D4MD – Deformation system for a vehicle simulation game

Tiago Rodrigues
ADETTI/ISCTE
Edifício ISCTE
1600-082 Lisboa, Portugal
+351964383671
tiago_rodrigues@users.sf.net

Rui Pires
ADETTI/ISCTE
Edifício ISCTE
1600-082 Lisboa, Portugal
+351964372164
rui_pires@users.sf.net

José Miguel Salles Dias
ADETTI/ISCTE
Edifício ISCTE
1600-082 Lisboa, Portugal
+351217826480
miguel.dias@adetti.iscte.pt

ABSTRACT
This paper presents a hybrid geometrical-physical, plastic deformation technique applicable for solids, in the context of a car simulation game. This technique doesn't aim to be mechanically correct but to produce visual appealing and realistic-looking results.

Categories and Subject Descriptors
I.3.5 [COMPUTER GRAPHICS]: Computational Geometry and Object Modeling – *Geometric algorithms, physically based modeling, splines*

General Terms
Algorithms.

Keywords
Deformable Objects, Dynamics, Physically-based Simulation, Collision Detection and Response, FFD Geometric Deformation, Game development.

1. INTRODUCTION
D4MD (pronounced: dĭ-fôrmd') stands for the deformation system presented in this paper. This system aims to produce visual appealing plastic deformations in 3D game objects as the result of instantaneous collisions in the context of a larger in-house on-going project aiming at developing a racing car game. It is our understanding, that due to its large contribution to the games graphical realism, physically-based geometry deformation is an essential characteristic of recent driving simulation games.
The paper is organized in 8 sections. Section 2 presents an overview of current geometry deformation techniques applied in car simulation games. Section 3 summarizes the most common geometry deformation algorithms available in the literature and draws some conclusions about their applicability in our project. In section 4 we present some of the main goals of our system and detail the requirements to achieve them. Section 5 is dedicated to the design of the D4MD deformation technique. In section 6, we identify and describe the main components of our deformation system. Section 7 discusses the main advantages of our approach. Finally, section 8 presents some conclusions and suggestions for future work.

2. DEFORMATION IN VIDEO GAMES
The enormous demand for realism that we witness today from the game industry, pushes hardware evolution that enables the implementation of more complex and consequently more visually realistic algorithms. Nowadays, game users are becoming more and more demanding and characteristics that used to be minor details, such as realistic (dynamic elasto-plastic) volumetric bodies deformation upon collision, have become mandatory.
One of the first game titles to include car geometry deformation was Destruction Derby™ (1995) [6]. Although the deformation system was rather simple and based on pre-computed deformations, the game caught many players' attention and was a big success. At about the same time, the first title of the very controversial series, Carmageddon™ [2] was released. The deformation system employed was more advanced than its peers and had some dynamic deformation. Other examples of posterior success titles, which included deformation, were Driver™ [7] and GTA3™ [8]. Many games followed perfecting the base deformation concept, by using better physically-based simulation algorithms and more geometric detail.
With few exceptions, like the popular NFS™ series [13], almost all the car simulation games edited today, implement some deformation technique. Some of the more recent examples from 2004 are Colin Mcrae Rally™ 5 [3], Xpand Rally™ [20], and the notable Toca Race Driver™ 2 [19], which has implemented a very realistic deformation system. As we can see, although we have many examples of games that have implement deformation, published information about the techniques employed is scarce, possibly due to high cost of R&D in this type of game technology. With the preview of the power offered by the next generation consoles like the PS3™ [16], we will certainly witness outstanding advances on this field, with the possibility of implementation of the most complex and realistic techniques[i].

