PVRGeoPOD Version 1.05

User Manual

Filename	:	PVRGeoPOD.User Manual.mht
Version	:	1.6f (PowerVR SDK 2.03.23.1153)
Issue Date	:	14 Aug 2008
Author	:	PowerVR

Contents

List of Figures

1. Introduction

PVRGeoPOD is a file-exporter plug-in for 3D Studio MAX and Maya. Some notable supported features are as follows:

- Parented nodes
- Mesh instancing
- Can export as many texture-coordinate sets as MAX and Maya supports
- Boned meshes
- Bone batching (can split a mesh into sub-meshes when the mesh has more bones than can fit into the bone palette)
- Complete choice of data formats (e.g. floats, bytes, …)
- Choice of interleaved vertex data or separate channels (e.g. position, normal, texture coordinates).
- Tangent space generation
- Polygon/vertex sorting
- Polygon stripping

1. Installation

1.1. 3ds max

1.1.1. 3ds Max Data Exchange Interface (3DXI) - previously known as IGame

PVRGeoPOD for 3ds max uses Autodesk's 3DXI interface, which relies on a file called "igame.dll" which resides in the 3ds max installation directory. This file is frequently updated by Service Packs for 3ds max, as and when Autodesk (and previously Discreet) releases them, and by specific updates they may release for 3DXI (previously IGame).

Unfortunately, the various versions of "igame.dll" are often incompatible, and only one may be installed at any point in time. It is therefore very difficult to use two different 3ds max plug-ins that both use 3DXI, but are compiled to use different versions of it.

PVRGeoPOD attempts to work around this issue by having each build of the plug-in simultaneously support as many versions of 3ds max and 3DXI as is possible and practical.

1.1.2. Supported configurations

Plug-in	3ds max release	"igame.dll" version
3dsmax6\PVRGeoPOD.dle	3ds max 6	6.0.0.56
	3ds max 6 + SP1	6.0.1.62
	3ds max 7	7.0.0.65
	3ds max 7 + SP1	7.0.1.76
	3ds max 7 + 3DXI 2.0	7.0.1.78
	3ds max 8 + 3DXI 2.0	8.0.0.40
	3ds max 8 (contains winding order bug?)	8.0.0.92
	3ds max 8 + SP1	8.0.1.11
	3ds max 8 + SP2	8.0.1.18
	3ds max 8 + SP3	8.0.1.24
3dsmax9\PVRGeoPOD.dle	3ds max 9	9.0.0.100
	3ds Max 2008	10.0.0.86
	3ds Max 2009	11.0.0.57

1.1.3. Installation

3ds max 6

Copy "3dsmax6\PVRGeoPOD.dle" to "C:\3dsmax6\plugins\", or the equivalent location for where your copy of 3ds max is installed.

3ds max 7

Copy "3dsmax6\PVRGeoPOD.dle" to "C:\3dsmax7\plugins\", or the equivalent location for where your copy of 3ds max is installed.

3ds max 8

Copy "3dsmax6\PVRGeoPOD.dle" to "C:\Program Files\Autodesk\3dsMax8\plugins\", or the equivalent location for where your copy of 3ds max is installed.

3ds max 9

Copy "3dsmax9\PVRGeoPOD.dle" to "C:\Program Files\Autodesk\3ds Max 9\plugins\", or the equivalent location for where your copy of 3ds max is installed.

3ds Max 2008

Copy "3dsmax9\PVRGeoPOD.dle" to "C:\Program Files\Autodesk\3ds Max 2008\plugins\", or the equivalent location for where your copy of 3ds max is installed.

3ds Max 2009

Copy "3dsmax9\PVRGeoPOD.dle" to "C:\Program Files\Autodesk\3ds Max 2009\plugins\", or the equivalent location for where your copy of 3ds max is installed.

1.2. Maya

1.2.1. Installation

Maya 7

Windows

Copy "PVRGeoPOD_v7.mll" to "C:\Program Files\Alias\Maya7.0\bin\plug-ins\", or the equivalent location for where your copy of Maya is installed.

Linux

Copy "PVRGeoPOD_v7.so" to "/usr/aw/maya7.0/bin/plug-ins/", or the equivalent location for where your copy of Maya is installed.

Maya 8

Windows

Copy "PVRGeoPOD_v8.mll" to "C:\Program Files\Alias\Maya8.0\bin\plug-ins\", or the equivalent location for where your copy of Maya is installed.

Linux

Copy "PVRGeoPOD_v8.so" to "/usr/aw/maya8.0/bin/plug-ins/", or the equivalent location for where your copy of Maya is installed.

