A Real-time Implementation of the Dynamic Particle Coating Method on a GPU Architecture

Kevin Sillam, Matthieu Evrard, Annie Luciani

4th Workshop in Virtual Reality Interactions and Physical Simulation "VRIPHYS" (2007), pp. 1-10
J. Dingliana, F. Ganovelli (Editors)

A real-time implementation of the dynamic particle coating method on a GPU architecture

Kevin Sillam[1], Matthieu Evrard[1] and Annie Luciani[1,2]

[1]ICA laboratory, INP Grenoble, France
[2]ACROE, INP Grenoble, France

Abstract

This paper deals with a real-time implementation on graphic processor of the "dynamic particle coating method" (DPCM) first proposed by [HL02] and founded on a mass-interaction formalism. When this method was proposed, it was too much time-consuming to be inserted in an interactive application. This paper solves this major drawback. Our real-time implementation allows inserting this method in a real-time simulation chain composed of a haptic device, an upstream mass-interaction model that interacts with the user through this device and the new real-time implementation of the DPCM method to visualize this model

Categories and Subject Descriptors (according to ACM CSS): I.3.5 Physically based modeling, I.3.3 real-time visualization of particle models, I.3.1 GP/GPU, I.3.6 haptic interactions

1. Introduction

At the heart of the engraving process, there is a relationship between two bodies, one that is hard and that can be called "marker" and that modifies the state of another one that can then be called the "engraving surface".

When considering natural phenomena, a sedimentary rock that shows a fossil is a durable engraving surface marked by a prehistoric animal body. On the contrary, water surface can be seen as a more voluble engraving surface for the memory of a ricochet for instance. But the engraving surface is always a memory, a trace of the marker presence, of the effect that it had on the engraving surface, whatever fleeting this memory can be.

A certain number of animation movies have used engraving as a central process for the creation of pictures. Indeed, the engraving process allows an artist to obtain shapes with fuzzy, fleeting outlines.

Amongst the amount of such work, Alexandre Alexeïeff's approach is to be noticed [Ben01]. In the early 30's, this atypical engraver built an engraving surface made of thousands of pins able to orthogonally slide along a fastening plan. Alexeïeff engraved this surface by moving these pins thanks to various marking objects. Four light sources light up this "pinscreen". White areas appear where the pins are the lowest and black areas when they are the highest. The successive shots taken by a camera with various configuration changes of the pinscreen artificially create animated pictures.

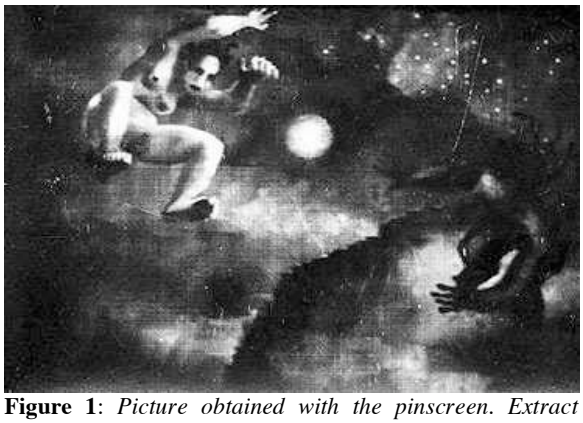

Figure 1: *Picture obtained with the pinscreen. Extract from Alexeïeff's short film "Eine Nacht Auf Dem Kahlen Berge", Berlin, 1933.*

In Computer Graphics community, Alexeïeff's pinscreen has been implemented by Faria Lopes et al. [FR92] to make this engraving technique possible through the computer.

On a different scale than Faria Lopes et al., Habibi et al. [HL02] goal is not to implement a new pinscreen method but to use this engraving metaphor, called "Dynamic

Particle Coating Method" (DPCM), in order to solve a part of the problem of particle models visualization.

Interacting particles whose movement is simulated are able to product various rich behaviors as different as deformable and non-deformable objects, smoke, pasts, liquids, crowds, etc. [ECL06]. It is also well adapted to interact with humans through haptic devices. But a major limit to this kind of formalism when producing animated pictures is that the space that particles occupy is restricted to moving points if a visualization process, a coating mean, is not applied. How do we pass from points to shapes?

This is an opened question that can hardly be solved in most cases. Partial answers can be brought by: (1) adding geometrical primitives to the points; (2) controlling explicit representation of shapes (polygons, NURBS) by the moving points; (3) extending points by using potential fields with isosurface as in the implicit surfaces methods [Bli82, BBB*97]. The first types of method (based on explicit geometrical features) are unwieldy to render fuzzy or highly deformable phenomena. The DPCM method can be considered as a method close to the implicit surfaces principle by considering that each moving point is creating a potential field, whose profile is evolving along time according to the dynamic of the point (velocities and acceleration) whereas implicit surface evolution results from a fixed potential field.

