KAUNAS UNIVERSITY OF TECHNOLOGY

INFORMATICS FACULTY

DEPARTMENT OF COMPUTER SYSTEMS

Dynamic Scene Analysis and Beautification for Hand-drawn Sketches

Master thesis

Done by:

Milda Gusaite

Supervisors:

prof. dr. E. Kazanavičius

prof. dr. T. Barkowsky

Kaunas, 2006

KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

KOMPIUTERIŲ KATEDRA

Milda Gusaitė

Ranka pieštų eskizų dinaminė analizė ir gražinimas

Magistro darbas

Darbo vadovai

prof. dr. E. Kazanavičius

prof. dr. T. Barkowsky

Kaunas, 2006

KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

KOMPIUTERIŲ KATEDRA

Milda Gusaitė

Ranka pieštų eskizų dinaminė analizė ir gražinimas

Magistro darbas

Kalbos konsultantė		Vadovai	
	Lietuvių k. katedros lekt. I. Mickienė		prof. dr. E. Kazanavičius prof. dr. T. Barkowsky
2006-05		2006-05	
Recenzentas		Atliko	
	doc. E. Toldinas		IFM-0/1 gr. stud. Milda Gusaitė
2006-05		2006-05-25	

Kaunas, 2006

Ranka pieštų eskizų dinaminė analizė ir gražinimas

Santrauka lietuvių kalba

Įvadas

Eskizų piešimas yra svarbi kūrybinio proceso dalis, kuri taip pat naudojama projektavimo bei inžinerijos disciplinose, tokiose kaip: mechanikos ir civilinė inžinerija, grafinis dizainas, architektūra ir kt. Daugelis projektuotojų vis dar pradeda projektuoti piešdami savo idėjų eskizus ant popieriaus ir tik po to juos įkelia į kompiuterį. Tai padeda modeliuotojams labai greitai, natūraliai reikšti atsirandančias idėjas ir greitina vizualinių problemų sprendimą. Pirminis piešimas yra paplitęs ir svarbus kūrimo procese, nes jis skatina kūrybos laisvę. Eskizas – tai neišbaigtas modelio ir funkcinių galimybių pristatymas, kuris iš esmės padeda suvokti perteikiamą idėją. Be to, piešdamas eskizus projektuotojas tarsi sąveikauja su savo eskizais ir detaliau nagrinėja alternatyvias problemos sprendimo galimybes. Daugelis modeliuotojų mąsto vizualiai ir įpratę viską įsivaizduoti grafiškai, todėl piešiant yra tiriami alternatyvūs sprendimai ir skatinama idėjų plėtra. Šią projektavimo dalį, kurioje yra netikslumų, idėjų formalizavimas, greitas alternatyvų tyrinėjimas, inžinierius vis dar atlieka pieštuku popieriuje.

Nors eskizų piešimas popieriuje yra įprastas ir mėgiamas modeliuotojų, šis būdas turi apribojimų. Eskizus galima lengvai piešti popieriuje, tačiau kyla pagrindinis trūkumas, kai tik juos reikia taisyti ar tobulinti. Jei projektuotojas nori kažką eskize keisti, dažniausiai jis turi iš esmės perpiešti eskizą kitame lape. Šiuo atveju kompiuteriniai įrankiai turi daug privalumų. Pirmiausiai, modeliuotojui nereikia kelis kartus perpiešti eskizą popieriuje ir tik po to įkelti modelį į kompiuterį. Antra, kaip buvo paminėta anksčiau, modeliuotojui daug lengviau koreguoti darbą kompiuteryje nei popieriuje. Dar daugiau, kompiuteris gali tapti asistentu, kuris piešiant eskizus siūlo modelio taisymo variantus.

Daugelis inžinierių, architektų ir kitų profesionalių dizainerių projektuojant naudoja pagalbines kompiuterinio modeliavimo sistemas (CAD – computer-aided design systems). Nors CAD sistemos turi didelį kompiuterinių įrankių pasirinkimą, bet skirtumas tarp įprasto eskizo piešimo ranka ant popieriaus ir jo piešimo kompiuteriu yra vis dar per didelis. Inžinieriai dažniausiai kuria pieštuku popieriuje, dažnai neįkeldami modelių į kompiuterį tol, kol šie nebūna beveik baigti. Taip yra dėl CAD įrankių teikiamo nenatūralumo piešimo jausmo, kuris slopina projektavimo proceso efektyvumą. Viena priežasčių, kodėl CAD įrankiai nepopuliarūs, yra ta, kad darbas su jais sudėtingas ir neefektyvus, neatstoja realios modeliavimo aplinkos, kur galima būtų piešti ranka. Be to, šios priemonės yra neefektyvios, kai eskizus piešti reikia didesniu tikslumu ir reikia atlikti daugiau sudėtingų veiksmų tikslui pasiekti. Nors CAD įrankiai tampa pakankamai modernūs ir daugkartiniai, tačiau jie dažniausiai yra naudojami tik paskutiniuose projektavimo etapuose.

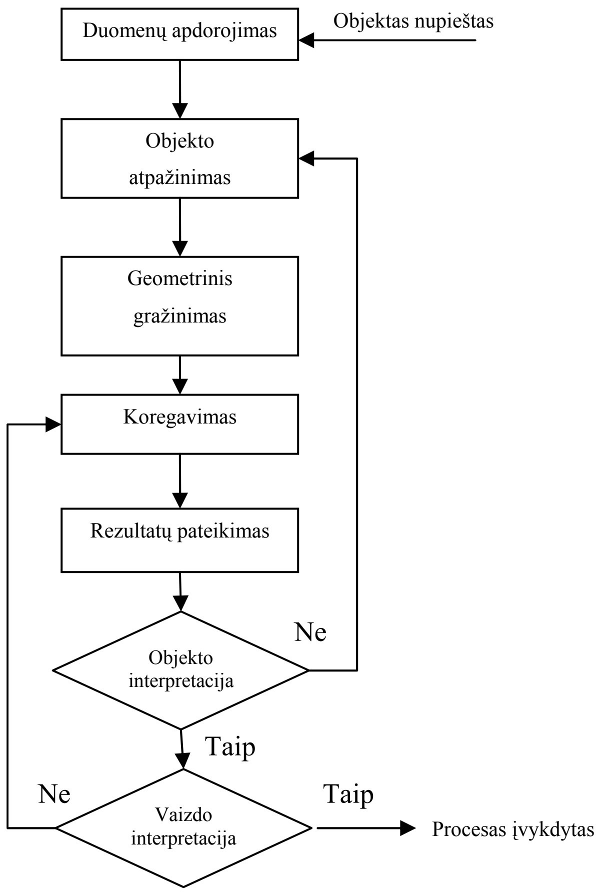
Šiuo darbu siekiama pradėti kompiuterį naudoti ankstesniuose modeliavimo etapuose, realiu laiku pertvarkant ranka pieštus eskizus, kad šie būtų tikslūs, aiškūs ir tvarkingi. Darbas apima kokybinę vaizdų analizę, geometrinį gražinimą, kokybinį erdvinį įvertinimą ir konceptualius kaimynystės metodus.

Darbo tikslas

Darbo tikslas yra tobulinti eskizų piešimo procesą, kuris gali būti padarytas popieriaus skiautėje, ir eskizų darymui modeliuoti sistemos prototipą, siekiant, kad kompiuteris būtų naudojamas ankstesniuose projektavimo etapuose. Tai labai svarbu, kadangi projektavime eskizai vaidina pagrindinį vaidmenį konceptualumo fazėje. Šiame darbe automatiškai, dizaineriui dar tebepiešiant, tiriami piešiami objektai ir, atsižvelgus į dizainerio daromus veiksmus, iš karto yra gražinamas eskizas. Yra labai svarbu, kad modeliuotojai galėtų greitai ir paprastai, naudodami pieštuką, škicuoti viską, ką tik nori. Tokia sistema projektuotojams suteiktų natūralų eskizo piešimo jausmą. Eskizų paišymo sistema turi leisti laisvai piešti ranka, atpažinti piešiamą modelį ir jį atitinkamai koreguoti pagal dizainerio ketinimus. Taip yra siekiama sukurti asistentą modeliuotojui, kadangi eskizai yra ne tik vienas iš modelio tobulinimo būdų, bet ir svarbus modeliuotojų tarpusavio bendravimo būdas.

Darbo apimtis

Darbe yra kuriamas sistemos algoritmas, kuris realiu laiku sąveikauja su vartotoju, piešiančiu eskizą (1 pav).



1 pav. Sistemos funkcionalumo diagrama

Kai objekto piešimas yra baigtas, sistema atpažįsta nupieštą objektą, analizuoja ir identifikuoja jį. Taip pat po atpažinimo proceso yra atliekamas objekto geometrinis gražinimas Tai apima geometrinių taisyklių aptikimą ir įvedimą, defektų koregavimą (linijų tiesinimą, teisingų kampų formavimą, linijų sujungimus ir kt.). Po to eskizas yra koreguojamas, atsižvelgiant į objektus, kurie buvo nupiešti anksčiau, ir konceptualius kaimyniškus santykius. Sistema, pateikdama objektų interpretacijas, sąveikauja su modeliuotoju ir suteikia galimybę taisyti klaidingą sistemos interpretavimą. Jei interpretacija teisinga, vartotojas toliau piešia eskizą, priešingu atveju vartotojas sistemai turi nurodyti, kad interpretacija yra neteisinga. Interpretavimo procesas apima visų galimų interpretacijų sąrašo pagal konceptualius kaimyniškus santykius sudarymą. Taigi, atsiradus klaidingam interpretavimui, vartotojui yra pateikiamos interpretacijos alternatyvos.

Galutinis šio darbo rezultatas yra algoritmas, sukurtas dinaminių vaizdų analizei ir gražinimui, bei jo dalinė realizacija demonstracinėje programoje.

Analizė

Šio darbo analizės dalyje nagrinėjami aspektai reikalingi sistemos modeliui sukurti.

Pradžioje nagrinėjamas pagrindimas diagramomis ir jo efektyvumas. Diagramomis perduodama informacija yra savaiminė ir lengviau suprantama nei ta pati informacija perduodama teksto pavidalu. Viena iš diagramos savybių yra ta, kad diagrama perduoda ne tik specifinę informaciją apie objektą pavaizduotą joje, bet taip pat ir perteikia objekto poziciją ir santykius jį supančių kitų objektų atžvilgiu. Kita savybė yra ta, kad diagramoje vaizduojami objektai skirtingiems žmonėms gali reikšti skirtingus dalykus, taip sukeliant neteisingas išvadas. Vadinasi diagramos ne tik turi savo kūrėjo interpretaciją, bet taip pat ir kiekvieno žiūrovo interpretaciją, tuo tarpu kai kitos informacijos formos turi tik vieną tiesioginę interpretaciją. Visų šių savybių dėka, pagrindimas ir gebėjimas pažinti diagramų pagalba yra labai mokslininkams patraukli sritis.

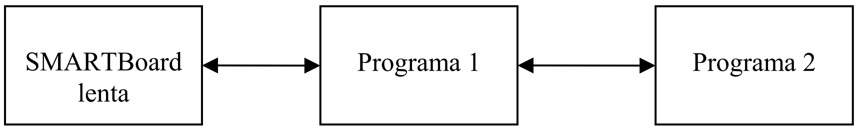
Toliau analizėje supažindinama su kokybiniu erdviniu pagrindimu ir erdviniais santykiais. Erdvinis pagrindimas ir suvokimas yra sritis nagrinėjanti konceptualią erdvę, kuri susideda iš tokių erdvinių pateikimų kaip tipologija, orientacija, forma, dydis ir atstumas. Erdvinis pagrindimas neatsiejamas nuo erdvinių santykių, kurie teikia informaciją apie objektą neatsižvelgiant į jo konkrečią geometriją. Erdvinius santykius mokslininkai pagrinde skirsto į tris grupes: kryptis, atstumus ir topologinius santykius. Krypties santykiai tarp dviejų objektų nusakomi kardinaliniais terminais („pietuose“, „rytuose“, „pietvakariuose“ ir t.t.) arba kasdieniniame gyvenime naudojamais terminais („priešais“, „dešinėje“ ir pan.) kito objekto atžvilgiu. Atstumai tarp dviejų objektų nusakomi abstrakčiais terminais, tokiais kaip „šalia“, „toli“ ir pan. Dviejų objektų topologiniai santykiai apibrėžia labiau konkrečius santykius, tokius kaip, kad vienas objektas dengia kitą arba vienas objektas yra kito objekto viduje ir pan.

Tuomet įvedamas konceptualių kaimyninių santykių terminas, naudojamas objekto pozicijos koregavimui. Konceptualūs kaimyniniai santykiai – tai ryšys tarp dviejų erdvinių santykių, kurie vienas nuo kito skiriasi minimaliu vienu pakitimu. Pavyzdžiui, atstumuose santykis „šalia“ yra kaimyninis su santykiu „toli“. Tuo tarpu, santykis „šalia“ negali būti kaimyninis su santykiu „labai toli“, nes tarp šių santykių yra per didelis perėjimas ir daugiau nei vienas pakitimas: „šalia“ ⇔ „toli“ ⇔ „labai toli“.

Galiausiai analizė užbaigiama aptariant ir paaiškinant objektų atpažinimą taikant statistiką ir Hu momentus.

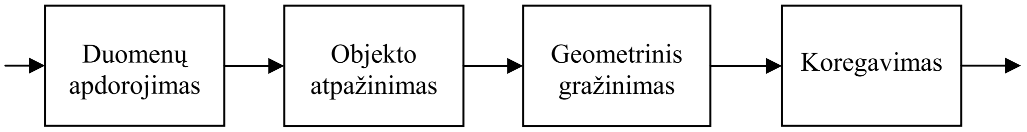
Algoritmo sudarymas

Šio darbu siekiama sukurti ranka pieštų eskizų dinaminės vaizdų analizės ir gražinimo sistemos modelį. Pagrindinė sistemos koncepcija remiasi sistemos realaus laiko sąveika su vartotoju tuo metu kai jis piešia eskizą.



2 pav. Sistemos modelio struktūra

Sistemos modelio struktūra, pateikta 2 paveiksle, susideda iš SMARTBoard lentos, tarpininkės programos (Programa 1) ir objekto atpažinimo ir gražinimo programos (Programa 2). Šios sistemos pagalba vartotojas piešia eskizus ant sąveikaujančios SMARTBoard lentos, kuri suteikia natūralaus eskizų piešimo jausmą. Tarpininkės programos užduotis yra aptikti objekto nupiešimo įvykį ir perduoti lentoje esantį eskizą objekto atpažinimo ir gražinimo programai. Šiai programai atpažinus ir pagražinus objektą, pakoreguotas eskizas yra perduodamas programai tarpininkei, kuri savo ruožtu pateikia jį vartotojui SMARTBoard lentoje. Kadangi šis darbas sutelktas ties dinamine scenų analize ir gražinimu, dėmesys sutelkiamas į Programa 2 algoritmo sudarymą.



3 pav. Algoritmo struktūra

Dinaminės vaizdų analizės ir gražinimo algoritmui siūloma struktūra (3 pav.), susidedanti iš keturių dalių, kurios toliau aptariamos detaliau.

Duomenų apdorojimas

Ši algoritmo dalis paruošia iš tarpininkės programos gautus duomenis kitiems algoritmo etapams. Gautas eskizo paveikslas yra konvertuojamas į dvejetainio tipo paveikslą, kur 0 simbolizuoja baltą tašką, o 1 reiškia juodą tašką. Toliau yra išskiriamas naujai nupieštas objektas, aptinkant pakeitimus eskize atliktus nuo praėjusio karto. Tuomet, yra nustatoma paveikslo reikšmingumo sritis, kuri turima omeny kaip stačiakampio formos sritis, kuri apima objektą ir tam tikrą kiekį baltų taškų. Iškirpus reikšmingumo sritį iš paveikslo, ji perduodama atpažinimo procesui.

Objekto atpažinimas

Objekto atpažinimas atliekamas, norint priskirti objektą tam tikrai klasei, kad vėliau galima būtų tą objektą pagražinti. Šis algoritmo žingsnis atliekamas remiantis Hu momentais, kurie yra nekintantys mastelio ir pozicijos keitimo atžvilgiu. Objektas vienu metu gali priklausyti tik vienai objektų grupei. Šis darbas sutelkiamas ties elementarių apskritimų ir kvadratų atpažinimu. Prieš sistemai pradedant darbą, ši algoritmo dalis turi būti apmokyta vartotojo pieštų objektų rinkiniu, kadangi yra manoma, kad žmonės piešia skirtingu būdu. Baigus apmokymą, rezultatai yra saugomi programos duomenų bazėje kaip objektų ir jiems būdingųjų Hu momentų vektorių rinkinys. Būdingasis Hu momentų vektorius, apibūdinantis tam tikrą figūrą, susideda iš septynių Hu momentų.

Kai gaunamas naujai nupiešto objekto reikšmingumo sritį iš aukščiau esančio algoritmo etapo, atpažinimo procesas apskaičiuoja tos srities būdingąjį vektorių:

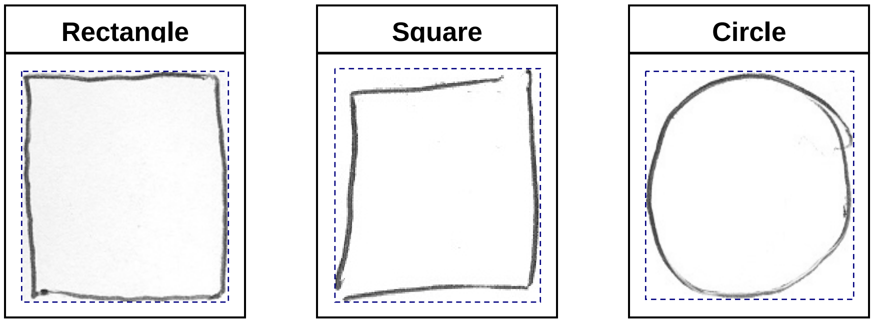
$$HU_A = \{ I_1 \quad I_2 \quad I_3 \quad I_4 \quad I_5 \quad I_6 \quad I_7 \} \tag{1}$$

kur HU_A yra A objekto būdingasis Hu momentų vektorius, o I_k yra atitinkamas Hu momentas, kai $k = 1, 2, ..., 7$.

Kai objekto būdingasis Hu momentų vektorius apskaičiuotas, pradedamas atpažinimo procesas lyginant atstumus tarp šio vektoriaus ir duomenų bazėje esančių objektų vektorių. Trumpiausias atstumas nulemia klasę, kuriai nupieštas objektas priklauso.

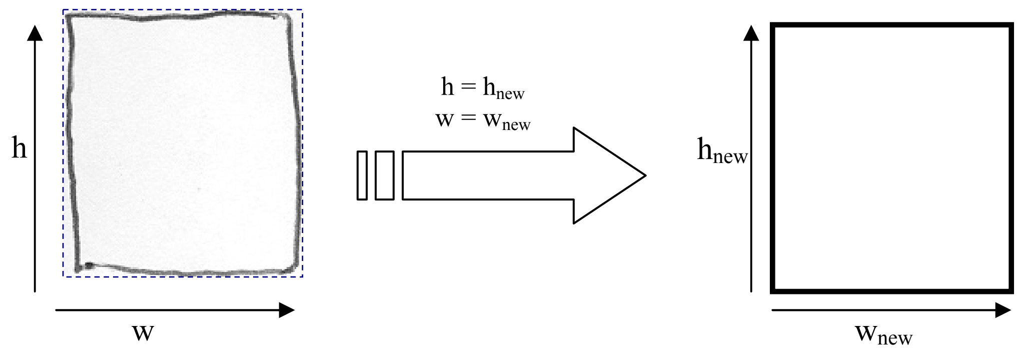
Geometrinis gražinimas

Kai baigiamas atpažinimo procesas, gauta informacija panaudojama objekto gražinime, norint atlikti geometrinius pataisymus ir sukurti tvarkingą ir aiškią ranka piešto objekto versiją. Gražinimo procesas gauna pradinius duomenis objekto pavadinimo ir jo reikšmingumo srities pavidalu (4 pav.).



4 pav. Gražinimo proceso pradiniai duomenys

Skirtingo sudėtingumo formos objektai reikalauja atitinkamo sudėtingumo gražinimo. Elementarių objektų, tokių kaip kvadratas, stačiakampis, apskritimas ir kt., gražinimas, nereikalauja sudėtingų skaičiavimų ir veiksmų. Pavyzdžiui, stačiakampio gražinimo procesas (5 pav.) yra paprastas ir reikalauja tik objekto reikšmingumo srities matmenų: aukščio ir pločio. Pagal šiuos matmenis yra sukuriamas tvarkingas idealus stačiakampis.



5 pav. Stačiakampio gražinimas

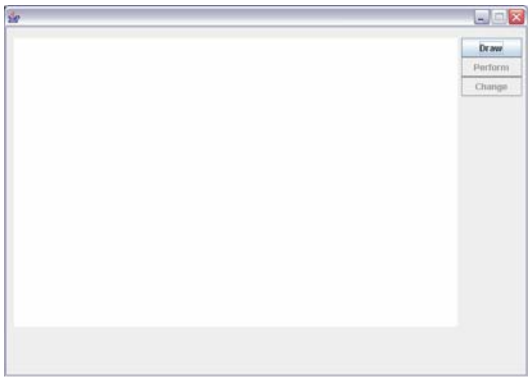
Jei nustatytas objektas priklauso sudėtingų objektų klasei, geometrinį gražinimą rekomenduojama atlikti remiantis H. Hse ir A. R. Newton (Hse ir Newton, 2005) pasiūlytu metodu. Šis metodas naudoja objektų segmentaciją į elementariąsias dalis: tiesias linijas ir elipsines arkas. Sudarius objekto segmentacijos modelį, jis yra naudojamas figūros geometrinių duomenų, tokių kaip kampai, kraštinių ilgis ir kt., nustatymui ir tvarkingo idealaus objekto atkūrimui.

