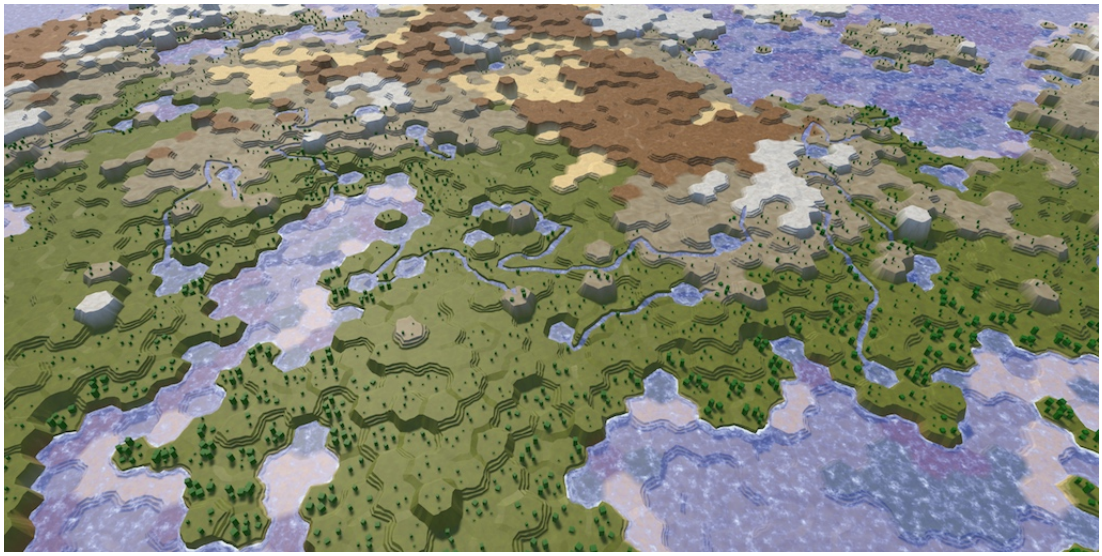
published 2023-04-11

Hex Map 2.0.0 URP

Switch to the Render Graph API.
Use passes to partition work.
Play nice with profiling.



Random maps can be generated, with biomes and rivers.

This tutorial is made with Unity 2021.3.11.f1 and follows Hex Map 27. It gives a summary of the conversion to URP.

1 URP Settings

Hex Map was originally made before SRPs existed, so it used what is now known as the Built-in RP. This release converts it to using URP.

This initial conversion from Built-in to URP translates the surface shaders verbatim to shader graphs with custom function nodes, keeping the changes to a minimum. This is the starting point from which the URP shaders can be improved in the future.

Import the *Universal RP* package and create a new URP settings asset, which will also create an accompanying renderer asset. Then adjust it to work well for our project.

1.1 Rendering

There is a single renderer, which I name the *URP Renderer* asset.

Only *SRP Batcher* is enabled, everything else is disabled. *Store Actions* are set to *Auto*.

Dynamic Batching can be considered legacy technology at this point. The SRP batcher does a good job, with the help of render queues to group renderers.

1.2 Quality

HDR is disabled, because Hex Map is entirely LDR.

Anti Aliasing (MSAA) is set to 4x and *Render Scale* is set to 1. You can of course change this to trade quality for performance.

1.3 Lighting

The *Main Light / Shadow Resolution* is set to 2048, which is used for two 1024 cascade maps. It could be lowered to improve performance.

Addition Lights are left at default values as Hex Map only uses a single directional light.

Reflection Probes aren't used thus its options are disabled. Likewise *Mixed Lighting* is disabled because there is no baked lighting. *Light Layers* aren't used either.

1.4 Shadows

Configure the directional shadows so they are visible quite far away, with higher resolution for close-ups and nearby high-altitude terrain. This is achieved with a *Max Distance* of 150, a *Cascade Count* of 2 and *Split* 1 set to 35%. *Last Border* is set to 5% to provide a small fade region. Note that, unlike what the URP documentation and inspector indicate, the fade region is a percentage of the entire shadow distance.

A *Depth Bias* of 1 is sufficient. *Normal Bias* must be zero otherwise holes appear in the terrain shadows.

Soft Shadows are enabled as they look better than hard shadows, but could be disabled to improve performance.

Conservative Enclosing Sphere is disabled as it results in far too many shadow casters being rendered beyond the max distance, making the last cascade mostly useless.

1.5 Post-Processing

Currently no post FX are used. *Volume Update Mode* is set to *Via Scripting* and is never updated, so volume system overhead is avoided.

1.6 URP Renderer

The single *URP Renderer Data* asset should be configured so a forward render path without depth priming is used. This approach work fine for the fairly simple Hex Map visuals.

Native RenderPass is disabled because this would only provide a benefit for TBDR GPUs when deferred rendering or similar complex approaches are used.

Transparent Receive Shadows is enabled so water and roads receive shadows.

Post-processing is disabled and there are no additional render features.

