Extended Primitives

Overview

Extended Primitives is an pack of parametric WYSIWYG 3d objects for prototyping, visual design and data visualization. All object has non-overlapped adjustable UV coordinates, parametric fillet edge, full control over detalization. Pack includes 3d objects:

- Box
- Cone
- Prism
- NGon
- Donut
- Pie
- Graph
- RadarChart.

Charts:

- Pie Chart
- Donut Chart
- Box Chart
- Cone Chart.

Numerical Grids (based on Line Renderer):

- Polar grid
- Quad grid.

Features

- Full procedural geometry
- Non-overlapped adjustable UV coordinates of 3 types (planar and two unfold mapping)
- Parametric fillet edges
- Full control over detalization (partial hidding, flip normals)
- All size parameters are animatable
- All 3d primitives has custom Raycast() methods which works without UnityEngine.Colliders

Latest Versions

- 1.0.1
  - Initial release
- 1.0.1
  - Improved Editor Pro Skin support
  - documentation reworked and converted to PDF

Installation

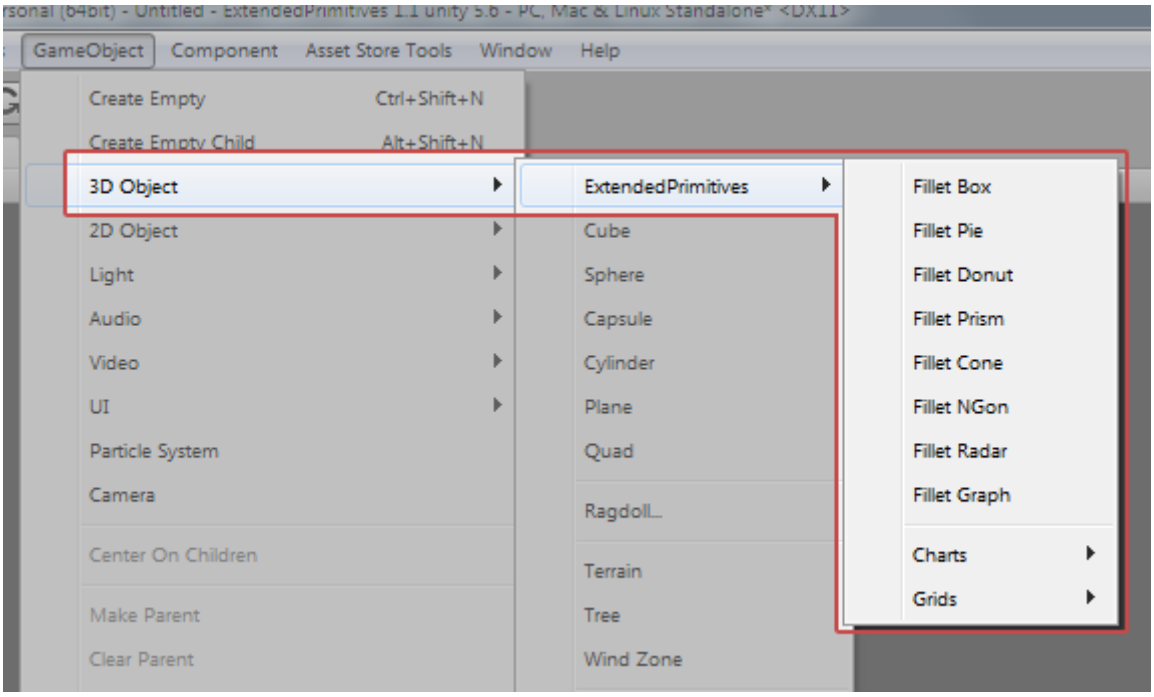
- import package

Compatibility

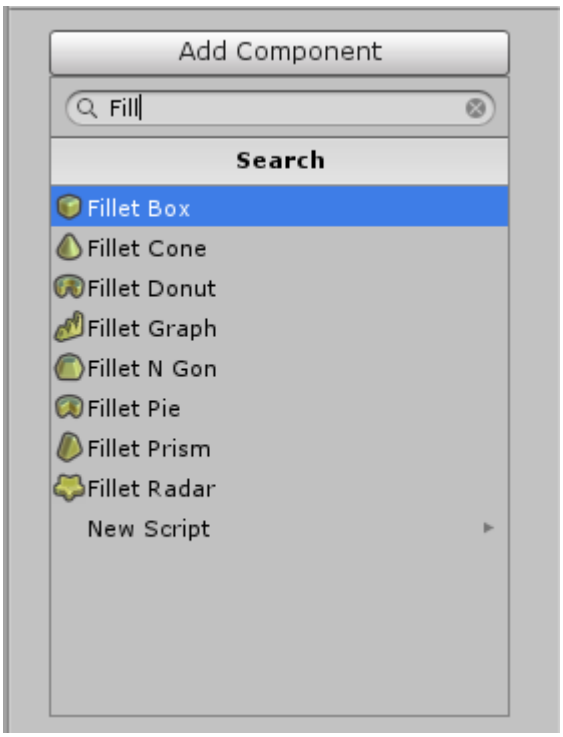
- Unity 5.6 or newer

Create primitives

- use menu *GameObject/3D Objects/Extended Primitves/*



- or add MonoBehaviour components *Assets/Plugins/ExtendedPrimitives 1.1/Scripts*

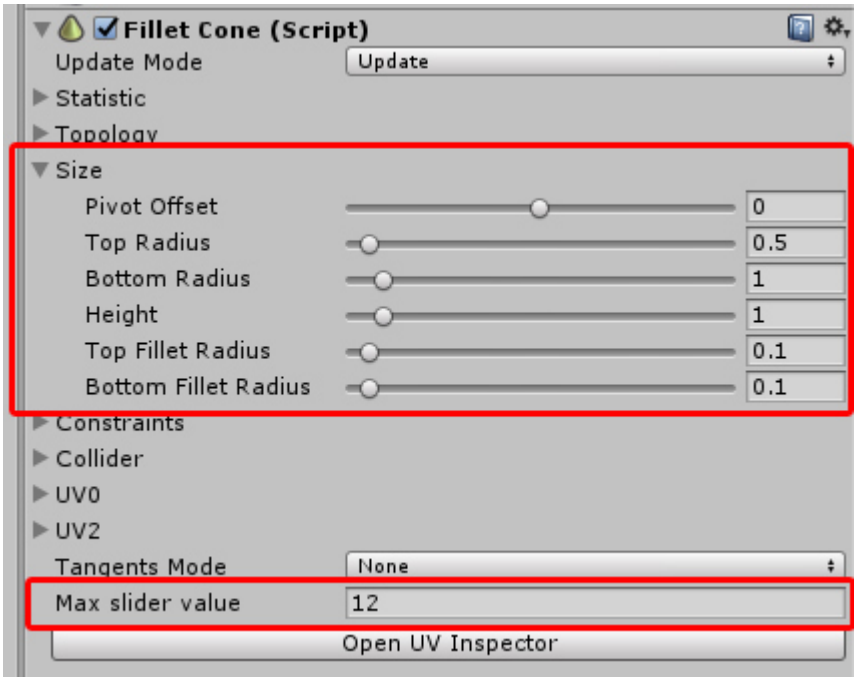


- or using static methods Create():

