Whitepaper

Granite SDK Version 2.0:

Texture Streaming Middleware

+32 9 247 01 05 | www.graphinesoftware.com | info@graphinesoftware.com

Document Revision History

Revision	Notes
June 6, 2013	First version of this document
February 14, 2014	Second version of this document

Contact us

+32 9 247 01 05

www.graphinesoftware.com

info@graphinesoftware.com

Notice

Trademarks

Copyright

Contents

Introduction

Whether you want to use massive amounts of unique texture data (think Google earth) or you just want to drastically reduce the disk storage footprint of your existing texture images, the Granite SDK texture streaming library can make it happen! Rock-solid and highly optimized for real-time rendering, the Granite SDK enables you to have the extremely detailed object surfaces needed to render in high definition. Why should you use Granite SDK?

- More than halve your storage footprint of your textures!
- Directly use your existing texture bitmaps. No extra artist time required.
- Much shorter loading times for your texture data.
- Use massive amounts of texture data (gigatexels).
- The amount of texture data you can use now depends solely on your storage budget.
- You have fine-grained control over the runtime resources (Disk, RAM, Video Memory) that you allocate for textures, without needing to constrain your artists.
- One elegant system for all your textures (e.g., terrain, characters, etc.).
- Easily integrates in any render engine, graphics API and development pipeline.
- The Granite SDK can easily be added to your game, even in the later stages of the production process.
- Highly scalable over different platforms and systems.

Library Overview

Texture Compression

The core component of the Granite SDK is the Graphine texture compression. Our texture codec is specifically designed for common texture content used in (real-time) rendering engines such as color data, alpha channels and normal vectors. These custom compression techniques can have a much higher compression ratio (at comparable texture quality) than commonly used fixed-rate compression techniques (e.g. DXT1, DXT5 ...). Additionally, the Graphine codec allows different quality settings so that you can control the storage size and quality for a specific texture. This reduces the disc space used by textures by up to 86% (see Figure 1) compared to DXT5.

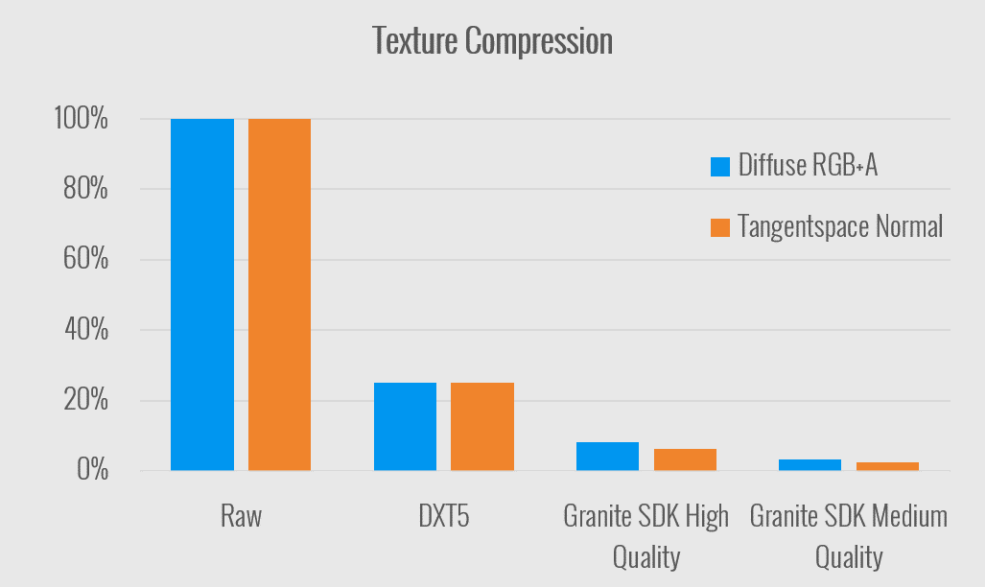

Figure 1: Number of bytes used for diffuse color data plus an alpha channel and normals when using different compression formats. Graphine compression allows for different quality settings and can reduce the storage size significantly.

Transcoding

Modern graphics hardware usually keeps textures compressed in (video) memory to maximize the amount of texture data that can be used. This hardware has support for a limited number of fixed-rate techniques (e.g. DXT1, DXT5 ...) such that sampling from these textures has no performance impact. Many games simply store these compressed textures on disc. However, while these formats are easy to implement decoding hardware for, they are far from optimal with respect to the size on disc.

To reduce the memory use and texture streaming bandwidth, the Granite SDK transcoder module (Quartz) can decompress from our optimized disc format straight to a GPU compatible compressed texture (a process called transcoding). The transcoder is highly optimized for a low CPU impact and uses SIMD optimized code.

