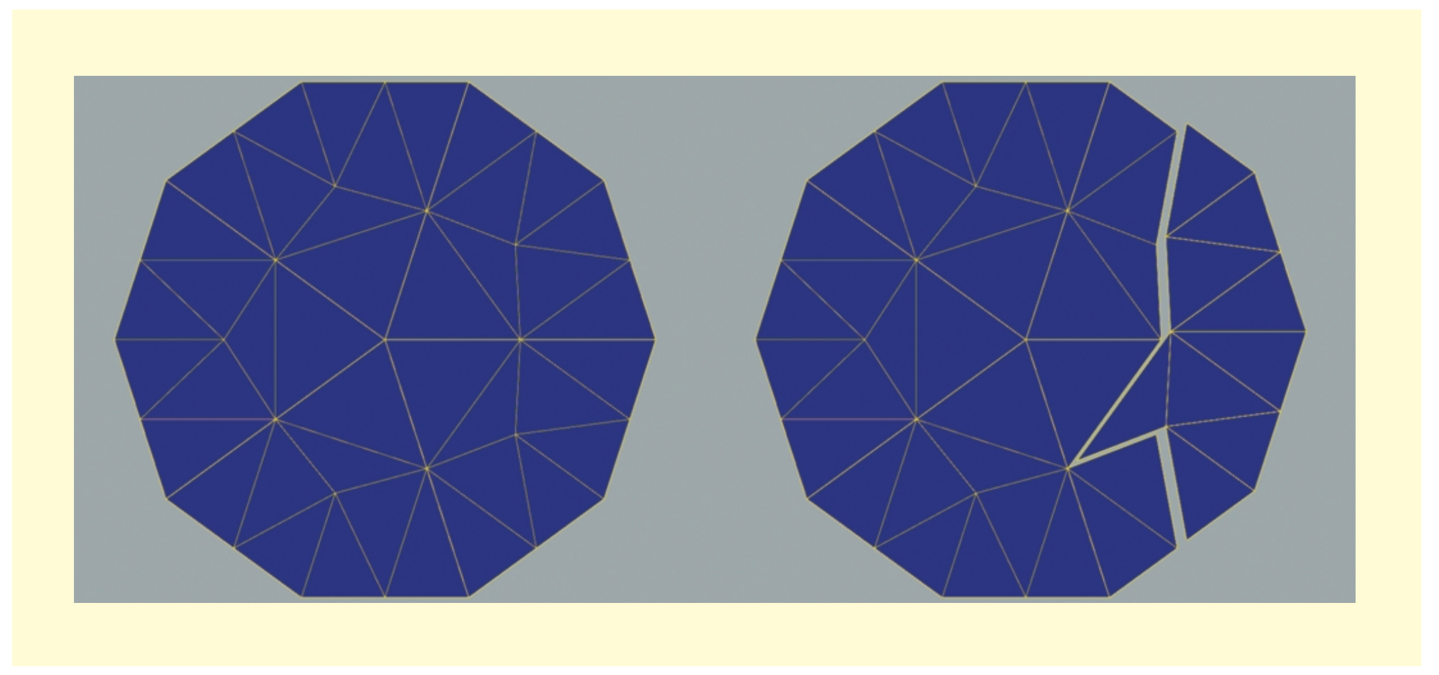
Tiled Crack Networks

Saty Raghavachary

DreamWorks Feature Animation

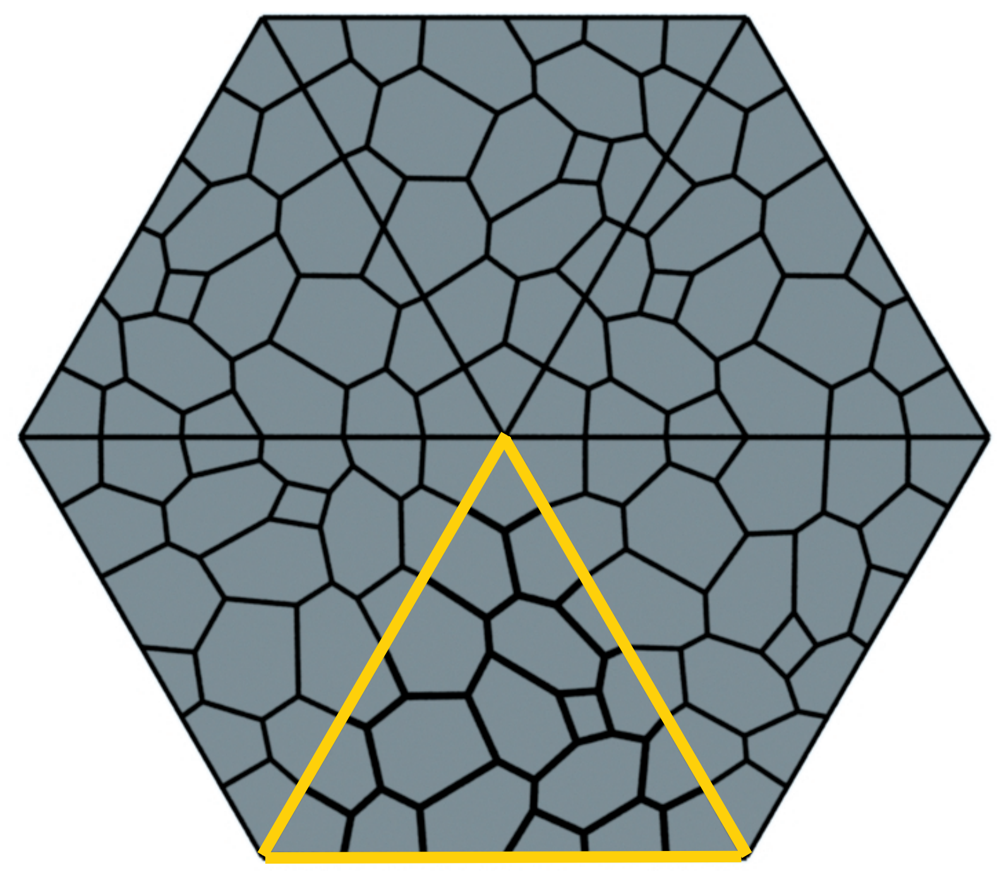
1. Creating realistic cracks on a polymesh is useful for games, visual effects, etc. Simply separating model faces is not adequate, as