Koregavimas

Algoritmo koregavimo dalis siekia patalpinti atpažintą ir pagražintą objektą toje pozicijoje, kurioje vartotojas ketino tą objektą nupiešti. Objekto patalpinimo vieta nustatoma remiantis erdviniais santykiais su prieš tai nupieštais objektais. Šis algoritmas naudoja visus tris erdvinių santykių tipus: kryptis, atstumus ir topologinius santykius. Yra sudaromas sąrašas santykių tarp einamojo objekto ir penkių paskutinių objektų. Neseni objektai yra naudojami, o seni objektai yra atmetami, nes manoma, kad tarpusavyje susijusius objektus žmonės paprastai piešia beveik vieną paskui kitą. Sudarius sąrašą nustatomas artimiausias objektas ir pagal jį atliekamas naujai nupiešto objekto pozicijos koregavimas, taikant erdvinius tų objektų tarpusavio santykius. Atlikus objekto pozicijos eskize koregavimą, rezultatas yra pateikiamas vartotojui.

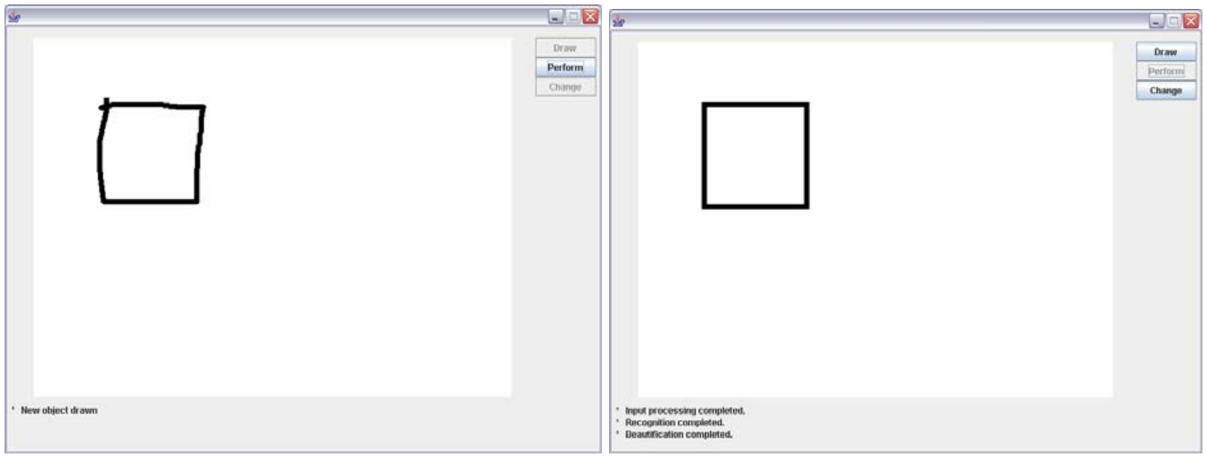
Realizacija

Siekiant patikrinti algoritmo modelį, dalinai realizuojamas algoritmas be koregavimo dalies. Realizacija atlikta Java programavimo kalba, nes SMARTBoard lenta yra suderinama su Java programavimo kalba ir ši dalinė realizacija gali būti panaudota tolimesniam sistemos kūrimui.



6 pav. Pradinis programos langas

Realizacijos programos langas imituoja SMARTBoard lentos vaizdą (6 pav.). Paspaudus mygtuką *Draw* yra parodomas vaizdas (7 pav. kairėje), vartotojui nupiešus objektą lentoje. Algoritmas paleidžiamas vykdyti, paspaudus mygtuką *Perform*, o algoritmo rezultatas iš kart pateikiamas programos lange (7 pav. dešinėje). Šiuo atveju matome, kad vartotojas norėjo nupiešti kvadratą, o sistemą atitinkamai jį atpažino, pagražino ir pateikė vartotojui.

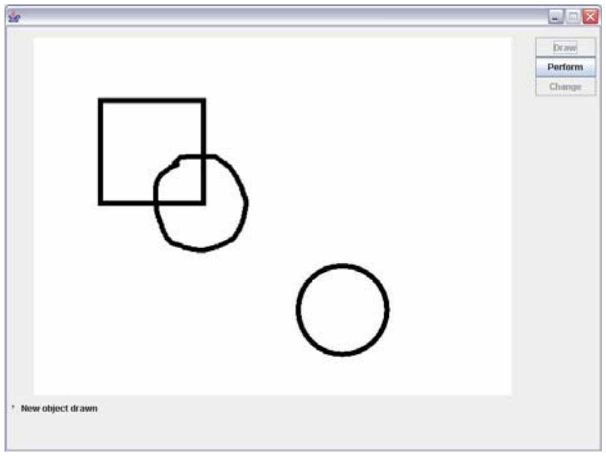


```
* New object drawn
```

```
* Input processing completed.
* Recognition completed.
* Beautification completed.
```

7 pav. Nupieštas kvadratas (kairėje), atpažintas ir pagražintas kvadratas (dešinėje)

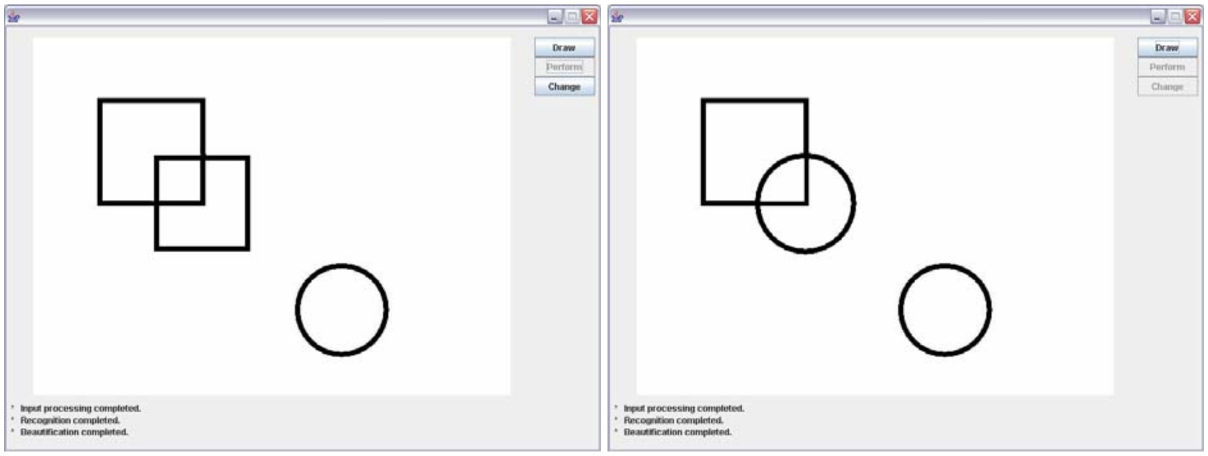
Taip pat yra pateikiamas atvejis, kai vartotojo nupieštas objektas yra neteisingai interpretuojamas. Tai atsitinka, kai nupieštas objektas yra deformuotas ir turi per mažai jo klasei būdingų savybių. Pavyzdžiui, pav. pavaizduotas eskizas, kuriame nupieštas naujas deformuotas apskritimas (8 pav.).



* New object drawn

8 pav. Nupieštas deformuotas apskritimas

Atpažinimo procedūra yra tokia pati, kaip paminėta ankstesniame pavyzdyje. Algoritmo interpretuotas objektas yra pagražinamas ir pateikiamas vartotojui (9 pav. kairėje).



* Input processing completed.
* Recognition completed.
* Beautification completed.

* Input processing completed.
* Recognition completed.
* Beautification completed.

9 pav. Neteisinga interpretacija (kairėje) ir teisinga interpretacija (dešinėje)

9 paveikslo kairėje pusėje pateikiama pirminė objekto interpretacija, kuri akivaizdžiai yra neteisinga. Tokiu atveju, spaudžiamas mygtukas *Change* ir algoritmas pateikia artimiausią pagal tikimybę kitą interpretaciją (9 pav. dešinėje). Šį kartą, interpretacija yra teisinga. Algoritmas objekto interpretacijų turi tiek, kiek objektų yra sistemos duomenų bazėje, ir rūšiuoja jas pagal tikimybę, kuri nustatoma pagal panašumą į sistemos objektus.

Atlikta algoritmo dalinė realizacija patvirtino sukurto algoritmo veiksmingumą.

Išvados

Atlikus darbą paaiškėjo, kad statistiniai metodai efektyvūs objekto atpažinimo sistemose, bet efektyvumo lygis labai priklauso nuo objektų sudėtingumo ir kiekio saugomo sistemos duomenų bazėje. Kuo sudėtingesni objektai ir kuo didesnis objektų rinkinys, tuo sudėtingesni statistiniai metodai turėtų būti naudojami sistemoje.

Elementarioms geometrinėms figūroms atpažinti pakanka algoritmo paremto pastoviais Hu momentais, tačiau naudojamas sudėtingesnės formos objektams atpažinti šis metodas nepasiekia labai gerų rezultatų.

Sudėtingų figūrų atpažinimui rekomenduojama naudoti sudėtinius Zernike momentus. Tokiu atveju, reikia patartina naudoti optimizuotus Zernike momentų apskaičiavimo metodus, nes Zernike momentų eilei didėjant, didėja ir jų apskaičiavimo sudėtingumas. Yra nustatyta, kad pakanka apsiriboti momentais nuo 2-os iki 8-os eilės, kadangi eilės padidinimas pastebimai pagerina efektyvumą tik apie 1%. Taip pat, kuo didesnės eilės momentai naudojami algoritmuose, tuo labiau jie yra jautrūs triukšmo efektui.

Kitas variantas sudėtingų objektų atpažinimui būtų panaudoti metodą, apjungiantį objekto karkaso Hu momentus ir Furje transformacijos savybes. Šio metodo sudėtingiausia dalis būtų objekto karkaso nustatymas, kurį galima būtų atlikti Voronojaus diagramų pagalba ar kitais metodais. Prieš pasirenkant vieną iš šių metodų, reikėtų įvertinti kiekvieno metodo reikalaujamų skaičiavimo sudėtingumo ir jo teikiamą efektyvumo santykį ir pasirinkti optimaliausią.

Gražinimo procesas įvairaus sudėtingumo objektams yra skirtingas. Elementarių figūrų gražinimas atliekamas nesudėtingais veiksmais, remiantis fundamentaliais figūros duomenimis. Tuo tarpu, sudėtingesnių objektų gražinimas reikalauja atlikti segmentaciją prieš pradedant geometrinį objekto koregavimą.

Objekto pozicijos eskize koregavimas pagal jo erdvinius santykius su jį supančiais objektais yra efektyviausias ir rekomenduojamas tik naudojant metodus, kure apima visus tris erdvinių santykių tipus.

Literatūra

Hse H. H.; Newton A. R. (2005). *Recognition and Beautification of Multi-Stroke Symbols in Digital Ink*. Computers & Graphics, 2005

Preface

This document reports on the thesis entitled “Dynamic Scene Analysis and Beautification for Hand-drawn Sketches”, carried out at the Department of Computer Systems, Informatics Faculty, Kaunas University of Technology and the Department of Cognitive Systems, Bremen University in partial fulfillment of the requirements for a Masters degree in Computer Science.

This thesis is organized into five chapters. The first chapter presents an introduction, motivation and the scope of the project. Chapter 2 focuses on problem analysis: qualitative scene analysis, geometric beautification, qualitative spatial reasoning and conceptual neighborhoods. The algorithm of dynamic scene analysis and beautification for hand-drawn sketches is described in Chapter 3. Chapter 4 discusses the implementation of the designed algorithm. Finally, there is provided a conclusion of the thesis.

Contents

List of Figures

Chapter 1

Introduction

Sketching is an important part of any creativity process and is used in the design disciplines, concerned with making physical form: mechanical and civil engineering, graphic design, and architecture and physical planning. Almost all designers still begin the design process by sketching their ideas before transferring them to a computer. This helps designers to express nascent ideas fast, naturally and to speed up visual problem solving. Moreover, the importance of sketching in design has been recognized emphasizing that initial drawing allows creative freedom. The sketches represent a rough semblance and functionality of the system and can be essential in understanding the reasoning behind a design. Furthermore, sketching activity lets designers to interact with their sketches, which results in a prospect for all alternative possibilities. Almost all designers are visually orientated and are basically used to think graphically, so the sketching process encourages designers to develop ideas and explore alternative design solutions in their minds. This important part of design, which supports imprecision and incremental formalization of ideas as well as rapid exploration of alternatives, is still performed by engineers with the help of paper and pencil.

Despite praxis and fondness of natural interface provided by paper, sketching on paper has its own limitations. Although a designer can easily draw a sketch on paper, the main disadvantage lies with the editing and improving of the design which is problematic. If the designer wants to make changes in the sketch, usually he has to take another sheet of paper and basically redraw the entire sketch. In contrast to a paper, computer tools potentially have many advantages. Firstly, the designer would not be forced to express his ideas twice by initially sketching on paper and then transferring the design to the computer. Secondly, as mentioned before, it would be easier for the designer to edit the design on the computer. Moreover, the computer could perform the role of an assistant in the process of sketching by making/offering adjustments in the sketch made by the designer.

Most engineers, architects and other design professionals use computer-aided design systems (CAD) in their design activities. Although CAD systems use a wide range of computer-based tools, the compromise between the ease of drawing sketches on paper and the power of representing it on computer is too great. The engineers design mostly with pencil and paper, rarely transferring their designs to the computer often until they are rather complete, because of an unnatural feel of CAD tools and simulation software inhibits the design process. One of the reasons for the delayed handling of CAD tools is that computers are too difficult to use, inefficient in working with drawings and also they lack integration into a real design environment where free hand drawing is used. Furthermore, these tools are ineffective for the sketching process as they require more precision and effort than it is needed for conceptual design. Although CAD tools are becoming suitably sophisticated and multiplex, the main usage of these tools is still focused in the final stages of the design.

This thesis is an attempt to involve computer in the earlier stage of design process. The main idea is concentrated on the problem of transforming hand-drawn sketches into neat drawings during the sketching process. The work will include qualitative scene analysis, geometric beautification, qualitative spatial reasoning and methods of conceptual neighborhoods.

1.1 Motivation of the thesis

This thesis aims at supporting the process of sketching that could be done on a scrap sheet and also to design a system prototype for sketching, thus involving computer in early stages of design processes. This is a relevant issue, because in the areas of design, sketches play a key role in the conceptual phases. This thesis explores the idea of automatically invoking dynamic scene analysis and beautifying the sketch based on the designer’s drawing actions. It is important that designers would be able to sketch everything they want, quickly and easily using a pencil. With such a system, the designers would have a natural common feel of the sketching process. The sketching support system should provide freehand drawing, recognize design intent from drawing and make corrections according to the designer’s intentions and the context of the drawing. We aim at providing contextual assistance to designers in the process of sketching, because it is not only means of developing design ideas but also an important mean of communication amongst designers.

1.2 Scope of the thesis

This thesis aims at designing a prototype, which in real-time interacts with the user in the process of sketching as shown in Figure 1.

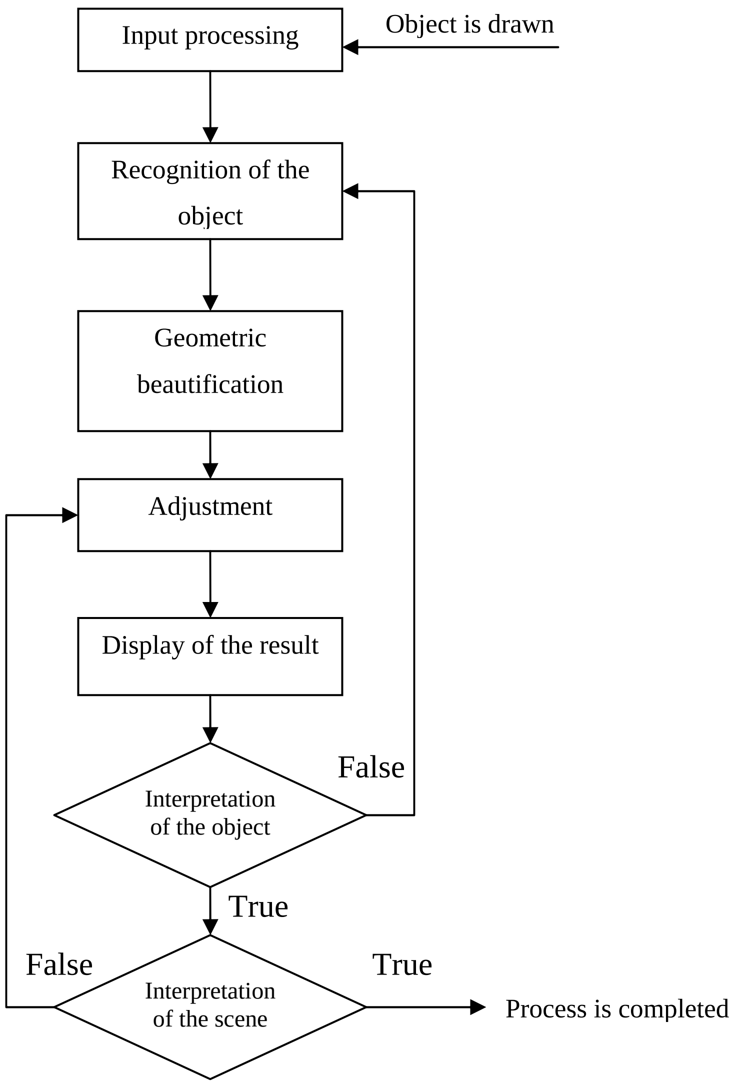


Figure 1: Flow diagram of system functionality

The system recognizes the drawn object as soon as the drawing process of that object is complete. The object is analyzed and identified. After the object recognition process, the system starts performing a geometric beautification of the object, if needed. The process of geometric beautification encompasses the detection, the imposition of geometric regularities and the required correction of defects (such as straightening lines, forming correct angles, connecting lines etc). Thereafter, the adjustments of the sketch are performed according to the objects, which have been sketched before the current one, in the context of qualitative scene analysis and methods of conceptual neighborhood. The system interacts with the designer by displaying the interpretation of objects and giving the means to fixing any misinterpretations made by system. If the interpretation is correct the user keeps on sketching, otherwise the user has to indicate to the system that the interpretation of the system is incorrect. The process of interpretation involves the making a list of possible interpretations by methods of conceptual neighborhoods. Hence, if a misinterpretation occurs, the user will be provided with alternative possibilities of interpretation.

The end result of this thesis is an algorithm designed and developed for dynamic scene analysis and beautification, and a partial implementation of it on a demonstrative system.

Chapter 2

Problem Analysis

This chapter explores and dissects the questions to be considered, solved, or answered in this thesis. As this thesis is concentrated on scene analysis, the issues of reasoning with diagrams are overviewed. Moreover, relevant background of qualitative spatial reasoning and conceptual neighborhood is revealed here. These are the main issues identified to be necessarily explained in this chapter.

2.1 Reasoning with diagrams

Before exploring the field of reasoning with diagrams, the meaning of the basic term “diagram” must be clarified. Although this term can mean different things for different people, the term “diagram” is used here with the meaning of a drawing that uses geometrical elements in order to abstractly represent a case. The situation reasoning can be done in four ways of deduction: diagram-to-sentence, sentence-to-sentence, sentence-to-diagram and diagram-to-diagram (Furnas, 1992). Usually reasoning systems are heterogeneous and use the first three ways of deduction. However, work on diagrammatic reasoning in order to express logical and set-theoretical properties and integrate it into reasoning systems has become critical and highly relevant (Gurr, 1999). As this thesis is concerned with dynamic scene analysis and beautification, an overview on issues of reasoning using graphical diagrammatic information is presented. In this work, the graphical representations have been classified into three classes, namely: static diagrams, animation and virtual reality (cf. Scaife & Rogers, 1996). The main concentration in this overview of graphical representations is focused on issues of static diagrams. There is a large variation in diagrammatic representations such as maps, flow diagrams, technical illustrations, pictures and etc. Each of this representational form is associated with wide range of functions (Scaife & Rogers, 1996).

2.1.1 Nature of Diagrams

Reasoning with diagrams attracts scientists, for diagrams can alleviate a problem solving process, because it is assumed that diagrams are often more effective than other propositional representations. The alleviation of a problem solving also happens, because generally diagrams reduce the amount of computation, required to understand the displayed information, by replacing relations of words and concepts with lines, arrows, shapes, and spatial arrangements. They also facilitate recognizing appropriate objects and inference rules (Cheng & Herbert, 1993; Gurr, 1999).

i All A are B

ii All B are C

iii (therefore) All A are C

Figure 2: Transitivity in Euler's circle in textual representation (taken from Gurr C. A, 1999, p. 4).

For example, Figure 2 and Figure 3 show textual and diagrammatic representations of the transitivity in Euler's circles. On one hand, transitive relation of set inclusion in textual representation (Figure 2) is captured by symbols in concatenation relation, which must be interpreted by intermediary syntax. On the other hand Euler's circles depicted in diagrammatic way (Figure 3) are easier to comprehend as it gives direct semantic information using labeled circles and spatial inclusion of the circles.

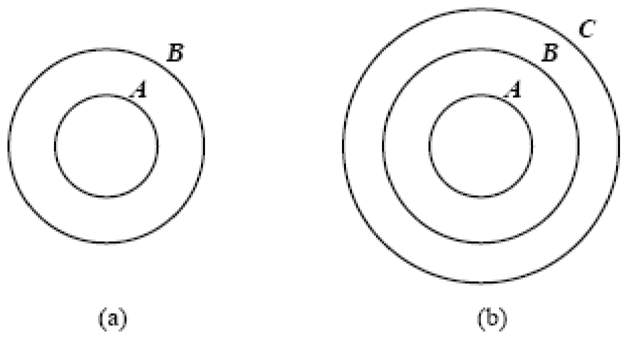


Figure 3: Transitivity in Euler's circle in diagrammatic representation (taken from Gurr C. A, 1999, p. 4).

One of the main features of diagrams is that their space and spatial properties preserve information about topological and geometric relations among the objects of the depicted problem. It provides information about the object's location in the way that makes it easier to track relations between the objects in space (Scaife & Rogers, 1996). However this characteristic of diagrams is not exclusive to them, because diagrammatic properties are also used to encode information in other representations just in lesser degree (Cheng et al., 2001). For example, in sentential representation of formula "$x = y + z$", it matters whether "$+z$" is on the left or on the right side of the equal sign. The other example would be logic sentence "$p \wedge (\neg q \vee r)$": it has some properties of visual representation, because its symbols are expressed to reader's visual sense by marking them on the page with ink.