Maya 8.5

Windows

Copy "PVRGeoPOD_v8.5.mll" to "C:\Program Files\Autodesk\Maya8.5\bin\plug-ins\", or the equivalent location for where your copy of Maya is installed.

Linux

Copy "PVRGeoPOD_v8.5.so" to "/usr/aw/maya8.5/bin/plug-ins/", or the equivalent location for where your copy of Maya is installed.

Maya 2008

Windows

Copy "PVRGeoPOD_v2008.mll" to "C:\Program Files\Autodesk\Maya2008\bin\plug-ins\", or the equivalent location for where your copy of Maya is installed.

Linux

Copy "PVRGeoPOD_v2008.so" to "/usr/autodesk/maya2008/bin/plug-ins/", or the equivalent location for where your copy of Maya is installed.

2. Supported File Formats

2.1. Header File

PVRGeoPOD can export to a header-file format (file extension: "H"), suitable for direct usage in source code.

```
// Include the scene data
#include "model.h"

CPVRTPODScene m_model;

// Load the model
if(!m_model.ReadFromMemory(c_MODEL_H))
	return false;

// Do stuff

// Free the memory
m_model.Destroy();
```

The advantages of a header file:

1. It is possible to produce an executable which does not require any separate data files to load; all data can be inside the executable. This is not an advantage for operating systems which allow "resources" inside the executable, since in that case a POD file can just as easily be compiled into the executable.
2. The content of the file is partially human-readable, and can be edited with a text editor.

2.1.1. Example

See *Skybox* in the PowerVR SDK.

2.2. POD File

PVRGeoPOD can also export to a binary format (file extension: "POD"). The PowerVR SDK Tools library, part of the PowerVR SDK, contains code to load and save POD files.

```
CPVRTPODScene m_model;

// Load the model
if(!m_model.ReadFromFile("model.pod"))
	return false;

// Do stuff

// Free the memory
m_model.Destroy();
```

The advantages of a POD file:

1. A POD file can be loaded into memory, the vertex data copied into HW friendly buffers – e.g. a vertex buffer object (OpenGL [ES]) or a vertex buffer (Direct3D [Mobile]) - and the POD file released, or even just the mesh-data memory freed (in which case set the freed memory pointers to NULL).
   This saves memory, compared to headers, on operating systems which do not support page files: the memory used by the data in a header file will be around for the lifetime of the application, and therefore two copies of mesh data are in memory. This is not a problem for operating systems which support a page-file, since the unused memory can be swapped out. On OpenGL [ES] this is not an advantage if `glVertexPointer()` and its ilk are used, since no copy is required.
2. It also allows POD files to be changed without rebuilding the application.
3. It is smaller on disk (although there should be little size difference between two compiled executables, one using a header file and one with a POD file in resources).

2.2.1. Example

See *ChameleonMan* in the PowerVR SDK.

3. Using the Exporter

3.1. 3D Studio Max

PVRGeoPOD can be used in 3ds max by:

1. From the menu, select "`File/Export`…" or "`File/Export Selected`…".
2. In the "`Save as type`" drop-down box, select "`PowerVR Exporter (*.POD,*.H)`".
3. Browse to the location you wish the file to be saved.
4. Double-click an old file to overwrite, or type in a filename; use the extension .H for a header-file, and .POD for a binary file. If the filename does not specify an extension, POD will be assumed.

You will now see the PVRGeoPOD dialog-box, as shown in Figure 1.

