STRESS

Spatio-Temporal Retinex-Inspired Envelope with Stochastic Sampling: A Framework for Spatial Color Algorithms

Pas de STRESS. Il y a Point S.

Idée de base

Principal objectif du papier :

Calculer, pour chaque pixel, une référence locale claire et une référence locale sombre qui seront utilisées pour effectuer différents traitements sur l'image.

Formellement, étant donné une image I, on veut trouver deux enveloppes Emax et Emin de même taille que I, telles que ∀i Emin(i) ≤ I(i) ≤ Emax(i).

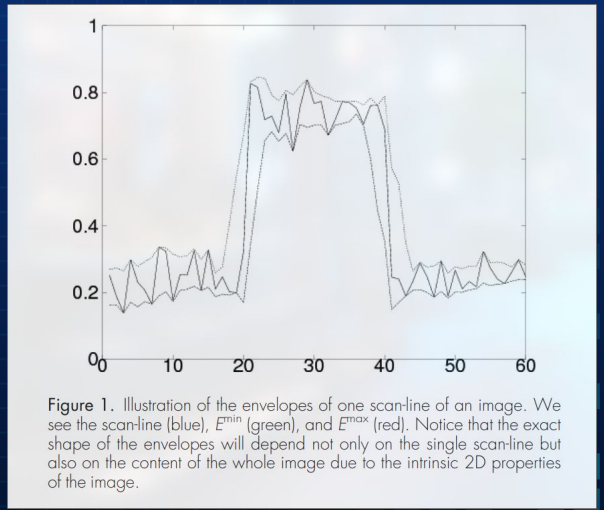


Figure 1. Illustration of the envelopes of one scan-line of an image. We see the scan-line (blue), E^{min} (green), and E^{max} (red). Notice that the exact shape of the envelopes will depend not only on the single scan-line but also on the content of the whole image due to the intrinsic 2D properties of the image.

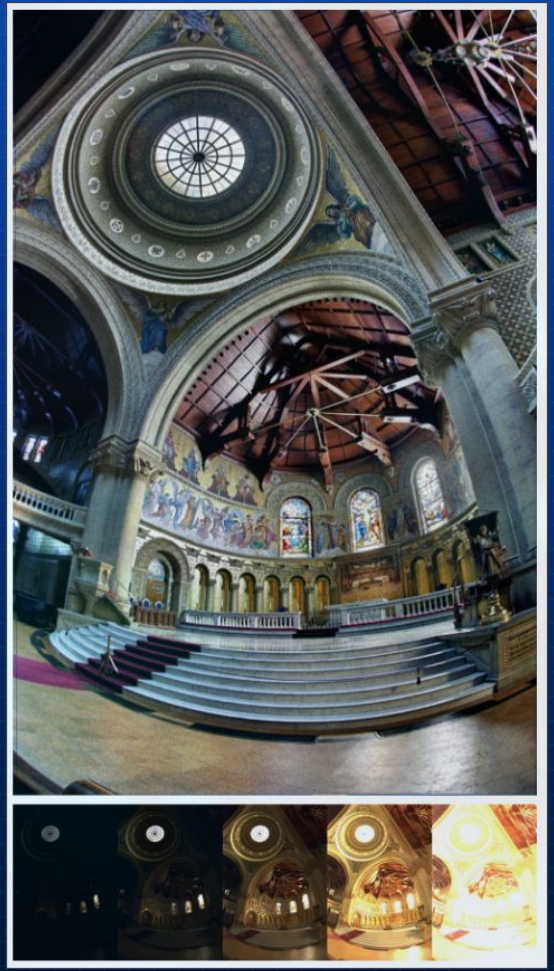
Applications

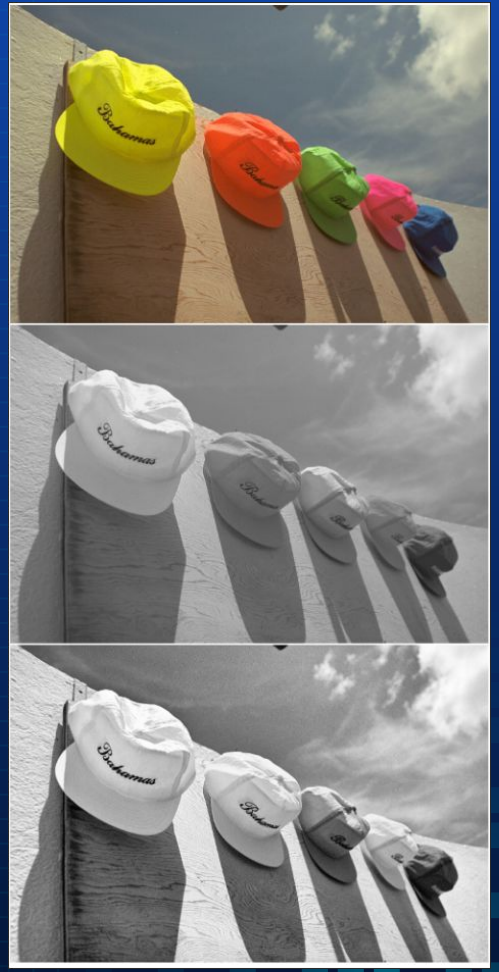
Amélioration locale du contraste des images en niveaux de gris / couleurs.

