Drift-Diffusion Based Real-Time Dynamic Terrain Deformation

M. Gilardi, P. L. Watten and P. Newbury

Dept. Engineering and Informatics, University of Sussex, UK

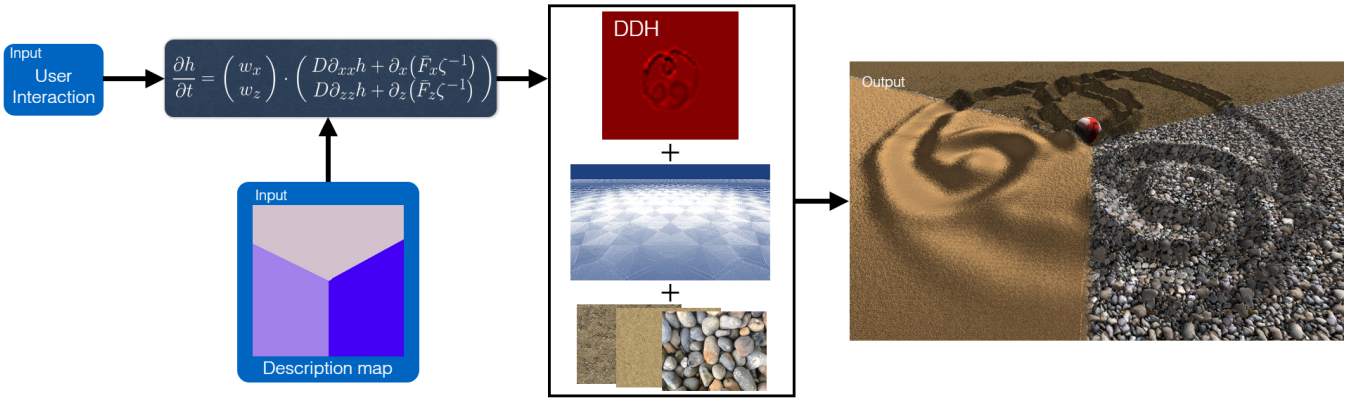

Figure 1: *Diagram of the method described in this paper. A modified form of the drift-diffusion equation is used, together with the parameters information contained in a description map and the information about objects interacting with the terrain, to update a Dynamic Displacement Map. This is then added to the height-map of the terrain to produce the terrain deformation due to object interaction.*

Abstract

In the natural world, terrains are dynamic entities which change their morphology due to their interaction with other agents in the environment. However, in real-time applications terrains are often represented as static meshes, which present no interaction capabilities. This paper presents a novel real-time 2D method for dynamic terrain simulations, aimed for applications in the entertainment industry. This method is based on a Dynamically-Displaced Height-map and on the numerical solutions, obtained using an Euler method, of a modified drift-diffusion equation. The method allows objects to interact with the terrain and to deform it in real time, it is easy to implement and generates different kinds of realistic tracks depending on the soil composition.

Categories and Subject Descriptors (according to ACM CCS): I.3.0 [Simulation and Modeling]: Types of simulation — Visual

1. Introduction

Dynamic terrains are terrain representations that are capable of changing their morphology due to their interaction with the user or other agents in the environment. Dynamic terrains can be classified in three categories: physics based, appearance based, or hybrid. Physics based dynamic terrains [MKM89, LM93, CLH96, ZB05, PCGFMD06] produce realistic deformations but are computationally expensive and, due to the nature of soil physics, often difficult to implement for generic soils. By contrast, appearance based methods [SOH99, Ono03, ON05, Aqu06] improve simulation performances to the detriment of accuracy, but they are also often elaborate to implement. Recently developed hybrid methods [ZCO11] try to balance the correctness of physics based methods and the performance of appearance based methods in order to obtain visual and physical realism during the simulation. However, their implementation is still elaborate.

The method presented in this paper is an appearance method that is loosely inspired by Zhu and Bridson [ZB05] method, which simulate sand as a viscous fluid by incorporating friction into the Navier-Stokes equations. The method presented in this paper also represent soil as a fluid with low Reynolds number but it aims to speed and simplicity of implementation, and even though it is physically inspired it focuses on producing visually believable tracks.

