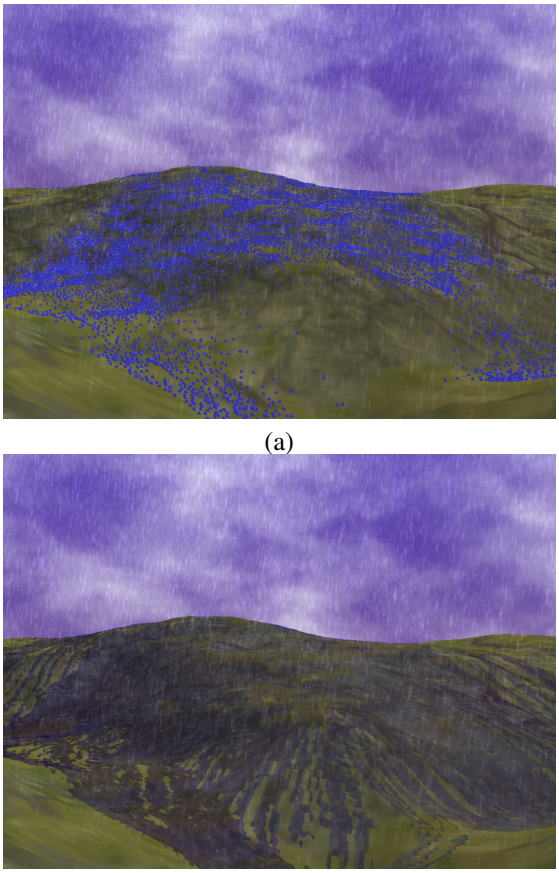
(b)

Fig. 6. Concave region in the DTM with surface flow. (a) Water accumulation after 2880 steps of rain simulation with $K = 2$ in expression (22). (b) Configuration at time 2880 but with $K = 4$ in expression (22).

Despite of this, we observe similarities between the surface flow patterns in Figures 6.(a) and 6.(b), which indicate that we can change the precipitation scale parameter to slow down the surface flow animation but still reproducing physically plausible effects. This example also indicates that such simulation can be useful not only for computer graphics applications, but also for water resources studies and flood evolution analysis. So, high consequence areas that could be impacted can be identified.

3) Carpet: The example of Figure 7 demonstrates the capability of the carpet, inside our framework, for the simulation of river and lake formations over a complex topography. In this figure, we placed the rain source in a large valley of the DTM. The Figure 7.(a) shows the

SPH particles while the 7.(b) pictures only the carpet of the simulation at the same time step. The visual effects of Fresnel law of refraction/reflection and environment mapping are presented in the carpet rendering. The flow reaches all the way down going towards concave regions in the terrain. Although there are DTM regions with sparse particle distribution (left-hand corner of the bottom of Figure 7) we observe that the carpet faithfully fills the topography.

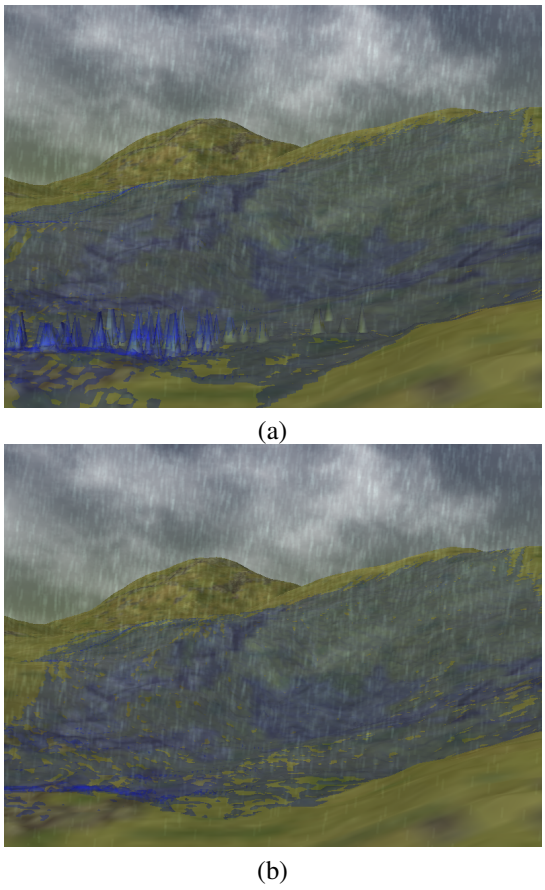


(a)

(b)

Fig. 7. (a) Surface flow visualization using only SPH particles. (b) Visualizing surface flow using carpet without SPH particles.