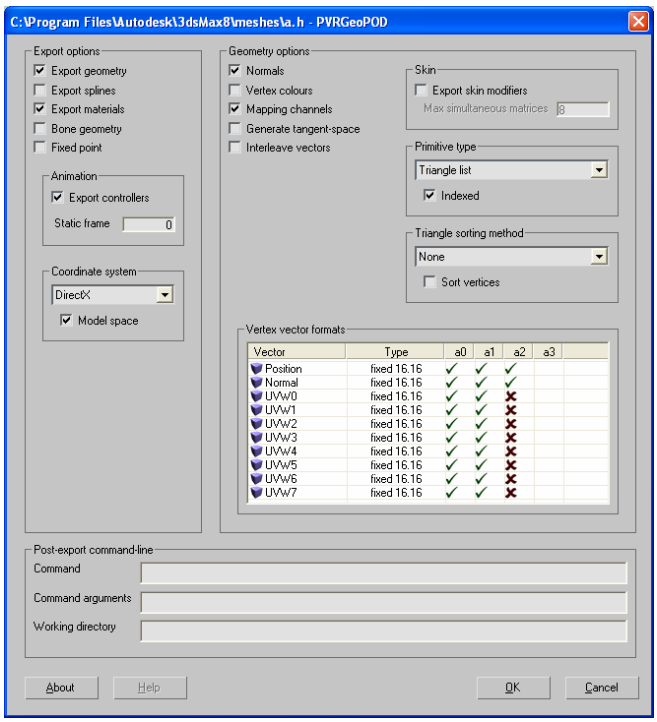

Figure 1 The PVRGeoPOD dialog box

The function of each dialog-box element is individually described in a tool-tip that will appear when the mouse is hovered over them, as shown in Figure 2.

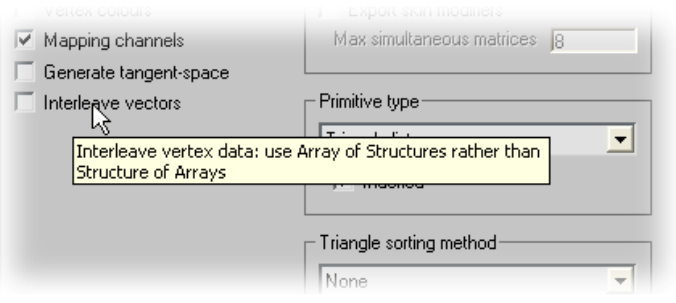

Figure 2 Everything has a tool-tip

More detailed help is available from the "`Help`" button at the bottom of the window.

3.2. Maya

PVRGeoPOD can be used in Maya by:

5. From the menu, select "`File/Export All`…" or "`File/Export Selection`…".
6. In the "`Files of type:`" drop-down box, select "`PowerVR Exporter (*.pod,*.h)`".
7. Browse to the location you wish the file to be saved.
8. Double-click an old file to overwrite, or type in a filename; use the extension .h for a header-file, and .pod for a binary file. If the filename does not specify an extension then the file will be saved in POD format.

You will now see the PVRGeoPOD dialog-box, as shown in Figure 3.

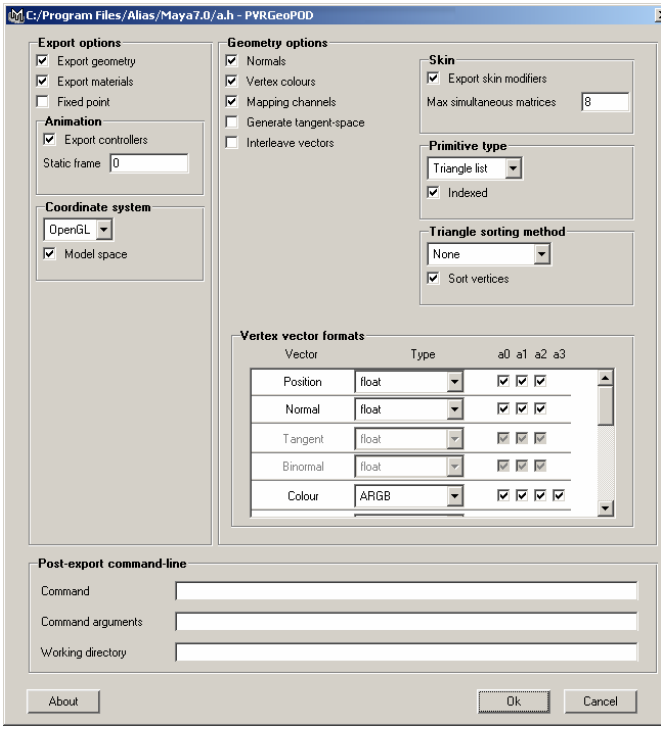

Figure 3 The PVRGeoPOD dialog box (Windows version)

The function of each dialog-box element is individually described in an annotation that will appear in Maya's Help Line when the mouse is hovered over them, as shown in Figure 4.

