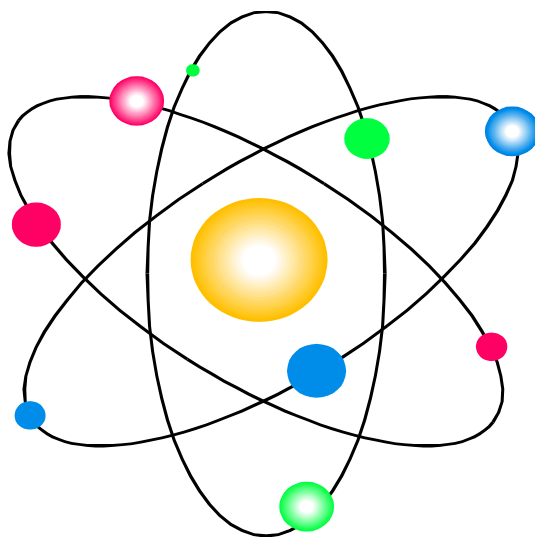


Katedra počítačov a informatiky
Fakulta elektrotechniky a informatiky
Technická univerzita Košice

Ing. Branislav Sobota, PhD.

Počítačová grafika

pre dištančné štúdium a štúdium popri zamestnaní



Košice





© 2002

Poznámky:

*

Predslov

Predkladaný učebný text je určený k predmetu Počítačová grafika I pre študentov dištančného štúdia a štúdia popri zamestnaní na KPI FEI TU Košice, vo väčšej miere ho môžu použiť aj študenti denného štúdia pre daný predmet. Vstupné predpoklady sú dané najmä zvládnutím základov algoritmiky a programovacích techník, ďalej v ovládaní aspoň jedného vyššieho programovacieho jazyka (optimálne jazyk C), všetko však systémy so schopnosťou pracovať s grafickou informáciou (napr. Borland C/C++, Visual C++) a samozrejme v ovládaní základov výpočtovej techniky. Pre porozumenie matematických základov sa predpokladá minimálne zvládnutie analytickej geometrie a vektorovej a maticovej algebry

Skriptá sú didakticky písané od základov počítačovej grafiky (ďalej PG) po niektoré vyspelé techniky a algoritmy v nej používané. Úvod poskytuje postačujúci historický vývoj výpočtovej techniky a na jej základe aj počítačovej grafiky, má len informačný charakter a neslúži ako základ pre ďalšie možné otázky. Každá kapitola je členená v princípe na dve časti: základná problematika a rozšírená problematika (označená symbolom ) , ktorá detailnejšie alebo navyiac popisuje niektoré špecializované prípady alebo ďalšie typy z preberanej problematiky. Vzhľadom na rozsah skript niektoré problémové okruhy však predpokladajú dovysvetľovanie formou prednášky resp. precvičenie na cvičení. V závere každej kapitoly je vždy zhrnutie a kontrolné otázky, ktoré budú tvoriť odrazový mostík pre skúšku.. Skriptá však nie sú učebnicou programovania programov na báze počítačovej grafiky, poskytuje však výborný teoretický základ pre túto činnosť. Pri samostatnom štúdiu sa odporúča najprv zvládnuť aspoň teoreticky prvú kapitolu t.j. grafické periférie počítačov. Nasledujúce časti sú delené tak, ako bolo uvedené skôr t.j. každá hlavná kapitola obsahuje kapitoly resp. časti označené symbolom . Tieto predstavujú rozšírené pojmy. Symbol , ak nie je uvedené inak, platí aj pre všetky podkapitoly takto označenej kapitoly. Pri prvotnom štúdiu nie je nutné sa takto označenou problematikou zaoberať, aj keď tematicky patrí tam, kde je uvedená. V prvom rade je vhodné najprv takto naštudovať prvý okruh. Po naštudovaní jeho obsahu a osvojení si týchto základov, sa doporučuje opätovne následné celkové štúdium vrátane kapitol označených symbolom . Po zvládnutí celého prvého okruhu je možné pristúpiť k druhému, náročnejšiemu okruhu. Forma spracovania je navyše prispôbena možnosťou písania vlastných poznámok ku preberanej problematike na každej strane.

Témy jednotlivých kapitol sú v požadovanom didaktickom slede a pre označovanie kapitol platí to, čo pre prvý okruh.. V úvode je uvedený popis grafických periférnych prostriedkov počítačov, pomocou ktorých sa môžu realizovať postupy a princípy popísané v nasledujúcich kapitolách. V ďalšom je publikácia rozdelená do dvoch veľkých okruhov. Prvý okruh sa venuje základným postupom a princípom využívaným v PG. Druhý okruh obsahuje popis pokročilých postupov a algoritmov používaných v súčasnej PG. V rámci prvého okruhu budú popísané základné grafické systémy a prvky. V nadväznosti na to sú následne popísané súradnicové systémy používané v PG, geometrické a premietacie transformácie. V závere tohto okruhu sú uvedené ešte krivky a plochy, ktoré sa používajú v rámci PG. V druhom okruhu sú uvedené niektoré pokročilejšie technológie používané v rámci PG. Tu sa jedná najmä o vyplňovanie oblastí, riešenie problematiky viditeľnosti, realistické zobrazovanie objektov prostriedkami PG a ako neodmysliteľná doplnková časť



v tomto okruhu je popísaná problematika farieb a ich používania v PG. V závere sú ešte v krátkosti uvedené niektoré oblasti modernej PG ako fraktály, multimédia a najmä virtuálna realita.











Ako už bolo uvedené súčasťou skript sú aj kontrolné otázky resp. cvičenia uvedené za každou kapitolou. Pri samostatnom štúdiu nepokračujte ďalej, pokiaľ nie ste schopný samostatne na ne zodpovedať resp. ich vypracovať. Skúška z predmetu pozostáva z písomnej a ústnej časti. Po prebratí prvého aj druhého okruhu budú bude krátke písomné testy, ktorých výsledky budú zarátané ako prvá časť písomnej časti skúšky. Druhou časťou písomnej skúšky je vytvorenie programového vybavenia vo forme zadania vrátane dokumentácie napísanej podľa noriem KPI. Na ústnu skúšku postupuje študent až po absolvovaní písomnej časti.











Záverom by som chcel poďakovať kolegom Doc. Ing. Milanovi Šujanskému CSc., Ing. Marekovi Strakovi a Ing. Martinovi Grofčíkovi za pomoc, rady a názory pri vytvorení niektorých kapitol tohto učebného textu.

autor

Obsah

1 ÚVOD.....	9
2 GRAFICKÉ PROSTRIEDKY POČÍTAČOV	14
2.1 ROZDELENIE GRAFICKÝCH ZARIADENÍ	14
2.2 VSTUPNÉ GRAFICKÉ ZARIADENIA	16
2.2.1 Mys	16
2.2.2 Trackball.....	17
2.2.3 Trackpad.....	17
2.2.4 Joystick.....	17
2.2.5 Tablet.....	18
2.2.6 Scanner.....	19
2.2.7 Optické pero.....	20
2.3 VÝSTUPNÉ GRAFICKÉ ZARIADENIA	20
2.3.1 Tlačiareň.....	20
2.3.2 Súradnicový zapisovač, ploter	22
2.3.3 Zobrazovače, monitory	23
2.4 ZOBRAZOVACIE ADAPTÉRY	27
2.4.1 Grafické procesory 	28
2.4.2 Základné zobrazovacie adaptéry PC	29
2.4.3 EGA/VGA zobrazovacie adaptéry.....	29
2.4.4 Krátko záverom k zobrazovacím adaptérom.....	33
2.5 OSTATNÉ GRAFICKÉ ZARIADENIA	34
3 GRAFICKÉ SYSTÉMY	36
3.1 GRAFICKÉ ŠTANDARDY	36
3.1.1 GKS (Graphics Kernel System).....	37
3.1.2 GKS-3D (3D Extension of GKS)	37
3.1.3 PHIGS (Programmer's Hierarchical Interaction Graphics System).....	38
3.1.4 OpenGL (Open Graphics Library).....	39
3.2 ZÁKLADNÉ GRAFICKÉ PRVKY	41
3.2.1 Bod.....	45
3.2.2 Sled bodov (polymarker)	45
3.2.3 Úsečka.....	46
3.2.4 Sled úsečiek (polyline).....	48
3.2.5 Kružnica	48
3.2.6 Elipsa.....	49
3.2.7 Kruhový (eliptický) výsek 	50
3.2.8 Antialiasing	51
4 TRANSFORMÁCIE V POČÍTAČOVEJ GRAFIKE	54
4.1 SÚRADNICOVÉ SÚSTAVY	54
4.2 ZOBRAZOVACIE TRANSFORMÁCIE.....	56
4.3 GEOMETRICKÉ TRANSFORMÁCIE.....	57
4.3.1 Posunutie.....	59
4.3.2 Zrkadlenie.....	59
4.3.3 Zmena mierky.....	61
4.3.4 Skosenie	65
4.3.5 Otočenie.....	67
4.4 2D PREMIETACIE TRANSFORMÁCIE	72
4.4.1 Orezanie	74
4.4.2 Transfokácia.....	76

4.4.3	Panorámovanie	77
4.5	3D PREMIETACIE TRANSFORMÁCIE	78
4.5.1	Kolmé premietanie	78
4.5.2	Axonometria	79
4.5.3	Perspektíva	80
4.6	NELINEÁRNE PREMIETACIE TRANSFORMÁCIE 	81
4.6.1	Rybie oko	81
5	KRIVKY POUŽÍVANÉ V POČÍTAČOVEJ GRAFIKE	84
5.1	KRIVKY DANÉ ANALYTICKÝM POPISOM	84
5.2	INTERPOLAČNÉ KRIVKY	85
5.2.1	Lagrangeova interpolácia 	85
5.2.2	Fergusonova krivka	85
5.2.3	Akimovská interpolácia 	86
5.2.4	Spline krivka 	87
5.3	INTERAKTÍVNE VYTVARANÉ KRIVKY	87
5.3.1	Beziérové krivky	88
5.3.2	B-spline krivky 	89
6	PLOCHY POUŽÍVANÉ V POČÍTAČOVEJ GRAFIKE	92
6.1	PLOCHY DANÉ ANALYTICKÝM POPISOM	92
6.2	COONSOVE PLOCHY	92
6.2.1	Bilineárna Coonsova plocha	93
6.2.2	Bikubická Coonsova plocha 	93
6.2.3	Všeobecná Coonsova plocha 	94
6.3	BEZIÉROVÉ PLOCHY	95
6.3.1	Beziérova bikubická plocha	95
7	VYPLŇOVANIE OBLASTÍ	98
7.1	METÓDA RIADKOVÉHO ROZKLADU	99
7.1.1	Algoritmus riadkového rozkladu	100
7.1.2	Nájdenie priesečníkov hrán s rozkladovým riadkom	100
7.2	INVERZNÉ VYPLŇOVANIE	101
7.2.1	Algoritmus inverzného vyplňania	101
7.3	ŠABLÓNOVÉ VYPLŇOVANIE 	102
7.3.1	Algoritmus vyplňania šablóny	102
7.4	POROVNANIE EFEKTÍVNOSTI METÓD	103
7.5	VYPLŇANIE INÝCH OBLASTÍ 	103
7.6	VÝPLŇ SPEKTROM (PRECHOD Z JEDNEJ FARBY DO DRUHEJ) 	103
7.7	SEMIENKOVÉ VYPLŇOVANIE	105
7.7.1	Nerekurzívne semienkové vyplňovanie	106
8	RIEŠENIE VIDITEĽNOSTI	109
8.1	RIEŠENIE VIDITEĽNOSTI GRAFOV FUNKCIÍ	111
8.1.1	Riešenie v priestore	111
8.1.2	Riešenie v priemeti (Algoritmus plávajúceho horizontu)	111
8.2	MALIAROV ALGORITMUS RIEŠENIA VIDITEĽNOSTI	112
8.3	WARNOCKOV ALGORITMUS	113
8.4	FREEMAN-LOUTRELOV ALGORITMUS	114
8.5	ALGORITMUS PAMÄTE HLĚBKY (Z-BUFFER)	116

8.6	ALGORITMUS RIADKOVÉHO ROZKLADU 	118
8.7	METÓDA BSP STROMOV 	120
8.7.1	Tvorba a prechod BSP stromu.....	120
8.7.2	Vlastnosti BSP.....	121
8.7.3	Použitie BSP.....	123
8.7.4	Výhody a nevýhody BSP.....	124
8.8	S-BUFFER 	124
8.9	URÝCHLOVACIE TECHNOLOGIE RIŠENIA VIDITEĽNOSTI 	125
8.9.1	3D orezávanie na zorný ihlan.....	126
8.9.2	Ohraničujúce útvary.....	127
8.9.3	Sektorovanie.....	128
8.9.4	Potenciál viditeľnosti.....	129
9	REALISTICKÉ ZOBRAZOVANIE	132
9.1	TIEŇOVANIE.....	132
9.1.1	Konštantné tieňovanie.....	132
9.1.2	Gouraudovo tieňovanie.....	132
9.1.3	Phongovo tieňovanie.....	133
9.2	FOTOREALISTICKÉ ZOBRAZOVANIE.....	134
9.2.1	Porovnanie metód fotorealistického zobrazovania.....	134
9.2.2	Metóda sledovania líča (raytracing).....	135
10	FARBY V POČÍTAČOVEJ GRAFIKE	138
10.1	FAREBNÉ MODELY.....	138
10.1.1	Model RGB.....	139
10.1.2	Model CMY.....	140
10.1.3	Model HSB.....	141
10.1.4	Model HLS.....	141
10.1.5	Model UWB 	142
10.2	PREVODY FAREBNÝCH MODELOV 	143
10.2.1	Prevody RGB a CMY.....	143
10.2.2	Prevod RGB do HSB.....	144
10.2.3	Prevod HSB do RGB.....	144
10.2.4	Prevod RGB do HLS.....	144
10.2.5	Prevod HLS do RGB.....	145
10.2.6	Prevod na úrovne šedej.....	146
10.3	GAMA-KOREKCIA.....	146
10.4	ALFA-MIEŠANIE.....	146
10.5	ROZPTYĽOVANIE.....	147
10.5.1	Náhodné rozptyľovanie.....	147
10.5.2	Maticové rozptyľovanie.....	148
10.5.3	Distribúcia chyby 	149
10.6	POLTÓNOVANIE (HALFTONING) 	150
10.7	MEDIÁN FILTER 	151
10.8	SPRIEMERNENIE FARIEB 	152
11	ZÁVER	154
12	PRÍLOHY	158
12.1	GDI-GRAPHICS DEVICE INTERFACE MS WINDOWS.....	158
12.1.1	Kontext (Trieda=Ikontext).....	158
12.1.2	Palety (Trieda=Ipaleta).....	159
12.1.3	Štetce (Trieda=Istetec).....	161

Poznámky:

12.1.4	<i>Perá (Trieda=Ipero)</i>	164
12.1.5	<i>Atribúty</i>	165
12.1.6	<i>Čiary</i>	167
12.1.7	<i>Elipsy</i>	169
12.1.8	<i>Bitmapy</i>	171
12.1.9	<i>Texty (Trieda=Itexty)</i>	175
12.1.10	<i>Fonty (Trieda=Ifonty)</i>	176
12.2	VYTVORENIE APLIKÁCIE POMOCOU WINAPI FUNKCIÍ	178
12.3	ARCHITEKTÚRA MFC APLIKÁCIE	180
12.3.1	<i>CWnd a HWND</i>	181
12.3.2	<i>Aplikácia - CWinApp</i>	181
12.3.3	<i>Dokument - CDocument</i>	182
12.3.4	<i>CCmdTarget</i>	183
12.3.5	<i>CView - pohľad</i>	183
12.3.6	<i>Hlavný rámec MDI aplikácie</i>	184
12.3.7	<i>Hlavný rámec SDI aplikácie</i>	185
12.4	VYTVORENIE PROJEKTU POMOCOU MFC AppWIZARDU	188
12.5	PROGRAMOVÉ ÚLOHY	189
13	ODPORÚČANÉ ROZŠIRUJÚCE ZDROJE	191
14	REGISTER	192

1 Úvod



Počítačová grafika je jednou z generačných premien vo výpočtovej technike. Jej rozvoj je natoľko búrlivý, že od drahých zariadení určených pre vojenské a špičkové priemyselné aplikácie sa behom necelých tridsiatich rokov rozšírila až do škôl a domácností ako výhodné informačné médium a prostriedok poučenia i zábavy; súčasne zaplavila konštrukčné kancelárie, reklamné a filmové štúdiá, nemocnice a svoje uplatnenie si našla aj v móde, automobilizme, letectve, umení.

Počítačová grafika napomohla masovému rozšíreniu počítačov a odstránila zábrany, ktoré bežný človek pri styku s nimi cíti, že je zaplavený stĺpcami čísel a textu a vťahovaný do sveta, v ktorom sa nevie orientovať.

Ako aj vyplýva z predchádzajúcich riadkov, vývoj ľudstva je zviazaný s procesom získavania, spracovania, prenosu a záznamu informácie. Ak k tomu prirátame, že najviac informácií človek získava zrakom, dnešné počítače tomu už veľmi pomáhajú. Uvedme si krátky historický prehľad, ktorého dôsledkom bol vývoj výpočtovej techniky a na základe toho vývoj počítačovej grafiky.

V 4. tisícročí pred n.l. vzniká prvé ucelené obrázkové písmo v Mezopotámii. O tisíc rokov neskôr (3. tisícročie pred n.l.), geniálni obchodníci fēničania zostrojili *abakus* (mechanické počítaadlo). O niekoľko desaťročí nato, sa objavujú prvé súvislé znakové sady, do tohto obdobia datujeme vznik abecedy. V slávnom Babylone v rokoch 1800-1600 pred n.l. uzreli svetlo sveta prvé písomne zachované matematické algoritmy (60-ková sústava). 570-500 rokov pred n.l. Pytagoras skúma vzťahy geometrických telies. O dvesto rokov neskôr (380-320 rokov pred n.l.) sa objavujú prvé filozofické úvahy Aristotela o estetike súmernosti a proporcionalite. Približne v roku 300 pred n.l. sú Euklidom vytvorené základy všeobecnej matematiky. V 8.-9. storočí píše Muhamed ibn Musa ibn Abdallah Al Chorezmi Al-Majdusi prvú učebnicu aritmetiky, v ktorej sa používa už desiatková sústava. Johannes Gutenberg v rokoch 1444-1448 objavil a zaviedol základnú technológiu prenosu informácií ľudstva - knihtlač. Približne v tom istom období (1452-1515) sú génium Leonardom da Vincim skúmané vzťahy a súvislosti vedy a umenia. Zhruba o 150 rokov potom, v roku 1580, je založený odbor analytickej geometrie Françoisom Viteom a následne René Descartes v období 1600-1650 vymedzuje algebru, ktorá je základným nástrojom nie len počítačovej grafiky. A keď už sme spomenuli *počítačovú* grafiku, je nutné do tohto historického prehľadu vsunúť aj informácie o vývoji výpočtovej techniky. Teda po abakuse pre ľudí v roku 1673 Gottfried Wilhelm Leibniz stavia zdokonalený kalkulačný stroj. Prelom nastáva, keď potom v roku 1725 B. Bouchon vynášiel diemu pásku, ktorou riadi tkací stroj a 1805-1808 Joseph Marie Jacquard zavádza tkáčsky stav riadený diemnymi štítkami. XIX. storočie je celkovo poznačené rôznymi technickými objavmi. V roku 1833 Charles Babbage vypracúva projekt univerzálneho počítacieho stroja, 1844 Samuel Finlay Breese Morse zostrojuje telegraf, 1847 Sir Henry Cole publikuje úvahy o strojoch a umelcoch, 1854 George Boole utvára dvojstavovú logiku (tak potrebnú pre dnešné počítače), 1855 James Clerk Maxwell zostrojuje integrátor, 1876 Graham Bell vynášiel telefón, 1888 Elmer Heller konštruje diemoštítkový počítací stroj (dokedy sa používali dierne štítky ako pamäťové médium?), 1890 Elmer Hollerith stavia diemoštítkový kalkulátor a tabelátor pri prvom sčítaní ľudu v USA. Prichádza slávne XX. storočie, ktoré je predovšetkým vo svojom počiatku poznamenané 1. svetovou vojnou. Až po

nej resp. tesne pred druhou svetovou vojnou sa začínajú objavovať prvé technické grafiky. V roku 1933 George David Birkhoff publikuje teóriu "numerickej estetickej miery". V rokoch 1935-1941 začína priekopnícka práca Konráda Zusa na návrhu elektrického počítača. Táto je korunovaná úspechom a Zuse zostrojuje programovateľný počítač Z3. Ním inšpirovaný Alan M. Turing v roku 1943 zostrojuje elektronický počítač COLLOSSUS (žiaľ svoje prsty v tom mala najmä armáda). V tom istom roku Norbert Wiener zavádza pojem "kybernetika". V rokoch 1944 a 1945 W. B. Hales vytvára prvé počítačové analógové kresby a John von Neumann utvára koncepciu matematických počítačov (veľmi dôležitý medzník, ved' o valnej väčšine dnešných počítačov hovoríme ako o počítačoch VonNeumannovského typu). Následný vývoj ide už pomerne rýchlo a preto ho uvedieme viacmenej heslovito. V rokoch 1946 až 1947 John William Mauchly a John Presper Eckert stavajú univerzálny elektronický (na báze elektróniek) počítač ENIAC. V roku 1951 Ivan Moscovich stavia kresliaci stroj. O rok neskôr Ben F. Laposky vystavuje analógovú počítačovú grafiku v Sanford Museum Cherokee v USA. V roku 1958 A. P. Rich publikuje článok o počítačovom programe pre textilný dizajn (prvá lastovička CAD?). Manchesterská Univerzita o rok pozdejšie zavádza superpočítač ATLAS aj pre pokusy s kreslením na počítači a na parížskom bienále je vystavený Tinguelyho stroj "Metamatic" na maľovanie. V týchto rokoch sa objavujú prvé vyššie programovacie jazyky ako FORTRAN (1957), ALGOL (1958) a COBOL (1959). V roku 1960 pracovníci firmy Boeing zavádzajú pojem "počítačová grafika". V roku 1961 Edward E. Zajac vytvára prvý film animovaný počítačom s názvom "Two-Gyro Gravity - Gradient Attitude Control System", ktorý trvá 4 minúty. Rok 1962 priniesol jednu zaujímavosť. V rámci svojej dizertačnej práce na univerzite v Massachussets vyvinul Ivan Sutherland (ďalšie významné meno) interaktívny (neobvyklé pre danú dobu) grafický program Sketchpad ovládaný svetelným perom s výstupom na grafickú obrazovku. V roku 1963 je vyrobený prvý "ploter" - kresliaci stroj riadený číslicovým počítačom. Tento rok je rokom začiatku prác na počítačovej grafike : Georg Nees, Frieder Nake, Michael A. Noll. V Massachussets Institute of Technology sa vyvíja systém pre dizajn. Začína sa tiež spolupráca programátora Kennetha Knowltona s filmárom Stanom Vanderbergom na tvorbe počítačového animovaného filmu. Následne potom v roku 1964 vzniká film "Man and His World" (prekl. Človek a jeho svet) animovaný počítačom. Tento rok predstavuje aj ďalší významný medzník. Douglas Engelbart zo Stanfordského výskumného inštitútu v Kalifornii vymyslel "x-y coordinate indicator for graphics systems", čo nie je nič iné ako dobre známa a dnes nepostrádateľná myš. V roku 1965 Georg Nees, Michael A. Noll a Bela Julesz usporiadávajú prvú výstavu počítačovej digitálnej grafiky v USA (Harward Gallery, New York, Nevada). V bývalom Československu výtvarník Zdeněk Sýkora spolu s matematikom Jaroslavom Blažkom uskutočňujú prvé experimenty s počítačovou grafikou. Do roku 1967 datujeme vznik videoartu a v Japonsku sa zakladá skupina "Computer Technique Group" (CTG). Slávny rok 1968 v Československu je rokom kedy sa usporiadávajú prvé výstavy počítačového umenia s medzinárodnou účasťou u nás (Brno, Jihlava, Olomouc). Našu tvorbu zastupujú Miroslav Klivar a Lubomír Sochor. V roku 1969 je zostrojený videosyntetizátor. V Londýne sa zakladá spoločnosť "Computer Art Society". Georg Nees vydáva knihu "Generative Computer Graphic". Objavuje sa prvý stolný počítač FLEX, ktorý používa grafiku, okienka a - myš. Zdeněk Sýkora realizuje výtvarné dielo v architektúre utvorené počítačom - mozaiku na stene vetracieho tunela na Letnej v Prahe. V rámci Bienále v Benátkach v roku 1970 je expozícia experimentálnej počítačovej grafiky "Umenie a technika". V

