Un arbre de Pythagore

Écrit à partir de la vidéo https://www.youtube.com/watch?v=BCR7IhRFNDo

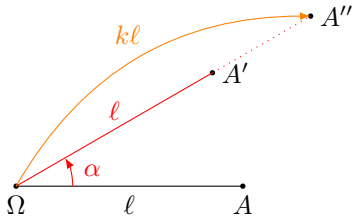
1 Notion de similitude

Considérons :

- un point Ω du plan,
- un réel k,
- un ange α.

À tout point A du plan, on peut considérer successivement :

- son image A' par la rotation de centre Ω et d'angle α,
- puis A'' l'image de A' par l'homothétie de centre Ω et de rapport k.



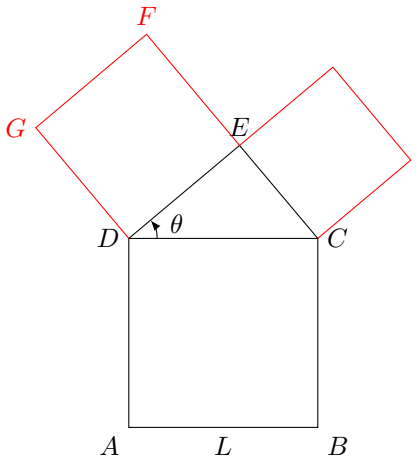
À l'aide de la trigonométrie, on démontre que :

$$\left.\begin{array}{l}\Omega\,(x_\Omega\,;y_\Omega)\\ A\,(a\,;b)\end{array}\right\}\Longrightarrow\begin{cases}A'\,(x_\Omega+(a-x_\Omega)\cos\alpha-(b-y_\Omega)\sin\alpha\,;y_\Omega+(a-x_\Omega)\sin\alpha+(b-y_\Omega)\cos\alpha)\\ A''\,(x_\Omega+k(a-x_\Omega)\cos\alpha-k(b-y_\Omega)\sin\alpha\,;y_\Omega+k(a-x_\Omega)\sin\alpha+k(b-y_\Omega)\cos\alpha)\end{cases}$$

On dit ici que A'' est obtenu à partir du point A par similitude de centre Ω, d'angle α et de rapport k.

2 Introduction à l'arbre de Pythagore

Partons d'un carré $ABCD$ de côté L, puis traçons un triangle rectangle ECD tel que $\widehat{CDE}=\theta$, sur lequel nous traçons des carrés :



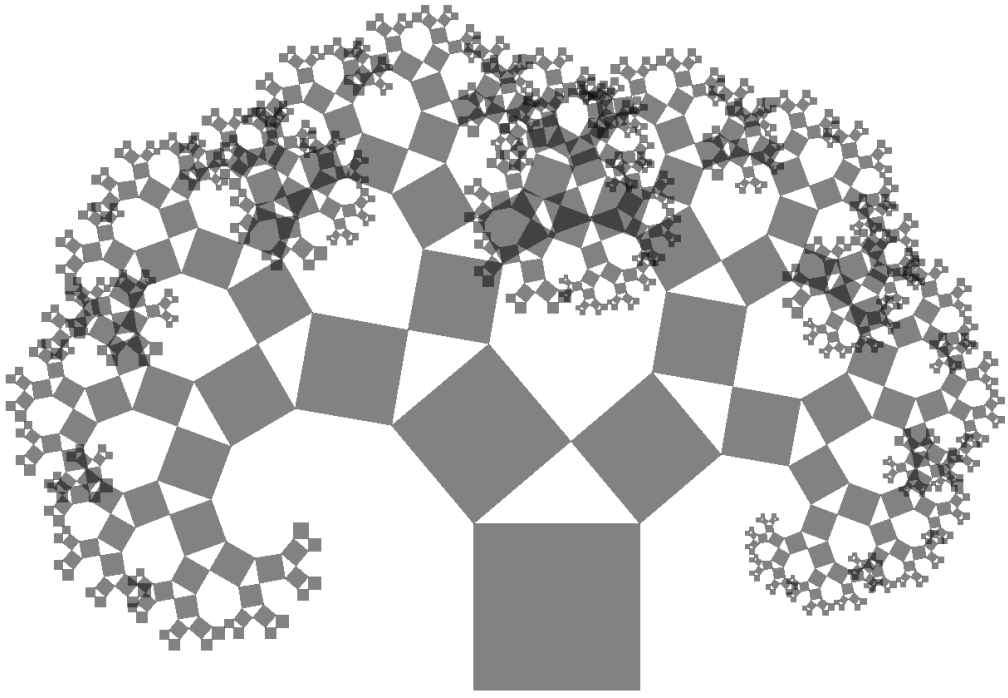
- Le point E est obtenu par similitude de centre D, d'angle θ et de rapport $k=L\cos\theta$ car $\dfrac{DE}{DC}=\cos\theta$.
- Le point F est obtenu à partir de D par rotation de centre E et d'angle $-\frac{\pi}{2}$.
- Le point G est obtenu à partir de E par rotation de centre D et d'angle $\frac{\pi}{2}$.

