Západočeská univerzita

Fakulta aplikovaných věd

Soubor vybraných prací
ke jmenovacímu řízení profesorem
pro obor

Informatika a výpočetní technika

10. dubna, 1995

Doc.Ing.Václav Skala,CSc.

Seznam publikací a citací

Doc.Ing.Václav Skala, CSc.
Katedra informatiky a výpočetní techniky
Západočeská univerzita
Americká 42, Box 314, 306 14 Plzeň
Tel. +019-2171-188 E-mail skala@kiv.zcu.cz

---------- 1985 - 1995 ----------

Publikace do r.1985 jsou přiloženy ve zvláštním seznamu
Známé citace jsou vždy uvedeny u příslušné publikace

--- 1985 ---

[S85a] Skala,V.: An Interesting Modification to the Bresenham Algorithm, NATO ASI 1985, in Fundamental Algorithms for Computer Graphics (Ed. R.A.E.Earnshaw), Springer Verlag, 1985.

Kramarczyk,W.G.: Efficient Raster Graphing of Bivariate Functions by Incremental Methods, 1988.

Alvisi,L., Casciola,G.: Two and Four Array Mask Algorithm in Practice, Technical Report, Univ. of Bologna, Italy, pp.82, 1988.

Alvisi,L., Casciola,G.: On the Two Array Mask Hidden-Line Algorithms, C & G, Vol.13, No.2, pp.193-206, 1989.

Andreev,R.D.: Graphics Systems: Architecture and Realization, North Holland Comp., 1993.

Amoroso,A., Casciola,G.: A Hidden Line Algorithm for Spherical Coordinate Functions, Technical Report CNR grant, Univ.of Bologna, Italy, 1994.

Amoroso,A., Casciola,G.: A Hidden Line Algorithm for Spherical Coordinate Functions, Computers & Graphics, 1994.

INSPEC 87 rec.2877384.

--- 1987 ---

[S87a] Skala,V.: Hidden-Line, Hidden-Surface, Hidden-Contouring Problem. Problem Solutions by the Modified Bresenham's algorithm, Electronic Displays'87, London, 1987.

[S87b] Skala,V.: Hidden-Line, Hidden-Surface and Hidden-Contouring Problem Solutions by the modified Bresenham's Algorithm, NATO ASI, Springer Verlag, 1987.

Schumann,H.: PhD Thesis, Univ. Rostock, 1988.

[S87c] Skala,V.: An Intersecting Modification to the Bresenham Algorithm, Computer Graphics Forum, North Holland Comp., Vol.6., No.4, 1987.

Alvisi,L., Casciola,G.: TAM rivisato: un metodo rapido ed esatto per la rappresentazione prospettica di superfici, PIXEL No.10,pp, 12-15, 1988.

Alvisi,L., Casciola,G.: Two and Four Array Mask Algorithm in Practice, Technical Report, Univ. of Bologna, Italy, pp.82, 1988.

Alvisi,L., Casciola,G.: On the Two Array Mask Hidden-Line Algorithms, Computers & Graphics, Vol.13, No.2, pp.193-206, 1989.

INSPEC 88 rec.3100571.

COMPENDEX 88 rec.0505883.

[S87d] Skala,V., Tesař,R., Zdráhal,K.: Zařízení pro měření účiníku, CS Patent No.239283, 1987.

--- 1988 ---

[S88a] Skala,V.: Filling and Hatching Operations for Non-Convex Areas with Conic Edges for the Raster Environment, ACM-SIGGRAPH Workshop Lisboa Local Group, Lisboa, Portugal, 1988.

Schumann,H.: PhD Thesis, Univ. Rostock, 1988.

[S88b] Skala,V.: Algorithms for 2D Line Clipping, ACM SIGGRAPH Workshop Lisboa Local Group, Lisboa, Portugal, 1988.

[S88c] Skala,V.: An Interesting Modification to the Bresenham Algorithm, ACM SIGGRAPH Workshop Lisboa Local Group, Portugal, 1988.

[S88d] Skala,V.: Základní grafiká instrukce pro kreslení ploch s respektováním viditelnosti, Microsystem'88, Mezinárodní konference, Bratislava, 1988.

--- 1989 ---

[S89a] Skala,V., Tesař,R.: Zařízení pro indikaci rozdílu žádaného a skutečného fázového posuvu, CS Patent No.251399, 1989.

[S89b] Skala,V.: Algorithms for 2D Line Clipping, in New Advances in Computer Graphics, (Ed.R.A.Earnshaw,B.Wyvill) Proceedings of Computer Graphics International'89, Springer Verlag, 1989.

Kolingerová, I.: Algoritmus pro ořezávání úsečky vůči rovinné oblasti, Konference Algoritmy 91, Vysoké Tatry, 1991.

Kolingerová,I.: Duální reprezentace a její využití v počítačové grafice, disertační práce, ZČU, Plzeň, 1994

INSPEC 90 rec. 3577896.

[S89c] Skala,V.: A Unifying Approach to the Line Clipping Problem Solution, TR 209-5-89, pp.1-16, 1989.

[S89d] Skala,V.: Recenze - Earnshaw,R.A.(Ed.): Theoretical Foundations of Computer Graphics and CAD, Springer Verlag, 1988 - Kybernetika, Vol.25, No.5, pp.436-437, 1989.

[S89e] Skala,V.: A Unifying Approach to the Line Clipping Problem Solution, SIAM Conference on Geometric Design, TEMPE AZ, USA, 1989.

[S89f] Skala,V.: A Unifying Approach to the Line Clipping Problem Solution, BISYCP'89, Beijing, 1989.

[S89g] Skala,V.: Hidden-Line, Hidden-Surface Problem Solutions by the modified Bresenham's Algorithm, 4-th Conference on Computer Graphics'89, Smolenice, Czechoslovakia.

[S89h] Skala,V.: Algorithms for 2D Line Clipping, in EUROGRAPHICS'89 Proceedings (Ed.W.Hansmann, F.R.A.Hopgood, W.Strasser), North Holland, 1989.

Liang,Y.D., Barsky, B.A.: The Optimal Tree Algorithm for Line Clipping, Technical Report, Univ. of California, Berkeley, 1992.

Krammer,G.: A Line Clipping Algorithm and its Analysis, EG'92, 1992.

Kolingerová,I.: Duální reprezentace a její využití v počítačové grafice, disertační práce, ZČU, Plzeň, 1994

Nielsen, H.,P.: Line Clipping Using Semi-homogeneous Coordinates, CGF accepted for publication, 1993.

Bukert,A.: Clipping an allgenmeinen dreidimensionalen Polyedrn, PhD Thesis, FhG IGD Darmstadt, 1992.

Owen,J.: Computer Graphics and CAD Literature - 1989, Computer Graphics, 1990

INSPEC 90 rec.3668038.

[S89i] Skala,V.: Hidden-Line, Hidden-Surface and Hidden-Contouring Problem Solutions by the Modified Bresenham's Algorithm, mezinárodní konference Algoritmy'89, pp. 280-288, Vysoké Tatry 1989.

--- 1990 ---

[S90a] Skala,V.: General Conics Clipping - Problem Solution, YUGRAPH'90, Dubrovnik, Yugoslavia, pp.25-29, June 20-22, 1990.

[S90b] Skala,V.: Algorithms for Clipping Quadratic Arcs, Computer Grahics International'90, June 27-29, 1990.

Kolingerová, I.: Algoritmus pro ořezávání úsečky vůči rovinné oblasti, Konference Algoritmy 91, Vysoké Tatry, 1991.

Database ISTP, Institution for Scientific Information Inc., 1992 - record.

[S90c] Skala,V.: Počítačová grafika I, skripta VŠSE, Plzeň, pp. 146, 1990.

Kolingerová, I.: Algoritmus pro ořezávání úsečky vůči rovinné oblasti, Konference Algoritmy 91, Vysoké Tatry, 1991.

[S90d] Skala,V.: Počítačová grafika II, skripta VŠSE, Plzeň, pp.142, 1990.

Ertl,J., Ferko,A.: Normalizované grafické systémy, skripta MFF UKo, Bratislava, 1993.

Kolingerová,I.: Programová realizace metod půltónování, Konference Algoritmy 91, Vysoké Tatry , 1991.

--- 1991 ---

[S91a] Skala,V.: Počítačová grafika - Algoritmy ořezávání, habilitační práce, ZČU, Plzeň1991.

[S91b] Kolingerová,I., Skala,V.: Metody půltónování a jejich srovnání, mezinárodní konference Algoritmy'91, Vysoké Tatry, 1991.

[S91c] Skala,V.: Algoritmy zobecněného ořezávání, mezinárodní konference Algoritmy'91, Vysoké Tatry, 1991.

[S91d] Skala,V. a kol.: Algoritmy počítačové grafiky a CAD systémů, Závěrečná zpráva grantu 151/91, ZČU, 1991.

--- 1992 ---

[S92a] Skala,V.: Algoritmy počítačové grafiky I, skripta ZČU, Plzeň, pp.117, 1992.

Ertl,J., Ferko,A.: Normalizované grafické systémy, skripta MFF UKo, Bratislava, 1993.

Kolingerová,I.: Duální reprezentace a její využití v počítačové grafice, disertační práce, ZČU, Plzeň, 1994.

[S92b] Skala,V.: Algoritmy počítačové grafiky II, skripta ZČU, Plzeň, pp.167, 1992.

Ertl,J., Ferko,A.: Normalizované grafické systémy, skripta MFF UKo, Bratislava, 1993.

Kolingerová,I.: Duální reprezentace a její využití v počítačové grafice, disertační práce, ZČU, Plzeň, 1994.

[S92c] Skala,V.: Algoritmy počítačové grafiky III, skripta ZČU, Plzeň, pp.133, 1992.

Ertl,J., Ferko,A.: Normalizované grafické systémy, skripta MFF UKo, Bratislava, 1993.

Ferko,A.: Prehľad normalizovaných grafických systémov, sborník mezinárodního semináře WSCG94'94, Plzeň, 1994.

Kolingerová,I.: Duální reprezentace a její využití v počítačové grafice, disertační práce, ZČU, Plzeň, 1994.

Žára,J.: Počítačová grafika, GRADA, 1992.
Sochor,J.,Žára,J.: Algoritmy počítačové grafiky, skripta ČVUT-FEL, Praha, 1993.

[S92d] Skala,V. a kol.: Algoritmy počítačové grafiky a CAD systémů, Závěrečná zpráva grantu č.151/92, ZČU, 1992.
[S92e] Skala,V.: Algoritmy ořezávání v E^2 a E^3 a jejich srovnání, zpráva grantu č.151/92, KIV, ZČU, 1992.

Kolingerová,I.: Využití duálního prostoru pro metodu sledování paprsku, Zimní škola počítačové grafiky a CAD systémů, 1992.
Kolingerová,I.: Programová realizace urychlení metody sledování parsku, Konference Algoritmy 93, Vysoké Tatry, duben 1993.
Kolingerová,I.: Možnosti urychlení metody sledování paprsku, Konference Algoritmy 93, Vysoké Tatry, duben 1993.

[S92f] Kolingerová,I., Skala,V.(Ed.): Zimní škola počítačové grafiky a CAD systémů, sborník mezinárodního semináře WSCG'92, Rybníky, ZČU, 1992.
[S92g] Skala,V.: Barevné systémy a jejich aplikace v počítačové grafice, v Zimní škola počítačové grafiky a CAD systémů, mezinárodní seminář WSCG'92, ZČU, pp.116-121, 1992.
[S92h] Skala,V.: Recenze - Dudek,P.: MathCAD - příručka uživatele, Grada, Computer Echo, 1992.

--- 1993 ---
[S93a] Skala,V.: Světlo, barvy a barevné systémy v počítačové grafice, kniha, Academia, Praha, pp.120, 1993.
[S93b] Skala,V.: An Efficient Algorithm for Line Clipping, Computer Graphics'93 Conference Proceedings, Bratislava, 1993.

Kolingerová,I.: Duální reprezentace a její využití v počítačové grafice, disertační práce, ZČU, Plzeň, 1994

[S93c] Skala,V.: Algoritmus ořezávání přímky v E^2 konvexním n-úhelníkem, mezinárodní konference Algoritmy'93, pp.201-206, Vysoké Tatry, 1993.
[S93d] Skala,V.: Barevné systémy a jejich aplikace v počítačové grafice, mezinárodní konference Algoritmy'93, pp.214-219, Vysoké Tatry, 1993.
[S93e] Skala,V.: Algoritmus sledování paprsku a jeho složitost, mezinárodní konference Algoritmy'93, pp.207-213, Vysoké Tatry, 1993.
[S93f] Kolingerová,I., Krutišová,J., Skala,V.: Použití počítačové grafiky při výuce algoritmizace, mezinárodní konference Algoritmy'93, pp.146-152, Vysoké Tatry, 1993.
[S93g] Skala,V.: An Efficient Algorithm for Line Clipping by Convex Polygon, Preprint No.38, Univ.of West Bohemia, Plzeň 1993.
[S93h] Skala,V.: O(lg N) Line Clipping Algorithm, Preprint No.44, Univ.of West Bohemia, Plzeň, 1993.
[S93i] Skala,V.: An Efficient Algorithm for Line Clipping by Convex Polygon, Computers & Graphics, Pergamon Press, Vol.17, No.4, pp.417-421, 1993.

Kolingerová,I.: Duální reprezentace a její využití v počítačové grafice, disertační práce, ZČU, Plzeň, 1994

[S93j] Skala,V.: Memory Saving Technique for Space Subdivision Technique, Machine Graphics and Vision, Vol.2, No.2, pp.237-250, 1993.

--- 1995 ---

[S95a] Skala,V.(Ed.): Winter School of Computer Graphics - WSCG'95, International Conference Proceedings, Univ. of West Bohemia, Plzeň, 1995.

[S95b] Skala,V.: Algoritmy se strukturální složitostí menší než optimální, přijato na Algoritmy 95, sborník mezinárodní konference, Nízké Tatry, Slovensko, 1995.

[S95c] Bláha,P., Skala,V.: Urychlování algoritmu sledování paprsku metodou dělení prostoru a paralelizací výpočtu, přijato na Algoritmy 95, sborník mezinárodní konference, Nízké Tatry, Slovensko, 1995.

Citace na starší práce

[S84a] Skala,V.: Hidden Line Processor, CSTR 209-03-84, Computer Sci.Dept, Technical Univ., Plzeň, 1984.

Kramarczyk,W.G.: Efficient Raster Graphing of Bivariate Functions by Incremental Methods, in [EAR88a], 1988.

[S85a] Skala,V.: Hidden Line Solution by the Bresenham Algorithm, Proceedings of the First National Conference on Computer Applications, Lagos, 1985.

Singh,B.: Computer Graphics Literature for 1986: A Bibliography, Computer Graphics, Vol.21, No.3, 1987.

Ostatní

The PL/M Programming Language INSPEC 88 rec. 3046996.
The PL/M Programming Language INSPEC 84 rec. 2187603.

Více než 50 recenzních posudků pro zahraniční časopisy, kde jsem nebo byl členem redakčních rad (Computer Graphics Forum, Computer Aided Design, Computers & Graphics, Machine Graphics and Vision, Applied Computer Translation).

Přehled publikační činnosti

Seznam publikací do r.1985

Výzkumné zprávy:

Skala V.: Grafické zpracování informace pomocí počítače ODRA 1204, Výzkumná zpráva 209-04-75, Plzeň, VŠSE 1975

Skala V.; Hlavnička H.: Kreslení vývojových diagramů počítačem, Výzkumná zpráva 209-01-76, Plzeň, VŠSE 1976

Skala V., Dundálek J., Chládek P.: Metody hledání extrémů funkcí více proměnných, Výzkumná zpráva 209-02-76, Plzeň, VŠSE 1976

Skala V., Hlavnička H.: Kreslení vývojových diagramů na počítači ze zdrojového programu napsaného v jazyce ALGOL 60, Výzkumná zpráva 209-02-77, Plzeň, VŠSE 1977

Skala V., Dundálek J., Chládek P.: Grafické zpracování mnohostěnů na počítači (řešení viditelnosti), Výzkumná zpráva 209-11-77, Plzeň, VŠSE 1977

Skala V., Dudáček K., Kozák V.: Programové vybavení pro řešení úloh z rovinné geometrie, Výzkumná zpráva 209-12-77, Plzeň, VŠSE 1977

Skala V., Grigar P.: Realizace grafického výstupu pro rozvrh hodin, Výzkumná zpráva 209-01-78, Plzeň, VŠSE 1978

Skala V.: Filosofie implementace relace v relační bázi dat, Výzkumná zpráva 209-04-78, dílčí státní úkol III-2-2/2, Plzeň, VŠSE 1978

Skala V.: Zobrazování funkcí dvou proměnných pomocí počítače, Výzkumná zpráva 209-05-78, Plzeň, VŠSE 1978

Skala V.: Použití makrogenerátoru ML/1 pro editaci programů v jazyce Algol 60 a Algol 68, Výzkumná zpráva 209-06-78, Plzeň, VŠSE 1978

Skala V.: Příspěvek k filosofii implementace relace v relační bázi dat,
Výzkumná zpráva 209-12-78, dílčí státní úkol III-2-2/2, Plzeň, VŠSE 1978

Skala V.: Kreslení ekvipotenciál ploch v kartézském i polárním souřadném systému na počítači TESLA 200,
Výzkumná zpráva 209-11-78, Plzeň, VŠSE 1978

Skala V., Petřík J.: Testy pro ověření správnosti MD - logiky při přenosu makrogenerátoru ML/1,
Výzkumná zpráva 209-19-80, Plzeň, VŠSE 1980

Skala V.: Přehled architektur relačních databázových systémů,
Výzkumná zpráva 209-20-80, Plzeň, VŠSE 1980

Skala V., Hroník Š., Petřík J.: Realizace makrogenerátoru ML/1 na počítači ODRA 1204,
Výzkumná zpráva 209-11-81, Plzeň, VŠSE

Skala V., Petrů J., Hroník Š.: Možnosti přenosu makrogenerátoru ML/1 na počítač EC 1010,
Výzkumná zpráva 209-09-82, Plzeň, VŠSE 1982

Skala V., Chládek P., Svěrák F.: Překladač jazyka PASCAL pro EC 1010 - úvodní studie,
Výzkumná zpráva 209-10-82, Plzeň, VŠSE 1982

Skala V., Vejvoda J.: Programové vybavení pro počítačovou grafiku,
Výzkumná zpráva 209-11-82, Plzeň, VŠSE 1982

Skala V., Vítek F.: Realizace makrogenerátoru na počítači EC 1010,
Výzkumná zpráva 209-12-82, Plzeň, VŠSE 1982

Skala V.: Řešení šachových úloh na počítači (programová dokumentace),
Výzkumná zpráva 209-13-82, Plzeň, VŠSE 1982

Skala V.: Řešení šachových úloh na počítače (algoritmy řešení-verse 1.0 a verse 3.0),
Výzkumná zpráva 209-14-82, Plzeň, VŠSE 1982

Skala V.: Relační databázový systém RDBMS verse 1.1,
Výzkumná zpráva 209-12-84, VŠSE Plzeň 1984

Skala V., Formánek J.: Základní metody řešení problémů umělé inteligence a jejich aplikace při řešení šachových úloh,
Výzkumná zpráva 209-07-84, VŠSE Plzeň 1984

Skala V., Houška P.: Návrh a realizace relačního databázového systému,
Výzkumná zpráva 209-13-84, VŠSE Plzeň 1984

Skala V., Houška P.: Programová dokumentace relačního databázového systému,
Výzkumná zpráva 209-14-84, VŠSE Plzeň 1984

Skala V., Svoboda K.: Dynamická alokace paměti v jazyce FORTRAN,
Výzkumná zpráva 209-04-84, VŠSE Plzeň 1984

Skala V.: Programovací jazyk PL/M - 80 a PL/M - 86 (srovnávací studie),
Výzkumná zpráva 209-05-84, VŠSE Plzeň 1984

Skala V.: Knihovní podprogramy pro EC 1010,
Výzkumná zpráva 209-06-84, VŠSE Plzeň 1984

Skala V.: Clipping Algorithm for Rectangular Area, CSTR/28,
Brunel University, Middlesex, England 1984

Skala V.: Intersection of two Line Segments,
CSTR/27, Brunel University, Middlesex, England 1984

Skala V.: Hidden-Line Processor,
CSTR/29, Brunel University, Middlesex, England 1984

Skala V., Heick L.: Kreslení funkcí dvou proměnných s respektováním viditelnosti na TEKTRONIX 4015,
Výzkumná zpráva 209-34-85 , VŠSE Plzeň 1985

Skala V., Křepinský I.: Křížové programové vybavení pro 18080/18085, Z80, MC 6800,
Výzkumná zpráva 209-15-85, VŠSE Plzeň 1985

Skala V., Křepinský I.: Křížové programové vybavení pro 18080/18085, Z80, MC 6800 (programová dokumentace), Výzkumná zpráva 209-16-85, VŠSE Plzeň 1985

Skala V., Křepinský I.: Křížové programové vybavení pro 18080/18085, Z80, MC 6800 (testovací překlady), Výzkumná zpráva 209-17-85, VŠSE Plzeň 1985

Konference:

Skala V.: Editace programů v Algolu 60 a Algolu 68 pomocí ML/1, The 5-th Symposium on Algorithms´79, Vysoké Tatry, Dům techniky SVTS Bratislava 1979

Skala V.: Makra a MD-logika pro převod makrogenerátoru ML/1 na počítač ODRA 1204, Symposium Algoritmy´81, Dům techniky SVTS Bratislava 1981

Skala V.: Programovací jazyk PL/M pro mikropočítače, Symposium Algoritmy´83, Dům techniky SVTS Bratislava 1983

Skala V.: Podprogramy pro prácci s bitovými maticemi, Symposium Algoritmy´83, Dům techniky SVTS Bratislava 1983

Skala V.: Realizacija reljacij v reljacionnoj baze dannych, Seminar MEI - BT, Moskva 1978

Skala V.: Osnovnyje modeli baz dannych, Seminar MEI-BT, Moskva 1978

Skala V.: Implementace relační banky dat, DATABANKA 78, Ústí n. Labem, ČSVTS 1978

Skala V.: Programovací jazyk PL/M, Moderní programování, MFF UK, Praha 1983

Skala V.: Respektování viditelnosti - základní grafická instrukce, Algoritmy´85, Dům techniky SVST, Bratislava 1985

Skala V.: Hidden - Line Problem Solution by the Bresenham Algorithum, First National Conference on Computer Applications, Lagos, Nigeria, 1985

Skala V.: An Interesting Modification to the Bresenham Algorithm for Hidden - Line Problem Solution, Advanced Study Institute on Fundamental Algorithms for Computer Graphies, Ilkley, England 1985.

Semináře

Skala V.: Podprogram pro tisk velkých znaků na řádkové tiskárně, Sborník uživatelů počítače EC 1010, Plzeň 1982

Skala V.: Programové vybavení pro počítačovou grafiku na EC 1010 Sborník přednášek IV. semináře EC 1010, Sdružení uživatelů JSEP a SMEP, Plzeň 1982

Skala V.: Podprogramy pro práci s bitovými maticemi, Sborník uživatelů počítače EC 1010, Plzeň 1982

Skala V.: Podprogramy pro zpracování optické informace na počítači EC 1010, Sborník uživatelů EC 1010, Plzeň 1982

Skala V., Svoboda K.: Dynamická alokace paměti v jazyce FORTRAN IV, Sborník uživatelů počítače EC 1010, Plzeň 1982

Skala V.: Programovací jazyk PL/M-80, Sdružení uživatelů JSEP a SMEP, Kladky 1983

Skala V.: Počítačová grafika - možnosti v konstruktérské praxi DIAGNOSTIKA´86, DT ČSVTS Plzeň 1986

Skala V.: Křížové programové vybavení pro mikropočítač 18080/18085 na počítač SM 3/20, Sdružení uživatelů počítačů JSEP a SMEP, Plzeň 1986.

Časopisy:

Skala V.: Zobrazování funkcí dvou proměnných pomocí počítače, SNTL, Automatizace č. 1, 1978

Skala V., Hlavnička H.: Kreslení vývojových diagramů počítačem, SNTL, Automatizace č. 3, 1978

Skala V., Hlavnička H.: Kreslení vývojových diagramů počítačem II, SNTL Automatizace č. 7, 1978

Skala V.: Dva jednoduché algoritmy pro dekódování hodnot typu Hollerith v jazyce FORTRAN IV, Výběr informací z org. a výp. techniky č. 2, 1982

Skala V.: Programovací jazyk PL/M-80, Výběr informací z org. a výpočetní techniky, 1983

Skala V.: Amstrad Interactive Cross - Referencing, Personal Computer World, May 1986, England.

Učební texty:

Skala V.: Počítače a programování, pomocné učební texty pro obor silnoproudá elektrotechnika, VŠSE Plzeň 1981

Skala V.: FORTRAN - EC 1010, Uživatelská příručka pro obor silnoproudá elektrotechnika, VŠSE Plzeň 1981

Skala V.: Programovací jazyk PL/M, ČSVTS Plzeň VŠSE 1981

Ostatní:

Skala V.: Studie k organizaci dat, Práce k odborné zkoušce z kandidátského minima, Plzeň VŠSE 1976

Skala V.: Příspěvek k filosofii implementace relace v relační bázi dat, Kandidátská disertační práce, Plzeň 1978

Skala V.: Základní organizace souborů,
ČSVTS Telekomunikace Č. Budějovice 1980

Knižně:

Skala V.: An Interesting Modification to the Bresenham Algorithm for Hidden - Line Solution, in Fundamental Algorithms for Computer Graphics (ed. R. Earnshaw), Springer - Verlag 1985

Autorská osvědčení

Skala V.,Tesař R., Zdráhal K.: Měření účiníku

Skala V. Tesař R., Zdráhal K.: Regulátor účiníku

Zlepšovací návrhy

Úprava děrovačů SOEMTRON pro kod Hollerith
Zlepšovací návrhy VU 6292 Opava 6x

ČESKOSLOVENSKÁ SOCIALISTICKÁ REPUBLIKA

ÚŘAD PRO VYNÁLEZY A OBJEVY V PRAZE

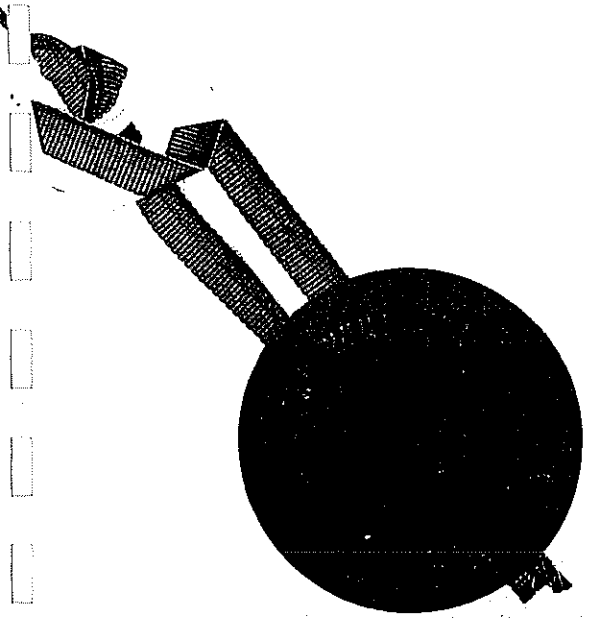
ČÍSLO 239283

ÚŘAD PRO VYNÁLEZY A OBJEVY V PRAZE UDĚLIL PODLE § 37, ODST. 1 ZÁKONA Č. 84/1972 SB. AUTORSKÉ OSVĚDČENÍ NA VYNÁLEZ, JEHOŽ POPIS JE PŘIPOJEN.
AUTOR VYNÁLEZU:

Tesař Rudolf ing., Plzeň

Skala Václav ing.CSc., Plzeň

Zdráhal Karel ing., Plzeň



AUTORSKÉ OSVĚDČENÍ BYLO ZAPSÁNO DO REJSTŘÍKU VYNÁLEZŮ POD SHORA UVEDENÝM ČÍSLEM.
VYNÁLEZ JE NÁRODNÍM MAJETKEM. STÁT MÁ PRÁVO VYUŽÍVAT A POVINNOST PEČOVAT O CO NEJŠIRŠÍ VYUŽÍVÁNÍ VYNÁLEZU (§ 6 ZÁKONA Č. 84/1972 SB.).

PŘIHLÁŠKA VYNÁLEZU PV 6132-83

PŘEDSEDA

V PRAZE 28. května 1987

ČESKOSLOVENSKÁ SOCIALISTICKÁ REPUBLIKA
(19)

ÚŘAD PRO VYNÁLEZY A OBJEVY

POPIS VYNÁLEZU K AUTORSKÉMU OSVĚDČENÍ

239283
(11) (B1)

(51) Int. Cl.[4]
G 01 R 25/00

/22/ Přihlášeno 23 08 83
/21/ PV 6132-83

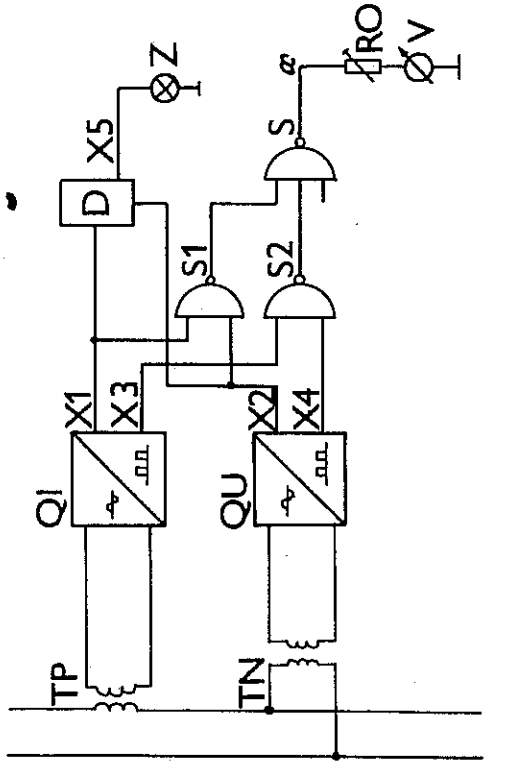
(40) Zveřejněno 15 05 85

(45) Vydáno 16 02 87

(75)
Autor vynálezu TESAŘ RUDOLF ing., SKALA VÁCLAV ing. CSc., ZDRÁHAL KAREL ing., PLZEŇ

(54) Zařízení pro měření účiníku

Zařízení pro měření účiníku, jehož vstupy jsou tvořeny napěťovým a proudovým transformátorem. Proudový transformátor je vstupem proudového pulsního obvodu a napěťový transformátor je vstupem napěťového pulsního obvodu. Pulsní výstupy obou pulsních obvodů vstupují do jednoho součinového obvodu a negované pulsní výstupy obou pulsních obvodů vstupují do druhého součinového obvodu. Výstupy obou součinových obvodů vstupují do součtového obvodu, na jehož výstup je připojen měřič napětí.



Obr. 1

ČESKOSLOVENSKÁ SOCIALISTICKÁ REPUBLIKA

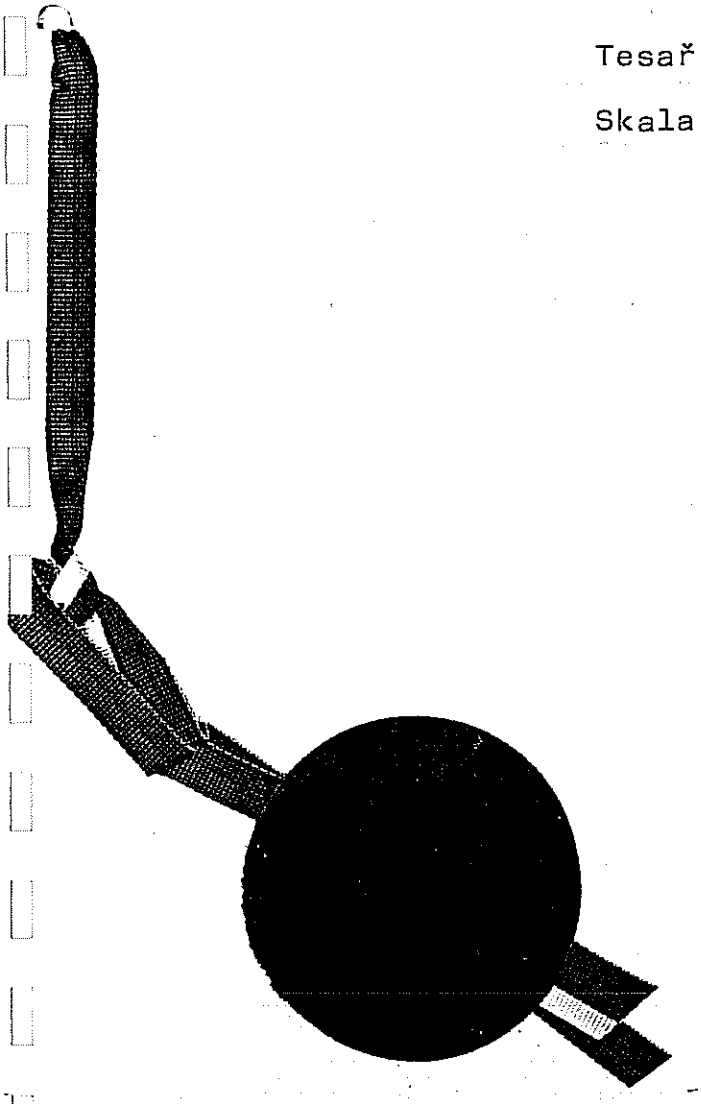
ÚŘAD PRO VYNÁLEZY A OBJEVY V PRAZE

ČÍSLO 251399

ÚŘAD PRO VYNÁLEZY A OBJEVY V PRAZE UDĚLIL PODLE § 37, ODST. 1 ZÁKONA Č. 84/1972 SB. AUTORSKÉ OSVĚDČENÍ NA VYNÁLEZ, JEHOŽ POPIS JE PŘIPOJEN. AUTOR VYNÁLEZU:

Tesař Rudolf ing.CSc., Plzeň

Skala Václav ing.CSc., Plzeň



AUTORSKÉ OSVĚDČENÍ BYLO ZAPSÁNO DO REJSTŘÍKU VYNÁLEZŮ POD SHORA UVEDENÝM ČÍSLEM.
VYNÁLEZ JE NÁRODNÍM MAJETKEM. STÁT MÁ PRÁVO VYUŽÍVAT A POVINNOST PEČOVAT O CO NEJŠIRŠÍ VYUŽÍVÁNÍ VYNÁLEZU (§ 6 ZÁKONA Č. 84/1972 SB.).

PŘIHLÁŠKA VYNÁLEZU PV 8626-85

PŘEDSEDA

V PRAZE 28. dubna 1989

ČESKOSLOVENSKÁ SOCIALISTICKÁ REPUBLIKA
(19)

ÚŘAD PRO VYNÁLEZY A OBJEVY

POPIS VYNÁLEZU K AUTORSKÉMU OSVĚDČENÍ

251399

(11) (B1)

(51) Int. Cl.[4]
G 01 R 25/00

(22) Přihlášeno 28 11 85
(21) PV 8626-85

(40) Zveřejněno 13 11 86

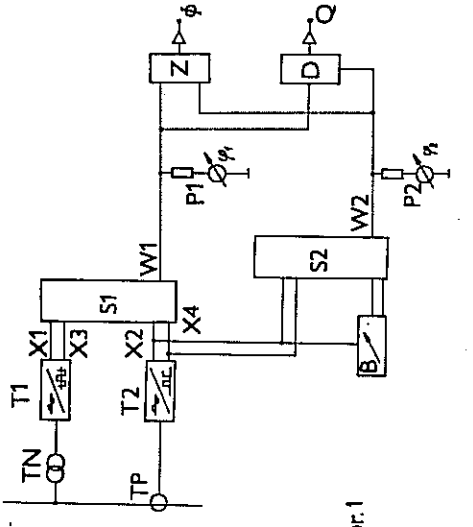
(45) Vydáno 15 03 88

(75)
Autor vynálezu

TESAŘ RUDOLF ing. CSc., SKALA VÁCLAV ing. CSc., PLZEŇ

(54) Zařízení pro indikaci rozdílu žádaného a skutečného fázového posuvu

Zařízení pro indikaci rozdílu žádaného a skutečného fázového posuvu, jehož vstupy jsou tvořeny napěťovým transformátorem, který je vstupem napěťového pulsního obvodu a proudovým transformátorem, který je vstupem proudového pulsního obvodu. Z napěťového pulsního obvodu a proudového pulsního obvodu jsou vedeny paralelně do prvního vyhodnocovacího obvodu, jehož výstup ústí do logického obvodu a klopného obvodu. Výstupy z proudového pulsního obvodu a přes zpožďovací obvod do druhého vyhodnocovacího obvodu. Výstup z druhého vyhodnocovacího obvodu ústí do klopného obvodu a logického obvodu.



Obr. 1

Comput. & Graphics Vol. 17, No. 4, pp. 417–421, 1993
Printed in Great Britain.

0097-8493/93 $6.00 + .00

Technical Section

AN EFFICIENT ALGORITHM FOR LINE CLIPPING BY CONVEX POLYGON

VÁCLAV SKALA
Department of Informatics and Computer Science, University of West Bohemia, Americká 42, Box 314, 306 14 Plzeň, Czech Republic

Abstract—A new line clipping algorithm against convex window based on a new approach for intersection detection is presented. Theoretical comparisons with Cyrus-Beck's algorithm are shown together with experimental results obtained by simulations. The main advantage of the presented algorithm is the substantial acceleration of the line clipping problem solution and that edges can be oriented clockwise or anti-clockwise.

1. INTRODUCTION

Many algorithms for clipping lines against rectangular, convex or non-convex windows have been published with many modifications derived from well-known Cohen-Sutherland, Liang-Barsky and Cyrus-Beck's algorithms[1–37]. Nevertheless, new developments or speed-up can be seen within every new paper.

The proposed algorithm is based on a simple rule: *Make tests first and then compute.*

A new criterion for testing whether the line intersects the given polygon or not is given, too.

The Cohen-Sutherland's algorithm (C-S) for clipping lines against a rectangular window is very well-known, not only because of its frequent usage, but for a very clever test whether the end-points are inside of the given rectangular area. The situation is a little bit more complicated when edges of the window are not collinear with axes.

1.1. *Algorithm* 1

```
procedure Clip 2D Cyrus Beck ( xA , xB );
begin { ! ! all vectors ni precomputed ! ! algorithm shortened }
  i := 1;  tmin := −∞;  tmax := ∞;  s := xB − xA;
  { for line segment  tmin := 0.0;  tmax := 1.0 }
  while i <= N do      { N is a number of edges }
  begin  { ni is a normal vector, e.g. ni = [ sy,−sx]^T }
         { and ni must point out of the convex window }
    ξ := s^T ni;  si := xi − xA;
    if ξ <> 0.0 then
    begin t := si^T ni / ξ;
          if ξ > 0.0 then tmax := min ( t , tmax )
                     else tmin := max ( t , tmin )
    end
    else Special case solution;
    i := i + 1
  end;
  if tmin > tmax then EXIT; { ! ! ⟨tmin,tmax⟩ = ∅ }
  { recompute end-points if changed }
  if tmax < 1.0 then xB := xA + s tmax;
  if tmin > 0.0 then xA := xA + s tmin;
  SHOW LINE( xA , xB );
end;
```

The known algorithms for clipping lines against general convex window do not make tests similar to C-S algorithm. The main reason seems to be the computational cost of such a test for convex window. If clipping algorithm is to be effective it is necessary to distinguish cases where lines pass through a given window from those where lines do not intersect the window. Cyrus-Beck's (C-B) algorithm solves this problem by direct computation of points of intersections, see Algorithm 1.

2. PROPOSED ALGORITHM

It is obvious that the line p intersects the edge $\mathbf{x}_0\mathbf{x}_1$ of the given polygon *if and only if* the vector $\mathbf{s}$ lies between two vectors $\mathbf{s}_0$ and $\mathbf{s}_1$, see Fig. 1.

Let us define ξ and η as the z-coordinate of the cross products as follows

$$\xi = [\mathbf{s} \times \mathbf{s}_i]_z \quad \eta = [\mathbf{s} \times \mathbf{s}_{i+1}]_z \quad i = 0, \ldots, N-1$$

where N is a number of edges.

It is possible to show that the line p given by the end-point $\mathbf{x}_A$ and by the vector $\mathbf{s}$ intersects the edge $\mathbf{x}_i\mathbf{x}_{i+1}$ *if and only if* the expressions

$$(\xi > 0) xor (\eta < 0), \quad \text{resp.} \quad \xi * \eta < 0$$

have value *true*, see Table 1 and Fig. 2. The above mentioned expressions are *independent* on polygon *orientation*. It means that edges can be clockwise or anti-clockwise oriented.

Now, it is possible to specify a new algorithm that is based on the presumption that a line can intersect a convex polygon only in two points, see Algorithm 2, if special cases are not considered, see Fig. 2. Special cases should be handled carefully, see [38].

In comparison with Algorithm 1 the proposed algorithm *only detects* whether the given line intersects the given polygon and then computes intersection points, while C-B algorithm does *compute all* possible intersection points with all edges.

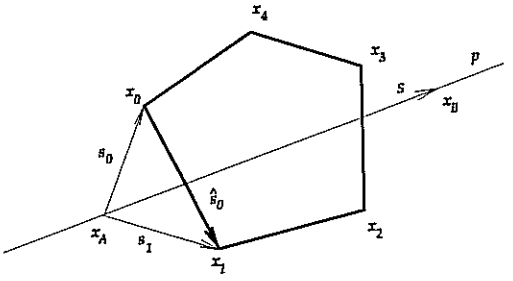


Fig. 1. Line segment clipping by convex polygon.

2.1. *Algorithm* 2

```
procedure Clip 2D ( x_A , x_B );
begin { !!!! all vectors ŝ_i are precomputed !!!! }
  k := 0; i := N − 1; j := 0;
  s := x_B − x_A;   ξ := [ s x s_i ]_z;
  while (j < N) and (k < 2) do
  begin η := [ s x s_j ]_z;
    if (ξ * η) < 0.0 then { intersection exists }
    begin
      index_k := i;   { save edge index having intersection }
      k := k + 1;   ξ := η;
    end
    else if ( ξ * η ) = 0.0 then Special cases { ξ=0 or η=0 }
                                 { cases i − m }
                          else Intersection does not exist;
                                 { cases e − h }
    i := j;   j := j + 1
  end;
  if k = 0 then { intersection does not exist } EXIT;
  { k = 2 intersections exist − edges saved in index_k }
  t_min := −∞; t_max := ∞; { for line segments t_min := 0; t_max := 1; }
  i := index_1;   t_1 := det[ x_i − x_A ¦ −ŝ_i ] / det[ s ¦ −ŝ_i ];
  { for effective implementation use identity x_i − x_A = s_i }
  i := index_2;   t_2 := det[ x_i − x_A ¦ −ŝ_i ] / det[ s ¦ −ŝ_i ];
  { recompute end-points if changed }
  if t_1 < t_2 then { swap t_1 <−> t_2 values ? }
  begin if t_2 < t_max then x_B := x_A + s t_2;
  if t_1 > t_min then x_A := x_A + s t_1
  end { t_2 < t_max; t_1 < t_min can be left out for lines }
  else begin if t_1 > t_min then x_B := x_A + s t_1;
  if t_2 < t_max then x_A := x_A + s t_2
  end;
  SHOW LINE( x_A , x_B );
  end
```

3. THEORETICAL ANALYSIS

Before making any comparisons it is necessary to point out that time needed for each operation (:=, <, ±, *, /) does differ from computer to computer, see Table 2.

Let us denote N number of edges of the given convex polygon. Then it is possible to express the complexity of the C-B algorithm as (the worst case)

$$(8, 3, 6, 4, 0) + (5, 3, 7, 4, 1) * N$$

and time of computation, according to Table 2 (PC 486 used), can be estimated as

$$T_{C\text{-}B} = 590 + 621 * N$$

while for proposed algorithm the complexity is given as

$$(15, 3, 11, 14, 2) + (3, 1, 3, 3, 0) * N$$

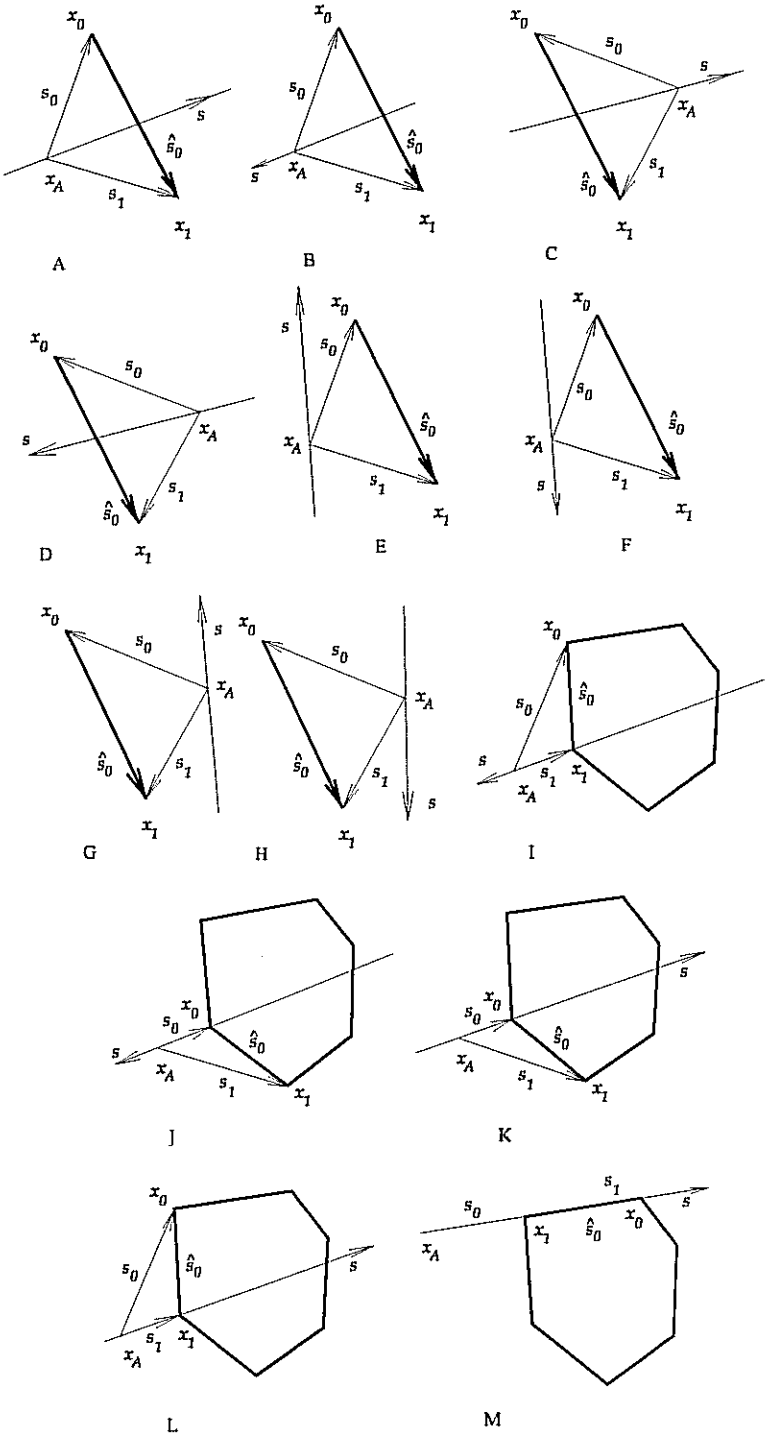


Fig. 2. Possible cases for line segment clipping.

and time of computation T can be estimated

$$T = 1329 + 257 * N$$

It means that the new algorithm should be better than C-B algorithm for $N \geq 2.1$. Let us consider a coefficient ν as a speed up coefficient, which is defined as

$$\nu = \frac{T_{\text{C-B}}}{T}$$

then some theoretical results of speed up are given in Table 3.

From the theoretical point of view the proposed algorithm should be for $N = 4$, approximately 30% faster than C-S algorithm. It can be seen that the efficiency increases dramatically with N, especially for $4 \leq N \leq 20$. For $N > 20$ the algorithm needs *only half* of the time needed by C-B algorithm. For $N > 50$ the factor ν is nearly constant. Table 3 shows expected results for

Table 1. Possible cases.

ξ	ν	Does the line intersect edge?
>0	>0	no—cases *g, h*
>0	=0	yes—special case *l*
>0	<0	yes—cases *a, c*
=0	>0	yes—special case *j*
=0	=0	no—special case *m*
=0	<0	yes—special case *k*
<0	>0	yes—cases *b, d*
<0	=0	yes—special case *i*
<0	<0	no—cases *e, f*

Table 2. Times for basic arithmetic operations.

	PC 386DX/387 25 MHz				
	:=	<	±	*	/
Int	14	42	10	40	58
Float	204	260	80	82	154

	PC 486 33 MHz 64KB cache—selected for evaluation				
	:=	<	±	*	/
Int	5	9	3	26	44
Float	33	50	16	20	114

Time is in 1/10 sec. for 5 000 000 operations.

Table 3. Theoretical results.

	Expected results for PC 386DX/387					
	Intersections exist			Intersections do not exist		
N	4	10	$\Rightarrow \infty$	4	10	$\Rightarrow \infty$
ν	1.26	1.60	2.09	1.75	1.87	1.97

	Expected results for PC 486					
	Intersections exist			Intersections do not exist		
N	4	10	$\Rightarrow \infty$	4	10	$\Rightarrow \infty$
ν	1.30	1.74	2.42	2.01	2.16	2.29

Table 4. Experimental results.

Experimental results for PC 386DX/387								
N	5	8	10	15	20	30	70	190
ν	1.41	1.54	1.73	1.81	1.90	2.08	2.27	2.41

Experimental results for PC 486 33 MHz								
N	5	8	10	15	20	30	70	190
ν	1.37	1.59	1.71	1.88	2.25	3.32	2.58	2.85

PC 386/387 DX, too. It can be seen that the efficiency of the proposed method should be higher for faster machines. It is necessary to point out that many simplifications have been made for theoretical analysis because time of operations for integers and time needed for instructions was not considered.

4. EXPERIMENTAL RESULTS

The C-B and proposed algorithms have been tested on data sets of end-points that have been randomly and uniformly generated over a space inside of a circle in order to eliminate an influence of rotation. Convex polygons were generated as regular N-sided convex polygons inscribed into a smaller circle. The results in Table 4 have been obtained if 80% of the lines intersect the window.

The coefficients ν are in a good correlation with theoretical results because the pessimistic estimations have been taken for the theoretical analysis. It is necessary to point out that different tests might be used instead of shown cross product, *e.g.*, test for detection on which side of the given line p the point x_i lies.

5. CONCLUSION

The new efficient algorithm for clipping lines against convex window has been developed. Edges of the given polygon can be arbitrarily oriented. The theoretical analysis showed the advances over well-known Cyrus-Beck's algorithm. Experimental results support the theoretical analysis in spite of the fact that many factors have been omitted, like computational cost with integers, *etc.*

All tests were implemented in C++ on a PC 386/387 25 MHz and PC 486 33 MHz. It can be expected that for workstations the efficiency ν will be higher than for PC 486.

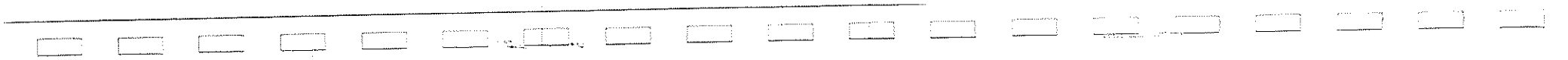
There is a hope that the proposed algorithm can be modified for clipping lines against non-convex windows, similarly to[31].

Acknowledgements—The author would like to express his thanks to students of Computer Graphics and CAD Systems who stimulated this work, especially Mr. P. Blaha for careful tests implementation of the proposed algorithm.

REFERENCES

1. K. Akeley and C. P. Korobkin, Efficient Graphics Processor for Clipping Polygons, US Patent No. 5 051 737 (1991).
2. R. D. Andreev, Algorithm for clipping arbitrary polygons. *Comp. Graph. Forum* **7**, 183–192 (1988).
3. A. Arokiasamy, Homogeneous coordinates and the principle of duality in two dimensional clipping. *Comp. & Graph.* **13**, 99–100 (1989).
4. J. F. Blinn and M. E. Newell, Clipping using homogeneous coordinates. *Comp. Graph. (SIGGRAPH'78)* **12**, 245–251 (1978).
5. J. F. Blinn, A trip down to graphics pipeline—Line clipping. *IEEE Comp. Graph. Appl.* **11**, 98–105 (1991).

6. E. A. Brewer and B. A. Barsky, Clipping after projection: An improved perspective pipeline. Manuscript submitted for publication.
7. A. Burkert and S. Noll, Fast algorithm for polygon clipping with 3D windows. *Eurographics'88 Proceedings*, 405–419 (1988).
8. F. Cheng and Y. Yen, A parallel line clipping algorithm and its implementation. *CGI'89 Conference Proceedings* (1989).
9. M. Cyrus and J. Beck, Generalized two and three dimensional clipping. *Comp. & Graph.* **3**, 23–28 (1979).
10. J. D. Day, A comparisons of line clipping algorithms. Queensland University of Technology, School of Computing Science, Report 2-91 (1991).
11. J. D. Day, A new two dimensional line clipping algorithms for small windows. Queensland University of Technology, School of Computing Science, Report 3-91 (1991).
12. J. D. Day, A new two dimensional line clipping algorithms for small windows. *Comp. Graph. Forum* **11**, 241–245 (1992).
13. M. Dorr, A new approach to parametric line clipping. *Comp. & Graph.* **14**, 449–464 (1990).
14. V. J. Duvanenko, W. E. Robins, and R. S. Gyurcsik, Improving line segment clipping. *Dr. Dobb's J. Software Tools* **15**, 36, 38, 40, 42, 44–45, 98, 100 (1990).
15. D. Y. Fong and J. Chu, A string pattern recognition approach to polygon clipping. *Pattern Recognition* **23**, 879–892 (1992).
16. I. Herman and J. Reviczky, Some remarks on the modelling clip problem. *Comp. Graph. Forum* **7**, 265–272 (1988).
17. S. Kaijian, J. A. Edwards, and D. C. Cooper, An efficient line clipping algorithm. *Comp. & Graph.* **14**, 297–301 (1990).
18. A. C. Kilgour, Unifying vector and polygon algorithm for scan conversion and clipping. TR CSC/87/R7, University of Glasgow (May 1987).
19. G. Krammer, A line clipping algorithm and its analysis. *Comp. Graph. Forum* (*EG'92 Conference Proceedings*) **11**, C253–266 (1992).
20. Y. D. Liang and B. A. Barsky, A new concept and method for line clipping. *ACM Trans. Graph.* **3**, 1–22 (1984).
21. Y. D. Liang and B. A. Barsky, An analysis and algorithms for polygon clipping. *CACM* **26**, 868–876 (1984).
22. Y. Liang and B. A. Barsky, The optimal tree algorithms for line clipping. Technical paper distributed at Eurographics'92 Conference, Cambridge (1992).
23. P. G. Maillot, A new, fast method for 2D polygon clipping: Analysis and software implementation. *ACM Trans. Graph.* **11**, 276–290 (1992).
24. T. M. Nicholl, D. T. Lee, and R. A. Nicholl, An efficient new algorithm for 2D line clipping: Its development and analysis. *ACM Comp. Graph.* **21**, 253–262 (1987).
25. H. P. Nielsen, An intersection test using dual figures. *CFG* (in press).
26. H. P. Nielsen, Line clipping using semi-homogeneous coordinates. *CFG* (in press).
27. R. M. O'Bara and S. Abi-Ezzi, An analysis of modeling clip. In *EG'89 Conference Proceedings*, 367–380 (1989).
28. A. Rappaport, An efficient algorithm for line and polygon clipping. *Visual Comp.* **7**, 19–28 (1991).
29. N. C. Sharma and S. Manohar, Line clipping revisited: Two efficient algorithms based on simple geometric observations. *Comp. & Graph.* **16**, 51–54 (1992).
30. V. Skala, Algorithms for 2D line clipping. In *CGI'89 Conference Proceedings*, 121–128 (1989).
31. V. Skala, Algorithms for 2D line clipping. In *EG'89 Conference Proceedings*, 355–367 (1989).
32. V. Skala, Algorithm for line clipping in E2 for convex window (in Czech). *Algorithms '93 Conference Proceedings* (in press).
33. M. Slater and A. B. Barsky, 2D line and polygon clipping based on space subdivision. Manuscript submitted for publication.
34. M. S. Sobkow, P. Pospil, and Y.-H. Yang, A fast two-dimensional line clipping algorithm via line encoding. *Comp. & Graph.* **11**, 459–467 (1987).
35. I. E. Sutherland and G. W. Hodgman, Reentrant polygon clipping, *CACM* **17**, 32–42 (1974).
36. D.-N. Ying, A new algorithm for polygon clipping and Boolean operations, University of Zhejiang, Hangzhou, China (1991).
37. L. Yong-Kui, A new algorithm for line clipping by convex polygon. Manuscript submitted for publication.
38. H. Edelsbrunner and E. P. Muche, Simulation of simplicity: A technique to cope with degeneration cases in geometric algorithms. *ACM Trans. Graph.* **9**, 66–104 (1990).



Pergamon

Comput. & Graphics, Vol. 18, No. 4, pp. 517–524, 1994

Printed in Great Britain
0097-8493/94 $6.00 + .00

0097-8493(94)E0046-Z

Technical Section

$O(lg\ N)$ LINE CLIPPING ALGORITHM IN E^2

VÁCLAV SKALA
Department of Informatics and Computer Science, University of West Bohemia, Americká 42, Box 314, 306 14 Plzeň, Czech Republic, e-mail: skala@kiv.zcu.cz

Abstract—A new $O(lg\ N)$ line clipping algorithm in E^2 against a *convex window* is presented. The main advantage of the presented algorithm is the principal acceleration of the line clipping problem solution. A comparison of the proposed algorithm with others shows a significant improvement in run-time. Experimental results for selected known algorithms are also shown.

1. INTRODUCTION

Many algorithms for clipping lines against convex or nonconvex windows in E^2 with many modifications derived from well-known Cohen-Sutherland's, Liang-Barsky's[1, 2] and Cyrus-Beck's[3] algorithms have been published. All of them have the same complexity $O(N)$, with an exception of Rappaport's algorithm[4], which has $O(lg\ N)$ complexity. Their speed is determined by more or less clever implementation of tests and intersection computation. The convexity feature of the clipping polygon and the possibility of binary search usage over polygon vertices, because of known vertices order, have been used for principal speed up of the *ECB* line clipping algorithm[5] that resulted into new line clipping algorithm with complexity $O(lg\ N)$. It has been expected that an algorithm for line clipping against convex polygon with complexity $O(lg\ N)$ exists, see [6]. An algorithm for a line segment clipping with $O(lg\ N)$ complexity was published in [4].

The known algorithms for clipping lines against a general convex window do not make tests similar to Cohen-Sutherland's clipping algorithm. The main reason seems to be the computational cost of such tests for convex windows. If a clipping algorithm is to be effective, it is necessary to distinguish cases where lines pass through a given window from those where lines do not intersect the window. Cyrus-Beck's (*CB*) algorithm solves this problem by direct computation of points of intersections, the *ECB* algorithm uses the separation theorem for Cyrus-Beck's algorithm to achieve a speed up of approximately 1.2–2.5 times. Cyrus-Beck's (*CB*), Efficient Cyrus-Beck's (*ECB*) and Rappaport's algorithms have been compared with the new proposed $O(lg\ N)$ algorithm.

The *ECB* algorithm does not use the known order of vertices of the given clipping polygon for a principal speed up of the algorithm, though it has the complexity $O(N)$.

The Rappaport's algorithm[4] is the only one algorithm with $O(lg\ N)$ complexity that could be used for *line segments* clipping against convex polygon. The algorithm (see Algorithm 1) is based on known fact that an answer whether a point is inside of the convex polygon can be given in $O(lg\ N)$ steps, where N is a number of vertices of the given polygon[7].

1.1. *Algorithm* 1

```
procedure RAPPAPORT (xA, xB);
{xA, xB are end-points of the clipped line segment}
begin
  if CLASSIFY (xA) = IN then
  begin
    (s, s1) := SECTOR (xA, xB);
    if xB is to the left of s-s1 edge of the polygon
                {s1 is the next vertex to vertex s}
    then OUTPUT (xB) {the line segment is totally inside}
    else
      begin
        compute the intersection point of the line segment with
        the edge s-s1 (x);
        OUTPUT (x);
      end
  end
  else
```

```
    begin
      (left_sup, right_sup) := SUPPORT_VERTICES (xA);
  if xB is left of left_sup or right of right_sup
  then DO_NOTHING
  else
  begin {find an intersected edge from the front chain}
      (s, s1) := FRONT_SECTOR (left_sup, right_sup);
      if xB is to the right of s-s1
      then DO_NOTHING
      else
      begin
        compute the intersection point of the line segment
          with the edge s-s1 (x);
        OUTPUT (x);
        (s, s1) := BACK_SECTOR (right_sup, left_sup);
        if xB is to the left of s-s1
        then OUTPUT (xB)
        else
        begin {find an intersected edge from the back chain}
          compute the intersection point of the line segment
            with the edge s-s1 (x);
          OUTPUT (x);
        end
      end
    end
  end
end {RAPPAPORT};
```

There are used the following functions in Algorithm 1:

- CLASSIFY (x) gives an answer if the point x is inside of the given convex polygon in $O(lg\ N)$ steps and has complexity {(:=, <, ±, *, /) counting FPP operations only}

$$(0, 2, 4, 4, 0) + lg\ N * (0, 1, 2, 2, 0),$$

- SECTOR (x_A, x_B) finds an edge with vertices ($s, s1$) that is intersected by the given line segment $x_A x_B$ in $O(lg\ N)$ steps and has complexity

$$lg\ N * (7, 2, 9, 5, 0),$$

- SUPPORT_VERTICES (x_A) finds the (left_sup, right_sup) indexes of endpoints of the back and front chains that are formed by edges of the given polygon in $O(lg\ N)$ steps and has complexity

$$(0, 2, 10, 4, 0) + lg\ N * (0, 2, 10, 4, 0),$$

- FRONT_SECTOR (left_sup, right_sup) finds from *front chain* of edges with vertices ($s, s1$) that is intersected by the given line segment $x_A x_B$ in $O(lg\ N)$ steps and has complexity

$$lg\ N * (0, 1, 2, 2, 0),$$

- BACK_SECTOR (left_sup, right_sup) finds from *back chain* of edges with vertices ($s, s1$) that is intersected by the given line segment $x_A x_B$ in $O(lg\ N)$ steps,

$$lg\ N * (0, 1, 2, 2, 0),$$

It can be seen that all steps are of $O(lg\ N)$ complexity and therefore the whole algorithm is of $O(lg\ N)$ complexity, too. Unfortunately, some steps are quite complex and the overall complexity for the worst case can be estimated as

$$(4, 2, 12, 22, 2) + lg\ N * (0, 4, 14, 8, 0)$$

Detailed description of the Rappaport's algorithm can be found in [4].