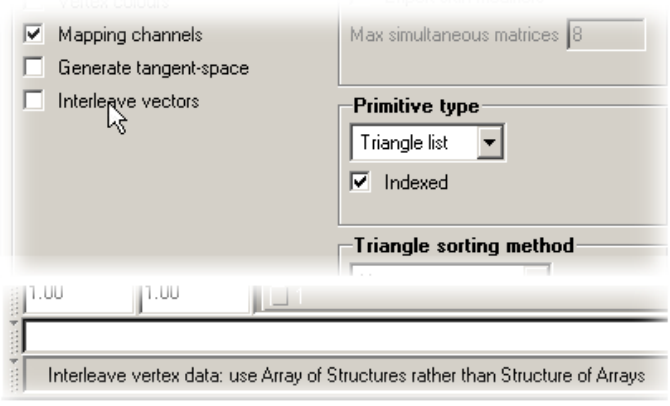

Figure 4 Everything has an annotation

4. Using POD Data in your Application

The PowerVR Tools library, part of the PowerVR SDK, contains code to load and save POD files; it also contains many utility functions to help use the resultant data structures, which can be used both when the data is loaded from a POD file **and** when compiled in from a header file.

The POD data structures are relatively straightforward and easy to use. For example, in OpenGL ES, the vertex (position) data pointer can be used directly as an input to `glVertexPointer()`; the same goes for the pointer to the normals, and all the others.

For tasks such as acquiring World matrices, bone matrices, etc, you can use the code in the `TModelPOD` module from the PowerVR Tools library.

For this beta, documentation for this code is only available from the comprehensive comments before each declaration in the "`*Tools.h`" file, and duplicated in "`PVRTModelPOD.cpp`". In due course there will be separate documentation available.

Appendix A. POD File Format

This information is irrelevant if the code supplied in "PVRTModelPOD.cpp" - to load binary POD files - is used, as we recommend.

A.1. Binary File ("POD")

A.1.1. Binary File Format

POD files are stored in a tagged, nested structure. A marker consists of two 32-bit values:

DWORD	Bit	Symbol	Description
0	31	End	End of data block bit 0 This marker is the beginning of a block 1 This marker is at the end of a block
	30:0	Name	Marker name, identifying the data which follows
1	31:0	Length	Amount of data which follows; AKA distance to next marker.

- After each marker, `Length` bytes of data follow.
- The list of possible `Name` values is in the enumerated type `EPODFileName` in the file "PVRTModelPOD.cpp".

A.1.2. Overview of File Reading

```
10 Read marker
20 If recognised marker, read the following data, otherwise skip it
30 GOTO 10
```

A.2. Text File ("H")

A.2.1. POD data-structure definitions

These data structures are used to store POD data; they are the types which need to be defined before including an exported "H" file.

From PVRTVertex.h:

```
enum EPVRTDataType {
	EPODDataNone,
	EPODDataFloat,
	EPODDataInt,
	EPODDataUnsignedShort,
	EPODDataRGBA,
	EPODDataARGB,
	EPODDataD3DCOLOR,
	EPODDataUBYTE4,
	EPODDataDEC3N,
	EPODDataFixed16_16
};
```

From PVRTBoneBatch.h:

```
class CPVRTBoneBatches
{
public:
	int	*pnBatches;			// Space for nBatchBoneMax bone indices, per
batch
	int	*pnBatchBoneCnt;	// Actual number of bone indices, per batch
	int	*pnBatchOffset;			// Offset into triangle array, per batch
	int nBatchBoneMax;		// Stored value as was passed into Create()
	int	nBatchCnt;			// Number of batches to render
};
```

From the D3DM PVRTFixedPointAPI.h:

```
#ifndef PVRTFIXEDPOINTENABLE
	#define VERTTYPE float
#else
	#define VERTTYPE int
#endif
```

Alternatively, from the OGLES `PVRTFixedPointAPI.h`:

```
#ifndef PVRTFIXEDPOINTENABLE
	#define VERTTYPE GLfloat
#else
	#define VERTTYPE GLfixed
#endif
```

