High-Level Game Development for PlayStation2

Hemal Bodasing (SCEE)

Overview

- Problems with starting PS2 development
- Overview of high-level library (HiG) + demos
- HiG Art production pipeline
- HiG Usability issues
- Game demo
- Further work

Introduction

- Will NOT talk about:
  - Algorithms used for specific effects
  - PS2 hardware architecture details
  - Low-level PS2 optimisation techniques

Problems with starting PS2 development

Problems with starting PS2 development

- I found it ****** hard!
- No exporters or engine available as part of the standard libs (or so I thought)
- Lack of a standard graphics file format
- Forces you to optimise from the start of the development cycle (eg VU coding)
- Sample Code vs Game Code

HiG (High-level Graphics Library) Overview

HiG (High-level Graphics Library) Overview

- Developed by SCEI (Japan)
- Open-source (to PS2 developers)
- Target users:
  - Beginner-level PS2 users

HiG (High-level Graphics Library): Goals

HiG aims are:

- Usability
- Efficiency
- Extensibility

HiG (High-level Graphics Library): Goals

- Usability:
  - Abstraction
  - Reduce effort
- Efficiency
  - VU1-based

HiG (High-level Graphics Library): Goals

- Extensibility
  - Data format
  - Plugin architecture

Relationship with Middleware

- “Free” to all PS2 developers
- PS2-specific
- Less comprehensive than middleware
- HiG is currently purely a rendering solution
- HiG architecture designed to allow middleware companies to develop plugins
- No dedicated support channel

High-level Functionality

High-level Functionality

Fundamental operations:

- Loads mesh, texture and animation data
- Performs transformation (rotation, translation and perspective correction) and rendering
- Handles hierarchical meshes and keyframe animation

High-level Functionality

Advanced operations:

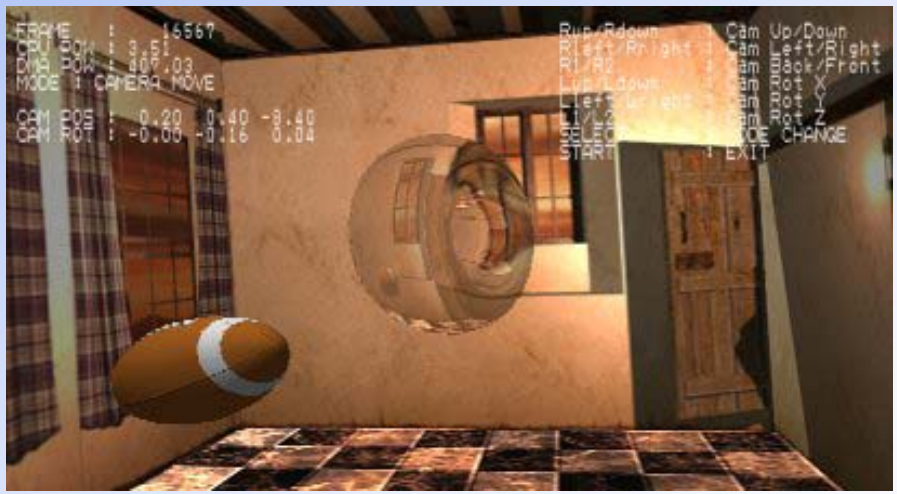
- Reflection
- Refraction
- Shadow-mapping
- Fish-eye lens rendering
- Clut-based bump-mapping

HiG Demos

- Shadow mapping
- Reflection mapping
- Real-time environment mapping

Plugin System

- A plugin system is used to implement most of HiG's functionality
- HiG allows you to create your own plugins
- Plugins can be used in combination, eg for multipass effects



Plugin System

- Fundamental:
  - shape (vertex + face data), microcode, texture, hierarchy, animation
- Advanced:
  - reflection, refraction, shadow-mapping, fish-eye lens rendering, clut-based bump-mapping

Low-level Functionality

- HiG ”Service” functions can be used to create your own plugins
- Service functionality covers:
  - Data format
  - Memory management
  - DMA
  - GS (Graphics Synthesiser)

Art Pipeline

- Exporters
- File formats
- Art Production Process

Art Pipeline - Exporters

- Exporters are available for 3DS Max, Maya and Lightwave
- Export to “ES” format

Art Pipeline - File formats

- All HiG file formats contain the following information:
  - Mesh data
  - Hierarchy data
  - Animation data
  - Texture data
  - Texture context data

Art Pipeline - File formats

- HiG File formats (all contain equivalent information):
  - .es : Readable ASCII
  - .s : Less readable ASCII
    ASM pre-processor
  - .bin : Binary - used in-game

Art Pipeline: Production Process

- Export from art package in .ES format
- If reqd, edit .es file to customise for specific features, eg shadow mapping. (Additional tools can be created to perform this)
- Use provided converter (esConv) to convert from .es to .bin file
- If problems exist with .bin file, .s file can be generated and inspected

HiG VCL Macro-Library

•HiG VCL Macro-Library

- What is VCL?
- What is the HiG macro-library?
- Macro-library functionality
- Macro-library implementation

What is VCL?

