Turbo Slicer Guide

Turbo Slicer version 2.0

Turbo Slicer can split any object featuring a mesh. There's a few things you can do with it, a few things you need to know and a lot you probably don't but I'll put here just in case.

You can call down a slice directly with the Turbo Slice API or use the "attachable slicer," which are both explained in this manual.

Turbo Slicer includes three demos: a conventional touch-slicer, a steerable sword and a set of oranges on a table. We've also included a glowing trails effect which, like the touch slicer, supports multitouch.

Viewing offline? Find the most current version of this document here.

Find Turbo Slicer on the Unity Asset Store here.

Find Noble Muffins on Facebook here or follow us on Twitter here.

Contact

Need help? Write us at support@noblemuffins.com.

Acknowledgments

Alex Patras created Turbo Slicer 2's cover art. Find his portfolio here, or find him on Upwork here.

Mathias Nielsen helped write the demo code. You can hire him here.

Tangent recalculation is based on the work of Eric Lengyel in his 2001 paper "Computing Tangent Space Basis Vectors for an Arbitrary Mesh." We've uploaded I copy of this document here.

John Ratcliff, a software engineer at NVIDIA, wrote the basic Plane-Triangle split in C++ and his code can be found here. This kit began as a translation of his code into C#, but has since been heavily expanded upon.

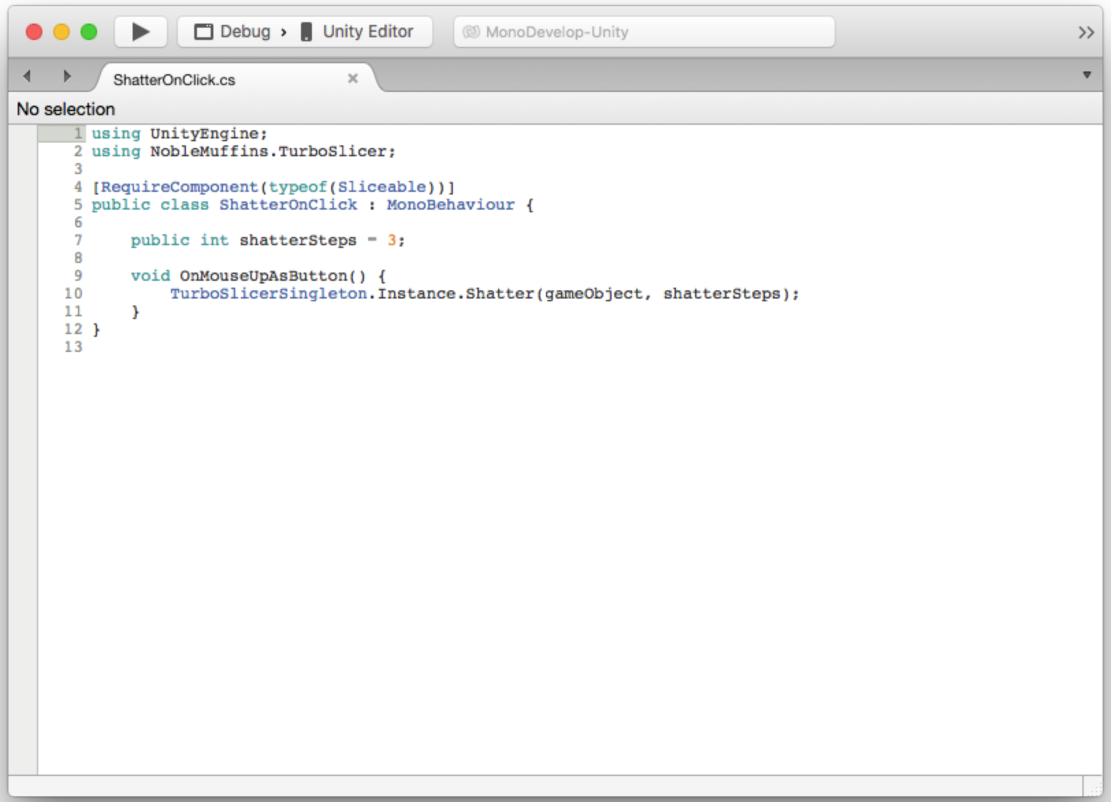
A vector-vector transformation algorithm used is pulled from a 1992 forum post by Benjamin Zhu, then an employee of Silicon Graphics Inc. The thread can be found here.

