

# IMN430

## Chapitre 2

### Techniques de base en visualisation

Olivier Godin & Michaël Bernier

Université de Sherbrooke

23 janvier 2017

# Plan de la présentation

- 1 Introduction
- 2 Visualisation de données scalaires
- 3 Références

# Introduction

1 Introduction

2 Visualisation de données scalaires

3 Références

# Introduction

Dans ce chapitre-ci, on s'attarde aux méthodes de base permettant **d'exprimer les données sous forme visuelle.**

Tel que mentionné dans le chapitre d'introduction, les **algorithmes de mappage** qui mettent en correspondance des données avec des primitives graphiques sont au cœur de la visualisation.

Ces algorithmes seront catégorisés **selon le type de donnée qu'ils prennent en entrée.**

# Introduction

Ici, une attention toute particulière sera portée aux algorithmes de mappage de **données scalaires**.

Par la suite, une introduction aux représentations de **données vectorielles** sera aussi présentée, mais ce sujet sera approfondi au chapitre 5.

# Visualisation de données scalaires

## 1 Introduction

## 2 Visualisation de données scalaires

- Placage de Couleurs
- Cartes de hauteur
- Contours

## 3 Références

# Placage de Couleurs

## 1 Introduction

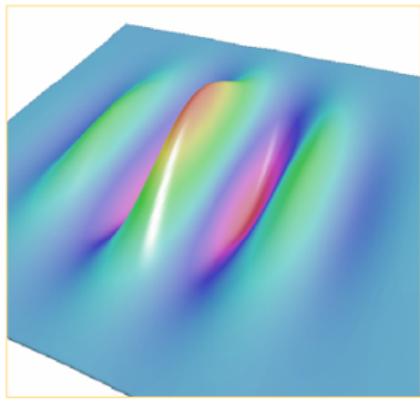
## 2 Visualisation de données scalaires

- Placage de Couleurs
- Cartes de hauteur
- Contours

## 3 Références

# Placage de couleurs

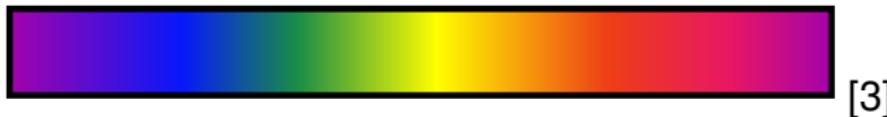
Le **placage de couleurs** est une technique commune de visualisation qui consiste à **associer des données scalaires à des couleurs**.



# Placage de couleurs

Cette association entre les données scalaires et les couleurs se fait à l'aide d'une **table de correspondance** où les valeurs scalaires font office d'indices.

Chaque entrée dans la table contient une couleur.



# Placage de couleurs

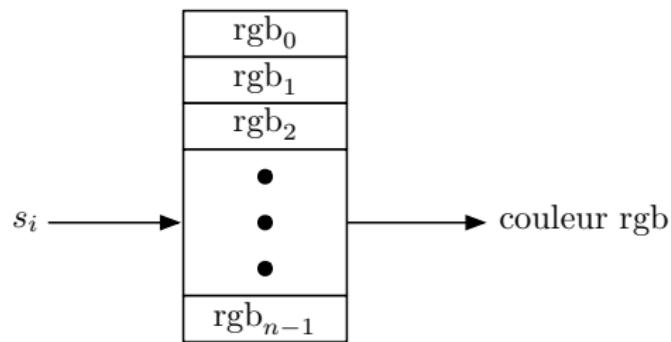
En plus de la table de correspondance, on doit **connaître l'étendue des données scalaires** (valeur minimale et valeur maximale).

Ce faisant, pour chaque scalaire  $s_i$ , on peut **évaluer l'indice  $i$  correspondant dans la table**.

$$s_i < \min \Rightarrow i = 0$$

$$s_i > \max \Rightarrow i = n - 1$$

$$i = n \left( \frac{s_i - \min}{\max - \min} \right)$$



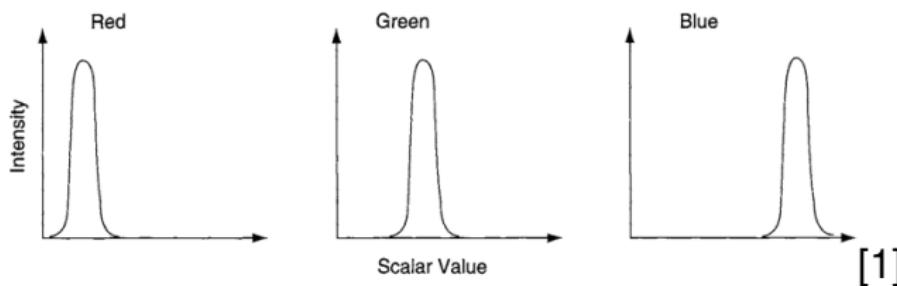
# Placage de couleurs

En pratique, les valeurs min et max peuvent être **déterminées automatiquement** (en parcourant les données à afficher) ou encore **spécifiées par l'utilisateur**.

Cette dernière option est particulièrement utile **lorsqu'on ne dispose pas de l'ensemble des données** à afficher dès le départ, comme dans le cas des données variant dans le temps.

# Placage de couleurs

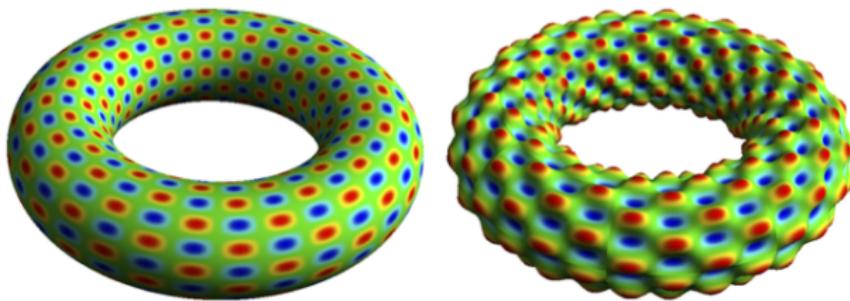
On peut généraliser le principe de la table de correspondance à celui de **fondation de transfert**. Une fonction de transfert est une **expression analytique** qui met en relation les valeurs scalaires et les couleurs.



Une table de correspondance peut être **obtenue par échantillonnage d'une fonction de transfert**.

# Placage de couleurs

À la base, le placage de couleur est une technique de visualisation 1D qui associe à un scalaire une unique couleur. Toutefois, **l'affichage des couleurs n'est pas limité à une dimension.**



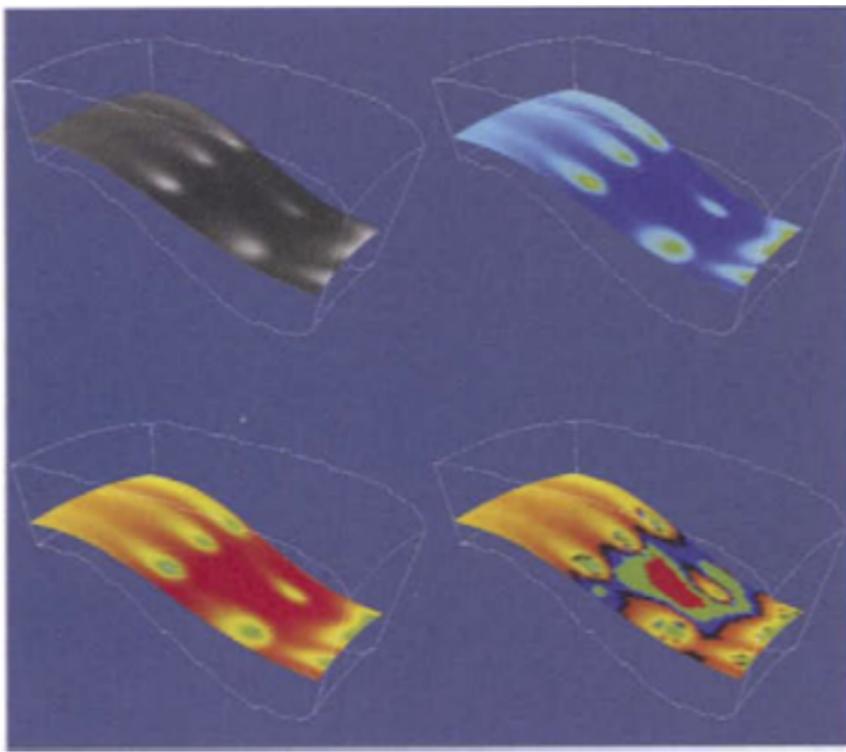
[4]

# Placage de couleurs

Le succès d'un placage de couleurs repose sur **la conception de la table de correspondance ou de la fonction de transfert.**

Un bon placage devrait **accentuer les caractéristiques d'intérêt** dans les données, tout en minimisant l'impact visuel des détails de moindre importance.

