Algoritmy a počítačová grafika

Poznámky k přednáškám

Pozor, musí být spuštěn na pozadí WEB prohlížeč
testováno s MS IE 8
DRAFT – 2.01

Rád bych požádal čtenáře o zaslání nalezených chyb, nekorektních či nejasných formulací atd.,
které bych rád v další verzi textu opravil.
(ideálně skala@kiv.zcu.cz předmět zprávy: ZPG /verze s uvedením stránky a řádky na stránce)

Text je doplněním poznámek a videozáznamů přednášek ZPG a APG z let 2007/8 -2012/13

Nešířit

prof.Ing.Václav Skala, CSc.
Katedra informatiky a výpočetní techniky
Fakulta aplikovaných věd
Západočeská univerzita v Plzni
skala@kiv.zcu.cz http://www.VaclavSkala.eu
Created: 2010-04-10

1

Obsah

Profil přednášejícího

(http://www.VaclavSkala.eu)

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 téma relačních databází se specializací na reprezentaci relací a optimalizaci dotazů uživatele. V současné době se odborně věnuje především algoritmům počítačové grafiky, vizualizaci dat a algoritmům obecně, včetně matematických aspektů. Je vedoucím Centra počítačové grafiky a vizualizací (http://Graphics.zcu.cz) na 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., na 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 impaktovaných 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).

Prof. Skala je profesorem na Západočeské univerzitě v Plzni. Odborně působil též na VŠB-Technické univerzitě v Ostravě v letech 2010-2013, na Ostravské univerzitě v Ostravě v letech 2010-2011.

V roce 2010 prof. Skala získal významné ocenění mezinárodní asociace pro počítačovou grafiku **EUROGRAPHICS** Association (http://www.eg.org)

„**Fellow of the Eurographics Association**“

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

Je milou povinností poděkovat všem, kteří 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. Mike 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

Tyto učební podklady jsou neveřejnými podklady pro výuku předmětu bakalářského studia

Základy počítačové grafiky (ZPG)

a předmětu magisterského studia

Algoritmy počítačové grafiky (APG)

na Fakultě aplikovaných věd Západočeské univerzity v Plzni. Pro předmět APG pak slouží především k prohloubení znalostí z již předpokládanému absolvovaní předmětu ZPG.

Doprovodné materiály byly získány z WEBu; pokud není stanoveno jinak, jsou poskytnuty s licencí Wikipedia Commons nebo jinou obdobnou. Za případné opomenutí se autorům předem omlouvám a rád zahrnu na jejich práci příslušný odkaz.
Text neprošel žádnou jazykovou kontrolou a slouží výhradně pro pracovní potřebu.

Rád bych upozornil na dodatečné zdroje k přednáškám, a to:

- Záznam přednášek předmětu ZPG z roku 2008 (CLICK)
- Skala,V: Algoritmy počítačové grafiky I, skripta 2011 (CLICK)
- Skala,V.: Algoritmy počítačové grafiky II, skripta 2011 (CLICK)
- Skala,V: Algoritmy počítačové grafiky III, skripta 2011 (CLICK)
- Skala,V.: Světlo, barvy a barevné systémy v počítačové grafice, Academia, 1993 (CLICK) http://herakles.zcu.cz/~skala/ZPG/Svetlo_barvy_barevne_systemy.pdf
- PowerPoint prezentace přednášek ZPG s příklady s OpenGL (CLICK)

http://www.iue.tuwien.ac.at/phd/wessner/diss.html

Ostatní relevantní dodatečné zdroje informací jsou pak vždy uvedeny na konci příslušné kapitoly.

Obsah přednášek ZPG

1. Úvod, typické aplikace počítačové grafiky a vizualizace dat. Základní architektura grafických systémů a grafická rozhraní OpenGL/DirectX/SVG - principy.

2. Souřadné systémy v počítačové grafice, homogenní souřadnice a jejich geometrická interpretace. Numerická reprezentace a stabilita výpočtů.

3. Základní geometrické transformace v E2 a E3, řetězení operací. Geometrické entity, princip duality.

4. Transformace Window-Viewport. Promítání, rovinné projekce, pozice kamery.

5. Datové struktury, hierarchické modely a geometrické transformace

6. Světlo a barevné modely. Modely osvětlení a metody stínování. Textury a bitové mapy.

7. Základní algoritmy řešení viditelnosti, metoda sledování paprsku a radiační metoda.

8. Interpolace, křivky a plochy v počítačové grafice.

9. Metody ořezávání v E2 a E3, množinové operace s n-úhelníky.

10. Vizualizace dat: datové struktury, geometrie a data, výšková pole a iso-čáry/plochy, zobrazování povrchů a skalárních polí (CT, MRI), zobrazování vektorových polí.

11. Animace, principy a inverzní kinematika

12. Rastrová grafika a základní algoritmy pro kreslení úseček a kružnic, algoritmy šrafování a plnění, anti-aliasing.

13. Zvaná přednáška nebo 3D displeje, haptické systémy a systémy virtuální reality. Architetura grafických systémů (GPU/CUDA/TESLA)

Obsah přednášek APG

1. Úvod, organizace předmětu. Složitost algoritmů. Metodologie porovnávání vlastností algoritmů.

2. Urychlovací techniky, předzpracování, dělení prostoru. Projektivní reprezentace a princip duality.

3. Geometrické transformace v E2 a E3, Pluckerovy souřadnice. Neplanární projekce. ,

4. Interpolace uspořádaných dat. Parametrické křivky a plochy.

5. Geometrické výpočty a algoritmy výpočtů průsečíků geometrických entit.

6. Metody a datové struktury pro reprezentaci objektů a volumetrická data.

7. Modelování složitých objektů, hierarchické struktury, CSG stromy.

8. Generace povrchu z objemových dat a objektů zadaných implicitním popisem.

9. Modely osvětlení a metody stínování.

10. Algoritmy sledování paprsku, výpočet průsečíku a metody urychlování, generování stínů.

11. Principy radiační metody a path tracing. Image based rendering.

12. Animace, kinematika a inverzní kinematika.

13. Interpolace neuspořádaných dat. Základy geometrické algebry.

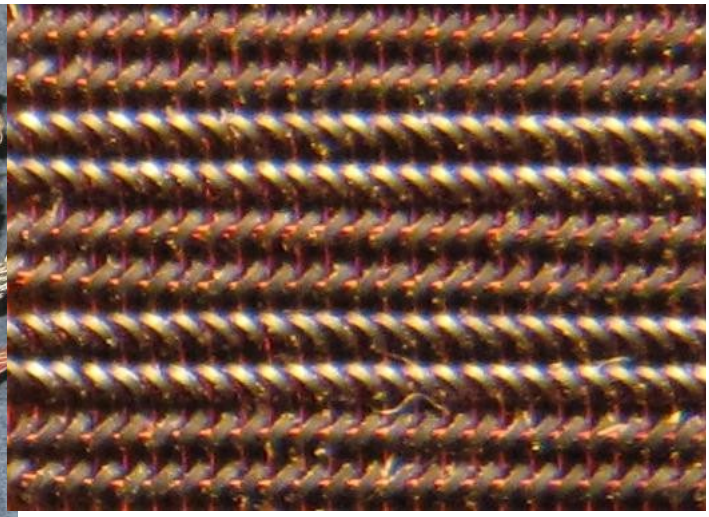
1. Úvod

1.1.Přehled vývoje počítačové grafiky

Počítačová grafika je poměrně mladá vědní oblast, která se formovala po II. světové válce, především po roce 1950 v oblastech spojených s vojenským průmyslem a vojenskými aplikacemi a byla především zpočátku chápána jen jako grafický výstup z počítače. Od jednoduchých grafických výstupů na „primitivních“ grafických výstupních zařízeních se rozvinula do nebývalé šíře nejen z teoretického hlediska, ale především v oblasti aplikací od počítačových her a tvorby filmů, technických i netechnických simulací s grafickým výstupem s možností interakce, až po aplikace haptických zařízení (zařízení se silovou zpětnou vazbou) pro tele-operace, vojenský výcvik s použitím 3D zobrazování a prostředků virtuální reality.

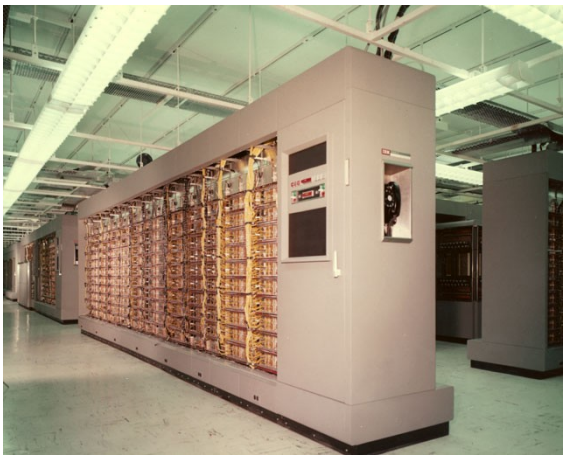
První aktivity lze datovat do roku 1954, kdy byl použit „první grafický display“ v rámci projektu SAGE (Semi-Automatic Ground Environment) jako součásti obraného protiraketového systému v době „studené války“. Je nutné zdůraznit, že použitý superpočítač SAGE měl v té době super kapacitní paměť 4x64Kslov, 60 000 elektronek, vážil 250 tun, obsluha měla 100 osob a cena je odhadována na 8-12 miliard dolarů v relacích r. 1964!

Operační paměť 451x452 bitů
Velikost 12,7x12,7x18,7 cm

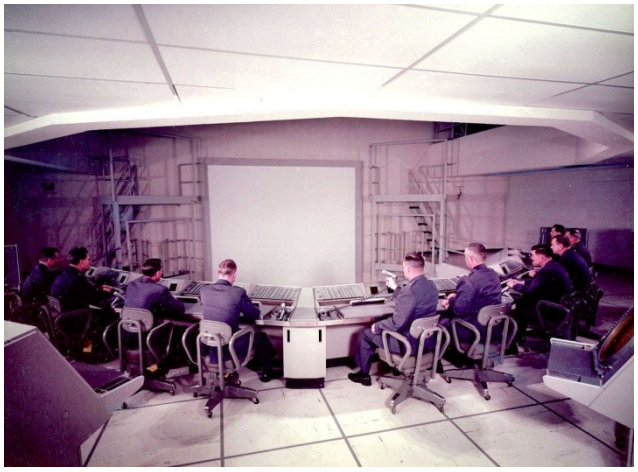


Detail paměti
Jadérka mají 0,457 mm

SAGE monitor

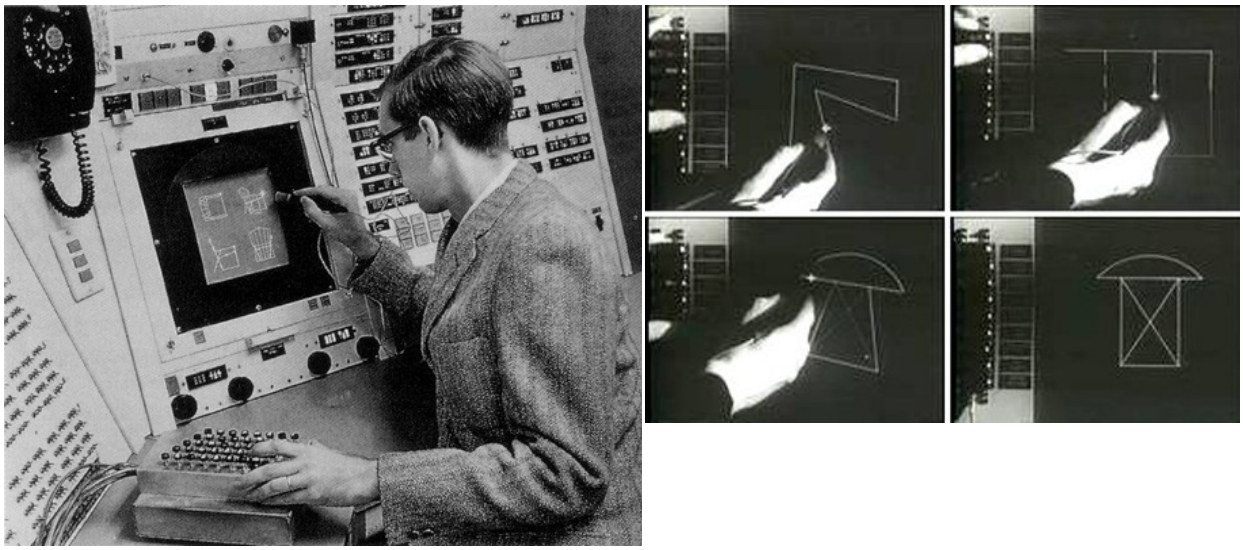


Typická konfigurace výpočetního systému



Za skutečný začátek éry počítačové grafiky lze považovat však až vznik legendárního programu SketchPad, který byl vytvořen Ivanem Sutherlandem v roce 1963 v rámci jeho PhD práce. Systém umožňoval nejen zobrazování „složitých“ objektů, ale především interakci pomocí světelného pera. To byl vlastně začátek oboru, který je dnes označován jako „Human Computer Interaction“ (HCI), tedy interakce člověk-počítač.

SketchPad sytém byl realizován na MIT (Massachusetts Institute of Technology) na počítači Lincoln TX-2, který měl „bájných“ 64Kslov o 36 bitech v době, kdy se používalo téměř výlučně dávkové zpracování a děrné štítky.



Click for video= Sutherland ScatchPad-YouTube

Práce Ivana Sutherlanda a další výzkum a vývoj pak otevřely cestu k mnoha následným navazujícím oblastem. Mezi nejdůležitější patří:

- Počítačová grafika (Computer Graphics) – zabývající se především metodami generování vizuálních objektů na základě jejich popisu geometrického apod. Hlavní aplikace byly zpočátku v oblasti vojenské, později pak i v herním a zábavním průmyslu
- CAD/CAM systémy – které umožňují nejen vlastní zobrazování geometrických objektů, ale i jejich strukturovaný návrh a navíc ve spojení se simulačními prostředky, včetně ověření jejich fyzikálních vlastností. Rozvoj CAD/CAM systémů byl dán především požadavky leteckého, lodního a raketového průmyslu.

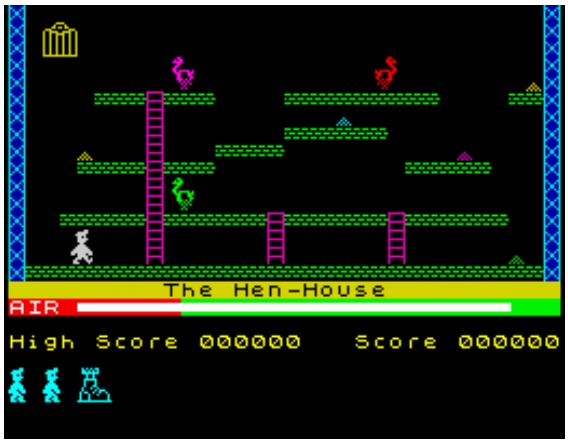
K zásadnímu rozvoji počítačové grafiky však především přispěl rozvoj personálních počítačů s legendárními počítači Sinclair ZX Spektrum se 48 KB paměti a programovacím jazykem Pascal 4T, Apple II nebo Amiga (technické popisy viz Wikipedia.org).

ZX Spectrum – 48KB paměť
1982

Amiga
1985

Apple II
1997

V té tobě také vznikaly první počítačové hry včele s legendárním Maniac Miner. Rozvoj her inicializoval vývoj levných HW prostředků pro počítačovou grafiku s jejich masovou výrobou.



http://www.youtube.com/watch?v=F7khL9Ms4ow
CLICK – MP4

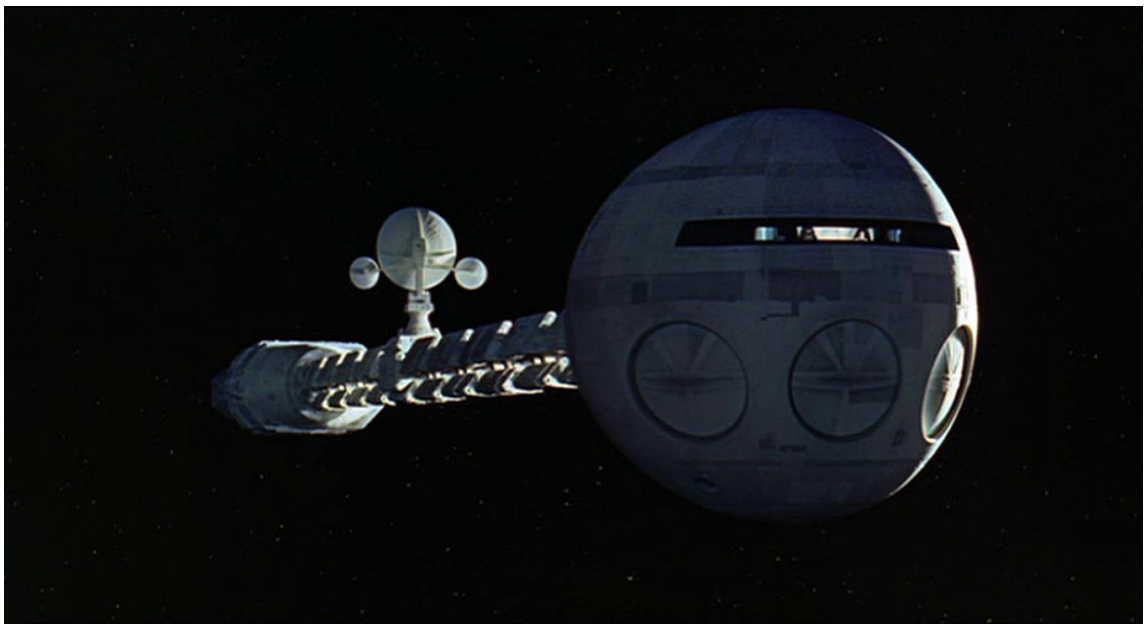
V současné době jsou principy vyvinuté v oblasti počítačové grafiky zastoupeny ve většině systémů, ať již ve formě knihoven GUI (Graphical User Interface), nebo moderních mobilních systémů.

Quake

Toy Story

Za prudkým rozvojem technických prostředků počítačové grafiky je nutné vidět především herní a filmový průmysl. Zajisté hlavními milníky bylo uvedení filmu Odysea roku 2001 Arthura C.Clarka z roku 1968, uvedení 3D filmu Toy Story v roce 1995 a počítačové hry Quake v roce 1966.

Zajímavý přehled historie lze nalézt na http://en.wikipedia.org/wiki/Computer_graphics .



http://www.youtube.com/watch?v=q3oHmVhviO8

Vesmírná Odyssea 2001 - CLICK

Vymezení oblasti počítačové grafiky není zcela jednoduché, neboť dnes již „prorostla“ do většiny aplikací výpočetních a mobilních systémů. V současné době je chápána jako interdisciplinární obor stojící na základech matematiky a computer science, nicméně prakticky zasahující dnes již do všech vědních oblastí.

Počátky počítačové grafiky na Západočeské univerzitě, vlastně na jejím předchůdci Vysoké škole strojní a elektrotechnické v Plzni, jsou datovány do roku 1975, kdy vzniká nový předmět Počítačová grafika a umělá inteligence, který je následně vyučován. Z počátku byla počítačová grafika chápána spíše jako matematická disciplina vzhledem k matematickým základům než jako disciplina patřící do Computer Science, což je dáno realizací programového vybavení, technickými prostředky.

Uveďme několik zajímavých oblastí aplikací počítačové grafiky:

- simulace v oblasti vojenství, ať už při nácviku vojenských operací, nebo např. při řešení post-traumatických onemocnění, viz např.
  Virtuální krajina (CLICK) z r.2002
- Simulace záplav v Plzni r. 2002:
  - Údolí Mže (CLICK),
  - Roudná (CLICK),
  - vliv průtoku na zaplavení (CLICK)

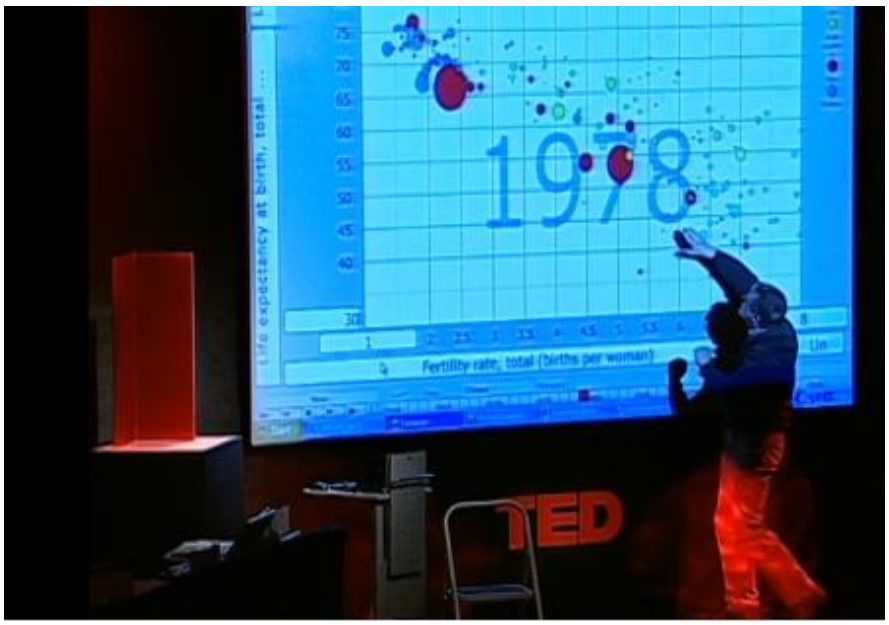
Courtesy of NVIDIA

nebo

Dr. Albert Rizzo: Post Traumatic Stress Disorder, WSCG2008 (CLICK –WMV)

- vyhodnocení rozsáhlých dynamických dat, např. v oblasti ekonomie.

  Hans Rosling v roce 2006 přednesl přednášku: „Stats that reshape your worldview“ (Statistika, která změní váš pohled na svět). Tato přednáška je pěknou ukázkou, jak důležitou roli hraje čas, resp. dynamika zobrazovaného fenoménu.



(CLICK)

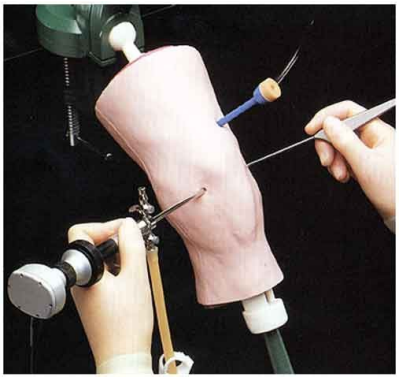
Zpracování grafické informace se zejména vyznačuje:

- velkým objemem zpracovávaných dat
- vysokými nároky prakticky na všechny parametry výpočetního systému, zejména pak na paměť
- numerickou náročností jednotlivých geometrických a numerických výpočtů

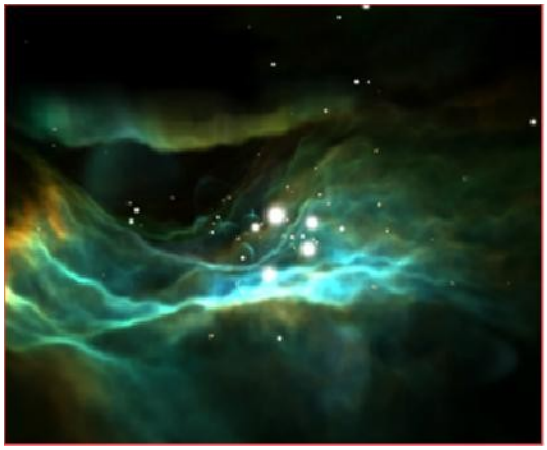
Pro návrh a realizaci metod počítačové grafiky je nutné pochopit základní principy algoritmů a metod, které však nelze jen naprogramovat, ale je nutné je optimalizovat vzhledem k velkému počtu prováděných operací. Mnohé také závisí na použitých technických prostředcích, architektuře výpočetního systému, na použitém CPU, vlastnostech sběrnice pro přenos dat a použitých specializovaných procesorů, např. GPU.

Počítačová grafika je chápána obvykle v úzkém slova smyslu, tj. tzv. „generativní“ grafika, kdy se ze zadaných dat, např. geometrického modelu, generuje vizuální výstup. V širším slova smyslu pak počítačová grafika dnes zahrnuje oblasti vizualizace dat a informací (Data Visualization & Information Visualization), zpracování obrazu (Image Processing), rozpoznávání obrazu, resp. vzorů (Pattern Recognition), počítačového vidění (Computer Vision), virtuální reality (Virtual Reality) a haptických systémů (Haptic Systems), tj. systémů se silovou zpětnou vazbou. Počítačová grafika tvoří podstatnou část nejen CAD/CAM systémů, tj. např. systémů pro návrh mechanických součástí, letadel, lodí, raket a kosmických systémů, systémů GIS (Geographical Information Systems) atd., ale i systémů výpočtové fyziky, chemie nebo biologie, kde je rozhraním pro uživatele pro zobrazení výsledků z výpočetního systému a interakci s počítačovým modelem.

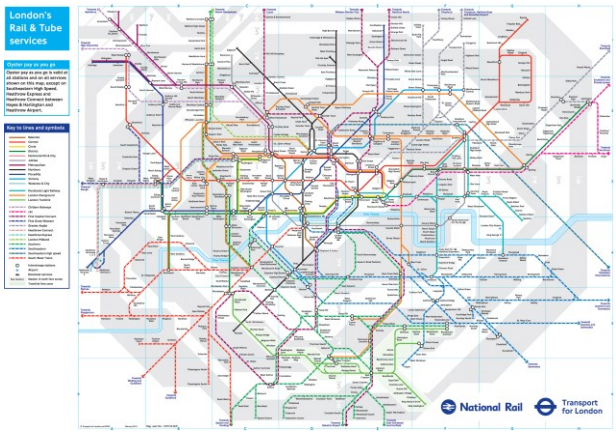
Haptické pero



Operace kolena – trénink s haptickým zařízením

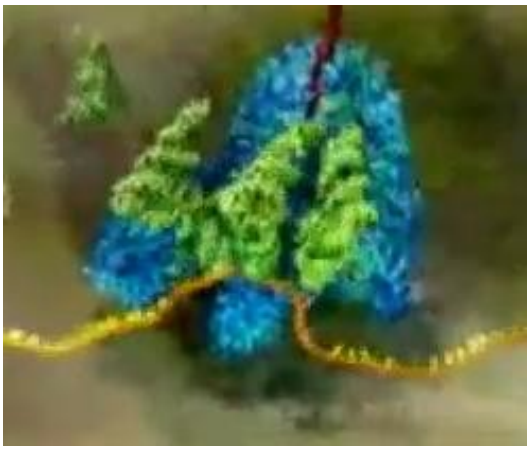


Vizualizace dat – mlhoviny (CLICK)



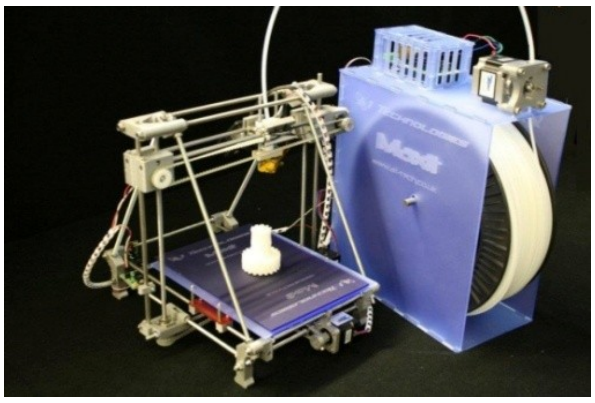
Vizualizace informací

Virtuální realita (CLICK)

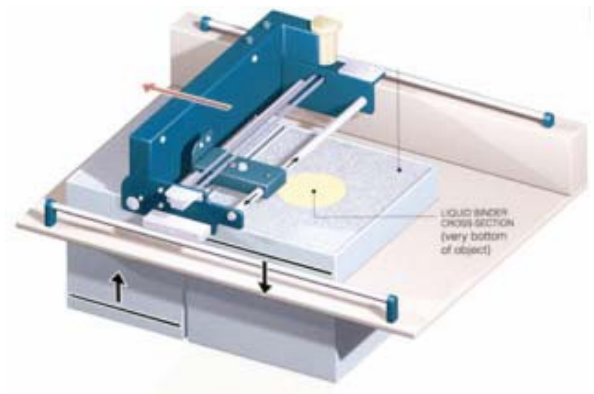


Simulace a vizualizace DNA chromosomů (CLICK)

Je možné je též „vytisknout“ generovaný objekt na 3D tiskárně (tzv. rapid prototyping), tj. vytvořit 3D fyzický model, a to i velkých rozměrů. V současné době se testuje možnost 3D tisku i pro tak náročné použití jako jsou trysky raketových motorů.



3D tiskárna založená na tavícím se plastu



3D tiskárna –lepený prášek

Lebka vytištěná na 3D tiskárně

3D tisk velkých objektů

Poměrně často se používá stereoskopie pro generaci 3D obrazů ve spojení se systémy virtuální reality. V současné době je možné nejen objekty zobrazovat na 2D průmětně, např. na stínítku obrazovky, ale též zobrazovat pomocí 3D zobrazení s použitím různých fyzikálních principů.

1.2.Počítačová grafika a ostatní oblasti

Je zřejmé, že počítačová grafika souvisí s mnoha vědními oblastmi, zejména pak s oblastí percepce člověka, tj. jak člověk vnímá vnější svět a naopak, jak jej ovlivňuje. Je proto otázkou, v jakém vztahu jsou vjemové „kanály“ člověka a která oblast se věnuje zpracování těchto signálů.

		výstup				
		popis	vizuální	zvukový	taktilní	orientace v prostoru
vstup	popis	symbolická manipulace	počítačová grafika	hlasový výstup	haptický systém	
	vizuální	rozpoznávání obrazu	zpracováni obrazu			
	zvukový	rozpoznávání zvuku		zpracování zvuku		
	taktilní	rozpoznávání hmatové informace				
	*orientace v prostoru					

* Orientace v prostoru – vnímání orientace polohy v gravitačním poli, zrychlení apod. je dalším faktorem, který nebývá zmíněn v souvislosti s lidskými senzory, nicméně je zde podmíněnost existencí gravitačního pole, principy zachování energie apod. Je vhodné upozornit na fakt, že člověk vnímá i změnu zrychlení, což je důležitý faktor např. pro pocit komfortu ve vlaku.

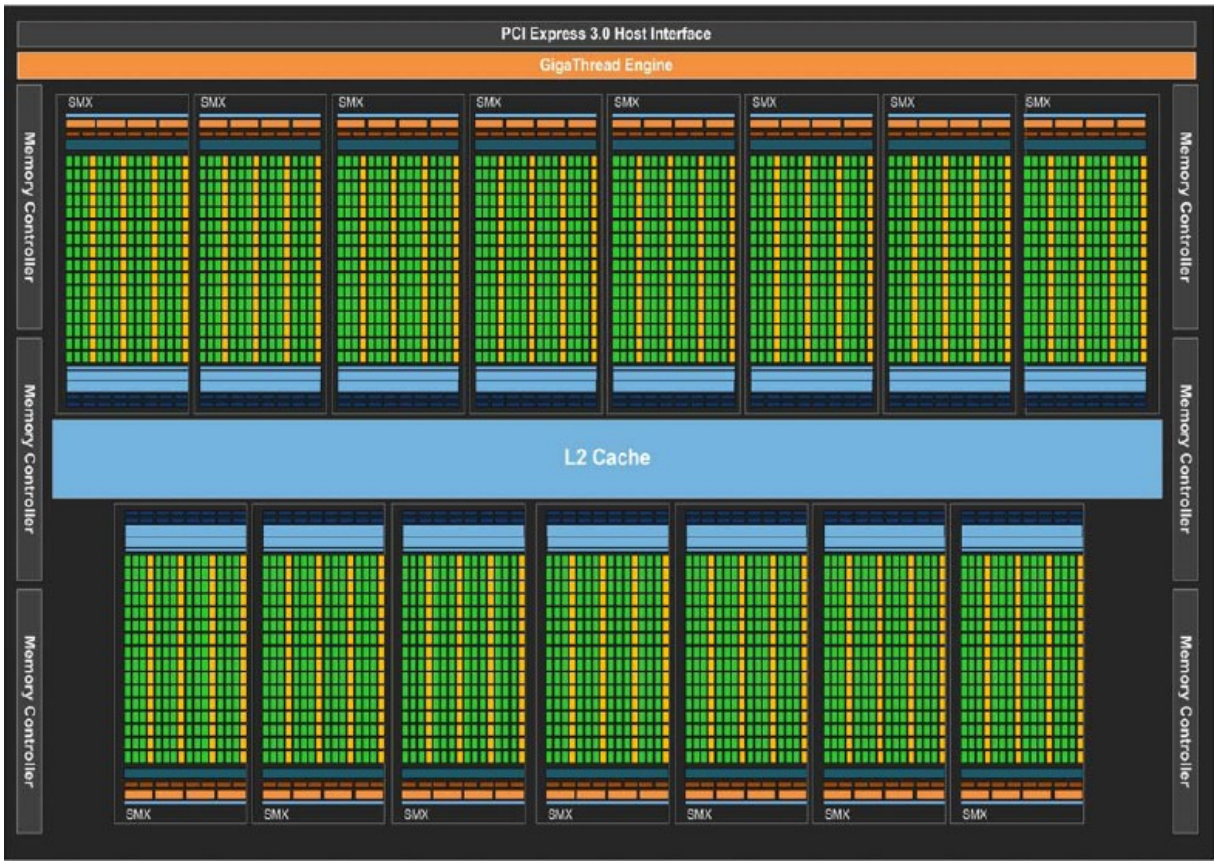
Tab.xx

1.3.Architektura grafických systémů

Architektura grafických systémů se v zásadě ustálila na dvou hlavních architekturách, a to:

- architektura pro aplikace v oblasti generovaných obrazů, tj. v oblasti počítačové grafiky založené na GPU procesorech, jejichž hlavními reprezentanty jsou NVIDIA a ATI
- architektura pro zpracování obrazu, videa a aplikace v oblasti počítačového vidění, kde hlavním reprezentantem je MATROX. Systémy jsou založeny převážně na architektuře FPGA.

V současné době je asi nejvýkonnější systém NVIDIA Kepler 110, který má 7,1 miliardy tranzistorů a poskytuje výpočetní výkon přes 1 Terra FLOP v dvojnásobné přesnosti.



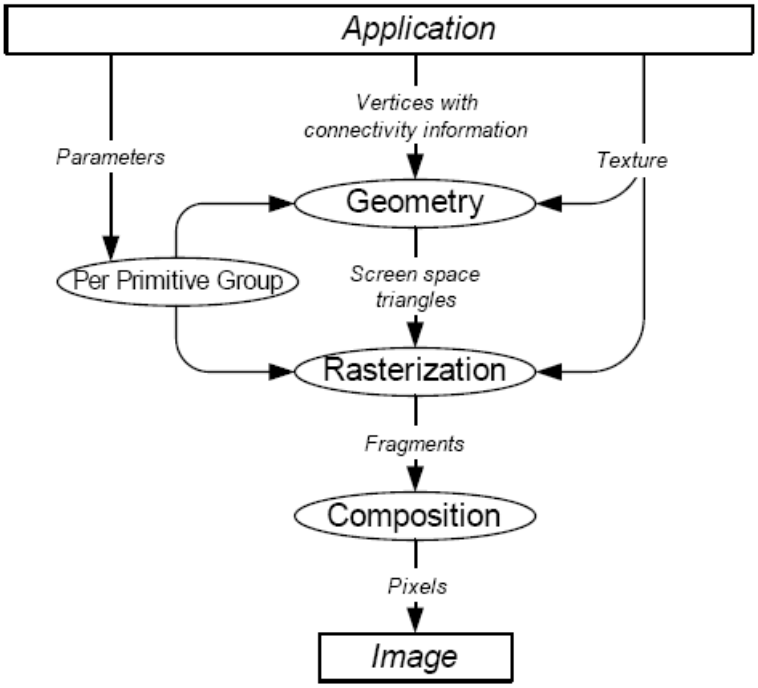
NVIDIA Kepler 110 architecture

Podrobnější informace lze nalézt na:

- http://www.nvidia.com/object/nvidia-kepler.html
- http://www.youtube.com/course?list=EC4A8BA1C3B38CFCA0 (10 hod. přednášek)
-

V současné době jsou systémy Kepler určeny především pro výpočty na GPU, tzv. GPGPU Computing (General Purpose GPU).

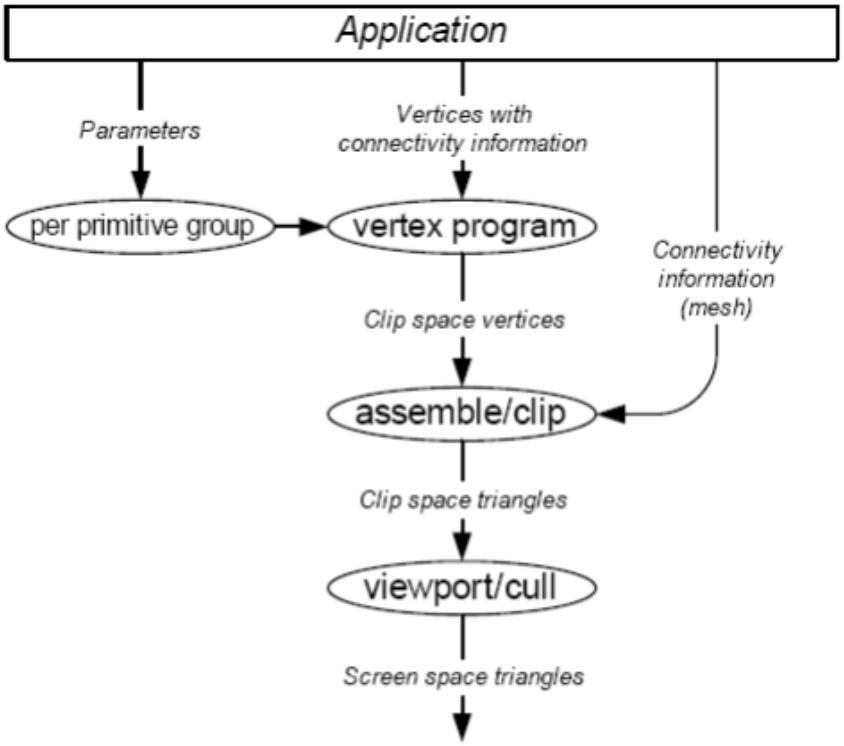
V následujícím textu se budeme především zabývat základními vlastnostmi současných běžně dostupných grafických systémů založených na GPU architektuře, jejichž základní vlastnosti jsou znázorněny na obr.xxx



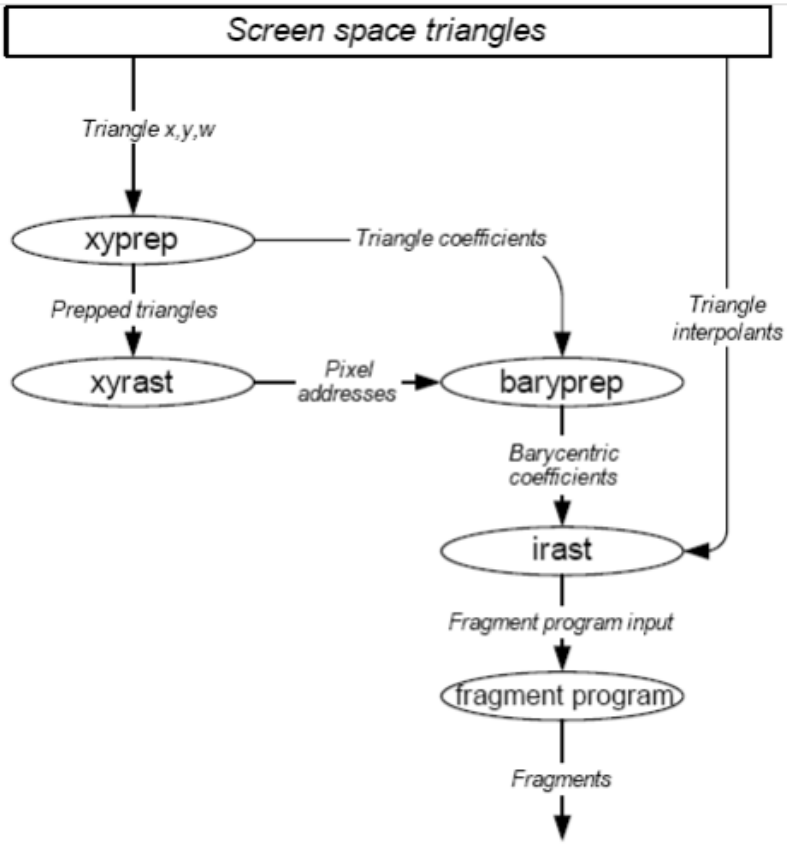
Obr.xx

Podrobnější informace lze nalézt např. v
http://graphics.stanford.edu/papers/jowens_thesis/jowens_thesis.pdf

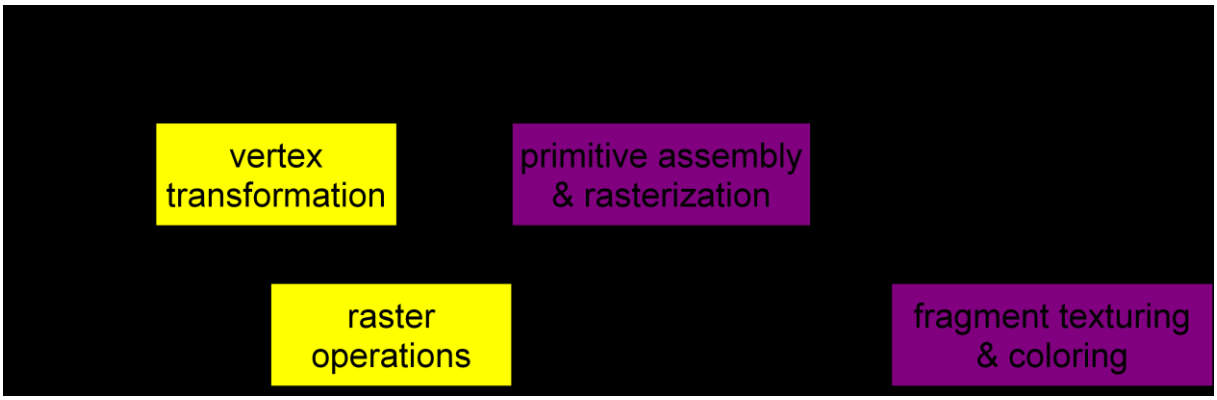
Poznámka – pod pojemem vertex je vhodné si představit některý bod základního primitiva, tj. např. koncové body úsečky nebo vrcholy trojúhelníka apod. Rasterizace je proces, kdy se trojúhelník daný svými vrcholy převede do rastrové podoby, tj. do pixelů odpovídající barvy, a výsledek se nazývá fragment. Následně pak jednotlivé fragmenty, tj. diskrétní reprezentace jednotlivých grafických primitiv, např. trojúhelníků, se sestaví do finálního obrazu.



Obr.XXXX Zpracování geometrie



Obr.xxxx Rasterizace objektů


vertex transformation
primitive assembly & rasterization
raster operations
fragment texturing & coloring

1.4.Grafické knihovny

Pro aplikace počítačové grafiky se používají programovací nástroje, které:

- produkují přímo spustitelný nativní kód, např. C, C++, Pascal/Delphi apod.
- produkují interpretovanou meziformu (např. IL - Intermediate Language), a to např. C# nebo Java
- jsou přirozeně interpretační, jako je např. Python

Jednou z nejrozšířenějších grafických knihoven je OpenGL.

1.4.1. OpenGL

Knihovna OpenGL je dostupná jak na platformě MS Windows, tak i na UNIX.
Velmi dobrý tutoriál a příklady jsou k dispozici na:

- http://www.opengl.org/ ,resp. http://www.opengl.org/archives/resources/code/samples/glut_examples/examples/examples.html
- NEHE - http://nehe.ceske-hry.cz/tut_obsah.php
- Tutorial prezentovaný na konferenci SIGGRAPG 2013 https://www.khronos.org/developers/library/2013-siggraph-opengl-bof

Knihovna OpenGL je použitelná jak pro C, C++, C#, Java, tak i pro další.

Pro realizaci aplikací v prostředí Python, pak lze doporučit použití PyOpenGL,
viz http://pyopengl.sourceforge.net/

1.4.2. DirectX

Knihovna DirectX je knihovna primárně zaměřená na platformu MS Windows.
Tutoriály jsou k dispozici na

- http://www.directxtutorial.com/
- http://msdn.microsoft.com/en-us/library/windows/desktop/bb153264(v=vs.85).aspx

V češtině jsou pak asi dobrým zdrojem:

- http://directx.kvalitne.cz/index.php?id=6
- http://www.monade.cz/item.php?item=11

Porovnání (poněkud staršího data) obou knihoven je k dispozici na adrese:

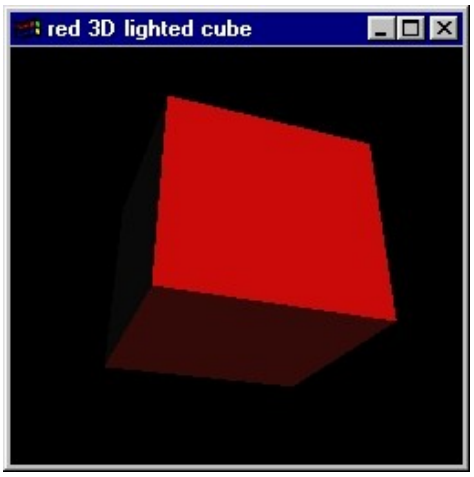
- http://woq.nipax.cz/cl_gldx.php

Další zdroje:

- Tichava,J.: OpenGL v Javě, BC.kvalifikační práce, ZČU, Plzeň 2007 http://jogl.tichava.cz/files/bp.pdf

DirectX pipeline
Podrobné informace viz
http://msdn.microsoft.com/en-us/library/windows/hardware/ff569022(v=vs.85).aspx

1.4.3. Příklad s OpenGL



Pro názornost uveďme jednoduchý příklad nakreslení krychle s použitím knihovny OpenGL.

```
/* Copyright (c) Mark J. Kilgard, 1997. */
/* This program is freely distributable without licensing
fees  and is provided without guarantee or warrantee
expressed or    implied. This program is -not- in the public
domain. */
/* This program was requested by Patrick Earl; hopefully
someone else will write the equivalent Direct3D
immediate mode program. */

#include <GL/glut.h>

GLfloat light_diffuse[] = {1.0, 0.0, 0.0, 1.0}; /* Red diffuse light. */
GLfloat light_position[] = {1.0, 1.0, 1.0, 0.0};  /* Infinite light location. */
GLfloat n[6][3] = {  /* Normals for the 6 faces of a cube. */
  {-1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {1.0, 0.0, 0.0},
  {0.0, -1.0, 0.0}, {0.0, 0.0, 1.0}, {0.0, 0.0, -1.0} };
GLint faces[6][4] = {  /* Vertex indices for the 6 faces of a cube. */
  {0, 1, 2, 3}, {3, 2, 6, 7}, {7, 6, 5, 4},
  {4, 5, 1, 0}, {5, 6, 2, 1}, {7, 4, 0, 3} };
GLfloat v[8][3];  /* Will be filled in with X,Y,Z vertexes. */

void drawBox(void)
{ int i;
  for (i = 0; i < 6; i++) {
    glBegin(GL_QUADS);
    glNormal3fv(&n[i][0]);
    glVertex3fv(&v[faces[i][0]][0]);
    glVertex3fv(&v[faces[i][1]][0]);
    glVertex3fv(&v[faces[i][2]][0]);
    glVertex3fv(&v[faces[i][3]][0]);
    glEnd();
  }
}

void display(void)
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  drawBox();
  glutSwapBuffers();
}

```

```
1   void init(void)
2   {
3     /* Setup cube vertex data. */
4     v[0][0] = v[1][0] = v[2][0] = v[3][0] = -1;
5     v[4][0] = v[5][0] = v[6][0] = v[7][0] = 1;
6     v[0][1] = v[1][1] = v[4][1] = v[5][1] = -1;
7     v[2][1] = v[3][1] = v[6][1] = v[7][1] = 1;
8     v[0][2] = v[3][2] = v[4][2] = v[7][2] = 1;
9     v[1][2] = v[2][2] = v[5][2] = v[6][2] = -1;
10
11    /* Enable a single OpenGL light. */
12    glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
13    glLightfv(GL_LIGHT0, GL_POSITION, light_position);
14    glEnable(GL_LIGHT0);
15    glEnable(GL_LIGHTING);
16
17    /* Use depth buffering for hidden surface elimination. */
18    glEnable(GL_DEPTH_TEST);
19
20    /* Setup the view of the cube. */
21    glMatrixMode(GL_PROJECTION);
22    gluPerspective( /* field of view in degree */ 40.0,
23      /* aspect ratio */ 1.0,
24      /* Z near */ 1.0, /* Z far */ 10.0);
25    glMatrixMode(GL_MODELVIEW);
26    gluLookAt(0.0, 0.0, 5.0,  /* eye is at (0,0,5) */
27      0.0, 0.0, 0.0,      /* center is at (0,0,0) */
28      0.0, 1.0, 0.);      /* up is in positive Y direction */
29
30    /* Adjust cube position to be asthetic angle. */
31    glTranslatef(0.0, 0.0, -1.0);
32    glRotatef(60, 1.0, 0.0, 0.0);
33    glRotatef(-20, 0.0, 0.0, 1.0);
34  }
35
36  int main(int argc, char **argv)
37  {
38    glutInit(&argc, argv);
39    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
40    glutCreateWindow("red 3D lighted cube");
41    glutDisplayFunc(display);
42    init();
43    glutMainLoop();
44    return 0;           /* ANSI C requires main to return int. */
45  }
```

2. Matematický aparát počítačové grafiky

Počítačová grafika v užším slova smyslu je „generativní“, tj. na základě algoritmické a matematické specifikace se generuje příslušný grafický výstup, ne nezbytně jen 2D obraz. Tento proces se často označuje výrazem „rendrování“ a příslušný modul je označován jako „Renderer“.

V následujícím textu je předpokládána základní znalost lineární algebry, zejména pak základní operace s vektory a maticemi. Je nutné zdůraznit, že předpokládané znalosti matematiky jsou na středoškolské úrovni se znalostí vektorové a maticové notace.

V textu budeme používat následující notace pro hodnoty, resp. proměnné:

- a – skalární hodnota
- $\boldsymbol{a} = [a_1, \dots, a_n]^T$ – sloupcový vektor (tučné malé písmenko)
- $\boldsymbol{A}$ – matice (tučné velké písmeno)

a operace:

- skalární součin (inner product) $\boldsymbol{a} \cdot \boldsymbol{b} = \boldsymbol{a}^T\boldsymbol{b}$ – výsledkem je skalární hodnota
- „vektorový“ součin (cross product) $\boldsymbol{a} \times \boldsymbol{b}$ - výsledek se označuje většinou vektorem, ale přesně vzato je to orientovaná plocha a v E^3 je to bi-vektor. Obecně pro n-dimenzionální bychom měli používat notaci $\boldsymbol{a} \wedge \boldsymbol{b}$,
  viz Geometrická algebra – http://Geometry.algebra.zcu.cz
- transpozice vektoru, resp. matice $\boldsymbol{a}^T$, resp. $\boldsymbol{A}^T$
- násobení matic $\boldsymbol{C} = \boldsymbol{AB}$
- inverze matic $\boldsymbol{C} = \boldsymbol{A}^{-1}$
- řešení soustav lineárních rovnic $\boldsymbol{Ax} = \boldsymbol{b}$, resp. $\boldsymbol{Ax} = \boldsymbol{0}$

Upozornění

Operace s maticemi nejsou obecně komutativní, tj.

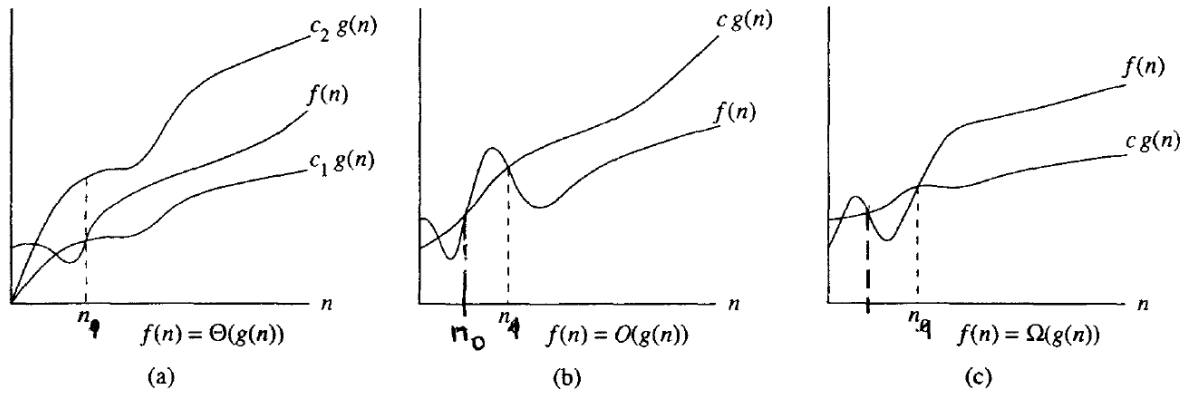
- pro součin matic $\boldsymbol{AB} \neq \boldsymbol{BA}$
- pro transpozici $(\boldsymbol{AB})^T = \boldsymbol{B}^T\boldsymbol{A}^T$
- pro inverzi $(\boldsymbol{AB})^{-1} = \boldsymbol{B}^{-1}\boldsymbol{A}^{-1}$

2.1. Algoritmy a složitost algoritmů

Algoritmus neformálně řečeno je postup, který řeší daný problém pro množinu vstupních dat s předpokládanými vlastnostmi a vede ke správnému výsledku v akceptovatelném čase. Je celá řada algoritmů, která řeší danou úlohu, viz např. algoritmy řazení. Nicméně každý algoritmus má jinou časovou a paměťovou náročnost, tzv. složitost. Proto je nutné algoritmy posuzovat z hlediska časové a paměťové složitosti.

Při posuzování algoritmů je nutné posuzovat:

- složitost předzpracování dat, pokud algoritmus používá předzpracování (pre-processing)
- složitost vlastního zpracování (run-time)
- složitost paměťovou



Obr.XXX: Klasifikace složitostí

Pro posouzení asymptotické složitosti se obvykle používají notace Θ, O, Ω (někdy se používá o) a jejich význam je znázorněn na obr.XXX, přičemž $c, c_1, c_2 > 0$.

O algoritmu $f(n)$ řekneme, že je asymptotické složitosti $O\big(g(n)\big)$, jestliže existuje konstanta c a hodnota n_1 taková, že pro $\forall n \geq n_1$ platí, že $f(n) < c\, g(n)$.

Je praxi však nutné rozlišovat:

- asymptotickou složitost, která je složitostí algoritmu pro $n \to \infty$.
- složitost algoritmu (časovou, paměťovou) pro interval počtu primitiv zpracovávaných dat, obvykle $n \in \langle n_0, n_1 \rangle$

Je nutné nahlédnout, že v dnešních aplikacích je kritickým faktorem více rychlost přenosu dat z paměti do procesoru a využití cache paměťové koherence, než vlastní rychlost procesoru. Také se pro velké hodnoty n začne projevovat faktor stránkování paměti, resp. vliv virtuální paměti apod.

Na obr.XXX.b je situace, kdy algoritmus se složitostí $O\big(f(n)\big)$ se asymptoticky chová lépe. Pro reálnou aplikaci, kdy počet zpracovávaných primitiv $n \in \langle n_0, n_1 \rangle$, je však lépe využít algoritmu se složitostí $O\big(g(n)\big)$.

K experimentálnímu ověření složitosti algoritmu se většinou používá metoda nejmenších čtverců, viz kap.9.5 (Aproximace - nejmenší čtverce).

Při zpracování většího objemu dat se určitě každý bude muset zamyslet nad tím, jak daný algoritmus zrychlit v daném kontextu, tj. s ohledem na předchozí a následný způsob zpracování a již použitých datových struktur, aby nedocházelo ke zbytečnému přepisování, resp. kopírování dat apod. K urychlení algoritmu jsou k dispozici v zásadě následující možnosti:

- nalezení jiného algoritmu, který má výhodnější časovou nebo paměťovou složitost
- předzpracování (pre-processing), který se vyplácí, pokud máme velké množství zpracovávaných dat
- paralelizace algoritmu – zde je nutno zdůraznit, že např. pro případ, kdy je možno zcela paralelizovat 97% kódu, získáme při nekonečném počtu procesorů urychlení cca 35× (viz Amdalův zákon viz http://en.wikipedia.org/wiki/Amdahl's_law). Jednotlivé techniky paralelizace kódu neuvádíme, viz např. předmět Paralelní programování.

Ukažme si výše uvedené přístupy na velmi jednoduchém případě, a to na testu, zda bod je uvnitř konvexního n-úhelníka (Point-in-Convex-Polygon) a n je počet vrcholů n-úhelníka. Běžně známé algoritmy jsou:

- algoritmus se složitostí $O(n)$ - konvexní n-úhelník je dán jako průnik polorovin a testuje se poloha bodu vůči každé polorovině. Tento algoritmus je evidentně složitosti $O(n)$. Sice využívá vlastnost konvexity n-úhelníka, avšak nevyužívá vlastnosti uspořádání indexů vrcholů. Úloha je vlastně ekvivalentní úloze, zda bod leží uvnitř konvexní obálky vrcholů n-úhelníka.
- algoritmus se složitostí $O(lg\, n)$ - algoritmus využívá vlastnosti *uspořádanosti* vrcholů ve směru nebo proti směru hodinových ručiček. Algoritmus je založen na půlení intervalu indexů, podobně jako metoda půlení intervalu pro řešení nelineární rovnice. Takový algoritmus se považuje za optimální.
- algoritmus s předzpracováním a run-time se složitostí $O(1)$ - algoritmus je založen na přezpracování, jehož složitost závisí na geometrických vlastnostech, tj. na pozici vrcholů konvexního n-úhelníka. Předzpracování je složitosti $O(m\, n\, lg\, n)$, kde m je faktor daný geometrickým rozložením vrcholů.
- paralelní algoritmus – brutální algoritmus může být založen na přímočaré paralelizaci algoritmu se složitostí $O(n)$ a zdánlivě dostaneme algoritmus se složitostí $O(1)$, což však není pravda, neboť výsledky získané z jednotlivých procesů musíme vyhodnotit a zřejmě tak dostáváme složitost $O(lg\, n)$.

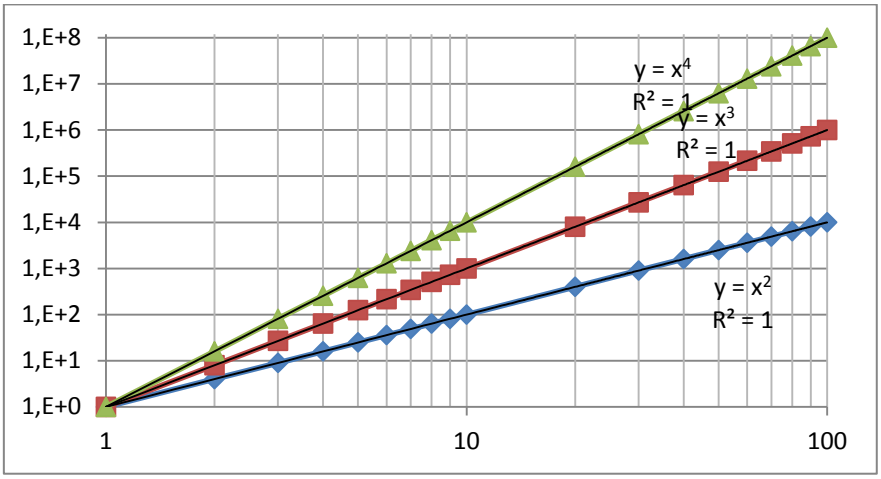
Výše uvedené algoritmy, kromě paralelního řešení, a jejich porovnání viz:

Skala,V.: Trading Time for Space: an O(1) Average time Algorithm for Point-in-Polygon Location Problem. Theoretical Fiction or Practical Usage? Machine Graphics and Vision, Vol.5., No.3., pp. 483-494, , ISSN 1230-0535, 1996. (CLICK off-line)

Experimentální vyhodnocení výpočetní náročnosti algoritmů

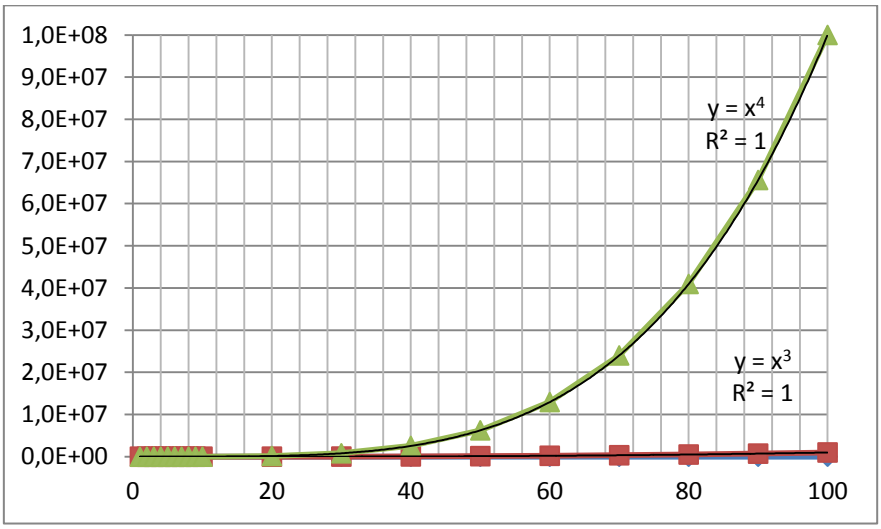
Experimentální porovnání algoritmů je nezbytnou nutností pro posouzení vlastností algoritmů. Vzhledem k tomu, že časová a paměťová náročnost se porovnává pro reprezentativní data, je nutné experimenty provést pro různý počet zpracovávaných dat. Označíme-li n počet zpracovávaných dat, pak budeme posuzovat paměťovou $mem(n)$ [Byte] a časovou $t(n)$[s] náročnost pro velký rozsah dat, např. $n \in \langle 10^6, 10^{12} \rangle$. To znamená, že nelze použít lineární měřítko na ose pro hodnoty n, ale

měřítko logaritmické. Pro posouzení složitosti je pak vhodné také logaritmické měřítko pro svislou osu, tj. pro hodnoty $mem(n)$ a $t(n)$.



Obr.xxxx: Graf závislosti výpočetní složitosti s logaritmickým měřítkem

Rozhodně není vhodné používat lineární měřítko na osách, neboť pro velký rozsah dat toto vede ke grafům, které nelze v zásadě spolehlivě interpretovat, viz obr.qqq.



Obr.qqq: Graf závislosti výpočetní složitosti s lineárním měřítkem

Připomeňme, že v případě logaritmického měřítka, větší mocnina složitosti se projeví větším sklonem regresní křivky grafu.

Takže nyní umíme vyhodnotit chování algoritmu jako takového, a to jak z hlediska paměťové náročnosti, tak i náročnosti časové. Otázkou je jak porovnávat algoritmy vzájemně.

Experimentální porovnávání algoritmů

Uvažme dva algoritmy, které chceme porovnat. Nový algoritmus má časovou náročnost $t_{new}(n)$ a referenční algoritmus má časovou náročnost $t_{ref}(n)$. Pro posouzení algoritmů budeme používat poměr:

$$v(n) = \frac{t_{ref}(n)}{t_{new}(n)}$$

To znamená, z grafu funkce $v(n)$ jsme schopni posoudit chování nového algoritmu vůči referenčnímu algoritmu. V mnoha případech však nemáme k dispozici implementaci referenčního algoritmu, zejména pak pokud chování referenčního algoritmu je pouze známo z odborné publikace. Otázkou je jak postupovat v takovém případě.

Obecně se při porovnávání algoritmů postupuje tak, že za referenční algoritmus se vezme algoritmus, jehož chování je stabilní, tj. pro dané n jeho paměťová a časová náročnost nezávisí na vlastních datech, např. na geometrickém rozložení bodů apod. Toto většinou splňují algoritmy, které řeší daný problém „brutální silou“ (Brute Force). U dříve zmíněného algoritmu testu, zda bod je uvnitř konvexního n-úhelníka, je to algoritmus se složitostí $O(N)$. Tím získáme chování nového algoritmu vůči algoritmu BF řešící daný problém. Algoritmus BF je nyní vlastně referenčním algoritmem. Pokud je obdobně i ohodnocena implementace se kterou nový algoritmus porovnáváme, pak:

$$v(n) = \frac{v(n)}{{}^{1}v(n)} = \frac{{t_{BT}(n)}/{t_{new}(n)}}{{{}^{1}t_{BT}(n)}/{{}^{1}t_{new}(n)}} = q\,\frac{{}^{1}t_{new}(n)}{t_{new}(n)}$$

kde ${}^{1}v(n)$ je chování „konkurenčního“ algoritmu vůči kterému nový algoritmus porovnáváme, $q = {t_{BT}(n)}/{{}^{1}t_{BT}(n)}$ je faktor, který určuje vliv technických parametrů použitých výpočetních systémů. Pokud je implementace algoritmu na stejných systémech, pak ideálně $q = 1$. Nicméně je otázkou, jak postupovat, pokud použité systémy jsou rozdílné, tj. $q \neq 1$. V tomto případě lze hodnotu q odhadnout na základě porovnání výpočetní výkonnosti z odpovídajících benchmarků.

