Mini-projets « Maillages et applications »

Les sujets sont à faire seul, en binôme ou exceptionnellement en trinôme (sujets étendus). Chaque sujet peut-être sélectionné au plus une fois (sur le mode premier demandé, premier servi). En cas de questions contactez Florent (florent.lafarge@inria.fr) pour les sujets 1 à 4, et Pierre pour le reste (pierre.alliez@inria.fr).

Le travail à produire est le suivant:

- Lire les articles en référence, lorsque listés.
- Mettre en œuvre l'algorithme et une démonstration soit en C++ en partant des TPs, soit en utilisant votre langage ou outil favori (Matlab, Scilab).
- Expérimenter
- Rédiger un rapport (max ~10 pages, au format pdf) qui détaille votre approche et vos expérimentations.
- Préparer une présentation (15 minutes en binôme, 10 tout seul ou 20 en trinôme) pour une soutenance prévue le lundi 23 avril 2018 (horaires et ordres de passage à venir).

Sujet 1 : Détection de cheminées et de chiens-assis. En partant des TPs 3 et 4, proposer un algorithme permettant de détecter les superstructures contenues sur les toits, à savoir les cheminées et les chiens-assis. Chaque superstructure détectée devra être localisée par un point (centre de masse ou centroid) et devra être identifiée comme étant soit (i) une cheminée, (ii) un chien-assis, ou (iii) autre. En plus des données du TP4, vous pourrez utiliser les données suivantes :
http://www-sop.inria.fr/members/Florent.Lafarge/other/sujets_inputs_mesh_additional.rar

Sujet 2 : Segmentation de maillage avec végétation. Sur la base du TP4, proposer un algorithme permettant de segmenter des maillages en 4 classes : toit, façade, sol et végétation. Utiliser les données d'entrée suivantes pour tester votre algorithme:
http://www-sop.inria.fr/members/Florent.Lafarge/other/sujets_inputs_mesh_additional.rar

Sujet 3 : Localisation des fenêtres. En partant des TPs 3 et 4, proposer un algorithme permettant de localiser les fenêtres dans un maillage urbain dense par des points (centroids des fenêtres) En plus des données du TP4, vous pourrez utiliser les données suivantes :
http://www-sop.inria.fr/members/Florent.Lafarge/other/sujets_inputs_mesh_additional.rar

Sujet 4 : **Régularisation de primitives planaires.** A partir d'une configuration de plans extraite par le code du TP3, proposer un algorithme permettant de détecter les paires de plans 1/ parallèles, 2/ coplanaires et 3/ orthogonaux. Dans un second temps proposer une méthode pour régulariser les plans entre eux en fonction des régularités détectées. Indications : la détection des régularités devra faire intervenir un paramètre d'angle et un paramètre de position.

Sujet 5 : convergence de l'itération de Lloyd. Le code « Lloyd » fournit déjà une mise en œuvre de l'itération de Lloyd, qui alterne diagramme de Voronoi et relocalisation des générateurs aux centres de masse des cellules. Un domaine au format .edg peut être ouvert par l'application. Ce format ASCII contient simplement le nombre de segments puis les

coordonnées des segments sur chaque ligne. Vous pouvez éditer vos propres fichiers mais attention : ces segments doivent former des cycles de lignes brisées fermées.

Dans ce code le domaine est représenté sous la forme d'une triangulation contrainte (cdt.h) où les faces triangulaires sont augmentées d'un booléen inside/outside. Le centre de masse de chaque cellule de Voronoi est calculé sur un polygone formé par une union de triangles, qui calculés en intersectant la cellule de Voronoi avec les triangles du domaine.

