IMN430

Chapitre 4
Visualisation de volumes

Olivier Godin & Michaël Bernier

Université de Sherbrooke

7 mars 2017

Plan de la présentation

1. Introduction
2. Notions de base en visualisation de volumes
3. Espace image et espace objet
4. Calcul d'enveloppe convexe
5. Références

Introduction

1 Introduction

2 Notions de base en visualisation de volumes

3 Espace image et espace objet

4 Calcul d'enveloppe convexe

5 Références

Mise en contexte

Au deuxième chapitre, on a vu plusieurs techniques permettant de **visualiser des données scalaires**.

On reviendra sur ce sujet dans ce chapitre, mais en portant une attention particulière aux **données scalaires 3D**, aussi appelées **données volumétriques**.

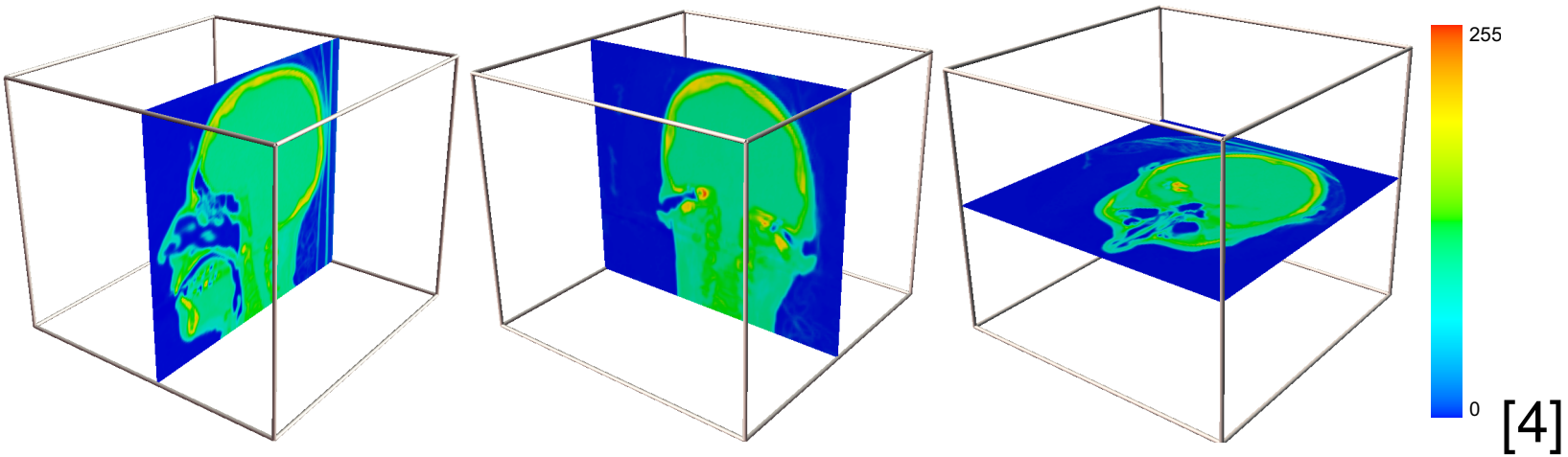
Mise en contexte

On retrouve ce type de données dans plusieurs domaines, mais c'est en **imagerie médicale** que son utilisation est la plus courante :

- Tomodensitométrie
- Imagerie par résonance magnétique
- Tomographie par émission de positrons
- Tomographie d'émission monophotonique

Mise en contexte

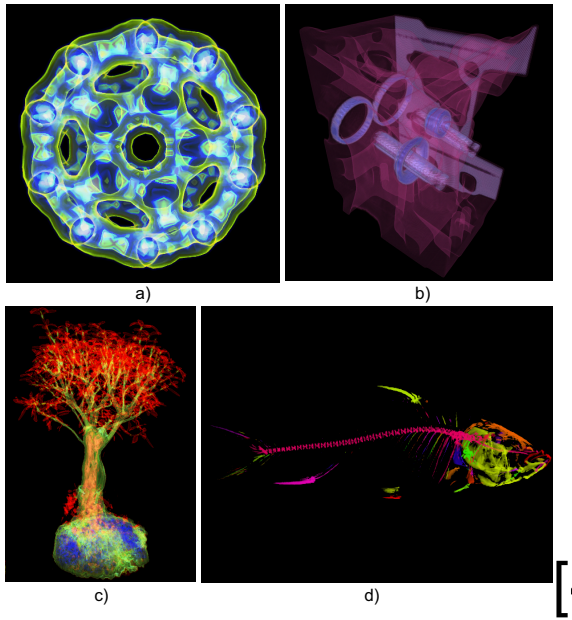
Ces volumes de données peuvent être visualisés **par tranche** ou encore **à l'aide d'isosurfaces**. Ces approches ont toutefois le désavantage de **n'illustrer qu'une partie des données à la fois**.



[4]

Mise en contexte

Dans ce chapitre, on verra des techniques permettant de produire **des images d'un volume entier** de données scalaires 3D.

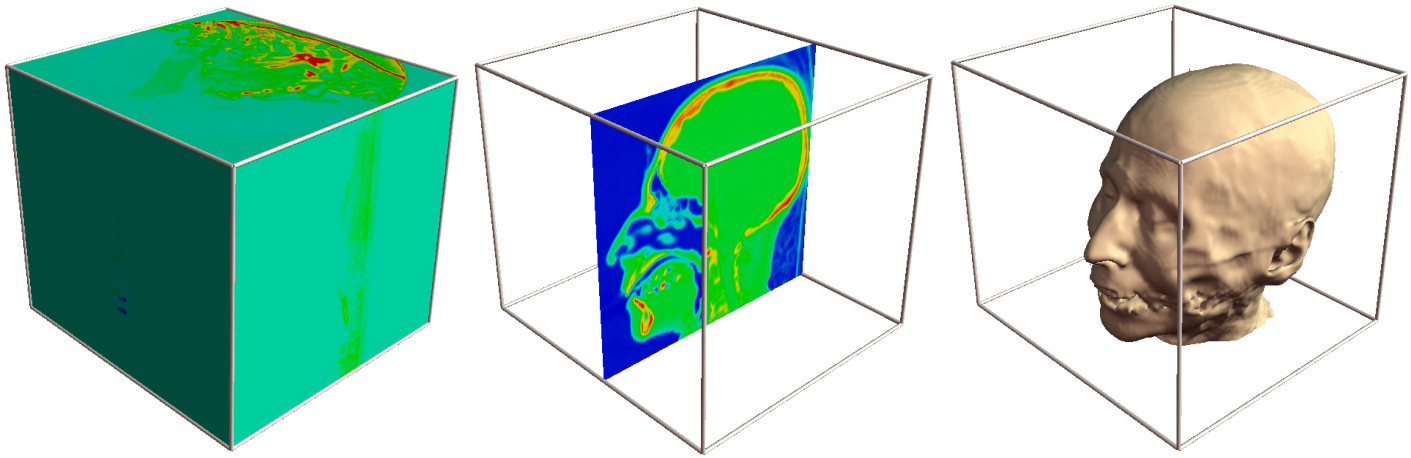


[4]

On sera confronté aux problèmes liés à l'**affichage d'un volume sur un plan image 2D** de même qu'aux défis associés au **traitement de grands ensembles de données**.

Motivation

Dans la figure suivante, on visualise un bloc de données 3D **selon trois représentations 2D** différentes :

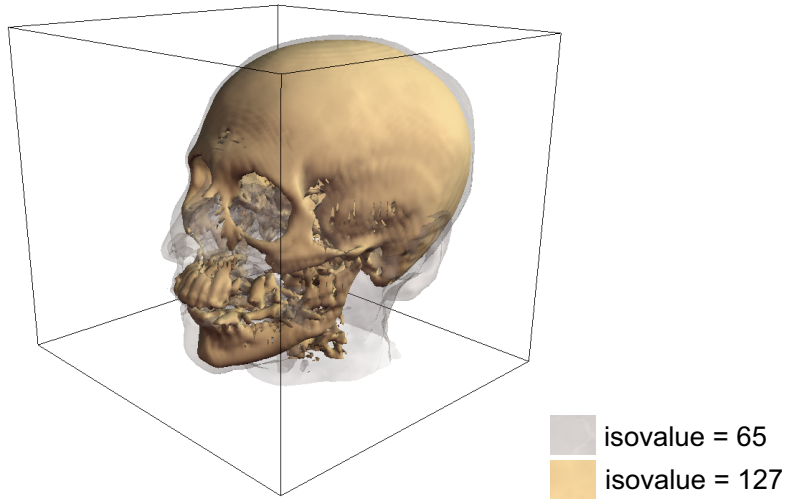


[4]

Que faire si l'information d'intérêt **ne peut être entièrement représentée sur une structure 2D** ?

Motivation

Pour introduire le concept de **visualisation de volumes**, on utilise comme point de départ deux techniques connues : les **isosurfaces** et la **visualisation par tranches**.

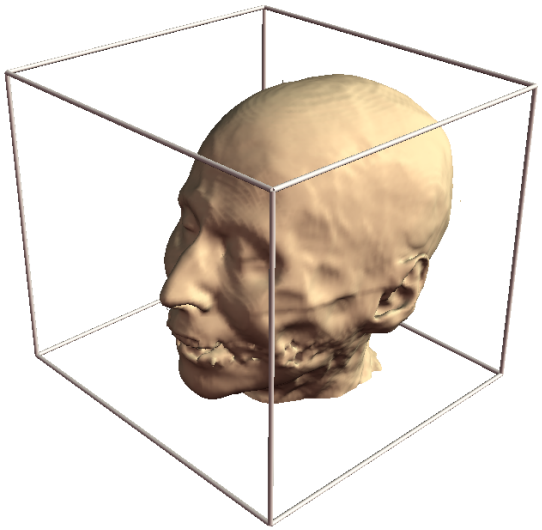


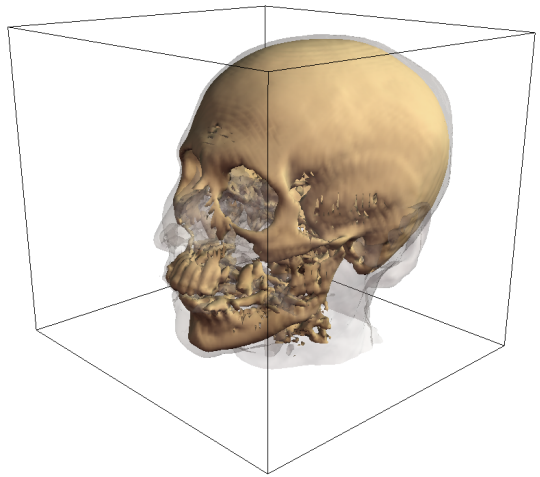
[4]

La figure précédente illustre **deux isosurfaces**, correspondant respectivement à **la peau** (valeur scalaire de 65) et **au crâne** (valeur scalaire 127).

Motivation

Les deux surfaces sont affichées avec **des couleurs et des niveaux de transparence différents**.





[4]

En comparaison, le nouveau résultat donne **davantage d'information**, soit la position de deux structures au lieu d'une seule.

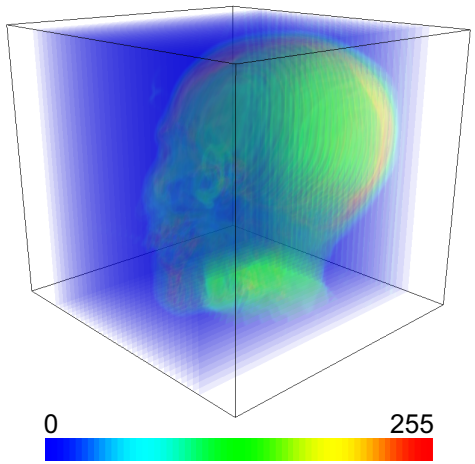
Motivation

On pourrait généraliser ce principe **en ajoutant davantage d'isosurfaces** associées à différentes valeurs scalaires, et en associant à chacune une couleur et un niveau de transparence spécifique.

Selon la distance entre les valeurs scalaires, on obtiendrait **une approximation plus ou moins fine** du volume complet de données.

Motivation

Le deuxième exemple généralise quant à lui **la visualisation par tranches** en superposant celles-ci et en leur assignant un certain niveau de transparence.



[4]

On voit apparaître, au centre, une structure ayant la forme d'une tête. Il s'agit encore d'une **approximation grossière** du volume complet de données.

Motivation

Même si la qualité des deux résultats précédents laisse à désirer, ils pavent la voie vers **des techniques plus avancées de visualisation de volumes**.

Dans la section suivante, on verra comment obtenir **un judicieux mélange de textures colorées et transparentes** afin de permettre une visualisation pertinente d'un volume de données scalaires 3D.

Notions de base en visualisation de volumes

1 Introduction

2 Notions de base en visualisation de volumes
- Rendu volumétrique et lancer de rayon
- Composition d'images et shading

3 Espace image et espace objet

4 Calcul d'enveloppe convexe

5 Références

Rendu volumétrique et lancer de rayon

1 Introduction

2 Notions de base en visualisation de volumes
- Rendu volumétrique et lancer de rayon
- Composition d'images et shading

3 Espace image et espace objet

4 Calcul d'enveloppe convexe

5 Références

Fonction rayon

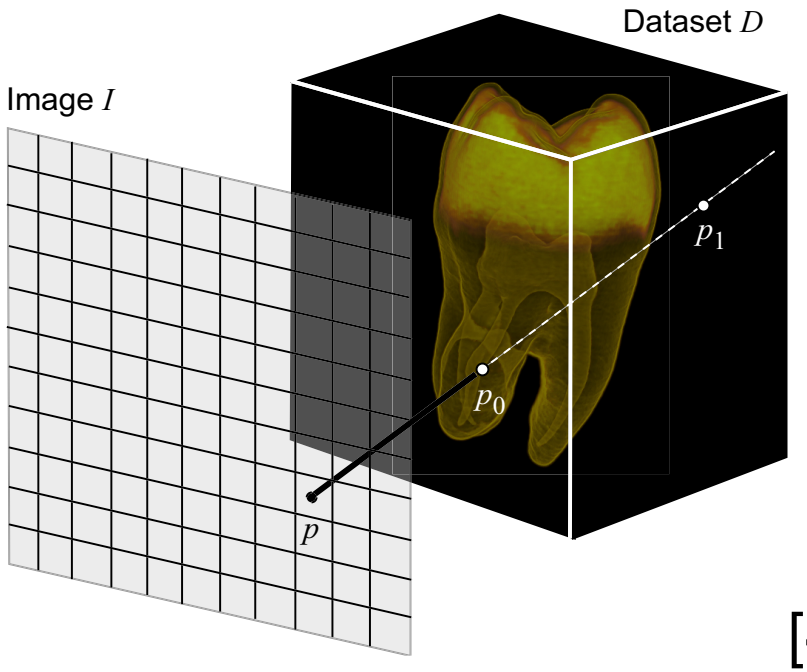
L'idée de base du **rendu volumétrique**, est fort simple : créer une représentation 2D qui illustre, à chaque pixel, **les données scalaires le long d'un rayon** dans un volume de données 3D.

La puissance du rendu volumétrique réside dans la possibilité de **combiner un ensemble de valeurs scalaires** dans un unique pixel.

Fonction rayon

Soient une **image** I divisée en pixels, et un **volume de données** $D \subset \mathbb{R}^3$ dont les valeurs scalaires sont données par $s : D \to \mathbb{R}$.

Pour chaque pixel p de l'image I, **on lance un rayon r** perpendiculaire au plan image qui intersecte D aux points p_0 et p_1.



[4]

Fonction rayon

La valeur $I(p)$ associée au pixel p sera **déterminée par les données scalaires du volume rencontrées par le rayon r**, entre p_0 et p_1.

Le **segment de droite** entre p_0 et p_1 est défini par l'équation

$$p_t = p(t) = (1-t)p_0 + tp_1 \quad \text{avec} \quad t \in [0,1],$$

tandis que les **données scalaires** associées à ces points seront notées $s(p_t)$.

Fonction rayon

La valeur du pixel p dans l'image I sera ainsi donnée par

$$I(p) = F(s(p_t)) \text{ avec } t \in [0, 1],$$

où $F : \mathbb{R} \to [0, 1]^3$ est la **fonction rayon** qui **combine les données scalaires du volume** le long du rayon **r** pour obtenir $I(p)$ dans l'espace RGB.

Le résultat visuel **dépendra beaucoup du choix de la fonction rayon**. Dans la suite de cette section, on s'attardera à détailler des exemples de fonctions rencontrées dans la pratique.

Fonction de transfert

Au coeur de toute fonction rayon se trouve un mécanisme servant à **mettre en correspondance chaque valeur scalaire rencontrée par le rayon à une couleur RGBA**, où la composante A dénote l'opacité.

On appelle ce mécanisme la **fonction de transfert**. Cette notion avait été introduite lorsqu'il était question de **placage de couleurs**, au chapitre 2.

Fonction de transfert

Une **fonction de transfert** f associe une couleur et une opacité à une valeur scalaire donnée. Elle est donc de la forme $f : \mathbb{R} \to [0, 1]^4$. Pour la suite, on identifiera par f_R, f_G, f_B et f_A les sous-fonctions de f associées à chacune des composantes de RGBA.

Une **fonction rayon** F sert à combiner les valeurs retournées par la fonction de transfert f le long d'un rayon afin d'obtenir une unique valeur RGB qui sera celle du pixel.

Fonction de transfert

Notons qu'**une même fonction rayon F peut être utilisée avec différentes fonctions de transfert f**.

En effet, comme la fonction de transfert met en correspondance une seule valeur scalaire avec une couleur et que la fonction rayon se charge de combiner ces informations le long du rayon pour obtenir l'intensité du pixel, le fait de **modifier la fonction de transfert utilisée influencera le résultat visuel**.

Fonction de transfert

Le processus de **conception** et d'**application d'une fonction de transfert** afin d'obtenir une séparation visuelle significative des différents éléments composant le volume s'appelle la **classification**.

Le choix d'une fonction de transfert permettant une bonne classification des matériaux présents est une étape cruciale de la visualisation de volume.

Fonction de transfert

Tel que mentionné, les fonctions de transfert les plus communes vont **mettre en correspondance des valeurs scalaires** du volume **avec des couleurs et des opacités**. Ce n'est toutefois pas la seule option possible.

On pourrait par exemple définir une fonction de transfert utilisant **le gradient du volume de données** pour calculer les composantes R, G, B et A associées à un point. Il serait aussi envisageable de combiner les données associées à la même source, mais **provenant de plusieurs modalités d'acquisition**.

Projection d'intensité maximale

Une des fonctions rayon les plus simples en rendu volumétrique est la **projection d'intensité maximale** (*maximum intensity projection*, ou MIP).

Pour un pixel p et un rayon **r**, la fonction MIP trouve **la valeur scalaire maximale le long de r** et évalue l'intensité de p en appliquant la fonction de transfert sur cette seule valeur :

$$I(p) = f\left(\max s(p_t)\right) \quad \text{avec} \quad t \in [0, 1].$$

Projection d'intensité maximale

Une variante de la technique précédente recherche plutôt la donnée le long de **r** qui **présente la plus grande opacité**.

On évalue donc les opacités $f_A(s(p_t))$ pour tous les points le long du rayon **r** et on assigne au pixel **la couleur du point présentant l'opacité maximale** :

$$I(p) = f(s_m),$$

où $s_m = \arg\max f_a(s(p_t))$, avec $t \in [0, 1]$.

Projection d'intensité maximale

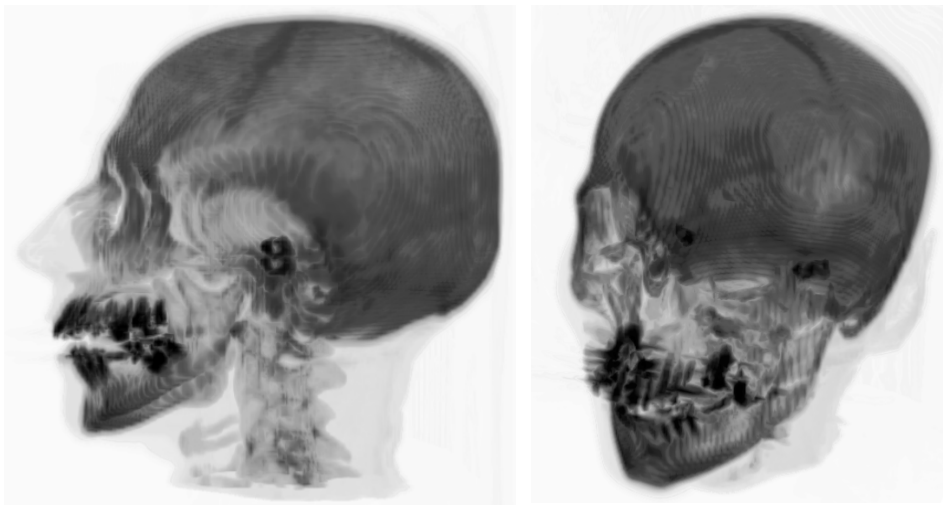
Cette variante est utile si l'on souhaite **mettre l'emphase sur un certain matériel** associé à une certaine valeur scalaire dans le volume de données.

Pour y arriver, il suffit de concevoir une fonction de transfert f dont **la composante f_A sera maximale pour la valeur scalaire en question**, et dont les composantes f_R, f_G et f_B donneront la couleur souhaitée pour l'affichage.

Projection d'intensité maximale

Un grand défaut de la projection d'intensité maximale est de **ne donner aucune information sur la profondeur**.

On voit quelle est l'intensité maximale le long d'un rayon, mais on ignore à quelle position (valeur $t \in [0, 1]$) celle-ci se trouve.



[4]

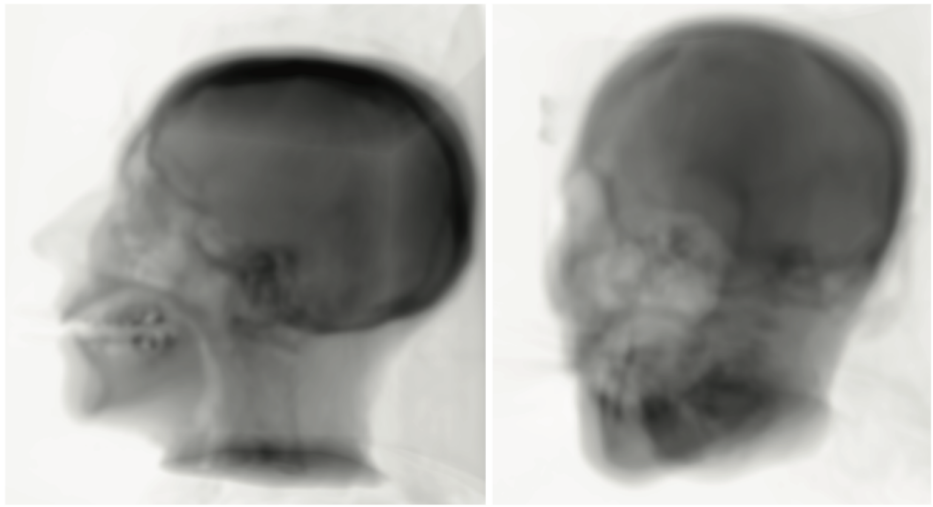
Intensité moyenne

Une autre fonction rayon de base est la **fonction d'intensité moyenne**.

À l'aide de la fonction de transfert, on évalue les intensités associées à chaque point p_t rencontré par le rayon et **on calcule la moyenne de celles-ci** :

$$I(p) = \int_0^1 f\left(s\left(p_t\right)\right)\ dt.$$

Intensité moyenne



[4]

Visuellement, le résultat sera similaire à une image obtenue par rayons X, où l'emphase est mise sur **la quantité totale de matériel** le long d'un rayon.

Distance au scalaire

Soit un scalaire $\sigma \in \mathbb{R}$. Cette troisième fonction rayon évalue, pour chaque pixel, **la distance le long du rayon jusqu'au premier point du volume dont la valeur dépasse** σ.

La fonction de transfert est alors appliquée **sur cette distance** pour obtenir l'intensité du pixel :

$$I(p) = f(t_{\text{min}}) \text{ avec } t_{\text{min}} = \min_{s(p_t) \geq \sigma} t, \text{où } t \in [0, 1].$$

À l'opposé des fonctions rayon vues jusqu'à maintenant, celle-ci **met l'emphase sur la position** (paramètre t) et non sur la valeur scalaire du point.

Retour sur les isosurfaces

Comme suite logique à la fonction rayon précédente, voyons comment **retrouver une structure d'isosurface** similaire à celle vue au deuxième chapitre.

Pour construire l'isosurface associée à une certaine valeur σ, on doit détecter la présence le long d'un rayon d'**au moins un point du volume ayant la valeur** σ.

Retour sur les isosurfaces

Si on trouve un tel point, alors **on assigne au pixel l'intensité associée à la valeur scalaire** σ par la fonction de transfert.

Si au contraire aucun point n'est trouvé, **on associe au pixel une intensité** I_0 correspondant à l'arrière plan.

$$I(p) = \begin{cases} f(\sigma) & \text{s'il existe } t \in [0,1] \text{ t.q. } s(p_t) = \sigma \\ I_0 & \text{sinon} \end{cases}$$

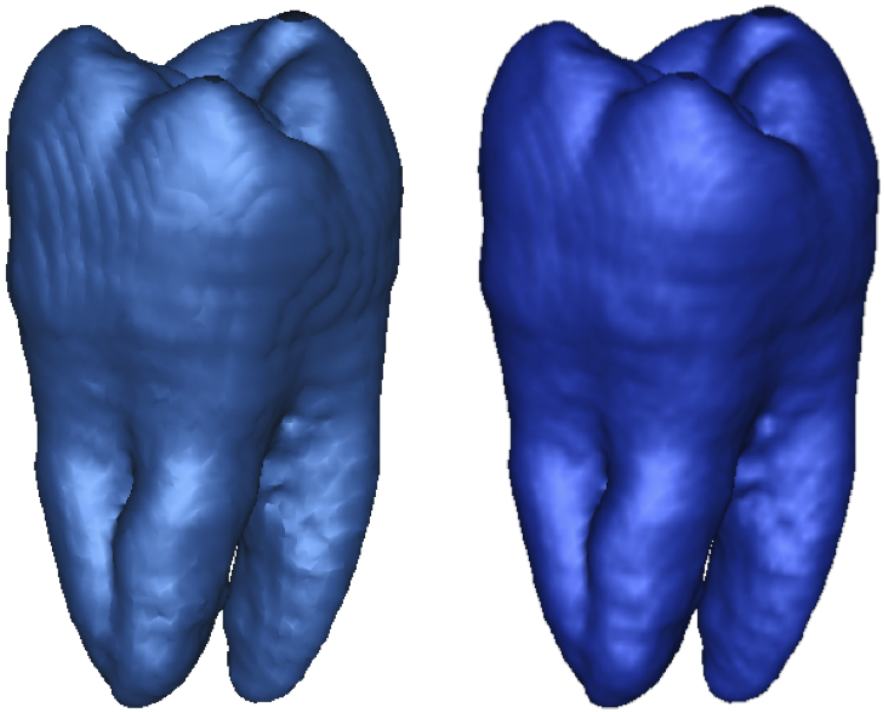
Retour sur les isosurfaces

Appliquée bêtement, **cette technique retourne une image binaire** composée des couleurs $f(\sigma)$ et I_0, ce qui correspond au rendu d'une isosurface avec comme seul élément d'éclairage une lumière ambiante.

En pratique, une telle approche n'est pas très utile. Le calcul d'une isosurface à l'aide d'une fonction rayon deviendra pertinent **lorsqu'on le combinera aux techniques de shading volumétrique** présentées dans la prochaine section.

Retour sur les isosurfaces

La figure suivante permet de **comparer les isosurfaces obtenues avec les deux techniques**. À gauche, on retrouve le résultat tel que calculé par les **Marching Cubes**, et à droit on a celui obtenu avec la **fonction rayon**.



[4]

Composition d'images et shading

1 Introduction

2 Notions de base en visualisation de volumes
- Rendu volumétrique et lancer de rayon
- Composition d'images et shading

3 Espace image et espace objet

4 Calcul d'enveloppe convexe

5 Références

Fonction de composition

Les fonctions rayon vues précédemment peuvent être considérées comme des instances particulières d'une fonction plus générale appelée **fonction de composition**.

Pour expliquer celle-ci, considérons un modèle d'illumination volumétrique simple : la couleur $I(p)$ d'un pixel est une **superposition des contributions des couleurs** $c(t)$ de tous les voxels $q(t)$ le long d'un rayon $r(p)$.

Fonction de composition

Notons par $C(t)$ la **contribution** à $I(p)$ d'un voxel $q(t)$ ayant la couleur $c(t)$.

Ainsi, $I(p)$ est donné par l'**intégrale des contributions** de tous les voxels le long du rayon $r(p)$:

$$I(p) = \int_0^1 C(t)\, dt.$$

Le défi est alors d'évaluer la contribution $C(t)$ du voxel $q(t)$ pour le pixel p.

Fonction de composition

Dans le modèle d'illumination, on suppose que chaque voxel $q(t)$ le long du rayon **émet une lumière** dont la couleur et l'intensité sont données par $c(t)$.

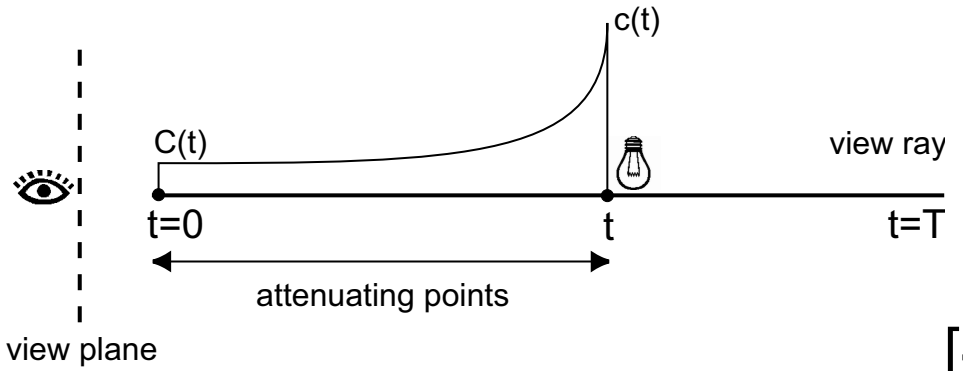
On suppose aussi que chaque voxel $q(t)$ le long du rayon **absorbe la lumière émise par les autres voxels** $q(t')$ avec $t' > t$, c'est-à-dire les voxels plus éloignés du plan image.

Fonction de composition

Notons par $\tau(x) \in [0, 1]$ le **facteur d'atténuation** à la position x.

On peut alors exprimer la variation décroissante de la contribution $C(t)$ le long du rayon à partir de la position $x = t$ jusqu'au bord du volume, à $x = 0$:

$$\frac{d\ C(t, x)}{dx} = -\tau(x)c(x)$$



[4]

Fonction de composition

En intégrant l'équation précédente le long du rayon, de $x = 0$ à $x = t$, on obtient la **contribution du voxel** $q(t)$ à l'intensité finale du pixel :

$$C(t) = c(t)e^{-\int_0^t \tau(x)\ dx}$$

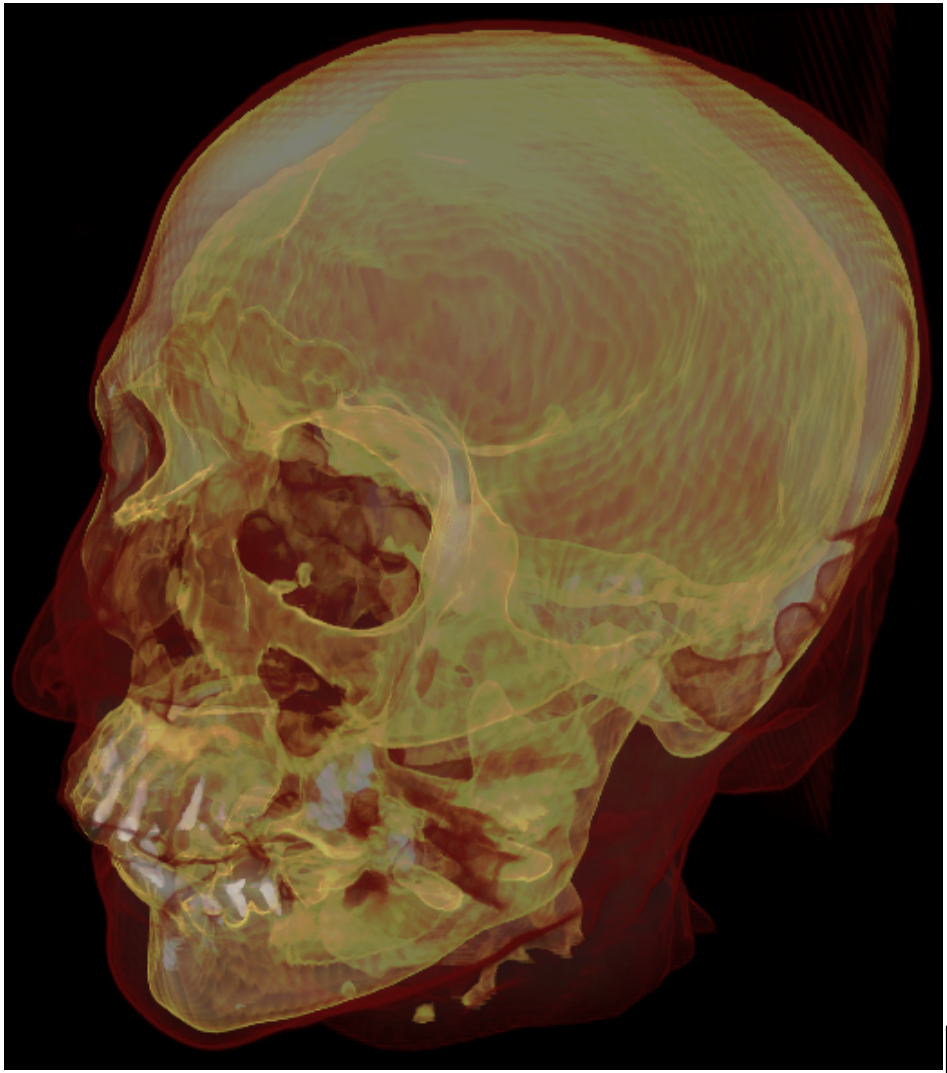
Ainsi, la contribution d'un voxel à une intensité dans le plan image **diminue exponentiellement selon l'intégrale des atténuations** associées aux voxels entre $x = 0$ et $x = t$.

Fonction de composition

À l'opposé des fonctions rayon mentionnées précédemment, la fonction de composition évalue la fonction de transfert à chaque voxel le long du rayon et **combine l'ensemble des résultats de couleur et d'opacité**.

En utilisant une fonction de transfert appropriée, on pourra **afficher plusieurs matériaux** présents le long du rayon sur l'image finale.

Fonction de composition



[4]

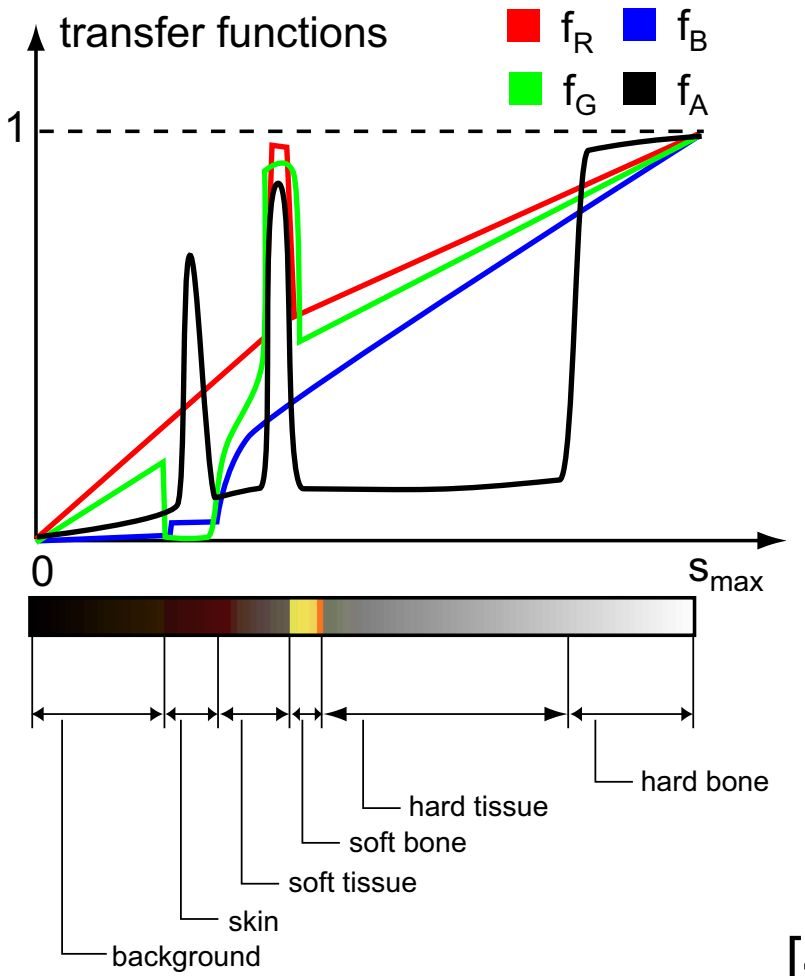
Fonction de composition

Dans le cas précédent, la fonction de transfert utilisée a été conçue pour **mettre l'emphase sur 3 matériaux** :

- la peau ;
- les os tendres (*soft bone*) ;
- les os durs (*hard bone*).

On obtient un tel résultat **en assignant de fortes valeurs d'opacité** (fonction f_A) aux valeurs scalaires associées à ces trois matériaux.

Fonction de composition



[4]

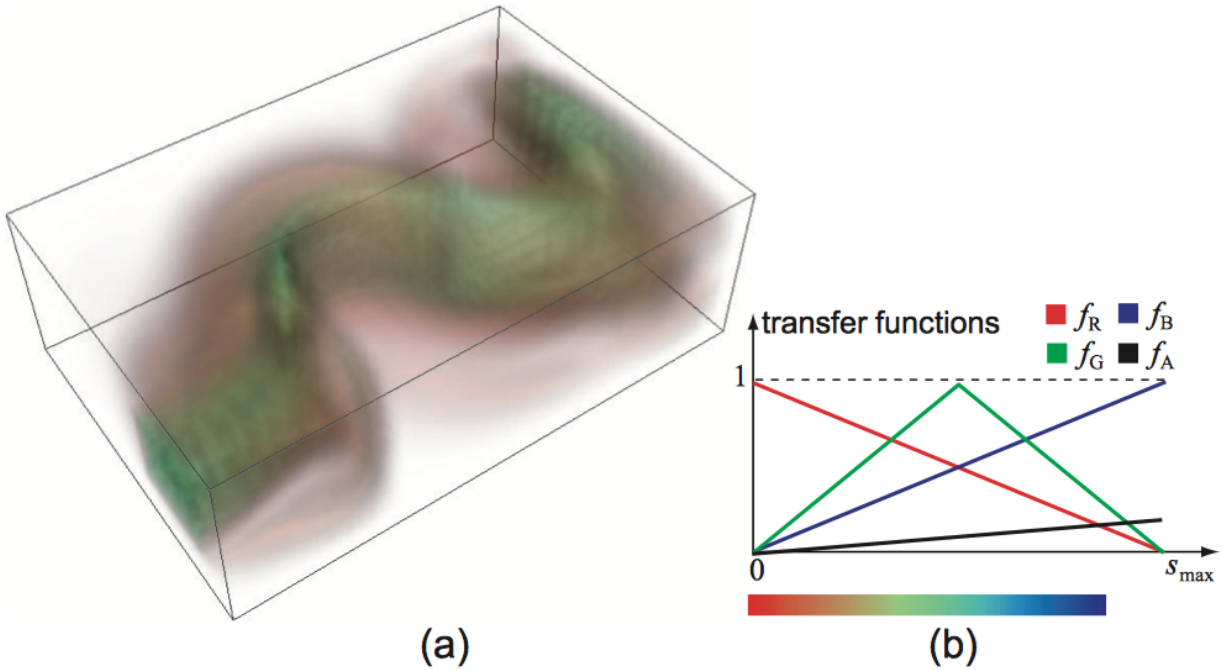
Fonction de composition

On comprend donc mieux pourquoi la conception de fonctions de transfert appropriées pour la couleur et l'opacité est une étape si cruciale dans la classification des matériaux basée sur les valeurs scalaires.

Une précaution supplémentaire consiste à utiliser des fonctions de transfert présentant des **variations douces**. On évitera ainsi les transitions brusques entre les matériaux et de nombres artefacts visuels en présence de données bruitées.

Fonction de composition

Dans cet exemple, on utilise une transition plus douce pour l'amplitude du mouvement d'un champs de vecteur (donc un scalaire), car la transition entre les régions était moins évidente que dans l'exemple précédent.



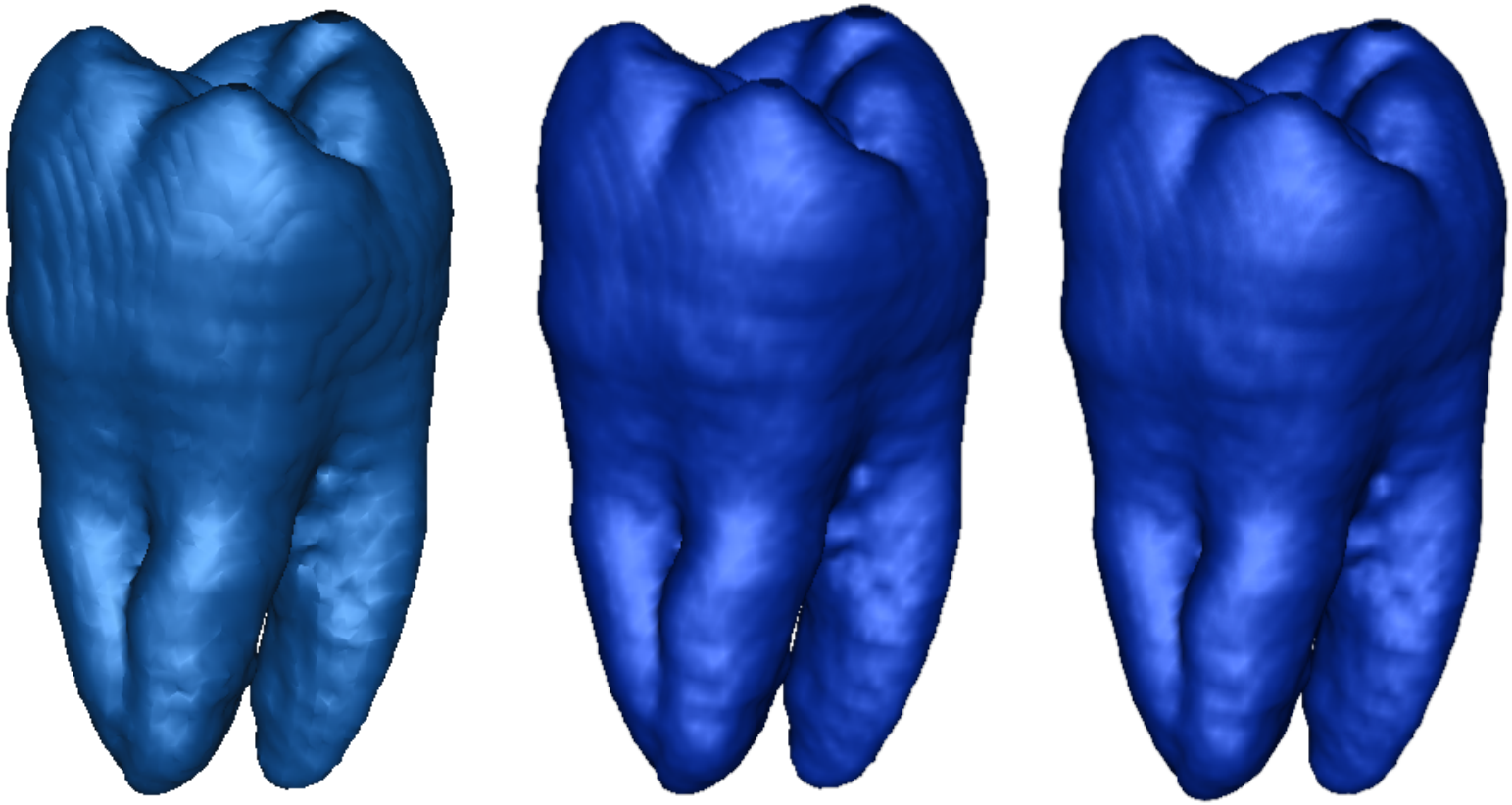
[4]

Fonction de composition

Pour conclure sur les fonctions de composition, revenons sur le fait qu'elles peuvent être vues comme une **généralisation du concept de fonction rayon**. Pour illustrer ce propos, attardons-nous à la construction d'isosurfaces à l'aide d'une fonction rayon.

Dans la fonction de composition, si on pose une **atténuation** $\tau(x) = 0$ pour tout x et une **émission** $c(t) = 0$ pour tout t à l'exception d'un certain t_0, on obtiendra l'**isosurface associée à la valeur** t_0.

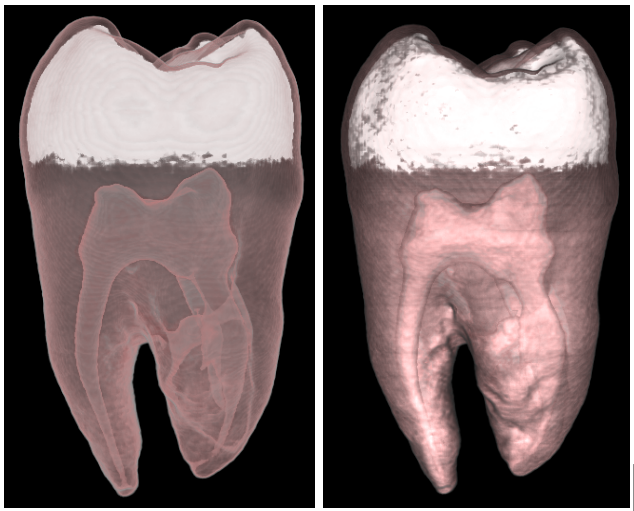
Fonction de composition



[4]

Shading volumétrique

La gestion des **effets d'éclairage** est un ajout qui peut grandement améliorer la qualité visuelle du rendu volumétrique.


[4]

La gestion du shading peut heureusement être facilement combinée avec l'intégrale dans le calcul de $I(p)$.

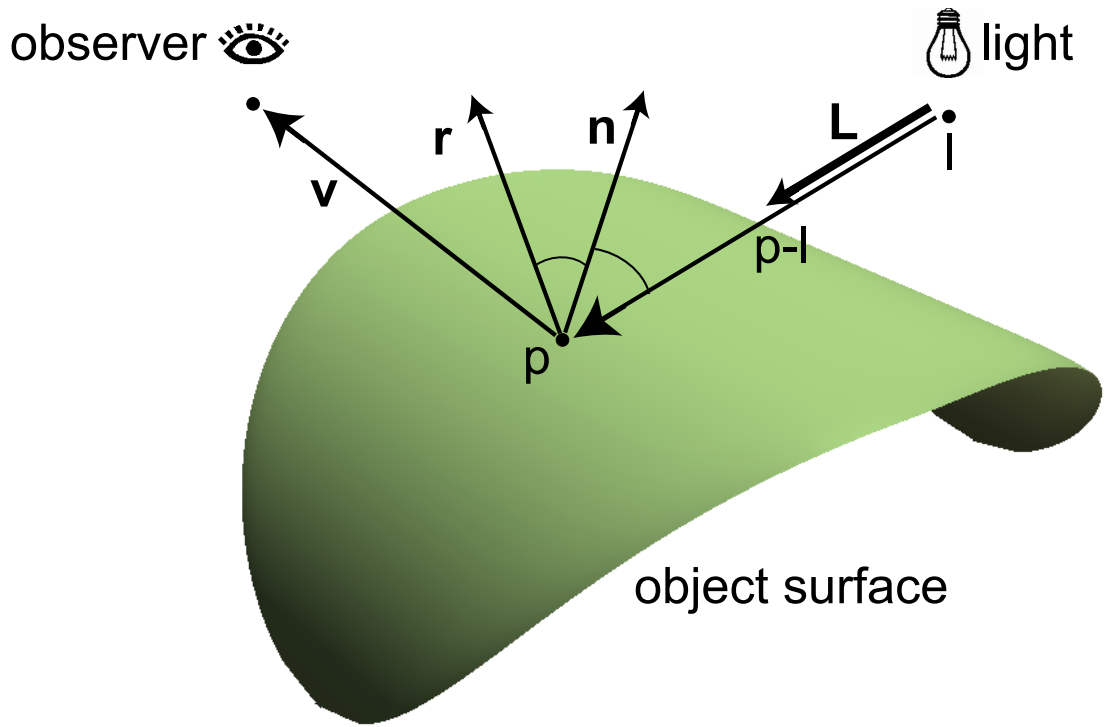
Shading volumétrique

Au lieu d'utiliser directement la couleur $c(t) = f(s(p_t))$ obtenue par l'application de la fonction de transfert f sur la valeur scalaire $s(p_t)$, on fait plutôt appel à une **fonction d'illumination**

$$i(t) = c_a + c_d(t)\max(-\mathbf{L}\cdot\mathbf{n}(t), 0) + c_s(t)\max(-\mathbf{r}\cdot\mathbf{v}, 0)^{\alpha}.$$

Cette fonction $i(t)$ correspond à l'application du **modèle d'illumination de Phong** à une surface imaginaire passant par le point p_t et ayant $\mathbf{n}(t)$ comme normale.

Shading volumétrique



[4]

Shading volumétrique

Comme on peut l'imaginer, c_a est le **coefficient d'éclairage ambiant**, qui est constant pour tout le volume.

De même, on devine que $c_d(t)$ et $c_s(t)$ sont respectivement les **coefficients diffus et spéculaire** qui, eux, dépendent de la position p_t.

Pour aisément obtenir un premier résultat visuel, on peut poser

$$c_d(t) = c_s(t) = c(t) = f(s(p_t)).$$

Shading volumétrique

Une des difficultés associées à l'application du modèle d'illumination de Phong au rendu volumétrique est la nécessité de **fournir une estimation de la normale $\mathbf{n}(t)$** à la surface en un point donné.

Pourquoi est-ce difficile ? Simplement **parce que la surface n'existe pas** ! Pour régler ce fâcheux contretemps, on fera appel aux propriétés des isosurfaces.

Shading volumétrique

On considère donc que la surface dont on cherche la normale est l'**isosurface associée à la valeur scalaire** $s(p_t)$.

Rappelons les propriétés des isosurfaces qui ont été vues au chapitre précédent et trouvons de quelle façon elles nous seront utiles.

1. Un contour peut être soit une courbe fermée ou une courbe ouverte. Il ne se termine jamais à l'intérieur du domaine.
2. Un contour ne peut se croiser lui-même ni avoir une intersection avec un autre contour.
3. Un contour est toujours perpendiculaire au gradient de la fonction affichée.

Shading volumétrique

Ainsi, comme le gradient d'une fonction $s(p_t)$ doit toujours être **perpendiculaire à l'isosurface**, on peut estimer la normale $\mathbf{n}(t)$ en évaluant simplement le gradient de $s(p_t)$:

$$\nabla s(p_t) = \left(\frac{\partial\, s(p_t)}{\partial x}, \frac{\partial\, s(p_t)}{\partial y}, \frac{\partial\, s(p_t)}{\partial z} \right)$$

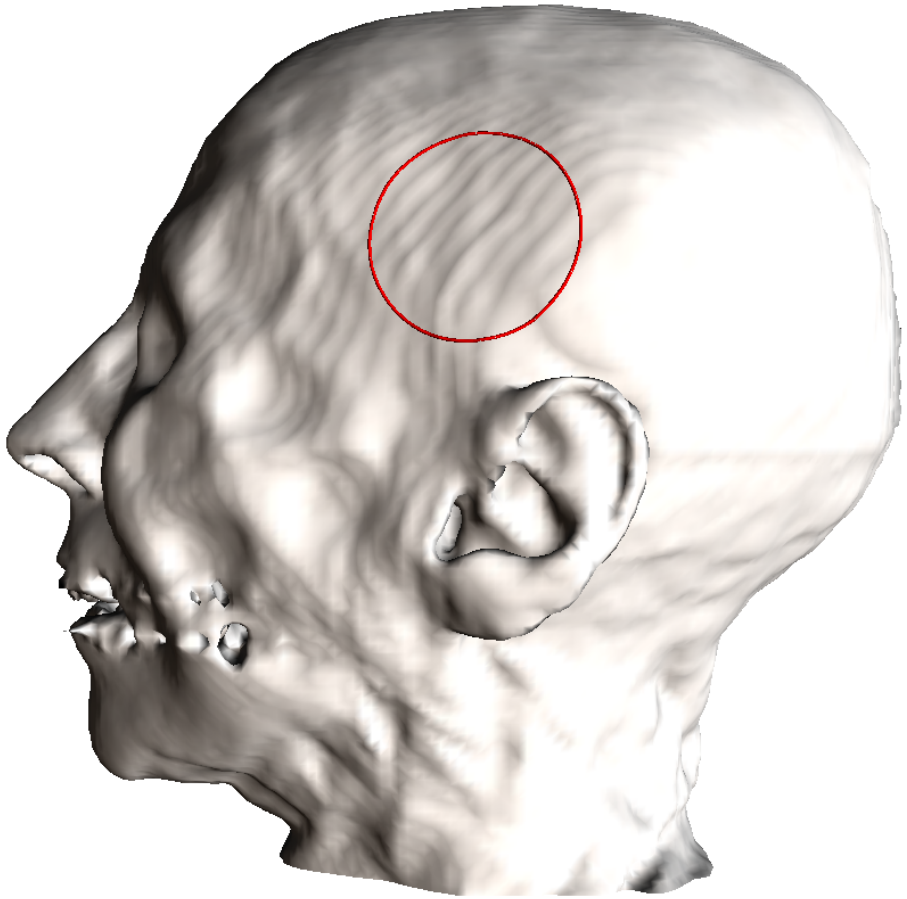
Il ne restera alors qu'à **normaliser le résultat** pour avoir la valeur de $\mathbf{n}(t)$ à utiliser dans le modèle d'illumination de Phong.

Shading volumétrique

Bien que cette dernière solution soit fonctionnelle, elle demeure **extrémement sensible aux irrégularités** dans l'ensemble de données, en raison des calculs de dérivées partielles.

Il est donc préférable de **préfiltrer les données** avant le calcul du gradient, de manière à éviter les variations brusques dans les directions de $\mathbf{n}(t)$ qui pourraient causer du crénelage dans le rendu volumétrique.

Shading volumétrique



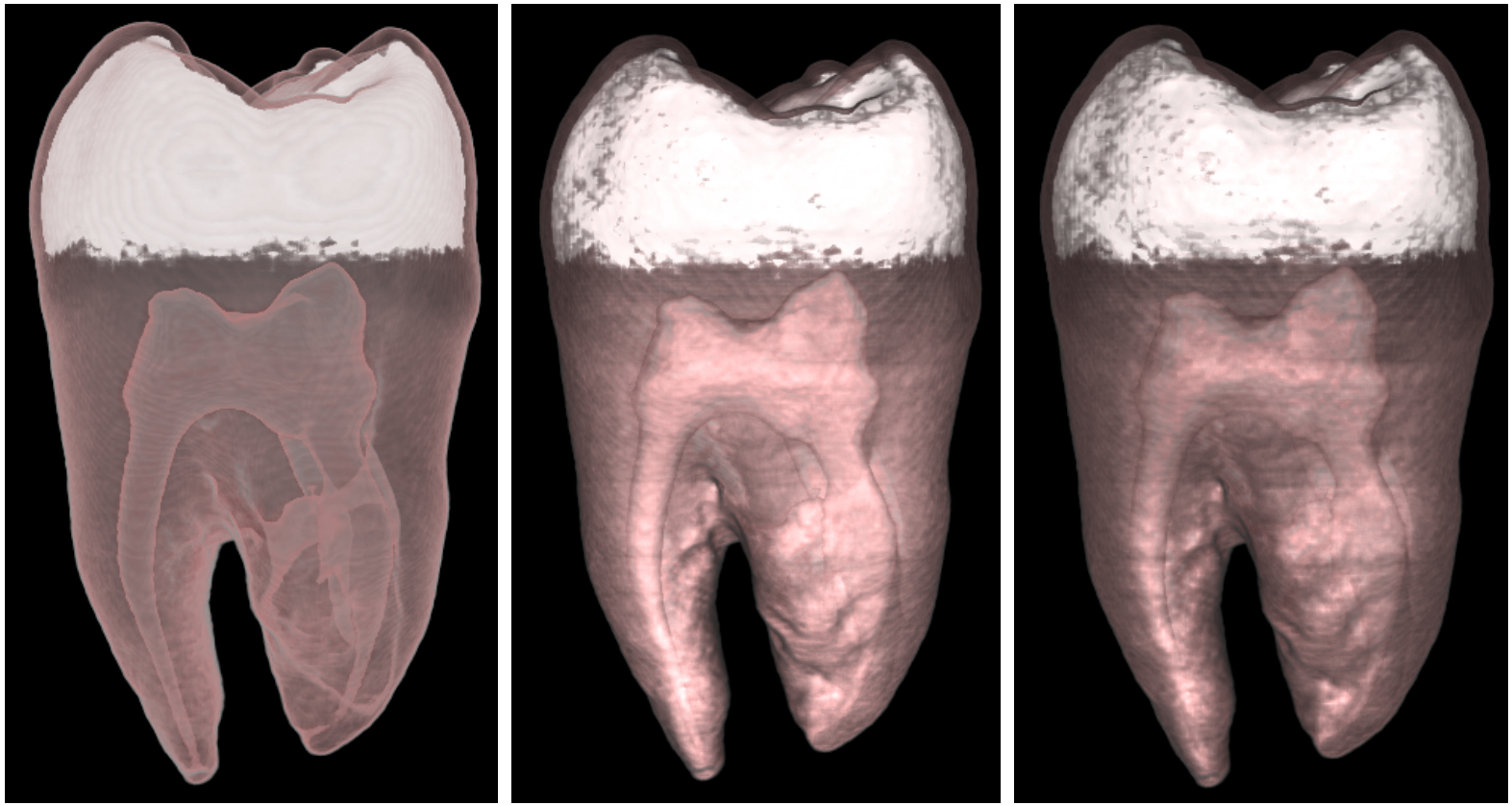
[4]

Shading volumétrique

Dans le résultat suivant, la fonction de transfert met l'emphase sur l'**émail** (blanc, opaque) et sur la **dentine** (brun, translucide).

La première image est obtenue **sans utiliser de shading volumétrique**. La deuxième se contente de l'utilisation de la **réflexion diffuse**, mais omet le spéculaire, tandis que la troisième a été obtenue en combinant les **réflexions diffuse et spéculaire**.

Shading



a) b) c) [4]

Espace image et espace objet

1 Introduction

2 Notions de base en visualisation de volumes

3 Espace image et espace objet
- Rendu en espace image
- Rendu en espace objet
- Comparaison avec le rendu géométrique

4 Calcul d'enveloppe convexe

5 Références

Rendu en espace image

1 Introduction

2 Notions de base en visualisation de volumes

3 Espace image et espace objet
- Rendu en espace image
- Rendu en espace objet
- Comparaison avec le rendu géométrique

4 Calcul d'enveloppe convexe

5 Références

Fonctionnement

On se souvient que l'intensité $I(p)$ d'un pixel est donnée par l'intégrale

$$I(p) = \int_0^1 c(t) e^{-\int_0^t \tau(x)\ dx}\ dt.$$

La façon la plus directe d'évaluer cette équation est d'**estimer la valeur de l'intégrale** en ne considérant que des échantillons le long du rayon.

Fonctionnement

Comme cette approche traite les pixels de l'image un par un, on dira qu'elle opère en **espace image**. En plus de facilement permettre la parallélisation des calculs, cette méthode laisse place à une certaine accélération en optimisant l'évaluation de l'intégrale.

En premier lieu, si on discrétise le rayon en utilisant **des pas de longueur** δ, alors l'équation à évaluer devient

$$I(p) = \sum_{i=0}^{N} c(i\delta) e^{-\sum_{j=0}^{i-1} \tau(j\delta)\delta} \delta.$$

Fonctionnement

On peut aussi remplacer l'exponentielle d'une somme par un produit d'exponentielles plus simples :

$$I(p) = \sum_{i=0}^{N} c(i\delta) \left(\prod_{j=0}^{i-1} e^{-\tau(j\delta)\delta} \right) \delta.$$

De plus, si le pas δ est assez petit, on peut utiliser **le premier terme de la décomposition de Taylor** pour fournir une approximation assez juste des exponentielles. La formule de Taylor dit que $e^{-t} \approx 1 - t$ pour de petites valeurs de t. On a donc

$$I(p) = \sum_{i=0}^{N} c(i\delta) \left(\prod_{j=0}^{i-1} (1 - \tau(j\delta)\delta) \right) \delta.$$

Fonctionnement

Pour alléger la notation, posons $c_i = c(i\delta)\delta$ et $\tau_j = \tau(j\delta)\delta$. On a alors

$$I(p) = \sum_{i=0}^{N} c_i \left(\prod_{j=0}^{i-1} (1 - \tau_j) \right) .$$

Il devient alors facile d'évaluer cette équation **de l'arrière du volume vers l'avant**, c'est-à-dire en inversant le sens de la somme.

Fonctionnement

Notons par C_i la couleur accumulée à la i-*ième* position et qui est **le résultat des contributions des points échantillonnés** aux position i à N. On a que

$$\begin{aligned} C_N &= c_N \\ C_{N-1} &= c_{N-1} + (1 - \tau_{N-1})c_N \\ &\vdots \\ I(p) = C_0 &= c_0 + (1 - \tau_0)c_1 + (1 - \tau_0)(1 - \tau_1)c_1 + \cdots \end{aligned}$$

Fonctionnement

Ainsi, de manière générale, on a

$$C_i = c_i + (1 - \tau_i) C_{i+1}.$$

Cette équation n'est rien de plus que la **mise en commun des couleurs** c_i multipliées par les opacités τ_i.

En effet, on a que la couleur accumulée en un point échantillonné i est donnée par **la couleur émise** par le point i, notée c_i, à laquelle on additionne **la contribution des points plus éloignés** C_{i+1}, multipliée par le coefficient de transparence $1 - \tau_i$.

Échantillonnage et interpolation

La qualité du rendu volumétrique dépend de notre capacité à fournir une **bonne estimation de l'intégrale originale** en discrétisant le rayon.

Deux facteurs sont particulièrement importants et peuvent avoir un grand impact sur le résulta visuel :

- le choix du **pas d'incrémentation** δ ;
- l'**interpolation de la couleur** c **et de l'opacité** τ le long du rayon.

Échantillonnage et interpolation

On peut s'attendre à ce que le choix d'**un plus petit pas d'incrémentation** améliore la qualité du rendu. Une telle décision aura toutefois un impact sur le temps de calcul nécessaire pour obtenir celui-ci.

Une meilleure stratégie consiste à détermine la valeur de δ **en fonction des variations dans l'ensemble de données**. Cela revient à prendre des plus petits pas lorsque les données varient rapidement, mais à espacer davantage les échantillons dans les régions constantes.

Échantillonnage et interpolation

Comme les valeurs de c et de τ **dépendent aussi de la fonction de transfert**, il est envisageable de faire intervenir les caractéristiques de celle-ci afin de mieux déterminer la valeur de δ.

Un bon point de départ est souvent de fixer δ à **une valeur proche de la taille d'un voxel**, de manière à s'assurer de la contribution de chacun de ceux-ci au résultat final $I(p)$.

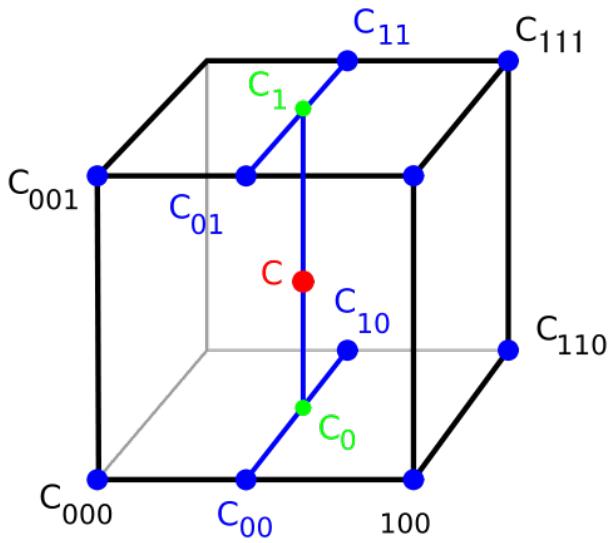
Échantillonnage et interpolation

De plus, comme les points échantillonnés le long du rayon ne coïncideront pas, en général, avec les centres des voxels, il sera nécessaire d'**effectuer une interpolation** pour évaluer c_i et τ_i.

La solution la plus simple, comme toujours lorsqu'il est question d'interpolation, consiste à récupérer la couleur et l'opacité associées au centre du voxel le plus près. Cela dit, l'**interpolation par le plus proche voisin** a le grand défaut de donner un signal discontinu qui résultera en une piètre qualité d'image.

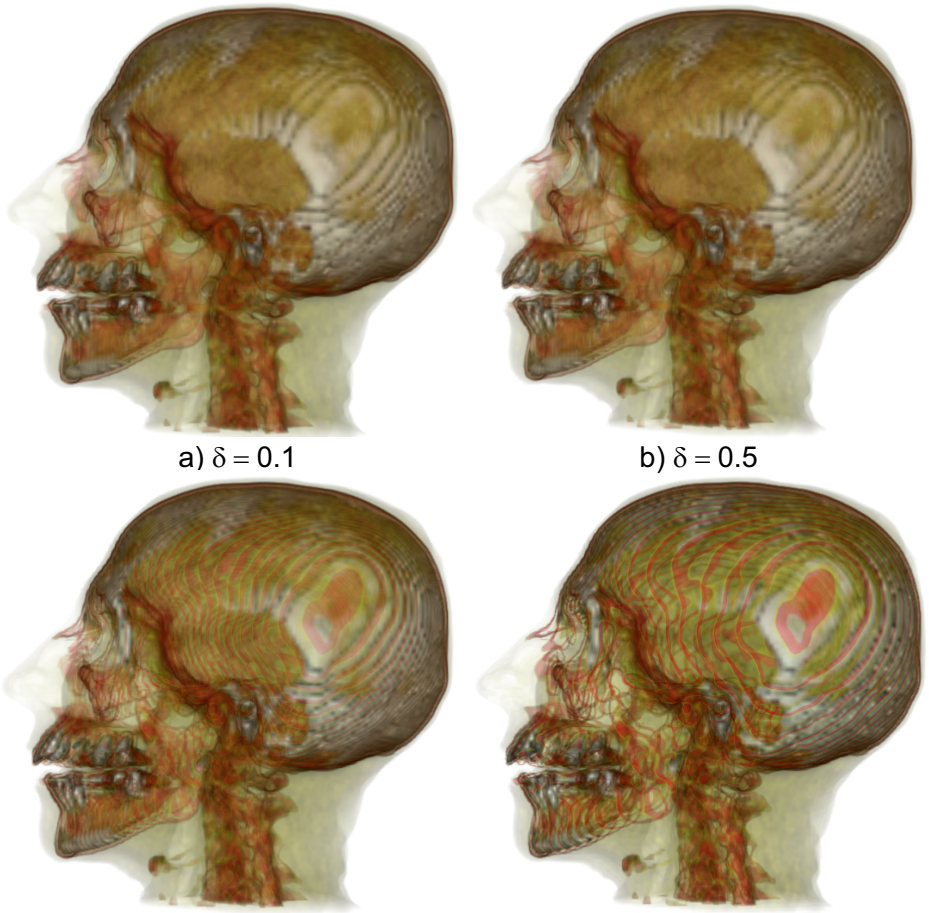
Échantillonnage et interpolation

Une meilleure solution consiste à utiliser l'**interpolation trilinéaire**.



Pour un même échantillonnage, cette méthode produit un bien meilleur résultat, tout en maintenant un coût raisonnable en temps de calcul.

Échantillonnage et interpolation



a) $\delta = 0.1$ b) $\delta = 0.5$

c) $\delta = 1.0$ d) $\delta = 2.0$

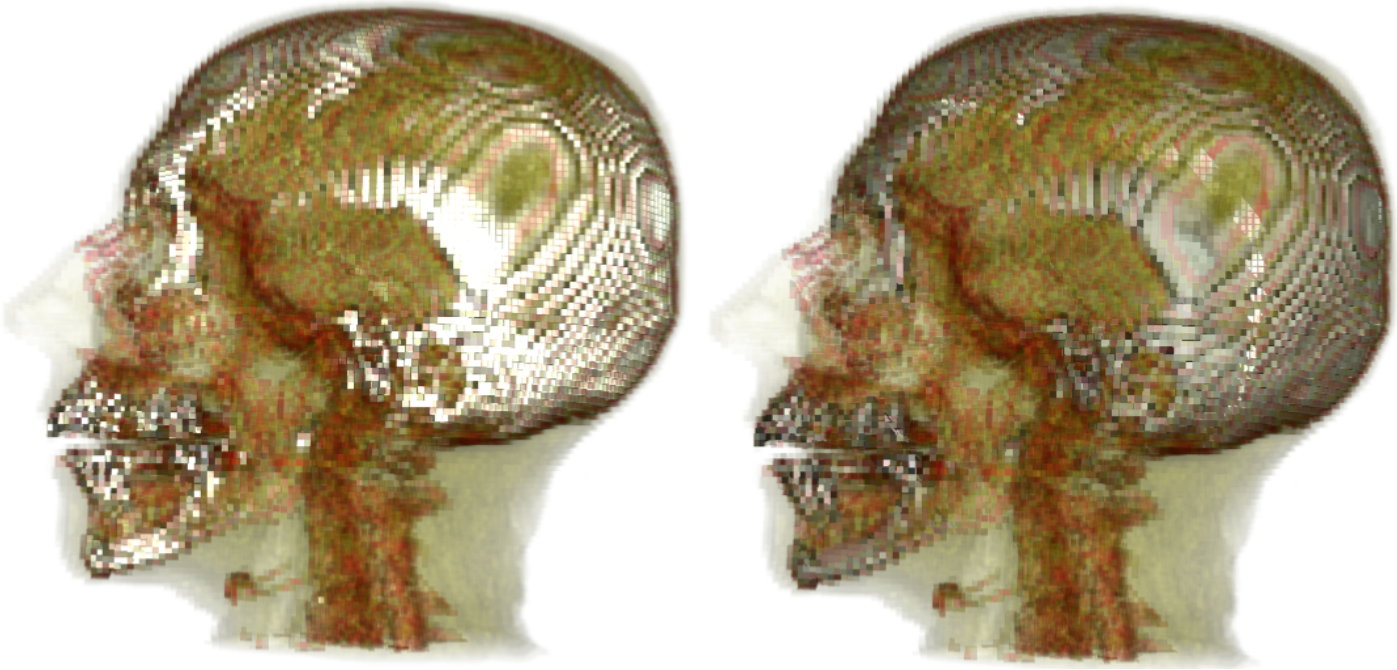
[4]

Échantillonnage et interpolation

Le résultat précédent illustre un rendu volumétrique en utilisant l'interpolation trilinéaire **pour différentes valeurs de** δ. Plus le pas d'incrémentation augmente, moins bon est le résultat, car l'approximation de l'intégrale avec la formule de Taylor devient moins valide.

Le résultat suivant présente quant à lui le rendu du même volume, mais cette fois-ci en utilisant l'**interpolation par le plus proche voisin**.

Échantillonnage et interpolation



a) $\delta = 0.1$ b) $\delta = 1.0$ [4]

Classification vs. interpolation

Si on utilise l'interpolation trilinéaire (ou toute autre technique reposant sur une moyenne de valeurs), on dispose de **deux options pour évaluer la couleur et l'opacité** :

- la **pré-classification** : classifier, puis interpoler ;
- la **post-classification** : interpoler, puis clasifier.

L'étape de classification repose essentiellement sur l'application de la fonction de transfert.

Classification vs. interpolation

Plus spécifiquement, avec la **pré-classification**, on applique la fonction de transfert sur les données originales pour obtenir les couleurs et les opacités associées aux centres des voxels concernés, puis on effectue l'interpolation sur celles-ci.

À l'opposé, avec la **post-classification**, on effectue l'interpolation sur les données originales, puis on applique la fonction de transfert sur la valeur scalaire interpolée.

Classification vs. interpolation

En général, l'utilisation de la pré-classification produira des images plus rugueuses où **les transitions entre les intensités seront plus brusques**.

De plus, l'interpolation des couleurs peut engendrer des résultats inattendus, en produisant par exemple **des couleurs qui ne se trouvent même pas dans la table de correspondance** prévue par la fonction de transfert.

Classification vs. interpolation

Quant à elle, la post-classification produira des images où **les transitions de couleurs seront moins marquées**, tout en garantissant que **les couleurs présentes dans l'image seront valides**.

Toutefois, comme cette approche repose sur l'interpolation linéaire de valeurs scalaires dans l'ensemble de données, elle revient à faire la supposition que **le signal original est localement continu** entre les centres des voxels.

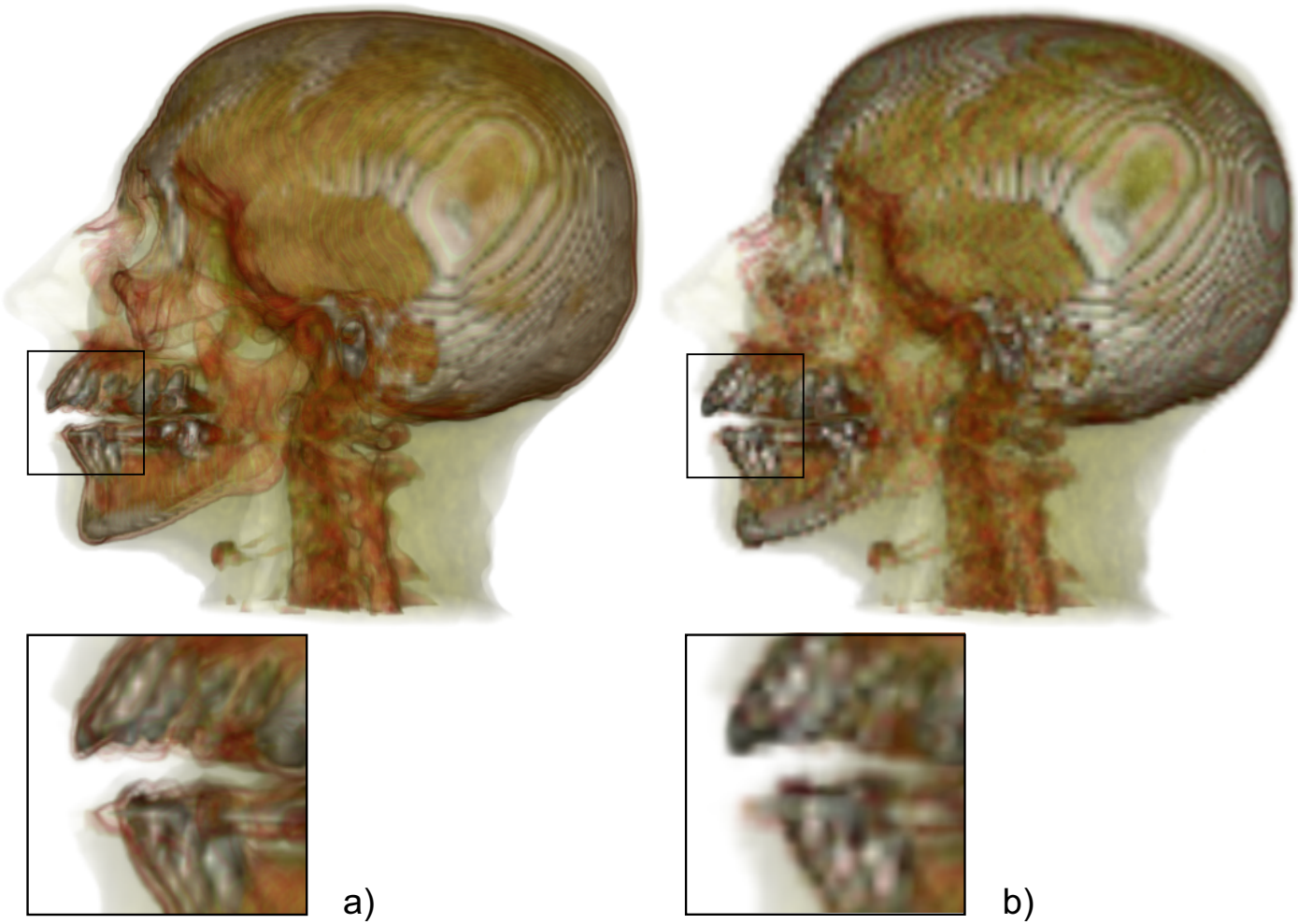
Classification vs. interpolation

La conséquence de cette supposition, si elle s'avérait fausse, serait que **les valeurs scalaires interpolées correspondraient à des matériaux inexistants** aux points où l'ensemble de données présente d'importantes discontinuités.

Notons toutefois que lorsqu'on utilise des **fonctions de transferts possédant des transitions douces**, alors les résultats obtenus par les deux approches seront visuellement très similaires.

La figure suivante illustre les résultats de la post-classification (a) et de la pré-classification (b).

Classification vs. interpolation



a) b)

[4]

Rendu en espace objet

1 Introduction

2 Notions de base en visualisation de volumes

3 Espace image et espace objet
- Rendu en espace image
- Rendu en espace objet
- Comparaison avec le rendu géométrique

4 Calcul d'enveloppe convexe

5 Références

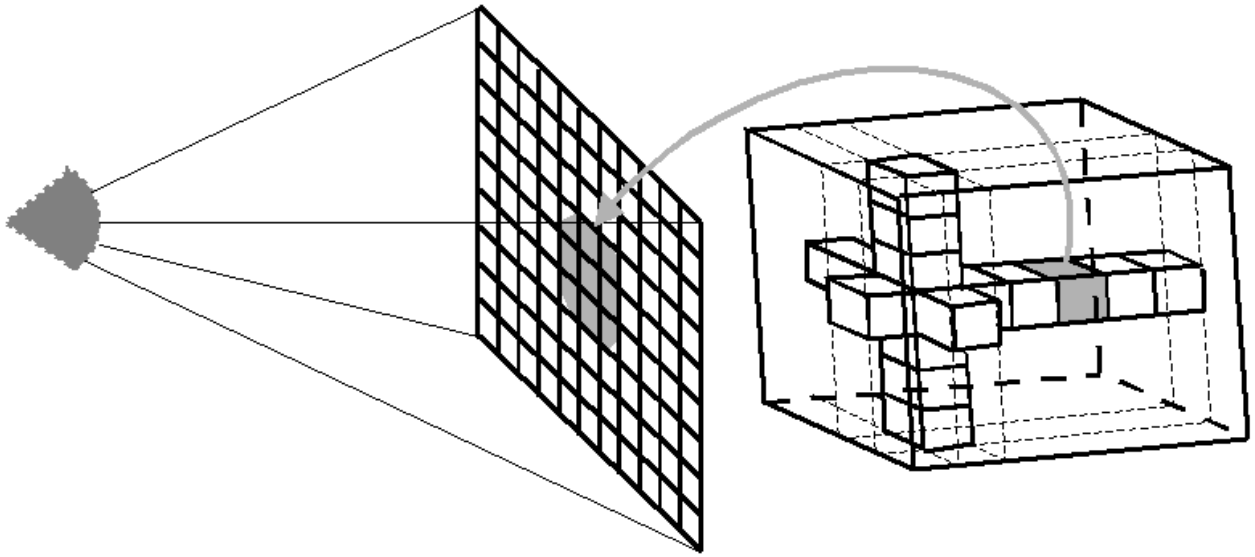
Fonctionnement

Une deuxième approche pour l'implantation des équations de rendu volumétrique propose plutôt de fonctionner **en espace objet**.

À l'opposé des algorithmes en espace image qui s'attardent une seule fois à chaque pixel, les techniques en espace objet **traversent chaque voxel du volume une seule fois**.

Fonctionnement

En passant par un voxel, l'algorithme évalue **sa contribution dans les pixels dont le rayon l'intersecte**.



[3]

Fonctionnement

Ainsi, alors que les algorithmes en espace image visitaient chaque pixel une seule fois, ceux fontionnant en espace objet **visiteront un même pixel à plusieurs reprises**.

Le nombre de visites dépendra du nombre de voxels qui contribuent à la couleur finale du pixel.

Fonctionnement

Une des méthodes les plus populaires en rendu volumétrique en espace objet fait appel aux accélérations matérielles associées au **rendu de textures**.

L'idée est de nouveau d'évaluer l'intégrale permettant d'obtenir $I(p)$ de l'arrière du volume vers l'avant par l'équation

$$C_i = c_i + (1 - \tau_i) C_{i+1},$$

mais cette fois-ci en utilisant les **fonctions de blending fournies par une librairie graphique** (OpenGL, par exemple).

Fonctionnement

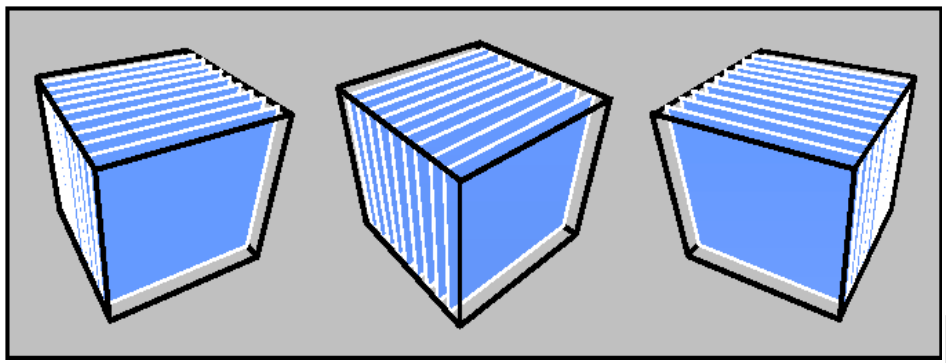
Selon **le type de texture supporté** par la carte graphique, on optera pour l'une ou l'autre des sous-classes suivantes d'algorithmes en en espace objet :

- Rendu volumétrique **par tranches** (textures 2D)
- Rendu volumétrique **par bloc** (textures 3D)

Rendu volumétrique par tranches

La première étape dans le **rendu volumétrique par tranches** est de découper le volume de données en un ensemble de plans orthogonaux à l'**axe du volume qui se rapproche le plus de la direction d'observation**.

On obtient ainsi une séquence de rectangles également espacés le long de l'axe de découpage.



[3]

Rendu volumétrique par tranches

On assigne par la suite à chaque rectangle une **texture correspondant aux voxels** d'où proviennent les rectangles. Les couleurs RGBA de la texture sont obtenues en appliquant la fonction de transfert aux valeurs scalaires des voxels.

Finalement, les rectangles texturés sont affichés de l'arrière vers l'avant en activant l'option

```
glBlendFunc(GL_ONE, GL_ONE_MINUS_SRC_ALPHA).
```

Rendu volumétrique par tranches

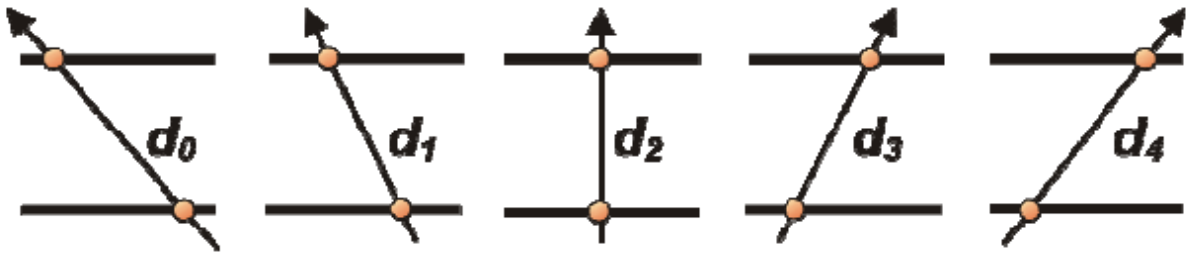
Comme on peut le constater, le rendu volumétrique par tranches est à la fois **simple à implanter** et **très efficace**, puisqu'il fait appel aux accélérations matérielles sans devoir fournir d'effort particulier.

Il est donc assurément plus rapide à exécuter que les techniques basées sur le lancer de rayons. Toutefois, **cette méthode ne possède pas que des avantages**...

Rendu volumétrique par tranches

Son principal défaut est sans nul doute que **la qualité du rendu final dépend de la direction d'observation**.

En effet, à mesure qu'on s'éloigne d'un point de vue parallèle à un des axes du volume, **la distance δ entre les tranches de texture augmente**.



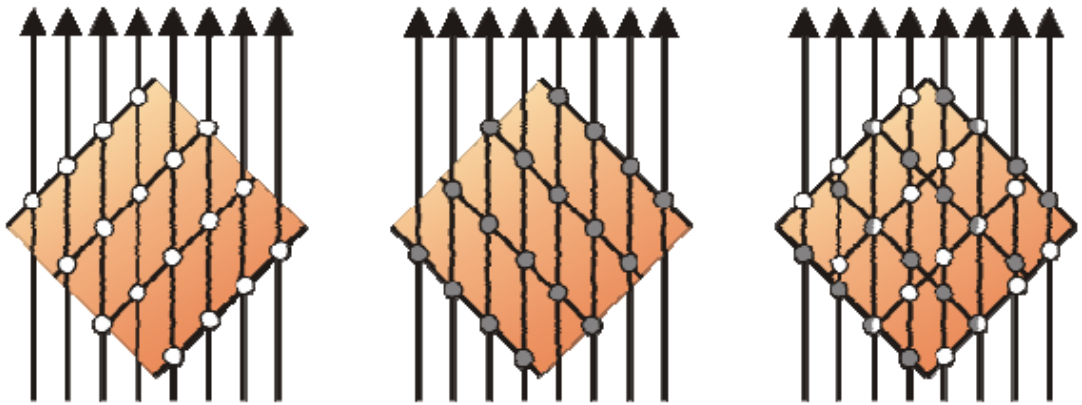
[3]

Cette distance δ entre les plans de découpage correspond à **la notion de pas d'incrémentation** dans le rendu en espace image.

Rendu volumétrique par tranches

Un autre problème survient lorsque **l'angle d'observation s'approche de** 45° par rapport à un axe du volume de données.

À un tel angle, le moindre changement peut avoir un impact sur **l'axe de découpage qui est utilisé**, ce qui aura une influence sur le rendu final, puisque les points d'intersection avec les tranches seront modifiés.



[3]

Rendu volumétrique par bloc

Une meilleure option est d'utiliser le **rendu volumétrique par bloc** qui fait appel aux textures 3D, si celles-ci sont supportées par le matériel graphique disponible.

Le principe est sensiblement le même qu'avec les textures 2D, mais **on élimine le principal inconvénient** de la méthode précédente, soit celui relié aux directions d'observation.

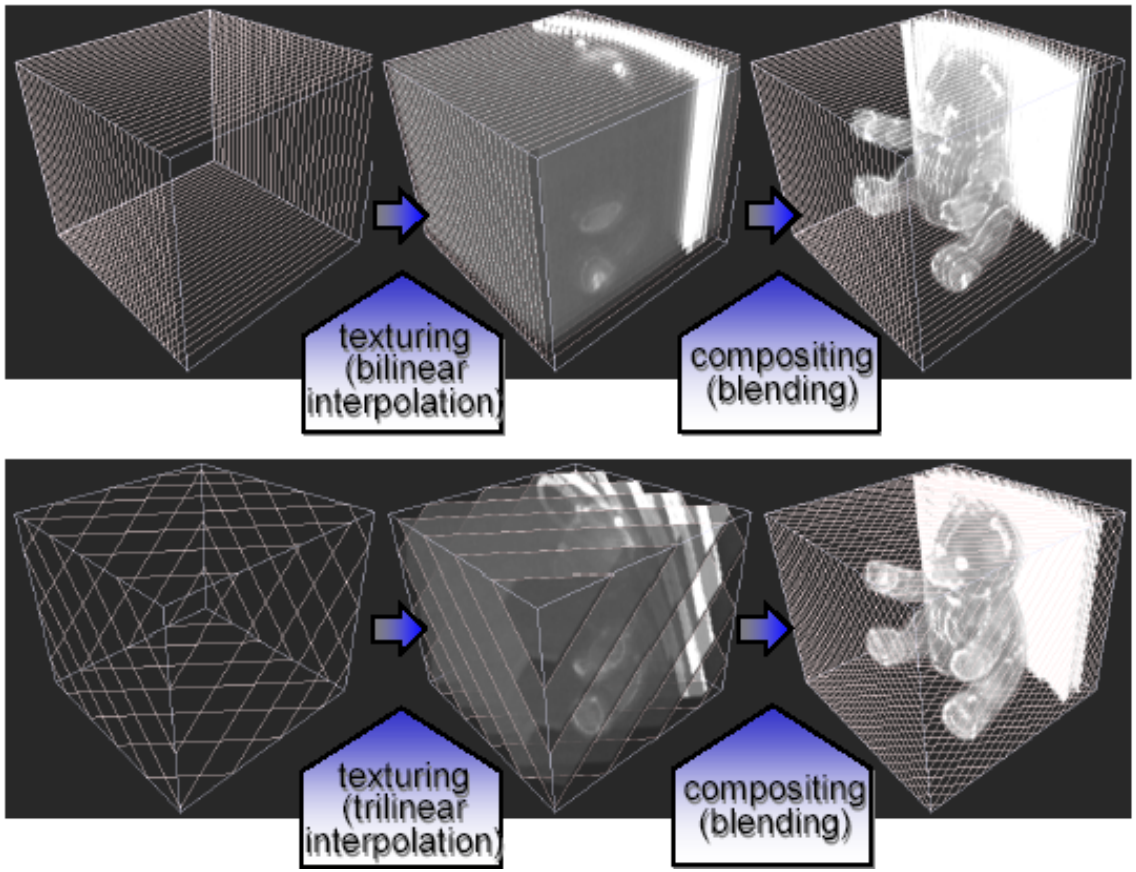
Rendu volumétrique par bloc

La première étape consiste à charger le volume de données en mémoire et de **créer un bloc de texture 3D RGBA** en appliquant la fonction de transfert sur la totalité des données.

Par la suite, **on tranche ce bloc** en un ensemble de plans perpendiculaires à la direction d'observation.

Le résultat sera un ensemble de polygones ayant des formes différentes : des triangles, des quadrilatères, des pentagones et des hexagones.

Rendu volumétrique par bloc



2D textures
axis-aligned

3D texture
view-aligned

[3]

Rendu volumétrique par bloc

Finalement, le rendu final est obtenu en **traitant les polygones de l'arrière vers l'avant** à l'aide des fonctionnalités de blending de la librairie graphique.

Ainsi, même si le fonctionnement est le même que dans le cas 2D, on s'assure d'un meilleur résultat, puisque **les tranches seront toujours parallèles au plan image**.

Rendu volumétrique par bloc

Même si les deux techniques de rendu en espace objets précédentes sont plus rapides que les approches basées sur le lancer de rayons, **elles présentent d'importantes limitations** qui s'appliquent autant au cas 2D qu'au cas 3D.

La première est que **la taille maximale de l'ensemble de données est limitée** par la quantité de mémoire disponible sur la carte graphique.

Rendu volumétrique par bloc

À titre d'exemple, une carte graphique dotée de 256 Mb de mémoire serait **limitée à des textures 3D RGBA de dimension** $512 \times 512 \times 256$ si on alloue 8 bits par composantes. De manière équivalente, elle serait aussi limitée à 256 textures 2D comptant 512^2 texels.

Une solution sera de **découper la texture 3D en plus petits blocs** et de les traiter en séquence, ce qui nécessiterait toutefois davantage d'échanges avec la mémoire vidéo.

Rendu volumétrique par bloc

Un autre défaut de ces approches est de **ne supporter que la pré classification**, c'est-à-dire que la fonction de transfert est d'abord appliquée sur les données originales.

En effectuant le blending des couleurs, on s'expose au même problème que précédemment, soit celui d'**obtenir des couleurs qui ne se trouvent pas dans la table de correspondance**.

Comparaison avec le rendu géométrique

1 Introduction

2 Notions de base en visualisation de volumes

3 Espace image et espace objet
- Rendu en espace image
- Rendu en espace objet
- Comparaison avec le rendu géométrique

4 Calcul d'enveloppe convexe

5 Références

Objectifs

Les techniques de rendu volumétrique présentées dans ce chapitre ont **beaucoup de points en commun avec des méthodes de rendu géométrique** comme les Marching Cubes.

Dans cette section, on s'intéressera à détailler ces points communs, mais surtout à **identifier les différences entre les deux familles d'approches**.

Objectifs

Premièrement, le but visé par les deux types de techniques est souvent le même : **produire une image à partir d'un ensemble de données scalaires 3D** afin de visualier l'information qu'il contient.

Ainsi, les résultats obtenus avec l'une ou l'autre des approches **peuvent être très similaires**, comme on a pu le constater avec les isosurfaces qui peuvent être obtenues de plusieurs façons.

Objectifs

En effectuant le rendu géométrique de **plusieurs isosurfaces ayant des opacités différentes** sur une même image, on s'approche du type de résultat obtenu avec le lancer de rayons.

Cependant, un des avantages du rendu volumétrique est qu'à chaque pixel de l'image, on affiche **de l'informations provenant de l'ensemble du volume** et non pas seulement de quelques positions discrètes dans celui-ci.

Complexité

Pour comparer **la complexité des deux familles d'approches**, on s'attarde de nouveau à l'exemple du calcul des isosurfaces. À cette même tâche, quelle méthode est la plus efficace ?

Premièrement, notons que la complexité des deux techniques varie **linéairement en fonction de la taille de l'ensemble de données**. Les différences seront plutôt reliées au calcul de l'image à afficher.

Complexité

La méthode des Marching Cubes représente une isosurface sous la forme d'un maillage de polygones et son rendu visuel est ensuite **évalué à partir du point de vue désiré**.

À l'opposé, le lancer de rayons devra **reprendre les calculs du début** si la position de l'observateur devait être modifiée.

Complexité

Un autre avantage du rendu géométrique est de fournir une structure géométrique **dont les propriétés ne dépendent pas de la résolution** de la fenêtre de rendu. La performance des Marching Cubes n'est ainsi pas influencée par le nombre de pixels à évaluer.

Contrairement à ça, les performances du lancer de rayons **dépendent grandement de la quantité de pixels** à afficher, puisqu'un rayon doit être évalué pour chaque pixel.

Calcul d'enveloppe convexe

1 Introduction

2 Notions de base en visualisation de volumes

3 Espace image et espace objet

4 Calcul d'enveloppe convexe
- Enveloppe convexe sur un plan
- Enveloppe convexe 3D

5 Références

Enveloppe convexe sur un plan

1 Introduction

2 Notions de base en visualisation de volumes

3 Espace image et espace objet

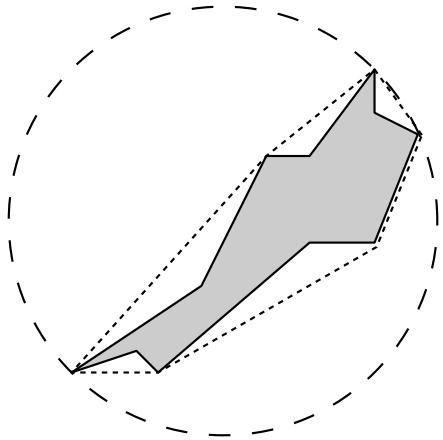
4 Calcul d'enveloppe convexe
- Enveloppe convexe sur un plan
- Enveloppe convexe 3D

5 Références

Mise en contexte

Supposons que l'on souhaite vérifier **si deux objets P_1 et P_2 sont en collision** l'un avec l'autre, mais que P_1 et P_2 possèdent des géométries complexes.

Une bonne approche consiste à **approximer P_1 et P_2 par des objets plus simples** $\hat{P}_1$ et $\hat{P}_2$ qui englobent les originaux.



[1]

Mise en contexte

Pour savoir si une intersection existe entre P_1 et P_2, **on vérifie d'abord si $\hat{P}_1$ et $\hat{P}_2$ possèdent une intersection**. Seulement dans ce cas on fera le calcul détaillé pour connaître la réelle intersection entre P_1 et P_2.

Idéalement, le calcul d'intersection entre $\hat{P}_1$ et $\hat{P}_2$ sera rapide en raison de leur simplicité géométrique.

Mise en contexte

Un compromis doit être fait lorsqu'on choisit **la forme des objets englobants** $\hat{P}_1$ et $\hat{P}_2$:

- On souhaite que **ces objets soient simples** de manière à faciliter le calcul d'intersection préliminaire.
- On souhaite qu'ils **fournissent une bonne approximation** des vrais objets afin de limiter les faux positifs dans la détection préliminaire.

Mise en contexte

À une extrémité du spectre, on retrouve **les sphères et les boîtes englobantes**. Ils sont géométriquement simples et les tests d'intersection entre ceux-ci sont faciles à faire.

Ils fournissent toutefois assez souvent une approximation grossière des objets réels.

Mise en contexte

À l'opposé, on retrouve **les enveloppes convexes** qui offrent une bien meilleure approximation des objets réels qu'une sphère ou un prisme.

Le coût des calculs d'intersection est toutefois nettement plus grand qu'avec des volumes simples.

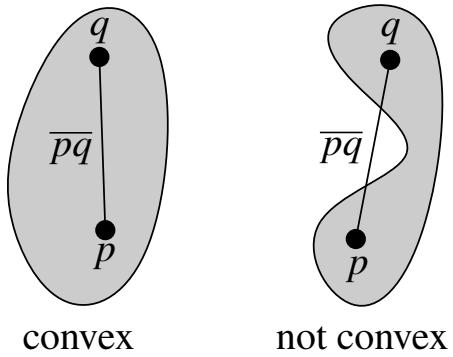
Mise en contexte

Dans ce chapitre, on s'intéressera au **calcul des enveloppes convexes**, d'abord dans le cas 2D pour un ensemble de points sur le plan $\mathbb{R}^2$, puis dans le cas 3D pour un ensemble de points dans l'espace $\mathbb{R}^3$.

Enveloppe convexe dans $\mathbb{R}^2$

On définit l'**enveloppe convexe** (*convex hull*) d'un ensemble de points P de $\mathbb{R}^2$ comme étant la plus petite région convexe de $\mathbb{R}^2$ contenant tous les points de P.

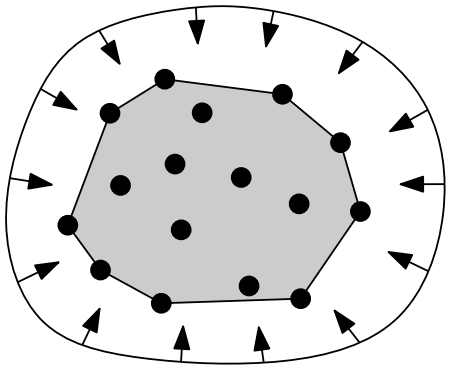
Une région S de $\mathbb{R}^2$ est dite **convexe** si et seulement si pour toute paire de points $p, q \in S$, le segment $\overline{pq}$ est entièrement contenu dans la région S.



[1]

Enveloppe convexe dans $\mathbb{R}^2$

Soit P un ensemble de n points sur le plan. Imaginons que **chacun de ces points donne la position d'un clou**. Si l'on place un élastique autour de tous les clous et qu'on le relâche, on obtiendra l'enveloppe convexe associée à P.



[1]

L'enveloppe convexe est l'unique polygone convexe **contenant tous les points de** P et **dont les sommets sont aussi des points de** P.

Enveloppe convexe dans $\mathbb{R}^2$

Une façon naturelle de représenter un polygone est de donner **une liste de ses sommets** dans le sens horaire.

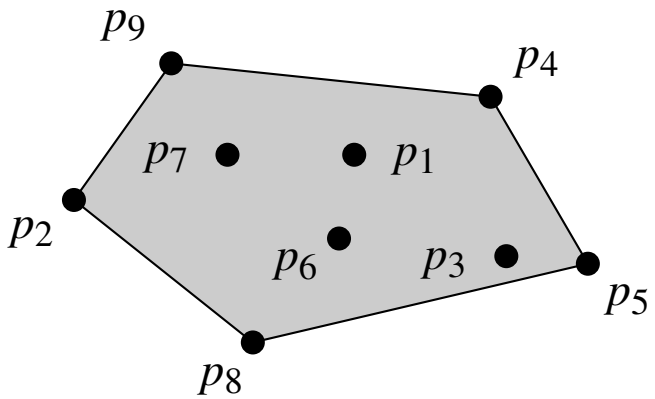
Notre objectif est donc, à partir d'un ensemble de points $P = \{p_1, p_2, \ldots, p_n\}$ d'obtenir une liste $\mathcal{L}$ de points de P contenant **les sommets de l'enveloppe convexe** $\mathcal{CH}(P)$ dans le sens horaire.

input = set of points:

$p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9$

output = representation of the convex hull:

p_4, p_5, p_8, p_2, p_9

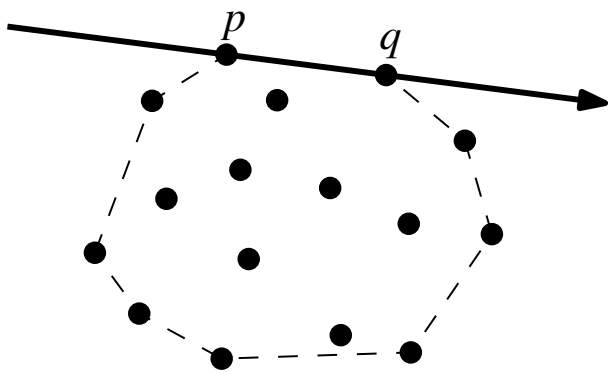


[1]

Enveloppe convexe dans $\mathbb{R}^2$

Soit une arête faisant partie de $\mathcal{CH}(P)$. On aura que **ses deux extrémités sont aussi des points de** P.

Si on trace la droite de support au segment $\overline{pq}$ de sorte que $\mathcal{CH}(P)$ soit à droite de la droite, alors **il en sera de même pour tous les points de** P.



[1]

Enveloppe convexe dans $\mathbb{R}^2$

L'affirmation inverse est aussi vraie : si tous les points de $P \setminus \{p, q\}$ sont situés à la droite de la droite de support du segment $\overline{pq}$, **alors celui-ci fait partie de** $\mathcal{CH}(P)$.

Maintenant que notre compréhension géométrique du problème est adéquate, on peut **élaborer un algorithme qui fera ce travail**.

Enveloppe convexe dans $\mathbb{R}^2$

Algorithme 1: EnveloppeConvexe2D

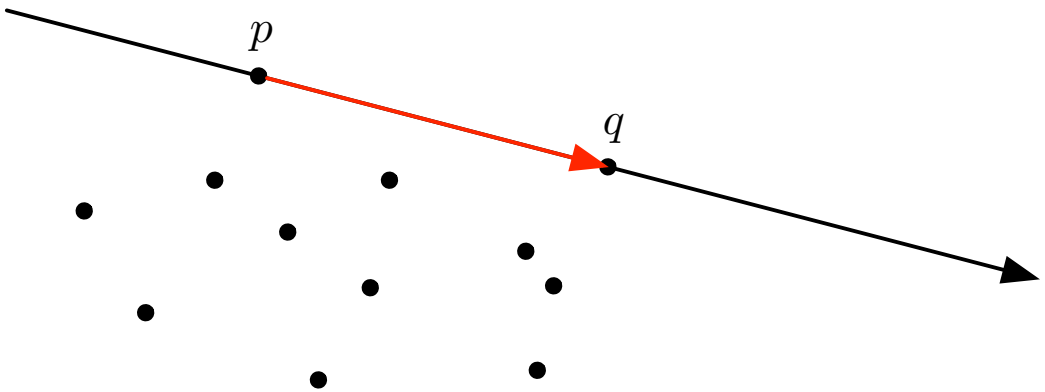
Entrées : Un ensemble $P = \{p_1, p_2, \ldots, p_n\}$ où $p_i \in \mathbb{R}^2$
Sorties : Une liste $\mathcal{L}$ contenant les sommets de l'enveloppe convexe $\mathcal{CH}(P)$ dans le sens horaire

```
1 E ← ∅;
2 pour chaque couple de points (p, q) ∈ P × P avec p ≠ q faire
3     SegmentEstValide ← vrai;
4     pour chaque point r ∈ P avec r ≠ p et r ≠ q faire
5         si r se trouve à la gauche du segment reliant p à q alors
6             SegmentEstValide ← faux;
7     si SegmentEstValide alors
8         Ajouter le segment [pq] à E;
9 À partir de l'ensemble E, construire la liste L des sommets de CH(P)
  dans le sens horaire;
```

Enveloppe convexe dans $\mathbb{R}^2$

La dernière étape (ligne 9) nécessire quelques explications.

Comme les arêtes sont orientées de telle sorte que le reste des points se trouve à leur droite, on aura nécessairement que **le point d'arrivée d'un segment viendra après son point de départ** lorsqu'ils sont listés en sens horaire.



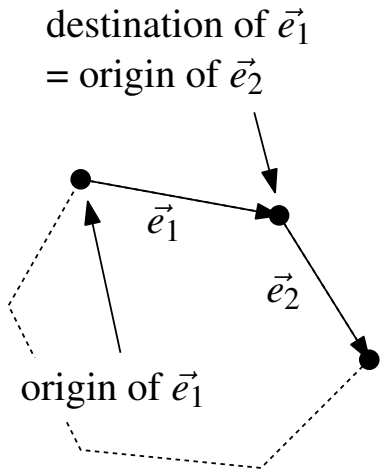
Enveloppe convexe dans $\mathbb{R}^2$

Supposons que l'ensemble E est composé des segments $e_1, e_2, \ldots e_n$. **On retire un segment quelconque de** E, disons e_1, et on ajoute son origine comme premier point de $\mathcal{L}$ et son point d'arrivée comme deuxième.

On part ensuite à la recherche du segment e_i dans E **dont l'origine est le point d'arrivée de** e_1.

Enveloppe convexe dans $\mathbb{R}^2$

On retire ce segment de E et **on ajoute son point d'arrivée à** $\mathcal{L}$.



[1]

On répète ce principe **jusqu'à ce qu'il ne reste qu'une seule arête dans** E, et le point d'arrivée de celle-ci sera nécessairement l'origine de e_1, qui se trouve déjà dans $\mathcal{L}$.

Constat d'échec

Analysons la complexité de l'algorithme EnveloppeConvexe2D.

On considère $n(n-1) = n^2 - n$ **paires de points**, et pour chacune de celles-ci, on vérifie si les $n-2$ **points restants** se trouvent à la droite d'un segment de droite, ce qui donne une complexité de $O(n^3)$.

La dernière étape (ligne 9) s'éffectue en temps $O(n^2)$.

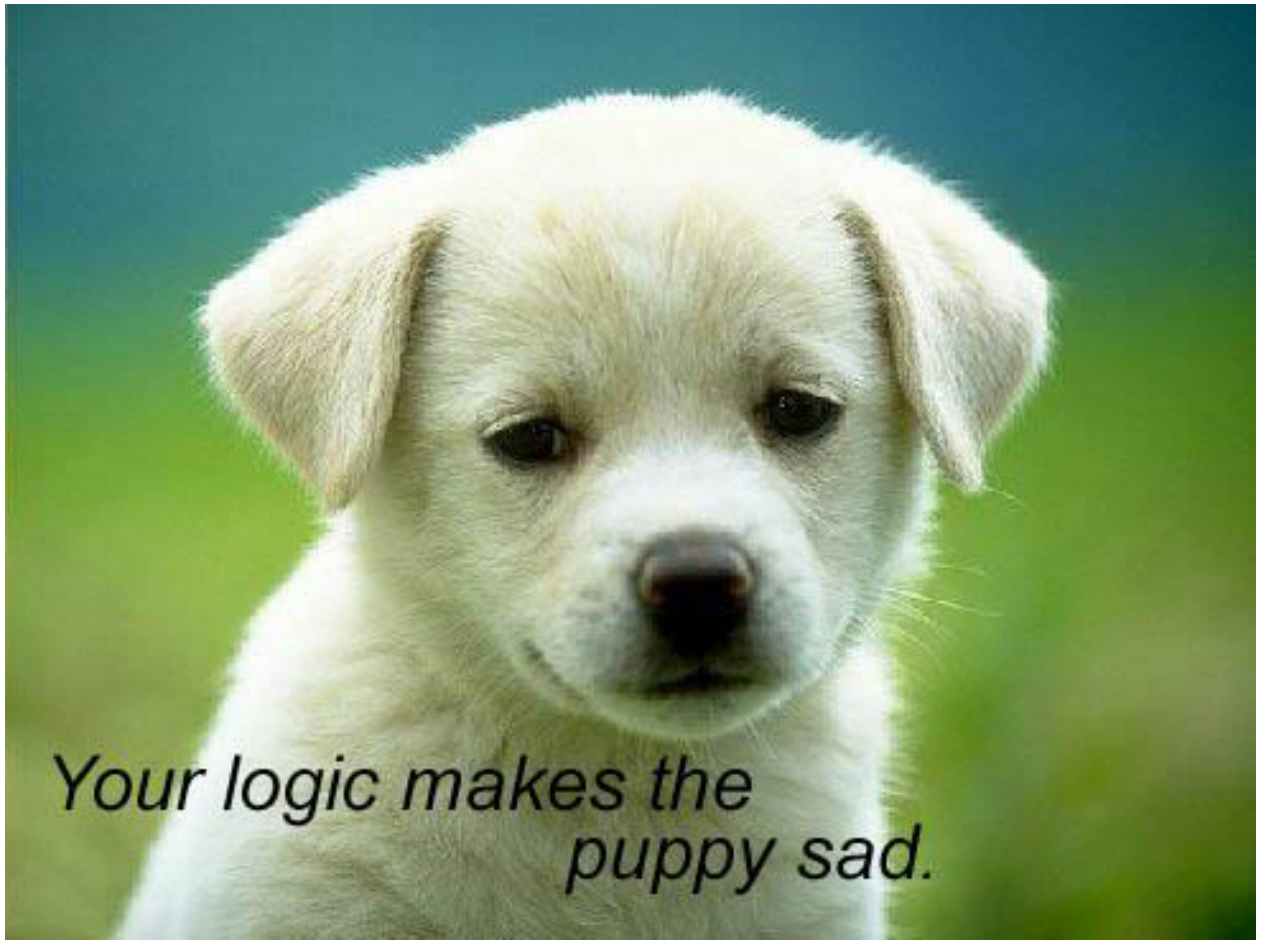
Constat d'échec

La **complexité totale** est ainsi $O(n^3)$, ce qui est **beaucoup trop lent** pour que l'algorithme soit utilisable avec autre chose que de minuscules ensembles de points.

L'erreur comise a été de **bêtement convertir une intuition géométrique en algorithme**... et si seulement c'était le seul problème de ce programme !

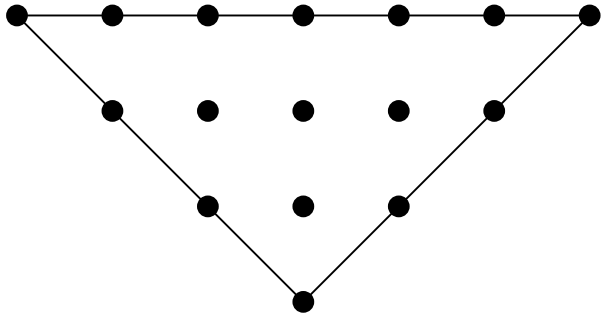
Nous ne sommes toutefois pas au bout de nos peines...

Constat d'échec



Constat d'échec

Considérons le segment défini par les points p et q. Un point r ne se trouve **pas nécessairement à gauche ou à droite** du segment... Il peut aussi être **directement sur celui-ci** !



[1]

Constat d'échec

Pour que l'algorithme demeure fonctionnel dans une telle situation, il est nécessaire de **reformuler le critère de la ligne 5**.

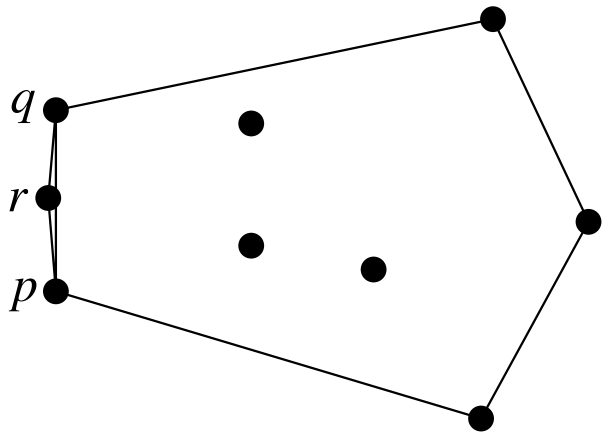
Un segment $\overline{pq}$ fera partie de $\mathcal{CH}(P)$ si et seulement si tous les autre points $r \in P$ se trouve **à la droite du segment** $\overline{pq}$ ou encore **directement sur celui-ci**.

On suppose ici que **tous les points de** P **sont différents**.

Constat d'échec

On aussi pris pour acquis qu'il était **toujours possible de tester si un point se trouvait à la gauche ou à la droite d'un segment**. Des erreurs d'arrondis peuvent toutefois venir nous jouer de vilains tours.

Considérons trois points, p, q et r, qui sont **presque colinéaires**.



[1]

Constat d'échec

L'algorithme précédent teste les segments $\overline{pq}$, $\overline{rq}$ et $\overline{pr}$. Comme les points sont presque colinéaires, on pourrait obtenir les résultats suivants **en raison d'erreurs d'arrondis** :

- r se trouve **à la droite** de $\overline{pq}$.
- p se trouve **à la droite** de $\overline{rq}$.
- q se trouve **à la droite** de $\overline{pr}$.

Dans ce cas, l'algorithme accepterait les trois segments **comme faisant partie de** $\mathcal{CH}(P)$... malgré l'impossibilité géométrique d'une telle situation.

Constat d'échec

Encore pire ! L'algorithme pourrait aussi fournir le résultat inverse et **rejeter les trois segments**, ce qui créerait une discontinuité dans l'enveloppe convexe.

Comme l'étape de la ligne 9 n'a pas été conçue pour gérer de telles inconsistances dans les données, **le programme risque de fournir des résultats désastreux**.

Constat d'échec

Résumons la situation... On a **un algorithme qui calcule l'enveloppe convexe d'un ensemble de points** dans $\mathbb{R}^2$ qui ...

1. ... est très lent ;
2. ... traite les cas dégénérés de manière curieuse ;
3. ... n'est pas robuste.

Constat d'échec

Enveloppe convexe dans $\mathbb{R}^2$ (Prise 2)

Pour obtenir une meilleure solution, on développera un **algorithme incrémental**, c'est-à-dire que **les points de P seront ajoutés un par un** et qu'on maintiendra à jour la solution à chaque étape.

L'ordre d'ajout sera **déterminé par la coordonnée en x des points**. La première étape consiste donc à trier les points de gauche à droite pour obtenir une liste ordonnée $p_1, \ldots, p_n$.

Enveloppe convexe dans $\mathbb{R}^2$ (Prise 2)

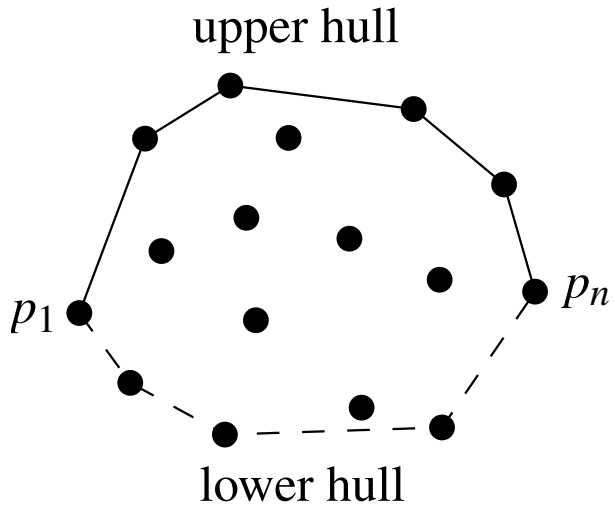
Comme nos points sont ordonnés de gauche à droite, il serait intéressant de **représenter les sommets de l'enveloppe convexe dans cet ordre**... ce qui ne fait malheureusement aucun sens !

Pour contourner ce problème, on sépare notre tâche en deux sous-tâches, soient le calcul de l'**enveloppe supérieure** et de l'**enveloppe inférieure**.

Enveloppe convexe dans $\mathbb{R}^2$ (Prise 2)

L'**enveloppe supérieure** est la section de l'enveloppe convexe partant du point le plus à gauche p_1 et se rendant au point le plus à droite p_n lorsque les sommets sont **listés dans le sens horaire**.

L'**enveloppe inférieure** est quant à elle calcule de droite à gauche et constitue la borne inférieure de l'enveloppe convexe.



[1]

Enveloppe convexe dans $\mathbb{R}^2$ (Prise 2)

L'étape centrale de l'algorithme incrémental est **la mise à jour de l'enveloppe supérieure** lors de l'ajout d'un point p_i.

En d'autres mots, connaissant l'enveloppe supérieure associée aux point $p_1, \ldots, p_{i-1}$, on doit **évaluer l'enveloppe supérieure associée aux points** $p_1, \ldots, p_i$.

Pour saisir les modalités d'ajout d'un point, prenons une marche en sens horaire sur le contour d'un polygone...

Enveloppe convexe dans $\mathbb{R}^2$ (Prise 2)

À chaque sommet que l'on rencontre durant cette promenade, **on effectue un virage**. Pour un polygone quelconque, il peut s'agir d'un virage à gauche ou d'un virage à droite.

Or, dans le cas d'un polygone convexe, **seuls les virages à droite sont autorisés**. Cette règle nous donne les indications nécessaires pour gérer l'ajout d'un nouveau point p_i.

Enveloppe convexe dans $\mathbb{R}^2$ (Prise 2)

Soit $\mathcal{L}_{\text{sup}}$, **une liste stockant les sommets de l'enveloppe supérieure** associée à $p_1, \ldots, p_{i-1}$, triés de gauche à droite. On ajoute le point p_i à $\mathcal{L}_{\text{sup}}$.

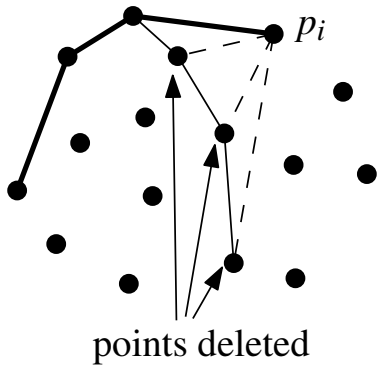
Comme p_i est le point le plus à droite rencontré jusqu'ici (puisque la liste des points est triée), **il fait nécessairement partie de l'enveloppe supérieure**.

Enveloppe convexe dans $\mathbb{R}^2$ (Prise 2)

Ensuite, on s'assure que les trois dernires points dans $\mathcal{L}_{\text{sup}}$ **forment un virage à droite**. Si c'est le cas, on ne change rien puisque $\mathcal{L}_{\text{sup}}$ ne contient que des sommets faisant partie de l'enveloppe supérieure associée à $p_1, \ldots, p_i$ et on passe au point p_{i+1}.

À l'opposé, si les trois derniers points **forment un virage à gauche**, on retire le point du milieu de $\mathcal{L}_{\text{sup}}$ et on reprend le test pour le nouveau groupe des trois derniers points.

Enveloppe convexe dans $\mathbb{R}^2$ (Prise 2)



[1]

On continue de cette manière **tant et aussi longtemps que les trois derniers points forment un virage à gauche**, ou jusqu'à ce qu'il ne reste plus que deux points dans $\mathcal{L}_{\text{sup}}$, puis on passe à p_{i+1}.

Le **principe de construction de l'enveloppe inférieure** est analogue à celui de l'enveloppe supérieure.

Enveloppe convexe dans $\mathbb{R}^2$ (Prise 2)

Algorithme 2: EnveloppeConvexe2DIncremental

Entrées : Un ensemble $P = \{p_1, p_2, \ldots, p_n\}$ où $p_i \in \mathbb{R}^2$
Sorties : Une liste $\mathcal{L}$ contenant les sommets de l'enveloppe convexe $\mathcal{CH}(P)$ dans le sens horaire

1 Trier les points de P selon leur coordonnées en x pour obtenir la liste $p_1, \ldots, p_n$;
2 Ajouter les points p_1 et p_2 à la liste $\mathcal{L}_{\text{sup}}$ avec p_1 comme premier point;
3 **pour** $i \leftarrow 3$ *à* n **faire**
4 Ajouter p_i à $\mathcal{L}_{\text{sup}}$;
5 **tant que** $\mathcal{L}_{\text{sup}}$ *contient plus de deux points et que ses trois derniers points ne forment pas un virage à droite* **faire**
6 Retirer le point milieu des trois derniers de $\mathcal{L}_{\text{sup}}$;
7 Ajouter les points p_n et p_{n-1} à la liste $\mathcal{L}_{\text{inf}}$ avec p_n comme premier point;
8 **pour** $i \leftarrow n-2$ *à* 1 **faire**
9 Ajouter p_i à $\mathcal{L}_{\text{inf}}$;
10 **tant que** $\mathcal{L}_{\text{inf}}$ *contient plus de deux points et que ses trois derniers points ne forment pas un virage à droite* **faire**
11 Retirer le point milieu des trois derniers de $\mathcal{L}_{\text{inf}}$;
12 Retirer le premier et le dernier point de $\mathcal{L}_{\text{inf}}$ pour éviter la duplication des points de jonction entre l'enveloppe supérieure et l'enveloppe inférieure;
13 Concaténer $\mathcal{L}_{\text{sup}}$ et $\mathcal{L}_{\text{inf}}$ pour donner la liste $\mathcal{L}$

Problèmes potentiels

En regardant attentivement l'algorithme précédent, on constate encore **quelques petits problèmes**.

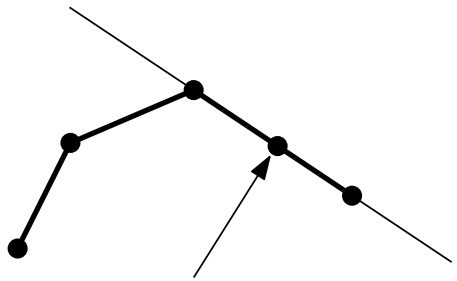
Par exemple, on a pris pour acquis que **deux points ne peuvent pas posséder la même coordonnée en** x. Si cette supposition n'est pas vérifiée, il est impossible de réaliser l'étape de tri à la ligne 1.

Pour régler ce problème, il suffit d'**utiliser l'ordre lexicographique pour le tri**, c'est-à-dire trier d'abord par rapport à x, puis en cas d'égalité, de trier par rapport à y.

Problèmes potentiels

Une autre situation qui a été ignorée est celle où **les trois points utilisés pour déterminer le sens d'un virage** (ligne 5) **sont colinéaires**.

Dans ce cas, le point milieu **ne doit pas être un sommet de l'enveloppe convexe**. Ce faisant, les points colinéaires doivent être traités **comme des virages à gauche**.



[1]

Problèmes potentiels

La solution est alors d'**utiliser un test qui retourne vrai si les trois points font un virage à droite** et d'ajuster le reste des opérations en conséquence.

Avec ces modifications, l'algorithme évalue correctement l'enveloppe convexe d'un ensemble de points, mais une question demeure : comment réagit-il en présence d'**erreurs d'arrondis dues à l'imprécision des calculs** ?

Problèmes potentiels

Lorsque de telles erreurs se produisent, **deux situations déplaisantes peuvent survenir** :

- Un point **qui devrait être dans la liste** des sommets de l'enveloppe convexe **est retiré** de celle-ci.
- Un point **qui se trouve à l'intérieur de l'enveloppe convexe demeure dans la liste** des sommets.

Il est aussi tout à fait possible que les erreurs d'arrondis n'aient aucune influence sur le résultat. Voyons malgré tout **les conséquences associées à ces deux cas désagréables**.

Problèmes potentiels

Dans tous les cas, l'algorithme retournera **une liste de points formant un polygone fermé** où tout ensemble de trois points consécutifs **forment approximativement un virage à droite**.

On dit **approximativement**, puisqu'un faible virage à gauche pourrait être interprété comme un virage à droite à cause de l'imprécision des calculs. Ceci créerait **une légère déformation dans l'enveloppe convexe**, ce qui, dans la plupart des applications, n'est pas catastrophique.

Problèmes potentiels

Finalement, **côté performances**, on a que l'algorithme incrémental s'exécute en temps $O(n \log n)$, ce qui est beaucoup plus intéressant que le $O(n^3)$ que proposait l'algorithme original.

La construction des enveloppes convexes supérieure et inférieure se fait en temps $O(n)$, mais c'est **le tri en ordre lexicographique** (ligne 1) qui augmente la complexité à $O(n \log n)$.

Enveloppe convexe 3D

1 Introduction

2 Notions de base en visualisation de volumes

3 Espace image et espace objet

4 Calcul d'enveloppe convexe
- Enveloppe convexe sur un plan
- Enveloppe convexe 3D

5 Références

Passage au 3D

À la section précédente, on a vu que l'enveloppe convexe d'un ensemble P contenant n points dans le plan est **un polygone convexe** dont les sommets sont des points de P. Ce faisant, l'enveloppe convexe **contenait au plus n sommets**.

En 3D, on peut faire une affirmation semblable : l'enveloppe convexe d'un ensemble P de n points 3D est **un polyèdre convexe** dont les sommets sont des points de P et qui **contient au plus n sommets**.

Passage au 3D

Dans le cas planaire, la borne supérieure sur le nombre de sommets avait comme conséquence que **la complexité géométrique de l'enveloppe convexe était linéaire**, puisque le nombre d'arêtes d'un polygone est égale au nombre de sommets de celui-ci.

Une fois passé en 3D, cette affirmation ne tient plus : le nombre d'arêtes d'un polyèdre peut être **supérieur à son nombre de sommets**.

Passage au 3D

Fort heureusement, **cette différence ne peut être trop grande**. On dispose même d'un théorème donnant une borne supérieure pour le nombre d'arêtes et le nombre de faces d'un polyèdre.

Théorème

Soit $\mathcal{P}$ un polyèdre convexe ayant n sommets. Le **nombre d'arêtes** de $\mathcal{P}$ est alors au plus $3n - 6$ et le **nombre de faces** de $\mathcal{P}$ est au plus $2n - 4$.

Passage au 3D

En combinant le théorème précédent avec le fait que l'enveloppe convexe d'un ensemble P de points 3D est un polyèdre convexe dont les sommets sont des éléments de P, on obtient que **la complexité géométrique de l'enveloppe convexe** d'un ensemble contenant n points 3D **est linéaire**.

Attention ! On parle bien ici de la complexité de la structure géométrique (nombre d'arêtes et de sommets), et non du temps d'exécution de l'algorithme de calcul de l'enveloppe convexe.

Calcul de l'enveloppe convexe 3D (Préambule)

Soit P un ensemble de n points $3D$. On calculera l'enveloppe convexe $\mathcal{CH}(P)$ en utilisant un **algorithme incrémental aléatoire**.

La première étape de la construction incrémentale consiste à **choisir quatre points de P qui ne soient pas coplanaires**, de telle sorte que leur enveloppe convexe soit un tétraèdre.

Calcul de l'enveloppe convexe 3D (Préambule)

Pour obtenir ceux-ci, on conserve p_1 et p_2, les deux premiers points de P, puis on parcourt l'ensemble à la recherche d'un point p_3 qui ne soit **pas colinéaire avec p_1 et p_2**.

On continue notre chemin pour ensuite trouver un point p_4 qui ne soit **pas coplanaire avec p_1, p_2 et p_3**.

S'il est impossible de trouver quatre points non coplanaires, alors **les données de P reposent sur un même plan** et on peut simplement utiliser la technique de la section précédente pour évaluer l'enveloppe convexe.

Calcul de l'enveloppe convexe 3D (Préambule)

On évalue ensuite une **permutation aléatoire des points restants** : $p_5, \ldots, p_n$. L'algorithme s'attardera aux points un par un dans cet ordre aléatoire en maintenant à jour l'enveloppe convexe à chaque étape.

Pour un entier $r \geq 1$, on définit l'ensemble $P_r = \{p_1, \ldots, p_r\}$. À une étape donnée de l'algorithme, **on aura à ajouter le point p_r à l'enveloppe convexe de P_{r-1}**.

On devra donc transformer $\mathcal{CH}(P_{r-1})$ en $\mathcal{CH}(P_r)$.

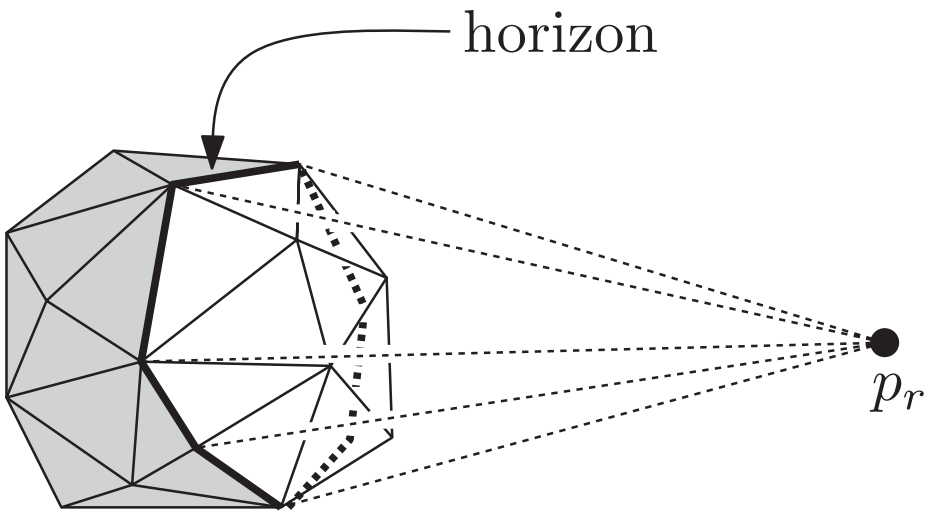
Calcul de l'enveloppe convexe 3D (Préambule)

On aura alors **deux situations possibles** :

- Si p_r **se trouve à l'intérieur ou sur la frontière de** $\mathcal{CH}(P_{r-1})$, alors on a $\mathcal{CH}(P_{r-1}) = \mathcal{CH}(P_r)$ et aucun calcul n'est nécessaire.
- Si p_r **se trouve à l'extérieur de** $\mathcal{CH}(P_{r-1})$, alors... la vie ne sera pas si simple !

Imaginons que l'on se trouve à la position p_r et que l'on regarde $\mathcal{CH}(P_{r-1})$. **On pourra alors voir certaines faces de l'enveloppe convexe**, tandis que d'autres seront invisibles.

Calcul de l'enveloppe convexe 3D (Préambule)

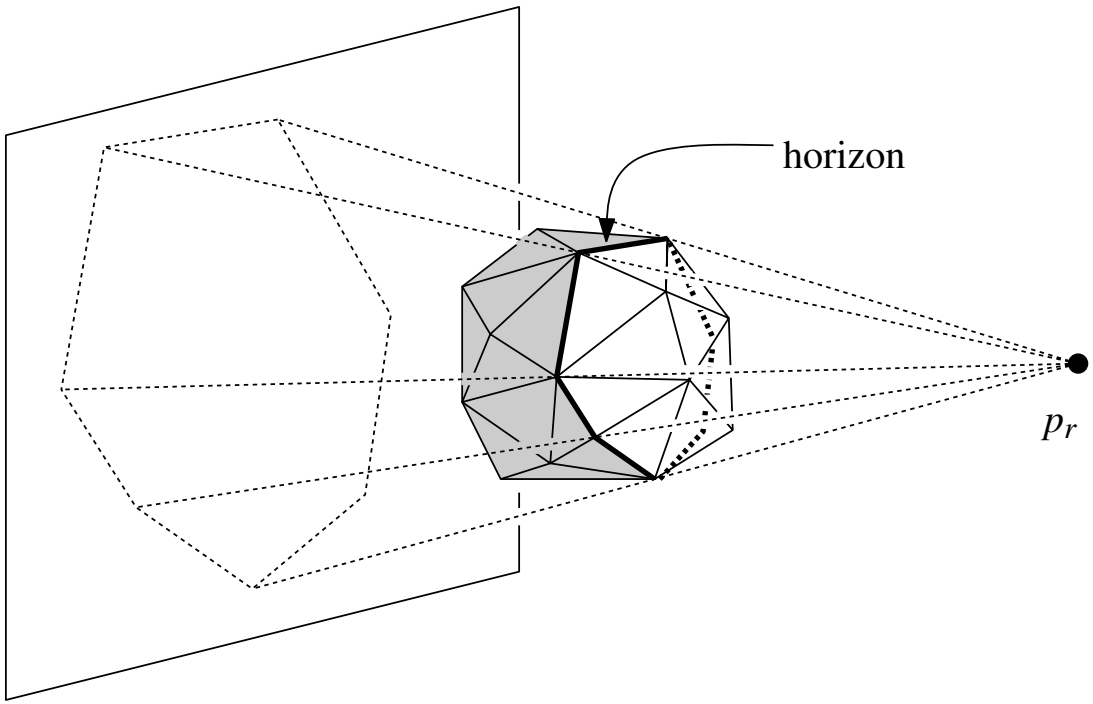


[1]

Les faces visibles forment la **région visible** de p_r sur $\mathcal{CH}(P_{r-1})$. Cette région est bornée par un ensemble fermé d'arêtes de $\mathcal{CH}(P_{r-1})$. On appelle **horizon** de p_r ce groupe d'arêtes.

Calcul de l'enveloppe convexe 3D (Préambule)

La projection de l'horizon donne la frontière du polygone convexe obtenu en projetant $\mathcal{CH}(P_{r-1})$ sur un plan en utilisant p_r comme centre de projection.



[1]

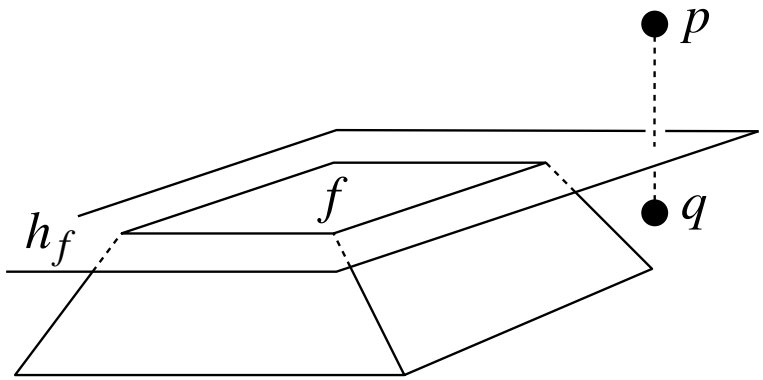
Calcul de l'enveloppe convexe 3D (Préambule)

Qu'entend-on géométriquement lorsqu'on dit qu'une face de $\mathcal{CH}(P_{r-1})$ est **visible à partir d'un point** p_r ? Pour comprendre, considérons h_f, le plan de support de la face f de $\mathcal{CH}(P_{r-1})$.

En raison de sa convexité, on a que $\mathcal{CH}(P_{r-1})$ **se trouve entièrement dans un des deux demi-espaces** définis par h_f.

Calcul de l'enveloppe convexe 3D (Préambule)

On aura alors que la face f sera visible à partir d'un point **si celui-ci se trouve dans le demi espace ne contenant pas** $\mathcal{CH}(P_{r-1})$.



f is visible from p, but not from q

[1]

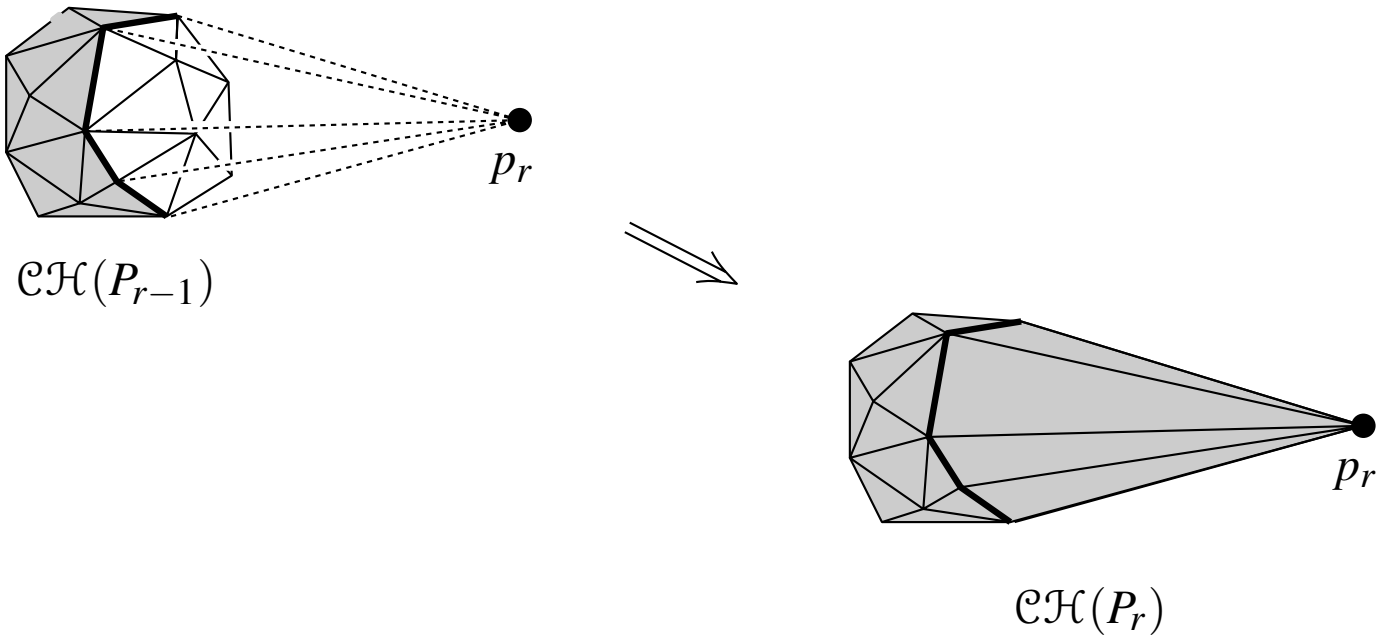
Calcul de l'enveloppe convexe 3D (Préambule)

L'horizon de p_r joue un rôle crucial quand vient le temps de **transformer** $\mathcal{CH}(P_{r-1})$ **en** $\mathcal{CH}(P_r)$, puisqu'il sépare les faces de l'enveloppe convexe en deux catégories :

- Celles qui seront conservées (**les faces invisibles**) ; et
- Celles qui seront remplacées (**les faces visibles**).

Calcul de l'enveloppe convexe 3D (Préambule)

Ces dernières seront remplacées **par des faces reliant p_r à son horizon**.



[1]

DCEL

Avant de poursuivre avec le processus de mise à jour de $\mathcal{CH}(P_{r-1})$ suite à l'ajout du point p_r, on doit **décider d'une structure de données permettant de stocker l'enveloppe convexe**.

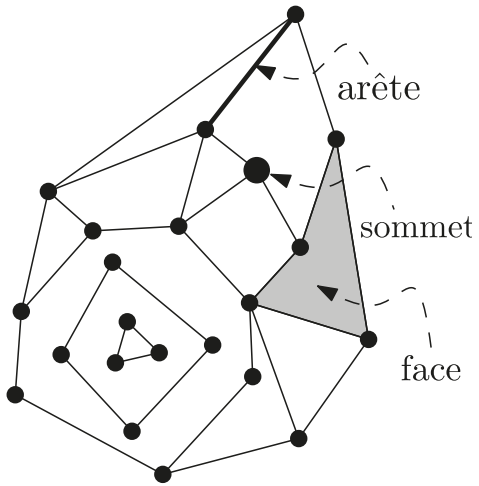
Contrairement au cas 2D, représenter l'enveloppe convexe comme une simple liste d'arêtes n'est pas une bonne idée.

On utilisera plutôt la **liste d'arêtes doublement chaînée** (DCEL), qui, à la base, est une structure élaborée pour **stocker une subdivision du plan**.

DCEL

Ainsi, une DCEL sert à représenter en mémoire **une subdivision planaire**, mais son fonctionnement se prête très bien au stockage d'une enveloppe convexe 3D.

Une DCEL contient un enregistrement pour chaque **face**, chaque **arête** et chaque **sommet** de l'enveloppe convexe.

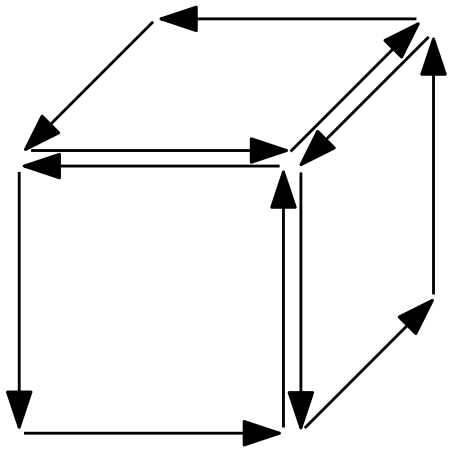


[1]

DCEL

La frontière d'un polyèdre convexe peut être **interprétée comme une subdivision planaire**. Ce faisant, la DCEL est une structure appropriée pour stocker une enveloppe convexe 3D en mémoire.

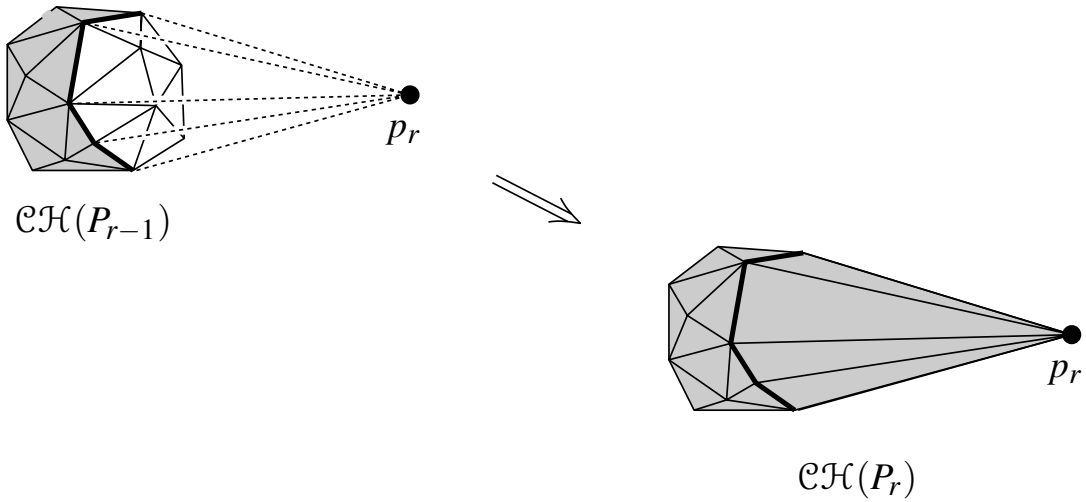
On construira la liste de manière à ce que les arêtes soient orientées dans le sens **antihoraire lorsqu'on les observe depuis l'extérieur du polyèdre**.



[1]

Calcul de l'enveloppe convexe 3D

Revenons donc à nos moutons... **on souhaite ajouter un point p_r à l'enveloppe convexe** $\mathcal{CH}(P_{r-1})$.



[1]

On dispose donc d'**une DCEL représentant** $\mathcal{CH}(P_{r-1})$ et on souhaite la modifier pour obtenir la DCEL de $\mathcal{CH}(P_r)$.

Calcul de l'enveloppe convexe 3D

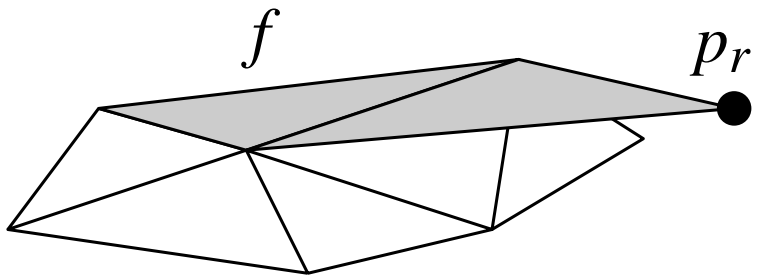
Si l'on connaissait **toutes les faces de** $\mathcal{CH}(P_{r-1})$ **qui sont visibles** depuis le point p_r, il serait facile de

1. **Retirer** ces faces de la DCEL ;
2. **Évaluer** les faces reliant p_r à l'horizon ;
3. **Ajouter** ces nouvelles faces à la DCEL

L'ensemble de ces opérations peut se faire en temps $O(n)$.

Calcul de l'enveloppe convexe 3D

Il resterait seulement à s'assurer que l'ajout des nouvelles faces **n'a pas créé de surface coplanaire dans** $\mathcal{CH}(P_r)$.



[1]

Cette situation survient **si** p_r **se trouve sur le plan de support** d'une face f dans $\mathcal{CH}(P_{r-1})$. Or, selon notre définition de la visibilité, une telle face n'est pas visible depuis la position p_r.

Calcul de l'enveloppe convexe 3D

La conséquence sera que, comme f n'est pas visible, elle restera inchangée dans la DCEL, mais **des arêtes reliant p_r aux sommets de** f faisant partie de l'horizon seront ajoutés.

Comme cette nouvelle face est coplanaire avec f, on doit les combiner afin de **former une seule face**.

Calcul de l'enveloppe convexe 3D

Pour arriver ici, on a fait la supposition que l'**on connaissait toutes les faces de** $\mathcal{CH}(P_{r-1})$ **qui sont visibles** depuis le point p_r, mais comment obtient-on celles-ci ?

Bien entendu, on pourait, pour chaque point ajouté, **tester l'ensemble des faces** pour savoir de quel côté se point se situe. Cela mènerait toutefois à un temps d'exécution $O(n^2)$... tâchons de faire mieux.

Calcul de l'enveloppe convexe 3D

L'idée sera ici de **prendre les devants sur le problème** : en plus d'une DCEL représentant l'enveloppe convexe, on conservera en mémoire de l'information supplémentaire :

- **Pour chaque face** $f \in \mathcal{CH}(P_{r-1})$, on stocke un ensemble

$$P_{\text{conflit}}(f) \subseteq \{p_{r+1}, p_{r+2}, \ldots, p_n\}$$

  contenant **tous les points qui peuvent voir** f.

- Inversement, **pour chaque point** p_t avec $t > r$, on stocke un ensemble

$$F_{\text{conflit}}(p_t) \subseteq \mathcal{CH}(P_r)$$

  contenant **toutes les faces visibles depuis** p_t.

Calcul de l'enveloppe convexe 3D

On dira qu'un point $p \in P_{\text{conflit}}(f)$ est **en conflit avec la face** f puisque p et f ne peuvent pas être ensemble dans l'enveloppe convexe. Lorsqu'on ajoute un point $p \in P_{\text{conflit}}(f)$ à l'enveloppe convexe, on doit en supprimer la face f.

On dira que $P_{\text{conflit}}(f)$ et $F_{\text{conflit}}(p_t)$ sont des **listes de conflits**.

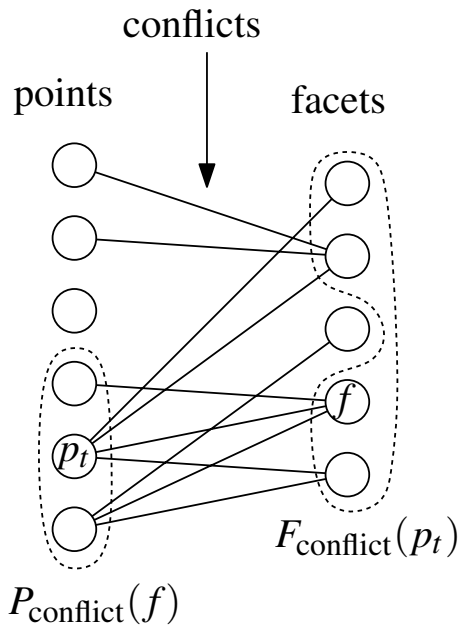
Calcul de l'enveloppe convexe 3D

Les listes de conflits sont maintenues à jour à l'aide d'un **graphe de conflits** noté $\mathcal{G}$.

Celui-ci possède **un sommet pour chaque point** de P n'ayant pas été inséré dans l'enveloppe convexe de même qu'**un sommet pour chaque face** de l'enveloppe convexe courante.

$\mathcal{G}$ possède aussi **une arête pour chaque conflit** entre un point et une face.

Calcul de l'enveloppe convexe 3D



[1]

En d'autres mots, **il y aura une arête entre un point $p_t \in P$ et une face $f \in \mathcal{CH}(P_r)$** si $r < t$ et f est visible depuis p_t.

Calcul de l'enveloppe convexe 3D

À l'aide du graphe $\mathcal{G}$, on peut retrouver **l'ensemble $F_{\text{conflit}}(p_t)$ pour un point p_t quelconque** et **l'ensemble $P_{\text{conflit}}(f)$ pour une face f quelconque**, et ce en temps $O(n)$.

Ainsi, lorsqu'on ajoute p_r à $\mathcal{CH}(P_{r-1})$, il suffit de récupérer $F_{\text{conflit}}(p_r)$ à partir de de $\mathcal{G}$ et de **remplacer les anciennes faces par les nouvelles qui connectent p_r à l'horizon**.

L'initialisation de $\mathcal{G}$ pour $\mathcal{CH}(P_4)$ peut être faite en temps $O(n)$. Il suffit de parcourir P et de déterminer quelles faces de $\mathcal{CH}(P_4)$ les points peuvent voir.

Calcul de l'enveloppe convexe 3D

La mise à jour du graphe $\mathcal{G}$ suite à l'ajout du point p_r se fait en trois étapes :

1. **On élimine du graphe** les sommets et les arêtes incidents associés aux faces de $\mathcal{CH}(P_{r-1})$ qui disparaissent de l'enveloppe convexe.
2. **On élimine du graphe** le sommet associé au point p_r.
3. **On ajoute des sommets au graphe** pour chaque nouvelle face qui relie p_r à l'horizon.

Il reste alors à **trouver les listes de conflits** associées à ces nouvelles faces.

Calcul de l'enveloppe convexe 3D

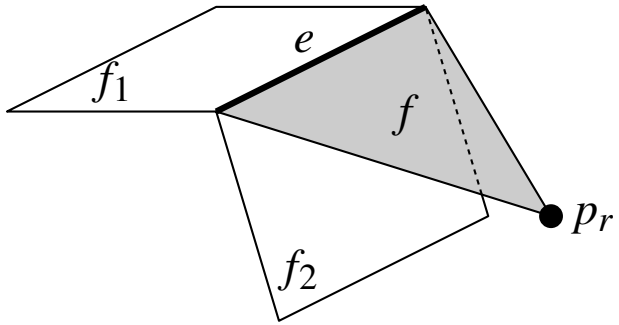
Les faces crées suite à l'insertion de p_r **sont toutes des triangles**, à l'exception de celles reposant sur le plan de support d'une face de $\mathcal{CH}(P_{r-1})$ et qui ont été combinées avec celles-ci.

La liste de conflits associée à une telle face **demeure la même que celle de la face de** $\mathcal{CH}(P_{r-1})$ avec laquelle elle a été combinée.

Trouver la liste de conflits associée à une nouvelle face triangulaire sera **moins trivial**...

Calcul de l'enveloppe convexe 3D

Soit f **une nouvelle face incidente à** p_r ajoutée dans $\mathcal{CH}(P_r)$. **Si** f **est visible depuis un point** p_t, alors p_t voit nécessairement l'arête **e** de f qui est opposée à p_r.



[1]

Cette arête **e** faisait partie de l'horizon du point p_r et **était ainsi nécessairement présente dans** $\mathcal{CH}(P_{r-1})$.

Calcul de l'enveloppe convexe 3D

Or, comme $\mathcal{CH}(P_{r-1}) \subseteq \mathcal{CH}(P_r)$, l'arête **e** devait aussi être visible depuis p_t dans $\mathcal{CH}(P_{r-1})$. Pour que cela se produise, **au moins une des deux faces incidentes à e dans** $\mathcal{CH}(P_{r-1})$ **devait être visible depuis** p_t.

Cela implique que l'on peut trouver la liste de conflits associée à la face f en testant **seulement les points en conflit avec les deux faces incidentes à l'arête e** dans $\mathcal{CH}(P_{r-1})$.

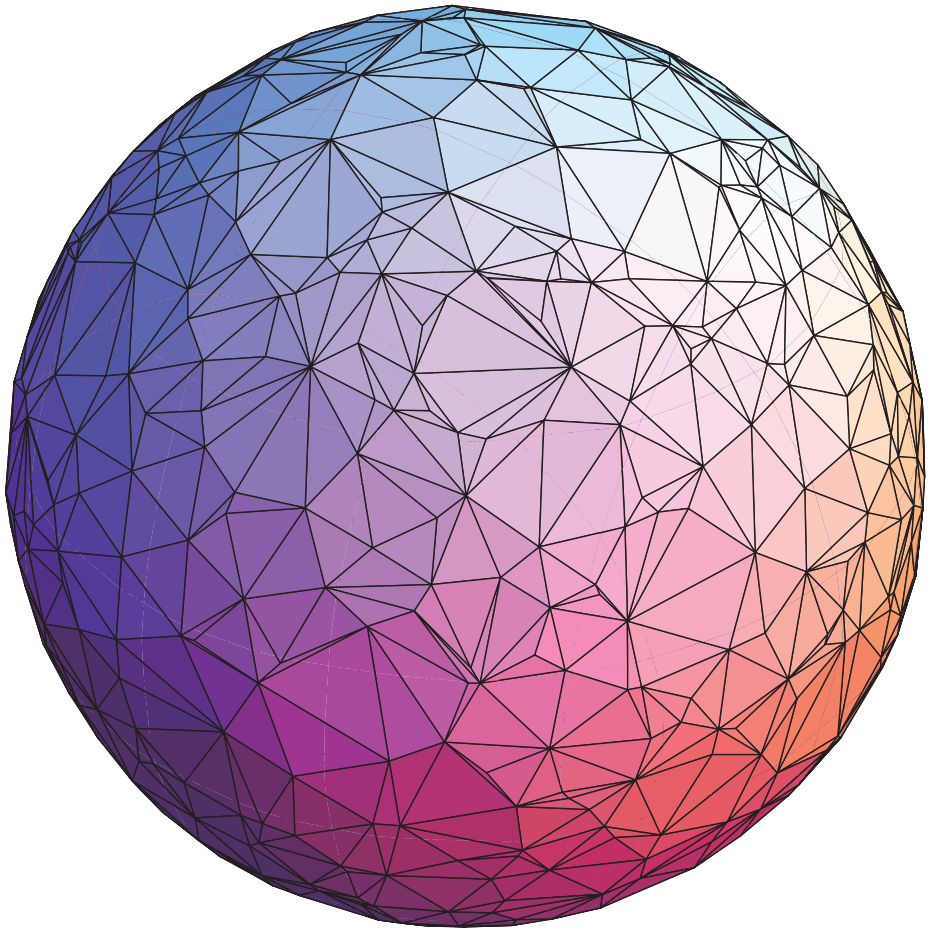
Calcul de l'enveloppe convexe 3D

Algorithme 3: EnveloppeConvexe3D

Entrées : Un ensemble $P = \{p_1, p_2, \ldots, p_n\}$ où $p_i \in \mathbb{R}^3$
Sorties : L'enveloppe convexe $\mathcal{CH}(P)$ de P

Trouver quatre points p_1, p_2, p_3, p_4 dans P qui ne soient pas coplanaires;
$\mathcal{C} \leftarrow \mathcal{CH}(p_1, p_2, p_3, p_4)$;
Obtenir une permutation aléatoire des points $p_5, p_6, \ldots, p_n$;
Initialiser le graphe de conflits $\mathcal{G}$ avec toutes les paires (p_t, f) visibles, où f est une face de $\mathcal{C}$ et $t > 4$.;
pour $r = 5$ *à* n **faire**
    si $F_{conflit}(p_r)$ *n'est pas vide (c-à-d si* $p_r \notin \mathcal{C}$*)* **alors**
        Ajouter p_r à $\mathcal{C}$;
        Retirer les faces dans $F_{\text{conflit}}(p_r)$ de $\mathcal{C}$;
        Créer une liste ordonnée $\mathcal{L}$ des arêtes formant l'horizon de p_r;
        pour chaque *arête* **e** $\in \mathcal{L}$ **faire**
            Relier **e** à p_r en créant une face triangulaire f et l'ajouter à $\mathcal{C}$;
            si f *est coplanaire avec la face voisine* f' *le long de* **e alors**
                Combiner f et f' en une seule face dont la liste de conflits est la même que pour f';
            sinon
                Créer un sommet pour f dans $\mathcal{G}$;
                $P(\mathbf{e}) \leftarrow P_{\text{conflit}}(f_1) \cup P_{\text{conflit}}(f_2)$ où f_1 et f_2 sont les deux faces incidente à **e** dans $\mathcal{CH}(P_{r-1})$;
                pour chaque *point* $p \in P(\mathbf{e})$ **faire**
                    Si f est visible depuis p, ajouter la paire (p, f) dans $\mathcal{G}$;
        Supprimer le sommet associé à p_r de $\mathcal{G}$;
        Supprimer les sommets correspondants aux faces dans $F_{\text{conflit}}(p_r)$ de $\mathcal{G}$ ainsi que leurs arêtes incidentes;
retourner $\mathcal{C}$;

Calcul de l'enveloppe convexe 3D



[2]

Calcul de l'enveloppe convexe 3D

Finalement, on s'attarde à **la complexité de l'algorithme** précédent. On acceptera ces résultats sans démonstration.

Soit P un ensemble de n points dans $\mathbb{R}^3$. Le **nombre total de faces générées par l'algorithme** est au plus $6n - 20$ et le calcul de l'enveloppe convexe associée à P s'effectue **en un temps attendu de** $O(n \log n)$, selon le résultat de la permutation aléatoire.

Références

1 Introduction

2 Notions de base en visualisation de volumes

3 Espace image et espace objet

4 Calcul d'enveloppe convexe

5 Références

Références I


M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars.
Computational geometry, 2010.


S. L. Devadoss and J. O'Rourke.
Discrete and computational geometry, 2011.


C. Taras, T. Ertl, R. Botchen, and I. Entina.
Online course for scientific visualization, 2007.


A. C. Telea.
Data visualization : Principles and practice, 2008.