Hello,

Every body had tried to produce the new G2 video system, but so far nobody is satisfied.

The current G2 implementation that is somehow extended pull-push method is too complicated, in both video filter system and video filter itself. The new out-of-order and duration time features doesn't help making the things simpler.

OK why i write this document? First I want to summing my ideas, and to give the others possibility to extend them and to criticize my work so far (please don't flame;)

Here are few features that I'm trying to achieve:

- decreasing memcpy by using one and same buffer by all filters that can do it (already done in g1 as Direct Rendering method 1)
- support of partial rendering (slices and DR m2)
- support for get/release buffer (ability to release buffers when they are no longer needed)
- out of order rendering – ability to move the data through the video filters if there is no temporal dependency
- display_order rendering – this is for filters that need to use temporal dependences
- ability to keep as many incoming images are needed and to output as many images as filter may need to (e.g. in case of motion blur we will have e.g.6 incoming images and 6 outgoing at once)
- support for PTS.
- ability to quickly reconfigure and if possible - to reuse data that is already processed (e.g. we have scale and the user resizes the image, - only images after scale will be redone), safe seeking, auto-insertion of filters.
- ability to have more complicated graph (than simple chain) for processing.
- simple structure and flexible design.

In short the ideas used are :

- common buffer and separate mpi – already exist in g1 in some form
- counting buffer usage by mpi and freeing after not used – huh, sound like java :O
- allocating all mpi&buffer before starting drawing (look obvious, doesn't it?) – in G1 filters had to copy frames in its own buffers or play hazard by using buffers out of their scope
- using flag IN_ORDER, to indicate that these frames are “drawn” and there won't come frames with “earlier” PTS.
- using common function for processing frame and slices – to make slice support more easier
- emulating complicated graph in a simple linked list.
- messaging system for dropping/rebuilding MPI's.(not yet finished)
- having prepared simple type filters (like non temporal – one input/one output, processing the frame as it came, without carre for buffer management) (not documented)

Data Flow, Buffer Management

From long time I propose something called data flow. In short all filters decode functions are called loop and they may do no rendering, or may render any number of

frames they have. The other basic idea is to allow filters to store frames in 2 arrays like the current vo2_drivers does. One array for input frames and the other for output. This idea complicates the things more because the video filter may need to have one and same frame in both arrays. This problem was resolved by “Dalias” when he propose him pull system.

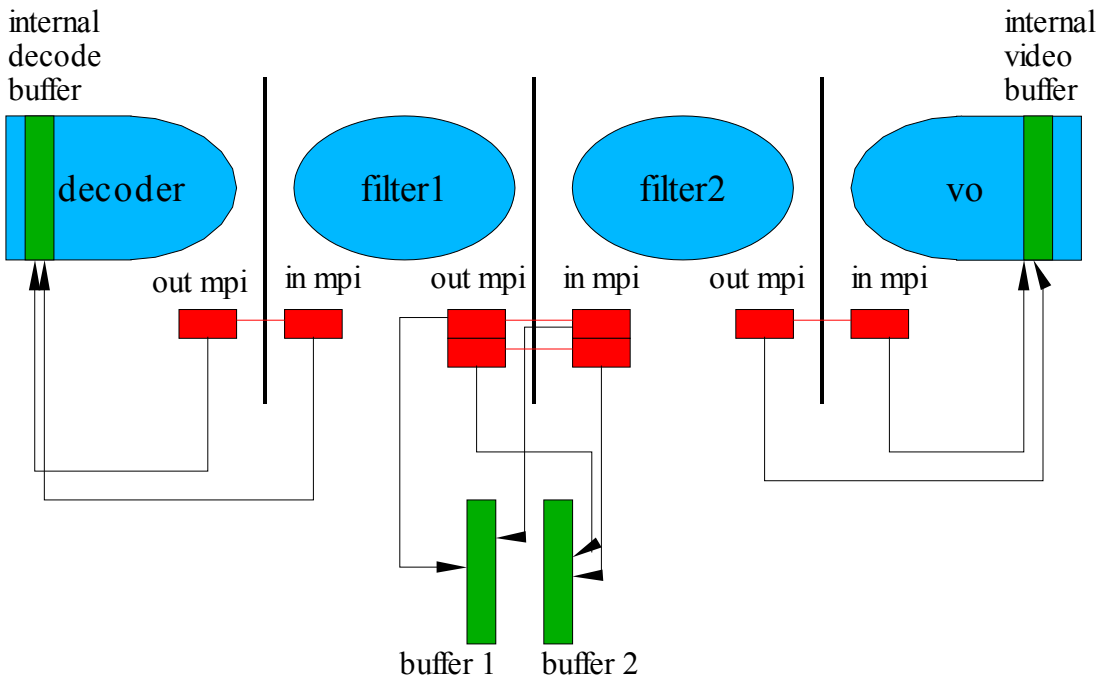
So, the frame is split on 2 parts, one I will call **mpi** and the other I will call **mp_buffer**. The mp_buffer part contains the memory buffer, usage count, buffer common width and height, maybe stride. The mp_buffer->count is the number of MPI-s that point to that buffer**.** Probably we may allow buffer to contain more than one piace if memory (e.g. 3 memory blocks for Y,U,V planes).

