Algoritmy počítačové grafiky II

(Algorithms for Computer Graphics II)

Prof.Ing.Václav Skala, CSc.

http://www.VaclavSkala.eu

Abstrakt

Algoritmy počítačové grafiky a jejich implementace je nedílnou součástí jak grafických systémů, tak i CAD/CAM systémů. V této práci jsou uvedeny základní algoritmy, metody a postupy, které se používají v různých modifikacích i v dnešních systémech

Abstract

Algorithms for computer graphics and their implementation is a part of graphical systems and CAD/CAM systems. Fundamental algorithms, methods and approaches used in today's systems are explained.

Algoritmy počítačové grafiky II
Algorithms for Computer Graphics II
Published & printed by: Vaclav Skala – UNION Agency
Na Mazinach 9
CZ 322 00 Plzen
Czech Republic
Year 2011
ISBN 978-80-86943-20-6

Electronic version http://www.VaclavSkala.eu http://Graphics.zcu.cz

Profil autora

Prof. Ing. Václav Skala, CSc. se zabývá počítačovou grafikou od r.1975, kdy na Vysoké škole strojní a elektrotechnické v Plzni zavedl a následně vyučoval předmět Počítačová grafika a umělá inteligence. V roce 1981 obhájil disertační práci na témate relačních databází se specializací na reprezentaci relací a optimalizaci dotazů uživatele. V současné době se odborně věnuje především algoritmům počítačové grafiky a vizualizaci dat a algoritmům včetně matematických aspektů. Je vedoucím Centra počítačové grafiky a vizualizací (http://Graphics.zcu.cz) při Katedře informatiky a výpočetní techniky na Fakultě aplikovaných věd Západočeské univerzity v Plzni.

V letech 1984-1989 působil na Brunel University v Londýně a později přednášel na NATO Advanced Study Institute v zahraničí. V roce 1996 se stal profesorem na Západočeské univerzitě, odborně působil na Bath University v U.K., Gavle University ve Švédsku a na dalších zahraničních odborných pracovištích.

Prof.Skala je řešitelem zahraničních i národních odborných výzkumných projektů, členem několika redakčních rad prestižních zahraničních odborných časopisů, členem programových výborů mezinárodních odborných konferencí. Od r.1992 je organizátorem mezinárodních odborných konferencí WSCG zaměřených na počítačovou grafiku, vizualizaci dat a počítačové vidění (http://www.WSCG.eu).

V současné době je prof. Skala profesorem na Západočeské univerzitě v Plzni a VŠB-Technické univerzitě v Ostravě. V roce2010 odborně působil též na Přírodovědné fakultě Ostravské univerzity v Ostravě (http://AlgVis.osu.cz).

V roce 2010 získal významné ocenění mezinárodní asociace pro počítačovou grafiku

„Fellow of the Eurographics Association“

za dlouhodobé odborné výsledky a organizační aktivity v oblasti počítačové grafiky.

Poznámky pro laskavého čtenáře

Toto je rekonstrukce učebního textu Algoritmy počítačové grafiky I – III, který vznikl pro studenty Západočeské univerzity v Plzni v roce 1991. Tato publikace vznikla rozšířením publikace Počítačová grafika I - II, která vznikala v roce 1989 pro potřeby studentů Vysoké školy strojní a elektrotechnické a která byla vydána v roce 1990.

Jde tedy o text zohledňující algoritmy, metody a stav technologie před více než před 20 lety. Je nutné si uvědomit, že obor počítačové grafiky je poměrně velmi mladý, bouřlivě se rozvíjející a poměrně hodně závislý na dostupných technologiích. Nicméně většina metod, postupů a algoritmů je používána dodnes.

Pochopitelně je oblast počítačové grafiky dnes podstatně širší a zahrnující mnohé jiné oblasti, které jsou odborně rozvíjeny kolegy v **Centru počítačové grafiky a vizualizací** při Katedře informatiky a výpočetní techniky Fakulty aplikovaných věd, viz http://Graphics.zcu.cz

Rád bych proto požádal čtenáře, aby uvedený text a kvalitu tisku posoudil v kontextu té doby, kdy:

- nebyl prakticky přístup k zahraniční literatuře a publikace se velmi obtížně získávaly přes mezinárodní výpůjčky, resp. přes osobní kontakty
- nebyl k dispozici Internet, e-mail, WEB a digitální knihovny
- byly k dispozici první 8mi bitové PC systémy s „fantastickou“ kapacitou paměti 512 KB
- byly k dispozici první jehličkové tiskárny (9 tiskových bodů na textovou řádku) a „unikátní“ textové procesory T602 a výborný graficko-textový procesor Chi-Writer (http://en.wikipedia.org/wiki/ChiWriter), ve kterém byl text napsán.

Je nutné si uvědomit, že před rokem 1989 bylo velmi obtížné získávat odborné publikace a publikovat odborné výsledky v mezinárodně uznávaných časopisech a na mezinárodních konferencích.

Již v roce 1975 na Vysoké škole strojní a elektrotechnické, Fakultě elektrotechnické vzniká předmět Počítačová grafika a umělá inteligence

Nové odborné výsledky, algoritmy a metody laskavý zájemce nalezne na URL:
http://www.VaclavSkala.eu.

Také bych rád zájemce o počítačovou grafiku odkázal na publikace z mezinárodních konferencí WSCG, na časopis Journal of WSCG a GraVisMa workshopy:

- **WSCG** – International Conferences on Computer Graphics, Visualization and Computer Vision
  http://www.WSCG.eu
- **GraVisMa –** Computer Graphics, Vision and Mathematics http://www.GraVisMa.eu

Tento text, který vznikl rekonstrukcí z archivu autora, byl doplněn o seznam publikací tak, aby tvořil ucelenou publikaci. Předpokládá se, že další části původní publikace budou rekonstruovány následně.

Je mi milou povinností poděkovat všem, kteří mi stimulovali moje odborné aktivity a které jsem měl tu čest nejen potkat na Brunel University a na NATO Advanced Study Institutes, ale i s mnohými odborně spolupracovat. Patří k nim zejména:

Prof. M.L.V. Pitteway

Brunel University,
U.K.

Prof. Jack E. Bresenham

IBM Corp., USA
a
Winthrop University,
USA

Prof. F.R.A.Hopgood

Rutherford Appleton Laboratory, U.K.
a
Oxford Brookes University, U.K.

Prof. David F.Rogers

U.S. Naval Academy,
USA

V Plzni dne: 2011-07-25 prof.Ing.Václav Skala, CSc.

Předmluva

V publikaci Algoritmy počítačové grafiky I - III jsem se pokusil zachytit současný stav v oblasti principů vybraných algoritmů počítačové grafiky a CAD systémů. Vzhledem k prudkému rozvoji použití technických a programových prostředků počítačové grafiky je tato publikace do určité míry pojata jako přehledová, přičemž podrobnosti lze nalézt v literatuře. Některé pasáže bylo nutno zredukovat na téměř informativní úroveň z důvodů přílišné teoretické složitosti, složitosti algoritmů, případně pro jejich nedostupnost, neboť firmy skutečně použité metody nepublikují.

Pro další studium v oblasti počítačové grafiky je nezbytné sledovat odbornou literaturu, na kterou bych rád upozornil, a to zejména na:

a) časopisy

Computer Graphics Forum, North Holland Comp.*
ACM Transaction on Graphics, ACM*
The Visual Computer, Springer Verlag+
Computer Aided Geometric Design, North Holland Comp.+
Computational Geometry, Theory and Applications, Elsevier Science Publ.+
Computer Aided Design, Butterworth & Co (Publishers) Ltd.+
Computers & Graphics, Pergamon Press

b) sborníky konferencí

EUROGRAPHICS* (EUROGRAPHICS Association),
SIGGRAPH+ (USA),
Computer Graphics International+

Publikace označené *, + je možné vypůjčit přes knihovnu Západočeské univerzity (publikace označené + netvoří celý komplet).

Za základní publikace v oblasti počítačové grafiky lze považovat [5], [11], [44], [55], [56], [66], [69], [72], [94], [97], [104], [108], [109], [112], [114], [123], přičemž ostatní, uvedené v seznamu literatury, lze označit za doplňkové, sloužící především k dalšímu studiu dané problematiky, případně partií souvisejících s počítačovou grafikou. Celkový seznam použité literatury je uveden v 3. díle této publikace.

Je mou milou povinností poděkovat všem, kteří přispěli ke vzniku této publikace, ať už vytvořením podmínek nebo stimulací k vlastní práci. Chtěl bych zejména poděkovat ing.I.Kolingerové za její konkrétní, podnětné a kritické připomínky, bývalým i současným studentům Západočeské univerzity v Plzni se zaměřením Počítačová grafika a CAD systémy, kteří byli neocenitelnými pomocníky při ověřování algoritmů a tvorbě demonstračních a výukových programů, které jsou případným zájemcům k dispozici. Skripta byla do formátu "camera ready" připravena pomocí textového procesoru ChiWriter, který byl laskavě zapůjčen firmou Horstmann Software (USA) a který byl neocenitelným pomocníkem. Většina obrázků byla realizována pomocí programu COREL DRAW, který se ukázal být velmi dobrým nástrojem pro daný účel.

Budu zavázán všem čtenářům za jakékoliv připomínky, návrhy či doplnění předkládaného textu. Uvítám rovněž případné informace o aplikaci předkládaných algoritmů, jejich modifikací či zcela nových algoritmech počítačové grafiky.

Obsah

4. Geometrické transformace

Každý soubor podprogramů pro počítačovou grafiku je vybaven podprogramy pro geometrické manipulace s geometrickými objekty. Mezi nejzákladnější operace patří posuv, rotace a změna měřítka (zvětšení, zmenšení). Při realizaci operací budeme požadovat možnost kombinace základních operací.

4.1 Transformace v rovině

Operace posunutí

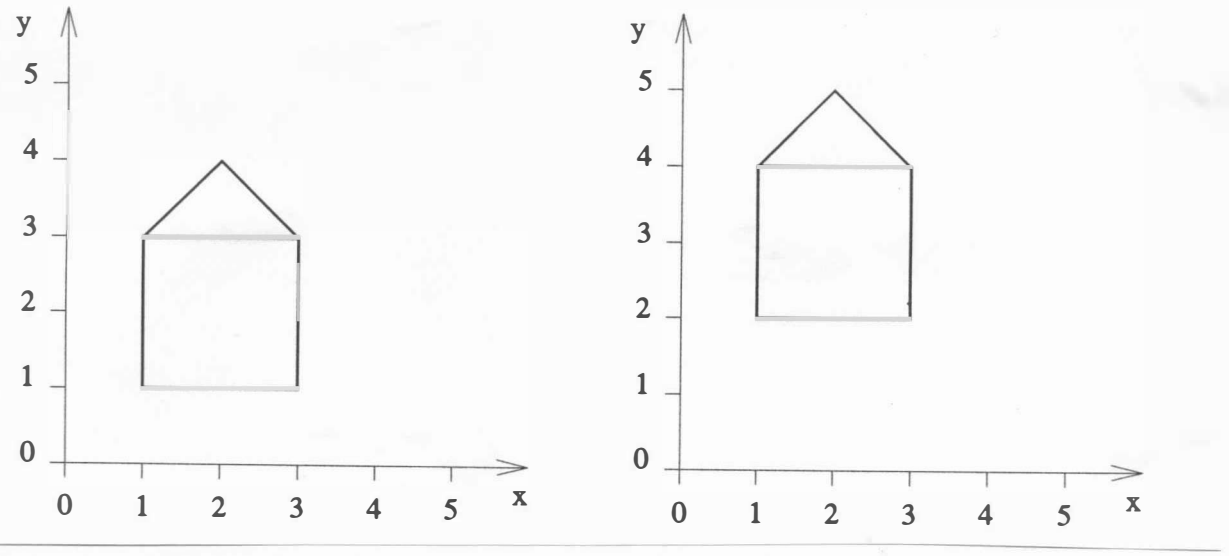
Má-li být bod (x,y) posunut o vektor (a,b) a označíme-li novou pozici bodu (x',y'), pak platí:

$$x' = x + a$$
$$y' = y + b$$

V maticové formě pak dostáváme:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} a \\ b \end{bmatrix}$$

Je zřejmé, že chceme-li posunout nějaký objekt, pak tuto transformaci musíme aplikovat na všechny body daného objektu. Předpokládejme, že máme jednoduchý geometrický objekt ve tvaru domečku, který chceme posunout o vektor (1,1). Stav před a po posunutí je na obr.4.1.1.



před posunutím po posunutí

Obr.4.1.1

Rotace

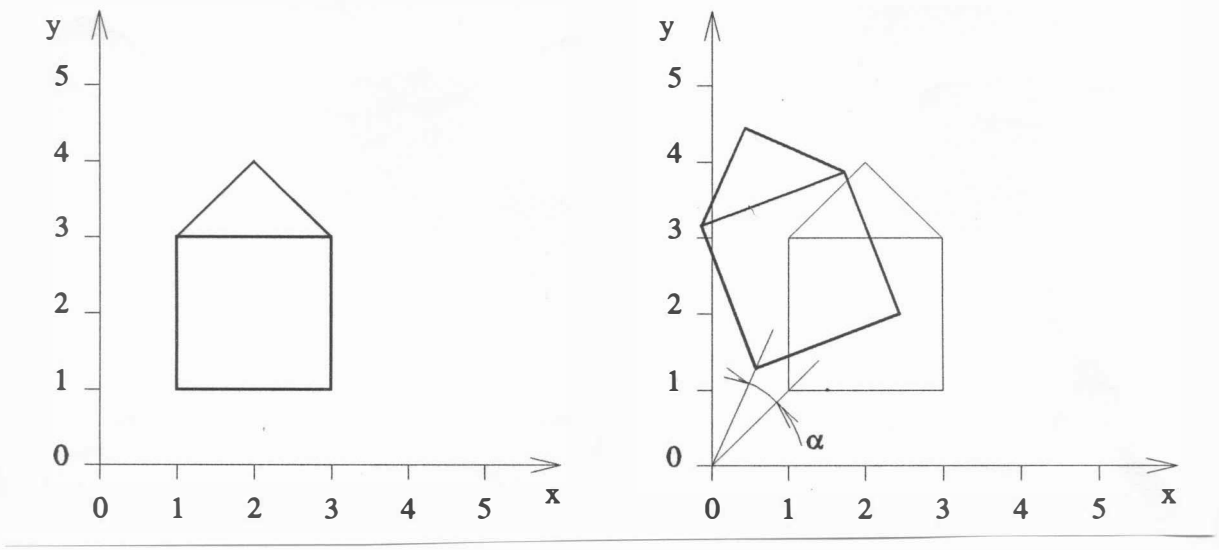
Další velmi často používanou operací je operace rotace. Pro operaci rotace okolo počátku souřadného systému platí:

$$x' = x \cdot \cos\alpha - y \cdot \sin\alpha$$
$$y' = x \cdot \sin\alpha + y \cdot \cos\alpha$$

V maticovém tvaru:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

kde (x,y) jsou souřadnice bodu v původním souřadném systému, (x',y') jsou souřadnice bodu po provedení rotace okolo počátku souřadného systému o úhel α.



před rotací po rotaci

Obr.4.1.2

Změna měřítka

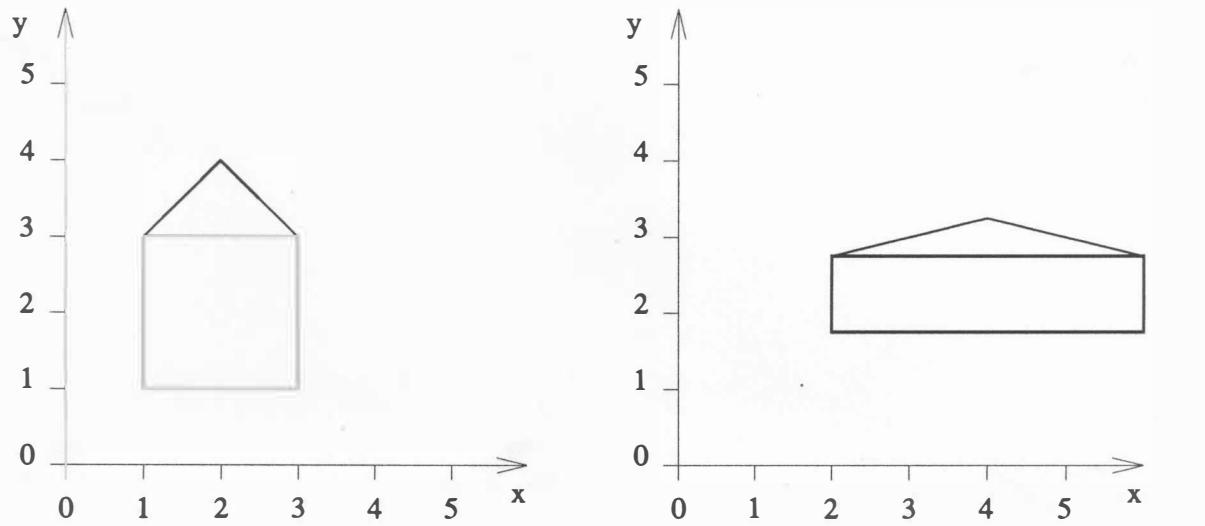
Změna měřítka je velmi potřebnou operací. Mnohé systémy nedovolují změnit měřítko na ose x rozdílně od měřítka na ose y. Chceme-li měřítko na ose x změnit Sx krát a na ose y změnit měřítko Sy krát, pak platí:

$$x' = Sx \cdot x$$
$$y' = Sy \cdot y$$

V maticovém tvaru:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} Sx & 0 \\ 0 & Sy \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

Tuto operaci lze opět demonstrovat pro Sx=2 (zvětšení) a Sy=0.5 (zmenšení) pomocí obr.4.1.3.



před změnou měřítek po změně měřítek

Obr.4.1.3

Pro reprezentaci všech operací by bylo vhodné mít jeden typ operací. Toho je možné docílit zavedením homogenních souřadnic. Bod (x,y) je reprezentován v homogenních souřadnicích (wx,wy,w), kde w<>0. Pak bod s homogenními souřadnicemi (X,Y,W) má kartézské souřadnice x=X/W a y=Y/W. Jednotlivé operace je možné nyní přepsat do maticového tvaru:

- posunutí o vzdálenost (a,b)

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & a \\ 0 & 1 & b \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad \mathbf{x'} = \mathbf{T}(a,b) \,.\, \mathbf{x}$$

- rotace o úhel α okolo počátku

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad \mathbf{x'} = \mathbf{R}(\alpha) \,.\, \mathbf{x}$$

- změna měřítka

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad \mathbf{x} = \mathbf{S}(Sx, Sy) \,.\, \mathbf{x}$$

Velkou výhodou homogenních souřadnic je, že umožňuje realizovat jednotlivé geometrické operace pomocí jediné maticové operace a pro některé aplikace reprezentovat bod v nekonečnu, což je limitní případ pro w=0.

Z dosud uvedeného vyplývá, že:

$$\mathbf{T}^{-1}(a, b) = \mathbf{T}(-a, -b)$$

$$\mathbf{R}^{-1}(\alpha) = \mathbf{R}(-\alpha)$$

$$\mathbf{S}^{-1}(Sx, Sy) = \mathbf{S}(1/Sx, 1/Sy)$$

Vzhledem k tomu, že matice **R** je ortogonální, platí, že:

$$\mathbf{R}^{-1}(\alpha) = \mathbf{R}^{T}(\alpha)$$

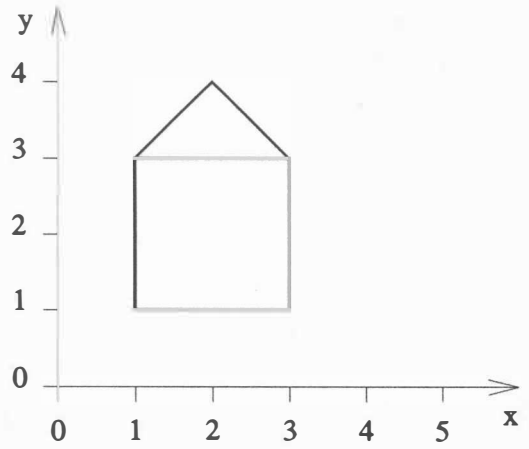
Transformace je možné řetězit. Vzhledem k tomu, že maticové operace nejsou obecně komutativní, nelze očekávat, že výsledek po rotaci a posunutí bude stejný jako po posunutí a rotaci. Uvažme několik jednoduchých příkladů pro demonstraci jednotlivých operací a jejich složení.

4.2 Řetězení dvourozměrných transformací

Otočení objektu okolo bodu

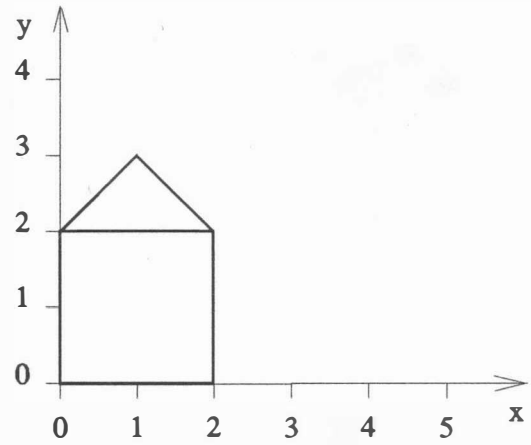
Mějme geometrický objekt, který chceme otočit o úhel α okolo daného bodu A, který není počátkem souřadného systému. Požadovanou operaci je nutné rozdělit na:

- posun všech bodů objektu tak, aby bod A byl v počátku souřadného systému
- pootočení všech bodů daného objektu o úhel α
- posunutí všech bodů objektu tak, aby bod A byl ve své původní pozici



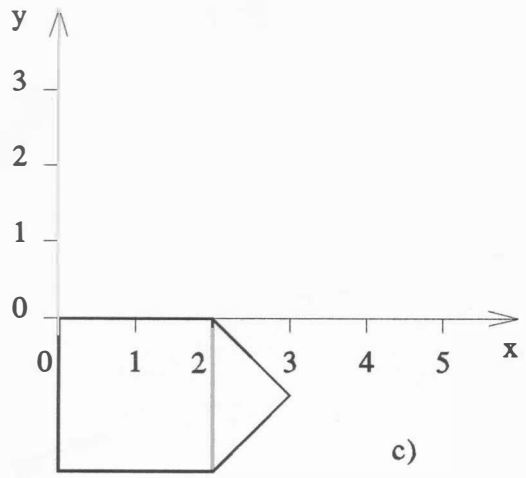
a)

původní stav



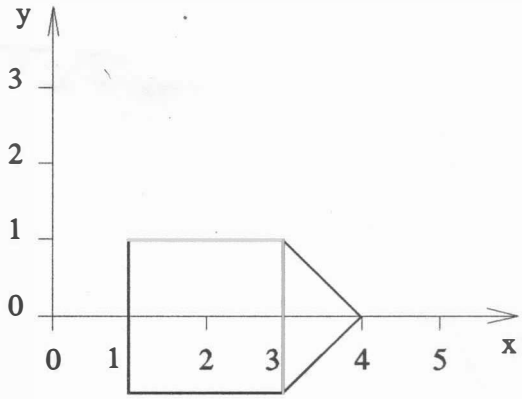
b)

po posunutí do počátku



c)

po otočení
o úhel



d)

po posunutí bodu A
do původní pozice

Obr.4.2.1

Celý postup lze přepsat pomocí rovnic

$$\mathbf{x}' = \mathbf{T}(-a, -b) \cdot \mathbf{x}$$

$$\mathbf{x}'' = \mathbf{R}(\alpha) \cdot \mathbf{x}'$$

$$\mathbf{x}''' = \mathbf{T}(a, b) \cdot \mathbf{x}''$$

tj.

$$\mathbf{x}''' = \mathbf{T}(a, b) \cdot \mathbf{R}(\alpha) \cdot \mathbf{T}(-a, -b) \cdot \mathbf{x}$$

lze psát, že

$$\mathbf{x}''' = \mathbf{Q}(a, b, \alpha) \cdot \mathbf{x}$$

kde:

$$\mathbf{Q}(a, b, \alpha) = \mathbf{T}(a, b) \cdot \mathbf{R}(\alpha) \cdot \mathbf{T}(-a, -b)$$

$$\begin{bmatrix} x''' \\ y''' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & a \\ 0 & 1 & b \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -a \\ 0 & 1 & -b \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Vyhodnocením výrazu pro **Q** dostáváme:

$$\mathbf{Q}(a, b, \alpha) = \left[\begin{array}{cc|c} \cos\alpha & -\sin\alpha & -a.\cos\alpha + b.\sin\alpha + a \\ \sin\alpha & \cos\alpha & -a.\sin\alpha - b.\cos\alpha + b \\ \hline 0 & 0 & 1 \end{array}\right]$$

Po dosazení za $\alpha = -\pi/2$ do $\mathbf{R}(\alpha)$ dostáváme:

$$\mathbf{R} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Po vyhodnocení maticového výrazu pro $\alpha = -\pi/2$ dostáváme:

$$\mathbf{Q}(a, b, -\pi/2) = \left[\begin{array}{cc|c} 0 & 1 & -b + a \\ -1 & 0 & -a + b \\ \hline 0 & 0 & 1 \end{array}\right]$$

Obecně platí, že obecnou matici transformace **Q** lze rozdělit na submatici $\mathbf{Q}_R$ rozměru 2 x 2 vyjadřující obecně rotaci a na matici $\mathbf{Q}_P$ rozměru 2 x 1 vyjadřující posuv, a tedy:

$$\mathbf{Q} = \left[\begin{array}{c|c} \mathbf{Q}_R & \mathbf{Q}_P \\ \hline 0 & 1 \end{array}\right]$$

Odtud vyplývá, že k reprezentaci obecné transformace v rovině stačí matice $\hat{\mathbf{Q}}$ rozměru 2 x 3 (toho využívá např. i GKS-2D), kde:

$$\hat{\mathbf{Q}} = \left[\begin{array}{c|c} \mathbf{Q}_R & \mathbf{Q}_P \end{array}\right]$$

Je zřejmé, že uvedený způsob provádění operací není příliš efektivní, a proto se ve skutečných aplikacích spočítají předem vztahy pro celou transformaci a vektory souřadnic **x** se pak vynásobí výslednou maticí.

V některých publikacích (zejména zahraničních) se používá místo sloupcových vektorů vektorů řádkových. Pak je nutné pořadí vektorů a matic operací obrátit a jednotlivé matice a vektory transponovat, neboť vztah

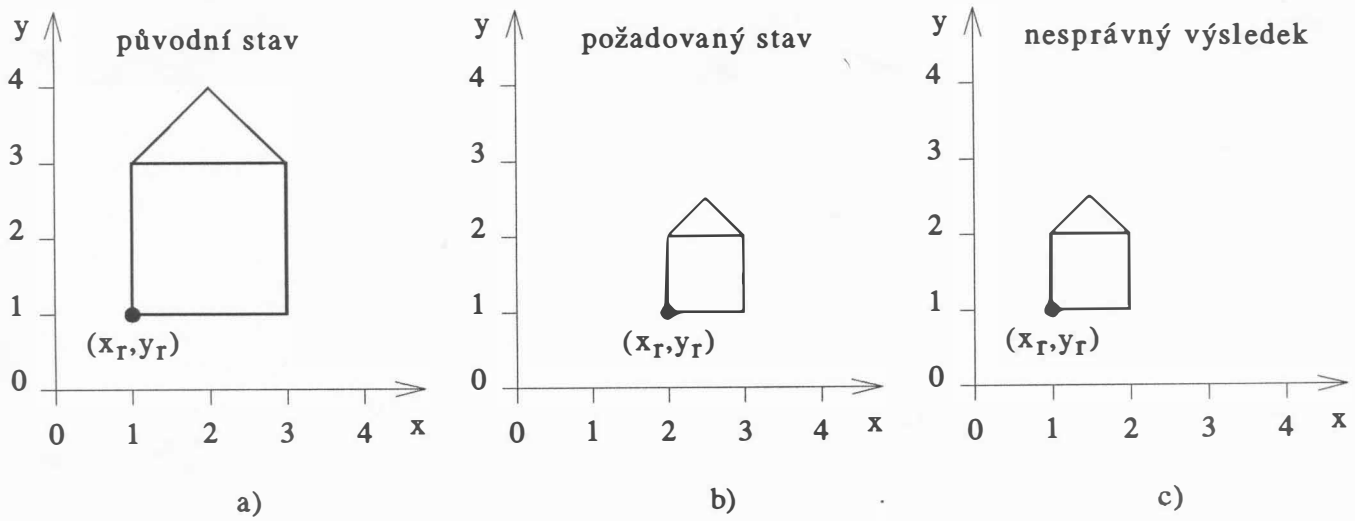
$$\mathbf{y} = \mathbf{A}\,\mathbf{B}\,\mathbf{x}$$

je možné přepsat jako

$$\mathbf{y}^T = \mathbf{x}^T\,\mathbf{B}^T\,\mathbf{A}^T$$

Relativní změna měřítka

V některých situacích může být i změna měřítka operací záludná. Uvažme jednoduchý objekt, který chceme zmenšit v poměru 1:2, viz obr.4.2.2.a.



Obr.4.2.2

Tento zdánlivě jednoduchý úkol může být realizován nesprávným způsobem, neboť vynásobíme-li souřadnice vrcholů transformační maticí **S**(Sx,Sy) (Sx = 0.5, Sy = 0.5), pak dostaneme situaci na obr.4.2.2.c. Objekt se nejen zmenší, ale zároveň se i **posune**, neboť se přepočtou i souřadnice referenčního bodu (x_r, y_r). Pokud chceme získat výstup, který je uveden na obr.4.2.2.b, pak je nutné:

- posunout objekt do počátku souřadného systému,
- provést přepočet souřadnic posunutého objektu podle požadovaného zmenšení,
- posunout objekt do původního referenčního bodu (x_r, y_r).

Tyto jednotlivé kroky lze popsat rovnicemi

$$\mathbf{x}' = \mathbf{T}(-x_r, -y_r) \cdot \mathbf{x}$$

$$\mathbf{x}'' = \mathbf{S}(Sx, Sy) \cdot \mathbf{x}'$$

$$\mathbf{x}''' = \mathbf{T}(x_r, y_r) \cdot \mathbf{x}''$$

tj.

$$\mathbf{x}''' = \mathbf{T}(x_r, y_r) \cdot \mathbf{S}(Sx, Sy) \cdot \mathbf{T}(-x_r, -y_r) \cdot \mathbf{x}$$

Pak lze psát

$$\mathbf{x}''' = \mathbf{Q}(a, b, \alpha) \cdot \mathbf{x}$$

kde

$$\mathbf{Q}(a, b, \alpha) = \mathbf{T}(x_r, y_r) \cdot \mathbf{S}(Sx, Sy) \cdot \mathbf{T}(-x_r, -y_r)$$

$$\begin{bmatrix} x''' \\ y''' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_r \\ 0 & 1 & y_r \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_r \\ 0 & 1 & -y_r \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Po vyhodnocení výrazu pro **Q** dostáváme:

$$\mathbf{Q}(x_r, y_r, Sx, Sy) = \left[\begin{array}{cc|c} Sx & 0 & (1 - Sx).x_r \\ 0 & Sy & (1 - Sy).y_r \\ \hline 0 & 0 & 1 \end{array}\right]$$

Dříve uvedené základní transformace, jako posuv, rotace a změna měřítka, obsahují pravděpodobně všechny programové celky podporující počítačovou grafiku. Jen některé programové celky však poskytují i další "odvozené" transformace, které jsou vhodné pro některé aplikace. Dvě takové operace jsou zrcadlení a zkosení.

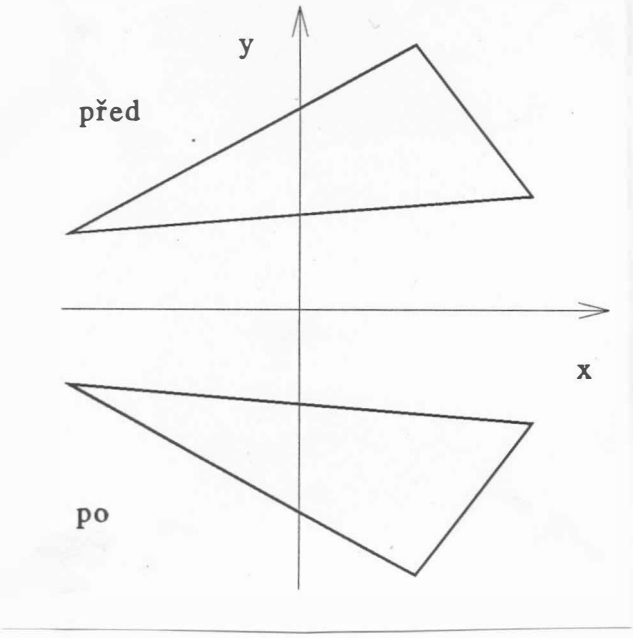
Zrcadlení

Zrcadlení je operace, kdy je objekt transformován do takové pozice, která vznikne zrcadlením vzhledem k nějaké ose, či bodu. Nejjednodušší je případ, kdy osa zrcadlení je totožná se souřadnou osou. Pro případ, že osou zrcadlení je osa x, viz obr.4.2.3, pak platí

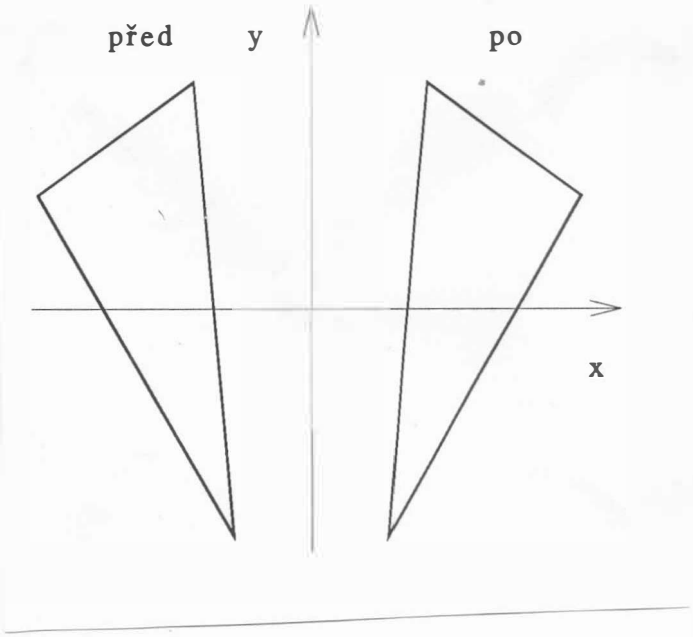
$$x' = x \qquad y' = -y$$

tj. v maticovém tvaru

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



Obr.4.2.3



Obr.4.2.4

V případě, že osou zrcadlení je osa y, viz obr.4.2.4, pak platí

$$x' = -x \qquad y' = y$$

tj. v maticovém tvaru

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Pro případ, že osou zrcadlení je osa $y = x$, resp. osa $y = -x$, viz obr.4.2.5 a obr.4.2.6, pak platí

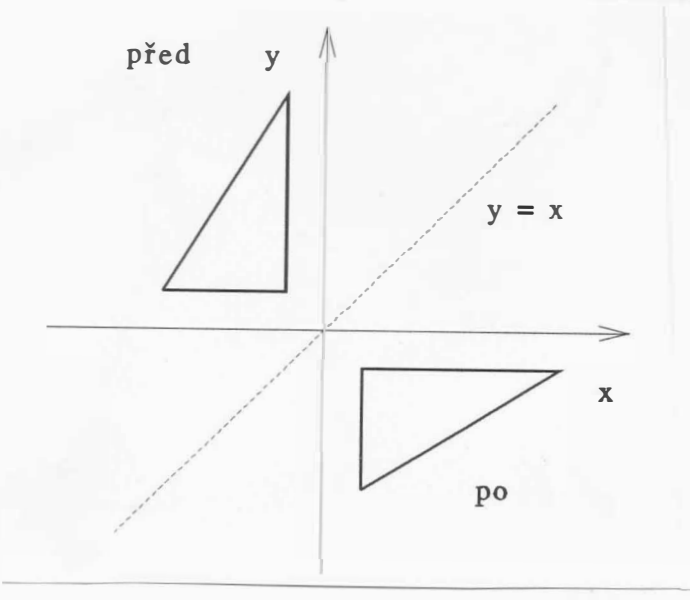
$$x' = y \qquad y' = x \quad , \text{ resp.} \qquad x' = -y \qquad y' = -x$$

tj. v maticovém tvaru

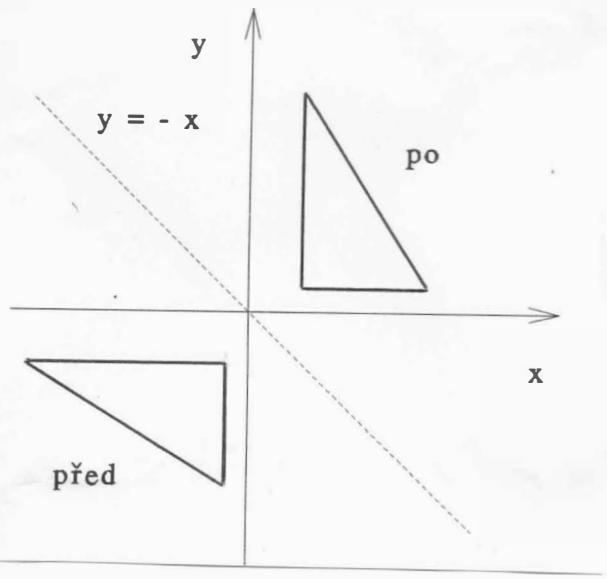
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \circ \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

resp.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \circ \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



Obr. 4.2.5



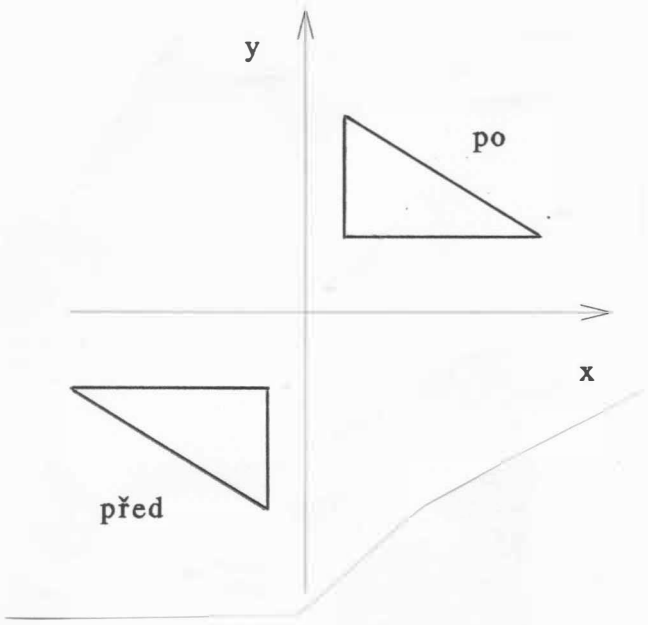
Obr. 4.2.6

Posledním případem je zrcadlení vzhledem k počátku, viz obr.4.2.7, kdy je transformační vztah popsán rovnicemi

$$x' = -x \qquad y' = -y$$

tj. v maticovém tvaru

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \circ \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



Obr. 4.2.7

Uvedených operací zrcadlení lze s výhodou využít při realizaci různých objektů, které jsou umístěny zrcadlově, zejména v oblasti strojírenství a elektrotechniky.

Zkosení

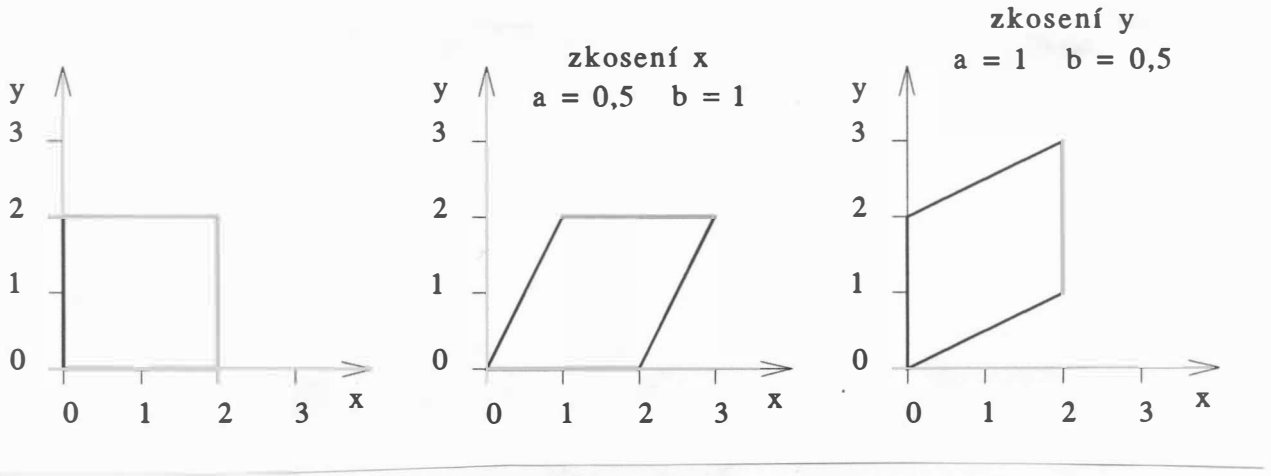
Zkosení je další operací, která je někdy k dispozici. Obvykle se rozeznává tzv. zkosení x, které je definováno maticí

$$H_x(a) = \begin{bmatrix} 1 & a & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

a zkosení y, které je určeno maticí

$$H_y(b) = \begin{bmatrix} 1 & 0 & 0 \\ b & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Na obr. 4.2.8 je uvedena oblast a výsledky po zkosení x a zkosení y.



Obr. 4.2.8

Příklad

Odvoďte vztahy pro geometrickou transformaci, která je dána maticí

$$H_{xy}(a) = \begin{bmatrix} 1 & a & 0 \\ b & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

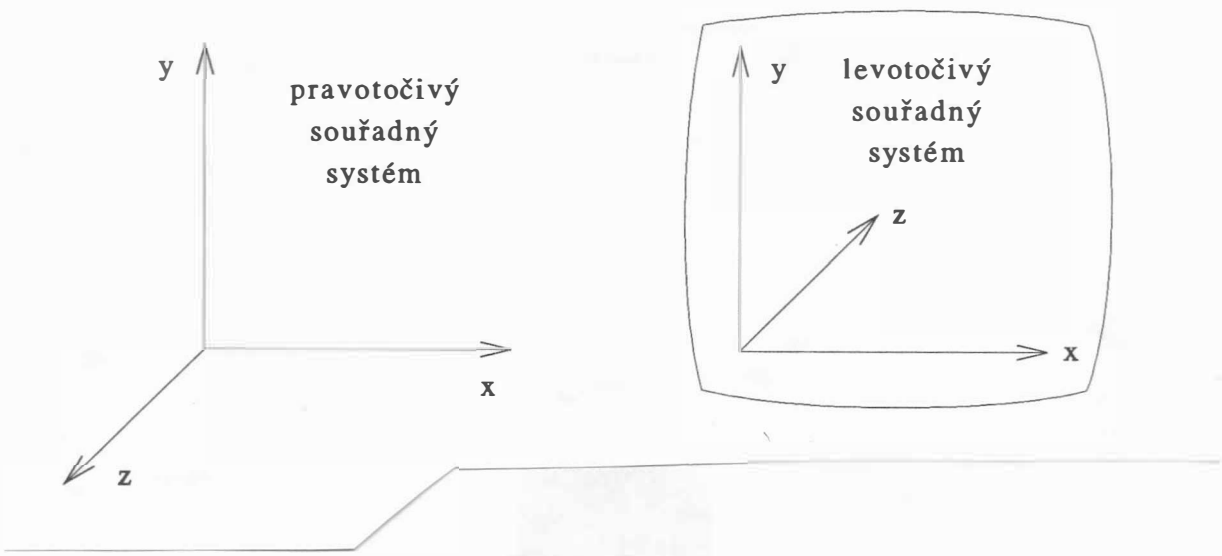
přičemž volte a = 0.5 , b = 0.7 . Výsledek operace ilustrujte na případě z obr. 4.2.8.a.

Příklad

Odvoďte vztahy pro vzájemnou transformaci mezi obdélníkem a lichoběžníkem (počátek souřadného systému ztotoněte vždy s jedním vrcholem. Diskutujte vlastnosti této operace a ukažte možnosti aplikací.

4.3 Maticová reprezentace operací pro třírozměrný prostor

Maticové reprezentace pro operace v rovině vyžadovaly matice o rozměru 3 x 3. Analogicky pro operace v třírozměrném prostoru založené na reprezentaci bodů homogenními souřadnicemi budeme požadovat matice 4 x 4. Na rozdíl od dvourozměrného případu, kdy je orientace kartézského souřadného systému vžitá, je nutné u třírozměrného prostoru jednoznačně definovat směr jednotlivých os.



Obr. 4.3.1

Na obr. 4.3.1 je ukázán jak pravotočivý, tak i levotočivý souřadný systém s vyznačeným kladným směrem rotací v jednotlivých rovinách. Dále budeme implicitně používat pravotočivý souřadný systém, pokud nebude řečeno jinak, i když levotočivý souřadný systém by byl "přirozenější" pro aplikace v případě grafických výstupních zařízení. Jednotlivé operace pro třírozměrný souřadný systém jsou vlastně rozšířením transformací pro dvourozměrný souřadný systém.

Posuv

Posuv o vzdálenost Dx, Dy, Dz lze vyjádřit rovnicemi:

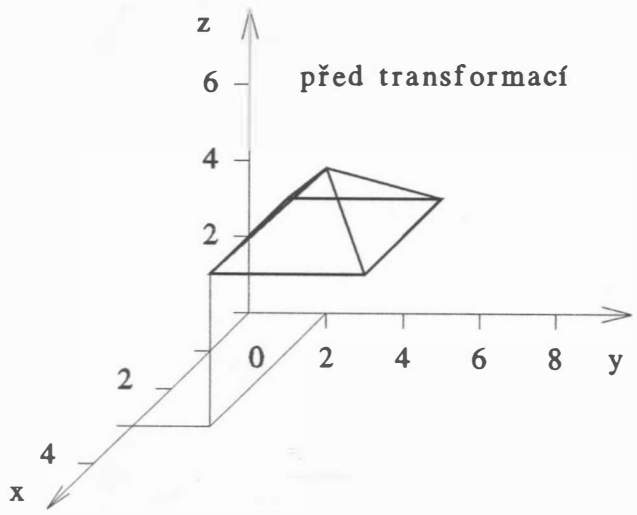
$$x' = Dx + x$$
$$y' = Dy + y$$
$$z' = Dz + z$$

V maticovém tvaru pak

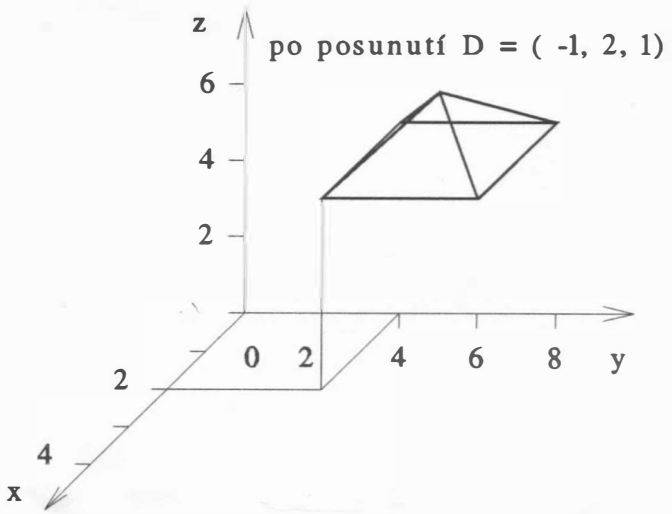
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & Dx \\ 0 & 1 & 0 & Dy \\ 0 & 0 & 1 & Dz \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$\mathbf{x'} = \mathbf{T}\ (\ Dx\ ,\ Dy\ ,\ Dz\ )\ .\ \mathbf{x}$$

Na obr.4.3.2 je znázorněna situace před posunutím. Na obr.4.3.3 je pak stav po posunutí dané vektorem (-1,2,1).



Obr.4.3.2



Obr.4.3.3

Změna měřítka

Operace změny měřítka je určena rovnicemi:

$$x' = Sx\ .\ x$$
$$y' = Sy\ .\ y$$
$$z' = Sz\ .\ z$$

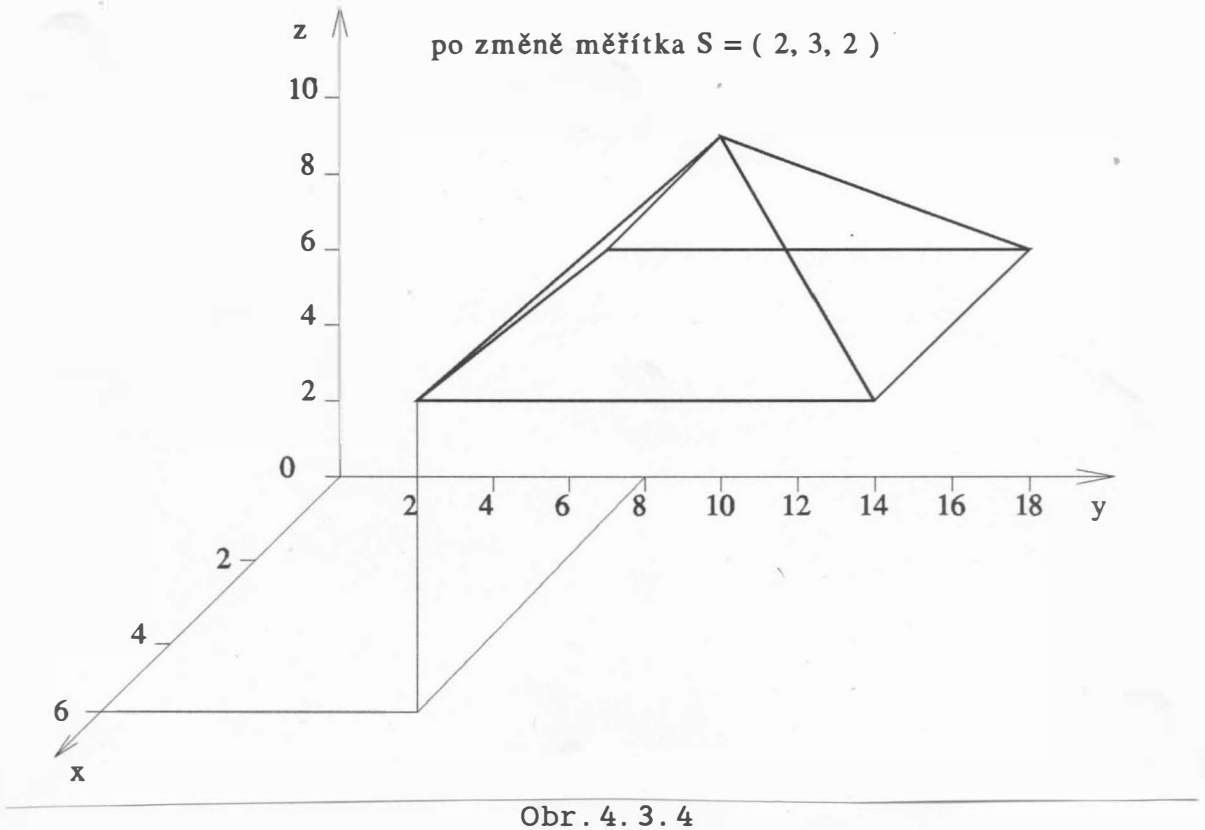
tj. v maticovém tvaru

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

resp.

$$\mathbf{x'} = \mathbf{S}\ (\ Sx\ ,\ Sy\ ,\ Sz\ )\ .\ \mathbf{x}$$

Na obr.4.3.4 je znázorněna situace po provedení změny měřítka pro Sx=2, Sy=3, Sz=2 aplikované na původní situaci z obr.4.3.2. Je nutné upozornit, že dochází i k **posunutí** zobrazovaného objektu. Pokud by se měla aplikovat změna měřítka **pouze** na objekt, je nutné postupovat obdobně jako v kap.4.1 pro případ relativní změny měřítka v E_2.



Obr.4.3.4

Rotace

Rotace kolem osy z, tj. v rovině xy, o úhel α je popsána rovnicemi:

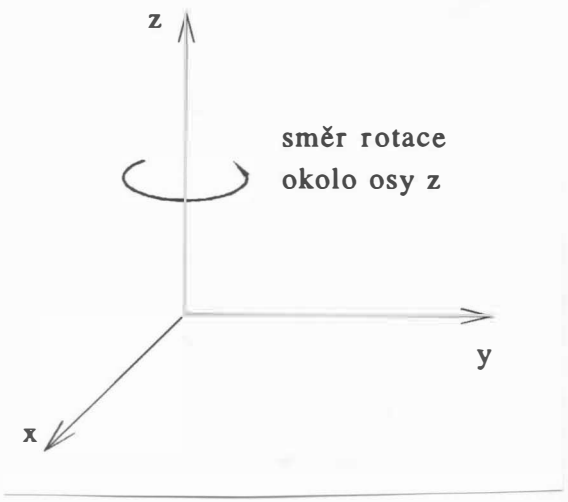
$$\begin{aligned} x' &= x\ .\ \cos\alpha\ -\ y\ .\ \sin\alpha \\ y' &= x\ .\ \sin\alpha\ +\ y\ .\ \cos\alpha \\ z' &= z \end{aligned}$$

V maticovém tvaru

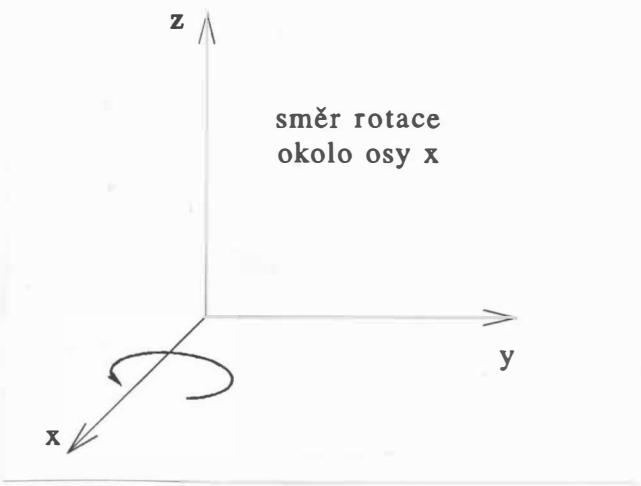
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

resp.

$$\mathbf{x}' = \mathbf{R}_{xy}(\alpha) \,.\, \mathbf{x} \qquad \text{resp.} \quad \mathbf{x}' = \mathbf{R}_{z}(\alpha) \,.\, \mathbf{x}$$



Obr. 4.3.5



Obr. 4.3.6

Rotace kolem osy x, tj. v rovině yz, o úhel β:

$$\begin{aligned} x' &= x \\ y' &= y \,.\, \cos\beta - z \,.\, \sin\beta \\ z' &= y \,.\, \sin\beta + z \,.\, \cos\beta \end{aligned}$$

V maticovém tvaru pak

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\beta & -\sin\beta & 0 \\ 0 & \sin\beta & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$\mathbf{x}' = \mathbf{R}_{yz}(\beta) \,.\, \mathbf{x} \qquad \text{resp.} \quad \mathbf{x}' = \mathbf{R}_{x}(\beta) \,.\, \mathbf{x}$$

Rotace kolem osy y, tj. v rovině zx, o úhel γ:

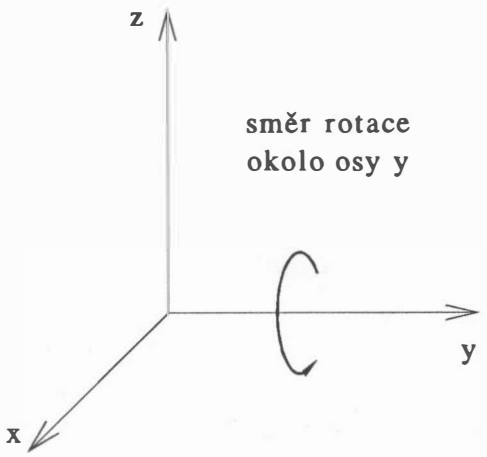
$$\begin{aligned} x' &= x \,.\, \cos\gamma + z \,.\, \sin\gamma \\ y' &= y \\ z' &= -x \,.\, \sin\gamma + z \,.\, \cos\gamma \end{aligned}$$

V maticovém tvaru pak

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\gamma & 0 & \sin\gamma & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\gamma & 0 & \cos\gamma & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

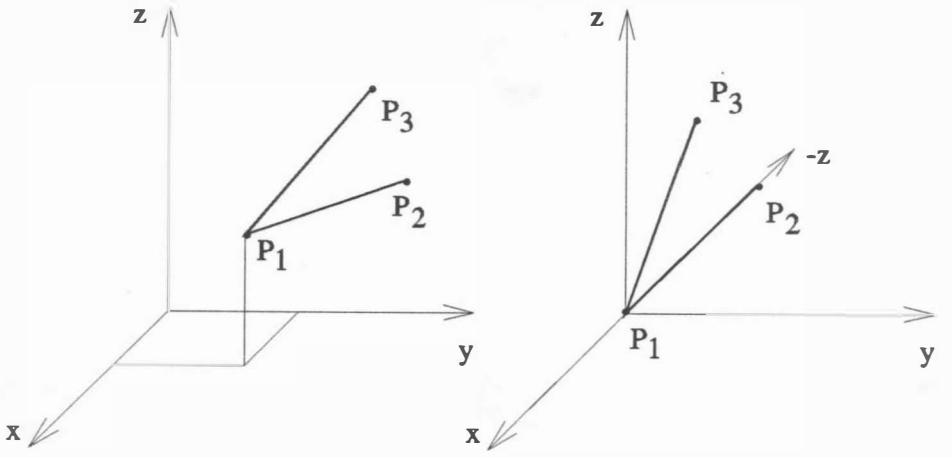
resp.

$$\mathbf{x}' = \mathbf{R}_{zx}(\gamma) \cdot \mathbf{x} \qquad \text{resp.} \qquad \mathbf{x}' = \mathbf{R}_{y}(\gamma) \cdot \mathbf{x}$$



Obr. 4. 3. 7

Při rotaci okolo osy y je nutné dát pozor na polohu znaménka v matici **R**, neboť v případě umístění znaménka minus do první řádky bychom dostali matici rotace pro levotočivý souřadný systém. Inverzní operace k operacím uvedeným lze získat podobným způsobem jako v případě dvourozměrného souřadného systému.



a) počáteční stav b) výsledný stav

Obr. 4. 4. 1.

4.4 Řetězení třírozměrných transformací

Podobně jako v dvourozměrném případě lze jednotlivé operace skládat. Vzhledem k tomu, že maticové násobení není obecně komutativní, je výsledek dán nejen jednotlivými parametry, ale také pořadím provádění jednotlivých operací. K ilustraci zřetězení jednotlivých operací uveďme příklad z [44]. Pokusme se vektory z obr.4.4.1.a uspořádat do takové polohy, že úsečka P_1P_2 je ztotožněna se zápornou poloosou z a úsečka P_1P_3 leží v rovině yz. Výsledek je uveden na obr.4.4.1.b.

Řešení daného problém může být rozděleno do několika kroků, a to:

1. Posuv bodů tak, aby bod P_1 byl v počátku.
2. Rotace bodů okolo osy y tak, aby úsečka P_1P_2 ležela v rovině yz.
3. Rotace bodů okolo osy x tak, by úsečka P_1P_2 ležela na záporné poloose z.
4. Rotace bodů okolo osy z tak, aby úsečka P_1P_3 ležela v rovině yz.