Applications

Rendu d'images HDR

Conversion couleurs -> niveaux de gris





Application choisie pour l’évaluation

Amélioration locale du contraste des images en niveaux de gris / couleurs.

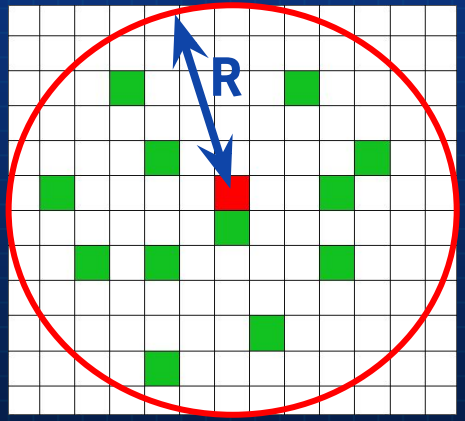
Principe : Calculer la position relative de chaque pixel dans l’enveloppe

Formulation mathématique :

$$p_{\text{stress}} = \frac{p_0 - E_{\min}}{E_{\max} - E_{\min}}$$

Méthode

1 itération



M = 12

Valeurs min/max

$$s_i^{\max} = \max_{j \in \{0,\ldots,M\}} p_j,$$

$$s_i^{\min} = \min_{j \in \{0,\ldots,M\}} p_j.$$

Amplitude et valeur relative

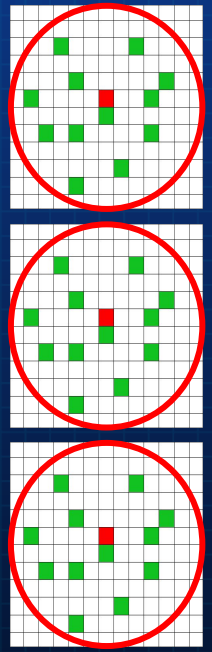
$$r_i = s_i^{\max} - s_i^{\min},$$

$$v_i = \begin{cases} 1/2 & \text{if } r_i = 0, \\ (p_0 - s_i^{\min})/r_i & \text{else.} \end{cases}$$

$$s_i^{\max} \leq p_0 \leq s_i^{\max}$$

Méthode

N itérations (pour le même pixel !!!)



$$r_i = s_i^{\max} - s_i^{\min},$$
$$v_i = \begin{cases} 1/2 & \text{if } r_i = 0, \\ (p_0 - s_i^{\min})/r_i & \text{else.} \end{cases}$$

$$r_i = s_i^{\max} - s_i^{\min},$$
$$v_i = \begin{cases} 1/2 & \text{if } r_i = 0, \\ (p_0 - s_i^{\min})/r_i & \text{else.} \end{cases}$$

$$r_i = s_i^{\max} - s_i^{\min},$$
$$v_i = \begin{cases} 1/2 & \text{if } r_i = 0, \\ (p_0 - s_i^{\min})/r_i & \text{else.} \end{cases}$$

Moyenne amplitude et valeur relative

$$\bar{r} = \frac{1}{N}\sum_{i=1}^{N} r_i,$$

$$\bar{v} = \frac{1}{N}\sum_{i=1}^{N} v_i.$$

Calcul des enveloppes au pixel courant

$$E^{\min} = p_0 - \bar{v}\bar{r},$$

$$E^{\max} = p_0 + (1 - \bar{v})\bar{r} = E^{\min} + \bar{r}.$$

Implémentation

→ **C++, OpenMP, OpenCV**

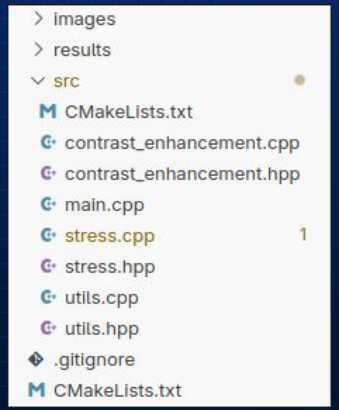
→ **main.cpp :** parse ligne de commande et lance les algos

