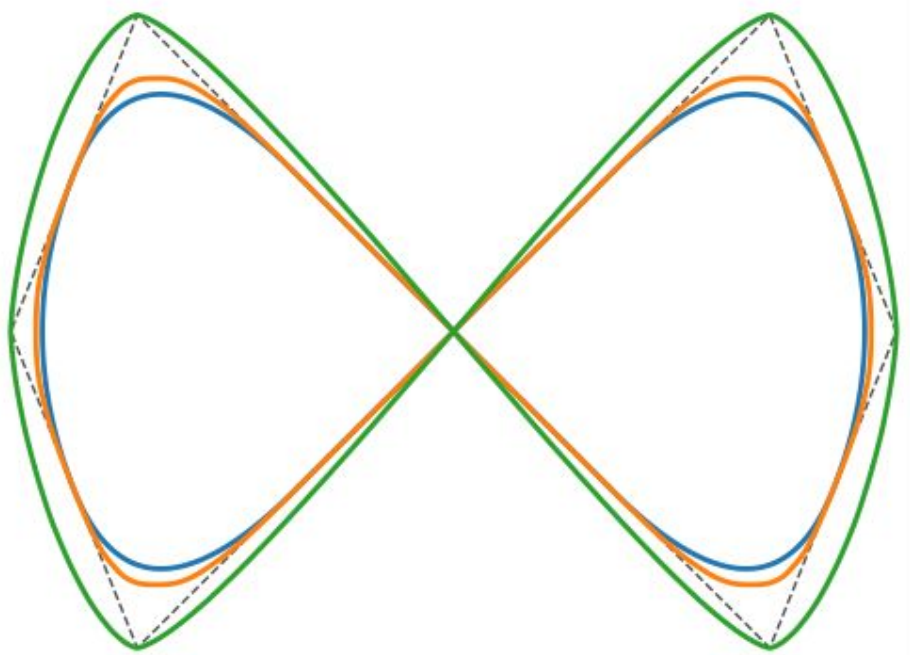
RAPPORT GÉOMÉTRIE NUMÉRIQUE

T.P. 4 - Courbes de subdivision



Calvin Massonnet

01/03/2021 - 07/03/2021

Université Grenoble Alpes - Master Informatique

Algorithme de Chaikin

L'algorithme de Chaikin permet, à partir d'un polygone de contrôle, d'obtenir un nouveau polygone de contrôle plus lisse que l'initial à l'aide d'une technique de subdivision. Cette technique consiste à séparer chaque segment en quatre, puis de relier le premier et le dernier points de séparation ensembles, ainsi qu'avec, respectivement, le dernier point de séparation du segment précédent et le premier point de séparation du segment suivant. Cet algorithme est également nommé *Corner cutting* pour l'impression qu'elle donne à couper les coins des polygones de contrôle. Il est à utiliser dans une boucle en passant en argument la sortie du traitement précédent. Le dernier polygone de contrôle, en sortie de boucle, est la courbe finale. La figure 1 présente une version simplifiée dans laquelle les segments sont coupés en quatre parts égales, tandis que la figure 2 en est une version généralisée dans laquelle l'utilisateur choisit la position des points de séparation.

```
def Chaikin(X0):
      # number of points
      n = X0.shape[0]

      # upsample
      X1 = np.zeros([2*n, 2])

      # Chaikin's subdivision scheme
      for i in range(n):
            X1[i*2] = (3.0/4.0) * X0[i] + (1.0/4.0) * X0[(i+1) % n]
            X1[i*2+1] = (1.0/4.0) * X0[i] + (3.0/4.0) * X0[(i+1) % n]

      return X1
```

Figure 1 : algorithme de Chaikin simplifié

```
def CornerCutting(X0, a, b):
      # number of points
      n = X0.shape[0]

      # upsample
      X1 = np.zeros([2*n, 2])

      # Corner cutting scheme
      for i in range(n):
            X1[i*2] = (1-a) * X0[i] + a * X0[(i+1) % n]
            X1[i*2+1] = (1-b) * X0[i] + b * X0[(i+1) % n]

      return X1
```

Figure 2 : algorithme de Chaikin généralisé (Corner cutting)

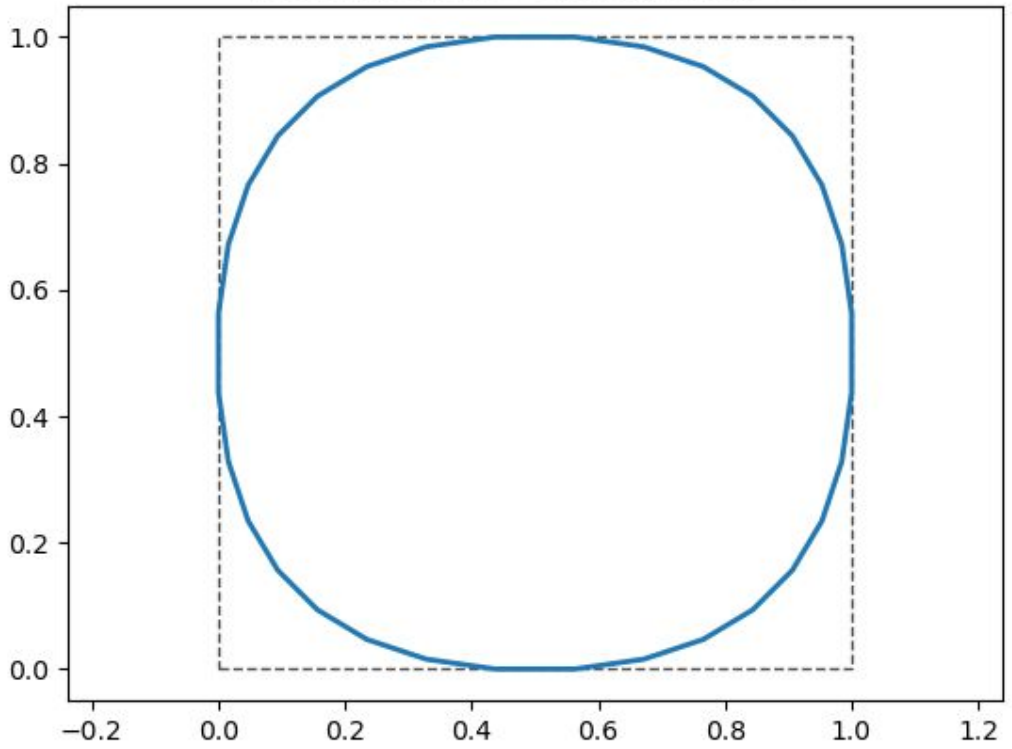


Figure 3 : “simple” de profondeur 3 avec l’algorithme de Chaikin (a = 0.25; b = 0.75)

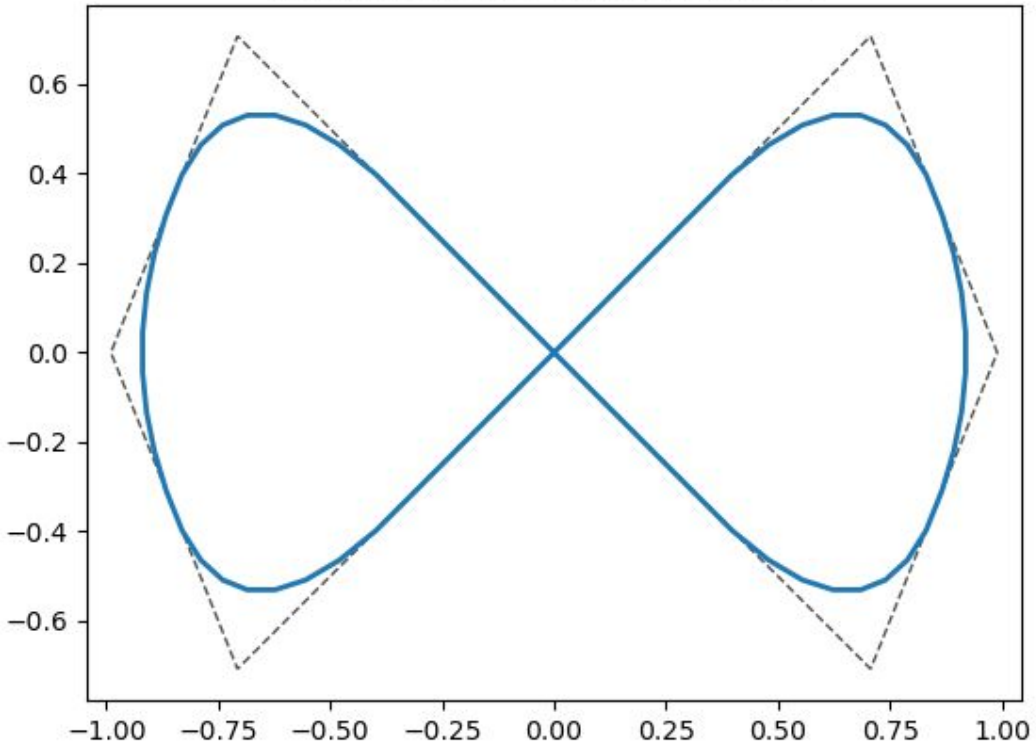


Figure 4 : “infinity” de profondeur 3 avec l’algorithme de Chaikin (a = 0.25; b = 0.75)

