Mike Riley

Dr. Micah Taylor

CSSE451-01

19 April 2019

Real-Time Deformation and Fracture in a Game Environment - Summary

A problem in computer graphics today is being able to display deformation (destruction) and fracture in real-time, like in a gaming environment. This needs to be done in a quick and efficient manner so that the game does not lag and experience drops in framerate. This paper tackles that issue, and attempts to explain the methodology behind the practice. This has actually been implemented, and if you, the reader, has played *Star Wars: The Force Unleashed*, you have used the very tool described in this paper.

They use the Finite Element Method to model simulated materials. Each element is defined by four nodes, and these nodes are $\mathbf{u}_1$, $\mathbf{u}_2$, $\mathbf{u}_3$, and $\mathbf{u}_4$. Together, these nodes make up a 3x3 matrix, $\mathbf{D}_\mathbf{u}$ with columns $\mathbf{u}_2 - \mathbf{u}_1$, $\mathbf{u}_3 - \mathbf{u}_1$, and $\mathbf{u}_4 - \mathbf{u}_1$. This matrix can be used to compute the deformation gradient $\boldsymbol{F}$. A polar decomposition is then performed on the gradient, to get $\boldsymbol{F} = \boldsymbol{QA}$ where $\boldsymbol{Q}$ is orthonormal and $\boldsymbol{A}$ is symmetric. $\boldsymbol{Q}$ is computed, now $\boldsymbol{F}$ can be replaced/refactored to be invariant with rotation. Thus, $\boldsymbol{F} = \boldsymbol{Q}^T\boldsymbol{F}$. The corotational strain is given by $\boldsymbol{\varepsilon} = \frac{1}{2} (\boldsymbol{F} + \boldsymbol{F}^T) - \boldsymbol{I}$ a variation of Cauchy's infinitesimal strain tensor. This strain is separated into elastic and plastic components. These strains directly affect one another; plastic is initially 0, but as elastic exceeds a threshold, plastic is updated accordingly. This comes into play when simulating deformation; once plastic reaches its material's maximum threshold, its clamped. This helps determine whether deformation occurs, or both deformation and fracture (bending vs. bending and breaking).

Modern Video Games typically have graphical meshes with thousands or even millions of triangles, stored in the GPU. This method performs deformation within the GPU using shader programs, stored using vertex barycentric coordinates rather than Cartesian coordinates. This allows the graphical meshes to be quickly available for operations. It's important that deformation and fracture are performed in a timely manner.

The method utilizes parallelism within code to build *islands* which are either *live, asleep,* or *kinematic*. These islands are formed at runtime based on nodal adjacency. Islands are initially asleep and are awoken (become kinematic) once they are struck by another object. If an island's velocity falls below a certain point for a set amount of time, the island goes back to sleep. These kinematic islands follow a scripted behavior. Islands are dissolved if the fracture occurs within the island, or if some of its nodes change state. The simulations are linearized using Euler integration. The equation that describes each island is $\boldsymbol{Ma + Cv + K(x - u) = f}$**,** where x, v, and a are the concatenated vectors of positions, velocities, and accelerations for all the nodes within the island, f is the vector of external forces acting on the nodes, and M, C, and K are respectively the mass, damping, and stiffness matrices. Solving for $\boldsymbol{K}$, one can rewrite the equation to be $\boldsymbol{M + \Delta t C + \Delta t\ 2K\ \ v + = \Delta t f + Mv - \Delta t K\ (x - u)}$. This allows for computation of new positions and velocities, a necessary tool for deformation and fracture.

By applying the formulas described, utilizing parallelism, and using collision detection, real time deformation and fracture can be achieved, and it has been utilized in the real world in video games made by top tier studios.