Rapport de projet 150h

Outil de démonstration pour l'algorithme du dual

Romain Perrin

5 janvier 2019

mots-clefs : géométrie discrète, traitement d'image, C++, DGtal

Encadrement :
- Étienne Baudrier, équipe IMAGeS, ICube bureau C221 - baudrier@unistra.fr
- Loïc Mazo, équipe IMAGeS, ICube bureau C219 - loic.mazo@unistra.fr
- Adrien Krähenbühl, équipe IMAGeS bureau C226 - krahenbuhl@unistra.fr

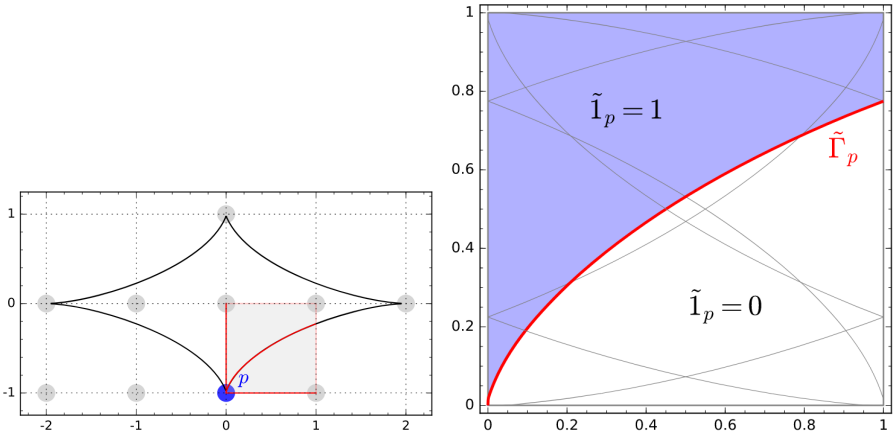
Je voudrais remercier Étienne Baudrier*, Loïc* Mazo *et Adrien* Krähenbühl *pour leur encadrement, suggestions et conseils qui m'ont beaucoup aidé tout au long de ce projet et sans qui ce projet serait probablement moins abouti.*

Table des matières

1 Dual de discrétisation

On s'intéresse dans le cadre de ce projet, à la discrétisation de formes paramétriques contraintes par une translation continue. On introduit la notion de dual de discrétisation. Le dual de discrétisation permet lorsque l'on applique une translation continue à une courbe, d'en représenter l'ensemble des discrétisations pour une résolution donnée.

Le dual en question permet d'évaluer et de comparer des estimateurs discrets (longueur, courbure...), sur des critères de robustesse. En effet, si deux discrétisations différentes représentent le même objet continu sous-jacent, on s'attend à ce qu'un estimateur robuste fournisse une estimation proche, sinon identique, par les deux discrétisations.



(a) Astroid (b) Dual

FIGURE 1 – (a) Une courbe continue (ici un astroid) (b) Le dual de discrétisation de la courbe (a) obtenu en ramenant chaque morceau de courbe dans le carré unité.

2 Objectifs

Ce projet comporte plusieurs objectifs convergeant vers un but commun : la construction d'un programme de démonstration de l'algorithme du dual destiné au journal en ligne IPOL [5].

Ce projet a la particularité de ne pas partir de rien, mais de s'articuler autour d'un développement préliminaire effectué dans le cadre d'un stage de M1 [4]. Un certain nombre de formats, comportements et traitements ont donc déjà été définis et l'objectif n'est ainsi pas de développer à nouveau les fonctionnalités mais de fonctionnaliser le code existant afin qu'il respecte les préceptes de la programmation orientée objet.

Dans un second temps, il s'agit d'optimiser le code en vue de sa publication dans le journal en ligne IPOL compte-tenu des restrictions de temps d'exécution appliqués aux démonstrations via l'outil en ligne.

3 Étude documentaire et étude de code

3.1 Étude documentaire

La première partie du travail consistait à prendre connaissance du cadre théorique relatif au dual de discrétisation. L'étude s'est notamment penchée sur les algorithmes de calcul du dual de discrétisation d'une forme paramétrique contrainte par un déplacement de type translation [3] et [1].

L'étude documentaire s'est également portée sur les algorithmes de calcul du noyau de discrétisation ainsi que des points de la toggling boundary d'une forme paramétrique tels que décrits dans [2].

Enfin, étant donné qu'un travail préliminaire avait déjà été réalisé pour implémenter un certain nombre de ces structures, formats et implémentation d'algorithmes, l'étude s'est terminée par la lecture du rapport de stage correspondant [4].

3.2 Étude du code existant

Comme dit précédemment, un travail préliminaire avait déjà été réalisé. Celui-ci avait pour objectif d'aboutir à l'écriture d'un programme utilisant les formes paramétriques et les estimateurs de longueur présents dans la bibliothèque de géométrie discrète DGtal [8], permettant de calculer le dual de discrétisation de chaque forme et d'effectuer des statistiques comparant ces estimateurs sur la discrétisation de la forme (algorithme naïf) et sur le dual de discrétisation de la forme (algorithme intelligent).

Ce travail préliminaire a donc permis de poser les bases structurelles, algorithmiques et comportementales du programme de démonstration réalisé dans le cadre de ce projet. En effet, les entrées-sorties, les algorithmes, traitements et structures de données utilisées avaient été fixées et ainsi ce projet consistait en la réalisation d'un programme de démonstration en reprenant une part significative du travail préliminaire.

4 Passage du code en objet

La première moitié du travail a donc consisté à développer un programme de démonstration de l'algorithme du dual. Bien que les algorithmes, structures de données, comportements et entrées-sorties aient été fixées par le travail préliminaire, celui-ci était un code de développement et de test non un produit fini, le programme comportant en tout et pour tout seulement deux fichiers cpp sans définition de classes permettant de tester rapidement le comportement et d'obtenir des résultats.

L'objectif premier a donc été de définir une organisation en classes respectant les préceptes de la programmation orientée objet. Le programme original était subdivisé en deux sous-programmes : un premier permettant de calculer le dual de discrétisation d'une forme et le second permettant d'effectuer soit une estimation au moyen de l'algorithme naïf, soit une estimation au moyen de l'algorithme intelligent (avec le dual).