2. PROPOSED ALGORITHM

Let us suppose that we have a given *convex* clipping polygon anti-clockwise oriented and a line p is determined by two end-points

$$\boldsymbol{x}_A = [x_A, y_A]^T, \quad \boldsymbol{x}_B = [x_B, y_B]^T$$

The convex window is represented by $n + 1$ points

$$\boldsymbol{x}_i = [x_i, y_i]^T, \quad i = 0, \ldots, n$$

where points $\boldsymbol{x}_0$ and $\boldsymbol{x}_n$ are identical (column notation is used), x_i and y_i are coordinates of the vertex $\boldsymbol{x}_i$.

The notation $\overline{\boldsymbol{x}_i \boldsymbol{x}_k}$ is used for a polyline from $\boldsymbol{x}_i$ to $\boldsymbol{x}_k$, *i.e.*, it is a *chain* of line segments from $\boldsymbol{x}_i$ to $\boldsymbol{x}_k$.

Let us define the separation function $F(\boldsymbol{x})$ in the form

$$F(\boldsymbol{x}) = Ax + By + C$$

where $F(x) = 0$ is an equation for the given line p and assume that the line has the orientation shown in Fig. 1, x is defined as $x = [x, y]^T$.
It can be seen (Fig. 2) that the oriented distance d of the point x from the line p can be determined as

$$d = \frac{Ax + By + C}{\sqrt{A^2 + B^2}}$$

It means that the value of the function $F(x)$ is actually proportional to the distance d for the given line p.

First of all, let us assume that (see Fig. 1)

$$i = 0, \quad j = n, \quad k = (i + j)/2 \quad i.e., \quad k = \lfloor n/2 \rfloor$$

and

$$x_0 \equiv x_n \quad x_i = x_0 \quad x_j = x_n \quad x_k = x_2$$

Let us concentrate on a special case shown in Fig. 1. If the points x_i and x_k are on the opposite sides of the line p, *i.e.*,

$$F(x_i) * F(x_k) < 0$$

then there must be just one intersection point on the chains $\overline{x_i x_k}$ and $\overline{x_k x_j}$ for each chain, because the given polygon is convex. Because $F(x_i) * F(x_k) < 0$ for the chain $\overline{x_i x_k}$ there *must exist* an index l so that

$$F(x_l) * F(x_{l+1}) < 0 \quad i \le l < k$$

i.e., an edge $x_l x_{l+1}$ must be intersected.

Similarly for the chain $\overline{x_k x_j}$. It is obvious that in this case the intersection point can be found in $O(lg\ M)$ steps using *binary search* over vertices, where M is a number of line segments in the given chain.

Unfortunately, other possible situations are more complex to solve, see Fig. 3. It is possible to distinguish four fundamental cases supposing the previously shown orientation of the separation function $F(x)$. In case (a) the chain $\overline{x_k x_j}$ can be removed, while in case (b) the chain $\overline{x_i x_k}$ can be removed. In the first, resp. second, case index j, resp. index i, must be changed to k. In both cases a new value of k must be computed as

$$k = (i + j)div\ 2$$

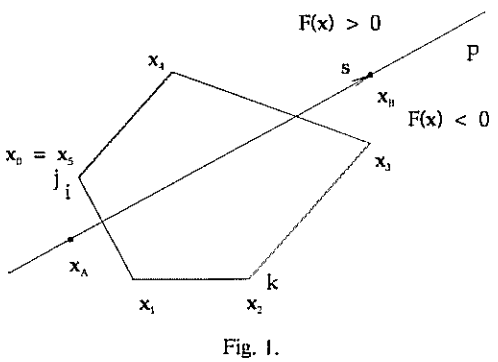


Fig. 1.

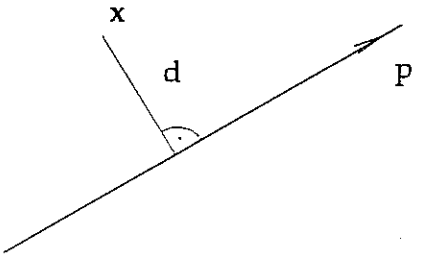


Fig. 2.

Both mentioned cases can be distinguished by a criterion

$$F(x_{i+1}) < F(x_i)$$

because if $F(x_{i+1}) < F(x_i)$ then the chain $\overline{x_i x_k}$ can intersect the line p, see Fig. 3. This condition actually expresses that we are getting closer to the line p, *i.e.*, the oriented distance d is smaller.

In both cases we assumed that the line p has the shown orientation, *i.e.*, $F(x_i) > 0$ and

$$F(x_i) \le F(x_k)$$

Possible situations as a variation of cases (a) and (b) in Fig. 3, when this condition is not true, are shown as cases (c) and (d).

A little bit more complex situation is shown by cases (c) and (d) where $F(x_i) > F(x_k)$. In case (c) the chain $\overline{x_k x_j}$ can be removed, while in case (d) the chain $\overline{x_i x_k}$ can be removed. In the first, resp. second, case index j, resp. index i, must be changed to k. In both cases a new value of k must be again determined as

$$k = (i + j)div\ 2$$

Both last mentioned cases can be distinguished by using criterion

$$F(x_{k+1}) > F(x_k)$$

Actually we must distinguish whether we are getting closer to the given line p or not. If the line p has an opposite orientation then similar situations must be solved, see Algorithm 2.

This procedure is repeated until

$$F(x_i) * F(x_k) < 0.$$

If this condition becomes true we will obtain two chains $\overline{x_i x_k}$ and $\overline{x_k x_j}$ that intersect the line p and binary search over vertices can be used again as we get a similar situation shown in Fig. 1.

Now it can be seen that all parts of the proposed algorithm are of complexity $O(lg\ M)$, where M is a number of edges in the given chain because we used for all steps the binary search over vertices of the clipping convex polygon. The whole proposed $O(lg\ N)$ algorithm is described by Algorithm 2. It is necessary to point out that for effective implementation values $F(x_i)$ should be stored in separate variables as they are used several times.

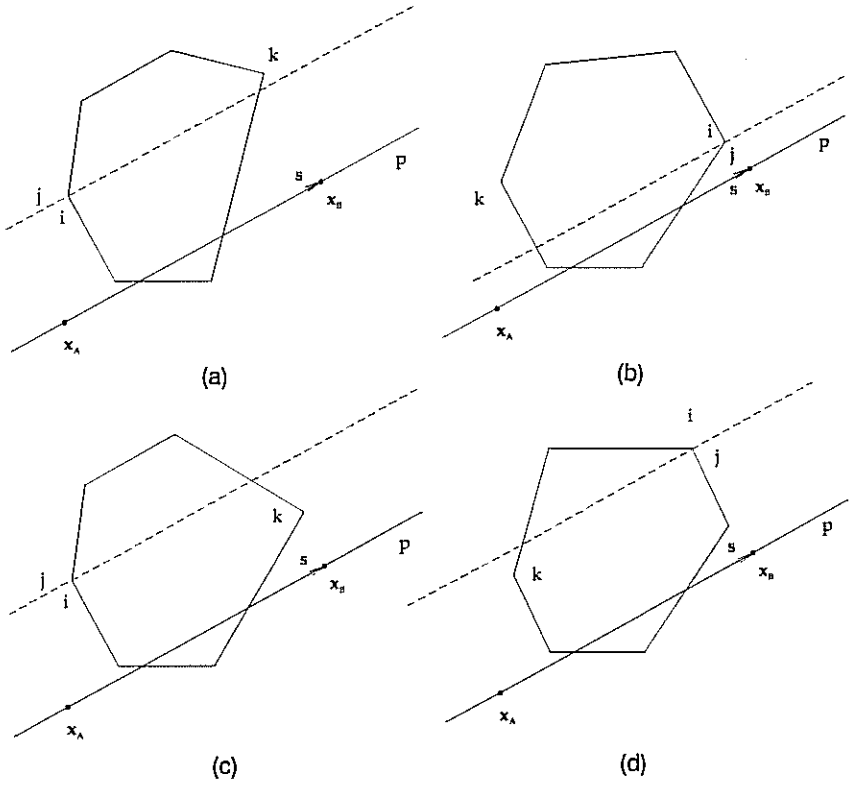


Fig. 3. Dashed lines mean points x, where $F(x) = F(x_i)$.

2.1. *Algorithm* 2

```
procedure CLIP_2D_lg (x_A, x_B);
{Note: initialization of the clipping window x_n := x_0}

function macro F (x): real;
{should be implemented as an in-line function}
begin
  F := A * x + B * y + C;
end {F};

function SOLVE (i, j): real;
{finds two nearest vertices on the opposite sides}
{of the given line p}
begin while (j - i) ≥ 2 do {j >= i always}
        begin k := (i + j) div 2; {shift to the right}
          if (F(x_i) * F(x_k)) < 0 then j := k
                                   else i := k;
        end {while};
        SOLVE := INTERSECTION (p, x_i, x_j);
        {gives the value t of an intersection point}
        {of the line p with the given line segment x_i x_j}
end {SOLVE};

begin {determine the A, B, C values for the function F(x)}
  A := y_1 - y_2;  B := x_2 - x_1;  C := x_1*y_2 - x_2*y_1;
  i := 0; j := n;  {for lines t_min := -∞;  t_max := ∞;}
  {for line segments t_min := 0; t_max := 1;}
  while (j - i) ≥ 2 do
  begin
    k := (i + j) div 2; {shift to the right}
    if (F(x_i) * F(x_k)) < 0 then

    begin {see fig. 1}
      t_1 := SOLVE (i, k); {find an intersection on x_i x_k chain}
```

```
      t2 := SOLVE (k, j); {find an intersection on x_k x_j chain}
      {for the line segment clipping include the next 5 lines}
      {if t1 > t2 then begin t := t1; t1 := t2; t2 := t1 end;}
      {compute ⟨t1, t2⟩ as ⟨t1, t2⟩ ∩ ⟨0, 1⟩}
      {t1 := max (tmin, t1); t2 := min (tmax, t2);}
      {if ⟨t1, t2⟩ ≠ ∅ then draw line segment}
      {if t1 ≤ t2 then}
        SHOW_LINE (x(t1), x(t2));
      EXIT {exit procedure CLIP_2D_lg};
    end {if};
    {for the polygon orientation shown in fig. 3}
    if F(xi) > 0 then
    begin {for the orientation of line p shown in fig. 3}
      if F(xi) < F(xk) then {cases a and b}
      begin  {DELETE CHAIN (i, j) removes the chain x_i x_j}
        if F(xi+1) < F(xi) then
          begin j := k; {DELETE CHAIN (k, j); case a} end
        else
          begin i := k; {DELETE CHAIN (i, k); case b} end
      end
      else {cases c and d}
      begin
        if F(xk+1) > F(xk) then
          begin j := k; {DELETE CHAIN (k, j); case c} end
        else
          begin i := k; {DELETE CHAIN (i, k); case d} end
      end
    end
    else
    begin {for an opposite orientation of the line p}
      if F(xi) > F(xk) then
      begin
        if F(xi+1) > F(xi) then
          begin j := k; {DELETE CHAIN (k, j);} end
        else
          begin i := k; {DELETE CHAIN (i, k);} end
      end
      else
      begin
        if F(xk+1) < F(xk) then
          begin j := k; {DELETE CHAIN (k, j);} end
        else
          begin i := k; {DELETE CHAIN (i, k);} end
      end
    end
  end {while}
end {CLIP_2D_lg}
```

3. THEORETICAL ANALYSIS AND EXPERIMENTAL RESULTS

Before making any experiments it is convenient to point out that time needed for operations (:=, <, ±, *, /) differ significantly from computer to computer.

Let us introduce coefficients of the effectivity ν as

$$\nu_1 = \frac{T_{CB}}{T}, \quad \nu_2 = \frac{T_{CB}}{T_0}, \quad \nu_3 = \frac{T_R}{T}$$

where T_{CB}, T_0, T_R, T are execution times needed by Cyrus-Beck's, *ECB*, Rappaport's and proposed $O(lg\ N)$ algorithms.

Descriptions of *CB* and *ECB* algorithms can be found in [5] together with their theoretical and experimental comparisons.

Generally it is possible to express the complexity of the *CB* algorithm

$$(8, 3, 6, 4, 0) + (5, 3, 7, 4, 1) * N$$

Table 1. Times for 5.10^6 floating point operations in $\frac{1}{10}$ sec. for PC 486/33MHz.

Float	:=	<	±	*	/
Time	33	50	16	20	114

Table 2. Theoretical estimations (worst case).

N	4	5	6	7	8	9	10	20	30	50	100
ν_1	1.28	1.54	1.80	2.06	2.01	2.23	2.45	4.13	6.11	8.98	16.08
ν_2	0.98	1.09	1.20	1.31	1.22	1.31	1.41	2.06	2.87	4.02	6.93
ν_3	1.19	1.19	1.10	1.19	1.24	1.24	1.24	1.27	1.27	1.30	1.33

and time of computation as T_{CB} (for PC 486, see Table 1) can be estimated

$$T_{CB} = 590 + 621 * N$$

The complexity of the *ECB* algorithm (in the worst case) as

$$(15, 3, 11, 14, 2) + (3, 1, 1, 3, 0) * N$$

and time of computation T_0 can be estimated as

$$T_0 = 1329 + 257 * N$$

Description of CB and ECB algorithms and their theoretical and experimental comparisons can be found in [5]. Their complexities are O(N).

Complexity of the *Rappaport's* algorithm can be expressed as

$$(4, 2, 12, 22, 2) + (0, 4, 14, 8, 0) * \lfloor lg(N + 1) \rfloor$$

and time of computation T_R can be estimated as

$$T_R = 1092 + 584 * \lfloor lg(N + 1) \rfloor$$

while for the *suggested algorithm* $O(lg\ N)$ the complexity is given as

$$(14, 4, 11, 15, 2) + (2, 4, 6, 6, 0) * \lfloor lg(N + 1) \rfloor$$

and time of computation T can be estimated as

$$T = 1267 + 376 * \lfloor lg(N + 1) \rfloor$$

The Rappaport's and proposed algorithms are of $O(lg\ N)$ complexity. Theoretical speed up is given in Table 2 (the worst cases and operations in floating point were considered only)

The proposed algorithm has been tested against Cyrus-Beck's, *ECB* and Rappaport's algorithms on data sets of line segments (10^3) with endpoints that have been randomly and uniformly generated inside a circle in order to eliminate an influence of rotation. Convex polygons were generated as N-sided convex polygons inscribed into a smaller circle.

There are practically no significant differences as far as the percentage is intersecting lines is concerned, see Table 3.

It can be seen, see Table 3, that the proposed algorithm is significantly faster then CB algorithm. A comparison of *ECB* and proposed algorithms shows that for $N < 7$ the *ECB* algorithm is faster than the proposed one. "Waves" for ν_2 are caused by the influence of binary division of an index interval and relation between data and convex polygon position. The waves can be seen in Table 2 with theoretical estimations, too. The significant difference for $N = 100$ is caused by considering the worst cases only in theoretical estimations.

The proposed $O(lg\ N)$ algorithm is approximately two times faster than Rappaport's algorithm and it is much more simple to implement.

It is necessary to point out that careful implementation of conditions like to $F(x_i) > F(x_k)$ might further improve the efficiency of the proposed algorithm, be-

Table 3.

ν_1	0%	20%	40%	60%	80%	100%
3	1.00	0.93	1.01	1.09	0.82	0.80
4	1.48	0.91	1.23	1.19	1.23	1.02
5	1.26	1.05	1.11	1.35	1.06	1.08
6	1.38	1.24	1.36	1.32	1.14	1.11
7	1.47	1.30	1.34	1.30	1.46	1.40
8	1.68	1.33	1.48	1.58	1.45	1.61
9	1.86	1.99	1.19	1.42	1.64	1.23
10	1.52	2.07	2.30	1.57	2.14	1.61
30	3.70	4.47	3.53	3.44	3.74	4.40
50	6.28	6.11	6.06	6.10	6.03	5.80
100	10.42	9.28	10.20	10.18	10.85	11.11
ν_2	0%	20%	40%	60%	80%	100%
3	1.47	1.19	1.33	1.17	0.91	0.98
4	1.81	1.27	1.27	1.14	1.22	1.14
5	1.81	1.40	1.19	1.33	1.40	1.35
6	1.89	1.81	1.39	1.52	1.24	1.26
7	1.12	1.66	1.37	1.38	1.62	1.49
8	1.77	1.61	1.72	1.73	1.79	1.75
9	1.89	1.70	1.79	1.55	1.32	1.54
10	1.61	3.28	1.90	1.47	1.63	1.46
30	2.00	1.96	1.95	1.70	1.86	2.14
50	2.29	1.96	1.91	2.06	2.12	2.14
100	2.37	1.98	2.03	2.13	2.21	2.35
ν_3	0%	20%	40%	60%	80%	100%
3	2.96	3.76	2.82	3.13	2.70	2.40
4	3.44	1.98	2.65	2.81	3.10	2.53
5	2.90	2.24	2.56	2.67	2.29	2.12
6	2.62	2.26	2.89	2.59	2.27	1.96
7	2.68	2.41	2.36	2.50	2.35	2.25
8	2.83	2.01	2.25	2.44	1.97	2.43
9	2.78	2.85	1.71	2.07	2.46	1.68
10	1.91	2.96	3.20	2.30	2.49	2.24
30	2.22	2.64	2.42	2.34	2.30	2.19
50	2.44	2.52	2.43	2.31	2.09	1.99
100	2.13	2.15	2.26	2.07	2.39	2.12

cause of comparison operation is the longest operation after division, see Table 1.

4. CONCLUSION

The new efficient algorithm of $O(lg\ N)$ complexity for clipping lines against convex window in E^2 has been developed. Edges of the given convex polygon can be arbitrarily oriented. It also proved the applicability of Computational Geometry results[6] even for small N. Similarly as the Rappaport's algorithm the proposed algorithm can be easily modified for polygon clipping. The suggested algorithm also proved the duality principle with the problem *point-in-polygon*, see[7, 8, 9]. It also proved applicability of principles of Computational Geometry results[6] even for small N. Similarly as Rappaport's algorithm the proposed algorithm can be modified for polygon clipping, where the clipped polygon might be nonconvex. Superiority of the proposed algorithm over CB, ECB, and Rappaport's algorithms was proved by theoretical estimations and experimental results.

All tests were implemented in Borland C++ on PC 486/33 MHz 256KB Cache. It is expected that for workstations the efficiency ν will be higher than for PC 486 as the comparison operation is the longest operation used in the algorithm, see Table 2, and the timing ratio of operations on workstations is better.

Acknowledgements—The author would like to express his thanks to students of Computer Graphics courses at the University of West Bohemia in Plzen and Charles's University in Prague who stimulated this work, especially to Mr. P. Blaha for careful test implementations and verification of the proposed algorithm, dr. A. Ferko and dr. F. Ježek for reading a manuscript, critical comments and suggestion they made and to anonymous referees who made critical and constructive recommendations that improved this paper very much.

REFERENCES

1. Y. D. Liang and B. A. Barsky, An analysis and algorithms for polygon clipping. *CACM* **26**(11), 868–876 (1983).
2. Y. D. Liang and B. A. Barsky, A new concept and method for line clipping. *ACM Trans. Graph.* **3**(1), 1–22 (1984).
3. M. Cyrus and J. Beck, Generalized two and three dimensional clipping. *Comp. & Graph.* **3**(1), 23–28 (1978).
4. A. Rappaport, An efficient algorithm for line and polygon clipping. *Visual Comp.* **7**(1), 19–28 (1991).
5. V. Skala, An efficient algorithm for line clipping by convex polygon. *Comp. & Graph.* **17**(4), 417–421 (1993).
6. B. Chazelle and D. P. Dobkin, Intersection of convex objects in two and three dimensions. *JACM* **34**(1), 1–27 (1987).
7. P. F. Preparata and M. I. Shamos, *Computational geometry: An introduction*, Springer Verlag, Berlin (1985).
8. H. P. Nielsen, *An intersection test using dual figures*, Technical report GKTR-0892, Department of Graphical Communication, Technical University of Denmark, Lyngby (October 1992).
9. H. P. Nielsen, *Line clipping using semi-homogeneous coordinates*. Submitted.

ADDITIONAL RECOMMENDED READINGS

S. S. Abi-Ezzi, and M. J. Wozny. Factoring a homogeneous transformation for a more efficient graphics pipeline, *Comp. Graph. Forum* **9**(3), 245–255 (1990).
K. Akeley, and C. P. Korobkin, Efficient graphics processor for clipping polygons, US Patent No. 5 051 737 (1991).
R. D. Andreev, Algorithm for clipping arbitrary polygons. *Comp. Graph. Forum* **8**(3), 183–192 (1989).
R. Andreev, and E. Sofianska, New algorithm for 2-dimensional line clipping. *Comp. & Graph.* **15**(4), 519–526 (1991).
A. Arokiasamy, Homogeneous coordinates and the principle of duality in two dimensional clipping. *Comp. & Graph.* **13**(1), 99–100 (1989).
J. F. Blinn, and M. E. Newell. Clipping using homogeneous coordinates. *Comp. Graph. (SIGGRAPH'78)* **12**, 245–251 (1978).
J. F. Blinn, A trip down to graphics pipeline—Line clipping. *IEEE Comp. Graph. Appl.* **11**(1), 98–105 (1991).
E. A. Brewer, and B. A. Barsky, Clipping after projection: An improved perspective pipeline. Submitted.
A. Burkert, and S. Noll, Fast algorithm for polygon clipping with 3D windows. *Eurographics '88 Proceedings*, 405–419 (1988).
B. Chazelle, An optimal algorithm for intersecting three-dimensional convex polyhedra. *SIAM J. Comp.* **21**(4), 671–696 (1992).
F. Cheng, and Y. Yen, A parallel line clipping algorithm and its implementation. *CGI '89 Conference Proceedings* (1989).
J. D. Day, A comparisons of line clipping algorithms, Queensland University of Technology, School of Computing Science, Report 2-91 (1991).
J. D. Day, A new two dimensional line clipping algorithms for small windows, Queensland University of Technology, School of Computing Science, Report 3-91 (1991).
J. D. Day, A new two dimensional line clipping algorithms for small windows. *Comp. Graph. Forum* **11**(4), 241–245 (1992).
M. Dorr, A new approach to parametric line clipping. *Comp. & Graph.* **14**(3/4), 449–464 (1990).
V. J. Duvanenko, W. E. Robins, and R. S. Gyurcsik, Improving line segment clipping. *Dr. Dobb's J. Software Tools* **15**(7), 36, 38, 40, 42, 44–45, 98, 100 (1990).
V. J. Duvanenko, W. E. Robins, and R. S. Gyurcsik, Simple and efficient 2D and 3D span clipping algorithms. *Comp. & Graph.* **17**(1), 39–54 (1993).
D. Y. Fong, and J. Chu, A string pattern recognition approach to polygon clipping. *Patt. Recognition* **23**(8), 879–892 (1992).
K. Y. Fung, T. M. Nicholl, R. E. Tarjan, and C. J. Van Wyk, Simplified linear time jordan sorting and polygon clipping information processing letters, **35**, 85–92 (1990).
I. Herman, and J. Reviczky, Some remarks on the modelling clip problem. *Comp. Graph. Forum* **7**(4), 265–272 (1988).
J. Hubl, and I. Herman, Modelling clip: Some more results. *Comp. Graph. Forum* **9**, 101–107 (1990).
J. Hubl, A note on 3D-clip optimisation. *Comp. Graph. Forum* **12**(2), 159–160 (1993).
S. Kaijian, J. A. Edwards, and D. C. Cooper, An efficient line clipping algorithm. *Comp. & Graph.* **14**(2), 297–301 (1990).
A. C. Kilgour, Unifying vector and polygon algorithm for scan conversion and clipping, TR CSC/87/R7, University of Glasgow (May 1987).
G. Kramer, Notes on the mathematics of PHIGS output pipeline. *Comp. Graph. Forum* **8**(3), 219–226 (1989).
G. Krammer, A line clipping algorithm and its analysis. *Comp. Graph. Forum (EG '92 Conference Proceedings)* **11**(3), C253–266 (1992).
Y. Liang, and B. A. Barsky, *The optimal tree algorithm for line clipping*, Technical Paper distributed at Eurographics '92 Conference, Cambridge (1992).
P. G. Maillot, A new, fast method for 2D polygon clipping: Analysis and software implementation. *ACM Trans. Graph.* **11**(3), 276–290 (1992).
A. Margalit, and G. Knott, An algorithm for computing the union, intersection or difference of two polygons. *Comp. & Graph.* **13**, 167–183 (1989).

N. Max, Polygon clipping—Response. *CACM* **36**(1), 115 (1993).

T. M. Nicholl, D. T. Lee, and R. A. Nicholl, An efficient new algorithm for 2D line clipping: Its development and analysis. *ACM Comp. Graph.* **21**(4), 253–262 (1987).

R. A. Nicholl, and T. M. Nicholl, *A definition of polygon clipping,* Report No. 281, Computer Science Department, University of West Ontario (1991).

H. P. Nielsen, *An intersection test using dual figures,* Technical Report GKTR-0892, Department of Graphical Communication, Technical University of Denmark, Lyngby (October 1992).

H. P. Nielsen, *Line clipping using semi-homogeneous coordinates.* Submitted.

R. M. O'Bara, and S. Abi-Ezzi, An analysis of modeling clip. *EG '89 Conference Proceedings,* 367–380 (1989).

D. Pinedo, Window clipping methods in graphics accelerators. *IEEE Comp. Graph. Appl.* **11**(3), 75–84 (1991).

N. C. Sharma, and S. Manohar, Line clipping revisited: Two efficient algorithms based on simple geometric observations. *Comp. & Graph.* **16**(1), 51–54 (1992).

V. Skala, Algorithms for 2D line clipping. *CGI '89 Conference Proceedings,* 121–128 (1989).

V. Skala, Algorithms for 2D line clipping. *EG '89 Conference Proceedings,* 355–367 (1989).

V. Skala, Algorithm for line clipping in E2 for convex window (in Czech). *Algorithms '93 Conference Proceedings,* Bratislava (1993).

V. Skala, *An efficient algorithm for line clipping by convex polygon,* Preprint No. 38, University of West Bohemia, Plzen (1993).

M. Slater, and A. B. Barsky, 2D line and polygon clipping based on space subdivision. *The Visual Comp.* (in press).

M. S. Sobkow, P. Pospisil, and Y.-H. Yang, A fast two-dimensional line clipping algorithm via line encoding. *Comp. & Graph.* **11**(4), 459–467 (1987).

R. F. Sproull, and I. E. Sutherland, A clipping divider. *Proc. APFIS FJCC* (1968).

I. E. Sutherland, and G. W. Hodgman, Reentrant polygon clipping. *CACM* **17**(1), 32–42 (1974).

T. Theoharis, and I. Page, Two parallel methods for polygon clipping. *Comp. Graph. Forum* **8**, 107–114 (1989).

B. Vatti, Polygon clipping—Response. *CACM* **36**(1), 115 (1993).

D. N. Ying, A new algorithm for polygon clipping and boolean operations. University of Zhejiang Internal Report, Hangzhou, China (1991).

L. Yong-Kui, A new algorithm for line clipping by convex polygon. Submitted.

M. Zachristen, Yet another remark on the modeling clip problem. *Comp. Graph. Forum* **8**, 237–238 (1989).

MEMORY SAVING TECHNIQUE FOR SPACE SUBDIVISION TECHNIQUE

Václav Skala
Computer Graphics and Scientific Visualization Laboratory
Department of Informatics and Computer Science
University of West Bohemia, Plzen, Czech Republic

Abstract. The ray tracing technique is very often used for image synthesis because it gives the possibility to render specular objects. Many techniques have been developed for ray tracing acceleration, more or less sofisticated which, are generally speaking, not easy to implement. A simple method how to speed up the primary and secondary ray tracing has been developed. The suggested method based on space subdivision method (not necessarily uniform) is convenient for scenes that consist of many small objects, resp. facets (for experiments only triangles have been used).

Key words: ray tracing, acceleration, data structure, rendering, computer graphics, algorithm complexity

1. Introduction

It is well known that ray tracing is the only one method which is capable of rendering specular effects like reflection through an object and refraction, even some other methods do integrate some features of ray tracing in order to handle these effects.

The bottleneck operation of ray tracing is *the search for the* **first** *object intersected by a given ray.*

There are many techniques based on many sofisticated algorithms [1], like space subdivision methods (uniform, non-uniform, adaptive), octree methods, etc.

The proposed Binary Map of Space Subdivision Method (BMSSM) is based on simple presumptions:

– scene consists of dozens of small objects with regard to the scene volume,

– scene consists of all kinds of objects, like polyhedrons, solids, given by an implicit functions $F(x, y, z) = 0$ or by parametrically defined patches, CSG trees etc.,

2. Principle of the proposed method

Let us suppose the parallel primary ray tracing and that the image resolution is $n \times m$ pixels and that scene consists of p facets, resp. objects.

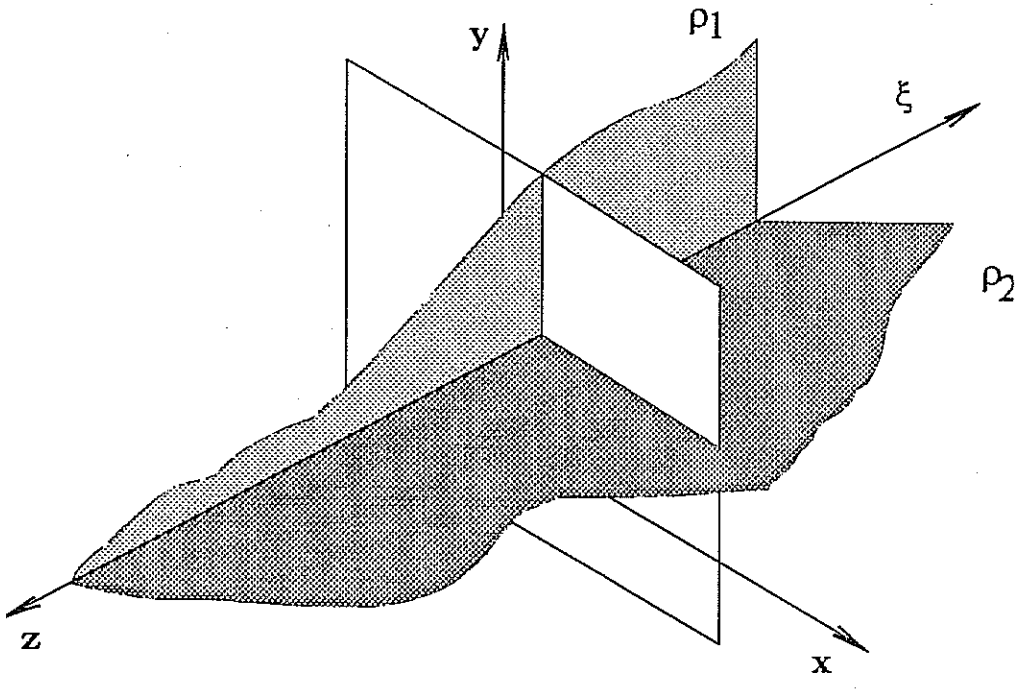


Fig. 1. Perspective primary ray tracing (an observer is in infinity for Parallel Primary Ray Tracing).

The fundamental requirement for ray tracing method is to find a facet, or generally an object and a facet, which is intersected by a given ray and which is the closest facet to the observer, as fast as possible. In general, it means to find all intersection points of the given ray with all facets and select the nearest one. Some bounding volumes like sphere or min-max box volumes are often used to speed up the computation of intersections.

For parallel primary ray tracing the min-max box test

$$x_{min} \leq x \leq x_{max} \quad \text{and} \quad y_{min} \leq y \leq y_{max}$$

or the circle bounding test

$$(x - x_s)^2 + (y - y_s)^2 < r^2$$

will be used for possible intersection detection because those tests seem to be the fastest possible solution.

The proposed BMSSM technique is a method which is based on a principle that any ray ξ can be given as an intersection of two non-collinear planes ρ_1 and ρ_2. In the case of parallel primary ray tracing we can choose planes so that ρ_1 is collinear to $z - x$ plane and ρ_2 is collinear to $y - z$ plane, see fig. 1.

Because the required resolution of the final image (which can be higher than the actual display resolution) is known, it is possible to define a ray at position (i, j) as the

intersection of i_{ρ_1}, j_{ρ_2} planes, where i, j means i-th row and j-th column (it is obvious that the plane equations must be expressed in world coordinates).

Now it is possible to define for each row an ordered p-tuple ${}^iR$ of binary values so that for i-th row is

$$ {}^iR = [{}^ir_1, {}^ir_2, \ldots, {}^ir_k, \ldots, {}^ir_p]^T, \quad 1 \leq k \leq p \quad 1 \leq i \leq n\,, $$

where ${}^ir_k$ indicates that the facet(k) is intersected by the plane ${}^i\rho_1$, p is a number of facets in the given scene, n is a number of rows.

Then it is possible to define an ordered n-tuple of binary maps $\mathcal{R}$ as

$$ \mathcal{R} = [{}^1R, {}^2R, \ldots, {}^nR]^T\,. $$

Similarly for j-th row

$$ {}^jS = [{}^js_1, {}^js_2, \ldots, {}^js_k, \ldots, {}^js_p]^T \quad 1 \leq k \leq p \quad 1 \leq j \leq m\,, $$

where ${}^js_k$ indicates that the facet(k) is intersected by the plane ${}^j\rho_2$, m is a number of columns.

Then it is possible to define an ordered m-tuple of binary maps $\mathcal{S}$ as

$$ \mathcal{S} = [{}^1S, {}^2S, \ldots, {}^mS]^T\,. $$

${}^iR$ are p-tuples which contain only logical values ${}^ir_k$, so that:

${}^ir_k = 0$ means that the given plane ${}^i\rho_1$ does not intersect a given facet(k),

${}^ir_k = 14$ means that the given plane ${}^i\rho_1$ intersects a given facet(k).

Similarly for p-tuples ${}^jS$.

It can be observed that p-tuples ${}^iR$ and ${}^jS$ have the same cardinality, e.g. number of columns for all possible primary rays,

$$ \mathrm{card}({}^iR) = \mathrm{card}({}^jS) = p \text{ for all } i, j\,, $$

while

$$ \mathrm{card}(\mathcal{R}) = n\mathrm{card}(\mathcal{S}) = m\,. $$

It is possible to define generalized bitwise operations + and & with p-tuples as

$$ {}^iR + {}^jS = [r_1 + s_1, r_2 + s_2, \ldots, r_p + s_p]^T\,, $$

$$ {}^iR \& {}^jS = [r_1 \& s_1, r_2 \& s_2, \ldots, r_p \& s_p]^T\,, $$

where &, resp.+, means boolean multiplication, resp. addition.

The ray at the position (i,j) is given as an intersection of ${}^i\rho_1$ and ${}^j\rho_2$ planes. It is possible to define a p-tuple ${}^{ij}Q$ which is defined as

$$ {}^{ij}Q = {}^i R \& {}^jS = [{}^{ij}q_1, {}^{ij}q_2, \ldots, {}^{ij}q_p]^T\,. $$

It is obvious that:

– if ${}^{ij}qk = 0$ then there is no intersection of the ray at position (i,j) with the facet(k),

– if ${}^{ij}qk = 1$ then the given ray at position (i,j) do intersect *minimal rectangular bounding box* of the facet(k) and it is necessary to use a more detailed test for facet-ray intersection computation.

Some similarities of the BMSSM method can be seen with uniform and adaptive space subdivision acceleration method [1], but used data structures are fairly simple and easy to implement. A_vector of bits can be the simplest way of implementation.

This approach offers one, probably, unusual feature which leads to **better image consistency**. It is well known that small objects may disappear from the final image as an observer moves. The BMSSM method gives at least the basic method how to detect image consistency. The fundamental requirement is that **all** facets must be intersected at least by one ray if we consider each facet alone in the scene.

Let us define p-tuple

$$\mathcal{V} = [v_1, v_2, \ldots, v_p]^T$$

as

$$\mathcal{V} = +_{i=1}^{n}({}^{i}R) = [+_{i=1}^{n}({}^{i}r_1), +_{i=1}^{n}({}^{i}r_2), \ldots, +_{i=1}^{n}({}^{i}r_k), \ldots, +_{i=1}^{n}({}^{i}r_p)]^T .$$

If any k exists so that $v_k = 0$ ($1 \leq k \leq p$), then the facet(k) is not intersected by **any** plane ${}^{i}\rho_1$, for all $i = 1, \ldots, n$.

It means that p-tuple $\mathcal{V}$ must have all items equal to the value 1, e.g.

$$\mathcal{V} +_{i=1}^{n}({}^{i}R) = [1, 1, \ldots, 1]^T .$$

Similarly if $\mathcal{W} = [w_1, w_2, \ldots, w_p]^T$ as

$$\mathcal{W} = +_{i=1}^{m}{}^{j}S = [1, 1, \ldots, 1]^T .$$

If any k exists so that $w_k = 0$ ($1 \leq k \leq p$) then the facet(k) is not intersected by **any** plane ${}^{j}\rho_1$, for all $j = 1, \ldots, m$.

The BMSSM method suppose that all facets are small according to the final image size. Of course if some large objects appear in the scene it is necessary to see that they will be often tested whether they have an intersection point with a given ray. In this case it is recommended to split such an object into small facets, if possible.

It can be easily proved that BMSSM is equivalent to minmax box test as far as the functionality is concerned.

3. Theoretical complexity estimation

There is a small overhead of the BMSSM method because it is necessary to determine the ${}^{i}R$ and ${}^{j}S$ p-tuples and some additional memory is needed to store $\mathbb{R}$ and $\mathbb{S}$ binary maps. It is important to point out that if the image resolution is $n \times m$ pixels and p is a number of facets in the given scene then the complexity of the overhead is given by:
– the complexity of determining ${}^{i}R$ and ${}^{j}S$ is only

$$O(n, m, p) = [n + m]pc_0 ,$$

where c_0 is the cost for determining whether a *plane* do intersect a facet, e.g. using eparation test,
– the memory requirements are approximately equal to

$$(n+m)p \quad [bits]\,.$$

Of course, it is not generally possible to avoid the ray tracing complexity for primary rays, which is given by

$$O_1(n,m,p) = nm[c_1 p + c_2 p_1]\,,$$

where: – c_1 is the cost for a *minmax* box or a sphere bounding volume tests, if used,
– c_2 covers the cost of detailed test for ray intersection with a_given facet(k)
– p_1 is a number of minmax boxes intersected by the given ray.

For each ray it is necessary to evaluate **only** boolean expression with bit vectors

$$^{ij}Q =^{i} R\&^{j}S\,,$$

which is *faster* than a test which evaluates an intersection with bounding volumes *for all facets.*

The complexity O_2 of primary ray tracing is generally given as

$$O_2(n,m,p) = mn[c_3 p + c_2 p_1]\,,$$

where c_3 covers the cost of $^{ij}q_k =^{i} r_k\&^{j}s_k$ computation for a given k and the cost of the test whether $^{ij}q_k \neq 0$ for all $k = 1,\ldots,p$. If we compare complexities O_1(n,m,p) and O_2(n,m,p-) we can see that BMSSM will be faster if and only if

$$O_1(n,m,p) > O_2(n,m,p) \quad \text{i.e.} \quad c_1 > c_3$$

Before making any comparisons it is necessary to point out that time needed for each operation ($:=, <, \pm, *, /$) does differ from computer to computer, see tab. 1.
So, it is possible for parallel primary ray tracing to estimate time T_{minmax} needed for minmax box volume test (float operations considered only with some probability estimations) if the cost 18 of one **for** cycle statement is considered then

$$T_{minmax} = (0, 4*23/12, 0, 0, 0) + 18 = 114\,.$$

The time T_{BMSSM} can be estimated as (see appendix)

$$T_{BMSSM} = 90\,.$$

It is possible to estimate the efficiency ν of the proposed BMSSM which can be expressed as (if worst case considered – ray intersects all facets)

$$\nu = \frac{T_{minmax}}{T_{BMSSM}} = \frac{218}{90} = 1.26\,.$$

If a circle bounding test is used, then time T_{circle} needed for this test (cost 18 of one **for** cycle statement must be considered) can be estimated as

$$T_{circle} = (2, 1, 3, 2, 0) + 18 = 222$$

PC 386DX/387 25 MHz

	:=	<	±	*	/
int	14	42	10	40	58
float	204	260	80	82	154

PC 486 33 MHz 64KB cache – **selected for evaluation**

	:=	<	±	*	/
int	5	9	3	26	44
float	33	50	16	20	114

Time is in 1/10 sec. for 5 000 000 operations

Tab. 1. Time needed for operations $(:=, <, \pm, *, /)$.

(2 assignments must be taken into account for storing $(x - x_s)$ and $(y - y_s)$)

$$\nu = \frac{T_{circle}}{T_{BMSSM}} = \frac{222}{90} = 2.54\,,$$

$$\nu = \frac{T_{circle}}{T_{minmax}} = \frac{222}{114} = 1.94\,.$$

Those results mean that **even for parallel primary** ray tracing the BMSSM should be faster than the usage of the minmax box test. Generally not every facet is intersected by a ray the expected ratio ν should be higher because some tests in BMSSM will be skipped.

4. Perspective primary ray tracing

The parallel primary ray tracing is a very special case of perspective ray tracing. Therefore it would be desirable to find a modification of BMSSM for perspective primary ray tracing, see fig.1. In this case it is necessary to use a sphere bounding volume or minmax box tests. If the minmax box test is considered the computational cost is significantly higher for perspective primary ray tracing than the sphere bounding volume, because it is necessary to compute the nearest intersection point of the ray with the minmax box.

In case of sphere bounding volume test, see fig. 2, it can be shown that if we consider a bounding sphere in the form

$$(x - x_q)^T (x - x_q) - r^2 = 0\,,$$

and a ray as

$$x(t) = x_A + s_2 t\,.$$

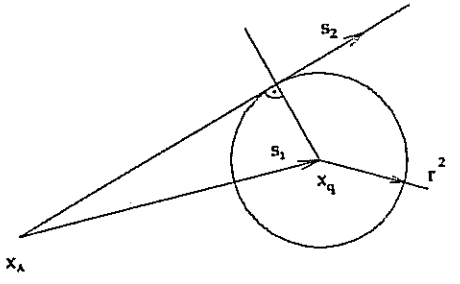


Fig. 2. Sphere bounding volume test.

Then it is possible to write

$$at^2 + bt + c = 0\,,$$

where

$$a = s_2^T s_2 \quad b = -2s_1^T s_2 \quad c = (s_1^T s_1 - r^2)\,,$$

and

$$s_1 = x_q - x_A\,.$$

In this case an intersection of the given ray with a given sphere exists if and only if

$$(s_1^T s_2)^2 - s_2^T s_2(s_1^T s_1 - r^2) \geq 0\,,$$

where $(s^T{}_1 s_1 - r^2)$ can be precomputed as it does not depend onthe ray because it depends on the sphere bounding volume and observer positions only.

If number of facets or objects is high enough it is convenient to normalize the vector s_2 so that $|s_2| = 1$. The condition can be rewritten as

$$(s_1^T s_2)^2 - q \geq 0\,,$$

where $q = ({}^T_s{}_1 s_1 - r^2)$ is constant for the given facet or object.

If the sphere bounding volume is used then time needed can be estimated as ($w := s_1^T s_2$ (1,0,2,3,0) ; $w * w - q > 0$ (0,1,1,1,0))

$$T_{sphere} = (1, 1, 3, 4, 0) + 18 = 229\,.$$

Then the estimation of the efficiency ν can be expressed as (the cost of the cycle 18 must be considered)

$$\nu = \frac{T_{minmax}}{T_{sphere}} = \frac{218}{229} = 0.95\,.$$

So we have got the theoretical ratio ν that was expected because the minmax box test should be faster for **parallel primary** ray tracing than the sphere bounding volume test for **perspective** case, but the ratio is approximately equal to one.

Now it is possible to compute the expected ratio ν as

$$\nu = \frac{T_{sphere}}{T_{BMSSM}} = \frac{229}{90} = 2.54\,.$$

It means, in the worst expected case, that BMSSM should be **2.54 times faster** than the sphere bounding volume test.

5. Secondary ray tracing

The usage of the BMSSM for the secondary ray tracing is quite simple and straightforward. Let us consider a similar approach as in the primary parallel ray tracing case and divide the given space in the z_axis direction by plane ρ_3, too. In this way binary maps

$$^kT = [^kt_1, ^kt_2, \ldots, ^kt_p]^T \quad 1 \leq k \leq L$$

are obtained with similar properties as binary maps ^iR and ^kS, where L is a number slices in z_direction, see fig. 3.

Similarly we can define $\mathcal{T}$ L-tuple as

$$\mathcal{T} = [T_1, T_2, \ldots, T_L]^T .$$

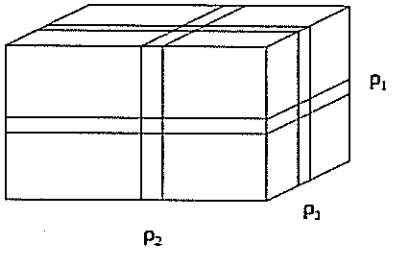


Fig. 3.

The BMSSM method is actually used for detecting whether the given ray can intersect bounding minmax box volume, i.e. supervoxel. Each supervoxel can be imagined as an intersection of three orthogonal slices ^iR, ^jS, ^kT .

The bit map for a supervoxel at the position (i,j,k) can be expressed similarly to the primary ray tracing case as

$$^{ijk}W = {}^iR \,\&\, ^jS \,\&\, ^kT ,$$

where:

$^{ijk}W = [^{ijk}w_1, ^{ijk}w_2, \ldots, ^{ijk}w_p]^T$, and

$^{ijk}w_r = 0$ means that the r-th object does not interfere with the supervoxel (i,j,k),

$^{ijk}w_r = 0$ means that the r-th object does not interfere with the supervoxel (i,j,k).

The proposed BMSSM method is similar to the **Space Subdivision Method (SSM)** that is very often used for substantial ray tracing computation speed up. The SSM method is based on space subdivision into **supervoxels** and **each** supervoxel is associated with information which objects interfere with such a supervoxel, see fig. 4.

This structure is actually an inverted list and can be generally used for interference tests with **non convex** objects, too.

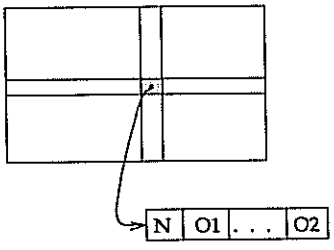


Fig. 4. Space subdivision method – SSM.

Time considerations

In the case of the usage of BMSSM method it is necessary to evaluate expression with bit maps

$$^{ijk}W = {}^{i}R \& {}^{j}S \& {}^{k}T\,,$$

where $^{ijk}W = [^{ijk}w_1, {}^{ijk}w_2, \ldots, {}^{ijk}w_p]^T$. It is not substantial according to the cost of intersection computation of the given ray and selected objects.

In both SSM and BMSSM methods a 3D DDA algorithm can be used for efficient finding of intersection of a supervoxel with the given ray.

Memory considerations

The great disadvantage of the SSM method for large scenes is the memory requirements, that can be approximately expressed as

$$M_{SSM} = 2N^3(q+1+1) \quad [Bytes]\,,$$

where: N^3 is a number of supervoxels for the given scene (N for each direction), q is an average number of objects interfering withthe given supervoxel ($0 < q < p$), if we count 2 Bytes for integer (counter of objects) and pointer (objects identification) implementation.

It is obvious that the amount of the required memory grows extremely fast so the SSM method can be implemented **only for small N**.

Let us consider a scene which is subdivided into $N \times N \times N$ supervoxels. It can be easily shown that the memory requirements for BMSSM can be expressed as

$$M_{BMSSM} = 3Np \quad [bits]\,,$$

or

$$M_{BMSSM} = \frac{3}{8}N\ p\ [Bytes]\,.$$

Then the memory efficiency ν_{mem} can be expressed as

$$\nu_{mem} = \frac{M_{SSM}}{M_{BMSSM}} = \frac{2N^3(q+2)}{\frac{3}{8}N\ p} = \frac{16}{3}\ \frac{q+2}{p}\ N^2\,.$$

There are some special cases that should be mentioned

a) large objects

For large objects can be seen that q will converge to p, so the memory requirements can be expressed ($p \gg 2$) as

$$\nu_{mem} = \frac{M_{SSM}}{M_{BMSSM}} \doteq \frac{16}{3} N^2 ,$$

b) small objects

case when each supervoxel contains one object in average, i.e. $q \doteq 1$; then

$$\nu_{mem} = \frac{M_{SSM}}{M_{BMSSM}} \doteq 16 \frac{N^2}{p} ,$$

c) small objects and extremely sparse

For small objects and extremely sparse $0 \leq q \ll 1$, so

$$\nu_{mem} = \frac{32}{3} \frac{N^2}{p} \doteq 11 \frac{N^2}{p} .$$

All those cases are very special but all show that the proposed BMSSM method **more efficient** according to the SSM method as far as the memory requirements are concerned.

The average number of intersection q can be defined as

$$q = p \frac{r^3}{N^3} ,$$

where r^3 is a average size on an object in supervoxels.

6. Experimental results

Basic experiments have been made on PC 386 for image resolution 256 × 256 pixels.

Parallel primary ray tracing

The following results were obtained for parallel primary ray tracing:

Number of objects	size of objects (one side of the box) 10	16	25	40	63	100
400	10.38	9.52	8.04	6.13	4.74	4.10
630	11.39	10.31	8.55	6.44	4.90	4.22
1000	11.96	10.72	8.83	6.59	4.98	4.28
1600	12.65	11.30	9.22	6.82	5.15	4.44
2500	13.06	11.64	9.48	6.99	5.26	4.55
4000	13.72	12.18	9.86	7.22	5.39	4.51
6300	14.03	12.43	10.03	7.30	5.43	4.65

Tab. 2. Time efficiency ν of the BMSSM against minmax box volume test.

Number of objects	size of objects (one side of the box) 10	16	25	40	63	100
400	26.11	23.40	19.10	13.79	9.86	7.64
630	28.62	25.30	20.29	14.43	10.17	7.82
1000	29.95	26.23	20.88	14.73	10.32	7.92
1600	31.91	27.77	21.87	15.24	10.59	8.11
2500	33.07	28.71	22.53	15.63	10.83	8.29
4000	33.97	29.40	22.96	15.83	10.91	8.35
6300	38.94	33.64	26.24	18.03	12.39	9.48

Tab. 3. Time efficiency ν of the BMSSM against sphere volume test.

From experimental results the time efficiency ν for circle and minmax box tests was $\nu \in < 1.81 , 2.52 >$.

Perspective primary ray tracing results

Number of objects	size of objects (one side of the box) 10	16	25	40	63	100
400	346.57	310.27	253.10	183.01	130.76	101.18
630	7 379.94	335.89	269.40	191.72	135.10	103.83
1000	399.93	350.26	278.72	196.73	137.80	105.68
1600	423.53	368.58	290.28	-	-	-

Tab. 4. Time efficiency ν of the BMSSM against minmax box.

Number of objects	size of objects (one side of the box) 10	16	25	40	63	100
400	33.80	30.48	24.76	17.89	12.83	9.88
630	37.19	32.76	26.32	18.73	13.16	10.15
1000	38.74	33.93	27.01	19.05	13.35	10.24
1600	42.07	36.58	28.86	-	-	-

Tab. 5. Time efficiency ν of the BMSSM against sphere volume.

Number of objects	size of objects (one side of the box) 10	16	25	40	63	100
400	0.72	1.70	3.90	9.44	21.99	50.82
630	1.14	2.68	6.15	14.86	34.69	80.41
1000	1.80	4.25	9.76	23.57	54.82	126.74
1600	2.89	6.81	15.66	37.85	87.94	202.91
2500	4.51	10.65	24.46	59.08	137.34	317.13
4000	7.23	17.06	39.23	94.77	220.22	507.10
6300	11.39	26.89	61.83	149.37	347.27	799.03

Tab. 6. Average number of intersection for a ray.

Number of objects	size of objects (one side of the box) 10	16	25	40	63	100
400	6	8	12	21	38	80
630	8	11	17	33	61	121
1000	10	15	23	48	93	192
1600	12	20	32	65	131	304
2500	14	26	45	94	196	436
4000	19	36	67	141	306	689
6300	27	49	98	206	450	1054

Tab. 7. Maximum number of intersection for primary rays.

The shown results **do not cover** time for complete detailed tests. Number of the facets used within the tests has been limited due to available memory on PC.

Experiments have shown that it is not convenient for **primary** ray tracing to subdivide space in z-direction.

7. Conclusion

The presented BMSSM method speed up the primary and secondary ray tracing substantially especially if small facets are used in the scene. The advantage of this approach is seemed in a simple data structure which can be used to represent the $\mathbb{R}$ and $\mathbb{S}$, resp. $\mathbb{T}$ tuples. The BMSSM method can be used even for non triangular facets, even for CGS trees and there is a straightforward usage of hierarchical data structures for solids or facets. Tuples $\mathbb{R}$ and $\mathbb{S}$, resp. $\mathbb{T}$ can be pre-computed at the scene definition stage to speed up the ray tracing computation.

The **fundamental advantage** of the proposed method is seen in small memory need with regard to space subdivision technique.

The proposed method gives at least the basic criterion for scene consistency evaluation. From the programmer's point of view, the BMSSM seems to be convenient for application of Object Oriented Programming techniques, too.

Acknowledgments
The author would like to express his thanks to Mr.P.Sebranek for testing BMSSM method and to all who contributed to this work, especially to his students as many suggestions were proposed.

Appendix

The T_{BMSSM} time can be determined from the algorithm shown bellow.

```
/* count is a number of 32 bits words */
count = No objects >> 5;
if ((No objects % 32) != 0)count++;
void COMPUTE (int i, int j, int count)
{ int k;
  long kk;
  unsigned long mask;
  unsigned long huge *mask_x, huge *mask_y;
  mask_y = array_y[i]; /* array stores iS bit maps */
  mask_x = array_x[j]; / * array stores jR bit maps */
  for (k = 0; k < count; k++)
  { mask = ( *( mask_x + k ) & ( *( mask_y + k ));
    kk = k << 5;
    while ( mask != 0L )
    { if (( mask & 1L) != 0 ) DETAIL_TEST (kk);
      /* DETAIL_TEST (kk) is a detail test for object kk */
      mask = mask >> 1; kk++;
    }
  }
```

References

1989
[1] Glassner A.S.: An Introduction to Ray Tracing. AP.
[2] Hansmann W., Hopgood F.R.A., Strasser W.: Proc. EUROGRAPHICS'89. North Holland.

1991
[3] Post F.H., Barth W.: Proc. EUROGRAPHICS'91. North Holland.

1992
[4] Speer L.R.: An updated cross indexed guide to the ray tracing literature. ACM SIGGRAPH, 2.

Václav Skala is a Research Professor of Computer Science at the University of West Bohemia in Plzen, previously Institute of Technology. He studied Technical Cybernetics and Computer Science at the Institute of Technology in Plze. In 1975 he took a master degree in Computer Science followed by a PhD degree specializing in Relational Database Systems at the Czech Technical University in Prague. Since 1975 to 1981 he worked as a researcher. In 1978 he studied Computer Science at MEI in Moscow and in the academic year 1983-84 Computer Graphics at Brunel University in London. In 1981 he took up position as a senior lecturer at the Cybernetics Department teaching Programming Languages, Database Systems and Computer Graphics.

An Interesting Modification to the Bresenham Algorithm for Hidden-Line Solution

Václav Skala

Department of Technical Cybernetics
Technical University, Nejedlého sady 14
306 14 PLZEŇ, CZECHOSLOVAKIA

1. Introduction

The solution of many engineering problems have as a result functions of two variables, that can be given either by an explicit function description, or by a table of the function values. The functions, that can be given either by an explicit function description, or by a table of the function values. The functions have been usually plotted with respect to visibility. The subprograms for plotting the functions of two variables were not so simple ([6]-[7]) although visibility may be achieved by the relatively simple algorithm at the physical level of the drawing, if we assume raster graphics devices are used. The Bresenham algorithm for drawing line segments can be modified in order to enable the drawing of explicit functions of two variables with respect to the visibility.

Though the order of curve drawing is essential for the method used the algorithm has not been published yet. Williamson [7] solved the problem by fixing the position of the view-point, Watkins [6] only pointed out that some rotation angles can cause wrong hidden-line elimination and Boutland's method [1] can use only one angle.

Therefore the algorithm that ensure the right order of the curve's drawing is presented here.

2. Problem specification

Let us have an explicit function of two variables x and y

$$z = f (x , y)$$

where: $x \in \langle ax, bx \rangle$ and $y \in \langle ay, by \rangle$

and we want to display that function by using the graphical raster display or plotter. For many scientific problems is enough to show the behaviour of that function by drawing the function slices according to the x and y axises, e.g. curves

$$z = f (x , y_i) \quad i=1,\ldots,n$$

NATO ASI Series, Vol. F17
Fundamental Algorithms for Computer Graphics
Edited by R. A. Earnshaw

where: $x \in \langle ax, bx \rangle$ and $ay = y_1 < y_2 < \ldots < y_n = by$

and curves:

$$z = f(x_j, y) \qquad j = 1, \ldots, m$$

where: $y \in \langle ay, by \rangle$ and $ax = x_1 < x_2 < \ldots < x_m = bx$

The given function can be represented either by a function specification or by a table of the function values for the grid points in the x-y plane. If the function is complex it can be very difficult to imagine the function behaviour because some parts are in the reality invisible. The problem has been solved by Watkins [6], Williamson [7] and Boutland [1] relatively very successfully. The principle of the solution is generally very simple. If we have drawn the first two slices parallel to the x axis we have produced two curves and the space between them is the strip of invisibility. Let us suppose that we draw the lines in the direction from foreground to background. Now if we want to draw the third curve it is obvious that those parts which are passing through the strip of invisibility are invisible and therefore ought not to be drawn, see figure 1.

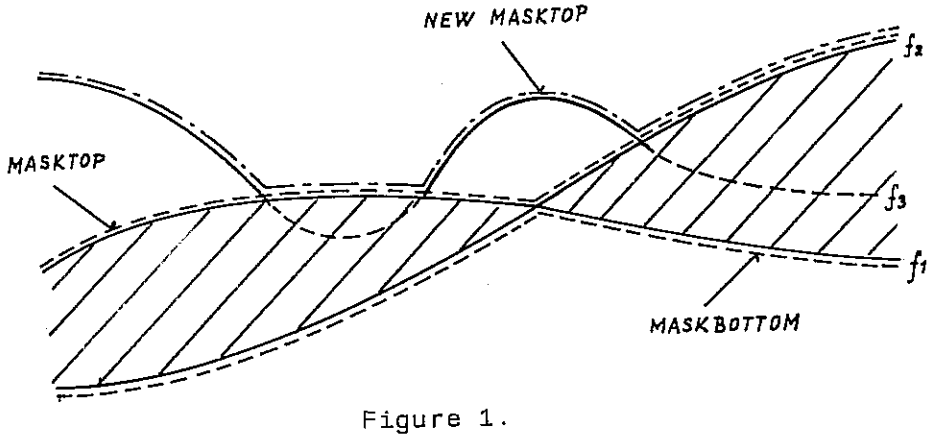
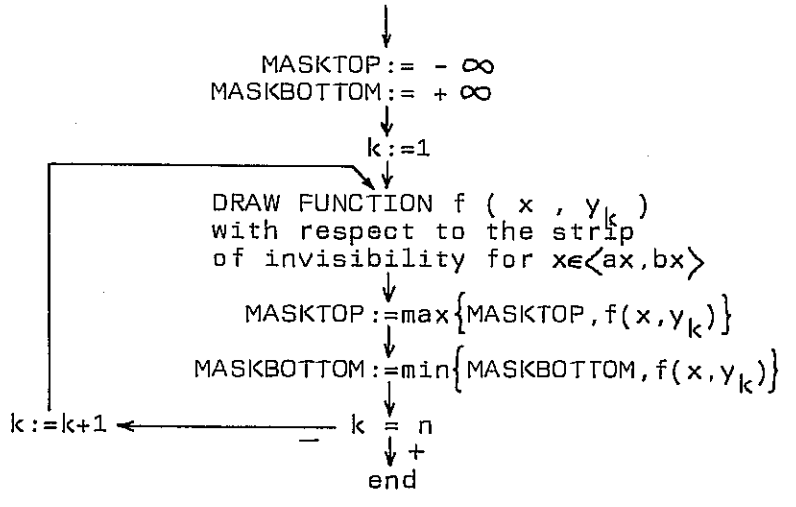


Figure 1.

If we analyze the problem in detail we will realize that we need to represent the borders of the strip of invisibility. It can be done by the MASKTOP and MASKBOTTOM functions. The real representation of the MASKTOP and MASKBOTTOM functions we will omit temporarily. Now the problem of drawing curves with respect to the visibility becomes simple, see algorithm 1., because we will draw the next function slice only if and only if the curve points are outside of the strip of invisibility.

The visibility problem has been solved by Watkins [6] by introducing mask vectors for the representation of the MASKTOP and MASKBOTTOM bounds. Several problems had to be solved because all computation was done in the floating point representation:

- the first problem is how to decide if we have set up MASK[i] or MASK[i+1] if the coordinate x is between values i and i+1 e.g. $i < x < i+1$
- the second problem is that the MASKTOP and MASKBOTTOM arrays have to be set up for all point of the curve. That means that an interpolation procedure has to be employed, with some suitable interpolation step length.
- the third problem is that special case has to be solved: when the curve is parallel with the z axis, the usual line segment slope computation can fail.



Algorithm 1.

3. Proposed method

In [6] the functions MASKTOP and MASKBOTTOM are represented by vectors with values in floating point representation. We can imagine the whole proces of hidden-line drawing as follows in figure 2.

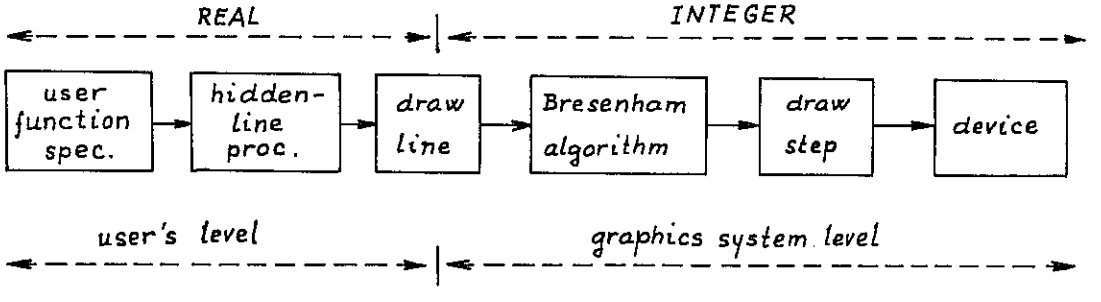


Figure 2.

Now we can ask ourselves if there is any possibility of increasing the efficiency of the hidden-line solution. One possibility is

to combine the complete Watkin s algorithm with the Bresenham algorithm directly at the physical level. Because we are dealing with the raster devices at the physical level we have got rid of all these above mentioned problems.

The solution of the hidden-line problem is now relatively very simple, becausy we have to change only the procedure DRAW-STEP, that generates code for the physical movement, in order to take account of the strip of invisibility. Because DRAW-STEP draws only one step we have to check only if the next end-point in the raster is inside of the strip of invisibility or not. The structure of the proposed method is shown on figure 3.

