T A R T U Ü L I K O O L
MATEMAATIKA-INFORMAATIKATEADUSKOND
Arvutiteaduse instituut
Tarkvarasüsteemide õppetool
Informaatika eriala

Jaanus Hansen

Graafikakeel SWOG

Bakalaureusetöö (10 AP)

Juhendaja: Jaak Vilo, PhD

Autor: …… "….." mai 2005
Juhendaja: ……. "….." mai 2005
Õppetooli juhataja: …… "….." …….. 2005

TARTU 2005

Sisukord

Sissejuhatus

Andmete inimesele arusaadaval kujul esitamise üks parimaid võimalusi on nende visuaalselt tajutaval kujul näitamine. Andmete visualiseerimise lihtsamaks muutmine ongi käesoleva töö üldine eesmärk.

Konkreetseks eesmärgiks on töötada välja võimalikult lihtne väljenduskeel piltide kirjeldamiseks. Keel on mõeldud olukorraks, kus visualiseeritavate andmete omapära tõttu pildi graafikaredaktori abil joonistamine ei ole reaalselt teostatav. Näiteks tuleb andmeid kujutada programmist, kui neid genereeritakse vastusena kasutaja päringule. Tänapäeval on programmist piltide joonistamiseks välja pakutud palju võimalusi, eesmärgiks on, et loodav keel eristuks teistest süntaksi lihtsuse ning väljundi formaatide paljususega.

Keele nimi SWOG on lühend inglise keelsetest sõnadest *Simple Web Object Graphics*[1], mis näitavad, et pildil olevaid kujundeid käsitletakse objektidena. Sellest tulenevalt saab keele interpretaator väljastada pildi vektorgraafika formaadis. Samas on sisseehitatud võimalus väljastada pilte ka rastergraafikas, mis on tänapäeval veebis laialdasemalt kasutatud. Samuti on keele abil võimalik luua lihtsamaid animatsioone.

Lisaks tavalisele joonistamisele on eesmärgiks võimaldada ka pildile paigutatud kujundite põhist lähenemist. Loodud keeles on võimalik pildi kirjelduse ning etteantud koordinaatide järgi tuvastada antud punkti alla jäävad objektid. Antud omaduse tõttu on keelega SWOG mugav luua interaktiivseid visualiseerimisvahendeid, sest peale pildile kujundite lisamist on automaatselt võimalik kasutada ka kliki järgi kujundite tuvastamist. Lisaks serveripoolsele punkti alla jäänud objektide tuvastamisele võimaldab keel väljastada ka lingiga seotud kujundite HTML formaadis klikkimise kaardi[2].

Keelele lisab võimsust ka juba pildile lisatud objektide omaduste muutmisvõimalus. See on mõeldud interaktiivse süsteemi loojale, näiteks võib tal kerkida vajadus klikitud kujundi värvi muutmiseks. Teiste joonistusvahenditega tuleks üldjuhul kogu pilt uuesti „joonistada“, samas SWOG võimaldab lisada olemasolevale pildi kirjeldusele vajaliku kujundi värvi muutmise käsu.

Keele süntaksi loomisel on võetud eesmärgiks, et võrreldes graafikateegi kasutamisega oleks

1 Vaba tõlge: lihtne veebi vektorgraafika, inglise keelse nime autor on töö juhendaja Jaak Vilo.
2 Inglise keeles HTML *clickmap.*

joonistamine kiirem ja lihtsam. Kasutaja elu lihtsustab võimalikult paljude arvutuste tegemise ülesande keele interpretaatorile suunamine. Näiteks saab loodavas keeles koordinaatide süsteemi muuta, et pildi kirjeldamisel ei peaks kasutaja andmeid pildi jaoks teisendama vaid saaks joonistuskeskkonna oma andmetele vastavaks muuta. Samuti aitab joonistamise kiirendamisele kaasa eeldefineeritud keerukamate kujundite, näiteks puude, kasutamise võimaldamine. Eesmärgiks on, et keelde uute kujundite lisamine oleks tehniliselt lihtne, et kasutajate erivajadustele oleks võimalik kiirelt vastu tulla. Lisaks on joonistamise kiirendamiseks lisatud võimalus pildi kirjelduses objektide defineerimine ning nende korduvkasutamine.

Et piltide kirjeldused võtaks võimalikult vähe salvestusruumi on eesmärgiks, et loodud graafikakeeles saaks pilte võimalikult lakooniliselt esitada. Samas, et pildi kirjeldus jääks inimsilmale mõistetavaks ei ole eesmärgiks lakoonilisust liialt nõuda.

Töö praktilise osa raames on välja töötatud keele süntaks ning kirjutatud keelt interpreteeriv tarkvara. Valmis Perli moodul, mis SWOG keelse pildi kirjelduse põhjal väljastab PNG, GIF või SVG formaadis pildi või GIF animatsiooni. Samuti on realiseeritud etteantud punkti alla jäänud kujundite tuvastamine ning pildi HTML formaadis klikkimise kaardi väljastamine. Loodud on ka Perli skript, mis võimaldab keele interpretaatorit kasutada käsurealt. Keele süntaksi tutvustamiseks valmis näidetega illustreeritud kasutusjuhend. Lisaks on keele interpretaatori üldine struktuur realiseeritud ka keeles C.

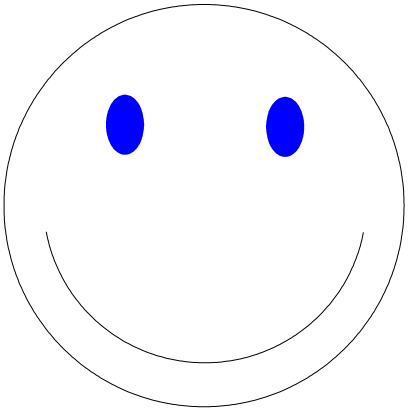
Keele süntaks sisaldab 45 käsku, lihtsamatest graafika primitiividest saab pildile lisada jooni, ristkülikuid, hulknurki, ellipseid, sektoreid ning kaari. Samuti on võimalik lisada tekste ning rasterpilte. Kujundil saab määrata ka värve, kusjuures kasutada saab ka poolläbipaistvaid toone. Koordinaatide süsteemi on võimalik skaleerida, pöörata ning nihutada. Primitiividest koosnevaid eeldefineeritud kujundeid on realiseeritud kolm: puu, koordinaatide telg ning graafik, mille abil saab visualiseerida kaetud väärtuste vahemikke.

Käesolev kirjatükk koosneb kolmest peatükist. Esimeses peatükis kirjeldatakse loodud keele võimalusi ja täpset süntaksit. Teine peatükk kajastab keele realisatsiooni: kerkinud probleeme ning lahendusi, kasutatud tööriistu ja töövahendite kiiruste võrdlust. Kolmandas peatükis tutvustatakse keele praktilisi kasutusvõimalusi.

Peatükk 1
Graafikakeel SWOG

1.1. Olemasolevad alternatiivid keele SWOG

Antud töö puhul on ilmselt lugejal kange kahtlus, et tegu on jalgratta leiutamisega, sest on üsna loogiline, et loodav visualiseerimisvahend peaks olema juba realiseeritud. Et hetkeolukorda uurida, on järgnevalt esitatud ülevaade vahenditest, mille abil saab programmist pilte genereerida. Ülevaade ei ole küll sugugi täielik, aga tuntumad ja paremad vahendid on püütud ära mainida. Võimaluse korral on toodud ka näide, milliseid käske tuleb kasutada, et joonistada pilti, mis on toodud joonisel 1.



Joonis 1: Näo pilt.

1.1.1. Graafika API kasutamine

Ilmselt on enamikes programmeerimiskeeltes realiseeritud ka funktsioonid, mille abil saab rastergraafikat väljastada. Näiteks võib tuua GD teegi, mida saab kasutada mitmetes keeltes. Joonisel 1 kujutatud pildi joonistamine Perl keeles kasutades GD teeki on toodud järgmisel leheküljel näites 1. Tihti on graafika väljastamise käsud keelde sisse ehitatud, näiteks Java keeles saab kasutada klassi *java.awt.Graphics* meetodeid.

Otse programmist funktsioonide välja kutsumise teel piltide loomise peamiseks eeliseks on ilmselt töökiirus, sest vahepeal ei toimu mingit argumentide parsimist. Kuid API kasutamisel on ka puudusi. Üks põhimõttelisi probleeme on programmeerimiskeelest sõltuvus, sest iga keele jaoks tuleb eraldi õppida, kuidas selles joonistatakse. Järjekordse joonistamise kallale asudes tuleb uurida millised on konkreetsed käsud ning kuidas interpreteeritakse parameetreid (näiteks kas ringi joonistamisel antakse mõõduna diameeter või raadius).

Samuti on graafika API kasutamine seetõttu ebamugav, et selles realiseeritud primitiivide maht on väike, näiteks tuleb ringi joonistamiseks kasutada ellipsi loomise käsku.

```
use GD;
#pildi kanga hõivamine
$im = new GD::Image(260,260);
#värvide allokeerimine
$black = $im->colorAllocate(0,0,0);
$white = $im->colorAllocate(255,255,255);
$blue = $im->colorAllocate(0,0,255);
$im->fill(0,0,$white);
$im->arc(150,150,200,200,0,360,$black);
$im->arc(150,150,160,160,10,170,$black);
$im->filledEllipse(110,110,20,30,$blue);
$im->filledEllipse(190,110,20,30,$blue);
#tagastame pildi st. väljundisse
print $im->png;
```

Näide 1: Näo joonistamine keeles Perl kasutades GD.pm moodulit.

1.1.2. MetaPost

MetaPost on graafikakeel, mis on keele MetaFont[3] edasiarendus. Sellest tulenevalt on tegu võimsa keelega, millega saab väljundit toimetada just vastavalt kasutaja vajadustele [Hob92].

Joonisel 1 kujutatud näo kirjeldus keeles MetaPost on toodud näites 2.

```
beginfig(1);
 %näo ümbruse ring:
 draw fullcircle scaled 200 shifted (150, 150);
 %suu kõver tuleb ise punktidena anda
 draw (229,135)..(150,70)..(73,135);
 for i=110 step 80 until 190:
   fill fullcircle xscaled 20 yscaled 30
        shifted ( i ,190) withcolor blue;
 endfor
endfig;
```

Näide 2: Näo kirjeldus keeles MetaPost.

Näitest on näha, et keeles saab kasutada ka muutujaid ning isegi tsükleid, mis teeb keele programmeerijatele meelepäraseks. Teisest küljest ei ole joonistamine piisavalt mugav, sest süntaks on suhteliselt keeruline. Siiski on MetaPost lihtsalt kasutatav ka visualiseerimisvahendite loomiseks [KK04].

3 MetaFont on programm, mille abil luuakse fonte tekstitoimetile TeX, fonte kirjeldatakse graafikakeeles, mis kuulub Algol 2 perekonda [url:Tob94].

1.1.3. Fly

Fly on programm, mille sisendfailis antakse nii pildi kirjeldus kui käsud, kuidas pilt väljastada – rasterpildina kas faili või standardväljundisse. Kirjeldus on sarnane tavalise graafika API käskudega – iga kujundi jaoks on oma kindel korraldus, näiteks ringi joonistab käsk `circle x,y,r,red,green,blue`, ehk parameetriteks antakse keskpunkti koordinaadid, raadius ning joone värv [url:Gle]. Joonisel 1 kujutatud näo kirjeldab näites 3 toodud Fly kood.

```
new
size 256,256
type png
fill 1,1,255,255,255
circle 150,150,200,0,0,0
arc 150,150,160,160,10,170,0,0,0
ellipse 110,110,20,30,0,0,255
fill 110,110,0,0,255
ellipse 190,110,20,30,0,0,255
fill 190,110,0,0,255
```

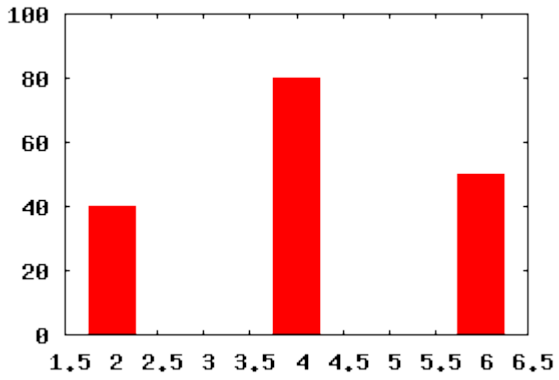
Näide 3: Näo kirjeldus programmi Fly jaoks.

Pildi kirjeldamise muudab muidugi igale kujundile värvi kaasa andmine tülikaks, samuti pole realiseeritud võimalust koordinaatide süsteemi transformeerimiseks. Siiski on tegu tööriistaga, millega saab pilte luua programmeerimiskeelest sõltumatult suhteliselt kiirelt ja mugavalt. Fly on ka käesolevaks tööks inspiratsiooni andja, nimelt kasutas töö juhendaja Jaak Vilo seda tööriista nii, et pildi kirjeldusse lisati kommentaaride vormis metainfot, mida rakendus kasutas pildil tehtava hiireklõpsu tuvastamisel järgmisena näidatava pildi määramiseks.

1.1.4. Gnuplot

Gnuplot on tekstiliste käskudega juhitav funktsioonide ning muude andmete diagrammide joonistamise programm [url:Gnuplot]. Kui visualiseeritavad andmed on esitatavad graafiku kujul, siis saab selle tööriistaga neid üsna lihtsalt kujutada, histogrammi loomist demonstreerib näide 4.

```
set style fill solid
#y-telje vahemiku määramine
set yrange [0.00000 : 100.000]
set boxwidth 0.5
#histogrammi väljastamine
plot "-" with boxes title ''
        2       40
        4       80
        6       50
e
```



Näide 4: Histogrammi kirjeldus programmi Gnuplot jaoks.

Paraku on probleemiks, et lisaks väga paljudele graafikuliikidele ei saa pildile mugavalt teisi kujundeid lisada. On küll võimalus noolepeade ja teksti lisamiseks, aga isegi selliste lihtsamate kujundite nagu ristkülikud ja ringid pildile lisamine on suhteliselt vaevarikas. Seetõttu ei ole ka lisatud näo joonistamise näidet, sest selleks pole Gnuplot mõeldud. Samas peab märkima, et tegu on väga võimalusterohke programmiga, sellelt võib saada väljundiks nii raster- kui vektorgraafikat, samuti on võimalik luua animatsioone.

1.1.5. SVG

SVG on kahemõõtmelise graafika kirjeldamise keel, kus kogu pilt on esitatud tekstifaili kujul XML formaadis [url:SVG]. Kuna programmist on andmetest XML väljundi genereerimine suhteliselt lihtne, siis teoreetiliselt on ka graafika väljastamine üsna kerge. Samas on probleemiks, et keeles pole realiseeritud piisavalt primitiive, mistõttu inimesel on pildi tekstifailina kirjeldamine suhteliselt keeruline. Näiteks kaare pildile lisamiseks tuleb määrata, milliste ellipsite lõikumisel kaar tekkib. Joonisel 1 kujutatud näo kirjeldab näites 5 toodud SVG fail.

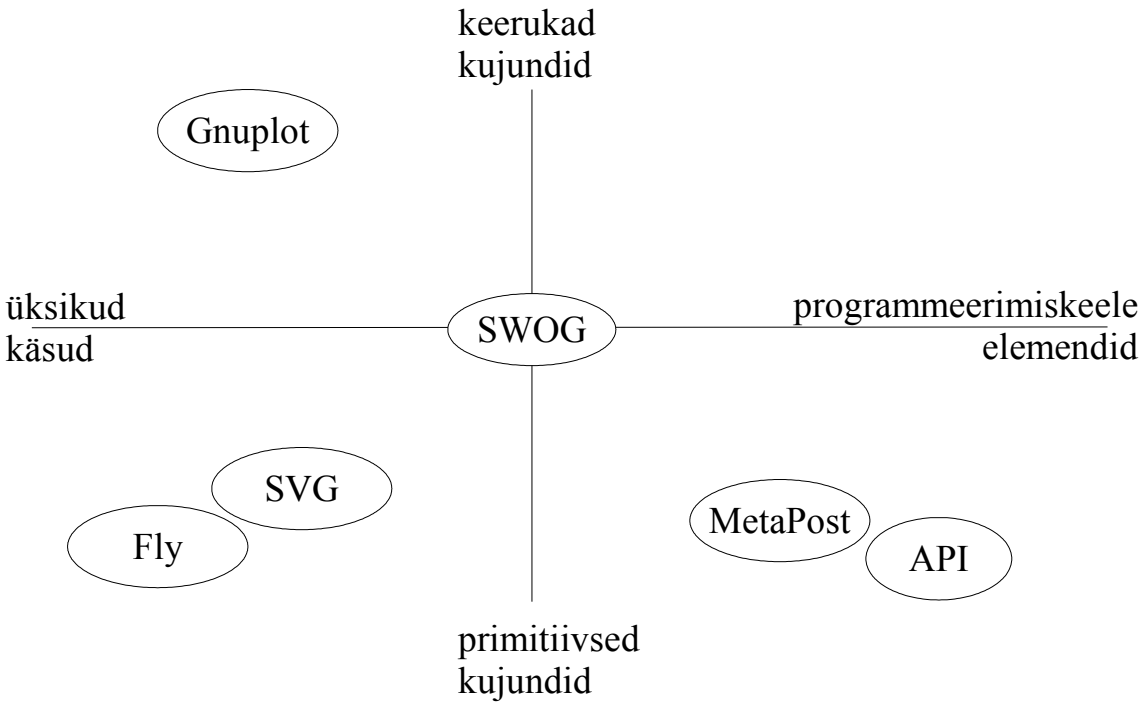
```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
"http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="256" height="256">
  <ellipse ry="100" cx="150" fill-opacity="0" stroke="black"
           rx="100" cy="150"/>
  <!--kaared tuleb anda radadena-->
  <path fill="none" stroke="black"
        d="M229,164 A80,80 0 0,1 71,164"/>
  <ellipse ry="15" fill="blue" cx="110" rx="10" cy="110"/>
  <ellipse ry="15" fill="blue" cx="190" rx="10" cy="110"/>
</svg>
```

Näide 5: Näo kirjeldus keeles SVG.

