ENGINEERING THE IDEAL GIGAPIXEL IMAGE VIEWER

Dominik Perpeet

Fraunhofer IOSB

Jan Wassenberg

Fraunhofer IOSB

Abstract

Despite improvements in automatic processing, analysts are still faced with the task of evaluating gigapixel-scale mosaics or images acquired by telescopes such as Pan-STARRS[1]. Displaying such images in 'ideal' form is a major challenge even today, and the amount of data will only increase as sensor resolutions improve.

In our opinion, the ideal viewer has several key characteristics. Lossless display – down to individual pixels – ensures all information can be extracted from the image. Support for all relevant pixel formats (integer or floating point) allows displaying data from different sensors. Smooth zooming and panning in the high-resolution data enables rapid screening and navigation in the image. High responsiveness to input commands avoids frustrating delays. Instantaneous image enhancement, e.g. contrast adjustment and image channel selection, helps with analysis tasks. Modest system requirements allow viewing on regular workstation computers or even laptops.

To the best of our knowledge, no such software product is currently available. Meeting these goals requires addressing certain realities of current computer architectures. GPU hardware accelerates rendering and allows smooth zooming without high CPU load. Programmable GPU shaders enable instant channel selection and contrast adjustment without any perceptible slowdown or changes to the input data. Relatively low disk transfer speeds suggest the use of compression to decrease the amount of data to transfer. Asynchronous I/O allows decompressing while waiting for previous I/O operations to complete. The slow seek times of magnetic disks motivate optimizing the order of the data on disk. Vectorization and parallelization allow significant increases in computational capacity. Limited memory requires streaming and caching of image regions.

We develop a viewer that takes the above issues into account. Its awareness of the computer architecture enables previously unattainable features such as smooth zooming and image enhancement within high-resolution data. We describe our implementation, disclosing its novel file format and lossless image codec whose decompression is faster than copying the raw data in memory. Both provide crucial performance boosts compared to conventional approaches. Usability tests demonstrate the suitability of our viewer for rapid analysis of large SAR datasets, multispectral satellite imagery and mosaics.

[1]Panoramic Survey Telescope And Rapid Response System, *Institute for Astronomy, University of Hawai'i, USA*

1 INTRODUCTION

Today, human analysts still play an important and irreplaceable part in the evaluation of images acquired by a variety of sensor technologies such as telescopes, laser scanners and radar systems. Despite improvements in automatic processing, increasing sensor resolutions (Fig. 1) and the availability of large storage media have resulted in the requirement of routinely evaluating gigapixel-scale image data. In our opinion, advances in hardware and software merit a reevaluation of image viewing concepts.

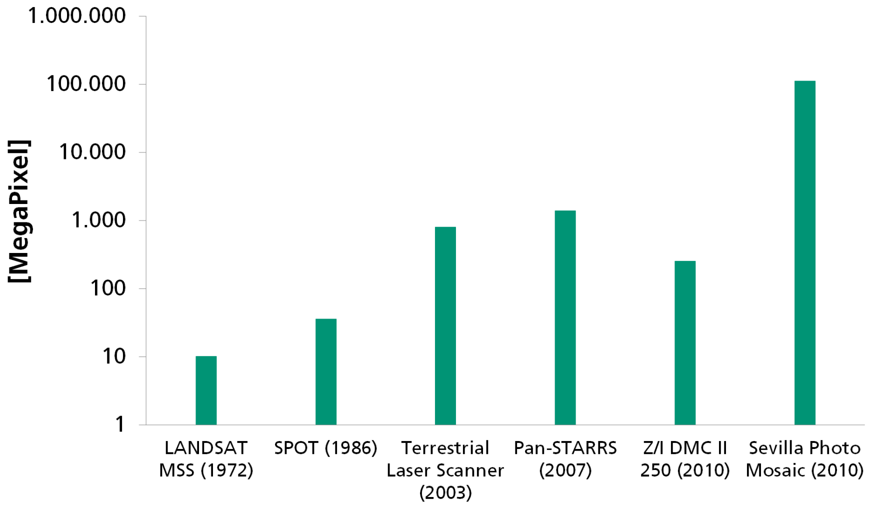

Figure 1: Increasing image sizes.

