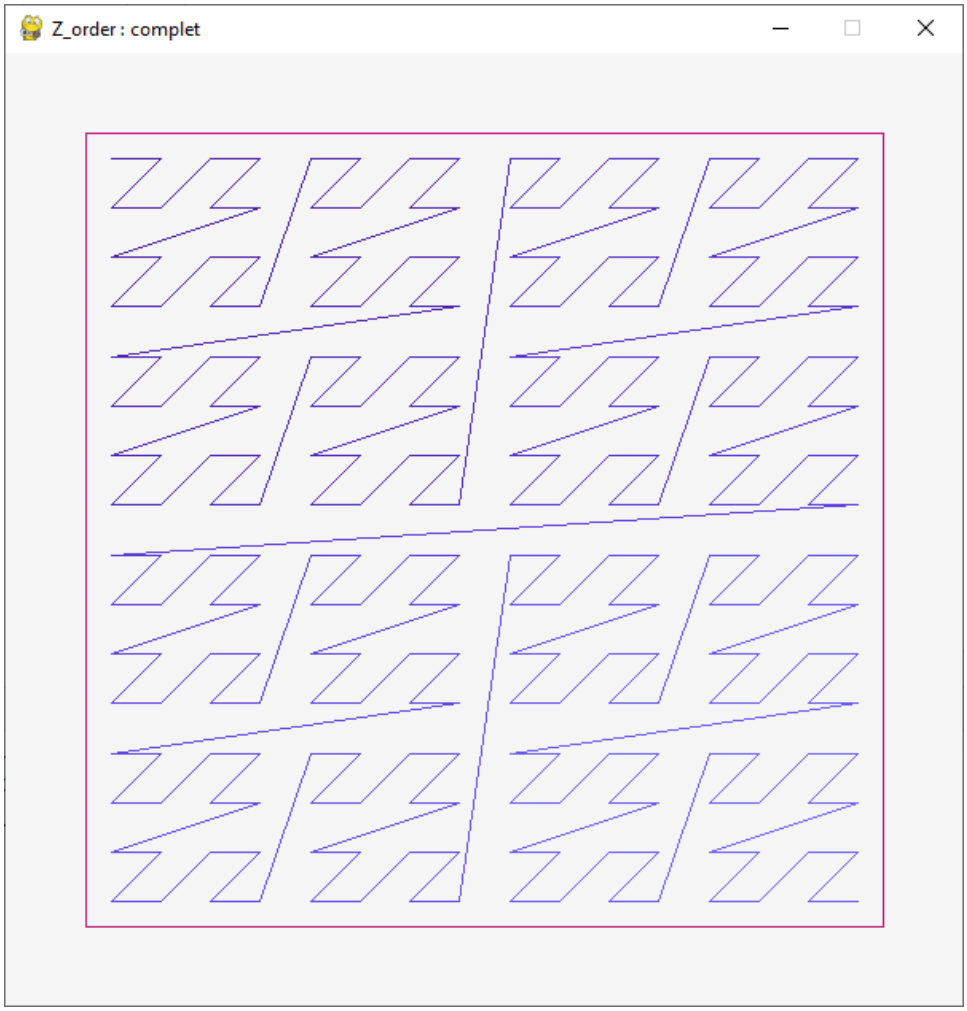
Curs 11

(*plan de curs*)

1. Curba lui Lebesgue: definiţie, proprietăţi



1.a. Abordare pe biţi:

```
def Z_orderPeBiti():
    k = 6  # atentie: 0 < k < 15
    nrPuncte = 1 << (2 * k)
    lat = 1 << k
    C.setXminXmaxYminYmax(-1, lat, lat, -1)
    zvechi = 0
```

