My project this year is entitled Mesh to Voxel Transformations for Optimised Physicals-Based Interactions, and concerns using volume data to resolve collision damage on objects in simulations.

My motivation for this project is the limitations imposed by common destruction methods in current physics engines, such as those used in digital games. These limitations come about from the use of rigid meshes to represent objects. These meshes are efficient for most tasks, especially rendering, where they only store the information which is needed. However, when it comes to reasoning about the interior of an object, only storing the surface makes things difficult.

Calculation of whether a point is inside or outside of an object is complex, and works best on convex objects, meaning most destruction methods spend time first breaking down concave meshes into convex submeshes.

By precomputing an object’s volumetric make-up we gain a constant time lookup of whether any given point is inside or outside of an object. Therefore I propose a method whereby an object’s volume is computed just prior to impact and then each volumetric unit, or voxel, is assigned to either stay with the main object or become part of a fragment, as shown in the figure.

One thing you may be wondering is, if it’s so hard to find whether a point is inside or outside of an object, why bother doing it for all points. However this is only the case for an arbitrary point and when systematically computing all points it becomes less complex.

To achieve this I am using a GPU voxelisation method made up of two phases. The first is a triangle phase where each triangle is processed on its own GPU thread and is projected onto the XZ plane and for each intersecting voxel column the first voxel below, in the Y direction, the triangle is marked.

We then have a propagation phase where we move up the Y plane, XORing each voxel with the one below. This allows us to fill in the solid as when the first voxel is met all above will be set until we reach another set by the triangle phase.

Using this method I can voxelise the Stanford Bunny model, as seen in the figure, in around 0.2seconds.

The first step in fragmenting the object is to calculate the collision information. As I wish all of these steps to be complete before the actual impact, I start the pipeline off prior to collision. Therefore I have to extrapolate to find the collision point and instantaneous force give the approach vector of the incoming object as well as the mass and velocity of both objects. From this information I can generate some normally distributed fragment points in a force dependent radius around the impact point. As shown by the red lines.

I then construct a voronoi partitioning of the object using these points as seeds. This involves labelling all voxels according to which seed they are closest to, or not if they are closer to the force dependant radius than any seeds, as shown by the red voxels.

I then have a flood phase to find any islands. These can be introduced if the object to be destroyed is concave and there is a seed whose partitioning extends over the concavity. My method involves looping over all labellings and for each voxel, flooding recursively to all neighbours changing to a new label. This gives any unconnected areas independent labellings.

My initial approach for meshing the fragments and remeshing the remaining portion of the original object was the marching tertrahedra algorithm. This works by looping over all voxels, forming cubes of 8 voxels and looking up in a predefined array which arrangement of triangles represents the state of that cube, depending on which of the voxels in the cube are set or unset.

I have however run into problems with this algorithm. Firstly, it appears to be running much too slowly. For comparison, the previously mentioned voronoi phase, which also loops over all voxels, runs in around 02,0.3seconds, however the marching tetrahedrons is taking around 2 seconds per fragment which is just not fast enough. Seeing as marching tetrahedrons just involves array loopup and the voronoi method involves vector math, I believe that this slow time could be due to the implementation rather than the algorithm, so I am currently debugging in that area.

Furthermore, as you can see, there is significant detail loss from this algorithm and the voxel structure becomes very clear, which is not ideal. However, from the figure you can see that the underlying fracturing methods are working, even if the remeshing process is not yet complete.

I am currently working on a new remeshing method which should better preserve the original mesh.

The next steps for me are to implement this new remeshing algorithm, which is ongoing at the moment. I also have to optimise all parts of the pipeline for speed. This involves putting the pipeline onto its own thread as currently it runs on the same thread as the rest of the physics engine, causing it to pause while the pipeline executes. I also need to remove redundant or excessive code that is there for debugging or written in the case that I wanted to make sure something would work before properly refining it.

The final stage in the pipeline is post destruction calculations such as giving each fragment its proper mass by taking the number of voxels it is made of as a fraction of the number of voxels in the original object.

I then need to evaluate my method, focussing on framerate, memory usage and visual physical accuracy and complete the write up.