Shatter-o-matic by Evolvedlabs SAS

staff@evolvedlabs.com

www.evolvedlabs.com

DOCUMENTATION 1.0.1

Table of Contents

Feature overview

Thank you for purchasing Shatter-o-matic! This Unity3D asset allows you to easily slice and shatter **static meshes**: with only a few clicks, your project will gain that extra layer of realism you were looking for!

This project offers two main features that can be accessed through the use of easy to use components, that can be easily triggered by either other gameobjects on collision or via specific code commands.

Shattering can be triggered from a specific point (which could be a contact point from a collision, or an arbitrary point e.g. picked via mouse).

How you compute that shatter point (which simply applies the force on the already computed point - the shattering itself doesn't *start* from that point) or the slice plane, is up to you. You can find some examples that show how to trigger this manually (via a mouse cut, or shatter click) or organically in your scene (via gameobjects interacting with each other).

Slicing is done via a plane that's passed to the slicing function. Such plane can be computed by a collider, or it can be passed as a computation (e.g. mouse cut) . Consider checking the example code if you need help in computing the cutting plane you wish to use.

Every function offered here is exposed with the full source code. This means that while this project is very portable, it may perform significantly differently in terms of speed when run on mobile devices vs a personal computer - always double check the performance on your target device before shipping.

Shatter-o-matic also comes with **baking** tools so that, especially where performances are a concern, you can pre-bake the desired fragments/slices.

The demo scenes are built for the built-in pipeline. If using URP:

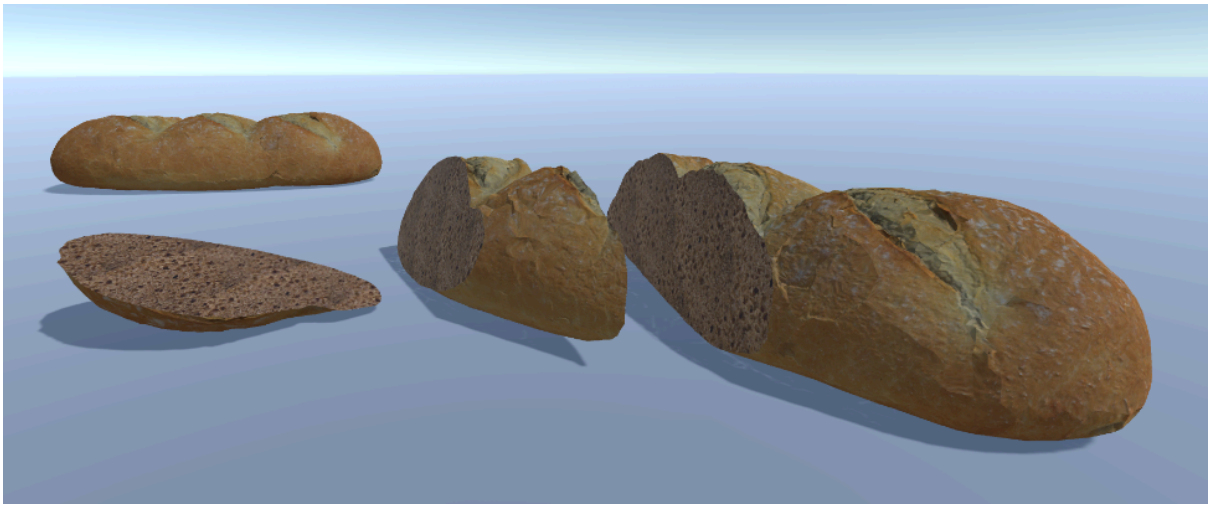
Open: Window → Rendering → Render Pipeline Converter
(or in older versions: Edit → Render Pipeline → Universal Render Pipeline)
Run:
    Material Upgrade
    Built-in to URP conversion
If needed:
    Select materials manually
    Change shader to:
    Universal Render Pipeline/Lit