# Placage de couleurs



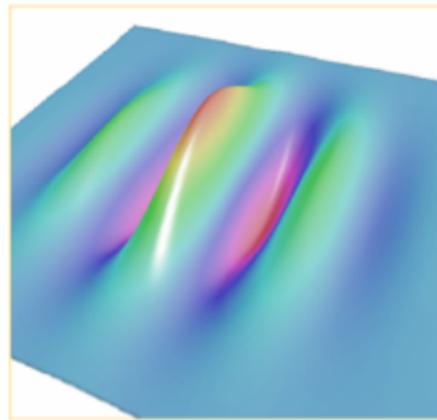
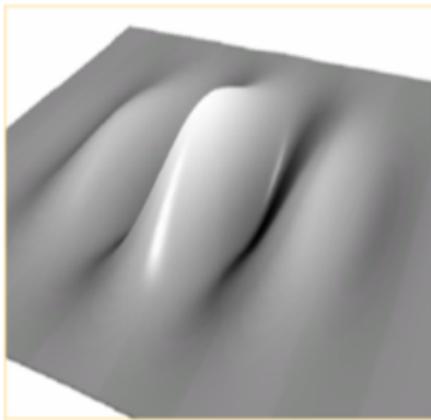
[1]

# Élaboration de la fonction de transfert

Un objectif courant en visualisation de données scalaires par placage de couleurs est d'être en mesure, en regardant une couleur, d'**estimer la valeur du scalaire qui lui est associée**.

En pratique, on atteint cet objectif en donnant une **légende faisant correspondre les couleurs à des valeurs numériques**.

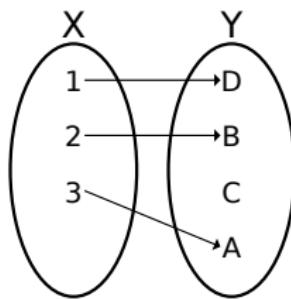
# Élaboration de la fonction de transfert



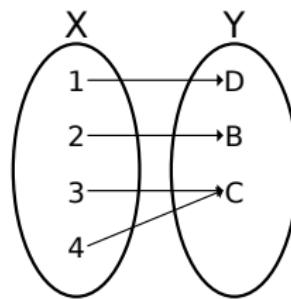
# Élaboration de la fonction de transfert

Pour que la correspondance inverse (Couleur  $\rightarrow$  Scalaire) soit possible, la fonction de transfert doit être **injective**.

Cela signifie que chaque valeur dans l'intervalle  $[min, max]$  doit être **associée à une unique couleur**.



Injectif

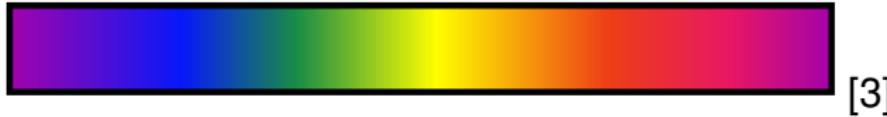


Pas injectif

# Élaboration de la fonction de transfert

Il n'est toutefois pas suffisant que les couleurs aient des valeurs RGB différentes. Pour qu'un observateur puisse faire la correspondance inverse, **les couleurs doivent être perceptuellement différentes.**

Une fonction de transfert souvent utilisée est l'*arc-en-ciel*.

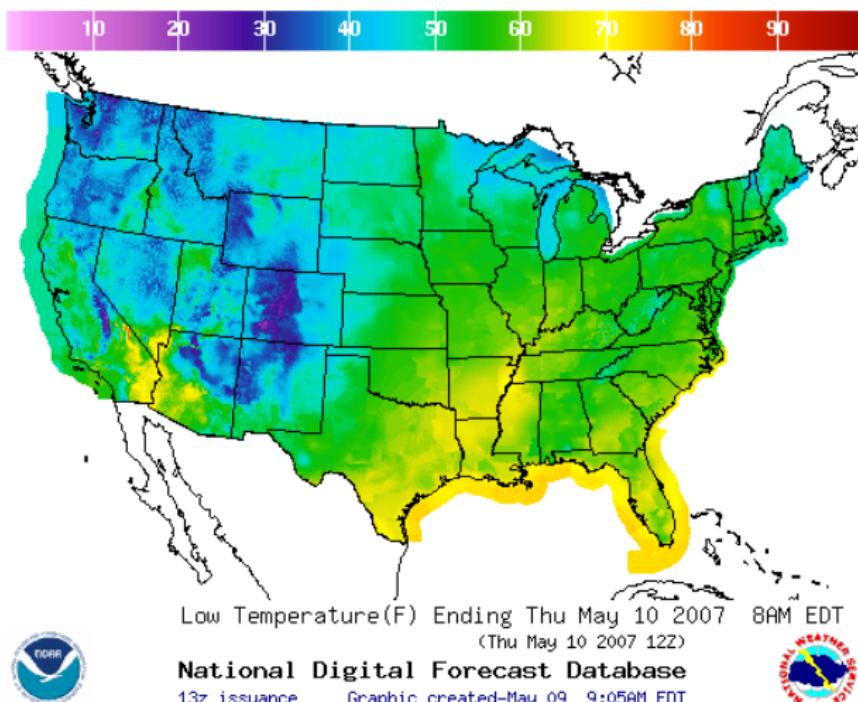


# Élaboration de la fonction de transfert

Elle est particulièrement présente en ingénierie et en météorologie.

Sa popularité est due au fait qu'elle repose sur l'intuition que le bleu, une couleur froide, sera **associé à des valeurs plus basses**, tandis que le rouge, une couleur chaude, sera **associée à des valeurs plus élevées**.

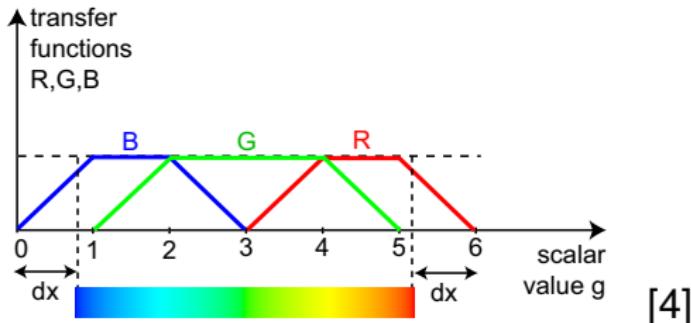
# Élaboration de la fonction de transfert



# Élaboration de la fonction de transfert

Cette fonction de transfert est définie par **trois sous fonctions**, chacune associée à une composante R, G ou B.

Ces sous-fonctions sont de forme trapézoïdale et présentent un important recouvrement, ce qui assure **une douceur dans les transitions**.

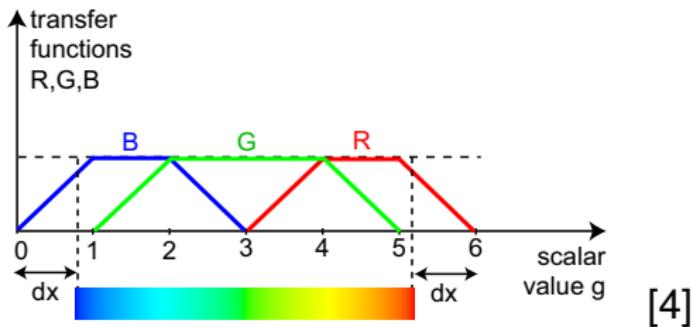


[4]

# Élaboration de la fonction de transfert

Par contre, elle a plusieurs désavantages **perceptuels** :

- **Luminance non constante (RGB vs HSV)**
- **Ordre perceptuel (vert = neutre, non intuitif)**
- **Linéarité plus difficile**



# Élaboration de la fonction de transfert

En imagerie médicale, on peut utiliser plusieurs fonctions de transfert basées sur la **luminance** en utilisant l'espace de couleur HSV

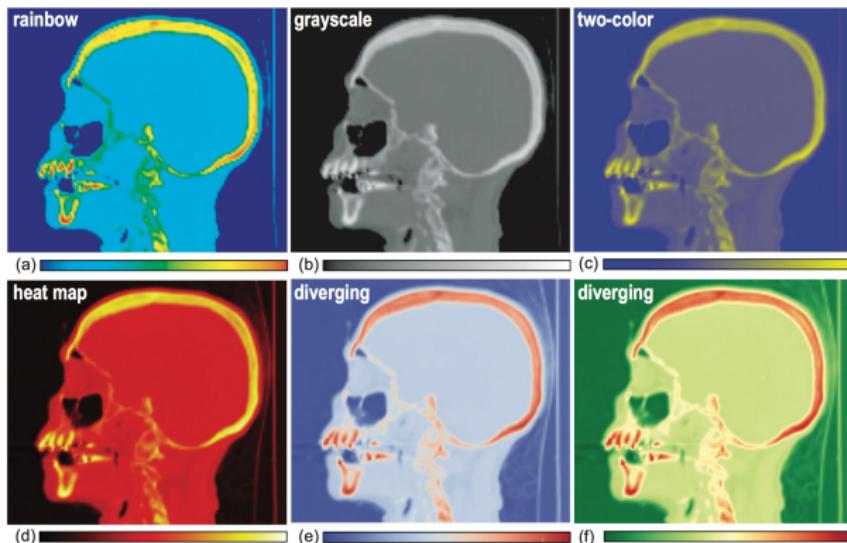


Figure 5.2. Scalar visualization with various colormaps.

