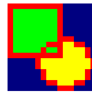
Umetna inteligenca in simbolično programiranje

# Adaptivni peresni vmesniki



Ljubljana, 23.05.2005

Marjan Pušnik, Mate Beštek, Blaž Kurent

# Kazalo

Uvod in predstavitev problema ........................................3
Potek razvoja programa....................................................3
Prepoznavanje kotov ........................................................3
Omejitve ...........................................................................6
Krivulje in zaobljeni liki ...................................................7
Zaključek ...........................................................................8
Literatura ...........................................................................9

## Uvod in predstavitev problema

Za seminarsko nalogo smo izbrali temo »adaptivni peresni vmesniki«. Motivacija za to je bila zanimiva tema ter izziv delati na področju, ki se še razvija.

Osnovna naloga pri naši seminarski nalogi je bila izdelava risarskega programa za prostoročno (pero oz. miška) risanje raznih diagramov in skic. Za uporabo smo dobili tudi tablični računalnik (tablet PC), pri katerem naj bi se v osnovi namesto miške uporabljalo pero (oz. pisalo), tako da je bila ena od poglavitnih nalog prilagoditi program temu namenu. Za zgled in ideje kakšen program naj bi izdelali, smo se na začetku oprli na delo lanske skupine pri tem predmetu.

## Potek razvoja programa

Prva naloga je bilo spoznavanje s tabličnim računalnikom, torej predvsem uporaba in način dela z peresom. Slednji se razlikuje glede na miško kar v nekaj pogledih. Prva lastnost je, da ga težko držimo dalj časa na mestu. Prednosti peresa se pokažejo pri risanju objektov, ki jih ponavadi rišemo s pisalom na papir. Lažje in lepše kot z miško narišemo krivulje, tekst ter preproste skice. To pa je dovolj velik razlog za uporabo in razvoj programa za pero.

Naslednja naloga je bil pregled sedanjega stanja na področju podobnih aplikacij. Na tem področju smo dobili veliko napotkov s strani našega mentorja. Ogledali smo si dela Takeo Agarashi-ja, ki se ukvarja s temami interakcij objektov na področju grafike. Naslednji raziskovalec je Christopher M. Bishop, ki se ukvarja med drugim z generičnimi Bayesovi modeli za prepoznavanje likov in krivulj. Eden od postopkov za prepoznavanje ki bi ga lahko uporabili, bi bil verjetnostni model, ki bi uporabljal Bayesovo statistiko. Zaradi prezapletene snovi tega avtorja, smo delo v tej smeri opustili. Videli smo tudi program, ki ga je napisal Casey Chesnut; ta za prepoznavo črk in številk uporablja nevronske mreže.

Sedaj je nastopil čas za razvoj ogrodja, na katerem bi preizkušali metode in algoritme za prepoznavanje. Za razvojno okolje smo si izbrali MS Visual Studio 2003, saj nam fakulteta omogoča brezplačno uporabo tega programa.

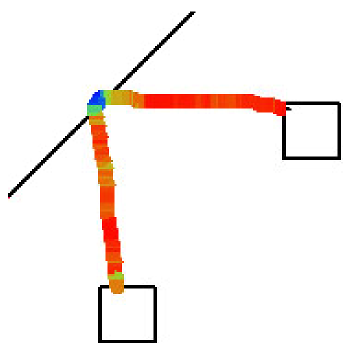
Glavna funkcionalnost tega ogrodja, ki smo ga razvijali, je bilo okno, na katerem je uporabnik lahko risal poljuben lik oz. krivuljo. Nadalje je bilo potrebno ta vnos shraniti v neke podatkovne strukture, katere smo potrebovali za nadaljno obravnavo.

