using System;

using UnityEditor;

using UnityEngine;

namespace UnityStandardAssets.ImageEffects

{

[CustomEditor (typeof( NoiseAndGrain))]

class NoiseAndGrainEditor : Editor

{

SerializedObject serObj;

SerializedProperty intensityMultiplier;

SerializedProperty generalIntensity;

SerializedProperty blackIntensity;

SerializedProperty whiteIntensity;

SerializedProperty midGrey;

SerializedProperty dx11Grain;

SerializedProperty softness;

SerializedProperty monochrome;

SerializedProperty intensities;

SerializedProperty tiling;

SerializedProperty monochromeTiling;

SerializedProperty noiseTexture;

SerializedProperty filterMode;

void OnEnable () {

serObj = new SerializedObject (target);

intensityMultiplier = serObj.FindProperty("intensityMultiplier");

generalIntensity = serObj.FindProperty("generalIntensity");

blackIntensity = serObj.FindProperty("blackIntensity");

whiteIntensity = serObj.FindProperty("whiteIntensity");

midGrey = serObj.FindProperty("midGrey");

dx11Grain = serObj.FindProperty("dx11Grain");

softness = serObj.FindProperty("softness");

monochrome = serObj.FindProperty("monochrome");

intensities = serObj.FindProperty("intensities");

tiling = serObj.FindProperty("tiling");

monochromeTiling = serObj.FindProperty("monochromeTiling");

noiseTexture = serObj.FindProperty("noiseTexture");

filterMode = serObj.FindProperty("filterMode");

}

public override void OnInspectorGUI () {

serObj.Update();

EditorGUILayout.LabelField("Overlays animated noise patterns", EditorStyles.miniLabel);

EditorGUILayout.PropertyField(dx11Grain, new GUIContent("DirectX 11 Grain"));

if (dx11Grain.boolValue && !(target as NoiseAndGrain).Dx11Support()) {

EditorGUILayout.HelpBox("DX11 mode not supported (need DX11 GPU and enable DX11 in PlayerSettings)", MessageType.Info);

}

EditorGUILayout.PropertyField(monochrome, new GUIContent("Monochrome"));

EditorGUILayout.Separator();

EditorGUILayout.PropertyField(intensityMultiplier, new GUIContent("Intensity Multiplier"));

EditorGUILayout.PropertyField(generalIntensity, new GUIContent(" General"));

EditorGUILayout.PropertyField(blackIntensity, new GUIContent(" Black Boost"));

EditorGUILayout.PropertyField(whiteIntensity, new GUIContent(" White Boost"));

midGrey.floatValue = EditorGUILayout.Slider( new GUIContent(" Mid Grey (for Boost)"), midGrey.floatValue, 0.0f, 1.0f);

if (monochrome.boolValue == false) {

Color c = new Color(intensities.vector3Value.x,intensities.vector3Value.y,intensities.vector3Value.z,1.0f);

c = EditorGUILayout.ColorField(new GUIContent(" Color Weights"), c);

intensities.vector3Value = new Vector3(c.r, c.g, c.b);

}

if (!dx11Grain.boolValue) {

EditorGUILayout.Separator();

EditorGUILayout.LabelField("Noise Shape");

EditorGUILayout.PropertyField(noiseTexture, new GUIContent(" Texture"));

EditorGUILayout.PropertyField(filterMode, new GUIContent(" Filter"));

}

else {

EditorGUILayout.Separator();

EditorGUILayout.LabelField("Noise Shape");

}

softness.floatValue = EditorGUILayout.Slider( new GUIContent(" Softness"),softness.floatValue, 0.0f, 0.99f);

if (!dx11Grain.boolValue) {

EditorGUILayout.Separator();

EditorGUILayout.LabelField("Advanced");

if (monochrome.boolValue == false)

{

Vector3 temp = tiling.vector3Value;

temp.x = EditorGUILayout.FloatField(new GUIContent(" Tiling (Red)"), tiling.vector3Value.x);

temp.y = EditorGUILayout.FloatField(new GUIContent(" Tiling (Green)"), tiling.vector3Value.y);

temp.z = EditorGUILayout.FloatField(new GUIContent(" Tiling (Blue)"), tiling.vector3Value.z);

tiling.vector3Value = temp;

}

else {

EditorGUILayout.PropertyField(monochromeTiling, new GUIContent(" Tiling"));

}

}

serObj.ApplyModifiedProperties();

}

}

}