→ **stress.cpp :** implémente stressGray et stressRGB

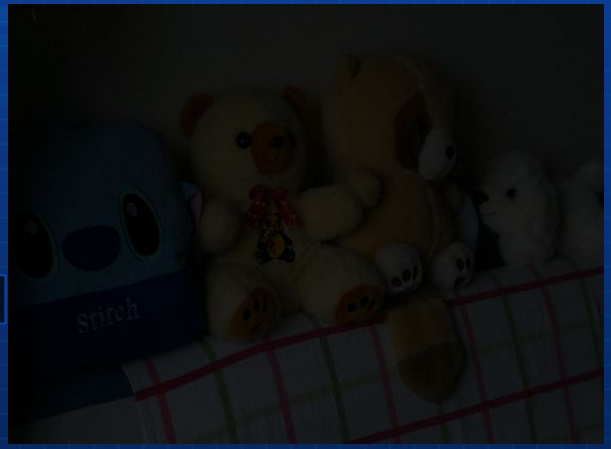
→ **contrast_enhancement.cpp :** implémente l'amélioration de contraste et l'évaluation associée

→ **utils.cpp :** PSNR et SSIM

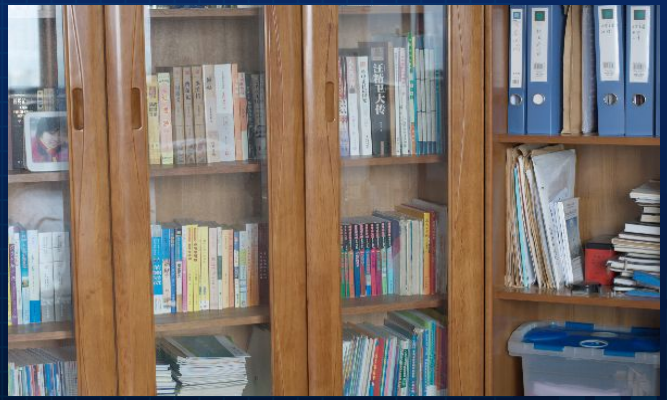
```
Usage: stress <algorithm> [options] <input>
Algorithms:
  hdr                   High Dynamic Range Rendering with STRESS
  contrast              Local Contrast Enhancement with STRESS
  clahe                 Contrast Limited Adaptive Histogram Equalization
Input:
  <input>               Input image/folder path. Depends on the chosen algorithm.
Options:
  -h, --help            Show this help message
  -N <int>              Set N (default: 100)
  -M <int>              Set M (default: 20)
  -R <int>              Set R (default: 300)
  -w, --show            Show the output image
  -e, --eval <str>      Evaluate the algorithm by comparing to the given groundtruth
  -t, --time            Show the time taken by the algorithm
  -p, --parallel        Use parallel processing
  -s, --save <str>      Save the output at the given path
```