Afin de mieux séparer les algorithmes, le programme réalisé est quant à lui subdivisé en trois sous-programmes : un premier nommé "computeDual" permettant de calculer le dual de discrétisation d'une forme, un second nommé "evalEstimNaive" permettant d'effectuer une estimation au moyen de l'algorithme naïf et un troisième nommé "evalEstimDual" permettant d'effectuer une estimation au moyen de l'algorithme intelligent en utilisant la structure de dual de discrétisation calculée par le premier sous programme.

Cette section décrit certains détails d'implémentation importants sur les quatre principales classes du programme. Le diagramme de classes UML est donnée en annexe [Annexe 1].

4.1 Classe Digitization

Afin de calculer le dual de discrétisation d'une forme paramétrique, il est nécessaire de disposer d'une discrétisation de ladite forme. La classe "Digitization" permet de générer une image représentant la discrétisation de l'une des cinq formes disponibles dans le programme de démonstration. Dans le programme réalisé lors du travail préliminaire, il y avait trois formes de la bibliothèque DGtal [8] (fleur, cercle et ellipse) et deux autre formes avaient été définies (astroid et lemniscate). Entre temps, ces deux classes ont été ajoutées à la bibliothèque et sont désormais toutes officielles.

La discrétisation utilisée est une discrétisation de Gauss et celle-ci est fournie par la bibliothèque DGtal au moyen de la classe "GaussDigitizer", celle-ci tirant partie du fait que les formes traitées sont des formes paramétriques étoilées. La classe de discrétisation est de fait contrainte à la seule utilisation des formes pouvant être discrétisées au moyen du GaussDigitizer de DGtal. La classe Dual est paramétrée par la forme à discrétiser et ses paramètres propres ainsi que la boîte englobante de la forme.

4.2 Classe Dual

La classe Dual permet de calculer le dual de discrétisation d'une des cinq formes proposée dans le programme de démonstration. Cette classe nécessite en entrée un objet de type Digitization contenant la discrétisation de la forme traitée. De fait la classe Dual est paramétrée par le discrétisation de la forme traitée.

Cette classe dispose d'une méthode principale "generateDual" calculant le dual de discrétisation. Pour ce faire, la classe dispose d'une méthode interne d'extraction de contour (__findBoundary), les formes de DGtal étant par défaut des formes pleines. L'extraction du contour utilise la 8-connexité pour la recherche du fond (méthode __neighbourBackground) et de fait le bord extrait est 4-connexe. Le dual de discrétisation est stocké en mémoire sous la forme d'une map de la STL associant à chaque point formant le bord de la forme, un point correspondant dans le dual.

La classe dispose d'une méthode saveDualAsSDP permettant de sauvegarder le dual de discrétisation dans un fichier au format SDP. Il est possible de sauvegarder une image de ce dual afin de le visualiser en utilisant la méthode saveDualAsPGM.

4.3 Classe Estimator

La classe Estimator contient des méthodes permettant d'effectuer une estimation de périmètre d'une des cinq formes proposées dans le programme de démonstration.

La méthode globalEstimateOnShapeDigitization est appliquée dans le cas de l'algorithme d'estimation naïf. Cette méthode effectue un estimation de longueur en utilisant la discrétisation de la forme, celle-ci étant obtenue par la méthode digitizeShape fournie par la classe.

La méthode estimateOnDualDigitization est appliquée dans le cas de l'algorithme intelligent. Elle effectue un estimation de longueur en utilisant le dual de discrétisation de la forme, celui-ci nécessitant d'être chargé en mémoire via l'utilisation de la méthode loadSupportDual. Il est à noter que le programme computeDual doit être lancé avant le programme evalEstimDual dans la mesure où le fichier contenant le dual de discrétisation nécessaire au fonctionnement de evalEstimDual est justement généré par computeDual.

La classe Estimator comporte également un ensemble de méthodes permettant de générer trois fichier en sortie :
- un fichier de statistiques au format CSV contenant pour chaque pas de translation, la valeur de l'estimation en ce pas et le temps de l'estimation. Ce fichier contient également la moyenne de l'estimation et la variance.
- un fichier DAT contenant un histogramme des estimations permettant d'évaluer les performances des estimateurs.
- un fichier image PPM permettant de visualiser l'erreur d'estimation utilisant un code tricolore bleu-noir-rouge ; bleu indiquant une sous-estimation, noir une convergence vers la valeur réelle et rouge une sur-estimation. Le pourcentage d'erreur relative indiqué par le code couleur est altérable via l'argument threshold de la méthode savePPM.

4.4 Classe EstimatorApplicator

La classe EstimatorApplicator est en quelque sorte un classe interface. Il ne s'agit pas d'une interface à proprement parler mais cette classe permet de rendre totalement abstrait les appels à la classe Estimator dont les profils de méthodes sont assez long.

Le cœur de cette classe réside dans le fait qu'elle implémente de manière générique les algorithmes naïf et intelligent. En effet cette classe dispose pour l'utilisateur de deux méthodes seulement : un constructeur par le biais duquel celui-ci choisi l'algorithme à utiliser pour l'estimation et une méthode run effectuant l'estimation à proprement parler.

Pour ce faire, la classe dispose d'une méthode interne __naiveAlgo implémen-

tant l'algorithme naïf, spécialisée pour chacune des cinq formes disponible. De manière similaire, la classe dispose d'une méthode interne __smartAlgo implémentant l'algorithme intelligent et spécialisée pour chaque forme.

La classe dispose également d'une méthode interne générique __inputShapeParameters qui originellement permettait de manière interactive de rentrer les paramètres de la forme. Le journal en ligne IPOL demandant qu'il n'y ai aucune interaction homme-machine à l'exécution [6], les paramètres de la forme sont désormais prodigués au programme par l'intermédiaire d'arguments en ligne de commande. Cette méthode ne réalise de fait plus que les calculs de boites englobantes et de certains paramètres découlants de la paramétrisation de la forme.

5 Optimisations temporelles

La seconde grande partie du projet a consisté à optimiser le programme de démonstration du point de vue temporel. En effet, IPOL contraint la durée d'exécution des programmes hébergés à 30 secondes [6]. De fait, si des optimisations temporelles peuvent êtres réalisées, il est logique d'effectuer ces optimisation toujours dans cette optique de publication dans IPOL.

Cette section décrit donc les différentes optimisations temporelles mises en place et présente les résultats obtenus pour chacun des trois sous-programmes.

