Eulerian Video Magnification and TBB

GRACE SZPYTMAN, CHERWA VANG

ECE 5772

Introduction to Eulerian Video Magnification (EVM)

EVM magnifies the subtle temporal variations in videos

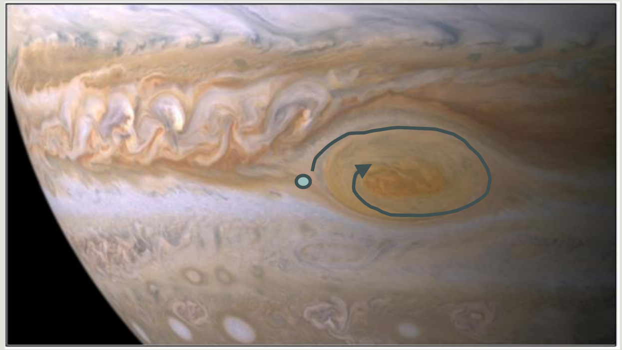
- **Motion magnification**
- Color magnification

EVM motion magnification works by:

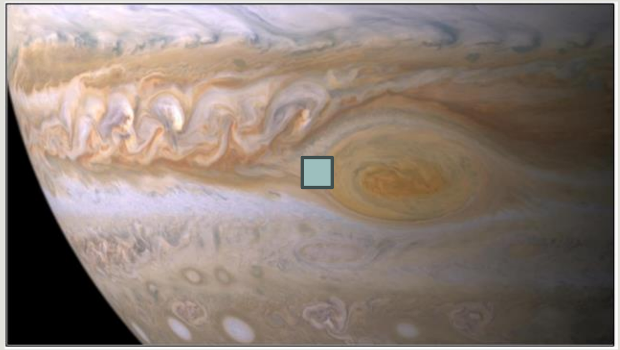
- Taking a video input
- Applying a spatial decomposition
- Applying a temporal filter
- Amplify resulting image
- Saving video output

Fluid Dynamics: Lagrangian vs Eulerian

- Lagrangian fluid dynamics
  - Tracks individual particles as they move through time and space
  - Particles in a 3d simulation



- Eulerian fluid dynamics
  - Analyzes fluid properties at fixed points in space
  - Pixels on an image from video feed



EVM: Video Feed

Convert from BGR to LAB/YIQ space

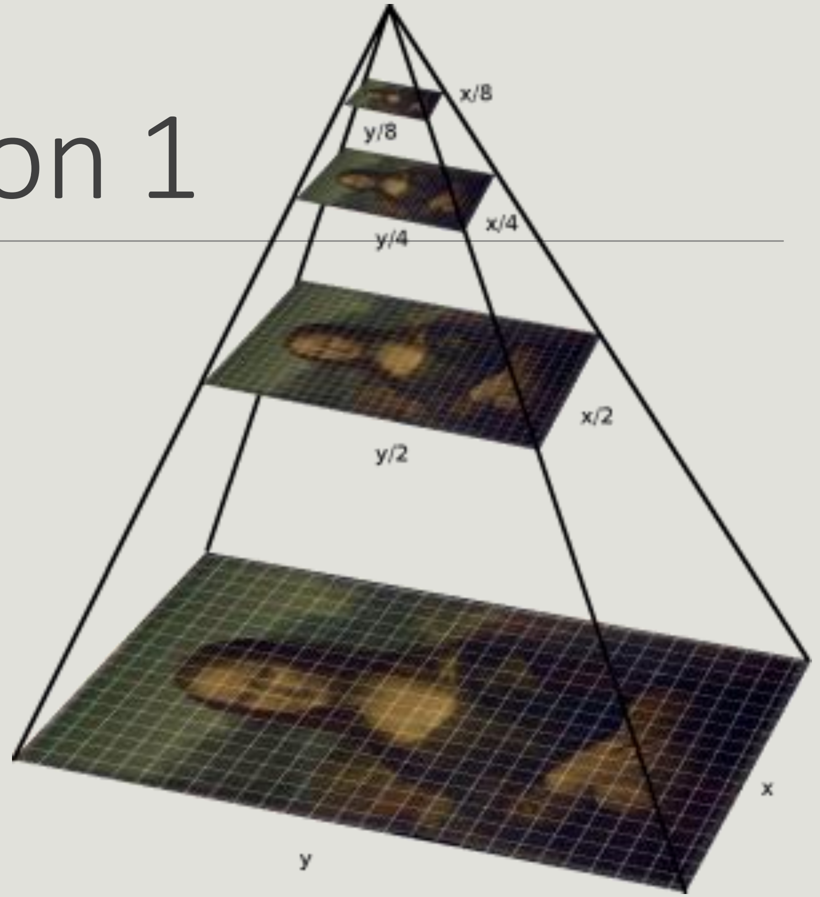
- BGR – blue, green, red channel
- LAB – Lightness, RedGreen, BlueYellow ch
  - More accurate way of representing how humans see color
  - Can better influence the colors of an image, without affecting its brightness
- YIQ – Brightness, I/Q chrominance
  - Same as LAB space