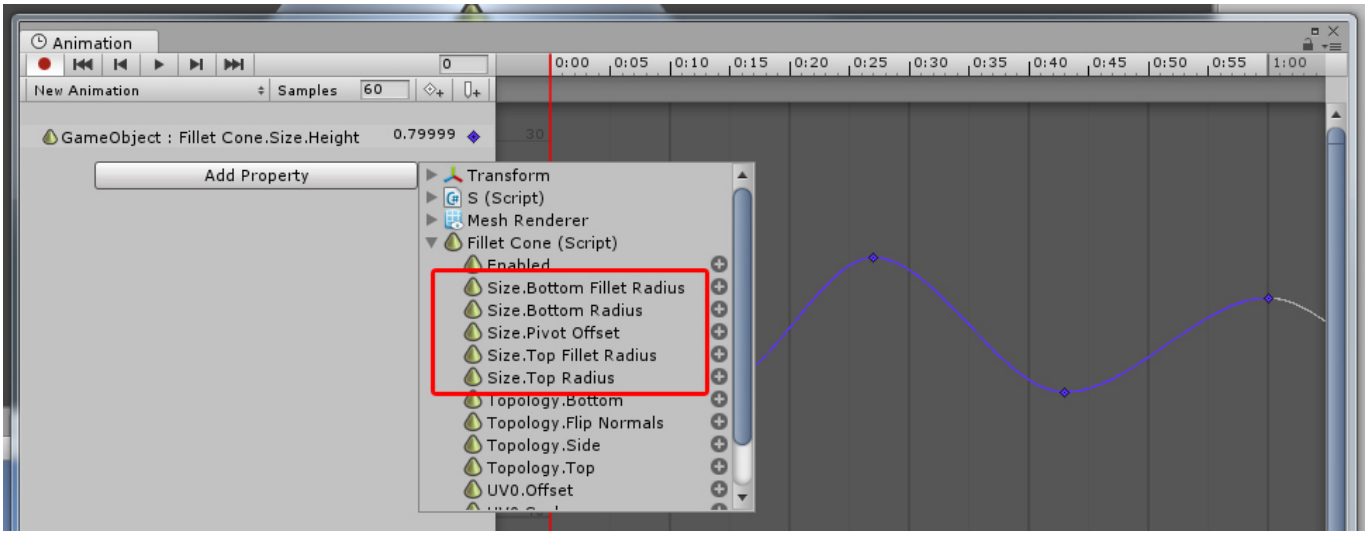
```
using UnityEngine;
using ExtendedPrimitives_11;


public class EpTutorial : MonoBehaviour {

    void Start () {
        FilletBox fb = FilletBox.Create();
    }
}
```

Modify size

to modify size use inspector



- or add animation properties to size fields



- or set Size struct by code

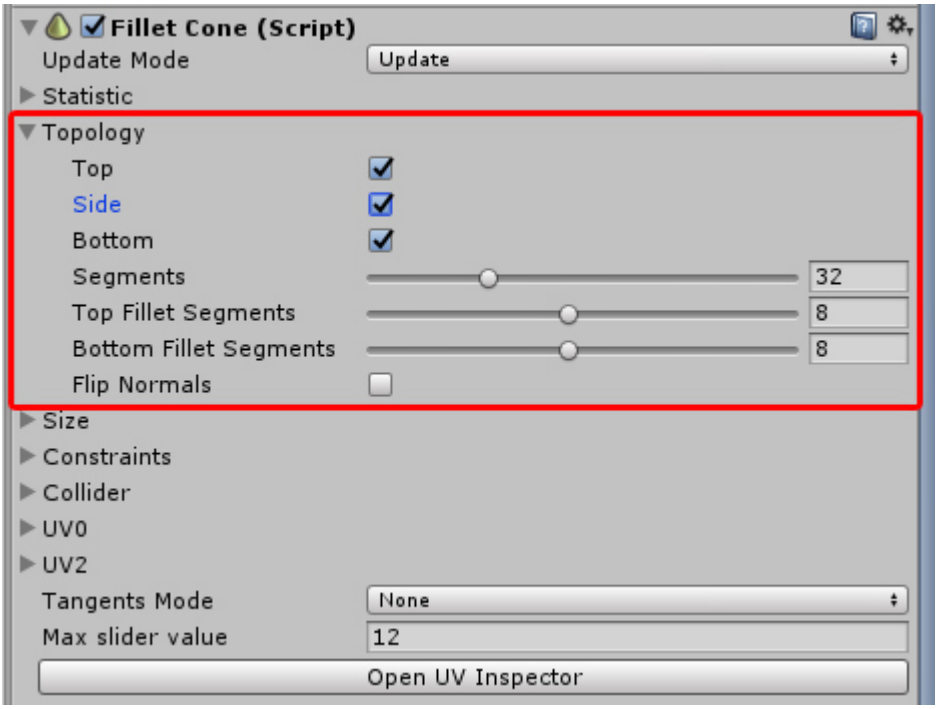
```
using UnityEngine;
using ExtendedPrimitives_11;

public class EpTutorial : MonoBehaviour {

   void Start ()
   {
       FilletBox fb = FilletBox.Create();
       fb.Size.Height = 10;
       fb.Size.Width = 7;
       fb.Size.Length = 5;
   }
}
```

Modify topology

Topology foldout describes Primitive`s detalization and partial surface hidding. Changing the topology leads to memory allocation, so avoid frequently change the topology in Play Mode.