From `PVRTModelPOD.h`:

```
/****************************************************************************
** Enumerations
****************************************************************************/
enum EPODLight {
	ePODPoint,
	ePODDirectional
};

/****************************************************************************
** Structures
****************************************************************************/
class CPODData {
public:
	EPVRTDataType  eType;		// Type of data stored
	unsigned int   n;			// Number of values per vertex
	unsigned int   nStride;		// Distance in bytes from one array entry to the next
	unsigned char  *pData;		// Actual data (array of values); if mesh is
interleaved, this is an OFFSET from pInterleaved
};

struct SPODCamera {
	int			nIdxTarget;			// Index of the target object
	VERTTYPE	fFOV;				// Field of view
	VERTTYPE	fFar;				// Far clip plane
	VERTTYPE	fNear;				// Near clip plane
	VERTTYPE	*pfAnimFOV;			// 1 VERTTYPE per frame of animation.
};

struct SPODLight {
	int			nIdxTarget;		// Index of the target object
	VERTTYPE	pfColour[3];	// Light colour (0.0f -> 1.0f for each channel)
	EPODLight	eType;			// Light type (point, directional etc.)
};

struct SPODMesh {
	unsigned int		nNumVertex;		// Number of vertices in the mesh
	unsigned int		nNumFaces;		// Number of triangles in the mesh
	unsigned int		nNumUVW;		// Number of texture coordinate channels
per vertex
	CPODData			sFaces;			// List of triangle indices
	unsigned int		*pnStripLength;	// If mesh is stripped: number of tris
per strip.
	unsigned int		nNumStrips;		// If mesh is stripped: number of strips,
length of pnStripLength array.
	CPODData			sVertex;		// List of vertices (x0, y0, z0,
x1, y1, z1, x2, etc...)
	CPODData			sNormals;		// List of vertex normals (Nx0,
Ny0, Nz0, Nx1, Ny1, Nz1, Nx2, etc...)
	CPODData			sTangents;		// List of vertex tangents (Tx0,
Ty0, Tz0, Tx1, Ty1, Tz1, Tx2, etc...)
	CPODData			sBinormals;		// List of vertex binormals (Bx0,
By0, Bz0, Bx1, By1, Bz1, Bx2, etc...)
	CPODData			*psUVW;			// List of UVW coordinate sets;
size of array given by 'nNumUVW'
	CPODData			sVtxColours;	// A colour per vertex
	CPODData			sBoneIdx;		// nNumBones*nNumVertex ints
(Vtx0Idx0, Vtx0Idx1, ... Vtx1Idx0, Vtx1Idx1, ...)
	CPODData			sBoneWeight;	// nNumBones*nNumVertex floats (Vtx0Wt0,
Vtx0Wt1, ... Vtx1Wt0, Vtx1Wt1, ...)

	unsigned char		*pInterleaved; // Interleaved vertex data

	CPVRTBoneBatches	sBoneBatches;  // Bone tables
};

struct SPODNode {
	int			nIdx;				// Index into mesh, light or
camera array, depending on which object list contains this Node
	char		*pszName;			// Name of object
	int			nIdxMaterial;		// Index of material used on this mesh

	int			nIdxParent;			// Index into MeshInstance array;
recursively apply ancestor's transforms after this instance's.
	VERTTYPE	pfPosition[3];		// Position in World coordinates
	VERTTYPE	pfRotation[4];		// Rotation in World coordinates
	VERTTYPE	pfScale[3];			// Scale in World coordinates
```

```
	VERTTYPE	*pfAnimPosition;	// 3 floats per frame of animation.
	VERTTYPE	*pfAnimRotation;	// 4 floats per frame of animation.
	VERTTYPE	*pfAnimScale;		// 7 floats per frame of animation.
};

struct SPODTexture {
	char	*pszName;			// File-name of texture
};

struct SPODMaterial {
	char		*pszName;			// Name of material
	int			nIdxTexDiffuse;		// Idx into textures for diffuse
texture
	VERTTYPE	fMatOpacity;		// Material opacity (used with vertex alpha ?)
	VERTTYPE	pfMatAmbient[3];	// Ambient RGB value
	VERTTYPE	pfMatDiffuse[3];	// Diffuse RGB value
	VERTTYPE	pfMatSpecular[3];	// Specular RGB value
	VERTTYPE	fMatShininess;		// Material shininess
};

struct SPODScene {
	VERTTYPE	pfColourBackground[3];	// Background colour
	VERTTYPE	pfColourAmbient[3];		// Background colour

	// The length of the following array, and the number of items in the Node array which
are cameras (these come second)
	unsigned int	nNumCamera;
	SPODCamera		*pCamera;

	// The length of the following array, and the number of items in the Node array which
are lights (these come third)
	unsigned int	nNumLight;
	SPODLight		*pLight;

	// Meshes may be instanced several times in a scene; i.e. multiple Nodes may reference
any given mesh.
	unsigned int	nNumMesh;
	SPODMesh		*pMesh;

	unsigned int	nNumNode;		// Number of items in the following array
	unsigned int	nNumMeshNode;  // Number of items in the following array which are
objects
	SPODNode		*pNode;			// Sorted as such: objects, lights,
cameras, Everything Else (bones, helpers etc)

	unsigned int	nNumTexture;
	SPODTexture		*pTexture;

	unsigned int	nNumMaterial;
	SPODMaterial	*pMaterial;

	unsigned int	nNumFrame;		// Number of frames of animation
	unsigned int	nFlags;			// PVRTMODELPODSF_* bit-flags
};
```