Triangulation de Delaunay
Rapport de projet de programmation
L3 Informatique, parcours Science Informatique

Losekoot Théo, Piau Rémi, Thuillier Kerian

ENS Rennes et ISTIC, Université Rennes 1

10 mars 2019

Résumé

Dans ce rapport, nous présentons notre travail sur la triangulation de Delaunay et les différentes extensions que nous avons réalisées autour de ce sujet.

Mots-clés : programmation ; triangulation ; delaunay ; ocaml ; 3D

Classification ACM : I.3.5 : computationnal geometry

Table des matières

1 Introduction **2**

1.1 Principe . . . 2

1.1.1 Historique . . . 2

1.1.2 Applications . . . 2

2 Contribution **2**

2.1 Algorithme : une triangulation de Delaunay incrémentale . . . 2

2.2 Implémentation . . . 3

2.2.1 Un code modulaire . . . 3

2.2.2 Validation . . . 3

3 Extensions **3**

3.1 Interface utilisateur . . . 4

3.2 Gradient de couleurs . . . 4

3.3 Étude de surfaces en 3 dimensions . . . 5

3.4 Complexité . . . 5

4 Conclusion **5**

A Bibliographie **7**

B Annexes **7**

1 Introduction

1.1 Principe

Lors de ce projet, nous nous sommes interessés à la triangulation de Delaunay. Le but de cette triangulation est de construire des triangles reliant un ensemble de points, tels que ces derniers soient le plus régulier possible, c'est-à-dire de maximiser le plus petit angle pour chacun des triangles construits. Nous pouvons voir un exemple de triangulation de Delaunay en [1].

1.1.1 Historique

Cette triangulation a été inventée par le mathématicien Boris Delaunay en 1924. Élève de Voronoï, il est donc peu surprenant qu'il ait continué dans la lignée de son maître, d'où le lien direct entre le diagramme de Voronoï d'un ensemble de points P et la triangulation de Delaunay de P qui est le graphe dual de ce premier.

1.1.2 Applications

La triangulation de Delaunay a de nombreuses applications. Elle est par exemple
— utilisée pour le calcul de la superrésolution [1], soit obtenir une image de grande qualité à partir d'un ensemble d'images de qualités inférieures.
— utilisée pour modéliser des objets en trois dimensions.

2 Contribution

2.1 Algorithme : une triangulation de Delaunay incrémentale

Pour réaliser la triangulation de Delaunay, nous appliquons l'algorithme [1] à un ensemble de points. Cet algorithme est illustré par un exemple dans [2].
Nous partons d'une triangulation de Delaunay à laquelle nous ajoutons les points un à un en faisant le nécessaire pour que la triangulation reste de Delaunay après chaque ajout.
Pour cela, il est nécessaire de déterminer si un point P appartient au cercle circonscrit d'un

1. Pour plus d'informations, `http://www.tsi.telecom-paristech.fr/pages/enseignement/ressources/beti/delaunay/delaunay_imprimable.pdf`.

triangle ABC : il suffit de vérifier le déterminant de la matrice suivante.

$$\begin{vmatrix} A_x & A_y & A_x^2 + A_y^2 & 1 \\ B_x & B_y & B_x^2 + B_y^2 & 1 \\ C_x & C_y & C_x^2 + C_y^2 & 1 \\ P_x & P_y & P_x^2 + P_y^2 & 1 \end{vmatrix}$$

Si le déterminant de cette matrice est positif, alors le point P est dans le cercle trigonométrique du triangle ABC.

Pour que cela soit possible, notre algorithme impose que les points à ajouter soient strictement à l'intérieur de l'enveloppe convexe de la triangulation de départ.

2.2 Implémentation

2.2.1 Un code modulaire

Pour ce projet nous avons créé différentes structures de données que nous avons séparées dans des modules (avec des types opaques) mais compilées en un seul fichier `cma` pour faciliter leur utilisation. Ces modules de gestion des données sont :

- `Point` ;
- `Matrix` pour les matrices et le calcul du déterminant ;
- `Triangle` pour les triangles ;
- `Circle` pour les cercles ;
- `Aset` pour des ensembles polymorphes.

La partie affichage et gestion de l'interface utilisateur a elle aussi été modularisée afin de faciliter son utilisation. Nous disposons en effet de deux modules `Draw` et `Render` s'occupant du dessin des ensembles de points ou de triangles, des gradients pour le premier, et de la gestion de l'interaction avec l'utilisateur pour le second.