1.7 Main Camera

The single camera sits at the deepest level of the *Hex Map Camera* game object hierarchy in the scene. It is set up to use the rely on the URP settings and renderer assets, overriding nothing.

Note that *Rendering / Anti-aliasing* is set to *No Anti-aliasing*, but this refers to post-FX AA like FXAA or SMAA. The *MSAA* setting is found under *Output* because it affects the render target type.

1.8 Environment

Hex Map relies on a uniform solid background color to hide unexplored areas. Besides that the camera always looks down such that the sky is never visible. Thus *Background Type* is set to *Solid Color*. The *Background* color is RGB 68615B.

2 Material Conversion

The conversion from Built-in RP to URP means that the contents of the *Materials* folder changes a lot. The only exception is the *Highlights* subfolder, which stays the same because Unity UI is RP agnostic.

2.1 Shader Keywords

I rename the custom shader keywords. `GRID_ON` became `_SHOW_GRID` and `HEX_MAP_EDIT_MODE` gained an underscore prefix.

2.2 HLSL Files

Change the file extensions of *HexMetrics* and *HexCellData* from `cginc` to `hlsl`. The contents of *HexMetrics* remains the same. Adjust *HexCellData* so it uses new macros for texturing and its edit mode is controlled via a boolean function parameter, which are supplied via the shader graph based on keywords. Also make it use Unity's texture macros.

```
TEXTURE2D(_HexCellData);
SAMPLER(sampler_HexCellData);
float4 _HexCellData_TexelSize;

float4 FilterCellData (float4 data, bool editMode) {
	if (editMode) {
		data.xy = 1;
	}
	return data;
}

float4 GetCellData (float3 uv2, int index, bool editMode) {
	float2 uv;
	uv.x = (uv2[index] + 0.5) * _HexCellData_TexelSize.x;
	float row = floor(uv.x);
	uv.x -= row;
	uv.y = (row + 0.5) * _HexCellData_TexelSize.y;
	float4 data = SAMPLE_TEXTURE2D_LOD(_HexCellData, sampler_HexCellData, uv, 0);
	data.w *= 255;
	return FilterCellData(data, editMode);
}

float4 GetCellData (float2 cellDataCoordinates, bool editMode) {
	float2 uv = cellDataCoordinates + 0.5;
	uv.x *= _HexCellData_TexelSize.x;
	uv.y *= _HexCellData_TexelSize.y;
	return FilterCellData(
		SAMPLE_TEXTURE2D_LOD(_HexCellData, sampler_HexCellData, uv, 0), editMode
	);
}
```

All shader graphs will use two custom functions, one for the vertex stage and one for the fragment state. The matching HLSL functions are named `GetVertexCellData` and `GetFragmentData` with suffixes when needed. These are all put in a *CustomFunctions* HLSL file, except for the terrain and feature shaders graphs, which each have their own HSLS file.

As a reminder, per-cell visibility and exploration data is provided via a global texture and all materials except features rely on custom mesh vertex data. Cell indices are stored in UV2 UVW. Cell weights are stored in vertex color RGB. This information is used by `GetVertexCellData` to determine visibility per vertex, which is a combination of observation and exploration status. It outputs four components for terrain and two for all other materials. Terrain also outputs terrain indices.

The input needed by `GetFragmentData` varies per shader graph, but it always requires the interpolated visibility data. It outputs a base color, plus separate alpha when needed, along with an exploration factor. This factor is used to hide unexplored cells. How that is done varies per shader graph.

3 Shader Graphs

Each material type has its own shader graph and their old surface shaders are removed. See the project repository for each graph.

3.1 Terrain

The terrain is most complex because it needs to blend terrain textures of up to three adjacent cells. It is an opaque material and hides unexplored cells by turning off all lighting and switching to an emissive color matching the background.

3.2 Feature

The feature material is the only one that doesn't rely on custom mesh data. It instead relies on the world position and a grid coordinates offset texture to retrieve per-vertex cell data. It hides unexplored parts the same way as the terrain does.

3.3 Road

The road material is transparent and sits on top of the terrain. To avoid Z fighting a vertex offset is added that pulls vertices a tiny bit toward the camera. Alpha and exploration are both used to fade out the roads. UV0 is used to control road opacity. Roads are drawn before all other transparent materials, using render queue *Transparent-10*.

3.4 Estuary, River, Water Shore, and Water

The water materials fade the same way as roads. UV0 and UV1 are used to control river flow and shore lines.

The different water materials are put in separate render queues, from *Transparent-9* to *Transparent-6*. This is done to help the SRP batcher be most efficient. It ensures that the SRP batcher doesn't switch back and forth between different materials. If everything was in the same queue batches would be broken due to depth sorting. The exact draw order doesn't matter, except that rivers should be drawn last because waterfalls are the only case of overlapping water.

The next tutorial is Hex Map 2.1.0.

Enjoying the tutorials? Are they useful? Want more?

Please support me on Patreon!

Or make a direct donation!

made by Jasper Flick