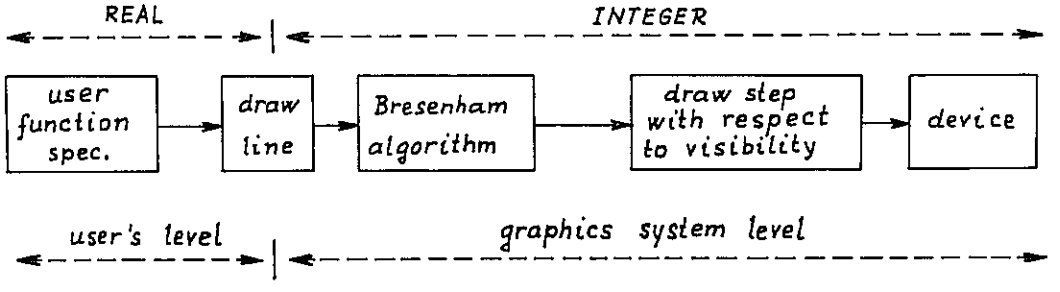


Figure 3.

It is obvious that we need only integer representation for the MASKTOP and MASKBOTTOM masking arrays. The simplified solution is shown by the algorithm 2.

It was find out that the lines (on the physical level) which are parallel to y axis cause some problems with setting of the masks arrays, see figure 4. Suppose that we have defined the strip of invisibility and we want to draw the line segment x_1x_2. The problem is that if we want to draw the segment between the points 1 and 2 we have to change the strip of invisibility so the future points 3 and 4 become inner points in the strip of invisibility; but that is not true. Therefore in the complete algorithm the content of the mask´s arrays is changed only if dx<>0. The whole algorithm can be found in [4], where the clipping is realized too.

Watkin´s original method and proposed solution have one common problem, that has not been published yet. Because of rotation sometimes the foreground and background can be altered and the order in which the curves are drawn cause a violation of the masking premises. The second problem is how to select the scales for scaling in order not to lose any part of the picture and use the full screen area. The first problem seems to be more complicated and it is more fundamental. The proposed solution is presented bellow. The second

problem can be solved easily by finding maximal and minimal values for the screen coordinates.

```
{ GLOBAL VARIABLES }
VAR x0,y0: REAL;
    masktop,maskbottom: ARRAY [0..1024] OF INTEGER;
PROCEDURE draw ( dx,dy: INTEGER );
VAR flag5: BOOLEAN;
BEGIN x0:=x0+dx; y0:=y0+dy;
      flag5:=FALSE;
      IF masktop[x0] <= y0 THEN
      BEGIN flag5:=TRUE; masktop[x0] :=y0; END;
      IF maskbottom[x0] >= y0 THEN
      BEGIN flag5:=TRUE; maskbottom[x0] :=y0; END;
      IF flag5 THEN physline(dx,dy)
               ELSE physmove(dx,dy)
END;

PROCEDURE bresenham (u,v: INTEGER );
VAR j,d,a,b: INTEGER;
BEGIN a:=v+v; d:=a-u; b:=a-u-u;
      FOR j:=1 TO u DO
          IF d<0 THEN BEGIN draw(1,0); d:=d+a; END
                 ELSE BEGIN draw(1,1); d:=d+b; END
END;
```

Algorithm 2.

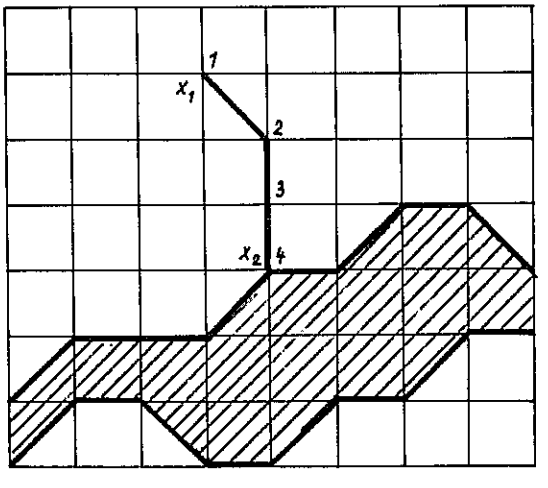


Figure 4.

4.Design of the drawing order

If we rotate the function or have a look at the function from different points, we have to keep the basic rule of the drawing. We have to draw at first the function slices that are nearer to us. Watkins [6] pointed out this problem, but the problem solution has not been published yet and many users have real difficulties to ensure that. Therefore when the function is rotated many pictures are drawn wrong. Let us try to find the solution.

Assume that the points:

$$X_1=(ax,ay,0) \qquad X_3=(bx,by,0)$$
$$X_2=(bx,ay,0) \qquad X_4=(ax,by,0)$$

are the corner-points of the grid in the x-y plane. We want to know the order of the drawing of the drawing slices.

Assumes that the points:

$$X_1' = T(X_1) \qquad X_3' = T(X_3)$$
$$X_2' = T(X_2) \qquad X_4' = T(X_4)$$

are the corner points of the grid after the rotation transformation. Now we have to pick up two margines from which we will start to draw the picture. We have to select the end-points of these margines that has the smallest z' coordinate. We will mark that point by the index r. In general there are two basic possibilities that are shown on figure 5.

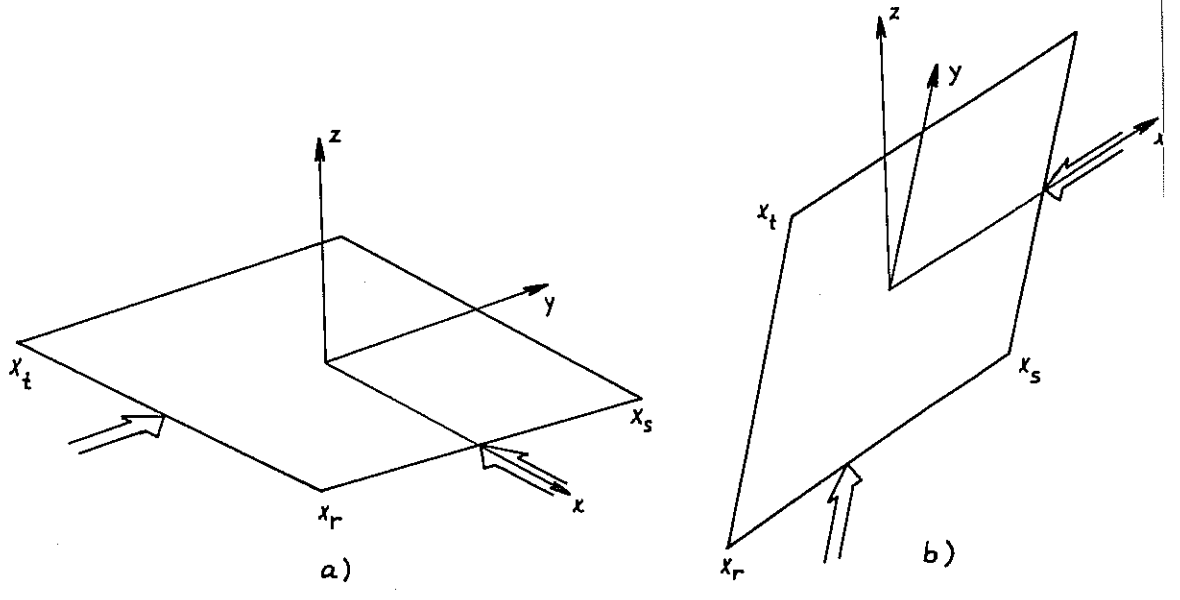


Figure 5.

The direction in which the slices are to be drawn are marked by ⇐.

In the case ad a) we can see that the margins from which we will start to draw line segments belongs to the end-points $X_r X_s$ and $X_t X_r$. In the case ad b) we can see that we will fail. Therefore we have to test if

$$x_t' <= x_r' <= x_s' \quad \text{or} \quad x_s' <= x_r' <= x_t'$$

If the boolean expression has value false then we have to find the second point which has minimal z' coordinate and which is different from the original point. The new point will be remarked by the index r.

The whole procedure can be described by the algorithm 3.

1. Find the index $r \in \langle 1,4 \rangle$ so that
   $$z_r' = \min \{ z_i' \} \quad i=1,\ldots,4$$
2. Find the indices of neighbours and mark them by indices t,s
3. If condition
   $$x_t' <= x_r' <= x_s' \quad \text{OR} \quad x_s' <= x_r' <= x_t'$$
   has value FALSE then
   begin
   Find index $u \in \langle 1,4 \rangle$ so that
   $$z_u' = \min \{ z_i' \} \quad i=1,\ldots,4 \text{ and } i \neq r$$
   r := u
   Find the indices of neighbours and mark them by indices t,s
   end

Algorithm 3.

Now we can draw the function by drawing the slices according to the selected margins, which are defined by line segments with the end-points $X_r X_t$ and $X_r X_s$.

But if we draw a function whose behaviour is wild enough then we receive a picture which is wrong, see figure 6 It seems to be more convenient in this situation to apply the Zig-zag method 1 and we will then obtain the correct results, see figure 7

The Zig-zag method can be described by:

1. Initialize the mask's arrays
2. Draw the margins that are defined by the end-points $X_r' X_s'$ and $X_r' X_t'$ (steps 1,2)
3. Draw the function values according to the grid and according to the directions on figure 8 (steps 3-12)

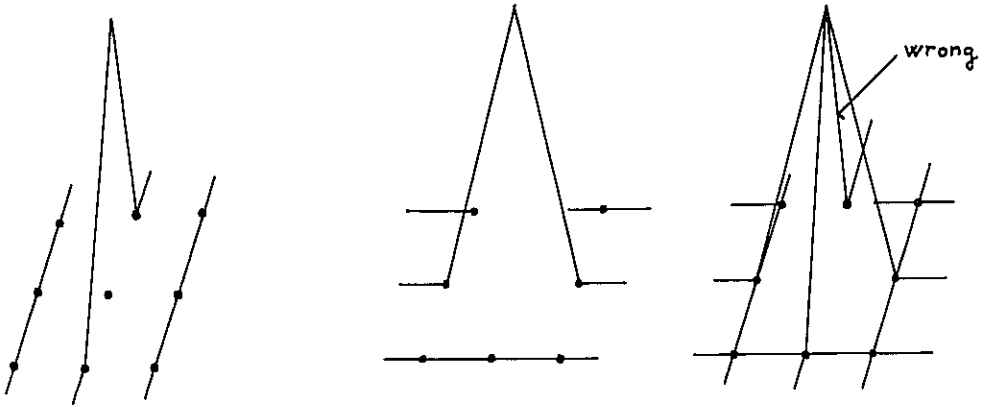


Figure 6.

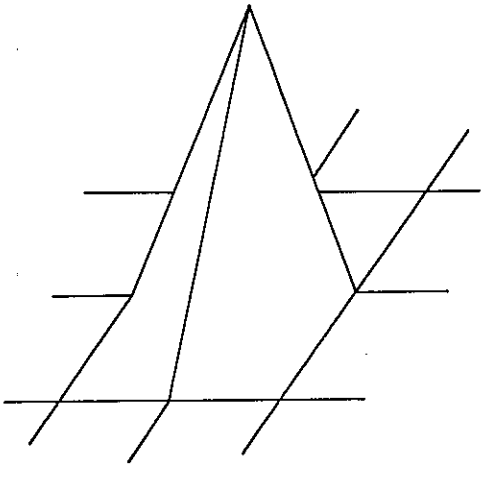


Figure 7.

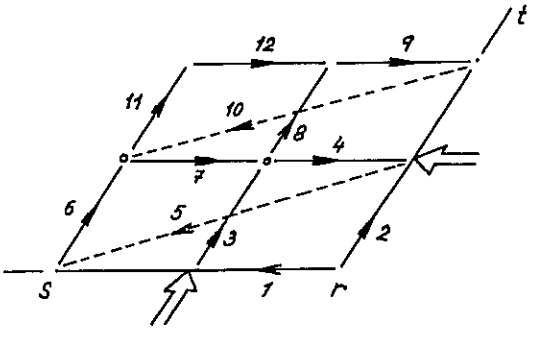


Figure 8.

Let us suppose that the function is given by the values

$$f[i,j] \qquad i=1,\ldots,n \quad \text{and} \quad j=1,\ldots,m$$

in the grid - points those coordinates are given by values

$$x[j] \qquad j=1,\ldots,m$$

$$y[i] \qquad i=1,\ldots,n$$

Then after transformation (rotation, translation) we receive values

$$x'[j] \;,\; y'[i] \;,\; f'[i,j] \quad i=1,\ldots,n \text{ and } j=1,\ldots,m$$

Now the whole process of drawing can be made in the integer representation without using a floating point processor.

5.Conclusion

The algorithm presented for drawing functions of two variables with respect to visibility is intended for the use with microcomputers. Because the basic algorithm for visibility respectation can be realized by about ten assembly instructions it seems to be convenient to build it directly into the algorithm for a drawing straight lines. Now we can see that the basic graphics menu can be extended by the operations:

- initialize mask´s arrays
- draw line with respect to the visibility,

that means, that the intelligence of graphic devices can be easily and significantly improved by adding several assembly instructions into the algorithm for the drawing lines. If the graphics display with grey scale is used the algorithm can be easily improved by using algorithm [3] for drawing straight lines.

We can ask ourselves whether primitive-level instruction for drawing lines which takes account of visibility, should be a part of any basic graphics software system, e.g. GKS.

6.Acknowledgement

I would like to thank Prof. L.M.V. Pitteway and Dr.J.P.A. Race for their many helpful discussions and suggestions that enabled me to finish this project succesfully.

7.Literature

[1] Boutland J.: Surface Drawing Made Simple, Computer Aided Design 11(1) January 1979, pp.19-22

[2] Bresenham J.E.: Algorithm for Computer Control of Digital Plotter, IBM Syst. J. 4(1) 1965, pp.25-30

[3] Pitteway M.,Watkinson D.: Bresenham´s Algorithm with Grey Scale, Comm. of ACM 23(11), November 1980, pp.625-626

[4] Skala V.: Hidden-Line Processor, CSTR/29, Computer Science Dept., Brunel University, Uxbridge, Middlesex, 1984

[5] Sowerbutts W.T.: A Surface-Plotting Program Suitable for Microcomputers, Computer Aided Design 15(6), November 1983, pp.324-327

[6] Watkins S.L.: Masked Three-Dimensional Plot Program with rotation, Comm. of ACM 17(9), September 1974, pp.520-523

[7] Williamson H.: Hidden-Line Plotting Program, Comm. of ACM 15(2), February 1972, pp.100-103

YU ISSN 0005-1144
ATKAAF 31 (1—2), 25—30 (1990)

Václav Skala

General Conics Clipping – Problem Solution

UDK 007.001.33
IFAC IA 1.2;2.8

Original scientific paper

New algorithms for 2D quadratic arcs clipping against convex and non-convex windows are presented. Algorithms do not use parametric equations for the arcs description. The only a square root function is needed. The algorithms require information how edges of the given window are oriented. The design, implementation and verification is the first step toward to the quadratic arcs usage in computer graphics as new basic primitives. The presented algorithms have not been published in a literature known to the author.

Key words: Clipping, Quadratic Arcs, Algorithms

1. INTRODUCTION

Clipping is a very important part of all graphics packages. There are many efficient algorithms as [1], [6], [7], [10] for clipping lines against convex windows or for clipping lines against a window with holes and with the non-linear boundaries, see [13], [14]. Unfortunately no one known algorithm deals with a problem how to clip circles or ellipses against a window. This problem is a fundamental one if we want to introduce quadratic arcs as a basic primitive for 2D graphics packages. The below described algorithms enable to clip general quadratic curves or arcs against convex or non-convex window.

2. CLIPPING BY A CONVEX AREA

Provided a convex area is given by its vertices in the clockwise order and a oriented circle given by its center $\mathbf{x}_w$ and radius r. We want to find those parts of the given circle which lie inside of the given convex window. For easier understanding the following notation will be used:

$\mathbf{x}_w$ circle center — $\mathbf{x}_i$ polygon vertex

$\mathbf{x}_k$ instead of $\mathbf{x}_{i+1}$; the $+$ is meant as modulo n addition

$\mathbf{s}_i = \mathbf{x}_i - \mathbf{x}_{i-1}$ — $\mathbf{s}_k = \mathbf{x}_k - \mathbf{x}_i$

${}^i\mathbf{s}_w = \mathbf{x}_w - \mathbf{x}_i$

$\mathbf{t} = [y_w - y, x - x_w]^T$ tangent vector at the point (x, y)

To solve this problem it is necessary to find intersection points of the circle and line $w(q)$ on which the window border lies, i. e. to solve the following equations:

$$\mathbf{x}(q) = \mathbf{x}_i + (\mathbf{x}_{i+1} - \mathbf{x}_i) \cdot q$$

$$(x - x_w)^2 + (y - y_w)^2 - r^2 = 0$$

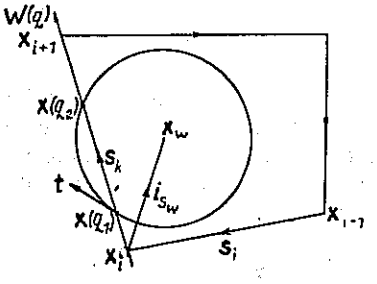


Figure 2.1.

Solving these equations with regard to the variable q the quadratic equation

$$a\,q^2 + b\,q + c = 0$$

will be obtained, where:

$$a = |\mathbf{s}_k|^2 \quad b = -2\,{}^i\mathbf{s}_w^T \cdot \mathbf{s}_k \quad c = |{}^i\mathbf{s}_w|^2 - r^2$$

In the case that the line $w(q)$ intersects or touches the given circle two solutions are generally obtained that are not necessarily different:

$$q_{1,2} = \frac{-b \pm \sqrt{b^2 - 4\,ac}}{2\,a}$$

The obtained values q_1, q_2 must be ordered so that $q_1 \leq q_2$. For the further processing only points for $q \in \langle 0, 1)$ will be considered. Of course some possible situations must be distinguished, see fig. 2.2. For a general case, when $q_1 \neq q_2$, the tangent vector $\mathbf{t}_1$ always points out of the given window while the tangent vector $\mathbf{t}_2$ always points into the given window, see fig. 2.2. cases *a* and *d*. Therefore the circular arc $\mathbf{x}(q_1)\,\mathbf{x}(q_2)$ cannot be considered for further processing. The cases *b* and *c* from fig. 2.2. are a little bit more complicated because the tangent vectors $\mathbf{t}_1$ and $\mathbf{t}_2$ are equal. It means that it is necessary to introduce some special attri-

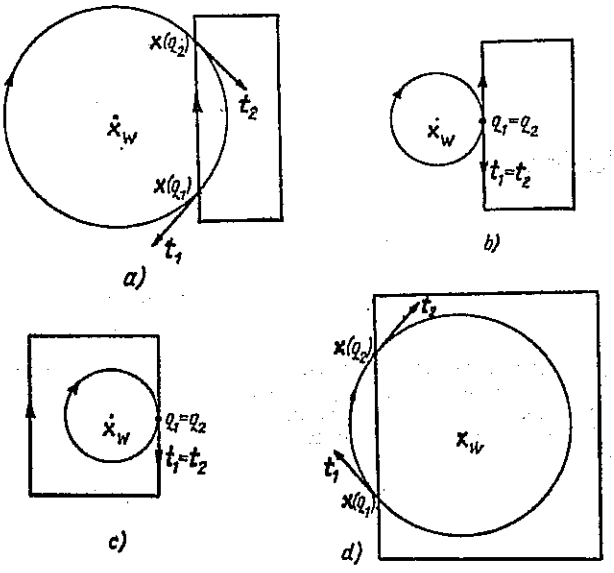


Figure 2.2.

butes for the case *b* while case *c* must be handled as no intersection points with the given edge have been found. In the case *b* only one point $\mathbf{x}(q_1)$ will be drawn. But there are still some special cases to be solved, see fig. 2.3.

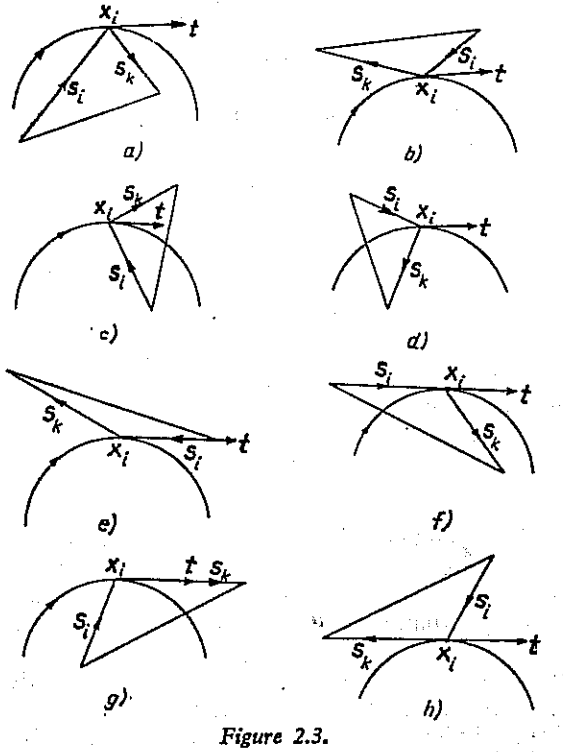


Figure 2.3.

It can be observed that it is necessary to introduce the additional attribute that would determine exactly whether the tangent vector **t** points into or out from the clipping window.

First of all it is necessary to distinguish between cases *a*, *b* and *c*, *d* in fig. 2.3., i. e. to distinguish between »touch« and »pass« types. Now let us examine the result of the cross product of the $\mathbf{s}_i$, $\mathbf{s}_k$ and **t** vectors, see fig. 2.3., in order to distinguish some cases, see table 2.1.

Table 2.1.

$[\mathbf{t} \times \mathbf{s}_i]_z$	$[\mathbf{t} \times \mathbf{s}_k]_z$	case	type	sequence
> 0	< 0	a	touch	$\mathbf{x}_i\ \mathbf{x}_i$ $+-$
< 0	> 0	b	touch	$\mathbf{x}_i\ \mathbf{x}_i$ $+-$
> 0	> 0	c	pass	$\mathbf{x}_i$ $+$
< 0	< 0	d	pass	$\mathbf{x}_i$ $-$
$= 0$	> 0	e	touch	$\mathbf{x}_i\ \mathbf{x}_i$ $+-$
$= 0$	< 0	f	pass	$\mathbf{x}_i$ $+$
> 0	$= 0$	g	pass	$\mathbf{x}_i$ $-$
< 0	$= 0$	h	touch	$\mathbf{x}_i\ \mathbf{x}_i$ $+-$
$= 0$	$= 0$	special cases	touch	not allowed

where $+-$ denotes the sign of the cross product z coordinate

It is obvious that when the both cross products are equal to zero it is necessary to distinguish some very special cases. Therefore it is necessary to use the direction of the tangent vector for determining the attribute.

The whole algorithm can be described by a sequence in PASCAL-type style, see algorithm 2.1.

```
k: = 0; i: = n − 1;
flag: = true; {circle is not intersected}
while k < n do
begin
  s_k: = x_k − x_i; {needed for intersection solution}
  if COMPUTE VALUES (q_1, q_2)
  then {intersection points exist and q_1 ≤ q_2}
  begin flag: = false;
    if q_1 = 0 then {pass / touch type}
    begin
      s_i: = x_i − x_{i−1}; t: = [y_w − y_i, x_i − x_w]^T;
      a: = [t × s_i]_z;  b: = [t × s_k]_z;
      if (a > 0) or (b ≥ 0) then GENERATE (x_i,
        '+');
      if (a ≤ 0) or (b < 0) then GENERATE (x_i,
        '−');
      if q_2 ∈ (0, 1) then
        begin t: = [y_w − y(q_2), x(q_2) − x_w]^T;
          GENERATE (x(q_2), sign ([t × s_k]_z))
          {for circle, ellipse the attribute is al-
          ways '+'}
        end
    end
```

```
    else
      if q1 ≠ q2 then
      begin
        for j: = 1 to 2 do
          if qj ∈ (0, 1) then
          begin t: = [yw − y(qj), x(qj) − xw]^T;
              GENERATE (x(qj), sign ([t × sk]z))
          end
      end
      else
        {if q1 = q2 then}
        begin t: = [yw − y(q1), x(q1) − xw]^T;
          if t · sk < 0 then
          begin
           GENERATE (x(q1), '+');
            GENERATE (x(q1), '−')
          end
        end;
  i: = k; k: = k + 1
end {while};
```

Algorithm 2.1.

As a result of the algorithm 2.1. sequences shown in table 2.1. are obtained. Now it is necessary to draw appropriate arcs that are inside of the given window. This process can be described by the algorithm 2.2.

```
m: = No of intersections;
if m ≠ 0 then
begin i: = 2;
  while i < m − 1 do
  begin if xi = xi+1 then PLOT (xi)
      else if attr (xi) = '+' then DRAW ARC (xi,
                                xi+1, r)
                                else DRAW ARC (xi−1
                                xi, r);
      i: = i + 2
  end
end {while}
else {it is necessary to distinguish cases when the}
    {circle is totally inside or outside of the given
    window}
  if flag then
  begin flag: = true;
      i: = n − 1; k: = 0;
      while (k < n) and flag do
      begin sk: = xk − xi; ¹sw: = xw − xi;
          if [sk × ¹sw]z > 0 then flag: = false;
          i: = k; k: = k + 1
      end;
      if flag then DRAW CIRCLE (xw, r)
  end
```

Algorithm 2.2.

The shown algorithm deals with a principal solution and does not particularly care of the arithmetic precision, see [8]. In some cases special criteria must be used in order to respect a limited precision.

The presented algorithm is capable to handle all kinds of quadratic arcs. In the general case the quadratic arc must be given as:

$$f(\mathbf{x}) = \mathbf{x}^T \mathbf{A}\, \mathbf{x} = 0$$

so that

$$f(\mathbf{x}_w) = \mathbf{x}_w^T \mathbf{A}\, \mathbf{x}_w < 0$$

where $\mathbf{x}_w$ is the center. The tangent vector $\mathbf{t}$ must be computed as:

$$\mathbf{t} = [f_y, -f_x]^T$$

where f_x, f_y are partial derivations of the function f and the matrix $\mathbf{A}$ represents a general quadratic curve.

3. NON-CONVEX WINDOW CLIPPING

So far presented algorithms have solved the quadratic arc clipping by the convex polygon. But some types of applications do require clipping by non-convex window. Of course, it is possible to split the given non-convex polygon into a set of convex polygons, e. g. [11].

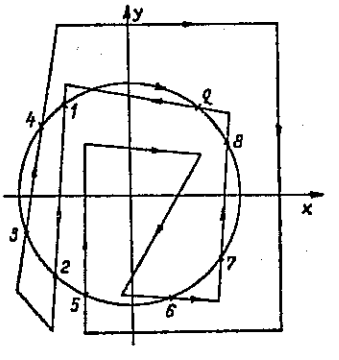


Figure 3.1.

Provided a non-convex polygon is given by its vertices in the clockwise order. It is also assumed that all vertices have different coordinates, that two edges might have only a point as a common point and that no one vertex lies on an edge. Contrary to the previous problem, an arc can intersect the polygon edges in a quite different order than would be previously expected, see fig. 3.1. Therefore some additional operations must be expected. A sequence of points together with their attributes

$$\underset{-}{\mathbf{x}_1}, \ldots, \underset{+}{\mathbf{x}_{10}}$$

is not the sequence that we need for the further processing, because only the following arcs should

be drawn:

$$\begin{matrix}\mathbf{x}_4\mathbf{x}_1 & \mathbf{x}_{10}\mathbf{x}_9 & \mathbf{x}_8\mathbf{x}_7 & \mathbf{x}_6\mathbf{x}_5 & \mathbf{x}_2\mathbf{x}_3\\ + - & + - & + - & + - & + -\end{matrix}$$

It is obvious that some kind of sort process must be employed in order to get the shown sequence. It is necessary to find a convenient criterion for sorting. The obtained points must be split into two sets according to y value of the given points, i. e.:

$$\Omega_1 = \{\mathbf{x}_j\} \quad y_j \geq y_w \quad \text{for all } j$$

$$\Omega_2 = \{\mathbf{x}_j\} \quad y_j < y_w \quad \text{for all } j$$

In the case shown in fig. 3.1. two sets Ω_1 and Ω_2 are obtained so that:

$$\Omega_1 = \underset{- + + - +}{\{\mathbf{x}_1, \mathbf{x}_4, \mathbf{x}_8, \mathbf{x}_9, \mathbf{x}_{10}\}}$$

$$\Omega_2 = \underset{+ - - + -}{\{\mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_5, \mathbf{x}_6, \mathbf{x}_7\}}$$

Both sets Ω_1 and Ω_2 must be sorted according to x value of the given points. The set Ω_2 must be sorted according to the descending x value of the given points. Then the ordered sets are:

$$\Omega_1 = \underset{+ - + - +}{\{\mathbf{x}_4, \mathbf{x}_1, \mathbf{x}_{10}, \mathbf{x}_9, \mathbf{x}_8\}}$$

$$\Omega_2 = \underset{- + - + -}{\{\mathbf{x}_7, \mathbf{x}_6, \mathbf{x}_5, \mathbf{x}_2, \mathbf{x}_3\}}$$

If a new set Ω is created as:

$$\Omega = \Omega_1 \ \mathbf{cont}\ \Omega_2$$

where **cont** is the concatenation operator, i. e.:

$$\Omega = \underset{+ - + - + - + - + -}{\{\mathbf{x}_4, \mathbf{x}_1, \mathbf{x}_{10}, \mathbf{x}_9, \mathbf{x}_8, \mathbf{x}_7, \mathbf{x}_6, \mathbf{x}_5, \mathbf{x}_2, \mathbf{x}_3\}}$$

then the required parts of the given quadratic curves are obtained.

If ellipses are considered the situation is a little bit more complicated. The above shown criterion for splitting the intersection points into two sets Ω_1, Ω_2 is not the right one, see fig. 3.2., because the arc for $y \geq y_w$ is not generally a function of x coordinate It means that two points $\mathbf{x}_a$, $\mathbf{x}_b$ must be found so that the tangent vectors of the given ellipse are collinear with y axis in these points. The points can be found as a solution of the quadratic equations:

$$\mathbf{x}^T \mathbf{A}\, \mathbf{x} = 0 \text{ and } \frac{\partial}{\partial y} \mathbf{x}^T \mathbf{A}\, \mathbf{x} = 0$$

Because the $\mathbf{x}_a$, $\mathbf{x}_b$ points are obtained it is possible to split the intersection points of the ellipse and the given window into two sets Ω_1 and Ω_2 so that:

$$\Omega_1 = \{\mathbf{x}_j\} \quad [(\mathbf{x}_b - \mathbf{x}_a) \times (\mathbf{x}_j - \mathbf{x}_a)]_z \geq 0 \quad \text{for all } j$$

$$\Omega_2 = \{\mathbf{x}_j\} \quad [(\mathbf{x}_b - \mathbf{x}_a) \times (\mathbf{x}_j - \mathbf{x}_a)]_z < 0 \quad \text{for all } j$$

The sets Ω_1 and Ω_2 must be sorted again and a new set Ω must be created in the same way.

If a parabolic arc is considered the rules can be derived in a similar way. But because only $\mathbf{x}_a$ point can be obtained it is necessary to find a different criterion how to split the obtained points. The points $\mathbf{x}_v$, $\mathbf{x}_f$, $\mathbf{x}_a$ can be determined easily for the parabolic arc. Therefore the Ω_1 and Ω_2 sets can be defined as follows:

$$\Omega_1 = \{\mathbf{x}_j\} \quad [(\mathbf{x}_f - \mathbf{x}_v) \times (\mathbf{x}_j - \mathbf{x}_a)]_z \geq 0 \quad \text{for all } j$$

$$\Omega_2 = \{\mathbf{x}_j\} \quad [(\mathbf{x}_f - \mathbf{x}_v) \times (\mathbf{x}_j - \mathbf{x}_a)]_z < 0 \quad \text{for all } j$$

If a hyperbolic arc is considered the main problem is to find a convenient criterion for splitting the intersection points into two sets Ω_1 and Ω_2 and the proper criterion for the ordering these sets. In this case it is necessary to find a vector $\mathbf{t}_0$ as:

$$\mathbf{t}_0 = \mathbf{x}_{f2} - \mathbf{x}_{f1}$$

and a vector $\mathbf{t}$ orthogonal to the vector $\mathbf{t}_0$ as:

$$\mathbf{t} = [t_y, -t_x]^T$$

Now the sets Ω_1 and Ω_2 can be defined as:

$$\Omega_1 = \{\mathbf{x}_j\} \quad [\mathbf{t} \times (\mathbf{x}_j - \mathbf{x}_w)]_z \geq 0 \quad \text{for all } j$$

$$\Omega_2 = \{\mathbf{x}_j\} \quad [\mathbf{t} \times (\mathbf{x}_j - \mathbf{x}_w)]_z < 0 \quad \text{for all } j$$

The sets Ω_1 and Ω_2 must be ordered and the ordering according to x coordinate of the given points is not the right one.

The sets Ω_1 and Ω_2 must be reordered with regard to x or y coordinate according to the rotation angle α defined as:

$$2\alpha = \operatorname{arccotg}(\xi)$$

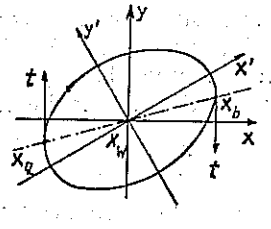


Figure 3.2.

where $\xi = \frac{a_{11} - a_{22}}{2\,a_{12}}$ a_{ij} are elements of the matrix **A**.
It means that if

$\xi \geq 0$ then the y coordinate must be used for sorting

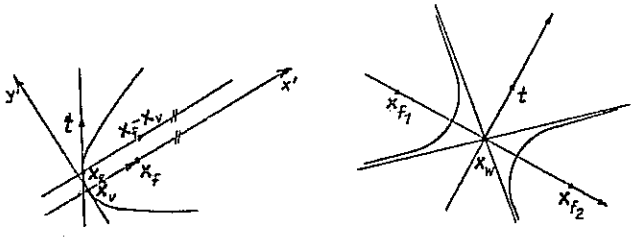
$\xi < 0$ then the x coordinate must be used for sorting

To display the obtained arc segments a similar algorithm to the algorithm 2.2. can be used.

For the non-convex window clipping a special test »point in polygon« must be employed if no intersection point is found in order to distinguish between cases when ellipse or circle lie totally inside or outside of the given window (in the other cases the whole curve must intersect window boundary if they are inside of the window).

4. CONCLUSION

The new algorithms for clipping quadratic curves and their parts against convex and non-convex window have been presented. The algoritmhs can be modified in order to handle different polygon or curves orientations and more general cases, too. The algorithms use only the implicit function definition for quadratic curves and only a square root function is needed. The algorithms do not solve the problem of the limited arithmetic precision because this problem was solved by [8] and the shown results can be applied straightforwardly with the presented algorithms.



5. ACKNOWLEDGMENT

The author would like to express his thanks to Mr. K. Karlovec for the help, interest and comments, to the students of Computer Graphics course that stimulated this work, to the members of Computer Graphics and CAD System Group for their invaluable comments, help and understanding, to Teplotechna Comp. (Czechoslovakia) for the support that enabled to finish this work successfully.

6. REFERENCES

[1] Cyrus, M., Beck, J.: **Generalized Two- and Three Dimensional Clipping.** Computers & Graphics, Vol. 3, No. 1, pp. 23—28, 1979
[2] Earnshaw, R. A., Wyvill, B. (Ed.): **New Advances in Computer Graphics.** Proceedings Computer Graphics International '89, Leeds, Springer-Verlag, 1989
[3] Foley, J. D., van Dam A.: **Fundamentals of Interactive Computer Graphics.** Addison Wesley, Reading, Mass., 1982
[4] Hansmann, W., Hopgood, F. R. A., Strasser, W. (Ed.): **Conference Proceedings EUROGRAPHICS '90.** Hamburg 1989, North Holland, 1989
[5] Kilgour, A. C.: **Unifying Vector and Polygon Algorithms for Scan Conversion and Clipping.** Report CSC 87/R7, Computer Sci. Dept., Univ. of Glasgow, 1987
[6] Liang, Y. D., Barsky, B. A.: **An Analysis and Algorithms for Polygon Clipping.** CACM 26, No. 11, pp. 868—876, 1984
[7] Liang, Y. D., Barsky, B. A.: **A New Concept and Method for Line Clipping.** ACM Trans. on Graphics, Vol. 3, No. 1, pp. 1—22, 1984
[8] Middleditch, A. E., Stacay, T. W., Tor, S. B.: **Intersection Algorithms for Lines and Circles.** ACM Trans. on Graphics, Vol. 8, No. 1, pp. 25—40, 1989
[9] Newman, W. M., Sproull, R. F.: **Principles of Interactive Computer Graphics.** 2nd ed., McGraw Hill, New York, 1979
[10] Nicholl, T. M., Lee, D. T., Nicoll, R. A.: **An Efficient New Algorthm for 2D Line Clipping: Its Development and analysis.** ACM Computer Graphics, Vol. 21, No. 4, pp. 253—262, 1987
[11] Rogers, D. F.: **Procedural Elements for Computer Graphics.** pp. 151—152, McGraw Hill, 1985
[12] Skala, V. (1989) **Algorithms for 2D Line Clipping.** in [2], pp. 121—128
[13] Skala, V.: **Algorithms for 2D Line Clipping.** in [4], pp. 355—366, 1989
[14] Van Vyk C. J.: **Clipping to the Boundary of a Circular Arcs Polygon.** Computer Vision, Graphics and Image Processing, Vol. 25, No. 3, 1984

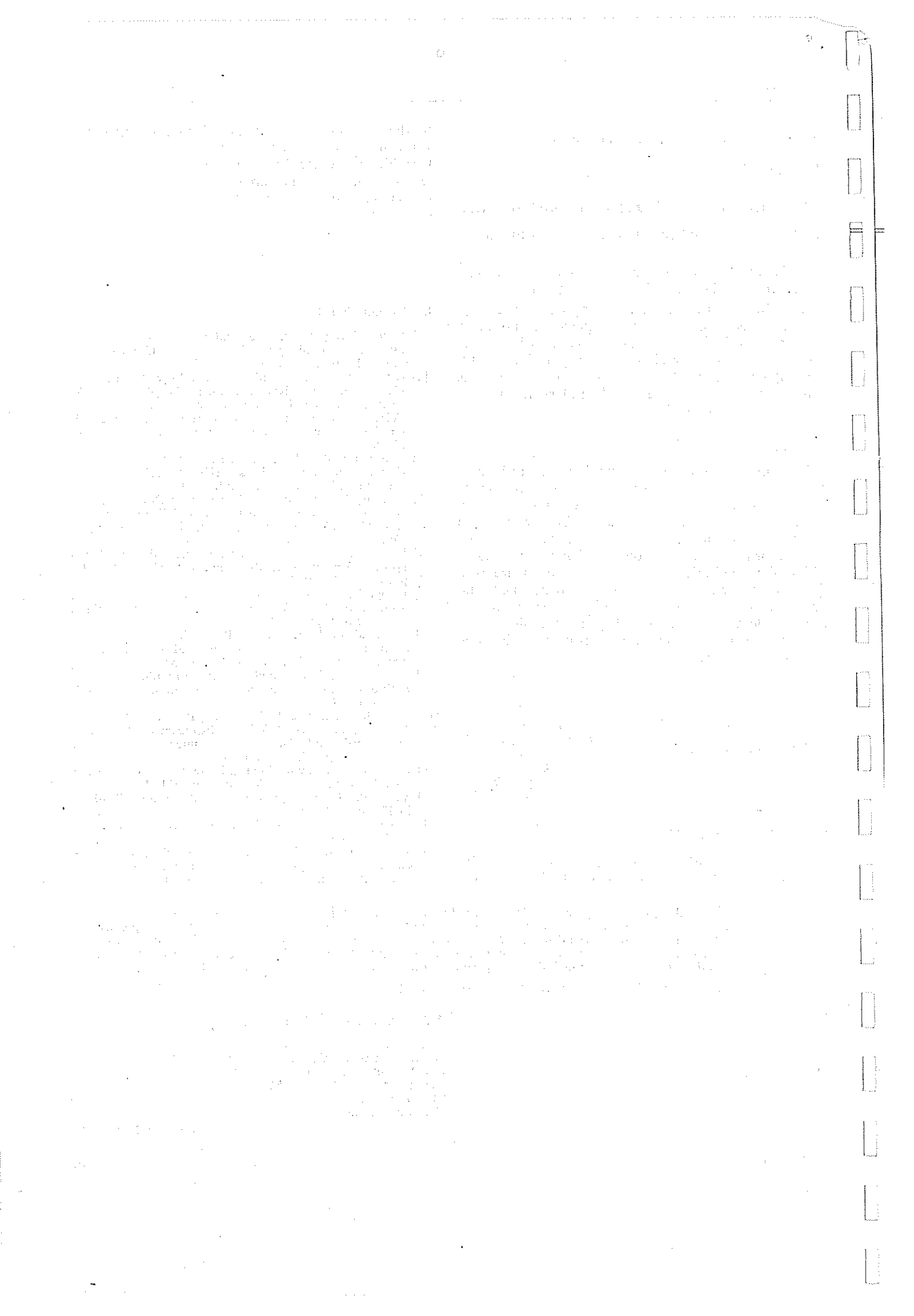
Odrezivanje stožnica — rješenje problema. Prezentiran je novi postupak odrezivanja lukova krivulja drugog reda za konveksne i nekonveksne prozore. Postupak ne koristi parametarske jednadžbe za opis lukova. Potrebna je jedino funkcija drugog korjena. Postupak zahtjeva informaciju o orjentaciji bridova datog prozora. Oblikovanje, primjena i provjera su prvi korak u korištenju lukova krivulja drugog reda kao novih osnovnih primitiva u računarskoj grafici. Pokazani postupak dosad nije bio publiciran u literaturi koliko je autoru poznato

Ključne riječi: odrezivanje, lukovi krivulja drugog reda, postupci.

AUTHOR'S ADDRESS:

Doc. Ing. Václav SKALA, CSc.,
Dept. of Informatics and Computer Science,
Institute of Technology,
Nejedlého sady 14, Box 314
306 14 Plzeň
Czechoslovakia

Received: 1990-3-20.



Kreslení vývojových diagramů počítačem

Ing. Václav Skala — Hynek Hlavnička, katedra technické kybernetiky VŠSE, Plzeň

681.3.05—5

V článku je popsán harmonogram pro kreslení vývojových diagramů počítačem na kreslicím stole DIGIGRAF 1008. Program byl realizován v jazyce ALGOL 60 na počítači ODRA 1204 a byl uvěřen několikaměsíčním používáním.

Úvod

S rozvojem výpočetní techniky se zvyšují nároky na technickou dokumentaci. Pro usnadnění práce při kreslení vývojových diagramů byl sestaven program, který nakreslí požadovaný vývojový diagram. Značky používané programem jsou převzaty z normy ČSN 36 9030 [1].

Program byl realizován v jazyce ALGOL 60 především proto, že

a) je snadno přenosný na jiný počítač, který má překladač pro jazyk ALGOL 60;

b) výhodně používá systému procedur pro ovládání kreslicího stolu DIGIGRAF 1008.

Rozložení značek

Pro jednoduchost zadávání vývojových diagramů byla definována stránka vývojového diagramu obsahující až 50 značek, které jsou umísťovány do pevného rastru. Uživatel zadává umístění bloku v rastru indexem řádku a indexem sloupce spolu s označením významu bloku. Rastr se skládá z pěti sloupců a deseti řádků. V realizovaném programu je celkem 26 nejdůležitějších značek. Soubor značek lze rozšířit tak, aby bylo možné nakreslit libovolný vývojový diagram.

Propojování značek

Nutnou podmínkou pro vytvoření spojení mezi dvěma bloky je znalost počátečního a koncového bodu spojení. Tuto informaci musí uživatel dodat algoritmu pro propojování značek. Některé dosavadní systémy pro kreslení vývojových diagramů počítačem vyžadují navíc ještě informaci o bodech, v nichž se čáry lomí, napojují se a kříží. V použitém algoritmu je nutná podmínka pro spojení dvou bodů také podmínkou postačující. Algorimus je založen na principu šíření vlnění v rovině. Jestliže vlna při šíření narazí na překážku, je nutno rozlišit dva případy:

a) je-li překážkou značka nebo hranice prostoru pro spojování, musí se tato překážka obejít.

b) je-li překážkou spojnice, je nutno rozpoznat případy:

— jde-li o spojnici vycházející ze stejného počátečního, popř. koncového bodu, nastane větvení, resp. napojování spojnic,

— jde-li o křížení, musí se provést test, zda nedojde ke splynutí spojnic.

Z principu použitého algoritmu vlnění vyplývá, že nakreslené spojnice budou nejkratší ze všech možných spojnic.

Zadávání vstupních dat

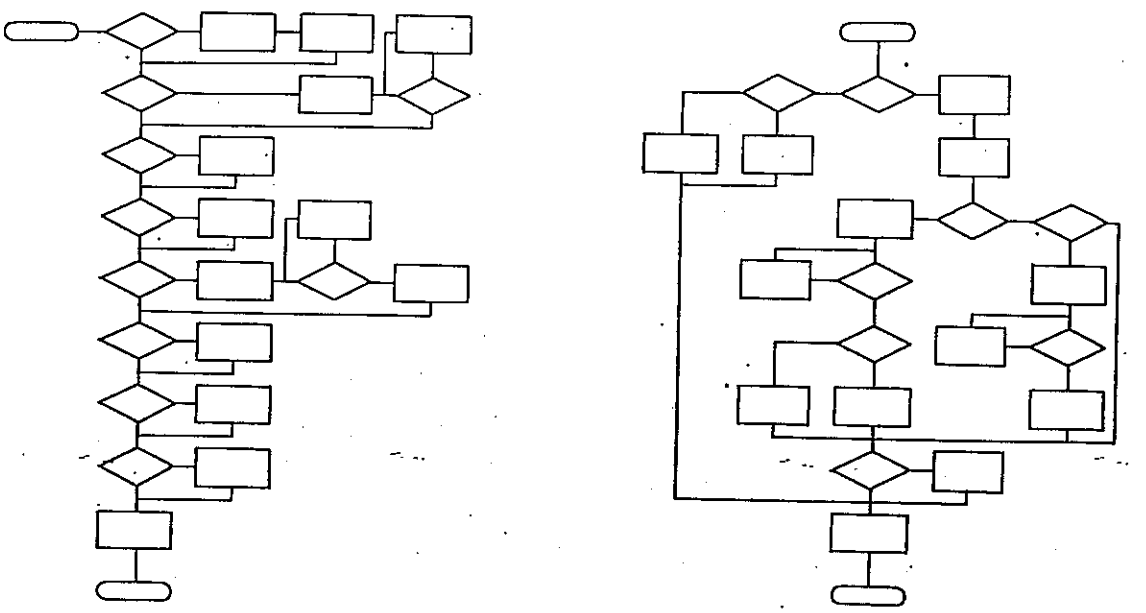
Při návrhu způsobu zadávání vstupních dat uživatelem byl respektován především požadavek rychlého zadání vstupních hodnot s minimální redundancí informace. Postup zadávání dat je tento:

a) uživatel navrhne rozložení značek v rastru;

b) očíslují se bloky a spoje pořadovými čísly pro snazší orientaci;

c) zadají se počítači data obsahující

— počet bloků,
— počet spojů,
— požadovaný formát,
— polohu bloků v rastru,
— spojnice bloků uvedením počátečního a koncového bodu spojnice.



Ukázka vývojových diagramů nakreslených počítačem.

Vstupní data jsou kontrolována testy, které odhalí tyto chyby v zadání:

a) blok má být umístěn mimo rastr,

b) požadovaná značka není v souboru realizovatelných značek,

c) více bloků je umístěno v jednom místě rastru,

d) požaduje se spojnice na neexistující blok.

Závěr

Cílem práce bylo usnadnit kreslení vývojových diagramů, a tedy i dokumentačního procesu. Použitý algoritmus nepopisuje vývojový diagram, protože se nepodařilo uspokojujícím způsobem vyřešit problémy spojené s umístěním a vepsáním požadovaného textu.

V nynější době se zabýváme možnostmi využití počítače k nakreslení vývojových diagramů přímo ze zdrojového programu napsaného v jazyce ALGOL.

Ukázka vývojových diagramů nakreslených počítačem dokládá použitelnost algoritmu.

Literatura

[1] Norma ČSN 36 9030: Značky vývojových diagramů pro zpracování informací.

[2] MÜLLER, H. J.—SMRČNA, J.—VOKURKA, J.: Vývojové diagramy. „Výběr z organizační a výpočetní techniky", 1973, č. 4 a 5, 1974, č. 1.

[3] NADRCHAL, J.: Generování vývojových diagramů pomocí počítače TESLA 200. Sborník Počítačová grafika FEL ČVUT, Praha 1974.

[4] Výzkumná zpráva 209-1-78 VŠSE, Plzeň.

Kreslení vývojových diagramů počítačem II

Ing. Václav SKÁLA — Hynek HLAVNIČKA, katedra technické kybernetiky VŠSE, Plzeň

V článku je popsán algoritmus kreslení vývojových diagramů programu napsaného v jazyce ALGOL 60 pomocí počítače na kreslicím stole DIGIGRAF 1008. Program byl realizován na počítači ODRA 1204 v jazyce ALGOL 60 a je ověřen několikaměsíčním používáním.

Úvod

V souladu s požadavky na technickou dokumentaci ve výpočetní technice je žádoucí, aby počítač kreslil vývojové diagramy přímo ze zdrojového programu. Proto byl sestaven algoritmus umožňující kreslení vývojových diagramů programů napsaných v jazyce ALGOL 60. Značky používané programem jsou převzaty z normy ČSN 36 9030 [1] kromě značky začátku a konce cyklu.

Lexikální a syntaktická analýza

Pro kreslení vývojových diagramů jsme upravili gramatiku jazyka ALGOL 60. Syntaktická analýza je prováděna metodou syntaktických procedur. Výstupem je poněkud složitější datová struktura, která obsahuje úplnou informaci o struktuře programu. V této části se zároveň provádí převod z obecně hnízdové blokové struktury, která je typická pro jazyk ALGOL 60, do lineární struktury, typické pro jazyk FORTRAN. Ošetření syntaktických chyb programu se neprovádí, neboť se předpokládá kreslení odladěných programů.

Generování tabulek pro kreslení

V této části se data získaná syntaktickou analýzou zpracovávají do tabulek, které slouží ke kreslení vývojového diagramu. Algoritmus umisťuje značky do libovolně dlouhého rastru, a ten se pak dělí na stránky. Výstupní data již obsahují informace, potřebné k nakreslení zvolených stránek vývojového diagramu.

Vývojové diagramy kreslí i algoritmus, který je popsán v [5] a který byl doplněn o automatické kreslení šipek.

Značky se rozmisťují počínaje prostředním sloupcem. Pokud se v programu vyskytuje větvení, umisťují se větve střídavě vpravo a vlevo od prostředního sloupce. Toto řešení je esteticky výhodné. Rozmisťování probíhá rekurzívním způsobem.

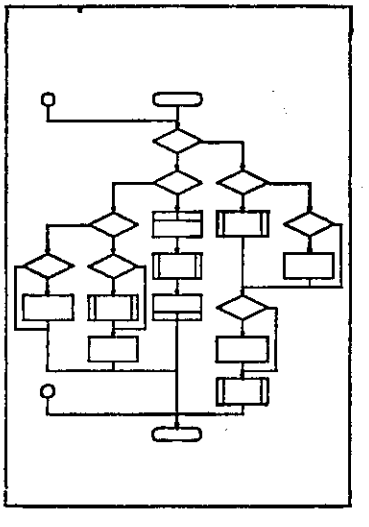
Stránkování se provádí především proto, aby bylo možno přikládat nakreslené vývojové diagramy k celkové dokumentaci programu.

Závěr

Realizovaný program podstatně usnadňuje dokumentační proces ve výpočtové technice, která nebývá obvykle důsledná. Lze konstatovat, že uvedený algoritmus lze použít pro dokumentaci programu napsaných v jiných jazycích, např. PL/I, FORTRAN a ASSEMBLER.

V nynější době se zabýváme nejen uvedenými možnostmi, ale i převodem na počítači řady TESLA, JSEP, ICL, SIEMENS.

```
BEGIN IF B THEN
L:      BEGIN IF B THEN
                BEGIN FOR L:=P DO P
                END ELSE
                IF B THEN
                BEGIN IF B THEN P;
                      L:=P
                END ELSE
                IF B THEN L:=P
        END ELSE
        BEGIN IF B THEN P ELSE
                IF B THEN L:=P;
                IF B THEN L:=P;
                P
        END;
L:
END
R
```



[1] ČSN 36 9030 Značky vývojových diagramů pro zpracování informací.
[2] MÜLLER, H. J. — SMRČINA, J. — VOKURKA, J.: Vývojové diagramy. „Výběr z organizační a výpočetní techniky", č. 4 a5 1973, č. 1/1974.
[3] NADRCHAL, J.: Generování vývojových diagramů pomocí počítače TESLA 200. Sborník Počítačová grafika FEL ČVUT, Praha 1974.
[4] Výzkumná zpráva 209-1-76 VŠSE, Plzeň.
[5] SKÁLA, V. — HLAVNIČKA, H.: Kreslení vývojových diagramů počítačem I. „Automatizace", 21, č. 3, s. 70.

Konference
Algoritmy 83, JSMaF při SAV-OSVS Bratislava, 11–15.4.1983,
Štrbské Pleso

PROGRAMOVACÍ JAZYK PL/M PRO MIKROPOČÍTAČE

Václav Skala
Katedra technické kybernetiky, VŠSE, Plzeň

1. Úvod

Programovací jazyk PL/M je vyšší programovací jazyk, který byl navržen za účelem zjednodušení systémového programování osmibitových mikropočítačů řady INTEL 8080, INTEL 8085. Jazyk je navržen tak, aby umožnil používat moderní metody strukturovaného programování, má blokovou strukturu a využívá výhod, které poskytuje práce se zásobníkem. Jazyk PL/M se vyznačuje poměrně vysokou efektivitou generovaného kódu.

2. Deklarace a datové typy

V jazyce PL/M jsou definovány dva základní typy, a to BYTE a ADDRESS. Proměnná nebo konstanta typu BYTE je osmibitová, proměnná nebo konstanta typu ADDRESS je šestnáctibitová. Příkaz pro deklaraci má např. tvar:

```
DECLARE (A,B) BYTE, D ADDRESS;
DECLARE JMENO(20) BYTE;
DECLARE ZAM(30) STRUCTURE( JMENO(20) BYTE, PRIJEM ADDRESS);
```

Příkazy zajistí, že proměnné A,B jsou typu BYTE, proměnná D je typu ADDRESS. Druhým příkazem je definována indexovaná proměnná JMENO s prvky $JMENO_1, \ldots, JMENO_{19}$. Třetí příkaz definuje 30 struktur, které se jmenují ZAM, z nichž každá má položku JMENO a PRIJEM. Je nutné zdůraznit, že nelze definovat dvourozměrné pole, což lze obejít konstrukcí se strukturou, např.:

```
DECLARE RADEK(100) STRUCTURE( SLOUPEC(100) BYTE );
```

Pak se na i-tý řádek a j-tý sloupec odvoláváme výrazem:

```
RADEK(I).SLOUPEC(J)
```

Definování struktury uvnitř struktury je nepřípustné. V PL/M je možné ještě definovat "bázované" proměnné, např. příkazy:

```
DECLARE POINTER ADDRESS;
DECLARE HODNOTA BASED POINTER BYTE;
```

Adresa proměnné HODNOTA je dána okamžitou hodnotou proměnné POINTER. Posloupnost příkazů:

```
POINTER = 375H ; HODNOTA = 25;
```

způsobí, že se na adresu 375 (hexadecimálně) uloží hodnota 25.

3. Výrazy a přiřazovací příkazy

Výraz v jazyce PL/M se skládá z operandů a operátorů. Při zápisu výrazů

v PL/M je nutné si uvědomit, že hodnoty jsou uvažovány v přímém kódu, a tedy platí: 0 - 1 = 255 v případě konstant typu BYTE. Operandem může být konstanta, proměnná nebo odkaz na adresu proměnné, tj. např. identifikátor. Pak výraz:

A + .B + .('TO_JE_PRIKLAD')

nabyde hodnoty, která je dána součtem hodnoty proměnné A, adresy proměnné B a adresy řetězce 'TO_JE_PRIKLAD'.

Přiřazovací příkaz má tvar:

proměnná = výraz; resp. proměnná := výraz; tzv. vnořené přiřazení, které lze použít kdekoliv, kde lze zapsat výraz. Příkaz:

A = A + (C := D + F);

lze vyjádřit méně efektivně příkazy:

C = D + F; A = A + C;

4. Příkazy pro řízení programu

a) Příkaz DO - END slouží jako závorky a má tvar:

```
DO;
  příkaz1;
    =
    =
  příkazn;
END;
```

S takto uzávorkovanými příkazy lze zacházet stejně jako s jedním příkazem.

b) Příkaz DO WHILE - END je příkazem cyklu a má tvar:

```
DO WHILE výraz;
  příkaz1;
    =
    =
  příkazn;
END;
```

Tělo cyklu se opakovaně provádí pokud je výraz hodnoty TRUE, tj. pravý bit výsledku výrazu je 1.

c) Iterační příkaz DO je analogií klasických příkazů cyklu a má tvar:

```
DO prom = výraz1 TO výraz2 STEP výraz3;
  příkaz1;
    =
  příkazn;
END;
```

Část STEP výraz$_3$ lze vynechat, je-li hodnota kroku 1.

d) Příkaz DO CASE - END má tvar:

```
DO CASE výraz;
  příkaz1;
    =
  příkazn;
END;
```

Po vyhodnocení výrazu, jehož hodnota K musí být z intervalu ⟨0,N-1⟩ se provede pouze příkaz$_K$. Není-li hodnota K∈⟨0,N-1⟩ není činnost příkazu definována.

e) IF příkaz má tvar:

```
IF výraz THEN příkaz1;
          ELSE příkaz2;
```

Pokud příkaz$_2$ by byl příkazem prázdným, lze část ELSE příkaz$_2$ vynechat. Z důvodů jednoznačnost nesmí být příkaz$_1$ IF příkazem.

f) Příkaz GOTO má tvar:

```
GOTO náv;
```

kde: nav je návěští, které je určeno svým výskytem v programu. Je zakázáno skákat do bloků, cyklů apod. Platí zásady stejné jako v jazyce Algol.

5. Podprogramy

Deklarace podprogramu má tvar:

```
jmeno: PROCEDURE ( par1,....,parn) typ;
       příkaz1;
         =
       příkazn;
END jmeno;
```

Procedury jsou v PL/M volány hodnotou. Každý parametr musí být deklarován. Práci podprogramu ukončí příkaz RETURN.

Procedury musí být definovány před jejich prvním vyvoláním, které má tvar:

```
      CALL jmeno ( parametry );   jde-li o proceduru bez typu,
nebo: jmeno ( parametry )         jde-li o proceduru s typem
```

Pokud potřebujeme realizovat reentrantní podprogram (může být rekurzivní), pak je nutné za typ dopsat doložku REENTRANT. Uvedme dva příklady:

```
FAKT: PROCEDURE ( N ) ADDRESS REENTRANT;
      DECLARE N BYTE;
      IF N = 1 THEN RETURN 1;
               ELSE RETURN N*FAKT(N-1);
END FAKT;
```

```
P:PROCEDURE (X,Y) ADDRESS;
  DECLARE (X,Y) ADDRESS;
  RETURN (X+Y)/2;
END P;
```

6. Zpracování přerušení

Jazyk PL/M poskytuje prostředky pro využití mechanismu přerušení k jejich zpracování. Na přerušení se pohliží jako na asynchronní vyvolání procedury, která zpracovává příslušné přerušení. Programátor má možnost napsat podprogram pro zpracování přerušení ve tvaru:

```
jmeno: PROCEDURE INTERRUPT n;
       DISABLE;..... ENABLE;
END jmeno;
```

kde: n je číslo přerušení 0 - 255, není-li v systému spec. přerušovací obvod, pak $n \in \langle 0,7 \rangle$.

7. Závěr

V předloženém příspěvku byly uvedeny základní konstrukce a vlastnosti programovacího jazyka PL/M. V jazyce jsou též příkazy vstupu a výstupu hodnot, možnosti zpracování maker v době překladu, inicializace proměnných a řadou poskytovaných funkcí, které umožňují rotaci, nastavování CARRY bitu a pod. Na základě experimentálních výsledků vychází efektivita generovaného kódu asi 1:1.2 vůči assembleru. Na základě získaných zkušeností lze jazyk PL/M považovat za vhodný k běžnému systémovému programování.

8. Literatura

[1] PL/M-80 Programming Manual, Intel Corp. 1976, Document No.98-200
[2] Skala V.PL/M -programovací jazyky pro mikropočítače,ČSVTS,VŠSE,Plzeň 1981

Konference Algoritmy '85, JSMaF při SAV - OSVS, Bratislava
Vysoké Tatry, 15-19.4.1985

KONCEPCE „COMPUTER AIDED" (CA) A ALGORITMY (AUTOMATIZOVANÉ PROJEKTOVÁNÍ INFORMAČNĚ-ŘÍDÍCÍCH SYSTÉMŮ)

Úvod do panelové diskuse

E. Kindler, MFF KU Praha
E. Ondráček, VUT Brno
J. Sedlák, INORGA Praha
V. Skala, VŠSE Plzeň

A. Možnosti (aplikační spektrum, oblasti použití) automatizovaného projektování informačně-řídících systémů (APIŘS)

Computer aided je velmi důležitý a bohatý pojem, jehož bohatství však v současné době teprve "krystalizuje", tj. vzniká, je rozvíjeno a je jen pozvolna systematicky zpracovávánо. To je sice poněkud na škodu vzájemnému dorozumění, ale zase to prozatím dovoluje větší volnost v rozvíjení a v aplikování tvůrčích principů, než kdyby bylo CA svázáno už nyní do systému. Na rozdíl od slova „automatické" pod termínem „automatizované" rozumíme činnosti člověka ve spolupráci s automatizačními prostředky. Z tohoto hlediska je třeba chápat jak současný stav a jeho tendence (obrazně odpovídající derivacím současného stavu podle času), stejně jako to, co bude dále uvedeno.

Existují oblasti, kde výpočetní technika může být velmi efektivně opravdu „aiding", pomáhající, a to tím, že ve smyčce člověk-počítač provádí velmi komplexní modelování. Jiný podnět je v hierarchiích programového vybavení pro CAD počítačů - od statického modelování stejnosměrného proudu k simulaci, od prostého modelování k heuristickým algoritmům zpracovávajícím výsledky modelování velmi netriviálním způsobem, od spojité úrovně modelování obvodů s využitím techniky řídkých matic a tuhých diferenciálních systémů přes diskrétní modelování konstantních struktur na úrovni logických funkcí, přenosu registrů a mikroprogramování, až po diskrétní modelování nekonstantních struktur na úrovni operačních systémů a počítačových sítí, a v dalších hierarchiích zaměřených na formu interakce atd. Je třeba upozornit na velmi odlišnou techniku modelování informačních a řídících systémů v případě, že výpočetní technika řídí v reálném čase, a v případě, že řízení je CA. Stejně jako modelování je APIRS v možnostech neomezeno co do aplikací.

