From DX11 to Mobile: Development of a multi-platform Game Engine

Florian Born
Lead Software Engineer, Havok

KGC 2011

Overview

- About me
  - Co-founded Trinigy in 2003, Technical director
  - 2011: Acquisition of Trinigy through Havok
  - Working as a Lead Engineer in the Vision Engine product team

Overview

- Vision Mobile: Introduction
- Vision Mobile: The Technology
  - Shader Authoring
  - Cross-platform Material Handling
  - Precomputed illumination
  - Visibility Culling
  - Input Management
  - Multithreading
- Vision Mobile: Future Features

Vision Game Engine

Platform support

Vision Game Engine

- PC and Console versions of Vision are widespread
  - High performance across all platforms
    - Cross-platform multithreading & stream processing
    - Many platform-specific optimizations “behind the scenes”
    - Full access to low-level features if desired
  - Highly modular
  - Streamlined workflow
    - Multi-platform development as a matter of recompilation
  - Core philosophy: Freedom is key

Vision Game Engine

[Demo]

Vision Mobile

- Mission:
  - Make the “Vision Experience” available on Mobile Devices
- Key features:
  - C++ API, fully compatible with PC and console versions
    - LUA as optional scripting language
  - Fully compatible with the Vision Tools Environment
  - Low Memory and CPU overhead
  - Fully support platform-specific features
  - Excellent visuals
  - Support for “all” mobile platforms, including mobile consoles!

- *One Engine for all Platforms*

Demo Stats

- Original Demo Material from PC/PS3/Xbox360 Demo
  - Demo scene shipped as part of the Vision SDK
  - >250k triangles per frame
  - Complex HLSL/CG shaders (Parallax Mapping, Rim Lighting, etc.)
  - Characters with 10k+ triangles
  - Highres textures
- Current iOS and Android platforms have limited resources
  - GPU performance
  - CPU performance
  - RAM
  - -> Platform-specific solutions without breaking compatibility

Vision Mobile: Shader Authoring

- OpenGL ES platforms use GLSL
  - But Vision Shaders are authored in HLSL/CG
  - Two separate languages not acceptable
- Solution: Automatic code conversion
  - Same shader code on all platforms
  - No need to juggle between GLSL and HLSL
  - Shader code is automatically converted from HLSL to GLSL at export time
  - Unified engine API for managing/modifying shaders at runtime
- Full support for Visual Shader Editor (VSE)
  - Convenient artist-driven Shader Editing

Vision Mobile: Shader Authoring

- Automatic HLSL to GLSL conversion
  - hlsl2glslfork
  - Originally developed by ATI,
  - now freeware
  - http://sourceforge.net/projects/hlsl2glsl/

hlsl2glslfork

```
float4x4 matWVP : register(c8);

struct VS_IN
{
   float3 ObjPos   : POSITION;
   float2 UV0 : TEXCOORD0;
};

struct VS_OUT
{
   float4 ProjPos  : POSITION;
   float2 UV0 : TEXCOORD0;
};

VS_OUT vs_main( VS_IN In )
{
   VS_OUT Out;
   Out.ProjPos = mul( matWVP,
float4(In.ObjPos,1.0) );
   Out.UV0 = In.UV0;
   return Out;
}
```

```
precision highp float;

struct VS_OUT
{
  highp vec4 ProjPos;
  highp vec2 UV0;
};

struct VS_IN
{
  highp vec3 ObjPos;
  highp vec2 UV0;
};

uniform highp mat4 matWVP;

VS_OUT vs_main( in VS_IN In )
{
  VS_OUT Out;
  Out.ProjPos = ( matWVP * vec4( In.ObjPos,
1.00000) );
  Out.UV0 = In.UV0;
  return Out;
}

attribute vec3 position;
attribute vec2 tex0;
varying highp vec2 xlv_TEXCOORD0;

void main()
{
  VS_OUT xl_retval;
  VS_IN xlt_In;
  xlt_In.ObjPos = vec3( position);
  xlt_In.UV0 = vec2( tex0);
  xl_retval = vs_main( xlt_In);
  gl_Position = vec4( xl_retval.ProjPos);
  xlv_TEXCOORD0 = vec2( xl_retval.UV0);
}
```

