Karlsruhe Institute of Technology

Paper Out of Core Processes

Proseminar Paper by

Jonathan Beller

At the Faculty for Computer Science
Institute for Visualization and Data Analysis,
Computer Graphics Chair

22. Mai 2018

Inhaltsverzeichnis

1. Abstract

With the ever increasing amount of data to be rendered for a single given scene, experimenting with new approaches for handling and structuring that data becomes more and more important. This paper covers the approach to enhance response times and quality using out-of-core techniques.

With the help of out-of-core techniques, it is possible to achieve real-time rendering performance on current-generation graphics cards by overcoming limits posed by the graphics cards' internal memory [CNLE09]. Thus, this topic is essential for everyone hoping to achieve such performance for scenes that cannot be efficiently rendered using conventional methods.

I will discuss different strategies used to load necessary data chunks into working memory taking into account its finiteness using intelligent data streaming. These include visibility culling and virtual texturing. Also, this paper will give an overview of the possibility to use volumetric texture units called "Voxels" instead of regular texels and the advantages and challenges that come with it. Lastly, the topic will be further explored using the examples [CNLE09] and [VW09].

2. Motivation

Over the course of the last years, the complexity of scenes to be rendered has risen further and further. Nowadays, performance and complexity demands cannot be satisfied by solely utilizing brute force methods and trying to fit all necessary data into the scarce graphics memory. There are a number of specific use cases where another approach is needed. For example, [CGG+03] look for a way to interactively render very large landscapes in the context of virtual reality using commmodity graphics hardware. Further, [GM05] name the fields of 3D scanning, computer-aided design and numerical simulation.

In essence, all of these use cases require a technique able to expand available memory by making use of larger but slower storage. While doing this, the searched technique should also minimize response times which are bound to increase due to longer memory access times. A technique that finds a suitable compromise between memory extension and response times is called an Out-of-Core technique.

3. Basics

This chapter aims to introduce key terminology that will be used in this paper. The chapter gives a basic understanding of the terms “Texture Mapping”, “Memory Virtualization”, “Virtual Texture” and “Level of Detail”.

3.1 Texture Mapping

Texture mapping provides a way of adding detail to rendered surfaces without increasing geometric complexity. In its simplest form, it takes a 2-dimensional image called *texture map* and a 3-dimensional model and paints the image onto the model. *Texture coordinates* are used to determine which point on the texture map corresponds to a point on the model.

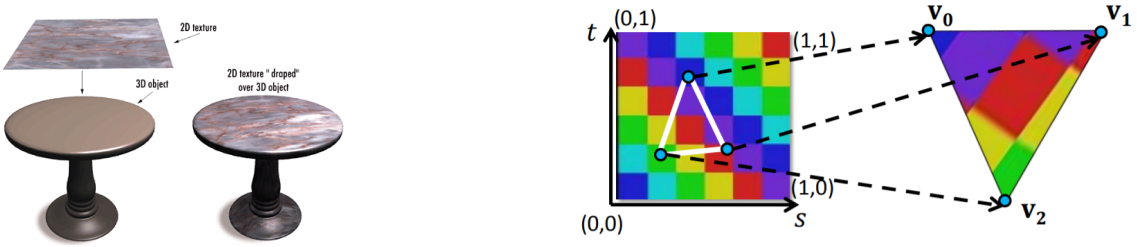

Abbildung 3.1: Left: A texture is mapped onto the table model, making it more realistic. Right: 3D points are translated to texture coordintates to determine vertex color. Images from [Tex] and [Dac17]

The first notable paper about texture mapping was published by [Cat74] in 1974. After that, texture mapping became particularly popular in video games of the 90’s and is now a crucial part of the vast majority of computer-rendered scenes. Because of its usefulness, the concept of texture mapping was extended from affecting the diffuseness to also altering other aspects like specularity, lighting and normals.