Při realizaci algoritmů je nanejvýš vhodné:

- vyhnout se dvojímu, resp. trojímu indexování, i když popis algoritmu tyto indexace používá, např. $T[i,j]$, resp. $T[i,j,k]$. Je nutné si uvědomit, že se indexy přepočítávají na adresu v lineární paměti, např. $addr = (i * n + j) * m + k$
- nepožívat extenzivně objekty – určitě bude neúnosné říci, že bod (x, y) je objekt generovat n objektů pro $n \in \langle 10^6, 10^{12} \rangle$, resp. implementovat matici hodnot jako matici objektů, kde každý objekt má hodnotu atd.

Poznámka

Při testech se doporučuje volit hodnoty n podle mocninné řady R5, resp. R10 a pro několik dekád, např. $n \in \langle 10^6, 10^{12} \rangle$. Výhodou je, že na ose pro n pak dostáváme rovnoměrné rozložení hodnot.

R10	1	1.25	1.6	2	2.5	3.15	4	5	6.3	8
R5	1		1.6		2.5		4		6.3	

(http://en.wikipedia.org/wiki/Preferred_number)

Při experimentálním vyhodnocování složitosti je vhodné naměřená data proložit regresní křivkou s popisem, např. s použitím metody nejmenších čtverců, viz např. kap.9.5 (Aproximace - nejmenší čtverce).

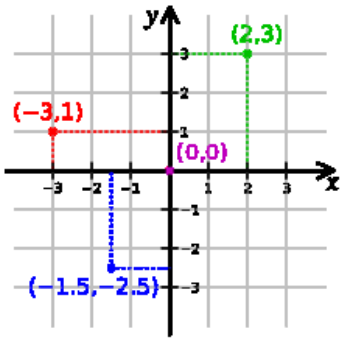
Je tedy zřejmé, že „poctivé“ porovnání algoritmů není až tak jednoduchou záležitostí, jak by se na první pohled mohlo zdát.

2.2.Souřadné systémy v počítačové grafice

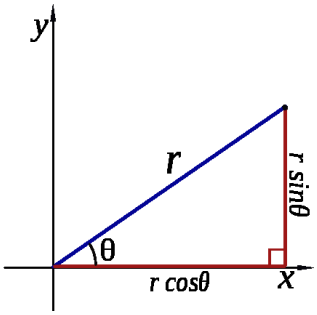
V následujícím textu uvedeme nejběžnější souřadné systémy používané v praxi, a to jak v E^2, tak i v E^3. Je nutné zdůraznit, že souřadný systém je primárně dán aplikační oblastí. Pro aplikace v oblasti zpracování dat z pozemních radarů rotačního typu půjde zřejmě o válcové souřadnice, např. pro řízení letového provozu apod.

2.2.1. Souřadné systémy v E^2

Souřadné systémy v E^2, tedy v rovině, jsou většinou charakterizovány proměnnými x, y v Eukleidovském prostoru, v parametrickém prostoru se pak většinou používá t, resp. u, v



Kartézský souřadný systém

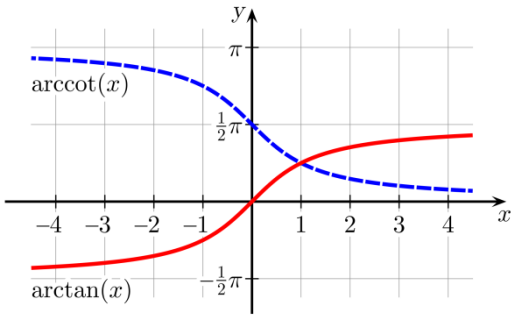


Polární souřadný systém

Vzájemný převod je dán rovnicemi:

$$x = r\cos\,\theta$$
$$y = r\sin\theta$$

$$r = \sqrt{x^2 + y^2}$$
$$\theta = arctg(^y/_x)$$

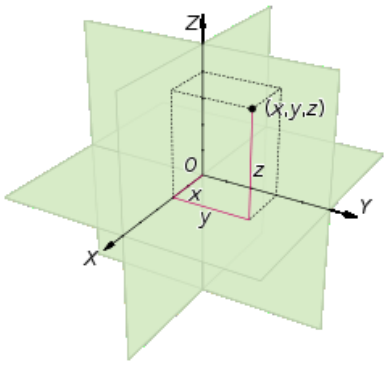


Z uvedených vztahů vidíme, že převod není úplně jednoduchý, neboť funkce $arctg(x)$ je cyklometrická a výsledkem funkce není úhel $\langle 0,2\pi\rangle$. Navíc také operace $^y/_x$ vede k numerické nestabilitě, pokud $|x| \to 0$.

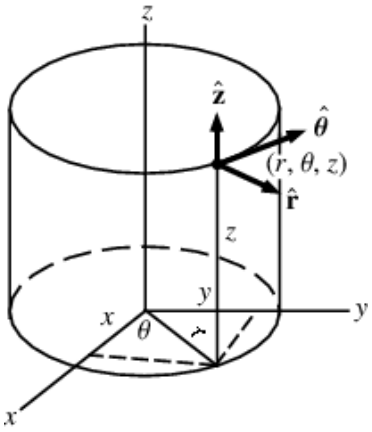
Je tedy zřejmé, že i takto jednoduchým vztahům je nutné věnovat náležitou pozornost také z hlediska numerické stability. Podrobněji viz kap. 2.5 (Numerická reprezentace a stabilita výpočtů).

2.2.2. Souřadné systémy v E^3

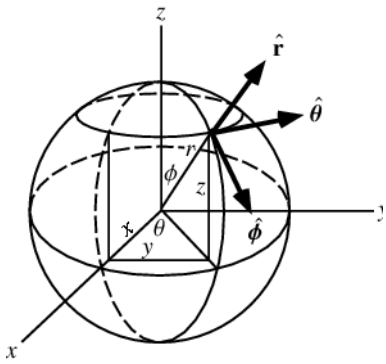
Nejčastěji používané souřadné systémy v E^3 jsou většinou:



Kartézský souřadný systém



Válcový souřadný systém



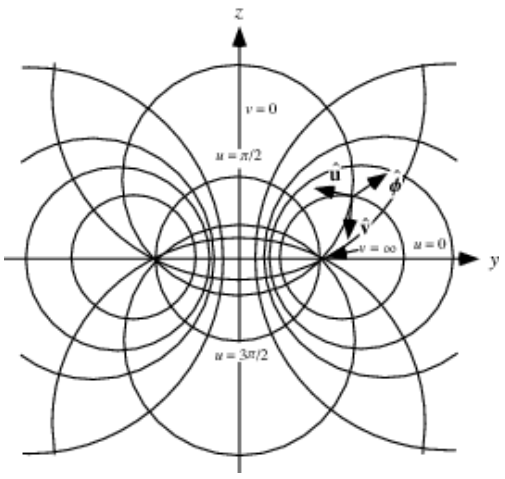
Sférický souřadný systém

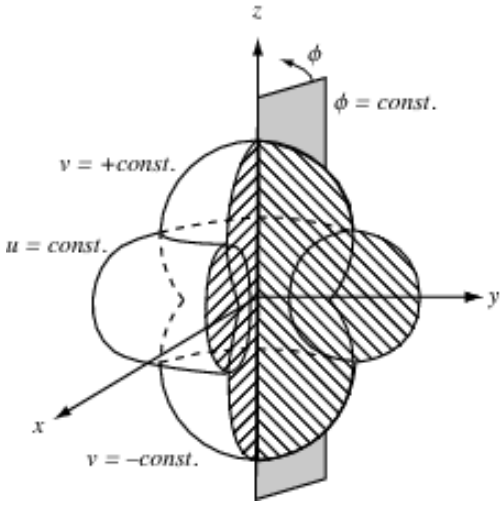
Vzájemný převod do/z je určen rovnicemi:

$x = r\cos\ \theta$	$x = r\cos\theta\sin\phi$
$y = r\sin\theta$	$y = r\ \sin\theta\sin\phi$
$z = z$	$z = r\cos\phi$
$r = \sqrt{x^2 + y^2}$	$r = \sqrt{x^2 + y^2 + z^2}$
$\theta = arctg(^y/_x)$	$\theta = arctg(^y/_x)$
$z = z$	$\phi = \arccos\,(^z/_r)$
Válcový souřadný systém	Sférický souřadný systém

Výše uvedené souřadné systémy jsou běžně známy a lze považovat za standardní. Jsou zajisté součástí standardního středoškolského vzdělání, zde jsou uvedeny jen pro úplnost.

Vedle výše uvedených souřadných systémů existuje celá řada dalších souřadných systémů používaných v praxi. Pro názornost uveďme toroidální souřadný systém:





Přehled mnoha dalších souřadných systémů lze nalézt např. na
http://mathworld.wolfram.com/topics/CoordinateGeometry.html

2.2.3. Datové typy

Používané datové typy vycházejí z datových standardů, dnes převážně ze standardu IEEE 578-2008. Ty jsou podporovány současnými programovými prostředky, a to buď přímo pomocí hardwarové podpory, nebo nepřímo pomocí softwarových knihoven.

	Name	**Digits**	**E min**	**E max**
B 16	Half	10+1	−14	15
B 32	Single	23+1	−126	127
B 64	Double	52+1	−1022	1023
B 128	Quad	112+1	−16382	16383

IEEE 578-2008 Floating point reprezentace

Zde je nutné zdůraznit, že součástí standardu IEEE 578-2008 jsou ještě definovány datové typy (http://en.wikipedia.org/wiki/IEEE_floating_point)

Name	Base	Digits	E min	E max	Decimal digits	Decimal E max
decimal32	10	7	−95	+96	7	96
decimal64	10	16	−383	+384	16	384
decimal128	10	34	−6143	+6144	34	6144

Pokud budeme více konkrétní, pak datové typy, které jsou obvykle k dispozici, jsou následující:

char, **byte**

integer, **long integer**

float, **double**, **extended**

complex , tj. $c = a + ib$ reprezentace komplexních čísel tento datový typ je k dispozici např. v jazycích FORTRAN xx

decimal - reprezentace dekadických hodnot používaná zejména u ekonomických výpočtů

Při reprezentaci hodnot a manipulaci s nimi musíme mít na paměti zejména spolehlivost (reliability) a robustnost (robustness), a to jak pro výpočty, tak i pro vizualizaci.

2.3.Homogenní souřadnice a jejich geometrická interpretace

Základní geometrické transformace jako posun (translation), rotace (rotation), změna měřítka (scaling), zkosení (shearing) jsou známé operace, včetně perspektivní projekce (perspective projection), která se obvykle používá. V následujícím výkladu zavedeme pojem projektivního rozšíření Eukleidovského prostoru (projektivní prostor) a homogenní souřadnice. V dalším textu se budeme snažit dodržovat značení pro:

- souřadnice v Eukleidovském prostoru $(X,Y)^T$, tedy velká písmena
- souřadnice v Eukleidovském prostoru $[x,y{:}\,w]^T$, tedy malá písmena.

2.3.1. Operace v Eukleidovském prostoru

Je známo, že operace posunu bodu (X,Y) o vzdálenost (A,B), je dána:

$$\begin{bmatrix} X' \\ Y' \end{bmatrix} = \begin{bmatrix} X \\ Y \end{bmatrix} + \begin{bmatrix} A \\ B \end{bmatrix} \tag{1}$$

a operace rotace bodu (X,Y) okolo počátku je určena:

$$\begin{bmatrix} X \\ Y' \end{bmatrix} = \begin{bmatrix} cos\varphi & -sin\varphi \\ sin\varphi & cos\varphi \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix} \tag{2}$$

Ostatní geometrické operace pro body lze definovat obdobně. Nicméně je zřejmé, že bude nutné kombinovat operace posunu a rotace, dělat též operace inverzní atd., což při Eukleidovské reprezentaci jednoduše nejde. V případě projektivního rozšíření Eukleidovského prostoru lze ukázat, že všechny základní geometrické transformace lze realizovat pomocí násobení matic. Toto je velmi výhodné, nejen z hlediska jednoduché reprezentace inverzních operací, neboť:

$$(\boldsymbol{AB})^{-1} = \boldsymbol{B}^{-1}\boldsymbol{A}^{-1} \tag{3}$$

ale i z důvodu hardwarové podpory geometrických transformací, neboť jsou všechny převedeny na tvar:

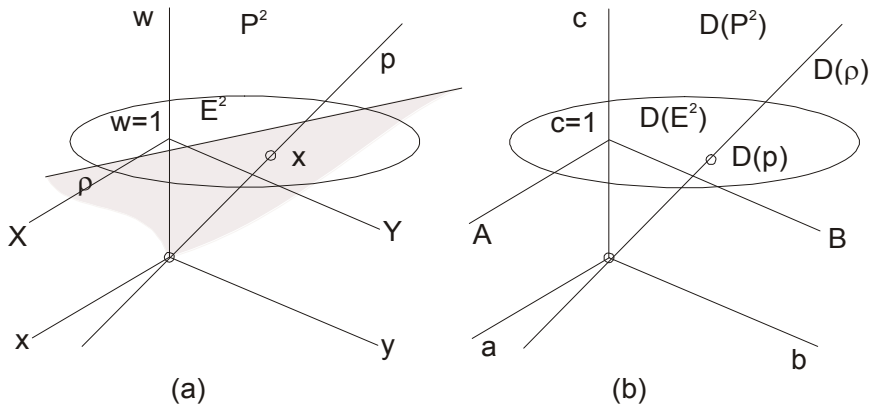
$$\boldsymbol{x}' = \boldsymbol{A}\boldsymbol{x} \tag{4}$$

kde $\boldsymbol{x} = [x,y{:}\,w]^T$, resp. $\boldsymbol{x}' = [x',y'{:}\,w']^T$ jsou souřadnice bodu $\boldsymbol{X}$, resp. $\boldsymbol{X}'$ v homogenních souřadnicích. Hodnota w se nazývá homogenní složkou vektoru $\boldsymbol{x}$ a je bezrozměrná, na rozdíl od hodnot x, y, které mají fyzikální rozměr, např. [m]. Proto hodnoty w budou oddělovány znakem ":".

Zde je asi vhodné krátce uvést projektivní rozšíření Eukleidovského prostoru, tedy vlastně projektivní rozšíření Eukleidovského souřadného systému, kde jsou hodnoty reprezentovány v homogenních souřadnicích. Jejich vzájemný převod:

	Dimenze = 2	Dimenze = 3
Eukleidovský prostor	$\boldsymbol{X} = (X,Y) \in E^2$	$\boldsymbol{X} = (X,Y,Z) \in E^3$
Projektivní prostor	$\boldsymbol{x} = [x,y{:}\,w]^T \quad w \neq 0$	$\boldsymbol{x} = [x,y,z{:}\,w]^T \quad w \neq 0$
Vzájemný převod	$X = \frac{x}{w} \quad Y = \frac{y}{w} \quad w \neq 0$	$X = \frac{x}{w} \quad Y = \frac{y}{w} \quad Z = \frac{z}{w}$
Alternativní popisy	$\tilde{\boldsymbol{x}} = [w{:}\,x,y,z]^T = [a_0{:}\,a_1,\ldots,a_n]^T$	*resp.* $\boldsymbol{x} = [a_0{:}\,a_1,\ldots,a_n]^T$

Z hlediska matematického popisu jde o poměrně jednoduchou záležitost. Podívejme se však, jaká je geometrická interpretace v případě E^2.



Projektivní rozšíření Eukleidovského prostoru a duální reprezentace

Eukleidovský prostor E^2 je vlastně rovinou v projektivním prostoru P^2 pro $w = 1$. Z obr.xxx vidíme, že bod $\boldsymbol{X}$ je vlastně reprezentován přímkou p v projektivním prostoru P^2 a všechny body kromě počátku ležící na přímce p jsou vlastně jednoparametrickou reprezentací tohoto bodu. V homogenních souřadnicích obecně nemusí být hodnota $w = 1$.
Pokud $w = 0$, pak jde o bod v nekonečnu, tzv. ideální bod (an ideal point – point in infinity), tedy směr. Vidíme tedy, že pomocí projektivní reprezentace můžeme reprezentovat též body v nekonečnu.

Zde je vhodné připomenout, že se někdy používá alternativní notace, kdy homogenní složka je na první pozici, tj.

$$\boldsymbol{x} = [w: x, y]^T \qquad \text{obecně pak pro } E^n \quad \boldsymbol{x} = [x_0: x_1, \dots, x_n]^T$$

Operace posuvu

Pokud nyní použijeme projektivní reprezentaci pro operaci posuvu bodů, pak dostáváme:

$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{bmatrix} 1 & 0 & A \\ 0 & 1 & B \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} x + Aw \\ y + Bw \\ w \end{bmatrix} \triangleq \begin{bmatrix} x/w + A \\ y/w + B \\ 1 \end{bmatrix} = \begin{bmatrix} X + A \\ Y + A \\ 1 \end{bmatrix}$$

kde $\triangleq$ značí projektivní ekvivalenci.

Operace rotace

$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{bmatrix} cos\varphi & -sin\varphi & 0 \\ sin\varphi & cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

Je tedy zřejmé, že se podařilo převést operaci posuvu bodu na maticové násobení, tj. operaci stejného typu jako je rotace a změna měřítka.

Je nutné upozornit, že transformace pro implicitní vyjádření přímek a rovin nejsou totožné s transformacemi pro body. Jednotlivé geometrické operace v projektivním prostoru budou vysvětleny později, viz kap.3 (Základní geometrické transformace).

2.4.Dualita a její aplikace

Použití projektivního rozšíření Eukleidovského prostoru je výhodné nejen pro „elegantní“ reprezentaci geometrických transformací, ale navíc poskytuje některé nové možnosti výpočtu geometrických entit, např. přímek nebo rovin, manipulace s nimi a též i elegantní řešení některých vybraných geometrických úloh.

2.4.1. Princip duality

Jednou ze základních vlastností reprezentace v projektivním prostoru je existence principu duality. Princip duality v P^2 říká, že:

- jakýkoliv teorém v E^2 zůstává platný, pokud zaměníme slova „bod“ a „přímka“, „leží na“ a “prochází“, „spojuje“ a „protíná“ apod.
- pokud byl dokázán jakýkoliv teorém, pak duální teorém dostaneme jak výše uvedeno.

To znamená, že ***jedním výpočetním postupem můžeme řešit jak primární úlohu, tak i úlohu duální.***
V dalším výkladu se nebudeme zabývat teoretickými vlastnostmi principu duality, ale tím, jak tento princip můžeme s výhodou využít pro řešení úloh počítačové grafiky.
Pro jednoduchost uvažme velmi jednoduchý případ v E^2, a to přímku p, která je dána v implicitní formě:

$$aX + bY + c = 0$$

Bez újmy na obecnosti můžeme rovnici vynásobit $w \neq 0$ a dostaneme

$$awX + bwY + cw = 0$$

a protože $x = wX$ a $y = wY$ můžeme psát:

$$ax + by + cw = 0$$

Ve vektorové notaci pak:

$$\boldsymbol{p}^T\boldsymbol{x} = 0$$

kde: $\boldsymbol{p} = [a, b: c]^T$, $\quad \boldsymbol{x} = [x, y: w]^T = [wX, wY: w]^T$.
Přímka p v E^2 je vlastně rovinou ρ v projektivním prostoru, přičemž bod $\boldsymbol{x} = [0,0: 0]^T$ je vyloučen.

Z notace $\boldsymbol{p^T x} = 0$ nelze určit význam jednotlivých symbolů, tj. zda $\boldsymbol{p}$ reprezentuje přímku a $\boldsymbol{x}$ reprezentuje bod nebo naopak v případě P^2. To znamená, že bod a přímka v E^2 jsou duální pojmy. V případě P^3 lze ukázat, že bod a rovina jsou pojmy duální.

Pochopitelně vzniká otázka, jak vlastně vypadá duální souřadný systém? Jeho geometrická reprezentace je na obr.XXX ad b (výše).

Je nutné upozornit, že:

- duální reprezentací se v principu nemění výpočetní složitost
- duální reprezentace umožňuje převést problém do duální reprezentace, následně se duální problém vyřeší a výsledek se pak převede zpět z duální reprezentace do reprezentace původní.

Příklad duálních primitiv a operátorů:

	Primitivum	**Duální primitivum**
P^2	bod přímka	přímka bod
P^3	bod rovina	rovina bod
	Operátor	**Duální operátor**
	sjednocení průsečík	průsečík sjednocení

Výsledkem je, že jednou programovou sekvencí můžeme tedy řešit primární a duální úlohu.

2.4.2. Vektorový součin a řešení soustav lineárních rovnic

Mnoho geometrických úloh vede k řešení soustavy lineárních rovnic, ať už s pravou nenulovou stranou, tj. $\boldsymbol{Ax} = \boldsymbol{b}$, nebo s nulovou pravou stranou, tj. $\boldsymbol{Ax} = \boldsymbol{0}$. K řešení soustavy $\boldsymbol{Ax} = \boldsymbol{b}$ se používají různé metody, přímé nebo iterační. Soustavy $\boldsymbol{Ax} = \boldsymbol{0}$ většinou představují trochu problém, neboť pokud existuje netriviální řešení, pak řešení je alespoň jedno-parametrické. V každém případě většinou prezentované postupy nejsou vhodné např. pro použití na GPU. V následujícím bude ukázána aplikace vektorového součinu pro řešení soustavy lineárních rovnic.

Vektorový součin

Nejprve malé opakování základů vektorových operací. Vektorový součin dvou vektorů v E^3 je definován

$$\boldsymbol{X}_1 \times \boldsymbol{X}_2 = det \begin{bmatrix} \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} \\ x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \end{bmatrix}$$

Vektorový součin lze vyjádřit i jako násobení matice vektorem, a to:

$$\boldsymbol{X}_1 \times \boldsymbol{X}_2 = \begin{bmatrix} 0 & -z_1 & y_1 \\ z_1 & 0 & -x_1 \\ -y_1 & x_1 & 0 \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} = \boldsymbol{T}\boldsymbol{x}_2$$

Nyní se podívejme, jak lze použít vektorový součin v projektivním prostoru.

Při použití vektorového součinu v projektivním prostoru P^2 dostáváme obdobné schéma, ale vektory $\boldsymbol{x}_1$ a $\boldsymbol{x}_2$ mají jiný význam, neboť reprezentují vlastně Eukleidovskou dvourozměrnou entitu. Nyní:

$$\boldsymbol{x}_1 = [x_1, y_1 : w_1]^T \qquad \boldsymbol{x}_2 = [x_2, y_2 : w_2]^T$$

Vektorový součin v P^2 je definován:

$$\boldsymbol{x}_1 \times \boldsymbol{x}_2 = det \begin{bmatrix} \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} \\ x_1 & y_1 & w_1 \\ x_2 & y_2 & w_2 \end{bmatrix}$$

kde: $\boldsymbol{i} = [1,0:0]^T \quad \boldsymbol{j} = [0,1:0]^T \quad \boldsymbol{k} = [0,0:1]^T$

nebo jako:

$$\boldsymbol{x}_1 \times \boldsymbol{x}_2 = \begin{bmatrix} 0 & -w_1 & y_1 \\ w_1 & 0 & -x_1 \\ -y_1 & x_1 & 0 \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \\ w_2 \end{bmatrix} = \boldsymbol{T}\boldsymbol{x}_2$$

Vektorový součin lze tedy opět vyjádřit i jako násobení matice vektorem.

Je nutné upozornit, že používáme reprezentaci v homogenních souřadnicích.

Průsečík dvou přímek

Uvažme dvě přímky, které jsou určeny vektory:

$$p_1 = [a_1, b_1 : c_1]^T \qquad p_2 = [a_2, b_2 : c_2]^T$$

Jejich průsečík je dán řešením soustavy rovnic:

$$a_1 x + b_1 y + c_1 = 0 \qquad a_2 x + b_2 y + c_2 = 0$$

tedy soustavy rovnic tvaru $\boldsymbol{Ax} = \boldsymbol{b}$

$$\begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} q_1 \\ q_2 \end{bmatrix} \quad \text{, kde} \quad \begin{bmatrix} q_1 \\ q_2 \end{bmatrix} = \begin{bmatrix} -c_1 \\ -c_2 \end{bmatrix}$$

Nyní se obvykle použije formule

$$x = \frac{Det_x}{Det} = \frac{det \begin{bmatrix} q_1 & b_1 \\ q_2 & b_2 \end{bmatrix}}{det \begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix}} \qquad y = \frac{Det_y}{Det} = \frac{det \begin{bmatrix} a_1 & q_1 \\ a_2 & q_2 \end{bmatrix}}{det \begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix}}$$

Programátor se nyní musí rozhodnout, zda a za jakých podmínek jde již o singulární řešení, tj. přímky jsou rovnoběžné, a kdy ještě ne (zde je nutné vidět koncept řešení i pro složitější soustavy rovnic).

Takže programátor „nadaný intuicí“ rozhodne, kdy Det je již příliš malý a způsobil by přetečení („overflow“) v pohyblivé řádové čárce. Tedy obvykle by se použila sekvence:

$$\boldsymbol{if}\ abs(\det(..)) \leq eps\ \boldsymbol{then} \ldots \ldots$$

Zkusme se nyní podívat na problém z druhé strany.

Věta

Nechť jsou dány dvě přímky $\boldsymbol{p}_1$ a $\boldsymbol{p}_2$. Pak souřadnice bodu $\boldsymbol{x}$, který průsečíkem těchto dvou přímek, jsou určeny vektorovým součinem homogenních souřadnic těchto dvou přímek, a to:

$$\boldsymbol{x} = \boldsymbol{p}_1 \times \boldsymbol{p}_2 \qquad x = [x, y : w]^T$$

přičemž

$$p_1 = [a_1, b_1 : c_1]^T \qquad p_2 = [a_2, b_2 : c_2]^T$$

kde: $\boldsymbol{x} = [x, y : w]^T$

Důkaz

Hledáme vlastně řešení následujících dvou rovnic

$$\boldsymbol{x}^T \boldsymbol{p}_1 = 0 \qquad \boldsymbol{x}^T \boldsymbol{p}_2 = 0$$

Poznámka: obvykle přímka v implicitní formě je dána jako $ax + by = q$ místo $ax + by + c = 0$, nebo v explicitní formě $y = kx + q$.

Na výše uvedeném příkladě bylo ukázáno, že k výpočtu průsečíku dvou přímek

- lze použít vektorový součin
- není zapotřebí operace dělení, neboť se jmenovatel v dělení vlastně „uloží“ do homogenní složky w souřadnice průsečíku $\boldsymbol{x}$
- pokud jsou přímky téměř rovnoběžné, pak hodnota $w \to 0$.

Zde je vhodné připomenout, že z důvodů limitované přesnosti výpočtů v pohyblivé řádové čárce:

- 3 body neleží na přímce,
- 4 body neleží na rovině,
- 2 přímky v prostoru se neprotínají,
- apod.

i když by se z čistě matematického hlediska měly!

Nyní se podívejme na řešení obdobné úlohy, a to určení přímky v E^2, která je určena dvěma body.

Přímka daná dvěma body

Nechť jsou dány dva body $\boldsymbol{x}_1$ a $\boldsymbol{x}_2$ a chceme určit koeficienty přímky p, která je určena těmito body. To znamená, že musíme určit 3 hodnoty, a to a, b, c ze dvou hodnot $\boldsymbol{x}_1, \boldsymbol{x}_2$, tj. musíme řešit rovnice:

$$ax_1 + by_1 + c = 0 \qquad ax_2 + by_2 + c = 0$$

Dostáváme tedy homogenní soustavu rovnic, tj. s nulovou pravou stranou. Je zřejmé, že jde o jednoparametrické řešení (pokud nejde o singulární případ, tj. dva body totožné). V maticové formě pak:

$$\begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \qquad \boldsymbol{Ax} = \boldsymbol{0}$$

Jak řešit danou soustavu rovnic? Toto je obvykle problém a programátoři obvykle se snaží zvolit jeden parametr, např. $c = 1.$, ale co když přímka prohází počátkem? Nebo $a = 1$ nebo $b = 1$. Také je možné řešení s použitím dodatečné podmínky $a + b = 1$ a pak:

$$\begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \qquad \boldsymbol{Ax} = \boldsymbol{b}$$

Toto je ale principiálně špatný přístup.

Ale tedy jak?

Elegantní a jednoduché řešení

Z principu duality víme, že přímka v E^2 je duální bodu a naopak. Díky principu duality tedy dostáváme řešení:

$\boldsymbol{x} = \boldsymbol{p}_1 \times \boldsymbol{p}_2$	<= **dualita** =>	$\boldsymbol{p} = \boldsymbol{x}_1 \times \boldsymbol{x}_2$
$\boldsymbol{Ax} = \boldsymbol{b}$	<= **ale proč různé?** =>	$\boldsymbol{Ax} = \boldsymbol{0}$

Věta plynoucí z principu duality

Nechť jsou dány dva body $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ v projektivním prostoru P^2. Pak koeficienty přímky p, která je určena těmito body, jsou určeny vektorovým součinem takto:

$$\boldsymbol{p} = \boldsymbol{x}_1 \times \boldsymbol{x}_2 = [a, b: c]^T$$

Důkaz

Nechť přímka p je v projektivním prostoru P^2 určena jako:

$$ax + by + cw = 0$$

pak rovnice přímky musí platit pro oba dva body, tj. musí platit rovnice:

$$\boldsymbol{p}^T \boldsymbol{x}_1 = 0 \qquad \boldsymbol{p}^T \boldsymbol{x}_2 = 0$$

kde: $\boldsymbol{p} = [a, b: c]^T$

Poznámka

Koeficient c „reprezentuje“ vzdálenost (je to vlastně násobek vzdálenosti) přímky od počátku souřadného systému, zatímco a, b reprezentují normálu přímky.

To znamená, že jakýkoliv bod $\boldsymbol{x}$, který leží na přímce p, musí vyhovovat oběma rovnicím výše a rovnici $\boldsymbol{x} = 0$. Jinými slovy vektor $\boldsymbol{p}$ je definován:

$$\boldsymbol{p} = \boldsymbol{x}_1 \times \boldsymbol{x}_2 = det \begin{bmatrix} \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} \\ x_1 & y_1 & w_1 \\ x_2 & y_2 & w_2 \end{bmatrix}$$

Pak můžeme psát:

$$(\boldsymbol{x}_1 \times \boldsymbol{x}_2)^T \boldsymbol{x} = 0 \qquad\qquad det\begin{bmatrix} x & y & w \\ x_1 & y_1 & w_1 \\ x_2 & y_2 & w_2 \end{bmatrix} = 0$$

Poznamenejme, že skalární a vektorový součin jsou instrukcemi v Cg/HLSL on GPU.

Vyhodnocením determinantu:

$$det\begin{bmatrix} a & b & c \\ x_1 & y_1 & w_1 \\ x_2 & y_2 & w_2 \end{bmatrix} = 0$$

pak dostáváme rovnici přímky p takto:

$$a = det\begin{bmatrix} y_1 & w_1 \\ y_2 & w_2 \end{bmatrix} \qquad b = -det\begin{bmatrix} x_1 & w_1 \\ x_2 & w_2 \end{bmatrix} \qquad c = det\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \end{bmatrix}$$

Obdobně jako v P^2 je princip duality platný i v prostoru P^3, ale musí se nadefinovat rozšíření vektorového součinu.

2.4.3. Aplikace duality v prostoru E^3

V prostoru P^3 jsou duální primitiva bod a rovina. Rovina je určena třemi body a bod je průsečíkem tří rovin. Parametry roviny a souřadnice průsečíku lze určit takto:

$$\boldsymbol{\rho} = \boldsymbol{x}_1 \times \boldsymbol{x}_2 \times \boldsymbol{x}_3 = \begin{vmatrix} \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} & \boldsymbol{l} \\ x_1 & y_1 & z_1 & w_1 \\ x_2 & y_2 & z_2 & w_2 \\ x_3 & y_3 & z_3 & w_3 \end{vmatrix} \qquad \boldsymbol{x} = \boldsymbol{\rho}_1 \times \boldsymbol{\rho}_2 \times \boldsymbol{\rho}_3 = \begin{vmatrix} \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} & \boldsymbol{l} \\ a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \\ a_3 & b_3 & c_3 & d_3 \end{vmatrix}$$

Je tedy zřejmé, že výpočet je opět velmi jednoduchý a není zapotřebí žádné dělení ani žádná volba hodnoty nějakého parametru apod.

V předchozím bylo ukázáno, že řešení obou typů soustav rovnic, tj.

$$\boldsymbol{Ax} = \boldsymbol{b} \qquad\qquad \boldsymbol{Ax} = \boldsymbol{0}$$

je speciálním případem při použití vektorového součinu.

Závěr

Vektorový součin

- je ekvivalentní řešení obou typů soustav lineárních rovnic
- není zapotřebí operace dělení.

Je důležité poznamenat, že souřadnice bodů $\boldsymbol{x}$, které mají obecně hodnotu homogenní složky $w \neq 1$, není při výpočtech zapotřebí převádět souřadnice do Eukleidovského prostoru, čímž se ušetří 2-3 operace dělení na 1 bod.

2.4.4. Vzdálenost v projektivním prostoru

Geometrie je úzce spojena s pojmem vzdálenosti a jejím měřením. V Eukleidovské geometrii je vzdálenost určena jako:

$$d = \sqrt{(\Delta x)^2 + (\Delta y)^2} \qquad \text{resp.} \qquad d = \sqrt{(\Delta x)^2 + (\Delta y)^2 + (\Delta z)^2}$$

Jedním z hlavních argumentů proti používání projektivní reprezentace je, že není jednoduše definována metrika. Toto sice eliminuje konformní geometrie, kterou se však nebudeme zabývat.

V případě projektivní reprezentace, vzdálenost dvou bodů je určena vztahem:

$$dist = \sqrt{\xi^2 + \eta^2}/(w_1 w_2)$$

kde: $\xi = w_1 x_2 - w_2 x_1 \qquad \eta = w_1 y_2 - w_2 y_1$

Vzdálenost bodu $\boldsymbol{x}_0$ od přímky v P^2 je určena vztahem:

$$dist = \frac{\boldsymbol{a}^T \boldsymbol{x}_0}{w_0 \sqrt{a^2 + b^2}}$$

kde: $\boldsymbol{x_0} = [x_0, y_0 : w_0]^T \qquad \boldsymbol{a} = [a, b : c]^T$

Extenze do E^3/P^3 je jednoduchá a vzdálenost bodu $\boldsymbol{x}_0$ od roviny je dána:

$$dist = \frac{\boldsymbol{a}^T \boldsymbol{x}_0}{w_0 \sqrt{a^2 + b^2 + c^2}}$$

kde: $\boldsymbol{x_0} = [x_0, y_0, z_0 : w_0]^T \qquad \boldsymbol{a} = [a, b, c : d]^T.$

V mnoha případech vlastně nepotřebujeme vzdálenost, např. pro určení, který bod je blíže, ale stačí nám monotónně rostoucí funkce vzdálenosti bodu $\boldsymbol{x}_i$ od přímky p, např. *distance*^{2} . Pak pro případ E^2 dostáváme:

$$dist_i{}^2 = \frac{(\boldsymbol{a}^T \boldsymbol{x}_i)^2}{w_i{}^2 (a^2 + b^2)} = \frac{(\boldsymbol{a}^T \boldsymbol{x}_i)^2}{w_i{}^2 \, \boldsymbol{n}^T \boldsymbol{n}}$$

kde: $\boldsymbol{a} = \lfloor a, b : c \rfloor^T = \lfloor \boldsymbol{n} : c \rfloor^T$ a normálový vektor $\boldsymbol{n}$ není normalizován. Tím můžeme podstatně snížit nároky na výpočetní výkon.

Pokud porovnáváme vzdálenosti více bodů $\boldsymbol{x}_i$, $i = 1, \ldots, n$ od dané přímky p, můžeme použít pro porovnání "*pseudodistance*":

$$(pseudo\_dist_i)^2 = \frac{(\boldsymbol{a}^T \boldsymbol{x}_i)^2}{w_i{}^2}$$

Analogicky pro případ roviny ρ v E^3:

$$dist_i{}^2 = \frac{(\boldsymbol{a}^T \boldsymbol{x}_i)^2}{w_i{}^2 (a^2 + b^2 + c^2)} = \frac{(\boldsymbol{a}^T \boldsymbol{x}_i)^2}{w_i{}^2 \, \boldsymbol{n}^T \boldsymbol{n}} \qquad \text{and} \qquad (pseudo\_dist_i)^2 = \frac{(\boldsymbol{a}^T \boldsymbol{x}_i)^2}{w_i{}^2}$$

kde: $\boldsymbol{a} = \lfloor a, b, c : d \rfloor^T = \lfloor \boldsymbol{n} : d \rfloor^T$

2.5.Numerická reprezentace a stabilita výpočtů

V současném technologicky orientovaném světě je zapotřebí posuzovat data, a tedy i informace, v širších souvislostech, kdy hlavními faktory jsou bezesporu:

- obrovský objem dat, který je nutné posoudit podle jednoho či více kritérií
- data jsou vícerozměrná, tj. multidimenzionální
- velká výpočetní náročnost potřebná nejen pro vlastní řešení, ale i pro generování výstupu např. pro vizualizaci,
- inovační technologicky cyklus je kratší než 6 měsíců

	Decimal usage	Binary usage
GigaByte [GB]	10^9	2^{30}
TeraByte [TB]	10^{12}	2^{30}
PetaByte [PB]	10^{15}	2^{30}
ExaByte [EB]	10^{18}	2^{30}
ZettaByte [ZB]	10^{21}	2^{30}
YottaByte [YB]	10^{24}	2^{30}
????	??	2^{64}

Dalšími faktory jsou pak také to, že

- každých 18 měsíců se zdvojnásobí výpočetní kapacita systémů, tj. za 15 let budou k dispozici pravděpodobně systémy s výpočetní i paměťovou kapacitou $2^{10} = 1024$ násobně vyšší než dnes,
- limitujícím faktorem zřejmě v dohledné době zůstane i přesnost výpočtů v pohyblivé řádové čárce daná standardem IEEE 578-2008 atd.
- objem získaných nebo generovaných dat, která jsou zpracovávána a zobrazována, bude narůstat rychleji než lineárně,
- lidská schopnost vyhodnocovat data zůstává víceméně stejná, neboť je dána fyziologií člověka.

Je tedy více než oprávněná otázka, jaké problémy budeme za 10 let řešit, jak je budeme počítat, jaká data a jakým způsobem je budeme zobrazovat, resp. vizualizovat, a pomocí jakých zařízeních.

Bohužel i v současnosti se lze setkat s typickou ignorancí problémů spojených s konečnou přesností reprezentace dat a lze běžně nalézt konstrukce, které typicky vedou k numerickým problémům.

Typickými ukázkami špatného kódu jsou:

- Testy A ? B pro hodnoty v pohyblivé řádové čárce, např.
  if A = B **then** ….. **else** ….. ; **if** A ≠ 0 **then** ….. **else** ….
  Takové konstrukce by měly být zakázány v programovacích jazycích.

- nekonečné cykly v důsledku hodnot v pohyblivé řádové čárce

```
double x = -1; double p =  ….;
while ( x < +1)
{          if (x == p) Console.Out.WriteLine(” *** ”);          x += p; }
/*     if p = 0.1 then no output,   if p = 0.25 then expected output */
```

2.5.1. Vybrané příklady a ukázky problémů

Z praxe je k dispozici celá řada příkladů, kdy numerické výpočty selhávají. Uveďme alespoň ty nejzákladnější, které by měl mít uživatel na paměti. (Pro názornost je použita jednoduchá přesnost).

Výpočet prosté sumy hodnot

$$\sum_{i=1}^{10^3} 10^{-3} = 0.999990701675415 \qquad \sum_{i=1}^{10^4} 10^{-4} = 1.000053524971008 \tag{5}$$

Výsledek by měl být vždy roven hodnotě 1

Závislost na pořadí výpočtů – výsledek také závisí na pořadí výpočtu

$$\sum_{n=1}^{10^6} \frac{1}{n} = 14.3\mathbf{57357} \qquad \sum_{n=10^6}^{1} \frac{1}{n} = 14.3\mathbf{92651} \tag{6}$$

Na výše uvedených příkladech vidíme, že výpočet, byť „primitivních“ matematických formulí, nemusí vést nutně ke správnému numerickému výsledku.

Konverze mezi datovými typy, které nemusí programový systém kontrolovat s následnou fatální ztrátou přesnosti výpočtu, např. **float** → **integer**

Neplatnost „matematických“ identit - i když se zdají být výše uvedené příklady „umělými“, je nutné si uvědomit, že ***matematické identity nejsou obecně platné*** při použití reprezentace čísel s omezenou přesností, např. identity

$$cos^2\alpha + cos^2\beta = 1 \qquad \mathrm{x}^2 - \mathrm{y}^2 = (x-y)(x+y)$$

Také řešení obyčejné kvadratické rovnice nemusí být korektní.

$at^2 + bt + c = 0$	obvyklé řešení	$t_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

Pokud $b^2 \gg 4ac$, pak by měla být použita „duální“ formule

$$q = -(b + sign(b)\sqrt{b^2 - 4ac}\,)/2 \qquad t_1 = {}^{q}/_{a} \qquad t_2 = {}^{c}/_{a}$$

Obecně diskriminant by měl být vypočten s dvojnásobnou přesností, neboť se počítá odmocnina.

Výše uvedená formule vychází z Vietova věty.

Vietova formule	$t_1 + t_2 = -{}^{b}/_{a}$	$t_1 t_2 = {}^{c}/_{a}$

Výpočet funkčních hodnot

Uveďme zde, sice trochu umělý, ale zajímavý problém, a to výpočet hodnoty funkce [reference]:

$$f(x,y) = 333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + x/(2y)$$

v bodě $x = 77617,\ y = 33096$. Výsledek, který obdržíme standardním výpočtem pak je:

- $f = 6.33835\ 10^{29}$ jednoduchá přesnost
- $f = 1{,}1726039400532$ dvojnásobná přesnost
- $f = 1{,}172603940053178631858834904520183\mathbf{8}$ „extended“ přesnost

avšak **správný výsledek** při použití intervalové aritmetiky

$$[-0{,}827396059946821368141165095479816292\mathbf{005},$$
$$-0{,}827396059946821368141165095479816291\mathbf{986}]$$

Přesné řešení je pak dáno výrazem: $f(x,y) = -2 + \frac{x}{2y} = \frac{54767}{66192}$

Existují tedy případy, kdy prostým prodlužováním mantisy nemusíme dojít ke korektnímu výsledku.

2.5.2. Rekurze

Rekurze je dalším potencionálním zdrojem chyb. Rekurze je vlastně volání funkce sebe sama, ať už přímo nebo zprostředkovaně, tj. tzv. vzájemnou rekurzí. Nebezpečnými faktory zejména jsou:

- rekurze, kdy hloubka rekurze i při středně rozsáhlém výpočtu nemusí být výpočetním systémem zvládnutelná. Uveďme např. problém „Hanojských věží“, který je definován sekvencí:

  MOVE (A, C, n);
  { MOVE (A, B, n-1); MOVE (A, C, 1); MOVE (B, C, n-1)
  } # MOVE (from, to, number) #

- rekurze, při níž datový typ není schopen reprezentovat hodnotu pro její velikost, tj. při „přetečení“ bez detekce chybného stavu. Toto je spojeno zejména s reprezentací celých čísel, kdy přetečení není detekováno současným hardwarem.

  Typickou ukázkou je Ackermannova funkce:

$$A(m,n) = \begin{cases} n+1 & if\ m = 0 \\ A(m-1,1) & if\ m > 0\ and\ n = 0 \\ A\big(m-1, A(m,n-1)\big) & if\ m > 0\ and\ n > 0 \end{cases}$$

  jejíž hodnoty rostou extrémně rychle, např.

$$A(4,4) = 2^{2^{2^{65536}}} = 2^{2^{10^{197296}}}$$

  viz http://en.wikipedia.org/wiki/Ackermann_function

2.5.3. Další možnosti zvýšení přesnosti

Pro zvýšení přesnosti je možné použít např.:

- intervalové aritmetiky, kdy se předpokládá, že uložená hodnota vlastně reprezentuje celý interval hodnot. Uveďme základní pravidla pro intervalovou aritmetiku:
  - x + y = [a + c, b + d] x = [a , b]
  - x - y = [a - d, b - c] y = [c , d]
  - x × y = [min(ac, ad, bc, bd), max(ac, ad, bc, bd)]
  - x / y = [min(a/c, a/d, b/c, b/d), max(a/c, a/d, b/c, b/d)] if $y \neq 0$

- speciální reprezentace hodnot, např. pomocí řetězových zlomků apod. Pro názornost uveďme vyjádření hodnoty $\pi = [3; 7,15,1,292,1,1,1,2,1,3,1 \dots]$, což je reprezentace řetězového zlomku

$$\pi = \cfrac{4}{1 + \cfrac{1^2}{3 + \cfrac{2^2}{5 + \cfrac{3^2}{\dots}}}}$$

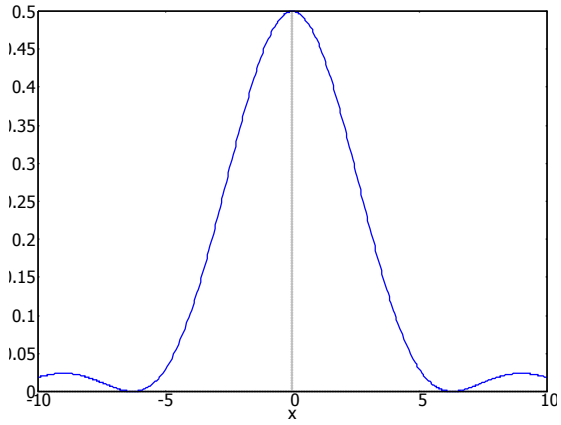
V praxi lze nalézt mnoho výpočetních postupů, které se snaží eliminovat vliv konečné reprezentace různými způsoby. Výše uvedený přehled snad poslouží k náhledu, že tato problematika není rozhodně jednoduchá a že v praxi se často vyskytuje.

2.5.4. Příklad překvapivé nestability výpočtu

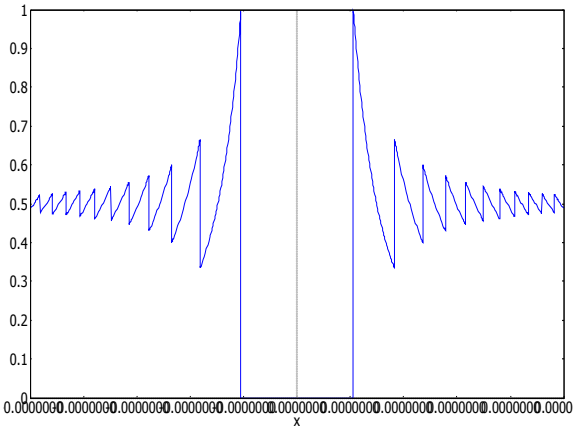
Další ukázkou nepřesnosti výpočtu je funkce

$$f(x) = \frac{1 - \cos x}{x^2}$$

Graf funkce je uveden na obr.xxx.a. Lze ukázat, že $f(0) = 0{,}5$. Nicméně pro hodnoty okolo počátku se funkce chová „zvláštně“, viz obr.xx.b vlivem numerické nepřesnosti a přesný průběh závisí silně na realizaci výpočtu vzorce a použitém HW (vypočtená hodnota funkce je v intervalu $(-\varepsilon, \varepsilon)$ dokonce **nulová**).



Průběh funkce $f(x) = \frac{1-\cos x}{x^2}$



Průběh funkce okolo počátku $\langle -10^{-8}, 10^{-8} \rangle$

Poznámka

Je nutné si uvědomit, že hodnota funkce $f(0)$ není definována pouze pro $x = 0$, neboť jde o výraz $^0/_0$, avšak numerické problémy vznikají již v poměrně velkém okolí tohoto bodu.

Hodnotu $f(0)$ lze určit pomocí L'Hospitalova pravidla:

$$f(0) = \lim_{x\to 0} \frac{1 - \cos x}{x^2} = \lim_{x\to 0} \frac{g(x)}{h(x)} = \lim_{x\to 0} \frac{g'(x)}{h'(x)} = \lim_{x\to 0} \frac{\sin x}{2x}$$

což je opět výraz $^0/_0$. Opětovným použitím L'Hospitalova pravidla dostáváme:

$$f(0) = \lim_{x\to 0} \frac{\sin x}{2x} = \lim_{x\to 0} \frac{\cos x}{2} = 0{,}5$$

Zde je nanejvýš vhodné zdůraznit, že existuje celá řada dnes méně známých, ale stále používaných programovacích jazyků, které poskytují specifické datové typy a zajímavé datové konstrukce, např. programovací jazyk **Algol68** má konstrukci **long**, která prodlužuje délku reprezentace, a tedy i přesnost datového typu. Programátor pak může mít i velmi dlouhou mantisu použitím konstrukce **long ……long real** A.

Jak je vidět, někdy i „koncepčně zastaralé“ programovací jazyky nabízejí více než ty „nové“.

2.6.Příklady katastrof způsobených výpočetními chybami

Je známa celá řada katastrof způsobených numerickým výpočtem. Po přečtení původní zprávy o příčinách, následné analýzy a doprovodných komentářů každý musí být překvapen, jaké elementární chyby se staly, jak se podcenilo testování správnosti a korektnosti výpočtů. Bohužel se zřejmě stále častěji bude stávat, že takové chyby nejsou eliminovány a vzhledem ke vzrůstající složitosti systémů budou pravděpodobně i důsledky takových katastrof větší. Uveďme alespoň pro ilustraci několik nejznámějších případů.

Exploze rakety Ariane 5

Evropská agentura ESA (European Space Agency) vypustila 4. června 1996 raketu Ariane 5. Její vývoj stála přes 7 miliard US$. Cena vlastního nákladu a rakety byla přes 500 miliónu US$. Raketa explodovala asi 40 [s] po startu. Příčinou bylo selhání programového vybavení inerciálního naváděcího systému SRI (Inertial Reference System), které bylo zapříčiněno konverzí 64 bitového čísla v pohyblivé řádové čárce do 16 bitového čísla v celočíselné reprezentaci a hodnota byla větší než reprezentovatelná na 16 bitech. Podrobné informace lze nalézt:

http://www.ima.umn.edu/~arnold/disasters/ariane5rep.html

Courtesy CNN

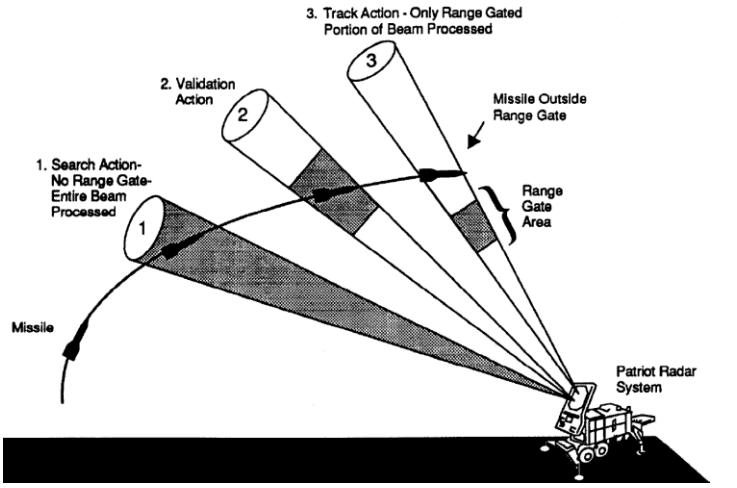
Selhání antirakety Patriot

Systém protivzdušné raketové obrany Patriot byl původně navržen pro krátké a operativní nasazení po roce 1960. Byl navržen pro sestřelení raket s rychlostí MACH 2. Systém byl při praktickém nasazení v pohotovostním režimu přes 100 hod. a byl použit pro sestřelení rakety typu SCUD letící rychlostí MACH 5. Výpočet pro detekci a určení pozice byly založeny na celočíselné reprezentaci o délce 24 bitů a s časovým taktem 1/10 [s] a výpočtem rychlosti v pohyblivé řádové čárce. Nastavení časového taktu 1/10 [s] bylo kritickým bodem, neboť není ani postačující pro sportovní účely.

Bohužel $1/10 = 1/2^4+1/2^5+1/2^8+1/2^9+1/2^{12}+....$nemá konečnou reprezentaci a chyba na 42 bitech byla asi 0.000000095. V průběhu 100 hod tak dosáhla hodnoty 0.34. Protože rakety typu SCUD mají rychlost MACH 5, aktuální chyba byla 687[m] a raketa byla mimo „okno“ pro samonavádění. O daném problému se vědělo, příslušný software korigující problém byl k dispozici, ale pracovníci neprovedli příslušný update programového vybavení.

Důsledkem špatných předpokladů, nekorektní technické a programové realizace a nezodpovědného přístupu bylo, že 25.února 1991 při útoku v Dhahranu iráckou raketou SCUD bylo 28 US vojáků zabito, 100 lidí na základně zraněno. Podrobnosti lze nalézt v reportu GAO

http://www.fas.org/spp/starwars/gao/im92026.htm



Courtesy of GAO report

Dalším velmi známým případem je potopení těžební plošiny Sliepner A v Severním moři.

Potopení těžební plošiny Sleipner A

V roce 1991 se potopila těžební plošina Sleipner A. Jen pro ilustraci uveďme, že samotná těžební plošina vážila přes 57 000 tun, nástroje a zařízení pak 40 000 tun a na plošině pracuje průběžně přes 200 osob. Struktura těžební plošiny byly „optimalizována“ s použitím systému založeného na konečných prvcích a smyková napětí byla podhodnocena téměř o 50%. Toto vedlo k porušení struktury a trhlinám s následným průnikem vody, který pumpy nebyly schopny zvládnout. Odhadovaná cena potopené těžební plošiny činí cca 700 miliónů US$.

Courtesy of SINTEF

Až dosud jsme se zabývali elementárními datovými typy. Nicméně je asi vhodné se podívat na problém dat i z jiné strany. V převážné většině dnes běžně zpracovávaných dat lze najít dvě hlavní datové množiny, které jsou obvykle viděny samostatně, a to geometrická data, obrazová data a data textová.

3. Základní geometrické transformace

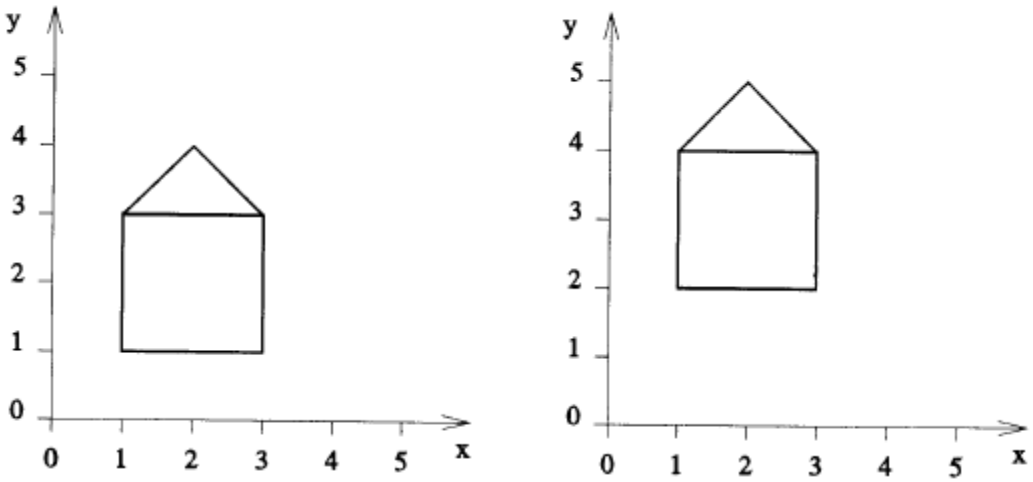
Geometrické transformace v počítačové grafice jsou využívány ke geometrickým manipulacím s objekty, resp. jejich částí, při vytváření objektů, tj. modelování, při animaci, kdy potřebujeme „rozpohybovat“ jednotlivé části objektu, v počítačových hrách atd. Mezi základní operace patří zejména posuv, rotace, změna měřítka a také rovinné projekce. Jednotlivé základní operace, které jsou reprezentovány jednotlivými maticemi geometrických transformací, se pak řetězí a vzniká pak jedna matice, která reprezentuje celkovou geometrickou transformaci. Zde je vidět výhoda použití projektivní reprezentace, viz kap.2.3 (Homogenní souřadnice a jejich geometrická interpretace). Jednotlivé transformace jsou popsány jako transformace pro jeden bod, je však nutné si uvědomit, že daná transformace se aplikuje na všechny body daného objektu, resp. té části, která se transformuje. U reálných úloh je takto transformováno běžně 10^5 - 10^{10} a i více bodů. Navíc v mnoha případech se vyžaduje, aby celý proces včetně generování výstupního obrazu probíhal v reálném čase.

Všechny geometrické transformace budou popsány rovnicí $\boldsymbol{x}' = \boldsymbol{Q}\boldsymbol{x}$ se *sloupcovou* notací vektorů. Je nutné zdůraznit, že mnoho publikací používá *řádkovou* notaci a tedy výše uvedený vztah v řádkové notaci má tvar $\boldsymbol{x}'^T = \boldsymbol{x}^T\boldsymbol{Q}^T$, tj. matice transformace je transformovaná. To v případě E^2 znamená např. rotaci o úhel opačný.

3.1.Základní transformace v E^2

Pro jednoduchost uveďme nejdříve základní operace v rovině, tj. v E^2.

Posuv (translation)



Geometrická transformace posuvu je reprezentována vztahem:

$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{bmatrix} 1 & 0 & A \\ 0 & 1 & B \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} \qquad \boldsymbol{x}' = \boldsymbol{T}(A,B)\boldsymbol{x}$$

V dalším textu budeme používat zkráceného zápisu $\boldsymbol{x}' = \boldsymbol{T}(A,B)\boldsymbol{x}$. Všimněme si, že $det(\boldsymbol{T}) = 1$. Rozepsáním a použitím formálních úprav je zřejmé, že výše uvedený vztah reprezentuje posuv o vektor (A,B):

$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{bmatrix} 1 & 0 & A \\ 0 & 1 & B \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} x + Aw \\ y + Bw \\ w \end{bmatrix} \triangleq \begin{bmatrix} x/w + A \\ y/w + B \\ 1 \end{bmatrix} = \begin{bmatrix} X + A \\ Y + A \\ 1 \end{bmatrix}$$

kde $\triangleq$ značí projektivní ekvivalenci.

Zde je vhodné připomenout, že homogenní w souřadnice transformovaných bodů může být $w \neq 1$. Tedy není nutná jejich konverze do Eukleidovského prostoru.

Inverzní operace posuvu

Inverzní operace je jednoduchá, neboť jde vlastně o posuv daný vektorem $(-A, -B)$ a tedy:

$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{bmatrix} 1 & 0 & -A \\ 0 & 1 & -B \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} \qquad \boldsymbol{x}' = \boldsymbol{T}(-A, -B)\boldsymbol{x} = \boldsymbol{T}^{-1}(A, B)$$

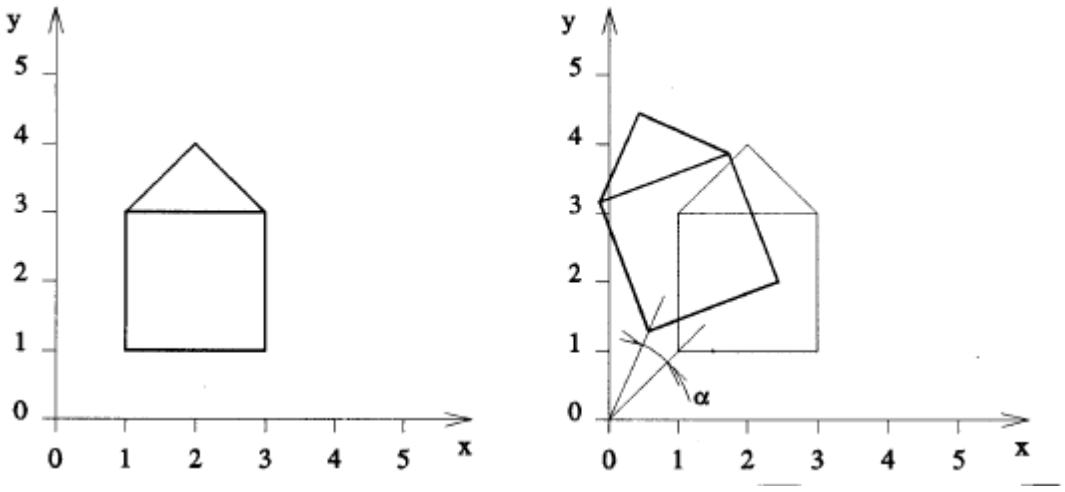
Lze ukázat, že pokud vektor posuvu $[a, b: c]^T \triangleq (A, B)$ je v homogenních souřadnicích, tj. obecně $c \neq 1$, lze operaci posuvu realizovat bez použití operace dělení. V tomto případě je pak transformace určena:

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} c & 0 & a \\ 0 & c & b \\ 0 & 0 & c \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} cx + aw \\ cy + bw \\ cw \end{bmatrix} \triangleq \begin{bmatrix} (cx + aw)/(cw) \\ (cy + bw)/(cw) \\ 1 \end{bmatrix} = \begin{bmatrix} x/w + a/c \\ y/w + b/c \\ 1 \end{bmatrix} = \begin{bmatrix} X + A \\ Y + B \\ 1 \end{bmatrix}$$

Další základná transformací je transformace rotace.

Rotace (rotation)

Operace rotace je operace, u které je nutné zdůraznit, že jde o rotaci *okolo počátku souřadného systému.* Pokud střed rotace není v počátku souřadného systému, pak jde o složenou transformaci, podrobněji viz kap.3.2 (Řetězení transformací).



Rotace bodu o úhel φ proti směru hodinových ručiček je určena vztahem:

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} cos\varphi & -sin\varphi & 0 \\ sin\varphi & cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} \qquad \boldsymbol{x}' = \boldsymbol{R}(\varphi)\boldsymbol{x}$$

Inverzní operace rotace

Inverzní operace je opět jednoduchá, neboť jde vlastně o rotaci s úhlem opačným, tj. o úhel – φ

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} cos(-\varphi) & -sin(-\varphi) & 0 \\ sin(-\varphi) & cos(-\varphi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} cos\varphi & sin\varphi & 0 \\ -sin\varphi & cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} \qquad \boldsymbol{x'} = \boldsymbol{R}(-\varphi)\boldsymbol{x}$$

Opět platí, že $det(\boldsymbol{R}) = 1$, tj. matice $\boldsymbol{R}$ je ortonormální a $\boldsymbol{R}^{-1}(\varphi) = \boldsymbol{R}(-\varphi) = \boldsymbol{R}^{T}(\varphi)$.

Lze ukázat, že pokud je rotace určena vektorem, který projektivně reprezentuje rotaci ve tvaru $[a, b: c]^T \triangleq \left(\frac{a}{c}, \frac{b}{c}\right) = (cos\varphi, sin\varphi)$, pak transformace je určena vztahem:

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} a & -b & 0 \\ b & a & 0 \\ 0 & 0 & c \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} \qquad \boldsymbol{x'} = \boldsymbol{R'}(a, b: c)\boldsymbol{x}$$

Rozepsáním pak lze ukázat korektnost vztahu:

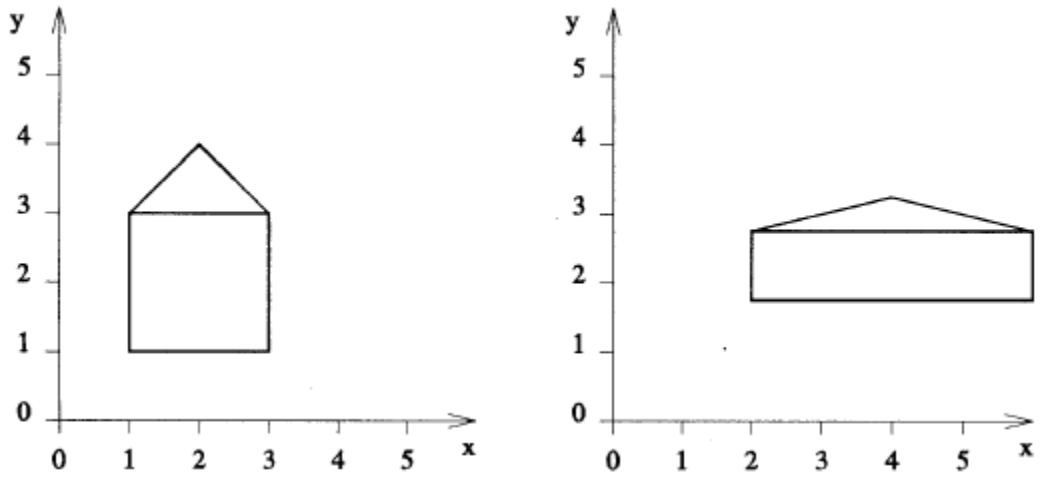
$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} ax - by \\ bx + ay \\ cw \end{bmatrix} \triangleq \begin{bmatrix} (ax - by)/(cw) \\ (bx + ay)/(cw) \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{x}{w}\frac{a}{c} - \frac{y}{w}\frac{b}{c} \\ \frac{x}{w}\frac{b}{c} + \frac{y}{w}\frac{a}{c} \\ 1 \end{bmatrix} = \begin{bmatrix} Xcos\varphi - Ysin\varphi \\ Xsin\varphi + Ycos\varphi \\ 1 \end{bmatrix}$$

Je nutné podotknout, že $\det(\boldsymbol{R'}) = (a^2 + b^2)c = c^3$, neboť $c^2 = (a^2 + b^2)$. Toto ale nevadí, neboť operace jsou realizovány v projektivním prostoru a není tedy nutné dělení pro převod parametrů rotace do Eukleidovského prostoru.

