Rope Minikit

version 2.1.0

The Rope Minikit brings stable rope physics to your project. The rope component can be used to simulate simple wires or more advanced setups with pulleys and weights that require active collision detection. The bundled example scene shows how to connect the rope for a physically simulated crane, a rope bridge and set of swings. The rope component is written with performance in mind and many compute intensive tasks are handled by Unity jobs on separate threads accelerated using the Burst compiler.

Contents

About the Minikits

Minikits are small, tweakable and extendable packages with full source code that illustrate how to implement interesting behaviours for your project. They're designed to be easy-to-use and self-explanatory.

1 What's new

v2.1.0

- Add interpolation property that may be used to smooth the motion of the rope
  - This is especially useful when a low fixed update rate is used
  - The behavior mimics the RigidbodyInterpolation setting and has the following values
    - None
    - Interpolate
    - Extrapolate
- Improve performance when generating rope geometry (by taking advantage of newer Unity APIs)

v2.0.0

- Replace the RopePin and RopeRigidbodyConnection components with a single RopeConnection component and add 2 new connection types
    - There are now 4 connection types:
        - Pin Rope To Transform
        - Pin Transform To Rope
        - Pull Rigidbody To Rope
        - Two Way Coupling Between Rigidbody And Rope
    - Warning: This change breaks backwards compatibility!
- Add gravity property for setting the gravity vector on a per-rope basis
- Fix bug where wrong impulse function was used for rigidbody feedback
    - Warning: This might require re-tweaking of the stiffness/damping values of your existing rope setups
- Fix bug where rigidbody rotational constraints would not be handled properly when connected to a rope
- Reskin example scene
- Switch to 3 digit semantic versioning

v1.11

- More accurate rope length calculation on SplitAt()
- Fix bug where multiple ropes with custom meshes would allocate unnecessary memory

v1.1

- Add support for dynamically splitting the rope using the new SplitAt() method
    - Try it out in the example scene by grabbing a rope with the mouse pointer and pressing space before letting go!
- Add support for custom meshes that can be rendered instead of the default rope cylinder
    - To illustrate this, the crane in the example scene now has a chain with links instead of a smooth rope
- Change RopePin and RopeRigidbodyConnection components to be lazily initialized
- Rename RopeMeasurements struct to Measurements and make it a sub-type of Rope
- Fix bug where toggling the simulation.enabled flag could result in an IndexOutOfBounds exception

v1.03

- Fix bug where one end of the rope could be moved if it collided with something even though it was pinned down using a RopePin component

- Fix bug where one end of the rope would show more sag than the other end when the rope was being stretched
- Add Rope.GetCurrentLength() method
- The example scene rope material now uses an explicit texture as the built-in one previously used caused iOS builds to fail

v1.02

- Add soft backdrop to example scene

v1.01

- Fix rope enable/disable logic
- Add ResetToSpawnCurve() method

v1.0

- Initial release

2 Overview

Features

- Stable rope physics
- Many tweakable user parameters
- Ability to dynamically split ropes using the SplitAt() method
- 4 different rope connection types allow interaction with the rest of the scene
    - Pin Rope To Transform
    - Pin Transform To Rope
    - Pull Rigidbody To Rope
    - Two Way Coupling Between Rigidbody And Rope
- Scene view handles for adjusting rope spawn curve
- High performance is achieved using Unity jobs and the Burst compiler
    - Typical performance for the example scene with collisions enabled:
        - ~0.2 ms spent in job threads
        - ~0.7 ms spent on the main thread
    - Typical performance for the example scene with collisions disabled:
        - ~0.15 ms spent in job threads
        - ~0.35 ms spent on the main thread

- Full source code
- Example scene contains:
  - Physically simulated crane
  - Rope bridge
  - Swings
  - Mouse interaction script

Current Limitations

- Collision support for convex Mesh, Box, Sphere and Capsule colliders only
- It is difficult to get perfectly stiff ropes unless very small physics time steps are used
- Rope bridge currently only takes the resting mass of colliding rigidbodies into account, there is no impact effect
- Scripting knowledge required for creating stable rope suspended platforms other than rope bridges

3 Requirements

- Burst 1.1.2 or above

4 Components

Rope

This is the main component that simulates and renders the rope

RopeConnection

This component connects the rope it is attached to to a transform or rigidbody component in the scene. The resulting behaviour depends on what type of connection is used:

Pin Rope To Transform

Pins a point on the rope to a point on a transform. The transform can move freely and the rope will always follow along.