3.2 Memory Virtualization

Memory virtualization is the process of abstracting from the physical memory available. The initial motivation for this concept was to enable the uniform usage of heterogeneous

storage devices [Eul08]. However, it also has more advantages. Firstly, memory fragmentation is reduced, decreasing average memory access times. Secondly, virtual memory can combine (graphics) main memory and slower storage and thus increase the total space available to load data into. Other aspects of memory virtualization are beyond the scope of this paper. The most important aspect of memory virtualization regarding rendering large amounts of data is its ability to extend available memory as this solves one of the issues mentioned in 2.

The Windows Display Driver Model v2.0 supports graphics memory virtualization under Windows 10 [Mic18]. However, page faults cause rendering to halt until the requested page is loaded. This makes this solution unsuitable on its own as it reduces responsiveness.

3.3 Virtual Texture

A virtual texture is, in general, a texture too large to fit into the physical graphics memory that is available in virtual memory.

Abbildung 3.2: Image only shows a tiny portion of the actual virtual texture. Different levels of detail can be seen. Image from [VW09]

The advantage of a virtual texture over a conventional physical texture is its ability to hold significantly larger amounts of rendering data. For example, id Tech 5 uses virtual textures that consist of approximately 16.3 billion texels [VW09]. Without special handling, virtual textures face the same responsiveness issues as virtual memory.

3.4 Level of Detail

Level of Detail refers to altering an object's complexity between rendered frames. An example of this technique applied to textures is mip-mapping.

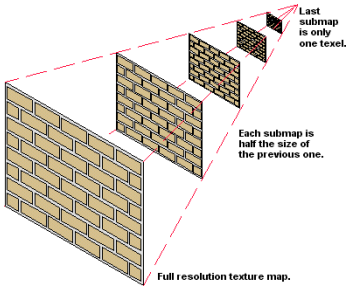

Abbildung 3.3: A texture is saved at different resolutions. As the viewer moves closer, resolution is increased and vice versa. Image from [PCM18]

Level of Detail addresses the issue of increasing scene complexity as the viewpoint moves away from a limited clip and its environment moves into view. Decreasing the complexity

of the objects furthest away from the camera increases rendering performance while not significantly affecting quality [Uni17]. Hence, Level of Detail can be used to address the aspect of responsiveness mentioned in 2.

4. Main Content

4.1 Memory Management

This section covers ways to utilize slow external memory to be able to store large datasets without losing wanted performance.

Once datasets reach a certain size it simply becomes impossible for them to be held inside the graphics card's main memory. This implies the need for using slower storage devices to reach sufficient capacity. However, simply separating the dataset in question into chunks small enough to be completely loaded into working memory leads to a worst case performance bad enough to prohibit real time performance. In conclusion, *intelligent data streaming* has to be used to ensure having all needed data at hand without significant performance losses [CNLE09]. The trade-off to consider here is transferring as little data as possible at a time to minimize processing delays on the one hand and keeping the number of necessary data transfers at a minimum on the other hand. A number of approaches to solve this issue have been developed.

A rather simple solution to extending the available memory is to transfer the idea of virtual pages and physical frames used by operating systems into the given context.

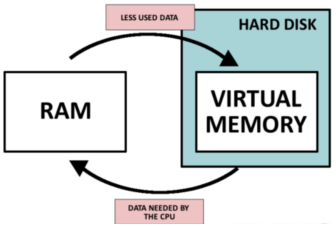

Abbildung 4.1: RAM is extended with part of the hard disk. Data is streamed to the RAM on demand, being exchanged for less used data. Image from [Tec]

This introduces a small unit of data chunk that can be loaded into working memory on demand rather quickly. Furthermore, replacement strategies known from the operating system context can also be used, ensuring that often used data is kept in the graphics card memory as much as possible. However, for very large virtual textures even these small chunks lead to physical memory oversubscription. This, in turn, causes page thrashing

which results in new frames being processed at a high latency [VW09]. As most datasets requiring Out-of-Core techniques have comparable texture sizes, this approach is not sufficient.

