GPU-based Framework for Interactive Visualization of SAR Data

Martin Lambers*, Andreas Kolb*, Holger Nies‡, Marc Kalkuhl§
*Institute for Vision and Graphics, University of Siegen
Email: martin.lambers@uni-siegen.de
Email: andreas.kolb@uni-siegen.de
‡Center for Sensorsystems (ZESS), University of Siegen
Email: nies@zess.uni-siegen.de
§Department of Simulation, University of Siegen
Email: marc.kalkuhl@uni-siegen.de

***Abstract*—Synthetic Aperture Radar data presents specific problems for interactive visualization. The high amount of multiplicative speckle noise has to be reduced. The high dynamic range of the amplitude data must be mapped to the lower dynamic range of display devices in a way that makes image features appropriately visible. In addition to interactive navigation in the data, it is desirable to allow interactive selection of despeckling and dynamic range reduction methods and adjustment of their parameters.**

Graphics processing units (GPUs) can be seen as ubiquitous parallel coprocessors with extreme computational power. In this paper, we propose a GPU-based framework for interactive visualization of SAR data. Data management techniques are used to make full use of the GPU. We reworked well-known despeckling and dynamic range reduction techniques for the GPU programming model and implemented them in our framework. Both navigation in large data sets and adjustment of processing parameters are fully interactive.

I. Introduction

Major processing steps of the visualization of SAR data include despeckling and dynamic range reduction.

The goal of despeckling is to reduce the multiplicative speckle noise that affects SAR images. This noise can hinder data analysis tasks, especially segmentation and classification. Common despeckling methods include convolution filters (Mean, Gauss), rank operators (Median), methods based on local statistics (Lee, Kuan, Frost, Gamma MAP), filters with adaptable masks (Oddy), and wavelet based methods (Soft Thresholding). Reviews and comparisons of these methods are available in [1], [2], [3].

The purpose of dynamic range reduction is to map the high dynamic range[1] of SAR amplitude data to the lower dynamic range of display devices. A direct linear mapping of amplitude values to gray values is usually not feasible. Often, a logarithmic mapping is used to preserve image features, but other methods are also possible.

In addition to these processing steps, interactive visualization must provide interactive navigation through large data sets. The user must be able to choose and adjust a *region of interest* (ROI) of the image and the zoom level in which it is displayed. Furthermore, it is desirable to allow interactive selection of despeckling and dynamic range reduction methods and adjustment of their parameters. This significantly reduces the time required to perform visual image analysis, especially in the context of new techniques such as bistatic SAR imaging. Here, a fixed set of standard methods is not yet established and a certain amount of testing and experimenting is necessary.

These requirements result in a large demand for computational power. Our framework uses the graphics processing unit (GPU) of modern programmable PC graphics hardware to meet this demand.

Current GPUs have access to fast dedicated graphics memory. They can store 2D image data with up to four 32bit floating point components per pixel. The hardware is optimized for fast parallel processing of such data with microprograms. High level programming languages such as the OpenGL Shading Language (GLSL) are available. These capabilities open a lot of possibilities to use GPUs for scientific and general purpose programming. See [4] for an overview.

The GPU is a parallel stream processor [5]. The corresponding programming model is highly optimized for hardware accelerated parallel data processing, but it is very restricted when compared to the general purpose CPU programming model. In the stream programming model, an action is defined and activated via a microprogram, and then applied to a data stream. Input and output data sets are strictly separated. While it is possible to read data elements from different positions in the input data set to compute a predefined element of the output data set (gathering), it is rather difficult to select a specific position of the output stream (scattering), e.g. with regard to a computational result. To adapt existing algorithms to this model, it is necessary to rework them. Their workflow must fit the data stream model, and their calculations must avoid or circumvent the limitations of the GPU. A discussion of the stream programming model and its specific restrictions can be found in [5].

In the following section, we present data management techniques for large SAR images. This basis allows us to make full use of the GPU's potential to achieve the processing speed

[1]The dynamic range is the defined as the ratio of the largest possible value and the smallest possible non-zero value in an image.