5.1 Programme computeDual

Le sous-programme computeDual concerne les classes Digitization et Dual ; les optimisations éventuelles peuvent donc être mises en place dans les méthodes de ces deux classes.

5.1.1 Classe Digitization

En dehors du constructeur, la classe Digitization ne comporte en tout et pour tout qu'une seule méthode : digitizeShape. Cette méthode parcourt les pixels du domaine de discrétisation et détermine si le pixel (i,j) est à l'intérieur de la forme ou non.

Étant donné que cette opération est locale au pixel et qu'il n'y a pas d'influence du voisinage du pixel (ie le fait que l'un des voisins du pixel traité soit à l'intérieur de la forme n'implique pas que le pixel traité l'est aussi), il n'y a pas d'ordre de parcours imposé. Les pixels peuvent donc être parcourus indépendamment et la méthode est de fait parallélisable.

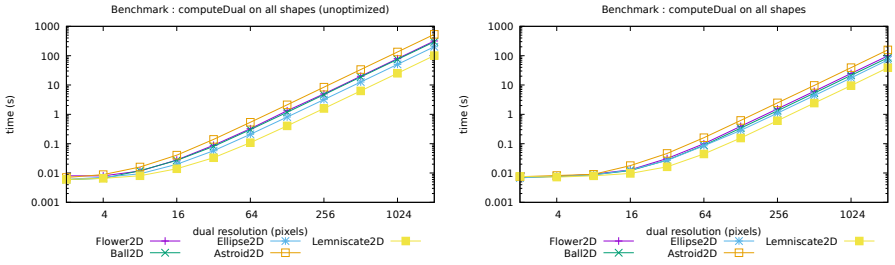
La méthode digitizeShape a été parallélisée en utilisant l'API de spécification OpenMP [7].

5.1.2 Classe Dual

La classe Dual comporte plus de méthodes, mais toutes ne sont pas parallélisables. Les méthodes saveDualAsSDP et saveDualAsPGM concernent la sauvegarde de points (resp. de pixels) et sont contraintes de s'effectuer de manière purement séquentielle.

En revanche, la méthode __findBoundary qui extrait la frontière de la forme discrétisée est parallélisable. En effet, l'image binaire obtenue au moyen de la classe Digitization est parcourue et pour chaque pixel dont la valeur est 1 (ie le pixel est à l'intérieur de la forme), on regarde si au moins un point de son 8-voisinage appartient au fond. Étant donné que l'image binaire n'est pas altérée par la méthode de recherche du bord, que tous les pixels de l'image sont parcourus et que le critère de décision est local uniquement, le traitement peut être effectué en parallèle sur tous les point de l'image. À nouveau, la méthode a été parallélisée par une directive OpenMP [7].

5.1.3 Performances



(a) Non optimisé (b) Optimisé

FIGURE 2 – (a) Les temps de calcul du sous-programme computeDual sur les formes sans optimisation. (b) Les temps de calcul du sous-programme computeDual sur les formes avec optimisation.

Afin de réaliser des tests de performances pour déterminer l'impact des optimisations effectuées, plusieurs scripts shell on été écrits. Ceux-ci permettent de lancer n itérations du sous-programme pour chaque pas de résolution entre deux bornes et de calculer le temps moyen d'exécution dudit sous-programme, puis de générer des courbes permettant de comparer les deux versions.

Les tests réalisés montrent un gain significatif permettant d'utiliser des résolutions plus adaptées (256, 512, 1024) dans la démonstration en ligne, ce qui n'aurait pas été possible avec la version non optimisée. Une grille détaillant les performances sur chaque forme est disponible en annexe [Annexe 2].

5.2 Programme evalEstimNaive

Le sous-programme evalEstimNaive effectue une estimation de périmètre sur une forme donnée en utilisant un estimateur donné. On s'intéresse dans cette partie à la possibilité d'accélérer cette estimation. Deux classes sont concernées : Estimator qui contient les méthodes permettant d'effectuer une estimation en un pixel et EstimatorApplicator qui contient l'implémentation de l'algorithme naïf (__naiveAlgo).

5.2.1 Classe EstimatorApplicator

EstimatorApplicator contient l'implémentation de l'algorithme d'estimation naïf en la méthode __naiveAlgo spécialisée pour chacune des cinq formes possibles. La méthode consiste pour chaque forme à parcourir le carré unité [0,1 [, instancier la forme centrée en (x-i,y-j) où (x,y) est le centre de la forme et (i,j) la position courante dans le carré unité, puis appeler la méthode globalEstimateOnShapeDigitization de la classe Estimator qui réalise l'estimation au point courant.

La forme instanciée ne dépend que des paramètres i et j qui sont propres à chaque point. La méthode globalEstimateOnShapeDigitization de la classe Estimator stocke les valeurs estimées (et les pas de translations) dans un vecteur qui du fait de l'exécution séquentielle de la méthode est rempli dans l'ordre de parcours (ligne par ligne) qui est donc fixe.

Le seul obstacle à la parallélisation de la classe est donc le stockage du résultat. Il suffirait de fait de changer la structure de donnée en une structure par dictionnaire pour pouvoir traiter les pixels dans un ordre aléatoire.

La méthode __naiveAlgo a donc été parallélisée en utilisant une directive OpenMP afin de traiter les pixels en parallèle.

5.2.2 Classe Estimator

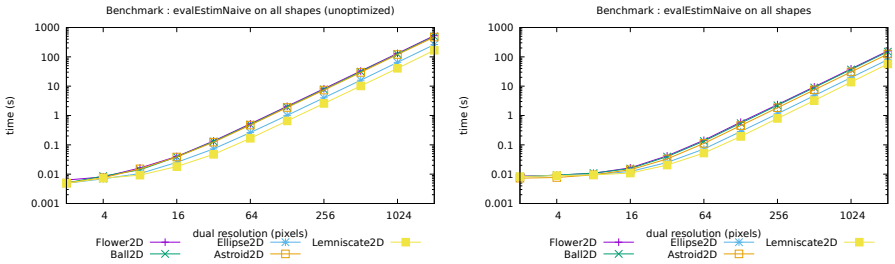
Du fait de la parallélisation de la méthode __naiveAlgo, la structure de donnée pour le stockage des pas de translation, des temps et valeurs d'estimation a du être modifiée. Étant donné que les coordonnées des points dans le parcours du carré unité sont uniques, il peuvent servir de clef pour le stockage des autres valeurs.