[4]

# Élaboration de la fonction de transfert

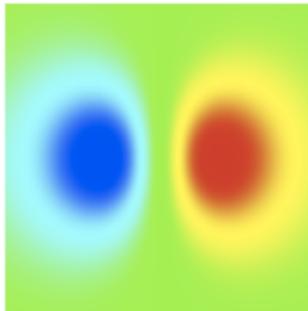
- *Grayscale* : On encode que la luminance
- *two-hue* : On interpole entre deux tons (ex : bleu-jaune) -> Moins de plage perceptuelle (ex : ombrage)
- *Heat map* : On interpole du noir-rouge au jaune-blanc -> Moins de plage perceptuelle (ex : ombrage)
- *Diverging* : On interpole entre couleur 1 (luminance moyenne) - blanc - couleur 2 (luminance moyenne)

# Élaboration de la fonction de transfert

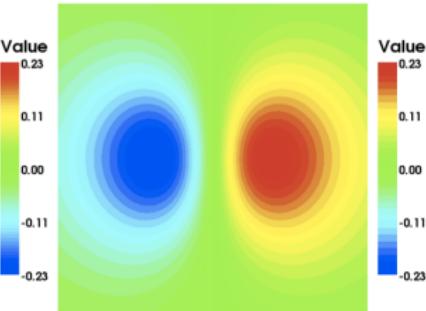
Si on opte pour l'utilisation d'une table de correspondance, il faut **déterminer le nombre de couleurs** qu'elle contiendra.

- Un **trop faible nombre de couleurs** diminue la précision de la représentation.
- Un **trop grand nombre de couleurs** rend la correspondance inverse (Couleur → Scalaire) plus difficile.

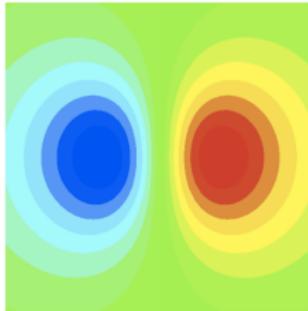
# Élaboration de la fonction de transfert



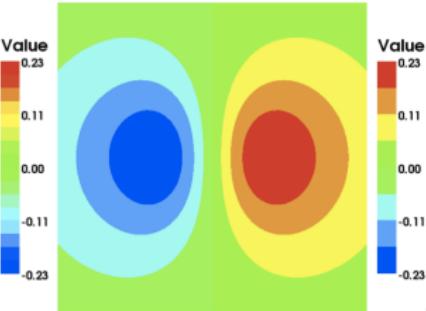
a) 256 colors



b) 32 colors



c) 16 colors



d) 8 colors

[4]

# Élaboration de la fonction de transfert

Ainsi, la conception d'une fonction de transfert ou d'une table de correspondance est une tâche complexe. La notion de **bonne ou mauvaise représentation** dépend énormément du contexte.

Il s'agit d'un domaine où l'**originalité est rarement la bienvenue**. Il est souvent préférable de s'en tenir aux fonctions qui sont déjà connues et acceptées.

# Élaboration de la fonction de transfert

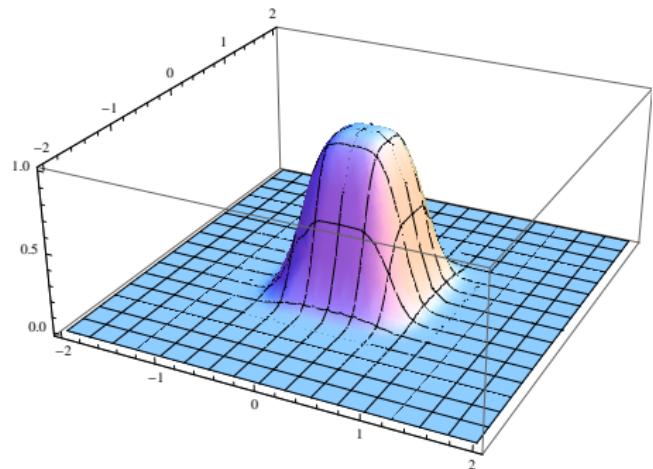
Dans certaines applications, on cherche plutôt à **mettre l'emphasis sur les variations** dans les données que sur les valeurs de celles-ci.

On peut ainsi, en un coup d'oeil, détecter **les régions présentant d'importantes fluctuations** ou encore **celles qui sont constantes**.

# Élaboration de la fonction de transfert

Pour illustrer ce principe, considérons la fonction suivante :

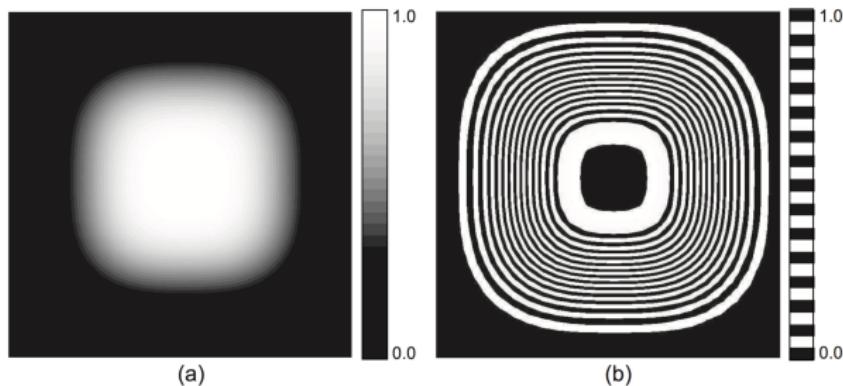
$$f(x, y) = e^{-10(x^4+y^4)}$$



On veut trouver une représentation visuelle qui rend facile la détection des régions **où les changements sont brusques**.

# Élaboration de la fonction de transfert

La représentation la plus logique consiste à assigner une couleur (niveau de gris) **selon la valeur prise par  $f(x, y)$** .

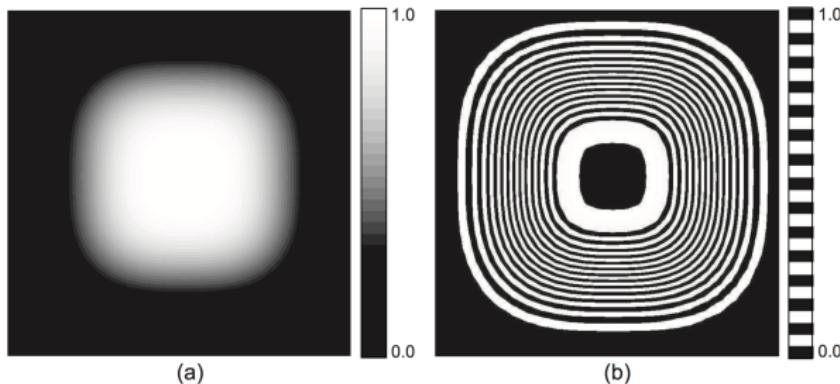


**Figure 5.3.** Visualizing the scalar function  $e^{-10(x^4+y^4)}$  with (a) a luminance colormap and (b) a zebra colormap. The luminance colormap shows absolute values, whereas the zebra colormap emphasizes rapid value variations.

[4]

# Élaboration de la fonction de transfert

Si on utilise plutôt une fonction de transfert basée sur les variations, on obtient



**Figure 5.3.** Visualizing the scalar function  $e^{-10(x^4+y^4)}$  with (a) a luminance colormap and (b) a zebra colormap. The luminance colormap shows absolute values, whereas the zebra colormap emphasizes rapid value variations.

[4]

Ici, de **fines lignes denses** indiquent des **changements brusques**, tandis que des **régions constantes** illustrent une **absence de variation**.

# Cartes de hauteur

## 1 Introduction

## 2 Visualisation de données scalaires

- Placage de Couleurs
- **Cartes de hauteur**
- Contours

## 3 Références

# Cartes de hauteurs

Soit une fonction  $f(x, y)$  ayant comme domaine une surface  $D$  à deux dimensions. La **carte de hauteurs** associée à la fonction  $f$  est définie par l'ensemble de points 3D

$$m(x, y) = (x, y) + f(x, y)\mathbf{n}(x, y) \quad \forall x \in D,$$

où  $\mathbf{n}(x, y)$  est la normale de la surface au point  $(x, y)$ .

Dans leur forme la plus simple, les cartes de hauteur **partent d'une surface plane** où toutes les normales pointent dans la même direction.

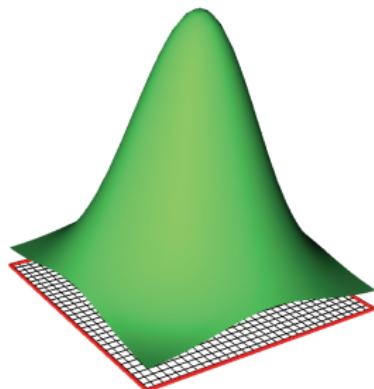
# Cartes de hauteurs

Supposons que l'on dispose d'une fonction  $f : D \rightarrow \mathbb{R}$ . À des fins d'exemple, on posera

$$f(x, y) = e^{-(x^2+y^2)}.$$

**La représentation graphique associée à cette fonction sera l'ensemble des points de la forme**

$$(x, y, f(x, y))$$



[4]

# Cartes de hauteurs

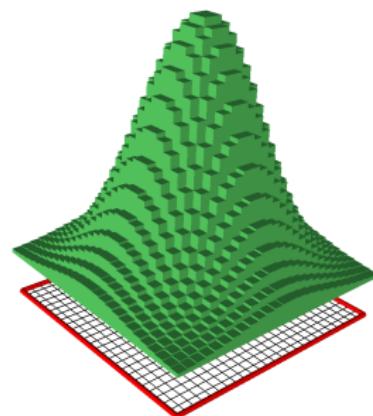
Si l'obtention de ce graphique peut sembler être une tâche simple, de prime abord, **il en est tout autrement en réalité.**

Le principal problème qui se pose est celui de **la continuité des données** : la fonction  $f$  est **définie pour toutes les valeurs sur  $D$** , mais un ordinateur n'est pas en mesure de traiter une infinité de données.

