Runtime Terrain Gizmos

Table of Contents

Introduction

Runtime Terrain Gizmos is a plugin for Unity that can be used for terrain editing purposes at runtime (in-game). The plugin implements a gizmo that allows you to change the height of terrain patches via a vertical slider. Elevation curves can be specified so that you can control the way in which elevation happens. The gizmo also allows you to move objects on the terrain vertically and even horizontally in order to change their position. Essentially, the gizmo can be used both as a terrain editor and an object position modifier.

Bonus features include:

- **Free Move Camera**
- **Undo/Redo**

This document serves as a quick introduction to the plugin API and it will get you up to speed with the most essential tasks such as creating a terrain gizmo, configuring it for your own needs, enabling/disabling the gizmo.

Don't forget to download the demo from here.

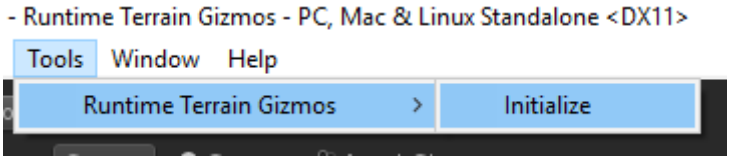
Note: Before moving on, please take the time to watch this video to get accustomed to how the gizmo works.

For any **questions** or **suggestions**, you can contact me at octamodius@yahoo.com

Setup

In order to start using the plugin, you will have to perform the necessary steps:

1. import the plugin into your project;
2. on the top menu, go to **Tools->Runtime Terrain Gizmos->Initialize** as shown in the image below:



3. in the hierarchy view, you will notice a hierarchy of objects that were created for you automatically. The root of the hierarchy is called **RTGApp**. This hierarchy with all its children needs to be in the scene. **Note:** All objects in the hierarchy need to be present in the scene, so you should not delete them.
4. in the **RTGApp** hierarchy there is a child object called **RTFocusCamera**. Select it, and make sure that the **Target camera** field points to a camera in the scene. If the scene contains a camera tagged as **Main Camera**, it will automatically be assigned to this field. Otherwise, you will have to assign a camera;
5. you can now use the plugin.

Integration with Runtime Transform Gizmos

RTRG can be integrated with the Runtime Transform Gizmos plugin. **Note:** it is important to follow the integration steps in the exact order as they are listed.

1. make sure you start with a clean slate in your project. That is, make sure that none of the 2 plugins are present in your project. If they are, delete both;
2. import **RTRG** into your project;
3. delete the **Common** folder. The folder can be found at the following path: **Runtime Terrain Gizmos/Scripts**;
4. import the **Runtime Transform Gizmos** pack into your project;
5. in the top menu, go to **Tools->Runtime Transform Gizmo->Initialize**;
6. the 2 plugins can now be treated as one.

Creating a Terrain Gizmo

In this section we are going to cover the necessary steps for creating a terrain gizmo. **Note:** we are going to assume that this code is part of a **MonoBehaviour** script.

Create a terrain gizmo:

```
// Create the gizmo
Gizmo gizmo = RTGizmosEngine.Get.CreateGizmo();
TerrainGizmo terrGizmo = gizmo.AddBehaviour<TerrainGizmo>();

// Inform the gizmo about the terrain that it will be working with
terrGizmo.SetTargetTerrain(GameObject.Find("Terrain").GetComponent<Terrain>());
```

Note: It is assumed that the scene contains a game object called **Terrain** that has a **terrain component** attached to it as well as a **terrain collider**.

Before moving on, let's take a look at the code above and talk about some of the things that might have caught your eye. First of all, notice that we first call the **CreateGizmo** function of the **RTGizmosEngine** singleton. The return type is **Gizmo**. You can think of a gizmo as the equivalent of Unity's **GameObject** class. The actual terrain gizmo is implemented as a gizmo behaviour. So

the next thing we do, is call the **AddBehaviour** function of the **Gizmo** class to create a terrain gizmo.

Finally, once we have a reference to the terrain gizmo, we call **SetTargetTerrain** to tell the gizmo about the terrain that it will be operating on.