Uxtridge na Brunel University sa koná medzinárodné sympóziu o počítačovej grafike. V tomto roku môžeme aj hovoriť o nástupe domácich počítačov (8-bitové home-computer, prvé mikroprocesory sú na svete). Na scéne sa objavuje štrukturovaný programovací jazyk PASCAL. Rok 1972 je rokom, kedy bol pre počítačovú definovaný nový pojem - CAD (Computer Aided Design), obsah ktorého sa stal výraznou časťou počítačovej grafiky a prispel k nemalému rozvoju interaktívnych grafických editorov. Vo vývojových dielňach firmy XEROX vzniká SMALLTALK pre stolné počítače s grafikou a myšou. Pre Slovensko je výrazným medzníkom nasledujúci rok (1973), kedy Jozef Jankovič a Imrich Bertók vytvorili prvé slovenské počítačové grafiky. Dobrý trend u firmy XEROX je definovaný počítačom ALTO, kde sa ako vstupné zariadenie používa myš a z hľadiska postupu sa využíva dnes tak samozrejmé "označ a urob". V roku 1976 sa v krajine vychádzajúceho slnka, v Japonsku koná výstava "International Computer Art Exhibition". O rok neskôr je vyvinutý dokonalejší počítač firmy XEROX, počítač STAR. V roku 1978 je v Smoleniciach konferencia "Počítačová grafika" spojená s výstavou československej počítačovej grafiky. Toto obdobie je takisto dôležité najmä z dôvodu prvých kontaktov pracovníkov firmy APPLE s jazykom SMALLTALK a počítačom STAR. Vo svete sa objavujú prvé domáce počítače s lepšou grafikou a programovacím jazykom BASIC a firma INTEL uvádza prvý 16-bitový mikroprocesor i8086. Na jeho báze bol v roku 1981 vyvinutý a predávaný počítač, ktorý spôsobil to, že dnes je počítačová grafika naozaj pre každého. Tento počítač sa nazýval IBM PC. V tom čase však ešte pre každého nebol a v domácnostiach sa presadzujú lacné počítače lorda Sinclaira ZX-81 a o rok neskôr ZX-Spectrum, ktorý znamenal naozaj prevrat (aj cenou) a bol pri raste mnohých súčasných počítačových grafikov. Na scéne sa objavuje dominantný jazyk súčasnosti - jazyk C. V tomto roku (1982) sa ale aj na poli výkonnejších počítačov objavujú už priamo pracovné stanice najmä pre CAD (napr. 32-bitový HP-9000). Firma Microsoft v tomto roku začína práce na CGI - Computer Graphics Interface, základ dnešných WINDOWS. V bývalom Československu sa v tomto roku koná v Prahe prvá celoštátna výstava počítačového umenia a pre Slovensko je tento rok opäť medzníkom: Daniel Fischer a Igor Kolčanský vytvárajú prvý slovenský film animovaný počítačom. Nimi inšpirovaní, o rok neskôr, Karol Doboš v spolupráci s Martinom Šperkom a Petrom Briatkom vytvorili počítačom animovanú zvučku televízneho seriálu "Odvážni muži na štartovacej dráhe". Vo svete sa objavuje počítačom generovaný celovečerný film "TRON" a medzník pre zavedenie počítačových efektov do filmárskej praxe vkladá George Lucas svojimi "Hviezdnyimi vojnami". Na scéne sa objavuje prvý slovenský domáci počítač PMD-85, ktorý "vychoval" mnoho ľudí u nás k počítačovej gramotnosti. Všeobecne je možné povedať, že počnúc 80-tymi rokmi ide vývoj v tejto oblasti mimoriadne rýchlo a aj heslovité zmienky by zabrali pomerne mnoho priestoru. Z tohto dôvodu si tu už len spomenieme niektoré firmy a pojmy (bez udania roku), ktoré prispeli k rozvoju počítačovej grafiky vôbec a spôsobili, že počítačová grafika je dnes na takom stupni. V prvom rade je to firma Silicon Graphics, ako "vlajková loď" a firma IBM, bez ktorej by nebolo PC. Následne firma Autodesk, ktorá dokázala svojím AutoCADom, že CAD sa stal nástrojom naozaj pre všetkých. Svoje nezastupiteľné miesto najmä v CAD systémoch si taktiež vydobila firma Intergraph. Ďalej firma Commodore, ktorá svojimi "priateľkami" (AMIGA) dlho viedla v používaní pomerne kvalitnej grafiky v domácnostiach. Nemožno nespomenúť priekopníka a dlhého vodcu grafických systémov, firmu APPLE a jej MACINTOSH s grafickým rozhraním, ktorý dlhú dobu dominoval nielen v štúdiách DTP. Ďalším adeptom je firma ATARI, ktorá priniesla počítačové hry (a tým vlastne

počítačovú grafiku) každému. Firma SUN Microsystems, ktorá ponúkla kvalitné pracovné stanice za primeranú cenu, čo prinieslo veľmi kvalitnú počítačovú grafiku opäť širšiemu okruhu používateľov. Ďalším "klasikom" je firma COREL, ktorá dokázala svojím CORELDRAW!, že kresliť pomocou počítača mohol naozaj jednoducho každý. Podobne aj firmy Adobe (Illustrator, Photoshop) a Aldus (Freehand) priniesli pred používateľov ďalšie netušené možnosti grafických editorov. Firme KODAK je potrebné poďakovať za grafický formát Kodak-Photo CD, ktorý umožňuje uchovávať a prenášať veľmi kvalitné digitálne obrazy resp. fotografie. Jej konkurenta, firmu XEROX tiež nie je možné obísť, veď okrem iného práve táto firma dala svetu základy CGI t.j. grafického používateľského rozhrania. Svoje nezastupiteľné miesto najmä v oblasti animácií a nasadenia špecializovaných grafických editorov do výroby filmov a videoprodukcie patrí firmám AVID, PIXAR a SOFTIMAGE. Takisto je potrebné tu spomenúť firmy Hewlett-Packard, Canon a Epson za prínos do rozvoja tryskových a laserových tlačiarň ako čiernobielych tak aj farebných, ktoré umožnili veľmi kvalitný výstup počítačových obrazov v hmatateľnej podobe. Nakoniec uvedme firmu Microsoft, ktorej MS-DOS "vdýchol dušu" PéCéčku a spôsobil ich masové rozšírenie, ale najmä MS-Windows resp. Windows95, ktoré spôsobili, že grafika sa stala naozaj "domácou paňou" na tejto platforme.

Áno, taký je vývoj fenoménu počítačová grafika, fenoménu ktorý v súčasnosti viac a viac naberá na význame aj v očiach laickej verejnosti:

- ✓ Forma komunikácie s počítačom a používateľské rozhranie? *Počítačová grafika.*
- ✓ Obaly v obchodoch ? *Počítačová grafika.*
- ✓ Design automobilov či výrobkov spotrebnej elektroniky ? *Počítačová grafika.*
- ✓ Projektovanie budov a interiérov ? *Počítačová grafika.*
- ✓ Noviny, časopisy, katalógy ? *Počítačová grafika.*
- ✓ Predpoveď počasia ? *Počítačová grafika.*
- ✓ Počítačové hry a zábava ? *Počítačová grafika.*
- ✓ Spracovanie hospodárskych či štatistických výsledkov ? *Počítačová grafika.*
- ✓ Film, video, reklama ? *Počítačová grafika.*

...

A mnoho, mnoho ďalších príkladov. Áno, zásah počítačovej grafiky do nášho života je naozaj významný a dnes je už veľmi ťažké nájsť oblasť, v ktorej by využitie počítačovej grafiky nemalo zmysel, ak už nie kvôli samotnému spracovaniu napr. obrázkov, tak aspoň kvôli čoraz viac sa vyvíjajúcim grafickým používateľským prostrediam (viď. napr. Windows 95/98).

V základe je možné pozorovať približne tieto trendy použitia počítačovej grafiky:

- ☐ počítačové hry
- ☐ manažérska grafika
- ☐ počítačom podporované kreslenie a konštruovanie
- ☐ vizuálna simulácia
- ☐ spracovanie obrazu
- ☐ fotorealistické zobrazovanie, multimédiá a virtuálna realita

Počítačové hry sú samostatnou kapitolou. Je možné povedať, že vlastne sprostredkovávajú prvý kontakt neznalejšieho človeka s počítačom. Počítačové hry majú svoj základ už na prvých sálových počítačoch. Ich rozmach ale nastal až po nástupe malých domácich (8-bitových) počítačov. Základné počítačové hry dosiahli svoj kulmináčny bod práve však na osobných počítačoch. Stále viac a

viac sa presadzuje čo dokonalejšia grafika a je jasný trend prechodu od klasických dvojrozmerných hier ku trojrozmerným priestorovým hrám, najmä v súvislosti s výrazným zvýšením výkonov a znížením ceny grafických akcelerátorov pre PC.

Manažérska grafika slúži na grafické znázornenie výsledkov napr. rôznych kalkulácií. Používateľ má obvykle možnosť výberu z niekoľkých typov diagramov napr. kruhové, stĺpcové a pod., alebo možnosť tzv. ikonickej grafiky (symbolické zobrazenia napr. postava, strom a.i. Potom napr. jedna postava môže znázorňovať určitý počet ľudí/km²). Príslušné merítka a transformácie väčšinou vypočítava systém sám.

Počítačom podporované kreslenie a konštruovanie (CAD - Computer Aided Design) našlo v počítačoch veľmi dobrú "pôdu". Od jednoduchých dvojrozmerných grafických editorov cez trojrozmerné až ku veľkým návrhovým komplexom. Najmä z hľadiska lepšej konštruktárskej predstavivosti alebo reálnejšiemu pohľadu (napr. v architektonických CAD-och) opäť prichádza k slovu riešenie viditeľnosti. V súčasnosti je asi už pokrytá každá oblasť výroby príslušným CAD-ovským programom. Svoje prvotné uplatnenie však mali tieto programy najmä v oblasti elektrotechniky, strojárstva a stavebníctva.

Vizuálna simulácia je oblasť počítačovej grafiky, ktorá sa rozvíja najmä v dvoch smeroch. Prvým je vizualizácia výsledkov rôznych simulačných programov napr. priebehy napätí a prúdov v elektrických obvodoch. Druhým je simulácia reálnych udalostí v rôznych počítačových tréningoch napr. leteckých alebo automobilových.

Spracovanie obrazu je časťou počítačovej grafiky, do ktorej zasahujú aj iné odbory. V princípe sem môžeme zahrnúť najmä procesy spracovania farieb, detekcie hrán, vyhladzovanie obrazu a pod. Vo veľkej miere sa používajú tieto postupy napr. v grafických editoroch ale aj v náročných vojenských alebo vesmírnych projektoch, napr. na vylepšenie prichádzajúcich snímok alebo v procese rozpoznávania obsahu obrazu.

Fotorealistické zobrazenie, multimédiá a virtuálna realita sú hitom dnešnej doby. U prvého ide o čo najvernejšie zobrazenie priestorových scén a objektov vrátane osvetlenia a riešenia viditeľnosti. Dokonalé sklbenie kvalitného obrazu a zvuku je doménou multimédií. Počítačom nagenarovat' dokonalý obraz a preniesť ho na videopásku spoločne s počítačom generovanou hudbou príp. hovoreným slovom ale vyžaduje kvalitné a rýchle technické ale aj programové vybavenie. Ak sa ku kvalitnému programu pridá dostatočne výkonný počítač, ktorý umožňuje toto prepočítavanie v reálnom čase je možné simulovať v danom čase neexistujúci svet (*virtual reality*), napr. "prechádzať sa" ešte v nepostavenom vlastnom dome Informačné systémy a počítačové siete v spolupráci s prostriedkami virtuálnej reality (najmä formu netradičného rozhrania komunikácie človek počítač) začínajú tvoriť nový fenomén veľmi blízkej budúcnosti.

O princípoch technológií, ktoré počítačová grafika používa je práve nasledujúci text.



1. OKRUH

2 Grafické prostriedky počítačov

Ciel': Po preštudovaní tejto kapitoly by mal byť študent schopný kategorizovať periférie počítačov používané v počítačovej grafike podľa rôznych hľadísk. Ďalej by mal získať prehľad o základných vstupných (ako je myš, trackball, joystick, tablet, scanner a optické pero) resp. výstupných (najmä tlačiarne, súradnicové zapisovače, monitory a zobrazovacie adaptéry) zariadeniach počítačov z pohľadu počítačovej grafiky. Rozšírené potrebné vedomosti by mal študent získať z oblasti grafických zobrazovacích adaptérov používaných v počítačoch na báze PC.

Ako už bolo v úvode povedané, oblasti nasadenia grafických počítačových systémov sú veľmi rôzne. Z hľadiska toho, že počítačová grafika asi najviac využíva senzitivnosť človeka a tým pádom možnosť vysokej komunikatívности, sú grafické periférie počítača definované ako user friendly t.j. používateľsky priateľské.

2.1 Rozdelenie grafických zariadení

Všeobecne rozdeľujeme grafické zariadenia počítača podľa rôznych hľadísk.

PODĽA SMERU KOMUNIKÁCIE S POUŽÍVATEĽOM NA:

- ☐ vstupné
- ☐ výstupné

Vstupné zariadenia najčastejšie akceptujú ako vstupnú grafickú informáciu x-ové a y-ové (niekedy aj z-ové) súradnice zadaného bodu resp. zmenu x-ovej a y-ovej súradnice, tieto prevedú do číselnej formy a poskytnú počítaču na ďalšie spracovanie.

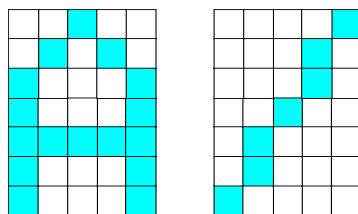
Výstupné zariadenia naopak zase graficky zobrazia polohu počítačom špecifikovaného bodu resp. grafického elementu, prípadne navyše jeho farbu alebo odtieň (súradnicové zapisovače alebo grafické monitory či tlačiarne).

PODĽA SPÔSOBU VYTŮVARANIA GRAFICKEJ INFORMÁCIE NA:

- ☐ vektorové
- ☐ rastrové

Vektorové zariadenia sa používali skôr na začiatku éry počítačovej grafiky. Dôvod bol pomerne jednoduchý. Na uchovanie grafickej informácie potrebujú kapacitne menšiu pamäť ako zariadenia rastrové. S rozvojom technológie a tým zlacnením najmä polovodičových pamätí bolo však aj toto zotreté. Zariadenia vektorového typu zobrazujú grafické entity tak, ako keby boli kreslené perom na papieri. Teda v podstate sa kreslí vektor, pretože kresliaca hlava (resp. elektrónový lúč) sa pohybuje od začiatku napr. úsečky do jej koncového bodu. Pri tomto sa nemusia kresliť len rovné čiary, ale dosť často sa využíva aproximácia (alias) do schodovitej čiary. Obdobne aj pri vektorových displejoch. Zobrazovacia plocha je síce reprezentovaná maticou bodov, avšak zobrazovanie je odlišné od rastrových displejov. Vychýľovanie lúča je riadené a lúč teda vysvieti presne želanú úsečku. Mnohokrát sa používajú pamäťové obrazovky a nie je nutné potom obraz obnovovať.

Rastrové zariadenia sa začali používať neskôr, najmä kvôli svojim pamäťovým nárokom. Rastrové zobrazovače pracujú obdobným spôsobom ako televízor. Zobrazovacia plocha je daná maticou bodov (obrazových prvkov, pixelov). Rastrové zariadenie pri zobrazovaní potom riadi vysvietenie a tým aj farbu a jas každého z týchto bodov. Na uchovanie zobrazenej informácie sa používa obrazová pamäť (videopamäť). Potom podľa počtu farieb, je každému pixelu priradený príslušný počet bitov. Postupným skladaním jednotlivých pixelov dostaneme výsledný obraz.



Obr. 1 Príklad rastrového zobrazenia znaku a úsečky.

Rastrové zariadenia dnes už dobyli svet. Dokážu zobrazovať celé plochy viacfarebne a tým zobrazovať obrazy vernejšie reálnemu svetu. Prvé rastrové displeje pracovali väčšinou čiernobielo alebo len s malým počtom farieb (max. 8) a s malou rozlišovacou schopnosťou. Pod rozlišovacou schopnosťou rastrového zariadenia rozumieme koľko bodov je zariadenie schopné zobraziť maximálne v smere osi x (vodorovný smer) a v smere osi y (zvislý smer). Postupným vývojom sa zväčšoval jednak počet zobrazovaných farieb a jednak rozlišovacia schopnosť. Súčasné komerčné rastrové displeje hravo dokážu 1024x768 v cca 16mil. farbách. Špičkové zariadenia majú samozrejme vyššiu rozlišovaciu schopnosť. Trend vylepšovania rastrových displejov stúpa a tak vektorové zobrazovače sa dostali do úzadia.

PODĽA SCHOPNOSTI SPRACÚVAŤ GRAFICKÉ INFORMÁCIE NA:

- ☐ *aktívne*
- ☐ *pasívne*

Aktívne grafické zariadenia majú schopnosť zobrazovať grafickú informáciu a zároveň aj akceptovať inú vstupnú grafickú informáciu, ktorá môže určitým spôsobom ovplyvniť zobrazenie želannej grafickej informácie (napr. grafické displeje s osovým krížom).

Pasívne grafické zariadenia slúžia najčastejšie na uchovanie vytvorenej grafickej informácie. Medzi takéto zariadenia patria aj súradnicový zapisovač alebo trysková tlačiareň.

PODĽA DĹŽKY UCHOVANIA GRAFICKEJ INFORMÁCIE NA:

- ☐ *zariadenia s dočasným záznamom*
- ☐ *zariadenia s trvalým záznamom*

Zariadenia s dočasným záznamom grafickej informácie slúžia na pohotovú, rýchlu a prípadne aj dynamickú zobrazenie grafickej informácie. Medzi takéto zariadenia patria najmä displeje. Záznam však nie je trvalý a napr. po vypnutí napájania sa stráca.

Zariadenia s trvalým záznamom grafickej informácie síce nie sú určené na rýchle zobrazenie, ale na "hmatateľné" zobrazenie. Najčastejším médiom, na ktorý je grafická informácia prenesená je papier. Medzi takéto zariadenia patrí najmä súradnicový zapisovač a trysková či laserová tlačiareň.

PODĽA MOŽNOSTI SAMOSTATNÉHO SPRACOVANIA GRAFICKEJ INFORMÁCIE NA:

- ☐ *neinteligentné*
- ☐ *inteligentné*

Toto delenie je jednoduché, ale tiež logické. *Neinteligentné systémy* nevedia inak nakladať s grafickou informáciou, len na úrovni kódu, ktorý prijali od nadradeného počítača. Všetko riadenie grafického zobrazovania je prenechané na nadradený pripojený počítač.

2.2 Vstupné grafické zariadenia

2.2.1 Myš

Patrí v súčasnej dobe už medzi základné vstupné zariadenia. V podstate sa do maximálnej miery v aplikačnom softvéri obchádza už použitie klávesnice a využíva sa myš. Robí sa to najčastejšie výberom z ponúk na obrazovke a rýchlym pohybom kurzora (grafického ukazovateľa spriahnutého s pohybom myši, najčastejšie vo forme šípky) po obrazovke.

Myš je v podstate malá škatuľka s dvomi alebo tromi tlačidlami. V súčasnosti sú k týmto tlačidlám pridávané rôzne ďalšie ovládacie a navigačné prvky (napr. rolovacie koliesko), ktoré sú určené najmä na zrýchlenie rolovania obrazu, čo je s výhodou používané najmä pri prezeraní niektorých stránok na Internete. Na spodnej strane myši je gumová guľička, ktorá sa pri pohybe myši odvaluje po podložke. Otáčanie guľičky je v x-ovom a y-ovom smere snímané a vysielané do počítača. Použitie myši a význam jej tlačidiel závisí od programu.

Takto definovaná myš je klasická opto-mechanická myš. Existujú rôzne variácie napr. *optická myš* alebo *ultrazvuková myš*. U optickej myši je guľička nahradená dvojicou svetelného vysielača a prijímača (najčastejšia na báze LED diód). Pohyb je potom snímaný zmenou odrazu vysielaného lúča od podložky. Znížením cien optických myši dochádza v súčasnosti k ich masovejšiemu používaniu, pretože majú lepšiu citlivosť a sú trvácnejšie ako opto-mechanické guľčkové myši. Na podobnom princípe ako optická, pracuje aj ultrazvuková myš. Všetky takto pracujúce myši sú vhodné na vstup dvojrozmernej informácie. V súčasnosti sa najmä na posledne dvoch spomínaných bázach začínajú objavovať aj myši umožňujúce trojrozmerný vstup. Tretia súradnica je definovaná vzdialenosťou myši (jej zdvihnutím) od podložky. Svoje pevné miesto si však tieto ovládače ešte musia vy dobyť.

Ďalším pojmom, s ktorým sa u myši stretne, je jej rozlišovacia schopnosť čiže presnosť. Táto sa udáva v *DPI* (Dot Per Inch = bodov na palec). Klasické opto-mechanické myši dosahujú presnosť od 200 dpi do 2400 dpi. Optické a ultrazvukové aj 6000 dpi. Napriek tomu je myš pomerne nepresné polohovacie zariadenie, nakoľko sa pri ich ovládaní ľahko prejaví nestabilita rúk ovládajúcej osoby, čo niekedy spôsobuje veľké chyby.

Myš sa najčastejšie pripája k počítaču pomocou kábla na sériový port alebo na PS/2 či USB port a menej často na paralelný port. Vyrábajú sa aj varianty s bezdrôtovým pripojením na báze infračervených lúčov alebo rádiových vln. Tieto, hoci sú elegantné, sú drahšie, čo bráni ich masovejšiemu nasadeniu.



Obr. 2 Optická myš s rolovacím kolieskom

2.2.2 Trackball

Trackball je veľmi populárne vstupné polohovacie zariadenie najmä u prenosných počítačov. Z hľadiska riadenia polohy si môžeme trackball predstaviť ako obrátenú myš. Je to v podstate buď malá samostatná krabička alebo väčšinou zabudovaná do krytu klávesnice resp. notebooku. Tentoraz je ale na vrchnej strane trackball-u guľička, ktorou sa pohybuje prstami. Guľička sa odvaluje a jej otáčanie je v x-ovom a y-ovom smere snímané a vysielané do počítača. Na vrchnej strane vpredú sú podobne ako u myši umiestnené dve alebo tri tlačidlá. Jeho použitie je ďalej podobné ako u myši. Všetci výrobcovia myši majú vo svojom výrobnom programe aj trackball-y.

2.2.3 Trackpad

Jedná sa opäť o ukazovacie vstupné zariadenie. Princíp je podobný ako u tabletu avšak spôsob snímania je odporový. Stačí sa po ploche trackpadu (jej veľkosť je približne plocha kreditnej karty) pohybovať prstom príp. špicatým predmetom a kurzor sleduje pohyb prstu. Dvojším poklopaním na miesto sa vykoná potvrdenie. Toto zariadenie sa montuje dnes už do notebookov namiesto trackball-ov.

Dnes sa ako polohovacie zariadenie namiesto *trackpad*-u alebo *trackball*-u používa aj tzv. **trackpoint**. Jedná sa v princípe o malý joystick umiestnený medzi klávesmi klávesnice (napr. medzi **h** a **j**) a jeho vychýľovanie je pomerne jednoduché aj pomocou jedného prstu.



Obr. 3 Typické umiestnenie trackpadu na notebooku

2.2.4 Joystick

Joystick alebo tiež pákový krížový ovládač patril a aj patrí medzi obľúbený vstupný ovládací prvok najmä u domácich počítačov. Jeho obľuba a nasadenie prišlo s rozvojom počítačových hier. Používateľ ho drží v jednej ruke (resp. je uchytený na stôl) a v druhej ruke drží ovládaciu páku joysticka. Vychýľovaním páky v príslušnom smere riadi grafický kurzor (u hier napr. postavičku, auto a pod.) na obrazovke. Používanou obdobou joysticka sú tzv. *gamepady*, kde nie je páka, ale príslušný počet tlačidiel resp. riadiaci kríž. Joystick v základe býva doplnený o jedno, dve alebo viac nezávislých potvrdzovacích tlačidiel (u hier napr. na strelbu a pod.). Podľa základnej konštrukcie, ktorá zisťuje smer vychýlenia riadiacej páky, rozdelujeme joysticky na:

- ❑ *digitálne*
- ❑ *analógové*

U tých prvých je možné zjednodušene povedať, sa pri vychýlení v príslušnom smere zopne len spínač a počítač detekuje podľa toho smer. Nie je možné detekovať promptnosť zopnutia ani veľkosť výchylky. Všetka dynamika je generovaná elektronikou.

Pri druhom type je v x-ovom aj y-ovom smere spriahnutý potenciometer. Na konci býva väčšinou ešte dorazový spínač. Vychýlením zo strednej polohy dôjde k zmene odporu potenciometra v príslušnom smere. Táto zmena je analógovo-digitálnym prevodníkom prevedená do číslícovej formy a spracovaná. U takto konštruovaných pákových ovládačov je možné detekovať aj rýchlosť zmeny vychýlenia (promptnosť) aj veľkosť výchylky (podľa veľkosti odporovej zmeny). Niektoré PC bývajú vybavené tzv. game-portom (15-vývodový), na ktorý je možné pripojiť práve tento typ pákového ovládača. V súčasnosti sa čoraz častejšie používa na pripojenie aj USB port.



Obr. 4 Joystick a gamepad

V súčasnosti sa začínajú čoraz častejšie objavovať joysticky pre priestorové ovládanie, najmä z dôvodu nových efektných 3D hier, ale aj nižších systémov virtuálnej reality. Pre pohyb v osiach x a y je zachovaná pôvodná koncepcia. V osi z sa pohybujeme zatláčaním resp. povytáňovaním ovládacej páky. Hitom poslednej doby sú joysticky so spätnou väzbou t.j. napr. pri leteckých simulátoroch počítač generuje späť riadiacemu obvodu joysticka signály, na základe ktorých dochádza napr. k traseniu riadiacej páky, čo má za následok zvýšenie reálnosti hrania.

Veľmi jednoduchým, ale sympatickým je tzv. **keystick**. Jedná sa o jednoduchý prípravok v tvare joysticku, ktorý sa upevní na klasickú klávesnicu nad klávesy s kurzorovými šípkami. Pri ovládaní sa potom vychýľuje riadiaca páčka, ktorej vychýlenie je prenášané priamo na klávesy, ako keby boli stláčané.



Obr. 5 Keystick

2.2.5 Tablet

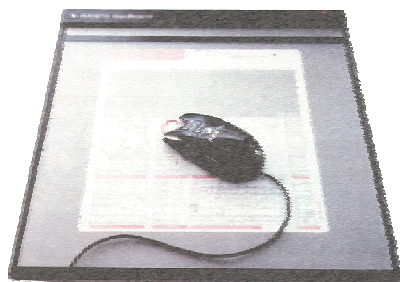
Patrí medzi vstupné zariadenia. Výzorom mobilnej časti pripomína myš a aj komunikácia s programami simuluje myš. Tablet dosahuje však vyššiu presnosť a rozlíšenie. Myš väčšinou sníma zmenu polohy pomocou odval'ovania guličky na spodnej časti. Tablet sa však skladá až z troch súčastí: *plochej snímacej podložky, pohyblivého snímacieho zariadenia* (tvaru myši alebo pera) *a riadiaceho elektronického obvodu.*

Plochá snímacia podložka (tabuľa) je doska s hladkým povrchom, pod ktorým je umiestnený citlivý elektromagnetický obvod. Táto časť je nepohyblivá.

Pohyblivé snímacie zariadenia, či už tvaru myši alebo pera, je vybavené snímačom elektrických signálov z elektromagnetického obvodu tabule.

Riadiaci elektronický obvod je väčšinou umiestnený v tabuli. Tento niekoľkokrát prechádza sieťou tabule a sníma impulzy z pohyblivej časti. Pomocou toho detekuje polohu pohyblivej zložky. Ak si predstavíme tabuľu ako obrazovku, sme schopní riadiť na nej kurzor pohybovaním pohyblivej časti.

Častokrát je tablet používaný, po prekrytí príslušnou šablónou, ako riadiaci panel s menu pre program v počítači (veľmi často v CAD aplikáciách). Niekedy sa tablet zvykne tiež nazývať aj *digitizér*.