Další základní geometrickou operací je změna měřítka.

Změna měřítka (scaling)

Operace změny měřítka je operací, která umožňuje zvětšení či zmenšení, ať už proporcionální v každé ose nebo neproporcionální. Je opět nutné zdůraznit, že operace má počátek souřadného systému jako referenční bod.



Operace změny měřítka je určena vztahem:

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} \qquad \boldsymbol{x}' = \boldsymbol{S}(S_x, S_y)\boldsymbol{x}$$

kde S_x,resp. S_y určují změnu měřítka v ose x, resp. y.

Je zřejmé, že $det(\boldsymbol{S}) = S_x\, S_y$ tedy obecně $det(\boldsymbol{S}) \neq 1$.

Inverzní operace změny měřítka

Inverzní operace změny měřítka je určena vztahem

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} \frac{1}{S_x} & 0 & 0 \\ 0 & \frac{1}{S_y} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} \qquad \boldsymbol{x}' = \boldsymbol{S}^{-1}(S_x, S_y)\boldsymbol{x} = \boldsymbol{S}\left(\frac{1}{S_x}, \frac{1}{S_y}\right)\boldsymbol{x}$$

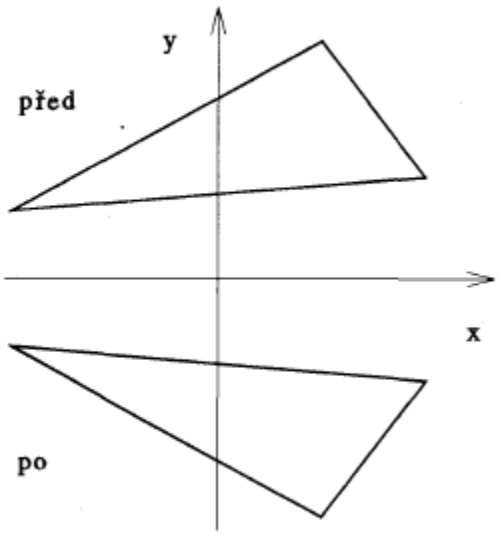
Lze ukázat, že pokud je změna měřítka určena vektorem, který projektivně reprezentuje změnu měřítka ve tvaru $[s_x, s_y : w_s]^T \triangleq (\frac{s_x}{w_s}, \frac{s_y}{w_s}) = (S_x, S_y)$, pak transformace je určena vztahem

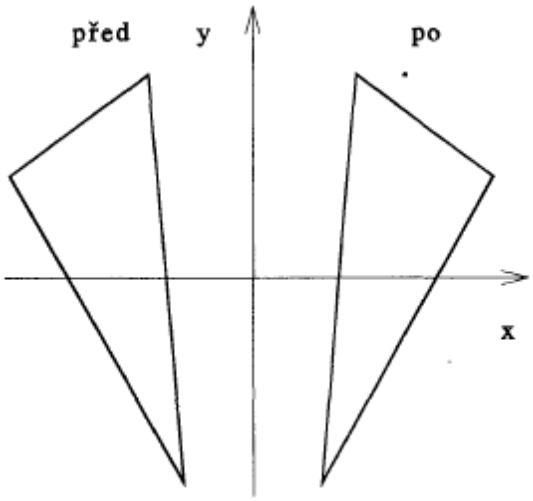
$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & w_s \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

přičemž $det(\boldsymbol{S}') = s_x\, s_y\, w_s$.

Zrcadlení (mirroring)

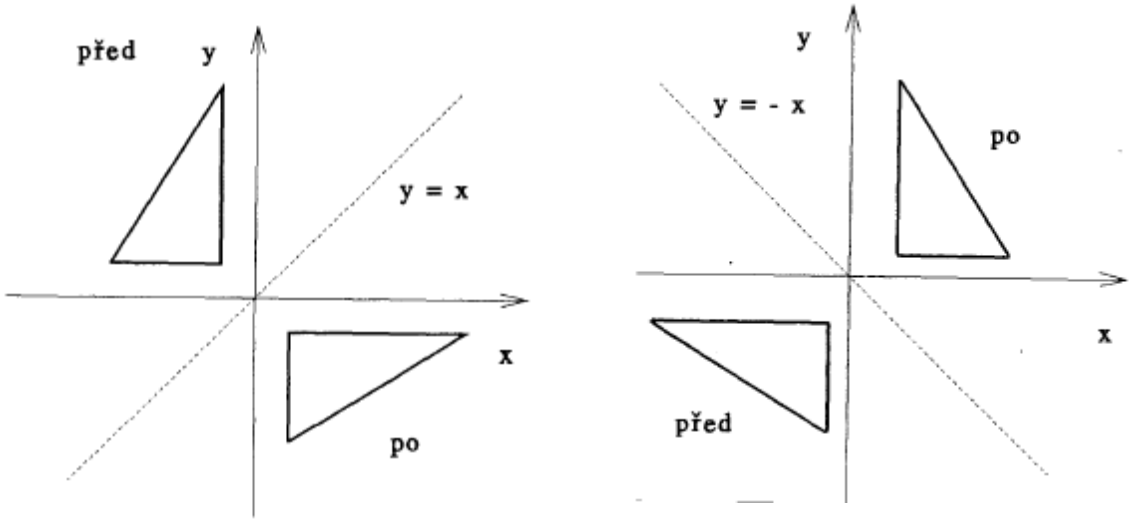
Zrcadlení podle nějaké osy je opět častou operací. Nejjednodušším případem je ten, kdy osa zrcadlení je totožná s osou souřadného systému viz Obr.xxxx





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

Osou zrcadlení může být i obecná osa. Uvažme opět jednoduchý případ, kdy osou je přímka $y = x$, resp. $y = -x$, viz obr.QQQQ



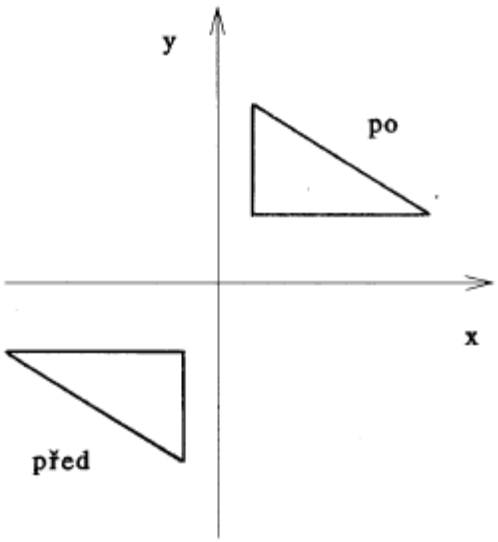
Obr.QQQQ

Transformační vztahy jsou pak určeny takto:

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

Posledním typem zrcadlení je zrcadlení vzhledem k počátku, viz obr.WWW

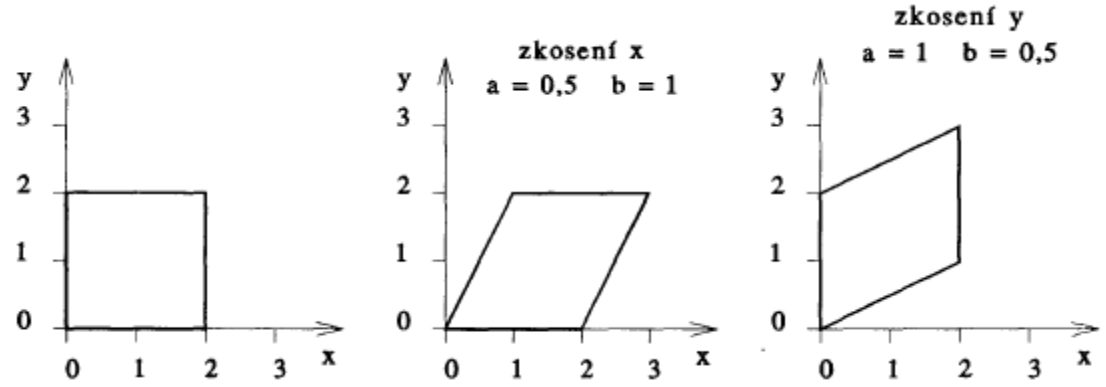
$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$



Obr.WWW

Poslední „základní“ geometrickou transformací, kterou zde uvedeme, je zkosení.

Zkosení (shearing)



Původní stav

Obr.TTTT

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & a & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} \qquad \begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ b & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

Příklad

Odvoďte vztahy pro geometrickou transformaci, která je dána maticí:

$$\boldsymbol{H}_{xy}(a,b) = \begin{bmatrix} 1 & a & 0 \\ b & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

a výsledek nakreslete pro $a = 0.5$ a $b = 0.7$ pro situaci z obr.TTT.a.

Až dosud byly ukázány základní geometrické operace. Vedle těchto operací jsou ještě složitější operace, které jsou vyjádřitelné jako kompozice základních operací. Tento postup se označuje pojmem „řetězení operací“.

3.2.Řetězení transformací

Řetězení geometrických transformací slouží k realizaci složitějších transformací. Představme si, že máme n po sobě jdoucích transformací popsaných transformačními maticemi $\boldsymbol{Q}_i$, $i = 1, \ldots, n$. Pak tedy jednotlivé transformační kroky jsou:

$$\boldsymbol{x}_1 = \boldsymbol{Q}_1\boldsymbol{x} \qquad \boldsymbol{x}_2 = \boldsymbol{Q}_2\boldsymbol{x}_1 \qquad \boldsymbol{x}_n = \boldsymbol{Q}_n\boldsymbol{x}_{n-1}$$

Celková transformace je pak dána:

$$\boldsymbol{x}_n = \boldsymbol{Q}_n \ldots \boldsymbol{Q}_2\boldsymbol{Q}_1\boldsymbol{x}_1 = \boldsymbol{Q}\boldsymbol{x}_1$$

přičemž $\boldsymbol{Q} = \boldsymbol{Q}_n \ldots \boldsymbol{Q}_2\boldsymbol{Q}_1$.

V praxi se všechny elementární transformace $\boldsymbol{Q}_i$ realizující danou složitější transformaci akumulují do výsledné matice $\boldsymbol{Q}$, která pak reprezentuje celkovou transformaci.

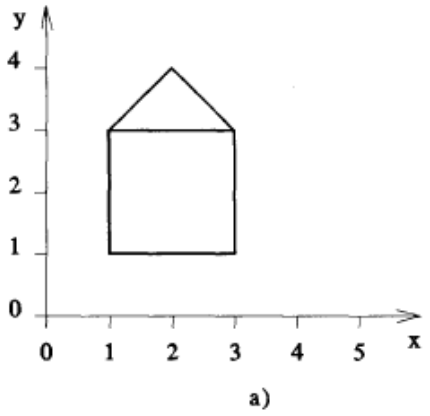
Připomeňme, že *násobení matic není komutativní*, tj. obecně platí že $\boldsymbol{AB} \neq \boldsymbol{BA}$.

Příklady

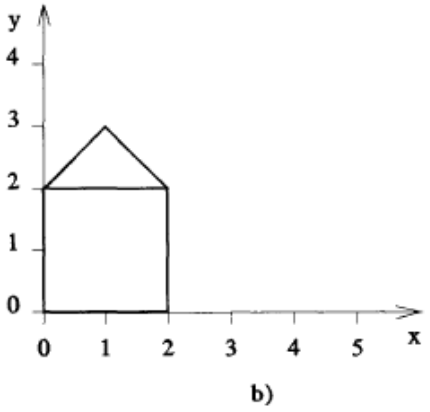
Pro názornost uveďme jednoduché příklady, a to:

- otočení objektu okolo daného bodu, který je obecně odlišný od počátku souřadného systému
- relativní změna měřítka

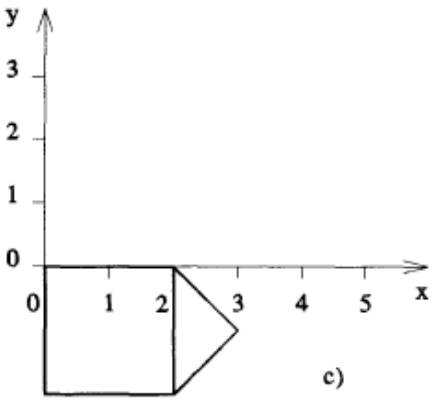
Příklad 1 - Otočení objektu okolo daného bodu



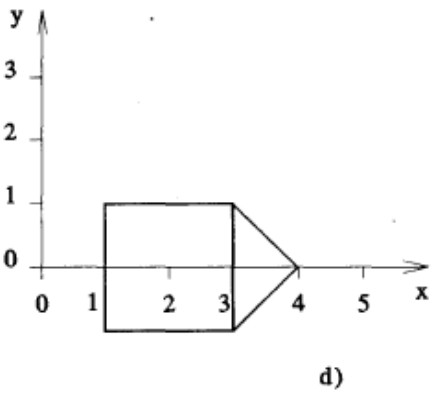
Původní stav



Po posunutí referenčního bodu do počátku



Po otočení o požadovaný úhel



Po posunutí referenčního bodu do původní pozice

Daná úloha se sestává z následujících kroků, které jsou aplikovány na všechny body objektu:

- posuv tak, aby referenční bod rotace (a, b), tj. v našem případě levý dolní roh objektu, byl v počátku
- pootočení o požadovaný úhel φ, v našem případě o úhel $\varphi = -\pi/2$
- posunutí referenčního bodu do původní pozice

Takže nyní můžeme specifikovat jednotlivé transformační kroky

$$\boldsymbol{x}' = \boldsymbol{T}(-a, -b)\boldsymbol{x} \qquad \boldsymbol{x}'' = \boldsymbol{R}(\varphi)\boldsymbol{x}' \qquad \boldsymbol{x}''' = \boldsymbol{T}(a, b)\boldsymbol{x}''$$

Výsledná transformační matice $\boldsymbol{Q}$ je dána vztahem:

$$\boldsymbol{Q} = \boldsymbol{T}(a, b)\, \boldsymbol{R}(\varphi)\, \boldsymbol{T}(-a, -b) = \begin{bmatrix} cos\varphi & -sin\varphi & -acos\varphi + bsin\varphi + a \\ sin\varphi & cos\varphi & -asin\varphi - bcos\varphi + b \\ 0 & 0 & 1 \end{bmatrix}$$

Po dosazení hodnot pro úhel φ dostáváme:

$$\boldsymbol{Q} = \begin{bmatrix} 0 & 1 & -b + a \\ -1 & 0 & -a + b \\ 0 & 0 & 1 \end{bmatrix}$$

Vidíme, že výsledná matice $\boldsymbol{Q}$ má vlastně strukturu se submaticemi, resp. vektory, a to:

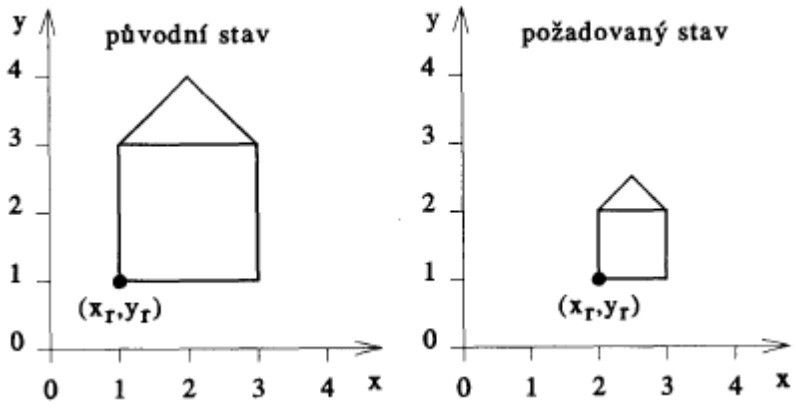
$$\boldsymbol{Q} = \begin{bmatrix} \boldsymbol{Q}_R & \boldsymbol{Q}_T \\ \boldsymbol{Q}_P & 1 \end{bmatrix}$$

kde $\boldsymbol{Q}_R$ reprezentuje vlastně rotaci, $\boldsymbol{Q}_T$ reprezentuje posuv (translaci) a $\boldsymbol{Q}_P$, která je zatím nulová, reprezentuje např. perspektivní projekci z E^3 do E^2, viz kap. 4 (Projekce).

Dalším příkladem je relativní změna měřítka.

Příklad 2 - Relativní změna měřítka

V některých případech může být i změna měřítka záludnou operací. Uvažme jednoduchý objekt, který chceme zmenšit v poměru 1:2 a referenční bod (x_r, y_r) posunout do nové pozice. Je zřejmé, že pokud vynásobíme souřadnice vrcholů transformační maticí $S\left(\frac{1}{2}, \frac{1}{2}\right)$, dostaneme výsledek nesprávný, neboť se posune i referenční bod do pozice $\left(\frac{1}{2}, \frac{1}{2}\right)$.



Takže nyní můžeme specifikovat jednotlivé transformační kroky:

$$\boldsymbol{x}' = \boldsymbol{T}(-a, -b)\boldsymbol{x} \qquad \boldsymbol{x}'' = \boldsymbol{S}(S_x, S_y)\boldsymbol{x}' \qquad \boldsymbol{x}''' = \boldsymbol{T}(a', b')\boldsymbol{x}''$$

Výsledná transformační matice $\boldsymbol{Q}$ je dána vztahem:

$$\boldsymbol{Q} = \boldsymbol{T}(a', b')\, \boldsymbol{S}(S_x, S_y)\, \boldsymbol{T}(-a, -b)$$

kde (a, b) je původní pozice referenčního bodu, (a', b') je nová pozice referenčního bodu.

V předchozím bylo ukázáno, jak se jednotlivé geometrické transformace spojují tak, aby bylo možné realizovat složité geometrické transformace.

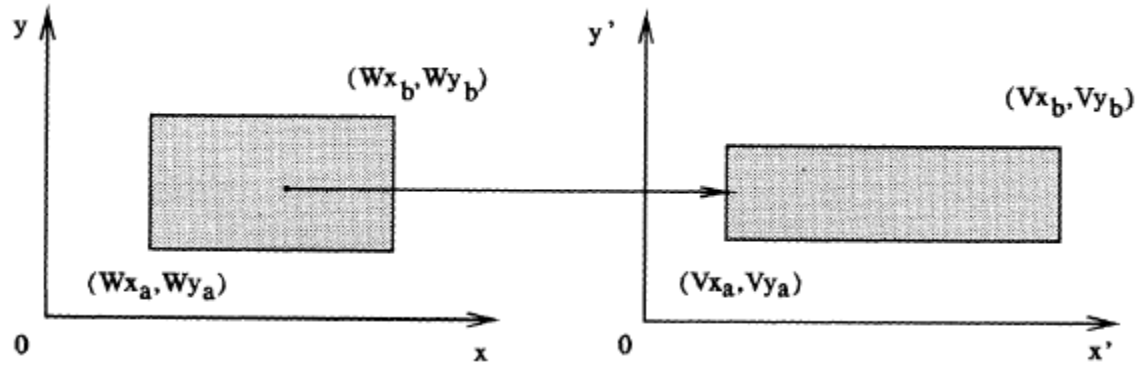
Jednou ze specifických transformací je transformace Window – Viewport (okno – pohled).

3.3.Window - Viewport transformace

Transformace Window – Viewport (okno – pohled) je velmi specifickou a velmi často používanou. Tato transformace se obecně sestává z více kroků, a to:

- změna měřítka a změna pozice referenčního bodu
- a volitelným odříznutí částí, které nejsou ve specifikované oblasti. Odříznutí části objektů, které nejsou v této oblasti, se nazývá „ořezávání“, resp. „clipping“ a tato operace je popsána v kap.5 (Metody ořezávání v E^2 a E^3 a operace s n-úhelníky).

Tato transformace zajišťuje ať už přímo, nebo nepřímo vlastně transformaci ze souřadného systému ve kterém je zpracovávaný objekt definován do souřadného systému daného výstupního média, např. obrazovky.



Transformace je složena z kroků:

- posuvem bodů tak, že referenční bod (Wx_a, Wy_a) je v počátku – transformace $\boldsymbol{T}_1$
- změnou měřítka tak, aby se interval $(\boldsymbol{Wx}_a, \boldsymbol{Wx}_b)$ transformoval na interval $(\boldsymbol{Vx}_a, \boldsymbol{Vx}_b)$
- posuvem bodů tak, že referenční bod je v pozici (Vx_a, Vy_a) – transformace $\boldsymbol{T}_2$

Celková transformace je pak určena transformační maticí:

$$\boldsymbol{Q} = \boldsymbol{T}_2\, \boldsymbol{S}\, \boldsymbol{T}_1$$

Jednotlivé transformace pak jsou určeny takto:

$$\boldsymbol{T}_1 = \begin{bmatrix} 1 & 0 & -Wx_a \\ 0 & 1 & -Wy_a \\ 0 & 0 & 1 \end{bmatrix} \qquad \boldsymbol{T}_2 = \begin{bmatrix} 1 & 0 & Vx_a \\ 0 & 1 & Vy_a \\ 0 & 0 & 1 \end{bmatrix}$$

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

a celková matice $\boldsymbol{Q}$ je určena:

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

kde:

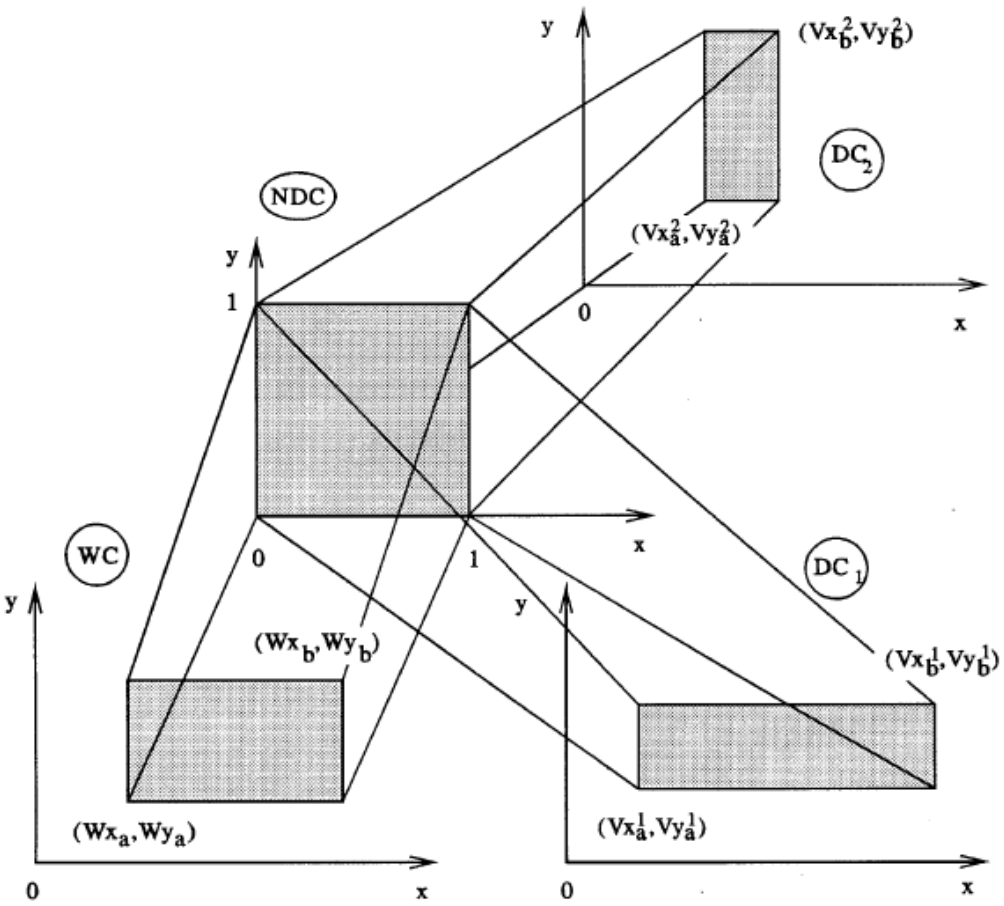
$$\alpha = (Vx_b - Vx_a)/(Wx_b - Wx_a) \qquad \gamma = Vx_a - \alpha Wx_a$$
$$\beta = (Vy_b - Vy_a)/(Wy_b - Wy_a) \qquad \delta = Vy_a - \beta Wy_a$$

Je zřejmé, že pokud některé části vykreslované scény přesahují velikost výstupního okna, resp. média apod., $(\boldsymbol{Vx}_a, \boldsymbol{Vx}_b)$ je nutno tyto části odříznout.

Velmi důležitým konceptem je „Normalizovaný souřadný“ (NDC) systém.

3.4.Normalizovaný souřadný systém

Koncept normalizovaného souřadného systému NDC (Normalized Device Coordinates) je klíčový nejen pro oblast počítačové grafiky.



V systémech počítačové grafiky jsou používány různé souřadné systémy v závislosti na kontextu:

- souřadný systém zařízení **DC** (Device Coordinates) – tento souřadný systém pracuje většinou ve fyzických jednotkách daného konkrétního zařízení. Typickým příkladem je souřadný systém obrazovky, kdy jednotkou je pixel a vertikální osa je orientována směrem dolů
- souřadný systém **WC** (World Coordinates) – v tomto souřadném systému je reprezentován objekt a pozice bodu může při stejné numerické hodnotě znamenat různě velké hodnoty, vzdálenosti apod. a navíc hodnoty mohou být v různé reprezentaci.
- souřadný systém normalizovaných souřadnic **NDC** (Normalized Device coordinates) - umožňuje efektivní transformaci mezi WC a DC souřadnými systémy. Za mapování mezi WC a NDC je zodpovědná aplikace, např. GIS systém, zatímco za mapování mezi NDC a DC je pak zodpovědný ovladač daného fyzického zařízení.

Je tedy zřejmé, že jde jen o vícenásobnou aplikaci transformace Window – Viewport, která byla uvedena dříve. Zavedením konceptu NDC se umožnilo snadné připojování vstupních a výstupních periférií k aplikačním programům.

Až dosud byly předloženy geometrické transformace v rovině, tj. v prostoru E^2. Nicméně prostor, ve kterém se pohybujeme, je třírozměrný, tj. E^3. Je tedy otázkou, jak jsou definovány geometrické transformace v E^3.

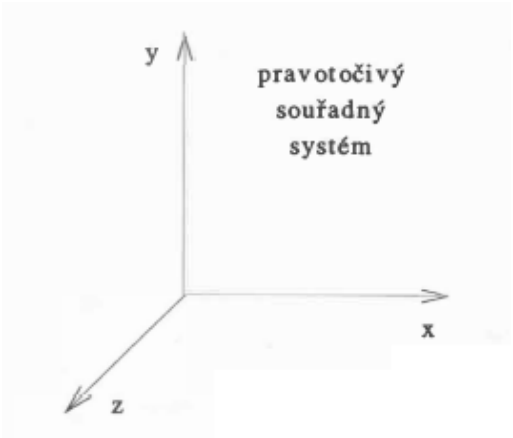
3.5.Základní transformace v E^3

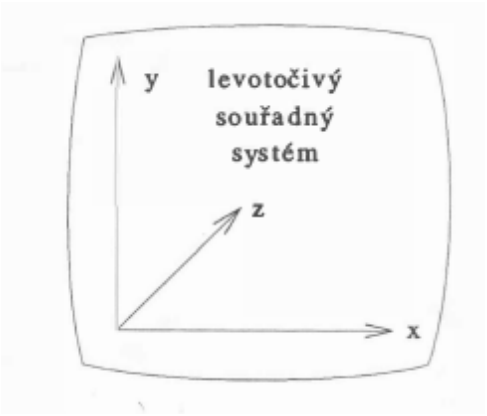
V případě dvourozměrného systému je orientace os systému vžitá, tj. osa x je horizontální s orientací doprava, osa y je vertikální s orientací nahoru, v případě obrazových aplikací je orientace osy y směrem dolu. V případě třírozměrného kartézského souřadného systému situace poněkud komplikovanější, neboť jsou dvě orientace souřadného systému, a to:

- *pravotočivý*, který bude implicitně používán pro geometrické operace v WC souřadném systému
- *levotočivý*, který bude použit pro reprezentaci pozice pozorovatele, resp. kamery.

Určení orientace souřadného systému pomocí pravé dlaně – pokud osa x protíná dlaň a prsty mají stejný směr jako osa y a palec ukazuje ve směru osy z, pak daný souřadný systém je pravotočivý. V opačném případě je levotočivý

Poznamenejme, že některé publikace, zejména pokud používají řádkovou notaci pro vektory, používají implicitně levotočivý souřadný systém.





Nyní uveďme základní geometrické transformace v E^3, které jsou jen vlastně jen přímým prostým rozšířením rovinným operací dříve definovaných. Pochopitelně vektor $\boldsymbol{x}$ nyní obsahuje navíc složku pro osu z:

$$\boldsymbol{x} = [x, y, z{:}\, w]^T$$

Operace posunutí

Operace posunutí o (A, B, C) je určena vztahem:

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & A \\ 0 & 1 & 0 & B \\ 0 & 0 & 1 & C \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \qquad \boldsymbol{x}' = \boldsymbol{T}(A, B, C)\boldsymbol{x}$$

Operace posuvu je nyní reprezentována maticí 4×4, místo matice 3×3.

Operace změny měřítka

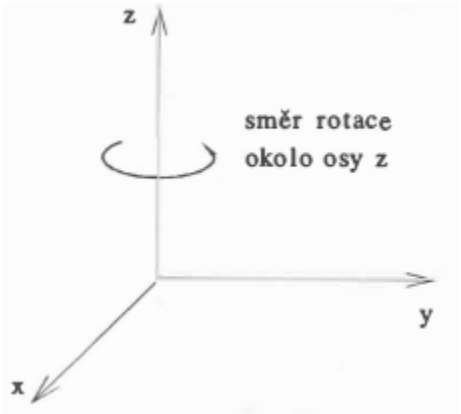
Operace změny měřítka je obdobná, a to:

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \qquad \boldsymbol{x}' = \boldsymbol{S}(S_x, S_y, S_z)\boldsymbol{x}$$

Operace rotace

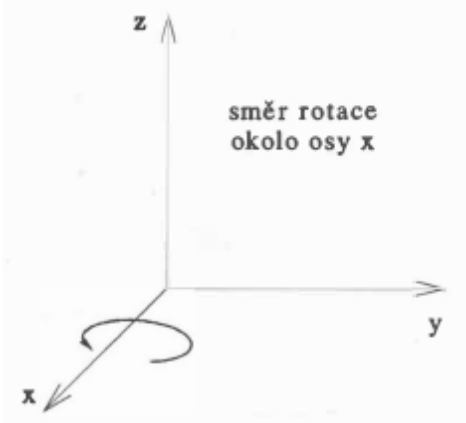
Operace rotace je poněkud složitější, neboť v rovinném případě jsme rotovali vlastně v rovině xy, tj. okolo „virtuální“ osy z, která směřovala vzhůru, tj. k pozorovateli. V případě třírozměrném máme 3 základní osy, okolo kterých rotaci můžeme definovat, resp. a 3 základní roviny, ve kterých můžeme rotaci realizovat.

Z faktického hlediska je jedno, zda mluvíme o rotaci v rovině nebo okolo osy. Nicméně po formální stránce je vhodnější mluvit o rotaci v rovině xy, yz nebo zx, neboť znaménko *minus* v matici bude vždy u řádku, který odpovídá 1. písmenku v notaci rotace. Pořadí písmenek v notaci není libovolné, neboť je dáno pravotočivostí souřadného systému. Např. rotace v rovině xz místo zx by vlastně byla rotací v levotočivém souřadném systému, tedy vlastně rotací s opačným úhlem.



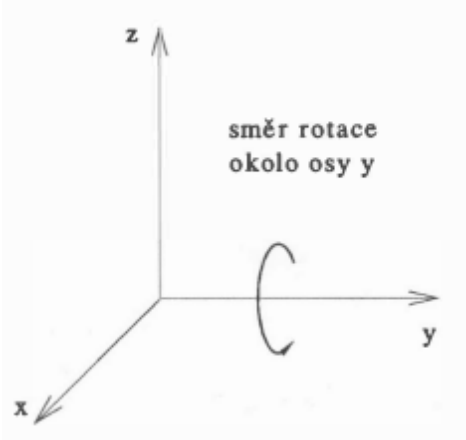
$$\boldsymbol{R}(\varphi)_{xy} = \begin{bmatrix} cos\varphi & -sin\varphi & 0 & 0 \\ sin\varphi & cos\varphi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotace okolo v rovině xy, tj. okolo osy z



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

Rotace okolo v rovině yz, tj. okolo osy x



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

Rotace okolo v rovině zx, tj. okolo osy y

Inverzní operace

Konstrukce inverzních operací v E^3 je analogická konstrukci inverzních operací v E^2.

Upozornění

Mnoho publikací používá řádkovou notaci pro vektory nebo levotočivý souřadný systém jako základní souřadný systém. V tomto případě jsou transformační matice transponované a pořadí transformací je opačné, tj. nikoliv zprava doleva, ale zleva doprava. Nelze tedy „míchat“ transformace ze zdrojů s řádkovou notací a se sloupcovou notací jednoduše dohromady.

Rotace je vlastně jednou z nejdůležitějších operací, neboť zobrazovaný objekt chceme prohlížet na daném výstupním zařízení, např. obrazovce, a pohyb v E^3 budeme zřejmě ovládat nějakým zařízením, např. myší, které je vlastně zařízením pracujícím v E^2. Takže vedle rotace budou zapotřebí ještě další transformace apod.

V praxi se však vyskytuje obecnější případ a to rotace okolo dané osy v prostoru E^3. Toto je už složitější geometrická transformace, která je dána jako kompozice základních geometrických transformací. Stejně jako u rovinných geometrických transformací lze použít řetězení geometrických transformací k získání složitých geometrických transformací.

Vidíme, že základní geometrické transformace jsou poměrně jednoduché z hlediska matematického popisu. Pro realizaci složitějších geometrických transformací se opět používá zřetězení geometrických transformací, které bylo uvedeno v kap.3.2 (Řetězení transformací).

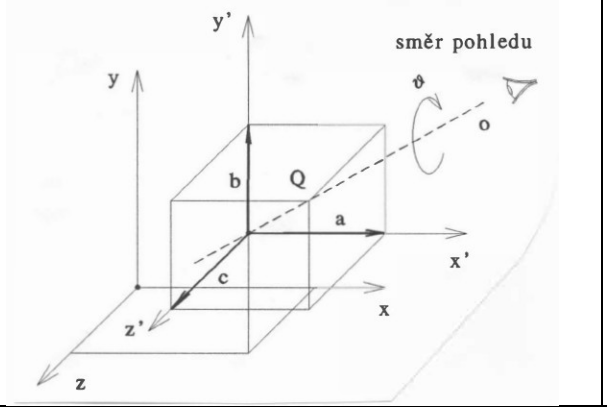
3.6.Rotace okolo dané osy v E^3

Rotace okolo dané osy v E^3 již není triviální geometrickou transformací. Uvažme jednoduchou situaci, a to:

- je dána osa rotace bodem a směrovým vektorem v E^3, tj.

$\boldsymbol{x}(t) = \boldsymbol{x}_A + \boldsymbol{s}\,t$	$\boldsymbol{s} = [a,b,c]^T$

- všechny body ve scéně se mají pootočit o úhel ϑ v naznačeném směru, viz obr.xxxx



ve skutečnosti nebudeme posouvat souřadný systém, ale objekty

Tato složená transformace má několik základních kroků (uvedená posloupnost transformace rotací není jedinou možnou), a to:

- posuv souřadného systému do bodu $\boldsymbol{x}_A$, resp. všechny body se posunou odpovídajícím způsobem do geometricky ekvivalentní pozice. Toto je nutné, neboť rotace má počátek souřadného systému za referenční bod
- rotace tak, aby přímka ležela v nějaké základní rovině, např. v rovině xy, tj. uděláme rotaci v rovině yz
- rotace tak, aby přímka byla totožná s nějakou osou, např. na ose x, tj. uděláme rotaci v rovině xy
- rotace o úhel ϑ
- inverzní operace provedené před rotací o úhel ϑ v opačném pořadí

Takže dostáváme matici $\boldsymbol{Q}$ kumulované transformace:

$$\boldsymbol{Q} = \boldsymbol{T}^{-1}\boldsymbol{R}_{yz}^{-1}\boldsymbol{R}_{xy}^{-1}\boldsymbol{R}(\varphi)\boldsymbol{R}_{xy}\boldsymbol{R}_{yz}\boldsymbol{T}$$

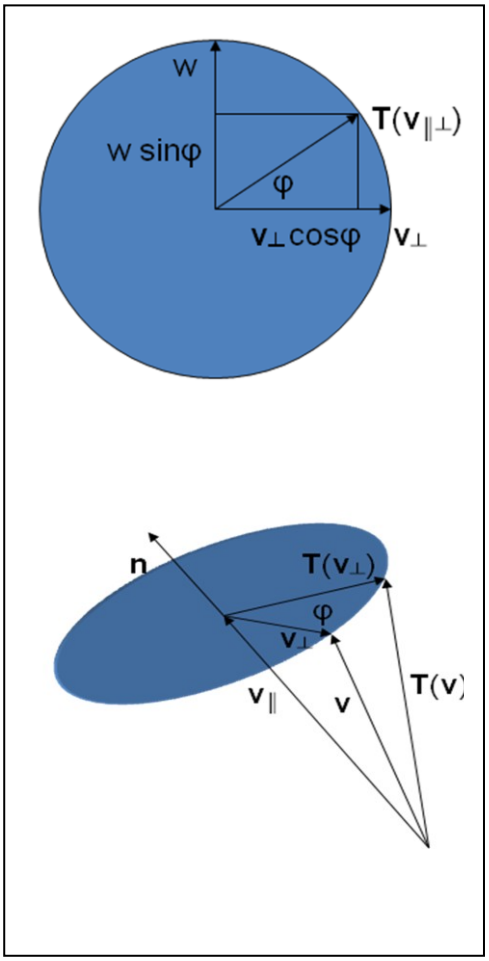
Podíváme-li se však pozorněji, např. na matici $\boldsymbol{R}_{yz}$

$$\boldsymbol{R}_{yz} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \dfrac{c}{\sqrt{b^2+c^2}} & \dfrac{-b}{\sqrt{b^2+c^2}} & 0 \\ 0 & \dfrac{b}{\sqrt{b^2+c^2}} & \dfrac{c}{\sqrt{b^2+c^2}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

pak vidíme, že transformace není stabilní, pokud $\sqrt{b^2+c^2} \to 0$, nebo je nepřesná pokud $b^2 \gg c^2$. Navíc další nepřesnosti vznikají násobením matic. V následujícím ukážeme, jak transformace může být elegantně řešena s podstatně lepší výpočetní přesností a stabilitou.

3.7.Transformace rotace v E³ založená na rotaci vektorů – dopracovat

Uveďme nyní zcela jiný přístup ke geometrickým transformacím.



Je dána osa rotace procházející počátkem souřadného systému a úkolem je pootočení scény o úhel φ. Pro daný bod $\boldsymbol{X}$ jde tedy o rotaci na rovině v obecné poloze, která má normálový vektor $\boldsymbol{n}$, přičemž normálový vektor je normalizován.

Transformace je pak dána vztahem:

$$\boldsymbol{X} = \boldsymbol{X}\,cos\varphi + (1 - cos\varphi)(\,\boldsymbol{n}^T\boldsymbol{X}).\boldsymbol{n} + (\boldsymbol{n} \times \boldsymbol{X})sin\varphi$$

$$\boldsymbol{Q} = \boldsymbol{I}cos\varphi + (1 - cos\varphi)(\boldsymbol{n}\otimes\boldsymbol{n}) + \boldsymbol{W}sin\varphi$$

kde: $\boldsymbol{n}\otimes\boldsymbol{n} = \boldsymbol{n}\,.\,\boldsymbol{n}^T$ je matice
(operace ⊗ se někdy nazývá tenzorový součin vektorů)

V Eukleidovském prostoru vektor $\boldsymbol{n}$ musí být normalizován a matice $\boldsymbol{W}$ je definována jako: $\boldsymbol{W}\boldsymbol{v} = \boldsymbol{w} \times \boldsymbol{v}$, tj. :

$$\boldsymbol{W} = \begin{bmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & w_x & 0 \end{bmatrix}$$

Pro odvození se použije „trik“, a to:

$$\boldsymbol{v} = \boldsymbol{v}_{\parallel} + \boldsymbol{v}_{\perp}$$

tedy „rozdělení“ vektoru v na dva vektory navzájem na sebe kolmé.

Podrobněji viz

- Miller,J.R.: The Mathematics of Graphical Transformations: Vector Geometric and Coordinate-Based Approaches, DesignLab, 1997 (CLICK PDF off-line)
- Miller,J.R.: Vector Geometry for Computer Graphics, IEEE Computer Graphics and Applications, pp.66-73, May/June 1999 (CLICK PDF off-line)
- Miller,J.R.: Applications of Vector Geoemtry for Robustness and Speed, IEEE Computer Graphics and Applications, pp.68-73, July/AugustMay/June 1999 (CLICK PDF off-line)

3.8. Transformace přímek a rovin

Dosud předložené geometrické transformace se týkaly transformace bodů. Nicméně počítačová grafika používá také přímky a roviny. Často vzniká otázka, jaká se změní rovnice přímky nebo roviny, pokud se body, které je definují, transformují. Typickou úlohou je problém stínování plochy, pokud se s danou plochou manipuluje, např. otáčí, mění se měřítko, které je v různých směrech různé atd. V předešlém textu bylo ukázáno, že normála není vlastně vektor, ale bivektor, neboť výsledek vektorového součinu dvou vektorů je orientovaná plocha.

Z předchozího je známo, že duální primitiva jsou:

	E^2	E^3
	$\boldsymbol{p} = \boldsymbol{x}_1 \times \boldsymbol{x}_2$	$\boldsymbol{\rho} = \boldsymbol{x}_1 \times \boldsymbol{x}_2 \times \boldsymbol{x}_3$
Duální problém	$\boldsymbol{x} = \boldsymbol{p}_1 \times \boldsymbol{p}_2$	$\boldsymbol{x} = \boldsymbol{\rho}_1 \times \boldsymbol{\rho}_2 \times \boldsymbol{\rho}_3$

Pokud se body definující přímku nebo rovinu transformují podle vztahu

$$\boldsymbol{x}' = \boldsymbol{T}\boldsymbol{x}$$

vzniká otázka, jaká bude implicitní reprezentace přímky resp. roviny. Lze ukázat, že přímka p po transformaci bodů, které ji definují, bude určena vektorem $\boldsymbol{p}'$takto:

$$\boldsymbol{p}' = (\boldsymbol{T}\boldsymbol{x}_1) \times (\boldsymbol{T}\boldsymbol{x}_2) = det(\boldsymbol{T})(\boldsymbol{T}^{-1})^T\boldsymbol{p} \triangleq (\boldsymbol{T}^{-1})^T\boldsymbol{p}$$

kde $\triangleq$ znamená projektivní ekvivalenci, neboť rovnice přímky je implicitní a je ji možné násobit libovolnou nenulovou hodnotou a pozice přímky se nezmění. Takže transformovaná přímka p je určena vektorem:

$$\boldsymbol{p}' = (\boldsymbol{T}^{-1})^T\boldsymbol{p} = [a', b' : c']^T$$

Obdobně pro případ roviny, jejíž definiční body byly transformovány, dostáváme obdobný vztah:

$$\boldsymbol{\rho}' = (\boldsymbol{T}\boldsymbol{x}_1) \times (\boldsymbol{T}\boldsymbol{x}_2) \times (\boldsymbol{T}\boldsymbol{x}_3) = det(\boldsymbol{T})(\boldsymbol{T}^{-1})^T\boldsymbol{\rho} \triangleq (\boldsymbol{T}^{-1})^T\boldsymbol{\rho}$$

a transformovaná rovina ρ je určena vektorem:

$$\boldsymbol{\rho}' = (\boldsymbol{T}^{-1})^T\boldsymbol{\rho} = [a', b', c' : d']^T$$

Závěr

Z výše uvedeného je podstatné to, že:

- při geometrických transformacích přímky nebo roviny je nutné normálu násobit maticí $(\boldsymbol{T}^{-1})^T$, což je matice obecně různá od matice $\boldsymbol{T}$, tj. obecně $(\boldsymbol{T}^{-1})^T \neq \boldsymbol{T}$
- není nutné transformovat body a následně počítat koeficienty transformované přímky, či roviny, lze určit koeficienty transformované přímky, či roviny přímo.

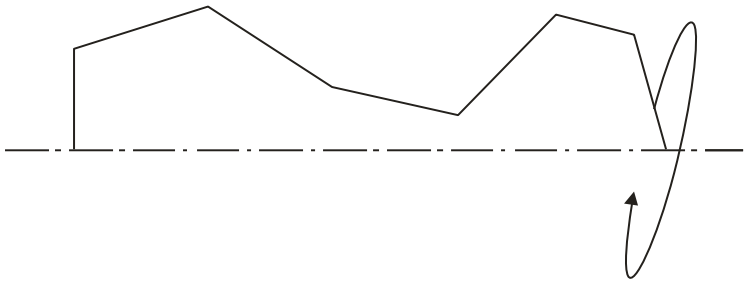
V předchozí části byly ukázány základní principy geometrických transformací se základními geometrickými primitivy.

3.9.Generace rotačního tělesa

Jednou z častých úloh je generace povrchu rotačního objektu daného obrysem, např. lomenou čárou.

Úkolem je vytvořit plášť rotačního objektu s daným profilem a s osou rotace x. Tato úloha už není zcela triviální, neboť vedle generace vlastních bodů a jejich transformací, je nutné ještě zvážit

- datovou strukturu, která se použije
- způsob reprezentace povrchu pomocí trojúhelníků, které grafické akcelerátory zpracovávají
- otázku vykreslování, tj. drátěný model, stínovaný s konstantním stínováním, resp. stínováním Gouraud (metody stínování viz kap.12.3 (Lokální metody). Je tedy nutné určit normály povrchu generovaných trojúhelníků, resp. normály ve vrcholech generovaných trojúhelníků



Obr.XXX

Rotační tělesa jsou velmi častá. Rotační symetrie umožňuje v mnoha inženýrských aplikacích podstatné snížení výpočetní složitosti, neboť umožňuje snadné řešení rotačně symetrického problému, např. elektro-magnetického pole, v E^2 místo E^3, tj. dojde ke snížení dimenze.

Transformace pro rotaci okolo osy x pak jsou:

$$x' = x$$
$$y' = y\,cos\varphi - z\,sin\varphi$$
$$z' = y\,sin\varphi + z\,cos\varphi$$

Poznámka

V tomto příkladě je jednodušší určovat normálu ve vrcholech, než normálu generovaných trojúhelníků. Je vhodné tento příklad „dotáhnout“ do konce včetně implementace.

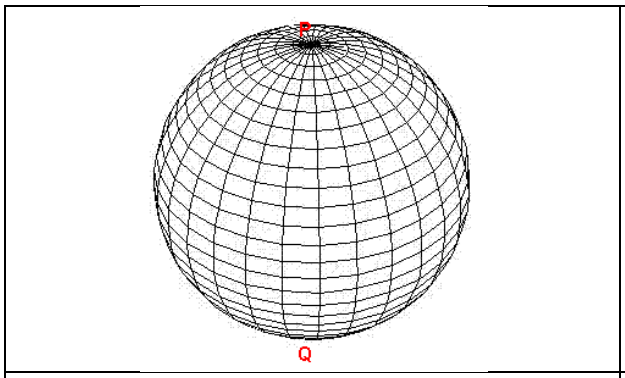
3.10. Generace povrchu koule

Dalším často generovým povrchem je povrch koule. V mnoha aplikacích se použije přímá aplikace sférických souřadnic. Body sítě se pak generují v pravidelné síti parametrů θ a ϕ.

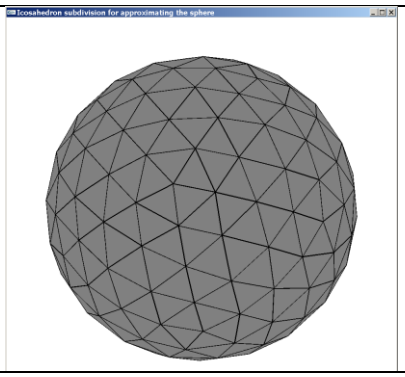
$x = r\cos\theta\sin\phi$	$y = r\ \sin\theta\sin\phi$	$z = r\cos\phi$

Daný způsob je jednoduchý, ale je několik vlastností, které je nutno mít na paměti, a to zejména:

- generované čtyřúhelníky nejsou rovinné a „rozbíjejí“ se na trojúhelníky
- generované trojúhelníky nejsou stejně velké
- na „pólech“ jsou degenerované trojúhelníky, neboť pro $\phi = \pm\frac{\pi}{2}$ je generován pouze jeden bod pro všechny úhly θ
- body sítě, tj. vrcholy trojúhelníků, jsou jednoduše indexovatelné



Povrch generovaný sférickými souřadnicemi



Povrch generovaný isometricky

Jiný postup je založen na tom, že trojúhelník lze rozdělit, a to:

- na 3 trojúhelníky vložením bodu, např. do těžiště. Postupným dělením však vznikají stále štíhlejší trojúhelníky
- na 4 trojúhelníky tak, že každou hranu rozdělíme na ½ a takto vzniklé body spojíme. V případě, že původní trojúhelník byl rovnostranný, i nově vzniklé trojúhelníky budou také rovnostranné. Toto je důležitá vlastnost vhodná v mnoha aplikacích.

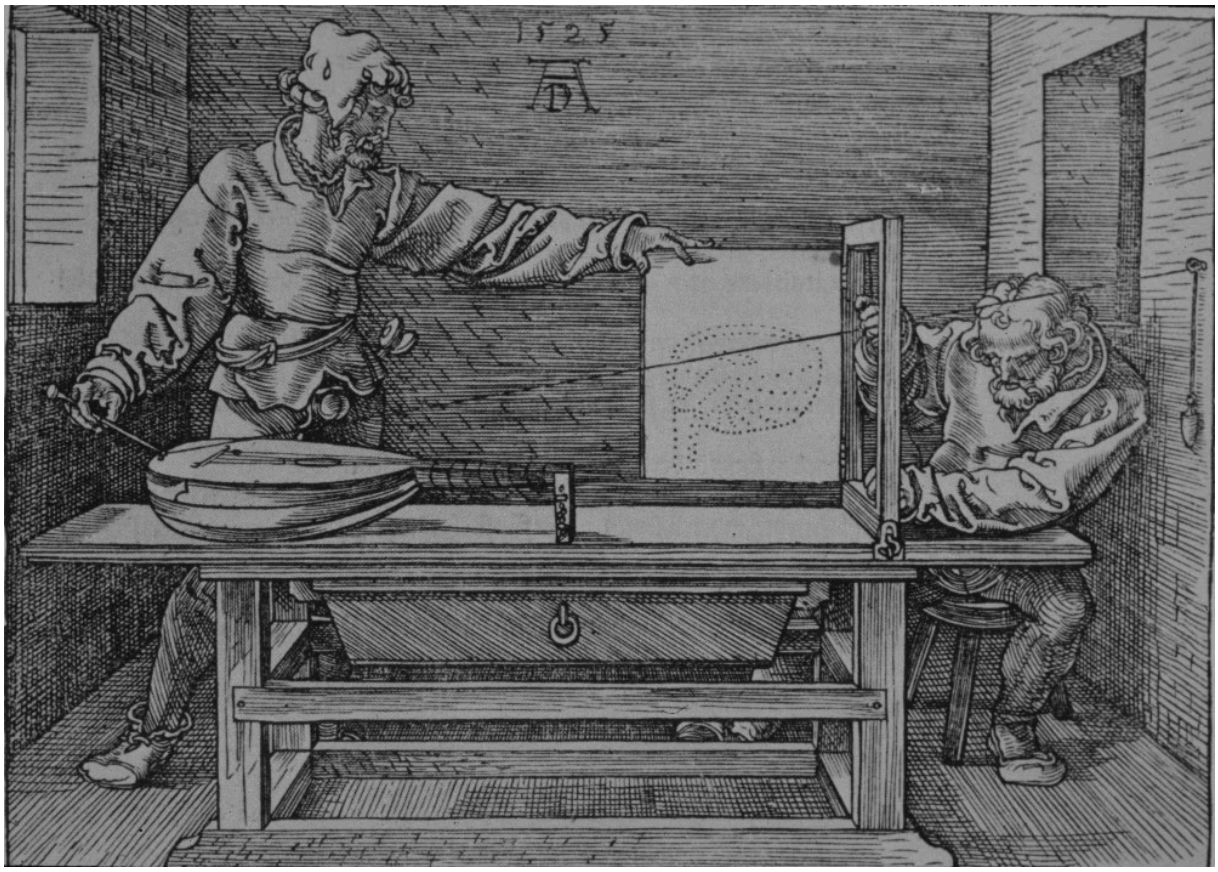
Tento způsob podporuje přirozeně hierarchické dělení, resp. adaptivní zjemňování trojúhelníkové sítě, avšak indexace je složitější.

Algoritmus

Do koule vepíšeme 2 pyramidy o hraně $r\sqrt{2}$ podstavou k sobě. Jednotlivé hrany pak rozdělíme na ½ a souřadnice takto vzniklých bodů „dotáhneme“ na povrch koule. Tímto postupem postupně zjemňujeme trojúhelníkovou síť aproximující povrch koule trojúhelníkovou sítí.

4. Projekce

Až dosud jsme se zabývali geometrickými transformacemi v E^2 a E^3. Při zobrazování třírozměrných dat obvykle dochází k promítání 3D dat na 2D průmětnu, a to i v případě virtuální reality, kdy dochází ke generování stereoskopických obrazů, které pouze vytvářejí vjem 3D prostoru. Promítání je tedy obecně proces, kdy se 3D objekty zobrazují na nějakou plochu, která nemusí být nutně rovinná. Jedním typem projekce je projekce perspektivní. Asi nejznámější kresba znázorňující perspektivní projekci pochází od Albrechta Dürera (1471-1528).



V počítačové grafice se převážně používají projekce rovinné, a ty lze klasifikovat takto:

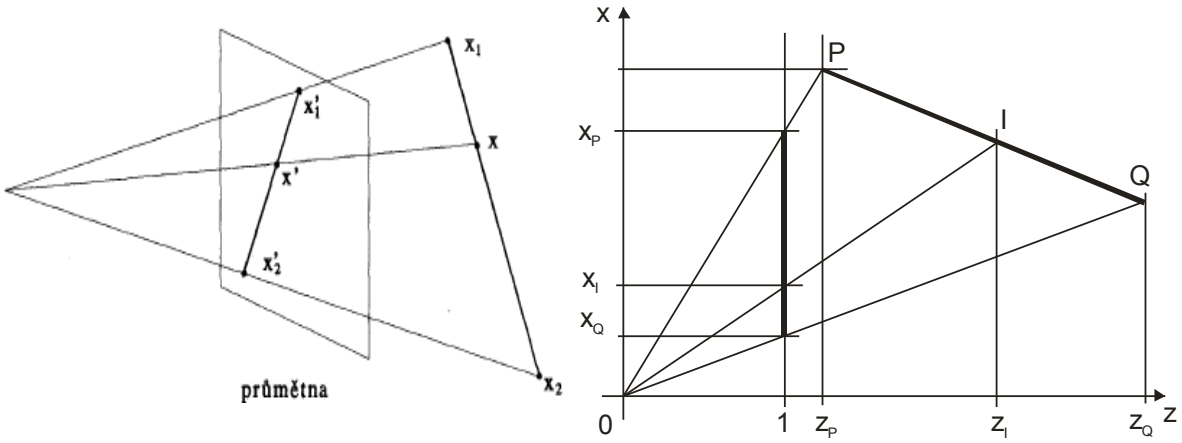
- rovinné projekce
  - perspektivní
    - jednoúběžníková
    - dvouúběžníková
    - tříúběžníková
  - paralelní
    - pravoúhlá
      - isometrická
      - dimetrická
      - trimetrická
    - kosoúhlá
      - kavalier
      - kabinet

přičemž paralelní projekce je vlastně speciálním případem projekce paralelní, kdy pozorovatel je v nekonečnu. Paralelní projekce se především používá v technické praxi, zatímco víceúběžníkové perspektivy se používají hlavně v architektuře.

4.1.Perspektivní projekce

Perspektivní projekce reprezentuje velmi dobře vjem prostoru člověkem. Je známo, že paralelní přímky se pozorovateli jeví jako přímky, které se protínají. Takový bod se nazývá úběžník (vanishing point). Podle počtu takových bodů mluvíme o jednoúběžníkové, dvouúběžníkové nebo tříúběžníkové perspektivě. Úběžník si lze tedy představit jako zobrazení bodu v nekonečnu. Je tedy zřejmé, že pokud budeme zobrazovat krychli různě natočenou v prostoru, pak přímky, na kterých její hrany leží, se budou po projekci protínat maximálně ve třech bodech.

Při perspektivní projekci je nutné si uvědomit, že dochází ke ztrátě informace o vzdálenosti, neboť všechny body jsou po transformaci na průmětně.



Pokud parametricky vyjádříme úsečky v E^3 a její průmět na rovinu, tj. v E^2, pak **není pravdou**, že parametr odpovídající původnímu bodu v prostoru a parametr odpovídajícího bodu na průmětně jsou si rovny. Tedy:

$$\boldsymbol{X} = \boldsymbol{X}_1 + (\boldsymbol{X}_2 - \boldsymbol{X}_1)\, t \qquad t \in \langle 0,1 \rangle$$
$$\boldsymbol{X}'_1 = \boldsymbol{X}'_1 + (\boldsymbol{X}'_2 - \boldsymbol{X}'_1)\, \lambda \qquad \lambda \in \langle 0,1 \rangle$$

Pak obecně:

$$t \neq \lambda$$

V případě projekcí se opět budeme snažit převést operaci projekce na maticové násobení, tj. do tvaru:

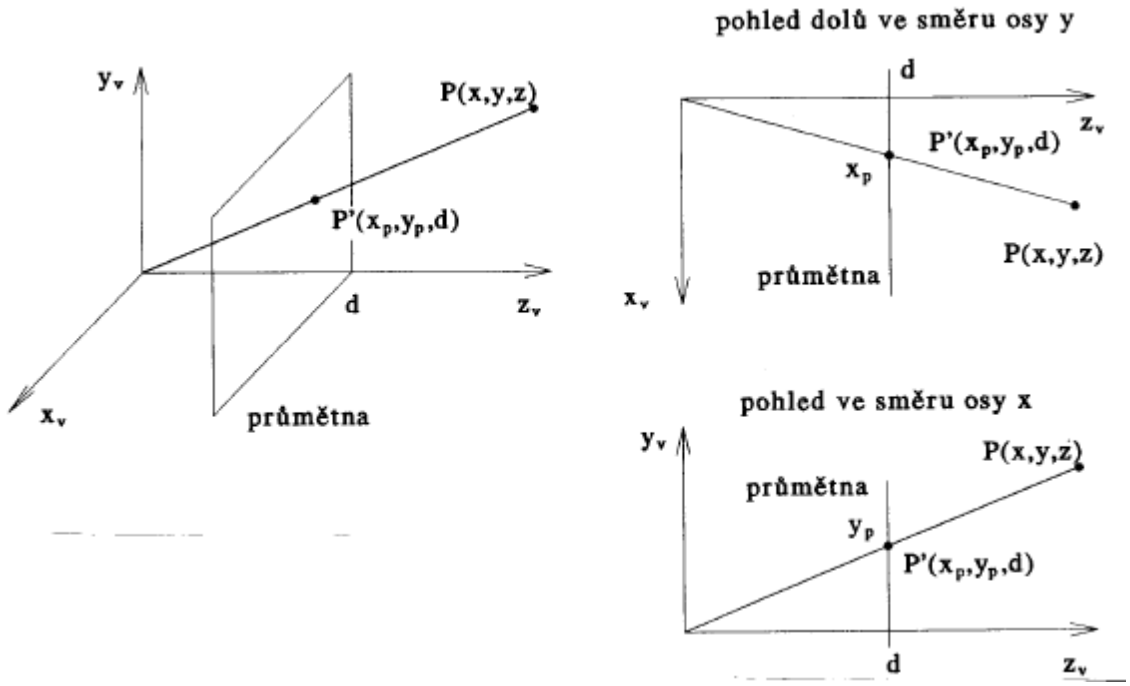
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1' \end{bmatrix} = \boldsymbol{M}_{projekce} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Je zřejmé, že takto realizovaná projekce však nesplňuje některé jisté vlastnosti, resp. požadavky, které jsou v počítačové grafice nutné, neboť:

- se ztratí informace o vzdálenosti, protože výsledné objekty jsou na průmětně, tj. výsledkem jsou vlastně dvourozměrné souřadnice bodů po projekci
- transformují se i body objektů, které jsou za pozorovatelem. Navíc, pokud geometrický element, např. trojúhelník, má jeden vrchol za pozicí pozorovatele a ostatní vrcholy před pozorovatelem, není výsledek operace korektní. Jinými slovy řečeno, je nutné odříznout objekty a jejich části, které jsou „za pozorovatelem“., tj. je nutné použít *clipping*.

4.2.Matematický aparát rovinné projekce

Souřadný systém, ve kterém je objekt definován, je pravotočivý. Takže pokud se pozorovatel dívá směrem k počátku na ose z tohoto souřadného systému, tak se vlastně dívá proti směru této osy. Souřadný systém pozorovatele, resp. kamery, je proto přirozeně *levotočivý*, viz obr.TTT.



Pro jednoduchost budeme uvažovat průmětnu kolmou na osu z a ve vzdálenosti d. Je zřejmé, že použitím podobnosti trojúhelníků dostaneme rovnice:

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

Jednoduchou úpravou pak dostáváme:

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

Dělení hodnotou z souřadnice vlastně způsobí, že pokud jsou objekty stejné velikosti, pak bližší objekt bude větší než objekt vzdálenější. Vzdálenost průmětny od počátku je vlastně jen měřítkovým faktorem.

Perspektivní projekci je pak možné popsat vztahem:

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \boldsymbol{M}_{per} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Všimněme si, že hodnota homogenní souřadnice $w = 1$.

Roznásobením dostáváme:

$$[x, y, z : z/d]^T = \left[\frac{x}{z/d}, \frac{y}{z/d}, \frac{z}{z/d} : 1\right]^T \triangleq \left(\frac{x}{z}d, \frac{y}{z}d, d\right)^T$$

kde: $\triangleq$ je operátor projektivní ekvivalence.

Zde je nutné poznamenat, že se předpokládá, že buď homogenní souřadnice bodu x je rovna hodnotě 1, tj. $w = 1$, nebo se musí následně ještě hodnota w' násobit hodnotou w. Nicméně v grafických akcelerátorech se operace projekce realizuje trochu jiným způsobem, neboť s každým bodem musí být, např. pro řešení viditelnosti, ještě k dispozici informace o vzdálenosti bodu od pozorovatele. Vzdálenost bodu od pozorovatele se ve skutečnosti řeší zavedením pseudovzdálenosti, viz kap.4.4 (Perspektiva a pseudovzdálenost).

Vzniká také otázka, zda lze určit vzdálenost od pozorovatele, tj. souřadnici z, pokud známe souřadnice původních a promítnutých koncových bodů úsečky z E^3.

Určení vzdálenosti bodu na úsečce po projekci

Pro výpočet použijeme tu souřadnici, která má největší diferenci. Nechť jde o osu x pro účely odvození vztahu. Pak platí

$$x_I = (1-\lambda)x_P + \lambda x_Q$$
$$y_I = (1-\lambda)y_P + \lambda y_Q$$

Výpočet hodnoty z_I je poměrně jednoduchý. Pokud vyjdeme z obrázku, kde pro jednoduchost dáme průmětnu do vzdálenosti 1, pak lze ukázat, že platí

$$\frac{1}{z_I} = (1-\lambda)\frac{1}{z_P} + \lambda\frac{1}{z_Q}$$

neboť

$$\frac{z_I - z_P}{x_I z_I - x_P z_P} = \frac{z_Q - z_P}{x_Q z_Q - x_P z_P}$$

Vynásobením jmenovateli a vydělením $z_P z_I z_Q$ dostáváme

$$\frac{x_Q - x_P}{z_I} = \frac{x_Q - x_I}{z_P} - \frac{x_I - x_P}{z_Q}$$

Pokud nyní použijeme substituci

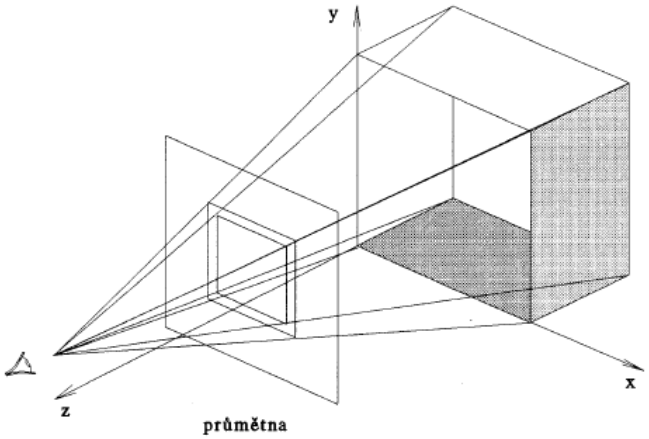
$$x_I - x_P = \lambda(x_Q - x_P) \qquad x_Q - x_I = (1-\lambda)(x_Q - x_P)$$

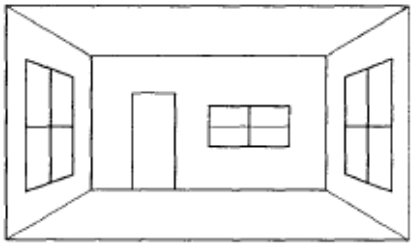
a následně vydělíme výslednou rovnici výrazem $x_Q - x_P$, pak dostaneme daný vztah.

