Hand Cursor for Unity

Asset Documentation

by Dr. René Bühling · www.gamedev-profi.de

Updated: May 19, 2020

René Bühling

Thank you for purchasing my asset!

This document guides you through the features and use cases of the product. If you have questions or need additional help, please send an email to support@buehling.org.

Good luck with your projects,
René

Index

Notes on Current Release

New in version of May 2020:

Video: https://youtu.be/le5XbnMwynI

- New 2D cursor poses for Fight, Left, Right and Zoom.
- 2D version is still included, but moved into legacy state.
- Completely new, additional real-time 3D version:
    - 21 Hand Poses.
    - Smooth 3D transitions between states.
    - Separate Lighting of cursor mesh and game scene possible.
    - Library scripts for easy use and inclusion.
    - Better isolation of demo code.
    - Utilities for debugging and testing.
    - Easy exchange and extension of cursor symbols (objects held in fingers for some states).
    - Support for 3D and 2D/GUI usage.
    - Optimization features for smoothing and adjusting visual presentation and transitions.
- Extended documentation PDF.
- Video explainers and tutorials for the new 3D version.

Compatibility

This package was tested with Unity ***2017.4****.18f1* and ***Unity 2019.3****.6f1*.

My model and scripts work for ***Unity 5.3****.3f1*, too, but as Unity made their Asset Store tools no longer work with version 5, I had to use 2017 for submission.

Setup and Installation

Video: https://youtu.be/08QSjdOQgwI

- Import the package from Asset Store.
- There are two versions included:
  - ***2D (Legacy)*** – Previous version using simple 2D sprites.
  - ***3D*** – New version with 3D cursor.
- These versions are completely independent. Only import what you need. If you only want 3D cursors, simply do not import *2D (Legacy)* folder or delete it later (or keep it).

Quick Demo

- For demo/testing **create an empty Unity project** of type 3D.
- Open one of the Demo scenes and start ▶ ***Play Mode***.
- The demos should work immediately, but for proper usage in your project, some additional setup steps may be required as described in the following sections.

Using 2D Cursor

Only this section describes the 2D cursor version, which uses pre-rendered Sprites. It is pretty simple and straightforward to use. The rest of this document relates to the 3D Cursor version.

1. Add a 3D object with collider.
2. Add a SetCursor script to the object and assign its fields:
    a. Assign Hand1.psd to ***Default Image***.
    b. Assign the image that to be shown on mouse over to ***Pointer Image***.
    c. Optional: Assign an image shown when pressed to *Pressed Image*.
    d. Also, assign the hotspot coordinates. See *Hotspots.xlsx* for a list of values.

Note: `SetCursor.cs` is no library script, but an example script that you might customize to your needs, i.e. remove the material coloring code.

Setup your Project for 3D Cursor

In order to render the cursor object on top of the scene and to avoid intersection with other objects, a separate render layer and a separate camera are required.

Video: https://youtu.be/oIwVCWK4PnY

1. Create Cursor instance below Main Camera

1. Drag Prefab ***Hand Cursor Cam*** from Project tab into Hierarchy.
2. Make sure the HandCursorCam is a child of the ***main camera***.

2. Setup render layers

1. In Hierarchy, select ***HandCursor3D*** below the *HandCursorCam* object.
2. In Inspector, check its ***Layer*** property. If this is empty or set to a layer, which is not reserved to cursor rendering, create a new layer. Click on the Layer dropdown and select *Add Layer*. Type the name `Hand Cursor` into an unused layer. By default, the asset expects to use Layer 8 for this, but it should work with other layers, too.
3. Return to the ***HandCursor3D*** object and make sure the object and all of its children are have the new layer assigned at their layer field in Inspector.
4. Select the ***main camera*** and exclude the new cursor layer from the list of ***Culling Mask*** in Inspector.

3. Testing

1. Drag Prefab ***Grab_Drag_Tester*** from Project window *Assets/HandCursor/3D* into hierarchy.
2. Start ▶ ***Play Mode*** and test if the hand cursor works.

> *Tipp:* Use Debugger Tool to scan for common problems in Hand Cursor setup. The tool can be opened with menu command *Window > Analysis > Hand Cursor Debugger*.

Show/hide the native mouse pointer

Video: https://youtu.be/cglG4kK8QFg

Use Script `SetNativeCursor` to hide the system mouse pointer or to show a helper image, i.e. a crosshair cursor instead.

- There is a ***Set Native Cursor*** component already present on the *HandCursorCam* → ***HandCursor3D*** prefab.
- ***Set Native Cursor*** has a property called ***Native Cursor*** that controls the visibility of the native cursor or cursor texture set in *Pointer Image* field. Usually it is set to ***Show in Editor only***, which means that the crosshair cursor is visible during development in Unity Editor, but hidden in the final build.