Le but est d'étudier la convergence de l'itération de Lloyd en traçant la courbe de la moyenne de la norme des vecteurs de déplacement des sites, en fonction des itérations (cette valeur est déjà calculée par la fonction lloyd dans le fichier dt.h). Il faudra tracer aussi la courbe pour l'énergie minimisée en fonction des itérations. Note : le calcul de l'énergie énergie requiert soit de mettre en œuvre une quadrature (un échantillonnage uniforme de points dans les cellules de Voronoi), soit de trouver une forme close à partir de la décomposition des cellules intersectées en triangles, et d'une transformée affine du triangle canonique pour lequel une forme close est triviale. Essayez les deux approches.

1. Parcourir le code pour comprendre sa structure. La scène stocke le domaine et la triangulation optimisée par Lloyd. Vous avez un domaine carré par défaut et pouvez ajouter des générateurs soit à la souris, soit avec le menu data/add random vertices. Le diagramme intersecté s'affiche à partir de 3 générateurs en position générale. Le raccourci « L » permet de lancer l'itération de Lloyd.
2. Ajouter une option pour visualiser les vecteurs de relocalisation.
3. Tracer la courbe de convergence pour différentes initialisations (points uniformes, ou concentrés dans un coin du domaine, sur une ligne, etc). Pour tracer la courbe vous pouvez soit générer un simple fichier ASCII avec les valeurs puis l'ouvrir sous Excel, soit utiliser gnuplot (http://www.gnuplot.info/) avec le bon format, soit directement en OpenGL dans la fenêtre de visualisation. Lancer également un ajustement de courbe pour vérifier si le taux de convergence est linéaire ou quadratique.
4. Tracer l'évolution des distributions des énergies par cellule, en fonction des itérations. Commenter.
5. Quelle est l'influence du domaine sur la convergence ? Mener des expériences avec un domaine en forme de « sablier », avec un goulet d'étranglement, où les sites initiaux sont générés uniformément et aléatoirement sur la partie haute du sablier. Varier la largeur du goulet et observer.
6. Quelles idées d'*initialisation* proposez-vous pour accélérer la convergence ?
7. Ajouter une option pour déplacer les générateurs non pas aux centres de masse des cellules, mais un peu au-delà (avec un paramètre « overshooting »). Observer le gain en termes de convergence en fonction de ce paramètre. Quel problème peut survenir pour un paramètre trop important ? Comment le prévenir ? Utilisez l'option de visualisation des vecteurs de relocalisation, et notez que le domaine cdt.h comprend une fonction inside().
8. Ajouter une option d'inertie temporelle qui garde en mémoire les centres de masse des itérations précédentes, et déplacer les générateurs avec une combinaison linéaire des N vecteurs déplacements précédents. Observer le gain en termes de convergence en fonction des coefficients de ladite combinaison linéaire.

Sujet 6 : itération de Lloyd et densité variable. Le code « lloyd » utilise une fonction de densité uniforme. Mettre en œuvre une variante utilisant une fonction de densité variable qui impacte le calcul du centre de masse et requiert une quadrature.

1. Mettre en œuvre la quadrature à base de « quadrature points » (chaque point associé à une masse) générés uniformément. Itérer sur ces points pour les localiser (cf fonction nearest_vertex de la triangulation de Delaunay) dans les cellules de Voronoi et calculer le centre de masse des cellules.
2. Mettre en œuvre la quadrature à base de triangles en utilisant le paradigme « recursive longest edge bisection » pour chaque triangle. Ceci implique de décomposer chaque cellule de Voronoi intersectée avec le domaine en triangles, ce qui est déjà fourni par le code. Contactez-moi pour des conseils sur cette question si besoin.
3. Expérimenter en définissant une fonction de densité linéaire (par exemple en fonction de la coordonnée x), puis radiale (fonction de la distance à un centre), puis sinusoïdale, etc. Calculer la distribution de l'énergie minimisée par l'itération, pour chaque cellule de Voronoi (requiert une quadrature via un échantillonnage uniforme de points dans les cellules de Voronoi), et observer l'équi-distribution de l'énergie minimisée par Lloyd au cours des itérations.
4. Expérimenter en calculant la distribution des degrés des cellules de Voronoi (hexagones, pentagones, etc.) en fonction de la vitesse de gradation de la fonction de densité, pour un grand nombre de sites et une quadrature précise.

