UCRL-TR-236965

LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

Extension of 4-8 Texture Hierarchies to Large Video Processing and Visualization

J. G. Senecal, A. E. Wegner

December 4, 2007

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

Extension of 4-8 Texture Hierarchies to Large Video Processing and Visualization

Joshua Senecal and Aaron Wegner

November 30, 2007

The purpose of this Techbase was to reduce to practice the tiled 4-8 texture hierarchy for the display of video imagery (i.e. sequences of frames). The immediate intent was to demonstrate its use in the analysis and display of sensor imagery. As sensors are increasing in resolution the physical amount of imagery that needs to be displayed can quickly overwhelm most display systems. For example, a sensor with a horizontal resolution of over 8000 pixels would generate an image over 10 feet wide on a standard 72 DPI display. Breaking an image into tiles, and then decomposing each tile into a multiresolution hierarchy, allows a user (or software) to efficiently select and display only those parts of the image that are of interest to the user.

The originator of the idea of 4-8 Texture Hierarchies was Dr. Mark Duchaineau, and we consulted with him in much of our work. We also consulted with Dan Knight, from SequoiaTek Corp., who is a contractor responsible for implementing the viewers for our applications.

Most of the code for actual 4-8 Texture Hierarchy generation already existed; a large focus of the Techbase was to determine how to best use what was available for video imagery. The majority of progress was made in specifying and implementing the software framework, which turned out to be rather involved. This framework is to support the creation, storage, and display of images, both tiled and untiled. A first albeit incomplete version was successfully tested in the field in August 2007. The framework structures the process of collecting and processing images conceptually as a pipeline, where work is passed along and a different operation is performed at each stage.

In practice, the pipeline is implemented by a group of processes (not threads), or “workers”, each responsible for a specific type of operation. Associated with these workers is a pool of memory (cache). As each process finishes its work, it places the results into the cache and sends a message to the process responsible for the next operation, including in the message an identifier allowing the recipient to locate the data to be worked on. Conceptually this is akin to transferring mail from person to person by placing it into and removing it from a group of mail slots, rather than handing it to the next person directly.

So for example, to capture, tile, and store imagery to disk, one worker would be responsible for obtaining an image from an electro-optical sensor. Another would be responsible for breaking the image into tiles, and another for performing a decorrelating transform for entropy reduction and multiresolution representation. Still another would be responsible for compression, and another for storage.

To maximize robustness in an experimental environment, we created a manager that could monitor, kill, restart, and configure the entire processing pipeline, including the shared cache and worker processes. In the event of hardware or software failure, individual elements of the pipeline were restored automatically and with minimum data loss.

Working with Mark Duchaineau we attempted to determine a set of ideal coefficients for the wavelet used to create the multiresolution hierarchy. While a workable set exists, it produces undesirable artifacts at high levels of loss, so more work needs to be done to identify a more ideal set.

The tiler and data structures for tiling and display have been added to the code base we developed. As currently implemented it takes approximately one second to tile and build a multiresolution hierarchy for an 11-megapixel image on a 3.8 GHz Pentium 4. This is too slow for some uses, so some work will probably need to be done to optimize the code. We may be able to leverage commodity graphics processors to gain extra speed. With the advent of quad-core CPUs it may also be possible to get required throughput by tiling the image and then having several processes run in parallel to build the multiresolution hierarchies for the resulting tiles.

Additionally, more work needs to be done to define a data format for storing image tiles on disk, as well as retrieving and displaying them. As this functionality is desirable (and required, if we are to make large data easily viewable) we will be spending a lot of time developing this.

As a final note, it was recently brought to our attention that Oracle provides an extension to its database technology that provides most, if not all, of the functionality that we are working on as part of this project. We will be investigating this further; if a suitable solution already exists it may be more valuable time-wise to integrate it into projects that require storing and displaying large images.