WINTER 2019

RESTORATION BY COMPRESSION

TABLE OF CONTENT

RESTORATION BY COMPRESSION – MOTIVATION

- The algorithm is intended to optimally restore an image that has been degraded during the acquisition phase, with respect to an output platform.
- This is done by introducing a novel compressor that uses existing, off-the-shelf compression methods.
- Our project focuses on optimising the restoration of the input image with respect to the acquisition, assuming perfect output.

PROJECT OBJECTIVES

- Familiarise ourselves with the “Complexity-Regularised Restoration” algorithm (version 1) presented in “Restoration by Compression” by Y. Dar, M. Elad and A. M. Bruckstein published on November 2018.
- Understanding the MATLAB implementation provided with the paper.
- Implement the algorithm in C++ to gain an improvement in runtime performance.
- Implement the algorithm in a modular and decoupled way to better allow for portability, thus taking a step towards making the algorithm production-ready.

ALGORITHM

1: Inputs: $\mathbf{y}$, β, θ.
2: Initialize $\hat{\mathbf{z}}^{(0)}$ (depending on the deterioration type).
3: $t = 1$ and $\mathbf{u}^{(1)} = \mathbf{0}$
4: **repeat**
5: $\quad \tilde{\mathbf{z}}^{(t)} = \hat{\mathbf{z}}^{(t-1)} - \mathbf{u}^{(t)}$
6: $\quad$ Solve the ℓ_2-constrained deconvolution:
$$\hat{\mathbf{x}}^{(t)} = \underset{\mathbf{x}}{\operatorname{argmin}} \|\mathbf{H}\mathbf{x} - \mathbf{y}\|_2^2 + \frac{\beta}{2}\left\|\mathbf{x} - \tilde{\mathbf{z}}^{(t)}\right\|_2^2$$
7: $\quad \tilde{\mathbf{x}}^{(t)} = \hat{\mathbf{x}}^{(t)} + \mathbf{u}^{(t)}$
8: $\quad \hat{\mathbf{z}}^{(t)} = CompressDecompress_\theta\left(\tilde{\mathbf{x}}^{(t)}\right)$
9: $\quad \mathbf{u}^{(t+1)} = \mathbf{u}^{(t)} + \left(\hat{\mathbf{x}}^{(t)} - \hat{\mathbf{z}}^{(t)}\right)$
10: $\quad t \leftarrow t + 1$
11: **until** stopping criterion is satisfied

