Navigational Data Tools

Reference Guide

Filename	:	NavDataTools.Reference Guide
Version	:	PowerVR SDK REL_4.0@3855898a External Issue
Issue Date	:	15 Dec 2015
Author	:	Imagination Technologies Limited

Contents

List of Figures

1. Introduction

Navigational data is a complex component of computer graphics, and this document is intended to act as a reference guide for the variety of tools available when dealing with 2D and 3D graphics rendering. This reference guide is intended to be used alongside the following whitepapers:

- Navigation Rendering Techniques.
- 3D Navigation Rendering Techniques.

2. Using 2D Navigation Data Tools

The following sections give short introductions to the tools that are used to generate the data which in turn is used by the Navigation demo. The tools are thoroughly documented in the source and come with the full source code. Please note that some of the tools require external libraries to compile, so it is important that the user makes sure to read the following in full if it is intended to reproduce the following steps and compile data. The current release only includes Visual Studio project files.

2.1. Conversion Workflow

This section gives a brief overview of the steps involved in the binary data generation process. The different input conversion steps are illustrated in Figure 1, starting at the top the transformation of the input data into the intermediate format. The “IntermediateFormatConverter” is used to extract the relevant information for visualization purposes and stores it in a binary container format used in the following steps.

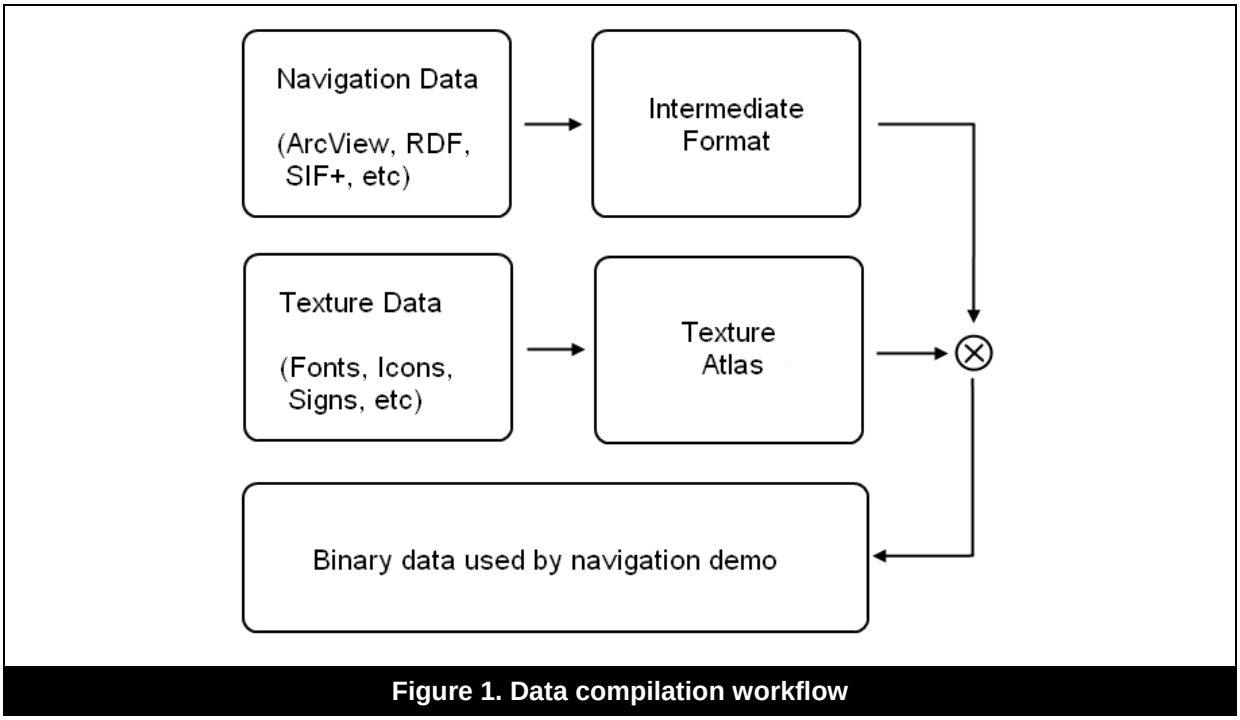

Figure 1. Data compilation workflow

The middle part of the workflow diagram illustrates the texture atlas description generation. In the case of the bitmap font, the description file given by the Bitmap Font Generator tool is transformed with the “FontConverter” in a texture atlas compatible description file. This and the intermediate format files are then used by the “NavDataCompiler” tool in the data compilation step. Figure 2 illustrates how the compiled data and textures are then used by the binary to render the navigation maps.