The **mpi** structure have it own planes[],width, height, strides[], it also have fill counter. I reuse the line/slice counter from my previous vo2 draft.

I still wonder should vf_filter1 outgoing mpi to be common with the vf_filter2 incoming mpi.

So all **mpi** are counted as buffer references, even if they are common for the filters. This will allow easier and natural locking of the buffers. Because when vf_filter1 frees mpi in the outgoing array, the corresponding mpi in the vf_filter2 incoming array will remain unchanged. This will allow filter e.g. to keep needed by filter but not needed by decoder images.

This scheme also allows to get rid of the static buffer type. Simply the decoder will never release it's mpi, but will pass it to the filter chain, multiple times (like ffmpeg's reuse). On the other side static buffers should always be in the main memory, otherwise they can take the only display buffer and stale displaying (e.g. vo with one buffer, and decoder with 2 static buffers)



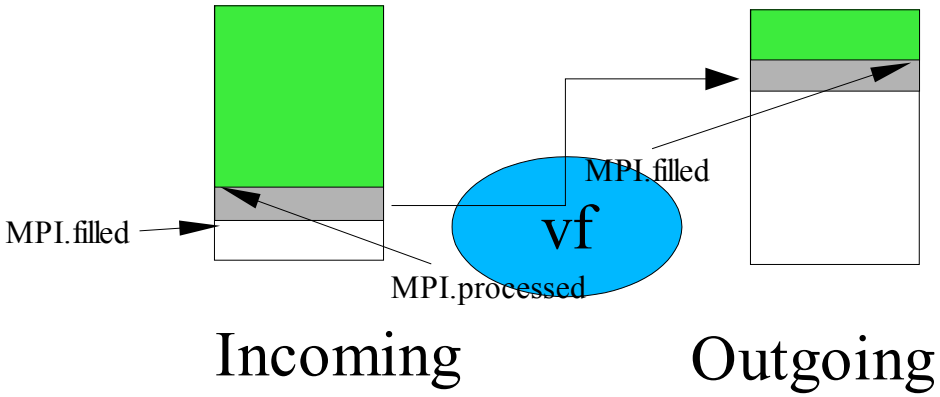
For now let's ignore the way mpi and mp_buffers are obtained (I think that light modification of get_image will do the trick)

Processing

Forget about put_image and draw_picture. All processing is done in one function that is very similar to draw_slice() or control(DRAW_IMAGE,...). I still have not found name, so for now i will call that function ***process()***. The process() functions of all filters are called in loop, until we have filled the **vo**, or until no processing is possible (filters need more data). The process function should process as mach data as it can. The mpi of incoming frames contain few variables and flags, that process should pay attention to. One of the are the counters.

The incoming **mpi** have counter that shows how many slices have been processed by our filter and other counter that show how many slices have been written to the image, and could be used to processing. The output **mpi** have the same counters but with different meaning – this time our filter will update the processed counter, to show what part of the image is been updated.