To modify topology

- use inspector:



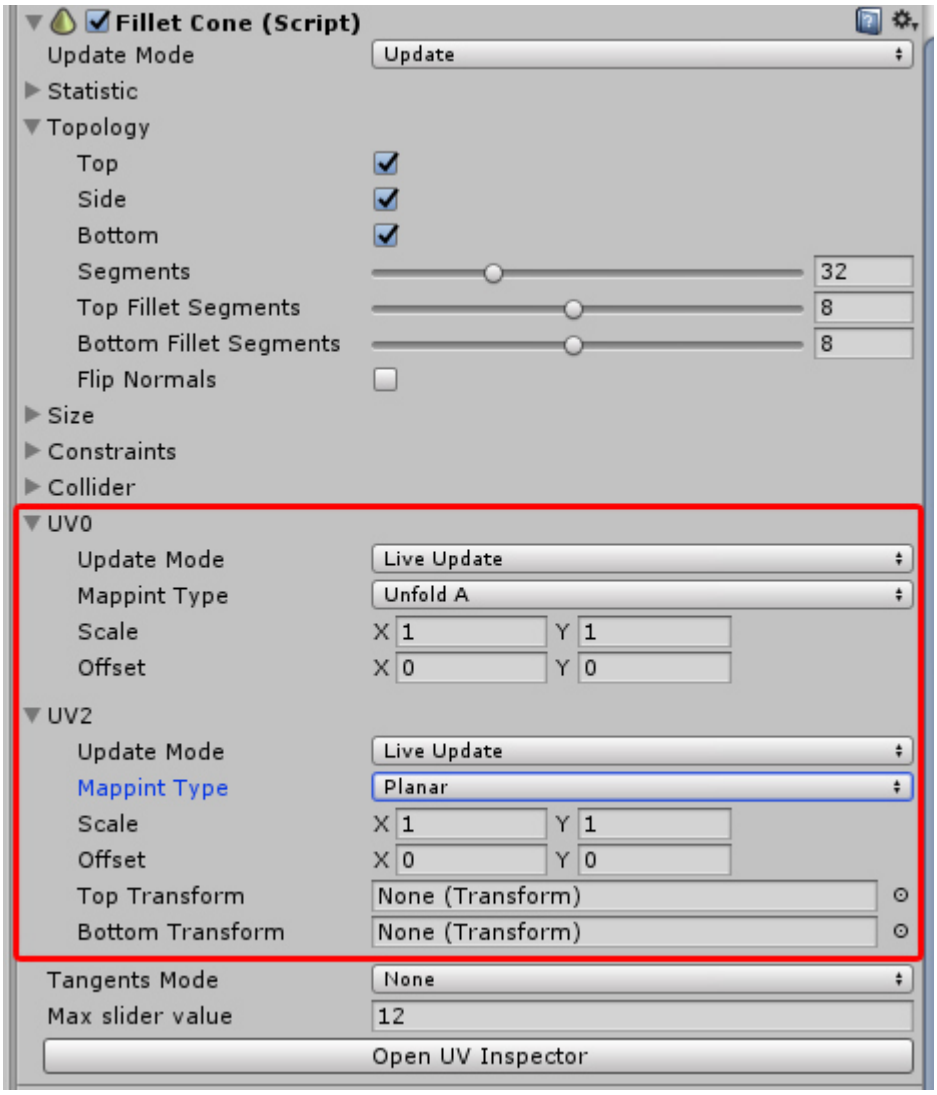
- or set Topology struct by code

```
using UnityEngine;
using ExtendedPrimitives_11;
```

```
public class EpTutorial : MonoBehaviour {

    void Start () {
        FilletBox fb = FilletBox.Create();
        fb.Topology.Bottom = false;
        fb.Topology.FilletSegments = 4;
    }
}
```