A more sophisticated approach is to continuously monitor the physical space demand and reduce some portions of information if the graphics card memory capacity is exceeded. For this, both a policy of portioning the data and a strategy for determining which piece of information should be compressed need to be formed. The portioning can be achieved by *tiling*, i.e. cutting the virtual texture into many small tiles of equal size. In general, Level of Detail is used for determining which texture tiles can be swapped and can, for example, be combined with mip-mapping [CNLE09]. For instance, as the viewer moves away from the scene more texels move into view and therefore have to be loaded. However, the pixel footprint of the texels that are already loaded becomes smaller. This means that less detail is required and the relevant parts of the texture can be replaced with lower-resolution versions if memory becomes scarce. This solution provides a rendering algorithm that is able to adapt to changing memory demands while limiting performance loss. Details about the storage of textures and update policies will be discussed in the examples.

4.2 Voxels

This section gives a detailed overview of the concept of voxels. It outlines their differences to texels, advantages, challenges and their suitability for out of core processes.

The term *voxel* firstly differs from the texel in that its prefix is taken from the word "volume" rather than "texture". Thus, it does not stand for a 2-dimensional element of a flat texture but makes up part of a 3-dimensional texture volume. Analogous to a texel often having the shape of a square, a voxel can, but does not have to be cubic.

Abbildung 4.2: Left: A cube textured with a texture made up of texels. Right: A cube textured with a texture made up of voxels. Images from [Bej] and [Ing]

Obviously, a 3-dimensional object made up of voxels requires much more memory space than the 2-dimensional projection of that object using equal-sided texels would. This is why the use of voxels at a reasonable resolution only became a realistic option for commodity hardware with the conception of Out-of-Core techniques. As rendering of voxels is very costly, some optimizations have to be made in order to make them a viable option. The probably most important thing to note is that not the whole texture volume looked at has to be in memory. For instance, voxels that are fully covered by opaque voxels in front of them are not visible anyway and can therefore be ignored while processing the frame.

As can be seen in figure 4.2, a huge advantage of using voxels is that they do not have to depend on base meshes in order to display 3-dimensional bodies. Displaying a comparable

3-dimensional scene using texel-based texturing would require quite a complex base mesh geometry, placing more load on the graphics card. However, one drawback of voxel-based respresentation is body deformation, which requires special handling and is easier done with a base mesh [LK11]. Voxels also have the very simple advantage of being capable of modeling terrains with overhanging cliffs or caves which is impossible using height maps. Voxels can be rendered as points, for instance in 3-dimensional scanning where they form point clouds. With a sufficient number of samples making up one of those point clouds, voxels make it possible to generate virtual solid objects without requiring precomputed model made up of triangles. Another advantage of voxels is their regular structure. This facilitates the solution of aliasing problems that would be hard to handle using triangulated models [CNLE09]. [GM05] wrote about a fully adaptive technique for rendering very detailed scenes whose textures are made up of voxels and named it “Far Voxels”.

4.3 Voxel-based Octrees

This section gives an overview of how the concepts of voxels and octrees can be combined to build a very advantageous representation structure for 3-dimensional scenes.

A voxel-based texture can potentially require large amounts of memory. Furthermore, especially for objects in the scene far away from the viewer the full resolution is not required. Octrees lend themselves as intuitive structures for adaptive level of detail control.

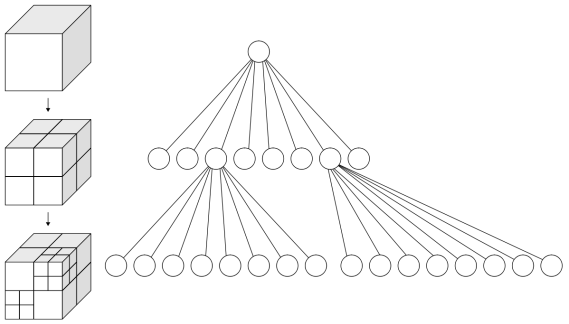

Abbildung 4.3: Illustration of the structure of a voxel-based octree. Image from [Wik]

As can be seen in figure 4.3, a node inside an octree can represent a 3-dimensional body - in this case a cube. In order to acquire the elements represented by the eight child nodes of this node, that body is sliced along the middle in all three dimensions, yielding eight cubes with half the parent cube’s side length each. With this rule, a tree structure can be built by having a source 3-dimensional body and slicing it up to get as many depth levels as needed.

