Wrapping Rope v3.0.2

Information

Wrapping rope is a tool for creating thin rope stretched between two points. Wrapping rope does not provide realistic physics, but very useful if you want to show very long rope, that could bend, if it collides with other objects. For example, you can use this tool for creating superhero's grappling hook, wire for cable railway, lashing effect, bungee cord etc.

Key features:

- Works in editor
- Profile customizing
- Fore texturing modes
- The body of the rope could be rendered as finite segments or procedural mesh.
- Elastic and swinging physics

PLEASE NOTE that rope could not wrap any object marked as "static" in Inspector due to mesh inaccessibility. Also rope can not wrap itself or another rope. More information about limitations and conditions for wrapping see in Collisions section.

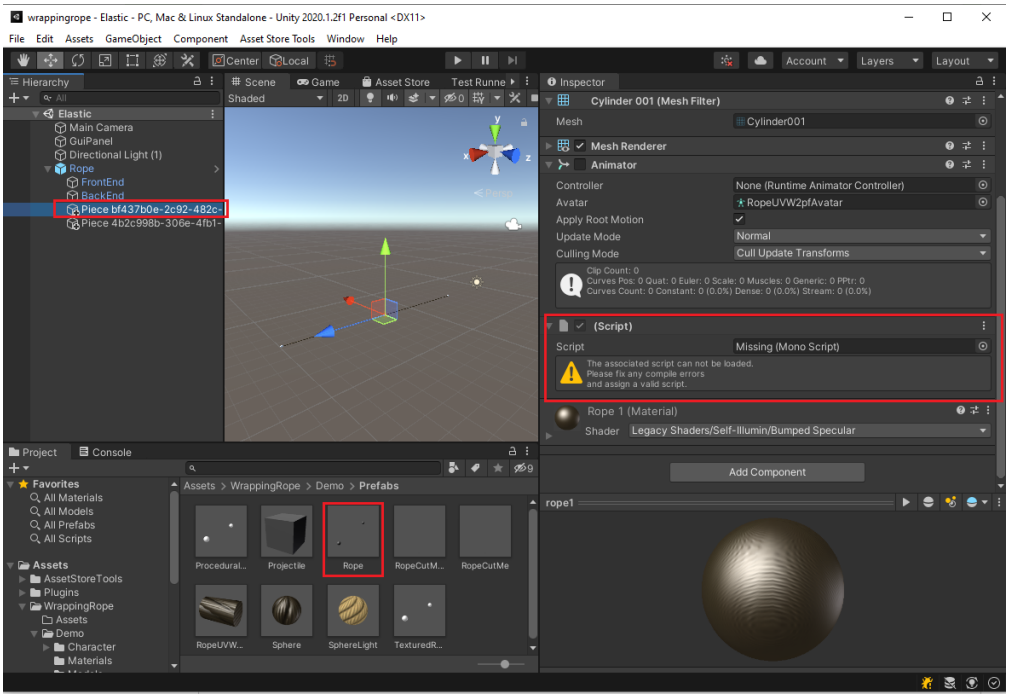
A short video instruction you can see at https://youtu.be/dmvq58QOM5g