1.2. SWOG võrreldes alternatiividega

Kokkuvõtvalt vaatleme joonistusvahendeid lähtuvalt kahest aspektist, mis on kujutatud joonisel 2. Vertikaalteljel näidatakse, kas keel võimaldab pildile lisada keerukaid või primitiivseid kujundeid. Primitiivsete all on silmas peetud lihtsaid kujundeid (jooned, ellipsid, ristkülikud) ning keerukamate all primitiivsetest koosnevaid (graafikud, puud). Sellel skaalal paikneb SWOG keskel, sest keeles on käsud mõlemat liiki kujundite joonistamiseks.

Horisontaalteljel on kujutatud pildi kirjeldamisel kasutatava keele süntaksi võimalusterohkust. Kõige suuremad võimalused programmeerimiskeele elementide (tsüklid, muutujad) kasutamiseks on graafika API tarbijatel, väiksemad aga Fly kasutajatel. Sisuliselt on



Joonis 2: Joonistusvahendite võimalused.

horisontaalteljel kujutatud pildi loomise keerukus ning kiirus. Kui pildi sisestamiseks saab kasutada ka muutujaid ja makrosid, siis läheb kasutajal ilmselt joonistamiseks vähem aega kui kõigi üksikute käskude sisestamist nõudva vahendi kasutamiseks. Teisalt on aga esimese variandi puhul keele süntaks märgatavalt keerulisem (vrd. näiteid 2 ja 3). Ka sellel skaalal paigutub SWOG keskele, sest siin saab küll kirjeldada kujundite gruppe ning neid erinevate argumentidega pildile lisada, kuid samas on eesmärgiks hoida süntaksi lihtsust Fly tasemel.

Samuti saab paigutada tööriista SWOG keskele skaalal vektor- vs. rastergraafika, sest see võimaldab väljastada pilti mõlemas formaadis.

1.3. Keele SWOG süntaks

Ilmselt kõige kriitilisem faktor, mis määrab keele kasutuskõlblikkuse on keele süntaks. Kui keele kirjakuju on liiga keeruline, siis ilmselt eriti inimesed ei hakkaks selle õppimisega vaeva nägema. Võib tuua näiteid ka tõeliselt keerulise süntaksiga keelte levikust (PostScript), aga uute keelte suhtes ollakse ilmselt veidi nõudlikumad. Sellest lähtuvalt on süntaks püütud disainida võimalikult lihtsaks ning loogiliseks. Et keel oleks kasutajatele tuttavam ning lihtsam kasutada, siis käskude nimedeks on eeskuju võetud keelest Fly, kuid süntaksi ülesehituse loogika on siiski teistsugune.

1.3.1. Ülevaade

Keele üldpõhimõte on lihtne – iga käsk on eraldi real ning rea üldkuju on järgmine:

```
käsk argumendid /alt/ :nimi <link>.
```

Rea esimene sõna määrab käsu nime ja nimest sõltuvas struktuuris järgnevad käsu argumendid. Iga rea lõpp on aga kõigil käskudel ühine: rea lõpus on kolm mittekohustuslikku parameetrit. Nimelt igale sisseloetud käsule seab keele semantika vastavusse objekti, millele saab anda nime, lingi ning vaba teksti. Nime on objektile vaja, et talle saaks edasises pildi kirjelduses viidata ning punkti järgi kujundite tuvastamisel tabatud kujundite identifitseerimiseks. Link ning vaba tekst on kasutuses pinnalaotusega kujundite HTML kujul klikkimise kaardi väljastamisel, kus kujundi poolt kaetud alale seatakse vastavusse link ning vaba teksti näidatakse lingi vihjena[4]. Samas saab neid kahte parameetrit kasutada ka serveri-poolsetes rakendustes. Keele interpretaatorilt võib küsida, mis on konkreetse nimega objekti ülalmainitud parameetrid ning neid enda rakendusepõhiselt interpreteerida. Veel kasutatakse mõningate käskude puhul vaba teksti sisu SVG formaadi väljastamisel, näiteks fondi kirjeldamiseks.

Lisaks on keeles võimalus koostada käskude gruppe ehk nn. objekte ning neid korduvalt kasutada. Sel juhul on pildi kirjeldus kujul

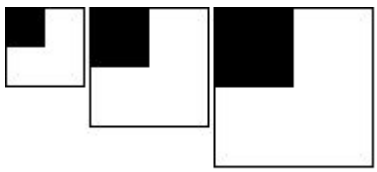
```
defObject objekti_klassi_nimi
vabad käsud
endObject,
```

kus `vabades käsk`udes võib kasutada ka objekti argumente tähistavaid muutujaid ning hiljem saab käsuga

```
object objekti_nimi argumendid
```

kõik objekti lisatud käsud ühe korraga pildi kirjeldusse lisada. Objekti konstrueerimisest on toodud näide 6.

```
new 200,100
defObject kast
  scale %1,%1
  rect 0,0 2 2
  frect 0,0 1 1
endObject
object 0,0 kast 20
object 42,0 kast 30
object 104,0 kast 40
```



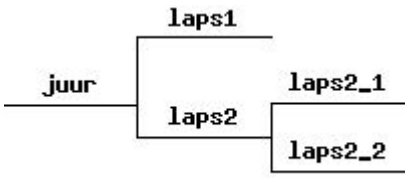
Näide 6: Enda loodud objektide kasutamine keeles SWOG.

4 Inglise keeles tooltip

Näites defineeritakse objekt *kast*, mille joonistamise suurus sõltub 1. argumendist ning seejärel lisatakse pildile kolm defineeritud kujundit.

Konstruktsioon `defObject` on ainus erand, mis ei järgi käsu ühetaolist kuju. Paljude argumentidega käskude puhul oleks sedasi pildi sisestamine muidugi ebamugav, selleks on võetud kasutusele erisümbol '$', mis tähistab, et jooksev rida järgneb järgmisel real. Lisaks võivad failis olla ka kommentaarid, neid määrab rea algusesse paigutatud sümbol '#', erisümbolite kasutamist demonstreerib näide 7.

```
new 200,100

#järgmine käsk on pikk,
#seetõttu poolitame teda
tree 0,0 200 100 ({juur}$
     ({laps1}; {laps2}$
     ({laps2_1}; {laps2_2})))
```



Näide 7: Kommentaaride ning rea poolitamise kasutamine keeles SWOG.

Keele kasutamiseks on vaja tunda ka argumentide esitamise kuju. Üldjuhul on käsu argumendid tühikutega eraldatud koordinaadid, kujundi mõõtude väljendajad, arvulised muutujad või sõned. Piltide kirjeldamise kiiruse suurendamiseks on koordinaatide esitamiseks pakutud 3 erinevat varianti:

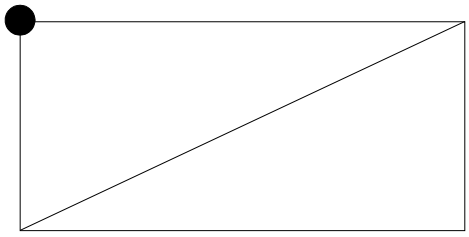
1. `X,Y` kuju – taoliselt antakse koordinaadid numbritega, esimesena on antud abstsiss ning teisena ordinaat. Punkt, mis sellele kirjeldusele ekraanil vastab, sõltub parajasti kehtivast koordinaatide süsteemist. Vaikimisi jooksevad koordinaadid vasakult paremale ning ülevalt alla ja ühele ühikule vastab üks piksel, kuid neid seadeid saab käskudega `origin`, `coordsys`, `scale` ning `rotate` muuta.

2. `(nimi)` kuju – nõnda antakse koordinaadiks nimega viidatud objekti koordinaadid.

3. (nimi.punkt) kuju – antud viisil viidatakse objekti teatud punktile. Näiteks ristküliku puhul on lubatud punktid `p1, p2, p3, p4` ning `center` (tipud ja keskpunkt). Samas võib punkt viidata alamobjektile või olla kujul `alamobjekti_nimi.punkt1`, sama reegel kehtib rekursiivselt ka `punkt1` kohta. Näiteks kasutaja defineeritud objekti *kast* kuuluva ristküliku *ristkylik* keskpunktile saab viidata kasutades kontruktsiooni `(kast.ristkylik.center)`.

Koordinaatide erinevate kujude kasutamist demonstreerib näide 8.

```
new 200,100

#X,Y kuju
rect 10,10 150 70 :kast


#(nimi) kuju
fcircle (kast) 5

#(nimi.punkt) kuju
line (kast.p2) (kast.p4)
```



Näide 8: Koordinaatide esitus keeles SWOG.

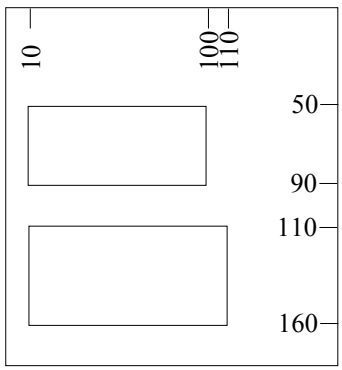
Veel on oluline tunda kujundi mõõte määravate argumentide kuju, nendega määratakse näiteks ristkülikute otspunktid. Väljendamiseks on kaks võimalust:

1. `koordinaat1 koordinaat2` kuju – sel juhul määratakse ristkülik kahe punktiga, kusjuures koordinaadid on juba ülaltutvustatud kujul.

2. `koordinaat laius pikkus` kuju – nõnda saab anda risküliku punkti ja arvulised väärtused, mis tuleb punkti abstsissile ning ordinaadile liita.

Kujundite mõõtude määramist illustreerib näide 9.

```
new 160,180

#pilti ümbritsev kast:
rect 0,0 160 180

#koordinaat1 koordinaat2 kuju:
rect 10,50 100,90

#koordinaat laius pikkus kuju:
rect 10,110 100 50
```



Näide 9: Kujundi mõõtude esitus keeles SWOG. Joonisel on toodud ka pildi koordinaadid.

Vaatleme ka teiste visualiseerimisvahendite võrdlemisel kasutatud näo pildi (joonis 1 lk. 6) kirjeldamist keeles SWOG (näide 10).

```
new 256,256
circle 150,150 100 :nagu
arc (nagu.center) 80 80 10 170
origin (nagu.center)
fellipse -40,-40 10 15 blue
fellipse 40,-40 10 15 blue
```

Näide 10: Näo kirjeldus keeles SWOG.

Näo joonistamisel lisati esiteks pildile ring nimega *nagu*. Järgmisena lisati kaar, mille keskpunkt on sama varem lisatud ringiga. Seejärel oleks vaja paigutada joonisele silmad, et aga nende täpset asukohta absoluutsetes koordinaatide on suhteliselt ebamugav välja arvutama hakata, siis muudetakse koordinaatide süsteemi ning määratakse uueks nullpunktiks näo keskpunkt. Järgnevalt on juba silmade lisamine lihtne, näiteks vasak silm lisatakse koordinaatidega (-40,-40), mis tänu eelnevale käsule `origin` on nüüd absoluutsetes koordinaatides $(150-40,150-40)=(110,110)$.

1.3.2. Käsud

Järgnevalt on toodud realiseeritud käskude lühiülevaade, täpse käskude süntaksi leiab trükitud kujul töö lisast 1 leheküljelt 48 ning koos näidetega internetist aadressilt http://kotkas.ebc.ee/u/hansen/docsource/doc.html.

Keele käsud võib tinglikult jagada sõltuvalt nende funktsioonidest kaheksasse gruppi:

1. Väljundi formaadi seadmine

   `export, new`

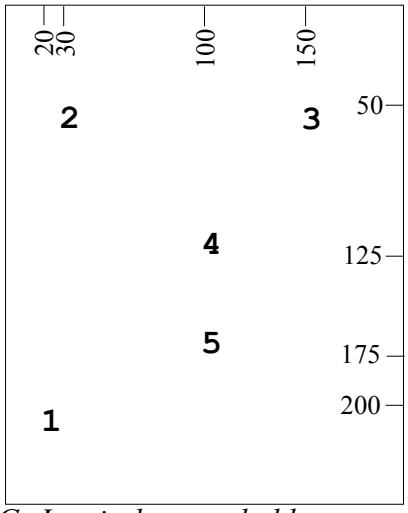
   Käsk `new` määrab pildi suuruse ning käsku `export` kasutatakse animatsioonide loomisel järjekordse kaadri väljastamiseks.

2. Koordinaatide süsteemi seadmine

   `coordsys, origin, rotate, scale`

   Nagu juba mainitud kehtib vaikimisi süsteem, kus nullpunkt on üleval vasakus nurgas ning teljed jooksevad paremale ja alla, nende käskudega saab neid seadeid muuta. Koordinaatide süsteemi transformeerimisest on toodud näide 11.

```
new 200,250
rect 0,0 200 250
string 20,200 {1}
#keskpunkt 30 üh. par. ja 50 alla
origin 30,50
string 0,0 {2}
#keskpunkt 120 üh. paremale
origin 120,0
string 0,0 {3}
#nullpukti keskele,
#teljed paremale ja alla
coordsys center right up
string 0,0 {4}
string 0,-50 {5}
```



Näide 11: Koordinaatide süsteemi muutmine keeles SWOG. Joonisel on toodud ka esialgsed koordinaadid.

3. Kujundite joonistamine

   `(f)arc, (f)circle, connect, (f)ellipse, fill, fillToBorder, image, line, (f)poly, (f)rect, string`

   Käsu nime ees (f), tähistab, et realiseeritud on ka täidetud kujundi joonistamse käsk. Näiteks käsk `rect` on ristküliku kontuuri ja `frect` täidetud ristküliku joonistamiseks.

4. Keerukamate kujundite väljastamine

   `drawAxis, lineGraph, object, tree`

   Käsk `object` väljastab kasutaja loodud kujundi, `tree` lihtsa puu, `drawAxis` koordinaatide telje ning `lineGraph` on kaetud väärtuste vahemike esitamiseks.

5. Abiobjektide defineerimine

   `axis, color, defObject, font, point, pointAdd`

   Selle grupi käskudega defineeritakse objektid, mida kasutatakse kujundite joonistamise käskude argumentides.

6. Joonistamise seadete määramine

   `setColor, setFont, setLineThickness, undo`

   Need käsud muudavad järgnevate joonistamise käskude vaikimisi parameetrite väärtusi.

7. Pildil olevate objektide muutmine

   `alter, moveAfter, moveBefore, newSECorner`

   Siia kuuluvad käsud, millega saab eelnevas pildi kirjelduses lisatud objektide omadusi muuta.

8. Pildi väljastamine

   `outputRawGif`, `outputRawGifAnim`, `outputRawHtml`, `outputRawPng`, `outputRawSVG`, `toGif`, `toGifAnim`, `toHtml`, `toPng`, `toSVG`

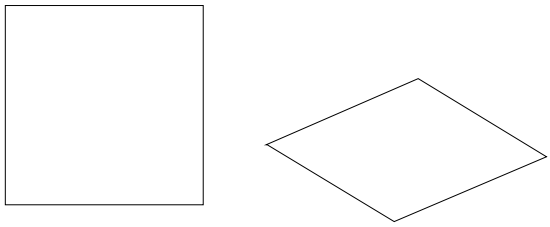
   Pildi kirjelduses saab määrata ka jooksvalt pildi väljastamise. Siiski see ei ole kohustuslik, pildi väljastamist saab juhtida ka keele interpretaatori poole pöördudes.

Peatükk 2
Realisatsioon

Et vähendada tööks kuluvat aega on keele interpretaatori realiseerimisel eesmärgiks kasutada võimalikult palju valmis teeke. Paraku pole aga võimalik piirduda ainult joonistuskäskude joonistusteegile edastamisega, sest ülesande püstitus nõuab ka antud punkti sisaldavate kujundite leidmist ning HTML kujul klikkimise kaardi väljastamist. Seega tuleb realiseerida pildi lugemine nii, et interpretaator teaks pildile lisatavate kujundite täpseid koordinaate ekraanil ning alles seejärel saab joonistamiseks kasutada teiste teekide võimalusi.

