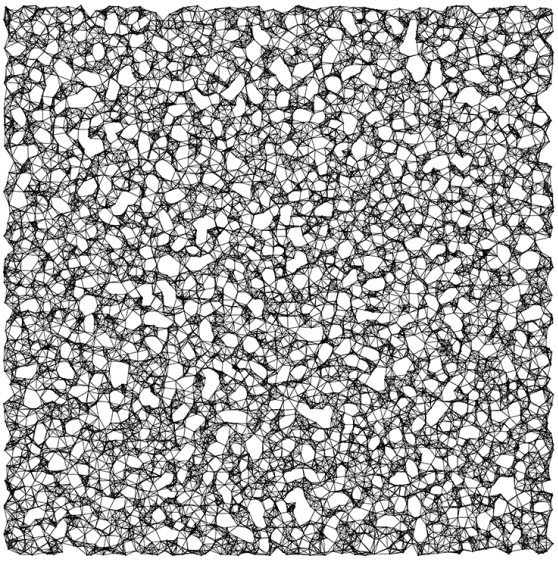
101.1 Programmation impérative
Prof. P.-A. Mudry
Semestre de printemps 2023-2024

Informatique et systèmes de communication ISC
Haute Ecole d'Ingénierie
Hochschule für Ingenieurwissenschaften



KNN graph – Inspired by Marcus Volg

Rapport de projet pour la filière ISC

Avec une mise en page avec `Typst`

P.-A. Mudry **Louis Lettry** **Pamela Delgado**

21 juin 2024

Table des matières

1 - Introduction

Écrire un rapport est un exercice autant **de fond que de forme**. Dans ce contexte, nous proposons dans ce document de quoi simplifier la rédaction de la forme sans avoir – à priori – d'avis sur le fond, ceci dans le contexte de la filière ISC[1].

Il convient tout d'abord pour présenter le contenu de se rendre compte que ce système de mise en page permet d'utiliser une forme de *markdown* comme entrée. Le *markdown* est une manière de formatter des fichiers textes afin de pouvoir les transformer avec un programme afin de les afficher dans différents formats, comme PDF ou encore sous forme de page web.

Le langage *markdown* utilise différents types de balises permettant de faire du **gras**, de *l'italique* ou encore du ***gras et de l'italique***. Il est également possible de faire des listes, des tableaux, des images, des liens hypertextes, des notes de bas de page, des équations mathématiques comme $x^2 = 3$, des blocs de code comme par exemple `def hello()` et encore bien d'autres choses.

Vous trouverez ici de la documentation sur la manière d'utiliser le langage `markdown` pour écrire des documents ici : https://www.markdownguide.org/basic-syntax/. Vous trouverez également une version spécifique sur l'écriture de documents en Typst ici https://typst.app/docs/guides/markdown-guide/.

En plus des choses simples montrées ci-dessus, le `markdown` simplifie la création de listes avec des nombres comme suit :

1. Un élément
2. Un autre élément de liste
3. Encore d'autres éléments si nécessaire

Des choses plus exotiques, comme mettre du **TODO** texte mis en évidence sont également possibles, tout comme les références à d'autres parties, comme dans le point 1.6.

1.1 - Insertion de code

Nous pouvons également avoir du `code brut directement en ligne` mais cela peut également être fait avec du code Scala comme par exemple dans `def foo(x: Int)`. Cela n'empêche pas d'avoir des blocs de code joliment mis en forme également. Ainsi, lorsque l'on souhaite avoir du code inséré dans une figure, on peut également utiliser le package `sourcecode` qui rajoute notamment les numéros de ligne. En complément avec une `figure`, il est possible d'avoir une *légende*, un numéro de figure ainsi que du code centré :

```scala
def foo(val a : Any) : Int = {
  a match :
    case a: Int  ⇒ 12
    case _ ⇒ 42
}
```

Listing 1 - Un tout petit listing en `Scala`

On peut si on le souhaite également avoir des blocs de code plus long si nécessaire, sur plusieurs pages :

```scala
object ImageProcessingApp_Animation extends App {
  val imageFile = "./res/grace_hopper.jpg"

  val org = new ImageGraphics(imageFile, "Original", -200, 0)
  val dest2 = new ImageGraphics(imageFile, "Threshold", 200, 0)

  var direction: Int = 1
```

[1] Voici d'ailleurs comment mettre une note de bas de page https://isc.hevs.ch

```
  var i = 1

  while (true) {
    if (i == 255 || i == 0)
      direction *= -1

    i = i + direction
    dest2.setPixelsBW(ImageFilters_Solution.threshold(org.getPixelsBW(), i))
  }
}
```

