Texture-size-independent address translation for virtual texturing

Charles-Frederik Hollemeersch[*] Bart Pieters Aljosha Demeulemeester Peter Lambert Rik Van de Walle
Ghent University - IBBT ELIS - Multimedia Lab[†]

1 Introduction

Virtual texturing (VT) is a promising technique to increase texture resolution and uniqueness in real-time applications such as GIS and games. VT manages large texture data sets by splitting the texture data into smaller tiles and assigning a unique address to every tile. Only the visible subset of these tiles is then kept in graphics memory. Any newly-visible tiles are loaded on-demand from disc. Several recent games such as id Software's Rage and Splash Damage's Brink have used VT to manage their texture data at run-time.

We first briefly describe how the render is traditionally implemented for VT [van Waveren 2009] [Sugden and Iwanicki 2011] [Hollemeersch et al. 2010]. When sampling the VT in the shader, we first calculate the tile address based on the texture coordinates. This tile address is then used to do a look up into the translation lookup table (TLT). The TLT is itself a texture that contains a texel for every tile in the virtual address space. The contents of the TLT's texels give us the location of the tile in a second texture containing the currently loaded tiles. The final sample color value is then determined by a lookup into this cache texture.

One of the main promises of virtual texturing is that the run-time resource demands are no longer relative to the whole dataset size but instead become relative to the screen size. While currently existing practical implementations largely fulfill this promise, the TLT is one key step which is still dependent on the texture dataset size. In all the aforementioned implementations, a single entry is needed in the TLT per virtual tile address. When using very large virtual address spaces this step quickly becomes a bottleneck since both generation and memory use of the TLT become prohibitively expensive. For example for a $10^6 \times 10^6$ texture with 128×128 pages, the TLT alone already uses 341 megabytes of data. This is in the order of the cache size needed for HD screen resolutions and could thus be better spend on caching more data.

In this work, we present a new approach to eliminate the TLT. Instead of having a data structure that contains an entry for every virtual page, we propose a system that allows efficiently querying if a page is available in the cache. The rest of the virtual texturing process, streaming, caching, and the final look-up in the cache tile are largely unmodified. We will show that our method is sufficiently fast for real-time use on current-generation graphics hardware.

2 The Cache Query Tree

There are several ways to efficiently implement querying if an item is available in a list. Caches and balanced search trees are two common ways to approach this problem. However, when implemented on a GPU, such systems reduce efficiency due to lots of diverging branches on the massively-parallel architecture of the GPU.

Since we know the maximum number of items up front (i.e. the number of tiles that fit in the cache) we can easily use a sorted list instead. This list can then efficiently be searched with a binary search in a fixed and constant number of steps. This search can also easily be implemented without diverging code paths in the pixel shader threads. In fact, our system can be implemented without any branching at all making it amenable to platforms which do not support dynamic branching such as the WebGL standard or embedded hardware.

Practically, our system consists of two data structures. The first is a search table that contains the tile addresses of the tiles present in the cache. This list is sorted by increasing tile addresses. The second data structure is a list where the i'th element contains the cache address of the i'th element in the search table. When T is the opaque address of the tile to find and N is the number of cache items we can find the item as follows:

```
$C =$ unavailable
$Min = 1$
$Max = N$
for $i := 1$ to log($N$) step 1 do
    $Mid = (Min + Max)/2$
    $id =$ tex1d($searchTab, Mid$)
    if $id == T$ then $C =$ tex1d($searchTab, Mid$) fi;
    if $T > id$ then $Min := Mid + 1$;
                else $Max := Mid - 1$; fi
od
```

Note that in practice these two tables are stored as a $N \times 2$ texture. Since log(N), is a constant the loop can easily be unrolled and implemented without branching, using only a single texture read.

Its up to the application to handle the situation where a page is not available in the cache. For example, in our application we currently do another search for a lower resolution version of the data on the next mipmap level. Note that only about 1% of the pixels need to follow this path since unavailable tiles will be streamed in by the cache manager.

3 Results

We have implemented the proposed method in our virtual texturing system. Our updated system now supports almost arbitrary texture address spaces. Most importantly, run-time performance and memory use are only dependent on the screen resolution. We did not notice any significant performance degradation compared to our traditional VT implementation since our system bottleneck is not the VT shader.

References

HOLLEMEERSCH, C., PIETERS, B., LAMBERT, P., AND VAN DE WALLE, R. 2010. Accelerating virtual texturing using cuda. In *Gpu Pro: Advanced Rendering Techniques*. ch. 10.2, 623–641.

SUGDEN, B., AND IWANICKI, M. 2011. Mega meshes: Modelling, rendering and lighting a world made of 100 billion polygons. In *Game Developers Conference 2011*.

VAN WAVEREN, J.-P. 2009. id tech 5 challenges. In *SIGGRAPH 2009 Beyond Programmable Shading Course Notes*.

[*]charles.hollemeersch@elis.ugent.be

[†]The research activities that have been described in this work were funded by Ghent University, the Interdisciplinary Institute for Broadband Technology (IBBT), the Institute for the Promotion of Innovation by Science and Technology in Flanders (IWT), the Fund for Scientific Research-Flanders (FWOFlanders), and the European Union.

SIGGRAPH 2012, Los Angeles, California, August 5 – 9, 2012.
ISBN 978-1-4503-1435-0/12/0008