The Quartz transcoder module is at the heart the Granite SDK streaming library. Its efficiency allows for large amounts of texture data to be transcoded at runtime when needed by your application. Your application can even be run directly from disk without needing to decompress (i.e., transcode) all data when installing your application.

Streaming

Besides compression, the Granite SDK also focuses on streaming texture data. This allows you to load textures asynchronously during the course of the game without needing to load all the textures up front. This increases flexibility and helps to reduce load times.

To accomplish this, the Granite SDK streaming system manages texture data in small tiles (e.g., 128x128 texels). Each tile can be requested separately by your application. The lay-out of the streaming file that contains all texture data is carefully designed for high data throughput from disk into memory. Tile files can be optimized for different storage devices taking the device's specific characteristics (e.g. cluster size) into account. The Granite SDK supports multiple levels of tile caches (hard disk, RAM, video memory) that can individually be configured in size. This ensures low latency access to the texture data while giving complete control on the amount of resources used at runtime.

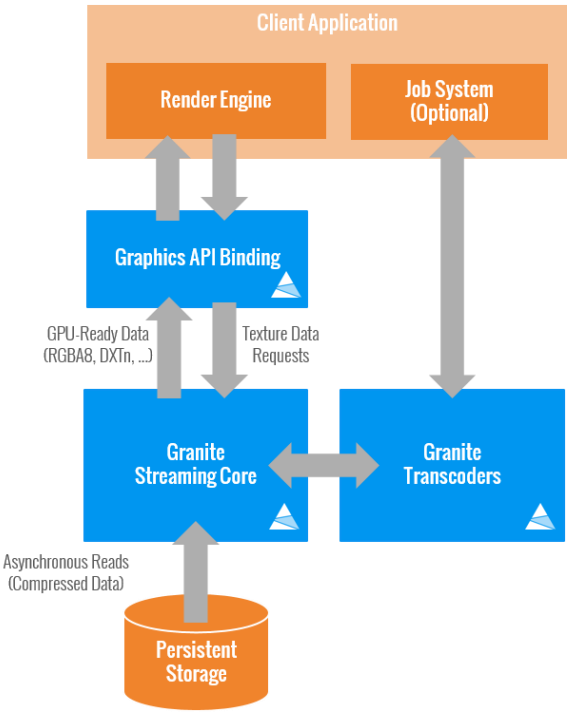

Figure 2: Overview of the Granite SDK streaming system.

Library Uses

The library consists of both production and run-time components. During or after production your source art is imported using our tools. The bitmaps are split into tiles, compressed, and stored in our compressed streaming file. At run-time the library can then be used in two different ways, which are described below.

Figure 3: Production-side use of the Granite SDK Library.

'Classic' Texture Streaming

When using this approach, memory for a texture's mipmap piramid is fully allocated on the GPU. The highest resolution mipmaps could be loaded later but within a mipmap all data is pre-allocated on the GPU. Textures are still streamed from disc and transcoded in tiles (this allows shorter latencies) and uploaded to the GPU as soon as they become available. While this approach doesn't save much GPU memory in itself, the cost of loading textures is reduced allowing you to swap textures in and out more easily. This approach is ideal if you simply want to reduce the storage size and bandwidth use of your textures.

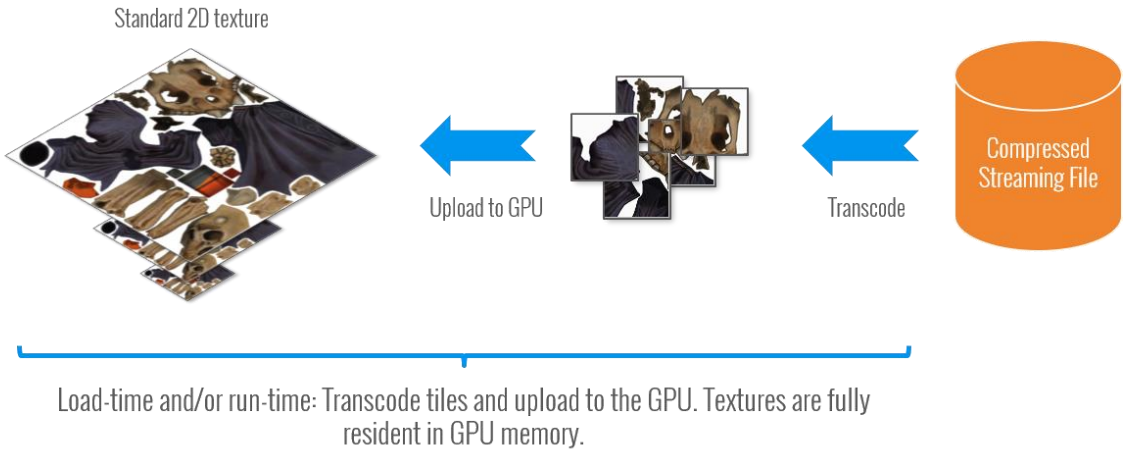

Figure 4: Using the Granite SDK run-time for classic texture streaming.