## Prepoznavanje kotov

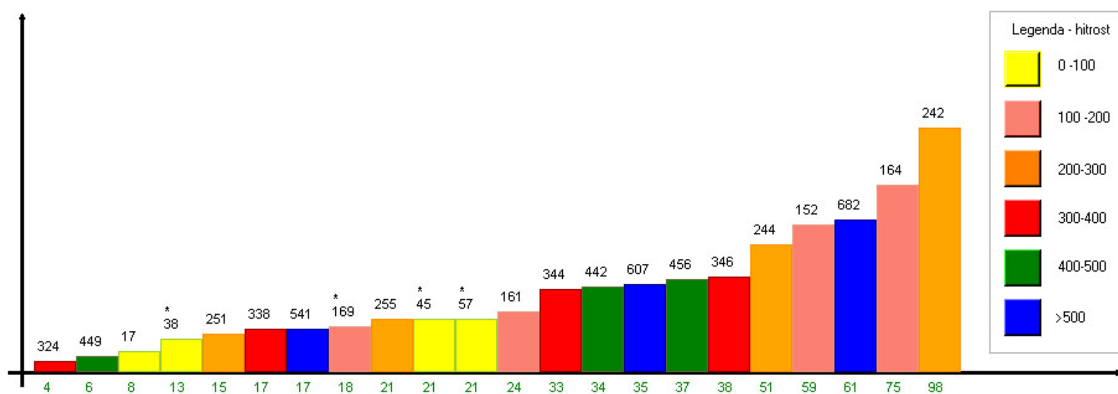
Za začetek smo si zadali cilj poiskati kot v uporabnikovem vnosu. Torej kako naj program loči ali je del (točka) krivulje (oz. lika) kot ali nekot, torej ravna črta. Za ta namen smo naredili novo okno (*set corner*), ki naj bi nam služil kot učni poligon za iskanje kota. Pri izločanju pomembnih atributov točk, smo si pomagali tudi z histogrami. Kmalu je postalo očitno, da so najbolj primerni parametri za ugotavljanje ali je točka kot ali ne, naslednji: kot med opazovano točko in sosednjima dvema (tremi, štirimi, ..) točkama, linearna regresija na intervalu sosednjih točk (idejo smo dobili pri lanski skupini) ter hitrost. Slednji atribut se je izkazal še za posebaj pomembnega, saj smo ugotovili, da programi, ki so trenutno na tržišču, tega atributa večinoma sploh ne upoštevajo.

Dobljene atribute za vsako točko smo povprečili ter tako dobili neko vrednost med 0 in 1, ki nam da neke vrste verjetnost, da je točka kot.

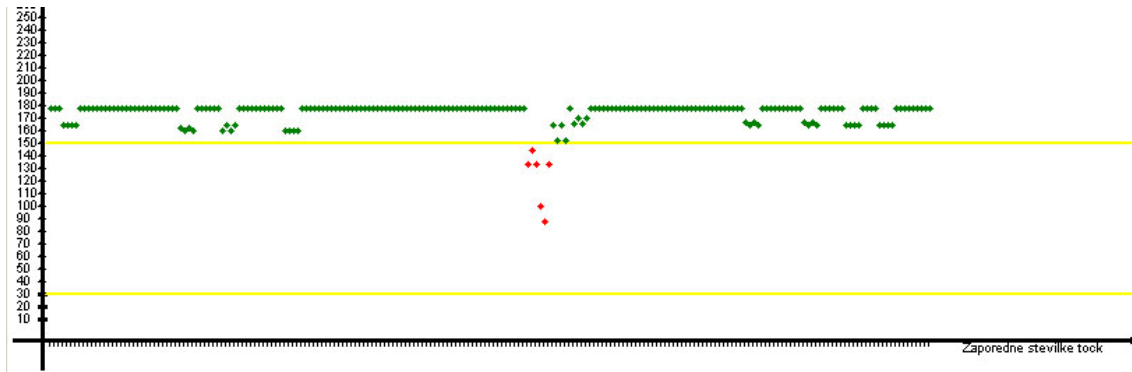
Naslednje slike prikazujejo okno *'set corner'* ter grafe za prikaz atributov teh vnosov. Uporabnik nariše potezo (kot) med obema kvadratkoma tako, da ju poveže in se vmes dodatkne še poševne črte.



*Rdeča barva označuje točke, ki imajo nizko »verjetnost« kota, modra pa tiste z visoko.*




*Histogram, kjer številke pod stolpci označujejo število točk, ki spadajo v določen interval (glede na atribut hitrosti). Števila nad stolpci označujejo povprečno histrost risanja za točke znotraj intervala. Znak '*' nad stolpcem pa pomeni, da je znotraj tega intervala prisotna točka, ki je kot v uporabniškem vnosu.*




*Atribut kota med sosednjimi točkami: na osi 'x' so po vtsti nanešene sosednje točke ene poteze (v oknu 'set corner'), na osi ' y' pa velikost kota med sosednjimi točkami (v stopinjah). Tukaj se lepo vidi, da je kot dejansko tam, kjer so koti med sosednjimi točkami približno 90°.*

Prepoznavo kotov smo naredili tako, da se sproti označujejo potencialni koti. Ta informacija naj bi bila koristna tudi za uporabnika, saj bi že med risanjem dobil občutek, kako program deluje.