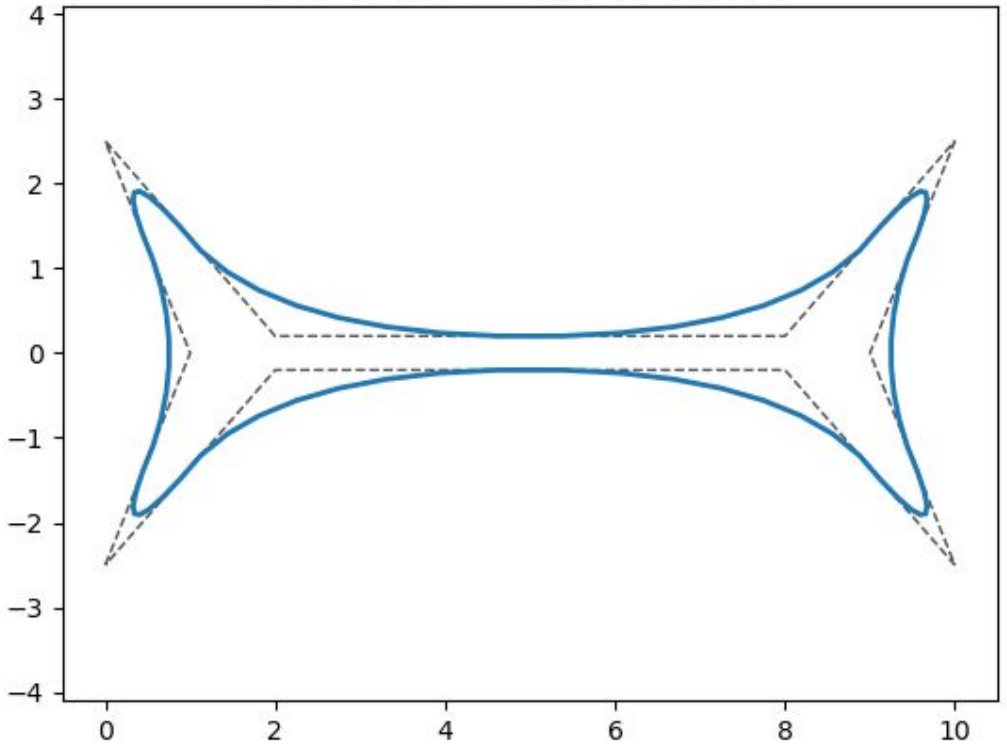


Figure 5 : “bone” de profondeur 3 avec l’algorithme de Chaikin (a = 0.25; b = 0.75)

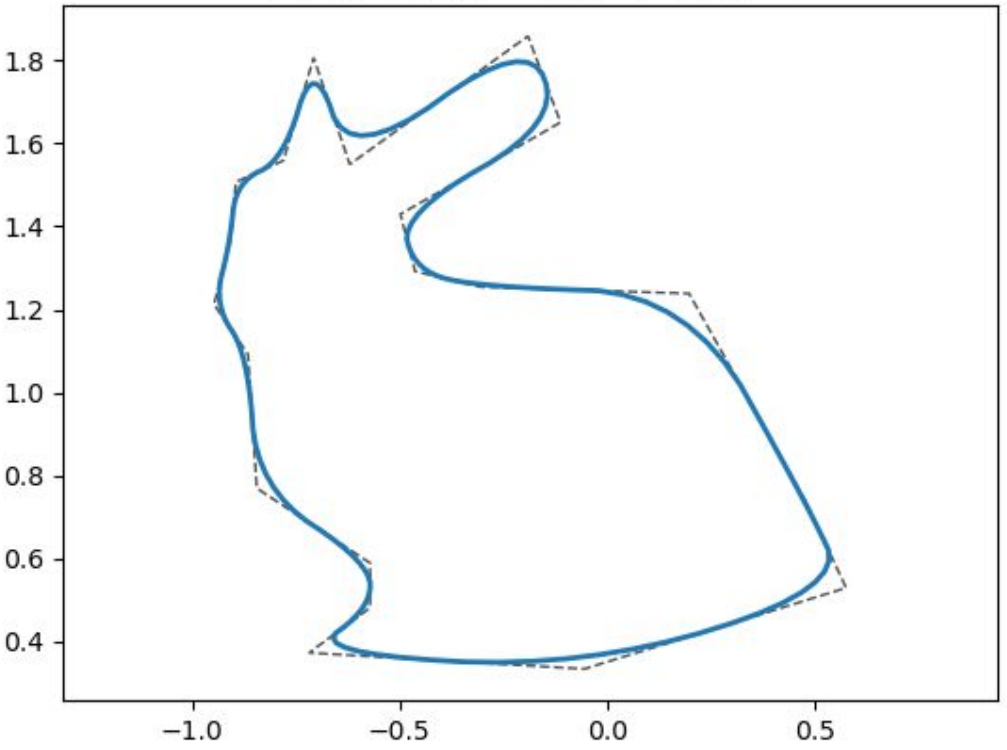


Figure 6 : “bunny” de profondeur 3 avec l’algorithme de Chaikin (a = 0.25; b = 0.75)

La liberté est donnée à l'utilisateur de choisir la position des points de contrôle pour plier la courbe à ses souhaits. Il faut cependant que la somme des paramètres reste égale à 1. Il est également possible, par erreur de manipulation par exemple, d'obtenir une courbe avec une allure inattendue, tel que peuvent le prouver les figures 8, 9 et 10.

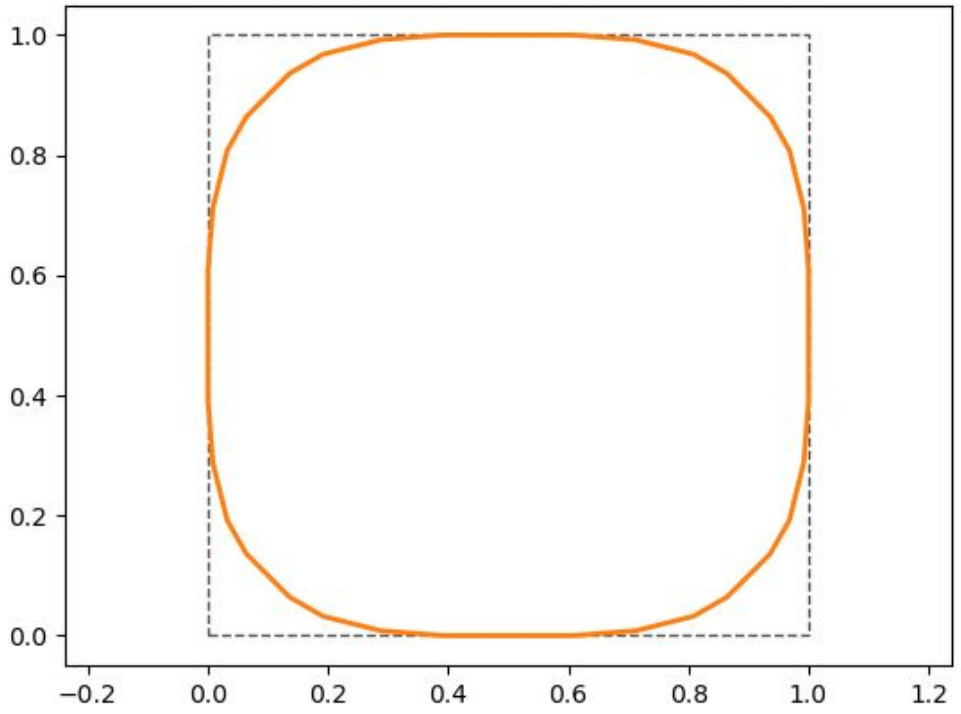


Figure 7 : "simple" de profondeur 3 avec l'algorithme de Corner cutting (a = 0.2; b = 0.8)