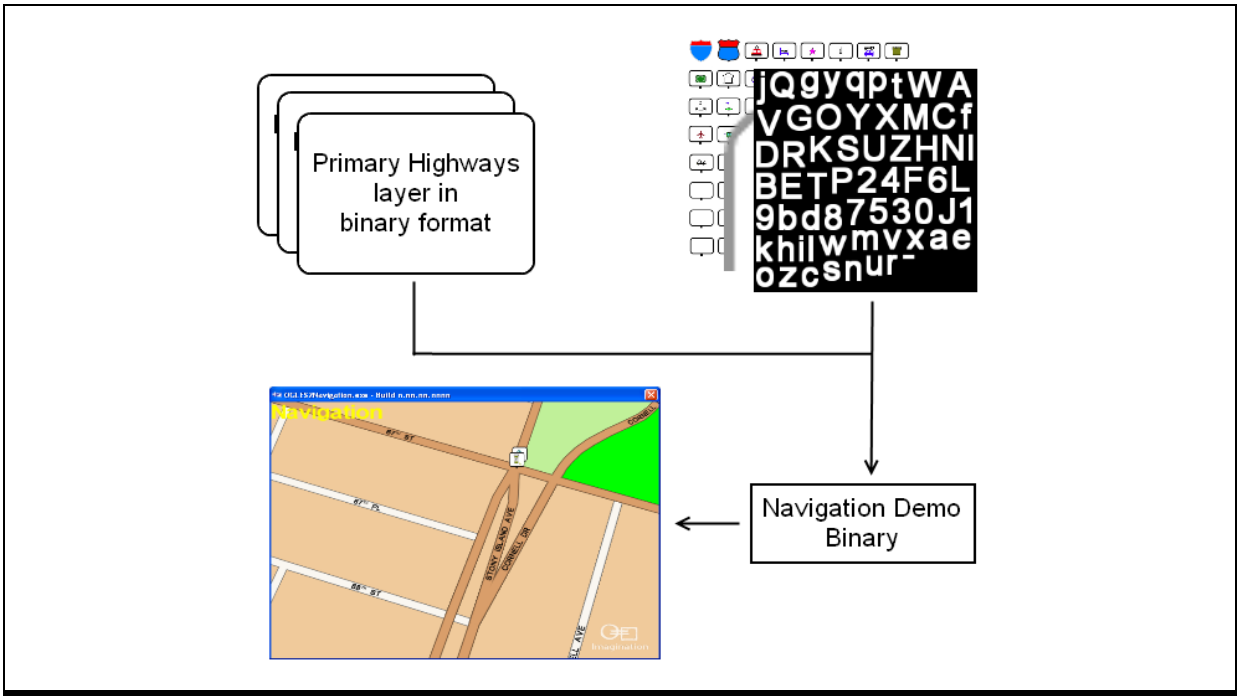

Figure 2. Overview of input to the navigation demo

2.2. IntermediateFormatConverter

This tool is used to convert the navigation map data from an arbitrary input format like ArcView™, RDF, SIF+, etc., into an intermediate data format which is processed by the final content compiler. For the task of reading the input data an open source library called OGR from GDAL (Geospatial Data Abstraction Library, http://www.gdal.org) has been used. The utilities can be found in a precompiled package at http://fwtools.maptools.org (version 2.4.6 has been used during development).

Please download and install these tools and set the `FWTOOLSDIR` environment variable to the installation directory if it is intended to compile map data with the PowerVR SDK utilities. The resulting executable accepts two parameters, the input and output directory:

```
IntermediateFormatConverter.exe [InputDirectory] [OutputDirectory]
```

The input directory should point directly to the navigation data directory and OGR automatically parses all contained files. The transformed files will be written to the output directory and every layer will be augmented by the suffix `_maplayer.txt`.

2.3. NavDataCompiler

This tool takes the output from the previous tool and transforms it into a binary container format which stores the data in a triangulated format plus additional information like enclosing bounding boxes, etc. As there are a lot more configuration options per layer and type of geometry available the tool takes a conversion configuration file as input. This file contains a transformation rule per line and global information, e.g., where to read the data from and store it to. The basic layout of the file is subdivided into two sections, the global options and the per layer options. An example configuration named `example_indexfile.txt` can be found in the same subdirectory.