Listing 2 - Un autre exemple de code, plus long

1.1.1 - Insérer du code à partir d'un fichiers

Il est tout à fait possible de mettre du code qui provient d'un fichier comme ci-dessous :

```
class Cons[+A](hd: A, tl: => MyStream[A]) extends MyStream[A] {
  override def isEmpty: Boolean = false
  override val head: A = hd
  override lazy val tail: MyStream[A] = tl
  override def #::[B >: A](elem: B): MyStream[B] =
    new Cons[B](elem, this)

  override def ++[B >: A](anotherStream: => MyStream[B]): MyStream[B] =
    new Cons[B](head, tail ++ anotherStream)

  override def foreach(f: A => Unit): Unit = {
    f(head)
    tail.foreach(f)
  }
}
```

Listing 3 - Code included from the file `example.scala`

1.2 - Insertion d'images

Une image vaut souvent mieux que mille mots ! Il est possible d'ajouter des images, bien entendu. La syntaxe est relativement simple comme vous pouvez le voir dans l'exemple ci-dessous:

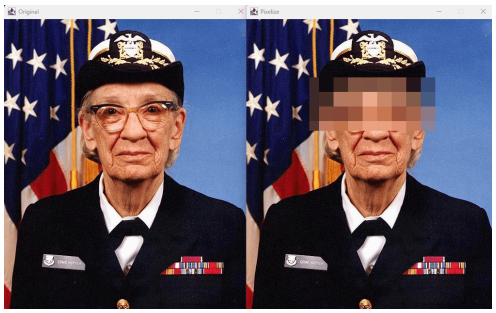


Figure 1 - Grace Hopper, informaticienne américaine

Pour le reste, voici un texte pour voir de quoi il retourne. Vous allez réaliser une fonction appelée *mean* qui va appliquer un filtre de moyenne à l'image. Ce filtre a pour but de flouter l'image et d'enlever ainsi ses aspérités. Le principe est le suivant : la valeur d'un pixel est remplacée par la moyenne des pixels se trouvant dans une zone carrée de 3 par 3 pixels autour du pixel. Si on veut calculer la nouvelle valeur du pixel situé à la position (x, y) selon la figure Figure 1, sa nouvelle valeur sera la moyenne des 9 valeurs affichées.

La dérivée doit se calculer selon les deux axes. Le calcul est très simple : la dérivée selon `x` du pixel situé en (x, y) vaut la valeur du pixel de droite $(x + 1, y)$ moins la valeur du pixel de gauche $(x - 1, y)$. Dans le cas de la figure, la dérivée selon x vaut $D_x = 234 - 255 = -21$.

De même, on peut calculer la dérivée selon y. Elle correspond au pixel du dessous $(x, y+1)$ moins le pixel $(x, y-1)$ du dessus. Dans le cas de la Figure 1, la dérivée selon y vaut $D_y = 230 - 127 = 103$.

La norme de la dérivée est calculée selon le théorème de Pythagore :

$$D = \sqrt{D_x^2 + D_y^2}$$

On peut également avoir des notations plus complexes :

$$\sum_{n=1}^{\infty} 2^{-n} = 1 \text{ ou encore } \int_{x=0}^{3} x^2 \, \mathrm{d}x$$

Stokes' theorem

Let Σ be a smooth oriented surface in $\mathbb{R}^3$ with boundary $\partial\Sigma \equiv \Gamma$. If a vector field $\boldsymbol{F}(x, y, z) = \big(F_x(x, y, z), F_y(x, y, z), F_z(x, y, z)\big)$ is defined and has continuous first order partial derivatives in a region containing Σ, then

$$\iint_{\Sigma} (\boldsymbol{\nabla} \times \boldsymbol{F}) \cdot \boldsymbol{\Sigma} = \oint_{\partial\Sigma} \boldsymbol{F} \cdot \mathrm{d}\boldsymbol{\Gamma}$$

1.3 - Des tables

Il est possible d'insérer des tables simples :

Monday	11.5	13.0	4.0
Tuesday	8.0	14.5	5.0
Wednesday	9.0	18.5	13.0

Table 1 - Une table simple

Des tables plus compliquées sont également possible. La page https://typst.app/docs/guides/table-guide/ donne d'ailleurs de bonnes informations.

