Glass system technical documentation

Contents

[Resource creation 2](#_Toc119971509)

[Pattern creation 2](#_Toc119971510)

[Glass creation 2](#_Toc119971511)

[Algorithm description 3](#_Toc119971512)

[SequenceDiagram 3](#_Toc119971513)

[Polygon2D extracted from glass mesh 4](#_Toc119971514)

[Pattern positioned 5](#_Toc119971515)

[Pattern clipped 5](#_Toc119971516)

[Shard polygon computation 6](#_Toc119971517)

[Polygon extruded into mesh 7](#_Toc119971518)

[Physic and behavior 8](#_Toc119971519)

# Resource creation

## Pattern creation

A picture containing chart

Description automatically generated

A fracture pattern is built in a modelling software (Houdini may be a good fit to generate them). It is then imported into Unity.

## Glass creation

TODO

# Algorithm description

## SequenceDiagram

Diagram

Description automatically generated

## Polygon2D extracted from glass mesh

When an impact is triggered (Glass.Break called), the glass mesh is first converted into a 2D polygon (Glass.BuildPolygon).

## Pattern positioned

A picture containing chart

Description automatically generated

“ClipPattern.Clip” extract a list of segment from the fracture Pattern mesh (“ClipPattern.MeshToSegments”) it also applies a random rotation to it and move it onto the impact point (offset). Impact point which was previously moved into object space and scaled at the beginning of “Glass.Break”.

## Pattern clipped

Diagram

Description automatically generated with medium confidence

Then the math start with “ClipPattern.ClipEdges”, each line position is compared to the glass polygon, any segment fully in is kept, fully out is discarded, and if it’s crossing, the point outside is replaced with the intersection point (“ClipPattern .IntersectPolygon”).

The edges lines are then built by using the initial glass polygon points list, adding all the points that were found on intersections to it, then sorted by vector angle to origin in “ClipPattern. BuildEdgeLines” to create the edge segments.

## Shard polygon computation

Chart, radar chart

Description automatically generated

The next step is to calculate each shard polygon from the list of segments. This is done by the ShardPolygonBuilder class. The algorithm is inspired by this stack overflow question: <https://stackoverflow.com/questions/35468830/efficient-algorithm-to-create-polygons-from-a-2d-mesh-verticesedges>

Each point gets an identifier number. and a list of all point is initialized, containing itself a list of all point that are linked to it. In our example above, the first 3 element of that list will be:

0: 1, 60, 17

1: 0, 2

2: 1, 60, 3

…

Then for each point, their edges are sorted in clockwise angular order. So we can later navigate into the graph. The resulting graph is duplicated because one is used for navigation and the other one is consumed to use each edge a single time (in both direction)

“PopLoop” is called as long as the consumable list of edges is not empty, it start from any remaining point, first jump to any connected edge, then continue jumping on the first edge at his right until it get back to the initial point and return that loop. Because they are sorted, the right edge is always the next element in the list of opposite edges.

## Polygon extruded into mesh

Graphical user interface

Description automatically generated

« Glass.CreateMesh » is then called for each shard. UV are interpolated using barycentric coordinate of the first triangle of 2D polygon.

Finally “Glass.SpawnShard” creates the new gameobject and pass tag, rigidbody velocity, materials, etc.

# Physic and behavior

TODO