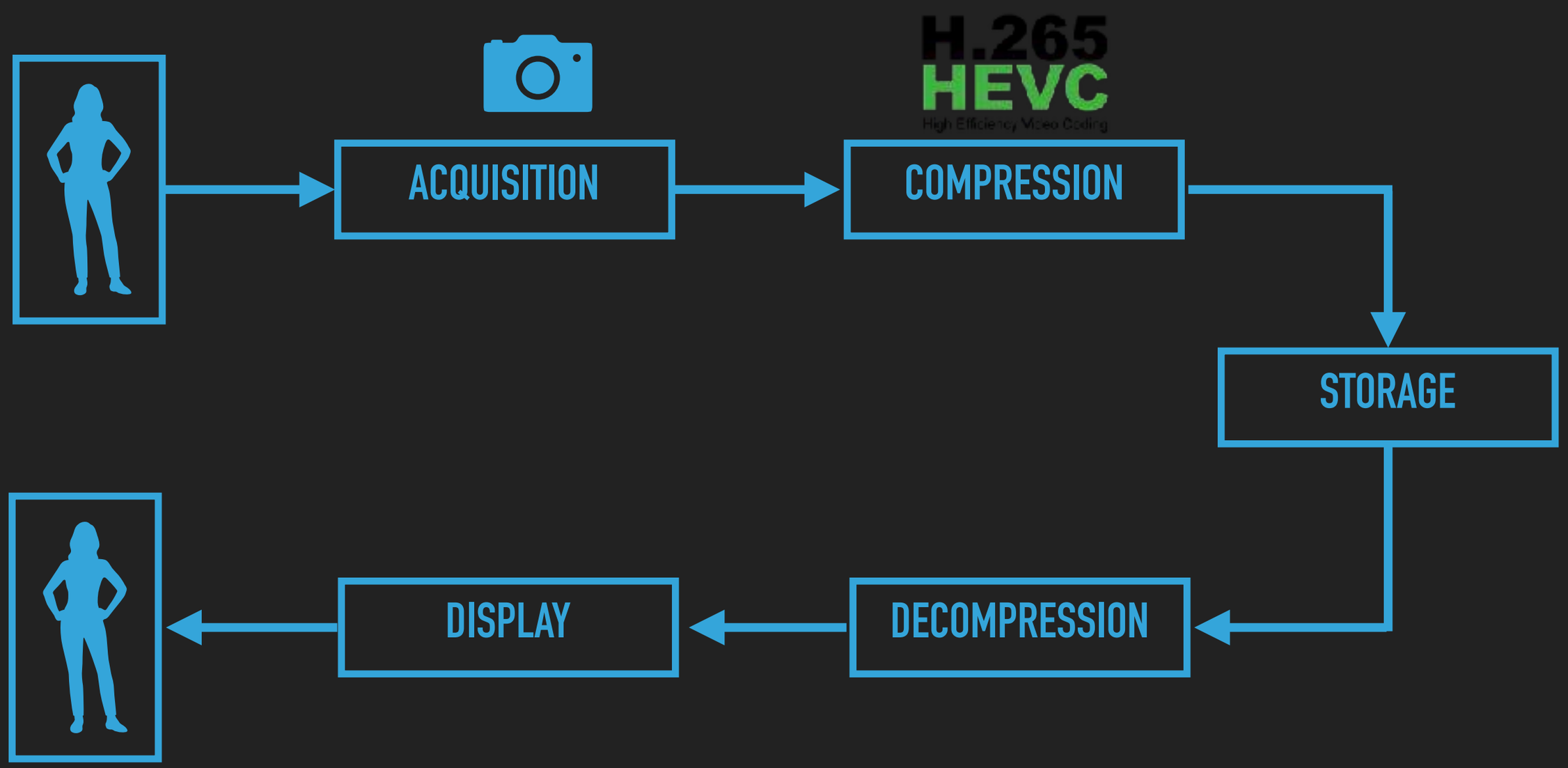
ALGORITHM DESCRIPTION

- Assumption:
  - The input image y is the degraded image x via a blurring operator, denoted by H, and additive zero-mean Gaussian noise with standard deviation of 1, denoted by n.
  - The degradation model is $y = Hx + n$ where x, y and n are a real M dimensional vector and H is an M x N matrix.
- Inputs:
  - y - the degraded image.
  - beta - optimisation parameter.
  - theta - compression parameter.

