START ASSETS

Powerful Preview

User Manual

2020

Introduction

The **Powerful Preview** asset is a unity editor plugin, that allows you to extend your assets with the preview, which works exactly the same way the original unity preview does, but it allows you to extend and control it.

In order to achieve that, it would be nice to have advanced knowledge of C# and experience with how to write your own editors for the assets.

The plugin has quite a lot of samples of what you can achieve with the **Powerful Preview**, all of them have source code available, so you can learn by example.

How to use?

You may set up the preview for your assets in two possible ways

1) Simple – by inheriting your class from `PreviewEditor<T>` base class,
2) Complex - by initializing the preview by yourself.

All examples of the further code can be found under

`Start Assets/PowerfulPreview/Samples/Tutorials`

Simple Approach

Under `Start Assets\PowerfulPreview\Editor\Editors` directory you will find all the built-in base editor classes. You may want to start just with the `PreviewEditor.cs`, which has everything you need to learn how to work with **Powerful Preview**.

Let's say you have some custom asset, implemented as a scriptable object with just a single string field:

```
using UnityEngine;

public class YourCustomAsset : ScriptableObject
{
    [SerializeField]
    private string m_Data;
}
```

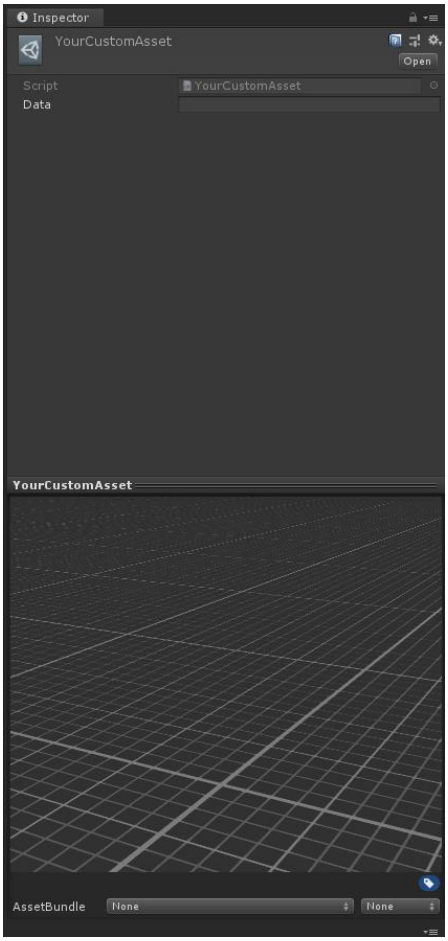
Then you will need to make an editor for it:

```
using UnityEngine;
using UnityEditor;
using StartAssets.PowerfulPreview;

[CustomEditor(typeof(YourCustomAsset))]
public class YourCustomAssetEditor : PreviewEditor<YourCustomAsset>
{
    protected override void OnCreate()
    {
        //Initialize your data here
    }

    protected override void OnGUIUpdate()
    {
        //Draw your GUI here
        DrawDefaultInspector();
    }
}
```

Now you can access the preview by simply using preview property, and the asset can be accessed by asset property:

```
asset.name = "YourCustomAsset";
preview.Update();
```

That’s all what is required to set up the preview for your custom asset with the simple approach. After everything is done you may click on your asset and you should see something like this:



Complex Approach

This is what we will start with:

```
using UnityEditor;
using StartAssets.PowerfulPreview;

[CustomEditor(typeof(YourCustomAsset))]
public class YourCustomAssetEditor : Editor
{

}
```

First thing that is required is to ask for constant repaint of the inspector, but also set that the editor has preview GUI:

```
public override bool RequiresConstantRepaint()
{
    return true;
}
public override bool HasPreviewGUI()
{
    return true;
}
```

After that you need to create the preview during `OnEnable` event call, and release it during `OnDisable` event call:

```
private void OnEnable()
{
    mPreview = Preview.Create(this);
}
private void OnDisable()
{
    mPreview?.Dispose();
}

private Preview mPreview;
```

Then you will need to implement a method to draw the preview, and inspector gui draw method, if you need one:

```
public override void OnInteractivePreviewGUI(Rect r, GUIStyle background)
{
    mPreview?.SetSurfaceRect(r);
    mPreview?.Update();

    //Do other things with the preview if you need...
}
public override void OnInspectorGUI()
{
    DrawDefaultInspector();

    //Draw custom inspector GUI if you need.
}
```

Then you should achieve exactly the same result as with the Simple Approach.

How to add game objects to preview

First of all, you need to learn how to add objects to the preview scene.