Table of Contents

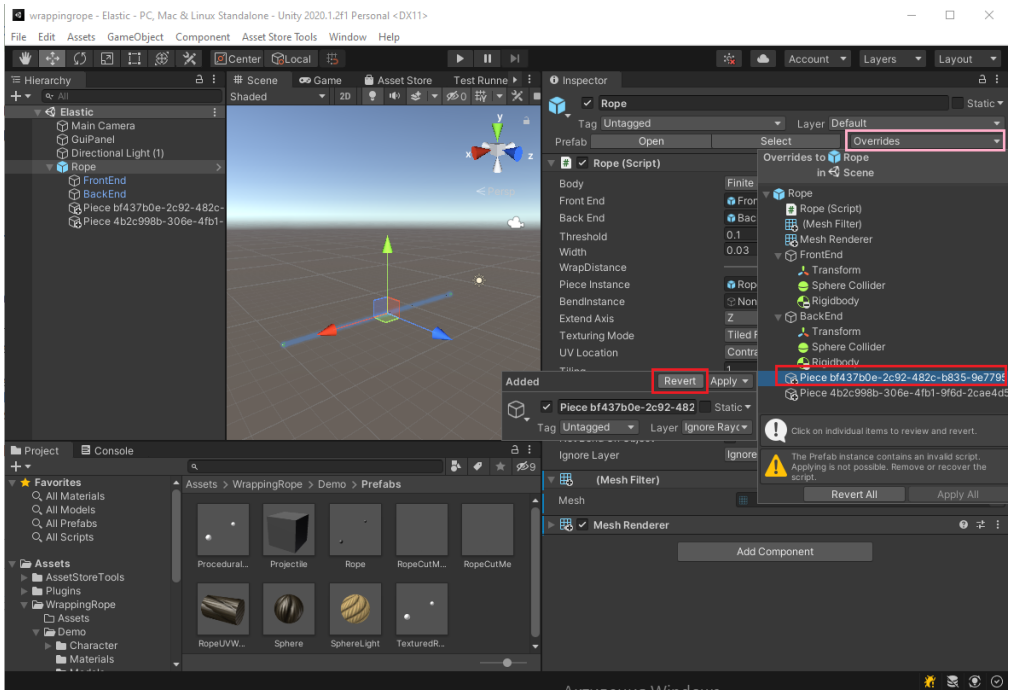
Migrating from 2.X versions

The main difference of this version from 2.X versions is cancellation of using .NET library and implementation all functionality in the scripts. These changes cause the issue «Missing script» for the references to the «MonoBehaviour» classes have been implemented in the .NET library. Generally, this issue related to the piece game object in three cases:

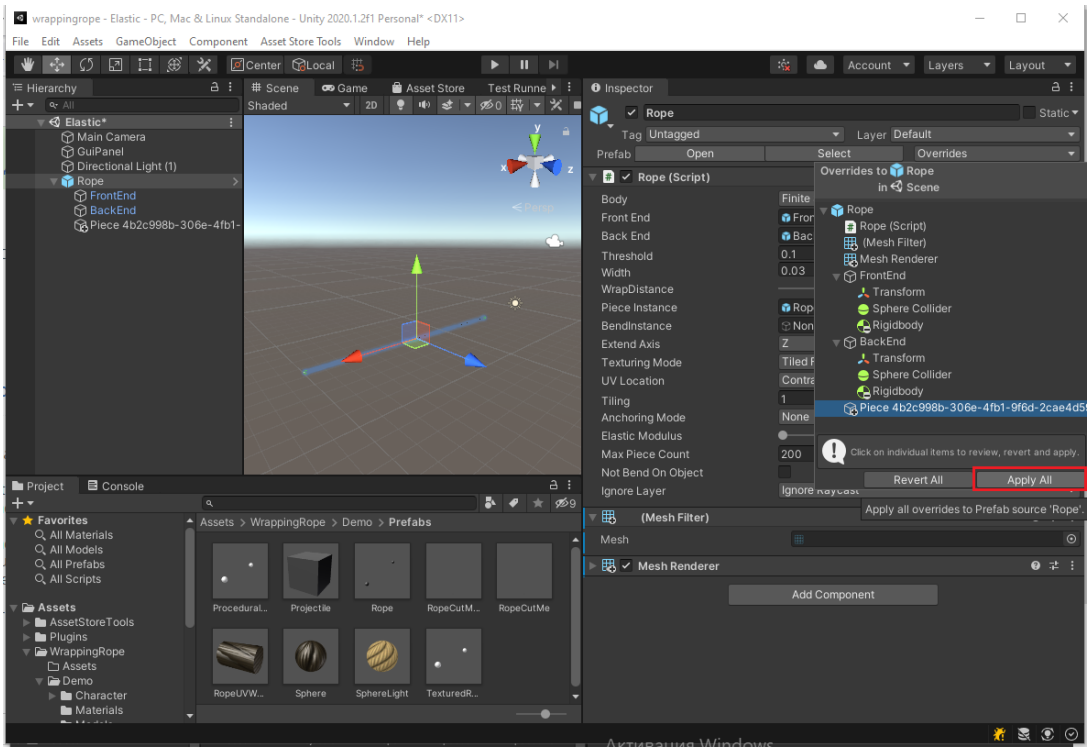
1. Game object is not a prefab. If the rope has only one piece game object, the rope automatically creates a new piece game object with correct reference to the script. All you need is delete the piece game object with missing script. If the rope wraps other object, it has more than one piece game objects. In this case you need to delete all piece game objects and wrap the object again.
2. Game object in the scene is prefab. Find the piece game object with missing script:



Select **Override** in the inspector. Find and select piece with missing script in the prefab. Then press **Revert** command:



After that the piece with missing script will disappear. Then press **Apply All** button:

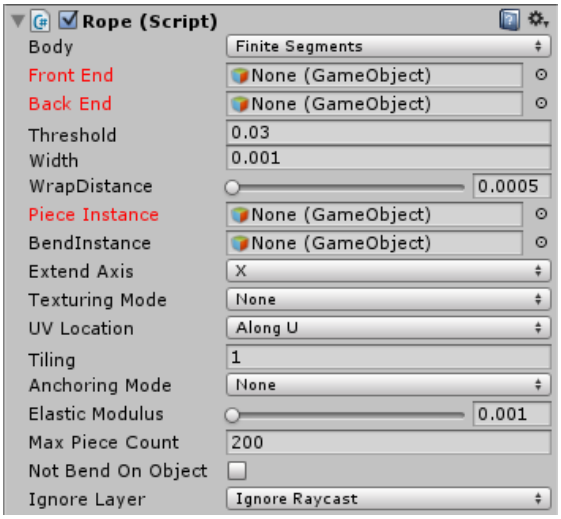


3. Prefab outside of scene. Open the prefab and follow the steps that described in the case when object is not a prefab.

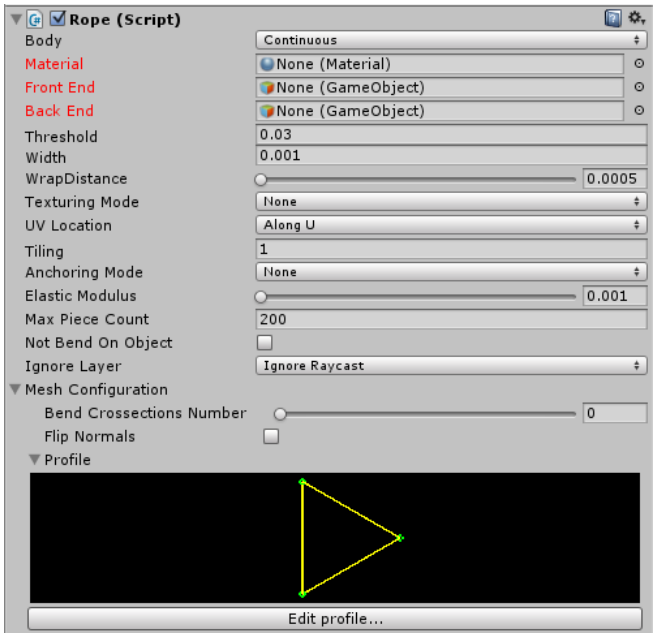
How to use

Creating of Rope

To create a rope use menu **GameObject/Wrapping Rope**. Depending on the **Body** property, the created rope can have two different sets of properties. Red color means required properties. Note, that created rope have Mesh Renderer and Mesh Filter components that not shown in figure.