Developing real time methods for terrain deformations is of interest in the video-game industry. The implementation of these methods in the industry either takes into account only soil com-

DOI: 10.2312/egsh.20161011

pression [Rei10, Øru15], thus ignoring material displacement, see figure 2, or uses multiple overlapping meshes and "push/extrude" primitives to deform the terrain, thus coupling the method with a specific shape and making it difficult to implement and manage, see section "2.c. Mud" of [Zag13].

Figure 2: *Comparison of tracks left by a vehicle in an interactive application (top), image courtesy of Andres Ørum [Øru15], and tracks left in a natural terrain, notice that the terrain is both compressed and displaced at the side of the track.*

This paper presents a novel method for dynamic terrain simulations, see figure 1, capable of representing different kinds of soils in real time. Although the tracks obtained with the method presented in this paper are similar to those obtained with other methods, such as Sumner [SOH99], Onoue and Nishita [Ono03, ON05] or Holz et al [HBK09], the methods here presented is simpler to implement. The method computes changes in the terrain height over time by using the numerical solutions of a modified drift-diffusion equation, explained in section 2. These numerical solutions are used to update a Dynamically-Displaced Heigh-map (DDH) [Aqu06] which is then used to update the height map of the terrain and for shading purposes, as explained in section 3. The method is implemented on a GPU through the use of a compute shader, section 3. Results and future work are discussed in sections 4 and 5.

2. Method

The Zhu and Bridson [ZB05] method uses a modified version of the Navier-Stokes equations to simulate sand, these equations compute the change of momentum of fluid over time and take into account

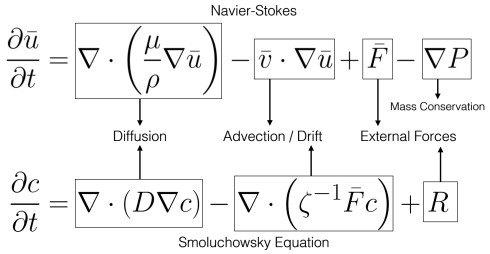

Figure 3: *Structural similarity between the Navier-Stokes equation and the Drift-Diffusion equation. Notice that in the Drift-Diffusion equation the mass conservation term is missing.*

friction forces. It can be observed that the general structure of the Navier-Stokes equations is similar to other physics models, such as the drift-diffusion model [DE86, Smo16], see figure 3, that can be adapted more easily to incorporate object interactions and terrain deformations. Height-map representations are a standard way to represent terrains in real-time applications. In these representations a deformation of the terrain corresponds to a local change of the scalar height field $h(x,z,t)$ over time. The remainder of this section explains how to compute a local change of $h(x,z,t)$ over time by using a modified version of the drift-diffusion equation.

2.1. Mathematical Model

To obtain the mathematical model the drift-diffusion equation, also known as the Smoluchowsky equation [DE86, Smo16], is applied to the scalar height field $h(x,z,t)$. In the following formulae explicit dependency from space and time will be dropped for simplicity of notation. Under the assumption that the diffusion coefficient D is constant, the drift-diffusion equation is [DE86]:

$$\frac{\partial h}{\partial t} = D\Delta h - \nabla \cdot \left(\bar{F}\zeta^{-1} h\right) \tag{1}$$

where Δh is the Laplacian operator applied to the scalar field h, ζ is the friction constant of the diffusion medium, $\nabla\cdot$ is the divergence operator, and $\bar{F}$ is an external force field generating an average velocity in the medium. In this paper $\bar{F}$ will be interpreted as the external force field exerted by an object interacting with the terrain. In this paper equation (1) is modified by removing from the force term the dependency from h, thus obtaining:

$$\frac{\partial h}{\partial t} = D\Delta h - \nabla \cdot \left(\bar{F}\zeta^{-1}\right) \tag{2}$$

