računalništvo

Stare krivulje, moderna orodja

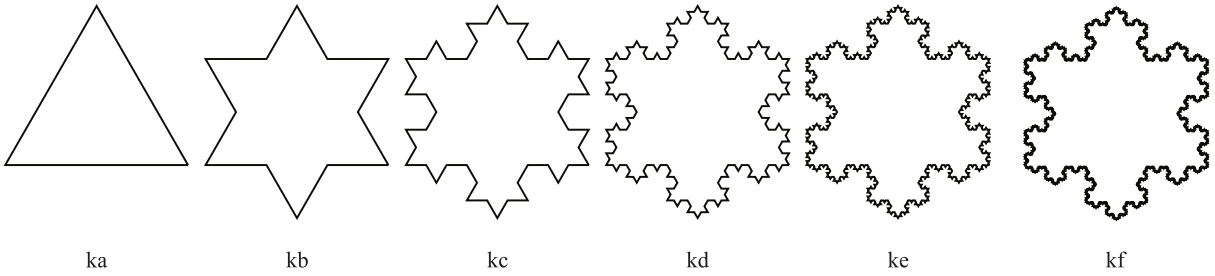
CIRIL PETR

Pričujoči prispevek je bil s podobno vsebino že objavljen v prvi številki sedemnajstega letnika Preseka leta 1990/91. Večina bralcev Preseka vas takrat še ni bila rojena, obstajala pa nista niti spletni protokol *http* niti programski jezik *Java*. Da bi tudi pri sedanji generaciji bralcev vzpodbudili zanimanje za te krivulje, smo se odločili stari prispevek posodobiti.

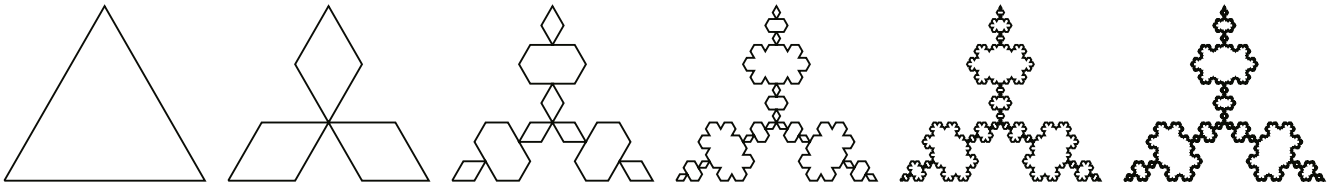
Najprej si bomo ogledali nekaj precej starih, vendar znamenitih krivulj in o vsaki kaj povedali. Potem se bomo seznanili z enostavnim načinom generiranja teh krivulj in na koncu predstavili preprost program, s katerim bomo lahko krivulje risali na računalniku, ter si izmišljali svoje.

Kochova snežinka

Začnimo z enakostraničnim trikotnikom (slika 1a). V prvem koraku razdelimo vsako stranico trikotnika na tretjine, srednje odseke stranic pa nadomestimo z novimi, za tretjino manjšimi enakostraničnimi trikotniki (slika 1b). V drugem koraku naredimo enako z vsako od stranic tako dobljenega poligona, ki spominja na Davidovo zvezdo. Postopek lahko ponavljamo, dokler nam to dopušča natančnost risanja. Že po nekaj korakih postopka lahko približno vidimo, kakšen bo izgled krivulje, obris lika, če bi postopek izvedli neskončnokrat (slika 1f). Krivulja, ki jo dobimo, če postopek izvedemo neskončnokrat, se imenuje *Kochova snežinka*, saj je podobna pravi snežinki. Prvi jo je raziskoval v začetku prejšnjega stoletja švedski matematik Niels Fabian Helge von Koch. Zelo je zanimiva, saj ima neskončno velik obseg, pa ven-