The original Turbo Slicer cover art is the work of artist Nigel Kitts.

Special thanks to Piotr Wolf, for creating the original Turbo Slicer demo, Synergy Blade.

Before You Start

You can access Turbo Slicer via the static property **TurboSlicerSingleton.Instance**, in the **NobleMuffins.TurboSlicer** namespace. You do not need to manually create an instance; one will be created automatically.



```
using UnityEngine;
using NobleMuffins.TurboSlicer;

[RequireComponent(typeof(Sliceable))]
public class ShatterOnClick : MonoBehaviour {

    public int shatterSteps = 3;

    void OnMouseUpAsButton() {
        TurboSlicerSingleton.Instance.Shatter(gameObject, shatterSteps);
    }
}
```

Preparing an object

At minimum, Turbo Slicer needs to be able to find a single **MeshFilter** and **MeshRenderer** on the object, or a single **SkinnedMeshRenderer**.

However, you should add a single **Sliceable** component to your object. This will allow you to configure Turbo Slicer's behaviour. Turbo Slicer will attempt to find a Sliceable component on your object and read it for configuration. This component (Sliceable) is also necessary if you want to execute code after a slice.

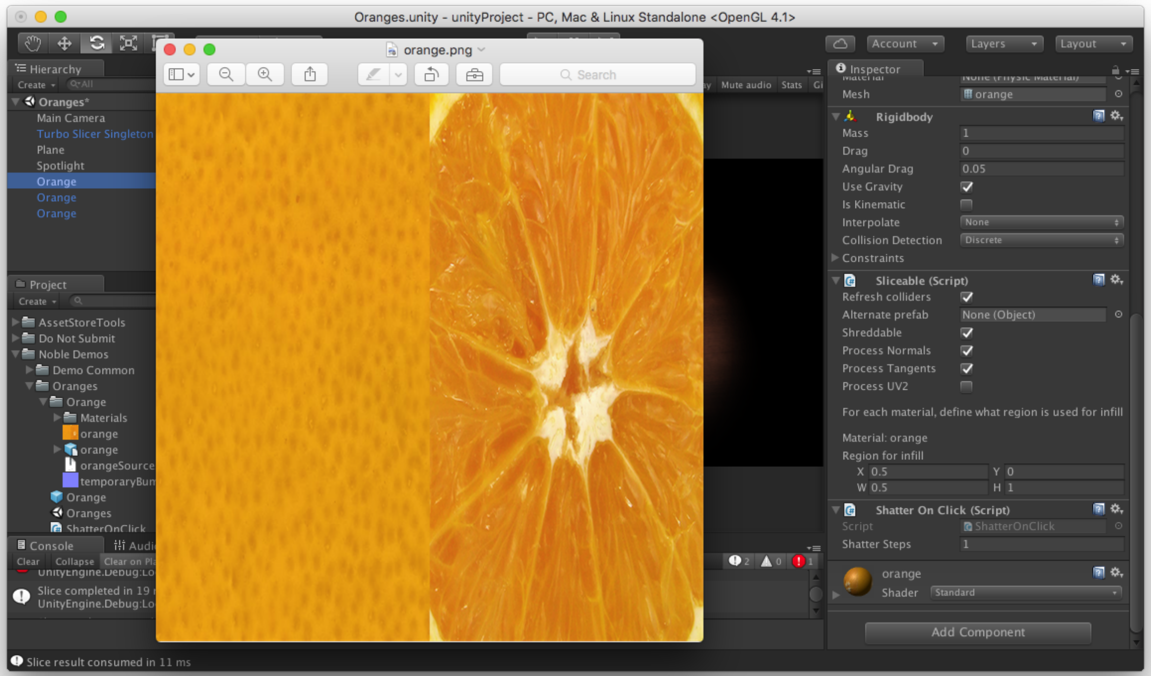
Infill

If you want Turbo Slicer to fill in the slice hole, you'll need to configure it, and you'll need to provide a whole, closed object without any duplicate faces.