Nous pouvons voir les dépendances principales entre les différents modules dans le diagramme de classes [3]. Ces modules ont des fonctions de construction avec un nom standard [4], les fonctions remplissant le même objectif portent le même nom, ce qui facilite de fait leur emploi.

2.2.2 Validation

Nous avons voulu valider l'implémentation de nos modules pour nous assurer de leur bon fonctionnement. Pour ce faire, nous avons procédé à la validation des fonctions de chacun des modules par des tests d'intégration et unitaires utilisant des valeurs aléatoires, lorsque cela était possible, et testant les cas connus comme étant problématiques.

3 Extensions

Après avoir implémenté l'algorithme de Delaunay tel que demandé, nous nous sommes attaqués à la réalisation de plusieurs extensions comme l'affichage des cercles circonscrits, la

coloration des triangles et l'études de fonctions à trois dimensions que nous allons vous présenter maintenant. Nous avons également ajouté une interface utilisateur permettant d'afficher uniquement les éléments désirées.

3.1 Interface utilisateur

Nous avons réalisé une interface utilisateur qui réagit aux actions de l'utilisateur à la fois à la souris et au clavier. Pour cela, nous avons suivi l'implémentation modulaire proposée par l'INRIA dans son guide sur la bibliothèque `graphics` [2]. Nous avons ainsi utilisé une boucle qui attend les actions utilisateurs. Ces actions modifient un environnement qui est une variable de type `environnement` [6]. Nous procédons ensuite à l'affichage de ce nouvel environnement avant d'attendre à nouveau une entrée utilisateur (cf organigramme [5]).

Le type `environnement` est composé de nombreuses variables mutables dont les principales sont la liste de points à trianguler, la liste de triangles de la triangulation courante et des booléens qui annonce l'affichage ou non d'un objet.
Notre interface graphique peut ainsi réaliser les opérations suivantes :

- afficher les points/triangles/cercles circonscrits aux triangles de la triangulation ;
- générer un nouvel ensemble de points et le trianguler : cet ensemble contient exactement le même nombre de points que le précédent ;
- ajouter ou retirer des points à la triangulation : un clique de souris permet d'en ajouter un à l'emplacement de la souris, et des touches du clavier permettent de re-générer un pavage avec dix points de plus ou de moins ;
- afficher les triangles de la triangulation avec un gradient de couleurs et naviguer entre les trois types de gradients (voir la section suivante) et en modifier les couleurs.

3.2 Gradient de couleurs

Nous avons également proposé une amélioration visuelle permettant de colorer les triangles avec un gradient entre deux couleurs. Ces gradients se basent sur la position du centre du cercle circonscrit à chaque triangle par rapport à x et/ou y pour définir leur couleur. Nous avons choisi d'utiliser le centre du cercle circonscrit pour déterminer la couleur d'un triangle car nous disposions déjà des coordonnées de ce point et que celui-ci est unique pour chaque triangle.

Pour déterminer la couleur d'un triangle, nous avons réaliser des fonctions prenant deux triplets de réels représentant des couleurs au format *rgb* (*Red Green Blue*) ainsi qu'une valeur maximale à partir de laquelle tous les triangles seront de la couleur finale : $rgb_1 = (r_1, g_1, b_1)$ et $rgb_2 = (r_2, g_2, b_2)$. Il faut ainsi réaliser un coefficient représentant la position du triangle par rapport à la valeur maximale, ce coefficient devant impérativement être compris entre 0 et 1 pour ne pas retourner une couleur abérrante. Ce coefficient c obtenu, il suffit d'appliquer l'opération ci-dessous pour obtenir le triplet de réels définissant la couleur du triangle : $rgb = (r, g, b)$ où,

$$r = r_1 + c \times (r_2 - r_1) \text{ et } g = g_1 + c \times (g_2 - g_1) \text{ et } b = b_1 + c \times (b_2 - b_1)$$

Nous avons implémenté trois types de gradients : un gradient selon l'axe des abscisses (voir [7]), selon l'axe des ordonnées (voir [8]) ainsi qu'un gradient circulaire (voir [9]).