Předpokládejme, že homogenní souřadnice jednotlivých bodů jsou:

$$\mathbf{p}_1 = [\, x_1 , y_1 , z_1 , 1 \,]^T \qquad \mathbf{p}_2 = [\, x_2 , y_2 , z_2 , 1 \,]^T$$

$$\mathbf{p}_3 = [\, x_3 , y_3 , z_3 , 1 \,]^T$$

Pak matice reprezentující posuv má tvar:

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & -x_1 \\ 0 & 1 & 0 & -y_1 \\ 0 & 0 & 1 & -z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Po provedení operace posuvu

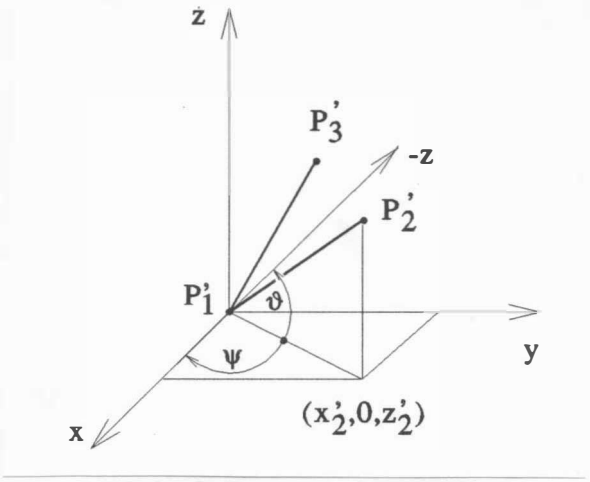
$$\mathbf{x}' = \mathbf{T}(-x_1 , -y_1 , -z_1) \cdot \mathbf{x}$$

dostáváme stav, který je uveden na obr.4.4.2, přičemž nové souřadnice bodů jsou:

$$\mathbf{p}'_1 = [\, 0 , 0 , 0 , 1 \,]^T$$

$$\mathbf{p}'_2 = [\, x_2 - x_1 , y_2 - y_1 , z_2 - z_1 , 1 \,]^T$$

$$\mathbf{p}'_3 = [\, x_3 - x_1 , y_3 - y_1 , z_3 - z_1 , 1 \,]^T$$



výsledek po posunutí

Obr. 4.4.2

Nyní je nutné provést rotaci okolo osy y tak, aby úsečka P_1P_2 ležela v rovině yz. K tomu je nezbytné, určit úhel rotace, která je prováděna o nezáporný úhel ϑ , pro který platí:

$$\cos \psi = \frac{z'_2}{D_1} = \cos (\pi/2 + \vartheta) \qquad \sin \psi = \frac{x'_2}{D_1}$$

$$\cos \vartheta = \frac{-z'_2}{D_1} \qquad \sin \vartheta = \frac{x'_2}{D_1}$$

kde

$$D_1 = \sqrt{ (z'_2)^2 + (x'_2)^2 }$$

Po dosazení dostáváme

$$\cos \vartheta = \frac{- (z_2 - z_1)}{D_1} \qquad \sin \vartheta = \frac{x_2 - x_1}{D_1}$$

kde:

$$D_1 = \sqrt{ (z_2 - z_1)^2 + (x_2 - x_1)^2 }$$

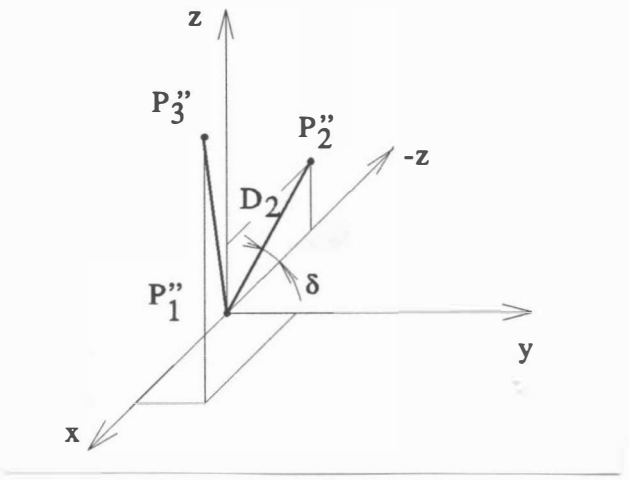
Po provedení operace rotace okolo osy y, tj. v rovině zx

$$\mathbf{x''} = \mathbf{R}_{zx} (\vartheta) \, . \, \mathbf{x'}$$

kde:

$$\mathbf{R}_{zx}(\vartheta) = \begin{bmatrix} \cos \vartheta & 0 & \sin \vartheta & 0 \\ 0 & 1 & 0 & 0 \\ - \sin \vartheta & 0 & \cos \vartheta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

dostáváme stav, který je na obr.4.4.3.



výsledek po rotaci okolo osy y

Obr.4.4.3

Po dosazení souřadnic bodu P_2' do rovnice pro rotaci dostáváme souřadnice bodu P_2'' , a to:

$$\mathbf{p}_2'' = [\ 0\ ,\ y_2 - y_1\ ,\ \frac{-(x_2 - x_1)^2 - (z_2 - z_1)^2}{D_1}\ ,\ 1\ ]^T$$

Rotace okolo osy x, tj. v rovině yz, musí být provedena o úhel δ , pro který platí

$$\cos(\delta) = -\frac{z_2''}{D_2} \qquad \sin(\delta) = \frac{y_2''}{D_2}$$

kde

$$D_2 = \sqrt{(y_2'')^2 + (z_2'')^2}$$

Vzhledem k obrácené orientaci musíme provést rotaci o úhel $-\delta$, přičemž

$$\cos(-\delta) = \cos(\delta) \qquad \sin(-\delta) = -\sin(\delta)$$

Po dosazení a úpravě dostáváme:

$$D_2 = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

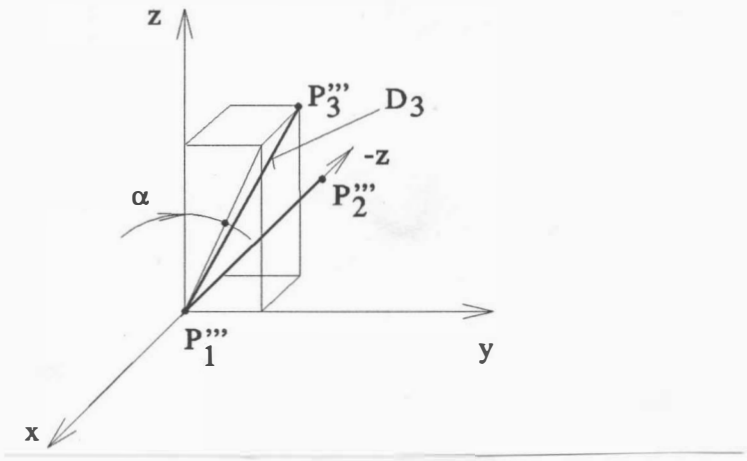
Po provedení operace rotace kolem osy x, tj. v rovině yz

$$\mathbf{x}''' = \mathbf{R}_{yz}(-\delta)\ \mathbf{x}''$$

kde

$$R_{yz}(-\delta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(-\delta) & -\sin(-\delta) & 0 \\ 0 & \sin(-\delta) & \cos(-\delta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

dostaneme stav, který je na obr.4.4.4.



výsledek po rotaci okolo osy x

Obr.4.4.4

Výsledné souřadnice bodu P_2'' jsou zřejmé, a to:

$$p_2'' = [\ 0\ ,\ 0\ ,\ -D_2\ ,\ 1\ ]^T$$

Posledním krokem je rotace okolo osy z o úhel α tak, aby úsečka P_1P_3 ležela v rovině yz. V daném případě platí, že

$$\cos\alpha = \frac{y_3''}{D_3} \qquad \sin\alpha = \frac{x_3''}{D_3}$$

kde:

$$D_3 = \sqrt{(x_3'')^2 + (y_3'')^2}$$

Poznamenejme, že souřadnice bodu P_3'' by byly získány aplikací stejných transformací, které byly uvedeny výše pro transformaci bodu P_2''. Uvedené operace mohou být zřetězeny tak, že

$$x'''' = R_{xy}(\alpha)\ .\ R_{yz}(\delta)\ .\ R_{zx}(\vartheta)\ .\ T(-x_1,-y_1,-z_1)\ .\ x$$

a jednotlivé operace mohou být sloučeny tak, že

$$x'''' = Q \cdot x$$

kde matice:

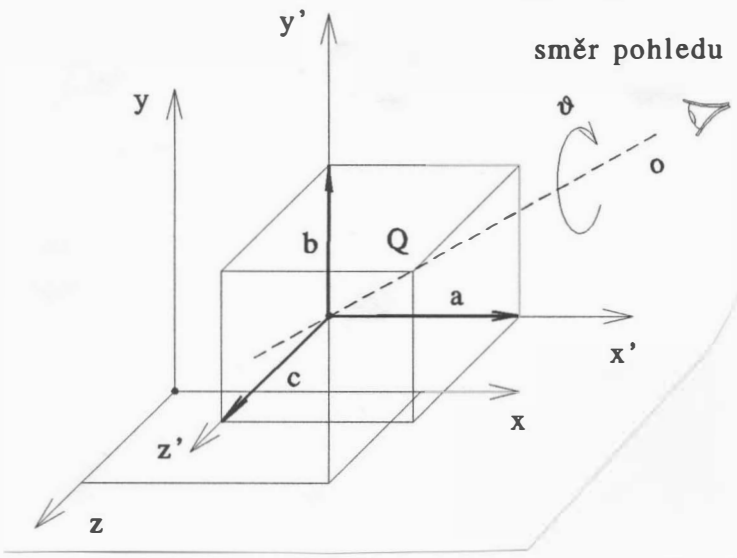
$$Q = R_{xy}(\alpha) \cdot R_{yz}(\delta) \cdot R_{zx}(\vartheta) \cdot T(-x_1, -y_1, -z_1)$$

Obecně opět platí, že matici Q rozměru 4 x 4 lze rozdělit na submatici Q_R rozměru 3 x 3 vyjadřující obecně rotaci a matici Q_P vyjadřující posuv. Pak lze psát:

$$Q = \left[\begin{array}{c|c} Q_R & Q_P \\ \hline 0 & 1 \end{array}\right]$$

Odtud vyplývá, že k reprezentaci obecné transformace v prostoru E_3 stačí matice $\hat{Q}$ rozměru 3 x 4 (toho využívá např. i GKS-3D), kde:

$$\hat{Q} = \left[\begin{array}{c|c} Q_R & Q_P \end{array}\right]$$



ve skutečnosti nebudeme posouvat souřadný systém, ale objekty

Obr. 4.5.1

4.5 Rotace kolem libovolné osy

Rotace kolem libovolné osy je velmi častou úlohou dekomponovatelnou na jednodušší kroky. Definujme bod $Q(x_q, y_q, z_q)$ jako bod, kterým osa **o** prochází, a směrový jednotkový vektor, který je dán pomocí směrového vektoru (a,b,c). Jednotlivé kroky pro pootočení o úhel ϑ jsou:

- posuv objektu do nového souřadného systému s počátkem v bodě Q (matice **T**)
- provedení rotace objektu okolo osy x a y tak, aby jednotkový vektor (a,b,c) byl po rotaci ztotožněn s jednotkovým vektorem (0,0,1) rovnoběžným s osou z (matice $\mathbf{R}_{yz}$, $\mathbf{R}_{zx}$)
- pootočení objektu o úhel ϑ
- provedení inverzních rotací okolo os y a x (matice $\mathbf{R}_{zx}^{-1}, \mathbf{R}_{yz}^{-1}$)
- inverzní posuv objektu (matice $\mathbf{T}^{-1}$)

Účelem 4. a 5. kroku je návrat do původního souřadného systému. Souřadný systém, který byl založen v rámci 1. a 2. kroku, slouží pouze k dočasným účelům. Kompletní transformace je pak definována maticí, která je dána výrazem:

$$\mathbf{T}^{-1} \cdot \mathbf{R}_{yz}^{-1} \cdot \mathbf{R}_{zx}^{-1} \cdot \mathbf{R}_{\vartheta} \cdot \mathbf{R}_{zx} \cdot \mathbf{R}_{yz} \cdot \mathbf{T}$$

Transformace posuvu souřadného systému do bodu $Q(x_q, y_q, z_q)$ je dána maticí:

$$\mathbf{T}(-x_q, -y_q, -z_q) = \begin{bmatrix} 1 & 0 & 0 & -x_q \\ 0 & 1 & 0 & -y_q \\ 0 & 0 & 1 & -z_q \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

tj.

$$\mathbf{x}' = \mathbf{T} \cdot \mathbf{x}$$

Matice rotace $\mathbf{R}_{yz}(\alpha)$ pro rotaci kolem osy x' je definována:

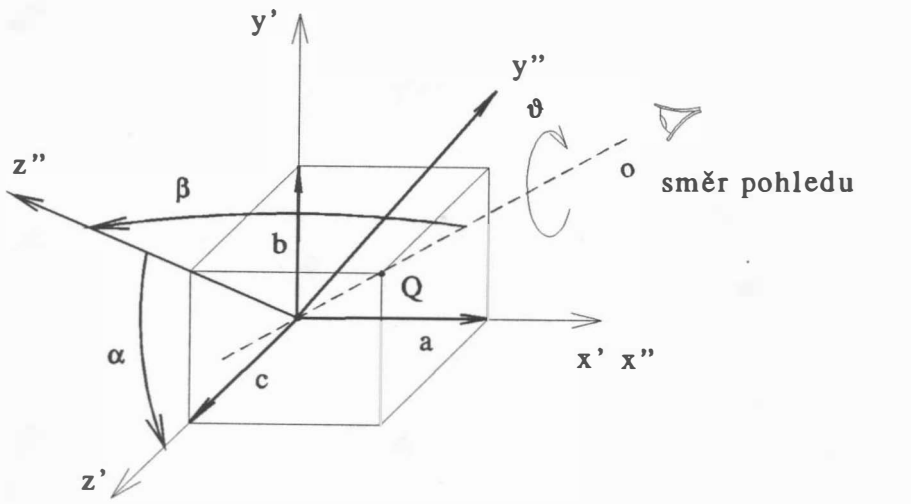
$$\mathbf{R}_{yz}(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

kde:

$$\cos\alpha = \frac{c}{v} \qquad \sin\alpha = \frac{b}{v} \qquad v = \sqrt{b^2 + c^2}$$

tj. $\mathbf{x}'' = \mathbf{R}_{yz}(\alpha) \cdot \mathbf{x}'$

Výsledek po posunu a rotaci kolem osy x'' o úhel α je uveden na obr. 4.5.2.



Obr. 4.5.2

Nyní je nutné provést rotaci kolem osy y'' o úhel β, který je určen vztahy:

$$\cos\beta = \frac{v}{r} \qquad \sin\beta = \frac{-a}{r}$$

kde $r = \sqrt{a^2 + b^2 + c^2}$ a $v = \sqrt{b^2 + c^2}$

V případě, že (a,b,c) je jednotkový směrový vektor, platí:

$$a^2 + b^2 + c^2 = 1 \ .$$

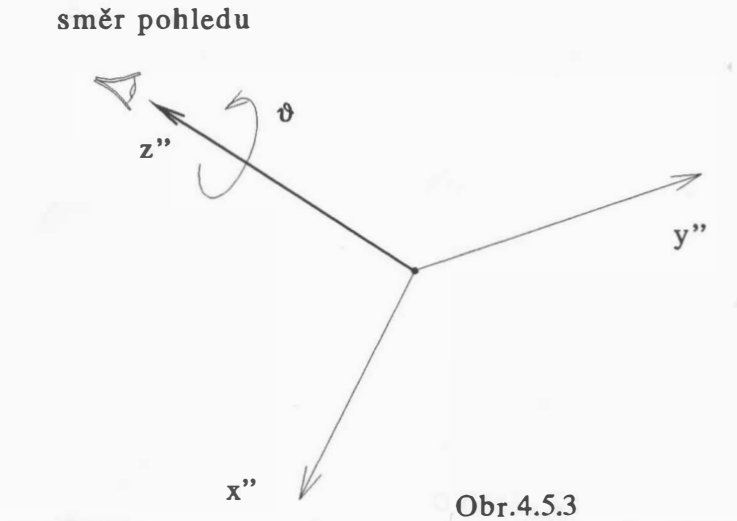
Matice rotace $\mathbf{R}_{zx}(\beta)$ pro rotaci kolem osy y'' má tvar:

$$\mathbf{R}_{zx}(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

tj. $\mathbf{x}''' = \mathbf{R}_{zx}(\beta) \cdot \mathbf{x}''$

Nyní zbývá provést rotaci objektu o úhel ϑ kolem osy o, která je ztotožněna s osou z''' (tj. rotaci v rovině xy nového souřadného systému), viz obr.4.5.3. Matice rotace **R** má tvar:

$$\mathbf{R}_{xy}(\vartheta) = \begin{bmatrix} \cos\vartheta & \sin\vartheta & 0 & 0 \\ -\sin\vartheta & \cos\vartheta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Obr.4.5.3

Pro matici $\mathbf{R}_{xy}(\vartheta)$ uvažujeme pootočení o úhel okolo osy z v kladném smyslu. Pak matice rotace $\mathbf{R}_{xy}(\vartheta)$ musí být pro souřadný systém pozorovatele, tj. pro levotočivý souřadný systém, transponována.

Obr.4.5.4

Příklad

Je dána krychle, viz obr.4.5.4.a, s vrcholy:

A = (0,0,0), B = (2,0,0), C = (2,2,0), D = (0,2,0),
E = (0,0,2), F = (2,0,2), G = (2,2,2), H = (0,2,2).

Pootočte tuto krychli okolo osy procházející body (0,0,0) a (2,2,2) o úhel 60°.

Označíme-li

$$W = R_{zx} \cdot R_{yz} \cdot T$$

pak lze celou transformaci vyjádřit výrazem

$$R = W^{-1} \cdot R(\vartheta) \cdot W$$

kde **W** vyjadřuje souhrnně posuv a rotaci objektu do požadované polohy a **R**(ϑ) je maticí rotace kolem zadané osy.

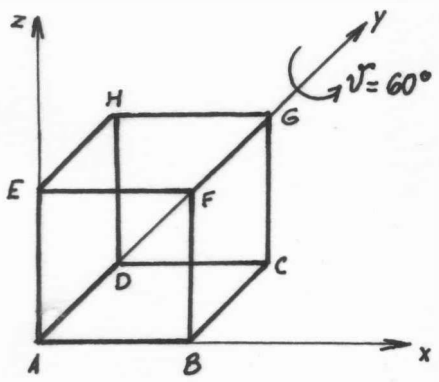
Celková matice transformace **R** má tvar:

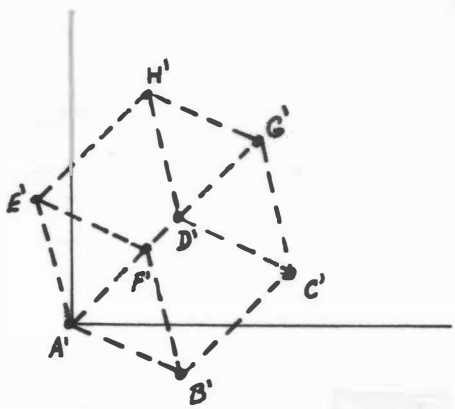
$$R = \begin{bmatrix} 2/3 & -1/3 & 2/3 & 0 \\ 2/3 & 2/3 & -1/3 & 0 \\ -1/3 & 2/3 & 2/3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

a vrcholy dané krychle po rotaci mají souřadnice:

A' = (0 , 0 , 0) B' = (4/3 , 4/3 , -2/3)
C' = (2/3 , 8/3 , 2/3) D' = (-2/3 , 4/3 , 4/3)
E' = (4/3 , -2/3 , 4/3) F' = (8/3 , 2/3 , 2/3)
G' = (2 , 2 , 2) H' = (2/3 , 2/3 , 8/3)

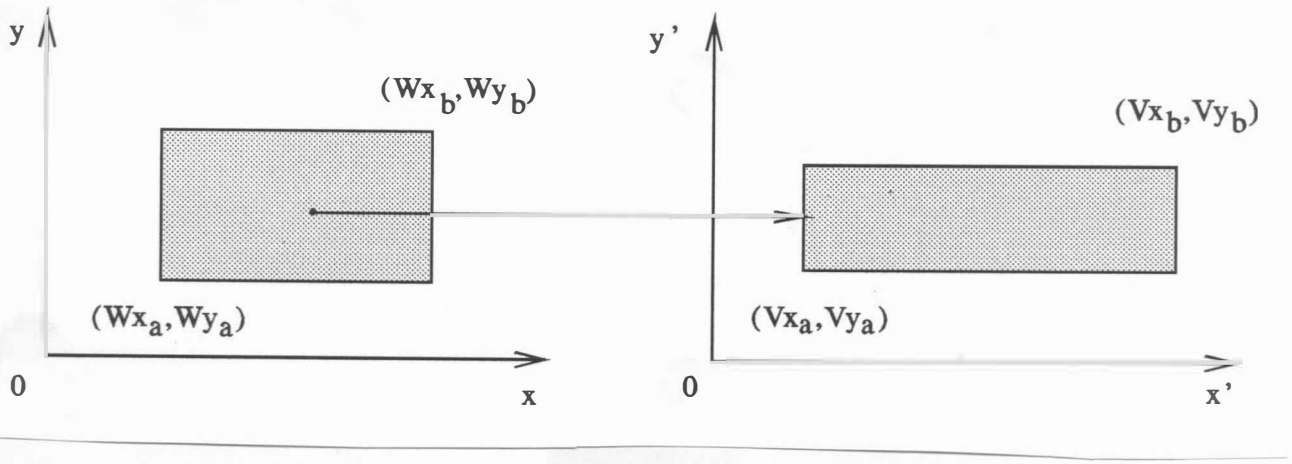
Výsledek po rotaci o úhel ϑ je na obr.4.5.4.b.





4.6 Transformace "okno-pohled"

Častou úlohou je zobrazení kresleného motivu z intervalu $\langle Wx_a, Wx_b \rangle$ x $\langle Wy_a, Wy_b \rangle$ do intervalu $\langle Vx_a, Vx_b \rangle$ x $\langle Vy_a, Vy_b \rangle$, viz obr.4.6.1. Tato transformace se používá zejména ve spojení s operací ořezávání, neboť obvykle nechceme zobrazit tu část kresleného motivu, která je mimo zobrazované okno.



Obr.4.6.1

I když jde o jednoduchou transformaci, je vhodné ji odvodit, zejména z důvodu častého používání. Je zřejmé, že transformace bude určena následujícími základními kroky:

- posuvem, který je reprezentován maticí T_1, bodu (Wx_a, Wy_a) do bodu (0,0) , tj. posuvem do počátku souřadného systému,
- změnou měřítka na ose x a y, která je reprezentována maticí S, tak, aby se interval $\langle Wx_a, Wx_b \rangle$ transformoval na interval $\langle Vx_a, Vx_b \rangle$, a podobně pro osu y,
- posuvem, který je reprezentován maticí T_2, původního bodu (Wx_a, Wy_a) do bodu (Vx_a, Vy_a).

Pak lze psát

$$\mathbf{x'} = \mathbf{Q} \cdot \mathbf{x} \qquad \text{kde} \qquad \mathbf{Q} = T_2 \cdot S \cdot T_1$$

přičemž:

$$T_1 = \begin{bmatrix} 1 & 0 & -Wx_a \\ 0 & 1 & -Wy_a \\ 0 & 0 & 1 \end{bmatrix} \qquad T_2 = \begin{bmatrix} 1 & 0 & Vx_a \\ 0 & 1 & Vy_a \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{S} = \begin{bmatrix} (Vx_b - Vx_a)/(Wx_b - Wx_a) & 0 & 0 \\ 0 & (Vy_b - Vy_a)/(Wy_b - Wy_a) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Matice **Q** reprezentující celou transformaci je pak určena takto

$$\mathbf{Q} = \begin{bmatrix} \alpha & 0 & \gamma \\ 0 & \beta & \delta \\ 0 & 0 & 1 \end{bmatrix}$$

kde: $\alpha = (Vx_b - Vx_a) / (Wx_b - Wx_a)$ $\quad \gamma = Vx_a - \alpha\, Wx_a$

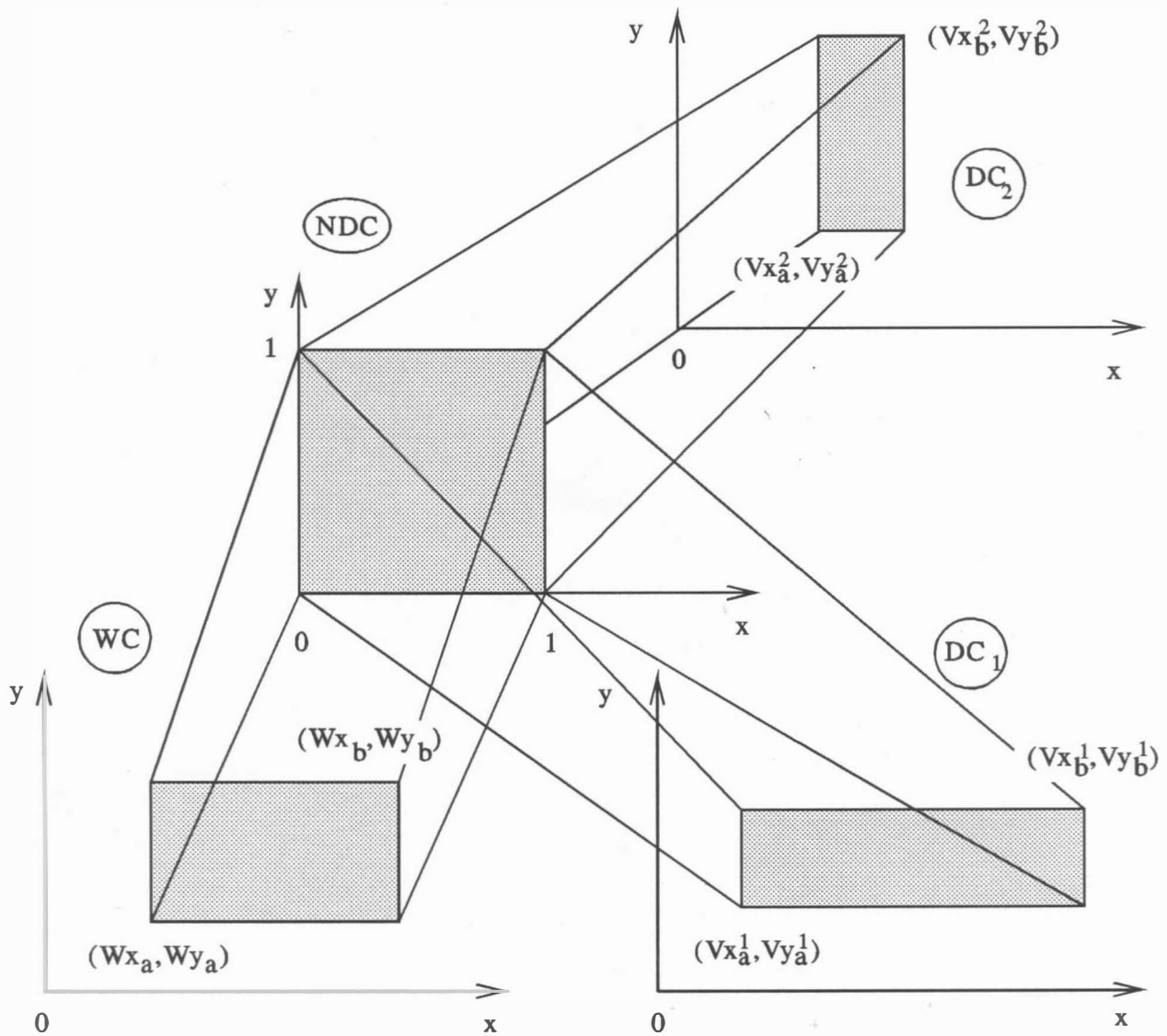
$\beta = (Vy_b - Vy_a) / (Wy_b - Wy_a)$ $\quad \delta = Vy_a - \beta\, Wy_a$

Je zřejmé, že pokud by některé části kresby přesahovaly zadaný interval, je nutné tyto části vhodně odříznout.

Takto složenou transformaci je možné používat, nicméně z mnoha praktických důvodů je vhodné tuto transformaci rozdělit na transformace dvě, a to:

- na transformaci z intervalu $\langle Wx_a, Wx_b \rangle \times \langle Wy_a, Wy_b \rangle$ na interval $\langle 0,1 \rangle \times \langle 0,1 \rangle$, reprezentovanou maticí $\mathbf{T}'_1$,
- na transformaci z intervalu $\langle 0,1 \rangle \times \langle 0,1 \rangle$ na interval $\langle Vx_a, Vx_b \rangle \times \langle Vy_a, Vy_b \rangle$, reprezentovanou maticí $\mathbf{T}'_2$.

Tímto způsobem dojde k rozpadu transformace závislé na velikostech okna a pohledu na dvě na sobě nezávislé transformace. Velikost okna je obvykle dána velikostí kreslicí plochy na výstupním zařízení, a tedy je **vlastností** výstupního zařízení, zatímco velikost zobrazovaného motivu, tj. velikost okna, je dána požadavky aplikačního programu, viz obr.4.6.2. Tímto způsobem lze docílit nezávislosti grafického výstupu na **aktuálním** výstupním zařízení. Norma **GKS** [56] pak přijala tento způsob řešení nezávislosti grafického výstupu. Prostor uživatelských souřadnic, tj. prostor, ve kterém se nachází interval $\langle Wx_a, Wx_b \rangle \times \langle Wy_a, Wy_b \rangle$, je označován jako **WC** (**W**orld **C**oordinates), prostor normalizovaných souřadnic, tj. interval $\langle 0,1 \rangle \times \langle 0,1 \rangle$, je označen jako **NDC** (**N**ormalized **D**evice **C**oordinates) a prostor souřadnic zařízení, tj. prostor, ve kterém se nachází interval $\langle Vx_a, Vx_b \rangle \times \langle Vy_a, Vy_b \rangle$, je označen jako **DC** (**D**evice **C**oordinates).


y
(Vx_b^2, Vy_b^2)
DC_2
NDC
y
1
(Vx_a^2, Vy_a^2)
0
x
x
WC
0
1
DC_1
y
y
(Wx_b, Wy_b)
(Vx_b^1, Vy_b^1)
(Wx_a, Wy_a)
(Vx_a^1, Vy_a^1)
0
x
0
x

Obr. 4.6.2

Pak matice T'_1 a T'_2 jsou definovány takto

$$T'_1 = \begin{bmatrix} \alpha_1 & 0 & \gamma_1 \\ 0 & \beta_1 & \delta_1 \\ 0 & 0 & 1 \end{bmatrix} \qquad T'_2 = \begin{bmatrix} \alpha_2 & 0 & \gamma_2 \\ 0 & \beta_2 & \delta_2 \\ 0 & 0 & 1 \end{bmatrix}$$

kde $\alpha_1 = 1 / (Wx_b - Wx_a)$ $\beta_1 = 1 / (Wy_b - Wy_a)$

$\gamma_1 = - \alpha_1 Wx_a$ $\delta_1 = - \beta_1 Wy_a$

a

$\alpha_2 = Vx_b - Vx_a$ $\beta_2 = Vy_b - Vy_a$

$\gamma_2 = Vx_a$ $\delta_2 = Vy_a$

Je zřejmé, že matice T'_2 je určena **pouze** vlastnostmi výstupního zařízení. Toto rozdělení umožňuje i současný výstup na více rozdílných zařízeních.

4.7 Transformace křivek a ploch druhého stupně

Dosud uvedené geometrické transformace byly odvozeny pro případ operací v E_2, resp. v E_3 , s body a úsečkami. Pro reálné použití křivek (kuželoseček, tj. kružnic, elips apod.) ležících na libovolné rovině v prostoru a ploch (elipsoidů, paraboloidů, válcových a kuželových ploch) druhého stupně, jako grafických primitiv, je nezbytné odvodit analogické transformace k transformacím souřadnic bodů.

Uvažme obecnou kuželosečku ve tvaru

$$\mathbf{x}^T \cdot \mathbf{A} \cdot \mathbf{x} = 0$$

rozepsáním pak dostáváme (předpokládáme symetrickou matici **A**):

$$a_{11}x^2 + a_{22}y^2 + a_{33}z^2 + 2a_{12}xy + 2a_{13}yz + 2a_{23}yz + 2a_{14}x + 2a_{24}y + 2a_{34}z + a_{44} = 0$$

Pak operace posunutí je definována:

$$\mathbf{z} = \mathbf{T} \cdot \mathbf{x} \qquad \mathbf{x} = \mathbf{T}^{-1} \cdot \mathbf{z}$$

Lze tedy psát:

$$\mathbf{z}^T \cdot \left(\mathbf{T}^{-1} \right)^T \cdot \mathbf{A} \cdot \mathbf{T}^{-1} \cdot \mathbf{z} = 0$$

Pak

$$\mathbf{z}^T \cdot \mathbf{B} \cdot \mathbf{z} = 0$$

kde: $\mathbf{B} = \left(\mathbf{T}^{-1} \right)^{T} . \mathbf{A} . \mathbf{T}^{-1}$ je matice reprezentující danou křivku, resp. plochu, druhého stupně po posunutí.

Je pochopitelně nutné též např. v případě eliptického oblouku transformovat i koncové body pomocí matice $\mathbf{T}^{-1}$. Podobným způsobem lze odvodit i ostatní geometrické transformace.

Pro případ rotace dostáváme

$$\mathbf{z} = \mathbf{R} . \mathbf{x}$$

$$\mathbf{z}^{T} . \left(\mathbf{R}^{-1} \right)^{T} . \mathbf{A} . \mathbf{R}^{-1} . \mathbf{z} = 0$$

V případě **rotace** je matice **R** ortogonální, tj. platí $\mathbf{R}^{-1} = \mathbf{R}^{T}$, a tedy lze psát:

$$\mathbf{z}^{T} . \mathbf{R} . \mathbf{A} . \mathbf{R}^{-1} . \mathbf{z} = 0$$

4.8 Akcelerátory

Při realizaci geometrických transformací se souřadnicemi bodů je vhodné nejdříve určit celkovou matici transformace a poté ji použít pro výpočet transformovaných souřadnic jednotlivých bodů. Pro urychlení výpočtů jsou používány tzv. akcelerátory, které realizují maticové operace hardwarově. Nejjednodušším případem je pak realizace při použití dvou procesorů s pohyblivou řádovou čárkou, viz obr.4.8.1.

Reprezentuje-li matice **R** celkovou transformaci, která má být aplikována na vektor **x**, a označíme-li **x'** výsledný transformovaný vektor, tj. platí:

$$\mathbf{x}' = \mathbf{R} . \mathbf{x}$$

Akcelerátor pak pronásobí postupně odpovídající složky vektoru **x** a řádky matice **R** tak, že dostáváme:

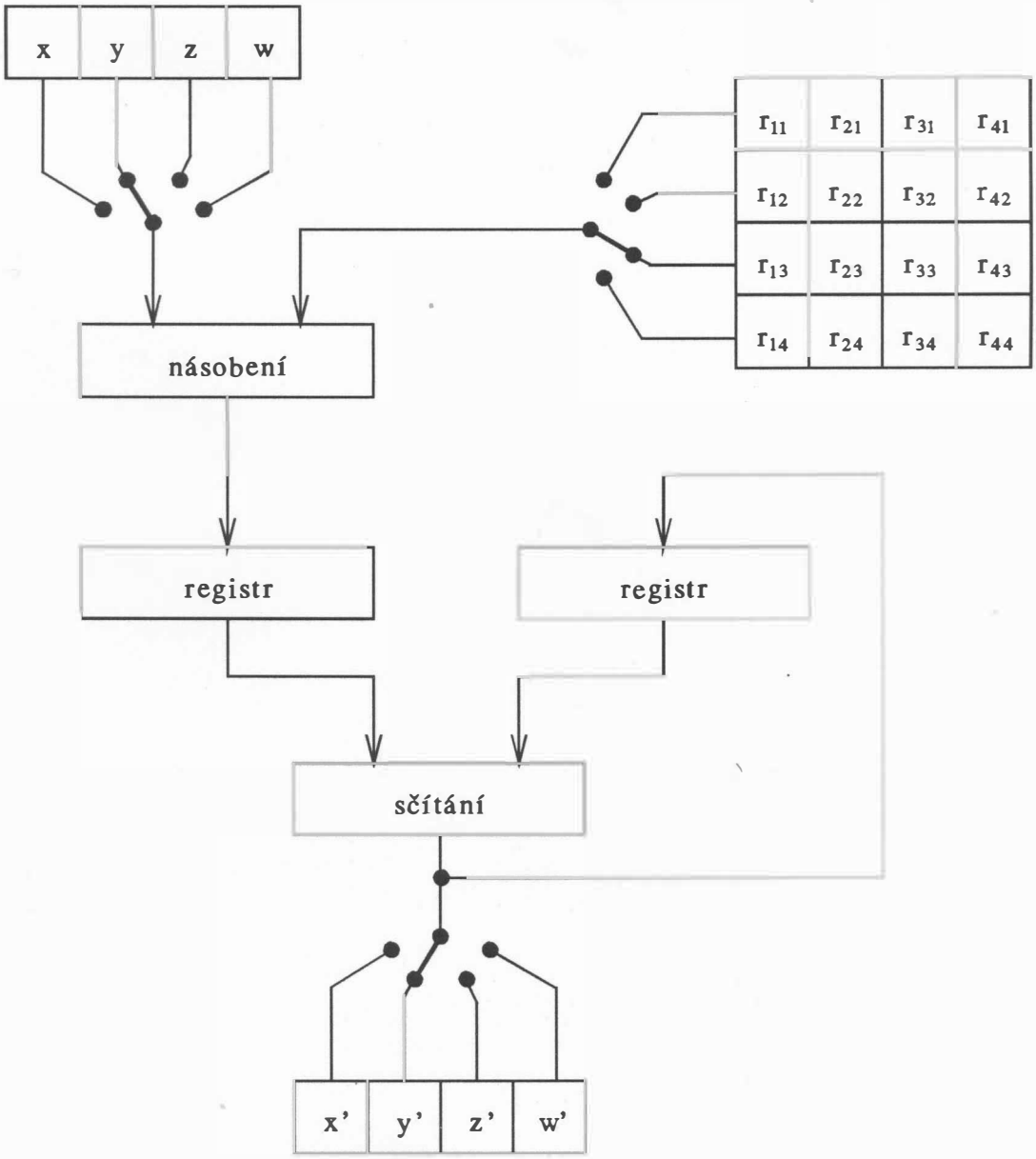
$$x' = r_{11}\, x + r_{12}\, y + r_{13}\, z + r_{14}\, w$$

$$y' = r_{21}\, x + r_{22}\, y + r_{13}\, z + r_{24}\, w$$

$$z' = r_{31}\, x + r_{32}\, y + r_{33}\, z + r_{34}\, w$$

$$w' = r_{41}\, x + r_{42}\, y + r_{43}\, z + r_{44}\, w$$

Pak na výpočet jedné složky vektoru **x'** se spotřebuje 7 kroků, tj. operací sčítání nebo násobení. Ke zrychlení lze použít možného paralelismu operací násobení a sčítání.

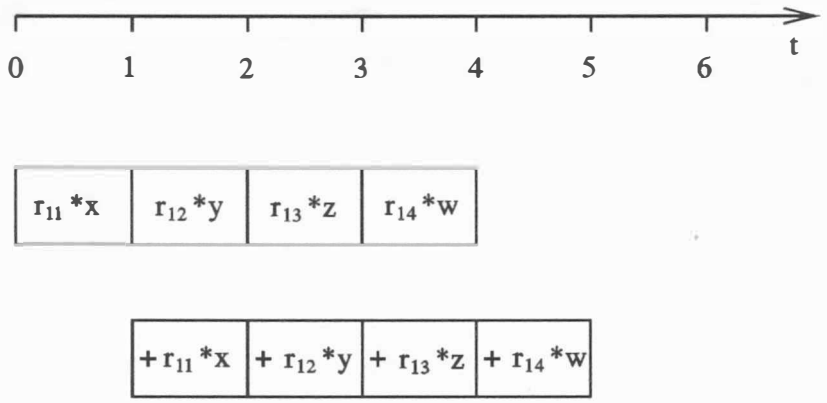


Obr. 4.8.1

Uvážíme-li např. operaci výpočtu hodnoty x', kde

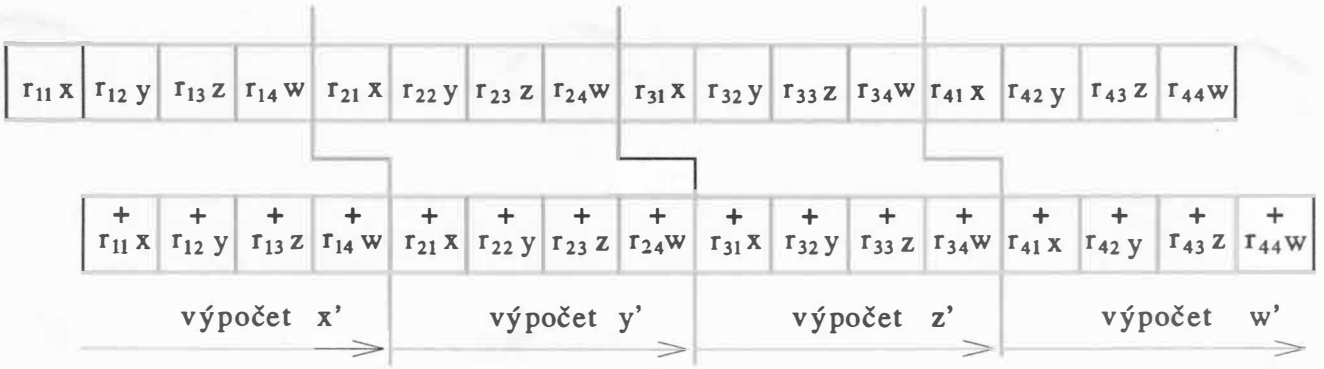
$$x' = r_{11}\,x + r_{12}\,y + r_{13}\,z + r_{14}\,w$$

pak lze ukázat, že operace násobení a sčítání mohou být v jednotlivých časových okamžicích prováděny současně, viz obr. 4.8.2.



Obr. 4.8.2

Uvedeným způsobem se za 5 časových jednotek provede 7 FLOPS (Floating Point Operations - operací v pohyblivé řádové čárce). Lze tedy urychlit provádění operací asi o 40 %. K dalšímu zrychlení lze použít zřetězení jednotlivých operací (pipelining). Použijeme-li výše uvedené schéma výpočtu pro jednotlivé dimenze, pak lze ukázat, že je zapotřebí N * 7 FLOPS na N + 1 časových jednotek, je-li N dimenze maticových operací, tj. v našem případě N = 4, viz obr. 4.8.3.

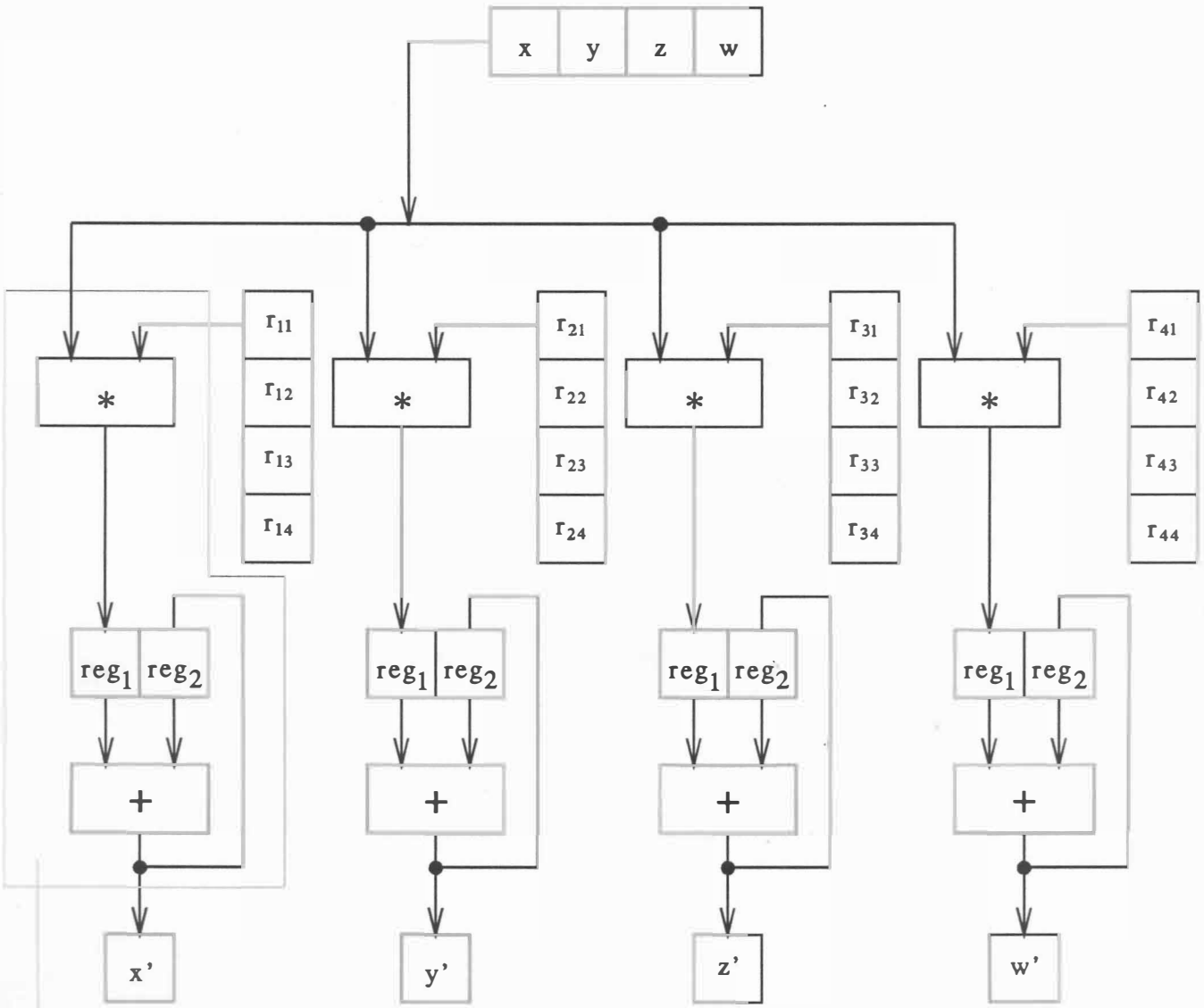


Obr. 4.8.3

Použití zřetězení vede asi k 75 % urychlení. Pro výkonné pracovní stanice však ani toto zrychlení není postačující, a proto jsou používány tzv. vektor - vektor procesory, viz obr. 4.8.4, kdy se vektory $\mathbf{r}_i^T$, které odpovídají řádkům matice **R**, pronásobí transformovaným vektorem **x** tak, aby byla vypočtena příslušná složka vektoru **x'**. Pak i-tou složku vektoru **x'**,

označenou x'_i, lze vyjádřit pomocí skalárního součinu:

$$x'_i = r_i^T \cdot x$$



transformace proběhne v 5-ti časových intervalech

Obr. 4.8.4

Je zřejmé, že celý procesor obsahuje vlastně čtyři samostatné celky realizující skalární součin. Lze ukázat, že při použití těchto procesorů je možné urychlit provádění operací až o 95 %.

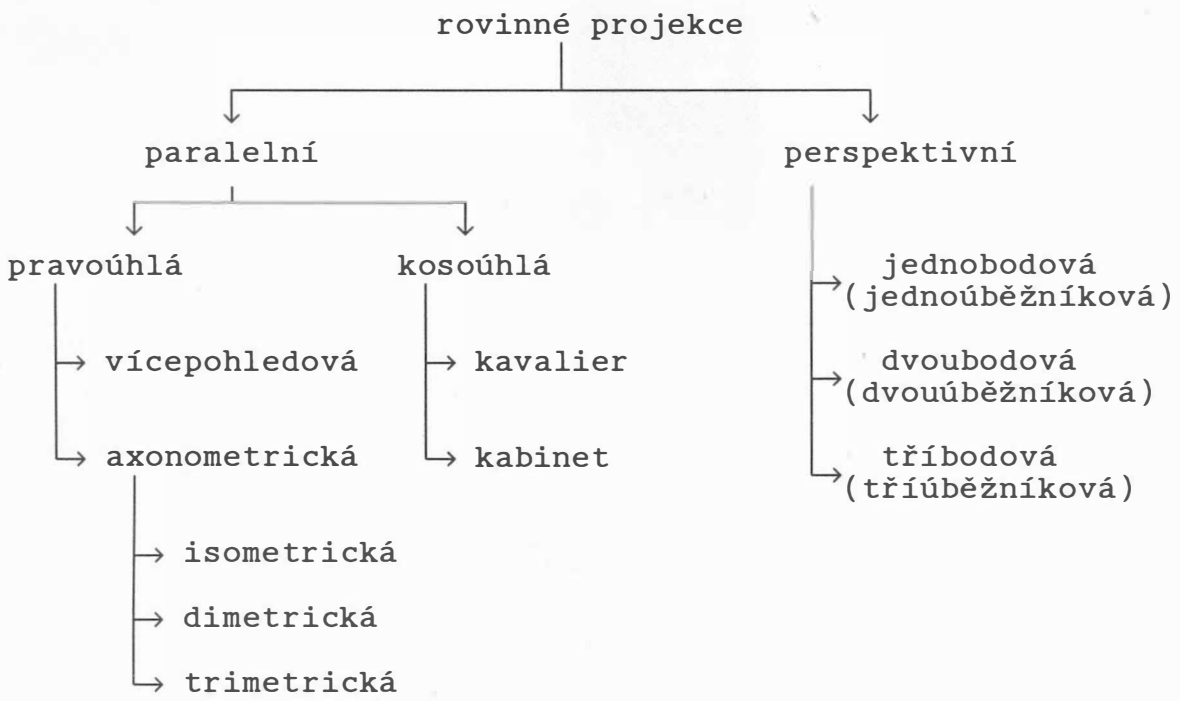
Příklady

Navrhněte strukturu akcelerátoru pro práci s křivkami a plochami druhého stupně a jejich částí (např. kruhový oblouk apod.).

5. Projekce

5.1 Možnosti zobrazování třírozměrného prostoru

Zobrazování třírozměrného prostoru je nepoměrně složitější než zobrazování prostoru dvourozměrného. V dvourozměrném případě obvykle definujeme transformaci okno-pohled, která je založena na principu, že část objektu se zobrazí na danou plochu, přičemž nežádané části se odříznou. Koncepčně je objekt odříznut vůči oknu a zbytek kresby je poté zobrazen. Vzhledem k tomu, že většina dostupných zařízení umožňuje zobrazovat pouze v dvourozměrném souřadném systému, musí se zavést transformace, která by transformovala třírozměrný prostor do prostoru dvourozměrného. Tento problém může být řešen pomocí projekce (též promítání), která transformuje třírozměrné objekty na dvourozměrnou projekční rovinu. Obecně je projekce transformací, která n-rozměrný prostor transformuje do k-rozměrného prostoru, přičemž n > k. Dále se omezíme pouze na projekce planární, tj. na promítání na rovinnou plochu. Z obr.5.1.1 vyplývá rozdělení na jednotlivé transformace:



Obr.5.1.1

Paralelní projekce je vlastně zvláštním případem projekce perspektivní, je-li pozorovatel v nekonečnu.

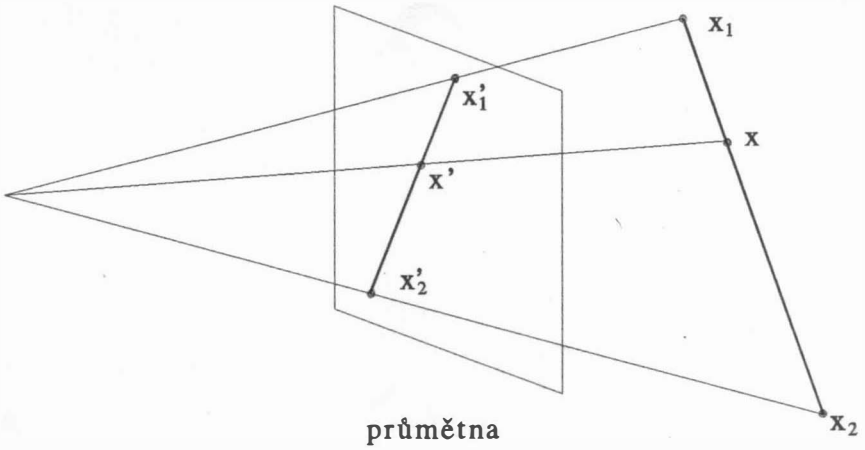
Perspektivní projekce

V případě perspektivní projekce vytvářejí rovnoběžné čáry, které nejsou kolmé k rovině promítání, na průmětně úběžník. Úběžník si lze představit jako zobrazení bodu v nekonečnu. Jednotlivé transformace jsou kategorizovány podle počtu úběžníků. Je zřejmé, že lze docílit maximálně tříbodové perspektivy, tj. perspektivní transformace vytvářející tři úběžníky. Z obr.5.1.4 je zřejmé, že jde o třídu transformací, která se často používá. V případě perspektivní projekce, viz obr.5.1.2, je nutné si uvědomit, že platí:

$$\mathbf{x} = \mathbf{x}_1 + (\mathbf{x}_2 - \mathbf{x}_1) \, . \, \Phi \qquad \Phi \in \langle 0 , 1 \rangle$$

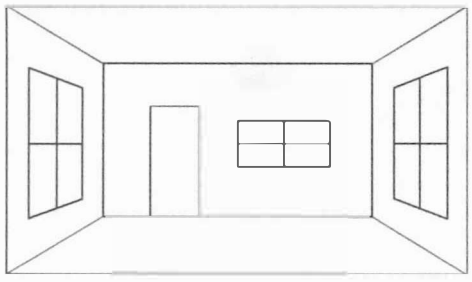
$$\mathbf{x}' = \mathbf{x}'_1 + (\mathbf{x}'_2 - \mathbf{x}'_1) \, . \, \lambda \qquad \lambda \in \langle 0 , 1 \rangle$$

přičemž není pravda, že: $\lambda = \Phi$

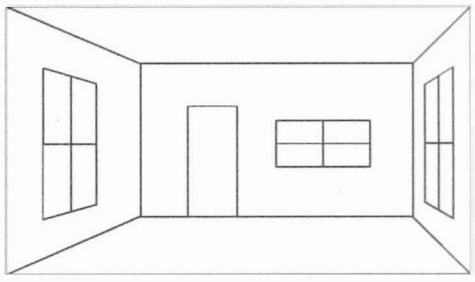


průmětna

Obr. 5. 1. 2



jednoúběžníková perspektiva

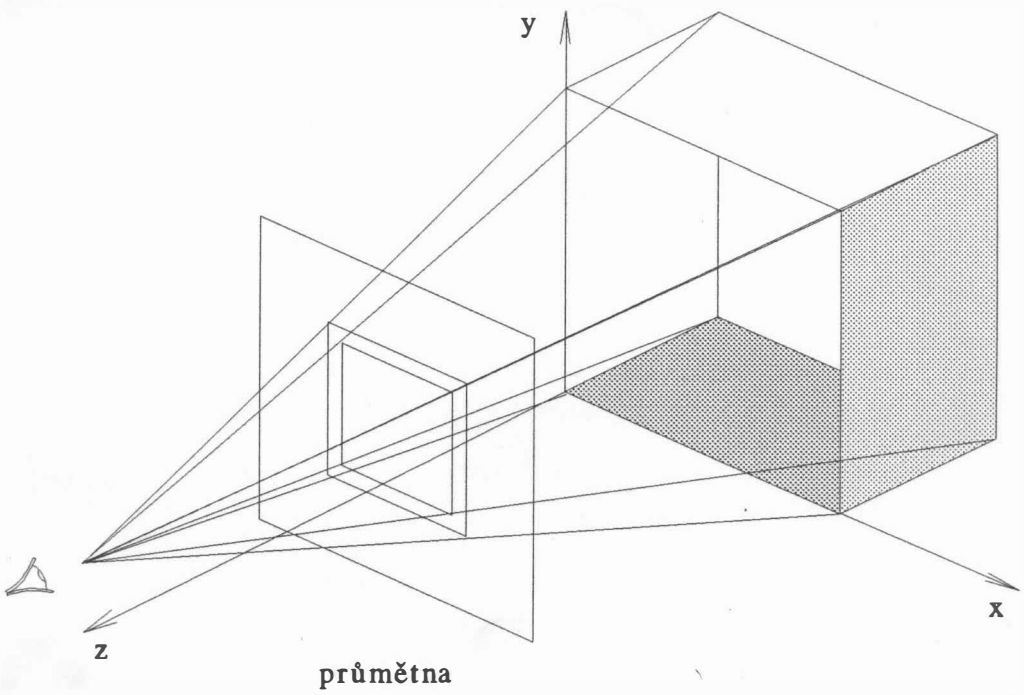


jednoúběžníková perspektiva

Obr. 5. 1. 3

Jednobodová perspektiva je velmi často využívána zejména v oblasti návrhu zařízení interiérů, viz obr.5.1.3.

Při konstrukci jednobodové perspektivy se vychází z toho, že průmětna je rovnoběžná s některou ze základních rovin souřadného systému, tj. protíná pouze jednu osu souřadného systému.



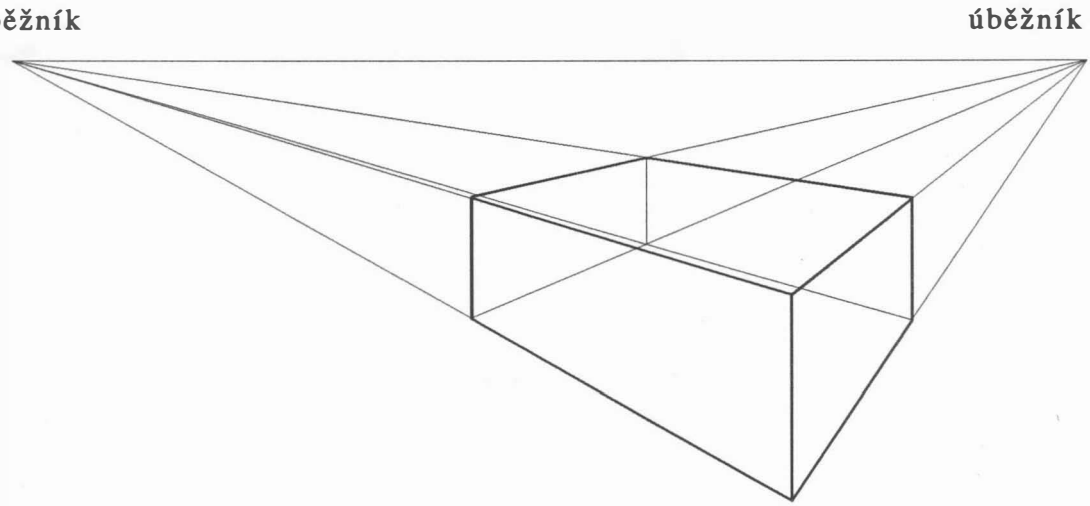
příklad konstrukce jednoúběžníkové prespektivy

Obr. 5.1.4

Při konstrukci dvoubodové perspektivy se vychází z toho, že průmětna je rovnoběžná s jednou osou, tj. protíná dvě osy souřadného systému.

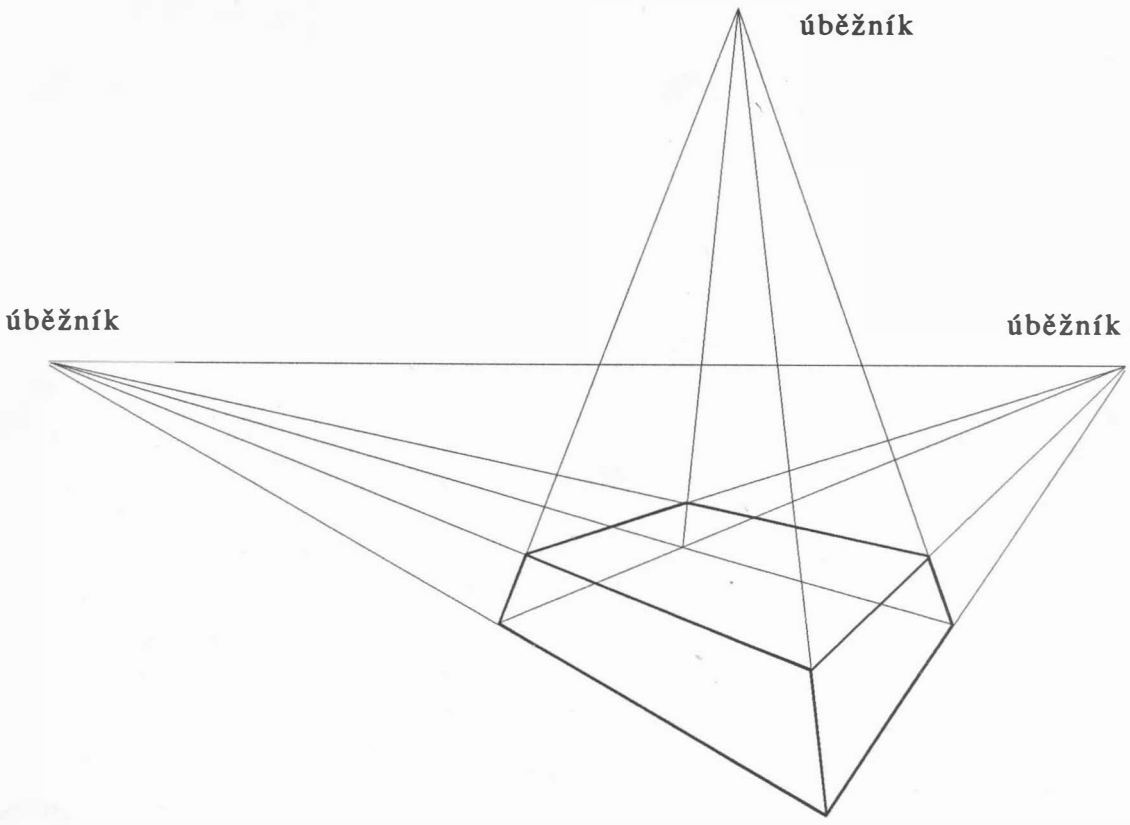
Pro tříbodovou perspektivu platí, že průmětna není kolmá na žádnou osu, a je tedy jimi protínána.

Dvoubodová perspektiva se velmi často používá zejména v architektuře, avšak tříbodová perspektiva se používá jen velmi zřídka.



Příklad konstrukce dvoúběžníkové perspektivy

Obr. 5.1.5

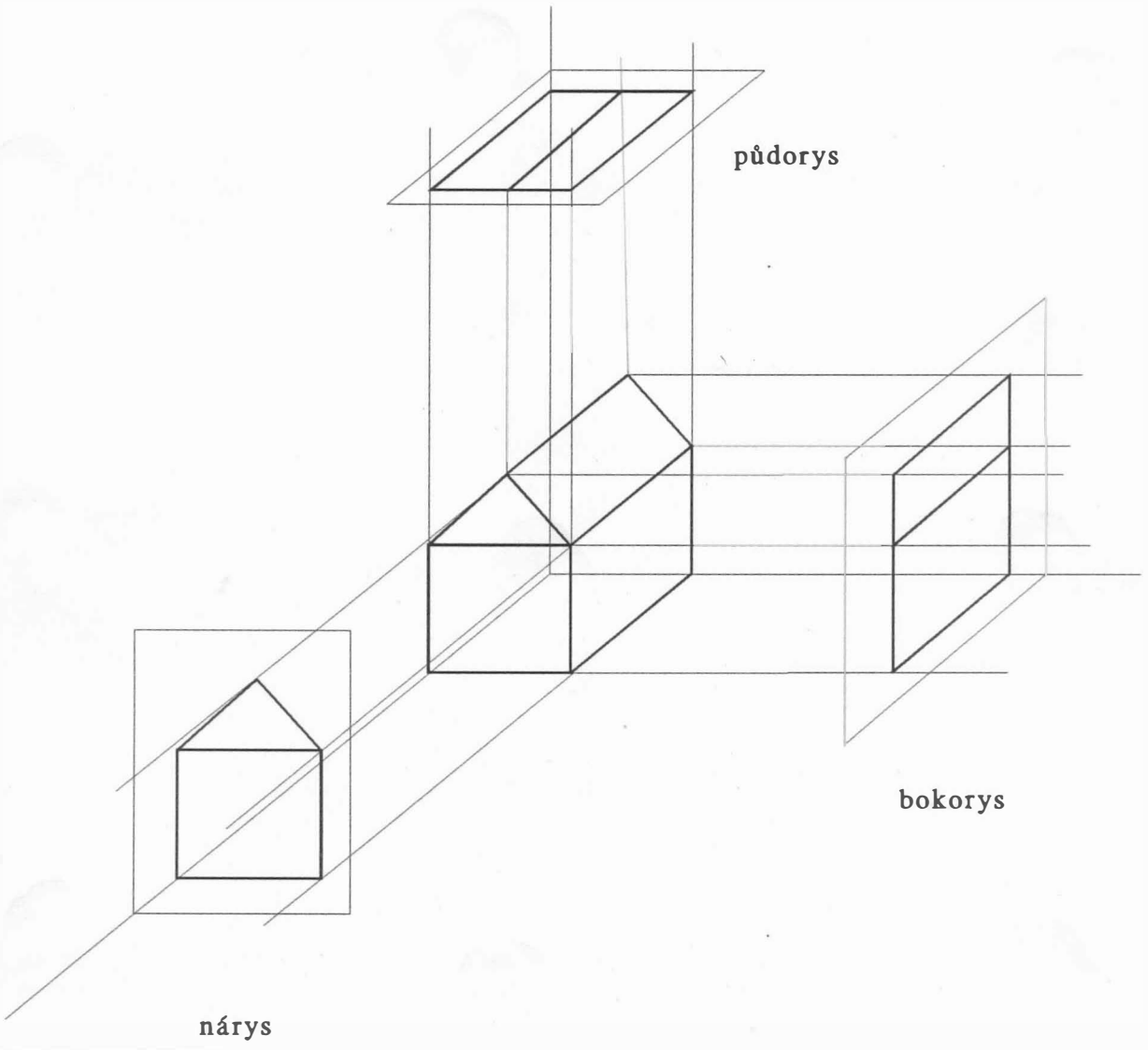


příklad tříúběžníkové perspektivy

Obr. 5.1.6

'aralelní projekce

Paralelní projekce lze rozdělit do dvou základních skupin, a to podle vztahu mezi směrem projekce a normálou roviny projekce. V pravoúhlé projekci jsou směry stejné, zatímco v případě kosoúhlé projekce nikoliv. Nejjednodušším typem pravoúhlé projekce je projekce vícepohledová, tj. nárys, bokorys, půdorys atd. Tento typ projekce se velmi často používá v technické praxi a jeho konstrukce je znázorněna na obr. 5.1.7.