3. BACKGROUND IN DEFORMATION TECHNIQUES
The range of deformation techniques presently used in games is quite vast, going from pre-computed deformations to really advanced and realistic physically-based algorithms. One of the most common and simple techniques used for this purpose is the pure visco-elastic mass-spring model, introduced by Demetri Terzopoulos in the field of 3D Computer Graphics, as early as 1987. This technique, now a classic one, models deformable objects through a network of "structural" springs connected to

masses, which try to simulate the stiffness properties of the materials. The conventional mass-spring models are generally used to simulate bi-dimensional objects due to the nature of the springs employed, which, in the simpler case, only support planar contractions and expansions (tensile deformations) along its axis. Modeling complex 3D objects generally requires, besides the “structural” springs, the addition of special interaction elements to enable the model to withstand torsion and bend forces. Because of this, it is generally necessary to have prior knowledge of the object's structure to build a stable network. Some models that try to overcome these limitations were recently presented. One such model, presented in Eurographics 2003 [10], describes an a particle system connected by interaction elements, that integrates the characteristics of the tensile and bending types of springs and simulates, rather accurately, the discrete tensile, bending and torsion mechanics of solids. However, the use of mass-spring models to simulate elasto-plastic deformation of complex 3D models generally requires continuous collision detection with a very small granularity level that is not feasible in most of the available physics SDKs, so this technique, although implemented, was later discarded by us.

Another known classic technique, brought from Structural Mechanics, that has been emerging in the latest games, is the Finite Element Method [18]. Its main characteristic is the quality and realism of the results obtained, due to its solid mathematical and classical linear and non-linear elasticity physical basis. The main concept of this technique is to divide objects in small building blocks with well defined material characteristics, where the elasticity theory of the continuum is still valid and physics are more easily simulated. Until a few years ago the use of this technique was prohibitive in real time applications, but with simplified techniques such as the one presented in SIGGRAPH 2002 [11], its use is becoming a viable option, although the memory and processing requirements can be overwhelming for most game and real-time applications.

As the last main category of techniques that were analyzed, we have the geometrically-based ones (as opposed to the previous ones, which are physically-based) that are based on space deformation instead of direct geometry deformation. One of the main representatives of these techniques is the FFD (Free Form Deformation) [4], brought to the 3D computer Graphics world by Sabine Coquillart. Today, it is mainly used in modeling applications as a deformation tool. The first step in FFD is to wrap the geometry to be deformed in a control point lattice that parameterizes a space deformation function. The controls have a weighted influence on the various space regions, and the alteration of their position deforms the space, not the geometry directly. The geometry is deformed by mapping it to that normalized space. The fact that this technique is independent of the geometry can be a limitation, because it can become very hard to apply a certain deformation to an arbitrary geometry. To overcome this limitation, there are various techniques such as the EFFD (extended FFD) [5], or FFD with direct manipulation [9]. In the EFFD, before the manipulation of the controls, these are approximated to the geometry, making the process of deforming an object more intuitive. However, this technique is more suited to interactive object modeling. In direct manipulation the deformations can be directly applied to the geometry and the controls get reconfigured automatically to produce the desired deformations.

4. GOALS AND REQUIREMENTS OF D4MD

Our main objective is to develop a deformation algorithm that realistically simulates the plastic instantaneous deformation of metal-like objects, such as the bodywork of a car. The emphasis, as in most games, is not on the precise mechanical simulation of the deformations but in producing visual appealing and realistic-looking results at a low performance cost. Given this requirement, our technique should address, to some extent, the physically-based simulation of deformable objects. Having in mind that availability of third parties rigid-body dynamics SDK for game development (although lacking deformation support), we concluded, that, in the context of our larger project of car game design, the main piece of external software we would be interacting with, would be the physics SDK. This element is responsible for the dynamic simulation of rigid bodies and collision detection. Although we have witnessed a considerable evolution in physics engines in recent years, both in speed and in features, the choice of a deformation technique to use in real time is still quite limited by the features and possibilities of the current engines. Most of the existing SDKs still lack needed features, like continuous collision detection, robust mesh-to-mesh simulation and the possibility to change the geometry of the objects with a low performance hit. Since these limitations are common to most of the available SDKs, we had to choose a physics SDK mostly based on performance and robustness. After making some tests with two SDKs (ODE [15] and NOVODEX [14]), which are more accessible to the academic community, regarding licensing schemes, NOVODEX presented itself as the winning candidate, mainly because of its astonishing performance when it comes to primitive shape physical-based simulation.