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.59590059 & -0.27455667 & -0.32134392 \\ 0.21153661 & -0.52273617 & 0.31119955 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

RGB to YIQ/LAB conversion

EVM: Video Feed

EVM: Spatial Decomposition 1



Apply spatial filter by building Gaussian pyramid

1) Apply 5x5 gaussian kernel
   - Low-pass filter that preserves low spatial frequencies

$$G_k = \frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

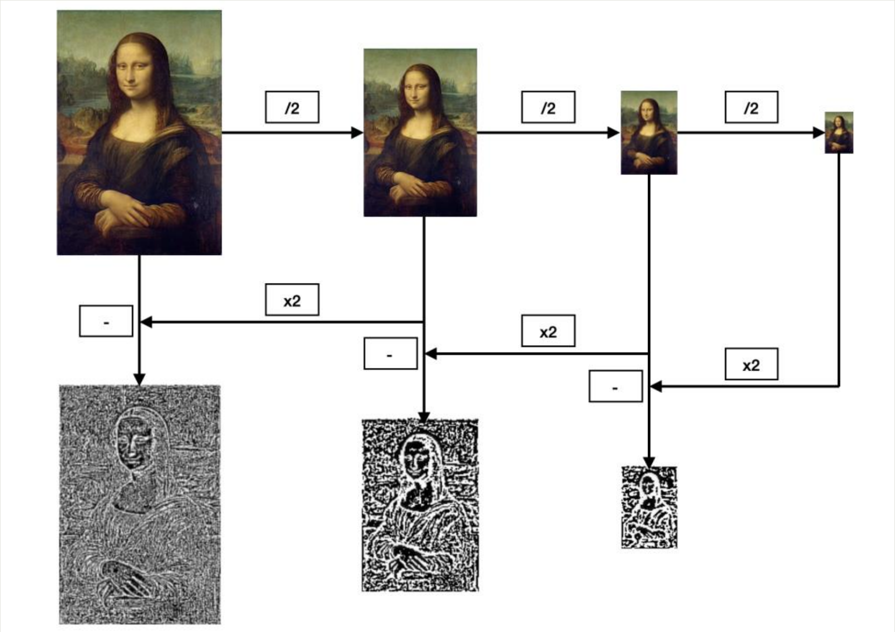
2) Downsample original image by 2 for each level

Video Feed → Spatial Decomp → Temporal Filter → Amplify Image → Store Frame

EVM: Spatial Decomposition 2

Apply spatial filter by building Laplacian pyramid (difference of two gaussian pyramid)

3) Upsample the downsampled images with black pixels
4) Apply Gaussian filter to smooth out
5) Get difference between Upsampled image
5) Repeat until original image size is achieved