Obr. 6 Tablet

2.2.6 Scanner

Je zariadenie na snímanie obrazov. K počítaču sa pripája:

- ❑ pomocou zvláštnej karty (v súčasnosti sa už tento spôsob takmer nepoužíva)
- ❑ pomocou rozhrania *SCSI*
- ❑ pomocou paralelného portu (pripojí sa do konektora pre tlačiareň a na scanneri je potom konektor pre tlačiareň t.j. scanner je pre tlačiareň priechodzí)
- ❑ pomocou rozhrania *USB*

Prevedenia s pripojením cez paralelný port sú v súčasnosti typické najmä pre lacné typy plošných scannerov. Daňou za to je dlhá doba snímania obrazu. Ak teda nie je faktor cenový taký podstatný, ale dominuje rýchlosť, je výhodnejšou voľbou verzia *SCSI*. V súčasnosti sa vhodným kompromisom môže stať *USB* verzia pripojenia. Scannery sa vyrábajú dvojakého typu:

- ❑ *ručné*: lacnejšie, menej presné a s menšou rozlišovacou schopnosťou (150-300 dpi), čierno-biele aj farebné snímanie.
- ❑ *plošné (automatické)*: drahšie, presnejšie (pohyb snímacej hlavy nad predlohou je motorický) a s vyššou rozlišovacou schopnosťou (300-1200 dpi), čierno-biele aj farebné (aj Truecolor) snímanie. V poslednej dobe došlo k výraznému zníženiu cien jednoduchých plošných scannerov pre bežnú veľkosť A4.

Každý scanner sa v súčasnosti dodáva aj so základným programovým vybavením, častokrát aj s grafickým editorom na úpravu nasnímaných obrazov. Potom pomocou týchto alebo iných grafických editorov je možné nasnímané obrázky ďalej upravovať alebo ukladať v rôznych grafických formátoch pre následné využitie v publikačnej činnosti (*DTP*). Pomocou zvláštného programového vybavenia (*OCR - Optical Character Recognition*) je možné zosnímať graficky aj text a potom ho rozanalyzovať a prekonvertovať do znakovkej formy.



Obr. 7 Plošný scanner

V súčasnosti sa hojne používajú súradnicové zapisovače alebo tlačiarne so scannovacou hlavou, čím dostaneme dve zariadenia v jednom. V CAD systémoch sa používajú na nasnímanie starých výkresov najprv do rastrovej formy. Potom pomocou programu sú tieto rastrové obrázky konvertované do vektorového formátu, ktorý je vhodnejší pre ďalšie spracovanie v CAD systémoch, ale aj na archiváciu.

2.2.7 Optické pero

Používa sa ako grafické vstupné zariadenie v spojení s grafickým displejom. Umožňuje interaktívne zasahovať do obrazu na displeji. Tvar je ceruzkovitý, pero je káblom pripojené k počítaču a navyše je spravidla vybavené ešte potvrdzovacím tlačidlom. Vnútri pera je umiestnený fotocitlivý prvok (napr. fototranzistor). Po priložení pera k obrazovke na príslušný bod a potvrdení potvrdzovacím tlačidlom sa zosníma poloha pera. Zistenie polohy je pomerne jednoduché, pretože k aktivačnému impulzu dôjde pri prechode elektrónového lúča oblasťou, kde je pero. V momente ako sa zisti aktivačný impulz, je počítačom na základe synchronizácie prepočítaná poloha bodu v momente aktivácie a súradnice bodu sú tým vyčíslené.

2.3 Výstupné grafické zariadenia

2.3.1 Tlačiareň

Tlačiareň je dnes základným výstupným zariadením počítača, ktoré umožňuje výstup najčastejšie na papier. Počítač vysiela do tlačiarne jednak informácie (znaky), čo sa má tlačiť, ale aj informácie o tom, ako sa má tlačiť. Prvá skupina informácií sa nazýva údajová časť, druhú skupinu nazývame riadiace znaky. Pomocou nich môžeme napr. zmeniť mód tlače, typ písma, riadkovanie a pod. To je u tých jednoduchších. Drahšie tlačiarne komunikujú s nadradeným počítačom väčšinou pomocou určitého jazyka (napr. PostScript, PCL a pod.). Podľa typu tlače poznáme rôzne typy tlačiarní napr.: *maticové* (ihličkové), *tryskové* (InkJET), *piezoelektrické*, *bublinové*, *tepelné*, *tlačiarne s typovým kolieskom*, *sublimačné*, *laserové* alebo *LED*, *osvitové jednotky*.

Ďalším pojmom, ktorý udáva kvalitu tlačiarne, je jej rozlišovacia schopnosť čiže presnosť. Táto sa udáva v *DPI* (Dot Per Inch = bodov na palec). Najnižšiu rozlišovaciu schopnosť majú maticové tlačiarne (75-150 dpi), najvyššiu laserové tlačiarne (300-1200 dpi) a osvitové jednotky.

Zvláštnym výstupným zariadením, ktoré v tejto publikácii zaradíme medzi tlačiarne, je *osvitová jednotka*. Používa sa na prípravu kvalitných predlôh najmä v tlačiarenskej praxi. Výstup je na špeciálny film a jej rozlišovacia je schopnosť je až 3200 dpi a viac.

Pri výbere tlačiarne je veľmi dôležité uvedomiť si, či je pri tlačení dokumentov požadovaná farebná tlač. Ak áno, cena hlavne laserových tlačiarní rastie geometrickým radom. Pri náročných požiadavkách možno využiť tlačiareň založenú na tzv. *sublimačnej technológii*. Tento druh tlačiarne sa od ostatných odlišuje zásadným rozdielom v spôsobe nanášania farby na papier. Pri všetkých ostatných typoch tlačiarní sa jednotlivé farebné odtiene dosahujú miešaním malých farebných bodov na papieri, kdežto pri sublimačnej technológii sa jednotlivé farebné zložky miešajú ešte pred nanosením na papier. Tým sa dosahujú veľmi kvalitné farebné prechody, porovnateľné s farebnou fotografiou.

- **Maticové tlačiarne** sú ešte stále najrozšírenejšie tlačiarne k osobným počítačom. Aj keď ich éra obľuby je za nami, ešte stále majú svoje miesto v dnešnom svete tlačiarní (napr. jednoduché tlačiarne pre registračné pokladnice). Sú to najlacnejšie tlačiarne aj vzhľadom k obstarávacím aj prevádzkovým nákladom. Vzhľadom na kompatibilitu väčšinou pracujú v móde EPSON alebo v móde IBM. Rozdiel medzi módmi je v riadiacich znakoch. Podľa kvality tlačiarne rozlišujeme maticové tlačiarne na 9-ihličkové a 24-ihličkové. Samozrejme, že 24-ihličkové poskytujú vyššiu kvalitu tlače ako tlačiarne 9-ihličkové. Z hľadiska tlače znakov rozlišuje tri kvality tlače:

- ❑ **DRAFT** - používa sa pri bežnej tlači. Je to najrýchlejší tlačový režim, ale aj najmenej kvalitný. Slúži napr. na kontrolné výpisy programov a pod. Táto kvalita je dosiahnuteľná ako na 9-ihličkových, tak aj na 24-ihličkových tlačiarniach.
- ❑ **NLQ** (Near Letter Quality) - je to kvalitnejšia, ale aj pomalšia tlač ako DRAFT. Slúži na bežné dokumenty, príp. vnútropodnikové dokumenty. Je dosiahnuteľná na oboch typoch tlačiarní. Pre 9-ihličkovú však predstavuje už vrchol kvality.
- ❑ **LQ** (Letter Quality) - je to najkvalitnejšia tlač na ihličkových tlačiarniach. Tlačí sa ňou korešpondencia príp. mimopodnikové materiály. Je dosiahnuteľná len na 24-ihličkových tlačiarniach a je aj najpomalšia.

Tlačiť na maticovej tlačiarňi je možné, podobne ako u monitorov, buď v textovom móde alebo v grafickom móde. Pri tlači v grafickom móde sa posielajú informácie o bodoch tlačenej mozaiky. Táto tlač je výrazne pomalšia ako tlač v textovom móde. V tomto móde sa posielajú tlačiarňi len kódy znakov a tlačiareň už sama "vie" z akej matice bodov sa znak skladá. Podobne ako monitor v textovom móde, aj tlačiareň má svoju sadu znakov - font. Font znakov môže byť uložený buď v pamäti ROM alebo je nutné najprv z riadiaceho počítača nahráť font do pamäte RWM. O tlačiarniach, ktoré majú takúto možnosť, hovoríme, že majú **DOWNLOAD**. Výhodou **DOWNLOADu** je, že používateľ si môže definovať vlastné fonty. Takto je možné naučiť tlačiareň aj slovenskú príp. českú abecedu aj keď dnes je už spravidla používaná grafická tlač, kde tento problém odpadá. V maticových tlačiarniach sa používajú dva druhy papierov:

- ❑ perforovaný skladaný pás papiera (tzv. traktor-papier) a
- ❑ samostatné listy kancelárskeho papiera.
- **Tryskové tlačiarne** predstavujú technologicky iný typ tlače ako tlačiarne maticové. Kým u maticových je na papier otláčaná farbiaca páska (podobne ako pri písacích strojoch), pri tryskových sú na papier striekané malé kvapôčky špeciálneho atramentu. Preto sa týmto tlačiarniam hovorí často aj *atramentové*. Trysková hlava obsahuje niekoľko trysiek, väčšinou 64. Atrament je v tryskách

pod tlakom a uzávery trysiek sú ovládané elektromagneticky. Písmo sa opäť skladá z bodov, kvalita tlače je však vyššia (300 DPI a viac). Tlačiarne tohto typu sa vyznačujú pomerne tichým chodom. V súčasnosti tento typ tlačiarň zaznamenal búrlivý rozvoj a ich cena klesla na, niekedy aj pod cenu maticových tlačiarň. Výrazný pokrok doznela najmä farebná častokrát fotorealistická kvalita tlače za veľmi slušnú cenu. Ak k tomu prirátame pomerne malé obstarávacie a prevádzkové náklady, pri danej kvalite tlače, niet sa čomu diviť, že tento typ tlačiarň je najviac obľúbený v domácnostiach a kanceláriách. V tých sa síce používajú aj laserové tlačiarne, oproti ktorým súčasné bežné tryskové tlačiarne sú pomalšie v rýchlosti tlače. Nie je však nezaujímavé, že za pomerne „lacný peniaz“ je možné pomocou tryskových tlačiarň vyrobiť kvalitné fóliové prezentačné priesvitky alebo dokonca vytlačiť nažehľovací obrázok, ktorý si následne môžete nažehliť na svoje tričko (trebárs aj Vašu podobizeň).

- **Laserové a LED tlačiarne** predstavujú v súčasnosti najdokonalejšie tlačiarne. Oproti predchádzajúcim typom sa líšia najmä tým, že tlačia celé strany dokumentov naraz, kým u tých predchádzajúcich to bolo po znakoch. U laserových sa najprv celá strana pomocou lasera "pripraví" na fotocitlivý valec. U LED tlačiarň je laser nahradený lacnejšou maticou svietiacich LED diód. Potom sa takýto valec pritlačí k tonerovému valcu (toner je jemný čierny prášok a slúži ako farbivo). Tam, kde laserový (LED) lúč pôsobil sa toner na fotocitlivom valci zachytí. Následne sa posúva papier a naň je z valca otláčaný želaný obraz. Na konci sa musí ešte papier s tonerom vytvrdiť v tzv. piecke, aby sa toner nezošúchal z papiera.

Laserové (LED) tlačiarne majú vysokú rozlišovaciu schopnosť 300 a viac DPI (Dot Per Inch - bodov na palec). Laserové oproti LED tlačiarňam poskytujú vyššiu rozlišovaciu schopnosť (aj nad 800 DPI), vyšší kontrast a snád' sýtejšie farby (to je ale subjektívne), pre LED tlačiarne hovorí najmä cena.

Rýchlosť a kvalita tlače je veľmi vysoká. Minimálne štyri a viac strán za minútu. Väčšina laserových tlačiarň komunikuje s nadradeným počítačom pomocou určitého jazyka. Z nich dominuje jazyk PostScript a PCL. Technológia tlače umožňuje čierno-bielu aj farebnú tlač, prevláda však (najmä kvôli cene) čierno-biela tlač.



Obr. 8 Rôzne typy tlačiarň: (zľava) maticová (ihličková), trysková (atramentová) a laserová

2.3.2 Súradnicový zapisovač, ploter

Je ďalším výstupným zariadením. Pomocou pera sa kreslia výstupy na vložený papier. Využíva sa najmä v oblasti počítačovej grafiky a CAD systémov. Súradnicové zapisovače môžeme rozdeliť podľa niekoľkých hľadísk.

- ❑ **podľa používaného pera** na *tušové, fixové, s guľčkovým perom* a *tryskové*.
- ❑ **podľa počtu používaných farieb** na *jednofarebné* a *viacfarebné*.
- ❑ **podľa spôsobu uloženia a pohybu papiera** na *stolové* a *valcové*. Pri stolových sa pohybuje nad papierom hlava v oboch smeroch. Pri valcových sa pohybuje hlava len v jednom smere a v druhom sa pohybuje celý papier.
- ❑ **podľa spôsobu riadenia** na *analogové* a *digitálne* (dnes už dominantne).

Mnohokrát sa miesto pera použije ostrý hrot, pomocou ktorého je do podložky vyrezaný želaný motív. Takto upravený súradnicový zapisovač sa nazýva aj *cuter*. Pomocou neho sú napr. vyrezávané reklamy zo samolepiacich fólií.

Pri kreslení je ovládaný pohyb pera v jednotlivých smeroch, jeho zdvihnutie resp. spustenie na papier a pri farebnom súradnicovom zapisovači aj výmena jednotlivých pier podľa požadovaných farieb.

Inteligencia súčasných súradnicových zapisovačov je pomerne vysoká. Tieto zariadenia obvykle pracujú aj s vyššími grafickými entitami ako je bod. Spravidla sú takéto súradnicové zapisovače vybavené vlastným komunikačným jazykom vyššej úrovne (napr. HPGL).

Významnou triedou sú tzv. *fotoplotre*. Namiesto papiera používajú fotocitlivý film a miesto pera osvetľovaciu hlavu, najčastejšie laserovú. Používajú sa najmä v CAD/CAM systémoch napr. na tvorbu predlôh pri výrobe plošných spojov.



Obr. 9 Stolový (plošný) a stojanový (valcový) ploter

2.3.3 Zobrazovače, monitory

Monitory alebo tiež displeje patria k základným výstupným grafickým zariadeniam. Ako bolo spomenuté skôr, boli vyrábané aj vektorové monitory. Vo vektorom displeji je úsečka reprezentovaná súradnicami počiatku a koncového bodu. Elektrónový lúč teda neprechádza cez celé tienidlo obrazovky, ale len cez úsečku. V súčasnosti sa však z väčšej miery používajú práve monitory rastrového typu. Pri tejto príležitosti treba spomenúť, že mnohé moderné zobrazovacie zariadenia sa z hľadiska programátora môžu javiť ako vektorové, konečný spôsob zobrazenia je však rastrový. Na tento prevod je v zobrazovacom adaptéri použitý špeciálny grafický procesor. Preto, po grafickej stránke, významnú úlohu pri zobrazovaní grafickej informácie nemá ani tak samotný monitor, ako skôr zobrazovacia karta (grafický adaptér), na ktorú je monitor pripojený.

- **Katódové obrazovky (CRT)** si udržiavajú dominantnú pozíciu medzi zobrazovačmi. Tieto sú v podstate takmer nepostrádateľnou súčasťou zariadení na spracovanie údajov (najmä a v prvom rade počítačov), pretože majú stále najvýhodnejší pomer cena k počtu a rýchlosti zobrazovaných informácií. Majú však pomerne veľkú spotrebu a hlavne sú pomerne hlboké, čím sa nehodia do

prenosných zariadení, ktoré sú v súčasnosti mimoriadne populárne. V týchto zariadeniach je potrebné aby bol zobrazovač plochý, s malou hĺbkou a spotrebou. Aj z tohto dôvodu bolo navrhnutých niekoľko iných technických riešení, najmä na zredukovanie hĺbky. Prvým je také riešenie, že hrdlo obrazovky je ohnuté o 90. Emitovaný zväzok elektrónov tak spočiatku letí rovnobežne s tienidlom a potom, v závere dráhy je ohnutý o 90 a dopadá na luminofor tienidla ako pri bežnej obrazovke. Druhé riešenie prezentuje patent japonskej firmy Matsushita. Tento patent je založený na úplne inom princípe. Napr. Obrazovka s uhlopriečkou 36cm má hĺbku iba 6.25cm. Klasická elektrónová dýza je nahradená plochým extraktom s 15x200 otvormi. Za nimi je 15 vláknových katód prebiehajúcich cez celú šírku obrazovky. Tým pádom zhruba 3000 elektrónových lúčov dopadá po prechode vychyľovacími a zaostrovacími elektródami na cca 192000 trojfarebných obrazových bodov (pixelov). Každý pixel je je samostatne adresovateľný. Celková spotreba obrazovky je len 7.5W. Oproti klasickej obrazovke má však cca o 25% menší jas. Okrem niekoľkých aplikácií sa však tieto konštrukcie v praxi neuplatnili. Z celkového pohľadu sa zdá, že aj keď sa konštruktéri a výrobcovia obrazoviek snažia neustále parametre obrazovky zlepšovať, je čas tejto poslednej spotrebnej elektrónky zrátať a začínajú sa presadzovať iné druhy zobrazovačov.

- **Plazmové zobrazovače** sú ďalším typom zobrazovača. Prenosné počítače stále viac získavajú na obľube. Tým stále viac aj rastú požiadavky na príslušné zobrazovacie prvky. Požiadavke tenších a ľahších zobrazovacích prvkov pre prenosné prístroje vyhovujú predovšetkým plazmové zobrazovače. Tieto najmä v monochromatickej verzii verzii sú dnes už výrobné naplno zvládnuté. Majú aj priaznivé vlastnosti z hľadiska počtu zobrazovaných informácií. Optické zobrazenie požadovanej informácie vzniká privedením napätia na elektródy, ktoré sú hermeticky zatavené do skla. Vnútro zobrazovača je naplnené plynom (obyčajne neónom). Privedené napätie vyvolá svetielkovanie v tomto plyne. Plazmové zobrazovače (podobne ako elektrónkové obrazovky) patria medzi emisné typy zobrazovačov. Vyžarujú vlastné svetlo a tým sú energeticky náročnejšie ako LCD zobrazovače. Priemerná spotreba plazmového zobrazovača je cca 10-15 W. Súčasný plazmové zobrazovače sú svojimi vlastnosťami pre prenosné počítače jedny z najvýhodnejších spomedzi plochých zobrazovačov. Bežne sa vyrábajú s uhlopriečkou cca 26cm s rozlíšením 640x480 pixelov, ale vyrábajú sa aj typy s rozlíšením 1024x768 pixelov (špičkové modely 1600x1200). Oproti svojim najväčším konkurentom, ktoré predstavujú LCD zobrazovače, majú výhodu najmä vo veľkosti, kvalitnejšom a najmä rýchlejšom zobrazovaní. Takisto nie je nutné meniť rozhranie, pretože sú na úrovni riadenia kompatibilné s klasickými obrazovkami. Vývoj rýchlo speje k znižovaniu cien, k viacfarebným príp. plnofarebným modelom. Možno očakávať, že začnú konkurovať klasickým obrazovkám.
- **Elektroluminiscenčné zobrazovače** patria k ďalšiemu typu plochých zobrazovačov. Tieto zobrazovače sa uplatnili pri konštrukcii počítačových terminálov využívajúc elektroluminiscenčný jav. Obraz u týchto zobrazovačov vzniká aktiváciou fosforujúcej vrstvy ZnS dotovanej mangánom pri pôsobení silného elektrického poľa. Elektroluminiscenčné zobrazovače sú monochromatické, ich svetlo je žltoranžové a sú výrobné i technologicky náročné (SHARP, FINLUX). Aj z týchto dôvodov sú drahé.

- **Zobrazovače z tekutých kryštálov (LCD)** patria všeobecne k najperspektívnejšiemu typu plochého zobrazovača. Z tekutými zobrazovačmi sa dnes možno stretnúť všade, najznámejšie sú ich aplikácie v náramkových hodinkách a kalkulačkách. Najväčšou a najpodstatnejšou ich výhodou, oproti iným typom, je ich mnohonásobne nižší príkon. Medzi ďalšie výhody patria najmä nízka hmotnosť, malá hrúbka, technologicky nenáročná výroba a pritom vysoká výťažnosť výroby. Medzi nevýhody možno zaradiť nižší kontrast, malý uhol pozorovania, nevhodnosť na multiplexný režim zobrazovania ale najmä ich pomerná pomalosť. Niekedy sa medzi nevýhody považuje aj ich neschopnosť emitovať vlastné svetlo, čo vedie k nutnosti ich presvetľovania a tým ku zvyšovaniu spotreby.

Základný princíp LCD (Liquid Crystal Display) zobrazovača je veľmi jednoduchý. Materiál tekutého kryštálu (olejovitého resp. koloidného charakteru) je vložený medzi dve sklenené doštičky, ktoré majú z vonkajšej strany nanesený priesvitný polarizátor a z vnútornej strany vyrovnávací prípravok, ktorý spôsobuje, že molekuly kryštálov sú zoradené rovnobežne so sklom. Keďže polarizátory sú natočené vzájomne o 90° , molekuly kryštálu vytvoria deväťdesiatstupňový špirálový útvar, takže dopadajúce svetlo sa polarizuje, otáča o 90° a vychádza v tej istej rovine ako pri vstupe. Po privedení napätia na príslušnú elektródu (môžu mať rôzny tvar, sú priesvitné a implantované na nosné sklíčko) sa molekuly kryštálu pôsobením elektrického poľa vzniknutého nad elektródou zoradia kolmo na sklo, nedôjde k natočeniu o 90° a oblasť neprepúšťa svetlo - javí sa ako čierna. Výhoda je, že tvar elektródy môže byť rôzny, čo sa bohato využíva najmä v elektronických hrách.

Pre zobrazovanie informácií vo výpočtovej technike však kvalita kalkulačkového zobrazovača zďaleka nestačí. Vývoj preto musel ísť ďalej. V súčasnosti existuje niekoľko typov tekutých kryštálov a spôsobov riadenia zobrazovačov LCD. Medzi najznámejšie a aj najpoužívannejšie patria nematické kryštály, označované ako TN (Twist Nematic) s uhlom natočenia 90° a ich dokonalejšia verzia STN (SuperTwist Nematic) s uhlom natočenia až 270° (ako základná surovina sa používajú dnes najviac kyanobifenily). Okrem týchto poznáme ešte aj smectické kryštály a najnovšie sa venuje veľká pozornosť výskumu feroelektrických kryštálov. Medzi ich základné prednosti patrí kratší čas odpovede a preto sú vhodné na zobrazenie pohybujúcich sa obrazov pri zachovaní príslušného kontrastu. Zobrazovací jav popísaný skôr platí pre NT-kryštály. Existujú aj kryštály matické. U týchto je to s otáčaním roviny presne opačne. Bez prítomnosti elektrického poľa sa polarizácia nemení a s elektrickým poľom sa polarizuje. V princípe nebráni nič použitiu aj matických kryštálov, ale ich výroba je drahšia a preto sa používajú kryštály nematické.

Základné spôsoby riadenia LCD zobrazovačov sú dva. Prvé jednoduchšie je multiplexné riadenie a druhé drahšie je aktívne maticové zobrazovanie. Multiplexované displeje LCD sú síce lacnejšie ale horšej kvality. Preto sa pri zobrazovaní dáva prednosť aktívnemu maticovému adresovaniu, ktoré je síce drahšie, má nižšiu výťažnosť, ale poskytuje väčší kontrast. Aktívne súčiastky sú vytvorené priamo na vlastnom substráte zobrazovača, v mieste každého obrazového bodu (pixelu). Vo väčšine prípadov sú to tenkovrstvé tranzistory vyrobené z amorfného kremíka alebo z CdSe (Selenid Cadmia). Priamym použitím STN kryštálov vznikli LCD zobrazovače väčších rozmerov. Najprv boli výrobne zvládnuté monochromatické typy, či už tmavomodré znaky na

žltom pozadí, alebo biele znaky na tmavomodrom pozadí. Pridaním polarizátora, ktorý čiastočne spätne natáča kryštály pri žltom pozadí, alebo pridaním dichroického farbiva k typu s modrým pozadím, vznikli čiernobiele LCD zobrazovače. Technológie výroby LCD zobrazovačov sa neustále zlepšujú a v poslednom období sa objavilo niekoľko nových typov: D-ST (Double layer SuperTwist = dvojvrstvový supertwist), NTN (Neutralized STN), FTN (Formulated STN), MTN (Modulated STN) a CTN (Compensated STN). Tieto zobrazovače sa najviac hodia do prenosných prístrojov, sú však ešte stále pomerne drahé. Posledným stupňom vývoja LCD zobrazovačov je prechod k farebným modelom. Ide v podstate o tri jednoduché LCD vrstvy na sebe a každá je riadená jednou zložkou RGB signálu. Najďalej vo vývoji týchto typov sú najmä japonci (Sharp, Matsushita a pod.). Bežne sú predstavované už typy s uhlopriečkou 36cm a špičkové modely s 51cm dokonca až s 55 cm.

Takmer všetky doteraz popísané kryštály typu NT (STN, DST, FTN, MTN, CTN) patria medzi tzv. pasívne LCD zobrazovače. Medzi základné nevýhody, ako už bolo spomenuté skôr, patrí nutnosť presvietenia. Takisto aby sa zvýšil kontrast sú tieto technológie dopĺňané rôznymi typmi a počtom filtrov a podsvietenia. Okrem týchto typov sa v súčasnosti derú do popredia síce drahšie, ale kvalitnejšie aktívne LCD zobrazovače. Ich určitý princíp bol naznačený už skôr. Táto technológia využíva tranzistor ako kondenzátor, čo je známe z technológie výroby pamäti RAM. Táto technológia dostala názov TFT (Thin Film Tranzistor). Na vytvorenie napr. farebnej VGA je potrebné viac ako 900 000 tranzistorov čo odpovedá zhruba jednej 1MB DRAM. Rozdiel je len v tom, že tu je to na väčšej ploche ako u čipu DRAM. Všetky bunky musia byť ale funkčné, pretože obraz by bol zlý. Preto sa jedna bunka vytvára s redundantným počtom tranzistorov. Táto aktívna technológia poskytuje dostatočne jasný aj kontrastný monochromatický alebo farebný obraz. Je však drahá a predstavuje minimálne polovicu ceny celého počítača (notebooku). Zlacnenou metódou je technológia MIM (Metal-Isolator-Metal). Je to v podstate obdoba diódovej matice. Nie je nutná redundancia diód a tým je lacnejší. Nutné je však dodať, že má menej kvalitný obraz ako pri technológii TFT. Aj keď jedinou nevýhodou oproti pasívnym technológiám je vysoká cena týchto technológií, presadzujú sa stále viac a v budúcnosti s rozvojom technológie je možné rátať so znížením ceny.