Create 3D Hotspots that change Cursor

Video: https://youtu.be/TSsHOK-oWxA

1. Call menu *Game Object* → *3D Object* → *Cube* to add new object.
2. In Inspector, add a ***Set Cursor 3D*** component to the cube.
3. Assign a Cursor State asset to the script's ***State*** field in Inspector, i.e. ***grab***.
4. Start ▶ ***Play Mode***. Move the mouse over the cube to see the hand change.

Using Push State for Mouse Pressed Event

- Assign a Cursor State asset to the ***Push State*** field to show a different state while the mouse button is pressed.
- Check ***Lock Push State*** to hold the Push State while the mouse button is pressed, even if other mouse events, like exit and enter, would trigger another state while dragging.

> ***Condition:*** The cursor requires a ***Collider*** component on any 3D hotspots.

Create 2D Hotspots that change Cursor

Video: https://youtu.be/G86tWmRKU-U

1. Call menu *Game Object* ⟶ *UI* ⟶ *Button* to create an UI setup.
2. Select the Canvas and make sure that Render Mode is ***Screen Space – Camera***. Also, assign the main camera to the ***camera*** property.
3. Select the button and in Inspector, add a ***Set Cursor 3D*** component.
4. Assign a Cursor State asset to the script's ***State*** field in Inspector, i.e. ***point***.
5. Start ▶ ***Play Mode***. Move the mouse over the cube to see the hand change.

> ***Condition:*** The UI canvas must use ***Screen Space - Camera*** mode to make the cursor render above the UI.

Custom Event Triggers

- When there is no *Event Trigger* component on the UI element (i.e. Button), the cursor script will automatically create one to attach the required cursor change events to the UI object.
- If you add an *Event Trigger* in Inspector, you must bind the Set Cursor script events manually:
    - Create a Pointer Enter trigger and bind `SetCursor.OnMouseEnter` to it.
    - Create a Pointer Exit trigger and bind `SetCursor.OnMouseExit` to it.

Limit Lights to Scene or Cursor

Video: https://youtu.be/Fwa5TgTACUk

Change Light Sources' ***Culling Mask*** property in Inspector to define their influence.

- Set the *Culling Mask* to ***Everything but not the Cursor Layer*** to use a light for the scene, but not for the Cursor meshes.
- Set the Culling Mask to ***Nothing but only the Cursor Layer*** to use a light for the Cursor meshes, but nothing else.

Decorative Objects: Cursor Symbols

Video: https://youtu.be/Z0YUzaL9q_Y

Some cursor poses like i.e. *read*, *put*, *drop*, *zoom* and *fight* support an additional item as a symbol for content or interaction elements. Position and scaling of the symbol are animated using a special bone named *symbolcontroller*. For easier usage there is an additional helper object called *symbol pivot* in Hierarchy.

- Challenge 1: Only show the symbol when the pose is suitable for a symbol.
  Solution: Use ***symbol pivot*** object as root and zero-position for your symbol meshes. It will copy its scaling from symbolcontroller bone.
- Challenge 2: Show different symbols for different states.
  Solution: Show *symbol pivot*'s `CursorSymbol` script in Inspector and define state modifiers there.
- To reduce the risk of unattractive model intersection when changing between states that carry a symbol, use the Option Grow & Shrink instead of De/Activate, which will create an additional size animation during show/hide events.

How to correctly adjust Transform of Symbol Objects

Video: https://youtu.be/UxTFgVOhf4M

1. Add your symbol object (i.e. capsule) below symbol pivot.
2. Verify that there is no collider active on your symbol object.

3. Start ▶ *Play Mode*.
4. Move the mouse over a hotspot that changes the hand cursor into the state for which you want to arrange your symbol.
5. Press `Ctrl+Shift+P` to pause the editor while the mouse is over the hotspot. This freezes the current game state, including the current cursor pose.
6. Use Transform tools inside a Scene window to rotate scale and place your symbol object into the hand pose.
7. Select your symbol object, go to Inspector, right-click on Transform component and choose *Copy* from popup menu.
8. Stop ▶ *Play Mode*. Transform of your symbol object is lost now.
9. Select your symbol object, go to Inspector, right-click on Transform component and choose *Paste Component Values* from popup menu.

To restrict visibility, define state modifiers in *symbol pivot*'s `CursorSymbol` script in Inspector.

Cursors States & Lazy Update Optimization

Video: https://youtu.be/JqTv-Zw1Tk8

- The actual Hand poses are defined through Cursor State assets.
- Cursor State assets' field `State` contains the name of the animation state to play when this cursor state is active. Usually there is no need to change this value.
- Cursor State assets' field `Transition Time` defines the number of seconds from blending from the previous pose into the pose described by this asset.
- Cursor State assets' field `Lazy Update` defines a number of seconds that will pass before the script starts blending into this asset's pose.
    - This is useful to skip the very short transition into idle state that happens when the mouse moves across a small gap between two hotspots.
    - In prefab setup it is sufficient to only set the Lazy Update value of the idle state as this is the state that is activated for a gap.
- Cursor State assets can be duplicated and customized freely. The design intension was to generate easy handling of cursor pose definitions.