This concept can be transferred to voxel-based scene representation by taking a voxel as the smallest possible cube that can be represented by a node and taking the scene itself as the source body, following the procedure outlined above to generate an octree. However, the resulting octree does not provide any advantages yet as the information represented by it is redundant. The first levels contain very big data chunks and only the lower ones become more reasonable. This can be changed by, for example, setting a resolution limit for each node and downsampling represented data that exceeds this limit using filtering which works well with voxel-based textures. The octree resulting by following these steps contains a low-resolution version of the scene in its root node and resolution increases further and further as one descends along any path of the tree while the represented scene proportion

decreases. While this specific approach may not be ideal for maximizing performance or memory efficiency it still shows how octrees can be combined with voxels to create a useful data structure for the representation of 3-dimensional scenes.

4.4 Example 1: GigaVoxels

This section uses the discussed concepts to explain how GigaVoxels can be used to render large volumetric datasets with high performance. It will explain how this is related to both voxels and out of core processes.

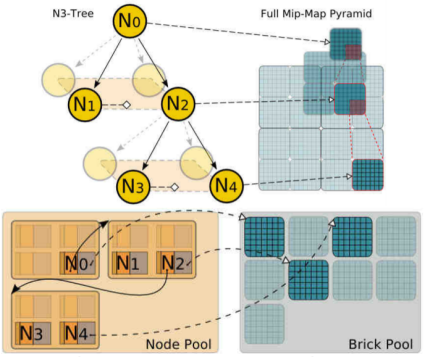

Abbildung 4.4: Illustration of relation between the tree structure and brick detail. Image from [CNLE09]

[CNLE09] came up with a structure that allows for interactive to real-time rendering of scenes with resolutions of 16384^3 and higher. They also further increase detail by making use of procedural methods. One building unit of the algorithm they conceived are *bricks*, which are small 3-dimensional grids of voxels approximating the part of the texture they represent. To get to the data needed for a specific part of the scene, they build and traverse an octree that stores pointers to such bricks inside its nodes or simply constants to indicate empty space. All loaded bricks of the scene are stored in the graphics card memory. However, this may exlude some bricks invisible due to being outside the view frustum or simply being covered - pointers to these bricks are invalidated.

When the viewpoint changes, unavailable bricks are loaded on demand and level of detail of each loaded brick is adjusted as appropriate and needed in order to not oversubscribe memory. This adjustment is made exploiting the special structure of the octree: as the wanted mip-map level is searched, lower mip-map levels are passed in ascending order. This enables one to use them in calculations made to create a smooth transition between different tree configurations. If a missing brick is encountered during ray marching, the Z-buffer is used to block further execution along that ray until the required brick is loaded. In order to be able to continue execution from the last state, status information of the blocked ray is saved in a compressed format and communicated to the CPU.

By preferring higher mip-map levels over missing bricks of the same section, they finally introduce a moderate level of approximation to limit slow data transfer to an extent at which the promised interactiveness can be achieved.

4.5 Example 2: ID Tech 5 Challenges

This section discusses in detail how out of core rendering is compatible with parallelization using the ID Tech 5 Challenges example. It summarizes the faced challenges and how they were solved.

[VW09] decided against using conventional virtual paging, because their textures with a side length of 1024 pages would have led to inacceptable performance due to thrashing. Instead, they went for the level of detail approach. With their solution, texture tiles becoming more detailed are upsampled first; further detail only starts being loaded after the higher-resolution tile has been acquired in the background.

As a frame of the scene would have required 38 milliseconds to be rendered at the required quality as opposed to a maximum available rendering time of 16 milliseconds in order to achieve 60 frames per second, parallelization became a necessity. Therefore, they developed a very invasive but highly scalable job processing system. Jobs are designed to never stall execution, be completely independent of other jobs and always complete - thus minimizing processing delays and bottlenecks.

The id Software team's approach shows that Out-of-Core techniques are indeed compatible with parallelization and therefore suited for future development as parallel programming becomes more and more crucial.

5. Epilogue

As demonstrated, Out-of-Core techniques offer a very promising way of utilizing commodity graphics cards' potential in order to reach a renderable scene complexity far beyond what would be possible otherwise.