2. Voir : `https://caml.inria.fr/pub/docs/oreilly-book/pdf/chap5.pdf`.

3.3 Étude de surfaces en 3 dimensions

Nous avons prolongé notre approche de la coloration avec une coloration dépendante d'une fonction $f : \mathbb{R}^2 \to \mathbb{R}$ continue. Ainsi, nous triangulons les points du plan grace à l'algorithme de triangulation de Delaunay puis nous colorions les triangles en fonction de leur inclinaison dans l'espace : plus un triangle est horizontal plus celui-ci est clair ; plus il est vertical plus il est foncé.

Posons un triangle de l'espace t_3 défini par ces trois sommets :

$$(A, B, C) = ((x_1, y_1, f(x_1, y_1)), (x_2, y2, f(x_2, y_2)), (x_3, y_3, f(x_3, y_3)))$$

à partir d'un triangle $t = ((x_1, y_1), (x_2, y2), (x_3, y_3))$ du plan.

Pour calculer l'inclinaison de ce triangle, l'angle α, nous procédons comme suit : posons $\vec{u} = \vec{AB}$ et $\vec{v} = \vec{AC}$. Puis nous posons $\vec{n} = \vec{u} \wedge \vec{v}$ et $\vec{u_z} = (0, 0, 1)$, nous obtenons alors

$$\frac{||\vec{n} \wedge \vec{u_z}||^2}{||\vec{n}||^2} = sin(\alpha)^2$$

et nous récupérons α facilement grâce à une racine et un arcsinus avant de normaliser cet angle et déterminer la couleur du triangle.

Pour s'assurer d'une bonne représentation, nous avons créé un remplissage intelligent qui évalue la fonction le long d'une grille et rajoute des points si elle varie trop entre deux points de la grille. Cette fonction part d'une grille de points initiale, et ajoute de nouveaux points entre 2 des points de la grille si la variation de la fonction entre deux des points est supérieure à un seuil. Nous pouvons voir un exemple avec la fonction $(x, y) \longmapsto sin(x + y)$ sur la figure [10].

3.4 Complexité

Nous nous sommes ensuite intéressés à la complexité de notre algorithme. Pour la mesurer, nous avons compté le nombre d'appels à la fonction `in_circle` . Cette mesure est significative car la fonction `in_circle` est appellée à chaque ajout de point, et le nombre d'appels par ajout de point est lié au nombre de points déjà sur le plan.

Nous avons donc effectué une régression linéaire sur la fonction $f(\log x) = \log(y)$ avec x le nombre de points et y le nombre d'appels. La pente de cette régression linaire est :

$$a = 1.944 \pm 0.01241$$

Nous avons obtenu ces valeurs et les courbes avec le langage R.
Ce coefficient montre que notre algorithme est en $O(n^2)$, n étant le nombre de points à trianguler. Nous pouvons le voir sur la figure [11] et [12], qui sont les courbes expérimentales des valeurs qui nous ont permis de calculer la complexité.

4 Conclusion

Pour réaliser ce projet, nous avons donc réalisé un algorithme incrémental de la triangulation de Delaunay. Pour cela, nous avons utilisé un code modulaire et testé chacune des fonctions de ces modules à l'aide de tests unitaires et d'intégration.

Nous nous sommes ensuite intéressés à la mise en valeur des triangles grâce à une coloration, tout d'abord par des gradients de couleurs, puis en donnant un sens à ces couleurs en étudiant l'inclinaison des triangles sur une surface en trois dimensions. Dans le but de rendre notre travail plus intéractif, nous avons réalisé une interface utilisateur simple d'utilisation et regroupant chaque point de notre travail.

Cependant, si nous avions disposé de plus de temps, nous aurions aimé pouvoir améliorer notre travail. Il aurait alors été possible de lisser les couleurs de l'inclinaison des triangles pour l'étude des surfaces en trois dimensions grâce à des convolutions.
Nous aurions également souhaité pouvoir améliorer la complexité de notre algorithme, notamment en implémentant l'algorithme de **Guibas** et **Stolfi**, une version de cette algorithme utilisant la méthode *diviser pour régner* et ayant une complexité en $O(n \times \log n)$ [3].

Auto-Évaluation

Forces. Notre implémentation de la triangulation de Delaunay est très modulaire et robuste. En effet, nous avons réalisé de nombreux modules, chacun d'entre eux spécialisé dans une tâche précise. L'ensemble de ces modules a d'ailleurs été testé par des tests unitaires et d'intégration.