La structure a donc été modifiée de la sorte : les trois vecteurs contenant chacun respectivement les pas de translation, les temps d'estimation et les valeurs d'estimation on été remplacés par une map de la STL indexée sur les coordonnées du pixel dans le carré unité (i,j) et contient un triplet de valeurs (pas de translation, temps d'estimation et valeur d'estimation).

Afin d'éviter de possibles erreurs de ré-allocation lors de l'insertion de valeurs

dans la map par différents threads, une méthode reserveMap a été également ajoutée à la classe Estimator. Celle-ci permet de remplir la map avec des valeurs temporaires (la taille étant connue mais la classe map de contenant pas de méthode de réservation) et ainsi d'éviter ce problème.

5.2.3 Performances



(a) Non optimisé (b) Optimisé

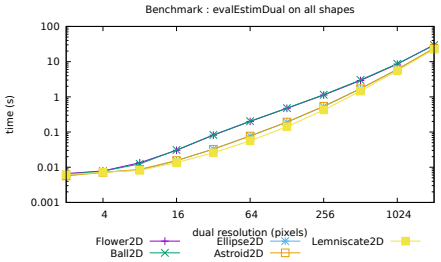
FIGURE 3 – (a) Les temps de calcul du sous-programme evalEstimNaive sur les formes sans optimisation. (b) Les temps de calcul du sous-programme evalEstimNaive sur les formes avec optimisation.

De la même manière que pour le sous-programme computeDual, des scrits shell ont été écrits pour évaluer les performances du sous-programme evalEstimNaive.

Les tests réalisés montrent un gain significatif permettant d'utiliser des résolutions plus adaptées (256, 512, 1024) dans la démonstration en ligne, ce qui n'aurait pas été possible avec la version non optimisée. Une grille détaillant les performances sur chaque forme est disponible en annexe [Annexe 3].

Il est à noter cependant que le point central de la démonstration est le sous-programme evalEstimDual et que le sous-programme evalEstimNaive n'est présent qu'à titre de comparaison des algorithmes.

5.3 Programme evalEstimDual



(a) Non optimisé

FIGURE 4 – (a) Les temps de calcul du sous-programme evalEstimDual sur les formes sans optimisation.

Le sous-programme evalEstimDual permet d'effectuer une estimation de périmètre sur une des formes proposées en implémentant un algorithme intelligent. Celui-ci utilise le dual de discrétisation calculé par le sous-programme computeDual ; les deux fonctionnent de fait en étroite collaboration.

Il y a cependant quelques obstacles à la parallélisation de cet algorithme. En effet, il est nécessaire d'utiliser une connaissance dépendant d'un parcours fixé. Lorsque l'on cherche à évaluer en un pixel (du dual de discrétisation), il est nécessaire de connaître le nombre de courbes passant par le pixel précédent ainsi que la valeur de l'estimation précédente. Cela force donc l'algorithme à être effectué de manière séquentielle.

Ce sous-programme n'a donc pas été optimisé. Toutefois, il est à noter que les temps d'exécution de ce sous-programme sont significativement plus faible que ceux de l'estimation naïve (evalEstimNaive) du fait qu'il n'y ait aucune discrétisation effectuée (la seule discrétisation est réalisée par computeDual) et que la taille du dual de résolution est inférieure à la taille de l'image de la discrétisation traitée par l'estimation naïve. En effet, celle-ci traite une image ayant la dimension de l'image de la forme discrétisée multipliée par la résolution du dual.

Une grille détaillant les performances sur chaque forme est disponible en annexe [Annexe 4].

6 Résultats

6.1 Détails

On peut visuellement parlant constater que les résultats obtenus avec l'algorithme naïf et ceux obtenus avec l'algorithme intelligent sont en tout point similaires. Les annexes montrent les résultats obtenus pour les cinq formes avec les quatre estimateurs avec une résolution de 1024 par 1024 [Annexes 5 et 6]. Les couleurs bleu, rouge et noir représentent respectivement une sous-estimation, une sur-estimation et une convergence vers la valeur réelle de périmètre (connue car les formules paramétriques des formes sont connues). Les images sont seuillées ici à une erreur relative de 10% et l'on peut faire quelques observations. Les histogrammes des valeurs estimées pour chaque forme et chaque estimateur sont disponibles en annexe [Annexes 7 et 8]

- L'estimateur DSS (Digital Straight Segments) donne de mauvais résultats visuels : les images sont bruitées un problème déjà présent dans [4]. Cela peut être du à l'implémentation de l'algorithme DSS dans DGtal et de la structure de données utilisée pour représenter le contour de la forme. En effet, le contour est stocké comme une liste de points non ordonnés. Il est possible que l'implémentation du DSS nécessite une liste de points contigus. Or tant dans l'implémentation de l'algorithme naïf que dans celle de l'algorithme intelligent, les points ne sont pas contigus ; pire encore dans la classe Dual la méthode d'extraction de contour (__findBoundary) est entièrement parallélisée. Il serait cependant possible d'essayer de corriger le problème en définissant une méthode de tri des points du bord. Il suffirait de noter les points déjà triés et de rechercher le point parmi les voisins du dernier point trié, celui qui est le suivant dans l'ordre de parcours choisi (en 4-connexité). Il est à noter cependant que statistiquement, l'estimateur DSS converge assez fidèlement vers la valeur réelle de périmètre des formes similairement à ce qui avait été observé lors du travail préliminaire sur des résolutions plus faibles [4].

- L'estimateur L1 a une forte tendance à sur-estimer et ce sur toutes les formes testées.

- L'estimateur FPL produit généralement de bon résultats tant en termes visuels que statistiques. Si l'on regarde le côté visuel on peut constater que FPL a tendance généralement (mais pas systématiquement ie pour le cercle) à effectuer à la fois une sous-estimation et une sur-estimation, encadrant assez fidèlement la valeur réelle du périmètre.

- L'estimateur MLP a une tendance à sous-estimer le périmètre. Cela est particulièrement visible par le fait que les images sont seuillées à 10% et qu'elles sont pratiquement complètement bleues.

- La lemniscate présente des problèmes qui restent pour l'instant sans réponse.

Si l'on observe les histogrammes des quatre estimateurs appliqués à la lemniscate, on peut constater l'apparition de deux classes de valeurs contiguës : une première classe relativement proche de la valeur réelle suivant l'estimateur considéré et une seconde classe sous-estimant de manière très significative le périmètre.