5. DESIGNING A DEFORMATION TECHNIQUE

Bearing in mind some of the main restrictions imposed by the chosen physics simulation SDK, specially, the lack of support to deformable model simulation, it seamed to us that we had to design a hybrid geometrical-physical deformation technique, which should have two algorithmic steps: a first, physical-based rigid-body dynamics step and a second, geometrically-based deformation one. This last step should evaluate the geometrical deformation whenever two objects, being simulated by the rigid-body dynamic physics simulation SDK, collide. A good candidate technique capable of addressing this requirement was the FFD. The geometry independence of the FFD enables it to keep applying deformation to a space ignoring the distribution of the geometry, allowing us to get good results even when approximating complex geometry by simple basic shapes such as boxes. After we had chosen the FFD technique, as the basis of the geometrical deformation step of our algorithm, we still had to choose the parameters that characterize it, such as the deformation function and the way displacements are applied to the control points. The deformation function chosen was a tri-variant cubic B-Spline tensor product. We decided in favor of B-Splines over other options, such as Bezier curves, because of the well known B-Spline's local control properties and their inherent independence between polynomial order and number of controls. Another advantage of the B-Spline over piecewise Bezier curves, is the guarantee of automatic parametric C^2 continuity between segments (in the Bezier case this continuity has to be explicitly maintained across segments). We use uniform cubic B-Splines because they result in simplifications of the control weight equations (shown in equation 1), and consequently in a performance increase. With cubic B-Splines, each surface point is influence by 4 controls.

$$\begin{aligned} weight_0 &= \frac{1}{6}\times\left(1-3\times t+3\times t^2-t^3\right)\\ weight_1 &= \frac{1}{6}\times\left(3\times t^3-6\times t^2+4\right)\\ weight_2 &= \frac{1}{6}\times\left(1+3\times t+3\times t^2-3\times t^3\right)\\ weight_3 &= \frac{1}{6}\times t^3 \end{aligned} \qquad (1)$$

One of the most common problems faced when using B-Splines for FFD, is the fact that they don't interpolate their end points. There are various solutions for this, such as the use of phantom control points [1] or the use of terminal knots with higher multiplicity. Again our approach was to try and simplify our algorithm, so we have only defined the deformation volume where the B-Spline exists. This solution is more efficient since there is no need to interpolate the terminal controls and also allowed us to use uniform B-splines. Manipulation of control points in order to achieve the required deformations of the geometry, and in particular in the case where we have multiple simultaneous deformations, can be quite challenging. The technique we have selected to overcome this limitation was direct manipulation as described by Hughes and Kaufman in 1992 [9]. In our case this technique seemed to be the most adequate, since the deformation process is automatic and there is no user interaction.

6. DEFORMATION SYSTEM

In this section we present a detailed description of our deformation system's components and algorithms.

6.1 Components

The central pieces of the deformation system are the "deformable object", the "geometry deformer", and the "deformation". The "deformable object" includes the properties that condition a deformation, such as the minimum deformation accepted, the maximum instantaneous deformation and the deformation factor of the object's material. A "deformable object" can be constituted by various "geometry deformers" that correspond to parallelepiped deformation volumes associated with part (or all) of the object's geometry. The "geometry deformers" are parameterized by their position, dimensions and control point distribution. As we said before, the deformation function of these volumes is a tri-variant cubic B-Spline tensor product. A deformation corresponds to a group of points from the same deformable object each associated with a displacement. The direction of the displacement must be similar for points of the same collision.

