LINÉARISATION DE L'IMAGE : REMETTRE DROIT CE QUI EST COURBÉ

PRÉFACE

La linéarisation consiste à préparer les données en amont pour les calculs et transformations nécessaires qui interviendront par la suite - notamment la conversion XYZ - qui est une transformation linéaire - qui possède des calculs matriciels nécessitant absolument des données linéaires en entrée.

QU'EST-CE QUE LINÉAIRE ET NON-LINÉAIRE ?

Linéaire ? Non-linéaire ? Quelle est la différence ?

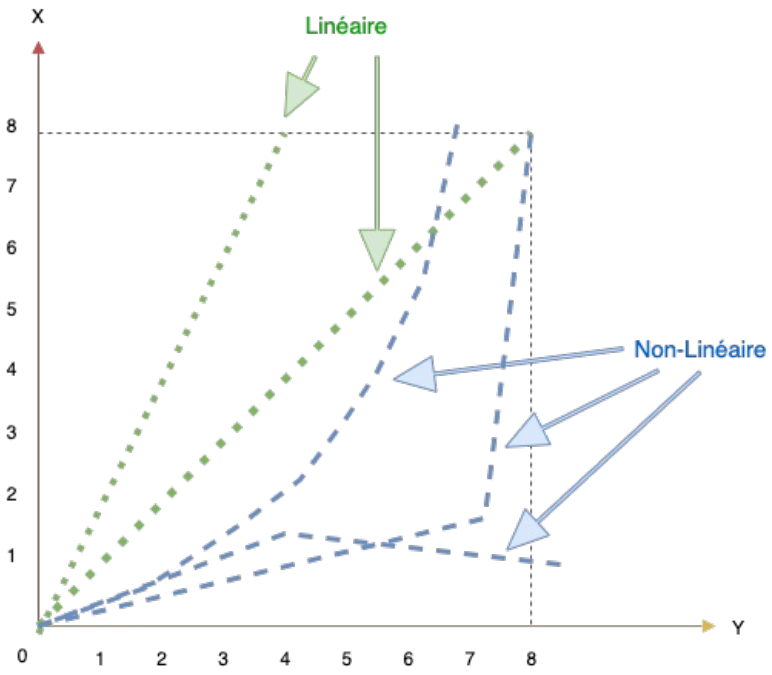
Si vous n'êtes pas familier avec le principe de linéarité en mathématique, c'est (relativement) simple : un système linéaire est la proportionnalité de chaque valeur.

Un exemple de système linéaire : vous achetez une baguette, elle vous coûte 1€ [1]. Vous en achetez 2, 2€, vous en achetez 10, 10€, et ainsi de suite. C'est un système linéaire.

Dans un système non-linéaire, c'est exactement l'inverse : la proportionnalité entre chaque valeur peut ne pas exister. En gros, vous n'avez pas forcément une suite logique.

Un exemple de système non-linéaire : le boulanger qui applique une certaine forme de réduction sur la quantité de pains achetés. Il décide qu'à chaque dizaine de baguettes achetées, il appliquera une réduction plus ou moins conséquente sur la dizaine achetée : Disons qu'après 10 baguettes, les baguettes suivantes ne vous coûteront plus que 0.9. Au-delà de 20, il appliquera une nouvelle réduction, et ainsi de suite. On arrive sur le contraire d'une linéarité du prix, on arrivera sur une non-linéarité car la valeur suivante peut ne pas être dans la continuité des précédentes.

Quelques exemples visuels entre linéarité et non-linéarité :



Dans le monde de l'image, nous allons avoir affaire à des systèmes linéaires et des systèmes non-linéaires, principalement logarithmique, ne vous enfuyez pas, nous allons essayer d'être simple ! :)

Mais pourquoi travailler avec les deux ? parce que dans un système informatique, il est préférable de travailler

sur des données linéaires, alors que dans notre monde réel, nous sommes plutôt sur des systèmes non-linéaires, il faut donc arriver à passer de l'un à l'autre sans trop de problème.