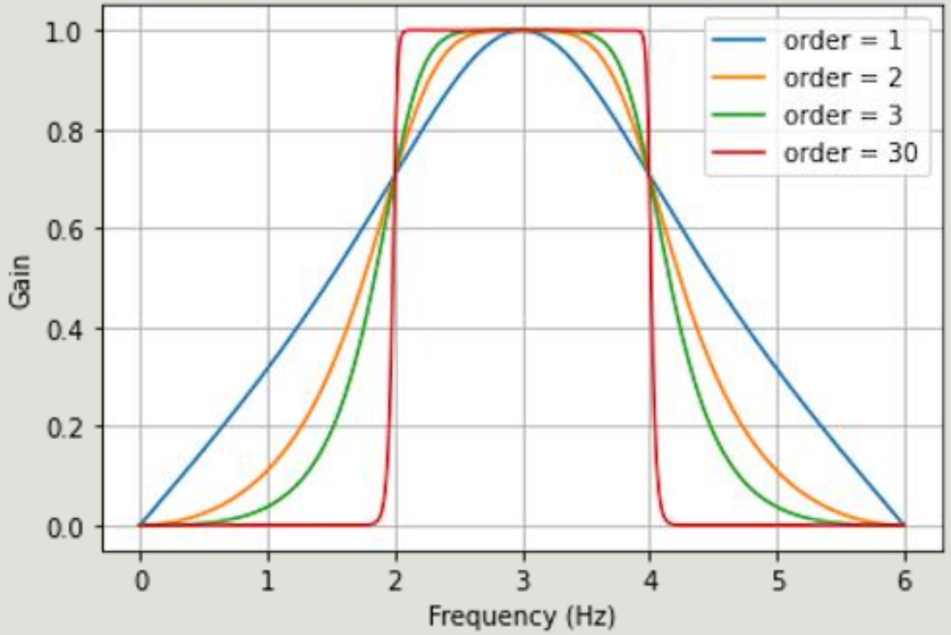
EVM: Spatial Decomposition 3

EVM: Temporal Filter

Apply a temporal filter with Butterworth Filter

- First order Butterworth bandpass filter
- Filter out frequencies that are not of interest
- Current frame relies on previous frames. Therefore, cannot process frames out of order
  - y[n] = nth temporal filtered frame
  - x[n] = nth Laplacian Pyramid
  - a0/a1 = feedback coefficient
  - b0/b1 = feedforward coefficient

$$y[n] = \frac{1}{a_0}(b_0 x[n] + b_1 x[n-1] - a_1 y[n-1])$$



Video Feed → cv::mat → Spatial Decomp → std::vector <cv::mat> → Temporal Filter → std::vector <cv::mat> → Amplify Image → std::vector <cv::mat> → Store Frame

EVM: Temporal Filter

EVM: Amplify Image

Amplify each level of Laplacian pyramid to amplify motion

- λ = llamda, special wavelength

$$\lambda = \sqrt{h^2 + w^2}$$

- δ = delta, displacement factor

$$\delta(t) = \frac{\frac{\lambda_c}{8}}{1+\alpha}$$

- α_{new} = alpha, amplify level

$$\alpha_{new} = \frac{\frac{\lambda}{8}}{\delta(t)} - 1$$

- Amplify each pixel value by the amplify value
  - $M_{Fp}[L]$ = output image
  - L = pyramid level

$$M_{F_p}[l] = \begin{cases} \alpha_l F_p[l] \text{ if Y component} \\ \alpha_l A F_p[l] \text{ if I or Q component} \end{cases}$$