Body set to Finite Segments



Body set to Continuous

Fig. 1 Inspector view

Name	Description
Body	The body type of the rope. This property can be one of two values (“Continuous” or “Finite Segments”). Setting this property defines a number of other properties in the component. More information about body types can be found in the Rope Body section.
Material	The material which is used when the **Body** property is set to **Continuous**.
Front End	The GameObject which the front end of the rope will attach to.
Back End	The GameObject which the back end of the rope will attach to.
Threshold	The Interpolation threshold for the rope. Making this smaller increases collision accuracy, at the cost of performance. The rope is processed during each fixed update frame. To check for potentially teleporting past intervening objects, the rope queries points between frames, at a distance of “Threshold”. (See Fig. 2).
Wrap Distance	The distance between the rope and the surface of the object it is wrapping. (See Fig. 3)
Piece Instance	The GameObject which will be used to render a segment of the rope. This GameObject must have a Mesh Filter and a Mesh Renderer.
Bend Instance	The GameObject which will be used to render the bends of the rope. This component can be used to smooth the transition between two piece instances. This GameObject must have a Mesh Filter and a Mesh Renderer. Note that instances will be scaled automatically based on the “Width” property of the rope, so the initial object scale must be 1 x 1 x 1.
Width	The Diameter of the rope.
Extend Axis	The direction that a piece instance will be elongated along (in its local coordinate system).
Texturing Mode	Specifies how the texture of the material should be stretched (see the Texturing section).

UV Location	Specifies how the texture for the material should be positioned (see the Texturing section).
Tiling	Specifies how the texture should tile.
Anchoring Mode	Specifies which end of the rope is anchored for processing swing physics (see the Anchoring section).
Elastic Modulus	Specifies the how stretchy/stiff the rope is. (see the Interaction with Game Objects section).
Max Piece Count	The maximum number of pieces in the rope. If a rope already has "Max Piece Count" pieces, any further bends will be ignored, and the rope will pass through the object.
Not Bend On Object	Set this to "On" to improve performance. This determines if a rope that is already wrapping an object can bend. If a rope is wrapping a convex hull it should not be able to bend anyway (because bending would push the rope into the object). Set this to "Off" if the rope needs to bend into concave objects. (Fig. 4).
Ignore Layer	Defines a set of layers which won't be processed for collisions with the rope.
Bend Crossections Number	The number of cross-sections between linear pieces. The higher the value of this property, the smoother the transition between linear pieces (see the Rope Body section).
Flip Normals	The direction of the normals in the procedurally generated mesh is dependent on the initial position of the ends of the rope. If the normals are defined incorrectly (which you will see because the rope will appear inside out) then checking this property will fix the normals.

