MMF Format description:

The MMF stores sequential frames of video and is designed for instances in which the background is unchanged between frames and only small parts of the image change between each frame. Compression is achieved by only storing the sections of the image that differ from frame to frame. Otherwise, image data is uncompressed, so further compression can be achieved using standard compression software, e.g. zip.

The basic structure of the MMF is the image stack. This represents a set of images with a common background. Each image stack begins with this common background image and is followed by a sequence of frames ("background removed images") which mark rectangular sections where the frame differs from the background image and provide replacement image data.

To reconstruct a given frame from an image stack, begin with the background image, then replace the image data in the rectangles specified by the background removed image.

A note on the generation of mmfs: In the most common implementation, the video is assumed to be a dark field image of objects moving against a static background. In this case, the background is taken to be the minimum of all the images in the stack. However, because the MMF specifies replacement of the background data, rather than addition or subtraction, there is no need to know how the stack was recorded in order to reconstruct it; simply replace the specified region with the specified data.

How an image stack is stored on disk:

Each image stack begins with a 512 byte zero padded header. The size of the header is specified in "StaticBackgroundCompressor.h." The first 4 bytes are an id code (0xbb67ca20, the CRC32 hash of "StaticBackgroundCompressor" from fileformat.info). This allows different implementations of the StaticBackgroundCompressor to be recognized and properly loaded. So far only the basic class has been defined. Image stacks with different properties can be created by subclassing the StaticBackgroundCompressor class. Subclasses should redefine the static constant IdCode and override the method idCode(). The corresponding id code and proper methods to load the stack from disk should be added to the StaticBackgroundCompressor loader methods "fromFile" and "getHeaderInfo"

Following the id code, the next 4 byte int is the size of the header in bytes (512 bytes here), then the total number of bytes occupied by the stack on disk, then the number of frames on disk (call this NFRAMES). The rest of the header is padded by zeros.

Following the header is the background image. This is stored as an IplImage header (see cxtypes.h from OpenCV 1.0 for a definition) followed by the image data. As is the image data stored by an IplImage in memory, the image data is row ordered (http://en.wikipedia.org/wiki/Row-major_order) and interlaced. This means that data from one row is stored contiguously and that if there is more than one channel

(e.g. R G B), the value for each channel is stored at each pixel location. For example, the usual layout of a color image is: $B_{00}G_{00}R_{00}B_{10}G_{10}R_{10}$ etc. For help with the IplImage standard, see learning openCV from o'reilly or http://opencv.willowgarage.com/documentation/basic_structures.html#iplimage When reconstructing the IPL image using non OpenCV methods, be sure to understand the relation between width, widthStep, the number of channels and the number of bytes per pixel. Note that recording and restoring MMFs has only been extensively tested for single channel 8 bit images.

Following the background image are NFRAMES (number of frames on disk) background removed image frames. These frames are stored in chronological order. E.G. the first frame of the first stack is frame 0, the second is frame 1, etc. Each background removed image frame consists of the following:

A 1024 byte zero padded header. The first 4 bytes of the header are the id code (0xf80921af, the CRC32 hash of "BackgroundRemovedImage" from fileformat.info). Again subclasses should create their own id codes and this id code and appropriate load method should be added to the BackgroundRemovedImageLoader class. The next 4 bytes are the header size in bytes followed by the image depth (defined using the appropriate OpenCV constants, e.g. IPL_DEPTH_8U) and number of channels; these values should be the same as those previously defined in the background image header. The next 4 bytes are the number of image blocks (NUMIMS) stored in the background removed image. The remainder of the header is occupied by optional metadata stored by the recording software; rules for loading this metadata vary according to the type of metadata, but appropriate load methods should be defined based on the IdCode that begins every metadata. Parsing the metadata is not required to reconstruct the images, but more information can be found by looking for the appropriate classes or id codes in the c++ source code.

Following the header are NUMIMS image blocks. Each block begins with a 16 byte CvRectangle – 4 integers specifying the x,y coordinate of the lower left (with the image origin in the lower left corner) coordinate of the rectangle followed by its width and height. Following this is the row-ordered (http://en.wikipedia.org/wiki/Row-major_order) interlaced data that replaces the background image in the specified rectangle. This means that data from one row is stored contiguously and that if there is more than one channel (e.g. R G B), the value for each channel is stored at each pixel location. For example, the usual layout of a color image is: $B_{00}G_{00}R_{00}B_{10}G_{10}R_{10}$ etc.

The MMF consists of a series of image stacks, as described above prefaced by a 10240 byte header. The beginning of the file is plain text and describes the contents of the MMF (for an example, see end of this document). The plain text is terminated by '\0' (the byte 0), then a 4 byte id code, the size of the header, then parameters used in the recording. This entire header can be skipped for the purpose of reconstructing the video sequence. The image stacks are stored in chronological order. EG, if there are 90 frames per stack, then the first frame of the first stack is frame 0, the second is frame 1, and the first frame of the second stack is frame 90, etc.

Thus the MMF format looks like

MMF:

HEADER: Plain Text \0 idCode, header size in bytes, recording params – zero padding to make header the right size
Image Stacks

Image Stacks:
HEADER: IdCode, header size in bytes, size of stack on disk in bytes, NFRAMES (number of images in a stack), zeros to make header correct size
IPLIMAGE: IplImage header, row-ordered interlaced data.
NFRAMES x Background Removed Image

Background Removed Image:
HEADER: IdCode, header size in bytes, depth, nChannels, NUMIMS, metadata, zeros to make header correct size
NUMIMS image blocks: [x y w h] row-ordered interlaced data

For an example of how to read an MMF, see the StackReader class. Note that on opening an MMF, the stack reader goes through the entire file and makes a table storing the location on disk of the beginning of each image stack and the indices of the video frames stored in that stack. This allows rapid access to individual frames. Future extensions of the MMF may include such a table in the header, but for now it is necessary to create it from scratch when loading the stack.

EXAMPLE MMF TEXT HEADER

Set of Image Stacks representing a movie. Beginning of file is a header, with this format:
10240 byte zero padded header beginning with a textual description of the file, followed by \0 then the following fields (all ints, except idcode)
4 byte unsigned long idcode = a3d2d45d, header size in bytes, key frame interval, threshold below background, threshold above background
Header is followed by a set of common background image stacks, with the following format:
Stack of common background images, beginning with this header:
512 byte zero-padded header, with the following fields (all 4 byte ints, except idcode):
4 byte unsigned long idcode = bb67ca20, header size in bytes, total size of stack on disk, nframes: number of images in stack
Then the background image, as an IplImage, starting with the 112 byte image header, followed by the image data
Then nframes background removed images containing only differences from the background, in this format:
BackgroundRemovedImage: header is a1024 byte zero padded header with the following data fields (all 4 byte ints, except id code)
4 byte unsigned long idcode = f80921afheadersize (number of bytes in header), depth (IplImage depth), nChannels (IplImage number of

channels), numims (number of image blocks that differ from background) then metadata:
Composite Meta Data: idcode (unsigned long) = 9844e951, int number of Image Meta Datas stored, then each Meta Data in succession:
TProcessedDataProperty, a 60 byte data structure containing information about image aquisition from Mightex Camera engine.
Name-Value MetaData: idcode (unsigned long) = c15ac674, int number of key-value pairs stored, then each pair
in the format \0-terminated string of chars then 8 byte double value
header is followed by numims image blocks of the following form:
(16 bytes) CvRect [x y w h] describing location of image data, then interlaced row ordered image data