Modify Texure (UV) coordinates

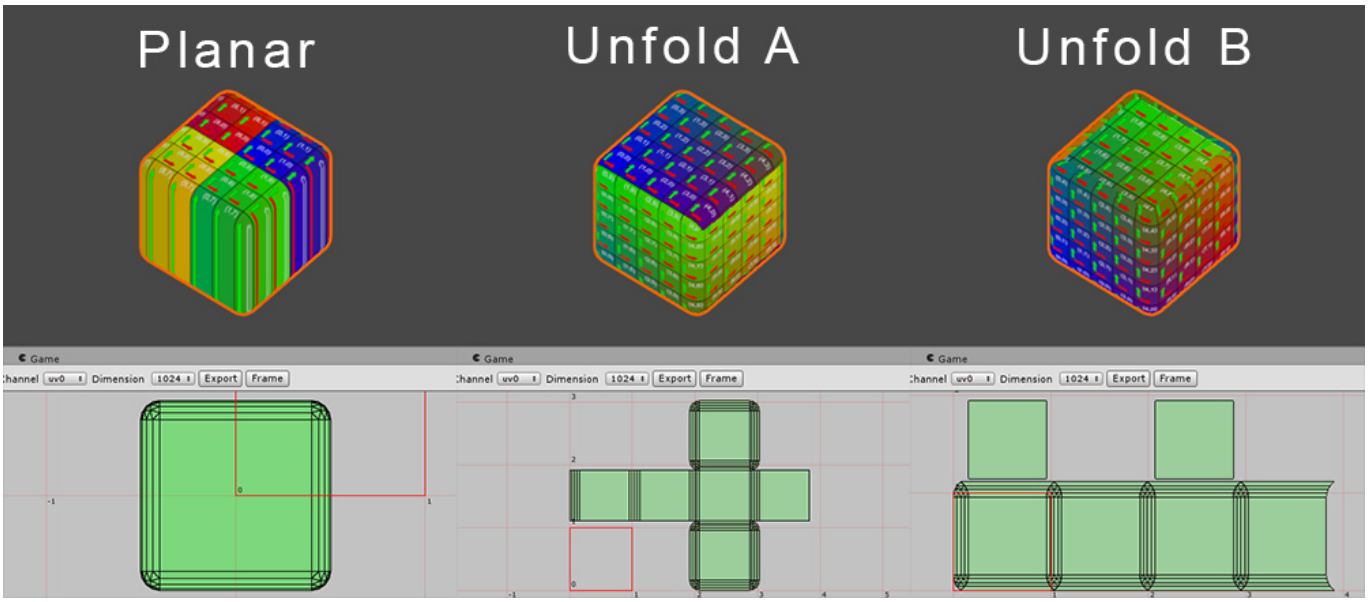
Each primitive can generate uv coordinates based on its own size



There are two UV channels that can be used, and 3 different types of UV generation.

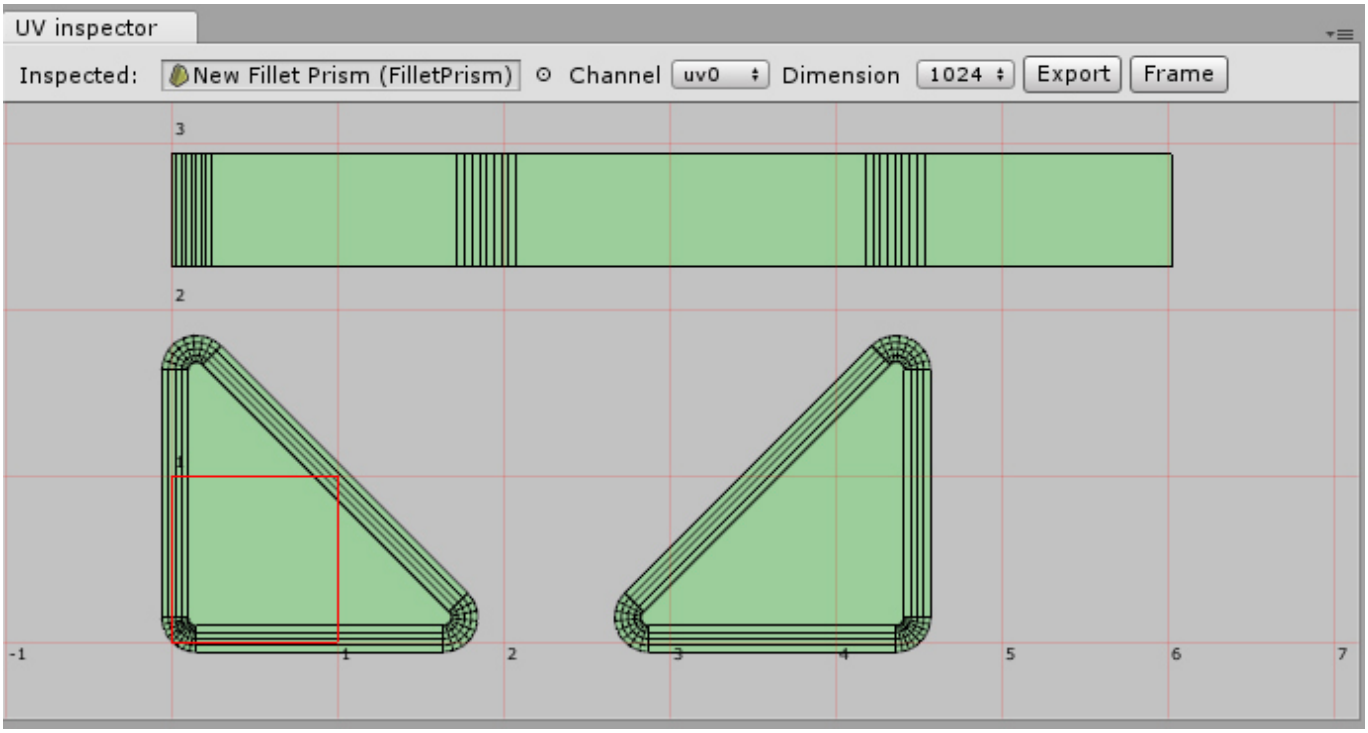
- Update Mode defines when UV will be recalculated
    - *None* disable UV
    - *Live Update* recalculates when Size, Topology UV properties is changed.
    - *Fixed* UVs will be recalculated once when object enabled. Will be used most recent Size and UV parameters wich cached before Fixed are enabled .