Le dataset



Eclairage faible



Eclairage normal

Résultats de notre implémentation

Image de départ

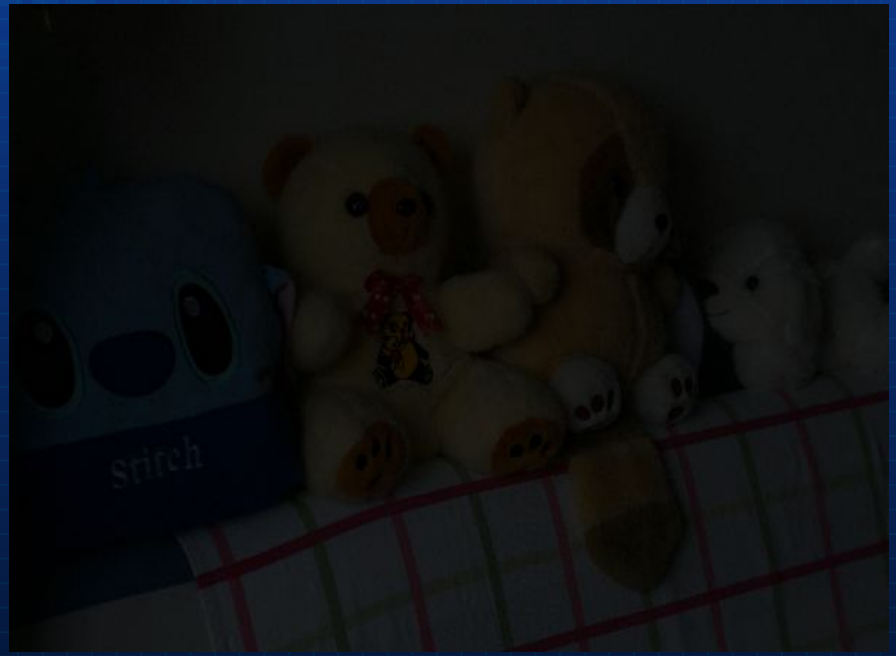
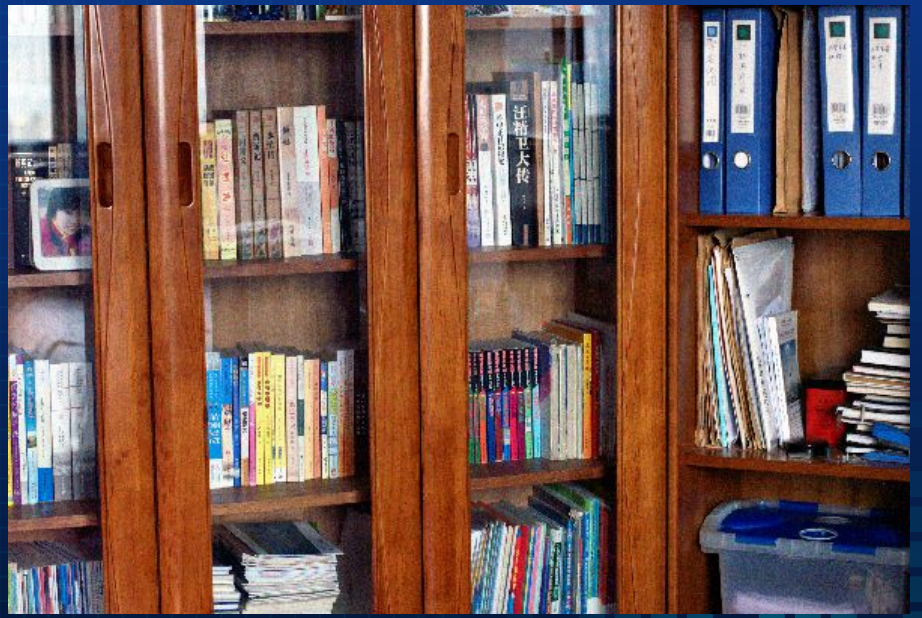


Image obtenue avec N = 20, M = 20, R = 721

Résultats de notre implémentation

Image de départ

Image obtenue avec N = 20, M = 20, R = 721



Limitations

complexité temporelle en O(NMn)

N = 20, M = 20, R = 721 → N = 100, M = 20, R = 721

Avec OpenMP : 12 secondes
(Sans OpenMP : 161 secondes)

Avec OpenMP : 64 secondes

Métriques utilisées

PSNR :

- Basée sur une différence pixels à pixels
- + Utile pour mesurer la proximité de l'image résultat à la vérité terrain
- - Ne prend pas en compte la qualité visuelle / structurelle du résultat

SSIM :

- Basée sur la similarité de structure dans l'image
- + Prend en compte les changements de structure
- + Utile pour mesurer la qualité visuelle du résultat
- - Proximité avec la vérité terrain moins importante

Évaluations de notre implémentation et celle de GIMP

Vérité terrain

Résultat

Par rapport à GIMP :
PSNR: 16.4851 dB
SSIM moyen : 0.834098
SSIM_L : 0.848045

Par rapport à la vérité :
PSNR: 18.1493 dB
SSIM moyen : 0.366973
SSIM_L : 0.488504

GIMP (N = 20, M = 100, R = 721)

Par rapport à la vérité :
PSNR: 18.3449 dB
SSIM moyen : 0.444489
SSIM_L : 0.597955

Évaluations de notre implémentation et celle de GIMP

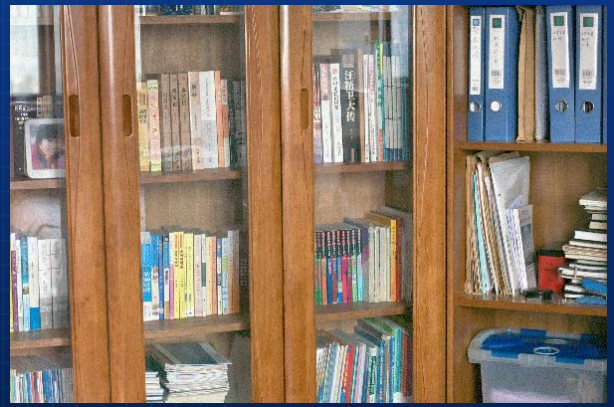
Vérité terrain

Résultat

Par rapport à GIMP :
PSNR: 15.698 dB
SSIM moyen : 0.807155
SSIM_L : 0.836443

Par rapport à la vérité :
PSNR: 16.7185 dB
SSIM moyen : 0.685101
SSIM_L : 0.773171