# Cartes de hauteurs

La solution est d'**échantillonner  $f$  en discrétilisant la surface  $D$ .**

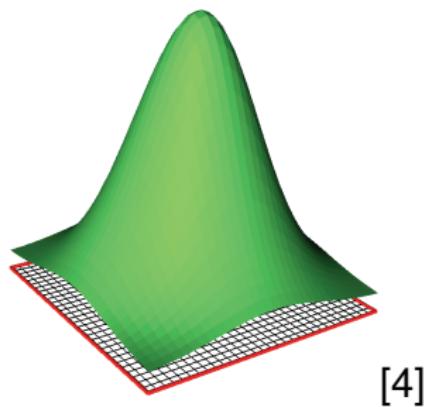
Par exemple, en optant pour une division  $30 \times 30$ , on obtient la **carte de hauteurs** suivante :



[4]

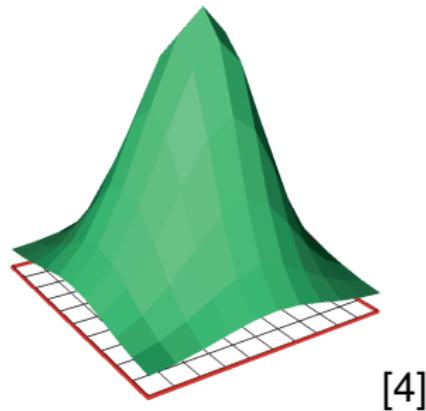
# Cartes de hauteurs

À l'aide de **techniques de triangulation** (section précédente), on peut rendre gracieuse la carte de hauteurs précédente, et ce **sans augmenter la densité des données**.



# Cartes de hauteurs

Qui plus est, en utilisant la triangulation, il n'est peut-être même **pas nécessaire d'utiliser une division si précise**. La figure suivante a été obtenue avec **neuf fois moins de données**.

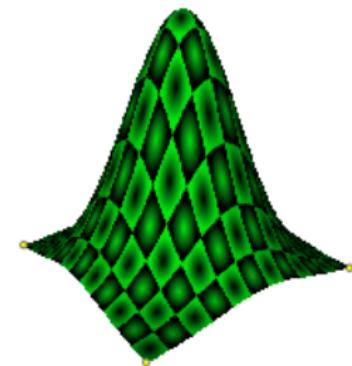


[4]

# Cartes de hauteurs

De plus, rien n'empêche d'**ajouter de l'information à une carte de hauteurs**. On pourrait, d'une part, utiliser la hauteur pour transmettre de l'information, **mais aussi la couleur de la surface affichée**.

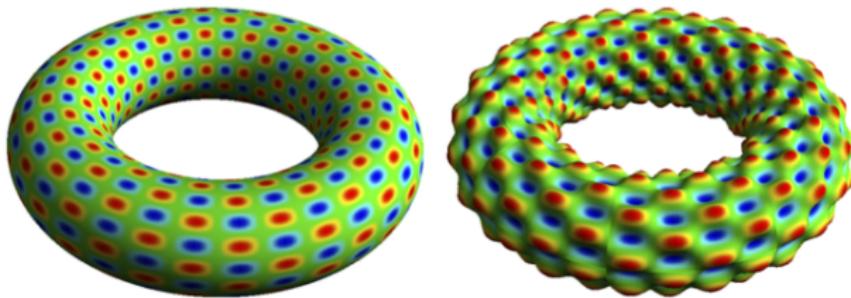
On se trouve ainsi à **exprimer deux quantités scalaires** sur une même représentation graphique.



[4]

# Cartes d'élévation

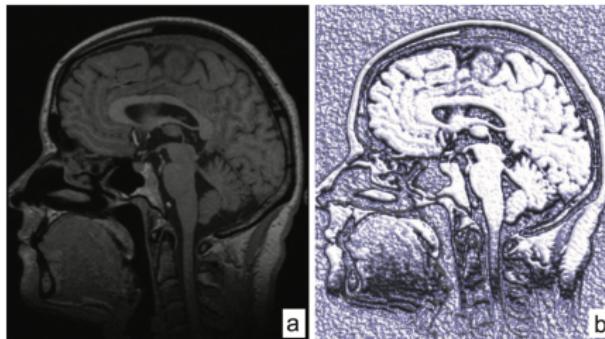
On peut également représenter une carte de hauteur sur une surface non-planaire :



[4]

# Cartes d'élévation

On peut également représenter une carte de hauteur sur une surface non-planaire :



**Figure 5.19.** (a) Grayscale color mapping of scalar dataset. (b) Height plot of the same dataset, emphasizing fine-grained data variations.

[4]

# Contours et volumes

## 1 Introduction

## 2 Visualisation de données scalaires

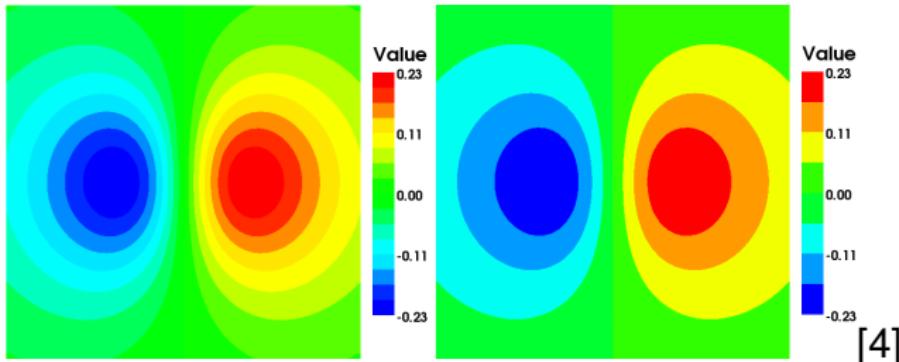
- Placage de Couleurs
- Cartes de hauteur
- Contours

## 3 Références

## Contours et volumes

Dans la première partie de cette section, on a vu comment représenter des données scalaires à l'aide du placage de couleurs.

Lorsqu'il avait été question du **nombre de couleurs à utiliser**, on avait dit qu'un nombre trop faible de celles-ci entraînerait des **transitions brusques entre les intensités**.

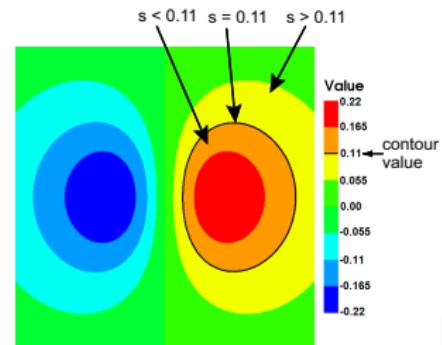


# Contours et volumes

Ces transitions brusques sont à la base d'une technique de visualisation très répandue : les **isocontours**.

Formellement, un tel contour  $C$  est défini comme l'ensemble des points  $p$  dans un domaine  $D$  auxquels sont associés la même valeur scalaire  $f(p) = x$  :

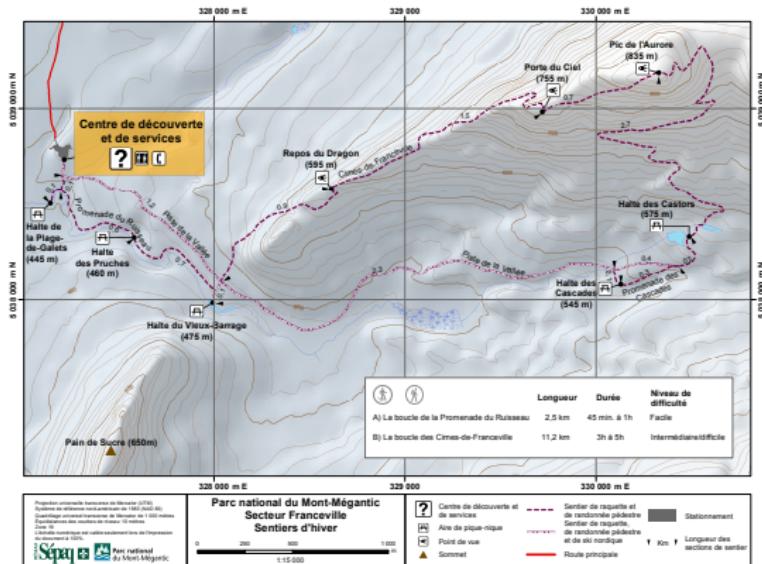
$$C(x) = \{p \in D | f(p) = x\}$$



[4]

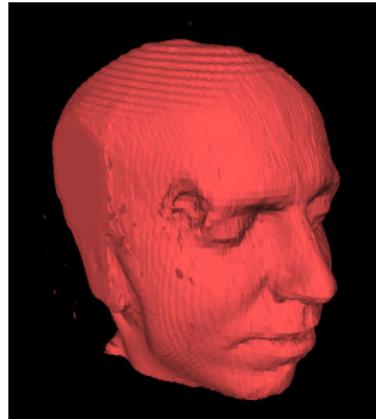
# Contours et volumes

Une application classique des isolignes se fait en **cartographie** : avec les **courbes de niveau** sur une carte géographique, on indique l'ensemble des points possédant **la même altitude**.