Getting started

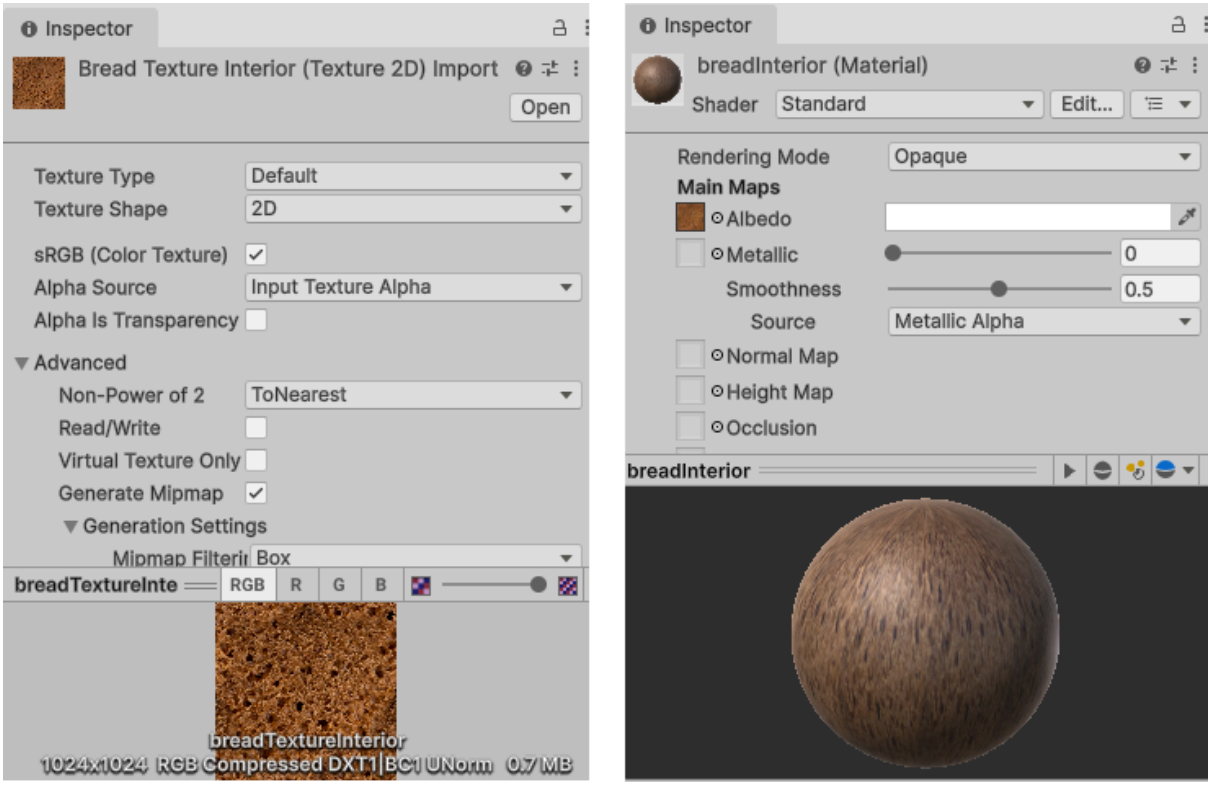
Depending on what you're trying to do, you probably want to attach one of the following components to your gameobjects: SlicerBehaviour.cs to slice meshes, and ShatterBehaviour.cs to shatter meshes.

SlicerBehaviour.cs

Attach this component to a static mesh. When the cut is triggered, the cut is done by extending a plane through the full geometry, resulting in two new static meshes.



When cutting, it is possible (and suggested) to also fill in the **mCutFaceMaterial** field - as the mesh is cut, the resulting empty part will be filled with triangles using such optional material.



In the above example, you can see how this material is used to fill in the loaf of bread with a proper material texture.