2 IMAGE VIEWER DESIGN CRITERIA

In the following, we focus on professional image analysts for the determination of image viewer design criteria. Key characteristics of an ideal viewer fall into the following categories:

- Image fidelity
- Supported image formats
- User interaction
- Image manipulation
- Hardware requirements

One option to increase the amount of image data that can be presented to a user is to use compression. An overview of common methods is presented in [2]. Whereas many modern compression techniques produce images that are visually *nearly equivalent* to the original, they are ill-suited for professional image analysis for two reasons. First, working with high pixel-resolution data stems from the need for more detail – otherwise an image could be downscaled before inspection. Lossy compression would not necessarily preserve the required details. Second, for lossy compression to be acceptable, it would have to be tailored to the different types of image data in order to preserve the features required for successful analysis. To allow the analyst to extract all possible information, the viewer should ensure fidelity to the original data by avoiding any loss of information.

A variety of data types from varying sources must be displayed. Although it is theoretically possible to convert between different data representations, it is preferable to support the native types of sensor systems. Some deliver 8-bit precision integer data per channel, others 16-bit integer or even 32-bit floating point values. Additionally there are varying numbers of channels, ranging from one channel grayscale representations to hyperspectral images with well over 100 channels. For an image viewer to be readily usable, it should support multiple data types and also be able to interface with existing file formats.

In order for an analyst to fully focus on the actual image data, interaction with the viewer should be intuitive and smooth. Two important factors with a negative impact on user interaction are disruptions, as shown in [1], and waiting times. User disruptions are avoided by only responding to user input – an example of an interruption is a popup during normal use that informs the user of additional program functionality. Waiting times are usually the result of computationally intensive operations that exclude further user interaction for the duration. To achieve a continuously smooth user interaction, core features should be optimized in this regard: program startup, image loading, panning, and zooming.

A modern image viewer is expected to perform basic image manipulation to ease image evaluation. Of these, two are essential for analyzing data from various sensors. Because many sensors deliver data with more than three channels, a user needs to be able to select which channels to view and map them to the visible RGB channels. Additionally, sensors may not utilize the full range of their respective data types, e.g. images with 16-bit precision that only contain values in the lower 8 bits. To better visualize these images, the viewer should be able to normalize the images, e.g. via histogram equalization. These image operations should be performed without perceptible delays, writing additional data to disk or processing the entire image when the option is selected.

Furthermore, due to continued performance increases in computing, standard workstation or even laptop computers should be able to view gigapixel-scale image data. To this end, available hardware features must be utilized: GPU, vectorization (SIMD: Single Instruction, Multiple Data) [4], and multicore CPUs [5].

3 Implementation

In the following, we will focus on the implementation of an image viewer that meets all of the design criteria. We will first discuss overall design decisions and then provide a detailed description of the individual viewer components (shown in Fig. 2).

