3D*labs*®

# THE VIRTUAL TEXTURE ENGINE

*Texture Management on Permedia®3 & GLINT® R3*

**WHITE PAPER**

**Issue 1**

## Proprietary Notice

The material in this document is the intellectual property of **3D***labs*. It is provided solely for information. You may not reproduce this document in whole or in part by any means. While every care has been taken in the preparation of this document, **3D***labs* accepts no liability for any consequences of its use. Our products are under continual improvement and we reserve the right to change their specification without notice. **3D***labs* may not produce printed versions of each issue of this document. The latest version will be available from the **3D***labs* web site.

**3D***labs* products and technology are protected by a number of worldwide patents. Unlicensed use of any information contained herein may infringe one or more of these patents and may violate the appropriate patent laws and conventions.

**3D***labs* is the worldwide trading name of **3D***labs* Inc. Ltd.

**3D***labs*, GLINT and PERMEDIA are registered trademarks of **3D***labs* Inc. Ltd.

Microsoft, Windows and Direct3D are either registered trademarks or trademarks of Microsoft Corp. in the United States and/or other countries. OpenGL is a registered trademark of Silicon Graphics, Inc. Macintosh and Power Macintosh are registered trademarks and QuickDraw is a trademark of Apple Computer Inc. All other trademarks are acknowledged and recognized.



Email: info@3dlabs.com
Web: http://www.3dlabs.com

**3D***labs* Ltd.
Meadlake Place
Thorpe Lea Road, Egham
Surrey, TW20 8HE
United Kingdom
Tel: +44 (0) 1784 470555
Fax: +44 (0) 1784 470699

**3D***labs* K.K
Shiroyama JT Mori Bldg
16F
40301 Toranomon
Minato-ku, Tokyo, 105,
Japan
Tel: +81-3-5403-4653
Fax: +91-3-5403-4646

**3D***labs* Inc.
480 Portrero Avenue
Sunnyvale, CA 94086, United States
Tel: (408) 530-4700 Fax: (408) 530-4701

**Overview**

Permedia3 and GLINT R3 graphics processors have a hardware texture management system called the Virtual Texture engine. The Virtual Texture engine gives the following application-level benefits:

- Smoother utilization of bus bandwidth to reduce application "stutter."
- Reduces amount of physical on-card memory required to run texture-heavy applications.
- Enables more textures per scene for a given memory size.
- Enables larger textures at greater color depths, up to 2048x2048 at 32-bit color.
- Enables mip-mapping on larger textures, up to 2048x2048 at 32-bit color.

**Justification**

The ability to handle large amounts of texture data is becoming very important for applications in multiple vertical market segments. Although the total amount of textures used in a given application may be quite large, many applications are structured so that only a small area of large texture is to be shown at full resolution. For example when textures are used to produce a realistic flight simulation environment, only the textured terrain close to the viewer has to show fine detail; terrain far from the viewer is textured using low resolution texture levels, since a pixel corresponding to these areas covers many texels at once. For many applications that use large texture maps, the maximum amount of texture memory in use for any given viewpoint is bounded.

Applications take advantage of this fact by doing texture data management in software. While very important, this technique can only be optimized so far. The ideal configuration is a graphics processor that manages texture allocation in hardware and only loads the minimum amount of texture data necessary for rendering.

By only loading the texels necessary for rendering, you get the following technology benefits:

- Use less bus bandwidth than schemes like traditional full texture allocation, AGP texture execute and texture compression.
- Fully utilize all available on-card memory without any of the memory holes associated with allocation of whole textures.

Virtual Texturing does exactly this and provides these technology benefits to the graphics sub-system. The end-user and application benefits outlined in the introduction are derived directly from these technology benefits.

**How it works**

The Virtual Texture engine is analogous to the instruction cache in your CPU. In an instruction cache, the CPU maintains a logical to physical address mapping of program instructions. When the CPU has a miss in the L1 and L2 cache, it DMAs the requested block of instructions from system memory into these cache levels. With Virtual Texturing, the principle is the same except that instead of instructions, it is fetching texture data, and instead of L1 and L2 instruction caches, we have on-chip and on-board texture memory.

Virtual texturing creates a logical address space for texture memory that maps to physical locations on-chip, on-card and in system memory. The maximum size for this logical address space is 256MB and is allocated in 4KB pages. When a texel is required for rendering by the graphics core, the Virtual Texture engine does a table lookup to see where the texture data is located. If the texture is not on-card, the Virtual Texture engine will DMA the required page into on-card memory where it can be accessed at full pixel rates by the graphics core.

Since the Virtual Texture engine operates on 4KB logical pages instead of on whole textures, it allows both optimization of bus bandwidth as well as full use of on-board texture memory.