Je třeba pamatovat na dvojí úlohu člověka - jako řešitele APIRS a jako uživatele APIRS. CA pak je třeba dělit hlavně dle druhů aplikací, např. pro: počítačové (elektronické) systémy, výrobu složitých strojních součástí, řízení technologických procesů, řízení výrobních procesů, informační systémy, řídicí systémy a programové systémy. Velmi jednoduše lze prokázat přínos CA především ve zvýšení efektivnosti a snížení podílu lidské práce při použití CA. Řada propočtů ukazuje, že náklady na hodinu provozu jednoho pracoviště CAD se rovnají zhruba 0.6 - 0.9-násobku průměrné hodiny konstruktéra. To znamená, že nasazení CAD systému je oprávněno, dosáhne-li se racionalizačního faktoru aspoň 2. Lze ukázat, že tuto hodnotu lze v různých případech vysoce překročit.

B. Speciální předpoklady dobrého využívání CA

a. Možnost interaktivního přístupu k počítači s vhodnými I/O prostředky.
b. Vhodné programovací prostředky pro styk s počítačem.
c. Algoritmická zralost daného odvětví pro realizaci programové podpory.
d. Zájem řídících pracovníků o zavedení CA.
e. Komplexní příprava kádrů a seriózní osvěta ve všech profesích.
f. Kvalitní údržba systému automatizovaného projektování.

Nutnou podmínkou při zavedení automatizovaného projektování je interakč-

ní práce člověka s počítačem. Tím je dán i požadavek na kvalitu výpočtového systému pro tento účel. Na druhé straně z toho plyne stále vyšší váha na práci člověka při využívání systému. Vždyť jde o automatizovaný systém projektování a programování. Zde je třeba si uvědomit, že to bude člověk, který bude projektovat, navrhovat různé varianty, ty ověřovat, zkoušet na zkušebních příkladech a pak vyhodnocovat a volit finální algoritmus. Snahou tedy bude spíše "počítačový" rozhovor o řešení, než předat uživateli uzavřený algoritmus řešení problému resp. úlohy. Oprávněně zde lze říci, že půjde o postupné zavádění prvků umělé inteligence do automatizovaného projektování a programování.

Účel automatizovaného projektování a programování nespočívá pouze v pracech na technickém a prováděcím projektu, nýbrž musí mít odpovídající dopad i na jeho zavedení do užívání a jeho údržbu. Životní cyklus zpravidla přežije u systému několik generací počítačů. Odtud je třeba zabezpečit, aby např. pro danou funkční, informační a organizační strukturu předmětu řízení bylo možné relativně snadno změnit příslušnou technickou a programovou strukturu např. při výměně počítačového systému nebo při volbě jiné třídy počítačů. Např. jiná programová struktura bude pro JSEP a jiná opět pro SMEP. Při implementaci automatického projektování a programování můžeme s úspěchem využít zkušeností z jiných oblastí automatizovaného návrhu.

Pro dobrou činnost CA je potřebné zabezpečit:

i/ terminálový počítačový systém s velkokapacitními pamětmi pro možnost vytváření rozsáhlé banky údajů o projektech příslušného druhu aplikace s výhledovým rozšířením (či přechodem) k expertnímu systému, který by sloužil projektantovi;

ii/ interaktivní režim práce s počítačem, grafický vstup a výstup - vedle alfanumerického - vhodný jazyk zaměřený na předmětně orientované programování: musí povolovat nejen tu strukturovanost algoritmů, kterou studuje strukturované programování, ale i tu, která je obrazem strukturovanosti mimo programování.

C. Současný stav, problémy a světové trendy CA

Od památné Dijkstrovy práce o strukturním programování byla zavedena řada technologií projektování a programování. Některé z nich se zabývají pouze informační strukturou, jiné funkční strukturou nebo logickou strukturou systému, případně kombinacemi těchto struktur. Samozřejmě, že musí směřovat ke struktuře procesní.

Využívání počítače v ČSSR pro účely automatizovaného projektování je v počátcích. Lze uvést několik technologií, které přispívají k této úloze. U nás se zatím vyjasňuje podstata problematiky a hledají se vztahy mezi některými aplikačními složkami CA - např. simulace versus CAD/CAM - což je sice ve shodě s některými trendy ve světě, ale ne všechny světové trendy jsou u nás reflektovány. Vzhledem k roztříštěnosti technik a aplikačních tříd CA a vzhledem k opatrnosti vytčené hned v prvním odstavci této stati je vhodné soustředit se především na esenciální faktor CA, odpovídající počítačovému modelování, a to jako výzkumné metodě (zkoumaný systém je nahražen svým počítačovým modelem a na něm "experimentálně" zjišťujeme vlastnosti zkoumaného systému) i jako „protéze" (část naší aktivity je nahražena výpočtem na počítači).

Další, na tom závislý faktor se týká interface mezi počítačem a uživatelem který je v CA velmi akcentován - interface zahrnuje prostředky pro popis problému ve formalizované formě (v širším smyslu) i aspekty psychologické, sociologické a aspekty aplikujících věd.

Tyto dva základní aspekty - modelování a interface - jsou materializovány v tzv. předmětně zaměřeném programování object oriented programming , což je nejaktuálnější faktor CA pro ty, kde se zabývají algoritmy.

Je též třeba zdůraznit heuristické algoritmy, jež dobře pracují, ale jejichž práci nelze dostatečně matematicky podložit: jde o algoritmy, jež často modelují první přístupy člověka ke zkoumanému materiálu a jež reflektují poměrně široké bohatství jeho psychiky. Nelze ignorovat různé oblasti CA, které si už samy vytvořily své bohatství technik (CAD/CAM, simulace apod.), a je nutné je synthetizovat, avšak ne tím, že z nich abstrahujeme jejich společné vlastnosti (pak by nám mohly vzniknout příliš chudé pojmy), ale tak, že budeme pracovat na jejich softwarové synthéze; příkladem velmi ilustrativním je vznik kombinované simulace, což je dnes jediná technika umožňující zkoumat systémy s typicky diskrétními i s typicky spojitými faktory, jež ještě navzájem interagují.

Zvláštní problémy, které je třeba překlenout, jsou tyto:

1. Nepochopení koncepce CA a její zkreslená interpretace,
2. nedostupnost již hotového a z pohledu CA zpracovaného software,
3. nedostatek zobrazovacích jednotek,
4. nedostatek velkokapacitních pamětí s přímým dostupem (magnetických diskových)
5. nedostatečná znalost životního cyklu AIS,
6. roztříštěnost přípravy systémů CA - vypracování takového systému je věcí celostátní.

Světové trendy v CA odrážejí celosvětové konference, které ve velkém rozsahu probíhají (např. Brighton 84, Berlín 84,...) a výstavy výrobců specializovaného hardware a software. Za zásadní trend lze označit systematické, důsledné a integrované využívání počítače jako pomocníka člověka v tvůrčí práci ke zvýšení jeho tvůrčích schopností.

D. Doporučená specielní literatura

1. Informations systems design methodologies: A comparative review. Proc. IFIP WG.8.1, edit. Olle, T. W., Sol, H.G., Verrijn-Stuart, A.A., North-Holland P.Co. 1982.
2. Ramsey, H.R., Atwood, M.E., Van Doren, J.R.: Flowchart versus program design languages. Comm. ACM, June 1983, Vol. 26, č.6, str. 445-449.
3. Kindler, E., Veselý, V.: Matematické modelování jako výzkumná technika. Mechanizace a automatizace administrativy, 1984, 1. část: č.7, str. 246-248, 2. část: č. 9, str. 324-327.
4. Newman, W.M., Sproul, R.F.: Principles of interactive computer graphics. Mc Graw-Hill, New York, 1973.
5. Ondráček, E.: Koncepce Computer Aided. Bulletin společnosti pro mechaniku, 1984, č. 2, str. 3-31.
6. Plander, I.: The Japanese project of the fifth-generation computer systems Computers and artificial intelligence, 1, 1982, č. 5, str. 441-456.
7. Sedlák, J., Juda, J.: Racionalizace projektování a programování pomocí PROJ a PROG. ASŘ sešit INORGA č. 92, Praha, 1983.

COMPUTER GRAPHICS EDUCATION AND RESEARCH IN FORMER CZECHOSLOVAKIA

Václav SKALA
Computer Science Department
University of West Bohemia
Americká 42, Box 314
306 14 Pilsen CZECH REPUBLIC
Tel. : +42-19-2171-212
Fax : +42-19-22-00-19
E-mail : skala@kiv.zcu.cz

INTRODUCTION

Several groups in the former Czechoslovakia have been actively working in Computer Graphics for many years behind the "Iron Curtain" and they suffered from lack of technology. There were many changes since 1989 and all groups are seeking collaboration in teaching and joint research. There are many activities in many areas in Eastern Europe, especially in the former Czechoslovakia. Last year the first Winter School of Computer Graphics, an international workshop, was held, having attracted more than 50 attendees. The workshop will be held again this academic year (see WORKSHOP section).

There are many projects in progress funded by the TEMPUS Program and other financing bodies. The following section includes representative groups but is far from complete.

DIRECTORY

Ivan JELÍNEK

Department of Computer Science
Czech Technical University
Karlovo nám 13
12 135 PRAHA 2 CZECH REPUBLIC
Tel. : +42-19-293485
Fax : +42-19-298098
E-mail : jelinek@cs.felk.cvut.cs

Courses

• CAD Systems

Structure of CAD systems, engineering databases, graphical data structures, CAD/CAM interfaces, customizing of AutoCAD, ADS, ASE, AME, engineering graphics, intelligent CAD systems, standards in CAD/CAM.

Research Activities

Logical formalization of design process in CAD systems.

Václav SKALA

(Contact as above)

Courses

• Fundamentals of Computer Graphics

Overview of computer graphics applications, fundamental algorithms, raster graphics, geometric transformations, projections, color systems, introduction to interactive computer graphics.

• Algorithms for Computer Graphics I & II

Fundamental algorithms, dual space representation, homogeneous coordinates, geometric transformations in E^2 and E^3, projections, solution of visibility problems,

color systems, computer graphics standards, rendering, scientific data visualization.

Research Activities

Fundamental Algorithms for Computer Graphics and CAD Systems, Digital Terrain Modeling Systems, Color Systems, Scientific Visualization, Medical Image Processing (CT & MRI images), Data Structures for Computer Graphics and Scientific Visualization.

Jiri SOCHOR

Department of Computer Science & Engineering
Technical University of Brno
UIVT FEI VUT
Bozetechova 2
612 66 Brno CZECH REPUBLIC
Tel. : +42-5-41321226 ext. 245
Fax : +42-5-41211141
E-mail : sochor@dcse.fee.vutbr.cz

Courses

• Introduction to Computer Graphics

Survey of fundamental algorithms for 2D graphics, introduction to image processing, principles of interactive graphics.

• Computer Graphics I

Elements and procedures of raster graphics, color models, image processing, antialiasing, polynomial curves, standards and formats.

• Computer Graphics II

Solid modeling, object and viewing transformations, sorting and searching in space, visibility, rendering methods, advanced rendering.

Research Activities

Rendering Algorithms and Architectures, Solid Modeling.

Vojtech JANKOVIC

Department of Applied Mathematics
Faculty of Mathematics and Physics
Comenius University
Mlynska dolina
842 15 Bratislava SLOVAK REPUBLIC
Tel. : +42-7-722430
E-mail : jankovic@fmph.uniba.sk

Courses

• Application of Computer Graphics in Medicine

The course provides a general overview of the most recent approaches to process, enhance and visualize pictorial data for the purposes of medical treatment. A short overview of image processing (filtering, discretization, segmentation) and pattern recognition techniques is given. The main goal is to present various classic and up-to-date techniques for the visualization of scientific data (ray tracing, volume rendering). All methods are demonstrated using real medical data of human brains obtained by CT and MR scanners.

• CAD/CAM Systems

The course provides a general overview of the state-of-the-art in the CAD/CAM area. Various software/hardware platforms and methodologies are presented with a focus on AutoCAD, CADKey and MicroStation. The main goal is to introduce an object-oriented programming approach to CAD with emphasis on graphic transfer formats and data structures. Special attention is placed on CAGD and the object-oriented structure of large software systems based on a unique geometric kernel. The ACIS modeling kernel is the most recent approach.

General Interests

Computer Graphics in Medicine :

processing and visualization of medical data sets obtained by various scanning techniques (CT, MR, ultrasound).

Object-Oriented Graphics and its application in visualization.

Physically-Based Modeling in Computer Graphics.

Modern Architectures of CAD/CAM Systems.

Particular Interests

Discrete Space :

digital topology, theoretical approach to representation and modeling of 3D objects. Transformations, matching and correction of spatial distortion among digitally represented objects.

Milos SRÁMEK

Institute of Measurement Science
Slovak Academy of Sciences
UM SAV
Dubravska cesta 9
842 19 Bratislava SLOVAK REPUBLIC
Tel. : +42-7-3782313
E-mail : miloss@umhp.savba.sk

Research Activities

Visualization of scalar volumetric data (surface and volume rendering), interactive segmentation of volumetric data, data pre-processing (anizotropic diffusion filtering), recursive ray-tracing of volumetric data, subvoxel precision surface detection, ray-tracing speed-up methods.

WORKSHOPS

The Winter School of Computer Graphics is a workshop held at the University of West Bohemia in Pilsen every academic year. The next edition will take place on January 19 and 20, 1994. Further information can be obtained by e-mail from wscg@kiv.zcu.cz.

ACKNOWLEDGEMENT

I would like to thank all the people who responded to my e-mail and sent the requested information about their groups.

3ISYCP'89
China

A Unifying Approach to the Line Clipping Problem Solution

Vaclav Skala
Computer Science and Informatics Dept.
Institute of Technology
Nejedleho sady 14, P.O.Box 314
306 14 Plzen, Czechoslovakia

Abstract

A new algorithm for 2D line clipping against non convex window that consists of linear edges and arcs is being presented. The general algorithm was derived from the Cohen Sutherland's and Liang-Barsky's algorithms and can be used especially for engineering drafting systems. The algorithm is easy to modify in order to deal with holes, too. The algorithm has been verified on IBM PC in TURBO-PASCAL.

1. Introduction

Clipping is an important part of all graphics packages. There are efficient algorithms as [1],[5],[6] but not for non convex windows that consist of linear edges and arcs. Therefore new algorithms have been developed [8] for clipping 2D line against the general windows. The below described algorithm enables to clip 2D line segments against the windows that are formed by linear edges and circular arcs without need to orient edges in the clockwise or anticlockwise order as some known algorithms require. The algorithm can be easily modified for the hatching. In this case some criteria can be simplified. If the clipping area consists of some holes it is necessary to apply the presented algorithm to all given holes themselves and merge the obtained intersection points together in a convenient way. Particular care was devoted to handle all special situations properly.

2. Non Convex Area Clipping

Provided a non-convex area is given by its vertices in the clockwise or anticlockwise order and if the edge is not linear then information whether the right or left

part of the circle is to be taken from the actual vertex, see fig.2.1. It is also assumed that all vertices have different coordinates, that no vertex lies on an edge or arc and that two edges or arcs might have only a vertex as a common point (for general conditions see [8]). Contrary to the clipping by convex window that consists of linear edges, the line w(q) can intersect the arc edge in two points. It partially increases the complexity of the given problem.

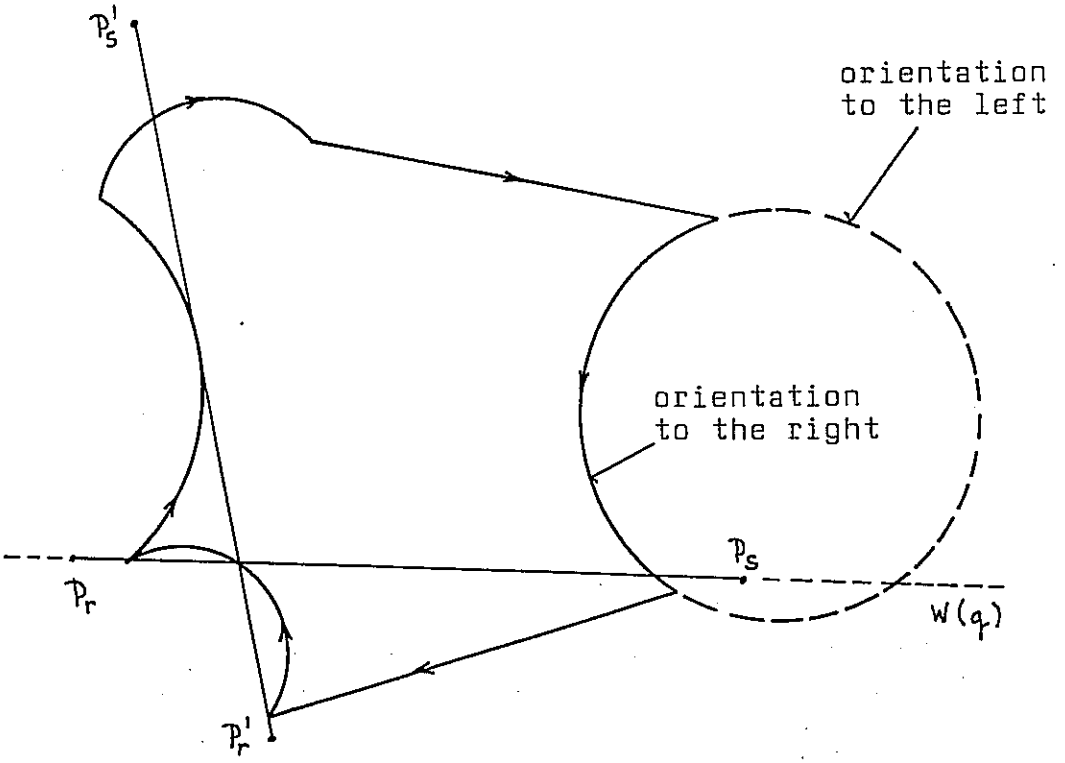


Figure 2.1.

The line w(q) is described by the parametric equation:

$$x(q) = x_r + (x_s - x_r) \cdot q \qquad q \in (-\infty, +\infty)$$

Now in case of arc edge it is necessary to solve the following equations:

$$x(q) = x_r + (x_s - x_r) \cdot q \qquad q \in (-\infty, +\infty)$$

$$(x - x_u)^2 + (y - y_u)^2 - r^2 = 0$$

where (x_u, y_u) is the centre of the given arc
$2r$ is the diameter of this arc.

Solving these equations with regard to variable q the quadratic equation

$$a\,q^2 + b\,q + c = 0$$

will be obtained, where:

$$a = (x_s - x_r)^2 + (y_s - y_r)^2$$

$$b = 2\,[(x_r - x_u) \cdot (x_s - x_r) + (y_r - y_u) \cdot (y_s - y_r)]$$

$$c = (x_r - x_u)^2 + (y_r - y_u)^2 - r^2$$

In the case that the line w(q) intersects or touches the given circle two solutions are obtained, not necessarily different, as:

$$q_{1,2} = (- b \pm \sqrt{b^2 - 4ac}) / (2a)$$

Now it is necessary to determine which part of the circle forms the boundary of the given area. Because the border is oriented it can be discerned whether the arc is on the right or on the left from the connection of $x_K x_{K+1}$ points. If the line w (q) is considered then it must be decided which intersection point ought to be taken. It is obvious that only the point which lies on the proper arc can be considered. It means that:

- if the left arc is considered then the point $x(q_i)$ will be taken into consideration if and only if

$$[s_1 \times s_2]_z > 0 \qquad i=1,2$$

- if the right arc is considered then the point $x(q_i)$ will be taken into consideration if and only if

$$[s_1 \times s_2]_z < 0 \qquad i=1,2$$

assuming that $x_K \neq x(q_i)$, $s_1 = x_{K+1} - x_K$ and $s_2 = x(q_i) - x_K$.

Of course some special situations must be solved again, e.g. when the line passes or touches the vertex x_K In those cases the tangent vectors s_1, s_2, s_3 are determined as:

- for the arc $s_1 = [y_K - y_u , x_u - x_K]$
  where (x_u, y_u) is the centre
  for linear edge $s_1 = [x_K - x_{K-1} , y_K - y_{K-1}]$

- for the arc $s_3 = [\, y_K - y_W \,,\, x_W - x_K \,]$
  where (x_W, y_W) is the centre
  for linear edge $s_3 = [\, x_{K+1} - x_K \,,\, y_{K+1} - y_K \,]$

-for the line w(q) $s_2 = [\, x_S - x_r \,,\, y_S - y_r \,]$

The possible situations are shown in table 2.1.

Table 2.1.

$[s_1 \times s_2]_z$	$[s_3 \times s_2]_z$	type "touch"/"pass"
< 0	< 0	pass
< 0	> 0	touch
> 0	> 0	pass
> 0	< 0	touch
< 0	$= 0$	if $-s_3 . s_2 > 0$ then pass else touch
> 0	$= 0$	if $-s_3 . s_2 > 0$ then touch else pass
$= 0$	< 0	if $-s_1 . s_2 > 0$ then touch else pass
$= 0$	> 0	if $-s_1 . s_2 > 0$ then pass else touch
$= 0$	$= 0$	if $-s_1 . s_2 > 0$ xor $-s_3 . s_2 > 0$ then pass else touch

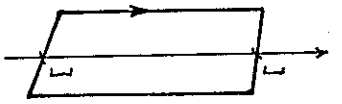
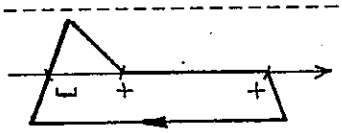
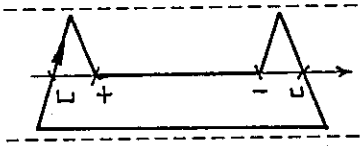
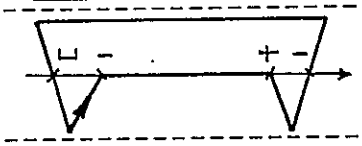
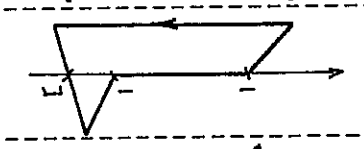
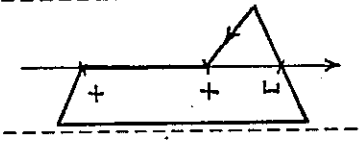
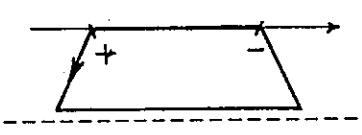
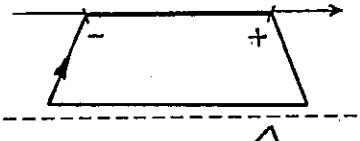
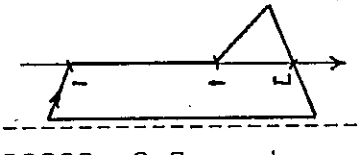
If the arc is oriented to the right then the sign of the tangent vector s must be changed in some situations.

```
PROCEDURE COMPUTE_TANGENT ( xA , xB , r , t );
BEGIN
  IF xA xB is linear
     THEN BEGIN
             s := xB - xA; r := [ s x s2]z;
          END
     ELSE BEGIN
             s := [ yK - yW, xW - xK];
             r := [ s x s2];
             (* (xW,yW) is the centre of the arc *)
             IF r=0 THEN IF t THEN r:=+s.s2 ELSE r:=-s.s2
                    ELSE IF the arc is to the right
                            THEN r := -r
          END
END (* COMPUTE_TANGENT *);

(*                                                        *)

k:=n-1; i:=0;
s2:= xS - xr; (* operation with vectors *)
WHILE i < n DO
BEGIN
  IF xK lies on the line w(q) THEN
```

```
BEGIN
  COMPUTE_TANGENT(x_K, x_i, b, TRUE);
  COMPUTE_TANGENT(x_K-1, x_K, a, FALSE);
  IF x_K x_i is linear THEN
  BEGIN
    COMPUTE_VALUE( q );
    IF a.b > 0
       THEN GENERATE( q , attribute ␣ )
       ELSE IF a.b < 0
               THEN GENERATE( q, q , attribute ␣ )
               ELSE IF a = 0 THEN
                       GENERATE( q , attribute sign b )
                    ELSE
                       GENERATE( q , attribute sign a )
  END
  ELSE (* x_K x_i is the arc *)
  BEGIN
    COMPUTE_VALUES( q_1, q_2);

       IF a.b > 0 THEN
                    GENERATE( q_1, q_2* , attribute ␣ )
       ELSE IF a.b < 0 THEN
                 GENERATE( q_1, q_1, q_2* , attribute ␣ )
            ELSE IF a = 0 THEN
                    GENERATE( q_1, attribute sign b )
                 ELSE GENERATE( q_1, attribute sign a )
  END
  ELSE IF x_K x_i linear THEN
       BEGIN
         COMPUTE_VALUE (q);
         IF an intersection point is inside of (x_K x_i )
         THEN GENERATE ( q with attribute ␣ )
       END
       ELSE
       BEGIN
         COMPUTE_VALUES( q_1, q_2);
         IF an intersection point exists
         THEN GENERATE( q_1*, q_2* , attribute ␣ )
         (* * means if the intersection point lies *)
         (* on the required side of the x_K x_i arc *)
       END
  k := i;
  i := i + 1;
END (* of while *);
SORT ( values q );
REDUCE ( set of q values according to table 2.2. );
SELECT ( subintervals as ⟨q_j, q_j+1⟩∩⟨0 , 1⟩ for all j );
COMPUTE ( the end points );
```

Figure 2.2.

Table 2.2. Basic possible situations for reduction

attributes q_i	q_{i+1}	q_{i+2}	situation	action
␣	␣	*		save (q_i , q_{i+1}); i:=i+2;
␣	+	+		save (q_i , q_{i+2}); i:=i+3;
␣	+	-		save (q_i , q_{i+2}); i:=i+2; change attribute of q to ␣
␣	-	+		save (q_i , q_{i+2}); i:=i+2; change attribute of q to ␣
␣	-	-		save (q_i , q_{i+2}); i:=i+3;
+	+	*		save (q_i , q_{i+1}); i:=i+1; change attribute of q to ␣
+	-	*		save (q_i , q_{i+1}); i:=i+2;
-	+	*		save (q_i , q_{i+1}); i:=i+2;
-	-	*		save (q_i , q_{i+1}); i:=i+1; change attribute of q to ␣

* means all cases, e.g. ␣ + -

The coordinates of the resulted points determined by their q values can be obtained from the equation for the line w(q):

$$x(q) = x_r + (x_s - x_r) \cdot q$$

It is obvious that the presented algorithm for clipping line by non-convex area can be easily modified for a case when the area is formed by linear segments and quadratic arcs. In this case it is necessary to define conveniently the quadratic arcs. A similar approach to the circle case can be chosen for this general case too. Generally all quadratic curves are described by the function $f(x,y)$ together with their tangent vectors as:

$$f(x, y) = 0 \qquad s = [\, f_y, -f_x]$$

where: $$f(x,y) = ax^2 + by^2 + 2cxy + 2dx + 2ey + g$$

If the given area consists of some holes it is necessary to apply the presented algorithm for all the given holes themselves and merge the obtained q values together.

3. CONCLUSION

The presented algorithms are based on the principle of the Liang-Barsky's algorithm. It is shown how the algorithms become more complicated if the requirements are more general. In general they do not need oriented half-planes of the clipping window. The second algorithm solves the situation when the clipping polygon is non convex. The increase of complexity is expressed in the need to distinquish between different cases and to sort the final set of intersection points. The last presented algorithm solves the problem when the clipping area is formed by line segments and arcs. This problem has not been solved in the accessible literature as far as it is known to the author. The algorithms are fast and all special cases are properly handled.

ACKNOWLEDGMENT

The author would like to express his thanks to Prof.L.M.V. Pitteway, Dr.J.P.A.Race (Brunel Univ.,U.K.), Dr.J.E.Bresenham (Winthrop College, U.S.A.) and Dr.R.A.Earnshaw (Univ. of Leeds, U.K.) for their helpful discussions during his stay in U.K., to Miss I. Kolingerova (Inst. of Technology, Plzen) for her interest, comments and the final implementation on IBM-PC, who enabled to find unspecified special cases and errors, and to the students of Computer Graphics course that stimulated this work.

LITERATURE

[1] Cyrus, M., Beck, J., Generalized Two- and Three Dimensional Clipping, Computers & Graphics, Vol.3, 1979, No.1.,pp.23-28.

[2] Foley,J.D.,van Dam A., Fundamentals of Interactive Computer Graphics, Addison-Wesley, 1982, Reading, Mass.

[3] Kilgour,A.C., Unifying Vector and Polygon Algorithms for Scan Conversion and Clipping, Report CSC 87/R7, Computer Science Dept., Univ. of Glasgow, 1987.

[4] Liang,Y.D., Barsky,B.A., An Analysis and Algorithms for Polygon Clipping, CACM 26, No.11 (November), 1984, pp.868-876.

[5] Liang,Y.D., Barsky,B.A., A New Concept and Method for Line Clipping, ACM Transaction on Graphics, Vol.3., No.1.,1984, pp.1-22.

[6] Newman,W.M., Sproull, R.F., Principles of Interactive Computer Graphics, 2nd ed., McGraw Hill, 1979, New York.

[7] Nicholl,T.M., Lee,D.T., Nicholl, R.A., An Efficient New Algorithm for 2D Line Clipping: Its Development and Analysis, ACM Computer Graphics, Vol.21, No.4., 1987, July, pp.253-262.

[8] Skala,V., A Unifying Approach to the Line Clipping Problem Solution TR-209-5-89 Computer Science Dept., Inst. of Technology Plzen, 1989.

Mezinárodní konference Algoritmy 89, JSMaF při SAV,
April 17-21, 1989, Vysoké Tatry, Štrbské Pleso (Algorithms 89)

Hidden-Line, Hidden-Surface and Hidden-Contouring Problem Solutions by the Modified Bresenham's Algorithm

Vaclav Skala
Computer Science Department, Technical University, Nejedleho sady 14, 30614 Plzen

1. Introduction

Solutions of many engineering problems result in functions of two variables which can be given either by an explicit function description or by a table of function values. Functions have usually been displayed on a display screen or have been drawn on a plotter in several manners. The known methods have been based either on drawing contours drawn in axonometric projection [3] or on drawing functions of two variables with respect to visibility ([12]-[14]). Each method of displaying must have its own specialised algorithms, which has not been simple so far. In all cases the problem of displaying has been complicated not only by an enormous volume of data but also from the point of view of programs and their structure. Many algorithms have been published in literature but they differ from each other.

In the following paragraphs a unifying approach to the Hidden-Line, Hidden-Surface and Hidden-Contouring problems will be shown. It is based on a modification of the Bresenham's algorithm for the straight line drawing. The advantage of the suggested solution is a simple hardware implementation, especially with devices controlled by microprocessors. The proposed algorithms are fast and they do not use floating-point operations at all.

2. Hidden-Line Problem Solution

Let us have an explicit function of two variables x and z $y = f(x, z)$ where: $x \in \langle ax, bx \rangle$ and $z \in \langle az, bz \rangle$ and we want to display this function by using a raster device, e.g. a raster display or a plotter. For many scientific problems it is enough to show the behaviour of the given function by drawing function slices according to the x and z axes, e.g. the curves:

$$y = f(x, z_i) \qquad i=1,\ldots,n$$

where: $x \in \langle ax, bx \rangle$ and $az=z1<z2<\ldots<zn=bz$

and the curves:

$$y = f(x_j, z) \qquad j=1,\ldots,m$$

where: $z \in \langle az, bz \rangle$ and $ax=x1<x2<\ldots<xm=bx$

The given function can be represented either by an explicit function specification or by a table of the function values for grid points in the x-z plane. If the function is rather complex it can be very difficult to imagine

the behaviour of the function because some parts are invisible. The problem has been very successfully solved by ([6],[12]-[14]).Sometimes the method is called the Floating Horizont Method.

The principle of the known solution is generally simple. If two slices parallel to the x-axis are drawn then the space between these two slices is a strip of invisibility. The strip is actually part of a surface of the given function. Let us suppose that curves are drawn from the foreground to the background. Now if the third curve should be drawn (far from the observer) it is obvious that those parts that pass through the strip of invisibility are invisible and therefore they should not be drawn, see fig.2.1.

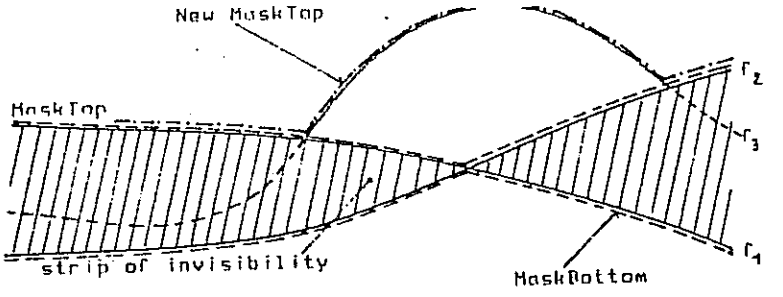
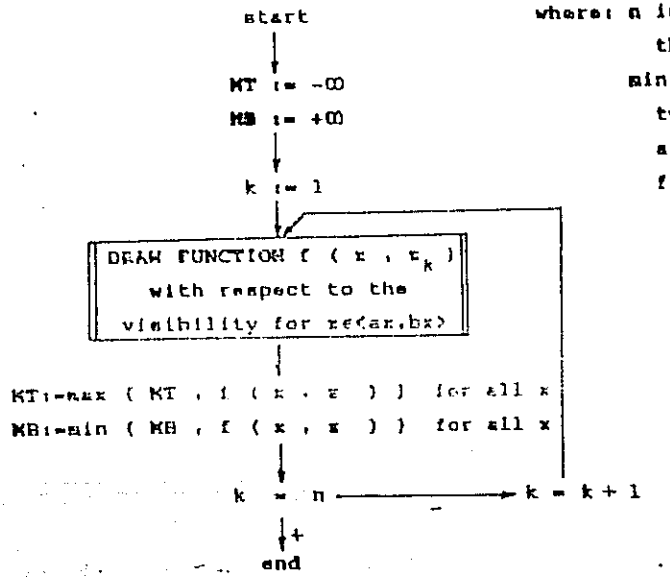


Figure 2.1.

Further the following abbreviations will be used:

MaskTop MT MaskBottom MB

If the problem is analyzed in detail, it is obvious that there is a necessity to represent borders of the strip of invisibility. It can be made by introducing MaskTop and MaskBottom functions. The real representation of these functions can be temporarily omitted. Now, the problem of drawing the curves with respect to visibility becomes very simple as it is shown in algorithm 2.1., because only those parts of the curves, the points of which are lying outside the strip of invisibility, are drawn. In [6] the problem is straightly solved and the description of the method is easy to understand.



where: n is the number of slices of the function f

min, max are functions whose two arguments are functions and the results are functions too

Algorithm 2.1.

The visibility problem has been solved by Watkins [12] introducing mask vectors for MaskTop and MaskBottom boundary representations. Several problems had to be solved because all computations were done in the floating point representation.

3. Proposed Method for Hidden-Line Solution

In [11] the functions MaskTop and MaskBottom are represented by vectors with values in the floating point representation. The whole process of drawing with respect to visibility can be imagined as follows in fig.3.1.

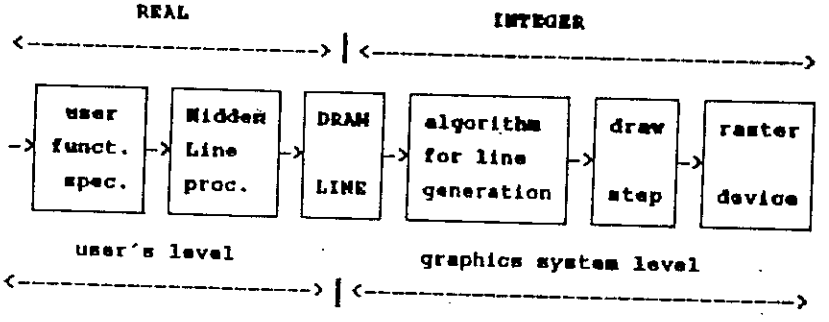


Figure 3.1.

Now a question might arise if there are any possibilities of increasing the efficiency of the hidden-line problem solution. One possibility is to combine the known Watkins's algorithm with the Bresenham's algorithm for drawing straight lines direct on the physical level.

A solution of the hidden-line problem becomes relatively very simple now because only the DRAW STEP procedure must be changed. This procedure must generate code for physical movement in order to take the invisibility into account. Because the DRAW STEP procedure draws one physical step it must only be checked whether the next end-point in the raster is inside or outside of the strip of invisibility. The structure of the proposed method is shown in fig.3.2.

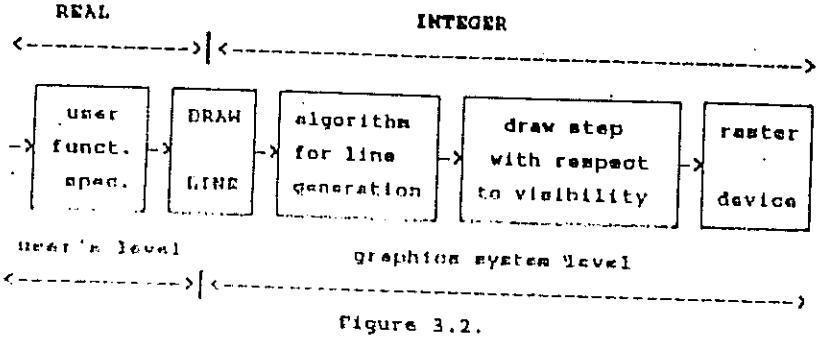


Figure 3.2.

It is obvious that only integer representation for MaskTop and MaskBottom arrays are acually needed. The simplified solution is shown in algorithm 3.1. There are two temporary arrays M1T and M1B for new masks in order to get rid of all very special problems with near horizontal lines. Spaces were included into identifiers for legibility and in the actual implementation will be omitted.

```
( Global variables )
CONST res=1023;
VAR xp,yp:INTEGER; ( absolute coordinates )
    mt,mb:ARRAY [0..res] OF INTEGER;( MaskTop & MaskBottom )
    mlt,mlb:ARRAY [0..res] OF INTEGER; ( temporary mt & mb )
    u,v,ix,iy:INTEGER;
( Procedure for the first initialization of masks )
PROCEDURE initialize;
VAR i: INTEGER;
BEGIN FOR i:=0 TO res DO
      BEGIN mlt[i]:=-maxint;mlb[i]:=maxint END
END;
( Procedure for masks setting after each slice has )
( been drawn )
PROCEDURE set up masks;
BEGIN mt:=mlt;mb:=mlb END;
( Draw step with respect to visibility of the given point )
PROCEDURE draw step;
BEGIN IF yp < mb[xp] THEN switch on (xp,yp);
      IF yp > mt[xp] THEN switch on (xp,yp);
      IF yp > mlt[xp] THEN mlt[xp]:=yp;
      IF yp < mlb[xp] THEN mlb[xp]:=yp
END;
( Procedure for drawing straight line 0<=abs(v)<=abs(u) )
PROCEDURE bresenham;
VAR j,d,a,b: INTEGER;
BEGIN IF u >= v THEN
      BEGIN ( 1.,4.,5.,8. octant )
            a:=v+v;d:=a-u;b:=a-u-u;
            FOR j:=1 TO u DO
            BEGIN IF d < 0 THEN d:=d+a
                           ELSE BEGIN yp:=yp+iy;d:=d+b END;
                  xp:=xp+ix;
                  draw step
            END
      END
      ELSE
      BEGIN ( 2.,3.,6.,7. octant )
            a:=u+u;d:=a-v;b:=a-v-v;
            FOR j:=1 TO v DO
            BEGIN IF d < 0 THEN d:=d+a
                           ELSE BEGIN xp:=xp+ix;d:=d+b END;
                  yp:=yp+iy;
                  draw step
            END
      END
END;
PROCEDURE draw to ( x,y: INTEGER );
( draw line with respect to visibility (xp,yp)->(x,y) )
BEGIN ix:=sign (x-xp);iy:=sign(y-yp);
```

```
      u:=abs(x-xp);    v:=abs(y-yp);
      bresenham
END;
BEGIN ( now the drawing itself - body of the main program )
      initialize;
9:    set up masks; ( masks must be set up for each slice )
      .....
      draw to (x,y); ( draw the function slice )
      .....
      goto 9;
      .....
END.
```

Algorithm 3.1.

In the above presented algorithm it is assumed that the order of the drawing is from the foreground to the background. Sometimes the foreground and the background are altered in the published algorithms, e.g. in [12], and the results are wrong, because the order in which the curves are drawn cause a violation of the given premises. Therefore the problem is how to select the order of the drawing if some transformations are made. It is also assumed that we use only parelel projection and that vertical lines remain vertical after all transformations too.

4. Hidden-Surface Problem Specification

The problem of the hidden-surface elimination is very often solved in a quite different ways in which many tricks are employed. Because all the known methods (for drawing functions of two variables) do not use the advantage of the solution in the raster environment a very simple algorithm will be described. The problem of the hidden-surface elimination will be solved in a very similar manner to the hidden-line elimination shown above. The slices will be drawn again from the foreground to the background and after drawing the first two slices two masks for borders and two masks for intensity levels will be set up, see fig.4.1.

Further the following abbreviations will be used:

Mask Current	MC	Intensity Current	IC
Mask Previous	MP	Intensity Previous	IP
Mask Top	MT	Mask Bottom	MB

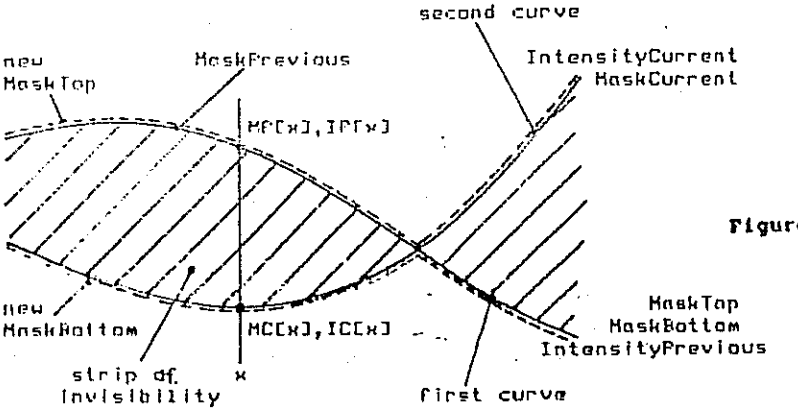
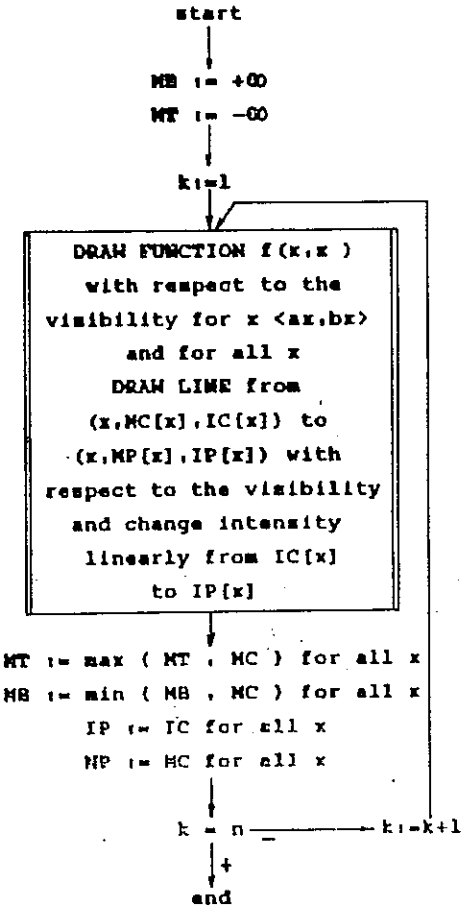
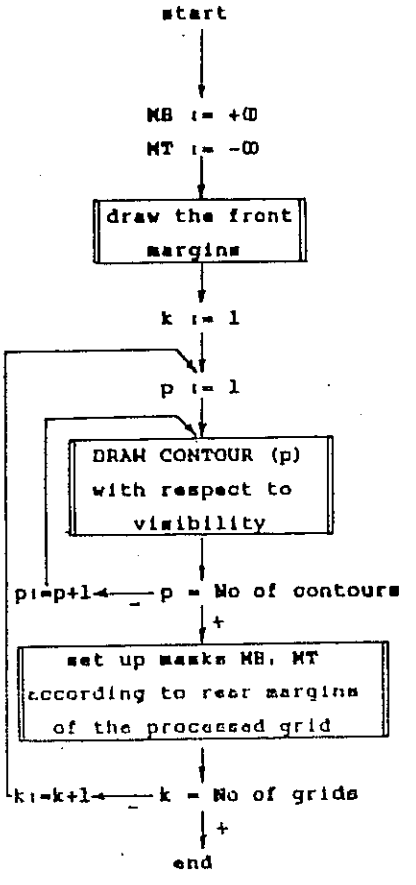


Figure 4.1.

The problem is that the invisible parts of the surface must be deleted and an appropriate visible part of the surface must be filled by an appropriate grey intensity level. To do this for all points of each curve an intensity level is needed. The intensity level can be computed as a function of the normal vector of the given surface. In the given coordinate system the x part of the normal vector must be taken, for details see [6].



Algorithm 4.1.



Algorithm 5.1.

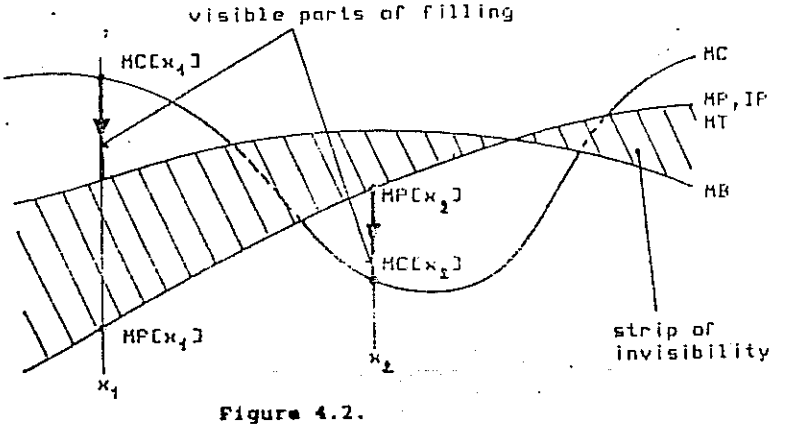


Figure 4.2.

It means that after drawing the first slice the functions MaskTop, MaskBottom (Maskbottom is equal to the MaskTop after the first slice was

drawn), MaskPrevious and IntensityPrevious are known. If the second slice is drawn then the functions MaskCurrent and IntensityCurrent are known too. Now it is necessary to fill the known part of the surface that is defined by the functions MaskPrevious and MaskCurrent with an appropriate level of grey. Because all functions will be represented again by vectors for all points in the x-axis a modified Bresenham's algorithm can be employed in order to get proper intensity levels for all points. It means that for the given x the intensity level will be approximated for all possible y, e.g. a line will be drawn from the point (x,MC[x]) to the point (x,MP[x]) and the intensity level will be slowly changing from IC[x] value to IP[x] value. This must be performed for all x. In this way we will get the strip of invisibility. After that MaskTop, MaskBottom are redefined again, see fig.4.2.

If a third curve is drawn then it will be drawn only outside the strip of invisibility. But for the hidden surface it is necessary to fill the area outside the strip of invisibility between the previous curve represented by MP and the current curve represented by MC, e.g. it is necessary to fill in only a visible part of the line segment from the point (x,MC[x]) to the point (x,MP[x]) with a proper intensity level from intensity IC[x] to intensity IP[x]. The filling can be done by a modified Bresenham's algorithm again in order to ensure that the intensity level will be properly changed. It can be seen that the proposed algorithm does need all the above mentioned vectors. The whole algorithm (schematically) is shown by algorithm 4.1.

If we omit for this moment all problems with the initilization then we can use all the procedures shown above but the procedures DRAW STEP and SET UP MASKS must be changed as it is shown in algorithm 4.2. The actual implementation is slightly more complex.

```
PROCEDURE draw step;
BEGIN ( draw step to the point (xp,yp) with intensity i)
      flag:=x0=xp; xo:=xp;
      IF yp > mlt[xp] THEN mlt[xp]:=yp;
      IF yp < mlb[xp] THEN mlb[xp]:=yp;
      IF yp <= mb[xp] OR yp >= mt[xp] THEN
         IF flag THEN fill(xp,yp,mp[xp],i,ip[xp])
           ( from the point (xp,yp) with an intensity i to )
           ( the point (xp,mp[xp]) with an intensity ip[xp])
                ELSE switch on(xp,yp,i);
      mc[xp]:=yp;
      ic[xp]:=i
END;
PROCEDURE set up masks;
BEGIN mt:=mlt;mb:=mlb;
      mp:=mc;ip:=ic
END;
```

Algorithm 4.2.

The algorithm shown above is very simple and clear to understand. We have not dealt with the problem how MP and IP arrays were originally set up. The FILL procedure is actually the Bresenham's algorithm that is modified so that x coordinate is constant, y coordinate is changed with the step 1 and the intensity level is appropriately changed in order to get the whole intensity

scale from the current intensity represented by IC[x] value to the previous intensity represented by IP[x] value or vice versa. The procedure itself must draw only outside the given strip of invisibility that is given by MT and MB arrays.

5. Hidden-Contouring Problem Specification

The hidden-contouring problem is described in literature very rarely because it covers several non-trivial tasks. The first is the contouring problem itself, the second is the problem of hidden-line elimination. The known algorithms are very complicated [3]. The main effort seems to be spent on the part that deals just with the hidden line removal. The known algorithms just use the algorithm for contouring and hidden line removal in the "pipe-line" way. The problem can be easily solved again in a very similar manner to the hidden-line elimination described above. If fig.5.1. is analysed we can see that the problem can be easily described as follows. Firstly two margins must be drawn and then for each grid all contours must be drawn with respect to the visibility. The contour drawing order is substantial because the contours must be drawn so that the higher contours are drawn later on, e.g.:

contour(p) < contour(p+1) for all p

When all contours for the given grid have been drawn it is necessary to draw "back margins" of the given grid. If any user needs smaller grids or smoothly interpolate contours he can subdivide grid mesh or he can employ the smooth interpolation (some kind of smooth interpolation can be done direct in the raster environment). The whole problem is now simple but we have to find a simple method which determines silhouetting in order to get the proper outlook. The silhouetting problem solution is described in literature very rarely because of high complexity of computation. Actually it is a problem of determining whether the partial derivation of the given function according to the observer's eye direction is zero. The hidden-contouring problem solution (without the solution of silhouetting) can be schematically described by the algorithm 5.1.

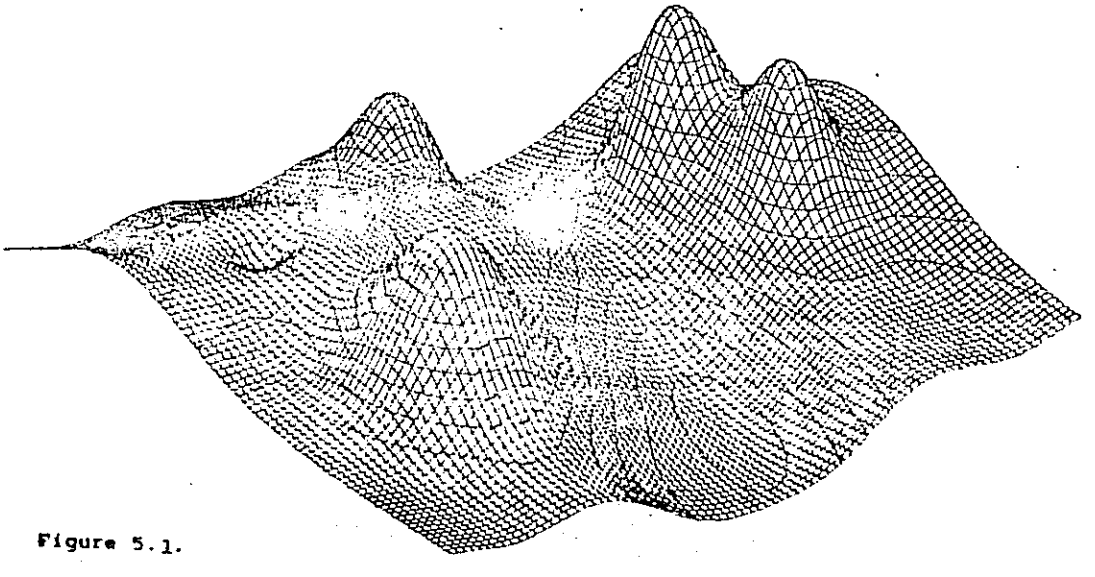


Figure 5.1.

6. Conclusion

The above presented algorithms are based on the Bresenham's algorithm for drawing straight lines in the raster environment. The algorithms for drawing functions of two variables are intended for the use in the raster environment, e.g. for raster displays or digital plotters. The presented algorithms for hidden-line, hidden-surface and hidden-contour removal are very fast, easy to implement and they do not need operations in the floating point representation for the determining whether the line segment or its part is visible or not.

All mentioned algorithms were implemented on the 8-bit microcomputer with 8080 microprocessor running at 3MHz. Presented results were drawn in 2 sec. approximately including the visibility solution with the resolution 256x256 points with 8 colours. The time of function computation, scaling and rotation is not included, because a host computer was used. The drawing was about 15% slower than drawing without the solution of the visibility.

7. Literature

[1] Bresenham,J.E.: Algorithm for Computer Control of Digital Plotter, IBM Syst.J. 4(1) 1965, pp.25-30.

[2] Boutland,J.: Surface Drawing Made Simple, Computer Aided Design 11(1), January 1979, pp.19-22.

[3] Clapworthy, G.J.: The Representation of a Geographical Terrain from Cartographic Data, MSc.Thesis, The City University of England,London, 1985

[4] Earnshaw,R.(Ed.): Fundamental Algorithms for Computer Graphics, NATO ASI, Series F, Vol.17, Springer Verlag 1985.

[5] Piteway, M.L.V.,Hatkinson, D.: Bresenham's Algorithm with Gray Scale, Communication of ACM 23(11), November 1980, pp.625-626.

[6] Skala, V.: Drawing of Functions of Two Variables with Respect to Visibility,TR 209-05-78, Computer Science Dept.,Technical University, Plzen 1978 (in Czech).

[7] Skala,V.: Hidden-Line Processor, TR 209-03-84, Computer Science Dept., Technical University, Plzen 1984.

[8] Skala,V.: Hidden-Line Processor (a special case), CSTR/29, Brunel Univ., Middlesex,England, 1984.

[9] Skala,V.: An Interesting Modification to the Bresenham Algorithm for Hidden-Line Solution, Fundamental Algorithms for Computer Graphics, NATO ASI, Series F, Vol.17, Springer Verlag, (Edited by Earnshaw R.A.), 1985, pp.593-601.

[10] Sowerbutts,W.T.: A Surface-Plotting Program Suitable for Microcomputers, Computer Aided Design 15(6), November 1983, pp.324-327.

[11] Watkins, S.L.: Masked Three-Dimensional Plot Program with Rotation, Communication of ACM 17(9), September 1974, pp.520-523.

[12] Williamson,H.: Hidden - Line Plotting Program, Comm. of ACM 15(2), February 1972, pp.100-103.

[13] Wright,H.A.: A Two-Space Solution for the Explicit Functions of Two Variables, IEEE Trans. on Computers 22(1),January 1973, pp.28-33.

[14] Boller,D.J.: Efficient Hidden Line Removal for Surface Plots Utilizing Raster Graphics, NATO ASI, Vol.17, SerF, Springer Verlag, pp.603-615, 1985.

Mezinárodní konference Computer Graphics International '89
New Advances in Computer Graphics, Springer Verlag, 1989

Algorithms for 2D Line Clipping

V. Skala

ABSTRACT

New algorithms for 2D line clipping against convex and non-convex windows are being presented. Algorithms were derived from the Cohen-Sutherland´s and Liang-Barsky´s algorithms. Algorithms are easy to modify in order to deal with holes too. The presented algorithms have been verified in TURBO-PASCAL. Because of unifying approach to the clipping problem solution algorithms are simple, easy to understand and implement.

Keywords: line clipping, convex polygon, non-convex polygon, algorithms

1. INTRODUCTION

Clipping is a very important part of all graphics packages. Generally it is the evaluation of a line intersection against a window boundary. Many efficient algorithms are known, as the Cohen-Sutherland´s (Newman, Sproull 1979), Liang-Barsky´s (1983) , Cyrus-Beck´s (1978) ones. All these algorithms have some presumptions, e.g. the windows must be orthogonal or convex with oriented edges, etc.

In the following new algorithms will be described for convex-polygon and non-convex polygon clipping without need to orient edges in any order. Particular care was devoted to handle all special situations properly. Algorithms are based on the only basic idea that is gradually widened for more general cases.

As far as the author is concerned none of these algorithms have been published in the accessible literature.

2. CONVEX POLYGON CLIPPING

The bellow given convex polygon clipping algorithm is based on the principle of Liang-Barsky´s algorithm and is simpler than the Cyrus-Beck algorithm and does not need an anticlockwise orientation of the polygon edges as Liang-Barsky´s algorithm does.

Provided a convex polygon is given by its vertices in the clockwise or in the anticlockwise order arbitralily and none pair of edges lies on the same line (it is not a principle restriction). Let us consider some situations that might occur if a line segment with end points P_r and P_s ought to be clipped, see fig.2.1.

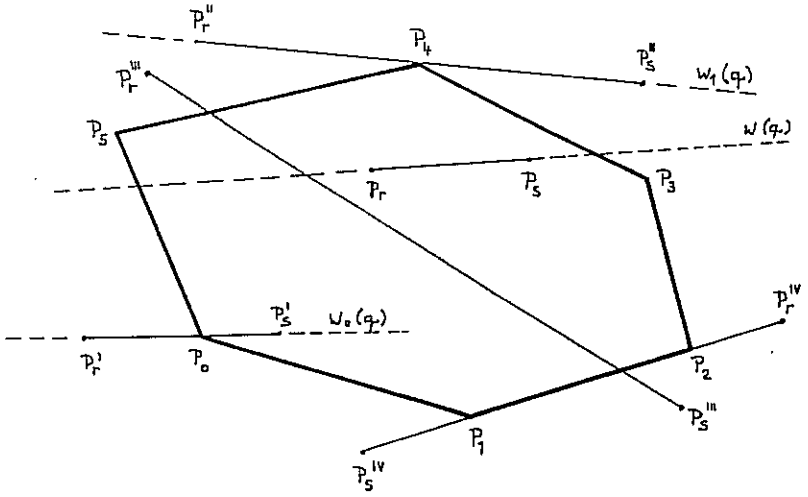


Fig.2.1.

All intersections of the line w with edges of the convex polygon are obtained by solving the following linear equations:

$$x(q) = x_r + (x_s - x_r) . q \qquad q \in \langle 0 , 1 \rangle$$

$$x(p) = x_i + (x_{i\oplus 1} - x_i) . p \qquad p \in \langle 0 , 1) \quad i=0,1,\ldots,n-1$$

where $\oplus$ means addition modulo n and point P_K has coordinates x_K

The interval for parameter p is not closed in order to get rid of all ambiguities in case that the line segment is "passing" (line $w_0(q)$) or "touching" (line $w_1(q)$) a polygon vertex. The bellow given algorithm is based on the fact that a line segment can intersect a convex polygon only in two points. The algorithm finds the q values for all intersection points of a line on which the given line segment lies with the edges of the convex polygon and then the proper part of the line segment that is inside the convex polygon can be found. The algorithm is shown in fig.2.2. It is necessary, of course, to solve special cases when a line segment touches or passes a vertex or when it lies on a polygon edge.

The algorithm given in fig.2.2. is faster than the Cyrus-Beck's algorithm (it doesn't need an inner normal computation) and for a rectangle polygon is equivalent to the Liang-Barsky algorithm in case that computation of all intersections is simplified for polygon edges parallel to the axes. The algorithm can be easily generalized or modified for a case when two edges of the given polygon lie on the same line Skala (1988).


```
VAR i,k: INTEGER; (* end points of the polygon edges *)
    j: INTEGER; (* counter *)
    t: ARRAY [1..2] OF REAL;

BEGIN
   j:=0;i:=0;k:=n-1; (* set end points for the first edge *);
   REPEAT
      IF an intesection point exists for the edge xK xi
      AND the line w(q) so that p ∈ <0,1) THEN
      BEGIN j:=j+1; t[j]:=q (* save the q value *) END
      ELSE
      IF the edge xK xi lies on the line w(q) THEN
      BEGIN t[1]:= a value q that corresponds to the
                   vertex xK;
            t[2]:= a value q that corresponds to the
                   vertex xi;
            j:=2
      END;
      k:=i; i:=i+1; (* take the next polygon edge *)
   UNTIL ( j = 2 ) OR ( i > n );
   IF j <> 0 THEN
   BEGIN
      IF j = 1 THEN t[2]:=t[1] (* the line w(q) "touches"
                                  a vertex *)
               ELSE IF t[1] > t[2] THEN t[1] SWAP t[2];
      t[1]:= max ( 0.0 , t[1] ); (* maximal value *)
      t[2]:= min ( 1.0 , t[2] ); (* minimal value *)
      LINE ( x (t[1]) , x (t[2]) )
   END
END;
```

Fig.2.2.

3. NON-CONVEX POLYGON CLIPPING

An algorithm for non-convex polygon clipping is based on the parametric equations that express linear segments. In this case the algorithm must be more complex, because the line w(q) can intersects the polygon in many points. Assume that a non-convex polygon is given by its vertices in clockwise or anticlockwise order. Further that two successive edges do not lie on the same line, that all vertices have different coordinates, that not a single vertex lies on any edge of the given polygon and that two edges might have only a vertex as a common point.