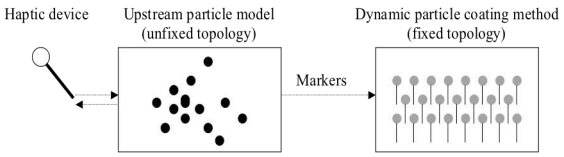

Figure 2: *Real-time simulation chain. Haptic device – upstream model – DPCM.*

Habibi's approach is to model a physical phenomenon (smoke, water, etc) by a cascade of two physically based particle models (see Figure 2). The first one (the upstream model) has a variable topology. For example, in smoke model, there are several moving masses between which the graph of interactions varies according to the distances or the velocities [HL02]. It models a general behavior and interacts with human through an haptic device. The second one, the DPCM, has a fixed topology. This means that masses are linked by a predefined graph of interactions (as it is the case in a deformable visco-elastic surface). These masses are not moving in the whole space. They are attached to a grid in the space and they only deform themselves. The DPCM is *engraved* by the first upstream model. This engraving is a deformation around the points of the upstream model that engrave it. DPCM is thus able to add to the points of the upstream model a spatial evolving extension. Following the metaphor of engraving, the upstream model plays the role of the markers and the DPCM plays the role of the engraving surface. The difference with material static engraving surface (such as copper plate, wood, etc.) is that the DPCM, as it is physically modeled, exhibits other physical behaviors (such as fluid propagation as if one engraves in water, plastic pastes, sand, …).

Similarly with the Alexeïeff's pinscreen, the engraved surface is discretized at the resolution of the final display (in our case the computer screen), but each pin is replaced by a dynamic pin able to move by itself. The marking objects, the markers come from the upstream simulation that sends particles positions that evolve in time.

Convincing sequences of smokes, sands, waters and even solids have been produced by [HL02]. However, their method was relatively time-consuming and it wasn't possible to integrate it inside an interactive application that could include an haptic device for VR applications.

This paper aims at using "shaders", which are programs that are compiled and executed on modern graphic cards, and taking advantage of calculation parallelization they offer to obtain a real-time implementation of the DPCM method described by Habibi et al. Indeed, the fixed and regular topology of the DPCM suits well to large parallel calculation.

Section 2 briefly reminds the DPCM method. Section 3 draw a brief state of the art on particle models on GPU. Section 4 deals with our implementation. Section 5 shows the incorporation of the method in a real-time simulation chain that includes an upstream mass-interaction network simulation. Section 6 gives obtained performances. Section 7 proposes an end-user interface. Section 8 finally exhibits results. The most of them are new ones because human gesture intervenes.

2. The Dynamic Particle Coating Method (DPCM)

More details on the DPCM can be found in [HL02]. One could notice that the DPCM is a mass-interaction network model as described in [LJF*91].

The inputs of the DPCM are a set of positions of points along time. These points are the "markers" of the engraving process. They mark, they engrave, the DPCM, i.e. they act on it by creating tracks but the DPCM do not act on it in return. That's why we can speak about the coating of an upstream particle model. We then simulate upstream particle model and the DPCM in separate architectures. Actually, any process like a simulation, gestural captors or the reading of a motion file can supply evolutions in time of the markers.

The DPCM is composed of a set of physical elements called "phyxels" (for "physical elements") by Habibi et al.

Such phyxels are slightly different than those defined by [MKN*04]. They are composed of a mass linked to a fixed point by a one-dimensional physical interaction (figure 3). The simplest interaction is the linear damped-spring but more complex interaction such as plastic one will also be used in this paper.

To model spatial physically based extensions of the tracks made by the markers, physical interactions are placed between phyxels. Each mobile mass interacts with the mobile masses of its neighboring phyxels (and more generally with any other phyxels). Our present implementation of the DPCM is a 2D implementation, i.e. a 2D array of phyxels, in which each phyxel interacts physically with its 8 neighbors.

The markers are physically acting on each mobile mass of the phyxels, pushing the phyxels and engraving the DPCM, perpendicularly to the DPCM 2D surface, creating a 3D engraving. From this action, the DPCM exhibits physical behaviors (deformations, propagations, etc.) according to its own physical parameters (i.e. the parameters of all the interactions between the phyxels).

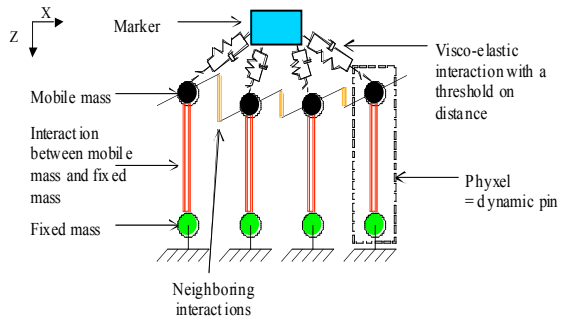

Figure 3: *Mass-interaction model of the DPCM.*

Outputs of the DPCM are deformation scalar values that correspond to the distance between the fixed and the mobile masses of a phyxel. This array of scalar is used as an input of a rendering method. The simplest method proposed by [HL02], is the control of pixels color by linear combination of the deformations values of some neighboring phyxels. An extension is to control surfaces points in a 3D space or orientations of normal orientations. Some other rendering methods are presented at §4.2.3 to extend the results of [HL02].