The class which is responsible for that is `PreviewScene`. It has the next important methods:

```
/// <summary> Creates new game object based on the prefab and adds it to the pre ...
public virtual GameObject Instantiate(GameObject prefab, bool gizmoLayer = false)...
/// <summary> Adds object to the preview scene.
public virtual void AddObject(GameObject gameObject, bool gizmoLayer = false)...
/// <summary> Adds object to the preview scene.
public virtual void AddObject(GameObject gameObject, Material customMaterial, bool gizmoLayer = false)...
/// <summary> Destroys the preview game object instance.
public virtual void DestroyInstance(GameObject instance)...
```

`Instantiate` should be sufficient in most of the cases when you are dealing with prefabs, but if you create an object in the runtime, you should add it as an object with `AddObject` call.

If you need to recreate the object, you should destroy the instance first with `DestroyInstance` call.

After you've added the object to preview you should see it, and now you can control it as you would do with just Unity script.

How to animate game objects

Characters animation

There is a class that will allow you to animate your characters - `PreviewAnimator`. `Animation Preview Asset` is a good example of how to use it.

First thing you need to create an instance of the `PreviewAnimator` and set the current preview instance in the class constructor. Then you should set `Animation` property, and set up the character, that will be used to play animation on it.

`PreviewAnimator` will use `PreviewAnimator.DefaultUnityCharacter` as a fallback, if you set the `Character` property to null.

Then on any update method call (depending on your editor implementation it might be either `OnGUIUpdate`, `OnInspectorGUIUpdate` or `OnPreviewUpdate`) you should just sample animation with `PreviewAnimator.SampleAnimation(…)` call. Example from the `Animation Preview Asset`:

```
protected override void OnPreviewUpdate()
{
    if( mPreviewAnimator == null || mTimeline == null )
    {
        return;
    }
    mPreviewAnimator.SampleAnimation( mTimeline.CurTime );
}
```

Legacy Animation

If you need to animate a primitive object with legacy animation, you should just use something like `animationClip.SampleAnimation( primitiveObject, time )`.

Camera Animation

Also, you can animate preview camera, or preview camera frustum. The way you do it is all the same across the code, it's a method named `SampleAnimation` and `Cutscene Preview Asset` is a good example of how to do this:

```
//Example of how to animate camera
preview.Camera.SampleAnimation(asset.cameraAnimation, animationTime);
//Example of how to animate frustum
mCameraFrustumObject.SampleAnimation(asset.cameraAnimation, animationTime);
```

Particle System Animation

To animate a particle system in the preview you need to use `ParticleSystemAnimator` class. When you create it, you need to set a prefab of the particle system, as the parameter of the constructor. Then on any update call you should use `Simulate` method. `Particle System Preview Asset` is a good example of how to do this.

Preview Camera

Preview camera has all the options that original Unity camera does (maybe a bit lighter set). So, for example, if you want to control the camera type, you can just use `preview.Camera.orthographic` property, if you want to change the field of view, you should set `preview.Camera.fieldOfView` property. If you want to move or rotate the car at some position, you can just use `preview.Camera.transform`.

Also, it's possible to cast the `PreviewCamera` to just `UnityEngine.Camera`, it happens implicit, good example of usage is `Canvas Preview Asset`, as the canvas is drawn in the world space, and the preview camera is used as the world camera:

```
var canvas = gameObject.GetComponent<Canvas>();
if( canvas != null )
{
    canvas.worldCamera = preview.Camera;
    canvas.renderMode = RenderMode.ScreenSpaceCamera;
    canvas.planeDistance = 1.0f;
}
```

If you want to change the settings of how the preview camera behaves, you should use `PreviewCameraController` class, through the preview property `preview.CameraController`. Here you can set the move/rotate/zoom speeds, disable/enable some specific possible state (Dragging, Rotating, Zooming), so you may disallow user from dragging the camera:

```
/// <summary> Sets all PreviewCameraStates to be enabled/disabled.
public void SetStatesEnabled(bool value)...
/// <summary> Sets some specific state to be enabled/disabled.
public void SetStateEnabled(PreviewCameraStates state, bool value)...
/// <param name="state">State to check.</param> ...
public bool IsStateEnabled(PreviewCameraStates state)...
```

The camera controller is implemented as an orbit camera, so it's possible to set the orbit radius, and also the target the camera is panning around:

```
/// <summary> Radius of the camera orbit.
public float OrbitRadius...
/// <summary> Target of the camera.
public Vector3 Target...
```