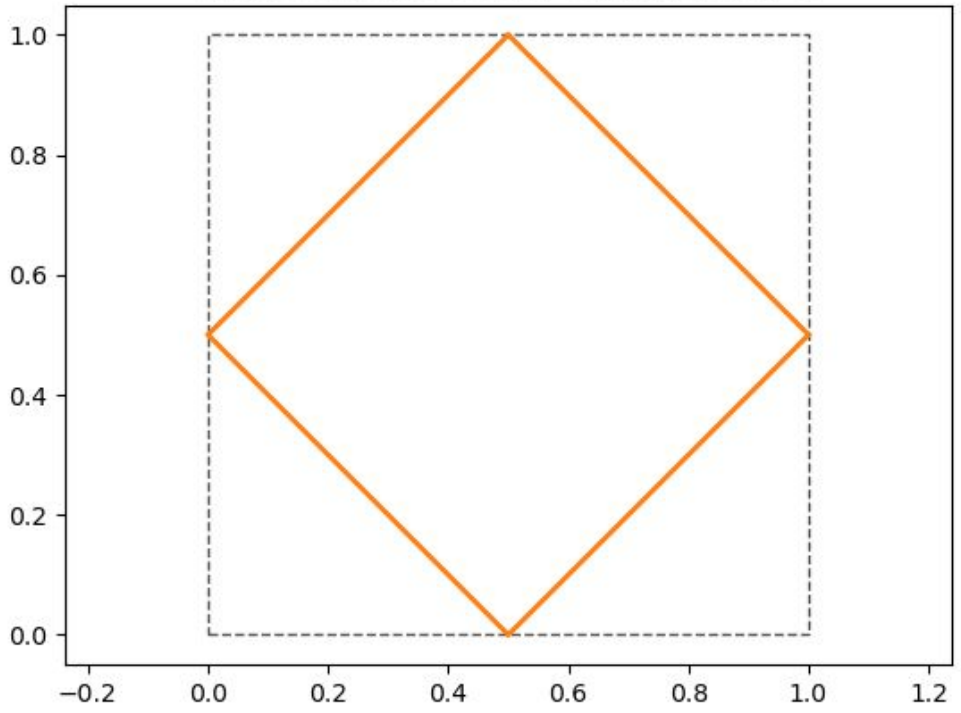


Figure 8 : "simple" de profondeur 3 avec l'algorithme de Corner cutting (a = 0.5; b = 0.5)

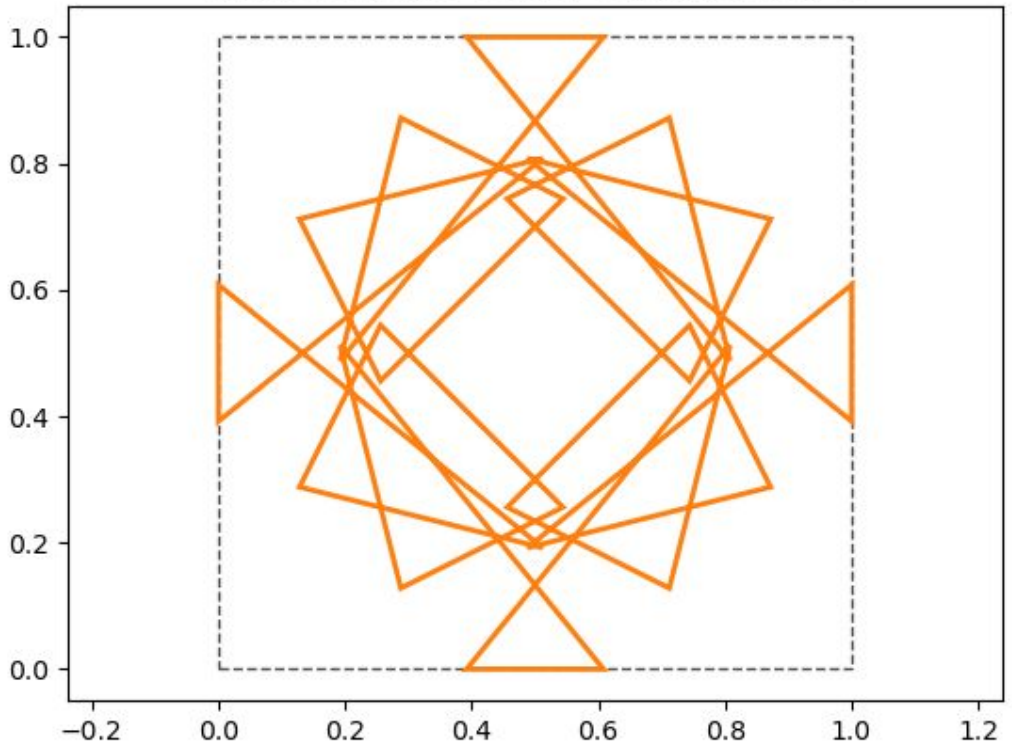


Figure 9 : “simple” de profondeur 3 avec l’algorithme de Corner cutting (a = 0.8; b = 0.2)

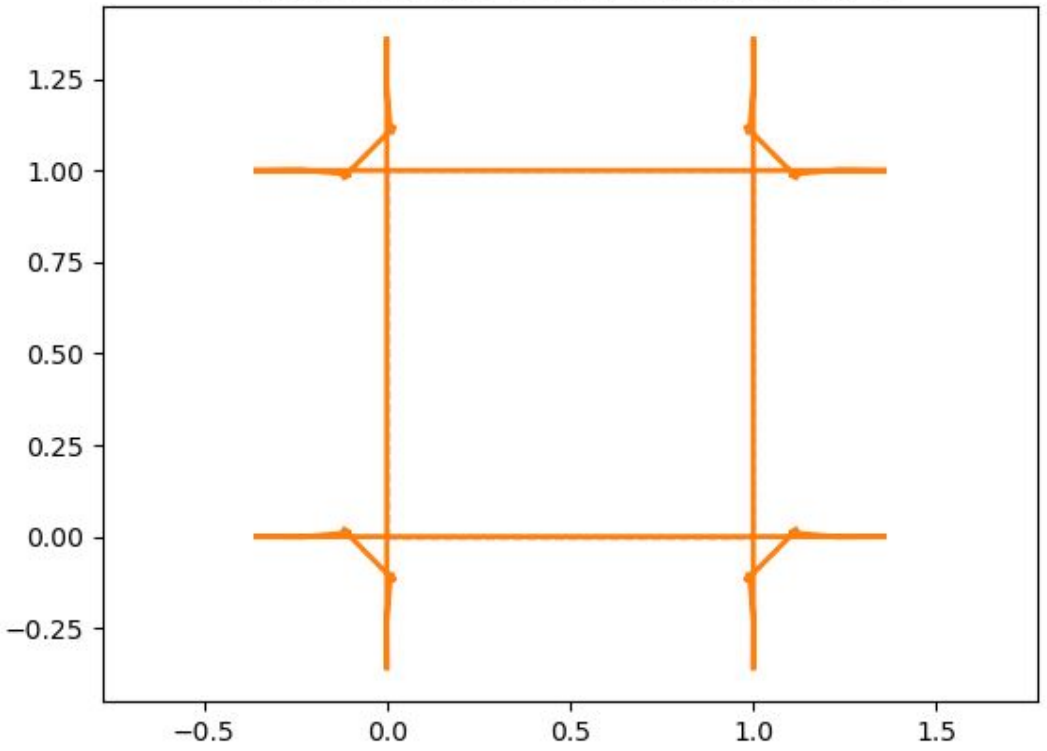


Figure 10 : “simple” de profondeur 3 avec l’algorithme de Corner cutting (a = -0.1; b = 1.1)

Une amélioration peut être apportée aux différents algorithmes de cet exercice afin de produire des courbes non-fermées. Il suffit de calculer le nombre de sommets du polygone de contrôle suivant comme suit : $2 * (n - 2) + 2$, n étant le nombre de sommets du polygone d'entrée. En portant une attention particulière aux extrémités des courbes des figures 11 et 12, il est possible de remarquer que ces courbes ne commencent, ni ne terminent, sur un point de contrôle du polygone de contrôle initial (polygone en pointillé).

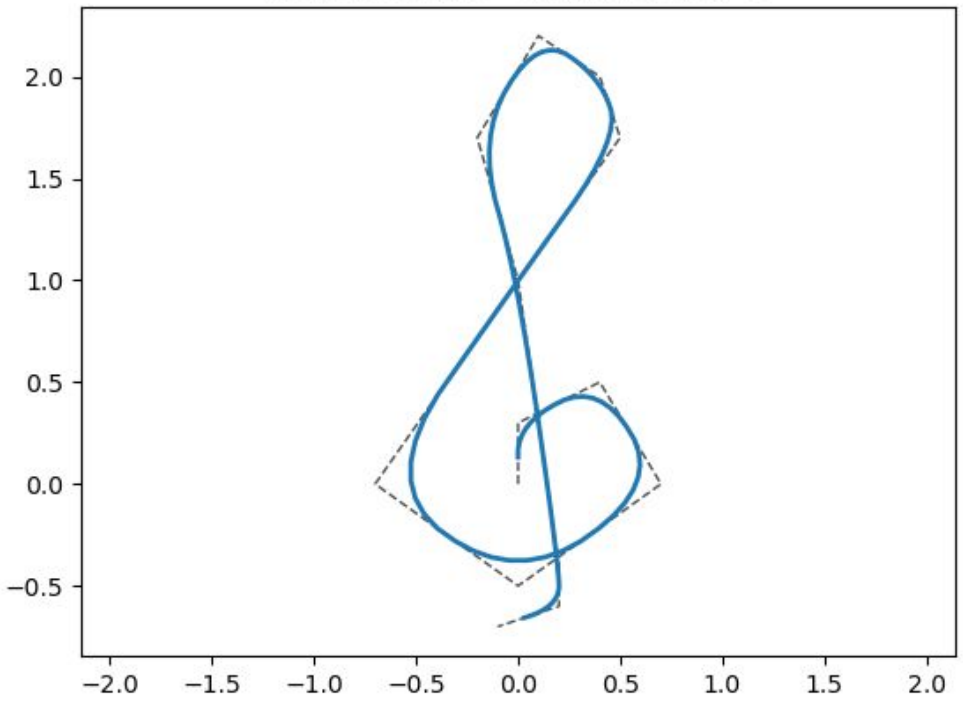


Figure 11 : "treble" de profondeur 3 avec l'algorithme de Chaikin (a = 0.25; b = 0.75)