3. Implementation of particle models on GPU.

Programming on graphical processor units (GPU) has been a success for some years. It is not only used for the rendering of 3D scene but also for any kind of numerical calculations that are well adapted to parallel processing. Indeed, a fundamental difference of GPU with CPU is this structural parallelization of calculations. For a global review of the use of GPU programming in Computer Graphics community see [OLG*05].

A general programming principle on this type of architecture is to stock simulation variables inside GPU-specific data structures like textures (that were initially created to stock the RGBA components of the texture that are usually displayed). Another principle is to use programmable area of the GPU called “shaders” that will process these data on the GPU. Calculation on these data will by parallelized.

A number of recent works have been dedicated to the implementation on GPU of particle systems immersed in force fields [KKK*05] by following these principles. Other implementations have taken into account interactions with rigid objects [KLR04] or with others particles [GW05, CL06]. Our work is quite close to these works while implementation and applications are noticeably different.

In the quoted works, the variables stocked inside textures are 3D positions of particles or of controlling points of shapes directly rendered. A vector value directly corresponds to a geometrical element of the 3D scene. In our case, the scalar values we product with the DPCM cannot directly correspond to a geometrical element in the 3D scene. These values control a rendering process: from the simple direct control of the pixels color to a more complex process (e.g. normal orientation used to determine illumination of a surface). Moreover, interactions that were implemented in quoted papers were simple linear interactions like damped-springs. Another difference is that more complex interactions are used in the model designed by the end-user.

4. Implementation of the DPCM on GPU

4.1. Encapsulation of the simulation variables

Variables and some parameters that will evolve during the simulation are encapsulated in texture data structures.

Given a DPCM of m by n phyxels, a texture whose size is m x n is created on the graphic card (we use rectangular textures to work with normalized coordinates). As a result, each phyxel is located by a (x,y) position inside the texture. For each phyxel, the texture contains its position at time t (red component), its position at time t-1 (green component). The blue component is used to stock interaction parameters that will evolve during the simulation.

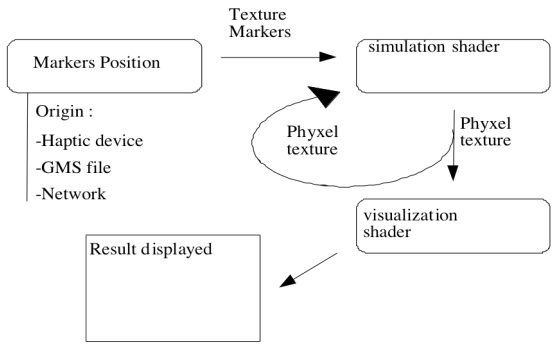

Figure 4: *Global implementation scheme of the software architecture of the DPCM simulation on GPU.*

4.2. Steps in the algorithm

The algorithm is a loop made of three distinct passes (see figure 4).

For each loop we have:

- to execute a "reading" pass during which the current position of each marker is written in a specific texture called "Markers texture".
- to launch execution of the "simulation" pass which writes inside the "phyxels texture" the new phyxel's height that depends on the calculation of the physical algorithm.
- to execute a "rendering" pass which displays an image controlled by the scalar value of the phyxels heights on screen.

4.2.1. The reading pass

A texture is created to encode the positions of markers at time t and t-1. This pass transfers this texture to the GPU. Its size is k x 1 for k markers. The first two components are used to encode positions. The last ones are used to encode parameters of the interaction between a marker and the DPCM like stiffness or a flattening coefficient [see HL02].

We have implemented 3 different origins for markers trajectories:

- A motion file that can be produced by a simulation, which is not necessarily a real-time simulation. We used the motion and gesture encoding file format called GMS (for gesture and motion signal [LEC*06]) that is well adapted to the real-time treatment of this kind of motion data.
- A data flow that comes from a network connection. An upstream simulation can then be done and be synchronized in real-time with the DPCM (see section 5).
- Positions of a gesture captor

As a result, the markers' evolution in time can come from an upstream physical simulation like a particle system or a mass-interaction network. This simulation can temporally be uncorrelated to the DPCM simulation because markers do not act on the DPCM. Nevertheless, the real-time is wished for the DPCM simulation, whatever the markers origin can be, the reading pass can't be a blocking step. That's why latency caused by the hardware access to this data must be strictly included in a simulation time-step.

