GPU Destruction: Real-Time Procedural Demolition of Virtual Environments

Derek John Morris[1] and Eike Falk Anderson[1]

[1]Interactive Worlds Applied Research Group, Coventry University, United Kingdom

Abstract

We introduce a method for the real-time simulation of destructible materials for use within videogame environments. Our system combines a number of existing techniques in a Graphics Processing Unit (GPU) based implementation that employs procedural geometry generation to reduce content creation times while retaining artist control.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation; I.6.8 [Simulation and Modeling]: Types of Simulation—Gaming

1. Introduction

In recent years, videogames have become increasingly complex, aiming to immerse the player in virtual game worlds that employ 3D animated graphics to provide players with the illusion of realism. A major contributing factor to this end has been a steep rise in the quality of real-time computer graphics, fuelled by dramatic advances in computer graphics hardware combined with the evolution of algorithms and methods that more realistically model the real world.

One particular area that contributes significantly to the perception of a realistic virtual environment is the behaviour of various materials when they are broken apart or destroyed.

'GPU Destruction' is a real-time simulator for destructible materials that can be used within a videogame or similar real-time graphics application. To achieve this, most of the simulation processing is performed on the GPU in order to make most efficient use of system resources (see Figure 1).

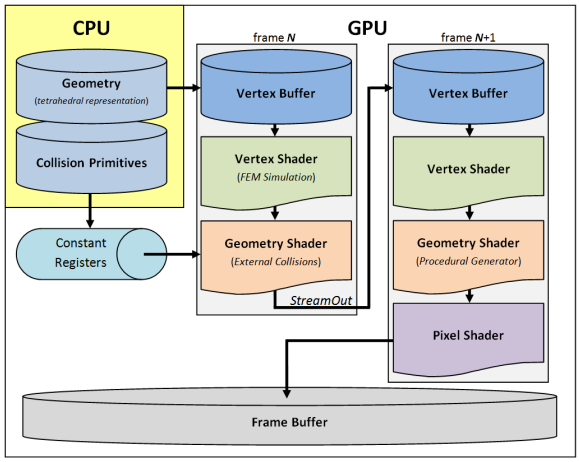

Figure 1: *GPU Destruction method.*

2. Background

The majority of investigations into the simulation of deformable bodies have aimed to obtain a high level of accuracy with the associated cost of high system resources. In the domain of real-time computer animation, such as found in videogames, however, some of these accuracies can be traded for higher operating speed as long as the perceived visual representation remains acceptable.

Significant progress in the computer graphics area was made by O'Brien and Hodgins who introduced methods for modelling and animating first 'brittle fracture' [OH99] using a tetrahedral finite-element method, which was then extended to also model 'ductile fracture' [OBH02].

Parker and O'Brien [PO09] present a fully working real-time system that has been implemented into a published videogame title, which has a tetrahedral finite element method at its core that uses strain and stress tensors to model the internal forces within a material. A simplified version of the techniques by O'Brien and Hodgins [OH99] is used to propagate cracks throughout a material at points where stresses exceed a certain threshold. In order to mask the low resolution of the underlying tetrahedral mesh in destroyed areas, pre-fabricated sections of geometry ('splinters') that more accurately represent the fractured material are inserted at the edges. The system runs on multiple Central Processing Unit (CPU) cores in order to optimize the operating speed.

A different approach to representing destroyed sections of a material has been demonstrated by Scheepers and Whittock [SW06]. In their method the material is created and stored in pre-shattered form as a collection of voronoi tiles. To achieve the splintering effect, a spline is carved along the required shatter line and the voronoi tiles adjacent to the line are lifted out of position.

3. GPU Destruction

Our method is based on the work presented by Parker and O'Brien [PO09], which is used in combination with other methods (see Figure 1). At the heart of our system lies the migration of the majority of processing from the original concept onto the GPU, making optimal use of the graphics hardware, in particular the geometry shader and StreamOut mechanism [Mic09] to output the results of the simulation timestep into a vertex buffer for further processing (implemented using DirectX and HLSL on DX10.1 class hardware). An element stiffness matrix per tetrahedra is derived within the vertex shader and output into a custom vertex format. These stiffness matrices are later combined into an overall stiffness matrix that fully describes the material domain. This is combined with a fast method of processing collisions with external objects [Zin08] that allows simple primitive types to be transferred to the GPU via the hardware's constant registers. Rather than requiring artists to provide the 'splinter' edges of the damaged materials, these are procedurally generated within the geometry shader. The method we are currently evaluating for this employs a 3D voronoi diagram that is generated based on object material specific parameters, effectively translating the 2D method by Scheepers and Whittock [SW06] into the third dimension. This significantly reduces content creation times but retains artist control, which is particularly important for videogames.

Our system provides:

- A real-time simulation of deformable and destructible materials suitable for implementation within a videogame environment.
- Full use of modern GPU features utilising implicit multi-threading and consequently freeing up CPU resources for use in other game components.
- Procedural geometry creation techniques that save valuable content creation time.

4. Summary and Future Work

The simulator presented here is work in progress, which promises to improve the efficiency of previous implementations both in terms of hardware acceleration and the cost to artists in terms of content creation time.

For the purposes of this work we have used objects that that have simple geometrical shapes, such as fences and walls that have a straightforward tetrahedral representation. For integration into a production environment beyond the prototype stage, rather than setting tetrahedra and all associated material properties explicitly within the application, it would be sensible to simplify usage of the system by providing a separate tool to create and tweak tetrahedra for different types of objects and to allow modification of material parameters in real-time.

With future advances in GPU technology on the horizon, it should not only be possible to simulate different material types that deform and fracture in a visually plausible manner under real-time conditions with minimal intervention by the CPU, but many of the used techniques could be implemented using GPGPU methods and expressed in a more general form with the intention of moving towards having a fully GPU-based physical simulation of all aspects of the game world.

References

[Mic09] MICROSOFT: DirectX 10 developer documentation: ParticlesGS sample. MSDN Library - available from: http://msdn.microsoft.com/en-us/library/ee416421(VS.85).aspx, 2009. 2

[OBH02] O'BRIEN J. F., BARGTEIL A. W., HODGINS J. K.: Graphical modeling and animation of ductile fracture. *ACM Trans. Graph. 21*, 3 (2002), 291–294. 1

[OH99] O'BRIEN J. F., HODGINS J. K.: Graphical modeling and animation of brittle fracture. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques* (1999), pp. 137–146. 1, 2

[PO09] PARKER E. G., O'BRIEN J. F.: Real-time deformation and fracture in a game environment. In *SCA '09: Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2009), pp. 165–175. 2

[SW06] SCHEEPERS F., WHITTOCK A.: The wrecked road in cars - or how to damage perfectly good geometry. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Sketches* (2006), p. 97. 2

[Zin08] ZINK J.: Dynamic particle systems. In *Programming Vertex, Geometry, and Pixel Shaders*. GameDev.Net, 2008. Available on-line at http://wiki.gamedev.net/index.php/D3DBook:Table_Of_Contents. 2