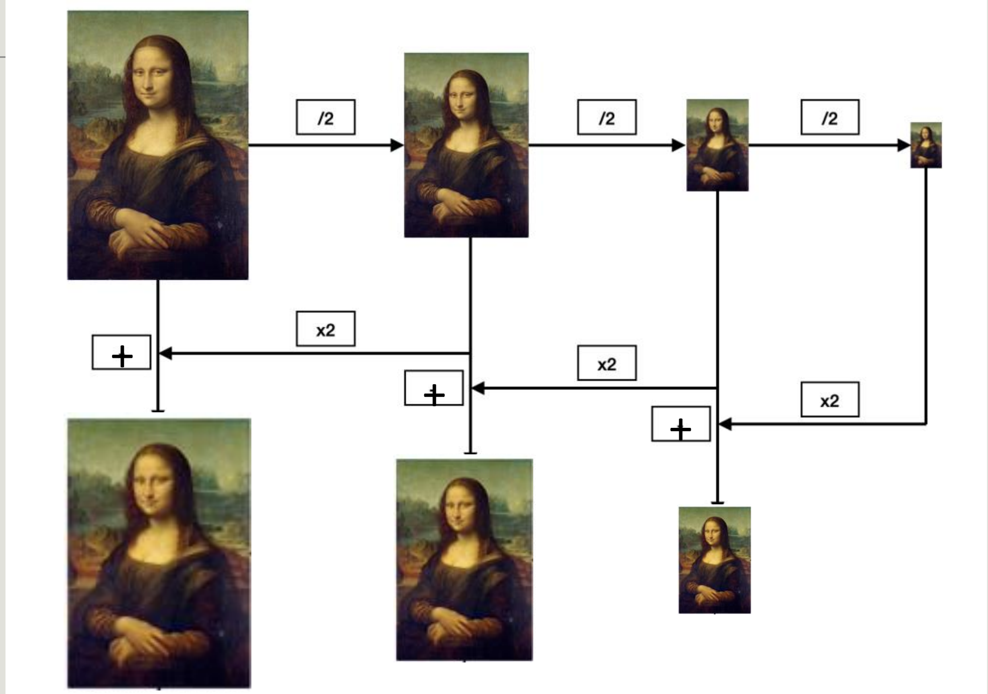
EVM: Amplify Image

Video Feed → cv::mat → Spatial Decomp → std::vector <cv::mat> → Temporal Filter → std::vector <cv::mat> → Amplify Image → std::vector <cv::mat> → Store Frame

EVM: Store Frame

Reconstruct image

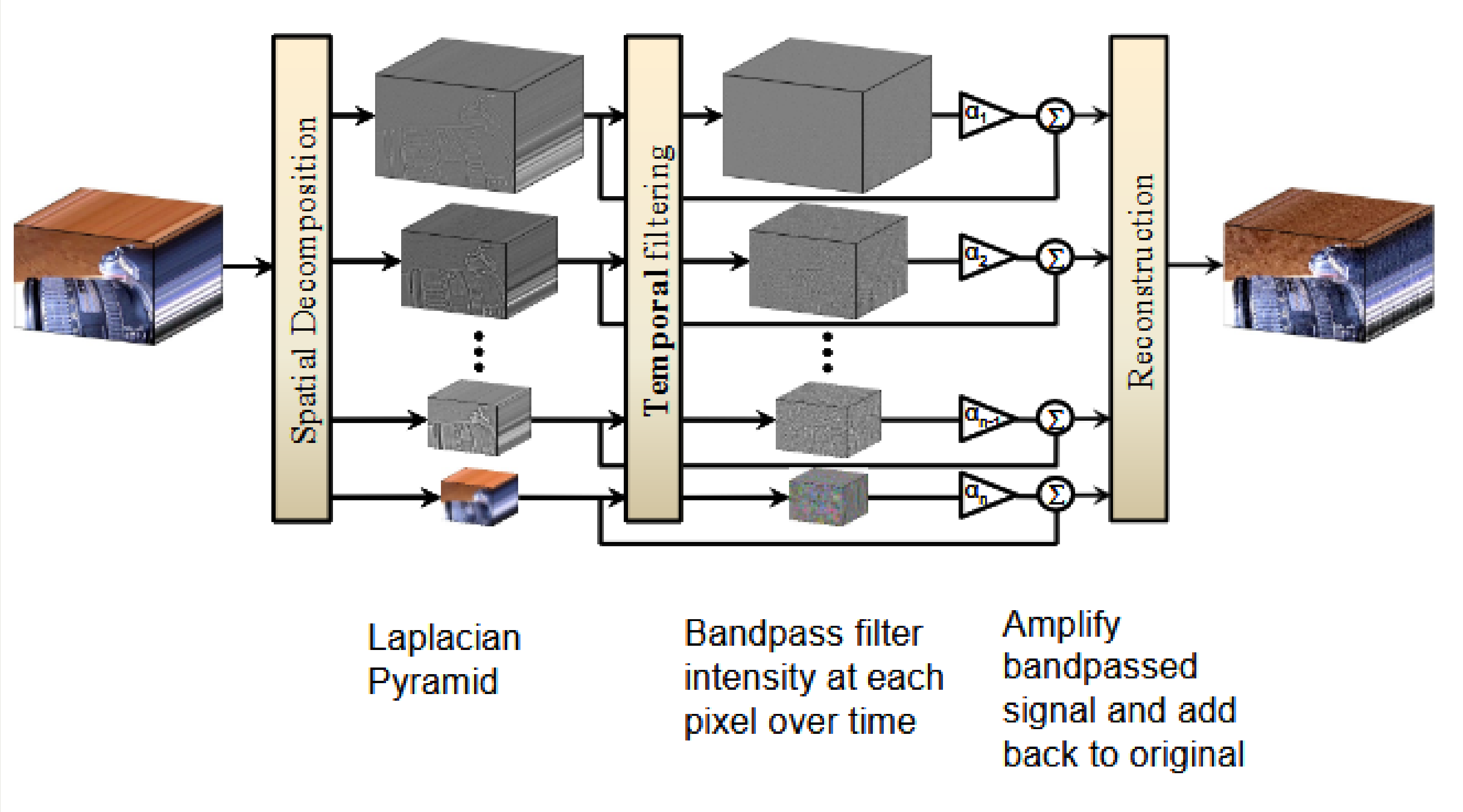
- Upsample downsampled images from Laplacian pyramid
- Add up all levels in Laplacian pyramid
- Attenuate IQ channels to normalize amplification
- Combine attenuated IQ image with original image
- Convert from LAB back to BGR color space
- Display/save output frame