6.2 Ouverture

Il serait intéressant par le suite de faire varier un plus grand nombre de paramètres dans les tests. Seuls les résolutions et les estimateurs ont été testés. Il serait donc tout à fait logique de tester également des formes plus grandes non testées ici par faute de temps d'une part et car l'intérêt premier de ce travail était de fournir une démonstration contrainte en terme de temps d'exécution.

À cet égard, les scripts permettant de générer les tests de performances sont paramétrables et permettent, moyennant quelques ajustements mineurs de regénérer entièrement les courbes et statistiques utilisés dans ce rapport. Les performances bien sûr, sont variables suivant la machine sur laquelle le programme est exécuté. Les tests de performances effectués dans le cadre de ce rapport ont été effectués sur un PC portable équipé d'un processeur Intel Core i7-4710MQ cadencé à 2.5 GHz possédant 4 cœurs physiques (8 logiques) et pouvant exécuter 8 threads en parallèle.

7 Bibliographie

[1] Mazo, L., Baudrier, É : "Object digitzation up to a translation", J Computer and System Science (2017), doi :10.1016/j.jcss.2018.08.001

[2] Mazo, L., Baudrier, É : "Study on the digitization dual combinatorics and convex case", in W.G. Kropatsch, N.M. Artner, I. Janush (Eds), Discrete Geometry for Computer Imagery, DGCI 2017, Lecture Notes in Computer Science, vol. 10502, pp 363-374, Springer, Heidelberg, IAPR (2017), doi :10.1007/978-3-319-66272-5

[3] Mazo, L., Baudrier, É. : "Combinatorics of the Gauss digitization under translation in 2D"

[4] Fellah, C. "Génération de toutes les discrétisations d'une courbe plane comparaison des estimateurs de longueurs", rapport de stage, Université de Strasbourg, 2018

[5] Site du journal en ligne IPOL : https://www.ipol.im/

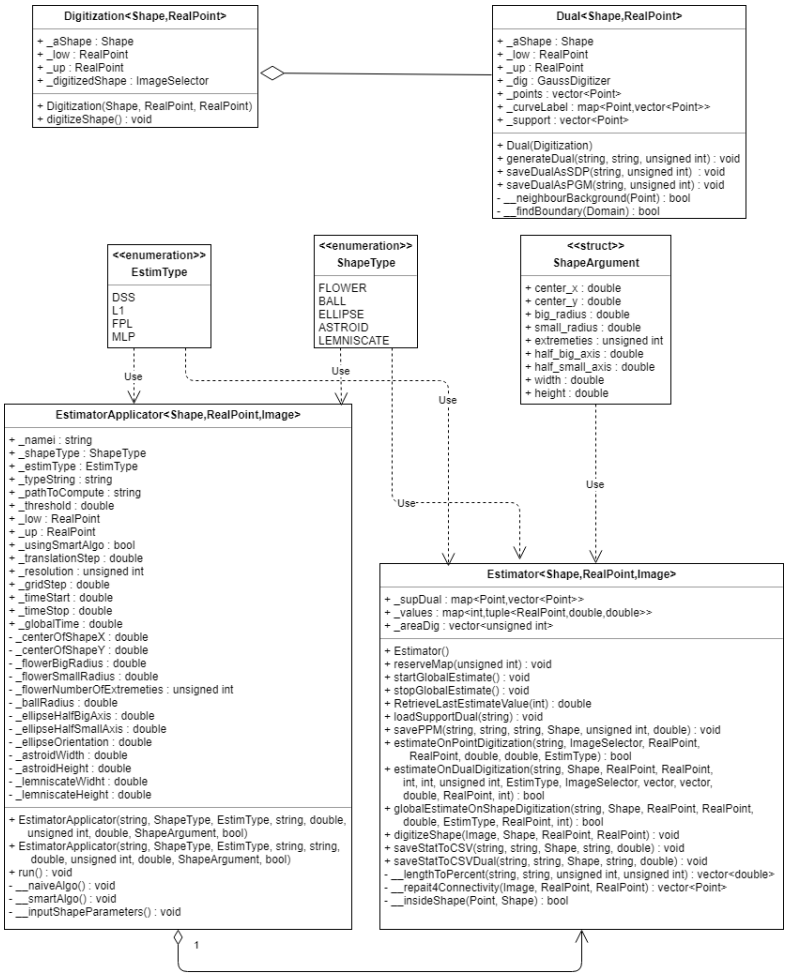
[6] IPOL sofware guidelines : https://tools.ipol.im/wiki/ref/software_guidelines/