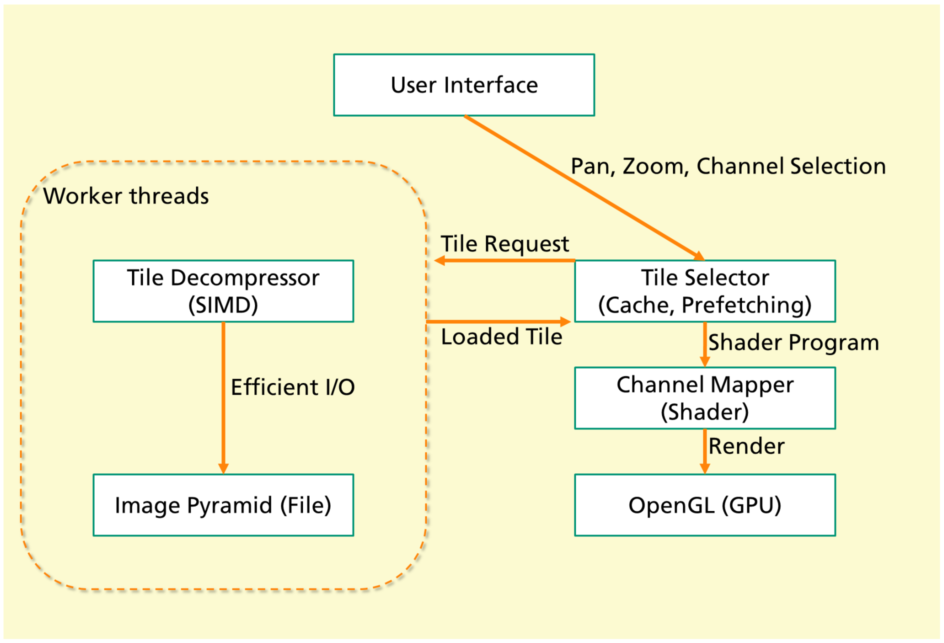

Figure 2: Components of the Image Viewer.

The stated requirement of running on a laptop computer can only be met with full hardware utilization. As a result, the implementation should use the GPU for display purposes, perform efficient disk I/O, and make use of multiple CPU cores. For compatibility, the code should use multi-platform libraries and features whenever possible. To this end, we decided to use OpenGL for communicating with the GPU, Qt as a multi-platform library for the user interface, and OpenMP for parallelization. Because gigapixel images greatly exceed computer display capabilities, pre-calculated image pyramids – successively downscaled versions of the original image – must be used to reduce the amount of data to load from disk. By selecting the appropriate level from the pyramid, the viewer need only load data corresponding to the viewport size, as opposed to the full image size. The pyramid levels are also tiled, enabling efficient access to sections of the image.

Unfortunately, we are not aware of an existing file format that ensures such a data layout. For example, TIFF files are organized in lines *or* tiles, and NITF also allows very flexible (and inefficient) pixel formats. We therefore introduce an intermediate representation of the data that guarantees an efficient layout, thus enabling smooth zooming and navigation within the image. Let us emphasize that it is not intended to replace existing formats. We convert such images into a separate file, which can co-exist alongside the original data. Because large images require significant time to generate, the conversion can be considered an off-line process. Significant effort can be expended in optimizing the image to ensure subsequent viewing is fast. To that end, we have introduced a lossless asymmetric compression scheme designed for extremely efficient decompression. The image is split into tiles to ensure good spatial locality, and these tiles are then compressed individually to enable fast random accesses. Our novel codec includes a SIMD-enabled predictor and entropy coder, which enables a decompression throughput in excess of 3 GB/s on a single CPU core. A compression ratio of approximately 0.5 is reached on 16-bit, four channel satellite imagery. [6]

Our LVT (Lossless Virtual Texture) file format allows storing these tiles in an application-defined order, which can reduce disk seeks. We store a mapping of coordinates to a tile ID, and their offsets within the file. Such metadata (including image statistics, geographic location or any other application-defined information) is stored in individual sections of the file. These are referenced by a section directory indicating their location in the file. Full details of the format are disclosed in [7].

The viewer accesses the intermediate representation through a library, which includes support for efficient I/O. The DMA (Direct Memory Access) capability of PCs allows the application to continue with other tasks while I/O is in progress. We decompress the individual portions of a tile while waiting for the subsequent I/Os to complete. This requires asynchronous I/O, for which the POSIX standard has defined a portable interface. We provide an extremely efficient implementation on Windows that avoids context switches in the kernel and exceeds the write throughputs of the ATTO and CrystalDisk benchmarks by 4% [7].

Similar to the virtual texture concept described in [3], we use OpenGL textures as a cache for tiles that were loaded from disk. In contrast to the described virtual texture techniques, we use the CPU for our tile selector: Depending on the current zoom level, pan, rotation and viewport size, missing tiles are requested from the LVT file. They are stored in OpenGL textures of a fixed size (a multiple of the tile size). The size of the cache textures is determined by the capabilities of the graphics adapter. Currently supported data (pixel) formats per channel are: 8-bit integer, 16-bit signed/unsigned integer, as well as 32-bit floating point values. Optionally, instead of only caching previously used tiles, prefetching tiles when idle may further increase performance.