EVM: Store Frame

Video Feed $\xrightarrow{\text{cv::mat}}$ Spatial Decomp $\xrightarrow{\text{std::vector <cv::mat>}}$ Temporal Filter $\xrightarrow{\text{std::vector <cv::mat>}}$ Amplify Image $\xrightarrow{\text{std::vector <cv::mat>}}$ Store Frame

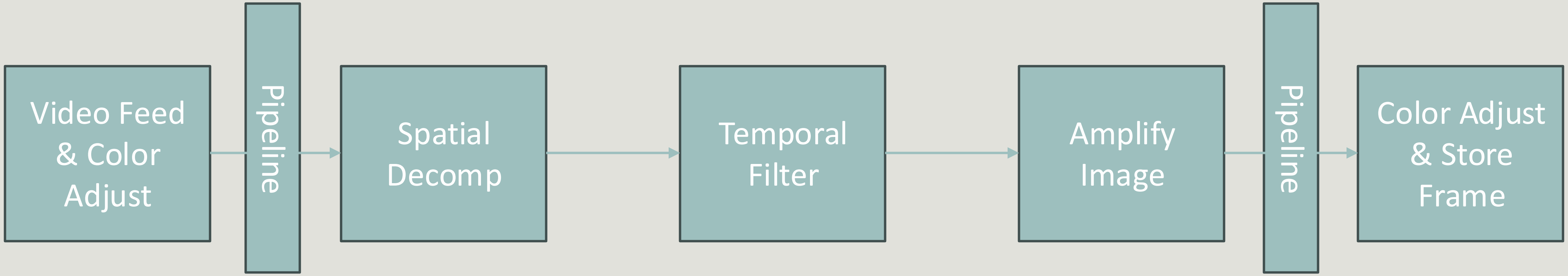
EVM High Level Overview



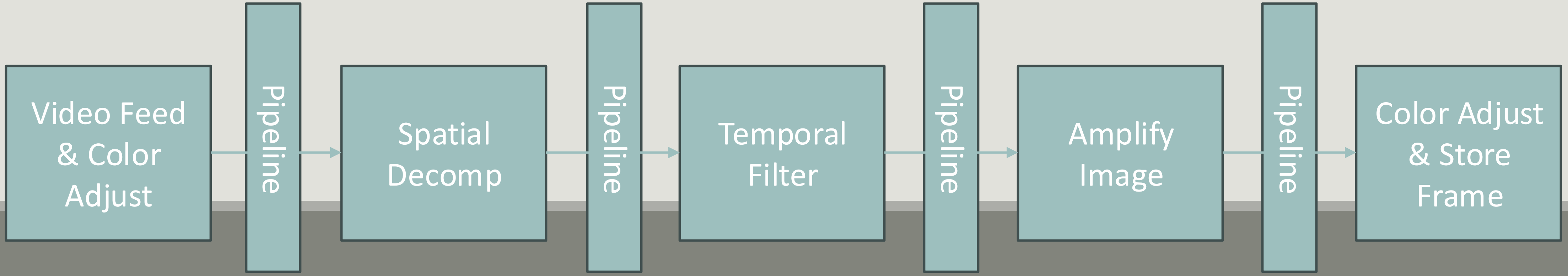
TBB: Parallelization Strategy 1

Parallel_pipe

- Divide the image processing into 3 stage pipelines