[7] site de OpenMP : https://www.openmp.org

[8] site de la bibliothèque de géométrie discrète DGtal : https://dgtal.org/

8 Annexes

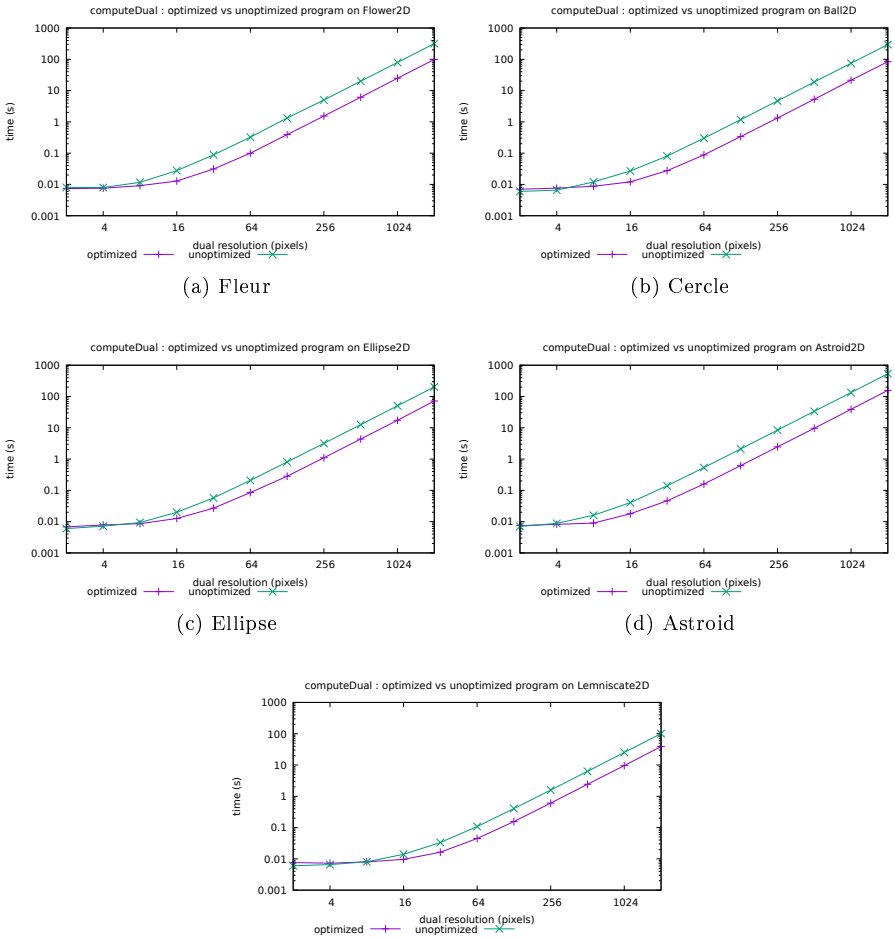
8.1 Diagramme de classes UML



(a) Diagramme de classes UML

FIGURE 5 – (a) Le diagramme de classes UML du programme de démonstration réalisé

8.2 Performances - computeDual



(a) Fleur

(b) Cercle

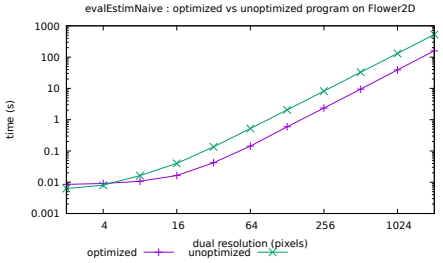
(c) Ellipse

(d) Astroid

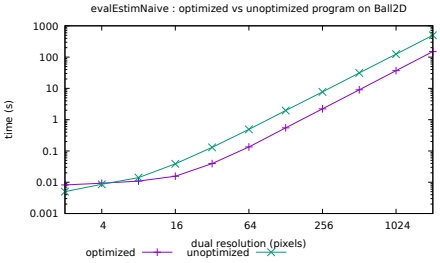
(e) Lemniscate

FIGURE 6 – (a-e) Pour chaque forme, la comparaison entre la version optimisée et la version non optimisée du sous-programme de calcul du dual de discrétisation computeDual.

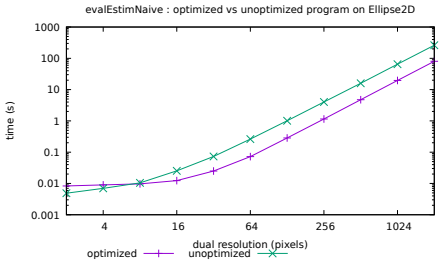
8.3 Performances - evalEstimNaive



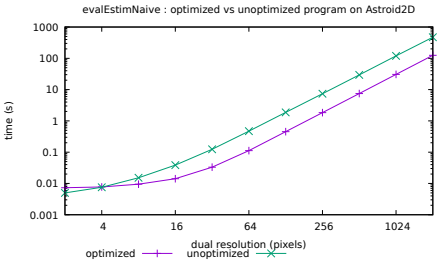
(a) Fleur



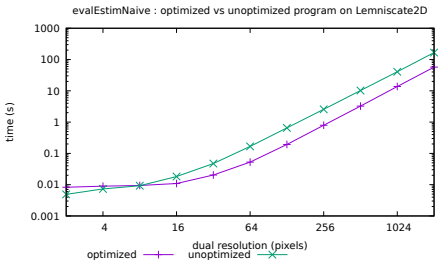
(b) Cercle



(c) Ellipse



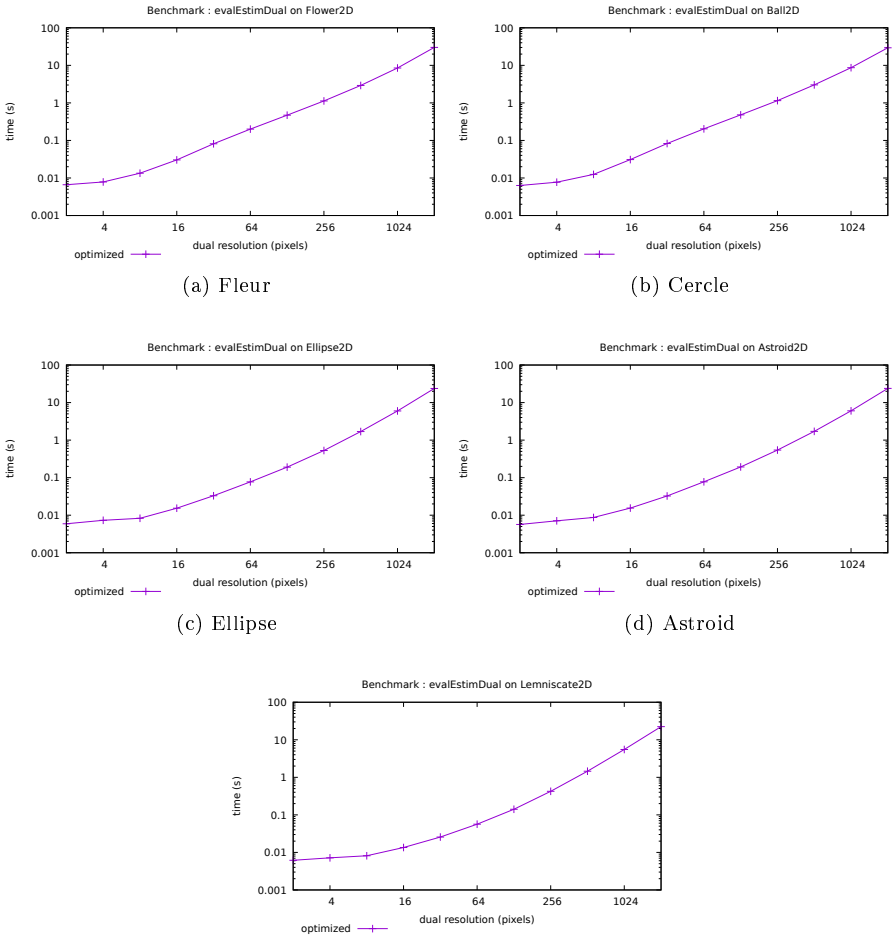
(d) Astroid



(e) Lemniscate

FIGURE 7 – (a-e) Pour chaque forme, la comparaison entre la version optimisée et la version non optimisée du sous-programme d'estimation naïve evalEstimNaive.