On peut construire de même le carré sur $[EC]$.

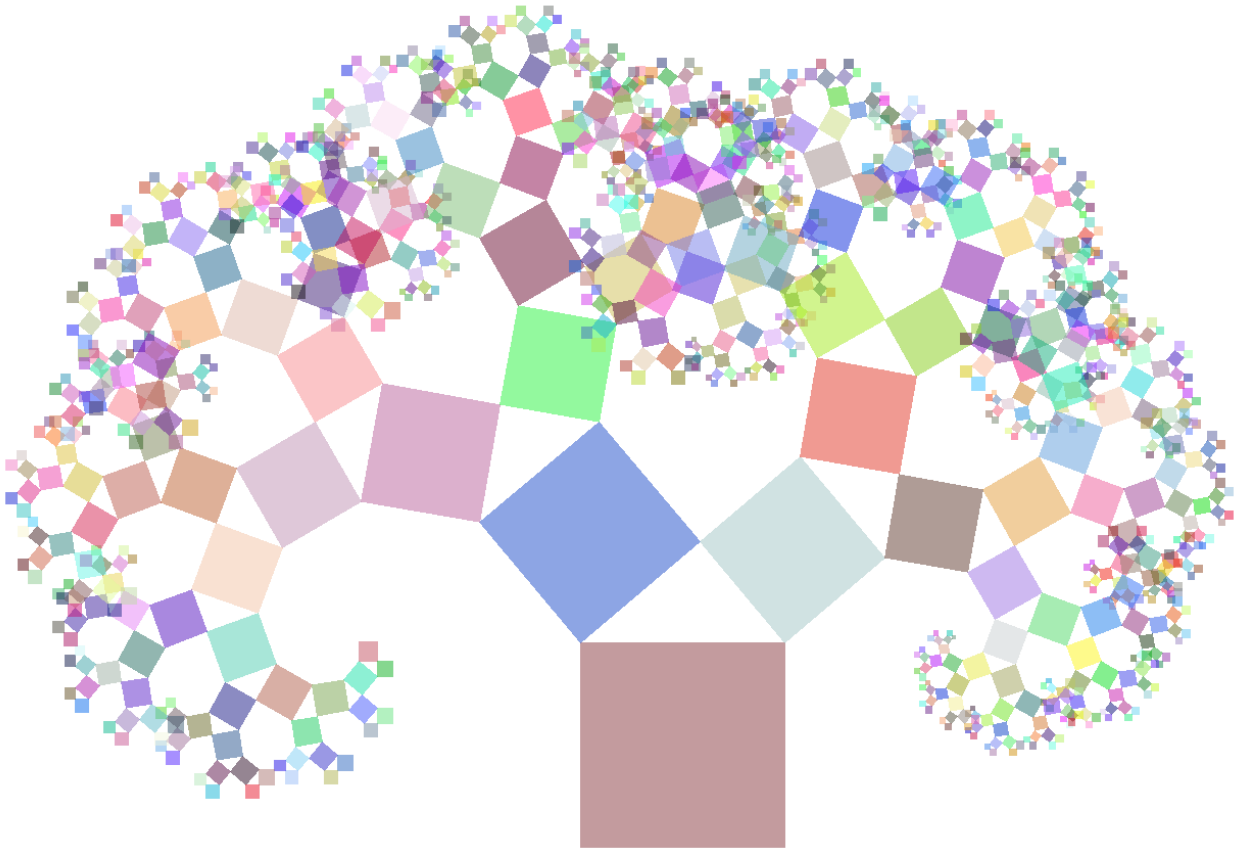
3 Construction de l'arbre de Pythagore

L'idée est de reproduire ce qui vient d'être expliqué sur les carrés obtenus, et sur ceux qui vont l'être par la suite. Nous allons faire cela à l'aide de Python :

```
from math import pi, cos, sin
from PIL import Image, ImageDraw

width = 1748
height = 1240

tree = Image.new('RGB', (width,height), (255,255,255))
draw = ImageDraw.Draw(tree,'RGBA')

theta = pi/180 * 40 # 40 degrés
L = 200 # longueur de l'arête du premier carré
A = (width/2 - L/2 , height-250)
B = (width/2 + L/2 , height-250)

def sim(centre, point, angle, k):
    # angle devient "-angle" dans les sinus à cause de la logique graphique de PIL
    x = centre[0] + k*(point[0]-centre[0])*cos(angle)-k*(point[1]-centre[1])*sin(-angle)
    y = centre[1] + k*(point[0]-centre[0])*sin(-angle)+k*(point[1]-centre[1])*cos(angle)

    return x,y

def arbre(A,B,angle,n):
    if n==0: return

    C = sim(B,A,-pi/2,1)
    D = sim(A,B,pi/2,1)
    E = sim(D,C,angle,cos(angle))

    draw.polygon([A,B,C,D], fill=(0,0,0,125))

    arbre(D, E, angle, n-1)
    arbre(E, C, angle, n-1)