Pin Transform To Rope

Pins a point on a transform to a point on the rope. The rope will move freely and the transform will always follow along. This connection takes control of the transform.

Pull Rigidbody To Rope

Pulls a point on a rigidbody towards a point on the rope by applying velocity changes to the rigidbody. This connection does not take control of the rigidbody, other forces and constraints are respected.

Two Way Coupling Between Rigidbody And Rope

Introduces a two-way coupling between the rope and a rigidbody. The rope will react to the rigidbody and feedback impulses back to the rigidbody allowing for complicated setups such as the crane in the example scene. Care must be taken so that the rope `mass per meter` value is comparable to the masses of connected rigidbodies, otherwise the simulation may blow up. This connection does not take control of the rigidbody, other forces and constraints are respected.

5 Instructions

Workflow

1. Create an empty game object
2. Attach the Rope script to it
3. Add a few spawn points either using the scene view buttons (`Push spawn point`, `Pop spawn point`) or by manually changing the `Spawn Points` property in the inspector
4. Move around the spawn points using the scene view handles
    - Hold down left `Shift` to switch to the ordinary transform tool for more fine-grained control
5. Assign a material to the `Material` property of the rope
    - Optionally adjust the `Tiling` of the material
    - 1 scene unit is mapped to 1 texture tile (x-axis) lengt-wise
    - 1 texture tile (y-axis) wraps around the rope curlwise
6. Optionally attach any number of `RopeConnection` components to the rope
    - Set the `Body` or `Transform` reference to the object the rope should be connected to
    - Set the `Local Connection Point` to be the point in object local space to which the rope should be attached

General tips

- Examine the example scene to get an understanding for how to connect the rope in a typical scene
- Change the tweakable parameters when in play-mode to get a feel for what they do
    - There are tooltips for all tweakable parameters

- Look at the helper scripts to get an understanding for how one can interact with the rope using custom scripts

Performance tips

- Avoid enabling collisions unless absolutely necessary
- Disable simulation of ropes that are far away or out of view using a custom script (they will still be rendered)
- Only call rope methods in a custom script from `FixedUpdate()` or `LateUpdate()`

Collisions

Collision detection is very performance intensive, as all physics queries have to be performed on the main thread. Aim to keep the `Stride` value as high as possible to reduce the amount of queries. Another approach is to disable simulation of ropes that require collisions more aggressively and keep the number of active ropes with collisions enabed to a bare minimum, even though the total number in the scene is high.

Stiffness

Rope stiffness depends on many factors: the `Stiffness` value of the rope, the `Resolution` of the rope, the `Solver Iterations` of the rope and finally the `Fixed Timestep` value in `Project Settings` → `Time`. To achieve a stiff rope, choose a high stiffness value, a low resolution value, many solver iterations and a low fixed time step.

6 Scripting interface

The rope is simuated using a set of inter-connected particles (visuaized by spheres when selecting a rope in edit-mode). Since the physics simulation is inherently stable, one can move around these particles in almost any way imaginable. This enables many custom setups such as the rope bridge in the example scene (see `RopeBridgePlank.cs`).

Interface

To fascilitate custom setups, the rope exposes a small scripting interface. The table below shows the properties and methods available. For more information on a particular property or method, see the description in the source file.

`measurements`

Returns the measurements of the rope. The measurements remain constant after the rope is first initialized.