Peamine realiseerimise keerukus tuleneb koordinaatide süsteemi transformeerimise võimaldamisest, kujundi omaduste muutumise kohta on toodud näide 12.

```
new 400,300

#vaikimisi tähistab järgnev
#käsk ruutu
rect 0,0 100 100

#koordinaatide süsteemi
#nihutatakse..
origin 130,70
#skaleeritakse..
scale 1,0.5
#ning keerutatakse..
rotate 0,0 40

#sellises seisus väljastab järgnev käsk hoopis rööpküliku
rect 0,0 100 100
```



Näide 12: Käsk „rect 0,0 100 100“ *võib nõuda nii ruudu kui rööpküliku väljastamist.*

2.1. Koordinaatide süsteem

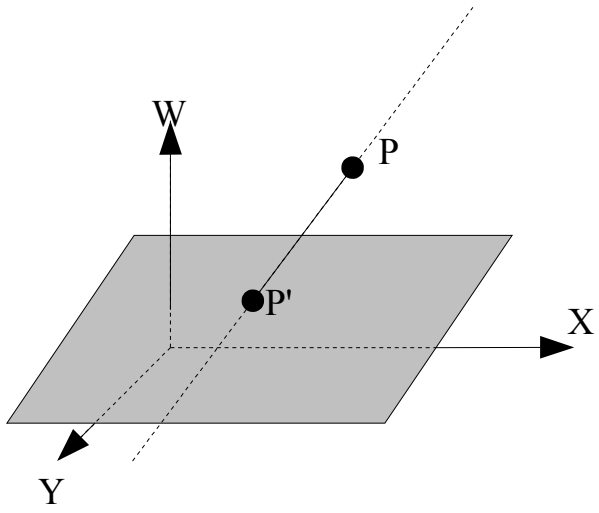
Pildi koordinaatide süsteemi manipuleerimine on realiseeritud kasutades afiinseid transformatsiooni maatrikseid. Järgnevates alamjaotistes tutvustataksegi nende maatriksite sisu ning esitatakse ka probleeme, mis esinesid nende kasutamisel.

2.1.1. Afiinne transformatsiooni maatriks

Eesmärgiks on, et koordinaatide teisendusi saaks esitada ühesel kujul, et igale koordinaatide süsteemile vastaks teatud maatriks ning selle maatriksi teatud kujul maatriksiga korrutamine vastaks kindlat liiki koordinaatide teisendusele. Nii on iga koordinaadistiku teisendus ühtsel kujul säilitatav ning teostatav.

Eesmärgi täitmiseks on kasutusele võetud homogeensed koordinaadid, kus kahemõõtmelisele punktile lisatakse kolmas koordinaat, ehk numbripaari (x, y) asemel hakkab punkti esitama kolmik (x, y, W). Lisaks öeldakse, et kaks homogeenset koordinaatide kolmikut (x, y, W) ja (x', y', W') esitavad sama punkti siis ja ainult siis, kui esimene on teise kordne. Veel nõutakse, et vähemalt üks koordinaatidest oleks nullist erinev, ehk punkti (0, 0, 0) ei defineerita. Punkte kujul, kus W=0 nimetatakse lõpmatuses olevateks punktideks, need jäetakse antud lähenemisel vaatlusest välja.

Homogeensed kordinaadid seotakse tavaliste kahemõõtmelistega kasutades järgnevat: kui võtta kõik kolmikud, mis esitavad sama punkti ning $W\neq0$ (ehk punktid kujul $(t\cdot x, t\cdot y, t\cdot W)$, $t\neq0$), siis neid kolmemõõtmelise ruumi koordinaatidena vaadeldes moodustavad nad ruumis sirge. Kui aga kõik homogeensed punktid (x, y, W), kus $W\neq0$ homogeniseerida (jagada koordinaadid läbi parameetriga W), siis moodustavad nad tasandi, seda illustreerib joonis 3. Seega võib kahe koordinaadiga antud punktid (x, y) esitada homogeensetena kujul (x, y, 1) ning homogeensele punktile (x, y, W) vastavusse seada punkti $\left(\frac{x}{W}, \frac{y}{W}\right)$.



Joonis 3: XYZ koordinaatide ruum W=1 tasandi ja homogeensete koordinaatidega punktiga P(X, Y, W), mis on projekteeritud tasandile W=1 kahemõõtmeliseks punktiks P'

Homogeensete koordinaatide korral on koordinaatide süsteemi teisendused esitatavad teatud maatriksite korrutistena, punkti (x, y) nihutamise (d_x, d_y) ühiku võrra, kus uueks punkti koordinaadiks saab $(x+d_x, y+d_y)$ teeb maatriksite korrutis

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}.$$

Ka punkti (x, y) läbi korrutamise teguritega (s_x, s_y), kus uueks koordinaadiks saab $(x \cdot s_x, y \cdot s_y)$ teostab maatriksite korrutis, sedakorda on selle kujuks

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}.$$

Samuti on punkti (x, y) keerutamine ümber punkti $(0,0)$ nurga θ võrra, mille puhul punkti uueks koordinaadiks saab $(x \cdot \cos\theta - y \cdot \sin\theta, x \cdot \sin\theta + y \cdot \cos\theta)$, esitatav maatrikskujul:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}.$$

Ülalmainitud kujul teisendusmaatriksitel on omadus, et nende omavahelise suvalises järjekorras korrutamise tulemusena saadakse koordinaatide teisendus, mis säilitab joonte paralleelsuse, küll aga ei pruugi säilitada nende pikkust ja joonte vahelisi nurki. Taolistele maatriksitele vastavaid teisendusi nimetatakse afiinseteks transformatsioonideks.

Samuti on teisendustel selline hea omadus, et korrutamine säilitab ka punktidega järjestikku tehtavate teisenduste sisu. Näiteks kui võtta transformatsioon, mis skaleerib ning teisendus, mis nihutab punkte, siis nendele transformatsioonidele vastavate maatriksite korrutise korrutamine algpunktiga annab sama tulemuse kui esiteks algpunkti korrutamine skaleeriva maatriksiga ning seejärel tulemuse korrutamine nihutajaga. Nõnda saab realiseerida koordinaatide keerutamise punkti $P(x, y)$ suhtes nurga θ võrra. Esiteks tuleb P teisendada nullpunktiks, teostada keere ümber nullpunkti ja seejärel nullpunkt teisendada tagasi punktiks P. Vastav transformatsiooni maatriks on

$$\begin{pmatrix} 1 & 0 & x \\ 0 & 1 & y \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & -x \\ 0 & 1 & -y \\ 0 & 0 & 1 \end{pmatrix}.$$

Tähele tasub panna tegurite järjekorda. Kuna punkti koordinaadid on transformeerimisel maatriksite korrutises teiseks teguriks, siis peavad teisendused olema järjestatud just „tagurpidi“ [FDFH90].

2.1.2. Koordinaatide teisendamine

Keele interpretaatori realiseerimisel kerkis esile ülesanne, et teades punkti koordinaate ekraanil tuleb leida koordinaadid, millele jooksvat transformatsiooni rakendades saab ette-antud koordinaadid. Ehk tuleb lahendada võrrand

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} m_0 & m_1 & m_2 \\ m_3 & m_4 & m_5 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix},$$

kus tundmatuteks on x ja y. Üldjuhul oleks tegu suhteliselt ebameeldiva ülesandega, kus lahendite olemasolu pole garanteeritud. Antud juhul on siiski teada, et lahend eksisteerib. Afiinse transformatsiooni maatriksil peab leiduma pöördmaatriks, sest igal teisendusel on pöördteisendus.

On võimalik tõestada, et maatriksil $A \in Mat_n(K)$ on pöördmaatriks olemas siis ja ainult siis, kui maatriks A on regulaarne. Lisaks kui $A = (a_{ij}) \in Mat_n(K)$ on regulaarne, siis

$$A^{-1} = \frac{1}{det(A)} \cdot \begin{pmatrix} A_{11} & A_{21} & \dots & A_{n1} \\ A_{12} & A_{22} & \dots & A_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ A_1 n & A_2 n & \dots & A_{nn} \end{pmatrix},$$

kus A_{ij} on elemendi a_{ij} algebraline täiend [Kil98]. Seega võib mainitud võrrandi vasakult läbi korrutada transformatsiooni maatriksi pöördmaatriksiga

$$\begin{pmatrix} m_0 & m_1 & m_2 \\ m_3 & m_4 & m_5 \\ 0 & 0 & 1 \end{pmatrix}^{-1} \cdot \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} m_0 & m_1 & m_2 \\ m_3 & m_4 & m_5 \\ 0 & 0 & 1 \end{pmatrix}^{-1} \cdot \begin{pmatrix} m_0 & m_1 & m_2 \\ m_3 & m_4 & m_5 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}.$$

Kasutades pöördmaatriksi leidmise valemit saame, et

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \dfrac{m_4 \cdot x - m_1 \cdot y + m_1 \cdot m_5 - m_2 \cdot m_4}{m_0 \cdot m_4 - m_3 \cdot m_1} \\ \dfrac{-(m_3 \cdot x - m_0 \cdot y + m_0 \cdot m_5 - m_2 \cdot m_3)}{m_0 \cdot m_4 - m_3 \cdot m_1} \\ 1 \end{pmatrix}.$$

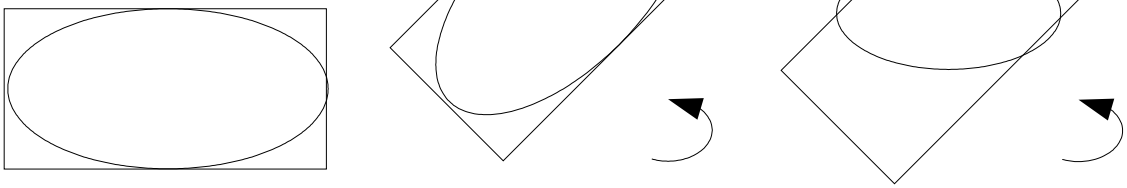
2.1.3. Probleemid ja lihtsustused

Rasterpiltide joonistamiseks kasutatakse GD teeki, paraku aga kõigi kujundite igasuguseid transformatsioone GD abil lihtsate vahenditega joonistada ei ole võimalik. Seetõttu tuli teha kompromisse ideaalsuse ning mõistlikkuse piiridesse jäämise vahel. Probleemid tulevad esile siis, kui koordinaatite süsteemi keerutama hakata. Järgnevalt esinenud probleemid kujundite kaupa:

1. Teegis on realiseeritud ainult koordinaatteljestikuga paralleelsete telgedega ellipsi joonistamine, mitteparalleelne juht on aga realiseerimata. Probleem on lahendatud nii, et ka keeratud kujund joonistatakse tavalise ellipsina, kusjuures see püütakse mahutada nõutud nelinurga sisse. Tekkivat pilti illustreerib näide 13. Antud lihtsustuse tulemusena ei anna küll keerutatud pildil ellipsite ning kaarte joonistamine õiget tulemust, kuid arvestades, et tegu on vahendiga lihtsate piltide joonistamiseks, võib loota, et reaalsetes rakendustes antud probleem esile ei kerki.

```
defObject naide
  rect 0,0 200 100
  ellipse 100,50 100 50
endObject

object 0,0 naide
```

```
rotate 100,100 45
object 100,100 naide
```



Näide 13: Ellipsi keerutamine 45 kraadi, vasakul pilt enne keerutamist, keskel korrektne keere ning paremal pilt keele SWOG realisatsioonist.

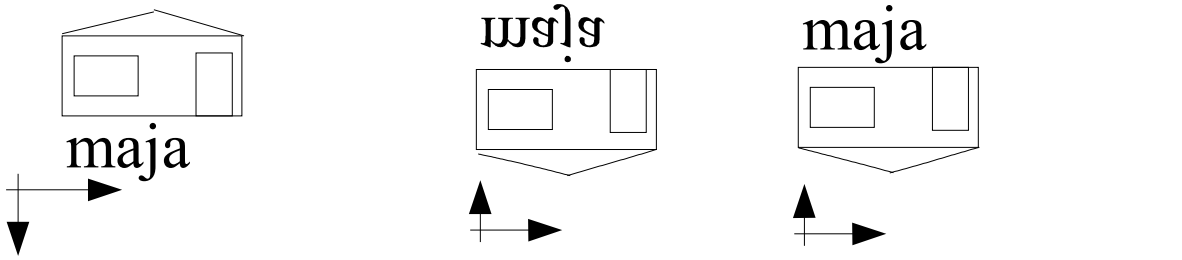
2. Kui kasutada teksti väljastamiseks ainult GD teegi pakutavaid fonte, ehk viite erinevat tähesuurust, siis sellist teksti on võimalik korrektselt joonistada ainult koordinaatide kas 0- või 90-kraadise vastupäevase pöörde puhul. Teiste pöörete korral on SWOG realiseeritud nii, et tekst joonistatakse kasutades 0-kraadist pööret. Oleks võimalik kasutada teksti abipuhvrisse joonistamist ning puhvri sisu sobiva keerdega pildile lisamist, kuid selline lähenemine ei tundu töökiiruse seisukohast lähtudes mõistlik.

   Kui kasutajal on vaja, et pildi keerutamisel ka tekstid kaldega joonistataks, siis tuleks kasutata *TrueType* tüüpi fonte. Kuid ka nende puhul ei võimalda GD teek lihtsate vahenditega realiseerida kõigi transformatsioonide korrektselt joonistamist. Nimelt ei ole võimalik joonistada tekste peegelpildis, seetõttu ei tule pilt korrektne, kui teisendus

korrutab koordinaadid läbi negatiivse arvuga. Selline juht on lahendatud nii, et tekst joonistatakse küll talle ettenähtud nelinurka, kuid mitte peegeldatult, tulemust illustreerib näide 14. Ka see lihtsustus ei tekita ilmselt kasutajatele probleeme, pigem isegi väldib neid, sest loetavuse huvides on selgelt parem kui teksti ei hakata peegeldama.

```
new 400,300
font {fonts/ARIAL.TTF} 30 :arial
setFont arial

defObject maja
  #katus
  line 50,0 0,10
  line 50,0 100,10
  rect 0,10 100 50 :karp
  rect 8,20 40 20 :aken
  rect 75,20 20 40 :uks
  string (%this.karp.p4) {maja}
endObject

object 0,0 maja
```

```
scale 1,-1
object 200,-100 maja
```



Näide 14: Teksti peegelpildis joonistamine. Vasakul on joonistus vaikimisi koordinaadistikus, keskel korrektne pilt koordinaatide süsteemis, kus on muudetud y-telje suund vastupidiseks, paremal aga muudetud teljestikuga pilt keele SWOG realisatsioonis.

2.2. Punkti järgi kujundi tuvastus

Etteantud punkti alla jäävate kujundite leidmise ülesanne on taandatud teatud lihtsate kujundite uurimisele, nende tüüpideks on: ristkülik, ring, ellips, sektor ning hulknurk. On võimalik keelt laiendada ning uusi kujundeid lisada, kuid hetkel on piirdutud 5 tüüpi pindadega.

Mainimist väärib punkti hulknurka kuuluvuse uurimine, sest ülesanne ei ole nii triviaalne kui teiste kujundite puhul. Algoritmi valikuks vaatleme kahte kandidaati:

1. algoritm: olgu antud hulknurk, mis on esitatud tippude järjendiga loomulikus ümber hulknurga käimise järjekorras ning punkt *P*, mille puhul tuleb testida, kas see kuulub antud hulknurka. Punkti hulknurka kuulumise kontrollimiseks leitakse hulknurga alumine vasakpoolseim tipp ning võetakse jooksvaks kontrollitavaks punktiks *J* sellele eelnev punkt.

Algoritm läbib kõik punktid alustades alumisest vasakpoolsest ja lõpetades sellele eelnevas tipus. Iga läbitava punkti *K* puhul toimitakse järgnevalt: kui punktist *P* paremale suunduv kiir *k* lõikub lõiguga [*J*, *K*], siis suurendatakse loendurit ühe võrra ning kui *k* ei läbi punkti *K*, siis väärtustatakse jooksev punkt *J* punktiga *K*. Peale kõigi punktide läbimist leitakse loenduri sisu põhjal tulemus – kui selleks on paarisarv, siis punkt ei kuulu hulknurka, vastasel juhul kuulub. Algoritm on esitatud ka pseudokoodis [Kih97].