arbre(A,B,theta,10)
tree.show()
```

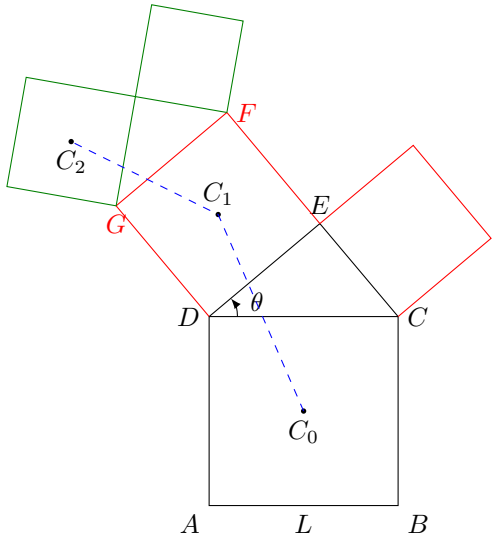


```python
from math import pi, cos, sin
from PIL import Image, ImageDraw
from random import randint

width = 1748
height = 1240

tree = Image.new('RGB', (width,height), (255,255,255))
draw = ImageDraw.Draw(tree,'RGBA')

theta = pi/180 * 40 # 40 degrés
L = 200 # longueur de l'arête du premier carré
A = (width/2 - L/2 , height-250)
B = (width/2 + L/2 , height-250)

def sim(centre, point, angle, k):
    # angle devient "-angle" dans les sinus à cause de la logique graphique de PIL
    x = centre[0] + k*(point[0]-centre[0])*cos(angle)-k*(point[1]-centre[1])*sin(-angle)
    y = centre[1] + k*(point[0]-centre[0])*sin(-angle)+k*(point[1]-centre[1])*cos(angle)

    return x,y

def arbre(A,B,angle,n):
    if n==0: return

    C = sim(B,A,-pi/2,1)
    D = sim(A,B,pi/2,1)
    E = sim(D,C,angle,cos(angle))

    draw.polygon([A,B,C,D], fill=(randint(0,255),randint(0,255),randint(0,255),125))

    arbre(D, E, angle, n-1)
    arbre(E, C, angle, n-1)


arbre(A,B,theta,10)
tree.show()
```



4 L'arbre est-il « infini » ?

Nous allons noter C_n le centre des carrés allant vers la gauche.