`currentBounds`

The current world-space bounds of the visual mesh

`PushSpawnPoint()`

Adds a new spawn point to the rope. May be called from edit-mode.

`PopSpawnPoint()`

Removes the last spawn point of the rope. May be called from edit-mode.

`GetParticleIndexAt(distance)`

Returns the index of the simulation particle at a particular distance along the curve of the rope

`GetScalarDistanceAt(particleIndex)`

Returns the scalar distance along the curve of the rope that a particular simulation particle is located at. The scalar distance is a value between 0 and 1. The lengthMultiplier is not taken into account. To get the distance along the rope in world space, multiply the scalar distance by the realCurveLength measurement.

`GetPositionAt(particleIndex)`

Returns the current position of a particular simulation particle

`SetPositionAt(particleIndex, position)`

Sets the position of a particular simulation particle

`GetVelocityAt(particleIndex)`

Returns the current velocity of a particular simulation particle

`SetVelocityAt(particleIndex, velocity)`

Sets the velocity of a particular simulation particle

`GetMassMultiplierAt(particleIndex)`

Returns the mass multiplier of a particular simulation particle. This value can be used to increase or decrease the weight of a section of the rope. A value of 0 will make the particle immovable. A value of 2 will make the particle twice as heavy as its neighbors. The default value is 1.

`SetMassMultiplierAt(particleIndex, value)`

Sets the mass multiplier of a particular simulation particle. This value can be used to increase or decrease the weight of a section of the rope. A value of 0 will make the particle immovable. A value of 2 will make the particle twice as heavy as its neighbors. The default value is 1.

`GetClosestParticle(point, out particleIndex, out distance)`

Finds the simulation particle closest to a particular point

`GetClosestParticle(ray, out particleIndex, out distance, out distanceAlongRay)`

Finds the simulation particle closest to a particular ray

`RegisterRigidbodyConnection(particleIndex, rigidbody, rigidbodyDamping, pointOnBody, stiffness)`

Registers a rigidbody connection for the next simulation frame. A rigidbody connection is a two-way coupling of a simulation particle to a traditional rigidbody. Make sure to call this method from FixedUpdate(). Any simulation particle involved in a rigidbody connection will get its mass multiplier reset to 1 at the end of the simulation frame.

`ResetToSpawnCurve()`

Resets the rope to its original shape relative to the current transform. Useful when activating a pooled game object that is deactivated and re-activated instead of destroyed and instantiated.

`GetCurrentLength()`

Computes the current length of the rope. In contrast to the measurements.realCurveLength field, this value includes the stretching of the rope due to stress.

`SplitAt(particleIndex, outNewRopes)`

Splits the rope at a specific simulation particle and returns the rope components of the newly instantiated game objects. Make sure that the supplied array has exactly 2 slots. A Unity message 'OnRopeSplit(Rope.OnSplitParams)' will be sent to each newly created rope.

Execution order

Make sure the custom script runs before the custom execution order of `Rope.cs`, which defaults to 100. Calling rope methods from `Update()` or after the rope's `FixedUpdate()` will halt the main thread as it waits for the rope simulation jobs to complete. This destroys parallelism and performance.

Example usage

```
using RopeMinikit;

public class RopeMover : MonoBehaviour
{
    public Rope rope;

    public void FixedUpdate()
    {
        if (rope == null)
        {
            return;
        }
        rope.SetMassMultiplierAt(0, 0.0f); // makes the particle immovable for the rope
        rope.SetPositionAt(0, transform.position);
    }
}
```

7 FAQ

Q: Why does a rigidbody with a freeze rotation constraint act strange when it is connected to a rope?

A: This is caused by a bug in Unity.

8 Contact

Please let me know if you run into any problems when using the minikit or if you have feedback on how I can improve it in the future. I am also interested in seeing projects that use any of my toolkits or minikits in practice!

Website: https://gustavolsson.com/

Contact: https://gustavolsson.com/contact/