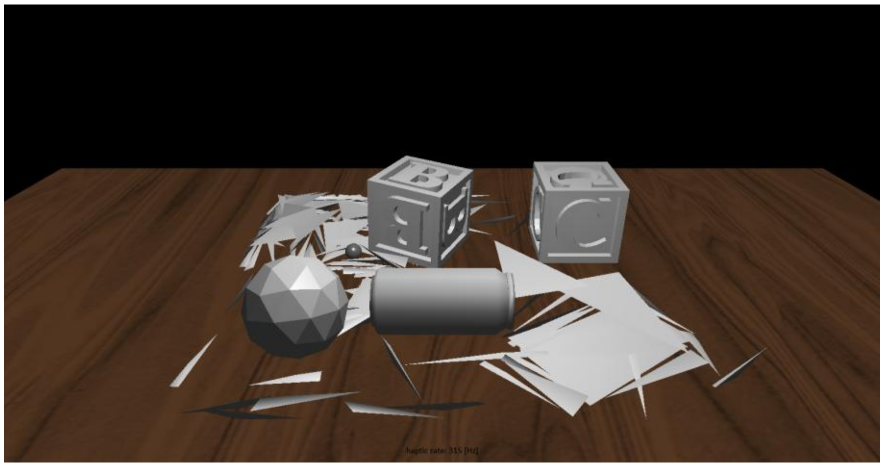
RAGE QUIT

Mindy Huang
Ben-han Sung

Introduction

Our project was born out of a particularly stressful office hour session for CS277, Experimental Haptics. In RAGE QUIT, the user can touch, smash and throw objects that shatter when struck with enough force, providing a satisfying form of stress relief. The project integrates several ideas of haptics and graphics, the key ones being event-based haptics, a robust dynamics engine, and fracturing of meshes.

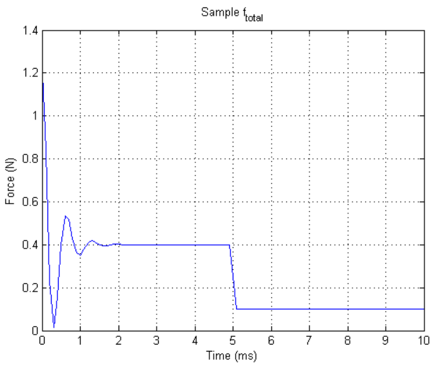


Event-based Haptics

Given that our project is largely about smashing objects, it was important to get the feeling of touching and shattering correct. We introduced event-based haptics, forces that are time-dependent instead of position dependent. We modeled our event forces as a decaying sinusoid summed with a fixed-width pulse like so:

$$f_{event} = A(f_0)e^{-kt}\cos(b\pi t) + \mathrm{r}(ct)$$

Where r(t) is the rectangular pulse. Note that the coefficient A is dependent on the force initially striking the object (f_0). Then our total output force would simply be the sum:



$$f_{total} = f_0 + f_{event}$$

We have two different types of events – a tap event (a gentle touch of an object), and a shatter event (smashing an object. Each event generates a force in the same model as above, simply different coefficients. The coefficients were chosen arbitrarily and hand-tuned.

While working with this model, we ran into stability issues where the added force would oscillate indefinitely. We remedied this by adding a tolerance distance that the tool would have to exceed before the event was triggered again.

Dynamics

We used NVIDIA's PhysX engine since it is a well-known solution and our game requires a robust dynamics engine. We initially evaluated Open Dynamics Engine, which was already integrated with chai3D, but found that the simulation of a large number of particles was unstable. Creating a chai3D-PhysX bridge took longer than expected because of documentation issues.

Fracturing

We attempted to add fracturing using NVIDIA's APEX Destruction extension for PhysX. However, we ran into some issues with the provided runtime libraries. In the interest of time, we decided to write our own fracturing solution based on an object's triangle mesh geometry. For each object, we precompute the fractured pieces by looping through its constituent triangles and forming a tetrahedron out of the ones whose surface area are above a chosen threshold. When a fracture event occurs, we simply replace an object with its precomputed children.

The advantage of this approach is that it works with all object formats based on triangle meshes. However, for best results, it requires a well-tessellated mesh where planar regions are subdivided into smaller triangles. If not provided, such tessellation can be done using graphics techniques like Loop subdivision.