PRINCIPE DE LA LINÉARISATION

Avant de partir sur des calculs complexes (comme la conversion XYZ), nous devons passer par linéariser chaque valeur colorimétrique. Les ordinateurs (et les matrices) préfèrent :)

Selon la documentation **SMPTE RP-431-2-2011 - D-Cinema Quality - Reference Projector and Environment**, la linéarisation consiste en deux étapes [2]:

- Enlever le gamma de l'image d'input (sauf si cela a déjà été fait et donc que le gamma est à 1.0, donc déjà linéarisé)
- Placer les valeurs colorimétriques dans une plage de nombres flottants compris entre 0.0 et 1.0 - que je nomme plage [0..1] dans le reste de la documentation. Même s'il est possible de travailler sans, la norme recommande donc de travailler dans une plage [0..1].

Concrètement, la linéarisation consiste à enlever le gamma - ou le remettre au neutre à 1.0 - avant de passer à la conversion XYZ qui demande absolument un input linéaire pour travailler correctement.

Si vous n'appliquez pas une linéarisation de votre input, la conversion XYZ pourra quand même travailler dessus (ce sont de simples calculs matriciels), mais le résultat ne sera pas terrible :)

EXEMPLE RAPIDE DE LINÉARISATION

Voici un exemple rapide de linéarisation R'G'B' [3] vers un RGB linéaire dans la plage [0..1].
Concepts et exemples que nous verrons plus en profondeur dans les chapitres suivants notamment dans Gamma et Bitdepth.

Dans notre exemple, notre gamma input est de 2.6, il faudra l'adapter au gamma d'input.
Par exemple, pour un sRGB, le gamma sera à 2.2

Nous supposons que nous travaillons dans un bitdepth de 12 bits par composant avec un gamma d'input à 2.6 :

```
# L'équation SMPTE :
# Channel = (Channel/4095)^2.6

# Notre rouge est 0x7ff (12 bits)
>>> Red = 0x7FF

# On passe notre valeur 0x7FF dans une plage [0..1]
# Pourquoi diviser par 4095 ? On le verra dans le chapitre Gamma et Bitdepth
# En résumé, pour passer de la valeur 0x7FF à une valeur compris entre 0 et 1,
# nous devons diviser 0x7FF par la valeur maximale du bitdepth,
# Ici pour 12 bits, c'est 0xFFF (les douzes bits à 1)
>>> Red = (Red/4095)   # 4095 == 0xFFF
>>> print(Red)
0.4998778998778999
# Notre Red est maintenant dans une plage [0..1]

# Maintenant, on va enlever le gamma qui est non-linéaire.
# La fonction de transfert "gamma" utilise la loi de puissance (power law) pour son calcul
# On enleve son gamma à l'aide de pow() :
>>> pow(0.4998778998778999, 2.6)
0.16483378645422764
```

Notre valeur Red est maintenant linéaire et possède la valeur de 0.164833(..). Il suffit d'appliquer ce principe sur les autres composants Green et Blue pour avoir une couleur RGB totalement linéarisée.

A partir des valeurs linéarisées, nous pouvons passer à notre conversion XYZ qui nécessite un input linéaire.

NOTES

1. ou n'importe quel prix, j'ai pris 1€ pour faire simple :) ↩
2. La linéarisation : ↩
    - *« The digital files were linearized (applying a gamma of 2.6), then a 3x3 matrix was applied to convert RGB to XYZ, followed by application of the (1)/2.6 gamma function. The finished color-corrected files were stored as 12-bit X'Y'Z' data in 16-bit TIFF files. »* -- Color and Mastering for Digital Cinema (Glenn Kennel)
    - *« Color conversion from R'G'B' to X'Y'Z' requires a three-step process which involves linearizing the color-corrected R'G'B' signals (by applying a 2.6 gamma function), followed by their passage through a linear 3x3 transform matrix. The resultant linearized and coded XYZ signals are then given an inverse 2.6 gamma transfer characteristic whose output is quantized to 12 bits. »* -- SMPTE RP-431-2-2011 - DCinema Quality Reference Projector and Environement - Chapitre « Color Conversion to XYZ »
3. `R'G'B` == `RGB + Gamma` ↩