If you don't want to work with slice, then you can do it easy, just wait the incoming **mpi** processed counter to reach the end, and process the whole image at once.



Dalias already pointed that processing may not be strictly top from bottom, may not be line, slice, or blocks based. This question is still open for discussion. Anyway the most flexible x,y,w,h way proved to be also the most hardier and totally painful. Just take a look of crop or expand filters in G1. More over the current G1 scheme have some major flaws:

- the drawn rectangles may overlap (it depends only on decoder)
- drawing could be done in any order. This makes it very hard to say in what part of the image is already processed
- skipped_blocks processing is very hard. Theoretically it is possible to draw only non-skipped blocks, but then the above problem raise.

Out-of-order / IN_ORDER

The main problem is the out-of-order rendering. The filters should be able to process, the frames in the order they came. On another side there are some filters that can operate only in display order. So what is the solution?

By design the new video system requires PTS (picture time stamp). I add new flag that I call IN_ORDER. This flag indicates that all frames before this one are already available in the in-coming/out-coming area. Lets make an example with MPEG IPB order.

We have deciding order IPB and display IBP.

First we have I frame. We decode it first and we output it to the filters. This frame is in order so the flag should be set for it (while processing). Then we have P-Frame. We decode it, but we do not set the flag (yet). We process the P-Frame too. Then we decode an B-Frame that depends on the previous I and P Frames. This B-Frame is in order when we process it. After we finish with the B-Frame(s) the first P-Frame is in order. As you can see it is very easy for the decoders to set the IN_ORDER flag, it could be done om G1's decode() end, when the frames are in order.

This IN_ORDER flag it read for the incoming by the filters. And filters should set it into the out-going.

After some brainstorming, one may realise that this is the very old push scheme, but without recursion.

If an MPI is freed without setting IN_ORDER then we could guess that it have been skipped.

All frames that accidentally have earlier (lower) PTS than the IN_ORDER frame (in the same incoming array), are skipped, to maintain coxerensy, and big fat warning should be displayed for developers.

It would be easy to let the IN_ORDER frames to be first in the incoming array, and they also to be sorted by PTS. This should simplify processing.

Now about get/release scheme. Actually the filters receive incoming mpi, and should get_image() on their own. Also the filter should release ALL mpi-s that are no longer needed.

Skipping/Rebuilding

Now the skipping issue is rising. I propose 2 flags, that should be added like IN_ORDER flag, I call them SKIPPED and REBUILD. I thought about one common INVALID, but it would have different meening depending from the array it resides (incoming or outgoing)

SKIPPED is requared when a get_image frame is gotten but the processing is not performed. The first filter sets this flag in the outgoing mpi, and when next filter process the date, if should free the mpi (that is now in the incoming). If the filter had allocated another frame, where the skipped frame should have been draw, then it can free it by setting it as SKIPPED.

SKIPPED could also be used for keeping sync for temporal filters, but it is question of another discution (e..g when decoder skip a frame, should a SKIPPED frame be passed to the chain, so other filters could use it PTS).

E.g. if we have this chain

-vf crop=720:540,spp=5:4,scale=512:384

This chain should give quite a trill to 2GHz processor. Now imagine that scale is auto inserted and that the vo is some window RGB only device (vo_x11). If a user change the window size, scale parameters change too. Scale should rebuild all frames that are processed, but now shown. Scale filter can safely SKIP all frames in the outgoing.

As you see this is could be done on the config change.

REBUILD is the opposite of SKIPPED, it is set by the last filter (vo), when some of the image parameters have been changed. When a filter finds this flag set in its outgoing array, it can do 2 things. First one is to pass the REBUILD flag to the previous filter (by setting REBUILD in the incoming). The second thing is to free the mpi, to allocate it again by get_image(), and to process it again.

Well not all things are clear, but the main idea is to reuse the processed frames.