Monitor počítača, ale takisto nazývaný displej, obrazovka či zobrazovacia jednotka, sa radí ku štandardným výstupom počítača. Počítač, podobne ako domácnosť, môže byť vybavený ako *monochromatickým* (napr. čierno-bielym, čierno-zeleným príp. jantárovo-čiernym) monitorom, tak aj monitorom *farebným*. Rozdiel je samozrejme v cene a farebný monitor stojí viac ako monitor monochromatický. V dnešnej dobe ale už ceny aj monitorov klesli natoľko, že monochromatické monitory používajú už kupujú menej a dávajú prednosť "farbe". Monitor pracuje v dvoch režimoch. Sú to:

- **Textový mód** sa nastavuje implicitne. Vtedy sú na obrazovku vypisované texty resp. znaky. Na to, aby sa znak vypísal stačí poslať len jeho kód. Tvar znaku je uložený v tabuľke znakov. Tabuľka znakov sa nazýva *font*. Často bývajú k dispozícii niekoľko fontov.
- **Grafický mód** je nutné zapnúť. Väčšina aplikácií to robí. Počas grafického módu sa predávajú informácie o každom bode, ktorý sa zobrazuje na monitore. Predáva sa teda pomerne veľké kvantum informácií. Ale na druhej strane, je možné v grafickom režime zobraziť skoro všetko, vrátane obrázkov.

Riadenie monitora je vykonávané prídavnou doskou (zobrazovacou kartou), ktorá sa tiež nazýva grafický zobrazovací adaptér. Typ adaptéra musí byť volený tak, aby mohol pracovať s príslušným typom monitora.

K základnej jednotke je monitor pripojený pomocou signálového kábla a väčšinou aj napájacieho kábla. Potom je možné zapnúť a vypnúť celú zostavu len sieťovým vypínačom základnej jednotky. Z hľadiska zobrazovacích možností a tým vlastne pracovnej plochy na obrazovke monitora, nutné ešte rozlišovať veľkosť (uhlopriečku) samotnej obrazovky monitora. Sú vyrábané monitory od 9" (palcov, 1 palec = 2,54 cm). Pre bežné domáce (teda častokrát najmä hry) alebo kancelárske použitie sa dnes ešte poväčšinou z hľadiska pomeru cena/výkon používa klasický 14" monitor, najmä farebný. Ak je však predpoklad práce v grafickom rozhraní (Windows95/98/NT a pod.) je vhodné si priplatiť a kúpiť aspoň 15" monitor najlepšie s digitálnym riadením. Optimálne pre prácu s počítačovou grafikou, multimédiami, CAD systémami a programami pre sadzbu dokumentov (DTP) sa doporučuje monitor s väčšou uhlopriečkou aspoň 17" optimálne až 21". Hitom poslednej doby sa stali najmä ploché LCD monitory. Tieto typy sa síce už štandardne používajú pri prenosných typoch počítačov, avšak dnes vo forme veľkých plochých monitorov čoraz dôraznejšie útočia na pozíciu klasických CRT monitorov, ktoré sa dominantne používajú pri stolných zostavách PC. Ich použitím sa znižuje spotreba, namáhanie očí, zaberaný priestor (t.j. najmä hĺbka monitora) a aj hmotnosť monitora.

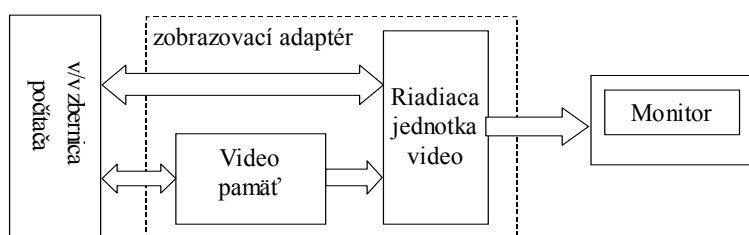


Obr. 10 Typy monitorov: (zľava) klasický CRT, plazmový a LCD

V nasledujúcej kapitole si uvedieme niektoré základné grafické vlastnosti počítačov triedy PC, ktoré dnes predstavujú najdostupnejší typ počítača, najmä z hľadiska možnosti zobrazenia t.j. z hľadiska používaných zobrazovacích adaptérov.

2.4 Zobrazovacie adaptéry

Základný princíp umiestnenia zobrazovacieho adaptéra v počítači ukazuje nasledujúci obrázok.



Obr. 11 Organizácia umiestnenia zobrazovacieho adaptéra v počítači

Z hľadiska ohodnotenia grafických zobrazovacích adaptérov a teda ich možností sa z pohľadu počítačovej grafiky sledujú najmä tieto parametre:

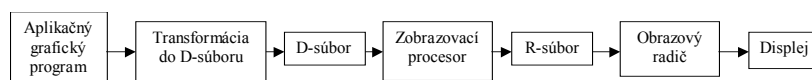
- ❑ *Rozlišovacia schopnosť*: definuje koľko obrazových bodov (pixelov) vodorovne (V) a koľko bodov zvislo (Z) je schopný adaptér maximálne zobrazit' a udáva sa ako $V \times Z$. Ak sa jedná o textový režim, potom sa neuvádza v pixeloch ale v znakoch.
- ❑ *Farebná hĺbka* (počet naraz zobraziteľných farieb)
- ❑ *Kapacita obrazovej pamäti* (dnes sa uvádza v MB a je prvým orientačným číslom, pretože od nej závisia predchádzajúce dva parametre. Čím vyššie rozlíšenie tým klesá farebná hĺbka a naopak).

2.4.1 Grafické procesory



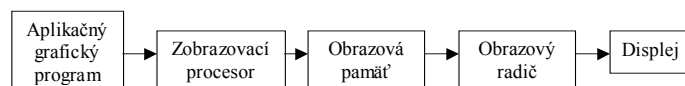
Grafické, alebo zobrazovacie procesory sú súčasťou riadiacej jednotky videa a vykonávajú všetky operácie, potrebné pre riadenie displeja. Najjednoduchší typ zobrazovacieho procesora iba mapuje časť operačnej pamäti, vyhradenej pre zobrazovanie (tzv. videopamäť, obrazovej pamäti) a túto informáciu prenáša na zobrazovaciu jednotku. Tie zložitejšie už majú aj vlastnú pamäť a umožňujú zobrazovať jednoduché objekty samostatne (úsečku, kruh atď.). Súčasný adaptér pracujú ako s dvojrozmernými tak s trojrozmernými objektami, riešia viditeľnosť, tieňovanie a realistické vykresľovanie v reálnom čase, príp. umožňujú v reálnom čase dekomprimáciu obrazu a tým umožňujú vo vysokej kvalite prehrávanie aj filmov.

Medzi procesormi vektorových a rastrových displejov je malý rozdiel. Diagram činnosti vektorového displeja je na nasledujúcom obrázku. Grafické príkazy aplikačného programu sa prevádzajú do podoby tzv. D-súboru (*D-file - Display file program*), ktorý je vstupom pre zobrazovací procesor. D-súbor obsahuje vlastne postupnosť grafických inštrukcií pre kresbu jednotlivých čiar. V niektorých systémoch sa používa ešte jeden, tzv. R-súbor (*R-file - Refresh display file*), ktorý je prístupný obrazovému radiču. Jeho význam spočíva v tom, že pomocou neho možno obnovovať iba časti obrazovky, čím sa rýchlosť zobrazovania niekoľkonásobne zvýši.



Obr. 12 Diagram vektorového zobrazovacieho systému

Rozdiel medzi rastrovými a vektorovými systémami spočíva v tom, že pamäť pre obnovovanie obrazu rastrových systémov neobsahuje postupnosť grafických inštrukcií, ale veľkosť jasu, či farbu pre každý bod (pixel) na obrazovke. Táto pamäť sa nazýva obrazová pamäť. Diagram činnosti rastrového zobrazovacieho procesora je na nasledujúcom obrázku. Zobrazovací procesor vlastne prevádza postupnosť grafických inštrukcií na maticu bodov (tzv. bodový rozklad). Rozmer matice sa pohybuje spolu s kvalitou zobrazovacieho procesora od 320 x 200 bodov (CGA), cez 640 x 480 bodov, až po 1024 x 1280 bodov a viac.



Obr. 13 Diagram rastrového zobrazovacieho systému

Rozhodujúcim kritériom pri výbere grafického procesora môže byť tiež počet farieb maximálne zobrazených naraz na obrazovke. Ten je tiež veľmi premenlivý a závisí na kapacite obrazovej pamäti. Počet farieb sa najčastejšie pohybuje od 4, cez 16, 256 a 65536 po 16,7 mil.

2.4.2 Základné zobrazovacie adaptéry PC

MDA (Monochrome Display Adapter) bol dodávaný k prvým PéCéčkam. Neumožňuje spracovanie grafickej informácie. Umožňuje zobraziť 80x25 znakov. Font (znaková sada) je uložený v pamäti ROM a v prípade snahy o implementáciu národného prostredia je nutné vymeniť túto pamäť. Dnes sa tento adaptér už nevyrába ani nedodáva.

CGA (Color Graphics Adapter) je prvou grafickou kartou dodávanou k PC. Ako už názov napovedá, umožňuje už prácu s grafikou. Grafické informácie spracováva vo dvoch režimoch:

- ❑ 320x200 (režim 4) obrazových bodov v štyroch farbách a
- ❑ 640x200 (režim 6) bodov čierno-bielo.

Pre prácu v textovom režime ponúka 80 alebo 40 stĺpcov, 25 riadkov a 16 farieb. Keďže bol prvý, podporujú ho skoro všetky programy. Font, obdobne ako u MDA, je ešte uložený v pamäti ROM a pri implementácii národných prostredí, je nutné túto pamäť vymeniť za novú. V grafickom režime sa však nepoužíva ROM generátor znakov, ale ich generuje BIOS. Preto BIOS umožňuje v grafickom režime definovanie znakov s kódom 128-255 používateľom. Po grafickej stránke CGA karta disponuje obrazovou pamäťou s kapacitou 16KB. Táto pamäť je do pamäťového priestoru PC mapovaná od adresy B8000h.

HGC (Hercules Graphics Card) nie je z vývojových dielní IBM a tak IBM tento adaptér nikdy príliš nerešpektovala. V textovom móde je prakticky zhodný s MDA, umožňuje však navyše monochromatický grafický režim s rozlišovacou schopnosťou 720x348, čo nie je málo. Okrem iného bol tento adaptér veľmi lacný a montoval sa do lacných PéCéčok a stal sa pomerne obľúbený. Prakticky v grafickom režime odpovedá každému obrazovému prvku (pixelu) jeden bit. Potom v jednom bajte je uložených osem obrazových bodov. Uloženie je mierne atypické, do štyroch oblastí. Po grafickej stránke HGC karta potrebuje teda obrazovú pamäť s kapacitou 32KB. Táto pamäť je do pamäťového priestoru PC mapovaná od adresy B0000h. Do obrazovej pamäti je možné priamo zapisovať resp. čítať. Oženie je nasledovné:

EGA (Enhanced Graphics Adapter) a **VGA** (Video Graphics Array) ako vyššie typy adaptérov si popíšeme podrobnejšie.

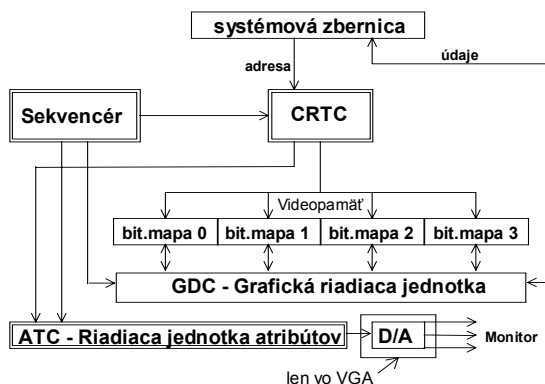
2.4.3 EGA/VGA zobrazovacie adaptéry

Popisu týchto dvoch typov adaptérov sa budeme venovať hlbšie. Táto kapitola by nemala nahrádzať komplexnejšiu literatúru o tejto problematike, avšak aspoň poukáže na to, čo by bolo vhodné vedieť o týchto adaptéroch (najmä VGA) aspoň na základnej úrovni. Popis adaptérov EGA a súčasného VGA sa bude prelínať nakoľko sú tieto adaptéry konštrukčne príbuzné a tým pádom majú množstvo rovnakých prvkov a príbuzné ovládanie. Je síce pravda, že EGA sa vlastne už nepoužíva a trh je takmer monopolne ovládaný VGA a jej "odrodami", pre výklad problematiky je však tento spôsob vhodný. EGA (následne aj VGA) adaptér má tieto hlavné časti:

Poznámky:

- ❑ RWM video pamäť - definovanie obrazu obrazovky
- ❑ ROM BIOS pamäť - základné programové vybavenie adap.
- ❑ HW registre - riadia činnosť adaptéru

Blokovú schému adaptéra EGA(VGA) ukazuje nasledujúci obrázok.



Obr. 14 Bloková schéma adaptéra EGA (VGA)

EGA (Enhanced Graphics Adapter). Tento adaptér bolo to, čo sa už dlho čakalo, vlastne už od CGA. Mal už na danú dobu dobrú rozlišovaciu schopnosť: 640x350 bodov. Umožňuje zobrazenie 16-tich farieb z palety 64-och možných farieb. Samozrejme, že sa dá prepnúť aj do grafických režimov CGA. V textovom móde umožňuje podobnú prácu ako CGA, ale navyše je možné zapnúť ešte mód so 43 riadkami. Tento adaptér, na rozdiel od predchádzajúcich typov, má tabuľku znakov jednak v pamäti ROM, ale je umožnené umiestniť font aj do k tomu určenej pamäti RAM a používať ho. Tak je možné ľahko implementovať národné prostredie len pomocou programového vybavenia.

- **Video pamäť** je časť pamäte RWM procesora, v ktorej sú uložené údaje definujúce obraz na obrazovke. V prípade EGA adaptéra je táto pamäť fyzicky priamo na doske adaptéra. Obecne môže mať EGA adaptér pamäť RWM kapacity 64, 128, 192, 256 KB. Vo všetkých prípadoch je táto pamäť organizovaná do 4 paralelných pamäťových máp (*memory maps*), každá mapa má rovnakú kapacitu. V prípade EGA RAM 256KB budeme mať k dispozícii 4 mapy po 64KB. Tieto 4 pamäťové mapy sa prostredníctvom nastavenia vnút. registrov adaptéra premietajú rôzne do logického adresného priestoru procesora. V grafických EGA módoch sa mapy premietajú zhodne do jedného adresného priestoru. To znamená, že 4 rôzne bajty na rovnakých pozíciách v pamäťových mapách majú rovnakú adresu zo strany adresnej zbernice procesora. Jednotlivé bity tohto logického bajtu sú vždy priemetom 4 bitov z jednotlivých máp a každý definuje jeden bod na obrazovke v rozsahu 16 farieb. Prístup k týmto štvoriciam bitov umožňuje zvláštna adresácia ktorú riadia HW registre EGA adaptéru. Pamäťové mapy sa tu nazývajú bitové roviny (bitové mapy) 0.až 3. Ak má EGA celkovo 64KB video pamäti, je 16KB v každej pamäťovej mape málo na to, aby definovali obrazovku v rozložení 640x350 bodov. Pamäťové mapy sú potom zret'azené dve a dve do seba a vytvárajú pamäťové roviny 0.a 1. Potom môže bod nadobúdať len 4 farby. V textových módoch sú ako video pamäť použité iba pamäťové mapy 0.a 1. Sú zret'azené do normálneho logického pamäťového priestoru a sú prístupné ako normálna pamäť. Mapy 2. a 3. nie sú prístupné zo strany procesora. Mapa 2. je využitá ako definičná tabuľka tvaru znakov pre HW generátor adaptéru, mapa 3. je nevyužitá.

Videopamäť sa môže premietat' do troch rôznych logických adresných priestorov: Ak je videopamäť mapovaná napr. do A segmentu, logický adresný priestor B segmentu neexistuje. V prípade, že videopamäť postačuje na definíciu viac ako jednej obrazovky, je videopamäť EGA BIOSom logicky rozdelená na video stránky (video pages), z ktorých každá obsahuje definíciu jednej obrazovky, pričom sa zobrazuje len aktuálna.

- **EGA (VGA) BIOS** v podobe pamäti ROM je umiestnený priamo na obvodovej doske video EGA (VGA) adaptéru, logicky tvorí časť C segmentu hlavného adresného priestoru procesora. Programovo je prístupný prostredníctvom vektoru prerušenia int 10h, číslo požadovanej služby BIOSu sa špecifikuje v registri ah a v ostatných registroch sa odovzdávajú príslušné parametre. BIOS je uložený v ROM pamäti, ale pre potrebu vnútorných premenných BIOSu je vyhradená časť pamäti RWM na adresách 0:400h tzv. BIOS Data Area, jej časť vyhradená pre video služby Video Display Data Area. Tam si BIOS udržiava informácie o stave video adaptéra.
- **HW Registre EGA (VGA)** sú dôležitou súčasťou adaptéra. Celú činnosť adaptéra riadi viac než 60 vnútorných registrov. Väčšina z nich je združená podľa svojich funkcií do 4 základných skupín predstavovaných radičmi:

CRTC - Riadiaci obvod zobrazovača

- ❑ vytvára synchro impulzy pre monitor, signál kurzora, podčiarkovanie znakov
- ❑ vytvára adresu pre videopamäť (4 bitové mapy), s čím súvisí jej refresh (dynamické pamäte)

Sekvencér

1. generuje riadiace signály pre videopamäť
2. generuje zákl. taktovací signál pre zobrazovanie
3. koordinuje prístup k videopamäti pre CRTC a CPU

ATC - Riadiaca jednotka atribútov

- ❑ mapovanie 16-tich naraz zobraziteľných farieb na možné farby monitora (EGA monitor až 64, analog VGA až TRUECOLOR)

GDC - Grafická riadiaca jednotka

- ❑ manipuluje s grafickými dátami na ceste od videopamäte k riadiacej jednotke atribútov (mask, rotácia, logické operácie)

EGA (VGA) adaptér je možné prepnúť do rôznych režimov v závislosti od toho či chceme zobrazovať textovú alebo grafickú informáciu a potom ešte následne meniť režimy podľa požadovaného rozlíšenia a počtu farieb. Je možné nastaviť niektoré štandardné režimy a priamym prístupom na registre sa takisto dá nastaviť aj niektoré neštandardné rozlíšenia príp. štruktúry pamätí. Tabuľku štandardných režimov nájdete v popise adaptéra VGA.

VGA (Video Graphics Array) nastúpil s počítačmi série IBM PS/2. Bloková schéma VGA je v princípe podobná schéme EGA. Rozdiel spočíva v rozšírenej funkcii grafickej riadiacej jednotky (niektoré VGA obsahujú v nej i grafický procesor) a riadiacej jednotky atribútov (VGA dokáže vďaka pripojenému analógovému monitoru zobraziť viac farieb na obrazovke). Prístupnosť rozširujúcich funkcií je zabezpečená pridaním registrov resp. rozšírením ich funkcií oproti EGA. Inak je programovanie VGA na úrovni registrov prakticky rovnaké ako u EGA. Zlepšenie voči EGA:

- ❑ nové 3 grafické módy s vysokou rozlišovacou schopnosťou (11 -13h)
- ❑ rozšírené funkcie grafickej riadiacej jednotky
- ❑ rozšírené funkcie riadiacej jednotky atribútov
- ❑ väčší počet HW registrov a ich funkcií

Základná paleta farieb je až 2^{18} odtieňov. Je možné povedať, že VGA sa stala súčasným štandardom. V dnešnej dobe došlo samozrejme k inováciám tohto adaptéru a to najmä zväčšením rozlišovacej schopnosti a počtu zobraziteľných farieb (2^{24} truecolor). Klasicky je už umožnené rozlíšenie 800x600 bodov a štandardom je SVGA (Super VGA) s rozlíšením 1024x768 bodov.

Pre prácu a nastavenie režimov platí to isté, čo sme napísali o módoch EGA adaptéra. VGA adaptér má však jednak módov viac a väčšiu variabilnosť v definovaní iných módov príp. iných prác s adaptérom. Prehľad základných módov prináša tabuľka Tab. 1. Tak ako sme aj napísali skôr, je rozmach VGA adaptérov a ich klonov mimoriadne búrlivý a je zrejme, že štandardné módy nepostačujú. Mnoho výrobcov sa snažilo (a aj snaží) zaviesť módy nové, lepšie. V princípe však pomaly čo grafický VGA adaptér to možnosť nájsť nové iné módy s iným číslom. Samozrejme každý kompatibilný adaptér musí podporovať základné režimy. V snahe zosúladiť aspoň niekoľko novších režimov zoskupenie VESA doporučilo niekoľko nových módov.

č. módu	rozlíšenie	farieb	typ	adr. pamäti	poznámka
0	40x25zn.	16	text	B800h	pre VGA sú 0 a 1 rovnaké
1	40x25zn.	16	text	B800h	
2	80x25zn.	16	text	B800h	pre VGA sú 2 a 3 rovnaké
3	80x25zn.	16	text	B800h	
4	320x200bod.	4	graf	B800h	pre VGA sú 4 a 5 rovnaké
5	320x200bod.	4	graf	B800h	
6	640x200bod.	2	graf	B800h	
7	80x25zn.	2	text	B800h	
Dh	320x200bod.	16	graf	A000h	
Eh	640x200bod.	16	graf	A000h	
Fh	640x350bod.	2	graf	A000h	
10h	640x350bod.	16	graf	A000h	EGA64KB len 4 farby
11h	640x480bod.	2	graf	A000h	
12h	640x480bod.	16	graf	A000h	
13h	320x200bod.	256	graf	A000h	

Tab. 1 Režimy adaptéra VGA

- **Riadiaca jednotka atribútov adaptéra VGA** je oproti EGA rozšírená. Jej zásadné rozšírenie je možný výber z 262144 možných odtieňov farieb pri analógovom monitore. Princíp spočíva v inom bloku farieb, ktorý tvorí 256 18-bitových registrov farieb. Obsah registra sa reprezentuje ako 3x6 bitov, pre každú základnú farbu jedna šestica. Táto 6-bitová hodnota sa posiela do D/A prevodníka (adaptér obsahuje 3 takéto D/A (digitálno/analógové) prevodníky). Súčasné adaptéry majú tieto registre väčšinou 8-bitové a tým sa paleta zobraziteľných farieb dostala na hodnotu 16.7 mil. (truecolor) tzv. pravých farieb. Tieto registre možno naplniť službami VGA BIOSu (int 10h napr. služba 0Ch - nakresli bod alebo služba 10h - nastaví obsah registrov palety).

2.4.4 Krátko záverom k zobrazovacím adaptérom

V súčasnosti na báze VGA sú štandardom adaptéry s grafickým procesorom resp. s grafickým urýchľovačom (*akceleratorom*). Zo začiatku tu bol veľmi populárny adaptér na báze 8514/A od IBM. Potom boli veľmi známe adaptéry *TIGA* (Texas Instruments Graphics Adapter) alebo adaptéry s procesorom *ET4000* firmy Tseng. Je možné povedať, že s nástupom Pentii sa stali „kultovými“ grafickými adaptérmí adaptéry s čipom série *S3*. Vrcholné adaptéry používajú aj 128 bitové grafické procesory.

Súčasný grafické adaptéry pracujú s týmito základnými rozlíšeniami: 640x480, 800x600 a 1024x768 pixelov. Niektoré modely majú ďalšie vyššie módy ako 1280x1024, či 1600x1200 pixelov a pod.

Počet zobraziteľných farieb je od 16 cez 256, 2^{16} (*Highcolor*) do 2^{24} (16,7 mil.) resp. až 2^{32} . Adaptérom, ktoré dokážu pracovať s takou paletou farieb hovoríme, že pracujú v móde *Truecolor* (tzv. pravé farby). Výrazne tým samozrejme stúpa nutná kapacita obrazovej pamäti na 8MB a viac. Bežné adaptéry majú dnes aspoň 32 MB obrazovej pamäti. Súčasne s rozlišovacou schopnosťou stúpa aj obnovovacia frekvencia adaptérov a monitorov od 60Hz cez 72Hz a 85Hz až po 160Hz a viac (u špičkových modelov), čo je významné najmä z hľadiska namáhania zraku používateľa. Vo výraznej miere sú súčasné grafické adaptéry osadzované aj konektorom pre priamy výstup na televízny prijímač (TV-OUT), pomocou ktorého je možné prepojiť výstup PC na domáce TV a pohodlne pozeráť napr. filmy v digitálnej kvalite v prostredí svojej obývacej izby.

Výrazný rozvoj grafických akceleratorov nastal až v posledných rokoch a to najmä v súvislosti s nástupom trojrozmerných (3D) hier v reálnom čase, s rozvojom multimédií a virtuálnej reality, so zavedením rôznych grafických štandardov a zvýšením nárokov na grafické modelovacie schopnosti súčasných PC. Z hľadiska počítačovej grafiky sa tu jedná najmä o podporu kvalitnej grafickej knižnice *OpenGL*. Z pohľadu hier a Windows9x/Me/NT/2000/XP zase o podporu systému *DIRECTX*. Grafické akcelerátory sú osadzované priamo na dosku grafického adaptéra. Medzi najznámejšie akcelerátory patria akcelerátory firiem *nVidia* (*GeForce*), *ATI* (*Radeon*), *SiS* (*Xabre*), *S3/VIA* (*Savage*), a *Matrox* (*Millenium*).

Významným faktorom najmä pri použití počítačovej grafiky je aj priepustnosť zbernice počítača, pretože pri prenášaní grafických údajov, ktorých je veľmi veľké množstvo, by bolo vhodné používať čo najrýchlejšiu zbernicu. Grafické zobrazovacie karty u PC síce mohli byť rýchle, avšak stále ich obmedzovala pomerne nízka rýchlosť zbernice (ISA). Práve rapidný prechod na grafické aplikácie (v podstate zavedenie *Windows*) v súčasnosti si vyžiadala potrebu omnoho rýchlejšej komunikácie s grafickou kartou. Preto sa objavili rýchle tzv. *lokálne zbernice*. Najmä pre grafiku bola vyvinutá zbernica označovaná **AGP** (*Advanced Graphics Port*), ktorá oproti klasickým zberniciam (PCI) umožňuje ešte rýchlejšie a ešte kvalitnejšie riadenie grafického zobrazovacieho adaptéra a v súčasnosti sú ňou vo forme jedného konektorového slotu vybavované všetky základné procesorové dosky. Vyrábajú sa aj základné procesorové dosky s integrovaným zobrazovacím adaptérom a zdieľanou videopamäťou. Tieto sú určené najmä do lacnejších zostáv osobných počítačov.