## Contours et volumes

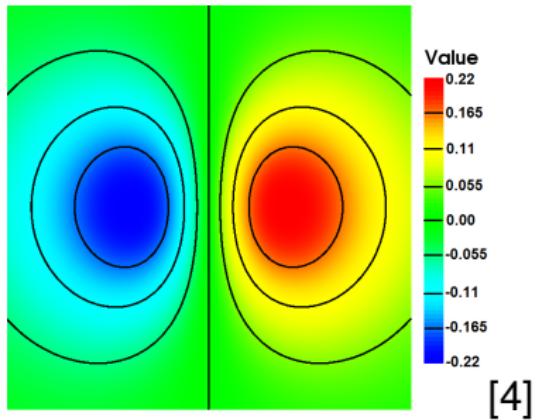
Plus loin dans cette section, on verra aussi une **généralisation de ce principe aux données à trois dimensions** avec les isosurfaces.



Plus spécifiquement, deux techniques seront présentées : les **marching squares** et les **marching cubes**.

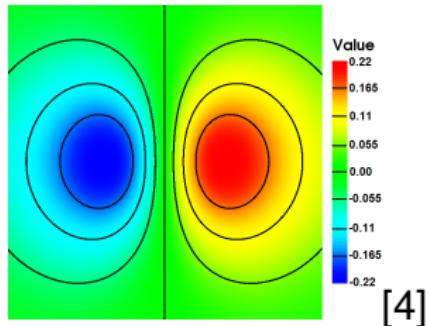
## Contours et volumes

En plus d'indiquer les points où les données sont de même valeur, les contours peuvent **donner de l'information sur la variation des données** : supposons un ensemble de contours associés à des valeurs scalaires qui sont **uniformément espacées** :



On constate aisément que les contours, eux, ne sont pas également espacés dans le domaine spacial.

# Contours et volumes



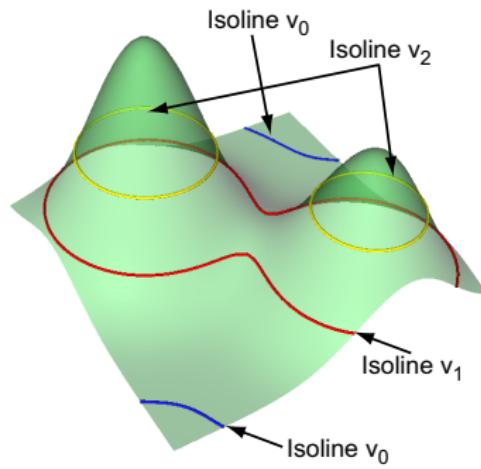
Qu'est-ce que cela signifie ?

Les régions où les contours sont **plus près les uns des autres**, comme c'est le cas au centre de l'image, présentent une **plus grande variation** des valeurs scalaires.

# Propriétés des contours

Les isocontours possèdent plusieurs propriétés utiles.

Pour les illustrer, on utilisera la **carte de hauteurs** associée à une fonction  $z = f(x, y)$ .



[4]

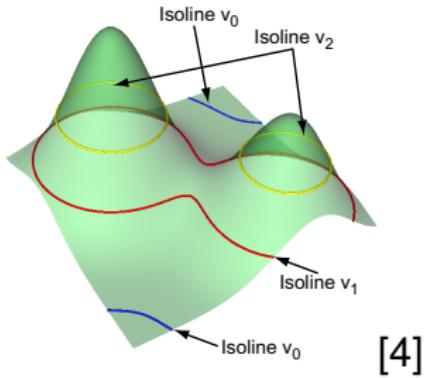
# Propriétés des contours

Sur la figure, trois isocontours ont été dessinés, et **chacun est associé à une valeur scalaire différente** :  $v_0$  (bleu),  $v_1$  (rouge) et  $v_2$  (jaune).

Ces contours correspondent aux **intersection entre les plans horizontaux**  $z = v_i$  et la carte de hauteurs.

# Propriétés des contours

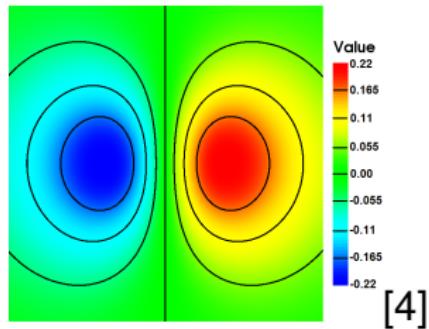
**Propriété # 1 :** un isocontour peut être soit une **courbe fermée** (jaune) ou une **courbe ouverte** (bleu et rouge).



Un contour **ne se termine jamais à l'intérieur du domaine**. Soit la ligne est fermée, soit elle se termine lorsqu'elle atteint la bordure du domaine.

# Propriétés des contours

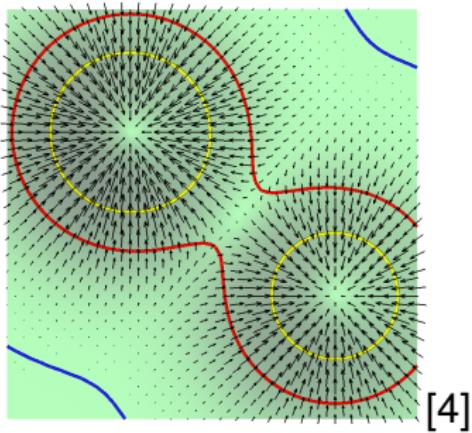
**Propriété # 2 : un isocontour ne peut se croiser lui-même, ni avoir une intersection avec un autre contour.**



Les contours sont plutôt **imbriqués les uns dans les autres**.

# Propriétés des contours

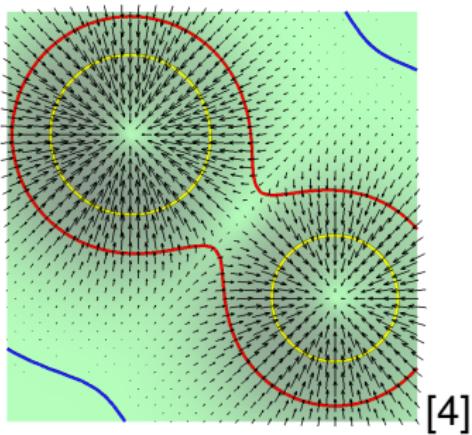
Considérons maintenant une représentation différente (vue de haut) de la même fonction  $f(x, y)$ .



Les vecteurs qui y sont affichés représentent **la direction de la dérivée** de la fonction  $f(x, y)$ .

# Propriétés des contours

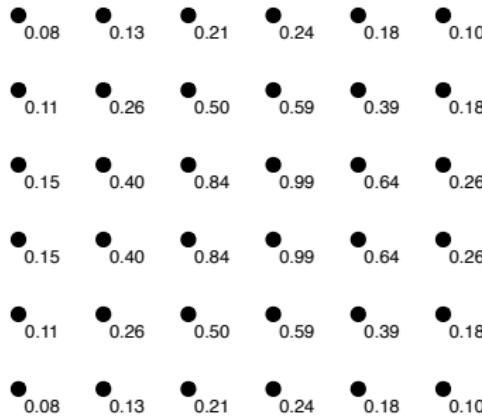
**Propriété # 3 :** un isocontour est toujours **perpendiculaire au gradient de la fonction** affichée.



# Construction d'un contour

À partir de données discrètes, comment en arrive-t-on à **construire un isocontour** ?

Soit un domaine  $D$  contenant l'ensemble des points  $(x, y)$  pour lesquels la valeur d'une fonction  $f(x, y)$  est connue.



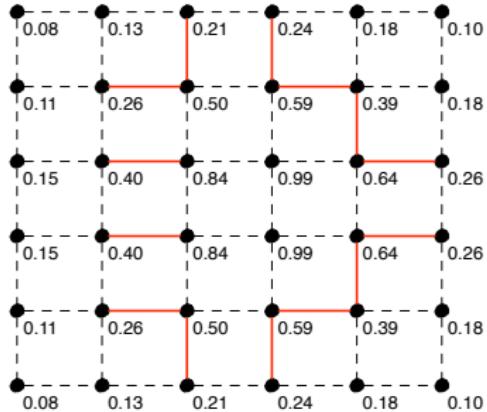
# Construction d'un contour

Pour chaque cellule de  $D$ , on cherche à savoir **si elle sera traversée par l'isocontour** associé à la valeur  $v$ .