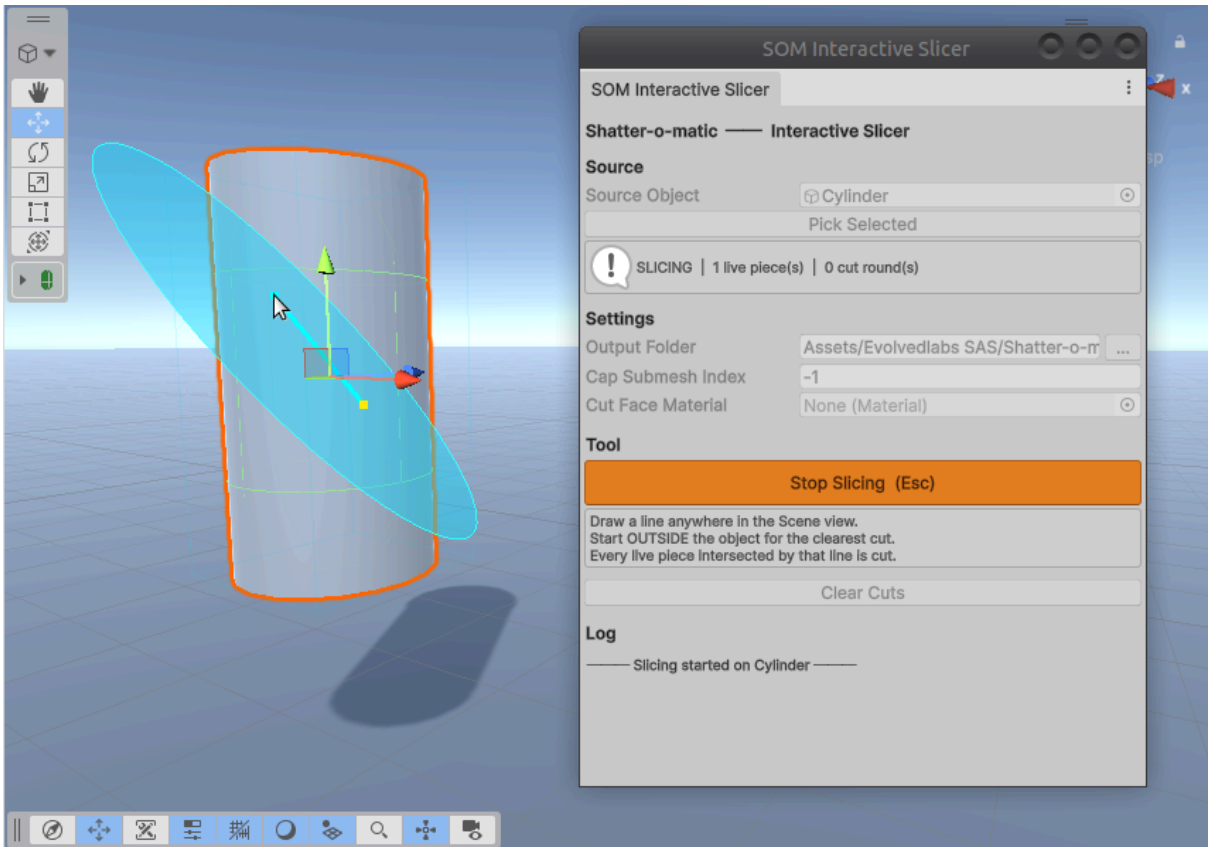
It is advised to use a textured and tileable material so that the result is pleasing to the eye, for example, so that when you're cutting the mesh of a loaf of bread, the inside of the bread is filled properly.

Such field accepts multiple materials: upon every new surface, a random material will be picked and filled.

It is possible to setup this behaviour so that every new half automatically gains the SlicerBehaviour component itself, allowing for continuing cutting the mesh as needed. If such value is set to **-1**, then there will be no limit to how many times you can cut a mesh.

Beware of setting a proper limit and/or the auto cleanup function depending on your game needs.