- Mapping type
    - *Planar* mapping projects UVs onto a Primitive through a plane. You can set any Transform as planar source. If planar Transforms is not defined used local object coordinates. PlanarMappingGizmo script is helpful to debug planar Transform position, scale and rotation.
    - *Unfold A* and *Unfold B* is methods of procedural mapping with minimal texture distortion. They differ where the seams are located.



UV inspector Window

UV inspector is an Editor window for preview of UV coordinates. Press *Open UV Inspector* button to inspect primitive`s UV-s.



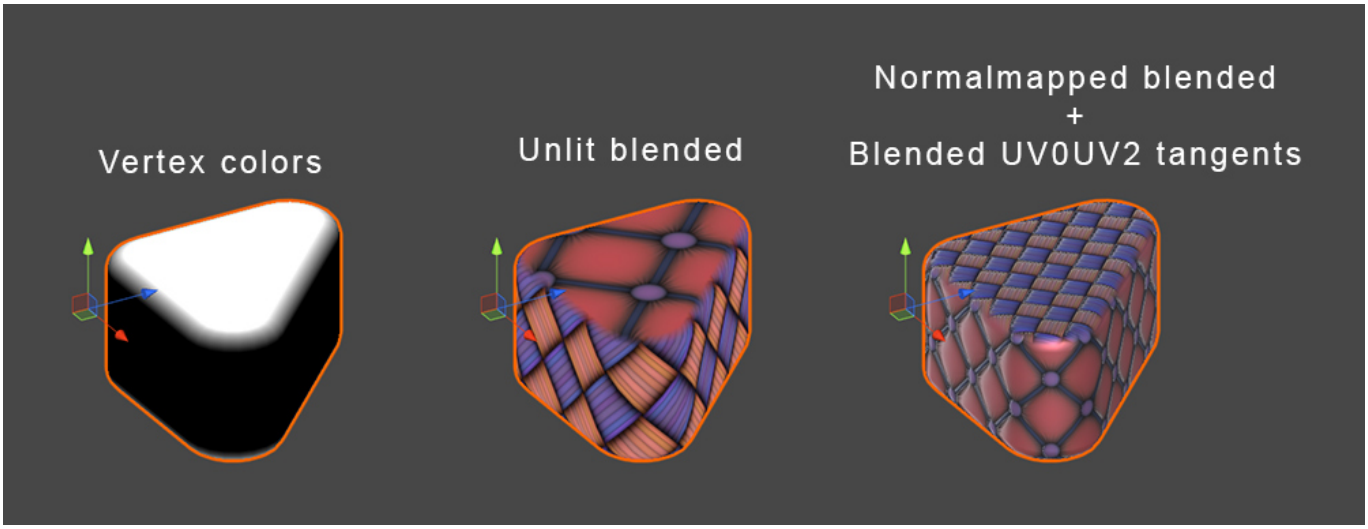
- *Inspected* - inspected Extended Primitive

- *Channel* - inspected UV channel
- *Export* - export UV layout to .png image with selected dimensions.
- *Alt + Right mouse button | mouse wheel* - zoom view.
- *Middle mouse button* - pan view
- *Left mouse button double click* - Frame view to UVs bounds

Tangents recalculation

Tangents are mostly used in bump-mapped Shaders. Tangents mode specifies how tangents will be recalculated. Recalculation is require UV coordinates at suitable channel.