Turbo Slicer will examine the **cross section** of the slice and try to find one or more **closed polygons**. An object like a plane will not work because there won't be any polygons in its cross section. A ball, a torus or any whole, **closed** object will have polygons in its cross section.

For performance reasons, Turbo Slicer is designed to fill the hole using the same material as the rest of the object. So you infill texture needs to be part of your main texture, and you need to use the Sliceable component to tell Turbo Slicer where, in your material, is the texture for the slice.

Let's have a look.



Mesh: orange

Rigidbody
Mass: 1
Drag: 0
Angular Drag: 0.05
Use Gravity: ✓
Is Kinematic: ☐
Interpolate: None
Collision Detection: Discrete
Constraints

Sliceable (Script)
Refresh colliders: ✓
Alternate prefab: None (Object)
Shreddable: ✓
Process Normals: ✓
Process Tangents: ✓
Process UV2: ☐

For each material, define what region is used for infill

Material: orange
Region for infill
X 0.5 Y 0
W 0.5 H 1

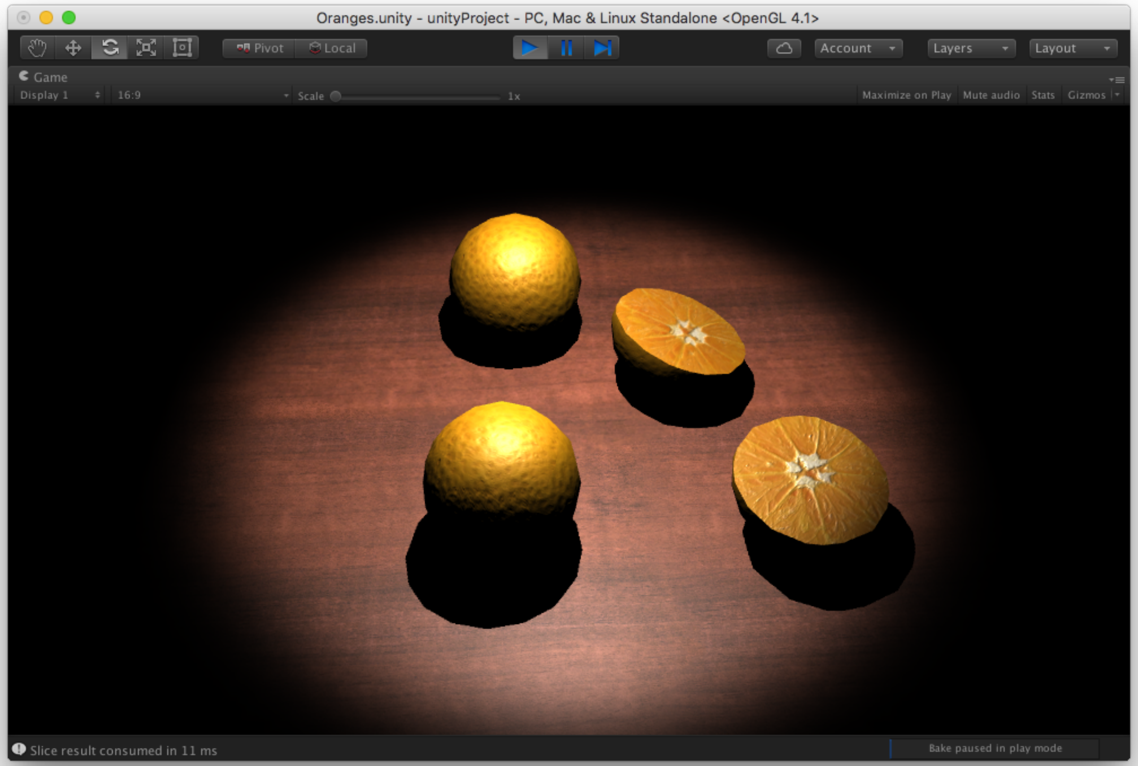
Shatter On Click (Script)
Script: ShatterOnClick
Shatter Steps: 1

orange
Shader: Standard

```
UnityEngine.Debug:Lo
Slice completed in 19
UnityEngine.Debug:Lo
Slice result consumed in 11 ms
```

Here we see a texture where the left half looks like an orange, and the right side looks like the inside of an orange. Our mesh – a simple sphere – has UVs mapped to the left side. We want to tell Turbo Slicer to texture the infill with the right half of this material.