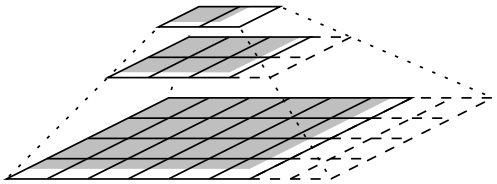

Fig. 1. A tiling pyramid for the SAR raw image represented by the gray rectangle. Each lower level has four times as many tiles as the next higher level, but some of them may be empty (dashed).

that is necessary for interactive visualization and parameter adoption. We then adapt despeckling and dynamic range reduction algorithms to the GPU programming model and show how to implement them in our framework.

II. Framework

Because both the available memory and computing power is limited, it is not feasible to repeat the processing steps for the whole SAR raw image each time a parameter changes. Data management techniques must ensure that only the currently visible part of the image is processed.

A. Data Management

If only a small part of the image is currently displayed, then only that part needs to be processed. If the whole image is displayed, then it will be displayed in a low zoom level because of the fixed display resolution, so that it is sufficient to process a downscaled version of the image. Following this idea, the SAR raw image is divided into tiles of fixed size at different resolution levels in a non-interactive preprocessing step. The result is a tiling pyramid as shown in Fig. 1. Each higher pyramid level halves the resolution of the image.

Depending on the current ROI and the zoom level in which it needs to be displayed, a subset of the tiles is chosen. This subset of tiles is loaded into graphics memory if necessary, processed according to the currently selected despeckling and dynamic range reduction methods and their parameters, and then displayed on the screen. This process is continuously repeated to acknowledge user interactions.

Fig. 2 gives an overview of the data flow in our framework. While loading the SAR data and building the tiling pyramid, it is possible to perform some (potentially time-consuming) preprocessing to compute e.g. statistical properties that may later be needed by despeckling or dynamic range reduction methods. Interactive navigation (ROI adjusting) and interactive adjustment of processing methods and parameters is done using appropriate GUI elements. All the computationally intensive work is done on the GPU: the tiles best representing the current ROI and resolution are processed with the currently selected despeckling and dynamic range reduction methods and then visualized. This is performed fast enough to provide immediate feedback to ROI or parameter adjustments.

To permit working with very large SAR raw images, the tiling pyramid can be stored on disk and the tiles can be cached in main memory. Only the currently needed subset of tiles must be available in the dedicated graphics memory.

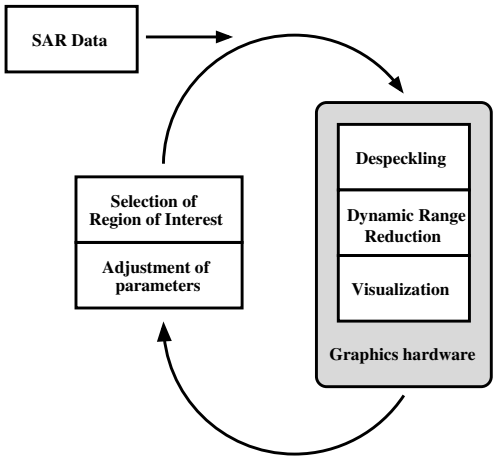

Fig. 2. Data flow in the framework.

Additional data can be prefetched if assumptions about the user's next actions are made. That way, data tiles can be readily available when they are needed. The necessary data transfers, both from disk to main memory and from main memory to graphics memory, can be done asynchronously without the need to halt the application.

In each tile of the tiling pyramid, we store additional information from its surrounding region, as shown in Fig. 3. This ensures that local neighborhood information is always available. Thus, local operations like despeckling and dynamic range reduction can always be performed without expensive data fetching from neighboring tiles.

Tiles are stored as *textures* in graphics memory. Floating point textures allow to store up to four 32bit floating point values of information per pixel. This capacity can be used to store phase information in addition to the amplitude information, and to temporarily store intermediate results of the various processing steps.

