GPU-BASED FRAMEWORK FOR DISTRIBUTED INTERACTIVE 3D VISUALIZATION OF MULTIMODAL REMOTE SENSING DATA

Martin Lambers and Andreas Kolb

Institute for Vision and Graphics, University of Siegen, Germany
{lambers,kolb}@fb12.uni-siegen.de

ABSTRACT

Interactive visualization of remote sensing data allows the user to explore the full scope of the data sets. Combining and comparing different modalities can give additional insight.

In this paper, we present a 3D visualization framework for interactive exploration of remote sensing data. Data from different modalities can be combined into a single view. The visualization can be distributed across multiple graphics processing units and/or hosts, allowing interactive exploration of remote sensing data in virtual reality systems.

Index Terms— Visualization, Virtual Reality

1. INTRODUCTION

Interactive visualization of remote sensing data encompasses both interactive navigation in very large data sets and interactive adjustment of visualization parameters. For example, interactive adjustment of dynamic range reduction methods for Synthetic Aperture Radar (SAR) amplitude images can reveal or accentuate details that might otherwise be overlooked [1]. Furthermore, interactive combination of multiple data sets from different sensors facilitates tasks such as comparison, interpretation, and quality assessment.

The programmable graphics processing units (GPUs) of today's commodity graphics hardware provide the computational power that is necessary to interactively process, combine, and render remote sensing data [2].

In this paper, we present a framework for GPU-based interactive 3D visualization of remote sensing data that allows interactive navigation, interactive adjustment of visualization parameters, and interactive combination of multimodal data sets. Geometry information for terrain visualization is extracted from digital elevation models (DEMs). The terrain is textured using an interactively defined combination of data sets from different sensors, e.g. aerial photographs or SAR amplitude images. The framework can be used on a wide variety of systems, from desktop computers with modern graphics cards and visualization workstations with stereo displays to high resolution display walls and virtual reality systems driven by render clusters.

2. DATA MANAGEMENT

To handle the large amount of data generated by high-resolution remote sensing techniques, it is essential to use hierarchical data structures.

Since remote sensing data sets are often representable as 2D rectangles that span a subset of the World Geodetic System 1984 (WGS84) map (e.g. aerial photographs, DEMs, or SAR images), variants of quadtrees are commonly used for this task [3].

2.1. Quadtree Variant

In our framework, we use a quadtree variant with the following properties:

The single quad in the lowest level 0 of the tree spans the whole globe in WGS84 coordinates (latitude from $+90^\circ$ to -90° and longitude from -180° to $+180^\circ$). This ensures that all data sets can be managed from one combined hierarchy, so that it is not necessary to match and fit different hierarchies during visualization. Since the number of quads in both latitude and longitude direction is 2^l for a level l in a quadtree, each quad represents a rectangular area of the WGS84 map that is two times wider than high.

The quadtree is restricted, i.e. the levels of two neighboring quads differ by not more than one. A quadtree with this property can easily be transformed into a mesh consisting of isosceles right-angled triangles [3]. The quadtree levels are contiguous at their left and right borders, to allow seamless visualization at the $-180^\circ/+180^\circ$ longitude transition. Each quad carries an additional border of data around its represented area. This data is omitted in visualization, but allows data processing techniques to work on local neighborhoods without expensive data fetches from neighboring quads.

This quadtree variant is a generalization of the tiling pyramid used in [2].

This choice of data hierarchy results in non-optimal behaviour at the north and south pole, because one side of the quads at the poles will be reduced to a point when projecting WGS84 to cartesian coordinates. However, this drawback is outweighed by the benefits provided by working on rectangular areas in WGS84 coordinates.

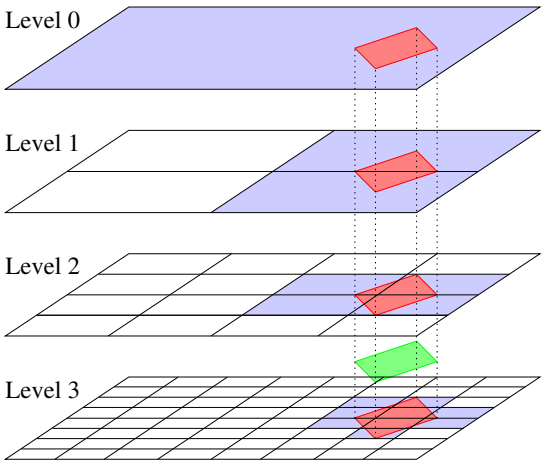

Fig. 1. Quadtree for a data set. The original data set is marked green. Quads that contain relevant data are marked blue.