- Divide the image processing into 5 stage pipelines



Challenges

Parallel vs Serial_in_order

- Temporal Filter Equation: The current frame y[n] relies on y[n-1], and the Laplacian pyramid x[n] relies on x[n-1]

$$y[n] = \frac{1}{a_0}(b_0 x[n] + b_1 x[n-1] - a_1 y[n-1])$$

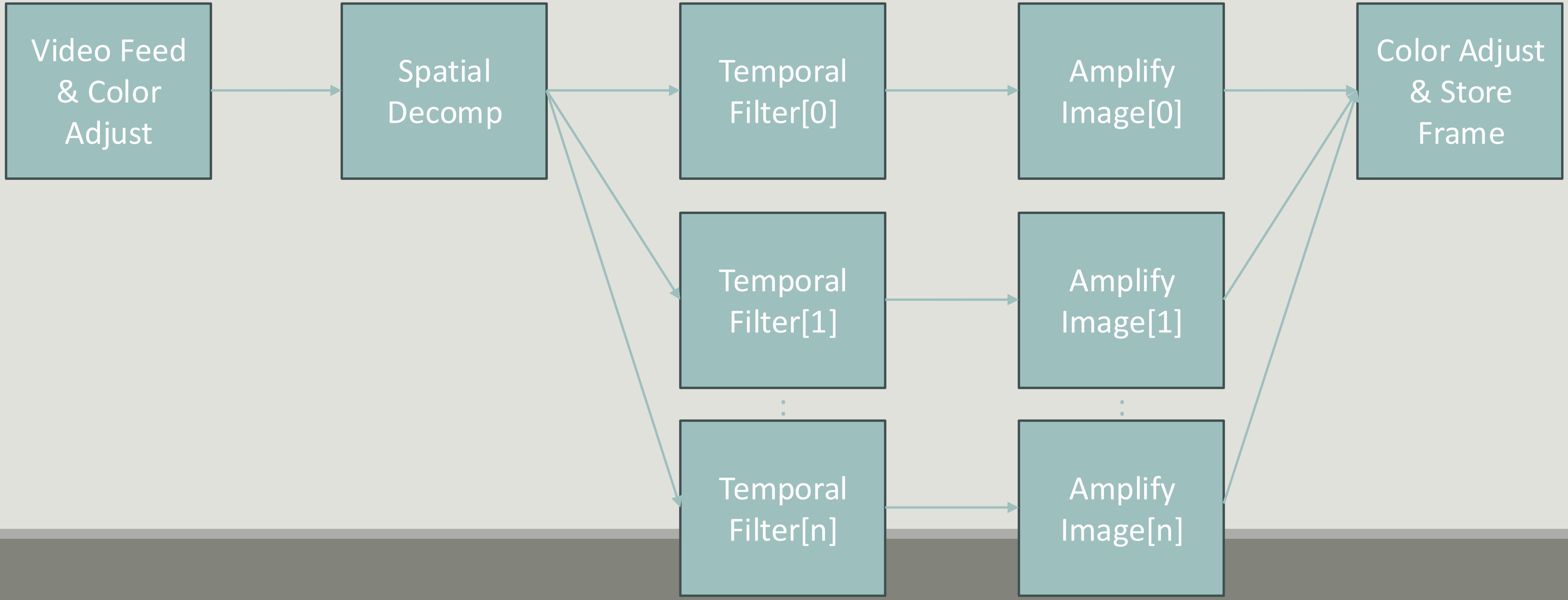
Performance improvement

- More pipes => less efficient
- Algorithm too efficient to optimize through adding pipes