The processing of a tile is a sequence of distinct processing steps. Since input and output data must be strictly separated on the GPU, two temporary tiles are needed: one for input, and one output. After each processing step, the role of the temporary tiles is swapped: the output of the last processing step becomes the input of the next processing step.

The complete processing chain is shown in Fig. 4. The processing steps are implemented using the render-to-texture functionality of the GPU: the source texture is rendered 1:1 into the destination texture. A microprogram implementing the current processing step computes the value of each output pixel with 32bit floating point precision.

The number of steps needed for despeckling and dynamic range reduction depends on the chosen methods. Simple methods may require only one step, while more complex methods require more steps.

B. Despeckling

Despeckling filters fall into several categories: convolution filters, e.g. Mean, Gauss, ranking operators, e.g. Median, methods based on local statistics, e.g. Lee, Kuan, Frost, Gamma

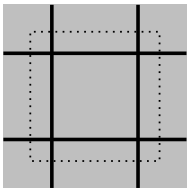

Fig. 3. A tile (dotted line) with an overlap area that includes information from neighboring tiles.

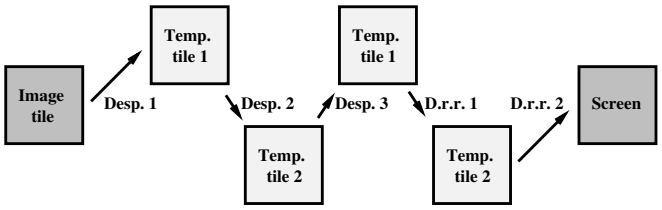

Fig. 4. A processing chain that prepares a tile for display using three despeckling (desp.) and two dynamic range reduction (d.r.r.) steps.

MAP, filters with adaptable masks, e.g. Oddy, and wavelet based methods, e.g. Soft Thresholding. The implementation of methods from each of these classes is described in the next paragraphs.

Simple convolution filters like the Mean and Gauss filters are easily implementable in the framework. A 3×3 Mean filter, for example, can be implemented using one processing step. The microprogram gathers the information from the 9 pixels of the local neighborhood and computes the filtered output pixel. Mean and Gauss filters with larger masks can be implemented in a separable manner to increase performance. In this case, each filter needs two processing steps: one for the horizontal mask, and one for the vertical mask.

Methods based on local statistics interpret a local neighborhood (e.g. 5×5 or 7×7) as a sampling distribution, and compute sample mean and variance from it. The value of the output pixel is then computed based on the underlying assumptions about statistical properties of SAR data. The sample mean and variance can be computed in two steps in a separable manner. The computation of the output pixel can usually be included into the second step.

Some methods based on local statistics need additional values that are computed globally from the SAR raw image. For example, the method of Xiao et.al. [6] needs to compute global minimum and maximum deviation values. This cannot be done efficiently in the GPU microprograms due to the tile-based data management. Generally, there are three methods to solve this kind of problem: 1) Compute the values in the preprocessing step, while building the tiling pyramid, and pass them to the GPU microprograms as parameters. 2) Estimate the values based on a local neighborhood. This can be done within the GPU microprograms. 3) Treat the values as additional parameters and let the user adjust them. Although method 1 is the method of choice in order to achieve results that are comparable to the original algorithm, we found that method 3 also works reasonably well. Method 2 will lead to unintended differences in filter behaviour in different regions of the image.

Ranking operators like the Median filter are based on sorting. This is still a hard problem on GPUs due to the constraints of the programming model. We implemented the separable approximation of the median filter. For each rectangular neighborhood, the median is approximated by first computing the median for each row, and then computing the median of these row medians. This can be done using two GPU microprograms. Computing the exact median would either limit the filter to small mask sizes, or introduce the need for a sophisticated texture sorting algorithm [4].

In addition to the above, we also implemented a filter based on adaptable masks (Oddy [1]) and wavelet based Soft Thresholding [3], to show the implementability of these classes of methods in our framework.

C. Dynamic Range Reduction

Dynamic range reduction is used to map the high dynamic range of the SAR amplitude data to the lower dynamic range of the display device. Often, only a part of the dynamic range of the amplitude values is mapped to gray levels for this purpose, e.g. high peaks in the amplitude data are clamped to white.

