The Witness on Android Post Mortem

Denis Barkar
3 March, 2017
Starting Point

• The Witness is in active development by Thekla
• Designed for PC and PS4/Xbox One
• Custom game engine
• Small codebase:
  about 1500 C++ source files and 200 DX11 shader source files
• OpenGL renderer is implemented but not perfect
• Extraordinary gameplay
Visual component is important for gameplay
Goals of the Port

• PC quality shaders and effects
• Mobile quality textures, geometry and audio
• Input support for most controllers
• Cloud saves
• A mix of PC and console code/data
• Targeting AArch64
Porting Workflow

• Nsight Tegra

• Tegra System Profiler

• Tegra Graphics Debugger
Nsight Tegra, Visual Studio Edition

• Same projects as PC development environment
• Relatively fast because of small code base
• Java/Native debug
Middleware integration

• Straightforward port:
  • freetype
  • harfbuzz
  • ogg
  • vorbis
Bink - not very straightforward port
SSE to NEON

API translation layer:

```c
FORCE_INLINE float __mm_cvtss_f32(__m128 a)
{
    return vgetq_lane_f32(a, 0);
}

FORCE_INLINE __m128i __mm_set_epi32(int i3, int i2, int i1, int i0)
{
    int32_t __attribute__((aligned(16))) data[4] = { i0, i1, i2, i3 };  
    return vld1q_s32(data);
}

FORCE_INLINE int __mm_movemask_ps(__m128 a)
{
    static const uint32x4_t movemask = { 1, 2, 4, 8 }; 
    static const uint32x4_t highbit = { 0x80000000, 0x80000000, 0x80000000, 0x80000000 };  
    uint32x4_t t0 = vreinterpretq_u32_f32(a); 
    uint32x4_t t1 = vtstq_u32(t0, highbit);  
    uint32x4_t t2 = vandq_u32(t1, movemask); 
    uint32x2_t t3 = vorr_u32(vget_low_u32(t2), vget_high_u32(t2));  
    return vget_lane_u32(t3, 0) | vget_lane_u32(t3, 1);
}
```
HL SL to GL SL

HL SL source → HL SL2GL SL → GL SL source → GL SL optimizer → optimized GL SL source
Shaders

• Shader compilation is slow
• Each android app has shader cache
• Pre warm: compile and release all shaders at every startup
• Only first startup is slow
• Now every shader is “compiled” immediately
• Whitelist: The Witness has about 2000 shader permutations (including debug/developer etc), only 300 are used
Tegra Graphics Debugger

• Deep dive via links into resource, event, texture, buffer views
• Dynamic editing and experiments
• Frame scrubber
• Frame profiler
GLSL Optimizer is not perfect:

```c
int j = 0;
for (; (j < sy); (j++)) {
    int i = 0;
    for (; (i < sx); (i++)) {
        for (int j_3; j_3 < sy_5; j_3++) {
            for (int i_7; i_7 < sx_6; i_7++) {
```

became

```c
for (int j_3; j_3 < sy_5; j_3++) {
    for (int i_7; i_7 < sx_6; i_7++) {
```

good enough to be compiled without errors and warnings
GLSL Optimizer output:

tmpvar_200.w = 0.0;
tmpvar_200.xy = uv_198;
tmpvar_200.z = tmpvar_164.z;
average_166 = (average_166 + ((tmpvar_170 * tmpvar_176) * texture (sun_shadow_texture, tmpvar_200.xyz)));
vec2 uv_201;
vec2 tmpvar_202;
tmpvar_202.x = tmpvar_173;
tmpvar_202.y = tmpvar_179;
uv_201 = (base_uv_165 + (tmpvar_202 * uv_inv_scale.zw));
uv_201.y = (1.0 - uv_201.y);
vec4 tmpvar_203;
tmpvar_203.w = 0.0;
tmpvar_203.xy = uv_201;
tmpvar_203.z = tmpvar_164.z;
average_166 = (average_166 + ((7.0 * tmpvar_176) * texture (sun_shadow_texture, tmpvar_203.xyz)));
vec2 uv_204;
vec2 tmpvar_205;
tmpvar_205.x = tmpvar_174;
tmpvar_205.y = tmpvar_179;
uv_204 = (base_uv_165 + (tmpvar_205 * uv_inv_scale.zw));
uv_204.y = (1.0 - uv_204.y);
vec4 tmpvar_206;
tmpvar_206.w = 0.0;
tmpvar_206.xy = uv_204;
tmpvar_206.z = tmpvar_164.z;
average_166 = (average_166 + ((tmpvar_171 * tmpvar_176) * texture (sun_shadow_texture, tmpvar_206.xyz)));
average_166 = (average_166 * 0.00694444);
Float packing