Differently from other types of representations, diagrams have a property of compelling specification of certain classes of information. For example, a term "triangle" in sentential representation does not contain full information about the object and implies abstract description of any object belonging to the class of triangles. In this case, it does not give any information about what kind of triangle is considered: equilateral, isosceles or right-angle triangle. On the other hand, in a diagrammatic representation of a triangle, information about the specific subclass to which the triangle belongs along with its relative size is evident. Different diagrams permit different levels of abstraction. Diagrams with little abstraction are easy to use in reasoning, but express only limited information. Conversely, diagrams with substantial level of abstraction can represent a huge amount of information, but are difficult to reason with (Gurr, 1994).

2.1.2 Cognition with Diagrams

Graphical representation can be regarded as an arrangement of various graphic objects in space and is based on directness of information represented by them, which determines the effectiveness of this representation to human reader. The usability and suitability of diagrammatic representations are influenced by issues of human reaction to representations (Gurr, 1999).

Visual and spatial characteristics of diagrams and perceptual properties of a diagram's elements enable cognition of the problematic scene. Diagrammatic representations are more effective for the cognition process, because some inferences are more immediate or even automatic in diagrams. Textual representations require additional logic inference to be made in order to make conclusions, whereas diagrams provide conclusions of its own accord. For example, Figure 3 provides more direct inference than the same information depicted in Figure 2. So, recognizing the desired conclusion is actually not automatic, because the same diagram can contain not only the desired conclusion but many other potential conclusions. For example, different people can realize the same diagram in quite different ways, which can be far from each other. In this sense, a diagram performs more or less the role of a guide by displaying functional relations between terms. For example, lines and arrows present in a diagram can show a path that has to be followed. Humans can often make the right conclusions more easily if the diagram is well matched with the task.

Furthermore, efficiency of cognition also relies on meanings of elements in the diagram (Cheng et al., 2001). Humans can make right conclusions from diagrams, only if perceptual information of diagrams is modulated by knowledge about meanings of the graphic elements. The more person already knows about the subject matter depicted in diagram, more efficient is the use of diagram in reasoning. Moreover, diagrams actually do not contain all the information needed to come to the right conclusion. The knowledge and skill of the person is highly domain-specific and influence the efficiency of diagram usage.

2.2 Qualitative Spatial and Temporal Reasoning

One of the goals of qualitative reasoning and the process itself is to make explicit the everyday common-sense knowledge of the physical world. This knowledge with given appropriate techniques is needed for a computer to make predictions, analyze and explain the actions of physical system (Cohn et al., 1997).

The research in qualitative reasoning is motivated not only by reasoning in the traditional domain of physical systems, but also by a variety of possible areas such as robotic navigation, high level vision, spatial propositional semantics of natural languages, engineering design and specifying visual language syntax and semantics. Qualitative reasoning approaches perform reasoning on the conceptual level and seek to represent continuous properties of the world by discrete systems of symbols. One of the ways to do that is to use the relevance principle: “The distinctions made by quantization must be relevant to the kind of reasoning performed” (Cohn & Hazarika, 2001).

As the information about surrounding space can be perceived through various channels such as vision, touch, hearing, smell and etc., the knowledge of space differs from all other knowledge. Physical space is one of the main issues in cognition, because it is the domain in which events take place and it is a good reference domain for the interpretation of non-spatial concepts (Freksa, 1991b). Spatial reasoning appears as a field dealing with the conceptual “space” which comprises spatial representations such as topology, orientation, shape, size and distance. The process of the development of spatial reasoning formalism can be divided into three steps: preparatory step and two qualitative abstraction steps, which liberate representations from insignificant details and focus on the significant distinctions. The aim of preparatory step is to fix reasoning task by specifying a configuration space. Next, the set of qualitative relations along with appropriate inference rules is described. Finally, the last qualitative abstraction step defines a conceptual neighborhood structure for the qualitative relations (Schlieder, 1996).

Research on qualitative relations and their fundamental theories has been motivated by objective of how spatial relations are expressed in natural language and thought. Researchers are inclined to divide qualitative relations into two groups: spatial relations and temporal relations. Spatial relations provide information about spatial objects regardless of their actual geometry. For example, the same information would be given about the triangle, which is placed on the table, no matter what kind of triangle it is or what kind of table it is. This group of spatial relations can be divided into three subgroups: directions, distances and topological relations. Qualitative temporal relations describe objects at different states in time (David & Egenhofer, 1994; Sharma el at., 1994).

2.2.1 Topological Relations

One definition of topological relations can be based on the relation algebra, which deals with algebraic manipulation of symbols that represent geometric configurations and their relationships to one another. This algebra analyzes topological relations between any combinations of objects such as regions, lines and points (Sharma el at., 1994).

A 2-complex in $\mathbb{R}^2$ with a non-empty, connected interior is considered to be a *region.* A *region* with a connected exterior and a connected boundary is called a *region without holes* (Figure 4a), and a *region* with disconnected exterior and disconnected boundary is a *region with holes* (Figure 4b).

Definition of a *line* states that it is a sequence of connected 1-complexes in $\mathbb{R}^2$ in a way that they do not cross each other and do not form loops. There can be a *simple line* with two disconnected boundaries (Figure 4c) or a *complex line* with more than two disconnected boundaries (Figure 4d). The object *point* is defined as a single 0-cell in $\mathbb{R}^2$.

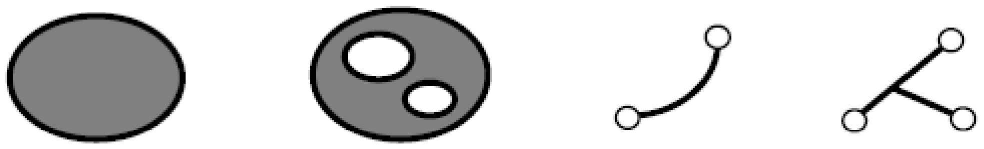


Figure 4: A region without holes (a), a region with holes (b), a simple line (c) and a complex line (d) (taken from Egenhofer & Herring, 1990, p. 6)

Topological relation between two objects, *A* and *B*, is described by the comparison of *A* object's interior (A^o), boundary (∂A) and exterior (A^-) with *B* object's interior (B^o), boundary (∂B) and exterior (B^-). The method called 9-intersection is based on the idea that these six parts combined together form nine fundamental descriptions of a topological relation between two objects (Egenhofer & Herring, 1990). These descriptions are:

- the intersection of *A*'s interior with *B*'s interior, noted as ($A^o \cap B^o$);
- the intersection of A's interior with B's boundary ($A^o \cap \partial B$);
- the intersection of A's interior with B's exterior ($A^o \cap B^-$);
- the intersection of A's boundary with B's interior ($\partial A \cap B^o$);
- the intersection of *A*'s boundary with *B*'s boundary ($\partial A \cap \partial B$);
- the intersection of A's boundary with B's exterior ($\partial A \cap B^-$);
- the intersection of A's exterior with B's interior ($A^- \cap B^o$);
- the intersection of A's exterior with B's boundary ($A^- \cap \partial B$);
- the intersection of *A*'s exterior with *B*'s exterior ($A^- \cap B^-$).

A topological relation R between two objects A and B is represented as 3 × 3 matrix of above mentioned intersections and noted:

$$R(A,B) = \begin{pmatrix} A^o \cap B^o & A^o \cap \partial B & A^o \cap B^- \\ \partial A \cap B^o & \partial A \cap \partial B & \partial A \cap B^- \\ A^- \cap B^o & A^- \cap \partial B & A^- \cap B^- \end{pmatrix} \qquad (1)$$

Different topological relations are described by different sets of 9-intersections and equivalent topological relations are described by relations with the same specifications (Egenhofer & Herring, 1990). In order to simplify the method the intersection content is assumed to be value empty ($\varnothing$) or non-empty ($\neg\varnothing$). Figure 5 shows the algebraic and visual interpretation of the eight relations between two arbitrary regions.

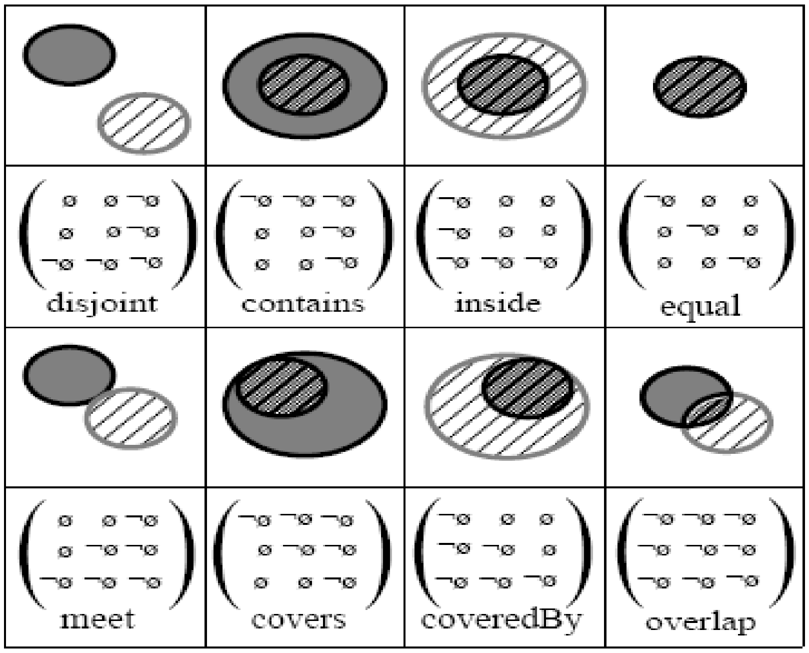


Figure 5: Geometric interpretation of the 8 relations between two regions in 2-D (taken from Egenhofer & Khaled, 1992, p. 200).

Corresponding variety of 9-intersections sets can be obtained to any combination of two objects, which can be any regions, any lines or points. The actual number of relations between two spatial objects depends on their topological properties.

2.2.2 Directions

Directional relations deal with the order in space and are commonly used in everyday life as we often describe one object on the basis of its directional relation with the other object. There are basic directions and cardinal directions. The term of basic directions is used here to mean the directions very often used in our daily life such as in front of, above, below, on the right and etc. The term of cardinal direction claims that it is binary relation involving a reference object and a target object. Cardinal directions can be described by quantitative values, such as azimuth or bearing, or quantitative symbols, such as east or south-west (Nabil el at., 1996; Sharma el at., 1994). The specific quantitative symbols, which are available in reasoning, depend on the system of directions used. The system of directions can consist only of four symbols: south, north, east and west, or it can be also extended by including four more symbols: south-west, south-east, north-east, north-west, etc. (Frank, 1996). The choice of description depends entirely on the system.

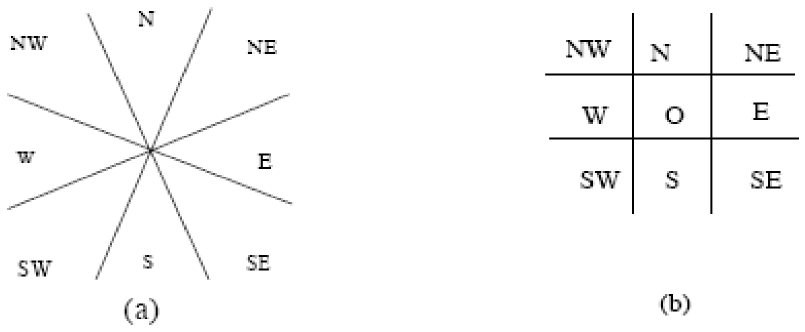


Figure 6: Cone-shaped (a) and projection-based (b) models for cardinal directions (taken from Sharma el at., 1994, p. 6).

The main concept of cardinal directions is taken from the compass. This also inspired the cone-shaped concept of direction approach, where directions are defined using angular regions between objects (Figure 6a). The other useful construction is based on projections (Figure 6b). This method defines cardinal directions using projection lines vertical to the coordinate axis (Theodoridis et al., 1996).

2.2.3 Distances

As the definition of the term distance proposes, it is quantitative value determined through measurements or calculated from known coordinates of two objects in some reference system (Sharma et al., 1994). In spite of this, approximations and qualitative concepts such as *near* and *far* are used to describe distances in reasoning. Measurement theory provides a theoretical base to approximate distance, which correspond to a set of ordered intervals and addition rules, which provide a complete partition such that the following interval is greater than or equal to the previous one (Hong et al., 1995). Reasoning using approximate distances can provide effective and meaningful results only if combined with reasoning using cardinal directions (Frank, 1996).

2.2.4 Temporal Relations

Temporal relations are based on James Allen’s popular temporal logic and represent temporal changes between spatial objects (Allen, 1983). Allen proposed to describe qualitative relations between events or objects using intervals (Sharma et al., 1994) and introduced method to derive relationships between intervals (Nabil et al., 1996). The interval is assumed to be a fully ordered set of points along a discrete time line between the endpoints of the interval. Based on this representation, Allen derived interval algebra with clear semantics (Rauh et al., 2000), according to which any ordered pair of intervals is related in one of the thirteen temporal relations (Figure 7).

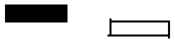
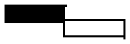
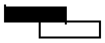
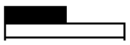
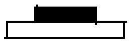
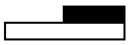
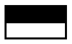
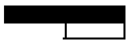
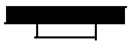
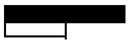
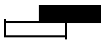
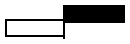
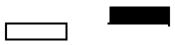
Relation symbol	Natural language description	Graphical example
X < Y	X lies to the left of Y	
X m Y	X touches Y at the left	
X o Y	X overlaps Y from the left	
X s Y	X lies left-justified in Y	
X d Y	X is completely in Y	
X f Y	X lies right-justified in Y	
X = Y	X equals Y	
X fi Y	X contains Y right-justified	
X di Y	X surrounds Y	
X si Y	X contains Y left-justified	
X oi Y	X overlaps Y from the right	
X mi Y	X touches Y at the right	
X > Y	X lies to the right of Y	

Figure 7: Allen's thirteen 1-dimensional interval relations (taken from Rauh et al., 2000, p 872).

Allen's thirteen 1-dimensional interval relations table is composed of six relationships with their inverses and one relationship, which has no inverse (Freksa, 1991b; Nabil et al., 1996). In case of the basic relation *"equal"*, it does not have any inverse. This algebra describes the properties of the temporal relations and defines sets of rules that permit inferences about relations (Rauh et al., 2000). Reasoning using temporal relations is done by making sets of possible relations between two objects in relation to the third object.

2.3 Conceptual Neighborhoods

Reasoning using neighborhoods is used to describe the possibility to transform two spatial or temporal configurations of the objects into each other using small changes in the position or size of the objects (Dylla & Moratz, 2005).

The concept of conceptual neighborhoods defines the similarity measures for a set of relations. It is graphically represented as a graph constructed from nodes, which define relations, and links, which connect relations which can be directly transformed to each other (Papadias et al., 2001). The shorter is the way to reach one relation from the other the more similar those relations are. If the structure of conceptual neighborhoods is known, there can be made predictions about changes and reasoning with relations.

As this thesis aims to explore dynamic scene analysis and beautification for hand-drawn sketches on geometric and qualitative levels of abstraction, the focus is set on conceptual neighborhoods of spatial relations not temporal ones.

2.3.1 Neighborhoods of Topological Relations

Topological relations define spatial configuration between two objects such as metric details, topology issues. If some topological constraints are changed, significant alterations between relations occur (Bruns & Egenhofer, 1996). Although the number and extent of changes can increase with every step, generally the change is gradual. It starts from equivalent scene, then transforms to very similar, then to less and less similar ending with totally different variation. The concept of gradual change has been the basis for the conceptual neighborhoods model of topological relations (Figure 8).

Figure 8 represents conceptual neighborhood of eight topological relations between two simple regions. Each relation in the figure is connected with its conceptual neighbors with the help of a line.

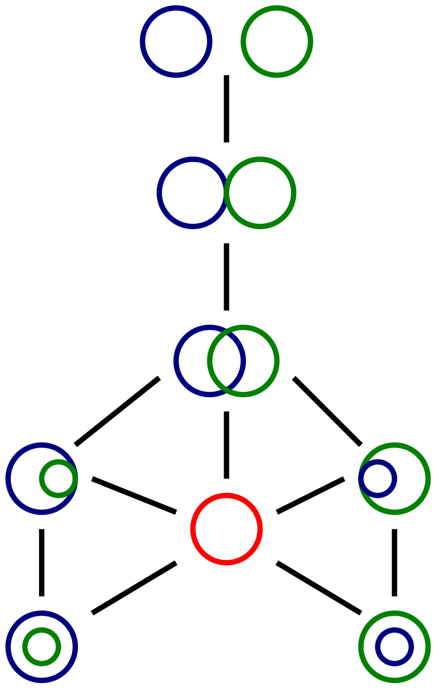


Figure 8: Conceptual neighborhood of topological relations (after Bruns & Egenhofer, 1996, p. 33).

Individually topological conceptual neighborhoods are effective in reasoning about simple two-object scenes (Bruns & Egenhofer, 1996), because the structure of the conceptual neighborhood is not so complex. Their usage in more complicated scenes is inefficient, because as the complexity of objects increases, the conceptual neighborhoods becomes more complex and hard to comprehend. However, the combination of conceptual neighborhoods of topological, cardinal and distance relations is an efficient utility for spatial reasoning of scenes consisting of more than two objects.

2.3.2 Neighborhoods of Direction Relations

Cardinal directions are very important in specifications of spatial scene, because they describe the orientation between spatial objects. The structure of neighborhood graph of cardinal relations is derived from Allen’s interval method applied to orthogonal projections with the constrain that any object can move continuously and smoothly in any direction but without jumping to a new location (Goyal & Egenhofer, 2001).

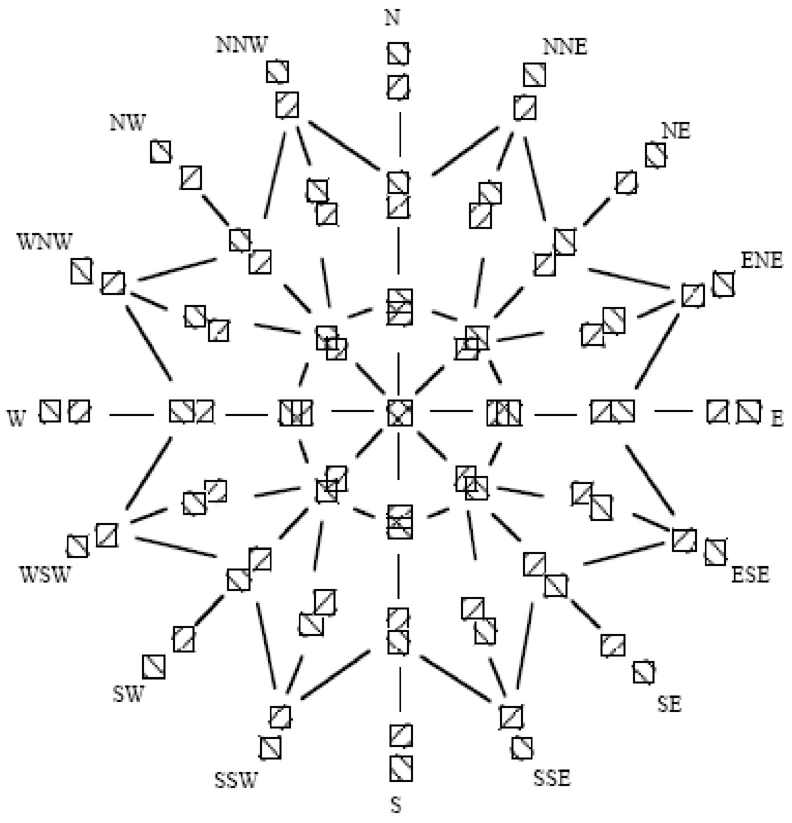


Figure 9: Conceptual neighborhood of cardinal relations of two squares (taken from Bruns & Egenhofer, 1996, p. 34).

Figure 9 depicts conceptual neighborhood graph constructed for cardinal relations of two equal squares. In this case, conceptual neighborhood is interpretation of either square in any direction. Conceptual neighbors are connected via links, indicating which cardinal relations can be gradually derived one from other. The similarity of relations is evaluated by the number of intermediate relations on the way from one relation to other. Although such neighborhood graphs of cardinal relations can be generated for other shapes, but they are more difficult to depict and interpret in the 2-dimensional space (Bruns & Egenhofer, 1996; Goyal & Egenhofer, 2001).

2.3.3 Neighborhoods of Distance Relations

In opposite to above mentioned spatial relations, it is difficult to characterize distance relations of spatial objects. All concepts and terms used in reasoning with distance relations are not objective ones and very sensitive to the spatial data which is being considered (Bruns & Egenhofer, 1996). One of the methods to describe similarity between distance relations is to use increasing buffer distances (Figure 10).

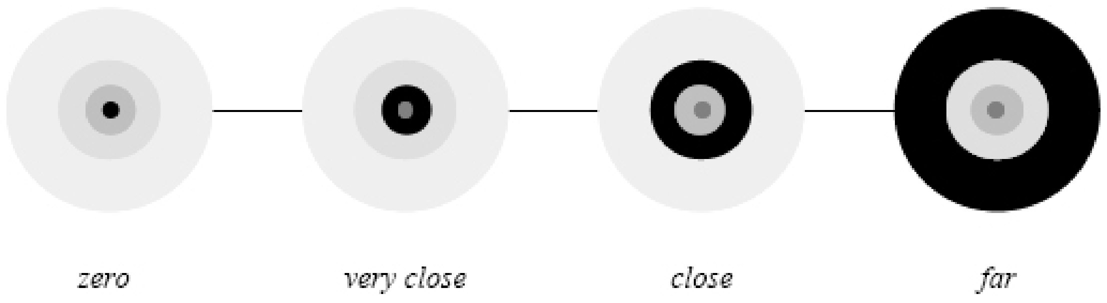


Figure 10: Conceptual neighborhood of distance relations (taken from Bruns & Egenhofer, 1996, p. 33).

Figure 10 graphically shows conceptual neighborhoods of distance relations for a of four distance relations. In this case, structure of conceptual neighborhoods is based on an order relation. In such neighborhood graph order relation "*less than*" (<) is used to depict the gradual transformation of one relation to more and more diverse one. Two adjacent relations are more similar to each other than the distant ones. For example, distance relations "*close*" and "*very close*" are more similar to each other than the latter one and the relation "*far*". This is because according order relation over distance symbols "*very close*" < "*close*" < "*far*". This type of reasoning can be good utility to determine how many gradual changes are required to complete transformation from one scene to another (Bruns & Egenhofer, 1996).

