

## White Paper – Building Megapixel Video Applications

Many industries are now starting to integrate megapixel capable cameras and imagery into their architectures. The trending toward megapixel sensors is similar to the migration from analog CCTV to IP Video in some respects. Megapixel cameras have advantages that include a broader field-of-view thus eliminating the need for multiple VGA level cameras covering the same surveillance space; the ability to detect and zoom in on detail such as a license plate or face; and the feature to create areas-of-interest or subdivisions of the megapixel video scene which can be independently controlled. The military, especially systems designed for intelligence analysis, require processing massive amounts of disparate video from large repositories. The video may not be megapixel resolution, but the repositories contain vast amounts of video. Coral creates an efficient method to add pluggable filters to process large amounts of data – these filters can be added or removed from the processing runtime pipeline on-the-fly without taking down the whole application and restarting.

The early adopters of megapixel imagery included retail and banking that placed megapixel cameras at entry points to insure good face and feature recognition in case of a security incident. As well, casinos use megapixel cameras to provide detailed coverage of gaming tables to detect chip swapping and slight-of-hand movement. As camera manufacturers compete and the cost of megapixel cameras decreases, the desire to add megapixel capabilities into existing systems or construct megapixel pure systems will greatly increase. Software therefore must progress to meet this challenge.

Mamigo Coral was designed as an integrated development environment providing a plug-n-play video processing architecture. Coral manages megapixels of video and takes advantage of multiprocessor machines with advanced core and memory management features. The primary objective was to build an efficient method to develop robust video and imagery applications that achieved high efficiencies with commonly available computing platforms. Coral is platform neutral and applications are compatible with Linux or Windows operating systems in 32 bit or 64 bit architectures.

There are some hurdles to fully integrating megapixel imagery into legacy architectures – the largest technical challenge is to have the ability to manage massive amounts of input from megapixel sensors in software but also the processing platform. A server grade machine can become quickly overwhelmed with eight to 12 megapixels of input on its memory bus without management.

Coral has overcome many of these challenges by adopting the following techniques:

- Video and imagery in a processing pipeline is not copied from one process to the next, but transported in packets or “buckets.”
  - Bucket pool – common reusable memory pool for allocating buckets

- Bucket counting – All buckets are reference counted and reclaimed upon release
- Bucket Builder – Allocated buckets from memory pool based on stream parameters
- Bucket Reader - Read data in a bucket, such as an image, is based on stream parameters

Coral also has a built-in threading model that manages initialization, starting and stopping; thread controls, input queue management, and thread affinity management. Our architecture provides kernel level management of memory and threading so the developer does not have to worry about these aspects when building a video/imagery application. The development energy is focused on building testable, reusable, and pluggable filters and drivers.

An administrator or developer assembles a pipeline of filters and drivers into a graph (processing pipeline). The pipeline could be very simple with an input filter, transform filter and renderer. Or, very complex with multiple input sources, branches in the pipeline, and multiple outputs. Once these filters are created, a systems administrator can assemble the graph using xml and execute the runtime environment. Because the pipeline consists of filters/drivers, Coral based applications can be built using a distributed development plan.

The building blocks for Coral are configurable filters. The following are different types of filters we have defined:

- Source Filters – capture data from external sources, e.g., capturing video, reading an AVI file.
- Transform Filters – processes and manipulates data, e.g., transcoding, motion detection.
- Synch Filters – transmit processed data out of the engine, e.g., encoded video, extracted metadata.
- Device Filters – supports source and synch to adapt to external communications protocols
- Control Filters – con troll runtime behavior of the graph, e.g., managing network communications.

Other building blocks are drivers which enable communication with external software and devices such as a pan-tilt-zoom camera. Drivers are written using a simple, low-level API with methods for connection ( ); disconnect ( ); read ( ); and write ( ). We also support the concept of having a device driver pool which is a resource of multiple drivers called by the Mamigo environment. Example drivers include HTTP driver which interacts with HTTP servers, Webcam driver for webcams; drivers for specific cameras such as a Point Grey megapixel camera.

Mamigo has advanced filters for 2D/3D visualization, video synthesis, advanced video analytics, video stabilization, and filters for digital video recorders, object detection and pattern matching. Using Coral, developers can design and build customized video and imagery

applications quickly without having to develop and re-develop the kernel functions to manage video/imagery in the processing pipeline.

Integrating megapixel video into existing, legacy applications may be a challenge and a reason to delay migration from CCTV to megapixel video. The output from Coral can be fused with existing Command and Control, Physical Security Information Systems easily by leveraging the Coral runtime and Coral API. And, building applications to leverage megapixel video with existing filters including motion detection, areas-of-interest and renderers is straight forward.

Coral can also be configured to process large amounts of data from a repository. For instance, an analyst who has the task of searching and analyzing video of any type can create a processing pipeline to run pattern detection looking for specific textual strings and then add a filter that matches vehicle types and then outputting results to the analyst workstation. These filters can be put into and removed from the pipeline based on the analyst's search requirements.

### **Point Of Contact:**

Manoj Aggarwal, [contact@mamigo.us](mailto:contact@mamigo.us)