The Unity Rendering Pipeline

Kuba Cupisz

http://is.gd/RenderingPipeline_UniteAsia13

unity

Who am I?

- Kuba
  - graphics programmer
  - at Unity for... quite.some.time
  - worked on small game projects before

Topics

What Unity is good/bad at

Built-in shaders

Shader combinations

Per material keywords

Lit shader replace

Endless runner with light probes

DX 11

- Tessellation
- Random writes
- Volume textures

What is Unity good at

- Very flexible rendering pipeline
  - vertex lit / forward / deferred
  - custom shaders
  - custom lighting models
  - completely customizable

unity

What is Unity bad at

- Very flexible rendering pipeline
  - difficult to configure (so.many.options)
- Some parts are better maintained then others
  - dual lightmaps in forward
- Default settings do not suit all projects
  - very general, scale from mobile to high end
  - aim your configuration at your target

Built-in shaders - good for

- Work with all rendering configurations
  - forward / deferred, realtime lighting / lightmaps
- Support standard lighting models
  - lambert / blinn-phong
- Work on all platforms

Built-in shaders - not good for

- Stylized games
  - only provide ‘standard’ lighting models
- Super-performance
  - if you know how your game will look, you can write more specific (faster) shaders
- Size
  - always have many shader variants due to supporting many possible configurations

When to write your own?

- When the built-in shaders
  - are not custom enough
  - are not fast enough - don’t assume, profile!
  - you know exactly how you want your game to be rendered

Shader combinations

- Compile variants of your shader
  - to do different things based on keywords
  - shader keywords are set globally
    - can be overridden per material

Shader combinations

- Each line declares a set of defines
  - only one per line can be active at a time
- Make sure you enable one, otherwise Unity will pick one for you

Shader combinations

```
#pragma multi_compile AAA BBB
```

- two variants
  - one where AAA is defined
  - another where BBB is defined (and AAA is not)

Shader combinations

```
#pragma multi_compile AAA BBB
#pragma multi_compile CCC
```

- two variants
  - AAA CCC
  - BBB CCC

Shader combinations

```
#pragma multi_compile AAA BBB
#pragma multi_compile CCC DDD
```

- four variants
  - AAA CCC
  - BBB CCC
  - AAA DDD
  - BBB DDD

Per material keywords

- Shader keywords property on a material
  - array of strings
  - each string a keyword
- Write a custom editor to make it easy to use

Material inspector

- Extend the MaterialEditor class
- Override OnInspectorGUI()
  - remember to call base.OnInspectorGUI() !

Shader combinations example

- Surface shader
  - 2 defines
  - 1 for *darken* blend mode
  - 1 for *difference* blend mode
  - configured via material inspector

Shader combinations

- Applications
  - switch the shaders in your scene via a keyword
    - completely change the look of your game
  - in the future
    - one shader for: diffuse, specular, normal maps, etc.

Shader replace

- Objects normally get rendered with whatever material /shader is configured on them
- Using shader replace you can swap out the shader
  - still uses same material (so textures / properties will be the same as in the original rendering)
- Sub-shader is selected based on tag-matching

Shader replace - Tags

- When setting a replacement shader, you can set the tag
  - Camera.SetReplacementShader (Shader s, string tag)
  - Camera.RenderWithShader (Shader s, string tag)

Shader replace - Tags

- No tag specified?
  - all objects will be rendered with the replacement shader
  - uses the first subshader of the replacement shader

Shader replace - Tags

- Tag set?
  - the real object's shader is queried for the tag value
  - no matching tag?
    - not rendered
  - tag found?
    - subshader from the replacement shader selected which matches the tag

Tags

- Builtin, e.g. RenderType:
  - Opaque
  - Transparent
  - TransparentCutout
  - etc.
- Custom
  - whatever you want!

Lit-Shader replacement

- New in Unity 4.1
- Useful for
  - scene view enhancement
  - special effects
- How does it work?
  - just like normal shader replace!
  - but you can use shaders that have lighting passes (surface shaders!)