Antud: Punkt *P*, hulknurk esitatuna tippude järjendina *Tipud* loomulikus ümber hulknurga käimise järjekorras.
Leida: Kas punkt *P* kuulub hulknurka.
01: leida alumine vasakpoolseim tipp *S* ja sellele eelnev tipp *E*
02: *J* := tipule *S* eelnev tipp
03: *loe* := 0
04: **for** *K* := *S* **to** *E* (läbides *Tipud*)
05: **if** punktist *P* paremale suunduv kiir lõikub lõiguga [*J*, *K*] **then**
06: *loe*++
07: **else**
08: *J* := *K*
09: **next**
10: **if** *loe* on paarisarv **then**
11: **return** *P* kuulub hulknurka
12: **else**
13: **return** *P* ei kuulu hulknurka

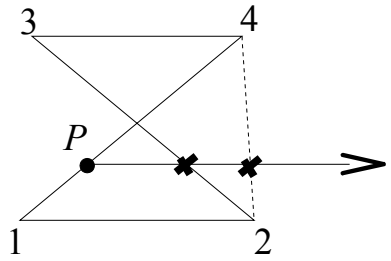
Algoritm 1: Punkti hulknurka kuuluvuse kontroll.

Antud: Punkt *P*, hulknurk esitatuna servade järjendina *Servad*.
Leida: Kas punkt *P* kuulub hulknurka.
01: **foreach** *S*∈*Servad*
02: **if** *P* paikneb serval *S* **then**
03: **return** *P* kuulub hulknurka
04: **next**
05: **while** mõni hulknurga tippudest ühtib punktiga *P*
06: keerutada hulknurga tippe ümber punkti *P*
07: **do**
08: *loe* := 0
09: **foreach** *S*∈*Servad*
10: **if** punktist *P* paremale suunduv kiir lõikub servaga *S* **then**
11: *loe*++
12: **next**
13: **if** *loe* on paarisarv **then**
14: **return** *P* kuulub hulknurka
15: **else**
16: **return** *P* ei kuulu hulknurka

Algoritm 2: Punkti hulknurka kuuluvuse kontroll.

Algoritm 2 on analoogne esimesega, erinevuseks on, et esiteks vaadatakse, kas punkt paikneb mõnel hulknurga servadest. Paiknemise korral tagastatakse, et punkt kuulub hulknurka. Vastasel juhul jätkatakse: seni kui kiir *k* läbib mõnda hulknurga tippu keerutatakse selle tippe ümber punkti *P*. Seejärel vaadeldakse kiire *k* lõikumist kõigi hulknurga servadega. Kui lõikumiskordade summa on paarisarv, siis punkt *P* ei kuulu hulknurka, vastasel juhul kuulub [Jen95].

On selge, et kui hulknurga servad ei lõiku, siis oleks parem esimene algoritm. Nii tänu kiirusele kui sellele, et teise algorimi puhul on teoreetiline võimalus, et punktide keerutamisel sattutakse lõpmatusse tsüklisse. Sooviga, et ka lõikuvate servadega hulknurga puhul annaks algoritm mõistlikke tulemusi on valitud realisatsiooniks 2. algoritm. 1. algoritm võib sellisel juhul juba väga lihtsate kujundite puhul valetama hakata, üks taoline olukord on kujutatud joonisel 4.



Joonis 4: Esimene algoritm tuvastamas punkti P kuuluvust hulknurka, ristidega on näidatud loendamisel arvesse minevad lõikepunktid.

2.3. Turvalisuse küsimus

Kuna pildi kirjelduses võib kasutada ka failidele viitamist, siis keele SWOG kasutamisel veebirakendustes kerkivad esile teatud turvariskid. Varem valmistatud või programmist koostatud sisendfailide korral ilmselt ei ole ohtu, et sisendisse sattuks failinimesid, mis võivad kasutajat kompromiteerida. Küll aga kerkib risk esile kui veebikasutajale antakse võimalus SWOG käskudele enda parameetreid sisestada. Taoline olukord võiks näiteks esineda, kui rakenduses küsitaks kasutajalt nime ning laisk rakenduse looja ei kontrolliks saadud sisendit vaid lisaks saadud sõne kohe SWOG sisendfaili. Ilmselt on hästi näha, mis oht süsteemi valitseb. Näiteks keeles Perl võiks olla rakenduse koodis:

```
$buf.='string 10,10 {Sinu nimi on:'.$sisend.'}';
SWOG->readObjectFromBuffer($buf);
```

Kui kasutaja sisestaks

```
$sisend='}\n toPng {suvaline faili nimi}',
```

saaks veebiliidese abil kirjutada üle suvalise CGI skriptile kirjutusõigustega liigipääsetava faili.

Taoliste turvaaukude kõrvaldamiseks on võimalus lisada käskude argumentides olevatele failinimedele piiranguid. Selleks kasutatakse seadete faili `SWOGConfig.txt`. Failis kirjeldatakse käskude kaupa ligipääsetavad failide ning kataloogide nimed. Nime kirjelduses saab kasutada ka sümboleid `+`, `*`, `?`, mis tähistavad teatud arvu ladina tähti `[a-zA-Z]`. Samuti on võimalik määrata, millisesse kataloogi SWOG sisendis viidatud failinimi kuulub. Täpsema konfiguratsioonifaili kirjelduse leiab tööle lisatud optiliselt kettalt failist `doc/juhend.html`.

Vaikimisi seadete fail laseb sisse lugeda kõiki faile, kuid väljastamine on lubatud ainult kataloogi `output/`. Kui aga programm seadete faili ei leia, lubatakse ligi kõigile failidele, millele ka Perli skript ligi pääseb.

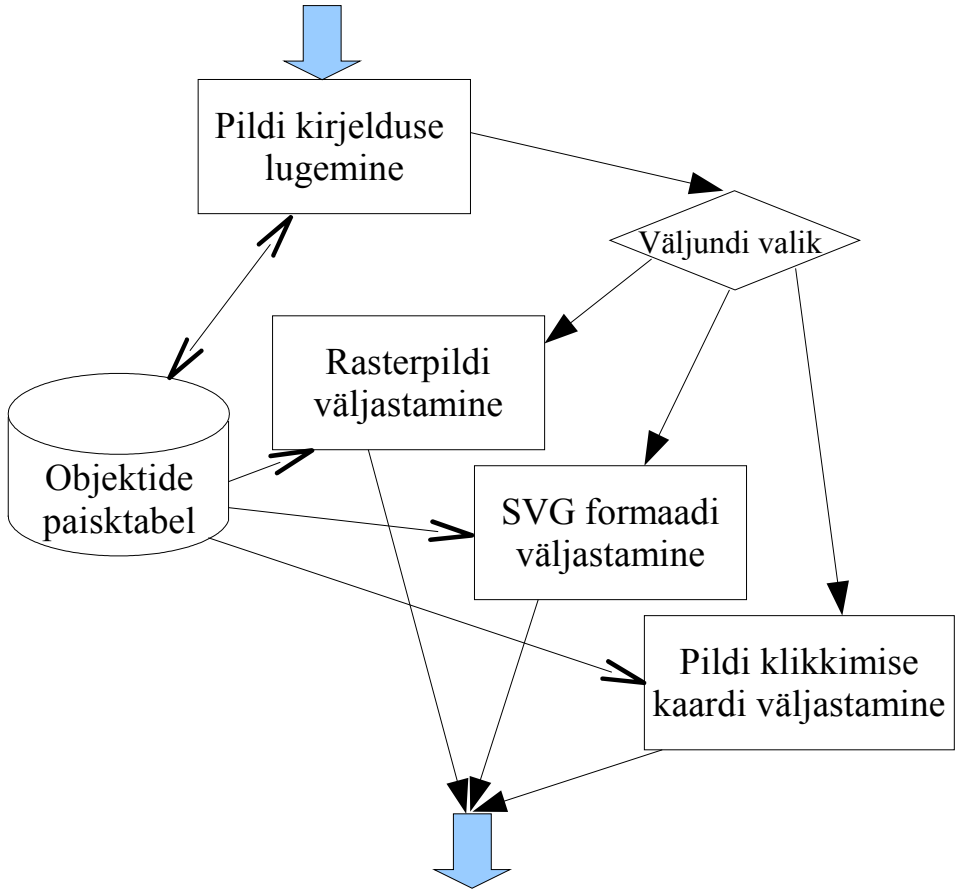
2.4. Väljastamine SVG formaati

Lisaks rasterpildi väljastamisele võib väljundiks olla ka SVG formaadis vektorgraafika pilt. Koos kujunditega väljastatakse ka nendega seotud lingid. SVG võimaldab ka animatsioonide esitust, et aga animatsioonid keeles SWOG on lihtsad kaadrite kaupa piltide kuvamised, siis ei tundunud taoliste animatsioonide SVG formaadis väljastamine mõttekas ning seda ei ole realiseeritud.

2.5. Realisatsioon keeles Perl

Perl on interpreteeritav programmeerimiskeel, mis on disainitud tekstifailidest info lugemiseks ning sisendfailil põhineva teksti formaadis väljundi loomiseks [Hol99]. Kuna realisatsiooni oluliseks osaks on pildi kirjelduse sisse lugemine, siis tänu oma regulaaravaldistele sobib Perl selleks ideaalselt. Et keeles Perl on realiseeritud ka moodul GD teegi kasutamiseks, siis saigi valitud esmaseks keele realisatsiooni vahendiks Perl.

2.5.1. Programmi struktuur



Joonis 5: Programmi lihtsustatud struktuur

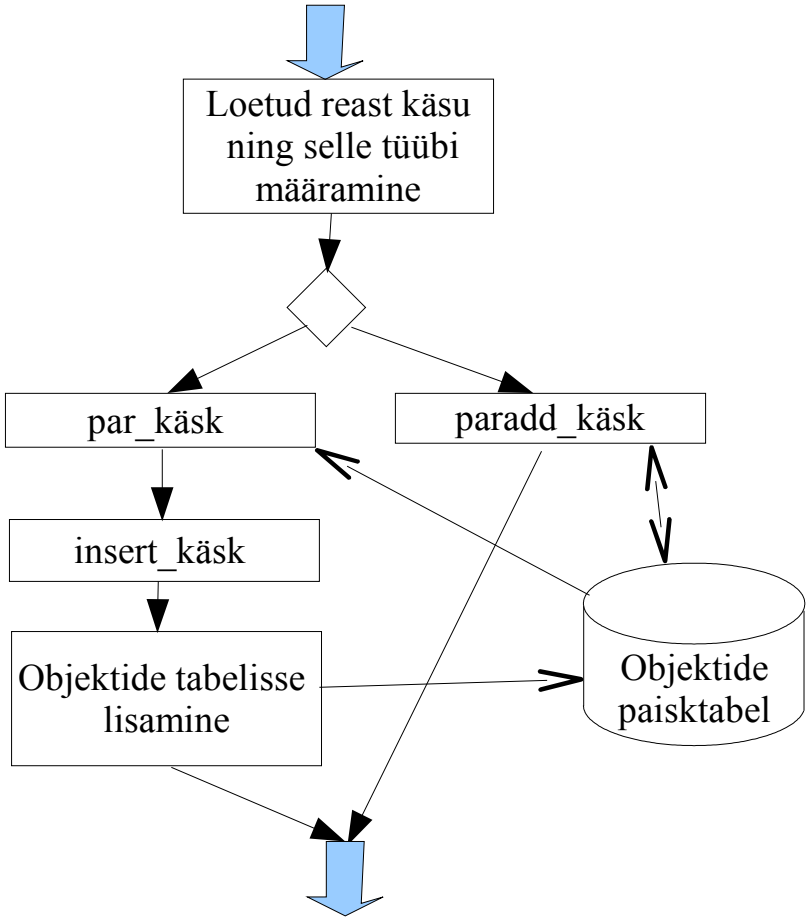
Programm on ülesehitatud taoliselt, et esiteks loetakse pildil olevad kujundid sisemisse struktuuri ning selle sisu kasutatakse hilisemateks tegevusteks, näiteks pildi joonistamiseks. Ilmselt töötaks programm kiiremini, kui pildi lugemise ajal joonistataks rida-realt ka pilt kohe valmis, aga seda lahendust ei luba kasutada keele SWOG võimalused. Takistuseks on, et keeles on võimalik varasematele objektidele viidata ning neid ka muuta.

Programmi üldist struktuuri kirjeldab lihtsustatud protsessidiagramm joonisel 5. Üldjuhul toimib programm nii, et esiteks loetakse pildi kirjeldus sisse ning seejärel väljastatakse see ühte kolmest väljundi liigist. Selline kirjeldus on tugevalt lihtsustatud, sest võib juhtuda, et pilti otseselt ei väljastatagi vaid ainult loetakse pilt sisse ning tuvastatakse etteantud punkti alla jäänud kujundid. Samuti ei pruugi tegevuste järjekord nii jäik olla, programm võib töötada ka nii, et ühel täitmisel antakse mitut liiki väljundit või toimub piltide väljastamine käskude lugemise vahel.

2.5.2. Pildi kirjelduse lugemine

Pilt loetakse sisse ridade kaupa. Rea lugemisel tehtavate tegevuste protsessidiagramm on

toodud joonisel 6. Esiteks määratakse reaga antud käsu nimi ning loetakse käsuga seotud nimi, link ja vaba tekst. Järgnevalt töödeldakse käsu argumente. See võib loetavast käsust sõltuvalt toimuda kahel viisil.



Joonis 6: Pildi kirjelduse ühe rea lugemise protsessidiagramm

Tavapäraselt tehakse lugemist kahes faasis. Esiteks loetakse käsk ja argumendid, lisatakse vaikimisi parameetrid ning teisendatakse sisendiks antud koordinaadid jooksvasse koordinaatide süsteemi. Seda teevad protseduurid, mille nimed algavad prefiksiga *par_*, ning lõpevad käsu nimega (näiteks *par_line*). Selline funktsioon saab sisendiks käsu argumendid ning tagastab väljundiks (*X*, *Y*, *data*), kus (*X*, *Y*) on objekti nn. baaspunkt jooksvas koordinaatide süsteemis ja *data* sisaldab objekti spetsiifilisi parameetreid. Näiteks joone puhul on *data* sisuks ($x_{2,}y_{2,}$ *color*), kus punkt $(x_{2,}y_{2})$ määrab joone lõpp-punkti ja *color* joone värvi.

Seejärel kutsutakse välja protseduur prefiksiga *insert_*, mis saab sisendiks *par_*-funktsiooni väljundi ning tagastab objekti kirjelduse. Tagastatav kirjeldus on paisktabeli kujul, mille levinumad võtmed on toodud tabelis 1.

Väli	Sisu kirjeldus
X *Y*	Kujundi baaspunkti koordinaadid selle lisamise ajal kehtinud koordinaatide süsteemis.
data	Paisktabel kujundi spetsiifilistest väljadest..
regionType *region*	Seatud kujunditel, mis omavad pinnalaotust ja millel on võimalik klikkida. Kasutatud kujundite piirkondade tüübid on *rect* – täisnurkne ristkülik, *circle* – ring, *ellipse* – ellips, *arc* – sektor, *poly* – hulknurk. Massiiv *region* sisaldab kujundit defineerivaid punkte
type	Kujundi tüüp, näiteks *line*, *frect*, *rect*, *image*...
affine	Viide kujundi lisamise ajal kehtinud koordinaatide transformatsiooni maatriksile.
objectPart	Kui seatud, siis määrab, millisesse objekti antud kujund kuulub.
link	Kujundiga seotud vabas vormis link.
alt	Kujundiga seotud vabas vormis tekst.
objects	Kui kujund omab alamobjekte, siis on tegu paisktabeliga, mille võtmeteks on alamobjektide nimed objekti nimeruumis ja sisuks kujundite nimed globaalses nimeruumis.

Tabel 1: Objekti kirjelduse väljad

Üldjuhul arvutab *insert_*-funktsioon väljad *X*, *Y*, *data*, *regionType* ning *region*. Lisaks võib osutuda tarvilikuks ka kujundi tüübi muutmine, näiteks kui jooksev koordinaatide süsteem on mingi punkti suhtes keeratud, siis muutub tüüp *rect* (täisnurkne ristkülik) tüübiks *poly* (hulknurk).

Peale *insert_*-funktsiooni tööd lisatakse kujund paisktabelisse ja ka massiivi *obj_order*, mis määrab kujundite väljundisse lisamise järjekorra.

Käsud, mis ei lisa joonistatavaid kujundeid, vaid muudavad joonistamise seadeid (näiteks vaikimisi värvi muutja `setColor`) läbivad sama *par_*-, *insert_*-funktsioonide jada ning lisatakse paisktabelisse. Kuna neil ei määrata *insert_*-funktsiooni väljundis *data* ega *objects* sisu, siis ei lisata neid massiivi *obj_order.*

Käsu lugemiseks on kasutuses ka teine lähenemine, nimelt on keerukamate kujundite jaoks funktsioonid prefiksiga *paradd_*. Taolised on näiteks puud (`tree`), joonte diagramm (`lineGraph`) ning ka failis defineeritud objekti lisamise käsk (`object`). Antud käskude

puhul *par_*- ja *insert_*-funktsioone kasutav lähenemine ei ole sobiv, sest nende argumentide formaalsel kujul esitus ei oleks nii lihtne kui primitiivsetel kujunditel. Samuti on nende eripäraks, et üldjuhul lisatakse ühe sisendrea põhjal kujundite paisktabelisse rohkem kui üks kujund.

