RenderWare Graphics

Instancing

Tool

Contact Us

Criterion Software Ltd.

For general information about RenderWare Graphics e-mail info@csl.com.

Developer Relations

For information regarding Support please email devrels@csl.com.

Sales

For sales information contact: rw-sales@csl.com.

Acknowledgements

With thanks to RenderWare Graphics development and documentation teams.

1.1 Introduction

The *instance* tool can be used to optimize RenderWare Graphics files for target platforms.

Preinstanced data will load faster and require a smaller memory footprint. It doesn't need the 'instancing' phase that platform-independent RenderWare Graphics data goes through on being rendered the first time. Instanced data is ideal for inclusion in production builds.

This document describes the use of the preinstancing tool, *instance*.

Artists may use the *instance* tool directly from the exporters. In this case, it forms the last stage of an export procedure.

instance can also be run from the command-line. It is intended to be used in this manner by developers or build engineers. *instance* can be incorporated into the regular production build.

1.1.1 What is instanced data?

Many RenderWare Graphics entities have two separate internal forms. One is a general, non-optimized form that is suitable for sharing between multiple platforms. The other is a platform-specific form that is optimized for rendering on that platform.

This platform-specific form is known as the *instanced* form.

This terminology should not be confused with the object-oriented-programming concept of an object instance.

Usually the instanced form of a RenderWare Graphics object is created and handled internally by RenderWare Graphics. Sometimes the instanced form is exposed through various RenderWare Graphics API functions.

Examples of objects and their instanced forms:

OBJECT	INSTANCED FORM
platform-independent texture dictionary	texture dictionary
world	world (handled internally)
clump	clump (handled internally)

RenderWare Graphics usually creates the instanced form of an object when the first attempt is made to render that object.

1.1.2 Why instancing should be done early

When an object is instanced, the original form of the RenderWare Graphics object is kept. This is often unnecessary, as only the instanced form is needed for rendering. Consequently, the applications memory footprint is increased.

The instancing process takes time to complete. This can increase the amount of memory needed for the object, and also results in longer initialization time if it occurs in game.

This process may also cause unwanted delays in rendering. If a non-instanced object is rendered mid-game, the instancing will happen there and then.

The downside of late instancing may be avoided through the use of various RenderWare Graphics API functions to force the instancing of an object prior to the use of that object in rendering. Usually these functions would be called as part of an application spawned by the 'build' process, to convert art-assets into a platform-optimized form.

Unfortunately, the instancing functions must usually be called from a program running on the target platform. It is often time-consuming and inconvenient to create such a program. Since this program must be spawned from a build process, the difficulty is magnified.

1.1.3 Early instancing

RenderWare Graphics now provides a preinstancing tool called '*instance*' as part of its tool chain. This tool will run on a PC under Windows 2000.

The simplest way to use *instance* is from within the exporters, as the last stage of an export process. Artists can export product-ready resource files in this manner. This usage is described within the exporter documentation.

instance also allows the creation of preinstanced data via a command-line interface. In this manner, the *instance* tool may be used from within a build process.

The preinstancing tool attempts to instance data via a connection to a Visualizer (see the document **`RenderWareVisualizer.pdf`**). The details of the connection must be set up prior to using *instance*. When *instance* is run, it obtains a list of connections, trying each one in turn.

The tool does as much work as possible on the PC in order to save time and network bandwidth. The tool uses **`.dll`**s for which source is provided. These **`.dll`**s may be overridden in order to carry out user-specific instancing.

1.2 Using '*instance*'

instance is a command line utility that takes a list of files, and converts them into an instanced form.

instance does all work locally if possible, but if necessary calls upon external visualizers running on the target consoles to do additional instancing work.

1.2.1 *instance* requirements

In order for *instance* to run correctly, a connection to a visualizer is usually required. The exception to this is when instancing platform independent texture dictionaries, which can be done entirely locally.

instance can use a 'pool' of connections to visualizers if there is no unique visualizer that can be assigned to the instancer.

1.2.2 *instance* processing stages

instance takes a list of files from the command line. The list may include wildcard specifications, such as "**`*.rws`**".

Any streamed RenderWare Graphics file format is supported, i.e. generic RenderWare streamed files (**`.rws`**).

Legacy formats such as clumps (**`.dff`**) and worlds (**`.bsp`**) are also supported.

instance processes each file in the list one by one. Each file is split into chunks. Each chunk is passed to a **`.dll`**, which tries to instance locally. If this succeeds, *instance* goes on to the next chunk. Otherwise, a remote instancing host is tried, i.e. a visualizer.

instance saves a new file in an output directory based upon the platform. PlayStation 2 files are placed in **`.\instance\sky`**, GameCube files in **`.\instance\gcn`** and Xbox files in **`.\instance\xbox`**. These directories may be overridden on the command line.