Pour cela, **on analyse les quatre arêtes qui le composent**. Pour savoir si un contour intersecte l'arête délimitée par  $(x_i, y_i)$  et  $(x_j, y_j)$ , on teste si  $v$  se situe entre  $f(x_i, y_i)$  et  $f(x_j, y_j)$ .

À des fins d'exemple, on fixe

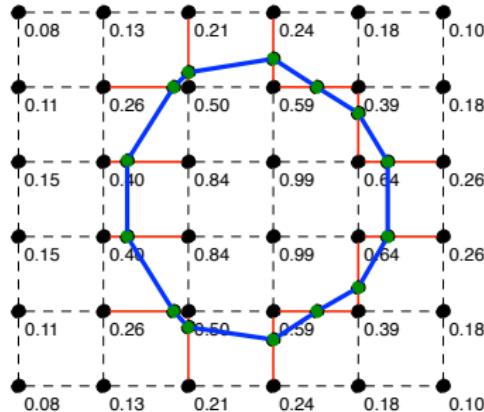
$$v = 0.48$$



# Construction d'un contour

Si ce test est réussi, l'**intersection entre le contour et l'arête** sera située au point

$$q = \frac{(x_i, y_i) \cdot (f(x_i, y_i) - v) + (x_j, y_j) \cdot (v - f(x_j, y_j))}{f(x_j, y_j) - f(x_i, y_i)}.$$



# Construction d'un contour

Cette méthode de construction peut toutefois **créer une certaine confusion**. En effet, considérons le cas de la cellule suivante :



On trouve que **chaque arête de la cellule possède une intersection avec l'isocontour si  $v = 0.37$** .

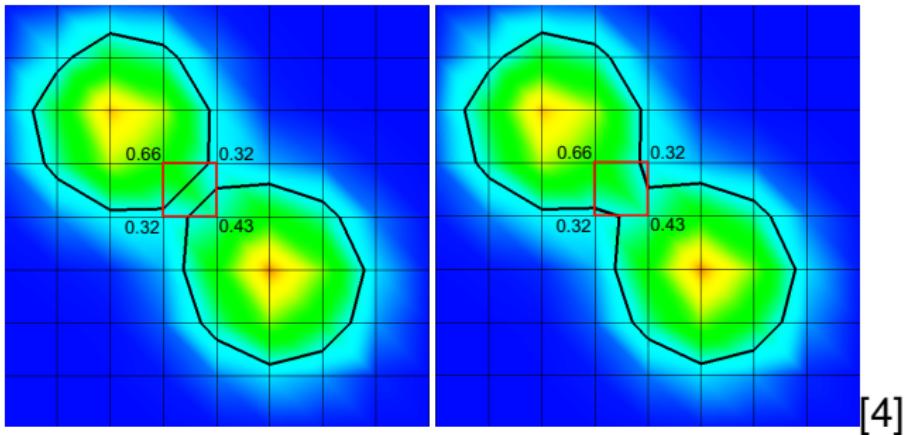
# Construction d'un contour

Partant du principe qu'un même contour ne peut se croiser lui-même, il existe **deux configurations possibles** :



Ce choix déchirant peut mener à **d'importantes différences dans la topologie des contours**.

# Construction d'un contour



En pratique, **à moins d'avoir des connaissances additionnelles sur la structure** que doivent avoir les contours (le nombre limite de composantes, par exemple), il est **impossible de discriminer** entre les deux options.

# Construction d'un contour

Un autre problème de cette méthode est **sa complexité** : pour une valeur donnée  $v$ , on doit tester si le contour passe dans chaque cellule, ce qui implique de **vérifier si une intersection existe avec les quatre arêtes** de celle-ci.



# Construction d'un contour

Il importe donc de **réduire le nombre d'opérations** à faire par cellule.

Les méthodes les plus populaires pour accomplir cette tâche sont les **Marching Squares** (pour les données 2D et les cellules à quatre côtés) et les **Marching Cubes** (pour les données 3D et les cellules à six faces).

# Marching Squares

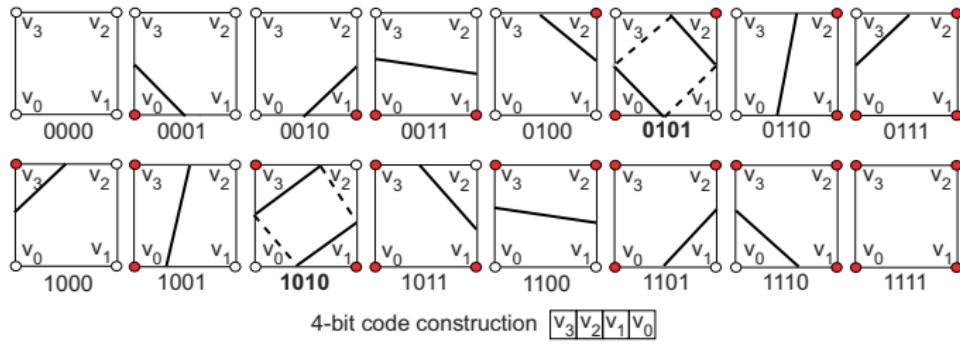
La méthode des **Marching Squares** repose sur la **détermination de l'état topologique des cellules** en se basant sur la valeur scalaire du contour  $v$ .

L'état topologique décrit de quelle façon les arêtes de la cellule sont intersectées par le contour.

# Marching Squares

Pour une cellule donnée, on dira qu'un de ses points  $(x_i, y_i)$  est à l'intérieur de l'isocontour si  $f(x_i, y_i) \leq v$  et à l'extérieur de celui-ci si  $f(x_i, y_i) > v$ .

Ce faisant, il existe **16 états topologiques possibles** pour une cellule.



[4]

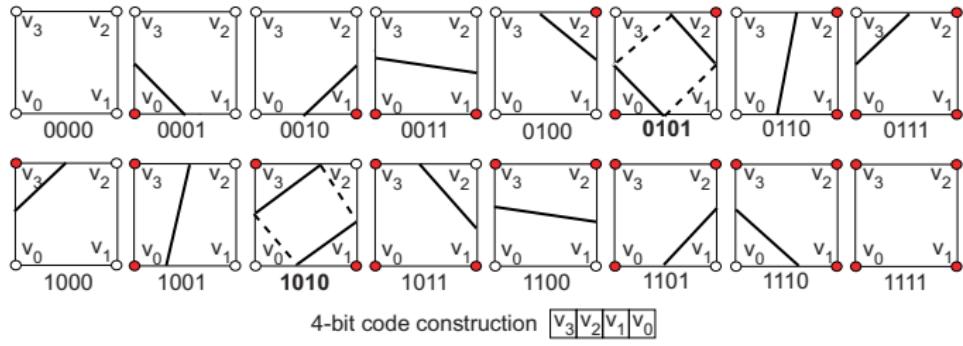
# Marching Squares

L'état topologique d'une cellule peut ainsi **être stocké sur un entier de 4 bits**, où chaque bit correspond au statut d'un des points de la cellule :

- 0 → Extérieur
- 1 → Intérieur

Cette valeur entière permettra d'**aiguiller le programme vers du code optimisé pour chaque état**.

# Marching Squares



[4]

Les états 1010 et 0101 correspondent aux **situations ambiguës** présentées précédemment. Il sera alors nécessaire d'imposer un **choix** entre les lignes pleines et les lignes pointillées.

# Marching Squares

```
Pour chaque cellule c_i de l'ensemble de données
{
    int index = 0;
    Pour chaque sommet s_i de c_i
        Enregistrer l'état de s_i
            dans le i-ème bit de l'index;
    Choisir le code optimisé à la cellule
        selon la valeur de l'index;
    Pour chaque arête concernée
        Évaluer l'intersection avec le contour;
    Tracer le segment de droite reliant les intersections;
}
```

# Marching Cubes

Le principe de l'algorithme des **Marching Cubes** est similaire à celui des Marching Squares à la différence qu'il utilise des données scalaires **positionnées dans un espace 3D** plutôt que sur un plan 2D.

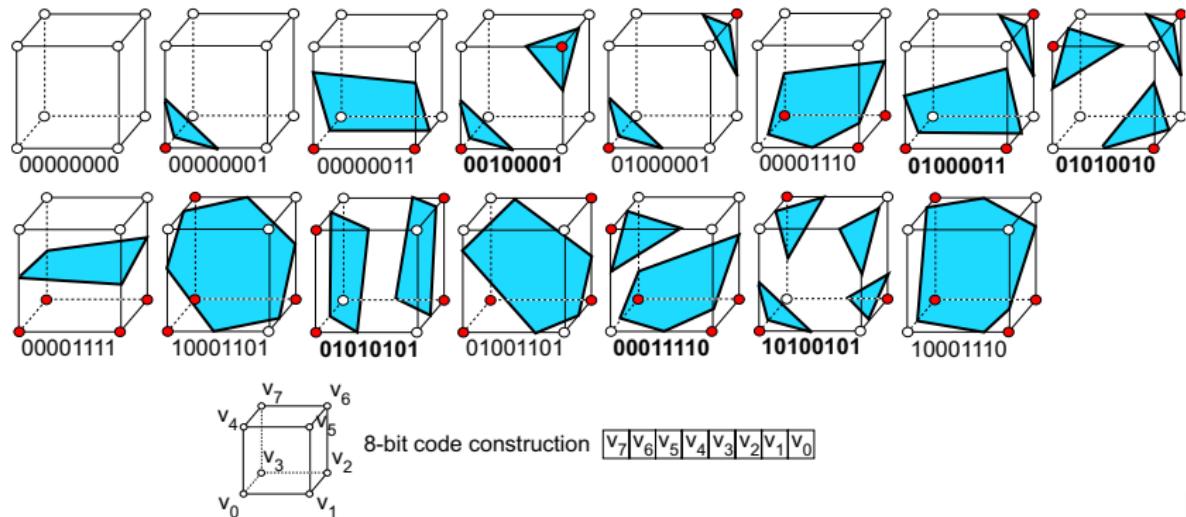
De plus, le résultat des Marching Cubes sera **une isosurface 2D** au lieu d'un isocontour 1D.