The RTGApp.Initialized Event

In the previous section we covered gizmo creation. Before moving on, there is one thing left to mention. All gizmo creation should be performed after the **RTGApp** object has performed all the necessary initializations. This is done by registering an event handler to the **RTGApp.Initialized** event.

Let's see a simple example of this:

```
public class MyGizmoApp : MonoBehaviour
{
    private void Awake()
    {
        // Register event handler for the Initialized event
        RTGApp.Get.Initialized += OnAppInitialized;
    }

    private void OnAppInitialized()
    {
        // Create gizmos
        // ...
    }
}
```

Enabling/Disabling the Gizmo

In order to enable or disable gizmos, you need to use the **SetEnabled** function of the **Gizmo** class.

Let's see 2 examples of this in action;

```
terrGizmo.Gizmo.SetEnabled(false);
terrGizmo.Gizmo.SetEnabled(true);
```

Disabling a gizmo might be necessary if you are working on an application which supports different modes. For example, in one mode, terrain editing is allowed. In this mode, the gizmo is visible. In another mode, terrain editing is not allowed so the gizmo would be to be disabled when switching to this other mode.

Elevation Curves

By default, the gizmo uses a linear curve to define how much influence the gizmo has on surrounding vertices. This means that the influence will decrease in a linear fashion as vertices get further away from the gizmo position. You can define your own elevation curves inside the Unity Editor.

An elevation curve is of type **AnimationCurve** defined by Unity. The easiest way to build these curves is to store them in a Monobehaviour and then edit them inside the Inspector. The demo scene uses this strategy.

At the API level, you can assign an elevation curve using the following code:

```
terrGizmo.ElevationCurve = myCurve;
```

From this point on, any changes that you make to the terrain or to the objects in radius, will be affected by this curve.

Specifying Gizmo Targets

By default, when you interact with the gizmo, both the terrain and the objects that fall inside the gizmo radius will be affected. You can change that from the API:

```
terrGizmo.TargetTypes = TerrainGizmo.TargetTypeFlags.Terrain; // Affect only terrain
terrGizmo.TargetTypes = TerrainGizmo.TargetTypeFlags.ObjectsInRadius; // Affect only objects in radius
terrGizmo.TargetTypes = TerrainGizmo.TargetTypeFlags.All;   // Affect all
```

The demo scene defaults to **TargetTypeFlags.All** but whenever you press the **Left Shift** key, it will set the target type to **TargetTypeFlags.ObjectsInRadius**.

Gizmo Settings

Look & Feel, Settings and Hotkeys

There are 3 categories of settings associated with the gizmo:

- **Look & Feel** - controls the gizmo appearance;
- **Settings** - controls functional settings (e.g. snapping);
- **Hotkeys** - allows you to change the hotkeys that are associated with different types of states that can be activated for the gizmo (e.g. enable/disable snapping).

The property names are pretty much self-explanatory and are not listed here, but let's look at a simple example which shows how to change different gizmo settings.

```
// Change look and feel
terrGizmo.LookAndFeel.AxisSliderColor = Color.green;
terrGizmo.LookAndFeel.AxisSliderCapColor = Color.red;
terrGizmo.LookAndFeel.MidCapType = GizmoCap3DType.Sphere; // Only Box and Sphere allowed

// Change the snap steps
terrGizmo.Settings.OffsetSnapStep = 1.5f;
terrGizmo.Settings.RadiusSnapStep = 0.5f;

// By default, snapping is enabled via the LCTRL key.
// In this example, let's disable the LCTRL key and use the 'V' key instead.
terrGizmo.Hotkeys.EnableSnapping.LCtrl = false;
terrGizmo.Hotkeys.EnableSnapping.Key = KeyCode.V;
```

Enabling/Disabling Snapping

By default, when you hold down the **Left Control** key, snapping will be activated and will allow you to elevate the terrain or offset objects in snap increments defined in the **Settings** instance associated with the gizmo.