Let us consider again some situations that might occur if a line segment with end points P. and P. ought to be clipped, see fig.3.1. The given line segment that ought to be clipped can be expressed by:

$$x(q) = x_r + (x_s - x_r) . q \qquad q \in \langle 0 , 1 \rangle$$

and the edges of the non-convex polygon can be expressed by:

$$x(p) = x_i + (x_{i\oplus 1} - x_i) . p \qquad p \in \langle 0 , 1) \quad i=0,1,\ldots,n-1$$

Then we will look for all intersection points of the line w(q) and non-convex polygon. The parametric equation:

$$x(q) = x_r + (x_s - x_r) . q \qquad q \in (-\infty , +\infty)$$

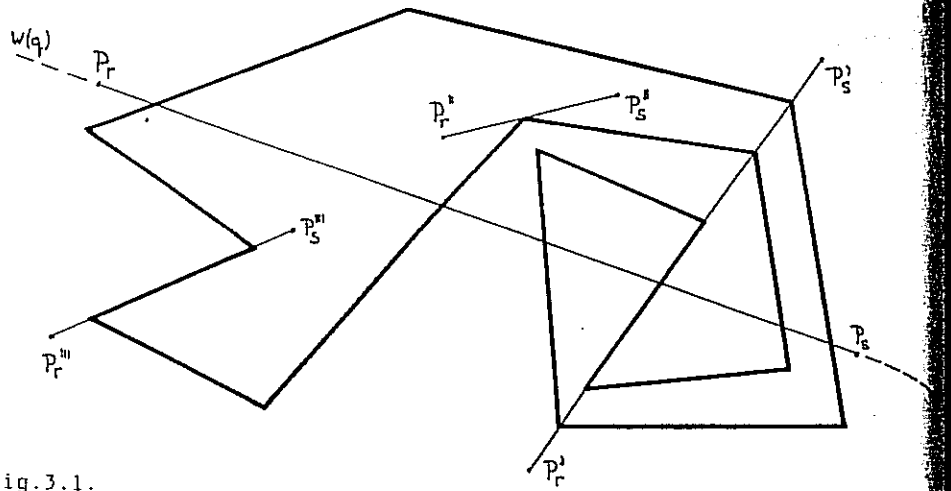
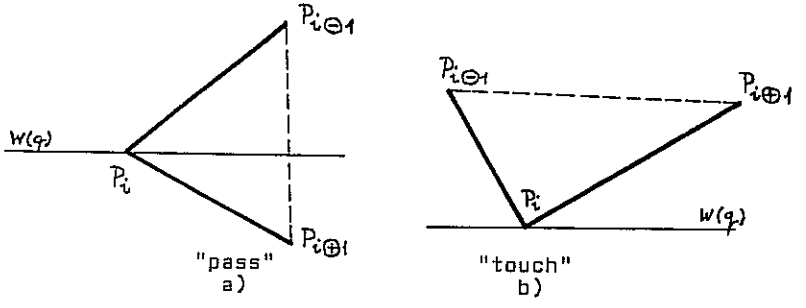


Fig.3.1.

expresses the line w(q). Coordinates of all intersection points will be determined by the value of the parameter q. But it is necessary to take into consideration the following special cases when the line w(q) passes or touches a vertex of the polygon. There are two possibilities, see fig.3.2.:



where $\oplus$ means addition modulo n, $\ominus$ means subtraction modulo n

Fig.3.2.

In fig.3.2.a. is generated only one intersection point, while in fig.3.2.b. double intersection point is generated. In both cases these points are processed as an ordinary intersection point.

A quite different situation is when the line w(q) lies on an edge of the polygon. There are two possible situations, see fig.3.3.

The values of q parameters that correspond to the points P_i and $P_{i\oplus1}$ in these cases must be generated. But it is necessary to distinguish between the type by attributes ++ or -- for these q values. If points $P_{i\oplus2}$, $P_{i\ominus1}$ are the same, then it is necessary to generate only q value which corresponds to the point P_i, see fig.3.3.c. (a special case when the given polygon is a triangle). Therefore the intersection point will be determined not only by value q but also by a type of the

intersection with edge, intersection of the type pass
or touch
line w(q) lies on edge - case ad a)
line w(q) lies on edge - case ad b)

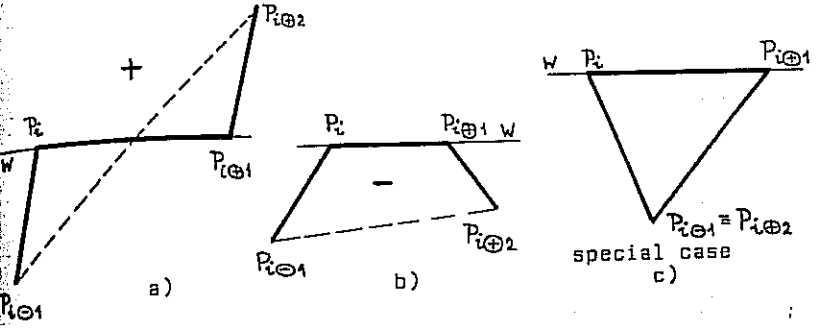


Fig.3.3.

When all intersection points are found together with their types, the given set of q values is sorted together with their attributes so that couples with attributes ++ or -- should not be split, e.g. sequencies:

$q_1\ q_1\ q_2$ (+ ␣ +) or $q_1\ q_2\ q_2$ (+ ␣ +)

must be transferred to sequencies:

$q_1\ q_1\ q_2$ (␣ + +) or $q_1\ q_2\ q_2$ (+ + ␣)

Similarly for attributes --.

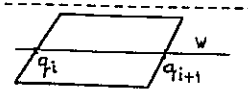
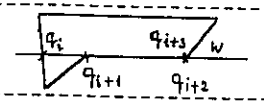
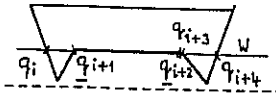
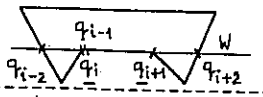
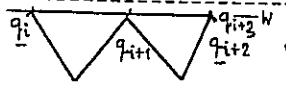
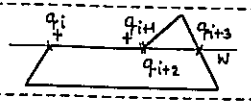
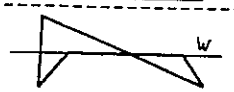
The set of q values will be processed according to table 3.1. Results that determine these parts of the line w(q) which are inside the given polygon are couples of q values.

Now it is necessary to determine which parts of the line segment P_rP_s are inside the given polygon, e.g. to determine those parts of the line w(q) that are inside and that are part of the line segment P_rP_s. According to the process of getting parts of the line w(q) it is necessary to make intersection of all couples of the q values with the interval $\langle 0 , 1 \rangle$, see fig.3.4.

```
i:=1;
WHILE i <= No of intersection -1 DO
BEGIN IF max ( 0.0 , q_i ) = min ( 1.0 , q_{i+1} )
      THEN save ( max ( 0.0 , q_i ) , min ( 1.0 , q_{i+1} ) );
      i:=i+2
END;
```

Fig.3.4.

Table 3.1. Possible situations

attributes q_i	attributes q_{i+1}	situation	action
␣	␣		save (q_i , q_{i+1}) i:=i+2
␣	+		save (q_i , q_{i+3}) i:=i+4
␣	-		save (q_i , q_{i+2}) i:=i+3
-	-		save (q_i , q_{i+2}) i:=i+3
-	␣		vertex touches an edge
+	+		save (q_i , q_{i+1}) i:=i+2
+	␣		improper polygon
-	+		impossible situation
+	-		impossible situation

The coordinates of the resulted points determined by their q values can be obtained from the equation for the line w(q)

$$x(q) = x_r + (x_s - x_r) . q$$

The whole algorithm can be described as follows in fig.3.5. The presented algorithm enables to clip a given line segment against non-convex polygon. The assumptions stated above are not substantial with the only exception that edges may not intersect one another and algorithm can be easily modified for these assumption. The algorithm is based on the similar idea as the previous one and because the polygon in non-convex some additional operations must be employed.


```
j:=0; (* counter of q values *)
k:=n-1;
FOR i:=0 TO n-1 DO
BEGIN IF the edge x_k x_i lies on the line w(q) THEN
BEGIN determine the type;
      IF x_k <> x_k⊕2 (* not a triangle *)
      THEN IF type = '+' THEN generate ( q_1 , q_2 )
                         ELSE generate ( q_1 , q_2 )
      ELSE generate ( q_1 )
END
ELSE IF the line w(q) passes or touches the vertex x_k
     THEN IF type is touch THEN generate ( q_1 , q_1 )
                           ELSE generate ( q_1 )
     ELSE IF the line w(q) intersects edge x_k x_i
          THEN generate ( q_1 );
k:=i
END;
SORT ( values q );
REDUCE ( set of q values );
SELECT ( subintervals as <q_k , q_k⊕1> ∩ <0 , 1> );
COMPUTE ( the end points );
```

Fig.3.5.

4. CONCLUSION

The presented algorithms are based on the principle of the Liang-Barsky´s algorithm. It is shown how the algorithms become more complicated if the requirements are more general. The presented algorithms were verified in TURBO-PASCAL on IBM-PC. The algorithm for the clipping line against a convex polygon is simpler than the Liang-Barsky´s, if it is simplified for a rectangle clipping window, and in general it doesn´t need oriented half-planes of the clipping window. The second algorithm solves the situation when the clipping polygon is non-convex. The increase of complexity is expressed in the need to distinquish between different cases and to sort the final set of intersection points. The algorithms are fast and all special cases are handled properly. Both algorithms might be used for hatching, too. Of course, in that case it is convenient to rotate the clipping area so that the hatching lines are horizontal, find intersection points and then rotate their coordinates back. In this case all algorithms can be significantely simplified. The presented algorithms can be generalized for non-convex areas when the clipping area is formed by line segments and arcs (Skala 1988).

5. ACKNOWLEDGMENT

The author would like to express his thanks to Prof.L.M.V. Pitteway, Dr. J.P.A. Race (Brunel Univ., U.K.), Dr. J.E. Bresenham (Winthrop College, U.S.A.) and Dr.R.A. Earnshaw (Univ. of Leeds, U.K.) for their helpfull discussions during his stay in U.K., to Miss I. Kolingerova (Inst. of Technology, Plzen) her interest, comments and the final implementation on IBM-PC, who enabled to find unspecified special cases and errors, and to students of Computer Graphics course that stimulated this work.

6. LITERATURE

Cyrus M, Beck J (1978) Generalized Two- and Three Dimensional Clipping, Computers and Graphics, Vol.3, No.1., pp.23-28.

Foley JD, van Dam A (1982) Fundamentals of Interactive Computer Graphics, Addison-Wesley, Reading, Mass.

Liang YD, Barsky BA. (1983) An Analysis and Algorithms for Polygon Clipping, CACM 26, No.11 (November), pp.868-876.

Liang YD, Barsky BA (1984) A New Concept and Method for Line Clipping, ACM Transaction on Graphics, Vol.3., No.1., pp.1-22.

Newman WM, Sproull RF (1979) Principles of Interactive Computer Graphics, 2nd ed., McGraw Hill, New York.

Nicholl TM, Lee DT, Nicholl RA (1987) An Efficient New Algorithm for 2D Line Clipping: Its Development and Analysis, ACM Computer Graphics, Vol.21, No.4., July, pp.253-262.

Skala V (1988) Line Clipping by Polygons and Areas with Arcs. TR-209-11-88 Computer Science Dept., Inst.of Technology, Plzen.

Vaclav Skala is a associate professor of computer science at the Institute of Technology. He studied Technical Cybernetics and Computer Science at the Institute of Technology in Plzen. In 1975 he took a master degree in Computer Science followed by a PhD degree specialising in Database Systems at the Technical University in Prague. From 1975 to 1981 he worked as a researcher. In 1978 he studied Computer Science at MEI in Moscow and in the academic year 1983-84 Computer Graphics at Brunel University in London. In 1981 he took up position as senior lecturer at the Cybernetics Department teaching programming languages, database systems and Computer Graphics. He is a member of Czechoslovak Scientific and Technical Society and a member of the Society of Cybernetics, chairman of Computer Graphics and CAD Systems Group within the Czechoslovak Academy of Sciencies.

METODY PŮLTÓNOVÁNÍ A JEJICH SROVNÁNÍ

Ivana Kolingerová Václav Skala
Katedra informatiky a výpočetní techniky,
Vysoká škola strojní a elektrotechnická, Plzeň

1. ÚVOD

V mnoha aplikacích je nutno řešit problém, kdy je dán digitalizovaný obraz o 2^m úrovních šedi (nebarevný) a je třeba jej zobrazit na výstupním zařízení (obrazovce, tiskárně), které má k dispozici 2^k úrovní šedi, $k < m$ (k,m jsou přirozená čísla), přičemž je požadováno, aby ztráta informace při redukci počtu jasových úrovní byla co nejmenší.

Ve většině případů je nutné se omezit na $k = 1$, tedy výstupní zařízení se dvěma úrovněmi jasu (černou a bílou).

Digitální obraz bude při výkladu metod chápán jako celočíselná matice intenzit jasu v jednotlivých obrazových bodech (pixelech), tedy $I[x,y]$, $x = xmin, xmin+1, \ldots, xmax$, $y = ymin, ymin+1, \ldots, ymax$. Hodnoty v matici se mohou pohybovat od 0 odpovídající černé do $I_{max} = 2^m - 1$ odpovídající bílé barvě.

Techniky řešící tuto problematiku se nazývají půltónování. Jsou založeny na následujícím principu :

Následkem omezené rozlišovací schopnosti oka, dané mozaikovou strukturou sítnice, integruje oko intenzity světla v oblastech menších než je mez rozlišení. Skupiny černých a bílých bodů sestavené podle určitých pravidel proto při dostatečně jemném rozlišení mohou vytvářet dojem šedé barvy.

2. METODY PŮLTÓNOVÁNÍ

Je nutné zdůraznit, že žádnou z dále uvedených metod nelze jednoznačně prohlásit za nejlepší. Dosažené výsledky se značně liší podle rozsahu a rozložení intenzit v původním obrázku, druhu použitého výstupního zařízení a v neposlední řadě i podle subjektivního názoru uživatele a účelu vytvářeného obrazu.

Prahování

Nejjednodušší možnou metodou řešení uvedené úlohy je prahování. Přesáhne-li intenzita pixelu určitou prahovou hodnotu, pixel se zobrazí jako bílý, v opačném případě jako černý, viz alg. 1. Prahová hodnota T je nastavena na vhodnou hodnotu, např. přibližně na střední hodnotu.

```
const xmin = 0; xmax = 1023; { rozsah rastru pro smer x }
      ymin = 0; ymax = 1023; { rozsah rastru pro smer y }
for y := ymax downto ymin do
    for x := xmin to xmax do
    begin if I[x,y] > T then pixel(x,y) := bila
                        else pixel(x,y) := cerna;
          Display pixel(x,y)
    end;
```

kde I[x,y] je intenzita pixelu (x,y)

Algoritmus 1

Je-li k dispozici více úrovní šedi, pak původních 2^m úrovní jasu je třeba sdružit do 2^k tříd tak, aby každé třídě odpovídala jedna úroveň.

Kvalita výsledného obrazu u této metody závisí především na způsobu sdružení intenzit do tříd. Např. pro k = 1 je možné volit práh oddělující obě třídy uprostřed intervalu intenzit, které se v obraze vyskytují, avšak tato volba vede k úspěchu pouze u obrazů s rovnoměrným zastoupením všech intenzit.

Pro volbu prahu respektující specifika konkrétního digitalizovaného obrazu je vhodné spočítat četnosti výskytu jednotlivých odstínů a z nich určit vážený průměr intenzity v obraze. Označíme-li četnost i-tého odstínu c_i a chápeme-li ji jako počet výskytů daného odstínu v obraze, pak vážený průměr intenzity v obraze je dán vztahem

$$Ip = \frac{1}{n} \sum_{i=0}^{I_{max}} c_i * i,$$

kde i nabývá všech hodnot intenzity jasu, které se v obraze mohou vyskytovat (0 až I_{max}), n je celkový počet obrazových elementů (pixelů).

Získaná hodnota se podle možného počtu odstínů na výstupním zařízení buď přímo použije jako práh, anebo se z ní odvodí všechny požadované meze jednotlivých tříd.

Metoda prahování je velice jednoduchá, avšak ani při pečlivé volbě prahu nejsou výsledky uspokojivé. Použití prahových funkcí přináší příliš velkou ztrátu obrazové informace, neboť dochází k poměrně velkým chybám při zobrazování jednotlivých pixelů. Metoda prahování vede k potlačení detailů a velmi tvrdým kontrastům mezi plochami jednotlivých odstínů.

Floyd - Steinbergův algoritmus

Jedná se v zásadě o adaptivní algoritmus dvouúrovňového prahování s korekcí. Intenzita v každém bodě obrazu se srovnává s prahovou hodnotou. Podle výsledku testu se daný pixel nastaví na černou nebo na bílou barvu. Chyba vzniklá prahováním se rozloží do těch sousedních pixelů, které ještě nebyly zpracovány. V literatuře ([1],[5],[7]) se lze setkat se dvěma schématy distribuce chyby, viz obr. 1, lze navrhnout i další, avšak zvolené váhové koeficienty by měly mít součet roven jedné.

Práh se obvykle volí jako aritmetický průměr maximální a minimální intenzity, protože jeho přesnější stanovení nemá na kvalitu obrazu výrazný vliv jako u metody prahování.

Pak algoritmus pro prahovou hodnotu T = (černá+bílá)/2 a sch

distribuce chyby z obr. 1.1 má tvar:

```
const xmin = 0; xmax = 1023; { rozsah rastru pro směr x }
      ymin = 0; ymax = 1023; { rozsah rastru pro směr y }
var T, Error: real;
    x, y: integer;
begin T := ( černá + bílá ) / 2;
   for y := ymax downto ymin do
       for x := xmin to xmax do
       begin { určení hodnoty pixelu pro práh T a chyby Error }
          if I[x,y] < T then
          begin Pixel(x,y) := černá;
                Error := I[x,y] - černá
          end
          else
          begin Pixel(x,y) := bílá;
                Error := I[x,y] - bílá
          end;
          Display Pixel(x,y);
          if x < xmax then I[x+1,y] := I[x+1,y] + 3*Error/8;
          if y > ymin then I[x,y-1] := I[x,y-1] + 3*Error/8;
          if (x < xmax) and (y > ymin)
              then I[x+1,y-1] := I[x+1,y-1] + Error/4
       end
end
```

Algoritmus 2

Vlivem distribuce chyby je výsledný obraz složen z teček, uspořádaných převážně diagonálně. Jsou-li jednotlivé tečky dostí malé, oko jejich jas integruje, takže výsledný vjem je velmi dobrý. Obraz zpracovaný uvedeným algoritmem vykazuje lepší zobrazení v detailech oproti jednoduchému použití prahové funkce při zobrazování.

Hlavním nedostatkem metody je její sériovost - zpracování současného bodu závisí na výsledcích předchozího postupu. Nevýhodou je také potřeba operací v pohyblivé řádové čárce pro násobení váhovými koeficienty. Dalším nedostatkem je občasný vznik "duchů", tj. stínů v okolí velmi světlých nebo velmi tmavých oblastí. Tento jev lze zčásti eliminovat výběrem takových váhových koeficientů distribuce chyby, jejichž suma je menší než jedna. Jinou možností je nahradit původní hodnoty intenzity I[x,y] hodnotami $0{,}1\ I_{max} + 0{,}8\ I[x,y]$, kde I_{max} je maximální možná hodnota intenzity v obrazu. Tato úprava většinou dává dobré výsledky, neboť oko je citlivější na kontrast než na absolutní úroveň jasu. Předpokladem jejího použití je však obraz, v němž spektrum intenzit není ochuzeno o mezní hodnoty. Pro jeho posouzení je vhodné zkonstruovat histogram (tj. závislost četnosti na hodnotě intenzity) a sledovat, zda jsou nejmenší a největší hodnoty intenzity dostatečně zastoupeny.

Přes uvedené nevýhody dává Floyd - Steinbergův algoritmus ve většině případů velmi dobré výsledky.

Dithering

Tato metoda je založena na principu vložení náhodných chyb do obrazu na základě porovnání prahové funkce a hodnoty intenzity pixelu. Přidávání zcela náhodných chyb nedává dobré výsledky, existuje však uspokojivý adaptivní vzor. Rozhodnutí, zda aktivovat daný pixel, je závislé nejen na hodnotě příslušné intenzity I[x,y], ale též na matici $^{(n)}D$ rozměru n x n, která obsahuje hodnoty od 0 do $n^2 - 1$. Je-li [x,y] pozice pixelu, pak po určení odpovídajícího prvku ve vzoru o souřadnicích i,j, kde

$j = x \bmod n + 1$ $\qquad$ $i = y \bmod n + 1$

je daný pixel aktivován jen tehdy, je-li

$$^{(n)}D_{i,j} < I[x,y]$$

Nejmenší vzor je podle [3] reprezentován maticí 2 x 2, a to:

$$^{(2)}D = \begin{bmatrix} 0 & 2 \\ 3 & 1 \end{bmatrix}$$

Větší vzory mohou pak být získány pomocí rekurentních vztahů takto:

$$^{(2n)}D = \begin{bmatrix} 4 \cdot {}^{(n)}U & 4 \cdot {}^{(n)}D + 2 \cdot {}^{(n)}U \\ 4 \cdot {}^{(n)}D + 3 \cdot {}^{(n)}U & 4 \cdot {}^{(n)}D + {}^{(n)}U \end{bmatrix}$$

kde

$$^{(n)}U = \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & & \vdots \\ 1 & \cdots & 1 \end{bmatrix}$$

a n je rozměr matice. Pro n = 4 pak dostáváme:

$$^{(n)}D = \begin{bmatrix} 0 & 8 & 2 & 10 \\ 12 & 4 & 14 & 6 \\ 3 & 11 & 1 & 9 \\ 15 & 7 & 13 & 5 \end{bmatrix}$$

Celý postup je znázorněn algoritmem 3 :

```
const xmin = 0; xmax = 1023; { rozlišení ve směru x }
      ymin = 0; ymax = 1023; { rozlišení ve směru y }
      n = 4; { velikost matice }
var x, y, i, j: integer;
    D: array [ 1..n , 1..n ] of integer;
begin READ(D);{ přečti matici vzoru nebo ji urči }
    for y := ymax downto ymin do
        for x := xmin to xmax do
        begin { určení pozice (i,j) v matici D }
              j := x mod n + 1; i := y mod n + 1;
              if D[i,j] < I[x,y] then Pixel(x,y) := bílá
                                 else Pixel(x,y) := černá;
              display pixel(x,y)
        end
end;
```

Algoritmus 3

Je zřejmé, že pro správnou funkci algoritmu 3 je třeba, aby se intenzita pohybovala v intervalu 0 až n^2. Toho docílíme sdružením intenzit do n^2+1 tříd. Pro volbu mezí jednotlivých tříd je opět možné využít váženého průměru.

Metoda vytváří ve výsledném obraze pravidelnou texturu, která je pro oko obvykle poněkud jednotvárná. Pro přijatelné výsledky je nutno volit n nejméně 4. Další nevýhodou je, že tento algoritmus má tendenci původní ostré kontrasty poněkud rozmazávat, což výslednému vjemu neprospívá.

Velkou výhodou je však jednoduchost metody, neboť se obejde bez operací v pohyblivé řádové čárce, a především její paralelnost - je-li k dispozici více procesorů, lze zpracovávat více bodů najednou. Výsledek v bodě [x,y] totiž závisí na intenzitě tohoto bodu, na jeho poloze a na hodnotě odpovídajícího prvku matice. Protože není závislý na ostatních bodech obrazu, pořadí zpracování bodů může být libovolné.

Algoritmus dithering předpokládá pouze 2 výstupní odstíny barvy, obvykle černou a bílou, lze však navrhnout jeho úpravu pro k > 1 (tj. pro výstupní zařízení, kde je dostupná šedá škála). Pro 4 odstíny šedi na grafickém monitoru s adaptérem EGA byl algoritmus modifikován tak, že interval možných intenzit v obraze se rozdělí na tři menší. V každém z nich jsou možné hodnoty intenzity sdruženy do n^2 tříd a nezávisle na ostatních intervalech aplikován dithering. V každém intervalu však k reprezentaci obrazových elementů slouží jiná dvojice odstínů šedi - černá - tmavošedá, tmavošedá - světlešedá a světlešedá - bílá. Tímto způsobem je možné reprezentovat $3n^2+1$ odstínů šedi, zatímco klasický dvoubarevný dithering s maticí téhož řádu poskytuje reprezentaci pouze pro n^2+1 odstínů.

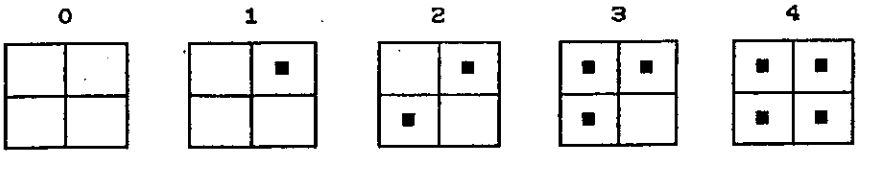
Poznamenejme, že rozdělení intenzit na n^2+1 tříd v jednom intervalu u popsané modifikace ditheringu není vhodné, neboť sousední třídy patřící do různých intervalů intenzit by byly reprezentovány plochou stejné nebo velmi podobné barvy.

Při použití uvedené metody lze i s nízkým řádem matice D dosáhnout zajímavých výsledků za cenu větší složitosti určování čísla třídy a odpovídající dvojice odstínů šedi.

Použití vzorků

Základní myšlenkou je pro reprezentaci šedé škály užít vzorků složených z černých a bílých bodů. Obsahuje-li použitý vzor n bodů, pak pro zobrazení intenzity i, $0 \le i \le n$, bude i bodů mít bílou a n-i bodů černou barvu.

Např. pro 5 úrovní šedi je zapotřebí použít vzor 2 x 2 bodů, tj.:



Vzory typu:

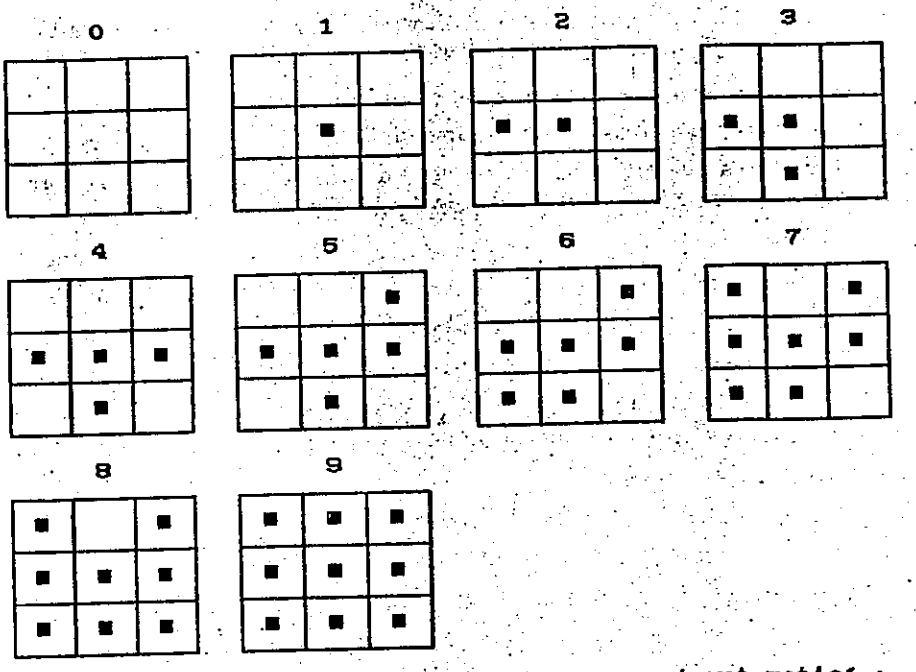
nemohou být použity, neboť vytvářejí nežádoucí vodorovné nebo svislé čáry.

Je-li vzor konstruován tak, že s růstem i o 1 přibude vždy do vzorku další aktivovaný pixel a pixely z předchozích úrovní již zůstávají beze

změny, lze vzor vyjádřit pomocí matice, v níž v pozici [i,j] je číslo úrovně, při níž je stejnolehlý pixel ve vzorku poprvé aktivován. Např. pro vzor 3 x 3 má matice tento tvar:

$$(3)_T = \begin{bmatrix} 7 & 9 & 5 \\ 2 & 1 & 4 \\ 6 & 3 & 8 \end{bmatrix}$$

Pro danou intenzitu se aktivují všechny pixely vzoru, jejichž hodnota v matici je menší nebo rovna požadované intenzitě, a tedy pro danou matici dostaneme vzory:



Dříve uvedený vzor 2 x 2 bod lze pak reprezentovat maticí :

$$(2)_T = \begin{bmatrix} 3 & 1 \\ 2 & 4 \end{bmatrix}$$

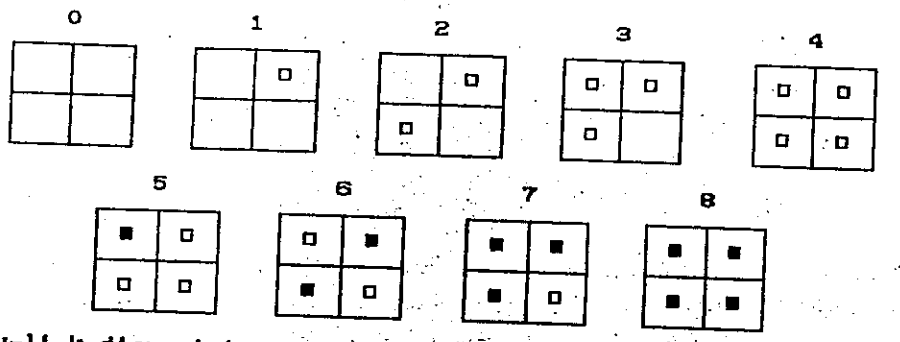
Pro více úrovní je někdy výhodný vzor 4 x 4 bodů, jehož matice je definována:

$$(4)_T = \begin{bmatrix} 1 & 9 & 3 & 15 \\ 13 & 5 & 14 & 7 \\ 4 & 10 & 2 & 12 \\ 16 & 8 & 11 & 6 \end{bmatrix}$$

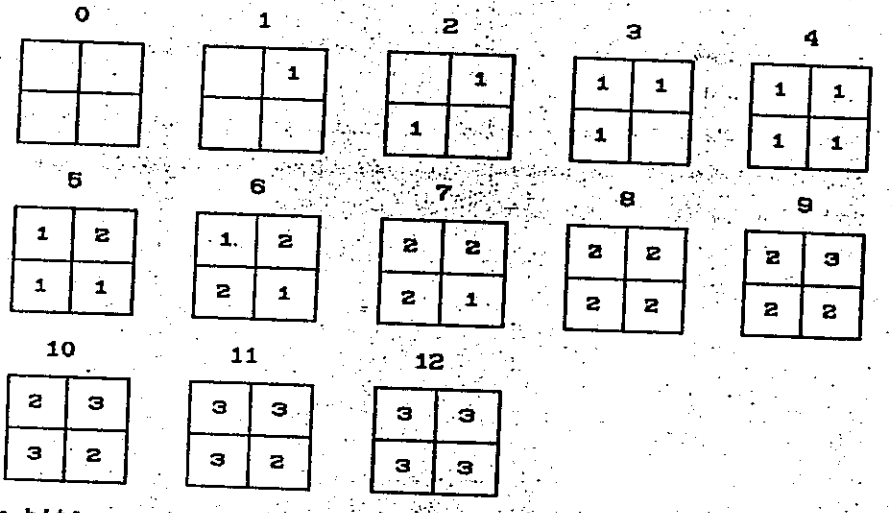
V některých případech je nutné nebo vhodné použít vzory, které jsou reprezentovány obdélníkovou maticí, např. pro 7 úrovní šedi je vzor definován maticí:

$$(3,2)_T = \begin{bmatrix} 4 & 1 & 5 \\ 6 & 3 & 2 \end{bmatrix}$$

V případě některých zařízení je možné zobrazit různě velké body (□ malý, ■ velký bod). Pak lze použít např. vzor 2 x 2 bodů k reprezentaci 9 úrovní šedi:



Jsou-li k dispozici např. 2 bity na pixel (lze případně realizovat i přetiskem), pak vzory mohou mít tvar (čísla označují intenzity - nula se neuvádí):



Více bitů na pixel nebo větší vzory vedou k odpovídajícímu zvýšení dostupných úrovní intenzit.

Použití vzorů opět předpokládá sdružení intenzit v obraze do tříd, pokud jejich rozsah a počet hodnot neodpovídá použitému rozsahu a počtu bodů vzoru.

Výhodou techniky vzorů je opět její jednoduchost, za niž však je nutné zaplatit poklesem rozlišení. Každý bod v původním obraze je totiž nahrazen vzorkem o délce hrany minimálně 2 pixely. Metoda je proto vhodná spíše pro velmi malé obrázky nebo výřezy, kde poslouží zároveň ke zvětšení. Pokud však původní rozlišení není dostatečně jemné, zvětšení bodů působí dosti rušivě.

Zvýraznění hran

[illegible] dávají nejlepší výsledky Floyd - Steinbergův algoritmus, někdy také dithering. Obě však mají tendenci obraz poněkud rozmazávat. Jarvis a Roberts ([4], [3]) objevili, že výsledný obraz bude podstatně lepší, pokud v původním obrázku předem zvýrazníme hrany. Je-li I[x,y] hodnota intenzity v bodě [x,y], pak tuto hodnotu nahradíme intenzitou

$$I'[x,y] = \frac{I[x,y] - \alpha\, I_p[x,y]}{1 - \alpha} \qquad (1)$$

kde $I_p[x,y]$ je průměrná hodnota v okolí bodu [x,y] o velikosti 3 x 3,

$$I_p[x,y] = \frac{1}{9} \sum_{u=x-1}^{x+1} \sum_{v=y-1}^{y+1} I[u,v] \qquad (2)$$

Průměrná intenzita v obrazu tedy zůstává zachována, avšak volbou $\alpha > 0$ vzrůstá rozdíl jasu bodu [x,y] od jeho sousedů.

Např. volbou $\alpha = 0.9$ a dosazením (2) do (1) dostáváme :

$$I'[x,y] = 9\, I[x,y] - \sum_{0 < (u-x)^2+(v-y)^2 < 3} I[u,v]$$

Vztahy (1) a (2) uvedené v [5] nejsou ve stávající podobě vyhovující, protože značně rozšiřují původní interval možných hodnot intenzit. Protože ve většině případů je třeba meze intenzit zachovat, lze doporučit následující úpravu : z I'[x,y] určené ze vztahů (1) a (2) se vypočte hodnota I''[x,y],

$$I''[x,y] = \min (I_{max}, \max (0, I'[x,y])) \quad , \qquad (3)$$

kde I_{max}, resp. 0, je maximální, resp. minimální možná hodnota intenzity v původním obraze.

Zdůraznění hran vyžaduje přepočet celého obrazu, což především v případě obrazových dat uložených ve vnější paměti vede ke značnému zdržení. Výsledky jsou však více než překvapující, jak je vidět i z obr.2.

Další techniky tohoto druhu lze nalézt v literatuře týkající se zpracování obrazu, např. [2], [6].

3. ZÁVĚR

Uvedené algoritmy mohou přispět ke zlepšení kvality grafického výstupu digitalizovaného obrazu, některé z nich, pokud nejsou sériového charakteru, lze použít i pro zlepšení kvality obrazu v generativní grafice.

Algoritmy byly implementovány v jazyce Turbo Pascal v.5.0 na mikropočítačích řady AT s monitorem EGA.

Žádnou z uvedených metod nelze prohlásit jednoznačně za nejlepší, neboť výsledky se výrazně liší podle charakteristik výstupního zařízení, digitalizovaného obrazu apod., avšak u všech algoritmů byly uvedeny ty jejich vlastnosti, které jsou na výstupním zařízení nezávislé a mohly by napomoci při volbě konkrétní metody.

4. LITERATURA

[1] F l o y d, R. W., S t e i n b e r g, L. : An Adaptive Algorithm for Spatial Grey Scale. SID 75 Digest. Society for Information Display, 1975, s. 36-37.

[2] G o n z a l e s, R. G., W i n t z, P. : Digital Image Processing. Addison Wesley, 1977.

[3] J a r v i s, J. F., J u d i c e, C. N., N i n k e, W. H. : A Survey of Techniques for the Display of Continuous Tone Pictures on Bilevel Displays. Comput. Graph. Image Process. 5, 1976, s. 13-40.

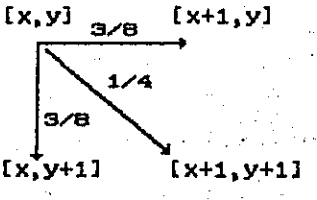
[4] J a r v i s, J. F., R o b e r t s, C. S. : A New Technique for Displaying Continuous Tone Images on a Bilevel Display. IEEE Trans. Commun. COM-24, 1976, s. 891-898.

[5] K n u t h, D. E. : Digital Halftones by Dot Diffusion. ACM Transaction

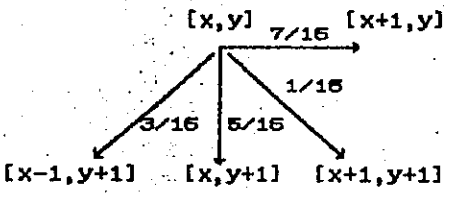
on Graphics, Vol.6, Num.4, s. 245-273.

[6] P a v l i d i s, T., : Algorithms for Graphics and Image Processing, Springer-Verlag Berlin Heidelberg, 1982.

[7] S k a l a, V. : Počítačová grafika II. VŠSE, Plzeň 1990.

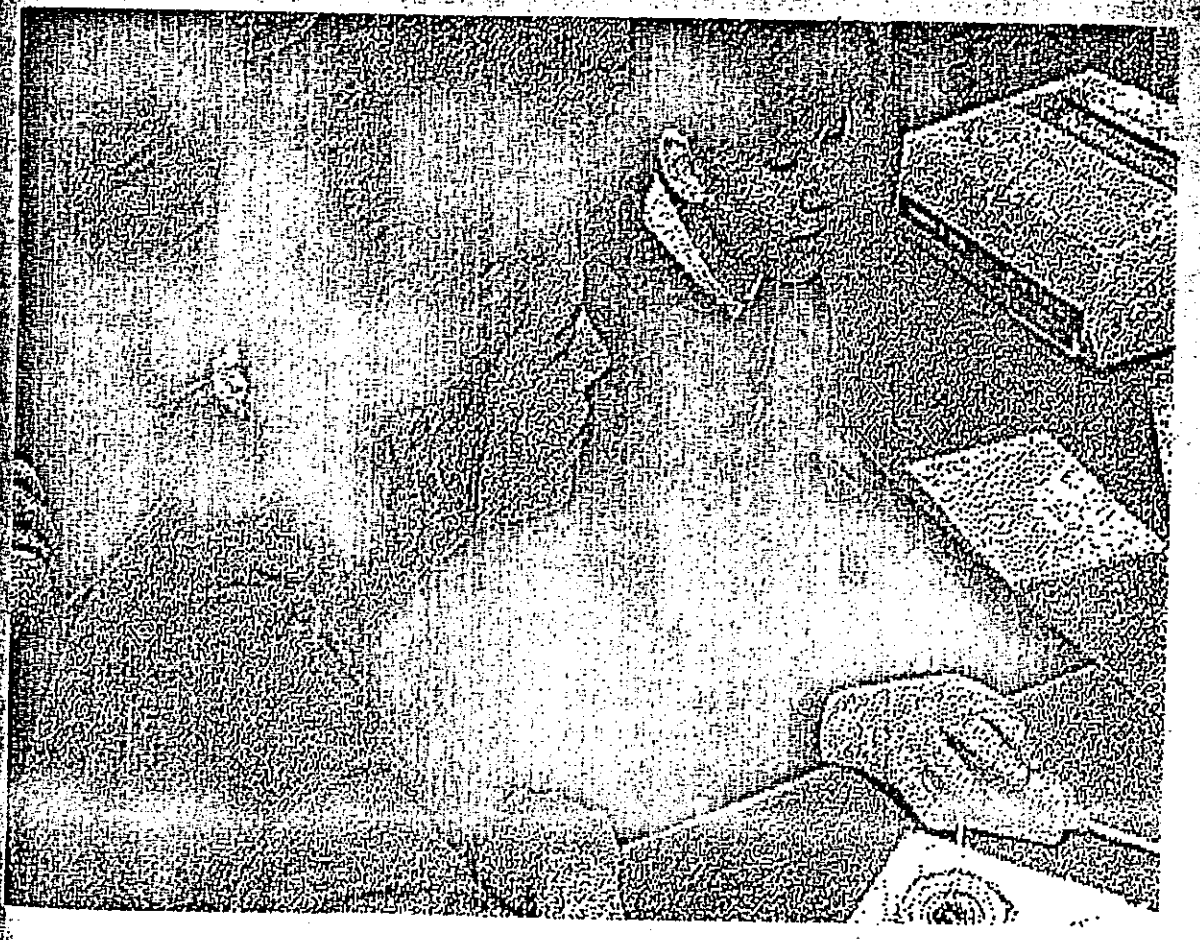


Obr.1.1



Obr.1.2

Schémata distribuce chyby ve Floyd-Steinbergově algoritmu



Obr.2

Digitalizovaný obraz s 16 úrovněmi jasu, se zvýrazněnými hranami a sníženým kontrastem, zobrazený Floyd-Steinbergovým algoritmem

bstract

The article is concerning methods which are suitable for displaying the cenes with more gray levels than the available graphical device can support. hese special techniques are called halftoning and they are used especially lth scanners and laser printers.

length parametrization of the circle. For the straight line k = 0 and therefore n = 1. That is exactly what we need in this case.

In addition to that we have to use a fast algorithm for finding the closest neigbour to the given point. If we use O(m) memory and O(m log m) time for preprocessing this can be done in optimal O(log m) time provided m is the number of points in the plane (see Theorem 5.17 in [Preparata, Shamos 1985]).

Another method for solving this problem could be based on sending a horizontal ray from the given point and computing all intersections with the curved boundary. This is possible but numerically unstable because the intersection points can be unequally spaced over the parameter interval and moreover it would require more than one iteration for each curve on the boundary. Such an algorithm is not robust enough since it is easy to miss one intersection. Notice we are calculating only one such iteration. On the other hand we are looking for the "best" initial guess for this iteration from among the number of test values chosen based on the curve intrinsic properties such as arc-length and curvature.

References

Dahlquist, G. and Björck, A. (1974) Numerical Methods, Prentice-Hall, Inc., Englewood Cliff, N.J.

Faux, I.D. and Pratt, M.J. (1980) Computational Geometry for Design and Manufacture, John Wiley and Sons, New York.

Mortenson, M.E. (1985) Geometric Modeling, John Wiley and Sons, New York.

Preparata, F.P. and Shamos, M.I. (1985) Computational Geometry, Springer-Verlag, New York.

An Efficient Algorithm for Line Clipping

Václav Skala
Dept. of Informatics and Computer Science
University of West Bohemia
Americká 42, Box 314, 306 14 Plzeň
Czech Republic

e-mail: skala@kiv.zcu.cs

Abstract

A new line clipping algorithm against convex window based on a new approach for intersection detection is presented. The main advantage of the presented algorithm is the substantial acceleration of the line clipping problem solution and that edges can be oriented clockwise or anti-clockwise.

1. Introduction

Many algorithms for clipping lines against convex or non-convex windows have been published with many modifications derived from well known Cohen-Sutherland's, Liang-Barsky's and Cyrus-Beck's algorithms, [1] - [38].

The proposed algorithm is based on a simple rule:

Make tests first and then compute.

The known algorithms for clipping lines against general convex window do not make tests similar to Cohen-Sutherland's clipping algorithm. The main reason seems to be the computational cost of such a test for convex window. If clipping algorithm is to be effective it is necessary to distinguish cases where lines pass through a given window from those where lines do not intersect the window. Cyrus-Beck's (C-B) algorithm solves this problem by direct computation of points of intersections.

Computer Graphics '95 | Inf. Conference | Budmerice | Slovakia | June 2-4, 9-

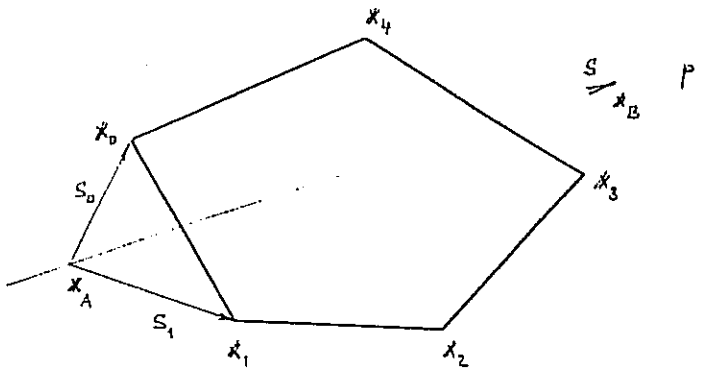


Figure 1

2. Proposed algorithm

It is obvious that the line p intersects the edge x_0x_1 of the given polygon if and only if the vector s lies between two vectors s_0 and s_1, see fig.1.

Let us define ξ and η as the z-coordinate of the cross products as follows

$$\xi = [\, s \times s_1 \,]_z \qquad \eta = [\, s \times s_{i+1} \,]_z \qquad i = 0,\ldots,N-1$$

where N is a number of edges.

It is possible to show that the line p given by the end-point x_A and by the vector s intersects the edge x_ix_{i+1} if and only if the expressions

$$(\, \xi > 0 \,) \text{ xor } (\, \eta < 0 \,) \quad , \text{ resp.} \qquad \xi * \eta < 0$$

have value true, see tab.1 and fig.1. The above mentioned expressions are independent on polygon orientation. It means that edges can be clockwise or anti-clockwise oriented.

Now, it is possible to specify a new algorithm which is based on the presumption that a line can intersect a convex polygon only in two points, see alg.1, if special cases are not considered. Special cases should be handled carefully, [33], [39].

ξ	η	intersection?	ξ	η	intersection?
>0	>0	no	=0	<0	yes
>0	=0	yes	<0	>0	yes
>0	<0	yes	<0	=0	yes
=0	>0	yes	<0	<0	no
=0	=0	no			

Table 1

In comparison with C-B algorithm the proposed algorithm only detects whether the given line intersects the given polygon and then computes intersection points, while C-B algorithm does compute all possible intersection points with all edges.

```
procedure Clip 2D ( x_A , x_B );
begin { !!!! all vectors s_i are precomputed !!!! }
  k := 0; i := N - 1; j := 0;
  s := x_B - x_A;   ξ := [ s x s_i ]_z;
  while (j < N) and (k < 2) do
  begin η := [ s x s_j ]_z;
    if (ξ * η) < 0.0 then { intersection exists }
    begin
      index_k := i;     { save edge index having intersection }
      k := k + 1;       ξ := η;
    end
    else if ( ξ * η )= 0.0 then Special cases { ξ=0 or η=0 }
                                { cases i - m }
                           else Intersection does not exist;
                                { cases e - h }
    i := j;     j := j + 1
  end;
  if k = 0 then { intersection does not exist } EXIT;
  { k = 2  intersections exist - edges saved in index_k }
  t_min := -∞; t_max := ∞; {for line segments t_min := 0; t_max := 1;
```

```
    i := index1;   t1 := det[ x1 - xA | - ŝi ] / det[ s | - ŝi ];
    { for effective implementation use identity x1 - xA = si }
    i := index2;   t2 := det[ x1 - xA | - ŝi ] / det[ s | - ŝi ];
    { recompute end-points if changed }
    if t1 < t2 then  { swap t1 <-> t2 values ? }
         begin if t2 < tmax then xB := xA + s t2;
               if t1 > tmin then xA := xA + s t1
         end { t2 < tmax  t1 < tmin can be left out for lines }
    else begin if t1 > tmin then xB := xA + s t1;
               if t2 < tmax then xA := xA + s t2
         end;
    SHOW LINE( xA , xB );
end
```

Algorithm 1

3. Experimental results

The C-B and proposed algorithms have been tested on data sets of end-points which have been randomly and uniformly generated over a space inside of a circle in order to eliminate an influence of rotation. Convex polygons were generated as *regular N sided convex polygons inscribed into a smaller circle*. The following results have been obtained if 80% of the lines intersect the window.

Experimental results for PC 386DX/387

N	5	8	10	15	20	30	70	190
ν	1.41	1.54	1.73	1.81	1.90	2.08	2.27	2.41

Table 2

The coefficients ν are in a good correlation with theoretical results because the pessimistic estimations have been taken for the theoretical analysis. It is necessary to point out that different tests might be used instead of shown cross product, e.g. test for detection on which side of the given line p the point x_i lies. If a separation function

$$F(x,y) = A\,x + B\,y + C$$

is used ($F(x,y) = 0$ is an equation for the clipped line), the algorithm is about 10% faster than algorithm alg.1.

4. Conclusion

The new efficient algorithm for clipping lines against convex window has been developed. Edges of the given polygon can be arbitrarily oriented. All tests were implemented in C++ on a PC 386/387 25 MHz and PC 486 33 MHz. It can be expected that for workstations the efficiency ν will be higher than for PC 486.

The proposed algorithm can be easily modified for clipping lines against non-convex windows, similarly to [31]. This paper is a shorten version of [33].

5. Acknowledgments

The author would like to express his thanks to students of Computer Graphics and CAD Systems who stimulated this work, especially Mr.P.Blaha for careful tests implementation of the proposed algorithm.

6. References

[1] Akeley,K., Korobkin,C.P.: Efficient Graphics Processor for Clipping Polygons, US Patent No.5 061 737, 1991.

[2] Andreev,R.D.: Algorithm for Clipping Arbitrary Polygons, Computer Graphics Forum, Vol.7, No.3, pp.183-192, 1988.

[3] Arokiasamy,A.: Homogeneous Coordinates and the Principle of Duality in Two Dimensional Clipping, Computers & Graphics, Vol.13, No.1, pp.99-100, 1989.

[4] Blinn,J.F.,Newell,M.E.: Clipping Using Homogeneous Coordinates, Computer Graphics (SIGGRAPH'78), Vol.12, pp.245-251, 1978.

[5] Blinn,J.F.: A Trip Down to Graphics Pipeline - Line Clipping, IEEE Computer Graphics and Applications, Vol.11, No.1, pp.98-105, 1991.

[6] Brewer,E.A., Barsky,B.A.: Clipping After Projection: An Improved Perspective Pipeline, submitted for publication.

[7] Burkert,A., Noll,S.: Fast Algorithm for Polygon Clipping with 3D Windows, Eurographics'88 Proceedings, pp.405-419, 1988.

[8] Cheng,F., Yen,Y.: A Parallel Line Clipping Algorithm and its Implementation, in CGI'89 Conference Proceedings, 1989.

[9] Cyrus,M.,Beck,J.: Generalized Two and Three Dimensional Clipping, Computers & Graphics, Vol.3, No.1, pp.23-28, 1979.

[10] Day,J.D.: A Comparisons of Line Clipping Algorithms, Queensland Univ. of Technology, School of Computing Science, Report 2-91, 1991.

[11] Day,J.D.: A New Two Dimensional Line Clipping Algorithms for Small Windows, Queensland Univ. of Technology, School of Computing Science, Report 3-91, 1991.

[12] Day,J.D.: A New Two Dimensional Line Clipping Algorithms for Small Windows, Computer Graphics Forum, Vol.11, No.4, pp.241-246, 1992.

[13] Dorr, M.: A New Approach to Parametric Line Clipping, Computers & Graphics, Vol.14. Nos.3/4, pp.449-464, 1990.

[14] Duvanenko,V.J., Robins,W.E., Gyurcsik,R.S.: Improving Line Segment Clipping, Dr.Dobb's Journal of Software Tools, Vol.15, No.7, pp.36,38,40,42,44-5,98,100, 1990.

[15] Fong,D.Y., Chu,J.: A String Pattern Recognition Approach to Polygon Clipping, Pattern Recognition, Vol.23, No.8., pp.879-892, 1992.

[16] Herman,I.,Reviczky,J.: Some Remarks on the Modelling Clip Problem, Computer Graphics Forum, Vol.7, No.4, pp.265-272, 1988.

[17] Kaijian,S., Edwards,J.A., Cooper,D.C.: An Efficient Line Clipping Algorithm, Computers & Graphics, Vol.14, No.2, pp.297-301,1990.

[18] Kilgour,A.C.: Unifying Vector and Polygon Algorithm for Scan Conversion and Clipping, TR CSC/87/R7, Univ. of Glasgow, May 1987.

[19] Kramer,G.: A Line Clipping Algorithm and its Analysis, Computer Graphics Forum (EG'92 Conference Proceedings), Vol.11, No.3, pp.C253-260, 1992.

[20] Liang,Y.D.,Barsky,B.A.: A New Concept and Method for Line Clipping, ACM Trans. on Graphics, Vol.3, No.1, pp.1-22, 1984.

[21] Liang,Y.D.,Barsky,B.A.: An Analysis and Algorithms for Polygon Clipping, CACM, Vol.26, No.11, pp.868-876, 1984.

[22] Liang,Y., Barsky, B.A.: The Optimal Tree Algorithm for Line Clipping, Technical paper distributed at Eurographics'92 Conference, Cambridge, 1992.

[23] Maillot,P.G.: A New, Fast Method For 2D Polygon Clipping: Analysis and Software Implementation, ACM Transaction on Graphics, Vol.11, No.3, pp.276-290, 1992.

[24] Nicholl,T.M.,Lee,D.T.,Nicholl,R.A.: An Efficient New Algorithm for 2D Line Clipping: Its Development and Analysis, ACM Computer Graphics, Vol.21, No.4, pp.253-262, 1987.

[25] Nielsen,H.P.: An Intersection Test Using Dual Figures, paper submitted to CGF for publication, 1992.

[26] Nielsen,H.P.: Line Clipping Using Semi-homogeneous Coordinates, paper submitted to CGF for publication, 1992.

[27] O'Bara,R.M.,Abi-Ezzi,S.: An Analysis of Modeling Clip, in EG'89 Conference Proceedings, pp.367-380, 1989.

[28] Rappaport,A: An Efficient Algorithm for Line and Polygon Clipping, The Visual Computer, Vol.7, No.1, pp.19-28, 1991.

[29] Sharma,N.C., Manohar,S.: Line Clipping Revisited: Two Efficient Algorithms based on Simple Geometric Observations, Computers & Graphics, Vol.16, No.1, pp.51-54, 1992.

[30] Skala,V.: Algorithms for 2D Line Clipping, in CGI'89 Conference Proceedings, pp.121-128, 1989.

[31] Skala,V.: Algorithms for 2D Line Clipping, in EG'89 Conference Proceedings, pp.355-367, 1989.

[32] Skala,V.: Algorithm for Line Clipping in E2 for Convex Window (in Czech), submitted for publication, Algorithms'93.

[33] Skala,V.: An Efficient Algorithm for Line Clipping by Convex Polygon, to appear in Computers & Graphics, Vol.17, No.4, 1993.

[34] Slater,M., Barsky,A.B.: 2D Line and Polygon Clipping Based on Space Subdivision, submitted for publication.

[35] Sobkow,M.S., Pospil,P., Yang,Y.-H.: A Fast Two-dimensional Line Clipping Algorithm via Line Encoding, Computers & Graphics, Vol.11, No.4, pp.469-467, 1987.

[36] Sutherland,I.E.,Hodgman,G.W.: Reentrant Polygon Clipping, CACM, Vol.17, No.1, pp.32-42, 1974.

[37] Ying,D.-N.: A New Algorithm for Polygon Clipping and Boolean Operations, Univ. of Zhejiang, Hangzhou, China.

[38] Yong-Kui,L.: A New Algorithm for Line Clipping by Convex Polygon, paper submitted for publication, 1991.

[39] Edelsbrunner,H.,Mucke,E.P.: Simulation of Simplicity: A Technique to Cope with Degeneration Cases in Geometric Algorithms, ACM Trans. on Graphics, Vol.9, No.1, pp. 66-104, 1990.

The Visual Potential and its Computation

Eduard Sojka
Department of Computer Science, Technical University of Ostrava
17.listopadu, 708 00 Ostrava-Poruba
Czech Republic

Abstract

This paper deals with the computation of the visual potential of 3-D and 2-D scenes. The scene can contain an arbitrary number of convex or concave objects bounded by planar faces. In the paper, the necessary notions are defined. The equations of the visual event region are derived. An algorithm for the computation of the visual potential is presented. The algorithm is implemented for 2-D/2.5-D scenes in C++ on a personal computer. The possibilities of the practical use of the visual potential are outlined.

Keywords: *Visual potential, aspect graph, visibility.*

1. Introduction

The visual potential contains information about the visual experience which an observer undergoes by looking at the object when traversing in space. The observer traverses all points in space step by step with the exception of points of the objects. Thus, an infinite number of different images can be obtained. If the relation of topological equivalence on a set of images is introduced, then the infinite set of images is decomposed into the finite number of classes, each of which being represented by certain image. Also the space surrounding the objects of the scene is decomposed into the discrete cells. The images perceived by an observer from every point of some particular cell are topologically equivalent. When the observer moves across the border from one such cell to another, a visual event is said to have occurred. The visual potential contains information about the classes of images, corresponding cells, and visual events for given scene. The visual potential can be represented as a graph, the nodes and edges of which correspond to classes of images and to visual events, respectively.

Fig. 1 shows a prism with the triangular base in the plane xy. Let us investigate what will be the perception of an observer due to the location of the viewpoint. For the sake of simplicity, we assume in this example that the observer moves only in the plane π: $z=z_0$, $0<z_0<h$ (see Fig. 1). We shall find out that six aspects are perceived by an observer. All the aspects are depicted in Fig. 1b and denoted $A_1,\ldots, A_6$. Furthermore, we can also distinguish between two groups of similar aspects $\mathcal{A}_1=\{A_1,A_3,A_5\}$ and $\mathcal{A}_2=\{A_2,A_4,A_6\}$. The aspect A_i is perceived when the viewpoint is located inside the region A^r_i. Aspect regions A^r_1 - A^r_6 are depicted in Fig. 1c. When the observer moves from one aspect region to another, the sudden change, called the visual event, is

konference Algoritmy '91, JSMaF pří SAV,
15. - 19. 4. 1991, Štrbské Pleso, 1991

Algoritmy zobecněného ořezávání

Václav Skala
Katedra informatiky a výpočetní techniky, VŠSE, Plzeň

1. Úvod

Velmi důležitou částí každého programového vybavení pro počítačovou grafiku je ořezávání těch částí scény, které jsou mimo zobrazovanou oblast. Nejjednodušším případem je ořezávání v dvourozměrném prostoru vůči obdélníku zobrazovací plochy. Tento případ je v praxi nejčastější, zejména při zobrazování plošných objektů. Nicméně v technické praxi jsou zapotřebí i operace ořezávání nekonvexním n-úhelníkem či oblastí, tvořenou kruhovými oblouky.

2. Algoritmy zobecněného ořezávání

Algoritmy pro ořezávání nekonvexními n-úhelníky byly publikovány v dostupné literatuře [11], avšak uspokojivě neřešily problematiku singulárních případů, když ořezávaná úsečka či přímka prochází vrcholem nebo se jej dotýká nebo když hrana n-úhelníka leží na ořezávané přímce. S rozšířením aplikací počítačové grafiky vyvstává stále častěji požadavek na ořezávání úseček oblastmi, jejichž hranice jsou tvořeny nejen úsečkami, ale i kruhovými oblouky nebo částmi jiných kuželoseček. Oblasti samotné pak mohou obsahovat i díry. Předkládané algoritmy pro ořezávání nekonvexním n-úhelníkem či oblastí jsou založeny na parametrickém vyjádření úseček.

Předpokládejme, že hrany nekonvexního n-úhelníka jsou opět dány ve směru anebo proti směru hodinových ručiček, a že se vzájemně neprotínají. Pro jednoduchost předpokládejme, že pouze hrany sousední mají společný bod, sousední hrany neleží na společné přímce a vrcholy jsou navzájem různé. Uvedený algoritmus lze modifikovat i pro n-úhelníky nesplňující výše uvedené předpoklady.

Označme $x(q)$ souřadnice bodů ořezávané přímky $v(q)$ a vyjádřeme je parametricky jako

$$x(q) = x_r + (x_s - x_r) . q \qquad q \in (-\infty , \infty)$$

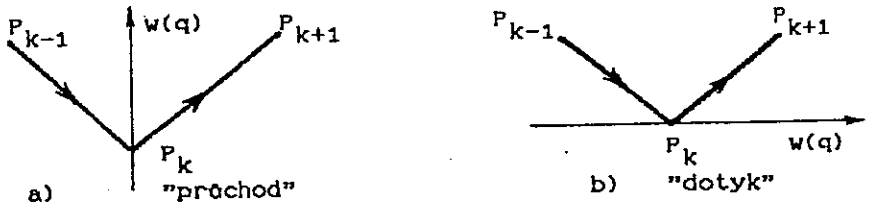
a souřadnice bodů $x(p)$ každé hrany n-úhelníka jako

$$x(p) = x_i + (x_{i+1} - x_i) . p \qquad p \in \langle 0 , 1 \rangle \qquad i = 0,\ldots,n-1$$

Budeme zatím hledat průsečíky hran n-úhelníka s přímkou $v(q)$, na které leží ořezávaná úsečka P_rP_s. Kromě průsečíku přímky s hranou musíme rozlišit případy, kdy hrana n-úhelníka leží na přímce $v(q)$ a kdy přímka $v(q)$ prochází

vrcholem. Při průchodu přímky w(q) vrcholem mohou nastat dva případy, viz obr.1. V případě ad a) se generuje pouze jeden průsečík, zatímco v případě ad b) se generují dva totožné průsečíky. V obou případech se průsečíky považují z hlediska dalšího zpracování za průsečíky s hranou. Ve skutečnosti se generují pouze hodnoty parametru q, které odpovídají poloze bodu P_k na přímce w(q).

$[s_1 \times s_2]_z . [s_3 \times s_2]_z > 0$ $\quad$ $[s_1 \times s_2]_z . [s_3 \times s_2]_z < 0$



kde + ,resp - v indexech značí součet, resp. rozdíl modulo n

a $s_1 = x_k - x_{k-1}$ $\quad s_2 = x_s - x_r$ $\quad s_3 = x_{k+1} - x_k$

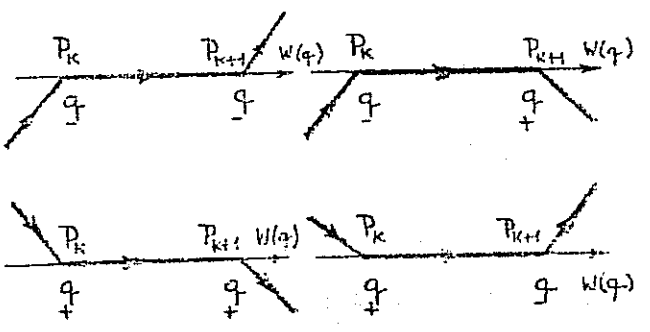
Obr.1

V případě, že na přímce w(q) leží některá hrana, mohou nastat situace, které jsou znázorněny na obr.2. V těchto případech není možné ihned rozhodnout, jaké hodnoty parametru q mají být generovány. Z tohoto důvodu musí být generován speciální **atribut** parametru q, který je určen znaménkem souřadnice z vektorového součinu vektorů s_1 a s_2, resp. s_3 a s_2. Průsečík bude určen nejen hodnotou q, ale též hodnotou atributu, který je dán jako:

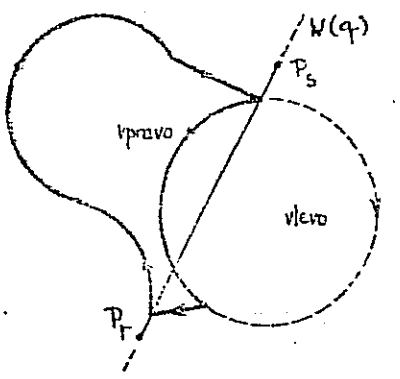
_ (mezera) pro průsečík s hranou

\+ nebo - znaménko souřadnice z vektorového součinu $[s_1 \times s_2]$, resp. $[s_3 \times s_2]$ pro "dotyk" nebo "průchod" vrcholem.

Po nalezení všech průsečíků, včetně určení jejich atributů, musí být získaná množina hodnot q setříděna spolu s atributy. V následujícím kroku je nutné provést redukci získaných hodnot q podle tab.1; úplná tabulka včetně případů pro obecnější předpoklady viz [6]. Výsledkem je pak množina dvojic hodnot q, které určují úseky **přímky** w(q) ležící uvnitř n-úhelníka. Pro určení úseků **úsečky** P_rP_s, které leží uvnitř n-úhelníka, je nutné vyhodnotit **průnik** intervalu <0,1> s jednotlivými intervaly, které jsou dány po sobě jdoucími dvojicemi hodnot q. Celý postup může být realizován označenými částmi algoritmu 2.



