using System;

using UnityEngine;

namespace UnityStandardAssets.ImageEffects

{

[ExecuteInEditMode]

[RequireComponent (typeof(Camera))]

[AddComponentMenu ("Image Effects/Camera/Camera Motion Blur") ]

public class CameraMotionBlur : PostEffectsBase

{

// make sure to match this to MAX\_RADIUS in shader ('k' in paper)

static float MAX\_RADIUS = 10.0f;

public enum MotionBlurFilter {

CameraMotion = 0, // global screen blur based on cam motion

LocalBlur = 1, // cheap blur, no dilation or scattering

Reconstruction = 2, // advanced filter (simulates scattering) as in plausible motion blur paper

ReconstructionDX11 = 3, // advanced filter (simulates scattering) as in plausible motion blur paper

ReconstructionDisc = 4, // advanced filter using scaled poisson disc sampling

}

// settings

public MotionBlurFilter filterType = MotionBlurFilter.Reconstruction;

public bool preview = false; // show how blur would look like in action ...

public Vector3 previewScale = Vector3.one; // ... given this movement vector

// params

public float movementScale = 0.0f;

public float rotationScale = 1.0f;

public float maxVelocity = 8.0f; // maximum velocity in pixels

public float minVelocity = 0.1f; // minimum velocity in pixels

public float velocityScale = 0.375f; // global velocity scale

public float softZDistance = 0.005f; // for z overlap check softness (reconstruction filter only)

public int velocityDownsample = 1; // low resolution velocity buffer? (optimization)

public LayerMask excludeLayers = 0;

private GameObject tmpCam = null;

// resources

public Shader shader;

public Shader dx11MotionBlurShader;

public Shader replacementClear;

private Material motionBlurMaterial = null;

private Material dx11MotionBlurMaterial = null;

public Texture2D noiseTexture = null;

public float jitter = 0.05f;

// (internal) debug

public bool showVelocity = false;

public float showVelocityScale = 1.0f;

// camera transforms

private Matrix4x4 currentViewProjMat;

private Matrix4x4 prevViewProjMat;

private int prevFrameCount;

private bool wasActive;

// shortcuts to calculate global blur direction when using 'CameraMotion'

private Vector3 prevFrameForward = Vector3.forward;

private Vector3 prevFrameUp = Vector3.up;

private Vector3 prevFramePos = Vector3.zero;

private Camera \_camera;

private void CalculateViewProjection () {

Matrix4x4 viewMat = \_camera.worldToCameraMatrix;

Matrix4x4 projMat = GL.GetGPUProjectionMatrix (\_camera.projectionMatrix, true);

currentViewProjMat = projMat \* viewMat;

}

new void Start () {

CheckResources ();

if (\_camera == null)

\_camera = GetComponent<Camera>();

wasActive = gameObject.activeInHierarchy;

CalculateViewProjection ();

Remember ();

wasActive = false; // hack to fake position/rotation update and prevent bad blurs

}

void OnEnable () {

if (\_camera == null)

\_camera = GetComponent<Camera>();

\_camera.depthTextureMode |= DepthTextureMode.Depth;

}

void OnDisable () {

if (null != motionBlurMaterial) {

DestroyImmediate (motionBlurMaterial);

motionBlurMaterial = null;

}

if (null != dx11MotionBlurMaterial) {

DestroyImmediate (dx11MotionBlurMaterial);

dx11MotionBlurMaterial = null;

}

if (null != tmpCam) {

DestroyImmediate (tmpCam);

tmpCam = null;

}

}

public override bool CheckResources () {

CheckSupport (true, true); // depth & hdr needed

motionBlurMaterial = CheckShaderAndCreateMaterial (shader, motionBlurMaterial);

if (supportDX11 && filterType == MotionBlurFilter.ReconstructionDX11) {

dx11MotionBlurMaterial = CheckShaderAndCreateMaterial (dx11MotionBlurShader, dx11MotionBlurMaterial);

}

if (!isSupported)

ReportAutoDisable ();

return isSupported;

}

void OnRenderImage (RenderTexture source, RenderTexture destination) {

if (false == CheckResources ()) {

Graphics.Blit (source, destination);

return;