The compiled executable accepts a single parameter as input, the configuration file location:

```
NavDataCompiler.exe [ConfigurationFile]
```

Please see the source code for effects of the various parameters which are defined per operation.

2.4. libPVRNavigation

This library is used by the previous two utilities and abstracts commonly used code to avoid repetition and ease maintenance. Examples of the included code are the polygon triangulation code or file loading/saving facilities.

2.5. FontConverter

The bitmap font used by the Navigation demo has been created with the “Bitmap Font Generator” from http://www.angelcode.com. It generates the bitmap font itself and a description file describing the layout of the glyphs in the bitmap. This layout file can be converted into a texture atlas compatible description file:

```
FontConverter.exe [InputFile] [OutputFile]
```

2.6. LayerMerger

As it is not possible to hide individual layers during runtime in the Navigation demo, the billboard signs map layer in the intermediate format can be merged into a single layer with this utility. This saves time during runtime in the culling stage and reduces the number of files in the project. The billboard map layers can be merged with this command line:

```
LayerMerger.exe [OutputFile] [InputFile0 InputFile1 …]
```

3. Using 3D Navigation Data Tools

Two new tools have been implemented to generate the additional data that is required for accelerated culling against the view frustum (see Section 3.1) and the Portal Visibility Sets (see Section 3.2). The NAVTEQ 3D Enhanced City Model data has been converted into the PowerVR POD format and will be used as input for the following tools.

3.1. 3dSceneCompiler

The 3dSceneCompiler creates the hierarchical index of all models found in a city block. A city block represented in the Collada interchange format consists of a list of individual models (or entities) which in most cases define a child-parent-node relationship, e.g., multi-storey buildings are often represented as separate basement, storeys and roof nodes referencing the same parent node.

It creates an index for all child-parent node relationships to ease the access of individual buildings, which optimizes the view-frustum culling process, as buildings can be culled as a whole instead of their individual nodes. Furthermore, it calculates the bounding box for each individual level of detail, creating bounding boxes on city block and parent node level.

As input, the tool itself requires a textual description file that contains a list of city block files (please see the example file in the source package). It will write the model index into a specified file, which is an additional option to specify another filename where the referenced texture filenames from each city block will be written to. This has been used to only include the required textures for the 3D Navigation demo to reduce its footprint. Figure 3 illustrates the file format for the model index generated by the 3dSceneCompiler.