6.2 Deformation Algorithm

The deformation process starts when two objects, in simulation by the rigid-body dynamic physics simulation SDK, collide. Current simulation techniques always allow some inter-penetration between colliding objects. Due to this fact, most of the time, the contact points returned by the Physics SDK are not on the surface of both objects. This way, the first step of the deformation algorithm is to find the surface collision points. To accomplish this we use the penetration depth value returned by the engine. Having the surface contact points we find the resulting collision impulse value in each of them. The impulse formula used is the one described by Baraff in SIGGRAPH 1992 [5]. The impulse is a good starting point to find the deformation value on an object, because it contemplates several of its mechanical properties such as velocity, mass and inertia tensor. As we are working with instantaneous deformations, we have defined a deformation factor of each object and have applied it to the impulse value, which is translated into the deformation value to be applied to the surface contact point, much like applying a force to a spring to obtain a displacement. In order to maintain the deformations applied to each object under control we check its value against the object's maximum and minimum admitted deformation values for that zone. Since the physics SDK doesn't give all contacts of a collision simultaneously (a well known restriction), we have to organize contact points into collisions that group similar contact points for a deformable object. A collision lasts for a determined configurable time after which it is dispatched to the objects that were involved. Upon receiving a collision, a deformable object passes this information to the "geometry deformer" responsible for the area affected. If it is required, both the contact points and the displacements values are linearly interpolated. This step insures that a collision that yields contacts that are too distant, such as the resulting from the vertexes of a face of a big box, results in a correct deformation of the body that suffers the collision. The next procedure is to perform the FFD of each of the deformation volumes. This is done by reconfiguring the position of the volume's control points, so that the correct deformations get translated in to the space. (Equation 2 shows the FFD transformation) The position of each of the points in matrix *P* is the result of a weighted sum of controls points in matrix *C*.

$$P = B\times C \Leftrightarrow \begin{bmatrix} p_0 \\ \vdots \\ p_{p-1} \end{bmatrix} = \begin{bmatrix} b_{00} & \cdots & b_{0n-1} \\ \vdots & \ddots & \vdots \\ b_{p-10} & \dots & b_{p-1n-1} \end{bmatrix} \times \begin{bmatrix} c_0 \\ \vdots \\ c_{n-1} \end{bmatrix} \qquad (2)$$

Using the direct manipulation technique [15], matrix *P* is filled by the contact points, and matrix *C* by the control points that affect them. *B* is calculated using the uniform cubic B-Splines equations defined in equation 1. Obtaining the pseudo-inverse of matrix *B* and multiplying it by the displacements on the contact points we obtain the displacements we should apply to each of the control points involved (shown in equation 3). Each control that gets displaced gets its modification flag marked.

$$\nabla C = pseudoB^{-1}\times\nabla P \qquad (3)$$

As the last step in the algorithm, we need to bring the affected geometry up to date with the deformations imposed on the volume's space. Using cubic B-Splines, finding which of the geometry vertexes need to be transformed is straightforward since we know that an altered control point affects 4 segments and that a point is influenced by 4 control points. To get the deformed vertexes we follow the process presented in equation 2. After the deformation, the volume's limits are recomputed and the control points redistributed over them. After this processing steps, the object is ready to sustain other deformations.

7. DISCUSSION

In the paper, we have presented a hybrid geometrical-physical deformable model, able to support plastic collision detection and response between volumetric objects, which evolve in a physically-based world, such as a racing game environment. Implementing this technique have showed us that visual appealing results can be obtained in real-time and at a low performance cost, as we can see in figure 2, where the deformation procedure completed under 0.016s and a total of 38 contacts resulted in 4984 of the total of 9580 triangles being updated (in a Pentium IV PC with a GeForce 6600). Since the deformation is done with FFD manipulation it's quite easy to control the complexity of the deformation structures, by changing the control point distribution, effectively managing the performance. The flexible way of associating deformable zones to objects also have enables us to build complex deformable shapes allowing us to use the physics SDKs primitive shapes efficiently. Most of these options wouldn't be possible with some of the other techniques mentioned, such as mass-spring, where the deformation structures are directly implied by the geometry. FEM looks like the most promising technique in this area, having as its only barrier, to real time performance with high processing requirements. However, the recent launch of PhysX™ [17] the world's first Physics Processing Unit (PPU), promises fast and affordable finite element analysis in a near future.