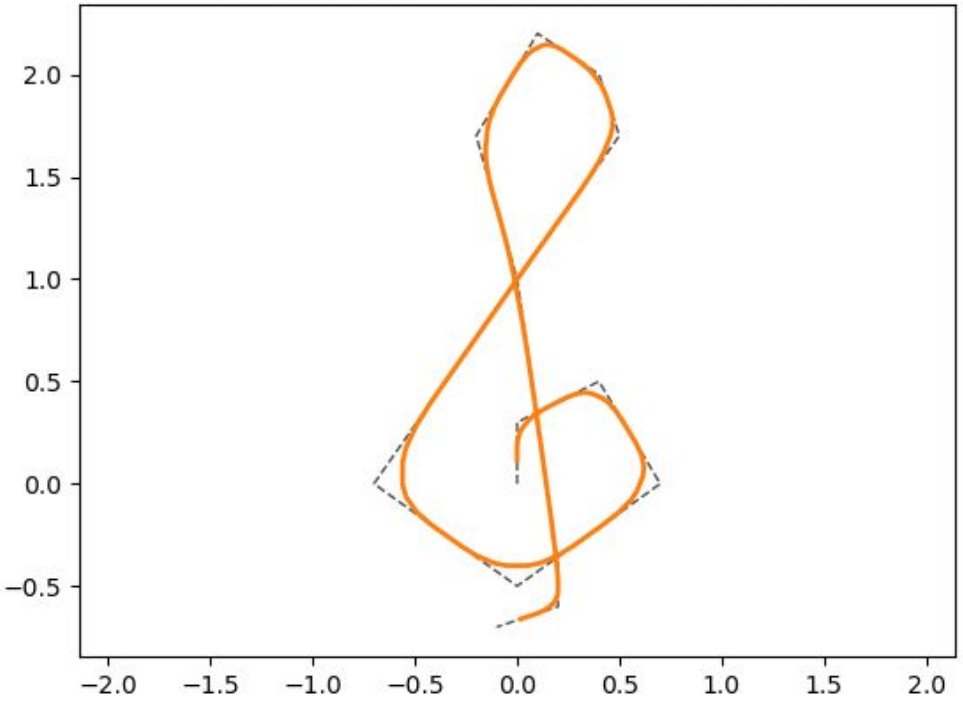


Figure 12 : "treble" de profondeur 3 avec l'algorithme de Corner cutting (a = 0.2; b = 0.8)

Algorithme de Four-point

La figure 13 présente l'algorithme de *Four-point*, également à utiliser dans une boucle. Celui-ci garde les points du niveau précédent et lisse la courbe en la faisant passer par les nouveaux points. Tous les points calculés et dessinés font donc partie de la courbe finale, ainsi que durant toute sa construction. Le paramètre w est le poids de construction de la courbe et décide de l'écart entre le nouveau point et le centre du segment au-dessus duquel il se situera. Comme pour l'algorithme de Chaikin, l'utilisateur a la responsabilité du choix sur le paramètre du poids et se doit de faire attention à sa valeur, comme le démontrent les figures 19 à 24.

```
def FourPoint(X0, w):

      # number of points
      n = X0.shape[0]

      # upsample
      X1 = np.zeros([2*n, 2])

      # Four-point scheme
      for i in range(n):
            X1[i*2] = X0[i]
            X1[i*2+1] = (-w * X0[(i-1) % n] + (w+0.5) * X0[i]
                      + (w+0.5) * X0[(i+1) % n] - w * X0[(i+2) % n])

      return X1
```

Figure 13 : algorithme de Four-point

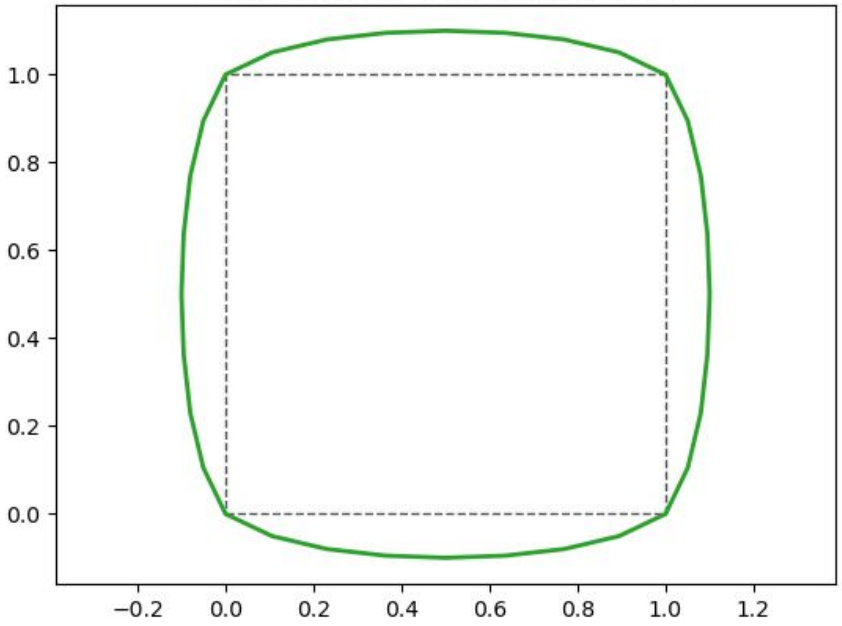


Figure 14 : "simple" de densité 3 avec l'algorithme de Four-point (w = 0.05)

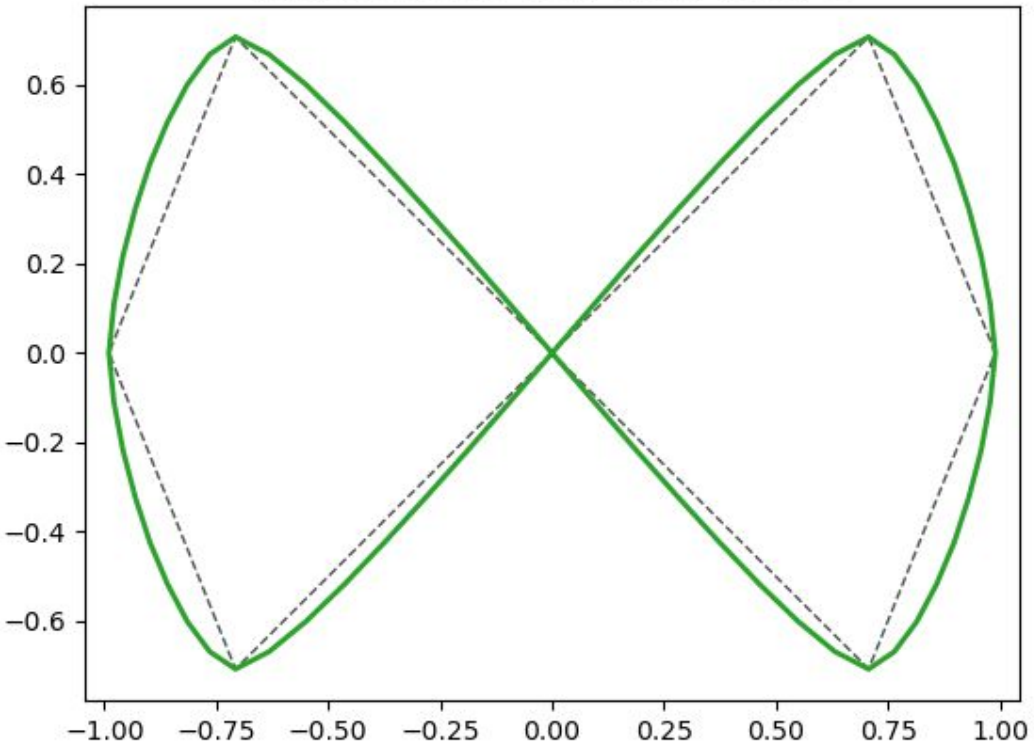


Figure 15 : “infinity” de densité 3 avec l’algorithme de Four-point (w = 0.05)

