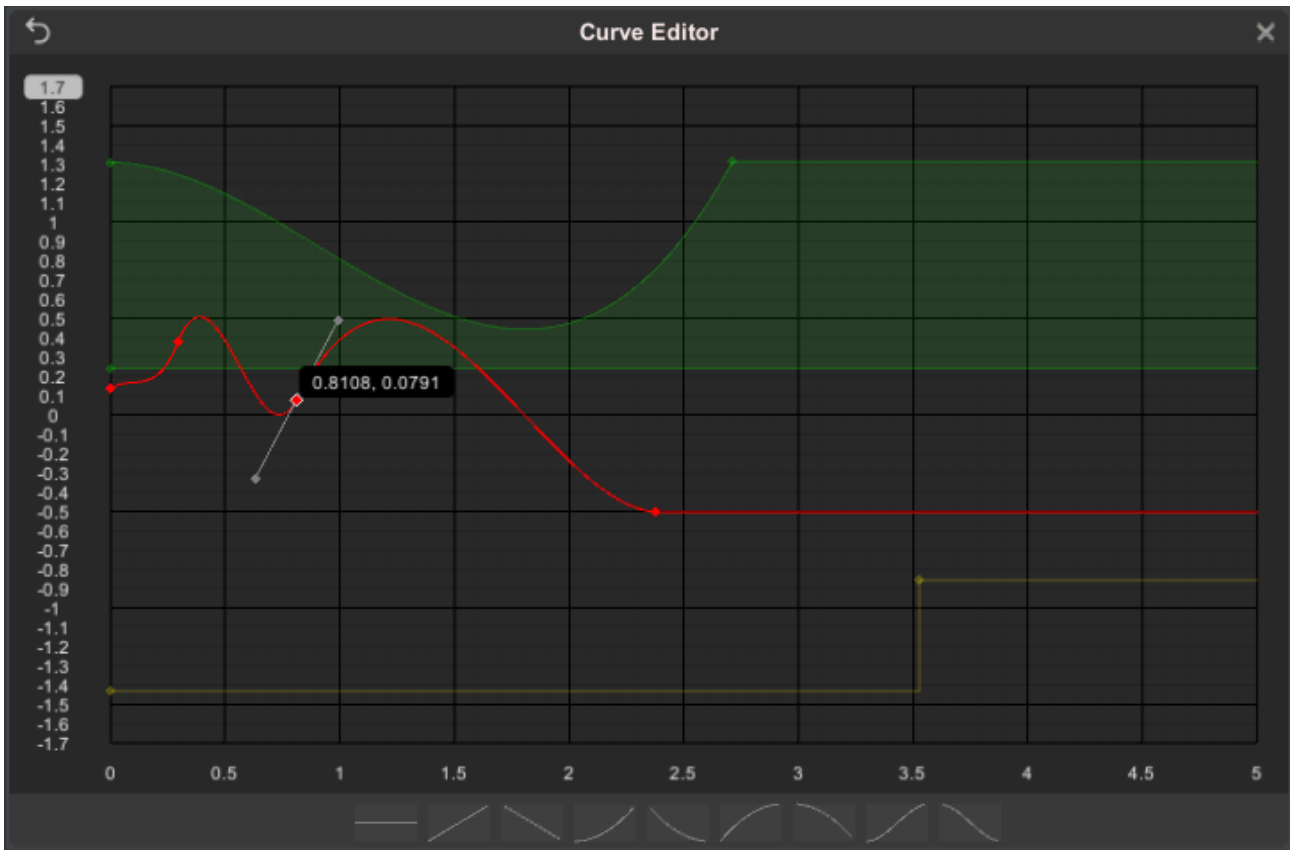


Runtime Curve Editor

The **runtime curve editor** can be used as part of any Unity application to visually edit, at runtime, an AnimationCurve object. It has almost identical functionality to the Unity's built-in curve editor.

In Unity Editor, as of 2025, there are four places where a curve editor is used:

- for editing AnimationClips
- for editing an **AnimationCurve** instance itself
- for Audio Source component
- for Particle System

It has to be noted, that this package, uses the AnimationCurve class, part of UnityEngine scripting API, to define the curves on the scripting side. Visually, this package was created to mimic as much as possible the curve editor used in Particle System.

The underlining curve equation used in this package is 100% identical with the equation used inside Unity engine itself.

This package has been tested on PC/Mac, WebGL, iOS and Android. It should work on any other platform without any flaws. The package doesn't need Unity Pro, it makes no use of any external library, and requires no other asset from the store. All the code is C#, available to the user for reading, understanding and/or modifying.

This package is not an Editor extension, it can be seen in action only in the play mode. If you didn't do it already, watch also the youtube demo video, its link should be visible in the description of this asset on Unity Asset Store.