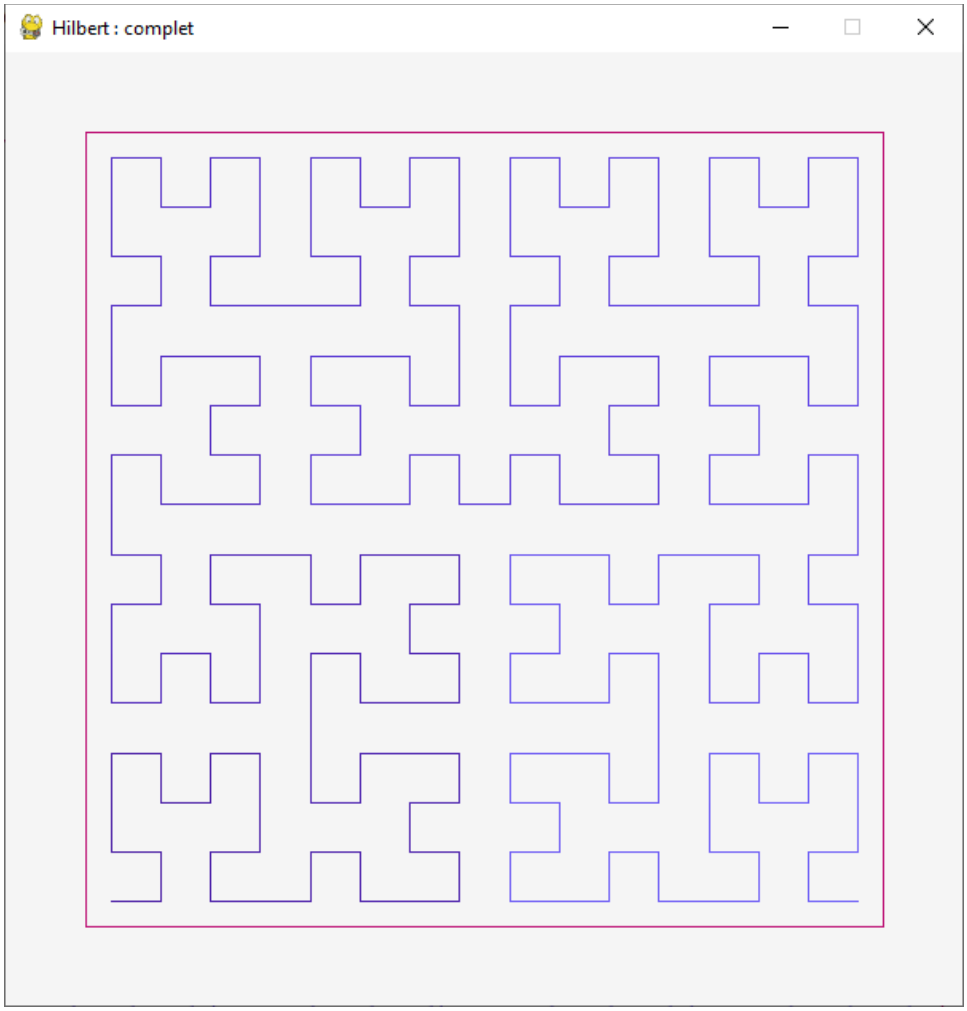
```
    for n in range(nrPuncte):
        x = 0
        y = 0
        nn = n
        for h in range(k):
            x |= (nn & 1) << h
            nn >>= 1
            y |= (nn & 1) << h
            nn >>= 1
        znou = complex(x, y)
        C.drawLine(znou, zvechi, Color.Index(n // 10))
        zvechi = znou
        if C.mustClose(): return
```