4.2.2. The simulation pass

The simulation pass takes the markers' positions and the phyxel's height at time t and t-1 (the height of the mobile mass compared to the height of the fixed mass). It writes in a texture called the "phyxels texture" the new values of the phyxels height by using the frame buffer object. It also keeps positions at time t and updates the value of the modifiable parameters of the interactions.

This simulation pass is divided into 4 steps:

- **Calculation of the force applied on each phyxel by the markers**

Each marker interact with each phyxel that have neighboring (x,y) coordinates. This interaction is a simple viscous-elastic interaction with a threshold on distance between the marker and the pin. This interaction is also a unidirectional one [LJF*91] as said before.

- **Calculation of the force applied on each phyxel by its neighbors.**

The mobile mass of each phyxel is linked by an interaction to some of its neighbors. The topological choice of the neighbors has a huge influence on the global dynamic behavior. Section 8.2 shows an interesting result that is obtained while considering a complex topology of neighborhood.

- **Calculation of the force applied on each phyxel by an interaction with the ground.**

This interaction is the interaction between the mobile mass and the fixed mass of each phyxel.

- **Calculation of the new heights of the phyxels**

The new height of the phyxels is assessed from the sum F_n of the forces described before.

New height is given by:

$Z_{n+1} = 2.Z_n - Z_{n-1} + F_n(Tech^2/m)$

Where m is the mass of the phyxel, Z_n and Z_{n-1} the height of the phyxel at time t and t-1, and Tech the sampling period of the movement.

These calculations are made by the "simulation" shader for each phyxel. This shader is a fragment shader of the GPU. The obtained values (heights of the phyxels at time t and t+1, updated parameters of the interactions) are sent through the "phyxels texture" to the rendering pass.

4.2.3. The rendering pass

We make a second pass on the GPU to display the scalar data (the height of the phyxels) on the computer screen. Actually, this pass aims at transforming phyxels into pixels. The way of applying this transformation can be more or less complicated. We have experimented 3 different rendering methods for the DPCM:

- **Mapping of colors:**

The color of the pixel is chosen in accordance with the height of the phyxels. Basically, this is a transformation from 1 scalar to 3 components in a color space like RVB. The user can choose the axis in this space. This kind of rendering is very simple but close to Alexeïeff's pinscreen philosophy. For instance by defining white for a minimal height and black for a maximal height the whole gray scale is obtained (see figure 10).

- **Bump Mapping [Bli78]:**

This technique consists in emulating a relief from a "normal" map and the position of a light source. We obtain this effect by creating a "normal map" with a shader in accordance with the derivation of the phyxels height (considering the neighborhood). It is then possible to brighten or to darken the diffuse light emitted by each pixel to obtain a relief (see figure 5).

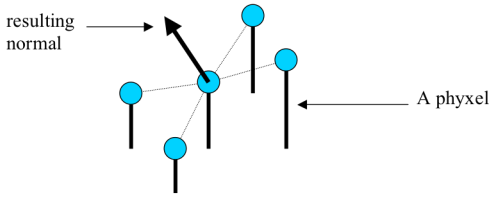

Figure 5: *The resulting normal from the derivation of neighboring phyxels.*

- **Displacement mapping:**

It can also be interesting to visualize the 2D DPCM inside a 3D space. This type of rendering method is notably used to calibrate the screen i.e. placing it in accordance with the markers positions. It also allows use to better understand some dynamic phenomena by watching the DPCM from another point of view.

We use the geometry shader to cut a surface in a huge number of vertices that are controlled by the DPCM. A dynamic surface is thus defined inside a 3D space.

4.2.4. Initialization

The system initialization is done by giving a height to the phyxels equal to the rest length of interaction between the mobile mass and the fixed one.

Noticeably it allows:

- being assured that each phyxel start from the same height
- making a first test on the stability of the DPCM (the heights of the phyxels do not vary before a perturbation).
- preparing the DPCM to interact with markers. This initialization is done each time the user wants the DPCM to be in its initial balance state.

4.2.5. Variable encoding

The first implementation exhibited some noise in the balance state due to data quantification. Indeed, data had been encoded in an 8 bits number clamped in 0 and 1 (low level mechanism in OpenGL). This quantification noise was very huge (see figure 6). It was then necessary to choose a higher appropriate quantification to make this noise disappear. We finally obtained good results with a 32 bits float quantification.

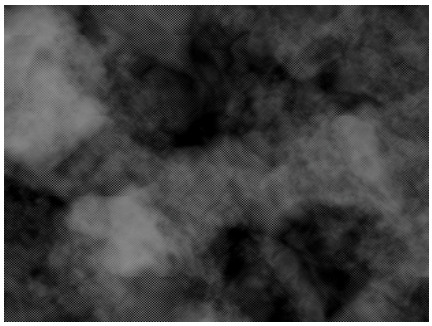

Figure 6: *Noise at the balance state while choosing a bad encoding of the variables (here 8 bits)*

5. The DPCM in a real-time simulation chain

A simulation is done on an upstream computer. It gives to the DPCM the trajectory of a certain number of markers. The problem is to transfer theses data as fast as possible, from this computer to the computer that hosts the GPU.