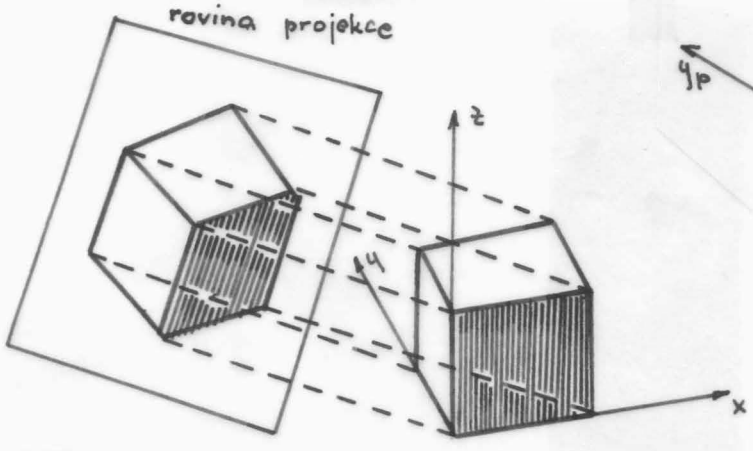


ukázka vícepohledové projekce

Obr. 5.1.7

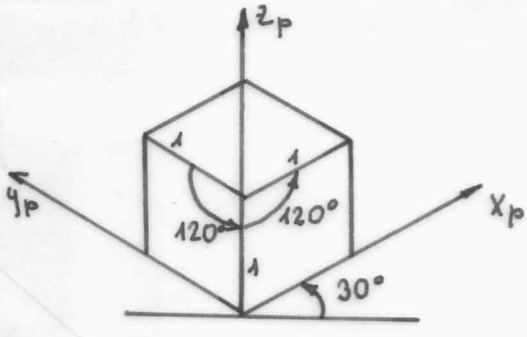
Axonometrické promítání

Axonometrická projekce je projekcí pravoúhlou. V axonometrické projekci jsou rovnoběžné čáry zkráceny stejným způsobem. Axonometrické projekce jsou rozděleny podle orientace projekční roviny, tj. podle úhlů mezi průmětnou a souřadnými osami. Jestliže všechny tři úhly jsou stejné, jde o izometrickou projekci. Jestliže jsou stejné pouze dva úhly, pak je výsledkem projekce dimetrická. Jestliže všechny úhly jsou různé, jde o projekci trimetrickou.



konstrukce izometrické projekce

Obr. 5.1.8



výsledek projekce

Obr. 5.1.9

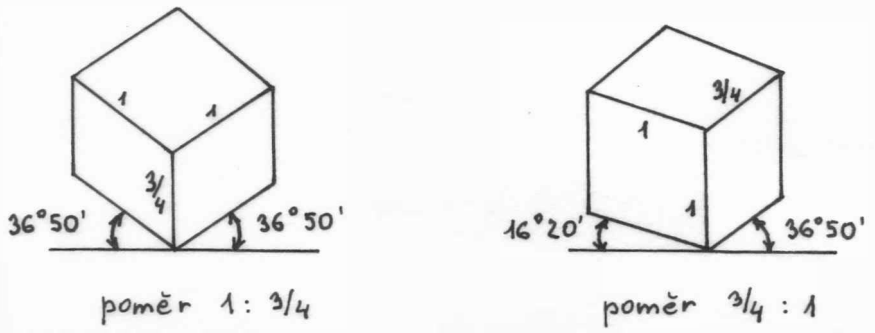
Výsledek izometrické projekce z obr. 5.1.8 je uveden na obr. 5.1.9. V případě izometrické projekce musejí být úhly, které svírá průmětna s jednotlivými souřadnými osami, stejné (osy x,y,z jsou promítnuté do x_p, y_p, z_p). Zkrácení je na všech osách stejné. To znamená, že normála průmětna musí být rovnoběžná s jednou ze čtyř přímek:

$x = y \ \& \ x = z$	nebo	$x = y \ \& \ x = -z$
$x = -y \ \& \ x = z$	nebo	$x = -y \ \& \ x = -z$

V případě dimetrické projekce jsou pouze dvě souřadné osy shodně zkráceny, a tedy pouze dva úhly mezi rovinou projekce a souřadnými osami jsou shodné. To znamená, že normála projekční roviny musí být rovnoběžná s jednou ze šesti rovin, a to:

$x = \pm y$ nebo $x = \pm z$ nebo $y = \pm z$

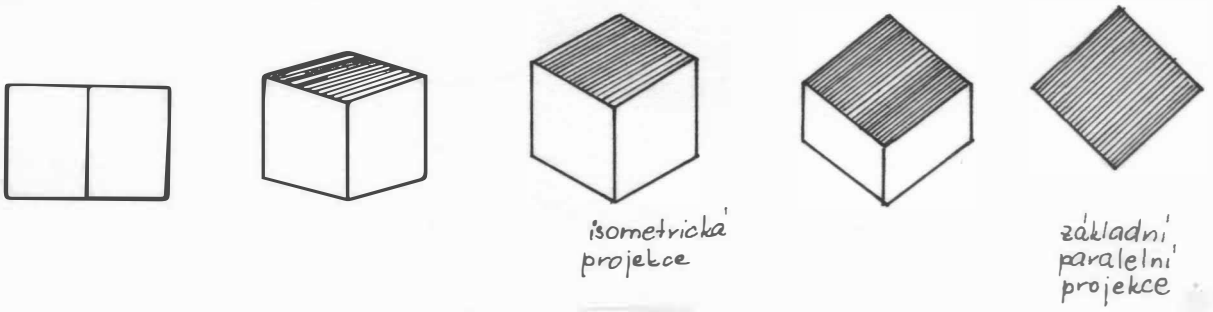
Změnou normály projekční plochy v jedné z těchto rovin může být změněno zvýraznění jednotlivých ploch.



dimetrická projekce krychle

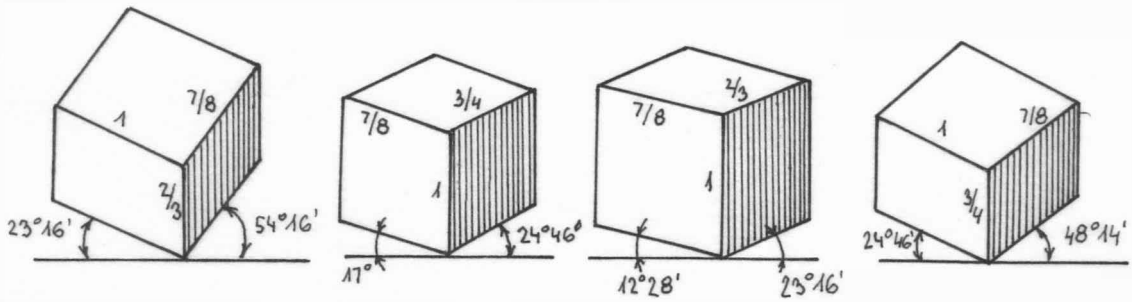
Obr. 5.1.10

Posloupnost různých dimetrických projekcí je znázorněna na obr. 5.1.11.



Obr. 5.1.11

Trimetrická projekce je obecnější. Výsledkem je různé zkrácení na jednotlivých osách, přičemž žádný úhel mezi osami a projekční rovinou není stejný. Na obr. 5.1.12 je uvedeno několik trimetrických projekcí krychle.



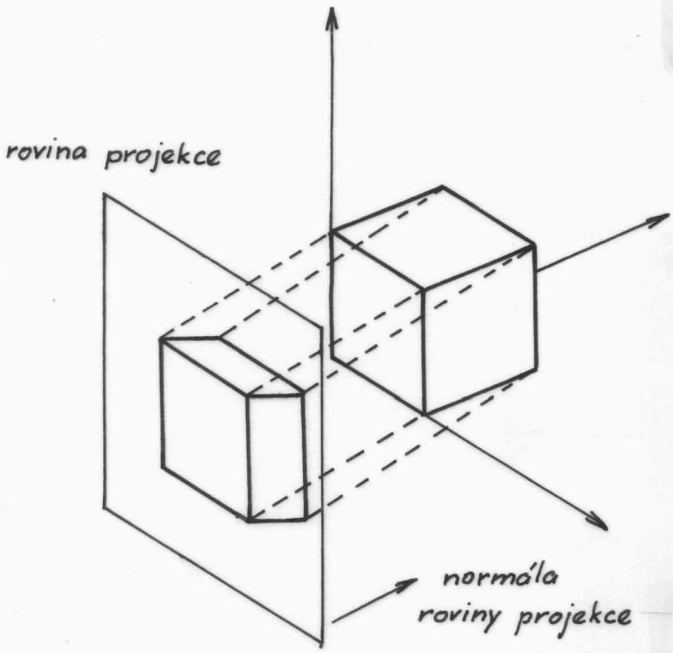
Obr. 5.1.12

Kosoúhlé projekce

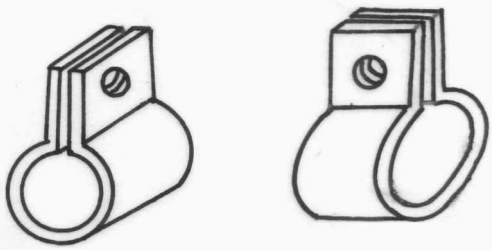
Kosoúhlé projekce, viz obr. 5.1.13, kombinují vlastnosti vícepohledové paralelní projekce s projekcemi axonometrickými. Vícepohledová paralelní projekce zobrazuje přesný tvar ploch daného objektu, nicméně třírozměrný tvar může být někdy velmi obtížné si představit. V axonometrické projekci dostáváme představu tvaru tělesa v třírozměrném prostoru, avšak jen zřídka obdržíme správný tvar tělesa. Odměřování v axonometrické projekci není obecně možné, zatímco ve vícepohledové paralelní projekci je běžné. Kosoúhlé projekce obvykle reprezentují skutečný tvar ploch třírozměrného tělesa, přičemž dobře zachycují i celkový vjem v třírozměrném prostoru.

Poznamenejme, že směr normály roviny projekce nemusí být kolmý se směrem projekce. Mezi nejpoužívanější kosoúhlé projekce patří projekce typu kavalier a kabinet. V obou případech se třetí osa promítne nejčastěji pod úhlem 30° nebo 45°. V případě projekce typu kavalier je osa zobrazena nezkráceně, zatímco při použití projekce typu kabinet se zkracuje v závislosti na použitém úhlu.

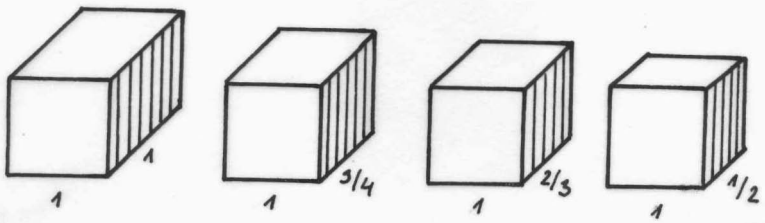
Na obr. 5.1.14 je ukázána kosoúhlá projekce krychle, kde míra zkrácení se mění podle úhlu mezi směrem projekce a třetí osou. Výběr směru projekce vzhledem ke srozumitelnosti kresby je důležitý zejména v případě existence zakřivených hran tělesa, viz obr. 5.1.15.



Obr. 5. 1. 13



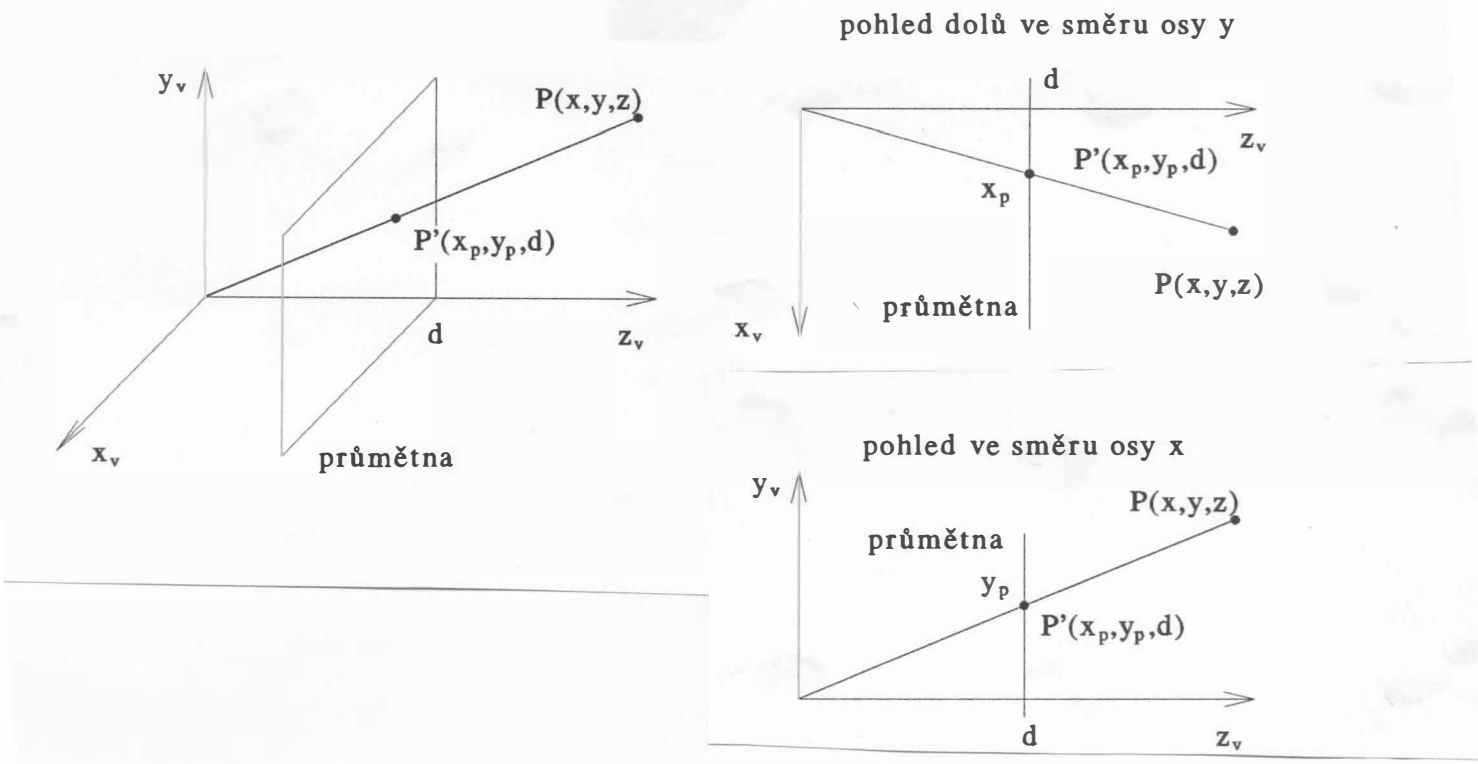
Obr. 5. 1. 15



Obr. 5. 1. 14

5.2 Matematický aparát rovinné projekce

Až dosud byly ukázány různé metody projekce z třírozměrného prostoru do prostoru dvourozměrného, a to pomocí planárních projekcí. Projekce na válcovou, kulovou či jinou plochu nejsou v technické praxi často používány, a proto též matematický aparát bude uveden pouze pro projekce planární. Pro jednoduchost se předpokládá, že v případě perspektivní projekce je rovina projekce kolmá k ose z ve vzdálenosti z=d a že v případě projekce paralelní je rovina projekce kolmá k ose z v počátku, tj. z = 0. Dále uvedená pravidla jsou odvozena pro souřadný systém pozorovatele, který je levotočivý a je přirozený vzhledem k předpokládané orientaci os na stínítku obrazovky, tj. osa x je orientována doprava, osa y vzhůru a osa z dovnitř obrazovky.



Obr. 5.2.1

Každá rovinná projekce může být reprezentována při použití homogenních souřadnic pomocí matice 4 x 4, což je výhodné vzhledem ke stejné reprezentaci transformačních operací, tj. operace transformace a projekce může být reprezentována pomocí jedné matice.

Pro perspektivní projekci, je-li průmětna ve vzdálenosti z = d v souřadném systému pozorovatele, platí:

$$\frac{x_p}{d} = \frac{x}{z} \qquad \frac{y_p}{d} = \frac{y}{z}$$

Jednoduchou úpravou dostáváme

$$x_p = \frac{x}{z}\, d = \frac{x}{z/d} \qquad y_p = \frac{y}{z}\, d = \frac{y}{z/d}$$

Vzdálenost d je v podstatě vlastně měřítkovým faktorem, který je aplikován na souřadnice x_p a y_p. Dělení hodnotou souřadnice z způsobí, že objekty vzdálenější se zobrazí menší než objekty v popředí. Poznamenejme, že hodnoty z musejí být různé od nuly a obecně bod může být za, resp. před průmětnou, a tedy i za pozorovatelem (body za pozorovatelem je nutné vhodně odříznout pomocí postupu, který se nazývá ořezávání).

Uvedená perspektivní transformace může být vyjádřena maticí:

$$\mathbf{M}_{per} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix}$$

Někdy je vhodnější tvar:

$$\mathbf{M}_{per} = \begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & d & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Vynásobíme-li matici projekce $\mathbf{M}_{per}$ souřadnicemi bodu P, tj.

$$\mathbf{p}' = \mathbf{M}_{per} \cdot \mathbf{p}$$

kde $\mathbf{p} = [x,y,z,1]^T$, pak dostaneme bod P', který je reprezentován vektorem $\mathbf{p}' = [X,Y,Z,W]^T$ v homogenních souřadnicích, kde

$$[\ X\ ,\ Y\ ,\ Z\ ,\ W\ ]^T = [\ x\ ,\ y\ ,\ z\ ,\ z/d\ ]^T$$

a po vydělení hodnotou W (což je z/d) dostáváme zpět v třírozměrném prostoru

$$[\ X/W\ ,\ Y/W\ ,\ Z/W\ ,\ 1\ ]^T = [\ x_p\ ,\ y_p\ ,\ z_p\ ,\ 1\ ]^T$$

$$= [\ x/(z/d)\ ,\ y/(z/d)\ ,\ d\ ,\ 1\ ]^T$$

což je výsledek očekávaný včetně změny hodnoty souřadnice z. V literatuře se vyskytuje ještě alternativní definice perspektivní projekce, a to s rovinou projekce v bodě z = 0 a středem promítání v bodě z = -d (viz obr.5.2.2). Pak lze snadno analogicky odvodit vztah pro transformaci

$$\frac{x_p}{d} = \frac{x}{z + d} \qquad \frac{y_p}{d} = \frac{y}{z + d}$$

jednoduchou úpravou pak dostáváme:

$$x_p = \frac{x}{z + d} = \frac{x}{(z/d)+1}$$

$$y_p = \frac{y}{z + d} = \frac{y}{(z/d)+1}$$

Matice transformace pak pro tento případ má tvar:

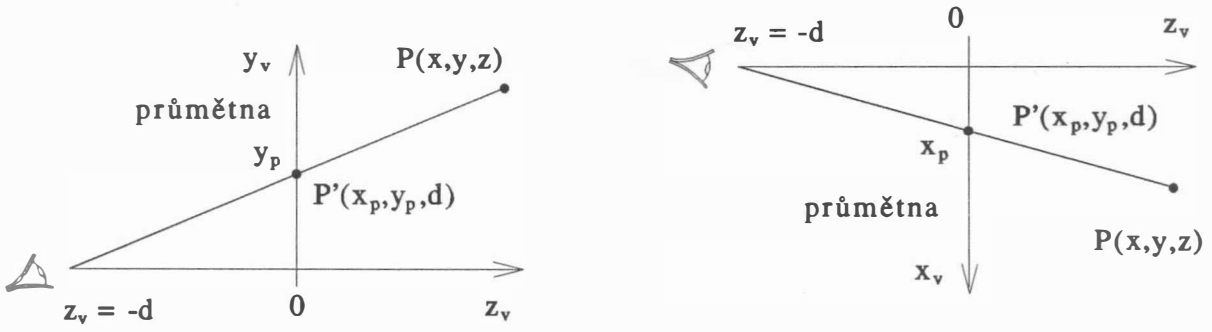
$$\mathbf{M}'_{per} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1/d & 1 \end{bmatrix}$$

Bod P reprezentovaný vektorem $\mathbf{p} = [x,y,z,1]^T$ je transformován na bod P' reprezentovaný vektorem $\mathbf{p}' = [X,Y,Z,W]^T$, kde:

$$[\ X\ ,\ Y\ ,\ Z\ ,\ W\ ]^T = [\ x\ ,\ y\ ,\ 0\ ,\ z/d + 1]^T$$

$$= [\ x/(z/d + 1)\ ,\ y/(z/d + 1)\ ,\ 0\ ,\ 1\ ]^T$$

což je výsledek opět očekávaný.



alternativní definice perspektivní projekce

Obr. 5.2.2

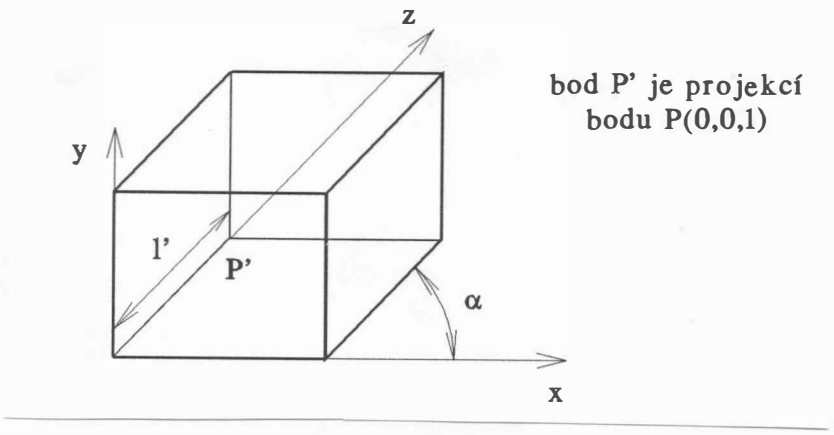
Paralelní projekce na průmětnu z = 0 je jednoduchá, neboť souřadnice x, y se nemění a souřadnice z je rovna nule, tudíž:

$$\mathbf{M}_{par} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Bod P určený vektorem $\mathbf{p} = [x,y,z,1]^T$ se promítne do bodu P', který je reprezentován vektorem $\mathbf{p}' = [x,y,0,1]^T$, přičemž platí, že:

$$x = x_p \qquad \text{a} \qquad y = y_p$$

Uvažme nyní kosoúhlé promítání. Matice reprezentující projekci musí být zapsána pomocí výrazů závisejících na parametrech α a l, jak je ukázáno na obr.5.2.3, kde jednotková krychle je promítnuta do roviny xy.

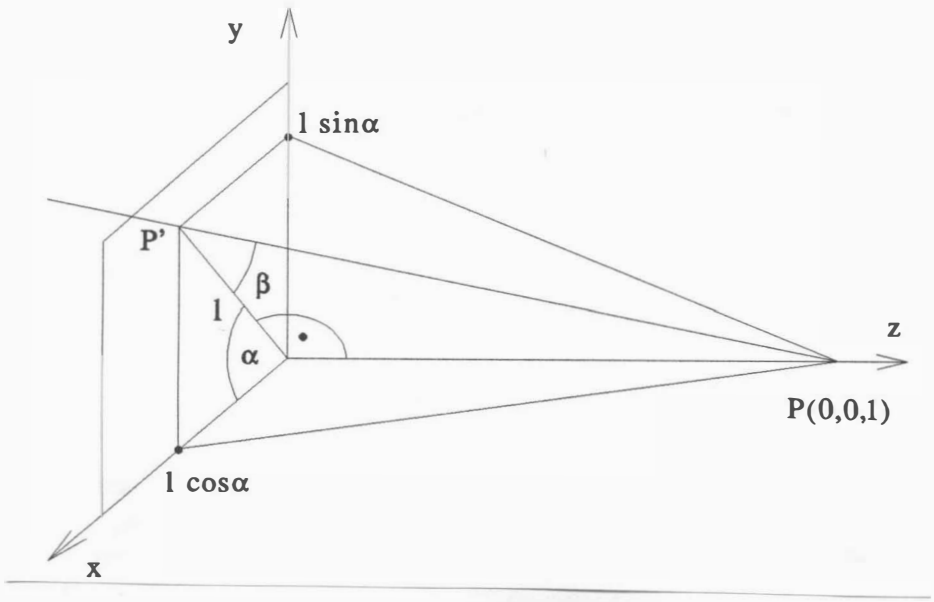


Obr.5.2.3

Je zřejmé, že bod P = (0,0,1) se promítne do bodu P' = (l.cosα,l.sinα,0) v rovině xy. Pak zřejmě přímka PP' procházející body P a P' definuje úhel projekce, viz obr.5.2.4.

Obecně bod (x,y,z) se při kosoúhlé projekci promítne do bodu v rovině xy se souřadnicemi (x_p,y_p). Pro rovnici přímky v rovině platí:

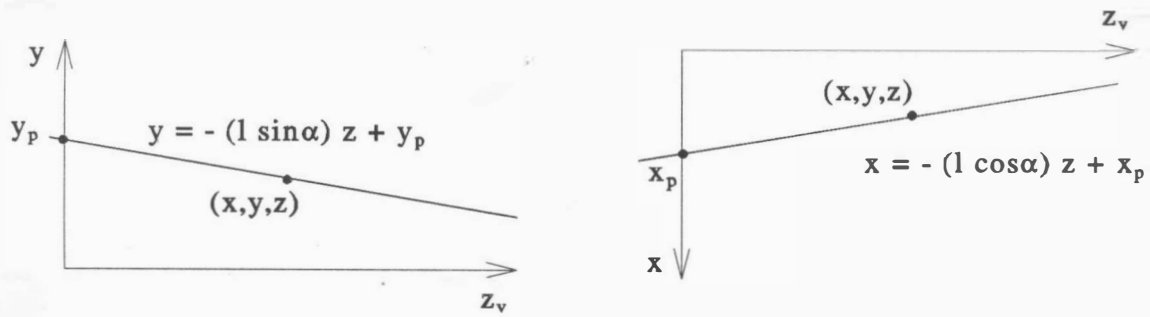
$$y = k \,.\, x + q$$



Obr. 5.2.4

Z rovnice pro x a y dostaneme (viz obr. 5.2.5):

$x_p = x + z.l.\cos\alpha$ $\qquad$ $y_p = y + z.l.\sin\alpha$

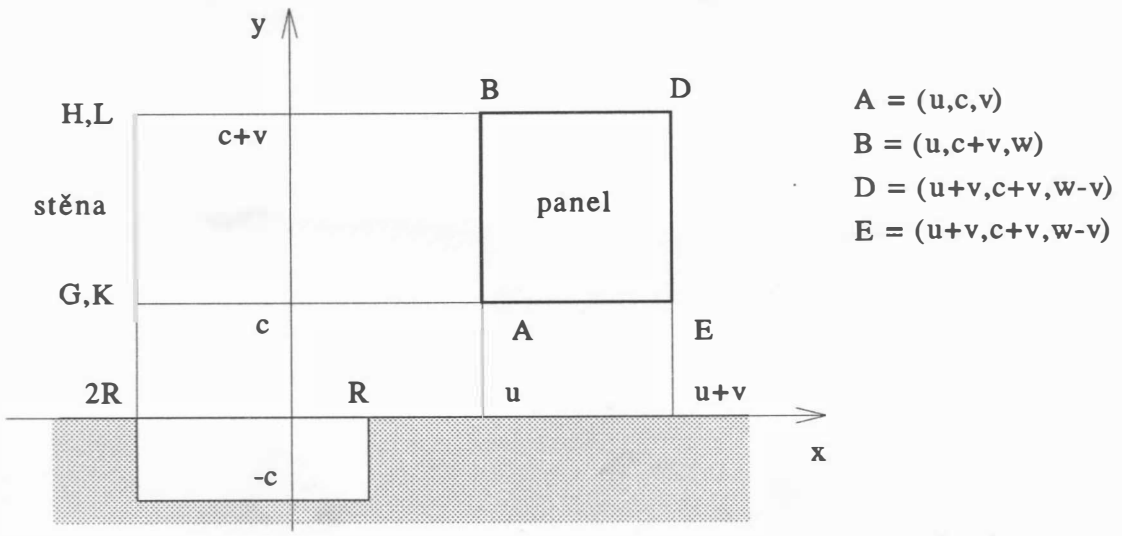


Obr. 5.2.5

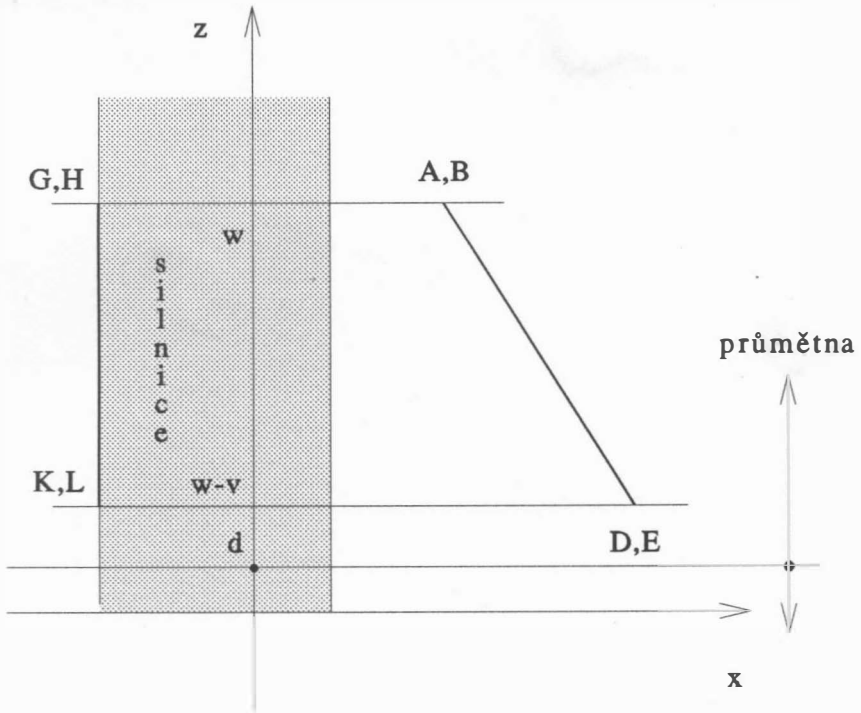
Pak matice reprezentující kosoúhlou projekci má tvar:

$$M_{kos} = \begin{bmatrix} 1 & 0 & l.\cos\alpha & 0 \\ 0 & 1 & l.\sin\alpha & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Pro projekci typu kavalier je l = 1 a úhel $\beta = 45^{\circ}$. Pro projekci typu kabinet pro l = 1/2 je úhel β = arctg(2), tj. $\beta = 63.4^{\circ}$. Pro paralelní projekci je l = 0 a $\alpha = 90^{\circ}$, takže se redukuje na M_{par}.



Obr. 5.2.6

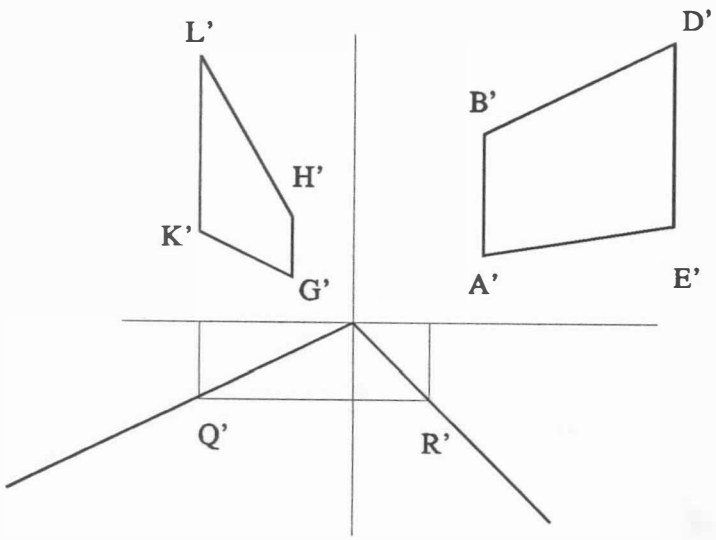


Obr. 5.2.7

Příklad

Vytvořte program simulující perspektivní pohled řidiče jedoucího podél rovné silnice. Pro jednoduchost umístíme řidiče do počátku souřadného systému a tvar silnice je na obr.5.2.6 - 5.2.7. Výsledný pohled je pak na obr.5.2.8.

Určete hodnoty souřadnic pro různé hodnoty R,u,v,c,(z-v),d a výsledný obrázek vykreslete.

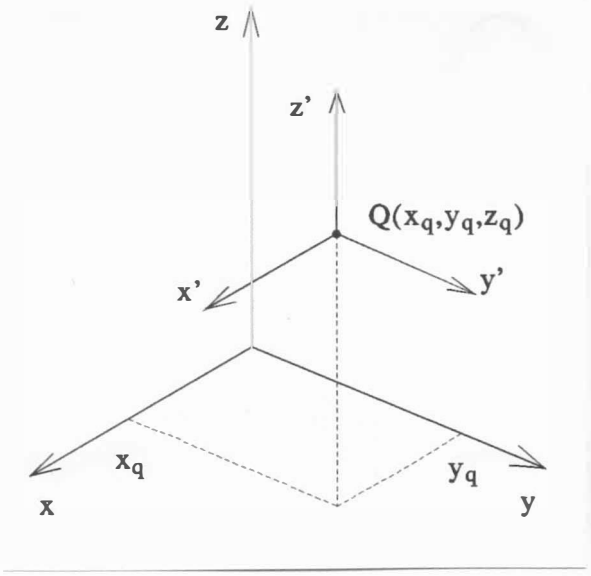


Výsledný pohled řidiče

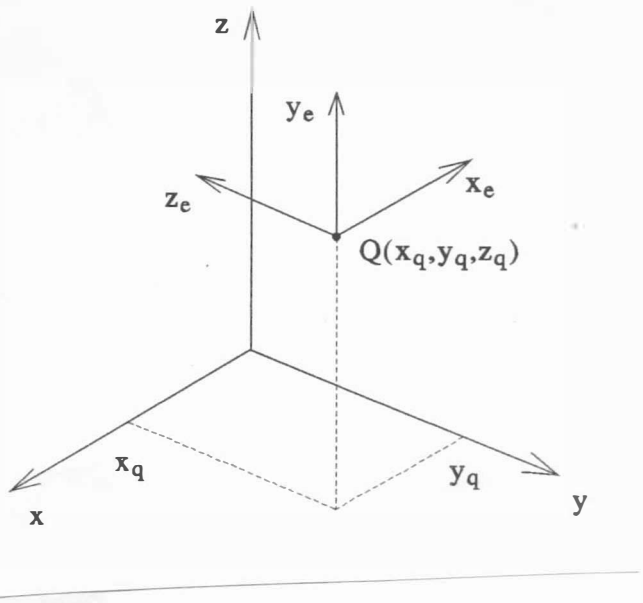
Obr.5.2.8

5.3 Perspektivní pohled na scénu

V mnoha aplikacích je žádoucí mít možnost perspektivního pohledu na scénu z určitého zadaného bodu Q, ve kterém se nachází pozorovatel, viz obr. 5.3.1.



Obr. 5.3.1.



Obr. 5.3.2

Budeme-li pozorovat tuto scénu z bodu $Q(x_q, y_q, z_q)$, pak bude pohledová osa z_e směřovat do počátku souřadného systému, viz obr. 5.3.4. Nicméně je stále ještě jeden stupeň volnosti při specifikaci transformací, neboť lze libovolně určit pootočení okolo osy z_e. Položme tedy dodatečný požadavek, aby osa z_e ležela v rovině rovnoběžné s původní rovinou x = y, tj. z = 0.

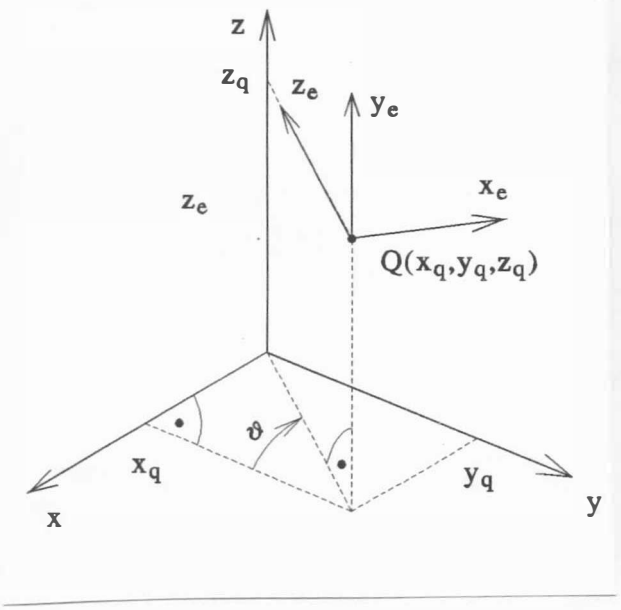
Pohledová transformace může být rozložena na následující kroky takto:

- posuv všech bodů ve scéně tak, aby bod Q byl ztotožněn s počátkem, viz obr. 5.3.1. Operace posuvu je reprezentována maticí **T**

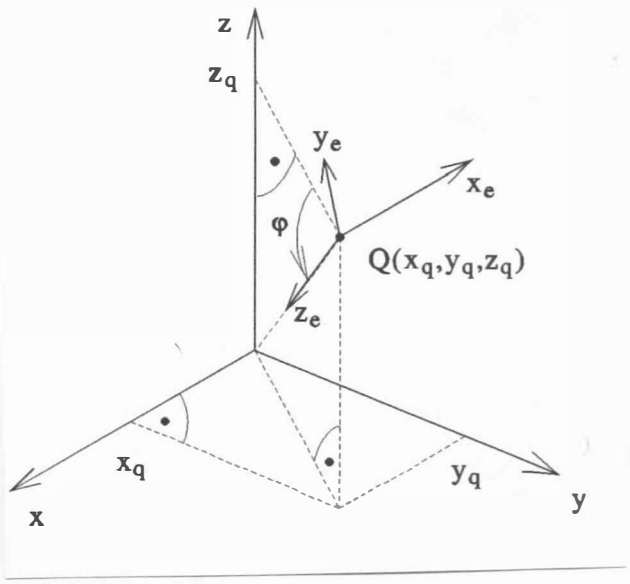
$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & -x_q \\ 0 & 1 & 0 & -y_q \\ 0 & 0 & 1 & -z_q \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- změna orientace souřadného systému tak, aby byl **levotočivý**. Obecně by bylo možné změnit orientaci jedné osy, avšak vhodným přeuspořádáním, viz obr. 5.3.2, ušetříme operace rotace. Tato změna orientace může pak být definována pomocí matice **Z**

$$\mathbf{Z} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Obr. 5.3.3



Obr. 5.3.4

- rotace okolo osy y_e o úhel ϑ tak, že osa z_e protíná osu z původního souřadného systému, viz obr. 5.3.3. Tato operace je definována maticí $\mathbf{R}_{x_e z_e}(\vartheta)$

$$\mathbf{R}_{x_e z_e}(\vartheta) = \begin{bmatrix} \cos\vartheta & 0 & -\sin\vartheta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\vartheta & 0 & \cos\vartheta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

kde: $\cos \vartheta = \frac{y_q}{\sqrt{x_q^2 + y_q^2}}$ $\sin \vartheta = \frac{x_q}{\sqrt{x_q^2 + y_q^2}}$

Zde je nezbytné upozornit na polohu znaménka minus, která je **zdánlivě** v rozporu s kap.4.3. Je nutné si uvědomit, že jde o rotaci v **levotočivém** souřadném systému, a tedy matice rotace musí být transformovány.

- rotace okolo osy x_e o úhel φ tak, aby osa z_e směřovala do počátku původního souřadného systému, viz obr.5.3.4. Operace rotace je určena maticí $\mathbf{R}_{z_e y_e}(\varphi)$

$$\mathbf{R}_{z_e y_e}(\varphi) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\varphi & \sin\varphi & 0 \\ 0 & -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

kde: $\cos \varphi = \frac{\sqrt{x_q^2 + y_q^2}}{\sqrt{x_q^2 + y_q^2 + z_q^2}}$ $\sin \varphi = \frac{z_q}{\sqrt{x_q^2 + y_q^2 + z_q^2}}$

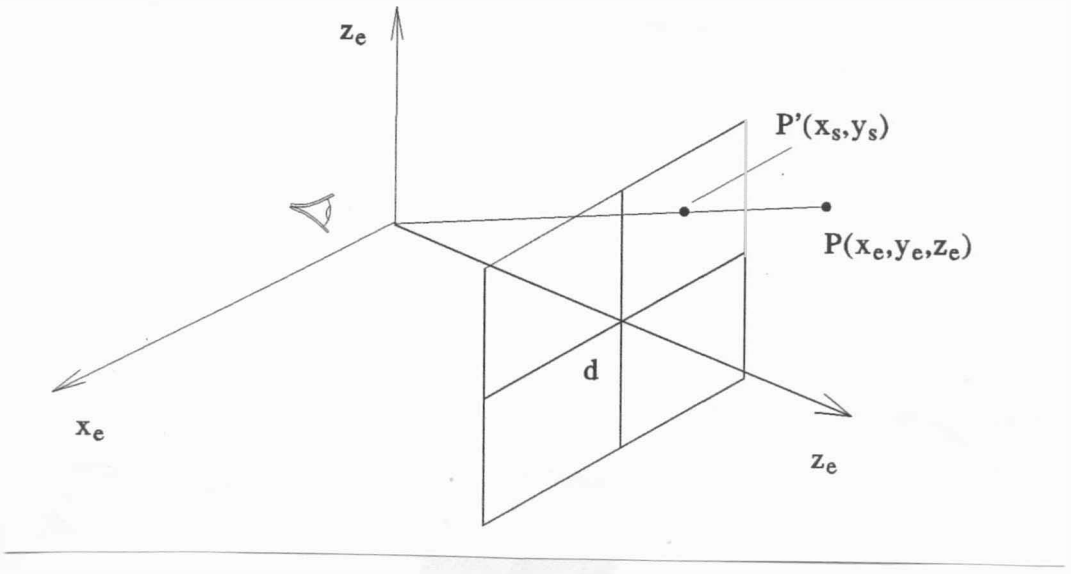
- perspektivní transformace dané scény, neboť je nezbytné použít transformaci respektující perspektivní pohled. Pro jednoduchost budeme předpokládat, že nebude docházet k nežádoucím efektům a že **všechny** body zobrazované scény budou před pozorovatelem. V případě nesplnění těchto předpokladů je nezbytné použít ořezávání v třírozměrném prostoru, viz kap.6.12, s určením "pohledové pyramidy".

Označíme-li d vzdálenost průmětny od pozorovatele, pak je projekce definována maticí $\mathbf{M}_{per}$

$$\mathbf{M}_{per} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix}$$

Pak celková matice transformace **W** je určena maticovým výrazem

$$W = M_{per} \cdot R_{z_e y_e}(\varphi) \cdot R_{x_e z_e}(\vartheta) \cdot Z \cdot T$$



Obr. 5.3.5

Matice transformace **W** musí být aplikována **na všechny** souřadnice bodů v uvažované scéně. Označíme-li **X** matici, jejíž sloupce jsou tvořeny vektory souřadnic jednotlivých bodů, pak lze psát

$$X' = W \cdot X$$

kde $X = \left[x_1^T \mid x_2^T \mid \ldots \mid x_N^T \right]$

Pro demonstraci uvedeného problému vezměte souřadnice vrcholů tělesa z obr. 3.10.1 a zobrazte je pro různé hodnoty polohy bodu Q při různých volbách vzdálenosti d projekční roviny od pozorovatele.

Úloha

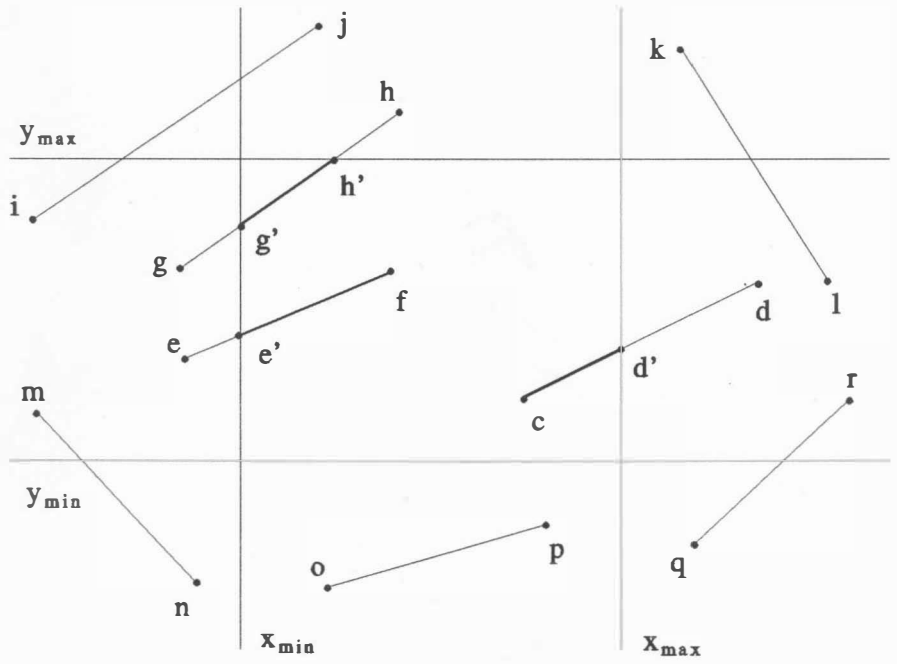
Zobrazte krychli v perspektivní projekci a měňte polohu pozorovatele tak, až se postupně dostanete dovnitř krychle. Získané výsledky diskutujte.

6. Ořezávání

Velmi důležitou částí každého programového vybavení pro počítačovou grafiku je ořezávání těch částí scény, které jsou mimo zobrazovací plochu. Nejjednodušším případem je ořezávání v dvourozměrném prostoru vůči obdélníku zobrazovací plochy. Tento případ je v praxi nejčastější, zejména při zobrazování plošných objektů.

6.1 Dvourozměrné ořezávání

Na obr.6.1.1 jsou uvedeny některé situace, které mohou nastat v případě, kdy je zobrazovací plocha ohraničena obdélníkem. Úkolem ořezávání je odříznout ty části úseček, které jsou mimo obdélník, tj. zobrazit jen ty části, které jsou uvnitř obdélníka $\langle x_{min} , x_{max} \rangle \times \langle y_{min} , y_{max} \rangle$.



Obr.6.1.1

Vzhledem k tomu, že operace ořezávání se používá velmi často, je nutné, aby příslušný algoritmus velmi rychle rozhodl, zda je část úsečky uvnitř obdélníka, a určil příslušné koncové body.

6.2 Cohen - Sutherlandův algoritmus

Algoritmus těží především z jednoduchého způsobu zakódování jednotlivých možností tak, aby jednotlivé případy mohly být snadno rozlišeny. Algoritmus začíná vždy určením čtyřbitového kódu pro každý koncový bod zkoumané úsečky, a to tak, že stavové slovo příslušející danému bodu nabývá hodnoty podle obr.6.2.1, kde hodnota 1 označuje hodnotu TRUE.

1 0 0 1	1 0 0 0	1 0 1 0
0 0 0 1	0 0 0 0	0 0 1 0
0 1 0 1	0 1 0 0	0 1 1 0

Obr.6.2.1

Podle polohy bodu vůči obdélníku ořezávání nabývají jednotlivé bity stavového slova následujících hodnot (bity číslovány zprava doleva):

bit 0	bod je vlevo od obdélníka
bit 1	bod je vpravo od obdélníka
bit 2	bod je pod obdélníkem
bit 3	bod je nad obdélníkem

V případě, že daný programovací jazyk nemá možnost nastavení i-tého bitu, lze nastavení jednotlivých bitů provést pomocí operace sčítání, a to:

```
if x < x_min then S := 1
             else if x > x_max then S := 2
                                else S := 0;
if y < y_min then S := S + 4
             else if y > y_max then S := S + 8;
```

Pro danou úsečku s koncovými body x_1 a x_2 označme S_1 stavové slovo příslušející bodu x_1 a S_2 stavové slovo příslušející bodu x_2. Je zřejmé, že pokud $S_1 = 0$ a zároveň $S_2 = 0$, je daná úsečka

zcela uvnitř daného obdélníka. Mají-li stavová slova S_1 a S_2 takovou hodnotu, že některý bit je v obou případech různý od nuly, pak daná úsečka nebude procházet daným obdélníkem, neboť bude celá nad obdélníkem (nebo pod ním, vlevo nebo vpravo od něho). Uvedený způsob kódování umožňuje rychlé určení takových případů, kdy je úsečka zcela vně a bude tedy vypuštěna, anebo je zcela uvnitř a není nutné počítat průsečíky s hranicí okna.

V případě, že nenastává ani jedna z výše uvedených situací, je nutné najít průsečík s příslušnou hranou obdélníka, přičemž souřadnice průsečíku se určí takto

$$(x_{min} , k . (x_{min} - x_1) + y_1) \quad , \text{je-li bod vlevo,} \quad k \neq \infty$$

$$(x_{max} , k . (x_1 - x_{max}) + y_1) \quad , \text{je-li bod vpravo,} \quad k \neq \infty$$

$$(q . (y_1 - y_{max}) + x_1 , y_{max}) \quad , \text{je-li bod nad,} \quad q \neq \infty$$

$$(q . (y_{min} - y_1) + x_1 , y_{min}) \quad , \text{je-li bod pod,} \quad q \neq \infty$$

kde

$$k = \frac{y_2 - y_1}{x_2 - x_1} \qquad \text{pro} \qquad x_2 \neq x_1$$

resp.

$$q = \frac{x_2 - x_1}{y_2 - y_1} \qquad \text{pro} \qquad y_2 \neq y_1$$

Celý Cohen-Sutherlandův algoritmus může být realizován např. algoritmem 6.2.1. Pro jednoduchost algoritmu jsou zavedeny operátory **land** a **lor** pracující s jednotlivými bity jejich celočíselných operandů jako s logickými proměnnými, přičemž výsledek je ukládán opět do odpovídajících bitů celočíselného výsledku. V Turbo Pascalu lze snadno použít přímo operátory **and** a **or**.

```
var xmin, xmax, ymin, ymax: real;
{ globální proměnné určující velikost výřezu }
procedure CLIP ( x1 , y1 , x2 , y2: real);
label 99;
var x, y: real;
    c, c1, c2: integer;
```

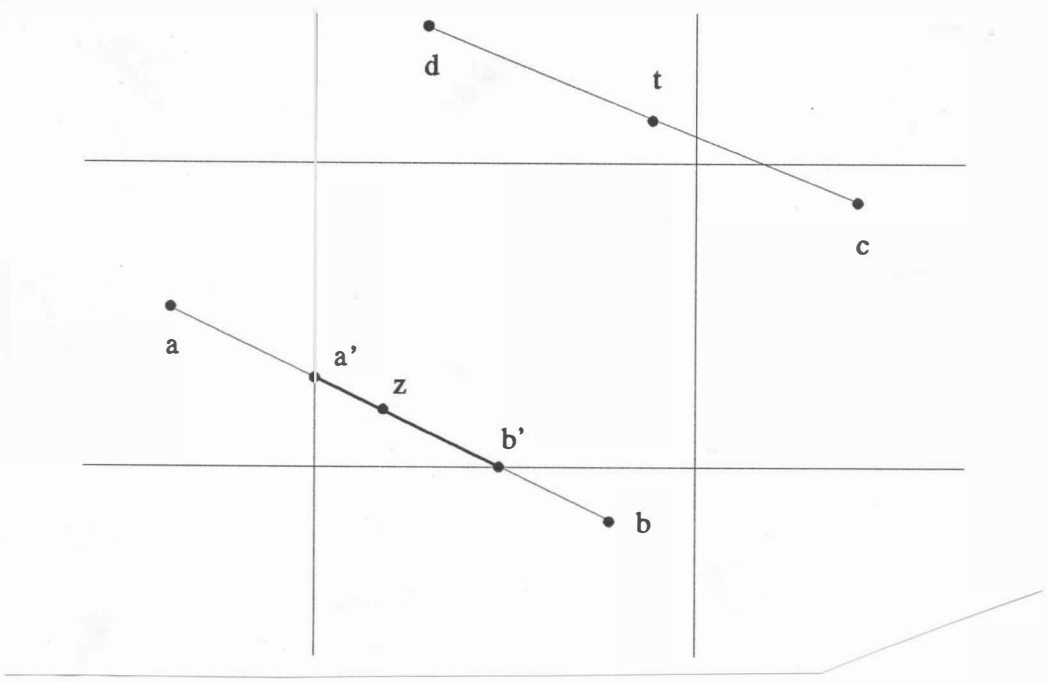
```
procedure CODE ( x, y: real; var c: integer);
begin c:=0;
      if x < xmin then c:=1
                  else if x > xmax then c:=2;
      if y < ymin then c:=c+4
                   else if y > ymax then c:=c+8
end { CODE };

begin
   CODE(x1,y1,c1);
   CODE(x2,y2,c2);
   { zjištění stavového slova }
   if ( c1 land c2 ) = 0 then
   begin { úsečka může procházet obdélníkem }
      if ( c1 lor c2 ) <> 0 then
      { úsečka neleží celá v obdélníku }
      repeat
         if c1 = 0 then c := c2 else c := c1;
         { operátory land, lor provádějí operace and, or }
         { nad všemi bity typu integer; výsledek je opět }
         { typu integer - v T-PASCALU lze použít and, or }
         if (c land 1) <> 0 then { bod je vlevo }
         begin y := y1+(xmin-x1)*(y2-y1)/(x2-x1);
               x:=xmin
         end
         else if (c land 2) <> 0 then { bod je vpravo }
              begin y:=y1+(xmax-x1)*(y2-y1)/(x2-x1);
                    x:=xmax
              end
              else if (c land 4) <> 0 then { bod je pod }
              begin y:=ymin;
                    x:=x1+(ymin-y1)*(x2-x1)/(y2-y1)
              end
              else if (c land 8) <> 0 then { bod je nad }
                   begin y:=ymax;
                         x:=x1+(ymax-y1)*(x2-x1)/(y2-y1)
                   end;
```

```
        if c = c1 then
            begin x1:=x; y1:=y; CODE(x1,y1,c1) end
        else
            begin x2:=x; y2:=y; CODE(x2,y2,c2) end;
        if (c1 land c2) <> 0 then go to 99
      until (c1 lor c2) = 0;
      LINE (x1,y1,x2,y2)
   end;
99:
end { konec procedury CLIP };
```

Algoritmus 6.2.1

Nevýhodou Cohen-Sutherlandova algoritmu je nutnost používání reprezentace čísel s pohyblivou řádovou čárkou včetně operací násobení a dělení, viz např. [44].

Vzhledem k tomu, že se ořezávání velmi často používá v posloupnosti operací až těsně před vlastním zobrazením na daném zařízení, kde jsou souřadnice určené v celočíselné reprezentaci, lze pro tento případ specifikovat speciální algoritmus.



Obr. 6.3.1

6.3 Ořezávání půlením

Algoritmus ořezávání půlením je založen na principu půlení intervalu, který se často používá např. při řešení nelineárních rovnic. Pro zrychlení celého procesu se musí opět co nejrychleji zjistit, zda kreslená úsečka je zcela uvnitř či zcela vně daného obdélníka. K tomu se využije analogický postup jako u Cohen-Sutherlandova algoritmu.

Ořezávání půlením je výhodné zejména z toho důvodu, že nevyžaduje násobení a dělení, neboť dělení dvěma lze realizovat pomocí operace posuvu vpravo. Při programové realizaci je tento algoritmus pomalejší než předchozí, avšak při hardwarové realizaci je nepoměrně rychlejší. Je nutné poznamenat, že lze použít reprezentace s pevnou desetinnou tečkou nebo reprezentace s pohyblivou řádovou tečkou. Počet binárních míst m reprezentující desetinnou část musí být alespoň tak velký, že:

$$2^m > \max \{ nx , ny \},$$

kde nx, resp. ny je rozlišovací schopnost ve směru osy x, resp. osy y.

Z důvodu názornosti algoritmu je zaveden fiktivní datový typ **fixed** m, který reprezentuje hodnoty v pevné řádové čárce o m desetinných místech. Při skutečné realizaci se pak použije reprezentace celočíselná s hodnotami vynásobenými konstantou 2^m, tj. posunutí o m míst vlevo.

```
{ xmin, xmax, ymin, ymax - konstanty určující velikost výřezu }
procedure MIDCLIP ( xa , ya , xb , yb: integer );
{ xa, ya , xb, yb - koncové body úsečky }
label 9;
const m = 10;
var x, y, x1, y1, x2, y2, c, c1, c2: fixed m;
{ definice proměnných s pevnou řádovou čárkou }
procedure CODE ( x, y: fixed m; var c: integer);
begin   c:=0;
   if x < xmin then c:=1
               else if x > xmax then c:=2;
   if y < ymin then c:=c+4
               else if y > ymax then c:=c+8
end { CODE };
```

```
procedure INTER ( x1 , y1 , x2 , y2: fixed m; { krajní body }
                  var x , y: fixed m { průsečík });
{ výpočet průsečíku s obdélníkem, bod (x1,y1) }
{ je uvnitř, bod (x2,y2) je vně }
var c: integer;
begin
   while ( abs (x2-x1) > 2^-m ) or ( abs (y2-y1) > 2^-m ) do
   begin
      x := (x1+x2)/2; { realizovat pomocí posuvu vpravo }
      y := (y1+y2)/2; { nalezení střední hodnoty }
      CODE (x,y,c);
      if c = 0 then begin x1 := x;  y1 := y end
               else begin x2 := x;  y2 := y end
   end;
   x := x1;
   y := y1
end { INTER };
begin
   x1 := xa; x2 := xb;
   y1 := ya; y2 := yb; { desetinná část reprezentována m bity }
   CODE (x1,y1,c1); CODE (x2,y2,c2);
   while not (( c1 land c2 ) <> 0) do
   begin
      if ( c1 lor c2 )=0 then { celá úsečka je uvnitř }
      begin
         LINE (xa,ya,xb,yb);          go to 9
      end;
      { nalezení průsečíku s obdélníkem }
      if c1 = 0 then { bod (x1,y1) je uvnitř }
      begin
         INTER (x1,y1,x2,y2,x,y);
         LINE (xa,ya,x,y);          go to 9
      end;
      if c2 = 0 then { bod (x2,y2) je uvnitř }
      begin
         INTER (x2,y2,x1,y1,x,y);
         LINE (x,y,xb,yb);          go to 9
      end;
```

```
      x:=(x1+x2)/2; y:=(y1+y2)/2;
      CODE (x,y,c);
      if c = 0 then { bod (x,y) uvnitř }
      begin
         INTER (x,y,x1,y1,x1,y1);
         INTER (x,y,x2,y2,x2,y2);
         LINE (x1,y1,x2,y2);
         go to 9
      end;
      { bod x není uvnitř }
      if ( c1 land c ) <> 0 then  { (x1,y1) , (x,y) není vidět }
          begin x1 := x;  y1 := y;  c1 := c end
      else { část (x,y) , (x2,y2) není vidět }
          begin x2 := x;  y2 := y;  c2 := c end ;
      if ( abs (x2-x1) <= 2^-m ) and ( abs (y2-y1) <= 2^-m ) then
                  go to 9
      { odstranění cyklu pro úsečku (4,6),(6,4), je-li obdélník }
      { <5,5> x <10,10>, viz obr.6.3.1 - úsečka LN }
   end { while };
9: end { MIDCLIP } ;
```

Algoritmus 6.3.1

6.4 Algoritmus Liang-Barského

Ke Cohen-Sutherlandovu algoritmu navrhli Liang a Barsky [85], [86] alternativní algoritmus, který vychází z parametrického vyjádření ořezávané úsečky, a to:

$$\mathbf{x}(t) = \mathbf{x}_r + (\mathbf{x}_s - \mathbf{x}_r).t \qquad t \in \langle 0 , 1 \rangle$$

Algoritmus předpokládá, že daná úsečka bude postupně ořezávána vzhledem k jednotlivým polorovinám, tj. uvažují se postupně jen ty části úsečky, které leží ve vyšrafované polorovině, viz obr.6.4.1.

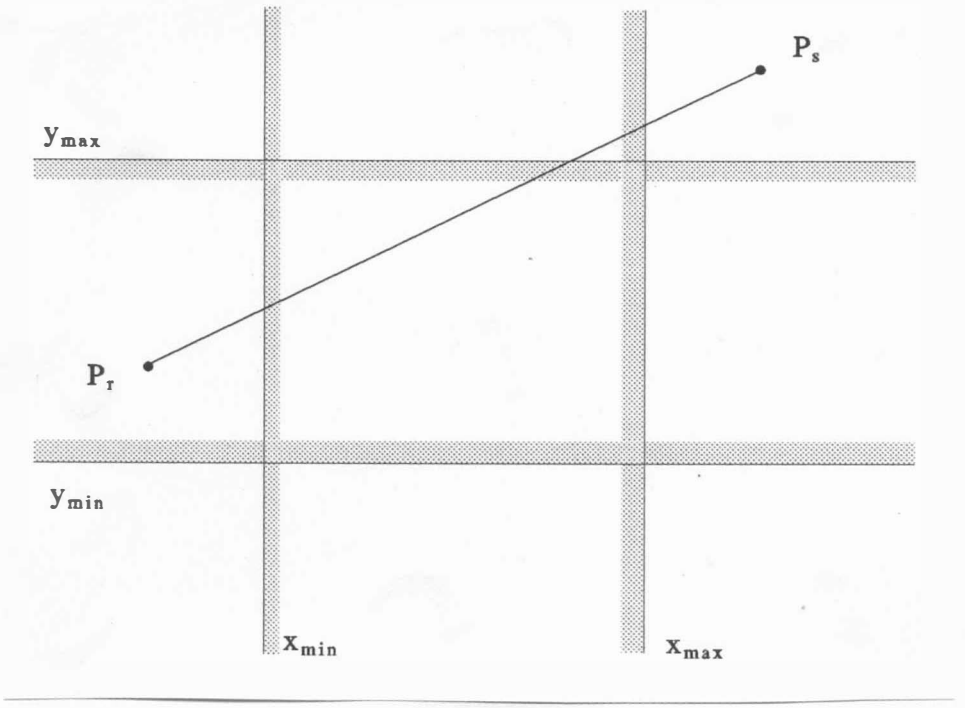
Rovnici lze přepsat do tvaru

$$\mathbf{x}(t) - \mathbf{x}_r = (\mathbf{x}_s - \mathbf{x}_r) \cdot t$$

přičemž musí platit, že

$$\mathbf{x}_{min} \leq \mathbf{x}(t) \leq \mathbf{x}_{max}$$

pro hledaný interval $t \in \langle t_1 , t_2 \rangle$.



Obr. 6.4.1

Uvedenou vektorovou rovnici lze přepsat do tvaru

$$p_k \cdot t \leq q_k \qquad k = 1,2,3,4$$

kde:

$$p_1 = -\Delta x \qquad q_1 = x_r - x_{min}$$

$$p_2 = \Delta x \qquad q_2 = x_{max} - x_r$$

$$p_3 = -\Delta y \qquad q_3 = y_r - y_{min}$$

$$p_4 = \Delta y \qquad q_4 = y_{max} - y_r$$

Nyní je zřejmé, že pro svislé či vodorovné úsečky je pro některá k hodnota $p_k = 0$. Je-li navíc i hodnota $q_k < 0$ pro toto k, pak úsečku lze odmítnout, neboť leží vně uvažované poloroviny. Celý postup je realizován algoritmem 6.4.1.