Each quad in a quadtree has the same fixed size in pixels (or, more general, data samples) for its represented area and the border. Since each level halves the represented area in latitude and longitude direction, this means that each higher level doubles the data resolution.

With a fixed size of 512×256 pixels, quads in level 16 provide a ground coverage of around 1 m^2 per pixel, and quads in level 26 provide around 1 mm^2 per pixel (at the equator).

The resolution of a given data set will lie between two quadtree levels, as shown in Fig. 1. To build the quadtree, the original data set is first resampled to the next higher quadtree level, using a magnification factor between 1 and 2. The required quads of a lower level in the quadtree can be computed by combining quads from the next higher level. Areas of a quad which are not covered by the given data set are marked with a special value.

Some processing methods might need more information about a data set than what is provided by the currently processed quad. Therefore, the quadtree may be extended with global data set properties, e.g. the average amplitude for a SAR image, and per-quad properties, e.g. the minimum and maximum altitude for a DEM data set.

2.2. Data Storage and Cache Hierarchy

A data set quadtree is stored on disk in a directory hierarchy. The top level directory contains a file that provides information about that data set (type, area, global properties of the data, etc.) and, if applicable, a file that contains per-quad properties of the data set.

The quad data is then stored in subdirectories with names of the form $l/x/y$, where l is the quadtree level and x and y are the coordinates of the quad within this level. The data itself is stored in a file format that allows good lossless compression of the specific type of data.

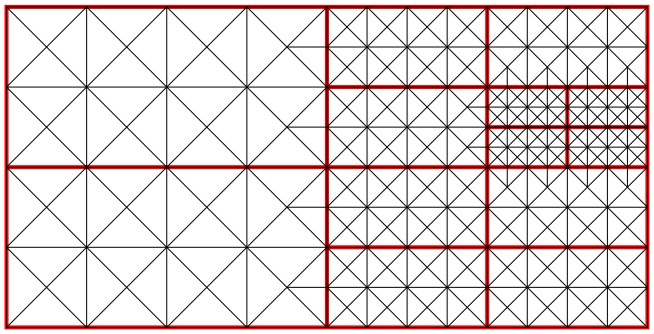

Fig. 2. Triangulation of the quadtree variant used in our framework.

This directory hierarchy allows easy access to a data set both from local storage and using common network protocols such as HTTP or FTP.

To allow interactive visualization of multiple large data sets, a caching hierarchy must be used. The first level of this cache hierarchy is the local disk, where data set quads are stored in the same directory structure as the original data set, but uncompressed. This cache level is not limited in size. The second cache level is a main memory cache with a fixed maximum size. Quads are replaced on a least recently used (LRU) basis. The third cache level is a GPU memory cache. It is similar to the main memory cache in that it has a fixed maximum size and replaces quads on an LRU basis, but it is typically both smaller and faster.

The transfer of a quad from the original data set to the first cache level and then to each higher cache level up to GPU memory has to be done asynchronously in separate threads, in order to provide fully interactive visualization at all times. The cost of this is that the visualization system must use approximations of quads that are not yet available in GPU memory, based on the nearest available match from lower quadtree levels, until all required data is in GPU memory. This typically only takes fractions of a second for data sets on local storage, but can take longer when transferring data over network links.

3. RENDERING

Only a relatively small number of quads needs to be displayed at one time. If a large part of the globe is visible, lower level quads with a low resolution will be displayed. If a small area is visible in detail, higher level quads with a high resolution will be displayed.

3.1. Rendering Quadtree

The subset of quads of the global hierarchy that needs to be rendered is represented by a rendering quadtree. The render-

ing quadtree is constructed from scratch for each new frame. This construction starts from quadtree level 0 which contains a single quad that represents the whole globe. For each quad, a bounding volume that is projected to screen space is used to determine whether the quad would occupy a larger area on screen than it provides data for, i.e. whether the on-screen area contains more pixels than the quad. If that is the case, the quad is split into four quads on the next higher quadtree level. This splitting process is done in a way keeps ensures that the levels of neighboring quads do not differ by more than one. The leaves (childless quads) of the resulting rendering quadtree are the quads that need to be rendered.