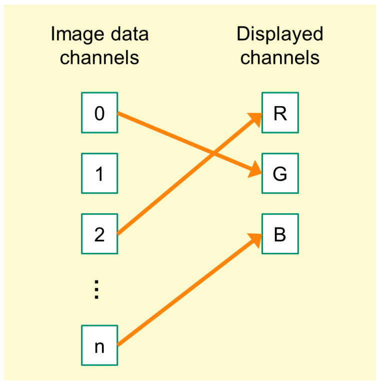

Figure 3: A display channel (RGB) can be set to any image channel.

At the end of each render pass, the GPU vertex and pixel shaders perform simple histogram equalization and map available data channels to display channels (RGB), as shown in Fig. 3. Display channels can also be individually disabled. The histogram equalization consists of two possible histogram stretches: either all channels are converted using the same factor, or each displayed channel is adapted individually. This feature is especially helpful when viewing data from images that do not utilize the entire value range.

Because all data is loaded from disk asynchronously and decoded in a dedicated I/O thread, user interaction remains smooth at all times. The OpenGL textures can only be accessed by one thread at a time. To avoid a reduction in performance when loading new tiles, there is a time limit on uploading new tiles to the texture cache during each render pass. The clean user interface with minimal potential for distracting the analyst provides support for seamless zooming to the mouse pointer, accurate panning, and easily accessible menu entries for channel selection and histogram equalization (Fig. 4).

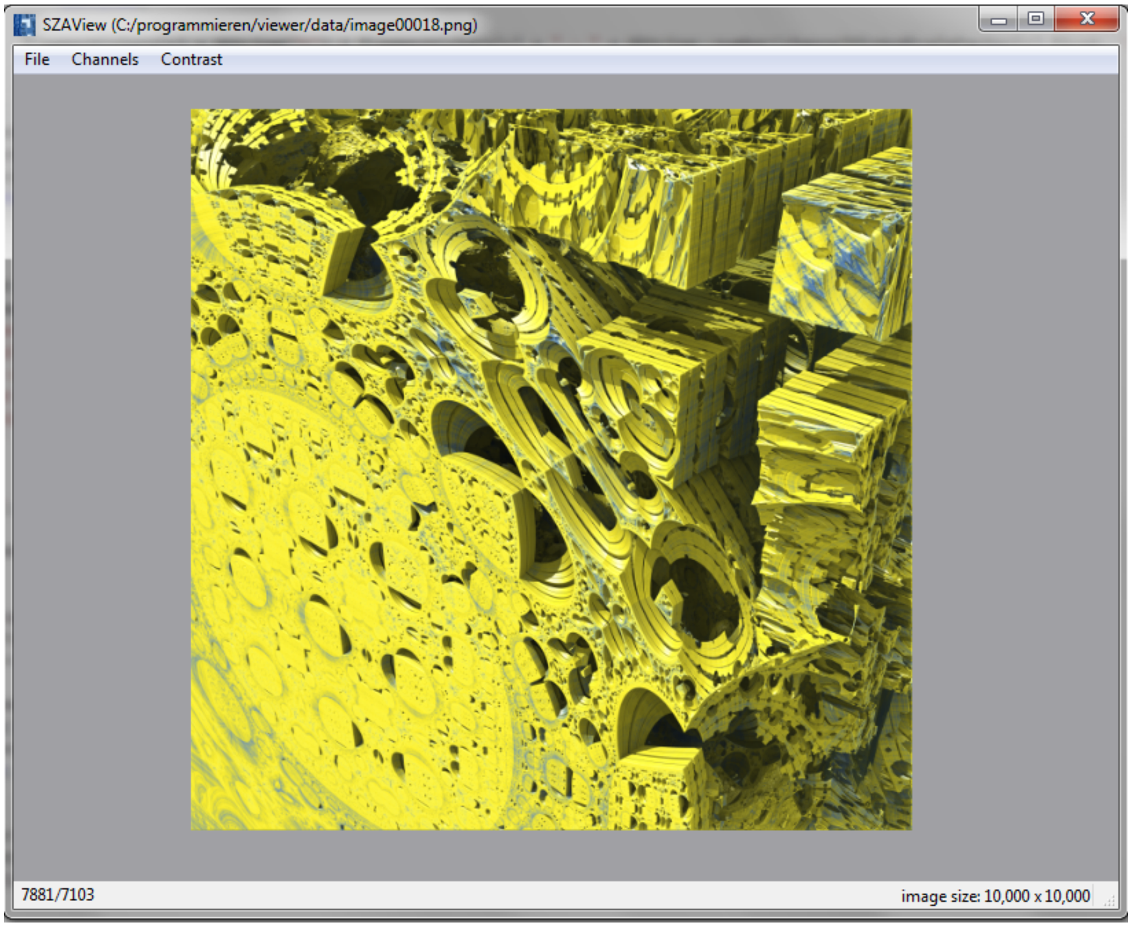
SZAView (C:/programmieren/viewer/data/image00018.png)
File
Channels
Contrast
7881/7103
image size: 10,000 x 10,000