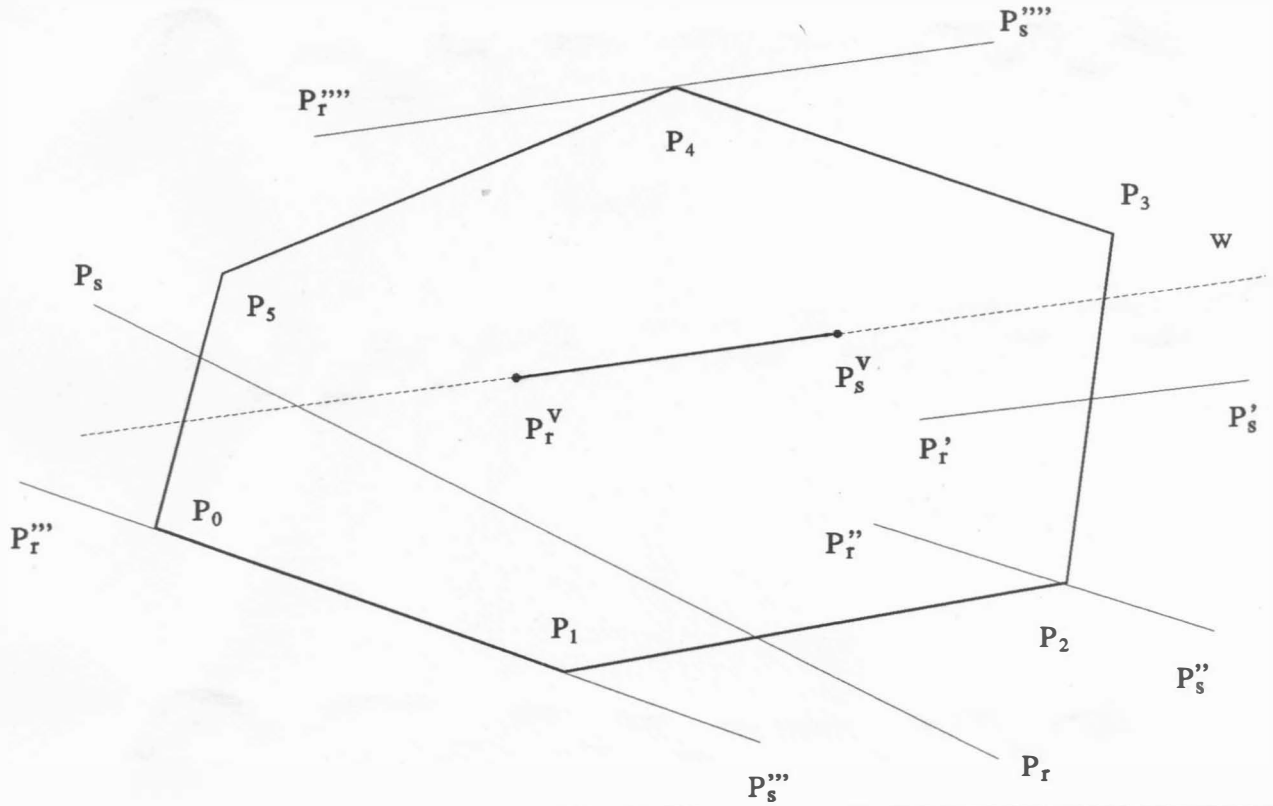
Výhodou Liang-Barského algoritmu je nižší počet operacínásobení, dělení a též nezanedbatelná úspora vzniklá eliminací kódování pozice koncových bodů úsečky.

Dosud uvedené algoritmy ořezávání jsou omezeny na ořezávání obdélníkem, jehož hrany jsou rovnoběžné s osami souřadného systému. V technických aplikacích je však nutné ořezávání vůči konvexnímu n-úhelníku, což dosud uvedené algoritmy neumožňovaly.

```
var xmin, xmax, ymin, ymax: real;
procedure LIANG-BARSKY ( var xr, yr, xs, ys: real );
var t1, t2, dx, dy: real;

function TEST ( p, q: real; var t1, t2: real ): boolean;
var r: real;
begin
   TEST: = true;
   if p < 0 then
   begin r := q / p;
         if r > t2 then TEST := false
                   else if r > t1 then t1 := r
   end
   else
   if p > 0 then
   begin
      r := q / p;
      if r < t1 then TEST := false
                else if r < t2 then t2 := r
   end
   else { úsečka je rovnoběžná k hraně  p = 0 }
      if q < 0 then TEST := false
end { TEST };

begin
   t1 := 0;  t2 := 1;
   dx := xs - xr;
   if TEST ( -dx, xr-xmin, t1, t2 ) then
     if TEST ( dx, xmax-xr, t1, t2 ) then
     begin dy := ys - yr;
        if TEST ( -dy, yr-ymin, t1, t2 ) then
          if TEST ( dy, ymax-yr, t1, t2) then
          begin
             if t1 > 0 then
             begin xr := xr + dx * t1;
                   yr := yr + dy * t1
             end;
```

```
            if t2 < 1 then
            begin xs := xr + dx * t2;
                  ys := yr + dy * t2
            end;
            LINE ( xr, yr, xs, ys )
         end
      end
end { LIANG-BARSKY };
{ (xr,yr),(xs,ys) jsou koncové body oříznuté usečky }
```

Algoritmus 6.4.1



Obr. 6.5.1

6.5 Ořezávání konvexním n-úhelníkem

Předpokládejme, že je dán konvexní n-úhelník, a to s orientací ve směru nebo proti směru hodinových ručiček, a uvažme různé situace, které mohou nastat, viz obr.6.5.1. Jednotlivé průsečíky přímky w, resp. úsečky $P_r P_s$ s konvexním n-úhelníkem získáme řešením soustavy lineárních rovnic, neboť pro ořezávanou úsečku platí rovnice

$$\mathbf{x}(q) = \mathbf{x}_r + (\mathbf{x}_s - \mathbf{x}_r) \,.\, q \qquad q \in \langle 0 , 1 \rangle$$

a pro hrany n-úhelníka platí

$$\mathbf{x}(p) = \mathbf{x}_i + (\mathbf{x}_{i+1} - \mathbf{x}_i) \,.\, p \qquad p \in \langle 0 , 1)$$

$$i = 0 , 1 , \ldots , n-1$$

kde operátor + (u indexů) značí sčítání modulo n.

Z důvodů jednoznačnosti, abychom v případě průchodu přímky vrcholem oblasti měli pouze jeden průsečík, je definiční interval pro parametr p polouzavřený. Vlastní algoritmus je založen na tom, že konvexní n-úhelník může mít nejvýše dva průsečíky s přímkou w (s výjimkou totožnosti s hranou), viz obr.6.5.1. Nejdříve se naleznou hodnoty parametru q odpovídající průsečíkům přímky s konvexním n-úhelníkem (označené t_i) a poté se určí společná část takto získané úsečky s úsečkou $P_r P_s$. Hodnoty t_1, t_2 získané algoritmem je nutné dosadit do příslušné rovnice pro $\mathbf{x}(q)$, abychom obdrželi souřadnice x, y koncových bodů oříznuté úsečky.

Vlastní algoritmus byl částečně zjednodušen, neboť předpokládá, že neexistují dvě hrany n-úhelníka takové, které by ležely na společné přímce. V případě, že tato možnost může nastat, je nutné podmínku označenou ξ redukovat jenom na podmínku i > n a sekvenci označenou ζ je nutné nahradit sekvencí jinou, viz algoritmus 6.5.2.

```
var i, j, k: integer;
    t: array [1..2] of real;
begin
   j := 0;  i := 0;  k := n - 1;
   repeat
      if existuje průsečík hrany x_k x_i a přímky w(q) tak,
         že p ∈ ⟨0,1) then
      begin j:=j+1;
            t[j]:=q { uložení hodnoty q }
      end
      else
         if hrana x_k x_i leží na w(q) then
         begin { sekvence ζ }
            t[1]:=hodnota q odpovídající vrcholu x_k;
            t[2]:=hodnota q odpovídající vrcholu x_i;
            j:=2 { konec sekvence ζ }
         end;
      k := i; i := i + 1
   until ( j = 2 ) or ( i > n ); { podmínka ξ }
   if j <> 0 then
   begin if j = 1 then t[2]:=t[1] { přímka w(q) se dotýká }
                  else if t[1] > t[2] then t[1] swap t[2];
         t[1] := max ( 0.0, t[1] ); { maximální hodnota }
         t[2] := min ( 1.0, t[2] ); { minimální hodnota }
          if t[1] ≤ t[2]  then LINE ( x(t[1]) , x(t[2]) )
   end
end;
```

Algoritmus.6.5.1

```
if j = 0 then begin t[1] := q[1];  t[2] := q[2] end
         else begin if t[1] > t[2] then t[1] swap t[2];
                    if q[1] > q[2] then q[1] swap q[2];
                    t[1] := min ( q[1] ,t[1] );
                    t[2] := max ( q[2] ,t[2] )
              end;
              { operátor swap realizuje vzájemnou výměnu hodnot }
```

Algoritmus 6.5.2

6.6 Cyrus-Beckův algoritmus pro ořezávání konvexním n-úhelníkem

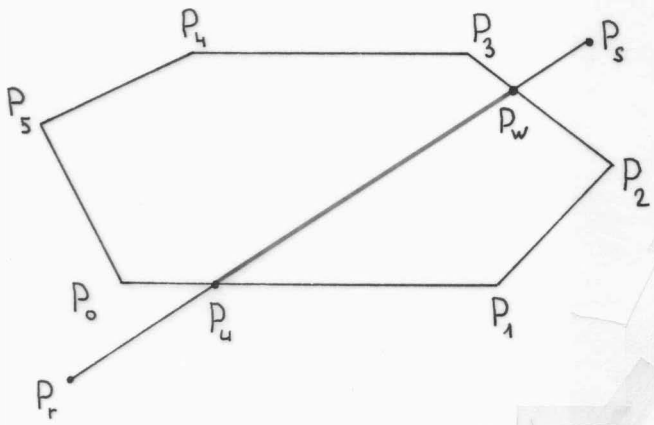
Cyrus-Beckův algoritmus je založen na znalosti normál hran konvexního n-úhelníka a jejich orientace. Je známo, že přímka může protínat hranice konvexního n-úhelníka nejvýše ve dvou bodech. Vyjádříme-li úsečku z bodu P_r o souřadnicích $\mathbf{x}_r$ do bodu P_s o souřadnicích $\mathbf{x}_s$ parametricky, dostáváme

$$\mathbf{x}(t) = \mathbf{x}_r + (\mathbf{x}_s - \mathbf{x}_t) \,.\, t \qquad 0 \le t \le 1$$

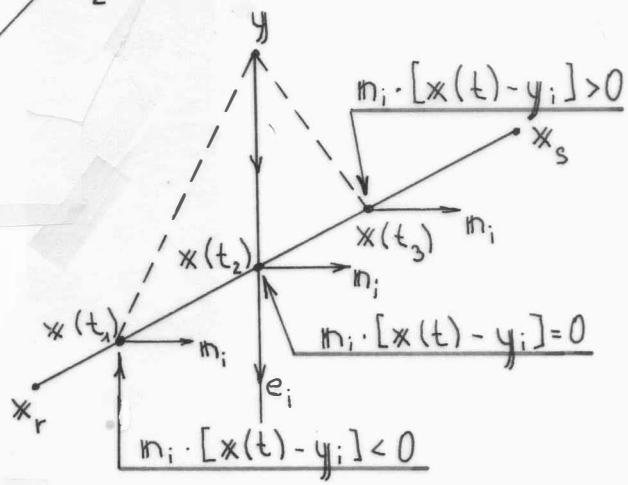
nebo po rozepsání:

$$x(t) = x_r + (x_s - x_r) \,.\, t$$

$$y(t) = y_r + (y_s - y_r) \,.\, t \qquad 0 \le t \le 1$$



Obr.6.6.1



Obr.6.6.2

Uvažujeme-li obecný konvexní n-úhelník, např. viz obr.6.6.1, vidíme, že se celý problém redukuje na nalezení dvou hran konvexní oblasti, které daná úsečka, resp. přímka, protíná, a na určení souřadnic průsečíků.

Je-li $\mathbf{e}_i$ orientovaná hrana konvexního n-úhelníka, $\mathbf{y}_i$ bod této hrany a $\mathbf{n}_i$ normála této hrany, pak směr výsledného vektoru $\mathbf{x}(t) - \mathbf{y}_i$ lze určit pomocí znaménka výsledku skalárního součinu, viz obr. 6.6.2, a to

je-li $\mathbf{n}_i \cdot [\, \mathbf{x}(t)-\mathbf{y}_i \,] < 0$, pak vektor $\mathbf{x}(t)-\mathbf{y}_i$ směřuje ven z konvexní oblasti,

je-li $\mathbf{n}_i \cdot [\, \mathbf{x}(t)-\mathbf{y}_i \,] = 0$, pak vektor $\mathbf{x}(t)-\mathbf{y}_i$ je kolmý na normálu,

je-li $\mathbf{n}_i \cdot [\, \mathbf{x}(t)-\mathbf{y}_i \,] > 0$, pak vektor $\mathbf{x}(t)-\mathbf{y}_i$ směřuje dovnitř konvexní oblasti.

Dosadíme-li za $\mathbf{x}(t)$, pak dostáváme výraz

$$\mathbf{n}_i \cdot [\, \mathbf{x}_r + (\mathbf{x}_s - \mathbf{x}_r) \cdot t - \mathbf{y}_i \,] = 0$$

jako podmínku pro bod průsečíku s hranou, tj:

$$\mathbf{n}_i \cdot [\, \mathbf{x}_r - \mathbf{y}_i \,] + \mathbf{n}_i \cdot [\, \mathbf{x}_s - \mathbf{x}_r \,] \cdot t = 0$$

Označíme-li

$$\mathbf{d} = \mathbf{x}_s - \mathbf{x}_r \qquad \mathbf{w}_i = \mathbf{x}_r - \mathbf{y}_i$$

pak $\mathbf{d}$ určuje směr úsečky $\mathbf{x}_r\mathbf{x}_s$, zatímco vektor $\mathbf{w}_i$ je úměrný vzdálenosti počátečního bodu od hranice obrysu. Lze tedy psát

$$t \cdot \mathbf{n}_i \cdot \mathbf{d} + \mathbf{w}_i \cdot \mathbf{n}_i = 0$$

a tedy

$$t = - \frac{\mathbf{w}_i \cdot \mathbf{n}_i}{\mathbf{n}_i \cdot \mathbf{d}} \quad , \text{ je-li } \mathbf{d} \neq \mathbf{0} \quad \text{a} \quad i = 1, 2, \ldots , n$$

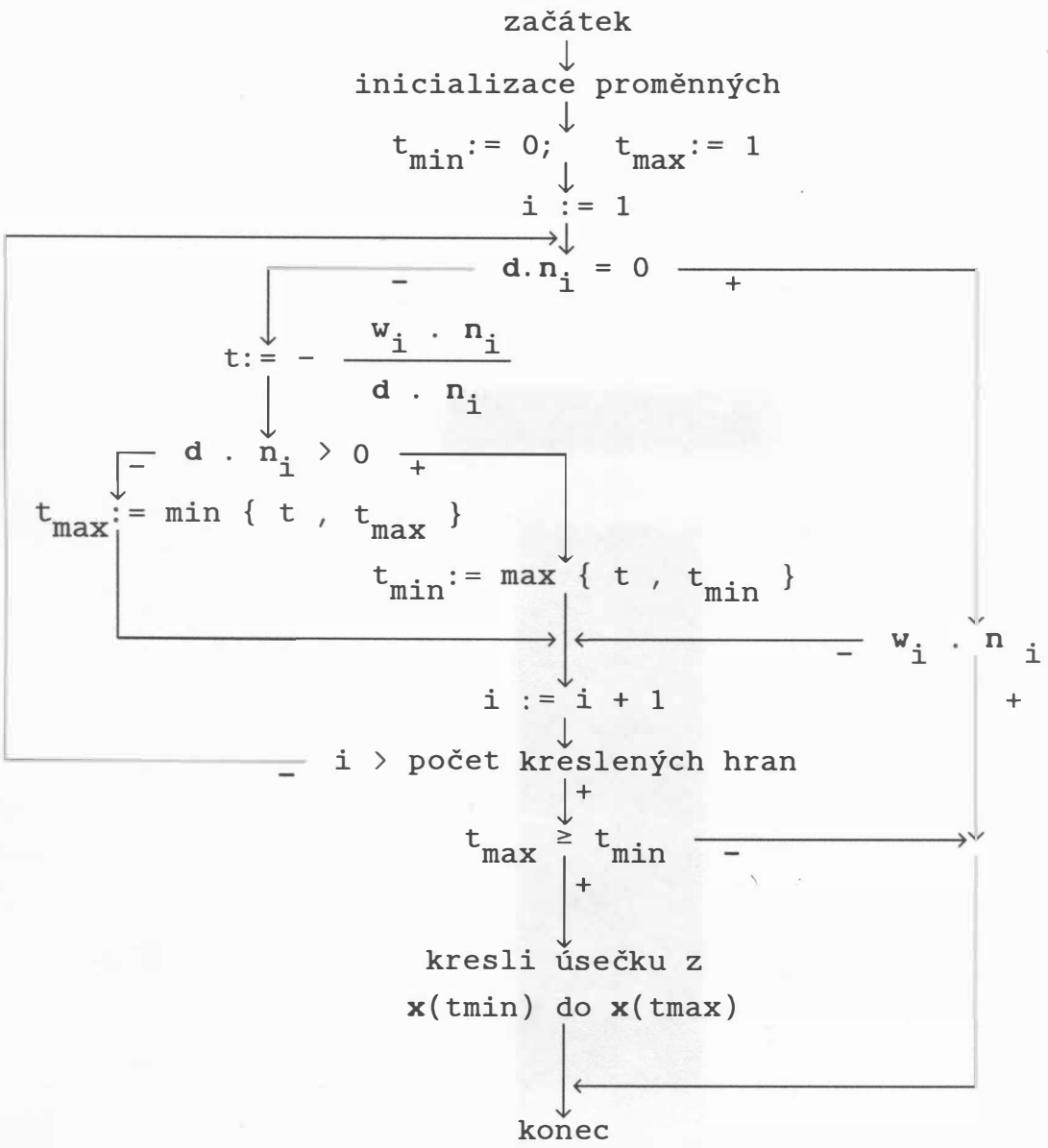
V případě, že $\mathbf{d} = \mathbf{0}$, je úsečka $\mathbf{x}_r\mathbf{x}_s$ nulové délky.

Je-li:

$$\mathbf{w}_i \cdot \mathbf{n}_i \begin{cases} < 0 \text{ , bod je vně oblasti} \\ = 0 \text{ , bod je na hranici oblasti} \\ > 0 \text{ , bod je uvnitř oblasti} \end{cases}$$

Je zřejmé, že budeme uvažovat jen ty průsečíky, pro které bude platit, že $t \in \langle 0 , 1 \rangle$. Celý algoritmus je pak znázorněn na obr. 6.6.3.

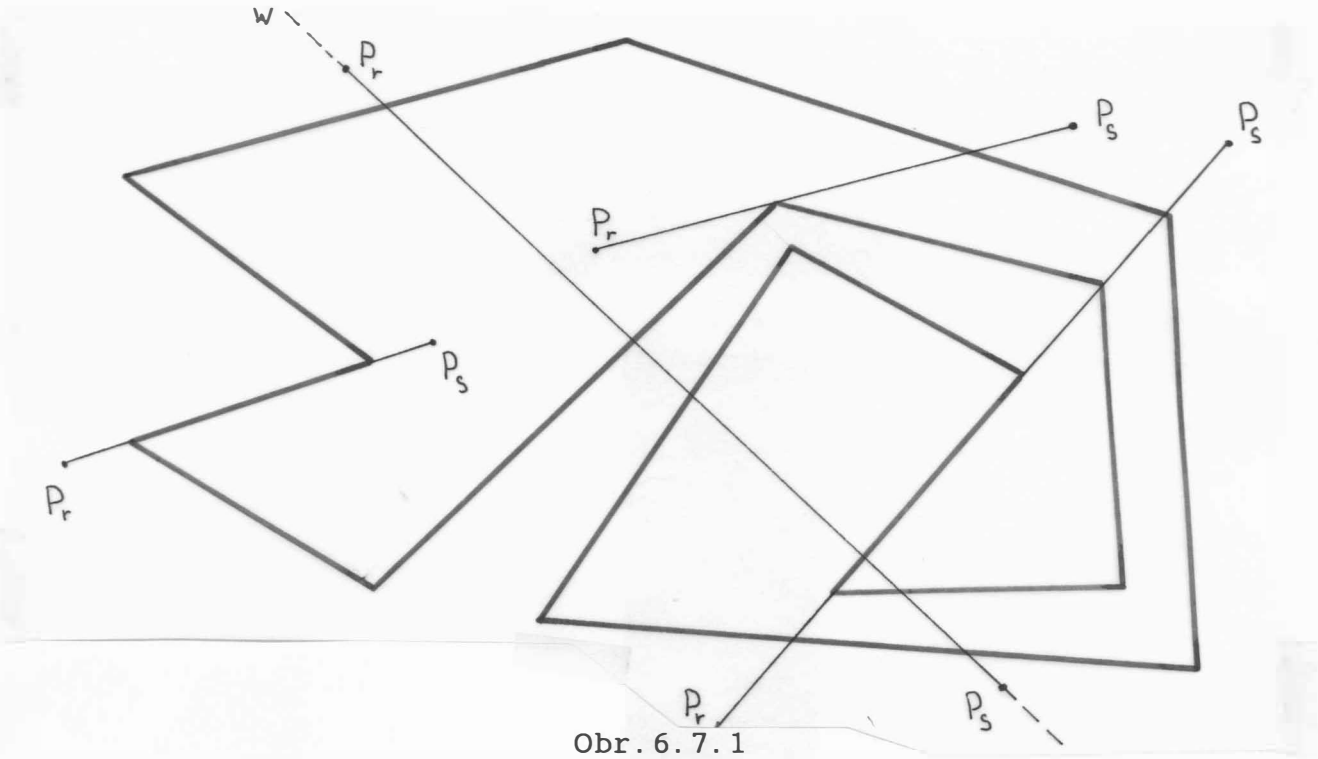
Nevýhodou uvedeného algoritmu je nutnost výpočtu normálového vektoru pro každou hranu dané oblasti v závislosti na orientaci konvexního n-úhelníka reprezentujícího danou oblast, podrobně viz [109]. Algoritmy založené na jiných přístupech lze nalézt např. v [80], [95].


začátek
inicializace proměnných
$t_{min} := 0$; $t_{max} := 1$
i := 1
$d.n_i = 0$
-
+
$t := -\frac{w_i \cdot n_i}{d \cdot n_i}$
$d \cdot n_i > 0$
-
+
$t_{max} := \min \{ t , t_{max} \}$
$t_{min} := \max \{ t , t_{min} \}$
$w_i \cdot n_i$
-
+
i := i + 1
i > počet kreslených hran
-
+
$t_{max} \geq t_{min}$
-
+
kresli úsečku z
x(tmin) do x(tmax)
konec

Obr. 6. 6. 3

6.7 Ořezávání nekonvexním n-úhelníkem

Algoritmus pro ořezávání nekonvexním n-úhelníkem je založen na parametrickém vyjádření úseček. Oproti algoritmu, který byl uveden v kap.6.5, však musí být řešeny jisté speciální případy. Uvažme situaci na obr.6.7.1 a několik možných případů ořezávání.



Obr.6.7.1

Předpokládejme, že hrany nekonvexního n-úhelníka jsou opět dány ve směru anebo proti směru hodinových ručiček a že se vzájemně neprotínají.

Pro jednoduchost předpokládejme:

- pouze hrany sousední mají společný bod,
- sousední hrany neleží na společné přímce,
- vrcholy jsou navzájem různé.

Dále uvedený algoritmus lze modifikovat i pro n-úhelníky nesplňující výše uvedené předpoklady.

Označme $\mathbf{x}(q)$ souřadnice bodů přímky **w** a vyjádřeme je parametricky jako

$$\mathbf{x}(q) = \mathbf{x}_r + (\mathbf{x}_s - \mathbf{x}_r) \, . \, q \qquad q \in (-\infty , \infty)$$

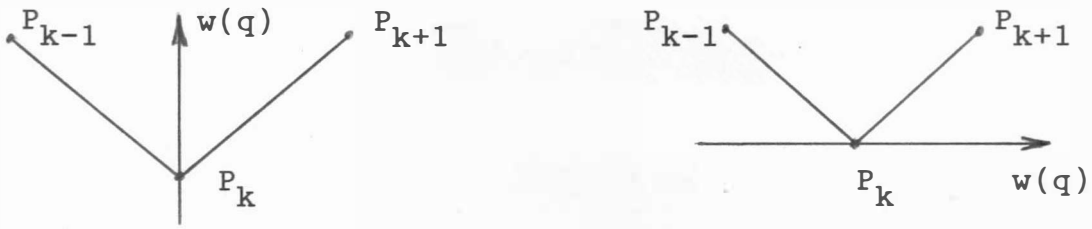
a souřadnice bodů $\mathbf{x}(p)$ každé hrany n-úhelníka jako

$$\mathbf{x}(p) = \mathbf{x}_i + (\mathbf{x}_{i+1} - \mathbf{x}_i) \, . \, p \qquad p \in \langle 0 , 1)$$
$$i = 0 , 1 , \ldots , n-1$$

Budeme zatím hledat průsečíky hran n-úhelníka s přímkou w, na které leží ořezávaná úsečka $P_r P_s$. Je nutné poznamenat, že operace + u indexů značí operaci modulo n.

Kromě průsečíku přímky s hranou musíme rozlišit případy, kdy hrana n-úhelníka leží na přímce w(q) a kdy přímka w(q) prochází vrcholem. Při průchodu přímky w(q) vrcholem mohou nastat následující případy:

$[s_1 \times s_2]_z . [s_3 \times s_2]_z > 0$ $\qquad$ $[s_1 \times s_2]_z . [s_3 \times s_2]_z < 0$



a) "průchod" $\qquad$ b) "dotyk"

kde + v indexech značí součet modulo n
- v indexech značí rozdíl modulo n

a $\quad s_1 = x_k - x_{k-1} \qquad s_2 = x_s - x_r \qquad s_3 = x_{k+1} - x_k$

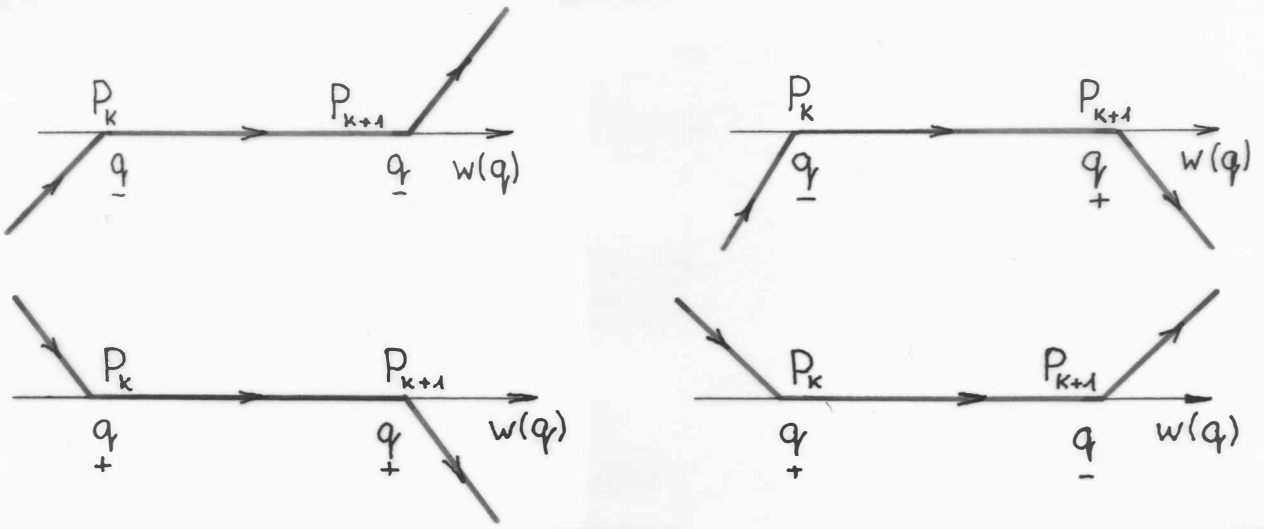
Obr. 6.7.2

V případě ad a) se generuje pouze jeden průsečík, zatímco v případě ad b) se generují dva totožné body. 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).

V případě, že na přímce w(q) leží některá hrana, mohou nastat situace, které jsou znázorněny na obr. 6.7.3. 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 - znaménkem z-ové souřadnice vektorového součinu $[s_1 \times s_2]_z$, resp. $[s_3 \times s_2]_z$ pro "dotyk" nebo "průchod" vrcholem.

Po nalezení všech průsečíků, včetně určení jejich atributů, musí být získaná množina hodnot q setříděna opět spolu s atributy. V následujícím kroku je nutné provést redukci získaných hodnot q podle tab.6.7.1. Výsledkem je pak množina dvojic hodnot q, které určují úseky přímky w(q) a které leží 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 algoritmem 6.7.1.

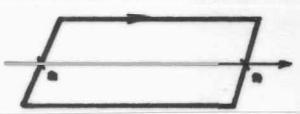
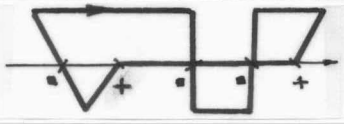
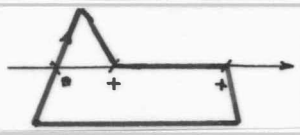
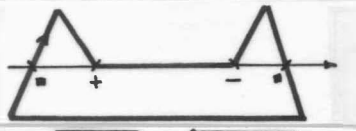
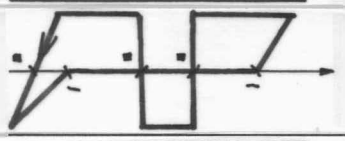
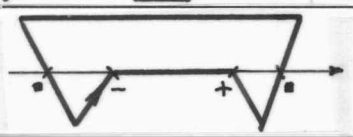


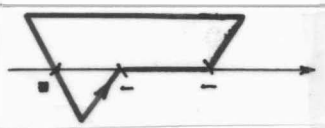
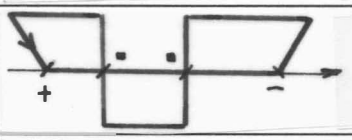
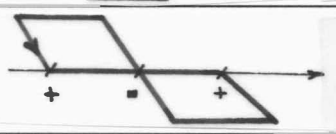
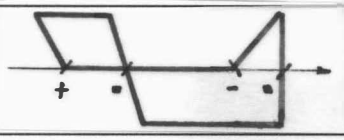
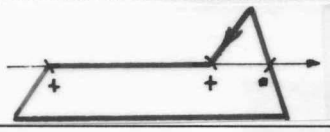
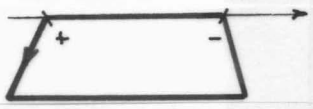
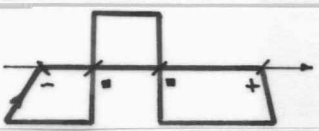
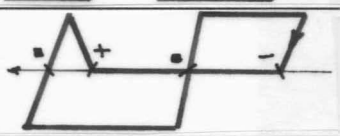
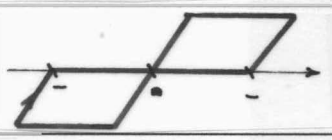
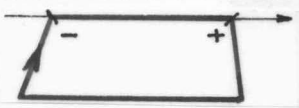
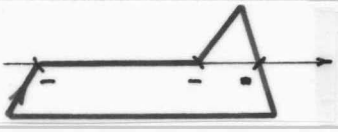
Obr.6.7.3

```
k:=n-1; i:=0;
s1 := xk - xk-1;   s2 := xs - xr; { s a x jsou vektory }
while i < n do
begin
   s3:= xi - xk;  { operace s vektory }
   Vypočti hodnotu (q);
   if vrchol xk leží na přímce w(q)
   then
   begin
      if [s3 x s2]z = 0
      then { hrana xkxi leží na přímce w(q) }
         Generuj (q s atributem sign ( [s1 x s2]z)
```

```
      else if [s1 x s2]z = 0
           then { hrana xk-1xk leží na přímce w(q) }
             Generuj (q s atributem sign ( [s3 x s2]z)
           else if [s1 x s2]z . [s3 x s2]z < 0
                then { typ dotyk }
                   Generuj ( q ,q s atributem • )
                else { typ průchod }
                  Generuj ( q s atributem • )
   end
   else
   if existuje-li průsečík přímky w(q) s hranou <xk,xi)
        then Generuj (q s atributem • );
   s1:=s3; k:=i; i:=i+1
end;
USPORADEJ VZESTUPNE( získané hodnoty q );
REDUKUJ ( hodnoty q podle tabulky );
VYBER ( podintervaly jako <qj,qj+1> ∩ <0,1>  ∀ j );
```

Algoritmus 6.7.1

atributy q_i	q_{i+1}	q_{i+2}	situace	činnost
∘	∘	*		ulož(q_i,q_{i+1}); i:=i+2
∘	+	∘		% ulož(q_i,q_{i+2}); i:=i+2 změň atribut q_i na +
∘	+	+		ulož(q_i,q_{i+2}); i:=i+3
∘	+	−		ulož(q_i,q_{i+2}); i:=i+2 změň atribut q_i na ∘
∘	−	∘		% ulož(q_i,q_{i+2}); i:=i+2 změň atribut q_i na −
∘	−	+		ulož(q_i,q_{i+2}); i:=i+2 změň atribut q_i na ∘

◦	−	−		ulož(q_i,q_{i+2}); i:=i+3
+	◦	◦		% ulož(q_i,q_{i+2}); i:=i+2 změň atribut q_i na +
+	◦	+		% ulož(q_i,q_{i+2}); i:=i+3
+	◦	−		% ulož(q_i,q_{i+2}); i:=i+2 změň atribut q_i na ◦
+	+	*		ulož(q_i,q_{i+1}); i:=i+1 změň atribut q_i na ◦
+	−	*		ulož(q_i,q_{i+1}); i:=i+2
−	◦	◦		% ulož(q_i,q_{i+2}); i:=i+2 změň atribut q_i na −
−	◦	+		% ulož(q_i,q_{i+2}); i:=i+2 změň atribut q_i na ◦
−	◦	−		% ulož(q_i,q_{i+2}); i:=i+3
−	+	*		ulož(q_i,q_{i+1}); i:=i+2
−	−	*		ulož(q_i,q_{i+1}); i:=i+1 změň atribut q_i na ◦

* znamená všechny případy, tj. ◦ + − .

% případy, kdy se hrany nebo vrcholy dotýkají, nebo se hrany protínají

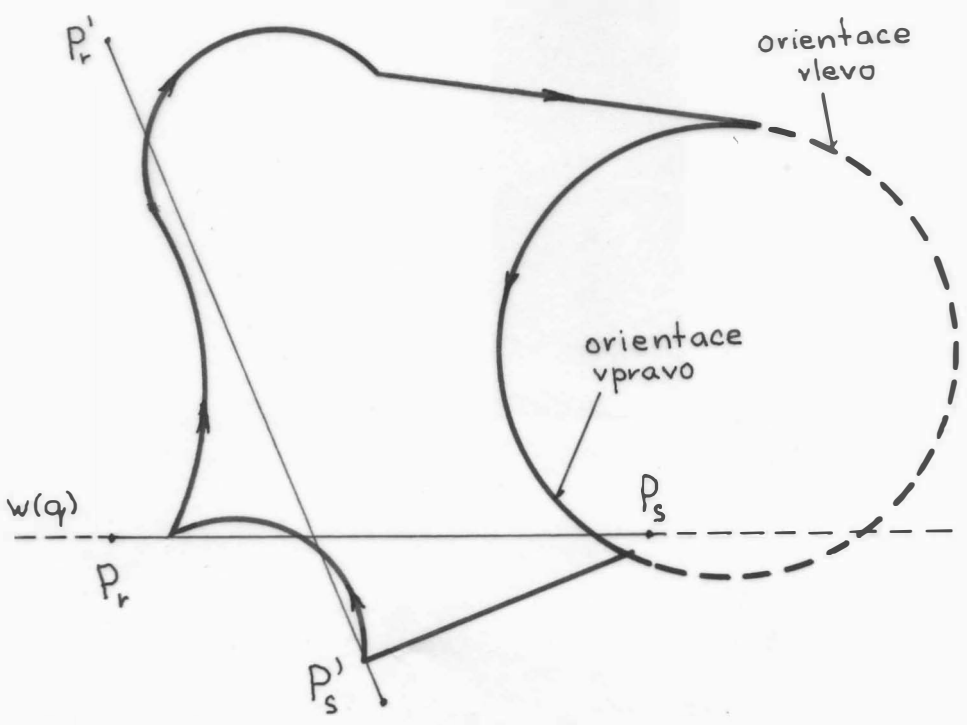
Tabulka 6.7.1

Příkaz pro výběr podintervalů může být realizován např. algoritmem 6.7.2.

```
i:=1;
while i ≤ počet průsečíků - 1 do
begin if max ( 0 , q_i ) ≤ min ( 1 , q_{i+1} )
      then ulož ( max ( 0 , q_i ) , min ( 1 , q_{i+1} ) );
      i:=i+2
end;
```

Algoritmus 6.7.2

Je zřejmé, že uvedený algoritmus je rozšířením algoritmu z kap.6.5, přičemž je nutné použít redukci a třídění vzhledem k tomu, že n-úhelník je nekonvexní (podrobně viz [64], [117]).



Obr.6.8.1

6.8 Ořezávání nekonvexních oblastí

Až dosud uváděné algoritmy umožňovaly ořezávání 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 ořezávat úsečky vůči oblasti s hranicemi tvořenými oblouky.

Předpokládejme, 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 k počátečnímu bodu kruhového oblouku) má být vzata v úvahu. Pro zjednodušení algoritmu uvažme následující omezení

- všechny vrcholy mají navzájem různé souřadnice,
- žádný vrchol neleží na hraně nebo na 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 algoritmu z kap.6.7 může mít přímka **w**(q) s kruhovým obloukem dva průsečíky, což poněkud komplikuje řešení problému. Postup pro nalezení všech průsečíků je obdobný, avšak v případě kruhového oblouku musíme řešit následující soustavu rovnic vzhledem k proměnné q

$$\mathbf{x}(q) = \mathbf{x}_r + (\mathbf{x}_s - \mathbf{x}_r) \, . \, q \qquad q \in (-\infty , +\infty)$$

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

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

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

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

V případě, že přímka **w**(q) danou kružnici protíná nebo se jí dotýká, obdržíme řešením rovnice obecně dva reálné kořeny, a to:

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

Nyní je nezbytné určit, které průsečíky leží na části kružnice tvořící hranici dané oblasti. Pomocí testu lze zjistit, zdali průsečík leží vpravo či vlevo od spojnice počátečního a koncového bodu kruhového oblouku. To znamená, že

- je-li oblouk orientován doprava, bude bod $\mathbf{x}(q_i)$ uvažován tehdy a jen tehdy, je-li $[\mathbf{s}_1 \times \mathbf{s}_2]_z < 0 \quad i=1,2$
- je-li oblouk orientován doleva, bude bod $\mathbf{x}(q_i)$ uvažován tehdy a jen tehdy, je-li $[\mathbf{s}_1 \times \mathbf{s}_2]_z > 0 \quad i=1,2$

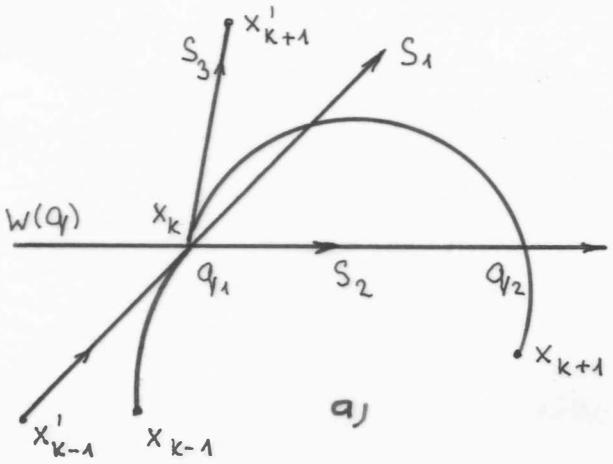
přičemž $\mathbf{x}_k \neq \mathbf{x}(q_i)$, $\mathbf{s}_1 = \mathbf{x}_{k+1} - \mathbf{x}_k$, $\mathbf{s}_2 = \mathbf{x}(q_i) - \mathbf{x}_k$

Je zřejmé, že opět musí být řešeny speciální případy, kdy např. přímka w(q) prochází vrcholem $\mathbf{x}_k$. V těchto případech budou vyhodnoceny vektory $\mathbf{s}_1$, $\mathbf{s}_2$, $\mathbf{s}_3$ takto:

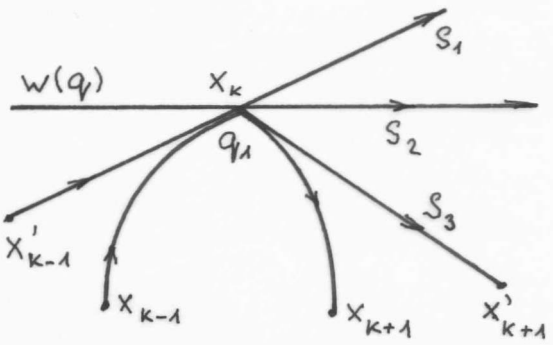
- pro oblouk $\mathbf{s}_1 = [\, y_k - y_u \,,\, x_u - x_k \,]^T$, kde (x_u, y_u) je střed kružnice
  pro hranu $\mathbf{s}_1 = [\, x_k - x_{k-1} \,,\, y_k - y_{k-1} \,]^T$,
  tj. $\mathbf{s}_1 = \mathbf{x}_k - \mathbf{x}_{k-1}$
- pro oblouk $\mathbf{s}_3 = [\, y_k - y_w \,,\, x_w - x_k \,]^T$, kde (x_w, y_w) je střed kružnice
  pro hranu $\mathbf{s}_3 = [\, x_k - x_{k-1} \,,\, y_k - y_{k-1} \,]^T$,
  tj. $\mathbf{s}_3 = \mathbf{x}_k - \mathbf{x}_{k-1}$

kde $\mathbf{s}_1$ a $\mathbf{s}_3$ jsou v případě kruhových oblouků tečnými vektory v bodě $\mathbf{x}_i$.

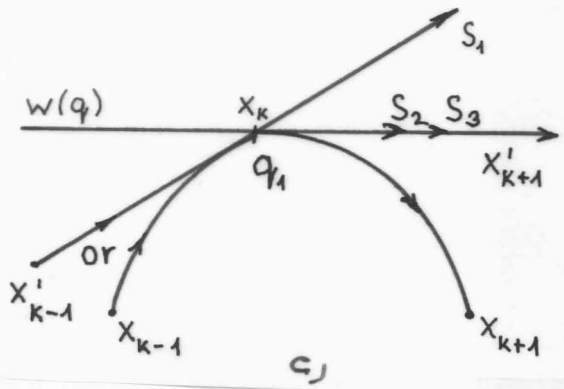
Některé možné situace jsou uvedeny na obr.6.8.2 a v tabulce 6.8.1. pak pravidla pro jejich vyhodnocení. Je-li oblouk orientován doprava, pak musí být změněno znaménko souřadnice z příslušného vektorového součinu. Pak můžeme definovat hodnoty proměnných a , b pomocí sekvencí:


x'_{k+1}
S_3
S_1
$w(q)$
x_k
$q_{/1}$
S_2
$q_{/2}$
x_{k+1}
x'_{k-1}
x_{k-1}

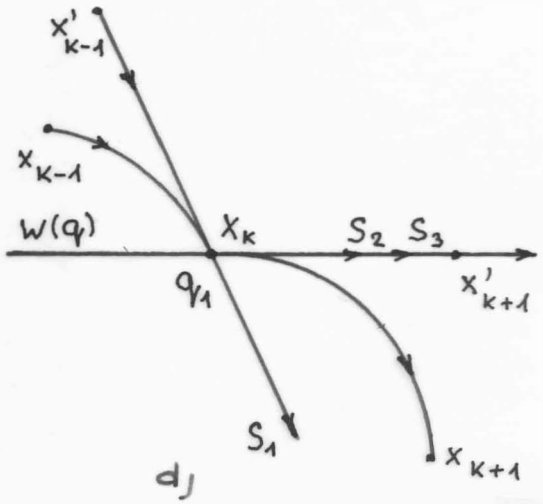
a)


S_1
$w(q)$
x_k
S_2
$q_{/1}$
S_3
x'_{k-1}
x_{k-1}
x_{k+1}
x'_{k+1}

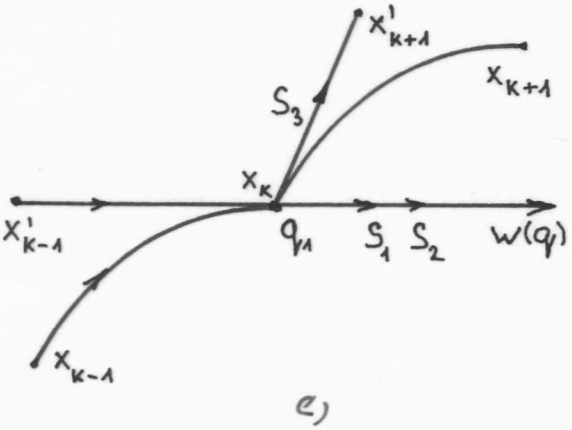
b)


S_1
$w(q)$
x_k
S_2 S_3
$q_{/1}$
x'_{k+1}
or
x'_{k-1}
x_{k-1}
x_{k+1}

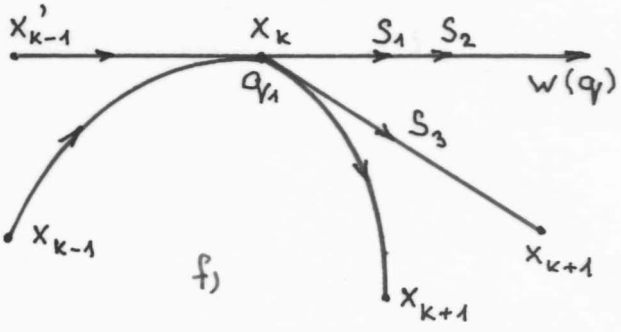
c)


x'_{k-1}
x_{k-1}
$w(q)$
x_k
S_2 S_3
$q_{/1}$
x'_{k+1}
S_1
x_{k+1}

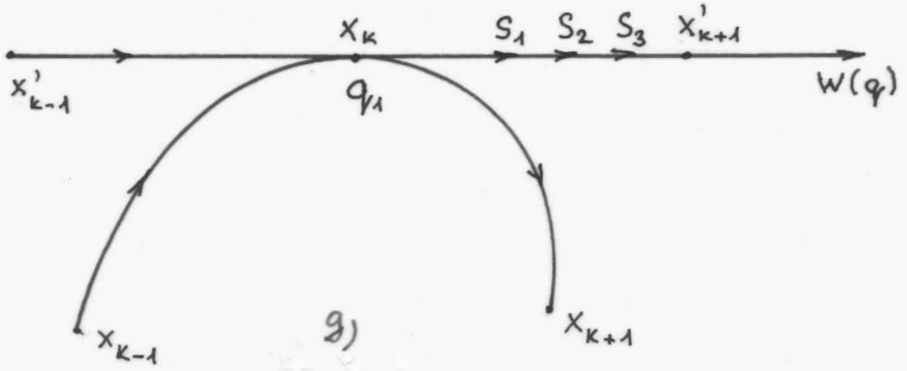
d)


x'_{k+1}
x_{k+1}
S_3
x_k
x'_{k-1}
$q_{/1}$
S_1 S_2
$w(q)$
x_{k-1}

e)


x'_{k-1}
x_k
S_1 S_2
$w(q)$
$q_{/1}$
S_3
x_{k-1}
x_{k+1}
x_{k+1}

f)



Obr. 6.8.2

a:=[s_1 x 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;

b:=[s_3 x s_2]$_z$;

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 6.8.1.

$[s_1 \times s_2]_z$	$[s_3 \times s_2]_z$	situace na obr.6.8.2	typ dotyk/průchod
< 0	< 0	a	průchod
< 0	> 0	b	dotyk
> 0	> 0	+ a	průchod
> 0	< 0	+ b	dotyk
< 0	$= 0$	c	**if** $-s_3.s_2>0$ **then** průchod **else** dotyk
> 0	$= 0$	d	**if** $-s_3.s_2>0$ **then** dotyk **else** průchod
$= 0$	< 0	e	**if** $-s_1.s_2>0$ **then** dotyk **else** průchod
$= 0$	> 0	f	**if** $-s_1.s_2>0$ **then** průchod **else** dotyk
$= 0$	$= 0$	g	**if** $-s_1.s_2>0$ **xor** $-s_3.s_2>0$ **then** průchod **else** dotyk

\+ pro opačnou orientaci přímky w(q)

Tabulka 6.8.1

```
procedure Comp ( xA, xB: vector; var r: real; t: boolean );
begin
   if xAxB je lineární
   then begin s := xB - xA;  r := [ s x s2]z end
   else
   begin s :=[ yk - yw , xw - xk ]T;
         r := [ s x s2]z;
         if r = 0
         then if t then r := s . s2
                   else r := - s . s2
         else if oblouk orientován doprava
              then r := -r
   end
end { Comp };
{ tělo vlastního algoritmu }
k := n - 1;  i := 0;
s2 := xs - xr;
while i < n do
```

```
begin
   if x_k leží na přímce w(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 { předpokládá se, že  x_k = x(q) }
            Výpočet hodnoty ( 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 ( q_1 , q_2 );
                   if průsečík existuje
                   then Generuj( q_1*, q_2* s atributem ∘ )
                   {* značí, že průsečík leží na požadované }
                   { straně oblouku x_k x_i }
             end;
```

```
   k := i;  i := i + 1;
end { while };
USPORADEJ VZESTUPNE ( získané hodnoty q );
REDUKUJ ( hodnoty q podle tabulky );
VYBER ( podintervaly jako <q_j,q_{j+1}> ∩ <0,1>  ∀ j );
```

Algoritmus 6.8.1

Uvedený algoritmus je možné modifikovat i pro eliptické oblouky a i pro obecný případ kuželosečky f(x,y)=0, kde:

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

přičemž směrový vektor s je dán:

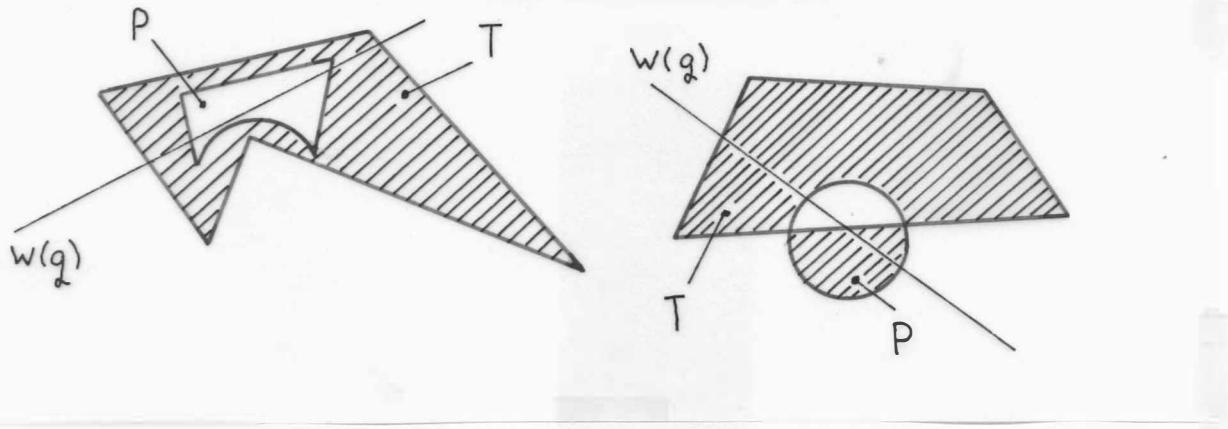
$$s = [\ f_y\ ,\ -f_x\ ]^T$$

Uvedený algoritmus je možné použít i v případě oblastí, které mají "díry" s tím, že algoritmus se aplikuje na vnější hranici a od získaných dvojic hodnot q se odečtou intervaly dvojic hodnot q získaných aplikací algoritmu na hranice děr.

V případě modifikace algoritmu pro šrafování je vhodné nejdříve pootočit oblast tak, aby šrafy byly rovnoběžné s osou x, resp. osou y, spočítat průsečíky šrafy s hranicí oblasti a pak provést zpětné pootočení. Tím se podstatně zjednoduší výpočty jednotlivých průsečíků úsečky s hranicí oblasti.

6.9 Ořezávání úseček složenými oblastmi

Dosud uvedené algoritmy řešily problematiku ořezávání úsečky obecně nekonvexní oblastí, jejíž hranice byla tvořena úsečkami a kruhovými oblouky. V technické praxi se však vyskytují i útvary, které obsahují např. díry. Je zřejmé, že tento případ lze jednoduše vyřešit pomocí množinové operace rozdílu, kdy od oblasti T reprezentující oblast bez díry odečteme množinově oblast reprezentující díru P, viz obr.6.9.1 Výslednou oblast C budeme nazývat složenou oblastí.



C = T - P
a

C = T Δ P
b

Obr.6.9.1

Stejně jako operace rozdílu lze použít i ostatních množinových operací

- sjednocení, ozn. ∪
- průniku, ozn. ∩
- rozdílu, ozn. - , resp. symetrického rozdílu, ozn. Δ

Je zřejmé, že v krajním případě je možné nahlížet na

- konvexní n-úhelník jako na sjednocení trojúhelníků,
- nekonvexní n-úhelník jako na rozdíl konvexní obálky a vhodných konvexních n-úhelníků, resp. jako na sjednocení konvexních n-úhelníků.

Vzhledem k řešené otázce ořezávání vzniká otázka, jakým způsobem je nutné modifikovat dosud uvedené algoritmy, aby je bylo možné použít i v případě množinových operací s oblastmi.

Předpokládejme, že P , T jsou oblasti a že složená oblast C je dána jako

$$C = T \ \mathbf{op} \ P$$

kde: $\mathbf{op} \in \{ \cup , \cap , - , \Delta \}$

Jsou-li části přímky w(q) oříznuté oblastí C dány množinou **c** hodnot dvojic parametrů q:

$$\mathbf{c} = \left\{ \langle c_k , c_{k+1} \rangle \right\}_{k=1}^{L}$$

části přímky w(q) oříznuté oblastí T dány množinou **t** hodnot dvojic parametrů q:

$$\mathbf{t} = \left\{ \langle t_i , t_{i+1} \rangle \right\}_{i=1}^{N}$$

části přímky w(q) oříznuté oblastí P dány množinou **p** hodnot dvojic parametrů q:

$$\mathbf{p} = \left\{ \langle p_j , p_{j+1} \rangle \right\}_{j=1}^{M}$$

Pak platí, že:

$$\mathbf{c} = \mathbf{t} \ \mathbf{op} \ \mathbf{p}$$

Lze zjistit, že v případě nutnosti lze i složité oblasti definovat jako výsledek množinových operací nad konvexními n-úhelníky a kruhy, což však povede k částečnému nárůstu požadavků na výpočetní systém v důsledku provádění operací nad množinami dvojic hodnot parametrů q.

Realizace operace sjednocení nad množinami dvojic hodnot parametrů q je naznačena algoritmem 6.9.1.

```
i:=1; j:=1; k:=1;
c_k:=+∞; c_{k+1}:=-∞;
while (i ≤ N) and (j ≤ M) do
begin
   if p_j < t_i
   then if <t_i,t_{i+1}> ■ < c_k,c_{k+1}> ≠ φ
        then begin c_k:= min (c_k,t_i); c_{k+1}:= max (c_{k+1},t_{i+1});
                   i:=i+2
             end
        else begin k:=k+2; c_k:=+∞; c_{k+1}:=-∞; end
```

```
   else if ⟨p_j,p_{j+1}⟩ ■ ⟨ c_k,c_{k+1}⟩ ≠ φ
        then begin c_k:= min (c_k,p_j); c_{k+1}:= max (c_{k+1},p_{j+1});
                   j:=j+2
             end
        else begin k:=k+2; c_k:=+∞; c_{k+1}:=-∞; end
end { while };
while i ≤ N do
   if ⟨t_i,t_{i+1}⟩ ■ ⟨ c_k,c_{k+1}⟩ ≠ φ
   then
   begin c_k:= min (c_k,t_i); c_{k+1}:= max (c_{k+1},t_{i+1});
         i:=i+2
   end
   else begin k:=k+2; c_k:=+∞; c_{k+1}:=-∞; end;
while j ≤ M do
   if ⟨p_j,p_{j+1}⟩ ■ ⟨ c_k,c_{k+1}⟩ ≠ φ
   then begin c_k:= min (c_k,p_j); c_{k+1}:= max (c_{k+1},p_{j+1});
              j:=j+2
        end
   else begin k:=k+2;
              c_k:=+∞; c_{k+1}:=-∞;
        end;
```

Algoritmus 6.9.1

Operátor ■ je definován jako ∪ (sjednocení) s tím, že interval (+∞,-∞) se považuje za prázdný interval označený φ. Jde vlastně o sjednocení intervalů hodnot q získaných oříznutím přímky w(q) oblastí T a P.

Je-li oblast složená, pak je nezbytné řešit též otázku určení hranice oblasti. Tato úloha není již úlohou zcela triviální. Pokud budeme používat následující řešení, i když neefektivní, dostaneme pouze hrany ohraničující oblast výslednou, které však nejsou uspořádány ve směru nebo proti směru pohybu hodinových ručiček.

Pak v případě operace

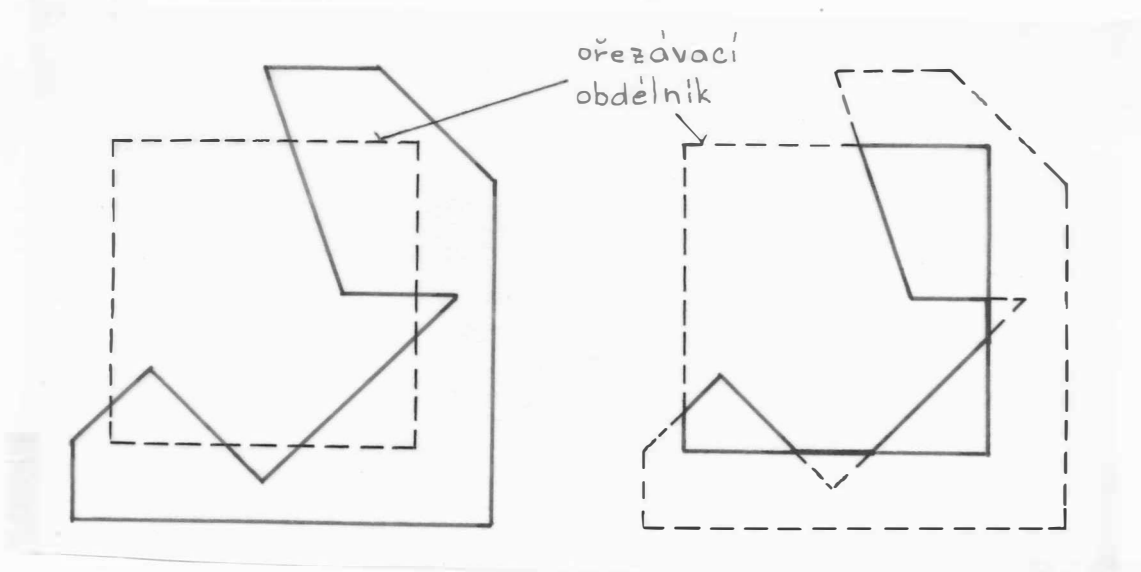
A op B , kde op ∈ { ∪ , ∩ , - , Δ }

je postup pro:

- sjednocení: vezmi všechny hrany oblasti B a proveď "vnější" ořezávání oblastí A; vezmi všechny hrany oblasti A a proveď "vnější" ořezávání oblastí B
- průnik: vezmi všechny hrany oblasti B a proveď "vnitřní" ořezávání oblastí A; vezmi všechny hrany oblasti A a proveď "vnitřní" ořezávání oblastí B
- rozdíl: vezmi všechny hrany oblasti B a proveď "vnitřní" ořezávání oblastí A; vezmi všechny hrany oblasti A a proveď "vnější" ořezávání oblastí B
- symetrický rozdíl: lze realizovat pomocí výše uvedených operací

Je zřejmé, že takto lze definovat jen obrys oblastí, které vzniknou jednou množinovou operací z nesložených oblastí, přičemž navíc je nutné dosud uvedené algoritmy rozšířit o ořezávání kruhového oblouku nekonvexních oblastí, což vede k podstatnému zvýšení složitosti dosud uvedených algoritmů.

Vzhledem k tomu, že je požadováno, aby výsledkem po množinové operaci s n-úhelníky, resp. oblastmi, byl opět n-úhelník, resp. oblast, je nutné zabývat se takovými algoritmy ořezávání n-úhelníků, resp. oblastí, jejichž výsledkem je opět n-úhelník, resp. oblast.