shown below - doing so leads to unrealistic fragments and crack contours.



Simplistic 'fracture' - not convincing.

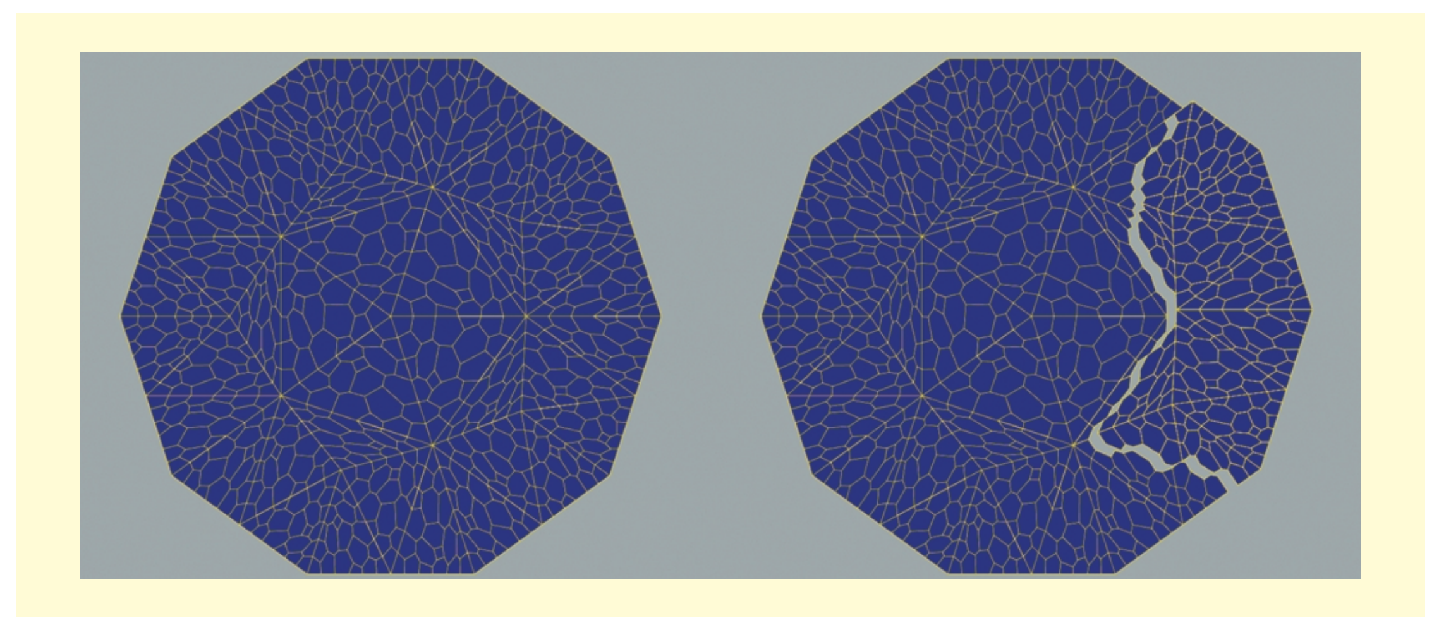
What would be useful is a way to 'decorate' each face so that better-looking crack lines and fractured pieces can result.