We have used UDP protocol in an Ethernet network because it is simple and fast. A frame is defined to encapsulate the position data and to deal with the coherence of the received data. Figure 7 describes this frame.

Flag	number of masses	N° Frame	N° Mass	posX	posY	N° Mass
FFFF FFFF	3	245	0	12.5	3.3	1
32 bits	32 bits	32 bits	32 bits	32 bits	32 bits	32 bits

Figure 7: *A frame containing one sample of the upstream simulation*

This organization allows the receiving computer to resynchronize itself on a new frame thanks to the flag. This is a non-blocking reading step that allows both simulations to run in parallel. Communication between both machines is asynchronous because a new frame constrains the DPCM simulation. But the goal is to obtain upstream simulations that are fast enough to prevent this locking. The number of masses is important information when the number of markers changes during the simulation. The DPCM is then able to readapt itself when this change occurs. The frame number locates a lost frame while the mass number avoids a discrepancy in the markers indexes. If a frame is not correct, the user is warned and this frame is ignored.

Figure 8 shows a psychophysics Virtual Reality experiment using a real-time platform. On the bottom left, on can see the haptic feedback device that is connected to the simulation.

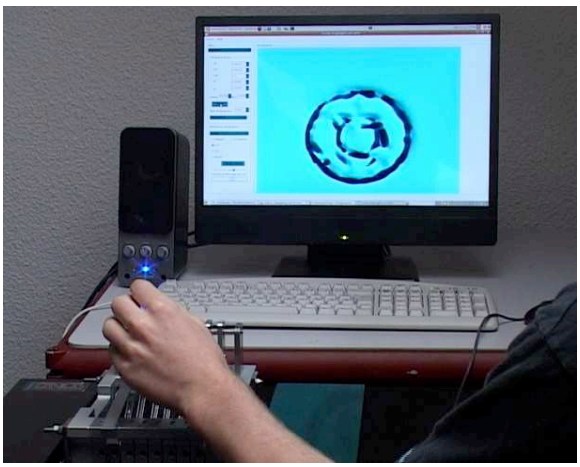

Figure 8: The DPCM within a real-time simulation platform

6. Obtained performances

The DPCM simulation has been achieved on a GTX 8800 (768 Mo) GPU plugged to a PC with a Dual Core II 2,4 GHz CPU.
These performances greatly depend on the number of markers. They are presented here for a 1100 x 1100 resolution of phyxels.

In the case of the real-time chain (section 5) one should take into account 3 possible blocking parameters:

- The speed of the upstream simulation.
- The speed of the communication between this simulation and the computer used to visualize this simulation. In our case it is a 100 Mbits/s Ethernet communication that uses UDP protocol.
- The simulation speed of the DPCM.

The DPCM is used for visualization aims, so we can accept a latency provided it is non visible. But a coherent rate with the upstream chain has to be respected. Problems of latency (due to the Ethernet communication as well as the DPCM) occur for a number of markers greater than 300. This is quite acceptable for a number of real-time applications like the one proposed at section 8.1. Indeed, if the number of markers is too high, the usefulness of the DPCM as a coating and visualization method for particle models progressively disappears because particles occupy the whole space. Then, the DPCM does not add a useful spatial extension anymore.

Table 1 gives the performances obtained when an upstream computer directly send markers to the DPCM through the network. To respect a certain maximal limit in the frequency of the display, frames are ignored when an overloading occurs.

.

Number of markers	1	20	50	100	300
Frequency	690	220	104	55	23

Table 1: *Performances for a 1100x1100 DPCM with data coming from Ethernet.*

7. Interface

Our DPCM implantation has been made usable for end-users thanks to an interface developed with QT 4.

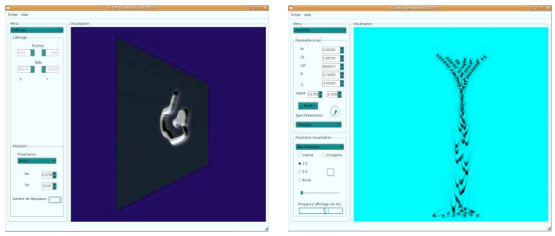

Figure 9: *User interface. On the left: the calibration mode. On the right: the simulation mode.*

This interface is divided into two different modes: a calibration mode and a simulation mode (see figure 9).

7.1. The calibration mode

This mode allows calibrating the DPCM before the simulation. The user can change its size and its position in accordance with positions of the markers in order to visualize the part of the space he wants. In this mode, the markers are represented by spheres whose ray is equal to the distance threshold of the viscous-elastic interaction with the DPCM. This threshold does not define a spherical shape that will be printed on the DPCM except when the stiffness is close to infinity. Nevertheless this representation gives an idea of the influence area of the markers. This information is very useful when the user defines the spatial parameters of the DPCM.