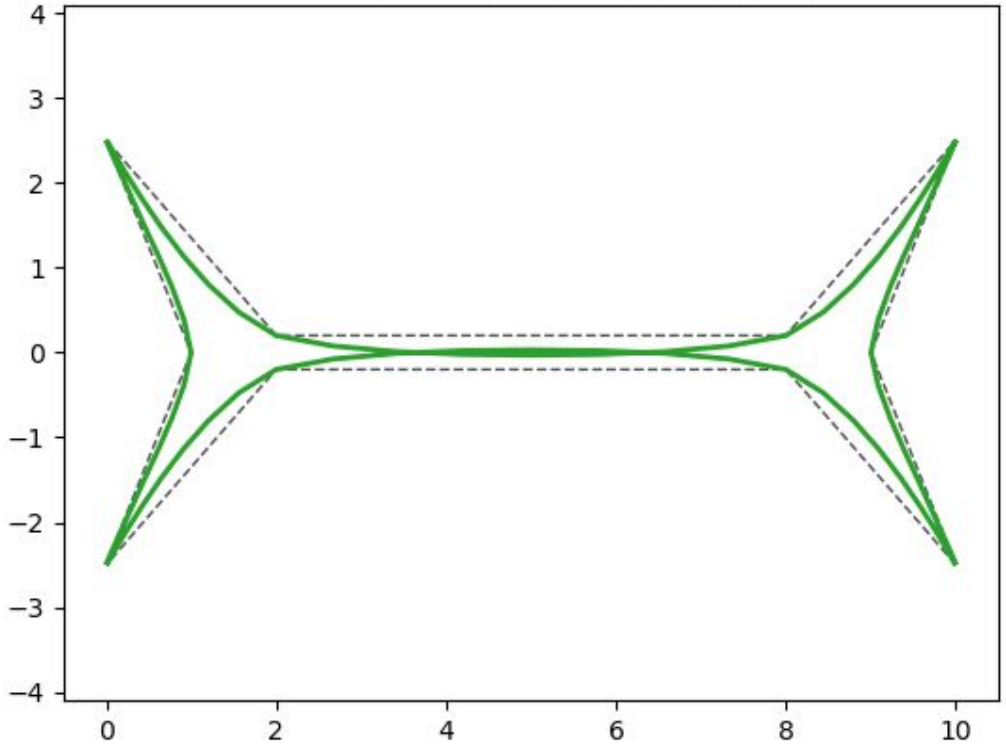


Figure 16 : “bone” de densité 3 avec l’algorithme de Four-point (w = 0.05)

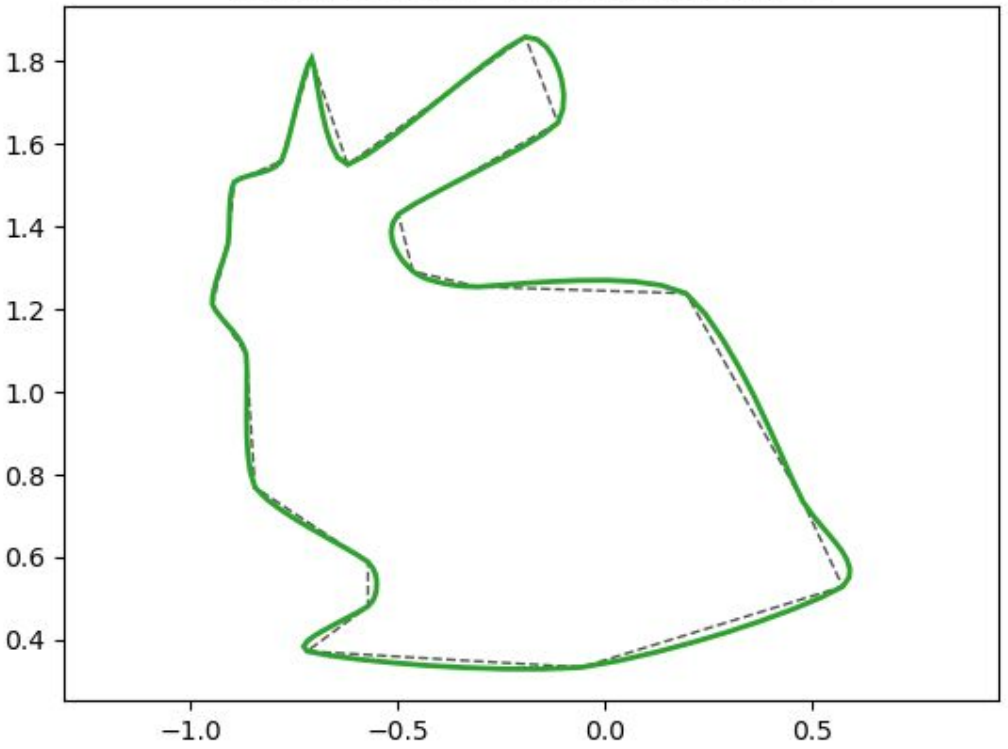


Figure 17 : “bunny” de densité 3 avec l’algorithme de Four-point (w = 0.05)

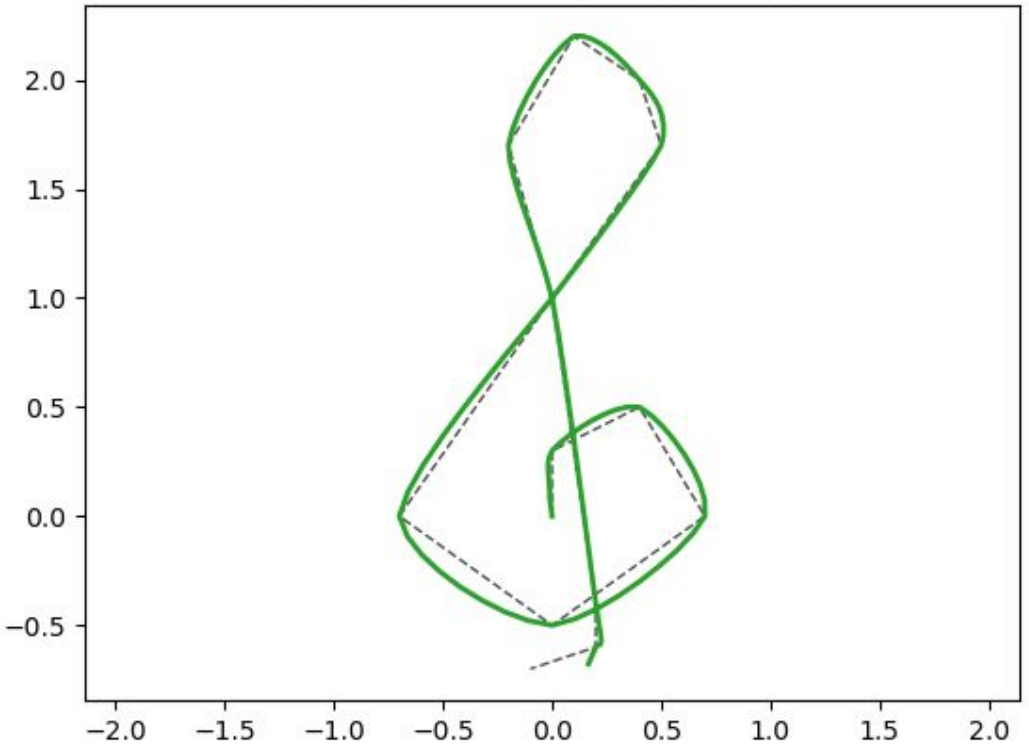


Figure 18 : “treble” de densité 3 avec l’algorithme de Four-point (w = 0.05)

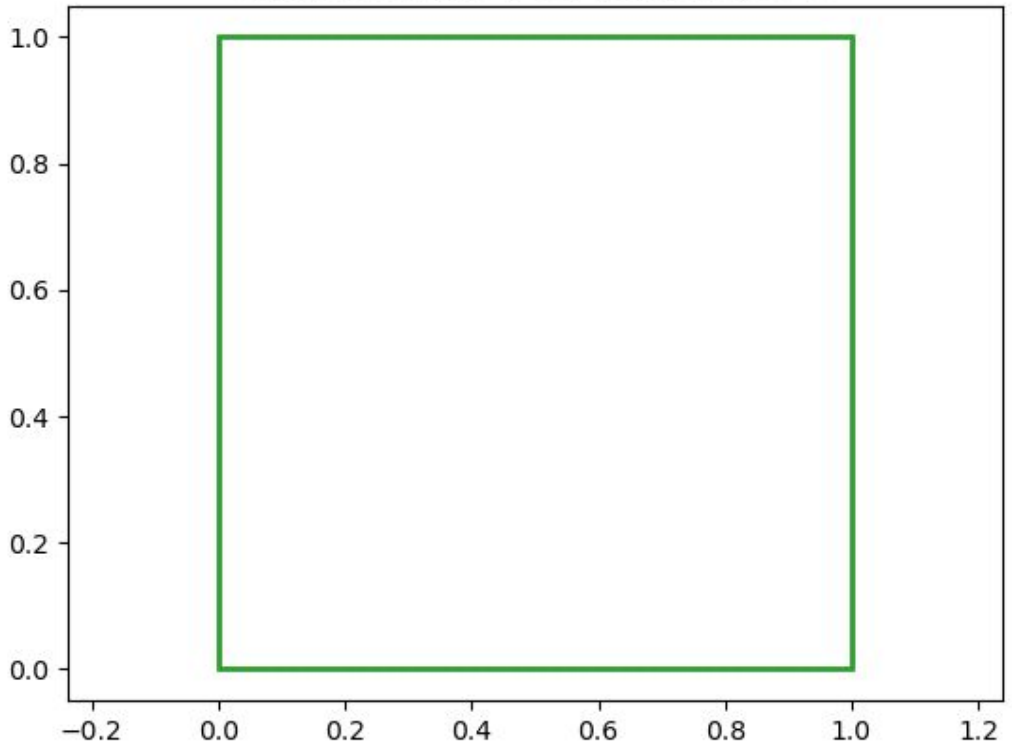


Figure 19 : “simple” de densité 3 avec l’algorithme de Four-point (w = 0.0)