• Packing: converting floating point values to bytes in range [0, 255]

```c
void * writePackedByte3(void * dst, const Bounding_Box &bbox, const float x, const float y, const float z) {
    uint8 * ptr = (uint8 *)dst;
    const Vector3 unpack_mesh_offs = bbox.min;
    const Vector3 unpack_mesh_scale = max((bbox.max - bbox.min), Vector3(0.1f));
    *ptr++ = nv::clamp(nv::iround(((x - unpack_mesh_offs.x) / unpack_mesh_scale.x) * 255.f), 0, 255);
    *ptr++ = nv::clamp(nv::iround(((y - unpack_mesh_offs.y) / unpack_mesh_scale.y) * 255.f), 0, 255);
    *ptr++ = nv::clamp(nv::iround(((z - unpack_mesh_offs.z) / unpack_mesh_scale.z) * 255.f), 0, 255);
    return ptr;
}
```

• Considering small/flat meshes to avoid divide by zero

• Sending bytes to shader program as GL_BYTE array

• Values are scaled to [0, 1] when vertex shader gets them
Floats packing

- Increasing scale when rendering to avoid some gaps in meshes
  
  ```cpp
  const Vector3 unpack_mesh_scale = max((bbox.max - bbox.min), Vector3(0.1f) * (256.f / 255.f));
  ```

- Unpacking: rescaling to original size
  
  ```cpp
  in vec4 attr_position;
  float3 unpack_position(float3 position) {
      return ((position * unpack_mesh_scale.xyz) + unpack_mesh_offs.xyz);
  }
  ```

- Same method is used to pack values in 2-byte integers
Float packing

How to pack sign into vector’s coordinates?

• Normalize vector
• Divide negative vector by two
• Unpack sign:

```cpp
float4 unpack_tangent(float4 tangent) {
    tangent.w = sign(dot(tangent.xyz, tangent.xyz) - 0.255f);
    tangent.xyz = normalize(tangent.xyz);
    return tangent;
}
```
Float packing

How to pack sign into vector’s coordinates?

• Normalize vector
• Divide negative vector by two
• Unpack sign:

```c
float4 unpack_tangent(float4 tangent) {
    tangent.w = sign(dot(tangent.xyz, tangent.xyz) - 0.255f);
    tangent.xyz = normalize(tangent.xyz);
    return tangent;
}
```

• Realize what would happen when you normalize zero vector
Float packing

Fixed version:

```cpp
float4 unpack_tangent(float4 tangent) {
    float d = dot(tangent.xyz, tangent.xyz);
    float s = sign(d - 0.56f);
    tangent.w = s * sign(d);
    tangent.xyz = tangent.xyz * (1.5f - 0.5f * s);
    return tangent;
}
```
Low Latency Audio

Usually, latency of normal audio mixer is about 200-300ms

But AudioFlinger can use fast mixer (10-20ms):

• Accessed only through OpenSL ES

• OpenSL AudioPlayer must match format of low level audio. Must be the same:
  • number of channels
  • sample rate
  • buffer size
Tegra System Profiler

• Sampling profiler, configurable rate and counters
• Top down/bottom up/flat
• Core migration
• Filter by thread
Sometimes shader can be compiled by driver during draw call
Memory issues

Extensive memory allocation causes:

• Blocks from memory management functions
• CPU downclocking
• Crash due to lack of virtual memory
Memory issues

What we fixed:

• Reduced memory allocations
  • Per frame allocations are very critical
  • Third party libraries and drivers should be considered too
  • Use stack when it’s possible
• Fixed memory leaks
• Used memory mapped files
OpenGL - Zero driver overhead

• Per-mesh vertex array object (ARB_vertex_array_object extension)
• Uniform buffer objects for shader parameters (ARB_uniform_buffer_object extension)
• Single parameter block was split in two parts: per_shader and per_item
• Direct state access (ARB_direct_state_access extension)
• True hardware instanced rendering
• Uploading textures via pixel buffer objects
• Bindless textures (NV_bindless_texture extension)
OpenGL - Zero driver overhead

• Unified memory for vertex buffer objects (NV_vertex_buffer_unified_memory extension)

• Unified memory for uniform buffer objects (NV_uniform_buffer_unified_memory extension)

• Multi-Draw Indirect (NV_bindless_multi_draw_indirect extension)

• Persistent mapped buffers (ARB_buffer_storage extension)
Tools summary

NVIDIA’s in-house porting tools can help you push game to the limits

Nsight Tegra

- Visual Studio efficiency, with cross-platform Android development
- C/C++ multi-platform projects, builds, and debugging

Tegra System Profiler

- Get the big picture! How are your algorithms, APIs, and system interacting?
- Find hotspots, thread, and hardware sync issues you never even expected

Tegra Graphics Debugger

- Debug complex graphics issues
- Analyze and tune shader performance
How to get it

Codeworks

https://developer.nvidia.com/codeworks-android

Simple single installer Android development environment

SDK, NDK, ANT, Gradle, NVIDIA tools, samples, drivers ...
Questions? Thank you

Denis Barkar
Tegra Game Porting Engineer
dbarkar@nvidia.com