Input files with the **`.rws`** extension are renamed on a platform specific basis. The extensions are changed as follows:

PLATFORM	NEW EXTENSION
PlayStation 2	**`.rp2`**
GameCube	**`.rg1`**
Xbox	**`.rx1`**

If instancing succeeds, *instance* exits with code 0. If instancing fails, *instance* prints an appropriate error message and returns 1.

1.2.3 Running *instance* from the command line

instance is located in **tools\instance** under the RenderWare Graphics SDK root directory. The developer may add it to their global path, or use the fully qualified name of the program in order to run it.

The syntax of *instance* is

```
instance [[-c <hostname>] ...] [-d <a_texture>.dds] [-h]
         [-i <filename>] [-l] [-o <outputdir>] [-p<platform>] [-r]
         [-s] [-t<timeout>] [-x<platform> <file>] [<file> ...]
         -c <hostname>
```

Command line options for *instance* are

-c <hostname>

Connect to the host named **<hostname>**. The list of connections used by the Visualizer is searched for **<hostname>**. This list may be edited using the Visualizer connection editor.

More than one connection may be specified. *instance* will attempt to use each connection in turn. If no connections work, *instance* will display an error and terminate.

-d <a_texture>.dds

Use the given **.dds** compressed texture. This will override textures of the same name when creating texture dictionary chunks.

.dds compressed textures are supported on the GameCube and Xbox.

-h

Print out the command line options help message.

-i <filename>

Get information on the given RenderWare Graphics file. *instance* will report if the file contains platform-specific texture dictionaries or geometry data.

-l

do local instancing only; disallow remote instancing. Elements of files that cannot be instanced locally will not be modified.

-o<dirname>

Change the default output root directory to **<dirname>**.

-p<platformname>

Attempt instancing on the platform **<platformname>**. This may be **PS2**, **Xbox** or **GameCube**. This option should be used in the event that all instancing may be performed locally, e.g. for converting platform independent texture dictionaries to platform specific texture dictionaries.

-r

add textures or animations to start of file prior to instancing. This switch must be used if the file is to be viewed with the Visualizer. The Visualizer cannot locate the additional data once the file is instanced.

This is necessary for **.dff** or **.bsp** files, because they don't contain textures or animations themselves. **.rws** files contain this data already.

-s

Don't use platform specific subdirectories for output.

-t<timeoutinseconds>

Set the time, in seconds, to wait to before giving up on a remote instancing target. The default is 180 seconds.

-x<platformname> <filename>

Supply platform-specific extension data. **<platformname>** must be one of **PS2**, **Xbox** or **GameCube**. **<filename>** must be a configuration file as described in *1.3 Instancing problems and platform specific details*.

Examples

instance –c tetsuo banana.rws

Instance the file **banana.rws** on the target 'tetsuo'. 'tetsuo' is a PlayStation 2, GameCube or Xbox running a Visualizer. The details of the console must have been set up within the Visualizer connection editor.

instance –c tetsuo –c ryoko banana.rws

Instance the file **banana.rws**, trying both 'tetsuo' and 'ryoko' as instancing targets. These must previously have been set up within the Visualizer connection editor.

For this next example, assume **mytextures.rws** contains a platform independent texture dictionary and nothing else.

instance –pGameCube mytextures.rws

Instance the file **mytextures.rws** locally. Note that no external connection was specified. If the **mytextures.rws** file had contained anything but platform independent texture dictionaries, the instancing process would have failed.

instance *.rws

Instance all **.rws** files in the current directory on the preinstancing targets enabled within the Visualizer connection editor.

1.3 Instancing problems and platform specific details

While instancing is a necessary part of an art tool chain, there are some things that cannot be instanced, or require particular attention before instancing.

1.3.1 General Issues

Collision data

Collision information is stripped out in the instancing stage, although empty collision data chunks may remain. Do not instance worlds or clumps containing collision data if you wish to use that data later.

Remote instancing

The instancer cannot instance on a remote visualizer that's already in use. If a visualization or instancing session is already in progress, that visualizer will not be available for instancing.

Skin splitting

Each platform has a maximum number of bones that it supports per skinned object. RenderWare Graphics provides a skin splitting toolkit, **`RtSknSplit`**, which may be used to break skins apart so that they fit within specified bone limits.