Partially Resident Textures/Virtual Texturing

By managing individual small tiles of texture data, the Granite SDK virtual texturing library gives you even more fine-grained control over what texture data is resident in video memory. This allows you to use extremely high-resolution textures (gigatexels) that are orders of magnitude larger than what can be stored in (video) memory. This approach is ideal for large landscapes and highly detailed characters.

The core of this system replaces the traditional fully allocated GPU texture (see 'Classic' texture streaming) with a special tiled cache texture. When reading from a partially resident texture a special shader is then used that translates your input texture coordinates into texture lookups in the cache texture. Even when using moderate size textures, the fine-grained control provided by the Granite SDK results in less video memory required for your texture data compared to an ad hoc texture streaming solution.

Since DirectX 11.2 special-function GPU hardware is exposed to accelerate reading from partially resident tiled textures. In addition to DirectX 11.2 this hardware is also available on next-gen platforms as well as in OpenGL through an extension. One of the benefits of using Granite is that you can easily exploit this new hardware when it is available while falling back to a shader-based implementation on older DirectX versions and hardware.

To determine what tiles to load into the cache texture the system first determines what tiles are visible on the screen and at what mipmap resolution they should be loaded. This works fully automatically and does not require manual tweaking or tagging of objects. Any tiles not present in the cache are then streamed from the storage device, transcoded, and uploaded to the GPU cache texture.

A big advantage of this approach is that it is inherently scalable, as texture data is only loaded at the mipmap resolution required on-screen. It automatically loads less data on smaller screen resolutions requiring less manual tweaking or user configuration.

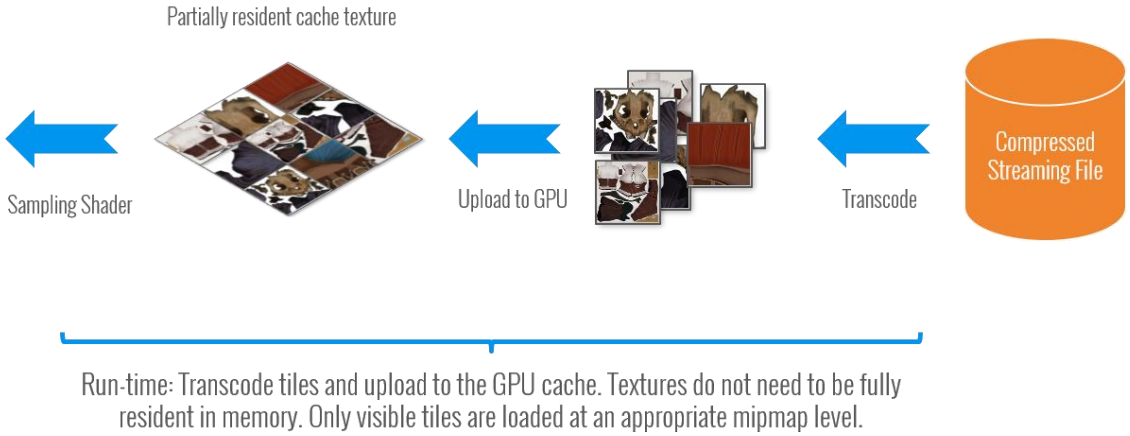

Figure 5: Using the Granite SDK run-time in a partially-resident texturing scenario.

Integration

Engine

The Granite SDK is designed from the ground up to be easy to integrate into existing engines and pipelines. It supports common practices such as easy to overload memory allocations and error reporting. Besides this, it also allows fully customizing its use of threads by integrating with your existing job system. Of course our system comes with a standard multithreading implementation if you don't have a job system in your engine. Besides threading, (asynchronous) disk-IO can also easily be customized allowing the Granite SDK to be used alongside your in-house streaming system our other external streaming middleware.

Tools

Our production pipeline is available both as a command line tool that easily integrates into any build system or as a software library that allows you to build it into any existing tools. Our tools also support all common image file formats (.tga, .bmp, .png, .tiff, .jpg, .exr).

Deep Integration

For popular licensable engines Graphine also offers pre-integrated versions of its middleware. The benefits of such an integration is that we have invested a lot of work in deeply integrating our middleware not only in the engine but also in the editor. When using one of our integrations switching to Granite is almost as easy as just switching on one flag. The Granite tool then will then automatically take care of importing textures in our tiled streamable format, and adjusting materials for rendering from it.