The justification for this modification is that the action of the force over the terrain does not depend on the height of the terrain but only on the terrain friction and the force itself. Every granular material when piled up rests with a specific angle α with respect the ground, this angle is called the *angle of repose* and it is a property of the material. However, equation (2) does not take the angle of repose of the material into account and diffuses piles of material until the ground is levelled. To include the angle of repose in equation (2) the slope of the pile of material at time t is split into two angles, one measured on the xy plane the other measured on the zy plane, where y is the up direction.

$$\alpha_k = \sin d_k \approx d_k \tag{3}$$

where $k \in \{x,z\}$ and the d_ks are the displacements along x and z on a single frame, which are small quantities between subsequent frames, and will be obtained in the following. The values obtained from equation (3) are mapped in $[0,1]$ with the following mapping:

$$w_k = clamp\left(\frac{\alpha_k - \alpha}{\frac{\pi}{2} - \alpha}, 0, 1\right) \tag{4}$$

where α is the angle of repose characteristic of the material and $k \in \{x,z\}$ and *clamp* forces w_k in $[0,1]$. Equation (2) is then re-written as:

$$\frac{\partial h}{\partial t} = \bar{w} \cdot \begin{pmatrix} D\partial_{xx}h - \partial_x \bar{F}_x \zeta^{-1} \\ D\partial_{zz}h - \partial_z \bar{F}_z \zeta^{-1} \end{pmatrix} = \bar{w} \cdot \begin{pmatrix} d_x \\ d_z \end{pmatrix} = \bar{w} \cdot \bar{d} \tag{5}$$

where $\bar{w} = (w_x, w_z)$ is the vector which has as components the angle's weights obtained in equation (4).

2.2. Discretization

Due to their simplicity of implementation explicit finite differences methods have been chosen to obtain a discrete form of equation (5). The discrete form is obtained using a square $n \times n$ grid with cells of side Δx equal to one. The scheme used is forward in time, centred in space for the first and second order terms on a von Neumann (4-cells) neighbourhood. Using the aforementioned scheme the first component of the vector $\bar{d}$ in equation (5) becomes:

$$\begin{aligned}(d_x)^n_{ij} \quad = \quad & D(h^n_{i-1\,j} + h^n_{i+1\,j} - 2h^n_{ij}) - \\ & 0.5\zeta^{-1}((F_x)^n_{i+1\,j} - (F_x)^n_{i-1\,j})\end{aligned} \tag{6}$$

similarly for the second component. The discrete form of equation (5) is then:

$$h^{n+1}_{ij} = h^n_{ij} + \Delta t \left(w^n_{ij} \cdot d^n_{ij}\right) \tag{7}$$

where w^n_{ij} is the discrete form of the weight vector in equation (4) obtained from:

$$(\alpha_k)^n_{ij} = |(d_k)^n_{ij}| \tag{8}$$

where $k \in \{x,z\}$.

3. Implementation

The numerical scheme in section 2.2 has been implemented on a GPU using a DirectX11 compute shader with 16×16 threads per group and tested for different grid sizes as explained in section 4. Two structured buffers have been passed to the compute shader, one containing the grid properties and one containing the properties of the objects interacting with the terrain. In addition to the buffers in the device global memory, two padded buffers of size 18×18 elements in shared memory have been used to store the data used in the group during the evaluation of equation (7). These buffers speed up the data access during computation. The first shared buffer contains the grid data while the second contains the distribution of forces applied by the interacting objects. The force distribution has been obtained by spreading the force exerted by the object over the area of interaction. After data fetching from global to shared memory equation (7) has been evaluated and the results have been stored in the DDH as a 2D texture. To deform the terrain mesh a tessellation shader has been used to tessellate a neighbourhood of the point where the camera is looking. The DDH obtained from the compute shader is used in the tessellation domain shader to displace the vertices of the mesh, and in the fragment shader to compute the normals to the surface dynamically. The dynamic computation of normals from the DDH allows the use of dynamic levels of details for the terrain mesh as the resulting shading simulates the mesh deformation in a visually convincing way when the deformed area is far from the camera. Moreover, the blending of dynamic normal mapping and texture normal mapping allows the removal of the grid artefacts which appear at the base of the deformation.