8.4 Performances - evalEstimDual



(a) Fleur

(b) Cercle

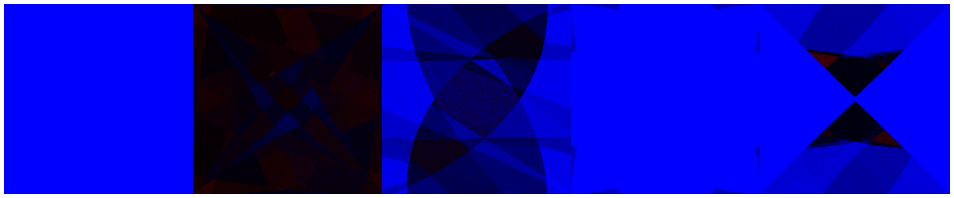
(c) Ellipse

(d) Astroid

(e) Lemniscate

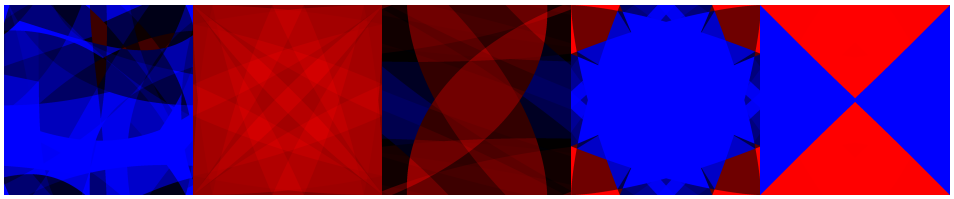
FIGURE 8 – (a-e) Pour chaque forme, la comparaison entre la version optimisée et la version non optimisée du sous-programme d'évaluation avec dual de discrétisation evalEstimDual.

8.5 Résultats des estimations naïves

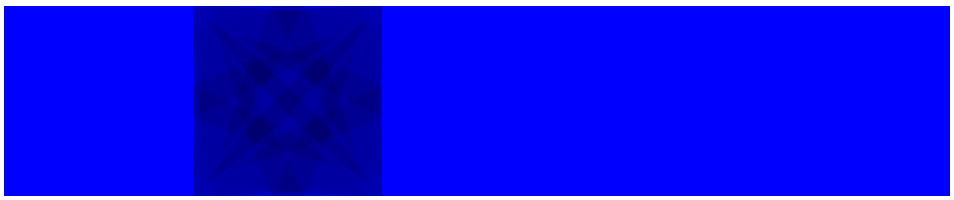


(a) Fleur - DSS (b) Cercle - DSS (c) Ellipse - DSS (d) Astroid - DSS (e) Lemniscate - DSS

(f) Fleur - L1 (g) Cercle - L1 (h) Ellipse - L1 (i) Astroid - L1 (j) Lemniscate - L1



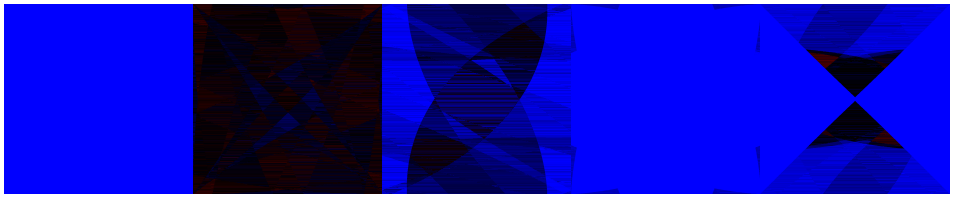
(k) Fleur - FPL (l) Cercle - FPL (m) Ellipse - FPL (n) Astroid - FPL (o) Lemniscate - FPL



(p) Fleur - MLP (q) Cercle - MLP (r) Ellipse - MLP (s) Astroid - MLP (t) Lemniscate - MLP

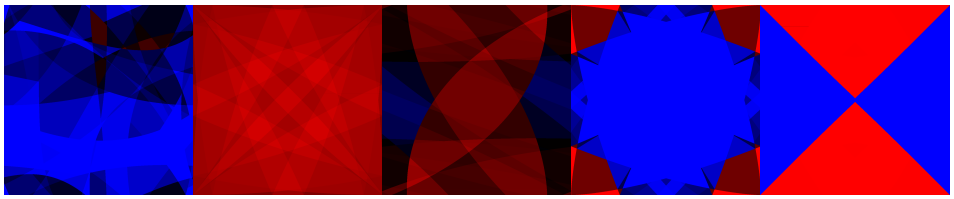
FIGURE 9 – Les résultats de l'estimation naïve des estimateurs sur les formes.

8.6 Résultats des estimations avec dual

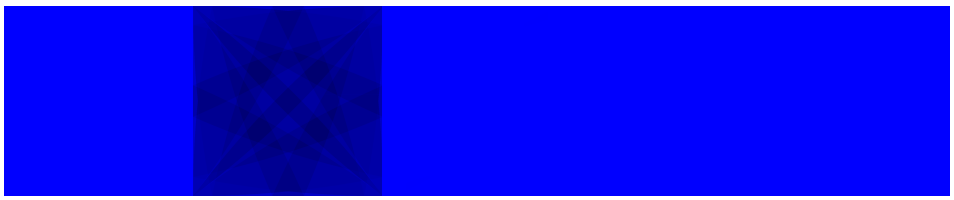


(a) Fleur - DSS (b) Cercle - DSS (c) Ellipse - DSS (d) Astroid - DSS (e) Lemniscate - DSS

(f) Fleur - L1 (g) Cercle - L1 (h) Ellipse - L1 (i) Astroid - L1 (j) Lemniscate - L1