The package is structured on 3 parts:

- **core functionality**: all the scripts under */Assets/RuntimeCurveEditor/RTAnimationCurve/Scripts/CurveEditor*
- **interface:** defined by */Assets/RuntimeCurveEditor/RTAnimationCurve/RTAnimationCurve.cs*
- **application**: defined by */Assets/RuntimeCurveEditor/Demo/Demo.cs*

This document describes, mainly the **interface** part(middle part), which is defined by *RTAnimationCurve.cs* class. This class is an interface for calling the runtime curve editor core module. The core module can be called directly, but calling through this interface should be the preferred way.

The provided methods/properties are:

```
public Rect GradRect;//gets/sets gradation range for the grid
public AnimationCurve ActiveCurve;
public Color ActiveCurveColor;

public void ShowCurveEditor();//pops up the window
public void CloseCurveEditor();//hides the window
public bool IsCurveEditorClosed();//true if the window is closed

public bool Add(ref AnimationCurve curve);//adds the curve to //the window, returns false only if the curve window is not yet initialized
public bool Add(ref AnimationCurve curve1, ref AnimationCurve curve2);//similar with the above version of Add but it adds a path of two curves
public void Remove(AnimationCurve curve);//removes the curve from //the window

public void SetGradYRange(float yMin, float yMax);//sets y axis's gradations range
public void SetGradXRange(float xMin, float xMax);//sets x axis's gradations range

public void SaveData(string name, Object obj);//saves the AnimationCurve fields related data, from 'obj', into the configuration with the given 'name' argument
public void LoadData(string name, Object obj);//loads the AnimationCurve fields related data, into 'obj', out of the configuration with the given 'name' argument

public void NewWindow();//removes all curves from editor and positions/resizes the window to initial.

public bool CurveVisible(AnimationCurve curve);//is the curve added to the editor
public bool CurvesVisible(AnimationCurve curve1, AnimationCurve curve2);//is the path of the two curves added to the editor

public bool DataAltered();//true, if any change in the editor has been made

public List<string> GetNamesList();//get the list of configurations saved so far
public void DeleteFile(string name);//deletes the named configuration
public string GetLastFile();//get the name of the last loaded configuration or null, if no configuration has ever been saved
```

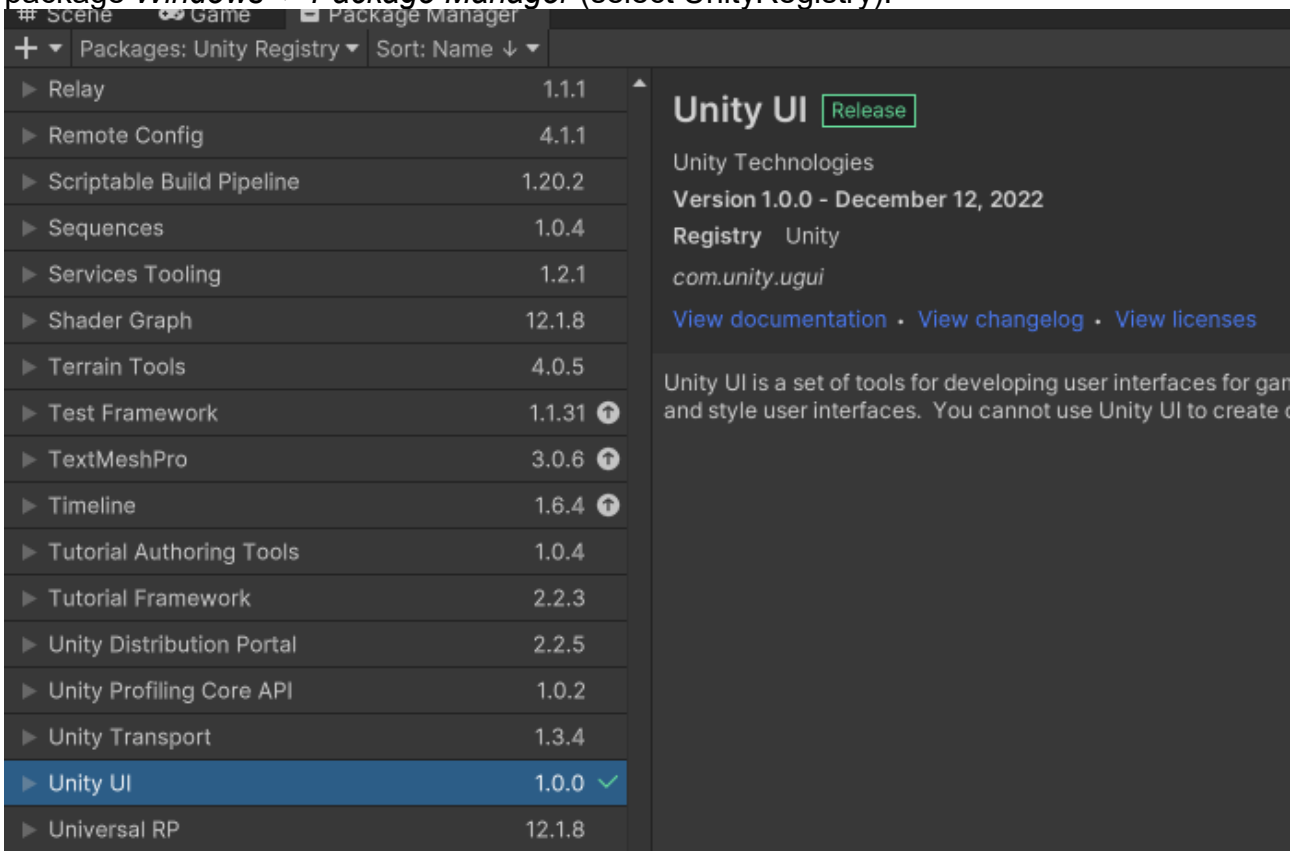
For understanding how to use this component, open up **DemoCurveUnity.scene** from /Assets/RuntimeCurveEditor/Scene and play the basic demo application, which allows you to edit two curves and a pair of curves as a path.