2. A 'crack prototile' is an equilateral triangle with a hand-generated crack network. The network has identical, equally-spaced intersections with the three triangle edges, making it seamlessly tileable.



Crack 'prototile' (outlined in yellow) and five more instances of it.

3. Generating cracks across a polymesh is now just a matter of instancing one of the six variations ("conformations") of the crack prototile inside each model triangle. Result: seamless crack networks across the mesh.



Improved fracture - better crack edges, shapes.

Note that the pieces can be fractured again either by using existing edges or by triangulating and instantiating another level of prototiles. The technique is suitable for real-time animation, eg. using DirectX10's support for triangles in geometry shaders and for per-pixel barycentric coordinates.

4. Texture coordinates of model vertices can be used to derive corresponding values on prototile vertices, via barycentric coordinates. As a result, crack patterns on texture maps can drive fracturing.

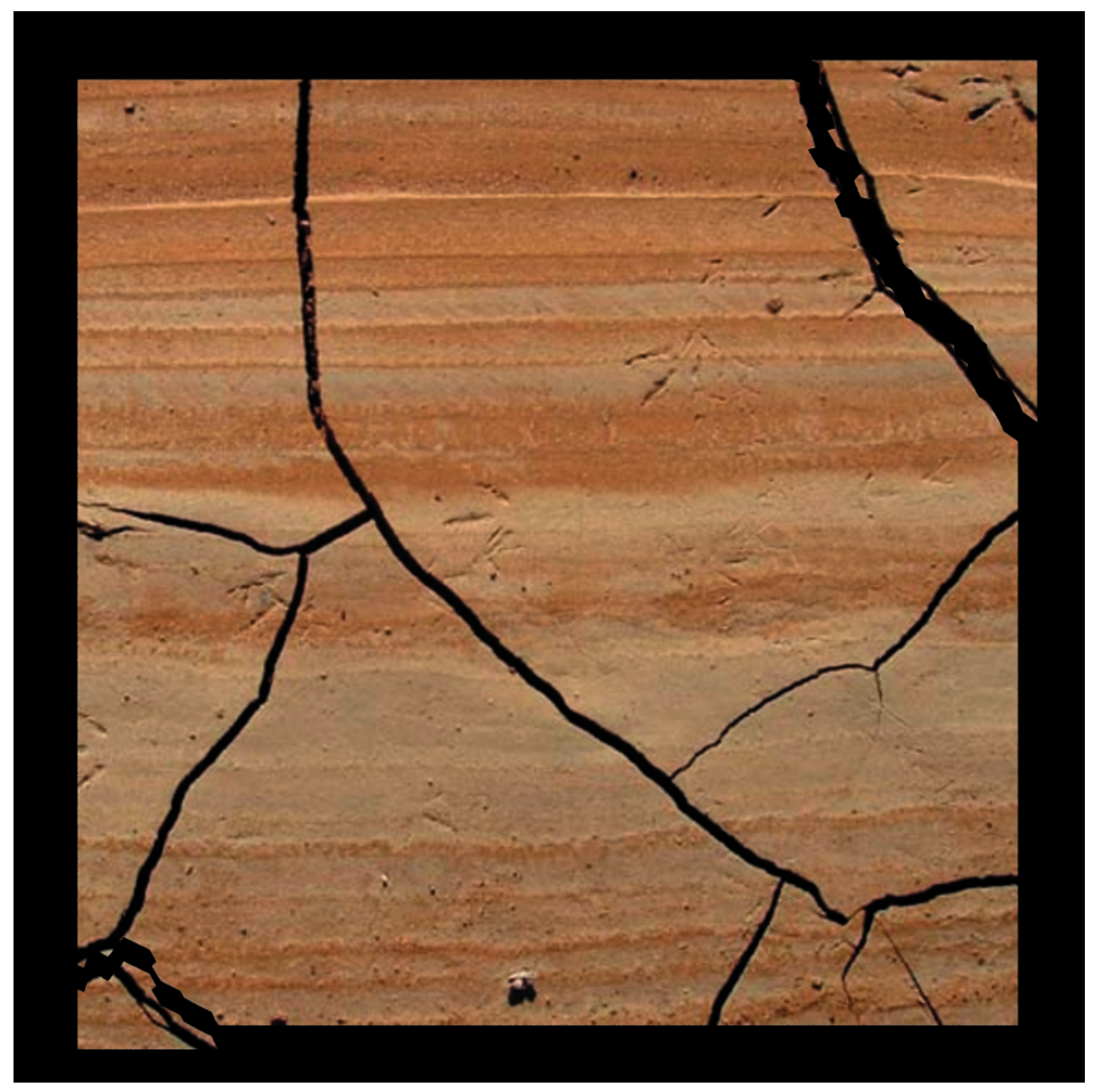
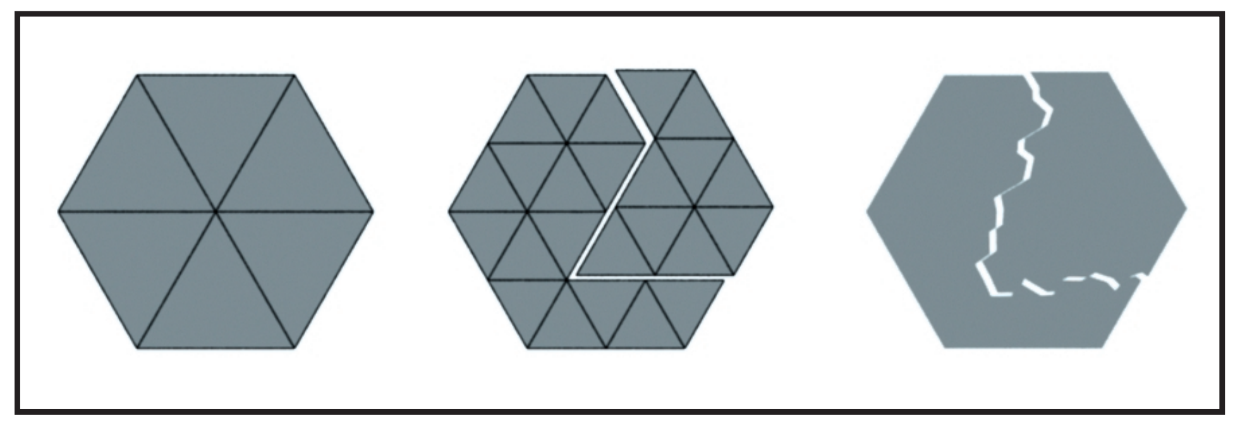


Image-derived fracture: note the broken-off corners at top-right and bottom-left.