4) Free Surface Tension: Now, we demonstrate the effect of the free surface tension, defined by expression (10). The Figures 8.(a)-(b) were generated with the same parameters and conditions but the first one did not apply the superficial tension. We can observe some instabilities in the free surface of Figure 8.(a) due to the density gradient nearby the free surface. In fact, particles tend to *scape* from the fluid which generates the patterns observed in some regions of the fluid. Things are clearly different in Figure 8.(b) where the free surface (carpet) is much more smooth and realistic.
5) Splash Animation: Figure 9 shows the splashing of raindrops, in a augmented scale, on the fluid free surface (carpet). A liquid drop hitting a surface often flattens into a thin sheet that then bounces to form a crown shape [23]. The texture sequence in Figure 4 pictures this phenomenon. In the Figure 9 we shall observe the effects obtained with point sprite rendering to combine the splash patterns with the local surface flow color (light blue spots). However, in our case, the local orientation of the carpet must be considered, following the method described in section III, that is, according to the range of the angle between the z-axis and the collision normal. That is way we observe different sprays of raindrop depending on the terrain slope.



(a)

(b)

Fig. 8. (a) Simulation without considering free surface tension. (b) Free surface flow with surface tension.

B. Computational Efficiency

To perform frame-rate tests to measure the FPS of the framework we must take into account the computational cost of animations involving Rain (R), Collision Detection (CD), SPH, Carpet (C) and visual effects like Splashes (S). We vary the K value in expression (22) and measure the FPS for three kind of simulations: (1) Including the rain, collision detection and SPH particles animation (R+CD+SPH), reported in the first line of Table I; (2) Simulating R+CD+SPH as well as the Carpet visualization (second line of Table I); (3) Including R+CD+SPH+C besides Splashes (R+CD+SPH+C+S).

The time scale of the rain is 0.025 and for the SPH simulation is 0.15. The DTM is the used in Figure 8. All the

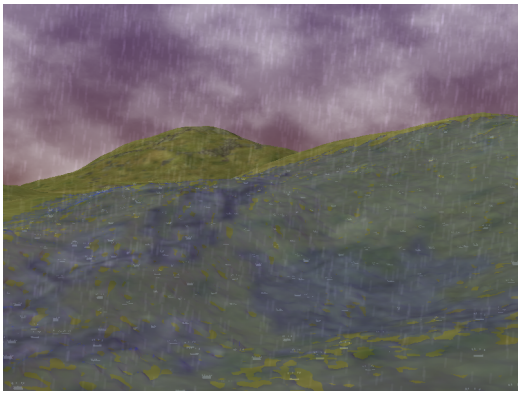


Fig. 9. Visual effect (in a augmented scale) obtained with splash animation.

TABLE I
TABLE LISTING THE FPS WHEN CONSIDERING RAIN (R), COLLISION DETECTION (CD), SPH, CARPET (C) AND SPLASHES (S).

	Precipitation Scale (K)			
	3	4	5	
R+CD+SPH	21.93	39.68	67.57	FPS
R+CD+SPH+C	22.51	29.07	39.68	
R+CD+SPH+C+S	18.38	26.04	33.78	
SPH Particles	7619	5734	2528	

examples were simulated until 2500 frames. The number of rain particles that enter the scene along the whole simulations is around 60000.

Table I provides the FPS obtained for each simulation. The last line of Table I gives the total number of SPH particles that was generated, following expression (22), along the simulation.

We observe that the framework achieves real-time performance for $K = 3$. For $K = 4$ it is beyond real-time FPS in the first line but it is nearby real-time in the other cases. In the case of $K = 5$ the animation achieves a FPS that goes behind the real time frame rate in all simulations.

It is clear from these results that the frame rate is very dependent from the SPH simulation, whose computational cost depends on the number of SPH particles (when increasing K we reduce the rate of SPH particles generation). Therefore, we do expect that a GPU implementation of the SPH will improve the performance of the proposed framework.

V. CONCLUSIONS AND FUTURE WORKS

In this work we presented a framework for rain scene animation in DTMs. The animating framework is composed by a rain animation technique in GPU and SPH surface flow simulation over the DTM. In the experimental results we emphasize the power of the proposed model when combined with efficient techniques for rendering. The reported frame rates achieve real-time performance in some setups being very promising.

A future direction for this work is the realization of a parallel implementation of the 3D fluid model using GPU or multicore platforms to improve performance. Besides we plain to add the simulation (rain plus surface flow) in a real video.