Obr.2



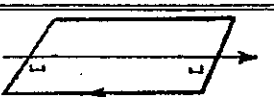
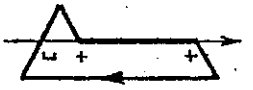
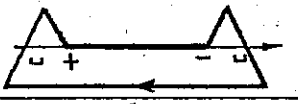
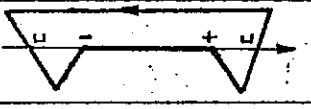
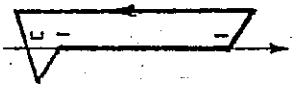
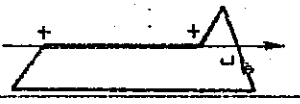
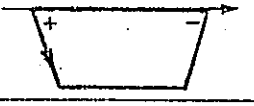
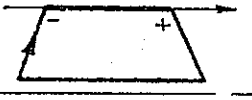
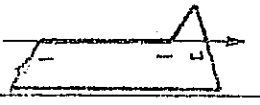
Obr.3

Příkaz pro výběr podintervalů může být realizován např. algoritmem 1.

```
i:=1;
while i ≤ počet průsečíků - 1 do
begin if max(0,q_i)≤min(1,q_{i+1}) then uloz(max(0,q_i),min(1,q_{i+1}));
      i:=i+2
end;
```

Algoritmus 1

Až dosud v literatuře publikované algoritmy umožňovaly orezávání úseček či přímek vzhledem k n-úhelníkům, tj. vzhledem k oblastem, jejichž hranice byly tvořeny úsečkami. Nicméně existuje poměrně široká třída úloh, kde je vhodné orezávat úsečky vůči oblasti s hranicemi tvořenými oblouky.

atributy q_i	q_{i+1}	q_{i+2}	situace	činnost
␣	␣	*		uloz(q_i,q_{i+1}); i:=i+2
␣	+	+		uloz(q_i,q_{i+2}); i:=i+3
␣	+	−		uloz(q_i,q_{i+2}); i:=i+2 změň atribut q_i na ␣
␣	−	+		uloz(q_i,q_{i+2}); i:=i+2 změň atribut q_i na ␣
␣	−	−		uloz(q_i,q_{i+2}); i:=i+3
+	+	*		uloz(q_i,q_{i+1}); i:=i+1 změň atribut q_i na ␣
+	−	*		uloz(q_i,q_{i+1}); i:=i+2
−	+	*		uloz(q_i,q_{i+1}); i:=i+2
−	−	*		uloz(q_i,q_{i+1}); i:=i+1 změň atribut q_i na ␣

* znamená všechny případy, tj. ␣ + − .

Tabulka 1

Předpokládejme nyní, že oblast je dána posloupností vrcholů ve směru nebo proti směru hodinových ručiček. Není-li hrana lineární, pak kromě poloměru a pozice středu oblouku je dána i informace o tom, která část kružnice (pravá nebo levá vzhledem ke spojnici počátečního bodu kruhového oblouku a jeho středu) má být vzata v úvahu. Pro zjednodušení algoritmu uvažme omezení, že všechny vrcholy mají navzájem různé souřadnice, žádný vrchol neleží na hraně nebo kruhovém oblouku, dvě hrany se nedotýkají, pokud nejsou sousední, dvě

sousední hranice oblasti, tj. hrany či oblouky, mohou mít pouze vrchol jako společný bod.

Na rozdíl od předchozího algoritmu může mít přímka $w(q)$ s kruhovým obloukem dva průsečíky, což poněkud komplikuje řešení problému. V případě kruhového oblouku musíme totiž řešit následující soustavu rovnic vzhledem k proměnné q

$$x(q) = x_r + (x_s - x_r) . q \qquad q \in (-\infty , +\infty)$$

$$(x - x_u)^2 + (y - y_u)^2 - r^2 = 0$$

kde (x_u,y_u) je střed kružnice a r je její poloměr.
Řešením obdržíme kvadratickou rovnici pro q

$$aq^2 + bq + c = 0$$

kde $a = (x_s - x_r)^2 + (y_s - y_r)^2$ $\quad c = (x_r - x_u)^2 + (y_r - y_u)^2 - r^2$

$b = 2\,[(x_r - x_u).(x_s - x_r) + (y_r - y_u).(y_s - y_r)]$

V případě, že přímka $w(q)$ danou kružnici protíná nebo se jí dotýká, obdržíme řešením rovnice obecně dva reálné kořeny.

Nyní je nezbytné určit, které průsečíky leží na části kružnice tvořící hranici dané oblasti. To lze zjistit pomocí testu, zda průsečík leží vpravo či vlevo od spojnice počátečního a koncového bodu kruhového oblouku. To znamená, že

- je-li oblouk orientován doprava, bude bod $x(q_i)$ uvažován tehdy a jen tehdy, je-li $[s_1 \times s_2]_z < 0 \quad i=1,2$
- je-li oblouk orientován doleva, bude bod $x(q_i)$ uvažován tehdy a jen tehdy, je-li $[s_1 \times s_2]_z > 0 \quad i=1,2$

přičemž $x_k \neq x(q_i)$, $s_1 = x_{k+1} - x_k$, $s_2 = x(q_i) - x_k$

Je zřejmé, že opět musí být řešeny speciální případy, kdy např. přímka $w(q)$ prochází vrcholem x_k. V těchto případech budou vyhodnoceny vektory s_1, s_3 takto:

- pro oblouk $s_1 = [\, y_k - y_u \,,\, x_u - x_k \,]^T$, kde (x_u,y_u) je střed kružnice
  pro hranu $s_1 = [\, x_k - x_{k-1} \,,\, y_k - y_{k-1} \,]^T$, tj. $s_1 = x_k - x_{k-1}$
- pro oblouk $s_3 = [\, y_k - y_w \,,\, x_w - x_k \,]^T$, kde (x_w,y_w) je střed kružnice
  pro hranu $s_3 = [\, x_k - x_{k-1} \,,\, y_k - y_{k-1} \,]^T$, tj. $s_3 = x_k - x_{k-1}$

V tabulce 2 jsou uvedena pravidla pro vyhodnocování možných situací, podrobnější ilustrace viz [5], [6]. Je-li oblouk orientován doprava, pak musí být změněno znaménko souřadnice z příslušného vektorového součinu. Pak můžeme definovat hodnoty proměnných a , b pomocí sekvencí:

a:=$[s_1 \times s_2]_z$; b:=$[s_3 \times s_2]_z$;
if $x_{k-1}x_k$ je oblouk then if a = 0 then a := $- s_1 . s_2$
else if orientace oblouku je doprava then a := -a;
if $x_{k-1}x_k$ je oblouk then if b = 0 then b := $s_3 . s_2$
else if orientace oblouku je doprava then b := -b;

Celý postup ořezávání úsečky nekonvexní oblastí je možné realizovat např. algoritmem 2.

$[s_1 \times s_2]_z$	$[s_3 \times s_2]_z$	typ dotyk/průchod
< 0	< 0	průchod
< 0	> 0	dotyk
> 0	> 0	průchod
> 0	< 0	dotyk
< 0	= 0	if $s_3.s_2>0$ then průchod else dotyk
> 0	= 0	if $s_3.s_2>0$ then dotyk else průchod
= 0	< 0	if $s_1.s_2>0$ then dotyk else průchod
= 0	> 0	if $s_1.s_2>0$ then průchod else dotyk
= 0	= 0	if $s_1.s_2>0$ xor $s_3.s_2>0$ then průchod else dotyk

Tabulka 2

```
procedure Comp ( x_A, x_B: vector; var r: real; t: boolean );
begin if x_A x_B je lineární
      then begin s := x_B - x_A;  r := [ s x s_2 ]_z end
      else begin s :=[ y_k - y_w , x_w - x_k ]^T; r := [ s x s_2 ]_z;
                 if r = 0 then if t then r := s . s_2 else r := - s . s_2
                          else if oblouk orientován doprava then r := -r
           end
end ( Comp );
{ tělo vlastního algoritmu }
k := n - 1;  i := 0;  s_2 := x_s - x_r;
while i < n do
begin
   if x_k leží na přímce v(q) then
   begin Comp ( x_k , x_i , a , true ); Comp ( x_k-1 , x_k , b , false );
         if x_k x_i je lineární then
         begin Výpočet hodnoty ( q ); { předpokládá se, že  x_k = x(q) }
            if a*b > 0 then Generuj( q s atributem ⌣ )
            else if a*b < 0 then Generuj( q, q s atributem ⌣ )
                            else if a = 0 then Generuj(q s atributem sign b)
                                          else Generuj(q s atributem sign a)
         end
         else { předpokládá se, že  x_k = x(q_1) }
         begin Výpočet hodnot ( q_1 , q_2 );
            if a*b > 0
            then Generuj( q_1 s atributem ⌣ )
            else if a*b < 0
                 then Generuj( q_1, q_1 s atributem ⌣ )
                 else if a = 0 then Generuj( q_1 s atributem sign b )
                               else Generuj( q_1 s atributem sign a );
            Generuj( q_2^* s atributem ⌣ );
         end
   end
   else if x_k x_i je lineární
        then begin Výpočet hodnoty ( q );
                   if průsečík je uvnitř <x_k,x_i) then Generuj(q  s atributem ⌣ )
             end
```

```
        else begin Výpočet hodnot ( q1 , q2 );
                   if průsečík existuje then Generuj( q1*, q2* s atributem ⌣ )
             (* značí, že průsečík leží na požadované straně oblouku xkx1 )
             end;
    k := i;  i := i + 1;
end ( while );
SORT( získané hodnoty q );  REDUKUJ ( hodnoty q podle tabulky );
VYBER ( podintervaly jako <qj,qj+1> ∩ <0,1>  ∀ j );
```

Algoritmus 2

3. Závěr

Pro skutečné použití kuželoseček jako grafických primitiv jsou nezbytné též algoritmy jejich orezávání konvexními a nekonvexními oblastmi. Popis těchto operací lze nalézt např. v [9], resp. [10]. Tyto algoritmy ve spojení s Weiler-Athertonovým algoritmem orezávání obecné nekonvexní oblasti oblastí též nekonvexní poskytují možnosti pro zavedení kuželoseček jako základních grafických primitiv. Na tomto místě je nutné zdůraznit, že aproximace kuželoseček lineárními úseky může vést k podstatnému snížení rychlosti prováděných grafických operací, zejména pak při orezávání a geometrických transformacích.

4. Literatura

[1] Earnshaw,R.A.(Ed.): Theoretical Foundations of Computer Graphics and CAD, NATO ASI Series, Series F, Vol.40, Springer Verlag, 1987.

[2] Earnshaw,R.A.,Wyvill,B.(Ed.): New Advances in Computer Graphics, Proceedings of CGI 89, Springer Verlag, 1989.

[3] Hansmann,W.,Hopgood,F.R.A.,Strasser,W.(Ed.): EUROGRAPHICS'89 Conference Proceedings, North Holland Publ. Comp., 1989.

[4] Skala,V.: Algorithms for 2D Line Clipping, in [2], 1989, pp.121-128.

[5] Skala,V.: Algorithms for 2D Line Clipping, in [3], 1989, pp.355-367.

[6] Skala,V.: Počítačová grafika I, skripta VŠSE Plzeň, 1990

[7] Skala,V.: A Unifying Approach to the Line Clipping Problem Solution, BISYCP'89, Beijing, 1989.

[8] Skala,V.: A Unifying Approach to the Line Clipping Problem Solution, SIAM Conference on Geometric Design, November 1989, TEMPE, AZ, USA.

[9] Skala,V.: General Conics Clipping - Problem Solution, YUGRAPH'90, June 20-22, 1990, Dubrovnik, Yugoslavia.

[10] Skala,V.: Algorithms for Clipping Quadratic Arcs, Computer Graphics International'90, June 27-29, 1990, Singapore.

[11] Newmann,W.M.,Sproull,R.F.: Principles of Interactive Computer Graphics, 2nd ed., McGraw Hill, 1981.

Konference
Algorithmy 85, JSMaFpř SAV - OSVS Bratislava
15.- 19.4.1985, Štrbské Pleso

RESPEKTOVÁNÍ VIDITELNOSTI - ZÁKLADNÍ GRAFICKÁ INSTRUKCE

Václav Skala
Katedra technické kybernetiky, VŠSE , Plzeň

1.Úvod

Řešení mnoha technických problémů má jako výsledek funkce dvou proměnných, které jsou buď vyjádřeny explicitním popisem nebo tabulkou funkčních hodnot. Funkce byly obvykle kresleny bez respektování viditelnosti. Podprogramy pro kreslení funkcí dvou proměnných s respektováním viditelností nebyly právě jednoduché (viz [6] - [8]), ačkoliv viditelnost může být řešena poměrně jednoduše na fyzické úrovni kreslení, pokud předpokládáme rastrové grafické výstupní zařízení. Bresenhamův algoritmus pro kreslení přímek může být jednoduše modifikován tak, abychom mohli kreslit funkce dvou proměnných s ohledem na viditelnost.

Williamson [7] řešil daný problém s tím,že neumožňuje obecnou rotaci, Watkinson [6] poukázal pouze, že pro některé kombinace natočení nebude jím předložený algoritmus správně eliminovat neviditelné části a Boutlandova metoda [1] je založena na pevném natočení. V předkládané metodě je podstatné pořadí kreslení a v [3], [4] je uvedena obecná metoda nalezení správného pořadí kreslení pro obecné natočení.

2.Úvod do problematiky

Mějme explicitně vyjádřenou funkci dvou proměnných x a y

$$z = f (x , y)$$

kde: $x \in \langle ax,bx \rangle \qquad y \in \langle ay,by \rangle$

a chtějme zobrazit tuto funkci použitím grafického rastrového displaye. Pro mnoho technických problémů je postačující ukázat chování této funkce kreslením funkčních řezů vzhledem k ose x a ose y, tj. křivky:

$$z = f (x , y_i) \quad i=1,\ldots,n$$

kde: $x \in \langle ax,bx \rangle$ a $ay=y_1 < y_2 < \ldots < y_n = by$

a křivky:

$$z = f (x , y) \quad j=1,\ldots,m$$

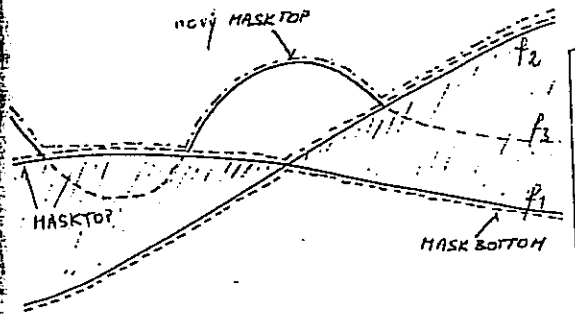
kde: $y \in \langle ay,by \rangle$ a $ax=x_1 < x < \ldots < x_m = bx$

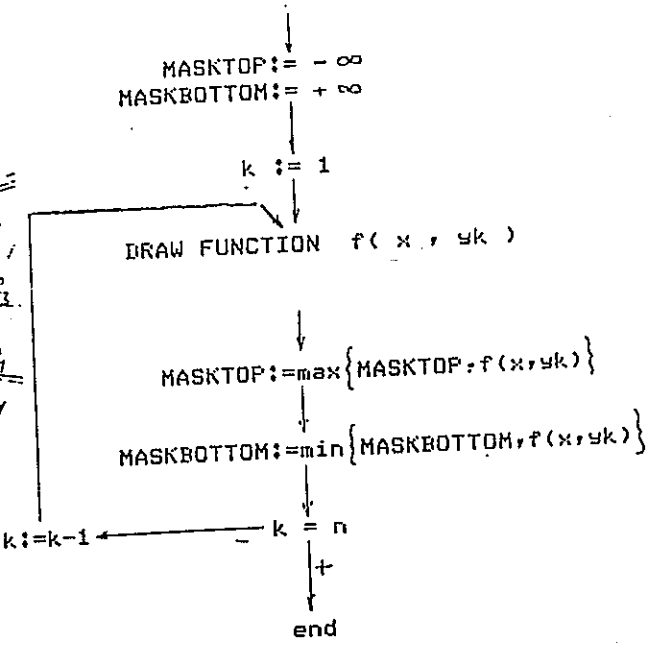
Daná funkce může být reprezentována buď specifikací funkce nebo tabulkou hodnot pro uzlové body sítě v rovině x-y. Je-li funkce složitá, je obtížné si představit chování funkce,neboť jsou kresleny i části, které jsou neviditelné při daném natočení. Úspěšně tento problém řešili Watkins[6],Williamson [7] a Boutland [1]. Princip řešení je obecně velmi jednoduchý.Jestliže nakreslíme první dva řezy rovnoběžné s osou x, pak se nám mezi těmito křivkami vytvoří

pás neviditelnosti.Předpokládejme, že budeme nejdříve kreslit křivky bližší, později pak vzdálenější. Jestliže budeme chtit kreslit třetí křivku, je zřejmé, že části které prochází timto pásem neviditelnosti jsou timto pásem zakryté a tedy neviditelné (viz. obr.1). Jestliže budeme analyzovat podrobně daný problém, vidime, že budeme potřebovat nějakým způsobem reprezentovat horni a dolni hranici pásu neviditelnosti, což může být uděláno funkcemi MASKTOP a MASKBOTTOM. Skutečnou reprezentaci těchto funkcí zatim opomineme. Nyni se problém kreslení křivek s ohledem na viditelnost stává jednoduchým (viz. algoritmus 1.),nebot budeme kreslit jen ty části řezů funkce,kdy body křivky jsou mimo pás neviditelnosti.

Problém jak reprezentovat funkce MASKTOP a MASKBOTTOM byl vyřešen Watkinsonem [6] zavedením maskovacích vektorů. V tomto případě musi být řešeny následující problémy:

- podle čeho rozhodneme, zda nastavime i-tý nebo i+1 -ní prvky maskovacích vektorů, je-li $x_i < x < x_{i+1}$
- vektory MASKTOP a MASKBOTTOM by měly být nastaveny pro všechny body křivky. To znamená, že musime použit nějaké interpolace s "vhodným" krokem interpolace.
- musíme řešit speciální připad, kdy řez funkce je paralelni s osou z.



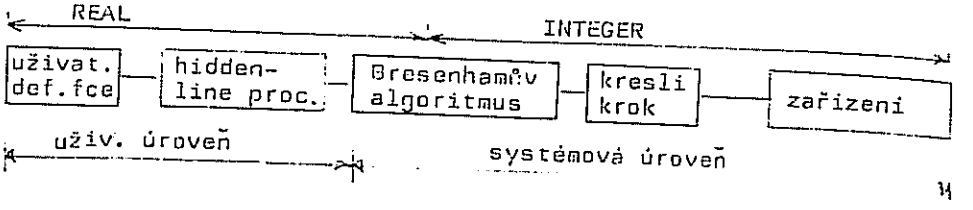


obr. .lgoritmus :.

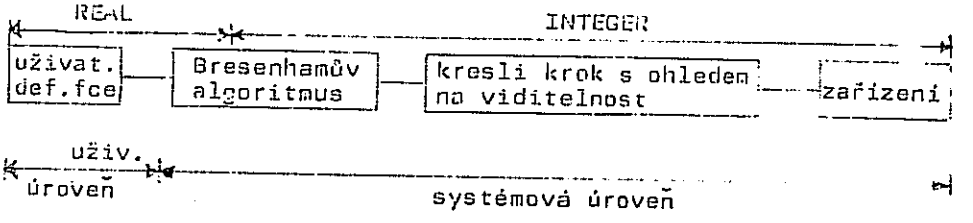
3.Navržená metoda

V [6] jsou funkce MASKTOP a MASKBOTTOM reprezentovány pomocí vektorů v pohyblivé řádové čárce. Proces kresleni lze pak znázornit obr.2. Nyni se můžeme ptát, zda existuje nějaká možnost zvýšení efektivnosti řešení daného problému. Jenou z možností je kombinace Watkinsonovy metody s Bresenhamovým algoritmem (viz. [2]) pro kresleni úsečky v rastrovém prostředi přímo na fyzické úrovni kresleni. Vzhledem k tomu, že pracujeme s rastrovým zařízením na úrovni fyzického kroku zbavime se všech výše uvedených problémů.

Řešení viditelnosti je nyní velmi jednoduché, neboť musíme pouze modifikovat metodu proceduru DRAW STEP, která generuje kód pro fyzický pohyb kurzoru. Procedura DRAW STEP kreslí pouze jeden fyzický krok a tudíž musíme pouze kontrolovat, zda koncový bod kroku je uvnitř nebo vně pásu neviditelnosti. Předkládaná metoda je znázorněna na obr.3.



obr.2.



obr. 3.

Je zřejmé, že nyní potřebujeme pouze celočíselnou reprezentaci pro maskovací vektory MASKTOP a MASKBOTTOM. Zjednodušené řešení je znázorněno algoritmem2.

```
{GLOBÁLNÍ PROMĚNNÉ}
VAR xo,yo: REAL; {Absolutní souřadnice}
    masktop,maskbottom: ARRAY [0..1023] OF INTEGER;
    PROCEDURE DRAWSTEP (dx,dy,i: INTEGER );
    VAR flag: BOOLEAN;
    BEGIN xo:=xo+dx;
          yo:=yo+dy;
          flag:=i=1;{posuv nebo čára ?}
          IF flag THEN
          BEGIN flag:=FALSE;
                IF masktop[xo]<=yo THEN
                BEGIN masktop[xo]:=yo; flag:=TRUE; END;
                IF maskbottom[xo]>=yo THEN
                BEGIN maskbottom[xo]:=yo; flag:=TRUE; END;
          END;
          IF flag THEN physline(dx,dy)
                  ELSE physmove(dx,dy);
    END;
```

Algoritmus 2.

4. Závěr

Předložený algoritmus pro kresleni funkci dvou proměnných s respektováním viditelnosti je určen pro implementaci v perifernich zařizenich rastrového typu, nebot algoritmus lze realizovat asi 20 strojovými instrukcemi bez použiti pohyblivé řádové čárky. Nyni lze rozšiřit soubor instrukci grafických periférii rastrového typu o instrukce:

- inicializuj maskovaci vektory
- kresli čáry s ohledem na viditelnost

V [3],[4] lze nalézt úplné řešeni i s algoritmem pro obecné natočeni při dodrženi správného pořadi kresleni a modifikace základniho algoritmu,která eliminuje chyby vzniklé skládáním dvou obrazů jež vznikly kreslenim řezů paralelnich s osou x a y. Podstatnou výhodou uvedeného přistupu je, že se uživatel nemusí zabývat volbou vhodného kroku apod. Daný algoritmus byl implementován na mikropočitačich Crommenco Z2H a Apple II, přičemž byl postatně rychlejši než Watkinsonův algoritmus a Williamsův algoritmus. Pokud dané grafická periférie bude vybavena mikroprocesorem a pamětí, neni problémem soubor instrukci periférie rozšířit o instrukci:

- Kresli funkci dvou proměnných s ohledem na viditelnost při daném natočeni pro úhly α a β (v tomto případě by asi fukce byla dána tabulkou hodnot funkce v uzlových bodech sitě).

5. Literatura

[1] Boutland J.: Surface Drawing Made Simple, Computer Aided Design 11(1) January 1979,pp.19-22

[2] Bresenham J.E.: Algorithm for Computer Control of Digital Plotter, IBM Syst. J. 4(1), 1965, pp.25-30

[3] Skala V.: Hidden - line Processor, CSTR/29, Computer Sci. Dept. Brunel University, Middlesex, 1984

[4] Skala V.: Hidden - line Processor, CSTR 209- -84, Katedra Technické Kybernetiky, VŠSE , Plzeň, 1984

[5] Sowerbutts W.T.: A Surface-Plotting Program Suitable for Microcomputers, Computer Aided Design 15(6), November 1983, pp.324-327

[6] Watkins S.L.: Masked Three-Dimensional Plot Program with Rotation, CACM 17(9), September 1974, pp.520-523

[7] Williamson H.: Hidden-Line Plotting Program, CACM 15(2), February 1972, pp.100-103

[8] Wright W.A.: A Two-Space Solution for the Explicit Function of Two Variables, IEEE Trans. on Comp.22(1), January 1973,pp.28-33

Mezinár. Konf. Algoritmy 93, April 26-29, 1993
Vysoké Tatry, ~~ISMAT při OS SAV~~ 1993

POUŽITÍ POČÍTAČOVÉ GRAFIKY PŘI VÝUCE ALGORITMIZACE

Ivana Kolingerová, Jana Krutišová, Václav Skala
Katedra informatiky a výpočetní techniky,
Západočeská univerzita, Americká 42, 306 14 Plzeň
Email
kolinger@kron.zcu.cs, krutis@kron.zcu.cs, skala@kron.zcu.cs

1. Úvod

Algoritmizace a základy programování se v současné době vyučují prakticky na všech středních a vysokých školách. Většina nám známých publikací na toto téma se věnuje převážně výkladu jednotlivých rysů toho či onoho programovacího jazyka, aplikaci vývojových diagramů, strukturogramů apod. Je nicméně otázkou, jaký postup při výuce algoritmizace volit, aby student nebyl nucen se zabývat několika problémy najednou. Musí řešit otázku návrhu algoritmu, jeho formalizace a verifikace jeho správné činnosti. To vše v okamžiku, kdy ještě zdaleka nemá "zažitý" programovací jazyk a kdy se v mnoha případech zabývá spíše otázkou, jak "myšlenku" vyjádřit pomocí jazyka než podstatou řešeného problému.

Podobně jako v uměleckých oborech je nutné studovat metody "starých mistrů", je v programování zapotřebí se zabývat již existujícími algoritmy. Každý si zajisté položí otázku, jak by skutečné studium "starých mistrů" a vlastně celá výuka algoritmizace a programování měly vypadat.

Každý programátor, který se někdy pokoušel o analýzu algoritmu jiného autora, jistě potvrdí, že mezi jeho přečtením a pochopením je značný rozdíl. Abychom algoritmu plně porozuměli, většinou se neobejdeme bez ruční simulace jeho chodu. Tato simulace je poměrně pracnou záležitostí, proto by bylo vhodné ruční práci nahradit užitím výukového programu. Zde také vidíme pole působnosti pro aplikace počítačové grafiky.

Z tohoto důvodu byl proveden experiment, kdy činnost známých algoritmů byla realizována prostředky počítačové grafiky a jejich běh demonstrován pomocí animace. Jednotlivé

programy pro animaci algoritmů byly vytvořeny v rámci předmětu Základy počítačové grafiky na ZČU jako jedna ze dvou prací zadaných v rámci semestru. (Dohromady takto vzniklo 26 programů).

Na obr.1 je zachycen obvyklý tvar grafického výstupu. Právě prováděné části jsou vhodným způsobem zvýrazněny, viz část obrazovky s programem a vývojovým diagramem, aktuální činnost se předvádí pomocí animace, viz výměna prvků ve spodní části. Zároveň se vypisují aktuální hodnoty proměnných, což umožňuje studentovi sledovat postup provádění jednotlivých algoritmů. Takto koncipovaná animace je vlastně speciálním případem výukového programu.

2. Požadavky na výukové programy

Výukovým programem nehodlámo suplovat výklad vyučujícího, protože se domníváme, že lidský faktor je ve výuce nenahraditelný. Využití programů předpokládáme pro procvičení již vyložené látky. Z tohoto předpokladu se také odvíjejí požadavky týkající se obsahu a formy výukových programů.

K výukovému programu by měla být k dispozici podrobná dokumentace shrnující jak procvičované téma, tak metodu řešení, a vysvětlující ovládání programu a význam všech na obrazovce se vyskytujících identifikátorů. Tato dokumentace by měla být dostupná i z programu, nejlépe na stisk určité klávesy, nebo alespoň ve formě úvodní informace. Z hlediska efektivního využití času je vhodné umožnit při opakovaném použití programu vypuštění tohoto úvodu. Žádoucí by také byla možnost nastavení různých úrovní obtížnosti a rychlosti běhu.

Důraz by měl být kladen na interaktivní práci. Uživatel by měl být veden k aktivní účasti, nikoliv pouze odsouzen k pasivnímu přihlížení. Např. formou dotazu jej nutit udržovat pozornost na dění na obrazovce. V dokonalejším případě by uživatelovy zásahy mohly ovlivňovat i průběh zobrazované činnosti v mezích mírných modifikací.

Před skončením programu neopomineme vyhodnotit výsledky dosažené při kontrolních dotazech.

Kromě interaktivní formy je vhodné mít možnost spustit program s předem zadanými hodnotami.

Program by měl být snadno ovladatelný, pokud možno pomocí obecně užívaných kláves (např. <CR>, <ESC> apod.) - svoji tvůrčí invenci napřeme jiným směrem než je vymýšlení nových, "neotřelých" kombinací kláves. Při každém požadavku na interaktivní vstup by měla být k dispozici přehledná a jednoduchá nabídka možných odpovědí, nejlépe formou menu. Pokud možno se vyhýbáme přímým alfanumerickým vstupům.

Vždy je nutno předpokládat, že uživatel bude zkoušet i jiné varianty, než jsou mu nabízeny, nelze tedy počítat s bezchybnou obsluhou. Je nutno předejít zhroucení programu při chybném ovládání, a tedy chyby ošetřovat přímo v programu; neošetřené chyby, které se "propracují" až na úroveň operačního systému a vyvolají adekvátní systémové hlášení, jsou nepřípustné. Je třeba dokázat zpracovat i opakované nesprávné odpovědi a blokovat určité klávesy. Při chybné reakci uživatele umožnit volbu návratu do posledního správného kroku nebo na začátek programu, dovolit opravu překlepů a změnu jen té části dat, která byla zadána chybně. Na každou platnou uživatelovu volbu je třeba adekvátně reagovat, dát najevo, že byla správně pochopena a že se realizuje.

Při návrhu dialogu se vyhýbáme počítačovému slangu a snažíme se o srozumitelné a krátké formulace.

Jako stálí uživatelé počítačové grafiky musíme potvrdit oprávněnost požadavků vyhýbat se při návrhu grafického řešení programu užívání většího množství barev, ostrým odstínům a příliš kontrastním barevným kombinacím unavujícím zrak. Tři vzájemně sladěné, dobře odlišitelné, ale ne přehnaně kontrastující barvy obvykle postačí. Také blikání částí obrazu je zajímavé pouze při jednorázovém užití programu, ale při častějším zacházení působí nepříjemně.

Podobná střídmost je žádoucí i pro zvukové efekty. Pokud si jako autoři programu tento typ poškozování životního prostředí nedokážeme odepřít, měli bychom alespoň umožnit vypnutí těchto efektů.

3. Programové vybavení pro výuku algoritmizace

Než přejdeme k podrobnějšímu popisu realizovaného programového vybavení, je třeba znovu zdůraznit, že je zamýšleno jako

podpůrný prostředek pro zvládnutí publikovaných algoritmů, nikoliv pro tvorbu nových. Jeho autory jsou studenti III. ročníku oboru Elektronické počítače ZČU v Plzni. Je napsáno převážně v jazyce C, menší část v jazyce Pascal.

Jedná se o následující adresáře s programy :

ARABRIM - převod čísel z arabské do římské soustavy
DAMA - společenská hra
DIAG - cyklická záměna diagonálních prvků matice
GENIUS - společenská hra
HANOJ - problém hanojských věží
HORNER - Hornerovo schéma
INTEGR - výpočet určitého integrálu pomocí lichoběžníkové metody a metodou Gaussových váhových koeficientů
INTERPOL - výpočet interpolačního polynomu v Newtonově tvaru
MAXPRVEK - stanovení vektorů řádkových maxim a vektoru sloupcových maxim v rámci jediného průchodu danou obdélníkovou maticí
NASPOLY - násobení dvou polynomů
NEVILL - Nevillův algoritmus pro výpočet hodnoty interpolačního polynomu pro dané x
NULMAT - nalezení největší nulové čtvercové submatice v matici
OBVMAT - součet hodnot prvků matice po obvodech všech soustředných čtverců a výpočet průměrných hodnot prvků matice v těchto čtvercích
POSUV - posun dat v matici : v lichých řádkách vpravo, v sudých vlevo, data, která "přetekla" z řádku, postupují do uvolněného místa následujícího řádku, data z posledního řádku pak do uvolněného místa v 1.řádku
PRUNIK - průnik uspořádané a neuspořádané množiny
PRUNIK1 - průnik dvou setříděných množin
PRUNIK2 - průnik dvou nesetříděných množin
PRVCIS - nalezení prvočísel
RIDMAT - převod řídké matice na tabulku
RIMARAB - převod čísel z římské do arabské soustavy
RUBIK - skládání Rubikovy kostky
SPLINE - interpolace spline - funkcí
SNEK - magická matice
TELESA - algoritmy pro vykreslování těles

VEKTOR - zrcadlové převrácení vektoru celých čísel kolem osy symetrie
ZAMENA - vzájemná výměna prvků hlavní a vedlejší diagonály matice

Ukázka grafického výstupu na obr.1 pochází z programu VEKTOR. Je na ní zachycen okamžik, kdy dochází k výměně dvou prvků ve vektoru. Aktuální místo je ve výpisu programu i vývojovém diagramu od ostatních částí barevně odlišeno. Program je možné krokovat nebo nechat proběhnout celý. Jiné aktivní zásahy nejsou možné.

4. Shrnutí zkušeností z tvorby a distribuce výukových programů

Jak je vidět z předchozí ukázky, vytvořené programové vybavení splňuje zatím pouze část požadavků kladených na výukové programy. Většinou chybí podrobný návod, jednotné ovládání pomocí standardních kláves, možnost nastavení různých úrovní obtížnosti a především aktivní zapojení uživatele. V některých případech není dokonale ošetřen chybný vstup nebo potvrzena uživatelova volba.

Jsme si těchto nedostatků vědomi. Neradi bychom však značné množství práce, které by si další zdokonalení vytvořeného programového vybavení podle výše uvedených požadavků vyžádalo, vynakládali zbytečně, na dílo, které skončí v šuplíku, protože je nikdo nepotřebuje. Rádi bychom proto nejprve vyprovokovali odezvu té části veřejnosti, která se výukou algoritmizace zabývá, a může tedy nejspíše zhodnotit užitečnost celého projektu.

Ve jménu této snahy jsme poskytli 19 vybraným školám informaci o vytvořeném programovém vybavení. Bylo možné je získat zdarma po zaslání diskety. Jedinou podmínkou bylo krátké zhodnocení programů. Bohužel, hodnocení jsme získali pouze od nepatrného procenta všech respondentů. To jistě nedává příliš lichotivý obrázek zájmu zúčastněných pedagogů o nové formy práce.

Nechceme se však na podkladě negativních zkušeností se statisticky nevýznamnou částí naší pedagogické veřejnosti vzdát naděje na odezvu a spolupráci pedagogů. Jim především by měly

naše programy sloužit; na jejich reakcích a názorech tedy záleží, zda budeme na výukových programech dále pracovat anebo ne.

Možná, že cesta, po níž jsme se prvními krůčky vydali, nevede nikam; pak nám nezbyde, než konstatovat, že jsme zase skončili v cimrmanovské roli průkopníků slepých uliček, a zkusit kopat zas v některé jiné.

5. Závěr

Pokusili jsme se v tomto článku zamyslet nad problematikou užití počítačové grafiky při výuce algoritmizace a podělit se o první zkušenosti z tvorby a distribuce příslušného programového vybavení. Svůj přístup jsme dokumentovali na konkrétní ukázce jednoho výukového programu.

Prezentované programové vybavení bylo vytvořeno studenty III.ročníku FAV ZČU v Plzni jako podpůrný prostředek pro výuku algoritmizace. Zájemci mohou programy po zaslání diskety získat zdarma za podmínky, že svoje dojmy a zkušenosti z jeho užívání krátce zhodnotí podle doporučených kritérií a svůj posudek nám zašlou.

Zajímají nás především tato hlediska hodnocení jednotlivých programů :

- možnost využití v konkrétních učebních předmětech
- názornost
- způsob ovládání
- srozumitelnost doprovodných textů
- oblasti, kam by měl být zaměřen výběr algoritmů
- nedostatky

Pokud vás náš příspěvek donutil zaujmout k problematice výukových programů vlastní stanovisko, pak byl jeho účel splněn. Za jakýkoliv námět, názor nebo příspěvek do polemiky předem děkujeme.

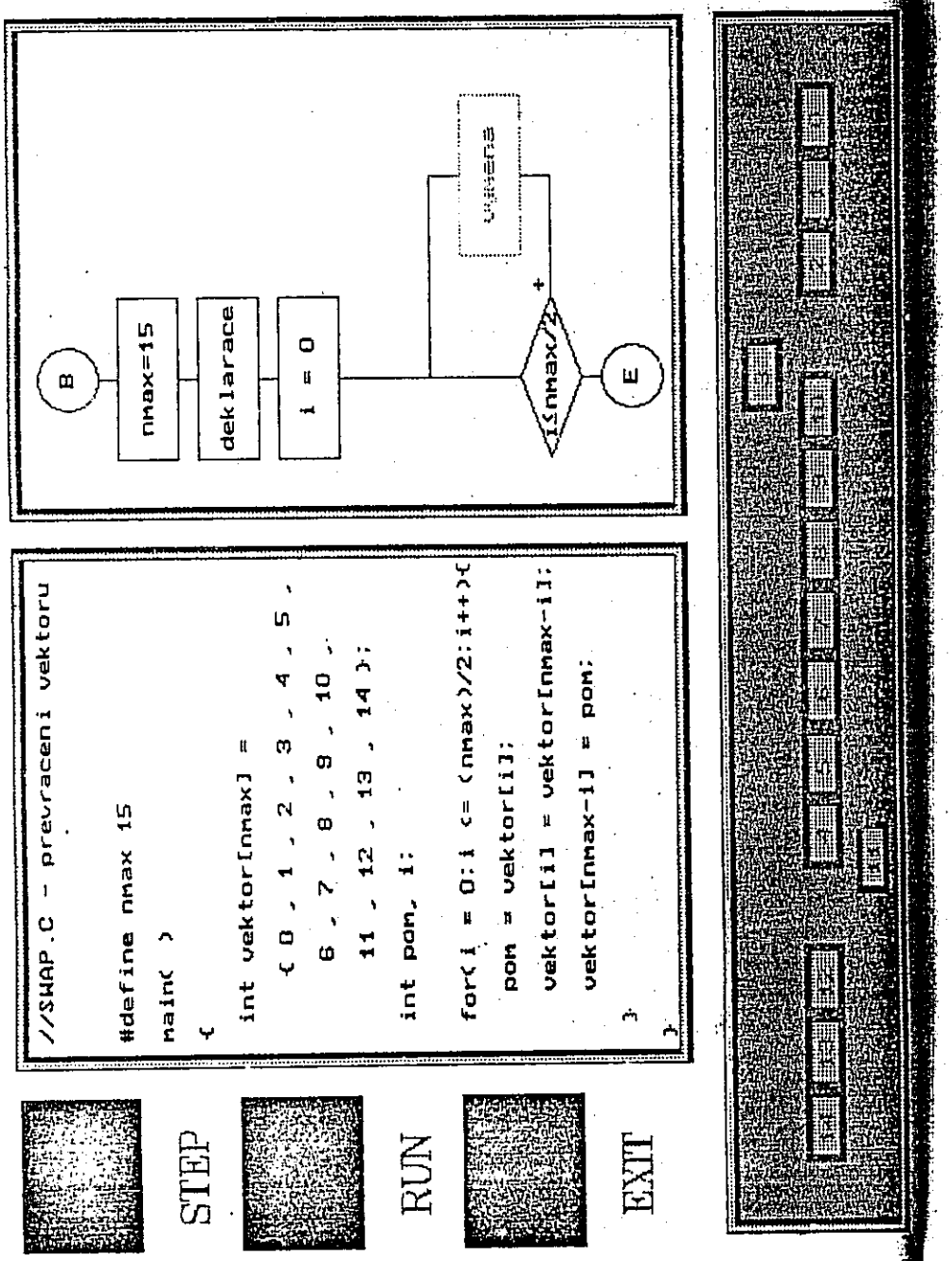
6. Literatura

[1] Mazák E. : Posuzování a hodnocení počítačových a výukových programů, Ústav školských informací, Praha, 1989.

Abstract

Computer graphics in learning of computer algorithms - is it a good idea or not ? Some remarks and experience in this field are discussed in this paper. Example of a teaching program is

Obr.1 : Ukázka grafického výstupu programu VEKTOR



Formalizácia opisu štruktúry hesiel Krátkeho Slovníka Slovenského Jazyka (KSSJ)

Eduard Kostolanský
Laboratórium počítačovej lingvistiky
PdF UK, Moskovská 3, 813 34 Bratislava

1. Úvod

Cieľom tohoto materiálu je: 1. podať výsledky analýzy (KSSJ); 2. zaradiť jednotlivé komponenty vystupujúce v heslách KSSJ do prijatej taxonómie lexikografických údajov; 3. opísať štruktúru hesiel, alebo ich častí pomocou vhodného formalizmu (pojmami formálnych bezkontextových jazykov).

Za účelom kompaktnosti materiálu v prvej časti uvedieme prijatú taxonómiu pre lexikografické údaje, pomocou ktorej sa pokúsime charakterizovať informačné komponenty hesiel KSSJ. Na základe dostupných materiálov obsahujúcich rôzne štruktúry slovníkových hesiel, ich vyhodnotenie a porovnávanie [2, 3] považujeme pre náš cieľ za vhodnú taxonómiu, ktorú vypracovali dánski autori [3].

2. Taxonómia lexikálnych údajov

Uvažovaná taxonómia obsahuje nasledovné informačné typy:

Projektovo-špecifické informačné typy

Jazyk	- národný jazyk - sociolekt - dialekt - jazykový stupeň
Metóda tvorby slovníka	- semaziologická - onomaziologická
Vyjadrovacie médium	- reč - písmo - obrazy (grafy)

Všeobecné informačné typy

hlavné skupiny	kategórie	podkategórie
Etymologické informácie	- paralely - pôvod	
Grafické informácie	- ortografické informácie - grafický symbol	
Gramatické informácie	- slovný druh	

Konference Algoritmy 93, Stará Lesná, Vysoké Tatry 1993

Barevné systémy a jejich aplikace v počítačové grafice

Václav Skala
Západočeská univerzita, Americká 42, Box 314, 306 14 Plzeň
e-mail: skala@kron.zcu.cs skala@eros.zcu.cs

1. Úvod

S rozvojem aplikací výpočetní techniky roste i použití barev v oblasti počítačové grafiky. Kromě známých barevných systémů existuje ještě celá řada systémů, které jsou méně známy, i když jsou významné z hlediska aplikací v oblasti výpočetní techniky. Vedle barevných systémů RGB, CMY, CMYK, YIQ existují systémy HLS, HSV, HSI, které jsou orientovány uživatelsky. Některé systémy, např. CIE-xy, CIE-uv, CIE-UVW, CIE-LAB a CIE-$L^*u^*v^*$, jsou známy více v oblasti aplikací osvětlovací techniky. Některé systémy zasluhují větší pozornosti i z hlediska aplikací počítačové grafiky. Informace o jednotlivých barevných systémech jsou shrnuty v [1] včetně vzájemných převodů a souvislostí.

Vidíme-li světlo určité vlnové délky, získáváme určitý vjem barvy. Přirozené zdroje světla však neobsahují pouze jednu vlnovou délku, i když vnímáme určitou barvu. Je známo, že většina reálných barev může být vytvořena pomocí základních barev, a to červené (R-Red), zelené (G-Green) a modré (B-Blue) v aditivním systému RGB, resp. modrozelené (C-Cyan), purpurové (M-Magenta), žluté (Y-Yellow) v subtraktivním systému CMY.

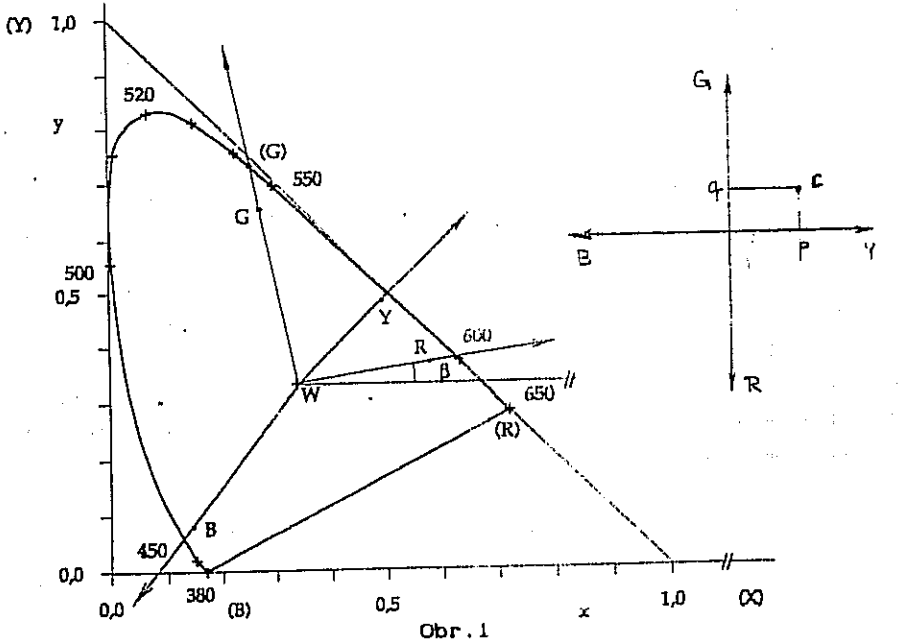
Kromě běžných systémů jsou používány též systémy respektující způsob vnímání barev. Podle teorie jsou na sítnici oka tři druhy čípků, které produkují tři různé signály na základě světelného podnětu, a to:

- signál jasový jako souhrnný vjem v oblasti červené a zelené,
- signál pro odlišení barev v oblasti červená - zelená,
- signál pro odlišení barev v oblasti žlutá - modrá.

Systémy založené na uvedeném principu vysvětlují celou řadu jevů v oblasti vnímání barev.

2. Systém Opponent

Jedním z nejjednodušších systémů je systém Opponent, který je založen na předpokladu, že barvy jsou určeny polohou (p,q) v diagramu RYGB. Obr.1 pak znázorňuje, jak jsou jednotlivé složky definovány při převodu ze systému CIE-xy, popř. RGB.



Obr.1

p	q	β
≥0	≥0	$\vartheta_Y + (\vartheta_G - \vartheta_Y)\ \frac{2}{\pi}\ \mathrm{arctg}(\frac{q}{p})$
<0	≥0	$\vartheta_G + (\vartheta_B - \vartheta_G)\ \frac{2}{\pi}\ \mathrm{arctg}(-\frac{p}{q})$
<0	<0	$\vartheta_B + (\vartheta_R - \vartheta_B)\ \frac{2}{\pi}\ \mathrm{arctg}(\frac{q}{p})$
≥0	<0	$\vartheta_R + (\vartheta_Y - \vartheta_R)\ \frac{2}{\pi}\ \mathrm{arctg}(-\frac{p}{q})$

Tabulka 1

kde úhel ϑ_Y je úhel sevřený osou x a spojnicí bodu x_w, který reprezentuje bílou barvu, a bodu Y , který reprezentuje žlutou barvu (Yellow); analogické vztahy platí pro ϑ_G , ϑ_R , ϑ_B .

Je-li barva c v systému Opponent určena polohou (p,q) v diagramu RYGB a jasem L, pak je poloha barvy c v systému CIE-xy určena úhlem β, vzdáleností od bodu reprezentujícího bílou barvu a jasem. Úhel β je definován vztahy, viz tab.1.

Vzdálenost od bodu x_w je pak určena vztahem

$$R = \sqrt{p^2 + q^2} \quad .$$

Pak souřadnice barvy c v systému CIE-xy jsou přibližně určeny takto:

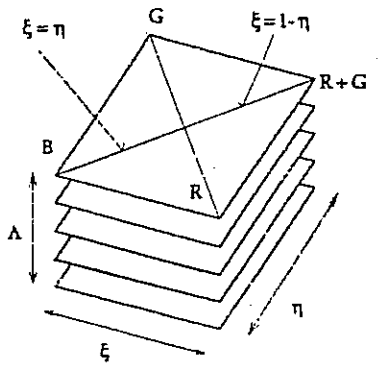
$$x = x_w + R \cos \beta \quad , \qquad x = x_w + R \sin \beta$$

$$z = 1 - x - y \quad .$$

V souřadném systému XYZ má barva c souřadnice

$$X = x \frac{L}{y} \quad , \qquad Y = L \quad , \qquad Z = z \frac{L}{y} \quad .$$

Tento přepočet je sice poměrně hrubý, ale je kvalitativně správnou aproximací fyziologického vizuálního systému. Pro případné srovnávání je vhodné přesně definovat referenční body systému Opponent.



Obr.2

I když tento systém vysvětluje mnohé jevy, je však výpočetně poměrně náročný, přičemž gamut barev pro display je nelineární a systém neposkytuje jednoduchá pravidla pro míšení barev. Z tohoto důvodu byl zaveden systém RGYB.

3. Systém RGYB

Systém RGYB byl zaveden pro odstranění některých vad systému Opponent. Jeho princip je znázorněn na obr.2. Z obrázku je zřejmé, že achromatické barvy, tj. bílá, odstíny šedé a černá jsou určeny bodem $\xi = 0{,}5$, $\eta = 0{,}5$, přičemž Λ určuje úroveň jasu, tj. šeď.

Pro systém RGYB platí tyto převodní vztahy pro převod do systému RGB

$$R = \xi \Lambda \quad , \qquad G = \eta \Lambda \quad ,$$

$$B = (1 - \max(\xi , \eta)) \Lambda \quad ,$$

kde

$$\xi \in \langle 0 , 1 \rangle \quad , \qquad \eta \in \langle 0 , 1 \rangle \quad , \qquad \Lambda \in \langle 0 , 1 \rangle \quad .$$

Z hlediska výpočetní náročnosti jsou požadavky zanedbatelné. Uvedený systém vykazuje navíc i jiné dobré vlastnosti, např. z hlediska míšení barev, kdy pravidla mají lineární charakter. Pro $\Lambda \in \langle 1 , 2 \rangle$ systém respektuje i vliv saturace.

Systém RGYB lze též modifikovat tak, že

$$R = 2 \xi - \Lambda \xi + \Lambda - 1 \quad ,$$

$$G = 2 \eta - \Lambda \eta + \Lambda - 1 \quad ,$$

$$B = \max (\xi,\eta) (\Lambda - 2) + 1 \quad .$$

Pak "horní vrstva", tj. $\Lambda = 2$, je celá bílá, zatímco "dolní vrstva", tj. $\Lambda = 0$, je celá černá a "prostřední vrstva", tj. $\Lambda = 1$, obsahuje úplnou paletu barev.

Až dosud byly předloženy různé barevné systémy a jejich vzájemné převody. Při použití moderních barevných výstupních zařízení je nezbytné zkoumat i otázku zajištění stejného barevného vjemu jak na obrazovce, tak i na výstupu z barevné tiskárny. Uspokojivé řešení tohoto problému je velmi náročné a zcela překračující možnosti dostupné techniky.

4. Systém AC_1C_2

Systém AC_1C_2 odvodil Meyer, který definoval převodní vztahy takto

$$\begin{bmatrix} A \\ C_1 \\ C_2 \end{bmatrix} = \begin{bmatrix} -0,0177 & 1,0090 & 0,0073 \\ -1,5370 & 1,0821 & 0,3209 \\ 0,1948 & -0,2045 & 0,5264 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

5. Systém SθW

Systém SθW není příliš znám a jde vlastně o vyjádření systému CIE UVW v polárním souřadném systému. Proměnná S popisuje saturaci barvy, θ barevný tón a W jas barvy, viz CIE-UVW. Pro souřadnice S , θ , W platí následující převodní vztahy

$$S = \sqrt{U^2 + V^2} = 13\ W \sqrt{(u - u_w)^2 + (v - v_w)^2}$$

$$\theta = \operatorname{arctg}\ (v/u) = \operatorname{arctg}\ [(v-v_w)/(u-u_w)]$$

kde $(u_w v_w)$ je pozice smluvního bílého světla v diagramu CIE-uv.

Mnohá grafická zařízení nemají k dispozici velkou paletu barev, např. laserové nebo inkoustové tiskárny, zatímco dnes již standardní karta Super VGA poskytuje alespoň 256 barev současně zobrazitelných. Výstupní rastrová grafická zařízení lze rozdělit z hlediska použití barev takto:

- výstupní zařízení
  - s pevnou paletou — inkoustové, laserové tiskárny 3 – 4 barvy v paletě
  - s volitelnou paletou
    - VGA 16 barev/256 možných
    - Super VGA 256 barev/4K možných
    - speciální 4K barev/10^6 možných

(1K = 1024)

Je zřejmé, že docílit velmi věrného barevného výstupu není víceméně problémem, pokud je použit vhodný video-interface a displej. Naproti tomu docílit **velmi kvalitního** výstupu např. na laserové nebo inkoustové tiskárně při použití pouze 3 – 4 barev je velmi obtížné. Obecně lze říci, že je možné použít modifikovaných technik půltónování [1] k docílení velmi věrného barevného výstupu.

6. Závěr

Barevné systémy a jejich aplikace v oblasti barevných výstupů nabízejí celou řadu nových možností, které budou aktuální zejména z hlediska dostupnosti barevných výstupů. Cílem příspěvku není podat vyčerpávající přehled a souvislosti, které mohou být nalezeny v [1], kde je uvedena i celková použitá literatura, ale poukázat na některé systémy, které mohou být z hlediska aplikací počítačové grafiky aktuální.

7. Literatura

[1] Skala V: Algoritmy počítačové grafiky III, skripta ZČU Plzeň, 1992.

[2] Skala V.: Světlo, barvy a barevné systémy v počítačové grafice, ACADEMIA, 1992

Abstract

A short survey of known color systems is presented. Basic principles and transformations are shown together with properties of the presented color systems which can be used within computer graphics applications and desk top publishing systems.

3. Závěr

Na základě provedených analýz chování v literatuře popsaných algoritmů ořezávání [1]-[5] a algoritmu navrženého lze předpokládat urychlení algoritmu ořezávání zejména pro ty scény, kdy většina hran protíná ořezávanou oblast.

Lze ukázat, že uvedený algoritmus je dostatečně robustní i pro speciální případy a je jej možné modifikovat i pro případ ořezávání nekonvexním n-úhelníkem.

4. Literatura

[1] Barsky B.A., Liang Y-D.: The Optimal Tree Algorithm for Line Clipping, EUROGRAPHIC'92 Coference Note, Cambridge, 1992.

[2] Krammer G.: A Line Clipping Algorithm and Its Analysis, Computer Graphics Forum (EG'92 Conference Proceedings), Vol.11, No.3, 1992, pp. C253-266.

[3] Skala V.: Algorithms for 2D Line Clipping, EUROGRAPHICS'89 Proceedings, North Holland, 1989, pp. 355-366.

[4] Skala V.: Metody ořezávání v E_2 a E_3, Habilitační práce, Západočeská univerzita, Plzeň, 1990.

[5] Skala V.: Algoritmy počítačové grafiky II, skripta ZČU, Plzeň, 1992.

Abstract

A new algorithm for clipping lines against convex window is described. The efficiency of the proposed algorithm seems to be substantially higher to the known algorithms. A theoretical comparison with the Cyrus-Beck algorithm is given, too. Algorithm can be easily modified for clipping lines against non-convex windows.

konference
Algoritmy 93, Stará Lesná, Vysoké Tatry, 26-29. dubna 1993

Algoritmus sledování paprsku a jeho složitost

Václav Skala

Západočeská univerzita, Americká 42, Box 314, 306 14 Plzeň

e-mail: skala@kron.zcu.cs skala@eros.zcu.cs

1.Úvod

Algoritmus sledování paprsku (Ray Tracing, dále jen RT) je v současné době velmi populární v oblasti věrného zobrazování scén. Jeho nespornou výhodou je možnost snadné programové realizace, snadný výpočet odlesků a zrcadlení, případně i realizace optických jevů, jako je lom světla apod. Velkou nevýhodou je pak především to, že doba výpočtu je neúnosně dlouhá i pro velmi jednoduché scény. Z tohoto důvodu se hledají různé metody urychlení. Na základě analýzy algoritmů byl navržen způsob urychlení výpočtu průsečíku paralelních primárních paprsků s tělesy nebo plochami.

2. Princip algoritmu

Každý primární paprsek, který je vlastně polopřímkou v prostoru E_3, může být určen jako průsečnice dvou rovin, které nejsou rovnoběžné. Označíme-li:

$^1\rho_i$ rovinu, která tvoří i-tý řádek na průmětně,

$^2\rho_j$ rovinu, která tvoří j-tý sloupec na průmětně

pak paprsek procházející pixlem (i,j) na průmětně je dán jako průsečík rovin $^1\rho_i$ a $^2\rho_i$. Pak lze vytvořit bitové vektory Σ_i pro i-tý řádek a Ω_j pro j-tý sloupec tak, že hodnota k-tého bitu $\Sigma_i[k]$ určuje, zda k-tá plocha je protnuta i-tou rovinou. Analogicky pak $\Omega_j[k]$ vyjadřuje, že j-tý sloupec protne na průmětně k-tou plochu. V zásadě lze říci, že minmax box test pro

k-tou plochu, resp. těleso, je ekvivalentní výsledku operace

$$\Sigma_i[k] \text{ and } \Omega_j[k]$$

Bitové vektory Σ_i a Ω_j je možné výhodně vytvořit předem ve fázi definice zobrazované scény. Navíc operaci **and** nad bitovými vektory je možné provádět podstatně rychleji než porovnávání na minmax box test, viz tab.1.

PC 486 33MHz 64KB cache

	=	<	+	-	*	/
int	6	9	3	1	26	44
float	33	60	16	16	20	114

Časy jsou v 1/10 sekundy a pro 5 000 000 opakovaní operací.

Tabulka 1

Uvedený způsob navíc umožňuje i **kontrolu konzistence** zobrazované scény, neboť je možné jednoduše kontrolovat, zda bude **každý** objekt na scéně vyhodnocen z hlediska výsledného zobrazení na scéně, tj. lze **detegovat**, že paprsek se **"nedotkne"** tělesa. Lze ukázat, že pokud platí

$$\bigcap_i \Sigma_i \neq [1,\ldots,1]^T \qquad \text{nebo} \qquad \bigcap_j \Omega_j \neq [1,\ldots,1]^T$$

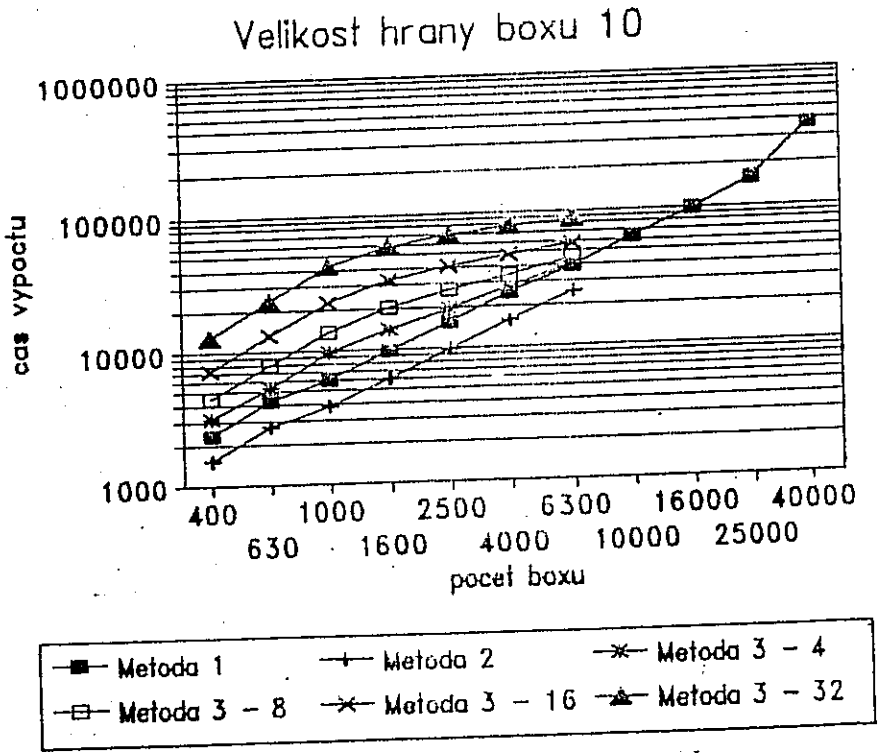
pak existuje q-té těleso, kterého se nedotkne žádný paprsek a bude ve vyhodnocování výsledné scény opominut, pro které platí

$$\Sigma_i[q] = 0 \quad \forall i \qquad \text{nebo} \qquad \Omega_j[q] = 0 \quad \forall j$$

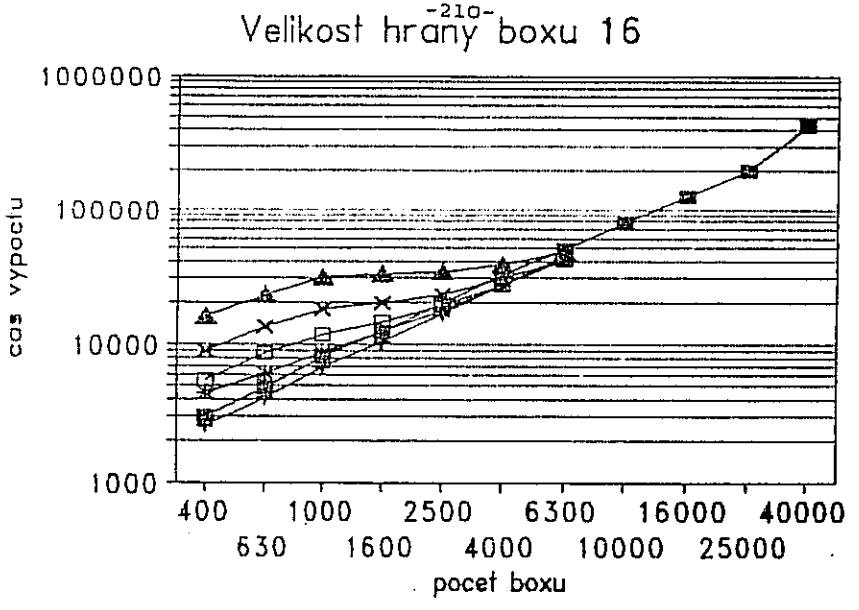
Uvedený princip lze rozšířit i o třetí dimenzi zavedením bitového pole Γ. Pak vlastně celou scénu "uzavřeme do" kvádru. V principu jde tedy o urychlení pomocí rovnoměrného rozdělení prostoru s tím, že neudržujeme seznam aktivních těles, resp. ploch, pro každý subkvádr, ale pouze odpovídající bitové mapy. Podobný přístup lze zvolit i pro perspektivní případ primárních paprsků.