Faiblesses. Nous avons passé beaucoup de temps à comprendre comment compiler nos fichiers sous Ocaml.
Nous avons également perdu du temps à chercher des erreurs dans notre code, erreurs se trouvant dans le calcul du déterminant des matrices et l'union des ensembles. Suite à cela, nous avons mis en place des tests unitaires et d'intégration plus poussés afin de nous assurer de la validité de nos modules.

Opportunités. Nous avons appris à utiliser l'outil **GIT** pour travailler en équipe, et notamment la notion de branche.
Ce projet nous a également permis de mieux assimiler le fonctionnement de la compilation en Ocaml, la compilation ayant été pendant un certain temps un problème pour nous.

Menaces. La gestion du temps est problématique car nous avons souvent été pris au dépourvu pour les dates de rendu.
De plus, l'ambition et la prévoyance peuvent poser problème. Nous avons pu apprendre qu'il était inutile de prévoir des fonctionnalités par avance pour le cas où il y en aurait besoin. Il nous faudra mieux respecter les méthodes **YAGNI** [4] et **KISS** [5].

3. Voir : `http://www.geom.uiuc.edu/~samuelp/del_project.html#algorithms`.
4. *You Aren't Gonna Need It.*
5. *Keep It Simple Stupid.*

A Bibliographie

— Wikipédia : Triangulation de Delaunay, `https://en.wikipedia.org/wiki/Delaunay_triangulation`
— Telecom-ParisTech : applications possibles de la triangulation de Delaunay, `http://www.tsi.telecom-paristech.fr/pages/enseignement/ressources/beti/delaunay/delaunay_imprimable.pdf`
— Documentation INRIA de la librairie `graphics`, `https://caml.inria.fr/pub/docs/oreilly-book/pdf/chap5.pdf`

B Annexes

Entrées : E : Ensemble de points disjoints et à l'intérieur d'un rectangle ABCD (formant une enveloppe convexe),
T : Triangles initiaux (ABC et ADC)

si *E est vide* **alors**
  retourner T
sinon
  $p \leftarrow$ un point de E
  $t \leftarrow$ triangles dont le cercle circonscrit contient p
  $c \leftarrow$ points de l'enveloppe convexe des triangles t
  $n \leftarrow$ triangles obtenus par les points c reliés au point p
  Retirer p de E
  Retirer t de T
  Ajouter n dans T
  Appeler Triangulation(E, T)
fin

Algorithme 1 : Triangulation(E, T)

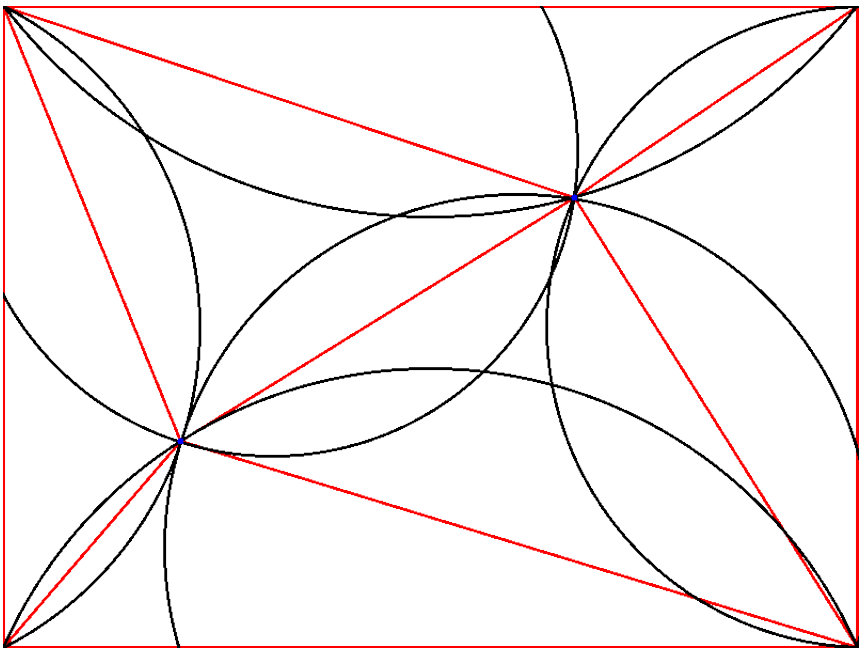

FIGURE 1 – Exemple d'une triangulation de Delaunay.