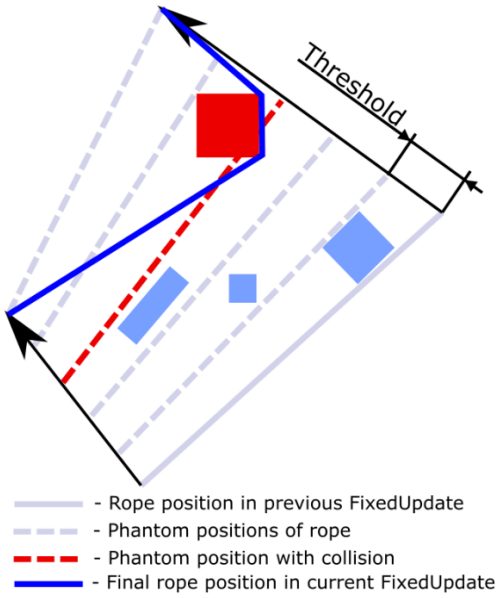


Fig. 2 The explanation of the Threshold

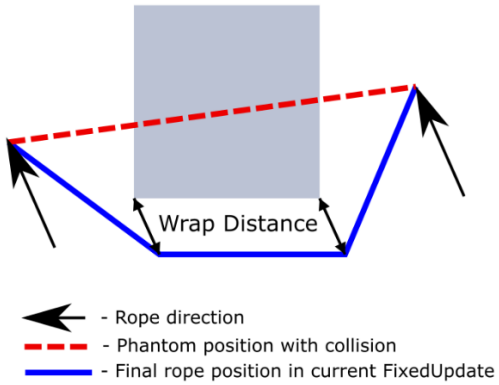


Fig. 3 The explanation of the Wrap Distance

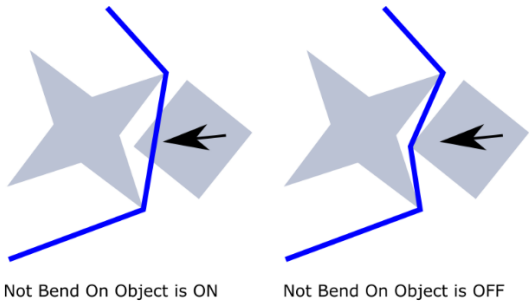


Fig. 4 The explanation of the Not Bend On Object

Collisions

Collision conditions and limitations

1) Rope is not able to collide (and wrap) itself or another Rope objects.
2) Static objects could not be processed in collisions.
3) If collision is detected with convex mesh collider and Rope is not intersecting object's mesh (for example pass a gap), collisions with objects behind the current object is not processed.
4) Object should have Collider.
5) Object should have Mesh Filter
6) Moving objects should have **Rigidbody** with unchecked **Is Kinematic** flag or have a script inherited from **IRopeInteraction** interface.
7) **Scale** property of game objects should have positive values.
8) Avoid clamping of rope between objects, which should be processed in collisions with rope.

Collision detection

Collision detection is based on Raycast. The Raycast always directed from front end to back end of Piece. Wrap direction is based on piece position in previous FixedUpdate, current piece position and velocities of objects which piece collides. Figure 5 shows the stages of collision detection and wrapping for one piece object.

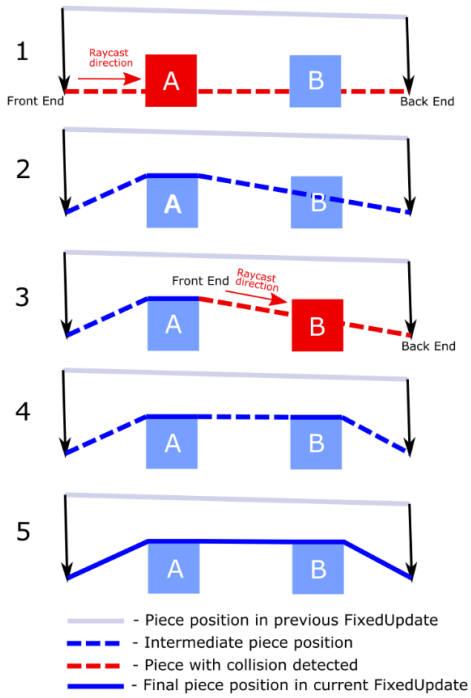


Fig. 5 Wrap stages

1. The piece tunnels boxes A and B. Raycast hits the box A.
2. The Piece wraps the box A. After wrapping there are new pieces at both sides of wrapped object. For each of this pieces the collision will be checked.
3. The piece tunnels the box B. Raycast hits the box B.
4. The piece wraps the box B.
5. All pieces positions are accepted.

Texturing

Texturing Mode

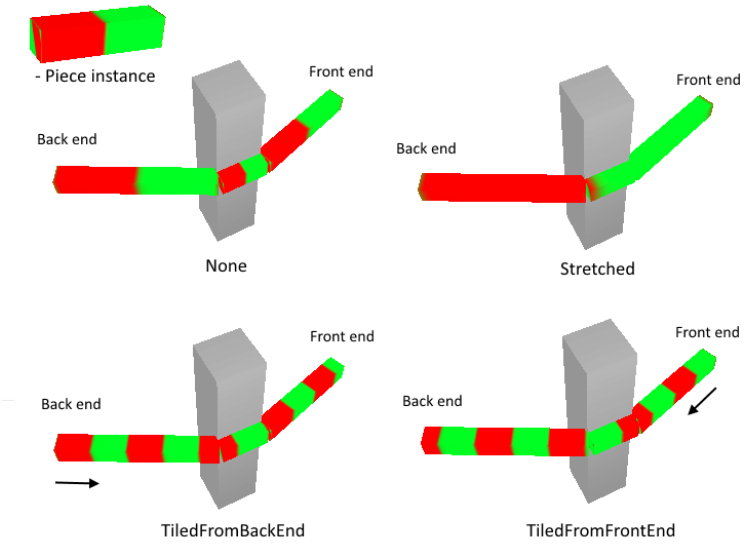


Fig. 6 Texturing Mode options

- **None** - texture not changed
- **Stretched** - texture stretches between back end (specified by **Back End** property) and front end (specified by **Front End** property)
- **TiledFromBackEnd** - texture anchored to back end and tiled along the rope
- **TiledFromFrontEnd** - texture anchored to front end and tiled along the rope

UVLocation

Texturing algorithm can't define how texture mapped along extend axis (specified by **Extend Axis** property), so this property used for solving this problem. Using texturing, remember, that extend axis always directed from back end to front end.