Technique	Advantage	Drawback
Diegetic	Immersive	May be contrived
Extradiegetic	Breaks immersion	Obstrusive
Omitted	Fosters engagement	May fracture audience

Table 2 - Une table plus complexe

1.4 - Citer ses sources

Il est important de citer les sources que l'on utilise. Par exemple, les deux travaux [1], [2] et [3] sont deux papiers très intéressants à lire et dont les références complètes se trouvent dans la bibliographie à la fin de ce document. Il est également d'utiliser des acronymes comme par exemple [Universal Serial Bus (USB)]

1.5 - Le filtre de Sobel

Une autre méthode pour extraire les contours à l'intérieur d'une image est d'utiliser l'algorithme de Sobel Cette méthode est très similaire à celle de la dérivée, mais un peu plus compliquée et donne de meilleurs résultats.

Pour l'exemple, la valeur du filtre de Sobel selon *x* vaudrait :

$$S_x = 100 + 2 \cdot 234 + 84 - 128 - 2 \cdot 255 - 123 = -109$$

De même la valeur du filtre de Sobel selon *y* vaudrait:

$$S_y = 123 + 2 \cdot 230 + 84 - 128 - 2 \cdot 127 - 100$$

Comme auparavant, la norme du filtre de Sobel se calcule selon Pythagore et vaut pour cet exemple :

$$S = \sqrt{S_x^2 + S_y^2} = \sqrt{109^2 + 185^2} = 214.47$$

1.6 - Problématique

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat.

1.7 - Plan du travail

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere.

2 - Conclusion

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut postea variari voluptas distinguique possit, augeri amplificarique non possit. At etiam Athenis, ut e patre audiebam facete et urbane Stoicos irridente, statua est in quo a nobis philosophia defensa et collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et aut officiis debitis aut rerum necessitatibus saepe eveniet, ut et voluptates repudiandae sint et molestiae non recusandae. Itaque earum rerum defuturum, quas natura non depravata desiderat. Et quem ad me accedis, saluto: 'chaere,' inquam, 'Tite!' lictores, turma omnis chorusque: 'chaere, Tite!' hinc hostis mi Albucius, hinc inimicus. Sed iure Mucius. Ego autem mirari satis non queo unde hoc sit tam insolens domesticarum rerum fastidium. Non est omnino hic docendi locus; sed ita prorsus existimo, neque eum Torquatum, qui hoc primus cognomen invenerit, aut torquem illum hosti detraxisse, ut aliquam ex eo est consecutus? – Laudem et caritatem, quae sunt vitae sine metu degendae praesidia firmissima. – Filium morte multavit. – Si sine causa, nollem me ab eo delectari, quod ista Platonis, Aristoteli, Theophrasti orationis ornamenta neglexerit. Nam illud quidem physici, credere aliquid esse minimum, quod profecto numquam putavisset, si a Polyaeno, familiari suo, geometrica discere maluisset quam illum etiam ipsum dedocere. Sol Democrito magnus videtur, quippe homini erudito in geometriaque perfecto, huic pedalis fortasse; tantum enim esse omnino in nostris poetis aut inertissimae segnitiae est aut fastidii delicatissimi. Mihi quidem videtur, inermis ac nudus est. Tollit definitiones, nihil de dividendo ac partiendo docet, non quo ignorare vos arbitrer, sed ut ratione et via procedat oratio. Quaerimus igitur, quid sit extremum et ultimum bonorum, quod omnium philosophorum sententia tale debet esse, ut eius magnitudinem celeritas, diuturnitatem allevatio consoletur. Ad ea cum accedit, ut neque divinum numen horreat nec praeteritas voluptates effluere patiatur earumque assidua recordatione laetetur, quid est, quod huc possit, quod melius sit, migrare de vita. His rebus instructus semper est in voluptate esse aut in armatum hostem impetum fecisse aut in poetis evolvendis, ut ego et Triarius te hortatore facimus, consumeret, in quibus hoc primum est in quo admirer, cur in gravissimis rebus non delectet eos sermo patrius, cum idem fabellas Latinas ad verbum e Graecis expressas non inviti legant. Quis enim tam inimicus paene nomini Romano est, qui Ennii Medeam aut Antiopam Pacuvii spernat aut reiciat, quod se isdem Euripidis fabulis delectari dicat, Latinas litteras oderit? Synephebos ego, inquit, potius Caecilii aut Andriam Terentii quam utramque Menandri legam? A quibus tantum dissentio, ut, cum Sophocles vel optime scripserit Electram, tamen male conversam Atilii mihi legendam putem, de quo Lucilius: 'ferreum scriptorem', verum, opinor, scriptorem tamen, ut legendus sit. Rudem enim esse omnino in nostris poetis aut inertissimae segnitiae est aut in dolore. Omnis autem privatione doloris putat Epicurus.