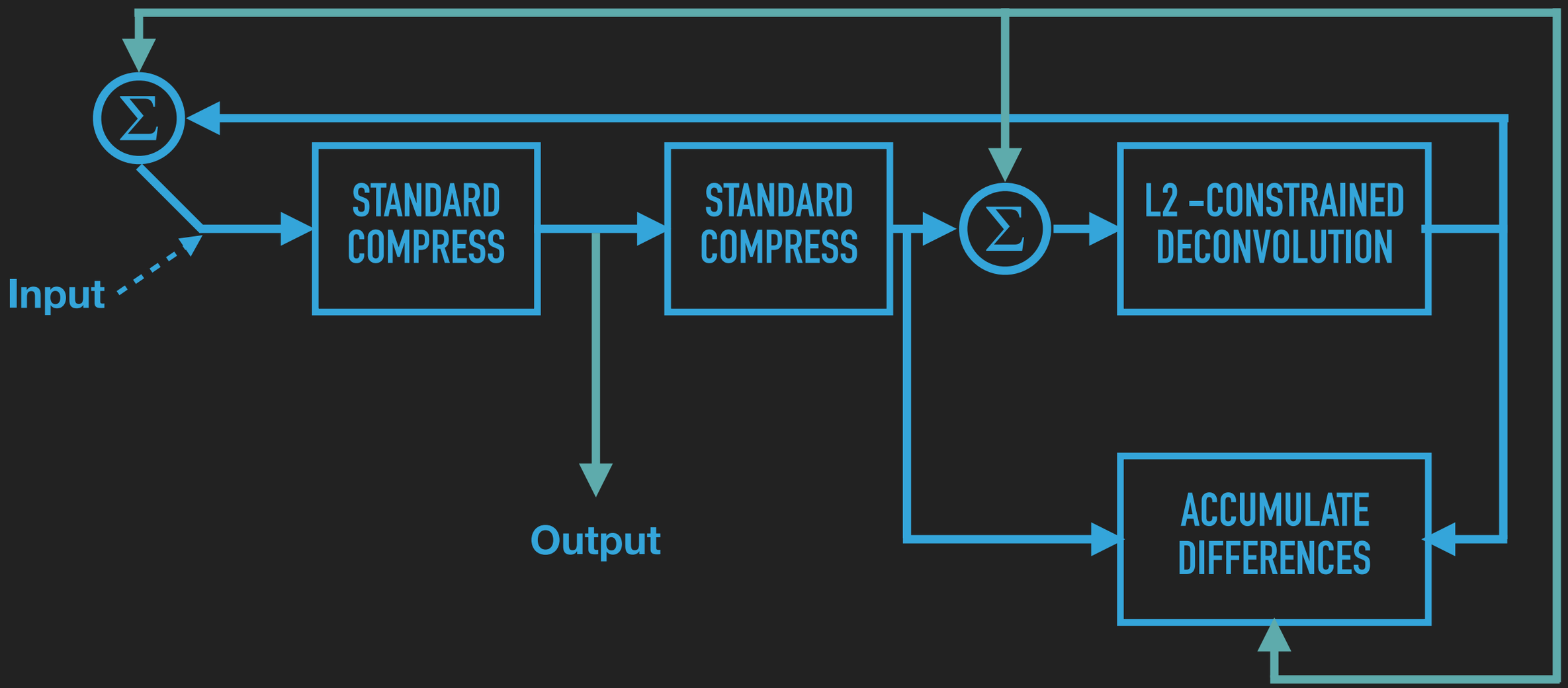
ALGORITHM DESCRIPTION

- The algorithm is composed of iterative improvement of the degraded image using standard compression and decompression methods (in our case H265).
- In each iteration we take the resulting image from the previous iteration and optimise it according to a predetermined metric. We then reiterate the compression and decompression phase to improve the image further.

IMAGING SYSTEM VISUALISATION



ALGORITHM DESCRIPTION



IMPLEMENTED CLASSES

- ImageWrapper - This class is used for holding the image as an OpenCV matrix and supplying the API to interact with it. It also helps with analysis and evaluation of the program's behaviour.
- InputImageSimulator - This class generates a degraded image based on the previously described assumptions. The result is the image the algorithm tries to restore.
- RestorationByCompression - This class handles the actual implementation of the algorithm by utilising a sequence of functions.

NOTABLE DESIGN ASPECTS

- The described design allowed us to create a decoupled program that is modular and can be easily modified and adapted for future use cases, while still being very efficient.
- While writing the implementation of this design we utilised object oriented design concepts to keep the program as decoupled as possible.

REPLACING MATLAB WITH A C++ FRAMEWORK

- In order to implement the algorithm we had to find a framework that was suitable for DSP usage, that is rich enough to provide the functionality that MATLAB provides and flexible enough to fit our needs.
- In the beginning we considered using Boost.GIL as our framework, though OpenCV has proven to be more flexible and widespread.

MULTI DIMENSIONAL LINEAR SOLVER

- To solve the optimisation problem, the MATLAB implementation used the BiCG (Bi-Conjugate Gradient descent) built-in function.
- OpenCV doesn’t provide this function out of the box.
- We’ve considered several possible solutions:
  - Eigen.
  - IML++.
  - OpenCV’s stochastic gradient descent.

IMPLEMENTING BICG USING IML++

- IML++ provided us with an efficient implementation of BiCG, but it was not compatible with OpenCV.
- We modified the given IML++ code to support OpenCV's matrices.
- This required us to gain a deep understanding of the BiCG algorithm.
- In addition, due to the fact that the operator H{*} is extremely large, the Matlab code represents it using its convolution kernel.
- This added an additional challenge to our implementation, since both IML++ and OpenCV don't support this kind of representation.

IMAGE RUNTIME REPRESENTATION DIFFERENCE

- We have noticed a significant difference in pixel runtime representations between the MATLAB and C++ implementations.
- This has led to different results in the numerical calculations performed in our implementation.
- As a result, the stopping criteria set by the MATLAB implementation were not met.
- We therefore modified the stopping criteria accordingly to stop after four iterations (to match MATLAB's iteration number).

