Combining Texture Streaming and Run-Time Generation

Aljosha Demeulemeester[1] and Charles-Frederik Hollemeersch[1] and Bart Pieters[1] and Peter Lambert[1] and Rik Van de Walle[1]

[1]Department of Electronics and Information Systems, Multimedia Lab, Ghent University - IBBT,
Gaston Crommenlaan 8 bus 201, B-9050 Ledeberg-Ghent, Belgium

Abstract

Virtual texturing systems have enabled gigapixel resolutions for textures used in real-time rendering. This allows for more detail and diversity in virtual worlds at the cost of greatly increasing disc storage requirements. In this paper, we propose a compression method that generates parts of these large textures at run-time using the original painting primitives (brushes) from the texture production process. This way, large featureless texture areas can be compressed to a few brush resources. A post-production analyzing phase determines those areas that should be stored as compressed texel data to ensure real-time performance at run-time.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

1. Introduction

Texture mapping is widely used to enrich the quality of real-time rendered graphics. Because of limited video memory, the textures of large surfaces (e.g., terrain) are usually procedurally generated by tiling and blending multiple smaller textures at run-time using the GPU. However, real-time procedural generation techniques are unable to generate highly detailed and diverse surface textures. Virtual texturing [HPLV10, Mit08, Bar08] offers a solution by enabling gigapixel resolutions for textures used in real-time rendering. This is accomplished by dividing the high resolution textures into small tiles (e.g., 128x128 texels). Tiles that are needed by the visualization are streamed on demand from disk to memory. However, two issues arise when using gigapixel textures in synthetic virtual environments.

- New production tools are required to allow efficient painting of these huge textures by artists (generation side).
- Gigapixel textures - frequently having eight or more channels per pixel - require a large amount of disk storage (storage and distribution side).

The Infinitex system [HPD*10] is a texture painting tool for artists. It provides painting primitives (brushes) that are a combination of source images, a shader and a blend mode (frequently called a material in modern rendering engines). Brush instances - defined by a brush ID, a coordinate and radius in texture space, and a set of parameters - are interactively added by the artists. These brushes can use information of the underlying terrain (e.g., slope, height, etc.) to procedurally generate the texel data (e.g., rocky surface on vertical slopes). By adding only a few brush instances that together cover the entire texture space, a base layer can be created that equals the quality of current generation of real-time procedural terrain texturing systems. All instance data is stored in a database so that texture tiles can be recomposed at any time.

To tackle the storage requirements for large gigapixel textures used in a virtual texturing system, we investigate a compression method that composites some of the tiles (i.e., procedural tiles) at run-time at the end-user side using the original painting primitives from the texture production process. Only the tiles for which this in not feasible due to the real-time constraint are composited offline and stored as texel data (i.e., baked tiles). We discuss the analyzing phase that selects the tile type after the texture has been painted. Also, we discuss the rendering at the end-user side.

2. Virtual Texture Tile Analysis for Compression

In conventional virtual texturing systems (using only baked tiles), the tiles are compressed using image compression algorithms. Still, this results in very large file sizes, even for moderate texture dimensions (e.g., 1.7GB for a 122880^2 24-bit RGB texture). To reduce the sizes on disk, 3D scene and game play analysis is used to aggressively throw away tiles that are never in view. These are usually tiles from higher

DOI: 10.2312/conf/EG2012/posters/003-004

resolution mip map levels. To even further reduce the storage requirements, we introduce a second phase that finds tiles for which the data needed for their composition requires less storage then their compressed image representation. This new phase consists of the following steps which will be discussed below. First, the data for compositing the tiles is reduced. Second, tiles are filtered out that cannot be composited at run-time in real-time. Finally, tiles are marked for run-time compositing based on the resulting impact on the total storage requirement.

2.1. Reduce Brush Instances

Artists can easily add thousands of brush instances to a texture region the size of one tile. This could result in instances that are completely covered. To reduce the total compositing time and required brush resources for a tile, a list of brush instances is created that actually contribute to its final texel data. Only instances that intersect with the tile in texture space are eligible. A spatial database query filters these out. To detect if the remaining instances really contribute to the tile, for each instance, a list of texels is created that it affects. The instance contributes if at least one texel value is not completely overwritten by another brush instance. Also, brush instances that are much smaller than a texel could potentially be dropped without impacting perceivable texture quality. Tiles from lower resolution mip map levels are usually covered by large number of these instances. However, a noticeable discontinuity is introduced between texture mip map levels if instances are not removed in all levels. Only if the first phase has thrown away the tiles of all higher resolution mipmap levels in the area covered by a brush instance can it be removed without visual impact.

2.2. Filter by Compositing Time

The time it takes to composite a tile should be under a certain threshold for a specific target platform to ensure adequate real-time performance. The threshold depends on the allocated time for tile compositing at run-time. The collective compositing time of tiles that come into view during the same frame should not exceed the allocated time. However, we currently use a conservative maximum threshold for each tile: maximum collective compositing time during one frame divided by the maximum amount of new tiles in view.

2.3. Filter by Required Resources

All brush resources (e.g., images) need to be available when compositing a tile covered by an instance of that brush. This adds to the overall storage requirements. By analyzing tile coverage for each brush, a heuristic method can search for an optimal selection of procedural tiles to minimize the storage size.

3. Run-Time Tile Compositing

The end-user texture run-time extends our virtual texturing system similar to [HPLV10] with a tile compositing module. The system has a tile cache in video memory (i.e., texture) that is managed to contain all tiles that were recently needed. To update the cache, procedural tiles are composited by the GPU using the brush resources and the brush instance data for those tiles. Baked tiles are loaded from storage and uploaded to video memory. The nature of the system results in cached texel data that can be reused multiple frames, as long as the tile remains in view and in cache. For a fast rotating (90 degrees/s) or translating (50km/h) camera rendering in HD resolution, tiles remain in view for 15 frames on average when using 2 mm texel size for a terrain texture. Early results indicate that the caching enables a performance advantage opposed to traditional texture compositing during forward rendering. This compensates for the rendering overhead added due to the deferred texturing.

4. Conclusions and Future Work

We can conclude that compositing texture tiles at run-time in real-time is feasible for a virtual texturing system. Featureless areas of the texture can be compressed to the size of the required brushes and the covering brush instances. Enabling the combination of offline and at run-time composited texture tiles allows for specific regions in the texture to receive unlimited detail by the artists. In the future, we will focus on the heuristic method that searches for the tile type assignation that requires the minimal amount of disk storage. To evaluate the attained compression, gigapixel textures will be analyzed that were painted by artists.

Acknowledgments

This research was 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 (FWO-Flanders), and the European Union.

References

[Bar08] BARRETT S.: Sparse virtual textures. In *GDC 2008 presentations* (February 2008). 1

[HPD*10] HOLLEMEERSCH C.-F., PIETERS B., DEMEULEMEESTER A., CORNILLIE F., VAN SEMMERTIER B., MANNENS E., LAMBERT P., DESMET P., VAN DE WALLE R.: Inifinitex: An interactive editing system for the production of large texture data sets. *Comput. Graph. 34* (2010), 643–654. 1

[HPLV10] HOLLEMEERSCH C.-F., PIETERS B., LAMBERT P., VAN DE WALLE R.: Accelerating virtual texturing using CUDA. In *GPU Pro : advanced rendering techniques*, vol. 1. AK Peters, 2010, pp. 623–641. 1, 2

[Mit08] MITTRING M.: Advanced virtual texture topics. In *ACM SIGGRAPH 2008 classes* (New York, NY, USA, 2008), SIGGRAPH '08, ACM, pp. 23–51. 1