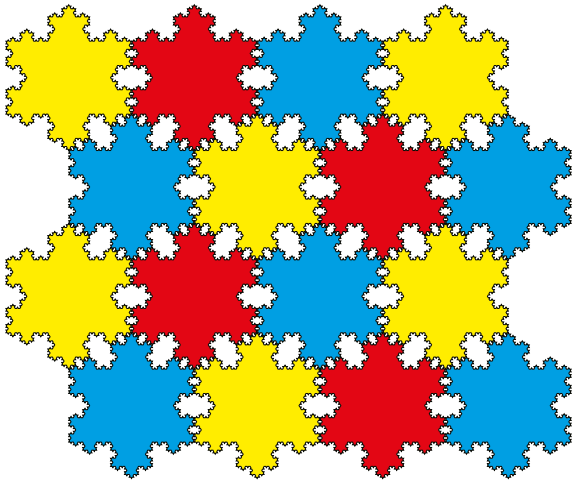
Slika 1. Prvih šest korakov razvoja Kochove snežinke



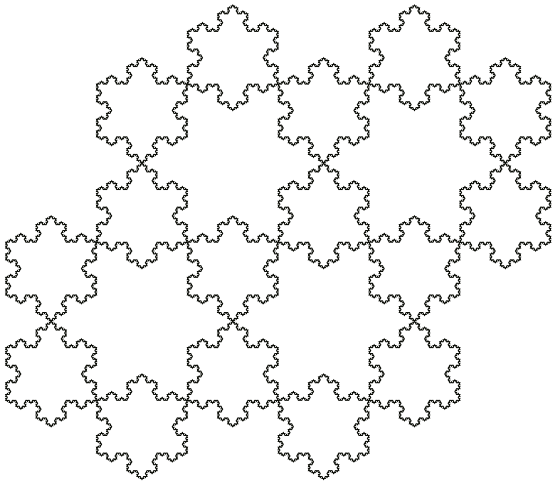
Slika 2. Prvih šest korakov razvoja obrnjene Kochove snežinke

dar obsega končno površino. Komur so domača geometrijska zaporedja, mu obsega in površine na določenem koraku postopka ne bo težko izračunati.

Poglejmo, kakšen lik dobimo, če srednje dele stranic nadomestimo s trikotniki, obrnjenimi navznoter (slika 2). Obseg lika, če postopek ponovimo neskončnokrat, je enak obsegu *Kochove snežinke*, površina lika pa je manjša od površine začetnega trikotnika za toliko, za kolikor je površina *Kochove snežinke* večja od površine istega začetnega trikotnika. Imenujmo ta lik *obratna Kochova snežinka*. Ravnino lahko popolnoma prekrijemo z izmenjajočima se likoma *Kochove snežinke* in *obratne Kochove snežinke* (slika 3). Na sliki 4 je prikazano prekrivanje ravnine z dvema, po njuni površini različnima Kochovima snežinkama. Razmerje njunih površin je ena proti tri. Omenimo še, da lahko podoben proces, kot smo ga prikazali nad trikotnikom, izvedemo nad tetraedrom.



Slika 3. Prekrivanje ravnine s Kochovo in obratno Kochovo snežinko



Slika 4. Prekrivanje ravnine z dvema, po površini različnima Kochovima snežinkama v razmerju ena proti tri

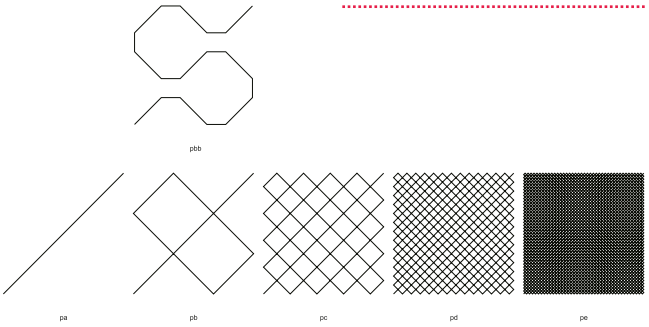
■ Peanova krivulja

Peanovo krivuljo dobimo z naslednjim postopkom. V prvem koraku kvadratu narišemo diagonalo (slika 5a). V drugem koraku kvadrat razdelimo na devet enakih kvadratov in jim narišemo vse diagonale z eno samo potezo, tako da nikjer ne sekamo svoje poti (slika 5b). V tretjem koraku naredimo enako z vsakim od manjših kvadratov (slika 5c). Ker se na slikah 5 diagonale kvadratov v kotih stikajo, ni jasno, kako krivuljo narišemo. Jasno pa bo, če zavoje krivulje porežemo (slika 5b'). Krivulja, ki jo dobimo, če postopek ponovimo neskončnokrat, se imenuje *Peanova krivulja*. Opazimo, da se krivulja z vsakim naslednjim korakom vedno bolj gosto vije po površini, omejeni s kvadratom. *Peanova krivulja* je *prostor zapolnjujoča krivulja*, kar pomeni, da gre skozi vsako točko površine, omejene s kvadratom. To krivuljo je v drugi polovici 19. stoletja odkril italijanski matematik in logik Giuseppe Peano.

■ Zmajeva krivulja

Zmajevo krivuljo lahko le za nekaj prvih korakov dobimo s prepogibanjem papirja. Vzemimo dolg papirnati trak, ki predstavlja začetek razvoja *zmajeve krivulje* (slika 6). Trak prepognemo po polovici in odpremo do pravega kota. Trak potem dvakrat zaporedoma prepognemo po polovici in ga odpremo do pravih kotov, trak trikrat zaporedoma prepognemo po polovici itd. Kdor ima izkušnje s prepogibanjem papirja ve, da se običajno papir ne da prepogniti več kot sedemkrat. Kot zanimivost lahko omenimo, da je dvanajst zaporednih pregibov papirja svetovni rekord, za kar je bil potreben več kot kilometer dolg trak zelo tankega papirja. Tako prepognjen paket papirja je imel kar $2^{12} = 4096$ slojev.

Naj bo papir na začetku še tako tako dolg, je prostor, ki ga zaseda *zmajeva krivulja*, po vsakem koraku manjši. Vendar pa je *zmajeva krivulja*, dobljena z neskončno mnogo prepogibanj, *prostor zapolnjujoča krivulja*, če na vsakem koraku dolžino papirja ustrezno podaljšamo. Če želimo, da je razdalja med začetkom in koncem *zmajeve krivulje* na vsakem koraku enaka, moramo krivuljo na vsakem koraku podaljšati za $\sqrt{2}$. Štiri *zmajeve krivulje* lahko lepo zložimo skupaj, kot prikazuje slika 7. Obstaja pa še mnogo drugih možnih prekrivanj ravnine z *zmajevimi krivuljami*.



Slika 5. Prvih pet korakov razvoja Peanove krivulje

Prvi so *zmajevo krivuljo* opisali in raziskovali fiziki v ameriški vesoljski agenciji NASA John Heighway, Bruce Banks in William Harter, širši javnosti pa jo je predstavil Martin Gardner leta 1978 v svoji popularni kolumni *Mathematical Games* v reviji Scientific American.

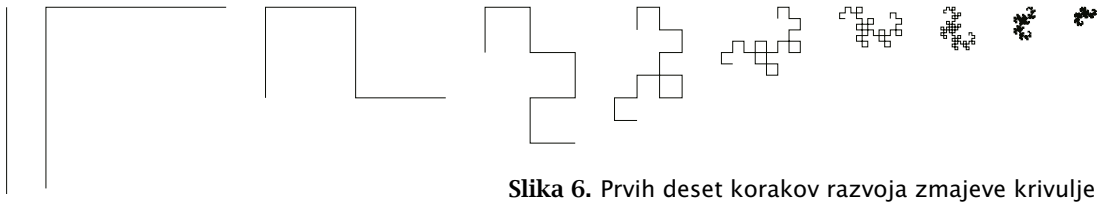
■ Preproga Sierpińskega

Začnimo z daljico (slika 8a), potem pa v vsakem naslednjem koraku daljico zamenjamo z likom (slika 8b), ki je sestavljen iz osmih za tretjino krajših daljic. Če postopek ponovimo neskončnokrat, dobimo krivuljo z imenom *preproga Sierpińskega*. *Preproga Sierpińskega* ni *prostor zapolnjujoča krivulja*. Vsota površin vseh „lukenj" v preprogi je enaka najmanjši površini kvadrata, ki jo potrebujemo, da vanj vrišemo krivuljo. Torej, sama krivulja ne prekrije nobene površine.

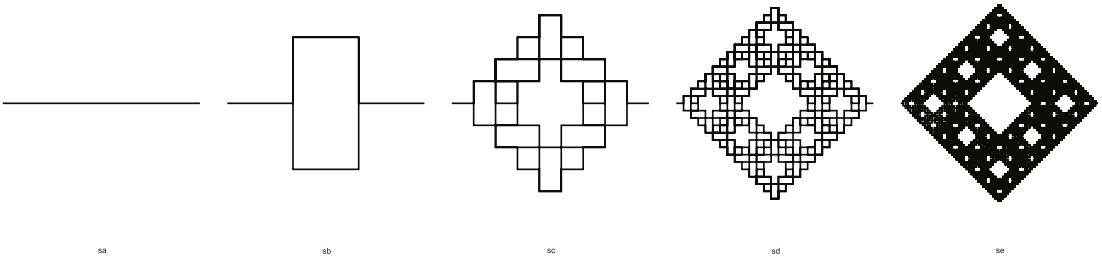
Krivuljo je prvi opisal poljski matematik Wacław Sierpiński leta 1916.

■ Hilbertova krivulja

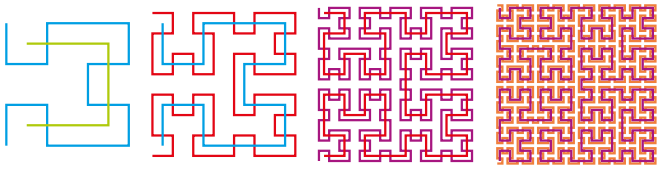
Osnovni element te krivulje je lik na sliki 9a. V prvem koraku uporabimo štiri osnovne elemente in jih povežemo med seboj s tremi daljicami, kot prikazuje slika 9b. Za vsak naslednji korak uporabimo zadnjo dobljeno sliko kot osnovni element. Krivulja, ki jo dobimo po neskončno mnogo korakih, se imenuje *Hilbertova krivulja* in je *prostor zapolnjujoča krivulja*. Krivuljo je prvi opisal nemški matematik David Hilbert leta 1891.



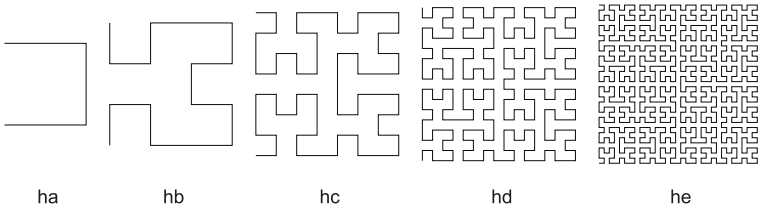
Slika 6. Prvih deset korakov razvoja zmajeve krivulje

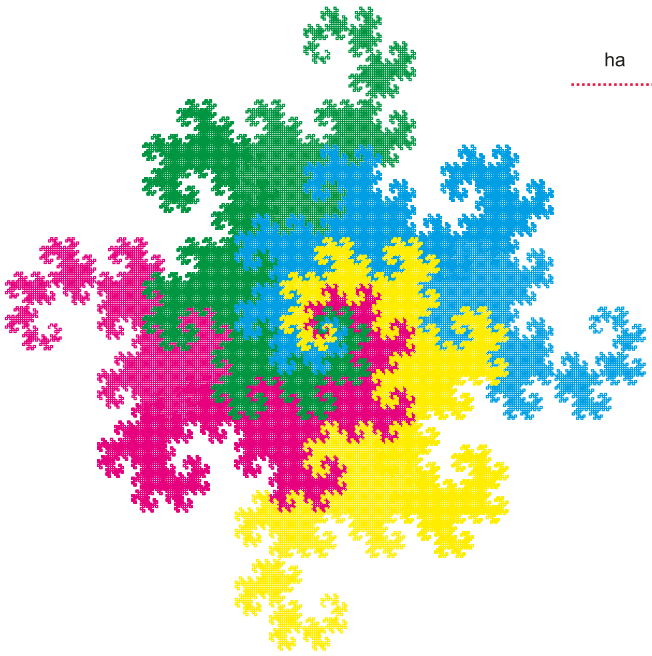


Slika 8. Prvih pet korakov razvoja preproge Sierpińskega



Slika 9. Prvih pet korakov razvoja Hilbertove krivulje





Slika 7. Štiri zmajeve krivulje lahko lepo zložimo skupaj

L-sistemi

L-sisteme je predstavil madžarski biolog Aristid Lindenmeyer leta 1968 predvsem z namenom opisovanja razvojnega procesa rastlin. Izraz L-sistemi zajema množico formalnih jezikov, vendar se ne bomo spuščali v natačno formalizacijo. Vpeljali bomo le najbolj preproste elemente takega jezika, s katerimi bomo lahko opisali vse do sedaj predstavljene krivulje.

Lik, ki ga uporabimo za začetek risanja krivulje, imenujemo *aksiom*. Postopek, kako iz prejšnjega lika dobimo naslednjega, pa opišemo s *pravili*. Naj simbol F predstavlja osnovno črto, ki jo rišemo z vnaprej določeno dolžino v smeri trenutne orientacije. Simbol + naj predstavlja zasuk trenutne orientacije za vnaprej določen kot v pozitivno smer ter simbol - v negativno smer. Aksiom predstavlja niz, s katerim začnemo. Na prvem koraku aksiom pretvorimo v

nov niz po danih pravilih. Za vsak naslednji korak moramo niz iz prejšnjega koraka prepisati v nov niz po danih pravilih. Niz, ki ga dobimo na tak način, lahko interpretiramo kot natančno navodilo za risanje krivulje. Simboli, ki nimajo grafične interpretacije, služijo le uporabi pravil. Pravila za generiranje predstavljenih krivulj so zbrana v tabeli 1.

IME KRIVULJE	AKSIOM	PRAVILA	KOT
Kochova snežinka	F++F++F	F=F-F++F-F	60
Preproga Sierpińskega	F	F=F+F-F-FF-F-FF-F-F+F	90
Zmajeva krivulja	FX	X=X+YF+, Y=-FX-Y	90
Peanova krivulja	F	F=F+F-F-F-F+F+F+F-F	90
Hilbertova krivulja	L	L=+RF-LFL-FR+, R=-LF+RFR+FL-	90
Vejica	X	X=F-[[X]+X]+F[+FX]-X), F=FF	25

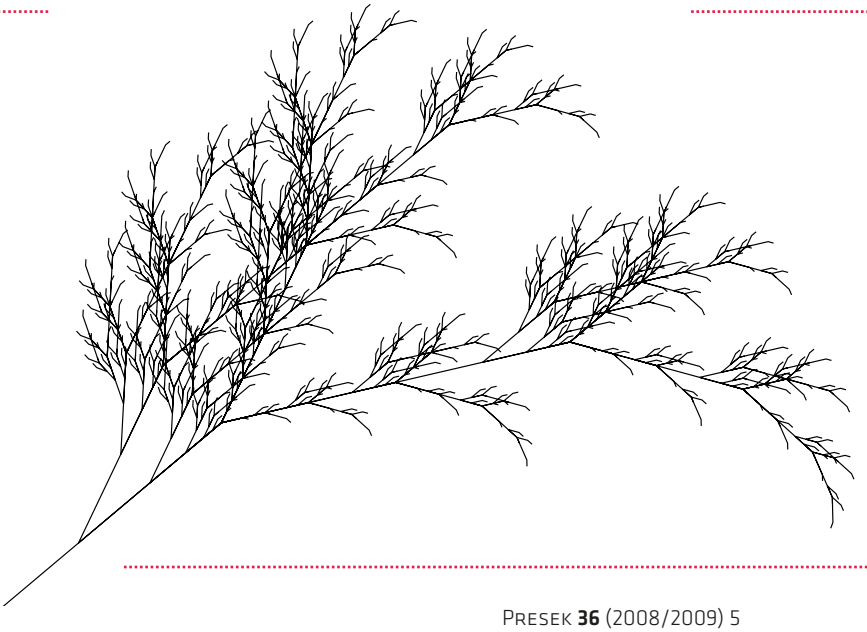
Tabela 1. Krivulje opisane v notaciji *L-sistema*

Spodaj navajamo preprost program, napisan v programskem jeziku *Java*, s katerim lahko rišemo vse krivulje iz tabele 1 razen *vejice*. Jedro programa sestavljata metodi generateCurve in paintComponent. Metoda generateCurve ustvari niz curve. Konstante na začetku programa, axiom, rules, generations, segmentLength in angle predstavljajo vhodne podatke metodi generateCurve. Metoda paintComponent interpretira niz curve in riše krivuljo na grafično matriko g2d.

Z le nekaj truda lahko program dopolnimo tako, da bo upošteval tudi simbola [in], ki sta potrebna za definicijo *vejice*. Simbol [naj pomeni shranitev trenutnega stanja (položaja, orientacije, velikosti črte, barve, ...), simbol] pa naj pomeni obnovitev predhodno shranjenega stanja. S pomočjo simbolov [in] je na sliki 10 narisana krivulja *vejica* na šestem koraku razvoja.

Program lahko postopoma razširjamo še z novimi simboli, ki imajo določen pomen pri risanju, ter tako bogatimo izrazno možnost našega jezika za opis krivulj. Nekaj idej, simbol f naj pomeni le premik brez risanja, simbol @f naj predstavlja spremembo velikosti črte za faktor f, simbol ! naj predstavlja obrnitev pomena simbolov + in -, simbol c{r,g,b} naj predstavlja barvo, s katero naj se od tu dalje riše (tako bi, recimo, niz c{255,0,0} predstavljal rdečo barvo izraženo z RGB komponentami). Osnovni element risanja je lahko tudi kaj drugega kot le preprosta črta.

Kdor ni vešč programiranja, lahko uporabi brezplačen program *Inkscape*, urejevalnik vektorskih grafik (http://www.inkscape.org). Podprt je tudi slovenski jezik. Z izbiro menijev *Učinki – Upodobi – Sistem L* se nam odpre okno za vnos parametrov krivulje. Program LSystem.java je dostopen preko medmrežja na povezavi
http://matematika-racunalnistvo.fnm.uni-mb.si/personal/petr/presek/LSystem.html.



Slika 10. Vejica

```
import javax.swing.*; // for JPanel, etc.
import java.awt.*;    // for Graphics, etc.

public class LSystem extends JPanel {
  private static int  FRAME_WIDTH = 800; // frame width
  private static int FRAME_HEIGHT = 500; // frame height

  private static String       axiom = "FX"; // rules for Dragon curve
  private static String[]     rules = new String[] {"X=X+YF+", "Y=-FX-Y"};
  private static int    generations = 13;
  private static int segmentLength = 2;
  private static int          angle = 90;

  private String curve;

  public LSystem() {
    curve = generateCurve(generations, axiom);
  }

  private String generateCurve(int gen, String curve) {
    if (gen==0) {
      return curve;
    } else {
      String newCurve = "";
      for (int curveSymbolIndex=0; curveSymbolIndex<curve.length(); curveSymbolIndex++) {
        boolean symbolWasReplaced = false;
        for (int ruleIndex=0; ruleIndex<rules.length; ruleIndex++) {
          if (curve.charAt(curveSymbolIndex)==rules[ruleIndex].charAt(0)) {
            newCurve += rules[ruleIndex].substring(2); // replacing symbol with a rule
            symbolWasReplaced = true;
            break;
          }
        }
        if (!symbolWasReplaced) {
          newCurve += curve.charAt(curveSymbolIndex); // copying symbol
        }
      }
      return generateCurve(gen-1, newCurve);
    }
  }

  public void paintComponent(Graphics g) {
    super.paintComponent(g); // clearing ofscreen pixmap
    Graphics2D g2d = (Graphics2D) g;
    int currX = super.getWidth()/2; // starting x position in the center
    int currY = super.getHeight()/2; // starting y position in the center
    int currAngle = 0; // starting direction
    for (int charIndex=0; charIndex<curve.length(); charIndex++) { // drawing the curve
      switch (curve.charAt(charIndex)) {
        case 'F': // drawing a line segment
          int deltaX = (int) Math.round(segmentLength*Math.cos(currAngle*Math.PI/180));
          int deltaY = (int) Math.round(segmentLength*Math.sin(currAngle*Math.PI/180));
          g2d.drawLine(currX, currY, currX+deltaX, currY+deltaY);
          currX += deltaX;
          currY += deltaY;
          break;
        case '+': // positive orientation rotating
          currAngle = (currAngle + angle) % 360;
          break;
        case '-': // negative orientation rotating
          currAngle = (currAngle - angle) % 360;
          break;
      } // end switch
    } // end for
  }

  public static void main(String[] args) {
    WindowUtilities.openInJFrame(new LSystem(), FRAME_WIDTH, FRAME_HEIGHT);
  }
}
```