Čeprav se je metoda za prepoznavanje kotov z uporabo treh združenimih atributov (hitrost, linearna regresija in kot med sosednjimi točkami) izkazala za uspešno, je bila za praktično uporabo še vedno nekoliko okorna. V primeru »lepih« kotov, kjer se v točki kota res ustavimo in narišemo pravilen kot, prepoznava deluje dobro. Čim pa narišemo potezo prehitro ali pa kot ni dovolj oster, pa naša metoda včasih odpove. Ena možnost je bila zmanjšati mero »verjetnosti« kota. V tem primeru pa zna zgoditi da najdemo kot tudi tam, kjer si ga ne bi želeli. Rešitev v temu primeru je odstranjevanje prepoznanih kotov, kjer so ti koti v neposredni bližini. Ohranimo torej samo tistega z največjo »verjetnostjo«.

Naslednja ideja pri iskanju kotov je bila, da bi naredili program specifičen za vsakega uporabnika. Pred prvo uporabo naj bi vsak narisal nekaj kotov, nakar bi to informacijo uporabili pri prepoznavanju likov v glavnem oknu. Tega na koncu nismo realizirali, saj smo ugotovili, da so si ti atributi med različnimi uporabniki dokaj podobni.

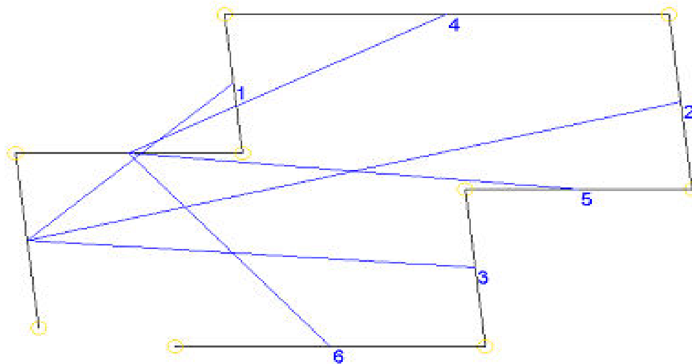
Zgoraj opisani postopki so bili uporabljeni v oknu 'test data'. Kar pa se tiče samega glavnega okna za risanje, smo uporabili malenkost drugačen pristop. Samega določanja oglov, kotov oz. oblike narisanega objekta, smo se lotili s pomočjo smernega vektoraj dveh sosednjih točk narisane poteze. Metoda deluje tako, da se ob začetku risanj, med prvima dvema narisanima točkama izračuna smerni vektor med njima. Ta vektor bomo v nadljevanju imenovali "primarni vektor smeri". Nato sem med vsako naslednjo narisano točko in točko ki je bila narisan točno pred njo izračuna smerni vektor, ki ga bomo imenovali "trenutni vektor smeri". S pomočjo trenutnega vektorja smeri in primarnega vektorja smeri se izračuna kot med njima, ki nam služi kot atribut s pomočjo katerega se odločamo ali je v dani točki ogel ali ne. Če je vrednost atributa dovolj velika v zadnji točki trenutnega vektorja, postavimo ogel in hkrati postane trenutni vektor smeri tudi primarni vektor smeri. Iz povedanega je kaj hitro razvidno, da uporabljena metod deluje dokaj dobro v primru, ko med sosednjimi narisanimi točkami dovolj velika razdalja. Zaradi slednjega je bilo potrebno poleg določanja oglov s pomočjo vektorjev vpeljati metode za izločanje oz. dodajanje oglov iz seznama prepoznanih oglov. Z uporabo opisanih postopkov nam je uspelo zadovoljivo rešiti problem prodobivanja oblike narisanih objektov, vendar kljub temu obstaja še precej možnosti za izboljšave predvsem pri določanju samih smernih vektorjev poteze.

Naš program je do tega trenutka prepoznaval kote, jih po potrebi odstranil (odvečne), ter te točke povezal z ravno črto.

**Omejitve**

Po prepoznavanju kotov je prišla na vrsto vpeljava omejitev. Ker smo imeli namen narediti program za hitro risanje diagramov, je bilo pomembno, da program proba v narisano potezo »vsiliti« določene omejitve. Le te so: vzporednost, zaokroževanje dolžine, pravilni koti, stikanje vozlišč, ...