Multiple possibilities exist for this mapping. Linear, logarithmic, or gamma-correction like are some that are commonly used. All of these methods transform one input value into one output value using a globally defined function. They can therefore easily be implemented using a single GPU microprogram.

More complex methods might take the local neighborhood of pixels into account and adapt themselves accordingly. This has the advantage that the visibility of important image details can be improved in certain areas. The disadvantage is that the resulting intensities in different regions are not directly comparable anymore.

As an example for such a local method, we adapted the Durand tone mapping operator [7] to work on SAR amplitude values instead of optical intensity values and integrated it in our framework as a single-step dynamic range compression operator. Analogous to the Xiao despeckling method, the Durand tone mapping operator needs to compute a global maximum value from the raw image. We replaced this value with a user-adjustable parameter.

In general, the task of dynamic range reduction of SAR data is related to the task of tone mapping of optical images. Tone mapping operators approximate the appearance of high dynamic range optical images on lower dynamic range displays. Many global and local techniques exist [8]. Examining which of these techniques are adaptable to SAR images is subject of future work.

III. Results

Fig. 5 shows a screenshot of our framework. The user can interactively choose the ROI and zoom level in the upper left part of the application window. Despeckling and dynamic range reduction methods can be chosen and interactively

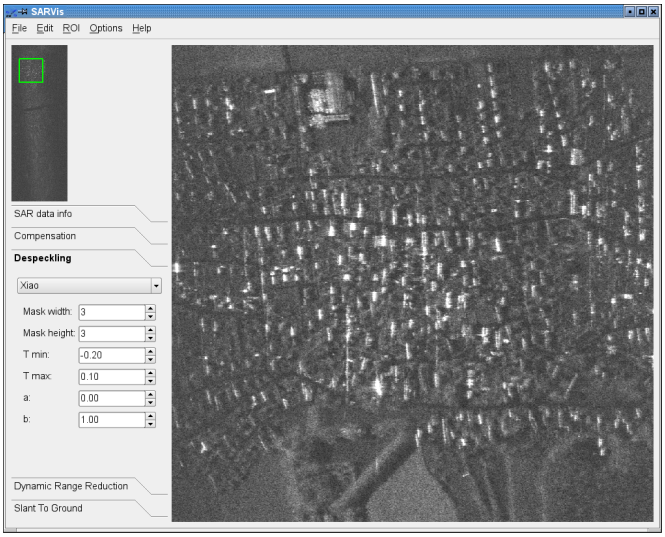

Fig. 5. A screenshot of our framework (raw data courtesy of FGAN).

adjusted using the toolbar in the lower left part. The ROI is displayed in the large view area. The effects of different processing methods and parameters are shown in Fig. 6.

An export function allows to save a processed version of the whole image at its original resolution. All of the necessary processing is then performed on the graphics hardware using the existing framework, which is typically much faster than CPU based processing.

On a system with an AMD X2 4200+ processor, 4GB RAM, and an NVIDIA GeForce 7900 GTX graphics card, building and storing the tiling pyramid for a SAR raw image with 8192×32768 pixels requires about 2 minutes. In this example, the tiles of the pyramid have 256×256 pixels. The overlap area is 9 pixels on each side, to allow mask sizes of up to 19×19 pixels for despeckling and dynamic range reduction methods that work on local neighborhoods. This leaves 238×238 pixels of image coverage per tile. Pyramid level 0, which stores the SAR raw image in its original resolution, therefore consists of $35\times138=4830$ tiles. The pyramid has 9 levels, storing a total of 6520 tiles. It needs to be computed only once and can be stored on hard disk for later reuse. Once the tiling pyramid is built, interactive visualization performance is maintained even for processing steps using large mask sizes.