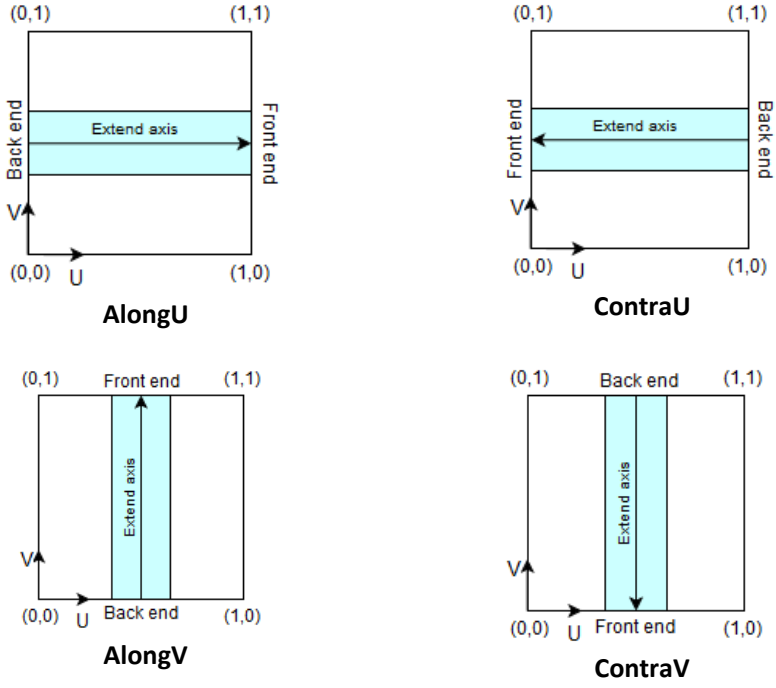


Fig. 7 UVLocation options

Texturing when Body set to Continuous

When **Body** set to **Continuous** (see the Rope Body section) **UVLocation** controls how the texture of **Material** will be projected to the rope mesh. In this case **AlongU** and **ContraU** means that U axis of texture is projected along rope length, while **AlongV** and **ContraV** means that V axis of texture is projected along rope length.

Anchoring

This feature is useful for imitation of swinging physics. Swinging object should have Rigidbody with unchecked **Is Kinematic** flag and should be assign to **Front End** or **Back End** property. To control swinging physics use **Anchoring Mode** property.

- **None** - anchoring feature is off
- **By Front End** - The rope end, specified by **Front End** property, will be fixed in space, while other end of the rope will be suspended. In this case, **CutRope** method call will result in movement of the Back End to Front End.
- **By Back End** - The rope end, specified by Back End property, will be fixed in space, while other end of the rope will be suspended. In this case, **CutRope** method call will result in movement of the Front End to Back End.

Interaction with Game Objects

Interaction with game objects is possible when collision is properly processed (see the Collisions section). There are three cases of collision of the rope with game objects.

1) The rope is moving and the game object is not moving.
2) The rope is not moving and the game object is moving.
3) The rope is moving and the game object is moving.

The first case requires that the game object has a collider. The second and the third cases require that the game object has **Rigidbody** with unchecked **Is Kinematic** flag or has a script inherited from **IRopeInteraction** interface (see IRopeInteraction interface section). When game objects are moved by physics simulation in a game the **Rigidbody** is sufficient for the second and the third cases. When game objects are animated by script, character animation or dragged by mouse in editor the **Rigidbody** is useless while **IRopeInteraction** works fine. The asset has two examples of classes that inherited from **IRopeInteraction** interface. The first class **DefaultRopeInteraction** could be used as a script component or as a base class for other classes with some custom logic. The second class **CharacterRopeInteraction** is used for interaction with a biped animated character in a demonstration scene. One important condition to use these classes is that a rope script should be executed before these classes.