GIMP (N = 20, M = 100, R = 721)



Par rapport à la vérité :
PSNR: 22.8417 dB
SSIM moyen : 0.752151
SSIM_L : 0.846647

Influence des paramètres : R

(N = 20, M = 20)

R = 50

R = 721

Mettre R égal à la diagonale de l'image pour échantillonner sur toute l'image et ne pas avoir d'artefacts

Influence des paramètres : M

(R = 721, N = 20)

M = 20

Par rapport à la vérité :
PSNR: 17.2169 dB
SSIM moyen : 0.318382
SSIM_L : 0.430367

M = 100

Par rapport à la vérité :
PSNR: 18.1493 dB
SSIM moyen : 0.366973
SSIM_L : 0.488504

Un M plus grand augmente le contraste plus uniformément

Influence des paramètres : N

(R = 721, M = 20)

N = 3

Par rapport à la vérité :
PSNR: 17.2829 dB
SSIM moyen : 0.51246
SSIM_L : 0.641649

N = 20

Par rapport à la vérité :
PSNR: 17.7667 dB
SSIM moyen : 0.778448
SSIM_L : 0.847743

Un N plus grand diminue le bruit

Méthode alternative : CLAHE

Image de départ :

Image après CLAHE :

Par rapport à la vérité :
PSNR: 11.4449 dB
SSIM moyen : 0.622992
SSIM_L : 0.62083

Perte des couleurs mais peut être utile pour un pré-traitement de l'image

Conclusion

- Parvient à rehausser le contraste des images faiblement éclairées/contrastées en limitant le bruit, mais il en reste une partie.
- Si N ou M est trop grand, l'algorithme est lent, surtout sans parallélisation.
- Améliorations possibles : Post-traitement sur le résultat pour supprimer le bruit restant (méthodes variationnelles ou du deep learning)
- Autres applications non implémentées : rendu d'images HDR, conversion d'images couleur en niveaux de gris etc...

Références

Pour le calcul du PSNR et SSIM : https://docs.opencv.org/4.x/dd/d3d/tutorial_gpu_basics_similarity.html

Annexes

```
34      #pragma omp for collapse(2)
35      for (int i = 0; i < n_rows; i++) {
36          for (int j = 0; j < n_cols; j++) {
37              int x = input.at<uchar>(i0: i, i1: j);
38              double range_mean = 0.0;
39              double relative_value_mean = 0.0;
40              for (int it = 0; it < N; it++) {
41                  vector<int> rand_neighbors(n: M+1,value: 0);
42                  for (int k = 0; k < M; k++) {
43                      bool valid_rand_pixel = false;
44                      while (!valid_rand_pixel) {
45                          int d = uniform_dist_R(&urng: e1);
46                          double teta = uniform_dist_teta(&urng: e1);
47                          double abs_pol, ord_pol;
48                          polarToCartesian(r: d,teta,&x: abs_pol,&y: ord_pol);
49                          int abs = i + round(x: abs_pol);
50                          int ord = j + round(x: ord_pol);
51                          if (abs >= 0 && abs < n_rows && ord >= 0 && ord < n_cols && abs != i && ord != j) {
52                              rand_neighbors[k] = input.at<uchar>(i0: abs, i1: ord);
53                              valid_rand_pixel = true;
54                          }
55                      }
56                  }
57                  rand_neighbors[M] = x;
58                  int min = *min_element(first: rand_neighbors.begin(), last: rand_neighbors.end());
59                  int max = *max_element(first: rand_neighbors.begin(), last: rand_neighbors.end());
60                  int r = max - min;
61                  range_mean += r;
62                  if (r == 0) {
63                      relative_value_mean += 1.0/2.0;
64                  } else {
65                      relative_value_mean += (double)(x - min) / (double)r;
66                  }
67              }
68              range_mean /= N;
69              relative_value_mean /= N;
70              Emin.at<double>(i0: i, i1: j) = (double)x - range_mean * relative_value_mean;
71              Emax.at<double>(i0: i, i1: j) = (double)x + range_mean * (1 - relative_value_mean);
72          }
73      }
```