Viz Ammeraal,L.: Programming Principles in Computer Graphics, p.146, John Wiley1992

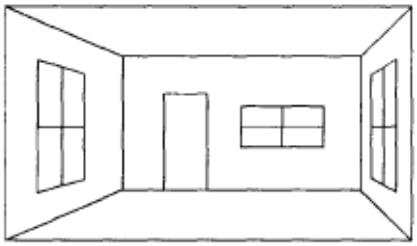
Jednoúběžníková perspektiva

Při konstrukci jednoúběžníkové perspektivního pohledu na krychli je průmětna rovnoběžná s jednou stěnou promítané krychle.





jednoúběžníková perspektiva



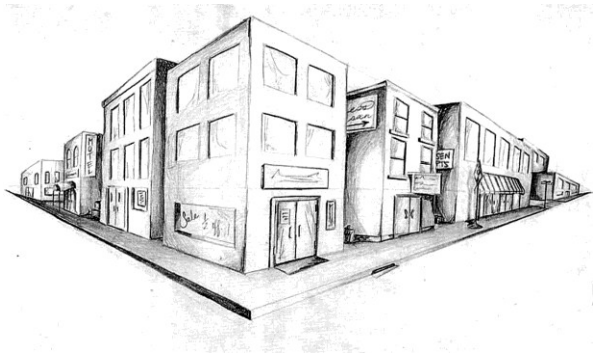
jednoúběžníková perspektiva

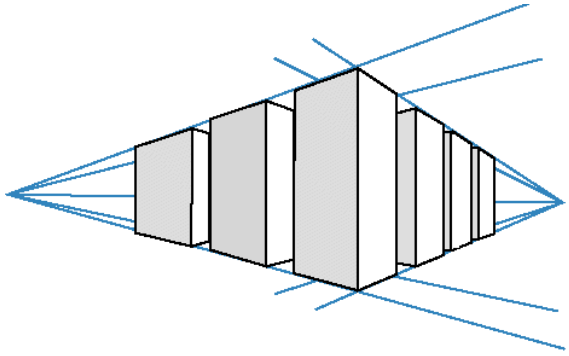
Je zřejmé, že pokud budeme pohybovat krychlí nahoru/dolů nebo vpravo/vlevo, úběžník bude mít jinou polohu. Jednoúběžníková prespektiva se používá velmi často, zejména při návrzích bytových interiérů apod.

Zkusme se nyní podívat, co se stane, pokud krychli budeme otáčet v rovině zx.

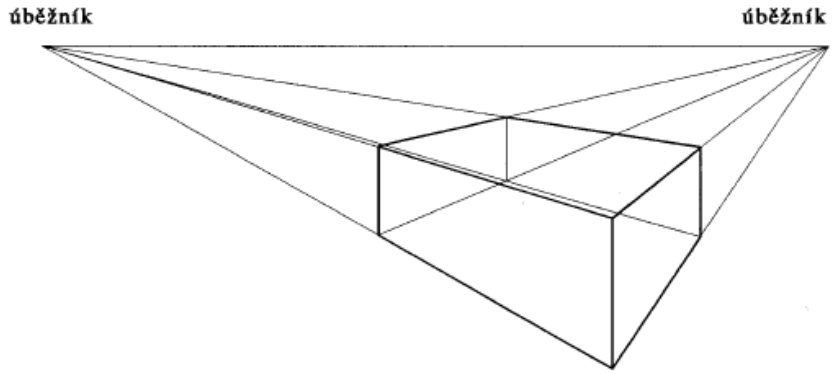
Dvouúběžníková perspektiva

Případ dvouúběžníkové perspektivy je celkem častý a v praxi takový pohled lze snadno získat tak, že směr pohledu je „diagonálně“ z opačného rohu křižovatky.





Poznamenejme, že dva úběžníky určují přímku, která je označována jako *horizont*. Jeho pozice závisí na vertikální pozici pozorovatele.

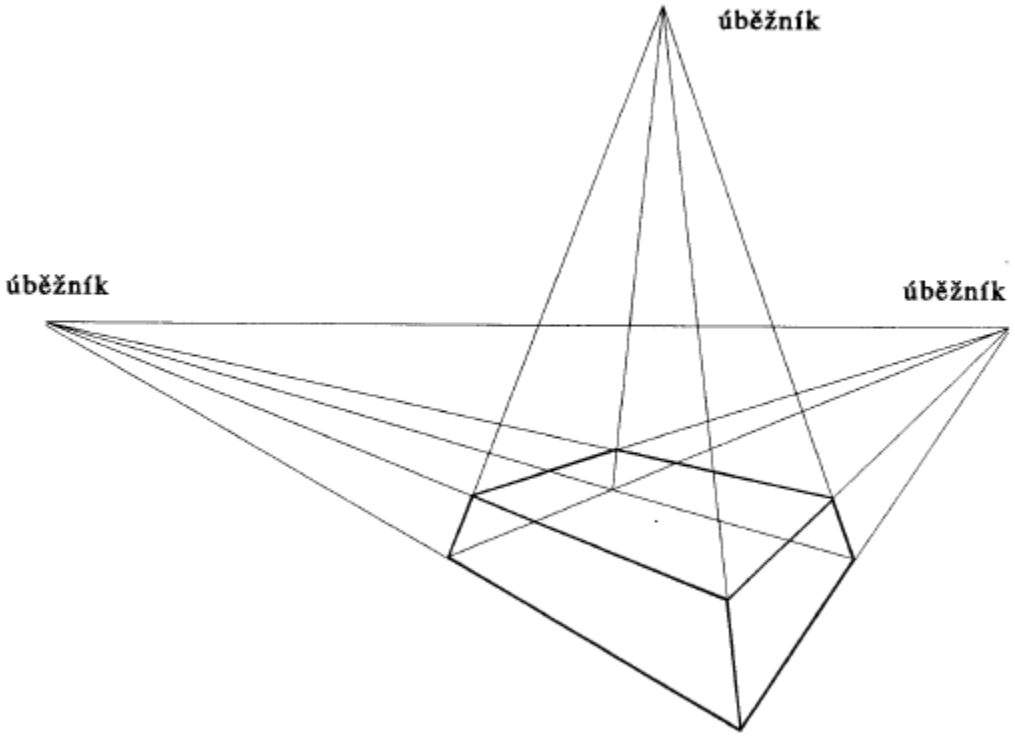


90

Obecně lze říci, že dvouúběžníkový perspektivní pohled na krychli vzniká tak, že jedna hrana krychle je rovnoběžná s jednou osou souřadného systému. Nicméně je zřejmé, že pokud budeme fotografovat vysoké objekty, např. kostelní věž, horní část bude na fotografii užší než šíře u paty věže, což je dáno tím, že se vlastně uplatní tříúběžníková perspektiva, tj. hrana krychle je po promítnutí rovnoběžná s osou y na průmětně. Pochopitelně obraz může být rotován, ale to už je operace na průmětně.

Tříúběžníková perpektiva

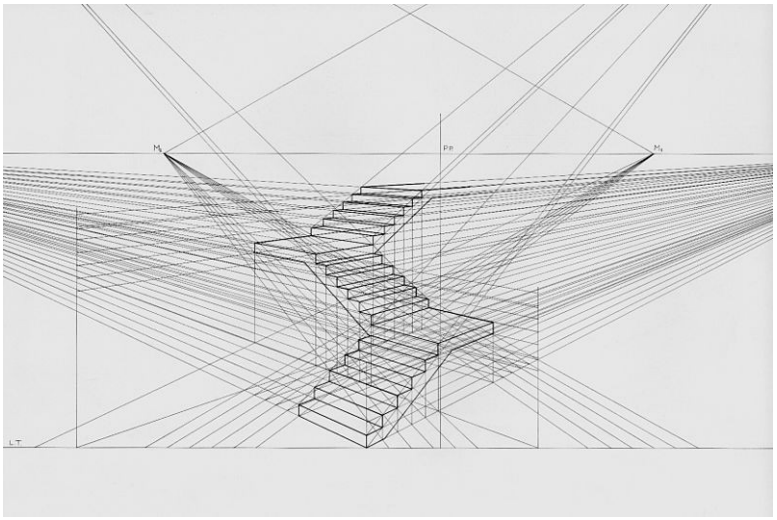
Tříúběžníková perspektiva u krychle vlastně vznikne tak, že průmětna je vůči krychli v libovolné poloze a ani hrana krychle už není rovnoběžná s plochou průmětny.



Tříúběžníková perspektiva se nepoužívá příliš často a i více úběžníkových projekcí jde v principu většinou o dvouúběžníkovou perspektivu s tím, že plochy objektu nejsou pouze na sebe kolmé apod.

Perspektiva s více úběžníky

Pokud si však představíme obecnější těleso než krychli, pak na obrázku po projekci najdeme více úběžníků.



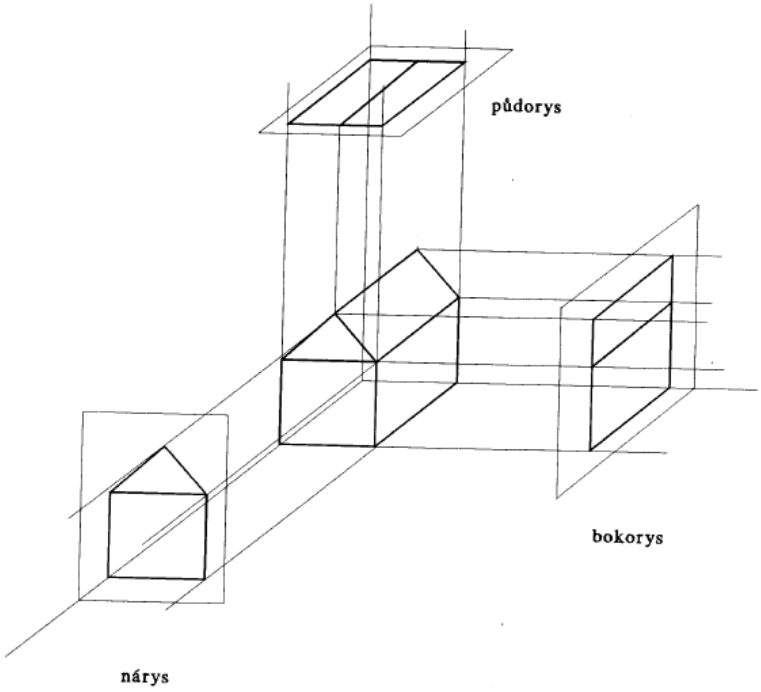
Další „pseudoúběžníky“ vznikají např. z hran schodiště apod.

4.3.Paralelní projekce

Paralelní projekce se používají především v technické praxi, a to buď projekce pravoúhlé, nebo kosoúhlé. Kosoúhlé projekce jsou oblíbené proto, že umožňují získat 3D představu.

Pravoúhlá projekce

Nejčastěji se používá pravoúhlá (orthogonal projection) projekce, a to zejména ve strojírenství a stavebnictví apod., neboť umožňuje odměřování, což perspektivní projekce z principu neumožňují.



Paralelní projekce jsou vlastně speciálním případem, kdy pozorovatel je v „nekonečnu“.

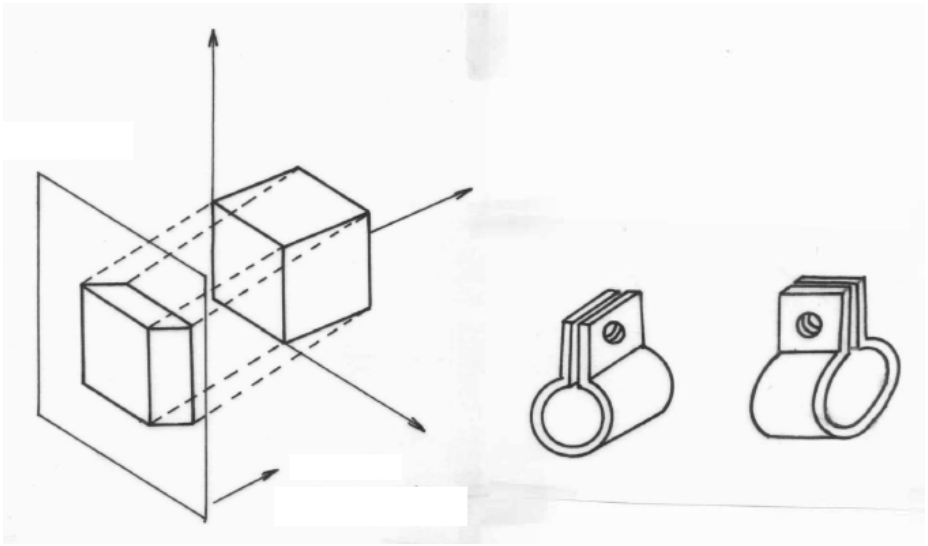
Pravoúhlou projekcí se zabývat speciálně nebudeme, neboť řešení je triviální a vlastní projekce je popsána vztahem:

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \boldsymbol{M}_{par} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & d \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Až dosud byly rovinné projekce reprezentovány pro případ $w = 1$, což není vždy možné, a navíc byla ztracena vzdálenost od pozorovatele, která je nutná pro řešení viditelnosti.

Kosoúhlé projekce

Kosoúhlé projekce se velmi často používají v praxi. Jejich výhodou je částečná možnost odměřování, např. z výkresu, hran ploch, které jsou rovnoběžné s průmětnou.



Obr.QQQQ

Obr.QQQ znázorňuje kosoúhlou projekci. Je nutné zdůraznit, že vlivem zobrazení dochází ke zkreslení tvarů, které nejsou rovnoběžné s rovinou průmětny.

4.4.Perspektiva a pseudovzdálenost

Pro řešení viditelnosti jednotlivých částí objektu se dnes převážně používá tzv. z-buffer (depth buffer), viz kap.11 (Algoritmy řešení viditelnosti). Ztráta informace o vzdálenosti od pozorovatele, tzv. o hloubce, je nepřijatelná, neboť by pak nebylo možné řešit otázku viditelnosti. Vzdálenost bodu P od pozorovatele, který je v počátku souřadného systému, je dána vztahem:

$$d = \sqrt{P_x^2 + P_y^2 + P_z^2}$$

což je evidentně nelineární vztah. Nicméně pro účely určení viditelnosti vlastně nepotřebujeme vzdálenost, ale potřebujeme informaci, co je dále a co je blíže k rozhodnutí, která část objektu je zakryta, resp. je viditelná pozorovatelem. Takže stačí vlastně funkce, jejíž hodnota monotónně roste se vzdáleností, tzv. pseudovzdálenost. Z výpočetního hlediska je rozumné, aby výpočet pseudovzdálenosti používal stejný jmenovatel, jak pro souřadnici x, tak i pro souřadnici y v perspektivní projekci s nějakým lineárním členem zahrnujícím hodnotu z souřadnice.

Upozornění: *Z důvodu konzistentnosti s většinou dostupných publikací a grafických knihoven budeme v následujícím používat pravotočivý souřadný pro výklad pseudo–vzdálenosti*.

Pro výpočet perspektivní projekce s pseudovzdáleností se používá vztah:

$$(x^*, y^*, z^*) = \left(d\frac{P_x}{-P_z}, d\frac{P_y}{-P_z}, \frac{aP_z + b}{-P_z}\right)$$

Z uvedené transformace vyplývá, že hodnoty x^*, y^* jsou stejné hodnoty jako v případě jednoduché perspektivní projekce a hodnota z^* určující pseudovzdálenost je ovlivněna dvěma parametry a a b. Tyto parametry by šlo volit prakticky libovolně, ale z praktických důvodů, kdy z^* hodnota je v pohyblivé řádové čárce, je rozumné volit hodnoty parametrů a a b tak, aby $z^* \in \langle -1,1 \rangle$. V reálné scéně jsou objekty „uzavřené“ mezi dvěma rovinami, a to rovinou přední (Near), která se označuje jako N, a rovinou zadní (Far), která se označuje jako F. Vzhledem k tomu, že vzdálenost průmětny má vliv jen na velikost, ztotožňuje se průmětna s rovinou N a tedy:

$$(x^*, y^*, z^*) = \left(N\frac{P_x}{-P_z}, N\frac{P_y}{-P_z}, \frac{aP_z + b}{-P_z}\right)$$

Z výše uvedených podmínek, tj.:

$$-1 = \frac{aN + b}{-N} \qquad 1 = \frac{aF + b}{-F}$$

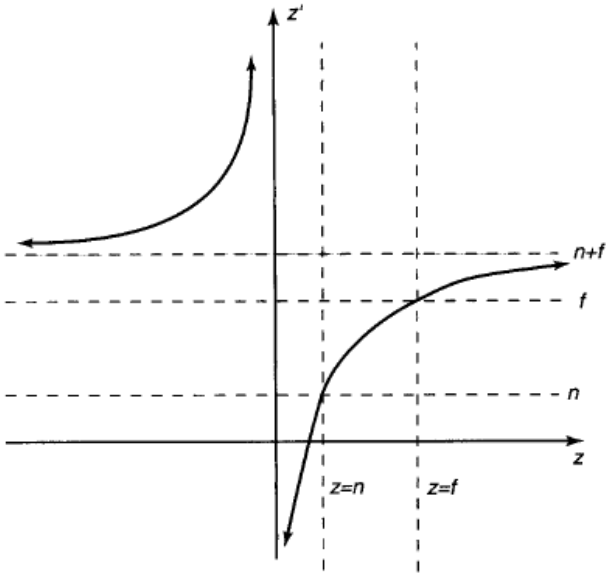
úpravou pak dostáváme:

$$N = aN + b \qquad -F = aF + b$$

Řešením pak“

$$a = -\frac{N + F}{F - N} \qquad b = -\frac{2NF}{F - N}$$

Hodnota z^* je vlastně hodnotou, která v z-bufferu slouží k rozhodnutí, zda daný pixel se má vykreslit, či nikoliv vzhledem k viditelnosti. Z grafu na obr.XXXX je zřejmé, že pro velké hodnoty z dochází k malým změnám, což může vést ke špatnému určení viditelnosti.



Obr.XXXX

Protože $N \ll F$, lze pseudovzdálenost aproximovat vztahem

$$z^* = 1 + \frac{2N}{P_z}$$

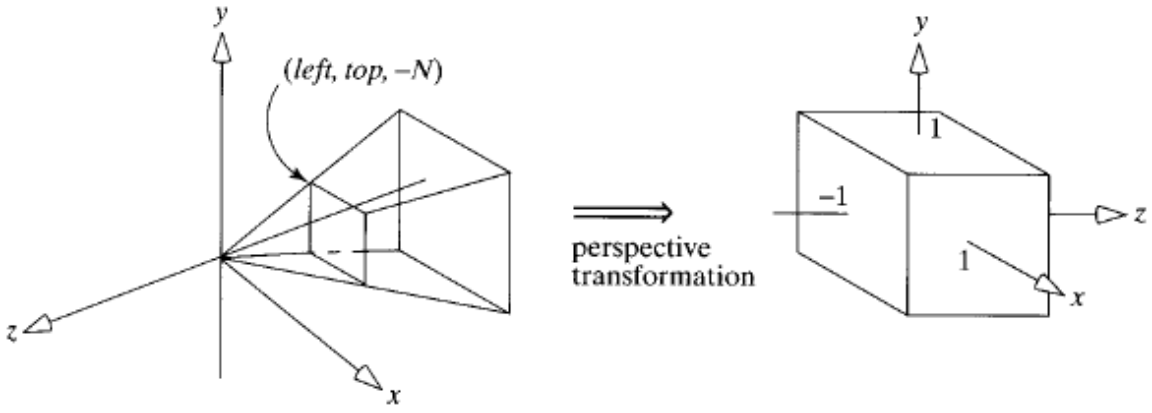
Výše uvedená formulace je vlastně pro kartézské souřadnice, nikoliv pro projektivní prostor, kde bod je reprezentován svými homogenními souřadnicemi.

Transformace pro perspektivní projekci pro body danými v homogenních souřadnicích, tj. bod $\boldsymbol{x}$ je určen jako $\boldsymbol{x} = [x, y, z: w]^T$, je určena transformací“

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} N & 0 & 0 & 0 \\ 0 & N & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} Nx \\ Ny \\ az + b \\ z \end{bmatrix}$$

Tato transformace však už nemá poslední řádek $[0,0,0: 1]^T$. Všechny geometrické transformace, které nemají poslední řádek transformační matice ve tvaru $[0,0,0: 1]^T$, už nerealizují afinní transformaci, ale transformaci perspektivní, v našem případě perspektivní projekci. Dá se ukázat, že perspektivní projekce je vlastně zřetězením operace perspektivní transformace a paralelní projekce.

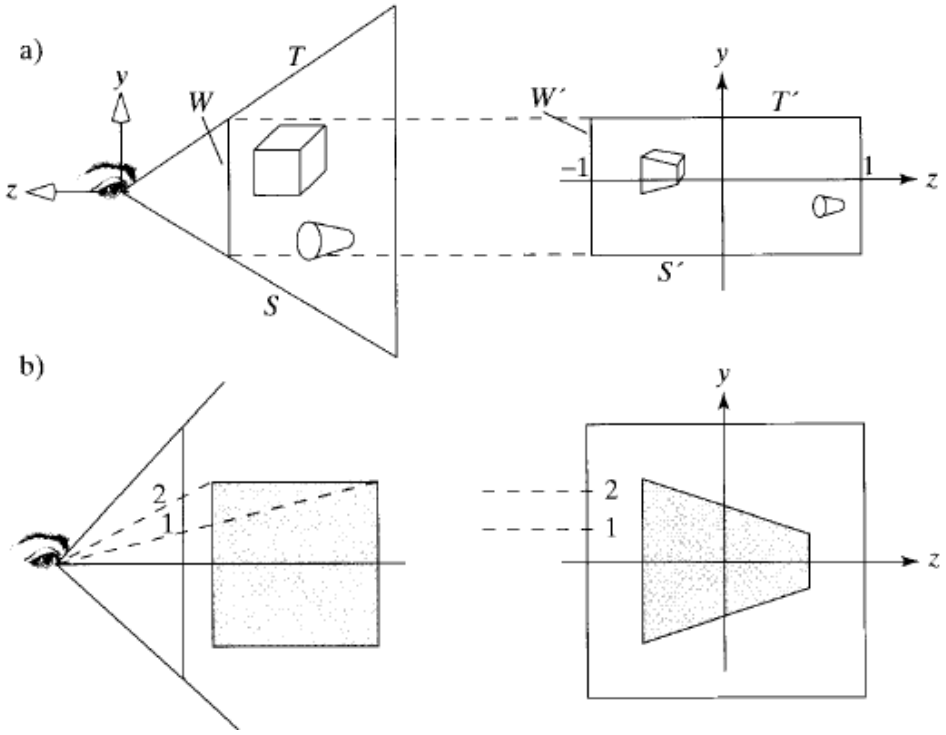
Při realizaci vlastního zobrazení se vlastně zobrazují ty objekty, resp. jejich části, které jsou v tzv. pohledové pyramidě. Ta je určena obdélníkem na průmětně a pozicí pozorovatele. Je tedy přirozenou otázkou, zda by bylo možné objekty „nějak transformovat“ tak, aby byl nahrazen vliv perspektivní projekce, a vše zpracovávat obdobně jako při paralelní projekci, viz obr.qqq



Vzhledem k tomu, že se transformují body, resp. vrcholy lineárních útvarů, jako je úsečka, trojúhelník apod., lze postupovat ve dvou krocích, a to:

- transformace objektů do tvaru, který po paralelní projekci dá stejný výsledek
- transformace do normalizovaných souřadnic NDC prostoru $\langle -1,1\rangle \times \langle -1,1\rangle \times \langle -1,1\rangle$, který se nazývá kanonický pohledový objem (canonical view volume).

Je zřejmé, že místo z souřadnice bude výsledkem opět pseudovzdálenost z^*.



obrr.TTTTTTTTTT

Nepřímým důsledkem této transformace je, že:

- v mnoha operacích se eliminuje operace násobení; jde o vlastně o násobení hodnotou ±1
- nezvyšují se výpočetní nároky, neboť transformace se „přidají“ do kumulované matice transformace

Zobrazovací okno je určenou obecně souřadnicemi $\langle left, right\rangle \times \langle bottom, top\rangle$ a výsledná transformace je pak určena maticí:

$$\begin{bmatrix} \frac{2N}{right-left} & 0 & \frac{right+left}{right-left} & 0 \\ 0 & \frac{2N}{top-bottom} & \frac{top+bottom}{top-bottom} & 0 \\ 0 & 0 & -\frac{F+N}{F-N} & -\frac{2FN}{F-N} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Matice projekce (Projection Matrix) zajišťuje perspektivní transformaci, posuv a změnu měřítka včetně převodu do kanonického pohledového objemu.

V případě použití knihovny OpenGL je tato matice je vytvořena OpenGL funkcemi:

glFrustrum(left, right, bottom, top, N,F)

nebo „uživatelsky přívětivou“ funkcí:

gluPerspective(viewAngle, aspect, N,F)

kde:

$$top = N\ tg\left(\frac{\pi}{180} viewAngle/2\right) \qquad right = top * aspect$$

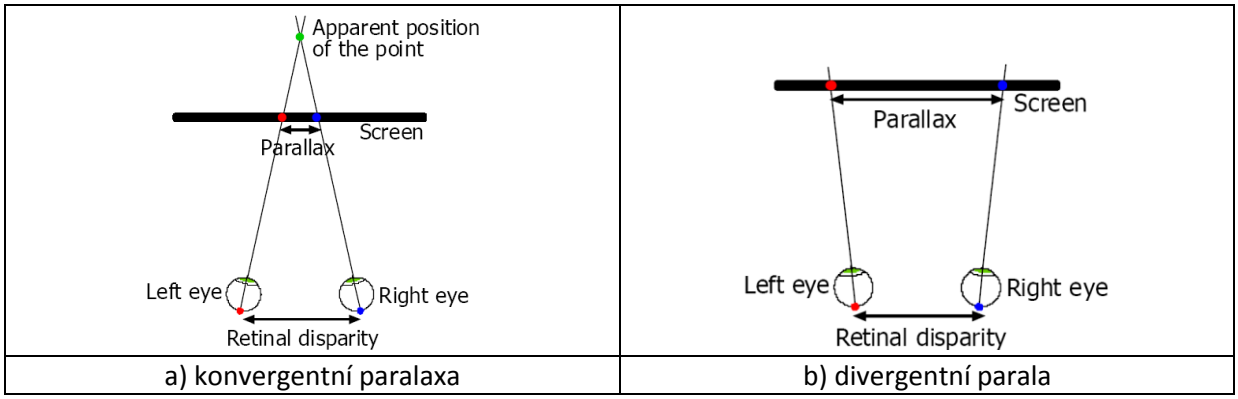
$$left = -right \qquad bottom = -top$$

Je zřejmé, že některé objekty jsou zcela nebo částečně vně pohledové pyramidy, a ty je tedy nutné vhodným způsobem odstranit. Tuto operaci zajistí operace ořezávání, tj. oříznutí vůči kanonickému (pohledovém) objemu, viz kap.5 (Metody ořezávání v E^2 a E^3 a operace s n-úhelníky).

V současné době se začíná prosazovat trh s 3D zobrazováním, které je v převážné většině realizováno stereoskopickou projekcí. Je tedy vhodné uvést alespoň základní informace.

4.5.Stereoskopická projekce

Stereoskopická projekce je v zásadě založena na perspektivní projekci, kdy se generují dva obrazy, a to jeden pro levé oko a druhý obraz pro pravé oko, přičemž se respektuje oční vzdálenost (*disparita*), tj. cca 65 [mm]. Tím se vytváří iluze 3D prostoru z hlediska percepce.



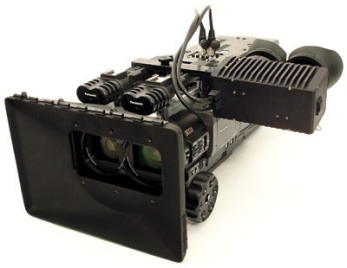
a) konvergentní paralaxa | b) divergentní parala

Je nutné zdůraznit, že zejména:

- stereoskopický obraz je „korektní“ jen pro takového pozorovatele, který je přesně v pozici, pro kterou byl obraz generován. To znamená, že např. 3D televize nebo v 3D kině mají ostatní pozorovatelé obraz nekorektní
- stereoskopický obraz musí být generován tak, aby respektoval vlastnosti projekčního zařízení, tj. skutečnou velikost promítací plochy a fyzické rozlišení. Pokud se změní velikost promítací plochy, např. 5x v každém směru, tak se disparita také 5x zvětší a oční osy nebudou konvergovat do společného bodu a budou případně divergovat, viz obr. xxx.b.

Při generování stereoskopických obrazů je celá řada dalších faktorů, které je nutné respektovat, aby výsledný vjem byl akceptovatelný. V mnoha případech výsledný 3D stereo obraz vykazuje „cartoon effect“, tj. jako kdyby scéna byla složena z 2D řezů.

Fotoaparát s 3D nástavcem



3D profesionální kamera

Rovinné projekce, které byly uvedeny výše, jsou zajisté velmi často používané, avšak také hodně limitující. Vedle rovinných projekcí se používají i projekce nerovinné, ať už v počítačové grafice, počítačovém vidění, nebo v kartografii. Tyto projekce však už nejsou realizovatelné lineárními vztahy ani při použití projektivního rozšíření kartézského souřadného systému.

4.6.Nerovinné projekce

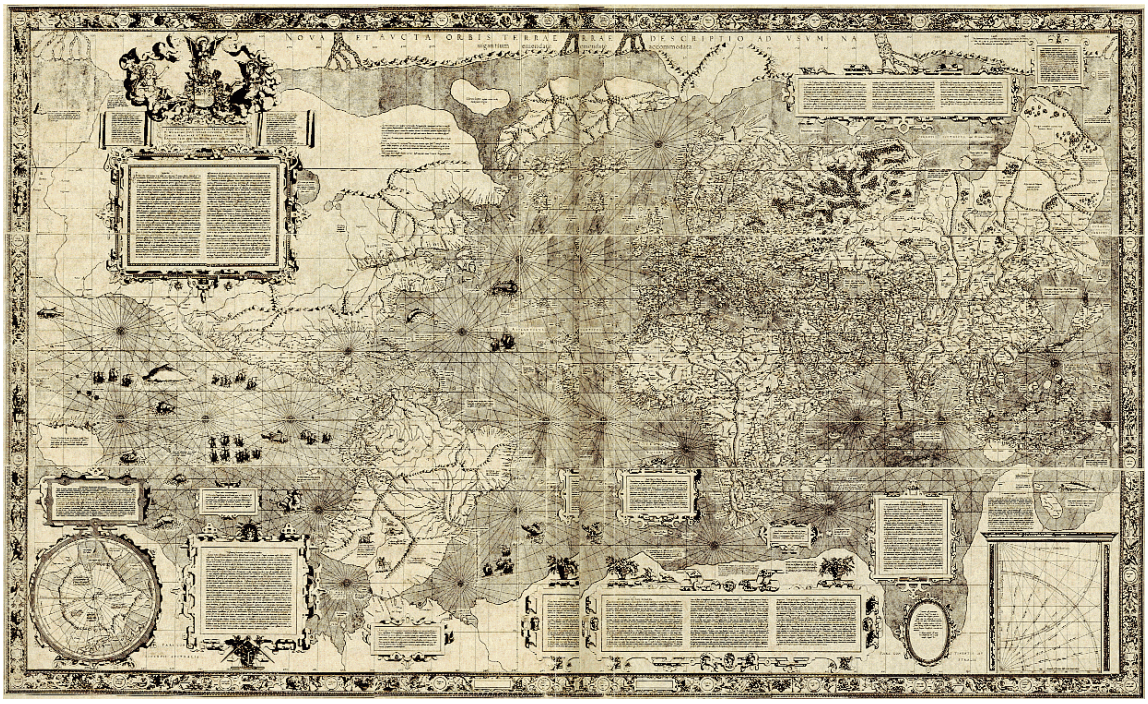
Typické nerovinné projekce, mezi které lze počítat válcovou a sférickou projekci, se běžně používají. Válcová projekce se používá pro zobrazení scény především se širokoúhlým záznamem. U fotografií s malou ohniskovou vzdáleností a širokým úhlem záběru je známo, že objekty na stranách jsou zkresleny. Takže by bylo správné obraz promítat na válcovou plochu. Sférická projekce se s výhodou používá např. ve vizualizaci astronomických dat.

Uveďme jen některé zajímavé nerovinné projekce:

- Panniniho projekce (Pannini projection), která je určena především pro perspektivní obrazy s velkým pohledovým úhlem (http://vedutismo.net/Pannini/).

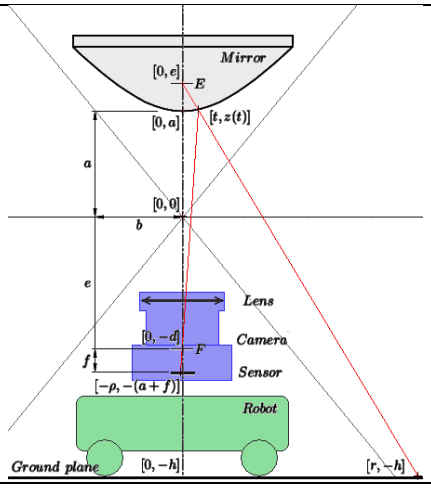
Gian Paolo Pannini, Interior of St. Peter's, Rome, c. 1754, oil on canvas, 60.75 x 77.5 in. National Gallery of Art, Washington

- Projekce Mercatorova (Mercator projection) používaná v GIS systémech a kartografii
  http://www.geo.utexas.edu/courses/420k/PDF_files/LABS/GIS/map_project.pdf

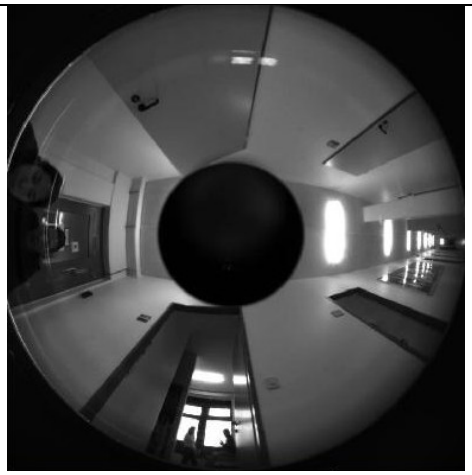


http://dawnemapy.com.pl/pages/mapy/swiat/xvi-w.php?p=40

Určitě zajímavou oblastí aplikace nelineárních projekcí je oblast počítačového vidění (computer vision). Na obr.XXX je ukázka snímání okolí robota s úhlem snímání 360°.



Viz http://jbe.wz.cz/wiki/omni/publ/2-theory



Typický výstup z kamery

Je zřejmé, že:

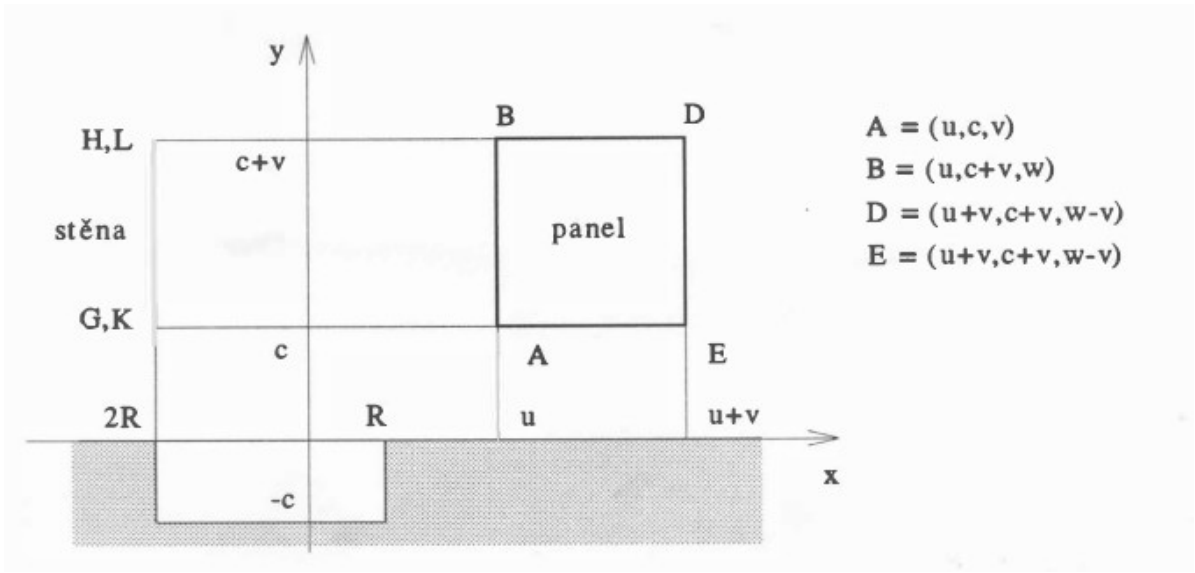
- z důvodů nelineárních transformací kamera musí mít vysoké rozlišení
- pokud jsou použity vhodně dvě takové kamery, dostáváme možnosti reprezentace prostoru

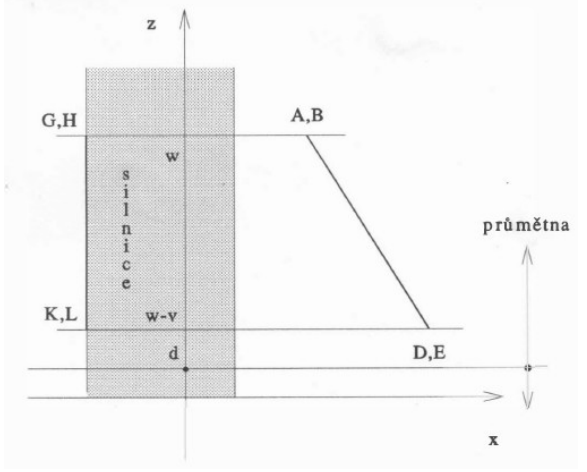
Drs, Všetečka: Objektivem počítače, SNTL 1985??

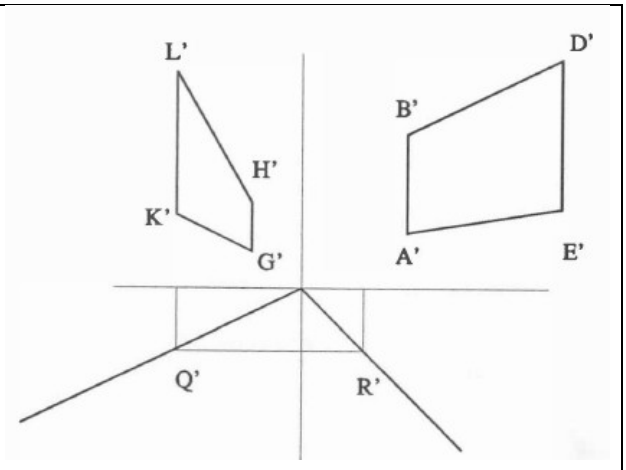
4.7.Příklad perspektivní projekce

Příklad 1

Uvažme následující úlohu, kdy chceme simulovat perspektivní pohled řidiče jedoucího na přímé silnici. Pozice počátku bude vždy v počátku souřadného systému. Jaký bude výsledný pohled řidiče?





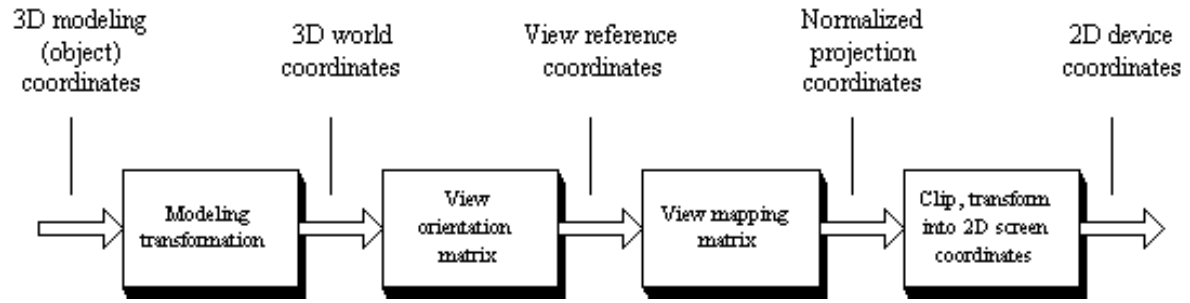


Příklad 2

Uvažme krychli $\langle -1,1\rangle \times \langle -1,1\rangle$, tj. s těžištěm v počátku a ve standardní pozici. Vrcholy označme $A, B, \ , H$. Jaký bude výsledek perspektivní projekce, pokud pozorovatel bude na ose z v pozicích $z = -2$, $z = -0.5$, $z = 0{,}5$, $z = 2$ a výsledky nakreslete včetně hran a zkontrolujte korektnost.

Co je zvláštního a proč?

Projekcí z E^3 do E^2 je vlastně zakončena část geometrických transformací s objekty.



Ty části objektu, které jsou mimo zobrazovaný prostor, se musí odstranit. Tento proces se nazývá *ořezávání*, neboli *clipping*.

5. Metody ořezávání v E^2 a E^3 a operace s n-úhelníky

Metody ořezávání jsou metodami, které jsou použitelné i v jiných oblastech. Ořezávání je „úzkým hrdlem“ grafického systému, neboť prakticky všechny elementy musí modulem ořezávání projít. V následujícím textu budou uvedeny jen základní algoritmy s tím, že mnoho dalších algoritmů lze nalézt v dostupné literatuře a z digitálních knihoven dostupných ze ZČU, resp. přístupem přes VPN, např.

- Skala,V.: Algoritmy ořezávání, ISBN 978-80-86943-22-0, Plzeň, 1990, 2011 (http://herakles.zcu.cz/~skala/EDU-PUB/Habilitace-komplet.pdf)
- Skala,V.: Algoritmy počítačové grafiky II, ISBN 978-80-86943-20-6, Plzeň, 2011 (http://herakles.zcu.cz/~skala/EDU-PUB/APG-2-OCR.pdf)
- Bui Duc Huy: Algorithms for Line Clipping and Their Complexity, PhD, Plzeň, 1999 (http://herakles.zcu.cz/~skala/MSc/Diploma_Data/DIS_1999_Bui_Duc_Huy.pdf)

Algoritmy ořezávání jsou především určeny k odstranění těch částí, které jsou mimo kreslenou plochu, nebo mimo pohledovou pyramidu, resp. k realizaci nějakého výřezu z původního celku.

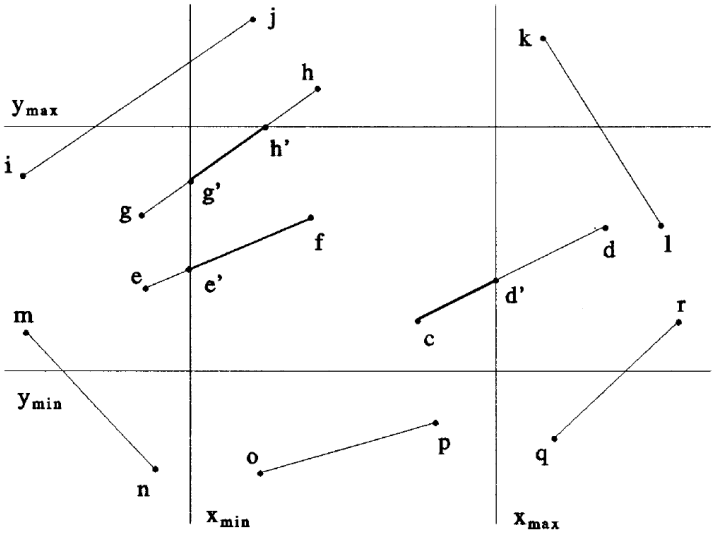
5.1.Algoritmy ořezávání v E^2

Algoritmy ořezávání v E^2 lze rozdělit podle různých hledisek. Téměř všechny algoritmy pracují v Eukleidovském souřadném systému, kromě algoritmu S-Clip pro ořezávání přímek a úseček, který ořezává přímky a úsečky v E^2 zadané v implicitní formě nebo body v projektivním prostoru. Také ořezávací okno, tj. obdélník, konvexní nebo nekonvexní n-úhelník, může být zadáno body v projektivním prostoru. Algoritmus nevyžaduje operaci dělení pro převod do Eukleidovského prostoru.

Dále uváděné algoritmy nejsou optimalizovány – při jejich aplikaci je nutné je optimalizovat.

5.1.1. Cohen-Sutherland algoritmus

Cohen-Sutherlandův algoritmus je asi nejznámějším algoritmem pro ořezávání úsečky obdélníkovým oknem. Tato úloha je zdánlivě jednoduchá, ale pokud se má realizovat operace efektivně, pak nejde o zcela triviální algoritmus. Uvažme proto základní případy, které je nutné uvažovat, viz obr.QQQ



Obr.QQQ: Základní situace pro ořezávání úsečky obdélníkem

Je zřejmé, že počet základních typických situací není triviální a přímý výpočet a testování by byly výpočetně neúnosné. Cohen a Sutherland navrhli efektivní řešení spočívající především v „chytrém“ kódování koncových bodů úsečky s následným využitím těchto kódů pro efektivní řešení úlohy.

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.TTTT: Kódování koncových bodů úsečky

Algoritmus je založen na rozdělení plochy na 9 oblastí, viz obr.TTTT, a následujících základních krocích:

- určení, do které oblasti koncové body úsečky patří
- detekce elementárních případů, kdy úsečka je zcela uvnitř nebo vně
- řešení ostatních případů *iterativně*. Maximální počet kroků je 4, ale jsou poměrně složité

Určení kódu pro bod:

```
function CODE(x, y);
byte c;
{
  if x < x_min then c := 1 else if x > x_max then c := [0010] else c := [0000];
  if y < y_min then c := c lor [0100] else if y > y_max then c := c lor [1000];
  CODE := c
}
```

Tímto postupem dostaneme kódová slova c_1 a c_2 pro oba koncové body úsečky $\boldsymbol{x}_1$ a $\boldsymbol{x}_2$.

Je zřejmé, že elementárními případy jsou:

- oba koncové body jsou uvnitř okna, tj. $(c_1 \textbf{ lor } c_2) = 0$, kde **lor** je operace **or** po bitech
- oba koncové body jsou nad, pod, vlevo nebo vpravo, tj. $(c_1 \textbf{ land } c_2) \neq 0$, kde **land** je operace **and** po bitech

Pokud nejde o jeden z těchto triviálních případů, je nutné počítat průsečíky s jednotlivými hranami okna a testovat, zda nový průsečík je vně okna, či nikoliv. Jednotlivé průsečíky se pak učí takto:

$(x_{min}, k(x_{min} - x_1) + y_1)$	Je-li bod vlevo
$(x_{max}, k(x_1 - x_{max}) + y_1)$	Je-li bod vpravo
$(q(y_1 - y_{max}) + x_1 + y_{max})$	Je-li bod nad
$(q(y_{min} - y_1) + x_1, x_{min})$	Je-li bod pod

kde:

$$k = \frac{y_2 - y_1}{x_2 - x_1} \quad \& \quad x_2 - x_1 \neq 0 \qquad\qquad k = \frac{x_2 - x_1}{y_2 - y_1} \quad \& \quad y_2 - y_1 \neq 0$$

Celý postup lze schematicky popsat takto:

c_1 := CODE (x_1, y_1); c_2 := CODE (x_2, y_2);
if $(c_1 \textbf{ lor } c_2) = 0$ **then** { LINE (x_1, y_1, x_2, y_2); EXIT } # úsečka je zcela uvnitř #
if $(c_1 \textbf{ land } c_2) \neq 0$ **then** EXIT # úsečka je zcela vně #
repeat
if $c_1 = 0$ **then** $c := c_2$ **else** $c := c_1$;
if $(c \textbf{ land } 1) \neq 0$ **then** # bod je vlevo #
{ $y := y_1 + (x_{min} - x_1)(y_2 - y_1)/(x_2 - x_1)$; $x := x_{min}$ }
else if $(c \textbf{ land } 2) \neq 0$ **then** # bod je pravo #
{ $y := y_1 + (x_1 - x_{max})(y_2 - y_1)/(x_2 - x_1)$; $x := x_{max}$ }
else if $(c \textbf{ land } 4) \neq 0$ **then** # bod je pod #
{ $x := x_1 + (y_{min} - y_1)(x_2 - x_1)/(y_2 - y_1)$; $y := y_{min}$ }
else if $(c \textbf{ land } 8) \neq 0$ **then** # bod je nad #
{ $x := x_1 + (y_1 - y_{max})(x_2 - x_1)/(y_2 - y_1)$; $y := y_{max}$ };
if $c = c_1$ **then** { $x_1 := x$; $y_1 := y$; CODE (x_1, y_1) }
else { $x_2 := x$; $y_2 := y$; CODE (x_2, y_2) };
if $(c_1 \textbf{ land } c_2) \neq 0$ **then** EXIT # poslední část úsečky je zcela vně #
until $(c_1 \textbf{ lor } c_2) = 0$;
LINE (x_1, y_1, x_2, y_2);
EXIT

Algoritmus TTTTTTTTTTTT

Z algoritmu je zřejmé, že algoritmus přes zdánlivou jednoduchost řešeného problému má poměrně vysokou výpočetní složitost. Je možné jej částečně optimalizovat, např. předpočítáním hodnot k, q, neboť odpovídající výrazy se někdy používají $2x$.

Otázka - V jakém případě provede algoritmus celkem 4 cykly??

Zde je nutné upozornit:

- na možné problémy s přesností výpočtu, které mohou vzniknout při výpočtu průsečíku a stanovení odpovídajícího kódového slova
- operace porovnání je nejdelší numerickou operací hned po operaci dělení v pohyblivé řádové čárce

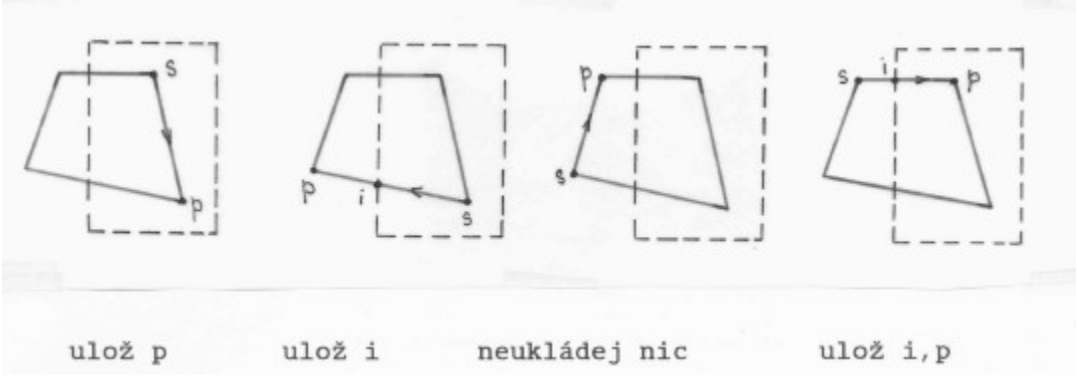
Úloha

Porovnejte Cohen-Sutherlandův (CS) algoritmus s následujícími algoritmy, které nahradí poslední část CS algoritmu:

- pro netriviální situace se spočtou 4 průsečíky s hranami okna přímo a hodnoty parametru t se porovnávají, protože obdélník je konvexní n-úhelník, tak i parametry t musejí být cyklicky uspořádány
- porovnejte CS algoritmus, navržený algoritmus bez použití vláken a paralelizovaný algoritmus. Lze např. podle hodnot s_x a s_y rozhodnout, které hrany mohou být protnuty?

Nápověda: Pokud jsou hrany obdélníka indexovány po směru nebo proti směru hodinových, pak i hodnoty parametrů t mají stejnou indexaci a musí mít cyklicky rostoucí nebo klesající hodnotu.

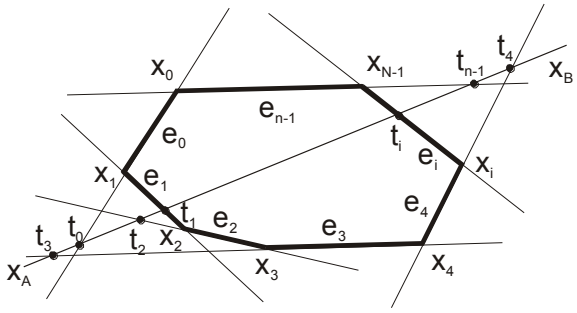
5.1.2. Sutherland Hodgman algoritmus - doplnit



POPIS

5.1.3. Cyrus Beck algoritmus

Cyrus-Beck algoritmus (CB) je algoritmus, který umožňuje ořezávání přímky i úsečky konvexním n-úhelníkem a je snadno modifikovatelný i pro případ přímky, resp. úsečky v prostoru. Algoritmus je složitosti $O(n)$, kde n je počet hran n-úhelníka.



Průsečíky počítané CB algoritmem

Přímka je zadána dvěma body $\boldsymbol{X}_A$ a $\boldsymbol{X}_B$ a tedy v parametrické formě:

$$\boldsymbol{X}(t) = \boldsymbol{X}_A + (\boldsymbol{X}_B - \boldsymbol{X}_A)t = \boldsymbol{X}_A + \boldsymbol{S}\,t$$

Předpokládejme hypotetického pozorovatele, který je v bodě $t = -\infty$ a dívá se ve směru vektoru $\boldsymbol{S}$. Algoritmus je založen na:

- rozdělení hran n-úhelníka do dvou skupin, a to: na hrany, které pozorovatel vidí, a na hrany, které nevidí
- výpočet průsečíků přímky s jednotlivými hranami
  - u hran, které vidí, hledá maximální hodnotu t_{min} parametru u příslušných průsečíků
  - u hran, které nevidí, hledá minimální hodnotu t_{max} parametru u příslušných průsečíků
- pokud $t_{max} < t_{min}$, tj. interval $\langle t_{min}, t_{max}\rangle = \emptyset$, pak daná přímka konvexní n-úhelník neprotíná

V případě ořezávání úsečky je nutné spočítat, zda vypočtený interval má neprázdný průnik s intervalem $\langle 0{,}1\rangle$. Pokud $\langle 0{,}1\rangle \cap \langle t_{min}, t_{max}\rangle = \emptyset$, pak úsečka nemá s n-úhelníkem průsečík. V opačném případě se body průsečíku dopočtou z parametrické rovnice přímky.

```
procedure CB;   # input: x_A , x_B #
{
    s := x_B - x_A;
    t_min:= -∞;    t_max:= ∞;     # t_min:= 0;    t_max:= 1; in case of a line segment #
    for i:=0 to N-1 do
    {
        q := n_i^T s;
        if q < 0 then
        {       # the i-th edge can be seen from infinity in the negative direction of –s #
            t:= - (n_i^T x_A + c_i)/q; t_min:=max (t, t_min);
        } else
            if q > 0 then
            { #the i-th edge can be seen from #
                  #infinity in the direction of s #
                t:= - (n_i^T x_A + c_i)/q; t_max:= min (t, t_max);
            } else solve a special case  n_i^T s = 0
    };
    if  t_min < t_max then
    {   x_A :=  x_A + s t_min ;        x_B :=  x_A + s t_max ;     output (x_A , x_B ) }
} # CB algoritmus #
```

Princip Cyrus-Beck algoritmu

5.1.4. Algoritmus se složitostí *O(lg n)*

Cyrus-beck algoritmus pro řezávání přímky nebo úsečky konvexním n-úhelníkem využívá konvexitu pouze k tomu, aby vyhodnocoval pouze 2 možné průsečíky. V případě E^2 jsou vrcholy uspořádané ve směru (clockwise) nebo v protisměru (anticlockwise) hodinových ručiček. Princip duality v E^2 říká, že bod je duální přímce a naopak, průnik je duální sjednocení a naopak. Takže problém, zda přímka protíná konvexní n-úhelník v E^2 je duální úloze, zda je bod uvnitř konvexního n-úhelníka. Tento problém je složitosti $O(lg\ n)$. Vlastní algoritmus ořezávání je založen na půlení intervalu indexů vrcholů daného konvexního n-úhelníka. Vzhledem k větší složitosti algoritmu, vlastní algoritmus není zde uveden. Podrobný popis viz:

> Skala, V.: *O*(lg *N*) Line Clipping Algorithm in E2, Computers & Graphics, Pergamon Press, Vol.18, No.4, 1994

Je nutné zdůraznit, že algoritmus je pro $n > 3$ rychlejší než algoritmus Cyrus-Beck. Je nutné zdůraznit, že pokud konvexní n-úhelník má např. 1024 vrcholů, pak Cyrus-Beck vyžaduje 1024 kroků, zatímco $O(lg\ n)$ algoritmus vyžaduje pouze 10 kroků, což je podstatný rozdíl.

Na tomto případě je názorná ukázka, jak aplikace principu duality může vést k návrhu nových efektivních algoritmů. Poznamenejme, že aplikací principu duality se však výpočetní složitost daného problému nemění.

5.1.5. Algoritmus Smart-Clip (S-Clip)

Až dosud jsme se zabývali algoritmy, které byly navrženy pro řešení problému v Eukleidovském prostoru. Nicméně souřadnice bodů jsou reprezentovány v homogenních souřadnicích a po aplikaci geometrických transformací nebo jiných výpočtů hodnota homogenní souřadnice je obecně $w \neq 1$. To znamená, že pro účely ořezávání by bylo nutné provést dělení souřadnicí w, tedy 3 operace dělení na 1 bod. Při počtu zpracovávaných bodů $10^5 - 10^{10}$ by tato konverze vyžadovala neúměrný výpočetní výkon.

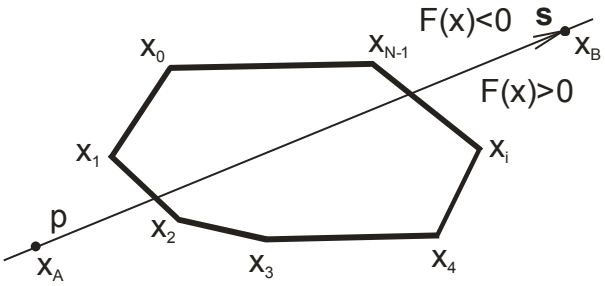
Algoritmus S-Clip je algoritmus, která umožňuje ořezávání přímky nebo úsečky obdélníkovým oknem, konvexním n-úhelníkem se složitostí $O(n)$ a nekonvexním n-úhelníkem se složitostí $O(n + m\,lg\,m)$, neboť je nutná operace řazení nad m nalezenými průsečíky.

Algoritmus je založen na vyhodnocení pozice vrcholů n-úhelníka vzhledem k přímce dané dvěma body, tj. ve které polorovině body leží. Na základě této klasifikace se určí indexy hran, které přímka protíná, a následně se spočtou průsečíky těchto hran.

Předpokládejme, že je dán konvexní n-úhelník a přímka p rovnicí $F(\boldsymbol{x}) = ax + by + c$. Přímka p rozdělí prostor na dvě poloroviny, tj. pro $F(\boldsymbol{x}) > 0$ a pro $F(\boldsymbol{x}) \leq 0$. Hodnota funkce $F(\boldsymbol{x})$ může být vyhodnocena pro každý vrchol n-úhelníka. Pro i-tý vrchol pak dostáváme i-tý bit kódového slova $\boldsymbol{c}$ jako:

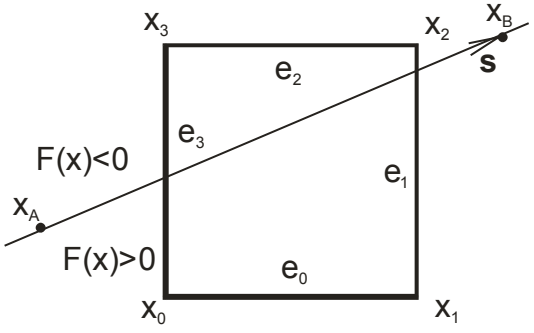
$c_i = \begin{cases} 1 & \text{pokud} \quad F(\boldsymbol{x}_i) > 0 \\ 0 & \text{jinak} \end{cases}$	pro $i = 0, \ldots, n-1$

To znamená, že každý vrchol je ohodnocen, zda leží „napravo“ nebo „nalevo“ od přímky p.



Obr.xxx: Ohodnocení vrcholů n-úhelníka vůči přímce p

V následujícím se omezíme na obdélníkové okno, bez újmy na principu, či na obecnost algoritmu.



Obr.XXXX: Ořezávání obdélníkovým oknem

V případě obdélníkového okna je kódové slovo $\boldsymbol{c}$ určeno:

$$\boldsymbol{c} = [c_3, c_2, c_1, c_0]^T$$

a kód $\boldsymbol{c}$ nyní určuje *jednoznačně* pozici přímky vůči vrcholům ořezávacího n-úhelníka. Pro jednotlivé hodnoty kódových slov $\boldsymbol{c}$ lze explicitně specifikovat hrany, které budou přímkou p protnuty, Tab.QQ.

C	c_3	c_2	c_1	c_0	**TAB1**	**TAB2**	**MASK**
0	0	0	0	0	None	None	None
1	0	0	0	1	0	3	0100
2	0	0	1	0	0	1	0100
3	0	0	1	1	1	3	0010
4	0	1	0	0	1	2	0010
5	0	1	0	1	N/A	N/A	N / A
6	0	1	1	0	0	2	0100
7	0	1	1	1	2	3	1000
8	1	0	0	0	2	3	1000
9	1	0	0	1	0	2	0100
10	1	0	1	0	N/A	N/A	N / A
11	1	0	1	1	1	2	0010
12	1	1	0	0	1	3	0010
13	1	1	0	1	0	1	0100
14	1	1	1	0	0	3	0100
15	1	1	1	1	None	None	None

kde:

- N/A (non-applicable case) označuje případy, které nemohou principiálně nastat, neboť kombinace $\boldsymbol{c} = [0,1,0,1]^T$ by značila, že přímka p má 4 průsečíky s konvexním n-úhelníkem, což pro konvexní n-úhelník není možné.
- None označuje případ, kdy přímka p neprotíná daný n-úhelník
- význam sloupce MASK bude vysvětlen později – použije se pro rozšíření algoritmu S-Clip pro ořezávání úsečky, tj. algoritmu S-Clip_Segment.

Indexy hran protnutých přímkou p jsou uloženy v TAB1 a TAB2 a lze nahlédnout, že tyto hodnoty jsou „symetrické“, což je dáno implicitním popisem přímky p, kdy $F(\boldsymbol{x}) = 0$ a $-F(\boldsymbol{x}) = 0$ definují stejnou přímku p. Proto je také nepodstatná orientace vrcholů okna, tj. ve směru nebo v protisměru hodinových ručiček.

Výše uvedenou tabulku lze vygenerovat automaticky pro konvexní n-úhelník, neboť pokud $\boldsymbol{c}_i \neq \boldsymbol{c}_{i+1}$, pak hrana $\boldsymbol{x}_i\boldsymbol{x}_{i+1}$ bude přímkou p protnuta. Délka tabulky je pak 2^n, kde n je počet vrcholů n-úhelníka.

procedure S-CLIP_Line ($\boldsymbol{x}_A$, $\boldsymbol{x}_B$);
# **input**: $\boldsymbol{x}_A = [x_A, y_A : w_A]^T$ $\boldsymbol{x}_B = [x_B, y_B : w_B]^T$ $\boldsymbol{p} = [a, b : c]^T$ $n = 4$ #
{
#1# $\boldsymbol{p} = \boldsymbol{x}_A \times \boldsymbol{x}_B$; # $p: ax + by + cw = 0$ #
#2# **for** $k := 0$ **to** $n - 1$ **do** # cyklus může být realizován paralelně #
#3# **if** $\boldsymbol{p}^T \boldsymbol{x}_k \geq 0$ **then** $c_k = 1$ **else** $c_k = 0$; # $\boldsymbol{x}_k = [x_k, y_k : w_k]^T$ #
#4# **if** $\boldsymbol{c} = [0000]^T$ **or** $\boldsymbol{c} = [1111]^T$ **then EXIT**; # numerický test **if** c=0 **or** c=15 **then EXIT** #
#5# i := TAB1[$\boldsymbol{c}$]; j := TAB2[$\boldsymbol{c}$];
#6# $\boldsymbol{x}_A := \boldsymbol{p} \times \boldsymbol{e}_i$; $\boldsymbol{x}_B := \boldsymbol{p} \times \boldsymbol{e}_j$; # $\boldsymbol{e}_i$ je i-tá hrana n-úhelníka #
#7# **DRAW** ($\boldsymbol{x}_A$, $\boldsymbol{x}_B$)
} # S-CLIP_Line#

Alg. Clip-LS

Z výše uvedeného algoritmu je zřejmé, že body definující přímku, resp. úsečku a vrcholy n-úhelníka jsou v homogenních souřadnicích, přičemž homogenní složky souřadnic bodů mohou být $w \neq 1$, což je důsledkem použití implicitní formulace. Není tedy zapotřebí použít operaci dělení pro převod do Eukleidovského prostoru.