TBB: Parallelization Strategy 2

Parallel_reduce

- Process multiple temporal filters in parallel
- Each parallel process amplifies a different frequency



Challenges

Parallel_reduce class instantiation used to much time for it to be a valid strategy for this application

- Class creation too heavy ~ 1s per frame

```
while(inputFrameParameter.frameNum >= 0){
  //get frame
  inputFrameParameter = GI();
  printf("*** frameNum = %i\r\n", inputFrameParameter.frameNum);
  //create new class
  EulerianMotionMag EMM(
                      input_width, input_height, levels,
                      cutoff_freq_low, cutoff_freq_high, lambda_c, alpha,
                      chrom_attenuation, exaggeration_factor, delta, lambda);
  parallel_reduce(blocked_range<size_t> (0,parallelFilterCount), EMM);

  //save data
  SO(EMM.img_input_);
```

Results 1: 3 stage pipeline

Single Threaded Timings 640x480

Trial	Frame (ms)
Trial1	19.92
Trial2	14.94
Trial3	19.92
Trial4	17.94
Trial5	16.95
Average	17.93

17.93 ms = 55.8 fps

TBB::Parallel_pipe Timings 640x480

Trial	Get (us)	Calc (ms)	Set (ms)
Trial1	176	6.7	33.525
Trial2	132	6.88	34.2
Trial3	196	6.87	34.7
Trial4	186	6.91	35.5
Trial5	152	6.58	33
Average	168.4	6.788	34.185

41.12 ms = 23.7fps

Single Threaded Timings 2280x3840

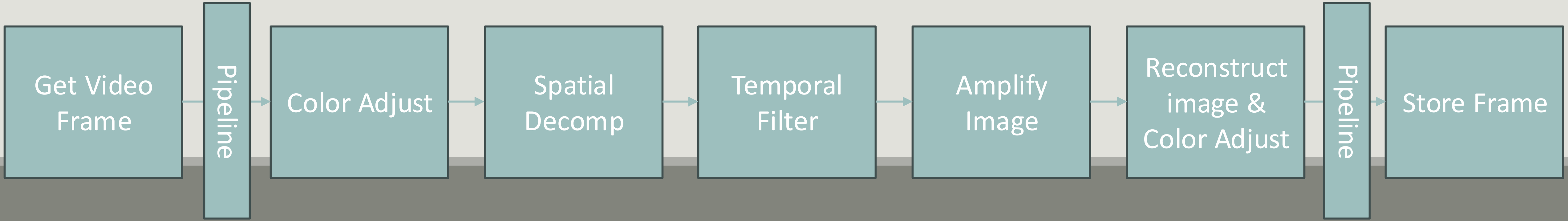
Trial	Frame (ms)
Trial1	913
Trial2	907
Trial3	925
Trial4	903
Trial5	896
Average	908.8

908.8 ms = 1.1 fps

TBB::Parallel_pipe Timings 2280x3840

Trial	Get (us)	Calc (ms)	Set (ms)
Trial1	4.5	1050	10.1
Trial2	4.2	1045	9.8
Trial3	4.6	980	11.6
Trial4	3.9	955	9.9
Trial5	4.4	1011	10.4
Average	4.32	1008.2	10.36