**Optimization of bus bandwidth**

The Virtual Texture engine only transfers texture data into on-card memory when it is needed. This is different from traditional texture architectures that must load the entire texture onto the card before even the first texel is rendered. The chart below illustrates how texture data is transferred across the bus in a fairly common case.

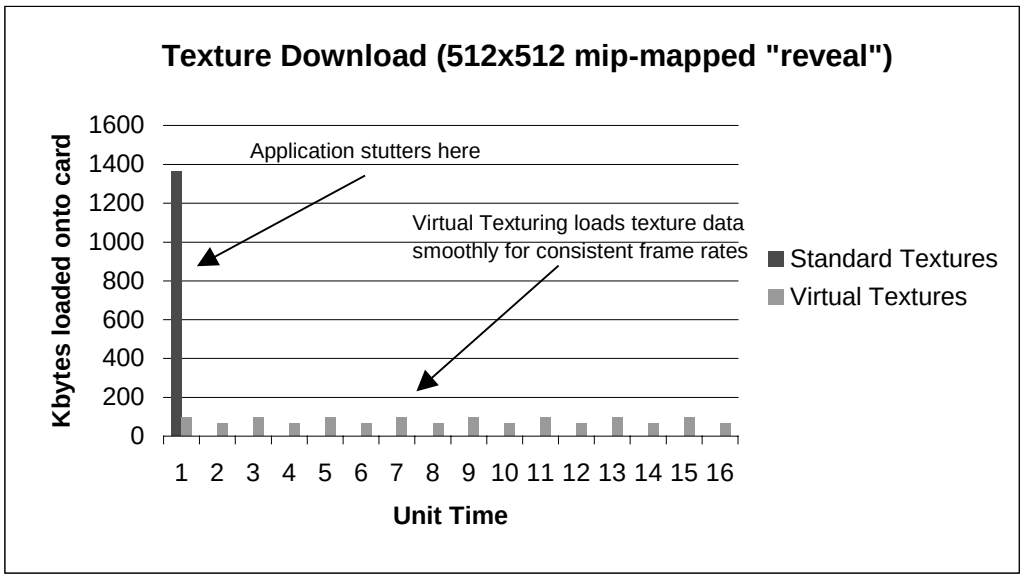


**Chart 1 – Texture download for a 512K texture being revealed from behind an occluding object.**

A 512x512 texel, 32-bit, mip-mapped texture is revealed from behind an occluding object. In the traditional case, the entire texture, including all mip-map levels, must be downloaded onto the card before the first stripe of the "revealed" texture can be rendered. This massive one-time hit leads to a significant drop in application performance that would be perceived as a stutter in frame rate. In contrast, the Virtual Texture engine downloads only the pages holding the currently visible texels from the currently blended mip-map levels. As more of the texture is rendered, the Virtual Texture engine DMAs the texels in smoothly without overwhelming the card bus. Also significant in this example is that since only the current mip-map levels are being loaded, the Virtual Texturing method uses less on-card memory to render the exact same scene.

The next example demonstrates the maximum benefit case for the transfer of texture data. In this case there is a 2Kx2K, 32-bit, mip-mapped texture applied to two sides of a Y corridor split. As the user approaches this split, the texture becomes progressively larger in the view, switching mip-map levels as it comes. In the traditional case, over 21MB of texture data needs to be loaded onto the card before the texture can be drawn. This leads to a delay in rendering so large that the application would become non-interactive. Also problematic, ignoring bandwidth issues, is that it is highly unlikely that there is 21MB of free texture memory on the card, even in 32MB configurations.

Virtual Texturing overcomes both of these problems. First, note that the Virtual Texture engine allows the graphics processor to start rendering after loading only 8KB of data for the first two mip-map levels. As the split in the corridor comes closer, the appropriate mip-map levels are loaded on a per-pixel basis. The result is a smooth utilization of the bus that prevents the application from stuttering. Second, the Virtual Texture engine only needs to keep the current