7.2. The simulation mode

Once the spatial properties of the DPCM are given, the user starts the physical modeling of the engraving process. To do so, he chooses: (1) the interactions for the mobile mass and the fixed mass of each phyxel; (2) the interactions between the phyxels and the topology for these interactions (for instance, interaction between 8 neighbors, 4 neighbors, or more) The interactions are chosen from a library of various interactions (viscous, elastic, plastic, etc.). Then, he tunes the physical parameters of the interactions in real-time to obtain the expected behavior for the DPCM.

A library of behavior examples is offered to the non-expert user of the DPCM that covers a large panel of physical behaviors (propagation, plasticity, etc.). Some examples of behaviors are presented in section 8.

The user finally chooses a type of visual rendering of the DPCM. He notably defines a color map that will correspond to a scale of deformations of the phyxels.

8. Results

This section shows some results we obtained. Results are classified by types of behavior.

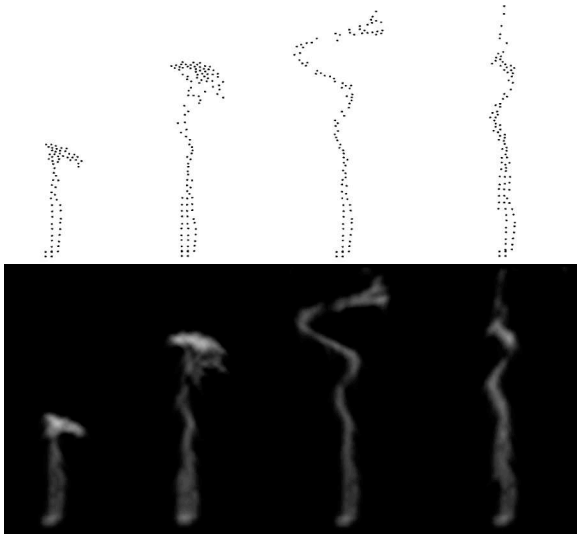

Figure 10: *Real-time Dynamic Coating of a mass-interaction model of smoke thanks to the DPCM method. Top: The upstream model without coating. Bottom: the same model coated thanks to the DPCM.*

8.1. Adding physically based warping and fraying

In this example, we test the maximal real-time capacities of our application. The model we choose to visualize is the same than in [HL02]. It is simulated on a quadriprocessor Opteron 248 with 3 GB of RAM linked by an Ethernet connection to the PC that hosts the GPU on which the DPCM has been implemented. The smoke is simulated by 300 masses interacting through 300^2 viscous interactions with spatial thresholds. So we are in a limit case of real-time physically based simulation. The final visualization frequency is around 23 Hz. The aim of the coating is to add in real-time physically consistent micro-effects such as warping, fraying, micro-turbulences that do not belong to the upstream mass-interaction model of smoke. Figure 10 shows the results.

8.2. Propagations effects

In this example, the DPCM is used to add propagation effects to the dynamic of the upstream model.

In a first example, the DPCM is used to render propagations on a lake surface. The physical parameters of the interaction have been tuned to obtain a fluid propagation (low viscosity and high stiffness). Markers are set on the water line (4 markers for each boat). We apply a background texture to simulate refraction. Reflection is achieved by making a first pass. This pass projects the

scene and reverses it on the water surface. The final rendering pass receives this texture and reads it to simulate the reflect deformation in accordance with the DPCM deformations (see figure 4). Figure 11 shows three boats on this lake. Waves are automatically mixed thanks to the physical model of the DPCM and bounce against the coast.

Figure 11: *Furrows caused by 3 boats on a lake*

The choice of topology of the interactions between neighboring phyxels is decisive in the propagation of behaviors exhibited by the DPCM.

In Figure 12, the effect is the same if we choose to connect the four nearest neighbors or if we choose to connect the four diagonal neighbors. But the propagation speed is higher in the second case.

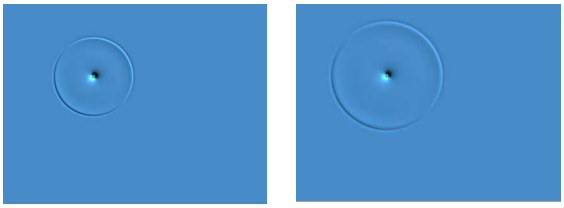

Figure 12: *Difference between two configurations in neighboring interactions for the same initial conditions. On the left: x-neighboring and y-neighboring phyxels interact. On the right: diagonal phyxels interact.*

Interesting chaotic behavior can be obtained by choosing more complex interactions topology. The behavior exhibited by figure 13 is a kind of dynamic Oil painting obtained by randomly deciding if two neighboring phyxels interact or not.

