# Mesh Slicer

## Content

# About the product

You can slice objects by plane synchronously/asynchronously. You call the Slice method and after a while, a callback method will be called.

Here you can find two folders: ObjectSlicer, CharacterSlicer, ObjectSlicerSamples and CharacterSlicerSamples. In your project, you can remove xxxSamples folders, because there are only examples of usage.

**You can try the free version of this package**. Check the reference to a free version on the product page.

If you have any problems, you can contact me. See contacts on my publisher page.

If you're interested, you can send me references to your game with this plugin and I'll add them to this documentation.

I would very much appreciate it if you would leave a review on the product page. )

# Performance

Test on quite a complex model (vertex count - 17004) takes ~ 55 milliseconds of time on my core i7 2.2GHz in a release.

# How does it work?

Each sliceable object in game have to implement 'IBzSliceable' interface:

```
/// <summary>
/// Asynchronously sliceable object
/// </summary>
public interface IBzSliceable
{
      /// <summary>
      /// Start slicing the object
      /// </summary>
      /// <param name="plane">Plane by which you are going to slice</param>
      /// <param name="callBack">Method that will be called when the slice
will be done</param>
      void Slice(Plane plane, Action<BzSliceTryResult> callBack);
}
```

The IBzSliceable interface is implemented in the abstract BzSliceableObjectBase class which you have to inherit. An example of an inherited class is ObjectSlicerSample.

How to slice:

1) Get sliceable component: var sliceable = GetComponent<IBzSliceable>();
2) Create plane by which you are going to slice: var plane = new Plane(...);
3) Slice: sliceable.Slice(plane,null);

# Base class BzSliceableBase

This is a base class. But you should not inherit from it directly. You need to use one of its derived classes BzSliceableObjectBase or BzSliceableCharacterBase.
Anyway, the base class provides you with some methods that you could override:

- StartWorker - a method that creates a worker thread. You may need it, for example, if you want to use your own thread pool.
- OnSliceFinishedWorkerThread - this method will be called in the worker thread after the work is done.
- GetNewObjects - control cloning of slicing objects.

And some base properties that you can use in the inspector:

- DefaultSliceMaterial - material that will be used on sliced faces
- Asynchronously - set this flag if you want that the object will be sliced asynchronously.
- UseLazyRunner - split processing to several frames. Increase performance.

# Implementations of BzSliceableObjectBase

In your class, you have to make your implementation of abstract BzSliceableObjectBase class. It requires you to implement these methods:

- protected abstract BzSliceTryData PrepareData(Plane plane) - Prepare data before slice
- protected abstract void OnSliceFinished(BzSliceTryResult result) - Will be called when the slice will be finished

## ObjectSlicerSample

For example, you can see sample implementation of ObjectSlicerSample class:

```
public class ObjectSlicerSample : BzSliceableObjectBase
{
	protected override BzSliceTryData PrepareData(Plane plane)
	{
		// remember some data. Later we could use it after the slice is done.
```

```
            // here I add Stopwatch object to see how much time it takes
            // and vertex count to display.
            ResultData addData = new ResultData();

            // count vertices
            var filters = GetComponentsInChildren<MeshFilter>();
            for (int i = 0; i < filters.Length; i++)
            {
                addData.vertexCount += filters[i].sharedMesh.vertexCount;
            }

            // remember start time
            addData.stopwatch = Stopwatch.StartNew();

            // colliders that will be participating in slicing
            var colliders = gameObject.GetComponentsInChildren<Collider>();

            // return data
            return new BzSliceTryData()
            {
                // componentManager: this class will manage components on sliced objects
                componentManager =
                    new StaticComponentManager(gameObject, plane, colliders),
                plane = plane,
                addData = addData,
            };
        }

        protected override void OnSliceFinished(BzSliceTryResult result)
        {
            // on sliced, get data that we saved in 'PrepareData' method
            var addData = (ResultData)result.addData;
            addData.stopwatch.Stop();
            drawText += gameObject.name +
                ". VertCount: " + addData.vertexCount.ToString() + ". ms: " +
                addData.stopwatch.ElapsedMilliseconds.ToString() + Environment.NewLine;
        }

        static string drawText = string.Empty;

        void OnGUI()
        {
            GUI.Label(new Rect(10, 10, 2000, 2000), drawText);
        }

        // DTO that we pass to the slicer and then receive back
        class ResultData
        {
            public int vertexCount;
```

```
        public Stopwatch stopwatch;
    }
}
```

## CharacterSlicerSampleFast (beta)

A simple implementation for skinned mesh slicer. Why I'm saying that it is beta, is because it has some downsides:

- Performance is not very good. So, as maximum, you can use it for low poly models.
- The number of slicing of the same model has to be limited to as max as 2.
- If the character was successfully sliced, no way it will continue its animation.
- Slicing will not break connections on each side even if visually nothing is connecting it.