# Marching Cubes

On pourrait croire que, puisqu'un cube possède huit sommets, on devrait **considérer**  $2^8 = 256$  états topologiques différents pour les Marching Cubes.

Or, on peut réduire ce nombre à **15 états** en éliminant les cas symétriques parmi les 256. On représentera malgré tout chaque cellule par un code à 8 bits.

# Marching Cubes

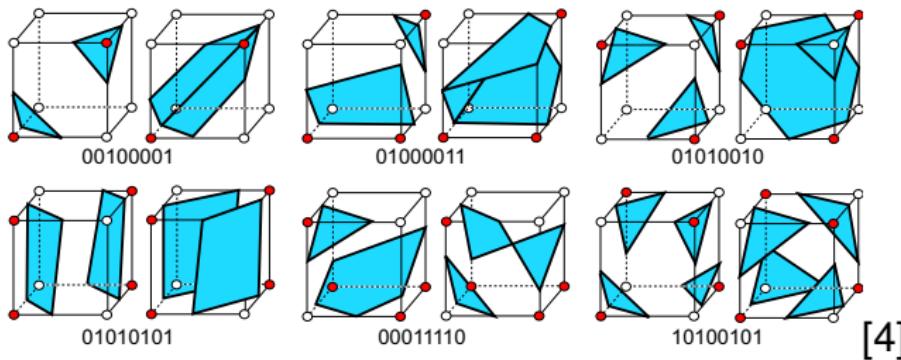


[4]

# Marching Cubes

Comme c'était le cas avec les Marching Squares, les Marching Cubes présentent des **configurations ambiguës**.

Il y en a cette fois-ci six, au lieu de seulement deux pour les Marching Squares :

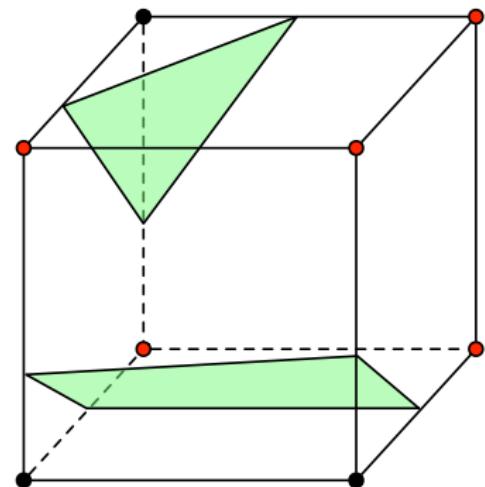
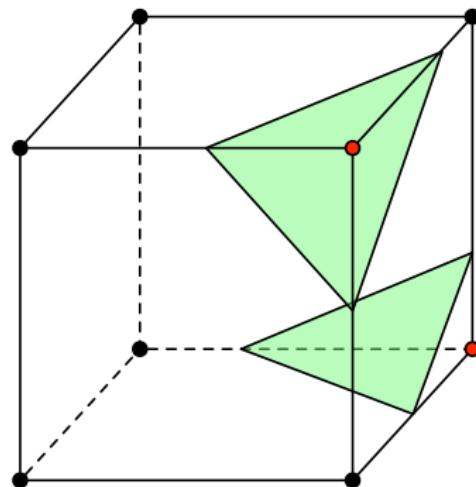


# Marching Cubes

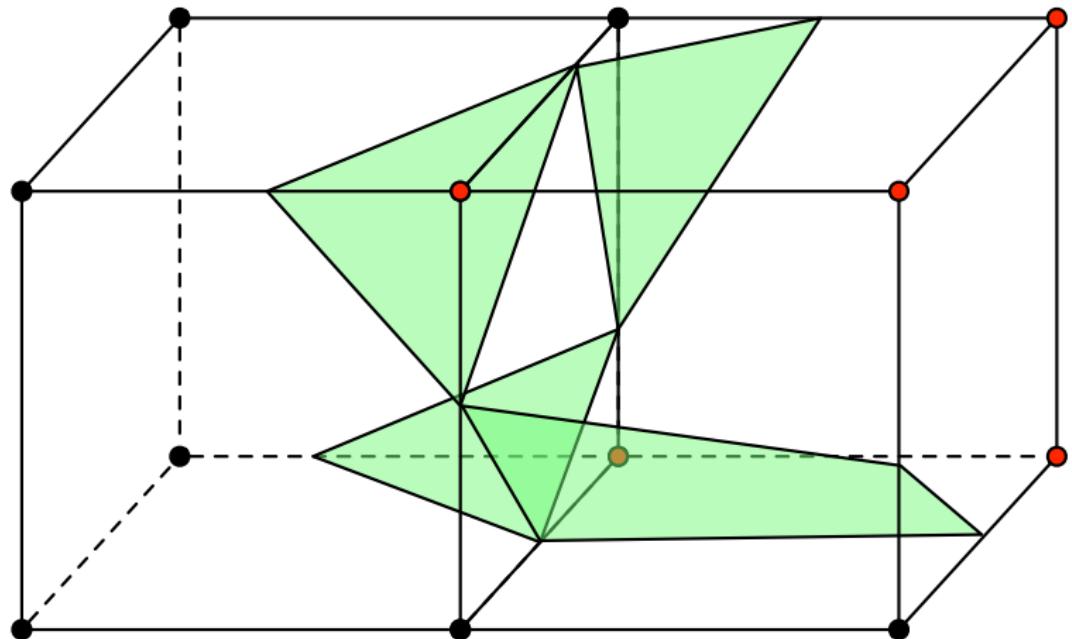
En 3D, la gestion de ces ambiguïtés est toutefois plus complexe : **on ne peut pas choisir la configuration d'une cellule indépendamment de son voisinage.**

En effet, une mauvaise gestion des ambiguïtés peut **engendrer des trous artificiels dans la surface.**

# Marching Cubes



# Marching Cubes



# Marching Cubes

La solution à ce problème est simple, mais peut être difficile à mettre en pratique.

On part du principe qu'une cellule cubique présentant une ambiguïté **possède au moins une face ambiguë**. Cette face est nécessairement **partagée avec la cellule voisine**. Il importe donc, pour ces deux cellules, de **gérer cette face problématique de la même façon**.

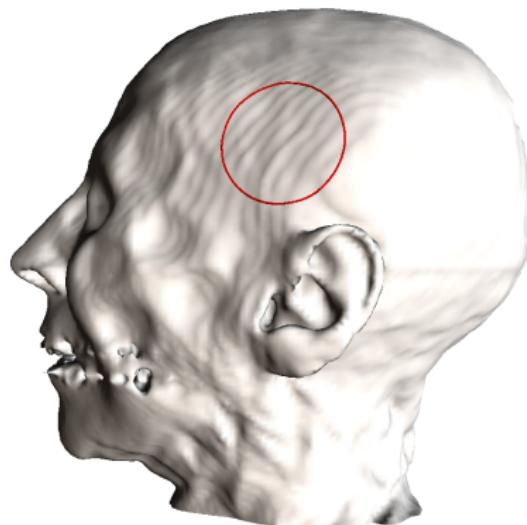
# Marching Cubes

Une fois l'isosurface calculée pour un ensemble de points, une étape supplémentaire doit être faite : **le calcul des normales.**

Cette étape est nécessaire afin de pouvoir effectuer les calculs d'**illumination** et de **lissage**.

# Marching Cubes

La figure suivante illustre une isosurface obtenue à partir d'un IRM. La résolution est de  $128 \times 128 \times 128$  et la valeur scalaire associée à la surface correspond à la peau du cobaye.



[4]

# Marching Cubes

Il faut interpréter les résultats de l'algorithme des Marching Cubes avec discernement.

À titre d'exemple, la figure précédente présente des **motifs de vague** sur la surface. Ceux-ci ne font toutefois pas partie des données originales, mais sont plutôt les résultat d'un **effet de crénelage** du à un **sous échantillonnage des données**.

# Marching Cubes

Comme c'était le cas pour les isocontours, on peut **afficher plus d'une isosurface à la fois** à partir d'un même ensemble de données.

Il faut alors opter pour des **surfaces translucides** afin d'éviter l'occultation.