ALGORITHM RESULTS

PSNR: 25.32 dB

PSNR: 33.21 dB

ALGORITHM RESULTS

PSNR: 30.09 dB

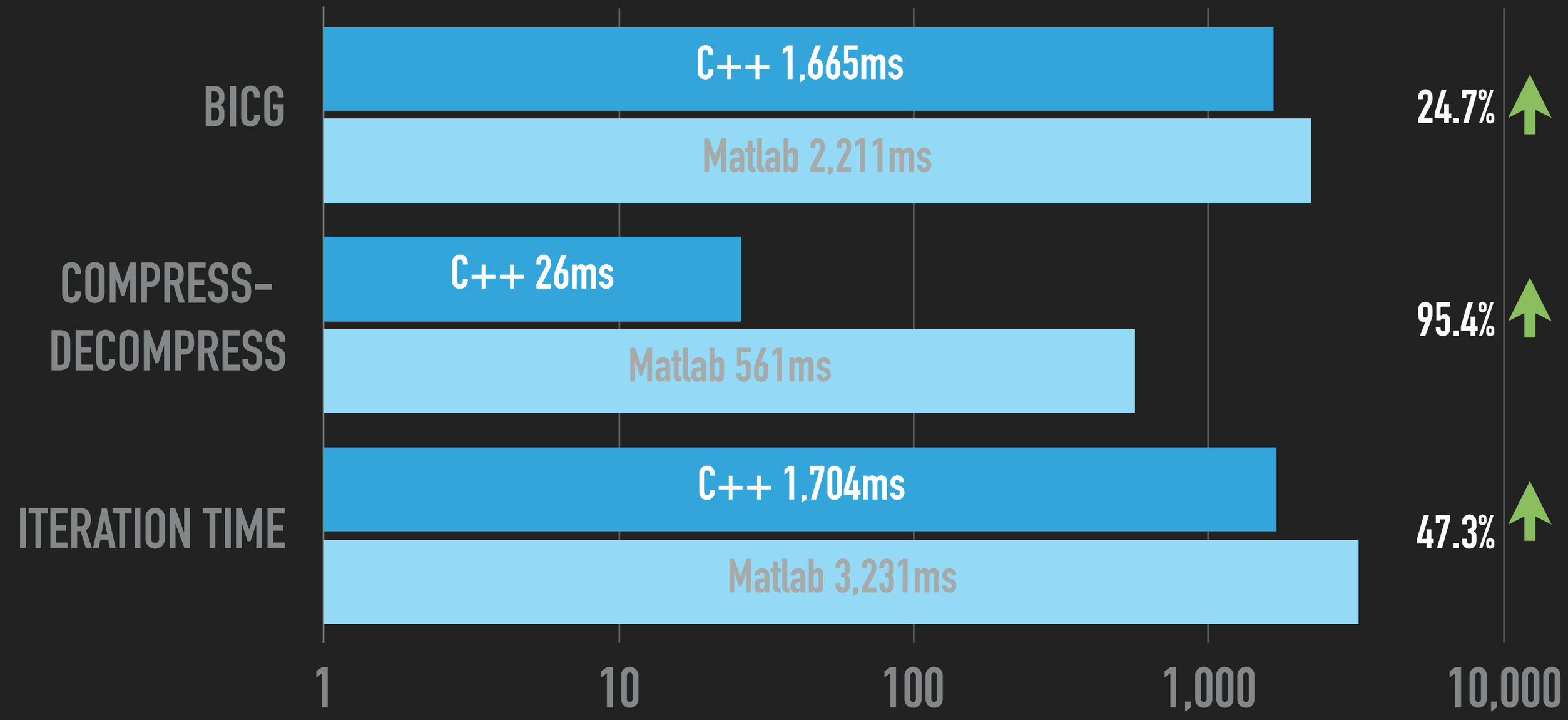
PSNR: 35.75 dB

ALGORITHM RESULTS

PSNR: 30.99 dB

PSNR: 32.72 dB

EFFICIENCY COMPARISON



CONCLUSION – ALGORITHM STOPING CRITERIA

- We believe the difference in pixel representations is due to the PNG library bundled with openCV versus the one provided by MATLAB.
- Because of these differences, our implementation did not meet the stopping criteria set by the MATLAB implementation, and tweaks had to be done a posteriori.
- It is left for further work to decipher whether the original mathematical operations still hold with these new pixel value representations.

CONCLUSIONS – PERFORMANCE

- The C++ implementation improves over the MATLAB implementation for a single iteration cycle.
- This results in total runtime improvement when the same number of iterations is carried out in both implementations.
- Due to C++'s lower overhead, the Compress-Decompress runtime improves drastically.

CONCLUSIONS – EVALUATION METHOD

- Evaluation and comparison of our results with respect to PSNR values has its faults, due to the differences in pixel representations.
- Future work should either utilise an identical representation, or adjust the mathematical operations the OpenCV's variant.
- Either way, integrating the changes should be a minor task, due to the modular design.