Sujet 7 : Diagramme de Voronoi et « stippling ». A partir des codes « lloyd » et « interpolation » et des images « stippling ». Vous pourrez aussi utiliser le code « interpolation » qui lit une image en entrée et écrit une image en sortie.

Le rendu demi-ton ou « halftoning », en reprographie, est une technique utilisée par les imprimantes qui permet de rendre plusieurs niveaux de gris d'une couleur à partir d'une impression monochrome, en général une encre noire. Les imprimantes ne peuvent imprimer que des points d'encre. Elles les arrangent de telle sorte que l'œil humain ne discerne plus ces points mais les intègre pour donner une illusion de plusieurs niveaux de gris. Il existe une multitude d'approches pour obtenir un rendu demi-ton. L'une d'elles appelée « stippling » consiste à placer des points (disques) dont la densité reproduit au mieux le niveau de gris d'une image en entrée. Lorsqu'il est effectué manuellement par un artiste, ce procédé peut prendre des jours pour une seule image, les degrés de liberté étant le diamètre de la pointe et le niveau de gris du stylo.

Dans ce projet vous concevrez un algorithme où les points seront placés comme générateurs d'un diagramme de Voronoi. Les positions des points seront optimisées à l'aide d'une méthode itérative (« farthest point optimization »), configurée avec une densité variable donnée par le niveau de gris de l'image en entrée.

Echauffements :

1. Vous lirez une image en entrée et utiliserez le niveau de gris comme densité. Vous devez générer en sortie une image de même résolution, dont certains pixels sont colorés en noir.
2. Supposons que les pixels noirs correspondent aux générateurs de Voronoi, associés à la masse totale 1. Concevoir une méthode pour calculer la *densité* (masse par unité d'aire) d'une cellule de Voronoi (normalisée, par exemple 1 pour une cellule toute noire et 0 pour tout blanc). Ajouter une option pour
3. Concevoir une méthode pour calculer la densité moyenne des pixels de l'image localisés dans une cellule de Voronoi donnée. Le but sera ensuite de rapprocher cette valeur avec la densité de l'image en demi-ton.
4. Etant donnée une image en entrée, mettre au point une méthode pour évaluer le nombre total N de pixels noirs requis. On peut voir N comme la quantité d'encre noire à répartir sur la feuille de papier lors de l'impression.
5. Mettre au point une méthode en complexité linéaire (en le nombre de pixels) pour générer N pixels noir en accord approximatif avec la densité de l'image. Une piste possible est d'utiliser un parcours ligne par ligne ou serpentin et d'accumuler la densité au fil du parcours pour finaliser un pixel noir à chaque fois que la densité accumulée est supérieure à 1. Attention à ne pas oublier de reporter l'erreur de quantification. Testez sur des images de type rampe linéaire et commenter vos expériences.

Algorithmes :

1. A partir de cette initialisation de N points, optimiser le placement à budget de points constant via la méthode « *farthest point optimization », (cf Chen & Gotsman)* qui consiste à parcourir *tous les générateurs en séquence*, et à les relocaliser (à l'intérieur de l'image) itérativement au centre du plus grand cercle vide (utiliser la dualité Delaunay-Voronoi et la propriété du cercle vide des triangles de Delaunay). Attention : parcourir tous les générateurs signifie que vous devrez d'abord itérer sur tous les sommets pour stocker les vertex handle dans une liste, puis opérer des relocalisations dans la triangulation avec les opérateurs remove puis insert. Tester avec une fonction de densité *uniforme* (donc pas besoin de lire une image). Qu'observez-vous après plusieurs passes ? Comment pouvez-vous caractériser la qualité du rendu demi-ton ? Expérimenter avec une initialisation triviale qui entasse N pixels noirs tout à gauche de l'image, ou les répartit aléatoirement sans lien avec la densité.
2. Reprendre la question précédente pour une image avec une densité variable. Attention : la notion de rayon du cercle vide doit être normalisée pour prendre en compte les différences entre la densité de l'image et celle de l'image demi-ton, pour déplacer les générateurs où cela réduit le plus cette différence.
3. Modifier l'algorithme comme suit: au lieu de relocaliser tous les générateurs en séquence, concevoir un algorithme qui relocalise à chaque itération le générateur le « plus dense » vers la zone la moins dense au sens de la différence mentionnée au-dessus. Cet algorithme peut-il boucler (cycle infini) ? Quel critère peut-on imaginer pour forcer la convergence de l'algorithme ?