Endless runner with light probes

- The track is assembled from blocks
- Any block can be matched with any other block
- It’s not feasible to bake light probes for all the combinations of block setups

Endless runner with light probes

- Bake light probes for each block separately
- After baking, light probes are tetrahedralized
- When the player moves from one block to the other you want to switch to light probes for the new block
  - set Lightmapping.lightProbes

Smooth transition

- Just switching from one light probe set to another will give a pop
- Solution:
  - make sure that start and end light probe positions have the same layout
  - when loading the new probes, set the start probes in the new set to the end probes of the previous set

Smooth transition

- Transition distance
  - add another set of light probes relatively closely to the start probes to control the transition distance

Baked data is not movable

- Loaded light probes will show up at the positions where they were baked
  - in our case the blocks' pivots are at (0,0,0)
- If you'd use player's position to sample light probes you'd sample outside of the volume defined by the probes -- not good

Baked data is not movable

- Luckily you can set lightProbeAnchor on the renderer to a transform of choice
- Parent the transform under the player
- Set local offset to -currentBlocksOffset on block change

DX 11

- Gives you more flexibility
- Allows for techniques that were impossible to do on the GPU before and not feasible on the CPU

Tessellation

- What is tessellation
  - subdivision of geometry
  - adds more triangles
    - by default in the same plane as the original triangle, so doesn’t add more detail yet

Tessellation

- To get more detail, tessellate and:
  - displace by sampling a displacement texture, or
  - use Phong tessellation
    - inflates the geometry and smooths the silhouette

Tessellation with surface shaders

- Use tessellate:FunctionName modifier in the surface shader declaration
  - float FunctionName () { return tessAmount; }
  - built-in
    - UnityDistanceBasedTess
      - tessellate more close to the camera
    - UnityEdgeLengthBasedTess
      - tessellate big triangles more

Random writes

- Unordered Access View
  - for RenderTextures or ComputeBuffers
  - allows writing at any position within the buffer

Random writes

- RenderTexture
  - generally stores colors
  - bajilion different formats
    - ARGB32, ARGBHalf, ARGBFloat, R(8|Half|Float), etc.

Random writes

- ComputeBuffer (StructuredBuffer in DX11)
  - stores structs or simple types
  - could be color, of course
  - int, float, you name it

Random writes

- Create a ComputeBuffer from script
  - provide the count and the stride (size of one element in bytes)
- Initialize the contents using SetData()
- Set the buffer for a material using SetBuffer()

Random writes

- In the shader declare the buffer as
  - StructuredBuffer<your_data_type>
    - random reads
  - RWStructuredBuffer<your_data_type>
    - random reads and writes

Random writes example

- Image analysis
  - run a shader on the scene render
  - analyze each pixel and write to an output buffer at a location of your choice
    - you can also atomically increment the value
      - InterlockedAdd()
      - requires a buffer of ints or uints though

Volume textures

- You know how textures are normally in 2D?!
  - Now they are in 3D!
  - BOOM

Volume textures

- How does DX 11 come into play?
  - you can fill the volume texture from a compute shader really quickly (do it each frame?)

Volume textures example

- Fills in the volume texture based on dispatch thread ID*
  - combined thread and thread group index

* Read up on semantics on MSDN

Questions?

- Kuba @kubacupisz

Appendix

Multiple render targets

- In the shader do:
  - `void frag(v2f i, out float4 Colour0 : COLOR0, out float4 Colour1 : COLOR1)` or
  - make the pixel shader return a color array or
  - make the pixel shader return a struct with `COLOR0...COLORn` semantics

Multiple render targets

- In the script do: (javascript)

```
// rt1, rt2, rt3, rt4 are RenderTextures

// create an array of color buffers
var mrt4 : RenderBuffer[] = [ rt1.colorBuffer, rt2.colorBuffer, rt3.colorBuffer, rt4.colorBuffer ];

// set those color buffers and the depth buffer from the first render texture as the current target
Graphics.SetRenderTarget (mrt4, rt1.depthBuffer);

// draw your stuff!
[...]
```