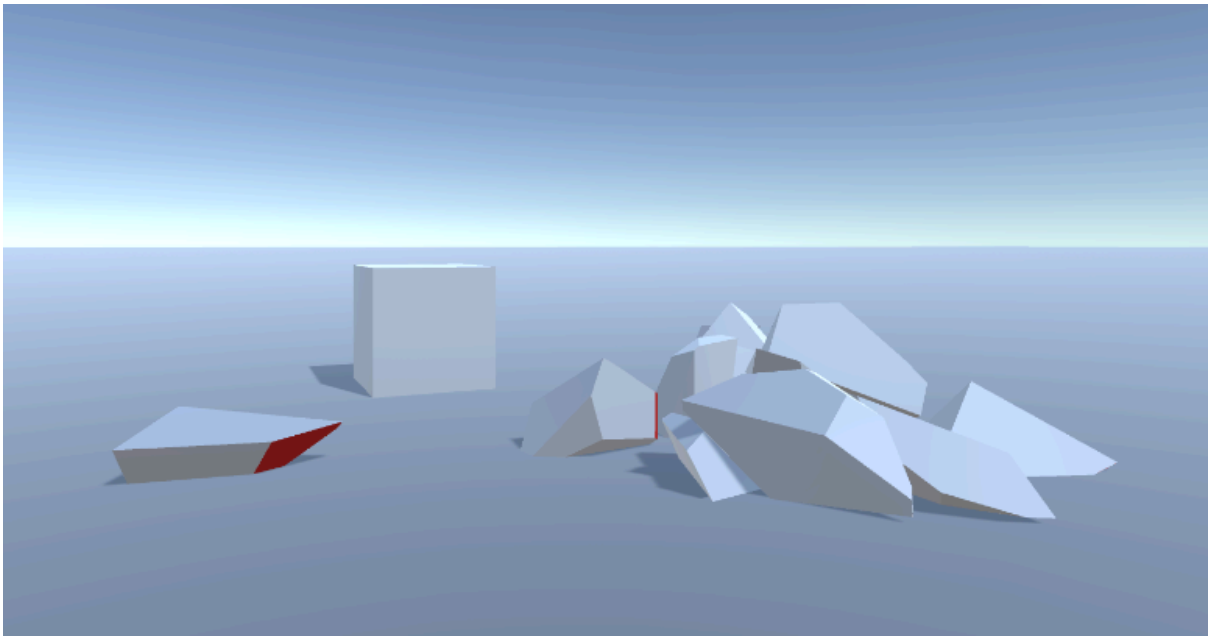
You can even pre-slice meshes directly in the editor via the **"Tools" →"Shatter-o-matic" -> "Interactive Slicer"** editor window: click start slicing, simply drag and drop to slice, click stop! Done!



Slices are generated and placed automatically by default as a child in the **/Assets/EvolvedLabs SAS/Shatter-o-matic/GeneratedSlices/<assetName>/** directory where this asset resides, each cut with its own dedicated prefab.

ShatterBehaviour.cs

Attach this component to a static mesh. When the shatter is triggered, the current gameobject will be substituted with the properly computed 3D objects that represent a shattered version of the item itself.



As computing the shattered component can be computationally taxing, it is advised to pre-bake the shattered component. You can easily do that from the **“Tools” →”Shatter-o-matic” → “Fragment Baker”** editor window (you can bake multiple gameobjects there too).

