Optimizing Tegra Apps and Games using Unity

Paul “Hodge” Hodgson (Manager, Tegra Developer Technologies)
What's to come?

- Generic issues with Unity based solutions.
- Can be applied to other engines.
- Based on analysis of many Tegra applications.
Know how you have spent your budget

- Use available tools both within and external to Unity
- Many possible bottlenecks
  - Vertex
  - Primitive
  - Fragment
  - Bandwidth
  - CPU
- Be aware of cumulative effects
- Spend your optimization budget wisely also
Use optimized triangle lists

- Effectively zero contribution
- Reduces three potential bottlenecks
  - Vertex transform and vertex cache re-use
  - Primitive count
  - Attribute bandwidth
- Important for multi-pass techniques
Use optimized triangle lists

- For imported meshes
  - Player settings -> Optimise mesh data
  - Import new asset -> Inspector -> Optimise mesh

- For dynamic meshes
  - Mesh.SetTriangles(triangles, submesh)
  - Mesh.Optimize()
Only clear what/when you need to

- Zero contribution is common case with HUD/overlays
- Saves bandwidth from clear
- Saves bandwidth from content by removing
  - Depth test
  - Depth write
Only clear what/when you need to

- For each camera
  - Camera->Inspector->Clear flags
- For each subshader

Shader “DepthIgnore Example" {
  SubShader {
    Pass {
      ZWrite off
      ZTest Always
      // Rest of shader
    }
  }
}
Use appropriate texture settings

- Near zero contribution depending on assets
- DXT
  - Compressed RGBA
  - GA compress normal maps
    - UnpackNormal (tex2D (_BumpMap, IN.uv_BumpMap))
- Mipmap
- Anisotropy
- Filter mode
Use appropriate texture settings

- Select texture -> Inspector
  - Filter mode
  - Format
  - Aniso level

- File -> Build settings -> Android -> Texture compression
Match shader cost to results

- Avoid uber-shaders
- Use GLSL ‘lowp’ precision where possible
  - Cg type ‘fixed’
- Move constant or near constant results to vertex shader
Render order optimizations

- Zero contribution
- Divide geometry appropriately
- Render largest occluders first
- Ensure skybox is rendered after all other opaque objects

Shader “LargestOccluder Example" {
    SubShader {
        Tags {"Queue" = " Geometry-1 " }
        Pass {
            // Rest of shader
        }
    }
}
Render order optimizations

- Consider depth pre-pass
  - Normally at \((\text{shadeCost} \times \text{fragments})\)
  - Opaque at \((0.5 \times \text{fragments}) + (\text{shadeCost} \times \text{visibleFragments})\)
  - Discards at \((\text{minDiscard} \times \text{fragments}) + (\text{shadeCost} \times \text{visibleFragments})\)
Render order optimizations

Shader “DepthPrepass Example” {
    SubShader {
        // Pass to render to the depth buffer only
        Pass {
            ColorMask 0
            // Rest of pre-pass shader
        }
        // Pass to shade only the finally visible opaque fragments
        Pass {
            ZWrite off
            ZTest Equal
            // Rest of shader
        }
    }
}
Questions?

- Paul “Hodge” Hodgson

- NVIDIA Developer Zone

- Next up in this room:
  - Stephen Jones with “Performance and Debugging Tools for High-performance Android Applications”