On peut observer que :

$$\frac{C_2C_1}{C_1C_0} = \cos\theta.$$

En effet, $DEFG$ est l'image de $ABCD$ (DC se transforme en DG) par la similitude de centre D, d'angle $\theta + \frac{\pi}{2}$ et de rapport $k = \cos\theta$ (par construction). De même, le carré vert est l'image du carré rouge par la similitude de centre G et de mêmes angles et rapport.

Toutes les longueurs sont donc multipliées par $\cos\theta$ pour passer d'un carré au suivant.

La suite $(C_{n-1}C_n)$ est donc géométrique de raison $\cos\theta$.

Calculons alors longueur :

$$\begin{aligned} L_n &= \sum_{k=0}^{n-1} C_kC_{k+1} = C_0C_1 + C_1C_2 + \cdots + C_{n-1}C_n \\ &= C_0C_1 \times (1 + \cos\theta + \cos^2\theta + \cdots + \cos^n\theta) \\ &= C_0C_1 \times \frac{1 - \cos^{n+1}\theta}{1 - \cos\theta}. \end{aligned}$$

Comme $0 \leqslant \cos\theta < 1$,

$$\lim_{n\to+\infty} L_n = \frac{C_0C_1}{1 - \cos\theta}.$$

L'arbre de Pythagore (noté $\mathcal{A}$) ne peut donc pas être « infini » (dans le sens où la distance entre C_0 et C_∞ ne peut pas être infinie).

On appellera *diamètre* le maximum de la distance entre C_0 et C_n, pour tout entier naturel n :

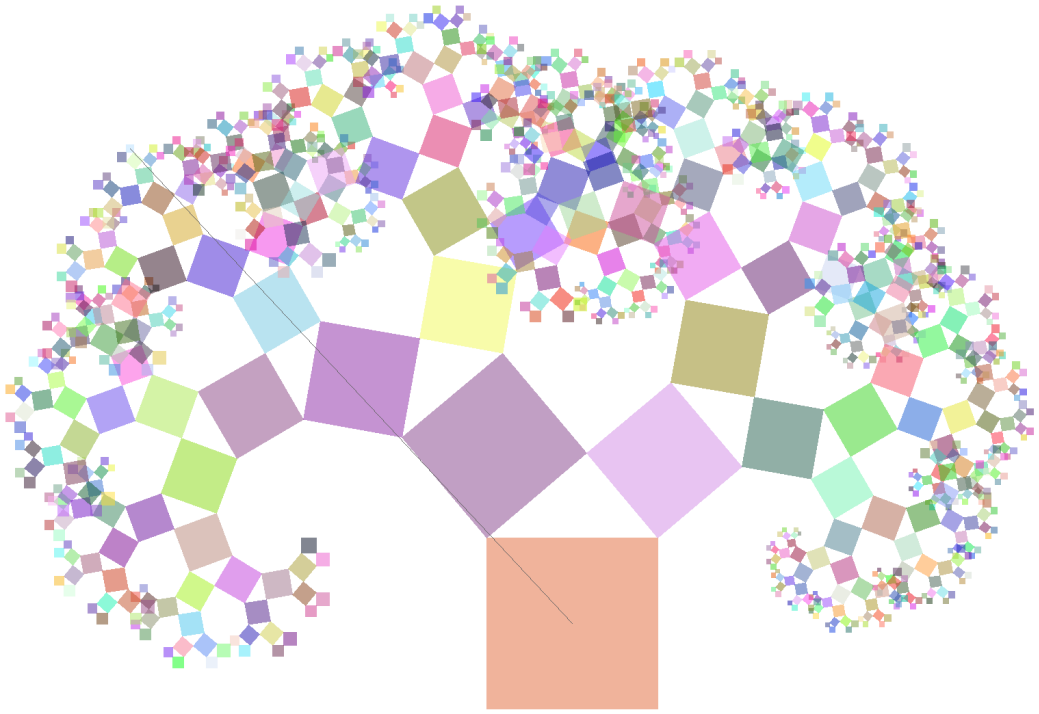
$$\text{diam}(\mathcal{A}) = \max\{C_0C_n,\ n \geqslant 1\}.$$

