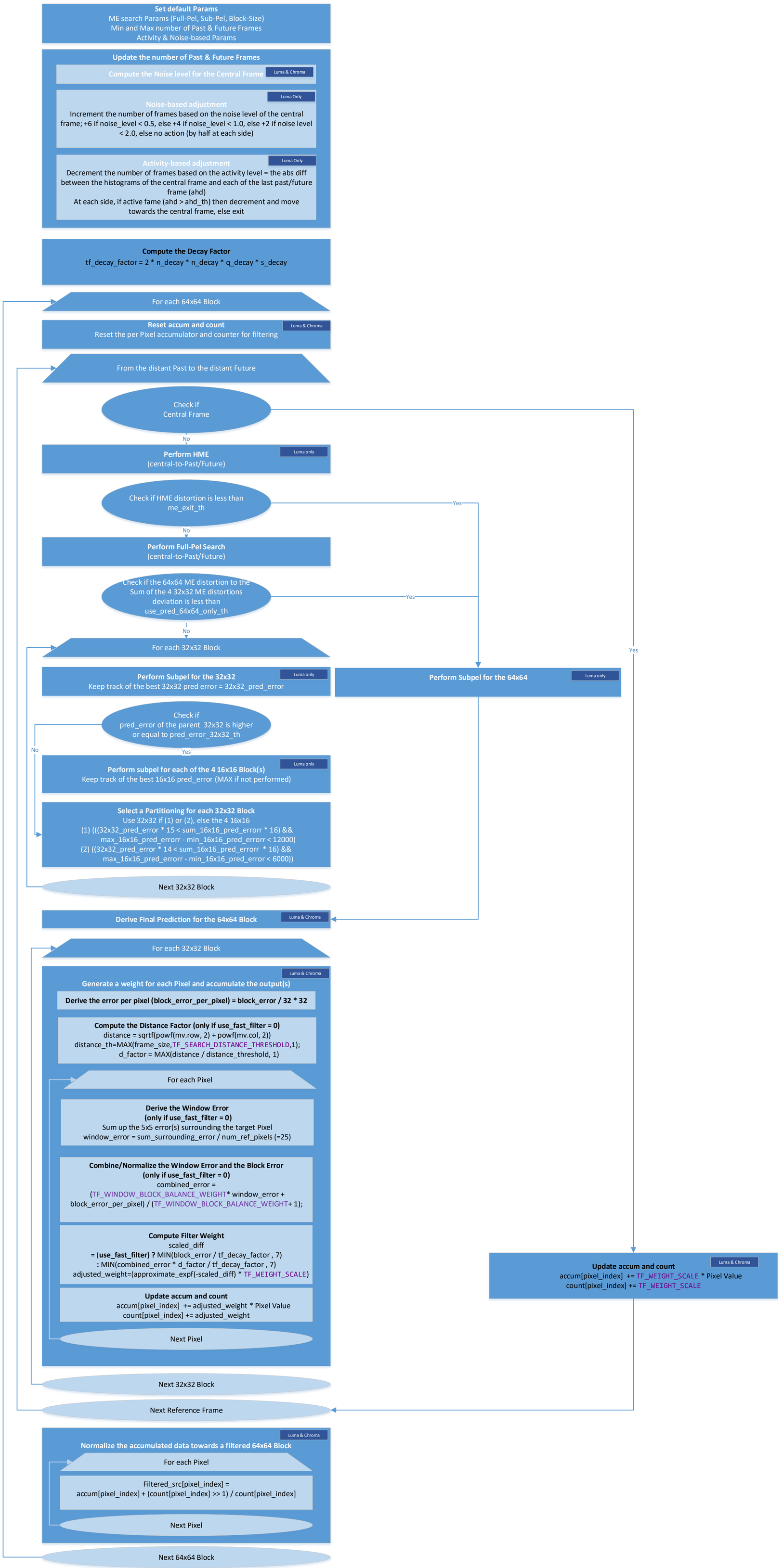
Set default Params
ME search Params (Full-Pel, Sub-Pel, Block-Size)
Min and Max number of Past & Future Frames
Activity & Noise-based Params
Update the number of Past & Future Frames
Compute the Noise level for the Central Frame
Luma & Chroma
Luma Only
Noise-based adjustment
Increment the number of frames based on the noise level of the central frame; +6 if noise_level < 0.5, else +4 if noise_level < 1.0, else +2 if noise level < 2.0, else no action (by half at each side)
Activity-based adjustment
Decrement the number of frames based on the activity level = the abs diff between the histograms of the central frame and each of the last past/future frame (ahd)
At each side, if active fame (ahd > ahd_th) then decrement and move towards the central frame, else exit
Compute the Decay Factor
tf_decay_factor = 2 * n_decay * n_decay * q_decay * s_decay
For each 64x64 Block
Reset accum and count
Reset the per Pixel accumulator and counter for filtering
From the distant Past to the distant Future
Check if Central Frame
No
Yes
Perform HME
(central-to-Past/Future)
Luma only
Check if HME distortion is less than me_exit_th
Perform Full-Pel Search
(central-to-Past/Future)
Check if the 64x64 ME distortion to the Sum of the 4 32x32 ME distortions deviation is less than use_pred_64x64_only_th
For each 32x32 Block
Perform Subpel for the 32x32
Keep track of the best 32x32 pred error = 32x32_pred_error
Perform Subpel for the 64x64
Check if pred_error of the parent 32x32 is higher or equal to pred_error_32x32_th
Perform subpel for each of the 4 16x16 Block(s)
Keep track of the best 16x16 pred_error (MAX if not performed)
Select a Partitioning for each 32x32 Block
Use 32x32 if (1) or (2), else the 4 16x16