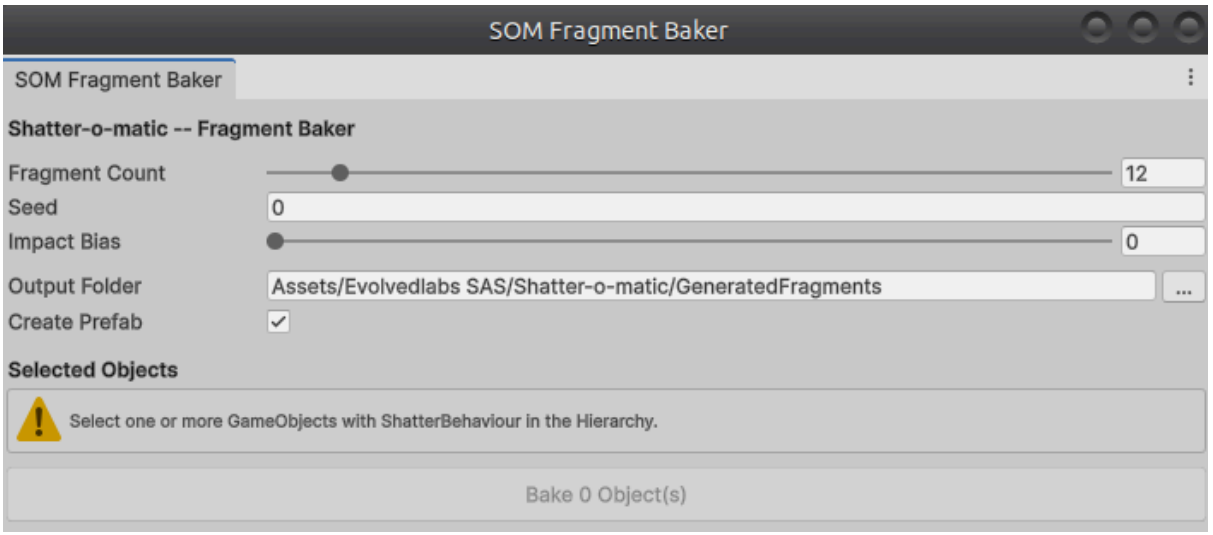
The **mCutFaceMaterial** field behaves exactly like in the SlicerBehaviour.

You can either choose to spawn the shattered-gameobjects at runtime, or pre-cache the elements upon instantiating the actual object.

It is possible to bake the fragments on awake, but please note this may slow down your load times.

Additionally, a fast shatter method (which relies on the SlicerBehaviour) that can be leveraged runtime is available, although for better visual representation the baked shatter method is strongly advised.

Like the SlicerBehavior, it is possible to automatically spawn the fragments with a SlicerBehaviour (not a ShatterBehaviour!) component attached, as well as which materials to use in order to complete the fragments “inner” used.

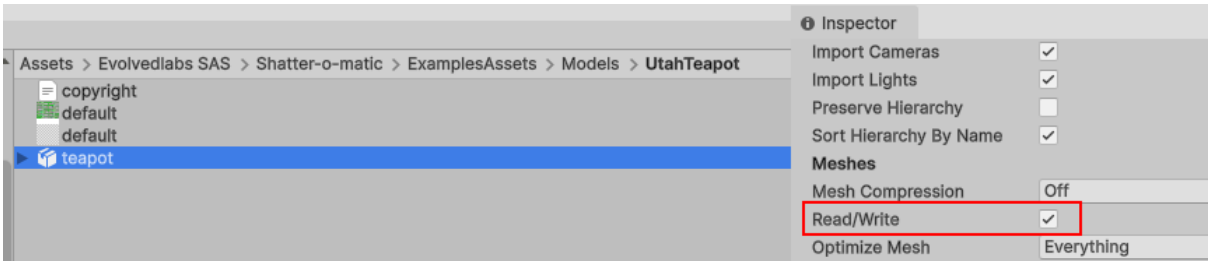


Fragments are generated and placed automatically by default as a child in the **/Assets/EvolvedLabs SAS/Shatter-o-matic/GeneratedFragments/<assetName>/** directory where this asset resides.

Note on baking (Both for slices and fragments)

Please remember that you need to set the **Read/Write** option in the Unity inspector in order to bake a mesh.

It is recommended that you slice/bake using the appropriate scaling value for your asset.