}

if (filterType == MotionBlurFilter.CameraMotion)

StartFrame ();

// use if possible new RG format ... fallback to half otherwise

var rtFormat= SystemInfo.SupportsRenderTextureFormat (RenderTextureFormat.RGHalf) ? RenderTextureFormat.RGHalf : RenderTextureFormat.ARGBHalf;

// get temp textures

RenderTexture velBuffer = RenderTexture.GetTemporary (divRoundUp (source.width, velocityDownsample), divRoundUp (source.height, velocityDownsample), 0, rtFormat);

int tileWidth = 1;

int tileHeight = 1;

maxVelocity = Mathf.Max (2.0f, maxVelocity);

float \_maxVelocity = maxVelocity; // calculate 'k'

// note: 's' is hardcoded in shaders except for DX11 path

// auto DX11 fallback!

bool fallbackFromDX11 = filterType == MotionBlurFilter.ReconstructionDX11 && dx11MotionBlurMaterial == null;

if (filterType == MotionBlurFilter.Reconstruction || fallbackFromDX11 || filterType == MotionBlurFilter.ReconstructionDisc) {

maxVelocity = Mathf.Min (maxVelocity, MAX\_RADIUS);

tileWidth = divRoundUp (velBuffer.width, (int) maxVelocity);

tileHeight = divRoundUp (velBuffer.height, (int) maxVelocity);

\_maxVelocity = velBuffer.width/tileWidth;

}

else {

tileWidth = divRoundUp (velBuffer.width, (int) maxVelocity);

tileHeight = divRoundUp (velBuffer.height, (int) maxVelocity);

\_maxVelocity = velBuffer.width/tileWidth;

}

RenderTexture tileMax = RenderTexture.GetTemporary (tileWidth, tileHeight, 0, rtFormat);

RenderTexture neighbourMax = RenderTexture.GetTemporary (tileWidth, tileHeight, 0, rtFormat);

velBuffer.filterMode = FilterMode.Point;

tileMax.filterMode = FilterMode.Point;

neighbourMax.filterMode = FilterMode.Point;

if (noiseTexture) noiseTexture.filterMode = FilterMode.Point;

source.wrapMode = TextureWrapMode.Clamp;

velBuffer.wrapMode = TextureWrapMode.Clamp;

neighbourMax.wrapMode = TextureWrapMode.Clamp;

tileMax.wrapMode = TextureWrapMode.Clamp;

// calc correct viewprj matrix

CalculateViewProjection ();

// just started up?

if (gameObject.activeInHierarchy && !wasActive) {

Remember ();

}

wasActive = gameObject.activeInHierarchy;

// matrices

Matrix4x4 invViewPrj = Matrix4x4.Inverse (currentViewProjMat);

motionBlurMaterial.SetMatrix ("\_InvViewProj", invViewPrj);

motionBlurMaterial.SetMatrix ("\_PrevViewProj", prevViewProjMat);

motionBlurMaterial.SetMatrix ("\_ToPrevViewProjCombined", prevViewProjMat \* invViewPrj);

motionBlurMaterial.SetFloat ("\_MaxVelocity", \_maxVelocity);

motionBlurMaterial.SetFloat ("\_MaxRadiusOrKInPaper", \_maxVelocity);

motionBlurMaterial.SetFloat ("\_MinVelocity", minVelocity);

motionBlurMaterial.SetFloat ("\_VelocityScale", velocityScale);

motionBlurMaterial.SetFloat ("\_Jitter", jitter);

// texture samplers

motionBlurMaterial.SetTexture ("\_NoiseTex", noiseTexture);

motionBlurMaterial.SetTexture ("\_VelTex", velBuffer);

motionBlurMaterial.SetTexture ("\_NeighbourMaxTex", neighbourMax);

motionBlurMaterial.SetTexture ("\_TileTexDebug", tileMax);

if (preview) {

// generate an artifical 'previous' matrix to simulate blur look

Matrix4x4 viewMat = \_camera.worldToCameraMatrix;

Matrix4x4 offset = Matrix4x4.identity;

offset.SetTRS(previewScale \* 0.3333f, Quaternion.identity, Vector3.one); // using only translation