4. Results

Results shown in figure 4 have been produced using a 4 core Intel(R) Xeon(R) CPU at 3.4 GHz, 8 GB RAM, and a NVIDIA Quadro 2000 graphics card with a rendered frame size of 1280×720. Table 1 reports times for different grid sizes and tessellation factors, times do not depend on the kind of soil simulated and represent simulation and rendering time for a single frame. In the last row the running time of the compute shader has been isolated. The method proposed scales linearly with the grid size and even with the largest grid and highest tessellation factor the simulation still runs over 60 fps. Figure 4 and the companion video show that different kind of terrains can be simulated changing the parameters ζ^{-1}, D and α in equation (5). All images have been obtained on a grid of size 512×512, for the sand and mud scenes the tessellation factor has been set to 8, while for the pebbles scene the tessellation factor has been set to 64 to better simulate the pebbles using a displacement map. The displacement map for this scene has been obtained rendering the depth buffer of the pebbles models reconstructed using the method described in [GWN14]. As shown in the companion video, objects can dynamically change sizes during the simulation. As long as the size of the object stays bigger than the size of one cell the method proposed is capable of capturing the object track on the terrain. During simulation a loss of volume has been noticed during pile of sand simulations while a slight increase of volume has been noticed during the interaction of objects with the terrain, which are due to a lack of mass conservation constraints in the simulation. The simulation produces good looking results with just one iteration but the material looks slightly viscous, increasing the num-

Table 1: *Average rendering times expressed in milliseconds for different grid sizes and tessellation factors. On the top of the table times refer to the rendering of a single frame comprising tessellation, a single iteration of the compute shader, dynamic normals computation, texturing, lighting and rasterization. The last row isolates the running time of a single iteration of the compute shader.*

Grid sizes Tessellation	512×512	1024×1024	2048×2048
8	2.12 *ms*	3.88 *ms*	10.56 *ms*
16	2.32 *ms*	4.08 *ms*	10.78 *ms*
32	2.80 *ms*	4.55 *ms*	11.28 *ms*
64	4.32 *ms*	6.08 *ms*	12.85 *ms*
Compute Shader	0.58 *ms*	2.09 *ms*	7.97 *ms*

Figure 4: *Left: comparison of tracks left on simulated terrains with track left on real terrains, notice that both simulations mimic both the compression of the soil and its displacement to the side of the track. Right: multi-material terrain. Parameters for the materials: Mud:* $\zeta^{-1} = 6 \times 10^{-4}$, $\alpha = \frac{2}{5}\pi$, $D = 0.25$*; Pebbles:* $\zeta^{-1} = 3 \times 10^{-4}$, $\alpha = \frac{\pi}{3}$, $D = 0.9$*; Sand:* $\zeta^{-1} = 4 \times 10^{-4}$, $\alpha = \frac{\pi}{4}$, $D = 1$*;*

ber of iteration reduces this problem. Visual comparisons between the simulation's results and tracks on real soil, are shown on the left side of figure 4, show that the method is well suited for applications in video-games and real-time applications.

5. Conclusions and Future Work

In this paper a novel method based on the drift-diffusion equation for dynamic terrain simulation has been presented. The method, implemented on GPU, is capable of simulating different kinds of materials in real time. Even though it produces visually appealing deformations, the method can be improved in many ways. Future work will focus on solving the issues reported in section 4. As an explicit method has been used to obtain the discrete form of equation (5) the simulation is stable for small time-steps, $\Delta t < 0.3$ has been used in this paper, stability can be improved by using an implicit method. Finally, the implementation can be optimized to take better advantage of the GPU architecture, for example by using either the method proposed by Schäfer et al [SKNn*14] or by Yusov [Yus12]t. Despite its shortcomings, the method presented in this paper is easy to implement and suitable for interactive applications and games.