On peut ainsi modifier le script précédent pour déterminer une valeur approchée du diamètre (en pixels) de l'arbre (voir page suivante).

```python
from math import pi, cos, sin
from PIL import Image, ImageDraw, ImageFont
from random import randint

width = 1748
height = 1240

tree = Image.new('RGB', (width,height), (255,255,255))
draw = ImageDraw.Draw(tree,'RGBA')

theta = pi/180 * 40 # 40 degrés
L = 200 # longueur de l'arête du premier carré
A = (width/2 - L/2 , height-250)
B = (width/2 + L/2 , height-250)

centres = []

def sim(centre, point, angle, k):
    # angle devient "-angle" dans les sinus à cause de la logique graphique de PIL
    x = centre[0] + k*(point[0]-centre[0])*cos(angle)-k*(point[1]-centre[1])*sin(-angle)
    y = centre[1] + k*(point[0]-centre[0])*sin(-angle)+k*(point[1]-centre[1])*cos(angle)

    return x,y

def arbre(A,B,angle,n):
    if n==0: return

    global centres

    C = sim(B,A,-pi/2,1)
    D = sim(A,B,pi/2,1)
    E = sim(D,C,angle,cos(angle))

    centres.append(((A[0]+C[0])/2,(A[1]+C[1])/2))

    draw.polygon([A,B,C,D], fill=(randint(0,255),randint(0,255),randint(0,255),125))

    arbre(D, E, angle, n-1)
    arbre(E, C, angle, n-1)

def diam():
    global centres
    origine = centres[0] # premier centre
    d = 0

    for c in centres:
        if c != origine:
            distance = (origine[0]-c[0])**2 + (origine[1]-c[1])**2
            if distance > d:
                d = distance
                faraway = c

    return d**0.5, faraway

arbre(A,B,theta,10)
diametre = f"Diamètre de l'arbre: {diam()[0]}"
draw.text((10,10), diametre, font=ImageFont.truetype("arial.ttf", size=30), fill=(0, 0, 0,
    255))
# on trace le diamètre
draw.line([centres[0],diam()[1]],fill=(100,100,100),width=1)

tree.show()
```