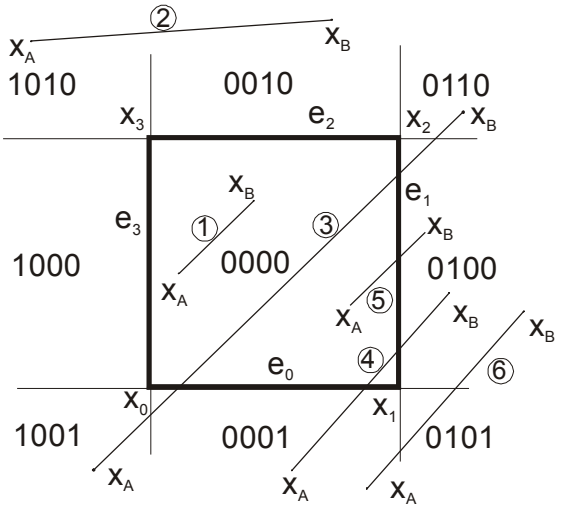
Výše uvedený algoritmus je jednoduchý, elegantní, a pokud implementován, např. na GPU architektuře, pak vektorový a skalární součin je hardwarovou instrukcí. Otázkou je, zda algoritmus nelze dále optimalizovat.

Optimalizace algoritmu pro okno $\langle -\mathbf{1}, \mathbf{1}\rangle \times \langle -\mathbf{1}, \mathbf{1}\rangle$

U geometrických transformací a projekcí byl zaveden princip NDC prostoru. Analýzou algoritmu výše uvedeného je zřejmé, že vektorové součiny v přiřazeních $\boldsymbol{x}_A := \boldsymbol{p} \times \boldsymbol{e}_i$ a $\boldsymbol{x}_B := \boldsymbol{p} \times \boldsymbol{e}_j$ mohou být podstatně zjednodušeny, neboť přímky hran obdélníka mohou být vyjádřeny tak, že jejich koeficienty nabývají hodnot $\{-1,0,1\}$. Není tedy zapotřebí násobení pro realizaci vektorového součinu, tj. lze ušetřit až 6 násobení na každou přímku.

Modifikace algoritmu S-Clip pro úsečky

Výše uvedený algoritmus S-Clip je určen pro ořezávání přímek. Počítačová grafika však většinou zpracovává „konečné“ elementy, např. úsečky, trojúhelníky apod. Je tedy nutné se podívat, jak algoritmus S-Clip lze modifikovat pro případ ořezávání úseček. Obdobně jako Cohen-Sutherland algoritmus použijeme kódování koncových bodů ořezávané úsečky.



Obr. Klasifikace koncových bodů ořezávané úsečky

Modifikace algoritmu S-Clip pro ořezávání úsečky je založena na jednoduchém principu. Dříve uvedený algoritmus S-Clip určuje přímo, které dvě hrany jsou protnuty přímkou, na níž daná úsečka leží. Pokud se eliminují elementární případy, kdy úsečka je zcela uvnitř nebo vně, pak pokud koncový bod leží uvnitř, je nutné vybrat správnou hranu již dříve z dvou vybraných hran, se kterou se vlastní průsečík spočte. Např. pokud přímka protíná hrany e_1 a e_3 a $\boldsymbol{c}$, je kód ohodnocení vrcholu ořezávacího okna, pak příkazy:

if $\mathbf{c}_B$ **land** MASK[**c**] $\neq$ **0**
then intersection p & e_1
else intersection p & e_3

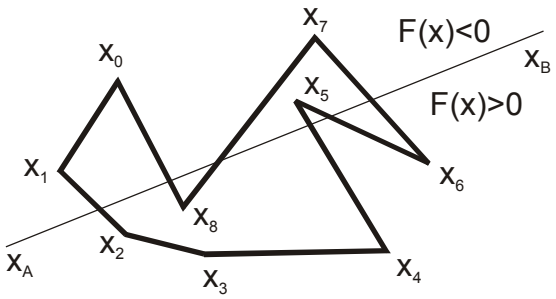
určí odpovídající průsečík. Tabulka MASK byla již uvedena dříve, viz Tab.xx.

```
1   function CODE (x);
2   {
3       c:= [0000];
4       if x < x_min then c:= [1000]
5                   else if  x > x_max then c:= [0100];
6       if y < y_min then c:= c lor [1001]
7                   else if  y > y_max then c:= c lor [0010];
8       CODE := c
9   } #CODE#;
10
11  procedure S-Clip_Segment (x_A , x_B);   # input: x_A , x_B #
12  # C_LS - Clipping Line Segment  x_A , x_B – can be in homogeneous coordinates #
13  # the EXIT statement ends the procedure #
14  # land / lor – bitwise  operations and / or #
15  { # end-points classifications#
16      c_A := CODE (x_A);    c_B := CODE (x_B);
17      # detection of trivial cases #
18      # case 1 – line segment inside #
19      if (c_A lor c_B) = 0 then
20          {  output (x_A; x_B ); EXIT }
21      # case 2 – line segment outside – just EXIT#
22      if (c_A land c_B) ≠ 0 then EXIT;
23      # all trivial cases solved #
24      # compute coefficients of the line p #
25      p := x_A x x_B;  # ax+by+c = 0; p = [a,b,c]^T #
26      for k:=0 to 3 do # x_k=[x_k,y_k,1]^T #
27          if p^T x_k ≥ 0 then c_k:=1 else c_k:=0;
28      # c = [ c_3, c_2, c_1, c_0 ]^T #
29      if  c = [0000]^T or c = [1111]^T  then EXIT;
30      # the line segment lies outside of the window - it distinguishes also the cases 4 and 6 in Fig.xx #
31      # there MUST be one intersection at least #
32      i:= TAB1[c];  j:= TAB2[c];
33      if  c_A ≠ 0 and c_B ≠ 0
34      then
35          # there are two intersections #
36          { x_A :=  p x e_i ;   x_B:=  p x e_j   }
37          # vector e_i is pre-defined for the i-th edge #
38      else # there is only one intersection point #
39          if  c_A = 0 then # x_B  is outside #
40              {   if c_B land MASK[c] ≠ 0  then  x_B :=  p x e_i  else x_B :=  p x e_j }
41          else if  c_B =0 then # x_A  is outside #
42              {   if c_A land MASK[c] ≠ 0  then  x_A :=  p x e_i  else x_A :=  p x e_j };
43      output (x_A , x_B )
44  end # S-Clip_Segment #
```

45 Modifikace algoritmu S-Clip pro ořezávání úsečky obdélníkem

Z výše uvedeného algoritmu je zřejmé, že S-Clip algoritmus pro ořezávání přímky a úsečky je jednoduchý, rozšiřitelný na konvexní n-úhelník. Tabulky TAB1, TAB2 a MASK se generují podle zadané hodnoty n, tj. počtu vrcholů zadaného n-úhelníka.

V případě nekonvexního n-úhelníka je nutné přímku, resp. úsečku reprezentovat pomocí parametrického vyjádření, výpočty modifikovat pro parametrické vyjádření a hodnoty parametru t jednotlivých průsečíků seřadit.



Obr. Ořezávání přímky nekonvexním n-úhelníkem

Podrobný popis algoritmu S-Clip a jeho modifikací viz:

- Skala,V.: A new approach to line and line segment clipping in homogeneous coordinates, The Visual Computer, ISSN 0178-2789, Vol.21, No.11, pp.905-914, Springer Verlag, 2005
- Skala,V.: S-Clip E2: A New Concept of Clipping Algorithms, Poster, SIGGRAPH Asia, ISBN 978-1-4503-1757-3, 2012, Singapore, 2012

5.1.6. Liang-Barsky algoritmus - doplnit

Doplnit

5.2.Algoritmy ořezávání v E^3

Algoritmy ořezávání v E^3 jsou určeny k odstranění těch částí základních grafických primitiv, které jsou mimo zobrazovanou oblast, tj. převážně úseček a trojúhelníků. V následujícím budou uvedeny jen základní algoritmy.

5.2.1. Cohen-Sutherland algoritmus

Rozšíření Cohen-Sutherland algoritmu do prostoru E^3 je vcelku triviální. Ořezávacím elementem je kvádr, jehož hrany jsou rovnoběžné s osami souřadného systému. Kódové slovo pozice koncových bodů úsečky $\boldsymbol{c}$ má nyní 6 bitů, tj. $\boldsymbol{c} = [near, far, left, right, bottom, top\ ]^T$, kde bit $near = 1$, pokud bod je před kvádrem, kterým ořezáváme. Bit $far = 1$, pokud bod je za kvádrem, kterým ořezáváme. Průsečíky přímky s kvádrem se počítají jako průsečík přímky s plochami kvádru a odpovídajícím způsobem je kód modifikován.

5.2.2. Cyrus-Beck algoritmus

Modifikace Cyrus-Beck algoritmu je také triviální. V případě E^2 byl konvexní n-úhelník tvořen vlastně průnikem polorovin, které jednotlivé hrany n-úhelníka definovaly. V případě E^3 je konvexní mnohostěn definován jako průnik poloprostorů, na nichž *stěny* konvexního n-úhelníka leží. Takže v algoritmu stačí zaměnit pojem hrany za pojem stěna a normála hrany za pojem normála roviny, resp. stěny. Algoritmus je stejný, pouze výpočet parametru musí obsahovat i koeficient roviny pro z složku, takže:

$$t_i = -\frac{\boldsymbol{n}_i^T \boldsymbol{x}_A + d_i}{\boldsymbol{n}_i^T \boldsymbol{s}}$$

kde každá rovina ρ_i je dána rovnicí $a_i x + b_i y + c_i z + d_i = 0$. Zbytek celého algoritmu je stejný.

Úloha

Formulujte CB algoritmus pro projektivní prostor, tj. když body jsou dány obecně v homogenních souřadnicích, tj. $\boldsymbol{x} = [x, y, z: w]^T$, vektor $\boldsymbol{s}$ je také určen v projektivním prostoru, tj. lineární interpolace s nelineární monotónní parametrizací, viz kap.9.1 (Lineární interpolace). Ukažte, že není zapotřebí operace dělení.

5.2.3. Sutherland-Hodgman algoritmus

Sutherland-Hodgmanův (S-H) algoritmus pro E^3 je vlastně jednoduchým rozšířením S-H algoritmu pro E^3, kdy místo průsečíku dané úsečky s hranou obdélníka se počítá průsečík úsečky se stěnou kvádru, obvykle jednotkové krychle, kdy výpočet je velmi jednoduchý. Toto se využívá v grafických akcelerátorech, kdy ořezávání úseček jednou rovinou krychle realizuje hardwarový modul, a tyto moduly jsou řazeny za sebou. Takže celý blok realizující ořezávání se skládá z 6 modulů, které jsou v sérii za sebou. Toto řešení má své výhody, a to:

- jednoduché HW řešení a
- stejnou dobu zpracování pro jednotlivé zpracovávané grafické elementy

Algoritmus S-H lze popsat sekvencí
(http://www.sunshine2k.de/coding/java/SutherlandHodgman/SutherlandHodgman.html)

```
for each clipping edge do
    for (i = 0; i < Polygon.length -1; i++)
        Pi = Polygon.vertex[i];
        Pi+1 = Polygon.vertex[i+1];
        if (Pi is inside clipping region)
            if (Pi+1 is inside clipping region)
                clippedPolygon.add(Pi+1)
            else
                clippedPolygon.add(intersectionPoint(Pi, Pi+1, currentEdge)
        else
            if (Pi+1 is inside clipping region)
                clippedPolygon.add(intersectionPoint(Pi, Pi+1, currentEdge)
                clippedPolygon.add(Pi+1)
    end for
    Polygon = clippedPolygon     // keep on working with the new polygon
end for
```

5.2.4. Clipping faces agains canonical volume - Hill, pp. 386

Doplnit

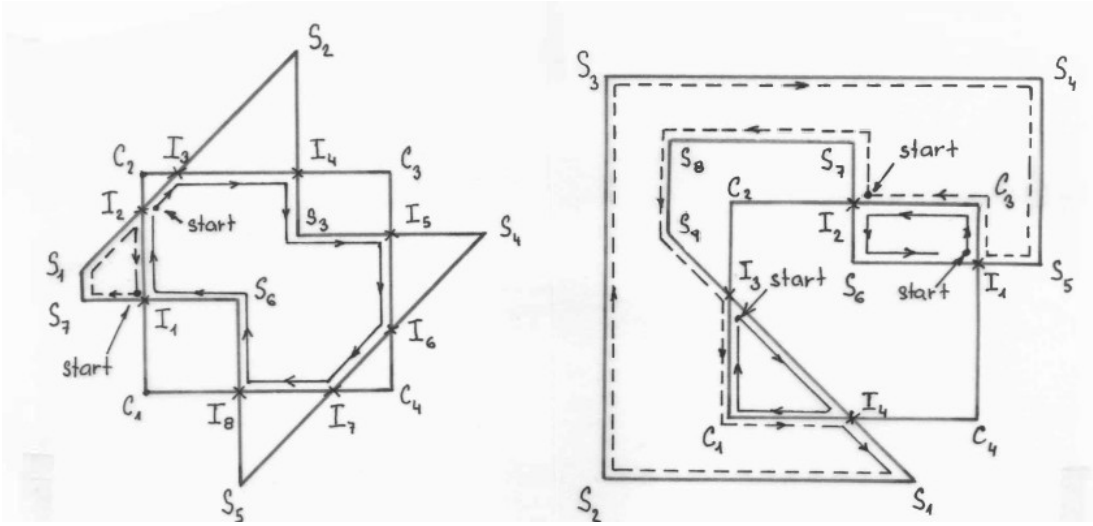
5.3.Operace s n-úhelníky v E^2

Mezi algoritmy ořezávání patří algoritmy pro operaci s n-úhelníky, zejména pak operace množinové, jako je průnik, sjednocení, rozdíl atd. Je zřejmé, že algoritmy se budou zásadně odlišovat podle typů n-úhelníků, se kterými jsou operace prováděny. Hlavním faktorem pak je, zda n-úhelníky jsou konvexní nebo nekonvexní. Při sjednocení dvou disjunktních konvexních n-úhelníků zajisté můžeme obdržet 2 samostatné konvexní n-úhelníky. Avšak v případě průniku dvou konvexních n-úhelníků můžeme obdržet n-úhelník nekonvexní. V případě nekonvexních n-úhelníků pak obě operace, tj. sjednocení i průnik, obecně vedou k množině nekonvexních n-úhelníků na výstupu dané operace. Operace s nekonvexními n-úhelníky jsou složitosti $O(m\,n)$, kde m, resp. n je počet vrcholů prvního, resp. druhého n-úhelníka. Pokud jeden n-úhelník je konvexní, lze očekávat složitost $O(m\,lg\,n)$.

Weiler-Athertonův je jedním ze základních algoritmů pro ořezávání nekonvexního n-úhelníka nekonvexním n-úhelníkem a vlastně realizuje operaci průniku.

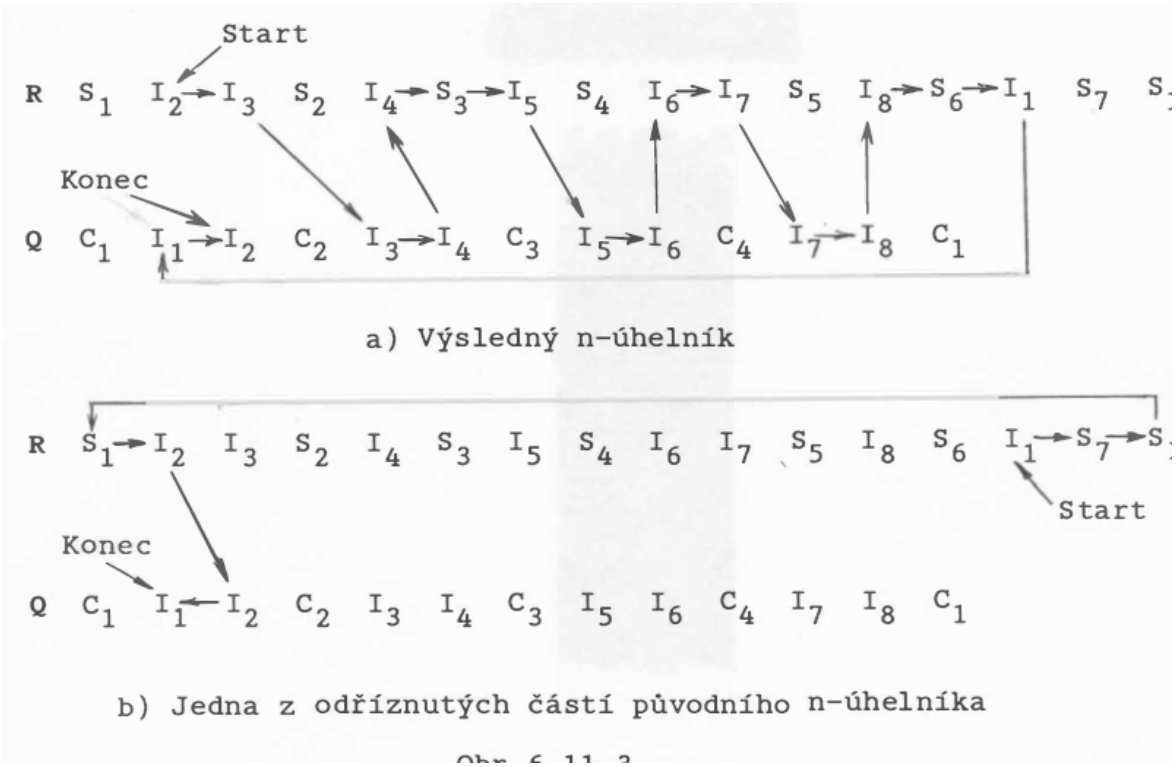
5.3.1. Weiler-Athertonův Algoritmus

Weiler-Athertonův (W-A)algoritmus používá orientovaný seznam vrcholů k reprezentaci n-úhelníka.



Obr.TTT: Reprezentace n-úhelníků a postup při generování výsledného n-úhelníka

Princip W-A algoritmu je založen na práci se seznamy vrcholů reprezentující nekonvexní n-úhelníky. Vzájemná orientace pořadí pak určuje, zda W-A algoritmus bude realizovat průnik, tj. oříznutí, nebo sjednocení apod.



Obr.XXX: postup procházení seznamem vrcholů a seznam reprezentující výsledný n-úhelník

Podrobnější popis W-A algoritmu lze nalézt v odborné literatuře nebo v:

- Skala,V.: Algoritmy počítačové grafiky II, skripta, Plzeň 2011
  (volně dostupné v elektronické formě)

5.3.2. Množinové operace

Množinové operace v rovině nejsou příliš časté, ale jejich použití v prostoru je poměrně časté. Uveďme alespoň základní typy operací:

Průnik	Sjednocení	Rozdíl	Doplněk
$C = A \cap B$	$C = A \cup B$	$C = A - B$	$C = -A$

kde A, B, C jsou obecně nekonvexní n-úhelníky, resp. množiny nekonvexních n-úhelníků.

Při složitějších operacích, které jsou vlastně definovány stromovou strukturou, je vhodné tuto strukturu transformovat tak, aby operace průniku a rozdílu byly pokud možno co nejníže, operace sjednocení pak naopak co nejvýše. Jde o proces známého pod názvem „optimalizace dotazu uživatele“ z databázových systémů. Množinové operace se používají obecně pro geometrické objekty a geometrické modelování, viz kap.13.6 (CSG stromy).

Je zřejmé, že pro efektivní algoritmy jsou nutné i datové struktury. Určitě není nejlepší způsob, jak reprezentovat např. konvexní n-úhelník, resp. konvexní mnohostěn pomocí průniku polorovin, resp. poloprostorů.

Volba datové struktury zásadně ovlivňuje efektivitu algoritmu.

6. Datové struktury

Datové struktury a jejich použití je klíčovou součástí všech aplikací. Datové struktury musí být pokud možno voleny tak, aby:

- byly používány v celé aplikaci, aby zbytečně nedocházelo k jejich převodu do jiných datových struktur apod.
- byly takové, aby mohly maximálně využívat *cache* paměti při předpokládaném zpracování, pokud možno eliminovat indexování pomocí dvou nebo tří indexů
- podporovaly konzistenci geometrických modelů, tj. musí umožňovat snadnou kontrolu konzistence dat a geometrických aspektů
- nevyžadovaly časté prohazování stránek virtuální paměti atd.

Je nutné si uvědomit, že při současném rozsahu zpracovávaných dat už není příliš kritická rychlost elementárních operací, ale kritická je rychlost přenosu dat z paměti, případně z velkokapacitní paměti do cache pamětí daného systému.

6.1.Základní typy popisu dat

Popis geometrických entit můžeme rozdělit v zásadě na:

$$\text{objekt zadaný} \begin{cases} \text{implicitně} \\ \text{parametricky} \\ \text{explicitně} \end{cases}$$

Jednotlivé typy jsou pro jednoduchost ukázány na rovnicích přímky a roviny.

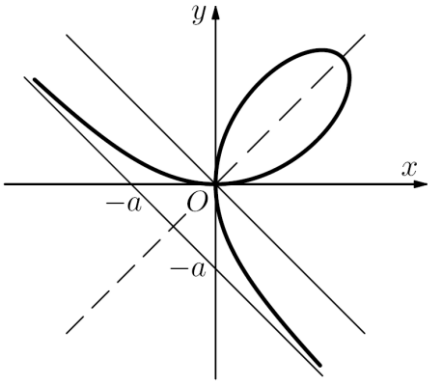
Implicitní zadání

Implicitní forma je určena implicitní funkcí ve tvaru $F(\boldsymbol{x}) = 0$. Objekt je tedy určen v případě E^2 křivkou nebo v případě E^3 povrchem, splňujícím danou rovnici. Je nutné zdůraznit, že obecně není dána orientace a vynásobení funkce libovolnou konstantou $q \neq 0$ určuje stejnou křivku, resp. stejný povrch.

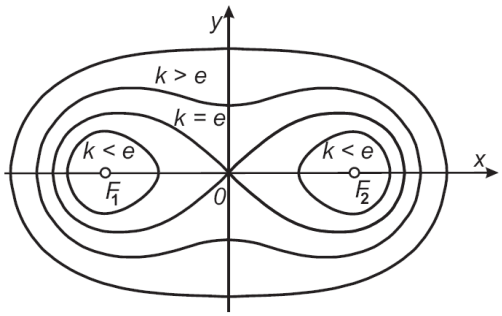
Přímka	Rovina	Kružnice
$Ax + By + C = 0$	$Ax + By + Cz + D = 0$	$x^2 + y^2 - R^2 = 0$

Implicitní popis je velmi silný, avšak v mnoha případech vede k řešení nelineární algebraické rovnice nebo jejich soustav.

Příklad složitějších funkcí



Descartův list $x^3 + y^3 - 3axy = 0$



Cassiniho křivky

Parametrické zadání

Parametrická forma je často používána, neboť poskytuje orientaci v rovině parametrů. Typickou ukázkou je např. lineární interpolace, nicméně parametricky lze vyjádřit mnohé křivky nebo plochy.

Přímka	Rovina	Kružnice
$\boldsymbol{x}(t) = \boldsymbol{x}_1 + (\boldsymbol{x}_2 - \boldsymbol{x}_1)t$	$\boldsymbol{x}(u,v) = \boldsymbol{x}_1 + (\boldsymbol{x}_2 - \boldsymbol{x}_1)u + (\boldsymbol{x}_3 - \boldsymbol{x}_1)v$	$\boldsymbol{x} = R\begin{bmatrix} cos\varphi \\ sin\varphi \end{bmatrix}$ nebo $\boldsymbol{x} = R\left[\frac{2t}{1+t^2}, \frac{1-t^2}{1+t^2}\right]^T$

Parametrická forma je poměrně často používána pro geometrické modelování křivek a ploch. Při vykreslování se parametrické křivky nahrazují lomenou čárou, parametricky zadané plochy pak množinou trojúhelníků nebo trojúhelníkovou sítí.

Podrobnější informace, viz kap.10 (Parametrické křivky a plochy), kdy budou popsány kubické křivky a bikubické plochy.

Parametrickým popisem se dají popisovat i objekty, které mají jen jednostrannou plochu. Jako příklad uveďme Kleinovu lahev, která je definována takto:

Pro $0 \le v \le 2\pi, r > 0$ a $0 \le u < \pi$

$$x = 6\cos u\,(1+\sin u) + 4r\left(1-\frac{\cos u}{2}\right)\cos u\cos v$$

$$y = 16\sin u + 4r\left(1-\frac{\cos u}{2}\right)\sin u\cos v$$

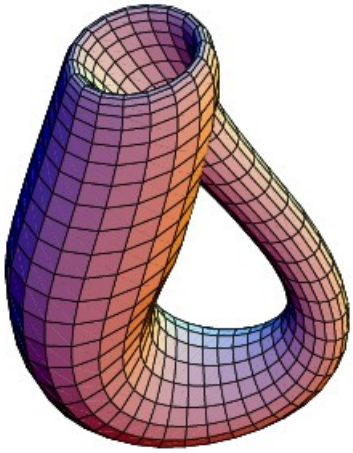
$$y = 4r\left(1-\frac{\cos u}{2}\right)\sin v$$

a pro $0 \le v \le 2\pi, r > 0$ a $\pi \le u < 2\pi$

$$x = 6\cos u\,(1+\sin u) - 4r\left(1-\frac{\cos u}{2}\right)\cos u\cos v$$

$$y = 16\sin u$$

$$y = 4r\left(1-\frac{\cos u}{2}\right)\sin v$$



Wikipedia

Explicitní zadání

Explicitní forma je asi v praxi nejčastějším případem, neboť pro zadané hodnoty nezávisle proměnné $\boldsymbol{x}$ umožňuje výpočet funkční hodnoty v tomto bodě, tj. $y = f(\boldsymbol{x})$.

Uvážíme-li rovnici přímky v implicitní formě, pak její explicitní ekvivalenty jsou:

$Ax + By + C = 0$	$y = kx + q$ $k \neq \infty$	$x = my + p$ $m \neq \infty$

Je tedy zřejmé, že pro přímku se sklonem do $\pm 45°$ je vhodnější pro numerické výpočty rovnice $y = kx + q$, v opačném případě pak je vhodné použít rovnice $x = my + p$. Např. v případě kružnice dostáváme $y_{1,2} = \pm\sqrt{R^2 - x^2}$ nejen dvě hodnoty, ale pro $x \to R$ bude hodnota y zřejmě vypočetna s velkou numerickou chybou.

6.2.Základní geometrická primitiva

V současných grafických systémech jsou základní geometrické elementy, pro které je optimalizován grafický hardware.

Základní primitiva

- body – zadávají se obecně v E^3, resp. v P^3 v homogenních souřadnicích
- úsečka (line segment) – je dána koncovými body
- trojúhelník (triangle) – je určen 3 body. Orientace vrcholů trojúhelníka v E^3 obecně není možná bez referenčního bodu, např. pozice pozorovatele, nebo nějakou úmluvou, např. že normála trojúhelníka směřuje z objektu ven atd.

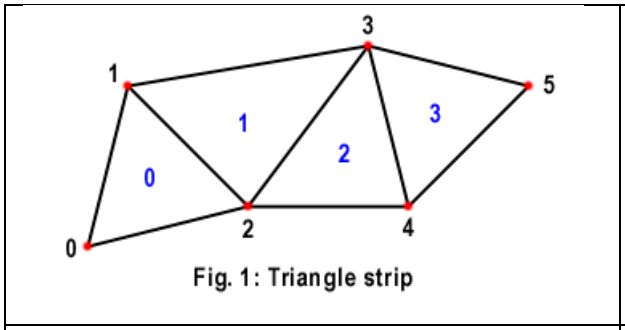
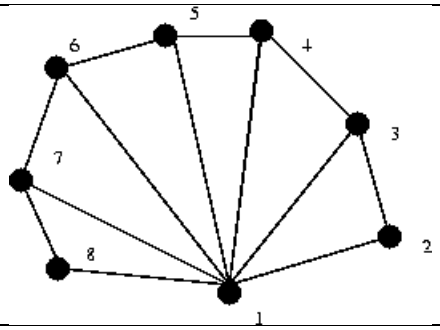


Fig. 1: Triangle strip



Složená primitiva, která se používají, pro urychlení vykreslování:

- pás trojúhelníků (triangle strip) Pro vykreslení se zadává jen posloupnost bodů $P_0, P_1, P_2, \ldots, P_5$. V případě vykreslování jako množiny trojúhelníků bychom body $P_1, P_2, \ldots, P_4$ transformovali vícekrát a byly by zpracovávány pro každý trojúhelník samostatně.

- vějíř trojúhelníků (triangle fan) – nepoužívá se příliš často, ale opět šetří čas zpracování. Pro vykreslení se zadává jen posloupnost bodů $P_1, P_2, \ldots, P_8$, místo zadávání 6 trojúhelníků.

V některých systémech se ještě používají čtyřúhelníky, ale zde je nutné zdůraznit, že i když jsou rovinné z hlediska čistě matematického, nejsou rovinné z důvodů numerické přesnosti.

Obecně pak vykreslování grafických primitiv je dáno sekvencí:

- nastav atributy – nastavuje se např. barva apod.
- vykresli dané primitivum, resp. primitiva

V mnoha postupech, např. při stínování, je nutné určit normálu trojúhelníka. Je zřejmé, že normálu lze určit např. jako $\boldsymbol{n} = (P_1 - P_0) \times (P_2 - P_0)$.

Dosud bylo víceméně předpokládáno, že se pracuje s povrchy objektů, které jsou reprezentovány množinou trojúhelníků, které jsou dále zpracovávány do vizuální podoby. Jde tedy o zpracování plošných elementů, které jsou v obecné poloze v prostoru E^3.

6.3.Reprezentace třírozměrných objektů a scén

V případě plošných elementů bylo možné reprezentovat n-úhelníky buď uspořádaným seznamem po sobě jdoucích vrcholů, neboť v E^2 je relevantní pojem orientace, nebo seznamem hran s odkazem do tabulky se souřadnicemi příslušných vrcholů. V případě E^3 je však nutné reprezentovat uzavřený povrch, resp. objem geometrického objektu.

Reprezentace geometrických objektů lze rozdělit na:

- *reprezentace hraniční* (Boundary representnation) B-rep
  - *hranové* reprezentace slouží převážně k reprezentaci „drátěných“ modelů a není možné řešit viditelnost ploch, neboť tato informace není v datové struktuře k dispozici
  - *plošné* reprezentace, např. okřídlená hrana, half-edge atd., reprezentují povrch tělesa, takže obsahují informaci o plochách, vrcholech a hranách daného objektu.
    V případě parametrických ploch, viz kap.10.3 (Transformace kubických parametrických křivek), datová struktura obsahuje informace o vrcholech a případně hraničních křivkách atd.
- *reprezentace objemové*
  - *diskrétní* – objekt je reprezentován v diskrétní 2D nebo 3D mřížce a uzel mřížky je asociován s hodnotou skalární nebo vektorovou. Typickou ukázkou jsou CT a MRI data, viz kap.16.1 (Vizualizace dat)
  - *CSG stromy* (Constructive Solid Geometry) apod., je objekt reprezentován např. pomocí „orientované“ implicitní funkce $F(\boldsymbol{x})$, kdy se např. předpokládá, že $F(\boldsymbol{x}) < 0$ pro všechny body uvnitř tělesa $F(\boldsymbol{x}) > 0$, pro body vně tělesa a $F(\boldsymbol{x}) = 0$ reprezentuje povrch objektu

Vedle uvedených datových struktur jsou používány speciální datové struktury, zejména pro účely urychlování, pro reprezentaci rozmístění objektů v rovině nebo prostoru.

Obecně lze datové struktury a reprezentaci objektů a jejich pozic v prostoru rozdělit na dvě skupiny, a to:

- hierarchické datové struktury sloužící především k účelům „zjemňování“, resp. zmenšování prostoru, který je nutno při výpočtu uvažovat
  - statické – sloužící především k uložení hierarchie, resp. vazeb mezi objekty, jejich hloubka je předem dána
  - adaptivní – hloubka struktury se může měnit podle složitosti scény v té které části prostoru
- ne-hierarchické datové struktury – tj. např. patří množina trojúhelníků, n-úhelníků, mnohostěnů atd. bez struktur popisující jejich vzájemný vztah

V následujícím budou uvedeny jen základní datové struktury. Pokročilé datové struktury a jejich aplikace lze nalézt v:

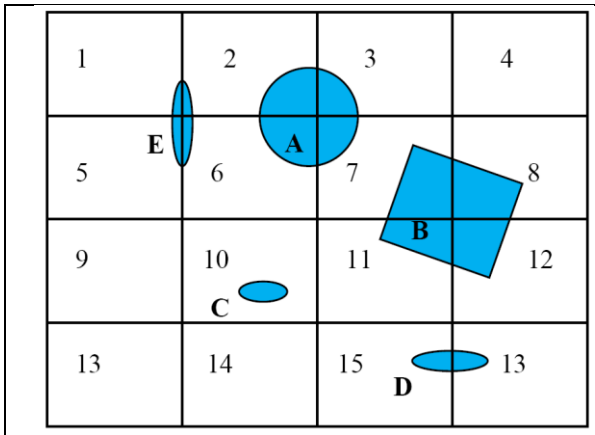
- Langetepe,E., Zachmann,G.: Geometric Data Structures for Computer Graphics, A.K.Peters, 2006

6.4.Dělení prostoru a Binární masky

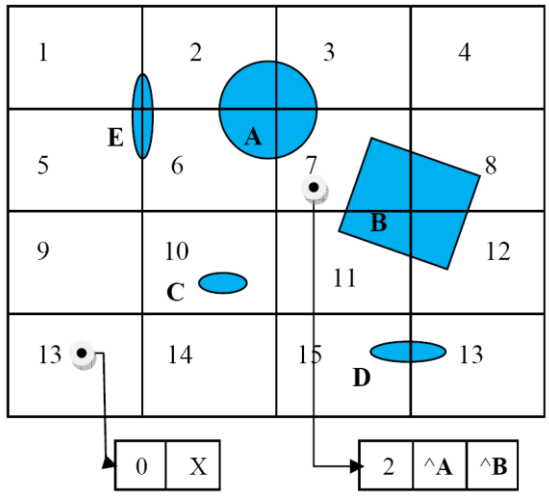
V mnoha aplikacích se je nutné zjistit, zda v nějaké části prostoru je nějaký objekt. Jde tedy o problém lokalizace objektu v určité části prostoru. Jednou z možností efektivního řešení této úlohy je dělení prostoru na menší prostory a pro ně pak uchovávat identifikaci objektů, které s daným prostorem incidují, tj. mají neprázdný průnik. Rento přístup v zásadě využívají techniky typu „rozděl a panuj“ (Divide and Conquer).

6.4.1. Dělení prostoru

Standardní dělení prostoru (Space Subdivision) je poměrně oblíbená jednoduchá technika, kdy daný prostor rozdělíme, většinou rovnoměrně, na menší prostory. Pro každý prostor pak uchováváme, např. v seznamu, identifikátory objektů, které s daným prostorem sdílí nějakou část.



Obr.XXX: Rozložení objektů v prostoru



Obr.ZZZ: Struktura pro dělení prostoru

Paměťová náročnost techniky dělení prostoru může být odhadnuta na:

$$O(p\ q\ M^d)$$

kde M je počet dělení na jedné ose, d je dimenze prostoru, p je pravděpodobnost, že „průměrný“ objekt zasáhne sub-prostor, $q = q(M, p, s)$ je funkce závisející na jemnosti dělení a velikosti „průměrného“ objektu, podrobně viz reference

Pak paměťové nároky pro standardní dělení prostoru (Standard Space subdivision) jsou:

$$Mss = M^d(1 + pq + 1) * 4 = M^d(pq + 2) * 4\ \ [B]$$

kde: alokují se 4 B na pointer.

Je tedy zřejmé, že paměťové nároky prudce vzrůstají s jemností dělení prostoru, tj. hodnotou M a dimenzí d.

6.4.2. Rezidenční maska

Jiný přístup zavedl jsou rezidenční masky (Residency masks), viz Cychosz, které používají bitové masky. Pro každý objekt je k dispozici vektor bitů, který říká, zda daný objekt se v dané oblasti nachází, tzn. že počet birů je dán celkovým počtem dělení prostoru. Matice Q ukazuje stav pro situalci na obr. XXX

$$\boldsymbol{Q} = \begin{bmatrix} 0000 & 0000 & 0110 & 0110 \\ 0000 & 0000 & 1100 & 0000 \\ 0000 & 0010 & 0000 & 0000 \\ 1100 & 0000 & 0000 & 0000 \\ 0000 & 0000 & 0011 & 0011 \\ 15 & bits & \dots\dots & 0 \end{bmatrix} = \begin{bmatrix} \text{Object } \mathbf{A} \\ \text{Object } \mathbf{B} \\ \text{Object } \mathbf{C} \\ \text{Object } \mathbf{D} \\ \text{Object } \mathbf{E} \\ \end{bmatrix}$$

Lze ukázat, že paměťové nároky jsou určeny

$$M_{RM} = pM^k/8 \ [B]$$

Metoda rezidenčních masek umožňuje rychlou detekci kolize objektů pomocí bitové operace ***land***, takže při zkoumání možné detekce objektů $\boldsymbol{C}$ a $\boldsymbol{D}$ řešíme jednoduchou podmínku:

$$\boldsymbol{Q}[3,*] \ \boldsymbol{land} \ \boldsymbol{Q}[4,*] \neq [0,\dots,0]$$

kde p je počet objektů v prostoru.

6.4.3. Binární masky

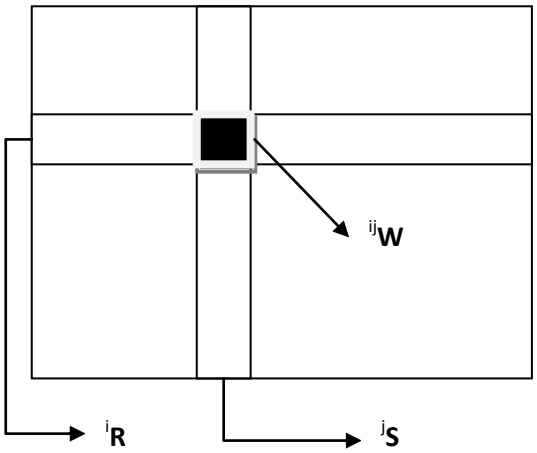
Binární masky (Binary Masks) jsou technikou, založenou na diametrálně jiném přístupu. Metoda rezidenčních masek velmi dobře zodpoví otázku

Q1: Nalezni oblasti, které incidují s daným objektem.

Nicméně v převážné většině případů potřebujeme odpovědět trochu jinou otázku, a to:

Q2: Nalezni všechny objekty, které incidují s danou oblastí.

Pokud dotazy zanalyzujeme, pak jsou to otázky komplementární. Uvažme situaci na obr.TTT



Obr.TTT: Princip binárních masek

Uvažme množinu ${}^{i}\boldsymbol{R}$, resp. ${}^{j}\boldsymbol{S}$ objektů, které incidují s řádkovým „řezem“ i, resp. se sloupcovým „řezem“ j. Pak množina ${}^{ij}\boldsymbol{W}$, která je definována jako:

$${}^{ij}\boldsymbol{W} = {}^{i}\boldsymbol{R} \cap {}^{j}\boldsymbol{S}$$

určuje objekty, které ***mohou*** incidovat s oblastí na pozici ij. Obdobně pro E^3 dostáváme:

$${}^{ijk}\boldsymbol{W} = {}^{i}\boldsymbol{R} \cap {}^{j}\boldsymbol{S} \cap {}^{k}\boldsymbol{T}$$

kde ${}^{k}\boldsymbol{T}$ je množina objektů, které incidují s „hloubovým řezem“ k.

Podrobnější analýzou snadno zjistíme, že se podařilo převést paměťovou složitost z $O(M^d)$ na $O(dM)$. V případě E^3 pro $M = 256$ paměťové nároky snížíme v poměru $\mathbf{256^3 \rightarrow 3 * 256}$!

Z hlediska implementačního je vhodné množiny ${}^{i}\boldsymbol{R}$, ${}^{j}\boldsymbol{S}$, ${}^{k}\boldsymbol{T}$ implementovat jako bitový vektor $\boldsymbol{\Omega}$ o délce p bitů, kde p je počet objektů v prostoru. Bit ${}^{\mathrm{i}}\mathrm{r}_{\mathrm{k}} = 1$, pokud k-tý objekt inciduje s i-tou řádkou, jinak ${}^{\mathrm{i}}\mathrm{r}_{\mathrm{k}} = 0$.

Paměťová náročnost metody bitových masek M_{BM} je:

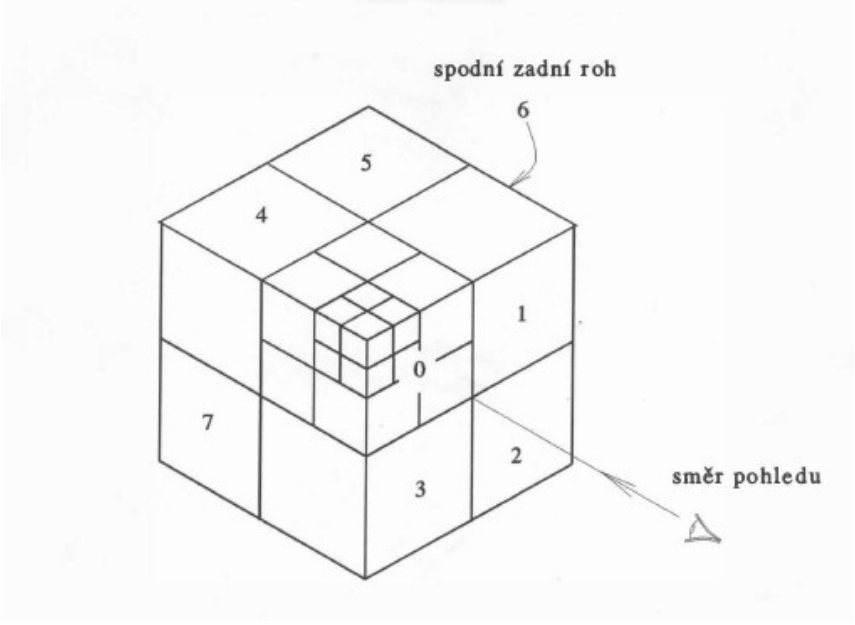
$$M_{BM} = \frac{dpM}{8} \quad [B]$$

Podrobněji viz:

- Skala,V.: An Efficient Space Partitioning Method Using Binary Maps, 11th Conference SIP'2012 conference , pp.121-124, ISBN: 978-1-61804-081-7, St.Malo, WSEAS, France, 2012 (CLICK off-line)
- Skala,V.: Memory Saving Technique for Space Subdivision Technique, Machine Graphics and Vision, Vol.2, No.3, pp.237-250, , ISSN 1230-0535, 1993 (CLICK off-line)
- Cychosz,J.M., Use of Residency Mask and Object Space Partitioning to Eliminate Ray Object Intersection Calculation, Graphics Gems III (Ed.Kirk,D.), pp.284-287, 1992.

6.4.4. Quadtree a Octree

Quadtree a Octree jsou techniky hierarchického dělení prostoru. V zásadě jde o stejný princip a oblast daného prostoru obsahující objekty se postupně dělí na menší a menší sub-oblasti, pokud s danou oblastí inciduje větší než specifikovaný počet objektů. V případě E^2 dostáváme techniku nazývanou *Quadtree*, v případě E^3 dostáváme *Octree*.



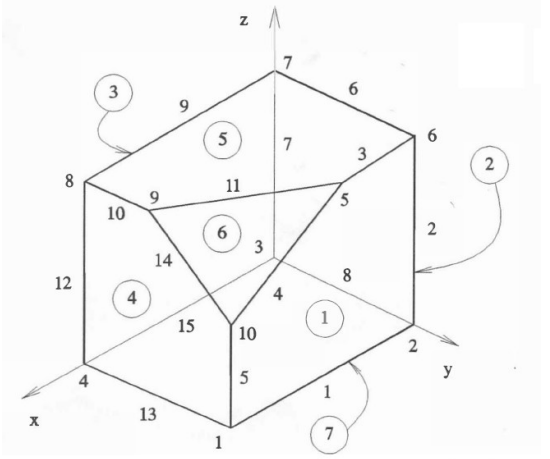
Obr.XXX: Octree datová struktura

Hlavní nevýhodou datových struktur Quadtree a Octree, že nejsou invariantní např. vůči rotaci. Takže velmi často dochází k nutnosti jejich opětovného vytvoření, což je výpočetně náročné.

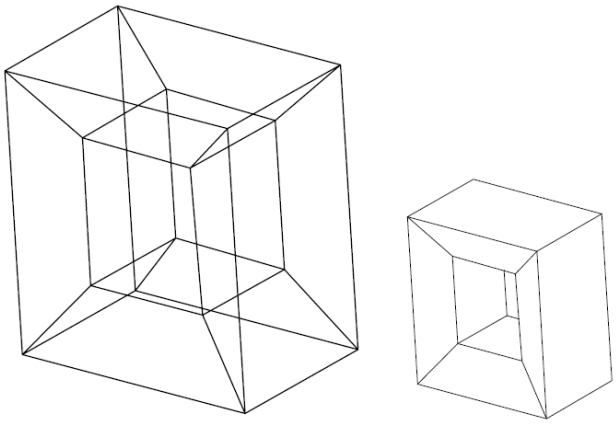
V mnoha aplikacích se vyžaduje hraniční reprezentace, zejména ve strojírensky orientovaných CAD systémech, k reprezentaci povrchu a manipulace s ním. Vedle toho je však v mnoha aplikacích počítat s mechanickými veličinami jako je objem, váha, moment setrvačnosti atd. K výpočtu těchto veličin však povrchová reprezentace není vhodná.

6.5.Hraniční reprezentace

Hraniční reprezentace jsou charakterizovány popisem hranic daného objektu a jsou označovány termínem *B-rep*, zkratkou od anglického *Boundary representation*. Pro jejich vysvětlení uvažme objekt na obr.TTT



Obr.TTT: Objekt reprezentovaný hraniční reprezentací



Obr.XXX: Drátěný model a jedna z možných reprezentacíe

Nejjednodušší reprezentací je reprezentace hranová.

Hranová reprezentace

Hranová reprezentace je nejjednodušší datovou strukturou, neboť je dána pouze tabulkou vrcholů V s jejich souřadnicemi a tabulkou hran H, která pro každou hranu určuje její koncové body.

Tabulka vrcholů V je dána:

$$\begin{array}{lcccccccccc} \texttt{vrchol} & \texttt{1.} & \texttt{2.} & \texttt{3.} & \texttt{4.} & \texttt{5.} & \texttt{6.} & \texttt{7.} & \texttt{8.} & \texttt{9.} & \texttt{10.} \\ \mathbf{V} = & \begin{bmatrix} 4 & 0 & 0 & 4 & 1 & 0 & 0 & 4 & 4 & 4 \\ 3 & 3 & 0 & 0 & 3 & 3 & 0 & 0 & 2 & 3 \\ 0 & 0 & 0 & 0 & 4 & 4 & 4 & 4 & 4 & 2 \end{bmatrix} \end{array}$$

Tabulka hran H je dána:

$$\begin{array}{lcccccccccccccc} \texttt{hrana} & \texttt{1.} & \texttt{2.} & \texttt{3.} & \texttt{4.} & \texttt{5.} & \texttt{6.} & \texttt{7.} & \texttt{8.} & \texttt{9.} & \texttt{10.} & \texttt{11.} & \texttt{12.} & \texttt{13.} & \texttt{14.} & \texttt{15.} \\ \mathbf{H} = & \begin{bmatrix} 1 & 2 & 6 & 5 & 10 & 6 & 7 & 3 & 7 & 8 & 9 & 8 & 4 & 9 & 3 \\ 2 & 6 & 5 & 10 & 1 & 7 & 3 & 2 & 8 & 9 & 5 & 4 & 1 & 10 & 4 \end{bmatrix} \end{array}$$

Hranová reprezentace není vhodná, pokud budeme chtít určovat viditelnost hran, neboť schází informace o plochách. Proto se také někdy označuje termínem *drátěný model*, který je nejednoznačný z hlediska interpretace ploch. na Obr.XXX je uvedena jedna ze 3 možných interpretací.

Plošná reprezentace

K řešení viditelnosti, tj. eliminaci neviditelných ploch a jejich částí, které jsou zakryty jinými plochami daného či jiného objektu, je zapotřebí informace o plochách. Je tedy nutné hranový model rozšířit o hraniční plochy, které jsou definovány tabulkou ploch P.

číslo plochy	počet hran	číslo hrany ohraničující plochu
1	5	1 , 2 , 3 , 4 , 5
2	4	2 , 6 , 7 , 8
3	4	7 , 9 , 12 , 15
4	5	12 , 13 , 5 , 14 , 10
5	5	6 , 9 , 10 , 11 , 3
6	3	4 , 11 , 14
7	4	1 , 8 , 15 , 13

Tabulka ploch P.

Tabulky V, H a P se někdy ještě doplňují o informaci, které dva n-úhelníky sdílejí danou hranu, viz tabulka H'.

hrana	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	11.	12.	13.	14.	15.
H' =	1	2	6	5	10	6	7	3	7	8	9	8	4	9	9
	2	6	5	10	1	7	3	2	8	9	5	4	1	10	5
	1	1	1	1	1	2	2	2	3	4	5	3	4	4	3
	7	2	5	6	4	5	3	7	5	5	6	4	7	6	7

Obr.xxx: Modifikovaná tabulka hran H

První dva řádky tabulky obsahují indexy vrcholů hran a druhé dva řádky obsahují indexy ploch, které danou hranu sdílí. Pokud lze dodržet uspořádání vrcholů jednotlivých ploch konzistentně, pak je možné tabulku hran H vynechat a modifikovaná tabulka P' má pak tvar:

číslo plochy	počet vrcholů	čísla vrcholů plochy
1	5	1 , 2 , 3 , 4 , 5
2	4	2 , 8 , 7 , 6
3	4	15 , 12 , 9 , 7
4	5	13 , 5 , 14 , 10 , 12
5	5	11 , 3 , 6 , 9 , 10
6	3	4 , 11 , 14
7	4	1 , 13 , 15 , 8

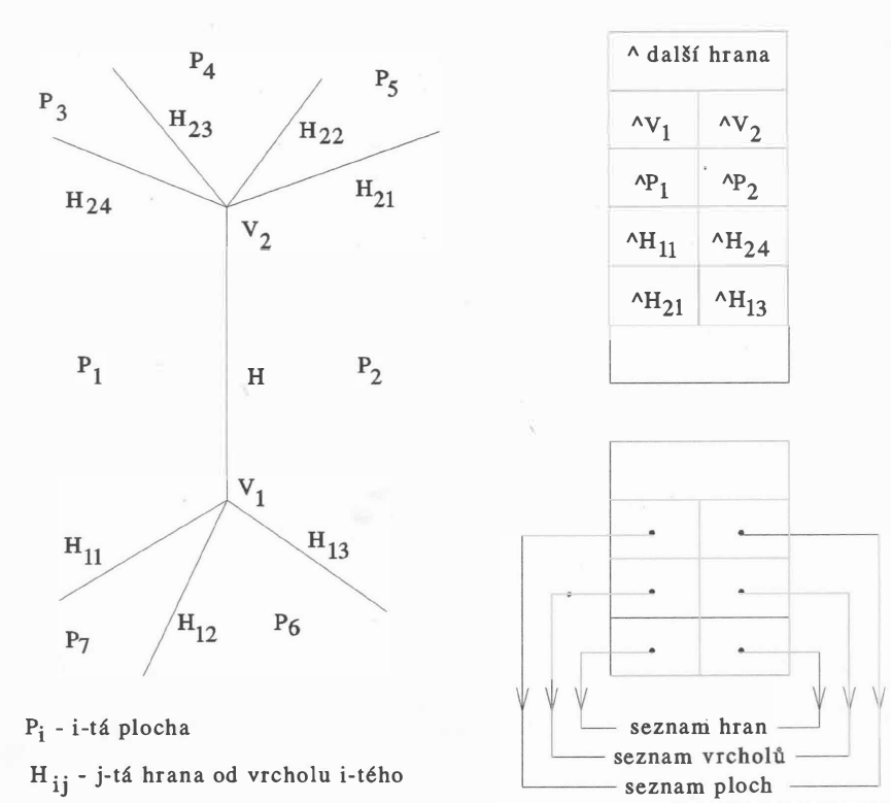
Obr.: Modifikovaná tabulka ploch P'

Pro vnitřní reprezentaci je však vhodné z tabulky P' odvodit tabulky P a H s tím, že u odkazu na hranu v tabulce P je uložena i informace o orientaci hrany.

Pro rozsáhlé aplikace byla vyvinuta speciální datová struktura známá pod označením okřídlená hrana, která je založena na seznamech.

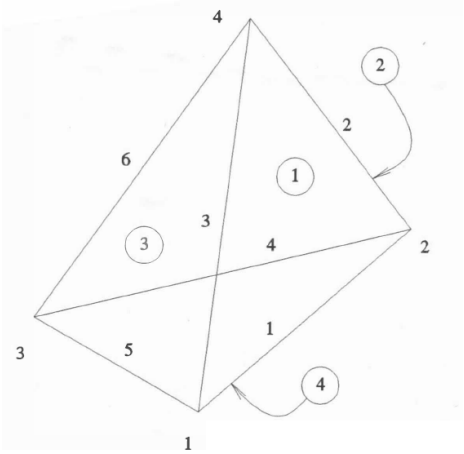
Okřídlená hrana

Okřídlená hrana (winged edge) je datová struktura byla původně vyvinuta pro reprezentaci povrchu objektu, který je tvořen rovinnými n-úhelníky, a efektivní zpracování objektů definovaných rovinnými plochami.



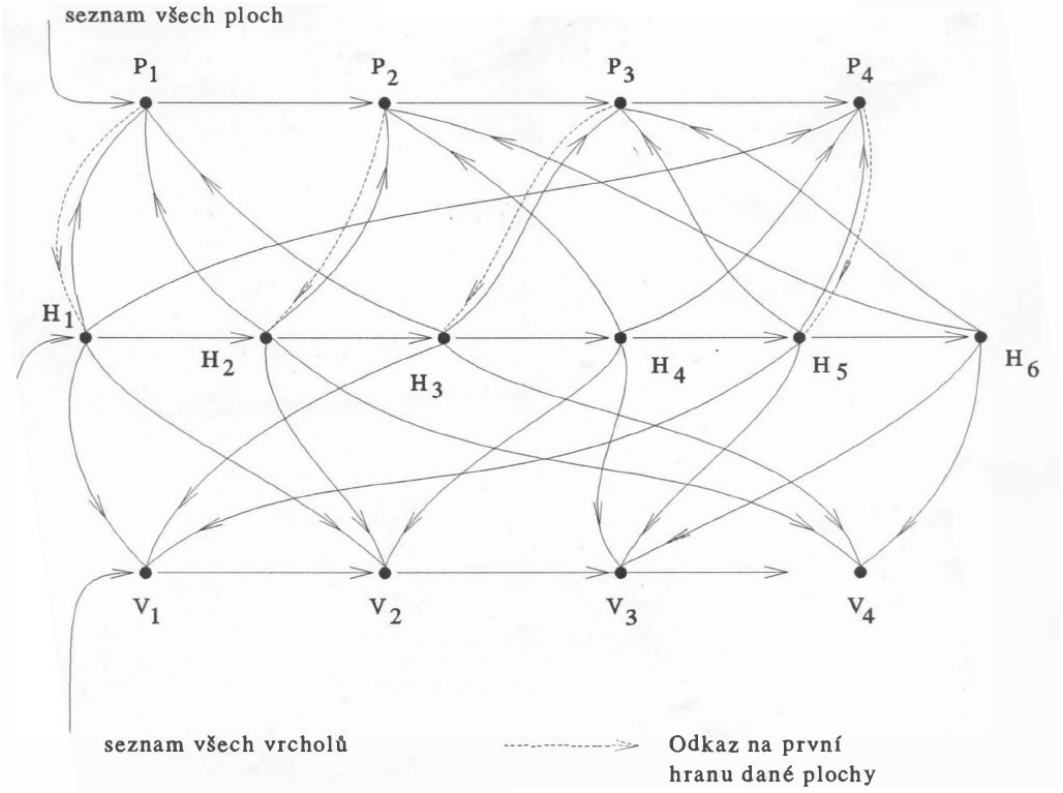
Obr.xxx: jhjjjjj

Datová struktura je založena na seznamové architektuře. Uveďme pro ilustraci, jak by datová struktura vypadala pro čtyřstěn, viz obr.QQQQ

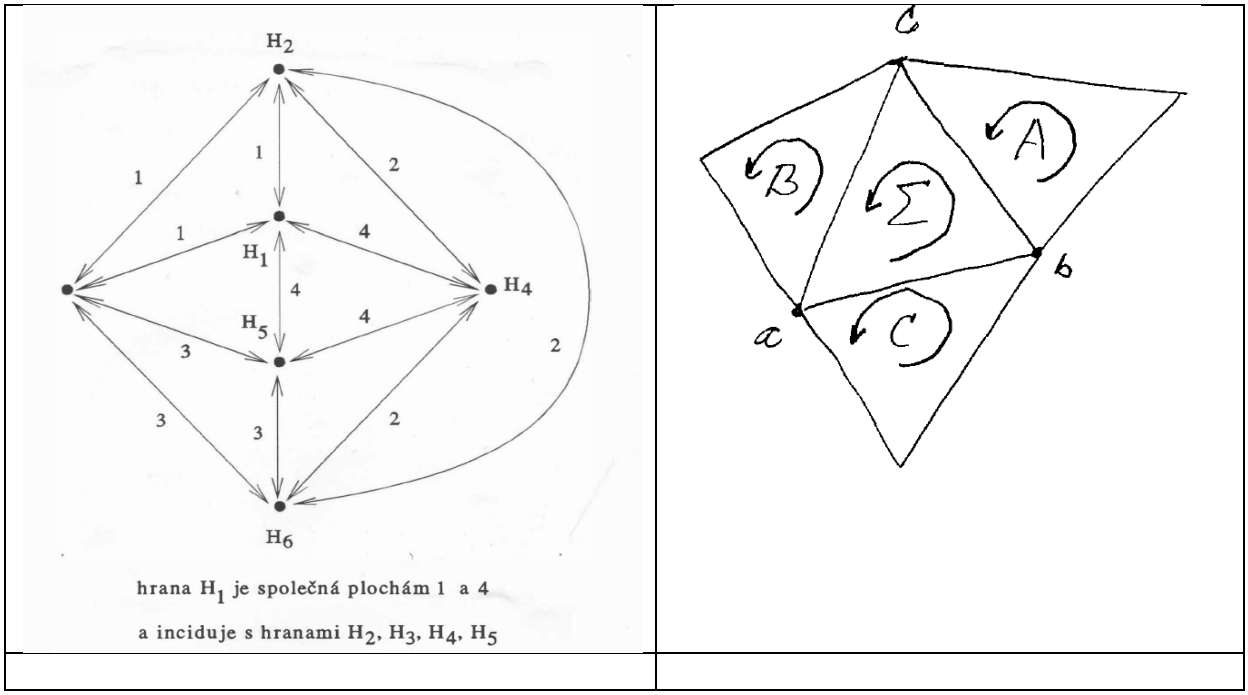


Obr.QQQQ: čtyřstěn a jeho plochy, hrany a vrcholy

Datová struktura okřídlená hrana pak pro daný čtyřstěn má tvar:



V „grafovém„ pojetí je pak struktura tvaru:



Obr.QQQQ

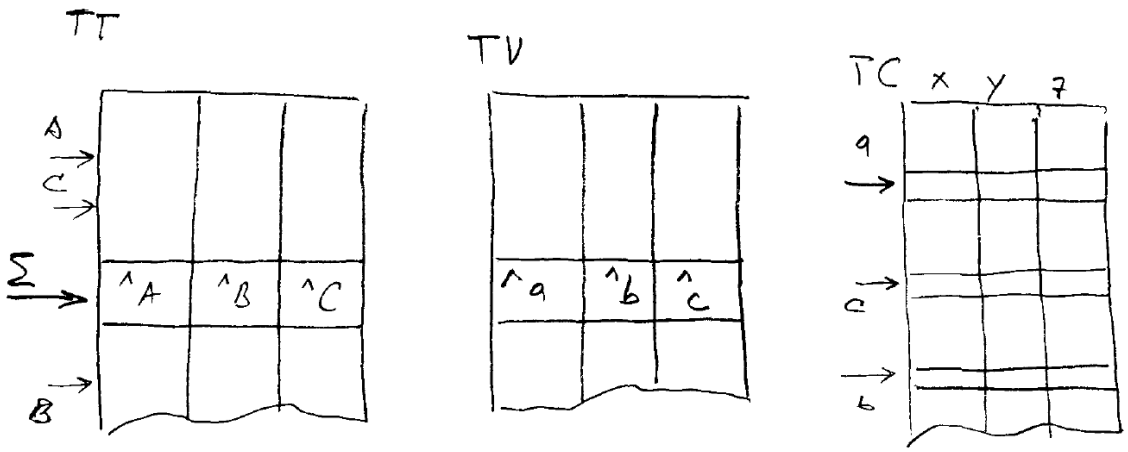
Vzhledem k omezené přesnosti v reprezentaci souřadnic vrcholů je rozumné n-úhelníky reprezentovat pomocí trojúhelníkové sítě. Pak se okřídlená hrana podstatně zjednoduší a je reprezentovatelná jednoduchými datovými strukturami.

Modifikace pro trojúhelníkové sítě

Datovou strukturu „okřídlená hrana“ je možné modifikovat pro trojúhelníkové sítě poměrně jednoduše. Uvažme část trojúhelníkové sítě, viz obr.xxxx, včetně jejich orientace. Pravidla indexace jsou jednoduchá, a to:

- proti vrcholu s indexem a je trojúhelník s indexem A
- pořadí v tabulce TT a TV si odpovídají
- orientaci všech trojúhelníků je konzistentní pro celý povrch, např. tak, že normálový vektor směřuje z objektu

Pak modifikovaná datová struktura je:



kde $\boldsymbol{TT}$ je tabulka trojúhelníků, $\boldsymbol{TV}$ je tabulka vrcholů, $\boldsymbol{TC}$ je tabulka souřadnic.
V případě potřeby lze datovou strukturu doplnit o tabulku $\boldsymbol{TN}$, kde jsou uloženy normálové vektory ve vrcholech.

Poznamenejme, že se používají indexy do tabulky, nikoliv ukazatelé, a tabulky by měly být implementovány jako jednorozměrné pole struktury, např. **struct** (x,z,z: **real**), aby nebylo nutné používat dvourozměrné indexování. Pokud se v datové struktuře provádějí změny typu zrušení a vkládání je vhodné implementovat „memory management“ pro vlastní přidělování volných pozic v tabulkách.

Modifikace pro trojúhelníkové sítě

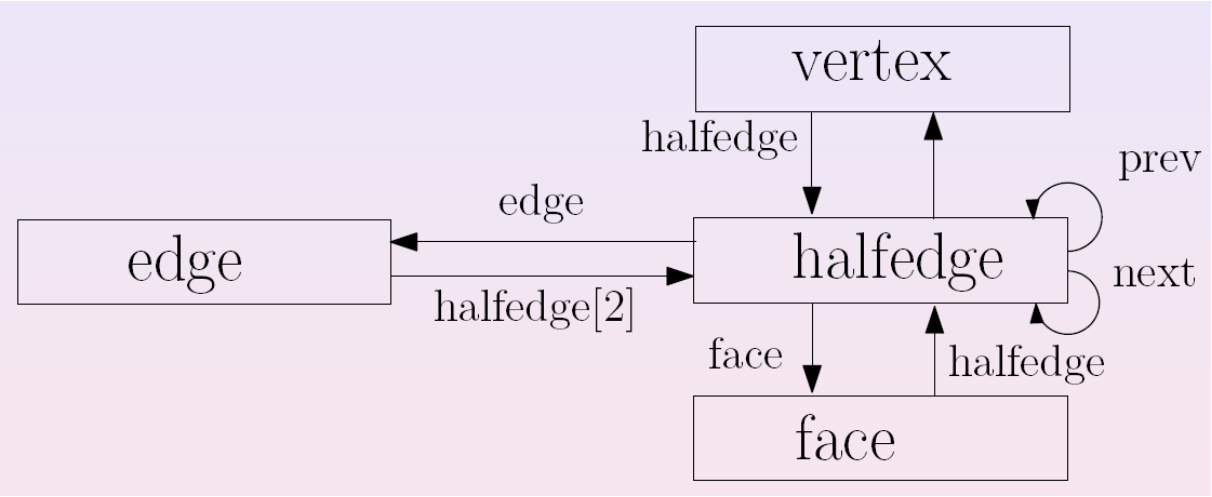
Modifikace výše uvedené datové struktury pro tetrahedronové sítě je velmi jednoduchá, neboť v zásadě stačí přidat jeden sloupec k tabulkám $\boldsymbol{TT}$ a $\boldsymbol{TV}$.

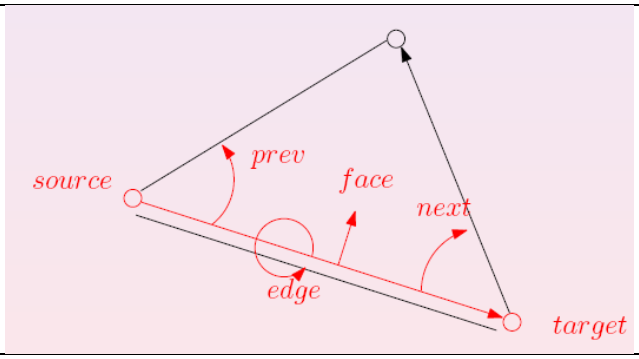
Datová struktura „Half-edge“

Dalším datovým typem je datová struktura „half-edge“. Její přesnou definici lze nalézt např. v http://www.cgal.org/Manual/latest/doc_html/cgal_manual/HalfedgeDS/Chapter_main.html
Podstatnou výhodou je především snadnost realizace Eulerových operátorů a spotřeba paměti je minimalizována.

Datová struktura má následující části:

- vrcholy
- half-edge, tj. orientovaná hrana
- hrana, tj. neorientovaná
- plocha orientovaná





DODELAT

Pro přenos geometrických dat se používá mnoho formátů, mnohé z nich byly standardizovány. Jedním ze základních a jednoduchých formátů je formát STL.

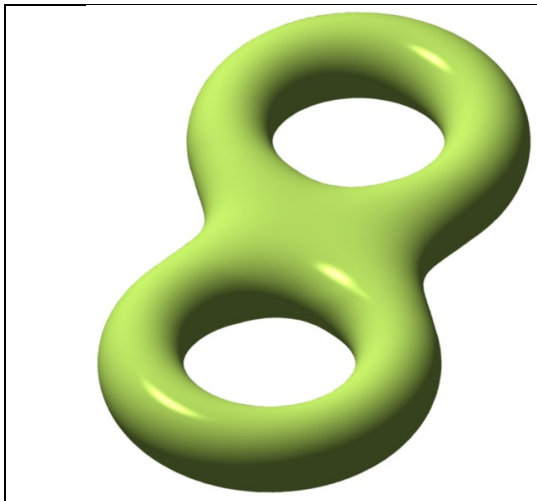
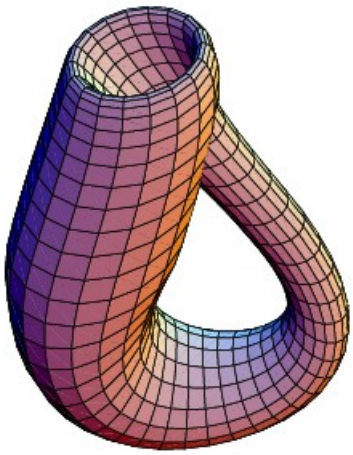
6.6.Eulerovy operátory a manifoldy

Výše popsaná tělesa, která jsou popsána, jako mnohostěny splňují jednu vlastnost, kterou jsme implicitně předpokládali, že hrana je sdílena sudým počtem ploch. Taková vlastnost se označuje pojmem *manifold*. Ve většině případů jde o *2-manifold*, tj. hranu sdílí právě 2 plochy. Pro mnohostěn bez děr pak platí Eulerův vztah, někdy též Eulerova rovnice:

$$F + V - E = \chi = 2$$

kde F je počet ploch (Face), V je počet vrcholů (Vertex), E je počet hran (Edge). Je nutné upozornit, že jde o implikaci, ne o ekvivalenci, tj. pokud platí uvedený vztah, pak o objektu nemůžeme tvrdit, že je to mnohostěn bez děr.

Eulerova charakteristika χ může nabývat hodnoty $\chi \neq 2$ pro jiná tělesa, např. pro:

	
Objekt s Genus-2 $\chi = -2$	Kleinova lahev - jednostranná plocha $\chi = 0$

Pokud objekt má díry, pak platí tzv. zobecněná Euler-Poincaré formule:

$$F + V - E = H + 2\,(S - G)$$

kde H je počet děr (Hole), S je počet spojených komponent (Shell) a G je genus objektu (Genus). Vedle Eulerovy formule jsou nad manifoldy dále definovány Eulerovy operátory, které definují, jak se změní Eulerovy vztahy při manipulaci, např. odebrání vrcholu apod. a slouží ke kontrole konzistentnosti geometrického modelu. Jdou zavedeny dva typy, a to:

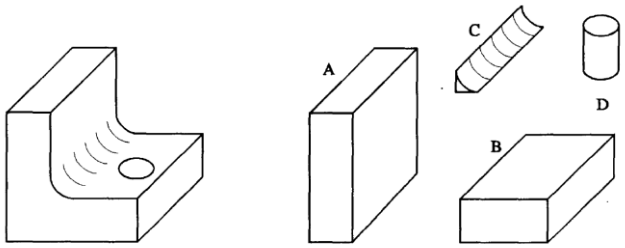
Name	**Description**	**Δ*V***	**Δ*E***	**Δ*F***	**Δ*H***	**Δ*S***	**Δ*G***
MBFLV	Make Body-Face-Loop-Vertex	1	0	1	0	1	0
MEV	Make Edge-Vertex	0	1	1	0	0	0
MEFL	Make Edge-Face-Loop	1	1	0	0	0	0
MEKL	Make Edge, Kill Loop	0	1	0	-1	0	0
KFLEVB	Kill Faces-Loops-Edges-Vertices-Body	-2	*-n*	*-n*	0	-1	0
KFLEVMG	Kill Faces-Loops-Edges-Vertices, Make Genus	-2	*-n*	*-n*	0	0	1

První písmeno operace M/K označuje akci Make/Kill, tj. vytvoř/zruš.

Podrobné info: viz Havemann,S.: Generative Mesh Modeling, PhD Thesis, 2005, (CLICK off-Line) http://www.generative-modeling.org/GenerativeModeling/Documents/Dissertation-Havemann_print.pdf

6.7.CSG stromy

Reprezentace objektů CSG (Constructive Solid Geometry) je poměrně jednoduchá a elegantní. Předpokládejme, že jsou dány základní objekty jako kvádr, koule, válec, jehlan, kužel atd. a z těchto objektů chceme realizovat složitější objekty pomocí množinových operací.



Obr.www: Tvořené těleso a použitá základní primitiva

Výsledné těleso T je tedy vlastně definováno“

$$T = (A \cup B \cup C) - D$$

nebo jako“

$$T = (B - D) \cup A \cup C$$

Je tedy zřejmé, že těleso:

- je vlastně reprezentováno stromem, který se nazývá CSG strom, kde listy stromu jsou základní objekty a uzly stromu reprezentují množinové operace
- může mít několik různých reprezentací, tj. CSG stromů, které ve výsledku definují stejné těleso

Předpokládejme, že povrch tělesa je reprezentován rovnicí:

$$F(\boldsymbol{x}) = 0$$

s tím, že $F(\boldsymbol{x}) < 0$ pro všechny body uvnitř tělesa, $F(\boldsymbol{x}) > 0$ pro body vně tělesa. Tedy funkce $F(\boldsymbol{x})$ definuje orientovanou plochu. Proto se také někdy tento postup nazývá funkcionální nebo modelování implicitními funkcemi apod.

Předpokládejme, že A je reprezentováno funkcí $F_1(\boldsymbol{x})$, B je reprezentováno funkcí $F_2(\boldsymbol{x})$. Pak základní operace nad tělesy lze pak definovat takto:

Název	Matematický symbol	Operátor C++	Matematický popis
Sjednocení	$A \cup B$	|	$F(\boldsymbol{x}) \stackrel{\text{def}}{=} \min\{F_1(\boldsymbol{x}), F_2(\boldsymbol{x})\}$
Průnik	$A \cap B$	&	$F(\boldsymbol{x}) \stackrel{\text{def}}{=} \max\{F_1(\boldsymbol{x}), F_2(\boldsymbol{x})\}$
Rozdíl	$A - B$	\	$F(\boldsymbol{x}) \stackrel{\text{def}}{=} F_1(\boldsymbol{x}) - F_2(\boldsymbol{x})$
Doplněk	$-$ A (unární)	$-$	$F(\boldsymbol{x}) \stackrel{\text{def}}{=} -F_1(\boldsymbol{x})$

Tab.XXX: Tabulka množinových operací

Je nutné si uvědomit, že tento způsob konstrukce objektů je poměrně velmi mocný. Uvažme objekt tvořený jako sjednocení dvou koulí. Pak je zřejmé, že:

- koule je reprezentována středem $\boldsymbol{x}_s$ a poloměrem R, tj. 4 hodnotami v pohyblivé řádové čárce. Pokud bychom reprezentovali povrch koule trojúhelníkovou sítí (pro jednoduchost předpokládejme generaci trojúhelníků pomocí sférických souřadnic), pak budeme potřebovat

cca $20\,000$ trojúhelníků (dělení v prostoru φ a ϑ na 100×100 bodů a dělení vzniklého čtyřúhelníka na 2 trojúhelníky). Tedy kompresní poměr je $20\,000 : 4 = 5000 : 1$.

- při realizaci sjednocení a i dalších operací vzniká na průsečíku ploch křivka. V případě trojúhelníkové sítě toto vede k následné „explozi“ malých trojúhelníků, neboť je nutné reprezentovat přesně povrch objektu na společné hranici daných objektů.
- V případě funkcionálního popisu a CSG stromů popis jednoduchý, kompaktní.
  funkční popis umožňuje realizaci i prostorově „neomezených“ objektů, jako poloprostory, „nekonečný“ válec, rotační paraboloid atd.

Je nutné upozornit, že gradient výsledné funkce není spojitý. Tento problém je fundamentální, neboť ani povrch není hladký. Proto se používají trochu složitější funkce, které „vyhlazují“ povrch v místě protínání, a tím pak i gradient funkce je spojitý. Tab.XXX ukazuje jedno možné řešení, kde koeficient α určuje „míru“ vyhlazení.

Celá řada systémů je založena na funkcionálním modelování s CSG stromy, např. systém Hyper-Fun viz http://cis.k.hosei.ac.jp/~F-rep/HF_descr.html

Název	Operátor	Matematický popis
Sjednocení	$A \cup B$	$F(\boldsymbol{x}) \stackrel{\text{def}}{=} \frac{1}{1+\alpha}\left(F_1(\boldsymbol{x}) + F_2(\boldsymbol{x}) + \sqrt{\lvert F_1^2(\boldsymbol{x}) + F_1^2(\boldsymbol{x}) - 2\alpha F_1(\boldsymbol{x})F_2(\boldsymbol{x})\rvert}\right)$
Průnik	$A \cap B$	$F(\boldsymbol{x}) \stackrel{\text{def}}{=} \frac{1}{1+\alpha}\left(F_1(\boldsymbol{x}) + F_2(\boldsymbol{x}) - \sqrt{\lvert F_1^2(\boldsymbol{x}) + F_1^2(\boldsymbol{x}) - 2\alpha F_1(\boldsymbol{x})F_2(\boldsymbol{x})\rvert}\right)$
Rozdíl	$A - B$	Realizuje se jako $A \cap (-B)$
Doplněk	$-$ A (unární)	$F(\boldsymbol{x}) \stackrel{\text{def}}{=} -F_1(\boldsymbol{x})$

Tab.XXX: Tabulka základních operací

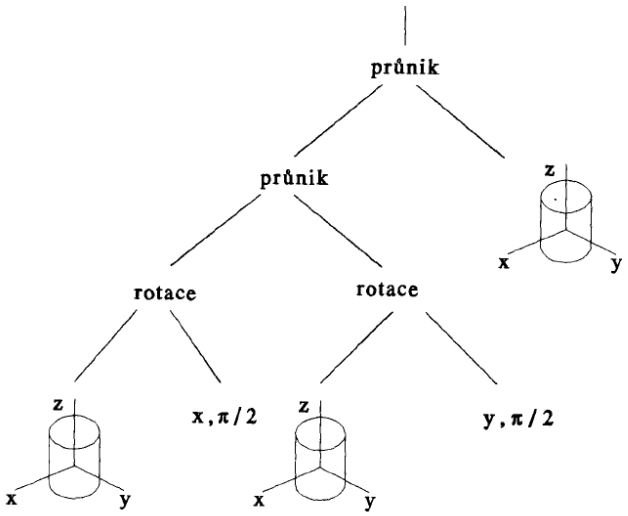
Zásadní výpočetní problém s funkcionální reprezentací je pak výpočet a zobrazení výsledného povrchu. Ve většině případů:

- prostor objektu se diskretizuje (obvykle uživatel musí specifikovat rozsah zobrazované oblasti) a použije se některá metoda extrakce iso-povrchu pro objemová data (metody pro zpracování CT, MRI dat), viz kap.16.1.1 (Vizualizace skalárních dat), nebo
- povrch se zobrazí metodou sledování paprsku (Ray Tracing), viz kap.13.1 (Metoda sledování paprsku) apod.

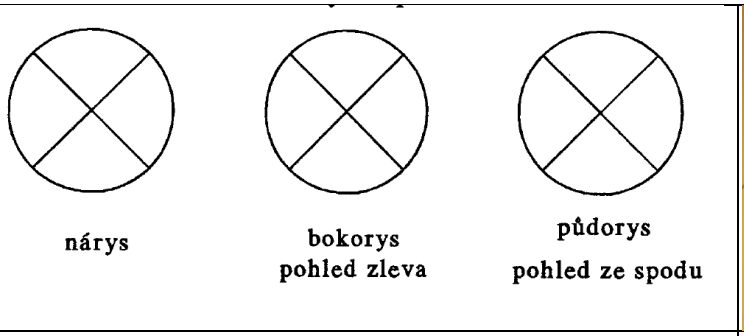
Pomocí výše uvedeného postupu můžeme konstruovat i velmi složité geometrické objekty, pokud umožníme do stromové struktury vkládat také geometrické operace, např. posuv, rotace apod.

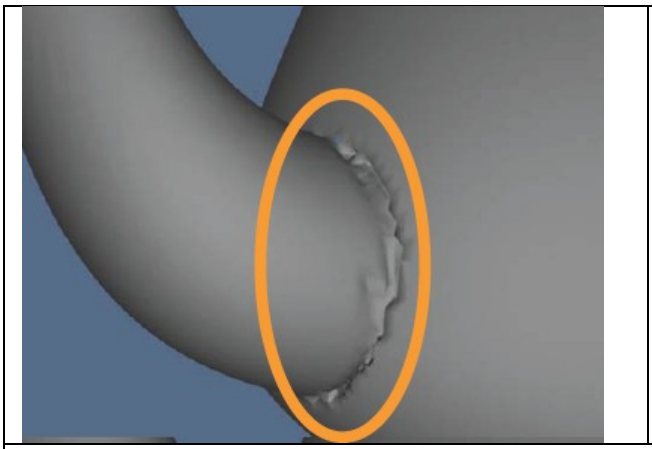
Příklad

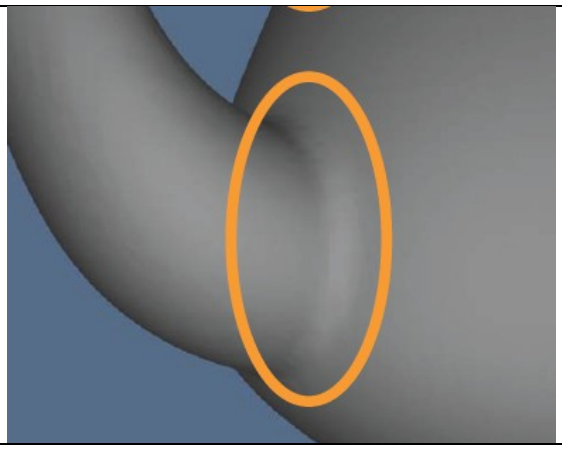
Úkolem je vytvoření objektu, který je určen jako průnik 3 válců o stejném průměru (dostatečně dlouhé nebo „nekonečné“), jejichž osy jsou navzájem na sebe kolmé. V případě použití trojúhelníkové sítě je zásadní problém volby velikosti trojúhelníků, které nahradí jednotlivé válce tak, aby průsečnice byla „dostatečně“ hladká. Při řešení je pak nutné očekávat „explozi“ generování malých trojúhelníků v místě průsečíku jednotlivých ploch. Geometrický objekt je popsán následujícím CSG stromem.



Pokud výsledné těleso zobrazíme v paralelní projekci, pak dostáváme následující pohledy. Z pohledu je zřejmé, že výsledné těleso by se mělo „koulet“, nicméně má jakési hrany. Nejde tedy o kouli, jak bývá prvotní odhad, ale o těleso, které se může „válet“ ve třech na sebe kolmých směrech.







Viz http://www.cad.zju.edu.cn/home/zhx/GM/012/00-ism.pdf

Nezbytnou součástí každého geometrického elementu u CSG stromů je informace o minimálním intervalu, resp. minimálním ohraničujícím objemu, který daný objekt zahrnuje. Je nutné zdůraznit, že např. pro rotační paraboloid v základní poloze je interval v jednom směru $\langle 0, \infty)$, interval může obecně být $(-\infty, +\infty)$.

Optimalizace CSG stromů

Při realizaci složitých objektů je nutné mít na paměti, že dva různé CSG stromy mohou reprezentovat stejný objekt. To znamená, že různí uživatelé mohou vytvořit pro stejné zadání různé CSG stromy. Nicméně efektivita při výpočtech a zobrazování je zásadně ovlivněna pořadím jednotlivých operací. Navíc při zobrazování povrchu objektu je nutné se omezit na prostor, ve kterém se objekt nalézá.

Z výše uvedeného vyplývá, že CSG strom je vhodné transformovat tak, aby operace průniku byly co nejblíže k listům CSG stromu a naopak operace sjednocení co nejvýše, tj. ke kořeni stromu.

Množinové operace se aplikují také na ohraničující objekty a může dojít k situacím:

- při operaci průniku:
  - průnik ohraničujících objektů je prázdný, pak i odpovídající podstrom může být ze struktury vypuštěn a jde pravděpodobně o chybový stav
  - obalové objekty podřízených podstromů mají neprázdný průnik a mohou se rozpadnout na množinu geometricky menších obalových objektů
- při operaci sjednocení:
  - obalové objekty mají prázdný průnik – výsledek operace sjednocení „zdědí“ původní obalové objekty
  - obalové objekty mají neprázdný průnik – je vhodné je redukovat, resp. nahradit tak, aby výsledná množina obalových objektů byla prostorově co nejmenší
- a mnoho dalších

Je zřejmé, že optimalizací CSG stromů lze docílit podstatné úspory ve výpočetních nárocích a vyplatí se tedy optimalizaci CSG stromu realizovat.

V mnoha případech, které jsou řešeny pomocí průsečíku přímky s objektem, se přímka vyjadřuje v parametrické formě a následně se pracuje s intervaly parametrů, se kterými se dělají množinové operace, např. průnik, sjednocení atd. Tohoto postupu se zejména využívá při aplikaci metody sledování paprsku.

Reference

Langetepe,E., Zachmann,G.: Geometric Data Structures for Computer Graphics, A.K.Peters, 2006

Pro přenos geometrických dat se používá mnoho formátů, mnohé z nich byly standardizovány. Jedním ze základních a jednoduchých formátů je formát STL.

6.8.STL formát

STL formát (STereoLithography) je formát používaný v CAD-CAM systémech a v systémech, které se označují termínem „rapid prototyping“ pro účely 3D tisku. Formát STL zachycuje pouze geometrii 3D objektů bez ostatních atributů jako je barva, textura apod. STL formát je definován jak ve formě textových dat, tj. *ASCII STL*, tak i pro binární data, tj. *binary STL*. Povrch objektu je definován množinou samostatných trojúhelníků, tedy nikoliv trojúhelníkovou sítí, posloupností dat pro každý trojúhelník, a to:

normálový vektor – 3 hodnoty
    $vrchol_1$ – 3 hodnoty souřadnicové
    $vrchol_2$ – 3 hodnoty souřadnicové
    $vrchol_3$ – 3 hodnoty souřadnicové
    :
normálový vektor – 3 hodnoty
    $vrchol_1$ – 3 hodnoty souřadnicové
    $vrchol_2$ – 3 hodnoty souřadnicové
    $vrchol_3$ – 3 hodnoty souřadnicové

Orientace normálového vektoru by vždy měla být taková, že normála plochy směřuje ven z objektu.

Některé nové modifikace STL formátu umožňují zahrnutí i atributů jako je barva, průsvitnost, což je důležité zejména pro 3D tisk, kde se barva již běžně používá.

Základní nevýhody STL formátu (podle původní definice standardu) jsou:

- všechny souřadnice vrcholů musí být větší než nula
- jde o množinu samostatných trojúhelníků, takže není žádná informace o sousedních trojúhelnících ani o trojúhelnících sdílející daný vrchol
- souřadnice vrcholů se vyskytují mnohonásobně, neboť se vrcholy uvádějí pro každý trojúhelník zvlášť
- je obtížná kontrola, zda objekt je konzistentní a zda plocha je uzavřená, tj. že tvoří objem
- náchylnost k nekonzistenci vlivem nepřesných dat v souřadnicích vrcholů
- neobsahuje žádné informace o měřítku, jednotkách ([mm] x [inch]) atd.

Přes všechny nevýhody je STL formát používaný pro svoji jednoduchost. Tento formát je postupně nahrazován formátem PLY. Formát PLY byl vyvinut na Stanford University pro zpracování dat velkého rozsahu z 3D skenování sochy Davida a následnou rekonstrukci povrchu. Tento formát je znám též pod názvy „Polygon File Format“ nebo „Stanford Triange Format“,
viz http://en.wikipedia.org/wiki/PLY_(file_format).

6.9.Algoritmy výpočtu průsečíků a ohraničující tělesa

V geometrických algoritmech se velmi často používá výpočet průsečíků geometrických entit, např. přímka s rovinou, přímka s koulí, trojúhelník s trojúhelníkem atd. Pro urychlení výpočtu se používají různé techniky k urychlení detekce, zda průsečík existuje. K tomuto slouží jednoduchá obalová tělesa jako je kvádr nebo koule. V případě složitých objektů, resp. dlouhých objektů, se používá více ohraničujících těles, přičemž mohou mít obdobnou strukturu jako CSG stromy. V převážné většině je výpočet průsečíku založen na parametrickém tvaru přímky a na implicitním rovnici definující danou entitu.

V následujícím budou ukázány základní techniky pro detekci průsečíku s přímkou.

Průsečík přímky s rovinou

Přímka je dána v parametrickém tvaru, tj.:

$$\boldsymbol{X}(t) = \boldsymbol{X}_A + \boldsymbol{S}\, t$$

a rovnice roviny ρ:

$$aX + bY + cZ + d = 0$$

tj.

$$\boldsymbol{a}^T\boldsymbol{X} + d = 0$$

Dosazením rovnice přímky do rovnice roviny dostáváme:

$$\boldsymbol{a}^T(\boldsymbol{X}_A + \boldsymbol{S}\, t) + d = 0$$

pak průsečík přímky s rovinou je určen hodnotou parametru t:

$$t = -\frac{\boldsymbol{a}^T\boldsymbol{X}_A + d}{\boldsymbol{a}^T\boldsymbol{S}}$$

Pokud přímka je kolmá na normálu roviny, pak $t = \pm\infty$. To znamená, že je zde numerický problém a otázka stability výpočtu. Zkusme analyzovat, jak by řešení vypadalo v případě reprezentace v projektivním prostoru. Přímka je nyní dána rovnicí:

$$\boldsymbol{x}(t) = \boldsymbol{x}_A + \boldsymbol{s}\,\tau$$

kde $\boldsymbol{x}_A = [x_A, y_A, z_A : w_A]^T$ a $\boldsymbol{s} = \left[s_x, s_y, s_z : s_w\right]^T$.

Rovnice roviny ρ v projektivním prostoru je dána (původní rovnici vynásobíme $w \neq 0$):

$$ax + by + cz + dw = 0$$

tj. :

$$\boldsymbol{a}^T\boldsymbol{x} = 0$$

Vidíme tedy, že jde o více kompaktní formu i z hlediska datové reprezentace. Pak průsečík je určen:

$$\boldsymbol{a}^T(\boldsymbol{x_A} + \boldsymbol{s}\,\tau) = 0$$

a tedy:

$$\tau = -\frac{\boldsymbol{a}^T\boldsymbol{x_A}}{\boldsymbol{a}^T\boldsymbol{s}}$$

Hodnotu parametru pro průsečík můžeme reprezentovat přímo v homogenních souřadnicích a pak:

$$\boldsymbol{\tau} \triangleq [-\boldsymbol{a}^T\boldsymbol{x_A} : \boldsymbol{a}^T\boldsymbol{s}]^T$$

Je zřejmé, že v případě projektivní reprezentace není problém s operací dělení ani se stabilitou výpočtu, neboť operace dělení je „odložena“ včetně rozhodnutí o singularitě.

V následujícím textu budou použity pouze Eukleidovské souřadnice, modifikace pro projektivní prostor jsou jednoduchá.

Průsečík přímky s kvádrem (AABB - Axis Aligned Bounding Box)

Algoritmus průsečíku přímky s kvádrem, jehož hrany jsou rovnoběžné s osami souřadného systému je velmi jednoduchý, neboť je o vlastně o Cyrus-Beck algoritmus v extrémní podobě, kdy rovnice jednotlivých rovin jsou ve tvaru:

$x = \pm a$	nebo	$y = \pm b$	nebo	$z = \pm c$

a přímka je dána v parametrickém tvaru:

$$\boldsymbol{x}(t) = \boldsymbol{x}_A + (\boldsymbol{x}_B - \boldsymbol{x}_A)\, t$$

V případě algoritmu sledování paprsku, viz kap.13.1 (Metoda sledování paprsku), pak hledáme jen hodnotu t_{min}. AABB box má, přes svoji extrémní výpočetní jednoduchost, zásadní nevýhodu a to, že není rotačně invariantní a není vhodný pro dlouhé štíhlé objekty v obecné poloze. Proto se někdy přímka transformuje do polohy, kdy je objekt v základní poloze, resp. v poloze, která umožňuje, aby AABB byl co nejmenší.

Ohraničující koule (Bounding sphere)

Koule jako ohraničující těleso má výhodu v tom, že je rotačně invariantní. Předpokládejme, že máme přímku p v Eukleidovském prostoru E^3 danou rovnicí

$$\boldsymbol{x}(t) = \boldsymbol{x}_A + (\boldsymbol{x}_B - \boldsymbol{x}_A)\, t = \boldsymbol{x}_A + \boldsymbol{s}\, t$$

a povrch koule je dán vztahem:

$$(\boldsymbol{x} - \boldsymbol{x}_s)^T(\boldsymbol{x} - \boldsymbol{x}_s) - r^2 = 0$$

Dosazením výrazu pro přímku do rovnice pro kouli pak dostáváme:

$$[\boldsymbol{x}_A + \boldsymbol{s}\, t - \boldsymbol{x}_s]^T[\boldsymbol{x}_A + \boldsymbol{s}\, t - \boldsymbol{x}_s] - r^2 = 0$$

a tedy:

$[\boldsymbol{s}t + \boldsymbol{\xi}]^T[\boldsymbol{s}t + \boldsymbol{\xi}] - r^2 = 0$	kde	$\boldsymbol{\xi} = \boldsymbol{x}_A - \boldsymbol{x}_s$

tedy kvadratickou rovnici a její řešení:

$$\boldsymbol{s}^T\boldsymbol{s}\, t^2 + 2\boldsymbol{s}^T\boldsymbol{\xi} + \boldsymbol{\xi}^T\boldsymbol{\xi} - r^2 = 0$$

$at^2 + bt + c = 0$	$t_{1,2} = \dfrac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

Je zřejmé, že pokud:

$$b^2 - 4ac < 0$$

pak průsečík a ani dotyk přímky s koulí neexistuje. Pokud se počítá větší počet průsečíků se stejnou přímkou, je výpočetně výhodné vektor $\boldsymbol{s}$ normalizovat.

Úloha

Porovnejte výpočetní náročnost standardního algoritmu s postupem, kdy se scéna transformuje tak, že koule má střed v počátku, tj. $\boldsymbol{x}_s = \boldsymbol{0}$, vektor $\boldsymbol{s}$ je normalizován, tj. $\|\boldsymbol{s}\| = 1$. Optimalizujte detekci průsečíku přímky s koulí. Zkuste odpovědět na otázku, zda, např. pro metodu sledování paprsku, není výhodnější všechna tělesa reprezentovat v základní poloze a počítat průsečíky objektů, resp. obalových těles, s přímkou, která je odpovídajícím způsobem transformována.

Dosud byly popisovány základní datové struktury pro reprezentaci objektů. Avšak v počítačové grafice se používají k zobrazení ještě další entity, atributy atd. Navíc scéna se obvykle skládá z více objektů, světelných zdrojů atd. To znamená, že potřebujeme reprezentovat zobrazovanou scénu.

6.10. Hashing a eliminace duplicit

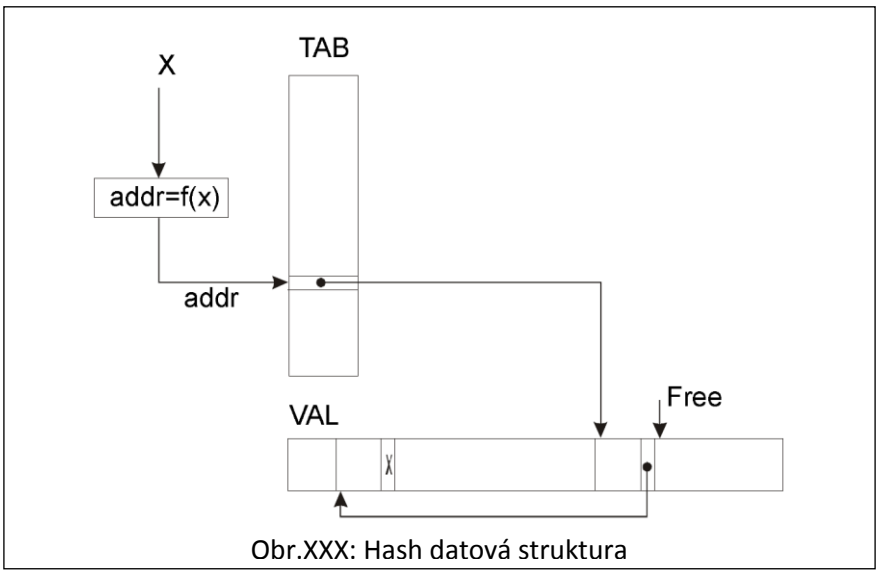
V počítačové grafice se jako paměťové struktury používají vícerozměrná pole, stromy, fronty a zásobníky, seznamy atd. Jednou z opomíjenou datovou strukturou je hashing, které se velmi často používá při zpracování textu. Hash datová struktura má výhodu, že v ideálním případě, tzv. perfect hashing, je uložení hodnoty a její nalezení složitosti $O(1)$. Zkusme se podívat, jaké jsou rozdíly mezi textovými data a daty geometrickými.

		Dimenzionalita	
		Malý	„Nekonečný“
Interval hodnot	Malý		Textová data Dimenze, tj. délka řetězce dlouhá Interval hodnot, např. 256 pro ASCII
	„Nekonečný“	Geometrická data Dimenze 2, 3 Interval hodnot $(-\infty, +\infty)$	

Tab.xx: Rozdílnost textových a geometrických dat

- textová data – rozsah hodnot je dán použitou alfabetou, např. 256 pro ASCII, zatímco dimenzionalita je „nekonečná“, neboť řetězce mohou být velmi dlouhé, např.
  - protein titin je popsán 189,819 znaky
  - železniční stanice *Llanfairpwllgwyngyllgogerychwyrndrobwllllantysiliogogogoch* ve Walesu atd.

  počet slov, tj. zpracovávaných elementů je např. pro český slovník cca 2-3 mil. slov.
- geometrická data – obvykle mají jen 2, resp. 3 souřadnice $\langle x, y \rangle$, resp. $\langle x, y, z \rangle$, ale „nekonečný“ interval hodnot $(-\infty, +\infty)$, přičemž běžně zpracováváme $10^6 - 10^{15}$ hodnot



Obr.XXX: Hash datová struktura

Princip hashování je poměrně jednoduchý. Z dané hodnoty (klíče v textovém pojetí“) se pomocí hashovací funkce určí adresa, kam se hodnota uloží. Je tedy zřejmé, že klíčovou otázkou je návrh hashovací funkce, která by neměla pro dvě různé hodnoty dávat stejnou adresu do hashovací tabulky. V případě geometrických dat však máme souřadnice x, y, z a interval hodnot je „nekonečný“. Standardní hashovací funkce obvykle využívají operaci $\boldsymbol{mod}$, která je vlastně kompozicí násobení, dělení a převodu na celé číslo:

$$a \; \boldsymbol{mod} \; p = a - p * \left\lfloor \frac{a}{p} \right\rfloor$$

Lze ukázat, že dobrou hashovací funkcí pro geometrická data je funkce:

$f(x,y,z) = h$ **and** (2^{q-1})	$h = C(\alpha x + \beta y + \gamma z)$ **and** $\left(2^{k-1}\right)$

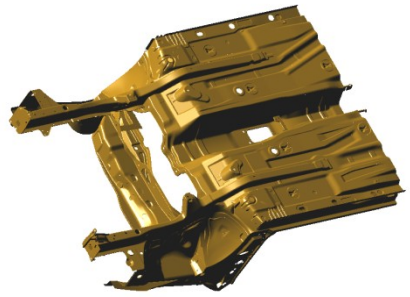
kde: α, β, γ jsou malé iracionální hodnoty, např. $\frac{1}{3}, \pi, e, \sqrt{2}, \sqrt{3}$, C je měřítková konstanta závislá na předpokládaném rozsahu hodnot, např. v zpracovávaném datovém setu, k je hardwarově hodnota závislá konstanta (32/64 bitová reprezentace pro integer a počet bitů mantisy), q určuje délku hashovací tabulky, která je 2^q.

Algoritmus je založen na tom, že se z hodnot x, y, z určí adresa do „virtualní“ tabulky $h \in \langle 0, 2^{k-1} \rangle$. Tento interval se pak mapuje na interval adres hashovací tabulky.

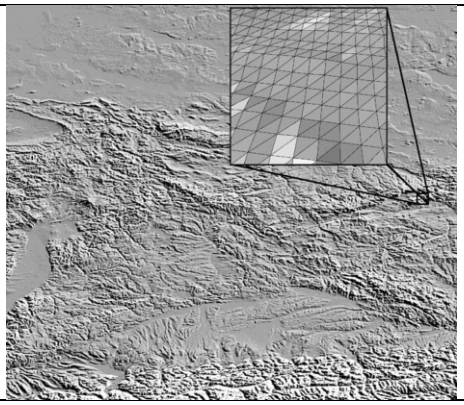
Podrobně viz:

Hradek,J., Skala,V.: Hash Function and Triangular Mesh Reconstruction, Vol.29, No.6., pp.741-751, Computers&Geosciences, Pergamon Press, ISSN 0098-3004, 2003

Jako příklad uveďme body podvozku automobilu, viz obr.XXX

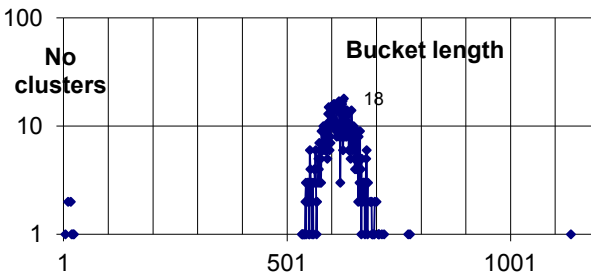


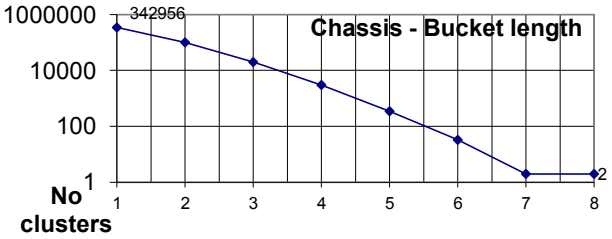
Obr.XXX: Datový set podvozku automobilu



Obr.ZZZ: reprezentace povrchu Země

Pro efektivní činnost hashování je nutné analyzovat délku vznikajících seznamů, viz obr.XXX





„Standardní“ hashing funkce

Obr.XXX: Porovnání „standardní“ a výše popsané hash funkce

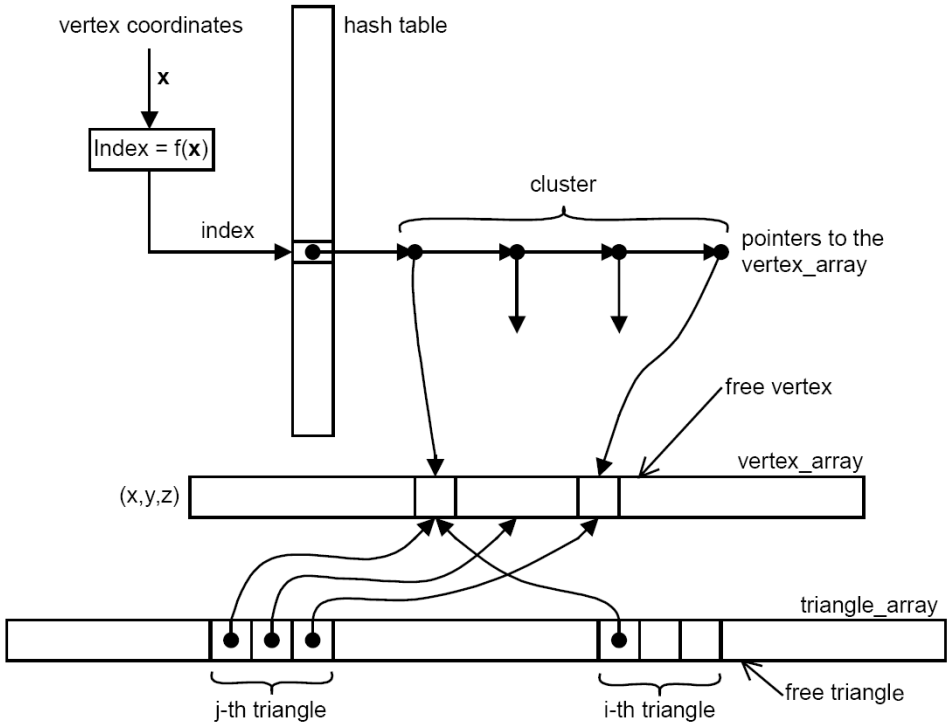
Z histogramu délky seznamu zřetězených hodnot, viz obr.XXX, je zřejmé, že v případě „standardní“ hash funkce musíme provést cca 600 dotazů, zatímco s popsanou hash funkcí podstatně méně, jen cca 2-4, což znamená podstatné urychlení.

Hashování lze velmi dobře použít k odstranění duplicit v datovém setu. Při běžném přístupu algoritmus zřejmě bude $O(n^2)$, nebo při použití třídění $O(n \, lg \, n)$. Při použití hashování pak můžeme dosáhnout $O_{expected}(n)$.

6.11. Rekonstrukce trojúhelníkové sítě z množiny trojúhelníků

V praxi je velmi častý případ, kdy z množiny samostatných trojúhelníků potřebujeme vytvořit trojúhelníkovou síť s použitím např. okřídlené hrany kap.6.5 (Hraniční reprezentace). Toto je už poněkud komplikovanější úloha, neboť musíme uvážit to, že počet zpracovávaných trojúhelníků bude běžně $n \in \langle 10^5, 10^8 \rangle$. a počet vrcholů $m = 3n$. Vytvořením trojúhelníkové sítě, kdy je každý vrchol uložen pouze jednou, podstatně zredukujeme paměťové nároky.

Princip je poměrně jednoduchý. Pro každý trojúhelník se uloží do tabulky souřadnice každého vrcholu trojúhelníka, pokud tam ještě nebyl. Nyní máme identifikaci, kde jsou všechny tři souřadnice trojúhelníka uloženy. Do seznamu trojúhelníků se uloží informace o vrcholech a k vrcholům se přidá informace na právě zpracovaný trojúhelník. Po zpracování všech trojúhelníků se dopočtou všechny ostatní požadované vazby.



Obr.QQQ: Datová struktura pro rekonstrukci trojúhelníkové sítě

Úlohu rekonstrukce trojúhelníkové sítě z množiny samostatných trojúhelníků už není možno realizovat modifikací algoritmu pro eliminaci duplicit se složitostí $O(n \, lg \, n)$. Takže mámě nyní pouze možnost algoritmu brutální síly se složitostí $O(n^2)$ nebo použití hashování se složitostí $O_{expected}(n)$. Je zřejmé, že algoritmus se složitostí $O(n^2)$ je absolutně nepoužitelný vzhledem ke zpracovávanému počtu trojúhelníků.

Podrobně viz:

- Hradek,J., Skala,V.: Hash Function and Triangular Mesh Reconstruction, Vol.29, No.6., pp.741-751, Computers&Geosciences, Pergamon Press, ISSN 0098-3004, 2003
- Glassner, A., 1994. Building vertex normals from an unstructured, polygon list. In: Graphic Gems IV. Academic Press, New York, pp. 60–73.

7. Reprezentace scény

Zobrazovaná scéna v reálné situaci neobsahuje jen popis geometrických objektů, ale též další informace, např.:

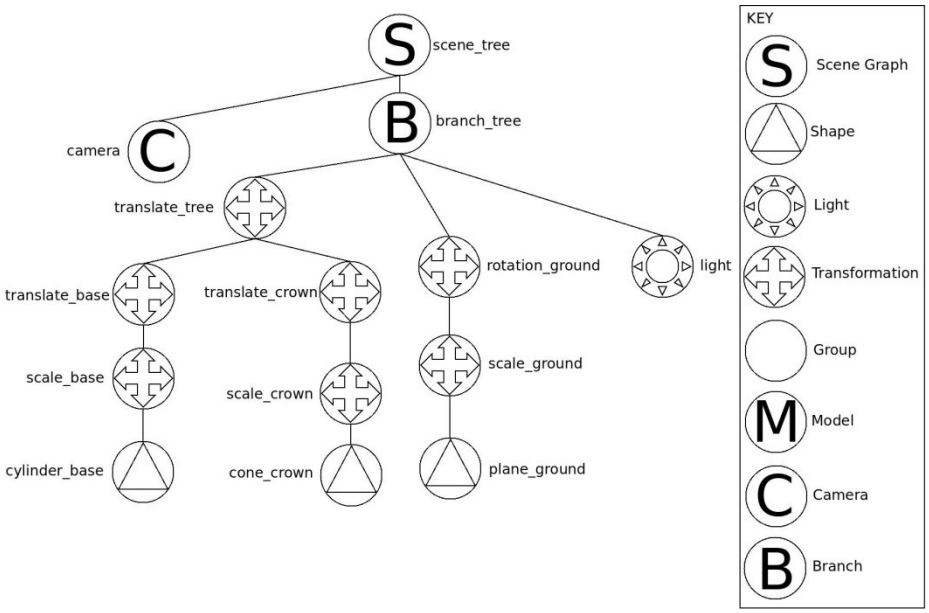
- pozice kamery, resp. pozorovatele, její směrovou orientaci, vlastnosti kamery, např. ohnisková vzdálenost atd.
- světelné poměry na scéně, např. pozice a charakteristika světelných zdrojů, případná fixace světelného zdroje na objekt, nebo skupinu objektů, kdy se při geometrické transformaci mění s objektem i pozice zdroje světla
- hierarchickou strukturu geometrie scény a atributy jednotlivých geometrických objektů, např. textury, průhlednost

Graf scény umožňuje efektivně budovat komplikované scény, jednoduchou editaci jejich částí a manipulaci s objekty ve scéně.

7.1. Graf scény

Tyto vlastnosti jsou zaznamenány ve struktuře, která se nazývá graf scény (*scene graph*) a je primárně tvořena stromovou strukturou, podobně jako u CSG stromu, avšak geometrické objekty mohou být definovány i jiným než funkcionálním popisem. Základní vlastností grafu scény je pak to, že jak transformace, tak i atributy atd. se „dědí“ na podřízené uzly stromu, pokud není stanoveno jinak.

Pro ilustraci uvažme jednoduchý graf scény (viz http://www.disgruntledrats.com/?p=266):



Obr.XXX: Ukázka grafu scény

7.2.Reprezentace grafu scény

Graf scény pak může být zapsán ve formě XML, např.:

```
<?xml version="1.0"?>
<scenegraph name="scene_tree">
      <branch name="branch_tree">
            <translation name="translate_tree" x="200.0" y="0.0" z="-100.0">
                  <translation name="translate_base" x="0.0" y="5.0" z="0.0">
                        <scale name="scale_base" x="5.0" y="5.0" z="5.0">
                              <shape name="cylinder_base" type="cylinder" color="brown" />
                        </scale>
                  </translation>
                  <translation name="translate_crown" x="0.0" y="15.0" z="0.0">
                        <scale name="scale_crown" x="15.0" y="15.0" z="15.0">
                              <shape name="cone_crown" type="cone" color="green" />
                        </scale>
                  </translation>
            </translation>
            <rotation name="rotation_ground" ray_x="1" ray_y="0" ray_z="0"
                  angle="90">
                  <scale name="scale_ground" x="1000.0" y="1000.0" z="1000.0">
                        <shape name="plane_ground" type="plane" color="gray" />
                  </scale>
            </rotation>
            <directional_light_source name="light" x="20.0"
                  y="300.0" z="20.0" />
      </branch>
      <camera name="camera" cop_x="0" cop_y="0" cop_z="0" vrp_x="20"
            vrp_y="150" vrp_z="20" vuv_x="0" vuv_y="1" vuv_z="0" />
</scenegraph>
```

Zásadní výhody grafu scény a jeho použití:

- je především snadná realizace úprav scény, tedy možnost editace, i pro velmi rozsáhlé scény
- možnost „klonování“ skupin objektů, tedy vlastně podstromů
- jednoznačný popis scény a možnost optimalizace grafu scény podobně jako u CSG stromů

Až dosud jsme se zabývali víceméně geometrickými vlastnostmi geometrických objektů s relevantním matematickým popisem. Při zobrazování scén je nutné brát v úvahu také světelné poměry ve scéně.

8. Světlo a barevné systémy

Použít

Skala,V.: Světlo, barvy a barevné systémy v počítačové grafice, Academia, 1991

8.1.Radiometrie, rovnice přenosu, fotometrie
8.2.Achromatické světlo
8.3.Chromatické světlo
8.4.RGB, CIE-uv, Lab, HLS,HSV, Opponent color space
8.5.Barevné vzorníky

9. Interpolace a aproximace uspořádaných a neuspořádaných dat

Interpolace a aproximace dat je opět jednou z velmi používaných oblastí v počítačové grafice. Obecně se pojmem interpolace označují ty metody, kdy výsledný interpolant zadanými body prochází. Např. u Bézierovy křivky interpolovaná křivka prochází body $\boldsymbol{x}_0$, $\boldsymbol{x}_{n-1}$, zatímco body $\boldsymbol{x}_1, \dots, \boldsymbol{x}_{n-2}$ jsou řídící body, které určují chování křivky. Pojmem aproximace se označují ty metody, kdy výsledný interpolant zadanými body nemusí procházet, ale dané hodnoty aproximuje podle zadaného kritéria, např. metodou nejmenších čtverců. Někdy se obě skupiny označují jako interpolace, avšak chápáno v širším smyslu. V následujícím budou probrány základní metody interpolací a aproximací s ohledem na aplikace v počítačové grafice a vizualizaci dat.

Asi ve všech publikacích pojednávajících o interpolacích se uvádí Lagrangeova interpolace. Předpokládejme, že máme $n+1$ navzájem různých bodů $x_0, \dots, x_n$ a dané hodnoty v těchto bodech, tj. $f(x_0), \dots, f(x_n)$. Pak Lagrangeův interpolační polynom je určen:

$$L_n(x) = f(x_0)l_0(x) + \cdots + f(x_n)l_n(x)$$

kde:

$$l_i(x_j) = \begin{cases} 1 \text{ pokud } i = j \\ 0 \text{ pokud } i \neq j \end{cases}$$

Tyto podmínky např. splňuje polynom:

$$l_i(x) = \prod_{\substack{0 \le k \le n \\ k \neq i}} \frac{x - x_0}{x_i - x_k} = \frac{x - x_0}{x_i - x_0} \dots \frac{x - x_{i-1}}{x_i - x_{i-1}} \frac{x - x_{i+1}}{x_i - x_{i+1}} \dots \frac{x - x_n}{x_i - x_n}$$

Vzhledem k výpočetní náročnosti se využívá rovnosti, a to:

$$L_n(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n = \sum_{i=0}^{n} a_i x^i = 1$$

ke konstrukci matice $\boldsymbol{\Lambda}_{\mathrm{n+1 \times n+1}}$, reprezentující Lagrangeův polynom. Pak lze koeficienty a_i vypočítat řešením soustavy lineárních rovnic:

$$\boldsymbol{\Lambda\alpha} = \boldsymbol{\beta}$$

kde:

$\boldsymbol{\Lambda} = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_i & x_i^2 & \cdots & x_i^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{bmatrix}$	$\boldsymbol{\alpha} = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \\ a_n \end{bmatrix}$	$\boldsymbol{\beta} = \begin{bmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_{n-1}) \\ f(x_n) \end{bmatrix}$

a kde matice $\boldsymbol{\Lambda}$ je známá Vandermondova matice, viz Alexandre Théophile Vandermonde.

Nicméně je nutné poznamenat, že:

- pro větší počet bodů může nastat problém s numerickým řešením soustavy rovnic
- základní vlastností Lagrangeovy interpolace je její charakteristické „rozkmitávání“ se s rostoucím počtem bodů

Vzdálenou podobnost s výše uvedeným postupem pak bude u aproximace pomocí radiálních bázových funkcí (RBF aproximace).

Osobní poznámka – přímý dopad do běžného života:

Čím více zákonů se zavede, které lidé musejí respektovat, tím větší je rozkmitávání chování lidí, tj. chaos.

9.1.Lineární interpolace

Lineární interpolace je zřejmě nejčastější v aplikacích počítačové grafiky. V zásadě jednotlivá interpolační schémata vycházejí buď z explicitní nebo implicitní rovnice přímky, roviny apod., nebo z parametrického tvaru. V následujícím se budeme zabývat parametrickou lineární interpolací.

Lineární interpolace

Předpokládejme, že jsou dány dva body v E^n, a to $\boldsymbol{X}_1$ a $\boldsymbol{X}_2$. Pak lineární interpolace, tj. „na přímce“, je dána vztahem:

$$\boldsymbol{X}(t) = \boldsymbol{X}_1 + (\boldsymbol{X}_2 - \boldsymbol{X}_1)\, t$$

kde $t \in \langle 0,1\rangle$ je parametr. Výše uvedený vztah se používá pro parametrické vyjádření přímky, resp. polopřímky nebo paprsku (ray), kdy parametr $t \in (-\infty, +\infty)$, resp. $t \in \langle 0,\infty)$. Parametrického vyjádření přímky se s výhodou používá pro výpočet průsečíku přímky s rovinou apod.

V případě lineární interpolace „na rovinné ploše“ v prostoru E^n (v námi uvažovaných aplikacích je to téměř výlučně prostor E^3) je interpolace určena vztahem:

$$\boldsymbol{X}(u,v) = \boldsymbol{X}_1 + (\boldsymbol{X}_2 - \boldsymbol{X}_1)\, u + (\boldsymbol{X}_3 - \boldsymbol{X}_1)\, v$$

kde u, v jsou parametry. Výše uvedený vztah se používá pro parametrické vyjádření trojúhelníka, kde pro parametry platí $u, v \in \langle 0,1\rangle \;\&\; 0 \leq u + v \leq 1$.

V mnoha aplikacích se používají barycentrické souřadnice:

- bodu na úsečce v E^1, kde určují poměr délek
- v trojúhelníku v E^2, kde určují poměr ploch
- ve čtyřstěnu (tetrahedronu) v E^3, kde určují poměr objemů

Je nutné upozornit, že barycentrické souřadnice bodu v trojúhelníku, který je v obecné poloze v E^3 nejsou přímo definovány, neboť souřadnice jsou ještě „svázány“ rovinou, na které daný trojúhelník leží. Toto se většinou řeší projekcí do nějaké základní roviny souřadného systému.

Barycentrické souřadnice bodu na úsečce

Barycentrické souřadnice jsou definovány takto:

$$\lambda_1 X_1 + \lambda_2 X_2 = X$$
$$\lambda_1 + \lambda_2 = 1$$

Pokud dosadíme $\lambda_1 = 1 - \lambda_2$ do rovnice, pak:

$$X_1 + \lambda_2 (X_2 - X_1) = X$$

Vidíme tedy přímou souvislost mezi lineární interpolací a barycentrickými souřadnicemi a platí

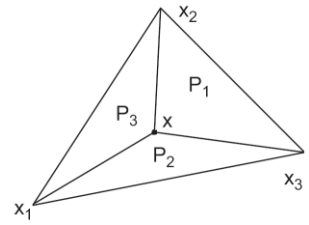
$$t = \lambda_2$$

Barycentrické souřadnice bodu v trojúhelníku

Barycentrické souřadnice jsou dány vztahy:

$$\lambda_1 X_1 + \lambda_2 X_2 + \lambda_3 X_3 = X$$
$$\lambda_1 Y_1 + \lambda_2 Y_2 + \lambda_3 Y_3 = Y$$
$$\lambda_1 + \lambda_2 + \lambda_3 = 1$$

Pokud dosadíme $\lambda_1 = 1 - \lambda_2 - \lambda_3$ do rovnice, pak opět dostaneme lineární interpolaci na parametrické rovině $\boldsymbol{X}(u,v)$. Barycentrická souřadnice λ_1 pak určuje poměr plochy trojúhelníka P_1, jehož vrcholem je daný bod $\boldsymbol{X}$ a plochy celého trojúhelníka.

$\lambda_1 = \frac{P_{X\ \ x_2x_3}}{P_{x_1x_2x_3}}$	$\lambda_2 = \frac{P_{x_1x\ \ x_3}}{P_{x_1x_2x_3}}$	$\lambda_3 = \frac{P_{x_1x_2x}}{P_{x_1x_2x_3}}$	

Je zřejmé, že určení hodnot barycentrických souřadnic vede na řešení soustav lineárních rovnic $\boldsymbol{Ax} = \boldsymbol{b}$.

Výše uvedená lineární parametrická interpolace a interpolace barycentrickými souřadnicemi v Eukleidovském prostoru E^n je lineární vzhledem k parametru, resp. parametrům.

V počítačové grafice se používají homogenní souřadnice, tj. používáme projektivní rozšíření Eukleidovského prostoru. Otázkou tedy je, jak postupovat, pokud souřadnice bodů jsou dány v homogenních souřadnicích.

Interpolace v projektivním prostoru

Předpokládejme, že body jsou dány souřadnicemi v homogenních souřadnicích, a to:

- $\boldsymbol{x} = [x : w]^T$ v případě E^1
- $\boldsymbol{x} = [x, y : w]^T$ v případě E^2
- $\boldsymbol{x} = [x, y, z : w]^T$ v případě E^3

a pro homogenní souřadnici *nemusí platit*, že $w = 1$. Obecně je možné převést souřadnice bodů do Eukleidovského prostoru, tj. vydělit souřadnice hodnotou homogenní složky w a následně použít lineární interpolaci nebo barycentrické souřadnice. Toto řešení je poměrně výpočetně náročné, neboť se musí použít operace dělení. Otázkou je, zda je možné realizovat lineární interpolaci bez použití operace dělení.

Předpokládejme, že jsou dány dva body $\boldsymbol{x}_1$ a $\boldsymbol{x}_2$ v homogenních souřadnicích. Pak zajisté je možné:

$$\boldsymbol{x}(t) = w_2\boldsymbol{x}_1 + (w_1\boldsymbol{x}_2 - w_2\boldsymbol{x}_1)\, t$$

neboť homogenní souřadnice reprezentuje „odložené“ dělení. Pak pro případ E^1 můžeme psát:

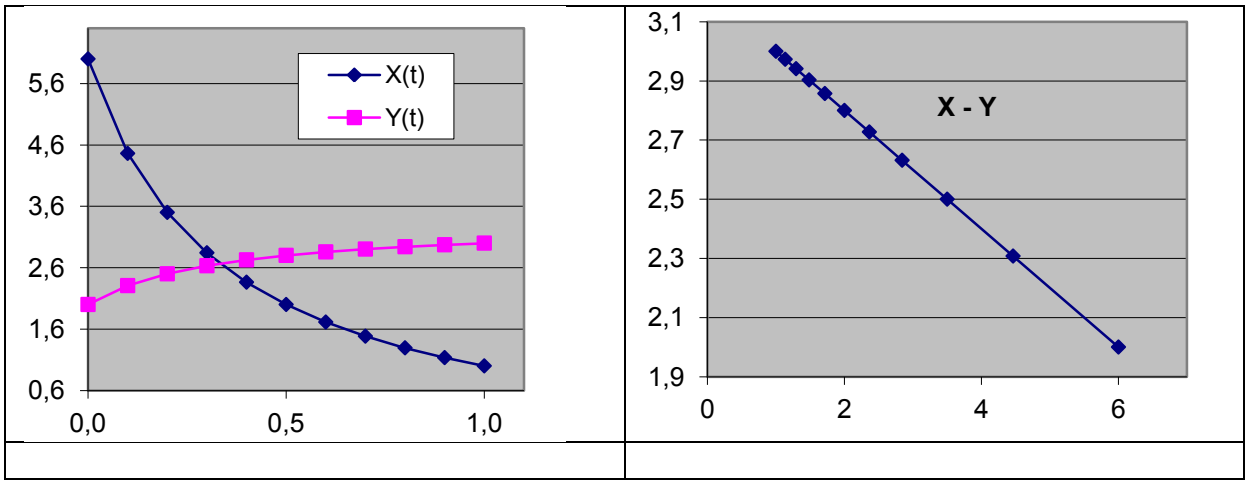
$$\begin{bmatrix} x \\ w \end{bmatrix}(t) = \begin{bmatrix} w_2x_1 \\ w_1w_2 \end{bmatrix} + \left(\begin{bmatrix} w_1x_2 \\ w_1w_2 \end{bmatrix} - \begin{bmatrix} w_2x_1 \\ w_1w_2 \end{bmatrix} \right) t$$

Analogicky i pro E^2 a E^3. Parametrizace parametrem t je *lineární*, avšak je zapotřebí několik operací násobení.

Otázkou je, zda je možné provést interpolaci přímo v projektivním prostoru a jaké to má důsledky. Lineární interpolaci v projektivním prostoru, tj. body jsou dány v homogenních souřadnicích, můžeme zapsat:

$$\boldsymbol{x}(t) = \boldsymbol{x}_1 + (\boldsymbol{x}_2 - \boldsymbol{x}_1)\, \tau$$

Lze ukázat, že výsledkem je přímka v projektivním prostoru s parametrizací danou parametrem τ. Z předchozího víme, zejména pak viz kap.4.1 (Perspektivní projekce), že parametr $\tau \neq t$, kromě krajních bodů, neboť při převodu do Eukleidovského prostoru jde vlastně o perspektivní projekci. *Parametr τ monotónně roste nebo klesá, avšak nelineárně*. V tomto případě projektivní reprezentace bodů jde o *lineární interpolaci s nelineární monotónní parametrizací*.



Výše uvedený postup lze aplikovat obecně pro E^n a i pro rovinou interpolaci na ploše, tj.:

$$\boldsymbol{x}(u,v) = \boldsymbol{x}_1 + (\boldsymbol{x}_2 - \boldsymbol{x}_1)\,u + (\boldsymbol{x}_3 - \boldsymbol{x}_1)\,v$$

Takže lze vlastně lineární interpolace rozdělit takto“

$$\text{lineární interpolace} = \begin{cases} \text{v Eukleidovském prostoru s lineární parametrizací} \\ \text{v projektivním prostoru} \begin{cases} \text{s lineární parametrizací} \\ \text{s nelinární monotónní parametrizací} \end{cases} \end{cases}$$

V mnoha algoritmech nepotřebujeme lineární parametrizaci, postačuje pouze monotónní parametrizace. Typickými ukázkami jsou:

- metoda sledování paprsku (Ray tracing), viz kap.13.1 (Metoda sledování paprsku), kdy je nutné určit, který objekt protnutý polopřímkou, tj. paprskem, je nejblíže počátku polopřímky
- algoritmus Cyrus-Beck pro ořezávání přímky, resp. úsečky konvexním n-úhelníkem – vhodnou modifikací lze takto algoritmus zrychlit cca o 15%

V předchozím textu byl ukázán výpočet barycetrických souřadnic, který vede na řešení soustavy lineárních rovnic $\boldsymbol{Ax} = \boldsymbol{b}$. Je tedy přirozenou otázkou, zda barycentrické souřadnice je možné počítat také přímo v projektivním prostoru. Vlastní odvození je nad rámec tohoto textu a lze jej najit v:

Skala,V.: Barycentric Coordinates Computation in Homogeneous Coordinates, Computers & Graphics, Elsevier, ISSN 0097-8493, Vol. 32, No.1, pp.120-127, 2008

Nicméně v kap.2.4 (Dualita a její aplikace) bylo ukázáno, že řešení soustav lineárních rovnic je ekvivalentní zobecněnému vektorovému součinu. Uvažme pro jednoduchost výpočet barycetrických souřadnic pro bod v trojúhelníku v E^2.

$\lambda_1 X_1 + \lambda_2 X_2 + \lambda_3 X_3 = X$	$\lambda_1 Y_1 + \lambda_2 Y_2 + \lambda_3 Y_3 = Y$	$\lambda_1 + \lambda_2 + \lambda_3 = 1$

Uvedenou soustavu rovnic lze přepsat do implicitní formy, tj.:

$\begin{bmatrix} X_1 & X_2 & X_3 & X \\ Y_1 & Y_2 & Y_3 & Y \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$	$\omega_1 X_1 + \omega_2 X_2 + \omega_3 X_3 + \omega_4 X = 0$ $\omega_1 Y_1 + \omega_2 Y_2 + \omega_3 Y_3 + \omega_4 Y = 0$ $\omega_1 + \omega_2 + \omega_3 + \omega_4 = 0$ $\lambda_i = -\frac{\omega_i}{\omega_4}$

Řešení uvedené soustavy rovnic $\boldsymbol{Ax} = \mathbf{0}$ je ekvivalentní zobecněnému vektorovému součinu a lze psát:

$$\boldsymbol{\omega} = \boldsymbol{\xi} \times \boldsymbol{\eta} \times \boldsymbol{\tau} = \begin{bmatrix} \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} & \boldsymbol{l} \\ X_1 & X_2 & X_3 & X \\ Y_1 & Y_2 & Y_3 & Y \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

kde:

$\boldsymbol{\xi} = [X_1, X_2, X_3, X\ \ ]^T$, $\boldsymbol{\eta} = [Y_1, Y_2, Y_3, Y\ \ ]^T$, $\boldsymbol{\tau} = [1,1,1,1]^T$, $\boldsymbol{\omega} = [\omega_1, \omega_2, \omega_3 : \omega_4]^T$

$\boldsymbol{i} = [1,0,0 : 0]^T$, $\boldsymbol{j} = [0,1,0 : 0]^T$, $\boldsymbol{k} = [0,0,1 : 0]^T$, $\boldsymbol{l} = [0,0,0 : 1]^T$

Je tedy zřejmé, že:

- pro výpočet barycentrických souřadnic není nutné řešit soustavu rovnic, např. Gaussovou eliminační metodou nebo nějakou jinou, a lze použít vektorový součin
- výpočet barycentrických souřadnic, pokud souřadnice jsou v homogenních souřadnicích je také jednoduchý, neboť determinant je multilineární

Výpočet barycentrických souřadnic pro body v projektivním prostoru:

$$\boldsymbol{\omega} = \begin{bmatrix} \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} & \boldsymbol{l} \\ X_1 & X_2 & X_3 & X \\ Y_1 & Y_2 & Y_3 & Y \\ 1 & 1 & 1 & 1 \end{bmatrix} = \frac{1}{w_1 w_2 w_3 w} \begin{bmatrix} \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} & \boldsymbol{l} \\ w_1 X_1 & w_2 X_2 & w_3 X_3 & wX \\ w_1 Y_1 & w_2 Y_2 & w_3 Y_3 & wY \\ w_1 & w_2 & w_3 & w \end{bmatrix} \triangleq \begin{bmatrix} \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} & \boldsymbol{l} \\ x_1 & x_2 & x_3 & x \\ y_1 & y_2 & y_3 & y \\ w_1 & w_2 & w_3 & w \end{bmatrix}$$

Takže lze psát

$$\boldsymbol{\omega} = \boldsymbol{\xi} \times \boldsymbol{\eta} \times \boldsymbol{\tau}$$

kde:

$\boldsymbol{\xi} = [x_1, x_2, x_3, x\ \ ]^T$, $\boldsymbol{\eta} = [y_1, y_2, y_3, y\ \ ]^T$, $\boldsymbol{\tau} = [w_1, w_2, w_3, w]^T$, $\boldsymbol{\omega} = [\omega_1, \omega_2, \omega_3 : \omega_4]^T$

$\boldsymbol{i} = [1,0,0 : 0]^T$, $\boldsymbol{j} = [0,1,0 : 0]^T$, $\boldsymbol{k} = [0,0,1 : 0]^T$, $\boldsymbol{l} = [0,0,0 : 1]^T$

Barycentrické souřadnice jsou pak dány:

$\lambda_1 = -\frac{\omega_1}{\omega_2 \omega_3 \omega_4}$	$\lambda_2 = -\frac{\omega_2}{\omega_1 \omega_3 \omega_4}$	$\lambda_3 = -\frac{\omega_3}{\omega_1 \omega_2 \omega_4}$

Poznámka

Z výše uvedeného je zřejmé, že výpočetní postup je jednoduchý, elegantní, nepotřebuje ke své realizaci operaci dělení a je vhodný pro GPU aplikace. Dále pak:

- výše uvedený způsob výpočtu barycentrických souřadnic má lineární parametrizaci
- pro zjištění, zda bod je uvnitř trojúhelníka je vhodné použít modifikovaný test

$0 \leq \omega_1 \leq -\omega_2 \omega_3 \omega$	$0 \leq \omega_2 \leq -\omega_1 \omega_3 \omega$	$0 \leq \omega_3 \leq -\omega_1 \omega_2 \omega$

Rozšířený vektorový součin může být implementován na GPU např. takto:

```
float4 cross_4D(float4 x1, float4 x2, float4 x3)
{
        float4 a;
        a.x = dot(x1.yzw, cross(x2.yzw, x3.yzw));
        a.y = -dot(x1.xzw, cross(x2.xzw, x3.xzw));
        a.z = dot(x1.xyw, cross(x2.xyw, x3.xyw));
        a.w = -dot(x1.xyz, cross(x2.xyz, x3.xyz));
        return a
}
```

Až dosud jsme se zabývali lineárními interpolacemi. Další specifickou interpolací je interpolace sférická, která je interpolací „po oblouku“ s *téměř lineární parametrizací*.

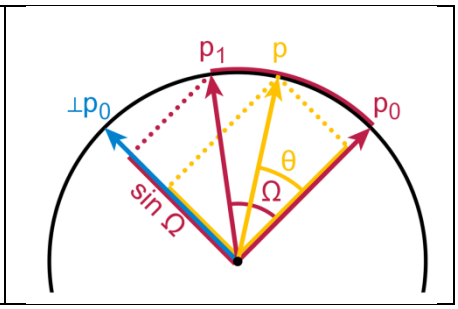
9.2.Sférická interpolace

Sférická interpolace je interpolací speciální a vychází z následujících předpokladů:

- střed rotace je v počátku souřadného systému
- pokud se interpolují normálové vektory, pak mají průsečík v počátku souřadného systému
- interpoluje se po kulové ploše

Sférická interpolace je interpolací s *téměř* lineární parametrizací.

$$slerp(\boldsymbol{X}_0, \boldsymbol{X}_1, t) = \frac{\sin[(1-t)\Omega]}{\sin\Omega}\boldsymbol{X}_0 + \frac{\sin[t\Omega]}{\sin\Omega}\boldsymbol{X}_1$$



Je nutné upozornit, že pro $\Omega \to k\pi,\ k = 0, \pm 1, \ldots$ je formule nestabilní, neboť se dělí hodnotou blízkou nule.

Proto je vhodné používat projektivní reprezentaci $slerp_p(\boldsymbol{x}_0, \boldsymbol{x}_1, t)$ a „odložit“ operaci dělení.

$$slerp(\boldsymbol{X}_0, \boldsymbol{X}_1, t) \triangleq slerp_p(\boldsymbol{X}_0, \boldsymbol{X}_1, t) = \begin{bmatrix} sin[(1-t)\Omega]\boldsymbol{X}_0 + sin[t\Omega]\boldsymbol{X}_1 \\ sin\,\Omega \end{bmatrix}$$
$$= [sin[(1-t)\Omega]\boldsymbol{X}_0 + sin[t\Omega]\boldsymbol{X}_1 : \sin\Omega]^T$$

Je určitě na místě se ptát, jak je sférická interpolace definována, pokud body jsou zadány obecně v homogenních souřadnicích. Pak lze psát pro x a w souřadnice (analogicky pro y, z):

$$x(t) = \frac{sin[(1-t)\Omega]}{sin\,\Omega}x_0 + \frac{sin[t\Omega]}{sin\,\Omega}x_1 \qquad w(t) = \frac{sin[(1-t)\Omega]}{sin\,\Omega}w_0 + \frac{sin[t\Omega]}{sin\,\Omega}w_1$$

Pak interpolované souřadnice bodu vyjádřené v Eukleidovských souřadnicích jsou:

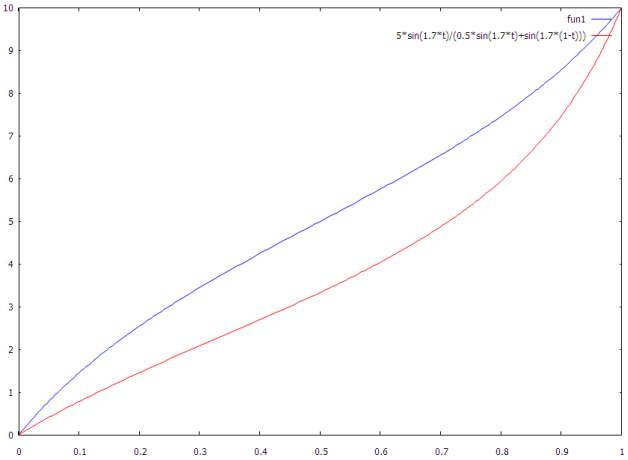
$$\boldsymbol{X}(t) = \frac{\frac{sin[(1-t)\Omega]}{sin\,\Omega}x_0 + \frac{sin[t\Omega]}{sin\,\Omega}x_1}{\frac{sin[(1-t)\Omega]}{sin\,\Omega}w_0 + \frac{sin[t\Omega]}{sin\,\Omega}w_1} = \frac{sin[(1-t)\Omega]\,x_0 + sin[t\Omega]\,x_1}{sin[(1-t)\Omega]\,w_0 + sin[t\Omega]\,w_1}$$

Pak lze psát pro x (analogicky pro y, z) a w souřadnice:

$$\boldsymbol{x}(t) = \begin{bmatrix} sin[(1-t)\Omega]\,x_0 + sin[t\Omega]\,x_1 \\ sin[(1-t)\Omega]\,w_0 + sin[t\Omega]\,w_1 \end{bmatrix}$$

To znamená, že máme lepší numerickou stabilitu.

Obr.XXX znázorňuje vliv hodnot homogenních složek w na míru nelineárnosti parametrizace sférické interpolace.



9.3.RBF interpolace

V mnoha oblastech dříve uvedené metody nejsou aplikovatelné vzhledem ke specifickým předpokladům. Až dosud jsme se zabývali interpolací uspořádaných dat v různém kontextu. Nicméně, zejména v úlohách s vyšší dimenzí nebo v případě časově proměnných dat, kdy se strukturovaná síť musí vytvářet, se upřednostňují tzv. bezsíťové („meshless“) interpolační techniky.

Jednou z takových interpolačních technik je interpolace pomocí radiálních bázových funkcí (Radial Basis Functions), tzv. RBF interpolace. RBF interpolace je vhodnou metodou pro interpolaci neuspořádaných dat, např. skalárního potenciálového nebo vektorového pole. Jde tedy o úlohu, kdy jsou zadány souřadnice bodů $\boldsymbol{x}_i \in E^n$ a hodnoty $\boldsymbol{h}_i \in E^p$, které se mají interpolovat.

RBF interpolace je v postatě úlohou řešení soustavy lineárních rovnic $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, kde $\boldsymbol{b}$ je vektor zadaných hodnot a $\boldsymbol{x}$ je vektor vah pro jednotlivé interpolační funkce. Výsledná interpolovaná hodnota je pak dána výrazem:

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} \lambda_i \, \phi(r_i)$$

kde λ_i jsou váhy, $\phi(r_i) = \phi(\|\boldsymbol{x} - \boldsymbol{x}_i\|)$ jsou hodnoty interpolačních funkcí a $\boldsymbol{x}_i$ jsou zadané hodnoty souřadnic i-tého bodu a $r_i = \|\boldsymbol{x} - \boldsymbol{x}_i\|$.

V následujícím si ukážeme princip RBF interpolace, tj. kdy interpolační křivka nebo plocha prochází danými body, a aproximace, kdy aproximační křivka nebo plocha nemusí nutně procházet danými body.

RBF interpolace má následující významné vlastnosti:

- RFB je určena především pro interpolaci neuspořádaných dat
- RBF interpolace je vhodná pro interpolaci d-rozměrných dat, $d \geq 2$
- RBF je založena na „parametrizaci“ na vzdálenostech bodů $r_{ij} = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|$, a tedy jde o interpolaci v principu neseparabilní, tj. není možné interpolovat po jednotlivých dimenzích

RBF interpolace používají rozdílné interpolační bázové funkce $\phi(r)$, tj. $\phi(\|\boldsymbol{x}_i - \boldsymbol{x}_j\|)$. RBF interpolace lze rozdělit do dvou základních skupin, a to dle typů funkcí, které se pro interpolaci používají:

- globální funkce, které mají vliv na celý interval, a které lze rozdělit na funkce, jejichž hodnota:
  - roste s rostoucí vzdáleností r, např. $\phi(r) = r^2 \lg r$ nebo $\phi(r) = r^3$
  - klesá s rostoucí vzdáleností r, např. $\phi(r) = {}^{1}/_{r^2}$ nebo $\phi(r) = e^{-\varepsilon r^2}$, kde ε je parametr

  Při použití globálních funkcí je matice $\boldsymbol{A}$ poměrně hodně zaplněna nenulovými hodnotami a podmíněnost soustavy rovnic se zhoršuje s rostoucím řádem matice.

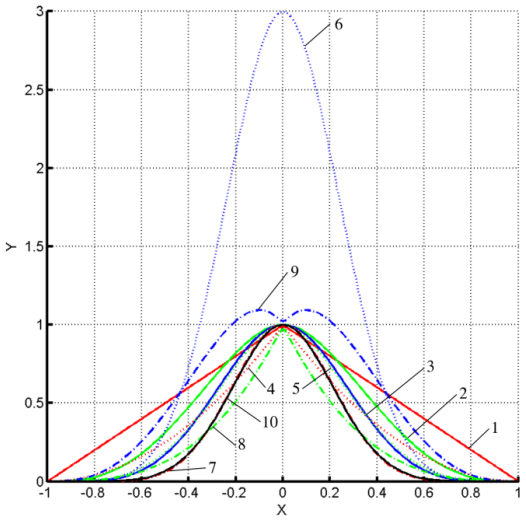
- lokální funkce („Compactly Supported“ RBF-CSRBF), jejichž hodnoty jsou nenulové jen pro určitý interval, většinou pro $r \in \langle 0,1 \rangle$. Některé typické CSRBF funkce jsou uvedeny níže.

  Výhodou použití CSRBF je především to, že matice $\boldsymbol{A}$ je maticí řídkou, což následně vede k podstatnému snížení výpočetní složitosti. Nicméně, vzhledem k tomu, že jejich vliv je omezen na interval $r \in \langle 0,1 \rangle$, je nutno všechny vzdálenosti r zvětšit, resp. zmenšit tak, aby v intervalu pro r bylo více zadaných bodů.

Uveďme základní „globální“ funkce používané při RB interpolacích:

Thin-Plate Spline (TPS)	$r^2 \lg r$	Inverse multiquadric (IMQ)	$1/\sqrt{1+\varepsilon r^2}$
Gauss	$e^{-\varepsilon r^2}$	Multiquadric (MQ)	$\sqrt{1+\varepsilon r^2}$
Inverse Quadric	$1/(1+\varepsilon r^2)$		

Tab x: “globální“ RBF funkce



Obr.X: Geometrické vlastnosti CSRBF

ID	CSRBF	ID	CSRBF
1	$(1-r)_+$	6	$(1-r)_+^6(35r^2+18r+3)$
2	$(1-r)_+^3(3r+1)$	7	$(1-r)_+^8(32r^3+25r^2+8r+3)$
3	$(1-r)_+^5(8r^2+5r+1)$	8	$(1-r)_+^3$
4	$(1-r)_+^2$	9	$(1-r)_+^3(5r+1)$
5	$(1-r)_+^4(4r+1)$	10	$(1-r)_+^7(16r^2+7r+1)$

Tab. x: Typické CSRBF funkce

V roce 1971 navrhl Hardy [Hardy 1971] interpolaci pomocí RBF s použitím funkce „multiquadric” a metodu nazval metodou radiálních bázových funkcí, která je založena na interpolačním vzorci:

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} \lambda_i \, \phi(r_i)$$

kde: $\phi(r_i) = \phi(\|\boldsymbol{x} - \boldsymbol{x}_i\|)$ a $\boldsymbol{x}$ je d-dimensionální vektor a λ_i jsou váhy. V roce 1977 Duchoň [Duchon 1977] použil funkci typu $\phi(r) = r^2 lg\, r$, kterou nazval „Thin-Plate Spline“ (TPS), Shagen [Shagen 1979] v roce 1979 navrhl funkci $\phi(r) = e^{-(\epsilon r)^2}$ a Wetland [Wetland 2005] v roce 2005 navrhl použití CSRBF („compactly supported“) funkcí:

$$\phi(r) = \begin{cases} (1-r)^q \, P(r), & 0 \le r \le 1 \\ 0, & r > 1 \end{cases}$$

kde: $P(r)$ je polynom a q je parametr. Otázky řešitelnosti a stability jsou řešeny např. v [Micchelli 1986] a [Wright 2003]. Wright rozšířil z důvodů stability původní RBF interpolaci polynomem k-tého stupně takto:

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} \lambda_i \, \phi(\|\boldsymbol{x} - \boldsymbol{x}_i\|) + P_k(\boldsymbol{x}) = \sum_{i=1}^{N} \lambda_i \, \phi_i(\boldsymbol{x}) + P_k(\boldsymbol{x})$$

a zavedl dodatečné podmínky:

$\sum_{i=1}^{N} \lambda_i = 0$	$\sum_{i=1}^{N} \lambda_i \, \boldsymbol{x} = \boldsymbol{0}$

Obvykle je místo polynomu k-tého stupně $P_k(\boldsymbol{x})$ použit polynom lineární a to:

$$P_k(\boldsymbol{x}) = a_0 + \boldsymbol{a}^T \, \boldsymbol{x}$$

Geometricky vzato, člen a_0 vlastně „nastavuje“ hodnotu pro $\boldsymbol{x} = \boldsymbol{0}$ a druhý člen $\boldsymbol{a}^T \boldsymbol{x}$, pak vlastně reprezentuje v případě E^3, tj. $\{\boldsymbol{x}, h\}$, naklonění“ roviny, v obecném případě „naklonění“ nadroviny. Z uvedeného je zřejmé, že RBF interpolace není nezávislá na posunutí a dalších geometrických transformací.

Protože hodnoty $h_j = f(\boldsymbol{x}_j)$ v bodech $\boldsymbol{x}_i$ jsou známy pro $j = 1, \dots, n$, rovnice tvoří systém lineárních rovnic $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, jehož řešením dostaneme koeficienty λ_j a $a_0, \boldsymbol{a}$, tj.:

$$f(\boldsymbol{x}_j) = \sum_{i=1}^{N} \lambda_i \, \phi(\|\boldsymbol{x}_j - \boldsymbol{x}_i\|) + P_k(\boldsymbol{x}_j) = \sum_{i=1}^{N} \lambda_i \, \phi_{i,j} + P_k(\boldsymbol{x}_j) \qquad j = 1, \dots, n$$

Je tedy zřejmé, že pro d-dimensionální interpolaci a N daných bodů dostáváme soustavu

$$(N + d + 1) \times (N + d + 1)$$

rovnic.

Pro $d = 2$ jsou tedy vektory $\boldsymbol{x}_i = [x_i, y_i]^T$ a $\boldsymbol{a} = [a_x, a_y]^T$. Výše uvedenou formuli můžeme rozepsat takto:

$$\begin{bmatrix} \phi_{1,1} & .. & \phi_{1,N} & 1 & x_1 & y_1 \\ : & \ddots & : & : & : & : \\ \phi_{N,1} & .. & \phi_{N,N} & 1 & x_N & y_N \\ 1 & .. & 1 & 0 & 0 & 0 \\ x_1 & .. & x_N & 0 & 0 & 0 \\ y_1 & .. & y_N & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ : \\ \lambda_N \\ a_0 \\ a_x \\ a_y \end{bmatrix} = \begin{bmatrix} f_1 \\ : \\ f_N \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

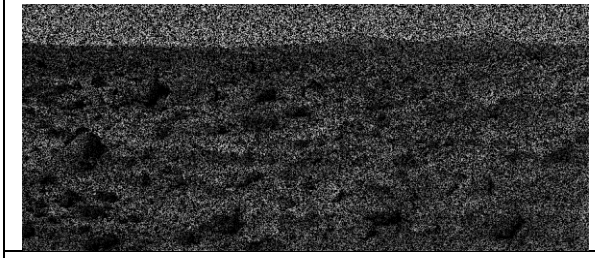
$$\begin{bmatrix} \boldsymbol{B} & \boldsymbol{P} \\ \boldsymbol{P}^T & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{\lambda} \\ \boldsymbol{a} \end{bmatrix} = \begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{0} \end{bmatrix}$$

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$$

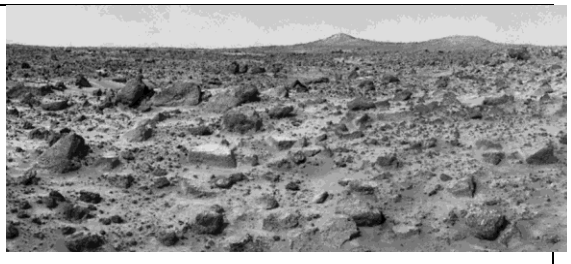
$$\boldsymbol{a}^T \, \boldsymbol{x_i} + a_0 = a_0 + a_x \, x_i + a_y \, y_i$$

RBF interpolací lze využít:

- při rekonstrukci poškozených obrazů

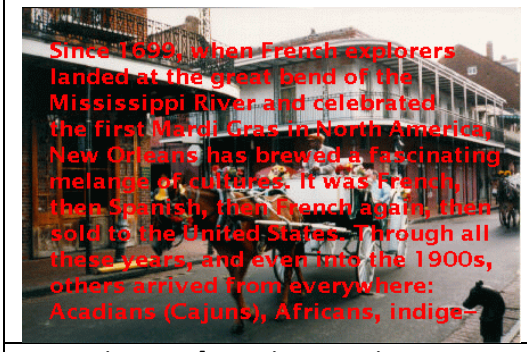


Původní obraz - 60% poškozených pixelů



Rekonstruovaný obraz

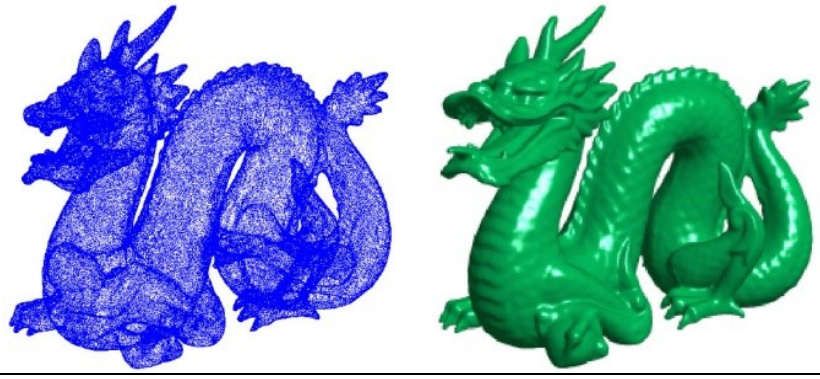
- pro odstranění nápisů na předlohách (inpaining), resp. trhlin na fotografiích a na nástěnných malbách apod.



Original image [Bertalmio et al. 2000

Reconstructed image [Uhlir and Skala 2006]

- při rekonstrukci 3D povrchu z nasnímaných bodů.



Surface reconstruction (438000 points) [Carr et al. 2001]

V případě rekonstrukce povrchu z nasnímaných bodů úloha vede na soustavu homogenních rovnic, tj. $\boldsymbol{Ax} = \boldsymbol{0}$, tj. nemáme žádnou informaci o orientaci plochy, a proto je nutné použít speciálních přístupů.

Je zřejmé, že výše uvedenou techniku lze s výhodou použít k „převzorkování“, např. převzorkování naměřených dat do pravoúhlé strukturované pravidelné datové struktury.

Až dosud jsme se zabývali interpolací neuspořádaných dat, tj. případem, kdy interpolovaná křivka či plocha prochází danými body. V mnoha případech jsou vzorky interpolované veličiny příliš „husté“, případně nepožadujeme naprostou přesnost, ale chceme snížit počet parametrů RBF. V tomto případě můžeme použít RBF aproximace.

9.4.Aproximace

Aproximace, na rozdíl od interpolací, prokládají danými daty křivku, která obecně neprochází zadanými body. Aproximace dat se používá při řešení mnoha problémů, kdy jsou dány vzorky dat a je nutné je přibližně reprezentovat nějakým funkčním popisem.

V technické praxi se velmi často používá metoda nejmenších čtverců (Least Square Error - LSE) pro nalezení „optimálního“ řešení. Při její aplikaci je však nutné mít na paměti, že „standardní“ metoda nejmenších čtverců může poskytovat neadekvátní výsledky, neboť se chyba měří „na svislici“ a vliv odchylky roste s kvadrátem. Lepší výsledky dává ortogonální metoda, která měří vzdálenosti „na kolmici“, nicméně je podstatně výpočetně náročnější.

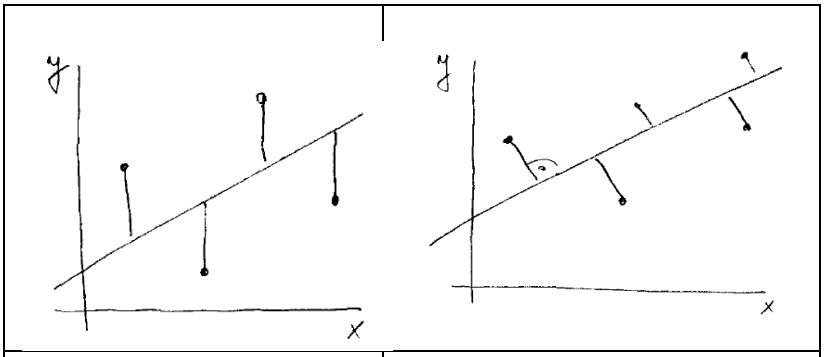
„Standardní“ metoda nejmenších čtverců je použitelná na situace, kdy výsledná aproximace je explicitního charakteru, tj.

$$y = f(x_1, \dots, x_n) = f(\boldsymbol{x}) \tag{7}$$

V případě, že výsledná aproximace je implicitního charakteru, tj.

$$F(x_1, \dots, x_n) = F(\boldsymbol{x}) = 0 \tag{8}$$

nelze výše uvedený postup aplikovat a musí se použít jiné postupy.



Standardní lin.regrese	Ortogonální lineární regrese

Rozdíl mezi zadanou hodnotou a hodnotou aproximovanou je pak chyba (Error), kterou se snažíme minimalizovat podle nějakého kritéria. Průměrnou chybu aproximace lze pak definovat takto:

$$\varepsilon = \frac{1}{n}\sum_{i=1}^{n} |f(\boldsymbol{x}_i) - h_i|$$

kde: h_i jsou zadané hodnoty v bodech $\boldsymbol{x}_i$, $f(\boldsymbol{x}_i)$ jsou hodnoty v daných bodech vypočtené a n je počet zadaných bodů. Hodnota ε je hodnota určující průměrnou chybu na jeden bod, tj. jde o relativizující kritérium vůči počtu bodů.

Je nutné upozornit, že velmi často používaná metoda nejmenších čtverců nemusí být vhodná, neboť penalizační váha odchylky roste s kvadrátem odchylky. Toto může být problém při prokládání daných bodů přímkou v E^2, resp. prokládání rovinou v případě E^3, kdy bychom potřebovali minimální odchylku vzdáleností, nikoliv jejich kvadrátů.

$\min_{a,b}\left\{\sum_{i=1}^{n}\left\lvert\frac{ax_i + by_i + c}{\sqrt{a^2+b^2}}\right\rvert\right\}$	$\min_{a,b}\left\{\sum_{i=1}^{n}\left(\frac{ax_i + by_i + c}{\sqrt{a^2+b^2}}\right)^2\right\}$
Minimalizace vzdáleností	Minimalizace čtverce vzdáleností metodou nejmenších čtverců

9.5.Aproximace - nejmenší čtverce

Asi nejčastěji používanou aproximací je metoda nejmenších čtverců. Nechť jsou dány skalární hodnoty h_i v bodech $\boldsymbol{x}_i$, a chceme těmito body proložit křivku $\phi(\boldsymbol{x})$, která je tvaru:

$$\phi(\boldsymbol{x}) = \xi_1\varphi_1(\boldsymbol{x}) + \cdots + \xi_p\varphi_p(\boldsymbol{x}) = \left[\xi_1, \ldots, \xi_p\right]\left[\varphi_1(\boldsymbol{x}), \ldots, \varphi_p(\boldsymbol{x})\right]^T$$

přičemž funkce $\varphi_i(\boldsymbol{x})$ nejsou závislé na žádném parametru a $n > p$. Hledáme takové koeficienty ξ_i, které minimalizují námi zadané kritérium, v našem případě použijeme nejmenší čtverce.

Lze nahlédnout, že pro jednotlivé body lze v maticové formě psát:

$\begin{bmatrix} \varphi_1(\boldsymbol{x}_1) & \cdots & \varphi_p(\boldsymbol{x}_1) \\ \varphi_1(\boldsymbol{x}_2) & \cdots & \varphi_p(\boldsymbol{x}_2) \\ \vdots & \ddots & \vdots \\ \varphi_1(\boldsymbol{x}_{n-1}) & \cdots & \varphi_p(\boldsymbol{x}_{n-1}) \\ \varphi_1(\boldsymbol{x}_n) & \cdots & \varphi_p(\boldsymbol{x}_n) \end{bmatrix} \begin{bmatrix} \xi_1 \\ \vdots \\ \xi_p \end{bmatrix} = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_{n-1} \\ h_n \end{bmatrix}$	Tedy dostáváme	$\boldsymbol{Ax} = \boldsymbol{b}$

čímž dostáváme přeurčenou soustavu rovnic $\boldsymbol{Ax} = \boldsymbol{b}$ a potřebujeme určit hodnoty vektoru $\boldsymbol{x}$. Nyní můžeme definovat chybu r a její kvadrát r^2 takto:

$r = \|\boldsymbol{Ax} - \boldsymbol{b}\|$	$r^2 = (\boldsymbol{\Phi\xi} - \boldsymbol{h})^T(\boldsymbol{\Phi\xi} - \boldsymbol{h})$

$$r^2 = (\boldsymbol{Ax} - \boldsymbol{b})^T(\boldsymbol{Ax} - \boldsymbol{h}) = (\boldsymbol{Ax})^T(\boldsymbol{Ax}) - (\boldsymbol{Ax})^T\boldsymbol{b} - \boldsymbol{b}^T\boldsymbol{Ax} + \boldsymbol{b}^T\boldsymbol{b}$$

Protože platí

$$\boldsymbol{b}^T\boldsymbol{Ax} = (\boldsymbol{Ax})^T\boldsymbol{b}$$

$$r^2 = (\boldsymbol{Ax} - \boldsymbol{b})^T(\boldsymbol{Ax} - \boldsymbol{h}) = (\boldsymbol{Ax})^T(\boldsymbol{Ax}) - 2\boldsymbol{x}^T\boldsymbol{A}^T\boldsymbol{b} + \boldsymbol{b}^T\boldsymbol{b}$$

Pro extrém musí platit podmínka

$$\frac{\partial r^2}{\partial \boldsymbol{x}} = 2\boldsymbol{A}^T\boldsymbol{Ax} - 2\boldsymbol{A}^T\boldsymbol{b} = 0$$

Takže dostáváme soustavu lineárních rovnic:

$$\boldsymbol{A}^T\boldsymbol{Ax} = \boldsymbol{A}^T\boldsymbol{b} \tag{9}$$

Matice $\boldsymbol{A}^T\boldsymbol{A}$ je symetrická pozitivně semidefinitní. Ve většině reálných případů je pozitivně definitní.

Příklad

Pro jednoduchost uvažme body reprezentující povrch koule získaných, např. odměřováním, skenováním apod. Je tedy dána množina bodů $\{\boldsymbol{x}_i\}_{i=1}^n$ reprezentující povrch koule a počet bodů $n > 4$. Body na povrchu koule musí vyhovovat rovnici:

$$x^2 + y^2 + z^2 + ax + by + cz + d = 0 \tag{10}$$

Pro jednotlivé body $\boldsymbol{x}_i$ by měla platit tato rovnice, tj.:

$$x_i^2 + y_i^2 + z_i^2 + ax_i + by_i + cz_i + d = 0 \quad i = 1, \ldots, n \tag{11}$$

Je zřejmé, že žádné měření není absolutně platné a tedy rovnice zřejmě nebude splněna ani pro jeden daný bod. Úkolem je však nalézt parametry pro kouli „nejlépe“ aproximující naměřený povrch.

$$\begin{bmatrix} x_1 & y_1 & z_1 & 1 \\ x_2 & \ddots & & \\ & & \ddots & 1 \\ x_n & y_n & z_n & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = -\begin{bmatrix} x_1^2 + y_1^2 + z_1^2 \\ x_2^2 + y_2^2 + z_2^2 \\ \vdots \\ x_n^2 + y_n^2 + z_n^2 \end{bmatrix} \tag{12}$$

Dostáváme opět přeurčenou soustavu rovnic $\boldsymbol{Ax} = \boldsymbol{b}$, kterou můžeme řešit různými numerickými postupy, např. metodou nejmenších čtverců nebo dekompozicí (Singular Value Decomposition - SVD) apod.

Je nutné zdůraznit, že stabilita řešení prudce klesá s rozměrem výsledné matice $\boldsymbol{A}^T\boldsymbol{A}$.

Příklad

V mnoha případech potřebujeme posoudit chování algoritmu, tj. časovou nebo paměťovou složitost na základě experimentálních měření. Uvažme algoritmus, který zpracovává „typická data“ pro různý počet zpracovávaných dat. To znamená, že z experimentů dostáváme množinu $\boldsymbol{\Omega} = \{\langle n_i, t_i \rangle\}_{i=1}^{N}$, kde: n_i je počet zpracovávaných primitiv, t_i je spotřebovaný výpočetní čas a N je celkový počet experimentů.

Úkolem je odhadnout výpočetní složitost algoritmu.

Předpokládejme, že základní uvažované složitosti jednotlivých částí algoritmu jsou např.

$$O(lg\ n),\ O(\sqrt{n}),\ O(n),\ O(n\ lg\ n),\ \ldots,\ O(n^2),\ \ldots$$

Pak tyto funkce jsou vlastně funkce $\varphi_i(n)$, viz předchozí text, tj.

$$\phi(n) = \xi_1\varphi_1(n) + \cdots + \xi_p\varphi_p(n)$$

protože každý algoritmus má fixní počáteční složitost $O(1)$, položíme:

$\varphi_1(n) = 1$	$\varphi_2(n) = lg\ n$	$\varphi_3(n) = \sqrt{n}$	$\varphi_4(n) = n\ lg\ n$	……	$\varphi_p(n) = n^2$

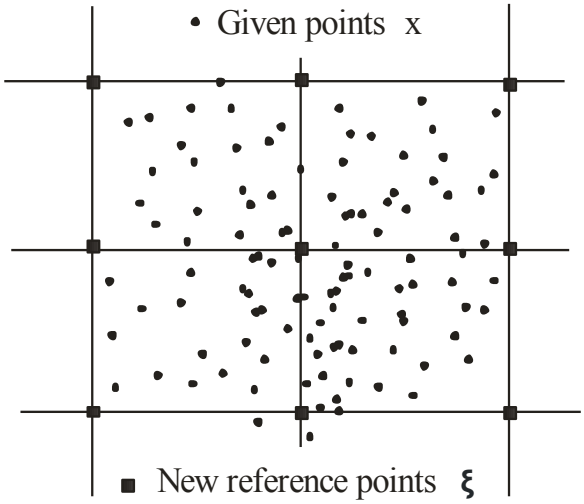
přičemž $p \ll N$.

Dostáváme tedy opět přeurčenou soustavu rovnic $\boldsymbol{Ax} = \boldsymbol{b}$, kterou vyřešíme výše uvedenými způsoby.

9.6.RBF aproximace

Interpolace RBF v zásadě spoléhá na řešení soustavy lineárních rovnic $\boldsymbol{Ax} = \boldsymbol{b}$ řádu $(N + d + 1)$, kde N je počet daných bodů a d je dimenze nezávisle proměnných. Ať už v případě použití „globálních“ nebo CSRBF funkcí, matice jsou řádově velké a v mnoha případech jsou data příliš hustě vzorkována. Je tedy legitimní otázkou, zda je možné „redukovat“ problém, tj. snížit počet hodnot λ_i s dobrou aproximační přesností. Typickou úlohou může být reprezentace terénu, kdy máme vlastně zadané body (x, y) s výškou h.

Zkusme se proto podívat na RBF interpolaci z jiného hlediska. Předpokládejme, že máme neuspořádaná data v E^2 (dále uvedené také obecně platí pro d-dimensionální případ) a do datového setu vložme „virtuálně“ body $\boldsymbol{\xi}_j$, viz obr.xxxx. „Virtuálně“ vložené body $\boldsymbol{\xi}_j$, které nemusí být v pravoúhlé mřížce, může uživatel vložit tak, aby co nejlépe vystihovaly daný povrch, např. profil terénu apod. Počet vložených bodů $\boldsymbol{\xi}_j$ bude M a bude podstatně menší než počet zadaných bodu N, tj. $M \ll N$.



Obr.xxxx: RBF aproximace a redukce bodů

Aplikujeme-li RBF, pak interpolovaná hodnota může být určena obdobně definovanou funkcí, přičemž hodnota r_{ij} je určena nyní vztahem $r_{ij} = \|\boldsymbol{x}_i - \boldsymbol{\xi}_j\|$. Jde tedy o vzdálenosti mezi body danými a body virtuálně vloženými. Interpolovaná hodnota je tedy určena vzorcem:

$$f(\boldsymbol{x}) = \sum_{j=1}^{M} \lambda_j \, \varphi(\|\boldsymbol{x} - \boldsymbol{\xi}_j\|)$$

přičemž interpolační funkce $\varphi(r)$ jsou stejné jako v případě RBF interpolací. Je zřejmé, že opět pro dané hodnoty dostáváme soustavu rovnic:

$$f(\boldsymbol{x}_i) = \sum_{j=1}^{M} \lambda_j \, \varphi(\|\boldsymbol{x}_i - \boldsymbol{\xi}_j\|)$$

$$h_i = f(\boldsymbol{x}_i) \qquad i = 1, \dots, N$$

kde $\boldsymbol{\xi}_j$ nejsou dané body, ale body „virtuálně“ vložené. Pro jednoduchost nyní uvažme, např. že pokud poměr počtu zadaných hodnot a počtu „virtuálně“ vložených je $N/M = 10^2$, pak rychlost vlastního výpočtu *pouze jedné aproximované hodnoty* bude také cca $10^2 \times$ rychlejší.

Výše uvedená formulace vede na řešení soustavy lineárních rovnic $\boldsymbol{Ax} = \boldsymbol{b}$, kde počet řádek $N \gg M$ a kde M je počet neznámých vah $[\lambda_1 , \dots , \lambda_M]^T$.

$$\begin{bmatrix} \varphi_{1,1} & \cdots & \varphi_{1,M} \\ \vdots & \ddots & \vdots \\ \varphi_{i,1} & \cdot\cdot & \varphi_{i,M} \\ \vdots & \ddots & \vdots \\ \varphi_{N,1} & \cdots & \varphi_{N,M} \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \vdots \\ \lambda_M \end{bmatrix} = \begin{bmatrix} h_1 \\ \vdots \\ \vdots \\ \vdots \\ h_N \end{bmatrix} \qquad \boldsymbol{Ax} = \boldsymbol{b}$$

Soustava rovnic je tedy přeurčenou soustavou, tj. je více rovnic než neznámých. Takovou soustavu můžeme řešit metodou nejmenších čtverců („Least Square Method“- LSM) nebo dekompozicí singulárních hodnot („Singular Value Decomposition“ - SVD).

Výhodou tohoto přístupu je, že volba „virtuálně“ vložených bodů je libovolná, a tedy lze snadno zlepšit přesnost aproximace v oblasti uživatelova zájmu, resp. lépe vystihnout vlastnosti a chování dat. Toto je zejména vítaná vlastnost v inženýrských aplikacích a aplikacích GIS.

Radiální bázové funkce se dnes používají i v jiných oblastech, např. pro řešení parciálních diferenciálních rovnic. Jejich výhodou je, že nepotřebují vytváření sítí, na druhé straně vzniká komplikace s ostrými přechody apod.

10. Parametrické křivky a plochy

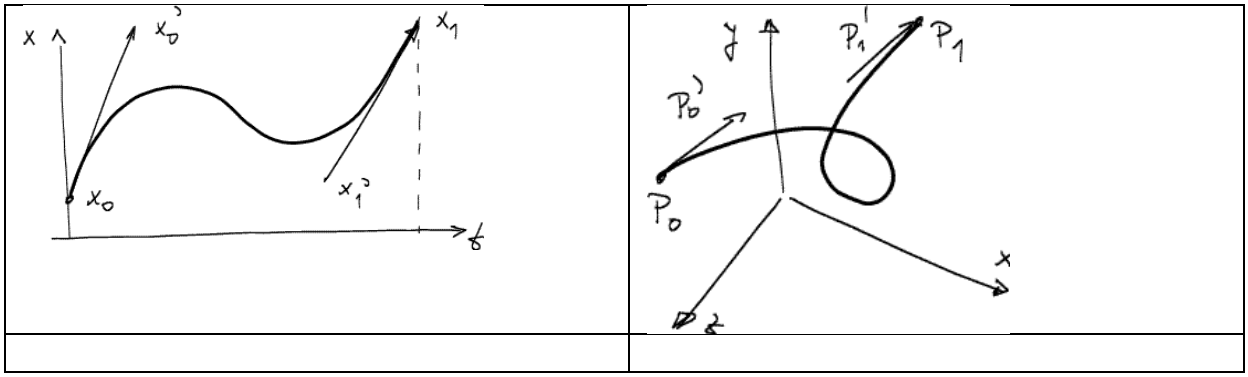
Parametrický popis je poměrně mocným a flexibilním popisek k popisu objektů, resp. jejich povrchů. Nicméně jsou poměrně komplikované některé „běžné“ operace. Uveďme alespoň běžné operace, např. průsečík přímky s parametrickou křivkou nebo rovinou, průsečík roviny s parametrickou plochou atd.

10.1. Parametrické křivky

Jedna třída běžných parametrických interpolací je založena na kubických parametrických křivkách a bikubických parametrických plátech. Pravděpodobně nejčastěji jsou používány křivky a pláty založené na Hermitově, Bézierově a Coonsově formulaci.

Hermitova (Fergusonova) křivka

Hermitova křivka je určena souřadnicemi počátečního a koncového bodu křivky a tečnými vektory v těchto bodech. Na obr.xx je křivka nakreslena pro souřadnici x, obr.x.b pak ukazuje křivku v prostoru E^3. V následujícím budeme používat značení $P = (x, y, z)$.



Pro pochopení uveďme jednoduché odvození Hermitovy křivky. Pokud chceme prokládat hladkou křivkou, pak křivka musí být alespoň 2. stupně. Pro proložení více body potřebujeme polynom vyššího stupně, nebo polynomy nižšího stupně, které budeme napojovat. Lze ukázat, že 2 křivky 2. stupně nelze obecně hladce napojit tak, aby v místě napojení nebyl „zlom“, tj. aby alespoň existovala první derivace v bodě napojení. Proto křivky musí být alespoň 3. stupně, tj. mající inflexní bod. Hermitova formulace vychází ze znalosti koncových bodů a směrnic křivky v koncových bodech. Lze tedy pro parametrickou křivku $x(t)$ psát:

$x(t) = at^3 + bt^2 + ct + d$	$x'(t) = 3at^2 + 2bt + c$

Dosazením za $t = 0$ a $t = 1$ dostáváme 4 rovnice pro 4 neznámé, tj. a, b, c, d:

$x(0) = d$	$x'(0) = c$
$x(1) = a + b + c + d$	$x'(1) = 3a + 2b + c$

Takže musíme vyřešit soustavu rovnic:

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} x(0) \\ x(1) \\ x'(0) \\ x'(1) \end{bmatrix}$$

Řešením této soustavy, tj. $\boldsymbol{Ax} = \boldsymbol{b}$, dostáváme koeficienty Hermitovy formy a, b, c, d a můžeme psát

$$x(t) = at^3 + bt^2 + ct + d = \boldsymbol{x}^T \boldsymbol{A}^{-1}\, \boldsymbol{t} \; = \boldsymbol{x}^T \boldsymbol{M}_H \boldsymbol{t}$$

kde $\boldsymbol{t} = [t^3, t^2, t, 1]^T$ a $\boldsymbol{x} = [x(0), x(1), x'(0), x'(1)]^T$.

Inverzní matice $\boldsymbol{A}^{-1}$ je pak v našem případě maticí Hermitovy formy $\boldsymbol{M}_H$, tj.

$$\boldsymbol{M}_H = \begin{bmatrix} 2 & -3 & 0 & 1 \\ -2 & 3 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 1 & -1 & 0 & 0 \end{bmatrix}$$

Obdobně postupujeme i pro ostatní souřadnice, tj. pro y , resp. z atd. Pro usnadnění zápisu se používá symbol P, který pak vlastně zatupuje jednotlivé souřadnice obecně v d-rozměrném prostoru. Je vhodné zdůraznit, že matice je pro danou formu konstantní.

Interpolační funkce (blending functions) $\boldsymbol{g}_H(t)$ jsou pak dány výrazem:

$$\boldsymbol{g}_H(t) = [g_0(t), g_1(t), g_2(t), g_3(t)]^T = \boldsymbol{M}_H \ [t^3, t^2, t, 1]^T \quad t \in \langle 0,1 \rangle$$

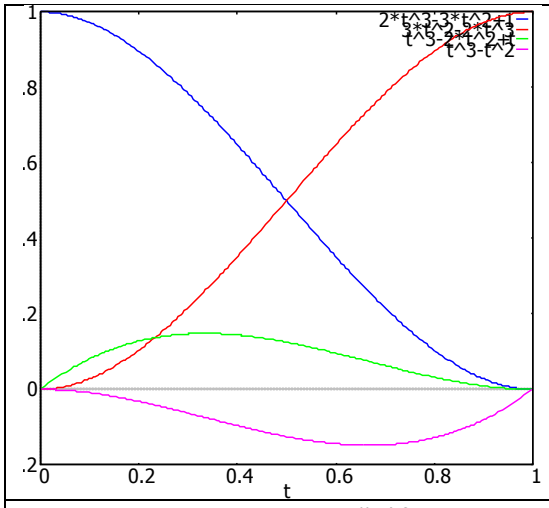
Rozepsáním maticové formy dostáváme:

$g_0(t) = 2t^3 - 3t^2 + 1$	$g_1(t) = -2t^3 + 3t^2$	$g_2(t) = t^3 - 2t^2 + t$	$g_3(t) = t^3 - t^2$

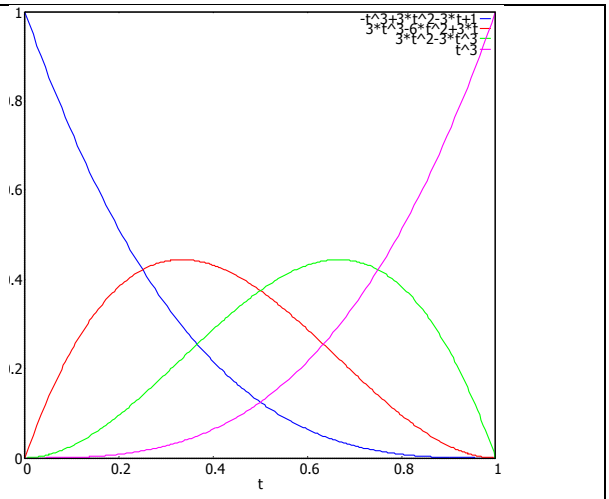
Bod křivky $P(t) = (x, y, z)$ je určen:

$$P(t) = [P_0, P_1, P_0', P_1'] \ \boldsymbol{M}_H \ [t^3, t^2, t, 1]^T \quad t \in \langle 0,1 \rangle$$

kde $\boldsymbol{M}_H$ je matice Hermitovy formy.



Obr.XX: Hermitovy interpolační funkce



Obr.XX: Bézierovy interpolační funkce

Bézierova křivka

Bézierova křivka obecně n-tého stupně je určena:

$P(t) = \sum_{i=0}^{n} B_{n,i} P_i = \sum_{i=0}^{n} \binom{n}{i} P_i \ t^i (1-t)^{n-i}$	$B_{n,i} = \binom{n}{i} \ t^i (1-t)^{n-i}$

kde: $B_{n,i}$ jsou Bernsteinovy polynomy.

Pro 3.stupeň je pak Bézierova křivka určena 4 body, viz obr.xx, a křivka je určena rovnicí:

$$P(t) = [P_0, P_1, P_2, P_3] \ \boldsymbol{M}_B \ [t^3, t^2, t, 1]^T \quad t \in \langle 0,1 \rangle$$

Matice Bézierovy formy je definována:

$$\boldsymbol{M}_B = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Interpolační funkce (blending functions) $\boldsymbol{g}_B(t)$ jsou dány výrazem:

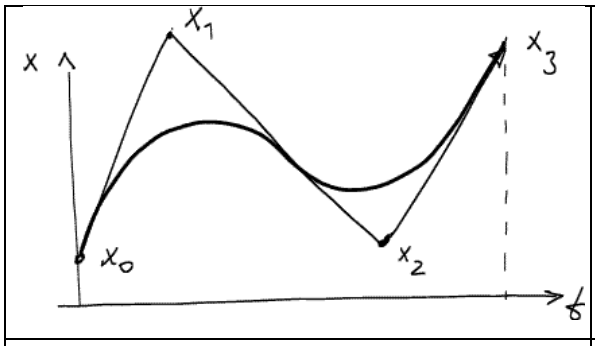
$$\boldsymbol{g}_B(t) = [g_0(t), g_1(t), g_2(t), g_3(t)]^T = \boldsymbol{M}_B \ [t^3, t^2, t, 1]^T \quad t \in \langle 0,1 \rangle$$

Rozepsáním maticové formy dostáváme interpolační funkce pro Bézierovu křivku:

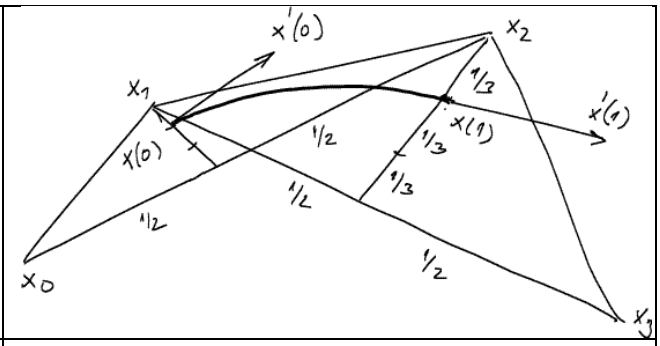
$g_0(t) = -t^3 + 3t^2 - 3t + 1$	$g_1(t) = 3t^3 - 6t^2 + 3t$
$g_2(t) = -3t^3 + 3t^2$	$g_3(t) = t^3$

Upozornění

Důležitou vlastností Bézierovy křivky je, že pro $t \in \langle 0,1 \rangle$ je křivka pro každou souřadnici vždy uvnitř **konvexní obálky** daných 4 řídících bodů. Tato vlastnost je dána tím, že součet interpolačních funkcí je roven hodnotě 1 pro všechny hodnoty $t \in \langle 0,1 \rangle$. To znamená, že všechny body křivky pro $t \in \langle 0,1 \rangle$ jsou uvnitř konvexní obálky dané řídícími body Bézierovy křivky. V případě křivky v E^3 je tedy křivka uvnitř tetrahedronu, který je dán čtyřmi řídícími body křivky.



Obr.XXX: Bézierova křivka



Obr.xxx: Coonsova křivka

Poznámka

Pozice řídících bodů v parametrickém prostoru, tj. (x,t), jsou $(x_0,0)$,$(x_1,1/3)$, $(x_2,2/3)$, $(x_3,1)$. Analogicky pro ostatní souřadnice.

Kubická Hermitova nebo Bézierova křivka je určena dvěma body, kterými křivka prochází, a další dvě řídící hodnoty určují její tvar, jde tedy o *interpolační křivky*. Existují však i jiné formulace parametrických křivek, založené na *aproximaci,* tj. křivka danými body obecně neprochází, např. Coonsova křivka, která patří do B-Spline parametrických křivek.

Coonsova křivka

Další parametrickou křivkou je Coonsova křivka, která patří do skupiny B-Spline interpolací, nicméně jde vlastně o aproximační křivku. Křivka je opět dána čtyřmi řídícími body a je určena rovnicí:

$$P(t) = [P_0, P_1, P_2, P_3]\, \boldsymbol{M}_C\ [t^3, t^2, t, 1]^T \quad t \in \langle 0,1 \rangle$$

Tato křivka, na rozdíl od předchozích křivek, neprochází počátečním a koncovým řídícím bodem. Jde tedy vlastně o *aproximační* křivku. Matice Coonsovy formy je definována:

$$\boldsymbol{M}_C = \frac{1}{6}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 0 & 4 \\ -3 & 3 & 3 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

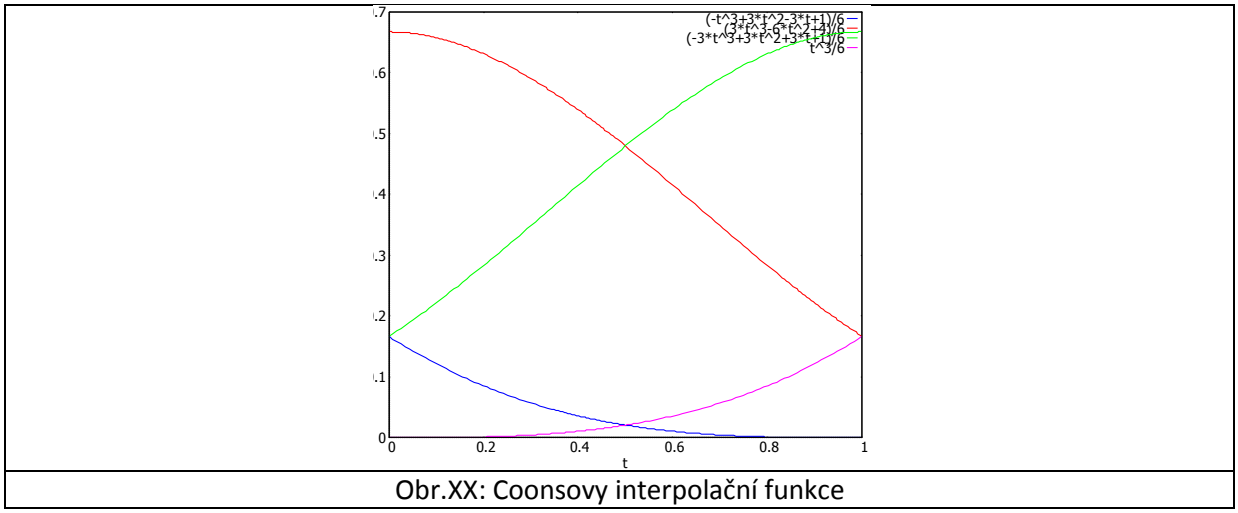
V případě interpolace pro více bodů se Coonsovy křivky hladce C^2 napojují. V případě Hermitovy a Bézierovy křivky jde o výpočetní postup. Jednotlivé segmenty Coonsovy křivky se hladce napojují vlastně pouhým „překrýváním“ po jednotlivých intervalech, tj. intervalů i a $i+1$.

Interpolační funkce Coonsovy formy $\boldsymbol{g}_C(t)$ jsou dány výrazem:

$$\boldsymbol{g}_C(t) = \boldsymbol{M}_C\ [t^3, t^2, t, 1]^T \quad t \in \langle 0,1 \rangle$$

Rozepsáním pak:

$g_0(t) = (-t^3 + 3t^2 - 3t + 1)/6$	$g_1(t) = (3t^3 - 6t^24)/6$
$g_2(t) = (-3t^3 + 3t^2 + 3t + 1)/6$	$g_3(t) = t^3/6$



Obr.XX: Coonsovy interpolační funkce

Závěr

Všechny výše uvedené křivky popsat vztahem:

$$P(t) = [P_0, P_1, P_3, P_4]\ \boldsymbol{M}_H\ [t^3, t^2, t, 1]^T \quad t \in \langle 0,1 \rangle$$

kde: $\boldsymbol{M}_F$ je matice příslušné formy a P_i jsou *řídící hodnoty* podle specifikace příslušné formy.

Interpolační (blending) funkce $\boldsymbol{g}_F(t)$ jsou ve všech případech dány výrazem:

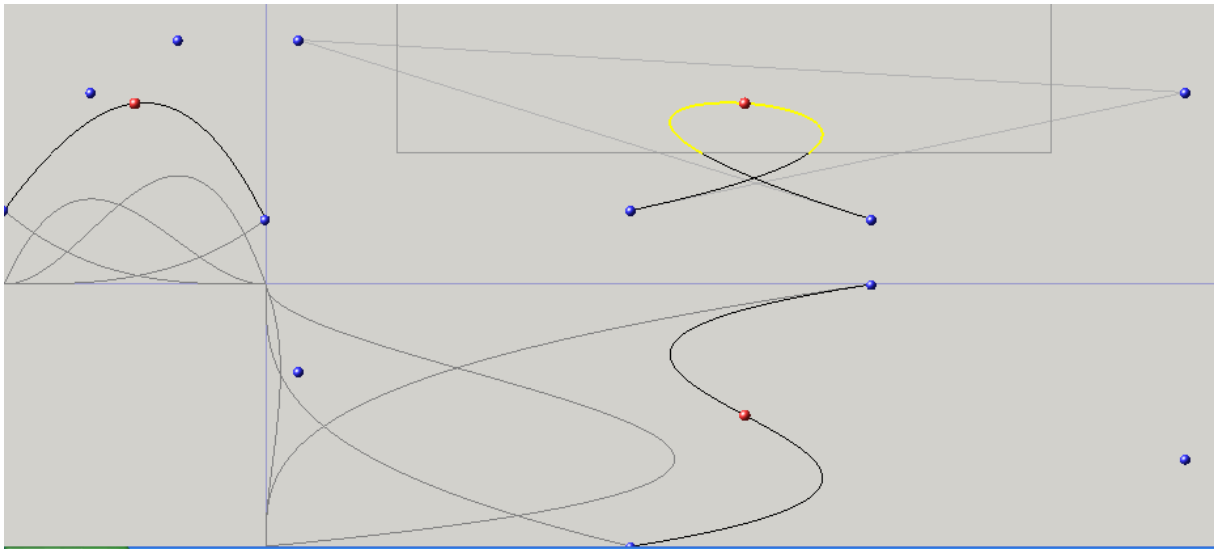
$$\boldsymbol{g}_F(t) = \boldsymbol{M}_F\ [t^3, t^2, t, 1]^T \quad t \in \langle 0,1 \rangle$$

Je tedy zřejmé, že popis parametrické interpolace kubickými křivkami je poměrně jednoduchý.

Je možná reparametrizace $t = \varphi(\tau)$ a tím ovlivnit „rychlost“ pohybu po křivce. Musí platit $\varphi(0) = 0$, $\varphi(1) = 1$, $\varphi(\tau_1) < \varphi(\tau_2)$ pro $\tau_1 < \tau_2$, tj. funkce $\varphi(\tau)$ musí být monotónně rostoucí pro $\tau \in \langle 0,1 \rangle$.

Příklad

Ukázka chování Bézierovy křivky v závislosti na řídících bodech.



Obr.xxx: Béziérova křivka – manipulace (CLICK-EXE)

Adresář *Bezier* obsahuje soubory nutné ke spuštění programu, který umožňuje manipulaci s Bézierovou křivkou.

10.2. Vykreslování parametrických křivek

Vykreslování parametrických křivek se zdá být poměrně jednoduchou záležitostí. Naivní řešení, které je velmi často realizováno, je založeno na zvolení přírůstku parametru, např. $\Delta t = 0.01$, a kreslení lomené čáry. Je nutné si uvědomit, že:

- pro konstantní přírůstek parametru Δt, není konstantní délka jednotlivých segmentů lomené čáry
- délka vykreslované křivky pro interval parametru $t \in \langle t_0, t_1 \rangle$ je určena nelineárním vztahem:

$$l = \int_{t_0}^{t_1} \sqrt{x'^2(t) + y'^2(t)}\ dt$$

  V případě křivky v E^3 je délka křivky pro $t \in \langle t_0, t_1 \rangle$ určena:

$$l = \int_{t_0}^{t_1} \sqrt{x'^2(t) + y'^2(t) + z'^2(t)}\ dt$$

  Pro celou křivku, kdy $t \in \langle 0,1 \rangle$ dostáváme:

$$l = \int_{0}^{1} \sqrt{x'^2(t) + y'^2(t) + z'^2(t)}\ dt$$

- pokud má být křivka vykreslena úseky o stejné délce, je nutné najít odpovídající hodnoty $t_0, t_1, \ldots, t_n$. Nicméně, aby vykreslená křivka byla hladká, bude nutné volit intervaly $\langle t_i, t_{i+1} \rangle$ dostatečně „malé“ a podle křivosti apod.

Je tedy zřejmé, že korektní a „dobré“ vykreslování křivky je poněkud složitější, než by se očekávalo. Podrobněji viz předměty Geometrické modelování apod.

Pokud máme určeny intervaly $\langle t_i, t_{i+1} \rangle$, např. pomocí $\Delta t = 0.01$, pak je otázkou, jak křivku vykreslit. Z předchozího je známo, že kubická křivka je dána výrazem:

$$P(t) = [P_0, P_1, P_2, P_3]\ \boldsymbol{M}_C\ [t^3, t^2, t, 1]^T \quad t \in \langle 0,1 \rangle$$

To znamená, že výraz

$$[P_0, P_1, P_2, P_3]\ \boldsymbol{M}_C$$

je konstantní pro všechny hodnoty t, takže dostáváme standardní mocninou řadu s konstantními parametry a její hodnota se vypočte Hornerovým schématem, a to:

$$S = \Big(\big((a_n t + a_{n-1})t + a_{n-2}\big)t + \cdots +\Big)\, t + a_0$$

Není tedy nutné počítat mocniny pro parametr t.

Další algoritmy pro vykreslování

Zajímavým algoritmem vykreslování kubických parametrických křivek je algoritmus *de Casteljau*, který je založen na postupném dělení úseček řídícího n-úhelníka Bézierovy formy. Další možností je použití diferenčního schématu, které je založeno na tom, že druhá diference pro Bézierovu křivku 3.stupně je konstantní, viz doporučená literatura.

Je zřejmé, že různé formy kubických křivek mají jisté výhody a nevýhody. Je otázkou, zda a jak lze jednotlivé formy vzájemně transformovat.

10.3. Transformace kubických parametrických křivek

Všechny výše uvedené kubické křivky jsou po formální stránce reprezentované jednou rovnicí, a to:

$$\boldsymbol{P}(t) = [P_0, P_1, P_2, P_3]\, \boldsymbol{M}_F\, [t^3, t^2, t, 1]^T \quad t \in \langle 0,1 \rangle$$

nebo:

$$\boldsymbol{P}(t) = [P_0, P_1, P_2, P_3]\, \boldsymbol{M}_F\, \boldsymbol{t} \quad t \in \langle 0,1 \rangle$$

kde: $\boldsymbol{t} = [t^3, t^2, t, 1]^T$ a $t \in \langle 0,1 \rangle$. Je tedy zřejmé, že po formální stránce lze jednotlivé reprezentace mezi sebou po formální stránce transformovat. Pokud máme dvě křivky, např. Hermitovu a Bézierovu, pak lze psát pro x souřadnici pro Hermitovu křivku:

$$x_H(t) = \left[{}^H x_0,\ {}^H x_1,\ {}^H x_0',\ {}^H x_1' \right] \boldsymbol{M}_H\, [t^3, t^2, t, 1]^T$$

a pro Bézierovu křivku:

$$x_B(t) = \left[{}^B x_0,\ {}^B x_1,\ {}^B x_2,\ {}^B x_3 \right] \boldsymbol{M}_B\, [t^3, t^2, t, 1]^T$$

Pokud chceme mít geometricky stejnou parametrickou křivku v obou případech, pak musí platit:

$$x_H(t) = x_B(t)$$

tj.

$$\left[{}^H x_0,\ {}^H x_1,\ {}^H x_0',\ {}^H x_1' \right] \boldsymbol{M}_H\, [t^3, t^2, t, 1]^T = \left[{}^B x_0,\ {}^B x_1,\ {}^B x_2,\ {}^B x_3 \right] \boldsymbol{M}_B\, [t^3, t^2, t, 1]^T$$

a tedy:

$$\left[{}^H x_0,\ {}^H x_1,\ {}^H x_0',\ {}^H x_1' \right] \boldsymbol{M}_H = \left[{}^B x_0,\ {}^B x_1,\ {}^B x_2,\ {}^B x_3 \right] \boldsymbol{M}_B$$

Pro převod z Hermitovy do Bézierovy formy pak dostáváme:

$$\left[{}^B x_0,\ {}^B x_1,\ {}^B x_2,\ {}^B x_3 \right] \boldsymbol{M}_B = \left[{}^H x_0,\ {}^H x_1,\ {}^H x_0',\ {}^H x_1' \right] \boldsymbol{M}_H$$

Matice $\boldsymbol{M}_H$ je regulární a tedy rovnici lze násobit maticí inverzní a pak:

$$\left[{}^B x_0,\ {}^B x_1,\ {}^B x_2,\ {}^B x_3 \right] = \left[{}^H x_0,\ {}^H x_1,\ {}^H x_0',\ {}^H x_1' \right] \boldsymbol{M}_H\, \boldsymbol{M}_B^{-1}$$

Zkráceně můžeme psát:

$$\boldsymbol{M}_{H \to B} = \boldsymbol{M}_H\, \boldsymbol{M}_B^{-1}$$

Je zřejmé, že transformační matice $\boldsymbol{M}_{H \to B}$ je platná také pro souřadnice y a z.

Vzájemné převody jsou tedy možné a vzájemné transformační matice jsou uvedeny v následující tabulce. Transformace jsou výhodné např. pro interaktivní modifikaci, kdy uživatelská interakce probíhá např. v Bézierově formě, zatímco výpočty hladkosti probíhají v Hermitově formě apod.

	Z	Hermite	Bézier	Coons
Do	Hermite	------	$\begin{bmatrix} 1 & 0 & -3 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & -3 \\ 0 & 1 & 0 & 3 \end{bmatrix}$	$\frac{1}{6}\begin{bmatrix} 1 & 0 & -3 & 0 \\ 4 & 1 & 0 & -3 \\ 1 & 4 & 3 & 0 \\ 0 & 1 & 0 & 3 \end{bmatrix}$
	Bézier	$\frac{1}{3}\begin{bmatrix} 3 & 3 & 0 & 0 \\ 0 & 0 & 3 & 3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix}$	------	$\frac{1}{6}\begin{bmatrix} 1 & 0 & 0 & 0 \\ 4 & 4 & 2 & 1 \\ 1 & 2 & 4 & 4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
	Coons	$\frac{1}{3}\begin{bmatrix} -3 & 6 & -3 & 6 \\ 6 & -3 & 6 & -3 \\ -7 & 2 & -1 & 2 \\ -2 & 1 & -2 & 7 \end{bmatrix}$	$\begin{bmatrix} 6 & 0 & 0 & 0 \\ -7 & 2 & -1 & 2 \\ 2 & -1 & 2 & -7 \\ 0 & 0 & 0 & 6 \end{bmatrix}$	------

Tab.XXX: Matice vzájemných převodů

10.4. Hladké napojování kubických křivek

Jako příklad uveďme napojování křivek pro Hermitovu formu. Představme si, že chceme interpolovat pohyb objektu po nějaké složitější prostorové křivce (trajektorii).

