Indicații de rezolvare tema 5

În cadrul temei 5, vom desena fractali **"clasici"** și fractali **obținuți prin auto-asemănare**:

1	2	f. clasici
3	4	f. obț. prin auto-asemănare
exemple	exerciții	

Termenul fractal a fost generalizat pentru a include obiecte din afara definiției originale a lui Mandelbrot. Prin obiect fractal înțelegem orice obiect cu proprietatea de auto-asemănare (self-similarity în lb. engleză). Obiectele obținute în cele ce urmează sunt aproximații ale unui obiect fractal ideal, fiind obținute într-un număr finit de iterații.

Mulțimea Julia-Fatou se obține utilizând un proces iterativ:

$$\begin{cases} z_0 \in \mathbb{C} \\ z_{n+1} = z_n^2 + c, c \in \mathbb{C} \end{cases}$$

Un număr $x \in \mathbb{C}$ este în mulțimea Julia-Fatou $\boldsymbol{\mathcal{J}_c}$ dacă, plecând mai sus cu $z_0 = x$, șirul $(z_n)_{n\geq 0}$ astfel obținut **nu** satisface niciuna din condițiile $(\exists z \in \mathbb{C})(\lim\limits_{n\to\infty} z_n = z)$, $\lim\limits_{n\to\infty} |z_n| = \infty$.

Cele două condiții se pot rescrie:

$(\lim\limits_{n\to\infty} z_n = z) \Leftrightarrow (\forall \varepsilon > 0)(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)(|z_n - z| < \varepsilon)$ și

$(\lim\limits_{n\to\infty} |z_n| = \infty) \Leftrightarrow (\forall \varepsilon > 0)(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)(|z_n| > \varepsilon)$.

În program cele 2 condiții au fost utilizate sub forma $(\exists n_0 > 0)(z_{n_0} = z_{n_0+1})$ și $(\exists n_0 \geq 0)$ $(\exists M > 0)(|z_{n_0}| > M)$, însemnând că într-un număr finit de iterații n_0 se testează dacă șirul $(z_n)_{n\geq 0}$ devine constant sau dacă $|z_n|$ depășește un număr M (ales suficient de mare).

Mulțimea Mandelbrot se obține tot utilizând un proces iterativ:

$$\begin{cases} z_0 = 0 + 0\mathrm{i} \\ z_{n+1} = z_n^2 + c, c \in \mathbb{C} \end{cases}$$

Un număr $c \in \mathbb{C}$ este în mulțimea Mandelbrot $\boldsymbol{\mathcal{M}}$ dacă șirul $(z_n)_{n\geq 0}$ satisface condiția $\lim\limits_{n\to\infty} |z_n| \neq \infty$.

Condiția se poate rescrie:

$(\lim\limits_{n\to\infty} |z_n| \neq \infty) \Leftrightarrow (\exists \varepsilon > 0)(\forall n_0 \in \mathbb{N})(\exists n \geq n_0)(|z_n| \leq \varepsilon)$.

Nu folosim această condiție, ci contrapoziția proprietății $c \in \boldsymbol{\mathcal{M}} \Rightarrow (\forall n \geq 0)(|z_n| \leq 2)$: $(\exists n \geq 0)(|z_n| > 2) \Rightarrow c \notin \boldsymbol{\mathcal{M}}$. Acest lucru înseamnă că procesul iterativ se oprește dacă găsim o valoare $|z_n| > 2$.

În privința clasificării punctelor care nu aparțin mulțimii Mandelbrot prin colorarea cu culori diferite în funcție de numărul de iterații necesar pentru a detecta neapartenența, rezultatul ar trebui să fie asemănător cu imaginea[1]:

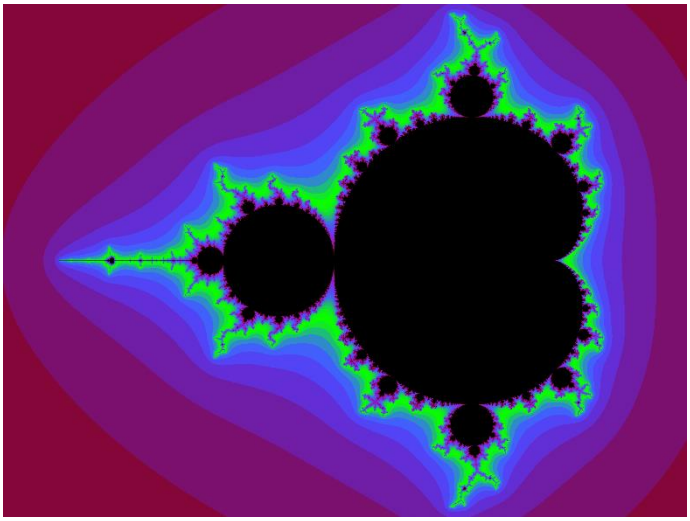

Fig. 1. Clasificare puncte neaparținând mulțimii Mandelbrot

Construim fractalii prin auto-asemănare utilizând **geometria "turtle"**. În acest stil de grafică imaginile sunt obținute prin deplasarea unui cursor pe ecran conform unor comenzi: desenare, rotație către stânga sau dreapta cu un anumit unghi. În construcția fractalilor ne putem folosi și de L-sisteme, dar utilizarea geometriei "turtle" rămâne obligatorie. Avem furnizate clase care implementează geometria "turtle":