Examples overview

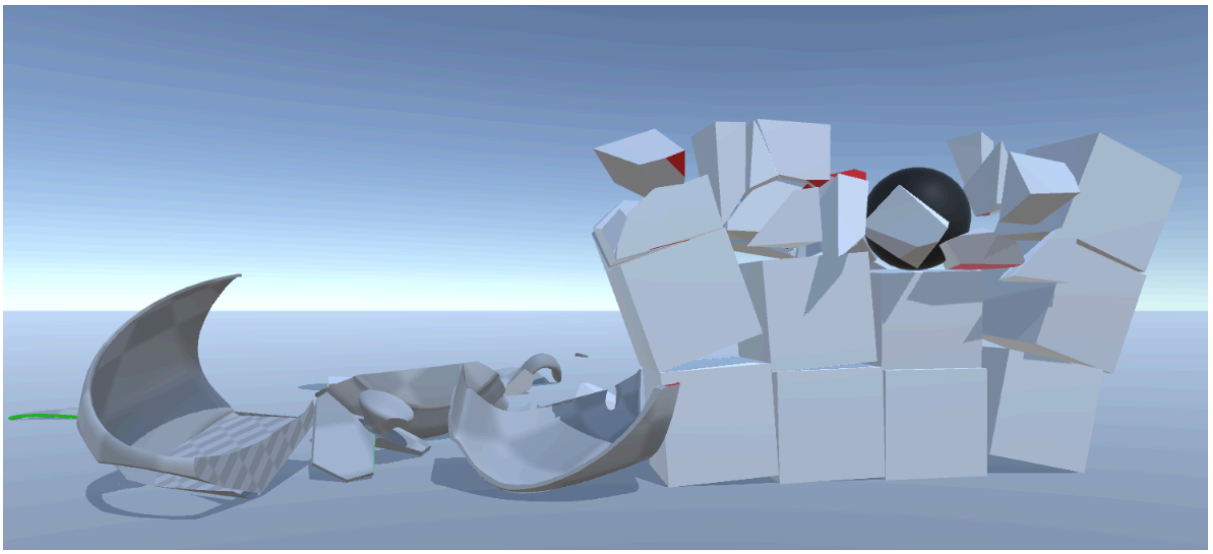
This project contains multiple example scenes to showcase what's possible with the many features offered. Here is a short overview:

Shatter Example

This example shows how to trigger the shatter behaviour. All the objects have an attached Shatter behaviour that can be triggered either manually via code (simply left click the item) or by organic interaction between gameobject (in this case, represented by two black wrecking balls).

Note that these gameobjects do not automatically inherit the Slicer behaviour (see the next paragraph), but it is possible to do so if you wish directly from the inspector. If you do, make sure to also enable **mAllowSlicing** in the ExampleManager.

Note that this asset comes with pre-baked fragments and a prefab item.



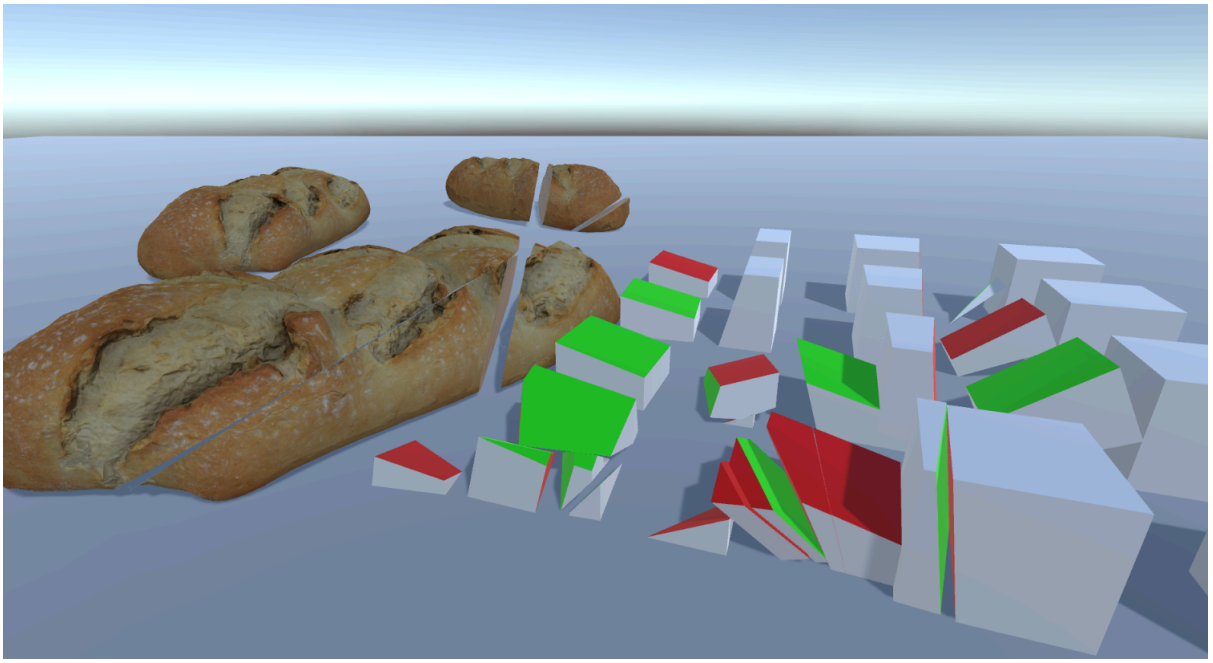
To see how the cut and shatters are triggered, refer to SOMExampleManager.cs and WreckingBallExample.cs files.