Tab. I gives some FPS (frames per second) measurements for different sizes of the view area and different combinations of despeckling and dynamic range reduction methods. A minimum and a maximum FPS value is given for each combination. The difference between them is due to the different number of tiles that are needed to fill the view area, depending on the current resolution. The FPS rate is limited by the processing power of the GPU. Large view areas combined with demanding despeckling and dynamic range reduction techniques may lead to low frame rates. Additionally, the amount of main memory is important for the pyramid building step and can also help visualization performance by allowing more tiles to be (pre-)cached.

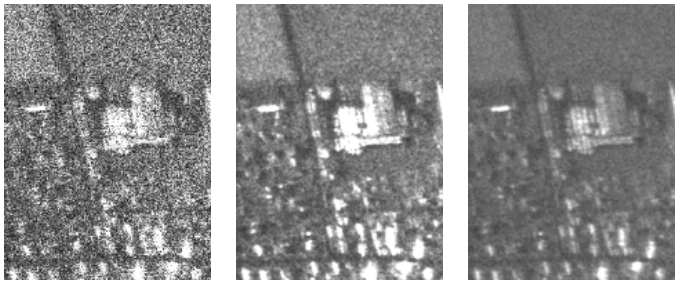

Fig. 6. The effects of different despeckling and dynamic range reduction parameters (raw data courtesy of FGAN). Left: no despeckling, logarithmic dynamic range reduction. Middle: with Xiao despeckling. Right: with gamma-correction like dynamic range reduction.

TABLE I
FRAMES PER SECOND FOR DIFFERENT SIZES OF THE VIEW AREA AND DIFFERENT COMBINATIONS OF DESPECKLING AND DYNAMIC RANGE REDUCTION (D.R.R.) METHODS.

	730×730	1130×920
No despeckling, linear d.r.r.	172 – 370	87 – 238
5×5 Xiao despeckling, logarithmic d.r.r.	42 – 99	24 – 54
7×7 Lee despeckling, 3×3 Durand d.r.r.	21 – 46	11 – 39

IV. Conclusion

In this paper, we have shown that interactive visualization of large SAR images, including interactive parameter adjustment, is made possible by exploiting the capabilities of modern GPUs. Efficient data management is necessary to make full use of the GPU's resources. Important algorithm classes have been reworked to fit the GPU programming model. The resulting system allows for fast, interactive analysis of SAR data.

Acknowledgements

This project is partially funded by grant KO-2960-3/1 from the German Research Foundation (DFG).

References

[1] L. Gagnon and A. Jouan, "Speckle filtering of SAR images – a comparative study between complex-wavelet-based and standard filters," in *Proc. SPIE Vol. 3169*, 1997, pp. 80–91.

[2] W. Hagg and M. Sties, "The EPOS speckle filter: a comparison with some well-known speckle reduction techniques," in *Proc. 18th Congress of the International Society for Photogrammetry and Remote Sensing*, 1996, pp. 135–140.

[3] E. Hervet, R. Fjørtfort, P. Marthon, and A. Lopès, "Comparison of wavelet-based and statistical speckle filters," in *EUROPTO Conference on SAR Analysis, Modeling, and Techniques*, 1998, pp. 43–54.

[4] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, and T. J. Purcell, "A survey of general-purpose computation on graphics hardware," *Computer Graphics Forum*, vol. 26, no. 1, pp. 80–113, 2007.

[5] R. Strzodka, M. Doggett, and A. Kolb, "Scientific computation for simulations on programmable graphics hardware," *Simulation Practice & Theory*, vol. 13, no. 8, pp. 667–680, 2005.

[6] J. Xiao, J. Li, and A. Moody, "A detail-preserving and flexible adaptive filter for speckle suppression in SAR imagery," *Int. J. Remote Sensing*, vol. 24, no. 12, pp. 2451–2465, 2003.

[7] F. Durand and J. Dorsey, "Fast bilateral filtering for the display of high-dynamic-range images," in *ACM Proc. SIGGRAPH*, 2002, pp. 257–266.

[8] E. Reinhard, G. Ward, S. Pattanaik, and P. Debevec, *High Dynamic Range Imaging: Acquisition, Display and Image-based Lighting*. Morgan Kaufmann, 2005.