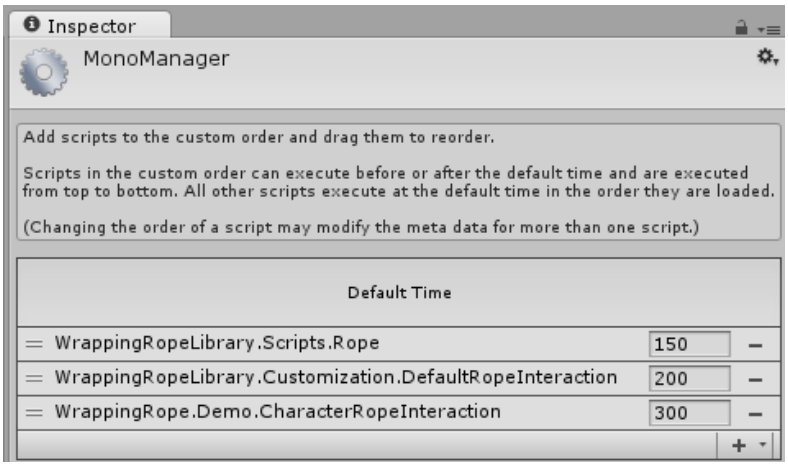
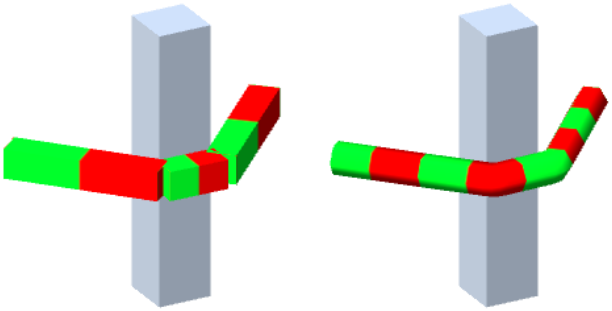


Fig. 8 Script execution order

When collision takes place, the rope could interact with object by applying a force. This feature is available in the same conditions as for collision: a game object should have **Rigidbody** with unchecked **Is Kinematic** flag

or a script inherited from **IRopeInteraction**. The degree of interaction is specified by **Elastic Modulus** property. The less value of this property, the less force for wrapped object.

Rope Body



Finite Segments Continuous

Fig. 9 Body types

To select the body type use **Body** property, which provides two values:

- **Finite Segments** – connections of linear instances of object, assigned to **Piece Instance** property
- **Continuous** – procedural mesh

To configure procedural mesh use **Mesh Configuration** section (see Creating of Rope section).

Polygon Editor

To edit profile of rope when **Body** property set to **Continuous** use Polygon Editor. To open Polygon Editor use the button below profile preview in inspector or menu **Window/Polygon Editor**.

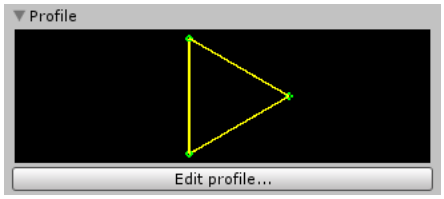


Fig. 10 Profile preview

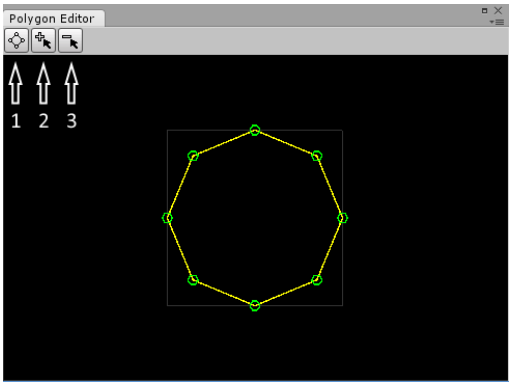


Fig. 11 Polygon Editor

- 1 – make regular polygon
- 2 – insert points
- 3 – delete points

The grey square limits an area with size of 1x1 Unity's units. To zoom use mouse wheel. Note, that minimal count of points is three. The polygon couldn't be self-intersecting.