- *None* - disables recalculation.
- *Builtin for UV0* - uses builin UnityEngine.Mesh.RecalculateTangents()
- *For UV0* - custom recalculation method based on UV0
- *For UV1* - custom recalculation method based on UV2
- *Blended UV0UV2* - If the Primitive uses two texture channels, this method allow to recalculate the blended tangents. Blending is based on the mesh`s vertex color alpha. Package include *EP_LitVertexColorBlend.shader* shader wich uses this feature. Please see example scene: *ExtendedPrimitives 1.1/(can be deleted) Examples/UVCoordinatesDemo/UVCoordinatesDemo.scene*



Raycast

Extended Primitive has its own scripted collider that allows you to perform Raycast on object without the using UnityEngine.Physics class or MeshCollider component. There is one parameter - Detalization based on visual detailization wich defined by Topology.

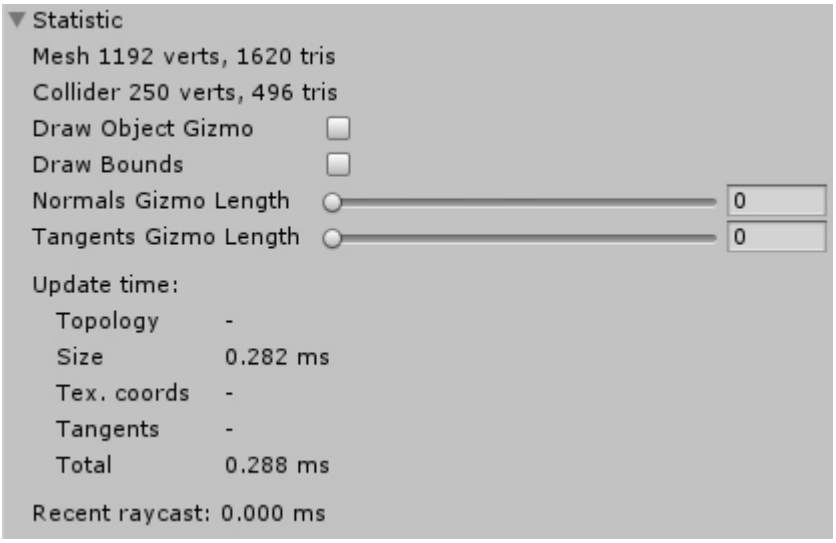
> Commonly, a One Half Detalization are enough to make an object picking using Raycast.

Create Mesh Collider button creates MeshCollider component with assigned mesh snapshot of current state of scripted collider.

```
using UnityEngine;
using ExtendedPrimitives_11;

public class EpTutorial : MonoBehaviour {

   void Start () {
       FilletBox fb = FilletBox.Create();
       Vector3 hitPoint = new Vector3();
       float distance = 0;
       Ray  ray = new Ray(Random.insideUnitSphere.normalized ,
Random.insideUnitSphere);
       if (fb.Collider.Raycast(ray, ref hitPoint, ref distance)) {
           Debug.LogFormat("ray {0} intersect {1} in point {2}", ray, fb.name,
hitPoint);
       }
   }
}
```