We do this by setting the "region for infill", seen in the lower right side of the above screenshot. The X and W values are both 0.5, tells Turbo Slicer that the infill texture is half the width of the whole image, and starts halfway across from the left.



Slice result consumed in 11 ms

Objects with Multiple Materials

If your object has multiple materials, Sliceable will allow you to set the region for infill for each. However, you must understand that for each material, Unity will split the mesh into different "submeshes," and Turbo Slicer will slice and infill each material separately.

If these separate submeshes appear to be a single, closed mesh, than your object actually consists of multiple, open meshes, none of which will infill correctly.

Refresh Collider

If you object has colliders, you probably want Turbo Slicer to update them after a slice. This feature is compatible with SphereCollider, BoxCollider and MeshCollider. If necessary, you can turn this off using the checkbox in the Sliceable component.

Clone from Alternate Prefab

Turbo Slicer creates the slice results by cloning the source object. If you want, however, Turbo Slicer can create slice results by cloning from a specified prefab.

Suppose you have an object from prefab A, and a matched prefab B. This matched prefab has a matching hierarchy, but (for example) has different scripts attached or a different material or configuration.

To set this up, add the Sliceable component to your object, and drop your alternative prefab into the "Alternate Prefab" slot.

Scripting

Slicing

There are five public APIs. They are not static.

```
public void Slice (GameObject subject, Vector4 planeInLocalSpace, bool destroyOriginal)
```

Given a plane in the object's local space, in point-normal form, slice the object. This method will spawn new objects, and the "destroy original" parameter tells Turbo Slicer whether it ought to delete the subject for you. Do not delete it yourself; slicing may occur later, asynchronously, and Turbo Slicer needs the subject to hang around until it's ready to work with it.

```
public void SliceByLine (GameObject subject, Camera camera, Vector3 start, Vector3 end, bool destroyOriginal)
```

Given a camera, and two points in screen space, slice the given object. This method generates a slice plane based on a line drawn across the screen.

```
public void SliceByTriangle (GameObject subject, Vector3[] triangleInWorldSpace, bool destroyOriginal)
```

Slice a given object by forming a plane from three given vertices (in world space) which lie on that plane.

```
public void Shatter (GameObject subject, int steps = 3, bool destroyOriginal = true)
```