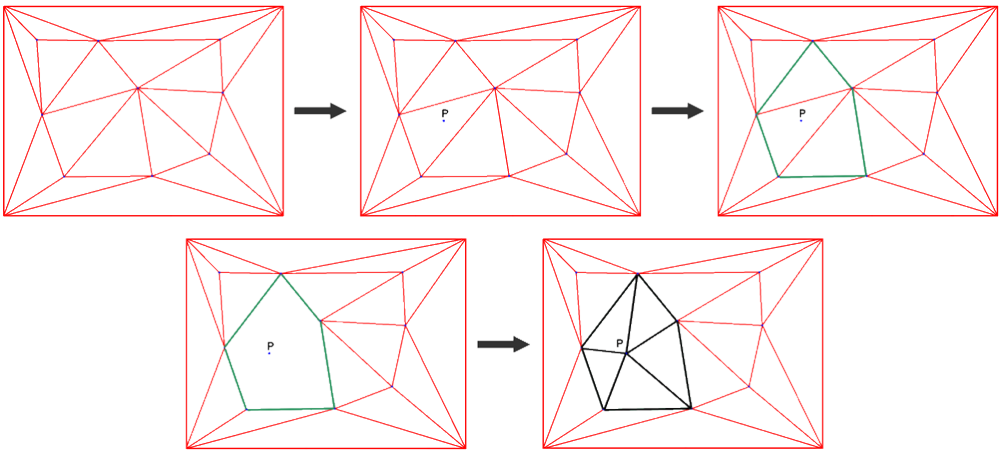

FIGURE 2 – Représentation d'une étape de l'algorithme sur un exemple.

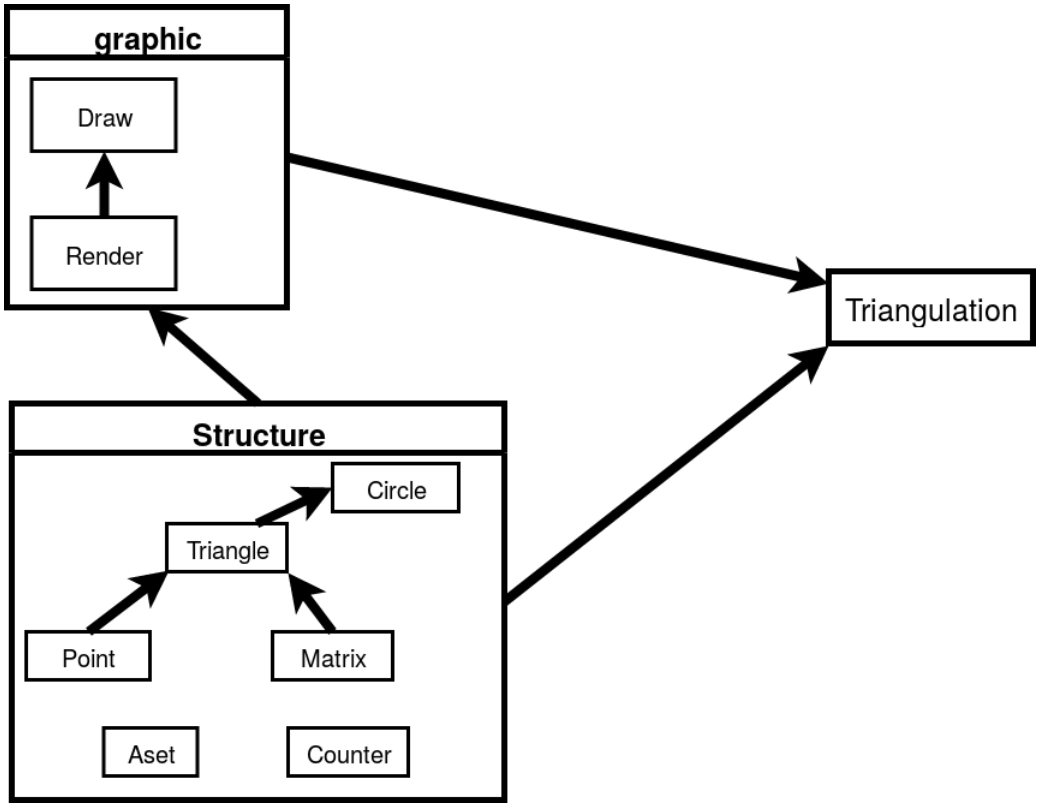

FIGURE 3 – Diagramme UML simplifié des modules.

```
type Point.point
val make : float -> float -> Point.point
type Triangle.triangle
val make : Point.point -> Point.point -> Point.point -> Triangle.triangle
```

FIGURE 4 – Des constructeurs génériques pour les types.