Preview Controls

The preview can draw the controls over itself, allowing you to implement some tools, that you can to extend the preview. At the moment, there is only one built-in control - `Timeline`, which is used a lot for anything that needs to be animated with time. To create add a control, you need to create an instance and use `preview.AddControl(…)` method, example from the `Animation Preview Asset`:

```
mTimeline = new Timeline();
mTimeline.Visible = asset.animationClip != null;
preview.AddControl( mTimeline );
if( asset.animationClip != null )
{
    mTimeline.EndTime = asset.animationClip.length;
    mTimeline.Framerate = asset.animationClip.frameRate;
}
```

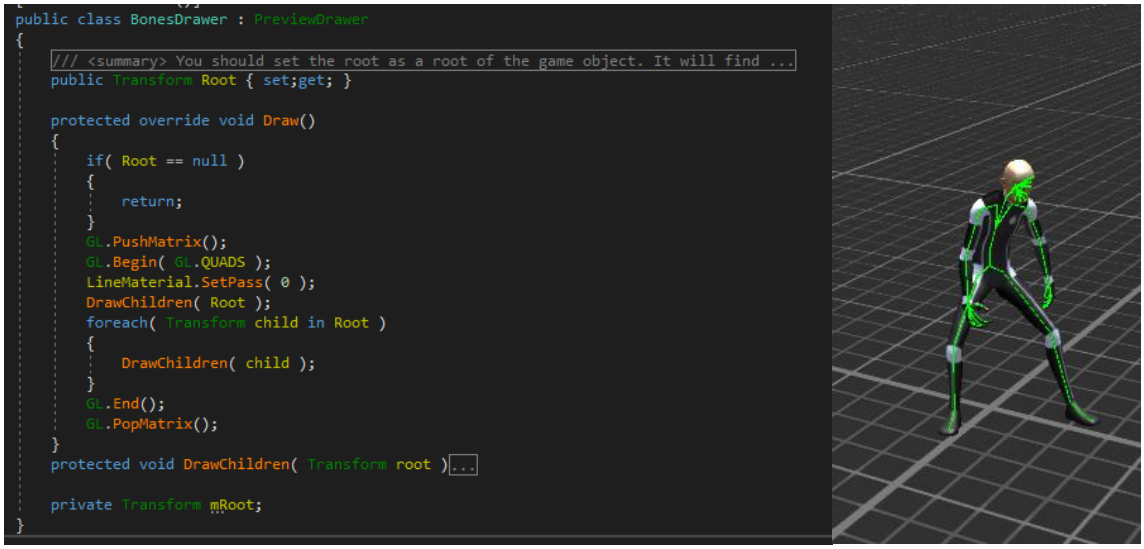
There is also a special preview editor class, which will utilize the timeline automatically - `TimelinePreviewEditor`, you can use it in case you need to use the timeline, but you don't want to set it up again, good example of usage is the `Particle System Preview Asset`:

```
/// <summary> An example of the preview editor which draws preview for some spec ...
[CustomEditor(typeof(ParticleSystemPreviewAsset))]
public class ParticleSystemPreviewAssetEditor : TimelinePreviewEditor<ParticleSystemPreviewAsset>
{
    [MenuItem("Assets/Create/Powerful Preview Samples/Particle System Preview Asset")]
    public static void CreateParticleSystemPreviewAsset()...

    protected override void OnCreate()...
    protected override void OnGUIUpdate()...

    private void CreateParticleSystemProvider()...

    private ParticleSystemAnimator mParticleSystemProvider;
}
```

Preview Drawers

It's possible to draw additional information in the preview. To do this you should simply inherit the `PreviewDrawer` class and implement abstract method `Draw`.

You might find a good example of how to do this in the `PreviewCameraFrustum` and `BonesDrawer` classes:



```
public class BonesDrawer : PreviewDrawer
{
    /// <summary> You should set the root as a root of the game object. It will find ...
    public Transform Root { set;get; }

    protected override void Draw()
    {
        if( Root == null )
        {
            return;
        }
        GL.PushMatrix();
        GL.Begin( GL.QUADS );
        LineMaterial.SetPass( 0 );
        DrawChildren( Root );
        foreach( Transform child in Root )
        {
            DrawChildren( child );
        }
        GL.End();
        GL.PopMatrix();
    }
    protected void DrawChildren( Transform root )...

    private Transform mRoot;
}
```

More information about how you can draw things you may find by these links:

https://docs.unity3d.com/ScriptReference/GL.html

https://docs.unity3d.com/ScriptReference/Graphics.html

So, anything that might be drawn by Unity side, can be shown in the preview. The `PreviewDrawer` class contains a small set of static methods for drawing primitive shapes like plane or line:

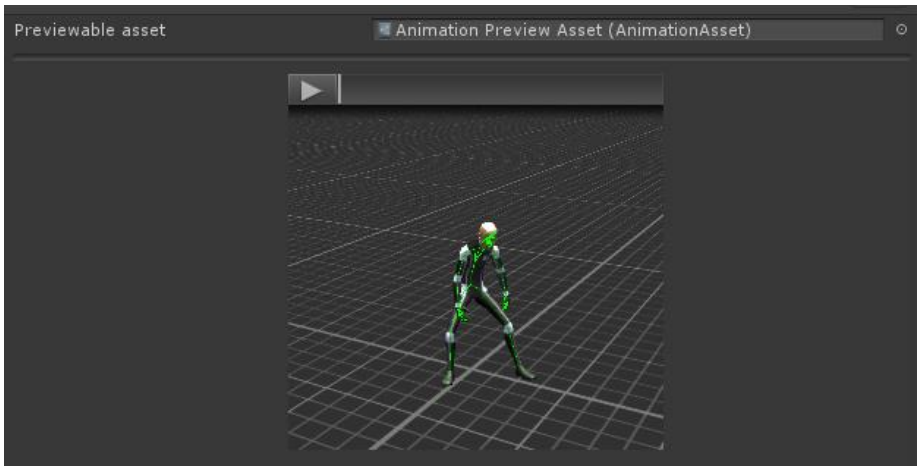
```
/// <summary> Draws one pixel line from one 3D point to another.
public static void DrawLine(Vector3 start, Vector3 end)...
/// <summary> Draws line with custom width from one 3D point to another.
public static void DrawLine(Vector3 start, Vector3 end, Color color, float width)...
/// <summary> Draws wireframe plane with four points.
public static void DrawPlane(Vector3 v1, Vector3 v2, Vector3 v3, Vector3 v4)...
/// <summary> Draws wireframe plane with four points and specific line width.
public static void DrawPlane(Vector3 v1, Vector3 v2, Vector3 v3, Vector3 v4, float width)...
```

Experimental (Advanced)

Embedded Preview

Thera are also examples of how to implement different advances techniques, using the Powerful Preview.

For example, you can implement an embedded preview inside the inspector of your custom asset, you can see how it's possible with `Embedded Preview Asset`:

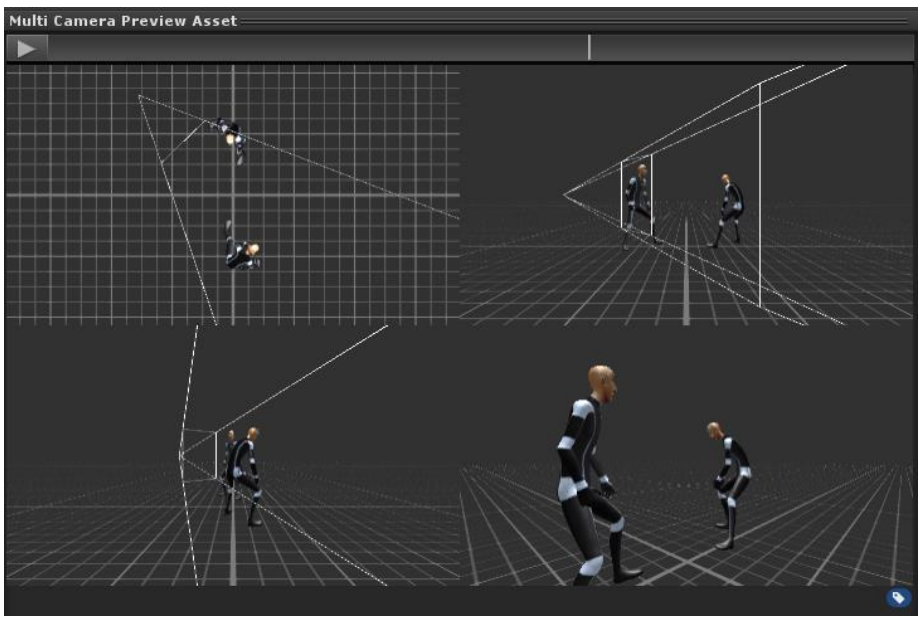


The key here is to just make the calls to `editor.OnInteractivePreviewGUI( rect, style)` with proper a rect to be drawn in:

```
var rect = new Rect( Screen.width / 2 - 128 + lastRect.xMin, yPreviewPos, 256, 256 );
if ( mAssetEditor != null )
{
    mAssetEditor.OnInteractivePreviewGUI( rect, GUI.skin.box );
}
```