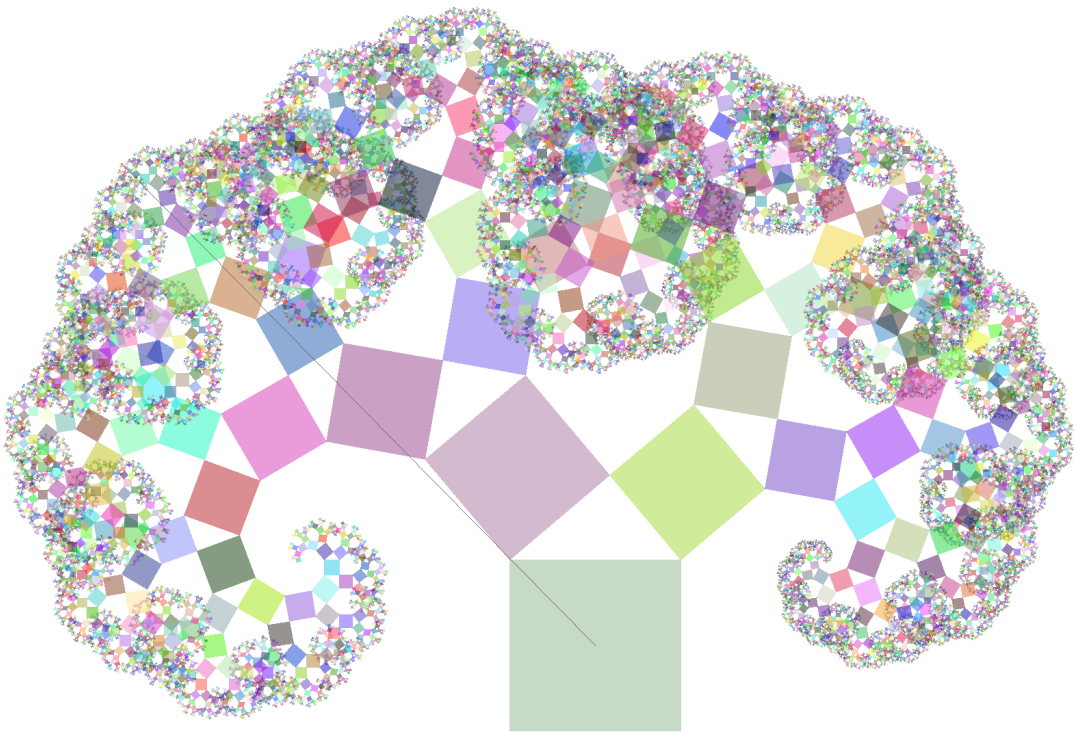
n=10

Diamètre de l'arbre: 948.0360910439098



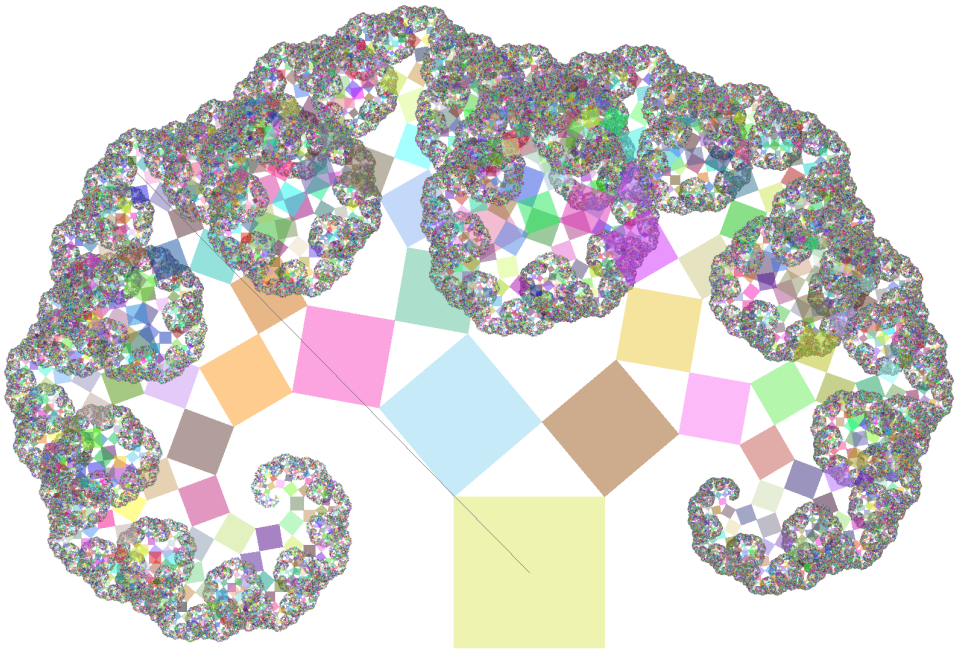
n=15

Diamètre de l'arbre: 986.4086876438117



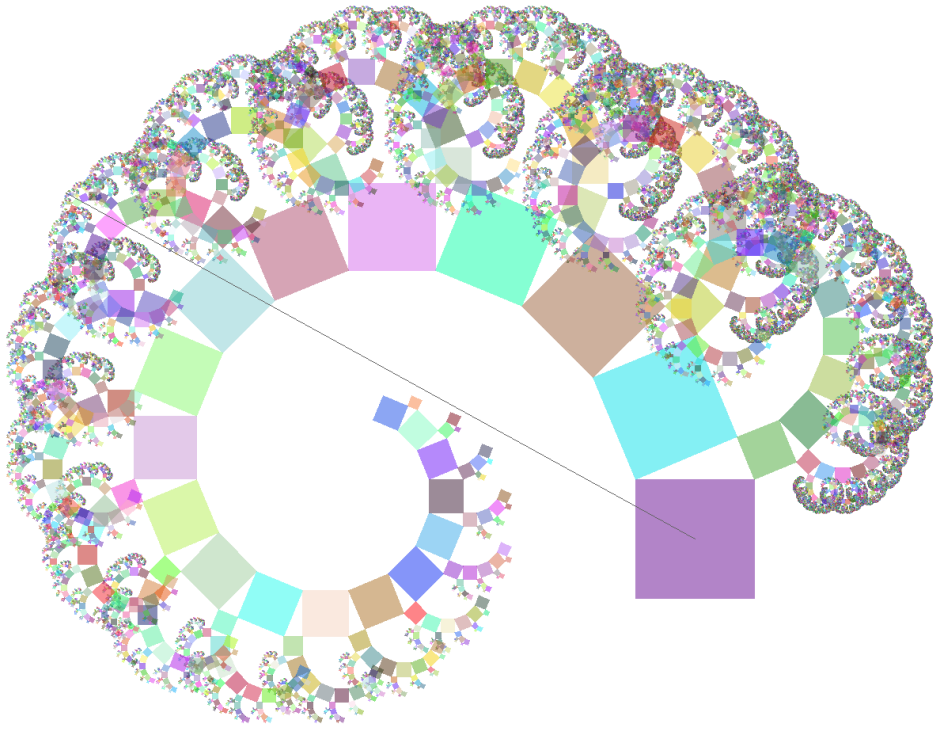
n=20

Diamètre de l'arbre: 993.0456320263777



n=20, angle=22.5

Diamètre de l'arbre: 906.0953567362632



Ce dernier exemple est intéressant : on aimerait que le nombre d'itérations soit plus grand afin que la « spirale » continue davantage... Nous allons donc légèrement modifier le script afin de le permettre car avec ce que nous avons, pour $n = 25$, ça plante...

```
from math import pi, cos, sin
from PIL import Image, ImageDraw, ImageFont
from random import randint

width = 1748
height = 1240

tree = Image.new('RGB', (width,height), (255,255,255))
draw = ImageDraw.Draw(tree,'RGBA')

theta = pi/180 * 22.5
L = 150 # longueur de l'arête du premier carré
A = (width/2 - L/2 +100, height-250)
B = (width/2 + L/2 +100, height-250)

centres = []

def sim(centre, point, angle, k):
    # angle devient "-angle" dans les sinus à cause de la logique graphique de PIL
    x = centre[0] + k*(point[0]-centre[0])*cos(angle)-k*(point[1]-centre[1])*sin(-angle)
    y = centre[1] + k*(point[0]-centre[0])*sin(-angle)+k*(point[1]-centre[1])*cos(angle)

    return x,y

def arbre(A,B,angle,n):