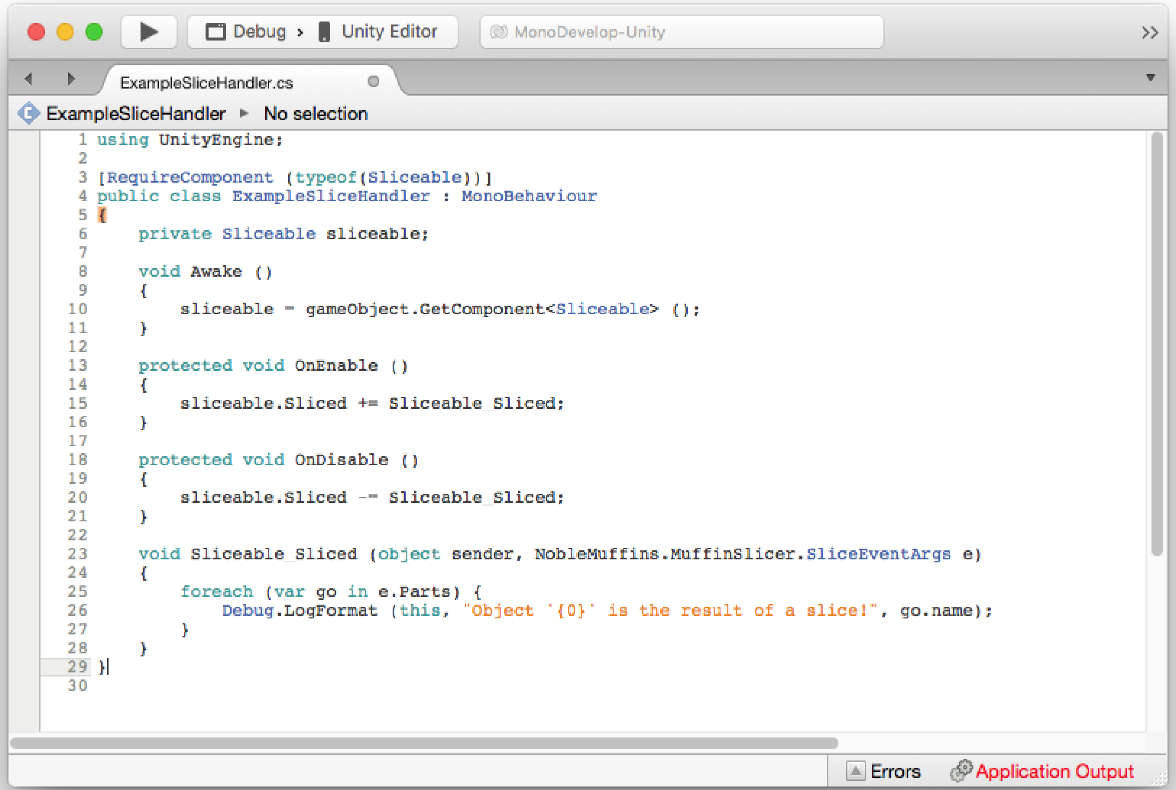
Shatter an object. Specifically, slice it randomly a given number of times. If done asynchronously, Turbo Slicer may perform the shatter over the course of a few frames.

Handle Slice Results

Often, you will want to process each shard produced by a slice or shatter.

To do so, you will need to write a script and attach it to your object. This script must subscribe to the Sliceable component's "Slice" multicast delegate.

If you import the examples ("Noble Muffins/Turbo Slicer 2/Examples.unitypackage") you'll find "ExampleSliceHandler.cs" which provides a simple, minimal example. Observe:



```
using UnityEngine;

[RequireComponent (typeof(Sliceable))]
public class ExampleSliceHandler : MonoBehaviour
{
    private Sliceable sliceable;

    void Awake ()
    {
        sliceable = gameObject.GetComponent<Sliceable> ();
    }

    protected void OnEnable ()
    {
        sliceable.Sliced += Sliceable_Sliced;
    }

    protected void OnDisable ()
    {
        sliceable.Sliced -= Sliceable_Sliced;
    }

    void Sliceable_Sliced (object sender, NobleMuffins.MuffinSlicer.SliceEventArgs e)
    {
        foreach (var go in e.Parts) {
            Debug.LogFormat (this, "Object '{0}' is the result of a slice!", go.name);
        }
    }
}
```

On enable, we subscribe to the event; on disable, we unsubscribe.

When the object is sliced, our private method ("Sliceable_Sliced" in this example) is called, and given an array ("e.Parts" in this example) of shards.

We simply iterate through this array and do whatever we want with each GameObject.