Vision Mobile: Shader Authoring

- Hlsl2glslfork works for DX9-style shaders
  - See what we do in the future when devices reach DX10/11 feature level

- HLSL shaders are pre-compiled to byte code
- GLSL shaders are compiled at application runtime
  - Shaders are simple enough that compile time is negligible

Vision Mobile: Shader Authoring

- Full support for Visual Shader Editor (VSE)
  - Convenient artist-driven Shader Editing
  - …but use with care as every instruction counts
  - E.g. iPad1: No fancy stuff - shouldn't do more than diffuse*lightmap
- But we will see rapid development of hardware
  - Outscoring Moore's Law on the GPU side

Cross-platform Materials (I)

- Performance
  - Materials on PC/consoles are pretty complex
    - Normal/Parallax/Specular/Glow Maps, Rim Lighting,…
    - Relatively heavy on the GPU side
  - Mobile Platforms are getting faster
    - …but aren‘t quite there yet
    - Only Sony PSVITA allows us to use full-blown PC shaders
  - Far too slow on iPad/iPhone 3GS
    - Every instruction counts
    - -> Separate Default Shader/Material Set on iOS

Cross-platform Materials (II)

- Assigning Shaders in the Editor: „AUTO“ vs. „MANUAL“
  - „Shader Provider“ decides about effect assignment in AUTO mode
  - Decision is made based on material properties
    - Maps (Diffuse, Specular, Normal, …)
    - Properties (Ambient Color, Blending, …)
- Shader Provider are easily customizable in C++
  - …and you can switch between them in the editor
- Separate Default Shader Providers on iOS & Android
  - Separate Shader Sets
    - Support for different graphics features
    - e.g. Dot3 Lightmapping, Parallax Mapping, Specular
    - Corresponding Shaders geared towards producing similar results

Cross-platform Materials (III)

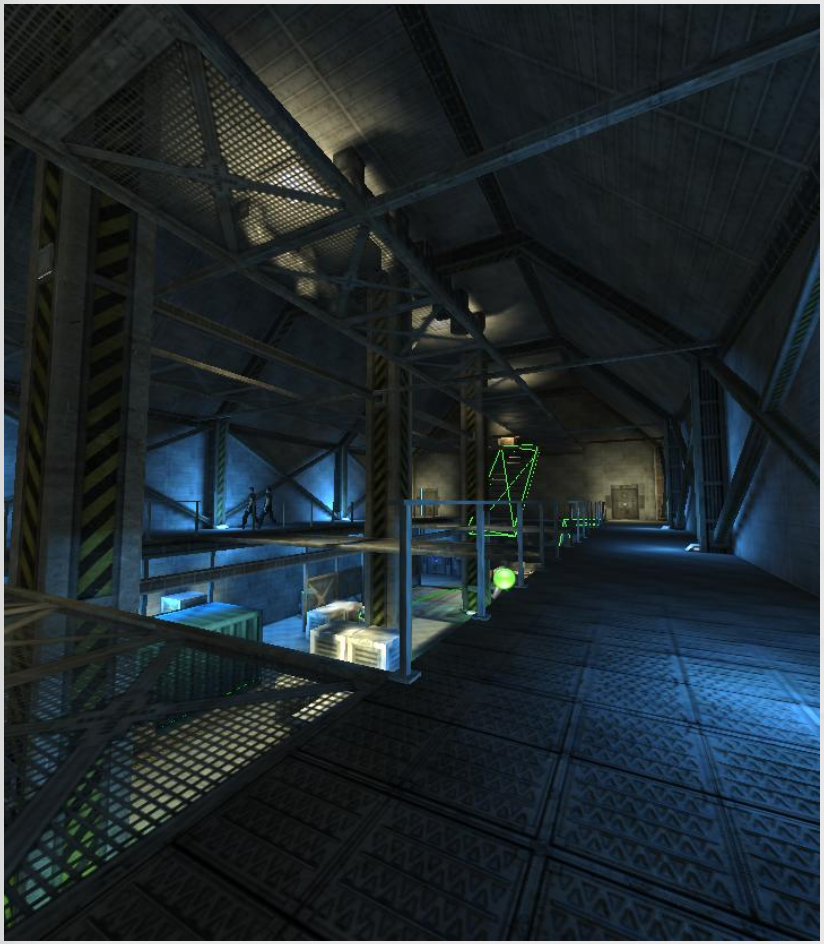
Features supported by the respective platform's default shader provider:

Feature	PC/Consoles/VITA	iPad 1	iPad 2
Normal Mapping	X	X	X
Parallax Mapping	X		
Specular	X		X
Dot3 Lightmapping	X		X
Rim Lighting	X		
Glow Maps	X	X	X
Reflections	X	X	X

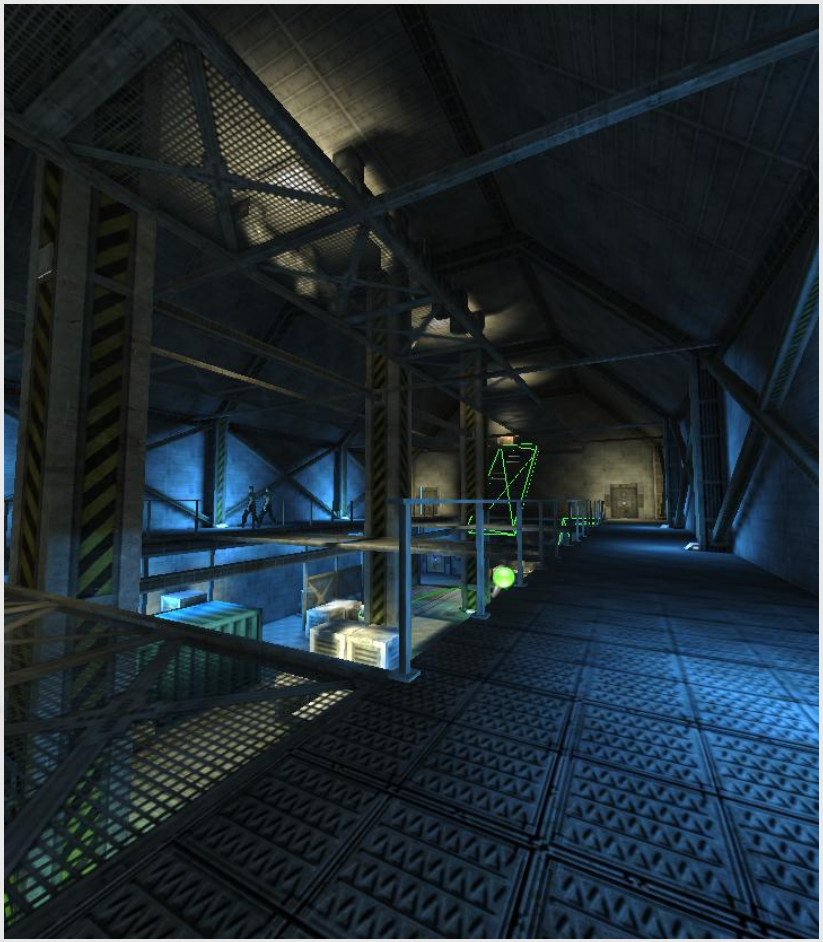
Cross-platform Materials (IV)

- Through the programmable „Shader provider“ class we can easily simulate Mobile Visuals on PC
  - Basically just use the reduced shaders
  - All written in HLSL anyway

Cross-platform Materials (V)