Pri realizaciji oz. implementaciji samih omejitev smo se osredotočili predvsem na iskanje vzporednih stranic znotraj narisanega objekta in v drugi fazi na iskanje vzporednih stranic pri objektih sestavljenih iz večih potez. Postopek deluje po podobnem principu kot smo ga opisali pri prepoznavanju kotov. Povezave med sosednjmi ogli predstavimo kot smerne vektorje, ki jih nato primerjamo med sabo, tako da računamo posamezne kote med dvema smernima vektorjema. Kadar najdemo dovolj majhen kot med dvema smernima vektorjema preprosto vzamemo enega izmed niju in in ga nadomestimo z drugim ter ga zaokorožimo na isto dolžino kot jo je imel prvotno. Pri objektih sestavljenih iz več potez uporabimo isti princip, le da pri iskanju vzporednost v dodani potezi pregledamo najprej smerne vektorje objektov, ki so bili narisani pred zadnjo potezo, ki smo jo dodali nekemu že predhodno narisanemu objektu.




*Primer narisnega objekta s prikazanimi vzporednimi stranicami oz. prikazom soodvisnosti med posameznimi narisanimi stranicami.*

Uveljavljanje omejitev preko odbojnih in privlačnih sil med oglišči

Lotili smo se naloge, kjer bi za vpeljavo omejitev uporabljali sile. Točke, ki smo jih prepoznali kot kote in pa še ostale točke (točke na stičiščih krivulj, končne točke, ...), bi delovale ena na drugo po modelu sil.

Za zgled in idejo smo si ogledali program Prefuse (avtor je Jeffrey Heer). Na kratko povedano, je to orodje za vizulizacijo vseh vrst podatkov in predstavitev povezav med njimi. Podatki so lahko predstavljeni kot vozlišča, relacije med njimi pa so predstavljene z povezavami med vozlišči. Ta vozlišča se nato glede različne atribute (sile, povezanost, medsebojno oddaljenost, ...) premikajo po prostoru, na njih pa lahko vplivamo tudi mi (npr. z miško).

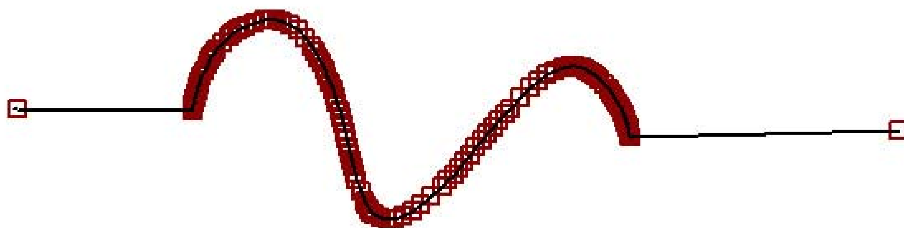
Pri sami realizaciji modela sil smo si pomagali z samo knjižnjico Prefuse, tako da smo s pomočjo orodja Java Language Conversion Assistant prevedli del, ki ga omenjena knjižnica uporablja za simuliranje delovanja sil, v izvorno kodo za naša platformo. Po uspešno izvedenem prevajanju smo nato izvedli še integracijo knjižnice v našo aplikacijo.Verjetno je je bil eden poglavitnih problemov, da se naša prizadevanja z uveljavljanje omejitev preko odbojnih in privlačnih sil med oglišči niso posrečila, ravno preveliko zanašanje na omenjeno knjižnico, saj v končni fazi slednja verjetno ni bila napisana ravno za uporabo v naši aplikaciji.

Če bi bila realizacija modela s silami uspešna, bi tako skoraj v celoti rešili problem z omejitvami. Objekti na zaslonu bi se samodejno »poravnali« na podlagi medsebojnega delovanja sil. Vendar nam žal tega postopka ni uspelo pripeljati do praktične uporabe.

**Krivulje in zaobljeni liki**

Naslednji izziv je predstavljalo prepoznavanje krivulj oz. bolje rečeno ločevanje med ravnimi črtami in krivuljami. Najprej je bilo potrebno ugotoviti ali je objekt na zaslonu sestabljen iz ravnih črt ali ne, nato pa te krivulje popraviti do boljšega izgleda. Prvi del smo rešili na dva načina. Eden je bil, da smo pri točkah prepoznanih za kote, merili vmesne kote (kot med vektorjema točk, ki potekata iz srednje točke do sosednjih dveh). Drugi način za prepoznavanje krivulj pa smo rešili s pomočjo linearne regresije. Pregledovali smo vsaj del objekta posebej in na delu, kjer je bila vrednost regresije nad neko konstanto vrednostjo, smo vedeli, da gre za krivuljo.