(1) (((32x32_pred_error * 15 < sum_16x16_pred_errorr * 16) &&
max_16x16_pred_errorr - min_16x16_pred_errorr < 12000)
(2) ((32x32_pred_error * 14 < sum_16x16_pred_errorr * 16) &&
max_16x16_pred_errorr - min_16x16_pred_error < 6000))
Next 32x32 Block
Derive Final Prediction for the 64x64 Block
For each 32x32 Block
Generate a weight for each Pixel and accumulate the output(s)
Derive the error per pixel (block_error_per_pixel) = block_error / 32 * 32
Compute the Distance Factor (only if use_fast_filter = 0)
distance = sqrtf(powf(mv.row, 2) + powf(mv.col, 2))
distance_th=MAX(frame_size,TF_SEARCH_DISTANCE_THRESHOLD,1);
d_factor = MAX(distance / distance_threshold, 1)
For each Pixel
Derive the Window Error
(only if use_fast_filter = 0)
Sum up the 5x5 error(s) surrounding the target Pixel
window_error = sum_surrounding_error / num_ref_pixels (=25)
Combine/Normalize the Window Error and the Block Error
(only if use_fast_filter = 0)
combined_error =
(TF_WINDOW_BLOCK_BALANCE_WEIGHT* window_error +
block_error_per_pixel) / (TF_WINDOW_BLOCK_BALANCE_WEIGHT+ 1);
Compute Filter Weight
scaled_diff
= (use_fast_filter) ? MIN(block_error / tf_decay_factor , 7)
: MIN(combined_error * d_factor / tf_decay_factor , 7)
adjusted_weight=(approximate_expf(-scaled_diff) * TF_WEIGHT_SCALE)
Update accum and count
accum[pixel_index] += adjusted_weight * Pixel Value
count[pixel_index] += adjusted_weight
Next Pixel
Next 32x32 Block
Update accum and count
accum[pixel_index] += TF_WEIGHT_SCALE * Pixel Value
count[pixel_index] += TF_WEIGHT_SCALE
Next Reference Frame
Normalize the accumulated data towards a filtered 64x64 Block
For each Pixel
Filtered_src[pixel_index] =
accum[pixel_index] + (count[pixel_index] >> 1) / count[pixel_index]
Next Pixel
Next 64x64 Block