using System;

using UnityEditor;

using UnityEngine;

namespace UnityStandardAssets.ImageEffects

{

[CustomEditor (typeof(CameraMotionBlur))]

class CameraMotionBlurEditor : Editor

{

SerializedObject serObj;

SerializedProperty filterType;

SerializedProperty preview;

SerializedProperty previewScale;

SerializedProperty movementScale;

SerializedProperty jitter;

SerializedProperty rotationScale;

SerializedProperty maxVelocity;

SerializedProperty minVelocity;

SerializedProperty velocityScale;

SerializedProperty velocityDownsample;

SerializedProperty noiseTexture;

SerializedProperty showVelocity;

SerializedProperty showVelocityScale;

SerializedProperty excludeLayers;

void OnEnable () {

serObj = new SerializedObject (target);

filterType = serObj.FindProperty ("filterType");

preview = serObj.FindProperty ("preview");

previewScale = serObj.FindProperty ("previewScale");

movementScale = serObj.FindProperty ("movementScale");

rotationScale = serObj.FindProperty ("rotationScale");

maxVelocity = serObj.FindProperty ("maxVelocity");

minVelocity = serObj.FindProperty ("minVelocity");

jitter = serObj.FindProperty ("jitter");

excludeLayers = serObj.FindProperty ("excludeLayers");

velocityScale = serObj.FindProperty ("velocityScale");

velocityDownsample = serObj.FindProperty ("velocityDownsample");

noiseTexture = serObj.FindProperty ("noiseTexture");

}

public override void OnInspectorGUI () {

serObj.Update ();

EditorGUILayout.LabelField("Simulates camera based motion blur", EditorStyles.miniLabel);

EditorGUILayout.PropertyField (filterType, new GUIContent("Technique"));

if (filterType.enumValueIndex == 3 && !(target as CameraMotionBlur).Dx11Support()) {

EditorGUILayout.HelpBox("DX11 mode not supported (need shader model 5)", MessageType.Info);

}

EditorGUILayout.PropertyField (velocityScale, new GUIContent(" Velocity Scale"));

if (filterType.enumValueIndex >= 2) {

EditorGUILayout.LabelField(" Tile size used during reconstruction filter:", EditorStyles.miniLabel);

EditorGUILayout.PropertyField (maxVelocity, new GUIContent(" Velocity Max"));

}

else

EditorGUILayout.PropertyField (maxVelocity, new GUIContent(" Velocity Max"));

EditorGUILayout.PropertyField (minVelocity, new GUIContent(" Velocity Min"));

EditorGUILayout.Separator ();

EditorGUILayout.LabelField("Technique Specific");

if (filterType.enumValueIndex == 0) {

// portal style motion blur

EditorGUILayout.PropertyField (rotationScale, new GUIContent(" Camera Rotation"));

EditorGUILayout.PropertyField (movementScale, new GUIContent(" Camera Movement"));

}

else {

// "plausible" blur or cheap, local blur

EditorGUILayout.PropertyField (excludeLayers, new GUIContent(" Exclude Layers"));

EditorGUILayout.PropertyField (velocityDownsample, new GUIContent(" Velocity Downsample"));

velocityDownsample.intValue = velocityDownsample.intValue < 1 ? 1 : velocityDownsample.intValue;

if (filterType.enumValueIndex >= 2) { // only display jitter for reconstruction

EditorGUILayout.PropertyField (noiseTexture, new GUIContent(" Sample Jitter"));

EditorGUILayout.PropertyField (jitter, new GUIContent(" Jitter Strength"));

}

}

EditorGUILayout.Separator ();

EditorGUILayout.PropertyField (preview, new GUIContent("Preview"));

if (preview.boolValue)

EditorGUILayout.PropertyField (previewScale, new GUIContent(""));

serObj.ApplyModifiedProperties();

}

}

}