Obr. 15 Grafický adaptér (rozhranie AGP)

2.5 Ostatné grafické zariadenia

Okrem už spomínaných vstupno/výstupných zariadení je možné k osobnému počítaču pripojiť celú škálu rôznych iných špecializovaných zariadení slúžiacich na grafický vstup ale výstup najmä dnes nastupujúce zariadenia ako *digitálne fotoaparáty a kamery* a pod. U týchto je najzaujímavejšou informáciou hustota snímacieho rastra, ktorá sa udáva v megapixloch. Ako digitálna kamera, tak digitálny fotoaparát sú vybavené kontrolným LCD displejom, na ktorom je možné okamžite po zábere resp. po nasnímaní skontrolovať výsledok a napr. ak sa záber nevydaril, ho vymazať a prípadne urobiť ďalší. Ako digitálna kamera, tak digitálny fotoaparát sú spravidla vybavené pamäťou RWM (RAM), ktorej kapacita je určujúca pre počet snímok resp. minút záznamu. Tieto zariadenia najmä digitálne kamery majú ešte veľkokapacitné záznamové médium najčastejšie vo forme magnetickej pásky, pevného disku alebo iného záznamového média. Obe zariadenia sú vybavované ešte minimálne sériovým rozhraním (RS232 alebo USB) pre pripojenie k počítaču. Niektoré typy digitálnych fotoaparátov umožňujú priamo pripojiť aj malé špecializované tlačiarňičky alebo modemy. Potom je možné fotografiu aj okamžite vytlačiť, poslať faxom alebo elektronickou poštou.



Obr. 16 Digitálny fotoaparát



1. Na základe akých kritérií je možné klasifikovať grafické zariadenia počítačov
2. Charakterizujte vektorové zariadenia
3. Charakterizujte rastrové zariadenia
4. Vymenujte a v krátkosti popíšte základné vstupné grafické zariadenia
5. Vymenujte a v krátkosti popíšte základné výstupné grafické zariadenia
6. Charakterizujte módy zobrazovacích monitorov
7. Charakterizujte na základe čoho sa ohodnocujú grafické zobrazovacie adaptéry
8. Vymenujte a v krátkosti popíšte základné zobrazovacie adaptéry PC

V tejto kapitole sme sa naučili:



- ☐ Grafické periférie rozdeľujeme podľa rôznych hľadísk, najmä podľa smeru komunikácie s používateľom, podľa spôsobu vytvárania grafickej informácie, podľa schopnosti spracúvať grafické informácie, podľa dĺžky uchovania grafickej informácie a podľa možnosti samostatného spracovania grafickej informácie.
- ☐ Najviac používaná kategorizácia grafických zariadení je podľa smeru komunikácie na vstupné a výstupné a podľa spôsobu vytvárania grafickej informácie na vektorové a rastrové.
- ☐ Medzi základné vstupné grafické periférie patria: myš, trackball, trackpad, joystick, tablet, scanner a optické pero.
- ☐ Medzi základné výstupné grafické periférie patria: tlačiareň, súradnicový zapisovač, monitor spolu so zobrazovacím adaptérom.
- ☐ DPI je skratka z Dot Per Inch t.j. Bodov na Palec a definuje rozlišovaciu schopnosť resp. presnosť alebo citlivosť zariadenia. Čím vyššie DPI, tým kvalitnejšie zariadenie.
- ☐ Podľa typu tlače sa najviac používajú maticové (ihličkové) tlačiarne, tryskové (atramentové) tlačiarne a laserové resp. LED tlačiarne.
- ☐ Monitory používajú rôzne zobrazovače. Medzi najviac používané zobrazovače patria: katódové obrazovky (CRT), plazmové zobrazovače, elektroluminiscenčné zobrazovače a zobrazovače z tekutých kryštálov (LCD).
- ☐ Monitor pracuje v dvoch režimoch: textový a grafický. Monitor je k počítaču pripojený pomocou zobrazovacieho adaptéra.
- ☐ Z hľadiska ohodnotenia grafických zobrazovacích adaptérov sa sledujú najmä tieto parametre: rozlišovacia schopnosť (definuje koľko obrazových bodov vodorovne a koľko zvislo je schopný adaptér zobrazit'), farebná hĺbka (počet naraz zobraziteľných farieb) a kapacita obrazovej pamäti.
- ☐ Režim, pri ktorom sa naraz môže zobrazit' 2^{16} farieb sa volá Highcolor a režim, pri ktorom sa môže naraz zobrazit' 2^{24} farieb sa nazýva Truecolor.
- ☐ Základné zobrazovacie adaptéry PC sú: MDA, CGA, HGC, EGA, VGA a SVGA.
- ☐ EGA/VGA adaptér má tieto hlavné časti: videopamäť, EGA(VGA) BIOS, registre EGA(VGA), ktoré riadia CRTC (Riadiaci obvod zobrazovača), sekvencér, ATC (Riadiaca jednotka atribútov) a GDC (Grafická riadiaca jednotka). U VGA je navyše číslicovo-analógový prevodník (RAM DAC), ktorého rýchlosť prevodu výrazne ovplyvňuje rýchlosť adaptéra.
- ☐ VGA adaptér má 15 základných režimov práce.
- ☐ Najmä pre prácu s trojrozmernou grafickou informáciou sa používajú grafické adaptéry s grafickým procesorom resp. s grafickým akceleračným procesorom. Súčasné adaptéry kvôli rýchlosti sa pripájajú k počítaču pomocou rýchlych lokálnych zberníc, najčastejšie PCI a AGP.



3 Grafické systémy

Cieľ: Preštudovaním tejto kapitoly by mal študent získať základné vedomosti o súčasných grafických štandardoch a systémoch najmä o ich základných vlastnostiach. Väčšia časť je venovaná v súčasnosti najviac preferovanému štandardu OpenGL. Ďalej bude študent oboznámený so základnými grafickými elementami, s ktorými súčasná počítačová grafika pracuje. Niektoré z uvádzaných elementov (bod, sled bodov, úsečka, kružnica, elipsa) by mal byť schopný detailnejšie charakterizovať. Navyše po preštudovaní tejto kapitoly by mal študent získať prvé teoretické základy pre programovú implementáciu uvádzaných elementov.

3.1 Grafické štandardy

Po pionierskych začiatkoch v odbore počítačovej grafiky a po preukázaní vysokej "životaschopnosti" jej produktov sa začali aj viacerí komerční výrobcovia zaoberať vývojom a výrobou grafických zariadení. Automaticky sa ukázali pomerne vysoké problémy s prenositeľnosťou t.j. kompatibilitou ako technických tak najmä programových produktov a údajov. Vtedy sa začali objavovať prvé tendencie ku štandardizácii a zavádzaniu tzv. abstraktných prvkov. Ako štandardná sa zaviedla koncepcia idealizovaného grafického pracoviska s jeho aplikáciou na rôzne technické platformy. Aplikácia potom komunikuje s technickým okolím (napr. grafickou stanicou) pomocou abstraktných vstupných resp. výstupných operácií, tieto operácie sú štandardizované a používateľ nemusí poznať špeciálne vlastnosti toho-ktorého zariadenia¹. Z hľadiska implementácie týchto systémov sú volené väčšinou knižničné funkcie (napr. pre jazyk C) alebo rozšírená funkčná sada resp. častejšie sada podprogramov (subroutine napr. pre FORTRAN).

V základe štandardizované grafické systémy pracujú s už spomínanými grafickými prvkami (entitami), každý prvok má svoje atribúty a štruktúrované (či už segmentové alebo hierarchické) ukladanie prvkov a ich atribútov vytvára výsledný obraz. Toto boli uvedené najmä výstupné charakteristiky. Pre interaktívnu prácu sú definované aj abstraktné vstupné zariadenia V týchto systémoch sú definované abstraktné vstupné prvky napr.:

- **locator** (lokátor): toto zariadenie určuje vstupné súradnice x, y bodu na zobrazovacej ploche. V súčasnosti sa objavuje aj lokátor3D.
- **valuator**: toto zariadenie slúži na zadanie číselnej hodnoty. Často môže byť spojená napr. s určitým typom posuvnej lišty v *GUI* (Graphic User Interface – grafické používateľské rozhranie) a pod.
- **choice** (select, výber, možnosť): slúži na výber z alternatívnych možností.
- **stroke** (polygon, spojenie): slúži na určenie postupnosti napr. súradníc resp. iných definičných bodov.
- **pick** (určenie): určenie určitého objektu z niekoľkých.

¹

Napr. aj platforma MS Windows sa snaží byť nezávislá od zariadenia. Typicky sa používa štruktúra Device Context, kde sa „popíše“ zariadenie a potom sa používajú už len štandardné funkcie pre manipuláciu s ním (napr. prístup na grafický adaptér).

Následne je potom pri implementácii možné využívať tieto prvky. Väčšina súčasných systémov používajúcich napr. *GUI*² je riadená udalosťami (napr. aj MS Windows). Z hľadiska grafických štandardov sa podľa pôvodcu udalosti (aplikácia alebo používateľ) definujú tri spôsoby prijímania údajov:

- **event** (udalosť) - práca používateľa a aplikácie je relatívne asynchrónna. Používateľ generuje vstupy (napr. pohybuje myšou), tie sú ukladané do schránok a aplikácia ich odtiaľ vyzdvihne v prípade potreby.
- **sample** (vzorkovanie) - práca používateľa a aplikácie je asynchrónna. Aplikácia len vzorkuje stav vstupného zariadenia, na reakciu používateľa sa však nečaká.
- **request** (žiadost') - práca používateľa a aplikácie je synchrónna. Aplikácia v prípade potreby žiada vstupný údaj a čaká na používateľov vstup.

Z hľadiska grafických štandardov si uvedieme štyri najzákladnejšie

3.1.1 GKS (Graphics Kernel System)

Tento štandard bol prijatý aj u nás. K jeho normalizácii na úrovni ISO (International Standard Organization) bolo v roku 1985. Základom určenia tohto štandardu sú 2D grafické aplikácie. Jeho jadrom je špecifikácia, na základe ktorej sú realizované jednotlivé implementácie na rôznych platformách. Teda nejde o programový systém ako taký. Tento štandard je komplexná grafická norma a pre jednoduchšie aplikácie sa častokrát implementujú len podmnožiny celého štandardu (definovaných je 9 úrovní podmnožín). V základe GKS používa všetky základné grafické elementy (mimo polygónov) a segmentovanie obrázkov (nie hierarchickú štruktúru). GKS používa normalizovanú súradnicovú sústavu s rozsahom súradníc x, y z intervalu $\langle 0, 1 \rangle$. Umožňuje samozrejme mapovanie súradnicového systému aplikácie na súradnicový systém príslušného zobrazovacieho zariadenia. Toto mapovanie je dvojúrovňové. Najprv sa vykonajú *normalizačné transformácie*. V praxi to znamená definovanie transformácie zo súradnicového systému aplikácie do normalizovaného súradnicového systému. Táto transformácia je teda transparentná od zobrazovacieho zariadenia a uplatňujúca sa ako pre celý obrazec, tak aj pre segment obrázku alebo primitívu. Po tejto úrovni nasleduje druhá úroveň. V nej sa vykoná *transformácia pracoviska*. Prakticky to je vlastne transformácia z normalizovanej súradnicovej sústavy do súradnicovej sústavy grafického pracoviska (zariadenia). Pri zobrazovaní sa môže použiť celá zobrazovacia plocha alebo len časť tzv. okno. Pomocou systému viacerých okien a ich prekryvu a aktivovania je možné na jednej ploche zobraziť viac úloh.

3.1.2 GKS-3D (3D Extension of GKS)

Rozvoj 3D grafických aplikácií priniesol aj nutnosť štandardizovať 3D grafické elementy a postupy. Aj keď sa zaoberáme najmä 2D systémami, z hľadiska vývoja si uvedieme aj túto normu. Logicky sa pristúpilo k rozšíreniu normy GKS. GKS-3D je kompatibilný s GKS a ako rozšírenie bol navrhnutý v roku 1987. Rozširujúcimi prvkami boli:

² Štandard, ktorý sa týka normalizácie rozhraní medzi grafického systému nezávislými od zariadenia (device independent) a časťami závislými na použítom zariadení, je *CGI* (Computer Graphics Interface). Tento štandard vznikol v roku 1988 a je stále upravovaný.

- ❑ pridanie tretej súradnice z (t.j. trojrozmerné výstupné primitívy,
- ❑ transformácie premietania z 3D do 2D (rovnobežné a stredové premietanie) a
- ❑ techniky riešenia viditeľnosti resp. odstránenia neviditeľných častí.

GKS-3D sa chová ako GKS, ak je tretia súradnica nulová. Podobne ako v GKS sa tu používajú trojrozmerné normalizačné transformácie, transformácie segmentov a transformácie pracoviska. Navyše sú tu potom transformácie určujúce orientáciu premietania a priestorové orezanie. Každý 3D element má priradený tzv. *view index* (atribút indexu pohľadu). Tento atribút určuje transformáciu premietania, ktorá sa bude na element aplikovať. Rôzne elementy môžu mať zdieľané tie isté indexy pohľadu.

Transformácie premietania sú určené pomocou matic 4x4. GKS-3D obsahuje aj pomocné programy umožňujúce používateľovi (alebo aplikácii) vygenerovať tieto matice. Na ich vygenerovanie sa definuje poloha kamery, smer pohľadu a poloha priemetne.

Z hľadiska techník riešenia viditeľnosti môže obsahovať primitíva aj tzv. HLHSR (Hidden-Line or Hidden-Surface Removal) identifikátor. Tento určuje spôsob aplikácie techniky odstránenia zakrytých častí v priemete.

3.1.3 PHIGS (Programmer's Hierarchical Interaction Graphics System)

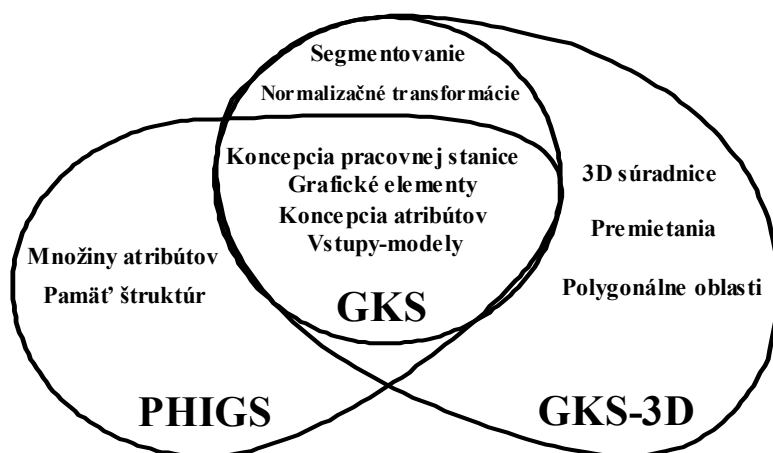
Tento štandard bol vytvorený súčasne so štandardom GKS-3D v roku 1987. Snahou tohto štandardu je pokryť tie oblasti aplikácie v 3D, ktoré majú dynamický charakter ako modelovanie a simulácia napr. v oblasti CAD/CAM.

PHIGS určuje sadu funkcií pre programovanie ako v 2D tak v 3D počítačovej grafike. Zobrazenie obrazu po aplikácii transformácie je chápané ako ďalší krok (GKS zobrazuje automaticky). Vytvorený objekt je teda nie ihneď zobrazený, ale je len uložený do tzv. CSS (centralizovanej štrukturovanej databázy). Táto štruktúra môže byť hierarchická. Jej hlavnou jednotkou je *prvok štruktúry* (sem patria aj elementy ako lomená čiara, polygon apod. príp. atribúty ako napr. typ čiary). Tieto prvky sú v databáze združované, čím reprezentujú hierarchickú štruktúru obrázku vrátane atribútov. Nad databázou sú vytvorené dva špeciálne prvky štruktúry - *riadiaci prvok štruktúry* a *editačný prvok štruktúry*. Tieto umožňujú aj po vytvorení databázy editovať jej prvky, rušiť ich, nahradzovať príp. pridávať ďalšie prvky. V rámci CSS je umožnené aby jeden prvok bolo možné využívať z viacerých nadradených prvkov (tzv. acyklické uzly, kde jeden uzol môže mať viac rodičov). Prístup ku prvkom štruktúry je pomocou smerníkov. CSS je možné vybudovať aj bez okamžitej znalosti skutočných elementov a pomocou editácie až potom vytvoriť reálne obsahy elementov.

PHIGS umožňuje vytvárať pre rôzne obrázky alebo ich časti rôzne súradnicové sústavy. Zavádza sa tzv. modelovacia súradná sústava, ktorá umožňuje kombinovať časti obrázku do seba, do jednotnej súradnicovej sústavy aplikácie.

Zobrazenie je vykonávané zaslaním na pracovnú stanicu. Súčasťou zaslania je prechod štruktúry CSS databázy, jej interpretovanie a postupné vykreslenie všetkých elementov. Zaslanie je vykonávané kontinuálne a zmeny v štruktúre CSS sa prejavujú okamžite na výstupe.

Jednotlivé prvky (črty) základných grafických štandardov a ich vzájomné vzťahy ukazuje nasledujúci obrázok.



Obr. 17 Črty a vzťahy medzi základnými grafickými štandardmi

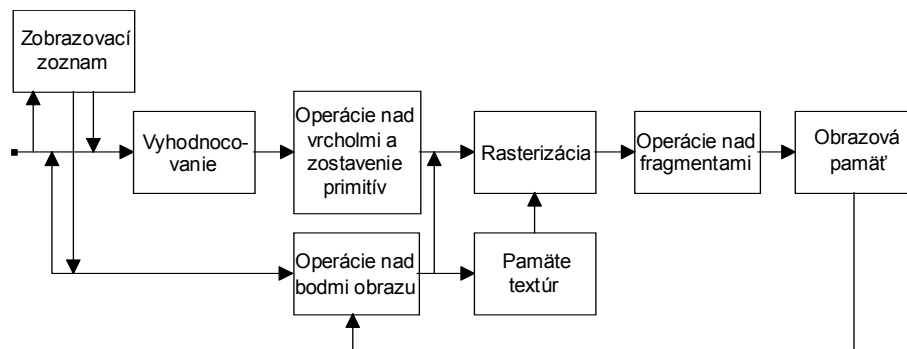
3.1.4 OpenGL (Open Graphics Library)

OpenGL nebol od začiatku vyvíjaný ako štandard a preto si ho uvedieme až po predchádzajúcich štandardoch a v tomto ponímaní ho nebudeme chápať ako základný štandard aj keď je dnes veľmi používaný. OpenGL používala firma Silicon Graphics (ešte nie pod názvom OpenGL) ako grafickú knižnicu na vývoj svojich CAD aplikácií, neskôr ju modifikovala a uvoľnila pod týmto oficiálnym názvom. Ako už bolo povedané jedná sa vlastne o grafickú knižnicu, ktorá je dnes už priemyselným štandardom najmä v oblasti vytvárania 3D scén na tej najvšeobecnejšej úrovni. V súčasnosti sa čoraz viac výrobcov najmä grafických kariet pýši už tým, že integrujú na ne akcelerátory podporujúce OpenGL. To zo sebou prináša nemalé zrýchlenie programov využívajúcich OpenGL. Ako výrobcovia hardvéru, tak aj veľké softvérové firmy (napr. Microsoft vo WindowsNT či IBM v OS/2) už implementovali podporu OpenGL priamo do jadra svojich operačných systémov.

OpenGL je vlastne skratka od *Open Graphics Library*. Dnes je táto knižnica portovaná na platforme PC, ale aj a najmä na výkonné pracovné stanice zamerané na spracovanie grafiky. Samotný systém OpenGL tvorí niekoľko stovák funkcií, ktoré umožňujú programátorovi špecifikovať objekty a operácie realizované nad nimi, pričom hlavné ťažisko je v spracovaní 3D grafiky a výsledkom je vysokokvalitné zobrazenie 3D scény, ktorého kvalita je obmedzená len možnosťami použitého hardvéru.

Ako už bolo povedané pre programátora sa OpenGL javí ako množina volaní funkcií (v princípe podobne napr. ako GKS). Začína volaním funkcií umožňujúcich otvorenie okna do obrazovej pamäte, do ktorej bude orientovaný výstup OpenGL. Ďalšie volania zabezpečia vytvorenie GL kontextu a jeho previazanie s týmto oknom. Potom, čo je GL kontext alokovaný, programátor môže v plnej miere využívať služby OpenGL. Určité volania sa používajú na kreslenie jednoduchých geometrických objektov (body, čiary a pod.), iné definujú atribúty prostredia (druh a spôsob osvetlenia, farbu, spôsob mapovania 3D scény na 2D zobrazovací priestor obrazovky). Z formálneho hľadiska je teda možné OpenGL definovať ako určitý stavový stroj, vykonávajúci určitú množinu špecifických operácií týkajúcich sa vykreslenia objektov.

Model interpretujúci činnosť OpenGL je klient-server. Klientom je program, ktorý volaním funkcií žiada o služby server, ktorým je vlastne samotná OpenGL. Efekty volaní funkcií OpenGL sú dané možnosťami toho-ktorého grafického systému. OpenGL je konštruovaný na čo najväčšiu platformovú nezávislosť. Preto sa pri istých volaniach môžu prejavovať na rôznych zariadeniach rôzne spôsoby vykreslenia a podobne.



Obr. 18 Základné operácie v systéme OpenGL

Predchádzajúci obrázok schématicky zobrazuje prácu systému OpenGL. Tento diagram je na najvyššej úrovni abstrakcie a teda nemusí presne odrážať konkrétnu realizáciu OpenGL. Príkazy do OpenGL vchádzajú zľava. Niektoré príkazy špecifikujú geometrické objekty, ktoré sa majú vykresliť, kým iné riadia spracovanie objektov do rôznych stavov. Väčšina príkazov sa môže akumulovať v zobrazovacom zozname, kde čakajú na neskoršie spracovanie. Inak sú príkazy spracovávané zretiazene. Vyhodnocovanie vykonáva aproximáciu kriviek ploch výkonným spracovaním došlých vstupných hodnôt. Ďalší blok slúži na spracovanie geometrických primitív popísaných vrcholmi. V tejto fáze sú vrcholy transformované, osvetlené a primitívy sú transformované do zobrazovacieho priestoru. Následne rasterizér produkuje sériu adries obrazovej pamäte a hodnôt na nich, využívajúc dvojrozmerný opis bodu, úsečky či mnohoúhelníka. Každý takýto fragment postupuje do ďalšej etapy, kde sa nad ním vykonávajú ďalšie operácie pred vlastným namapovaním do obrazovej pamäti. Toto v sebe zahŕňa najmä podmienené zmeny založené na hodnotách hĺbky, miešanie prichádzajúcich farieb fragmentu s už uloženými farbami, takisto ako aj maskovanie a vykonávanie iných logických operácií nad hodnotami fragmentu. Potom sa hodnoty bodov zapisujú do obrazovej pamäti. Hodnoty v obrazovej pamäti môžu byť taktiež presúvané z miesta na miesto a môžu byť tiež vyňaté z obrazovej pamäti. Takéto usporiadanie je viacmenej logickou štruktúrou systému OpenGL a konkrétna implementácia sa od toho môže do istej miery odlišovať.

Ako bolo v predchádzajúcom odseku povedané, do mechanizmu procesu vykonávania v systéme OpenGL vstupuje niekoľko typov údajov, medzi ktoré patria:

- ❑ vrcholy
- ❑ mnohoúhelníky, špecificky pravouholníky definované týmito vrcholmi
- ❑ aktuálna pozícia v rasti (lokácia)
- ❑ aktuálna normála (normálový vektor) definovaného vrcholu
- ❑ aktuálna farba definovaného vrcholu
- ❑ aktuálne súradnice textúry

Z hľadiska spracovania fragmentov po rasterizéri sa vykonávajú aj rôzne testy a to najmä:

- ❑ test na vyrezanie oblasti
- ❑ alfatest
- ❑ test na značkovanie
- ❑ test pamäte hĺbky

Okrem týchto testov pristupujú na tejto úrovni ďalšie dôležité akcie:

- ❑ miešanie (blending)
- ❑ dithering
- ❑ logické operácie

Výsledkom týchto etáp je spomínané naplnenie obrazovej pamäti bodmi (pixlami), ktoré sú vlastne konvertovanými fragmentami. Samotná obrazová pamäť je organizovaná do množiny logických pamätí. Túto množinu tvoria tieto prvky:

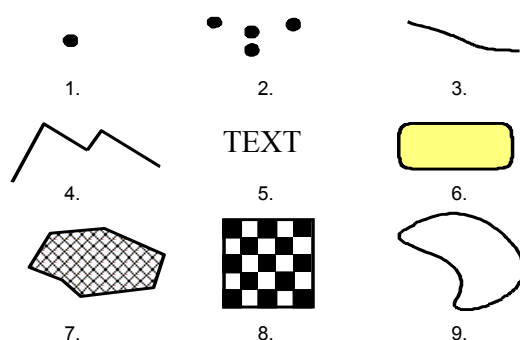
- ❑ pamäť farby (color buffer)
- ❑ pamäť hĺbky (depth buffer)
- ❑ pamäť značkovania (stencil buffer)
- ❑ akumulčné pamäte (accumulation buffer)

Pamäť farby sama ešte pozostáva z niekoľkých ďalších zložiek zameraných na pravú prednú, pravú zadnú, ľavú prednú a ľavú zadnú časť scény. Okrem toho obsahuje ešte ďalšie pomocné zložky. Nad samotnou obrazovou pamäťou sa dajú robiť operácie výberu konkrétnej vyrovnávacej pamäti a čítania z nej.

3.2 Základné grafické prvky

Každý grafický systém pracuje s určitými množinami nástrojov, pomocou ktorých sa dosahuje výsledný grafický efekt. Základom (a to aj pri všetkých dosiaľ definovaných grafických štandardoch) je *množina grafických elementov* (prvkov, primitív). Množina týchto elementov môže byť v rôznych systémoch definovaná rôzna. Je možné však nájsť niektoré prvky vyskytujúce sa najčastejšie:

1. *bod*
2. *zoznam* (sled) *bodov* (polymarker)
3. *krivka* (špeciálne priamka, častokrát aj elipsa stojí samostatne)
4. *lomená čiara* (polyline)
5. *text* (t.j. grafický text)
6. *plocha* (špeciálne rovina)
7. *vyplnená oblasť* (fill area)
8. *výplňový vzor* definovaný aj ako *pole bodov* (pixel array)
9. *všeobecný grafický prvok* (generalised drawing primitive)



Obr. 19 Množina základných grafických elementov

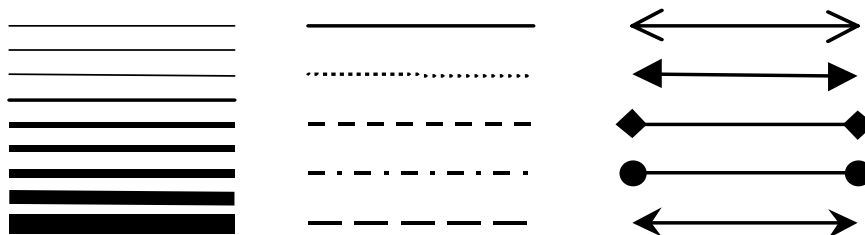
Poznámky:

Okrem toho konečný tvar týchto primitív je možné riadiť ich *atribútmi*. Medzi základné atribúty z pohľadu počítačovej grafiky zaradujeme:

- ❑ *farba*
- ❑ *typ* (napr. čiar, písma a pod.)
- ❑ *hrúbka* (napr. čiar, písma a pod.)
- ❑ *poloha* (napr. písma)
- ❑ *smer vykreslenia* (napr. horizontálny, vertikálny atď.)