Scene in Standard Mode



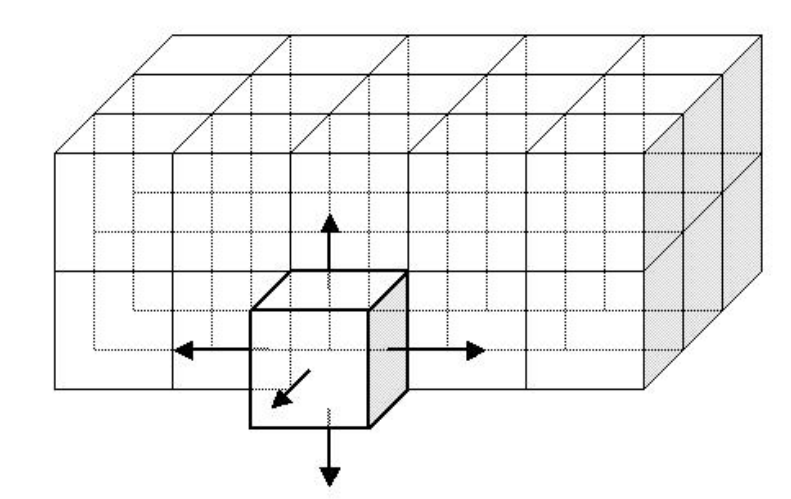
Scene in „Mobile Mode“ (iPad 1)

Precomputed Illumination (I)

- Static illumination in the Vision Engine:
  - Full Radiosity computation
  - Dot3 Lightmapping
  - Light Grids (light probes)
  - Resolution of Lightmaps and Light Grids can vary across the scene
  - Can be arbitrarily combined with dynamic illumination

Precomputed Illumination (II)

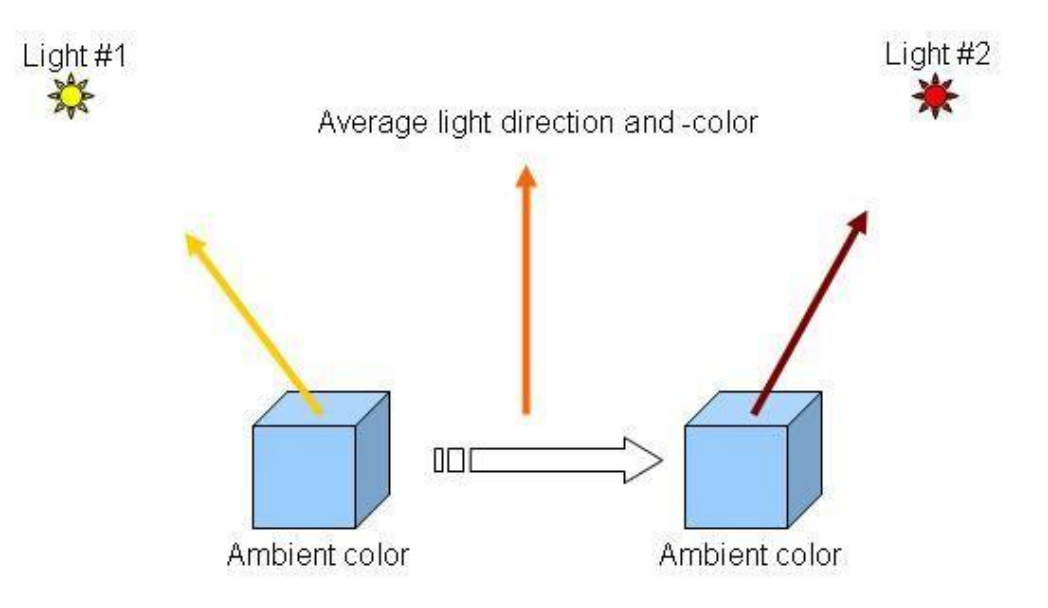
- Light Grids
  - Used for lighting dynamic geometry
  - „Full Version“: Six incoming light directions (main axes)



  - Can be used on iPad 2, but too expensive on iPad 1/iPhone 3GS

Precomputed Illumination (III)

- „Simple Light Grid“ on iPad 1, Android, Wii:
  - One primary direction plus Ambient
  - Very efficient, looks surprisingly good



Precomputed Illumination (IV)