References

[Aqu06] AQUILIO A. S.: *A framework for dynamic terrain with application in off-road ground vehicle simulations*. PhD thesis, 2006. 1, 2

[CLH96] CHANCLOU B., LUCIANI A., HABIB A.: Physical Models of Loose Soils Dynamically Marked by a Moving Object. In *Proceedengs 9th IEEE Comput. Animat. Conf.* (1996), pp. 27–35. 1

[DE86] DOI M., EDWARDS S. F.: The Smoluchowsky equation. In *Theory Polym. Dyn.* Oxford Science Publications, 1986, ch. 3.2, pp. 46–50. 2

[GWN14] GILARDI M., WATTEN P. L., NEWBURY P.: Unsupervised three-dimensional reconstruction of small rocks from a single two-dimensional image. In *Eurographics 2014 Short Pap.* (2014), pp. 29–32. 3

[HBK09] HOLZ D., BEER T., KUHLEN T.: Soil Deformation Models for Real-Time Simulation: A Hybrid Approach. In *Proceedings of the 6th Workshop on VRIPHYS* (2009), pp. 21–30. 2

[LM93] LI X., MOSHELL J. M.: Modeling Soil : Realtime Dynamic Models for Soil Slippage and Manipulation. In *Proc. 20th Annu. Conf. Comput. Graph. Interact. Tech. (ACM SIGGRAPH)* (1993). 1

[MKM89] MUSGRAVE F. K., KOLB C. E., MACE R. S.: The Synthesis and Rendering of Eroded Fractal Terrains. *Comput. Graph. (ACM). 23*, 3 (1989), 41–50. 1

[ON05] ONOUE K., NISHITA T.: An Interactive Deformation System for Granular Material. *Comput. Graph. Forum 24*, 1 (Mar. 2005), 51–60. 1, 2

[Ono03] ONOUE K.: Virtual Sandbox. *Proceedings. 11th Pacific Conf. Comput. Graph. Appl. 2003.* (2003), 252–259. 1, 2

[Øru15] ØRUM A.: Unity realistic vehicle physics - real time terrain deformation test 1, 2015. URL: https://www.youtube.com/watch?t=2&v=0MOvirz0UKU. 2

[PCGFMD06] PLA-CASTELLS M., GARCÍA-FERNÁNDEZ I., MARTÍNEZ-DURÁ R. J.: Interactive terrain simulation and force distribution models in sand piles. *Cell. Autom. - Lect. Notes Comput. Sci. 4173* (2006), 392–401. 1

[Rei10] REIMER D.: Real-time terrain deformation in unity3d, 2010. URL: http://blog.almostlogical.com/2010/06/10/real-time-terrain-deformation-in-unity3d/. 2

[SKNn*14] SCHÄFER H., KEINERT B., NIESS NER M., BUCHENAU C., GUTHE M., STAMMINGER M.: Real-Time Deformation of Subdivision Surfaces from Object Collisions. In *Proc. 6th High-Performance Graph. Conf.* (2014), pp. 1–8. 4

[Smo16] SMOLUCHOWSKY M. V.: Über Brownsche Molekularbewegung unter Einwirkung äußerer Kräfte und deren Zusammenhang mit der verallgemeinerten Diffusionsgleichung. *Ann. Phys. 353*, 24 (1916), 1103–1112. 2

[SOH99] SUMNER R. W., O'BRIEN J. F., HODGINS J. K.: Animating Sand, Mud, and Snow. *Comput. Graph. Forum 18*, 1 (Mar. 1999), 17–26. 1, 2

[Yus12] YUSOV E.: Real-Time Deformable Terrain Rendering with DirectX 11. In *GPU Pro 3*. 2012, ch. 2, pp. 13–39. 4

[Zag13] ZAGREBELNYY P.: Rendering and simulation in offroad driving game, 2013. URL: http://www.gamasutra.com/blogs/PavelZagrebelnyy/20130613/194247/Rendering_and_simulation_in_offroad_driving_game.php. 2

[ZB05] ZHU Y., BRIDSON R.: Animating Sand as a Fluid. *ACM Trans. Graph.* (2005), 965–972. 1, 2

[ZCO11] ZHU Y., CHEN X., OWEN S. G.: Terramechanics Based Terrain Deformation for Real-Time Off-Road Vehicle Simulation. *Adv. Vis. Comput. - Lect. Notes Comput. Sci. 6938* (2011), 431–440. 1