3. Experimentální výsledky

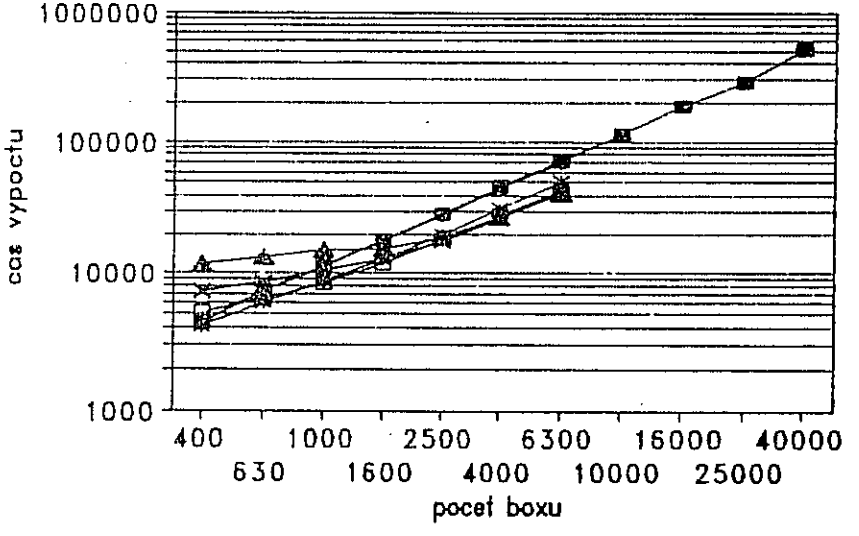
Pro porovnání byl realizován algoritmus, který využíval minmax box testu a který je v tabulkach označen jako metoda 1. Jako metoda 2 je označena realizace používající pouze dva bitové vektory Σ_i a Ω_i. Pro metodu 3, která využívá i dělení do hloubky prostoru, je vždy uveden počet dělení do hloubky. Všechny algoritmy byly realizovány na PC 386 s koprocesorem 16 MHz pro pro 256 x 256 pixelů. Scéna se sestávala z náhodně generovaných trojúhelníkových plošek, jejich velikost a počet, který byl volen, jsouuvedeny jako parametry. Vzhledem k dostupné technice nebylo možno otestovat navržený přístup pro více jak 6300 trojúhelníků. Uvedené časy (v desetinách sec.) zahrnují v sobě i čas nezbytný pro vlastní výpočet průsečíků paprsku s tělesem.



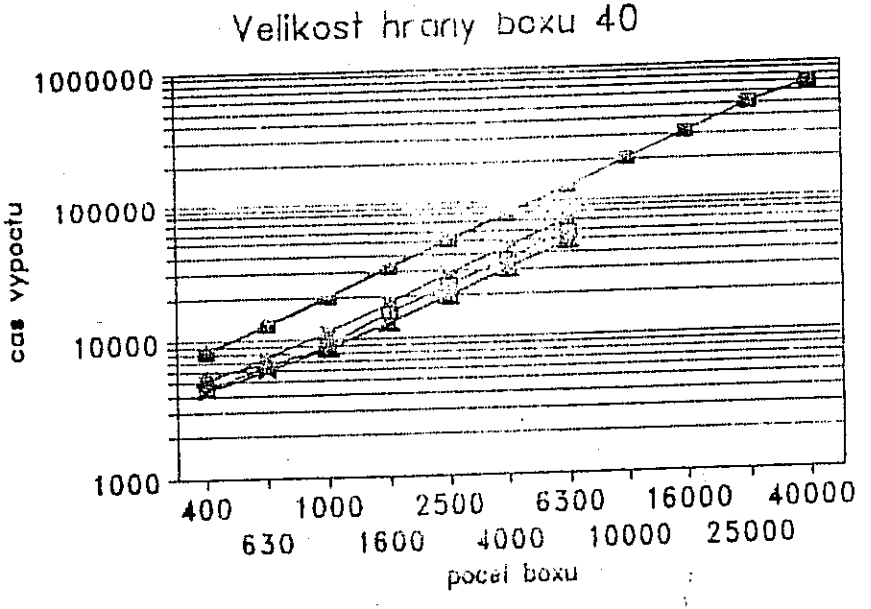
Obr.1



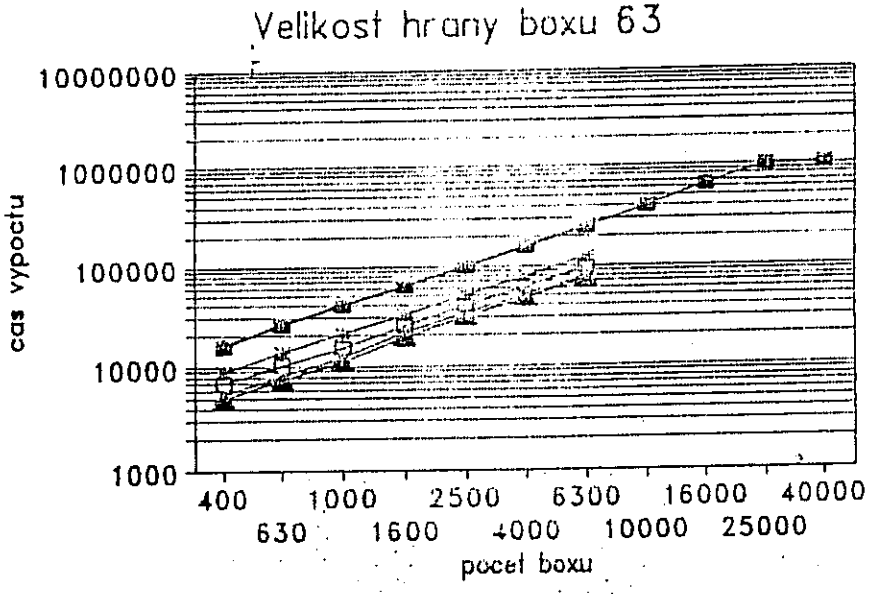
Obr.2



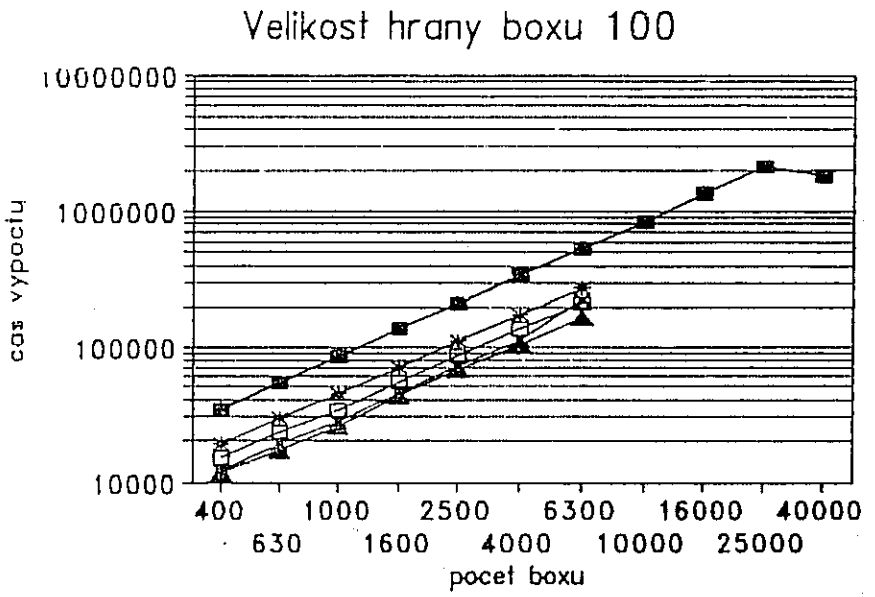
Obr.3



Obr.4



Obr.5



Obr.6

4. Závěr

Z dosažených výsledků vyplývá, že uvedený přístup ke zrychlení vyhodnocení primárních paprsků je výhodná zejména v případě velkého počtu objektů, a to i přes velkou jednoduchost použitého minmax box test ve tvaru $x_{min} \le x \le x_{max}$. Uvedený algoritmus je možné použít i pro případ řešení obecného primárního paprsku. V případě použití proobecný případ sledování paprsku je nutné použít metodu 3 a speciální algoritmus pro detegci incidence paprsku a "subkvádru" prostoru scény. Uvedený algoritmus umožňuje při použití hierarchických struktur vytvořit efektivní řešení i složitých scén. Z výsledků testů je zřejmé, že pro malé trojúhelníky bude navržený způsob rychlejší pro velké množství malých trojúhelníků, přičemž pro větší trojúhelníky je algoritmus výhodnější již pro menší počet.

5. Literatura

[1] Glassner A.S.(Ed.): An Introduction to Ray Tracing, Academic Press, 1989.

[2] Skala V.: Algoritmy počítačové grafiky III, skripta ZČU, Plzeň, 1992.

Poděkování

Je mi milou povinností poděkovat Pavlu Sebránkovi, studentovi zaměření počítačová grafika a CAD systémy, za pečlivou implementaci navrženého algoritmu a realizaci testů.

Abstract

A new algorithm for acceleration of primary ray tracing has been developped. Some tests have been made in order to prove the speed up. Comparisons showed that the proposed method is convenient for complex scenes and enables usage of hierachical object structures. The proposed algorithm can be used for the complete ray tracing, but for proper evaluation will be necassary to use workstation.

[2] R. Cieslak, C. Desclaux, A. S. Fawaz and P. P. Varaiya. Supervisory Control of Discrete-Event Processes with Partial Observations. *IEEE Trans. on Automatic Control*, AC-33, No.3, 249-260, 1988.

[3] F. Lin and W. M. Wonham. Decentralized Control and Cooperation of Discrete-Event Systems with Partial Observation. *IEEE Trans. on Automatic Control*, Vol. AC-35, No.12, 1330-1337, 1990.

[4] S. Young and V. K. Garg. Transition Uncertainty in Discrete Event Systems. *In: Workshop on Discrete Event Systems*, June, 21-23, 1991, Amherst, U.S.A., 1991.

[5] V. I. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Sov. Phys. Dokl.*, Vol.10, No.8, 707-710, Feb., 1966.

[6] R. A Wagner and M. J. Fischer. The String-to-String Correction Problem. *J.ACM*, Vol.21, No.1, 168-173, 1974.

[7] O. H. Ibarra, T. Jiang and H. Wang. String Editing on a One-Way Linear Array of Finite-State Machines. *IEEE Trans. Comput.*, Vol. C-41, No. 1, pp. 112-118, 1992.

[8] W. J. Masek and M. S. Paterson. A Faster Algorithm Computing String Edit Distance. *Journal of Computer and System Sciences*, Vol. 20, pp. 18-31, 1980.

[9] H. Bunke and J. Csirik. Inference of Edit Costs using Parametric String Matching. *Proc. of the 11th IAPR Int. Conf. on Patt. Rec.*, The Hague, Vol.2, 549-552, 1992.

[10] S. Y. Lu and K. S. Fu. Stochastic Error-Correcting Syntax Analysis for Recognition of Noisy Patterns. *IEEE Trans. on Comp.*, Vol. C-26, No.12, 1268-1276, 1977.

[11] J. Pik. Structural Analysis of Experimental Curves in Numerical Taxonomy. *Proc. of the 8th IAPR Int. Conf. on Pattern Recognition*, Paris, Vol.1, 389-391, 1986.

[12] K. Abe and N. Sugita. Distances between Strings of Symbols – Review and Remarks. *Proc. of the 6th IAPR Int. Conf. on Patt. Recognition*, Munich, Vol.1, 172-174, 1982.

[13] S. B. Gershwin. Hierarchical Flow Control: A Framework for Scheduling and Planning Discrete Events in Manufacturing Systems. *Proceedings of the IEEE*, Vol.77, No.1, 196-209, 1989.

[14] S. Y. Lu and K. S. Fu A Sentence-to-Sentence Clustering Procedure for Pattern Analysis. *IEEE Trans. on Syst., Man, and Cyber.*, Vol. SMC-8, No.5, 381-389, 1978.

[15] P. Hess and H. Brezowsky. Katalog der Grosswetterlagen Europas. *Deutsch. Wetterd.*, Offenbach a.M., 1969.

[16] C. Mares and I. Mares. Testing of the Markov dependence for certain meteorological parameters. *Ninth Prague Conf. on Inf. Theory, Stat. Dec. Functions and Random Proc.*, Prague, 1982.

ABSTRACT

A similarity of discrete events in discrete event systems is considered. Using this, the event hierarchy and the event uncertainty are introduced. Following a hierarchical event structure, a concept of macro-events is proposed. To characterize the used formalism, algebraic linguistics and structural pattern recognition techniques are utilized, especially those of string matching. A simple example is included to illustrate the developed ideas in a real application.

Konference Algoritmy 93, ~~ISMAP 93~~ Stará Lesná, Vysoké Tatry
26.-29. dubna 1993

Algoritmus ořezávání přímky v E_2 konvexním n-úhelníkem

Václav Skala
Západočeská univerzita, Americká 42, Box 314, 306 14 Plzeň
e-mail: skala@kron.zcu.cs skala@eros.zcu.cs

1.Úvod

Algoritmy ořezávání jsou důležitou součástí každého programového vybavení v oblastech aplikací počítačové grafiky, kdy je nutné odstranění těch částí kresby, které leží vně dané oblasti. Vzhledem k tomu, že jde o operaci velmi často používanou, je v literatuře popsáno několik základních algoritmů pro ořezávání přímek, resp. úseček s mnoha variantami, viz [1]-[3]. Na základě analýzy zpracovávaných dat byl navržen nový algoritmus pro ořezávání přímky konvexním n-úhelníkem.

2. Algoritmus ořezávání přímky konvexním n-úhelníkem

Mezi základní algoritmy pro ořezávání přímek, resp. úseček lze řadit algoritmus Liang-Barského, Cyrus-Beckův a další, viz [1],[2]. Nejkritičtějším místem všech algoritmů je vlastně test, který určí, zda přímka protíná či neprotíná danou ořezávací oblast. Uvažme pro jednoduchost případ na obr.1.

Aby metoda ořezávání byla efektivní, je nutné rozpoznat případy, kdy přímka p protíná hranu daného n-úhelníka a kdy nikoliv, s minimální výpočetní náročností.

Označme ξ a η z-ové složky vektorového součinu, tj.

$$\xi = [\, s \times s_i \,]_z \qquad \eta = [\, s \times s_{i+1} \,]_z \qquad i = 0,\ldots,n-1$$

Lze ukázat, že přímka p protíná či neprotíná hranu $x_i x_{i+1}$ (na obr.1 je uveden případ pro i=0) podle tabulky tab.1.

$\xi > 0$	$\eta < 0$	protíná přímka p hranu?
+	+	ne
+	−	ano
−	+	ano
−	−	ne

Tabulka 1

Závislost definovanou tabulkou tab.1 lze vyjádřit pomocí výrazu

$$(\xi > 0) \text{ xor } (\eta < 0) \qquad \text{nebo} \qquad \xi * \eta < 0$$

přičemž tabulka i vzorec platí nezávisle na orientaci vektoru **s** a orientaci hran daného n-úhelníka (poznamenejme, že druhý způsob vyhodnocení podmínky je rychlejší, viz tab.2).

Nyní lze specifikovat úplný algoritmus, který je založen na tom, že přímka p může protínat konvexní n-úhelník pouze ve dvou bodech. Porovnáním navrženého algoritmu, viz alg.2, s algoritmem Liang-Barského a Cyrus-Beckova lze nahlédnout, že místo výpočtu průsečíku přímky p s každou hranou daného n-úhelníka se u navrhovaného algoritmu počítá průsečík pouze pro dvě hrany, pokud průsečík vůbec existuje, které byly určeny výše uvedeným testem. Je nutné zdůraznit, že doby provádění jednotlivých operací (:= , < , ± , * , /) jsou i při použití koprocesoru pro různé počítače poměrově různé, viz tab.2. Navíc je předpokládáno, že hodnoty vektorů $\mathbf{s}_i$ a $\mathbf{n}_i$ jsou předpočteny $\forall i$, neboť oblast ořezávání se obvykle nemění pro velkou skupinu ořezávaných úseček.

PC 386DX / 387 25MHz

.	=	<	+	−	*	/
int	14	42	10	10	40	58
float	204	280	80	80	82	154

PC 486 33MHz 64KB cache – **vybrán pro vyhodnocení testu**

	=	<	+	−	*	/
int	5	9	3	1	26	44
float	33	50	16	16	20	114

Časy jsou v 1/10 sekundy a pro 5 000 000 opakování operací.

Tabulka 2

Označme N počet hran daného n-úhelníka. Srovnáme-li výpočetní náročnosti pečlivě implementovaného Cyrus-Beckova algoritmu [3] a algoritmu navrhovaného, pak dostáváme pro Cyrus-Beckův algoritmus složitost ve tvaru (pro případ protínání n-úhelníka přímkou):

$$(11, 3, 9, 6, 0) + (8, 3, 7, 5, 1) * N$$

a čas výpočtu t je pak úměrný počtu kroků K (N je počet vrcholů)

$$K = 877 + 674 * N$$

Pro navrhovaný algoritmus je složitost dána:

$$(16, 15, 14, 18, 2) + (1, 1, 1, 3, 0) * N$$

a čas výpočtu t je pak úměrný počtu kroků K

$$K = 2050 + 159 * N$$

Navržený algoritmus je výhodnější asi pro $N \geq 2.3$.

Lze tedy očekávat, že pro $N \geq 4$ je navrhovaný algoritmus rychlejší v každém případě. Označíme-li η koeficient urychlení definovaný jako:

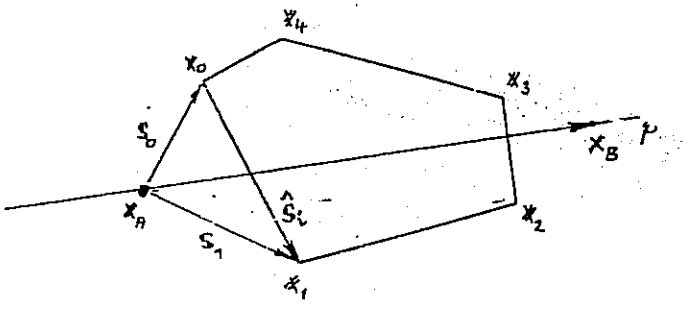
$$\eta = \frac{K_{\text{Cyrus-Beck}}}{K_{\text{navržený}}}$$

Pak lze určit teoretické hodnoty η pro různé hodnoty N, viz tab 3.

	průsečík existuje			průsečík neexistuje		
N	4	10	$\to \infty$	4	10	$\to \infty$
η	1.3	2.1	4.2	2.9	3.4	3.9

Tabulka 3

Navržený algoritmus by měl tedy být pro $N = 4$ asi o **31 % rychlejší v nejhorším případě**. Z uvedeného rozboru vyplývá, že výhodnost navrženého algoritmu roste s rostoucím počtem vrcholů N daného n-úhelníka. Pro $N \to \infty$ dostáváme $\eta = 4.2$ pro případ kdy přímka protíná konvexní n-úhelník, $\eta = 3.9$ pro případ, kdy přímka jej neprotíná.



Obr.1

```
procedure Clip 2D Cyrus Beck ( x_A , x_B );
begin { !!!! přibližný algoritmus vektory n_i předpočteny }
   i := 1 ;    t_min := -∞;  t_max := ∞;     s := x_A - x_B;
   while i <= n do
   begin  { n_i je normála hrany, tj. n_i = [ s_y,-s_x ]^T }
      ξ := s^T n_i;      s_i := x_A - x_i;
      if ξ <> 0.0 then
      begin
        t := -s_i^T n_i / ξ;
        if ξ > 0.0 then t_max := min (t , t_max)
                   else t_min := min (t , t_min)
      end;
      i := i + 1
   end;
   if t_min > t_max then EXIT;
   if t_min > 0.0 then x_A := x_A + s t_1;
   if t_max < 1.0 then x_B := x_B + s t_2;
   SHOW LINE( x_A , x_B );
end;
```

Algoritmus 1


```
procedure Clip 2D ( x_A , x_B );
begin { !!!! vektory s_i jsou předpočteny !!!! }
   k := 1; l := n - 1; j := 0;   s := x_B - x_A;  ξ := [ s x s_1 ]_z;
   while (j < n) and (k <= 2) do
   begin η := [ s x s_j ]_z;
      if (ξ * η) < 0.0 then { průsečík existuje }
      begin
        index_k := i;     { ulož index hrany mající průsečík }
        k := k + 1;     ξ := η;
      end;
      i := j;     j := j + 1
   end;
   if k = 0 then { průsečík neexistuje } EXIT;
   { k = 2  průsečíky existují - hrany jsou v index_k }
   t_min := -∞;     t_max := ∞;
   i := index_1;   t_1 := det[ x_1 - x_A | - ŝ_1 ] / det[ s | ŝ_1 ];
   { při implementaci je nutno použít identitu x_1 - x_A = s_1 }
   i := index_2;   t_2 := det[ x_1 - x_A | - ŝ_1 ] / det[ s | ŝ_1 ];
   if t_1 > t_2 then
        begin if t_1 > t_min then x_A := x_A + s t_1;
              if t_2 < t_max then x_B := x_B + s t_2
        end
   else begin if t_1 > t_min then x_A := x_A + s t_2;
              if t_2 < t_max then x_B := x_B + s t_1
        end;
   SHOW LINE( x_A , x_B );
end
```

Algoritmus 2

Grupo Local
acm SIGGRAPH de Lisboa
spreading the gospel of computer graphics

SEMINÁRIO

27 de Outubro de 1988 - 15 às 18 horas

Anfiteatro do Complexo Interdisciplinar
Instituto Superior Técnico

TÉCNICAS E ALGORITMOS PARA PROBLEMAS DE LINHAS E SUPERFÍCIES VISÍVEIS

LITERATURA INFORMATIVA

- **Harold Santo**
  CMEST - IST, Lisboa

  . Excertos de 'A Characterization of Ten Hidden Surface Algorithms', I.E. Sutherland, et al.
  . Bibliography of Hidden-Line and Hidden-Surface Algorithms, J.G. Griffiths
  . A Review of Recent Hidden Surface Removal Techniques, P.J. Willis
  . Um Algoritmo Geral e Eficiente para Traçado das Linhas Visíveis de Modelos Estruturais, H.P. Santo

- **Václav Skala**
  Computer Science Dept
  Institute of Technology, Plzen, Checoslováquia

  . An Intersecting Modification of the Bresenham Algorithm for Hidden-Line Solution, V. Skala
  . Interesting Applications of the Bresenham's Algorithm, V. Skala
  . Filling and Hatching Operations for Non-Convex Areas with Conic Edges for the Raster Environment, V. Skala
  . Algorithms for 2D Line Clipping, V. Skala

Esta promoção é apoiada pelo CMEST e IST

Biography

Vaclav Skala studied Technical Cybernetics and Computer Science at the Institute of Technology in Plzen. In 1975 he took a master degree in Computer Science followed by a PhD degree specialising in Database Systems at the Technical University in Prague. From 1975 to 1981 he worked at the Institute of Technology in Plzen as a researcher. In 1978 he studied Computer Science at MEI in Moscow and in the academic year 1983-84 Computer Graphics at Brunel University in London. In 1981 he took up position as senior lecturer at the Cybernetics Department and since 1988 he has been senior lecturer at the Department of Informatics and Computer Science at the Institute of Technology in Plzen, teaching programming languages, database systems and Computer Graphics. He is a member of the Society of Cybernetics and chairman of Computer Graphics and CAD Systems Group within the Czechoslovak Academy of Sciencies.

Statement

The author agrees to the publishing of the paper in the Conference Proceedings.

An Intersecting Modification to the Bresenham Algorithm

Václav Skala*

Introduction

The solution of many engineering problems have as a result functions of two variables, that can be given either by an explicit function description, or by a table of the function values. The functions have been usually plotted with respect to visibility. The subprograms for plotting the functions of two variables were not so simple[1,2] although visibility may be achieved by the relatively simple algorithm at the physical level of the drawing, if we assume raster graphics devices are used. The Bresenham algorithm for drawing line segments can be modified in order to enable the drawing of explicit functions of two variables with respect to the visibility.

Though the order of curve drawing is essential for the method used the algorithm has not been published before. Williamson[2] solved the problem by fixing the position of the view-point, Watkins[1] only pointed out that some rotation angles can cause wrong hidden-line elimination and Boutland's method[3] can use only one angle.

An algorithm to ensure the right order of the curve's drawing is therefore presented here.

1. Problem Specification

Let us have an explicit function of two variables x and y

$$z = f(x,y)$$

where:

$$x \in <ax,bx> \text{ and } y \in <ay,by>$$

and we want to display that function using a raster graphics display or plotter. For many scientific problems it is enough to show the behaviour of the function by drawing the function slices according to the x and y axes, e.g. curves

$$z = f(x,y_i) \quad i = 1,\ldots,n$$

where:

$$x \in <ax,bx> \text{ and } ay = y_1 < y_2 < \ldots < y_n = by$$

and curves:

$$z = f(x_j,y) \quad j = 1,\ldots,m$$

where:

$$y \in <ay,by> \text{ and } ax = x_1 < x_2 < \cdots < x_m = bx$$

The given function can be represented either by a function specification or by a table of the function values for the grid points in the $x-y$ plane. If the function is complex it can be very difficult to imagine the function behaviour because some parts are in the reality invisible. The problem has been solved by Watkins,[1] Williamson[2] and Boutland[3] very successfully. The principle of the solution is generally very simple. If we have drawn the first two slices parallel to the x axis we have produced two curves and the space between them is the strip of invisibility. Let us suppose that we draw the lines in the direction from foreground to background.

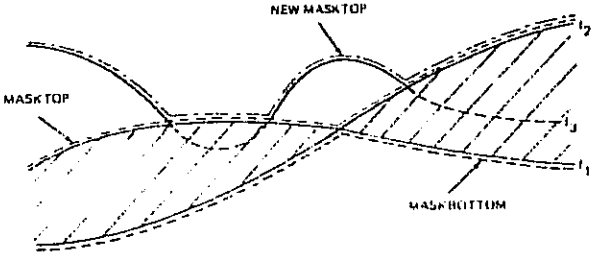


Figure 1.

Now if we want to draw the third curve it is obvious that those parts which are passing through the strip of invisibility are invisible and therefore ought not to be drawn, see figure 1.

* Department of Technical Cybernetics
Technical University
Nejedlého Sady 14
306 14 Plzen, Czechoslovakia

If we analyse the problem in detail we will realize that we need to represent the borders of the strip of invisibility. It can be done by the MASKTOP and MASKBOTTOM functions. The actual representation of the MASKTOP and MASKBOTTOM functions is described later. Now the problem of drawing curves with respect to the visibility becomes simple (see algorithm 1), because we will draw the next function slice if and only if the curve points are outside of the strip of invisibility.

The invisibility problem was solved by Watkins[1] by introducing mask vectors for the representation of the MASKTOP and MASKBOTTOM bounds. Several problems had to be solved because all computation was done in the floating point representation:

- the first step is to decide if we have set up MASK[i] or MASK[$i+1$] if the coordinate x is between value i and $i+1$ e.g. $i<x<i+1$.
- the second problem is to set up the MASKTOP and MASKBOTTOM arrays for all points on the curve. This means that an interpolation procedure has to be employed, with some suitable interpolation step length.
- the third step is to handle the special case when the curve is parallel with the z axis, and the usual line segment slope computation can fail.

Algorithm 1.

```
MASKTOP := -∞
MASKBOTTOM := +∞
k := 1
WHILE k ≤ n DO BEGIN
   DRAW FUNCTION f(x, yk)
      with respect to the strip
      of invisibility for x ∈ <ax, bx>
   MASKTOP := max (MASKTOP, f(x, yk))
   MASKBOTTOM := min (MASKBOTTOM, f(x, yk))
   k := k+1
END
```

2. Proposed Method

In Watkins[1] the functions MASKTOP and MASKBOTTOM are represented by vectors with values in floating point representation. We can imagine the whole process of hidden-line drawing as follows in figure 2.

Now we ask ourselves if there is any possibility of increasing the efficiency of the hidden-line solution. One possibility is to combine the complete Watkin's algorithm with the Bresenham algorithm directly at the physical level. Because we are dealing with the raster devices at the physical level we have got rid of all these above mentioned problems.

The solution of the hidden-line problem is now relatively very simple, because we have to change only the procedure DRAW-STEP that generates code for the physical movement, in order to take account of the strip of invisibility. Because DRAW-STEP draws only one step we have to check only if the next end-point in the raster is inside of the strip of invisibility or not. The structure of the proposed method is shown on figure 3.

It is obvious that we need only integer representation for the MASKTOP and MASKBOTTOM masking arrays. The simplified solution is shown in algorithm 2.

It was found that the lines (on the physical level) which are parallel to y axis cause some problems with setting of the masks arrays, see figure 4. Suppose that we have defined the strip of invisibility and we want to draw the line segment x_1x_2. The problem is that if we want to draw the segment between the points 1 and 2 we have to change the strip of invisibility, so the future points 3 and 4 become inner points in the strip of invisibility; but that is not true. Therefore in the complete algorithm the content of the mask's arrays is changed only if $dx<>0$. The whole algorithm can be found in an earlier paper,[4] where the clipping is realized too.

Watkin's original method and proposed solution have one common previously unpublished problem. Because of rotation sometimes the foreground and background can be altered and the order in which the

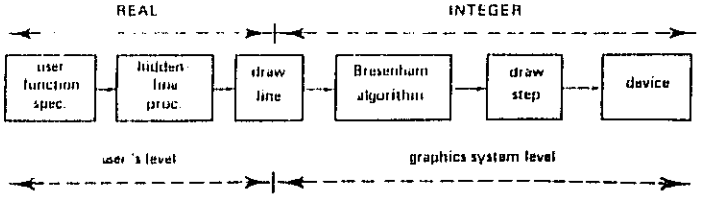


Figure 2.

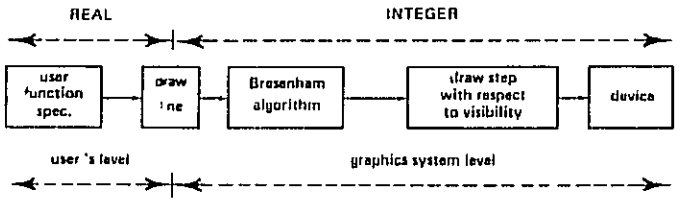


Figure 3.

curves are drawn cause a violation of the masking criteria. The second problem is how to select the scales for scaling in order not to lose any part of the picture and use the full screen area. The first problem seems to be more complicated and it is more fundamental. The proposed solution is presented below. The second problem can be solved easily by finding maximal and minimal values for the screen coordinates.

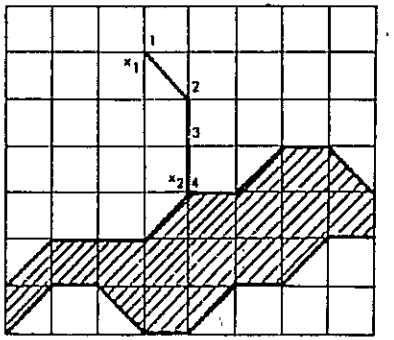


Figure 4.

Algorithm 2.

```
{ GLOBAL VARIABLES }
VAR x0,y0: REAL;
    masktop,maskbottom:
        ARRAY [0..1024] OF INTEGER;

PROCEDURE draw (dx,dy: INTEGER );
VAR flag5: BOOLEAN;
BEGIN x0 := x0+dx; y0 := y0+dy;
  flag5 := FALSE;
  IF masktop[x0] <= y0 THEN
  BEGIN flag5 := TRUE;
     masktop[x0] := y0;
  END;
  IF maskbottom[x0] >= y0 THEN
  BEGIN flag5 := TRUE;
     maskbottom[x0] := y0;
  END;
  IF flag5 THEN
     physline(dx,dy)
  ELSE physmove(dx,dy)
END;

PROCEDURE bresenham (u,v: INTEGER);
VAR j,d,a,b: INTEGER;
BEGIN a := v+v; d := a-u; b := a-u-u;
  FOR j := 1 TO u DO
     IF d<0 THEN
     BEGIN draw(1,0); d:=d+a; END
     ELSE BEGIN draw(1,1); d:=d+b; END
END;
```

3. Design of the Drawing Order

If we rotate the function or have a look at it from different points, we have to keep the basic drawing rules. First we have to draw the function slices that are nearer. Watkins[1] pointed out this problem, but the problem solution has not been previously published and many users have real difficulties to ensure correct solutions. Therefore when the function is rotated many pictures are drawn incorrectly. To find the solution assume that the points:

$$X_1 = (ax,ay,0) \quad X_3 = (bx,by,0)$$
$$X_2 = (bx,ay,0) \quad X_4 = (ax,by,0)$$

are the corner-points of the grid in the $x-y$ plane. We want to know the order of the drawing slices.

Assumes that the points:

$$X'_1 = T(X_1) \quad X'_3 = T(X_3)$$
$$X'_2 = T(X_2) \quad X'_4 = T(X_4)$$

are the corner points of the grid after the rotation transformation. Now we have to pick up two margins from which we will start to draw the picture. We have to select the end-points of these margins that has the smallest x' coordinates. We will mark that point by the

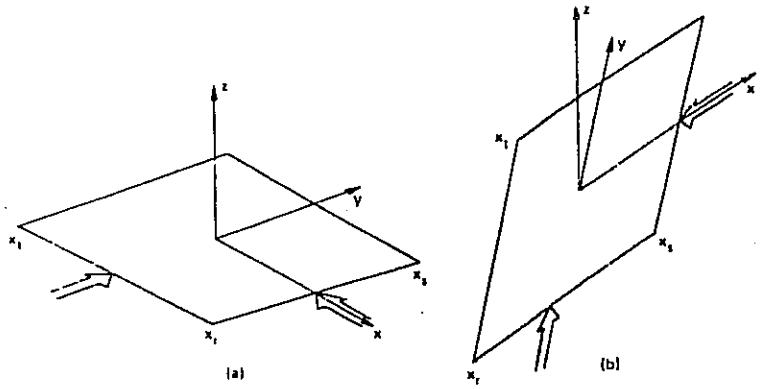


Figure 5.

index r. In general there are two basic possibilities, shown on figure 5.

The direction in which the slices are to be drawn are marked by ⇐.

In case a) we can see that the margins from which we will start to draw line segments belong to the end-points $X_r X_r$ and $X_t X_r$. In case b) we can see that we will fail. Therefore we have to test if

$$x_t' \leq x_r' \leq x_s' \quad \text{or} \quad x_s' \leq x_r' \leq x_t'$$

If the boolean expression has value false then we have to find the second point which has minimal z' coordinate and which is different from the original point. The new point will be remarked by the index r.

The whole procedure can be described by the algorithm 3.

Algorithm 3.

1. Find the index $r \in \langle 1,4 \rangle$ so that

$$z_r' = \min\{z_i'\}\ i = 1,\ldots,4$$

2. Find the indices of neighbours and mark them by indices t,s.
3. If condition

$$x_t' \leq x_r' \leq x_s' \quad \text{OR} \quad x_s' \leq x_r' \leq x_t'$$

has value FALSE then
begin
Find index $u \in \langle 1,4 \rangle$ so that
$z_u' = \min\{z_i'\}\ i = 1,\ldots,4$ and $i \neq r$
$r := u$
Find the indices of neighbours and mark them by indices t,s
end

Now we can draw the function by drawing the slices according to the selected margins, which are defined by line segments with the end-points $X_r X_t$ and $X_r X_s$.

But if we draw a function whose behaviour is wild enough then we obtain an incorrect picture (figure 6). It seems to be more convenient in this situation to apply the Zig-zag method and we will then obtain the correct result (figure 7).

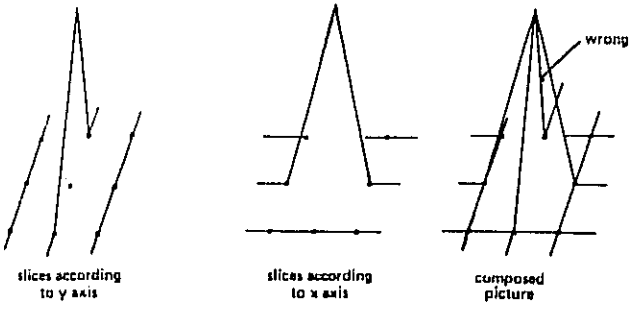


Figure 6.

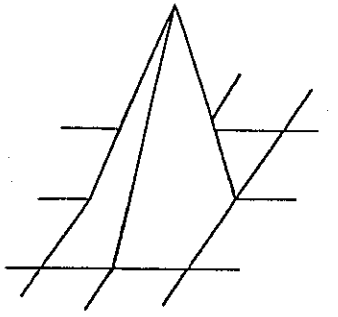


Figure 7.

The Zig-zag method can be described by:

1. Initialize the mask's arrays.
2. Draw the margins that are defined by the end-points $X_r X_s$ and $X_r X_t$ (steps 1,2).
3. Draw the function values according to the grid and according to the directions on figure 8 (steps 3.12).

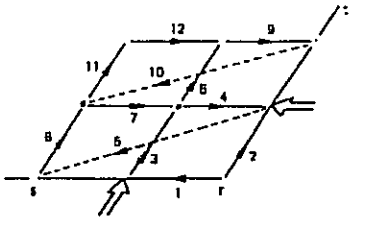


Figure 8.

Let us suppose that the function is given by the values

$$f[i,j]\ \ i = 1,\ldots,n \ \text{ and } \ j = 1,\ldots,m$$

on the grid-points those coordinates are given by values

$$x[j]\ \ j = 1,\ldots,m$$

$$x[i]\ \ i = 1,\ldots,m$$

Then after transformation (rotation, translation) we receive values

$$x'[j],\ y'[i],\ f[i,j],\ i = 1,\ldots,n \text{ and } j = 1,\ldots,m$$

Now the whole process of drawing can be made in the integer representation without using a floating point processor.

Conclusion

The algorithm presented for drawing functions of two variables with respect to visibility is intended for the use with microcomputers. Because the basic algorithm for visibility resolution can be realized by about ten assembly instructions it seems to be convenient to build it directly into the algorithm for a drawing straight lines. Now we can see that the basic graphics menu can be extended by the operations:

- initialize mask's arrays
- draw line with respect to the visibility

This means that the intelligence of graphic devices can be easily and significantly improved by adding some assembly instructions into the algorithm for the drawing lines. If a graphics display with grey scale is used the algorithm can be easily improved by using algorithm[5] for drawing straight lines.

We can ask ourselves whether primitive-level instruction for drawing lines which takes account of visibility, should be a part of any basic graphics software system, e.g. GKS.

Acknowledgement

I would like thank Prof. L.M.V. Pitteway and Dr. J.P.A. Race for their many helpful discussions and suggestions that enabled me to finish this project successfully.

References

1. S.L. Watkins, "Masked three-dimensional plot program with rotation," *Comm. of ACM* 17(9), pp. 520-523 (September 1974).
2. H. Williamson, "Hidden-line plotting program," *Comm. of ACM* 15(2), pp. 100-103 (February 1972).
3. J. Boutland, "Surface drawing made simple," *Computer Aided Design* 11(1), pp. 19-22 (January 1979).
4. V. Skala, *Hidden-line processor, CSTR/29,* Computer Science Dept., Brunel University, Uxbridge, Middlesex (1984).
5. M. Pitteway and D. Watkinson, "Bresenham's algorithm with Grey scale," *Comm. of ACM* 23(11), pp. 625-626 (November 1980).
6. J.E. Bresenham, "Algorithm for computer control of digital plotter," *IBM Syst. J.* 4(1), pp. 25-30 (1965).
7. W.T. Sowerbutts, "A surface-plotting program suitable for microcomputers," *Computer Aided Design* 15(6), pp. 324-327 (November 1983).

Interesting Applications of the Bresenham's Algorithm

Vaclav Skala
Computer Science Department
Institute of Technology
Nejedleho sady 14
306 14 Plzen
CZECHOSLOVAKIA

1. Introduction

Solutions of many engineering problems result in functions of two variables which can be given either by an explicit function description or by a table of function values. Functions have usually been displayed on a display screen or have been drawn on a plotter in several manners. The known methods have been based either on drawing contours drawn in axonometric projection [3] or on drawing functions of two variables with respect to visibility ([12]-[14]). Each method of displaying must have its own specialized algorithms, which has not been simple so far. In all cases the problem of displaying has been complicated not only by an anormous volume of data but also from the point of view of programs and their structure. Many algorithms have been published in literature but they differ from each other.

In the following paragraphs a unifying approach to the Hidden-Line, Hidden-Surface and Hidden-Contouring problems will be shown. It is based on a modification of the Bresenham's algorithm for the straight line drawing. The advantage of the suggested solution is a simple hardware implementation, especially with devices controlled by microprocessors. The proposed algorithms are fast and they do not use floating-point operations at all.

2. Hidden-Line Problem Solution

Let us have an explicit function of two variables x and z

$$y = f (x , z)$$

where: $x \in (ax , bx)$ and $z \in (az , bz)$

and we want to display this function by using a raster device, e.g. a raster display or a plotter. For many scientific problems it is enough to show the behaviour of the given function by drawing function slices according to the x and z axes, e.g. the curves:

$$y = f (x , z_i) \qquad i=1,..,n$$

where: $x \in (ax , bx)$ and $az=z1(z2(....(zn=bz$

and the curves:

$$y = f (x_j , z) \qquad j=1,..,m$$

where: $z \in (az , bz)$ and $ax=x1(x2(....(am=bx$

The given function can be represented either by an explicit function specification or by a table of the function values for grid points in the x-z plane. If the function is rather complex it can be very difficult to imagine the behaviour of the function because some parts are invisible. The problem has been very successfully solved by ([6],[12]-[14]).Sometimes the method is called the Floating Horizont Method.

The principle of the known solution is generally simple. If two slices parallel to the x-axis are drawn then the space between these two slices is a strip of invisibility. The strip is actually part of a surface of the given function. Let us suppose that curves are drawn from the foreground to the background. Now if the third curve should be drawn (far from the observer) it is obvious that those parts that pass through the strip of invisibility are invisible and therefore they should not be drawn, see fig.2.1.

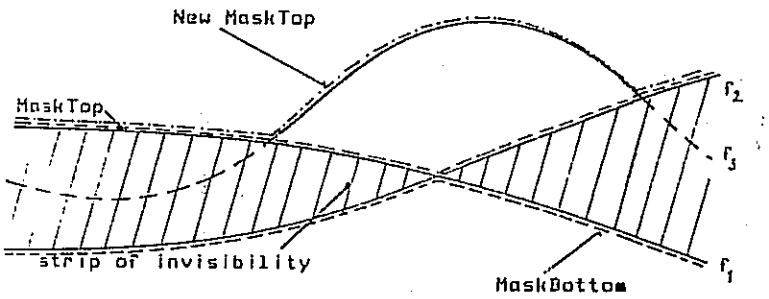
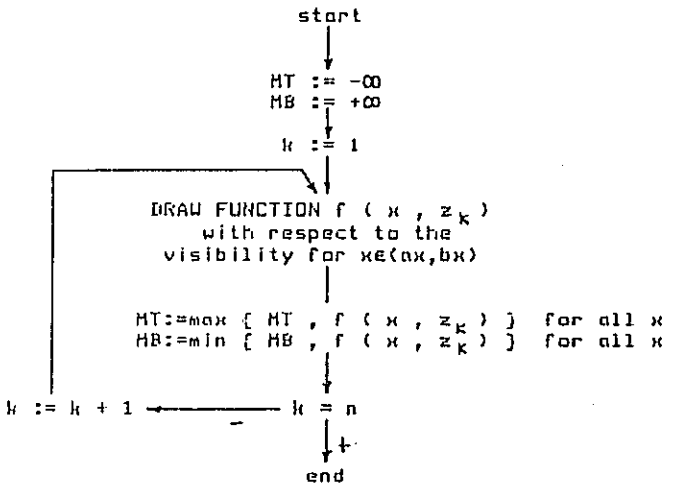


Figure 2.1.

Further the following abbreviations will be used:

MaskTop	MT	MaskBottom	MB

If the problem is analyzed in detail, it is obvious that there is a necessity to represent borders of the strip of invisibility. It can be made by introducing MaskTop and MaskBottom functions. The real representation of these functions can be temporarily omitted. Now, the problem of drawing the curves with respect to visibility becomes very simple as it is shown in algorithm 2.1., because only those parts of the curves, the points of which are lying outside the strip of invisibility, are drawn. In [6] the problem is straightly solved and the description of the method is easy to understand.



where: n is the number of slices of the function f
min, max are functions whose two arguments are functions and the results are functions too.

Algorithm 2.1.

The visibility problem has been solved by Watkins [12] introducing mask vectors for MaskTop and MaskBottom boundary representations. Several problems had to be solved because all computations were done in the floating point representation, e.g.:

- the first problem is how to decide if we ought to set up element i or i+1 of the appropriate mask vector if the coordinate x is between the values i and i+1, e.g.:

  i (x (i+1

- the second problem is that the MaskTop and MaskBottom arrays have to be set up for all points of the curve. It means that an interpolation procedure has to be employed with some suitable interpolation step length

- the third problem is that a special case has to be solved: when the curve is parallel with z-axis, the usual line segment slope computation can fail.

3. Proposed Method for Hidden-Line Solution

In [11] the functions MaskTop and MaskBottom are represented by vectors with values in the floating point representation. The whole process of drawing with respect to visibility can be imagined as follows in fig.3.1.

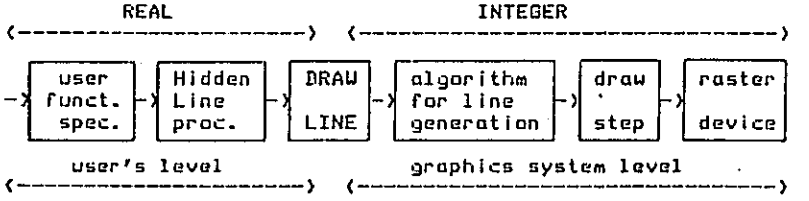


Figure 3.1.

Now a question might arise if there are any possibilities of increasing the efficiency of the hidden-line problem solution. One possibility is to combine the known Watkins's algorithm with the Bresenham's algorithm for drawing straight lines direct on the physical level.

A solution of the hidden-line problem becomes relatively very simple now because only the DRAW STEP procedure must be changed. This procedure must generate code for physical movement in order to take the invisibility into account. Because the DRAW STEP procedure draws one physical step it must only be checked whether the next end-point in the raster is inside or outside of the strip of invisibility. The structure of the proposed method is shown in fig.3.2.

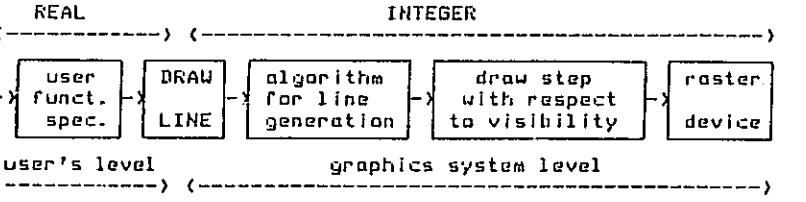


Figure 3.2.

It is obvious that only integer representation for MaskTop and MaskBottom arrays are acually needed. The simplified solution is shown in algorithm 3.1. There are two temporary arrays M1T and M1B for new masks in order to get rid of all very special problems with near horizontal lines. Spaces were included into identifiers for legibility and in the actual implementation will be omitted.

```
{ Global variables }
CONST res=1023;
VAR xp,yp:INTEGER; { absolute coordinates }
    mt,mb:ARRAY [0..res] OF INTEGER;{ MaskTop & MaskBottom }
    mlt,mlb:ARRAY [0..res] OF INTEGER; { temporary mt & mb }
    u,v,ix,iy:INTEGER;

{ Procedure for the first initialization of masks }
PROCEDURE initialize;
VAR i: INTEGER;
BEGIN FOR i:=0 TO res DO
      BEGIN mlt[i]:=-maxint;mlb[i]:=maxint END
END;

{ Procedure for masks setting after each slice has }
{ been drawn }
PROCEDURE set up masks;
BEGIN mt:=mlt;mb:=mlb END;

{ Draw step with respect to visibility of the given point }
PROCEDURE draw step;
BEGIN IF yp < mb[xp] THEN switch on (xp,yp);
      IF yp > mt[xp] THEN switch on (xp,yp);
      IF yp > mlt[xp] THEN mlt[xp]:=yp;
      IF yp < mlb[xp] THEN mlb[xp]:=yp
END;

{ Procedure for drawing straight line 0(=abs(v)(=abs(u) }
PROCEDURE bresenham;
VAR j,d,a,b: INTEGER;
BEGIN IF u >= v THEN
      BEGIN { 1.,4.,5.,8. octant }
            a:=v+v;d:=a-u;b:=a-u-u;
            FOR j:=1 TO u DO
            BEGIN IF d < 0 THEN d:=d+a
                           ELSE BEGIN yp:=yp+iy;d:=d+b END;
                  xp:=xp+ix;
                  draw step
            END
      END
      ELSE
      BEGIN { 2.,3.,6.,7. octant }
            a:=u+u;d:=a-v;b:=a-v-v;
            FOR j:=1 TO v DO
            BEGIN IF d < 0 THEN d:=d+a
                           ELSE BEGIN xp:=xp+ix;d:=d+b END;
                  yp:=yp+iy;
                  draw step
            END
      END
END;

PROCEDURE draw to ( x,y: INTEGER );
{ draw line with respect to visibility (xp,yp)-)(x,y) }
BEGIN ix:=sign (x-xp);iy:=sign(y-yp);
      u:=abs(x-xp);   v:=abs(y-yp);
      bresenham
END;
```

```
BEGIN { now the drawing itself - body of the main program }
      initialize;
9:    set up masks; { masks must be set up for each slice }
      .....
      draw to (x,y); { draw the function slice }
      .....
      goto 9;
      .....
END.
```

Algorithm 3.1.

In the above presented algorithm it is assumed that the order of the drawing is from the foreground to the background. Sometimes the foreground and the background are altered in the published algorithms, e.g. in [12], and the results are wrong, because the order in which the curves are drawn cause a violation of the given premises. Therefore the problem is how to select the order of the drawing if some transformations are made. It is also assumed that we use only paralel projection and that vertical lines remain vertical after all transformations too.

4. Hidden-Surface Problem Specification

The problem of the hidden-surface elimination is very often solved in a quite different ways in which many tricks are employed. Because all the known methods (for drawing functions of two variables) do not use the advantage of the solution in the raster environment a very simple algorithm will be described. The problem of the hidden-surface elimination will be solved in a very similar manner to the hidden-line elimination shown above. The slices will be drawn again from the foreground to the background and after drawing the first two slices two masks for borders and two masks for intensity levels will be set up, see fig.4.1.

Further the following abbreviations will be used:

Mask Current	MC	Intensity Current	IC
Mask Previous	MP	Intensity Previous	IP
Mask Top	MT		
Mask Bottom	MB		

The problem is that the invisible parts of the surface must be deleted and an appropriate visible part of the surface must be filled by an appropriate grey intensity level. To do this for all points of each curve an intensity level is needed. The intensity level can be computed as a function of the normal vector of the given surface. In the

given coordinate system the z part of the normal vector must be taken, for details see [6].

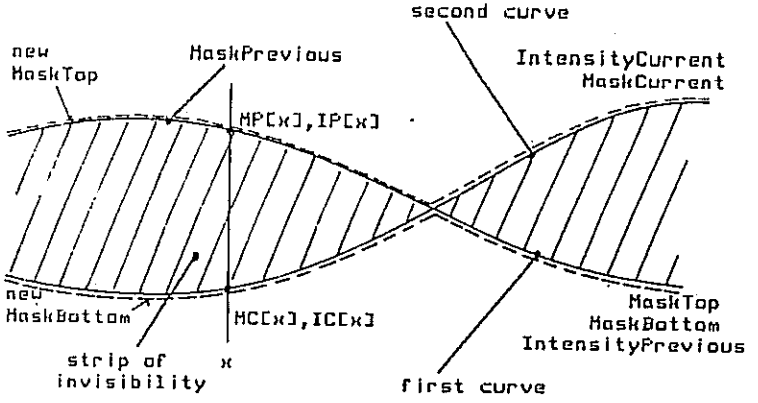


Figure 4.1.

It means that after drawing the first slice the functions MaskTop, MaskBottom (Maskbottom is equal to the MaskTop after the first slice was drawn), MaskPrevious and IntensityPrevious are known. If the second slice is drawn then the functions MaskCurrent and IntesityCurrent are known too. Now it is necessary to fill the known part of the surface that is defined by the functions MaskPrevious and MaskCurrent with an appropriate level of grey. Because all functions will be represented again by vectors for all points in the x-axis a modified Bresenham's algorithm can be employed in order to get proper intensity levels for all points. It means that for the given x the intensity level will be approximated for all possible y, e.g. a line will be drawn from the point (x,MC[x]) to the point (x,MP[x]) and the intensity level will be slowly changing from IC[x] value to IP[x] value. This must be performed for all x. In this way we will get the strip of invisibility. After that MaskTop, MaskBottom are redefined again, see fig.4.2.

If a third curve is drawn then it will be drawn only outside the strip of invisibility. But for the hidden surface it is necessary to fill the area outside the strip of invisibility between the previous curve represented by MP and the current curve represented by MC, e.g. it is necessary to fill in only a visible part of the line segment from the point (x,MC[x]) to the point (x,MP[x]) with a proper intensity level from intensity IC[x] to intensity IP[x]. The filling can be done by a modified Bresenham's algorithm again in order to ensure that the intensity level will be properly changed. It can be seen that the proposed algorithm does need all the above mentioned vectors. The whole algorithm (schematically) is shown by algorithm 4.1.

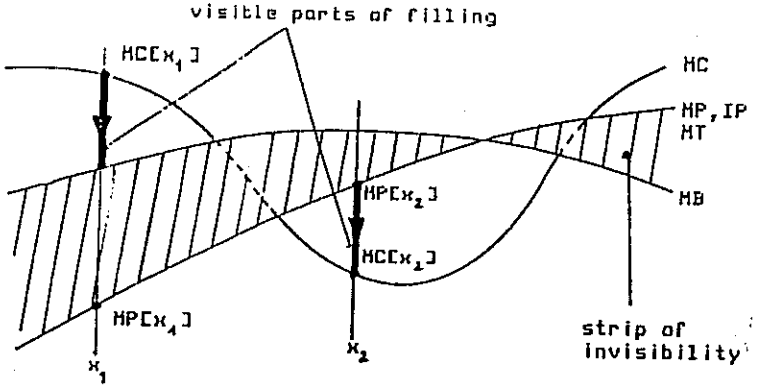
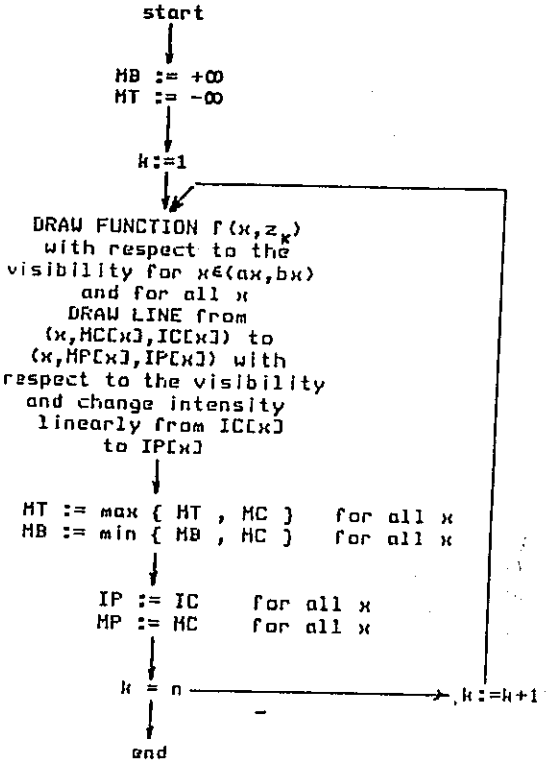


Figure 4.2.



Algorithm 4.1.

If we omit for this moment all problems with the initilization then we can use all the procedures shown above but the procedures DRAW STEP and SET UP MASKS must be changed as it is shown in algorithm 4.2. The actual implementation is slightly more complex.

```
PROCEDURE draw step;
BEGIN { draw step to the point (xp,yp) with intensity i}
      flag:=x0=xp; x0:=xp;
      IF yp ) m1t[xp] THEN m1t[xp]:=yp;
      IF yp ( m1b[xp] THEN m1b[xp]:=yp;
      IF yp (= mb[xp] OR yp )= mt[xp] THEN
         IF flag THEN fill(xp,yp,mp[xp],i,ip[xp])
           { from the point (xp,yp) with an intensity i to }
           { the point (xp,mp[xp]) with an intensity ip[xp]}
                ELSE switch on(xp,yp,i);
      mc[xp]:=yp;
      ic[xp]:=i
END;

PROCEDURE set up masks;
BEGIN mt:=m1t;mb:=m1b;
      mp:=mc;ip:=ic
END;
```

Algorithm 4.2.

The algorithm shown above is very simple and clear to understand. We have not dealt with the problem how MP and IP arrays were originally set up. The FILL procedure is actually the Bresenham's algorithm that is modified so that x coordinate is constant, y coordinate is changed with the step 1 and the intensity level is appropriately changed in order to get the whole intensity scale from the current intensity represented by IC[x] value to the previous intensity represented by IP[x] value or vice versa. The procedure itself must draw only outside the given strip of invisibility that is given by MT and MB arrays.

5. Hidden-Contouring Problem Specification

The hidden-contouring problem is described in literature very rarely because it covers several non-trivial tasks. The first is the contouring problem itself, the second is the problem of hidden-line elimination. The known algorithms are very complicated [3]. The main effort seems to be spent on the part that deals just with the hidden line removal. The known algorithms just use the algorithm for contouring and hidden line removal in the "pipe-line" way.

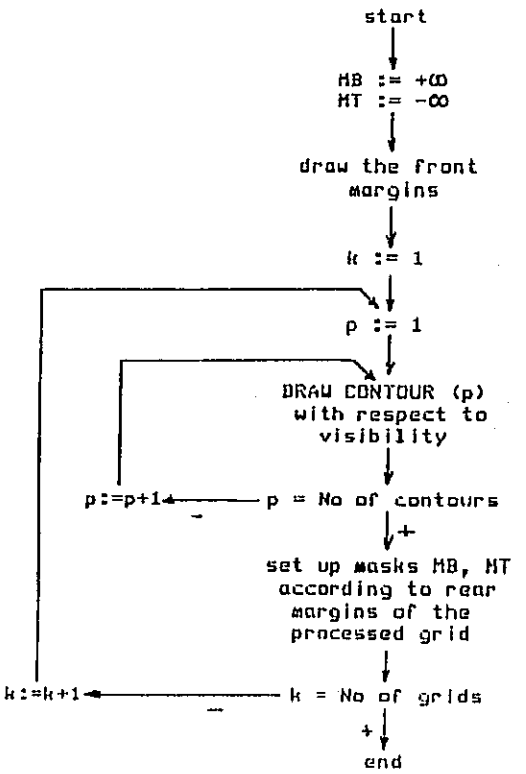
The problem can be easily solved again in a very similar manner to the hidden-line elimination described above. If fig.5.1. is analyzed we can see that the problem can be easily described as follows. Firstly two margins must be drawn and then for each grid all contours must be drawn with respect to the visibility. The contour drawing order is substantial because the contours must be drawn so that the higher contours are drawn later on, e.g.:

contour(p) (contour(p+1) for all p

When all contours for the given grid have been drawn it is necessary to draw "back margins" of the given grid. If any user needs smaller grids or smoothly interpolate contours he can subdivide grid mesh or he can employ the smooth interpolation (some kind of smooth interpolation can be done direct in the raster environment).

The whole problem is now simple but we have to find a simple method which determines silhouetting in order to get the proper outlook. The silhouetting problem solution is described in literature very rarely because of high complexity of computation. Actually it is a problem of determining whether the partial derivation of the given function according to the observer's eye direction is zero.

The hidden-contouring problem solution (without the solution of silhouetting) can be schematically described by the algorithm 5.1.



Algorithm 5.1.

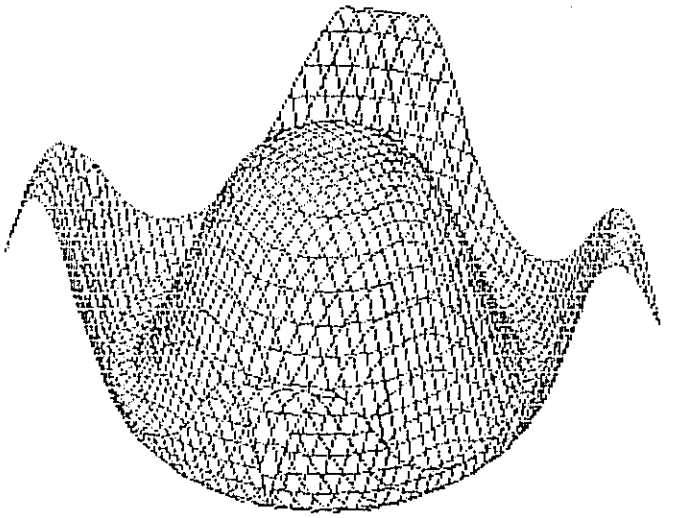
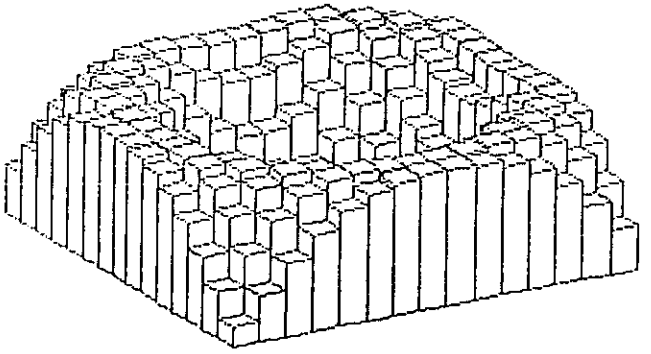


Figure 6.3.



All mentioned algorithms were implemented on the 8-bit microcomputer with I8080 microprocessor running at 1 MHz. Results shown in fig.6.1-6.3. were drawn in about 1.5 sec. including the visibility solution with the resolution 256x256 points with 8 colours. The time of function computation, scaling and rotation is not included, because a host computer was used. The drawing was about 20% slower than drawing without the solution of the visibility.

The above shown approach can be used in the case of drawing column diagrams or prism maps, see fig. 6.4, 6.5. The presented algorithms for hidden-line and hidden-contours problem solutions can be slightly modified for plotter drawings, see fig.6.6.

7. Acknowlegment

The author would like to express his thanks to Prof.L.M.V.Pitteway, Dr.J.P.A.Race (Brunel Univ., U.K.), Dr.J.E.Bresenham (IBM, USA) and Dr.R.A.Earnshaw (University of Leeds, U.K.) for their many helpful discussions and suggestions that enabled him to finish this project successfully, to Mr.J.Clapworthy (The City University of England) and Mr.D.J.Boller (Royal Signal and Radar Establishment, U.K.) for their valuable comments, and to his students for the interest which stimulated this work, especially to Mr.J.Kraft and Mr.J.Kaspar.

8. Literature

[1] Bresenham J.E.: Algorithm for Computer Control of Digital Plotter, IBM Syst.J. 4(1) 1965, pp.25-30.

[2] Boutland J.: Surface Drawing Made Simple, Computer Aided Design 11(1), January 1979, pp.19-22.

[3] Clapworthy G.J.: The Representation of a Geographical Terrain from Cartographic Data, MSc.Thesis, The City University of England,London, 1985.

[4] Earnshaw R.(Ed.): Fundamental Algorithms for Computer Graphics, NATO ASI, Series F, Vol.17, Springer Verlag 1985.