1.b. Abordare cu transformări iterate:

```
def Z_order():
    c0 = (1 + 1j) / 2
    c1 = (1 + 1j) / 4
    c2 = (1 + 3j) / 4
    c3 = (3 + 3j) / 4
    c4 = (3 + 1j) / 4

    #  sk(z)=c0+0.5*(z-c0) +(ck-c0)=ck+0.5*(z-c0)
    def s1(z):
        return c1 + 0.5 * (z - c0)

    def s2(z):
        return c2 + 0.5 * (z - c0)

    def s3(z):
        return c3 + 0.5 * (z - c0)

    def s4(z):
        return c4 + 0.5 * (z - c0)

    def transforma(li):

        rez = []
        for z in li:
            rez.append(s2(z))
        for z in li:
            rez.append(s3(z))
        for z in li:
            rez.append(s1(z))
        for z in li:
            rez.append(s4(z))
        return rez

    def traseaza(li):
        C.fillScreen()
        #  trasam chenarul
        C.drawNgon([0, 1, 1 + 1j, 1j], Color.Index(500))
        #  desenam curba curenta
```

```
        for n in range(1, len(li)):
            C.drawLine(li[n - 1], li[n], Color.Index(n // 5))

C.setXminXmaxYminYmax(-0.1, 1.1, -0.1, 1.1)
fig = [c0]
for k in range(5):
    fig = transforma(fig)
    traseaza(fig)
    if C.mustClose(): return
    C.wait(100)
```

2. Curba lui Hilbert: definiţie, proprietăţi



2.a. Abordare recursivă:

```
def HilbertRecursiv():
    za = 0
    zb = 0

    #   z0 = coltul curent
    #   d1 = prima deplasare
    #   d2 = a doua deplasare
    #  -d1 = a treia deplasare
    def genereaza(z0, d1, d2, niv):
        nonlocal za, zb
        if niv <= 0:
            zb = z0 + (d1 + d2) / 2  # zb = centrul patratului curent
            C.drawLine(za, zb, Color.Red)
            za = zb  # za = centrul vechi
        else:
            niv -= 1
            d1 *= 0.5
            d2 *= 0.5
            genereaza(z0, d2, d1, niv)
            genereaza(z0 + d1, d1, d2, niv)
            genereaza(z0 + d1 + d2, d1, d2, niv)
            genereaza(z0 + d1 + d2 + d2, -d2, -d1, niv)

    C.setXminXmaxYminYmax(-0.1, 1.1, -0.1, 1.1)
    z0 = 0
    d1 = 1j
    d2 = 1
    #   traseu prin patratul initial:
    #   z0 = coltul din stanga jos
    #   z0+d1 = coltul din stanga sus
    #   z0+d1+d2 = coltul din dreapta sus
    #   z0+d1+d2-d1 = z0+d2 = coltul din dreapta jos
    nrEtape = 5
    za = z0
    genereaza(z0, d1, d2, nrEtape)
    C.drawLine(za, z0 + d2, Color.Red)
    C.refreshScreen()
```

2.b. Abordare cu transformări iterate:

```
def Hilbert():
    c0 = (1 + 1j) / 2
    c1 = (1 + 1j) / 4
    c2 = (1 + 3j) / 4
    c3 = (3 + 3j) / 4
    c4 = (3 + 1j) / 4
```

```
def s1(z):
    return c1 + 0.5j * (z - c0).conjugate()

def s2(z):
    return c2 + 0.5 * (z - c0)

def s3(z):
    return c3 + 0.5 * (z - c0)

def s4(z):
    return c4 - 0.5j * (z - c0).conjugate()

def transforma(li):

    rez = []
    for z in li:
        rez.append(s1(z))
    for z in li:
        rez.append(s2(z))
    for z in li:
        rez.append(s3(z))
    for z in li:
        rez.append(s4(z))
    return rez

def traseaza(li):
    C.fillScreen()
    #  trasam chenarul
    C.drawNgon([0, 1, 1 + 1j, 1j], Color.Index(500))
    #  desenam curba curenta
    for n in range(1, len(li)):
        C.drawLine(li[n - 1], li[n], Color.Index(n // 5))

C.setXminXmaxYminYmax(-0.1, 1.1, -0.1, 1.1)
fig = [c0]
for k in range(5):
    fig = transforma(fig)
    traseaza(fig)
    if C.mustClose(): return
    C.wait(100)
```