    if (n==0) or (length(A,B)<8): return

    global centres

    C = sim(B,A,-pi/2,1)
    D = sim(A,B,pi/2,1)
    E = sim(D,C,angle,cos(angle))

    centres.append(((A[0]+C[0])/2,(A[1]+C[1])/2))

    draw.polygon([A,B,C,D], fill=(randint(0,255),randint(0,255),randint(0,255),125))

    arbre(D, E, angle, n-1)
    arbre(E, C, angle, n-1)

def length(A,B):
    return (A[0]-B[0])**2 + (A[1]-B[1])**2

def diam():
    global centres
    origine = centres[0] # premier centre
    d = 0

    for c in centres:
        if c != origine:
            distance = length(origine,c) #(origine[0]-c[0])**2 + (origine[1]-c[1])**2
            if distance > d:
                d = distance
                faraway = c

    return d**0.5, faraway

arbre(A,B,theta,50)
diametre = f"Diamètre de l'arbre: {diam()[0]}"
draw.text((10,10), diametre, font=ImageFont.truetype("arial.ttf", size=30), fill=(0, 0, 0,
    255))
draw.line([centres[0],diam()[1]],fill=(100,100,100),width=1)

tree.show()
```

Nous avons introduit :

- une nouvelle fonction `length` qui permet de renvoyer le carré de la distance entre deux points ;
- un test dans la fonction `arbre` : on n'exécute la fonction que si $AB^2 > 8$ (par exemple). En effet, inutile d'aller plus loin si l'on ne voit plus les points.

Cette modification donne, pour $n = 50$:

Diamètre de l'arbre: 899.7479565860084