A possible future step would be to elaborate on [CGG+03]'s approach and exploit the next generations of graphics cards to create an even more realistic and immersive virtual reality experience. Also, further compression of output data representation and better prediction of occlusion and visibility events would be topics worth looking into [GM05].

Literaturverzeichnis

[Bej] Andres Bejarano: *OpenGL Examples*. https://www.cs.purdue.edu/homes/abejara/courses/cs334/index.html. Accessed: 2018-05-15.

[Cat74] Edwin Catmull: *A subdivision algorithm for computer display of curved surfaces*. Technischer Bericht, UTAH UNIV SALT LAKE CITY SCHOOL OF COMPUTING, 1974.

[CGG+03] Paolo Cignoni, Fabio Ganovelli, Enrico Gobbetti, Fabio Marton, Federico Ponchio und Roberto Scopigno: *Interactive Out-of-Core Visualisation of Very Large Landscapes on Commodity Graphics Platform*. In: Olivier Balet, Gérard Subsol und Patrice Torguet (Herausgeber): *Virtual Storytelling. Using Virtual RealityTechnologies for Storytelling*, Seiten 21–29, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg, ISBN 978-3-540-40014-1.

[CNLE09] Cyril Crassin, Fabrice Neyret, Sylvain Lefebvre und Elmar Eisemann: *GigaVoxels: Ray-guided Streaming for Efficient and Detailed Voxel Rendering*. In: *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games*, I3D '09, Seiten 15–22, New York, NY, USA, 2009. ACM, ISBN 978-1-60558-429-4. http://doi.acm.org/10.1145/1507149.1507152.

[Dac17] Prof. Dr. Ing. Carsten Dachsbacher: *Kapitel 4: Texture Mapping*. 2017.

[Eul08] *Some History about Virtual Memory*. http://euler.vcsu.edu:7000/14315/, 2008. Accessed: 2018-05-14.

[GM05] Enrico Gobbetti und Fabio Marton: *Far Voxels: A Multiresolution Framework for Interactive Rendering of Huge Complex 3D Models on Commodity Graphics Platforms*. ACM Trans. Graph., 24(3):878–885, Juli 2005, ISSN 0730-0301. http://doi.acm.org/10.1145/1073204.1073277.

[Ing] Ingen: *Voxel People*. https://cartrdge.com/zmei/voxel-people. Accessed: 2018-05-15.

[LK11] Samuli Laine und Tero Karras: *Efficient sparse voxel octrees*. IEEE Transactions on Visualization and Computer Graphics, 17(8):1048–1059, 2011.

[Mic18] *GPU virtual memory in WDDM 2.0*. https://docs.microsoft.com/en-us/windows-hardware/drivers/display/gpu-virtual-memory-in-W, 2018. Accessed: 2018-05-14.

[PCM18] *Level of Detail (LOD)*. https://www.pcmag.com/encyclopedia/term/47078/mip-mapping, 2018. Accessed: 2018-05-14.

[Tec] *7. Virtual memory*. http://www.teach-ict.com/2016/GCSE_Computing/AQA_8520/3_4_computer_systems/344_systems_architecture/memory/miniweb/pg7.htm. Accessed: 2018-05-15.

[Tex] *texture map - Computer Definition.* `http://www.yourdictionary.com/texture-map`. Accessed: 2018-05-14.

[Uni17] *Level of Detail (LOD).* `https://docs.unity3d.com/Manual/LevelOfDetail.html`, 2017. Accessed: 2018-05-14.

[VW09] JMP Van Waveren: *id tech 5 challenges-from texture virtualization to massive parallelization.* Talk in Beyond Programmable Shading course, SIGGRAPH, 9:5, 2009.

[Wik] Wikipedia: *Octree.* `https://en.wikipedia.org/wiki/Octree`. Accessed: 2018-05-19.

Erklärung

Ich versichere, dass ich die Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht und die Satzung des KIT zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet habe. Die Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt und von dieser als Teil einer Prüfungsleistung angenommen.

Karlsruhe, den 22. Mai 2018

(Jonathan Beller)