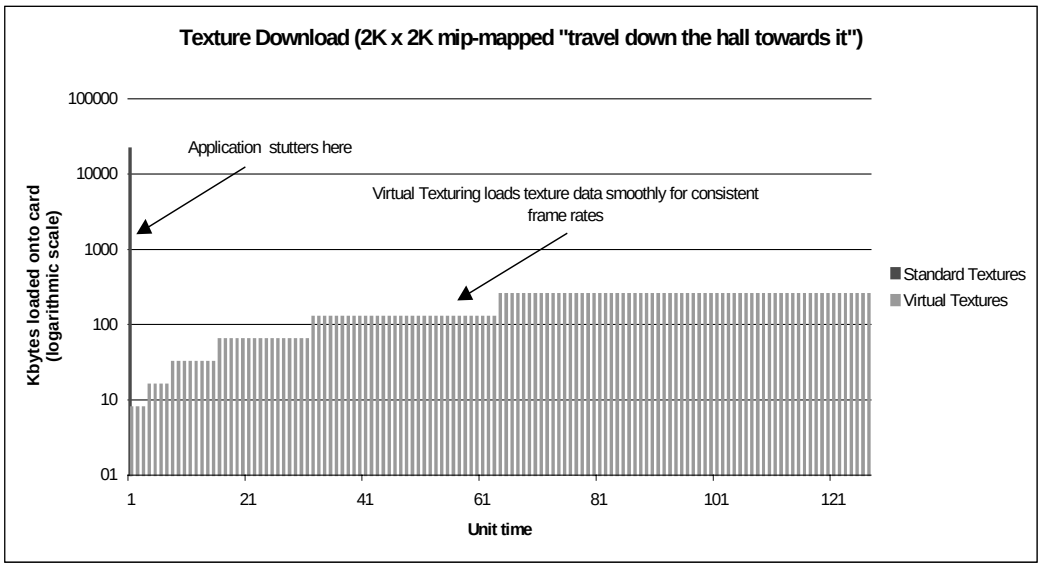


**Chart 2 – Texture download for 2Kx2K mip-mapped texture on a Y in the corridor *(note: log scale)***

mip-map level in memory. This means that even at full resolution, the texture only takes up 16MB on-card as compared to 21MB required by the traditional method.

**Full utilization of on-board texture memory**

Virtual Texturing provides further benefit by allowing all available on-card memory to be allocated for texture data. Traditional architectures require that textures placed in on-card memory be allocated in a contiguous block. This means that even if the card has 512KB of free texture memory that is scattered around in the "holes" between the currently allocated textures, a 512KB texture could not be loaded onto the card because the memory is not all together. By using logical to physical address mapping, the Virtual Texture engine has full random access to the texture data it needs in 4K pages. This allows texture data to be placed into whatever memory is available, even if it is scattered around at different physical addresses.

**Comparison to AGP Texture Execute**

Another way to handle textures which don't fit into graphics memory is to use AGP texturing (AGP execute). In this case, system memory is used as texture memory and texels are fetched across AGP in 128-bit chunks as needed. This allows easy application texture management - just keep all the textures in system memory. Although this method is easy, application performance is still not optimal.

*Bandwidth comparison:*

| **Bus** | **Peak Bandwidth** |
|---|---|
| On-card Memory | 2,000 MB/s |
| AGP 4x | 1,066 MB/s |
| AGP 2x | 533 MB/s |
| PCI-32 | 132MB/s |

The on-card memory bus is much faster than even an AGP 2x bus, so rendering out of on-card memory is required for maximum performance.

Even if the bandwidths were identical, the AGP bus and system memory bus need to be used for things other than fetching texels:

- Program instructions (system memory to L2 and L1 cache)
- Polygon data (system memory and AGP)

- Chip state data (system memory to AGP)

These other uses for the pathways inside the AGP chipset require that the full AGP bandwidth cannot be allocated to transferring texture data. By interleaving 4K texture page reads with the normal AGP chipset datastream and keeping often-used data on-card, the Virtual Texture engine creates an efficient dataflow that keeps applications from stuttering.

**Comparison to texture compression**

One popular method for increasing available texture memory is through texture compression. Although texture compression does allow for more textures in system memory and in on-card memory than traditional uncompressed texture allocation, it still has three major drawbacks:

- Texture compression still allocates textures as whole-chunks, leaving gaps of wasted on-card memory.
- Texture compression still downloads texture as whole-objects, wasting bus bandwidth on texels that will potentially not be used.
- Texture compression requires custom application support.
- Texture compression sacrifices image quality.

Virtual Texturing provides the best of both worlds by combining transparent application support with optimal memory and bus utilization.

**Application use of the Virtual Texture engine**

Virtual Texturing is available for both OpenGL 1.x and Direct 3D 6.x. In both environments, the Virtual Texturing engine operates transparently. OpenGL applications will automatically use it. Direct3D applications will use it if they select the appropriate Direct3D texture storage mode.

*OpenGL*

OpenGL does not include a texture memory management interface. This means that OpenGL applications cannot manage texture memory themselves, and that (without the Virtual Texture engine) the OpenGL ICD has to do it. This is slower, much less efficient, and much less robust than hardware texture memory management.

*Direct3D*

Virtual Texturing is also supported in Direct3D. Developers who use Direct3D will be given the choice of using Virtual Texturing, or managing texture memory themselves. Permedia3 and GLINT R3 support the AGP execute mode and AGP 2x.

**References**

http://www.3dlabs.com/products/index.html – Permedia3 datasheet