2.5.3. Pildi väljastamine

Kui kujundid on paisktabelisse lisatud, siis edasised tegevused on juba suhteliselt lihtsad, sest järgnevalt saab pildile läheneda üksikute lihtsate kujunditega tegeledes. Kui näiteks on soovitud pilt rasterpildina väljastada, siis läbitakse kujundite paisktabel massiivi *obj_order* salvestatud järjekorras ning kutsutakse iga kujundi jaoks välja protseduur prefiksiga *gd_*. Vastava protseduuri sisuks on nõutud kujundi pildile joonistamine.

Täpselt analoogselt toimub ka SVG formaadis pildi väljastamine, sel juhul kasutatakse üksikute kujundite XML faili lisamiseks protseduure, mille nimi algab sümbolitega *svg_*. Sarnaselt toimitakse ka pildi klikkimise kaardi väljastamisel.

Kui täpne olla, siis päris rangelt üksikute kujunditega tegelemisega eelmises lõigus märgitud protseduurid ei pääse. Nimelt lisandub neile ülesanne, et kujundite grupile ehk keele SWOG mõistes objektile antud link tuleb seostada kõigi grupi liikmetega. Võib küll tunduda, et lihtsam oleks objektide lingid kopeerida alamobjektidele juba pildi sisse lugemise käigus, kuid sel juhul oleks olnud keeruline realiseerida hilisemat peaobjekti lingi muutmisega kaasnevat alamobjektide lingi uuendamist. Et paisktabelis on igal alamobjektil ka viide oma ülemusele, siis on õigete linkide leidmine suhteliselt triviaalne ülesanne.

Enne pildi välja joonistamist oleks põhimõtteliselt võimalik kujundite hulka ka optimeerida, et eirata teiste taha jäävaid kujundeid. Kuid seda pole realiseeritud, sest eeldatavalt võtaks kujundi eemaldamise lubatavuse arvutamine rohkem aega kui selle üleliigne joonistamine.

2.5.4. Keele laiendamise võimalused

Programmi struktuur võimaldab keelde lihtsalt uusi käske lisada. Kui vaja on lisada primitiivsetest kujunditest koosnevat objekti, siis tuleb kirjutada *paradd_*-funktsioon. Kui aga soovitakse lisatavat objekti ise joonistada, siis tuleks realiseerida käsu jaoks *par_*-, *insert_*- ning *gd_* ja *svg_* funktsioonid.

2.5.5. Programmi liides

Programmi kasutamiseks on vaja Perli moodulit `SWOG.pm`. Keele interpretaatoriga

suhtlemiseks on mitmed protseduurid, pildi kirjelduse lugemiseks on näiteks funktsioonid `readObjects` ja `readObjectsFromBuffer`. Esimene neist loeb pildi kirjelduse failist ja teine argumendiks antud puhvrist.

Järgnevalt näide, kuidas SWOG faili pildina kuvada, näiteks on toodud CGI skript ning SWOG formaadis pildi kirjedus:

Fail **kuva.cgi:**

```
#!/usr/bin/perl -w
#
# kuvab pildi 'pilt.swog', eeldus, et
# selle faili viimane käsk on outputRawPng

# lisame kasutusse SWOG.pm mooduli, vajalik on, et fail
# SWOG.pm oleks kataloogis, mis sisaldub Perli massiivis @INC
use SWOG;

# ütleme brauserile, et väljund PNG formaadis pilt
print "Content-type: image/png\n\n";

# käseme SWOGil pildi sisse lugeda ning reageerida käskudele:
SWOG::readObjects('pilt.swog');
```

Fail **pilt.swog:**

```
new 150,50
string 10,10 red {Tervitus SWOGILT!}
#järgnev käsk väljastab pildi std. väljundisse:
outputRawPng
```

Serveripoolse punkti järgi kujundite tuvastamiseks läheb vaja ka liidese teisi funktsioone. Lisaks kujundi tuvastusele on Perlis realiseeritud turvalisuse ning mugavuse nimel ka funktsioonid, mis väljastavad pilti soovitud formaati, näiteks `file_png` jt. Kõik funktsioonid ning nende selgitused on toodud lisas 2 leheküljel 56 ja lisas 4 optilisel kettal failis `/doc/perl.html`.

2.6. Interpretaatori kasutamine käsurealt

Keele interpretaatorit saab kasutada ka käsurealt. Skript on kirjutatud keeles Perl. Programmi tööd võib juhtida nii käsurea argumentidega kui käskude interaktiivselt sisestamisega. Skripti SWOG.pl kasutamiseks saab abi käivitamisel käsureale `--help` võtme lisamisel. Seejärel näidatakse inglise keelset abiinfot, kuvatav tekst on toodud ka tööle lisatud kettal failis `/doc/juhend.html`. Faili `pilt.swog` sisu väljastamiseks faili `pilt.png` tuleks näiteks sisestada

```
perl SWOG.pl --input=pilt.swog --toPng=pilt
```

2.7. Realisatsioon keeles C

Kuna Perl on interpreteeriv keel, siis keele SWOG interpretaatori töökiirusele on võimalik lisa saada kirjutades programm ümber mõnda kompileeritavasse keelde. Kuna GD teek on kirjutatud keeles C, siis on loogiline jätk ka SWOG sellesse keelde ümber kirjutada. Samas peab nentima, et see teisendamine on üsna mahukas töö, seetõttu on antud töö raames eesmärgiks uurida, kas programmi C keelde ümber kirjutamisest võiks saada olulist kiiruse võitu. Eesmärgiks on seatud realiseerida pildi sisse lugemine ning GD teegi abil väljastamine ristkülikute joonistamise ning objektide defineerimise käskude jaoks. Samas tuleb muidugi programmi struktuur üles ehitada nii, et ka teiste käskude lisamine oleks lihtne.

2.7.1. Koodi keelest Perl C keelde teisendamine

Kuna programm on juba korra realiseeritud, siis ilmselt lihtsaim on võtta aluseks Perl keelse programmi ülesehitus ning Perl kood tõlkida keelde C. Et aga Perl pakub programmeerijale veidi rohkem valmis vahendeid, siis tuleb leida võimalused nende vahendite realiseerimiseks.

Üks Perli visiitkaartidest on korralik regulaaravaldiste tugi, mida keeles C otseselt pole. Ilmselt kõige kiiremini töötava käskude lugeja saaks, kui kirjutada ise spetsiaalne keele süntaksist lähtuv teksti tuvastaja. Kuid see ei tundu mõistlik, selle asemel on regulaaravaldistega tegelemiseks kasutatud PCRE teeki, mille lühend tuleb sõnadedest *Perl Compatible Regular Expressions*, ehk tegu on teegiga, kus on realiseeritud Perli süntaksile vastavad regulaaravaldised. Nüanssides see realisatsioon veidi erineb Perli regulaaravaldiste käsitlusest [url:Haz04a], aga need erinevused ilmselt ei mõjuta keele SWOG realisatsiooni. Samuti on selle teegi eeliseks, et see on lisaks Unixile kompileeritud ka operatsioonisüsteemi Windows teegiks.

Veel kasutab Perli realisatsioon laialdaselt paisktabeleid, mida keeles C samuti kandikul ei pakuta. Kuid loomulikult pakutakse kasutamiseks rohkelt valmis realisatsioone. Antud realisatsioonis on kasutatud paistabeli võtme genereerimiseks funktsiooni mida tuntakse nime all *ElfHash*. Funktsioon on nime saanud sellest, et see on kasutuses Unixi ELF[5] formaadis failides [Bin96].

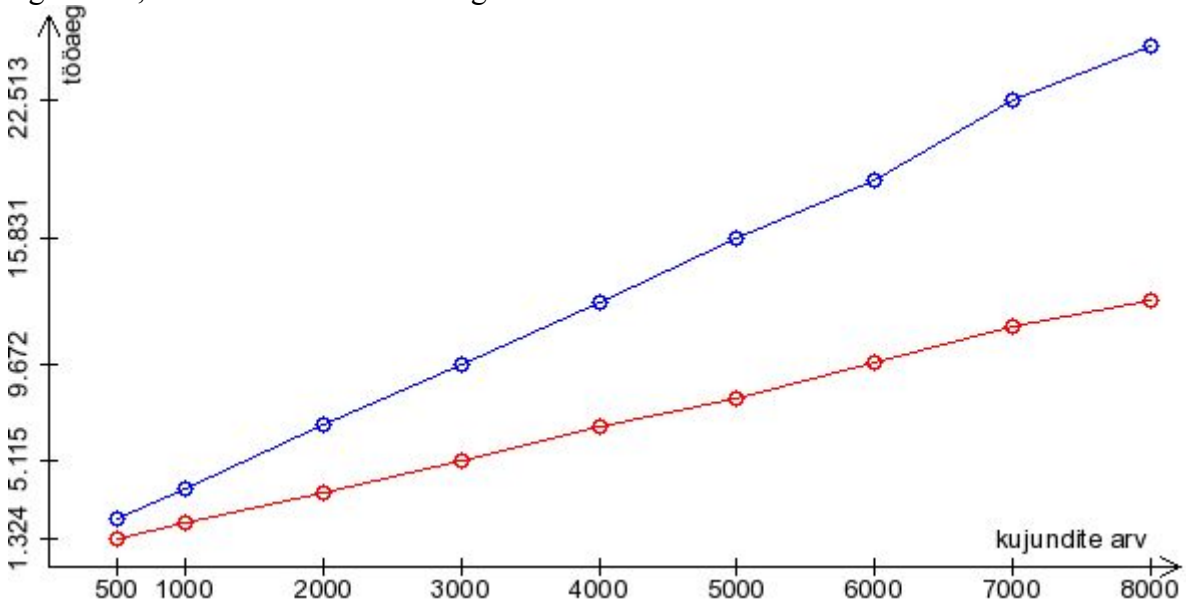
2.8. Kiiruste võrdlus

Graafikakeel SWOG on realiseeritud keeles Perl täies ulatuses ning eksperimendi korras

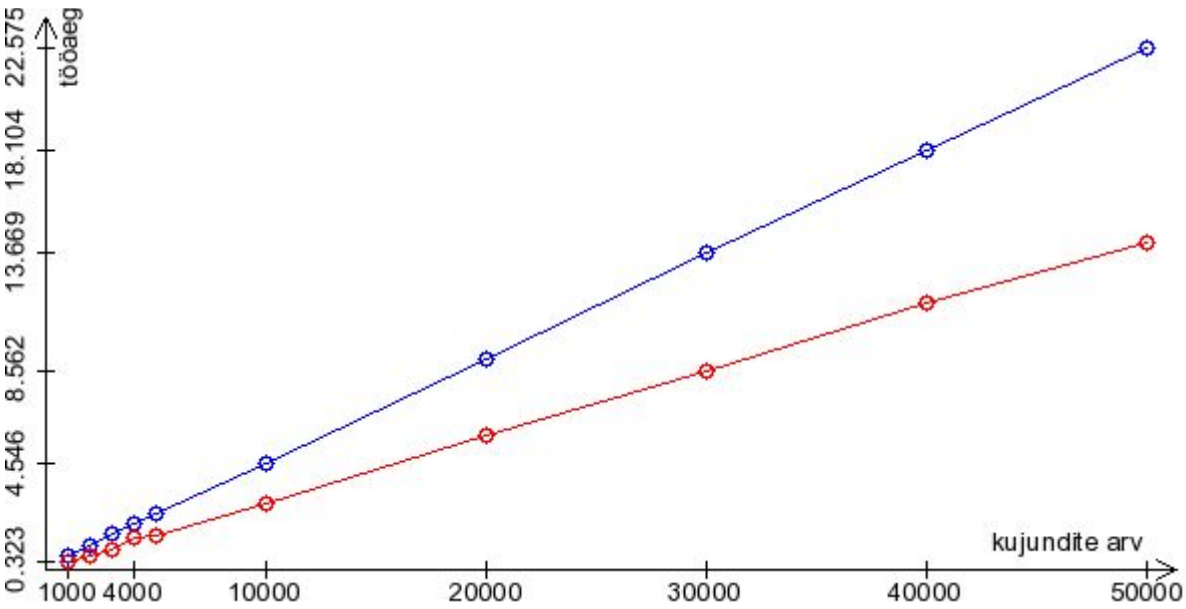
5 ELF - Executable and Linking Format (käivitatav ja lingitav failiformaat)

osaliselt ka keeles C. Ehkki C realisatsioon ei ole täielik ning toetatud on ainult paar käsku tehakse siiski sisendfaili lugemisel kujundite paisktabelisse paigutamiseks läbi samad arvutused, mis Perli analoogis. Seega on programmide kiiruste võrdlemine õigustatud.

Programmide testimiseks genereeriti käskude fail, mis koosnes etteantud arvust ristkülikutest. Eesmärgiks oli uurida, kuidas sõltub programmi tööaeg sisendfaili käskude arvust. Teste tehti kahe erineva võimsusega masinal: 2,80 GHz protsessori ning 3,7 GB operatiivmäluga serveril ja 634 Mhz ning 64 MB operatiivmäluga lauaarvutil. Programmide töökiirused on esitatud graafikuna joonistel 7 ja 8. Mõlemalt jooniselt on silma järgi näha, et tööaja sõltuvus sisendi suurusest on lineaarne. Samuti on näha, et keele SWOG realiseerimine C keeles on igal juhul õigustatud, sest kiiruste vahe on märgatav.



Joonis 7: Kulunud sekundeid pildi loomiseks lauaarvutil, ülemise joonega on toodud Perli ning alumisega C programmi tööaeg.

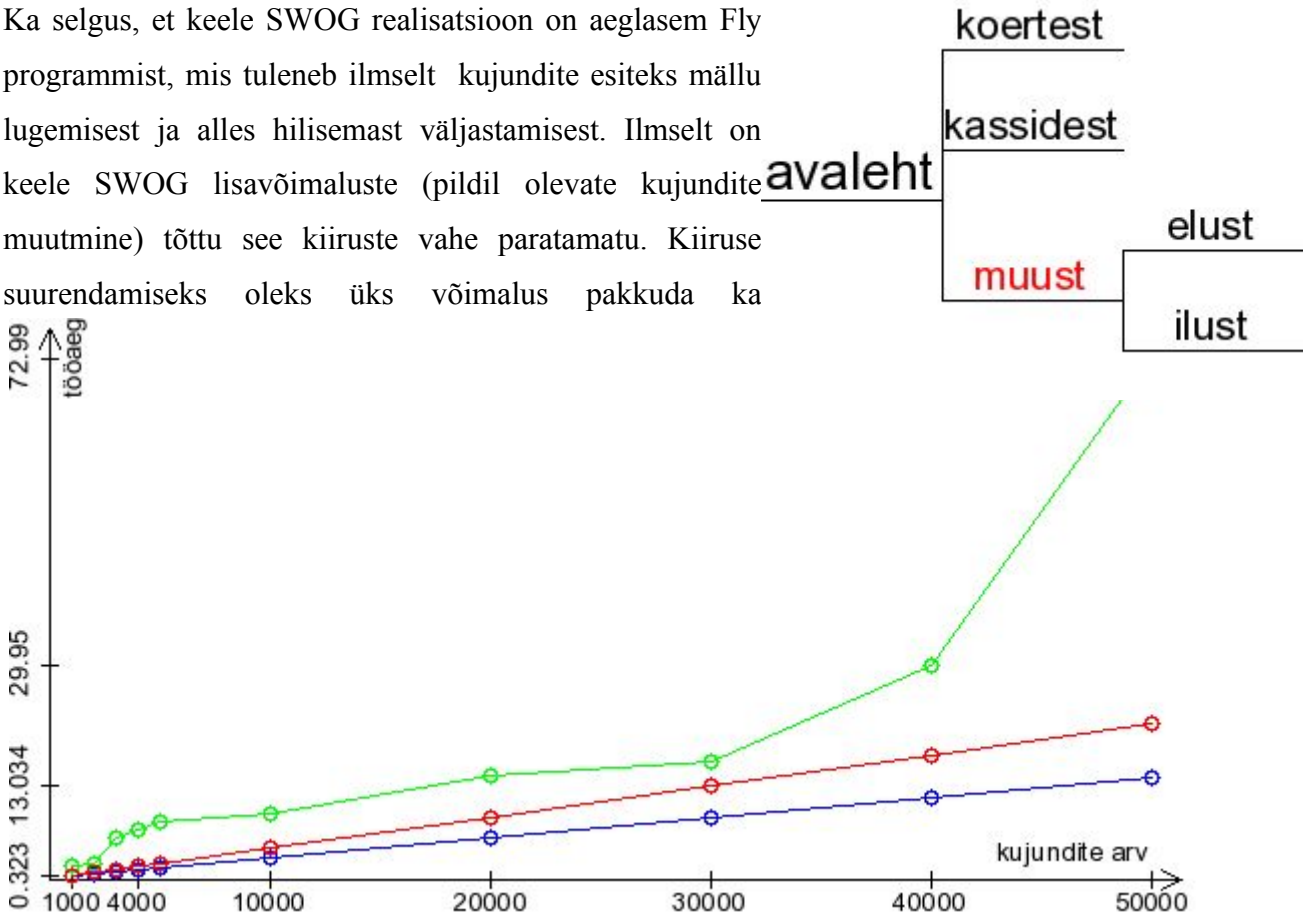


Joonis 8: Kulunud sekundeid pildi loomiseks serveril, ülemise joonega on toodud Perli ning alumisega C programmi tööaeg.