Attachable Slicer

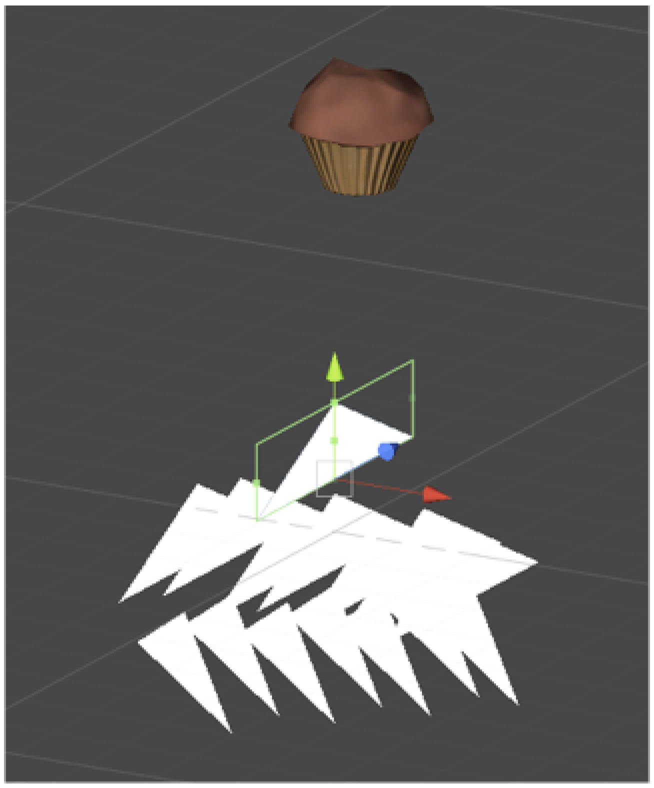
Slicer

The Slicer & Sliceable components are an easy way to make an in-game object cut another. There's two parts; the Slicer **prefab** and the Sliceable **component**.

Slicer

The Slicer prefab is a blade that cuts Sliceables.

You can rotate and size it, then place it in a hierarchy. You can move it at runtime. You might, for example, attach it to a sword. Remember that the white triangle will only appear in the editor; you do not need to manually hide it.



The prefab carries a few empty GameObjects for "Plane Definition." Do not remove them.

Sliceable

The Slicer will only automatically slice objects if they're configured for it. To do this, you must attach the Sliceable component to the object, with an additional specification; it must have

RigidBody and Collider components as well. (The Slicer employs both OnTriggerEnter and OnCollisionEnter and is thus subject to those constructs' requirements.)

If you want to suppress sliceability, your object controller can disable the Sliceable by setting the property **currentlySliceable** to false.

For example, an enemy that can take multiple hits might leave currentlySliceable off until the final hit, at which point it's flipped on. If this is done in an Update, than the Slicer (which uses LateUpdate) will observe this and perform the slice on the same frame (assuming you use the same collider to detect hits that the Slicer does).

The Slicer performs all operations in LateUpdate so it will observe these properties after you've set it without waiting until the next frame.

Performance

Background Slicing

On desktops and mobiles, Turbo Slicer will perform all slice computation on a background thread. This means that your game will continue to run while the slice is performed. You should be able to slice or even shatter a complex mesh with little impact on framerate.

Note that this feature is not supported on WebGL, because WebGL builds do not support threading.

In Detail

When you ask Turbo Slicer to slice an object, it must first pull the mesh data from Unity and prepare the "job." With a complex mesh this may take a few milliseconds, and must done on the Unity main thread, and therefore may impact your frame rate.

This first stage will take longer if you are slicing an object with a SkinnedMeshRenderer, because Turbo Slicer must ask Unity to "bake" the mesh, and must then marshal the snapshot into its own data structures.

See below about "preloading" and the "shreddable" option to learn about features that may affect how long this phase takes.

The second stage is the computation itself. Turbo Slicer must perform the slice, create the infill, generate new UVs and tangents and finally strip vestigial vertices from the two resulting meshes. Turbo Slicer will do this on a worker thread, and this work can therefore be done on a second core if multiple cores are available.

Most devices now feature multiple cores.

The third and final stage is to take the slice results and push them onto the Unity view graph. This final phase can also take a few milliseconds with a complex object, and depends primarily on *mesh complexity* and *hierarchy complexity.* This is because Turbo Slicer must ask Unity to push mesh data into the graphics engine, and must alter the scene graph.