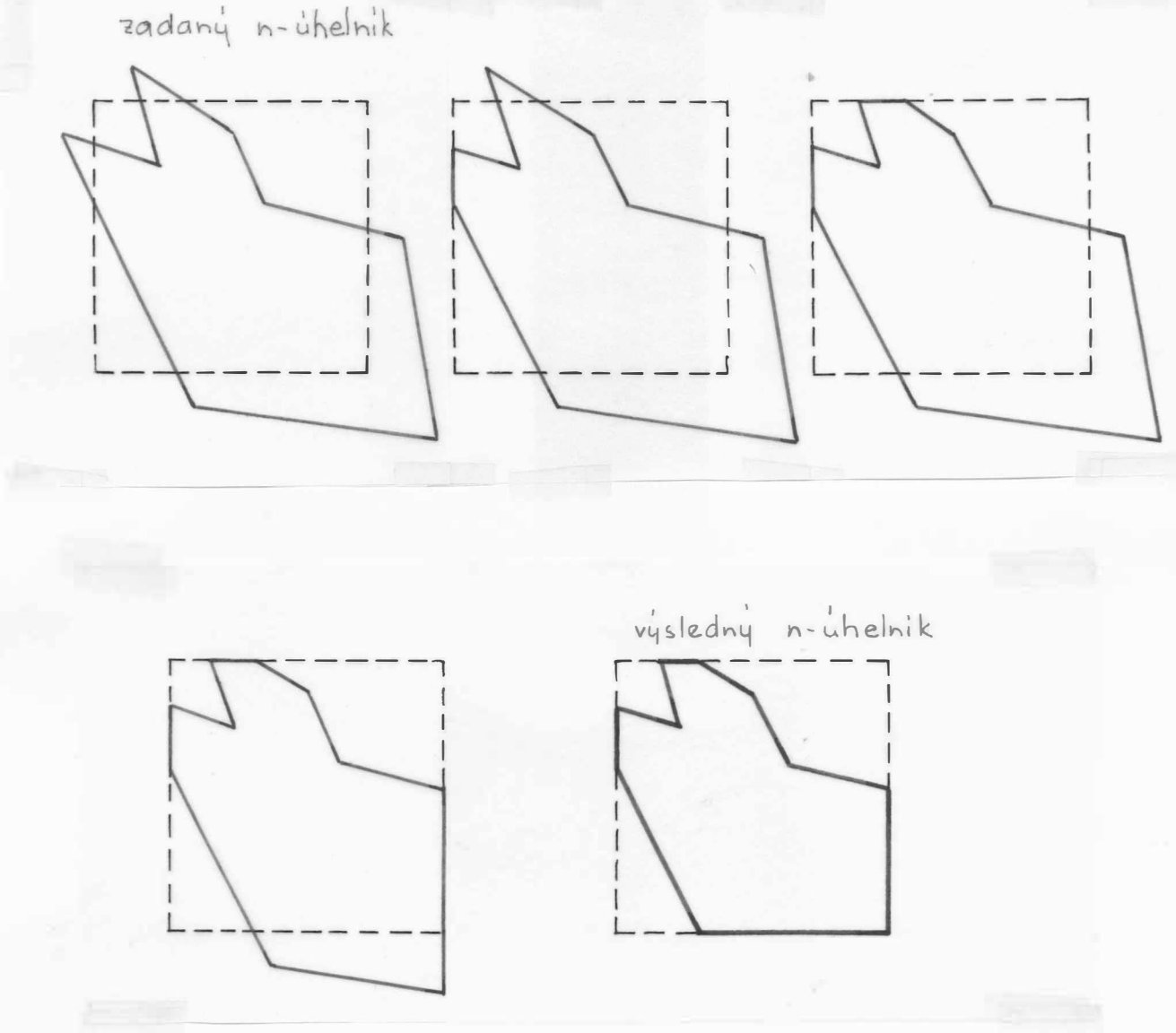


Obr. 6.10.1

6.10 Sutherland-Hodgmanův algoritmus pro ořezávání

Na rozdíl od dříve uvedených algoritmů pro ořezávání přímek či úseček obdélníkem, konvexním či nekonvexním n-úhelníkem nebo oblastí algoritmy určené pro ořezávání n-úhelníka n-úhelníkem vytvářejí jako výstup opět n-úhelník, resp. n-úhelníky. Na obr.6.10.1 je uveden výsledek ořezávání nekonvexního n-úhelníka obdélníkem, kdy se původní n-úhelník rozpadl na tři nové n-úhelníky.

Základní myšlenkou Sutherland-Hodgmanova algoritmu (dále jen S-H algoritmus) je ořezávání n-úhelníka vůči jedné hraně či ořezávací rovině, přičemž se postupně provádí ořezávání hran n-úhelníka vůči jednotlivým hranám obdélníka, viz obr.6.10.2.



Obr.6.10.2

```
const max = 100; { maximální počet vrcholů n-úhelníka }
type vertex = array [1..2] of real; { typ pro vrchol }
     boundary = array [1..2] of vertex; { typ pro hrany }
     polygon = array [1..max] of vertex; { typ pro n-úhelník }
procedure CLIP (in: integer; { počet vrcholů daného n-úhelníka }
                in_v: polygon; { vrcholy vstupního n-úhelníka }
                var out: integer;
                { počet vrcholů výsledného n-úhelníka }
                var out_v: polygon;
                { vrcholy výsledného n-úhelníka }
                clip_boundary: boundary
                { hrana, vůči které se ořezává });

var i, p, s: vertex;
    j: integer;
begin
   out:=0; { nastavení počtu vrcholů výsledného n-úhelníka }
   s:=in_v[in]; { s a p reprezentují vrcholy n-úhelníka }
   for j:=1 to in do
   begin
      p:=in_v[j];
      if INSIDE (p,clip_boundary)
                { s a p odpovídají vrcholům na obr.6.10.3 }
      then { případ 1 a 4 }
           if INSIDE(s,clip_boundary)
           then OUTPUT(p) { případ 1 }
           else begin { případ 4 }
                      i:=INTERSECT(s,p,clip_boundary);
                      OUTPUT(i); OUTPUT(p)
                end
      else { případ 2 a 3 }
         if INSIDE(s,clip_boudary) { případ 2 }
         then begin i:=INTERSECT(s,p,clip_boundary);
                    output(i)
         end; { žádná akce pro případ 3 }
      s:=p; { vezmi další dvojici vrcholů }
   end { od j }
end { CLIP };
```

Algoritmus 6.10.1

Algoritmus 6.10.1 je založený na strategii postupného zkoumání jednotlivých hran n-úhelníka vzhledem k hranici oblasti, která musí být konvexní.

Pro snadnost pochopení algoritmu byly vynechány sekvence příkazů zajišťující ošetření chyb, vyjímečných stavů a též zápis procedur INSIDE, INTERSECT a OUTPUT.

Procedura OUTPUT(q: **vertex**) zajistí uložení vrcholu q do výstupního pole out_v reprezentujícího vrcholy vytvářeného n-úhelníka.

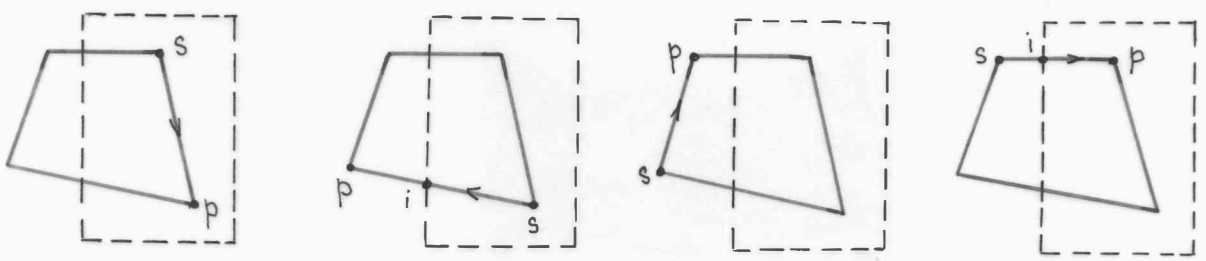
Funkce INTERSECT (s, p: **vertex**; q: **boundary**): **vertex** určí průsečík hrany spojující vrcholy s a p s hranicí q dané oblasti, viz obr.6.10.3.

Procedura INSIDE (s: **vertex**; q: **boundary**): **boolean** nabývá hodnoty **true**, je-li bod s uvnitř ořezávané oblasti. V případě, že hranice oblasti je zadána orientovaně, lze využít vektorového součinu k určení polohy bodu s vůči hraně q. Obecně platí, viz obr.6.10.4, že označíme-li

$$\mathbf{v} = \mathbf{a} \times \mathbf{c} \qquad \mathbf{w} = \mathbf{a} \times \mathbf{b}$$

pak

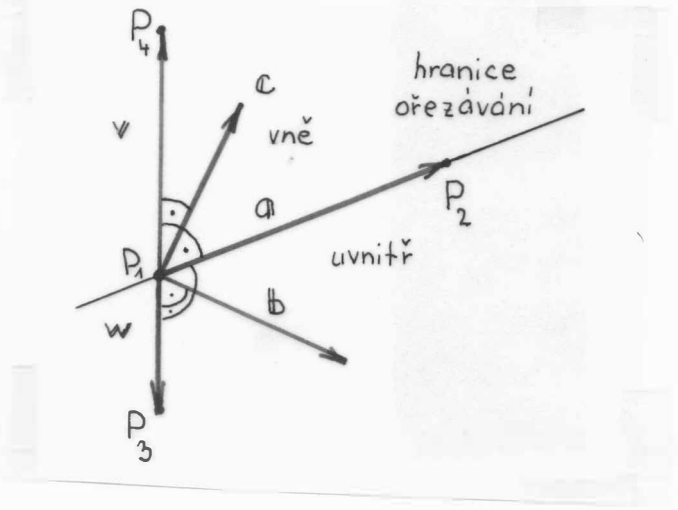
- je-li $\mathbf{v}_z > 0$, tj. z-ová složka vektoru **v**, pak bod P_4 je vně ořezávacího n-úhelníka
- je-li $\mathbf{w}_z < 0$, tj. z-ová složka vektoru **w**, pak bod P_3 je uvnitř ořezávacího n-úhelníka



ulož p ulož i neukládej nic ulož i,p

Obr.6.10.3

Výše uvedený algoritmus lze modifikovat tak, že je rekurzivně volán, a tím lze ušetřit dočasně přidělenou paměť, viz [125]. Algoritmus může být též realizován jako hardwarový procesor využívající "pipe-line" bez požadavků na vnější prostor. Nevýhodou Sutherland-Hodgmanova algoritmu je, že v případě oříznutí nekonvexního n-úhelníka vznikají hrany, které by správně neměly být zahrnuty do výsledného n-úhelníka. Tato skutečnost je dána tím, Sutherland-Hodgmanův algoritmus neposkytuje možnost "roztržení" n-úhelníka oříznutím na více n-úhelníků, viz obr. 6.10.1.



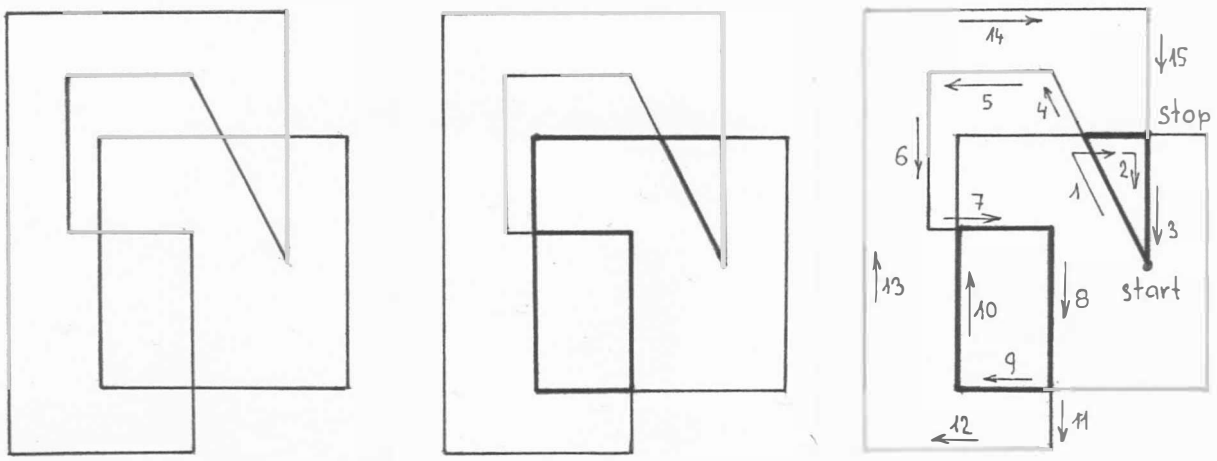
$\mathbf{v} = \mathbf{a} \times \mathbf{c}$ $\quad v_z > 0$ bod P_4 je vně

$\mathbf{w} = \mathbf{a} \times \mathbf{b}$ $\quad w_z < 0$ bod P_3 je uvnitř

Obr. 6.10.4

6.11 Weiler-Athertonův algoritmus

Výše uvedený S-H algoritmus vyžaduje, aby n-úhelník, který tvoří oblast oříznutí, byl konvexní. V mnoha případech je takový předpoklad téměř nesplnitelný. Je pochopitelně možné nekonvexní n-úhelník rozdělit na několik konvexních n-úhelníků, provést oříznutí S-H algoritmem a pak je opět složit dohromady. Uvažme jednoduchý případ, viz obr.6.11.1.a.



ořezávaný n-úhelník | oblast oříznutí | výsledek oříznutí S-H algoritmem | postup a výsledek oříznutí

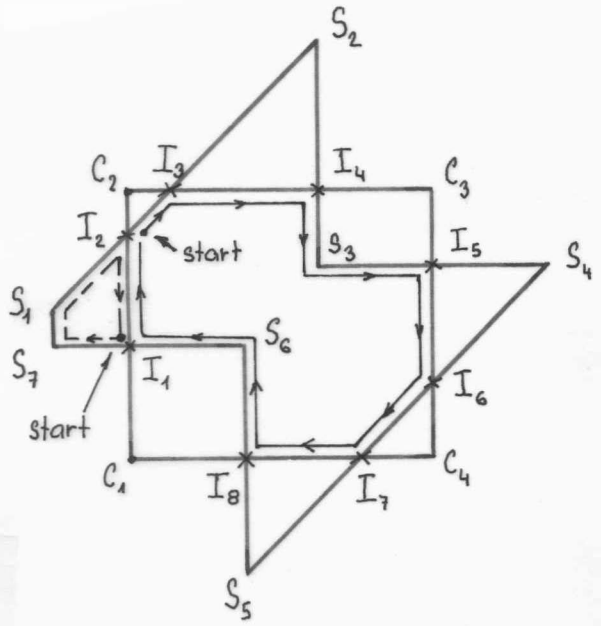
a) b) c)

Obr.6.11.1

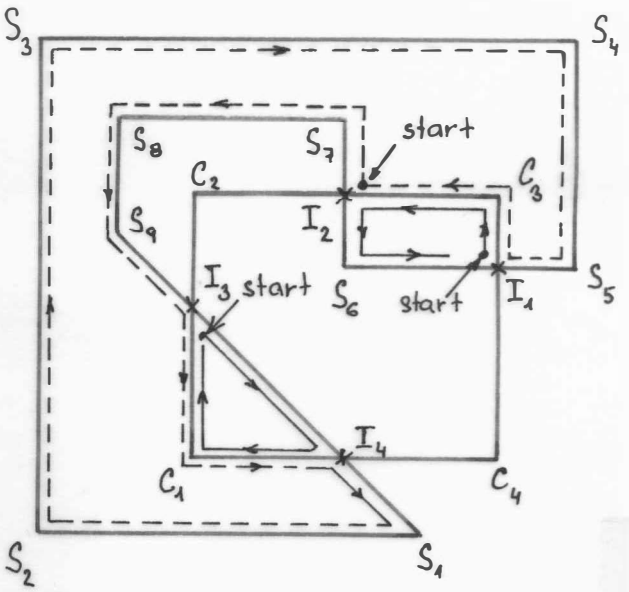
Výsledek po oříznutí S-H algoritmem je na obr.6.11.1.b. Z obrázku je zřejmé, že kromě očekávaných částí ořezávaného n-úhelníka jsou ve výsledném tvaru ještě obsaženy části některých hran n-úhelníka ořezávacího.

Weiler-Athertonův algoritmus (dále jen W-A algoritmus) umožňuje oříznutí obecně nekonvexního n-úhelníka oblastí, která je opět nekonvexním n-úhelníkem. Oba n-úhelníky mohou obsahovat i vnitřní díry. Výsledek oříznutí W-A algoritmem je na obr.6.11.1.c. Z obrázku vyplývá, že výsledek může obsahovat několik samostatných n-úhelníků.

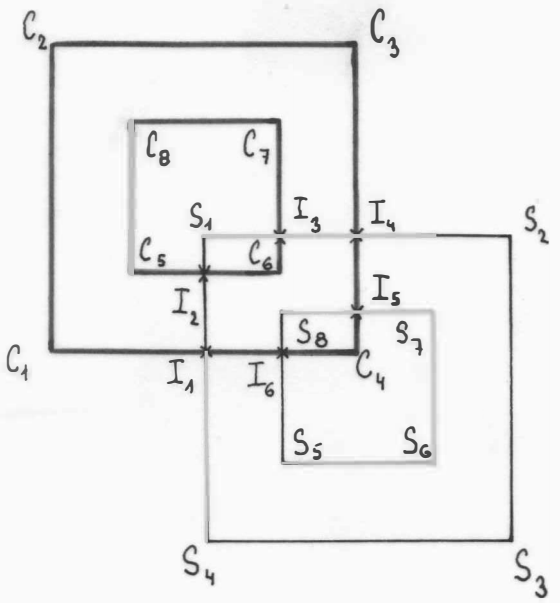
W-A algoritmus vyžaduje, aby n-úhelníky byly zadány pomocí seznamu vrcholů ve směru hodinových ručiček, přičemž vrcholy děr je nutné zadat ve směru opačném, tj. předpokládá se, že vnitřek n-úhelníka je vždy vpravo od orientované hrany, nebo naopak. Pro názornost uvažme tři jednoduché charakteristické možnosti, viz obr.6.11.2.



Jednoduchý nekonvexní n-úhelník



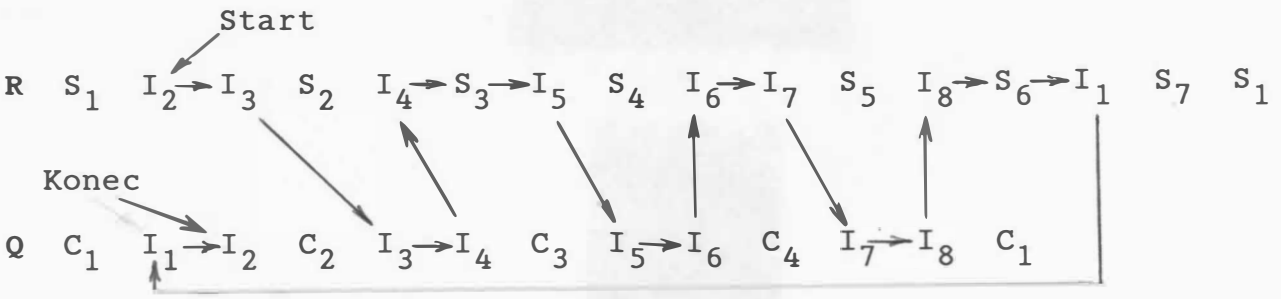
n-úhelník obklopující ořezávací n-úhelník



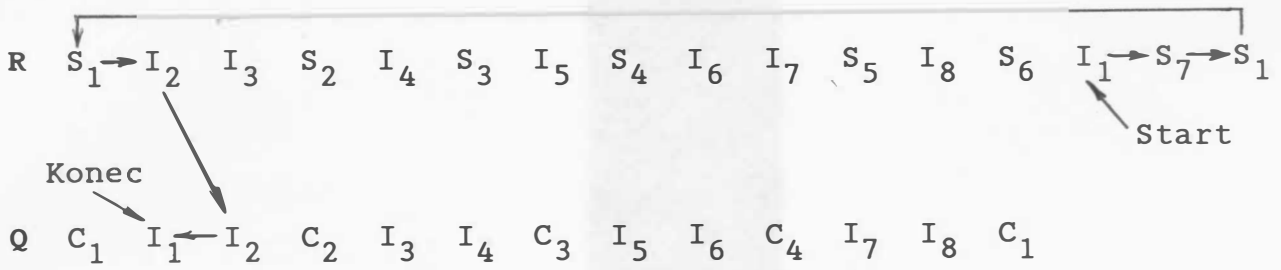
n-úhelník s dírou

Obr.6.11.2

Uvažme nejdříve případ z obr.6.11.2.a. Průsečíky jednotlivých hran n-úhelníků označené symbolem I_k jsou zařazeny do seznamu vrcholů jednotlivých n-úhelníků tak, abychom neporušili jejich orientaci, přičemž je nutné vytvořit vzájemné odkazy jednotlivých průsečíků I_k v seznamu vrcholů. Označíme-li **R** uspořádaný seznam vrcholů S ořezávaného n-úhelníka a průsečíků I_k, **Q** uspořádaný seznam vrcholů C n-úhelníka tvořícího oblast ořezávání a průsečíků I_k, pak dostáváme dva seznamy s příslušnými odkazy, viz obr.6.11.3.a.



a) Výsledný n-úhelník



b) Jedna z odříznutých částí původního n-úhelníka

Obr.6.11.3

Zároveň je nutné vytvořit seznam vstupních průsečíků I_2, I_4, I_6 a I_8, tj. bodů, kde hrana ořezávaného n-úhelníka vstupuje do n-úhelníka definujícího oblast ořezávání, a seznam výstupních průsečíků I_1, I_3, I_5 a I_7, tj. bodů, kde hrana ořezávaného n-úhelníka vystupuje z n-úhelníka definujícího oblast ořezávání. K vytvoření n-úhelníka, který vznikne oříznutím, je nutné začít od průsečíku, který je ve vstupním seznamu, a procházet seznamy **R** a **Q**, jak je naznačeno na obr.6.11.3.a. Výsledkem je seznam vrcholů:

I_2 I_3 I_4 S_3 I_5 I_6 I_7 I_8 S_6 I_1 I_2

V případě, že proces je započat od jiného vstupního průsečíku, je výsledek stejný (vrcholy jsou pouze cyklicky zaměněny). Průsečíky I_2, I_4, I_6 a I_8 musejí být odstraněny ze seznamu vstupních průsečíků, tj. seznam je nyní prázdný.

K vytvoření n-úhelníků zbylých po ořezávání je nutné začít od průsečíků z výstupního seznamu, tj. např. od průsečíku I_1 s tím, že seznam Q je prohlížen v opačném směru, viz obr.6.11.3.b. Výsledkem je pak seznam

$$I_1 \quad S_7 \quad S_1 \quad I_2 \quad I_1$$

přičemž průsečík I_1 musí být odstraněn z výstupního seznamu. Začneme-li od I_3, I_5 nebo I_7, dostáváme seznamy

$$I_3 \quad S_2 \quad I_4 \quad I_3 \qquad I_5 \quad S_4 \quad I_6 \quad I_5 \qquad I_7 \quad S_5 \quad I_8 \quad I_7$$

přičemž je opět nutné odstranit příslušné průsečíky z výstupního seznamu.

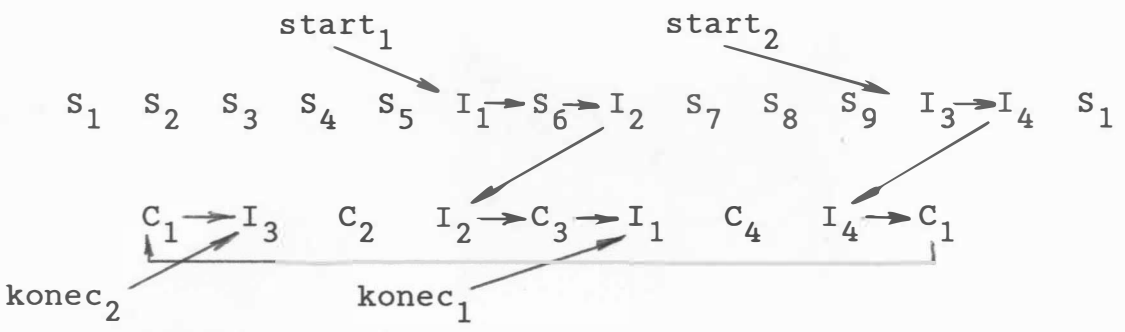
Poněkud složitější situace nastává v případě, který je uveden na obr.6.11.2.b, kdy výsledkem oříznutí n-úhelníka jsou dva výsledné n-úhelníky. Postup je stejný jako v předešlém případě, viz obr.6.11.4.a, s tím, že po vytvoření prvního seznamu

$$I_1 \quad S_6 \quad I_2 \quad C_3 \quad I_1$$

a odstranění průsečíku I_1 ze vstupního seznamu je nutné ještě vytvořit seznam odpovídající vstupnímu bodu I_3

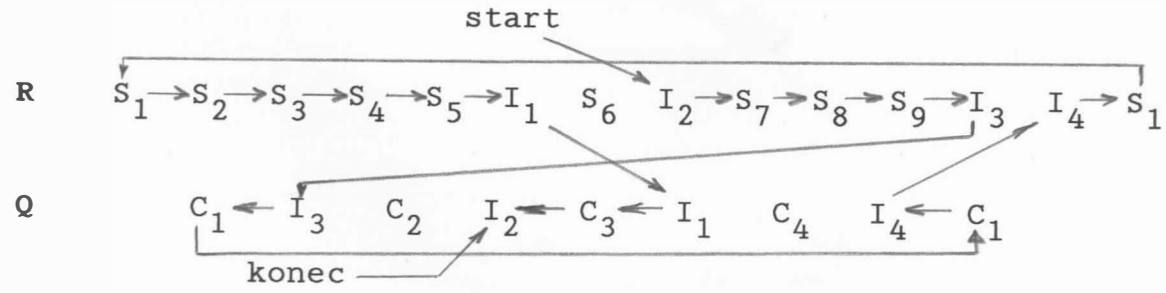
$$I_3 \quad I_4 \quad C_1 \quad I_3$$

a vypuštění průsečíku I_3 ze vstupního seznamu.



vstupní seznam: I_1, I_3 výstupní seznam: I_2, I_4

a) výsledné n-úhelníky



vstupní seznam: I_1, I_3 výstupní seznam: I_2, I_4

b) zbylá část

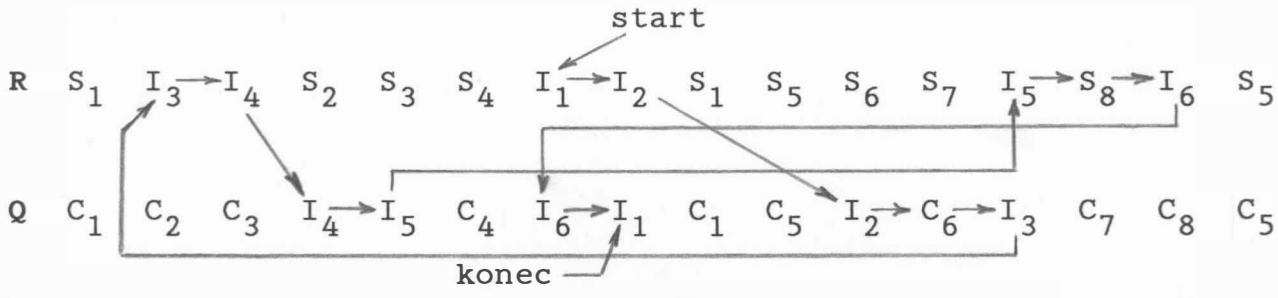
Obr. 6.11.4

Obdobným způsobem je vytvořen n-úhelník, který je zbylou částí původního n-úhelníka, s tím, že je nutné začít od libovolného prvku výstupního seznamu a seznam Q je prohlížen v opačném směru. Výsledkem je pak seznam:

I_2 S_7 S_8 S_9 I_3 C_1 I_4 S_1 S_2 S_3 S_4 S_5 I_1 C_3 I_2

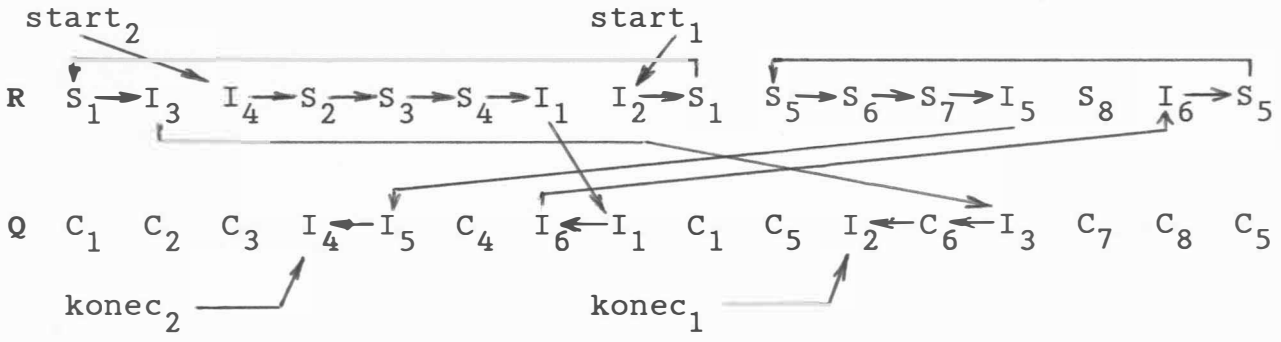
přičemž je nutné odstranit prvky I_2 a I_4 z výstupního seznamu, tj. seznam je nyní prázdný.

Uvažme nyní poslední případ z obr. 6.11.2, kdy ořezávaný n-úhelník obsahuje díru a oblast ořezávání je tvořena též n-úhelníkem s dírou. Seznamy vytvoříme podobným způsobem jako v předchozích případech, viz obr. 6.11.5.



vstupní seznam : I_1, I_3 , I_5 výstupní seznam: I_2, I_4, I_6

a) výsledný n-úhelník



vstupní seznam : I_1, I_3, I_5 výstupní seznam: I_2, I_4, I_6

b) zbylé části

Obr.6.11.5

Postup je opět obdobný předchozím případům s tím rozdílem, že pro každý n-úhelník, který tvoří vnitřní či vnější hranici, je nutné vytvořit kruhový seznam. Je-li průsečík I_1 vzat jako první bod, pak výsledný seznam definující výsledek oříznutí je

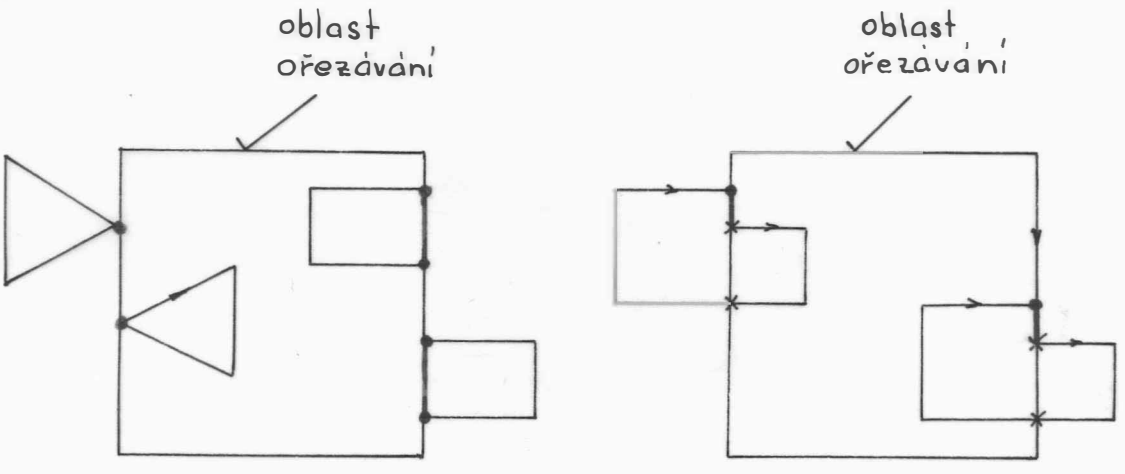
I_1 I_2 C_6 I_3 I_4 I_5 S_8 I_6 I_1

a zbytek po oříznutí je definován seznamy

I_2 S_1 I_3 C_6 I_2

a

I_4 S_2 S_3 S_4 I_1 I_6 S_5 S_6 S_7 I_5 I_4

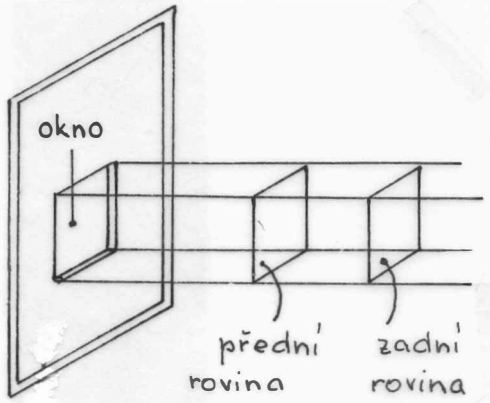


Obr.6.11.6

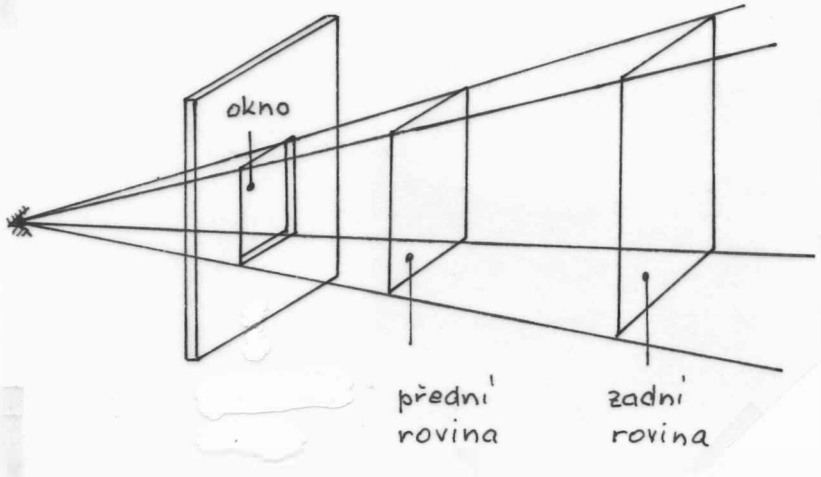
Je pochopitelné, že v případě úplné specifikace W-A algoritmu je nutné též řešit speciální případy, kdy se zadané n-úhelníky dotknou buď vrcholem, nebo hranami, viz obr.6.11.6. Do seznamů se pak zařadí jen ty průsečíky, které jsou označené *, zatímco průsečíky označené ◦ se nezařadí, podobně jako v kap.6.7.

Nevýhodou uvedených algoritmů je nutnost orientace hran n-úhelníků, což může znamenat částečné omezení v praktických aplikacích. Orientaci je pak možné určit tak, že se např. vyhledá vrchol, který je nejvíce vpravo a podle hodnoty souřadnice y následujícího vrcholu se určí orientace n-úhelníka. V případě, že n-úhelník je konvexní, lze rozhodnout o orientaci na základě vektorového součinu směrových vektorů dvou po sobě jdoucích hran, které neleží na stejné přímce.

Jistou výhodou W-A algoritmu je, že jej lze modifikovat i pro případ použití kruhových oblouků, případně i jiných kvadratických křivek.



paralelní projekce



perspektivní projekce

Obr.6.12.1

6.12 Ořezávání v třírozměrném prostoru

Dosud uvedené algoritmy řešily problematiku ořezávání úseček či n-úhelníka obecně vůči rovinné ořezávací oblasti. Při zobrazování třírozměrných objektů je nutné používat modifikované ořezávání v prostoru z několika důvodů, a to:

- objekt by mohl přesáhnout plochu, na kterou jej chceme zobrazovat (podobně jako při ořezávání v rovině),
- objekty či jejich části, které jsou mimo zobrazovanou oblast tzv. pyramidu pohledu, by při projekci vytvářely nežádoucí efekty.

Z těchto důvodů bylo zavedeno ořezávání vůči prostorové oblasti, která v případě používání perspektivní projekce má tvar pyramidy, viz obr.6.12.1.b. Pyramida pak reprezentuje viditelnou oblast pro pozorovatele. Obvykle je přední rovina ořezávání oblasti v pozici pozorovatele a zadní rovina pak nekonečně daleko. Vzhledem k tomu, že okno může být různě velké, viz obr.6.12.2, je vhodné nejdříve transformovat souřadnice tak, že ořezávání probíhá v "jednotkové" pyramidě, a poté aplikovat inverzní transformaci souřadnic.

Jednotková pyramida je pak dána polorovinami:

$$z \geq x \qquad z \leq -x \qquad z \geq y \qquad z \leq -y \qquad z \geq 0$$

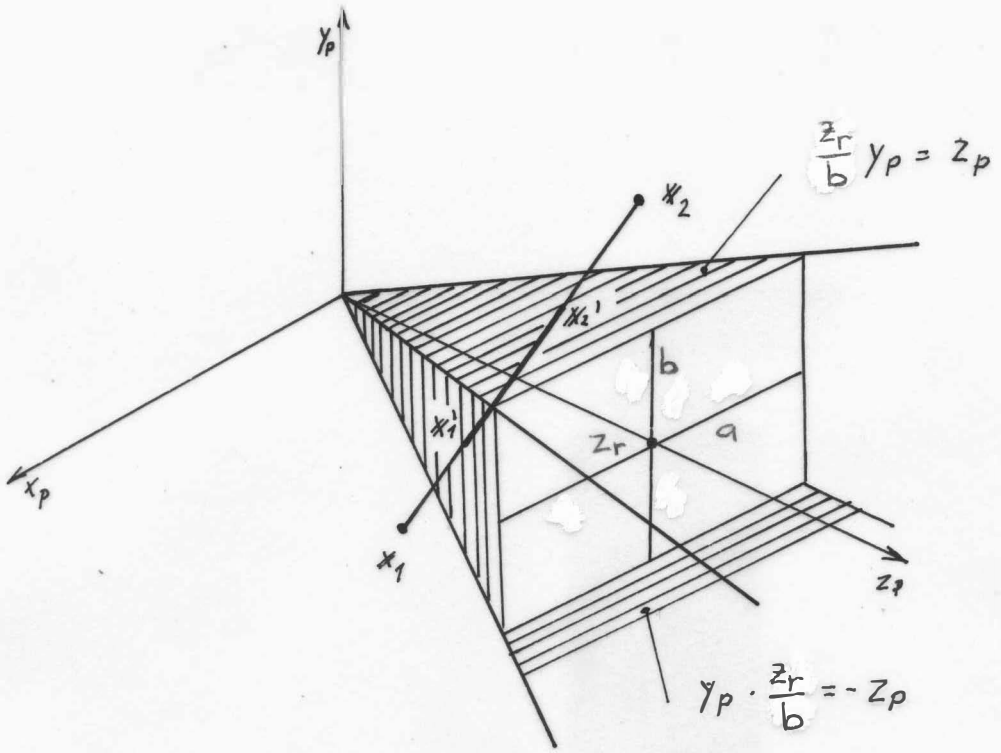
Celý proces ořezávání pak může být vyjádřen pomocí transformace

$$\mathbf{x}' = \mathbf{U}^{-1} \,.\, \mathbf{Clip} \,.\, \mathbf{U} \,.\, \mathbf{x}$$

přičemž operace změny měřítka je reprezentována maticí **U** tak, že

$$\mathbf{U} = \begin{bmatrix} z_r/a & 0 & 0 & 0 \\ 0 & z_r/b & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

kde z_r určuje polohu promítací roviny na ose z,
2a, 2b jsou pak rozměry okna.



Obr. 6.12.2

Operátor **Clip** je pak popsán algoritmem 6.12.3 a je vlastně modifikací Cohen-Sutherlandova algoritmu, viz kap. 6.2. Bity kódovací oblasti mají následující význam:

bit 0 - bod je vlevo od pyramidy, tj. x < -z
bit 1 - bod je vpravo od pyramidy, tj. x > z
bit 2 - bod je pod pyramidou, tj. y < -z
bit 3 - bod je nad pyramidou, tj. y > z

Je zřejmé, že na rozdíl od algoritmu z kap. 6.2 musí být též určena odpovídající hodnota souřadnic y a z.

```
procedure CLIP3D ( x1, y1, x2, y2, z1, z2: real );
label 99;
var x, y, z, t: real;
    c, c1,c2: integer;
```

```
procedure CODE3 ( x, y, z: real; var c: integer );
begin c := 0; { hodnoty 1,2,4,8 odpovídají příslušným kódům }
      if x < -z then c:=1 else if x > z then c:=2;
      if y < -z then c:=c+4 else if y > z then c:=c+8;
end { CODE3 };
begin { zjištění stavových slov }
  CODE3(x1,y1,z1,c1);  CODE3(x2,y2,z2,c2);
  if (c1 land c2) = 0 then
  begin { úsečka může procházet pyramidou }
    if (c1 lor c2) <> 0 then
    begin { úsečka neleží celá v pyramidě }
      repeat
        if c1 = 0 then c:=c2 else c:=c1;
        if (c land 1) <> 0 then
        begin { bod je vlevo }
          t:=(z1+x1)/((x1-x2)-(z2-z1));
          z:=t*(z2-z1)+z1;  x:=-z;   y:=t*(y2-y1)+y1
        end
        else if (c land 2) <> 0 then
             begin { bod je vpravo }
               t:=(z1-x1)/((x2-x1)-(z2-z1));
               z:=t*(z2-z1)+z1;  x:=z;  y:=t*(y2-y1)+y1
             end
             else if (c land 4) <> 0 then
                  begin { bod je pod }
                    t:=(z1+y1)/((y1-y2)-(z2-z1));
                    z:=t*(z2-z1)+z1;
                    x:=t*(x2-x1)+x1;  y:=-z
                  end
                  else if (c land 8) <> 0 then
                       begin { bod je nad }
                         t:=(z1-y1)/((y2-y1)-(z2-z1));
                         z:=t*(z2-z1)+z1;
                         x:=t*(x2-x1)+z1;  y:=z
                       end;
        if c=c1 then
          begin x1:=x; y1:=y; z1:=z;
                CODE3(x1,y1,z1,c1)
          end
```

```
        else
          begin x2:=x; y2:=y; z2:=z;
                CODE3(x2,y2,z2,c2)
          end;
        if (c1 land c2)<>0 then goto 99 { úsečka leží vně }
      until (c1 lor c2)
    end;
    { nyní je nutné úsečku zobrazit }
    LINE 3D(x1,y1,z1,x2,y2,z2)
  end;
99:
end { CLIP3D } ;
```

Algoritmus 6.12.1

Příklad

Uvažme příklad z kap. 5.3 pro případy, kdy je nutné použít "pyramidální" ořezávání. Předpokládejme, že výstup bude prováděn na výstupním zařízení s rozlišovací schopností 1024 x 1024 a s počátkem v levém dolním rohu. Na výstup se budeme dívat ze vzdálenosti 1000 mm, přičemž velikost výstupu je 500 x 500 mm. Pak na všechny body scény je nutné aplikovat jednotlivé již uvedené transformace kromě perspektivní transformace, která je nahrazena "pyramidálním" ořezáváním a následnou perspektivní projekcí. Celková transformace může být vyjádřena takto

$$\mathbf{W} = \mathbf{Persp} \cdot \mathbf{U}^{-1} \cdot \mathbf{Clip} \cdot \mathbf{U} \cdot \mathbf{R}_{z_e y_e}(\varphi) \cdot \mathbf{R}_{x_e z_e}(\vartheta) \cdot \mathbf{Z} \cdot \mathbf{T}$$

Pak matice **U** a $\mathbf{U}^{-1}$ mají v našem případě tvar

$$\mathbf{U} = \begin{bmatrix} 4 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{U}^{-1} = \begin{bmatrix} 0.25 & 0 & 0 & 0 \\ 0 & 0.25 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Operátor **Clip** označuje operaci třírozměrného pyramidálního ořezávání. Takto získané souřadnice koncových bodů je obecně nutné pronásobit maticí $\mathbf{U}^{-1}$. Získané body $\mathbf{x}_c$ jsou uvnitř nebo na povrchu pohledové pyramidy. Nyní je nezbytné aplikovat operátor **Persp**, tj. určit souřadnice bodů po jejich perspektivní projekci. Pro souřadnice bodů na promítací rovině lze psát

$$x_p = \frac{z_r}{a} \frac{x_c}{z_c} r_x + x_0$$

$$y_p = \frac{z_r}{b} \frac{y_c}{z_c} r_y + y_0$$

kde (x_0, y_0) je požadovaná poloha počátku v souřadném systému zařízení

$2r_x$, resp. $2r_y$, je rozlišení ve směru osy **x** , resp. osy y

Lze ukázat, že operaci $\mathbf{U}^{-1}$ a **Persp** lze výhodně sloučit, neboť zlomek $\frac{z_r}{a}$, resp. $\frac{z_r}{b}$, v operaci **Persp** a $\frac{a}{z_r}$, resp. $\frac{b}{z_r}$, v matici $\mathbf{U}^{-1}$ se krátí. Pak lze psát

$$x_p = \frac{x_c}{z_c} r_x + x_0$$

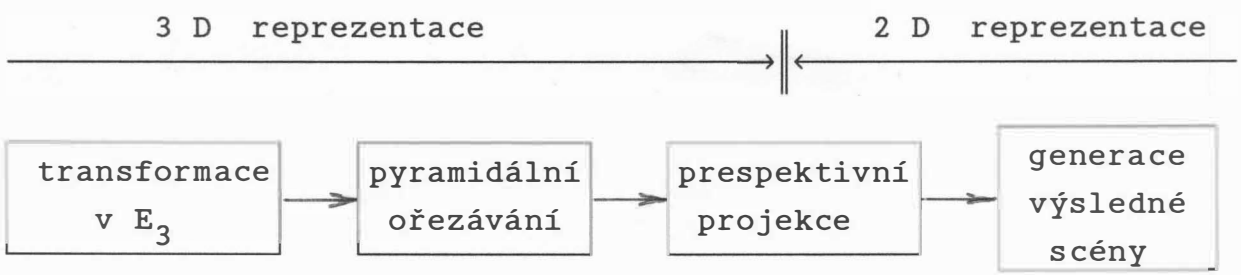
$$y_p = \frac{y_c}{z_c} r_y + y_0$$

kde x_c, y_c, z_c jsou souřadnice bodů $\mathbf{x}_c$ získaných jako přímý výstup třírozměrného ořezávání, tj. bez pronásobení maticí $\mathbf{U}^{-1}$. V našem případě dostáváme

$$x_p = \frac{x_c}{z_c} 511.5 + 511.5$$

$$y_p = \frac{y_c}{z_c} 511.5 + 511.5$$

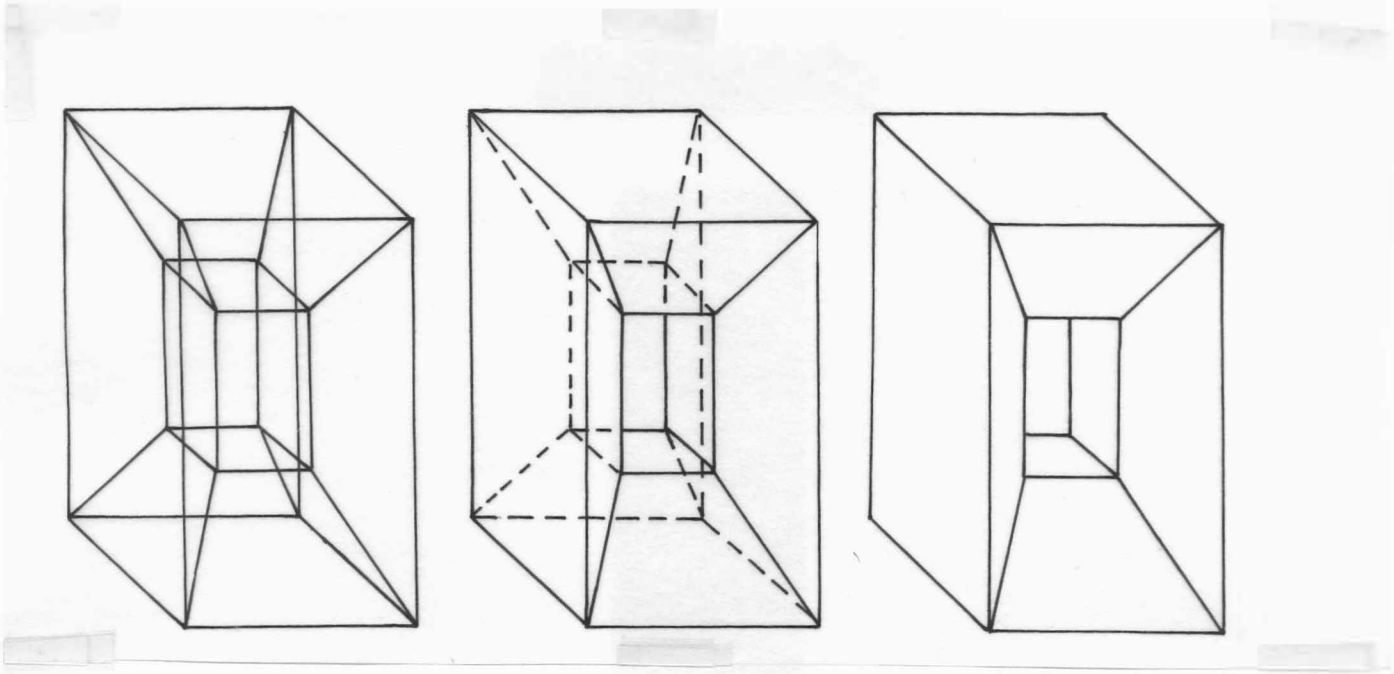
Celý postup řešení obecného pohledu v třírozměrném prostoru lze znázornit obr. 6.12.3.



Obr. 6.12.3

7. Eliminace neviditelných hran a povrchů

Problém řešení viditelnosti je jedním z nejobtížnějších problémů v počítačové grafice. V zásadě jde o určení bodů, částí hran a povrchů, které jsou neviditelné (zakryté jinými plochami) z dané pozice pozorovatele. Potřeba vypuštění neviditelných hran je zřejmá z obr.7.1, kde je uvedena typická ukázka tzv. drátěného modelu, jehož interpretace není jednoznačná.



Obr.7.1

Složitost problému eliminace neviditelných hran a povrchů vyústila v nepřeberné množství algoritmů, které používají různé principy a jejich kombinace. Pravděpodobně neexistuje nejlepší algoritmus. U algoritmů používaných v aplikacích reálného času je neúprosným požadavkem poskytnout řešení vždy v rámci 1/50 nebo 1/30 sec., např. u simulátorů pilotáže letadel. Budeme-li však požadovat realističtější výstup s více podrobnostmi včetně stínování, průhlednosti a průsvitnosti objektů s odrazy atd., budou algoritmy pomalejší, často vyžadující mnoho minut až hodin k řešení daného problému. Existující postupy využívají též dalších dostupných informací k podstatnému zrychlení, např. při animaci a simulaci lze s výhodou využít toho, že zobrazené scény po sobě následující jsou si velmi podobné, apod.

7.1 Zobrazování explicitních funkcí dvou proměnných

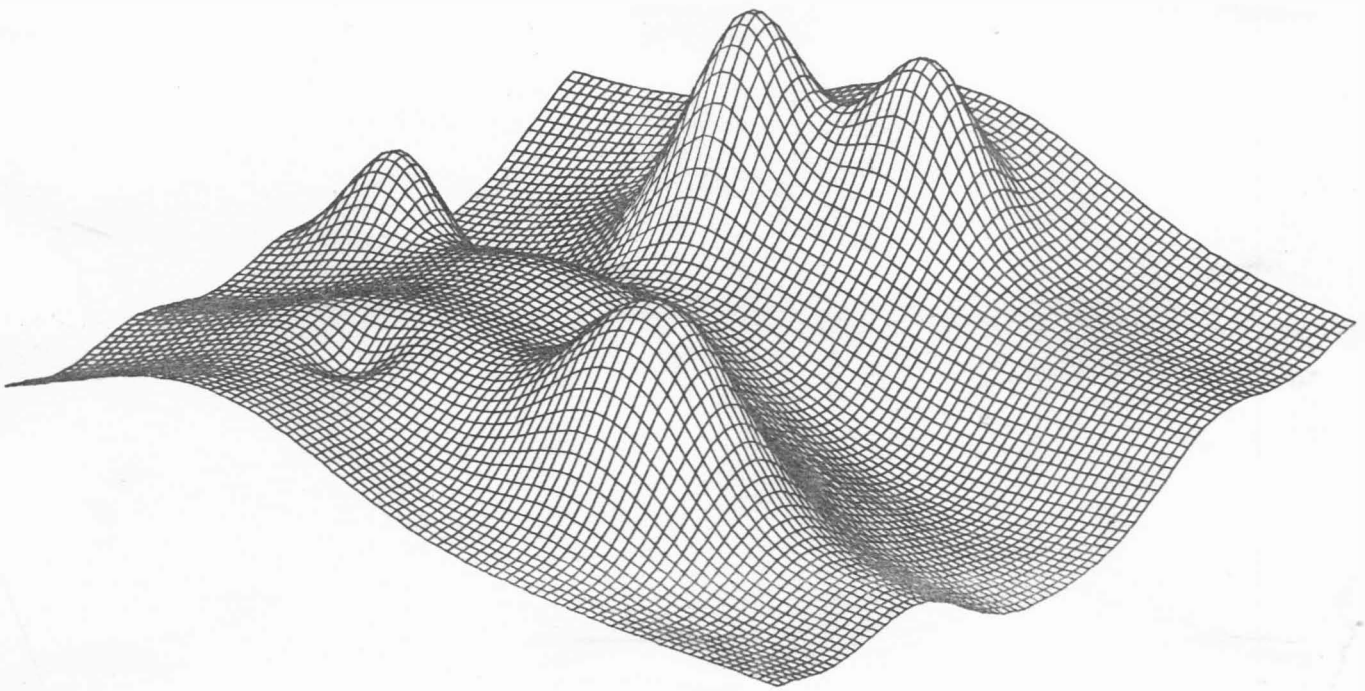
Výsledkem řešení mnoha problémů je často explicitní funkce dvou proměnných ve tvaru:

$$y = f(x, z) \qquad x \in \langle a_x, b_x \rangle \quad a \quad z \in \langle a_z, b_z \rangle$$

která bývá zadána buď přímo analytickým popisem nebo funkčními hodnotami v bodech sítě v rovině zx. Metody zobrazování explicitních funkcí dvou proměnných jsou většinou založeny na principu tzv. plovoucího horizontu [39],[116], neboť použití obecných metod vede k podstatnému prodloužení doby výpočtu.

Eliminace neviditelných hran

Pro mnoho aplikací je postačující zobrazení funkce dvou proměnných pomocí řezů vedených rovnoběžně s osou x, resp. s osou y, přičemž řezy vytvářejí obecně nepravidelnou obdélníkovou síť, viz obr.7.1.1.



Obr.7.1.1

Chceme tedy zobrazit křivky:

$$y = f (x , z_i) \qquad i = 1,\ldots,n$$

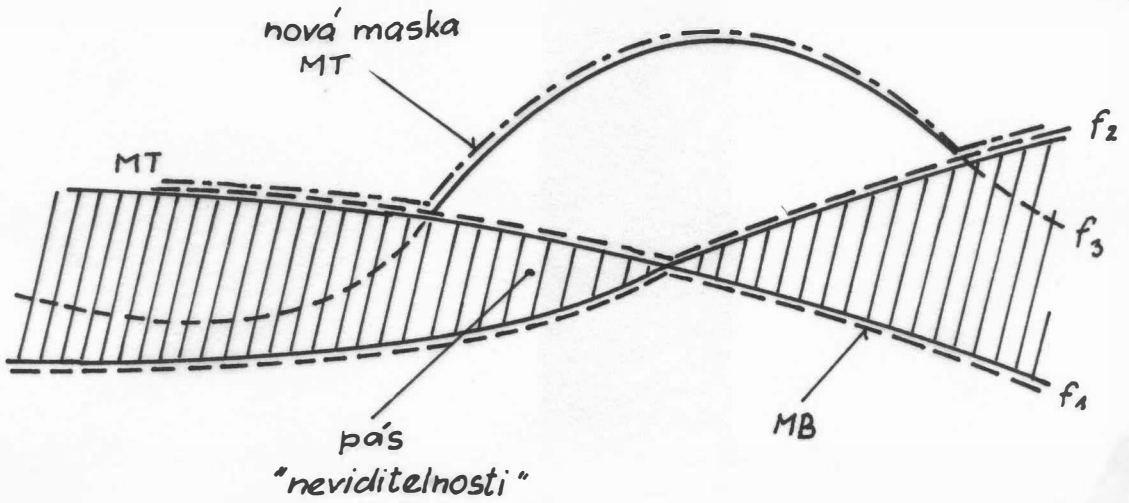
$$x \in \langle a_x , b_x \rangle \quad a \quad a_z = z_1 < z_2 < \ldots < z_{n-1} < z_n = b_z$$

a křivky:

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

$$z \in \langle a_z , b_z \rangle \quad a \quad a_x = x_1 < x_2 < \ldots < x_{n-1} < x_n = b_x$$

Princip řešení viditelnosti je velmi jednoduchý. Předpokládejme, že řezy jsou rovnoběžné s osou x, přičemž jsou kresleny postupně od pozorovatele směrem dozadu. Nakreslením prvních dvou řezů (ozn. f_1 a f_2) je zobrazena část plochy, která je ohraničena právě prvním řezem vpředu a druhým řezem vzadu, viz obr.7.1.2.



Obr.7.1.2

Bude-li nyní kreslen třetí řez (ozn. f_3), pak je zřejmé, že bude kreslena jen ta část, která je pod dolním nebo nad horním obrysem již zobrazené části plochy. Část řezu, která je mezi horním a dolním obrysem, je zakryta tou částí kreslené plochy, která je vymezena prvním a druhým řezem.

Označíme-li horní obrys jako MT a dolní obrys jako MB, přičemž budeme prozatím předpokládat, že jde o funkce, lze algoritmus pro kreslení řezů rovnoběžných s osou x realizovat algoritmem 7.1.1.

```
MT := -∞; MB := ∞;
for k:=1 to počet řezů do
begin Kresli funkci f(x,z_k) s respektováním
      viditelnosti     ∀x ∈ < a_x,b_x>;
      MT := max { MT , f(x,z_k) }          ∀x ∈ < a_x,b_x>
      MB := min { MB , f(x,z_k) }          ∀x ∈ < a_x,b_x>
      { max, min jsou funkce, jejichž argumenty jsou }
      {      funkce a výsledkem je opět funkce       }
end
```

Algoritmus 7.1.1

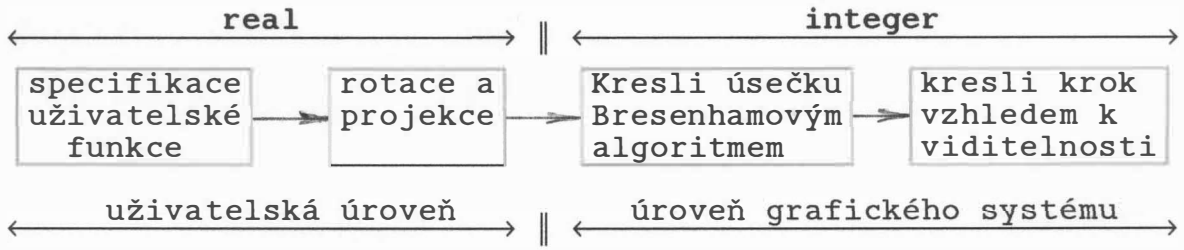
Funkce min a max je možné reprezentovat pomocí maskovacích vektorů, viz např. [140], reprezentujících vlastně hodnotu funkce pro danou hodnotu x.

Vzhledem k tomu, že funkce $f(x,z_k)$ bude kreslena jako posloupnost lineárních úseků vzhledem k proměnné x, je možné použít Bresenhamův algoritmus pro kreslení úsečky, viz kap.3. Aby nebyly kresleny neviditelné body dané čáry, je nutné modifikovat příkaz DRAW STEP tak, aby byly zobrazovány jen ty body, které jsou nad nebo pod již nakreslenou částí plochy. Příkaz DRAW STEP může být nyní realizován (zjednodušeně), např. algoritmem 7.1.2.

```
procedure DRAW STEP;
begin if yp < MB[xp] then
         begin MB[xp] := yp; PLOT ( xp , yp ); end;
      if yp > MT[xp] then
         begin MT[xp] := yp; PLOT ( xp , yp ); end;
end;
```

Algoritmus 7.1.2

Pro použití ve skutečných aplikacích, kdy se mohou vyskytnout téměř svislé úsečky, je nutné zavést pomocné maskovací vektory M1T a M1B, které reprezentují aktuální stav, přičemž maskovací vektory MT a MB vlastně reprezentují stav z předcházejícího řezu. Po nakreslení jednoho celého řezu je pak nutné přepsat obsah dočasných maskovacích vektorů M1T a M1B do maskovacích vektorů MT a MB. Celý postup kreslení funkce dvou proměnných je znázorněn na obr.7.1.3.



Obr. 7.1.3

Uživatelem specifikovanou funkci, ať už zadanou analytickým předpisem nebo daty, je možné otáčet v prostoru s tím omezením, že průmět vektoru původně rovnoběžného s osou y **musí** zůstat svislý, tj. lze aplikovat pouze rotace okolo osy y a osy x v uvedeném pořadí a změnu měřítka. Vzhledem k principu zobrazování lze použít paralelní projekci nebo "slabou" perspektivní projekci, při níž libovolná svislá přímka v prostoru E_3 na průmětně protíná zobrazenou plochu **pouze** v jednom bodě.