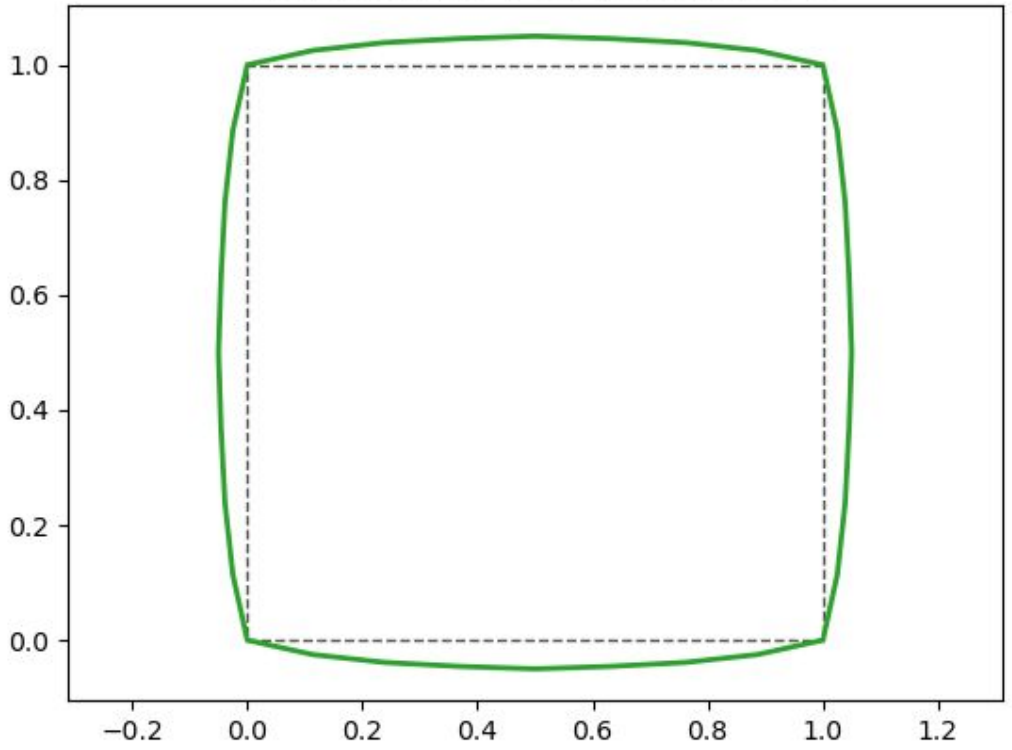


Figure 20 : “simple” de densité 3 avec l’algorithme de Four-point (w = 0.025)

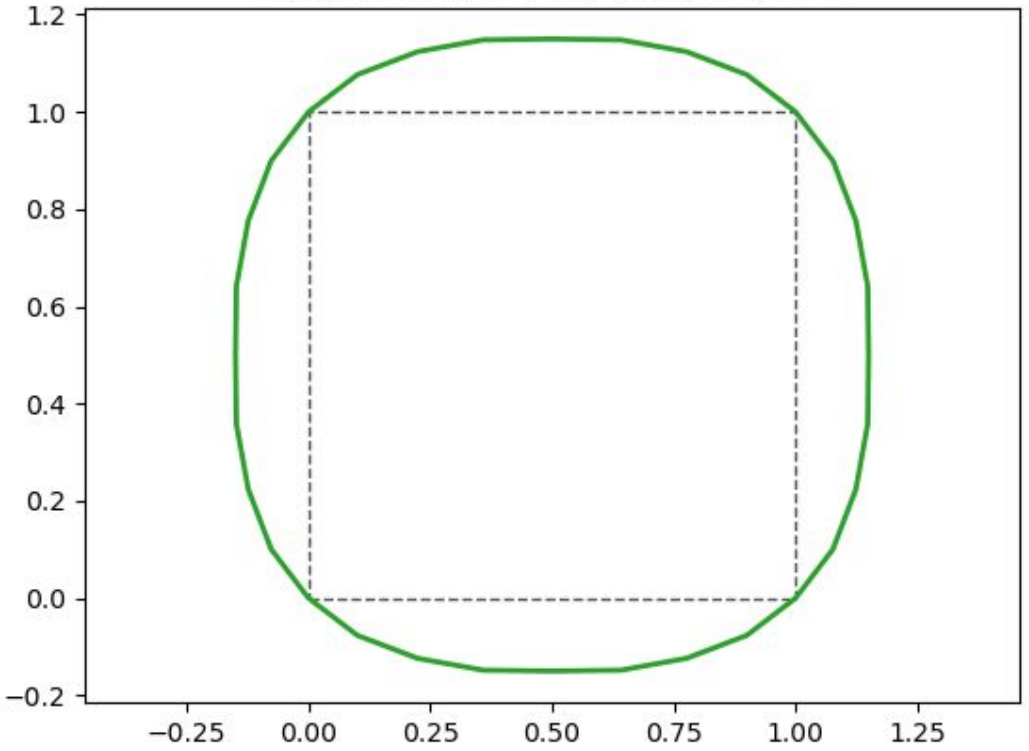


Figure 21 : “simple” de densité 3 avec l’algorithme de Four-point (w = 0.075)

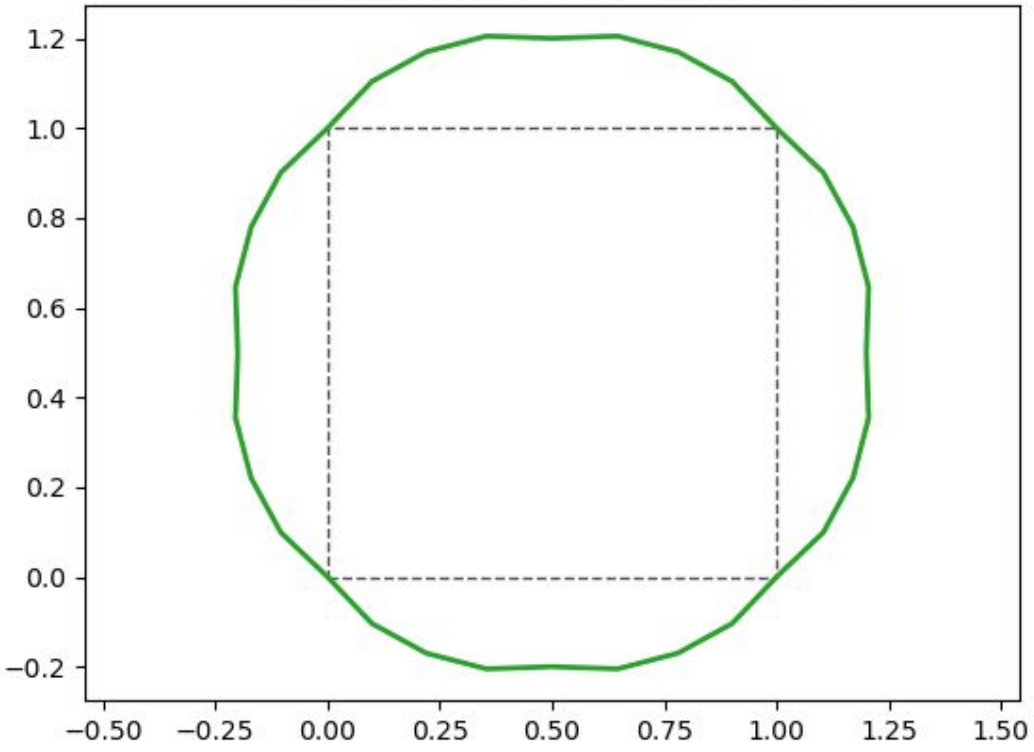


Figure 22 : “simple” de densité 3 avec l’algorithme de Four-point (w = 0.1)