To avoid overestimating the screen space area occupied by a quad, it is important to know the minimum and maximum altitude value for each quad. These can be stored with the relevant altitude data sets as described above.

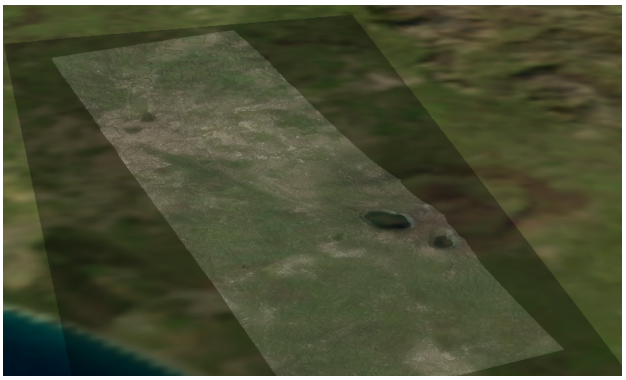

Fig. 3. The area around Rome, Italy, viewed in an overlay of a TerraSAR-X image and the NASA Blue Marble Next Generation data set for June 2004.

3.2. Data Processing

From the visualization point of view, there are two types of data sets: those that influence geometry, i.e. data sets that provide altitude information, and those that provide texturing information, i.e. aerial photographs, SAR images, multispectral images, or data from any other imaging technique.

For each leaf of the rendering quadtree, the subset of data sets that provide data for the represented area of that leaf is determined. The data from each data set is then transformed using a GPU-based processing chain that is fast enough to be used interactively. Interactive adjustment of processing parameters is especially useful for data sets that cannot be transformed to a single RGB image without loss of information. For SAR images, such a processing chain is described in [2]. Similar processing chains can be built for other modalities.

The result of this data processing step is a collection of quads with altitude information and a collection of quads with texture information, depending on the available data sets.

This information is then combined on the GPU into a single quad with altitude information and a single quad with texture information per leaf. This combination can be a simple weighted blending of data sets, but more sophisticated methods can also be implemented.

3.3. Creating a Mesh

When all required data is available for all leaves of the rendering quadtree, a single triangle mesh is computed from the leaves. This mesh is free of cracks and T-junctions, and all triangles are isosceles and right-angled in quad space. These properties are important to avoid degenerated triangles in terrain rendering.

There are many methods to produce such meshes from restricted quadtrees; two are summarized in [3]. In our approach, each leaf should produce a mesh with two times more triangles in longitude than in latitude direction, to account for the rectangular area covered by our quads. Therefore, we use the following method (see Fig. 2): Every leaf is divided into $2n \times n$ subquads. Every subquad is divided into four isosceles right-angled triangles. The border triangles are further divided into two triangles if the neighboring leaf has a higher level than the current leaf.

Since the core mesh is of the same form for each leaf and there are only 16 possibilities of border triangle setups (each of the four sides of the quad may or may not require subdivision), the necessary data can be precomputed and stored in graphics memory. For each leaf, the appropriate mesh can then be chosen and transformed to world coordinates.

All leaves are numbered sequentially, and the leaf number is stored as an attribute for each triangle. Using this information, following steps can work with properties of the leaf, such as quadtree level, coordinate, and neighboring leaves and their properties.

When working with cartesian coordinates on a global scale, the single precision floating point data type is problematic since it does not provide enough precision for subcentimeter or sub-millimeter accuracy of the terrain, resulting in spatial jitter and other artifacts. This problem can be solved by rendering the scene relative to the viewer, i.e. with viewer coordinates $(0,0,0)$. This makes sure that inaccuracy and loss of precision occur where it does not matter: far away from the viewer, where they do not result in a visible screen space error [4]. In our framework, we need to compute cartesian coordinates on the GPU and then subtract the viewer coordinates. These two steps must be performed using the double precision floating point data type available on current GPUs. The result can then again be stored using a single precision data type.