The instancer uses this toolkit during the instancing process, but depending on your requirements you may wish to modify the limits that are used (see the API reference on **`RpSkin`** as to why you may wish to do this). Consult section *1.4 Extending instance* for details on how to change the limits.

Non-instanceable data

Some RenderWare Graphics objects need to be reinstanced on the fly, and preinstancing does not make sense for them.

- Atomics containing patches are not instanced.
- Atomics containing more than one morph target are not instanced.
- Clumps containing DMorph targets are not instanced.
- Toon geometry (worlds or clumps) is not instanced.
- Particle systems are not instanced.

1.3.2 PlayStation 2

For artwork to be rendered correctly in the presence of mipmapping on the PlayStation 2, MipK&L values need to be assigned to the geometry.

Details of the cameras to be used must be provided in order for these values to be calculated correctly. Unfortunately, within the instancer there is no way of determining what sort of camera would be appropriate.

MipK&L values may be assigned manually in the exporters, or through the use of a custom program using the **RtMipK** toolkit. This should be done prior to instancing, as after instancing the geometry information used to calculate the MipK&L values is no longer present.

1.3.3 GameCube

The GameCube implementation of RenderWare Graphics supports compressed vertex formats that may be used to optimize representations of geometries in either worlds or atomics.

instance provides a way of specifying these formats for an instancing session.

Two vertex formats may be declared in a single instancing session – one for all worlds that will be instanced, and another for all geometries.

These formats are described within a configuration file that must be supplied to *instance* via the **-x** command line switch.

For convenience, the format of this configuration file is identical to the code that's needed to set up the compressed vertex format. This code can be placed in a small header file, which is included in mainline code as needed.

It may also be passed to *instance* as a command line argument.

Inside this header file, the vertex formats being declared must be named **worldFmt** and **geomFmt**, depending on whether they are to be used for worlds or geometries.

The configuration file parser is very basic, but discards comments, whitespace and handles the necessary enumerated constants.

Examples of suitable files follow.

vertexFormat.h (`#include`'d in mainline code, also passed to *instance*)

```
RpGameCubeVtxFmtSetPosition(worldFmt, rpU16, 128);
RpGameCubeVtxFmtSetNormal(worldFmt, rpU16, TRUE);
...
```

mainlineCode.c

```
RpWorld *world = RpWorldStreamRead(stream);
```

```
RpGameCubeVtxFmt *worldFmt = RpGameCubeVtxFmtCreate();
#include "vertexFormat.h"
/* Main code using worldFmt */
RpGameCubeWorldSetVtxFmt(world, worldFmt);
RpGameCubeVtxFmtDestroy(fmt);
```

sample command line

```
instance –xGameCube vertexFormat.h myworld.rws
```

1.4 Extending *instance*

Two options exist for extending data created in the instancing process.

If custom plugin data must be attached to instanced data, this must be done *after* the instancing process. This is because the Visualizer is not linked to the custom plugins, and therefore doesn't know how to handle the custom data.

The other point at which custom data can be instanced is in the local instancing **.dll**s. There is one of these for each platform; **rwinstance_sky.dll**, **rwinstance_gcn.dll** and **rwinstance_xbox.dll** for the PlayStation 2, GameCube and Xbox respectively.

These files are contained within the **export\bin** subdirectory of the RenderWare Graphics distribution.

Source for the existing **.dll**s is provided in the **tools\instance** directory.

This source may be modified in order to do custom instancing. It may also be changed in order to influence some forms of default instancing, such as skin splitting.

At present, these custom instancers carry out the following operations:

All platforms

- instance platform independent texture dictionary chunks into platform dependant texture dictionaries
- split skins based on platform specific limits

PlayStation 2

- skin split limit used is 64 bones
- perform pipeline overrides as dictated by the exporters

GameCube

- skin split limit used is 10 bones for single-weight skins
- DDS texture overrides are implemented

Xbox

- skin split limit used is 50 bones, with skins split down to 30 bones by default
- DDS texture overrides are implemented

The local instancing **.dll**s are built against the **null<platform>** libraries in each case.

This means that some, but not all RenderWare Graphics functions are available. Please consult the platform documentation to determine which operations are legal.

1.5 Summary

Preinstancing converts artwork from a platform-independent to a platform-dependent form. The platform specific form is faster to load and render, and consumes less memory.

Preinstancing can be carried out through the exporters or via the use of the *instance* command line tool.

instance attempts to preinstance locally, and then through Visualizer connections.

The local instancing **`.dll`**s used by *instance* can be customized by users. Source is provided.

The custom instancing **`.dll`**s carry out operations such as creation of texture dictionaries and skin splitting.

instance may be added to art tool chains as part of a build process.