Multiple Cameras

Also, it's possible to use multiple cameras at the same time to draw a scene from different point of views, you can even animate them separately. An example of this technique you can find in `Multi Camera Preview Asset`:



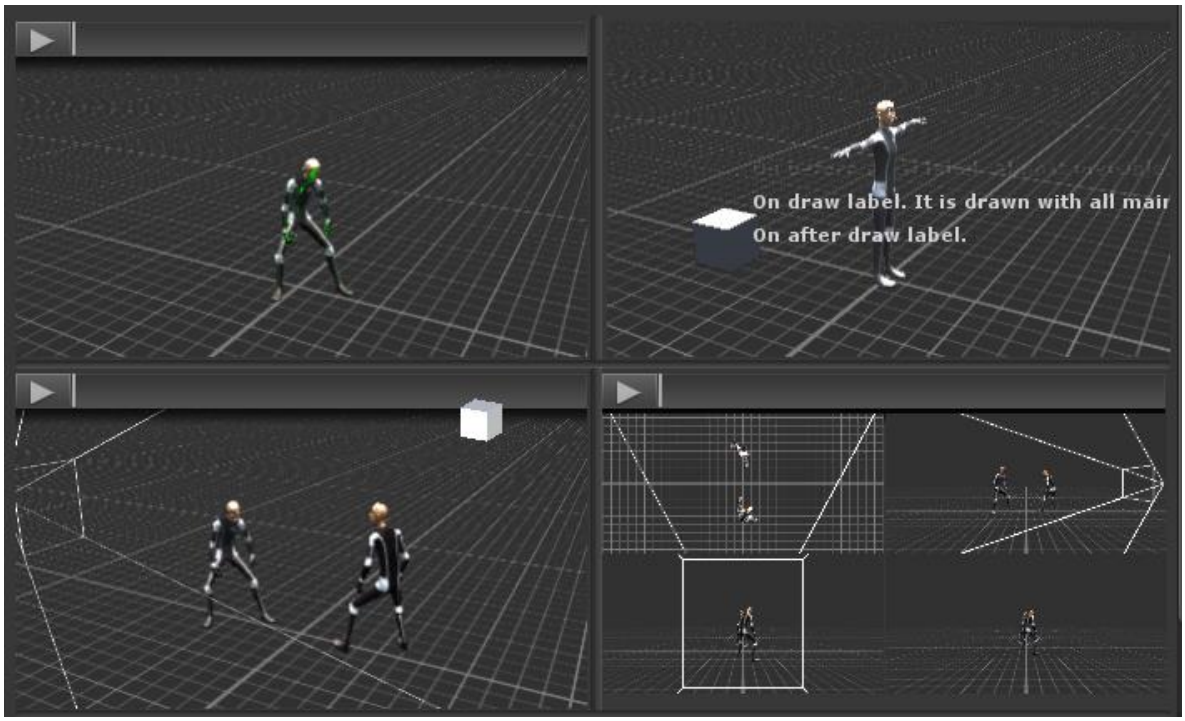
To achieve that you need to use `PreviewCameraSetup` class, which will store all the parameters of each camera, like position, rotation, FOV etc., then you iterate through array of the cameras and apply the settings with the `Apply` call:

```
mCameras = new PreviewCameraSetup[4]
{
    new PreviewCameraSetup( preview ),
    new PreviewCameraSetup( preview ),
    new PreviewCameraSetup( preview ),
    new PreviewCameraSetup( preview )
};

mCameras[TopCamera].Orthographic = true;
mCameras[TopCamera].OrthographicSize = 2;
mCameras[TopCamera].Position = new Vector3(0, 2, 0);
mCameras[TopCamera].Rotation = Quaternion.LookRotation(Vector3.down);

///...
```

```
for( int iRect = 0; iRect < rects.Length; iRect++ )
{
    mCameras[iRect].Apply();
    ///...
```

Multiple Asset Previews

Sometimes you may want to create a preview for multiple assets, first you need to make sure that those assets implemented preview in any way and support it. An example of how to use it you can find in `Multi View Preview Asset`:



UnityEditor.Handles

It's also possible to draw with `UnityEditor.Handles` inside the preview, but it's still raw, so there is only one simple example, you can find it in the `Game Object Asset`, there is a cube at the top right corner, that represents current orientation of the preview camera.

Feedback

If you have questions, write me to startassets@gmail.com and I'll be happy to answer them!