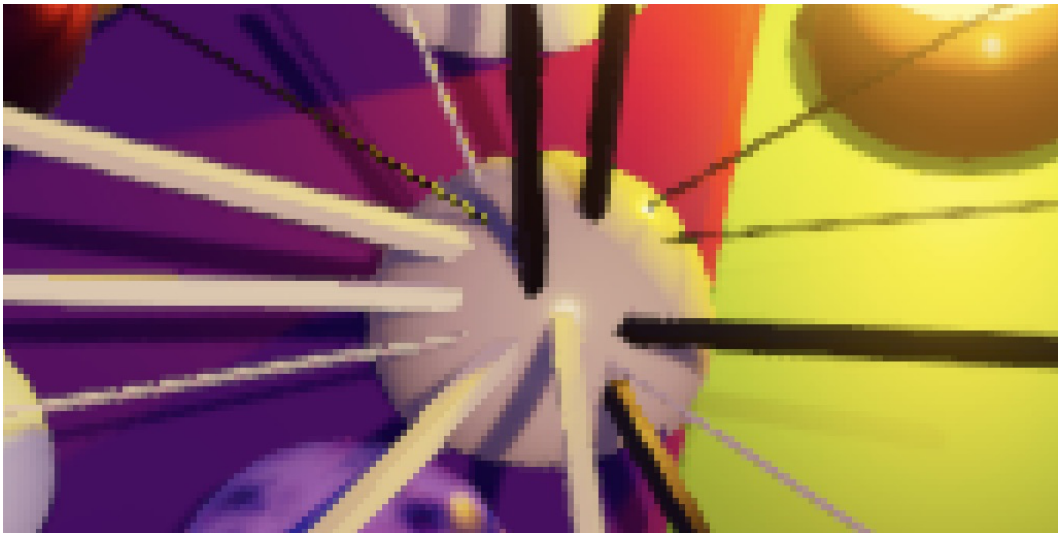
published 2023-08-15

Custom SRP 1.0.0 Modernization

Upgrade from Unity 2019 to 2022.
Modernize the project.
Introduce a few fixes.



FXAA was the last feature covered by the tutorial series.

1 Unity Version Upgrade

Before starting the project I have upgraded the entire tutorial series to Unity 2022.3.5f1. This required a few changes. I mentioned in the tutorial series where changes have been made during the upgrade. Then I modernized the project a bit. These changes are covered in the changelog of the 1.0.0 release and are also described below.

1.1 Baked Inner Cone Angle and Falloff

As was already mentioned when the Point and Spot Lights tutorial was first published, it wasn't possible to specify the inner light angle and falloff for spot light in `CustomRenderPipeline.lightsDelegate`, but in future Unity versions this should be possible. That future is now so it was added.

```
                case LightType.Spot:
                    var spotLight = new SpotLight();
                    LightmapperUtils.Extract(light, ref spotLight);
                    spotLight.innerConeAngle = light.innerSpotAngle * Mathf.Deg2Rad;
                    spotLight.angularFalloff =
                        AngularFalloffType.AnalyticAndInnerAngle;
                    lightData.Init(ref spotLight);
                    break;
```

1.2 Camera List

Unity developers realized that it isn't a good idea to allocate an array per frame to pass to `RenderPipeline.Render`, so they added the ability to use a list instead. The old `Render` method with an array still exists as an abstract method though, so we have to keep a dummy implementation of it in `CustomRenderPipeline`.

```
	protected override void Render (ScriptableRenderContext context, Camera[] cameras) {}

	protected override void Render (
		ScriptableRenderContext context, List<Camera> cameras
	) {
		for (int i = 0; i < cameras.Count; i++) {
			renderer.Render(
				context, cameras[i], cameraBufferSettings,
				useDynamicBatching, useGPUInstancing, useLightsPerObject,
				shadowSettings, postFXSettings, colorLUTResolution
			);
		}
	}
```

1.3 Explicit Batch Culling Projection Type

Unity 2022 asks us to explicitly tell us which batch culling projection type is needed when creating `ShadowDrawingSettings` in `Shadows`. We need to do this for all three light types, using orthographic mode for the directional light and perspective mode for point and spot lights.

```
		var shadowSettings =
			new ShadowDrawingSettings(
				cullingResults, light.visibleLightIndex,
				BatchCullingProjectionType.Orthographic
			) {
				useRenderingLayerMaskTest = true
			};
```

This parameter has been marked as obsolete in Unity 2023, so it will no longer be needed in the future. However, to avoid compilation warnings we do include them.

1.4 Scene Lighting Settings

Separate lighting settings assets didn't exist yet when the series was made, but they exist now and they've been included in the project. I also disabled automatic baking for the *Baked Light* scene so the maps can immediately be inspected. Each scene has also been moved into its own folder together with all its assets and the common materials have been placed in the *Scenes* folder as well.

1.5 Avoiding Domain and Scene Reloading

I have disabled domain and scene reloading when entering play mode. This is the fastest way to start play mode in the editor. This doesn't require any code changes but supporting the mode is important and always having it on ensures that it works.

1.6 Code Style Change

In more recent tutorials I have changed my C# code style to more closely match modern C# coding conventions, which also mostly matches Unity's modern style. I have also introduced usage of `readonly` modifiers where appropriate and shorthand constructor notation if it doesn't reduce clarity. These changes are not included in the code fragments shown in this tutorial.