2.4 Recognition

2.4.1 Statistical Moments

Recognition of visual patterns independent of position, size and orientation is a goal of research a much long time. Statistical moments are extensively used in many different aspects of image processing, ranging from invariant pattern recognition and image encoding to pose estimation (Schutler, 2002). Image recognition can be done using statistical moments such as the mean, variance, and higher-order moments, because they describe the image distribution with respect to its axes (Gonzalez & Woods, 2002).

The moment of an image is expressed by the standard equation (Taubman, 2005):

$$M_{pq} = \sum_{x}^{X} \sum_{y}^{Y} (x^p y^q) P_{xy} \tag{2}$$

where $p, q \geq 0$. In the above equation X and Y are the width and height of an image respectively and $x^p y^q$ is monomial product. P_{xy} is the value of the pixel, when pixel x, y is white it is 0, and when it is 1, pixel x, y is black. The image usually has to be inverted and adjusted according threshold in order to meet binary requirements of P_{xy}. The moments of the image can be computed only after these adjustments.

As there is infinite number of moments, a subset of moments has to be chosen for recognition purposes (Taubman, 2005). Usually all moments from $p = 0, q = 0$ to $p = 3, q = 3$ are used because they change with image position and image scale.

The zero order moment M_{00} represents the total mass (or power) of the image. If it is applied to binary image, then it is literally a pixel count of the number of pixels comprising the object (Schutler, 2002).

It is well known, that centralized moments do not change under the translation of coordinates. This leads to the statement that centralized moments μ_{pq} are invariant under translation and are defined as (Hu, 1962):

$$\mu_{pq} = \sum_{x}^{X} \sum_{y}^{Y} (x - \overline{x})^p (y - \overline{y})^q P_{xy} \tag{3}$$

where

$$\overline{x} = \frac{M_{10}}{M_{00}} \tag{4}$$

$$\overline{y} = \frac{M_{01}}{M_{00}} \tag{5}$$

In order to enable invariance under scaling normalized moments η_{pq} are calculated:

$$\eta_{pq} = \frac{\mu_{pq}}{(\mu_{00})^{\gamma}} \tag{6}$$

where

$$\gamma = \frac{p+q}{2} + 1 \qquad \forall (p+q) \geq 2$$

Hu derived seven invariant moments of second- and third-order. The first six Hu moments encode shape of the object with invariance to scale, position and rotation. The seventh moment is calculated with invariance to skew, which lets to distinguish between mirrored images. Seven Hu moments are expressed by these equations (Poppe & Poel, 2006):

$$I_1 = \eta_{20} + \eta_{02} \quad (7)$$

$$I_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \quad (8)$$

$$I_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \quad (9)$$

$$I_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2 \quad (10)$$

$$\begin{aligned} I_5 &= (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})((\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2) + \\ &+(3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})(3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2) \end{aligned} \quad (11)$$

$$I_6 = (\eta_{20}\eta_{02})((\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2 + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03})) \quad (12)$$

$$\begin{aligned} I_7 &= (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})((\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2) + \\ &+(\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})(3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2) \end{aligned} \quad (13)$$

These moments can be generalized to accomplish pattern recognition not only independently of size, position and rotation but also independently of parallel projection (Hu, 1962).

Chapter 3

Design

This chapter presents the algorithm for dynamic scene analysis and beautification of hand-drawn sketches. There will be presented and discussed each part of the algorithm separately with the focus on recognition, beautification and adjustment methods.

3.1 System Model

This work is concentrated on the design of system prototype for dynamic scene analysis and beautification of hand-drawn sketches. The main concept of the system is that it interacts with user in real-time while the process of sketching is going on. Every time user draws an object the system should give the response by making appropriate corrections in the sketch. The structure of this system model is presented in Figure 11:

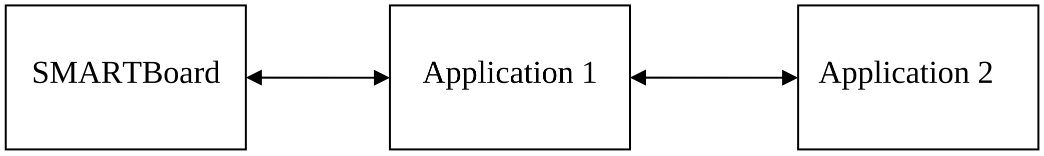


Figure 11: Structure of the system model

As it is shown in Figure D1, one of the components of the system is a SMARTBoard. A SmartBoard is an interactive whiteboard that is connected to a computer and a data projector. It is used because it will give to the user a sense of natural sketching. Next component of the system is Application 1, which is designed for drawn object detection. The main aim of this application is to detect the event when user finishes drawing of the object on the SMARTBoard. It is assumed that application is tracking the motion of pen used by the user and the strength of the pen's pressing. When the pen is withdrawn for longer time then N seconds, it is assumed that the user finished drawing of the object. The waiting period of N seconds enables users to draw not only continuous shapes but also to sketch using strokes. If the event of object finishing is registered the SMARTBoard's screen is captured and transferred to the Application 2, which is designed for dynamic scene analysis. This application analyzes and identifies the object, performs geometric beautifications if they are needed and adjustments of the sketch, according to the objects which have been sketched before. As the result, Application 2 sends the corrected sketch scene back to Application 1, which displays the result to the user on the SMARTBoard. When the responds to the user's made drawings, the user can indicate whether the interpretation of his sketching is correct or not. If the interpretation is incorrect then system gives to the user the next more likely interpretation of the sketch. If the user in time duration of N seconds doesn't indicate that the interpretation is incorrect, the system gets ready for the processing of the next object.

As this thesis is more based on the process of dynamic scene analysis and beautification of hand-drawn sketches on geometric and qualitative levels of abstraction, the focus is set on the design of Application 2.

3.2 Algorithm for dynamic scene analysis and beautification

An algorithm has been designed in order to implement the functionality of Application 2 discussed in the above section.

As it is shown in Figure 12 the proposed algorithm consists of four steps:

- Input processing – the preparation of input data for the coming operations.
- Recognition process – the process of the drawn object recognition in comparison to object classes stored in the system.
- Geometric beautification – the process of the drawn object beautification by the detection and imposition of geometric regularities and the required corrections of defects.
- Adjustment – the adjustment of sketch according to the spatial relations between the current object and the object, which has been sketched before the current one.

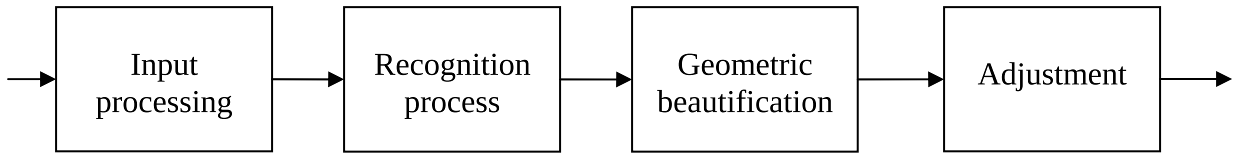


Figure 12: Structure of the algorithm for dynamic scene analysis and beautification

Each step of the algorithm is discussed below in more details introducing the approach, which has been used to fulfill the purpose of it. The restrain of this algorithm is that it has been designed based mainly on some database of geometric basic shapes. This means that there no guarantees, that it will work with the shapes, which were not used for the design of the algorithm.

3.2.1 Input Processing

This part of the algorithm prepares data for the following steps. It gets the image of drawing screen and converts it into binary image, where 0 represents a white pixel and 1 represents a black one. Afterwards, the changes which have been made to that picture are distinguished by comparing the current sketch with the lastly processed sketch. The difference of these sketches indicates an object, which has been recently drawn by a user. Next, the object's interest area is clipped out of the sketch area. The term of interest area is used here with intention to designate a rectangle shape area containing the actual object and some additional white pixels. When the interest area of the object is excised, it is transmitted to the recognition process.

3.2.2 Recognition Process

The recognition process is performed in order to classify to which class of shapes the drawn object can be assigned. This step of algorithm is based on Hu moments, which are invariant to scale, position and rotation. A sketched object can be only recognized as the one belonging to some specific class of shapes. Beforehand, there has to be trained a system with the samples of a users drawn shapes. It is done with the assumption that different people have different specific way of sketching.

After training is completed, results are kept in a database in a form of shapes' definition with their Hu moments feature properties. A Hu moment feature vector is here assumed to mean the vector consisting of seven Hu moments which characterize the shape. This work concentrates on the recognition of circles and squares with assumption that user is drawing these shapes in small range of sizes.

When the object A is sketched and its interest area is clipped out of the sketch by the input processing part, the recognition process calculates the feature vector of the drawn object:

$$HU_A = \{ I_1 \quad I_2 \quad I_3 \quad I_4 \quad I_5 \quad I_6 \quad I_7 \} \tag{14}$$

where HU_A is a feature vector and I_k is appropriate Hu moment, when $i = 1, 2, ..., 7$.

When the feature vector of the object is calculated, the recognition is processed. There is performed the comparison of the distances between object feature vector and feature vector of the shapes kept in database. In this way, it is checked to which shape's domain the drawn object belongs. The shortest distance determines the class of the object.

In order to confirm that the chosen concept is working as it is assumed, there have been made some tests (Figure 13). Testing is performed on the two basic shapes: circle and square. There have been made 15 samples for each shape, in total 30 samples. The recognition algorithm was trained with all 30 samples. The feature vector of seven Hu moments for each shape has been generated by taking mean value of each shape's Hu moments values for each sample.

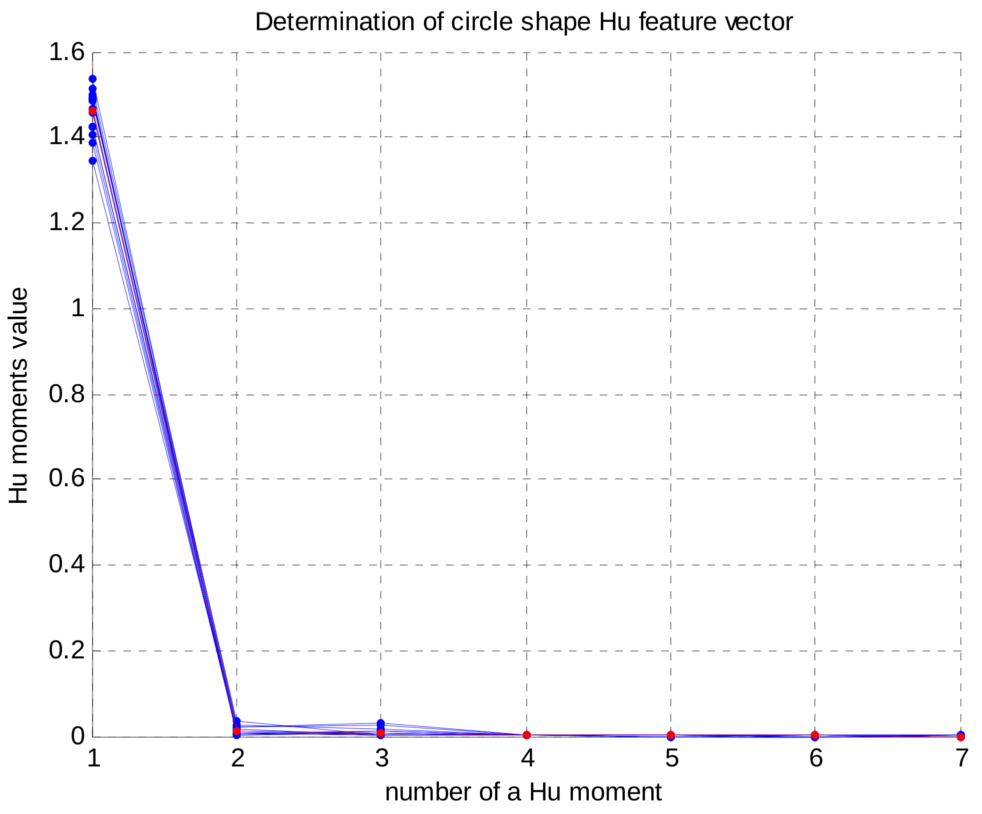


Figure 13: Determination of Hu feature vector a circle shape

Figure 13, shows the Hu moments (blue line) for all training 15 samples, which have been used to define circle shape's Hu feature vector (red line). Figure 14, presents the calculation of Hu feature vector for a square shape, where blue lines shows Hu moments of training samples and red line indicates the estimated Hu vector for the square shape.

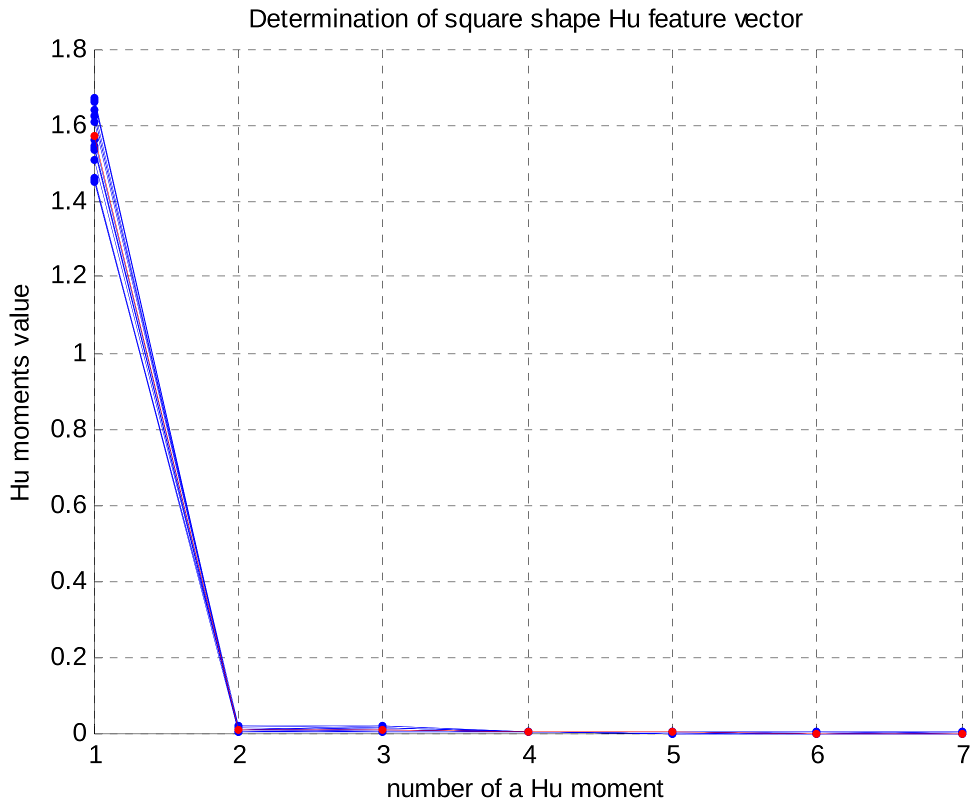


Figure 14: Determination of Hu feature vector a square shape

It can be seen that these two shapes have some different Hu moments' features. Firstly, the most informative Hu moments for a circle shape are the first four moments, whereas for a square the most informative ones are the first three Hu moments. Moreover we can see that the most significant moment is the first Hu moment. In the case of a circle shape first Hu moment values are in the range from 1.3 to 1.5, when in the case of a square shape values are in the range from 1.5 to 1.8.

Afterwards, the system is asked to classify 60 test samples, which are the mixture of samples from the training set and the completely new samples. From the Figure 15, the efficiency of recognition can be seen. Images assumed to be of the circle's shape are recognized with 85 percent, and squares are recognized with 95 percent.

	Object recognition, %
Circle samples	85
Square samples	95

Figure 15: Sample testing

Although Hu moments are invariant to scale, rotation and position, their performance is limited in some cases. The performed tests showed that the size of the object matters to the recognition using Hu moments.

When the shape database comprises a large range of different shapes, it is more efficient to use Zernike moments for the recognition (Hse & Newton, 2004). However, this algorithm is not considering the usage of these moments, because in order to calculate Zernike moments there have to be performed huge complex calculations.

As this system prototype is assumed to be more used by designers, who are sketching mostly using basic shapes, it is better to use Hu moments which are more fast and easy to calculate.

3.2.3 Geometric Beautification Process

After completion of object recognition, the obtained information about the object is utilized in the beautification process, in order to make the required corrections such as the straightening of lines, the forming of correct angles, connecting lines, etc. and to create a neat and clean version of the hand-drawn image.

The algorithm’s beautification step gets information about an object in two aspects (Figure 16). Firstly, the exact area of interest is taken from the sketch. Secondly, the name of the object's shape is assigned to the object. This information is passed on from the earlier steps of the algorithm.

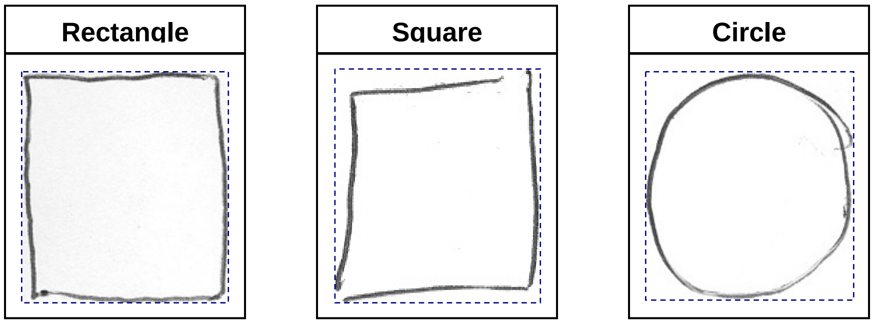


Figure 16: Input to the beautification step of the algorithm

The aim of beautification is to use information, which is provided by object recognition, in reproducing the recognized object into an exact and pre-defined shape. Different shapes require appropriate beautification of different complexity levels. The process of beautification for a few basic shapes such as squares, rectangles, circle, ellipse and etc. does not demand complex calculations and amendments, while beautification of complex shapes takes a lot more actions and lot more complex computations.

3.2.3.1 Beautification of Basic Shapes

The process of beautification for a few basic shapes does not demand complex calculations and amendments. The term of basic shapes is used here to mean the basic two-dimensional geometrical objects such as squares, rectangles, circle, ellipse and etc. The methods for beautification of each of this method are discussed below in more detail.

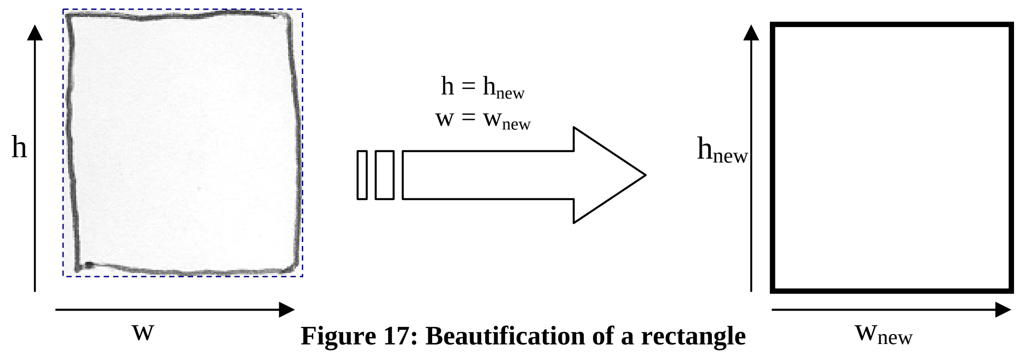


Figure 17: Beautification of a rectangle

The example shown in Figure 17 presents beautification of the object in the case, when the object is recognized to be a rectangle. The beautification process of a rectangle is straightforward and involves the measuring of the interest area. According to the size of the height and width of interest area, the neat and ideal rectangle is produced.

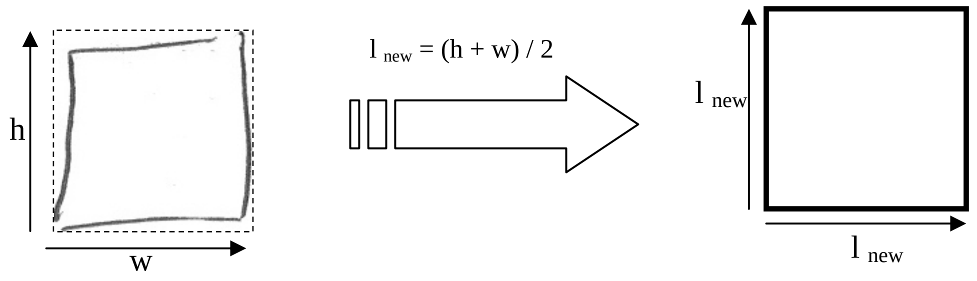


Figure 18: Beautification of a square

As it is shown in Figure 18, beautification of an object recognized as a square requires more actions. Since the recognized object is a square, the intention is to draw a symmetrical shape with four equal borders. In order to define such a shape, the height and the width of the object’s interest area are measured. The average of the height and width is calculated to determine the length of the border for the beautified square. When the length of the border is calculated, the neat and ideal square is generated instead the sketched one.

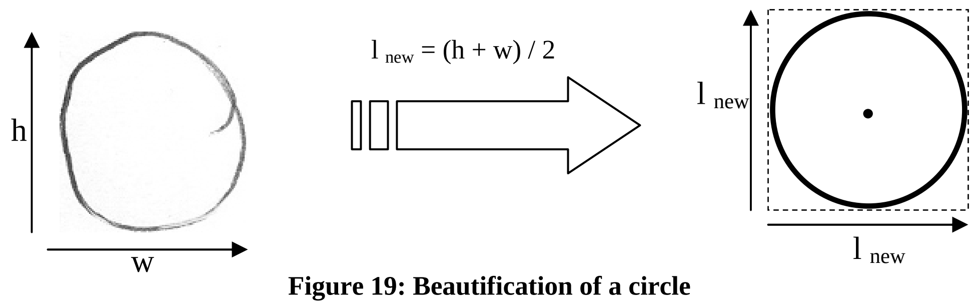


Figure 19: Beautification of a circle

Similarly, the beautification process of a circle is done (Figure 19). As in the case with a square, first the size of a border for a new interest area for object is defined. In addition, the beautification process of a sketched circle requires the calculation of the center point for the new interest area. If the size of the border is equal to an even number, then an additional point is added by the beautification engine. This is done, because an additional point is required to fit ideal circle shape into the interest area of the drawn object. Next, a radius of an ideal circle is calculated and circle is generated in the new interest area.