Click **'Left mouse button'** to trigger the shatter on a GameObject.
Press **'F1'** to trigger the left wrecking ball fall.
Press **'F2'** to trigger the right wrecking ball fall.
Press **'R'** to reload the scene.

Mouse Cut Example

This example shows how to slice static meshes via an arbitrary plane computed via code, in this case, projected from the camera position.

Simply left click anywhere in the screen, keeping the left mouse button pressed - a line will be drawn on screen to show the plan that will be projected from the camera towards that line.



Make sure that line overlaps (even partially) any of the 3D objects in the scene, and release the left mouse button when you want to perform the cut.

Note how the bread loaf only has one material set up in the mCutFaceMaterial field. This means that every time it gets cut, the same material will be applied to fill the cut face on both sides.

Note how the cube instead, have two materials, redMaterial and greenMaterial. Each time they're cut, the filling material will be picked at random from the pool, for each side of the cut.

Notice that the newly cut parts for static meshes can be furtherly cut down up to 16 times, since the option to limit the cut is set to 16 (default value). Beware of this value in production as you likely want to set a limit (You can set it as -1 for no limit).

To see how the cut and shatters are triggered, refer to SOMExampleManager.cs.

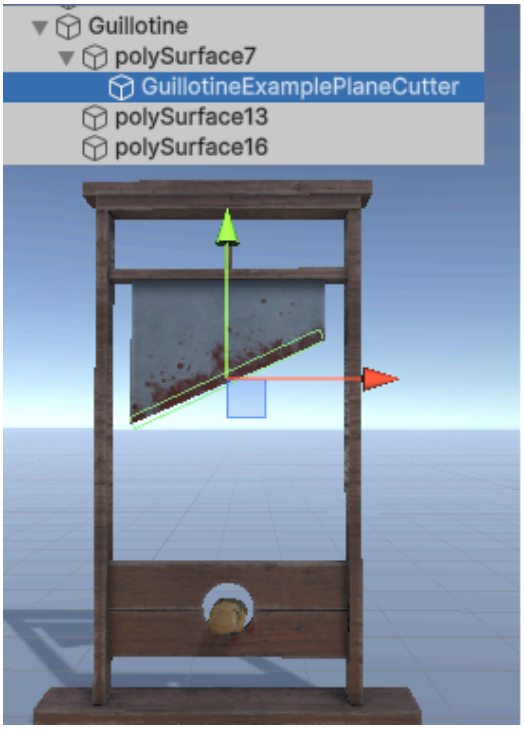
Press **'R'** to reload the scene.
Use **'Left mouse button' (hold, drag and release)** to draw a line on the screen and cut any object in the scene.

Guillotine Example

This example shows an animate guillotine 3D model, where a gameobject acts as the blade that will compute the plane used for cutting, in this case, the loaf of bread. Such model is setup with attached the SlicerBehaviour component.