Lisaks on tehtud põgusad testid ka Fly ning SVG kiiruste mõõtmiseks. SVG testimisel mõõdeti käsistopperiga pildi kuvamiseks kulunud aega, pildid loeti kohalikust arvutist, reaalses rakenduses lisanduks ka pildi allalaadimise aeg. Testitavad pildi koosnesid etteantud arvust üksteise peale asetatud punastest ristkülikutest, kusjuures pildi kuvamise lõpu fikseerimiseks oli viimane ristkülikuks sinine. Testarvutiks oli 1,51 GHz protsesssori ning 192 MB muutmäluga lauarvuti, piltide kuvamiseks kasutati programmi *Adobe SVG Viever 3.03*, mida jooksutati brauserist *Internet Explorer*.

Keelt Fly testiti serveril analoogselt keele SWOG testimisega. Joonisel 9 on toodud Fly ja SVG testi tulemused. Et Fly on realiseeritud keeles C on võrdluseks lisatud ka keele SWOG C versiooni tööaeg. Jooniselt on näha, et ilmselt SVG testimismeetod ei olnud päris täpne, sest tulemuste joon ei ole lineaarne. Samuti on näha, et ajalises mõttes on keerukamate piltite serverist rastergraafikana väljastamine tunduvalt kasulikum kui vektorgraafika kasutamine, sest kliendil kulub vektorgraafika pildina kuvamiseks tarbetult palju aega.

Ka selgus, et keele SWOG realisatsioon on aeglasem Fly programmist, mis tuleneb ilmselt kujundite esiteks mällu lugemisest ja alles hilisemast väljastamisest. Ilmselt on keele SWOG lisavõimaluste (pildil olevate kujundite muutmine) tõttu see kiiruste vahe paratamatu. Kiiruse suurendamiseks oleks üks võimalus pakkuda ka vähendatud võimalustega SWOG keelt, mis ei loeks kujundeid vahepeal mällu vaid analoogselt programmiga Fly väljastaks nad kohe soovitud formaati.



Joonis 9: Ülemine joon tähistab pildi kuvamise aega SVG interpretaatoril, keskmine näitab SWOG C versiooni ning alumine programmi Fly tööks kulunud aega.

2.9. Kasutatud teegid

Selles jaotises on antud väike ülevaade keele SWOG realiseerimisel kasutatud olulisematest teekidest, nende ajaloost ja autoritest.

2.9.1. GD

GD on avatud lähtekoodiga teek dünaamiliselt piltide loomiseks. Selle autor on Thomas Boutell, kes hetkel elab Ameerika Ühendriikides Philadelphias ja töötab oma firmas Boutell.Com, Inc programmeerijana [url:Bou]. GD teegi avaldas ta esialgu veebilehel GIF piltide joonistamise hõlbustamiseks augustis 1994 [news:Bou94]. Praeguseks on teegil mitmeid kaasautoreid ning loomulikult on toetatud ka muud failiformaadid, näiteks PNG ja JPEG, töö kirjutamise hetkel oli viimane versioon välja lastud 11. märtsil 2004 [url:GD].

2.9.2. GD.pm

Teek GD on tehtud kasutatavaks mitmetes programmeerimiskeeltes, samuti Perlis. Perli mooduli, mis võimaldab otsepöördust GD teegi poole, autor on Lincoln Stein, kes hetkel töötab teadurina Cold Spring Harbori Laboratooriumis, mis asub Ameerika Ühendriikides New Yorgi osariigis [url:Ste]. Esimene versioon oli valmis hiljemalt juunis 1995 [news:Hed95], koos GD teegi uuendustega on ka uuendatud selle Perli liidest, töö kirjutamise hetkel oli viimane versioon välja lastud 7. märtsil 2005 [url:GD.pm].

2.9.3. PCRE

PCRE teek pakub funktsioone, mis realiseerivad Perli süntaksi ning semantikaga regulaaravaldiste kasutamist. Teegi autor on rakendusmatemaatiku kraadiga PhD Philip Hazel, kes töötab Inglismaal Cambridge'i Ülikoolis. Algselt lõi ta PCRE selleks, et seda kasutada meiliserveri Exim loomisel, aga praeguseks on teek kasutusel paljudes projektides. Teek valmis aastal 1997, töö kirjutamise hetkel oli viimaseks versiooniks 5.0, mis on välja lastud 13. septembril 2004 [url:Haz04b, url:HazelBio].

Peatükk 3

Keele SWOG kasutusvõimalusi

Järgnevalt on paari pikema näite varal esitatud võimalusi, kuidas loodud graafikakeelt ning interpretaatorit kasutada. Töövahendit SWOG on juba ka praktiliselt kasutatud, näitena saab tuua Marten Teino rakenduse, mille jaoks on keelde lisatud ka käsk `lineGraph` [Tei05].

3.1. Kiire võimalus veebilehele pildi klikkimise kaardi lisamiseks

Keel SWOG pakub võimalust, et peale pildi joonistamist on võimalik niiöelda tasuta lisana saada ka selle kujundite klikitavat kaarti, antud näide seda demonstreeribki.

Ülesande püstitus: Veebilehele on vaja puu kujulist graafilist menüüd nagu näha näite 15 joonisel.

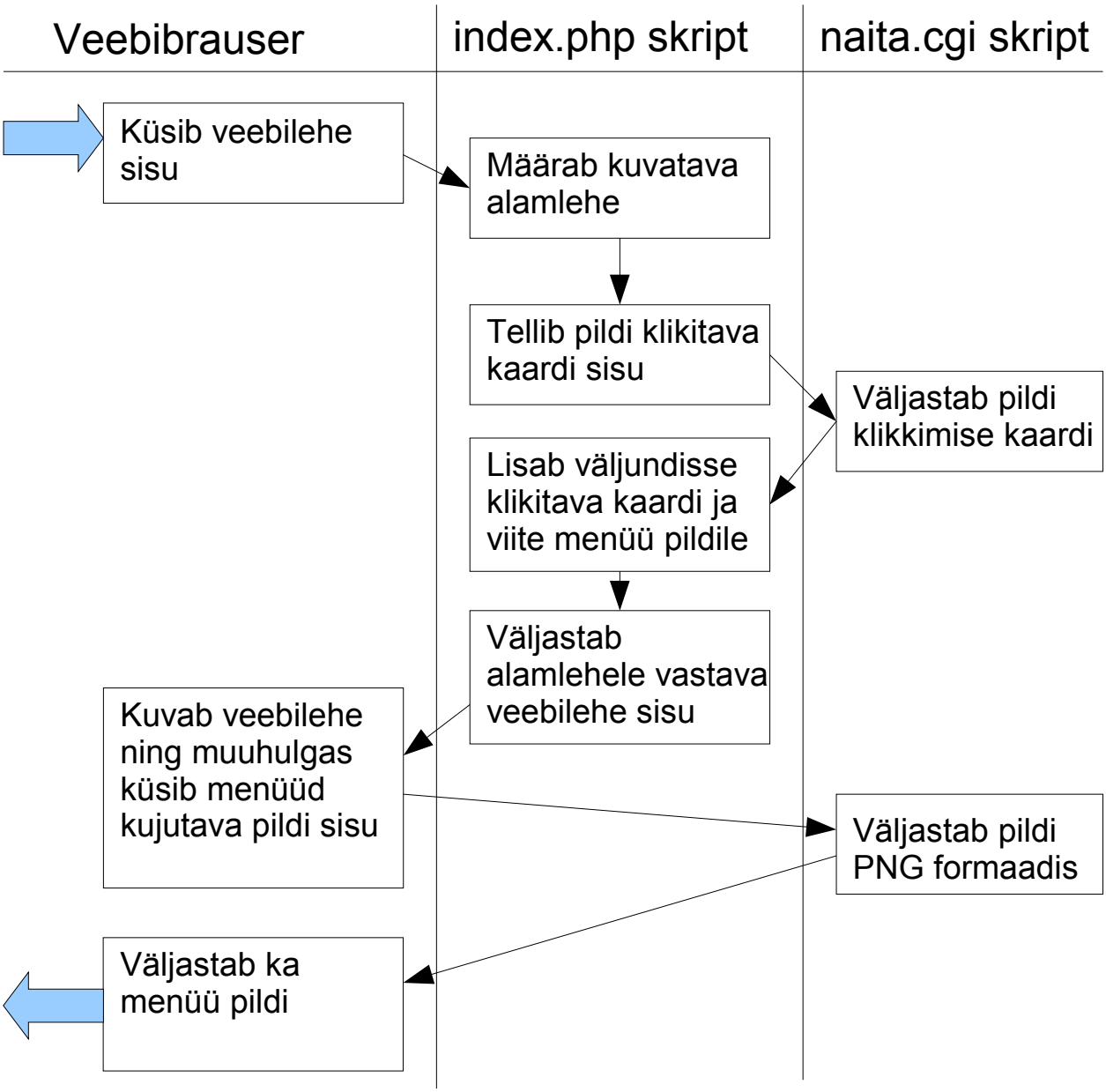
```
new 270,200
font {fonts/ARIAL.TTF} 20 :arial20
font {fonts/ARIAL.TTF} 16 :arial16
#määratakse vaikimis font:
setFont arial16

#lisatakse lehekülje menüü puuna,
#puu tippudeks on alamlehed
tree 0,0 270 200 $
(arial20 {avaleht} :main <?leht=main> $
   ({koertest} :dogs <?leht=dogs>;$
    {kassidest} :cats <?leht=cats>;$
    {muust} :other <?leht=other>$
       ({elust} :life <?leht=life>;$
        {ilust} :nice <?leht=nice>))$
) :puu
```

*Näide 15: Fail **pilt.swog** ning menüü pildina.*

Lahendus: Lehekülje struktuur kirjeldatakse keele SWOG formaadis failis, sinna lisatakse alamlehekülge nimed ning lingid nendele. Esitatud pilti kuvatakse veebilehel klikitava kaardina, kusjuures pildi kirjeldusse lisatakse käsk, mis muudab aktiivse lehekülje värvi. Lahenduses on kasutatud keeli PHP, Perl ning SWOG. Ülevaate andmiseks on lihtsustatud protsessidiagramm esitatud joonisel 10, täpsemaks kirjelduseks on lisatud ka süsteemi loovad failid (*pilt.swog* näites 15 ja *naita.cgi* ning *index.php* ülejärgmisel leheküljel).

Lisaks on toodud näide HTML kujul väljundist:

```
<map name="map"><area shape="poly"
coords="206,171,244,171,244,154,206,154" href="?leht=nice">
<area shape="poly"
coords="202.5,121,247.5,121,247.5,104,202.5,104"
href="?leht=life">
<area shape="poly" coords="106,146,164,146,164,129,106,129"
href="?leht=other">
<area shape="poly" coords="91,71,179,71,179,54,91,54"
href="?leht=cats">
<area shape="poly" coords="97.5,21,172.5,21,172.5,4,97.5,4"
href="?leht=dogs">
<area shape="poly" coords="2.5,96,87.5,96,87.5,75,2.5,75"
href="?leht=main">
</map><img src=naita.cgi?type=png&item=main usemap="#map">
<b>Ees on lehekülg: main</b>
```



Joonis 10: Veebilehele klikitava kaardiga menüü lisamise lihtsustatud protsessidiagramm veebibrauseri, PHP ning Perli skripti ülesannetest lähtuvalt.

Fail **naita.cgi**, mis annab edasi PHP korraldused keele SWOG mootorile

```
#!/usr/bin/perl -w
use strict;
no strict "refs";
use POSIX;
use CGI;
use SWOG;

my $cgi=new CGI;
SWOG::readObjects("pilt.swog");
my $page=$cgi->param("item");

#aktiivse lehekülje värvi muutmine:
SWOG::execute_command("alter red color puu.".$page);

#sõltuvalt skripti avamise parameetrist type väljastatakse
#kas rasterpilt PNG formaadis või pildi klikkimise kaart
if ($cgi->param("type") eq 'png'){
        print "Content-type: image/png\n\n";
        SWOG::output_raw_png();
}else{
        print "Content-type: text/html\n\n";
        SWOG::output_html();
}
```

Fail **index.php**, mis on kogu rakenduse loogika eest vastutav:

```
<?php
//abifunktsioon, mis kuvab antud URLi sisu tekstina
function sisu($url){
        $handle = fopen($url, "rb");
        $contents = '';
        while (!feof($handle)) {
                $contents .= fread($handle, sizeof($handle));
        }
        fclose($handle);
        return $contents;
}
if (isset($_REQUEST["leht"]))
        $item=$_REQUEST["leht"];
else
        $item="main";
//kuvame html klikkimise kaardi:
echo "<map name=\"map\">";
echo sisu($_SERVER["SCRIPT_URI"]."naita.cgi?item=$item");
echo "</map>";
echo '<img src=naita.cgi?type=png&item='.$item.'
          usemap="#map">';
//järgnevalt kuvaks PHP lehekülje sisu,
//seda sõltuvalt muutuja $item väärtusest
print "<b>Ees on lehekülg: $item</b>";
?>
```

3.2. Logode joonistamine

Antud näide demonstreerib kuidas kasutada enda defineeritud objekte. Logod on bio-informaatikas kasutatavad visualiseerimisvahendid, mis näitavad erinevaid tähtede suurusi kasutades sümbolite osatähtsust, antud näites loodav logo on näha joonisel 11.

Joonis 11: Logo

Pilt on joonistatud nõnda, et iga tähe joonistamiseks defineeritakse tähe nimeline objekt, objektis joonistatakse täht sobivatest kujunditest valmis ning kõige ette lisatakse koordinaatide süsteemi skaleerimise käsk, mis korrutab ordinaadid tähe joonistamiseks antud argumendiga läbi, nõnda saadakse joonistamisel tähe vajalik kõrgus. Reaalses visualiseerimis-rakenduses tuleks programmis lisada pildi kirjeldusse veel ainult vajalikud tähed ning nende suhtarvud.

Koodi näide keeles SWOG, mis loob pildi mida esitatakse joonisel 11:

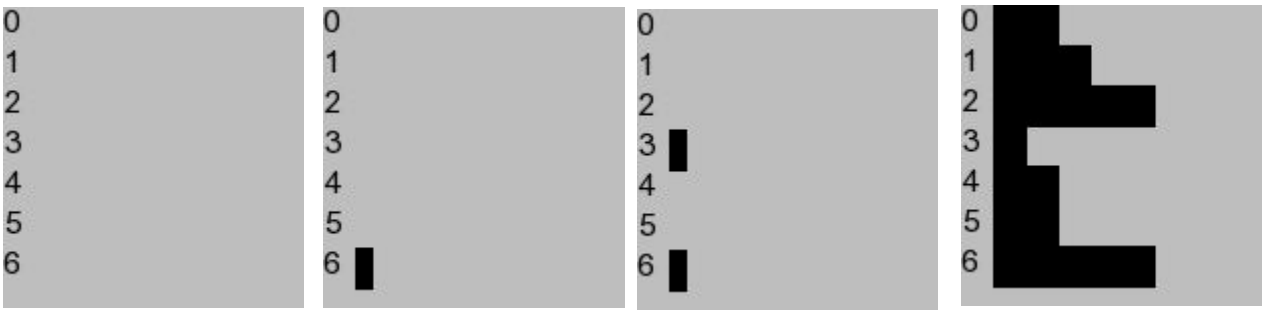
```
new 350,210

#joonistab antud punkti 70 piksli laiuse tähe,
#kusjuures punkt kuhu objekt lisatakse jääb tähe
#alla vasakusse nurka
#igal tähel on defineeritud punktid ylemVasak ning alamParem,
#mis määravad tähtede piirjooned
#argument: %1 - tähe kõrgus suhtarvuna, normaalskaalal vastab
#ühele ühikule 100 pikslit

defObject G
  scale 1,-%1
  setColor orange
  farc 35,50 35 50 10 315 :kaar
  fellipse 35,50 25 40 white
  frect (%this.kaar.p1) -25 10
  frect (%this.kaar.p2) -7 -15
  point 0,100 :ylemVasak
  point 70,0 :alamParem
endObject
```

```
defObject T
  scale 1,-%1
  setColor red
  fpoly 0,100 70,100 70,70 45,70 45,0 25,0 25,70 0,70
  point 0,100 :ylemVasak
  point 70,0 :alamParem
endObject
defObject A
  scale 1,-%1
  setColor green
  fpoly 0,0 27,100 43,100 70,0 50,0 35,80 20,0
  frect 20,30 50,50
  point 0,100 :ylemVasak
  point 70,0 :alamParem
endObject
#ja järgnevalt maatriksi lisamine, need käsud reaalselt lisaks
#igal täitmisel kasutaja skript
object 20,200 A 1.5 :t11
object (t11.alamParem) T 0.5 :t21
object (t21.ylemVasak) G 0.5 :t22
object (t21.alamParem) G 0.4 :t31
object (t31.ylemVasak) A 0.6 :t32
object (t31.alamParem) G 0.3 :t41
object (t41.ylemVasak) T 0.5 :t42
#joonistame ka koordinaadistiku:
line 20,200 300 0
line 20,100 300 0
line 20,0 300 0
```

Märkuse korras peab mainima, et antud näitest on ka näha, et keele SWOG realiseeritud käskude hulk ei pruugi alati piisav olla. Tähe G kujutamisel tuntakse näiteks puudust sellise sektori joonistamisest, millel oleks seest ring välja lõigatud, aga kahjuks on paratamatu, et kõigi võimalike kujundite joonistamist lihtsalt ei ole võimalik toetada, kuid õnneks on realiseeritud primitiivid, millest keerukamaid kõveraid kokku panna.

3.3. Animatsioonide loomine

Keele SWOG abil saab luua ka lihtsamaid animatsioone. Järgnevalt näide rakendusest, mis visualiseerib paisktabeli mälupesade täitumist. Rakendus on kirjutatud keeles Perl ning mõeldud veebipõhiseks kasutamiseks. Väljundiks annab programm GIF animatsiooni, selle esimesi ning viimast kaadrit on näha joonisel 12.



Joonis 12: Animatsiooni kaadrid (1., 2., 3. ning viimane).

Animatsiooni loov Perl kood:

```
#!/usr/bin/perl -w
use lib ".";
use SWOG;
#muutujasse $com lisame kõik SWOG käsud
$com="new 150,150 gray\n";
$com.="font {fonts/ARIAL.TTF} 12 :arial\n";
$com.="setFont arial\n";
#mugavuse mõttes teema skaala suureks,
#et saaks kasutada koordinaatidena arve
#1,2,3...
$com.="scale 8,20\n";
#lisame pesade numbrid
for ($i=0;$i<7;$i++){
  $com.="string 0,$i {$i}\n";
}
$com.="origin 2,0\n";
#paisktabel, kus peame meeles,
#palju on vastavaid väärtuseid lisatud
%hash=();
for ($i=0;$i<40;$i++){
   $key=(rand()*100000)%7;
   $pos=$hash{$key};
   $pos=0 if (!defined $pos);
   #teeme animatsiooni sekundilise pausi:
   #ja joonistame järgmise mälu hõivamise
   $com.="frect $pos,$key 1 1\n";
   $hash{$key}=$pos+1;
}
#kui pilt valmis, siis enne
#uuesti joonistamist ootame 10 sekundit
$com.="export 1000\n";
SWOG::readObjectsFromBuffer($com);
print "Content-type: image/gif\n\n";
SWOG::output_raw_gifanim('/tmp/');
```

3.4. Graafikute joonistamine

Pikema näite raames on näidatud, kuidas on loodud töös esitatud graafikud (joonis 7, lehekülg). Graafiku joonistamise põhiidee on selles, et tänu koordinaatide transformeerimise võimalustele saab graafiku punkte lisada andmete enda ühikutes. Selleks defineeritakse käsuga `defObject` objekt *jooned*, sinna lisatakse andmefailis antud ühikutes graafiku punktid ja neid ühendavad jooned. Peale andmefaili on läbimist on tuvastatud ka graafikul kujutatavate andmete skaala ulatus. Seejärel muudetakse käsuga `scale` koordinaadistik sobivaks ning lisatakse käsuga `object` *jooned* pildile. Täpse programmi koodi leiab lisast 3.

Kokkuvõte

Tänapäeval on programmeerijal oma andmete esitamiseks võimalik valida paljude visualiseerimisvahendite vahel, paraku on aga nende kasutamine kas liiga keeruline või on nende poolt pakutavate graafika primitiivide arv nii väike, et üldjuhul kulub visualiseerimiseks tarbetult palju aega ning vaeva.

Tühimiku täitmiseks on käesoleva töö raames loodud graafikakeel SWOG, mis on mõeldud andmete visualiseerimiseks. Keele süntaks on loodud võimalikult lihtne, iga käsu puhul järgitakse ühtset stiili. Süntaks võimaldab luua ka oma objekte, mida on võimalik pildi kirjeldamisel erinevate parameetritega korduvalt kasutada. Keeles on toetatud koordinaatide süsteemi muutmine, mis lubab pildi kirjeldajal lisada kujundeid enda andmetele vastavas süsteemis.

Lisaks pildi välimuse kirjeldamisele on võimalik tuvastada etteantud punkti sisaldavad kujundid, mis hõlbustab interaktiivsete visualiseerimissüsteemide loomist.

Keele esimene versioon on realiseeritud keeles Perl, rasterpiltide joonistamiseks on kasutatud GD teeki. Samuti on võimalik väljastada pilt SVG formaadis vektorgraafikana, toetatud on ka GIF animatsioonide loomine. Väljundiks võib olla ka pildi klikkimise kaart, mis on mõeldud veebilehtedele interaktiivsuse lisamiseks.

Töös on esitatud ka kirjeldus, kuidas keelt saaks realiseerida programmeerimiskeeles C, valminud on ka programmi esmane versioon, kuid selle lõplik realiseerimine on üks töö edasiarenduse võimalustest.

Graphics language SWOG

Bachelor Thesis

Jaanus Hansen

Abstract

At present, there still is a need for a simple tool which could be used in programs to visualize data. There are several tools which complete the task (e.g. graphics APIs, MetaPost, Fly, Gnuplot), but their syntax for describing pictures is usually complex or offers so little commands that drawing is laborious and time consuming.

In this paper a graphics language Simple Web Object Graphics (SWOG) is represented. SWOG is meant to be used in programs to visualize data. The language has simple syntax with a common form for all commands. It is also possible to define new objects and use them while describing the picture. SWOG implements commands for coordinate system transformation to make drawing more flexible. The language interpreter includes a facility to enumerate objects containing the given point which helps to create interactive visualisation tools.

The first version of SWOG is implemented in Perl, the implementation uses GD.pm module to output raster images. It is also possible to convert from SWOG to SVG vector graphics format, create GIF animations and output client side image maps for HTML.

The paper also represents skeleton for implementation of SWOG in C, but the complete implementation is still under development.

Kasutatud kirjandus

[Bin96] A. Binstock. *Hashing Rehashed*, Dr Dobbs' Journal April 1996, lk. 24-33, 1996.

[FDFH90] J. D. Foley, A. van Dam, S. K. Feiner, J. F.Hughes. *Computer graphics: principles and practice*, Adisson-Wesley, lk. 201-210, 1990.

[Hob92] J. D. Hobby. *Introduction to MetaPost*, Proceedings of EuroTeX'92, 1992. http://cm.bell-labs.com/who/hobby/92_2-21.pdf - viimati vaadatud 22.05.05.

[Hol99] S. Holzner. *Perl Core Language: Little Black Book*, Coriolis Group, lk. 4, 1999.

[Jen95] I. Jentson. *Arvutusgeomeetria*, Õppevahend TK õpilastele, lk. 2-3, 1995.

[Kih97] J. Kiho. *Algoritmid ja andmestruktuurid*, Tartu Ülikooli kirjastus, lk. 98-100, 1997.

[Kil98] M. Kilp. *Algebra I*, Tartu, lk. 138-139, 1998.

[KK04] D. Krushevskaja, M. Kull. *Andmete visualiseerimise süsteemi disain ja prototüüp. Rühmatöö aines Andmekaevandus*, Arvutiteaduse instituut, Tartu Ülikool, 2004. http://www.egeen.ee/u/vilo/edu/2004-05/Andmekaevandus/Ryhmatood/T3_Visualiseerimine/ryhm3_visual.pdf - viimati vaadatud 22.05.05.

[Tei05] M. Teino. *Stringide ja mustrite otsimine ja visualiseerimine*, Semestritöö, Tartu Ülikool, 2005.

[url:Bou] T. Boutell. *Thomas Boutell's Home Page*. http://www.boutell.com/boutell/ - viimati vaadatud 22.05.05.

[url:GD] *GD Graphics Library*. http://www.boutell.com/gd/ - viimati vaadatud

22.05.05.

[url:GD.pm] L. Stein. *GD.pm module*. http://search.cpan.org/dist/GD/ - viimati vaadatud 22.05.05.

[url:Gle] M. Gleeson. *Fly: create images on the fly*. http://martin.gleeson.com/fly/ - viimati vaadatud 22.05.05.

[url:Gnuplot] *GNUPLOT: An Interactive Plotting Program*. http://www.gnuplot.info/docs/gnuplot.html - viimati vaadatud 22.05.05.

[url:Haz04a] P. Hazel. *PCRE man pages*. http://www.pcre.org/pcre.txt - viimati vaadatud 22.05.05.

[url:Haz04b] P. Hazel. *PCRE - Perl Compatible Regular Expressions*. http://www.pcre.org/ - viimati vaadatud 22.05.05.

[url:HazelBio] *Philip Hazel Biography*. http://www.uit.co.uk/exim-book/author-bio.htm - viimati vaadatud 22.05.05.

[url:Ste] L. Stein. *Lincoln Stein's Home Page*. http://stein.cshl.org/~lstein/ - viimati vaadatud 22.05.05.

[url:SVG] *Scalable Vector Graphics (SVG) 1.1 Specification*. http://www.w3.org/TR/SVG/ - viimati vaadatud 22.05.05.

[url:Tob94] G. Tobin. *MetaFont for beginners*. http://www.ntg.nl/doc/tobin/mf4begin.pdf - viimati vaadatud 22.05.05.

[news:Bou94] T. Boutell. *[comp.infosystems.www] ANNOUNCE: gd 1.0: a GIF-drawing library for your programs*, comp.archives, 18.08.1994.

[news:Hed95] M. Hedlund. *Re: gd anyone?*, comp.lang.perl.misc, 16.06.1995.

Lisad

Lisa 1

Keele SWOG käsud

Veidi käskude esituse formaadist: Iga käsu tutvustuse esimesel real on rasvases kirjas käsu nimi ning sellele järgnevad argumendid. Kohustuslikud argumendid on allajoonitud ning vabatahtlikud joonimata. Järgnevalt on vajadusel lisatud argumentide üksikasjalikud kirjeldused, konstruktsioon (*väärtus1*|*väärtus2*|..|*väärtusN*) tähistab, et argumendiks peab olema üks sõnedest hulgast {*väärtus1*, *väärtus2*, .., *väärtusN*}. Käsu nime ette lisatud (f) tähistab, et käsul on ka täidetud kujundi joonistamise versioon, taolised käsud on näiteks `fcircle` ja `circle`, esimene neist väljastab täidetud ringi ning teine ringjoone.

Järgnevalt käskude alfabeetiline loend, kasutusnäited on toodud ainult kolme keerukama käsu jaoks, ülejäänute näited leiab tööle lisatud optiliselt kettalt või internetist aadressilt http://kotkas.ebc.ee/u/hansen/docsource/doc.html.

alter new_value property object

Muudab objekti omaduse, üldjuhul on lubatud `property` väätusteks `color` või `font`. Arvestada tuleb siiski sellega, et muutmine ei mõjuta objekti suurust keele SWOG mootori jaoks: kui muuta fondi suurust, siis tekst joonistatakse küll uue fondiga, aga näiteks kliki tuvastus kasutab ikka vana suurust.

(f)**arc** coordinate x_radius y_radius start_deg end_deg color
`start_deg`: Antud kraadides, määrab kust kaare joonistamist alustada
`end_deg`: Antud kraadides, määrab mis nurgani kaar joonistada

Joonistab kaare, nurk ellipsil suureneb parempoolseimast küljest päripäeva.

axis `arrowStyle coverage ticked labeled-ticks label style`
`arrowStyle`: (*arrow*|*none*) noolepea tüüp. Vaikimisi väärtus: arrow
`coverage`: (*start..end*|*auto*) Kui suurt osa skaalast kajastab, kui antud auto, siis leitakse piirid sõltuvalt graafikust. Vaikimisi väärtus: auto
`ticked`: (*ticks*|*noTicks*) Kas skaalal on kriipsud?. Vaikimisi väärtus: noTicks
`labeled-ticks`: (*{value1;value2;..;valueN}*|*start step*|*auto*|*none*|*all*) mis tipud on märgistatud, võimalikud kujud: {value1;value2;..;valueN} - loetletud väärtused start step - algus ja samm auto - automaatne, nii et mahuks võimalikult palju none - ei ühtegi all - kõik graafikul väärtust omavad. Vaikimisi väärtus: auto
`label style`: (*below=right*|*above=left fontName*) märgistuse positsioon ja font kas below=right, above=left või none ning fondi nimi

Defineerib koordinaatteljestiku telje stiili, vaikimisi on kasutatav telg `defAxis`, mis oleks määratud järgneva reaga:
`axis arrow auto ticks auto above medium :defAxis .`

`(f)`**`circle`** `coordinate radius color`
`coordinate`: Keskpunkt
`radius`: Raadius
`color`: Värv

Joonistab ringi.

`color` `red green blue alpha :name`
või
`color` `color-name alpha :name`
`red`: (*0..255*)
`green`: (*0..255*)
`blue`: (*0..255*)
`alpha`: (*0..127*) läbipaistvus. Vaikimisi väärtus: 0
`color-name`: defineeritud värv, mille läbipaistvust muudetakse
`:name`: värvile antav nimi

Defineerib värvi määrates tema punase, sinise, rohelise ning läbipaistvuse komponendid.

`connect` `p1 p2 color`
`p1`: Võib olla kas objekti nimi (objekt peab olema ring või ellips) või punkt
`p2`: Võib olla kas objekti nimi (objekt peab olema ring või ellips) või punkt

Ühendab antud ringid või ellipsid sirgjoonega nii, et joon tõmmatakse kaarest kaareni. Kumbaksi argumendiks võib olla ka punkti koordinaat, sel juhul tõmmatakse joon antud punkti.

`coordsys` `origin-and-direction`
`origin-and-direction`: argumentide järjekord pole oluline, nullpunkti määravad south east west north ning telgede suuna left up right down. Vaikimisi väärtus: center right up

Määrab koordinaatide süsteemi, vaikimisi kehtib `coordsys west north right down` (ehk siis 0-punkt on üleval vasakus nurgas ja teljed jooksevad paremale ja alla).

`defObject` `class-name`
row1
...
rowN
endObject
`class-name`: objekti klassile antav nimi

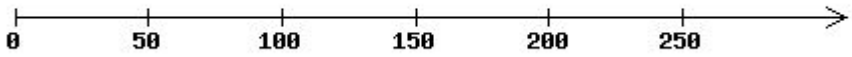
Defineerib objekti klassi, mida saab hiljem joonistamiseks kasutada. `row1..N` võivad sisaldada kõiki keele SWOG käske (v.a. `defObject`) lisaks võib ridades kasutada ka objektile ette antud parameetreid kujul `%1, %2 ...` ja samuti muutujat `%this`, millega asendatakse sisestatava objekti nimi.

`drawAxis` `coordinate length axis start end {values} color`
`length`: Telje pikkus
`axis`: Telg, mis on defineeritud käsuga axis või "defAxis"
`start`: Esimene teljel kuvatav väärtus.

`end`: Viimane teljel kuvatav väärtus.
`{values}`: Semikoolonitega eraldatud väärtused, mida teljel kuvada

Joonistab koordinaatide telje, kasutamise kohta on toodud näide 16.

```
new 450,60

#määrame skaala kajastamise vahemiku
axis ticks 0 50 below medium :telg

#joonistame telje, mis kajastab vahemikku 0-300
drawAxis 20,30 400 telg 0 300
```



Näide 16: Käsu drawAxis kasutamine.

(f)**ellipse** coordinate x_radius y_radius color
`x_radius`: raadius mööda x-telge
`y_radius`: raadius mööda y-telge

Joonistab ellipsi.

export clean-screen duration
`clean-screen`: (*clean*) Kui väärtuseks on "clean", siis puhastab enne järgmise kaadri joonistamist ekraani
`duration`: Kaadri kestvus sajandiksekundites

Animatsiooni kaadri väljastamine toimub `export` käsule reageerides, seega peaks see animatsiooni puhul olema ka faili viimaseks käsk, muidu viimast kaadrit animatsiooni ei lisataks. Kaadreid väljastatakse ainult siis, kui on antud korraldus luua gif animatsioon, vastasel juhul `export` käske ignoreeritakse.

fill coordinate color

Täidab pildi antud punktist alates antud värviga, seda kuni mõni teine värv vastu tuleb.

fillToBorder coordinate borderColor fillColor

Täidab pildi antud punktist alates värviga `fillColor`, seda kuni `borderColor` värvi punktid vastu tulevad.

font {filename} size angle
`{filename}`: .ttf faili nimi
`size`: suurus punktides
`angle`: nurk kraadides, pööre toimub vastupäeva. Vaikimisi väärtus: 0

TrueType (.ttf) fondi kasutamiseks (vaikimisi on GD fontide jaoks on eeldefineeritud tiny, small, medium, large ja giant nimega fondid)

Programmi vaikimisi paigalduses on kataloogis `fonts/` fondid `ARIAL.TTF` ning `Manga Temple.ttf`, need on pärit lehelt http://www.webpagefonts.com/ ja on väidetavalt vabalt kasutamiseks.