Ta postopek nam je omogočal tudi da se nariše lik, ki je delno sestavljen iz ravnih črt, delno pa iz krivulj. Na delu, ki je določen za krivuljo pa smo nato uporabili še metodo 'subdivision', ki je opisana spodaj.



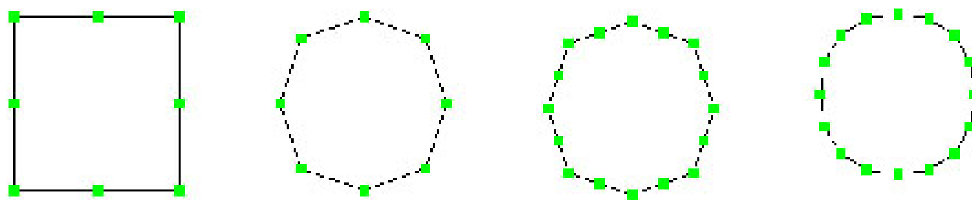
*Primer krivulje, kjer je en del prepoznan kot ravna črta, vmesen del pa kot krivulja.*

<u>Subdivision</u>

Ko smo enkrat ugotovili, da je objekt krivulja, je bilo potrebno to krivuljo še nekoliko spremeniti do boljšega izgleda. Za ta namen smo uporabili metodo *subdivision.* Delovanje te metode lahko opišemo v dveh korakih:

1) Vzamemo določeno število točk na krivulji, kjer bomo delali ta postopek. Več točk kot vzamemo bolj zaobljena bo krivulja. Nato vsak par sosednjih točk povežemo z ravno črto ter kreiramo novo točko na sredini le teh.
2) Ponovno poiščemo srednje točke na parih sosednjih točk (podobno kot zgoraj). Sedaj pregledamo vse trojke sosednjih točk. Preverimo, če srednja točka leži na daljici, ki ima za krajišči zunanji točki. V primeru, da ne, srednjo točko odstranimo.

Ta postopek lahko ponovimo večkrat. Z večjim številom iteracij dobimo vedno bolj zaobljeno krivuljo.



Prepoznavanje kroga

Prepoznavanje krogov smo rešili na zelo preprost način. Za vsak objekt, ki je prepoznan kot krivulja, se preveri še naslednje: izačuna se »povprečna« točka cele krivulje. Tako dobimo neko točko, ki je pri liku podobnem krogu, približno na sredini le tega. Nato izračunamo povprečno oddaljenost vseh točk krivulje od te točke. Če le ta ne presega določene vrednosti, lahko predpostavimo da gre za krog. Na podoben način bi lahko prepoznali tudi elipso, vendar pa tega nismo implementirali.

## Zaključek

Čeprav naši aplikaciji še malo manjka do točke, ko bi lahko zaživela v praksi, smo z opravljenim delom zadovoljni. Mogoče smo nekoliko preveč poudarka dali raziskovalnemu delu naloge, tako da nam je na koncu zmanjkalo malo časa za izgradnjo takega uporabniškega vmesnika, kakršnega smo si zamislili.

Uspešno smo realizirali prepoznavanje kotov, osnovno prepoznavanje likov in krivulj. Med stvarmi, ki bi jih bilo potrebno še dodelati, je integracija vseh metod za prepoznavo v osrednje okno, dokončanje začetega dela s silami in robustnejši uporabniški vmesnik.

**Literatura**


- Nikola Pavešić: Razpoznavanje vzorcev
- C. M. Bishop: Neural networks for pattern recognition
- Rob Jerrett, Philip Su: Building tablet PC applications
- http://research.microsoft.com/~cmbishop/
- http://zeus.fri.uni-lj.si/~aleks/
- http://www-ui.is.s.u-tokyo.ac.jp/~takeo/java/ pegasus/pegasus-e.html
- http://www.subdivision.org/subdivision/demos/tutorial.jsp
- http://jheer.org/publications/2004-Heer-prefuse-Masters.pdf


Osnovna razdelitev dela:

**Marjan**: izdelava uporabniškega vmesnika, implementacija postopkov za prepoznavanje kotov 'smerni vektorji', implementacija omejitev za iskanje vzporednosti

**Mate**: pomoč Marjanu pri izdelavi uporabniškega vmesnika, delo z silami na podlagi projekta 'Prefuse'

**Blaž**: implementacija postopkov za prepoznavanje kotov, delno histrogrami, krivulje,subdivision