With this package is possible to save/load a configuration created by the user; a configuration is defined by all the data related to the curve fields of the class (**DemoAnimationCurves** in this case) of the object given as an argument when calling SaveData and the coordinates of the editor's window. The data related to the curve means both AnimationCurve data and the selections made on context menu (e.g. when right clicking on a key).

The gradation ranges tested are:

- 0 .. 5 for **x** axis (these hardcoded values are defined in CurveForm class)
- 0 .. 'any positive value' or any symmetrical range (e.g. -17 .. 17) for **y** axis

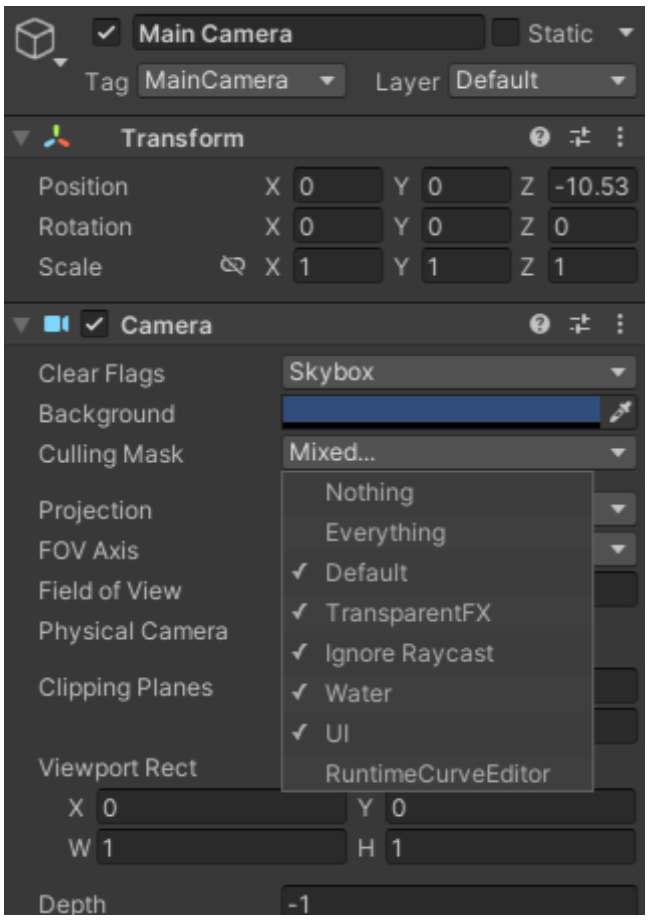
All the lines, are drawn as rendered meshes. While the texts, inputs, and panels are drawn using Unity.UI. For that you've got to assure that your Unity has installed this package *Windows* **->** *Package Manager* (select UnityRegistry):



The meshes (that keep the vertices for all the lines) and all the UI related, are rendered on a specially added layer 30 (named RuntimeCurveEditor). If you want to use a different layer number, change also the below line in CurveWindow.cs:

public static int MESH_LAYER = 30;

Keep in mind that this curve editor, uses an attached orthogonal camera, with transparent background (clear flags set to '*Don't Clear*') and culling mask set to the specially created 'RuntimeCurveEditor' layer (30). Normally this curve editor is going to be used in a scene with an existing main camera. The culling mask of this main camera has to disable 'RuntimeCurveEditor' layer (to avoid the re-rendering of this curve editor in that scene).



mailto:center@republicofhandball.com