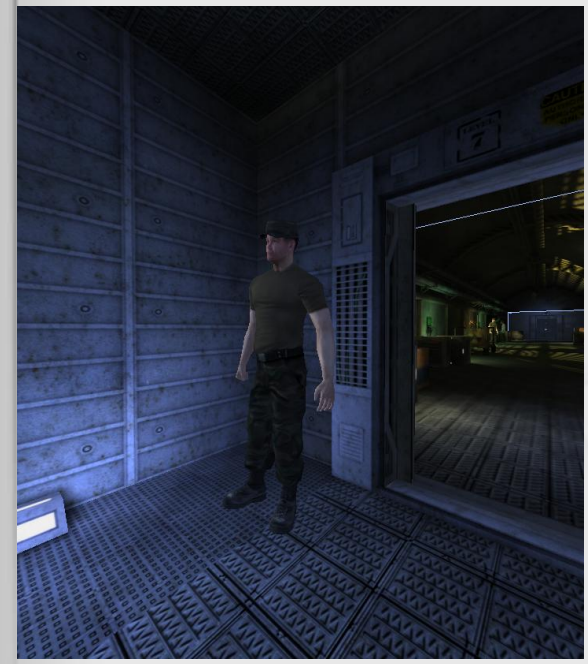
- Lightmaps
  - Dot3 Lightmapping on iPad 2 / Tegra 2
    - too expensive on iPad 1
  - iPad 1: Primary light direction instead of 3 orthogonal ones
    - Diffuse Normal Mapping
    - plus one ambient value per lightmap sample
    - 60fps!
  - No pre-baked textures
    - Data can remain compact
    - No need for cumbersome 500MB+ downloads!
    - Vision iPad 1 demo is around 25 MB!

Visibility Determination / Culling (I)

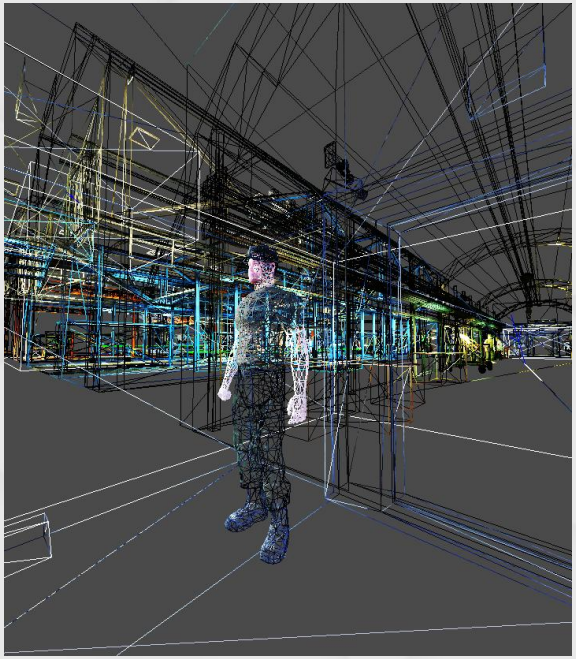
- Default in Vision: Portal System, Occlusion Queries
  - Visibility Zones
  - Object grouping
  - Specialized solution for terrain and vegetation objects

- Occlusion Queries not supported on many mobile devices
  - But: Portal System is extremely efficient for indoor areas
    - A lot better than PVS…
  - Minimal CPU Impact: <0,5ms/frame
  - Very high culling rates
  - Portals can be set up in 3DS Max or Maya

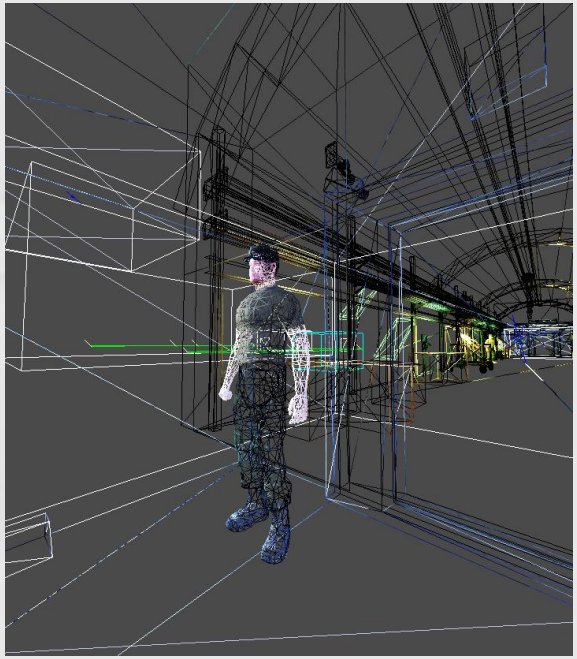
Visibility Determination / Culling (II)