Figure 13: *A Real time "dynamic Oil painting" behavior obtained by a stochastic topology of the neighboring interactions.*

8.3. Plastic behaviors

Non-linear interactions are used to create irreversible behaviors that allow obtaining permanent engraved tracks. We use a plastic interaction as proposed by [CLH96]. This interaction exhibits a hysteretic behavior by modifying its rest length when an elastic threshold is reached. Phyxels do not interact with their neighbors because propagation effects are not necessary.

In a first example shown in (figure 14), we modeled tracks of vehicles engraved in a loose soil.

Figure 14: *Vehicles on a loose soil.*

In a second example shown in (figure 15), the plastic interaction is used to interactively generate sandy terrain outlines. In this case, the user digs the soil in real time by means of a gesture device such as mouse of force feedback device.

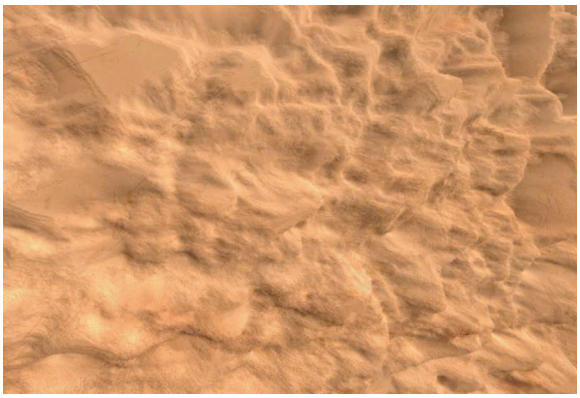

Figure 13: *Sandy terrain.*

8.4. Plastic behaviors mixed with propagation

By mixing plastic behaviors and propagation effects, markers create waves that permanently mark the DCPM. Shapes, that can be more or less permanent according to the values of the physical parameters of the DPCM, are then emerging. One can use the metaphor of the wind or of the earthquake that model a terrain outline.

Figure 14 shows a terrain profile generated in real time by the user through a gesture device.

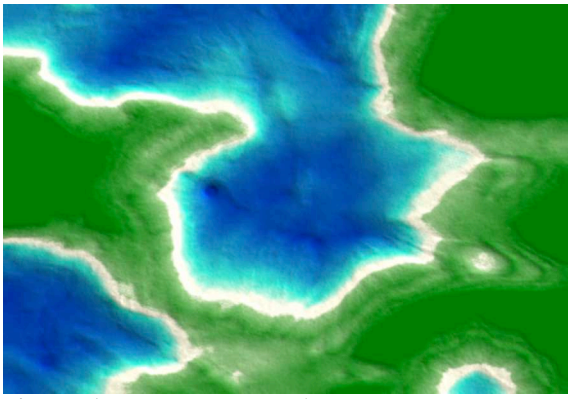

Figure 14: *Interactive Real time terrain generation obtained by propagation and plastic behavior.*

In the last two examples (figure 15 and 16), a similar model is used to create sedimentary materials. An upstream mass-interaction model controlled by a gesture device engraves its tracks in this material to render a kind of fossilization-like effect.

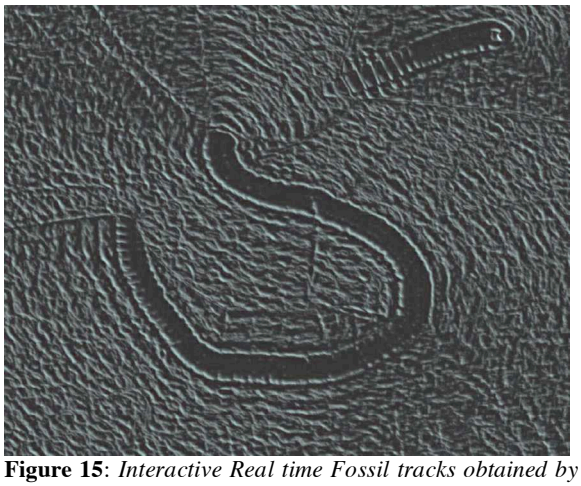

Figure 15: *Interactive Real time Fossil tracks obtained by means of user action.*

Figure 16: *Interactive Real time Fossil tracks in a harder material*

9. Conclusion and perspectives

This paper proposed a real-time implementation of the DPCM method on GPU architecture. Real-time limits are reached with 1100x1100 phyxels and about 300 markers or more. Thanks to these performances, it is possible to cover a wide number of real-time coating processes in virtual reality interactive scenes.

We have presented a new VR platform including a complete real-time simulation chain with a haptic force feedback device, a mass-interaction model that runs at the gesture frequency (more than 1000 Hz) and a complex coating process for complex phenomenon such as smokes, sand, water effects.

Optimizations of the DPCM implementation can still be foreseen. One of the main drawbacks are the dependence of

the computation time to the number of the upstream markers. A new implementation is in progress to decorrelate the performances of the DPCM and the number of markers.