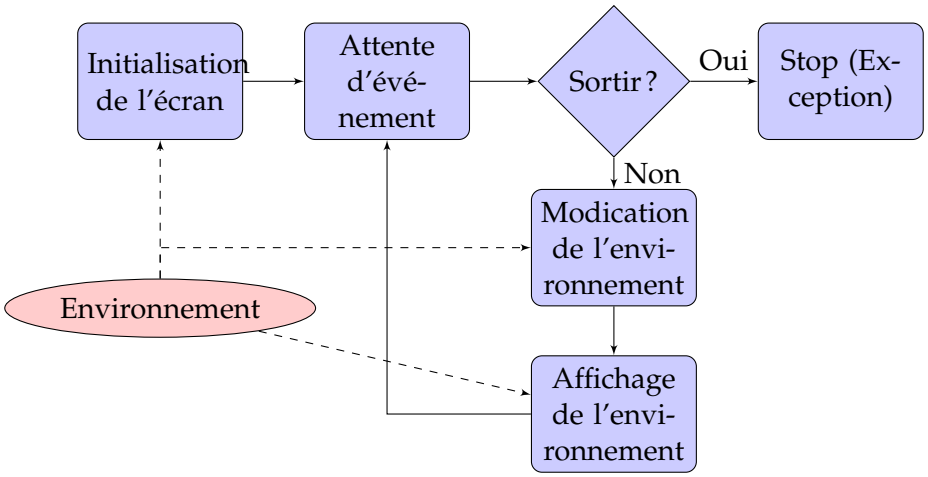

FIGURE 5 – Organigramme de la boucle de réaction de l'interface utilisateur.

```
type environnement = {
  window_width : int;
  window_height : int;
  add_point_function : ((Triangle.triangle) Aset.aset) -> Point.point ->
                       ((Triangle.triangle) Aset.aset);
  triangulation_function : ((Point.point) Aset.aset) -> int -> int ->
                           ((Triangle.triangle) Aset.aset);
  mutable nb_point : int;
  mutable point_set : ((Point.point) Aset.aset);
  mutable triangle_set : ((Triangle.triangle) Aset.aset);
  mutable show_menu : bool;
  mutable show_point : bool;
  mutable show_triangle : bool;
  mutable show_circle : bool;
  mutable show_filltriangle : bool;
  mutable gradient_type : int;
  mutable colors : ((float * float * float) * (float * float * float));
  mutable click : Point.point
};;
```

FIGURE 6 – Des constructeurs génériques pour les types.

FIGURE 7 – Gradient en x.

FIGURE 8 – Gradient en y.