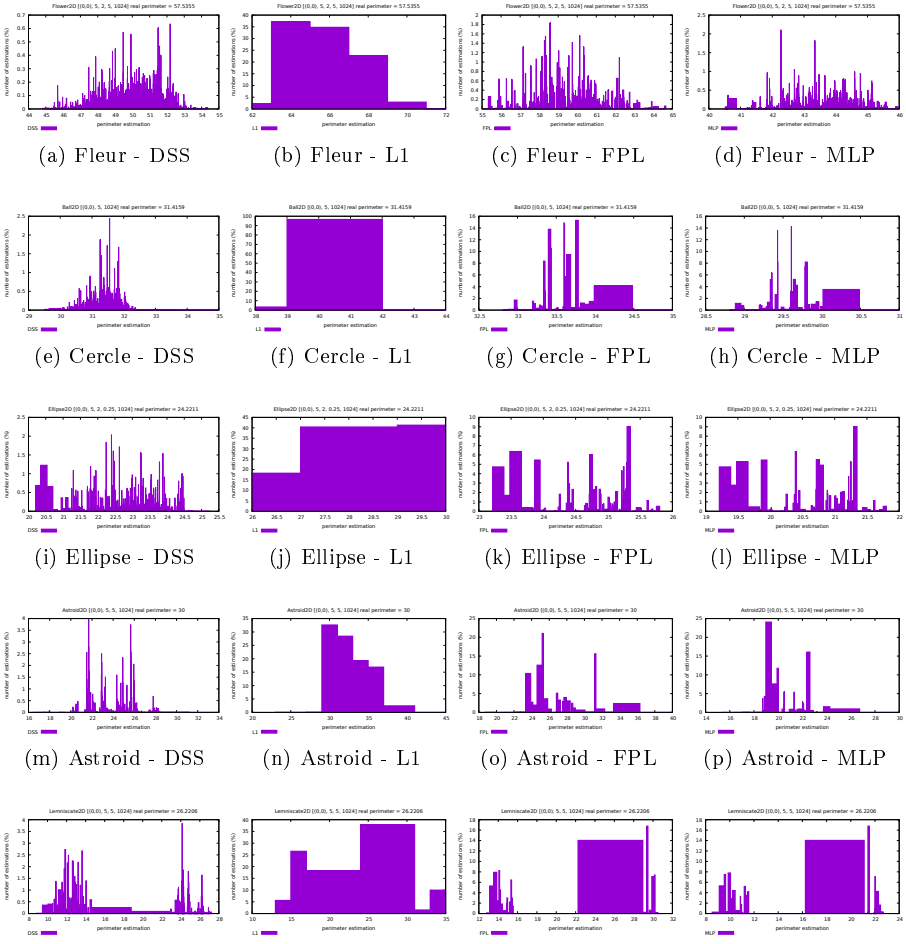
(k) Fleur - FPL (l) Cercle - FPL (m) Ellipse - FPL (n) Astroid - FPL (o) Lemniscate - FPL



(p) Fleur - MLP (q) Cercle - MLP (r) Ellipse - MLP (s) Astroid - MLP (t) Lemniscate - MLP

FIGURE 10 – Les résultats de l'estimation duale des estimateurs sur les formes.

8.7 Performances des estimateurs - Algorithme naïf



(a) Fleur - DSS (b) Fleur - L1 (c) Fleur - FPL (d) Fleur - MLP

(e) Cercle - DSS (f) Cercle - L1 (g) Cercle - FPL (h) Cercle - MLP

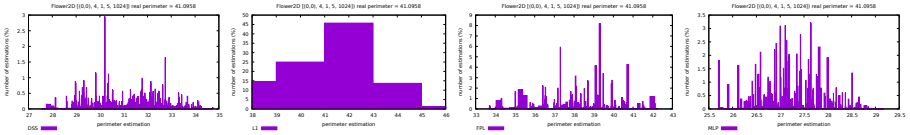
(i) Ellipse - DSS (j) Ellipse - L1 (k) Ellipse - FPL (l) Ellipse - MLP

(m) Astroid - DSS (n) Astroid - L1 (o) Astroid - FPL (p) Astroid - MLP

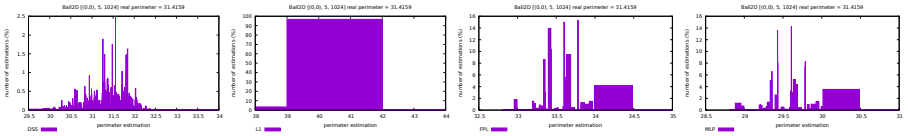
(q) Lemniscate - DSS (r) Lemniscate - L1 (s) Lemniscate - FPL (t) Lemniscate - MLP

FIGURE 11 – Histogrammes des valeurs estimées par l'algorithme naïf.

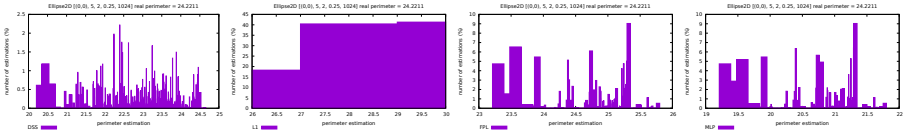
8.8 Performances des estimateurs - Algorithme intelligent



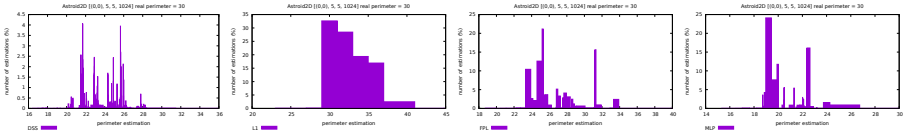
(a) Fleur - DSS (b) Fleur - L1 (c) Fleur - FPL (d) Fleur - MLP



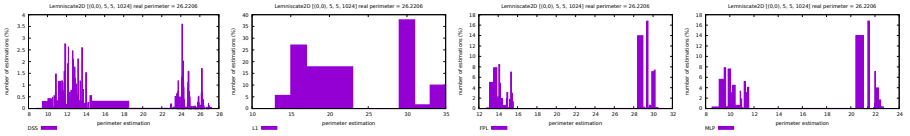
(e) Cercle - DSS (f) Cercle - L1 (g) Cercle - FPL (h) Cercle - MLP



(i) Ellipse - DSS (j) Ellipse - L1 (k) Ellipse - FPL (l) Ellipse - MLP



(m) Astroid - DSS (n) Astroid - L1 (o) Astroid - FPL (p) Astroid - MLP



(q) Lemniscate - DSS (r) Lemniscate - L1 (s) Lemniscate - FPL (t) Lemniscate - MLP

FIGURE 12 – Histogrammes des valeurs estimées par l'algorithme intelligent.