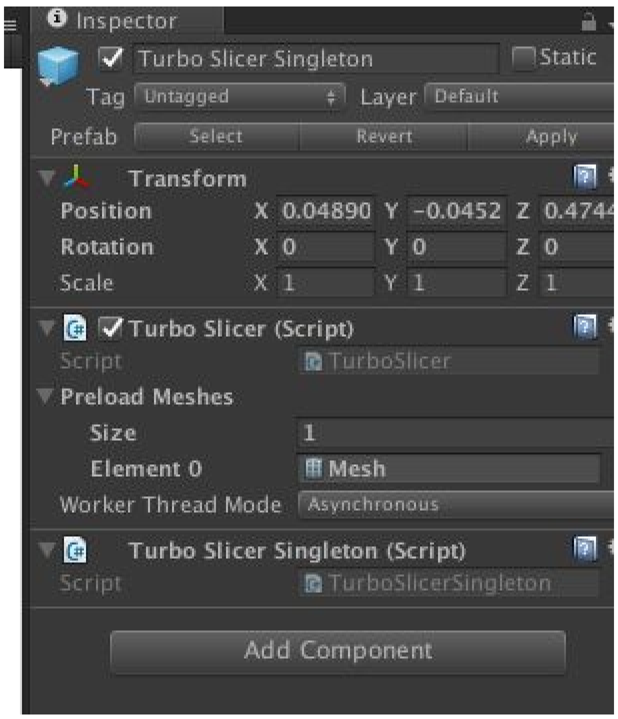
Preloading

You probably do not need this.

Before Turbo Slicer can slice a mesh, it must convert the mesh into its own internal working format. This may take a few milliseconds with a complex mesh.

In order to use this feature, you need to manually place the Turbo Slicer component into your scene.

Create a new GameObject. Then, add the "Turbo Slicer Singleton" component, like so:



Turbo Slicer Singleton ☐ Static
Tag Untagged Layer Default
Prefab

Transform
Position X 0.04890 Y -0.0452 Z 0.4744
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

✓ Turbo Slicer (Script)
Script TurboSlicer
Preload Meshes
Size 1
Element 0 Mesh
Worker Thread Mode Asynchronous

Turbo Slicer Singleton (Script)
Script TurboSlicerSingleton

You'll find the Turbo Slicer component has also been added automatically. Adding the Turbo Slicer Singleton component allows scripts to reach this instance via TurboSlicerSingleton.Instance.

Once this is done, you can add particular meshes to the "Preload Meshes" array. On start, Turbo Slicer will create its private data structures so that this does not need to be done on slice.

Shreddable

In order to configure an object for slicing, you must add the Sliceable component. Here you may notice an option called "Shreddable."

This option allows you to slice an object multiple times; or, rather, to slice the results of a prior slice. Turning it off will reduce memory usage but you will not be able to slice the results of prior slices.

This feature is turned on by default.

Work Avoidance

When given an object to slice, Turbo Slicer will process it on a *per vertex* basis to determine if it should be sliced, *even if* the slice plane comes nowhere near the object.

So for better performance, you should only ask Turbo Slicer to slice an object if you are reasonably certain that the slice plane actually intersects it.

Normals

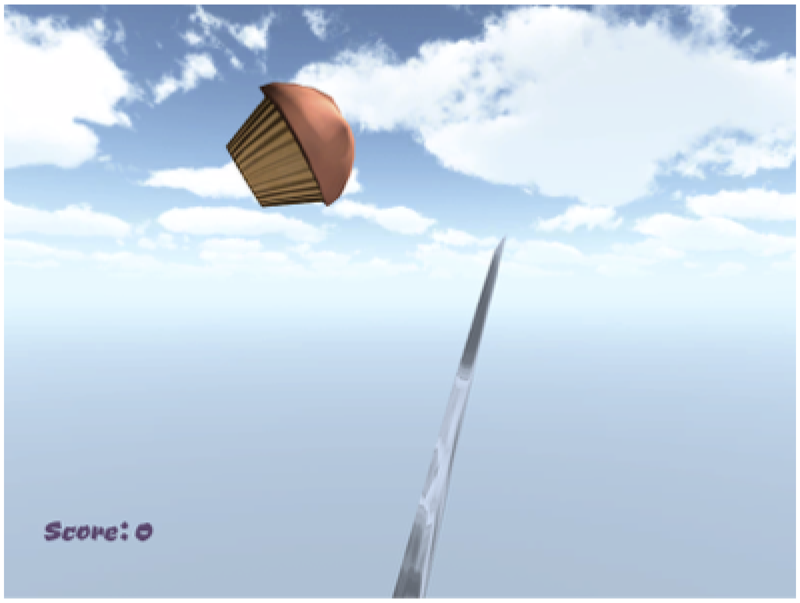
A given mesh has multiple data channels that need to be processed. Every data channel creates work, but not every one is needed. Specifically: **normals**, **tangents** and **UV2** can be toggled. Add a Sliceable component to an object to permit editing its configuration and use the check boxes.