ACKNOWLEDGMENT

We would like to thank CAPES/CNPq for the support provided for this work. We also thank Sicilia F. Judice and Ricardo Marroquim for all discussions and help.

REFERENCES

[1] O. Deusen, D. S. Ebert, R. Fedkiw, F. K. Musgrave, P. Prusinkiewicz, D. Roble, J. Stam, and J. Tessendorf, "The elements of nature: interactive and realistic techniques," in *ACM SIGGRAPH 2004 Course Notes*, 2004, p. 32.

[2] P. Rousseau, V. Jolivet, and D. Ghazanfarpour, "Realistic real-time rain rendering," *Computers & Graphics*, vol. 30, no. 4, pp. 507–518, 2006.

[3] A. Puig-Centelles, O. Ripolles, and M. Chover, "Creation and control of rain in virtual environments," *The Visual Computer*, vol. 25, no. 11, pp. 1037–1052, Nov. 2009.

[4] K. Garg and S. Nayar, "Photorealistic rendering of rain streaks," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 996–1002, 2006.

[5] S. Stanik and W. Werman, "Simulation of rain in videos," *Int. J. Comput. Vis. Texture*, pp. 95–100, 2003.

[6] L. Wang, Z. Lin, T. Fang, X. Yang, X. Yu, and S. Kang, "Real-time rendering of realistic rain," in *SIGGRAPH*, 2006, p. 156.

[7] K. Garg and S. K. Nayar, "Detection and removal of rain from videos," *IEEE Conference on Comp. Vis. and Patt. Recognition*, vol. 1, pp. 528–535, 2004.

[8] N. Wang and B. Wade, "Rendering falling rain and snow," in *SIGGRAPH*, 2004, p. 14.

[9] Z. Feng, M. Tang, J. Dong, and S. Chou, "Real-time rain simulation," in *Comp. Supp. Coop. Work in Design (CSCWD)*, 2005, pp. 626–635.

[10] N. Tatarchuk, "Artist-directable real-time rain rendering in city environments," in *SIGGRAPH Courses*, 2006, pp. 23–64.

[11] S. Tariq, "Rain," in *Technical report (NVIDIA)*, 2007.

[12] Y. Yang, X. Zhu, J. Mei, and D. Chen, "Design and realtime simulation of rain and snow based on lod and fuzzy motion," in *ICPCA*, 2008, pp. 510–513.

[13] R. Green, "Spherical harmonic lighting: The gritty details," *Archives of the Game Developers Conference*, March 2003. [Online]. Available: http://citeseer.ist.psu.edu/contextsummary/2474973/0

[14] P.-P. Sloan, J. Kautz, and J. Snyder, "Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments," in *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM, 2002, pp. 527–536.

[15] G. R. Liu and M. B. Liu, *Smoothed particle hydrodynamics : a meshfree particle method*. New Jersey: World Scientific, 2003.

[16] T. Lenaerts, B. Adams, and P. Dutré, "Porous flow in particle-based fluid simulations," *ACM Trans. Graph.*, vol. 27, no. 3, pp. 1–8, 2008.

[17] T. Harada, S. Koshizuka, and Y. Kawaguchi, "Smoothed particle hydrodynamics on GPUs," in *Proc. of Computer Graphics International*, 2007, pp. 63–70.

[18] M. Müller, D. Charypar, and M. Gross, "Particle-based fluid simulation for interactive applications," in *Proceedings of ACM SIGGRAPH symposium on Computer animation*, 2003.

[19] P. Kipfer and R. Westermann, "Realistic and interactive simulation of rivers," in *Proc. of Graphics Interface*, 2006, pp. 41–48.

[20] C. Hirsch, *Numerical Computation of Internal and External Flows*. Livros Tecnicos e Cientificos Editora S.A., RJ-Brasil, 1988, vol. 1.

[21] C. Schlick, "An inexpensive BRDF model for physically-based rendering," *Computer Graphics Forum*, vol. 13, pp. 233–246, 1994.

[22] Z.-X. Feng, M. Tang, J.-X. Dong, and S.-C. Chou, "Real-time rendering of raining animation based on the graphics hardware acceleration," *Int. Conf. on Comp. Supp. Cooperative Work in Design*, vol. 2, pp. 734–739, 2005.

[23] S. Mandre, M. Mani, and M. P. Brenner, "Precursors to splashing of liquid droplets on a solid surface," *Phys. Rev. Lett.*, vol. 102, no. 13, Mar 2009.