Atribúty môžu byť jednotlivým elementom priradené:

- ❑ *konvenčne* alebo tiež *individuálne*, pevne, čo vedie niekedy k nekompatibilitate na rôznych zobrazovačoch.
- ❑ *symbolicky*, najčastejšie formou kódu. Vtedy hovoríme o viazaných (bundled) atribútoch. Tieto sú vzhľadom na zobrazovač transparentné.

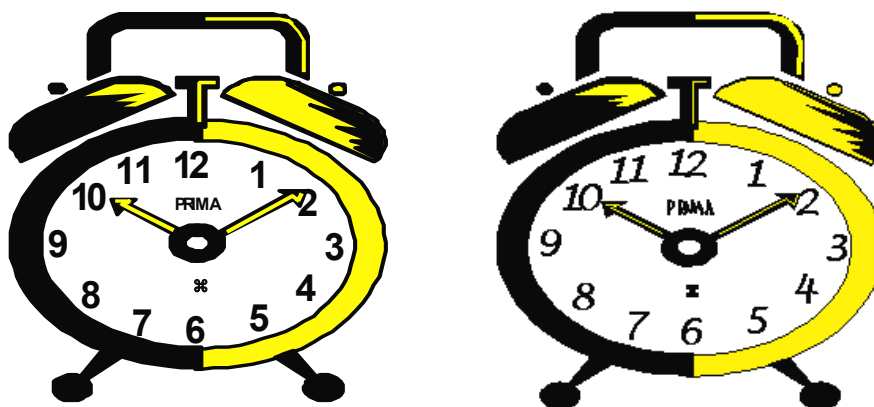


Obr. 20 Príklady rôznych atribútov čiar ako hrúbka, typ a spôsob ukončenia

Tak ako sme si rozdelili v/v grafické zariadenia podľa typu, tak sú delené v základe aj grafické objekty. Potom budeme teda rozoznávať grafické objekty:

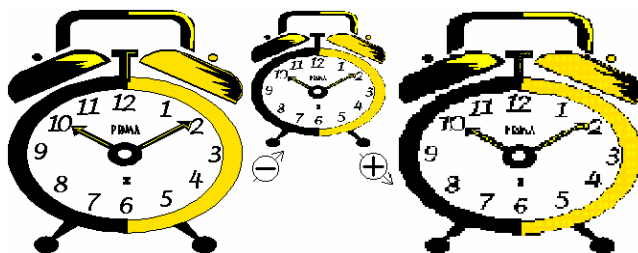
- ❑ *vektorové* a
- ❑ *rastrové*.

Podľa toho s akými dominantnými objektmi pracuje grafický systém, aj ten bude vektorový alebo rastrový. V prípade, že systém umožňuje pracovať aj s vektorovými aj s rastrovými typmi, potom budeme hovoriť o kombinovanom grafickom systéme. Ak systém v princípe pracuje z pohľadu základných grafických elementov *len s bodom*, potom sa jedná o *rastrový typ*. Ak systém pracuje pri vnútornej reprezentácii *aj s inými elementmi* ako s bodom napr. *s krivkami* potom sa jedná o *vektorový typ*. Potom vektorový obrazec sa skladá z množiny týchto elementov, ich súradníc príp. iných atribútov a akákoľvek transformácia tohto obrazca sa vykonáva nad týmito elementmi resp. ich atribútmi. Na nasledujúcom obrázku je možné vidieť rozdiel medzi vektorovým a rastrovým objektom.



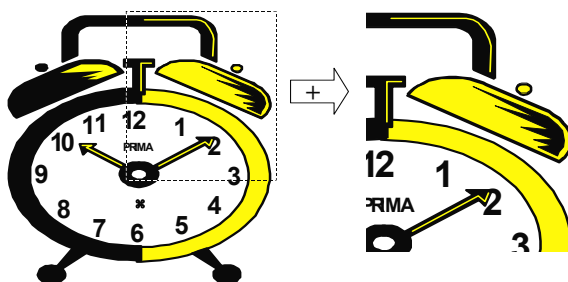
Obr. 21 Vektorový a rastrový objekt

Jednou z nevýhod rastrovej reprezentácie je deformácia grafickej informácie resp. dokonca jej strata napr. pri zmenšení a opätovnom zväčšení, ako ukazuje nasledujúci obrázok.

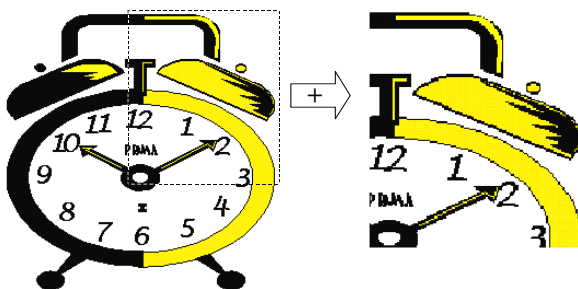


Obr. 22 Skreslenie a strata pri zmenšení a opätovnom zväčšení rastrového objektu

Ďalším rozdielom medzi vektorovým a rastrovým objektom, ktorý je možné porovnať na nasledujúcich dvoch obrázkoch, je to, že pri výraznom zväčšení si vektorový objekt zachováva svoju kvalitu a tvar. Rastrový objekt pri výraznom zväčšení, síce nestráca svoju pôvodnú informáciu, ako tomu je pri zmenšení, ale stráca „hladkosť“ a tým pádom svoju kvalitu. Niektoré z týchto chýb, je možné u rastrových objektov eliminovať pomocou rozptyľovania a vyhladzovania.



Obr. 23 Zväčšenie časti vektorového objektu



Obr. 24 Zväčšenie časti rastrového objektu

Aj tieto rozdiely naznačujú domény použitia jednotlivých typov grafických systémov. Vektorové typy sú výhodné pre technickejšie kreslenie napr. na kreslenie vývojových diagramov, technických výkresov či obchodných grafov. Rastrové typy dominujú pri spracovaní obrazu, digitálnych fotografií a pri voľnej grafickej tvorbe. V tomto zmysle je nutné podotknúť, že tu abstrahujeme od typov nástrojov, ktorými disponuje to-ktoré programové vybavenie. Program môže byť rastrový, ale môže disponovať vektorými typmi nástrojov. V úvode práce s nástrojov sa s ním pracuje ako s vektorovým typom, napr. nakreslenie úsečky, ale po uložení, už to predstavuje len jednotlivé body rastra a pristupovať k nim ako k objektu – úsečka a tým meniť

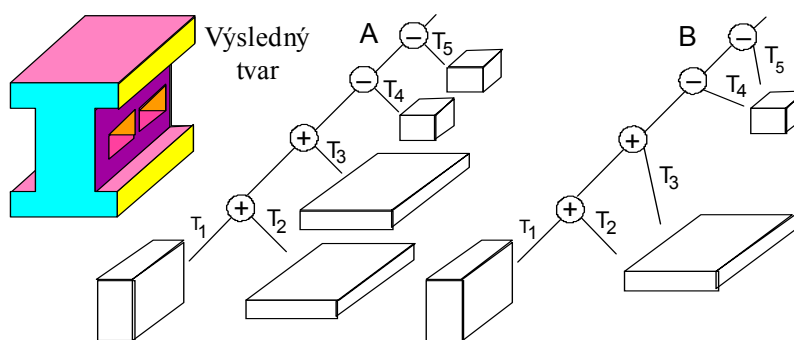
atribúty tohto objektu, už nie je možné. Po „položení“ úsečky je to už len množina bodov. Typickým príkladom je napr. známy grafický editor *Paint* (Skicár, Malovanie) v MS Windows 95/98.

Ďalším dôležitým faktorom, ktorý dominuje pri spracovaní objektov v počítačovej grafike, je ich rozmer. V základe sa v počítačovej grafike pracuje s dvojrozmernými (2D – 2 Dimensional) a s trojrozmernými (3D – 3 Dimensional) objektami. V prípade animácie alebo napr. určitých častí systémov virtuálnej reality sú objekty aj štvorrozmerné, kde štvrtým rozmerom je čas. V ďalšom texte sa budeme dominantne venovať dvoj- alebo trojrozmerným objektom.

Spočiatku bolo možné resp. sa na počítačoch pracovalo len s 2D grafikou. Postupom času sa prešlo na 3D grafiku. Možnosti 3D systémov sú rozmanitejšie ako u 2D systémov. 3D systémy je možné v podstate podľa geometrie modelovania, kvality zobrazenia a ďalších možností spracovania rozdeliť na:

- systémy založené na drôtovom modeli (*Wire Frame Model*)
- systémy založené na povrchovom modeli (*Surface Model*)
- systémy založené na objemovom modeli (*Solid Model*)

Systémy na báze drôtového modelu boli logickým rozšírením 2D systémov a začali sa používať ako prvé. Druhým bol povrchový model, avšak až objemový model stojí na najvyššom stupni. Na rozdiel od svojich predchodcov je jediný model, ktorý jednoznačne definuje teleso. Hlavným dôvodom nasadzovania systémov s objemovým modelovaním je potreba jednotnej prezentácie priestorových objektov. Takéto systémy sú ale pomerne rýchlostne aj pamäťovo náročné. Systémy pre objemové modelovanie v podstate umožňujú používať jednotnú údajovú štruktúru pre prezentáciu modelovaných objektov, ponúkajú súbor elementárnych telies (primitív ako kváder, guľa, ihlan, kužel, valec a pod.), umožňujú vykonávať množinové operácie nad telesami ako zjednotenie a prienik, poskytujú bežné geometrické transformácie (posunutie, otočenie, orezanie, zmena mierky a pod.), umožňujú vypočítať niektoré integrálne charakteristiky telies (objem, ťažisko, zotrvačný moment a pod.) a v neposlednej miere napr. aj axonometrické resp. perspektívne premietanie.



Obr. 25 Model telesa v reprezentácii CSG A-jednoduché, B-s instanciami

Z hľadiska počítačovej grafiky a geometrického modelovania zvlášť je zaujímavá najmä údajová štruktúra. Možnosť popisu telies je rôzna. Výhodnou je možnosť popisu telesa stromom (CSG - *Constructive Solid Geometry* – Konštruktívna geometria telies). Uzliami stromu sú buď základné elementárne telesá (primitívy) alebo už hotové knižničné telesá tiež na báze CSG, alebo množinové operácie nad telesami. Aby bolo možné použiť určité detaily na viacerých miestach modelu

zavádzajú tzv. instance (podmodely). Potom sa teda odkazuje na rovnaký model, ale líši sa to určitou geometrickou transformáciou, ktorá definuje umiestnenie instance v priestore (pozri Obr. 25).

Výhodou tejto reprezentácie sú malé pamäťové nároky a možnosť jednoduchých modifikácií modelov. Nevýhodou tejto metódy je, že nedáva takmer žiadne informácie o telese. Kedykoľvek je potrebné mať povrch telesa k dispozícii, musí sa CSG strom vyhodnotiť a vypočítať. K tomu je určený algoritmus. Tento je poväčšine asi najzložitejšou a najpomalejšou stránkou systému. Okrem CSG reprezentácie sa používa aj **hraničná reprezentácia** (*B-rep* - *Boundary Representation*). Táto má vyššie pamäťové nároky. Popis telesa na základe vzťahov medzi vrcholmi, hranami a stenami.

V nasledujúcich podkapitolách nájdete popis základných grafických prvkov z pohľadu ich implementácie v počítačovej grafike najmä z pohľadu rastrových systémov. Popis zložitejších prvkov je predmetom potom ďalších samostatných kapitol, ktoré sa nachádzajú v tejto publikácii.

3.2.1 Bod

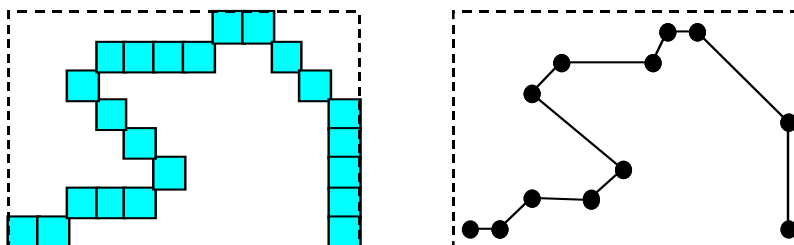
Bod je elementárnym objektom v počítačovej grafike vôbec. Základnými atribútmi bodu z pohľadu počítačovej grafiky, sú jeho poloha a farba príp. čas. V počítačovej grafike sú používané „viaceré typy“ bodov. Základným používaným bodom je *pixel*. Jedná sa o obrazový bod, ktorý je charakterizovaný dvomi súradnicami polohy a svojou farbou. Ďalším „typom“ bodu je *voxel*, tiež nazývaný aj objemový bod. V počítačovej grafike sa používa pri objemovom modelovaní objektov. Jeho základnými atribútmi sú tri súradnice polohy a farba. Pri vyplňovaní a textúrovaní sa používa bod výplne resp. textúry (pozri kapitolu o vyplňovaní) nazývaný *texel*, čo je skratka z *texture pixel*. Tento okrem polohy v rámci určitej súradnicovej sústavy má ešte definovanú aj polohu v rámci výplňového vzoru (textúry) a reláciu svojho priradenia vyplňanej oblasti. Tejto relácii hovoríme aj mapovanie výplňového vzoru (textúry). Posledným zo zaujímavých „typov“ bodov je *tixel*, čo je skratka z *time pixel*. Tento „typ“ bodu má navyše aj časový atribút a používa sa napr. v animačných grafických systémoch. S bodom, ako v základe s jediným a najvyšším typom grafického prvku pracujú už spomínané rastrové grafické systémy resp. zariadenia.

3.2.2 Sled bodov (polymarker)

Je rozširujúcim prvkom a priamo nadväzuje na objekt bod. Polymarker definuje logicky zviazanú množinu bodov na základe určitej *relácie* medzi prislúchajúcimi atribútmi týchto bodov. Pri akejkoľvek operácii (napr. presúvaní) sa táto operácia deje nad všetkými bodmi spolu. Relácia medzi atribútmi bodov polymarkra môže byť definovaná *homogénne* alebo *heterogénne*. Homogénna relácia je relácia definovaná medzi rovnakými atribútmi jednotlivých bodov polymarkra, napr. medzi atribútmi polohy. Heterogénna relácia je definovaná medzi rôznymi atribútmi jednotlivých bodov v polymarkra, napr. farba niektorého bodu je závislá od polohy druhého.

Polymarker sa používa napr. v grafických editoroch na operáciu kreslenia voľnou rukou (angl. *freehand*). V *rastrových editoroch* sa táto operácia vykoná tak, že na

okamžitú pozíciu kurzora sa vykreslí bod so svojimi atribútmi (pozri Obr. 26 vľavo). Jediné, čo je potrebné dodržať je obmedzenie rýchlosti posunu kurzora po kreslajúcej ploche, lebo inak by mohlo dôjsť k opticky nekontinuálnej kresbe. Po uložení bodov je možné k nim pristupovať aj ako k polymarkeru aj ako k samostatným bodom, čo je vyslovene záležitosť tvorcu editora. Vo *vektorových editoroch sa implementuje mierne iným spôsobom*. Pohyb kurzora sa vzorkuje a kreslenie voľnou rukou nie je implementované bodmi, ale najčastejšie sa vykonáva lineárna interpolácia medzi jednotlivými bodmi (pozri Obr. 26 vpravo).



Obr. 26 Ukážky implementácie kreslenia voľnou rukou v rastrovom aj vektorovom editore

3.2.3 Úsečka

Je definovaná ako najkratšia spojnica dvoch bodov. Je veľmi často používaný objekt v počítačovej grafike. Pre zobrazenie na rastrovom displeji je potrebné určiť, ktoré body budú rozsvietené a ktoré nie. Umiestnenie týchto bodov sa určí na základe rovnice priamky, ktorej časťou je daná úsečka. Výsledkom je množina bodov, ktorá sa svojim tvarom čo najviac približuje danej úsečke (tzv. alias úsečky). Kostnatosť výslednej čiary na displeji, ktorá je zapríčinená diskretnosťou bodov sa dá eliminovať použitím displeja s vyšším rozlíšením prípadne metódami na vyhladzovanie čiar (antialiasing).

Úsečka vo všeobecnosti môže byť zadaná:

$$y = k \cdot x + c \quad (1)$$

kde: y - súradnica bodu na osi y
 x - súradnica bodu na osi x
 k - smernica priamky, na ktorej leží úsečka
 c - posun na osi y

Ak máme dva body $A[x_A, y_A]$ a $B[x_B, y_B]$, ktoré sú koncovými bodmi úsečky, potom základné koeficienty vypočítame nasledovne:

$$k = \frac{y_B - y_A}{x_B - x_A} \quad \text{a} \quad c = \frac{(x_B y_A - x_A y_B)}{(x_B - x_A)} \quad (2)$$

Kresliť úsečku v počítačovej grafike môžeme viacerými algoritмами:

- **Algoritmus založený na výpočte oboch súradníc** je založený na tom, že z počiatočného bodu kreslíme čiaru po výpočte nasledujúcej dvojice súradníc bodu úsečky podľa smernice. Tento spôsob je jednoduchý, výpočtovo však veľmi náročný, čo sa prejaví malou rýchlosťou kreslenia úsečky. Z toho dôvodu sa v praxi takmer nepoužíva.
- **Algoritmus DDA** alebo tiež prírastkový algoritmus (*DDA - digital differential analyzer*) je založený na postupnom pripočítavaní konštantných prírastkov k

obom súradniciam x a y . Rozlišujeme výpočet pre priamku so smernicou k menšou ako 1 a so smernicou k väčšou ako 1. Vychádzajúc zo vzťahu (2) sa naskôr musia vyrátať diferencie (rozdiely) medzi jednotlivými súradnicami, čím sa určí smernica k . To, či je $k > 1$ alebo $k < 1$, určujú vlastne veľkosti diferencií jednotlivých súradníc. Počet krokov nám definuje potom väčšia diferenciacia. Prírastky v jednotlivých osiach (px , py) sa potom definujú ako podiely diferencií jednotlivých súradníc a počtu krokov. Toto je možné zapísať nasledovne:

$$\begin{aligned} dx &= |x_B - x_A| \\ dy &= |y_B - y_A| \end{aligned} \Rightarrow \text{pocet_krokov} = \max(dx, dy) \Rightarrow \begin{aligned} px &= \frac{dx}{\text{pocet_krokov}} \\ py &= \frac{dy}{\text{pocet_krokov}} \end{aligned} \quad (3)$$

Pre smernicu menšiu ako 1 budeme meniť súradnicu na osi x o jednotku, pretože $px=1$ (táto os sa potom nazýva riadiaca os) a výpočtom určíme súradnicu na osi y .


$$y_{i+1} = y_i + py \quad (4)$$

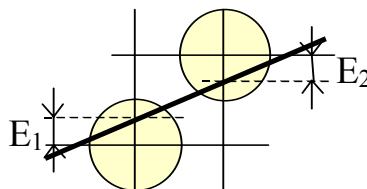
Pre smernicu väčšiu ako 1 je postup analogický, ale riadiaca os bude os y .

$$x_{i+1} = x_i + px \quad (5)$$

Toto pripočítavanie sa deje potom v cykle pre celkový počet krokov.

DDA algoritmus je citlivý na presnosť výpočtu a pri dlhších úsečkách môže dôjsť ku kumulácii zaokrúhľovacej chyby pri definovaní prírastkov.

 **Bresenhamov algoritmus pre kreslenie úsečky** je to veľmi efektívny algoritmus generovania bodov na úsečke. Nachádza body ležiace najbližšie danej skutočnej úsečke na základe hodnoty predikčného chybového člena E_D (Error diferencial). Rovnako ako DDA sa uvažuje s úsečkou, ktorej smernica je menšia ako 1 alebo naopak väčšia ako 1. Takisto sa vyberie súradnica s väčším prírastkom t.j. s väčšou diferenciou súradníc a tým vyberie aj riadiacu os. V smere riadiacej osi sa potom opäť bude súradnice inkrementovať o 1. V smere druhej súradnice sa určuje daný predikčný chybový člen E_D a podľa neho sa volí bližší bod (pixel) k úsečke. Predikčný chybový člen tak zachytáva vzdialenosť medzi ideálnou (skutočnou) úsečkou a zobrazovaným bodom. Táto vzdialenosť je meraná kolmo na smer rýchlejšieho rozvoja (príklad pre y -ovú os pozri Obr. 27). Ak je táto chyba menšia ako polovica jednotky rastra volí inkrementácia len v smere riadiacej osi. Ak je chyba väčšia ako polovica jednotky rastra potom sa volí inkrementácia v oboch smeroch. Nech je napríklad riadiaca os x . Potom ak je $E_D < \text{polovica jednotky rastra}$, potom nová súradnica je $(x+1, y)$, inak je nová súradnica $(x+1, y+1)$. Ak sa chybový člen vhodne inicializuje, stačí postaviť algoritmus len na testovaní znamienka chybového člena a potom celý algoritmus upraviť na tvar, v ktorom sa používa len celočíselná aritmetika. Navyše algoritmus používa len operáciu sčítania a násobenie dvomi. Bresenhamov algoritmus, oproti DDA algoritmu, vo výpočtovom cykle obsahuje test, ktorý zaisťuje doladovanie interpolácie s ohľadom na spomínanú vzdialenosť pixla od skutočnej úsečky. Tým je zabezpečená maximálna chyba polovicu jednotky rastra.

Obr. 27 Stanovenie chybového člena E_D pri Bresenhamovom algoritme

Z hľadiska implementácie potom je nutné rozlišovať spomínaný prípad, či je smernica väčšia alebo menšia ako 1. Ako príklad budeme riešiť prípad so smernicou menšou ako 1, pre prípad smernice väčšej ako 1, bude sa tento riešiť analogicky ale s druhou riadiacou osou. Najprv sa teda nakreslí prvý bod so súradnicami x, y a stanoví sa počítačová hodnota predikcie chyby E_D . Pri kreslení nasledujúceho bodu bude súradnica x zvýšená o jednotku a súradnica y môže byť rovnaká alebo zvýšená o jednotku. To sa teda rozhodne na základe vzdialenosti skutočnej úsečky k bodom rastra definovanej spomínaným chybovým členom E_D . Najskôr sa vyrátajú diferencie oboch súradníc:

$$\begin{aligned} dx &= |x_B - x_A| \\ dy &= |y_B - y_A| \end{aligned} \quad (6)$$

Potom vypočítame predikčnú chybu E a podľa nej budeme určovať nasledujúcu polohu súradnice y .

$$\begin{aligned} E_1 &= 2 \cdot dx - dy && \text{pre prvý bod} \\ E_i < 0 &\Rightarrow E_{i+1} = E_i + 2 \cdot dy ; y_{i+1} = y_i \\ E_i \geq 0 &\Rightarrow E_{i+1} = E_i + 2 \cdot dy - 2 \cdot dx ; y_{i+1} = y_i + 1 \end{aligned} \quad (7)$$

Toto sa deje opäť v cykle pre celkový počet krokov stanovený analogicky ako pri DDA algoritme. Samozrejme nesmieme zabudnúť v každom kroku czklu inkrementovať súradnicu v smere riadiacej osi o 1 (v tomto prípade x).

3.2.4 Sled úsečiek (polyline)

Je obdobou zoskupenia úsečiek ako u polymarkera. Navyše však môžeme hovoriť o otvorených alebo uzavretých sledoch. Existujú používané špeciálne prípady sledov úsečiek ako trojuholník, štvorec, obdĺžnik, lichobežník, rovnobežník, šesťuholník, osemuholník a podobne.

3.2.5 Kružnica

Je definovaná ako množina bodov rovnako vzdialených od stredového bodu. Kružnica je najčastejšie definovaná pomocou stredového bodu $S[x_s, y_s]$ a polomeru r (prípadne bodu, ktorý na nej leží). Rovnicu kružnice môžeme vyjadriť:

$$y = y_s \pm \sqrt{r^2 - (x - x_s)^2} \quad (8)$$

Algoritmov kreslenia kružnice je niekoľko. V nasledujúcom si uvedieme dva najpoužívanejšie postupy.

- **Algoritmus kreslenia kružnice na základe parametrického vyjadrenia** vykonáva kreslenie kružnice tak, že ju nahradíme kreslením mnohoúhelníka, pričom jeho stred je identický so stredom kružnice a polohy bodov určíme parametricky na základe parametra u v intervale $\langle 0, 2\pi \rangle$:

$$\begin{aligned}x &= x_s + r \cdot \cos(u) \\ y &= y_s + r \cdot \sin(u)\end{aligned}\quad (9)$$

Použijeme funkciu na vykreslení úsečky medzi jednotlivými bodmi. Tento spôsob je výpočtovo náročný pretože pracuje s funkciami reálnych čísel a nevyužíva symetrické body kružnice.

- **Algoritmus kreslenia kružnice podľa predikcie chyby** vychádza z rovnakých princípov ako kreslenie úsečky Bresenhamovým algoritmom výpočtom predikcie chyby a podľa nej určenia súradnice y nasledujúceho bodu, keď súradnica nasledujúceho bodu x je zvýšená o jednotku.

$$\text{Počiatočné body: } \begin{aligned}x_1 &= 0 \\ y_1 &= r\end{aligned}\quad (10)$$

$$\text{Poloha bodu: } \begin{aligned}x_{i+1} &= x_i + 1 \\ y_{i+1} &\text{ sa stanoví predikciou chyby } E_{i+1}\end{aligned}\quad (11)$$

Počiatočná predikcia chyby: $E_1 = 1 - r$, potom

$$\begin{aligned}E_i < 0 &\Rightarrow E_{i+1} = E_i + 2 \cdot x_i + 3 \\ E_i \geq 0 &\Rightarrow E_{i+1} = E_i + 2 \cdot x_i + 5 - 2 \cdot y_i\end{aligned}\quad (12)$$

Výpočet budeme robiť pre jednu osminu kružnice a celú kružnicu vykreslíme pomocou funkcie na základe symetrie kružnice. Výpočet sa tým zníži na minimálnu hodnotu. Tento algoritmus je vďaka použitiu len celočíselných operácií jedným z najrýchlejších algoritmov na vykreslenie kružnice.

3.2.6 Elipsa

Je definovaná ako množina bodov, ktorých súčet vzdialeností od dvoch bodov (ohnísk) je konštantný. Rovnica elipsy zadanej stredom $S[x_s, y_s]$ a poloosami a, b rovnobežnými so súradnicovými osami je:


$$\frac{(x - x_s)^2}{a^2} + \frac{(y - y_s)^2}{b^2} = 1\quad (13)$$

Vykreslenie elipsy sa používa aj na vykreslenie kružnice na zobrazovacie zariadenie, ktoré nemá pomery strán zobrazovacieho priestoru 1:1. Algoritmy kreslenia elipsy:

- **Algoritmus kreslenia elipsy na základe parametrického vyjadrenia** je jej kreslenie, kedy ju nahradíme kreslením mnohoúhelníka, pričom jeho stred je identický so stredom elipsy a polohy bodov určíme parametricky na základe parametra u v intervale $\langle 0, 2\pi \rangle$:

$$\begin{aligned}x &= x_s + a \cdot \cos(u) \\ y &= y_s + b \cdot \sin(u)\end{aligned}\quad (14)$$

Použijeme funkciu na vykreslení úsečky medzi jednotlivými bodmi. Tento spôsob je výpočtovo náročný pretože pracuje s funkciami reálných čísel a nevyužíva symetrické body elipsy.

 **Algoritmus kreslenia elipsy na základe predikcie chyby** umožňuje použiť algoritmus, ktorý umožňuje robiť výpočty polohy jednotlivých bodov v celočíselnej aritmetike. Výpočet bude podobný ako v prípade kružnice, s výnimkou niekoľkých základných rozdielov:

- ❑ pomocou symetrie určíme pre jeden vypočítaný bod 4 body na elipse
- ❑ počítame časť s riadiacou osou x aj časť s riadiacou osou y

Pre počiatočné body platí: $x_1 = 0$
 $y_1 = a$ (15)

Poloha bodu: $x_{i+1} = x_i + 1$
 y_{i+1} sa stanoví predikciou chyby E_{i+1} (16)

Počiatočná predikcia chyby: $E_1 = b^2 - b \cdot a^2 + \frac{a^2}{4}$, potom

$$\begin{aligned} E_i < 0 &\Rightarrow E_{i+1} = E_i + b^2 \cdot (2 \cdot x_i + 1) \\ E_i \geq 0 &\Rightarrow E_{i+1} = E_i + b^2 \cdot (2 \cdot x_i + 1) - 2 \cdot a^2 \cdot y_i \end{aligned} \quad (17)$$

Výpočet budeme robiť pre jednu štvrtinu elipsy a celú elipsu vykreslíme pomocou funkcie na základe symetrie elipsy. Náročnosť výpočtu sa tým zníži na minimálnu hodnotu. Tento algoritmus je vďaka použitiu len celočíselných operácií jedným z najrýchlejších algoritmov na vykreslenie elipsy.

3.2.7 Kruhový (eliptický) výsek

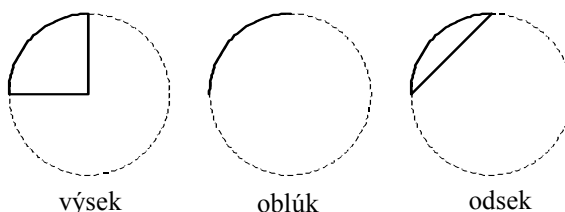


Kruhový resp. aj eliptický výsek býva častokrát samostatným prvkom v grafických systémoch. Niektoré zdroje ho uvádzajú ako jeden zo základných prvkov (oblúk, arc), iné ako prvok odvodený z kružnice resp. elipsy. V našom ponímaní napriek tomu, že sú tieto prvky uvedené explicitne, budeme ich tiež chápať ako odvodené.

Podľa toho, ako sa potom definovaný objekt vykreslí, resp. ktoré jeho riadiace body sa chápu ako významové, dostávame v princípe až tri druhy objektov:

- ❑ *výsek*
- ❑ *oblúk*
- ❑ *odsek*

Po stránke algoritmickej sa v princípe vychádza zo vzťahov (9) a (14). Až na to, že oproti kružnici resp. elipse, uhol nie je definovaný z intervalu $\langle 0, 2\pi \rangle$, ale z určitého definovaného intervalu (α_1, α_2) , čím sa vlastne definuje veľkosť uhla výseku, ako rozdiel týchto hraničných uhlov intervalu.



Obr. 28 Príklady kruhového (eliptického) výseku, oblúku a odseku

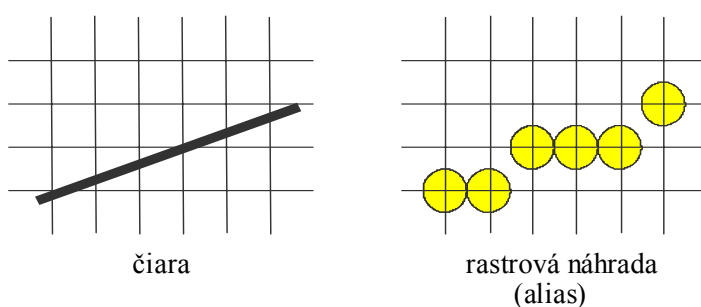
3.2.8 Antialiasing

Tento pojem úzko súvisí s kreslením úsečiek na rastrových zariadeniach. *Antialiasing* (vyhladzovanie) je metóda používaná k odstráneniu alebo zmierneniu nežiadúceho skreslenia kresby na rastrových zariadeniach. Tento jav na rastrových zariadeniach sa nazýva *alias*. Ide v podstate o nepresné vyjadrenie tvaru resp. potlačenie významných detailov čiže o stratu informácie. Spojité, hladké čiary sú na rastrových zariadeniach zobrazené schodovito. Pri kreslení v rastrovej sieti (táto je diskretná) je skutočná scéna zobrazovaná vzorkovaním všeobecne spojitých priebehov. V počítačovej grafike sa *alias* môže objaviť v troch rôznych prípadoch:

1. schodovité zobrazovanie priamych rovných čiar a hraníc polygónov na rastrových displejoch.
2. tento prípad sa objaví ak je zobrazovaný objekt menší ako veľkosť pixelu (obrazového bodu) alebo pri veľmi tenkých čiarach. Malé objekty potom nie sú vôbec zobrazené alebo napr. tenké čiary nie sú opticky hladké resp. ucelené (sú zobrazené ako nepravidelná postupnosť bodov).
3. pri zobrazení zložitejšej scény s blízkymi detailmi (napr. pri generovaní základných obrázkov z raytracingu). Tieto detaily sú buď potlačené alebo skreslené tak, že nie je možné rozoznať ich pôvodný tvar.

Tento jav sa prejavuje výrazne aj pri animačných sekvenciách alebo pri interaktívnej práci (napr. pri premiestňovaní alebo iných transformácií objektov v grafických editoroch), kde aj obyčajná úsečka vykáže nepríjemné skreslenie pri jej rotácii a keď je príliš malá, môže sa striedavo objavovať a miznúť v závislosti od polohy.

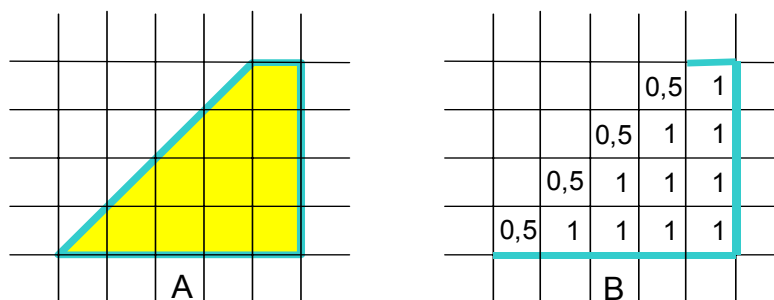
Základná idea *antialiasingu* spočíva v tom, že pixel sa nepovažuje za ideálny bod, ale pracuje sa s ním ako s plochou, ktorá je súčasťou scény. Každý objekt v scéne následne ovplyvňuje hodnotu minimálne jedného pixelu. Informáciu strácajúcu sa pri prevode do rastra prevedieme na jasovú (pri odtieňoch šedej) alebo na farebnú hodnotu pixelu. Potom intenzitu (farbu) každého pixelu určíme ako súčet (funkciu) intenzít (farieb) všetkých objektov, ktoré sú zobrazené v danom pixely. Na nasledujúcom obrázku je zobrazená úsečka a jej rasterizácia (rastrová náhrada, alias).



Obr. 29 Rastrová náhrada (alias) čiary

Keďže je čiara tenšia ako šírka pixelu, nemôže byť dokonale zobrazená napr. Bresenhamovým algoritmom. Lepšie zobrazenie dosiahneme, keď intenzitu (farbu) každého pixelu odvodíme z veľkosti plochy prvkom zakrytej (napr. pozri nasledujúci obrázok). Teda jasovú informáciu stotožníme s veľkosťou plochy vymedzenej skutočnou úsečkou v príslušnom rastrovom štvorci, kde jednotlivé stredy pixelov sú posunuté do stredov týchto štvorcov.

Poznámky:



Obr. 30 Antialiasing

Ďalšou často používanou metódou (najmä v súvislosti s tretím prípadom aliasingu) je metóda vzorkovania s vyššou frekvenciou, než je požadované rozlíšenie. Obraz je vygenerovaný do subpixelov a následne je intenzita pixelu určená ako vážený priemer intenzít susedných subpixelov (napr. ak chceme získať obraz s rozlíšením 640x480 pixelov, musíme vygenerovať obraz s rozlíšením aspoň 1280x960 subpixelov a intenzitu (farbu) každého pixelu určíme ako napr. aritmetický priemer intenzít štyroch subpixelov). Tento postup má však niekoľko nevýhod. V prvom rade sú to zvýšené nároky na čas (len dvojnásobná presnosť zvýši čas asi štvornásobne), pre objekty ešte menšie ako je veľkosť subpixelu sa problém opäť nerieši a použitie len aritmetického priemeru neodpovedá vždy skutočnosti, preto sa používajú iné vážené priemery. Častokrát sa používajú miesto antialiasingu aj niektoré filtrovacie a rozptyľovacie techniky.



1. Vymenujte a v krátkosti popíšte základné abstraktné vstupné prvky grafických systémov
2. Vymenujte a v krátkosti popíšte známe grafické štandardy.
3. Vymenujte základné grafické prvky a ich atribúty.
4. Charakterizujte 3D modelovacie systémy
5. Charakterizujte bod ako jeden zo základných grafických prvkov
6. Charakterizujte sled bodov ako jeden zo základných grafických prvkov
7. Popíšte spracovanie úsečky v rámci počítačovej grafiky a uveďte základné metódy jej generovania.
8. Vysvetlite DDA algoritmus
9. Vysvetlite Bresenhamov algoritmus.
10. Popíšte spracovanie kružnice v rámci počítačovej grafiky a uveďte základné metódy jej generovania.
11. Popíšte spracovanie elipsy v rámci počítačovej grafiky a uveďte základné metódy jej generovania.
12. Charakterizujte antialiasing

V tejto kapitole sme sa naučili:



- ☐ V grafických systémoch sú definované nasledujúce abstraktné vstupné prvky: locator (lokátor), valuátor, choice (výber), stroke (spojenie) a pick (určenie).
- ☐ Z hľadiska grafických štandardov sa podľa pôvodcu udalosti (aplikácia alebo používateľ) definujú tri spôsoby prijímania údajov: event (udalosť), sample (vzorkovanie) a request (žiadosť).
- ☐ GUI je skratka pre Graphic User Interface t.j. pre grafické používateľské rozhranie.
- ☐ Medzi najznámejšie grafické štandardy radíme: GKS (Graphics Kernel System), GKS-3D (3D Extension of GKS), PHIGS (Programmer's Hierarchical Interaction Graphics System) a OpenGL (Open Graphics Library).
- ☐ Základné grafické elementy (primitíva, prvky) sú: bod, sled bodov (polymarker), krivka, lomená čiara, grafický text, plocha, vyplnená oblasť, výplňový vzor a všeobecný grafický prvok.
- ☐ Konečný tvar elementov je možné riadiť ich atribútmi. Medzi základné atribúty zaradíme: farbu, typ, hrúbku, polohu a smer vykreslenia. Atribúty môžu byť elementom priradené individuálne (konvenčne) alebo symbolicky.
- ☐ Podľa toho s akými objektami dominantne grafický systém pracuje rozoznávame objekty vektorové a rastrové. Rastrový systém pracuje len s bodom. Vektorový systém pracuje vo vnútornej reprezentácii aj s inými elementami.
- ☐ Jednou z nevýhod rastrovej reprezentácie je deformácia grafickej informácie resp. dokonca jej strata pri zmenšení a opätovnom zväčšení.
- ☐ Vektorové typy sú výhodné pri technickom kreslení alebo pri obchodných grafoch. Rastrové typy dominujú pri spracovaní obrazov, digitálnych fotografiách a pri voľnej grafickej tvorbe.
- ☐ Trojrozmerné grafické systémy pracujú s tromi typmi modelov: drôtový model, povrchový model a objemový model.
- ☐ Z hľadiska vnútornej reprezentácie grafických modelov sa používa reprezentácia CSG (Constructive Solid Geometry – konštruktívna geometria telies) a hraničná reprezentácia (boundary representation).
- ☐ Základné typy bodov, ktoré sa v počítačovej grafike používajú sú: pixel (obrazový bod), voxel (objemový bod), texel (bod textúry) a tixel (časový bod).
- ☐ Polymarker definuje logicky zviazanú množinu bodov na základe určitej relácie medzi prislúchajúcimi atribútmi bodov. Táto relácia môže byť homogénna a heterogénna.
- ☐ Kreslenie úsečky môžeme vykonať niekoľkými spôsobmi: pomocou analytického vyjadrenia, pomocou DDA (Digitálny Diferenciálny Algoritmus) a pomocou Bresenhamovho algoritmu.
- ☐ Kružnicu a elipsu kreslíme buď pomocou parametrického vyjadrenia alebo pomocou predikcie chyby.
- ☐ Antialiasing (vyhladzovanie) je metóda používaná k odstráneniu alebo zmerneniu nežiadúceho skreslenia kresby na rastrových zariadeniach. V počítačovej grafike sa alias môže objaviť v troch prípadoch: schodovité zobrazenie priamych čiar, ak je objekt menší ako veľkosť pixelu resp. tenkých čiarach a pri zobrazení zložitejšej scény s blízkymi detailami.



4 Transformácie v počítačovej grafike

Cieľ: *Obsah tejto kapitoly patrí medzi základné znalosti pri práci s počítačovou grafikou. Z tohto dôvodu je táto kapitola rozsiahlejšia. Po absolvovaní tejto kapitoly by študent mal vedieť popísať a používať základné súradnicové sústavy používané v rámci PG. Ďalej bude vedieť rozlišovať základné zobrazovacie reťazce v rámci počítačovej grafiky a jednotlivé transformácie v nich používané. Študent bude schopný rozoznať a aplikovať základné geometrické transformácie (posunutie, zrkadlenie, zmena mierky, skosenie a otočenie) používané v počítačovej grafike na rôzne typy objektov ako v dvojrozmernom tak trojrozmernom priestore. V rámci zobrazovacieho reťazca študent by mal následne vedieť rozlíšiť a aplikovať rôzne typy lineárnych príp. nelineárnych premietacích transformácií.*

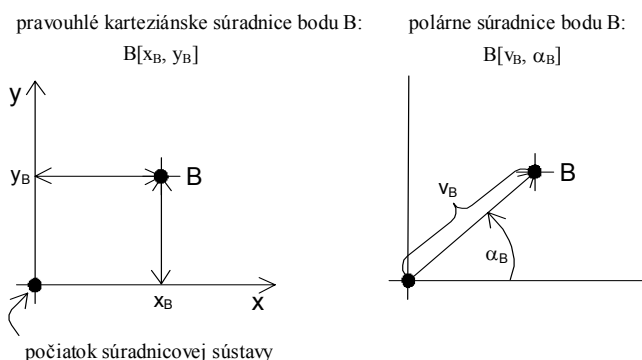
Geometrické transformácie, zobrazenia a morfovanie modelov resp. objektov je jednou z najčastejších činností systémov počítačovej grafiky. Pre geometrické transformácie sa používajú transformácie používané v počítačovej grafike, takisto v zobrazeniach sa používa najmä perspektívne zobrazenie, menej často axonometrické zobrazenie príp. možnosť viacerých typov zobrazení resp. iných zobrazení. Novším prvkom je 3D morfovanie objektov v zmysle transformácií riadiacich prvkov modelovacích prostriedkov, v zmysle simulácie bud' morfovanie objektu v časovom kontinuu (napr. starnutie, rozklad a pod.) alebo morfovania objektu pri interakcii. Najmä posledne uvedená problematika je silne závislá od použitých modelovacích prostriedkov a v používaní alebo nepoužívaní podporných hardvérových urýchľovačov. Osobitnou kapitolou je nepoužívanie 2D zobrazovačov ale snaha o použitie 3D zobrazovačov, čím problematika zobrazenia a riešenia viditeľnosti odpadá. U tohto typu zobrazovačov sa však v súčasnosti ešte jedná o pomerne robustné zariadenia s nízkymi parametrami za priveľmi vysokú cenu. To však neznamená, že ich použitie v budúcnosti nemá zmysel.

Ako definovanie či modelovanie, tak aj následné transformácie či zobrazenia objektov patria medzi základné procesy aplikované v počítačovej grafike. Tu má veľký význam aký je spracovávaný grafický objekt. Typovo sa spracovávané objekty delia na *vektorové* a *rastrové*. Podľa rozmeru spracovávanej grafickej informácie sa zase delia na: *1D* (1-dimenzional), *2D* (rovinné, plošné), *3D* (priestorové) *4D* (k trom súradniciam pribúda čas). Aby však bolo možné uvedené operácie vykonať, je nutné definovať si určité súradnicové sústavy, vzhľadom na ktoré sa uvedené operácie vykonávajú.

4.1 Súradnicové sústavy

Ako už bolo povedané, dôležitým krokom pre aplikáciu či už zobrazovacích alebo geometrických transformácií je definovanie vhodnej súradnicovej sústavy, vzhľadom na ktorú sa bude pracovať. Prvým kritériom delenia súradnicových sústav je ich *rozmer*. V počítačovej grafike sa najčastejšie používajú *2D* (dvojrozmerné) a *3D* (trojrozmerné) súradnicové sústavy. Tam, kde sa v rámci počítačovej grafiky používa aj *čas* napr. pri animáciách sa používajú štvorrozmerné súradnicové sústavy. Podľa *typu súradníc* (pozri Obr. 31), ktoré príslušná súradnicová sústava využíva, sa delia súradnicové sústavy používané v počítačovej grafike na:

- *karteziánske a*
- *polárne.*

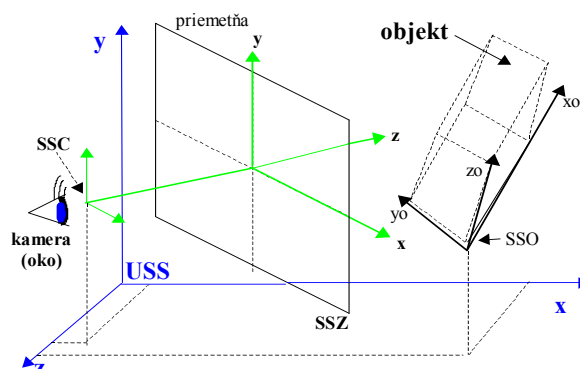


Obr. 31 Karteziánske a polárne súradnice bodu B v 2D

Aj keď z hľadiska matematických výpočtov sú tieto delenia veľmi podstatné (v ďalšom sa bude primárne pracovať s karteziánskymi súradnicami), omnoho podstatnejšie je definovanie *charakteru súradnicovej sústavy* z hľadiska logickej výstavby grafického systému. Je to z toho dôvodu, že práve s týmto charakterom vlastne operuje používateľ na primárnej úrovni. Až v sekundárnej úrovni sú súradnicovej sústave priradené *atribúty* ako napr. *rozmer* a *typ súradnic* príp. *merné jednotky* (cm, pixely a pod.). Pri nasledujúcom popise, z hľadiska charakteru, sa budú používať nasledujúce súradnicové sústavy:

1. *USS - Univerzálna (Používateľská, Globálna) Súradnicová Sústava* - predstavuje súradnicovú sústavu aplikácie, ktorú používa používateľ (pozorovateľ). Jednotky súradníc sú dané príslušnou aplikáciou (napr. palce, milimetre a pod.).
2. *SSO - Súradnicová Sústava Objektu* - predstavuje súradnicovú sústavu príslušného objektu. Každý objekt má svoju súradnicovú sústavu. Používa sa najmä u dynamických objektov. Štruktúra je obdobná ako u *USS*.
3. *NSS - Normalizovaná Súradnicová Sústava* - predstavuje súradnicovú sústavu s rozsahom unifikovaným do intervalu $\langle 0,1 \rangle$. Táto sústava môže tvoriť medzisústavu medzi *USS* a *SSZ* a je ideálna na dosiahnutie nezávislosti grafického zariadenia (*device independent*) od *USS*.
4. *SSZ - Súradnicová Sústava Zariadenia* - predstavuje súradnicovú sústavu príslušného grafického zariadenia. Takéto zariadenie má spravidla omedzený rozsah súradníc vzhľadom k svojej zobrazovacej ploche (napr. displej má spravidla počiatok v ľavom hornom rohu a napr. rozsah súradníc $x \in \langle 0,639 \rangle$, $y \in \langle 0,479 \rangle$ (u VGA)).
5. *SSC - Súradnicová Sústava kamery* - predstavuje súradnicovú sústavu virtuálnej kamery (objektívu, pozorovateľa), pomocou ktorého je snímaná 3D scéna. Oproti predchádzajúcim súradným systémom obsahuje vo svojej definícii aj vektor smeru snímania (pohľadu), ktorý je najčastejšie stotožňovaný s osou z tejto súradnicovej sústavy. V princípe sa jedná vlastne o súradnicový systém oka pozorovateľa (stred tejto súradnicovej sústavy je ukázaný na Obr. 32).
6. *SST - Súradnicová Sústava Textúry* - predstavuje súradnicovú sústavu príslušnej textúry. Každá textúra má svoju súradnicovú sústavu, ktorá je

závislá od typu, veľkosti (rozlíšenia) a farebnej hĺbky textúry. Z hľadiska typu môže byť 2D a 3D textúra, najčastejšie 2D typ. Z hľadiska rýchlosti vizualizačného jadra je vhodné voliť textúry s jednotnou veľkosťou a farebnou hĺbkou. Často sa je *SST* označovaná aj ako tzv. "uv"-priestor.



Obr. 32 Vzťahy jednotlivých súradnicových sústav pre 3D

Záverom tejto kapitoly ešte niekoľko poznámok. *SSO* je definovaná ako podsústava *USS*. Niektoré charakteristiky *SSC* sú závislé od *SSZ*. Jednotkový rozmer *SSC* a *SSO* voči *USS* je definovaný rovnaký. Vzhľadom na snahu dosiahnuť nezávislosť zobrazovacích algoritmov sa na zobrazenie objektov z *USS* do *SSZ* používa *NSS*.

4.2 Zobrazovacie transformácie

V ďalšom sa hlavne budeme zaoberať zobrazovacími transformáciami pre 2D a 3D priestor. V princípe sú používané tri typy zobrazovacích transformácií:

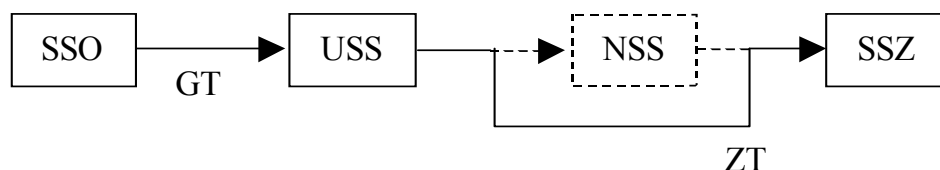
- ❑ *globálnu transformáciu*,
- ❑ *pohľadovú transformáciu*,
- ❑ *premietaciu transformáciu*.

Každý objekt je vytvorený vzhľadom na príslušný vzťažný bod, ktorý spravidla reprezentuje počiatok *SSO*. Pomáha to pri umiestňovaní objektu do scény t.j. do *USS*. Súradnice, ktoré definujú objekt sú relatívne vzhľadom na vzťažný bod a sú definované ako súradnice objektu. *Globálna transformácia (GT)* reprezentuje spôsob, akým sa objekt transformuje zo súradnicového systému objektu (*SSO*) do globálneho súradnicového systému (*USS*) t.j. predstavuje transformačný vzťah $SSO \rightarrow USS$. Globálna transformácia sa používa iba pri geometrickom transformovaní objektu. Pri 3D priestore sa neaplikuje na iné atribúty scény napr. osvetlenie.

Pohľadová transformácia (PT) reprezentuje spôsob transformácie súradníc globálneho súradnicového systému (*USS*) do súradnicového systému kamery (*SSC*) t.j. predstavuje transformačný vzťah $USS \rightarrow SSC$. Je možné uvažovať o nej ako o aplikovaní toho, kde sa kamera v scéne nachádza. O tejto transformácii sa uvažuje ak pracujeme v 3D.

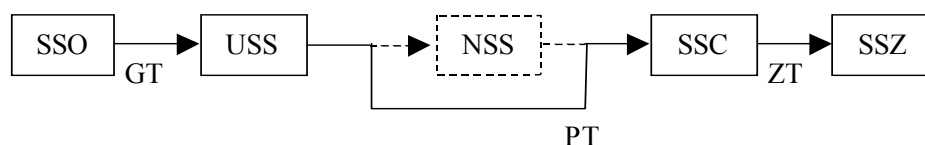
Premietacia transformácia (ZT) transformuje súradnice z priestoru kamery do súradníc zobrazovacieho priestoru (spravidla orezaného) t.j. do súradníc výstupného zariadenia. Takto v 3D táto transformácia predstavuje transformačný vzťah $SSC \rightarrow SSZ$ a v 2D $USS \rightarrow SSZ$ alebo $USS \rightarrow NSS \rightarrow SSZ$, ak sa *NSS* používa.

Celkovo pre 2D je možné celý transformačný reťazec zobraziť tak, ako je na nasledujúcom obrázku.



Obr. 33 Transformačný zobrazovací reťazec 2D

Pre 3D potom bude stav zobrazovacieho reťazca nasledujúci.