API

Rope script

Namespace: `WrappingRopeLibrary.Scripts`

Public Fields

Minimal size of objects, that reliably will be processed in collisions with rope.
```
public float Threshold
```

Distance between object surface and rope in wrap zone.
```
public float WrapDistance
```

The direction of piece instance expansion (in local coordinate system).
```
public Axis ExtendAxis
```

Specifies how texture of material should be expand.
```
public TexturingMode TexturingMode
```

Specifies how texture of material should be placed.
```
public UVLocation UVLocation
```

Specifies the degree of elastic physics.
```
public float ElasticModulus
```

Texture tiling.
```
public float Tiling
```

Maximal number of pieces.
```
public int MaxPieceCount
```

If «On» then If two ends of piece are belonging to one object then piece can't wrap anything.
```
public bool NotBendOnObject
```

Public Properties

Gets a body type of the rope.
```
public BodyType Body { get; }
```

Gets a material used when Body property set to Continuous.
```
public Material Material { get; }
```

Gets a game object used for position of the front end of the rope.
```
public GameObject FrontEnd { get; }
```

Gets a game object used for position of the back end of the rope.
```
public GameObject BackEnd { get; }
```

Gets or sets a width of the rope.
```
public float Width { get; set; }
```

Gets or sets an anchored end of the rope for swinging physics.
```
public AnchoringMode AnchoringMode { get; set; }
```

Gets a first piece in the beginning of the rope.
```
public Piece FrontPiece { get; }
```

Gets a last piece of the rope.
```
public Piece BackPiece { get; }
```

Gets a scale of PieceInstance to the width of the rope by two dimensions and 1 unit by third dimension depending on ExtendAxis.
```
public Vector3 PieceInstanceRatio { get; }
```

Gets a uniform scale of BendInstance to the width of the rope. Scale factor is defined by x axis of bounds of BendInstance.

```
public float BendInstanceRatio { get; }
```

Gets a number of cross sections between linear pieces.

```
public int BendCrossectionsNumber { get; }
```

Public Methods

Changes the length of the rope. If value of AnchoringMode property is AnchoringMode.None, the length of the rope will be only decreased, otherwise the length of the rope can be whether decreased or increased depending on the sign of the length parameter.

```
public void CutRope(float length, Direction dir)
```

Shrinks the length of the rope.

```
public void CutRopeNotAnchoring(float length, Direction dir = Direction.BackToFront)
```

Sets the length of the rope. This method will work when the AnchoringMode property is not set to the AnchoringMode.None.

```
public void SetRopeLength(float length)
```

Gets the current length of the rope. Be careful: this method iterates through all pieces of the rope.

```
public float GetRopeLength()
```

Events

Triggered when the rope is about to wrap a game object

```
public event ObjectWrapEventHandler ObjectWrap
```

Enums

Namespace: `WrappingRopeLibrary.Enums`

```
public enum Direction
{
    FrontToBack,
    BackToFront
}

public enum BodyType
{
    FiniteSegments,
    Continuous
}

public enum Axis
{
    X, Y, Z
}

public enum TexturingMode
{
    None,
    Stretched,
    TiledFromBackEnd,
    TiledFromFrontEnd
}
```

```
public enum UVLocation
{
    AlongU,
    ContraU,
    AlongV,
    ContraV
}

public enum AnchoringMode
{
    None,
    ByFrontEnd,
    ByBackEnd
}
```

Events

Namespace: `WrappingRopeLibrary.Events`

```
public delegate void ObjectWrapEventHandler(RopeBase sender, ObjectWrapEventArgs args)
```

Public Properties of ObjectWrapEventArgs Class

Gets a game object that the rope is about to wrap.

```
public GameObject Target { get; }
```

Gets an array of points in space that specifies the wrap path.

```
public Vector3[] WrapPoints { get; }
```

Gets or sets a value indicating whether the event should be canceled.

```
public bool Cancel { get; set; }
```

IRopeInteraction interface

Namespace: `WrappingRopeLibrary.Customization`

Public Methods

Gets the velocity of the game object at the point worldPoint in global space.

```
Vector3 GetPointVelocity(Vector3 worldPoint);
```

Applies force at position.

```
void AddForceAtPosition(Vector3 force, Vector3 position, ForceMode mode)
```

Support

If you have any questions or there are any problems, please email me at: dkirillovd@mail.ru – Denis