So do not blame me for that, I'm trying to make it better )).

To be able to cut a character you have to:

- Add your character to the scene
- Setup ragdoll
- Add a CharacterSlicerSampleFast component
- Then you can call a Slice method on this component

# Slice objects with a knife

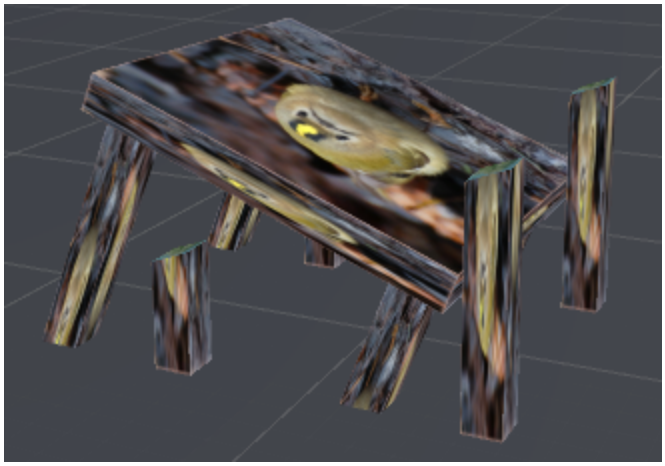
In the sample folder you and find two files "KnifeSliceableAsync" and "BzKnife":

- KnifeSliceableAsync.cs - for objects that you want to slice via knife (e.g. an apple);
- BzKnife.cs - for an object that you want to slice with (e.g. knife).

KnifeSliceableAsync must be attached to the objects with rigidbody on it.

## BzKnifeSliceable

A simple implementation that slices mesh.
Test scene: "BzKovSoft/ObjectSlicerSamples/testSceneAsync". If you click on the Table in the Game window, it will be sliced into two objects. And every time you click on it, it will be sliced again:



Important to note, the table is sliced only into two objects, so the legs at the bottom will be connected.

# BzSliceConfiguration

If your sliceable object contains multiple renderers, you can configure each differently by applying BzSliceConfiguration script to it.
Here you can configure slice material and slice type.
Slice types:

- Slice - object will be sliced
- Duplicate - the object will be duplicated after the slice
- KeepOne - object will not be sliced, and it stays only on one side of the plane

# Garbage Collector for falling objects

If you cut an object into very small parts, it sometimes happens that some very small parts fall through the ground and fall infinitely. These objects are useless and consume CPU time. To prevent this, use the component FallingObjGC.
This component should be only one per scene. So, attach it to some static empty game object. Set Min Pos Y to determine the minimum Y threshold. And if some objects with a BzSliceableBase component will be below this value, they will be destroyed.

# Extensions/Events

If you attach an object that implements an IBzObjectSlicedEvent interface, will be invoked after slicing:

```
public interface IBzObjectSlicedEvent
{
```

```
        void ObjectSliced(GameObject original, GameObject resultNeg, GameObject resultPos);
}
```

The method ObjectSliced will be invoked after successful slicing where it passes an old object and two new instantiated parts.

## BzFixMass

BzFixMass - is an example implementation of IBzObjectSlicedEvent where it fixes the weight and center of the mass of sliced objects.

## BzFixMassSmart

Correctly fixes the weight and center of the mass of sliced objects. It works correctly only for closed objects.

## BzReaplyForce

If you slice an object, it creates two new objects that do not inherit velocity and angularVelocity. This class fixes this problem.

## BzDeleteSecondJoint

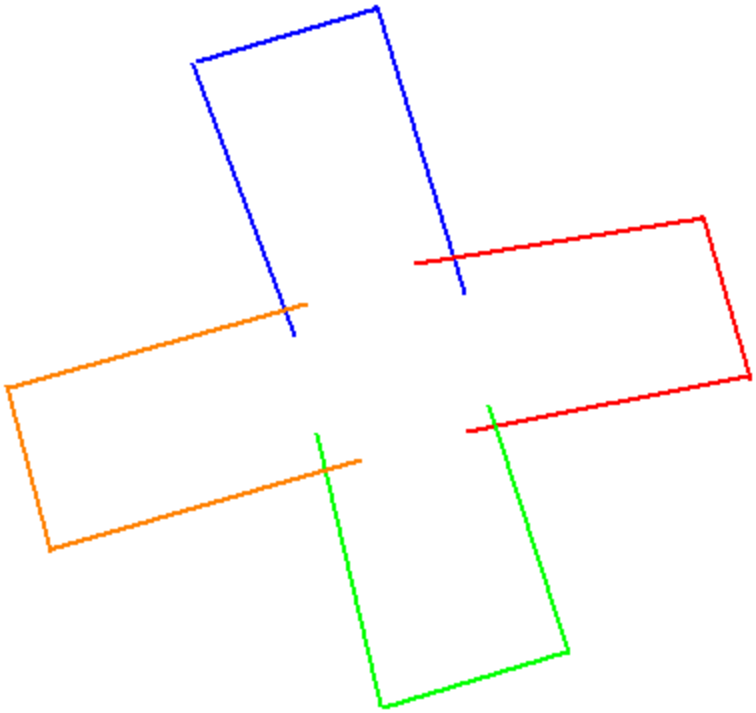
Suppose you slice an object that is attached to another object via Joint. But the two duplicated copies will both have their own Joint. The script deletes the Joint from the farthest object from the anchor.