4. Après convergence il reste une erreur résiduelle de quantification, de telle sorte que la différence de densité est en générale non nulle. Modifier la couleur du pixel de noir à un niveau de gris pour réduire cette erreur.

Avancé (optionnel) :

1. Supposons maintenant que les pixels noirs ne sont pas des points mais des *carrés* de côté unitaire, centrés sur les pixels qui contiennent les points placés dans l'image pour le stippling. Concevoir une méthode et le code pour calculer la *densité* d'une cellule de Voronoi (normalisée, par exemple 1 pour noir et 0 pour blanc), en considérant les bords de l'image comme la limite du domaine utilisé pour le calcul de la densité. Attention : on suppose que chaque cellule contient un seul générateur qui correspond à un carré noir de côté unitaire et centré sur ledit générateur. Comme il faut intersecter la cellule de Voronoi avec des carrés (pixels de l'image) vous pouvez réutiliser la fonction cvt_intersect() dans le fichier dt.h en lui donnant deux triangles qui partitionnent ledit carré.
2. Supposons que chaque générateur corresponde à des disques noirs de rayon K pixels, où K est un paramètre fourni par l'utilisateur. Expérimenter et commenter. Comme un disque pixellisé est une approximation au bord du disque, vous pourrez utiliser des pixels en niveaux de gris pour réduire ce problème (comment calculer alors le niveau de gris ?).

Références**:**

https://fr.wikipedia.org/wiki/Halftoning

Chen and Gotsman. *Parallel blue-noise sampling by constrained farthest point optimization*. Computer Graphics Forum. (Proceedings of Symposium on Geometry Processing), 2012. https://people.mpi-inf.mpg.de/~chen/

Deussen, Hiller, van Overveld, Strothotte: Floating Points: A Method for Computing Stipple Drawings. Computer Graphics Forum, 2000.

http://cg.postech.ac.kr/research/stippling/stippling_paper.pdf

http://www.geometry.caltech.edu/pubs/dGBOD12.pdf

Sujet 8 : Interpolation de Shepard (distance inverse). L'interpolation dite de Shepard (par pondération inverse à la distance) est une méthode d'interpolation spatiale permettant d'assigner une valeur à un espace non connu à partir d'un ensemble de points connus. Dans ce projet vous interpolerez les couleurs d'une image à partir d'un ensemble de points.

Le sujet est décrit dans le fichier shepard.docx sous code/interpolation

Reference**:**
https://fr.wikipedia.org/wiki/Pond%C3%A9ration_inverse_%C3%A0_la_distance

Sujet 9 : Interpolation de Sibson (voisins naturels). L'interpolation par voisins naturels (dite aussi de Sibson) est une méthode d'interpolation spatiale permettant d'assigner une valeur à un espace non connu à partir d'un ensemble de points connus. L'interpolation par voisins naturels utilise les diagrammes de Voronoi, cf composant CGAL mentionné ci-dessous.

1. Expliquer le principe de cette méthode en dimension 2, et donner ses propriétés.
2. Mettre en œuvre une version 2D avec CGAL (TP Delaunay 2D) qui interpole soit le niveau de gris d'une image (par exemple l'image terrain dans data/interpolation) à partir de points 2D, soit les couleurs des images. Comme pour le projet « Shepard » au-dessus vous devez générer une image en sortie.
3. Démarrer les expériences avec uniquement 4 points aux coins de l'image.
4. Générer des points d'interpolation aléatoirement sur l'image et observer.
5. Ajouter le code pour générer une image d'erreur d'interpolation.
6. Supposons qu'on se donne un paramètre de tolérance d'erreur maximale : concevoir une méthode itérative (qui ajoute un point après l'autre) pour construire une interpolation avec une erreur inférieure à cette tolérance.
7. Concevoir une méthode hiérarchique pour construire une interpolation avec une erreur inférieure à cette tolérance.
8. Avancé (optionnel) : Concevoir une méthode qui optimiserait les positions des points d'interpolation pour minimiser l'erreur totale d'interpolation.

Articles à lire :

[1] R. Sibson. A brief Description of Natural Neighbour Interpolation, Interpreting Multivariate Data, Woley, 1981.

[2] G. Farin. Surfaces over Dirichlet Tesselations, CAGD, pages 281-292, 1990.

Codes et autres indications :

Voir CGAL 2D and Surface Function Interpolation.

http://www.cgal.org/Manual/3.4/doc_html/cgal_manual/packages.html#part_XIV

Sujet 10 : Diagramme de Voronoi borné. A partir du TP « maillage 2D » qui utilise une triangulation de Delaunay contrainte, écrire un programme qui calcule et affiche le diagramme de Voronoi borné.

Le diagramme de Voronoi borné est le pseudo-dual de la triangulation de Delaunay contrainte. On peut le construire à partir de la triangulation contrainte en marquant tous les triangles ("aveugles" ou non) dont le sommet de Voronoi dual n'est pas visible depuis ce triangle, la visibilité étant bloquée par une ou plusieurs contraintes. Pour chaque sommet de Delaunay on peut ensuite construire sa cellule de Voronoi duale en circulant sur ses triangles incidents pour assembler des arêtes de Voronoi soit en reliant deux sommets de Voronoi dual de deux triangles successifs t1 et t2 (dans le cas général), soit en intersectant le segment de

Voronoi dual de l'arête entre t1 et t2 avec l'arête de contrainte la plus proche (ou les deux arêtes de contrainte les plus proches) bloquant la visibilité (d'autres cas se produisent au bord de l'enveloppe convexe lorsque les arêtes de Voronoi sont des rayons ou des droites).

Dans la démo on devra pouvoir saisir des segments de contraintes et/ou les charger à partir d'un fichier texte.

Avancé (optionnel) : Pouvoir déplacer un générateur serait un plus (attention : il faut stocker un vertex handle dans la triangulation puis faire des appels à remove/insert pour déplacer).

Articles à lire: partie BVD de "Interleaving Delaunay Refinement and Optimization for 2D Triangle Mesh Generation". Jane Tournois, Pierre Alliez and Olivier Devillers. 16th International Meshing Roundtable 2007: ftp://ftp-sop.inria.fr/geometrica/alliez/imr16-meshing.pdf

Codes et autres indications: La méthode décrite ci-dessus est naïve car effectue des tests exhaustifs sur les contraintes. En remarquant que l'ensemble des triangles aveugles est un ensemble de composantes connexes de triangles incidentes aux arêtes de contraintes, on peut pre-calculer le marquage des triangles aveugles en itérant sur les contraintes, et en propageant ledit marquage de triangle en triangle en partant d'un cote puis de l'autre de chaque contrainte. On peut en profiter pour stocker dans chaque triangle aveugle la contrainte qui bloque sa visibilité.

Codes et autres indications : Voir CGAL 2D Constrained Delaunay triangulations
http://www.cgal.org/Manual/last/doc_html/cgal_manual/Triangulation_2/Chapter_main.html#Section_35.8