Matrix4x4 projMat = GL.GetGPUProjectionMatrix (\_camera.projectionMatrix, true);

prevViewProjMat = projMat \* offset \* viewMat;

motionBlurMaterial.SetMatrix ("\_PrevViewProj", prevViewProjMat);

motionBlurMaterial.SetMatrix ("\_ToPrevViewProjCombined", prevViewProjMat \* invViewPrj);

}

if (filterType == MotionBlurFilter.CameraMotion)

{

// build blur vector to be used in shader to create a global blur direction

Vector4 blurVector = Vector4.zero;

float lookUpDown = Vector3.Dot (transform.up, Vector3.up);

Vector3 distanceVector = prevFramePos-transform.position;

float distMag = distanceVector.magnitude;

float farHeur = 1.0f;

// pitch (vertical)

farHeur = (Vector3.Angle (transform.up, prevFrameUp) / \_camera.fieldOfView) \* (source.width \* 0.75f);

blurVector.x = rotationScale \* farHeur;//Mathf.Clamp01((1.0ff-Vector3.Dot(transform.up, prevFrameUp)));

// yaw #1 (horizontal, faded by pitch)

farHeur = (Vector3.Angle (transform.forward, prevFrameForward) / \_camera.fieldOfView) \* (source.width \* 0.75f);

blurVector.y = rotationScale \* lookUpDown \* farHeur;//Mathf.Clamp01((1.0ff-Vector3.Dot(transform.forward, prevFrameForward)));

// yaw #2 (when looking down, faded by 1-pitch)

farHeur = (Vector3.Angle (transform.forward, prevFrameForward) / \_camera.fieldOfView) \* (source.width \* 0.75f);

blurVector.z = rotationScale \* (1.0f- lookUpDown) \* farHeur;//Mathf.Clamp01((1.0ff-Vector3.Dot(transform.forward, prevFrameForward)));

if (distMag > Mathf.Epsilon && movementScale > Mathf.Epsilon) {

// forward (probably most important)

blurVector.w = movementScale \* (Vector3.Dot (transform.forward, distanceVector) ) \* (source.width \* 0.5f);

// jump (maybe scale down further)

blurVector.x += movementScale \* (Vector3.Dot (transform.up, distanceVector) ) \* (source.width \* 0.5f);

// strafe (maybe scale down further)

blurVector.y += movementScale \* (Vector3.Dot (transform.right, distanceVector) ) \* (source.width \* 0.5f);

}

if (preview) // crude approximation

motionBlurMaterial.SetVector ("\_BlurDirectionPacked", new Vector4 (previewScale.y, previewScale.x, 0.0f, previewScale.z) \* 0.5f \* \_camera.fieldOfView);

else

motionBlurMaterial.SetVector ("\_BlurDirectionPacked", blurVector);

}

else {

// generate velocity buffer

Graphics.Blit (source, velBuffer, motionBlurMaterial, 0);

// patch up velocity buffer:

// exclude certain layers (e.g. skinned objects as we cant really support that atm)

Camera cam = null;

if (excludeLayers.value != 0)// || dynamicLayers.value)

cam = GetTmpCam ();

if (cam && excludeLayers.value != 0 && replacementClear && replacementClear.isSupported) {

cam.targetTexture = velBuffer;

cam.cullingMask = excludeLayers;

cam.RenderWithShader (replacementClear, "");

}

}

if (!preview && Time.frameCount != prevFrameCount) {

// remember current transformation data for next frame

prevFrameCount = Time.frameCount;

Remember ();

}

source.filterMode = FilterMode.Bilinear;

// debug vel buffer:

if (showVelocity) {

// generate tile max and neighbour max

//Graphics.Blit (velBuffer, tileMax, motionBlurMaterial, 2);

//Graphics.Blit (tileMax, neighbourMax, motionBlurMaterial, 3);

motionBlurMaterial.SetFloat ("\_DisplayVelocityScale", showVelocityScale);

Graphics.Blit (velBuffer, destination, motionBlurMaterial, 1);