Nevertheless, when dealing with implementation on hardware architecture that is not initially designed for the desired goal, such as the use of GPU for physically based models, we have to take care of what could seem to be an optimization. For instance, we have made geometrical tests on markers and phyxels to prevent the assessment of useless interactions. When a marker is to far from a phyxels, the calculation of the square root could seem useless. One can imagine not calculating this square root by using a kind of octree structure. But this kind of optimization has dramatically reduced the performances of the method. A test on a condition was more time consuming than a lot of parallel square root calculations.

The 3D generalization of this 2D DPCM is now started. Nevertheless, a lot of physically based visual rendering can already be obtained with a 2D DPCM.

9.1. Acknowledgements

We would like to thank Mathias Paulin of IRIT (Toulouse, France) for its fruitful advises on GP/GPU. Thank you also to Julien Castet (ICA-INPG) and Jean-Loup Florens (ACROE) for their help in the real time implementation of the model of smoke and the Ethernet communication protocol.

References

[BBB*97] BLOOMENTHAL J., BAJAJ C., BLINN J., CANI-GASCUEL M.P., ROCKWOOD A., WYVILL B., WYVILL G.. In Introduction to implicit surfaces. Morgan Kaufmann, 1997.

[Ben01] BENDAZZI G. Alexeieff: Itinerary of a Master. Dreamland, Paris, France. ISBN 2-910027-75-9. 2001.

[Bli78] BLINN J. F., James F. Simulation of Wrinkled Surfaces, Computer Graphics, Vol. 12 (3), pp. 286-292 SIGGRAPH-ACM (August 1978)

[Bli82] BLINN J.F.. A Generalization of algebraic surface drawing. ACM Transaction on Graphics, 1(3), pp 235-256, 1982.

[CL06] CHANG L., LIU D.. Deformable Object Simulation in Virtual Environment. In Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications ISBN:1-59593-324-7, pp 327-330.

[ECL06] EVRARD M., CASTAGNE N., LUCIANI L. MIMESIS: Interactive Interface for Mass-Interaction Modeling. In Proceedings of CASA 2006, Geneva, July 2006, Nadia Magnenat-Thalmann & al. editors. pp177-186.

[FR92] FARIA LOPES P., RUI GOMES M.. A Computer Model For Pinscreen Simulation: A New Animation Paradigm. Computer Graphics Forum 11 (1), 31–42 (1992).

[CLH96] CHANCLOU B., LUCIANI A., HABIBI A. Physical Models of Loose Soils Dynamically Marked by a Moving Object. In Proceedings of the Computer Animation. June 1996, Geneva, Switzerland, pp 27-35.

[GW05] GEORGII J., WESTERMANN R. Mass-Spring Systems on the GPU. In Simulation Practice and Theory 2005, Elsevier Science July 2005

[HL02] HABIBI A., LUCIANI A.. Dynamic Particle Coating. In IEEE Transactions on Visualisation and Computer Graphics, vol. 8, no. 4, Octobre-Decembre 2002, pp 383-394.

[KKK*05] KRÜGER J., KIPFER P., KONDRATIEVA P., WESTERMANN R. A Particle System for Interactive Visualisation of 3D Flows. In IEEE transactions on visualization and computer graphics, ISSN: 1077-26262005 Nov-Dec,11(6):744-56

[KLR04] KOLB A., LATTA L., REZK-SALAMA C. Hardware-based Simulation and Collision Detection for large Particle Systems. In Graphics Hardware 2004. August 2004, pp 123-132.

[LEC*06] LUCIANI A., EVRARD M., CASTAGNÉ N., COUROUSSÉ D., FLORENS J.-L., CADOZ C.. A basic gesture and motion format for virtual reality multisensory applications. Proceedings of the GRAPP conference, February 2006.

[LJF*91] LUCIANI A., JIMENEZ S., FLORENS J.-L., CADOZ C., RAOULT O. Computational physics: a modeler simulator for animated physical objects. Proceedings of the European Computer Graphics Conference and Exhibition. Eurographics'91, pp 425-436, Vienna, Austria, Elsevier Ed, Sep. 1991.

[MKN*04] MÜLLER M., KEISER R., NEALEN A., PAULY M., GROSS M., ALEXA M. Point Based Animation of Elastic, Plastic and Melting Objects. In Proceedings of ACM SIGGRAPH Symposium on Computer Animation (SCA), pp.141-151, 2004.

[OLG*05] OWENS J.D., LUEBKE D., GOVINDARAJU N., HARRIS M., KRÜGER J., LEFOHN A., PURCELL T. J.. A Survey of General-Purpose Computation on Graphics Hardware. In Proceedings of Eurographics 2005, State of the Art Reports. Dublin, Ireland, 29 August - 2 September, pp 21-51.