Physical Simulation of an Embedded Surface Mesh Involving Deformation and Fracture

Billy Clack*  
Texas A&M University

John Keyser†  
Texas A&M University

Figure 1: (a) Input surface mesh and physical control mesh, (b) tessellation shaders are used to make interior portions of the surface mesh appear like solid geometry, (c) the object is broken apart but all geometry for display has remained static in GPU memory during the entire simulation.

1 Introduction

Physically simulating non-rigid virtual objects which can deform or break apart within their environments is now common in state-of-the-art virtual simulations such as video games or surgery simulations. Real-time performance requires a physical model which provides an approximation to the true solution for fast computations but at the same time conveys enough believability of the simulation to the user. By embedding a complex surface mesh within simpler physical geometry, the mesh complexity can be separated from the algorithmic complexity of the physical simulation. Embedding methods have shown to be successful in production quality products such as the method of [Parker and O’Brien 2009]. In the presence of fracture it is still unclear how to derive the graphical representation of a solid object defined only as a surface mesh with no volume information.

In this work we show how to simulate the deformation and fracture of a solid object defined only as a surface mesh. The algorithm completely separates the physical complexity of the simulation from the graphical complexity of the input mesh. We use GPU shaders to automatically make the surface mesh appear to have interior boundaries when it breaks apart within its environment. Our algorithm is completely automatic and does not require the user to specify any pre-defined cracks for fracturing behavior. We further take advantage of tessellation shaders to simulate different kinds of materials and procedurally generate interior surfaces of the broken object. Our algorithm completely decouples the physical simulation from the surface mesh and does not modify the mesh data residing in GPU memory during the simulation.

2 Our Method

The input to the algorithm consists of a triangle surface mesh and a tetrahedral physical mesh. The tetrahedral mesh can be either created by an artist or automatically generated.

Both the surface mesh and the portions of the physical mesh inside the object’s volume are used for collisions since the physical mesh may be very coarse and might cause ghost collisions outside of the object. The faces of the tetrahedral physical mesh are textured and alpha culling is used during display to only draw the regions which are within the volume of the object defined by the surface mesh. Each triangle of the surface mesh is embedded into a tetrahedron containing the triangle and is stored in GPU memory via barycentric coordinates. This data will only be read during the simulation and can be stored in static memory on the GPU before the simulation begins.

The simulation stage is performed completely on the physical mesh. We use a co-rotational finite element method across all the tetrahedra and derive deformation and fracture depending on the magnitude of the stresses and strains across the volume of the physical mesh.

We display the triangles of the surface mesh by using their barycentric coordinates with respect to their tetrahedron. Since barycentric coordinates are linear across the volume of a tetrahedron, we can rely on the interpolation capabilities of the graphics pipeline to quickly reject fragments of a rendered triangle if they are outside their embedding tetrahedron. We restrict fracture propagation to be along the linear boundaries of the tetrahedral elements in order to prevent costly remeshing. We use GPU tessellation shaders to add geometric detail and coarsen up the flat internal surfaces to provide additional fidelity. The use of tessellation shaders to display interior regions of the object allows quick parameterization of the material relatively cheaply. Throughout the simulation, the surface mesh’s geometry never needs to be modified even when the object breaks into pieces from colliding with its environment.

References