1023 ms = .98 fps

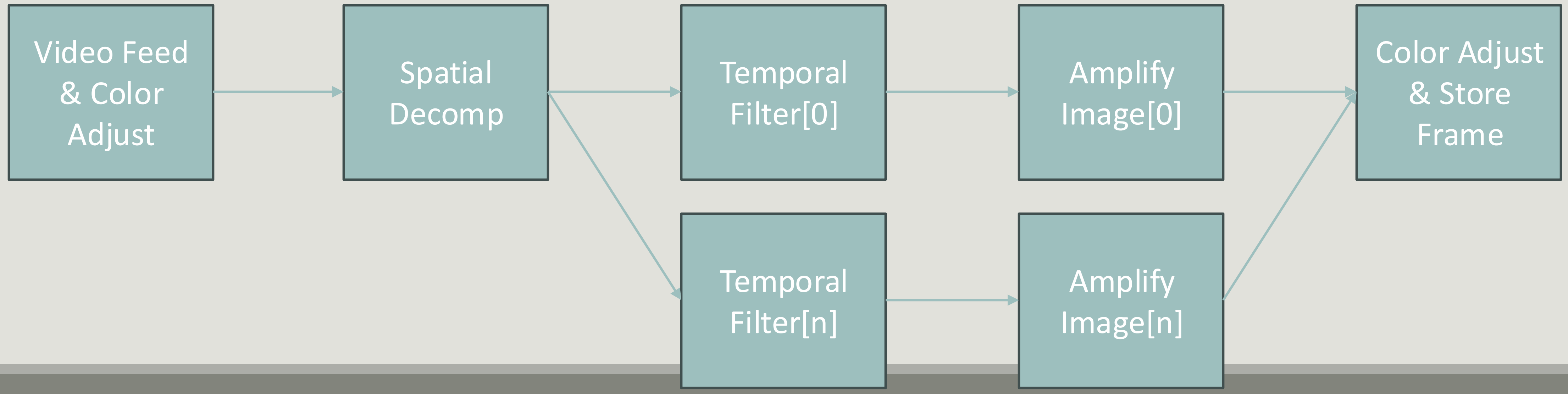


Results 2: Parallel Reduce

Slower than sequential ~1s per frame

Parallel Reduce Timings 640x480

Trial	Calc (s)
Trial1	1.04
Trial2	1.1
Trial3	1.16
Trial4	1.1
Trial5	1.07
Average	1.094



Result 3: 5 Stage Pipeline

Currently not working

Single Threaded Timings 2280x3840

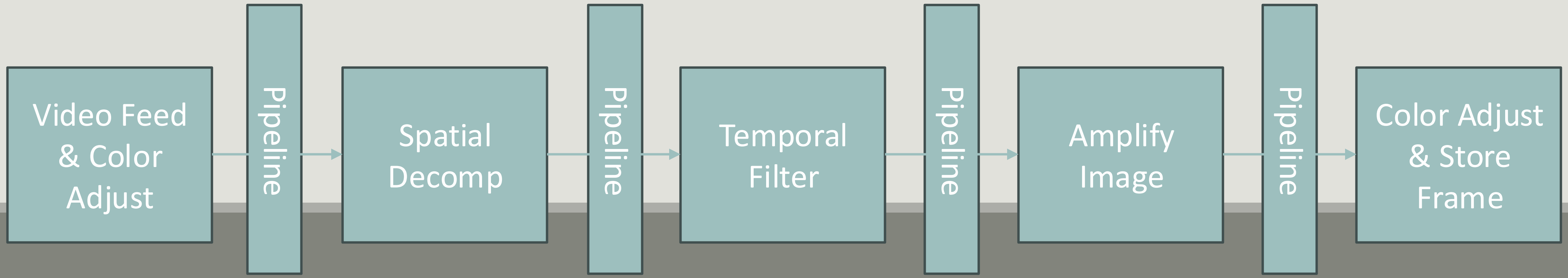
Trial	Frame (ms)
Trial1	913
Trial2	907
Trial3	925
Trial4	903
Trial5	896
Average	908.8

908.8 ms = 1.1 fps

Theoretical 5 Pipeline Timings 2280x3840

Trial	Get (us)	Spatial (ms)	Temp (ms)	Amp (ms)	Set (ms)
Trial1	4	360	409	245	33

Critical path => 409 ms = 2.44 fps



Future Improvements

Pass Laplacian level of frames to each pipeline
- Increases latency of video filtering
- Allows for out of order execution, since each parallel_pipeline has all the frames it needs to calculate the temporal filter

Increase Frame Rate
- Decreases processing time
- Some software calls wait on an input frame before continuing

Multiple frequency filters in parallel with parallel_reduce

In between filter analysis for feature detection

Demo/Questions