# Problem zones (issues, limitations)

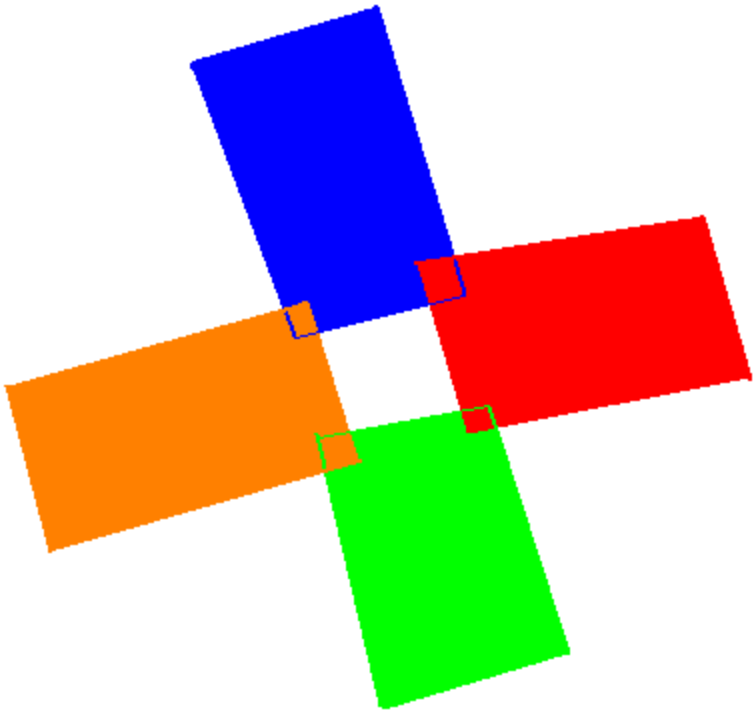
Of course, I'm working to get rid of problem zones. But they still exist. And there are some that I noticed:

## Different sequences

If section view looks like this:

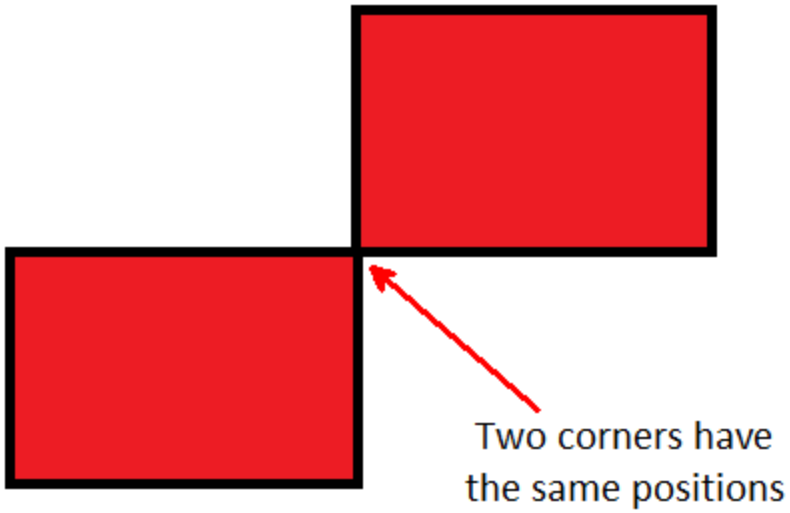


Each connected sequence of vertices will be joined to its own polygon. The result is 4 polygons, but the center will stay empty:



## Chessboard

In this case, the model could be closed incorrectly




## Flipped normals

If your model has back-flipped normals, it will close the sliced part incorrectly.

## Non-1 scale

The scale of each game object should be 1,1,1.

## First slice lag

This issue is related to JIT implementation. On the first slice, it has to compile the whole IL code to native machine code and it takes extra time.
To solve this issue I can suggest doing the first slice on scene load. For example, I added a component ObjectSlicerInitializer and CharacterSlicerInitializer that will do it for you. Just add it to the scene and that's it.

# FAQ

## Error: Not allowed to access vertices on mesh '...' (isReadable is false; Read/Write must be enabled in import settings)

Read/Write operations are disabled for your model. You can enable it in the model properties window: