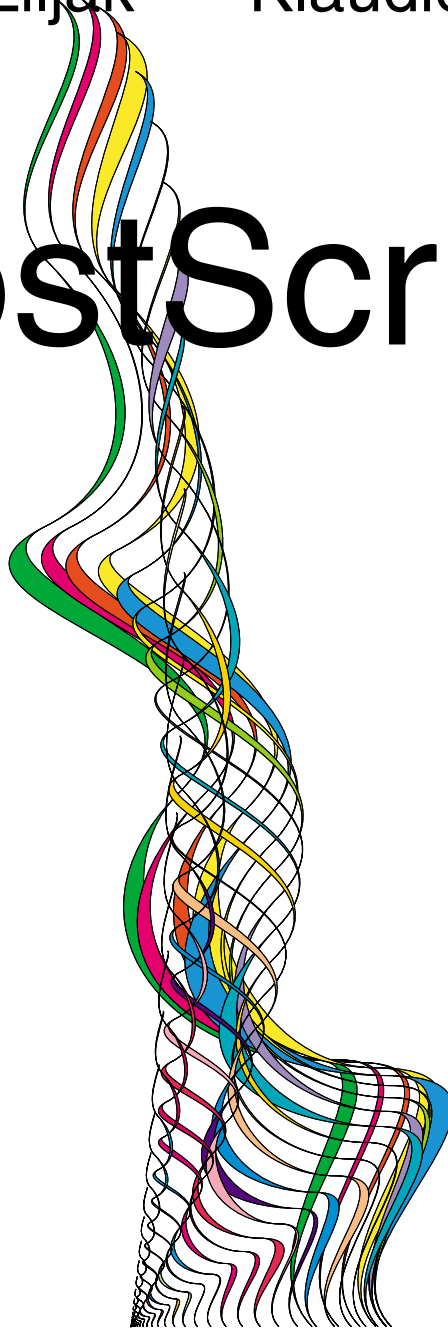


Vilko Žiljak • Klaudio Pap

# PostScript



Sadržaj:

I poglavlje

programiranje grafike ..... 1

predgovor ..... 3

PostScript ..... 4

linije, pozicioniranje; **moveto**, **lineto**, **stroke**, **showpage** ..... 6

zatvorene staze, debljina linije; **rlineto**, **closepath**, **setlinewidth** ..... 8

popunjavanje, siva površina; **fill**, **setgray** ..... 10

o završetku linija; **setlinecap** ..... 12

spajanje linija; **setlinejoin**, **setmiterlimit** ..... 14

Isprekidane linije; **setdash** ..... 17

kružni oblici; **arc** ..... 18

pomicanje ishodišta stranice; **translate** ..... 18

kružni i tangentni oblici; **arc**, **arcn**, **arcto** ..... 20

bazierova krivulja; **curveto** ..... 22

ponavljanja; **repeat** ..... 28

sačuvanje i obnavljanje grafičkog stanja; **gsave**, **grestore** ..... 30

rotacija; **rotate** ..... 30

vodoravna i vertikalna transformacija; **scale** ..... 32

višestruko popunjavanje objekata po smjeru; **eofill** ..... 34

upravljanje memorijom; **stack** ..... 36

procedure, dupliciranje, oduzimanje; **dup**, **neg** ..... 38

operacije na stacku; **index** ..... 40

matematičke operacije; **add**, **sub**, **mul**, **div**, **sgrt**, **atan** ..... 42

simetrični kontinuitet bazierove linije ..... 44

ponavljanja s indeksom; **for** ..... 46

programi mreža ..... 50

procesne boje-CMYK; **setcmykcolor** ..... 54

RGB boje; **setrgbcolor** ..... 56

HSB boje; **sethsbcolor** ..... 58

saturacija ..... 60

prelazni tonovi ..... 62

## II poglavlje

programiranje tipografije .....	65
veličina i definicija fonta; <b>findfont</b> , <b>scalefont</b> , <b>setfont</b> , <b>show</b> .....	66
indeksi u polju; <b>get</b> .....	70
prikaz ovojnice slova ili grafike; <b>charpath</b> .....	72
spajanje grafike i staze; <b>clip</b> , <b>newpath</b> .....	76
površine po stazi; <b>strokepath</b> .....	78
transformacije znakova iz fonta; <b>makefont</b> .....	80
spacioniranje i podrezivanje; <b>ashow</b> .....	82
širina teksta i brojanje slova; <b>stringwidth</b> , <b>length</b> .....	84
pozicioniranje teksta; <b>widthshow</b> , <b>xyshow</b> , <b>awidtshow</b> .....	86
o kružnim ispisima tekst; <b>kshow</b> .....	88
operatori; <b>eg</b> , <b>ne</b> , <b>gt</b> , <b>ge</b> , <b>lt</b> , <b>le</b> .....	90
logički operatori; <b>and</b> , <b>or</b> , <b>xor</b> , <b>not</b> .....	91
operatori polja; .....	92
operatori stringa; .....	93
isključivanje, poravnavanje, uređenje teksta .....	94
spajanja i insertiranja u tekstu .....	98
kontrola znakova (vokala) .....	102
čitanje i ispisivanje vanjskih tekstova; <b>file</b> , <b>readstring</b> , <b>writestring</b> , <b>closefile</b> .....	104
manipulacije sa slovima unutar riječi .....	106
dijeljenje riječi hrvatskog jezika .....	108
prijelom teksta .....	111

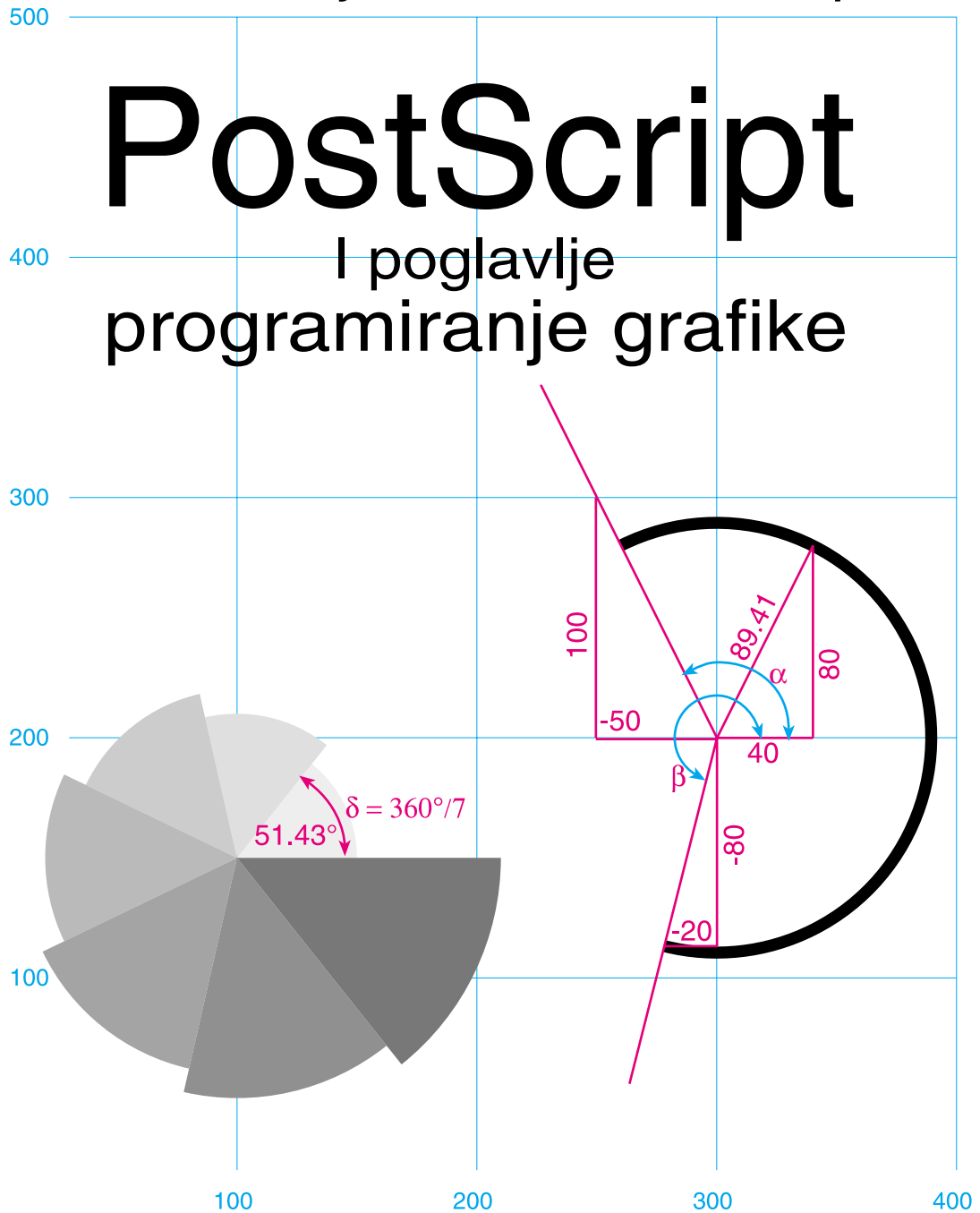
## III poglavlje

programiranje piksel grafike .....	117
slaganje piksla; <b>image</b> .....	121
definiranje veličine slike sa <b>scale</b> komandom .....	123
definicija piksla preko inverzne transformacije .....	124
bit po pikslu i razina sivoće .....	134
rezolucija .....	140
piksli u boji; <b>colorimage</b> .....	142

Vilko Žiljak • Klaudio Pap

# PostScript

I poglavlje  
programiranje grafike



## Predgovor

Ovaj digitalni udžbenik namijenjen je učenju programiranja PostScript grafike, bez obzira da li se radi o studentima, učenicima ili zaljubljenicima u računarsku grafiku i računarstvo. Obuhvaća se programiranje tipografije, boja i oblikovanje složenih grafičkih rješenja. To je digitalni udžbenik o grafičkom jeziku za opis stranice koji je podloga mnogih programa od PageMakera, QuarkXPressa, FreeHanda, CorelDrawa do programa koji su zaštićeni i primjenljivi samo na računalima za posebne namjene u području multimedije koji uključuju video, zvuk, crtež, sliku i animaciju. Namjera nam je, a i dužnost kao nastavnicima na studijima koji imaju računarsku grafiku, da ovaj digitalni udžbenik omogući čitaocu ulaz u svijet tajanstvenog PostScripta te omogući rješavanje složenih grafičkih situacija. PostScript je dio kolegija iz računarske tipografije i računarske grafike koji se predaju na studijima: Grafičkom fakultetu, studiju dizajna pri Arhitektonskom fakultetu, studiju poslovne informatike i studiju Informatičkog dizajna. Sličnu smo publikaciju, ali u tiskanom obliku, napravili prije deset godina kada smo objavili postupke dizajna PostScript fontova koji su zbog specifičnosti naših znakova u to vrijeme otvarali nepotrebne diskusije: kako, gdje, s kojim pravima, da li je moguće, što je zaštićeno i mnoga druga pitanja. PostScript je idealni alat za sve one koji imaju volju eksperimentirati u računarskoj tipografiji, grafici, slici, rasterima, i bojama. Nadamo se da ovaj digitalni udžbenik bude osvježenje i stimulativni početak aktivnog programiranja grafike u širem značenju te riječi. Primjere i način prezentacije kreirali smo onako kako smo našli najbolji odziv kod studenata za najlakše razumijevanje programiranja PostScripta.

## PostScript

Na početku želimo prikazati kako PostScript jezik za opis stranice brzo i jednostavno prikazuje grafiku, a tek u drugom koraku njegovu apstraktnu strukturu. Ne zahtjeva se da čitaoc prethodno znade bilo kakvo programiranje računala, ali da ima sklonosti prema grafičkoj umjetnosti i tipografiji. Napravili smo primjere sa objašnjenjima kako program radi, što je omogućilo baš prikazano rješenje i na što treba pripaziti da se ne realizira željena grafika. PostScript je programski jezik kao i Basic, Pascal, C te ima sličnu strukturu komandi ali ipak, namijenjen je samo području oblikovanja grafike na stranicama za tisak. PostScript ima svoj specifični grafički rječnik koji je nastao razvojem računarske tipografije, fotosloga, i računarske repro-fotografije.

Da bi se odredio grafički oblik slova ili slike koristi se komandni jezik kojim se određuje "staza" ili put te "operator" i "operandi" (parametri). Operatori, a mi ćemo ih nazivati "komande" pisane su kurentnim i verzalnim slovima na engleskom jeziku (na pr. `image`, `show`, `fill`) ili kraticama (na primjer: `arc`, `def`, `div`) izvedenim iz engleske riječi. Većina komandi je sastavljena od nekoliko engleskih riječi ili kratica ali tako da se te riječi i kratice pišu kompaktno kao jedna tipografska riječ (na pr. `setlinewidth`, `rlineto`). Parametri, ili operandi su najčešće numeričke veličine pisane ispred komandi, a određuju način djelovanje komande kao na primjer: gdje komanda počinje, završava, koliko traje, kako se prikazuje, kako djeluje. U pisanju se jednakovrijedno koriste razmak između riječi, tabulator ili kôd za novi redak, kao razdjelna oznaka između parametara i komandi. Niz komandi može se kontinuirano pisati u istom retku. Upotreba tabulatora i "carriage return" (cr) kôda, omogućuje preglednije pisanje programa. Znak postotka % određuje početak komentara koji služi programeru kao podsjetnik, ili za olakšavanje snalaženja u programu i taj komentar nema utjecaja na PostScript komande. Komentar prestaje aktiviranjem (cr) prelaza u novi redak.

Koordinatni sustav PostScripta je zasnovan u mjernim veličinama "točka" sa X/Y kordinatama, a početak je u donjem lijevom dijelu stranice. Standardna veličina točke (tako je i u našim primjerima) određena je preko inča. Inč je podijeljen u 6 dijelova nazvanih pica (pajk - engleski cicero), a pica u 12 točaka. Tako inč ima 72 točke, odnosno jedna točka je 0.353 mm. Koristili smo decimalnu podjelu pa primjeri imaju mrežu 100 x 100 točaka ili 35,3 x 35,3 mm.

Program i grafike u prvim primjerima, koje su rezultat programskih komandi i parametara prikazane su crnom bojom. Rasteri koji dočaravaju sivilo, a rezultat su programskih komandi također su prikazani crnom bojom. Da bi se bolje snalazili u položaju grafike, dodat je preko grafike koordinatni sustav u plavoj boji. Program te mreže objašnjen je tek sa rutinama ponavljanja i petlje. Komentar i neke numeričke veličine koje su nužne da bi se pojasnile komande dočrtane su crvenom bojom.

Svi primjeri se pišu u tekst editoru koji može zapisati tekst u čistoj ASCII formi, odnosno bez ikakvih tipografskih zahvata. Da bi se vidio rezultat tako pisanog PostScript programa može se poslati prema PostScript printeru sa programom za download sa bilo kojeg operativnog sustavu, ili se može vidjeti na sustavima koji podržavaju Display PostScript. Predlažemo kao najlakše rješenje rad preko programa Ghostscript koji je PostScript Level 2 interpreter za Mac, Windows, Unix, Amiga i Atari platforme, a sve informacije o njemu mogu se naći na internetu:

<ftp.cs.wisc.edu/pub/ghost/aladdin>

<http://www.cs.wisc.edu/~ghost/index.html>

Svi primjeri su originalni i planirani s namjerom da čitaocu što lakše uđe u svijet PostScripta. Digitalni udžbenik ima digitalne indeks veze i digitalne veze preko sadržaja. Čitaocu preporučamo proučavanje i čitanje od početka jer su primjeri građeni od jednostavnijih prema složenijima ili preko digitalnih veza do željenog dijela s ciljanim učenjem.

# *moveto*

# *lineto*

# *stroke*

# *showpage*

```
x y moveto
x y lineto
stroke
showpage
```

Najčešći početak PostScript stranice je `moveto` komanda. Ova komanda postavlja početak novog puta grafike u točki koja je određena parametrima pisanim ispred komande `moveto`.

```
150 50 moveto
```

Sama komanda nije prikazala neku točku na ekranu već je samo odredila početak nekog puta. Prvi primjer oblikovat ćemo tako da nacrtamo nekoliko povezanih pravaca. Dužine se povlače od točke do točke koje su određene sa obje koordinate: horizontalna i vertikalna. To omogućuje iscrtavanje istim alatom kose, vodoravne i okomite linije. Od točke koja je posljednja određena, povlači se dužina do točke koja je definirana komandom `lineto`. Prošećemo se do točaka na pozicijama 200 50 pa dalje do točaka 300 70 i 50 150.

Premda su ove komande odredile put spojivši četiri točke, ipak nisu postale vidljive. Sama linija je bezdimenzionalna dokle joj se ne pridruži debljina. Komanda `stroke` omogućuje prikazivanje linija. Sama komanda nema parametara. `Stroke` iscrtava liniju prema prije postavljenim karakteristikama za debljinu, boju, sivilo. Ako prethodno nije ništa određeno tada će se iscrtati crna linija debljine jedne točke. Grafička stranica šalje se na ispisni uređaj: ekran, pisac, fotoosvjetljivač, već prema tome kako je ispis zamišljen da se realizira. Komanda `showpage` briše postojeće stanje i postavlja parametre za ispis sljedeće stranice.

```
% primjer br. 1
```

```
150 50 moveto
```

```
200 50 lineto
```

```
300 70 lineto
```

```
50 150 lineto
```

```
stroke
```

```
% primjer br. 2
```

```
200 480 moveto
```

```
60 420 lineto
```

```
25 260 lineto
```

```
120 140 lineto
```

```
280 140 lineto
```

```
375 260 lineto
```

```
340 420 lineto
```

```
200 480 lineto
```

```
stroke
```

```
% primjer br. 3
```

```
200 400 moveto
```

```
100 300 lineto
```

```
200 200 lineto
```

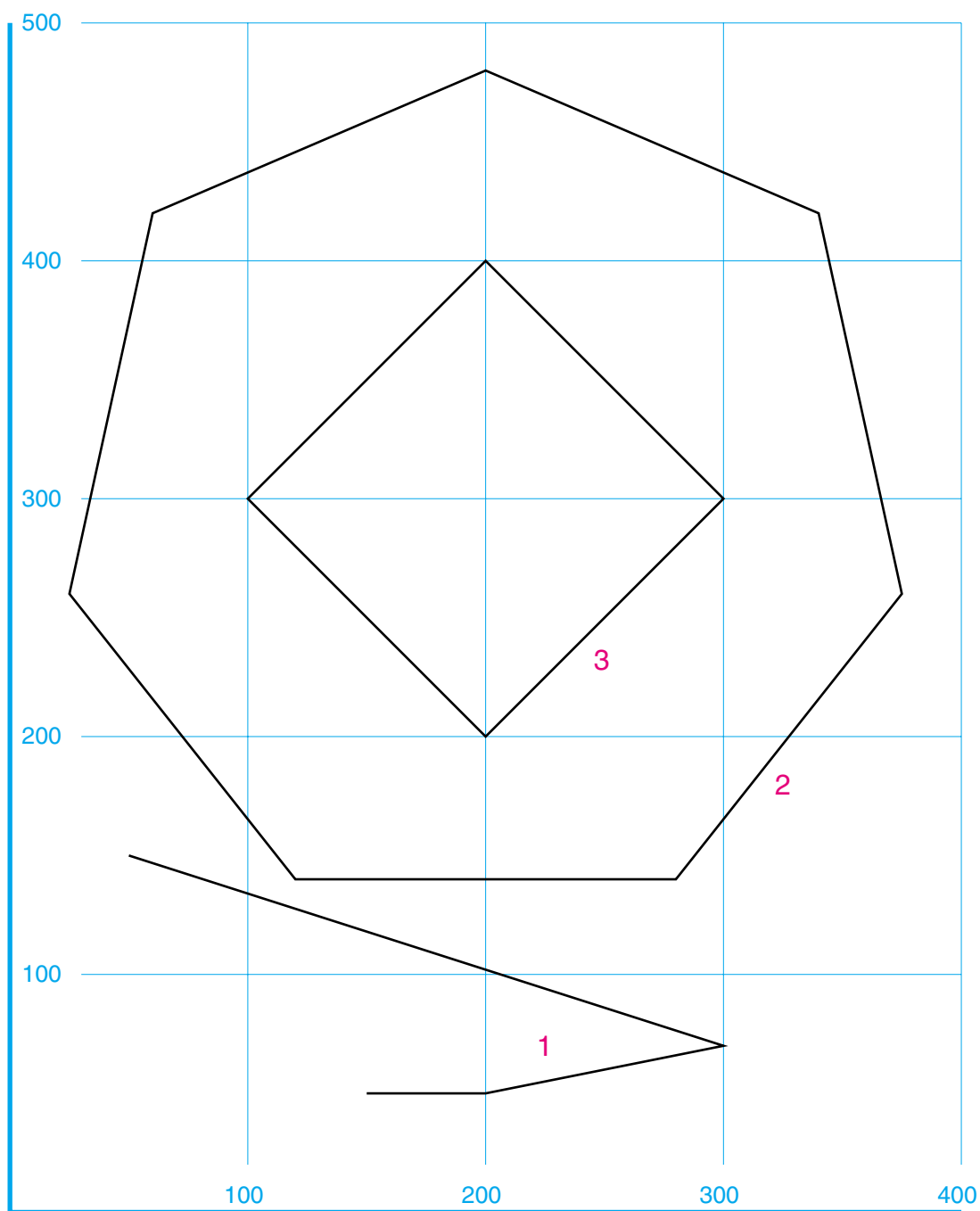
```
300 300 lineto
```

```
200 400 lineto
```

```
stroke
```

```
showpage
```





## ***rlineto***

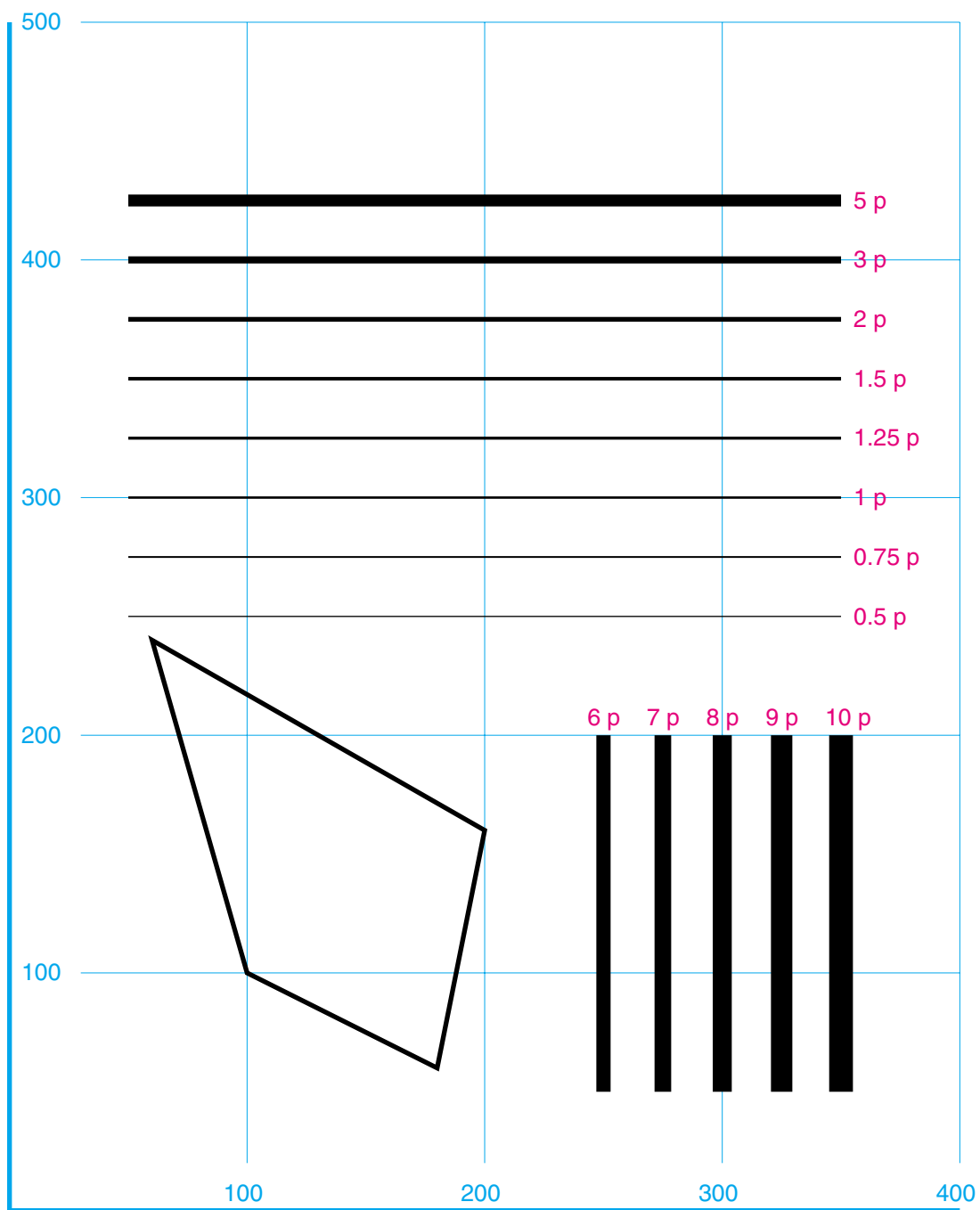
## ***closepath***

## ***setlinewidth***

x y rlineto  
closepath  
d setlinewidth

Nakon što se odredi polazna točka, pomicanje do sljedeće točke za neku udaljenost horizontalno ili vertikalno postiže se komandom `rlineto`. Ili, parametri komande `rlineto` određuju za koliko je pomak u vodoravnom ili okomitom smjeru od zadnje točke gdje smo se zaustavili. U primjeru je nacrtan četverokutni lik koji počinje u točki 100 100 te nastavlja desno za 80 točaka i dolje za 40 točaka, zatim u točku: gore za 100, desno za 20 pa u treću točku: gore za 80 i lijevo za 140 točaka. Višestruko korištenje komande `rlineto` otežava precizno računanje položaja početne točke. Trebali bi sada izračunati koliki je pomak gore ili dolje, lijevo ili desno da dođemo do početne točke. Zatvaranje lika demonstriramo komandom `closepath`. Komanda nema parametre već potraži polaznu točku u nizu prije zadnjeg pozicioniranja startne točke, na primjer točka nastala komandom `moveto`, i povuče liniju do nje. Ako debljina linije nije definirana ona je debela jednu točku kao što smo imali linije na predašnjoj stranici. Željena debljina linije određuje se komandom `setlinewidth`. U primjerima horizontalnih linija demonstriraju se debljine linija od 0.5 do 5 točaka, a u vertikalnim linijama su linije debljine od 6 do deset točaka.

50 250 moveto 300 0 rlineto 0.5 setlinewidth stroke	100 100 moveto 80 -40 rlineto 20 100 rlineto -140 80 rlineto closepath 2 setlinewidth stroke
50 275 moveto 300 0 rlineto 0.75 setlinewidth stroke	250 50 moveto 0 150 rlineto 6 setlinewidth stroke
50 300 moveto 300 0 rlineto 1 setlinewidth stroke	275 50 moveto 0 150 rlineto 7 setlinewidth stroke
50 325 moveto 300 0 rlineto 1.25 setlinewidth stroke	300 50 moveto 0 150 rlineto 8 setlinewidth stroke
50 350 moveto 300 0 rlineto 1.5 setlinewidth stroke	325 50 moveto 0 150 rlineto 9 setlinewidth stroke
50 375 moveto 300 0 rlineto 2 setlinewidth stroke	350 50 moveto 0 150 rlineto 10 setlinewidth stroke showpage
50 400 moveto 300 0 rlineto 3 setlinewidth stroke	
50 425 moveto 300 0 rlineto 5 setlinewidth stroke	



## *fill* *setgray*

`fill`  
`s setgray`

Komanda `fill` omogućuje popunjavanje i bojanje likova. Bojanje linija i zatvorenih površina, u željenim razinama sivog, postiže se komandom `setgray`. Komanda `setgray` postavlja vrijednost svjetline sivog tona. Ima jedan parametar koji određuje inverznost sive: 1 označuje suprotno od prirodnog tiska, tj. potpuno svjetlo odnosno bijelo. Vrijednost parametra 0.9 određuje 90% svjetli ton a vrijednost nula je isčezavanje svjetline tj. lik će se prikazati u crnom tonu. Linijama su demonstrirane svjetline određene komandom `setgray`.

U primjerima četverokuta date su kombinacije različitih situacija prekrivanja površina. Redosljed pisanja komandi u ovim primjerima određuje redosljed prekrivanja površina. Prva je površina siva, druga crna a treće bijela. Četvrta površina napisana je na kraju programa i ona kao bijela prekriva dio crne površine i dvije sive vertikalne linije pa ove izgledaju kao da imaju prekid. Druga i treća površina nisu završile sa komandom `closepath` već samo komandom `fill`.

```
0.5 setgray
100 100 moveto
80 -40 rlineto
50 100 rlineto
-160 80 rlineto
closepath fill

0 setgray
140 100 moveto
60 40 rlineto
20 140 rlineto
-140 -30 rlineto fill

1 setgray
140 130 moveto
20 0 rlineto
0 20 rlineto
-100 50 rlineto fill

8 setlinewidth
50 300 moveto 0 200 rlineto
0 setgray stroke

100 300 moveto 0 200 rlineto
0.1 setgray stroke

150 300 moveto 0 200 rlineto
0.2 setgray stroke

200 300 moveto 0 200 rlineto
0.3 setgray stroke

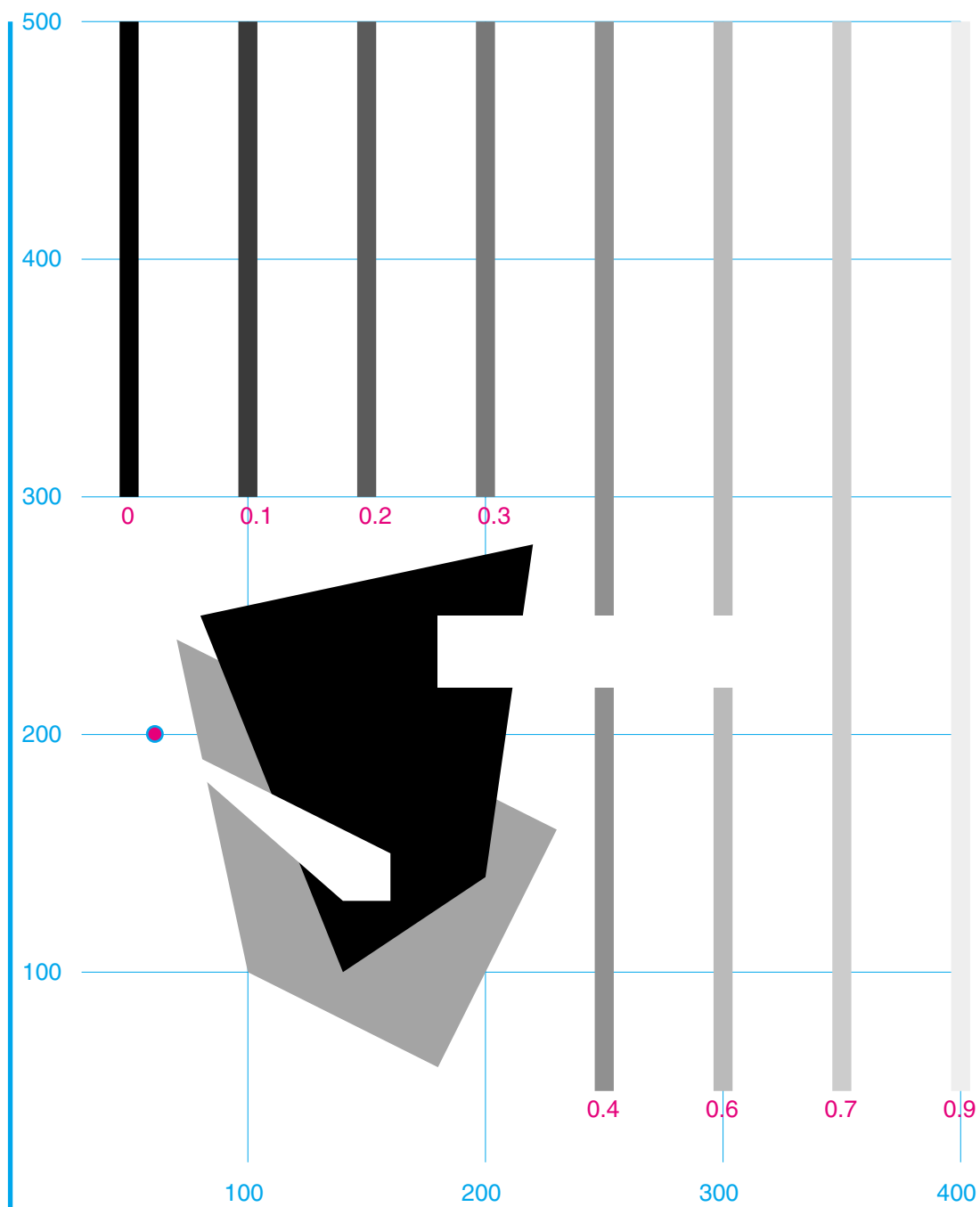
250 50 moveto 0 450 rlineto
0.4 setgray stroke

300 50 moveto 0 450 rlineto
0.6 setgray stroke

350 50 moveto 0 450 rlineto
0.7 setgray stroke

400 50 moveto 0 450 rlineto
0.9 setgray stroke

1 setgray
180 220 moveto 320 220 lineto
320 250 lineto 180 250 lineto
closepath fill
showpage
```



## *setlinecap*

**p** `setlinecap`

Linije se iscrtavaju komandom `stroke`. Njihova debljina se određuje komandom `setlinewidth`. Početni i završni oblik linije, što postaje važno kod debljih linija, može se oblikovati komandom `setlinecap`. Komanda ima jedan parametar koji može imati vrijednosti:

- 0 - kvadratni rub koji završava s definicijom linije,
- 1 - zaobljeni rub radijusa poludebljine linije i
- 2 - kvadratni završetak ali produžen za polovicu debljine linije.

Dok se prva dva oblika koriste kod samostojećih linija, zadnji oblik važan je u oblikovanju spoja vertikalnih i vodoravnih linija kakve spojeve najčešće imamo u slaganju tabličnih linija. U primjerima je preko linija povučena tanka bijela linija da bi se demonstrirao centar linije. U programu ispisanom na ovoj stranici izostavljene su komande za crtanje tih bijelih linija. Ostali primjeri pokazuju spajanje kosih i okomitih linija u različitim situacijama. Svaka linija koja se iscrtava sa `lineto` ili `rlineto` morala je prije imati definiran početak sa `moveto` naredbom da bi se izbjeglo automatsko spajanje koje inače PostScript ima definirano po početku (vidi komandu `setlinejoin`).

```
%primjer br.1
20 setlinewidth 0 setlinecap
50 110 moveto 130 0 rlineto
stroke
1 setlinecap
50 150 moveto 130 0 rlineto
stroke
2 setlinecap
50 190 moveto 130 0 rlineto
stroke

%primjer br.2
0 setlinecap
300 200 moveto 300 300 lineto
300 300 moveto 400 300 lineto
stroke

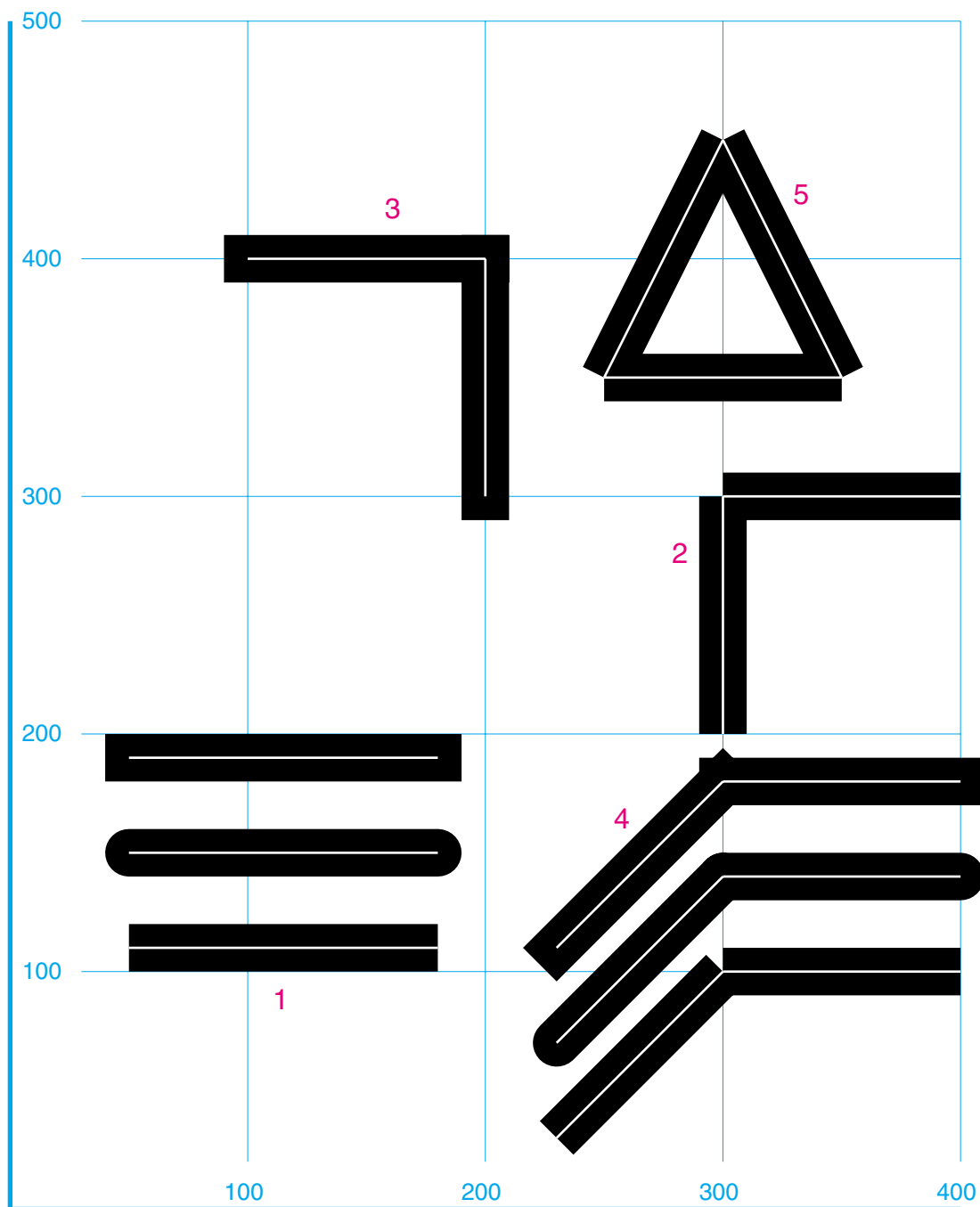
%primjer br.3
2 setlinecap
100 400 moveto 100 0 rlineto
200 400 moveto 0 -100 rlineto
stroke
```

```
%primjer br.4
0 setlinecap
230 30 moveto 300 100 lineto
300 100 moveto 400 100 lineto
stroke

1 setlinecap
230 70 moveto 300 140 lineto
300 140 moveto 400 140 lineto
stroke

2 setlinecap
230 110 moveto 300 180 lineto
300 180 moveto 400 180 lineto
stroke

%primjer br.5
0 setlinecap
250 350 moveto 300 450 lineto
300 450 moveto 350 350 lineto
350 350 moveto 250 350 lineto
stroke showpage
```



## *setlinejoin*

## *setmiterlimit*

p setlinejoin  
p setmiterlimit

Način spajanja linija rješava se komandom `setlinejoin`. Komanda ima jedan parametar koji može poprimit 3 vrijednosti:

0 - linije se spajaju tako da se njihova slika produžuje do tvorbe šiljastog vrha.

1 - dvije linije zatvaraju se kružnim oblikom čiji dijametar je jednak debljini linije i

2 - vrh spoja dviju linija je okomit na simetralu kuta spajanja, a dužina je jednaka debljini linije (tupo spajanje).

Početno stanje u PostScriptu za način spajanja je 0 tj. šiljasto spajanje i zbog toga u prvom primjeru gdje se prikazuju načini spajanja na početku nema komande `0 setlinejoin` za šiljasti način jer se ona podrazumijeva.

`Miterlimit` je maksimalni dozvoljeni omjer između dužine dijagonalne linije i debljine linije u šiljastom spoju što ovisi o kutu spajanja. On određuje kada će se šiljasto spajanje pretvoriti u tupo. U drugom se primjeru za zadani kut spajanja tek sa parametrom `3` komande `setmiterlimit` prekinulo formiranje šiljastog vrha. To znači da će se za kuteve za koje je dijagonala spoja tri i više puta veća od debljine linije primjeniti tupo spajanje, a za manje šiljasto spajanje. Duljina dijagonalne linije se dobije iz omjera debljine linije i sinusa polovice kuta između linija u spoju.

```
%primjer br.1
20 setlinewidth
200 250 moveto 100 250 lineto
200 320 lineto stroke
```

```
1 setlinejoin
200 150 moveto 100 150 lineto
200 220 lineto stroke
```

```
2 setlinejoin
200 50 moveto 100 50 lineto
200 120 lineto stroke
```

```
%primjer br.2
0 setlinejoin
250 200 moveto 350 200 lineto
250 270 lineto stroke
```

```
3 setmiterlimit
250 100 moveto 350 100 lineto
250 170 lineto stroke
```

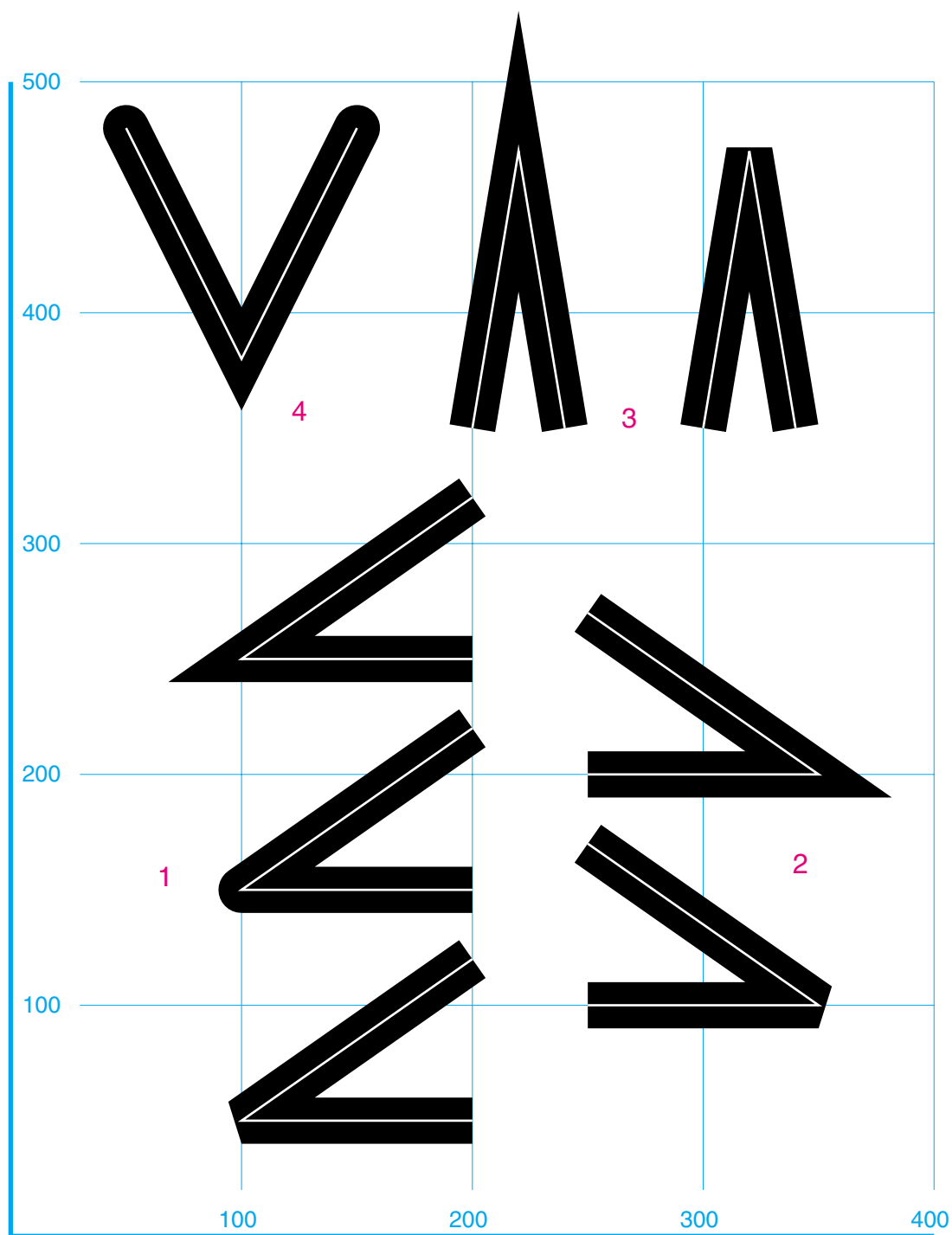
```
%primjer br.3
10 setmiterlimit
200 350 moveto 220 470 lineto
240 350 lineto stroke
```

```
6 setmiterlimit 20
setlinewidth 0 setgray
300 350 moveto 320 470 lineto
340 350 lineto stroke
```

```
%primjer br.4
10 setmiterlimit 1 setlinecap
50 480 moveto 100 380 lineto
150 480 lineto stroke
```

```
showpage
```





## *setdash*

$[C_1 B_2 \dots C_n B_n]$  *p* *setdash*

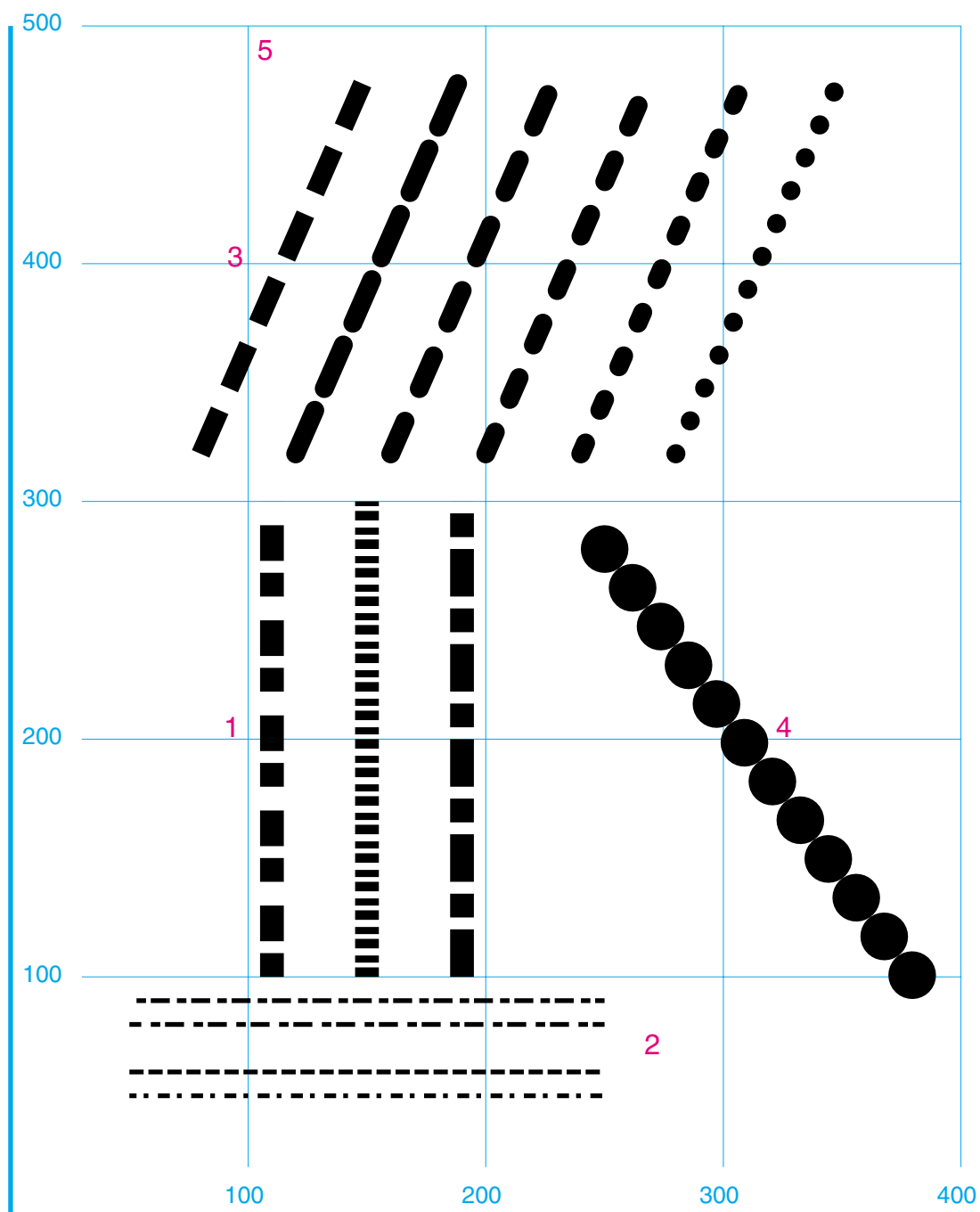
Linija se najčešće prikazuje kao puna, ali moguće ju je pretvoriti u crtkanu, točkastu ili kombinaciju kraćih i dužih crtica. Izmjenično iscrtavanje crnih i bijelih crtica postiže se komandom *setdash*.

Komandi prethodi niz brojeva u uglatoj zagradi, koji određuju elementarnu kombinaciju crne, bijele, crne i bijele crtice od kojih će se formirati isprekidana linija. Parametrom *p* se precizira skraćenje prve linije. Svi podaci dužina zadaju se u točkama. Tako na primjer komanda `[5 3 2 4] 2 setdash` određuje da će se linija, bez obzira na debljinu, sastojati od ponavljanja dviju različitih crtice i dviju različitih bjelina između crtica: prva crtica dužine 5 točaka, bjelina duljine 3 točke, crna crtica duljine 2 točke i bjelina od 4 točke. Prva crtica, ali samo u prvom javljanju, skraćena je za 2 točke.

Primjerima se demonstriraju i druge mogućnosti definiranja crtica kao naprimjer samo jedna crtica i jedna praznina što je najčešći oblik korištenja komande *setdash*. S komandom *setlinewidth* mogu se dobiti i ostale vrste isprekidanih linija kao npr. točkasta. Komanda se vraća u stanje pune linije tako da se u uglatoj zagradi ne definira niti jedan podatak.

```
10 setlinewidth %primjer br.1
[10 5 15 10] 0 setdash
110 100 moveto 110 300 lineto
stroke
[4 2 3 3] 0 setdash
150 100 moveto 150 300 lineto
stroke
[20 5 10 5] 0 setdash
190 100 moveto 190 300 lineto
stroke
2 setlinewidth %primjer br.2
[5 3 2 4] 2 setdash
50 50 moveto 250 50 lineto
stroke
[6 2] 0 setdash
50 60 moveto 250 60 lineto
stroke
[8 4 4 2] 3 setdash
50 80 moveto 250 80 lineto
stroke
[8 3 4 2] 8 setdash
50 90 moveto 250 90 lineto
stroke
```

```
8 setlinewidth %primjer br.3
[20 10] 0 setdash
80 320 moveto 150 480 lineto
stroke
[20 10] 0 setdash 1 setlinecap
120 320 moveto 190 480 lineto
stroke
[15 15] 0 setdash 1 setlinecap
160 320 moveto 230 480 lineto
stroke
[10 15] 0 setdash 1 setlinecap
200 320 moveto 270 480 lineto
stroke
[5 15] 0 setdash 1 setlinecap
240 320 moveto 310 480 lineto
stroke
[0 15] 0 setdash 1 setlinecap
280 320 moveto 350 480 lineto
stroke
20 setlinewidth %primjer br.4
[0 20] 0 setdash 1 setlinecap
250 280 moveto 380 100 lineto
stroke showpage
```



## *arc* *translate*

```
x y r L1 L2 arc
x y translate
```

Likovi: krug, kružnica, isjecci kruga i njihovi dijelovi programiraju se komandama `arc`, `arcn` i `arcto`. Komande `arc` i `arcn` imaju 5 parametara: koordinate središta i radijus zadaju se u točkama, a kut početka i kut završetka luka kruga zadaju se u stupnjevima.

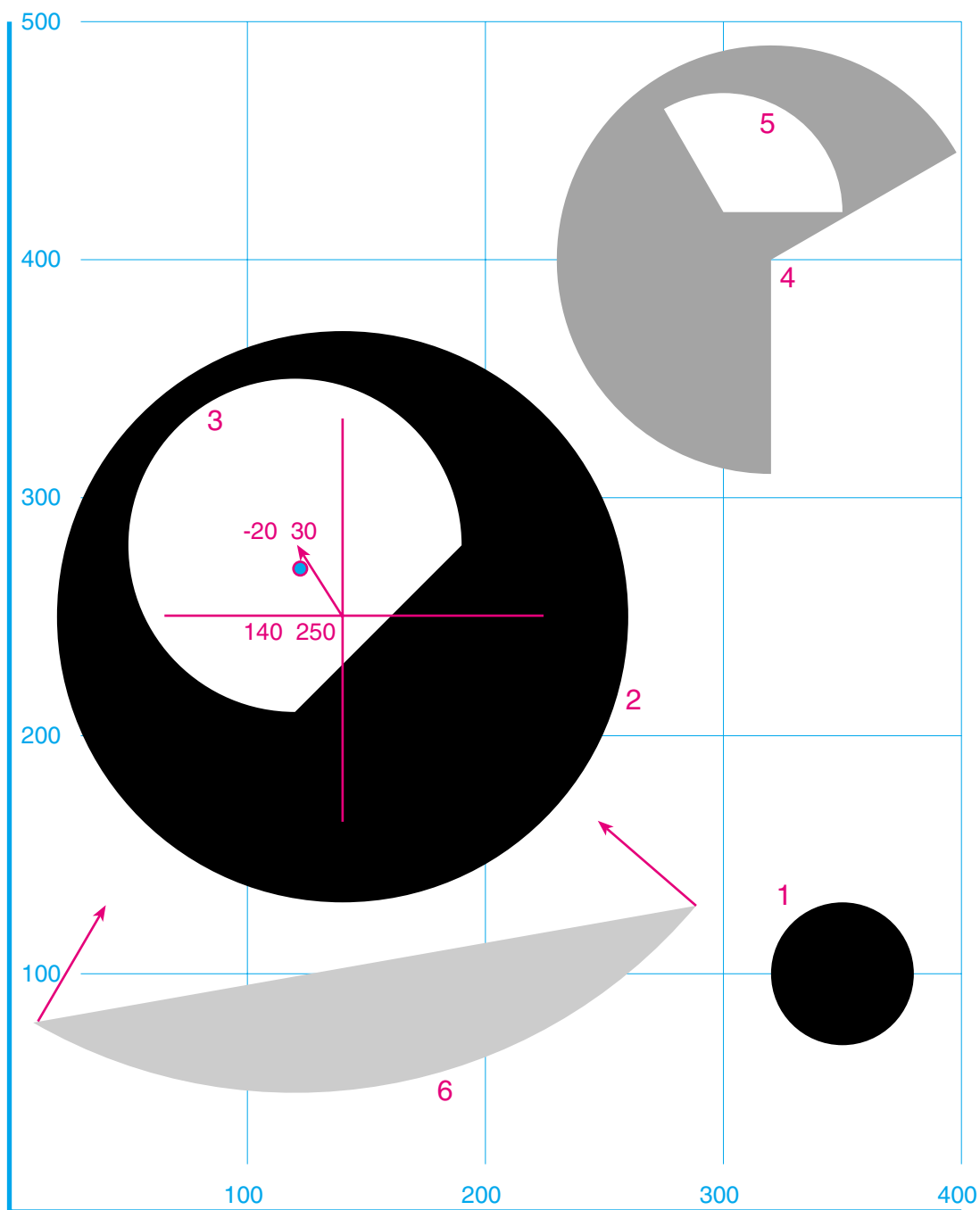
Prvi krug na primjeru nalazi se na poziciji  $x = 300$ ,  $y = 100$ ,  $r = 30$  a luk polazi od nule do 360 stupnjeva.

Drugi i treći krug pomaknuti su (translatirani) u novi koordinatni sustav. Komanda `translate` ima dva parametra: horizontalni = 140 i vertikalni = 250 točaka. Crni krug zadan je tako da mu je centar u ishodištu novog koordinatnog sustava. Treći krug s radijusom 80 točaka, odmaknut je lijevo za 20 i gore za 30 točaka od ishodišta. Luk kruga zatvara površinu od nula do 270 stupnjeva sa smjerom suprotan kazaljki na satu. Krug je popunjen bijelom bojom - najviši stupanj svjetline.

Četvrti i peti kružni oblici pomaknuti su od prethodnog ishodišta za još 250 po x osi i 100 po y osi. Lukovi ne zatvaraju polaznu i završnu točku u komandi `arc` već se završna točka luka spaja s centrom kruga pomoću komande `lineto` a cijeli oblik se zatvara sa početnom točkom komandom `closepath`.

Polumjesec, kao šesti lik, ima ishodište pomaknuto od ishodišta primjera 4 i 5 za -200 i -100 točaka. Kamanda `fill` spaja početak i kraj luka definiranog od 240 do 320 stupnjeva. Centar je u komandi određen sa (0, 0), a stvarna pozicija zbog višekratnog korištenja komande `translate` je:  $x = 140 + 180 - 200 = 120$ ,  $z = 250 + 150 - 130 = 270$  (mali kružić u polukrugu trećeg primjera).

```
% primjer br.1      180 150 translate
350 100 30 0 360 arc fill 0 0 90 30 270 arc
                    0 0 lineto closepath fill
% primjer br.2      % primjer br.5
140 250 translate   1 setgray
0 0 120 0 360 arc fill -20 20 50 0 120 arc
                    -20 20 lineto closepath fill
% primjer br.3
1 setgray           % primjer br.6
-20 30 70 0 270 arc fill -200 -130 translate
                    0.7 setgray
% primjer br.4      0 0 220 240 320 arc fill
0.5 setgray         showpage
```



***arc******arcn******arcto***

```

xy r L1 L2 arcn
x1y1 x2y2 r arcto

```

Luk u **arcn** komandi ima smjer kretanja kazaljke na satu. Lukovi pod brojem 1 koncentrično su poslagane linije od 70 do 280 stupnjeva, izvedene komandom **arc**. Isti kutevi u komandi **arcn** prikazani su primjerom broj 2. Primjer 2 pomaknut je lijevo prema dolje. Treći primjer demonstrira zatvaranje lika omeđenog s dva kružna luka gdje put unutarnjeg luka ide obrnuto kazaljke na satu, a vanjski luk ide u smjeru kazaljke na satu.

Četvrti primjer demonstrira komandu **arcto**. Komanda je određena s dvije točke i radijusom: **x1 y1 x2 y2 r arcto**. Prva točka povezuje zadnje stanje (u primjeru dato komandom **moveto** koja je na primjeru označena kao **x0 y0**), druga točka je smjer tangente kraja luka. Za različite radijuse u petom primjeru ilustriramo tangentne završetke kružnih linija za iste polazne i završne točke u komandi **arcto**.

```

gsave    %primjer br.1    200 30 70 90 0 arcn
300 150 translate          closepath
0 0 10 70 280 arc          stroke
stroke                               grestore
0 0 25 70 280 arc
stroke
0 0 40 70 280 arc
stroke
grestore

gsave    %primjer br.4
0 350 moveto 2 setlinewidth
100 400 0 550 40 arcto
stroke
grestore

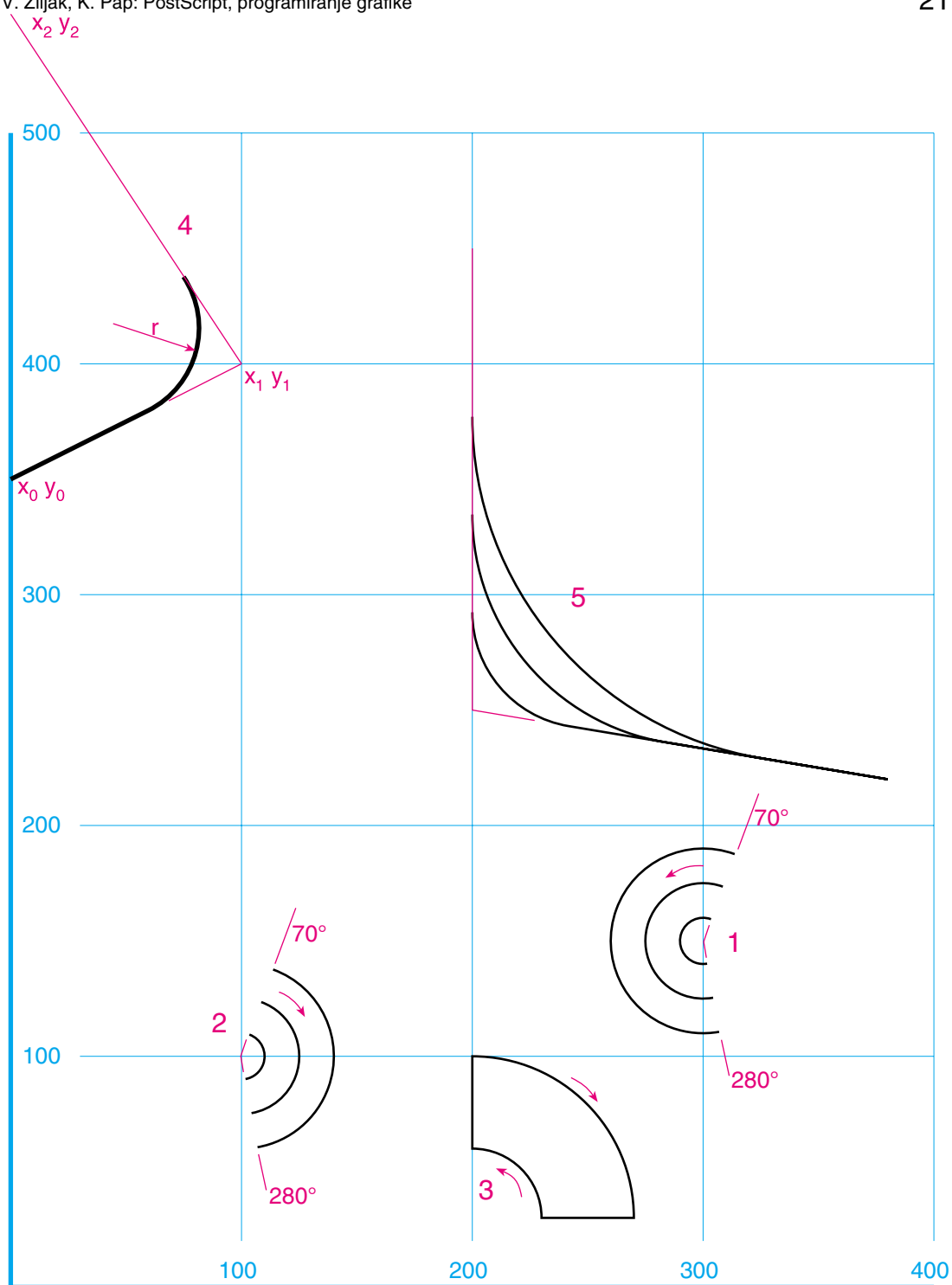
gsave    %primjer br.2
100 100 translate
0 0 10 70 280 arcn
stroke
0 0 25 70 280 arcn
stroke
0 0 40 70 280 arcn
stroke
grestore

gsave    %primjer br.5
380 220 moveto 1 setlinewidth
200 250 200 450 50 arcto
stroke
380 220 moveto
200 250 200 450 100 arcto
stroke
380 220 moveto
200 250 200 450 150 arcto
stroke
grestore

gsave    %primjer br.3
230 30 moveto
200 30 30 0 90 arc
200 100 lineto

showpage

```



## *curveto*

$x_1y_1$   $x_2y_2$   $x_3y_3$  *curveto*

Bezierova krivulja osnova je mnogih zaobljenih linija kao na primjer, ovojnica slovnih znakova. Bezierova krivulja je polinom trećeg stupnja. Definirana je s četiri točke: prva, od koje točke počinje, zadnja, gdje završava krivulja te dvije tangentne točke koje određuju smjer napredovanja linije.

Komanda *curveto* koja opisuje stazu Bezierove krivulje ima samo tri para  $x/y$  točaka. Prva Bezierova točka se ne zadaje već je to ona točka koja je zadnja ostala u upotrebi. U prvom primjeru polazna točka je određena komandom **280 300 moveto**. Krivulja **c o r v e t o** u istom primjeru ima tri točke s koordinatama: prvi par  $x_1/y_1$  točaka je vrh tangente od prve Bezierove točke (320 330), drugi par  $x_2/y_2$  (400 200) je vrh tangente zadnje Bezierove točke, a treći par  $x_3/y_3$  (280 100) su koordinate zadnje Bezierove točke.

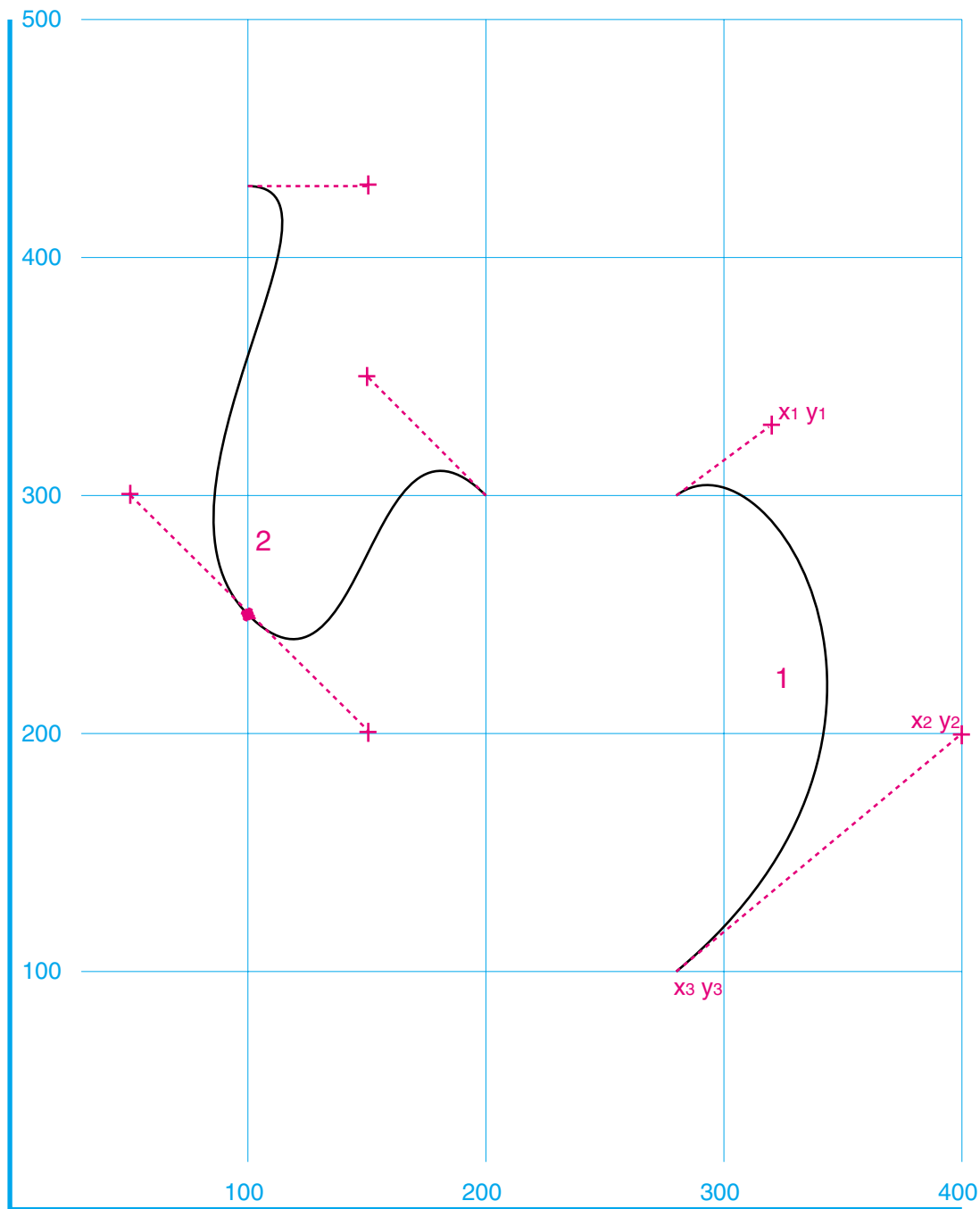
Drugi primjer demonstrira tangentno spajanje dvije Bezierove krivulje. Primjer je podignut za 200 točaka (**0 200 translate**). Spoj dvije krivulje je u točki 100 50 koja je određena kao zadnja točka prve krivulje. Druga krivulja u programu je napisana odmah poslije prve krivulje. Vrhovi tangenata u toj točki su simetrično postavljeni. Dvije Bezierove krivulje imaju glatki tangentni prelaz.

```
280 300 moveto
320 330 400 200 280 100
curveto
stroke
```

```
0 200 translate
100 230 moveto
150 230 50 100 100 50
curveto
150 0 150 200 100
curveto
stroke
```

```
showpage
```





## *curveto*

### *arcto*

U trećem primjeru, oblikovanja automobila s Bezierovim krivuljama, koristimo više komandi i povezujemo nekoliko linija. Prva grupa komandi je zatvoren grafički oblik popunjen svjetlim rasterom datim komandom `0.7 setgray` i izbijanjem: `1 setgray`. Druga grupa komandi su te iste Bezierove krivulje ali realizirane kao crne linije debljine 2 točke.

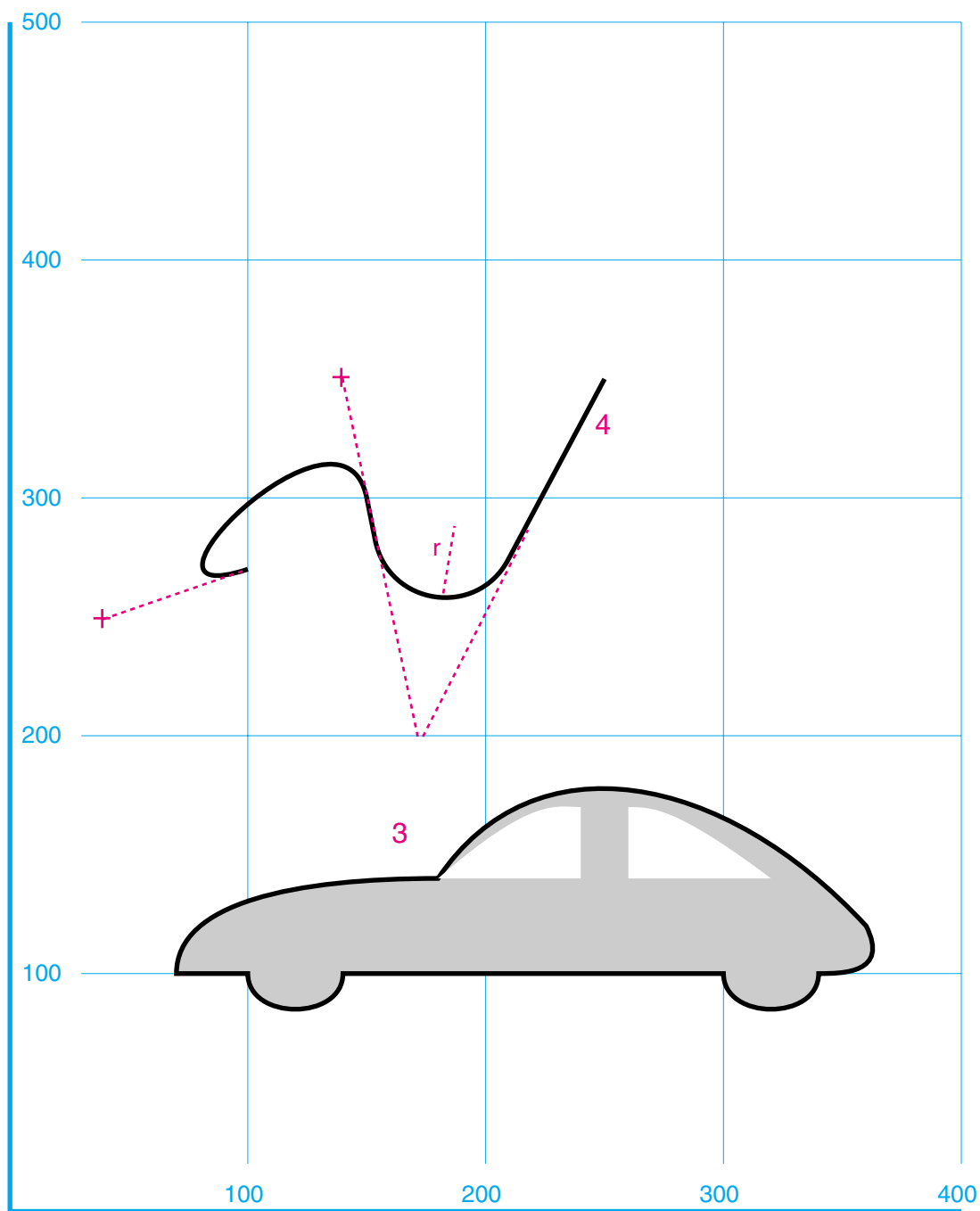
U povezivanju komandi `curveto` i `arcto` treba voditi brigu o tangentnim nastavcima kao i primjerima višestrukog povezivanja komandi `curveto`. Četvrti primjer demonstrira povezivanje zadnje Bezierove točke sa kružnom linijom. Vektor tangente kružne linije je dvostruko duži od vektora zadnje Bezierove točke ali je postavljen točno u suprotnom smjeru. Treba primjetiti da je dužina između dvije krivulje iscrtana bez da je definirana nekom programskom rutinom.

```
70 100 moveto %primjer br. 3
70 130 120 140 180 140 curveto % hauba
220 200 300 185 360 120 curveto % krov
370 100 350 100 340 100 curveto % pozadina
340 80 300 80 300 100 curveto % kotač
140 100 lineto % pod
140 80 100 80 100 100 curveto % kotač
closepath 0.7 setgray fill
```

```
70 100 moveto
70 130 120 140 180 140 curveto
220 200 300 185 360 120 curveto
370 100 350 100 340 100 curveto
340 80 300 80 300 100 curveto
140 100 lineto
140 80 100 80 100 100 curveto
closepath 0 setgray 2 setlinewidth
stroke
```

```
%1. prozor
180 140 moveto
220 175 230 170 240 170 curveto
240 140 lineto closepath
%2. prozor
260 170 moveto
270 170 280 170 320 140 curveto
260 140 lineto closepath
1 setgray fill
```

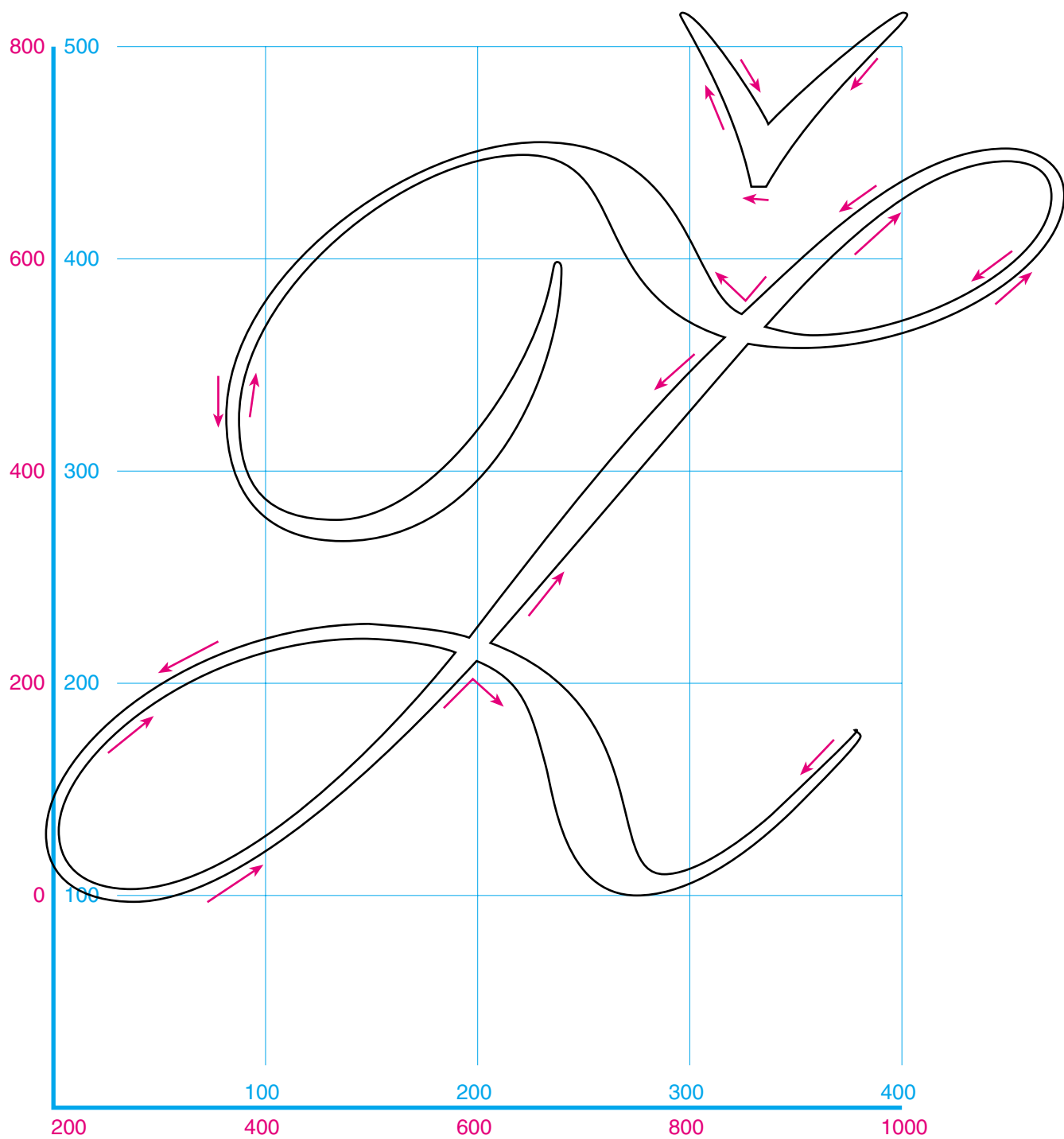
```
%primjer br. 4
0 100 translate 0 setgray
100 170 moveto
40 150 140 250 150 200 curveto
170 100 250 250 30 arcto
250 250 lineto
stroke showpage
```



PostScript tipografija je riješena Bezierovim krivuljama. Ilustriramo rukopisno slovo Ž iz fonta Shelly iz FS biblioteke. Podaci za krivulje su prepisani iz originalnog slova pomoću Fontographer-a.

Ispis slova je smanjen na polovicu sa komandom `0.5 0.5 scale`. Pismovna linija cijele tipografske biblioteke nalazi se na visini od 0 točaka. Slika slova Ž ima pomak u desno za 200 točaka. Smjer unutarnjih linija, suprotan je smjeru vanjskih linija.

```
-100 100 translate
0.5 0.5 scale
276 -6 moveto
376 -6 514 129 599 221 curveto
643 203 650 175 664 123 curveto
672 93 681 0 750 0 curveto
789 0 839 24 896 79 curveto
921 104 961 144 961 150 curveto
961 154 952 156 957 156 curveto
960 156 955 150 877 76 curveto
843 46 803 20 776 20 curveto
718 20 771 179 612 238 curveto
695 333 774 427 855 520 curveto
871 517 887 516 906 516 curveto
1027 516 1153 590 1153 660 curveto
1153 687 1130 704 1098 704 curveto
1004 704 911 605 849 548 curveto
800 568 803 710 659 710 curveto
536 710 363 584 363 451 curveto
363 380 398 334 473 334 curveto
611 334 679 496 679 590 curveto
679 594 678 597 675 597 curveto
671 597 672 590 667 570 curveto
646 483 555 354 467 354 curveto
401 354 375 384 375 449 curveto
375 570 533 698 643 698 curveto
747 698 699 570 833 526 curveto
750 448 668 342 592 243 curveto
569 251 524 254 497 256 curveto
333 254 193 141 193 58 curveto
193 15 227 -6 276 -6 curveto
%unutarnje krivulja donja
579 229 moveto
515 149 378 6 273 6 curveto
236 6 205 22 205 61 curveto
205 132 338 242 492 242 curveto
512 242 557 238 579 229 curveto
%unutarnje krivulja gornja
871 536 moveto
922 593 1013 692 1099 692 curveto
1121 692 1141 684 1141 659 curveto
1141 595 1014 528 916 528 curveto
901 528 886 532 871 536 curveto
%kvacica
872 668 moveto
872 668 858 668 858 668 curveto
842 749 791 826 791 830 curveto
791 831 792 832 793 832 curveto
810 832 867 744 874 727 curveto
918 773 992 832 1001 832 curveto
1004 832 1005 831 1005 830 curveto
1005 821 919 751 872 668 curveto
stroke
showpage
```



## *repeat*

n proc repeat

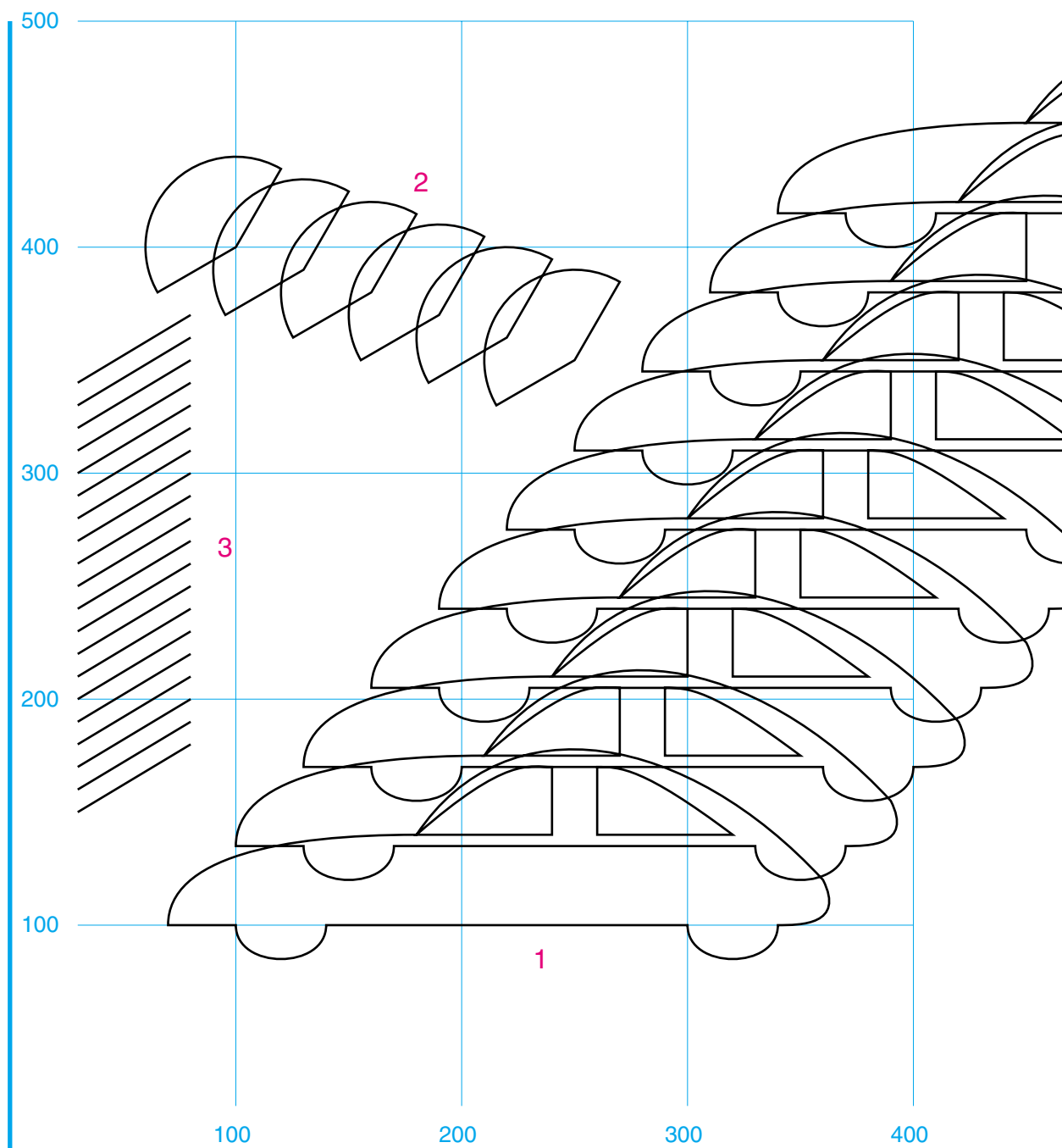
Grafike rađene pomoću računala prepune su likova s ponavljanim oblicima. Dizajneri najčešće planiraju ponavljanje crteža po nekom pravcu, krivudavom putu, kružnici, simetriji. Komande **repeat** i **for** imaju u PostScriptu slična pravila kao i u drugim programskim jezicima. U dosadašnjim primjerima imali smo ponavljanje likova upisujući podatke položaja, pomaka i multipliciranjem komandi. Ali, ako nam je namjera da višestruko i precizno oblikujemo lik u pravilnom pomaku tada je računalo idealno da samo izračuna te korake. Automobil ponavljamo 10 puta sa stalnim pomakom za 30 vodoravnih i 35 vertikalnih točaka. Komanda **repeat** počinje brojem planiranih ponavljanja ispred vitičaste zagrade. Sama komanda piše se nakon zatvorene vitičaste zagrade. Brojač **repeat** petlje se ne može dohvatiti i iskoristiti kao varijabla unutar same petlje za razliku od **for** petlje.

Sličan primjer je i sa kružnim isječkom koji se pomiče za x 30, i y -10 točaka, 6 puta. U trećem primjeru dvadeset puta ponavljamo kosu liniju u vertikalnom smjeru odozgo prema dolje.

```
%primjer br.1
10 {
180 140 moveto
220 175 230 170 240 170 curveto
240 140 lineto closepath
260 170 moveto
270 170 280 170 320 140 curveto
260 140 lineto closepath stroke
70 100 moveto
70 130 120 140 180 140 curveto
220 200 300 185 360 120 curveto
370 100 350 100 340 100 curveto
340 80 300 80 300 100 curveto
140 100 lineto
140 80 100 80 100 100 curveto
closepath stroke
30 35 translate } repeat
```

```
%primjer br.2
-200 50 translate
6 { 0 0 moveto
0 0 40 60 210 arc 0 0 lineto
30 -10 translate stroke
} repeat
```

```
%primjer br.3
-250 0 translate
20 { 0 0 moveto
50 30 rlineto stroke
0 -10 translate
} repeat
showpage
```



## ***gsave***

## ***grestore***

## ***rotate***

`gsave`  
`grestore`  
`rotate`

Programiranje grafičkog puta sa komandama kao što su `moveto`, `lineto`, `translate` i `rotate` definira jedno grafičko stanje. U njemu je zapamćena i odabrana debljina linija, svjetlina, vrsta spajanja i sve ostale komande koje se brinu za način iscrtavanja puta. Kada se upotrebe komande `stroke` ili `fill` vrši se iscrtavanje po tim parametrima i definiranom putu. Nakon njih nestati će informacija o prethodno programiranom putu, a svi ostali parametri ostaju sačuvani.

Ako želimo zapamtiti put i način iscrtavanja za kasniju upotrebu možemo upotrebiti komandu `gsave` za spremanje tekućeg grafičkog stanja i komandu `grestore` koja obnavlja (restaurira) zadnje spremljeno grafičko stanje. Tako je moguće spremati više grafičkih stanja i restaurirati ih po redoslijedu spremanja. To se izvršava preko stacka grafičkih stanja. Komanda `gsave` sprema sve parametre i putanju na vrh stacka, a `grestore` ih skida sa stacka i čini ih opet aktuelnim. Stanje koje se skinulo sa `grestore` više ne postoji na stacku pa ukoliko ga želimo opet kasnije, moramo ga opet spremati sa `gsave`. Zbog toga je poželjno iza svakog `grestora` izvršiti i `gsave` čime slobodno možemo mijenjati tek obnovljeno stanje, a da se još uvijek možemo vratiti na početno.

Komanda `rotate` zakreće koordinatni sistem po pozitivno zadanom kutu u stupnjevima u smjeru suprotnom od sata. Ako se želi izvršiti rotacija objekta, prvo se mora zakrenuti koordinatni sistem pa tek onda izvršiti programiranje objekta u zakrenutom sistemu.

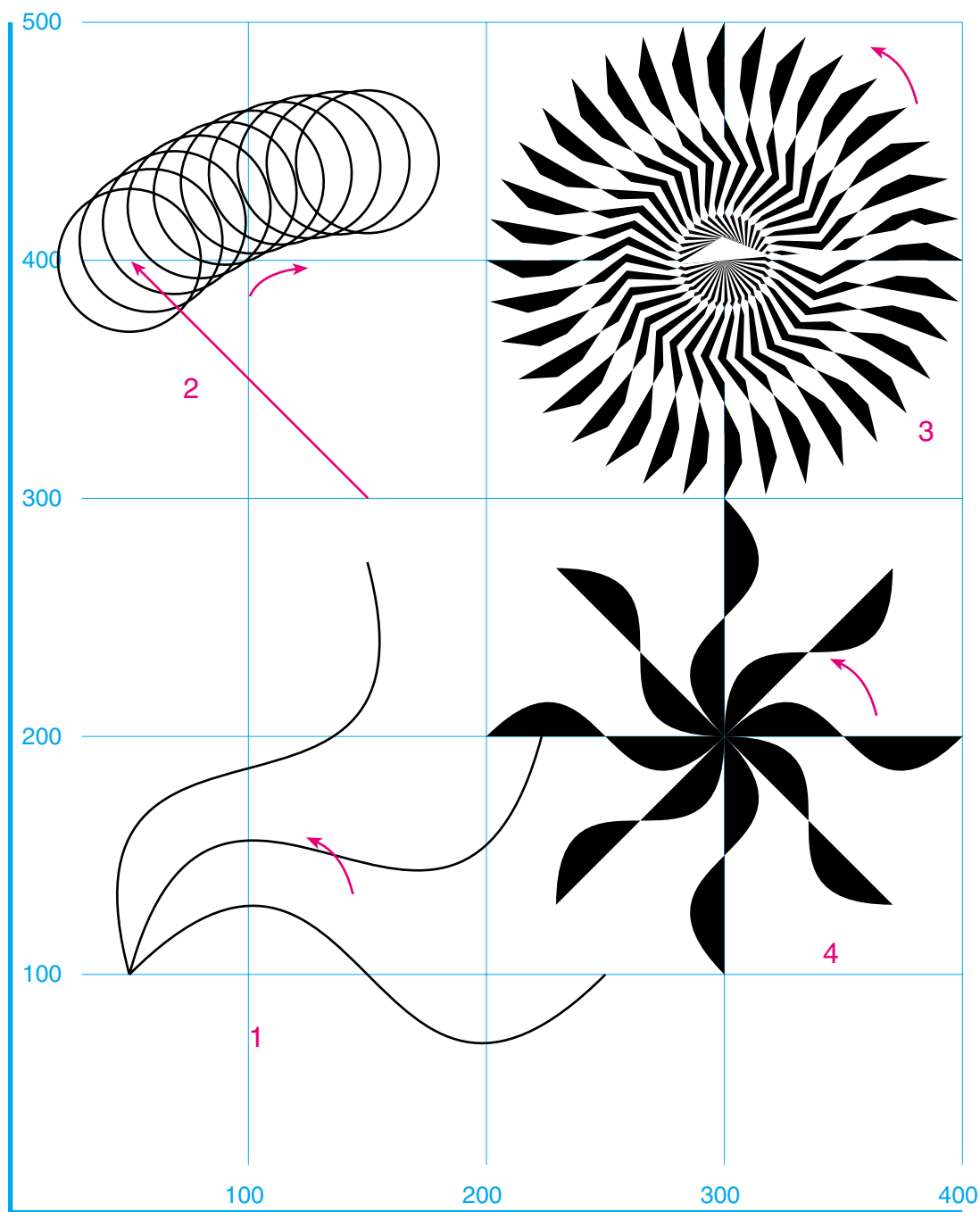
```
%primjer br.1
gsave
50 100 translate
3 { 0 0 moveto
100 100 100 -100 200 0 curveto
stroke
30 rotate } repeat
grestore

%primjer br.2
gsave
150 300 translate
10 { -100 100 30 0 360 arc
stroke
-5 rotate } repeat
grestore

%primjer br.3
gsave
300 400 translate
37 { -20 0 moveto 20 10 rlineto
40 -20 rlineto 40 20 rlineto
20 -10 rlineto closepath fill
5 rotate 1 setgray
-20 0 moveto 20 10 rlineto
40 -20 rlineto 40 20 rlineto
20 -10 rlineto closepath fill
5 rotate 0 setgray
} repeat
grestore

%primjer br.4
gsave
300 200 translate
8 { 0 0 moveto
50 50 50 -50 100 0 curveto
closepath fill
45 rotate } repeat
grestore
showpage
```





## scale

$s_x$   $s_y$  scale

```
%primjer br.1
gsave
5 setlinewidth
300 400 translate
gsave
1 1 scale
-50 -50 moveto
100 0 rlineto
0 100 rlineto
-100 0 rlineto
closepath stroke
grestore
gsave
1.5 1.5 scale
-50 -50 moveto
100 0 rlineto
0 100 rlineto
-100 0 rlineto
closepath stroke
grestore
gsave
0.5 0.5 scale
-50 -50 moveto
100 0 rlineto
0 100 rlineto
-100 0 rlineto
closepath stroke
grestore
grestore

%primjer br.2
1 setlinewidth
gsave
150 150 translate
4 { 45 rotate
1 2 scale
```

Proširivanje, sužavanje, smanjivanje, povećavanje i zrcaljenje i to proporcionalno ili neproporcionalno postiže se komandom `scale`. Njezina dva parametra  $s_x$  i  $s_y$  su faktori s kojima se množi nezavisno  $x$  i  $y$  koordinata grafičkog koordinatnog sistema pa se dobiva sužena ili proširena  $x$  ili  $y$  os. Vraćanje u prvobitno stanje koordinatnog sustava iza komande `scale` je najjednostavnije upotrebom `gsave` i `grestore` komandi. Tako imamo sačuvan referentni početak za daljnje programiranje.

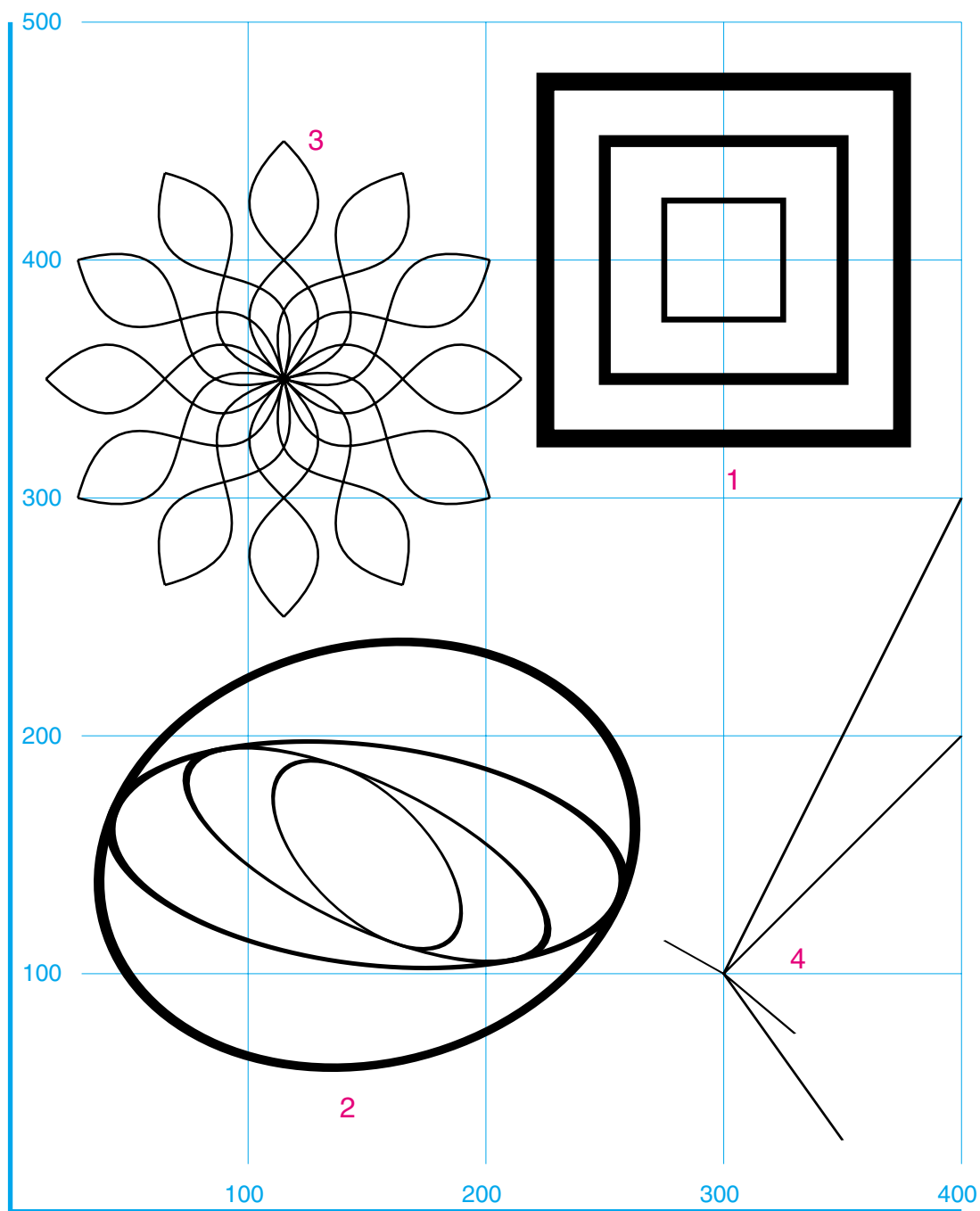
U prvom primjeru su kvadrati crtani istim komandama, ali sa istim faktorima skaliranja ( $s_x=s_y$ ). Debljina linija se također transformirala ovisno o faktorima. Drugi primjer crta elipse koje se dobivaju sa `1 2 scale` komandom koje se superponiraju u svakom krugu `repeat` petlje. Tako se je promjenio odnos  $x$  osi naprema  $y$  osi iz početne kružnice zadane komandom `arc`. U svakom krugu petlje zarotirao se koordinatni sistem i na njemu primjenio `scale` i `arc`.

Zrcaljenje objekta se vrši promjenom orijentacije koordinatnog sistema sa upotrebom negativnih faktora `scale` komande. U trećem primjeru se sa unutrašnjom `repeat` petljom s brojačem 12 iscrtalo međusobno zakrenutih 12 krivulja, a sa vanjskom `repeat` petljom s brojačem 2 se sve to još jednom ponovilo sa `-1 1 scale` komandom. Tako je dobiveno 12 listova iscrtanih od zrcalnih Bezier krivulja. U četvrtom primjeru se prikazuje igra sa linijama upotrebom različitih `scale` komandi.

```
0 0 25 0 360 arc
stroke
} repeat
grestore
```

```
%primjer br.3
1 setlinewidth
gsave
115 350 translate
2 {
12 { 0 0 moveto
50 50 50 -50 100 0 curveto
30 rotate
} repeat
-1 1 scale
} repeat stroke
grestore
```

```
%primjer br.4
4 setlinewidth
gsave
300 100 translate
1 1 scale 0 0 moveto
100 200 rlineto
1 0.5 scale 0 0 moveto
100 200 rlineto
0.5 -0.7 scale 0 0 moveto
100 200 rlineto
-0.5 -0.2 scale 0 0 moveto
100 200 rlineto
-1.2 -1.8 scale 0 0 moveto
100 200 rlineto
stroke
grestore
showpage
```



## *eofill*

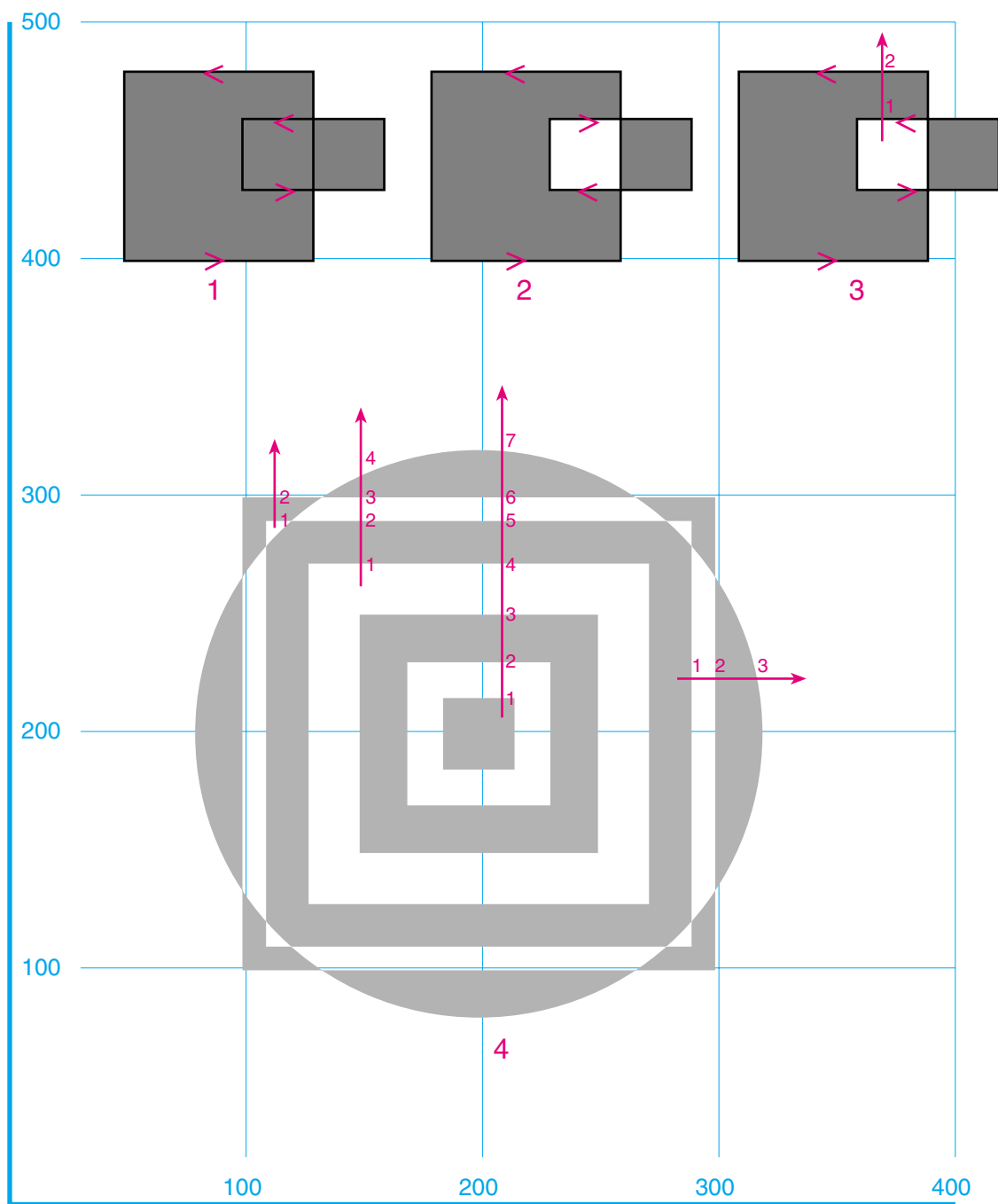
`eofill`

U PostScriptu se može na više načina definirati kako će se preklapati dva objekta i njihovo različito ponašanje nakon naredbe `fill`. Ako su obje staze objekata programirane u istom smjeru onda će i njihov presjek imati isti `setgray` iznos (primjer br.1 ). U primjeru br. 2 mali pravokutnik crtan je u suprotnom smjeru pa je dobivena "rupa" na mjestu presjeka.

Isti efekt se dobiva ako se crtaju oba objekta u istom smjeru, ali se umjesto komande `fill` upotrijebi komanda `eofill` (even-odd fill). Ona radi tako što se odluka, da li će neki presjek biti ispunjen ili ne, donosi pomoću zamišljenog polupravca od točke iz presjeka do beskonačnosti i broja prelaska preko staza objekata do vanjske okoline. Ako je taj broj paran tada taj presjek nema ispune, a ako je neparan ima ispunu. U primjeru br. 4 na slici mogu se prebrojati prelasci preko staza za nekoliko preklapanja pa je razumljiv način njihove ispune ovisno o parnom ili neparnom broju prelaska.

Primjetimo kako je u primjerima br. 1,2 i 3 upotrebljen `gsave` i `grestore` za iscertavanje linija pravokutnika čime ponovno programiranje staze za komandu `stroke` postaje suvišno. Ta mjesta su označena u programu crvenom točkom.

<code>%primjer br.1</code>	<code>80 0 rlineto</code>	<code>60 0 rlineto</code>
<code>0.5 setgray</code>	<code>0 80 rlineto</code>	<code>0 30 rlineto</code>
<code>gsave</code>	<code>-80 0 rlineto</code>	<code>-60 0 rlineto closepath</code>
<code>50 400 translate</code>	<code>closepath</code>	● <code>gsave eofill grestore</code>
<code>0 0 moveto</code>	<code>50 30 moveto</code>	<code>0 setgray</code>
<code>80 0 rlineto</code>	<code>0 30 rlineto</code>	<code>stroke</code>
<code>0 80 rlineto</code>	<code>60 0 rlineto</code>	<code>grestore</code>
<code>-80 0 rlineto</code>	<code>0 -30 rlineto</code>	
<code>closepath</code>	<code>closepath</code>	<code>%primjer br.4</code>
<code>50 30 moveto</code>	● <code>gsave fill grestore</code>	<code>0.7 setgray</code>
<code>60 0 rlineto</code>	<code>0 setgray stroke</code>	<code>gsave</code>
<code>0 30 rlineto</code>	<code>grestore</code>	<code>200 200 translate</code>
<code>-60 0 rlineto</code>		<code>0 0 moveto</code>
<code>closepath</code>	<code>%primjer br.3</code>	<code>0 0 120 0 360 arc</code>
● <code>gsave fill grestore</code>	<code>gsave</code>	<code>1 -0.1 0.4 { dup scale</code>
<code>0 setgray stroke</code>	<code>310 400 translate</code>	<code>-100 -100 moveto</code>
<code>grestore</code>	<code>0 0 moveto</code>	<code>200 0 rlineto</code>
	<code>80 0 rlineto</code>	<code>0 200 rlineto</code>
<code>%primjer br.2</code>	<code>0 80 rlineto</code>	<code>-200 0 rlineto closepath</code>
<code>gsave</code>	<code>-80 0 rlineto</code>	<code>} for eofill</code>
<code>180 400 translate</code>	<code>closepath</code>	<code>grestore</code>
<code>0 0 moveto</code>	<code>50 30 moveto</code>	<code>showpage</code>



## *stack*

U PostScriptu postoje četiri vrste stacka: stack operanada, stack riječnika, stack grafičkih stanja i izvršni stack. Stack operanada je spremnik (memorijski lanac) podataka i rezultata za skoro sve PostScript komande (operatore). Stack riječnika uspostavlja parove između imena varijabli i procedura sa njihovim sadržajem. Stack grafičkih stanja pamti programirani put i njegov način iscrtavanja. PostScript izvršava samo ono što je na vrhu izvršnog stacka koji mu služi kao radni stack. On je praktički transparentan za programera za razliku od ostalih.

PostScript jezik zasnovan je na stack procedurama i to najviše za stack operanada. Interpreter dodaje novi objekt (parametre, imena) iz programa i postavlja ga na vrh stacka gurajući prijašnje vrijednosti u stacku za jedno mjesto dublje. Kada PostScript interpreter pokrene izvršenje komande uzima se podatak sa vrha stacka i uklanja. Gledajući s pozicije programa tada program čita parametre s lijeva na desno, a komanda uzima podatke s desna na lijevo.

Na primjer za `100 200 move to` imat ćemo ove situacije na stacku:

1. `100`
2. `100 200 -> vrh stacka`
3. `-----` (prazno iza komande)

Najčešći operatori (komande) za rad sa sadržajem stacka su `exch`, `dup`, `pop`, `clear`, `roll`, `neg`, `index` i `copy`.

`exch` zamjenjuje pozicije prva dva podatka na stacku

Ako je stanje na stacku: `70 20 0 300 40`,

nakon `exch`, stanje je: `70 20 0 40 300`.

`dup` komanda duplicira podatak na vrhu stacka

Ako je stanje na stacku: `30 40 700`,

nakon `dup`, stanje je: `30 40 700 700`.

`pop` komanda briše podatak na vrhu stacka

Ako je stanje na stacku: `60 50 100`,

nakon `pop`, stanje je: `60 50`.

`clear` komanda briše sve podatke na stacku

Ako je stanje na stacku: `30 90 60`,

nakon `clear`, stanje je: `-----`.

`n p roll` komanda rotira `n` gornjih elemenata na stacku `p` puta

Ako je stanje na stacku: `70 20 0 300 40`,

nakon **4 2 roll**, stanje je: **70 300 40 20 0**.

**neg** komanda mijenja predznak podatku na vrhu stacka  
 Ako je stanje na stacku: **0 500 30 70**,  
 nakon **neg**, stanje je: **0 500 30 -70**.

**i index** komanda duplicira i-ti element na stacku i postavlja kopiju na vrh stacka, a ostale podatke gura dolje  
 Ako je stanje na stacku: **30 70 120 50 40**,  
 komanda **3 index** će napraviti: **30 70 120 50 40 70**.

**n copy** komanda duplicira n podataka sa vrha stacka  
 Ako je stanje na stacku: **50 40 30 60 80**,  
 komanda **3 copy** će napraviti: **50 40 30 60 80 30 60 80**.

Slijedeći primjeri pokazuju stanje stacka za pojedine komande po koracima izvršavanja:

<b>10 20 add</b> komanda zbrajanja	<b>-20 10 mul</b> komanda množenja
1. <b>10</b>	1. <b>-20</b>
2. <b>10 20</b> (potrebno stanje za <b>add</b> )	2. <b>-20 10</b> (potrebno stanje za <b>mul</b> )
3. <b>30</b> (rezultat)	3. <b>-200</b> (rezultat)
<b>30 50 sub</b> komanda oduzimanja	<b>10 3 div</b> komanda dijeljenja
1. <b>30</b>	1. <b>10</b>
2. <b>30 50</b> (potrebno stanje za <b>sub</b> )	2. <b>10 3</b> (potrebno stanje za <b>div</b> )
3. <b>-20</b> (rezultat)	3. <b>3.33333325</b> (rezultat)
<b>30 70 80 90 0 40 curveto</b>	<b>10 3 idiv</b> komanda za cijelobr. dijelj.
1. <b>30</b>	1. <b>10</b>
2. <b>30 70</b>	2. <b>10 3</b> (potrebno stanje za <b>div</b> )
3. <b>30 70 80</b>	3. <b>3</b> (rezultat)
4. <b>30 70 80 90</b>	<b>5 3 mod</b> daje ostatak dijeljenja 5/3
5. <b>30 70 80 90 0</b>	1. <b>5</b>
6. <b>30 70 80 90 0 40</b> ( <b>curveto</b> )	2. <b>5 3</b> (potrebno stanje za <b>mod</b> )
7. <b>-----</b> (prazno iza komande)	3. <b>2</b> (rezultat)
<b>10 -50 atan</b> komanda koja daje kut čija je tangentna točka (10, -50)	
1. <b>10</b>	
2. <b>10 -50</b> (potrebno stanje za <b>atan</b> )	
3. <b>168.690063</b> (rezultat)	
<b>10 5 add /x exch def</b> definiranje varijable x koja sadržava rezultat kom. <b>add</b>	
1. <b>10</b>	
2. <b>10 5</b>	
3. <b>15</b>	
4. <b>/x 15</b>	
5. <b>15 /x</b>	
6. <b>-----</b> (prazno iza komande <b>def</b> )	

## *dup, neg procedure (okvir)*

Procedure su skup rutina sa korisnikovim imenom a upotrebljavaju se, pozivaju, kao i bilo koja PostScript komanda. Ilustracija procedure nazvane **okvir** pokazana je primjerom iscrtavanja pravokutnika tako da se zadaje samo dijagonalna točka, točka nasuprot polaznoj točki. Budući da je time dato dovoljno podataka moguće je rješenje iscrtavanja puta od početne točke preko ostalih točaka pravokutnika. Svaki programer bi na svoj način rješio proceduru, redosljed postavljanja i uzimanja vrijednosti u stacku prije izvršenja komande **rlineto** ili **lineto**. U slijedeće dvije stranice data su dva rješenja. Na početku programa data je kratka procedura s imenom "**s**" za ispis stanja stacka i teksta "**mark**". Plavom bojom označena su mjesta ispisa stacka, a crvenom bojom veza programa sa ispisom točke na lijevoj dijelu stranice.

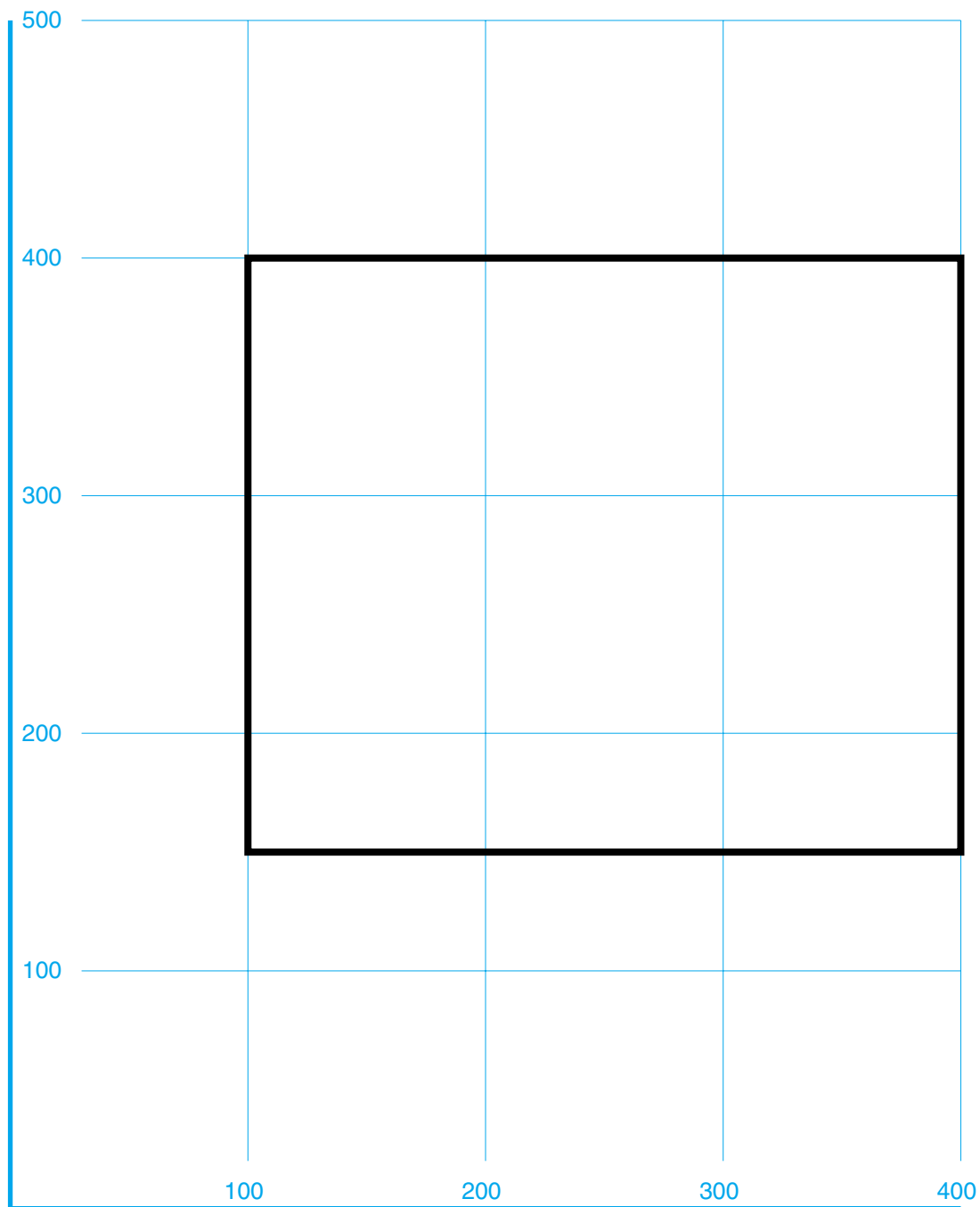
```
-mark-
100
-mark- 1
150
100
-mark-
-mark-
300
-mark- 2
250
300
-mark-
300
250
-mark-
300
300
250
-mark- 3
0
300
300
250
-mark- 4
300
250
-mark-
250
300
-mark-
0
250
300
-mark- 5
250
0
300
-mark- 6
300
-mark- 7
-300
-mark-
0
-300
-mark-
-mark- 8
-mark- 9
-mark- 10
3
-mark- 11
-mark- 12
```

```
/s { mark pstack pop } def

/okvir {
  exch s dup s 0 s 3 rlineto s 4
  exch s 0 s exch s 5 rlineto s 6
  neg s 7 0 s rlineto s
  closepath s 8
} bind def
100 s 1 150 s moveto s
300 s 250 s 2 okvir s 9
3 s 10 setlinewidth s 11 stroke s 12

showpage
```





## *Index*

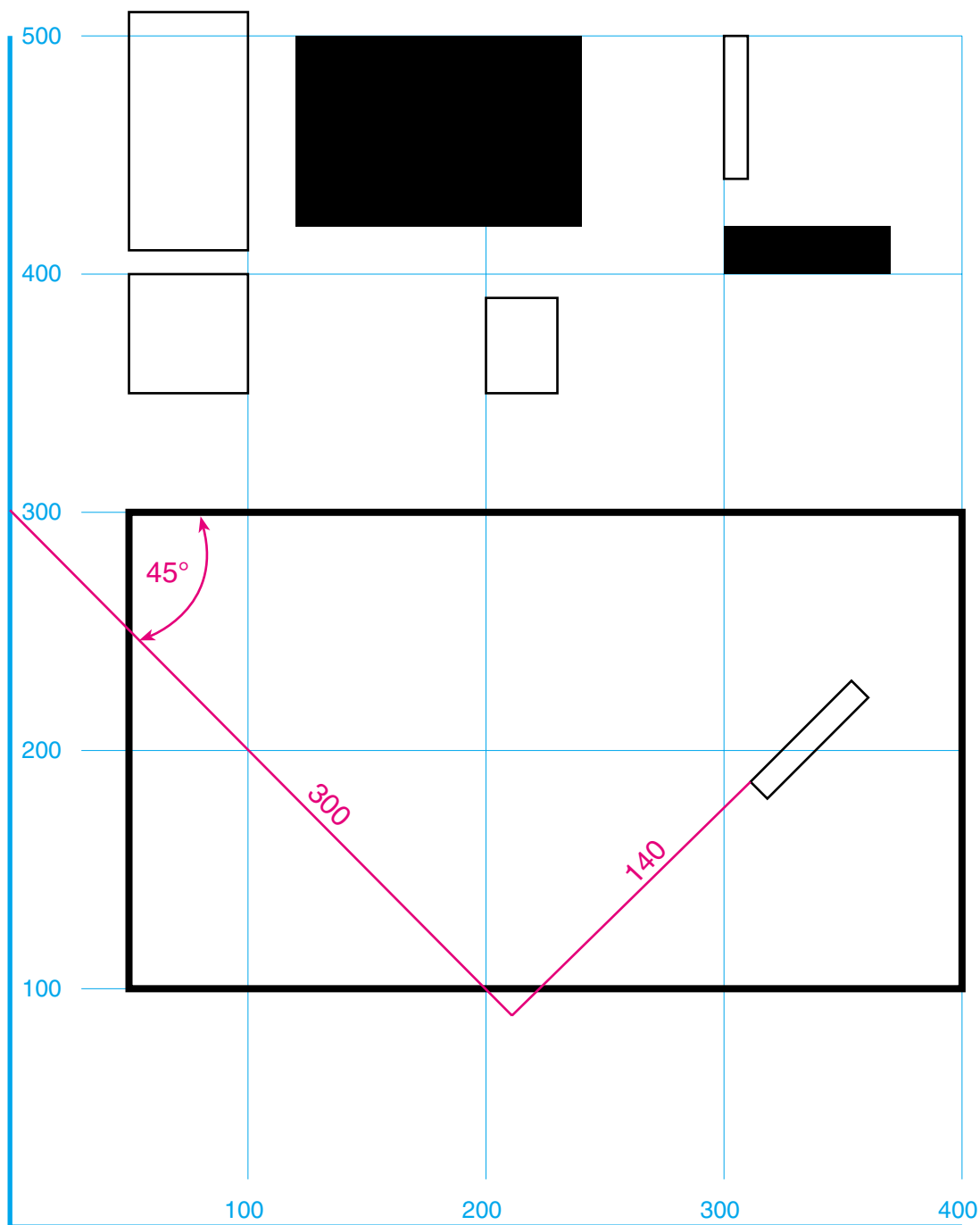
Procedura drugog rješenja okvira koristi komandu `index`. Tom se komandom duplicira i prebacuje neki podatak iz sredine stacka na vrh stacka. `Index` komanda kopira broj udaljen od vrha stacka za toliko, kolika je vrijednost (index) u prvom stacku. Taj broj stavlja na prvo mjesto, a stari broj na vrhu (index) briše se. Ilustracije radi, ispisali smo stanje stacka za sedam okvira. Zadnji okvir je rotiran za 45 stupnjeva. Podaci u stacku za taj okvir su u jedinicama novog - rotiranog koordinatnog sustava.

```
-mark-
50
-mark-
100
50
-mark- moveto
0
350
200
350
-mark- 2
200 y
0 x
350
-mark- 3
0
-350
-mark- 4
0
50
50
50
50
-mark- 2 I
50
0
50
-mark- 3 I
0
-50
-mark- 4 I
0
50
100
50
-mark- 2 II
100
0
50
-mark- 3 II
0
-50
-mark- 4 II
0
120
80
120
-mark- 2 III
80
0
120
-mark- 3 III
0
-120
-mark- 4 III
0
30
40
30
-mark- 2 IV
40
0
30
-mark- 3 IV
0
-30
-mark- 4 IV
0
70
20
70
-mark- 2 V
20
0
70
-mark- 3 V
0
-70
-mark- 4 V
0
10
60
10
-mark- 2 VI
60
0
10
-mark- 3 VI
0
-10
-mark- 4 VI
0
10
60
10
-mark- 2 VII
60
0
10
-mark- 3 VII
0
-10
4 VII
```

```
/s { mark pstack pop } def
```

```
/okvir {
1 index 0 s 2 rlineto
0 exch s 3 rlineto
neg 0 s 4 rlineto
closepath
} bind def
50 s 100 s 1 moveto
350 200 okvir
3 setlinewidth stroke

1 setlinewidth
0 300 translate
50 50 moveto 50 50 okvir stroke I
50 110 moveto 50 100 okvir stroke II
120 120 moveto 120 80 okvir fill III
200 50 moveto 30 40 okvir stroke IV
300 100 moveto 70 20 okvir fill V
300 140 moveto 10 60 okvir stroke VI
-45 rotate
300 140 moveto 10 60 okvir stroke VII
showpage
```



***add******sub******mul******div******sqrt******atan***

```

a b add
a b sub
a b mul
a b div
  a sqrt
a b atan

```

Matematičke operacije: zbrajanje, oduzimanje, množenje, dijeljenje, korjen, arkus-tanges imaju parametre slično kao i u drugim programskim jezicima. Zbrajanje uzima dva podatka s vrha stacka, zbraja ih, a rezultat stavlja na vrh stacka. Slično je i sa komandama **sub**, **mul** i **div**. U prvom primjeru, kut za sedam isječaka dobiva se dijeljenjem  $360/7$  (**360 7 div**). Za svaki isječak smanjuje se svjetlina:  $(siva - 0.1)$  za deset posto: **siva 0.1 sub**. Radijus se povećava za 10 točaka: **polumjer 10 add**. Za svih sedam isječaka ispisana su stanja stacka sive boje (crvena točka).

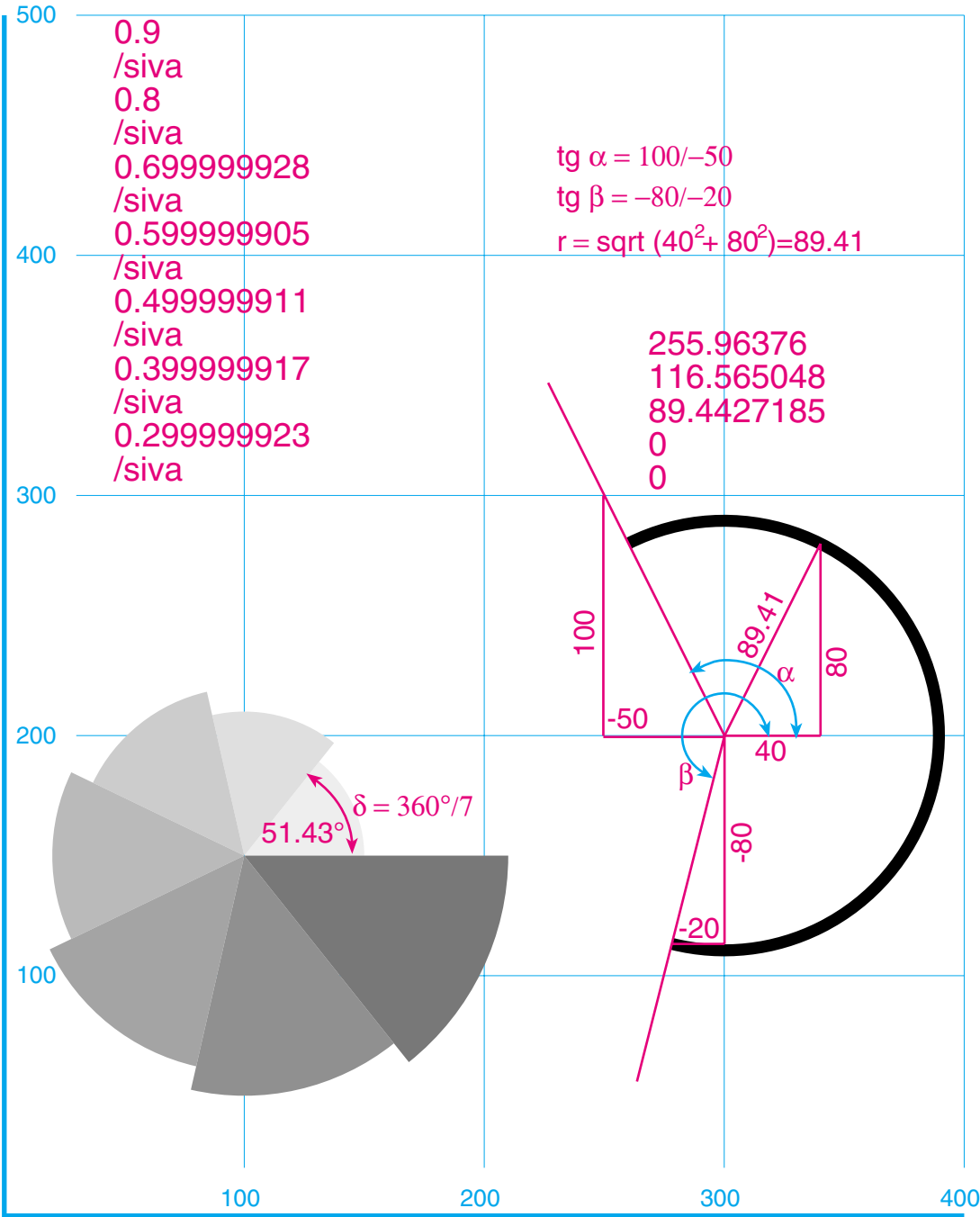
Drugi korjen se izračunava od podatka na vrhu stacka, a rezultat se stavlja na to isto mjesto. Arkus-tangens izračunava se na temelju dva podatka trigonometrijske kružnice kao što je ilustrirano na drugom primjeru. Početak luka je i drugom kvadrantu dat podacima:  $\text{atan}(100/-50) = 116,5^\circ$ . Kraj luka je u trećem kvadrantu određen podacima:  $\text{atan}(-80/-20) = 255,9^\circ$ . Radijus se izračunava iz trokuta s katetama duljina 80 i 40 točaka:  $\text{sqrt}(80^2 + 40^2)$ . Prije aktiviranja crtanja luka ispisano je stanje u stacku (crvena točka).

```

clear
/polumjer 50 def
/siva 1 def
gsave
100 150 translate
7 {
  siva 0.10 sub
  /siva exch ● def
  siva setgray
  0 0 moveto
  0 0 polumjer 0 360 7 div arc
  0 0 lineto
  polumjer 10 add
  /polumjer exch def
  closepath fill
  360 7 div rotate
} repeat
grestore

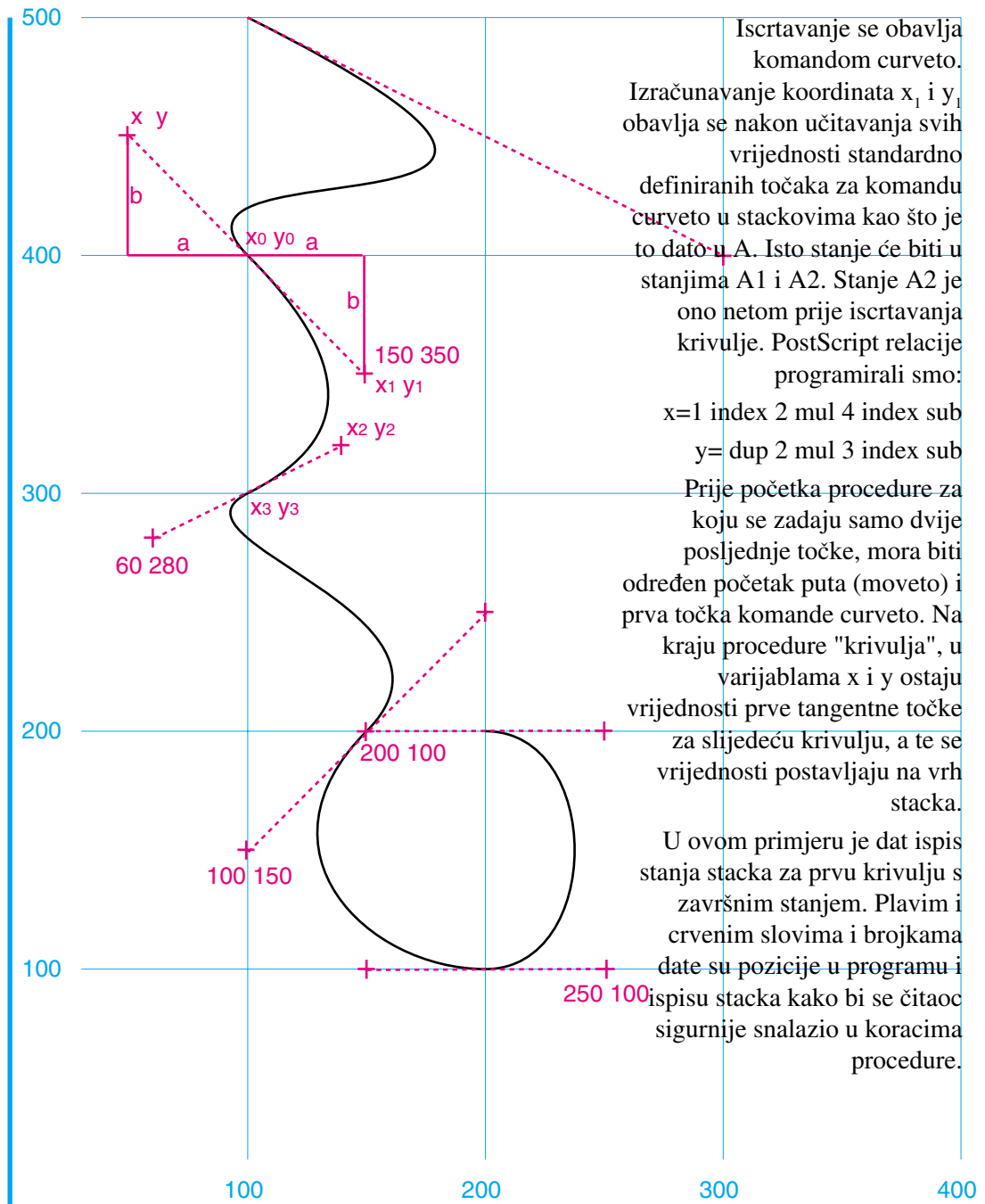
300 200 translate
0 0 40 40 mul 80 80 mul add sqrt
100 -50 atan -80 -20 atan ● arcn
5 setlinewidth
stroke
grestore showpage

```



```

- mark- /s { mark pstack pop } def
500     /krivulja {
100     dup s 1 2 s 2 mul s 3 s 3 index s 4 sub s 5
- mark- /y exch s 6 def s 7
400     1 s 8 index s 9 2 s 10 mul s 11 4 s 12 index s 13
450     A sub s 14
50      /x exch s 15 def s 16 curveto
400     x y s 17
300     } def
- mark-
400
400
100     100 500 s moveto
450     300 400 50 450 100 400 s krivulja
50      140 320 100 300 krivulja
400     200 250 150 200 krivulja
300     150 100 200 100 krivulja
- mark- 1 250 200 200 200 krivulja
2
400     stroke
400     clear
100
450     showpage
50
400
300     350
- mark- 2 400
800     100
400     450
100     50
450     400
50      300
400     -mark- 5
300     350
- mark- 3 /y
400
800     100
400     450
100     50
450     400
50      300
400     -mark- 6
300     350
- mark- 7 /y
400
800     100
400     450
100     50
450     400
50      300
400     -mark- 8
300     350
- mark- 9 /y
400
800     100
400     450
100     50
450     400
50      300
400     -mark- 10
300     350
- mark- 11 /y
400
800     100
400     450
100     50
450     400
50      300
400     -mark- 12
300     350
- mark- 13 /y
400
800     100
400     450
100     50
450     400
50      300
400     -mark- 14
300     350
- mark- 15 /y
400
800     100
400     450
100     50
450     400
50      300
400     -mark- 16
300     350
- mark- 17 /y
400
800     100
400     450
100     50
450     400
50      300
400     -mark- 18
300     350
- mark- 19 /y
400
800     100
400     450
100     50
450     400
50      300
400     -mark- 20
300     350
- mark- 21 /y
400
800     100
400     450
100     50
450     400
50      300
400     -mark- 22
300     350
- mark- 23 /y
400
800     100
400     450
100     50
450     400
50      300
400     -mark- 24
300     350
- mark- 25 /y
400
800     100
400     450
100     50
450     400
50      300
400     -mark- 26
300     350
- mark- 27 /y
400
800     100
400     450
100     50
450     400
50      300
400     -mark- 28
300     350
- mark- 29 /y
400
800     100
400     450
100     50
450     400
50      300
400     -mark- 30
300     350
- mark- 31 /y
400
800     100
400     450
100     50
450     400
50      300
400     -mark- 32
300     350
- mark- 33 /y
400
800     100
400     450
100     50
450     400
50      300
400     -mark- 34
300     350
- mark- 35 /y
400
800     100
400     450
100     50
450     400
50      300
400     -mark- 36
300     350
- mark- 37 /y
400
800     100
400     450
100     50
450     400
50      300
400     -mark- 38
300     350
- mark- 39 /y
400
800     100
400     450
100     50
450     400
50      300
400     -mark- 40
300     350
- mark- 41 /y
400
800     100
400     450
100     50
450     400
50      300
400     -mark- 42
300     350
- mark- 43 /y
400
800     100
400     450
100     50
450     400
50      300
400     -mark- 44
300     350
- mark- 45 /y
400
800     100
400     450
100     50
450     400
50      300
400     -mark- 46
300     350
- mark- 47 /y
400
800     100
400     450
100     50
450     400
50      300
400     -mark- 48
300     350
- mark- 49 /y
400
800     100
400     450
100     50
450     400
50      300
400     -mark- 50
300     350
- mark- 51 /y
400
800     100
400     450
100     50
450     400
50      300
400     -mark- 52
300     350
- mark- 53 /y
400
800     100
400     450
100     50
450     400
50      300
400     -mark- 54
300     350
- mark- 55 /y
400
800     100
400     450
100     50
450     400
50      300
400     -mark- 56
300     350
- mark- 57 /y
400
800     100
400     450
100     50
450     400
50      300
400     -mark- 58
300     350
- mark- 59 /y
400
800     100
400     450
100     50
450     400
50      300
400     -mark- 60
300     350
- mark- 61 /y
400
800     100
400     450
100     50
450     400
50      300
400     -mark- 62
300     350
- mark- 63 /y
400
800     100
400     450
100     50
450     400
50      300
400     -mark- 64
300     350
- mark- 65 /y
400
800     100
400     450
100     50
450     400
50      300
400     -mark- 66
300     350
- mark- 67 /y
400
800     100
400     450
100     50
450     400
50      300
400     -mark- 68
300     350
- mark- 69 /y
400
800     100
400     450
100     50
450     400
50      300
400     -mark- 70
300     350
- mark- 71 /y
400
800     100
400     450
100     50
450     400
50      300
400     -mark- 72
300     350
- mark- 73 /y
400
800     100
400     450
100     50
450     400
50      300
400     -mark- 74
300     350
- mark- 75 /y
400
800     100
400     450
100     50
450     400
50      300
400     -mark- 76
300     350
- mark- 77 /y
400
800     100
400     450
100     50
450     400
50      300
400     -mark- 78
300     350
- mark- 79 /y
400
800     100
400     450
100     50
450     400
50      300
400     -mark- 80
300     350
- mark- 81 /y
400
800     100
400     450
100     50
450     400
50      300
400     -mark- 82
300     350
- mark- 83 /y
400
800     100
400     450
100     50
450     400
50      300
400     -mark- 84
300     350
- mark- 85 /y
400
800     100
400     450
100     50
450     400
50      300
400     -mark- 86
300     350
- mark- 87 /y
400
800     100
400     450
100     50
450     400
50      300
400     -mark- 88
300     350
- mark- 89 /y
400
800     100
400     450
100     50
450     400
50      300
400     -mark- 90
300     350
- mark- 91 /y
400
800     100
400     450
100     50
450     400
50      300
400     -mark- 92
300     350
- mark- 93 /y
400
800     100
400     450
100     50
450     400
50      300
400     -mark- 94
300     350
- mark- 95 /y
400
800     100
400     450
100     50
450     400
50      300
400     -mark- 96
300     350
- mark- 97 /y
400
800     100
400     450
100     50
450     400
50      300
400     -mark- 98
300     350
- mark- 99 /y
400
800     100
400     450
100     50
450     400
50      300
400     -mark- 100
300     
```



## for

Petlja s komandom `for` kontrolira brojač petlje za koji su date granice i korak napredovanja slično kao u drugim programskim jezicima, definirano kao na primjer: `for ... to ... next`. Komanda `for` ima 4 parametra: početna vrijednost, korak povećanja, konačna vrijednost i brojač petlje te proceduru izvršavanja komande. Početna vrijednost odmah postavlja brojač (trenutna vrijednost petlje) na vrh stacka (**2 2A 2B 2C**). Brojač `for` petlje u PostScriptu može imati vrijednost cjelobrojnu, necjelobrojnu, pozitivnu i negativnu. Brojač se može povećavati ili smanjivati. Prije prvog početka `for` procedure brišu se tri podatka iz stacka (**1**). U prvom primjeru komanda `moveto` koristi vrijednost `for` petlje za poziciju x. Komanda `move to` miče dvije vrijednosti u stacku. Početkom slijedećeg kruga petlje na vrhu stacka postavlja se vrijednost slijedećeg brojača petlje.

Drugi primjer koristi vrijednost brojača petlje dva puta. Zbog toga je ta vrijednost duplicirana (**4**). Prije iscrtavanja linije **E** uvađa se x pozicija kraja linije. Za drugu grupu linija ispisane su vrijednosti stacka (**4 i 5**) u programu za prve tri linije i za zadnje tri linije (**Z**).

```
-mark-
300
60
100 1
-mark-
100 2
-mark-
200
100 3
-mark-
160 2A
-mark-
200
160 3A
-mark-
220 2B
-mark-
200
220 3B
-mark-
280 2C
-mark-
200
280
```

```
-mark-
350
400
400 4E
-mark-
400
200 5E
-mark-
350
390
390 4'
-mark-
390
200 5'
-mark-
350
380
380 4''
-mark-
380
200 5''
-mark-
```

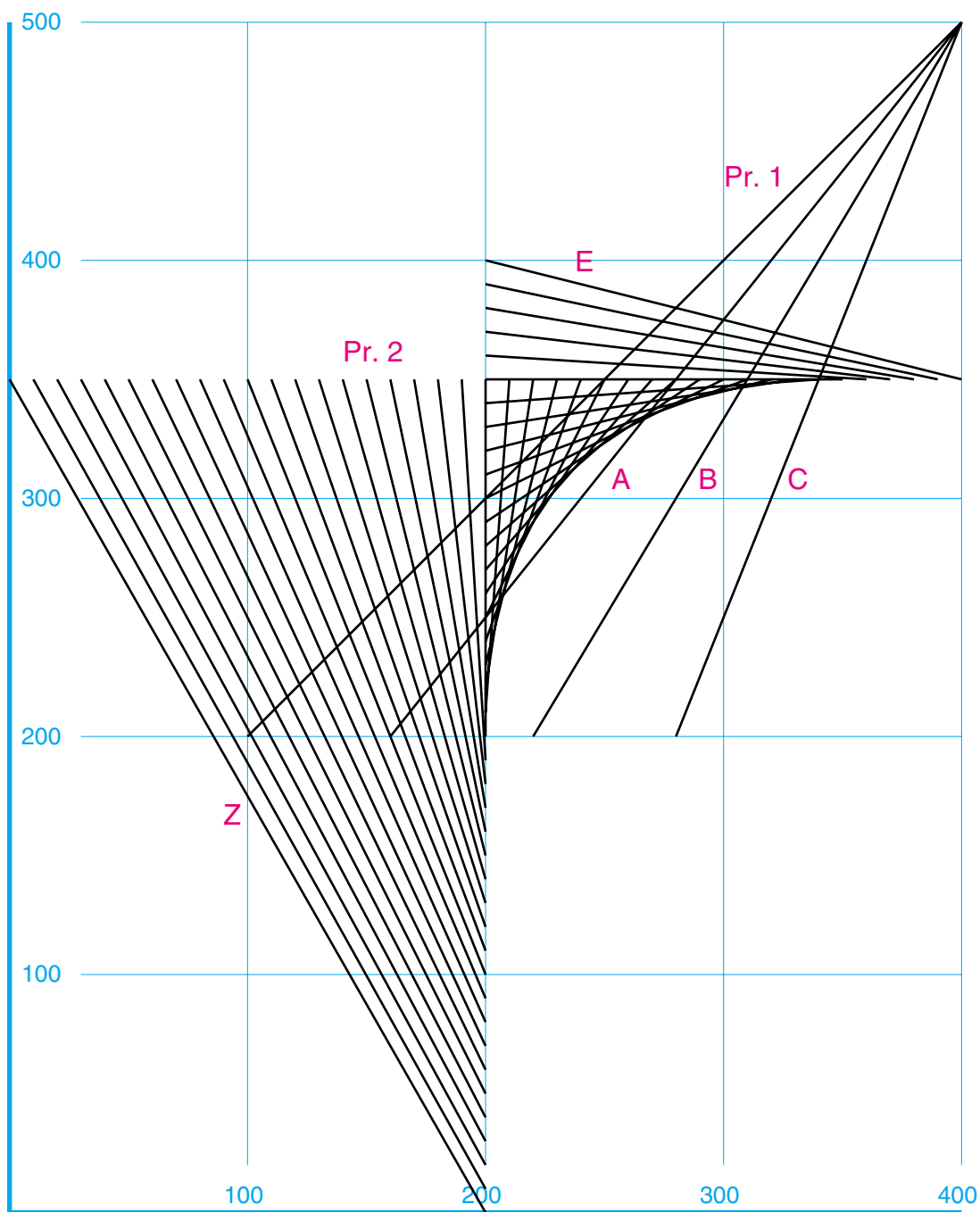
```
.....
350
20
20 4
-mark-
20
200 5
-mark-
350
10
10 4X
-mark-
200 5X
-mark-
350
380
0
0 4Z
-mark-
0
200 5Z
```

```
/s { mark pstack pop } def
%primjer br.1
100 60 300 s 1 { s 2 200 s 3 moveto
400 500 lineto stroke } for

%primjer br. 2
400 -10 0 { dup 350 s 4 moveto
200 exch s 5 lineto stroke } for

showpage
```





*for*

Ako se brojač petlje želi višestruko koristiti unutar petlje tada predlažemo da se brojač upiše u neku varijablu. U prvom primjeru vrijednost brojača je stavljena u varijablu `j` koja se koristi kao `x` koordinata središta kružnice. Nadalje, u relaciji `y` koordinate:  $y=300+(350-j)$  te u izračunavanju radijusa:  $r=j/6$ . U svakom krugu petlje `j` se smanjuje za 10 od početnih 350 do 200.

U drugom primjeru `y` koordinata druge tangentne točke Bezierove krivulje ima vrijednost brojača petlje. Postavlja se na početak stacka u momentu kada je bila gurnuta na četvrto mjesto učitavanjem parametara 100, 350 i 150. Na kraju izvršavanja Bezierove krivulje u stacku je ostala stara vrijednost brojača. Poslije svih krugova petlje ostale su na stacku sve vrijednosti brojača. To bi se trebalo ukloniti brisanjem te vrijednosti, na primjer, poslije komande `curveto` dodati komandu `pop`. Nakon kraja petlje stavili smo komandu `clear`.

U trećem primjeru, vrijednost brojača koristi se tri puta. Jednom, da odredi svjetlinu (`setgray` će iskoristiti vrh stacka), a drugi i treći puta kao parametri u komandi `scale`. Zbog toga se vrijednost brojača dva puta kopira (`dup dup setgray scale`).

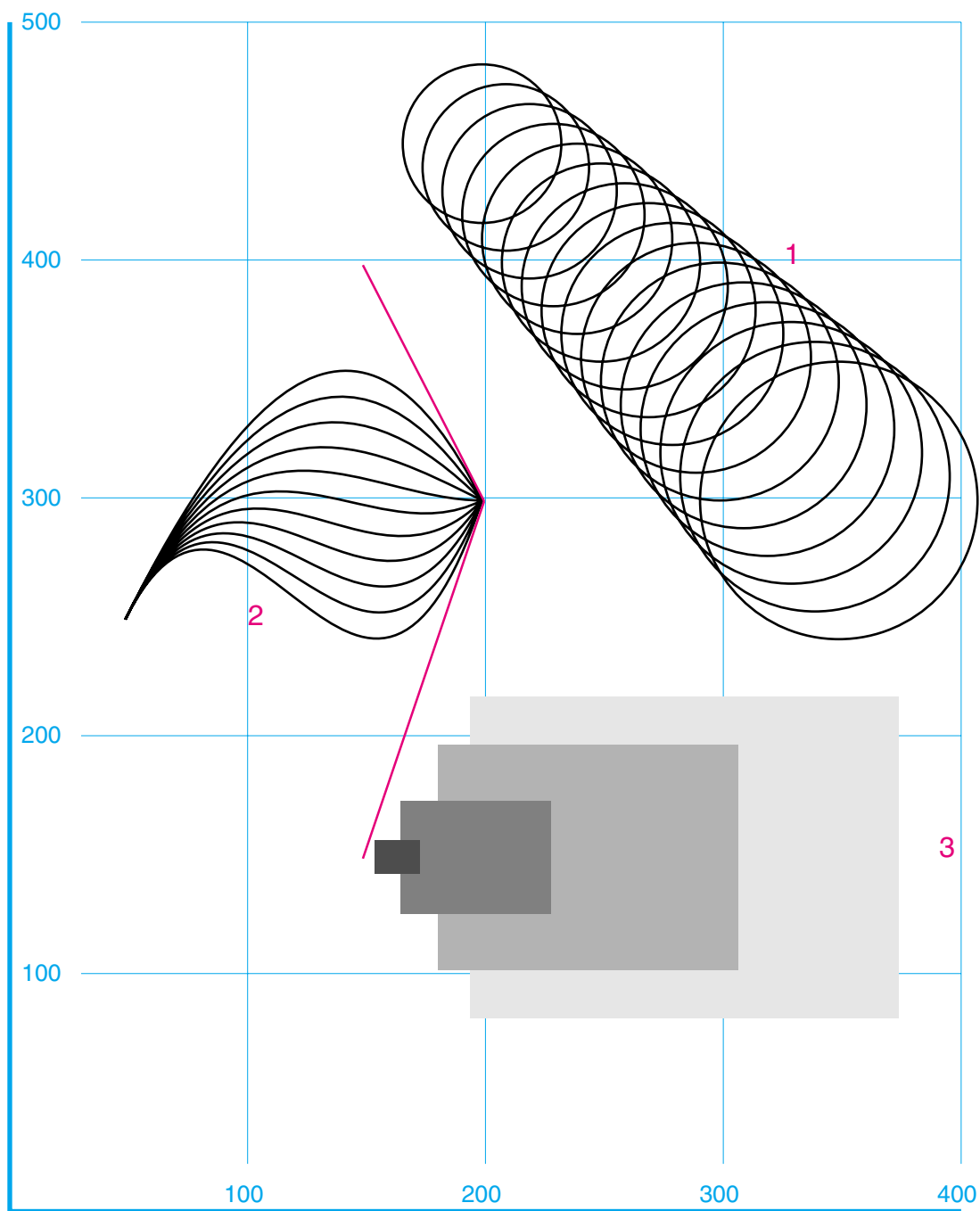
```
%primjer br.1
350 -10 200 { /j exch def
  j 300 350 j sub add j 6 div 0 360 arc
stroke }
for
```

Ispis pstack:

```
150
175
200
225
250
275
300
325
350
375
400
```

```
%primjer br.2
400 -25 150 {
50 250 moveto
100 350 150 3 index 200 300 curveto
stroke }
for pstack clear
```

```
%primjer br.3
150 150 translate
0.9 -0.2 0.3 { dup dup setgray scale
50 -75 moveto
200 0 rlineto 0 150 rlineto -200 0 rlineto closepath
fill }
for
showpage
```



## *Program mreže*

U dosadašnjim primjerima na početku svakog programa programirana je mreža. Tanke horizontalne linije počinju 30 točaka udaljene od nultočke. Svaki novi krug **repeat** petlje translatira koordinatni sustav vertikalno za 100 točaka. Vertikalne linije počinju na visini 20 točaka.

Vertikalno ispisivanje brojaka za označavanje vodoravnih linija riješen je **for** petljom gdje se brojač petlje koristi za ispis slova brojaka te za izračunavanje pozicije ispisa brojaka. Vrijednost brojača petlje postavlja se u varijablu **y**. Tekst brojaka se ispisuje: **y tekst cvs** gdje je u varijabli **tekst** određena dužina od 3 znaka (**/tekst 3 string def**), a komanda **cvs** pretvara numeričku vrijednost u string. Tekst se postavlja na udaljenost 5 točaka, a pomaknut je dolje za 2 točke od vrijednosti koju ispisuje: **5 y 2 sub moveto**. Ispis teksta iz fonta FSHelvetica obavlja se procedurom: **/FSHelvetica findfont 10 scalefont setfont**.

Brojke u podnožju koje označuju koordinate vertikalnih linija riješene su na drugi način. Podignute su za 5 točaka od nultočke. Vrijednost brojača petlje kreće se od 90, 190 ... do 390. Pozicija početka ispisa brojke u tom slučaju jednaka je: **x 5 moveto**, gdje je **x** jednak brojaču petlje. Sama slika brojaka izračunata je zbrajanjem: **x 10 add**. Ta vrijednost konvertirana je na kraju u string.

```
/mreza {1 0 0 0 setcmykcolor    %Linije se iscrtavaju cijan bojom
gsave                          %Linije koordinatnih osi
2 setlinewidth 400 0 moveto 0 0 lineto 0 500 lineto  stroke
grestore
gsave                          %Mreža horizontalnih linija
0.3 setlinewidth
5 { 30 100 moveto 400 100 lineto stroke 0 100 translate } repeat
grestore
gsave                          %Mreža vertikalnih linija
0.3 setlinewidth
4 { 100 20 moveto 100 500 lineto stroke 100 0 translate } repeat
grestore
gsave                          %Numeričke vrijednosti horizontalnih linija
/tekst 3 string def /FSHelvetica findfont 10 scalefont setfont
100 100 500 { /y exch def 5 y 2 sub moveto y tekst cvs show } for
                          %Numeričke vrijednosti vertikalnih linija
90 100 400 { /x exch def x 5 moveto x 10 add tekst cvs show } for
grestore
0 setgray} bind def
%Prije pocetka pisanja ostalih programa treba kopirati ovu proceduru i
%pozvati je sa mreza kao u slijedecem primjeru
mreza
100 200 translate 0 0 moveto /FSHelvetica findfont 100 scalefont setfont
(mreža) show showpage
```



## *Mreža tip2*

U knjizi se još koriste tri vrste mreža. Prikazan je ispis i programski kod procedura `mreza2` i `mreza3`. Postoji i procedura `mreza4` koja se razlikuje od `mreze3` samo u tome što se ne ispisuju tanke vodoravne linije. Naredbe za proceduru `mreza4` nisu prikazane u knjizi.

```

350 /mreza2 {1 0 0 0 setcmykcolor
    gsave          %Linije koordinatnih osi
    2 setlinewidth 400 0 moveto 0 0 lineto 0 500 lineto  stroke
    grestore
        %Mreža horizontalnih linija
    gsave
    0.3 setlinewidth 30 50 moveto 400 50 lineto stroke 0 50 translate
    2 { 30 150 moveto 400 150 lineto stroke 0 150 translate } repeat
    grestore
        %Numeričke vrijednosti horizontalnih linija
    gsave
    /tekst 3 string def /FSHelvetica findfont 10 scalefont setfont
    50 150 350 { /y exch def 5 y 2 sub moveto y tekst cvs show } for
    grestore
    0 setgray} bind def
200 mreza2
    showpage

```

## *Mreža tip3*

```

500   Kada u nekim primjerima u knjizi primjetite razliku u mreži
      podloge od standardne procedure mreža znači da se radi o nekoj
480   od ovih mreža. Tada treba programske naredbe pripadne proce-
460   dure kopirati prije programskog primjera i pozvati sa imenom da
      se izvrši njen ispis.
440
420
400   /mreza3 {1 0 0 0 setcmykcolor
380   gsave          %Linije koordinatnih osi
      2 setlinewidth 400 0 moveto 0 0 lineto 0 500 lineto stroke
360   grestore
      %Mreža horizontalnih linija
340   gsave
320   0.3 setlinewidth
      25 { 30 20 moveto 400 20 lineto stroke 0 20 translate } repeat
300   grestore
      %Numeričke vrijednosti horizontalnih linija
280   gsave
260   /tekst 3 string def /FSHelvetica findfont 10 scalefont setfont
      20 20 500 { /y exch def 5 y 2 sub moveto y tekst cvs show } for
240   grestore
220   0 setgray} bind def
      mreza3
200   showpage

```

## *setcmykcolor*

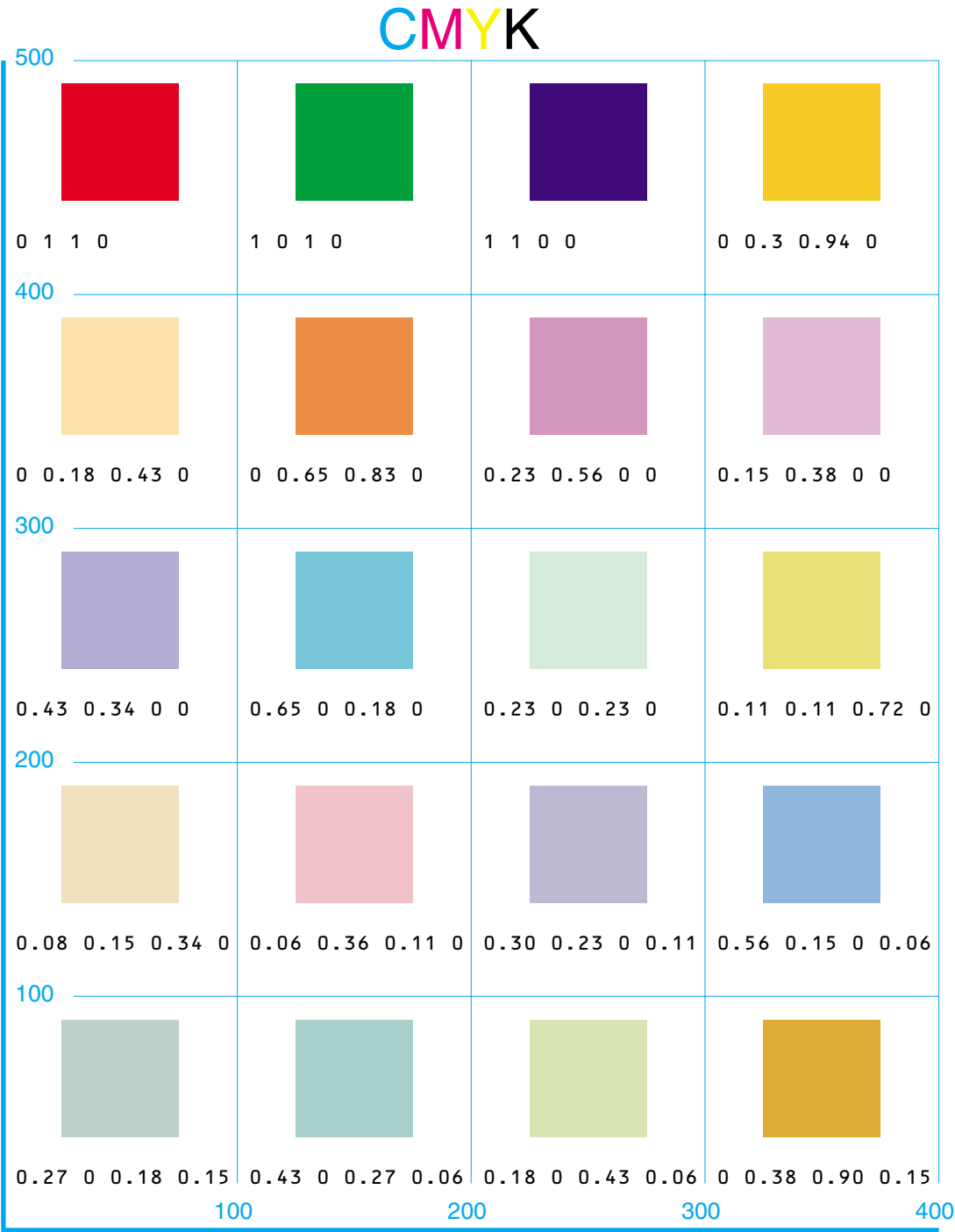
**c m y k** setcmykcolor

Komanda `setcmykcolor` definira boju sa četiri parametra: cijan(c), magenta(m), žuta(y), crna(k). Parametri se zadaju u intervalu od 0.0. do 1.0 na inverzni način od `setgray` komanda. Kada parametar iznosi 1.0 tada je pokrivenost boje 100%.

Program `CMYK` crta tablicu boja koristeći procedure `kvadrat` i `txt` da bi se prikazale pojedine boje tog tiskarskog modela boja i pripadni iznos parametara komande.

```
%program CMYK
gsave
/kvadrat {translate 0 0 moveto 50 0 rlineto 0 50 rlineto
-50 0 rlineto closepath} bind def
/txtr {-20 -20 moveto 0 setgray show } bind def
/FSHelvetica findfont 24 scalefont setfont
160 505 moveto
1 0 0 0 setcmykcolor (C) show 0 1 0 0 setcmykcolor (M) show
0 0 1 0 setcmykcolor (Y) show 0 0 0 1 setcmykcolor (K) show
/FSOcb-B findfont 8 scalefont setfont
25 440 kvadrat 0 1 1 0 setcmykcolor fill (0 1 1 0) txt
100 0 kvadrat 1 0 1 0 setcmykcolor fill (1 0 1 0) txt
100 0 kvadrat 1 1 0 0 setcmykcolor fill (1 1 0 0) txt
100 0 kvadrat 0 0.3 0.94 0 setcmykcolor fill (0 0.3 0.94 0) txt
-300 -100 kvadrat 0 0.18 0.43 0 setcmykcolor fill (0 0.18 0.43 0) txt
100 0 kvadrat 0 0.65 0.83 0 setcmykcolor fill (0 0.65 0.83 0) txt
100 0 kvadrat 0.23 0.56 0 0 setcmykcolor fill (0.23 0.56 0 0) txt
100 0 kvadrat 0.15 0.38 0 0 setcmykcolor fill (0.15 0.38 0 0) txt
-300 -100 kvadrat 0.43 0.34 0 0 setcmykcolor fill (0.43 0.34 0 0) txt
100 0 kvadrat 0.65 0 0.18 0 setcmykcolor fill (0.65 0 0.18 0) txt
100 0 kvadrat 0.23 0 0.23 0 setcmykcolor fill (0.23 0 0.23 0) txt
100 0 kvadrat 0.11 0.11 0.72 0 setcmykcolor
fill (0.11 0.11 0.72 0) txt
-300 -100 kvadrat 0.08 0.15 0.34 0 setcmykcolor
fill (0.08 0.15 0.34 0) txt
100 0 kvadrat 0.06 0.36 0.11 0 setcmykcolor fill (0.06 0.36 0.11 0) txt
100 0 kvadrat 0.30 0.23 0 0.11 setcmykcolor fill (0.30 0.23 0 0.11) txt
100 0 kvadrat 0.56 0.15 0 0.06 setcmykcolor fill (0.56 0.15 0 0.06) txt
-300 -100 kvadrat 0.27 0 0.18 0.15 setcmykcolor
fill (0.27 0 0.18 0.15) txt
100 0 kvadrat 0.43 0 0.27 0.06 setcmykcolor fill (0.43 0 0.27 0.06) txt
100 0 kvadrat 0.18 0 0.43 0.06 setcmykcolor fill (0.18 0 0.43 0.06) txt
100 0 kvadrat 0 0.38 0.90 0.15 setcmykcolor fill (0 0.38 0.90 0.15) txt
grestore
showpage
```





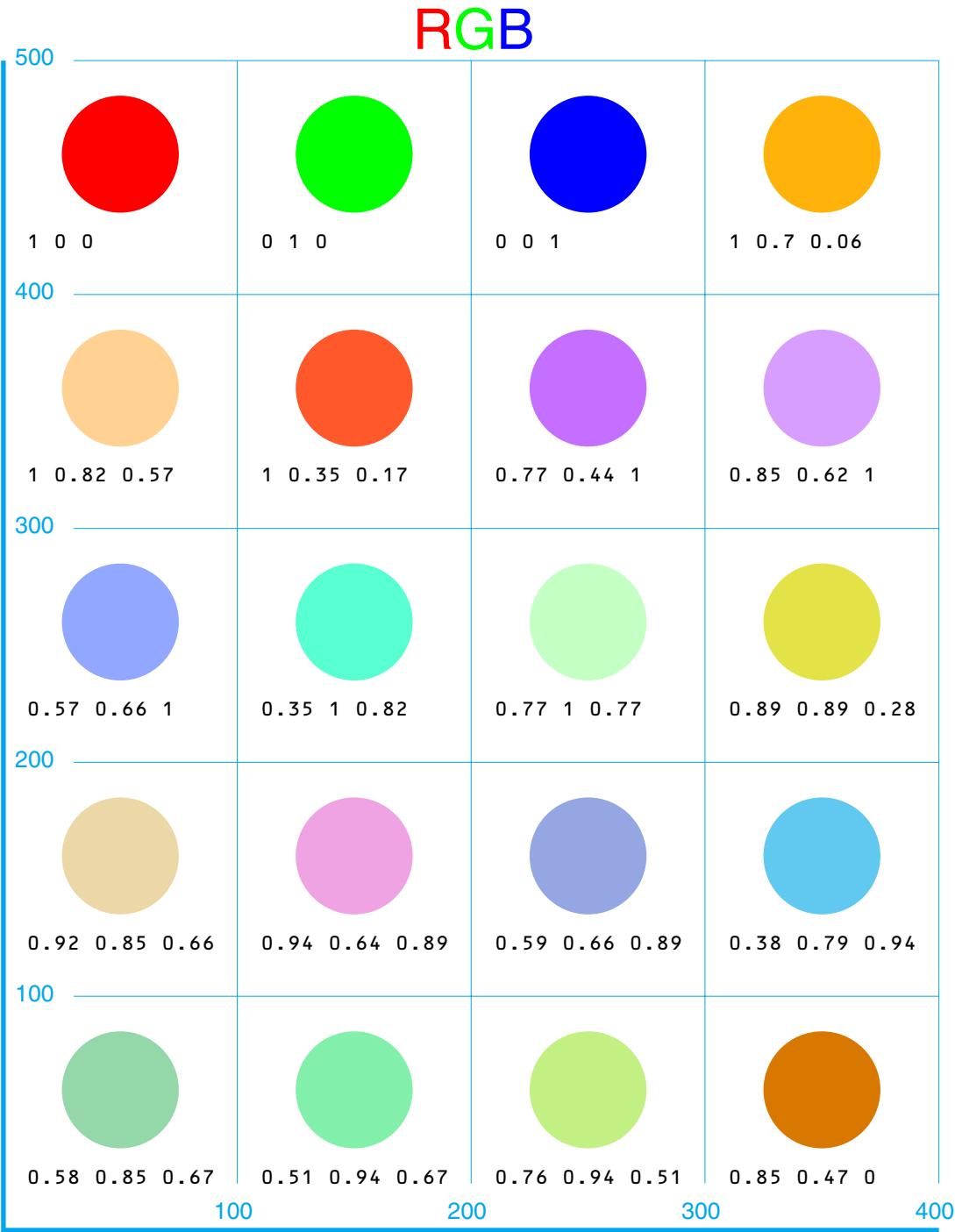
***setrgbcolor******sethsbcolor***

```
r g b setrgbcolor
h s b sethsbcolor
```

Komanda `setrgbcolor` definira boju sa tri parametra: crvena(r), zelena(g), plava(b). Oni se zadaju u intervalu od 0.0 do 1.0. Iznos 1.0 predstavlja pokrivenost od 100% te boje. Pošto se tisak izvršio četverbojnim digitalnim tiskom konverzija iz RGB u CMYK sustav boja se događa automatski u RIP-u u momentu slanja programa.

Komanda `sethsbcolor` definira boju sa parametrom tona (hue), zasićenosti (saturation) i svjetline (brightness). Da bi se predstavio HSB model boja isprogramirana su na slijedećoj programskoj stranici četiri kolorna kruga. Svaki od njih ima drugu svjetlinu (B), svi zajedno istu zasićenost (S=1), a unutar kruga mijenja se ton boje (H).

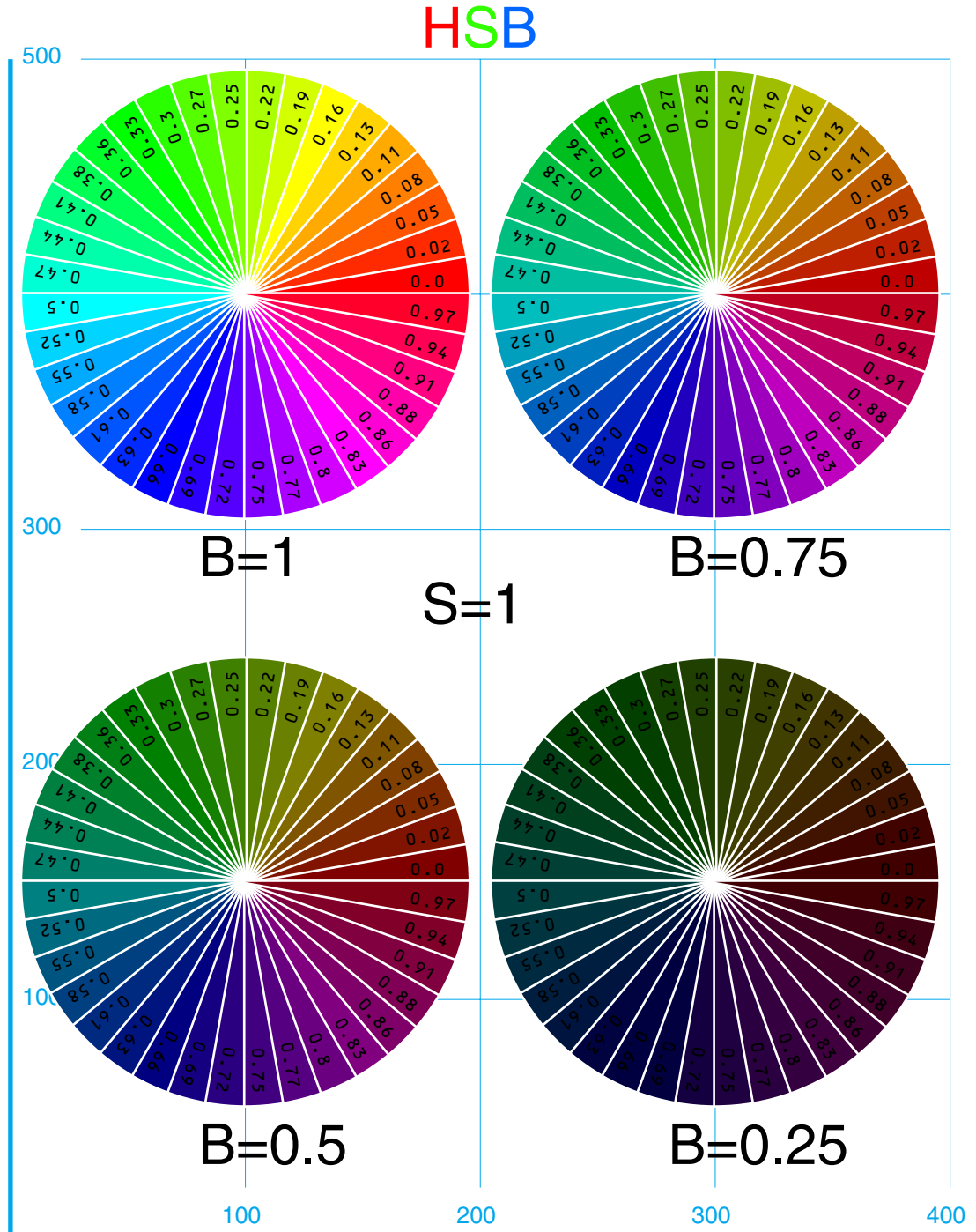
```
%program RGB
gsave
/krug {translate 0 0 moveto 0 0 25 0 360 arc} bind def
/txt {-40 -40 moveto 0 setgray show } bind def
/FSHelvetica findfont 24 scalefont setfont
175 505 moveto
1 0 0 setrgbcolor (R) show 0 1 0 setrgbcolor (G) show
0 0 1 setrgbcolor (B) show
/FS0cb-B findfont 8 scalefont setfont
50 460 krug 1 0 0 setrgbcolor fill (1 0 0 ) txt
100 0 krug 0 1 0 setrgbcolor fill (0 1 0) txt
100 0 krug 0 0 1 setrgbcolor fill (0 0 1) txt
100 0 krug 1 0.7 0.06 setrgbcolor fill (1 0.7 0.06) txt
-300 -100 krug 1 0.82 0.57 setrgbcolor fill (1 0.82 0.57) txt
100 0 krug 1 0.35 0.17 setrgbcolor fill (1 0.35 0.17) txt
100 0 krug 0.77 0.44 1 setrgbcolor fill (0.77 0.44 1) txt
100 0 krug 0.85 0.62 1 setrgbcolor fill (0.85 0.62 1) txt
-300 -100 krug 0.57 0.66 1 setrgbcolor fill (0.57 0.66 1) txt
100 0 krug 0.35 1 0.82 setrgbcolor fill (0.35 1 0.82) txt
100 0 krug 0.77 1 0.77 setrgbcolor fill (0.77 1 0.77) txt
100 0 krug 0.89 0.89 0.28 setrgbcolor fill (0.89 0.89 0.28) txt
-300 -100 krug 0.92 0.85 0.66 setrgbcolor
fill (0.92 0.85 0.66) txt
100 0 krug 0.94 0.64 0.89 setrgbcolor fill (0.94 0.64 0.89) txt
100 0 krug 0.59 0.66 0.89 setrgbcolor fill (0.59 0.66 0.89) txt
100 0 krug 0.38 0.79 0.94 setrgbcolor fill (0.38 0.79 0.94) txt
-300 -100 krug 0.58 0.85 0.67 setrgbcolor
fill (0.58 0.85 0.67) txt
100 0 krug 0.51 0.94 0.67 setrgbcolor fill (0.51 0.94 0.67) txt
100 0 krug 0.76 0.94 0.51 setrgbcolor fill (0.76 0.94 0.51) txt
100 0 krug 0.85 0.47 0 setrgbcolor fill (0.85 0.47 0) txt
grestore
showpage
```



```

%program HSB
/FSHelvetica findfont 24 scalefont setfont
/str 4 string def 1 36 div /korak exch def
gsave 175 505 moveto
0 1 1 sethsbcolor (H) show 0.3 1 1 sethsbcolor (S) show
0.6 1 1 sethsbcolor (B) show grestore
gsave /FS0cb-B findfont 7 scalefont setfont          %B=1
/H 0 def /S 1 def /B 1 def 100 400 translate
36 { H S B sethsbcolor
0 0 moveto 0 0 95 0 10 arc
gsave 1 setgray 2 setlinewidth 0 0 lineto stroke grestore
closepath fill
gsave 0 setgray 70 3 moveto H 100 mul cvi 100 div str cvs show grestore
10 rotate H korak add /H exch def
} repeat grestore gsave 80 280 moveto (B=1) show grestore
gsave /FS0cb-B findfont 7 scalefont setfont          %B=0.75
/H 0 def /S 1 def /B 0.75 def 300 400 translate
36 { H S B sethsbcolor
0 0 moveto 0 0 95 0 10 arc
gsave 1 setgray 2 setlinewidth 0 0 lineto stroke grestore
closepath fill
gsave 0 setgray 70 3 moveto H 100 mul cvi 100 div str cvs show grestore
10 rotate H korak add /H exch def
} repeat grestore gsave 280 280 moveto (B=0.75) show grestore
gsave /FS0cb-B findfont 7 scalefont setfont          %B=0.5
/H 0 def /S 1 def /B 0.5 def 100 150 translate
36 { H S B sethsbcolor
0 0 moveto
0 0 95 0 10 arc
gsave 1 setgray 2 setlinewidth 0 0 lineto stroke grestore
closepath fill
gsave 0 setgray 70 3 moveto H 100 mul cvi 100 div str cvs show grestore
10 rotate H korak add /H exch def
} repeat grestore gsave 80 30 moveto (B=0.5) show grestore
gsave /FS0cb-B findfont 7 scalefont setfont          %B=0.25
/H 0 def /S 1 def /B 0.25 def 300 150 translate
36 { H S B sethsbcolor
0 0 moveto 0 0 95 0 10 arc
gsave 1 setgray 2 setlinewidth 0 0 lineto stroke grestore
closepath fill
gsave 0 setgray 70 3 moveto H 100 mul cvi 100 div str cvs show grestore
10 rotate H korak add /H exch def
} repeat grestore
gsave 280 30 moveto (B=0.25) show grestore
175 260 moveto (S=1) show
showpage

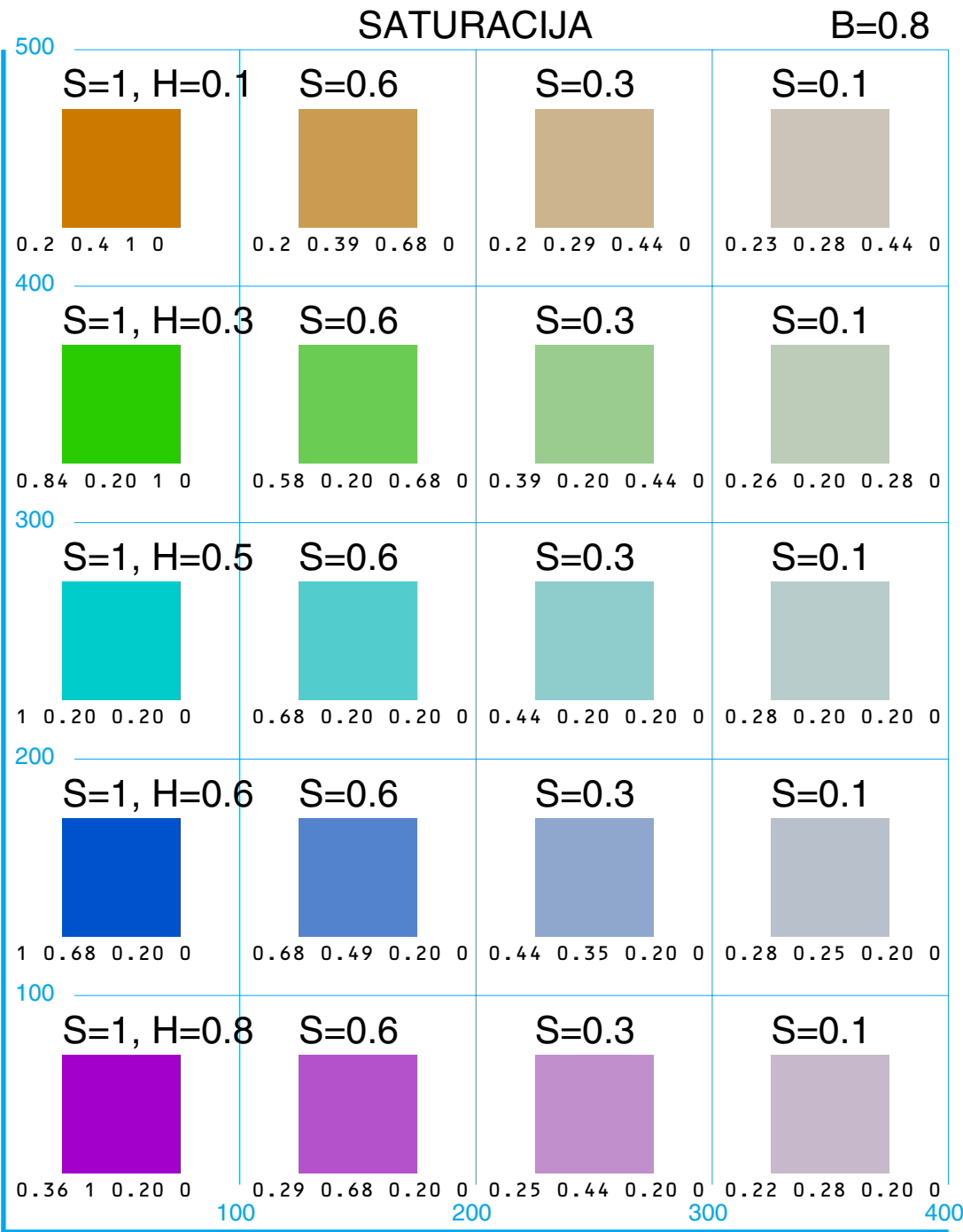
```



U prethodnom primjeru sa HSB krugovima parametar S je uvijek bio 1, odnosno sve boje su imale maksimalnu zasićenost (saturaciju). U ovom primjeru sve boje imaju istu svjetlinu ( $B=0.8$ ), a svaki redak ima isti ton boje ( $h$ ). Unutar svakog retka mijenja se zasićenost od  $S=1$  do  $S=0.1$ .

Kao i kod RGB modela konverzija iz HSB modela u CMYK događa se kada se PostScript komanda `sethsbcolor` pošalje prema PostScript RIP-u koji ima podešenu konverziju za taj vanjski CMYK uređaj. Slično se događa kada šaljemo kolor komande na crno bijeli PostScript printer pri čemu se vrši konverzija u ekvivalentni parametar `setgray` komande.

```
%program HSB saturacija
gsave
/kvadrat {translate 0 0 moveto 50 0 rlineto 0 50 rlineto
-50 0 rlineto closepath} bind def
/txt {/FSHelvetica findfont 16 scalefont setfont
0 55 moveto 0 setgray show } bind def
/txt2 {/FS0cb-B findfont 8 scalefont setfont
-20 -10 moveto 0 setgray show } bind def
/FSHelvetica findfont 16 scalefont setfont
150 505 moveto (SATURACIJA) show 350 505 moveto (B=0.8) show
25 425 kvadrat 0.1 1 0.8 sethsbcolor fill (S=1, H=0.1) txt (0.2 0.4 1 0) txt2
100 0 kvadrat 0.1 0.6 0.8 sethsbcolor fill (S=0.6) txt (0.2 0.39 0.68 0) txt2
100 0 kvadrat 0.1 0.3 0.8 sethsbcolor fill (S=0.3) txt (0.2 0.29 0.44 0) txt2
100 0 kvadrat 0.1 0.1 0.8 sethsbcolor fill (S=0.1) txt (0.23 0.28 0.44 0) txt2
-300 -100 kvadrat 0.3 1 0.8 sethsbcolor fill (S=1, H=0.3) txt (0.84 0.20 1 0) txt2
100 0 kvadrat 0.3 0.6 0.8 sethsbcolor fill (S=0.6) txt (0.58 0.20 0.68 0) txt2
100 0 kvadrat 0.3 0.3 0.8 sethsbcolor fill (S=0.3) txt (0.39 0.20 0.44 0) txt2
100 0 kvadrat 0.3 0.1 0.8 sethsbcolor fill (S=0.1) txt (0.26 0.20 0.28 0) txt2
-300 -100 kvadrat 0.5 1 0.8 sethsbcolor fill (S=1, H=0.5) txt (1 0.20 0.20 0) txt2
100 0 kvadrat 0.5 0.6 0.8 sethsbcolor fill (S=0.6) txt (0.68 0.20 0.20 0) txt2
100 0 kvadrat 0.5 0.3 0.8 sethsbcolor fill (S=0.3) txt (0.44 0.20 0.20 0) txt2
100 0 kvadrat 0.5 0.1 0.8 sethsbcolor fill (S=0.1) txt (0.28 0.20 0.20 0) txt2
-300 -100 kvadrat 0.6 1 0.8 sethsbcolor fill (S=1, H=0.6) txt (1 0.68 0.20 0) txt2
100 0 kvadrat 0.6 0.6 0.8 sethsbcolor fill (S=0.6) txt (0.68 0.49 0.20 0) txt2
100 0 kvadrat 0.6 0.3 0.8 sethsbcolor fill (S=0.3) txt (0.44 0.35 0.20 0) txt2
100 0 kvadrat 0.6 0.1 0.8 sethsbcolor fill (S=0.1) txt (0.28 0.25 0.20 0) txt2
-300 -100 kvadrat 0.8 1 0.8 sethsbcolor fill (S=1, H=0.8) txt (0.36 1 0.20 0) txt2
100 0 kvadrat 0.8 0.6 0.8 sethsbcolor fill (S=0.6) txt (0.29 0.68 0.20 0) txt2
100 0 kvadrat 0.8 0.3 0.8 sethsbcolor fill (S=0.3) txt (0.25 0.44 0.20 0) txt2
100 0 kvadrat 0.8 0.1 0.8 sethsbcolor fill (S=0.1) txt (0.22 0.28 0.20 0) txt2
grestore showpage
```



Programskim mjenjanjem parametara kolor komandi možemo dizajnirati različite kolorne efekte. U prvom primjeru se u `for` petlji sa promjenom radijusa mijenja i parametar tona boje (h) komande `sethsbcolor`.

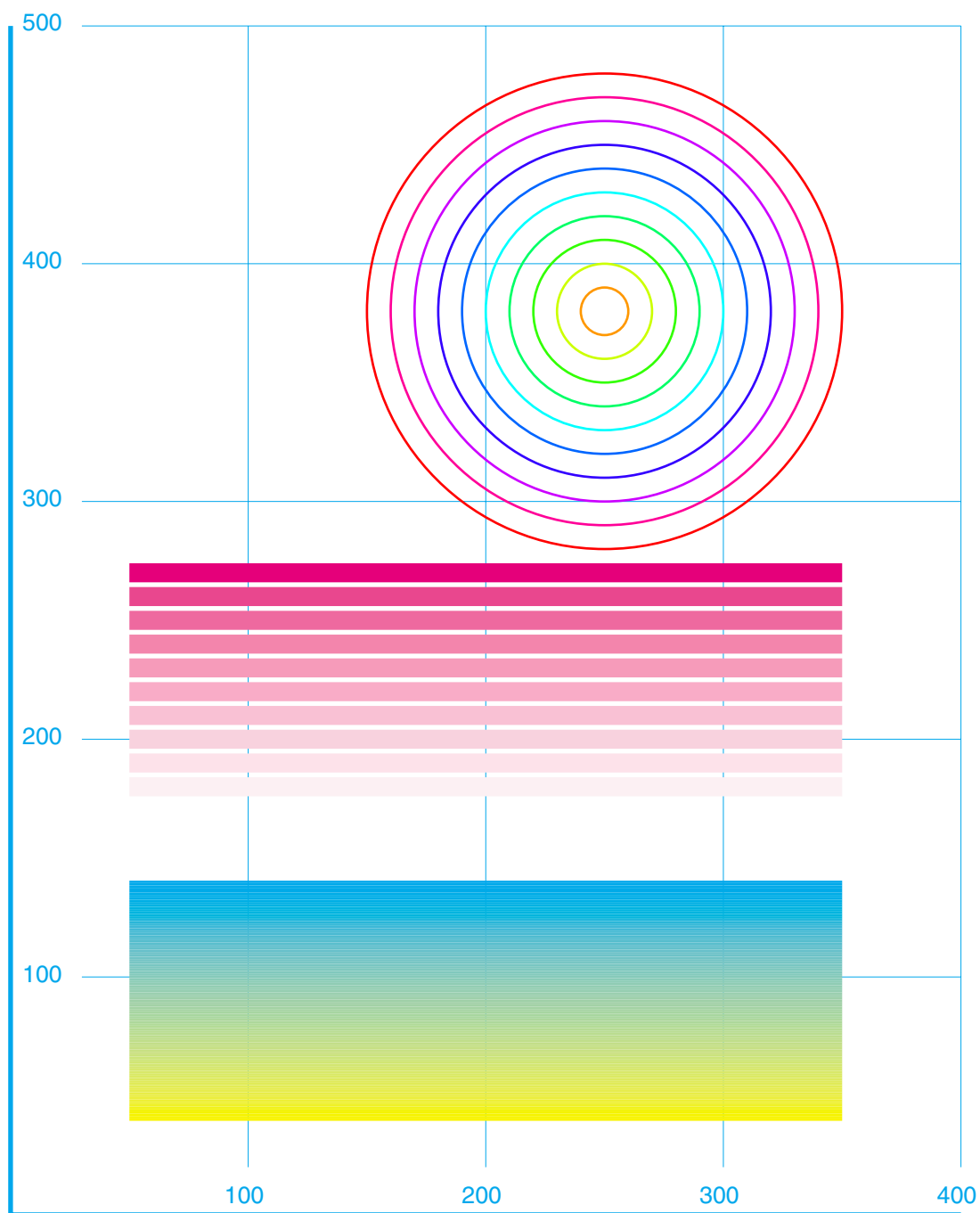
U drugom primjeru se u petlji crta linija debljine 8 točaka s time da se u svakom koraku petlje vrši pomak prema gore i povećava parametar magente u `setcmykcolor` komandi.

Treći primjer ostvaruje gradaciju od žute do cijan boje crtajući u petlji liniju po liniju s pomakom od jedne točke i ujedno promjenom njene boje u `setcmykcolor` komandi.

```
%program BOJE
gsave
250 380 translate
/boja 0 def
1 1 10 {/b exch def /r {b 10 mul} def /boja {b 10 div}
def
0 0 r 0 360 arc
boja 1 1 sethsbcolor
stroke
} for
grestore

gsave
50 170 translate
/boja 0 def
1 1 10 {/b exch def /boja {b 10 div} def
0 b 10 mul moveto 300 0 rlineto
0 boja 0 0 setcmykcolor
8 setlinewidth
stroke
} for
grestore
gsave
50 40 translate
0 0.01 1
{dup /C exch def 1 exch sub /Y exch def C 0 Y 0
setcmykcolor
0 0 moveto 300 0 rlineto stroke 0 1 translate} bind for
grestore
showpage
```







Vilko Žiljak • Klaudio Pap

# PostScript

II poglavlje  
programiranje tipografije

# *findfont*

## *scalefont*

### *setfont*

#### *show*

```
/imefonta findfont
s scalefont
font setfont
string show
```

Slovni znakovi oblikuju se unutar "četverca". Četverac je pravokutnik unutar kojeg se postavljaju slika slovnog znaka i njegova geometrija. Tokom pisanja teksta, kada definiramo visinu slova, mi definiramo visinu četverca a ne visinu slike slova. Slika slova, za većinu slova, smještena je unutar četverca (A B C a b c..), a neki slovni znakovi djelomično (g j  $\Sigma$   $\Pi$   $\sqrt{\text{..}}$ ) ili u cjelosti (neki samostalni akcenti) izlaze iz četverca. Dno četverca leži na pismovnoj liniji, donji lijevi ugao četverca je nulta točka slovnog znaka. Debljinska vrijednost slova uglavnom uključuje cijelu širinu slike slova sa dodatnom bjelinom do slijedećeg slovnog znaka. Nulta točka pozicioniranja slijedećeg slova u tekstu je na točki debljinske vrijednosti prethodnog slova.

Latiničko pismo je četverolinijsko: pismovna linija, krov verzala, krov kurenta i descender. Visina slike verzalnih slova (A B C D..) je oko 70% četverca, visina kurentnih slova (a c e i m..) je do 50% četverca, spuštanje descendera u kurentnim slovima (j g..) je od 20 do 30%. Točne veličine za karakteristična slova i neke fontove ilustrirali smo programom na ovoj stranici. Dizajneri su dodali još četiri linije: dvije linije za krov nekih kurentnih slova (t b d f..), liniju akcenta (Ž Š Č Ć) i liniju spajanja kurentnih slova nekih rukopisnih fontova.

Komande:

**findfont** - određuje traženje fonta koji treba biti na raspolaganju programu slaganja teksta

**scalefont** - definiranje visine četverca u točkama

**setfont** - postavljanje fonta aktivnim

**show** - prikaz fonta na postscript stranici

Komandom **scale** na početku programa i parametrima 0.9 i 1 suzila su se slova na 90% osnovne širine.

```
0.9 1 scale
30 50 moveto
/FSFutura findfont 100 scalefont setfont
(ARČjktbgf) show
30 200 moveto
/FSShellyAllegro findfont 100 scalefont setfont
(ARČjktbgf) show
30 350 moveto
/FSTimesRom findfont 100 scalefont setfont
(ARČjktbgf) show
showpage
```



***findfont***  
***scalefont***  
***setfont***  
***show***

Ovdje je prikazano 24 različita fonta veličine četverca od 12 točaka. Međusobno se razlikuju po debljinskim vrijednostima i po tome što nemaju istovrsni set znakova. Zato se moralo definirati više setova znakova (stringova) koji su se željeli prikazati. Da bi program radio treba prije njegovog slanja na PostScript uređaj (printer, fotojedinicu...) poslati fontove sa programom kao što je Downloader. To je jednostavnije nego pakirati fontove unutar našeg programa jer je jedan znak u fontu PostScript program za sebe .

```
%program FONTOVI
/str1 (abcčćdđefghijklmnoprsštuvzžABCČČDĐEFGHIJKLMNOPRSŠTUVZŽ0123456789,!:.) def
/str2 (abcčćdđefghijklmnoprsšžABCČČDĐEFGHIJKLMNOPRSŠŽ0123456789,!:.) def
/str3 (abcčćdđefghoprsšžABCČČDĐEFGHOPRSŠŽ0123456789,!:.) def
/str4 (abcdefghoprsABCDEFGHIOPRS0123456789,!:.) def
/str5 (abcdefghijklmnopqrstuvwxyzDFGJLPQSVWXY) def
/str6 (abcdefghijklmnopqrstuvwxyz{~ABCDEFGHIJKLNOSTUVWXYZ) def
/str7 (abcdefghijklmnopqrstuvwxyzDIJLNOQTXYZ[\];) def

30 20 moveto /FSShellyAllegro findfont 12 scalefont setfont str2 show
30 40 moveto /FSFetteFraktur findfont 12 scalefont setfont str2 show
30 60 moveto /FSTimesRom findfont 12 scalefont setfont str2 show
30 80 moveto /FSLinoText findfont 12 scalefont setfont str2 show
30 100 moveto /FSPresentScript findfont 12 scalefont setfont str3 show
30 120 moveto /FSPostAntiqua findfont 12 scalefont setfont str2 show
30 140 moveto /FSPeignotBold findfont 12 scalefont setfont str2 show
30 160 moveto /FSIjessica-Regular findfont 12 scalefont setfont str2 show
30 180 moveto /FSTekton findfont 12 scalefont setfont str2 show
30 200 moveto /FSFutura findfont 12 scalefont setfont str2 show
30 220 moveto /FSArhaicNormal findfont 12 scalefont setfont str6 show
30 240 moveto /FSAntiqueOliveRoman findfont 12 scalefont setfont str2 show
30 260 moveto /FSBodoni findfont 12 scalefont setfont str2 show
30 280 moveto /FSCourier findfont 12 scalefont setfont str2 show
30 300 moveto /FSGlagoljica findfont 12 scalefont setfont str7 show
30 320 moveto /FSMilosav findfont 12 scalefont setfont str1 show
30 340 moveto /FSMistral findfont 12 scalefont setfont str1 show
30 360 moveto /FS0cr-A findfont 12 scalefont setfont str4 show
30 380 moveto /FS0cb-B findfont 12 scalefont setfont str3 show
30 400 moveto /FSSymbol findfont 12 scalefont setfont str5 show
30 420 moveto /FSMoskva findfont 12 scalefont setfont str3 show
30 440 moveto /FSOptima findfont 12 scalefont setfont str2 show
30 460 moveto /FSStoneSans findfont 12 scalefont setfont str2 show
30 480 moveto /FSHelvetica findfont 12 scalefont setfont str2 show
showpage
```

500 \_\_\_\_\_

480 abcčćddefghijklmnoprsšžABCČĆDDEFGHIJKLMNOPRSŠŽ0123456789,!:

460 abcčćddefghijklmnoprsšžABCČĆDDEFGHIJKLMNOPRSŠŽ0123456789,!:

440 abcčćddefghijklmnoprsšžABCČĆDDEFGHIJKLMNOPRSŠŽ0123456789,!:

420 абцчдѐфгхопрсшжАБЦЧТДѐЕФГХОПРСШЖ0123456789,!:

400 αβχδεφγηιφκλμνοπθρστυωξψζΔΦΓθΛΠΘΣςΩΞΨ

380 abcčćddefghopršžABCČĆDDEFGHOPRSŠŽ0123456789,!:

360 abcdefghopršABCDEFGHOPRS0123456789,!:

340 *abcčćddefghijklmnoprsštuvzžABCČĆDDEFGHIJKLMNOPRSŠTUVZŽ0123456789,!:*

320 abcčćddefghijklmnoprsštuvzžABCČĆDDEFGHIJKLMNOPRSŠTUVZŽ0123456789,!:

300 аБГВДЕЖЗІЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯабвгдежзіїйклмнопрстуфхцчшщъыьэюя

280 abcčćddefghijklmnoprsšžABCČĆDDEFGHIJKLMNOPRSŠŽ0123456789,!:

260 abcčćddefghijklmnoprsšžABCČĆDDEFGHIJKLMNOPRSŠŽ0123456789,!:

240 abcčćddefghijklmnoprsšžABCČĆDDEFGHIJKLMNOPRSŠŽ0123456789,!:

220 еЗЙИЛМОПкУТФЦХЪЫЬѸЯГЁЖШГЖЦѢГГЪЛьЬкхЬКѢѢДГѸЙѢЦ

200 abcčćddefghijklmnoprsšžABCČĆDDEFGHIJKLMNOPRSŠŽ0123456789,!:

180 abcčćddefghijklmnoprsšžABCČĆDDEFGHIJKLMNOPRSŠŽ0123456789,!:

160 abcčćddefghijklmnoprsšžABCČĆDDEFGHIJKLMNOPRSŠŽ0123456789,!:

140 **abcčćddefghijklmnoprsšžABCČĆDDEFGHIJKLMNOPRSŠŽ0123456789,!:**

120 abcčćddefghijklmnoprsšžABCČĆDDEFGHIJKLMNOPRSŠŽ0123456789,!:

100 abcčćddefghopršžABCČĆDDEFGHOPRSŠŽ0123456789,!:

80 abcčćddefghijklmnoprsšžABCČĆDDEFGHIJKLMNOPRSŠŽ0123456789,!:

60 abcčćddefghijklmnoprsšžABCČĆDDEFGHIJKLMNOPRSŠŽ0123456789,!:

40 **abcčćddefghijklmnoprsšžABCČĆDDEFGHIJKLMNOPRSŠŽ0123456789,!:**

20 *abcčćddefghijklmnoprsšžABCČĆDDEFGHIJKLMNOPRSŠŽ0123456789,!:*

## *scalefont* *get*

```
[a1 a2 ...] j get
```

Čitljivost ovisi o pravilnom odabiru visine slova. Savjetujemo relaciju odnosa visine slova i udaljenosti gledanja pisma. Visina slike verzala neka je tri milimetra ako čitamo tekst s udaljenosti 30 centimetara (četverac cicera je  $4,51 \text{ mm} = 300 \text{ mm} / 133$ , slika verzala:  $4.51 * 0.70 = 3.15 \text{ mm}$ ). Taj grubi savjet namjerno dajemo kako bi se zapamtilo da je odnos visine slike verzala i udaljenosti čitanja 1:100. Na primjer, na plakatu kojeg gledamo s udaljenosti 20 metara, visina verzala A, B, C... treba biti 20 centimetra a kurenta a, c, e... 10 centimetra.

Sličan savjet dajemo i za širinu retka. Zadaje se indirektno: neka se redak sastoji od 50 slovnih znakova. U slovne znakove ubraja se i razmak između riječi, štoviše, razmak između riječi je najčešći slovni znak.

Prikazan program demonstrira mogućnost dohvaćanja parametra za visinu `scalefont` preko polja (vektora). Brojač `for` petlje se preuzima u `j` varijablu (`/j exch def`) koju koristimo kao indeks člana polja u `get` naredbi. Iza svakog ispisa sa `show` naredbom se izvršava translaticiranje ishodišta koje se prije ulaska u petlju postavilo na (30, 480). Prelaženje u novi redak (`y` okomita koordinata naredbe `translate`) funkcijski se povezalo sa brojačem petlje kako bi ispis što bolje ispunio prostor budući da se veličina pisma dinamički povećava prelaženjem u slijedeći redak.

```
%program SKALIRANJE FONTOVA
/str1 (ačćđegijlnštuvzžAČĆĐEFGIJLMŠTUŽ0123456789,!:. ) def
30 480 translate
0 1 14 {/j exch def 0 0 moveto
/FSHelvetica findfont
[5 6 7 8 9 10 12 14 18 24 32 36 48 60 72] j get
scalefont setfont str1 show
0 j neg 4 mul 5 neg add translate
} for
showpage
```

500

480 aććđegijlnštuvzžAČĆĐEFGIJLMŠTUŽ0123456789,!..

aććđegijlnštuvzžAČĆĐEFGIJLMŠTUŽ0123456789,!..

460 aććđegijlnštuvzžAČĆĐEFGIJLMŠTUŽ0123456789,!..

aććđegijlnštuvzžAČĆĐEFGIJLMŠTUŽ0123456789,!..

440 aććđegijlnštuvzžAČĆĐEFGIJLMŠTUŽ0123456789,!..

aććđegijlnštuvzžAČĆĐEFGIJLMŠTUŽ0123456789,!..

420 aććđegijlnštuvzžAČĆĐEFGIJLMŠTUŽ0123456789,!..

aććđegijlnštuvzžAČĆĐEFGIJLMŠTUŽ0123456789,!..

400 aććđegijlnštuvzžAČĆĐEFGIJLMŠTUŽ0123456789,!..

aććđegijlnštuvzžAČĆĐEFGIJLMŠTUŽ0123456789,!..

380

360 aććđegijlnštuvzžAČĆĐEFGIJLMŠTUŽ0123456789,!..

aććđegijlnštuvzžAČĆĐEFGIJLMŠTUŽ0123456789,!..

340 aććđegijlnštuvzžAČĆĐEFGIJLMŠTUŽ0123456789,!..

aććđegijlnštuvzžAČĆĐEFGIJLMŠTUŽ0123456789,!..

320

300 aććđegijlnštuvzžAČĆĐEFGIJLMŠTUŽ0123456789,!..

aććđegijlnštuvzžAČĆĐEFGIJLMŠTUŽ0123456789,!..

280

260 aććđegijlnštuvzžAČĆĐEFGIJLMŠTUŽ0123456789,!..

aććđegijlnštuvzžAČĆĐEFGIJLMŠTUŽ0123456789,!..

240

220 aććđegijlnštuvzžAČĆĐEFGIJLMŠTUŽ0123456789,!..

aććđegijlnštuvzžAČĆĐEFGIJLMŠTUŽ0123456789,!..

200

180 aććđegijlnštuvzžAČĆĐEFGIJLMŠTUŽ0123456789,!..

aććđegijlnštuvzžAČĆĐEFGIJLMŠTUŽ0123456789,!..

160

140 aććđegijlnštuvzžAČĆĐEFGIJLMŠTUŽ0123456789,!..

aććđegijlnštuvzžAČĆĐEFGIJLMŠTUŽ0123456789,!..

120

100 aććđegijlnštuvzžAČĆĐEFGIJLMŠTUŽ0123456789,!..

aććđegijlnštuvzžAČĆĐEFGIJLMŠTUŽ0123456789,!..

80

60 aććđegijlnštuvzžAČĆĐEFGIJLMŠTUŽ0123456789,!..

aććđegijlnštuvzžAČĆĐEFGIJLMŠTUŽ0123456789,!..

40

20



## *charpath*

`string sud charpath`

Slova u računarskoj grafici, još od fotosloga treće generacije, određena su putanjom ovojnice na različite načine: pravci, dijelovi kružnice. PostScript koristi Bezierovu stazu. Slovni znak najčešće se prikazuje kao popunjen prostor omeđen unutarnjom i vanjskom ovojnicom. Za to nam je dovoljna komanda `show`.

Slova se mogu ispisivati tipa "outline" tj. samo linijama koje leže na Bezierovim putanjama. Pri tome se mora zadati debljina linije. Naredbom `charpath` stvaraju se ovojnice slova zadanog stringa koje će se prikazati tek upotrebom naredbe `stroke`.

Sa logičkim sudom *true* ili *false* definira se vrsta outline koja će se dobiti. To ovisi o vrsti fonta koje želimo pretvoriti u outline. Postoje fontovi koji su definirani upravo kao outline sa PostScript programom koji se popune (`fill`) u momentu ispisivanja, nadalje fontovi čiji su znakovi definirani nezatvorenim linijama ili fontovi definirani bitmapom. Danas se upotrebljava samo prva vrsta fonta (outline font) za koju je rezultat naredbe `charpath` identičan i za *false* i *true* logički sud pa ćemo upotrebljavati u svim primjerima *false* parametar.

U našim primjerima ispisana su outline slova sa posve tankim linijama i linijama koje su deblje i od nekih dijelova slova. Linija se iscrtava tako da je središte debljine linije na položaju Bezierove staze - debljina se širi okomito na tu stazu.

Varijabla `j` se u svakom krugu petlje puni sa trenutnom vrijednošću brojača petlje i sa njom se, kao indeksom polja, dohvaća slijedeći parametar naredbe `setlinewidth`. Ispis svakog slijedećeg primjera u petlji se translatira po `y` koordinati za -50.

```
%program OVOJNICE SLOVNIH ZNAKOVA
/tekst (Informacijski dizajn) def
/FSTimesRom findfont 50 scalefont setfont
30 480 translate 0 0 moveto tekst show
0 1 8 {/j exch def 0 50 neg moveto
tekst false charpath
[0.3 0.5 0.8 1 1.5 2 3 5 8 10] j get
setlinewidth
stroke
0 50 neg translate
} for
showpage
```

500 Informacijski dizajn  
480 Informacijski dizajn  
460 Informacijski dizajn  
440 Informacijski dizajn  
420 Informacijski dizajn  
400 Informacijski dizajn  
380 Informacijski dizajn  
360 Informacijski dizajn  
340 Informacijski dizajn  
320 Informacijski dizajn  
300 Informacijski dizajn  
280 Informacijski dizajn  
260 Informacijski dizajn  
240 Informacijski dizajn  
220 Informacijski dizajn  
200 Informacijski dizajn  
180 Informacijski dizajn  
160 Informacijski dizajn  
140 Informacijski dizajn  
120 Informacijski dizajn  
100 Informacijski dizajn  
80 Informacijski dizajn  
60 Informacijski dizajn  
40 Informacijski dizajn  
20 Informacijski dizajn

## *charpath*

Višeslojno iscrtavanje ovojnice s različitim debljinama, zacrnjenjem, bojama i redosljedom prikaza, omogućuje dizajneru veoma kompleksna rješenja. Charparh se može kombinirati sa linijama bilo kojeg geometrijskog lika kao naprimjer kvadrat, krug, slobodna Bezierova linija. Treba voditi računa da se linija proširuje okomito na njenu putanju jednako prema centru i prema van.

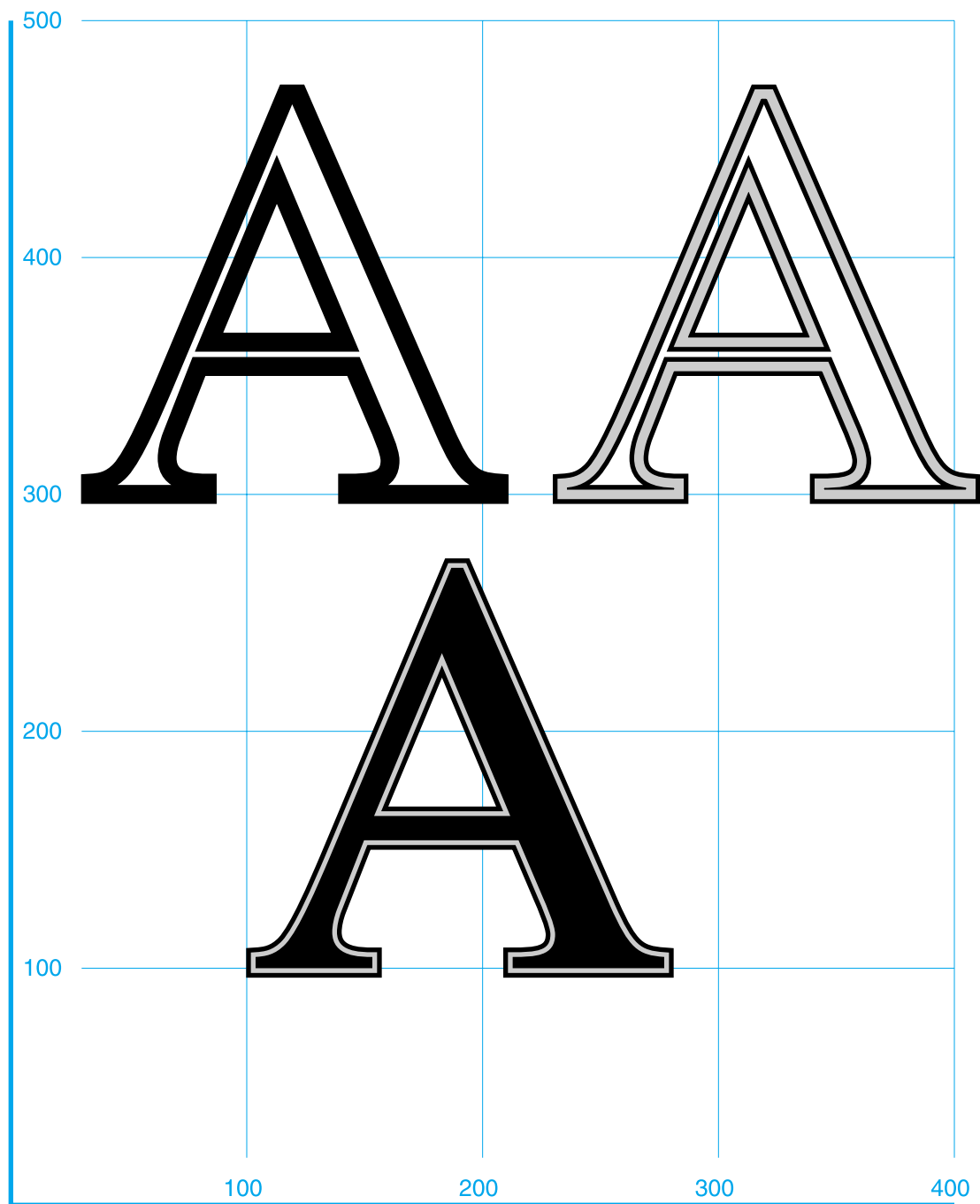
Treći znak je završni dizajn. Prethodna dva su faze njegove izrade. Prvi znak je prvi sloj gdje je crnom bojom otisnut outline slova A s debljinom linije od 8 točaka. Na taj sloj se u drugom primjeru dodao isti outline samo sa 30% sive i debljinom 4 točke. Konačno treći primjer se dobio dodavanjem crnog sloja s ispunjenim outlineom (`fill`) na prethodna dva sloja.

```
%program OVOJNICA U OVOJNICI
/str (A) def
/FSTimesRom findfont 250 scalefont setfont

gsave 30 300 translate
%Prvo slovo
0 0 moveto str false charpath 8 setlinewidth stroke

%Drugo slovo
%Prvi sloj
200 0 moveto str false charpath 8 setlinewidth stroke
%Drugi sloj
200 0 moveto str false charpath 4 setlinewidth
0.7 setgray stroke
grestore

gsave 100 100 translate
%Treće slovo
%Prvi sloj
0 0 moveto str false charpath 8 setlinewidth stroke
%Drugi sloj
0 0 moveto str false charpath 4 setlinewidth
0.7 setgray stroke
%Treći sloj
0 0 moveto str false charpath 0 setgray fill
grestore
showpage
```



## *clip newpath*

`clip  
newpath`

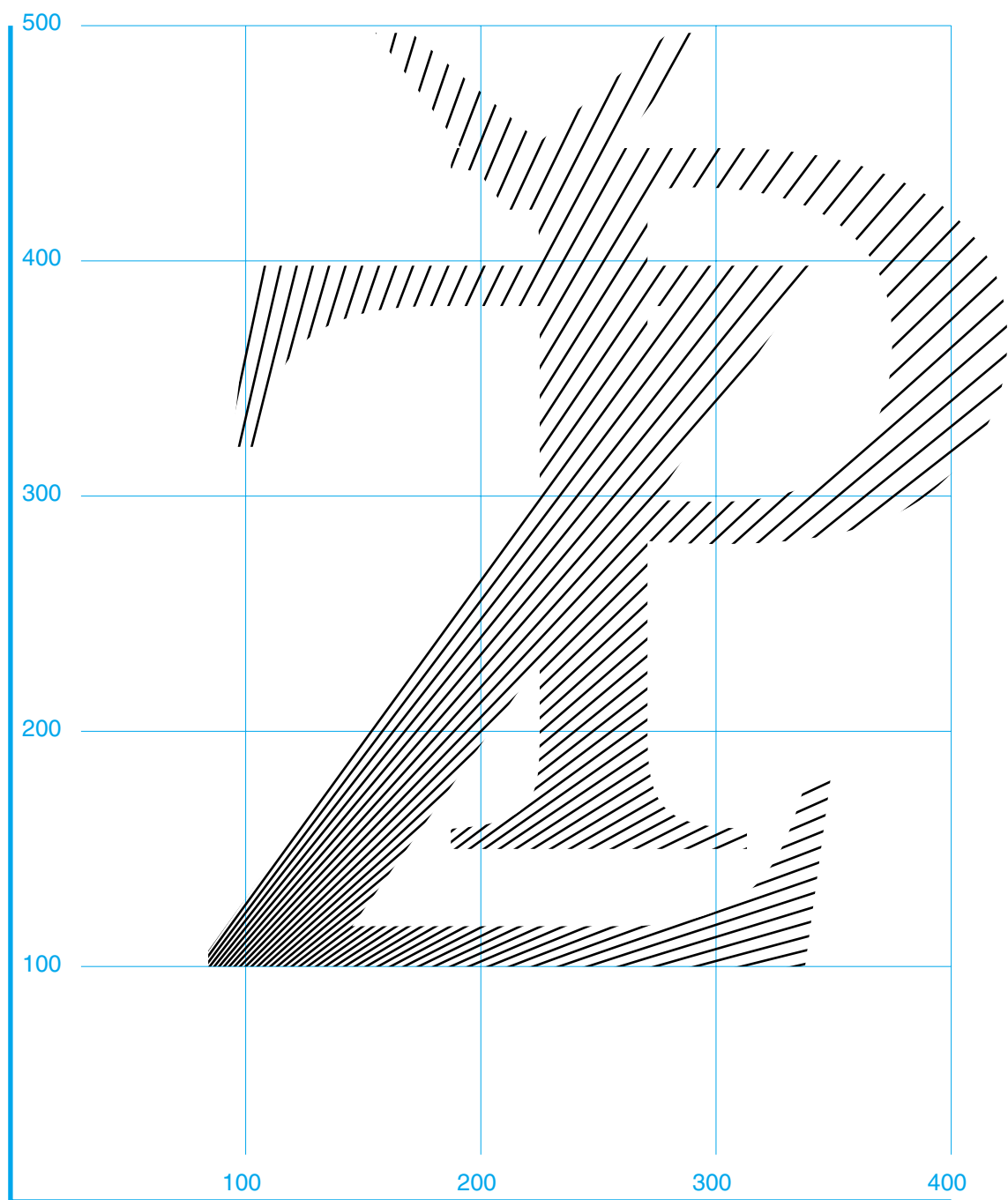
Popunjavanje zatvorenog puta nekim grafičkim rješenjem realizira se komandom `clip` bez obzira da li je osnova grafički lik ili ovojnica slova. Prema toj naredbi možemo se odnositi kao maski, prozoru ili izrezu za neki dizajn. Kada se iza programiranih zatvorenih staza upotrijebi naredba `clip` sve ono što će se nadalje programirati biti će vidljivo samo kroz stvorenu masku.

Naredba `clip` ne stvara iza sebe mogućnost programiranja početka nove staze kao što rade naredbe `fill` i `stroke` već ako se to želi mora se iza tih naredbi upotrijebiti eksplicitno naredba `newpath`. U našem primjeru, bez naredbe `newpath`, vidjela bi se `clip` staza nakon završne upotrebe naredbe `stroke`.

Nakon što se isprogramirala `clip` maska od slova `P` i `Ž` slijedi program koji crta linije rotirane u svakom koraku `repeat` petlje za jedan stupanj. `Repeat` petlja se vrti 90 puta pa se dobije jedna četvrtina kruga pomoću linija iz jednog žarišta. To je vidljivo samo kroz definiranu masku.

```
%program CLIP PATH
80 100 moveto
/FSTimesRom findfont 450 scalefont setfont
(Ž) false charpath

180 150 moveto
(P) false charpath
clip
newpath %da se ne vidi outline zbog završnog stroke
1 setlinewidth
30 30 translate
90 {
0 0 moveto 550 0 rlineto
1 rotate
} repeat
stroke
showpage
```



## *strokepath*

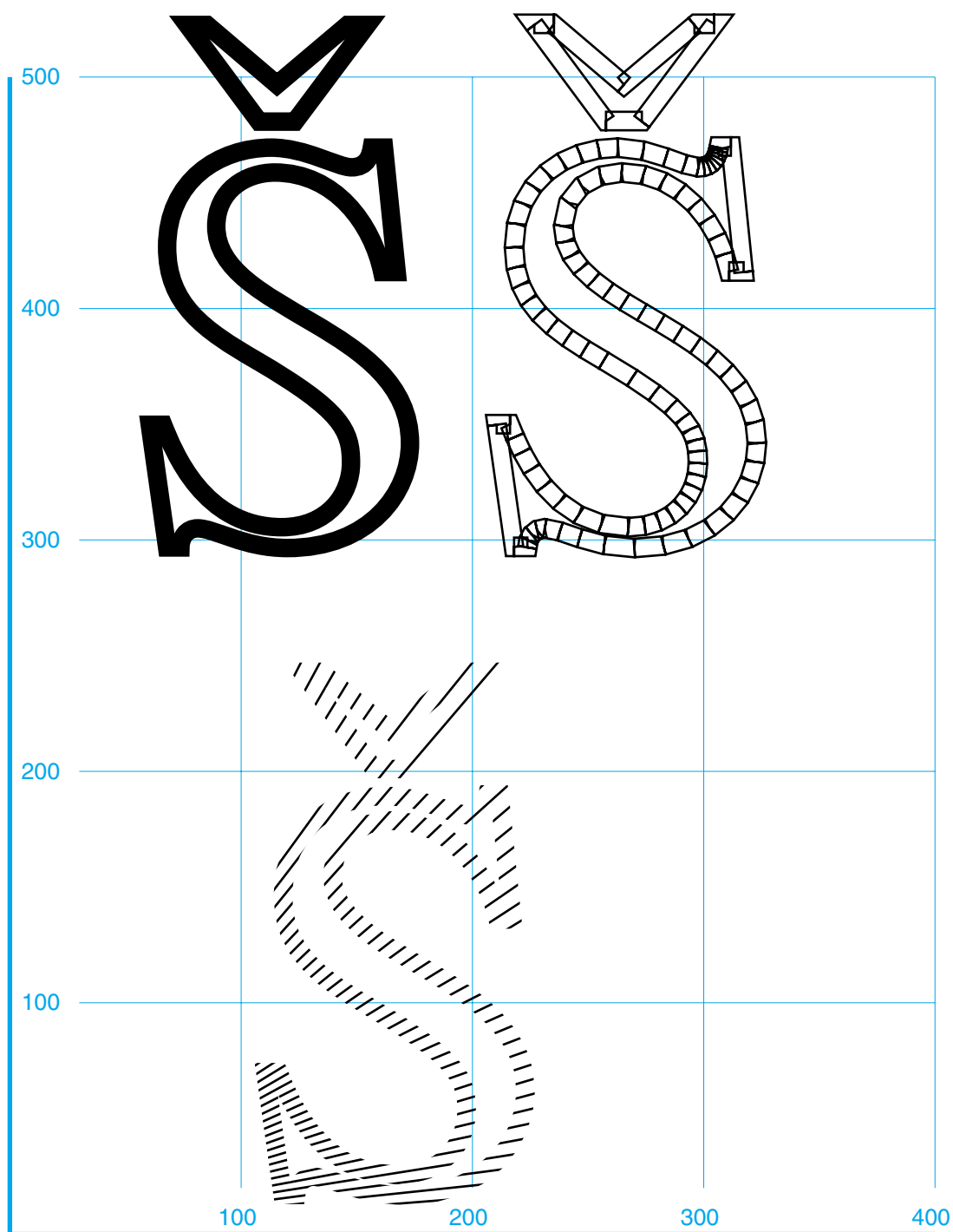
`strokepath`

Komanda `strokepath` omogućuje određivanje ovojnice linija, tj. stazu oko linije koja ima zadanu debljinu. Ako su linije nastale komandom `charpath` tada komanda `strokepath` daje nove površine oko linija koje su tako nastale. Ta se površina može popunjavati, bojati, podlagati (komande `fill` ili `clip`). Staze nastale komandom `strokepath` sastoje se od poligona kao što je prikazano na drugom primjeru iscrtavanjem tih poligona. Poligoni se sastoje od okomica na početnu liniju i dužina koje su paralelne sa početnom linijom. Što je početna linija više zaobljena to su poligoni gušći i uži. Zbog toga nije prikladno iza te naredbe upotrijebiti naredbu `stroke` već prvenstveno `fill` ili `clip`. Nakon aktiviranja komande `strokepath` znatno se povećava potreba za memorijom pa se predlaže što prije iscrtavanje takve površine kako bi se očistio stack.

```
%program CLIP STROKEPATH
/FSTimesRom findfont 250 scalefont setfont
gsave
50 300 moveto 8 setlinewidth
(Š) false charpath stroke

200 300 moveto 8 setlinewidth
(š) false charpath
strokepath 1 setlinewidth stroke
grestore

gsave
100 20 moveto 8 setlinewidth
(Š) false charpath
strokepath
clip newpath
1 setlinewidth
0 0 moveto
180 {
0 0 moveto 400 0 lineto
1.5 rotate
} repeat
stroke
grestore
showpage
```





## *makefont*

Komanda `makefont` transformira font u drugi font po transformacijskom polju (matrici). Transformacijsko polje definira linearnu transformaciju koordinatnog para  $(x, y)$  u  $(x', y')$  na ovaj način:

$$x' = ax + cy + t_x; \quad y' = bx + dy + t_y.$$

`font [a b c d tx ty] makefont`

Komande `translate`, `scale`, `rotate` i `scalefont` definiraju se preko transformacijskog polja na ovaj način:

`10 20 translate => [1 0 0 1 10 20]`

$$x' = x + 10; \quad y' = y + 20$$

`2 4 scale => [2 0 0 4 0 0]`

$$x' = 2x; \quad y' = 4y$$

`45 rotate => [cos(45) sin(45) -sin(45) cos(45) 0 0]`

$$x' = \cos(45)x - \sin(45)y; \quad y' = \sin(45)x + \cos(45)y$$

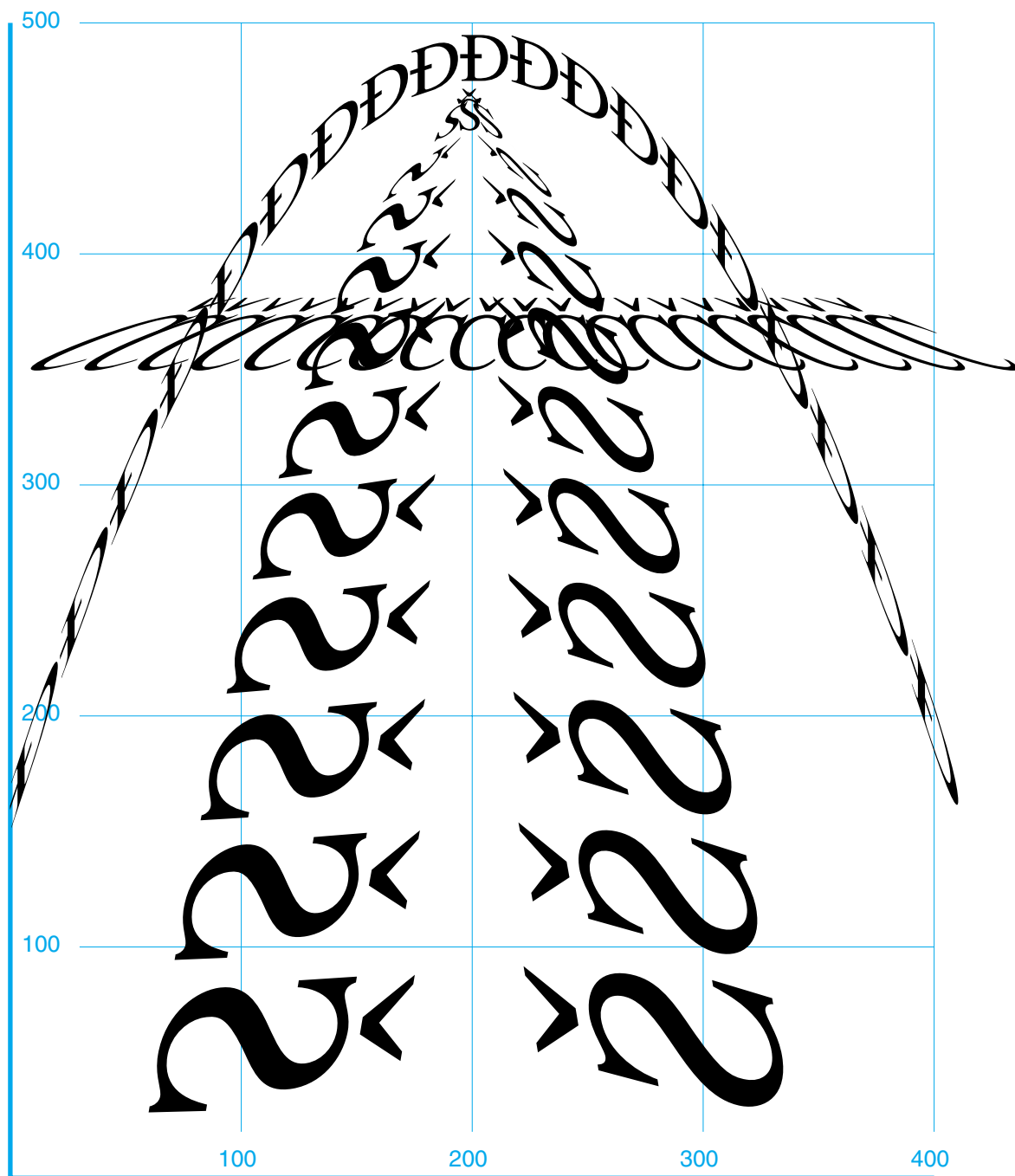
`30 scalefont => [30 0 0 30 0 0] makefont`

Transformacija fonta po transformacijskom polju izvodi se tako da se transformira koordinatni sustav četverca svakog znaka u fontu po navedenim formulama, a postojeći koordinatni sustav ostaje netaknut. Sa `makefont` naredbom stvaraju se novi deformirani fontovi. Početak slijedećeg znaka definira se pomakom nakon naredbe `show`.

```
%program MAKEFONT
0 150 moveto
90 -10 -90 {/j exch def
/FSTimesRom findfont [30 j 0 30 0 0 ] makefont setfont
(Đ) show % x'=30x; y'=jx+30y
} for

0 350 moveto
90 -10 -90 {/j exch def
/FSTimesRom findfont [35 0 j 35 0 0 ] makefont setfont
(Č) show % x'=35x+jy; y'=35y
} for

60 20 moveto
120 -10 -120 {/j exch def
/FSTimesRom findfont [20 j j 20 0 0 ] makefont setfont
(Š) show % x'=20x+jy; y'=jx+20y
} for
showpage
```



## *ashow*

Ako želimo vodoravno spacionirati tekst možemo to raditi sa naredbom **ashow**. Parametri **dx** i **dy** su relativni pomaci po x odnosno y osi u odnosu na pismovnu liniju za svaki znak iz zadanog stringa. Kada je definiran samo **dx** parametar, a **dy** je 0 tada se radi o horizontalnom spacioniranju.

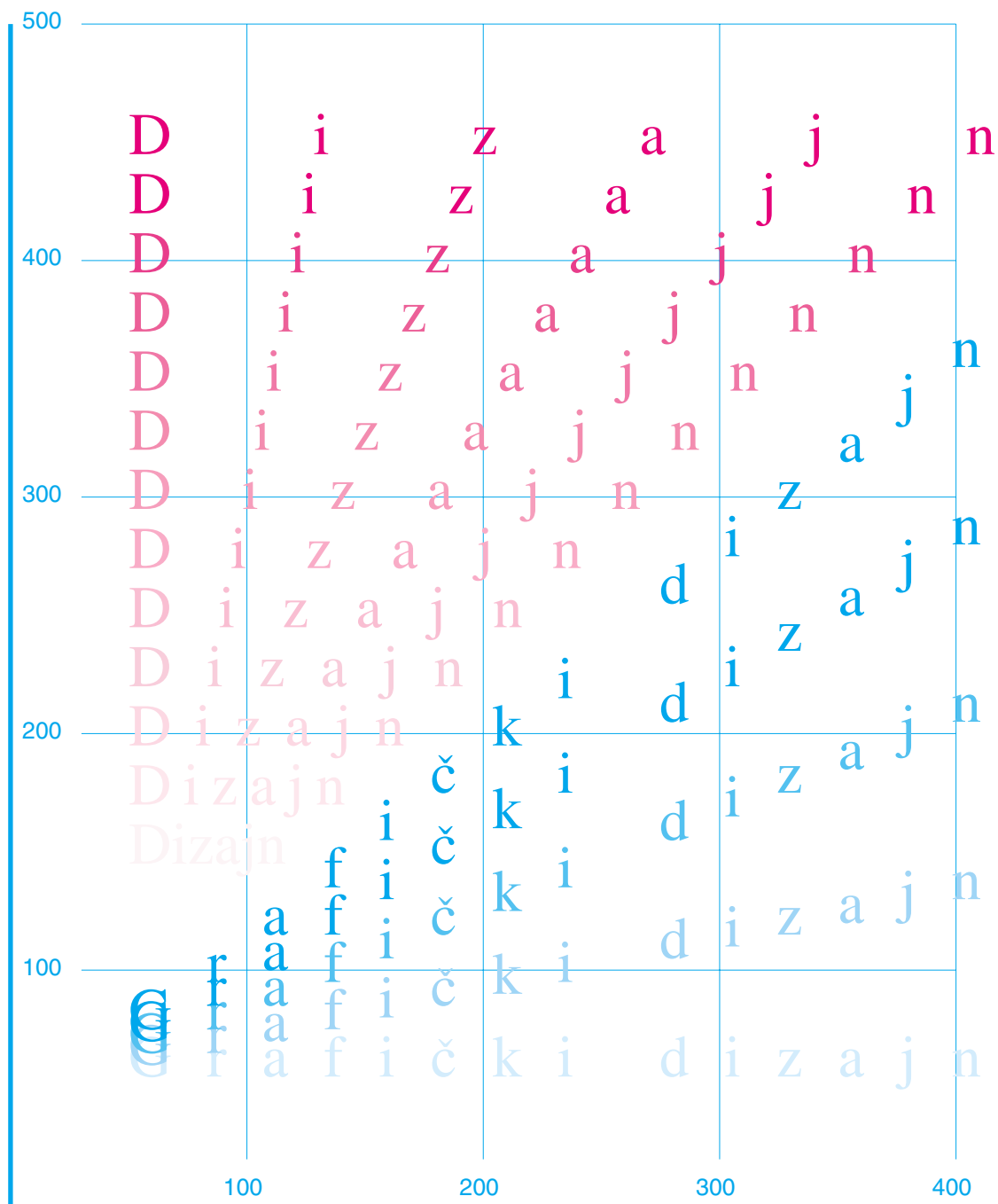
Naredbe **j 0 (Dizajn) ashow**

**15 j (Grafčki dizajn) ashow**

**dx dy string ashow**

stavile su se u for petlje čiji su indeksi ujedno i **dx** odnosno **dy** parametri **ashow** naredbe. Zajedno sa tim naredbama u petlji se vrtila naredba **setcmykcolor** čiji se parametar magente odnosno cijana mjenjao u svakom krugu petlje sa varijablom **brojac** koja se mijenja sa varijablom **korak**.

```
%program ASHOW
/FSTimesRom findfont 25 scalefont setfont
gsave
50 145 moveto
1 12 div /korak exch def
/brojac korak def
0 5 60 {/j exch def
0 brojac 0 0 setcmykcolor
gsave
j 0 (Dizajn) ashow
grestore
0 25 rmoveto
brojac korak add /brojac exch def
} for
grestore
gsave
50 55 moveto
1 4 div /korak exch def
/brojac korak def
0 5 20 {/j exch def
brojac 0 0 0 setcmykcolor
gsave
15 j (Grafčki dizajn) ashow
grestore
0 5 rmoveto
brojac korak add /brojac exch def
} for
grestore
showpage
```



## *length*

## *stringwidth*

## *ashow*

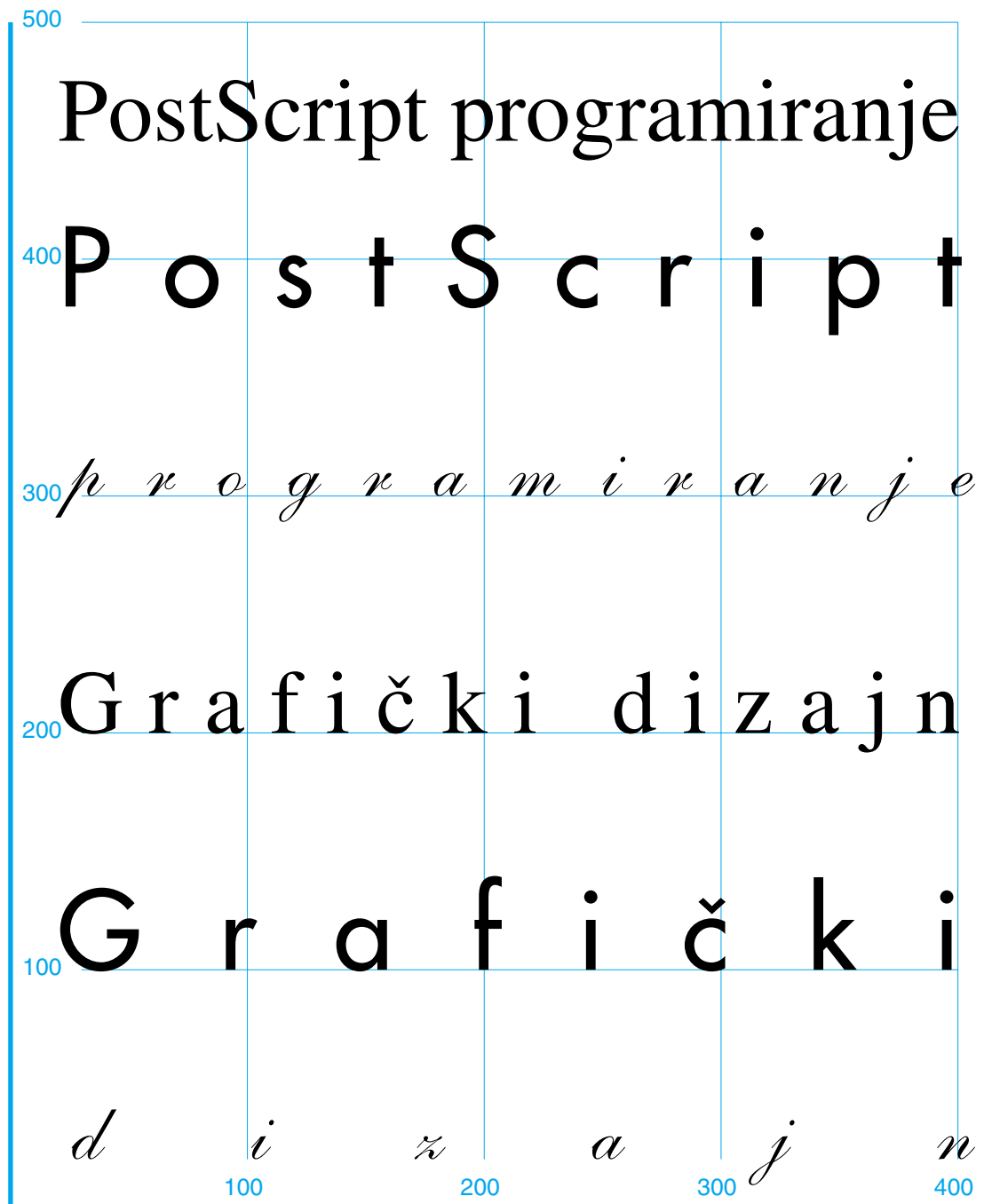
`string` length  
`string` stringwidth

Ovaj program pokazuje kako se može proširiti neki tekst na zadani puni format povećavajući razmak između slova. Ovdje je format definiran sa širinom prvog stringa (PostScript programiranje) koji se dobio sa naredbom `string1 stringwidth`. Naredba `stringwidth` daje zbroj širine svih debljinskih vrijednosti u stringu zadanog fonta odnosno jednak je relativnom pomaku tekuće pozicije na pismovnoj liniji koji bi se dogodio nakon naredbe `show`. Na stacku ostaje `x` i `y` relativni pomak. Pošto se naši znakovi nižu vodoravno, za razliku od npr. japanskih koji se nižu vertikalno, `y` relativni pomak treba zanemariti pa se iza svake upotrebe naredbe `stringwidth` upotrebljava `pop` naredba za skidanje `y` parametra.

Algoritam ovog problema je: 1. Napravi razliku zadanog formata i izmjerene širine stringa zadanog fonta; 2. Razlika se podjeli sa brojem razmaka između znakova (broj znakova-1); 3. Rezultat dijeljenja je `dx` parametar `ashow` naredbe.

Naredba `length` ostavlja na stacku broj znakova zadanog stringa. Da bi se lakše programiralo, prethodno su definirane varijable `w` (širina stringa) i `l` (broj znakova u stringu).

```
%program STRINGWIDTH
/font1 {/FSTimesRom findfont 38 scalefont setfont} def
/font2 {/FSFutura findfont 45 scalefont setfont} def
/font3 {/FSShellyAllegro findfont 35 scalefont setfont} def
/string1 (PostScript programiranje) def
/string2 (PostScript) def /string3 (programiranje) def
/string4 (Grafički dizajn) def /string5 (Grafički) def
/string6 (dizajn) def
/w2 {string2 stringwidth pop} def /l2 {string2 length} def
/w3 {string3 stringwidth pop} def /l3 {string3 length} def
/w4 {string4 stringwidth pop} def /l4 {string4 length} def
/w5 {string5 stringwidth pop} def /l5 {string5 length} def
/w6 {string6 stringwidth pop} def /l6 {string6 length} def
font1 string1 stringwidth pop /w1 exch def
20 450 moveto string1 show
20 380 moveto font2 w1 w2 sub l2 1 sub div 0 string2 ashow
20 300 moveto font3 w1 w3 sub l3 1 sub div 0 string3 ashow
20 200 moveto font1 w1 w4 sub l4 1 sub div 0 string4 ashow
20 100 moveto font2 w1 w5 sub l5 1 sub div 0 string5 ashow
20 20 moveto font3 w1 w6 sub l6 1 sub div 0 string6 ashow
showpage
```



## *widthshow*

## *xyshow*

## *awidthshow*

```
dx dy kod str widthshow
str [dx1 dy1 dx2 dy2...]xyshow
cx cy kod dx dy str awidthshow
```

Ovdje su prikazani primjeri koji demonstriraju još tri načina kontrole ispisa znakova iz zadanog stringa. Naredba **widthshow** mijenja poziciju ispisa slijedećeg znaka za dx i dy iza znaka koji je definiran dekadskim ASCII kodom.

U prvom primjeru u riječi POSTSCRIPT dolazi do relativnog pomaka za dx=30 i dy=15 iza svakog pojavljivanja slova S (dekadski ASCII kod 83). Drugi primjer dodaje 25 točaka vodoravno iza svakog pojavljivanja znaka razmaka između riječi (kod 32). To je pogodno kada želimo formatirati tekst na puni format samo sa povećavanjem razmaka između riječi.

Treći primjer koristi naredbu **xyshow** koja iza svakog znaka stringa vrši (dx,dy) pomak definiran parom točaka iz polja. Zbog toga mora broj članova polja biti točno dvostruko veći od broja znakova stringa. Ukoliko nije dobiti će se greška.

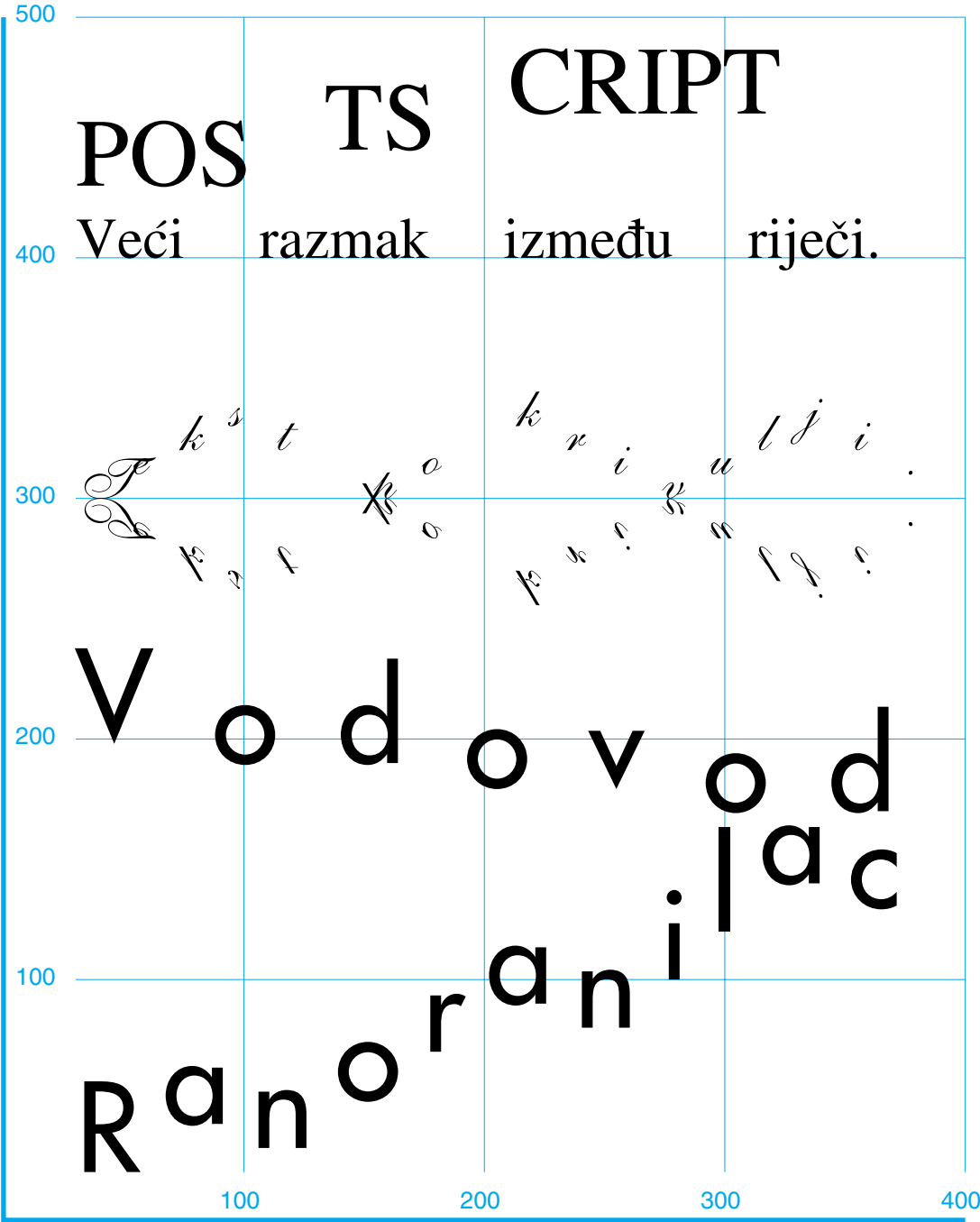
Komanda **awidthshow** je kombinacija **ashow** i **widthshow** naredbe. Sa dx i dy se mijenja relativna pozicija za sve znakove u stringu, a sa cx i cy se još dodatno mijenja pozicija iza znaka definiranog dekadskim ASCII kodom. U primjeru sa riječi "Vodovod" su svi znakovi sa pomakom dx=25, dy=-10, dok se iza znaka o (kod 111) vrši pomak jedino za cy=10. Tako se dobio efekt da se znak ispisuju vodoravno samo onda kada se pojavilo slovo o, inače za sva ostala slova koso prema dolje. Primjer sa riječi Ranoranilac pokazuje kako izgleda efekt kada se iza slova a (kod 97) vrši veći okomiti pomak prema dolje (cy=-30) nego što je okomiti pomak prema gore (dy=20) za sve znakove u riječi.

```
gsave
/F {ffindfont exch scalefont setfont} bind def
30 430 moveto 40 /FSTimesRom F 30 15 83 (POSTSCRIPT) widthshow

30 400 moveto 24 /FSTimesRom F
25 0 32 (Veći razmak između riječi.) widthshow

/proc1 {(Tekst po krivulji.)
[20 10 20 10 20 10 20 -10 20 -10 20 -10 20 10 20 10 20 10
20 -10 20 -10 20 -10 20 10 20 10 20 10 20 -10 20 -10 20 -10 ]
xyshow} bind def
30 300 moveto 24 /FSShellyAllegro F gsave proc1 grestore
1 -1 scale proc1 grestore

30 200 moveto 50 /FSFutura F 0 10 111 25 -10 (Vodovod) awidthshow
30 20 moveto 0 -30 97 9 20 (Ranoranilac) awidthshow showpage
```





## *kshow*

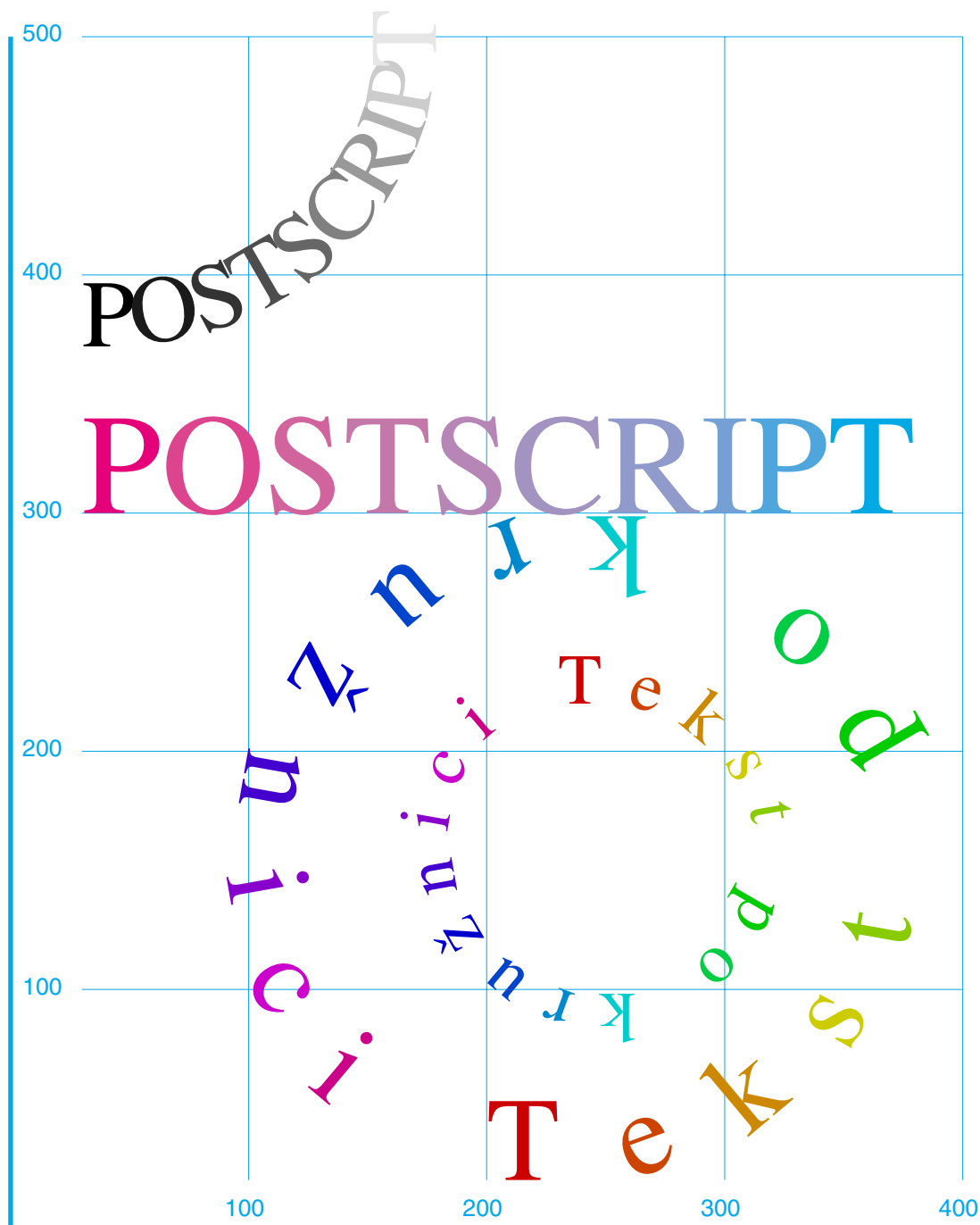
```
{proc} string kshow
```

Naredba `kshow` je naprednija od svih dosadašnjih naredbi koje se tiču kontrole ispisa znakova iz stringa. Ona daje mogućnost izvršavanja programske procedure između svakog znaka stringa i to na ovaj način: 1. Ispisuje se prvi znak stringa, a tekuća pozicija se pomakne sa širinu tog znaka; 2. Postavlja se na stack dekadski ASCII kod prvog znaka, a potom drugog znaka; 3. Izvršava se procedura `{proc}`; 4. Ispisuje se drugi znak... .

Kada se poziva procedura na stacku su dvije vrijednosti: kôd znaka koji je upravo ispisan i kôd slijedećeg znaka. Procedura se ne poziva prije nego što se prikaže prvi znak. Ako je broj znakova u stringu `n`, `kshow` poziva `n-1` puta proceduru.

U našim primjerima koristimo `kshow` naredbu tako da sa stacka skidamo kôdove na početku svake procedure (`pop pop`) jer ih ne koristimo. Pri svakom pozivu procedure, mijenjaju se obojenja i vrše se relativni pomaci i rotacije ispisa znakova da bi se postigli željeni efekti na našem primjeru.

```
%program KSHOW
/F {findfont exch scalefont setfont} bind def
gsave
40 /FSTimesRom F 30 370 moveto
/sivo 0 def
{pop pop /sivo sivo 0.1 add def sivo setgray 10 rotate}
(POSTSCRIPT) kshow %Iza svakog slova rotacija 10st.
grestore gsave
60 /FSTimesRom F 30 300 moveto
/sivo 0 def 0 1 0 0 setcmykcolor
{ pop pop /sivo sivo 0.1 add def /csivo 1 sivo sub def
sivo csivo 0 0 setcmykcolor} (POSTSCRIPT) kshow
grestore gsave
%Vanjska kružnica obrnuto od smjera sata
50 /FSTimesRom F 200 20 moveto
/H 0 def 1 18 div /korak exch def 0 1 0.8 sethsbcolor
{ pop pop 30 0 rmoveto 20 rotate /H H korak add def
H 1 0.8 sethsbcolor} (Tekst po kružnici) kshow
grestore gsave
%Unutarnja kružnica u smjeru sata
30 /FSTimesRom F 230 220 moveto
/H 0 def 1 18 div /korak exch def 0 1 0.8 sethsbcolor
{ pop pop 10 0 rmoveto -20 rotate /H H korak add def
H 1 0.8 sethsbcolor } (Tekst po kružnici) kshow
grestore showpage
```



## Operatori

PostScript posjeduje aritmetičke operatore (sub, add, mul, div...), stack operatore (exch, dup, pop...), relacijske operatore, logičke operatore, operatore uvjetnog izvršavanja procedura, operatore polja, string operatore i mnoge druge.

Pomoću relacijskih operatora se uspoređuju se dva elementa na stacku. Rezultat te usporedbe (relacije) je logički sud koji može biti istina ili laž i pojavljuje se na stacku sa riječima *true* ili *false*. Te logičke sudove najčešće koriste operatori uvjetnog grananja if i ifelse, kao i logički operatori not, and, or i xor.

Relacijski operatori i primjeri:

eq	jednako Ako je stanje na stacku: (graf) (graf), >vrh stacka nakon eq, stanje je: <i>true</i> .
ne	nije jednako Ako je stanje na stacku: 10 10, nakon ne, stanje je: <i>false</i> .
gt	veće od Ako je stanje na stacku: 20 10, nakon gt, stanje je: <i>true</i> .
ge	veće ili jednako od Ako je stanje na stacku: 20 20, nakon ge, stanje je: <i>true</i> .
lt	manje od Ako je stanje na stacku: 10 5, nakon lt, stanje je: <i>false</i> .
le	manje ili jednako od Ako je stanje na stacku: 5 5, nakon le, stanje je: <i>true</i> .

## Logički operatori:

and	"i"	stanje na stacku	rezultat na stacku
		<i>false false</i>	<i>false</i>
		<i>false true</i>	<i>false</i>
		<i>true false</i>	<i>false</i>
		<i>true true</i>	<i>true</i>
or	"ili"	stanje na stacku	rezultat na stacku
		<i>false false</i>	<i>false</i>
		<i>false true</i>	<i>true</i>
		<i>true false</i>	<i>true</i>
		<i>true true</i>	<i>true</i>
xor	"ex ili"	stanje na stacku	rezultat na stacku
		<i>false false</i>	<i>false</i>
		<i>false true</i>	<i>true</i>
		<i>true false</i>	<i>true</i>
		<i>true true</i>	<i>false</i>
not	"ne"	stanje na stacku	rezultat na stacku
		<i>false</i>	<i>true</i>
		<i>true</i>	<i>false</i>

## Operatori uvjetnog izvršavanja procedura (grananja):

sud {proc} if    Ako je sud *true* izvršava se procedura proc, a inače se preskače.

Nakon    *c 0.5 ge c 1.0 le and {/c 1 def} if*  
 stack je prazan, a varijabla c se napuni sa 1 ako je prethodni logički uvjet dao *true*.

sud {proc1} {proc2} ifelse    Ako je sud *true* izvršava se procedura proc1, a ako je bio *false* izvršava se proc2.

Nakon    *c 0.5 ge c 1 le and {/c 1 def} {/c 0 def} if*  
 stack je prazan, a varijabla c će biti napunjena sa 1 ako je prethodni logički uvjet dao *true*, odnosno sa 0 ako je bio *false*.

### Operatori polja:

`n array` stvara na stacku polje od `n` članova koji su svi null objekti

Nakon `2 array`  
stanje na stacku je: `[null null]`.

`[polje] length` daje na stacku broj članova polja

Nakon `[54.6 (grafika)] length`  
stanje na stacku je: `2`.

`[polje] j get` daje na stacku član na poziciji `j`  
(indeksiranje polja počinje sa 0)

Nakon `[43 34 0 0 (271CVC)] 0 get`  
stanje na stacku je: `43`.

`[polje] j član put` stavlja u polje novi član na poziciju `j`  
preko stare vrijednosti

Nakon `/A [2 3 5] def A 1 (ab) put`  
polje `A` izgleda ovako: `[2 (ab) 5]`.

`[polje] j d getinterval` daje na stacku novo polje stvoreno  
od `d` članova polja počevši od `j` pozicije

Nakon `[43 34 0 0 (271CVC)] 0 4 getinterval`  
stanje na stacku je: `[43 34 0 0]`.

`[polje1] j [polje2] putinterval` zamjenjuje dio polja1 sa  
članovima polja2 počevši od `j` pozicije

Nakon `/B [2 0 0 6] def B 1 [10 11] putinterval`  
polje `B` izgleda ovako: `[2 10 11 6]`.

`[polje] {proc} forall` izvršava se procedura `proc` za  
svaki član polja

Nakon `1 [2 6 3 1] {mul} forall`  
stanje na stacku je: `36`.

### Operatori stringa:

`n string` stvara na stacku string (niz znakova) od  
`n \000` kôdova. To je oktalni zapis za 0.

Nakon `4 string`  
stanje na stacku je: `(\000\000\000\000)`.

string length      daje na stacku broj znakova stringa  
     Nakon (dizajn) length  
     stanje na stacku je:      6.

string j get      daje na stacku dekadski ASCII kôd znaka  
     na poziciji j (indeksiranje stringa počinje sa 0)  
     Nakon (grafika) 2 get  
     stanje na stacku je:      97. (dek. ASCII kôd "a")

string j kôd put      stavlja u string novi znak definiran s  
     dekadski ASCII kôdom na poziciju j preko  
     stare vrijednosti  
     Nakon /A (rupa) def A 1 107 put  
     string A izgleda ovako:      (rkpa). (107 je dek. ASCII kôd "k")

string j d getinterval      daje na stacku novi string stvoren od d  
     znakova početnog stringa od j pozicije  
     Nakon (Zagreb) 2 3 getinterval  
     stanje na stacku je:      (gre).

string1 j string2 putinterval      zamjenjuje dio stringa1 sa  
     znakovima stringa2 počevši od pozicije j  
     Nakon /A (abcdef) def A 2 (rkl) putinterval  
     string A izgleda ovako:      (abrklf).

string {proc} forall      izvršava se proc za svaki znak stringa  
     dobivenog kao dekadski ASCII kôd.  
     Nakon /b 0 def  
     (grafika) {97 eq {b 1 add /b exch def} if} forall  
     u varijabli b je:      2.

string stringpogotka search

- ako je string pogotka nađen u stringu,  
     na stacku će biti stanje:  
     (postpogodak) (string pogotka) (predpogodak) *true*
- ako string pogotka nije nađen u stringu, na stacku će  
     biti stanje:      (string) *false*

Nakon (grafika) (af) search  
     stanje na stacku je:      (ika) (af) (gr) *true*.  
     Nakon (grafika) (c) search  
     stanje na stacku je:      (grafika) *false*.

***LTEKST******CTEKST******RTEKST***

```
string xl yl LTEKST
string xc yc CTEKST
string xr yr RTEKST
```

Ovdje su prikazane procedure kojima se isključuje redak u lijevo, sredinu i desno. Prva procedura **LTEKST** prikazuje tekst poravnat s lijeve strane počevši od zadane pozicije *xl*, *yl*. Ona se sastoji samo od **moveto** i **show** naredbe.

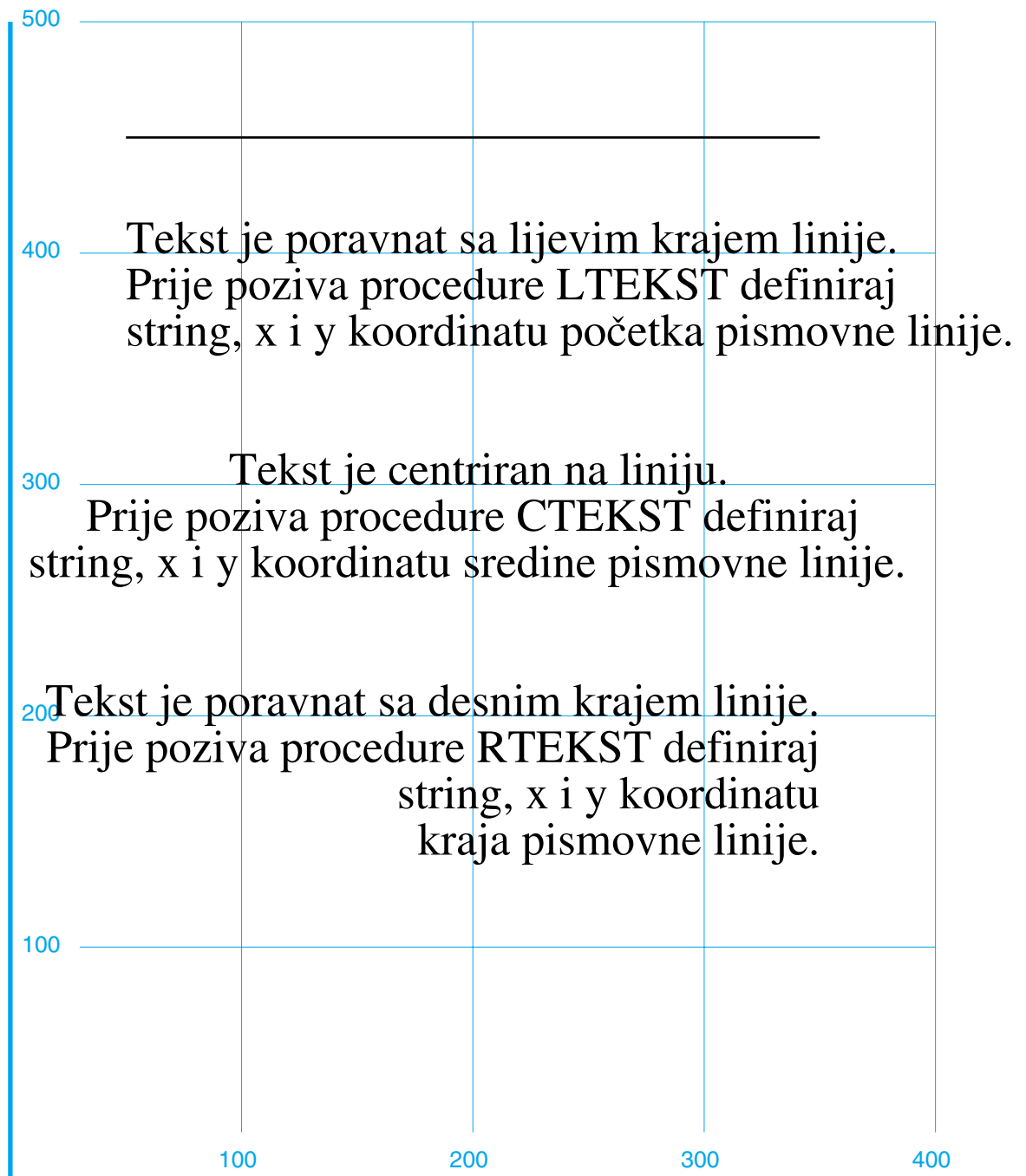
Procedura **CTEKST** centrirá tekst u odnosu na zadanu koordinatnu točku. Nakon postavljanja na poziciju *xc*, *yc* sa **moveto** naredbom vrši se negativni relativni pomak sa **rmoveto** po *x* koordinati za (širina stringa)/2. Dupliza **moveto** je potreban radi dupliciranja stringa na stacku jer ga potroši naredba **stringwidth**, a treba ga završiti **show**.

Treći primjer prikazuje proceduru **RTEKST** koja, nakon postavljanja teksta na poziciju *xr*, *yr* sa **moveto** vrši negativni relativni pomak teksta sa **rmoveto** po *x* koordinati za punu širinu stringa.

```
%program POZICIONIRANJE TEKSTA U ODNOSU NA ZADANU PISMOVNU LINIJU
/F {findfont exch scalefont setfont} bind def
/LTEKST
{moveto show} bind def
/CTEKST
{ moveto dup stringwidth pop neg 2 div 0 rmoveto show} bind def
%pop skida y parametar od stringwidth koji je bespotreban
/RTEKST
{moveto dup stringwidth pop neg 0 rmoveto show} bind def
20 /FSTimesRom F
50 450 moveto 300 0 rlineto stroke %nacrtana pismovna linija
(Tekst je poravnat sa lijevím krajem linije.) 50 400 LTEKST
(Prije poziva procedure LTEKST definiraj) 50 380 LTEKST
(string, x i y koordinatu početka pismovne linije.) 50 360 LTEKST

( Tekst je centriran na liniju.) 200 300 CTEKST
(Prije poziva procedure CTEKST definiraj) 200 280 CTEKST
(string, x i y koordinatu sredine pismovne linije.) 200 260 CTEKST

(Tekst je poravnat sa desnim krajem linije.) 350 200 RTEKST
(Prije poziva procedure RTEKST definiraj) 350 180 RTEKST
(string, x i y koordinatu) 350 160 RTEKST
(kraja pismovne linije.) 350 140 RTEKST
showpage
```





## JTEKST

string d xl yl JTEKST

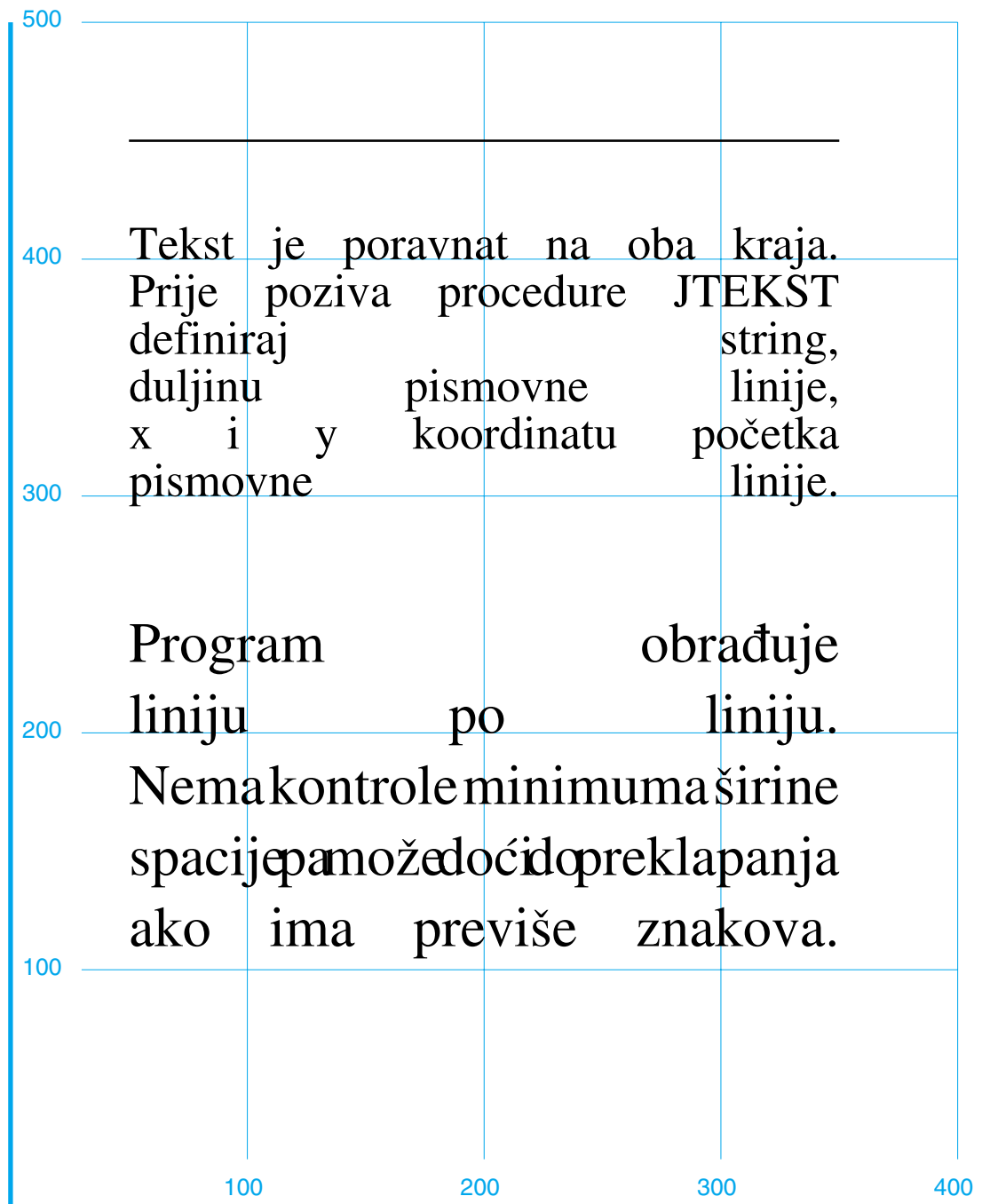
Ovdje je prikazana procedura JTEKST koja isključuje, odnosno poravnava zadani tekst na lijevu i desnu stranu pismovne linije korigirajući širinu razmaka između riječi. Prije upotrebe te procedure potrebno je na stack staviti string, duljinu pismovne linije i koordinate početka pismovne linije. Nakon izvršavanja naredbe `forall` u varijabli `brojpac` nalazi se broj pojavljivanja razmaka između riječi (kôd 32) u tekstu. Potrebnu korekciju širine razmaka između riječi sadrži varijabla `dspacija` koja se dobila formulom

$$(\text{duljina pismovne linije} - \text{širina stringa}) / \text{broj razmaka}.$$

Varijabla `dspacija` je `dx` parametar `widthshow` naredbe sa kojom se izvršava ispis teksta na način da se iza svake pojave razmaka između riječi (kôd 32) vrši relativni pomak tekuće pozicije za `dx=dspacija`, `dy=0`.

U glavnom programu poziva se procedura JTEKST za svaki ispis jednog tekstualnog retka. Prelazak u novi redak vrši se apsolutnim definiranjem koordinata početka nove pismovne linije za svako pozivanje JTEKST procedure.

```
/F {findfont exch scalefont setfont} bind def
/JTEKST
  {moveto /duljpislin exch def /tekst exch def
   tekst stringwidth pop /sirstringa exch def
   /brojac 0 def
   tekst
   {32 eq {brojac 1 add /brojac exch def} if
    } forall      %za svaki znak stringa izvršava se proc
   /brojpac brojpac def
   duljpislin sirstringa sub brojpac div /dspacija exch def
   dspacija 0 32 tekst widthshow } bind def      %32 je spacija
%%GLAVNI PROGRAM
20 /FSTimesRom F
50 450 moveto 300 0 rlineto stroke %nacrtana pismovna linija
(Tekst je poravnat na oba kraja.) 300 50 400 JTEKST
(Prije poziva procedure JTEKST) 300 50 380 JTEKST
(definiraj string,) 300 50 360 JTEKST
(duljinu pismovne linije,) 300 50 340 JTEKST
(x i y koordinatu početka) 300 50 320 JTEKST
(pismovne linije.) 300 50 300 JTEKST
24 /FSTimesRom F (Program obrađuje) 300 50 230 JTEKST
(liniju po liniju.) 300 50 200 JTEKST
(Nema kontrole minimuma širine) 300 50 170 JTEKST
(spacije pa može doći do preklapanja) 300 50 140 JTEKST
(ako ima previše znakova.) 300 50 110 JTEKST showpage
```



## ***SPOJI INSERT***

```
str1 str2 SPOJI
str vektor znak
```

```
INSERT
```

Stvaranjem biblioteke procedura obogaćujemo i olakšavamo programiranje složenih PostScript programa. Plavom bojom su označene procedure koje ćemo kasnije samo pozvati po imenu podrazumijevajući da smo ih u ovom obliku stavili ispred poziva. Da bi se takva procedura mogla pravilno upotrijebiti kao gotova naredba treba korisniku dati definiciju argumenata na stacku (ako je potrebno) prije upotrebe procedure i rezultata na stacku (ako ga daje) nakon upotrebe procedure.

Ovdje demonstriramo procedure SPOJI i INSERT. Procedura SPOJI nam služi za spajanje dva stringa u novi string. Prije poziva procedure SPOJI moraju biti dva stringa na stacku. Njihov poredak na stacku je bitan jer spajanje stringova nije komutativna operacija. Nakon izvršenja procedure na vrhu stacka nalazi se spojeni string, a početni stringovi su nestali sa stacka. Svaki put kada želimo da procedure, koje stvaramo, skidaju početne argumente na stacku, rješenje je sa preuzimanjem argumenata u lokalne varijable procedure kao na primjer u proceduri SPOJI sa naredbama `/S2 exch def /S1 exch def`.

Algoritam za spajanje stringova počinje prebrojavanjem znakova jednog i drugog stringa sa naredbom `length` da bi se mogao stvoriti inicijalni prazni string `wspoj` koji ima broj znakova jednak zbroju znakova ulaznih stringova. Njega punimo sa naredbom `putinterval`. Prvo se puni od pozicije 0 sa znakovima iz prvog stringa, a zatim sa znakovima iz drugog stringa od slijedeće pozicije definirane duljinom prvog stringa varijablom `L1`. Na kraju procedure stavljamo rezultat procedure na stack koji se nalazi u varijabli `wspoj`. U glavnom programu prikazan je primjer slaganje rečenice od stringova riječi uključujući i razmak između riječi koji se može stavljati u string na kraj riječi ili prije početka riječi.

Procedura INSERT ima zadatak da ubacuje definirani znak ili string u zadani string na pozicije definirane vektorom (poljem). Na primjer, ako je u vektoru broj dva, a znak je crtica, tada se crtica insertira (ubacuje) iza drugog znaka u stringu. Prije poziva procedure INSERT na stacku mora biti string, vektor i znak ili string. Nakon izvršavanja procedure na stacku će se nalaziti string sa ubačenim znakom. Nakon uzimanja pozicije ubacivanja iz vektora (varijabla `poz`) spaja se prethodno formirani string (procedura SPOJI) sa dijelom riječi definirane početkom u varijabli `poc` i duljinom u varijabli `interval`. Nakon toga se znak za ubacivanje spaja na prethodno formirani string. Rezultat se sprema u varijablu `wcrt`, korigira se novi početak (`poc`) i kreće se u novi krug petlje. Crvenom bojom prikazan je ispis stacka procedure `/s` nakon izvršavanja glavnog programa.

```

/s {mark pstack pop} def
/SP0JI
{/S2 exch def /S1 exch def
/L1 S1 length def /L2 S2 length def
/Lw L1 L2 add def
/wspoj Lw string def
wspoj 0 S1 putinterval
L2 0 ne
{wspoj L1 S2 putinterval }
if
wspoj
} bind def
%
/INSERT %ispis stringa sa znakom po vektoru
{/znak exch def /vektor exch def /rijec exch def
/VL vektor length def
/poz 0 def
/poc 0 def %index zadnje crtice
/wcrt () def /interval {poz poc sub} def
0 1 VL 1 sub
  {/i exch def
   vektor i get /poz exch def
   wcrt rijec poc interval getinterval SP0JI znak SP0JI
/wcrt exch def
  /poc poz def
  }bind for
wcrt rijec poc rijec length poc sub getinterval SP0JI
}bind def
%
%GLAVNI PROGRAM
(Danas ) (je ) SP0JI (lijepi ) SP0JI (dan.) SP0JI
(Laringitis) [2 5 7] (-) INSERT
(Rascjepkano) [1 2 3 4 5 6 7 8 9 10] (.) INSERT
(NASLOV) [0 6] (***) INSERT s

```

```

-mark-
(***)NASLOV(***)
(R.a.s.c.j.e.p.k.a.n.o)
(La-rin-gi-tis)
(Danas je lijepi dan.)

```

## ***CISCVEKTOR***

Sa procedurom CISCVEKTOR mogu se izbaciti svi jednaki elementi iz zadanog vektora. Sadržaj vektora može biti raznovrstan, od brojaka do stringova i miješano. Prije poziva procedure na stacku mora biti ulazni vektor koji se želi pročištititi, a iza izvršenja procedure na stacku ostane pročišćeni vektor.

**vektor** CISCVEKTOR

Za vektore čija je duljina veća ili jednaka od 3 primjenjen je algoritam koji koristi pomoćni vektor **pomocv**. Koriste se dvije petlje. Jedna se vrti od 0 do predzadnjeg člana vektora (i-petlja), a druga od i+1 do zadnjeg člana vektora (j-petlja). Znak koji se pretražuje definiran je i-petljom, a sa j-petljom dohvaćamo znakove od i-tog do kraja stringa. Usporedba i-tog i j-tog člana radi se sa naredbom **vektor i get vektor j get eq**. Kada se dogodi pogodak, varijabla **sud** puni se sa **true** i izlazi se iz j-petlje s naredbom **exit**. Pročišćeni vektor se slaže u vektoru **pomocv**. U njega se slažu i-ti članovi početnog vektora koji nemaju više svog duplikata do kraja tog vektora. To je samo onda kada je varijabla **sud false** odnosno kada se izašlo iz j-petlje bez pogotka. Iza i-petlje potrebno je još staviti zadnji znak u varijablu **pomocv**. Varijabla **vektor**, koja se na kraju stavlja na stack kao rezultat procedure, puni se sa onim sadržajem dijela vektora **pomocv** koji se tokom programa napunio. To se radi naredbom **pomocv 0 inter get interval /vektor exchange def**.

Vektori, koji imaju dva člana, pročišćavaju se jednom usporedbom. Vektori sa jednim članom, odnosno bez članova (nul vektor) prolaze kroz cijelu proceduru netaknuti i opet se na kraju pojavljuju na vrhu stacka.

Crvenom bojom prikazan je ispis stacka nakon izvršenja glavnog programa koji koristi proceduru CISCVEKTOR u različitim situacijama.

```

/s {mark pstack pop} def
/CISCVEKTOR %stack: vektor;izlaz:vektor
{/vektor exch def
/pomocv 50 array def
/VL vektor length def
/poz 0 def
VL 3 ge
    {0 1 VL 2 sub                                %brojac for petlje
    {/i exch def
    /sud false def
        i 1 add 1 VL 1 sub %brojac for petlje
        {/j exch def
        vektor i get vektor j get eq
            {/sud true def exit} if
        }bind for
    sud false eq
    {pomocv poz vektor i get put /poz poz 1 add def}
    if
    }bind for
    pomocv poz vektor VL 1 sub get put
    /inter poz 1 add def
    pomocv 0 inter getinterval /vektor exch def
    }
    {VL 2 eq
    {vektor 0 get vektor 1 get eq
    {vektor 0 1 getinterval /vektor exch def}
    if
    }if
    }
    ifelse
vektor
}bind def
%GLAVNI PROGRAM
[1 1 2 2 5 5 5 5 7 7 7 7 7 7 4 3 3 3 3 3] CISCVEKTOR
[(abc) (abc) 1 3 5 1 (cc)] CISCVEKTOR
[4 4] CISCVEKTOR
s

```

```

-mark-
[4]
[(abc) 3 5 1 (cc)]
[1 2 5 7 4 3]

```

## JEVOKAL

znak JEVOKAL

Ovdje demonstriramo proceduru JEVOKAL koja prepoznaje vokale u tekstu. Prije procedure JEVOKAL mora na stacku biti znak (string od jednog elementa) kojeg ona testira da li je iz skupa samoglasnika (vokala) i na stacku ostavlja rezultat u obliku suda (*true* ili *false*).

Nakon (a) JEVOKAL stanje na stacku je: *true*.

U proceduri JEVOKAL definirano je polje samoglasnika čiji se članovi dohvaćaju preko indeksa for petlje. Ako operator eq daje na stacku *true* (postignut pogodak) tada se varijabla sud puni sa *true* i sa exit naredbom izlazi iz for petlje.

U nastavku je prikazan primjer glavnog programa koji ispisuje samoglasnike, njihove pozicije unutar zadanog stringa i njihov ukupni broj. Koristeći proceduru JEVOKAL testira se svaki znak koji je dohvaćen forall operatorom. Crvenom bojom su označene naredbe potrebne za pretvaranje dekadskog ASCII koda u pripadni znak jer procedura JEVOKAL koristi znakove, a ne kodove.

Svaki put kada procedura JEVOKAL daje *true* na stacku if naredba pokreće proceduru u kojoj se ispisuje pozicija samoglasnika i nađeni samoglasnik sa procedurom za ispis stanja stacka /s. Varijabla "brojac" poveća se također za jedan.

Na kraju programa se iznos varijable "brojac" postavlja na stack i ispisuje sa procedurom /s. String (Tipografsko oblikovanje) spremljen u string varijabli filetxt ima 9 samoglasnika što se vidi na zadnjem ispisu stanja stacka.

Na sličan način se može odrediti procedura za bili koji set znakova. Potrebno je samo promijeniti polje znakova koje tražimo. Na primjer umjesto polja

```
/V [(a) (e) (i) (o) (u) (A) (E) (I) (O) (U)] def
stavimo
/V [(B) (D) (W) (k) (s)] def.
```

```

/s {mark pstack pop} def
/JEVOKAL      %stack:znak; izlaz na stack: sud
{/V [(a) (e) (i) (o) (u) (A) (E) (I) (O) (U)] def
/brojzn V length 1 sub def
/test exch def /sud false def
0 1 brojzn  {/k exch def
              test V k get eq
              {/sud true def exit}
              if
            }bind for
sud          %postavljanje rezultata na stack
}bind def
/filetxt (Tipografsko oblikovanje) def
%GLAVNI PROGRAM
/pozicija 0 def /brojac 0 def
/str 1 string def
filetxt
{/kod exch def /pozicija pozicija 1 add def str 0 kod put
str JEVOKAL
{pozicija str s pop pop /brojac brojac 1 add def}if}forall
brojac s
-mark-
(i)
2
-mark-
(o)
4
-mark-
(a)
7
-mark-
(o)
11          18
-mark-      -mark-
(o)          (a)
13          20
-mark-      -mark-
(i)          (e)
16          23
-mark-      -mark-
(o)          9

```



## *file* *readstring* *writestring* *closefile*

```
imedat (r) file
imedat (w) file
imedat string readstring
imedat string writestring
imedat closefile
```

Tekst može ući u PostScript program preko ulazne tekstualne datoteke (*file*) koja se prethodno napunila sa nekim tekst editorom, pisanjem rukom, programskim zapisivanjem ili generiranjem podataka.

Sa naredbom `imedat (r) file` definira se ulazna datoteka koja se želi čitati (opcija `r-read`). Znakovi se čitaju iz ulazne datoteke sa naredbom `imedat string readstring`. Znakovi koji se čitaju pune `string` koji prethodno mora biti dimenzioniran i postavljen na vrh stacka. Čitanje se odvija sve do oznake kraja datoteke (EOF-End Of File) ili dok se ne napuni `string`. Nakon izvršenja naredbe `readstring` na stacku se mogu pojaviti dvije situacije. Ukoliko je `string` bio dovoljno dimenzioniran da preuzme sve znakove iz datoteke, na vrhu stacka će biti cijeli tekst iz datoteke i riječ *true* koja to pokazuje. Ukoliko je `string` bio popunjen prije oznake EOF, na vrhu stacka će biti dio teksta iz datoteke koji je uspio stati u `string` i riječ *false* koja to pokazuje.

Naredba `imedat (w) file` definira izlaznu datoteku u koju će se zapisivati tekst (opcija `w-write`). Zapisivanje u izlaznu datoteku se radi naredbom `writestring` kojoj se prethodno na stacku mora staviti ime izlazne datoteke i niz znakova koji se žele zapisati u nju. Prilikom ispisa nekog stringa mogu se koristiti kontrolni znakovi za vrstu isključivanja kao `\n` `\r`, a koji su ujedno i komande za prelazak u novi redak (LF i CR).

Nakon upotrebe ulazne i izlazne datoteke, datoteke se moraju zatvoriti sa naredbom `closefile`. Na taj način se prekinula veza između programa i datoteka koja je cijelo vrijeme bila prisutna tokom programa.

U našem primjeru tekst **Pretraživanje teksta** nalazi se izvan programa u datoteci s imenom ULAZ, a željeni rezultat programa ispisuje se u datoteku IZLAZ.txt. Ovaj program je samo primjer. Zadatak je ispitati da li je četvrti znak iz ulaznog teksta konzonant iz skupa K2 i da li je sedmi znak iz skupa konzonanta K1 čiji su članovi definirani u programu. Procedure JEK1 i JEK2 rade po principu procedure JEVOKAL. Rezultat ispitivanja četvrtog i sedmog člana se ispisuje u obliku rečenica u izlaznu datoteku koju možemo otvoriti i pročitati sa nekim editorom teksta. Na kraju je ispisan sadržaj izlazne datoteke crvenom bojom.

Sadržaj ulazne datoteke ULAZ na disku:

### Pretraživanje teksta

```

/CETVRTI {3 1 getinterval} bind def
/SEDMI {6 1 getinterval} bind def
/JEK1    {/K1 [(s) (š) (z) (ž) (S) (Š) (Z) (Ž)] def
          /brojzn K1 length 1 sub def
          /test exch def /sud false def
          0 1 brojzn {/m exch def
                     test K1 m get eq
                     {/sud true def exit}
                     if
                   }bind for

          sud
        }bind def
/JEK2    {/K2 [(j) (l) (r) (v) (J) (L) (R) (V)] def
          /brojzn K2 length 1 sub def
          /test exch def /sud false def
          0 1 brojzn {/n exch def
                     test K2 n get eq
                     {/sud true def exit}
                     if
                   }bind for

          sud
        }bind def
%GLAVNI PROGRAM
/spremnik 1000 string def
(ULAZ) (r) file /ULAZ exch def
(IZLAZ.txt) (w) file /IZLAZ exch def
ULAZ spremnik readstring pop /filetxt exch def
filetxt CETVRTI JEK2
{IZLAZ (4. znak je iz skupa K2 konzonanta\n\r) writestring}
{IZLAZ (4. znak nije iz skupa K2 konzonanta\n\r) writestring}
ifelse
filetxt SEMI JEK1
{IZLAZ (7. znak je iz skupa K1 konzonanta) writestring}
{IZLAZ (7. znak nije iz skupa K1 konzonanta) writestring}
ifelse IZLAZ closefile ULAZ closefile

```

Sadržaj datoteke IZLAZ.txt na disku:

4. znak nije iz skupa K1 konzonanta  
7. znak je iz skupa K2 konzonanta

## IZBACICLAN

### DVOSTRUKI

vektor indeks IZBACICLAN  
string vektor DVOSTRUKI

Često je potrebno izbaciti neki član vektora po zadanom indeksu odnosno poziciji na kojoj se član nalazi unutar vektora. Prvi član vektora je na poziciji 0. Tu je pokazana procedura IZBACICLAN kojoj su argumenti na stacku ulazni vektor i indeks, a njen rezultat na stacku je reducirani vektor za izbačeni član po zadanom indeksu.

Algoritam procedure IZBACICLAN rješava četiri moguća slučaja: ulazni vektor ima jedan član, indeks izbacivanja pokazuje početnu poziciju, indeks izbacivanja pokazuje zadnju poziciju i indeks izbacivanja pokazuje na unutrašnju poziciju. Samo u slučaju izbacivanja unutrašnjeg člana vektora mora se upotrijebiti preslagivanje pomoću pomoćnog vektora PV sa naredbom `put interval`, u koji se slaže finalni vektor, i preko pomoćnih vektora V1 i V2 koji preuzimaju dijelove od početne pozicije do pozicije izbacivanja odnosno od pozicije izbacivanja do kraja vektora. Ostali slučajevi pune direktno izlaznu varijablu `vektor` koristeći naredbu `get interval`.

Procedura DVOSTRUKI koristi kao argumente na stacku jednu riječ (string) i zadani vektor pozicija znakova te riječi. Izlaz iz procedure je vektor reduciran za one članove čija vrijednost pokazuje na pozicije znakova unutar riječi koja odgovaraju drugim slovima glasova lj, nj i dž pisanih u kombinacijama: lj, nj, dž, LJ, NJ, DŽ, Lj, Nj i Dž. To ćemo kasnije koristiti u proceduri CRODIJELI. Ti se glasovi nalaze unutar procedure u polju `pol jeznakova`.

Sa i-petljom dohvaćaju se članovi vektora koji pokazuju pozicije znakova unutar riječi (varijabla `poz`), a sa k-petljom se dohvaćaju članovi polja `pol jeznakova`. Prvi cilj je dohvat dva susjedna znaka iz riječi na pozicijama `poz-1` i `poz` u varijablu `dznak`. Kada se u k-petlji dogodio pogodak sa naredbom `dznak pol jeznakova k get eq` izbacuje se i-ti član vektora umanjeno za korekciju `korek` koji je doveo do pogotka. Varijabla `korek` korigira indeks vektora `i` za broj izbacivanja koji su se dogodili do tog trenutka. To je potrebno jer se sa svakim izbacivanjem početni vektor smanjio za jedan član kao i njegova indeksacija od indeksa izbacivanja do kraja vektora.

Crvenom bojom je prikazan ispis stacka nakon izvršavanja glavnog programa koji koristi procedure IZBACICLAN i DVOSTRUKI. U primjeru sa procedurom DVOSTRUKI izbačeni su članovi vektora sa vrijednostima 3, 7, i 12. Te vrijednosti pokazuju na istaknute pozicije u riječi (`Zaljublivanje`).

```

/IZBACICLAN
{/indeks exch def /vektor exch def
/PV 50 array def /VL vektor length def
VL 1 eq
  {/vektor [] def}
  {indeks 0 eq
    {vektor 1 VL 1 sub getinterval /vektor exch def}
    {indeks VL 1 sub eq
      {vektor 0 VL 1 sub getinterval /vektor exch def}
      {vektor 0 indeks getinterval /V1 exch def
        vektor indeks 1 add VL indeks 1 add sub getinterval
        /V2 exch def
        PV 0 V1 putinterval PV V1 length V2 putinterval
        PV 0 VL 1 sub getinterval /vektor exch def
      }ifelse
    }ifelse
  }ifelse
vektor}bind def
%
/DVOSTRUKI %stack:rijec, vektor; iza na stacku:vektor
{/vektor1 exch def /rijec1 exch def
/VL vektor1 length def
/poljeznakova [(lj) (nj) (dž) (LJ) (NJ) (DŽ) (Lj) (Nj) (Dž)] def
/PL poljeznakova length def
/korek 0 def
0 1 VL 1 sub
  {/i exch def /poz vektor1 i korek sub get def
  /dznak rijec1 poz 1 sub 2 getinterval def
  0 1 PL 1 sub{/k exch def
    dznak poljeznakova k get eq
    {/vektor1 vektor1 i korek sub IZBACICLAN def
    /korek korek 1 add def exit}
    if
  }bind for
}bind for
vektor1}bind def
%GLAVNI PROGRAM
[(a) 4 (cc) 10 1] 0 IZBACICLAN
[2 (ww) 1 0.6 3] 3 IZBACICLAN
(Zaljubljanje) [2 3 6 7 9 11 12] DVOSTRUKI
-mark-
[2 6 9 11]
[2 (ww) 1 3]
[4 (cc) 10 1]

```

## *Dijeljenje riječi*

### ***CROVEKTOR*** ***CRODIJELI*** ***PRVIZADNJI***

Algoritam za dijeljenje riječi po pravilima hrvatskog pravopisa objavljen je u knjizi "Stolno izdavaštvo-DeskTop Publishing", V. Žiljak, 1989. na strani 132 i 133. Tamo je prikazan dijagram toka kao i pripadni Basic program osnovnog dijeljenja. Ovdje je taj algoritam usavršen i isprogramiran kao PostScript procedura koju kasnije mogu pozivati programi za automatsko dijeljenje riječi na kraju retka.

```
string CROVEKTOR
string CRODIJELI
string vektor PRVIZADNJI
```

Procedura CRODIJELI daje na stacku riječ koja u sebi ima znak crtice na mjestima dozvoljenog dijeljenja riječi po hrvatskom pravopisu. Argument procedure je riječ postavljena na vrh stacka. Takva riječ će se kasnije koristiti za automatsko dijeljenje riječi na kraju retka u proceduri LPRELOM.

Procedura se izvršava tako da se prvo poziva procedura CROVEKTOR koja na bazi ulazne riječi daje **vektor** s pozicijama dozvoljenog dijeljenja osnovnog algoritma. Sa procedurom CISCVEKTOR vrši se pročišćavanje duplih pozicija za dijeljenje što ne riješava osnovni algoritam. Početna ulazna riječ i pročišćeni vektor su argumenti procedure DVOSTRUKI koja riješava izbacivanje pozicija dozvoljenog dijeljenja ukoliko su se pojavile između lj, nj i dž što su iznimke osnovnog algoritma. Sa procedurom PRVIZADNJI izbacuju se iz vektora članovi koji pokazuju na moguće dijeljenje iza prvog znaka u riječi i ispred zadnjeg znaka u riječi. Nakon toga se može pristupiti insertiranju znaka crtice na pozicije unutar riječi po finalnom vektoru sa procedurom INSERT.

Procedura CROVEKTOR analizira znakovne trojke unutar riječi od početka do kraja riječi. Svaka trojka se dohvaća i-petljom, a sprema se u varijablu **t r o j k a**. Svaki znak u trojci testira se sa već prikazanim procedurama JEVOKAL, JEK1, JEK2. Ovisno o testovima po algoritmu, izvršavaju se procedure RASTAV1 i RASTAV2 ili se rastavljanje preskače. Procedura RASTAV1 stavlja u **vektor** broj pozicije unutar riječi po pravilu  $Z_1-Z_2Z_3$ , a procedura RASTAV2 po pravilu  $Z_1Z_2-Z_3$  gdje su  $Z_1$ ,  $Z_2$  i  $Z_3$  znakovi tekuće trojke koja se analizira.

Crvenom bojom je ispisan stack kao rezultat glavnog programa. Naredba **pop** iza poziva procedure **/s** skida ispisanu riječ sa stacka da se ne bi ponovno pojavila u slijedećem ispisu stanja stacka.

```

/CROVEKTOR stack: rijec ; izlaz na stacku: vektor
{/wfin exch def
/RASTAV1 {vektor j i 1 add put /j j 1 add def} bind def
/RASTAV2 {vektor j i 2 add put /j j 1 add def} bind def
/PRVI {trojka 0 1 getinterval} bind def
/DRUGI {trojka 1 1 getinterval} bind def
/TRECI {trojka 2 1 getinterval} bind def
/vektor 50 array def /LR wfin length def
/j 0 def %inicijalizacija indeksa polja
LR 3 ge
{ 0 1 LR 3 sub %brojac for petlje
{/i exch def wfin i 3 getinterval /trojka exch def
DRUGI JEVOKAL
{TRECI JEVOKAL {RASTAV2} if}
{TRECI JEVOKAL
{PRVI JEVOKAL {RASTAV1} if}
{DRUGI JEK1
{RASTAV1}
{TRECI JEK2
{DRUGI JEK2
{DRUGI (l) eq TRECI (j) eq and
{RASTAV1}
{RASTAV2} ifelse}
{RASTAV1}
ifelse}
{RASTAV2}
ifelse}
ifelse}
ifelse}
ifelse}
}for
}
if
vektor 0 j getinterval /vektor exch def
vektor
}bind def

```

```

/PRVIZADNJI %stack:riječ vektor; iza na stacku:vektor
{/vektor2 exch def /rijec exch def
/VL2 vektor2 length def
VL2 0 ne
{/DR rijec length def
/prvi vektor2 0 get def /zadnji vektor2 VL2 1 sub get def
prvi 1 eq
  {/vektor3 vektor2 0 IZBACICLAN def
  zadnji DR 2 sub eq
    {/vektor3 vektor3 vektor3 length 1 sub IZBACICLAN def}
    if}
  {zadnji DR 2 sub eq
    {/vektor3 vektor2 VL2 1 sub IZBACICLAN def}
    {/vektor3 vektor2 def}
    ifelse}
ifelse
vektor3
} {vektor2} ifelse
}bind def
%
/CRODIJELI
{/rijec exch def
rijec CROVEKTOR CISCVEKTOR
rijec exch DVOSTRUKI rijec exch PRVIZADNJI
rijec exch (-) INSERT}bind def
%
/s {mark pstack pop} def
%GLAVNI PROGRAM
(Pojednostavljenje) CRODIJELI s pop
(naglašavanje) CRODIJELI s pop
(prostorno) CRODIJELI s pop (uređivanje) CRODIJELI s pop
(troškovnik) CRODIJELI s pop (korektura) CRODIJELI s
-mark-
(Po-jed-no-stav-lje-nje)
-mark-
(na-gla-ša-va-nje)
-mark-
(pro-stor-no)
-mark-
(ure-đi-va-nje)
-mark-
(tro-škov-nik)
-mark-
(ko-rek-tu-ra)

```

## *Prijelom teksta*

Na sljedeće dvije stranice prikazan je program koji prelama tekst u fontu FSHelvetica sa veličinom od 16 točaka na zadanu širinu stupca i zadanim vertikalnim pomakom (proredom). Prije poziva procedure LPRELOM mora se u glavnom programu definirati početak prve pismovne linije sa `move to` naredbom i postaviti tekst (string) na vrh stacka.

Na početku programa nalaze se svi parametri prijeloma: zadani font, širina stupca **M**, vertikalni pomak **L**, širina crtice **dc** za prijelom na kraju retka u zadanom fontu, širina razmaka između riječi **ds** u zadanom fontu i širina stupca (retka) umanjena za širinu crtice **MC**. Varijabla **dM** sadrži tekuću poziciju u pismovnoj liniji. Varijabla **dw** tokom programa sadrži cijelu riječ ili slog riječi.

Procedura LPRELOM preuzima poziciju početka prve pismovne linije i šalje riječ po riječ u proceduru PRELOM. U proceduri LPRELOM rješavaju se dva moguća slučaja:

1. Nađen razmak između riječi (nije zadnja riječ) -> šalji riječ u proceduru PRELOM
2. Nije nađen razmak između riječi (zadnja riječ)

-> šalji riječ u proceduru PRELOM i završi proceduru sa `exit` (gotov prijelom)

Procedura PRELOM rješava ove situacije:

1. Riječ stane u redak -> ispiši riječ i razmak, korigiraj  $dM = dM + dw + ds$
2. Riječ ne stane u redak -> šalji riječ u CRODIJELI za ubacivanje (-) i odi u PRELOMRIJECI

Ulaz u proceduru PRELOMRIJECI je riječ koja nije stala u tekući redak. Ta riječ ima u sebi ugrađene crtice od procedure CRODIJELI. Procedura CRODIJELI rješava ove moguće situacije:

1. Riječ nema crtice -> prelazi u novi redak, ispiši riječ i razmak, korigiraj  $dM = dw + ds$
2. Riječ ima crticu

2.1 Nije zadnji slog i stane u  $MC = \text{redak} - \text{širina crtice}$  -> ispiši slog i korigiraj  $dM = dM + dw$

2.2 Prvi slog i ne stane u  $MC$  -> prelazi u novi redak, ispiši slog i korigiraj  $dM = dw$

2.3 Nije zadnji slog niti prvi slog i ne stane u  $MC$

-> ispiši (-), prelazi u novi redak, ispiši slog, korigiraj  $dM = dw$

2.4 Zadnji slog i ne stane u puni redak

-> ispiši (-), prelazi u novi redak, ispiši slog, razmak između riječi, korigiraj  $dM = dw + ds$

Na stranicama iza programa prikazani su ispisi za dva različita prijeloma. Parametri prijeloma kao i pripadni glavni programi ispisani su plavom bojom unutar mreže. Ulazni tekst se nalazi pri kraju programa kao string varijabla `fi le t x t`. U string varijabli je dozvoljena upotreba specijalnog znaka (`\`) koji omogućuje pisanje stringa u više redova. Inače bi kôd za prijelaz u novi redak bio dio stringa.



```

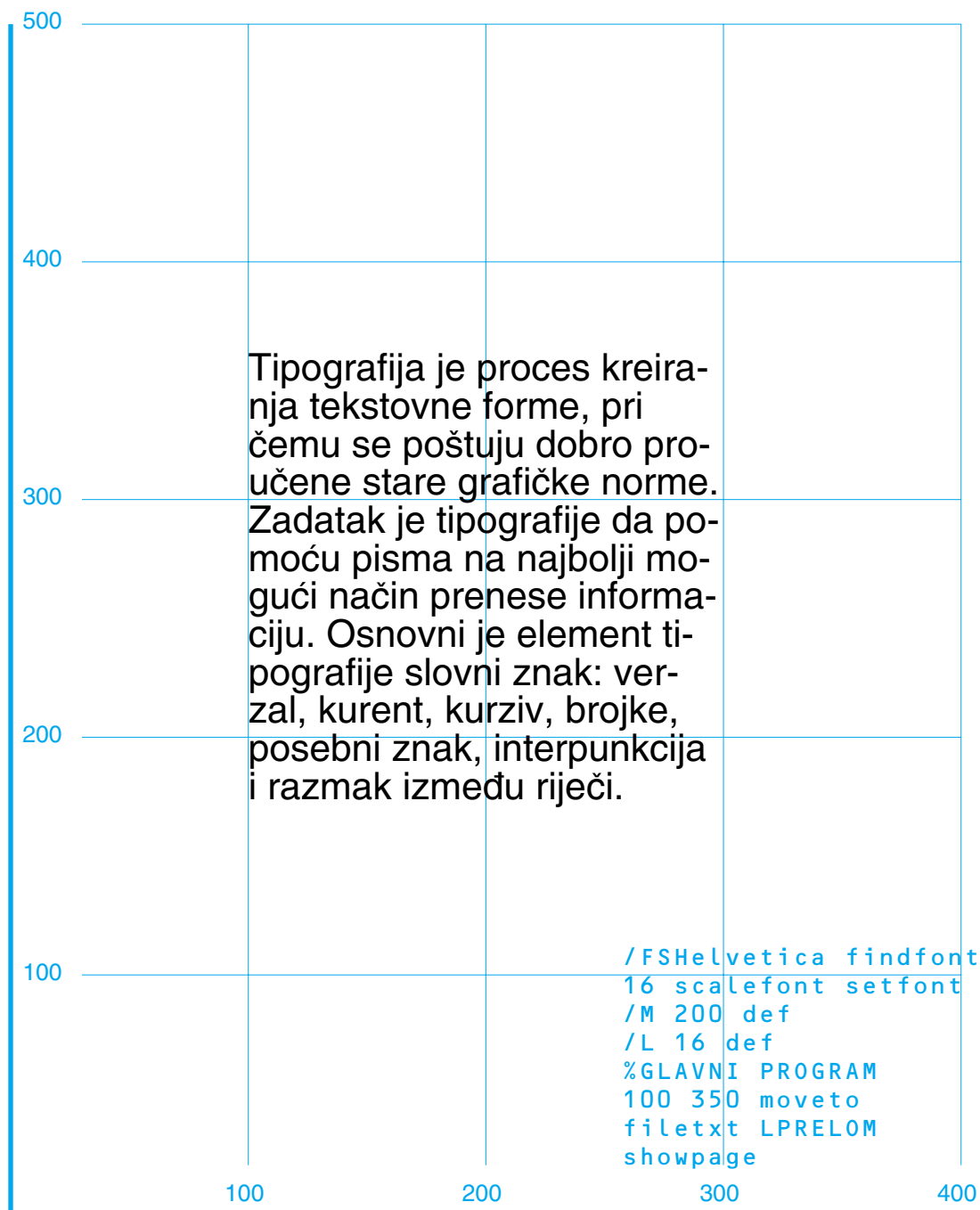
/FSHelvetica findfont 16 scalefont setfont
/dM 0 def      %tekuca pozicija u liniji
/M 200 def     %sirine stupca (linije, teksta)
/L 16 def      %prored
/ds ( ) stringwidth pop def
/dc (-) stringwidth pop def
/MC M dc sub def
%
/PRELOMRIJECI
{/restw exch def
restw (-) search
  {pop pop pop /PRVISLOG 1 def
   {restw (-) search
    { /w exch def pop
     /restw exch def
     /dw w stringwidth pop def
     dM dw add MC le
     {w show /dM dM dw add def /PRVISLOG 0 def}
     {PRVISLOG 1 eq
      {/PRVISLOG 0 def /Y Y L sub def X Y moveto
       w show /dM dw def}
      { (-) show
       /Y Y L sub def X Y moveto
       w show /dM dw def}
      ifelse}
     ifelse}
    {/w exch def
     /dw w stringwidth pop def
     dM dw add M le
     {w show ( ) show
      /dM dM dw add ds add def}
     {(-) show
      /Y Y L sub def X Y moveto
      w show ( ) show
      /dM dw ds add def}
     ifelse exit}
   ifelse
  }loop}
{pop /Y Y L sub def X Y moveto
 w show ( ) show /dM dw ds add def}
ifelse
}bind def

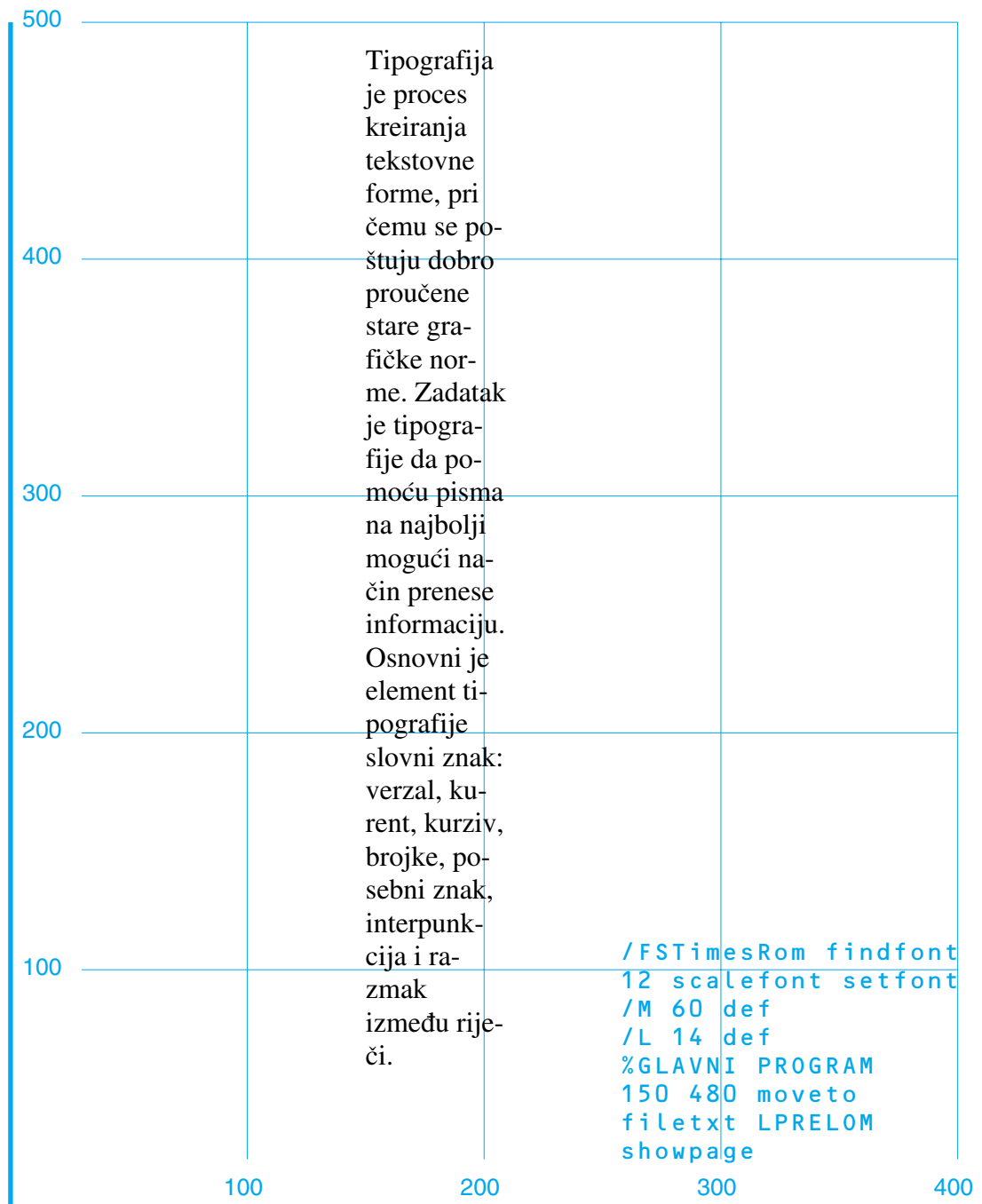
```

```

/PRELOM
{/w exch def
/dw w stringwidth pop def
dM dw add M le
  {w show ( ) show /dM dM dw add ds add def}
  {w CRODIJELI PRELOMRIJECI }
  ifelse
}bind def
%
/LPRELOM
{currentpoint /Y exch def /X exch def
/txt exch def
/resttxt txt def
  {resttxt ( ) search
    {/w exch def pop /resttxt exch def w PRELOM }
    {/w exch def w PRELOM exit}
    ifelse
  }loop
}bind def
%
/filetxt
(Tipografija je proces kreiranja tekstovne forme, pri čemu \
se poštuju dobro proučene stare grafičke norme. Zadatak je \
tipografije da pomoću pisma na najbolji mogući način prenese \
informaciju. Osnovni je element tipografije slovni znak: \
verzal, kurent, kurziv, brojke, posebni znak, interpunkcija \
i razmak između riječi.) def
%
%
%GLAVNI PROGRAM
100 350 moveto
filetxt LPRELOM
showpage

```





# PostScript

III poglavlje  
programiranje piksel  
grafike

## Piksel grafika

U ovom poglavlju prikazujemo sliku sastavljenu od sivih kvadratića nazvanih piksel (picture element). Za svaki pojedinačni piksel karakteristično je da na cijeloj svojoj površini ima jednoličnu sivoću. Vrijednost sivoće piksla data je jednim brojem u jednom bajtu a to znači da je raspon od bijelog do crnog razdjeljen nekontinuirano na 265 stepenica sivog tona. U digitalnoj reprofotografiji korist se termin "siva skala" kako bi se razlikovala od kontinuiranog tonskog prelaza sivoće kakvu imamo u tradicionalnoj fotografskoj tehnici. Ljudsko oko razlikuje oko 50 nijansi sive skale pa razdiobu od 256 doživljavamo kao kontinuirani ton.

Slika se određuje nizom brojaka koje pravilno grade ukupnu površinu slike od gornjeg lijevog ugla, horizontalno desno tvoreći retke od vrha slike do dna, završavajući s donjim desnim uglom kao zadnjim pikslom. Prikaz slike na ekranu ili pisaču traži podatke o broju piksla u retku, broju stupaca, veličini otisnutog piksla i položaju slike na stranici.

Interpretacija slike tiskom, metodom piksla i raster, uzima u obzir nekoliko faktora: reprodukcija detalja na slici, zauzetost memorije računala, brzina procesora računala, ograničenja tiskarskog procesa digitalnog i analognog. Slika, česće nazivana original, obično se unosi u računalo skaniranjem. Pri tome se određuje veličina piksla to jest, elementarna kvadratična površina od kojih će se sastojati digitalni zapis slike. Na površini svakog elementarnog djelića slike integralno se čitata zacrnjenje i taj broj pridružiti pikslu. O detaljima ili strukturi originala unutar površine piksla, nakon skaniranja, neće postojati nikakva informacija. Razlikovanje tih detalja moguće je jedino ponovnim skaniranjem, smanjivanjem elementarne površine čitanja, a to znači da se ista slika interpretira s mnogo više podataka.



```

/GRGA <
2B372A52D6D04A005C4F162325222425211511151E25221F201F1D
28362A8CE3D17C100E3316181C1519180C21312A111A20191B1D1A
223441C3C3B078410029180E0D201A1442757A774F222C1A1B1E18
11162B4818110D0E110F0E09034B745B5F4E3B4A78722A211C1916
0E240F020A0E0C0B130F1A2310284C342E2E1E182D6252241A1919
155E29050C0E0C0C0F101027271E151114110B151013171B191718
0C0F0B0B0D0C0A0E1B23180A0B0F0B0A0D0F121C1A1C1319191514
0E0B0A0C0B0A070C131B12100E10110D0F161A1E2729221E1D2413
0E0C0A0C0A0D19080D11121F162027171010121D232027272B3827
0D0C0B0C062F5D0A0C11181E2524200E100F131D241B1B2633303A
0D0D0C0D0A1530330B161E4556493730221632343B2E212C383D41
0D0D0C0D0E0612221A2B37646E686D816458695E634C363A48473C
090F0F0D0D0B0903394F5671707A93ADB4B5AC98997C646F6C595E
386320080E0B0A083A515A6B70859FBED0DADAD2CBB7B2B0996B6C
304F1E0A0E0C0B07264564556186ACC8DCE7E8E5DAD6D2C1A37760
064554130B0D06061A375A4B5B8FB2C8D8E2E8EADDD6CEBD9E693B
0C17230E0B0E23061230545D648FB1C4CFDAE3E4D9D2CCBA9B5C30
0E0D0B0A0A1F5640202B63736B8EAAA9ACC4D1D3CAC6BAAA9C694D
0E0F0D0A0C53528D7A31698E9796705A7EA7C2CAB7957D7A83746A
0E0E0D0C0B0C55A0813E71ABB088560072A4BDCE99520B44628477
0501000507011286BF5C61A6B5C2BFAAB5AFB1B8919D7197A8B686
39656833144A083B935A52A3BBC7CBCDC8A6A2BC97AEC6C7CABB85
E5F3F4E2B6931A05586B4A9AB6CEDCE0D79DB6C4A5B4DACFC8AC81
EBEBEBEDF06E00061C433C89B0CEDDD8CCBDBDB2ABBFEDBBE9566
ECE2D2C0DCA50C0007032079A1B6B193B2C9D8DBC5AFBDC9A86B57
E5C08280B0BD854005030B499DAC9B694B64666A596492A27E483E
E7DCB97FB2D0EBEDBD49000B51A3B2A2404E76483483A6926B2A07
DFC3A3B8CBD9E2EEFAED7905073282BD9C7B8483A3B4AA7E32080E
A09F91928BC3D3E9F3F0F9890804104BBBCCC4C2B3824C1C080B0F
91516287A6CDECFDEE8F0F57B060600318BB3AB702606080C0D0C
5E62777F96DDFAEEE8E6E9F1EF750A08050A1E20315E0E0A0F0B23
436C5F527DBAE5F4F9FFFCEEDFE683040A0906040F5711090B0FA7
544F49585B4067838BA8C4EAF2E0B51307090807031F100B0818D0
5B6097A37332395155514C5988D7D72B040B0B0B0A0B0D0E080EAF
3651999DC3781C2D415D523D37458132010A0B0D0B0A0B0A0A0044
56606D80471807030F7A886E565B534E1E050A0D0B0A090B0A0143
2A2E3324040509080221836657545B726C21070C0B0B0A0A0B0051
02010003080A0909090034C04286987260781C090C0C0B0A0B0523
080708090809090908056C9BBB85897C46210B0B0B090B0A0518
0809080708080808090A090625728E7388605F1E090A090B0B1B55
> def

```

%600/40=405/27=15x15 točka po pikslu

```

405 600 scale
27 40 8 [27 0 0 40 neg 0 40] {GRGA}
image showpage

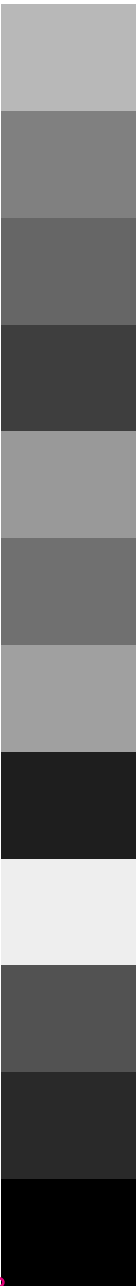
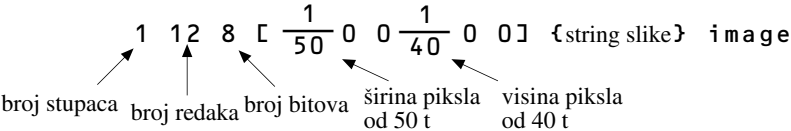
```



```
1 12 8
[ 0.02 0 0 0.025 0 0 ]
{<002952EE1EA070993E6680B8>} image
```

} minimalni PostScript program  
prikaza piksel slike

Slika u piksel grafici definira se nizom podataka koji određuju zacrnjenje pojedinog piksla u nizu. Naš primjer je dat preko heksadecimalnih veličina. Prvi piksel ima svjetlinu 0, drugi piksel svjetlinu 29<sub>H</sub> (heksadecimalno) od maksimalne svjetline FF<sub>H</sub>. Treći piksel ima vrijednost 52<sub>H</sub> do zadnjeg s vrijednošću B8<sub>H</sub>. Svaki piksel zapisan je preko 8 bitova, a prikazat će se u jednom stupcu svih dvanaest vrijednosti slike. Veličina ispisa pojedinog piksla je 50 horizontalnih točaka i 40 vertikalnih točaka. Prva dva parametra programa određuju slaganje niza piksla u stupce i redove. Treći parametar određuje broj bitova za sivu skalu: 2<sup>8</sup> = 256 sivih razina.



8-bitna svjetlina		razina sivog	
hex	dec	255-dec	postotak
00	0	255	100%
29	41	214	84%
52	82	173	68%
EE	238	17	7%
1E	30	225	88%
A0	160	95	37%
70	112	143	56%
99	153	102	40%
3E	62	193	76%
66	102	153	60%
80	128	127	50%
B8	184	71	28%

Tablica konverzije zadanog heksadecimalnog zapisa svjetline u dekadski zapis svjetline, pripadnu diskretnu dekadsku razinu sivog (0 -255) i postotak zacrnjenja površine piksla

Dvodimenzionalni raspored piksla ima dva načina slaganja. Prvi način odgovara točnom (potpunom) iskorištenju podataka u stringu slike (1, 2, 3 i 4). Drugi način odnosi se na nejednak broj piksla u slici s brojem podataka u stringu slike. Smjerovi slaganja prikazani su strelicama. Slike (5, 6 i 7) prikazuju kako se slažu piksli kada je piksel matrica  $S \times R$  manja (5) odnosno veća (6 i 7) od ponuđenog niza piksela. Kada je manja, popunjavanje prestaje na podatku koji zadnji popunjava zadanu matricu, a ako je veća onda **image** naredba počinje nanovo uzimati podatke od početka stringa dok se ne popuni zadana matrica piksela. U oba načina slaganja piksla popuniti će se dvodimenzionalni oblik slike.

12 1 8

[ 1 20 div 0 0 1 40 div 0 0 ]

{<002952EE1EA070993E6680B8>}

image

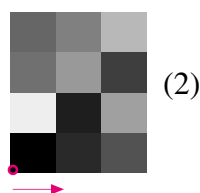


3 4 8

[ 1 15 div 0 0 1 15 div 0 0 ]

{<002952EE1EA070993E6680B8>}

image

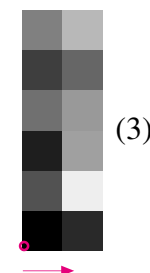


2 6 8

[ 1 15 div 0 0 1 15 div 0 0 ]

{<002952EE1EA070993E6680B8>}

image



6 2 8

[ 1 15 div 0 0 1 15 div 0 0 ]

{<002952EE1EA070993E6680B8>}

image

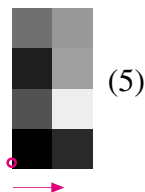


2 4 8

[ 1 15 div 0 0 1 15 div 0 0 ]

{<002952EE1EA070993E6680B8>}

image

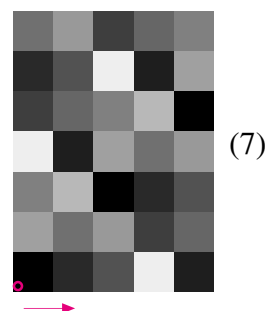


5 7 8

[ 1 15 div 0 0 1 15 div 0 0 ]

{<002952EE1EA070993E6680B8>}

image

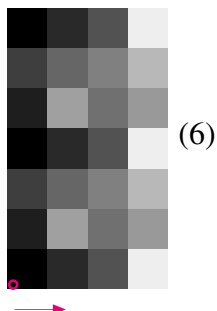


4 7 8

[ 1 15 div 0 0 1 15 div 0 0 ]

{<002952EE1EA070993E6680B8>}

image



Naredba `image` koristi 5 parametara koji moraju biti postavljeni na stacku:

- broj piksela u retku čime je određen broj stupaca S;
- broj piksela u stupcu (broj redaka) R;
- broj bitova po pikslu G čime je određen broj stepenica sive skale ( $2^G$ );
- transformacijska matrica slike;
- string vrijednosti sivih razina slike.

```
S R G
[A B C D tx ty] {string slike}
image
```

Transformacijska matrica data je relacijama:

$$x' = Ax + Cy + t_x$$

$$y' = Bx + Dy + t_y$$

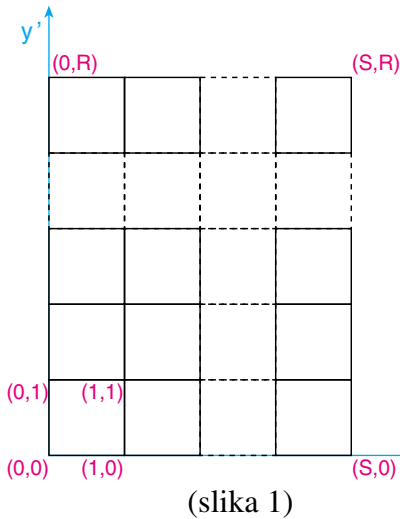
### Određivanje veličine slike naredbom `scale`

Transformacijska matrica  $\begin{bmatrix} 1 & 0 & 0 & 1 & t_x & t_y \end{bmatrix}$  daje dimenziju piksla kao kvadrat od 1 točke, a time je širina slike od S piksla jednaka S točaka, a visina slike od R piksla je R točaka. Za transformacijsku matricu  $\begin{bmatrix} 2 & 0 & 0 & 5 & t_x & t_y \end{bmatrix}$  piksl je pravokutnik širine  $1/2=0.5$  točaka i visine  $1/5=0.2$  točaka, a širina cijele slike  $0.5S$  točaka, a visina  $0.2R$  točaka. Ako je transformacijska matrica slike takva da je  $A=S$ , a  $D=R$   $\begin{bmatrix} S & 0 & 0 & R & t_x & t_y \end{bmatrix}$  tada je piksl širine  $1/S$  točaka i visine  $1/R$  točaka, a dimenzija cijele slike je kvadrat od 1 točke.

Vidi se da je dimenzija slike s matricom  $\begin{bmatrix} A & 0 & 0 & D & t_x & t_y \end{bmatrix}$  zadana indirektno i to tako da kvocijent  $S/A$  definira širinu slike, a kvocijent  $R/D$  visinu slike. Direktno definiranje širine i visine slike može se postići upotrebom naredbe `scale` prije `image` naredbe i prikladnom transformacijskom matricom:

```
W H scale
S R G [S 0 0 R tx ty] {string slike}
image
```

Tako podešena transformacijska matrica formira dimenziju slike kao jedinični kvadrat koji se skalira (povećava ili smanjuje) na dimenziju W i H točaka.



### Definicija piksla preko inverzne transformacije

Uvriježeno je razmišljanje da je piksel kvadratičnog oblika jer mnogi programi imaju samo takovu mogućnost manipulacije s njim. Pikseli su definirani kao paralelogrami pa svaki pojedini piksel može poprimiti zakrenute deformirane oblike.

Koordinatni prostor slike je zaseban koordinatni prostor gdje su pikseli predstavljeni kao kvadrati sa stranicama od 1 točke (slika 1), pa su kutne koordinate prvog piksla (0,0), (1,0), (1,1) i (0,1), a kutne koordinate cijele slike u tom prostoru su (0,0), (S,0), (S,R) i (0,R).

Stvarne koordinate svakog piksla dobivaju se inverznom transformacijom zadanom sa transformacijskom matricom  $[A \ B \ C \ D \ t_x \ t_y]$  na način da je ciljni koordinatni prostor jedinični koordinatni prostor slike. Transformacijska matrica definira relacije s time da su koordinate  $(x', y')$  iz jediničnog koordinatnog prostora slike:

$$\begin{aligned} x' &= Ax + Cy + t_x \\ y' &= Bx + Dy + t_y \end{aligned} \quad \text{ili} \quad \begin{bmatrix} x' - t_x \\ y' - t_y \end{bmatrix} = \begin{bmatrix} A & C \\ B & D \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Koordinate piksela koji će se prikazati (u tisku ili na ekranu) dobivaju se rješavanjem gornjeg sustava jednačbi:

$$\begin{aligned} x &= \frac{\det_1}{\det} & \det_1 &= \begin{vmatrix} x' - t_x & C \\ y' - t_y & D \end{vmatrix} & \det &= \begin{vmatrix} A & C \\ B & D \end{vmatrix} \\ y &= \frac{\det_2}{\det} & \det_2 &= \begin{vmatrix} A & x' - t_x \\ B & y' - t_y \end{vmatrix} \end{aligned}$$

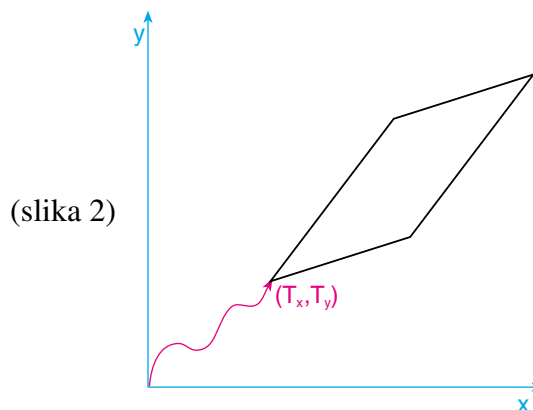
$$x = \frac{(x' - t_x)D - (y' - t_y)C}{AD - BC} ; \quad y = \frac{(y' - t_y)A - (x' - t_x)B}{AD - BC}$$

Koordinate prvog piksela izračunavaju se iz četiri sustava jednažbi s dvije nepoznanice:

$$\begin{array}{ll} 0 = Ax + Cy + t_x & 1 = Ax + Cy + t_x \\ 0 = Bx + Dy + t_y & 0 = Bx + Dy + t_y \\ 1 = Ax + Cy + t_x & 0 = Ax + Cy + t_x \\ 1 = Bx + Dy + t_y & 1 = Bx + Dy + t_y \end{array}$$

Točka  $(x, y)$  koja se dobiva kada je  $(x', y') = (0, 0)$  označena je sa  $(T_x, T_y)$  jer ona predstavlja translaciju slike (slika 2):

$$T_x = \frac{Ct_y - Dt_x}{AD - BC} \quad ; \quad T_y = \frac{Bt_x - At_y}{AD - BC}$$

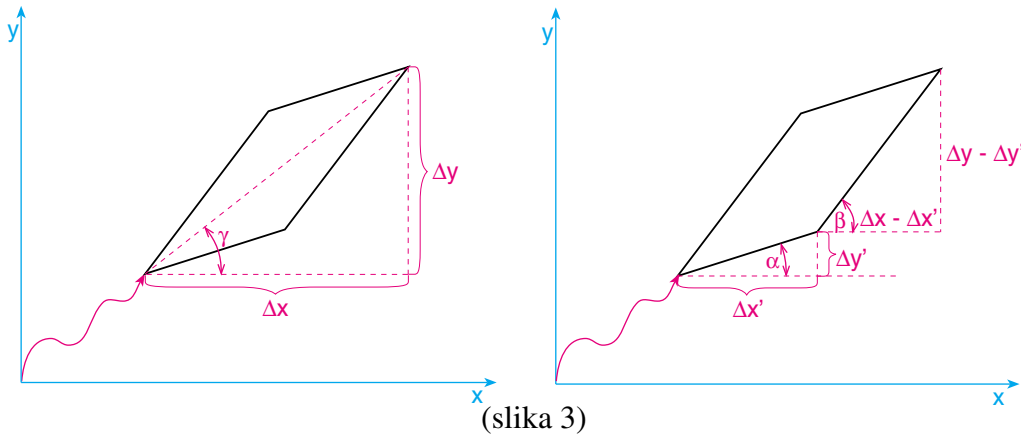


Iz izračunavanja preostalih  $(x, y)$  točaka za jedinični piksel izvedene su relacije preko kojih se vide nagibi i izdužavanja jednog piksela (slika 3):

$$\begin{array}{ll} \Delta x = \frac{D - C}{AD - BC} & \text{tg} \gamma = \frac{\Delta y}{\Delta x} \quad \text{tg} \gamma = \frac{A - B}{D - C} \\ \Delta y = \frac{A - B}{AD - BC} & \gamma = \arctg \left( \frac{A - B}{D - C} \right) \\ \Delta x' = \frac{D}{AD - BC} & \text{tg} \alpha = \frac{\Delta y'}{\Delta x'} \quad \text{tg} \alpha = \frac{-B}{D} \\ \Delta y' = \frac{-B}{AD - BC} & \alpha = \arctg \left( \frac{-B}{D} \right) \end{array}$$

$$\Delta x - \Delta x' = \frac{-C}{AD - BC} \quad \operatorname{tg} \beta = \frac{\Delta y - \Delta y'}{\Delta x - \Delta x'} \quad \operatorname{tg} \beta = \frac{-A}{C} \quad \beta = \arctg\left(\frac{-A}{C}\right)$$

$$\Delta y - \Delta y' = \frac{A}{AD - BC}$$



Bez ponovnog rješavanja sustava jednačbi za svaku koordinatu piksela čitave matrice slike  $S \times R$ , može se pomoću prethodnih relacija za prvi piksel izračunati koordinatna točka bilo kojeg piksela slike samo na temelju poznavanja pozicije piksela preko broja retka  $i$  i stupca  $j$  slike ( $(x_I, y_I)$ ,  $(x_{II}, y_{II})$ ,  $(x_{III}, y_{III})$ ,  $(x_{IV}, y_{IV})$ ) (slika 4):

$$x_I = (i - 1)(\Delta x - \Delta x') + (j - 1)\Delta x' \quad x_I = \frac{(j - 1)D - (i - 1)C}{AD - BC} + T_x$$

$$y_I = (i - 1)(\Delta y - \Delta y') + (j - 1)\Delta y' \quad y_I = \frac{(i - 1)A - (j - 1)B}{AD - BC} + T_y$$

$$x_{II} = x_I + \Delta x' \quad x_{II} = \frac{jD - (i - 1)C}{AD - BC} + T_x$$

$$y_{II} = y_I + \Delta y' \quad y_{II} = \frac{(i - 1)A - jB}{AD - BC} + T_y$$

$$x_{III} = x_I + \Delta x \quad x_{III} = \frac{jD - iC}{AD - BC} + T_x$$

$$y_{III} = y_I + \Delta y \quad y_{III} = \frac{iA - jB}{AD - BC} + T_y$$



$$x_{SR} = \frac{SD - RC}{AD - BC} + T_x$$

$$y_{SR} = \frac{RA - SB}{AD - BC} + T_y$$

$$x_{1R} = \frac{-RC}{AD - BC} + T_x$$

$$y_{1R} = \frac{RA}{AD - BC} + T_y$$

$$\Delta x_{SLIKA} = x_{SR} - x_{11}$$

$$\Delta x_{SLIKA} = \frac{SD - RC}{AD - BC}$$

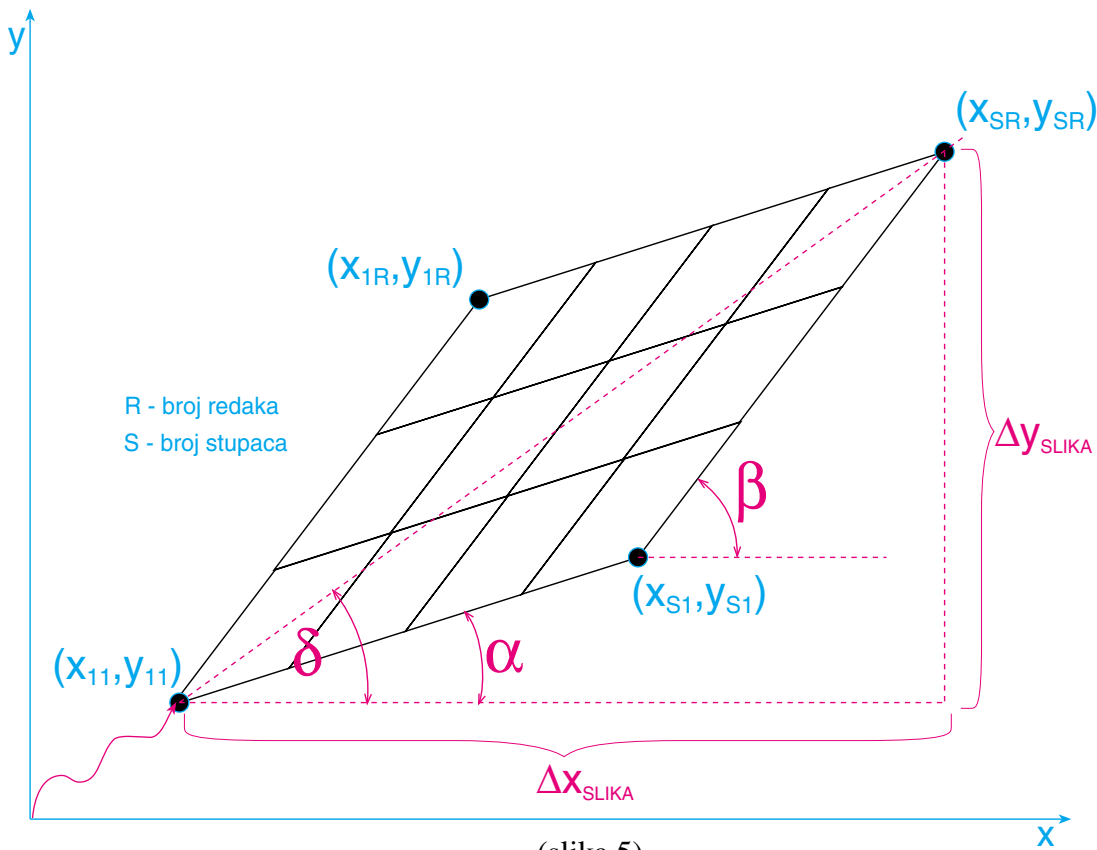
$$\Delta y_{SLIKA} = \frac{RA - SB}{AD - BC}$$

$$\Delta y_{SLIKA} = y_{SR} - y_{11}$$

$$\operatorname{tg} \delta = \frac{\Delta y_{SLIKA}}{\Delta x_{SLIKA}}$$

$$\operatorname{tg} \delta = \frac{RA - SB}{SD - RC}$$

$$\delta = \arctg \left( \frac{RA - SB}{SD - RC} \right)$$





Sve relacije su date za početni koordinatni sustav definiran s `1 1 scale`. Ako se ispred naredbe `image` definira skaliranje `W H scale` tada će se x relacije množiti sa skalarom W, a sve y relacije sa skalarom H. Koordinate slike i kutevi deformacije slike u takvom skaliranom koordinatnom prostoru imaju relacije:

$$\begin{aligned} x_{11} &= T_x W & x_{S1} &= \left( \frac{SD}{AD - BC} + T_x \right) W \\ y_{11} &= T_y H & y_{S1} &= \left( \frac{SB}{AD - BC} + T_y \right) H \end{aligned}$$

$$\begin{aligned} x_{SR} &= \left( \frac{SD - RC}{AD - BC} + T_x \right) W & x_{IR} &= \left( \frac{-RC}{AD - BC} + T_x \right) W \\ y_{SR} &= \left( \frac{RA - SB}{AD - BC} + T_y \right) H & y_{IR} &= \left( \frac{RA}{AD - BC} + T_y \right) H \end{aligned}$$

$$\alpha = \arctg\left(\frac{-B}{D} \frac{H}{W}\right) \quad \beta = \arctg\left(\frac{-A}{C} \frac{H}{W}\right)$$

$$\gamma = \arctg\left(\frac{A - B}{D - C} \frac{H}{W}\right) \quad \delta = \arctg\left(\frac{RA - SB}{SD - RC} \frac{H}{W}\right)$$

U primjerima koji slijede prikazane su razne kombinacije parametara rasporeda piksla po slici S i R, matrice transformacije i parametara W i H u naredbi `scale`. U prve četiri slike postignut je isti oblik slike premda su korištene različite kombinacije parametara W, H, A i D. U četvrtom primjeru je prvi piksel podignut od ishodišta slike (`20 300 translate`) za cijelu visinu slike, a napredovanje slaganja piksla je u tekućem retku desno a potom vertikalno prema dolje (negativni smjer). U primjerima od šestog do desetog demonstrirano je djelovanje parametara B, C,  $t_x$  i  $t_y$ , s time da imaju zajedničko ishodište u (0, 0).

```

/slika <002952401EA070993E6680B8> def
/font1 {/FSHelvetica findfont 10 scalefont setfont} def
gsave
20 400 translate font1 40 63 moveto (1) show
1 1 scale 4 3 8 [0.05 0 0 0.05 0 0]
{slika} image grestore

gsave
150 400 translate font1 40 63 moveto (2) show
20 20 scale 4 3 8 [1 0 0 1 0 0]
{slika} image grestore

gsave
280 400 translate font1 40 63 moveto (3) show
80 60 scale 4 3 8 [4 0 0 3 0 0]
{slika} image grestore

gsave
20 300 translate font1 40 63 moveto (4) show
80 60 scale 4 3 8 [4 0 0 -3 0 3]
{slika} image grestore

gsave
300 300 translate font1 65 42 moveto (5) show
80 60 scale 3 4 8 [4 0 0 3 0 0]
{slika} image grestore

gsave
300 100 translate font1 50 -50 moveto (6) show
200 200 scale 4 3 8 [8 2 4 3 0 0]
{slika} image grestore

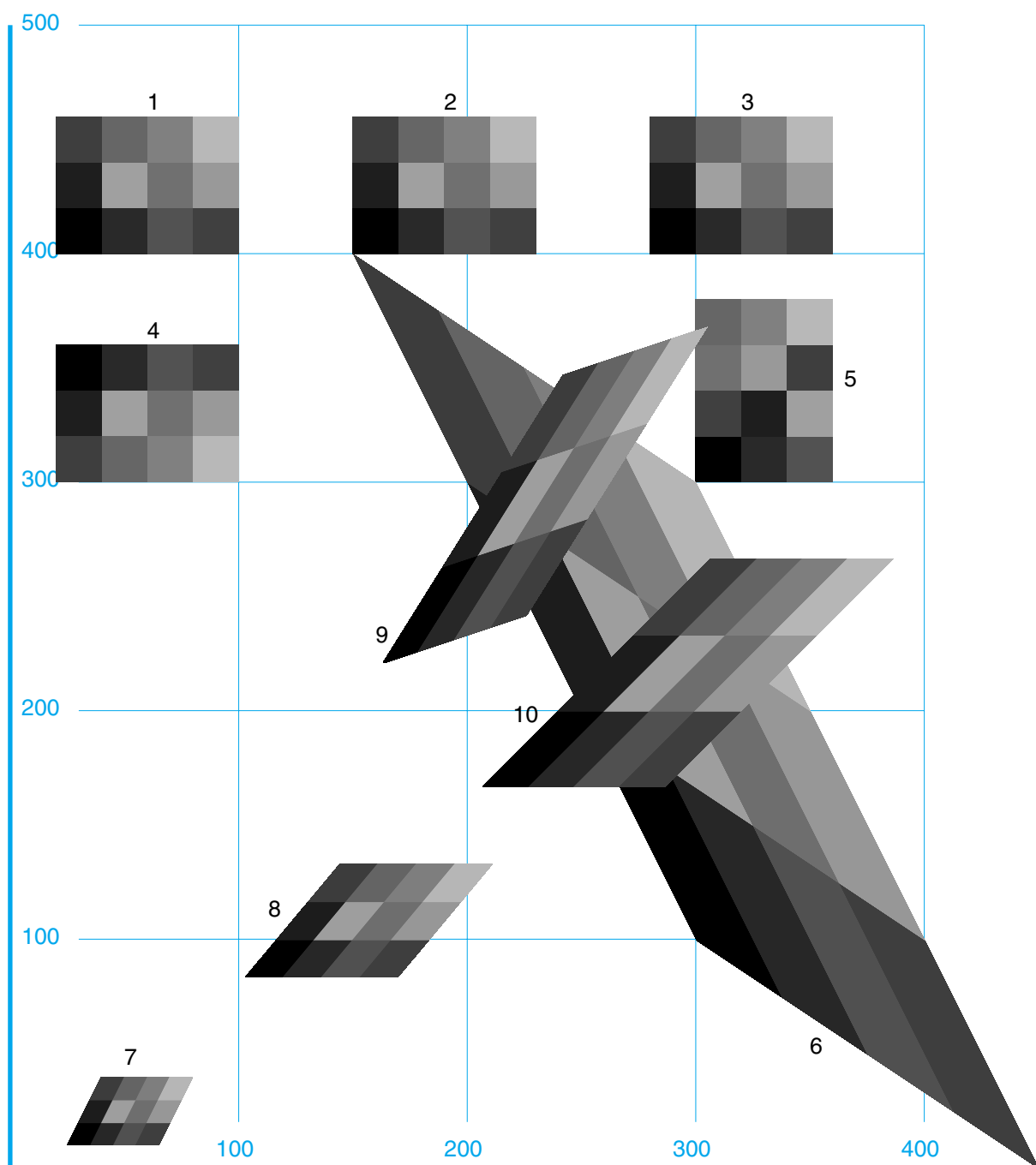
gsave
0 0 translate font1 50 45 moveto (7) show
1 1 scale 4 3 8 [0.10 0 -0.05 0.10 -2 -1]
{slika} image grestore

gsave
0 0 translate font1 113 110 moveto (8) show
10 10 scale 4 3 8 [0.6 0 -0.5 0.6 -2 -5]
{slika} image grestore
%%% Isto bi se dobilo sa 1 1 scale i [0.06 0 -0.05 0.06 -2 -5]

gsave
0 0 translate font1 160 230 moveto (9) show
1 1 scale 4 3 8 [0.08 -0.01 -0.05 0.03 -2 -5]
{slika} image grestore

gsave
0 0 translate font1 220 195 moveto (10) show
1 1 scale 4 3 8 [0.05 0 -0.05 0.03 -2 -5]
{slika} image grestore showpage

```



S pikslima koji opisuju sliku GRGA demonstriraju se paralelogramske transformacije realne slike. U svim primjerima korišten je postupak određivanja veličine slike s relacijama  $S=A$  i  $R=D$ , a veličina slike se kontrolira s parametrima  $W$  i  $T$  naredbe `scale`.

Transformacijski kutevi slika su:

	$\alpha^\circ$	$\beta^\circ$	$\gamma^\circ$	$\delta^\circ$
1.	$0^\circ$	$90^\circ$	$-45^\circ$	$-56^\circ$
2.	$20^\circ$	$90^\circ$	$54^\circ$	$62^\circ$
3.	$0^\circ$	$-70^\circ$	$-36^\circ$	$-44^\circ$
4.	$20^\circ$	$63^\circ$	$-52^\circ$	$-77^\circ$

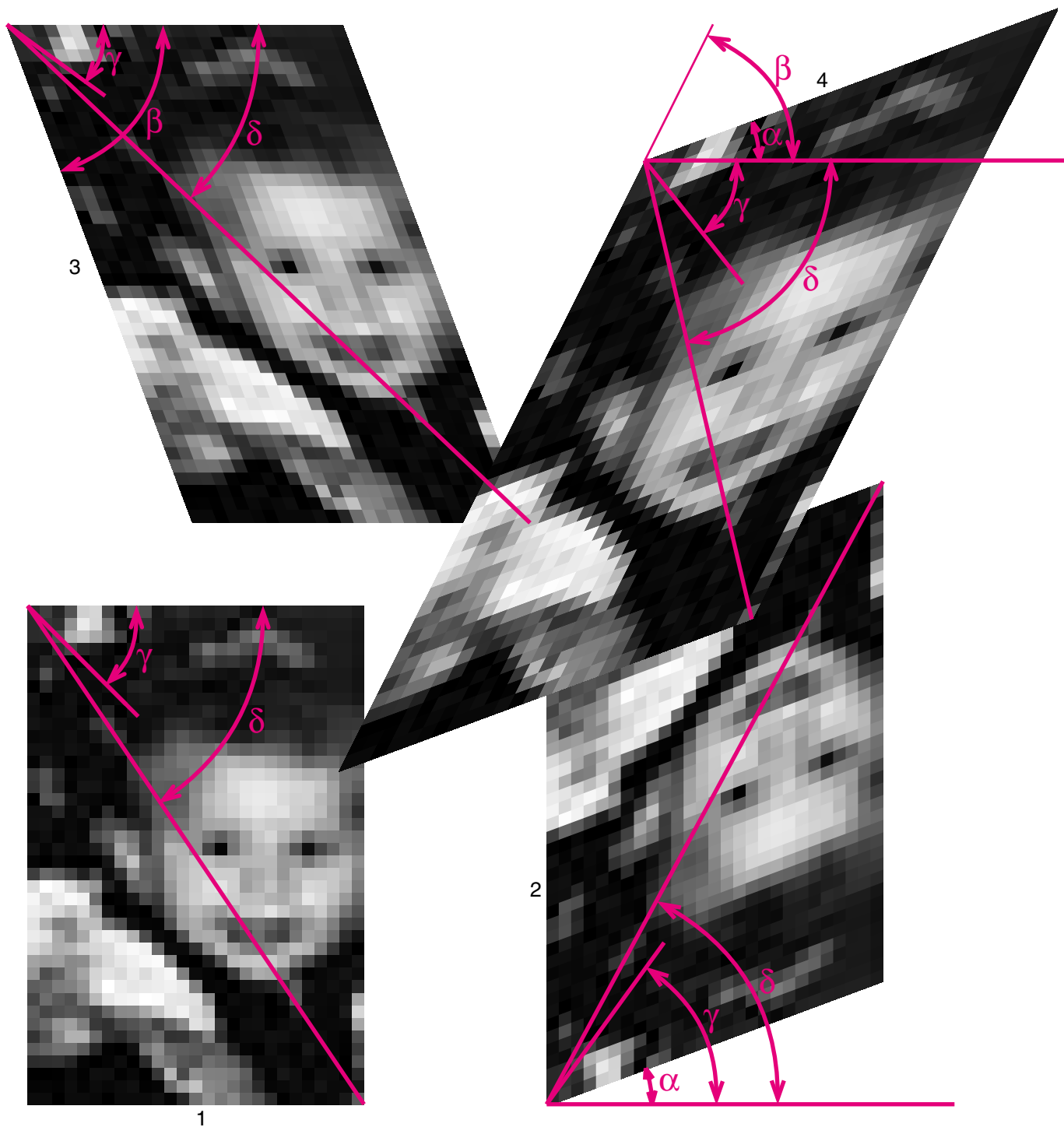
```
/font1 {/FSHelvetica findfont
10 scalefont setfont} def
```

```
gsave
50 70 translate font1 81 -10 moveto (1)
show
162 240 scale
27 40 8 [27 0 0 40 neg 0 40] {GRGA}
image grestore
```

```
gsave
300 70 translate font1 85 15 moveto (2)
show
162 240 scale
27 40 8 [27 -10 0 40 0 0] {GRGA}
image grestore
```

```
gsave
130 350 translate font1 -60 120 moveto (3)
show
162 240 scale
27 40 8 [27 0 15 40 neg 0 40] {GRGA}
image grestore
```

```
gsave
200 230 translate font1 230 330 moveto (4)
show
162 240 scale
27 40 8 [27 10 -20 40 neg 0 40] {GRGA}
image grestore
showpage
```



1-bitna svjetlina		razina sivog	
bin.kod	dec	255-dec	postotak
0	0	255	100%
1	255	0	0%

2-bitna svjetlina		razina sivog	
bin.kod	dec	255-dec	postotak
00	0	255	100,00%
01	85	170	66,67%
10	170	85	33,33%
11	255	0	0,00%

4-bitna svjetlina		razina sivog	
hex-kod	dec	255-dec	postotak
0	0	255	100,00%
1	17	238	93,33%
2	34	221	86,67%
3	51	204	80,00%
4	68	187	73,33%
5	85	170	66,67%
6	102	153	60,00%
7	119	136	53,33%
8	136	119	46,67%
9	153	102	40,00%
A	170	85	33,33%
B	187	68	26,67%
C	204	51	20,00%
D	221	34	13,33%
E	238	17	6,67%
F	255	0	0,00%

Broj razina sive skale je prva karakteristika digitalne slike. PostScript ima mogućnost interpretirati sliku sa 2, 4, 16, 256 i 4096 stepenica sivih tonova. Parametrom G (1, 2, 4, 8, 12) zadanim neposredno prije matrice transformacije piksla, određujemo broj bitova, a time i stepenice sive skale. Prvi redak našeg uzorka dat je heksadecimalno odnosno binarno:

3 E 0 3 C 1 F C  
 0011 1110 0000 0011 1100 0001 1111 1100

Za raspored od 2 bita po pikslu imamo:

00 11 11 10 00 00 00 11 11 00 00 01 11 11 11 00  
 100% 0% 0% 33% 100% 100% 100% 0% 0% 100% 100% 66% 0% 0% 0% 100%

Za raspored od 4 bita po pikslu imamo:

0011 1110 0000 0011 1100 0001 1111 1100  
 80% 6,67% 100% 80% 20% 93% 0% 20%

Za raspored od 8 bita po pikslu imamo:

0011 1110 0000 0011 1100 0001 1111 1100  
 75% 99% 24% 1%

U primjerima sa 2, 4, 16 i 256 razina sivog naredba image je podešena da su piksli u ispisu kvadrati dimenzije 12 točaka.

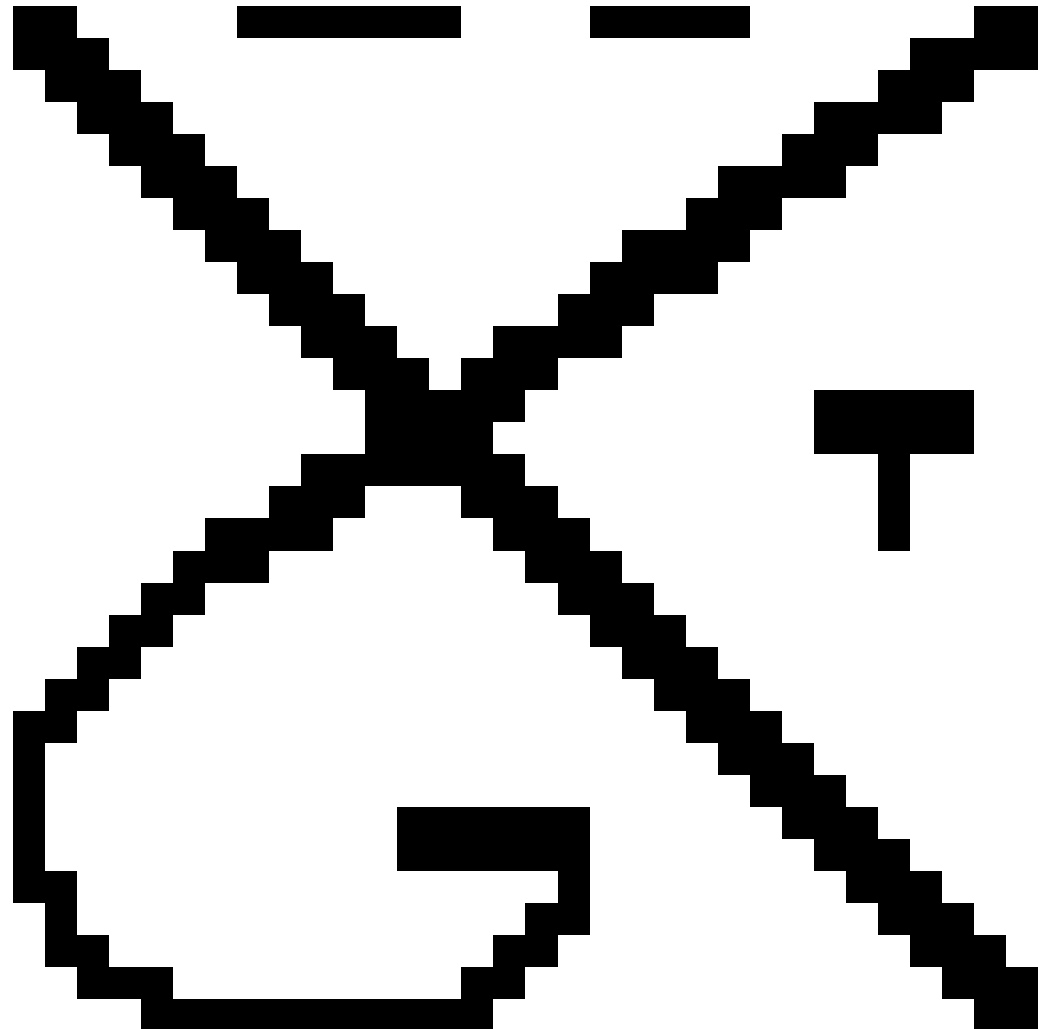
```
% 2 razine sivog=1 bit po pikslu
384 384 scale
```

```
32 32 1
[32 0 0 32 neg 0 32]
{<
```

```
3E03C1FC
1FFFFFF0
8FFFFFFE3
C7FFFF87
E3FFFF1F
F1FFFC3F
F8FFF8FF
FC7FE1FF
FE3FC3FF
FF1F8FFF
FF8E1FFF
FFC47FFF
FFE0FF83
FFE1FF83
FF80FFEF
FF1C7FEF
FC3E3FEF
F8FF1FFF
F3FF8FFF
E7FFC7FF
CFFFE3FF
9FFFF1FF
3FFFF8FF
7FFFFC7F
7FFFFE3F
7FF03F1F
7FF03F8F
3FFFBFC7
BFFF3FE3
9FFE7FF1
C7FCFFF8
F001FFFC
>}
```

```
image
```

```
showpage
```

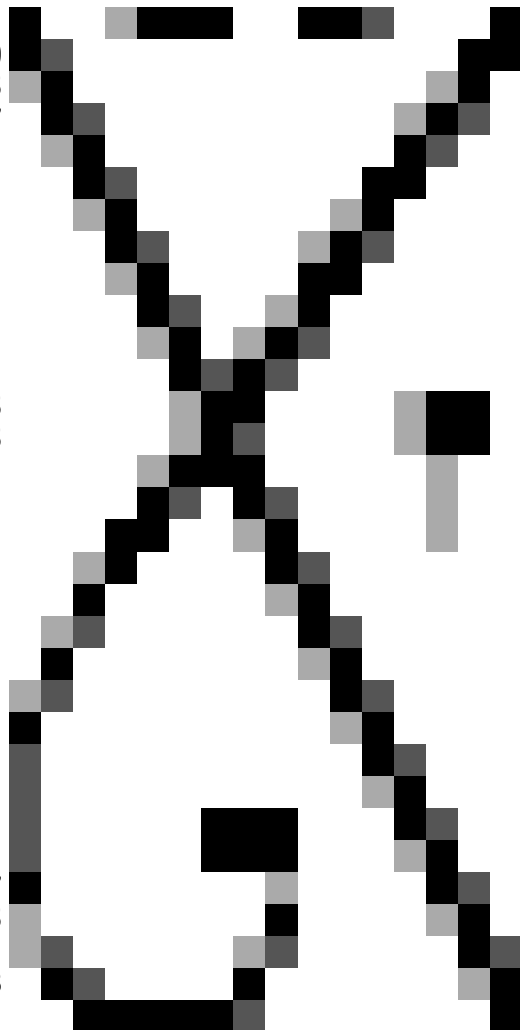


```
% 4 razine sivog=2 bit po pikslu
192 384 scale
```

```
16 32 2
[16 0 0 32 neg 0 32]
```

```
{<
```

```
3E03C1FC
1FFFFFF0
8FFFFFFE3
C7FFFF87
E3FFFF1F
F1FFFC3F
F8FFF8FF
FC7FE1FF
FE3FC3FF
FF1F8FFF
FF8E1FFF
FFC47FFF
FFE0FF83
FFE1FF83
FF80FFEF
FF1C7FEF
FC3E3FEF
F8FF1FFF
F3FF8FFF
E7FFC7FF
CFFFE3FF
9FFFF1FF
3FFFF8FF
7FFFFC7F
7FFFFE3F
7FF03F1F
7FF03F8F
3FFFBFC7
BFFF3FE3
9FFE7FF1
C7FCFFF8
F001FFFC
```



```
>}
```

```
image
```

```
showpage
```

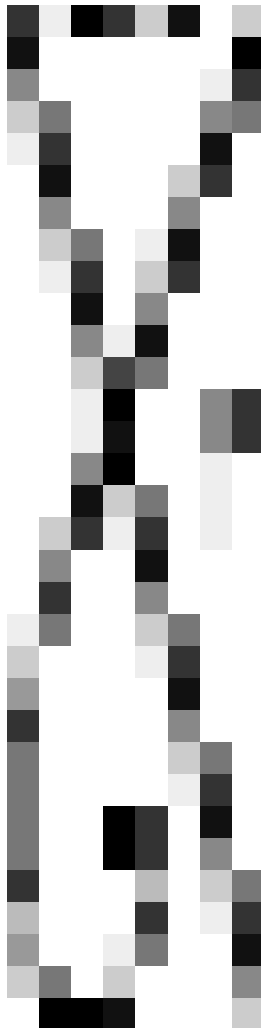


```
% 16 razine sivog=4 bit po pikslu
96 384 scale
```

```
8 32 4
[8 0 0 32 neg 0 32]
```

```
{<
```

```
3E03C1FC
1FFFFFFF0
8FFFFFFE3
C7FFFF87
E3FFFF1F
F1FFFC3F
F8FFF8FF
FC7FE1FF
FE3FC3FF
FF1F8FFF
FF8E1FFF
FFC47FFF
FFE0FF83
FFE1FF83
FF80FFEF
FF1C7FEF
FC3E3FEF
F8FF1FFF
F3FF8FFF
E7FFC7FF
CFFE3FFF
9FFFF1FF
3FFFF8FF
7FFFFC7F
7FFFFE3F
7FF03F1F
7FF03F8F
3FFFBFC7
BFFF3FE3
9FFE7FF1
C7FCFFF8
F001FFFC
```



```
>}
```

```
image
```

```
showpage
```

```
% 256 razine sivog=8 bit po pikslu
48 384 scale
```

```
4 32 8
[4 0 0 32 neg 0 32]
```

```
{<
```

```
3E03C1FC
1FFFFFFF0
8FFFFFFE3
C7FFFF87
E3FFFF1F
F1FFFC3F
F8FFF8FF
FC7FE1FF
FE3FC3FF
FF1F8FFF
FF8E1FFF
FFC47FFF
FFE0FF83
FFE1FF83
FF80FFEF
FF1C7FEF
FC3E3FEF
F8FF1FFF
F3FF8FFF
E7FFC7FF
CFFE3FFF
9FFFF1FF
3FFFF8FF
7FFFFC7F
7FFFFE3F
7FF03F1F
7FF03F8F
3FFFBFC7
BFFF3FE3
9FFE7FF1
C7FCFFF8
F001FFFC
```



```
>}
```

```
image
```

```
showpage
```

Reprodukcija fotografije danas je najčešća sa 8 bitnim razinama sivog. Ljusko oko dobro razlikuje 50 stepenica sive skale za što bi bilo dovoljno 6 bita ( $2^6 = 64$ ). Standard od 8 bita proširen je i na boju RGB, CMYK. Crno bijele slike maloga Grge prezentirane su s jednim, dva, četiri i osam bita. Proporcionalno tome, memorija pojedinih slika je:

stepenice sivog	broj bitova po pikslu	broj piksla		memorija (byte)
		horiz.	vert.	
2	1	825	1241	127.978
4	2	825	1241	225.956
16	4	825	1241	511.912
256	8	825	1241	1.023.825

Slike su ispisane na jednake dimenzije 148.5 x 223.38 točaka (naredba `scale`).

2 stepenice sivog



```
148.5 223.38 scale
825 1241 1 [825 0 0 1241 neg 0 1241]
{<00000000000000000000000000000000...
...00000000000000000000000000000000>}
image showpage
```

4 stepenice sivog



```
148.5 223.38 scale
825 1241 2 [825 0 0 1241 neg 0 1241]
{<00000000000000000000000000000000...
...5555555555556AAAAAAAAAAAA>}
image showpage
```

16 stepenice sivog



```
148.5 223.38 scale
825 1241 4 [825 0 0 1241 neg 0 1241]
{<112222223332233333333333...
...888888889988899888899989>}
image showpage
```

256 stepenice sivog



```
148.5 223.38 scale
825 1241 8 [825 0 0 1241 neg 0 1241]
{<1D1F22272827292E2F303031...
...8E919189838B8D9093908D91>}
image showpage
```


U standardnim programima za piksel grafiku, kao što je npr. Photoshop, piksl na ekranu je uvijek kvadratične dimenzije. Veličina piksla se zadaje implicitno preko pojma rezolucije, odnosno brojem piksla po nekoj dužinskoj mjernoj jedinici npr. po inchu ili centimetru. Na primjer rezolucija 400 ppi (pixl per inch) određuje kvadratični piksel čija je stranica velika 0,18 tipografskih točaka (1 inch=72tp). Ako se širina i visina cijele slike u takvim programima ne izrazi kao višekratnik od 0,18tp tada se ulazi u za njih nedozvoljeno stanje koje se može ispraviti jedino zaokruživanjem na najbliže višekratnike od 0,18 tp. To tržišni softveri kao Photoshop rade pri upisivanju željene dimenzije slike automatski pretvarajući nedozvoljeno upisanu dimenziju u najbliži višekratnik dimenzije piksla.

Piksel se može opisati i kao prostor slike koji je na cijeloj svojoj površini istog tona. Skaniranjem, prostor piksla integrira sve tonove na tom području, usrednjuje ih u samo jednu vrijednost. Ukoliko je piksel grub (to je relativno) tada su nakon skaniranja ili resempliranja zauvijek izgubljeni detalji manji od piksla.

Četiri 8-bitne slike demonstriraju promjene detalja smanjenjem memorije, odnosno smanjenjem broja piksla iste slike za faktor 10 od 1MB do 1kB.

slika	broj piksla		memorija		
	horiz.	vert.	(byte)		
1	825	1241	1.023.825	≈	1 MB
2	260	391	101.660	≈	100 kB
3	80	211	9.680	≈	10 kB
4	27	40	1.080	≈	1 kB

Veličina piksla određuje se ispisom, prikazom slike. Dobri rezultati se dobe ako je ispis slike oko 300 piksla po inču. Budući da je kvaliteta otiska određena veličinom piksla, prilikom skaniranja fotografije, rezolucija skaniranja podređena je dvema brojkama: povećanje (smanjenje) originala i broja piksla po dužnom inču (najčešće 300). U obzir može biti uzet i treći faktor - udaljenost gledanja otiska. Značajan je ako se otisci gledaju iz veće udaljenosti (plakati) jer dozvoljavaju proporcionalno povećanje dimenzije piksla u otisku. Predložimo grubu relaciju:



$$\frac{b}{a} \approx 3000, \text{ gdje je } a \text{ veličina stranice jednog piksla}$$

ili

$$\text{udaljenost (inč)} \times \text{broj piksla / (inču)} \approx 3000$$

$$\text{udaljenost (cm)} \times \text{broj piksla / (centimetru)} \approx 3000$$



400 ppi

1.023.825 B  
≈ 1 MB

```
148.5 223.38 scale
825 1241 8 [825 0 0 1241 neg 0 1241]
{<1D1F22272827292E2F303031...
...8E919189838B8D9093908D91>}
image showpage
```



126 ppi

101.660 B  
≈ 100 kB

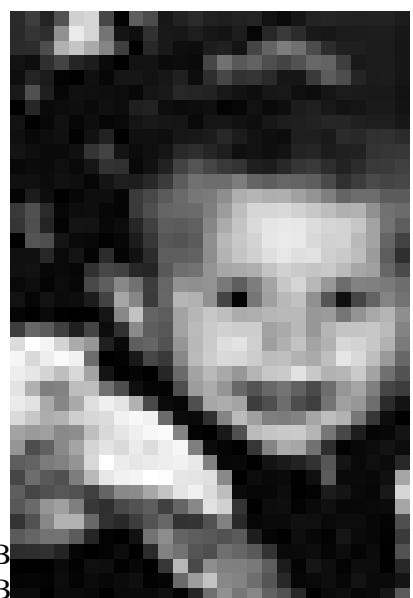
```
148.5714 223.4286 scale
260 391 8 [260 0 0 391 neg 0 391]
{<20272C2F3134333535333434...
...5D646B7277797D898A888B8D>}
image showpage
```



39 ppi

9.680 B  
≈ 10 kB

```
147.6923 223.3846 scale
80 121 8 [80 0 0 121 neg 0 121]
{<25313435353638292A324DCA...
...0D0C0D0E1118273B51657F91>}
image showpage
```



13 ppi

1.080 B  
≈ 1 kB

```
149.5385 221.5385 scale
27 40 8 [27 0 0 40 neg 0 40]
{GRGA}
image showpage
```

Pikseli se mogu ispisivati sivi (Gray), RGB ili CMYK sa komandom `colorimage`:

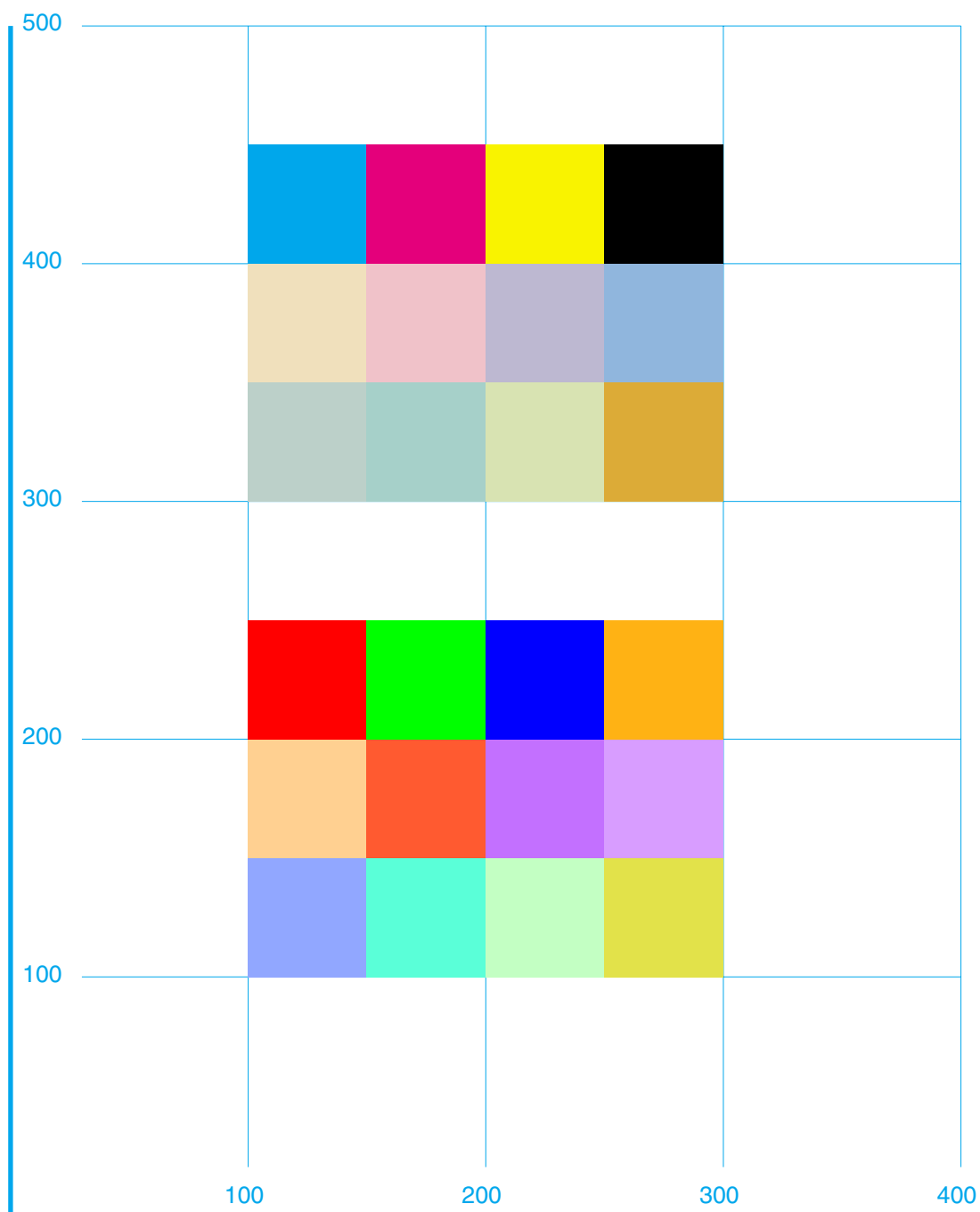
```
4 3 8 [ 4 0 0 -3 0 3] {string1 slike}...{stringN slike} METODA N colorimage
```

transformacijska matrica

Prva tri parametra su istog značenja kao i kod `image` naredbe. S parametrom `N` definira se kolorni prostor odnosno broj komponenata boje po pikslu. Za sivi zapis (ponaša se kao `image` komanda) `N=1`, za RGB `N=3`, a za CMYK `N=4`. RGB i CMYK komponente mogu se zapisivati na jednostruki i višestruki način. U jednostrukom zapisu se sve komponente zapisuju u jednom stringu naizmjenično piksl po piksl (npr.  $R_1 G_1 B_1 R_2 G_2 B_2 R_3 G_3 B_3 \dots$ ), a u višestrukom se komponente zapisuju u odvojenim stringovima kao u primjeru na ovoj stranici. Ako je parametar `METODA` *true* definira se višestruki, a ako je *false* definira se jednostruki zapis komponenata.

```
gsave
100 100 translate
200 150 scale
4 3 8
[4 0 0 -3 0 3]
{<FF0000FFFFFFC4D99159C4E3>}      prvih 12 boja sa 57 str.
{<00FF00B3D159709EA8FFFE3>}
{<0000FF0F912BFFFFFFD9C447>}
true
3
colorimage
grestore
```

```
gsave
100 300 translate
200 150 scale
4 3 8
[4 0 0 -3 0 3]
{<FF000000140F4D8F456E2E00>}      osnovne 4 i
{<00FF0000265C3B2600000061>}      8 zadnjih boja sa 57 str.
{<0000FF00571C00002E456EE6>}
{<000000FF00001C0F260F0F26>}
true
4
colorimage
grestore
showpage
```



## INDEX

- add, 37, 42
- and, 91
- arc, 18, 20
- arcn, 20
- arcto, 20, 24
- array, 92
- ashow, 82, 84
- atan, 42, 37
- awidthshow, 86
  
- charpath, 72, 74
- ciscvektor, 100
- clear, 36
- clip, 76
- closefile, 104
- closepath, 8
- colorimage, 142
- copy, 37
- crodiجلي, 108
- crovektor, 108
- ctekst, 94
- curveto, 22, 24, 26, 37, 44
  
- div, 42, 37
- dup, 36, 38
- dvostruki, 106
  
- eofill, 34
- eq, 90
- exch, 36
  
- file, 104
- fill, 10
- findfont, 66, 68
- for, 46, 48, 50
- forall, 92, 93
  
- ge, 90
- get, 70, 92, 93
- getinterval, 92, 93
- grestore, 30
  
- gsave, 30
- gt, 90
  
- idiv, 37
- if, 91
- ifelse, 91
- image, 121, 123
- index, 37, 40
- insert, 98
- izbaciclan, 106
  
- jevokal, 102
- jtekst, 96
  
- kshow, 88
  
- le, 90
- length, 84, 92, 93
- lineto, 6
- lt, 90
- ltekst, 94
  
- makefont, 80
- mod, 37
- moveto, 6
- mreza, 50, 52, 53
- mul, 42, 37
  
- ne, 90
- neg, 37, 38
- newpath, 76
- not, 91
  
- operatori, 90 - 93
- or, 91
  
- piksel, 118, 139
- pop, 36
- prelom, 111
- procedure, 38
- privizadnji, 108
- put, 92, 93
  
- putinterval, 92, 93
- readstring, 104
- repeat, 28, 50
- rlineto, 8
- roll, 36
- rotate, 30
- rtekst, 94
  
- saturacija, 58 - 61
- scale, 32, 123, 129
- scalefont, 66, 68, 70
- search, 93
- setcmykcolor, 54, 62
- setdash, 16
- setfont, 66, 68
- setgray, 10
- sethsbcolor, 58 - 61, 62
- setlinecap, 12
- setlinejoin, 14
- setlinewidth, 8
- setmiterlimit, 14
- setrgbcolor, 56, 62
- show, 66, 68
- showpage, 6
- sort, 42
- spoji, 98
- sqr, 42
- stack, 36
- string, 92
- stringwidth, 84
- stroke, 6
- strokepath, 78
- sub, 37, 42
  
- tipografija, 65
- translate, 18
  
- widthshow, 86
- writestring, 104
  
- xor, 91
- xyshow, 86



## LITERATURA

1. I. Adobe Systems: „PostScript Language Reference Manual”, Addison-Wesley, 1985
2. I. Adobe Systems: „PostScript Language Tutorial and Cookbook”, Addison-Wesley, 1985
3. I. Adobe Systems, G. C. Reid: „PostScript Language Program Design”, Addison-Wesley, 1988
4. S. F. Roth: „Real World PostScript”, Addison-Wesley, 1988
5. D. Holzgang: „PostScript Programmer’s Reference Guide”, Scott, Foresman and Company, 1989
6. R. Smith: „Learning PostScript: A Visual Approach”, Peachpit Press, 1990
7. I. Adobe Systems: „PostScript Language Reference Manual”, Addison-Wesley, 1990
8. C. R. Glenn: „Thinking in PostScript”, Addison-Wesley, 1990
9. I. Adobe Systems: „Adobe Type 1 Font Format”, Addison-Wesley, 1990
10. H. McGilton, M. Campione: „PostScript by Example”, Addison-Wesley, 1992
11. P. Fink: „PostScript Screening: Adobe Accurate Screens”, Adobe Press, 1992
12. T. Merz: „PostScript & Acrobat, PDF”, Springer, 1996

Ime PostScript® je registrirani znak u vlasništvu Adobe Systems Incorporated