Až dosud byla předpokládána superpozice dvou obrázků vzniklých kreslením řezů rovnoběžných s osou x a řezů rovnoběžných s osou z. Při prosté superpozici však vznikají chyby, které působí rušivě, viz obr.7.1.4. Z tohoto důvodu je vhodné použít tzv. "Zig-zag" metodu postupného kreslení, při které chyby nevznikají. Na obr.7.1.5 je naznačen postup kreslení za předpokladu, že bod q je nejblíže pozorovateli. Celý postup kreslení funkcí dvou proměnných s eliminací neviditelných částí lze popsat algoritmem 7.1.3.

```
program HIDDEN;
const n = 45; { Počet řezů v ose x }
      m = 30; { v ose y - osa z je svislá v tomto programu ! }
      rsty = 199; { rozlišení průmětny na ose y }
      rstx = 639; { na ose x }
type mat = array [1..n,1..m] of real;
     vectx = array [1..n] of real;
      vecty = array [1..m] of real;
     rot = array [1..3,1..3] of real;
     mask = array [0..rstx] of integer;
     intmat = array [1..n,1..m] of integer;
```

```
var xp,yp: integer;
    fz: mat; { pole funkčních hodnot }
    x: vectx; { pole hodnot x }
    y: vecty; { pole hodnot y }
    MT,MB,MT1,MB1: mask;
    r: rot;
    xsc,ysc: intmat;
    ax,bx,ay,by: real;
    alfa,beta: real;
    sx,sy,isx,isy: integer;
function EXP MY (x: real): real;
begin if x < -10 then EXP MY:=0 else EXP MY:= exp (x)
      { toto je nezbytné z důvodů možného podtečení }
      {        funkce EXP v Turbo-Pascalu            }
end { EXP MY };
function FCE (px,py: real): real;
begin     FCE:=8* EXP MY (-40*(sqr(px-0.6)+sqr(py-0.3)))+
          6* EXP MY (-35*(2*sqr(px-0.7)+sqr(py+0.1)))+
          8.5* EXP MY (-15*(sqr(px+0.5)+sqr(py+0.3)))+
          5* EXP MY (-50*(sqr(px-0.1)+sqr(py-0.8)))-
          2* EXP MY (-90*(sqr(px+0.6)+sqr(py-0.4)))+
          5* EXP MY (-0.25*sqr(px+4*py-1))-
          4* EXP MY (-0.6*sqr(4*px-py-1))+0.4*cos(12*px)-
          2* sin(py)+15
end { FCE };
function SIGN ( s: integer ): integer;
begin if s>0 then SIGN:=1
             else if s<0 then SIGN:=-1
                         else SIGN:=0
end { SIGN };
function MIN ( aa, bb: integer ): integer;
begin if aa<=bb then MIN:=aa else MIN:=bb
end { MIN };
function MIN REAL ( aa, bb: real ): real;
begin if aa<=bb then MIN REAL:=aa else MIN REAL:=bb
end { MIN REAL };
function MAX REAL ( aa, bb: real ): real;
begin if aa>=bb then MAX REAL:=aa else MAX REAL:=bb
end { MAX REAL };
```

```
procedure INIT MASK;
var k: integer;
begin for k:=0 to rstx do
      begin MT1[k]:= -maxint; MB1[k]:= maxint end
end { INIT MASK };
procedure SET MASK;
var k: integer;
begin MT:=MT1;  MB:=MB1
end { SET MASK };

procedure DRAW TO ( x,y: integer);
var u,v,ix,iy: integer;
procedure DRAW STEP;
begin if ( yp < MB[xp] ) or ( yp > MT[xp] ) then
      begin plot (xp,yp,1);
            if yp > MT1[xp] then MT1[xp]:=yp;
            if yp < MB1[xp] then MB1[xp]:=yp
            { nastavení masek }
      end
end { DRAW STEP };
procedure BRESENHAM;
var a,b,j,d: integer;
begin if u > v then
      begin { 1.,4.,5.,8. oktant }
            a:=2*v;
            d:=a-u;
            b:=d-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. oktant }
             a:=2*u;
             d:=a-v;
             b:=d-v;
```

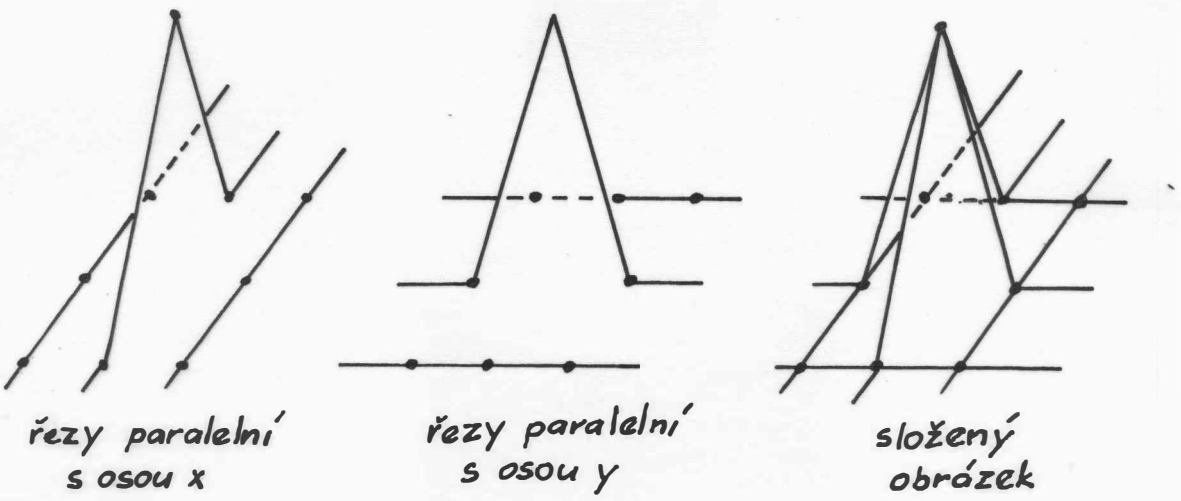
```
            for j:=1 to v do
            begin if d < 0 then d:=d+a
                           else begin xp:=xp+ix; d:=d+b end;
                  yp:=yp+iy;
                  DRAW STEP
            end
       end
end { BRESENHAM };
begin ix:=SIGN(x-xp); iy:=SIGN(y-yp);
      u:=abs(x-xp);   v:=abs(y-yp);
      BRESENHAM
end { DRAW TO };
procedure SET TRANSFORMATION ( alfa,beta: real; var r: rot );
var cosa,sina,cosb,sinb: real;
begin cosa:=cos(alfa);
      cosb:=cos(beta); sina:=sin(alfa); sinb:=sin(beta);
      r[1,1]:=cosa;      r[1,2]:=-sina;     r[1,3]:=0;
      r[2,1]:=sina*cosb; r[2,2]:=cosa*cosb; r[2,3]:=-sinb;
      r[3,1]:=sina*sinb; r[3,2]:=cosa*sinb; r[3,3]:=cosb
end { SET TRANSFORMATION };
procedure TRANSFORM;
var i,j: integer;
    xmin,xmax,zmin,zmax,qx,qz: real;
    px,pz: array [1..n,1..m] of real;
begin
   for i:=1 to n do
      for j:=1 to m do
      begin px[i,j]:=r[1,1]*x[i]+r[1,2]*y[j];
            pz[i,j]:=r[3,1]*x[i]+r[3,2]*y[j]+r[3,3]*fz[i,j];
      end;
      xmin:=px[1,1]; xmax:=xmin;
      zmin:=pz[1,1]; zmax:=zmin;
      for i:=1 to n do
         for j:=1 to m do
         begin xmax:= MAX REAL ( xmax,px[i,j] );
               xmin:= MIN REAL ( xmin,px[i,j] );
               zmax:= MAX REAL ( zmax,pz[i,j] );
               zmin:= MIN REAL ( zmin,pz[i,j] )
         end;
```

```
      qx:=rstx/(xmax-xmin);
      qz:=rsty/(zmax-zmin);
      for i:=1 to n do
         for j:=1 to m do
         begin xsc[i,j]:=round((px[i,j]-xmin)*qx);
               ysc[i,j]:=round((pz[i,j]-zmin)*qz);
         end
end { TRANSFORM };

procedure ZIG ZAG (sx,sy,isx,isy,nol: integer );
var n1: integer;
begin xp:=xsc[sx,sy];
      yp:=ysc[sx,sy];
      for n1:=1 to nol do
      begin sx:=sx+isx;
            DRAW TO (xsc[sx,sy],ysc[sx,sy]);
            sy:=sy-isy;
            DRAW TO (xsc[sx,sy],ysc[sx,sy])
      end
end { ZIG ZAG };

procedure FIND VERTEX ( r: rot; var isx,isy,sx,sy: integer );
const xr : array [1..4] of real = (-1.,1.,1.,-1);
      yr : array [1..4] of real = (-1.,-1.,1.,1);
var a,b: real;
    k,l: integer;
begin a:=1e20;
      for l:=1 to 4 do
      begin b:=r[2,1]*xr[l]+r[2,2]*yr[l];
            if b<a then begin a:=b; k:=l end
      end;
      case k of
1:    begin isx:=1; isy:=1; sx:=n; sy:=1 end;
2:    begin isx:=-1; isy:=1; sx:=1; sy:=1 end;
3:    begin isx:=-1; isy:=-1; sx:=1; sy:=m end;
4:    begin isx:=1; isy:=-1; sx:=n; sy:=m end;
      end
end { FIND VERTEX };
```

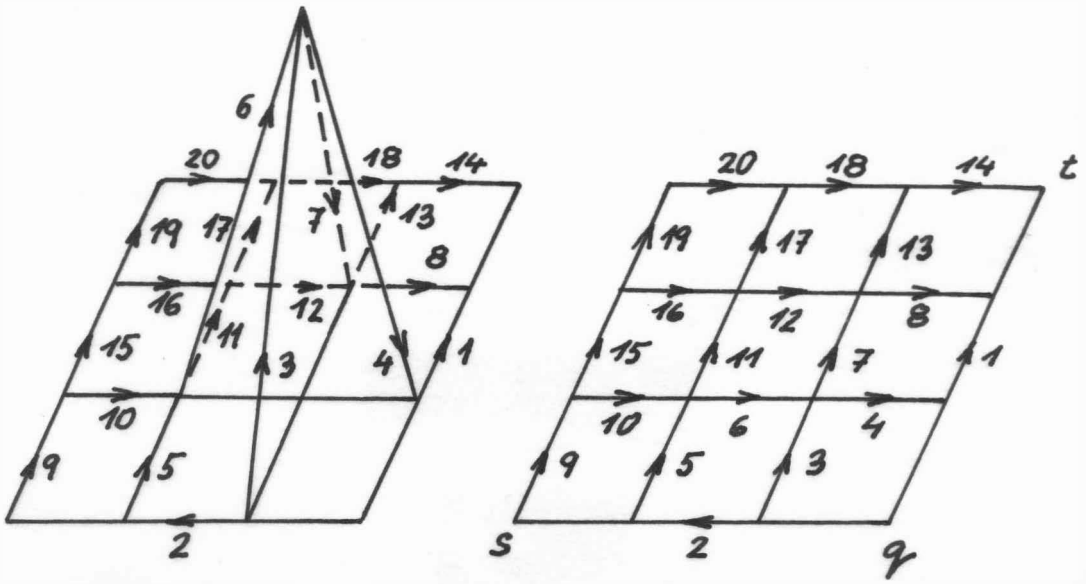
```
procedure COMPUTE OR LOAD (var x: vectx; var y: vecty;
                           var fz: mat);
{ procedura musí být změněna v případě čtení dat z disku }
var dx,dy: real;
    i,j: integer;
begin dx:=(bx-ax)/(n-1);
      dy:=(by-ay)/(m-1);
      for i:=1 to n do
          x[i]:=ax+dx*(i-1);
      for j:=1 to m do
          y[j]:=ay+dy*(j-1);
      for i:=1 to n do
          for j:=1 to m do
              fz[i,j]:= FCE ( x[i] ,y[j] );
end { COMPUTE OR LOAD };

procedure HIDDEN LINE ( isx,isy,sx,sy: integer;
                        xsc,ysc: intmat);
var i,j,nol,sxx,syy: integer;
begin INIT MASK;
      xp:=xsc[sx,sy];
      yp:=ysc[sx,sy];
      SET MASK;
      for i:=2 to n do
      begin sx:=sx-isx;
           DRAW TO ( xsc[sx,sy], ysc[sx,sy] )
      end;
      sxx:=sx;
      syy:=sy;
      for j:=2 to m do
      begin sy:=sy+isy;
            DRAW TO ( xsc[sx,sy], ysc[sx,sy] )
      end;
      for j:=2 to m do
      begin syy:=syy+isy;
            nol:= MIN (j-1,n-1);
            SET MASK;
            ZIG ZAG ( sxx,syy,isx,isy,nol )
      end;
```

```
      for i:=3 to n do
      begin sxx:=sxx+isx;
            nol:= MIN (n-i+1,m-1);
            SET MASK;
            ZIG ZAG ( sxx,syy,isx,isy,nol )
      end
end { HIDDEN LINE };
begin HiRes;
      gotoxy(10,10); writeln(' Výpočet funkčních hodnot');
      ax:=-1; bx:=1; ay:=-1; by:=1;
      { implicitní rozsah pro x a y souřadnice }
      COMPUTE OR LOAD (x,y,fz);
      gotoxy(10,12); { alfa - okolo osy z; beta - okolo osy x }
      writeln(' Zadej úhel alfa a beta ve stupních ');
      readln(alfa,beta); alfa:=alfa/57.296; beta:=beta/57.296;
      SET TRANSFORMATION (alfa,beta,r);
      writeln(' Transformace souřadnic ');
      TRANSFORM;
      FIND VERTEX ( r,isx,isy,sx,sy ); HiRes;
      HIDDEN LINE(isx,isy,sx,sy,xsc,ysc)
end.
```

Algoritmus 7.1.3



Obr. 7.1.4

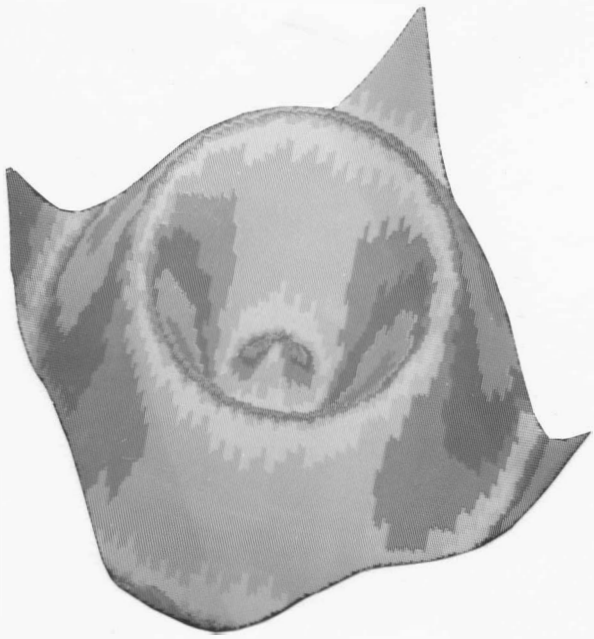


Obr. 7.1.5

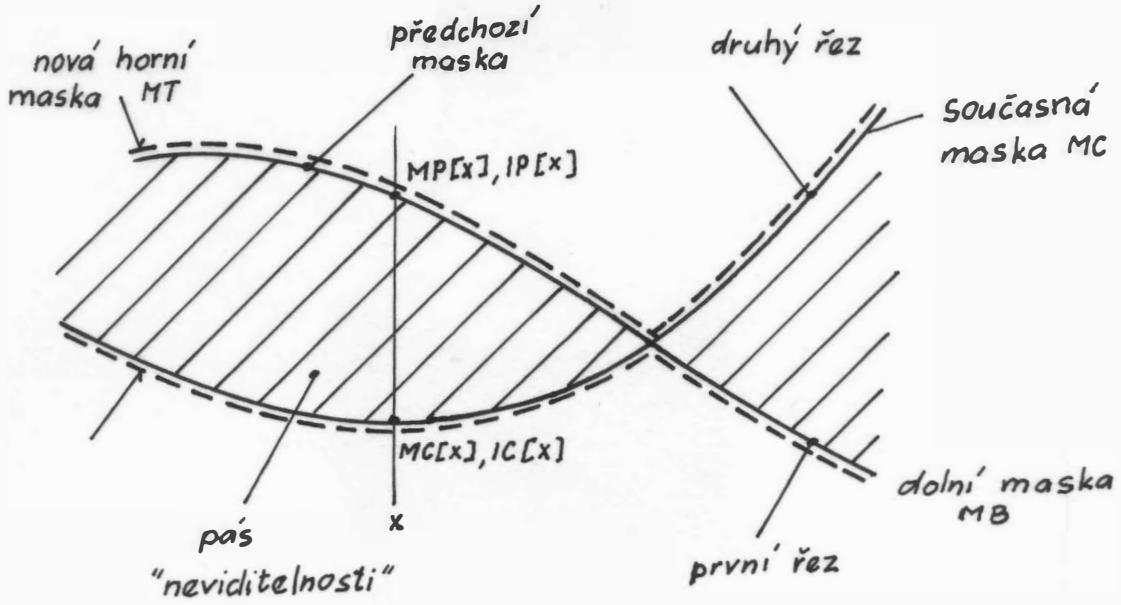
Eliminace neviditelných povrchů

Problém eliminace neviditelných povrchů, viz obr. 7.1.6., může být řešen mnoha způsoby. Pro jednoduchost lze opět předpokládat, že budou kresleny řezy funkce paralelní s osou x, přičemž kromě funkčních hodnot budou zadány i intenzity jasu v jednotlivých bodech (intenzitu lze určit např. jako funkci úhlu mezi normálou plochy a pozorovatelem), viz obr. 7.1.7. Řezy se budou opět kreslit postupně od pozorovatele směrem dozadu. Kromě funkcí MT a MB reprezentujících horní a dolní obrys je nutné zavést funkci IC průběhu intenzity právě zpracovávaného řezu a funkci průběhu intenzity předchozího řezu IP. Pak řešení problému eliminace neviditelných povrchů lze popsat např. algoritmem 7.1.4.

Nutným předpokladem k celkové realizaci uvedeného algoritmu je, že algoritmus pro generaci úsečky musí nejen poskytovat souřadnice bodů, které se mají aktivovat, ale též i jejich jas. Je zřejmé, že je nutné změnit odpovídajícím způsobem podprogram DRAW STEP, viz algoritmus 7.1.5.



Obr. 7.1.6



Obr. 7.1.7

```
MT := -∞; MB := ∞;
for k:=1 to počet řezů do
begin
      Kresli funkci f(x,z_k) s respektováním viditelnosti
      ∀x ∈ < a_x,b_x> a ∀x kresli úsečku z (x,MC[x],IC[x])
      do (x,MP[x],IP[x]) vzhledem k viditelnosti, přičemž
      interpoluj intenzitu lineárně z IC[x] do IP[x];
      MT := max { MT , f(x,z_k) }        ∀x ∈ < a_x,b_x>
      MB := min { MB , f(x,z_k) }        ∀x ∈ < a_x,b_x>
      { max, min jsou funkce, jejichž argumenty jsou }
      { funkce a výsledkem je opět funkce }
      IP := IC; MP := f(x,z_k)           ∀x ∈ < a_x,b_x>
      { nastavení masek na konci řezu }
end
```

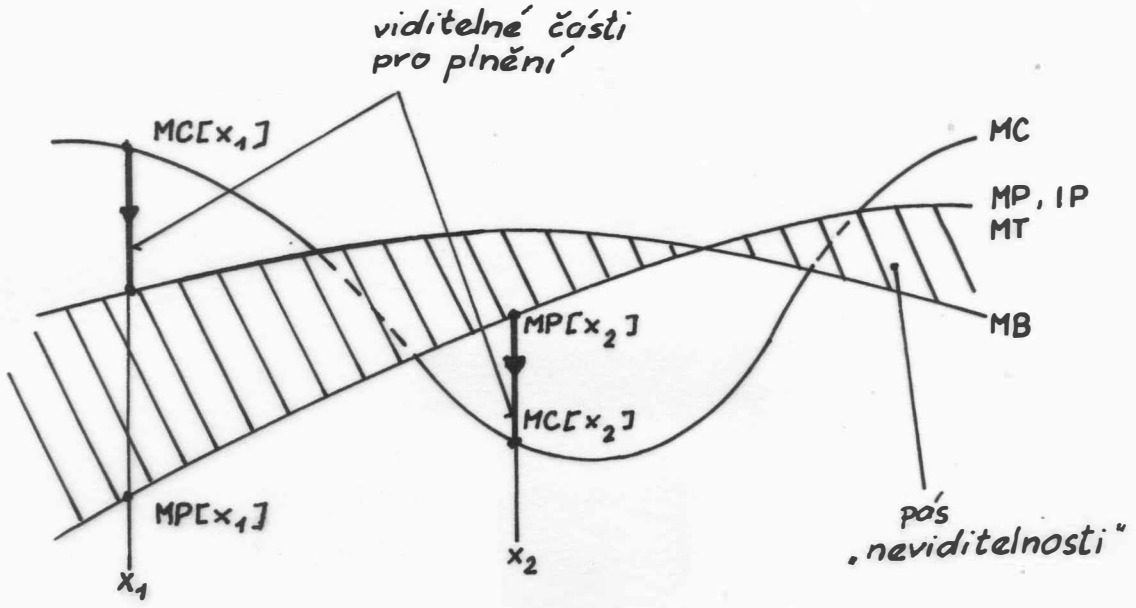
Algoritmus 7.1.4

Na rozdíl od většiny publikovaných algoritmů je uvedený algoritmus jednoduchý a poměrně snadno realizovatelný.

Je vhodné poznamenat, že procedura FILL je vlastně Bresenhamův algoritmus, který pro dané xp a jednotlivé hodnoty yp určí hodnoty intenzity, se kterou se aktivuje odpovídající bod. Po nakreslení celého řezu je nyní nutné též aktualizovat vektory MP a IP, kterými jsou reprezentovány odpovídající funkce.

```
procedure DRAW STEP;
begin if yp < M1B[xp] then M1B[xp]:=yp;
      if yp > M1T[xp] then M1T[xp]:=yp
      if (yp < MB[xp]) or (yp > MT[xp])
      then FILL(xp,yp,MP[xp],i,IP[xp]);
        { z (xp,yp) do (xp,MP[xp]) aktivuj body }
        { s intenzitou i do intenzity IP[xp] }
        { pokud souřadnice yp je nad nebo pod }
        { již nakreslenou plochou viz obr.7.1.8 }
      MC[xp] := yp;
      IC[xp] := i
end;
```

Algoritmus 7.1.5



Obr. 7.1.8

Eliminace neviditelných vrstevnic

Řešení problému eliminace neviditelných částí vrstevnic (izolinií) zobrazovaných nikoliv jako pohled shora na funkci reprezentovanou soustavou vrstevnic, ale pod nějakým úhlem, je v literatuře popsáno jen zřídka, viz např. [20], neboť zahrnuje řešení více úloh najednou. Prvním problémem je řešení vrstevnic samotných, tj. hledání křivek takových, že:

$$f(x,z) = c_p \qquad p = 1, \ldots, r$$

kde c_p je zvolená hodnota určující vlastně "výšku" vrstevnice. Druhým problémem je pak řešení problému viditelnosti vrstevnic, tj. určení bodu, kde vrstevnice "zachází za obrys plochy", tj. přestává být viditelná. Třetím problémem je pak kreslení siluety (obrysu) v případě, že není kreslena síť, která siluetu vytváří, viz obr. 7.1.9.

Z důvodů jednoduchosti algoritmu budeme předpokládat, že každá vrstevnice může mít jen dva průsečíky s hranicí "políčka" sítě, která je opět v rovině zx. Předpokládejme, že se bude kreslit pomocí metody "Zig-zag", a to odpředu směrem dozadu. Dále pak, že hodnoty jednotlivých vrstevnic jsou uspořádány vzestupně, tj. platí:

$$c_p < c_{p+1} \qquad p = 1, \ldots, r-1$$

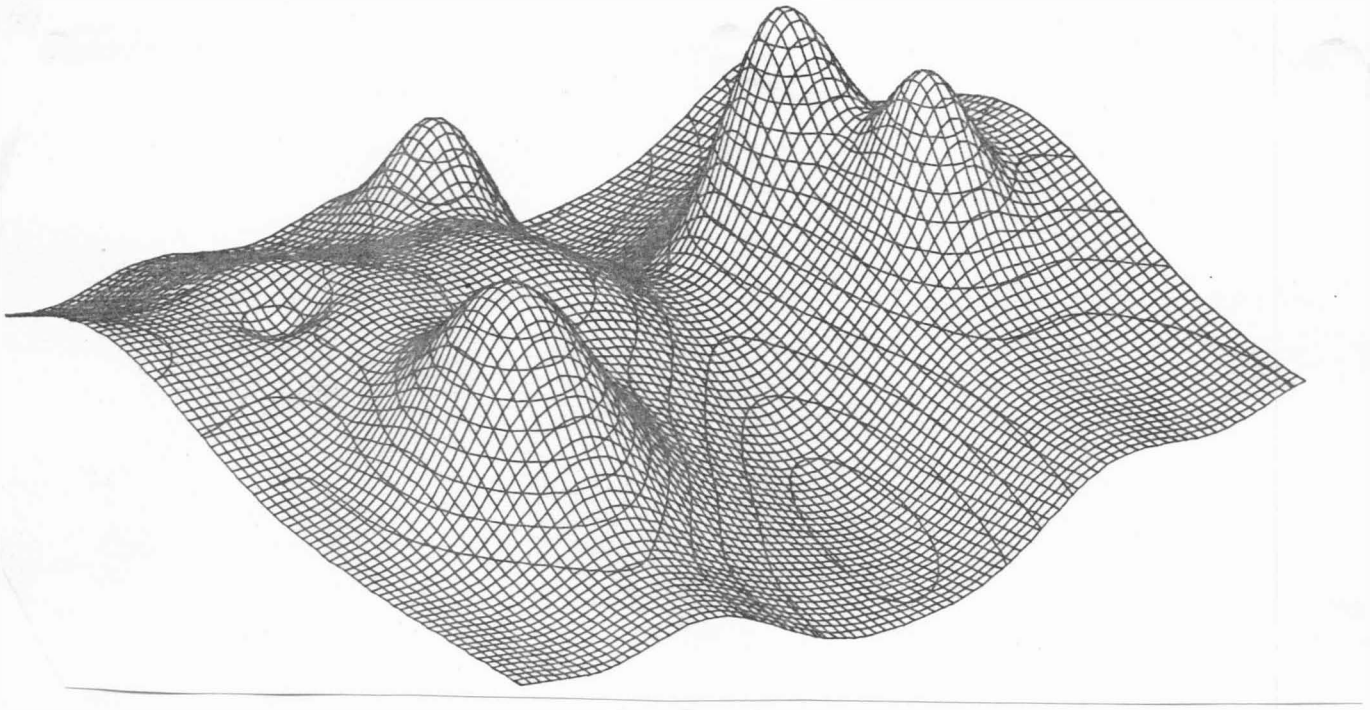
Celý postup může být reprezentován algoritmem 7.1.7.

```
MT := -∞; MB := ∞;
for k:=1 to počet políček sítě do
begin
   for p:=1 to počet vrstevnic do
   begin Vypočti průsečíky p-té vrstevnice s hranicí
         k-tého políčka sítě;
         Kresli spojnice průsečíků s respektováním
         viditelnosti;
   end;
   Kresli zadní hranice zpracovávaného políčka sítě
   s respektováním viditelnosti;
   nastav MT a MB tak, aby vyjadřovaly horní a dolní obrys
end;
```

Algoritmus 7.1.7

Z uvedeného je zřejmé, že se opět s výhodou používá příkazu kreslení úsečky vzhledem k viditelnosti.

Uvedené algoritmy je možné poměrně snadno modifikovat pro trojúhelníkovou síť, která vznikne diagonálním půlením políček dříve uvažované nepravidelné obdélníkové sítě.



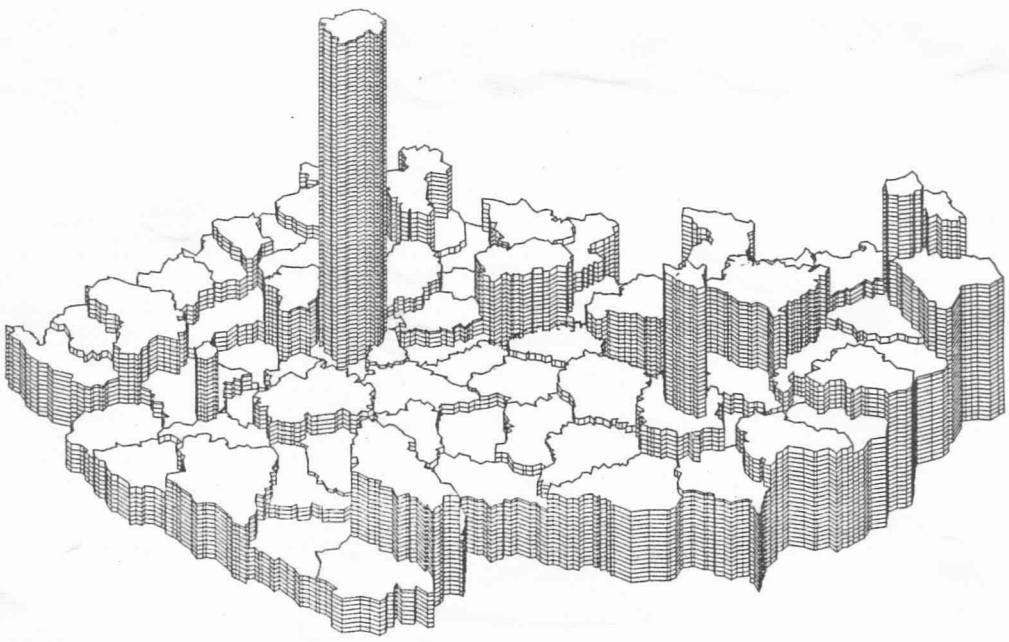
Obr. 7.1.9

V případě obecné nepravidelné sítě je pak nutné provést vhodné uspořádání hran, což podstatně zpomaluje rychlost řešení daných problémů. Nicméně i přes toto zpomalení je řešení rychlejší než při použití obecnějších algoritmů.

Zobrazování prizmatických diagramů

Prizmatické diagramy jsou speciálním případem sloupečkových diagramů, kdy půdorys sloupečku není obdélníkem, ale je obecně nekonvexním n-úhelníkem. Prizmatické diagramy se velmi často používají ve spojitosti s kartografickými či jinými údaji, např. výška ukazuje hustotu obyvatelstva apod., a proto se též někdy nazývají prizmatickými mapami, viz obr.7.1.10.

Řešení uvedeného problému je již poněkud složitější, neboť pořadí kreslení jednotlivých úseček není možné určit dopředu, avšak jednotlivé hrany musejí být uspořádány podle určitého kritéria. To znamená, že od algoritmu s lineární složitostí o(n), kde n je počet hran, se dostáváme k algoritmu složitějšímu, obecně se složitostí $o(n.\log_2 n)$. I přes toto zvýšení složitosti je stále výhodnější odvodit speciální algoritmus, neboť obecný algoritmus pro eliminaci neviditelných hran je složitosti $o(n^2)$, což při 10 000 zpracovávaných hranách již není zanedbatelné.



Obr.7.1.10

7.2 Robertsův algoritmus

Řešení problému eliminace neviditelných hran a povrchů je v obecném případě velmi komplikované. Eliminace neviditelných hran a povrchů u konvexních těles ohraničených rovinami je poměrně jednoduché, neboť v tomto případě lze těleso vyjádřit jako průnik poloprostorů, jejichž hranicí jsou roviny ohraničující dané těleso. Robertsův algoritmus řeší problém eliminace neviditelných hran a povrchů u scén s konvexními tělesy, jejichž plochy jsou planární. Řešení tohoto problému je realizováno elegantním matematickým postupem pomocí postupů lineárního programování.

Každý poloprostor je obecně dán nerovnicí

$$[\ a\ ,\ b\ ,\ c\ ,\ d\ ]\ .\ [\ x\ ,\ y\ ,\ z\ ,\ 1\ ]^T \geq 0$$

tj.

$$\mathbf{v}^T\ .\ \mathbf{x} \geq \mathbf{0}$$

kde: $\mathbf{v}$ je vektor koeficientů $[\ a\ ,\ b\ ,\ c\ ,\ d\ ]^T$

Konvexní těleso je pak dáno jako průnik poloprostorů, a lze tedy psát

$$\mathbf{V}^T\ .\ \mathbf{x} \geq \mathbf{0}$$

kde

$$\mathbf{V} = \begin{bmatrix} a_1 , & b_1 , & c_1 , & d_1 \\ \vdots & \vdots & \vdots & \vdots \\ a_n , & b_n , & c_n , & d_n \end{bmatrix} = \begin{bmatrix} \mathbf{v}_1^T \\ \vdots \\ \mathbf{v}_n^T \end{bmatrix}$$

je maticí, jejíž každá řádka reprezentuje jednu rovinu ohraničující dané těleso. V dalším budeme předpokládat, že roviny jsou orientovány tak, že

$$\forall i \quad \mathbf{v}_i^T\ \mathbf{x}_I > 0$$

$$\exists i \quad \mathbf{v}_i^T\ \mathbf{x}_O < 0$$

$$\exists i \quad \mathbf{v}_i^T\ \mathbf{x}_S = 0 \qquad \& \qquad \forall j \neq i \qquad \mathbf{v}_j^T\ \mathbf{x}_S > 0$$

kde $\mathbf{x}_I$ je libovolný bod uvnitř tělesa,
$\mathbf{x}_O$ je libovolný bod vně tělesa,
$\mathbf{x}_S$ je libovolný bod na povrchu tělesa.

Pokud není orientace rovin (tedy i poloprostorů) předem určena, je nutné vybrat bod $\mathbf{x}$, který je uvnitř, a provést kontrolu, je-li:

$$\mathbf{v}_i^T \mathbf{x} > 0 \qquad \forall i$$

V případě, že

$$\mathbf{v}_i^T \mathbf{x} < 0$$

je nutné nahradit vektor $\mathbf{v}_i^T$ vektorem $-\mathbf{v}_i^T$, tj. změnit znaménka u koeficientů a_i, b_i, c_i, d_i. Určení vnitřního bodu je vcelku jednoduché vzhledem ke konvexnosti. Jsou-li $\mathbf{x}_i$ a $\mathbf{x}_j$ vrcholy konvexního tělesa, které neleží na stejné rovině, pak bod

$$\mathbf{x} = (\mathbf{x}_i + \mathbf{x}_j) / 2$$

je vnitřním bodem tělesa.

Na základě předchozích vztahů je pak možné přímo eliminovat ty roviny, které ohraničují poloprostory a které jsou "odvrácené" od pozorovatele.

Uvážíme-li paralelní projekci, tj. pozorovatel je v nekonečnu, lze pozici reprezentovat pomocí vektoru:

$$\mathbf{x}_p = [\ 0\ ,\ 0,\ -1\ ,\ 0\ ]^T$$

který zároveň určuje pozici testovacího bodu.

Vektor $\mathbf{x}_p$ reprezentuje po přepočtu z homogenních souřadnic libovolný bod na rovině $z = -\infty$, tj. libovolný bod $(x,y,-\infty)$. Pak pro roviny, reprezentované vektorem $\mathbf{v}_j$, které jsou "odvrácené" od pozorovatele, platí:

$$\mathbf{v}_j^T \mathbf{x}_p < 0$$

Z rovnice vyplývá, že uvedená podmínka je ekvivalentní podmínce:

$$c_j > 0$$

Uvedený postup je patrně nejjednodušším algoritmem pro eliminaci neviditelných ploch a jsou-li plochy obecně nekonvexního tělesa orientovány, lze jej využít k podstatnému zrychlení algoritmu. Tímto způsobem lze eliminovat všechny neviditelné plochy konvexního tělesa. Hrany, které jsou tvořeny průsečnicí dvou neviditelných ploch, je možné odstranit jednoduchým postupem.

I když je pochopitelně nutné určit vrcholy daného konvexního tělesa, je vhodné spolu s maticí mít zaznamenány informace,

které hrany a které body leží na které ploše. Pokud roviny p_1 , p_2 , p_3 reprezentované vektory $\mathbf{v}_1$, $\mathbf{v}_2$, $\mathbf{v}_3$ se protínají v bodě **x**, pak musí platit:

$$\mathbf{v}_1^T \,.\, \mathbf{x} = 0 \qquad \mathbf{v}_2^T \,.\, \mathbf{x} = 0 \qquad \mathbf{v}_3^T \,.\, \mathbf{x} = 0$$

Přepsáním do maticového tvaru při použití homogenních souřadnic lze psát

$$\mathbf{B} \,.\, \mathbf{x} = \mathbf{r}$$

kde $\mathbf{B} = [\, \mathbf{v}_1 \,,\, \mathbf{v}_2 \,,\, \mathbf{v}_3 \,,\, \mathbf{s} \,]^T$

$\mathbf{s} = [\, 0 \,,\, 0 \,,\, 0 \,,\, 1 \,]^T$ $\qquad$ $\mathbf{r} = [\, 0 \,,\, 0 \,,\, 0, \, d \,]^T$

přičemž $d > 0$ je měřítkový faktor.

Vynásobením rovnice maticí inverzní, tj. $\mathbf{B}^{-1}$, obdržíme

$$\mathbf{x} = \mathbf{B}^{-1} \,.\, \mathbf{r}$$

Z analýzy rovnice vyplývá, že vrchol je vlastně určen posledním sloupcem matice $\mathbf{B}^{-1}$.

Nyní je nezbytné zjistit, zdali jednotlivé hrany nejsou zakryty jiným tělesem. To znamená, že je nutné porovnat každou hranu vůči ostatním těm tělesům na scéně, ke kterým tato hrana nepatří. Ke zrychlení tohoto procesu lze použít tzv. "z-sort" a testy typu **minmax**, které podstatně urychlují proces zjišťování, zdali hrana může být zakryta jiným tělesem.

Hranu tělesa lze vyjádřit pomocí parametrické rovnice

$$\mathbf{v} = \mathbf{s} + \mathbf{d} \,.\, t$$

kde: **v** je vektor bodu na přímce, s je vektor počátečního bodu a **d** je směrový vektor přímky. Je-li tato hrana zakryta, pak je nezbytné určit ty hodnoty parametru t, pro které je zakryta. Vyjádříme-li svazek přímek mezi pozorovatelem a body **x**(t) ležícími na úsečce $\mathbf{x}_1\mathbf{x}_2$ pomocí parametrické rovnice s parametrem α, pak (viz obr.7.2.1.a)

$$\mathbf{Q}(\alpha, t) = \mathbf{v} + \mathbf{g} \,.\, \alpha = \mathbf{s} + \mathbf{d} \,.\, t + \mathbf{g} \,.\, \alpha \qquad t \in \langle\, 0 \,,\, 1 \,\rangle$$

kde **g** je směrový vektor mezi bodem na úsečce a pozorovatelem
v je vektor bodu na úsečce $\mathbf{x}_1\mathbf{x}_2$.

Vzhledem k použití paralelní projekce platí, že:

$$\mathbf{g} = [\, 0 \,,\, 0 \,,\, 1 \,,\, 0 \,]^T$$

To znamená, že $Q(\alpha,t)$ reprezentuje část roviny v E_3 a parametry α a t jednoznačně určují pozici bodu na této ploše. Chceme-li určit část úsečky, která je zakryta jiným tělesem, pak lze využít s výhodou opět skalárního součinu.

Označíme-li

$$\mathbf{h} = \mathbf{B} \,.\, Q(\alpha,t)$$

pak $\quad \mathbf{h} = \mathbf{B} \,.\, \mathbf{s} + t \,.\, \mathbf{B} \,.\, \mathbf{d} + a \,.\, \mathbf{B} \,.\, \mathbf{g}$

$$t \in \langle 0 , 1 \rangle \;\&\; \alpha \geq 0$$

a označíme-li

$$\mathbf{p} = \mathbf{B} \,.\, \mathbf{s} \qquad \mathbf{q} = \mathbf{B} \,.\, \mathbf{d} \qquad \mathbf{w} = \mathbf{B} \,.\, \mathbf{g}$$

lze psát:

$$\mathbf{h} = \mathbf{p} + t \,.\, \mathbf{q} + \alpha \,.\, \mathbf{w} \qquad t \in \langle 0 , 1 \rangle \;\&\; \alpha \geq 0$$

Je-li

$$h_j = p_j + t.p_j + \alpha.w_j > 0 \qquad t \in \langle 0 , 1 \rangle \;\&\; \alpha \geq 0 \quad \forall\, j$$

pak daný bod je zakryt tělesem, které je reprezentováno maticí **B**. Vzhledem k tomu, že hledáme hodnoty parametrů $t \in \langle t_1 , t_2 \rangle$, pro které je zkoumaná úsečka zakryta tělesem, je nutné řešit soustavu nerovnic:

$$h_j > 0 \qquad \forall\, j$$

Hraničním případem mezi viditelností a neviditelností bodu je případ, kdy $h_j=0$, tj. bod leží na rovině. Položíme-li

$$h_i = 0 \quad \& \quad h_j = 0 \quad \forall\, i,j \quad \& \quad i \neq j$$

dostáváme soustavu $n(n-1)/2$ rovnic, které je nutné řešit vzhledem k proměnným t a α, přičemž je nezbytné testovat, zdali:

$$t \in \langle 0 , 1 \rangle \quad \& \quad \alpha \geq 0 \quad \& \quad h_k > 0 \quad \forall\, k \neq i,j$$

Aplikací uvedeného postupu na všechny plochy h_i a h_j obdržíme interval hodnot parametru t, který určuje tu část úsečky, která je zakryta daným tělesem. Postup lze zapsat algoritmem 7.2.1.

```
procedure TEST ( i,j: integer; t: real );
var k: integer; { kontrola podmínek h_k>0, nastavení t_min, t_max }
begin for k:=1 to n  do
        if (k ≠ i) and (k ≠ j) then if h_k < 0 then EXIT;
      if t < t_min then t_min := t;   if t > t_max then t_max := t
end { TEST };
t_min := 1; t_max := -1;
for i:=1 to n-1 do
    for j:=i+1 to n do
    begin { platí, že i ≠ j }
          SOLVE ( h_i = 0, h_j = 0 , t , α );
          { řešení soustavy dvou rovnic o neznámých t a α }
          if (α ≥ 0) and (t ∈ <0,1>) then TEST ( i , j , t )
    end;
```

Algoritmus 7.2.1

Z dosud uvedeného vyplývá, že proces je velmi složitý z hlediska výpočtové náročnosti. Je tedy nanejvýš žádoucí jej urychlit, což znamená najít taková kritéria, která umožní detekovat ty případy, kdy je úsečka zcela viditelná. Lze ukázat, např. viz [94],[109], že jsou-li pro nějaké j splněny podmínky

$$w_j \leq 0 \quad \& \quad p_j \leq 0 \quad \& \quad p_j + q_j \leq 0$$

pak daná úsečka je zcela viditelná. Tyto podmínky zajistí, že relace

$$h_j \leq 0$$

nemůže být splněna pro žádné $t \in \langle 0 , 1 \rangle$ a $\alpha \geq 0$. Tedy žádná část úsečky nemůže být zakryta a tudíž celá úsečka je viditelná. Test na zjištění, zda úsečka je zcela neviditelná, není již triviální a je nutné aplikovat výše uvedený postup.

Robertsův algoritmus může být rozdělen v zásadě do tří částí, a to:

- odstranění neviditelných hran a povrchů při uvažování pouze samotného konvexního tělesa,
- porovnání hrany a ostatních těles a nalezení zakrytých částí,
- nalezení hran vznikajících v případě, že se tělesa protínají, tj. nalezení průsečnic dvou těles.

Algoritmus předpokládá, že scéna obsahuje pouze konvexní tělesa, která jsou tvořena planárními stěnami. Každá stěna je určena hranami, které jsou definovány koncovými body. Robertsův algoritmus není přímo použitelný pro komplikovanější scény, neboť jeho výpočtová složitost je $o(n^2)$, a proto byly vyvinuty různé modifikace, které aplikují rozdělení scény do menších částí a tím snižují výpočetní složitost na téměř lineární, viz [88],[109]. Zásadní nevýhodou je však omezení na konvexní tělesa. Nicméně Robertsův algoritmus názorně ukazuje, jaké matematické prostředky mohou být použity i v jiných algoritmech.

Příklad$_1$

Předpokládejme, že je dáno 6 rovin popisujících krychli, a to:

p_1: $x = 1/2$ $\quad$ p_2: $x = -1/2$

p_3: $y = 1/2$ $\quad$ p_4: $y = -1/2$

p_5: $z = 1/2$ $\quad$ p_6: $z = -1/2$

Rovnice roviny p_1 , viz obr.7.2.1.b, je pak:

$$2x - 1 = 0$$

Celková matice **V** definující konvexní těleso má pak tvar

$$\mathbf{V} = \begin{bmatrix} 2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 2 \\ -1 & 1 & -1 & 1 & -1 & 1 \end{bmatrix}^T$$

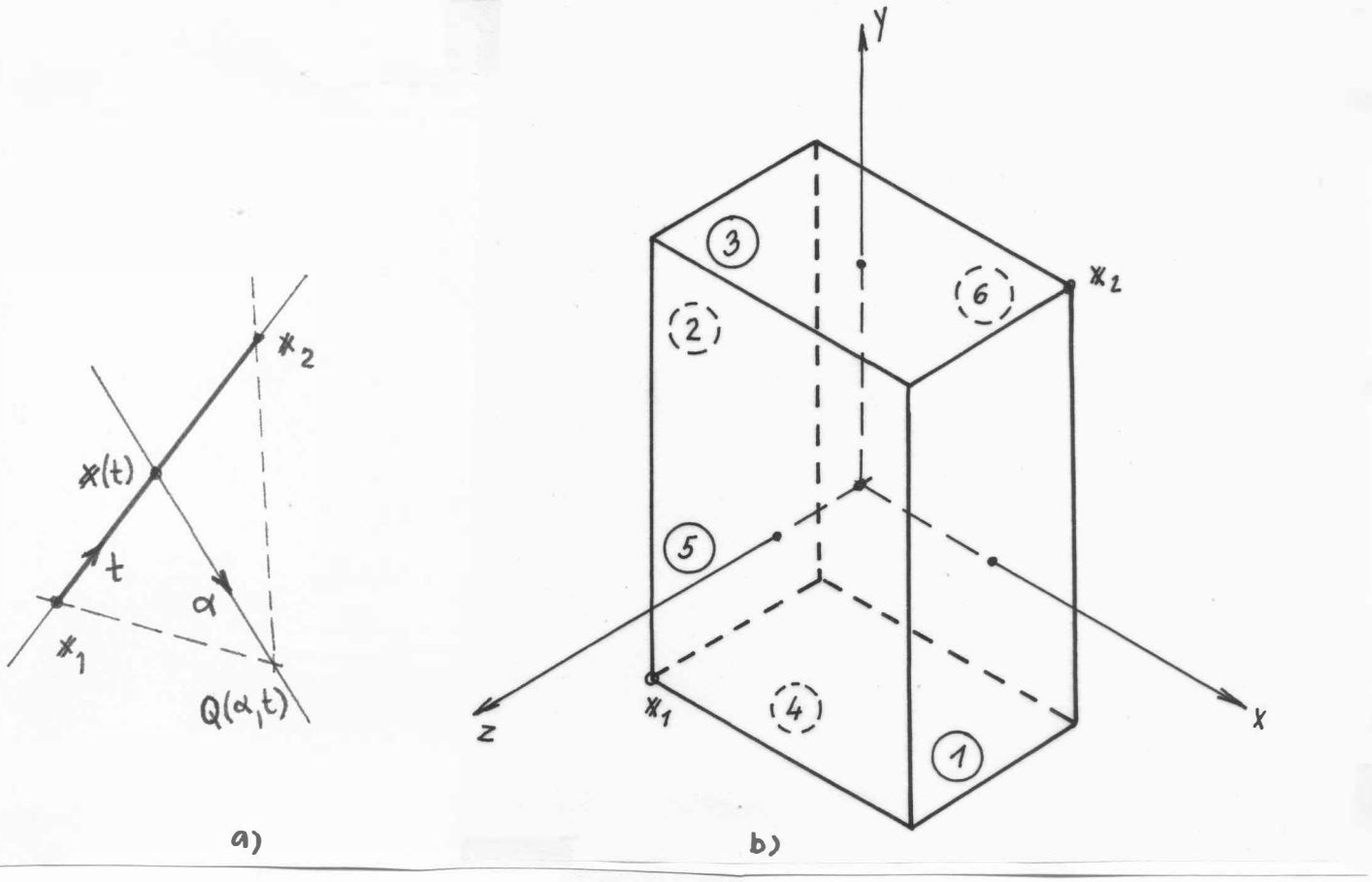
Vybereme-li vnitřní bod $\mathbf{x} = (0.25, 0.25, 0.25)$ za testovací bod, pak v homogenních souřadnicích je určen pomocí vektoru:

$$\mathbf{x} = [\ 1\ ,\ 1\ ,\ 1\ ,\ 4\ ]^T$$

Nyní je nutné určit orientaci jednotlivých ploch, a to:

$$\mathbf{V} \cdot \mathbf{x} = \begin{bmatrix} 2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 2 \\ -1 & 1 & -1 & 1 & -1 & 1 \end{bmatrix}^T \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \\ 4 \end{bmatrix}$$

$$= [\ -2\ ,\ 6\ ,\ -2\ ,\ 6\ ,\ -2\ ,\ 6\ ]^T$$



Obr. 7.2.1

Je tedy nutné vynásobit 1., 3., 5. řádek matice **V** konstantou -1, tj. obrátit orientaci dané plochy. Takto modifikovaná matice **V** pak reprezentuje konvexní těleso.

$$\mathbf{V} = \begin{bmatrix} -2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}^T$$

Nyní

$$\mathbf{V} \cdot \mathbf{x} = [\ 2\ ,\ 6\ ,\ 2\ ,\ 6\ ,\ 2\ ,\ 6\ ]^T$$

Je zřejmé, že v případě geometrických transformací je nutné geometrické transformace aplikovat nejen na jednotlivé roviny, ale i na body.

Předpokládejme, že chceme posunout danou krychli v kladném směru osy x, tj. matice **T** vyjadřující posuv má při použití homogenních souřadnic tvar:

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & 3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Pak matice **V'** definující konvexní těleso po posunutí je definována jako:

$$\mathbf{V'} = \mathbf{V} \cdot \mathbf{T}^{-1} = \begin{bmatrix} -2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}^{T} \cdot \begin{bmatrix} 1 & 0 & 0 & -3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{V'} = \begin{bmatrix} -2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2 & 2 \\ 7 & -5 & 1 & 1 & 1 & 1 \end{bmatrix}^{T}$$

Jestliže nyní použijeme původní bod $\mathbf{x} = [\ 1\ ,\ 1\ ,\ 1\ ,\ 4\ ]^T$ k testování orientace, pak dostáváme:

$$\mathbf{V'}.\ \mathbf{x} = [\ 26\ ,\ -18\ ,\ 2\ ,\ 6\ ,\ 2\ ,\ 6\ ]^T$$

To znamená, že bod (0.25,0.25,0.25) leží vně daného n-úhelníka.

V případě, že i na bod **x** aplikujeme operaci posuvu, tj.

$$\mathbf{x'} = \mathbf{T} \cdot \mathbf{x} = [\ 13\ ,\ 1\ ,\ 1\ ,\ 4\ ]^T$$

pak

$$\mathbf{V'} \cdot \mathbf{x'} = [\ 2\ ,\ 6\ ,\ 2\ ,\ 6\ ,\ 2\ ,\ 6\ ]^T$$

což je výsledek očekávaný, neboť bod **x'** je uvnitř daného konvexního tělesa.

Analogicky je možné postupovat i v případě transformace rotace. Budeme-li danou krychli otáčet o úhel $\phi = \pi/4$ okolo osy y, pak transformační matice má tvar:

$$\mathbf{R}_{zx}(\phi) = \begin{bmatrix} \cos\phi & 0 & \sin\phi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\phi & 0 & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_{zx}^{-1}(\phi) = \begin{bmatrix} \cos\phi & 0 & -\sin\phi & 0 \\ 0 & 1 & 0 & 0 \\ \sin\phi & 0 & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_{zx}(\pi/4) = \begin{bmatrix} \sqrt{2}/2 & 0 & \sqrt{2}/2 & 0 \\ 0 & 1 & 0 & 0 \\ -\sqrt{2}/2 & 0 & \sqrt{2}/2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_{zx}^{-1}(\pi/4) = \begin{bmatrix} \sqrt{2}/2 & 0 & -\sqrt{2}/2 & 0 \\ 0 & 1 & 0 & 0 \\ \sqrt{2}/2 & 0 & \sqrt{2}/2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Pak matice **V''** reprezentující konvexní těleso po rotaci má tvar

$$\mathbf{V''} = \mathbf{V} \cdot R_{zx}^{-1} =$$

$$\begin{bmatrix} -2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}^T \cdot \begin{bmatrix} \sqrt{2}/2 & 0 & -\sqrt{2}/2 & 0 \\ 0 & 1 & 0 & 0 \\ \sqrt{2}/2 & 0 & \sqrt{2}/2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} =$$

$$\begin{bmatrix} -\sqrt{2} & \sqrt{2} & 0 & 0 & -\sqrt{2} & \sqrt{2} \\ 0 & 0 & -2 & 2 & 0 & 0 \\ \sqrt{2} & -\sqrt{2} & 0 & 0 & -\sqrt{2} & \sqrt{2} \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}^T$$

Uvážíme-li nyní pozici pozorovatele v bodě $[\ 0\ ,\ 0\ ,\ 1\ ,\ 0\ ]^T$, tj. pozorovatel je v poloze $z = \infty$, je testovací bod určen jako

$$\mathbf{x} = [\ 0\ ,\ 0\ ,\ -1\ ,\ 0\ ]^T$$

a tedy:

$$\mathbf{V''}.\ \mathbf{x} = [\ -\sqrt{2}\ ,\ \sqrt{2}\ ,\ 0\ ,\ 0\ ,\ \sqrt{2}\ ,\ -\sqrt{2}\ ]^T$$

To znamená, že 1. a 6. rovina jsou neviditelné.

Je zřejmé, že místo transformace matice **V** reprezentující dané těleso je možné transformovat testující bod, a tedy:

$$\mathbf{x''} = \mathbf{R}_{zx}^{-1} \cdot \mathbf{x} = \begin{bmatrix} \sqrt{2}/2 & 0 & -\sqrt{2}/2 & 0 \\ 0 & 1 & 0 & 0 \\ \sqrt{2}/2 & 0 & \sqrt{2}/2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \\ -1 \\ 0 \end{bmatrix}$$

$$\mathbf{x''} = \sqrt{2}/2 \cdot [\ 1\ ,\ 0\ ,\ -1\ ,\ 0\ ]^T$$

Pak

$$\mathbf{V} \cdot \mathbf{x''} = [\ -\sqrt{2}\ ,\ \sqrt{2}\ ,\ 0\ ,\ 0\ ,\ \sqrt{2}\ ,\ -\sqrt{2}\ ]^T$$

což je výsledek očekávaný.

Je tedy zřejmé, že hrana, která je průsečnicí 1. a 6. plochy, nemusí být testována zdali je viditelná, neboť průsečnice dvou neviditelných (zakrytých) ploch je neviditelná též.

Příklad$_2$

Předpokládejme, že je opět dána krychle z předchozího příkladu v základní poloze, viz obr.7.2.2., a úsečka s koncovými body $\mathbf{x}_1$ a $\mathbf{x}_2$ a chceme určit, která část úsečky je zakryta. Je tedy dáno:

$$\mathbf{x}_1 = [\ -2\ ,\ 0\ ,\ -2\ ,\ 1\ ]^T \qquad \mathbf{x}_2 = [\ 2\ ,\ 0\ ,\ -2\ ,\ 1\ ]^T$$

$$\mathbf{V} = \begin{bmatrix} -2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}^T$$

přičemž předpokládáme, že pozorovatel je v bodě $(x,y,-\infty)$. Pak lze psát, že

$$\mathbf{g} = [\ 0\ ,\ 0\ ,\ 1\ ,\ 0\ ]^T$$

a

$$\mathbf{s} = [\ -2\ ,\ 0\ ,\ -2\ ,\ 1\ ]^T \qquad \mathbf{d} = [\ 4\ ,\ 0\ ,\ 0\ ,\ 0\ ]^T$$

Pak jednotlivé koeficienty určující výraz pro h_j lze určit jako:

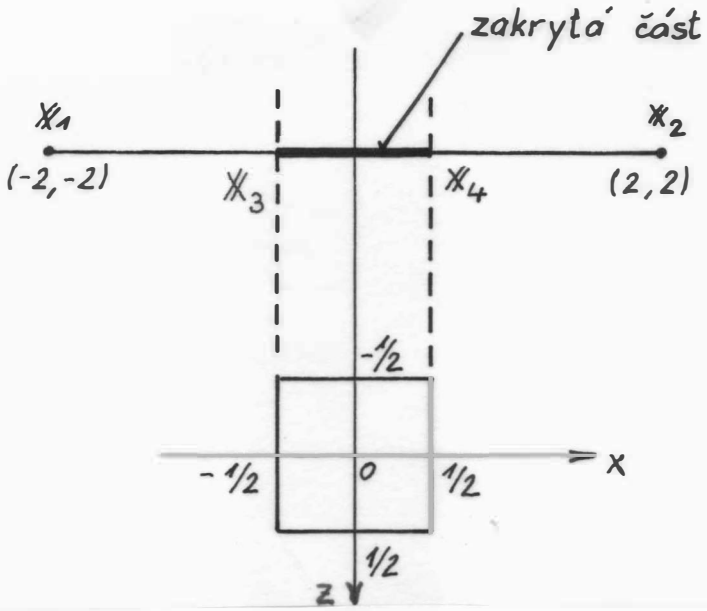
$$\mathbf{p} = \mathbf{V} \cdot \mathbf{s} = [\ 5\ ,\ -3\ ,\ 1\ ,\ 1\ ,\ 5\ ,\ -3\ ]^T$$

$$\mathbf{q} = \mathbf{V} \cdot \mathbf{d} = [\ -8\ ,\ 8\ ,\ 0\ ,\ 0\ ,\ 0\ ,\ 0\ ]^T$$

$$\mathbf{w} = \mathbf{V} \cdot \mathbf{g} = [\ 0\ ,\ 0\ ,\ 0\ ,\ 0\ ,\ -2\ ,\ 2\ ]^T$$

Nerovnosti $h_j > 0$ lze vyjádřit rozepsáním takto:

$$
\begin{aligned}
5 - 8t &> 0 \\
-3 + 8t &> 0 \\
1 &> 0 \\
1 &> 0 \\
5 - 2\alpha &> 0 \\
-3 + 2\alpha &> 0
\end{aligned}
$$



Obr. 7.2.2

Uvedenou soustavu nerovnic lze řešit graficky, viz obr. 7.2.3, v prostoru α a t.

Je zřejmé, že nerovnice

$$\mathbf{h} > \mathbf{0}$$

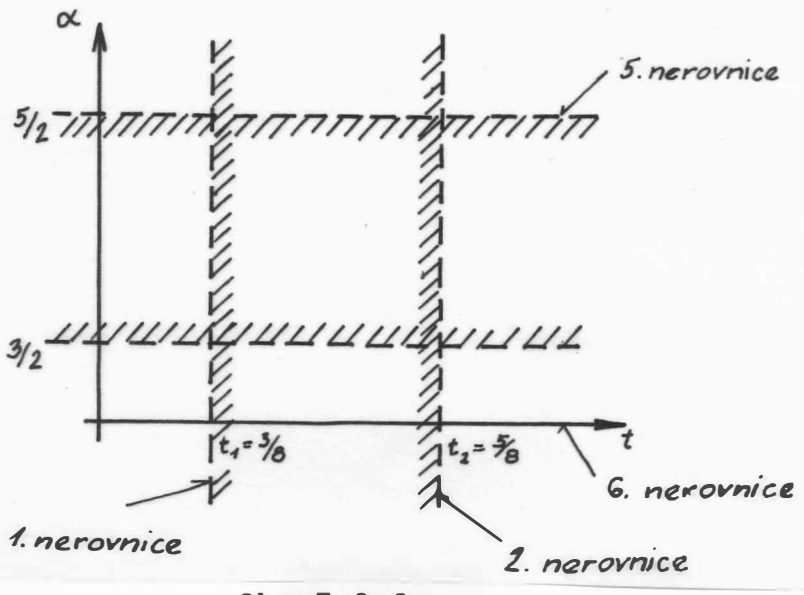
jsou splněny pouze uvnitř oblasti dané kartézským součinem:

$$t \in \langle\, 3/8 \,,\, 5/8 \,\rangle \quad \mathbf{x} \quad \alpha \in \langle\, 3/2 \,,\, 5/2 \,\rangle$$

To znamená, že daná úsečka je neviditelná z dané pozice pozorovatele pro $t \in \langle\, 3/8 \,,\, 5/8 \,\rangle$ a viditelné části jsou tedy pro $t \in \langle\, 0 \,,\, 3/8 \,\rangle$ a $t \in \langle\, 5/8 \,,\, 1 \,\rangle$. Pak lze určit jednotlivé koncové body úseků takto:

$$\mathbf{x}_3 = \mathbf{s} + \mathbf{d}.t = [\, -1/2 \,,\, 0 \,,\, -2 \,,\, 1 \,]^T$$

$$\mathbf{x}_4 = \mathbf{s} + \mathbf{d}.t = [\, 1/2 \,,\, 0 \,,\, -2 \,,\, 1 \,]^T$$



Obr. 7.2.3

Příklad$_3$

Uvažme nyní opět stejnou krychli v základní poloze a jinou úsečku, která je opět dána svými koncovými body, a to:

$$\mathbf{x}_1 = [\ 1\ ,\ 0\ ,\ -1\ ,\ 1\ ]^T \qquad \mathbf{x}_2 = [\ 0\ ,\ 0\ ,\ -1\ ,\ 1\ ]^T$$

viz obr. 7.2.4.

Pak vektory **s** a **d** jsou dány:

$$\mathbf{s} = [\ 1,\ 0\ ,\ -1\ ,\ 1\ ]^T \qquad \mathbf{d} = [\ -1\ ,\ 0\ ,\ 0\ ,\ 0\ ]^T$$

přičemž vektor **g** reprezentující polohu pozorovatele je opět:

$$\mathbf{g} = [\ 0\ ,\ 0\ ,\ 1\ ,\ 0\ ]^T$$

Vektory **p** , **q** , **w** jsou určeny takto:

$$\mathbf{p} = \mathbf{V}\ .\ \mathbf{s} = [\ -1\ ,\ 3\ ,\ 1\ ,\ 1\ ,\ 3\ ,\ -1\ ]^T$$

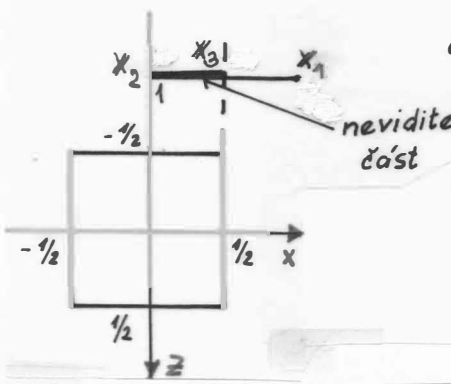
$$\mathbf{q} = \mathbf{V}\ .\ \mathbf{d} = [\ 2\ ,\ -2\ ,\ 0\ ,\ 0\ ,\ 0\ ,\ 0\ ]^T$$

$$\mathbf{w} = \mathbf{V}\ .\ \mathbf{g} = [\ 0\ ,\ 0\ ,\ 0\ ,\ 0\ ,\ -2\ ,\ 2\ ]^T$$

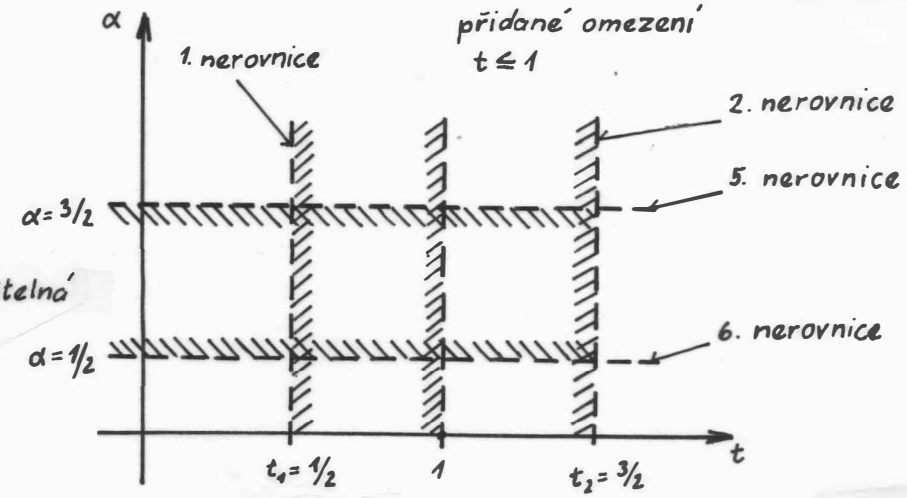
Rozepsáním vektorové nerovnice $\mathbf{h} > \mathbf{0}$ do složek dostáváme:

$$\begin{aligned} -1 \quad &+ 2t \quad &&> 0 \\ 3 \quad &- 2t \quad &&> 0 \\ 1 \quad & \quad &&> 0 \\ 1 \quad & \quad &&> 0 \\ 3 \quad & \quad &- 2\alpha \ &> 0 \\ -1 \quad & \quad &+ 2\alpha \ &> 0 \end{aligned}$$

Grafickým řešením opět dostáváme oblast hodnot parametrů t a α, viz obr. 7.2.5



Obr. 7.2.4



Obr. 7.2.5

Je zřejmé, že hodnota $t_1 = 3/2$ musí být odmítnuta, neboť pro hodnoty parametru t musí platit, že $t \in \langle 0 , 1 \rangle$. Pak tedy viditelná část úsečky je určena hodnotami parametru $t \in \langle 1/2 , 1 \rangle$ a neviditelná část je určena hodnotami parametru t tak, že $t \in \langle 0 , 1/2 \rangle$. Bod $\mathbf{x}_3$ je pak určen takto:

$$\mathbf{x} = \mathbf{s} + \mathbf{d}.t = [\ 1/2 , 0 , -1 , 1\ ]^T$$

což je výsledek očekávaný, viz obr. 7.2.4.