Statistic foldout

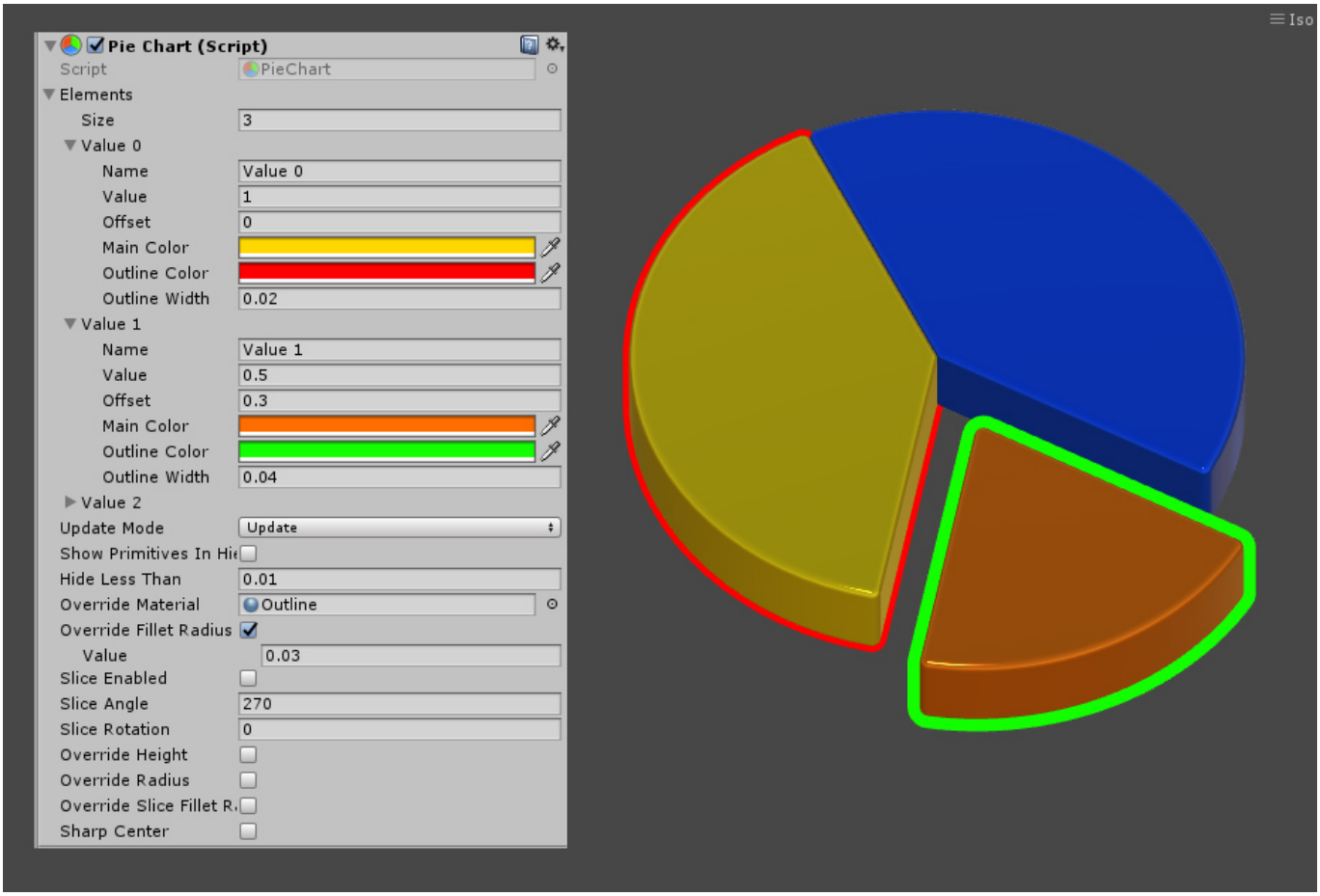
Statistic shows metric information about Mesh , Scripted collider, current update time in milliseconds, visualize mesh normals and tangents (if present)



Charts

Charts is a compound objects that handle multiple primitives at once. Charts elements is hidden by default. To display elements set toggle Display Primitives in Hierarchy. You can set child primitives

parameters separately using their inspector or using chart`s override properties.



To set element values use inspector or via code:

```
Chart.Elements[int elementIndex].Value = float
```

Raycast to Chart

To perform Raycast to Chart use

```
Chart.Raycast(Ray r, ref Vector hit, ref float hitDistance, ref int elementIdx);
```

Charts can manage material`s propertyes of their elements. To display outline use outlined shader. Package include two outlined shader

- Custom/ExtendedPrimitives/OutlineLit
- Custom/ExtendedPrimitives/OutlineUnlit

Grid

Grids is an compound objects based on LineRender and TextMesh renderer. To display Grid`s labels:

- enable it in inspector
- set up string format rules and text parameters then
- press *Apply Text* button.