- Text pre-processor for VU (Vector Unit) micro code.
  - Note: Vector Units are are vector processors that perform high-speed floating-point operations such as vertex transformations
  - VU coding requires dual-pipeline assembler
- VCL takes a single stream of instructions
- Produces optimised vu code.
- Basic support for macros and structs

VCL Macro-Library

- Functionality:
  - Gives VU implementations of basic and advanced techniques (eg reflection / refraction mapping, real-time shadow mapping and fish-eye lens rendering)

VCL Macro-Library

- Implementation:
  - Macros used for individual units of functionality, eg transformation, lighting, backface culling
  - Each macro is about 10-20 lines long which makes for easy readability
  - Number of microprograms available which use the macros as building blocks
  - Excellent example of how VU code can be developed in a modular fashion
  - Useful for non-HiG developers to look at

Game Demo

Game Demo

- Objectives
- Design & Implementation
- HiG Usability issues
- Techniques used
- Stats and performance

Game Demo - Objectives

- To investigate the usability of HiG within a game architecture
- To produce a simple game demonstrating HiG's functionality
- If I can write this anyone can!

Game Demo - Design & Implementation

- Use of C++ for:
  - Modularity, re-usability, maintainability, etc
  - Map HiG functionality to game entities
- Fundamental classes for meshes, cameras, etc
- Instruction cache thrashing issue

HiG Usability - Problems Encountered

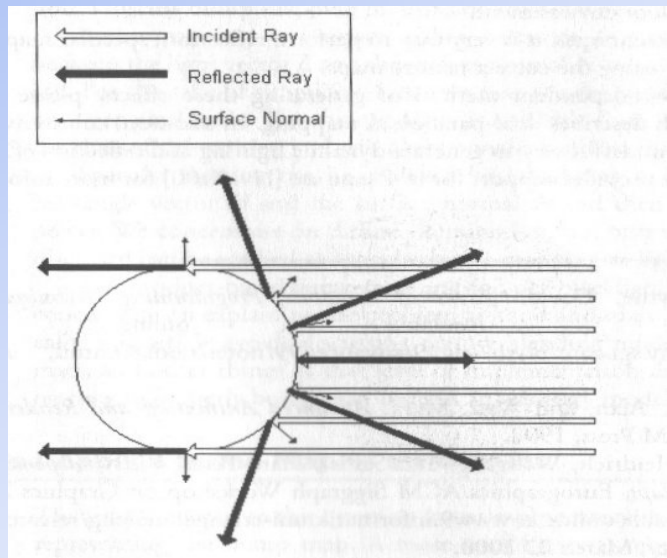
- Data generation (Art pipeline)
  - Much time spent resolving MAX exporter problems
  - Conversion to and from intermediate formats
- Documentation
  - Lack of English docs initially!
  - Consists mainly of a lib reference
  - Lack of “How to...” docs
  - Other sources of documentation are scattered
  - Have created a HiG overview doc which hopefully will address these issues

HiG Usability

- Using HiG in a game architecture
  - HiG works by building DMA chains
  - Doesn't map easily to game objects / entities
  - Mesh instancing
- Combining different functionality, eg reflections with shadow mapping

Game Demo - Techniques

- Problem with “true” reflection-mapping:
  - Area of the world seen in reflection is dependent upon the orientation of the reflecting object
  - Often this is the “uninteresting” area of the world!



Game Demo - Techniques

“Game reflection" effect:

- Real-time reflection map used with refraction microcode
- ”Interesting" area of the world can always be seen as a reflection, regardless of the orientation of the reflecting object
- Although the resulting effect is inaccurate, it works well in a game environment
- Small refraction index used to give discontinuity at edges
- Large missiles used for off-screen render!

ToDo: Game Demo - Techniques

- Static texture used for reflection effect on player ship
- Simple ship physics
  - Interpolation used to give smooth motion
- Skydome
  - Large hemisphere fixed relative to the camera
- Gameplay designed to show off effects
  - Shadow mapping: ship pitching and rolling
  - Reflection mapping: Large enemy character

Game Demo - Stats

Mesh	Num Triangles	Texture
Player ship	1163	256x256 32 bit
Enemy ship	368	256x256 32 bit
Landscape	8015	256x256 32 bit
Skydome	640	256x256 32 bit
Missile	66	Untextured

- Note: All models are triangle-stripped

HiG Microcode Performance

Micro name	Shading type	Back-face culling	Volume clip	Light type	Vertices/sec
vu1basicVo	vertex color	N	N	directional	15,668,800
vu1basicClip	vertex color	N	Y	directional	12,379,976
vu1cullVo	vertex color	Y	Y	directional	10,188,080
vu1fisheye	vertex color	Y	Y	directional	8,204,112
vu1reflectR	-	N	Y	-	7,057,624
vu1refractR	-	N	Y	-	7,057,624