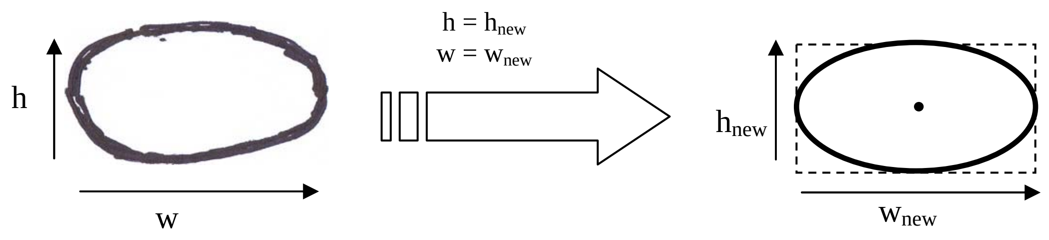


Figure 20: Beautification of an ellipse

Appropriately, when an object is recognized as ellipse (Figure 20), the beautification process performs actions very similar as in the case with a rectangle and a square. The interest area of an ideal shape is calculated similarly as in the case with a rectangle. Similarly to the case of circle beautification, there has to be estimated center of the ellipse. If the width of interest area is equal to an even number, then an additional point is added to the width. Appropriately, if the height of interest area is equal to an even number, then an additional point is added to the height of the interest area. These corrections of the interest area ensures that ellipse is symmetrical along x and y axis. When the interest area is amended, two radius of ellipse are calculated and an ideal ellipse is produced in the interest area.

3.2.3.2 Beautification of Complex Shapes

If object is recognized to be from the class of complex shapes, geometrical beautification is performed based on method proposed by H. H. Hse and A. R. Newton (Hse & Newton, 2005).

In order to make meaningful beautification of complex shapes such as parallelograms, trapezoids, hexagons and etc. more detailed structural information about the object is needed. Such information is obtained with the help of object segmentation into straight lines (L) and elliptical arcs (E). As the result of segmentation there is generated a segmentation template consisting of the number of L's and the number of E's. For the better understanding of the concept of this method, beautification of several shapes will be explained in more detail below.

For example, when the object is recognized as a parallelogram, the line segments are ordered consecutively (Figure 21). Processing of the data is started with the segment (1) that forms an acute angle ($H1$) with another segment.

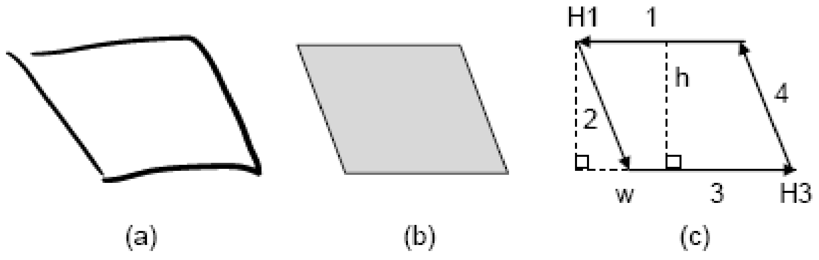


Figure 21: **Sketched parallelogram (a), beautified parallelogram (b) and graphical description of the parameters (c) (taken from Hse & Newton, 2005, p. 4)**

In the case of a trapezoid (Figure 22), there is computation of the dot products of the opposing segments in order to determine two parallel sides. Afterwards, the top-short (1) and bottom-long (3) sides of trapezoid are defined, all segments arranged in clockwise order stating from the top one and rotation angle is computed.

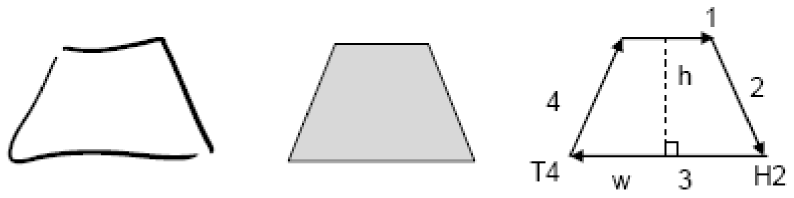


Figure 22: Sketched trapezoid (a), beautified trapezoid (b) and graphical description of the parameters (c) (taken from Hse & Newton, 2005, p. 4)

Other complex shapes are also beautified applying this method, but extent of beautification of all objects is limited by the available support of the designed application in terms of how many shapes are definite in the application.

3.2.4 Adjustment

The adjustment step of this algorithm aims to place object on the screen in the position at which the user has meant to draw it. The place of the object is located according to its spatial relations with other previously drawn objects. This step considers all three above mentioned types of spatial relations: directions, distances and topological relations. The list of spatial relations between the current object and the five the most recently drawn other objects is generated in order to decide which previously drawn object is in the most "closest" relations with the current one. The most recently drawn objects are considered and the oldest ones are left out assuming that mostly people are sketching objects in turn and structurally. It is assumed that it is less likely that a user draws more closely related objects one after each other or within five objects. When the "closest" object is determined, adjustments of the current object are made on the basis of its relations with the closest one. Basically adjustment step can be divided into three sub-steps:

- Generating list of relations with other objects: directions, distances and topological relations.
- Determining the "closest" object to the current object
- Performing adjustments according the "closest" object

There are explained in more detail all sub-steps of the adjustment. The result of the first sub-step is the list of previously drawn objects with the description of their spatial relations with the current object. The description of spatial relations consists of direction, distance and topological relations properties. Below the method of determining each type of spatial relations is described more exactly. Furthermore there comes the explanation how the adjustment is done according to the properties of spatial relations.

3.2.4.1 Determination of Directions

This algorithm is using cardinal directions as one of the properties used for the adjustment step. Here we are using system of directions defined by eight quantitative symbols: south, north, east, west, south-east, south-west, north-east and north-west. The method of directions determination between two objects is depicted in Figure 23.

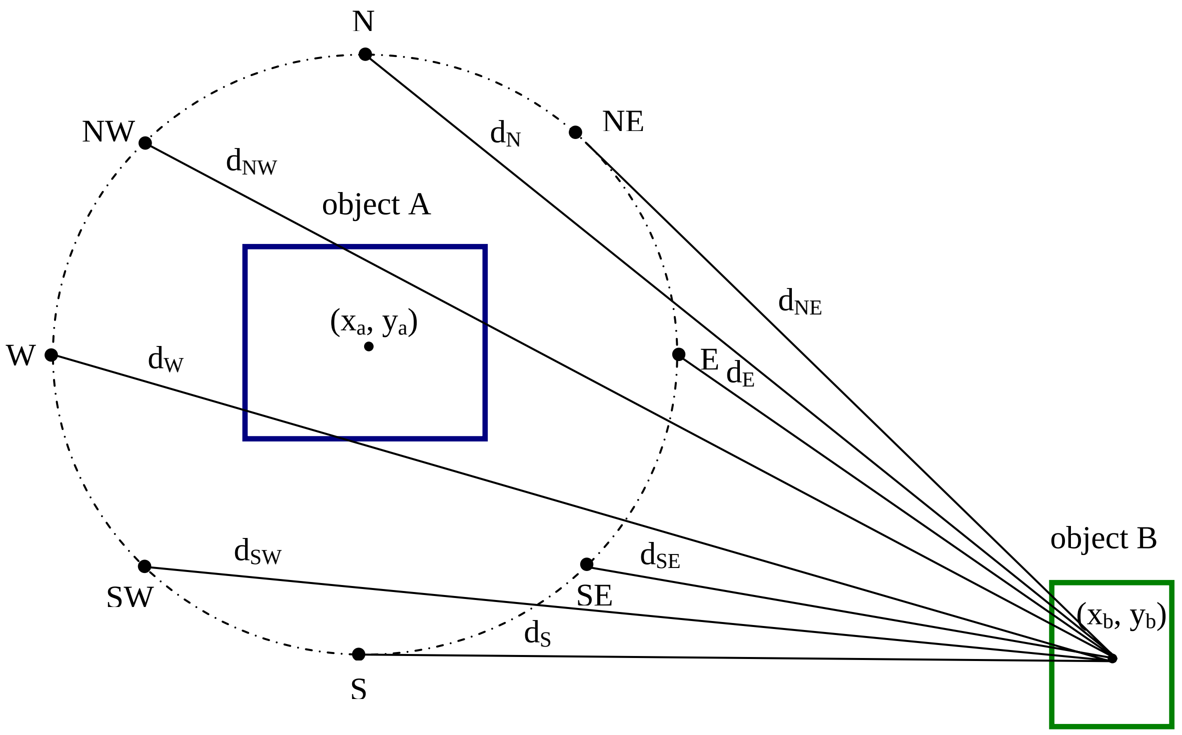


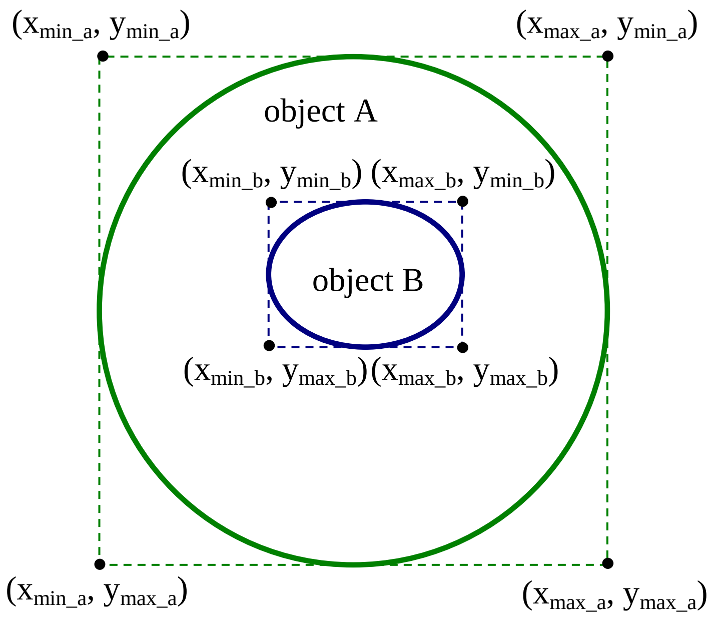
Figure 23: Determination of the direction between objects A and B

Figure 23 shows objects A and B, which have been drawn by a user. It is considered that A object is drawn before B object, so A object is used as reference object and B object is a target object. Now the task is to find out in what cardinal position is target object B according to the reference object A. Firstly, there is set an imaginary circle, which center coordinates (x_a, y_a) are the same as the center coordinates of the reference object A. The length of the radius of the imaginary circle is not relevant, but the bigger it is the more distinctive the determination of direction is. Eight points are set on the imaginary circle in order to represent all eight cardinal directions. South is depicted by S point, north is N point, E is east point and etc. When cardinal points are set, the distances between those eight points and the center point (x_b, y_b) of the target object B are calculated. A distance between the N point and center point of the target object is represented by d_N distance, a distance distances to the south-east point is denoted by d_{SE} line and etc. The shortest distance shows the target object's B position according to the reference object A. When the direction is defined, it is assigned to the relation between two objects. Afterwards the algorithm starts the determination of the distance between these objects.

3.2.4.2 Determination of Distances

Besides directions this algorithm is also using distances in order to make adjustments of the drawn object's position in the sketch. The distances are defined using the qualitative concepts: *zero*, *near* and *far*. In order to detect where one object is in comparison to the other object we are using direction's relations between two objects. The information about direction relations between those objects is obtained from the previous step, which is described in the sub-section above.

The determination of the distance between two objects requires several actions. First, it is determined whether one of those objects covers the other one. The term "covers" is used here to mean that one object's interest area is covered by other objects interest area (Figure 24).



(x_{min_a}, y_{min_a})
(x_{max_a}, y_{min_a})
object A
(x_{min_b}, y_{min_b})
(x_{max_b}, y_{min_b})
object B
(x_{min_b}, y_{max_b})
(x_{max_b}, y_{max_b})
(x_{min_a}, y_{max_a})
(x_{max_a}, y_{max_a})

Figure 24: Object's A interest area covers object's B interest area

Figure 24 shows the case, where the interest area of object A covers the interest area of object B and satisfy conditions:

$$\begin{cases} x_{\min\_a} \le x_{\min\_b} \\ x_{\max\_a} \ge x_{\max\_b} \\ y_{\min\_a} \le y_{\min\_b} \\ y_{\max\_a} \ge y_{\max\_b} \end{cases} \tag{15}$$

where $(x_{\min\_a}, y_{\min\_a})$ and $(x_{\max\_a}, y_{\max\_a})$ are coordinates of A object's interest area's upper left and bottom right corners, $(x_{\min\_b}, y_{\min\_b})$ and $(x_{\max\_b}, y_{\max\_b})$ are coordinates of B object's interest area's bottom right corners.

In such a case it is claimed, that object A is in *zero* distance from object B, otherwise next actions of distance determination are performed.

When it is clear that object A is not covering object B, the algorithm checks whether the intersection between these objects exists and what the size of it is. Actually the term "intersection" is used here to indicate the intersection between one object's interest area and other object's interest area (Figure 25).

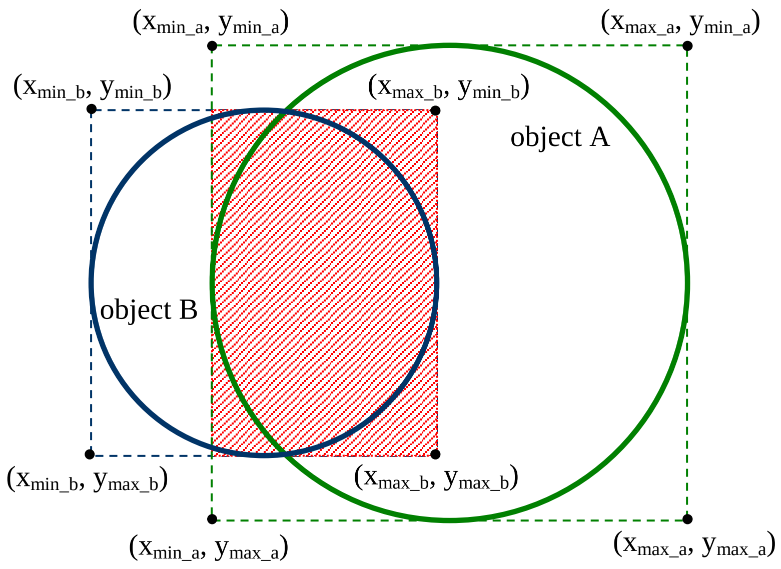


Figure 25: Object's A interest area intersects with object's B interest area

Figure 25 shows the case, where the interest area of object A intersects with the interest area of object B. The intersection region is distinguished by red stripes. The intersection is detected, if at least two conditions out of four are satisfied:

$$\begin{aligned} x_{\min\_a} &\leq x_{\min\_b} \\ x_{\max\_a} &\geq x_{\max\_b} \\ y_{\min\_a} &\leq y_{\min\_b} \\ y_{\max\_a} &\geq y_{\max\_b} \end{aligned} \qquad (16)$$

where $(x_{\min\_a}, y_{\min\_a})$ and $(x_{\max\_a}, y_{\max\_a})$ are coordinates of A object's interest area's upper left and bottom right corners, $(x_{\min\_b}, y_{\min\_b})$ and $(x_{\max\_b}, y_{\max\_b})$ are coordinates of B object's interest area's bottom right corners.

When the intersection is detected, the next step is to find out the distances between the intersecting borders of objects' A and B interest areas, which determine the distance between object A and object B. If one of the distances between the intersecting borders is bigger than the threshold defined by the thickness of the pen, it is considered that the distance between objects A and B is *zero*. If this is not the case, then objects A and B are assumed to be *near* to each other.

If objects A and B are not covering and are not intersecting, then it is clear that they can be near or far from each other but not in the zero distance. In order to determine whether two objects are near or far from each other there have to be measured the distance between appropriate borders of two objects (Figure 26)

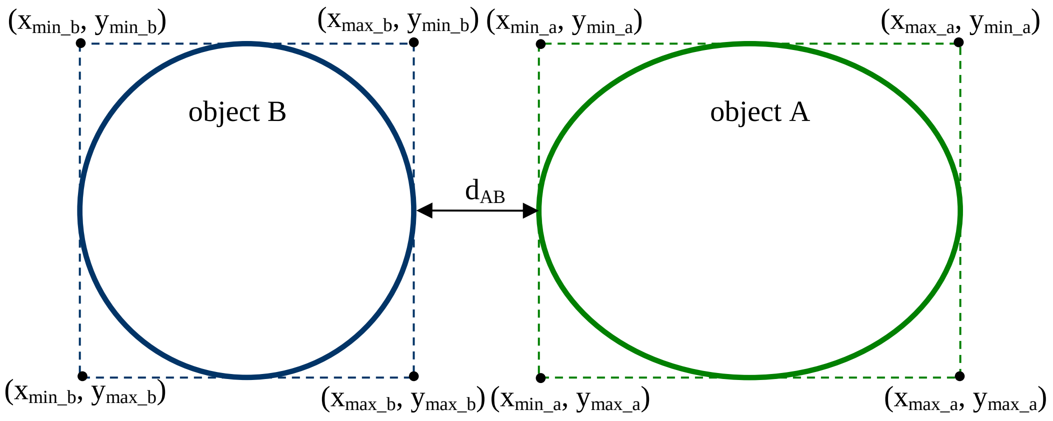


Figure 26: The calculation of distance between object A and object B with direction defined as "west"

Figure 26 shows two objects A and B, which are in distance defined by d_{AB} value. This vale is used to determine whether objects A and B are near or far from each other. In order to find out distance d_{AB} direction's relation between objects A and B is used. In this example, object A is a reference object, object B is a target object. The direction's relation is defined as "*west*", which means that object B is in west to object B. In this case distance between objects A and B is calculated:

$$d_{AB} = x_{\min\_a} - x_{\max\_b} \tag{17}$$

where $x_{\min\_a}$ is x axis coordinate of A object's interest area's left border, $x_{\max\_b}$ is x axis coordinate of B object's interest area's right border.

If direction between objects A and B is "*east*", distance d_{AB} is calculated appropriately:

$$d_{AB} = x_{min\_b} - x_{\max\_a} \tag{18}$$

where $x_{\min\_b}$ is x axis coordinate of B object's interest area's left border, $x_{\max\_a}$ is x axis coordinate of A object's interest area's right border.

When object's B position according to object's A position is defined as "*north*", distance d_{AB} is:

$$d_{AB} = y_{min\_a} - y_{\max\_b} \qquad (19)$$

where $y_{\min\_a}$ is y axis coordinate of A object's interest area's upper border, $y_{\max\_b}$ is y axis coordinate of B object's interest area's bottom border.

In the case, when object B is in "*south*" to object A, distance d_{AB} is obtained by:

$$d_{AB} = y_{min\_b} - y_{\max\_a} \qquad (20)$$

where $y_{\min\_b}$ is y axis coordinate of B object's interest area's upper border, $y_{\max\_a}$ is y axis coordinate of A object's interest area's bottom border.

When direction's relation between two objects is defined in terms of four basic directions ("*south*", "*north*", "west", "east"), it is assumed that in order to determine distance relation, it is enough to calculate distance according to one appropriate axis (x or y).

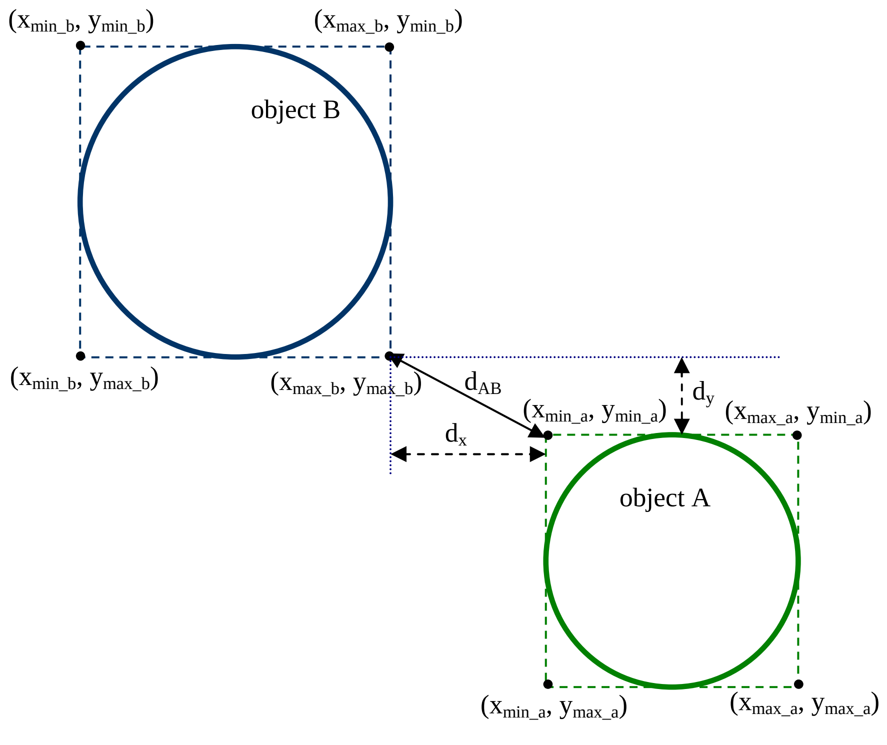


Figure 27: The calculation of distance between objects A and B with direction defined as "north-west"

Figure 27 shows the case, where object B is "*north-west*" position to object A. In order to find the distance d_{AB} between these objects it is not enough to make calculations utilizing only one axis measurements. As the direction property "*north-west*" indicates, there should be used combination of calculations performed in the case of "*north*" and "*west*". In the case depicted by

Figure 27, distance d_{AB} is calculated using Euclidian distance estimation method and calculations performed in the case of "*north*" and "*west*":

$$d_{AB} = \sqrt{(y_{min\_a} - y_{\max\_b})^2 + (x_{\min\_a} - x_{\max\_b})^2} \qquad (21)$$

where $(x_{\min\_a}, y_{\min\_a})$ are coordinates of A object's interest area's upper left corner, $(x_{\max\_b}, y_{\max\_b})$ are coordinates of B object's interest area's bottom right corner.

In the case, when object B is in "*north-east*" to object's A, distance d_{AB} is obtained by:

$$d_{AB} = \sqrt{(y_{min\_a} - y_{\max\_b})^2 + (x_{min\_b} - x_{\max\_a})^2} \qquad (22)$$

where $(x_{\max\_a}, y_{\min\_a})$ are coordinates of A object's interest area's upper right corner, $(x_{\min\_b}, y_{\max\_b})$ are coordinates of B object's interest area's bottom left corner.

When object's B position according to object's A position is defined as "*south-west*", distance d_{AB} is:

$$d_{AB} = \sqrt{(y_{min\_b} - y_{\max\_a})^2 + (x_{\min\_a} - x_{\max\_b})^2} \qquad (23)$$

where $(x_{\min\_a}, y_{\max\_a})$ are coordinates of A object's interest area's bottom left corner, $(x_{\max\_b}, y_{\min\_b})$ are coordinates of B object's interest area's upper right corner.

If direction between objects A and B is "*south-east*", distance d_{AB} is calculated appropriately:

$$d_{AB} = \sqrt{(y_{min\_b} - y_{\max\_a})^2 + (x_{min\_b} - x_{\max\_a})^2} \qquad (24)$$

where $(x_{\max\_a}, y_{\max\_a})$ are coordinates of A object's interest area's bottom right corner, $(x_{\min\_b}, y_{\min\_b})$ are coordinates of B object's interest area's upper left corner.

It has to be mentioned that d_{AB} is not the precise distance between objects A and B, it just gives an abstract idea of the position between the two objects. This adjustment's sub-step does not require a precise measuring of distance, because the distance relation is defined by qualitative concepts. In the case, when d_{AB} value is greater than the threshold defined by the thickness of the pen, it is considered that two objects are located *far* from each other. Otherwise, the distance relation between two objects is assumed to be *near*.

3.2.4.3 Determination of Topological Relations

In order to determine the topological relation between two objects there is used the distance relation property assigned in the sub-section above. Topological relations, which are interpreted as Figure 5 shows, are determined according distance relations in a way described below.

If the distance between objects A and B is considered to be *far*, the topological relation between them is determined as *disjoint*. When object B is located near to object A, it is considered that objects *meet* each other. If objects A and B are in *zero* distance from each other, there are several topological relations possible: *contains* (object A contains object B), *inside* (object A inside object B), *equal* (object A equal to object B), *covers* (object A covers object B), *coveredBy* (object A covered by object B), *overlap* (object A overlapped by object B). In this case, there have to be performed some additional actions to determine, which relation of the five six ones is the right one.

Although, in order to determine the topological relation of objects A and B, Equation 1 is used, this algorithm is using a simplified its version:

$$R(A,B)=\begin{pmatrix} A^o \cap B^o & A^o \cap \partial B \\ \partial A \cap B^o & \partial A \cap \partial B \end{pmatrix} \qquad (25)$$

where $A^o \cap B^o$ is the intersection of object's A interior with object's B interior, $A^o \cap \partial B$ represents the intersection of object's A interior with object's B boundary, $\partial A \cap B^o$ is the intersection of object's A boundary with object's B interior and $\partial A \cap \partial B$ notes the intersection of object's *A* boundary with object's *B* boundary.

It is enough to know only these four intersections, because *disjoint* and *meet* relation in the case, when distance is *zero*, is assumed to be not possible. After the application of Equation 25, the interpretation of six relations could be defined as it is shown in Figure 28:

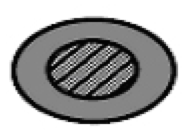
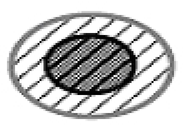
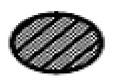
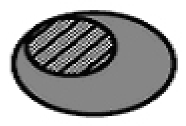
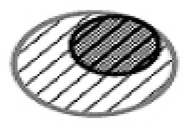
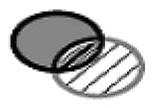
		
$\begin{pmatrix} \neg\varnothing & \neg\varnothing \\ \varnothing & \varnothing \end{pmatrix}$ contains	$\begin{pmatrix} \neg\varnothing & \varnothing \\ \neg\varnothing & \varnothing \end{pmatrix}$ inside	$\begin{pmatrix} \neg\varnothing & \varnothing \\ \varnothing & \neg\varnothing \end{pmatrix}$ equal
		
$\begin{pmatrix} \neg\varnothing & \neg\varnothing \\ \varnothing & \neg\varnothing \end{pmatrix}$ covers	$\begin{pmatrix} \neg\varnothing & \varnothing \\ \neg\varnothing & \neg\varnothing \end{pmatrix}$ coveredBy	$\begin{pmatrix} \neg\varnothing & \neg\varnothing \\ \neg\varnothing & \neg\varnothing \end{pmatrix}$ overlap

Figure 28: The interpretation of 6 topological relations between two objects (after from Egenhofer & Khaled, 1992, p. 200)

When the current topological relation of two objects is determined, there has to be generated a list of other alternative relations according conceptual neighborhoods. The list is needed to perform corrections as fast as possible, if initial adjustment is incorrect. The structure of conceptual neighborhoods is used the one defined by Figure 8. In order to make it more clear, how the list of possible relations looks like and how those relations are interconnected, all eight relations are numbered as it is shown in Figure 29.

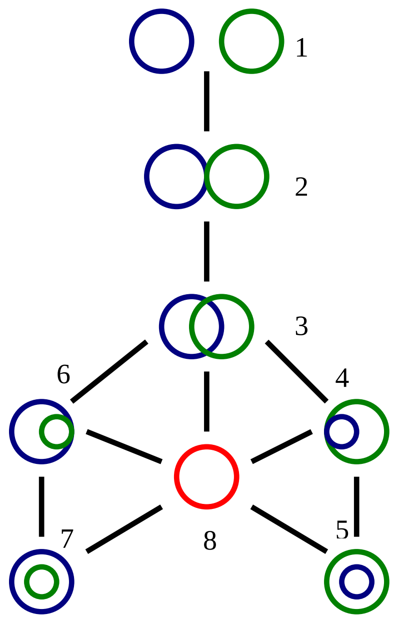


Figure 29: Conceptual neighborhood of topological relations (after Bruns & Egenhofer, 1996, p. 33) , with numbered relations

For each of eight forms of topological relations the list of conceptual neighborhood is generated in a form of transition graph. This graph is needed to perform faster correction if the initial interpretation of a topological relation between two objects is incorrect. Each node of a graph carries the information about the type of relation and its likelihood, which is defined by the properties of the objects.

If the relation between objects A and B is determined to be as *disjoint*, then the transition graph would be generated according this structure:

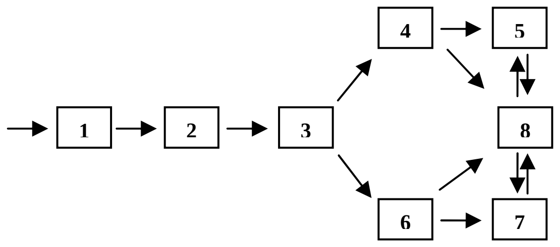


Figure 30: Transition graph for "disjoint" topological relation

As it can be seen from the Figure 30, the transition graph is completely appropriate to the conceptual neighborhood of topological relations presented in Figure 31. The transition between relations initially is clear and easy. When the transition from the relation 3 is needed, then it is checked which transition is more likely to be the correct one.

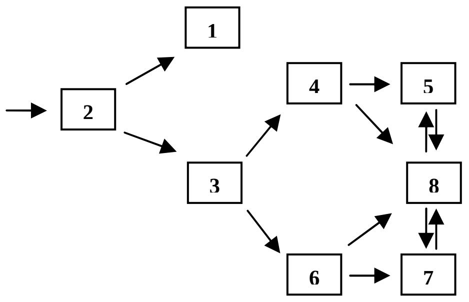


Figure 31: Transition graph for "meets" topological relation

Transition from one relation to the other, if the initial relation is *meets*, is depicted in Figure 33. In comparison with structure presented in Figure 30, this structure gets more complex from the initial state.

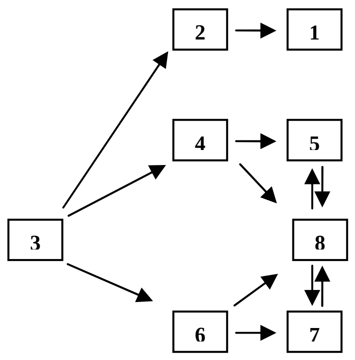


Figure 32: Transition graph for "overlap" topological relation

Figure 32 shows the transition between conceptual neighbors in the case when initial relation is assumed to be *overlap*.

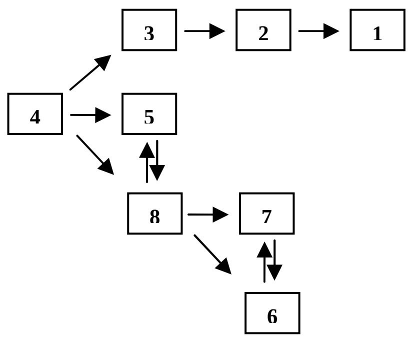


Figure 33: Transition graph for "covers" topological relation

In the case of *covers* relation, transition structure is depicted as it is shown in Figure 33.

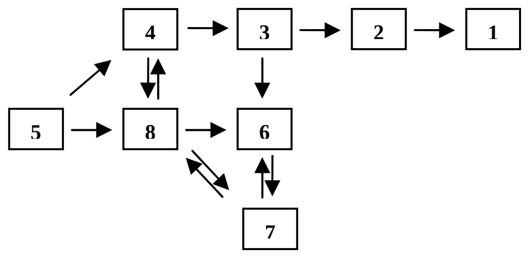


Figure 34: Transition graph for "contains" topological relation

Figure 34 shows a case when initial relation between object is assumed to be *contains*.

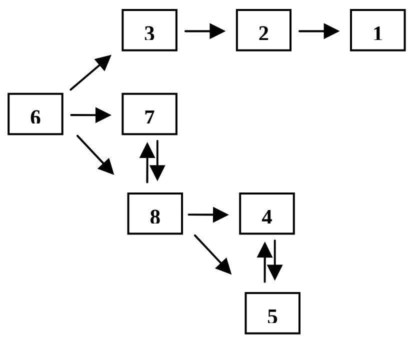


Figure 35: Transition graph for "inside" topological relation

When initial relation is assumed to be *inside*, transition graph is as it is shown in Figure 35. It can be seen that *inside* graph is analogical to transition graph of *contains* relation shown in Figure 33.

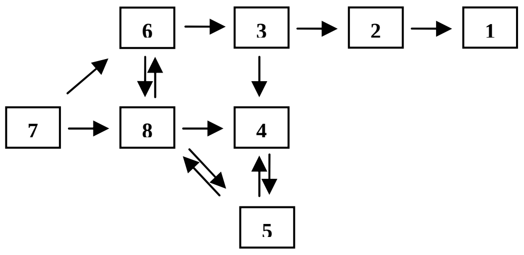


Figure 36: Transition graph for "coveredBy" topological relation

Similarly, transition graph of initial relation *coveredBy* shown in Figure 36 is analogical to the transition graph of relation *covers* depicted in Figure 34.

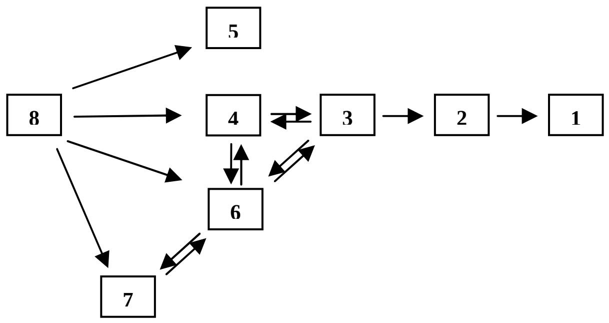


Figure 37: Transition graph for "equal" topological relation

Lastly, Figure 37 presents transition graph, when the initial topological relation is assumed to be *equal*. From the examples given above, it can be claimed, that the more two objects are topologically close to each other, the more complex transition is initially. After all spatial relations are determined; the adjustment of the object can be performed.

3.2.4.4 Adjustment of Object

Adjustment of the recognized and beautified object in the sketch is done according to spatial relations determined above. Basically, the main focus is set on topological relations and other spatial relations are used as supporting information. The adjustments made in every case of topological relation are elucidated.

If the drawn object is *disjoint* with the other object, no adjustments are made, because it is assumed that a user wanted to draw two objects separately from each other. If the drawn object *meets* the reference object, the beautified object is placed exactly near to the reference object in such a way that interest areas of two objects reach each other. It is performed, because it is supposed that user wanted to draw two objects, which are connected to each other. When target object and reference object *overlap*, as in the case with *disjoint* relation no actions are made, because it is hard to distinguish how much overlapping user intended to make, and the current object is placed back into its initial position in the sketch. If target object *contains* the reference object, the target object's area is placed exactly so that appropriate borders of both objects' interest areas would be connected in such a way that center points of those borders would be placed one on the other. Similar adjustments are made in the case when the target object is *covered by* the reference object. When two objects are presumed to be *equal* to each other, the adjustments are made by placing target object in such away that its center point would match reference object's center point. When there is *inside* or *contains* relation between the target and reference objects, the adjustments are made similarly to the ones made for the case of *equal* relation.

When the adjustment of the beautified object is done, the interpretation of the drawn object is proposed to a user. If a user denotes that the interpretation is wrong, the algorithm assumes that drawn object's relation to the target object is the next one from the topological transition graph and performs appropriate adjustments. This action is performed until the user admits the interpretation as the correct one or until the last interpretation offered by the system is displayed.

Chapter 4

Implementation

4.1 Implementation of the Algorithm

This thesis aimed to design algorithm for a system used for dynamic scene analysis and beautification of hand-drawn sketches. In order to prove that designed algorithm is serving the purpose, it is partially implemented in Java. As this thesis is focused on the recognition, beautification and adjustment processes, which are the essential parts of the algorithm, the implementation is basically performed for the Application 2 (Figure 11).

The concept of the implementation is based on the assumption that algorithm gets the screenshot of the SMARTBoard every time the event of an object being drawn is registered. The task of the algorithm is to perform the recognition of the shape and beautification of the object, whereas the adjustment part is left out as it doesn't have much value in partial implementation. Adjustment of the object's position is effective if it is implemented fully with the concept of conceptual neighborhoods. The implementation is partial and is done for only two basic shapes: squares and circles. Basically program loads pictures simulating as if it got the information from Application 1 (Figure 11) and partially applies the designed algorithm. If the system interprets an object as not appropriate shape, there is given a possibility to indicate it to the system in order for it to display other possible shape options.

The source code of the implementation is presented in the Appendix A.

4.1.1 Class Diagram of the Algorithm

In order to make the implementation more clear the class diagram of the program written in Java is provided in Figure 38. There can be seen the structure of the program and all the class used to perform the task, which is defined by the algorithm.

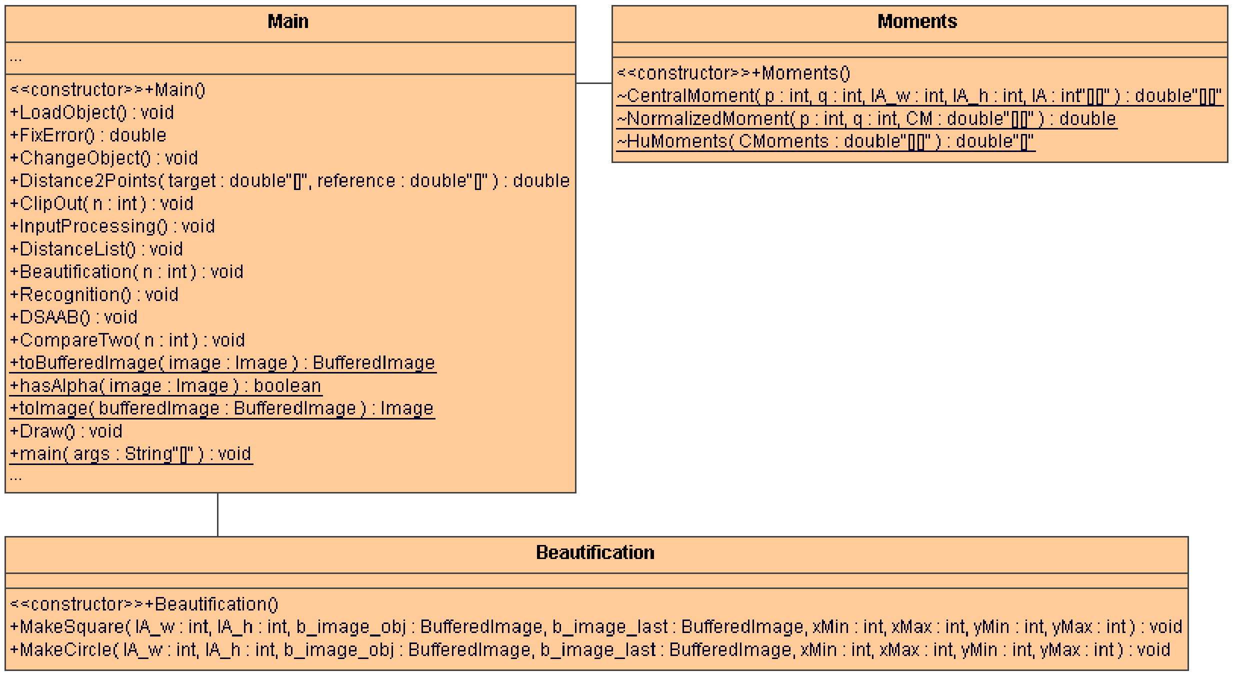

Main
...
<<constructor>>+Main()
+LoadObject() : void
+FixError() : double
+ChangeObject() : void
+Distance2Points(target : double"[]", reference : double"[]") : double
+ClipOut(n : int) : void
+InputProcessing() : void
+DistanceList() : void
+Beautification(n : int) : void
+Recognition() : void
+DSAAB() : void
+CompareTwo(n : int) : void
+toBufferedImage(image : Image) : BufferedImage
+hasAlpha(image : Image) : boolean
+toImage(bufferedImage : BufferedImage) : Image
+Draw() : void
+main(args : String"[]") : void
...
Moments
<<constructor>>+Moments()
~CentralMoment(p : int, q : int, IA_w : int, IA_h : int, IA : int"[][]") : double"[][]"
~NormalizedMoment(p : int, q : int, CM : double"[][]") : double
~HuMoments(CMoments : double"[][]") : double"[]"
Beautification
<<constructor>>+Beautification()
+MakeSquare(IA_w : int, IA_h : int, b_image_obj : BufferedImage, b_image_last : BufferedImage, xMin : int, xMax : int, yMin : int, yMax : int) : void
+MakeCircle(IA_w : int, IA_h : int, b_image_obj : BufferedImage, b_image_last : BufferedImage, xMin : int, xMax : int, yMin : int, yMax : int) : void

Figure 38: Class diagram of a program

As it is shown in Figure 38, in order to partially implement the algorithm and verify it there are used three classes (*Main*, *Recognition* and *Beautification*) , which are here explained in more details.

Main Class is program's main class, responsible for implementing user interface, data handling and managing all the processes of the algorithm. Moreover, this class is responsible for the interaction with a user by reacting to the decisions of the user about the correctness of system's interpretation. One of the most important methods of this class is *DSAAB()*, which is initiated in order to start the process of dynamic scene analyzes and beautification when the image is loaded and user indicates that the object is "drawn". There is no physical fact of drawing, as program loads a new screenshot of a SMARTBoard with a new object by the click of a user. In order to perform a task of scene analysis and beautification methods of *Main* class, such as *Recognition()* and *Beautification()*, initiate methods of other two classes, which are explained in more detail below. When the recognition and beautification process is finished Main class methods displays the result back to the user for the evaluation of its correctness. If a user indicates that algorithm's interpretation of an object is incorrect, a method *ChangeObject()* is initiated in order to provide other options of interpretation to the user. When the recognition and beautification of an object is completed, the user is offered to load another object, as a new and additive object of the previous scene.

Main class consists of these methods:

- *LoadObject()* - this method loads a screenshot imitating the persons action of drawing an object.
- *DSAAB()* – this method handles data and other methods needed for the dynamic scene analysis and beautification. It is initiating other *Main* class methods such as *InputProcessing*(), *Beautification*(), *Draw()* and etc.
- *Distance2Points()* - makes a list of possible interpretations of the drawn object according to the likelihood of its possibility.
- *FixError()* – a method, used to fix the probable Hu moments variance caused by the difference of the sizes between the target object and the reference objects, according to which the Hu moment vectors for the shapes are made.
- *ChangeObject()* – this method changes the current interpretation of user’s drawn object’s shape to the other one which is in the list of possible travelers.
- *Distance2Points()* - counts the Euclidian distance between the object's Hu moments and the reference object's Hu feature vector.
- *ClipOut()* - method which clips out the interest area of the currently drawn object.
- *InputProcessing()* – this method is meant to process initial data for the dynamic scene analysis and beautification with the help of *ClipOut()* and *CompareTwo()*.
- *DistanceList()* - makes a list of possible interpretations of the drawn object according to the likelihood of it.
- *Beautification()* - this method uses Beautification class methods in order to perform appropriate beautification actions towards a recognized object.
- *Recognition()* - uses the methods of Recognition class in order to recognize a recently drawn object.
- *CompareTwo()* – method which points out the changes made in the last screen and finds the object which is drawn recently.
- *toBuffered Image()* - method returns a buffered image with the contents of an image.
- *toImage( )* - this method converts the Buffered Image to an image.
- *hasAlpha()* - method returns true if the specified image has transparent pixels.
- *Draw()* - method is accountable for interpreted object on the screen

Moments Class – this class is intended for the calculation of statistical centralized, normalized and Hu moments with the help of appropriate methods:

- *CentralMoment()* – calculates the central moments of the defined order for the object contained as an image.
- *NormalizedMoment()* – calculates the normalized moment of definite order for the object contained as an image.
- *HuMoment()* – calculates seven Hu moments for the image with the processed object.

Beautification Class – is designed for the beautification step of the algorithm in order to perform object's beautification in the form of a square or a circle using methods:

- MakeSquare() – this method draws a nice and neat square within the boundaries of the processed object's interest area.
- MakeCircle() - this method draws a nice and neat circle within the boundaries of the processed object's interest are.

The instructions and performance, implemented with the help of these three classes partially perform the task defined by the algorithm, is discussed below.

4.1.2 Core-mechanics of the Implementation

4.1.2.1 Instructions for Running the Program

In order to run this algorithm presentation program, a computer is supposed to have Java Virtual Machine (JVM), because the partial algorithm has been programmed in Java language. Moreover, it is important to use Java of 1.4.2 version or higher. When there must be this implementation executed, DSAB.jar file should be executed by clicking on it. It must be ensured that DSAB.jar executable file is in the same directory with the pictures needed for the demonstration of algorithm's performance.

4.1.2.2 Scenario of Using the Program

After the loading of the program is done as it is explained above, a user is presented with the initial program window (Figure 39).

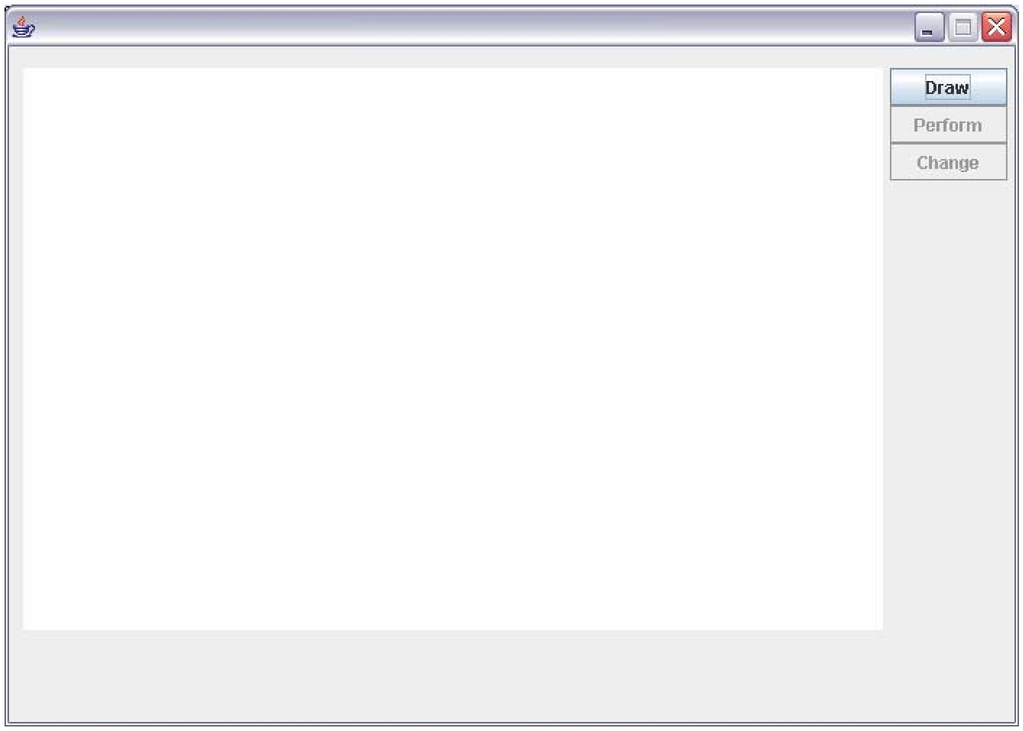
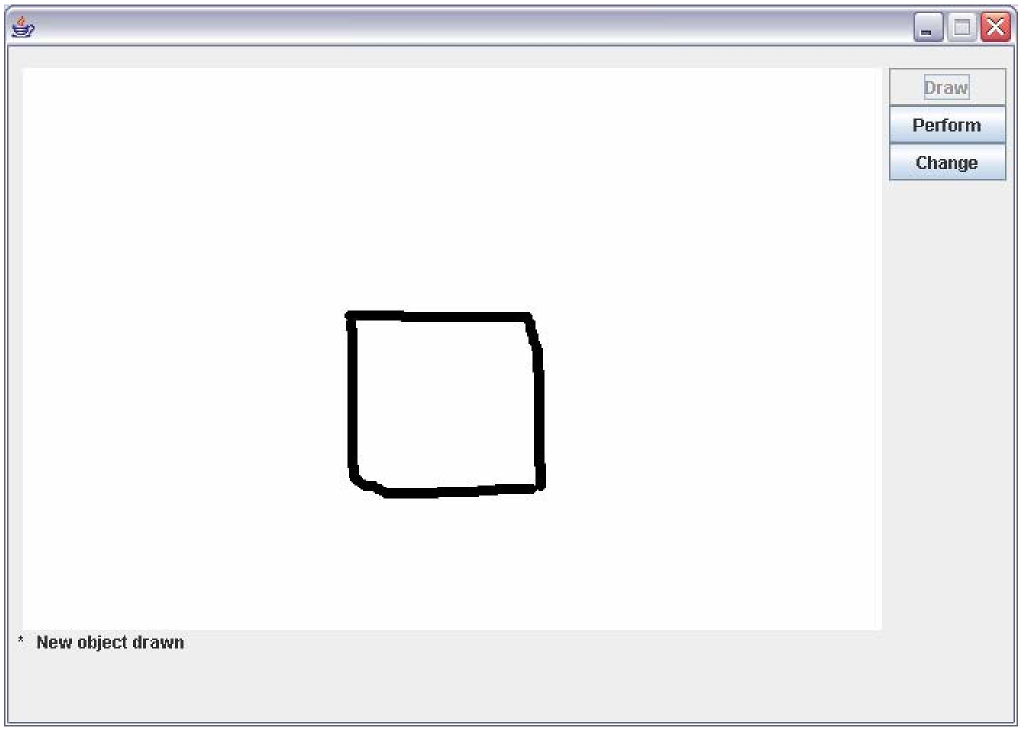


Figure 39: Initial window of a program

As it is shown in Figure 39, the initial window is designed in a simple way and not over-burdened with some additional stuff. The main components of this program are three buttons and one panel. Each of the buttons has its own purpose. For example, button named *Draw* performs loading the new drawn object on the sketch. The other button called *Perform* is used to initialize the dynamic scene analysis. Finally, *Change* button clicked by a user performs the beautification of the other object when user considers current interpretation to be incorrect.

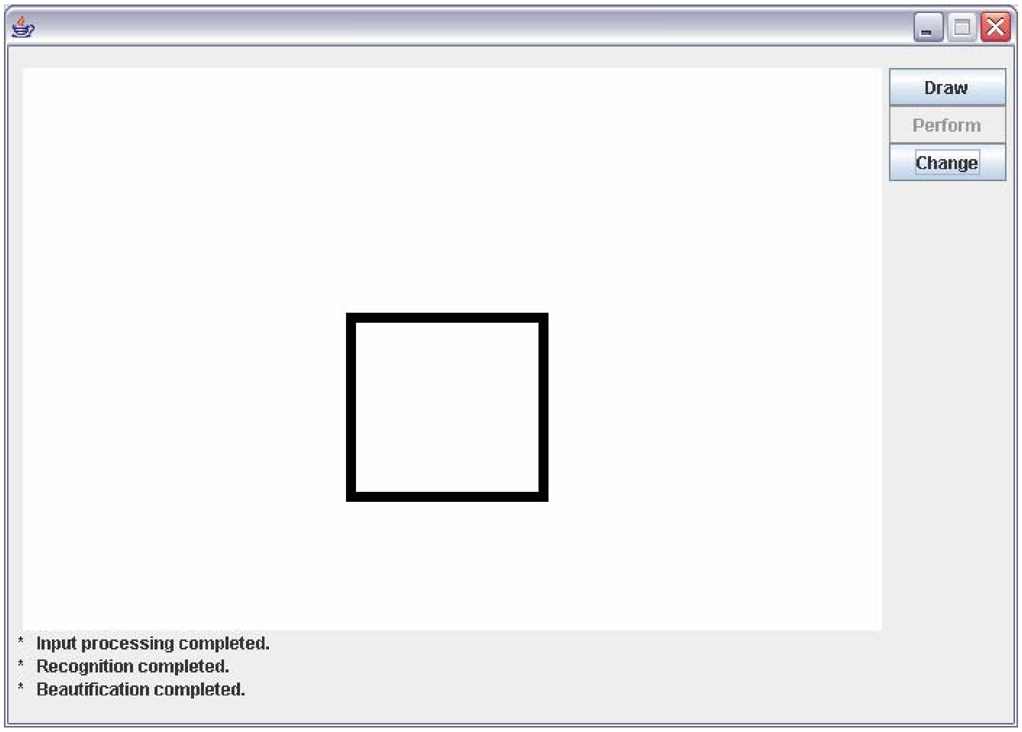
By clicking on the only one enabled button *Draw* the object is drawn on the sketch panel (Figure 40).



* New object drawn

Figure 40: Object loading on the sketch panel

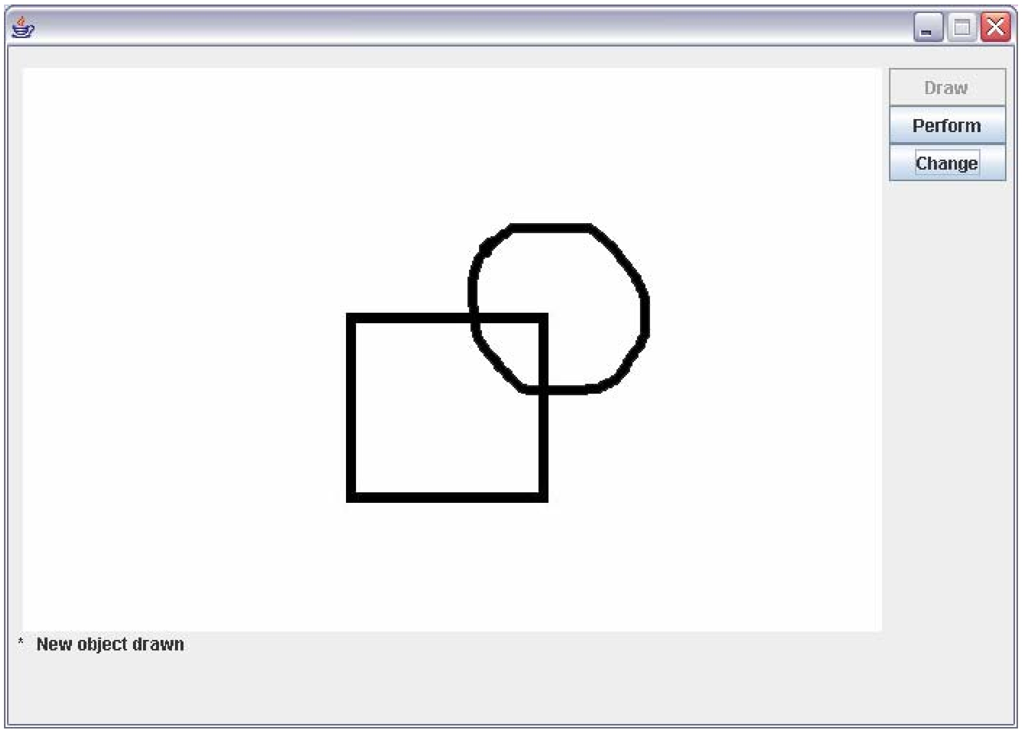
As it is shown in Figure 40, the distorted object is displayed on the sketch screen. By clicking the button *Perform*, it is simulated the event of algorithm getting the data in order to perform dynamic scene analysis and beautification.



* Input processing completed.
* Recognition completed.
* Beautification completed.

Figure 41: Displaying result to a user

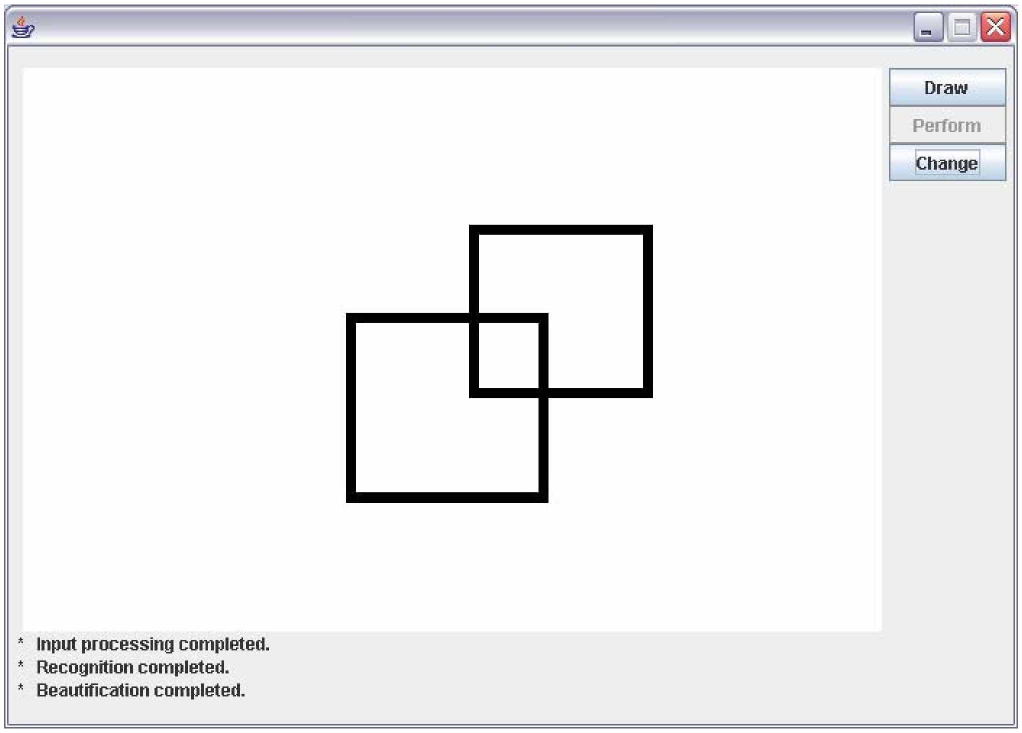
Figure 41 shows a view of systems interpreted object presented to the user after completion of the input data processing, recognition and beautification. If you look at Figure 40, it is seen with the naked eye that the object has been intended to be a square, and the system's interpretation is correct. Further, we can "draw" another object, by clicking on *Draw* button.



* New object drawn

Figure 42: Loading next object on the sketch panel

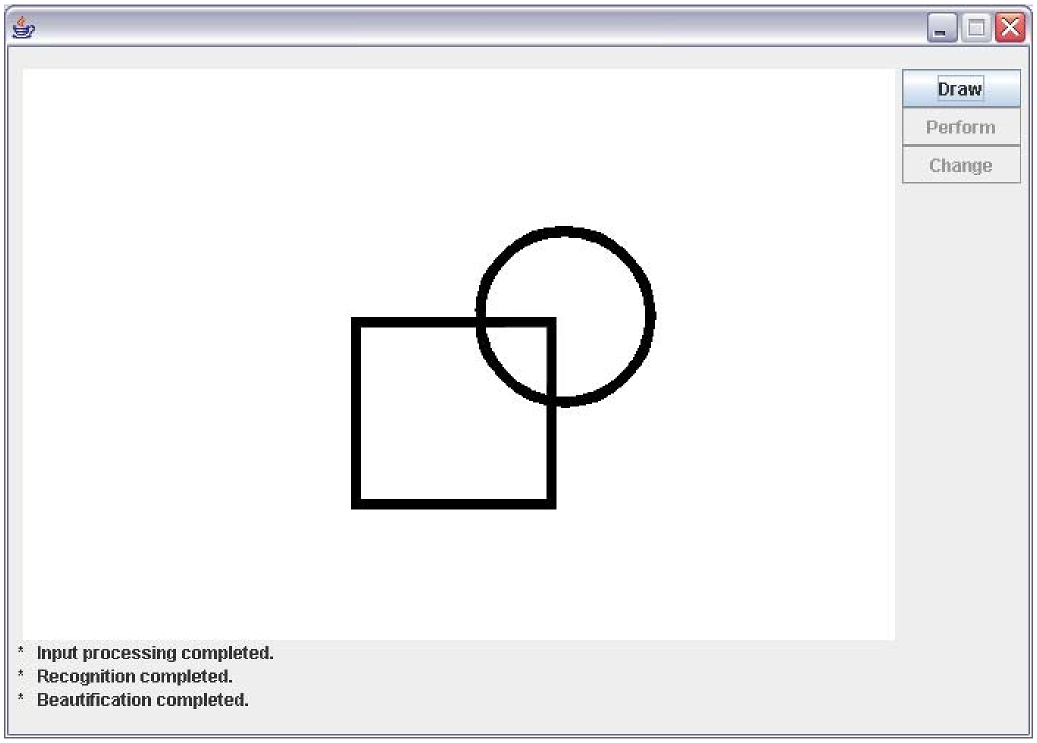
The newly drawn object displayed in Figure 42, seems to be a circle but a much distorted one. By clicking *Perform* button, the scene analysis and beautification is initiated in order to make the sketch more clear and neat.



* Input processing completed.
* Recognition completed.
* Beautification completed.

Figure 43: Display of the incorrect interpretation

Figure 43 shows the result of the recognition of the newly drawn object in Figure 42. It can be seen that the interpretation is incorrect. In this case, a button *Change* should be pressed asking the system to show some other interpretation.



* Input processing completed.
* Recognition completed.
* Beautification completed.

Figure 44: Display of an alternative interpretation of the drawn object

Figure 44 shows the program window which displays the alternative interpretation of a drawn object, when the user indicated to the program that its previous interpretation (Figure 43) is not what the user intended to draw. The displaying of alternative interpretations of the processed object depends on the amount of shapes in a database. In this case, as the database consists of only two shapes, only one alternative interpretation is possible.

This partial implementation showed that the designed algorithm serves its purpose, as it recognizes and beautifies the sketched objects with the possibility of alternative interpretations if the system performed incorrect interpretation.

Conclusions

Although statistical methods are quite efficient in object recognition, the real efficiency depends on the difficulty and the amount of shapes contained in a database of objects’ shapes. The more complex shapes are stored in database and the more intelligent system is designed to be the more complex statistical methods should be used in the process of object recognition. For the basic geometric shapes it is enough to use Hu moments. One of the options for the recognition of more complex shapes would be to use complex Zernike moments. In this case, there should be considered the necessity of optimization of the Zernike moments calculation methods, as the complexity of its calculations augment with the increase of the order of moments. It has been also determined that it is enough to use statistical moments of order ranging from second to eighth, because the increase of the order gives only the significant improvement in recognition of around 1% and the more higher-order moment is the more it is susceptible to noise. Another option for the recognition of complex shapes could be usage of method where Hu moments’ features are combined with Fourier transformation’s features calculated for the object’s skeleton. The complex part of this method is the determination of the skeleton, which could be performed with the help of Voronoi diagrams or other methods. Before choosing any of these methods, there should be considered ratio of the method’s calculations complexity and the efficiency provided by this method.

The beautification process is different for different complexity of the shapes. Beautification of basic shapes requires only basic information about the size of the object and/or the center point of the object, whereas the beautification of more complex shapes can require segmentation of the object and only then the steps of beautification.

Adjustment of the object according to its spatial relations with surrounding objects is the most efficient and recommended to apply only using the method where all three types of spatial relations are used.