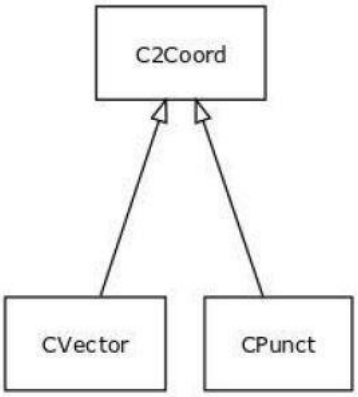

Fig. 2. Bibliotecă pentru grafica "turtle"

Clasa `CVector` conține următoarele metode pentru cursorul "turtle":

- `CPunct CVector::getDest(CPunct &orig, double lungime)` – returnează un punct care reprezintă destinația vectorului $\vec{v}$ = `lungime` $\cdot \overrightarrow{(*this)}$ aplicat în punctul `orig`,
- `void CVector::rotatie(double grade)` – rotește cu `grade` vectorul $\overrightarrow{(*this)}$,

[1] https://commons.wikimedia.org/wiki/File:Mandelbrot_set_with_coloured_environment.png

- `void CVector::deseneaza(CPunct p, double lungime)` – desenează un segment de dreaptă între punctul `p` și punctul destinație a vectorului $\vec{v}$ = `lungime` $\cdot$ $(*\overrightarrow{this})$ aplicat în punctul `p`.

Vectorul clasei este întotdeauna normalizat.

Fractalii obținuți prin auto-asemănare dați ca exemple sunt realizați recursiv, o curbă de nivel *N*+1 fiind alcătuită din mai multe curbe de nivel *N*.

De exemplu, un segment de **curbă Koch** *CK* de nivel *N*+1 este alcătuit din 4 segmente de curbă *CK* de nivel *N*. Un segment *CK*(0) este un segment de dreaptă. Fig. 3 descrie cum construim un segment de curbă *CK*(*N*+1) care începe în punctul *A* și se termină în punctul *B* ($\overrightarrow{OB} = \overrightarrow{OA} +$ `lungime` $\cdot \vec{v}$), prin concatenarea a 4 segmente de curbă *CK*(*N*) ($|AC| = |CE| = |ED| = |DB| = \frac{|AB|}{3}$): primul începe din *A* și se termină în *C*, după care rotim $\vec{v}$ cu 60°, al doilea începe în *C* și se termină în *E*, rotim $\vec{v}$ cu -120°, al treilea începe în *E* și se termină în *D*, după care rotim $\vec{v}$ cu 60° și al patrulea începe în *D* și se termină în *B*. Această construcție este implementată în metoda `void CCurbaKoch::segmentKoch(double lungime, int nivel, CPunct &p, CVector v)` care, dacă nivelul este 0 atunci desenează un segment utilizând `v.deseneaza(p, lungime)`, altfel desenează cele 4 segmente *CK* care au o treime din lungimea segmentului *CK* de nivel superior:

```
segmentKoch(lungime / 3.0, nivel - 1, p, v);
p1 = v.getDest(p, lungime / 3.0);
v.rotatie(60);
segmentKoch(lungime / 3.0, nivel - 1, p1, v);
p1 = v.getDest(p1, lungime / 3.0);
v.rotatie(-120);
segmentKoch(lungime / 3.0, nivel - 1, p1, v);
p1 = v.getDest(p1, lungime / 3.0);
v.rotatie(60);
segmentKoch(lungime / 3.0, nivel - 1, p1, v);
```

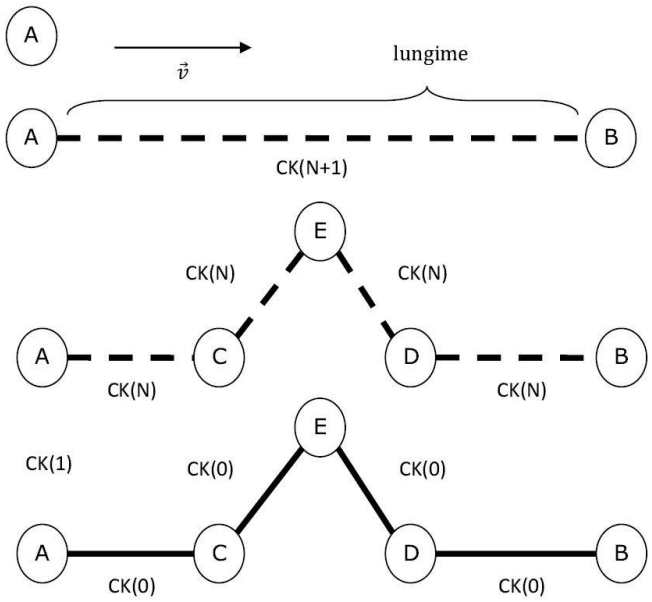

Fig. 3. Desenarea unui segment al curbei lui Koch

Observații

Apăsarea tastelor incrementează valoarea variabilei `nivel`. Această funcționalitate trebuie să rămână.

Funcțiile `glRasterPos2d` și `glutBitmapCharacter` ne permit să afișăm text în viewport-ul OpenGL.