SVG formaati väljastamisel kasutatakse ka lingi selgitavat teksti /alt/, selle formaat on: `omadus1|väärtus1|..`, kui sisuks on näiteks `family|Serif|weight|bold`, siis SVG väljundisse lisatakse `font-family="Serif" font-weight="bold"`

`image` `coordinate width height {filename} imageStartPoint imWidth imHeight`
`coordinate`: Punkt, kuhu paigutatakse pildi ülemine vasak nurk
`width height`: väljundi pikkus laius (ekraani mõõdud sõltuvad jooksvast koordinaatide süsteemist) Vaikimisi sellised, et pilt joonistakse originaalmõõdus
`{filename}`: pildifaili nimi
`imageStartPoint`: Failist võetava pilti vasaku ülemise nurga koordinaadid. Antud pikselites, sõltumatu hetke koordinaatide süsteemist. Vaikimisi väärtus: 0,0
`imWidth imHeight`: Kui suur osa failist kuvatakse. Vaikimisi kuni pildi parema alumise nurgani

Välise pildifaili joonisele lisamise käsk.

`line` `coordinate dimension color`

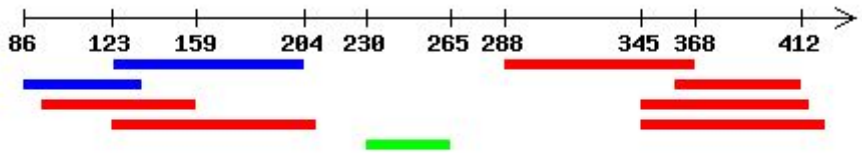
Joone joonistamine.

`lineGraph` `axisStyle colWidth lineWidth align coordinates dimension (lines1) :line1Group line1Color ...`
`axisStyle`: telje stiil, defineeritud käsuga axis. Vaikimisi väärtus: defAxis
`colWidth`: Rea laius
`lineWidth`: Joone laius
`align`: (*up*|*down*|*none*) koord. telje paigutus andmete suhtes. Vaikimisi väärtus: up
`:line1Group`: grupi nimi, sama grupiga jooned pannakse kokku, erinevate gruppide omad aga mitte. Vaikimisi pannakse iga joon eraldi gruppi.

`linesN` on kujul `lineCoords /alt/ :name <link>` ja eraldatud semikoolonitega. Kusjuures lineCoords määrab joone pikkuse ning on kujul kas `(algus, lõpp)` või `(algus pikkus)`.

Käsk laseb paigutada antud jooned graafikule selliselt, et ükski neist ei lõikuks ja kõik oleksid nähtavad.
Kui lastakse joonistada ka telg, siis selle skaala punktidele pääseb ligi viitega `(graafi_nimi.val_VALUE)`, kus `VALUE` on väärtus skaalal.
Kasutamise kohta on toodud näide 17.

```
new 500,120

defineerime kasutatava telje:
axis auto ticks all below medium :telg

#punased ja sinised pannakse samadele ridadele,
#aga rohelised eraldi..

lineGraph telg 10 5 up 60,10 400 200  $
(230 35) green  $
(123,209 94,159 345,422 345,416 359,412 288,368 ) :a red  $
(86,136 124,204 ) :a blue
```



Näide 17: Käsu lineGraph kasutamine.

moveAfter object1_name object2_name
object1_name: Objekt, mille asukohta muudetakse
object2_name: Objekt, mille järele objekt1 viiakse (st. object1 joonistakse peale object2-te)

Muudab objektide joonistamise järjekorda.

moveBefore object1_name object2_name
object1_name: Objekt, mille asukohta muudetakse
object2_name: Objekt, millest ettepoole objekt1 viiakse (st. object1 joonistakse enne object2-te)

Muudab objektide joonistamise järjekorda.

new x,y background color
x,y: laius ja kõrgus pikselites
background color: taustavärv. Vaikimisi väärtus: white

Määrab pildi suuruse ja taustavärvi, võiks olla faili esimene käsk.

newSECorner coordinate

Muudab pildi suurust, uueks alumiseks paremaks nurgaks saab antud punkt.

object coordinate object class name arguments
coordinate: enne objekti joonistamist tehakse käsk origin coordinate. Vaikimisi väärtus: 0,0
arguments: tühikutega eraldatud sõned, tühikut sisaldava sõne edastamiseks tuleks ta panna märkide {} vahele

Objekti kasutamine. Objektis asendatakse `%1 %2 ...` antud argumentidega.

origin coordinate

Määrab uue koordinaatide süsteemi nullpunkti, punkti koordinaadid on antud hetkel kehtivas süsteemis.

outputRawGif

Võimalus koodist pildi GIF formaadis std. väljundisse väljastamiseks.

outputRawGifAnim temp-dir
temp-dir: Kataloog, mida kasutatakse ajutiste piltide hoidmiseks. Vaikimisi väärtus: /tmp/

Võimalus koodist pildi GIF formaadis animatsiooni std. väljundisse väljastamiseks.

outputRawHtml

Võimalus koodist pildi HTML kujul klikkimise kaardi std. väljundisse väljastamiseks.

outputRawPng

Võimalus koodist pildi PNG formaadis std. väljundisse väljastamiseks.

outputRawSVG

Võimalus koodist pildi SVG formaadis std. väljundisse väljastamiseks.

point coordinate :name

Defineerib punkti asukoha.

pointAdd coordinate width height
coordinate: punkt, mille koordinaatidele lisatakse antud pikkus-kõrgus
width: mitu punkti laiust lisada?
height: mitu punkti kõrgust lisada?

Määrab punkti koordinaadid liites antud punkti abstsissile ning ordinaadile antud mõõdud.

(f)**poly** coordinate coordinate coordinate other coordinate color

Hulknurga joonistamine.

(f)**rect** coordinate dimension color

Ristküliku joonistamine.

rotate coordinate degrees

Pöörab järgnevat pilti antud punkti suhtes degrees kraadi vastupäeva. Kõigil käskude jaoks

pole keerutamine tänu GD teegi omapärale korrektselt realiseeritud. Näiteks ringe oskab joonistada ainult ellipsitena, mitte ei venita neid välja. Samuti tekstide kirjutamine ei ole korrektselt realiseeritud, sest kirjutamine peegelpildis ei ole toetatud.

scale x,y

Muudab koordinaatide süsteemi korrutades edaspidi koordinaadid antud väärtustega läbi. Kui koodis on järjestikku mitu `scale` käsku, siis toimub korrutamine korduvalt st. näiteks kui antud on käsud `scale 2,2` ning `scale 3,3`, siis need käsud on samaväärsed käsuga `scale 6,6`.

setColor color_name

Määrab värvi, millega kujundeid joonistatakse, juhul kui neil pole parameetrina värvi kaasa antud, vaikimisi on selleks värviks `black`.

setFont fontName

Määrab vaikimisi fondi, esialgu on selleks `medium`.

setLineThickness thickness
`thickness`: Joone laius pikselites

Muudab järgnevalt joonistatavate joonte laiust, vaikimisi on laiuseks 1. Kui laius on ühest suurem, siis joon lisatakse pildile täidetud hulknurgana.

string orientation coordinate color font {text}
`orientation`: määrab milline sõne ümbritseva ristküliku punkt läheb alguspunkti, võimalikud väärtused on center west east north south. Kui tühjaks jätta, siis vaikimisi koordinaadistikus on selleks punktiks vasak ülemine nurk. Vaikimisi väärtus: west north

Väljastab teksti.

toGif {filename}
`{filename}`: Failinimi, kuhu salvestatakse .gif pilt, faililaiend .gif lisatakse automaatselt

Võimalus koodist pildi GIF formaadis faili väljastamiseks.

toGifAnim {temp-dir} {filename}
`{temp-dir}`: Kataloog, mida kasutatakse ajutiste piltide hoidmiseks. Vaikimisi väärtus: /tmp/
`{filename}`: Failinimi, kuhu salvestatakse .gif animatsioon, faililaiend .gif lisatakse automaatselt

Võimalus koodist pildi GIF formaadis animatsiooni faili väljastamiseks.

toHtml {filename}
`{filename}`: Failinimi, kuhu salvestatakse html formaadis klikkimise kaart

Võimalus koodist pildi HTML kujul klikkimise kaardi faili väljastamiseks.

toPng {filename}
{filename}: Failinimi, kuhu salvestatakse .png pilt, faililaiend .png lisatakse automaatselt

Võimalus koodist pildi salvestamiseks PNG formaadis.

toSVG {filename}
{filename}: Failinimi, kuhu salvestatakse .svg pilt, faililaiend .svg lisatakse automaatselt

Võimalus koodist pildi SVG formaadis faili väljastamiseks.

tree halign coordinate dimension color (rootNode(childNode1(..);childNode2(..);..))
halign: (*left*|*right*) Määrab, millises pildi servas paikneb juur. Vaikimisi väärtus: left
(rootNode(childNode1(..);childNode2(..);..)): puu struktuuri defineerimine

Joonistab ristkülikusse antud puu. Iga nodeInfo süntaks on sarnane käsu string süntaksile: color font {node_name} /alt/ :name <link>. Kasutamise kohta on toodud näide 18.

```
new 295,248

font {fonts/ARIAL.TTF} 12 :arial

setFont arial

tree 0,0 295 248 ({root} ({child1}; {child2}; {child3}
({child3_1}; {child3_2})))
```

Näide 18: Käsu tree kasutamine.

undo command1_name command2_name command3_name ...

Taastab parameetrid, mida antud käsud muutsid, taastamine toimub järjekorras paremalt vasakule.

Lisa 2

Perli mooduliga suhtlemise funktsioonid

Järgnevalt on toodud Perli protseduurid, mis on moodulis SWOG.pm kasutajale suunatud.

debug_objects()

Väljastab std-väljundisse sisse loetud objektid, kuvades nende omadused.

detect_click(X, Y)

Väljastab pildil punkti X,Y alla jäänud nime või lingiga objektid. Väljund on massiiv, mille iga element on kujul:

(objekti nimi, link, alt,\@alluvad)

Ja alluvad on juba samal kujul, mis funktsiooni väljundiks antud massiivgi.

execute_command(line)

SWOG lähteteksti ridade kaupa sisse lugemiseks. Seda funktsiooni on mugav kasutada, kui soovitakse pildile midagi lisada, näiteks teha hiirekliki kohale vastav märge. Samuti võib seda kasutada programmis ka rida realt pildi genereerimiseks.

file_gif(filename)

Lisab filename-ile .gif ja väljastab pildi GIF formaadis faili.

file_gifanim(temp directory, file name)

temp directory: Kataloog, kuhu kirjutatakse ajutised failid, hiljem need kustutatakse, soovitav näiteks /tmp/

file name: Väljundfaili nimi, automaatselt lisatakse faililaiend .gif

Väljastab .gif animatsiooni. Animeerimiseks joonistakse ajutised .png failid ja kasutades programmi convert pannakse neist animatsioon kokku.

file_png(filename)

Lisab `filename`-ile `.png` ja väljastab pildi PNG formaadis faili.

`file_svg`(`filename`)

Lisab `filename`-ile `.svg` ja väljastab pildi SVG formaadis faili. Et seda funktsiooni kasutada, on vaja, et kättesaadav oleks Perli moodul XML::Writer.

`html_clickmap`()

Tagastab pildi HTML clickmapi, mis tuleb HTML kujul kirjutada `<map name="">vahele</map>`

`image_dimensions`()
Tagastab pildi mõõdud pikselites.

`object_info`(`object_name`)

Ligipääs objekti infole. Tagastatakse paisktabel, mille liikmed sõltuvad konkreetse objekti tüübist.

Levinumad osad:
`alt` objekti küljes olev alt tekst
`affine` objekti joonistamise ajal kasutatv koordinaatide teisenduse nimi
`belongsToObject` objekti nimi, millesse objekt kuulub
`data` objekti andmete massiiv
`link` objekti küljes olev link
`type` objekti tüüp

`output_html`()
Väljastab HTML kujul klikkimise kaardi std-väljundisse.

`output_raw_gif`()
Trükib gif formaadis pildi std-väljundisse.

`output_raw_gifanim`(`temp directory`)

`temp directory:` Kataloog, kuhu kirjutatakse ajutised failid, hiljem need kustutatakse, soovitav näiteks /tmp/

Väljastab std. väljundisse GIF animatsiooni.

output_raw_png()

Trükib png formaadis pildi std-väljundisse.

output_raw_svg()

Trükib SVG formaadis pildi std-väljundisse. Et seda funktsiooni kasutada, on vaja, et kasutatav oleks Perl moodul XML::Writer.

readObjects(file)

file: SWOG koodi sisendfail

Loeb failist SWOGi käsud sisse ja moodustab pildist sisemise struktuuri.

readObjectsFromBuffer(buffer)

Loeb käsud antud puhvrist. Käsku läheb vaja, kui soovite koodi ise failist lugeda või genereerida

Lisa 3

Graafikute joonistamise skript

Järgnevalt on toodud Perl skript sisendfaili põhjal graafiku joonistamiseks. Sama skripti kasutati ka töös lk. 32 toodud graafikute loomiseks.

```
#!/usr/bin/perl -w

# loeb faili data.txt, milles rea kuju on X Y1 Y2 ja kuvab
# antud punktide graafiku

use IO;
use SWOG;

#arv regulaaravaldises:
$ARV="([0-9\.]+)";

#pildi laius ning kõrgus
$W=600;
$H=300;

SWOG::readObjectsFromBuffer("
new $W,$H
font {fonts/ARIAL.TTF} 10 :arial
");

#Loeme faili sisse:

my $input=IO::File->new("<data.txt");
if (defined $input){
     my $line;

     #joonte värv:
     my %color=(1=>"red", 2=>"blue",3=>"green");

     #joone tõmbamiseks peame meeles viimast punkti:
     my $lastX;
     my %lastYs=();

     #punktide loendur
     my $objNr=0;

     #alustame joonte lisamist, et skaalat algul ei tea,
     #peame lisama objektina, et see hiljem
     #õige skaalana lisada:
     SWOG::execute_command("defObject jooned");

     #peame meeles max väärtused:
     my $maxX,$maxY;
```

```
#loetleme väärtused, et öelda teljele mis väärtusi kuvada
my $xVals="";
my @yVals=();
   while(defined($line=$input->getline())) {
     if ($line =~ s/^$ARV//){
          my $X=$1;
          $xVals.=";" if (length($xVals)>0);
          $xVals.=$X;
          if (!defined $maxX || $maxX<$X){
               $maxX=$X;
          }
          my $group=0;
          while ($line=~ s/\s*$ARV//){
               $objNr++;
               $Y=$1;
               push(@yVals,$Y);
               if (!defined $maxY || $maxY<$Y){
                    $maxY=$Y;
               }
               $group++;
               if (defined $lastYs{$group}){
                    #tõmbame joone eelmisest
                    #punktist praegusesse
                    SWOG::execute_command("line ".
                         $lastX.",".$lastYs{$group}." ".
                         $X.",".$Y." ".$color{$group});
               }
               #kuna tahame joonistada fiks. raadiusega
               #ringi, siis selleks tuleb koordinaatide
               #süsteem süsteemi skaala viia
               #ühekordsele suurendusele

               SWOG::execute_command("point $X,$Y :h".
                         $objNr);
               SWOG::execute_command("coordsys :u".
                         $objNr);
               SWOG::execute_command("circle (%this.".
                         "h$objNr) 4 ".$color{$group});
               SWOG::execute_command("undo %this.".
                         "u$objNr");
               $lastYs{$group}=$Y;
          }
          $lastX=$X;
     }
}
SWOG::execute_command("endObject");
#y-väärtused tuleb sorteerid:
@yVals=sort @yVals;
my $yText="";
foreach(@yVals){
     $yText.=";" if (length($yText)>0);
     $yText.=$_;
}
```

```
     # mõõdud, mis jäetakse diagrammil andmete kuvamiseks
     my $dataWi=$W-50;
     my $dataHe=$H-40;

     #sõltuvalt ekstreemumpunktidest
     #paneme paika koordinaadistiku

     SWOG::readObjectsFromBuffer("
origin 20,".($H-20)."

axis ticks all below arial :telg
drawAxis 0,0 $dataWi telg 0 $maxX {$xVals}
string east $dataWi,-20 arial {kujundite arv}
rotate 0,0 90 :keere

axis ticks all above arial :vertTelg
drawAxis 0,0 $dataHe vertTelg 0 $maxY {$yText}

string center south $dataHe,20  arial {tööaeg}

undo keere

scale ".($dataWi/$maxX).",".($dataHe/-$maxY)."
object jooned
");
}
print "Content-type: image/png\n\n";
SWOG::execute_command("outputRawPng");
```

Lisa 4

Töö optilisel kettal