Introduction

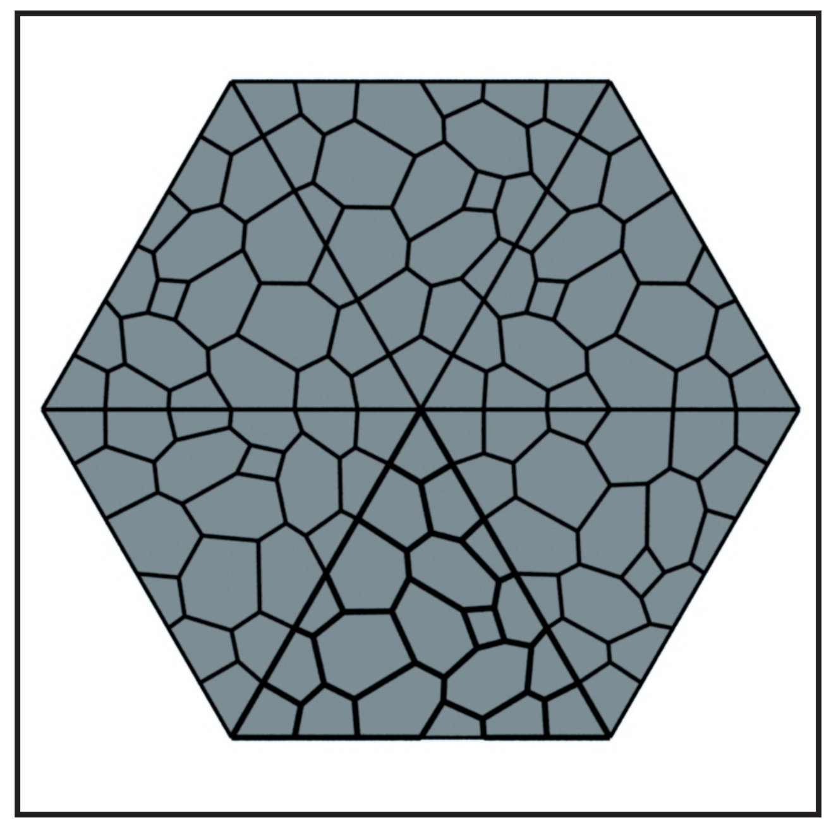
This poster presents a simple, easy-to-implement way to create realistic cracks and fracture fragments on a polygonal mesh.

In the figure below, we would like to crack the hexagon on the left. As seen in the middle, using model polygons to separate the object gives rise to unrealistically smooth edges and sharp corners along the crackfront. The crack pattern on the right is more credible.



Instancing a crack prototile

In the diagram below, the equilateral triangle on the bottom middle (outlined in bold) contains a crack network reminiscent of Voronoi polygons. The polygon edges meet at 'Y' junctions, which produce realistic cracks when the polygons are separated along their edges. Also, the edge polygons intersect the three triangle edges at equal intervals (in our example these are spaced 1/6 units apart) which allows cracks to extend to neighboring triangles when they are tiled. We call such a pre-cracked equilateral triangle a *crack prototile*. Vertices in the crack prototile (including triangle corners and vertices on the edges) are expressed using barycentric coordinates, where any crack vertex V is a weighted combination of triangle vertices V0, V1 and V2 (if w1, w2 and w3 are the weights, V = w1*V1 + w2*V2 + w3*V3).