Příklad 4

Předpokládejme, že je dána opět krychle v základní poloze a úsečka, která je dána svými koncovými body $\mathbf{x}_1$ a $\mathbf{x}_2$ tak, že přímka protíná krychli (jde tedy o případ, kdy se plochy jednotlivých těles protínají, viz obr. 7.2.6.), přičemž:

$$\mathbf{x}_1 = [\ 1 , 0 , 2 , 1\ ]^T \qquad \mathbf{x}_2 = [\ -1 , 0 , -2 , 1\ ]^T$$

Pak vektory **s** a **d** jsou dány:

$$\mathbf{s} = [\ 1 , 0 , 2 , 1\ ]^T \qquad \mathbf{d} = [\ -2 , 0 , -4 , 0\ ]^T$$

přičemž vektor **g** reprezentující polohu pozorovatele je opět:

$$\mathbf{g} = [\ 0 , 0 , 1 , 0\ ]^T$$

Vektory **p** , **q** , **w** jsou určeny takto:

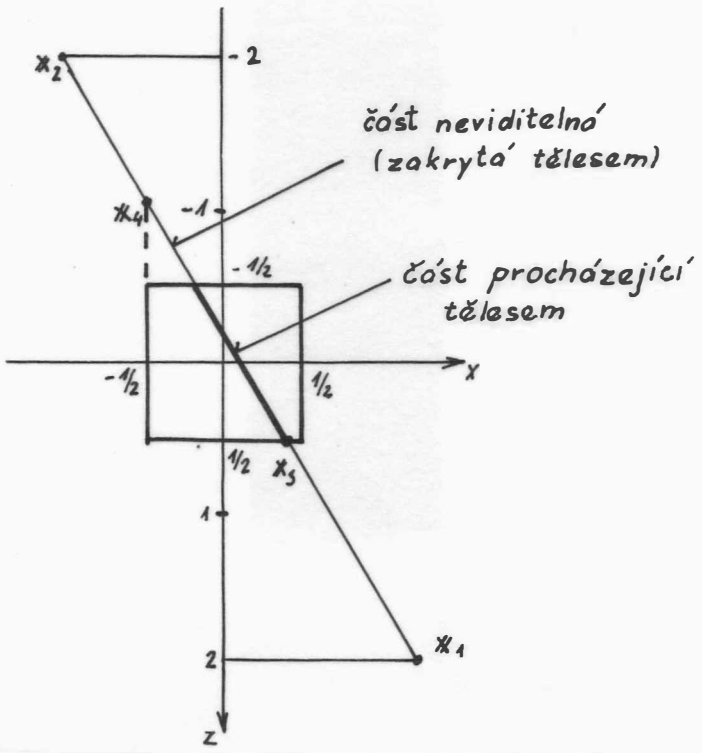
$$\mathbf{p} = \mathbf{V} \,.\, \mathbf{s} = [\ -1\ ,\ 3\ ,\ 1\ ,\ 1\ ,\ -3\ ,\ 5\ ]^T$$

$$\mathbf{q} = \mathbf{V} \,.\, \mathbf{d} = [\ 4\ ,\ -4\ ,\ 0\ ,\ 0\ ,\ 8\ ,\ -8\ ]^T$$

$$\mathbf{w} = \mathbf{V} \,.\, \mathbf{g} = [\ 0\ ,\ 0\ ,\ 0\ ,\ 0\ ,\ -2\ ,\ 2\ ]^T$$

Rozepsáním vektorové nerovnice $\mathbf{h} > \mathbf{0}$ do složek dostáváme:

$$\begin{aligned} -1 \quad &+ 4t \qquad\quad &> 0 \\ 3 \quad &- 4t \qquad\quad &> 0 \\ 1 \quad & \qquad\quad &> 0 \\ 1 \quad & \qquad\quad &> 0 \\ -3 \quad &+ 8t \ - 2\alpha \quad &> 0 \\ 5 \quad &- 8t \ + 2\alpha \quad &> 0 \end{aligned}$$



Obr. 7.2.6

Řešení uvedené soustavy nerovnic již není tak triviální, neboť hodnoty parametrů t a α jsou mezi sebou svázány. Grafické řešení je uvedeno na obr. 7.2.7.

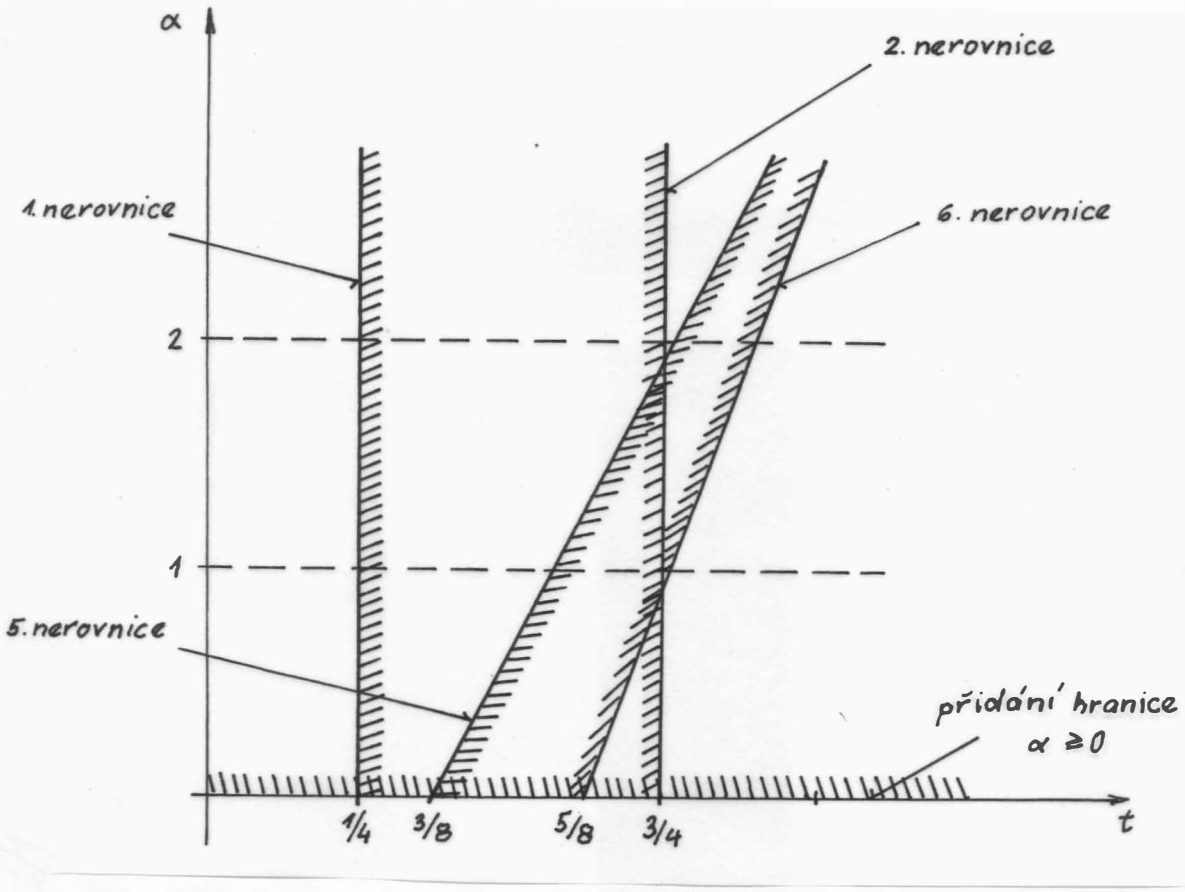
Je zřejmé, že rozsah hodnot parametrů t, pro které je daná úsečka neviditelná, je určen jako $t \in \langle\ 3/8\ ,\ 3/4\ \rangle$ a úseky viditelné části jsou určeny parametrem t takto:

$t \in \langle 0 , 3/8 \rangle$ $\quad$ $t \in \langle 3/4 , 1 \rangle$

Pak

$$\mathbf{x}_3 = \mathbf{s} + \mathbf{d}.t = [\ 1/4 , 0 , 1/2 , 1\ ]^T \quad \text{pro } t = 3/8$$

$$\mathbf{x}_4 = \mathbf{s} + \mathbf{d}.t = [\ -1/2 , 0 , -1 , 1\ ]^T \quad \text{pro } t = 3/4$$



Obr. 7.2.7

Příklad 5

Uvažme opět danou krychli a dvě úsečky dané koncovými body $\mathbf{x}_1\mathbf{x}_2$ a $\mathbf{x}_3\mathbf{x}_4$, kde

$$\mathbf{x}_1 = [\ -2 , 0 , 2 , 1\ ]^T \qquad \mathbf{x}_2 = [\ 2 , 0 , 2 , 1\ ]^T$$

$$\mathbf{x}_3 = [\ -1 , 1 , 1 , 1\ ]^T \qquad \mathbf{x}_4 = [\ 1 , 1 , -1 , 1\ ]^T$$

viz obr. 7.2.8.

Pak pro úsečku $\mathbf{x}_1\mathbf{x}_2$ jsou vektory **s** a **d** dány

$$\mathbf{s} = [\ -2 , 0 , 2 , 1\ ]^T \qquad \mathbf{d} = [\ 4 , 0 , 0 , 0\ ]^T$$

přičemž vektor **g** reprezentující polohu pozorovatele je opět:

$$\mathbf{g} = [\ 0 , 0 , 1 , 0\ ]^T$$

Vektory **p**, **q**, **w** jsou určeny takto:

$$\mathbf{p} = \mathbf{V} \,.\, \mathbf{s} = [\ 5\ ,-3\ ,\ 1\ ,\ 1\ ,\ -3\ ,\ 5\ ]^T$$

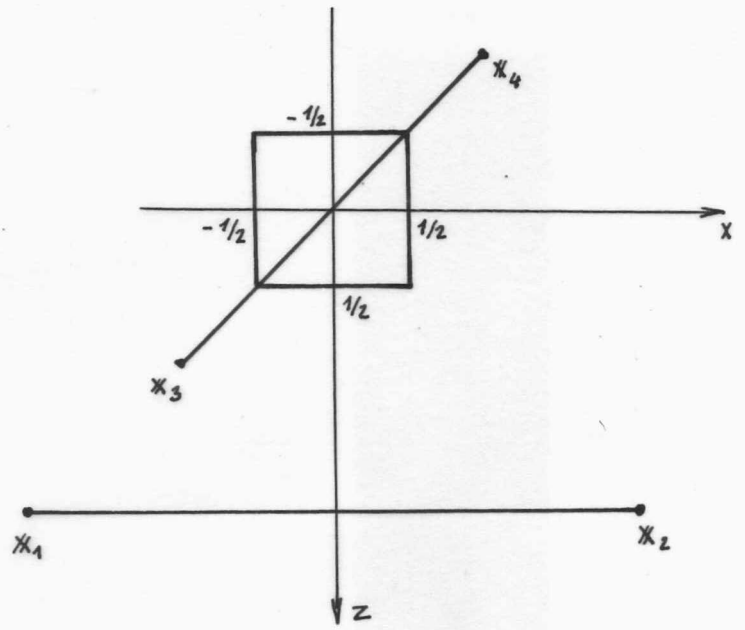
$$\mathbf{q} = \mathbf{V} \,.\, \mathbf{d} = [\ -8\ ,\ 8\ ,\ 0\ ,\ 0\ ,\ 0\ ,\ 0\ ]^T$$

$$\mathbf{w} = \mathbf{V} \,.\, \mathbf{g} = [\ 0\ ,\ 0\ ,\ 0\ ,\ 0\ ,\ -2\ ,\ 2\ ]^T$$

Poznamenejme, že

$$w_5 \leq 0 \quad \& \quad p_5 \leq 0 \quad \& \quad p_5 + q_5 \leq 0$$

a tedy daná úsečka je zcela viditelná.



Obr. 7.2.8

Uvážíme-li úsečku $\mathbf{x}_3\mathbf{x}_4$, pak dostáváme pro vektory **s** a **d**:

$$\mathbf{s} = [\ -1,\ 1\ ,\ 1\ ,\ 1\ ]^T \qquad \mathbf{d} = [\ 2\ ,\ 0\ ,\ -2\ ,\ 0\ ]^T$$

přičemž vektor **g** reprezentující polohu pozorovatele je opět:

$$\mathbf{g} = [\ 0\ ,\ 0\ ,\ 1\ ,\ 0\ ]^T$$

Vektory **p** , **q** , **w** jsou určeny takto:

$$\mathbf{p} = \mathbf{V} \,.\, \mathbf{s} = [\ 3\ ,\ -1\ ,\ -1\ ,\ 3\ ,\ -1\ ,\ 3\ ]^T$$

$$\mathbf{q} = \mathbf{V} \,.\, \mathbf{d} = [\ -4\ ,\ 4\ ,\ 0\ ,\ 0\ ,\ 4\ ,\ -4\ ]^T$$

$$\mathbf{w} = \mathbf{V} \,.\, \mathbf{g} = [\ 0\ ,\ 0\ ,\ 0\ ,\ 0\ ,\ -2\ ,\ 2\ ]^T$$

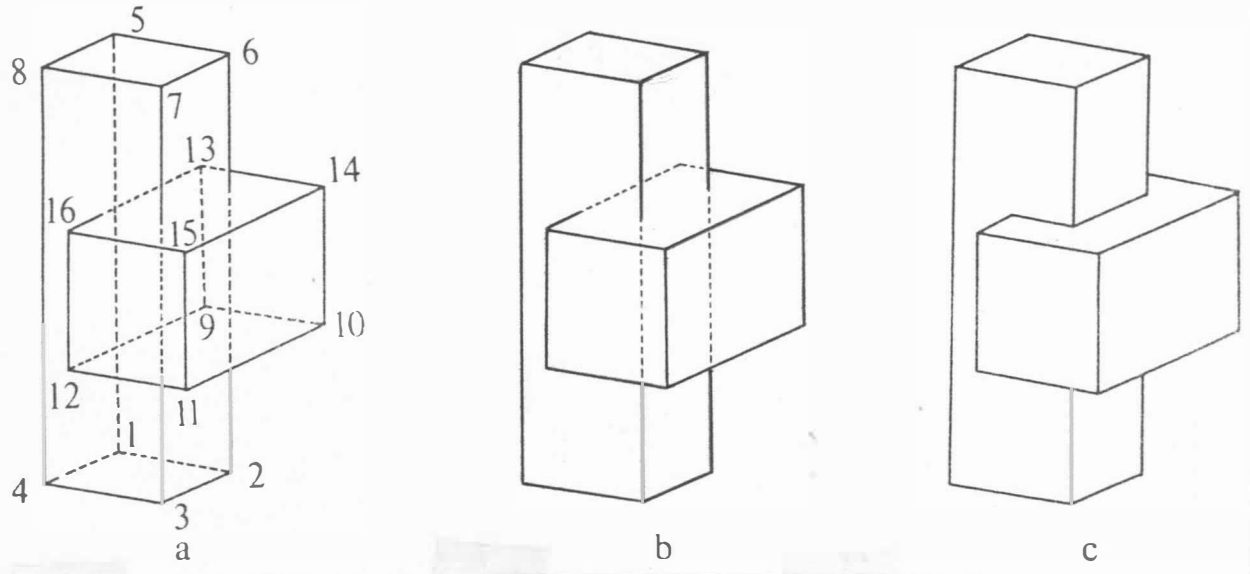
Nyní platí, že:

$$w_3 \leq 0 \quad \& \quad p_3 \leq 0 \quad \& \quad p_3 + q_3 \leq 0$$

a tedy zkoumaná úsečka je také zcela viditelná.

Příklad$_6$

Pro demonstraci Robertsova algoritmu uvažujme jednoduchou scénu [109] sestávající se pouze ze dvou kvádrů, viz obr.7.2.9.a.



Obr.7.2.9

vrchol	1	2	3	4	5	6	7	8
x	0	2	2	0	0	2	2	0
y	0	0	0	0	6	6	6	6
z	1	1	3	3	1	1	3	3

souřadnice vrcholů tělesa$_1$

vrchol	9	10	11	12	13	14	15	16
x	1	3	3	1	1	3	3	1
y	2	2	2	2	4	4	4	4
z	0	0	4	4	0	0	4	4

souřadnice vrcholů tělesa$_2$

Tabulka 7.2.1

hrana	1	2	3	4	5	6	7	8	9	10	11	12
$vrchol_1$	1	2	3	4	5	6	7	8	1	2	3	4
$vrchol_2$	2	3	4	1	6	7	8	5	5	6	7	8

hrany $tělesa_1$

hrana	13	14	15	15	17	18	19	20	21	22	23	24
$vrchol_1$	9	10	11	12	13	14	15	16	9	10	11	12
$vrchol_2$	10	11	12	9	14	15	16	13	13	14	15	16

hrany $tělesa_2$

Tabulka 7.2.2

n-úhelník	hrany	n-úhelník	hrany
1	2, 11, 6, 10	7	14, 23, 18, 22
2	4, 12, 8, 9	8	21, 20, 24, 16
3	5, 6, 7, 8	9	17, 18, 19, 20
4	1, 2, 3, 4	10	13, 14, 15, 16
5	3, 12, 7, 11	11	15, 24, 19, 23
6	1, 10, 5, 9	12	13, 22, 17, 21

tabulka n-úhelníků (stěn)

Tabulka 7.2.3

Předpokládejme nyní, že danou scénu otočíme nejdříve o úhel ϕ okolo osy y. Matice rotace má tvar:

$$\mathbf{R}_{zx}(\phi) = \begin{bmatrix} \cos\phi & 0 & \sin\phi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\phi & 0 & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Poté otočíme scénu o úhel ϑ okolo osy x. Tato transformace je pak reprezentována maticí:

$$\mathbf{R}_{yz}(\vartheta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\vartheta & -\sin\vartheta & 0 \\ 0 & \sin\vartheta & \cos\vartheta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{R}(\vartheta,\phi) = \begin{bmatrix} \cos\phi & 0 & \sin\phi & 0 \\ \sin\vartheta\sin\phi & \cos\vartheta & -\sin\vartheta\cos\phi & 0 \\ -\cos\vartheta\sin\phi & \sin\vartheta & \cos\vartheta\cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Položíme-li např. $\phi = -\pi/6$ a $\vartheta = \pi/12$, pak dostáváme:

$$\mathbf{R}(\pi/12,-\pi/6) = \begin{bmatrix} 0.866 & 0 & -0.5 & 0 \\ -0.129 & 0.966 & -0.224 & 0 \\ 0.483 & 0.259 & 0.837 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transformujeme-li nyní souřadnice bodů každého tělesa, které jsou reprezentovány pomocí matic ${}^{(1)}\mathbf{x}$ a ${}^{(2)}\mathbf{x}$, dostáváme:

$${}^{(1)}\mathbf{x}' = \mathbf{R} \,.\, {}^{(1)}\mathbf{x} \qquad {}^{(2)}\mathbf{x}' = \mathbf{R} \,.\, {}^{(2)}\mathbf{x}$$

Po pronásobení pak

$${}^{(1)}\mathbf{x}' = \begin{bmatrix} -0.5 & 1.232 & 0.232 & -1.5 & -0.5 & 1.232 & 0.232 & -1.5 \\ -0.224 & -0.483 & -0.933 & -0.672 & 5.571 & 5.313 & 4.864 & 5.123 \\ 0.837 & 1.802 & 3.475 & 2.510 & 2.389 & 3.355 & 5.028 & 4.062 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$${}^{(2)}\mathbf{x}' = \begin{bmatrix} 0.866 & 2.598 & 0.598 & -1.134 & 0.866 & 2.598 & 0.598 & -1.134 \\ 1.802 & 1.544 & 0.647 & 0.906 & 3.734 & 3.475 & 2.579 & 2.838 \\ 1.001 & 1.967 & 5.313 & 4.347 & 1.518 & 2.484 & 5.830 & 4.864 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Nyní je nezbytné určit rovnice jednotlivých rovin ohraničujících daná konvexní tělesa, která jsou reprezentována pomocí matic ${}^{(1)}\mathbf{v}$ a ${}^{(2)}\mathbf{v}$.

Rovnice roviny definované třemi body $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$ je dána výrazem:

$$\det \begin{bmatrix} x & y & z & 1 \\ x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \end{bmatrix} = 0$$

Je-li pak $ax + by + cz + d = 0$ rovnice roviny, pak pro jednotlivé koeficienty platí:

$$a = \det \begin{bmatrix} y_1 & z_1 & 1 \\ y_2 & z_2 & 1 \\ y_3 & z_3 & 1 \end{bmatrix} \qquad b = -\det \begin{bmatrix} x_1 & z_1 & 1 \\ x_2 & z_2 & 1 \\ x_3 & z_3 & 1 \end{bmatrix}$$

$$c = \det \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{bmatrix} \qquad d = -\det \begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{bmatrix}$$

Definujeme-li vektory **r**, **s** v prostoru E_3 (nikoliv v homogenních souřadnicích) jako

$$\mathbf{r} = \mathbf{x}_2 - \mathbf{x}_1 \qquad \text{a} \qquad \mathbf{s} = \mathbf{x}_3 - \mathbf{x}_1$$

pak lze určit normálu plochy jako

$$\mathbf{n} = \mathbf{r} \times \mathbf{s} = \det \begin{bmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ r_x & r_y & r_z \\ s_x & s_y & s_y \end{bmatrix}$$

$$\mathbf{n} = \det \begin{bmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ x_2 - x_1 & y_2 - y_1 & z_2 - z_1 \\ x_3 - x_1 & y_3 - y_1 & z_3 - z_1 \end{bmatrix}$$

kde **i**, **j**, **k** jsou jednotkové vektory ve směru osy x, y, z

s_x, s_y, s_z, resp. r_x, r_y, r_z jsou jednotlivé složky vektoru s, resp. **r**.

Lze ukázat, že pro normálu platí

$$\mathbf{n} = a\,\mathbf{i} + b\,\mathbf{j} + c\,\mathbf{k}$$

přičemž hodnota d může být získána prostým dosazením libovolného bodu **x** ležícího na rovině do rovnice roviny, neboť

$$ax + by + cz + d = 0$$

kde pouze d je neznámá, a lze ji tedy vyjádřit jako:

$$d = -\mathbf{n}^T \cdot \mathbf{x}$$

Vezmeme-li např. vrcholy 2, 3, 7 po transformaci, pak:

$$\mathbf{r} = [\ 0.232,\ -0.933,\ 3.475\ ]^T - [\ 1.232,\ -0.483,\ 1.802\ ]^T =$$

$$[\ -1,\ -0.45,\ 1.673\ ]^T$$

$$\mathbf{s} = [\ 0.232,\ 4.864,\ 5.028\ ]^T - [\ 1.232,\ -0.483,\ 1.802\ ]^T =$$

$$[-1\ ,\ 5.347\ ,3.226\ ]^T$$

Pak pro normálový vektor **n** dostáváme:

$$\mathbf{n} = \det \begin{bmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ r_x & r_y & r_z \\ s_x & s_y & s_y \end{bmatrix} = \det \begin{bmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ -1 & -0.45 & 1.673 \\ -1 & 5.347 & 3.226 \end{bmatrix}$$

Po vyčíslení

$$\mathbf{n} = [\ -10.3967\ ,\ 1.553\ ,\ -5.797\ ]^T$$

Nyní je nezbytné určit hodnotu koeficientu d, např. dosazením druhého vrcholu. Pak

$$d = -\ [\ -10.3967,\ 1.553,\ -5.797\ ]\ .\ [\ 1.232,\ -0.483,\ 1.802\ ]^T =$$
$$-\ (\ -12.809 - 0.75 - 10.45\ ) = 24.009$$

Výsledná rovnice roviny určené body 2., 3., 7. má pak tvar:

$$-\ 10.3967\ x\ +\ 1.553\ y\ -\ 5.797\ z\ +\ 24.009\ =\ 0$$

Pokud by rovnice roviny byla určena před pootočením a transformace rotace by byla aplikována na koeficienty vyjadřující danou rovinu, pak by rovnice roviny měla tvar:

$$-\ 0.866\ x\ +\ 0.129\ y\ -\ 0.483\ z + 2\ =\ 0$$

Lze nahlédnout, že obě výše uvedené rovnice reprezentují stejnou rovinu. Matice reprezentující daná tělesa po aplikaci transformace mají tvar:

$$^{(1)}\mathbf{V}' = {}^{(1)}\mathbf{V}\ .\ \mathbf{R}^{-1}(\vartheta,\phi) \qquad\qquad ^{(2)}\mathbf{V}' = {}^{(2)}\mathbf{V}\ .\ \mathbf{R}^{-1}(\vartheta,\phi)$$

Po pronásobení dostáváme

$$^{(1)}\mathbf{V}' = \begin{bmatrix} -0.866 & 0.866 & 0 & 0 & 0.5 & -0.5 \\ 0.129 & -0.129 & -0.966 & 0.966 & 0.224 & -0.224 \\ -0.483 & 0.483 & -0.259 & 0.259 & -0.837 & 0.837 \\ 2 & 0 & 6 & 0 & 3 & -1 \end{bmatrix}^T$$

$$^{(2)}\mathbf{V}' = \begin{bmatrix} -0.866 & 0.866 & 0 & 0 & 0.5 & -0.5 \\ 0.129 & -0.129 & -0.966 & 0.966 & 0.224 & -0.224 \\ -0.483 & 0.483 & -0.259 & 0.259 & -0.837 & 0.837 \\ 3 & -1 & 4 & -2 & 4 & 0 \end{bmatrix}^T$$

přičemž plochy jsou již správně orientovány.(Tuto orientaci musí zajistit řešitel.) Nyní je třeba nalézt plochy, které jsou zakryty jinými plochami téhož tělesa, je-li pozorovatel v bodě $[\ 0\ ,\ 0\ ,\ 1\ ,\ 0\ ]^T$. Pro testovací bod

$$\mathbf{x}_p = [\ 0\ ,\ 0\ ,\ -1\ ,\ 0\ ]^T$$

dostáváme:

$$^{(1)}\mathbf{v}' \ .\ \mathbf{x}_p = [\ 0.483,\ -0.483,\ 0.259,\ -0.259,\ 0.837,\ -0.837\ ]^T$$

$$^{(2)}\mathbf{v}' \ .\ \mathbf{x}_p = [\ 0.483,\ -0.483,\ 0.259,\ -0.259,\ 0.837,\ -0.837\ ]^T$$

Je tedy zřejmé, že plochy 2, 4, 6 z prvního tělesa a plochy 8, 10, 12 z druhého tělesa jsou neviditelné, neboť koeficienty ve vypočtených vektorech jsou záporné. Z tab.7.2.3 lze nalézt ty hrany, které jsou průsečnicemi neviditelných ploch a jsou též neviditelné. Plochy 2 a 4 mají společnou hranu 4. Analogicky lze určit, že hrany 1, 4, 9, 13, 16, 21 jsou neviditelné. Výsledek po odstranění neviditelných hran je znázorněn na obr.7.2.9.b.

Vzhledem k tomu, že plochy tělesa se protínají, musí být k ostatním podmínkám přidána podmínka $\alpha \geq 0$. Předpokládejme, že budou testovány hrany prvního tělesa vůči tělesu druhému a že vybereme hranu druhou, tj. spojnici vrcholu 2 a 3. Lze ukázat, že

$$\mathbf{x} = \mathbf{s} + \mathbf{d}.t =$$

$$[\ 1.232,\ -0.483,\ 1.802,\ 1\ ]^T - [\ -1,\ -0.45,\ 1.673,\ 0\ ]^T.t$$

Pak

$$\mathbf{p} = {}^{(2)}\mathbf{v}' \ .\ \mathbf{s} = [\ 1\ ,\ 1\ ,\ 4\ ,\ -2\ ,\ 3\ ,\ 1\ ]^T$$

$$\mathbf{q} = {}^{(2)}\mathbf{v}' \ .\ \mathbf{d} = [\ 0\ ,\ 0\ ,\ 0\ ,\ 0\ ,\ -2\ ,\ 2\ ]^T$$

Vzhledem k tomu, že pozorovatel je v bodě (x,y,∞) a tedy vektor pohledu je

$$\mathbf{g} = [\ 0\ ,\ 0\ ,\ 1\ ,\ 0\ ]^T$$

pak

$$\mathbf{w} = {}^{(2)}\mathbf{v}' \ .\ \mathbf{g} =$$

$$[-0.483\ ,\ 0.483\ ,\ -0.259\ ,\ 0.259\ ,\ -0.837\ ,\ 0.837\ ]^T$$

Test, zda daná úsečka je zcela viditelná, daný podmínkami:

$$w_j \leq 0 \quad \& \quad p_j \leq 0 \quad \& \quad p_j + q_j \leq 0$$

není splněn pro žádnou rovinu. Je tedy nutné nalézt zakryté části. Rozepsáním vektorové nerovnice **h** > **0** dostáváme pro jednotlivé složky vztahy:

$$
\begin{aligned}
1 \qquad\quad & - 0.483\,\alpha > 0 \\
1 \qquad\quad & + 0.483\,\alpha > 0 \\
4 \qquad\quad & - 0.259\,\alpha > 0 \\
-2 \qquad\quad & + 0.259\,\alpha > 0 \\
3 - 2t \quad & - 0.837\,\alpha > 0 \\
1 + 2t \quad & + 0.837\,\alpha > 0
\end{aligned}
$$

Řešení této soustavy neexistuje, a tedy zvolená úsečka je zcela viditelná, neboť neexistuje žádná část úsečky, která by byla zakryta druhým tělesem. Výsledky testů pro hrany prvního tělesa vůči tělesu druhému jsou uvedeny v tab.7.2.4 a tab.7.2.5 Poznamenejme ještě, že vektory **g** a **w** jsou konstantní a hrany 1, 4, 9 jsou již vynechány, neboť jsou zcela neviditelné.

Hrana 10 je neviditelná pro $t \in \langle 0.244, 0.667 \rangle$, tj. od bodu (1.232,0.815,2.150) do bodu (1.232,3.381,2.837). Hrana 11 je zakryta pro $t \in \langle 0.282, 0.667 \rangle$, tj. od bodu (0.232,0.703,3.913) do bodu (0.232,2.933,4.510).

Hodnota $\alpha = 0$ určuje body, kde hrana těleso protíná, což odpovídá hodnotě parametru $t = 0.333$ a $t = 0.667$ pro obě hrany. Tyto hodnoty t odpovídají bodům (1.232,1.449,2.320) a (1.232,3.381,2.837) pro hranu 10 a bodům (0.232,1.001,3.993) a (0.232,2.933,4.510) pro hranu 11. Všechny čtyři body je nutné uložit jako body určující protnutí druhého tělesa hranami tělesa prvního.

hrana	vrcholy	$\mathbf{s}^T$	$\mathbf{d}^T$
2	2 - 3	[1.232,-0.483,1.802,1]	[-1,-0.45,1.673,0]
3	3 - 4	[0.232,-0.931,3.46,1]	[-1.732,0.259,-0.966,0]
5	5 - 6	[-0.5,5.571,2.389,1]	[1.732,-0.259,0.966,0]
6	6 - 7	[1.232,5.313,3.355,1]	[-1,-0.448,1.673,0]
7	7 - 8	[0.232,4.864,5.028,1]	[-1.732,0.259,-0.966,0]
8	8 - 5	[-1.5,5.123,4.062,1]	[1,0.448,-1.673,0]
10	2 - 6	[1.232,-0.483,1.802,1]	[0,5.796,1.553,0]
11	3 - 7	[0.232,-0.931,3.475,1]	[0,5.796,1.553,0]
12	4 - 8	[-1.5,-0.672,2.510,1]	[0,5.796,1.553,0]

Tabulka 7.2.4

hrana	vrcholy	$\mathbf{p}^T$	$\mathbf{q}^T$	poznámka
2	2 - 3	[1,1,4,-2,3,1]	[0,0,0,0,-2,2]	(1)
3	3 - 4	[1,1,4,-2,1,3]	[2,-2,0,0,0,0]	(1)
5	5 - 6	[3,-1,-2,4,3,1]	[-2,2,0,0,0,0]	(2)
6	6 - 7	[1,1,-2,4,3,1]	[0,0,0,0,-2,2]	(2)
7	7 - 8	[1,1,-2,4,1,3]	[2,-2,0,0,0,0]	(2)
8	8 - 5	[3,-1,-2,4,1,3]	[0,0,0,0,-2,2]	(2)
10	2 - 6	[1,1,4,-2,3,1]	[0,0,-6,6,0,0]	(3)
11	3 - 7	[1,1,4,-2,1,3]	[0,0,-6,6,0,0]	(4)
12	4 - 8	[3,-1,4 -2,1,3]	[0,0,-6,6,0,0]	(1)

(1) úplně viditelná - úplným řešením

(2) úplně viditelná $w_3 < 0 \quad p_3 < 0 \quad p_3 + q_3 < 0$

(3) hrana je zakryta pro $t \in \langle 0.244, 0.667 \rangle$, obr.7.2.9.a.

(4) hrana je zakryta pro $t \in \langle 0.282, 0.667 \rangle$, obr.7.2.9.b.

Tabulka 7.2.5

Porovnáme-li analogicky hrany druhého tělesa vůči tělesu prvému, pak pouze hrana 17 je částečně neviditelná. Tab.7.2.6 a tab.7.2.7 ukazují výsledky jednotlivých kroků.

hrany	vrcholy	$\mathbf{s}^T$	$\mathbf{d}^T$
14	10 - 11	[2.598,1.544,1.967,1]	[-2.0,-0.897,3.346,0]
15	11 - 12	[0.598,0.647,5.313,1]	[-1.732,0.259,-0.966,0]
17	13 - 14	[0.866,3.734,1.518,1]	[1.732,-0.259,0.966,0]
18	14 - 15	[2.598,3.475,2.484,1]	[-2.0,-0.897,3.346,0]
19	15 - 16	[0.598,2.579,5.830,1]	[-1.732,0.259,-0.966,0]
20	16 - 13	[-1.134,2.838,4.864,1]	[2.0,0.897,-3.346,0]
22	10 - 14	[2.598,1.544,1.967,1]	[0,1.932,0.518,0]
23	11 - 15	[0.598,0.647,5.313,1]	[0,1.932,0.518,0]
24	12 - 16	[-1.134,0.906,4.347,1]	[0,1.932,0.518,0]

Tabulka 7.2.6

hrany	vrcholy	$\mathbf{p}^T$	$\mathbf{q}^T$	pozn.
14	10 - 11	[-1,3,4,2,3,-1]	[0,0,0,0,-4,4]	(1)
15	11 - 12	[-1,3,4,2,-1,3]	[2,-2,0,0,0,0]	(2)
17	13 - 14	[1,1,2,4,3,-1]	[-2,2,0,0,0,0]	(3)
18	14 - 15	[-1,3,2,4,3,-1]	[0,0,0,0,-4,4]	(1)
19	15 - 16	[-1,3,2,4,-1,3]	[2,-2,0,0,0,0]	(2)
20	16 - 13	[1,1,2,4,-1,3]	[0,0,0,0,4,-4]	(4)
22	10 - 14	[-1,3,4,2,3,-1]	[0,0,-2,2,0,0]	(1)
23	11 - 15	[-1,3,4,2,-1,3]	[0,0,-2,2,0,0]	(1)
24	12 - 16	[1,1,4,2,-1,3]	[0,0,-2,2,0,0]	(2)

(1) zcela viditelná $w_1 < 0 \quad p_1 < 0 \quad p_1 + q_1 < 0$

(2) zcela viditelná $w_5 < 0 \quad p_5 < 0 \quad p_5 + q_5 < 0$

(3) částečně zakrytá $t \in \langle 0 , 0.211 \rangle$

(4) protínající hrana zakryta $t \in (0.25 , 1.0 \rangle$

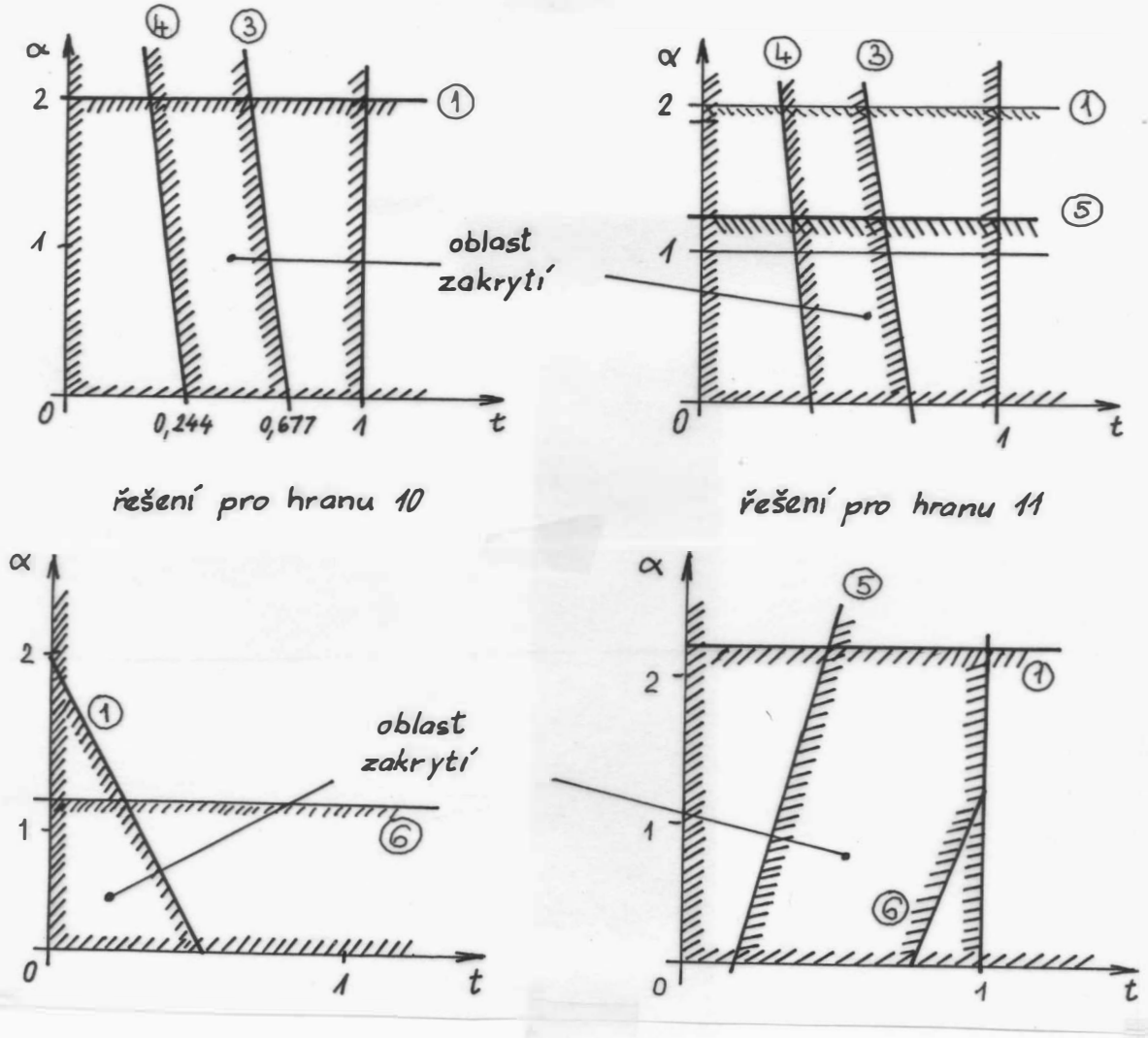
Tabulka 7.2.7

Z tab.7.2.7 vyplývá, že hrana 17 je zakryta pro hodnoty $t \in \langle 0,0.211 \rangle$, což odpovídá bodům (0.866,3.734,1.518) a (1.232,3.679,1.722). Hrana 20 protíná přední stěnu prvého tělesa pro t = 0.25. Je tedy zakryta pro $t \in \langle 0.25,1.0 \rangle$, což odpovídá úsečce z bodu (-0.634,3.062,4.028) do bodu (0.866,3.734,1.518), viz obr.7.2.10.b. Bod (-0.634,3.062,4.028) je uložen do seznamu bodů, v nichž hrana protíná nějakou stěnu. Řešením dané soustavy dostáváme ještě pro t=0.75 a α=0 další bod, kde hrana protíná plochu a který má souřadnice (0.366,3.511,2.355).

Vypočtené body, kde hrany protínají plochu, lze reprezentovat pomocí matice $\hat{\mathbf{x}}$.

$$\hat{\mathbf{x}} = \begin{bmatrix} 1.232 & 1.232 & 0.232 & 0.232 & -0.634 & 0.366 \\ 1.449 & 3.381 & 1.001 & 2.933 & 3.062 & 3.511 \\ 2.320 & 2.837 & 3.993 & 4.510 & 4.028 & 2.355 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

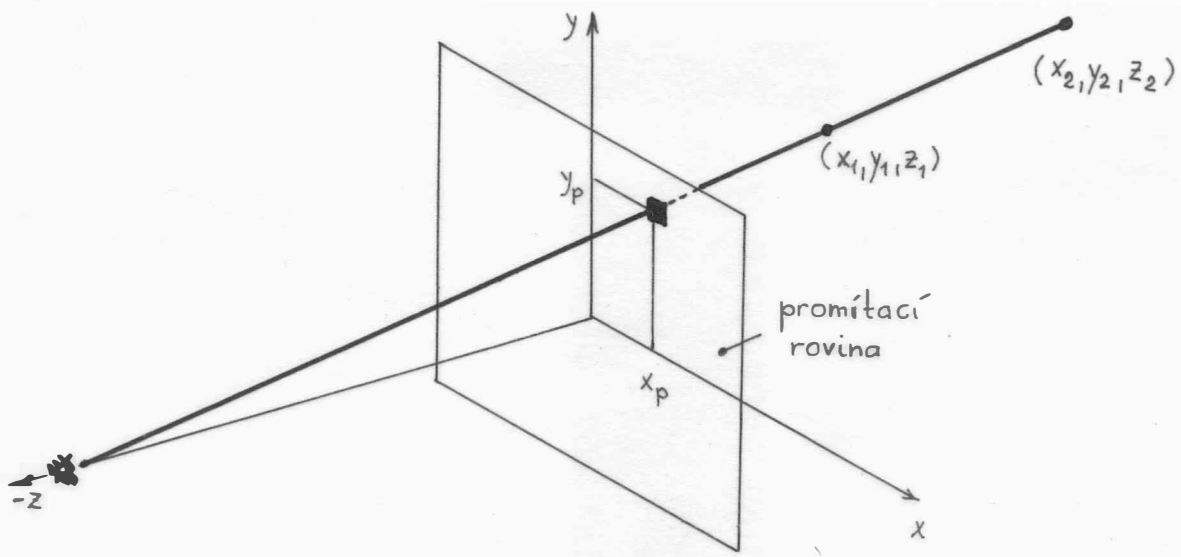
Nalezení hran, které vznikají jako průsečnice rovin dvou ploch, lze jednoduchým způsobem určit tak, že každý bod spojíme s každým (takto získáme 15 hran), přičemž aplikujeme analogický postup jako na hrany dané při zadání těles. Výsledek úplného testu (řešení je ponecháno na čtenáři) je uvedeno na obr.7.2.10.c.


α
2
1
0
0,244
0,677
1
t
4
3
1
oblast
zakrytí
řešení pro hranu 10
5
6
řešení pro hranu 11

Obr. 7.2.10

7.3 Z - buffer

Jedním z nejjednodušších algoritmů pro řešení viditelnosti je tzv. z-buffer algoritmus, někdy též nazývaný hloubkový buffer (depth-buffer). Jde v podstatě o zobecnění principu frame-bufferu, neboť kromě vlastní reprezentace obrazu je navíc používáno jedno pole, jehož obsah určuje vzdálenost nejbližšího bodu v prostoru, který se promítne na danou pozici promítací roviny, viz obr.7.3.1.



Obr.7.3.1

Zaznamenává se tedy nejen intenzita, resp. barva, jednotlivých pixelů dané scény, ale též nejmenší dosažená vzdálenost od pozorovatele. Obrovskou výhodou tohoto algoritmu je jeho jednoduchost, která umožňuje řešit všechny možné komplikace (složité tvary ploch, dotyky ploch a jejich průsečíky) velmi jednoduše, přičemž výpočtová složitost je lineární vzhledem k počtu zpracovávaných n-úhelníků, tj. o(n). Cenou za jednoduchost algoritmu je však velká paměťová náročnost. Vzhledem ke způsobu zpracování je čas potřebný ke zpracování každého n-úhelníka neúměrně velký při použití běžných hardwarových prostředků.

Označíme-li I(x,y) intenzitu či barvu pixelu na pozici (x,y) a Z(x,y) vzdálenost nejbližšího bodu zpracované scény od pozorovatele, který se promítne do bodu (x,y), lze celý algoritmus zapsat např. pomocí alg.7.3.1.

1. Nastav I(x,y) ∀x,y na barvu pozadí (I(x,y) může být přímo frame-buffer obrazovky) a ∀x,y nastav Z(x,y) na největší možnou vzdálenost.
2. Pro všechny pixely,na které se zobrazí daný n-úhelník:
   a) vypočti z-ovou souřadnici,
   b) je-li hodnota z < Z(x,y), pak nastav Z(x,y) na hodnotu z a nastav pixel I(x,y) na hodnotu odpovídajícího jasu či barvy bodu (x,y).

Algoritmus 7.3.1

Nespornou výhodou algoritmu je, že nepotřebuje žádné dodatečné struktury pro ukládání dat. Potřebuje pouze informace o zpracovávaném n-úhelníku. To znamená, že počet n-úhelníků může být v zásadě libovolný. Na druhé straně je však dosud nutné mít na zřeteli vysoké paměťové nároky.

Označíme-li:

r_x , resp. r_y, rozlišovací schopnost na výstupu ve směru osy x, resp. y,

n_i počet bytů nutných k reprezentaci jedné hodnoty barvy či intenzity

n_r počet bytů nutných k reprezentaci jedné hodnoty v Z-bufferu, pak celková požadovaná kapacita paměti je dána vztahem:

$$M = r_x \cdot r_y \cdot (n_i + n_r)$$

Pro $r_x = r_y = 1024$, $n_i = 1$, $n_r = 5$ dostáváme:

$$M = 1024 \cdot 1024 \cdot (1 + 5) = 6 \text{ MB}$$

Je tedy zřejmé, že uvedené paměťové nároky povedou prozatím k segmentaci obrazu nebo použití vnější paměti. Nicméně hardwarová realizace je velmi jednoduchá.

Vzhledem ke způsobu zpracování je nezbytné co nejrychleji pro danou hodnotu (x,y) určit z-ovou souřadnici. V případě, že je pro každý n-úhelník k dispozici rovnice odpovídající roviny ve tvaru:

$$\mathbf{n}^T \cdot \mathbf{x} - d = 0$$

tj.

$$a\,x + b\,y + c\,z + d = 0$$

pak pro $c \neq 0$ lze psát:

$$z = - (a x + b y + d) / c$$

Použijeme-li myšlenky řádkové konverze, tj. hodnota y je konstantní pro daný řádek, lze ze znalosti rovnice roviny, na níž zpracovávaný n-úhelník leží, a souřadnice prvního pixelu (x_1, y_1, z_1) určit souřadnici z takto:

$$z = - (a . (x_1 + \Delta x) + b . y + d) / c$$

Je-li $\Delta x = 1$, pak po dosazení a úpravě dostáváme:

$$z = z_1 - a / c$$

Z uvedeného vyplývá, že výpočet z-ové souřadnice je velmi jednoduchý.

Příklad$_1$

Aplikujte uvedený postup na kvadratické plochy $\mathbf{x}^T \mathbf{A} \mathbf{x} = 0$

Rozepsáním do složek dostáváme:

$$a_{11}x^2 + a_{22}y^2 + a_{33}z^2 + 2a_{12}xy + 2a_{23}yz + 2a_{13}xz + 2a_{14}x + 2a_{24}y + 2a_{34} + a_{44} = 0$$

Pro kouli obecně platí:

$$(x - x_s)^2 + (y - y_s)^2 + (z - z_s)^2 - r^2 = 0$$

Vzhledem k tomu, že hodnota y se pro daný řádek nemění, lze psát:

$$(x - x_s)^2 + k^2 + (z - z_s)^2 - r^2 = 0$$

kde $k = y - y_s$ je konstanta pro daný řádek.

Pak pro body $\mathbf{x}_0$ a $\mathbf{x}_1$ lze psát:

$$(x_0 - x_s)^2 + k^2 + (z_0 - z_s)^2 - r^2 = 0$$

$$(x_1 - x_s)^2 + k^2 + (z_1 - z_s)^2 - r^2 = 0$$

Je-li $\Delta x = 1$, tj. $x_1 - x_0 = 1$, pak odečtením rovnic dostaneme, že:

$$z_1^2 = z_0^2 + 2 x_0 - 1$$

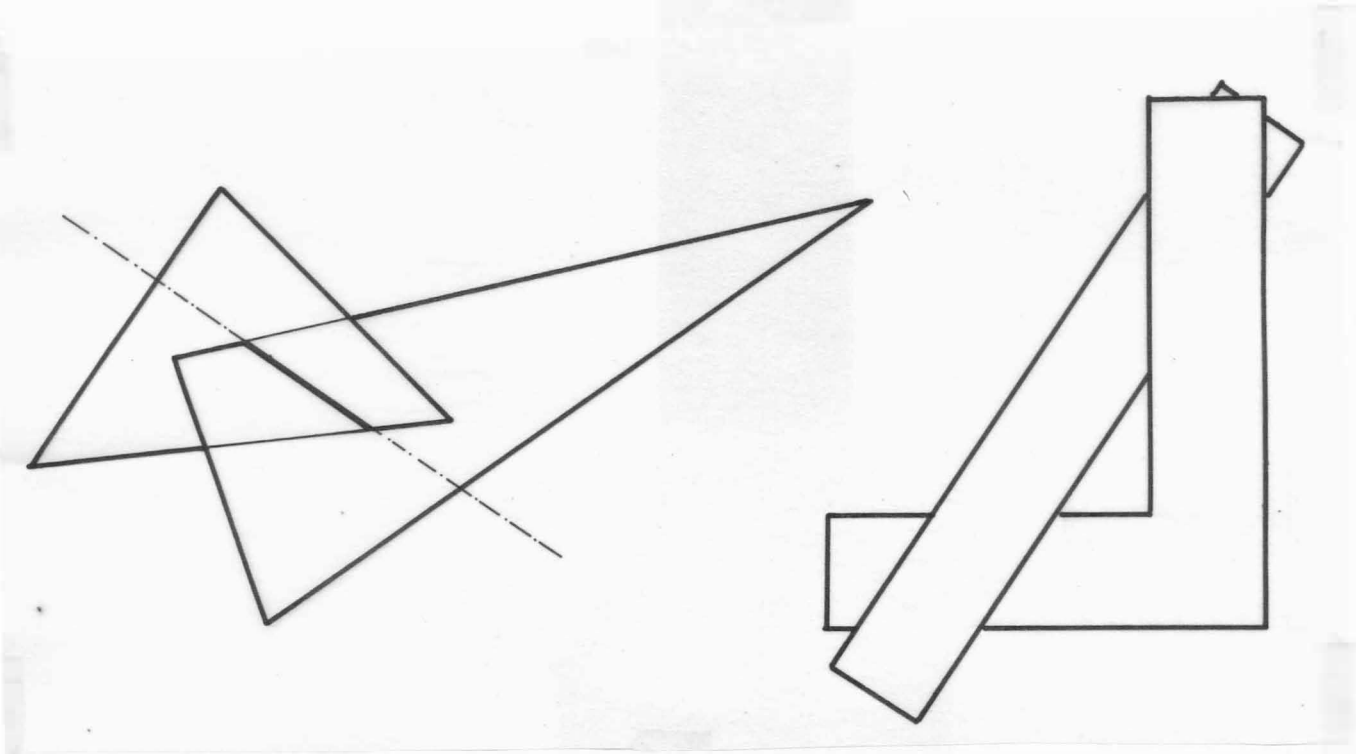
Analogicky pro krok n+1 dostáváme:

$$z_{n+1}^2 = z_n^2 + 2 x_n - 1$$

Vzhledem k monotónnosti lze v některých případech nahradit vzdálenost jinou metrikou a vyhnout se tak použití funkce odmocniny.

7.4 Algoritmus malíře

Paměťové nároky předcházejícího algoritmu lze podstatně zredukovat, pokud by bylo možné určit vhodné pořadí vykreslování jednotlivých ploch takové, aby postupným překreslováním vzdálenějších ploch plochami bližšími k pozorovateli byl získán výsledný očekávaný výstup. Tento postup se nazývá algoritmus malíře (**painter's algorithm**, resp. **list priority**). Je zřejmé, že tento algoritmus vyžaduje pouze paměť určenou k reprezentaci obrazu, která je podstatně menší, než v případě algoritmu z-bufferu. Snížení požadavků na paměť však vede k požadavku určení správného pořadí zobrazování ploch. V případě, že se plochy protínají (obr.7.1.4.a) nebo se zakrývají navzájem (obr.7.4.1.b), je nutné přistoupit k dělení ploch, což vede k jisté komplikaci jinak jednoduchého algoritmu.



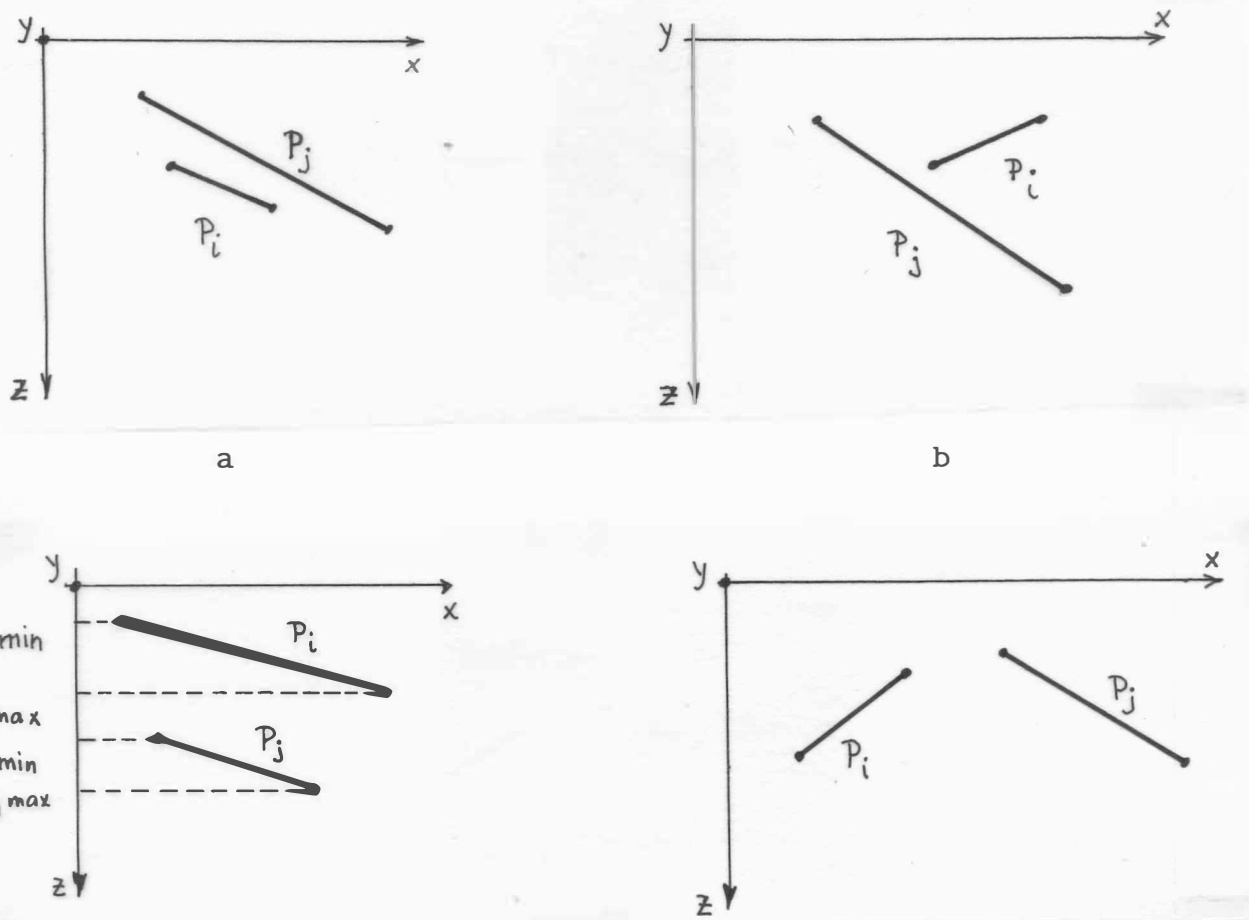
a b

Obr.7.4.1

Pokud není nutné přistoupit k dělení ploch, pak lze plochy v zásadě setřídit podle z-ových souřadnic, přičemž kritériem pro třídění je porovnání z-ových souřadnic a výsledek testu překrytí (tzv. overlap test).

Problém lze pak rozdělit na následující kroky prováděné v uvedeném pořadí (v obr.7.4.2 předpokládáme plochy rovnoběžné s rovinou xy):

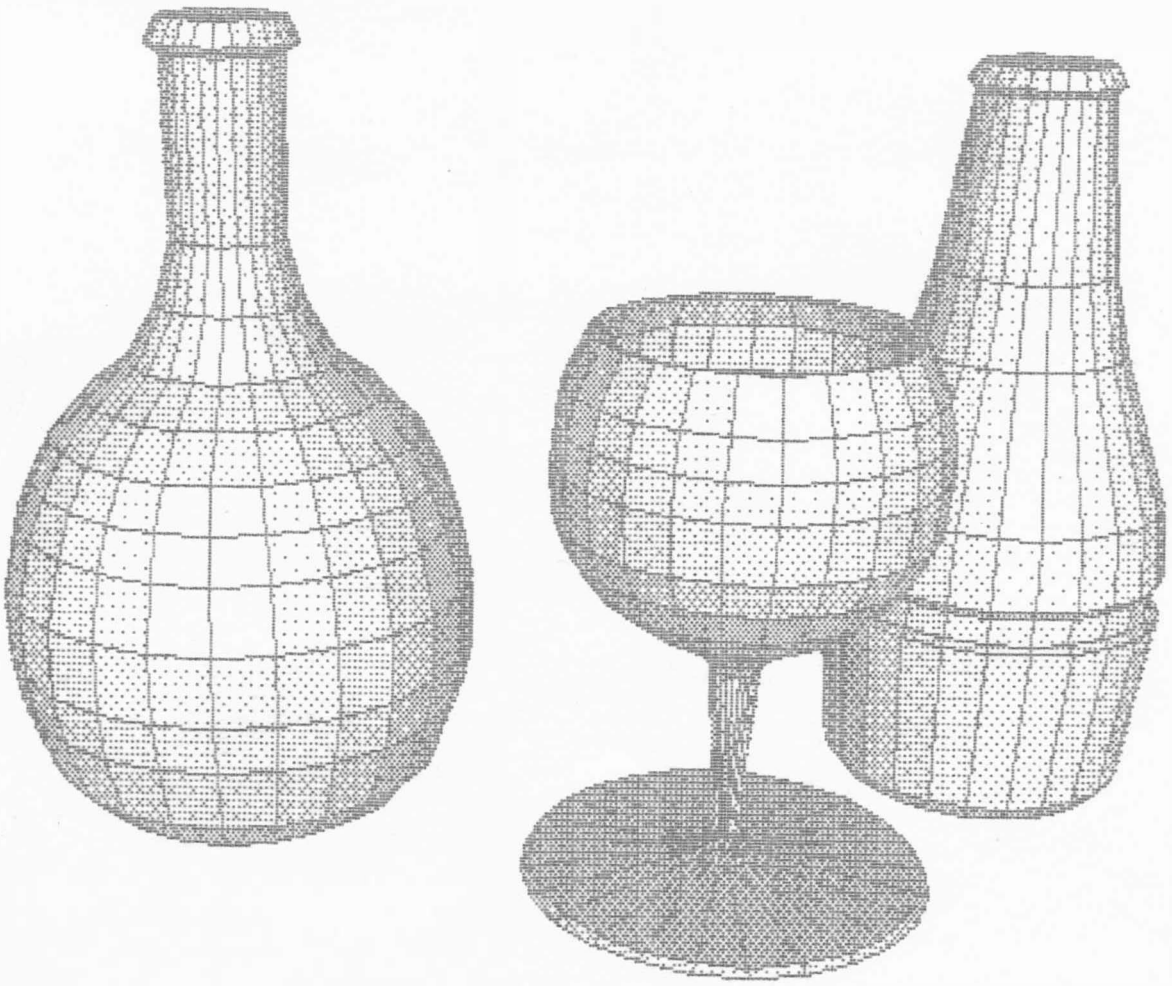
1. Porovnej plochy P_i a P_j. Je-li $z_{i_{max}} < z_{j_{min}}$, pak plocha P_i se bude kreslit před plochou P_j, viz obr.7.4.2.a.
2. V případě, že plochy P_i a P_j se nepřekrývají v průmětně (xy), pak jejich pořadí je libovolné.
3. Jsou-li všechny vrcholy ploch P_i na jedné straně od roviny, ve které leží plocha P_j, lze určit pořadí kreslení ploch, viz obr.7.4.2.c a 7.4.2.d.
4. V případě, že výše uvedené testy nedovolují rozhodnout jednoznačně pořadí, pak je nezbytné nalézt průsečík hran daných ploch v průmětně (hrany v prostoru mohou obecně být mimoběžkami) a pomocí souřadnic z zdánlivého průsečíku určit pořadí kreslení ploch.



Obr. 7.4.2

Ke zrychlení zpracování lze využít testů či informací, např.:

- konvexní n-úhelníky, které mají společnou hranu, se nemohou protínat
- aplikací **min - max** testu, který spočívá v tom, že daný n-úhelník ohraničíme obdélníkem, jehož hrany jsou rovnoběžné s osami souřadného systému. Testy provádíme nejdříve nad ohraničujícími obdélníky a v případě, že se protínají, testujeme vlastní n-úhelníky.



Ukázka výsledku algoritmu a použitím techniky vzorů (kap. 8.1)
Obr. 7.4.2

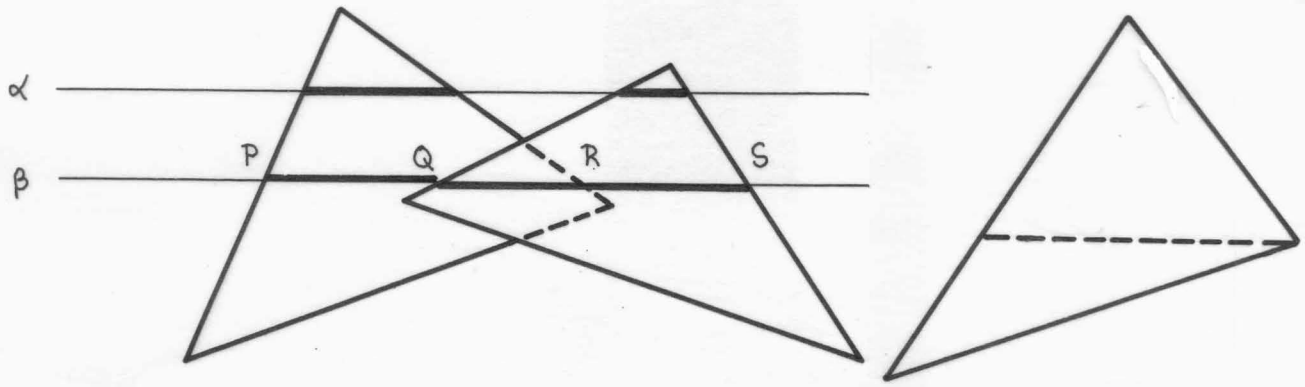
7.5 Algoritmy řádkové konverze

Algoritmy založené na řádkové konverzi operují v obrazovém prostoru a jejich různé modifikace byly popsány např. v [19], [139]. V podstatě používají reprezentace n-úhelníků, které jsou již nyní obecně v prostoru, pomocí tzv. řádkové konverze, viz kap.3, přičemž je nutné modifikovat datové struktury tak, aby obsahovaly též informace o souřadnici z. Jednotlivé plochy ve scéně jsou obvykle reprezentovány pomocí konvexních, resp. trojúhelníkových ploch. V zásadě je možné algoritmy řádkové konverze modifikovat i pro obecné parametrické plochy, viz [16].