Performance - PA Scan

Performance - PA Scan

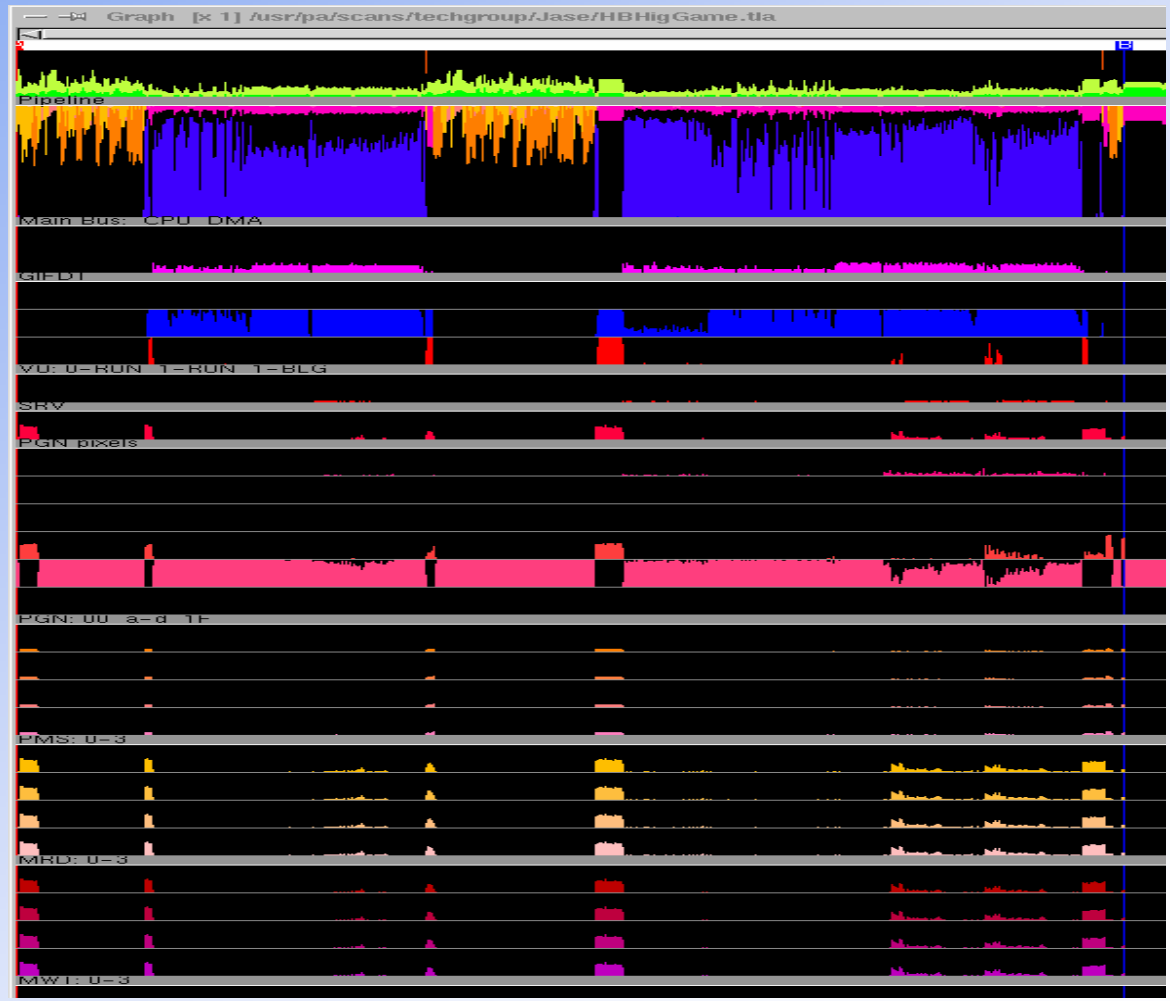
CPU cycles

Data Cache miss

DMA Transfer

VU activity (micromode)

GS Idle



Performance

- PA scans:
  - DMA bound
  - Sends all the data which may be reqd - XYZ, normal, ST, colour
  - Some data may be redundant, eg ST's not reqd for reflection / refraction mapping
  - Data is uncompressed
  - I-cache trashing

Performance

- Tip:
  - HiG clears DMA buffer every time a chain is sent
  - Fixed cost per frame
  - => Keep DMA buffer small (buffer size specified in sceHiDMAInit)
  - Game demo uses 0.5K

Further Work - Optimisation

- Application-level:
  - Use low level-of-detail meshes for off-screen render
  - Reduce texture sizes
  - Double buffering DMA chains
- Library-level:
  - VIF compression of mesh data
  - Currently DMA's 4 quadwords per vertex

Further Work - Functionality

- Hierarchical animation
- Scene culling
- Multiple shadows
- Bump mapping (clut-based)

Summary - 1

- VCL-macro library worthwhile examining
- Demo and doc on PS2 website soon
- Where to start: sce\ee\sample\graphics\hig

Summary - 2

- Game demo achieved in a few months
- Most of the problems I encountered have now been resolved
- By re-using code and tools, completely different demo using same techniques could be achieved in a few days

  => High-level libs do exist and are usable!