Expand the guillotine gameobject hierarchy to find the blade with the GuillotineExample component attached to the blade.

Upon **collision**, the slice with the computed plane will be fired. See the example file for guidance on how to compute a proper plane upon collision.



To see how the cut and shatters are triggered, refer to SOMExampleManager.cs and GuillotineExample.cs.

To trigger the guillotine, simply press **'F1'** while in play mode.
Use **'Left mouse button' (hold, drag and release)** to draw a line on the screen and cut the bread loaf.
Press **'R'** to reload the scene.

Credits and acknowledgements

The example scenes in this package contain assets that have been made available for usage under the Creative Commons Commercial license (or similar). Such assets and their attribution link are listed below:

https://sketchfab.com/3d-models/the-utah-teapot-1092c2832df14099807f66c8b792374d
https://sketchfab.com/3d-models/guillotine-animation-d6578126d0af4641b588cd7039562fd1
https://sketchfab.com/3d-models/bread-16428c0c6c964a3d80b9e791116aae65
https://www.freepik.com/free-photo/close-up-view-texture-rye-bread-background-uses_8328409.htm

Should you choose to re-use these assets in your project, it is up to you to make sure that the relevant attribution is included in your project.

This product contains snippet of codes from https://github.com/hugoscurti/mesh-cutter released under MIT License Copyright (c) 2019 Hugo Scurti - such parts, mostly math libraries (and only such parts) are under the following terms:

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Some other parts of the code are based upon https://github.com/unitycoder/SimpleMeshExploder

Shatter-o-matic has been developed by Evolvedlabs SAS - all rights reserved. A valid license purchased via the Unity3D asset store is required to use this product in any capacity, commercial or otherwise.

Check out our Unity3D Asset Store page for more assets!
➤➤➤ https://assetstore.unity.com/publishers/116354

Troubleshooting and support

The full source code in C# is available for you to extend functionality. The examples provided here are just a quick way to showcase what's possible and you're expected to extend / write your own code in order to trigger the cut or shatter behaviour as you see fit for your game.

If you have questions or you'd like to report a bug, please reach out to staff@evolvedlabs.com - below, some common troubleshooting questions and their answer.

Q: Can I cut or shatter skinned meshes?
A: Unfortunately no. At this time Shatter-o-matic only supports static meshes.
Q: Do I have to bake fragments or slices?
A: This depends on your application. Depending on your target device, runtime generation could be acceptable. For faster performance, baking is always advised.
Q: How can I manipulate fragments/slices after they spawn?
A: Both the SlicerBehaviour and the ShatterBehaviour will return a list of fragments - simply use those as you see fit, or you can get started by extending the SOMmanager class if you wish to keep track of everything.
Q: The auto cleanup function doesn't work!
A: You need a gameobject with the SOMmanager component attached. See the examples.
Q: The auto cleanup function simply destroys the gameobjects. What If I want to do something visually nicer, for example with a shader?
A: You can extend SOMmanager's
protected virtual IEnumerator executeCleanup(GameObject iGo, float iSeconds)
method.
Q: Can I change the default path where baked fragments and slices are saved to?
A: Yes, you can change the default paths in SOMbehaviour.cs

```
public static string sFragmentsPath  = "Assets/Evolvedlabs SAS/Shatter-o-matic/GeneratedFragments";
public static string sSlicesPath        = "Assets/Evolvedlabs SAS/Shatter-o-matic/GeneratedSlices";
```

Q: How do I enable logging?
A: You can define the SOM_LOGGING symbol in your Project settings -> Player -> Script Compilation -> Scripting Define Symbols.
Q: Everything in the demo scenes is pink/purple! Help!
A: You need to convert the materials used in the scene to URP. Use Window → Rendering → Render Pipeline Converter
Q: I get "InvalidOperationException: You are trying to read Input using the UnityEngine.Input class, but you have switched active Input handling to Input System package in Player Settings." errors in the console when running the demo. Why?
A:You need to change your "Active input handling" to "Old" or "Both"