3.4. Texturing

Each triangle of the mesh stores its leaf number and its original quad space coordinates. This allows to access the quad

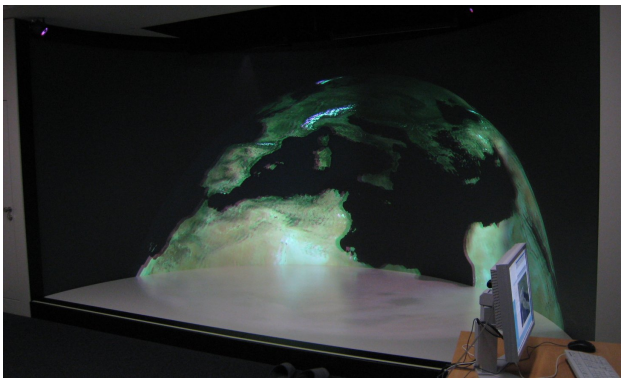

Fig. 4. The framework running in the Virtual Reality Lab at University of Siegen, Germany. The displayed data set is the NASA Blue Marble Next Generation data set for June 2004.

that provides the relevant texture information from the data processing step and to set the right texture coordinates. To allow seamless texturing of the leaves and avoid interpolation artifacts at leaf borders, it is necessary that the combined texture quad contains an additional border of at least one pixel.

3.5. Distributed Rendering

To allow distributed rendering, the visualization framework encapsulates the render state, which mainly consists of the viewer position and orientation and the list of active data sets and their processing parameters. The main application process manages the master render state and may distribute it over a network.

In a common desktop environment, there is only one render state, and only a single view of the scene is rendered. For high resolution display walls or virtual reality systems, multiple synchronized render states can exist for multiple GPUs (e.g. on visualization workstations) and/or on multiple hosts (e.g. the nodes of a render cluster), allowing different views of the scene to be rendered.

The construction of the rendering quadtree based on screen space metrics, and thus the decision which quads are processed and displayed, depends only on the render state and the current view frustum. Each process or thread with a render state can decide which parts of the data are needed to render its view of the scene, and only these parts need to be fetched and processed. The disk cache can either reside on a local storage system, or on a shared cluster file system, to reduce the number of data fetches from the (potentially remote and slow) data set server.

4. RESULTS

Our visualization framework implements the techniques described in the previous sections and is based on OpenGL. Since double precision is not yet available in traditional OpenGL shaders, we implemented the relevant steps in CUDA. The Equalizer Parallel Rendering Framework [5] is used for distributed rendering.

Fig. 3 shows a view that combines a high resolution SAR image and a lower resolution aerial image, demonstrating the ability to display data sets of different modality at the same time.

Fig. 4 shows our framework running in the virtual reality lab of the University of Siegen. There are six display areas: four on the curved screen, and two on the floor. Each of the stereo display areas is driven by one render node with two graphics cards, one for the left eye view and one for the right eye view. The six render nodes share a single disk cache directory on a shared file system.

5. CONCLUSION

We have presented a GPU-based framework for interactive visualization of remote sensing data. In contrast to existing systems, the presented framework allows interactive adjustment of visualization parameters for different modalities, and interactive combination of different data sets into a single view. Additionally, the framework can be used in distributed visualization systems with multiple GPUs and/or render nodes, such as display walls or virtual reality installations.

Acknowledgements

This project is partially funded by grant KO-2960-3/1 from the German Research Foundation (DFG). The NASA Blue Marble Next Generation data sets were produced by NASA Earth Observatory (NASA Goddard Space Flight Center). TerraSAR-X data sets © Infoterra GmbH.

6. REFERENCES

[1] M. Lambers, H. Nies, and A. Kolb, "Interactive Dynamic Range Reduction for SAR Images," *Geoscience and Remote Sensing Letters*, vol. 5, no. 3, pp. 507–511, 2008.

[2] M. Lambers, A. Kolb, H. Nies, and M. Kalkuhl, "GPU-based framework for interactive visualization of SAR data," in *Proc. Int. IEEE Geoscience and Remote Sensing Symposium (IGARSS)*, July 2007, pp. 4076–4079.

[3] R. Pajarola and E. Gobbetti, "Survey of semi-regular multiresolution models for interactive terrain rendering," *Vis. Comput.*, vol. 23, no. 8, pp. 583–605, 2007.

[4] C. Thorne, "Using a floating origin to improve fidelity and performance of large, distributed virtual worlds," in *Proc. Int. Conf. on Cyberworlds*, Nov. 2005, pp. 263–270.

[5] S. Eilemann, "The Equalizer parallel rendering framework," `http://www.equalizergraphics.com/`.