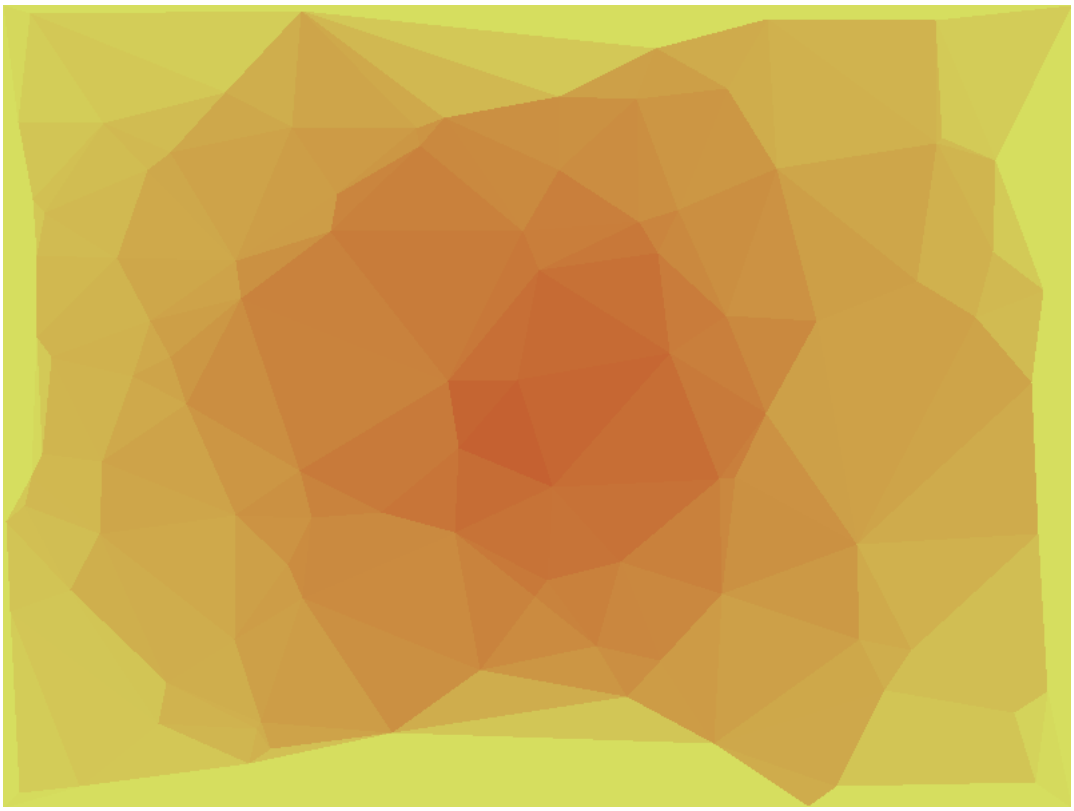

FIGURE 9 – Gradient circulaire.

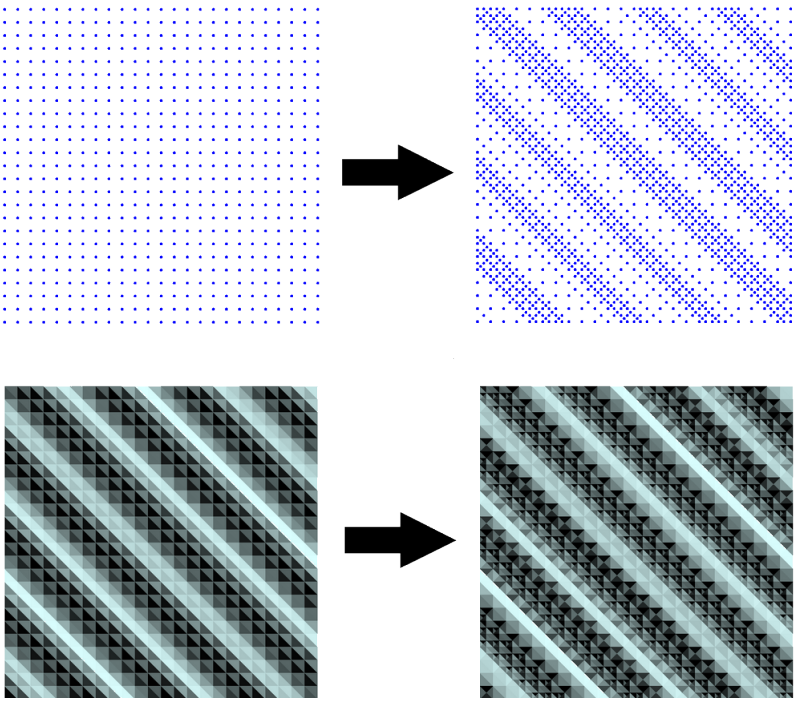

FIGURE 10 – Remplissage intelligent et coloreur en fonction de l'angle pour $(x, y) \longmapsto sin(x + y)$.