2 Fixes

Some things needed to be fixed for the tutorials to function correctly or at all in Unity 2022. Some of these are specific to tile-based GPU, which weren't a big issue back then and I didn't have hardware to test with yet.

2.1 Missing Shader Matrices

Unity 2022's *Core Library* shader include files assume the existance of a few more matrices. In *Common.hlsl* we needed to add three extra macro definitions, for the inverse view matrix, and two matrices for the previous frame, which is used for TAA.

```
#define UNITY_MATRIX_M unity_ObjectToWorld
#define UNITY_MATRIX_I_M unity_WorldToObject
#define UNITY_MATRIX_V unity_MatrixV
#define UNITY_MATRIX_I_V unity_MatrixInvV
#define UNITY_MATRIX_VP unity_MatrixVP
#define UNITY_PREV_MATRIX_M unity_prev_MatrixM
#define UNITY_PREV_MATRIX_I_M unity_prev_MatrixIM
#define UNITY_MATRIX_P glstate_matrix_projection
```

And we need to add the accompanying matrices in *UnityInput.hlsl*.

```
float4x4 unity_MatrixVP;
float4x4 unity_MatrixV;
float4x4 unity_MatrixInvV;
float4x4 unity_prev_MatrixM;
float4x4 unity_prev_MatrixIM;
float4x4 glstate_matrix_projection;
```

2.2 Tangent-Space Vector Normalization

Unity used to normalize normal vectors when unpacking them after rescaling, but no longer does so. To keep our shaders working correctly we have to do this ourselves in *Common.hlsl*. This isn't the best approach but ensures that everything works as it should with the least amount of changes. In the future we could switch to a better approach for normal vectors.

```
float3 DecodeNormal (float4 sample, float scale) {
	#if defined(UNITY_NO_DXT5nm)
		return normalize(UnpackNormalRGB(sample, scale));
	#else
		return normalize(UnpackNormalmapRGorAG(sample, scale));
	#endif
}
```

2.3 Clear Color Before Skybox

It turns out that in some cases not clearing color when rendering a skybox can produce artifacts, caused by lingering `NaN` or `Inf` values in the frame buffer. We avoid that by always clearing the color unless explicitly told not to, in `CameraRenderer.Setup`.

```
		buffer.ClearRenderTarget(
			flags <= CameraClearFlags.Depth,
			flags <= CameraClearFlags.Color,
			flags == CameraClearFlags.Color ?
				camera.backgroundColor.linear : Color.clear
		);
```

2.4 Load Buffers for Reduced Viewports

Because tile-based GPU render per tile things can go weird when rendering to a viewport that covers less than the entire frame buffer. Unless the viewport exactly matches the tiles the GPU will end up rendering partially outside the viewport. It will mask this, but that requires the frame buffer data to be available. So we'll always use the load action when rendering with a reduced viewport, in the `DrawFinal` methods of both `CameraRenderer` and `PostFXStack`. We could be more specific but that gets rather convoluted and fragile, so we keep things simple.

```
	static Rect fullViewRect = new Rect(0f, 0f, 1f, 1f);

		…
		buffer.SetRenderTarget(
			BuiltinRenderTextureType.CameraTarget,
			finalBlendMode.destination == BlendMode.Zero && camera.rect == fullViewRect ?
				RenderBufferLoadAction.DontCare : RenderBufferLoadAction.Load,
			RenderBufferStoreAction.Store
		);
```

2.5 Copy Depth Twice for Gizmos

When the scene camera renders to intermediate buffers we have to copy depth to the camera target to make gizmos render correctly. To make this work in all cases `CameraRenderer` has to do this twice, both for pre-FX gizmos and again for post-FX gizmos. Note that I have been unable to find an example of pre-FX gizmos and know of no reason for their existence, so I'll merge them in a single post-FX step in the future.

```
	partial void DrawGizmosAfterFX () {
		if (Handles.ShouldRenderGizmos()) {
			if (postFXStack.IsActive)
			{
				Draw(depthAttachmentId, BuiltinRenderTextureType.CameraTarget, true);
				ExecuteBuffer();
			}
			context.DrawGizmos(camera, GizmoSubset.PostImageEffects);
		}
	}
```

2.6 Removal of Duplicate FXAA Iterator

The final fix is the removal of a duplicate iterator variable declaration in `GetEdgeBlendFactor` in *FXAAPass.hlsl*. While not critical it generates shader compiler warnings.

```
	int i;
	UNITY_UNROLL
	for (i = 0; i < EXTRA_EDGE_STEPS && !atEndP; i++) {
		…
	}
	if (!atEndP) {
		uvP += uvStep * LAST_EDGE_STEP_GUESS;
	}

	…

	UNITY_UNROLL
	for (i = 0; i < EXTRA_EDGE_STEPS && !atEndN; i++) {
		…
	}
	if (!atEndN) {
		uvN -= uvStep * LAST_EDGE_STEP_GUESS;
	}
```

The next tutorials is Custom SRP 2.0.0.

Enjoying the tutorials? Are they useful? Want more?

Please support me on Patreon!

Or make a direct donation!

made by Jasper Flick