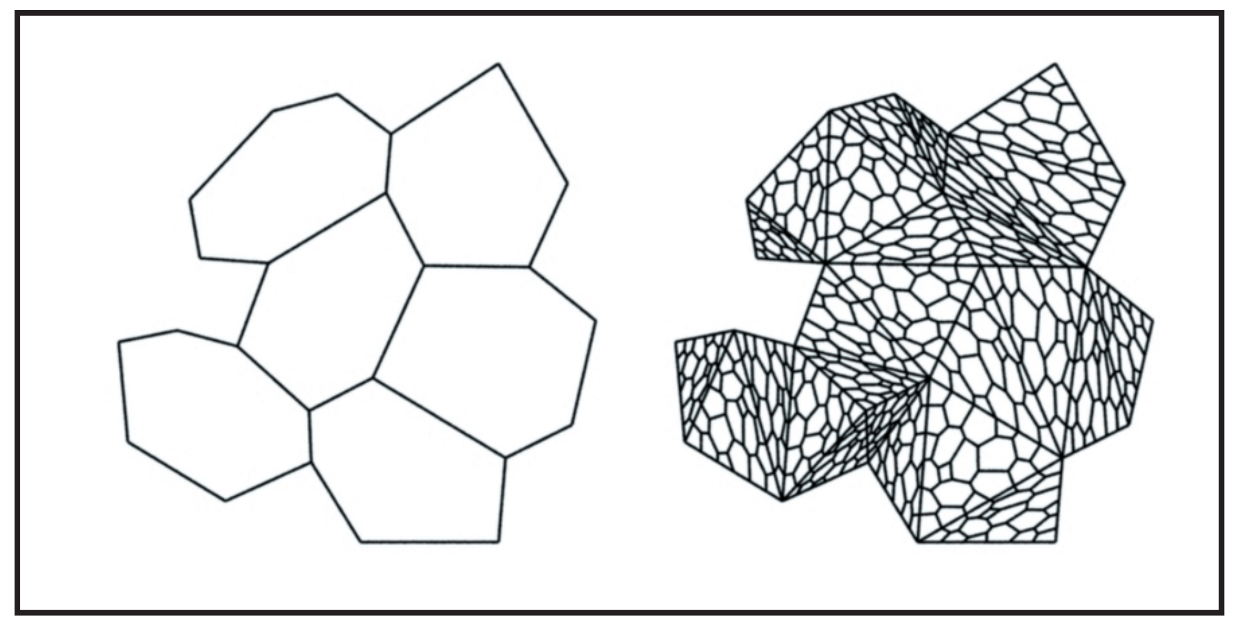
In order to create a seamless crack network over a polymesh, we first triangulate the mesh and then simply instantiate our prototile on every triangle of the mesh. This involves using each mesh triangle's vertex positions along with our location-independent barycentric prototile crack coordinates to recreate the prototile network inside the mesh triangle. In other words, the prototile gets appropriately deformed, translated, rotated and scaled to fit each triangle. Doing this on every mesh triangle creates a seamless crack network across the entire model. We can treat this as a network of 'pre cracks' from which realistic fragments can be derived in many different ways; these can be simulation-derived, procedural or user-driven.

The previous diagram also illustrates that the prototile can exist as six different *conformations* which are obtained by rotating and mirroring the prototile (in terms of barycentric coordinates, this is equivalent to the six permutations of triangle vertices V0, V1 and V2). When we instantiate the prototile on a mesh, we randomly pick one of these six conformations to do so, which creates a pleasing variation over the mesh and effectively masks the fact that a single prototile is being tiled across an entire mesh.

As just mentioned, a single prototile can tile an entire polymesh. However, for extra variation, a small library of them can be employed where each tile has a different polygon distribution.

After a polymesh is fractured into multiple fragments, repeated cracking and fracturing can be achieved by continuing to separate already-fractured pieces into smaller fragments. This is possible because each fragment itself contains seamless crack networks. Alternately, the crack polygons in a fragment can be re-triangulated, and prototile instantiation can occur over those new triangles. This is illustrated below for six polygons from a cracked fragment. Such recursive prototiling does lead to a combinatorial explosion of polygon count, but the resulting crack network is once again seamless across the shard, and can give rise to even smaller cracked pieces and finer crack contours.

In the figure below, the re-triangulation is done by simply connecting existing vertices. This gives rise to skewed triangles, which in turn produce a distorted crack network when populated with prototiles. To avoid this, new evenly-spaced seed points could be added to the interior and edges of the shard polygons, the result triangulated, and populated with prototiles as before.



As shown below, the technique can be used to drive fracturing using photographs of cracks. If the underlying mesh (a square, in our case) is pre-cracked at an appropriately fine level using our prototile, cracks in a texture map can be used to create fragments out of the pre-crack network. In the image below, the broken-off pieces at the top-right and bottom-left illustrate this idea.