However, there may be times when you want to enable/disable snapping based on certain conditions. In that case, you can use the **SetSnapEnabled** function:

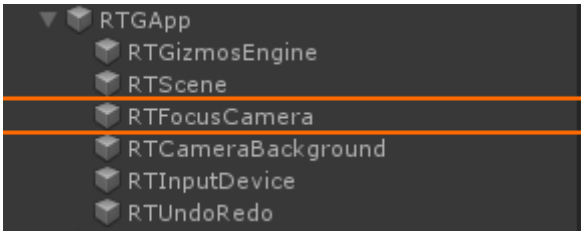
```
terrGizmo.SetSnapEnabled(isEnabled);
```

Note: Even if you call this function passing **false** as parameter, snapping will still be activated when the snap enable hotkey is held down. If you wish to disable the hotkeys, you can do it like this:

```
boxColliderGizmo.Hotkeys.EnableSnapping.IsEnabled = false;
```

The Free Move Camera

The free move camera included with the plugin behaves in a similar manner to the camera associated with the Unity Editor. The camera behaviour is implemented in the **RTFocusCamera** script. When you initialize the plugin, a **RTFocusCamera** object is automatically created for you and is part of the **RTGApp** hierarchy as shown in the image below:

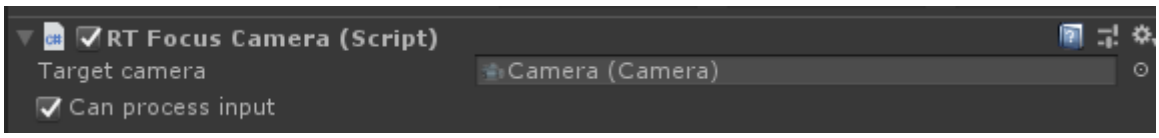


The camera provides the following operations:

- move around;
- rotate/look around;
- orbit;
- pan;
- zoom;
- focus;
- projection switch;
- rotation switch;

When you select the camera object in the hierarchy view, the Inspector will allow you to change different settings that control the way in which the camera behaves.

There are 2 important fields here that are worth discussing:



- **Target camera** - this is filled out automatically when the plugin is initialized if there is a camera tagged **Main Camera** in the scene. If not, you will have to assign a camera here. This is a mandatory field. a camera has to be assigned here.

- **Can process input** - If you do not wish to use the camera functionality (e.g. rotate, move, orbit etc), then you can uncheck this toggle. You may want to do this if you have your own camera script that you would like to use instead. **Note:** Even if you decide to use your own camera script, the **RTFocusCamera** object needs to stay in the scene as part of the **RTGApp** hierarchy. It is needed by the gizmo engine.

Camera Focus

You can instruct the camera to focus on a particular object or group of objects in the same way as you can do inside the Unity Editor. The following code shows you how you can do this:

```
// Assume this is the list that contains the object on which the camera should focus
List<GameObject> targetObjects = new List<GameObject>();

// Populate the list
// ...

// Focus
RTFocusCamera.Get.Focus(targetObjects);
```

Camera Rotation Switch

The camera supports an operation called **Rotation Switch**. A rotation switch aligns the camera to a specified rotation. It is what happens when you click on one of the scene gizmo cone axes in the top right corner of the scene view in the Unity Editor.

```
RTFocusCamera.Get.PerformRotationSwitch(targetRotation);
```

Camera Projection Switch

A projection switch happens when the camera switches from perspective to orthographic projection. This is the same behaviour as the one that can be observed when clicking on the scene gizmo cube in the top right corner of the scene view in the Unity Editor.

```
RTFocusCamera.Get.PerformProjectionSwitch();
```

Hotkeys

Gizmo Hotkeys

- **LCTRL** - enable snapping;
- **C** - enable object rotation;

Undo/Redo

- **Undo** - LCTRL + Z
- **Redo** - LCTRL + Y

Camera

- **LMB** + **WASDQE** - move camera forward, left, backwards, right, down and up respectively;

- **LMB** + **MOUSE MOVE** - rotate to look around;
- **LMB** + **LALT** + **MOUSE MOVE** - orbit around focus point;
- **MMB** + **MOUSE MOVE** - pan;
- **MOUSE SCROLL WHEEL** - zoom in/zoom out;