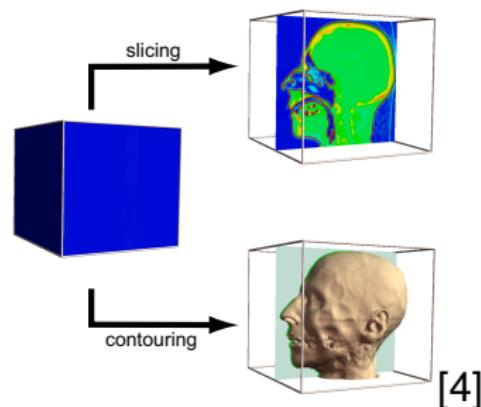


[4]

# Contour vs. surface

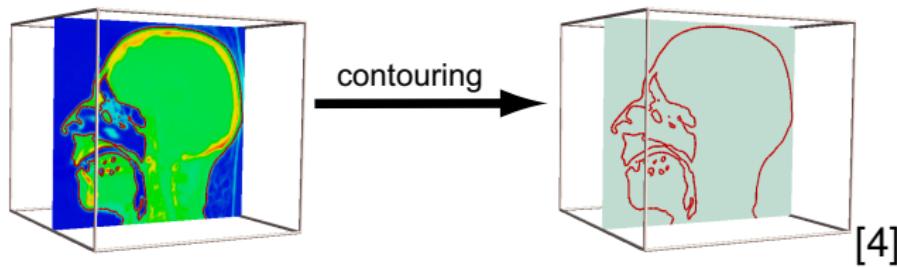
Les notions d'**isocontour** et d'**isosurface** sont très semblables, comme en font foi les algorithmes très similaires pour leur construction. Les ressemblances ne s'arrêtent toutefois pas là.

Utilisons le même bloc de données 3D que précédemment, mais en ne considérant qu'**une tranche de données**.



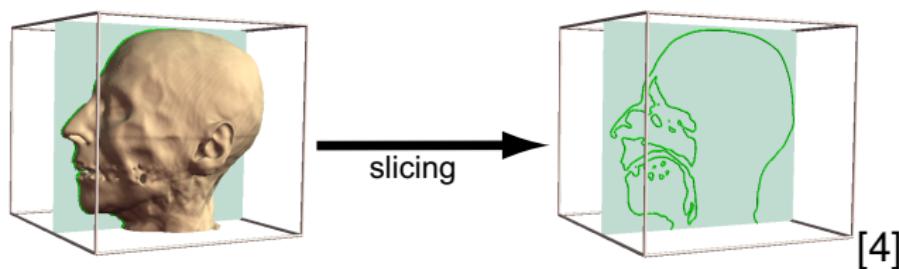
# Contour vs. surface

À partir de cette tranche, on peut **construire un isocontour** associé à la valeur scalaire correspondant à la peau du cobaye.



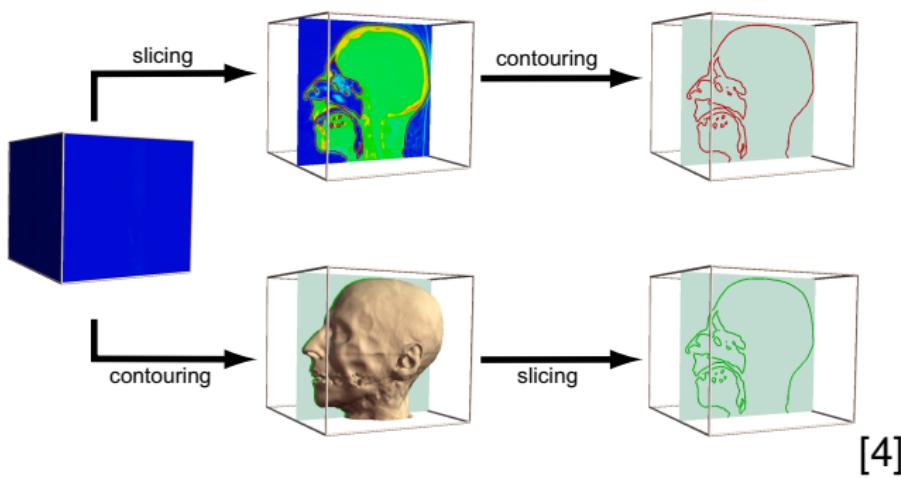
# Contour vs. surface

Au lieu de cela, si on décidait plutôt d'**évaluer l'isosurface** en entier, puis de ne considérer qu'**une seule tranche** de celle-ci. On obtiendrait alors



# Contour vs. surface

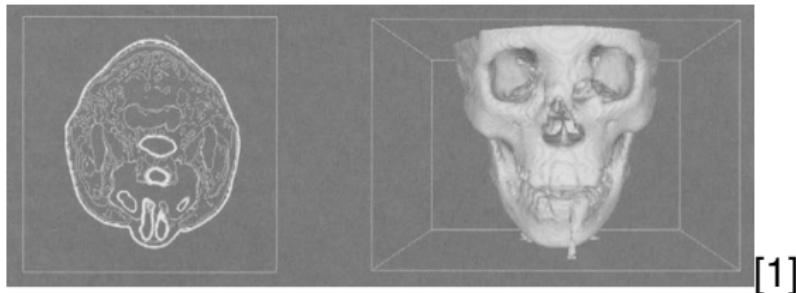
On en conclut donc que la construction d'un isocontour ou d'une surface et l'opération de tranchage d'un bloc de données sont **des opérations commutatives**.



# Contours vs. surface

Quelles sont les conséquences de cette affirmation ?

Cela suggère que si l'on dispose d'**un ensemble assez dense d'isocontours** associés aux tranches d'un volume 3D de données scalaires, on sera en mesure de **reconstruire une isosurface**.



[1]

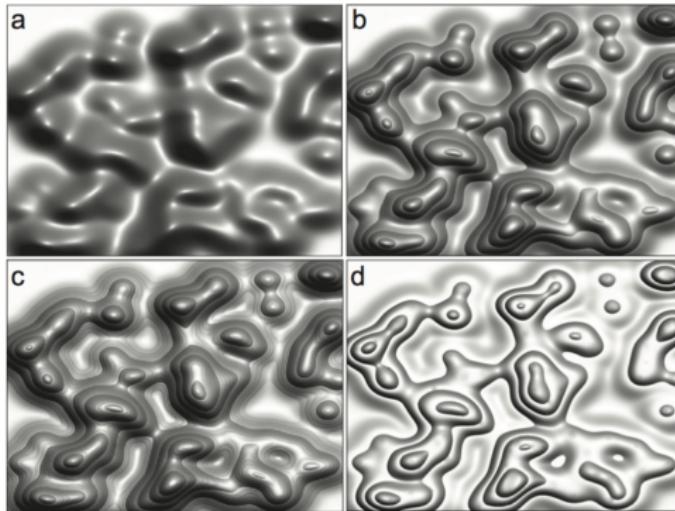
# Carte de hauteur, Placage de couleur, Contours

Chaque type de cartographie scalaire possède ses avantages/désavantages :

- **Carte de hauteur** : Facile à apprendre, intuitive, montre l'information de gradient en terme de pente, MAIS *Information quantitative difficile à extraire, peut être difficile de trouver le maxima, ombrage*
- **Placage de couleur** : Facile à apprendre, intuitive, montre l'information de gradient en terme de pente, MAIS *Information quantitative difficile à extraire, choix complexe de la plage de couleur*
- **Contours** : Information quantitative facile à extraire MAIS *ne montre pas toute l'étendue des valeurs, moins intuitif*

# Carte ondulée

Une **carte ondulée** (*enridged plot*) permet de combiner les avantages de chacun :

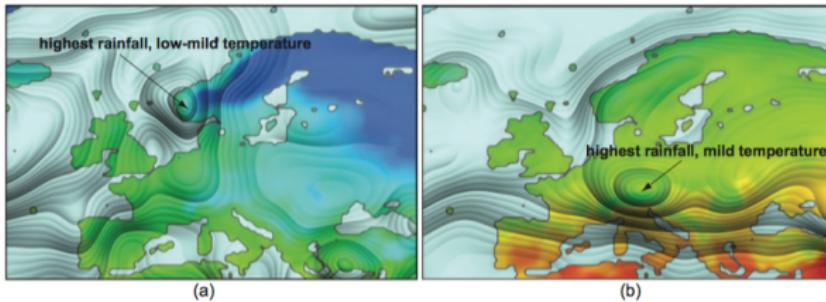


**Figure 5.20.** (a) Height plot. (b-d) Enridged height-plot variations for the same dataset.

[4]

# Carte ondulée

Une **carte ondulée** (*enridged plot*) permet de combiner les avantages de chacun :



**Figure 5.21.** Average rainfall and temperature over Europe for January (a) and July (b), visualized using enridged plots.

[4]

# Références

1 Introduction

2 Visualisation de données scalaires

3 Références

# Références I

-  C. D. Hansen and C. R. Johnson.  
The visualization handbook, 2004.
-  W. Schroeder, K. Martin, and B. Lorensen.  
The visualization toolkit : An object-oriented approach to 3d graphics, 2006.
-  C. Taras, T. Ertl, R. Botchen, and I. Entina.  
Online course for scientific visualization, 2007.
-  A. C. Telea.  
Data visualization : Principles and practice, 2008.
-  M. Ward, G. Grinstein, and D. Keim.  
Interactive data visualization : Foundations, techniques and applications, 2010.
-  H. Wright.  
Introduction to scientific visualization, 2007.