Hangar Scene



Hangar Scene
~80k Tris rendered
wireframe w/o portals



Hangar Scene
<10k Tris rendered
wireframe with portals

Input Management (I)

- Mobile Platforms have „different“ controls
  - PSVITA has gamepad-like controls, but also multitouch + touch panel
  - Tablets have Multitouch, Accelerometers,…
  - Handled all through the same input abstraction system in Vision
    - ‘ifdef’ only in the initialization, not in the game code

Input Management (II)

- Multitouch:
  - Can define separate “Touch Areas” on the touch panel/screen
    - Each has a priority assigned, since touch areas may overlap
  - When touch occurs in a specific area, the new touchpoint is assigned to this Touch Area
    - If multiple Touch Areas overlap the touchpoint, the one with the highest prio is selected
    - Touchpoints are tracked while they are in a specific Touch Area

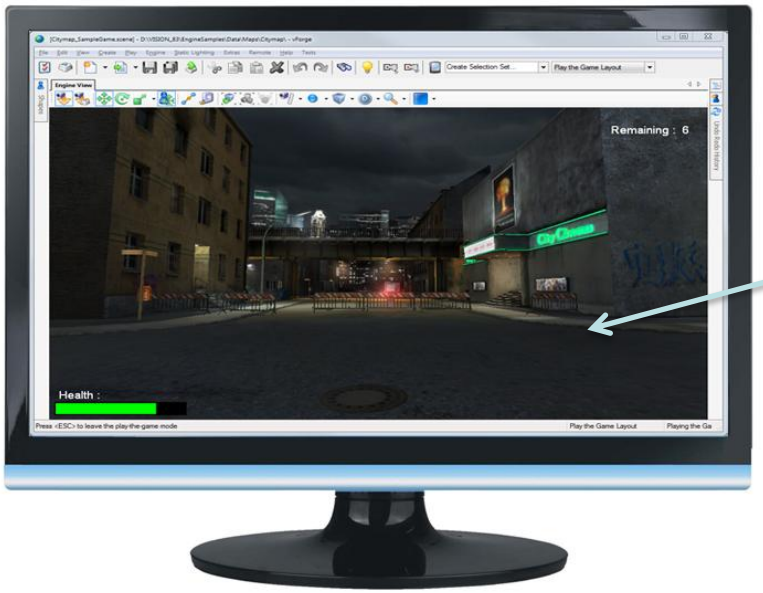
Input Management (III)

- Cross-platform handling:
  - Input events and Axes can be mapped to arbitrary devices
  - Thus, a touchpad can be treated the same way as e.g. a mouse
  - …which has multiple axes and therefore can be treated in a similar way as a joystick

Input Management (IV)

- Remote Input handling
  - Rather than deploying game on the device…
    - …run game on PC (same shaders, same screen resolution)
    - …and pipe input from device to PC
  - Device runs a HTML5 website
    - Optionally displays static image of the controls
  - PC runs a mini-webserver (freeware: mongoose)
  - Communication in both directions
    - HTML5 website can respond to game state changes
  - Nice side effect:
    - Devices that are not yet officially supported can be tested

Input Management (V)



Interactive game, webserver

Static HTML5 website

Multithreading

- Multicore
  - Fully supported on iPad2, Tegra2, …
  - Processes Animation, Mesh Deformation, Visibility Culling, Particles, Physics, … on n Cores
  - Significant performance gain on multi-core tablets/phones and PSVITA
  - Two systems to easily create custom multi-core functionality:
    - VThreadManager
      - Thread Pool approach
      - Supports priorities & task dependencies
    - VStreamProcessor
      - Cross-platform Stream processing system
      - Also supports PS3 SPUs

Vision Mobile: Future Features

- Android Support (Q4/2011)
- On-target preview with full input support
- Automatic selection of full- vs. half-precision floats in shader code

Questions?