Virtual Texturing

13.11.2009
Albert Julian MAYER (e0126505/066932)
Praktikum aus Computergraphik und digitaler Bildverarbeitung (186.168)

The Problem:
To generate high-quality output realtime-rendering applications or games commonly need more texture data than can be simultaneously held in graphics- or even in main-memory.

Other Solutions:
• Repeating/Blending: A common approach in video games is to have a repeating base texture that is blended with other detail textures to simulate the variety of a single high-resolution texture. Although this approach reduces the texture storage requirements it commonly is still higher than the available graphics memory. Also this approach may be visually unsatisfying, requires skilled artist labour and is inapplicable to many applications.

• Texture streaming: The entire texture data may not fit into graphics-memory, but the data needed at any point in time commonly does. Texture streaming dynamically loads and unloads textures to keep only those that are currently necessary in VRAM. Streaming may either happen asynchronously all the time or when changing the virtual viewpoint between predefined areas, stalling the application until the (un)loading is complete. The main problem here is that textures are streamed at a coarse granularity, requiring the whole texture to be available even if only a small portion and/or a low resolution version would be necessary. Furthermore for efficient rendering it is currently preferred to combine the textures of multiple objects to save texture switches (=> ”texture atlas”), but this further intensifies the granularity problem.

Virtual Texturing:
“A virtual texture is a mip-mapped texture used as cache to allow a much higher resolution texture to be emulated for real-time rendering, while only partly residing in texture memory.”
[Martin Mittring “Advanced Virtual Texture Topics”]
Virtual texturing solves the texture memory problem by loading only those *parts* of the mip-chains of the textures that are needed for rendering each frame. Unlike traditional manual texture streaming this happens transparently to the client application, there is no need to give hints about which textures may be viewable from which parts of the virtual world. Since virtual texturing operates on *parts* of the mip-chain the total graphics memory consumption is significantly lowered, being roughly proportional to the number of rendered pixels. Performance is also improved since streaming only the necessary parts instead of whole textures (or whole mip-levels) reduces bandwidth requirements. Furthermore since all of the objects can now share the same enormous virtual texture (atlas) the texture switch problem is also eliminated.

Virtual Graphics Memory:
The graphics memory currently is not virtualized like main memory. Ideally, akin to virtual memory a texture could be stored only partly in graphics memory. If parts of a texture were requested that aren’t available a *page-fault* could happen, asking the application to asynchronously provide the missing data while continuing with a lower-resolution version of the data as fallback.

The Implementation:
Since graphics hardware currently does not feature this kind of texture memory virtualization the facility is emulated using pixel shaders. My implementation (similar and inspired by "Sparse Virtual Textures" [Sean Barrett http://www.silverspaceship.com/src/svt/]) can be roughly divided into three parts:

• Tile Determination: Prior to rendering each frame it must be determined which parts (*tiles* from now on) of the mip-chain of the virtual texture are needed. Currently this happens with a pre-pass with a special pixel shader that calculates coordinates and mip level of the needed tile. The information is streamed to the CPU and processed there. Other tile determination methods include rendering to UV-space or analytic determination.

• Tile Management & Page-table Management: The tiles of the virtual texture that are needed are streamed to a "physical" texture. The physical texture thus contains equally sized tiles of different parts of different mip-levels of the virtual texture. There is only either one mip-map level in the physical texture, or two levels if tri-linear filtering is used. Furthermore a page-table texture is created whose width and height are equal to the width/height of the virtual texture *in tiles*. The page-table texture stores the positions of the tile in the physical texture and must be updated whenever a tile is mapped or unmapped.

• Virtual Texturing Shader: Since the virtual texture doesn't exist in graphics memory the virtual texturing shader samples the page-table texture instead. The result of the lookup is the address of the tile in the physical texture. Now all that is left to do is bit of math for the within-tile coordinates and sampling the physical texture.

Drawbacks: Apart from a lower graphics memory consumption virtual texturing does not provide any benefits for applications where the data already fits into the graphics memory and because of the incurred overhead (especially in the pixel shader) it is not suited to those applications. Also since the physical texture is composed of uncorrelated texture parts, borders must be used to avoid artifacts when using texture filtering or compression. These borders results in wasted texture memory. Precision issues and graphics driver problems represent another challenge.

Delivered Files:

VirtualTexturing/	The base directory
Dependencies/	Contains dependencies like boost::threads for threading
Documentation/	Contains the documentation for the LibVT
LibVT_Documentation.html	Doxygen documentation including usage guide
ProjectLog.pdf	The practical course project diary.
VirtualTexturing.pdf	This very document.
LibVT/	Contains source code for the virtual texturing library
LibVT_Config.h	Adapt the library by changing #defines here
LibVT.h	Public header for apps using LibVT
LibVT.cpp	Contains most of the implementation of LibVT
readback.[frag/vert]	Contains the shader-code for the tile determination
renderVT.[frag/vert]	Contains the virtual texturing shader code
Sources/	The source-code for the Demo with uses LibVT
Resources/	Miscellaneous needed resource files for the Demo
Win32/	Support files like DLLs for (cross)compiling for Win32