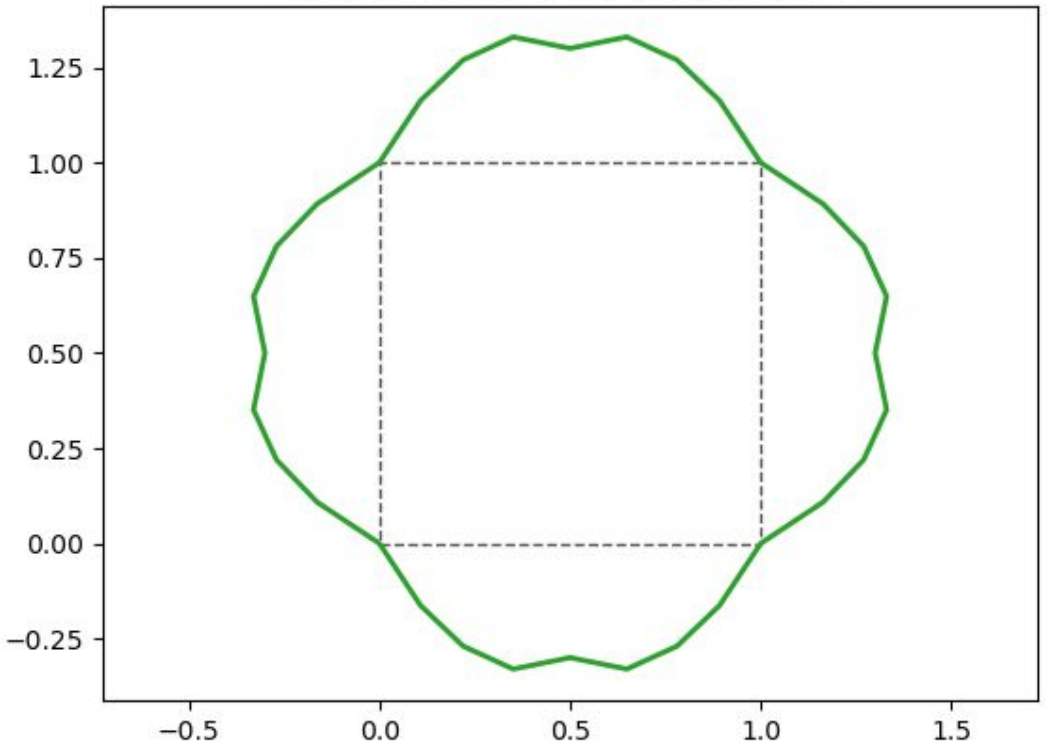


Figure 23 : “simple” de densité 3 avec l’algorithme de Four-point (w = 0.15)

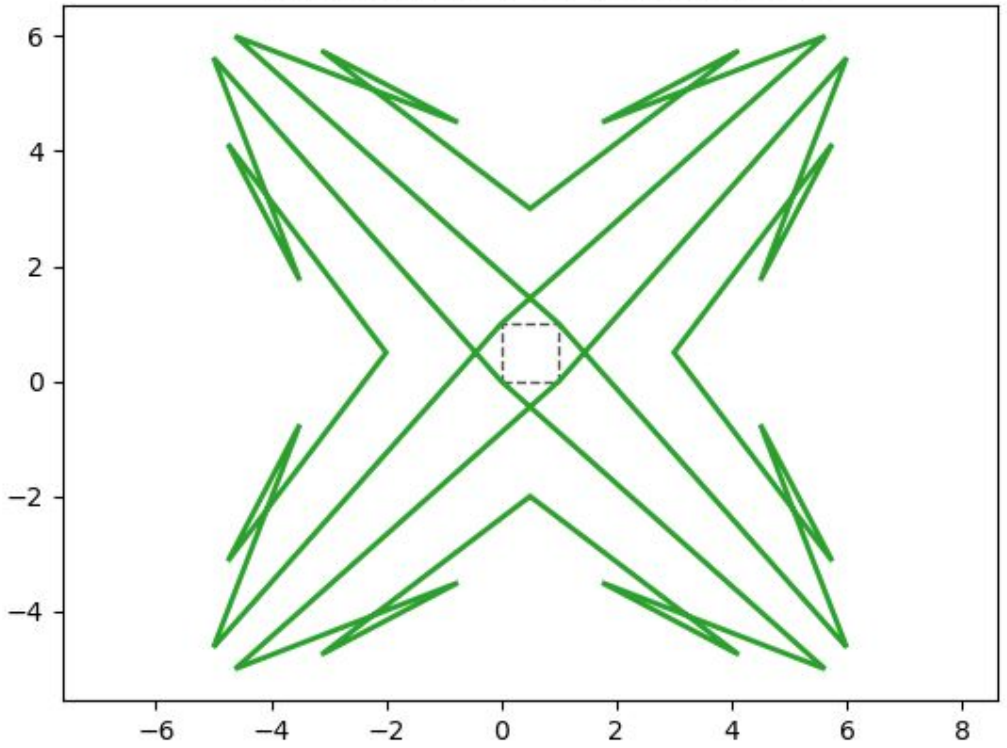


Figure 24 : “simple” de densité 3 avec l’algorithme de Four-point (w = 1.0)

Comparaison des trois algorithmes

En superposant les mêmes courbes des différents algorithmes sur un même plan, cette section à pour but de faire ressortir les différences des courbes résultantes de ses algorithmes et de mieux les identifier.

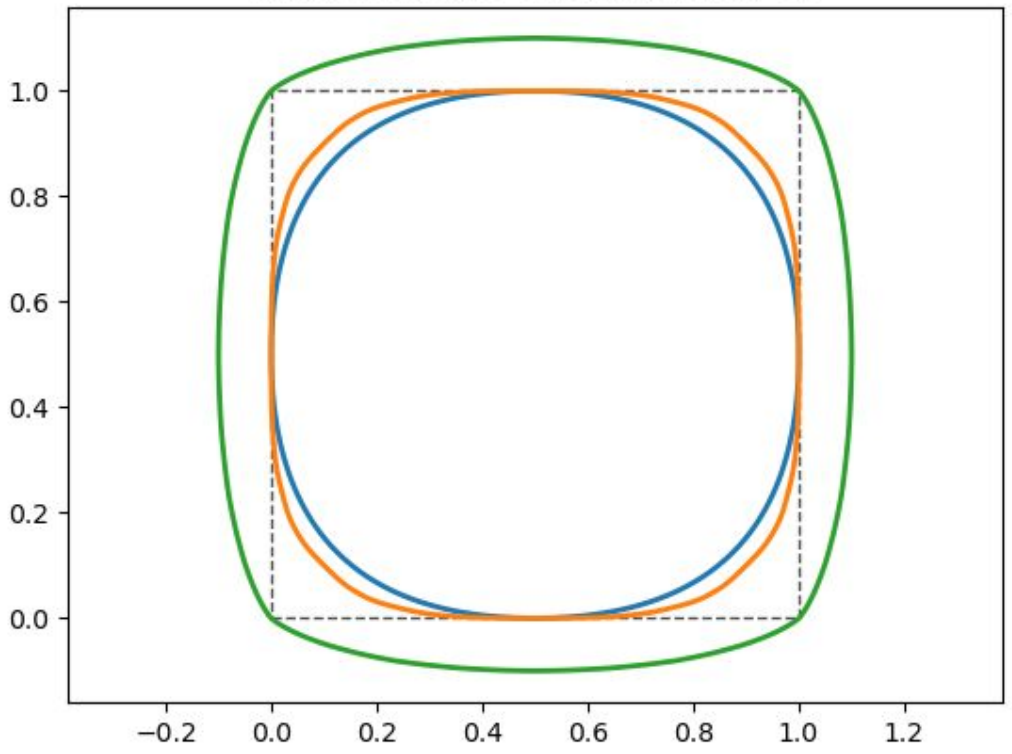


*Figure 25 : "simple" de profondeur 10 avec l'algorithme de **Chaikin (a = 0.25; b = 0.75) en bleu**, de **Corner cutting (a = 0.2; b = 0.8) en orange** et de **Four-point (w = 0.05) en vert***

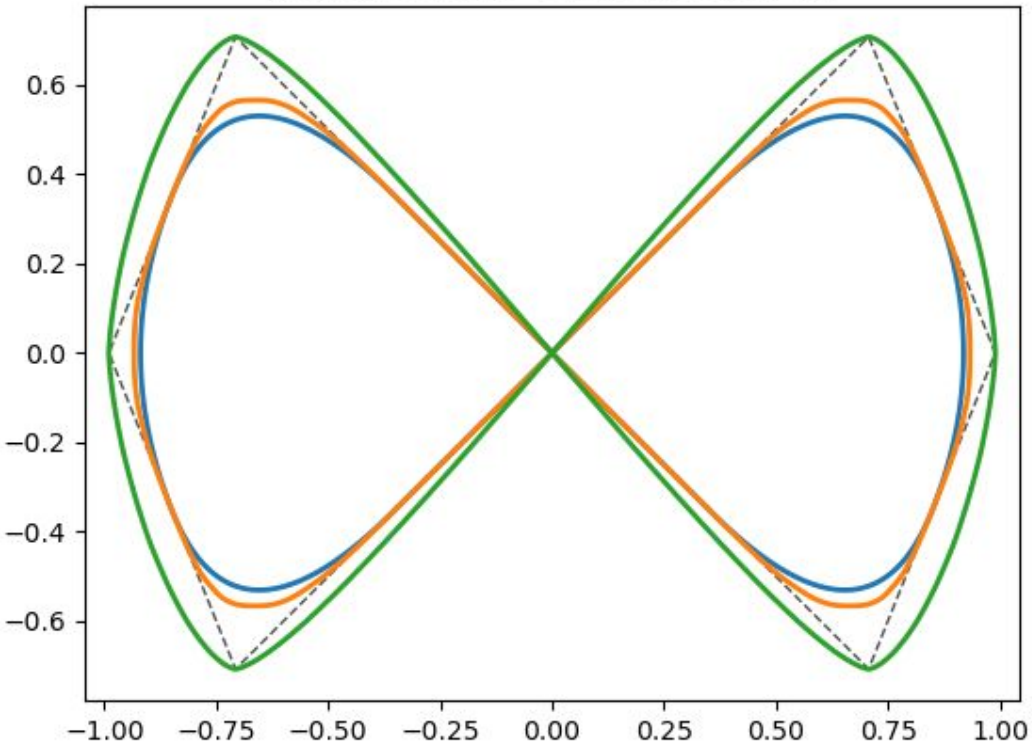


*Figure 26 : "infinity" de profondeur 10 avec l'algorithme de **Chaikin (a = 0.25; b = 0.75) en bleu**, de **Corner cutting (a = 0.2; b = 0.8) en orange** et de **Four-point (w = 0.05) en vert***