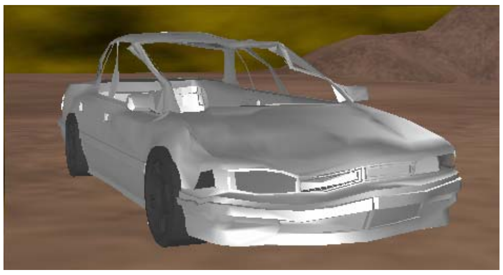

Figure 2. Deformed car after a violent impact

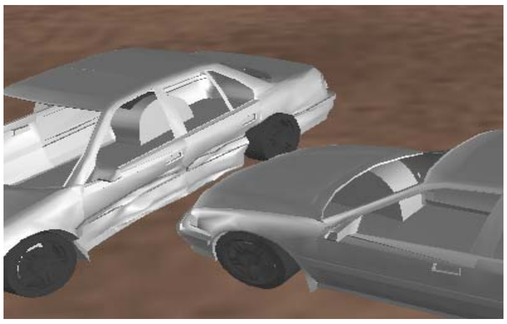

Figure 3. Deformed car after a lateral impact

8. CONCLUSIONS AND FUTURE WORK

This paper presented part of the work of an ongoing project whose objective is to produce a car simulation game. The main characteristics of this larger project are realistic simulation of the mechanical aspects of the vehicles and realistic geometric deformation of objects. We have presented a hybrid geometrical-physical deformable model, able to support plastic collision detection and response with deformation, between volumetric objects, which evolve in a physically-based world. In today's games, a car model can have as much as 50.000 polygons, and guaranteeing that our algorithm scales well with the models complexity is also in our plans. We are also implementing a reasonably accurate vehicle simulation, featuring engine torque curves, drag resistance, transmission efficiency, gear ratios, wheel grip and slip, among others. This implementation will mostly follow the guidelines described by Marco Monster [12]. The work being done in the simulation of the mechanical characteristics of a car, coupled with our deformable geometry algorithm, will give us the definite test of the robustness of the algorithm in a car game environment.

9. REFERENCES

[1] Bartels, Richard H., Beatty, John C., Barsky, Brian A. *An Introduction to Splines for use in Computer Graphics and Geometric Modeling*. Morgan Kaufmann, 1987.

[2] Carmageddon, Stainless Software (1997), www.sci.co.uk

[3] Colin Mcrae Rally 2005, Codemasters (2004), www.codemasters.co.uk/colinmcrae2005

[4] Coquillart, Sabine. "Extended Free-Form Deformation: A Sculpting Tool for 3D Geometric Modeling". Proceedings of ACM SIGGRAPH , In Computer Graphics,1990.

[5] D. Baraff, *Dynamic Simulation of Non-Penetrating Rigid Bodies*, (Ph. D thesis), Technical Report 92-1275, Computer Science Department, Cornell University, 1992.

[6] Destruction Derby, Reflections Interactive (1995).

[7] Driver: You Are the Wheelman, Reflections Interactive (1999).

[8] GTA3, DMA Design (2001), www.rockstargames.com/grandtheftauto3/

[9] Hughes, J., Kaufman, H. Direct manipulation of free-form deformations, Cambridge Research Lab, DEC Corporation (1992).

[10] Il-Kwon Jeong, Inho Lee, "A New 3D Spring for Deformable Object Animation", Proceedings of Eurographics 2003.

[11] M. Müller, J. Dorsey, L. McMillan, R.Jagnow and B.Cutler. "Stable real-time deformations". Proceedings of ACM SIGGRAPH Symposium on Computer Animation, pp 49-54, 2002.

[12] Marco Monster, Car Physics for Games (1993), http://home.planet.nl/~monstrous

[13] Need For Speed series, EA Games (1994), www.eagames.com

[14] NovodeX, www.novodex.com

[15] ODE, ode.org

[16] Playstation, www.playstation.com

[17] PhysX, www.ageia.com

[18] Reddy, J.N., *An introduction to the finite element method*, McGraw-Hill, New York, 1993.

[19] Toca Race Driver 2, Codemasters (2004), www.codemasters.co.uk/tocaracedriver2/

[20] Xpand Rally, Techland (2004), www.xpandrally.com