E.g. if we have

-vf spp=5,scale=512:384,osd

and scale filter had taken direct buffer from the vo(vo_xv with buffer in the main memory under AGP aperture). Of course OSD can use only one buffer, so it reuses the direct buffer scale had taken.

Now the user turns off OSD that have been already rendered into a frame. Then

vf_osd set REBUILD for all affected frames in the incoming array. The scale filter will draw the frame again, but it won't call spp again. And this gives a big win because vf_spp could be extremly slow.

Now you see why both flags could be combined as INVALID.

On another side, there is one big problem – the mpi could already be freed by the previous filter. To workaround it we may need to keep all buffers until the image is shown (something like control(FLIP,pts) for all filters). Same thing may be used on seek, to flush the buffers.

The get_image chain stays the same e.g. get_image stays the same with the difference that when a filter returns mpi, then the same mpi is added as its incoming mpi.

Filter Graph

Now, about multiple chains. I don't like the idea of complicated graphs. I propose to first go with simple solution and let all headaches by callbacks for the next video system. The simple solution is to use non circling, one way graph , that could be “smashed” and converted to chain. How?

take this is example graph,

Fig3. <TODO>

You see that the graph is quite complicated. Fortunately it could be broken of few linear segments. We number these segments. Now we smash all segments into one chain. The filters should process only mpi's that have same segment number and simply bypass mpi-s that don't have (like they are transparently).

SPEED

All filters should process as mach data as possible before returning. This mean that processing of a frame is completed when all processing is done. This guaranties that on slice rendering filters won't start buffer management stuff, or processing another frames, that are not in the CPU cache.

Also there should be flags passed to the process() functions, mainly to indicate the changes of the data structures:

NEW_INCOMING_MPI – there is new mpi in the incoming array
INVALID/SKIPPED/REBUILD_MPI – some of the mpi have the attribute set
SLICE_MPI_CHANGE – new data in the buffer
IN_ORDER_MPI_CHANGE – image have been drawn

I think that it is possible to separate the buffer management, and image processing in 2 different functions. This way slice processing should be speed up.

There is another problem. You see that REBUILD requares calling filters in reverse order, then in the straight order. Maybe something like buble-sort shaking could help a little (calling filters process() in pull order then in push).

Problems remaining!

1. Interlacing – should the second field have its own PTS?
   Probably it would be easier to process all fields separately. If frames had duration, then half of the duration is for one field. But in my implementation duration is redundant. Duration could be calculated by offset between two PTS, but this requere having of 2 ready(in_order) frames, to display one. You see that this is a general problem and adding duration will only complicate the things (e.g. what a filter should do if duration is 2 times bigger than the time to the next frame.
   The possible solutions are:
   1.1. To have separate mpi for every fields
   1.2. To have second PTS for the second field
   1.3. To have time offset for the second field, compared to the first field (aka duration of the first field)

All methods have their good and their bad sides.

2. Filters that depend on frame number or depend on frame pattern.
   I'm afraid that the data flow filters could delete and insert random number of frames, this way the filters (e.g like hard telecine removal that counts frames to get 4-5 frame pattern) will be totally fooled.
   One possible solution is to query if the filter could insert delete frames, and to avoid putting filters that do that before such filter. Unfortunately the fix is worse than the problem.
3. get_image thingie.
   Now imagine that we can get rid of get_image() function. Yeh, it is quite hard to imagine that it is possible to do it so. get_image() is brilliant, it allows not only allocating new frames but also making parameters trick, before and after image is gotten, and before or after return of mpi.
   Unfortunately it is quite complicated to be used, but it's flexibility is hard to achieve by other way.
   Well, It may be possible to change it and to build it into the new system. Let imagine that we have separate functions for buffer management. We could add flag GET_IMAGE, and fill the required values into the mpi. Then we could call the next filter. Instead of processing the frame , the next filter may try to get_image, by the very same way, or to allocate the buffer by itself (calling mp function).
   As you see there is no big win, but only more troubles.

P.S.

I absolutely forbid this document to be published anywhere. It is only for mplayer developers' eyes. And please somebody to remove the very old vo2 drafts, from the g1 CVS.