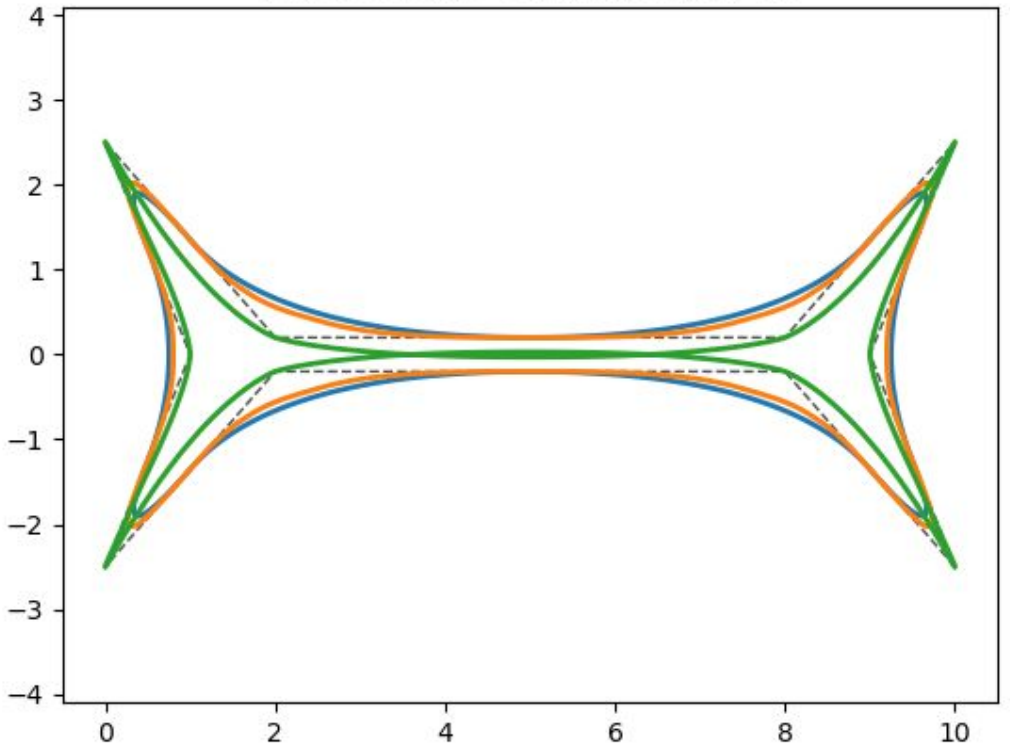


*Figure 27 : "bone" de profondeur 10 avec l'algorithme de **Chaikin (a = 0.25; b = 0.75) en bleu**, de **Corner cutting (a = 0.2; b = 0.8) en orange** et de **Four-point (w = 0.05) en vert***

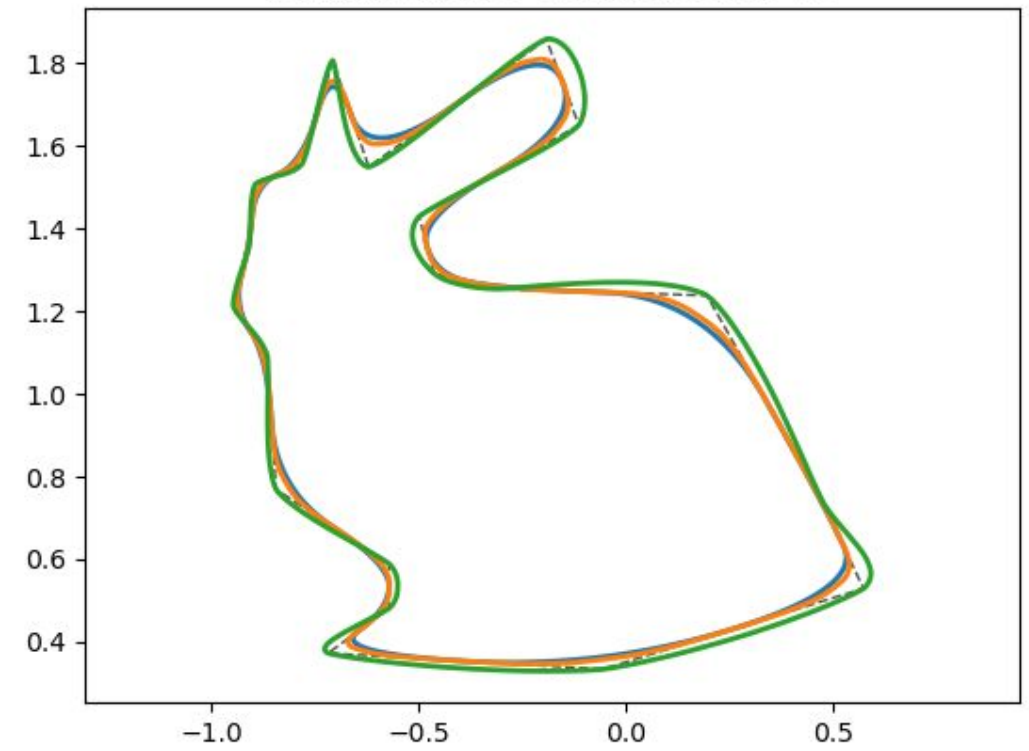


*Figure 28 : "bunny" de profondeur 10 avec l'algorithme de **Chaikin (a = 0.25; b = 0.75) en bleu**, de **Corner cutting (a = 0.2; b = 0.8) en orange** et de **Four-point (w = 0.05) en vert***

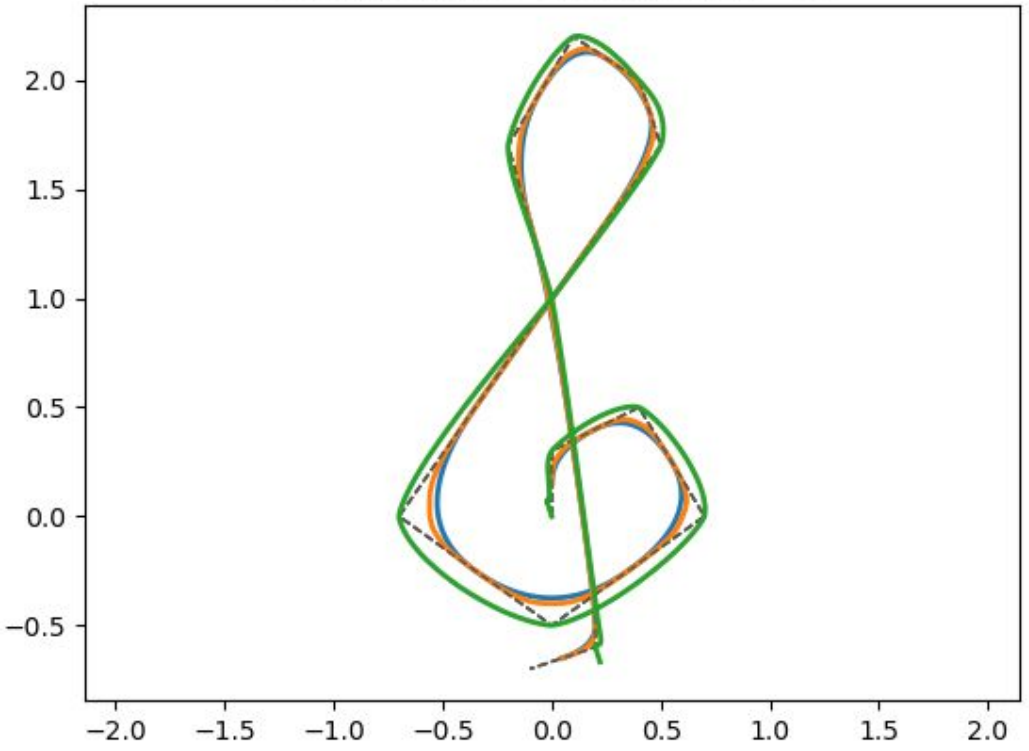


*Figure 29 : "treble" de profondeur 10 avec l'algorithme de **Chaikin (a = 0.25; b = 0.75) en bleu**, de **Corner cutting (a = 0.2; b = 0.8) en orange** et de **Four-point (w = 0.05) en vert***