Figure 4: Screenshot of the viewer and its straightforward user interface.

4 Conclusion

Our efficient and responsive image viewer demonstrates that an awareness of low-level and computer architecture issues brings into reach previously unattainable goals such as lossless compression and seamless zoom in gigapixel-scale imagery. The viewer ensures fidelity to the original pixels, is able to import many existing image formats via the GDAL library, supports the most common pixel formats (8-bit integer, 16-bit signed and unsigned integer, 32-

bit floating point[2]) with an arbitrary number of channels, allows smooth user interaction and enables basic real-time image manipulation in the form of channel selection and histogram stretch. We have validated the performance on SAR, multispectral, mosaic and custom rendered images with sizes exceeding 2 GigaPixels on a laptop computer.

The intermediate file format and viewer implementation are optimized for parallel processing, especially on tiles. Therefore, a possible future extension would be to run efficient image processing algorithms such as screening for point-like objects [8] directly on the intermediate format. Results could quickly be visualized, or even be calculated as needed for visible tiles only, if the algorithm supports it. Future work could also include a more extensive set of shader operations, such as advanced histogram equalization methods.

References

[1] B.P. Bailey, J.A. Konstan, and J.V. Carlis. Measuring the effects of interruptions on task performance in the user interface. In *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, volume 2, pages 757 –762 vol.2, 2000. Available from: `http://interruptions.net/literature/Bailey-SMC00.pdf`.

[2] P. Krause. Texture compression, November 2007. Available from: `http://www.colecovision.eu/graphics/texture_compression.pdf`.

[3] M. Mittring and Crytek GmbH. Advanced virtual texture topics. In *ACM SIGGRAPH 2008 classes*, SIGGRAPH '08, pages 23–51, New York, NY, USA, 2008. ACM. Available from: `http://developer.amd.com/documentation/presentations/legacy/Chapter02-Mittring-Advanced_Virtual_Texture_Topics.pdf`.

[4] J. Parri, D. Shapiro, M. Bolic, and V. Groza. Returning control to the programmer: SIMD intrinsics for virtual machines. *Commun. ACM*, 54(4):38–43, 2011. Available from: `http://cacm.acm.org/magazines/2011/4/106583-returning-control-to-the-programmer-simd-intrinsics-for-virtual-machines/fulltext`.

[5] H. Sutter. The free lunch is over: A fundamental turn toward concurrency. *Dr. Dobb's Journal*, March 2005. Available from: `http://www.ddj.com/web-development/184405990`.

[6] J. Wassenberg. Lossless asymmetric single instruction multiple data codec. *Software: Practice and Experience*. Accepted for publication.

[7] J. Wassenberg. *Efficient Algorithms for Large-Scale Image Analysis*. PhD thesis, KIT, 2011.

[8] J. Wassenberg, W. Middelmann, and P. Sanders. Highly efficient screening for point-like targets via concentric shells. In *Advanced Maui Optical and Space Surveillance Technologies Conference*, September 2010.

[2]Not all current graphics adapters support 32-bit pixel values.