Bibliographie

[1] P.-A. Mudry et G. Tempesti, « Self-Scaling Stream Processing: A Bio-Inspired Approach to Resource Allocation through Dynamic Task Replication », in *2009 NASA/ESA Conference on Adaptive Hardware and Systems*, 2009, p. 353-360. doi: 10.1109/AHS.2009.25.

[2] P.-A. Mudry, G. Zufferey, et G. Tempesti, « A hybrid genetic algorithm for constrained hardware-software partitioning », in *2006 IEEE Design and Diagnostics of Electronic Circuits and systems*, 2006, p. 1-6. doi: 10.1109/DDECS.2006.1649561.

[3] P.-A. Mudry, *A hardware-software codesign framework for cellular computing*. Lausanne: EPFL, 2009, p. 287-288. doi: 10.5075/epfl-thesis-4354.

Annexes

Table des acronymes

DVI: Digital Visual Interface

FPGA: Field Programmable Gates Array

HDMI: High Definition Multimedia Inteface

LCD: Liquid Cristal Display

LED: Light Emitting Diodes

PMod: Peripheral Module

USB: Universal Serial Bus

VGA: Virutal Graphics Array

Table des figures

Table des listings

Code annexé

```scala
class Cons[+A](hd: A, tl: => MyStream[A]) extends MyStream[A] {
  override def isEmpty: Boolean = false
  override val head: A = hd
  override lazy val tail: MyStream[A] = tl
  override def #::[B >: A](elem: B): MyStream[B] =
    new Cons[B](elem, this)

  override def ++[B >: A](anotherStream: => MyStream[B]): MyStream[B] =
    new Cons[B](head, tail ++ anotherStream)

  override def foreach(f: A => Unit): Unit = {
    f(head)
    tail.foreach(f)
  }
}
```

Listing 4 - Code included from the file example.scala

```scala
class Cons[+A](hd: A, tl: => MyStream[A]) extends MyStream[A] {
  override def isEmpty: Boolean = false
  override val head: A = hd
  override lazy val tail: MyStream[A] = tl
  override def #::[B >: A](elem: B): MyStream[B] =
    new Cons[B](elem, this)

  override def ++[B >: A](anotherStream: => MyStream[B]): MyStream[B] =
    new Cons[B](head, tail ++ anotherStream)

  override def foreach(f: A => Unit): Unit = {
    f(head)
    tail.foreach(f)
  }
}
```

Listing 5 - Code included from the file example.scala

```scala
class Cons[+A](hd: A, tl: => MyStream[A]) extends MyStream[A] {
  override def isEmpty: Boolean = false
  override val head: A = hd
  override lazy val tail: MyStream[A] = tl
  override def #::[B >: A](elem: B): MyStream[B] =
    new Cons[B](elem, this)

  override def ++[B >: A](anotherStream: => MyStream[B]): MyStream[B] =
    new Cons[B](head, tail ++ anotherStream)

  override def foreach(f: A => Unit): Unit = {
    f(head)
    tail.foreach(f)
  }
}
```

Listing 6 - Code included from the file example.scala

```scala
class Cons[+A](hd: A, tl: => MyStream[A]) extends MyStream[A] {
  override def isEmpty: Boolean = false
  override val head: A = hd
  override lazy val tail: MyStream[A] = tl
  override def #::[B >: A](elem: B): MyStream[B] =
    new Cons[B](elem, this)

  override def ++[B >: A](anotherStream: => MyStream[B]): MyStream[B] =
```

```
    new Cons[B](head, tail ++ anotherStream)

  override def foreach(f: A ⇒ Unit): Unit = {
    f(head)
    tail.foreach(f)
  }
}
```

Listing 7 - Code included from the file example.scala

```
class Cons[+A](hd: A, tl: ⇒ MyStream[A]) extends MyStream[A] {
  override def isEmpty: Boolean = false
  override val head: A = hd
  override lazy val tail: MyStream[A] = tl
  override def #::[B >: A](elem: B): MyStream[B] =
    new Cons[B](elem, this)

  override def ++[B >: A](anotherStream: ⇒ MyStream[B]): MyStream[B] =
    new Cons[B](head, tail ++ anotherStream)

  override def foreach(f: A ⇒ Unit): Unit = {
    f(head)
    tail.foreach(f)
  }
}
```

Listing 8 - Code included from the file example.scala