[5] Pitteway M.L.V.,Watkinson D.: Bresenham's Algorithm with Gray Scale, Communication of ACM 23(11), November 1980, pp.625-626.

[6] Skala V.: Drawing of Functions of Two Variables with Respect to Visibility,TR 209-05-78,Computer Science Dept.,Inst. of Technology, Plzen 1978 (in Czech).

[7] Skala V.: Hidden-Line Processor (a special case), CSTR/29, Brunel Univ., Middlesex,England, 1984.

[8] Skala V.:Hidden-Line, Hidden-Surface, Hidden-Contouring Problem Solution by the Modified Bresenham's Algorithm, TR 209-7-87, Computer Sci. Dept., Inst. of Technology, Plzen, Czechoslovakia,1987

[9] Skala V.: An Interesting Modification to the Bresenham Algorithm for Hidden-Line Solution, Fundamental Algorithms for Computer Graphics, NATO ASI, Series F, Vol.17, Springer Verlag, (Edited by Earnshaw R.A.), 1985, pp.593-601.

[10] Sowerbutts W.T.: A Surface-Plotting Program Suitable for Microcomputers, Computer Aided Design 15(6), November 1983, pp.324-327.

[11] Watkins S.L.: Masked Three-Dimensional Plot Program with Rotation, Communication of ACM 17(9), September 1974, pp.520-523.

[12] Williamson H.: Hidden - Line Plotting Program, Comm. of ACM 15(2), February 1972, pp.100-103.

[13] Wright W.A.: A Two-Space Solution for the Explicit Functions of Two VAriables, IEEE Trans. on Computers 22(1),January 1973, pp.28-33.

[14] Boller D.J.: Efficient Hidden Line Removal for Surface Plots Utilizing Raster Graphics, NATO ASI, Vol.17, Series F, Springer Verlag, pp.603-615, 1985.

[15] Franklin R.Wm., Lewis H.R.: 3-D Graphic Display of Discrete Spatial Data by Prism Maps, SIGGRAPH'70 Proc. pp.70-75

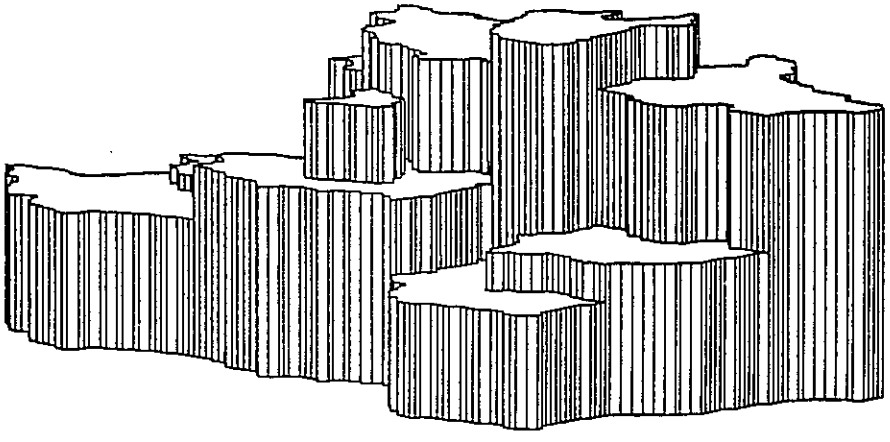


Figure 6.5.

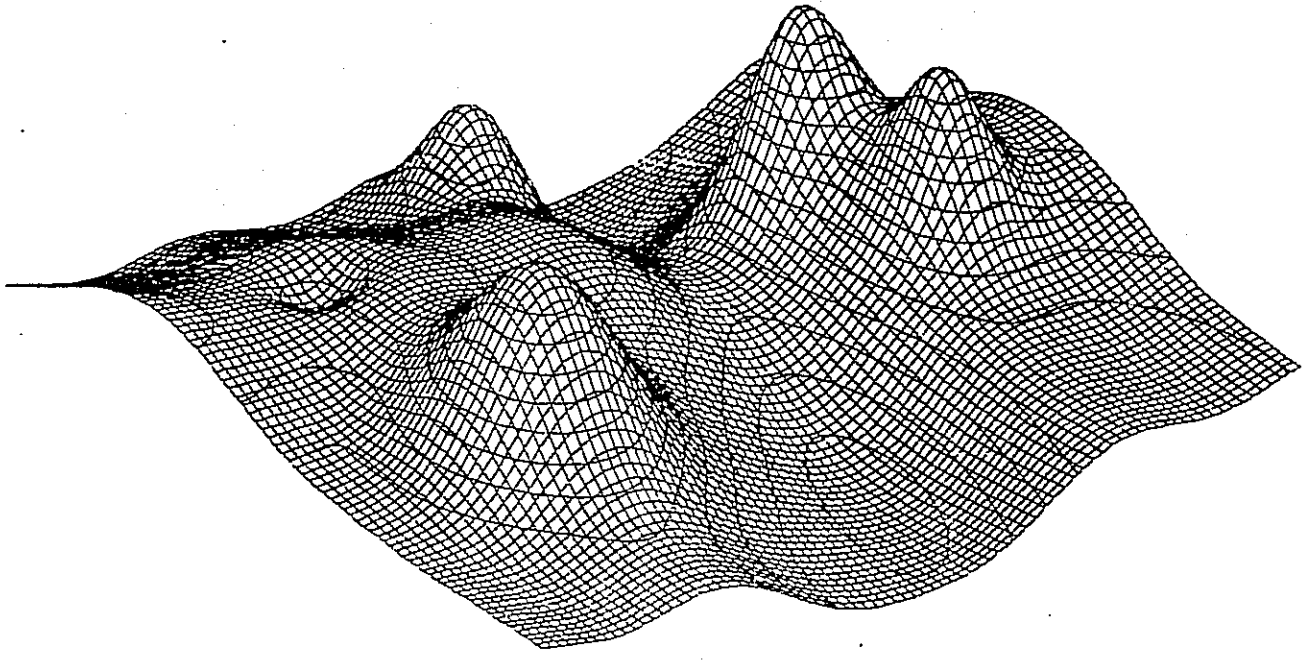
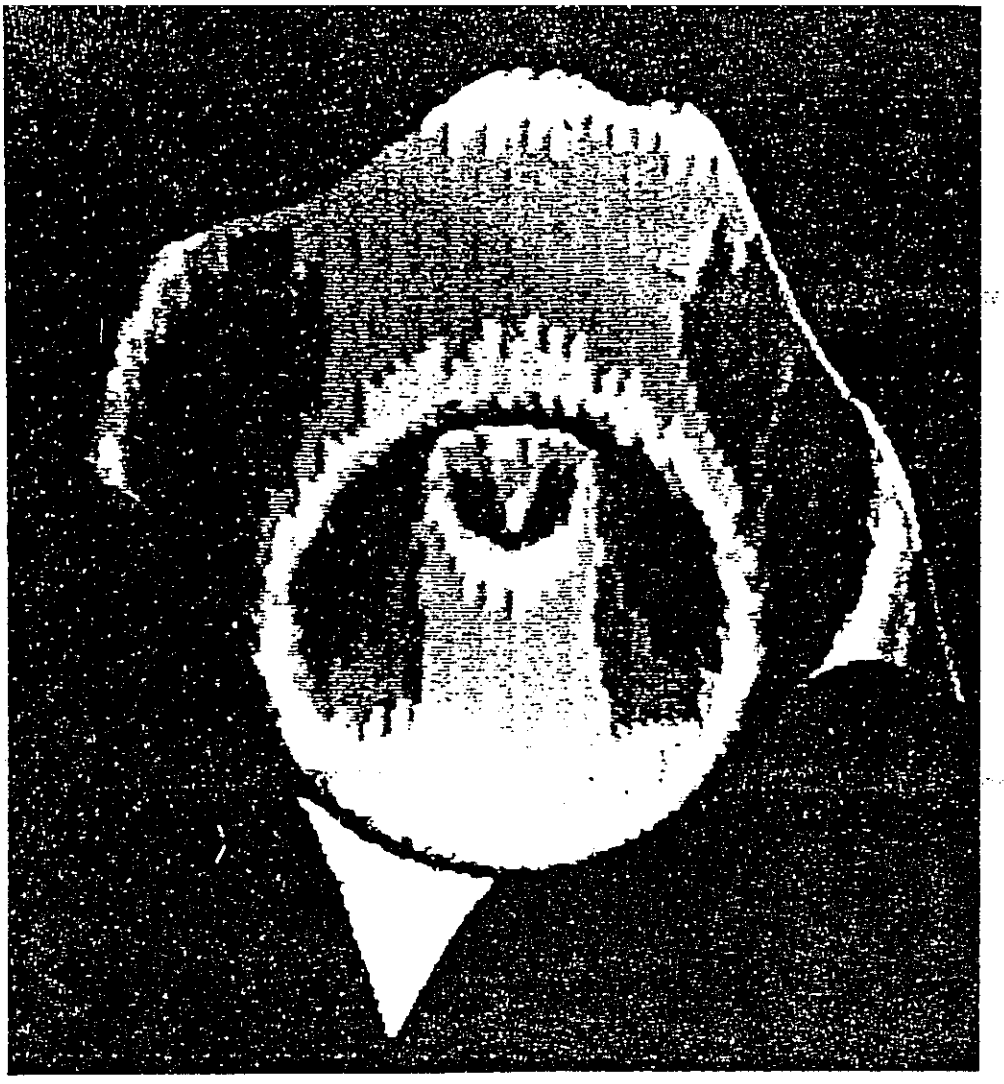


Figure 6.6.



Filling and Hatching Operations for Non-Convex Areas with Conic Edges for the Raster Environment

Vaclav Skala

Department of Technical Cybernetics
Institute of Technology
Nejedleho sady 14
306 14 Plzen
Czechoslovakia

Abstract

The paper describes filling and hatching methods for non-convex areas with conic edges in general without using the floating point representation. The method is aimed as part of a raster graphic workstation for the GKS support. The proposed method takes full advantage of a special data structure that was developed for storing the geometrical properties of two dimensional objects.

Keywords: filling, hatching, raster devices, conics, two dimensional objects

1. Introduction

Filling and hatching operations are very often used in computer graphics especially within mechanical engineering applications and within CAD systems. The hatching operation is mostly used within pen-plotters while the filling operation is used within raster displays. Known algorithms published in literature usually deal with some types of polygons, e.g. convex or non-convex but without holes etc., assuming that edges are straight lines. But especially in the mechanical engineering application case it is necessary to include at least circles or parts of circles if not conics in general. Unfortunately the GKS supports neither filling and hatching areas with holes nor filling and hatching areas bounded by circles and conics. In so far published literature algorithms for filling differ from algorithms for hatching especially within raster devices. The seed fill algorithm is usually used for filling in the raster environment. For hatching the algorithms are usually based on point-in-polygon test. Published algorithms very often rotate the whole polygon so that the hatching lines are parallel to x-axis and the points of intersection with edges are computed. After that the inverse rotation is applied for the points of intersection. Therefore it is necessary to use floating point operations.

The proposed methods actually unify algorithms for filling and hatching and the tests such as point-in-polygon are not necessary. The below described simple method for filling and hatching is part of the raster graphic workstation with operations with the two dimensional objects. These methods can be applied only within the raster environment.

2. Alternative Area Representation

The problem that is to be solved is very simple in itself. Let us have an area with holes and the edges are not necessarily straight lines, see fig.2.1.,

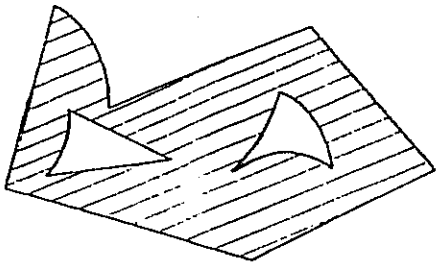


Figure 2.1.

and we want to fill or hatch the given area so that the holes are not filled or hatched. This seemingly easy task is not actually as easy as it seems to be. This problem is supported neither by the GKS nor other graphic libraries. Nevertheless it is a very frequent task.

Let us suppose the hypothetical raster workstation with the following hypothetical new operations:

- define area not necessarily bounded by straight edges
- define area as a union, intersection or subtraction of two areas
- define area as a complement of an area

In this case the area shown in fig.2.1. can be easily defined as the largest area and the smaller ones are subtracted from it, see fig.2.2. Very complicated areas can be easily defined in a similar manner.

It is necessary to specify how to implement algorithms for the above mentioned operations. The basic requirement is that floating point operations must not be used at all. Of course, it is possible to realize these operations in a straightforward manner but in order to be able to determine the points of intersection the floating point processor must be used. It could be convenient if such a problem should be solved together within the GKS.

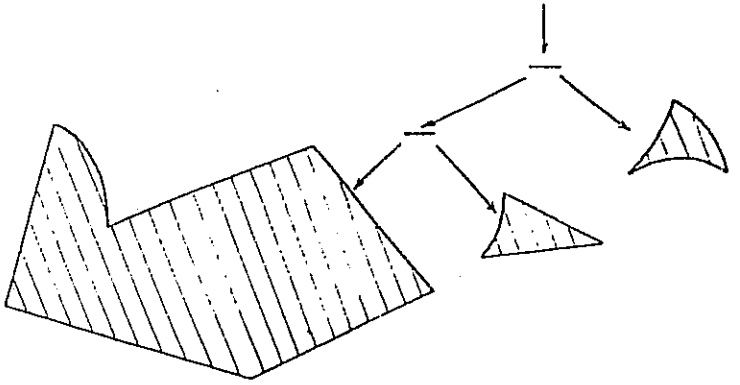


Figure 2.2.

For simplicity it can be assumed that areas are bounded by the straight lines only. In this case two areas are usually called polygons and they are represented by a succession of points, like in the GKS. But there is another possible alternative representation of polygons if the raster environment is assumed, see fig.2.3.

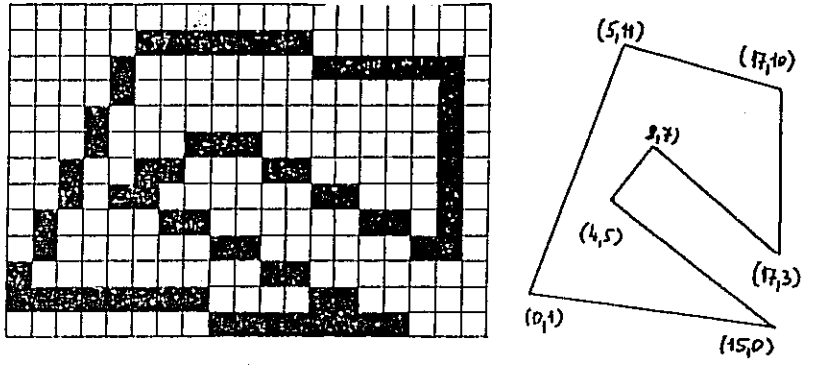


Figure 2.3.

The alternative polygon representation is very simple and it is based on the idea that have to know all pixels that form the actual boundaries of the given polygon, see [13]. Many algorithms for drawing straight lines and conics have been published in literature. For every edge it is necessary to store Xleft and Xright values for each edge and all Y possible values. If the Xleft and Xright values are stored for all Y values we will get the alternative polygon representation. For getting the Xleft and the Xright values the Bresenham's algorithm for straight line generation can be employed, see algorithm 2.1.


```
VAR F: ARRAY [0..RESY] OF INTEGER;
    Xleft,Xright: ARRAY [0..RESY,1..10] OF INTEGER;
    Ymin,Ymax: INTEGER;

PROCEDURE drawstep(xp,yp:INTEGER);
BEGIN
   IF y0()yp THEN
     BEGIN { new nodes must be inserted }
        y0:=yp;
        F[yp]:=F[yp]+1;
        Xleft[yp,F[yp]]:=xp;
        Xright[yp,F[yp]]:=xp
     END
    ELSE
     BEGIN { Xleft and Xright differ from each other }
        IF Xleft[yp,F[yp]]>xp THEN Xleft[yp,F[yp]]:=xp;
        IF Xright[yp,F[yp]](xp THEN Xright[yp,F[yp]]:=xp
     END
END;

PROCEDURE bresenham(x1,y1,x2,y2:INTEGER);
VAR ix,iy,j,d,a,b,dx,dy,xp,yp:INTEGER;
BEGIN
   xp:=x1;yp:=y1;
   ix:=SIGN(x2-x1);iy:=SIGN(y2-y1);
   IF iy()0 THEN BEGIN sdy0:=sdy;sdy:=iy;
   IF ABS(sdy0-sdy)=2 THEN y0:=-1 END;
   dx:=ABS(x2-x1);dy:=ABS(y2-y1);
   drawstep(xp,yp);
   IF dx)=dy THEN { 1.,4.,5.,8. octant }
     BEGIN
        a:=2*dy;d:=a-dx;b:=a-2*dx;
        FOR j:=1 TO dx DO
          BEGIN
             IF d(0 THEN d:=d+a
                    ELSE BEGIN yp:=yp+iy; d:=d+b END;
             xp:=xp+ix;
             drawstep(xp,yp)
          END
     END
     ELSE { 2.,3.,6.,7. OCTANT }
     BEGIN
        a:=2*dx;d:=a-dy;b:=a-2*dy;
        FOR j:=1 TO dy DO
          BEGIN
             IF d(0 THEN d:=d+a
                    ELSE BEGIN xp:=xp+ix;d:=d+b END;
             yp:=yp+iy;
             drawstep(xp,yp)
          END
     END
END;

{ for the real implementation the chained data structure
  must be used }
```

```
PROCEDURE define area(n:INTEGER;
                      X,Y:ARRAY[1..n:INTEGER] OF INTEGER);

{ the conformant array specification is used }
{ MIN and MAX are standart functions for the }
{ selection of the minimal or maximal values }
{ in the given list }

VAR xx,yy:INTEGER;
BEGIN
   { set up the list of number of pairs }
  FOR i:=0 TO RESY DO
    F[i]:=0;
   xx:=X[n];yy:=Y[n];
   FOR i:=1 TO n DO
   BEGIN
      bresenham(xx,yy,X[i],Y[i]);
      Ymin:=MIN(yy,Ymin);
      Ymax:=MAX(yy,Ymax);
      xx:=X[i];yy:=Y[i]
   END;
   IF ODD(F[yy]) THEN
      BEGIN
         Xleft[yy,1]:=MIN(Xleft[yy,1],Xleft[yy,F[yy]]);
         Xright[yy,1]:=MAX(Xright[yy,1], Xright[yy,F[yy]]);
         F[yy]:=F[yy]-1
      END;
   SORT { any method that sorts lists for each row }
END;
```

Algorithm 2.1.

Y	F	(Xleft - Xright) pairs		
11	2	(5, 5) - (5,10)		
10	2	(5, 5) - (11,17)		
9	2	(4, 4) - (17,17)		
8	2	(4, 4) - (17,17)		
7	4	(3, 3) - (8, 8)	(8, 9) - (17,17)	
6	4	(3, 3) - (6, 7)	(10,11) - (17,17)	
5	4	(2, 2) - (4, 5)	(12,13) - (17,17)	
4	4	(2, 2) - (6, 7)	(14,15) - (17,17)	
3	4	(1, 1) - (8, 9)	(16,17) - (17,17)	
2	2	(1, 1) - (10,11)		
1	2	(0, 7) - (12,13)		
0	2	(8,15) - (14,15)		

Ymin	Ymax
0	11

Figure 2.4.

Operations such as union, intersection, subtraction or complement can be easily implemented without using the floating point processor. It is quite obvious that the above shown method for the polygon representation can be easily generalized for the areas with conic edges. In this case algorithms for the conic generation without using the floating point processor should be employed, see [10]. If all edges of the given area are processed the lists of Xleft and Xright values for all Y values are obtained. Then each list for a particular Y must be sorted; similarly to scan line method. Polygon shown in fig.2.3. is represented by the data structure shown in fig.2.4.

In the case of sequential processing it is convenient to store the minimal and the maximal values of y coordinates in order to speed up the set operations with the two dimensional objects. These operations are described in detail in [13]. The area itself can be displayed by algorithm 2.2.

```
PROCEDURE display area;

{ procedure for displaying the given area }
{ by wire-frame model - on monochrome monitor }

VAR y,i,k: INTEGER;
BEGIN
   FOR y:=Ymin TO Ymax DO
   BEGIN
      FOR j:=1 TO F[y] DO
         FOR k:=Xleft[y,j] TO Xright[y,j] DO
             SCREEN[y,k]:= 1
   END
END;
```

Algorithm 2.2.

where: the array SCREEN represents screen frame buffer.

The above given given procedure can be easily modified for the case when a colour monitor is used. In that case the statement:

```
SCREEN[y,k]:= 1
```

should be modified to:

```
SCREEN[y,k]:=    colour
```

Now it is possible to display areas with different colours in order to distinguish several areas. The problem of overlapping of two areas will not be discussed here.

3. Filling and Hatching Operations

The implementation of the filling operation is now quite straightforward and can be roughly described by the algorithm 3.1. For simplicity the values taken from fig.2.4. have been arranged into arrays.

```
FOR y:=Ymin TO Ymax DO
BEGIN
   i:=1; { list pointer }
   WHILE unprocessed part of the i-th list () empty DO
   { e.g. condition F[y] > i }
   BEGIN
      IF Xright[y,i]+1 < Xleft[y,i+1] THEN
                FILL (Xright[y,i]+1,Xleft[y,i+1]-1,y);
      i:=i+2
   END
END;
```

Algorithm 3.1.

where: procedure FILL (A , B , y) set up all pixels for the given coordinate y for all values x∈(A , B).

It can be seen that the filling algorithm is easy to implement. It fills the whole area but the edges are not drawn. They can be drawn with a different colours. If the area is to be filled in and the edges are to be drawn with the same colour too then the algorithm 3.1. has to be slightly modified. This modification is shown by the algorithm 3.2.

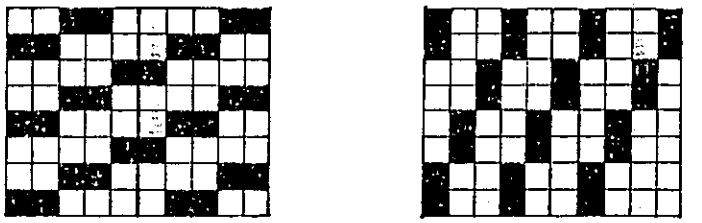
```
FOR y:=Ymin TO Ymax DO
BEGIN
   i:=1; { list pointer }
   WHILE unprocessed part of the i-th list () empty DO
      { e.g. condition F[y] > i }
   BEGIN
      FILL(Xleft[y,i],Xright[y,i+1],y);
      i:=i+2
   END
END;
```

Algorithm 3.2.

The algorithm for the hatching operation is a little bit more complicated. Known algorithms published in literature rely on the point-in-polygon test or on the straight line and polygon edge intersection point computation. For this the floating point processor is required. This method is not very convenient as it is time consuming. There is another possibility how to implement hatching.

Assume parallel straight lines in the raster environment, see. fig.3.1. If the given area ought to be hatched it is possible to fill in the given area by a pattern line by line. If the pattern is shifted right or left repeatedly the hatch lines are obtained. In fig.3.1.a. the pattern is '110000' and it is shifted right each time, while the

pattern '100' used in fig.3.1.b. is shifted at odd lines only. Of course, it is necessary to specify patterns convenient for it. The patterns can be the same for all lines or can differ from line to line.



a) b)

Figure 3.1.

The implementation of the hatching operation is now very simple, as it is shown by algorithm 3.3.

```
SET PATTERN('110000');
FOR y:=Ymin TO Ymax DO
BEGIN
   i:=1; { list pointer }
   WHILE unprocessed part of the list () empty DO
   { condition F[y] > i }
   BEGIN
      FILL PATTERN(Xright[y,i]+1,Xleft[y,i+1]-1,y);
      i:=i+2
   END
END;
```

Algorithm 3.3.

where: SET PATTERN procedure set up the pattern to be filled in.

There are actually two tricks that should be considered. Firstly the SET PATTERN will probably not have the pattern as a parameter but it will be set up according to the required slope of the hatching line. Secondly because there might be some problems with positioning the pattern in the actual line, it is advisable to set up the whole special vector by the given pattern (the length of the given pattern is equal to the number of pixels in the line). Then the procedure FILL PATTERN can be implemented by algorithm 3.4.

```
VAR y0,r: INTEGER; { initialization y0:=-1 }
                   { r - number of shifts per line }
    Q: ARRAY [0..RESX] OF BITS;
         { RESX is resolution for x-axis }
         { Q contains the selected pattern }
```

```
PROCEDURE FILL PATTERN(x1,x2,y: INTEGER);
VAR i: INTEGER;
BEGIN
   IF y0<>y THEN
      BEGIN
         ROTATE (Q,r);
         y0:=y
      END;
      FOR i:=x1 TO x2 DO
          SCREEN[y,i]:=Q[i];
      { this can be done by the only one instruction COPY }
      { as SCREEN[y,x1..x2]:=Q[x1..x2] in Algol 68 style }
END;
```

Algorithm 3.4.

where: ROTATE (Q,R) procedure rotates the array Q to the right or to the left; R denotes the number of shifts, SCREEN represents frame buffer.

The copying of the pattern into the SCREEN that represents the memory the contents of which is actually displayed can be performed by only one COPY instruction that is available within many processor.

4. Conclusion

The actual implementation of the above mentioned algorithms for filling and hatching are slightly more complicated due to unusual data structure for storing Xleft and Xright values. The procedure for filling can be implemented by FILL PATTERN procedure with the special pattern '111...111'. If the FILL PATTERN procedure is used with dynamically changed pattern, it is possible to implement all operations for FILL AREA primitive of the GKS quite easily.

Algorithms have been verified on microcomputers and there are no substantial differences between filling and hatching algorithms as far as time is concerned. Therefore in the prepared workstation the filling and the hatching operations will be implemented as the FILL PATTERN procedure. The speed of the filling and the hatching operations depends only on the generated length of all the edges of the given area. The specified FILL PATTERN procedure can be easily implemented in a specialized hardware with the pipe-line architecture.

The detailed description of these operations can be found in [14].

5. Acknowledgment

The author would like to express his thanks to Prof. L.M.V. Pitteway, Dr.J.P.A. Race (Brunel University, U.K.), Dr.J.E. Bresenham (IBM, USA) and Dr.R.A.Earnshaw (University of Leeds, U.K.) for their many suggestions, that enabled him to finish this project successfully, and to his students especially Mr.M.Knizek and Mr.J.Hruby, for the interest which stimulated this work.

6. Literature

[1] Bresenham J.E.: Algorithm for Computer Control of Digital Plotter, IBM Syst.J. 4(1),1965,pp.25-30.
[2] Cartledge C.J., Weeks G.H.: The Implementation of Fill Area for GKS, in [4], pp.161-185.
[3] Dunlavey M.R.: Efficient Polygon-Filling Algorithm for Raster Display, Trans. on Graphics, Vol.2.,1983, pp.264-273.
[4] Earnshaw R.A. (Ed.) : Fundamental Algorithms for Computer Graphics, NATO ASI, Series F, Vol.17, Springer Verlag 1985.
[5] Foley J.D., van Dam A.: Fundamentals of Interactive Computer Graphics, Addison Wesley Publ.Comp., 1982.
[6] Gay A.: Experience in Practical Implementation of Boundary-Defined Area Fill, in [4], pp.153-160.
[7] Knizek M.,Skala V.: Basic Algorithms for Raster Devices - Filling and Hatching (Implementation), TR 209-10-87, Institute of Technology, Plzen, 1986 (in Czech).
[8] Newman W.M., Sproull R.F.: Principles of Interactive Computer Graphics, McGraw Hill Comp., 1981.
[9] Pavlidis T.: Algorithms for Graphics and Image Processing, Springer Verlag, 1982.
[10] Pitteway L.M.V.: Algorithms of Conic Generation, in [4] pp.219-238.
[11] Rogers D.F.: Procedural Elements for Computer Graphics, McGraw Hill Comp., 1985.
[12] Skala V.: The Bresenham's Algorithm as a Tool for the Hidden-Line, Hidden-Surface and Hidden-Contouring Problem Solution, TR 209-7-87, Institute of Technology, Plzen, 1984.
[13] Skala V.: An Alternative Polygon Representation in the Raster Environment, TR 209-8-87, Institute of Technology, Plzen, 1987.
[14] Skala V.: The Filling and Hatching Operations for the Raster Environment, TR 209-9-87, Institute of Technology, Plzen, 1987.
[15] Smith A.R.:Tint Fill, Computer Graphics, Vol. 13, 1979, pp.276-283,(Proc. SIGGRAPH'79).

Algorithms for 2D Line Clipping

Abstract

New algorithms for 2D line clipping against convex, non-convex windows and windows that consist of linear edges and arcs are being presented. Algorithms were derived from the Cohen-Sutherland's and Liang-Barsky's algorithms. The general algorithm with linear edges and arcs can be used especially for engineering drafting systems. Algorithms are easy to modify in order to deal with holes too. The presented algorithms have been verified in TURBO-PASCAL. Because of unifying approach to the clipping problem solution all algorithms are simple, easy to understand and implement.

Keywords: clipping, line clipping, convex polygon, non-convex polygon, areas, algorithms

Dr. Vaclav Skala
c/o Department of Informatics and Computer Science
Institute of Technology
Nejedleho sady 14
306 14 Plzen
CZECHOSLOVAKIA

tel. (019) 33650, 37461-5, 36881-5 ext 318
telex 154292 tech c

1. Introduction

Clipping is a very important part of all graphics packages. Generally it is the evaluation of a line intersection against a window boundary. Many efficient algorithms are known, asthe Cohen-Sutherland's [5], Liang-Barsky's [4], Cyrus-Beck's [1] ones. All these algorithms have some presumptions, e.g. the windows must be orthogonal or convex with oriented edges, etc.

In the following new algorithms will be described for convex-polygon and non-convex polygon clipping without need to orient edges in any order. Algorithm for non-convex area clipping, where boundaries are formed by line segments or arcs, is described, too. Particular care was devoted to handle all special situations properly. All algorithms are based on the only basic idea that is gradually widened for more general cases.

As far as the author is concerned none of these algorithms have been published in the accessible literature.

2. Convex polygon clipping

The bellow given convex polygon clipping algorithm is based on the principle of Liang-Barsky's algorithm and is simpler than the Cyrus-Beck algorithm and does not need an anticlockwise orientation of the polygon edges as Liang-Barsky's algorithm does.

Provided a convex polygon is given by its vertices in the clockwise or in the anticlockwise order arbitralily and none pair of edges lies on the same line (it is not a principle restriction). Let us consider some situations that might occur if a line segment with end points P_F and P_S ought to be clipped, see fig.2.1.

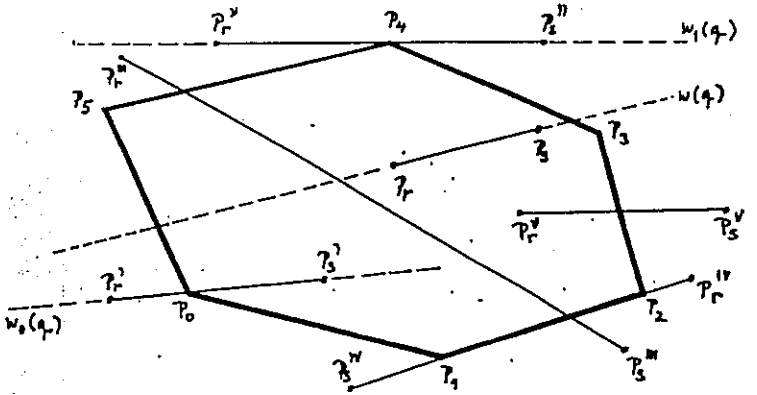


Fig.2.1.

All intersections of the line w with edges of the convex polygon are obtained by solving the following linear equations:

$x(q) = x_r + (x_s - x_r) \cdot q \qquad q \in \langle 0 , 1 \rangle$

$x(p) = x_i + (x_{i\oplus1} - x_i) \cdot p \qquad p \in \langle 0 , 1) \qquad i=0,1,\ldots,n-1$

where $\oplus$ means addition modulo n and point P_k has coordinates x_k.

The interval for parameter p is not closed in order to get rid of all ambiguities in case that the line segment is "passing" (line $w_0(q)$) or "touching" (line $w_1(q)$) a polygon vertex. The bellow given algorithm is based on the fact that a line segment can intersect a convex polygon only in two points. The algorithm finds the q values for all intersection points of a line on which the given line segment lies with the edges of the convex polygon and then the proper part of the line segment that is inside the convex polygon can be found. The algorithm is shown in fig.2.2. It is necessary, of course, to solve special cases when a line segment touches or passes a vertex or when it lies on a polygon edge.

The algorithm given in fig.2.2. is faster than the Cyrus-Beck's algorithm (it doesn't need an inner normal computation) and for a rectangle polygon is equivalent to the

Liang-Barsky algorithm in case that computation of all intersections is simplified for polygon edges parallel to the axes. The algorithm can be easily generalized or modified for a case when two edges of the given polygon lie on the same line, see [7].

```
VAR i,k: INTEGER; ( end points of the polygon edge )
    j: INTEGER;   ( counter )
    t: ARRAY [1..2] OF REAL;
BEGIN
    j:=0;i:=0;k:=n-1; ( set end points for the first edge )
    REPEAT
        IF an intesection point exists for the edge x_K x_i
        AND the line w(q) so that p∈<0,1) THEN
        BEGIN j:=j+1; t[j]:=q ( save the q value ) END
        ELSE
        IF the edge x_K x_i lies on the line w(q) THEN
        BEGIN t[1]:= a value q that corresponds to the
                     vertex x_K;
              t[2]:= a value q that corresponds to the
                     vertex x_i;
              j:=2
        END;
        k:=i; i:=i+1; ( take the next polygon edge )
    UNTIL ( j ≠ 2 ) OR ( i > n );
    IF j <> 0 THEN
    BEGIN
       IF j = 1 THEN t[2]:=t[1] ( the line w(q) "touches"
                                  a vertex )
                ELSE IF t[1] > t[2] THEN t[1] SWAP t[2];
       t[1]:= max ( 0.0 , t[1] );
       t[2]:= min ( 1.0 , t[2] );
       LINE ( x (t[1]) , x (t[2]) )
    END
END;
```

Fig.2.2.

3. Non-convex polygon clipping

An algorithm for non-convex polygon clipping is based on the parametric equations that express linear segments. In this case the algorithm must be more complex, because the line w(q) can intersects the polygon in many points. Assume that a non-convex polygon is given by its vertices in clockwise or anticlockwise order. Further that two successive edges do not lie on the same line, that all vertices have different coordinates, that not a single vertex lies on any edge of the given polygon and that two edges might have only a vertex as a common point. Let us consider again some situations that might occur if a line segment with end points P_r and P_s ought to be clipped, see fig.3.1.

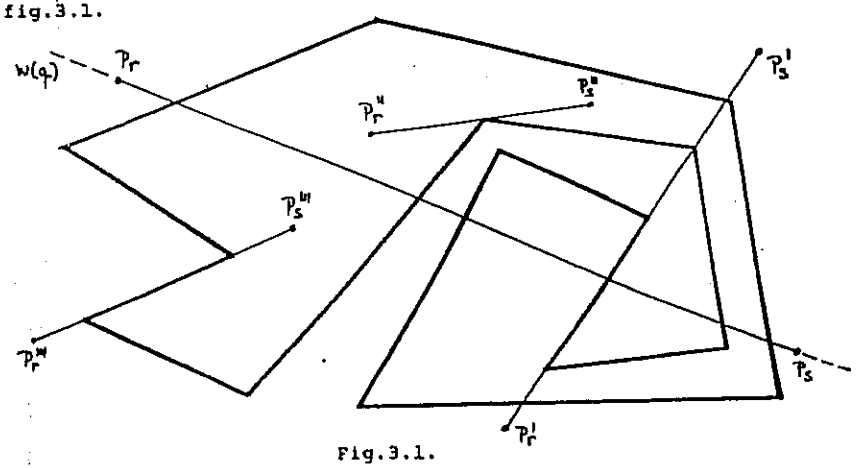


Fig.3.1.

The given line segment that ought to be clipped can be expressed by:

$$x(q) = x_r + (x_s - x_r) \cdot q \qquad q \in \langle 0 , 1 \rangle$$

and the edges of the non-convex polygon can be expressed by:

$$x(p) = x_i + (x_{i\oplus 1} - x_i) \cdot p \qquad p \in \langle 0 , 1) \quad i=0,1,\ldots,n-1$$

Then we will look for all intersection points of the line w(q) and non-convex polygon. The parametric equation:

$$x(q) = x_r + (x_s - x_r) \cdot q \qquad q \in (-\infty , +\infty)$$

expresses the line w(q). Coordinates of all intersection points will be determined by the value of the parameter q. But it is necessary to take into consideration the following special cases when the line w(q) passes or touches a vertex of the polygon. There are two possibilities, see fig.3.2.:

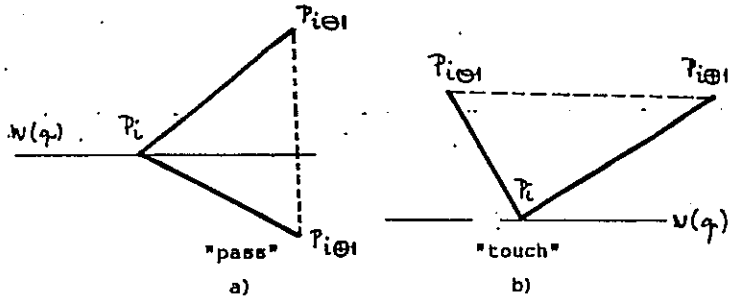


Fig.3.2.

where $\oplus$ means addition modulo n, $\ominus$ means subtraction modulo n

In fig.3.2.a. is generated only one intersection point, while in fig.3.2.b. double intersection point is generated. In both cases these points are processed as an ordinary intersection point.

A quite different situation is when the line w(q) lies on an edge of the polygon. There are two possible situations, see fig.3.3.:

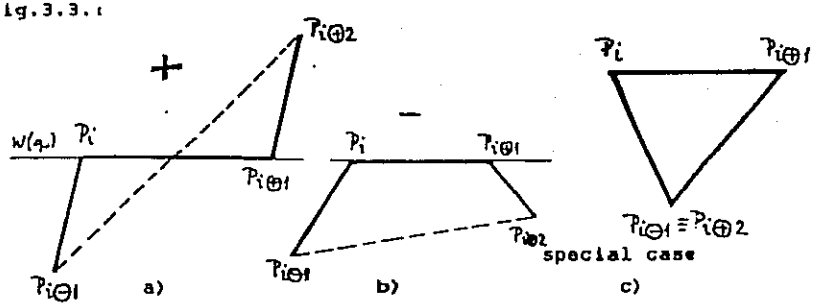


Fig.3.3.

The values of q parameters that correspond to the points P_i and $P_{i\oplus 1}$ in these cases must be generated. But it is necessary to distinguish between the type by attributes ++ or -- for these q values. If points $P_{i\oplus 2}$, $P_{i\oplus 1}$ are the same, then it is necessary to generate only q value which corresponds to the point P_i, see fig.3.3.c. (a special case when the given polygon is a triangle). Therefore the intersection point will be determined not only by value q but also by a type of the intersection as follows:

- ␣ intersection with edge, intersection of the type pass or touch
- \+ line w(q) lies on edge - case ad a)
- \- line w(q) lies on edge - case ad b)

When all intersection points are found together with their types, the given set of q values is sorted together with their attributes so that couples with attributes ++ or -- should not be split, e.g. sequencies:

$q_1\ q_1\ q_2$ or $q_1\ q_2\ q_2$

\+ ␣ + + ␣ +

must be transferred to sequencies:

$q_1\ q_1\ q_2$ or $q_1\ q_2\ q_2$

␣ + + + + ␣

Similarly for attributes --.

The set of q values will be processed according to table 3.1. Results that determine these parts of the line w(q) which are inside the given polygon are couples of q values.

Now it is necessary to determine which parts of the line segment P_FP_S are inside the given polygon, e.g. to determine those parts of the line w(q) that are inside and that are part of the line segment P_FP_S. According to the process of getting parts of the line w(q) it is necessary to make intersection of all couples of the q values with the interval < 0 , 1 >, see fig.3.4.

```
i:=1;
WHILE i <= No of intersection -1 DO
BEGIN IF max ( 0 ; qi ) <= min ( 1 , qi+1)
      THEN save ( max ( 0 , qi ) , min ( 1 , qi+1) );
      i:=i+2
END;
```

Fig.3.4.

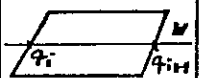
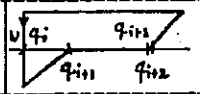
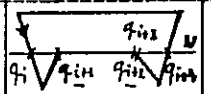
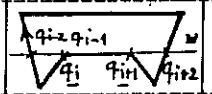
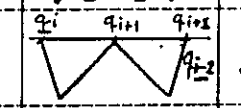
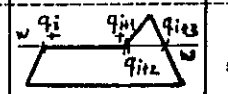
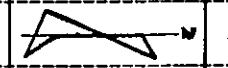
attributes		situation	action
q_i	q_{i+1}		
␣	␣		save (q_i , q_{i+1}) i:=i+2
␣	+		save (q_i , q_{i+3}) i:=i+4
␣	-		save (q_i , q_{i+2}) i:=i+3
-	-		save (q_i , q_{i+2}) i:=i+3
-	␣		vertex touches an edge
+	+		save (q_i , q_{i+1}) i:=i+2
+	␣		improper polygon
-	+		impossible situation
+	-		impossible situation

Table 3.1.

The coordinates of the resulted points determined by their q values can be obtained from the equation for the line w(q)

$$x(q) = x_r + (x_s - x_r) \cdot q$$

The whole algorithm can be described as follows in fig.3.5.

```
j:=0; ( counter of q values )
k:=n-1;
FOR i:=0 TO n-1 DO
BEGIN IF the edge x_k x_i lies on the line w(q) THEN
      BEGIN determine the type;
            IF x_K <> x_KB2 ( not a triangle )
            THEN IF type = '+' THEN generate ( q_1+, q_2+)
                               ELSE generate ( q_1-, q_2-)
            ELSE generate ( q_j )
      END
      ELSE IF the line w(q) passes or touches the vertex x_K
           THEN IF type is touch THEN generate ( q_1, q_2)
                                 ELSE generate ( q_1 )
           ELSE IF the line w(q) intersects edge x_k x_i
                THEN generate ( q_j );
      k:=i
END;
SORT ( values q );
REDUCE ( set of q values );
SELECT ( subintervals as < q_K , q_K+1 > ∩ < 0 , 1 > );
COMPUTE ( the end points );
```

Fig.3.5.

The presented algorithm in fig.3.5. enables to clip a given line segment agains non-convex polygon. The assumptions stated above are not substantial with the only exception that edges may not intersect one another and algorithm can be easily modified for these assumptions [7]. The algorithm is based on the similar idea as the previous one and because the polygon in non-convex some additional operations must be employed.

4. Non-convex area clipping

So far presented algorithms have solved the line clipping by convex or non-convex polygon, e.g. by areas that consist of linear edges. But plenty of applications require clipping over areas that are formed by linear edges and arcs, see fig.4.1.

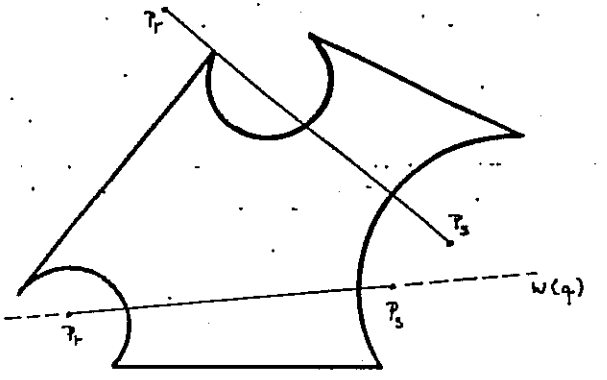


Fig.4.1.

Provided a non-convex area is given by its vertices in clockwise or in anticlockwise order and if the edge is not linear then the centre of arc is given together with information whether the right or left part of the circle is taken from the actual vertex, see fig.4.2. It is also assumed that no three successive vertices lie on a common line, that all vertices have different coordinates, that none vertex lies on an edge or arc and that two edges or arcs might have only a vertex as a common point.

In difference from the previous problem the line w(q) can intersect the arc edge in two points, that increases the complexity of the given problem substantially.

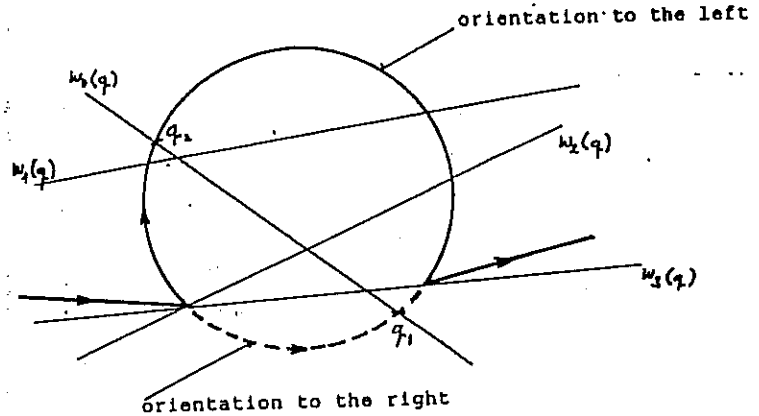


Fig.4.2.

The line w(q) is described by the parametric equation:

$$x(q) = x_r + (x_s - x_r) \cdot q \qquad q \in (-\infty, +\infty)$$

The procedure for finding all intersection points is similar to the previously stated algorithm, but now in the case of arc edge it is necessary to solve equations:

$$x(q) = x_r + (x_s - x_r) \cdot q \qquad q \in (-\infty, +\infty)$$

$$(x - x_u)^2 + (y - y_u)^2 - r^2 = 0$$

where (x_u, y_u) is a centre of the given arc

r is a diameter of this arc.

Solving these equations with regard to variable q a quadratic equation

$$a q^2 + b q + c = 0$$

will be obtained, where:

$$a = (x_s - x_r)^2 + (y_s - y_r)^2$$

$$b = 2 [(x_r - x_u) \cdot (x_s - x_r) + (y_r - y_u) \cdot (y_s - y_r)]$$

$$c = (x_r - x_u)^2 + (y_r - y_u)^2 - r^2$$

In the case that the line w(q) intersects or touches the given circle two solutions are obtained, not necessarily different, and:

$$q_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Of course some special situations must be solved again, e.g. when the line passes or touches a vertex or when the line is passing the start and the end points of the arc.

Now it is necessary to determine which part of the circle forms the boundry of the given area. Because the border is oriented it can be distinquish whether the arc is on the right or on the left from the connection of $x_K x_{K\oplus 1}$ points. If the line $w_o(q)$ is considered then it must be decided which intersection point ought to be taken. It is obvious that the only point which lies on the proper arc will be considered. It means that:

- if the left arc is considered then the point $x(q_i)$ will be taken if and only if

$$\det \begin{pmatrix} x_{K\oplus 1} - x_K \\ x(q_i) - x_K \end{pmatrix} > 0 \qquad i=1,2$$

- if the right arc is considered then the point $x(q_i)$ will be taken if and only if

$$\det \begin{pmatrix} x_{K\oplus 1} - x_K \\ x(q_i) - x_K \end{pmatrix} < 0 \qquad i=1,2$$

Now the situations when the line w(q) passes the vertices x_K and $x_{K\oplus 1}$ must be solved, see fig.4.3.

Provided

$$x(q_1) = x_K \qquad x(q_2) = x_{K\oplus 1}$$

then the intersection points muast be generated according to the type of intersection in the points x_K and $x_{K\oplus 1}$, see table 4.1.

It is necessary to express explicitely that the second value q_2 will be generated in the next step, when the intersection point will be computed for the $x_{K\oplus 1} x_{K\oplus 2}$ border, e.g. for the line segment or arc.

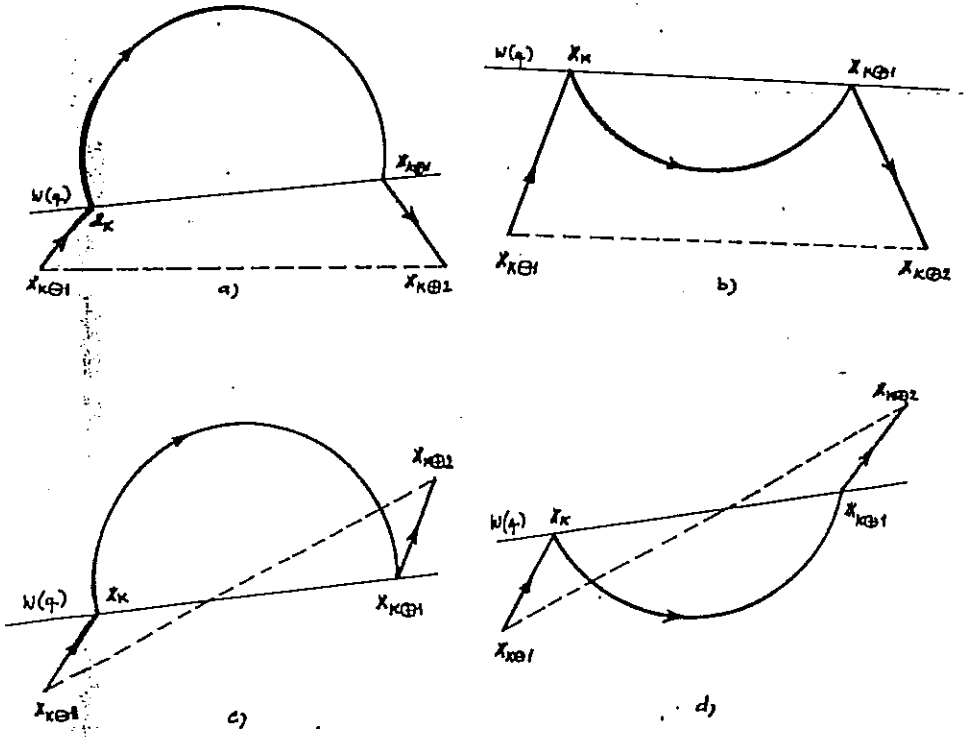


Fig.4.3.

generated values of q		type of $x_{K\oplus 1}$	
		"pass"	"touch"
type of x_K	pass	q_1 fig.4.3.a.	q_1 q_2 fig.4.3.c.
	touch	q_1 q_1 fig.4.3.d.	q_1 q_1 q_2 fig.4.3.b.

Table 4.1.

The last special case is when the line w(q) passes the x_K vertex, e.g.

$$x(q_1) = x_K$$

Then the second intersection point $x(q_2)$ with the arc must be generated again according to the type of intersection point, see fig.4.4.

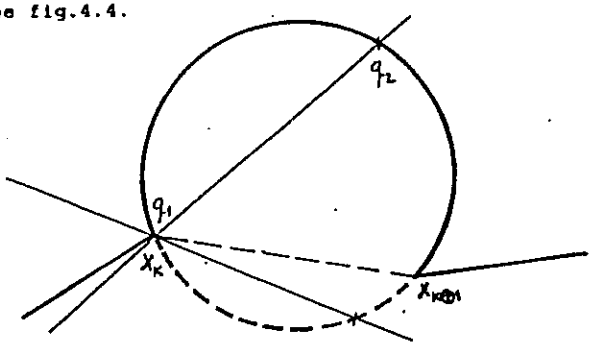


Fig.4.4.

The rules for the generation are given in table 4.2.

x_K	generated values of q
pass	q_1 q_2^*
touch	q_1 q_1 q_2^*

Table 4.2.

*The values q_2 are generated if and only if the point $x(q_1)$ lies on the required arc, e.g. when

$$\det\begin{pmatrix} x_{K\oplus 1} - x_K \\ x(q_2) - x_K \end{pmatrix} > 0$$

and the arc is to the left from the line segment $x_K x_{K\oplus 1}$

or

$$\det\begin{pmatrix} x_{K\oplus 1} - x_K \\ x(q_2) - x_K \end{pmatrix} < 0$$

and the arc is to the right from the line segment $x_K x_{K\oplus 1}$

The conditions that determine the type of the intersection point must be specified.

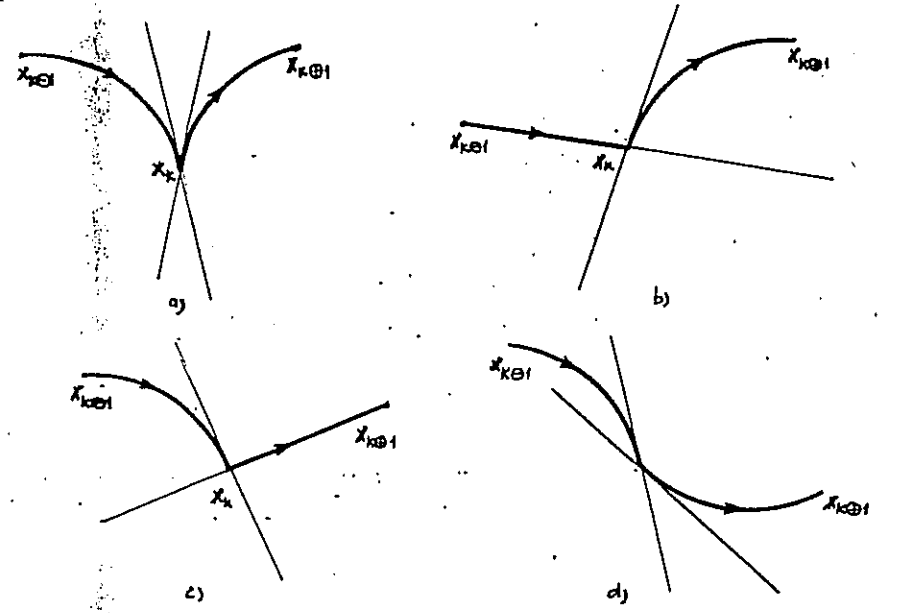


Fig.4.5.

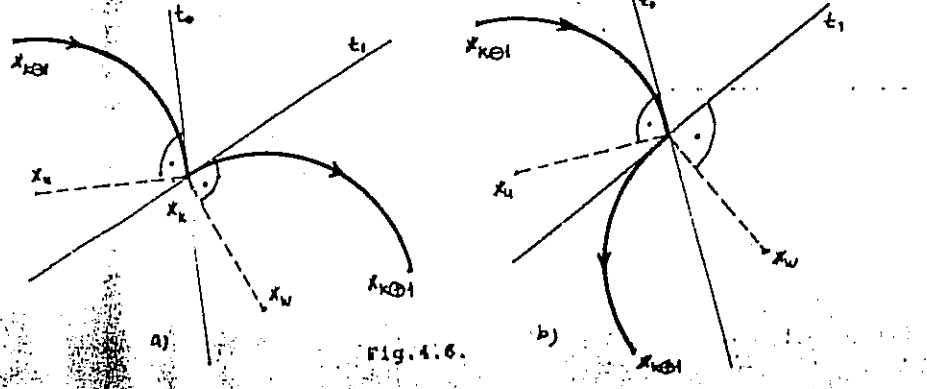


Fig.4.6.

It is obvious that if the line w(q) lies in the filled part, see fig.4.5., then the line w(q) "passes" the vertex x_k otherwise the line w(q) "touches" the vertex x_k. If the boundry is formed by an arc its is necesary to take tanget to this arc in the point x. otherwise the line segment is taken itself. Of course, the arc orientation must be taken into account, see fig.4.6.

If the border between $x_{k\oplus1}$ and x_k is formed by an arc with the centre at x_u then the equation for its tangent t_o in the point x_k has the form:

$$(x - x_u) . (x_k - x_u) + (y - y_u) . (y_k - y_u) - r_o^2 = 0$$

where r_o is the radius of the given arc and

$$r_o^2 = (x_u - x_k)^2 + (y_u - y_k)$$

If the equation for the tangent t_o is expressed in the form

$$f_o (x) = 0$$

then the inequality

$$f_o (x) \leqslant 0 \quad \text{and} \quad f_o (x_u) < 0$$

expresses a half-plane whose border is made by the tangent t_o and which cover the centre of the given arc. Similarly for the arc between x_k and $x_{k\oplus1}$ points with the center at x_w. The equation for the tangent t_1 has the form:

$$(x - x_w) . (x_k - x_w) + (y - y_w) . (y_k - y_w) - r_1^2 = 0$$

where r. is an radius of the given arc and

$$r_1^2 = (x_w - x_k)^2 + (y_w - y_k)^2$$

The equation tangent t_1 can be expressed in the form

$$f_1 (x) = 0$$

and the inequality

$$f_1 (x) \leq 0 \quad \text{and} \quad f_1 (x_w) < 0$$

expresses a half plane whose border is made by the tangent t_1 which cover the centre of the given arc.

Now it is possible to define a condition for the touch of the line w(q) in the vertex x_k, where the line w(q) is defined by equation:

$$x (q) = x_r + (x_s - x_r) . q \qquad q \in (-\infty, +\infty)$$

The line w(q) touches the vertex at the x_k point if and only if

the appropriate expession from table 4.3. has value true.

arc orientation		$\overset{\frown}{x_K x_{K\oplus 1}}$	
		to the left	to the right
$\overset{\frown}{x_{K\ominus 1} x_K}$	to the left	p_0 xor p_1	p_0 xor not p_1
	to the right	not p_0 xor p_1	not p_0 xor not p_1

Table 4.3.

where: $p_0 = f_0 (x (q)) \leq 0$ for such a value q that $x(q) \neq x_K$, e.g. q=0 or q=1

and $p_1 = f_1 (x (q)) \leq 0$ for such a value q that $x(q) \neq x_K$, e.g. q=0 or q=1

It is possible to show that the condition

not p_0 xor not p_1 is equal to p_0 xor p_1

In case that the $x_{K\ominus 1}x_K$ or $x_K x_{K\oplus 1}$ borders are edges formed by line segments then the equation for the line segment must be taken, e.g.

$$(y_{K\ominus 1} - y_K) . x + (x_K - x_{K\ominus 1}) y + x_{K\ominus 1} . (y_K - y_{K\ominus 1}) + y_{K\ominus 1} . (x_{K\ominus 1} - x_K) = 0$$

instead of the equation for the tangent t_0 . Similarly for $x_K x_{K\oplus 1}$ border and the tangent t_1.

The whole algorithm can be described as follows in fig.4.7.

```
( Main body )
BEGIN k:=n-1; i:=0; ( the index k+1 is now indexed by i )
      WHILE i<>n DO
      BEGIN IF the border between points x_k x_i is linear
            THEN find intersections of the line w(q) with the
                 line segment x_k x_i ( see fig.4.8. )
            ELSE find intersections of the line w(q) with the
                 arc x_k x_i ( see fig.4.9. );
```



```
            k:=i;
            i:=i+1
      END;
      SORT ( values q );
      REDUCE ( set of q values );
      SELECT ( subintervals as < q_k ; q_{k+1} >/\< 0 , 1 > );
      COMPUTE ( the end points )
END;
```

Fig.4.7.

```
( Find intersections of the line w(q) with the line segment )
BEGIN IF the line segment x_k x_i lies on the line w(q)
      THEN
      BEGIN determine the type;
            IF x_K <> x_{K⊕2} ( special case - triangle )
            THEN IF type='+' THEN generate ( q_+1 , q_+2 )
                             ELSE generate ( q_-1 , q_-2 )
            ELSE generate ( q_u1 )
      END
      ELSE IF the line w(q) passes or touches the vertex x_K
           THEN IF type is touch THEN generate ( q_u1 , q_u2 )
                                 ELSE generate ( q_u1 )
           ELSE IF the line w(q) intesects the edge x_K x_i
                THEN generate ( q_u1 )
END;
```

Fig.4.8.

```
( Find intersections of the line w(q) with the arc x_K x_i )
BEGIN IF the line w(q) passes or touches the vertex x_K
      THEN generate ( q_1 , q_2 ) if the are on the right side
      ELSE IF the line w(q) passes or touches vertex x_i
           THEN generate ( q_1 , q_2 ) according to the type of
                vertex x_K ( see tab.4.2.)
           ELSE generate ( q_1 , q_2 ) according to the type of
                vertices x_K x_i ( see tab. 4.1. )
END;
```

Table 4.9.

5. Conclusion

The presented algorithms are based on the principle of the Liang-Barsky's algorithm. It is shown how the algorithms become more complicated if the requirements are more general. The presented algorithms were verified in TURBO-PASCAL on IBM-PC. The algorithm for the clipping line against a convex polygon is simpler than the Liang-Barsky's, if it is simplified for a rectangle clipping window, and in general it doesn't need oriented half-planes of the clipping window. The second algorithm solves the situation when the clipping polygon is non-convex. The increase of complexity is expressed in the need to distinquish between different cases and to sort the final set of intersection points. The last presented algorithm solves the problem when the clipping area is formed by line segments and arcs. This problem has not been solved in the accessible literature as far as to the author knowledge is concerned. The algorithms are fast and all special cases are handled properly. All algorithms might be used for hatching, too. of course, in that case it is convenient to rotate the clipping area so that the hatching lines are horizontal, find intersection points and then rotate their coordinates back. In this case all algorithms can be significantely simplified. In general these algorithms can be easily modified for case that the area contains holes or for case of the set operations with areas [10].

6. Acknowledgment

The author would like to express his thanks to Prof.L.M.V. Pitteway, Dr. J.P.A. Race (Brunel Univ., U.K.), Dr. J.E. Bresenham (Winthrop College, U.S.A.) and Dr.R.A. Earnshaw (Univ. of Leeds, U.K.) for their helpfull discussions during his stay in U.K., to Miss I. Kolingerova (Inst. of Technology, Plzen) her interest, comments and the final implementation on IBM-PC, who enabled to find unspecified special cases and errors, and to students of Computer Graphics course that stimulated this work.

7. Literature

[1] Cyrus,M., Beck,J.: Generalized Two- and Three Dimensional Clipping, Computers and Graphics, Vol.3, No.1., 1978, pp.23-28.

[2] Foley,J.D., van Dam,A.: Fundamentals of Interactive Computer Graphics, Addison-Wesley, Reading, Mass.,1982

[3] Liang,Y.D., Barsky,B.A.: An Analysis and Algorithms for Polygon Clipping, CACM 26, No.11 (Nov. 1983), pp.868-876.

[4] Liang,Y.D., Barsky,B.A.: A New Concept and Method for Line Clipping, ACM Transaction on Graphics, Vol.3., No.1., 1984, pp.1-22.

[5] Newman,W.M., Sproull,R.F.: Principles of Interactive Computer Graphics, 2nd ed., McGraw Hill, New York, 1979.

[6] Nicholl,T.M., Lee,D.T., Nicholl,R.A.: An Efficient New Algorithm for 2D Line Clipping: Its Development and Analysis, ACM Computer Graphics, Vol.21, No.4., July 1987, pp.253-262.

[7] Skala,V.: Convex Polygon Clipping Algorithm, TR-209-10-88, Computer Science Dept., Inst. of Technology Plzen, 1988.

[8] Skala,V.: Non-convex Line Clipping Algorithm, TR-209-11-88 Computer Science Dept., Inst.of Thechnology, Plzen,1988.

[9] Skala,V.: Clipping Lines by Non-convex areas, TR-209-12-88 Computer Science Dept., Inst. of Thechnology, Plzen, 1988.

[10] Skala,V.: The Set Operations with Polygons and Areas in the Raster Environment, TR 209-11-87, Computer Science Dept., Plzen, 1987.