Appendix

Source Code of the Implementation Part

The source code of *Main.java*:

```
package dsab;
import java.awt.*;
import java.awt.event.*;
import java.awt.Graphics2D;
import javax.swing.*;
import java.io.*;
import javax.imageio.stream.ImageInputStream;
import java.awt.image.*;
import java.awt.color.*;
import java.lang.Math;
import javax.imageio.*;
import java.awt.geom.*;

public class Main {
	JFrame pagrindas = new JFrame();
	JPanel p1 = new JPanel();
	JPanel p2 = new JPanel();
	JPanel p3 = new JPanel();
	JPanel p4 = new JPanel();
	JPanel p5 = new JPanel();
	JPanel p6 = new JPanel();
	JButton load = new JButton("Draw");
	JButton perform = new JButton("Perform");
	JButton change = new JButton("Change");
	JLabel scr_label = new JLabel();
	static Image image = null;
	JPanel scr = new JPanel();
	String[] pics_List;
	public static int[][] differenceA, objA, IA;
	static double[] HM_IA;
	double[][] HM_shapes = new double[3][8];
```

```
     double[][] CMoments = new double [5][5];
     double[] distance_list = new double[3];
     String[] name_list = new String[3];
     Moments m = new Moments();
     Beautification b = new Beautification();
     double error = 0.000015;
     BufferedImage b_image_obj, b_image_last;
     static int w, h, i, j, count = 0, width = 640, height = 480;
     static int IA_w, IA_h, IA_wn, IA_hn;
     static int xMin[] = new int[20];
     static int yMin[] = new int[20];
     static int xMax[] = new int[20];
     static int yMax[] = new int[20];
     static int counter = 0;
     boolean right = false;
     JLabel status1 = new JLabel(" ");
     JLabel status2 = new JLabel(" ");
     JLabel status3 = new JLabel(" ");
     JLabel status4 = new JLabel(" ");
 /** Creates a new instance of Main */
public Main() {
      // Main panel
     pagrindas.setLayout(new BorderLayout());
     pagrindas.getContentPane().add(p2, BorderLayout.CENTER);
     pagrindas.getContentPane().add(p1, BorderLayout.BEFORE_FIRST_LINE);
     pagrindas.getContentPane().add(p4, BorderLayout.BEFORE_LINE_BEGINS);
     pagrindas.getContentPane().add(p5, BorderLayout.AFTER_LINE_ENDS);
     pagrindas.getContentPane().add(p3, BorderLayout.AFTER_LAST_LINE);
     // Center panel
     p2.setLayout(new GridLayout(1, 1));
     p2.add(scr);
     scr.setSize(640, 480);
     scr.add(scr_label);
     // Button panel
     p6.setLayout(new GridLayout(5, 1));
     p5.add(p6);
     p6.add(load);
     p6.add(perform);
     p6.add(change);
```

```
//Status bar panel
p3.setLayout(new GridLayout(4, 1));
p3.add(status1);
p3.add(status2);
p3.add(status3);
p3.add(status4);
//Setting the screen size
Dimension d = Toolkit.getDefaultToolkit().getScreenSize();
pagrindas.setBounds(0, 0, 700, 500);
pagrindas.setResizable(false);
pagrindas.setVisible(true);
image = new ImageIcon("screen.jpg").getImage();
scr_label.setIcon(new ImageIcon(image));
b_image_last = toBufferedImage(image);
perform.addActionListener( new ActionListener() {
    public void actionPerformed(ActionEvent evt)
    { DSAAB(); } });
change.addActionListener( new ActionListener() {
    public void actionPerformed(ActionEvent evt)
    { ChangeObject(); } });
load.addActionListener( new ActionListener() {
    public void actionPerformed(ActionEvent evt)
    { LoadObject(); } });
// Hu moments feature vectors for a square and circle shapes
HM_shapes[1][1] = 1.6259902276143283;
HM_shapes[1][2] = 0.00831938637943255;
HM_shapes[1][3] = 31.323960471113846;
HM_shapes[1][4] = 3.003469282995453;
HM_shapes[1][5] = -15.317351600218247;
HM_shapes[1][6] = -0.17739450541462043;
HM_shapes[1][7] = 15.940870668599066;
HM_shapes[2][1] = 1.6444676698320924;
HM_shapes[2][2] = 0.011664203402877578;
HM_shapes[2][3] = 33.40893125317925;
HM_shapes[2][4] = 81.2581228164873;
HM_shapes[2][5] = 0.29732616824797675;
HM_shapes[2][6] = 20.254375262463274;
HM_shapes[2][7] = 1.601712657504255;
File dir = new File("Sample/");
```

```
    pics_List = new String[1];
    if ( dir.isDirectory() ){
        pics_List = dir.list();
         }
    perform.setEnabled(false);
    change.setEnabled(false);
}
/* loads a screenshot of "SMARTBoard" provided by Application1
 */
public void LoadObject(){
    if (right == true){
        b_image_last = b_image_obj;
    }
    status1.setText("  *   New object drawn");
    status2.setText(" ");
    status3.setText(" ");
    status4.setText(" ");
    image = new ImageIcon("Sample/" + pics_List[counter]).getImage();
    scr_label.setIcon(new ImageIcon(image));
    scr_label.repaint();
    load.setEnabled(false);
    change.setEnabled(true);
    count = 0;
    right = false;
    perform.setEnabled(true);
}
/* FixError() fixes the probable Hu moments variance caused
 * by the differenece of the sizes between the target object
 * and the reference objects, according to which the Hu moment
 * vectors for the shapes are made.
 */
public double FixError(){
    double fix = 0;
    double d = 0;
    if ((IA_w + IA_h)/2 > 163 )
        d = (-1)*((IA_w + IA_h)/2)/163;
    else
        d = ((IA_w + IA_h)/2)/163;
    if (d == 1)
```

```
        d = 0;
    fix = d * error;
    return fix;
}
/* ChangeObject() changes the interpreted object to the other shape which
 * is the next by the likelihood.
 */
public void ChangeObject(){
    count++;
    b_image_obj = b_image_last;
    Beautification(1 + count);
    Draw();
    change.setEnabled(false);
}
/* Distance2Points(...) counts the Euclidian distance between the object's
 * Hu moments and the reference object's Hu feature vector
 */
public double Distance2Points(double[] target, double[] reference){
    double distance = 0;
    double a = 2;
    double d = 0;
    d = FixError();
    target[1] = target[1] + d;
    distance = Math.sqrt(Math.pow((target[1] - reference[1]),a) + Math.pow((target[2] - reference[2]),a)+
            Math.pow(target[3] - reference[3],a) + Math.pow((target[4] - reference[4]),a) +
            Math.pow((target[5] - reference[5]),a) + Math.pow((target[6] - reference[6]),a) +
            Math.pow((target[7] - reference[7]),a));
    return distance;
}
/* ClipOut(...) clips out the interest area of the currently drawn
 * object.
 */
public void ClipOut(int n){
    xMin[n] = new Integer(10000);
    xMax[n] = new Integer(0);
    yMin[n] = new Integer(10000);
    yMax[n] = new Integer(0);
    for (int i = 1; i <= width; i++){
        for (int j = 1; j <= height; j++) {
```

```
                        if (differenceA[i][j] == 1){
                            if (i < xMin[n])
                                xMin[n] = i;
                            if (i > xMax[n])
                                xMax[n] = i;
                            if (j < yMin[n])
                                yMin[n] = j;
                            if (j > yMax[n])
                                yMax[n] = j;
                        }
                    }
                }
                IA_h = yMax[n] - yMin[n] + 1;
                IA_w = xMax[n] - xMin[n] + 1;
                IA = new int[IA_w + 1][IA_h + 1];
                for (int i = 1; i <= IA_w; i++){
                    for (int j = 1; j <= IA_h; j++) {
                        IA[i][j] = differenceA[xMin[n] - 1 + i][yMin[n] - 1 + j];
                    }
                }
                Graphics2D g2d = b_image_obj.createGraphics();
                g2d.setColor(Color.red);
                for (int i = xMin[n]; i <= xMax[n]; i++){
                    g2d.drawRect(i - 1, yMin[n] - 1, 1, 1 );
                    g2d.drawRect(i - 1, yMax[n] - 1, 1, 1 );
                }
            }
            /* Processes initial data for the dynamic scene analysis and beautification.
             */
            public void InputProcessing(){
                b_image_obj = toBufferedImage(image);
                CompareTwo(counter);
                ClipOut(counter);
            }
            /* DistanceList() makes a list of possible interpretations of the drawn object
             * according to the likelihood of it
             */
            public void DistanceList(){
                double a = Distance2Points(HM_IA, HM_shapes[1]);
```

```
        double b = Distance2Points(HM_IA, HM_shapes[2]);
        if (a < b) {
            distance_list[1] = a;
            name_list[1] = "square";
            distance_list[2] = b;
            name_list[2] = "circle";
        }
        else {
            distance_list[1] = b;
            name_list[1] = "circle";
            distance_list[2] = a;
            name_list[2] = "square";
        }
    }
    /* Beautification() he method which uses Beautification class methods in
     * order to perform appropriate beautification actions towards a recognized
     * object
     */
    public void Beautification(int n){
        if (name_list[n].equals("square"))
        {
            b.MakeSquare(IA_w, IA_h, b_image_obj, b_image_last, xMin[counter], xMax[counter],
yMin[counter], yMax[counter]);
        }
        else
            b.MakeCircle(IA_w, IA_h, b_image_obj, b_image_last, xMin[counter], xMax[counter],
yMin[counter], yMax[counter]);
    }
    /* Recognition() uses the metods of Recognition class in order to
     * recognize a recently drawn object
     */
    public void Recognition(){
        CMoments = m.CentralMoment(3, 3, IA_w, IA_h, IA);
        HM_IA = m.HuMoments(CMoments);
    }
    /* The method which encompases all the parts of algorithm
     */
    public void DSAAB(){
        counter++;
        perform.setEnabled(false);
        status1.setText("  *   Input processing...");
        status1.repaint();
        InputProcessing();
        status1.setText("  *   Input processing completed.");
        status1.repaint();
        status2.setText("  *   Recognition...");
        status2.repaint();
        Recognition();
        DistanceList();
        status2.setText("  *   Recognition completed.");
        status2.repaint();
        status3.setText("  *   Beautification...");
        status3.repaint();
        Beautification(1 + count);
        status3.setText("  *   Beautification completed.");
        status3.repaint();
        Draw();
        load.setEnabled(true);
    }
    /* CompareTwo() method points out the changes made in the last screen and
     * finds the object which is drawn recently.
     */
    public void CompareTwo(int n){
        int i, j;
        differenceA = new int[width + 1][height + 1];
        for (i = 1; i <= width; i++){
            for (j = 1; j <= height; j++) {
                if (((b_image_obj.getRGB(i - 1, j - 1) == -16777216) ||(b_image_obj.getRGB(i - 1, j - 1) == -
16711423) || (b_image_obj.getRGB(i - 1, j - 1) < -1100000))  & (b_image_last.getRGB(i - 1, j - 1) == -1))
                {
                    differenceA[i][j] = 1;
                }
                else
                {
                    differenceA[i][j] = 0;
                }
            }
        }
```

```
}
/* This method returns a buffered image with the contents of an image
 */
public static BufferedImage toBufferedImage(Image image) {
    if (image instanceof BufferedImage) {
        return (BufferedImage)image;
    }
    image = new ImageIcon(image).getImage();
    boolean hasAlpha = hasAlpha(image);
    BufferedImage bimage = null;
    GraphicsEnvironment ge = GraphicsEnvironment.getLocalGraphicsEnvironment();
    try {
        int transparency = Transparency.OPAQUE;
        if (hasAlpha)
        {
            transparency = Transparency.BITMASK;
        }
        GraphicsDevice gs = ge.getDefaultScreenDevice();
        GraphicsConfiguration gc = gs.getDefaultConfiguration();
        bimage = gc.createCompatibleImage(image.getWidth(null), image.getHeight(null), transparency);
    }
    catch (HeadlessException e) {}
    if (bimage == null) {
        // Create a buffered image using the default color model
        int type = BufferedImage.TYPE_INT_RGB;
        if (hasAlpha) {
            type = BufferedImage.TYPE_INT_ARGB;
        }
        bimage = new BufferedImage(image.getWidth(null), image.getHeight(null), type);
    }
    Graphics g = bimage.createGraphics();
    g.drawImage(image, 0, 0, null);
    g.dispose();
    return bimage;
}
/* This method returns true if the specified image has transparent pixels
 */
public static boolean hasAlpha(Image image) {
    if (image instanceof BufferedImage) {
            BufferedImage bimage = (BufferedImage)image;
            return bimage.getColorModel().hasAlpha();
        }
        // Use a pixel grabber to retrieve the image's color model;
        // grabbing a single pixel is usually sufficient
        PixelGrabber pg = new PixelGrabber(image, 0, 0, 1, 1, false);
        try {
            pg.grabPixels();
        }
        catch (InterruptedException e) { }
        ColorModel cm = pg.getColorModel();
        return cm.hasAlpha();
    }
    /* This method converts the Buffered Image to an image
     */
    public static Image toImage(BufferedImage bufferedImage) {
        return Toolkit.getDefaultToolkit().createImage(bufferedImage.getSource());
    }
    /* Draw() method displayes interpreted object on the screen
     */
    public void Draw(){
          image = toImage(b_image_obj);
          scr_label.setIcon(new ImageIcon(image));
          scr_label.repaint();
          File f = new File ("anImage.jpg");
          try
          { ImageIO.write (b_image_obj, "jpeg", f);}
          catch(IOException e){}
    }
    public static void main(String[] args) {
        final Main a = new Main();
    }
}
```

The source code of *Moments.java*:

```
package dsab;
import java.lang.Math;
```

```
public class Moments {
    /** Creates a new instance of Moments */
    public Moments() {
    }
    /* This method estimates the Central Moments
     */
    static double[][] CentralMoment(int p, int q, int IA_w, int IA_h, int[][] IA){
        double[][] R = new double[p + 2][q + 2];
        double sum;
        double sx, sy, sxl, syl, m00 = 0, m01 = 0, m10 = 0;
        for (int i = 1; i <= IA_w; i++){
             for (int j = 1; j <= IA_h; j++){
                m00 += IA[i][j];
                m10 += IA[i][j]*i;
                m01 += IA[i][j]*j;
             }
        }
        sx = m10/m00;
        sy = m01/m00;
        for (int ip = 0; ip <= p; ip++){
            for (int iq = 0; iq <= q; iq++){
                sum = 0;
                for (int i = 1; i <= IA_h; i++){
                    for (int j = 1; j <= IA_w; j++){
                        sxl = Math.pow((j - sx), ip);
                        syl = Math.pow((i - sy), iq);
                        sum = sum + sxl*syl*IA[j][i];
                    }
                }
                R[iq + 1][ip + 1] = sum;
            }
        }
        return R;
    }
    /* This method estimates the normalized moment M_pq
     */
    static double NormalizedMoment(int p, int q, double[][] CM){
        double alpha = (p + q + 2)/2.0 ;
        double momval = CM[p + 1][q + 1];
        double norm = 1;
            for (int i = 1; i <= alpha; i++)
                norm = norm * CM[1][1];
            if (norm == 0)
                momval = 0;
            else
                momval = momval/norm;
        return momval;
    }
    // estimates the Hu moments for a given area of pixel
    static double[] HuMoments(double[][] CMoments){
        double[] HM_IA = new double[8];
        int xMin = Integer.MAX_VALUE;
        int xMax = Integer.MIN_VALUE;
        int yMin = Integer.MAX_VALUE;
        int yMax = Integer.MIN_VALUE;
        double dx;
        double dy;
        double n01 = NormalizedMoment(0, 1, CMoments);
        double n02 = NormalizedMoment(0, 2, CMoments);
        double n03 = NormalizedMoment(0, 3, CMoments);
        double n10 = NormalizedMoment(1, 0, CMoments);
        double n11 = NormalizedMoment(1, 1, CMoments);
        double n12 = NormalizedMoment(1, 2, CMoments);
        double n20 = NormalizedMoment(2, 0, CMoments);
        double n21 = NormalizedMoment(2, 1, CMoments);
        double n30 = NormalizedMoment(3, 0, CMoments);
        HM_IA[1] = n20 + n02;
        HM_IA[2] = ((n20 - n02) * (n20 - n02)) + (4 * n11 * n11);
        HM_IA[3] = ((n30 - (3 * n12)) * (n30 - (3 * n12))) + ((n03 - (3 * n21)) * (n03 - (3 * n21)));
        HM_IA[4] = ((n30 + n12) * (n30 + n12)) + ((n03 + n21) * (n03 + n21));
        HM_IA[5] = ((n30 - (3 * n12)) * (n30 + n12) * (((n30 + n12) * (n30 + n12)) -  (3 * (n21 + n03) *
                   (n21 + n03)))) + ((n03 - (3 * n21)) * (n03 + n21) * (((n03 + n21) * (n03 + n21)) -
                   (3 * (n12 + n30) * (n12 + n30))));
        HM_IA[6] = ((n20 - n02) * (((n30 + n12) * (n30 + n12)) - ((n03 + n21) * (n03 + n21)))) + (4 * n11 * (n30
+ n12) * (n03 + n21));
        HM_IA[7] = (((3 * n21) - n03) * (n30 + n12) * (((n30 + n12) * (n30 + n12)) - (3 * (n21 + n03) * (n21 +
n03))))
```

```
                  - (((3 * n12) - n30) * (n03 + n21) * (((n03 + n21) * (n03 + n21)) - (3 * (n12 + n30) * (n12 +
n30))));
        return HM_IA;
    }
}
```

The source code of *Beautification.java*:

```
package dsab;
import java.awt.*;
import java.awt.event.*;
import java.awt.Graphics2D;
import javax.swing.*;
import java.io.*;
import java.awt.image.*;
import java.awt.color.*;
import java.lang.Math;
import javax.imageio.*;
import java.awt.geom.*;

public class Beautification {

    /** Creates a new instance of Beautification */
    public Beautification() {
    }
    /* Beautification of the object representing a square
     */
    public void MakeSquare(int IA_w, int IA_h, BufferedImage b_image_obj, BufferedImage b_image_last,
int xMin, int xMax, int yMin, int yMax)
   {
        Main.IA_wn = (IA_w + IA_h)/2;
        Main.IA_hn = Main.IA_wn;
        Graphics2D g2d = b_image_obj.createGraphics();
        g2d.setColor(Color.white);
        for (int i = 0; i <= IA_w; i++){
            for (int j = 0; j <= IA_h; j++){
                if (b_image_last.getRGB(xMin + i - 2, yMin + j - 2) == -1 )
                {
                    g2d.drawRect(xMin + i - 2, yMin + j - 2, 1, 1);}
                 }
        }
        // Draw on the image
        g2d.setColor(Color.black);
        for (int i = 0; i < 7; i++){
            g2d.drawRect(xMin + i, yMin + i, (xMax - xMin + 1 - 2*i), (yMax - yMin + 1 - 2*i));
        }
        g2d.setColor(Color.black);
        for (int i = xMin - 2; i <= xMax; i++) {
            for (int j = yMin - 2; j <= yMax; j++) {
                if (b_image_last.getRGB(i, j) == -16777216)
                    b_image_obj.setRGB(i, j, -16777216);
            }
        }
        g2d.dispose();
    }
    /* Beautification of the object representing a circle
     */
    public void MakeCircle(int IA_w, int IA_h, BufferedImage b_image_obj, BufferedImage b_image_last,
int xMin, int xMax, int yMin, int yMax)
    {
        Main.IA_wn = (IA_w + IA_h)/2;
        Main.IA_hn = Main.IA_wn;
        Graphics2D g2d = b_image_obj.createGraphics();
        g2d.setColor(Color.white);
        for (int i = 0; i <= IA_w; i++){
            for (int j = 0; j <= IA_h; j++){
                if (b_image_last.getRGB(xMin + i - 2, yMin + j - 2) == -1 )
                {
                    g2d.drawRect(xMin + i - 2, yMin + j - 2, 1, 1);}
                 }
        }
        g2d.setColor(Color.black);
        if (Main.IA_wn%2 == 0)
            Main.IA_wn++;
        g2d.setColor(Color.black);
        g2d.fill(new Ellipse2D.Float(xMin - 1, yMin - 1, Main.IA_wn, Main.IA_wn));
        g2d.setColor(Color.white);
        g2d.fill(new Ellipse2D.Float(xMin + 6, yMin + 6, Main.IA_wn - 14, Main.IA_wn - 14));
        g2d.setColor(Color.black);
        for (int i = xMin - 2; i <= xMax + 2; i++) {
            for (int j = yMin - 2; j <= yMax + 2; j++) {
                if (b_image_last.getRGB(i, j) == -16777216)
                    b_image_obj.setRGB(i, j, -16777216);
            }
        }
        g2d.dispose();
    }
}
```

Bibliography

Allen J. (1983). *Maintaining Knowledge about Temporal Intervals.* Communications of the ACM 26 (11), p. 832-843, 1983.

Bruns H. T., Egenhofer M. (1996). *Similarity of Spatial Scenes.* Seventh International Symposium on Spatial Data Handling (SDH '96), p. 4A.31-42, 1996

Cheng P., Lowe R., Scaife M. (2001). *Cognitive Science Approaches to Understanding Diagrammatic Representations.* Thinking with Diagrams, Kluwer Academic Publishers, Dordrecht, 2001

Cheng P., Herbert S. (1993). *Scientific Discovery and Creative Reasoning with Diagrams.* The Creative Cognition Approach, Cambridge, MA, 1993

Cohn A.G, Bennett B., Gooday J., Gotts N.M. (1997). *Representing and Reasoning With Qualitative Spatial Relations about Regions*. Temporal and spatial reasoning, 1997

Cohn A. G., Hazarika S. M. (2001). *Qualitative Spatial Representation and Reasoning: An Overview*. Fundamenta Informaticae 43, 2001, p. 2 -32.

David M., Egenhofer M. (1994). *Modeling Spatial Relations Between Lines and Regions: Combining Formal Mathematical Methods and Human Subjects Testing.* Cartography and Geographical Information Systems 21, p. 195 - 212, 1994

Dylla F., Moratz R. (2005). *Exploiting Qualitative Spatial Neighborhoods in the Situation Calculus.* Spatial Cognition IV Reasoning, Action, Interaction Vol. 3343, pp. 304 – 322, 2005

Egenhofer M., Khaled K. A. (1992). *Reasoning about Gradual Changes of Topological Relationships.* Theory and Methods of Spatio-Temporal Reasoning in Geographic Space, volume 639 of Lecture Notes in Computer Science, p. 196 – 219**,** 1992**.**

Egenhofer M., Herring J. (1990). *Categorizing Binary Topological Relations Between Regions, Lines, and Points in Geographic Databases.* Technical Report, Department of Surveying Engineering, University of Maine, 1990

Frank A. (1996). *Qualitative Spatial Reasoning: Cardinal Directions as an Example.* International Journal of Geographical Information Science, 1996

Freksa C. (1991a). *Conceptual Neighborhood and its Role in Temporal and Spatial Reasoning.* Workshop on Decision Support Systems and Qualitative Reasoning, p. 181-187, Amsterdam, North-Holland, 1991.

Freksa C. (1991b). *Qualitative Spatial Reasoning*. Cognitive and Linguistic Aspects of Geographic Space, p. 361-372.

Furnas G. W. (1992). *Reasoning with Diagrams Only*. AAAI Symposium on Reasoning with Diagrammatic Representations, 1992

Goyal R., Egenhofer M. (2001). *Similarity of Cardinal Directions.* Seventh International Symposium on Spatial and Temporal Databases, Lecture Notes in Computer Science Vol. 2121, p. 36-55, 2001

Gurr C. A. (1994). *Diagrams and Human Reasoning*, 1994

Gurr C. A. (1999). *Effective Diagrammatic Communication: Syntatic, Semantic and Pragmatic Issues.* Journal of Visual Languages and Computing Vol. 10, p. 317-342, 1999

Hong J., Egenhofer M., Frank A. (1995). *On the Robustness of Qualitative Distance- and Direction-Reasoning.* Twelfth International Symposium on Computer- Assisted Cartography, 1995

Hse H. H., Newton A. R. (2004). *Sketched Symbol Recognition using Zernike Moments*. International Conference on Pattern Recognition , 2004

Hse H. H., Newton A. R. (2005). *Recognition and Beautification of Multi-Stroke Symbols in Digital Ink*. Computers & Graphics, 2005

Hu M. (1962). *Visual Pattern Recognition by Moment Invariants*. IEEE Transactions on Information Theory, 8:179--187, 1962

Nabil M., Shepherd J., Ngu A. (1996). *2D Projection Interval Relationships: A Symbolic Representation of Spatial Relationships.* Symposium on Large Spatial Databases, 1996

Papadias D., Mamuolis N., Delis V. (2001). *Approximate Spatio-Temporal Retrieval.* ACM Transactions on Information Systems Vol. 19 (1), p. 53-96, 2001

Poppe R., Poel M. (2006). Comparison of Silhouette Shape Descriptors for Example-based Human Pose Recovery. 2006

Rauh R., Hagen C., Schlieder C., Strube G., Knauff M. (2000). *Searching for alternatives in spatial reasoning: Local transformations and beyond.* Proceedings of the Twenty Second Annual Conference of the Cognitive Science Society, p. 871-876, 2000.

Scaife M., Rogers Y. (1996). *External Cognition: How Do Graphical Representations Work?* International Journal of Human-Computer Studies, p. 185 – 213, 1996

Schlieder C. (1996). *Qualitative Shape Representation.* Proceedings of GISDATA Specialist Meeting on Geographical Objects with Undetermined Boundaries, 1996

Schutler J. (2002). *Statistical Moments*. http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/SHUTLER3/, 03 05 2006

Sharma J., Flewelling D., Egenhofer M. (1994). *A Qualitative Spatial Reasoner*. Sixth International Symposium on Spatial Data Handling, Edinburgh, Scotland, p. 665 - 681, 1994

Taubman G. (2005). *MusicHand: A Handwritten Music Recognition System*. Thesis. Brown University, 2005

Theodoridis Y., Papadias D., Stefanakis E. (1996). *Supporting Direction Relations in Spatial Database Systems.* Proceedings of the 7th International Symposium on Spatial Data Handling (SDH'96), 1996