Předpokládejme, že pohyb objektu je popsán po částech kubickými křivkami

$P(t) = \boldsymbol{P}^T \boldsymbol{M}_H \boldsymbol{t}$	$\boldsymbol{P} = [P_0, P_1, P_0', P_1']^T$	$\boldsymbol{t} = [t^3, t^2, t, 1]^T$

Je zřejmé, že můžeme mít následující požadavky, a to:

- C^0 spojitost - objekt nemění skokem svoji **pozici**, tedy koncový bod 1.segmentu je stejný jako počáteční pozice 2.segmentu, tj. ${}^{(1)}P(1) = {}^{(2)}P(0)$, resp. obecně ${}^{(i-1)}P(1) = {}^{(i)}P(0)$
- C^1 spojitost - objekt nemění skokem svůj směr, tedy že časová derivace pozice koncového bodu 1.segmentu je stejná jako počáteční časová derivace pozice 2.segmentu, tj. $\frac{d}{dt}{}^{(i)}P(1) = \frac{d}{dt}{}^{(2)}P(0)$, resp. obecně ${}^{(i-1)}P'(1) = {}^{(i)}P'(0)$
- C^2 spojitost – rychlost se nemění skokem $\frac{d^2}{dt^2}{}^{(1)}P(1) = \frac{d^2}{dt^2}{}^{(2)}P(0)$, resp. obecně $\frac{d^2}{dt^2}{}^{(i-1)}P(1) = \frac{d^2}{dt^2}{}^{(i)}P(0)$

Hladké napojení Hermitových křivek

Hladké napojení Hermitovy křivky C^2 spojením segmentů ${}^{(1)}P(t)$ a ${}^{(2)}P(t)$:

$P(t) = \boldsymbol{P}^T \boldsymbol{M}_H \boldsymbol{t}$	$\boldsymbol{P} = [P_0, P_1, P_0', P_1']^T$	$\boldsymbol{t} = [t^3, t^2, t, 1]^T$

Pro:

$P(t) = at^3 + bt^2 + ct + d$	$\frac{d}{dt}P(t) = 3at^2 + 2bt + c$	$\frac{d^2}{dt^2}P(t) = 6at + 2a$

${}^{(i-1)}P''(1) = {}^{(i)}P''(0)$	vede na podmínku:	$6\,{}^{(i-1)}a + 2\,{}^{(i-1)}b = 2\,{}^{(i)}b$

$a = P(0)$	$b = P'(0)$
$c = -3P(0) + 3P(1) - 2P'(0) - P'(1)$	$d = -2P(0) - 2P(1) + P'(0) + P'(1)$

Pokud podmínky spojíme dohromady spolu s podmínkou ${}^{(1)}P(1) = {}^{(2)}P(0)$ dostáváme:

$$2\left[3\left({}^{(i)}P - {}^{(i-1)}P\right) - 2\,{}^{(i-1)}P' - {}^{(i)}P'\right] + 6\left[2\left({}^{(i-1)}P - {}^{(i)}P\right) + 2\,{}^{(i-1)}P' + {}^{(i)}P'\right]$$
$$= 2\left[3\left({}^{(i+1)}P - {}^{(i)}P\right) - 2\,{}^{(i)}P' - {}^{(i+1)}P'\right]$$

Zjednodušením pak:

$${}^{(i-1)}P' + 4\,{}^{(i)}P' + {}^{(i+1)}P' = 3\left({}^{(i+1)}P - {}^{(i-1)}P\right)$$

a v maticové formě pak:

$$\begin{bmatrix} 1 & 0 & & & & \\ 1 & 4 & 1 & 0 & & \\ 0 & 1 & 4 & 1 & 0 & \\ & & & \ddots & & \\ & & 0 & 1 & 4 & 4 \\ & & & & 0 & \end{bmatrix} \begin{bmatrix} {}^{(0)}P' \\ {}^{(1)}P' \\ \vdots \\ \vdots \\ {}^{(m-2)}P' \\ {}^{(m-1)}P' \end{bmatrix} = \begin{bmatrix} {}^{(0)}P' \\ 3\left({}^{(2)}P - {}^{(0)}P\right) \\ \\ \\ 3\left({}^{(m-1)}P - {}^{(m-3)}P\right) \\ {}^{(m-1)}P' \end{bmatrix}$$

Pokud ${}^{(0)}P''(0) = {}^{(m)}P''(0) = 0$ pak jde o tzv. přirozený kubický spline (*natural cubic spline*).

Coonsova křivka

Coonsova křivka je aproximačního typu, tj. křivka neprochází zadanými body, neboť má pouze řídící body. Jednotlivé segmenty se napojují C^2 spojitě prostým překrýváním intervalů. Je však otázkou, jak

- křivku „dotáhnout“ do koncových bodů
- docílit „uzavřené“ křivky

Oba problémy jsou snadno řešitelné, pokud si uvědomíme vlastnosti Coonsovy křivky a konstrukci vykreslované křivky.

Konstrukce uzavřené křivky

Konstrukce uzavřené křivky je poměrně jednoduchá, pokud si uvědomíme, že tvar výsledné křivky nemůže záviset na pořadí zadávaných bodů. Takže uzavřená křivka daná Coonsovými kubikami je určena řídícími body jednotlivých segmentů (pořadí jednotlivých segmentů je vhodné dodržet, ale není to nutné) takto:

- standardní sekvence:
  $[\boldsymbol{x}_0, \boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3]$, $[\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4]$, $[\boldsymbol{x}_{n-4}, \boldsymbol{x}_{n-3}, \boldsymbol{x}_{n-2}, \boldsymbol{x}_{n-1}]$
- „uzavírací“ sekvence:
  $[\boldsymbol{x}_{n-3}, \boldsymbol{x}_{n-2}, \boldsymbol{x}_{n-1}, \boldsymbol{x}_0]$, $[\boldsymbol{x}_{n-2}, \boldsymbol{x}_{n-1}, \boldsymbol{x}_0, \boldsymbol{x}_1]$, $[\boldsymbol{x}_{n-1}, \boldsymbol{x}_0, \boldsymbol{x}_1, \boldsymbol{x}_2]$

Vidíme tedy, že postup je přímočarý.

Dotažení křivky do koncových bodů

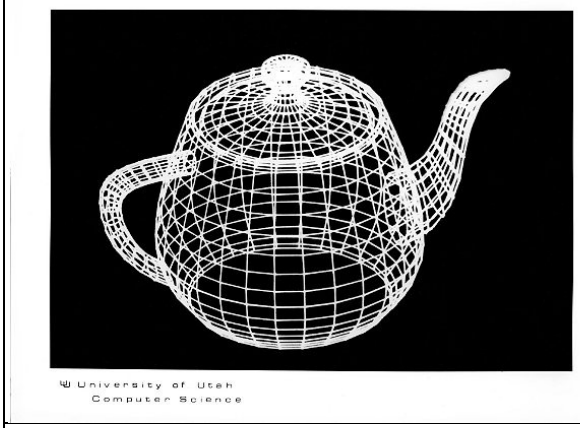
Konstrukce křivky, kdy chceme „dotáhnout“ Coonsovu křivku do koncových bodů, vychází z její konstrukce, kdy se vlastně vykresluje křivka „jen pod vnitřními řídícími body“. Postup je opět velmi jednoduchý, neboť je pouze nutné krajní body „duplikovat“, viz následující sekvence:

- úvodní sekvence:
  $[\boldsymbol{x}_0, \boldsymbol{x}_0, \boldsymbol{x}_0, \boldsymbol{x}_1]$ (křivka z bodu $\boldsymbol{x}_0$ do ½ hrany $\boldsymbol{x}_0\boldsymbol{x}_1$)
  $[\boldsymbol{x}_0, \boldsymbol{x}_0, \boldsymbol{x}_1, \boldsymbol{x}_2]$ (křivka z ½ hrany $\boldsymbol{x}_0\boldsymbol{x}_1$ do počátečního bodu standardně vykreslované křivky
- standardní sekvence:
  $[\boldsymbol{x}_0, \boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3]$, $[\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4]$, $[\boldsymbol{x}_{n-4}, \boldsymbol{x}_{n-3}, \boldsymbol{x}_{n-2}, \boldsymbol{x}_{n-1}]$
- „uzavírací“ sekvence analogická té úvodní:
  $[\boldsymbol{x}_{n-3}, \boldsymbol{x}_{n-2}, \boldsymbol{x}_{n-1}, \boldsymbol{x}_{n-1}]$ a
  $[\boldsymbol{x}_{n-2}, \boldsymbol{x}_{n-1}, \boldsymbol{x}_{n-1}, \boldsymbol{x}_{n-1}]$

10.5. Parametrické plochy

Parametrické plochy umožňují realizaci hladkých ploch popsaných jednoduchým matematickým aparátem. Většinou jsou parametry plochy u, v omezeny na interval $u, v \in \langle 0,1 \rangle$ a pro vyváření složitějších ploch se musí napojovat. Proto se takové plochy označují termínem *pláty* (patches) a proces jejich napojování pak termínem plátování. Dnes již klasickou ukázkou je slavný Utah Teapot (čajník z Utahu) Martina Newella, který je založen na Bézierově reprezentaci bikubického plátu.

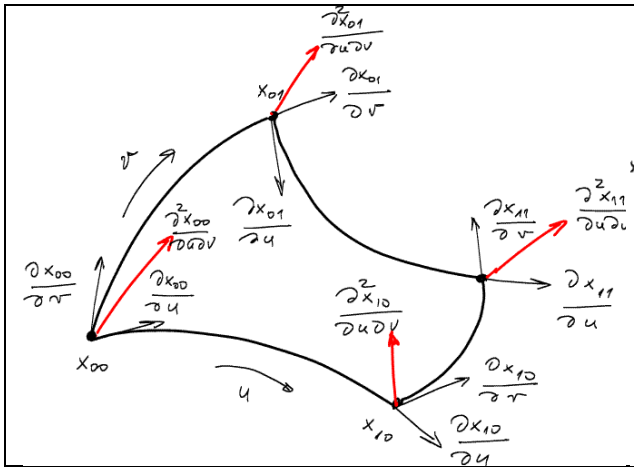
Čtyřúhelníkové pláty



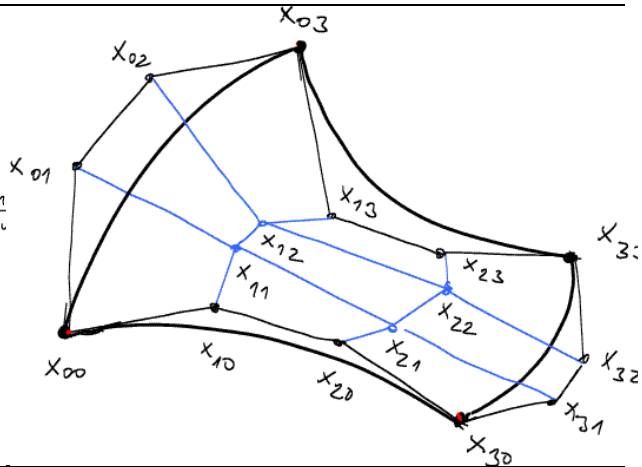
Původní data viz http://www.sjbaker.org/teapot/teaset.tgz

Java aplet viz http://mrl.nyu.edu/~perlin/experiments/teapot/

Parametrické bikubické pláty splňují podmínku, že pro $u = konst$ a $v = konst$ dostáváme dříve uvedené kubické křivky. Pro názornost uveďme pouze Hermitův a Bézierův bikubický plát:



Hermitův bikubický plát



Bézierův bikubický plát

Bikubické pláty můžeme popsat rovnicí :

$$P(u,v) = [u^3, u^2, u, 1]\, \boldsymbol{M}_F^T\, \boldsymbol{P}\, \boldsymbol{M}_F\, [v^3, v^2, v, 1]^T \quad u, v \in \langle 0,1 \rangle$$

kde $\boldsymbol{M}_F$ je matice příslušné formy, tj. Hermitovy, Bézierovy a Coonsovy, $\boldsymbol{P}$ je matice 4×4 řídících hodnot pro každou souřadnici, tj. x, y, z, a u a v jsou parametry bikubického plátu.

$\boldsymbol{P}_H = \begin{bmatrix} \begin{bmatrix} P_{00} & x_{01} \\ P_{10} & x_{11} \end{bmatrix} & \frac{\partial}{\partial v}\begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix} \\ \frac{\partial}{\partial u}\begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix} & \frac{\partial^2}{\partial u \partial v}\begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix} \end{bmatrix}$	$\boldsymbol{P}_B = \begin{bmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{bmatrix}$
Matice řídících hodnot Hermitovy formy	Matice řídících hodnot Bézierovy formy
$\boldsymbol{X}_C = \begin{bmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{bmatrix}$	
Matice řídících hodnot Coonsovy formy	

V případě parametrických kubických křivek byla odvozena z podmínek jejich maticová forma, viz kap.10.1 (Parametrické křivky), kdy bod parametrické křivky $P(t) = (x, y, z)$ je určen:

$$P(t) = [P_0, P_1, P_0', P_1']\, \boldsymbol{M}_H\, [t^3, t^2, t, 1]^T \quad t \in \langle 0,1 \rangle$$

kde $\boldsymbol{M}_H$ je matice Hermitovy formy. Matice Hermitovy formy je definována

$$\boldsymbol{M}_H = \begin{bmatrix} 2 & -3 & 0 & 1 \\ -2 & 3 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 1 & -1 & 0 & 0 \end{bmatrix}$$

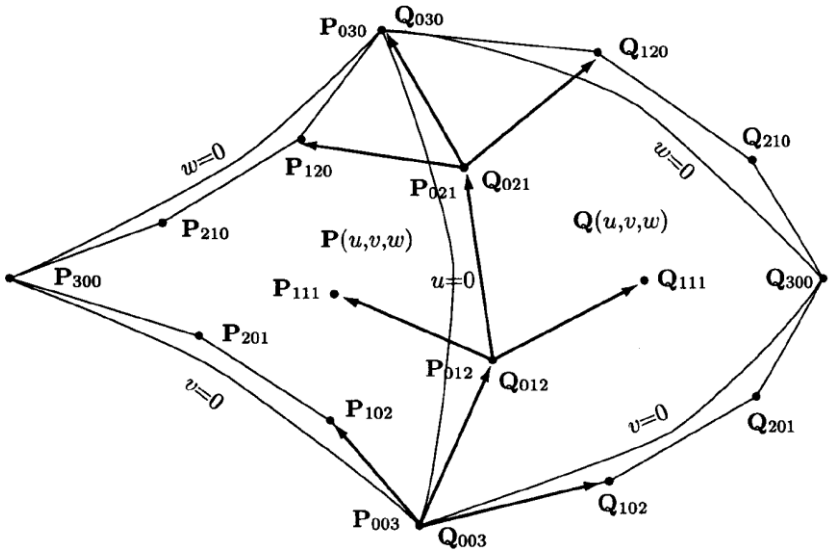
Trojúhelníkové pláty

De Casteljau, který pracoval v Citroënu, vyvinul parametrický trojúhelníkový plát před plátem čtyřúhelníkovým již v roce 1959. Kontrolní body jdou indexovány s indexy i, j, k přičemž platí, že $0 \leq i, j, k \leq n$ a $i + j + k = n$ a body na plátu jsou určeny vztahem:

$$P(u, v, w) = \sum_{i+j+k=n} P_{ijk} \frac{n!}{i!\, j!\, k!} u^i v^j w^k \qquad u + v + w = 1$$

Hodnotu n je určena uživatelem podle složitosti povrchu. V počítačové grafice je většinou $n = 3$. Plát je funkcí dvou parametrů, neboť platí podmínka $u + v + w = 1$.

Podrobněji viz: Salomon,D.: Computer Graphics and Geoemtric Modeling, Springer, str.483, 1999



Obr. Trojúhelníkový parametrický plát a jeho napojování.

Odvození rovnic bikubického plátu pro Hermitovu formu

Při odvození rovnic pro parametrickou bikubickou plochu v Hermitově formě budeme uvažovat pouze souřadnici $x(u,v)$, která je parametrizována nyní dvěma parametry u a v, neboť jde nyní o plochu.

Základními požadavky na bikubickou parametrickou plochu v Hermitově formě jsou:

- pro parametry platí podmínka $u \in \langle 0,1\rangle$ & $v \in \langle 0,1\rangle$
- bikubický plát má 4 vrcholy
- všechny křivky pro $u \in \langle 0,1\rangle$ & $v = konst$ a $v \in \langle 0,1\rangle$ & $u = konst$ včetně křivek hraničních, tj. pro $u = 0$ nebo $v = 0$, jsou také kubické křivky Hermitovy formy

Hermitova křivka pro danou hodnotu parametru v je dána vztahem:

$$x(u) = [x_0, x_1, x_0', x_1']\, \boldsymbol{M}_H\, [u^3, u^2, u, 1]^T \quad t \in \langle 0,1\rangle$$

nebo při použití transpozice:

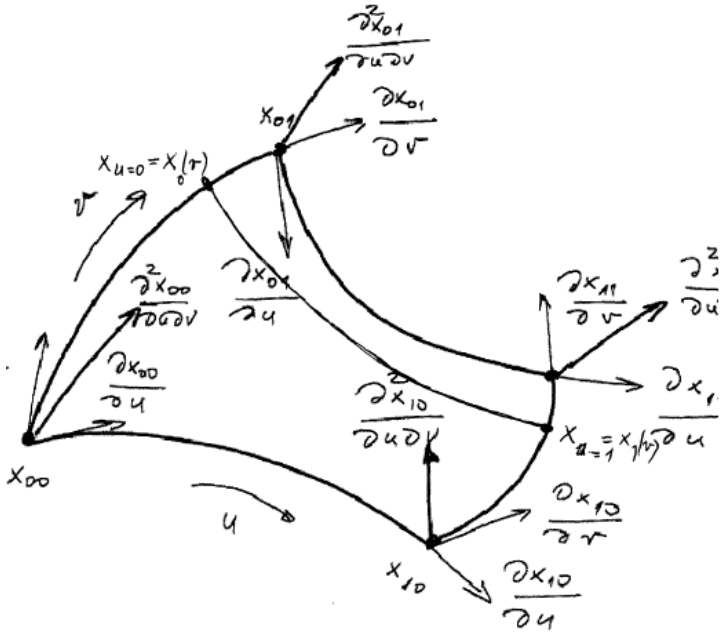
$$x(u) = [u^3, u^2, u, 1]\, \boldsymbol{M}_H^T\, [x_0, x_1, x_0', x_1']^T \quad t \in \langle 0,1\rangle$$

Křivka je určena koncovými body x_0, x_1 a derivacemi x_0', x_1', přičemž $x' = \frac{dx}{du}$, tj. derivace podle u. V případě bikubického plátu pak tyto derivace jsou derivace parciální podle u, tj. $x^{(u)} = \frac{\partial x}{\partial u}$.

V případě bikubického plátu jsou koncové body a jejich derivace parametrizovány parametrem v a tedy můžeme formálně psát

$$x(u,v) = [u^3, u^2, u, 1]\, \boldsymbol{M}_H^T\, [x_{u=0}(v), x_{u=1}(v), x_{u=0}^{(u)}(v), x_{u=1}^{(u)}(v)]^T \quad u,v \in \langle 0,1\rangle$$

přičemž křivky $x_{u=0}(v), x_{u=1}(v), x_{u=0}^{(u)}(v), x_{u=1}^{(u)}(v)$ jsou opět kubické křivky v Hermitově formě. V následujícím bude použito značení $x^{(v)} = \frac{\partial x}{\partial v}$ a $x^{(u,v)} = \frac{\partial^2 x}{\partial u \partial v}$



Obr. XXX: Hermitův bikubický plát

Takže můžeme psát pro:

- pro $u = 0$

$$x_{u=1}(v) = \left[x_{00}, x_{01}, x_{00}^{(u)}, x_{01}^{(u)}\right] \boldsymbol{M}_H \ [v^3, v^2, v, 1]^T$$

- pro $u = 1$

$$x_{u=1}(v) = \left[x_{10}, x_{11}, x_{10}^{(u)}, x_{11}^{(u)}\right] \boldsymbol{M}_H \ [v^3, v^2, v, 1]^T$$

Nyní je nutné vyjádřit ještě parciální derivace pro křivky $x_{u=0}^{(u)}(v), x_{u=1}^{(u)}(v)$. Lze nahlédnut, že:

- pro $u = 0$

$$x_{u=0}^{(u)}(v) = \left[x_{00}^{(u)}, x_{01}^{(u)}, x_{00}^{(u,v)}, x_{01}^{(u,v)}\right] \boldsymbol{M}_H \ [v^3, v^2, v, 1]^T$$

- pro $u = 1$

$$x_{u=1}^{(u)}(v) = \left[x_{10}^{(u)}, x_{11}^{(u)}, x_{10}^{(u,v)}, x_{11}^{(u,v)}\right] \boldsymbol{M}_H \ [v^3, v^2, v, 1]^T$$

Parametrickou bikubickou plochu v Hermitově formě lze tedy zapsat ve tvaru:

$$x(u,v) = [u^3, u^2, u, 1] \ \boldsymbol{M}_H^T \ \boldsymbol{X}_H \ \boldsymbol{M}_H \ [v^3, v^2, v, 1]^T \quad u, v \in \langle 0,1 \rangle$$

kde:

$$\boldsymbol{X}_H = \begin{bmatrix} x_{00} & x_{01} & x_{00}^{(u)} & x_{01}^{(u)} \\ x_{10} & x_{11} & x_{10}^{(u)} & x_{11}^{(u)} \\ x_{00}^{(u)} & x_{01}^{(u)} & x_{00}^{(u,v)} & x_{01}^{(u,v)} \\ x_{10}^{(u)} & x_{11}^{(u)} & x_{10}^{(u,v)} & x_{11}^{(u,v)} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\Sigma} & \dfrac{\partial \boldsymbol{\Sigma}}{\partial v} \\ \dfrac{\partial \boldsymbol{\Sigma}}{\partial u} & \dfrac{\partial^2 \boldsymbol{\Sigma}}{\partial u \partial v} \end{bmatrix}$$

kde matice $\boldsymbol{\Sigma}$ je definována:

$$\boldsymbol{\Sigma} = \begin{bmatrix} x_{00} & x_{01} \\ x_{10} & x_{11} \end{bmatrix}$$

Matice $\frac{\partial^2 \boldsymbol{\Sigma}}{\partial u \partial v}$ je vlastně maticí zkrutů (Twist), neboť pro x, y, z jde vlastně o vektor zkrutů v jednotlivých vrcholech Hermitova plátu.

Rovnici pro bikubický parametrický plát v Hermitově formě pak dostáváme rovnici:

$$x(u,v) = \boldsymbol{u}^T \boldsymbol{M}_H^T \ \boldsymbol{X}_H \ \boldsymbol{M}_H \boldsymbol{v} \quad u, v \in \langle 0,1 \rangle$$

kde:

$\boldsymbol{u} = [u^3, u^2, u, 1]^T$	$\boldsymbol{v} = [v^3, v^2, v, 1]^T$

Obdobně pro ostatní souřadnice, tj. y, z:

$$y(u,v) = \boldsymbol{u}^T \boldsymbol{M}_H^T \ \boldsymbol{Y}_H \ \boldsymbol{M}_H \boldsymbol{v} \quad u, v \in \langle 0,1 \rangle$$
$$z(u,v) = \boldsymbol{u}^T \boldsymbol{M}_H^T \ \boldsymbol{Z}_H \ \boldsymbol{M}_H \boldsymbol{v} \quad u, v \in \langle 0,1 \rangle$$

Matice $\boldsymbol{X}_H$, $\boldsymbol{Y}_H$, $\boldsymbol{Z}_H$ jsou matice 4×4 řídících hodnot daného bikubického parametrického plátu.

Zkráceně pro reprezentaci bodu $P(u,v) = [x(u,v), y(u,v), z(u,v)]^T$ pak lze psát:

$$P(u,v) = \boldsymbol{u}^T \boldsymbol{M}_H^T \ \boldsymbol{P}_H \ \boldsymbol{M}_H \boldsymbol{v} \quad u, v \in \langle 0,1 \rangle$$

a tedy:

$$\boldsymbol{P}_H = \begin{bmatrix} \begin{bmatrix} P_{00} & x_{01} \\ P_{10} & x_{11} \end{bmatrix} & \dfrac{\partial}{\partial v} \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix} \\ \dfrac{\partial}{\partial u} \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix} & \dfrac{\partial^2}{\partial u \partial v} \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix} \end{bmatrix}$$

Pro reprezentaci plochy jedním bikubickým plátem je tak zapotřebí $3 \times 16 = 48$ hodnot.

V předchozím jsme ukázali, že převod mezi jednotlivými formami, tj. Hermite, Bézier a Coons, je dán lineárním vztahem, viz kap.10.3 (Transformace kubických parametrických křivek).

Převod z Hermitovy formy do Bézierovy je dán vztahem:

$$\boldsymbol{M}_{H\to B} = \boldsymbol{M}_H\, \boldsymbol{M}_B^{-1}$$

tedy:

$$\boldsymbol{M}_{H\to B}\boldsymbol{M}_B = \boldsymbol{M}_H$$

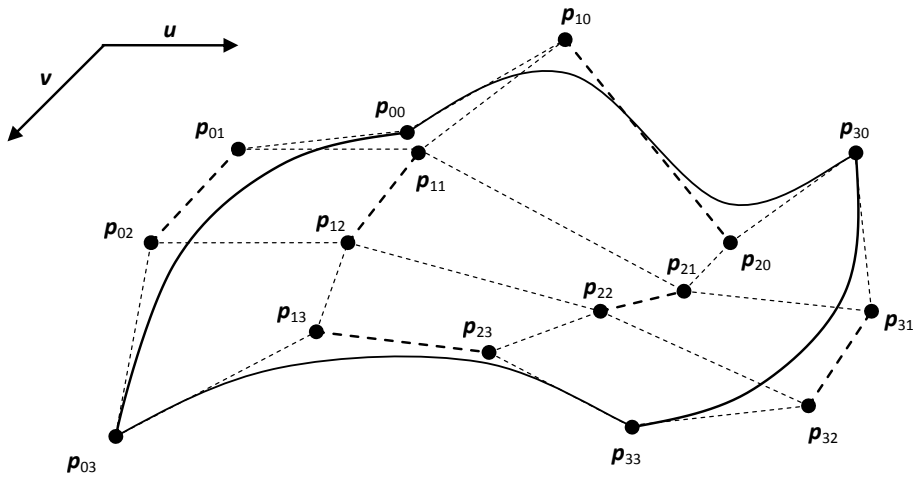
Rovnici pro Hermitovu formu můžeme přepsat do tvaru pro Bézierovu formu:

$$P(u,v) = \boldsymbol{u}^T \boldsymbol{M}_B^T \boldsymbol{M}_{H\to B}^T\, \boldsymbol{P}_H\, \boldsymbol{M}_{H\to B}\boldsymbol{M}_B\, \boldsymbol{v} = \boldsymbol{u}^T \boldsymbol{M}_B^T\, \boldsymbol{P}_B \boldsymbol{M}_B\, \boldsymbol{v}$$

kde: $\boldsymbol{P}_B = \boldsymbol{M}_{H\to B}^T\, \boldsymbol{P}_H\, \boldsymbol{M}_{H\to B}$ je matice řídících bodů pro Bézierovu formu, $\boldsymbol{M}_B$ je matice Bézierovy formy. Matice řídících bodů pro Bézierovu formu je pak pro souřadnici x dána:

$$\boldsymbol{X}_B = \begin{bmatrix} x_{00} & x_{01} & x_{02} & x_{03} \\ x_{10} & x_{11} & x_{12} & x_{13} \\ x_{20} & x_{21} & x_{22} & x_{23} \\ x_{30} & x_{31} & x_{32} & x_{33} \end{bmatrix}$$

a analogicky pak pro souřadnice y, z.



Obr.XXX: Bézierův čtyřúhelníkový plát

Obdobně pro Coonsův plat dostáváme rovnici:

$$P(u,v) = \boldsymbol{u}^T \boldsymbol{M}_C^T \boldsymbol{P}_C \boldsymbol{M}_C\, \boldsymbol{v}$$

Je tedy zřejmé, že bikubický parametrický čtyřúhelníkový plát je popsán rovnicí:

$$P(u,v) = \boldsymbol{u}^T \boldsymbol{M}_F^T \boldsymbol{P}_F \boldsymbol{M}_F\, \boldsymbol{v}$$

kde $\boldsymbol{M}_F$ je matice příslušné formy a $\boldsymbol{P}_F$ jsou matice řídících hodnot pro souřadnice x, y, z.

Upozornění

- Výhodou Bézierovy formy opět je, že bikubický plát je uzavřen v konvexním obalu řídících bodů Bézierova plátu, tj. konvexního obalu 16 bodů v E^3.
- U Coonsovy formy je dotažení plochy do hraničních křivek nebo „uzavření“do uzavřené plochy realizováno analogicky jako u Coonsovy křivky. Je vhodné uvést, že tímto se vlastně vytváří hraniční plocha objemu daného objektu.

Poznámka

Béziérův plát je někdy popisován jako „tenzorový“ součin Bézierových křivek pro u a v takto:

$$P(u,v) = \sum_{k=0}^{3} P_k(v) B_k^3(u) = \sum_{k=0}^{3} \left(\sum_{i=0}^{3} P_{i,k} B_i^3(v) \right) B_k^3(u)$$

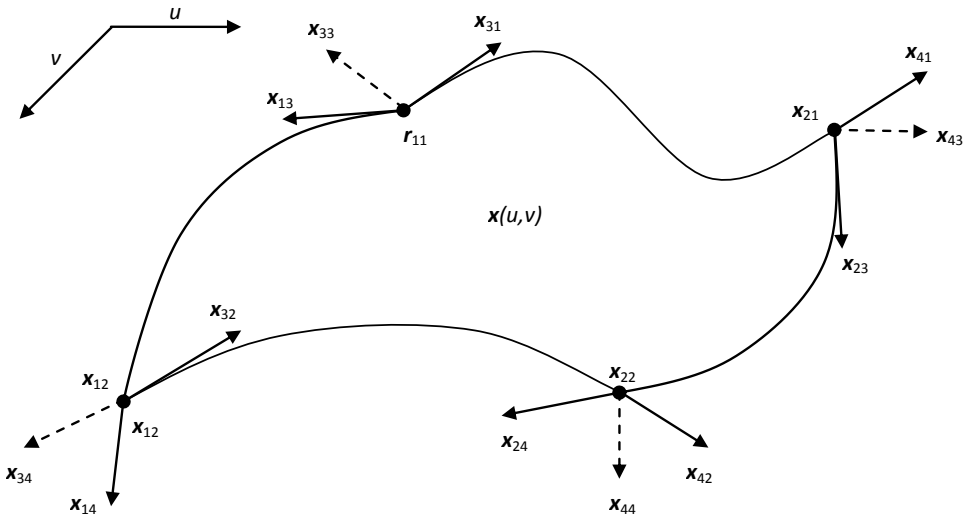
Příklad

V praxi je častý požadavek na převod Hermitovy formy plátu do formy Bézierovy. Lze ukázat, že převod z Hermitovy formy do Bézierovy formy je určen pro souřadnici x:

$$\boldsymbol{X}_B = \begin{bmatrix} x_{00} & x_{01} & x_{02} & x_{03} \\ x_{10} & x_{11} & x_{12} & x_{13} \\ x_{20} & x_{21} & x_{22} & x_{23} \\ x_{30} & x_{31} & x_{32} & x_{33} \end{bmatrix}$$

$$= \begin{bmatrix} x_{11} & x_{11} + \frac{1}{3}x_{13} & x_{12} - \frac{1}{3}x_{14} & x_{12} \\ x_{11} + \frac{1}{3}x_{31} & x_{11} + \frac{1}{3}(x_{13} + x_{31}) - \frac{1}{9}x_{33} & x_{12} + \frac{1}{3}(x_{32} - x_{14}) - \frac{1}{9}x_{34} & x_{12} + \frac{1}{3}x_{32} \\ x_{21} - \frac{1}{3}x_{41} & x_{21} + \frac{1}{3}(x_{23} - x_{41}) - \frac{1}{9}x_{43} & x_{22} - \frac{1}{3}(x_{24} + x_{42}) - \frac{1}{9}x_{44} & x_{22} - \frac{1}{3}x_{42} \\ x_{21} & x_{21} + \frac{1}{3}x_{23} & x_{22} - \frac{1}{3}x_{24} & x_{22} \end{bmatrix} \quad (13)$$

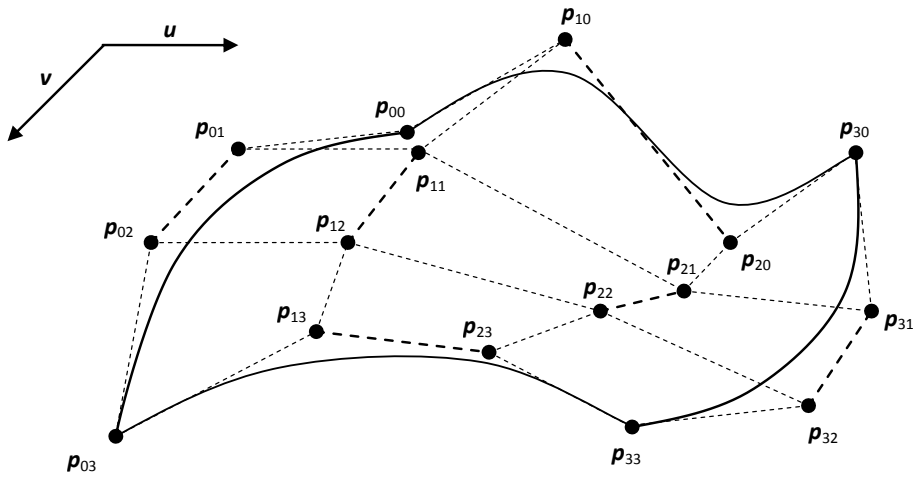
pokud použijeme indexaci, viz obr.xx. Pro ostatní souřadnice y a z postupujeme obdobně.



Obr.XX: Řídící body Hermitova plátu

Příklad

Opačný převod, tj. převod z Bézierovy formy do Hermitovy formy, při použití indexace, viz obr.TTT, je určen:



Obr.TTT: Indexace Bérierovy plochy

Pak:

$$\boldsymbol{X}_H = \begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \\ x_{31} & x_{32} & x_{33} & x_{34} \\ x_{41} & x_{42} & x_{43} & x_{44} \end{bmatrix}$$

$$= \begin{bmatrix} x_{00} & x_{03} & 3(x_{01} - x_{00}) & 3(x_{03} - x_{02}) \\ x_{30} & x_{33} & 3(x_{31} - x_{300}) & 3(x_{33} - x_{32}) \\ 3(x_{10} - x_{00}) & 3(x_{13} - x_{03}) & 9(x_{00} - x_{01} - x_{10} + x_{11}) & 9(x_{02} - x_{03} - x_{12} + x_{13}) \\ 3(x_{30} - x_{20}) & 3(x_{33} - x_{23}) & 9(x_{20} - x_{21} - x_{30} + x_{31}) & 9(x_{22} - x_{23} - x_{32} + x_{33}) \end{bmatrix} \quad (14)$$

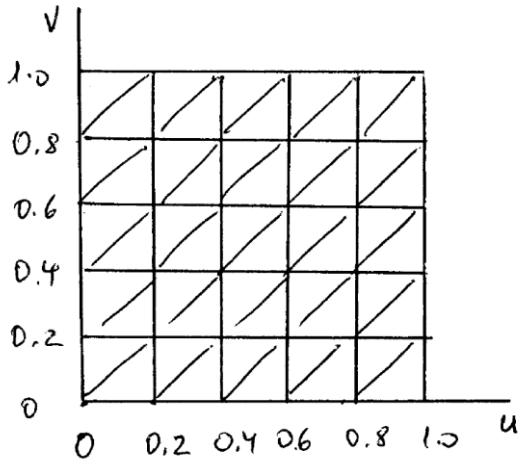
Obdobné převodní vztahy jsou možné i pro další parametrické pláty, viz:

Skala,V., Ondracka,V.: BS-Patch: Constrained Bezier Parametric Patch, Trans.on Mathematics, 2013

Nyní, když je bikubický čtyřúhelníkový plát definován a popsán, je přirozenou otázkou, jak jej vykreslit, když geometrická standardně zpracovávaná primitiva jsou vlastně jen bod, úsečka, trojúhelník a jejich odvozeniny.

10.6. Vykreslování parametrických ploch

Vykreslování kubických parametrických křivek pomocí lomené čáry bylo uvedeno v kap.10.2 (Vykreslování parametrických křivek). Pokud chceme vykreslit kružnici pomocí n-úhelníka, pak potřebujeme cca 100 vrcholů rovnoměrně rozložených na kružnici, abychom dostali vjem hladké křivky. Vykreslení bikubického plátu se většinou realizuje pomocí trojúhelníkové sítě, resp. množinou trojúhelníků, s normálovými vektory ve vrcholech. Parametrický prostor (u, v) se rozdělí na $n \times m$ obdélníků a každý obdélník se pak dále rozdělí na 2 trojúhelníky. To znamená, že pro dosažení vjemu hladkosti při vykreslování plátu budeme uvažovat $n, m \in \langle 50{,}100 \rangle$ a tedy počet generovaných bodů bude cca $\langle 2\,500, 10\,000 \rangle$ na jeden bikubický plát. Protože složité objekty se skládají z velkého počtu plátů, např. $10^2 - 10^4$ je nutné věnovat pozornost efektivitě generování souřadnic vrcholů trojúhelníků, které bikubický plát nahrazují.



Obr.SSSS: Rozdělení intervalu $u, v \in \langle 0{,}1 \rangle$ na trojúhelníkovou síť

Uvažme opět rovnici bikubického plátu pro jednoduchost pouze pro souřadnici x.

$$x(u, v) = \boldsymbol{u}^T \boldsymbol{M}_F^T \boldsymbol{X}_F \boldsymbol{M}_F \boldsymbol{v}$$

Je zřejmé, že výraz $\boldsymbol{M}_F^T \boldsymbol{X}_F \boldsymbol{M}_F$ je konstantní pro celý plát a může být předem uložen do matice:

$$\boldsymbol{Q}_x = \boldsymbol{M}_F^T \boldsymbol{X}_F \boldsymbol{M}_F.$$

Pak vlastní výpočet souřadnic vrcholů trojúhelníků nahrazující daný bikubický plát lze zapsat:

$\boldsymbol{Q}_x = \boldsymbol{M}_F^T \boldsymbol{X}_F \boldsymbol{M}_F$; $\boldsymbol{Q}_y = \boldsymbol{M}_F^T \boldsymbol{Y}_F \boldsymbol{M}_F$; $\boldsymbol{Q}_z = \boldsymbol{M}_F^T \boldsymbol{Z}_F \boldsymbol{M}_F$;

u := 0;
for i := 0 **to** n
{ $\boldsymbol{g}_x := \boldsymbol{u}^T \boldsymbol{Q}_x$; $\boldsymbol{g}_y := \boldsymbol{u}^T \boldsymbol{Q}_y$; $\boldsymbol{g}_z := \boldsymbol{u}^T \boldsymbol{Q}_z$; # Použít Hornerovo schéma #
v:=0;
for j := 0 **to** m
{ $x := \boldsymbol{g}_x^T \boldsymbol{v}$; $y := \boldsymbol{g}_y^T \boldsymbol{v}$; $z := \boldsymbol{g}_z^T \boldsymbol{v}$; # Použít Hornerovo schéma #
SAVE (x,y,z,i,j); # uložení x, y, z do vhodné datové struktury #
}
}

QQQQQ: Ideový algoritmus pro výpočet bodů bikubického plátu

Vlastní vykreslení bikubického plátu lze realizovat:

- s konstantním stínováním, pak se zadává pouze normála trojúhelníka. Ta se určí např. jako vektorový součin vektorů hran jednotlivých trojúhelníků
- pro stínování Gouraudovo nebo Phongovo je zapotřebí určit normálu v každém vrcholu trojúhelníka. To je trochu obtížnější úloha, neboť:
  - lze ji odhadnout jako průměr normál trojúhelníků sdílejících daný vrchol, nebo
  - určit přesným výpočtem, a to vektorovým součinem derivací:

$$\boldsymbol{n} = \frac{\partial u(u,v)}{\partial u} \times \frac{\partial u(u,v)}{\partial v}$$

  Uvědomme si, že výše uvedené parciální derivace jsou vlastně řídícími hodnotami Hermitovy formy. V případě Bézierovy formy lze derivace určit z rozdílu souřadnic řídících bodů, viz transformace Bézierovy a Hermitovy formy kap.10.3 (Transformace kubických parametrických křivek).

Poznámky

- Plát není obvykle vykreslován samostatně, ale spolu s dalšími pláty, které jsou na daný plát napojeny, je nutné brát v úvahu také sousední pláty při určování normálového vektoru bodů ležících na hraničních křivkách, tj. pro $u = 0$, $u = 1$ a $v = 0$, $v = 1$.
- Výpočet normály může využít analytickou formu pro plát a počít normálu přímo ze vzorce pro daný bikubický plát.

Hladké napojování jednotlivých plátů, které je nutné k reprezentaci složitých povrchů, je netriviální výpočetní operací a se nazývá plátování.

10.7. Hladké napojování plátů - dodělat

VIZ předmět Geometrické modelování

10.8. Výhody a nevýhody parametrické reprezentace – dodělat

Nyní je asi vhodné porovnat výhody a nevýhody parametrické reprezentace vůči ostatním.

Výhody	Implicitní	• snadná klasifikace bodů-pozice (bod uvnitř-vně) • snadná reprezentace průsečíků, interference, množinové operace • jednoduchá reprezentace uzavřených křivek a ploch, multi-hodnotové křivky, „nekonečná“ směrnice
	Explicitní	• snadnost trasování • jednoduchý výpočet funkční hodnot
	Parametrická	• snadnost transformací (nezávislost na souřadných osách) • snadná manipulace a formování tvarů – „free form shapes“ • jednoduchá generace složených křivek
Nevýhody	Implicitní	• obtížná manipulace a formování tvarů – „free form shapes“ • závislost na osách souřadného systému • složité trasování křivky, extrakce povrchu
	Explicitní	• závislost na osách souřadného systému • složitá reprezentace uzavřených křivek a ploch, multihodnotových křivek apod. • reprezentace „nekonečné“ směrnice pro polynomiální funkce
	Parametrická	• vysoká flexibilita podstatně komplikuje výpočet průsečíků, např. přímka -parametrická křivka, rovina-bikubická plocha atd.

11. Algoritmy řešení viditelnosti – vlozit text

11.1. BSP strom

Vlozit text

11.2. z-buffer

12. Modely osvětlení a stínování – vlozit text

12.1. Modely osvětlení

12.2. Metody stínování

12.3. Lokální metody– vlozit text

Dosud jsme se zabývali lokálními metodami stínování, které jsou v zásadě jednoduché a výpočetně nenáročné, i když Phong stínování není běžně implementováno v grafických akcelerátorech z důvodů výpočetní složitosti. V následující části se budeme zaobírat základními globálními metodami zobrazování scény. Tyto metody jsou podstatně výpočetně náročnější, a to jak z hlediska časové náročnosti, tak i z hlediska náročnosti paměťové.

13. Globální metody

Globální metody zobrazování scény se snaží o maximální respektování optických poměrů ve scéně a jsou založeny na různých modelech. Lokální metody stínování prezentované v předchozí části jsou vlastně založeny na přímém zobrazování geometrických elementů, včetně jejich překrývání, plnění barvou nebo vzorem, řešení viditelnosti atd. ¨

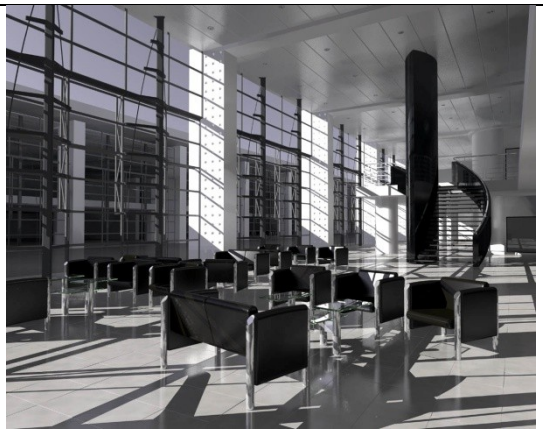
V následujícím textu si uvedeme alespoň hlavní techniky, a to:

- **metodu sledování paprsku**, která je založena na reprezentaci chování optického paprsku na optickém rozhraní, tj. paprsek se odráží od rozhraní dvou materiálů a propouští se do materiálu prostředí druhého.
  Metoda sledování paprsku dobře zpracovává scény se zrcadlovým charakterem nebo s objekty průhlednými a scény s bodovými zdroji světla. Při změně pozice pozorovatele se však celý výpočetní proces musí opakovat. Typickým představitelem je např.
  POV-Ray: Persistence of Vision Raytracer http://www.povray.org/

- **radiační metodu**, která je založena na energetické bilanci ve scéně, kdy každá elementární ploška přijímá a vydává nějakou energii.
  Radiační metoda velmi dobře zpracovává scény s difuzním odrazem a plošnými zdroji světla. Protože je založena na energetické bilanci, energetické poměry ve scéně se spočtou pouze jednou, při změně pozice pozorovatele se pouze jednotlivé plošky odpovídajícím způsobem zobrazují.
  Typickým představitelem je např.
  Radiance http://radsite.lbl.gov/radiance/download.html

Obě metody jsou „hraniční“ metody z hlediska optického modelu. Radiační metoda není primárně vhodná pro zrcadlové odrazy, metoda sledování paprsku není vhodná pro difuzní odrazy.

Výstup z POV-Ray – Bonsai Girl



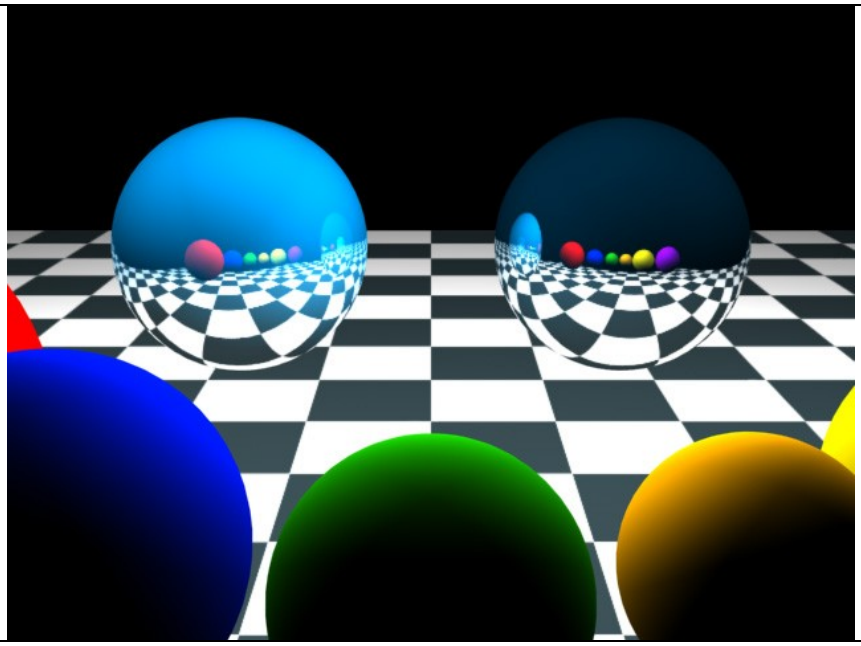
Výstup z Radiance - Lobby

Z uvedeného je zřejmé, že je možné řešit i velmi složité scény a obě metody dnes umějí řešit jak difuzní, tak i zrcadlový odraz.

Upozornění: Výpočetní náročnost obou globálních metod je nepoměrně vyšší než metod lokálních.

13.1. Metoda sledování paprsku

Jedna z globálních metod je metoda sledování paprsku (Ray tracing). Metoda je založena na geometrické představě, že z oka pozorovatele se „vypustí“ paprsek (Ray), který protne danou pozici na obrazovce, tj. daný pixel, a hledá se nejbližší průsečík paprsku se všemi objekty ve scéně. Po nalezení nejbližšího průsečíku se paprsek „rozštěpí“ na paprsek odražený od povrchu tělesa podle zákona o odrazu a na paprsek do tělesa propuštěný podle zákona o indexu lomu. Každý z těchto paprsků je dále nezávislý a celý proces se opakuje, pokud není zastaven např. hloubkou stromu apod.



Obr.xxx: http://www.neilblevins.com/cg_education/metal_and_refs/metal_and_refs.htm

Vytváří se tak vlastně stromová struktura s průsečíky. Po skončení této fáze výpočtu průsečíků se určí světelné poměry průsečíků odpovídajících listům stromu a zpětně se pak počítají světelné poměry průsečíků odpovídajících nadřízeným uzlům stromu až se určí intenzita, která se zobrazí v odpovídajícím pixelu.

Z uvedeného je zřejmé, že metoda sledování paprsku:

- umožňuje ve scéně dobře respektuje optické vlastnosti objektů, tj. průhlednost, průsvitnost, lom světla, barvu atd., viz obr.xxx
- je snadno paralelizovatelná, neboť každý paprsek „žije“ nezávislým životem
- dokáže respektovat charakteristiky světelných bodových zdrojů světla
- má velkou výpočetní složitost, a to $O(M\,N^2\,2^k)$, kde $N \times N$ je rozlišení výstupu, M je počet objektů ve scéně, k je počet úrovní výpočetního stromu. Takže pro $N = 1024 = 2^{10}$, $M = 2^{20}$, což je cca 1 mil. objektů a $k = 2^3$ se bude počítat cca

  $$2^{33} \approxeq 10^{10}$$

  průsečíků paprsku s objekty, resp. obalovými tělesy! Takže, pokud

  Je také nutné zdůraznit, že výpočetní nároky rostou s kvadrátem rozlišení generovaného obrazu.

Algoritmus s detailním výpočtem pro jeden paprsek, viz:

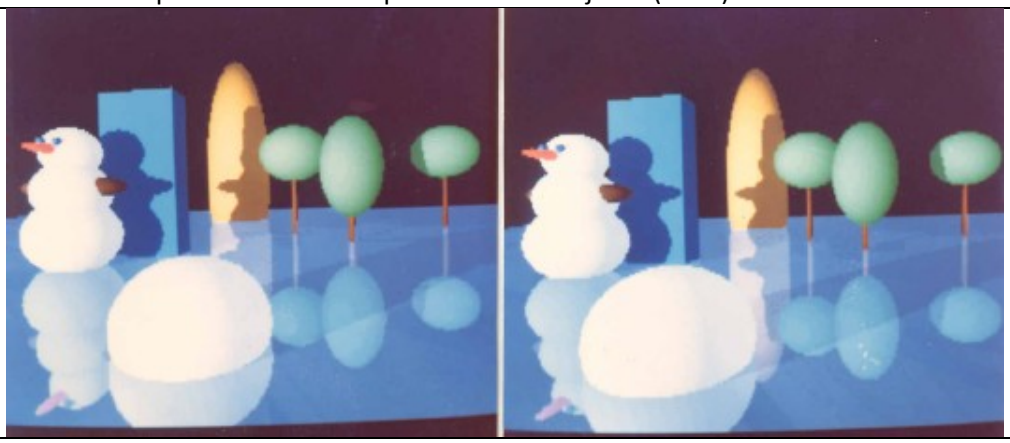
- Skala,V. Algoritmy počítačové grafiky III, Plzeň, 2011, str.94-107 (CLICK off-line)
- Skala,V.: Světlo, barvy a barevné systémy v počítačové grafice, Academia, 1993

Poznámka

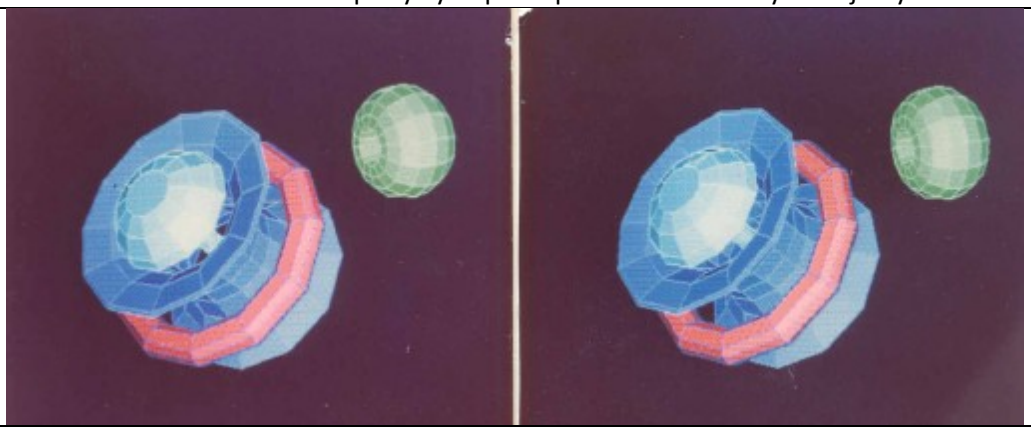
První experimenty s metodou sledování paprsku na Vysoké škole strojní a elektrotechnické v Plzni (předchůdce ZČU) byly již v roce 1976 (tehdy se jednalo jen o tzv. primární ray tracing s použitím CSG stromů a objektů definovaných implicitní funkcí). V té době nebyly k dispozici odpovídající výpočetní kapacity a ani možnosti grafického výstupu.

První reálné grafické výstupy byly realizovány v roce 1991 na počítačích s 48KB paměti, s 8 bitovým procesorem I8080 procesorem (Intel) bez hardwarové podpory operací v pohyblivé řádové čárce a výstupem na TV obrazovku 256x256 pixelů se 16 barvami.

Experimenty s metodou sledování paprsku realizované jako semestrální práce předmětu ekvivalentního ZPG s použitím stereoskopie k získání 3D vjemu (1991).



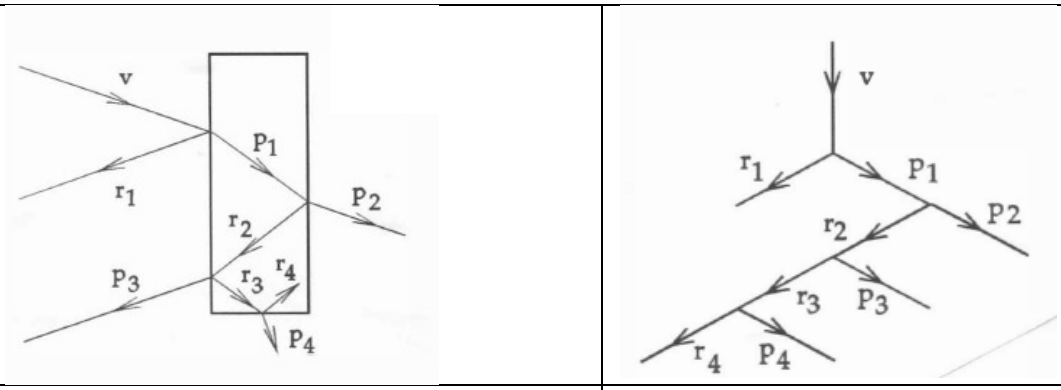
Obr.XXX: Stereoskopický výstup s implicitně definovanými objekty



Obr.QQQ: Množinové operace s objekty definovanými trojúhelníkovou sítí a se stereoskopickým výstupem

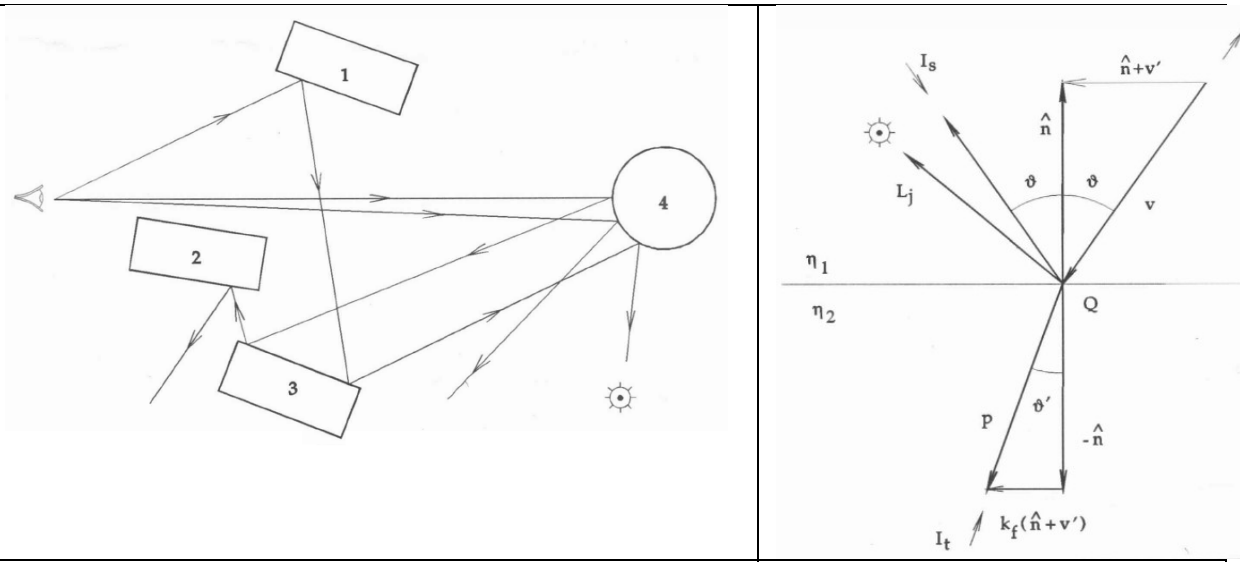
Pro realizaci stereoskopického výstupu je nutné vlastně generovat dva výstupní obrazy, tj. pro dvě různé pozice pozorovatele s odpovídající disparitou, viz kap.4.5 (Stereoskopická projekce).

Představme si pro jednoduchost extrémně jednoduchou scénu, kdy paprsek $\boldsymbol{v}$, který je dán pozicí pozorovatele a příslušného pixelu na průmětně dopadá na kvádr, který je obecně průhledný s jinými optickými vlastnostmi. Tento paprsek se na povrchu „rozštěpí" na paprsek odražený $\boldsymbol{r}_1$ a propuštěný $\boldsymbol{p}_1$, viz obr.TTT. Paprsky jsou nadále na sobě nezávislé a zase se odrážejí, resp. pronikají z daného optického prostředí do jiného. Je tedy zřejmé, že se vytváří vlastně výpočetní strom.



Obr.TTT: XXXXXXXXXXXXXXXXXXXX

Pro reálnou scénu, která obsahuje více objektů a zdroje světla je již výpočetní proces komplikovanější, viz obr.QQQQ. Je zřejmé, že pozorovatel uvidí, pokud jsou povrchy tělesa 1 a tělesa 3 zrcadlové, zdroj světla a povrch tělesa 3, které jsou pro pozorovatele zakryté, např. tělesem 2, resp. neviditelné, neboť jejich plocha je odvrácená od pozorovatele.



Obr.QQQQ: Schéma scény se světelným zdrojem

Primární paprsek je určen pozicí pozorovatele a daného pixelu na průmětně, a to:

$\boldsymbol{x}(t) = \boldsymbol{x}_A + (\boldsymbol{x}_B - \boldsymbol{x}_A)$	$t \in \langle 0, \infty)$

kde: $\boldsymbol{x}_A$ je pozice pozorovatele a $\boldsymbol{x}_B$ je pozice pixelu na průmětně v daném souřadném systému.

Nejdříve je nutné najít nejbližší průsečík paprsku s objekty ve scéně. Je zřejmé, že tento proces je výpočetně poměrně náročný, neboť se provádí pro N^2 pixelů a scéna má M objektů. Navíc po dopadu paprsku na optické rozhraní se paprsek rozštěpí na paprsky dva, což při k optických rozhraní, na něž paprsek a jeho potomci narazí, se počítá 2^k průsečíků. Je tedy zřejmé, že je nutné aplikovat některé akcelerační techniky. Mezi základní akcelerační techniky lze řadit:

- **obalová tělesa** pro rychlou detekci průsečíku paprsku s objektem. Účelem je rychlá detekce, zda může existovat průsečík paprsku s daným objektem. Obalové těleso, resp. jejich kombinace, by mělo co nejlépe vystihovat tvar daného objektu, aby pravděpodobnost toho, že byl detekován možný průsečík paprsku s objektem a existencí takového průsečíku, byla co nejvyšší. Jako obalová tělesa se používá koule (sphere), která je rotačně invariantní, nebo osově orientovaný kvádr (Axis Aligned Bounding Box – AABB), resp. jejich kombinace.
  Viz kap.6.9 (Algoritmy výpočtu průsečíků a ohraničující tělesa)

  dělení prostoru scény, tj. E^3, kdy každý element prostoru má informaci o objektech, které mají společnou část s daným prostorem. Jde vlastně o formu předzpracování, což v důsledku obvykle vede ke snížení výpočtu. Tyto objekty jsou pak uloženy v seznamu pro každý prostorový element. Trik spočívá v tom, že se určí, které prostorové elementy paprsek protíná ve smyslu rostoucí hodnoty t a na průsečík se testují pouze ty objekty, které jsou pro daný prostorový element v seznamu uvedeny. Je zřejmé, že i pro dělení prostoru můžeme použít hierarchické datové struktury.
  Je nutné si uvědomit, že pokud prostor scény rozdělíme na $K \times K \times K$ elementů, pak je nutno pro každý element otestovat, zda s daným prostorovým elementem má nějakou společnou část a pro datovou strukturu je nutno alokovat paměť $O(qK^3)$, kde q je průměrný počet v daném prostorovém elementu. Tento krok má výpočetní složitost $O(MK^3)$ a paměťovou složitost $O(qK^3)$.
  Takže pokud $K = 32 = 2^5$ a $M = 2^{20} \cong 10^6$, pak tento krok má výpočetní složitost $O(MK^3) = C\ 2^{35}$, kde konstanta $C > 0$.
  Viz kap.6.4 (Dělení prostoru a Binární masky)

- **koherence paprsků** a mnoho dalších

13.2. Základní algoritmus Ray-tracing– vlozit text

13.3. Stíny– vlozit text

13.4. Zrcadlové odrazy – vlozit text

13.5. Transparence– vlozit text

transparence

13.6. CSG stromy v RT– vlozit text

14. Radiační metoda a Monte Carlo– vlozit text

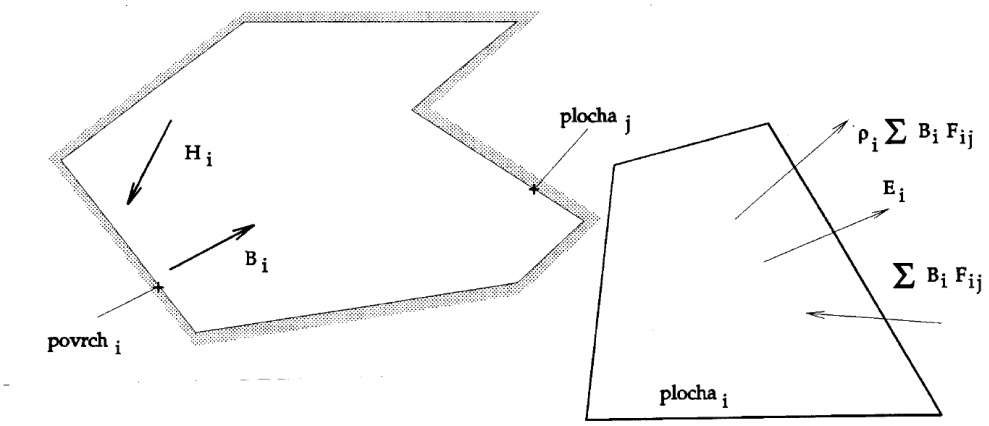
Radiační metody jsou založeny na energetické bilanci ve scéně, kdy se scéna uzavře do energetického obalu a spočítají následně energetický příjem a výdaj jednotlivých plošek reprezentující povrch objektů ve scéně. Z principu je tedy zřejmé, že radiační metody nejsou schopny jednoduše respektovat např. lom světla na rozhraní dvou opticky různých materiálů.

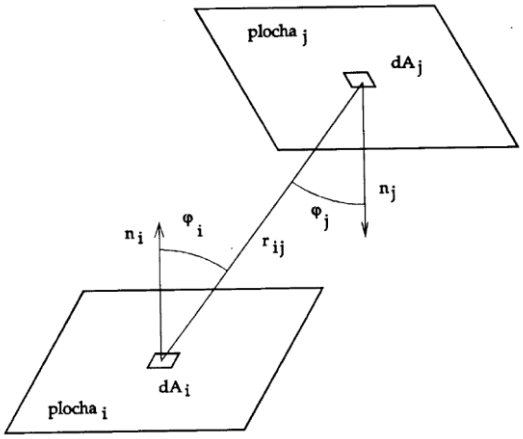
Celý proces vede na řešení soustav lineárních rovnic velkého rozsahu. Existuje celá řada různých modifikací základní metody.

Porovnání základních přístupů:
Willmott,A., Heckbert,P.S. An Empirical Comparison of Radiosity Algorithms, Tech.Rep. Carnegie Mellon Univ., 1997 http://www.cs.cmu.edu/~radiosity/emprad-tr.html (CLICK off-line)

14.1. Princip radiační metody– vlozit text

Viz Skala,V.: Algoritmy počítačové grafiky III, Plzeň, 2011, str.108-113 (CLICK off-line)





Reference

Vince,J.: Introduction to Virtual Reality, Springer, 2004

18. Rastrová grafika – základní algoritmy– vlozit text

18.1. Bresenhamův algoritmus– vlozit text

18.2. Rasterizace trojúhelníka– vlozit text

18.3. Alfa-kanál– vlozit text

18.4. Antialiasing– vlozit text

18.5. Obrazová data - snímání a ukládání– vlozit text

18.6. Hexagonální rastr– vlozit text

Reference

Middleton,L., Sivaswamy,J.: Hexagonal Image Processing –A practical Approach, Springer2005

19. Animace, principy a inverzní kinematika– vlozit text

19.1. Kinematika, Inverzní kinematika, Dynamika– vlozit text

19.2. Ryv a jeho dusledky pro animaci a taktilní I/O– vlozit text

20. Doporučená literatura– vlozit text

Blinn,J.: Jim Blinn's Corner: A Trip Down the Grahics Pipeline, 1996

Přílohy

TBD