Examples

We've included some examples, which can be found in "Noble Muffins / Turbo Slicer 2 / Examples.unitypackage".

Sliceable Demo

This can be found in Noble Demos / Sword Demo.



Muffins are flung up into view where the player, wielding a mouse controlled sword, can slice them. At its core, it is nothing more than a controllable sword with a Slicer attached to that sword. The muffins, which have Sliceable instances, can be cut by the Slicer.

In case you want to fork this demo, or use any of its content, we'll describe here its classes.

SwordTransformController is responsible for steering and turning the sword to chase the mouse cursor. It needs to know the relevant camera and the reach of the blade in order to transform mouse coordinates in screen space to a sensible-enough orientation in three space. The tracking time controls the speed at which the blade chases the cursor; lower is faster.

SwordVelocityFilter is responsible for answering a question: Is the sword moving fast enough to perform a slice? The length of the sword is given by the user because parsing the mesh isn’t reliable. The minimum tip speed for cutting is in units per second. It has one public boolean property: **IsFastEnoughToCut**. The next two components refer to this class and use this public property.

SwordSwishController observes SwordVelocityFilter and when it goes from too slow to fast enough, it will emit a swish using the adjacent AudioSource. It can be given a set of sound effects and play them at random.

AttachableSlicerController observes the SwordVelocityFilter and keeps a reference to the Slicer. It ensures that the Slicer is enabled or disabled depending on whether or not the sword is fast enough to slice.

Spawner throws objects upwards at the necessary speed to reach a given height (from the starting point) based on the current gravity.

Touch Slicer Demo

This demo is a rewrite of the older “Slice by Line” demo. It’s a partial implementation of a Fruit Ninja clone, but with “donut oranges” and has been rewritten from its earlier, messier form to mimic the decentralized, single-responsibility approach of the sword demo above. Let’s walk through its classes.

Touch Slicing

This is managed by two components; one on the targets responsible for identifying the targets to the slice core and one on an empty object responsible for calling for slices based on input.

Target is responsible for maintaining a list of all live targets on the screen. That’s all. The list is exposed through the static property TSD4Target.targets. This static property returns a read only list.

TargetSlicer is responsible for observing the mouse and touch inputs through Unity’s Input interface and slicing targets as appropriate. To work, it needs to know through what camera is the player looking at the targets (the public “camera” field).

If you have targets in the camera’s field of view, the TargetSlicer can be trusted to slice them when a finger or mouse cursor is dragged across that target.

It must be known that it does not use ray casting; rather it attempts to determine the target’s size on screen, and then performs a collision test in two-space. This gives it freedom to test multiple input samples on each target per frame without punishing lower end mobile CPUs.

Spawner throws objects upwards at the necessary speed to reach a given height (from the starting point) based on the current gravity.

Shared Scripts

These scripts are used by both demos.

SoundWhenSliced is a slice handler. It must be attached to a Sliceable, and causes a sound to play when the object to which it’s attached is sliced.

BurstApartWhenSliced is a is a slice handler responsible for applying explosive force to pop the slice results apart.

Trails Kit

Dropping the Trails Kit prefab into a scene activates the trails kit.

Important! You'll want to either move this somewhere far out of the way or use the layers and rendering settings to make it not interfere with other elements. Unfortunately we cannot export layers, so you will need to do this on your end.

Double-Sided Shaders

There are four simple shaders included which have visible interiors. Two of them support lighting while two do not, and two of them support separate interior, exterior textures while the others do not.

In Synergy Blade, we used the two-texture, unlit double-sided texture to paint the bosses; they would have a boss texture outside, with a "meat" texture inside (found on cgtextures.com).

This kit is used on the muffin material in the Sliceable demo.

Regarding SkinnedMeshRenderer

Turbo Slicer cannot slice a skinned mesh. If you ask it to do so, it will **bake** the mesh, and the slice results will use the non-animated MeshRenderer. Skinned meshes therefore cannot be preloaded and slicing them may have greater impact on frame rate.