}

else {

if (filterType == MotionBlurFilter.ReconstructionDX11 && !fallbackFromDX11) {

// need to reset some parameters for dx11 shader

dx11MotionBlurMaterial.SetFloat ("\_MinVelocity", minVelocity);

dx11MotionBlurMaterial.SetFloat ("\_VelocityScale", velocityScale);

dx11MotionBlurMaterial.SetFloat ("\_Jitter", jitter);

// texture samplers

dx11MotionBlurMaterial.SetTexture ("\_NoiseTex", noiseTexture);

dx11MotionBlurMaterial.SetTexture ("\_VelTex", velBuffer);

dx11MotionBlurMaterial.SetTexture ("\_NeighbourMaxTex", neighbourMax);

dx11MotionBlurMaterial.SetFloat ("\_SoftZDistance", Mathf.Max(0.00025f, softZDistance) );

dx11MotionBlurMaterial.SetFloat ("\_MaxRadiusOrKInPaper", \_maxVelocity);

// generate tile max and neighbour max

Graphics.Blit (velBuffer, tileMax, dx11MotionBlurMaterial, 0);

Graphics.Blit (tileMax, neighbourMax, dx11MotionBlurMaterial, 1);

// final blur

Graphics.Blit (source, destination, dx11MotionBlurMaterial, 2);

}

else if (filterType == MotionBlurFilter.Reconstruction || fallbackFromDX11) {

// 'reconstructing' properly integrated color

motionBlurMaterial.SetFloat ("\_SoftZDistance", Mathf.Max(0.00025f, softZDistance) );

// generate tile max and neighbour max

Graphics.Blit (velBuffer, tileMax, motionBlurMaterial, 2);

Graphics.Blit (tileMax, neighbourMax, motionBlurMaterial, 3);

// final blur

Graphics.Blit (source, destination, motionBlurMaterial, 4);

}

else if (filterType == MotionBlurFilter.CameraMotion) {

// orange box style motion blur

Graphics.Blit (source, destination, motionBlurMaterial, 6);

}

else if (filterType == MotionBlurFilter.ReconstructionDisc) {

// dof style motion blur defocuing and ellipse around the princical blur direction

// 'reconstructing' properly integrated color

motionBlurMaterial.SetFloat ("\_SoftZDistance", Mathf.Max(0.00025f, softZDistance) );

// generate tile max and neighbour max

Graphics.Blit (velBuffer, tileMax, motionBlurMaterial, 2);

Graphics.Blit (tileMax, neighbourMax, motionBlurMaterial, 3);

Graphics.Blit (source, destination, motionBlurMaterial, 7);

}

else {

// simple & fast blur (low quality): just blurring along velocity

Graphics.Blit (source, destination, motionBlurMaterial, 5);

}

}

// cleanup

RenderTexture.ReleaseTemporary (velBuffer);

RenderTexture.ReleaseTemporary (tileMax);

RenderTexture.ReleaseTemporary (neighbourMax);

}

void Remember () {

prevViewProjMat = currentViewProjMat;

prevFrameForward = transform.forward;

prevFrameUp = transform.up;

prevFramePos = transform.position;

}

Camera GetTmpCam () {

if (tmpCam == null) {

string name = "\_" + \_camera.name + "\_MotionBlurTmpCam";

GameObject go = GameObject.Find (name);

if (null == go) // couldn't find, recreate

tmpCam = new GameObject (name, typeof (Camera));

else

tmpCam = go;

}

tmpCam.hideFlags = HideFlags.DontSave;

tmpCam.transform.position = \_camera.transform.position;

tmpCam.transform.rotation = \_camera.transform.rotation;

tmpCam.transform.localScale = \_camera.transform.localScale;

tmpCam.GetComponent<Camera>().CopyFrom(\_camera);

tmpCam.GetComponent<Camera>().enabled = false;

tmpCam.GetComponent<Camera>().depthTextureMode = DepthTextureMode.None;

tmpCam.GetComponent<Camera>().clearFlags = CameraClearFlags.Nothing;

return tmpCam.GetComponent<Camera>();

}

void StartFrame () {

// take only x% of positional changes into account (camera motion)

// TODO: possibly do the same for rotational part

prevFramePos = Vector3.Slerp(prevFramePos, transform.position, 0.75f);

}

static int divRoundUp (int x, int d)

{

return (x + d - 1) / d;

}

}

}