Z hlediska nároků na paměť a třídění lze říci, že algoritmy řádkové konverze jsou jistým kompromisem mezi algoritmem z-bufferu a algoritmem malíře.

Algoritmy založené na řádkové konverzi používají obvykle pro tabulku hran (tabulka ET) datovou strukturu obsahující:

- souřadnici x vrcholu s nejmenší souřadnicí y
- souřadnici y druhého vrcholu hrany
- inverzní směrnici (převrácená hodnota) dané hrany
- odkaz na n-úhelník, ke kterému hrana přináleží.



a b

Obr.7.5.1

Kromě tabulky hran je pak ještě nutná tabulka n-úhelníků PT obsahující:

- koeficienty rovnice roviny, v níž n-úhelník leží
- informace o barvě, resp. jasu pro výpočet barvy, resp. intenzity

- informaci udávající, zda daný n-úhelník je aktivní při kreslení části daného řádku.

Je zřejmé, že při kreslení řádku α žádné komplikace nenastávají a práce s tabulkou hran a seznamem aktivních hran je obdobná postupu v kap. 3. V případě zobrazování řádku β však již přímé použití vede k jistým komplikacím, neboť pokud by nebyla kontrolována příslušnost hrany k n-úhelníku, pak by výsledkem byly úsečky PQ a RS, což neodpovídá skutečnosti, přičemž při uvažování barvy, resp. intenzity by výsledky byly zcela matoucí. Je tedy nutné, aby v rámci dané datové struktury bylo možné najít další hranu příslušející k danému n-úhelníku. Pak lze určit úsečku PR, barvu, resp. intenzitu odpovídajících pixelů, přičemž aktivaci pixelů daného n-úhelníka ukončíme v bodě Q, neboť hrana procházející bodem Q není součástí zobrazovaného n-úhelníka.

Jednoduchou modifikací datových struktur a algoritmu dostáváme tzv. **řádkový z-buffer** algoritmus [109], který určuje viditelnou část n-úhelníků v daném řádku pomocí algoritmu z-bufferu. Datová struktura obsahuje pak pro každý pár hran n-úhelníka informace:

x_l levý průsečík n-úhelníka s daným řádkem
x_r pravý průsečík n-úhelníka s daným řádkem
Δx_l přírůstek v ose x pro levý průsečík pro následující řádek
Δx_r přírůstek v ose x pro pravý průsečík pro následující řádek
z_l hloubka n-úhelníka ve středu pixelu odpovídajícího bodu x_l
Δz_x přírůstek z v rámci řádky $\Delta z_x = -a/c \quad c \neq 0$
Δz_y přírůstek z při přechodu na následující řádku

Pro zjednodušení operací, které je nutno realizovat pro každý řádek, je vhodné daný n-úhelník rozdělit tak, aby dvojice hran protínající určitou množinu řádků měly stejné počáteční a koncové řádky, viz obr. 7.5.1.b. Pak je nutné udržovat seznam aktivních hran uspořádaný jen vzhledem k hodnotě x_l.

Výhodou uvedeného postupu je pak menší náročnost na paměť, neboť je nutné reprezentovat z-buffer pouze pro jeden řádek. Jistou výhodou algoritmu založeného na řádkové konverzi je jeho možná modifikace pro případ řešení problému zakrytých hran. Lze jej také modifikovat pro případ parametricky zadaných ploch, tj.

$$x = x(u, v) \qquad y = y(u, v)$$

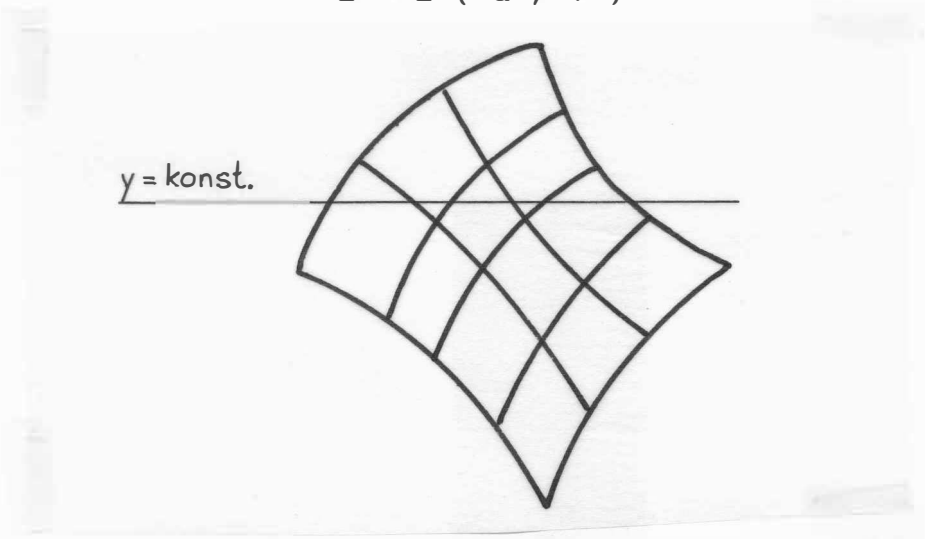
$$z = z(u, v)$$

Pro daný řádek y = konst lze určit řešením rovnic hodnoty u a v, neboť platí, že:

$$u = u(x, y) \qquad v = v(x, y)$$

přičemž hloubku z lze obdržet vyhodnocením rovnice

$$z = z(u, v)$$



Obr. 7.5.2

Jistou nevýhodou je, že uvedené rovnice je nezbytné řešit numericky, viz [16], [143], přičemž je nutné numerické metody volit opatrně vzhledem ke stabilitě řešení. Algoritmus lze pak hrubě zapsat takto:

```
pro každý řádek y
    pro každý pixel x na řádku
        pro každou plochu protínající řádek v x:
            Řeš rovnice:
                        u = u ( x , y )     v = v ( x , y )
            Vypočti hloubku pomocí:
                        z = z ( u , v )
            Urči viditelnost plochy v bodě (x,y) a zobraz jej
```

Algoritmus 7.5.1

Řešení uvedeného problému je možné i pomocí jiných principů, viz [150], které se víceméně blíží technice postupného dělení plochy vycházejícího z vlastností Warnockova algoritmu.

7.6 Warnockův algoritmus

Warnockův algoritmus, viz [94], [138], je založen na strategii **rozděl a panuj** (divide and conquer) a je základním principem mnoha algoritmů, viz [150], [142] atd. Algoritmus klade důraz na výsledné zobrazení scény místo na přesné určení vztahů na scéně, a operuje tedy v prostoru obrazu. Proces zobrazování je chápán jako rekurzivní, přičemž se rozpoznávají čtyři základní stavy, a to:

1) pokud všechny n-úhelníky jsou zcela vně zobrazované oblasti (disjoint),pak zobrazovaná oblast je prázdná a zobrazí se s barvou pozadí bez dalšího dělení, viz obr.7.6.1.a

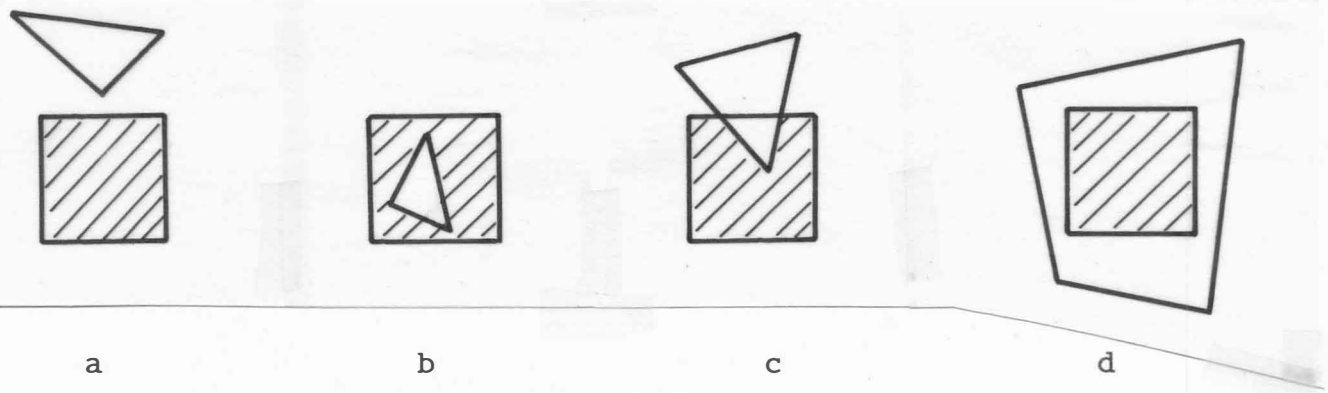
2) pokud oblast obsahuje právě jeden celý n-úhelník (contained), pak se oblast zobrazí s barvou pozadí, daný n-úhelník se zobrazí a vyplní odpovídající barvou, viz obr.7.6.1.b

3) pokud hranici oblasti protíná právě jeden n-úhelník (intersecting), pak se oblast zobrazí s barvou pozadí a příslušná část n-úhelníka, která je uvnitř zobrazované oblasti, se vykreslí s odpovídající barvou, viz obr.7.6.1.c

4) pokud zobrazovaná oblast je uvnitř právě jednoho n-úhelníka (surrounding), tj. není žádný jiný n-úhelník, který by byl uvnitř zobrazované oblasti, pak se zobrazovaná oblast vyplní barvou n-úhelníka, který jej "obklopuje", viz obr.7.6.1.d

5) pokud je nalezen alespoň jeden "obklopující" n-úhelník a je nejblíže pozorovateli ze všech n-úhelníků incidujících s danou oblastí, pak se zobrazovaná oblast vyplní barvou tohoto n-úhelníka

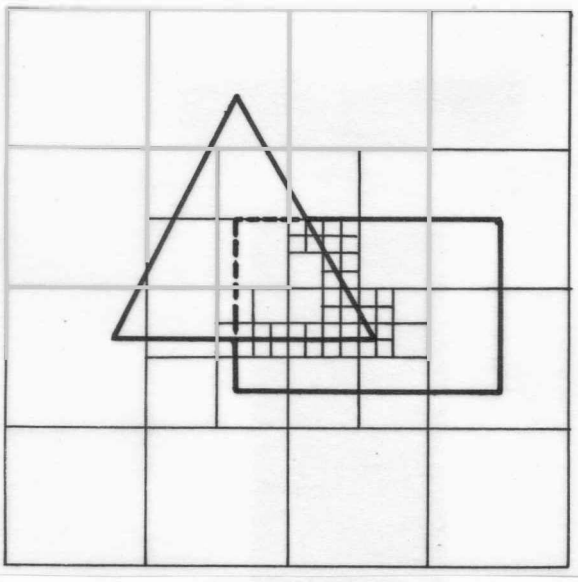
6) v ostatních případech se zobrazovaná oblast rozdělí.



a b c d

Obr.7.6.1

Je zřejmé, že uvedený algoritmus musí mít určenou maximální možnou hloubku rekurze, tj. možný počet dělení dané oblasti. Maximální počet dělení je většinou dán rozlišovací schopností obrazovky, tj. pro rozlišovací schopnost 1024 x 1024 je roven 10, pokud oblast bude dělena na čtyři stejně velké podoblasti, viz obr.7.6.2. (Je možné snadno realizovat antialiasing výpočtem obrazu ve větším adresovatelném prostoru a poté aplikovat dříve uvedené techniky antialiasingu.)



Obr.7.6.2

Za předpokladu, že zobrazovací plocha je čtvercová, lze Warnockův algoritmus popsat algoritmem 7.6.1. Zobrazovací plocha je pak reprezentována trojicí hodnot $\langle x , y , V \rangle$, kde:

x , y je souřadnice počátku

V je velikost oblasti

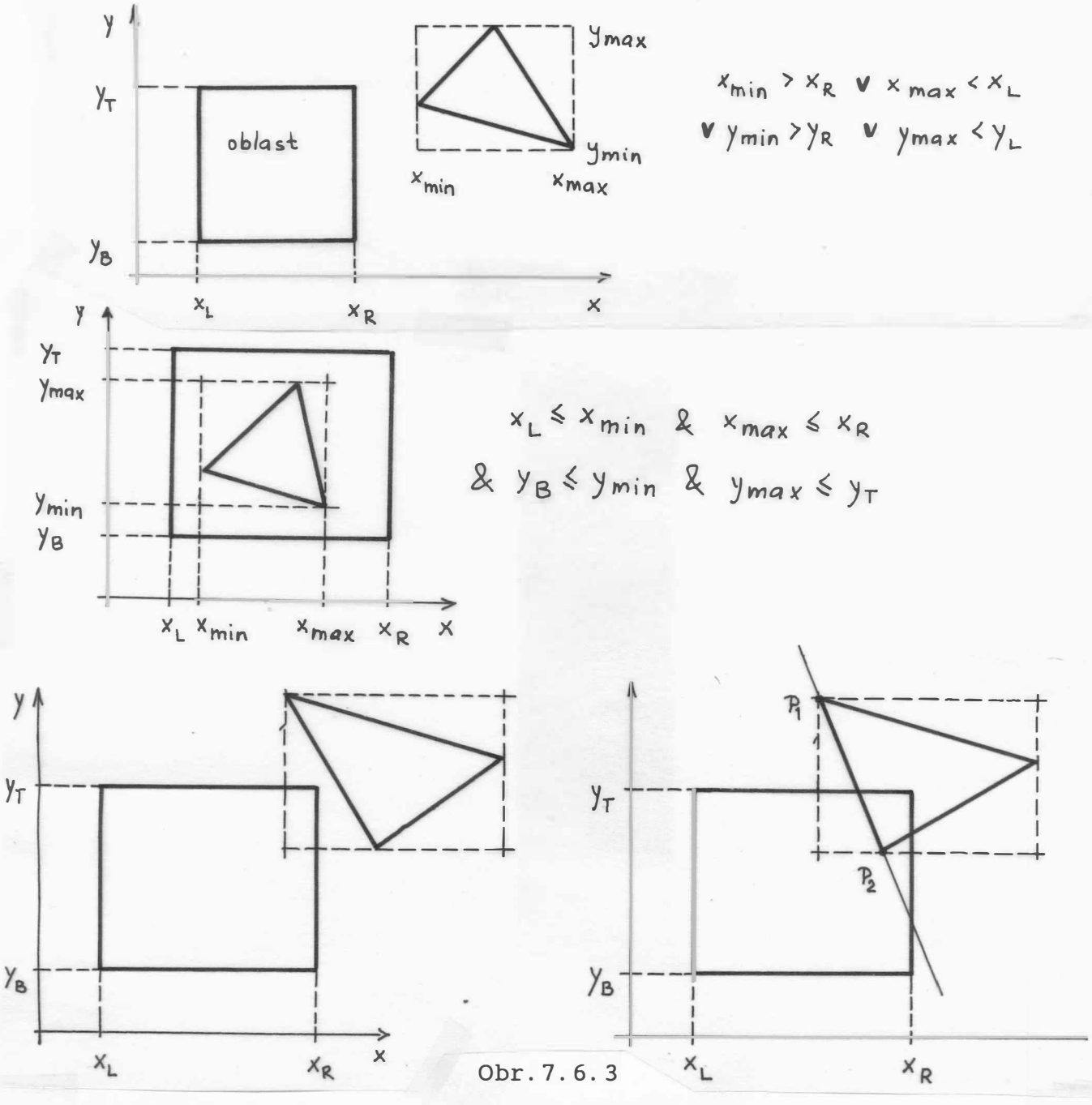
Na obr.7.6.3 jsou uvedeny testy pro základní případy, které mohou nastat při zkoumání vzájemné polohy zobrazované oblasti a zkoumané plochy. Případy obr.7.6.3.a, obr.7.6.3.b jsou poměrně jednoduché, neboť stačí vyhodnotit příslušné podmínky. Případy obr.7.6.3.c, obr.7.6.3.d nelze již rozhodnout jednoduchým způsobem. Je možné je rozlišit pouze tak, že se určí rovnice přímky P_1P_2 ve tvaru:

$$A x + B y + C = 0$$

tj. ve vektorovém tvaru

$$\mathbf{x}^T . \mathbf{a} = 0 \qquad \text{kde} \quad \mathbf{x} = [x , y , 1]^T \qquad \mathbf{a} = [A , B , C]^T$$

Pak pro všechny vrcholy definující zobrazovanou oblast musí mít výraz $\mathbf{x}^T$. a stejné znaménko, jde-li o případ ad d.



Obr. 7.6.3

Princip Warnockova algoritmu, viz alg. 7.6.1, byl též modifikován pro použití neplanárních ploch, viz např. [23].

Dosud uvedené základní algoritmy byly v dostupné literatuře popsány v různých modifikacích při využití rozličných datových struktur. Důvodem odvození různých modifikací byla především vysoká cena paměťových prvků a rychlost výpočetních systémů. V současné době, kdy hustota integrace je nepoměrně vyšší než

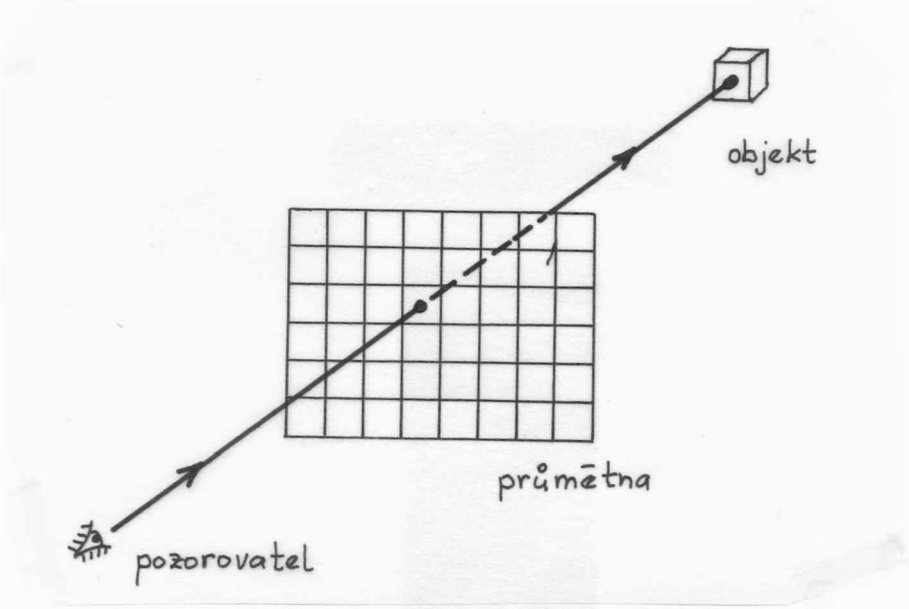
před 10 lety a je možné realizovat speciální procesory, vystupují do popředí techniky založené na "brutální síle", vycházející vlastně z myšlenky z-bufferu. Vzhledem k neustálému růstu požadavků na věrnost zobrazované scény se vyvíjejí techniky založené na "sledování paprsku" a techniky založené na radiačním výpočtu vyzařovaných energií.

```
{ pro jednoduchost algoritmu nejsou zahrnuty případy  }
{ ad 2), ad 3) a ad 4), neboť pouze výpočet urychlují }
x:=0; y:=0; V:=1024;  { N počet n-úhelníků }
PUSH W ( x , y , V );
{ ulož do zásobníku velikost zpracovávané plochy }
repeat
  POP W ( x , y , V );
  { vyber ze zásobníku velikost zpracovávané plochy }
  flag := true;  i:=1;
  while ( i <= N ) and flag do
  begin if n-úhelník_i není zcela vně zkoumané oblasti
        { test vůči obdélníku - neúplný test }
        then flag := false;
        i := i + 1
  end;
  if flag
    then { všechny n-úhelníky jsou vně zkoumané oblasti }
       zobraz oblast s barvou, resp. intenzitou pozadí
    else
      if V > 1
        then { velikost oblasti je větší než 1 }
        begin V := V div 2;
              PUSH W ( x , y , V );  PUSH W ( x+V , y , V );
              PUSH W ( x , y+V , V );PUSH W ( x+V , y+V , V );
        end
        else { velikost zobrazované plochy je rovna 1 }
          if pixel není součástí žádného n-úhelníka
            then zobraz pixel s barvou, resp. jasem pozadí
            else zobraz pixel s barvou, resp. jasem odpovídající
                 nejbližšímu n-úhelníku
until zásobník prázdný
```

Algoritmus 7.6.1

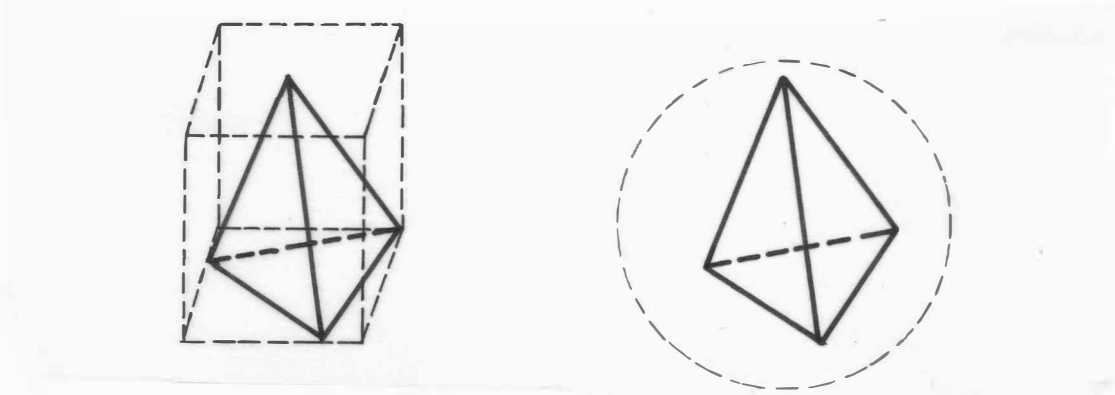
7.7 Algoritmy sledování paprsku

Princip algoritmu sledování paprsku (Ray-tracing) je založen na myšlence hledání takového průsečíku paprsku s tělesem, který je nejblíže k průmětně, resp. pozorovateli, viz obr.7.7.1. Pixel, kterým paprsek prochází, je pak aktivován podle atributů příslušné plochy, resp. pozadí.



Obr.7.7.1

Tento postup je nutné opakovat pro všechny paprsky, tj. pro všechny pixely průmětny. V případě, že pozorovatel je v konečné vzdálenosti od průmětny, je určení příslušných průsečíků trochu složitější, neboť je nutné aplikovat principy perspektivní transformace. Je tedy zřejmé, že nejkritičtějším místem algoritmu je určení průsečíků paprsku s tělesem a případné jejich uspořádání, neboť ve scéně mohou být různé typy těles, které mohou být ohraničeny rovinnými, kvadratickými nebo parametrickými plochami. Vzhledem k tomu, že až 95% času je spotřebováno určováním, zda těleso má nebo nemá průsečík s daným paprskem, a jeho případným určením, jsou nezbytné testy, které by rychlým způsobem určily, zda daný paprsek může či nemůže protínat dané těleso. Z tohoto důvodu se zavádějí tzv. ohraničující tělesa, většinou kvádr či koule. Důvodem jejich zavedení je rychlé určení těch případů, kdy paprsek neprotíná ani tento hraniční objem, aby bylo možné se vyhnout složitějším testům nutným k určení průsečíků paprsku s vlastním tělesem.



Obr. 7.7.2

Výhodou koule jako ohraničujícího tělesa je invariance vůči operacím rotace. Použití kvádru jako ohraničujícího tělesa může vést k jeho zvětšování, resp. zmenšování. Koule je navíc přirozenějším ohraničujícím tělesem pro tělesa, která jsou jí tvarově blízká, viz obr. 7.7.2.

V případě, že ohraničující těleso je koule, test, zda paprsek protíná či neprotíná danou kouli, lze realizovat poměrně jednoduše. Jsou-li $\mathbf{x}_1$ a $\mathbf{x}_2$ body, jimiž prochází paprsek, pak paprsek je určen rovnicí

$$\mathbf{x}(t) = \mathbf{x}_1 + (\mathbf{x}_2 - \mathbf{x}_1).t = \mathbf{x}_1 + \mathbf{s}.t$$

kde **s** je směrový vektor. Rozepsáním do složek dostáváme

$$x(t) = x_1 + s_x.t$$

$$y(t) = y_1 + s_y.t$$

$$z(t) = z_1 + s_z.t$$

kde s_x, s_y, s_z jsou jednotlivé složky vektoru **s**.

Označíme-li $\mathbf{x}_0$ střed ohraničující koule, pak vzdálenost bodu $\mathbf{x}_0$ a libovolného bodu **x** ležícího na paprsku **x**(t)je určena:

$$d^2 = (\mathbf{x} - \mathbf{x}_0)^T . (\mathbf{x} - \mathbf{x}_0)$$

$$= (\mathbf{x} - \mathbf{x}_0)^2 + (y - y_0)^2 + (z - z_0)^2$$

Odpovídající hodnota parametru t pro bod **x**(t) s minimální vzdáleností od bodu $\mathbf{x}_0$ je určena výrazem (viz poznámka):

$$t = - \frac{\mathbf{s}^T . (\mathbf{x}_1 - \mathbf{x}_0)}{\mathbf{s}^T . \mathbf{s}}$$

Rozepsáním:

$$t = - \frac{s_x(x_1 - x_0) + s_y(y_1 - y_0) + s_z(z_1 - z_0)}{s_x^2 + s_y^2 + s_z^2}$$

Je-li nyní d větší než poloměr ohraničující koule se středem v bodě $\mathbf{x}_0$, pak paprsek nemůže protínat dané těleso.

Je-li použito paralelní promítání a ohraničujícím tělesem je kvádr, jehož hrany jsou rovnoběžné s osami souřadného systému, pak je test velmi jednoduchý, neboť musí platit, že:

$$x_{min} \leq x \leq x_{max} \qquad \text{a} \qquad y_{min} \leq y \leq y_{max}$$

přičemž nezáleží na hodnotě z. Je zřejmé, že v případě perspektivní projekce dochází k jistému zvýšení výpočetní složitosti.

Uvedené výpočty lze podstatně zrychlit, pokud znormujeme směrový vektor s pro každý paprsek tak, že:

$$\hat{\mathbf{s}} = \frac{\mathbf{s}}{| \mathbf{s} |}$$

pak

$$t = - \hat{\mathbf{s}}^T (\mathbf{x}_1 - \mathbf{x}_0)$$

a

$$d^2 = (\mathbf{x}(t) - \mathbf{x}_0)^T . (\mathbf{x}(t) - \mathbf{x}_0)$$

Je tedy zřejmé, že je postačující pouze procesor pro výpočet skalárního součinu.

V případě, že uvedený postup bude aplikován na kvadratické plochy, které lze vyjádřit pomocí kvadratické formy

$$\mathbf{x}^T . \mathbf{A} . \mathbf{x} = 0$$

(matice **A** reprezentující danou kvadratickou plochu je symetrická), je nutné uvedený postup zjednodušit pro rychlý výpočet průsečíku paprsku s danou plochou. Aplikováním transformace rotace (matice **R**) lze docílit toho, aby paprsek byl rovnoběžný např. s osou z. V tomto případě (při použití

paralelní projekce) je hodnota souřadnice x a y konstantní a dosazením do rovnice plochy dostáváme:

$$\mathbf{x}^T \cdot \mathbf{R}^T \cdot \mathbf{A} \cdot \mathbf{R} \cdot \mathbf{x} = 0$$

tj.

$$\mathbf{x}^T \cdot \mathbf{B} \cdot \mathbf{x} = 0$$

Rozepsáním dostáváme:

$$b_{11}x^2 + b_{22}y^2 + b_{33}z^2 + 2b_{12}xy + 2b_{23}yz + 2b_{13}xz +$$

$$2b_{14}x + 2b_{24}y + 2b_{34}z + b_{44} = 0$$

Pak lze určit souřadnice průsečíků jako:

$$z_{1,2} = \frac{-s \pm \sqrt{s^2 - 4rw}}{2r}$$

kde $r = b_{33}$

$$s = (2b_{13}x + 2b_{23}y + 2b_{34})$$

$$w = b_{11}x^2 + b_{22}y^2 + 2b_{12}xy + 2b_{14}x + 2b_{24}y + b_{44}$$

Nebo lze daný paprsek ztotožnit s osou z pomocí operace rotace a posuvu (matice T obsahuje obě transformace). Pak lze psát:

$$\mathbf{x}^T \cdot \mathbf{T}^T \cdot \mathbf{A} \cdot \mathbf{T} \cdot \mathbf{x} = 0$$

tj.

$$\mathbf{x}^T \cdot \mathbf{Q} \cdot \mathbf{x} = 0$$

Rozepsáním pak po dosazení x = 0 a y = 0 dostáváme:

$$q_{33}z^2 + 2q_{34}z + q_{44} = 0$$

Průsečíky paprsku pak získáme opět řešením kvadratické rovnice. Je zřejmé, že pokud řešení kvadratické rovnice vede ke komplexnímu řešení, paprsek danou plochu neprotíná. Vzhledem k tomu, že je nutné počítat i jiné veličiny než průsečík paprsku s danou plochou, je většinou výhodnější zvolit druhý postup.

Poznámka

Pro výpočet parametru t lze využít toho, že pro minimum musí platit nutná podmínka:

$$\frac{d}{dt}\, d^2 = 0$$

$$\frac{d}{dt}\, (\mathbf{x}(t) - \mathbf{x}_0)^T . (\mathbf{x}(t) - \mathbf{x}_0)$$

$$= \mathbf{s}^T . (\mathbf{x}(t) - \mathbf{x}_0) + (\mathbf{x}(t) - \mathbf{x}_0)^T . \mathbf{s}^T = 2\, \mathbf{s}^T . (\mathbf{x}(t) - \mathbf{x}_0)$$

$$= 2 . (\mathbf{s}^T . \mathbf{s} . t + \mathbf{s}^T . \mathbf{x}_1 - \mathbf{s}^T . \mathbf{x}_0) = 0$$

Nyní lze vyjádřit hodnotu t takto:

$$t = - \frac{\mathbf{s}^T . (\mathbf{x}_1 - \mathbf{x}_0)}{\mathbf{s}^T . \mathbf{s}}$$

Literatura

[1] Akimoto,T., Mase,K., Hashimoto,A., Suenaga,Y.: Pixel Selected Ray Tracing, in [64], 1989, pp.29-50.

[2] Acland,B.,West,N.: Real Time Animation on a Frame Store Display System, Computer Graphics (SIGGRAPH'80), Vol.14, 1980, pp.182-188.

[3] Ackland,B.,West,N.: The Edge Flag Algorithm - A Method for Raster Scan Displays, IEEE Trans. on Computers, Vol. C30, 1981, pp.41-48.

[4] ACM Computer Science Conference, Proceedings of the 1984, Philadelphia, ACM 1984.

[5] Angell,I.O.: A Practical Introduction to Computer Graphics, MacMillan Press, 1985.

[6] Alvisi,L., Casciola,G.: Two and Four Array Mask Algorithms in Practice, TR Dept. of Mathematics, Univ. of Bologna, 1988.

[7] Alvisi,L., Casciola,G.: TAM rivisitato: un metodo rapido ed astto per la rappresentazione prospettica di superfice, PIXEL No.10, 1988, pp.15-24.

[8] Baker,M.P.,Hearn,D.: Computer Graphics, Prentice Hall, International Edition, 1986.

[9] Barrett,R.C.,Jordan,B.W.: A Cell Organized Raster Display for Line Drawings, CACM, Vol.17, 1974, pp.70-77.

[10] Bartsch,H.J.: Matematické vzorce, SNTL, 1971.

[11] Berger,M.: Computer Graphics with Pascal, Benjamin Cummings Publishing Comp., Menlo Park, 1986.

[12] Bergeron,R.D.(Ed.): SIGGRAPH'82 Conference Proceedings, ACM SIGGRAPH, Vol.16, No.3, July 1982.

[14] Birren,F.: Creative Color, Van Nostrand Reinhold Co., New York, 1961.

[15] Blinn,J.F.: Models of Light Reflection for Computer Synthesized Pictures, Computer Graphics (SIGGRAPH'77), Vol.11, 1977, pp.191-198.

[16] Blinn,J.F.: Computer Display of Curved Surfaces, PhD Thesis, Univ. of Utah, 1978.

[17] Blinn,J.F.,Newell,M.E.: Clipping Using Homogeneous Coordinates, Computer Graphics (SIGGRAPH'78), Vol.12, 1978, pp.245-251.

[18] Bono,P.R.,Herman,I.(Ed.): GKS Theory and Practice, EUROGRAPHICS, 1987.

[19] Bouknight,J.: A Procedure for Generation of Three Dimensional Half-toned Computer Graphics Presentation, CACM, Vol.13, 1970, pp.527-563.

[20] Butland,J.: Surface Drawing Made Simple, CAD Journal, Vol.11, 1979, pp.19-22.

[21] Casciola,G.: Basic Concepts to Accelerate Line Algorithms, Computer & Graphics, Vol.12, 1988, pp.489-502.

[22] Castle,C.M.A.,Pitteway,M.L.V.: An Application of Euklid's Algorithm to Drawing Straight Lines, in[39],, 1985, pp.135-139.

[23] Catmull,E.E.: A Subdivision Algorithm for Computer Display of Curved Surfaces, PhD Thesis, Univ. of Utah, 1974.

[24] Catmull,E.: A Tutorial on Compensation Tables, SIGGRAPH'79, Computer Graphics, Vol.14, No.3, July 1980, pp.279-285.

[25] Cheng,F.,Yen,Y.: A Parallel Line Clipping Algorithm and Its Implementation, in [41], 1989.

[26] Clark,J.H.: The Geometry Engine: A VLSI Geometry System for Graphics, Computer Graphics (SIGGRAPH'82), Vol.16, 1982, pp.127-133.

[27] Claussen,U.: On Reducing the Phong Shading Method, in [64], 1989, pp.333-380.

[28] Computational Geometry, Proceedings of the Fifth Annual Conference, ACM, 1989.

[29] Cook,R.L.: A Reflection Model for Realistic Image Synthesis, PhD. Thesis, Cornell Univ., 1982.

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

[31] Devillers,O.: The Macro Regions: An Efficient Space Subdivision Structure for Ray Tracing in [64], 1989, pp.27-38.

[32] Drs,L.,Všetečka,J.: Objektivem počítače, SNTL, 1981.

[33] Duce,D.A.,Jancene,P.(Ed.): EUROGRAPHICS'89, Conference Proceedings, North Holland Publ. Comp., 1989.

[34] Dunlavey,M.R.: Efficient Polygon Filling Algorithms for Raster Displays, Trans. on Graphics, Vol.2., 1983, pp.264-273.

[35] Ellis,T.M.R.,Semenkov,O.I.(Ed.): Advances in CAD/CAM, North Holland Publ. Comp., IFIP, 1983.

[36] Encarnacao,J.,Schlechtendahl,E.G.: Computer Aided Design - Fundamentals and System Architectures, Springer Verlag, 1983.

[37] Enderle,G.,Kansy,K.,Pfaff,G.: Computer Graphics Programming, Springer Verlag, 1984.

[38] Enderle,G.,Grave,M.,Lillenhagen,F.(Ed.): Advances in Computer Graphics I, Springer Verlag, 1986.

[39] Earnshaw,R.A.(Ed.): Fundamental Algorithms for Computer Graphics, NATO ASI Series, Series F, Vol.17., Springer Verlag, 1985.

[40] Earnshaw,R.A.(Ed.): Theoretical Foundations of Computer Graphics and CAD, NATO ASI Series, Series F, Vol.40, Springer Verlag, 1987.

[41] Earnshaw,R.A.,Wyvill,B.(Ed.): New Advances in Computer Graphics, Proceedings of Computer Graphics International 89, Springer Verlag, 1989.

[42] Fitzgerald,W.,Gracer,F.,Wolfe,R.: GRIN: Interactive Graphics for Modeling Solids, IBM Res.& Devel., Vol.25, No.4., July 1981, pp.281-294.

[43] Floyd,R.,Steinberg,L.: An Adaptive Algorithm for Spatial Gray Scale, SID 1975, Int. Symp. Dig. Techn., 1975, pp.36-37.

[44] Foley,J.D.,van Dam,A.: Fundamentals of Interactive Computer Graphics, Addison-Wesley, 1984.

[45] Foley,J.D.: Next Generation User Interface Development Tools, in [64], 1989, pp.537-538.

[46] Franklin,W.R.,Lewis,H.R.: 3-D Graphic Display of Discrete Spatial Data by Prism Maps, Computer Graphics (ACM SIGGRAPH'78), Vol.12, No.3, August 1978.

[47] Franklin,W.R.: An Exact Hidden Sphere Algorithm That Operates in Linear Time, Computer Graphics and Image Processing, Vol.15, 1981, pp.364-379.

[48] Fuchs,H.(Ed.): SIGGRAPH'81 Conference Proceedings, ACM, SIGGRAPH, Vol.15, No.3, August 1981.

[49] Gervantz,M.,Purgathofer,W.: A Simple Method for Color Quantization: Octree Quantization, Proceedings Computer Graphics International'88, Springer Verlag, 1988, pp.219-231.

[50] Getto,P.: Fast Ray Tracing of Unevaluated Constructive Solid Geometry Models, in [41], 1989, pp.563-578.

[51] Ghazanfarpour,D.,Peroche,B.: Anti - aliasing by successive Steps with a Z-Buffer, in [64], 1989, pp.235-244.

[52] Gonzales,R.G.,Wintz,P.: Digital Image Processing, Addison Wesley, 1977.

[53] Gottlieb,M: Hidden Line Subroutines for Three Dimensional Plotting, Byte, Vol.3, No.5, 1978, pp.49-58.

[54] Gorelik,A.G.: Logical Functions as a Means of Modelling Geometrical Objects, in [35], pp.135-151.

[55] Granát,L.,Sechovský,H.: Počítačová grafika, SNTL, 1980.

[56] Graphical Kernel System for Three Dimensions (GKS-3D) - Functional Description, Norma ISO/TC97/SC21 IS 7942.

[57] Greenaway,D.S.,Warman,E.A.(Ed.): EUROGRAPHICS'82 Conference Proceedings, North Holland Publ.Comp., 1982.

[58] Greenberg,D.P.,Meyer,G.W.: Perceptual Color Spaces for Computer Graphics, Computer Graphics, Vol.14, 1980, pp.254-261.

[59] Greenberg,D.P.,Marcus,A.,Schmidt,A.,Gorter,V.: The Computer Image-Applications of Computer Graphics, Addison Wesley, 1982.

[60] Greenberg,D.P.,Meyer,G.W.: Color Education and Color Synthesis in Computer Graphics, Color Research and Application, Vol.11, John Wiley & Sons, Supplement 1986, pp.S39-44.

[61] Greenberg,D.P.: Advances in Global Illumination Algorithms, in [64], 1989, pp.401-402.

[62] ten Hagen,P.J.W.,Tomiyama,T.(Ed.): Intelligent CAD Systems I, Springer Verlag, 1987.

[63] Hamlin,G., Gear,C.: Raster Scan Hidden Surface Algorithm Techniques, Computer Graphics (SIGGRAPH'77), Vol.11, 1977, pp.206-213.

[64] Hansmann,W.,Hopgood,F.R.A.,Strasser,W.(Ed.): EUROGRAPHICS'89 , Conference Proceedings, North Holland Publ. Comp., 1989.

[65] Haralick,R.M.: Pictorial Data Analysis, NATO ASI, Series F, Vol.4., Springer Verlag, 1983.

[66] Harrington,S.: Computer Graphics - A Programming Approach, McGraw Hill, 1987.

[67] Heckbert,P.: Color Image Quantization for Frame Buffer Display, Computer Graphics, Vol.16, No.3, July 1982, pp.297-305.

[68] Hilbert,R.:Construction and Display of Three Dimensional Polygon Histograms, Computer Graphics, Vol.15, No.2, July 1981.

[69] Hopgood,F.R.A.,Duce,D.A.,Gallop,J.R.,Sutcliffe,D.C.: Introduction to the Graphical Kernel System (GKS), Academic Press, 1983.

[70] Hopgood,F.R.A.,Hubbolt,R.J.,Duce,D.A.(Ed.): Advances in Computer Graphics II, Springer Verlag, 1986.

[71] Hubbolt,R.J.(Ed.): EUROGRAPHICS'82, Tutorial Notes, EUROGRAPHICS Assos., Geneva, 1982.

[72] Hubbolt,R.J.,Arnold,A.C.,Hewitt,W.T.: Interactive Computer Graphics - Course Notes, Univ. of Manchester, Computer Graphics Unit, 1984.

[73] Inselberg,A.: The Plane with Parallel Coordinates, The Visual Computer, Vol.1, 1985, pp.69-91.

[74] Inselberg,A.,Comut,T.,Reif,M.: Convexity Algorithms in Parallel Coordinates, JACM, Vol.34, No.4, October 1987, pp.765-801.

[75] Inselberg,A.,Dimsdale,B.: Parallel Coordinates for Visualizing Multi-Dimensional Geometry, in [84], 1987, pp.25-44.

[76] Jarvis,J.F.,Judice,C.N.,Ninke,W.H.: A Survey of Techniques for the Display of Continues Tone Pictures on Bilevel Displays, Computer Graphics and Image Processing, Vol.5, 1976, pp.13-40.

[77] Jevans,D.A.J.: Optimistic Multiprocessor Ray Tracing, in [41], 1989, pp.507-522.

[78] Joseph,H.: Computer Graphics Hardware - Introduction and State of the Art, EUROGRAPHICS'87 Tutorial, EUROGRAPHICS, 1987.

[79] Kay,D.S.: Transparency, Refraction and Ray Tracing for Computer Synthesized Images, PhD Thesis, Cornell Univ., 1979.

[80] Kilgour,A.C.: Unifying Vector and Polygon Algorithm for Scan Conversion and Clipping, TR CSC/87/R7, Univ. of Glasgow, May 1987.

[81] Knuth,D.E.: Digital Halftones by Dot Diffusion, ACM Trans. on Graphics, Vol.6, No.4, October 1987, pp.245-273.

[82] Kubo,S.: Continuous Color Presentation Using a Low Cost Ink Jet Printers, in [84],1987, pp.348.

[83] Kunii,T.L.(Ed.): Frontiers in Computer Graphics, Springer Verlag, 1985.

[84] Kunii,T.L.(Ed.): Computer Graphics 1987, Proceedings of the 5th International Conference on Computer Graphics, Springer Verlag, 1987.

[85] Liang,Y.D.,Barsky,B.A.: An Analysis and Algorithms for Polygon Clipping, CACM, Vol.26, No.11, 1984, pp.868-876.

[86] 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.

[87] Lewell,J.: Computer Graphics - `A Survey of Current Techniques and Applications, Orbis Publ. Ltd., London, 1985.

[88] Mach,K.D.,Petty,J.S.: Contouring and Hidden Line Algorithms for Vector Graphic Display, Rep. AFAPL-TR-77-3, 1977.

[89] Měření barev, ČSN 01 1718.

[90] Meyer,G.W.: Wavelength Selection for Synthetic Image Generation, Computer Vision, Graphics and Image Processing, Vol.41, 1988, pp.57-79.

[91] Meyer,G.W.: Tutorial on Color Science, The Visual Computer, Vol.2, Springer Verlag, pp.278-290.

[92] Murch,G.M.: Human Factors of Color Displays, TR, Tektronix, Oregon, 1989.

[93] Murch,G.: Color Matching of Display and Printer, in [64], 1989, pp.313-314.

[94] Newmann,W.M.,Sproull,R.F.: Principles of Interactive Computer Graphics, 2nd ed., McGraw Hill, 1981.

[95] 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.

[96] O'Bara,R.M.,Abi-Ezzi,S.: An Analysis of Modeling Clip, in [64], 1989, pp.367-380.

[97] Pavlidis,T.: Graphics and Image Processing, Springer Verlag, 1982.

[98] Peitgen,H.O.: The Impact of Fractal Geometry for Computer Graphics, in [64], 1989, pp.315-316.

[99] Perdue,L.: Supercharging Your PC, McGraw Hill, 1987.

[100] Phillips,R.L.(Ed.): SIGGRAPH'78, Conference Proceedings, ACM SIGGRAPH, Vol.12, No.3, August 1978.

[101] Pins,M.,Hild,H.: Variation on Dither Algorithm, in [64], 1989, pp.381-392.

[102] Pitteway,M.L.V.: The Algebra of Algorithms - A New Toy for the Theoretician?, IUCC Bulletin, Vol.1, 1979, pp.139-144.

[103] Pitteway,L.M.V.,Watkinson,D.J.: Bresenham's Algorithm with Gray Scale, CACM, Vol.23, 1980, pp.625-626.

[104] Plastock,R.A.,Kaley,G.: Theory and Problems of Computer Graphics, McGraw Hill, New York, 1986.

[105] Pollack,B.W.(Ed.): SIGGRAPH'79 Conference Proceedings, ACM, SIGGRAPH, Vol.13., No.2., August 1979.

[106] Popsel,J.,Hornung,C.: Highlighting Shading - Lighting and Shading in a PHIGS+/PEX Environment, in [64], 1989, pp.317-332.

[107] Preparata,F.P.,Shamos,M.I.: Computational geometry - An Introduction, Springer Verlag, 1985.

[108] Rogers,D.F.,Adams,J.A.: Mathematical Elements for Computer Graphics, McGraw Hill, New York, 1976, 2. vydání 1990.

[109] Rogers,D.F.: Procedural Elements for Computer Graphics, McGraw Hill, 1985.

[110] Rogers,D.F.,Earnshaw,R.A.(Ed.): Techniques for Computer Graphics, Springer Verlag, 1987.

[111] de Ruiter,M.M.(Ed.): Advances in Computer Graphics III, Springer Verlag, 1988.

[112] Santo,H.P.: Métodos Gráficos e Geometria Computacionais, Dinalivro, Lisboa, 1985.

[113] Sheppard,J.: Human Color Perception - A Critical Study of the Experimental Foundation, Elsevier, New York, 1968.

[114] Shirai,Y.: Three Dimensional Computer Vision, Springer Verlag, 1987.

[115] Skala,V.: An Interesting Modification to the Bresenham Algorithm for Hidden-Line Problem Solution, in [39], 1985, pp.593-602.

[116] Skala,V.: An Intersecting Modification to the Bresenham Algorithm, Computer Graphics Forum, Vol.6, No.4, 1987, pp.343-347.

[117] Skala,V.: Algorithms for 2D Line Clipping, in [41], 1989, pp.121-128.

[118] Skala,V.: Algorithms for 2D Line Clipping, in [64], 1989, pp.355-367.

[119] Slavkovský,P.: Problém viditelnosti v počítačové grafice, kandidátská disertační práce, MFF UK, Bratislava, 1987.

[120] Smith,A.R.: Color Gamut Transform Pairs, SIGGRAPH'78 Conference Proceedings, ACM, SIGGRAPH, 1978, pp.12.

[121] Smith,A.R.: Tint Fill, Computer Graphics (SIGGRAPH'79), Vol.13, 1979, pp.276-283.

[122] Světelně-technické názvosloví, ČSN`36 0000.

[123] Staudhammer,J.,Livadas,P.E.: Computer Graphics - A Tutorial, The Second International Conf. on Computers and Applications, Beijing, China, 1987.

[124] Strasser,W.(Ed.): Advances in Computer Graphics Hardware I, Springer Verlag, 1987.

[125] Sutherland,I.E.,Hodgman,G.W.: Reentrant Polygon Clipping, CACM, Vol. 17., No.1, January 1974, pp.32-42.

[126] Sutherland,I.E.,Sproul,R.F.,Schumacker,R.A.: A Characterization of Ten Hidden-Surface Algorithms, Computing Surveys, Vol.6, 1974, pp.1-55.

[127] Tanner,P.(Ed.): SIGGRAPH'83, Conference Proceedings, ACM, SIGGRAPH, Vol.17, No.3, July 1983.

[128] Teunissen,W.J.M.: HIRASP - A Hierarchical Modelling System for Raster Graphics, PhD Thesis, 1988.

[129] Thalmann,N.M.,Thalmann,D.(Ed.): Computer Generated Images, Proceedings of Graphics Interface'85, Springer Verlag, 1985.

[130] Thomas,J.J.(Ed.): SIGGRAPH'80, Conference Proceedings, ACM, SIGGRAPH'80, Vol.14,, No.3, July 1980.
[131] Toifl,J.: Grafické vstupní zařízení počítače, Výběr informací, č.2, SNTL, 1973.
[132] Toifl,J.: Grafické výstupní zařízení počítače, Výběr informací, č.4, SNTL, 1973.
[133] UNIRAS - Firemní materiály firmy European Software Contractors, 1985.
[134] UNIRAS - Universal Raster Report, Firemní materiály, 1985.
[135] Vandoni,C.E.(Ed.): EUROGRAPHICS'85, Conference Proceedings, North Holland Publ. Comp., 1985.
[136] Vít a kol.: Televizní technika, SNTL, Praha, 1979.
[137] Warnock,J.E.: A Hidden Line Algorithm for Halftone Picture Representation, Univ. of Utah, Comp.Sci.Dept., Report TR 4-5, May 1968.
[138] Warnock,J.E.: A Hidden Surface Algorithm for Computer Generated Halftone Pictures, Univ. of Utah, Comp.Sci.Dept., TR 4-15, June 1969.
[139] Watkins,G.S.: A Real Time Visible Surface Program, Univ. of Utah, Comp.Sci.Dept., Report UTEC-CSC-70-101, June 1970.
[140] Watkins,S.L.: Masked Three Dimensional Plot Program with Rotation, Algorithm 483, CACM, Vol.17, 1974, pp.520-523.
[141] Watters,G.,Willis,P.: Scan Converting Extruded Lines at Ultra High Definition, Computer Graphics Forum, Vol.6, No.2, May 1987, pp.133-140.
[142] Weiler,K.,Atherton,P.: Hidden Surface Removal Using Polygon Area Sorting, Computer Graphics (SIGGRAPH'77), Vol.11, 1977, pp.214-222.
[143] Whitted,T.: An Improved Illumination Model for Shaded Display, CACM, Vol. 23,1980, pp.343-349.
[144] Williams,H.: Hidden-Line Plotting Program, Algorithm 420, CACM, Vol.15, 1972, pp.100-103.
[145] Wright,T.J.: A Two-Space Solution to the Hidden Line Problem for Plotting Functions of Two Variables, IEEE Trans. on Computers, Vol.C-22, 1973, pp.28-33.
[146] Wyvill,G.,Sharp,P.: Fast Antialiasing of Ray Traced Images, in [41], 1989, pp.579-590.

[147] Xu,H.,Peng,Q.S.,Liang,Y.D.: Accelerated Radiosity Method for Computer Environment, in [64], 1989, pp.51-62.

[148] Zhang,J.: A Fast Hidden-Line Removal Algorithm, in [41], 1898, pp.591-602.

[149] Blinn,J.F.,Carpenter,L.C.,Lane,J.M.,Whitted,T.: Scan Line Methods for Displaying Parametrically Defined Surfaces, CACM, Vol.23, pp.23-34, 1980.

[150] Phong,B.T.: Illumination for Computer Generated Images, PhD Thesis, Univ. of Utah, 1973.

[151] Carpenter,L.C.,Lane,J.M.: A Generalized Scan Line Algorithm for the Computer Display of Parametrically Defined Surfaces, Computer Graphics and Image Processing, Vol.11, pp.290-297,1979.

[152] Gouraud,H.: Computer Display of Curved Surfaces, PhD Thesis, Univ. of Utah, 1971.

[153] Kay,Douglas Scott: Transparency, Refraction and Ray Tracing for Computer Synthesized Images, MSc Thesis, Cornell Univ., 1979.

[154] Kay, Douglas Scott, Greenberg,D.: Transparency for Computer Synthesized Images, Computer Graphics (SIGGRAPH'79 Proceedings), Vol.13, pp. 158-164, 1979.

[155] Beckmann,P.,Spizzichiono,A.: Scattering of Electromagnetic Waves from Rough Surfaces, MacMillan Press, New York, 1963, pp.1-33,70-98.

[156] Cook,R.L.,Torrance,K.E.: A Reflectance Model for Computer Graphics, ACM on Graphics, Vol.1., pp.7-24, 1982.

[157] Torrance,K.E., Sparrow,E.M.: Polarization, directional distribution, and off-specular peak phenomena in light reflected from roughened surfaces, Journal of the Optical Society of America, Vol.57,7 (July 1966),916-925.

[158] Torrance,K.E., Sparrow,E.M.: Theory for off-specular reflectiom from roughened surfaces, Journal of the Optical Society of America, Vol.57,9 (Sept. 1967),1105-1114.

[159] Trowbridge,T.S., Reitz,K.P.: Average irregularity representation of roughened surface for ray reflection, Journal of the Optical Society of America, Vol.65, 5 (May 1975),531-536.

[160] Agoston,G.A.: Color Theory and Its Applicaton in Art and Design, Springer Verlag 1987

[161] Baldwin,L.: Color Consideration, BYTE September 1984, pp.227-246

[162] Cahill,B.: Drawing on the 8514/A, BYTE March 1990, pp.279-289

[163] Drs,L.: Plochy ve výpočetní technice, Matematický seminář SNTL, SNTL 1984

[164] 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

[165] Samet,H.: The Quadtree and Related Hierarchical Data Structures, Computing Surveys, Vol.16, No.2, June 1984, pp.187-260

[166] Samet,H., Webber,R.E.: Storing a Collection of Polygons Using Quadtrees, ACM Trans. on Graphics, Vol. 4, No. 3, July 1985, pp. 182-222

[167] Kessener,L.R.A., Peters,F.J.,van Lierop,M.L.P.(Ed.): Data Structures for Raster Graphics, Proceedings of a Workshop held at Steensel, Springer Verlag, 1986

[168] Baumgart,B.G.: A Polyhedron Representation for Computer Vision, Proc.Nat.Comp.Conf., AFIPS 1975, pp.589-596

[169] Kilgour,A.: Techniques for Modelling and Displaying 3D Scenes, Technical Report, Univ. of Glasgow, 1986

[170] Pratt,M.J.: Types of Modeller, Technical Report, Dept. of Mathematics, Cranfield Inst. of Technology, Cranfield U.K., September 1982

[171] Thalmann,N.M.,Thalmann,D.(Ed.): Computer Animation, Theory and Practice, Springer Verlag, 1985.

[172] Greenberg,D.P.: Ray Tracing and Radiosity, State of Art in Image Synthesis, course notes, SIGGRAPH'86, ACM, 1986

[173] Greenberg,D.P.,Cohen,M.F.,Torrance,K.E.: Radiosity: A Methods for Computing Global Illimination, The Visual Computer, Vol.2., No.5.,September 1986.

[173] Burgoon,D.A.: Global Illumination Modeling Using Radiosity, Hewlett Packard Journal, December 1989, pp.78-88.

[174] Goral,C.M.,Torrance,K.E.,Greenberg,D.P.,Battaile,B: Modelling the Interaction of Light between Diffuse Surfaces, SIGGRAPH'84, ACM, 1984.

[175] Cohen,M.F.,Greenberg,D.P.: The Hemi-Cube: Radiosity Solution for Complex Environments, SIGGRAPH'85, ACM, 1985.
[176] Graphics Databook, firemní materiály INMOS SGS-Thompson, 1990.
[177] Systems Solutions, firemní materiály IChips Europe, 1990.
[178] Jarvis,J.F., Roberts,C.S.: A New Technique for Displaying Continuous Tone Images on a Bilivel Display, IEEE Trans. Communic., Vol.24, pp.891-898, 1976.
[179] Abhyankar,S.S.,Chandrasekar,S.,Chandru,V.: Improper Intersection of Algebraic Curves, ACM Trans. on Graphics, vol.9, No.2, 1990, pp.147-159.
[180] Andreev,R.D.: Algorithm for Clipping Arbitrary Polygons, Computer Graphics Forum, Vol.7, No.3, 1988, pp.183-192.
[181] Brunet,P.,Navazo,I.: Solid Representation and Operation Using Extended Octrees, ACM Trans. on Graphics, vol.9, No.2, 1990, pp.170-197.
[182] Day,A.M.: The Implementation of an Algorithm to find the Convex Hull of a Set of Three Dimensional Points, ACM Trans. on Graphics, vol.9, No.1, 1990, pp.105-132.
[183] Dobkin,D.P.,Levy,S.L.F.,Thurston,W.P., Wilks,A.R.: Contour Tracing by Piecewise Linear Approximations, ACM Trans. on Graphics, vol.9, No.4, 1990, pp.389-423.
[184] Edelsbrunner,H.,Mucke,E.P.: Simulation of Simplicity: A Technique to Cope with Degenerate Cases in Geometric Algorithms, ACM Trans. on Graphics, vol.9, No.1, 1990, pp.66-104.
[185] Falcidieno,B.,Floriani,L.: A Hierarchical Boundary Model for Solid Object Representation, ACM Trans. on Graphics, vol.7, No.1, 1988, pp.42-60.
[186] Fournier,A.,Fussel,D.: On the Power of the Frame Buffer, ACM Trans. on Graphics, vol.7, No.2, 1988, pp.103-128.
[187] Gaude,S.,Hobson,R.,Chilka,P.,Calvert,T.: Multiprocessor Experiments for High Speed Ray Tracing ACM Trans. on Graphics, vol.7, No.3, 1988, pp.151-179.
[188] Heal,B.: Hidden Octree Removal, Computer Graphics Forum, Vol.7, No.3, 1988, pp.199-206.
[189] Herman,I.,Reviczky,J.: Some Remarks on the Modelling Clip Problem, Computer Graphics Forum, Vol.7, No.4, 1988, pp.265-272.

[190] Herman,I.: On The Projective Invariant of Conics in Computer Graphics, Computer Graphics Forum, Vol.8, No.4, 1990, pp.301-314.

[191] Hobby,J.D.: Rasterization of Nonparametric Curves, ACM Trans. on Graphics, vol.9, No.3, 1990, pp.3262-277.

[192] Kuijk,A.A.M.,Blake,E.H.: Faster Phong Shading via Angular Interpolation, Computer Graphics Forum, Vol.8, No.4, 1990, pp.315-325.

[193] Lamming,L.,Rhodes,W.L.: A Simple Method for Improved Color Printing of Monitor Images, ACM Trans. on Graphics, vol.9, No.4, 1990, pp.345-375.

[194] Levoy,M.: Efficient Ray Tracing of Volume Data,, ACM Trans. on Graphics, vol.9, No.3, 1990, pp.245-261.

[195] Nicholl,R.A., Nicholl,T.M.: Performing Geometric Transformations by Program Transformation, ACM Trans. on Graphics, vol.9, No.1, 1990, pp.28-40.

[196] Preparata,F.P.,Vitter,J.S.,Yvinec,M.: Computation of the Axial View of a Set of Isothetic Parallelopides, ACM Trans. on Graphics, vol.9, No.3, 1990, pp.278-300.

[197] Pun,T.,Blake,E.: Relationships Between Image Synthesis and Analysis: Towards Unification?, Computer Graphics Forum, Vol.9, No.2, 1990, pp.149-164.

[198] Rokne,J,G.,Wyvill,B.,Wu,X.: Fast Line Scan Conversion, ACM Trans. on Graphics, vol.9, No.4, 1990, pp.376-388.

[199] Rossignac,J.,Voelcker,H.B.: Active Zones in CSG for Accelerating Boundary Evaluation, Redundancy Elimination, Interference Detection, and Shading Algorithms, ACM Trans. on Graphics, vol.8, No.1, 1989, pp.51-87.

[200] Rushmeier,H.E.,Torrance,K.E.: Extended the Radiosity Method to Include Specularly Reflecting and Transluent Materials, ACM Trans. on Graphics, vol.9, No.1, 1990, pp.1-17.

[201] Stone,M.C.,Cowan,W.B.,Beatty,J.C.: Color Gamut Mapping and the Printing of Digital Color Images, ACM Trans. on Graphics, vol.7, No.4, 1988, pp.249-293.

[202] Thomas,D.,Netravali,A.N.,Fox,D.S.: Anti-aliased Ray Tracing with Covers, Computer Graphics Forum, Vol.8, No.4, 1990, pp.315-324.

[203] Veenstra.J.,Ahuja,N.: Line Drawing of Octree Represented Objects, ACM Trans. on Graphics, vol.7, No.1, 1988, pp.61-75.

[204] Ware,C.,Cowan,W.: The RGYB Color Geometry, ACM Trans. on Graphics, vol.9, No.2, 1990, pp.226-232.

[205] Zyda,M.J.: A decomposable Algorithm for Contour Surface Display Generation, ACM Trans. on Graphics, vol.7, No.2, 1988, pp.129-148.

[206] Ježek,F.: AutoCAD - učební text, VŠSE Plzeň, 1991.

[207] Poláček,J.,Ježek,F.,Kopincová,E.: Počítačová grafika, skripta ČVUT-FS Praha, 1991.

[208] Ranklin,J.R.: Computer Graphics Software Construction, Advances in Computer Science Series, Prentice Hall, 1989.

[209] Rushmeier,H.E.,Torrance,K.E.: The Zonal Method for Calculating Light Intensities in the Presence of a Participating Medium, Computer Graphics, Vol.21, No.4, July 1987.

[210] Počítačová grafika - Názvosloví, ČSN 36 9001 část 13.

[211] Systémy zpracování informací (GKS), ČSN 36 9180.

[212] Firemní materiály firmy INTERGRAPH, June 1991.

[213] Skala,V.: Počítačová grafika I, Skripta VŠSE (Západočeská univerzita), Plzeň, 2. vydání, 1991.

[214] Skala,V.: Počítačová grafika II, Skripta VŠSE (Západočeská univerzita), Plzeň, 2. vydání, 1991.

[215] Drzaic,P.S.: Nematic Droplet Polymer Films for High Contrast coloured Reflective Displays, Displays Technology and Applications, Butterworth-Heinemann Ltd., Vol.12, No.1, January 1991, pp.2-13.

[216] Schwarz,M.W.,Cowan,W.B.,Beatty.J.C.: An Experimental Comparison of RGB, YIQ, LAB, HSV, and Opponent Color Models, ACM Transaction on Computer Graphics, Vol.6,No.2, April 1987, pp.123-158.

[217] Colour Addendum to ISO 8613 - Working Draft, ISO TC97 SC18 WG5, X3H3/88-47.

[218] Prett,V.: Cifrovaja obrobotka izobraženij, Mir, Moskva, 1982.

[219] Serba,I.: Termodynamický přístup k výpočtu osvětlení prostorové scény v počítačové grafice, seminář MOP 91, 1991.

[220] Sochor,J.: Sledování paprsku ve 3D scéně, seminář MOP 91, 1991.

[221] Hall,R.: Illumination and Color in Computer Generated Imaginary, Springer Verlag, 1989.