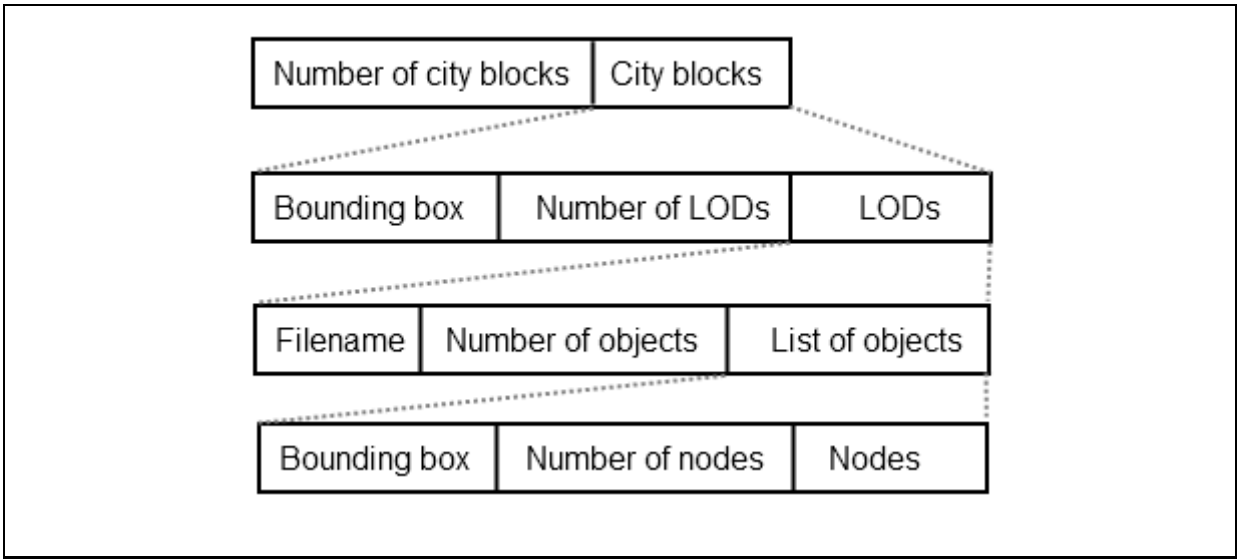

Figure 3. File format for the model index generated by the 3dSceneCompiler

3.2. OcclusionCalculator

The OcclusionCalculator tool calculates the Portal Visibility Set. It creates a set of viewpoints to calculate the visibility information for and then renders the scene from each of these viewpoints in all six directions (up, down, left, right, front and back). In order to calculate the visibility results two methods requiring different levels of hardware support have been implemented and will be explained in subsequent sections.

The visibility result of each viewpoint is then merged into a visibility set and stored to a file that can be used to index objects. At runtime the index is then queried for the nearest visibility information data point and the list of objects found at that location rendered. Figure 4 illustrates the file format for the occlusion data generated by the OcclusionCalculator.

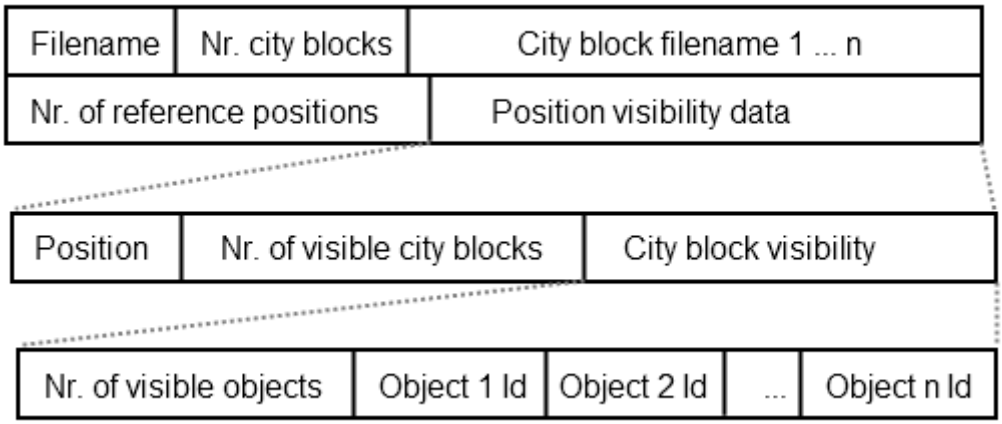

Figure 4. File format for the occlusion data generated by the OcclusionCalculator

3.2.1. Occlusion Queries

Occlusion queries are a very fast way to count the number of pixels an object takes up when being rendered. It requires OpenGL hardware support and comes in two flavours: the first always renders the whole object and counts the total amount of pixels that end up in the framebuffer whereas the other simply reveals if at least one pixel ends up in the framebuffer. The latter is the newer one and not every hardware supports it at the time of writing, therefore, both methods have been implemented and the appropriate one picked at runtime.

As no colour data is required for the analysis, framebuffer writes have been disabled by setting the colour mask (`glColorMask`). In the first pass the whole scene is rendered into the depth buffer. In the second pass each individual object is rendered again, but this time the depth test is set to only pass objects which are nearer or equal to the depth buffer content (`GL_LEQUAL`). Each object has been allocated a unique occlusion query id and it will be set before rendering the object in the second pass.

After finishing the second pass it is possible to read back the result of the occlusion query for each object, which reveals whether an object is visible in the current view or not.

3.2.2. Framebuffer Read-back

The framebuffer read-back method is used as fall-back method when occlusion queries are not supported by the hardware. For rendering, each individual object is assigned a unique custom colour which encodes the object and city block ID. The scene is then rendered from the current viewpoint using depth buffering. After the render is complete the framebuffer is read back (`glReadPixels`) and visible objects determined by analysing each pixel of the framebuffer.

4. Contact Details

For further support, visit our forum:
http://forum.imgtec.com

Or file a ticket in our support system:
https://pvrsupport.imgtec.com

To learn more about our PowerVR Graphics SDK and Insider programme, please visit:
http://www.powervrinsider.com

For general enquiries, please visit our website:
http://imgtec.com/corporate/contactus.asp