Spatio-temporal retinex-inspired envelope with stochastic sampling: A framework for spatial color algorithms

N8EN11D Projet Traitement d'image / Image Processing project

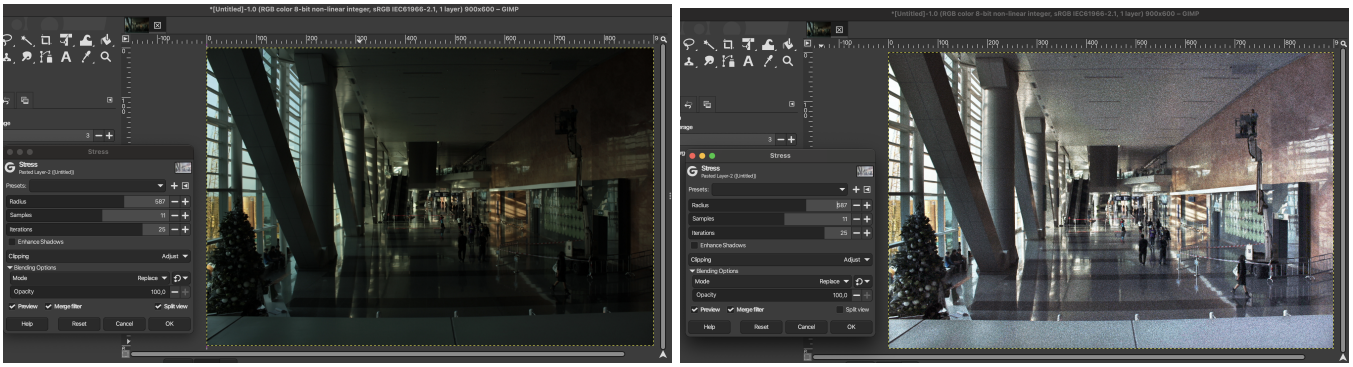

Figure 1: The Stress algorithm in GIMP, here used to enhanced the details (right) of the low-light image on the left. In GIMP you can access the STRESS algorithm via Color -> Tone Map -> Stress

1 Introduction

GIMP [6], a widely used open-source image editing software, includes a tone mapping algorithm called "Stress" (Spatio-Temporal Retinex-Inspired Envelope with Stochastic Sampling). This algorithm enhances local contrast, performs color correction, and applies to various image processing tasks.

In this project, you will implement the Stress algorithm based on the original research paper [2] and compare your implementation with the GIMP version. The goal is to understand the method, evaluate its performance, and explore its applications in different image-processing tasks.

2 Project Objectives

1. **Implementation**
   - Implement the Stress algorithm in a programming language of your choice (C++/Python with OpenCV or Matlab are preferable).
   - Ensure the code is **well-structured and modular**, with proper decomposition into functions.
   - Use the **same parameters as described in the paper** and avoid hardcoding inputs; your implementation should be easy to run with **any given image and a set of user-defined parameters**.
   - *While not required, optimizing the implementation with **parallelization techniques** (e.g., OpenMP for C++, `parfor` for MATLAB, etc.) could be a **bonus**— but only if you are familiar with these techniques, as the main focus should remain on the implementation and evaluation.*
2. **Evaluation & Comparison**
   - Compare your results **qualitatively and quantitatively** with GIMP's implementation. Use appropriate **image quality metrics** such as SSIM, PSNR, or other relevant metrics.

- Investigate the role of key parameters in the algorithm and their effect on the results as explained in the paper.
- Select **one of the applications** described in the paper (see below) and evaluate the algorithm on the specific application using some public dataset providing a ground truth reference and, where possible, add a comparison with alternative, off-the-shelf methods.

  (a) **Local contrast enhancement of grayscale/color images**
  The goal is to improve the visibility of features in an image with low-light or low-contrast without over-enhancing noise.
  Example of dataset: LOw Light paired dataset (LOL) [7] has pairs of images in low and normal light.
  Alternative methods: see CLAHE and/or MultiBandBlender in OpenCV, adapthisteq and/or imlocalbrighten in Matlab.

  (b) **High Dynamic Range (HDR) image rendering**
  High Dynamic Range (HDR) images contain a wider range of brightness levels than standard images. However, displaying HDR content on standard screens requires tone mapping techniques to compress the dynamic range while preserving perceptual details. For more information you can check Szeliski's book [5, Chapter 10.2]
  Example of dataset: This dataset has several HDR scenes [1] with a computed HDR image.
  Alternative methods: see OpenCV tutorial [4] for a simple example on how to create HDR image from a set of input images of the same scene taken at different exposure level. Check the Tonemap implementation to generate a tonemapped version of the hdr file automatically.

  (c) **Color to grayscale conversion**
  Converting a color image to grayscale is not a trivial task when trying to preserve perceptual details. A naive conversion (*e.g.* the average of the 3 channels) may lose important luminance and contrast relationships, making objects indistinguishable. The challenge is to retain as much structural and contrast information as possible while discarding color information in a meaningful way.
  Example of dataset: Color250 [3] has pairs of images in color and grayscale and some code for the evaluation of the quality.
  Alternative methods: see cvtColor in OpenCV or rgb2gray in Matlab and the code available in [3]

References

[1] Nima Khademi Kalantari and Ravi Ramamoorthi. "Deep High Dynamic Range Imaging of Dynamic Scenes". In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2017)* 36.4 (2017). URL: https://cseweb.ucsd.edu/~viscomp/projects/SIG17HDR/.

[2] Øyvind Kolås, Ivar Farup, and Alessandro Rizzi. "Spatio-Temporal Retinex-Inspired Envelope with Stochastic Sampling: A Framework for Spatial Color Algorithms". In: *Journal of Imaging Science and Technology* 55.4 (July 2011), 40503–1–40503–10. ISSN: 1943-3522. DOI: 10.2352/j.imagingsci.technol.2011.55.4.040503. URL: https://ivarfa.folk.ntnu.no/publications/journal/Kolaas_11_jist.pdf.

[3] Cewu Lu, Li Xu, and Jiaya Jia. "Contrast Preserving Decolorization with Perception-Based Quality Metrics". In: *International Journal of Computer Vision* 110.2 (July 2014), pp. 222–239. ISSN: 1573-1405. DOI: 10.1007/s11263-014-0732-6. URL: https://www.cse.cuhk.edu.hk/~leojia/projects/color2gray/.

[4] OpenCV. *High Dynamic Range Imaging*. URL: https://docs.opencv.org/4.2.0/d3/db7/tutorial_hdr_imaging.html.

[5] Richard Szeliski. *Computer Vision: Algorithms and Applications*. 1st. Berlin, Heidelberg: Springer-Verlag, 2010. ISBN: 1848829345. URL: https://szeliski.org/Book/drafts/SzeliskiBook_20100903_draft.pdf.

[6] The GIMP Development Team. *GIMP*. Version 3.0. June 12, 2019. URL: https://www.gimp.org.

[7] Chen Wei et al. "Deep Retinex Decomposition for Low-Light Enhancement". In: *British Machine Vision Conference*. 2018. URL: https://daooshee.github.io/BMVC2018website/.