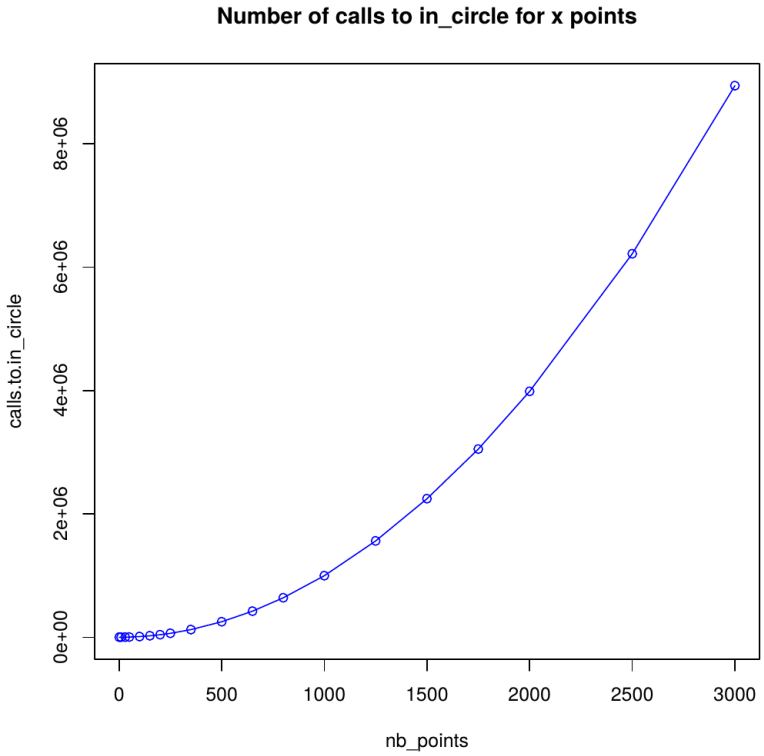

FIGURE 11 – Nombre d'appels à in_circle pour x points.

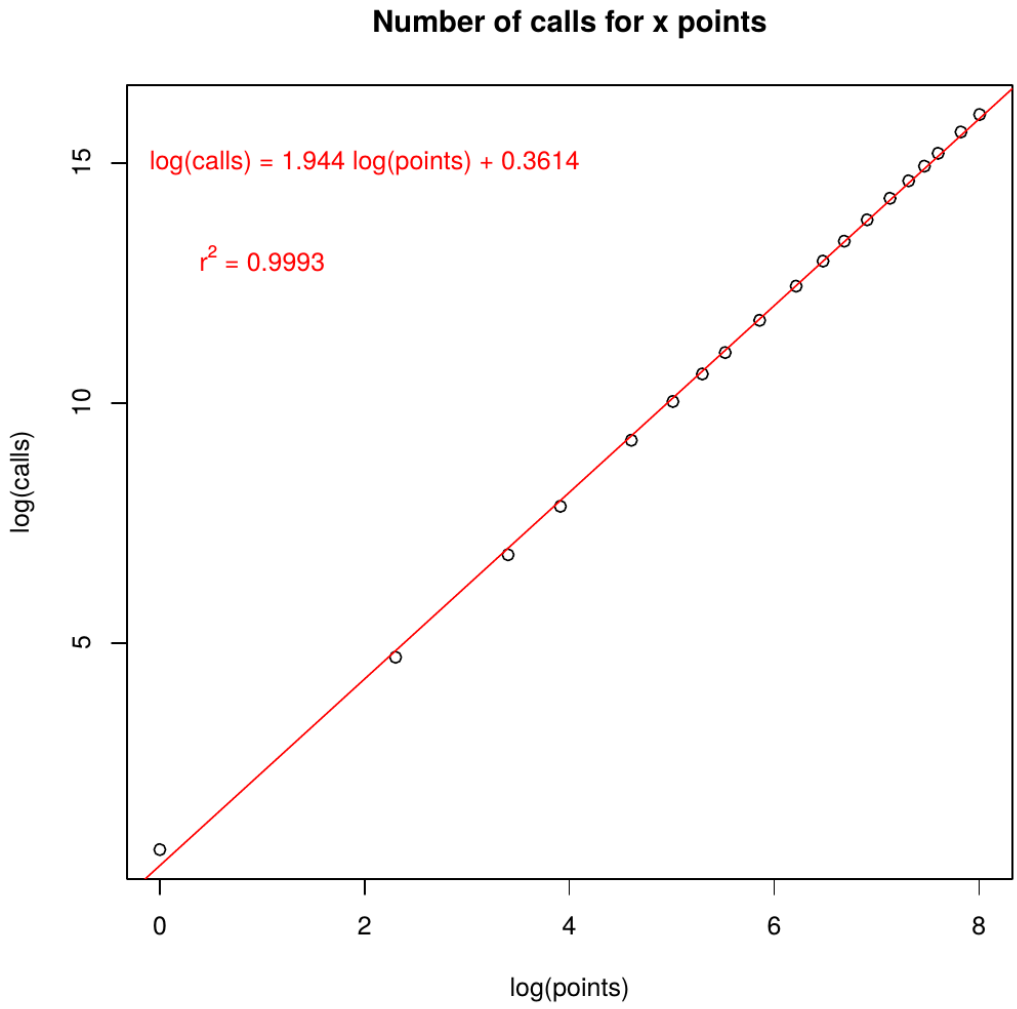

FIGURE 12 – Nombre d'appels à in_circle pour x points.