Obr. 34 Transformačný zobrazovací reťazec 3D

4.3 Geometrické transformácie

Geometrické transformácie nad objektami v počítačovej grafike sú realizované na úrovni globálnej transformácie. Tieto transformácie menia geometriu objektov. Podľa rozmeru *USS* sa tieto transformácie aplikujú najmä v 2D alebo v 3D priestore. Z hľadiska charakteru zdroja a výsledku môžu byť tieto transformácie *lineárne* alebo *nelineárne*. Štandardne sa používajú lineárne transformácie a týmto sa budeme venovať aj v nasledujúcom. Medzi základné geometrické transformácie radíme:

- ☐ posunutie
- ☐ zrkadlenie
- ☐ zmena mierky
- ☐ skosenie
- ☐ otočenie

Týmto transformáciám sa hovorí aj *afinné transformácie*. Aplikácia týchto transformácií je závislá od typu objektov t.j. či sa jedná o *vektorové* alebo *rastrové* objekty. V prípade aplikácie na vektorové typy sa príslušné transformácie aplikujú spravidla na príslušné vrcholové resp. riadiace body objektu. Pri aplikácii na rastrový typ je spravidla nutná aplikácia na každý jednotlivý bod rastra. Vo všeobecnosti je teda možné povedať, že pod transformáciou objektu sa myslí aplikácia príslušnej transformácie na všetky významové body objektu, z ktorých sa tento skladá. Z tohto dôvodu bude každá z transformácií popísaná z hľadiska aplikácie na oba typy.

Z hľadiska problematiky vysvetľovania bude uvedený ako explicitný, tak aj maticový zápis transformácie, ktorý sa pri implementácii týchto transformácií spravidla používa. Aj keď explicitné vzťahy pre geometrické transformácie sú prehľadnejšie, z hľadiska implementácie sa skôr používa maticové vyjadrenie. Základný transformačný vzťah je možné vyjadriť nasledujúcou rovnicou:

$$\begin{aligned} [x', y'] &= \mathbf{T} * [x, y] & \text{v 2D} \\ [x', y', z'] &= \mathbf{T} * [x, y, z] & \text{v 3D} \end{aligned} \quad (18)$$

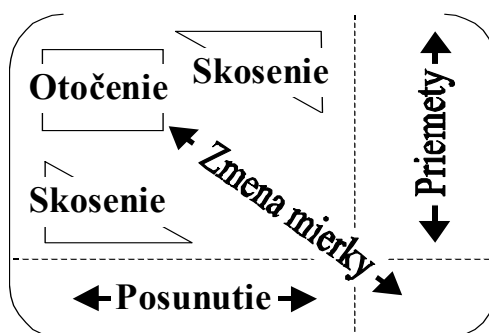
Poznámky:

kde: x, y, z sú pôvodné súradnice
 x', y', z sú nové súradnice po transformácii
 T je transformačná matica
 všetky súradnice sú spravidla myslené v USS a karteziánske.
 Príslušná transformačná matica bude uvedená pri každej transformácii.

Pre zjednodušenie týchto výpočtov zavedieme *homogénne súradnice* a použijeme matice rozmeru $(n+1) \times (n+1)$, kde n je rozmer priestoru, v ktorom sa transformácia aplikuje. Doplnkový rozmer získame tak, že súradnice vynásobíme resp. vydělíme číslom w (homogenizačný faktor). Pričom musí platiť $w \neq 0$ (hodnota w sa najčastejšie nastavuje na 1, a tak sa bude používať aj v nasledujúcom popise). Potom získame pre každý vektor súradníc homogénne súradnice podľa vzťahu (19). Použitie homogénnych súradníc nám umožní vyjadrenie základných lineárnych geometrických transformácií pomocou jednej matice a navyše pomocou matice v homogénnych súradniciach je vyjadriteľná aj premietacia perspektívna transformácia, čo pri použití nehomogénnych karteziánskych súradníc nie je možné.

$$\begin{aligned} [xw, yw] &\equiv [x, y, w] \\ [xw, yw, zw] &\equiv [x, y, z, w] \end{aligned} \quad \text{alebo} \quad \begin{aligned} \begin{bmatrix} x \\ y \\ w \end{bmatrix} &\equiv [x, y, w] && \text{pre 2D} \\ \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} &\equiv [x, y, z, w] && \text{pre 3D} \end{aligned} \quad (19)$$

Potom transformačná matica T má v 2D rozmer 3×3 a v 3D rozmer 4×4 . Pre implementáciu viacerých transformácií sa použije zložená transformácia v tomto prípade potom reprezentovaná násobením transformačných matíc v príslušnom poradí. Inverzná transformácia bude potom reprezentovaná inverznou transformačnou maticou T^{-1} . Pre zjednodušenie pamätania si rozmiestnenia jednotlivých koeficientov transformačných matíc pri použití homogénnych súradníc si teraz uvedieme jednoduchý obrázok so schématickým znázornením ich rozmiestnenia.



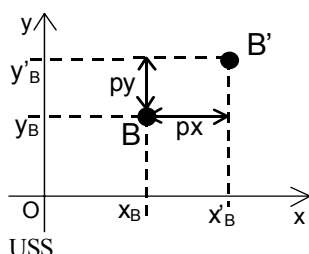
Obr. 35 Schématické rozmiestnenie parametrov transformačných matíc

Pri nasledujúcom popise bude každá transformácia popísaná najprv pomocou explicitných vzťahov a to v 2D. Budeme používať pravouhlú karteziánsku súradnicovú sústavu USS s počiatkom O a osami x, y a bod B so súradnicami x_B, y_B t.j. $B[x_B, y_B]$. Následne bude uvedená transformačná matica v homogénnych súradniciach. Potom bude nasledovať popis pre 3D ($B[x_B, y_B, z_B]$) a nakoniec aplikácia na rastrový objekt. U tohto sa bude jednať o aplikáciu v dvojrozmernom priestore.

4.3.1 Posunutie

Posunutie je transformácia, ktorá mení polohu objektu. Samotné posunutie bodu B v danej súradnicovej sústave je dané vektorom posunutia $P[px, py]$. Potom pre nové súradnice x'_B, y'_B bodu B' môžeme písať:

$$\begin{aligned} x'_B &= x_B + px \\ y'_B &= y_B + py \end{aligned} \quad (20)$$



Obr. 36 Posunutie v 2D

Príslušná transformačná matica je:

$$\mathbf{T}_P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ px & py & 1 \end{bmatrix} \quad (21)$$

Transformačná matica sa podľa vzťahu (18) pri vektorových objektoch aplikuje na všetky ťažiskové body objektu (napr. vrcholy). Pri rastrových typoch sa transformačná matica aplikuje na všetky body rastra.

POSUNUTIE V 3D

Analogicky ako pri 2D je vektor posunutia definovaný ako $P[px, py, pz]$. Potom pre nové súradnice x'_B, y'_B, z'_B bodu B' je možné písať:

$$\begin{aligned} x'_B &= x_B + px \\ y'_B &= y_B + py \\ z'_B &= z_B + pz \end{aligned} \quad (22)$$

Transformačná matica posunutia v 3D je nasledujúca:

$$\mathbf{T}_P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ px & py & pz & 1 \end{bmatrix} \quad (23)$$

4.3.2 Zrkadlenie

Zrkadlenie je transformácia, ktorá vytvára zrkadlový obraz objektu vzhľadom na ťažisko zrkadlenia. Ťažiskom zrkadlenia môže byť objekt, ktorého rozmer je menší ako rozmer používanej súradnicovej sústavy t.j. v 2D to môže byť bod alebo priamka a v 3D to môže byť bod, priamka alebo rovina. Potom zrkadlenie bodu B v danej súradnicovej sústave (t.j. v 2D) je možné vykonať buď vzhľadom na niektorú os

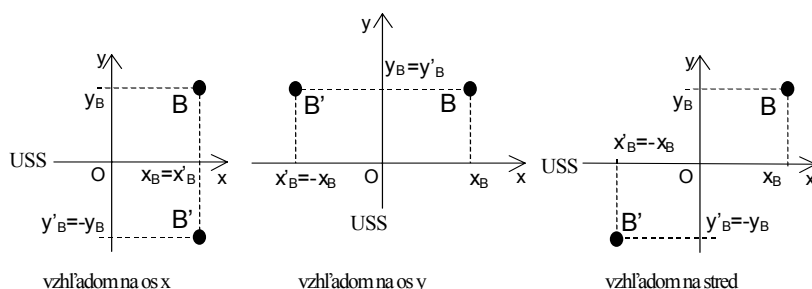
alebo vzhľadom na počiatok súradnicovej sústavy. Potom pre nové súradnice x'_B, y'_B bodu B' po zrkadlení môžeme písať:

$$\begin{array}{lll} x'_B = x_B & x'_B = -x_B & x'_B = -x_B \\ y'_B = -y_B & y'_B = y_B & y'_B = -y_B \end{array} \quad (24)$$

vzhľadom na os x vzhľadom na os y vzhľadom na stred

Jednotlivé transformačné matice zrkadlenia sú nasledujúce:

$$\mathbf{T}_{Zx} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{T}_{Zy} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{T}_{Zs} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (25)$$



Obr. 37 Zrkadlenie v 2D

ZRKADLENIE V 3D

Vzhľadom na tvrdenie o ťažiskovom bode zrkadlenia, môže ním byť v 3D bod, priamka alebo rovina. Potom sa najčastejšie vykonáva zrkadlenie vzhľadom na stred súradnicovej sústavy, vzhľadom na jednotlivé osi a vzhľadom na roviny definované dvojicami osí. Napríklad nové súradnice x'_B, y'_B, z'_B bodu B' po zrkadlení vzhľadom na os x, rovinu xy a počiatok súradnicovej sústavy môžeme získať podľa vzťahov (26). Analogicky ich získame pre ostatné osi (y,z) a roviny (xz, zy).

$$\begin{array}{lll} x'_B = x_B & x'_B = x_B & x'_B = -x_B \\ y'_B = -y_B & y'_B = y_B & y'_B = -y_B \\ z'_B = -z_B & z'_B = -z_B & z'_B = -z_B \end{array} \quad (26)$$

vzhľadom na os x vzhľadom na rovinu xy vzhľadom na stred

Príslušné transformačné matice zrkadlenia je možné napísať nasledovne:

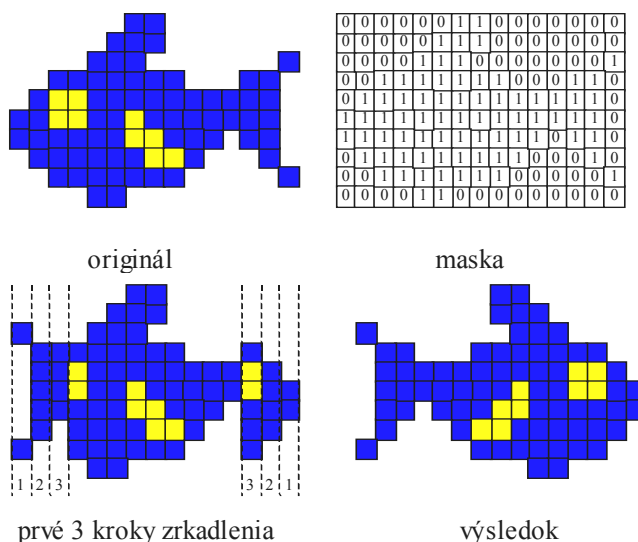
$$\mathbf{T}_{Zx} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{T}_{Zxy} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{T}_{Zs} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (27)$$

ZRKADLENIE RASTROVÝCH OBJEKTŮV



Pre zrkadlenie rastra sa vykoná len výmena stĺpcov (riadkov) celého rastra v cykle napr. pre j od 1 po k, kde k je polovicou rozmeru rastra (n) v príslušnom smere. Potom sa vždy teda vymenia j-ty a (n-j)-ty stĺpec (riadok). Jedná sa teda o zrkadlenie vždy voči jednej z osí súradnicového systému. Ak chceme zrkadliť naraz v oboch smeroch, je možné takúto funkciu implementovať, avšak v princípe je

najjednoduchšie ju definovať ako zloženú transformáciu. Teda najskôr odzrkadlíme voči jednej osi a potom voči druhej, aj keď pre používateľa to môže byť nedeliteľná operácia. Tento postup je určený pre rastrové objekty tvaru štvorstranného pravouholníka. Ak rastrový objekt nemá tento tvar, je nutné vytvoriť jeho opísanú binárnu masku v tvare štvorstranného pravouholníka. Hodnoty prvkov tejto masky sa uložia do matice, kde bod patriaci rastrovému objektu sa označí 1 a bod nepatriaci rastrovému objektu sa označí 0. Potom je postup analogický postupu popísanému v úvode s doplnenou podmienkou aplikácie len na tie body, ktorých hodnota masky je rovná 1. Ukážku horizontálneho zrkadlenia je možné vidieť na nasledujúcom obrázku.



Obr. 38 Postup zrkadlenia nepravidelného rastrového objektu

4.3.3 Zmena mierky

Zmena mierky je transformácia, pomocou ktorej sa dá meniť veľkosť objektu. Zmena mierky je daná ako M -násobok pôvodnej hodnoty v príslušnej osi. M je konštanta ($M > 0$) udávajúca veľkosť zmeny mierky. Pre zväčšenie objektu použijeme $M > 1$ a pre zmenšenie $M < 1$. Nové hodnoty potom budú:

$$\begin{aligned} x'_B &= M_x \times x_B \\ y'_B &= M_y \times y_B \end{aligned} \quad (28)$$

Príslušná transformačná matica je:

$$\mathbf{T}_M = \begin{bmatrix} M_x & 0 & 0 \\ 0 & M_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (29)$$

Mierka M_x mení veľkosť v smere osi x , M_y v smere osi y . Aj keď môže byť M pre obidve osi rozdielne, používa sa najčastejšie rovnaké v oboch osiach.

ZMENA MIERKY V 3D

Pre zmenu mierky v 3D platí rovnaká definícia ako pre jej 2D analógiu s rozšírením o jeden rozmer.

Poznámky:

$$\begin{aligned}x'_B &= M_x \times x_B \\ y'_B &= M_y \times y_B \\ z'_B &= M_z \times z_B\end{aligned}\quad (30)$$

Transformačnú maticu zmeny mierky v 3D zapíšeme nasledovne:

$$\mathbf{T}_M = \begin{bmatrix} M_x & 0 & 0 & 0 \\ 0 & M_y & 0 & 0 \\ 0 & 0 & M_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (31)$$

Pre hodnoty koeficientov zmeny mierky v jednotlivých osiach platí to isté čo pre 2D t.j. ak sa definuje $M_x = M_y = M_z$, potom nedochádza k deformácii objektu.

ZMENA MIERKY RASTROVÝCH OBJEKTOV



Zväčšenie rastrových objektov

Ak koeficienty zmeny mierky M_x aj M_y sú celé čísla, každý bod pôvodného rastra sa nahradí obdĺžnikom so stranami dĺžky mierky M_x a M_y a vyplní farbou bodu pôvodného rastra. Ak koeficient zmeny mierky M_x alebo M_y nie je celé číslo, potom je situácia komplikovanejšia, nakoľko môžeme pracovať výhradne s celočíselnými počtami bodov v rasti. Potom niektoré riadky a/alebo niektoré stĺpce budú mať iný rozmer ako štandardný celočíselný zväčšovací koeficient. V nasledujúcich riadkoch si uvedieme algoritmus, ktorý nájde, ktoré riadky/stĺpce to budú a takisto určí skutočný nový rozmer. Algoritmus bude uvedený pre jeden rozmer, jeho aplikácia na druhý rozmer je analogická.

1. vypočíta sa predpokladaný nový rozmer rastra
 $nový_rozmer = pôvodný_rozmer * M$
2. určí sa celočíselná a desatinná časť koeficientu zmeny mierky
 $ckrok = celá\ časť(M)$
 $dkrok = desatinná\ časť(M)$
3. vykoná sa test, či desatinná časť koeficientu zmeny mierky je väčšia ako 0.5.

Ak *áno*, potom sa

- ❑ nastaví pomocný príznak na 1
 $príznak = 1$
- ❑ vytvorí sa súmerná desatinná časť koeficientu zmeny mierky voči 0.5
 $dkrok = absolútna\ hodnota(1 - dkrok)$
- ❑ pre ďalší výpočet sa upraví M na koeficient súmerný voči 0.5
 $M = ckrok + dkrok$

Ak *nie*, potom sa neurobí nič iné, iba sa nastaví pomocný príznak na 0

$Príznak = 0$

4. určí sa pomocný výpočtový nový rozmer. Tento výpočtový nový rozmer je buď totožný s novým rozmerom (ak desatinná časť koeficientu zmeny mierky M bola ≤ 0.5) alebo je rovný rozmeru po prenasobení upraveným, vlastne doplnkovým koeficientom M (ak desatinná časť koeficientu zmeny mierky M bola > 0.5).
 $výpočtový_nový_rozmer = pôvodný_rozmer * M$

5. následne sa vypočíta veľkosť rozmeru rastra, ak by bol koeficient zmeny mierky len celá časť M .

$$\text{celočíselný_rozmer} = \text{pôvodný_rozmer} * \text{ckrok}$$

6. zistí sa celočíselný rozdiel medzi výpočtovým novým rozmerom a celočíselným rozmerom aby sa zistilo, koľko bodov je nutné ešte doplniť. Vypočítaný rozdiel sa zväčší o 1, aby sa priebežne korigovala zaokrúhľovacia chyba.

$$\text{rozdiel} = \text{výpočtový_nový_rozmer} - \text{celočíselný_rozmer}$$

$$\text{rozdiel} = \text{rozdiel} + 1$$

7. vypočíta sa korekčný krok, ktorý bude určovať, kedy sa vykoná zmena celočíselného koeficientu zmeny mierky o 1 (+ alebo – podľa toho či nebola alebo bola desatinná časť koeficientu zmeny mierky > 0.5). Tento korekčný krok sa navyše ešte opäť kvôli vyrovnaní chyby zaokrúhlenia zväčší o 1.

$$\text{korekčný_krok} = \text{pôvodný_rozmer} / \text{rozdiel}$$

$$\text{korekčný_krok} = \text{korekčný_krok} + 1$$

Obidve korekcie, táto aj v predchádzajúcom kroku síce pomáhajú korigovať zaokrúhľovacie chyby, napriek tomu pri určitých hodnotách koeficienta M dochádza k určitej chybe medzi konečným skutočne získaným počtom bodov nového rozmeru rastra a predpokladaným počtom bodov nového rozmeru rastra, ktorý bol vlastne vypočítaný v bode 1 algoritmu, spravidla o ± 1 bod. Toto je možné v konečnom dôsledku korigovať tak, že posledný krok sa zväčší alebo zmenší ešte navyše o rozdiel medzi skutočne získanou hodnotou a predpokladanou hodnotou nového rozmeru rastra.

8. nakoniec sa vykoná konečné priradenie celočíselnej štandardnej hodnoty koeficienta zmeny mierky a celočíselného korekčného koeficienta zmeny mierky, ktorý sa aplikuje v každom korekčnom kroku. Toto priradenie sa vykoná podľa hodnoty príznaku udávajúceho hodnotu desatinnej časti koeficientu zmeny mierky M .

Ak $\text{príznak} = 1$, potom

- ☐ celočíselná štandardná hodnota koeficienta zmeny mierky je $\text{štandardný_koeficient} = \text{ckrok} + 1$
- ☐ celočíselná korekčná hodnota koeficienta zmeny mierky je $\text{korekčný_koeficient} = \text{ckrok}$

Inak (t.j. ak $\text{príznak} = 0$), potom

- ☐ celočíselná štandardná hodnota koeficienta zmeny mierky je $\text{štandardný_koeficient} = \text{ckrok}$
- ☐ celočíselná korekčná hodnota koeficienta zmeny mierky je $\text{korekčný_koeficient} = \text{ckrok} + 1$

9. tým sme určili všetko potrebné pre zväčšenie rastrového objektu a je možné písať, že v príslušnom rozmere budú jednotlivé body rastra v každom kroku. zväčšované štandardným koeficientom a v každom korekčnom kroku korekčným koeficientom. Celkový počet krokov je daný pôvodným rozmerom rastra. Ak sčítame postupne hodnoty aplikovaných koeficientov v každom kroku, dostaneme skutočný počet bodov nového rastra.

Aplikáciu si ukážeme na príkladoch. Ak je pôvodná šírka rastra 15 a pôvodná výška 10, potom pri koeficiente zmeny mierky $M=2.2$ je nová šírka rastra 33 a nová výška

Poznámky:

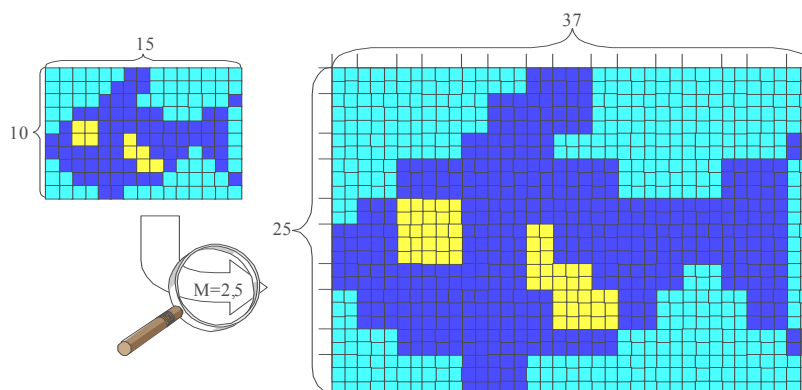
22, kde štandardný koeficient je 2 a korekčný koeficient 3 sa bude aplikovať v každom 4.-om kroku. Potom postupnosť pre jednotlivé smery bude nasledovná:

šírka: 2 2 2 3 2 2 2 3 2 2 2 3 2 2 2 a výška: 2 2 2 3 2 2 2 3 2 2 2

Ak bude koeficient zmeny mierky $M=2.8$, potom je nová šírka rastra 42 a nová výška rastra 28, kde štandardný koeficient je 3 a korekčný koeficient 2 sa bude aplikovať v každom 4.-om kroku. Výsledná postupnosť pre jednotlivé smery bude:

šírka: 3 3 3 2 3 3 3 2 3 3 3 2 3 3 3 a výška: 3 3 3 2 3 3 3 2 3 3 3

Ako to bude vyzerat' v skutočnosti pri koeficiente $M=2.5$ ukazuje nasledujúci obrázok.



Obr. 39 Ukážka zväčšenia rastrového objektu pri koeficiente $M=2.5$

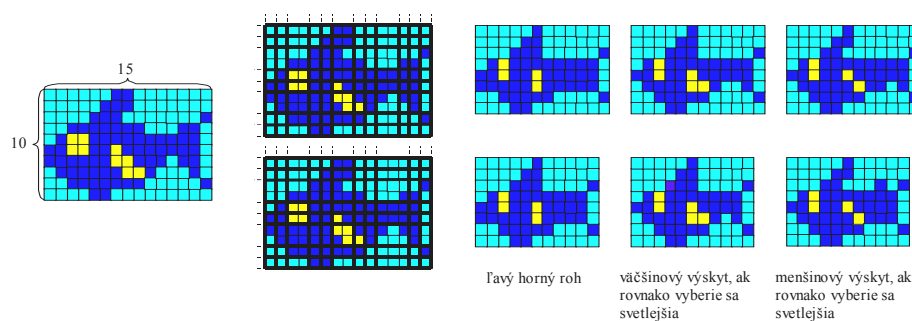
Zmenšenie rastrových objektov

Ak koeficienty zmeny mierky M_x aj M_y sú celé číslo, každý obdĺžnik so stranami dĺžky mierky M_x a M_y v origináli sa nahradí jedným bodom vo výsledku, pričom jeho farba sa dá získať viacerými spôsobmi:

1. priemerom bodov obdĺžnika alebo použitím mediánovej funkcie.
2. zistí sa početnosť výskytu farieb v obdĺžniku a vyberie sa tá farba, ktorá sa vyskytuje najčastejšie. Ak je výskyt farieb rovnaký, vyberie sa farba podľa iného pravidla alebo ľubovoľná z vyskytujúcich sa farieb.
3. vyberie sa farba, ktorá sa vyskytuje najmenej krát.
4. vyberie sa farba ľavého horného bodu obdĺžnika.

Spôsobom 1. a 2. by mohlo dôjsť k orezaniu hrany, spôsob 3. zas môže spôsobiť náhodný výskyt farieb.

Ak niektorý koeficient zmeny mierky (v prípade, že sú rozdielne) nie je celé číslo, je postup obdobný ako pri zväčšovaní rastrového objektu. Na nájdenie počtu bodov rastra v oboch smeroch na definovanie strán nahradzovaného pravouholníka, ktoré vo výsledku budú reprezentovať jeden výsledný bod rastra, je možné postupovať obdobným spôsobom ako pri zväčšovaní. Potom je nutné v algoritme ale zameniť zdroj a cieľ. Pri zmenšovaní navyše zohráva úlohu aj to, ako sa zaokrúhli nový predpokladaný rozmer. Na nasledujúcom obrázku je možné vidieť výsledky aplikácie tohto algoritmu pri koeficiente zmenšenia $M=0.7$ v oboch smeroch. Nakoľko je vodorovný nový rozmer 10.5, v hornom riadku je stav ak sa zvolí zaokrúhlenie nahor a v dolnom ak sa zvolí zaokrúhlenie nadol. Jednotlivé prípady vpravo ukazujú výsledok pri rôznych spôsoboch určenia farby výsledného bodu.



Obr. 40 Ukážka zmenšenia rastrového objektu pri koeficiente $M=0.7$

4.3.4 Skosenie

Skosenie je transformácia, ktorá spôsobí deformáciu objektu vo forme vyklonenia v príslušnej osi. Skosenie bodu B v danej súradnicovej sústave je dané koeficientom skosenia S . Potom pre nové súradnice x'_B, y'_B bodu B môžeme písať:

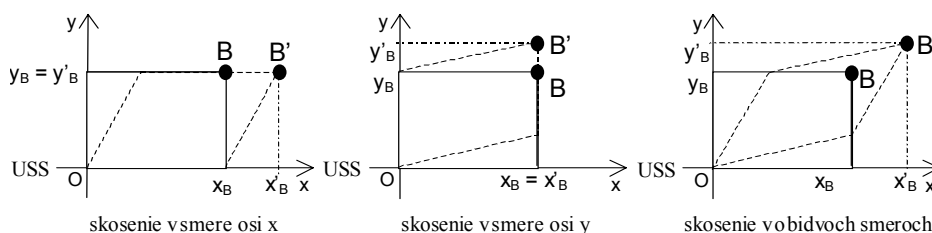
$$\begin{aligned} x'_B &= x_B + S_x \times y_B & x'_B &= x_B \\ y'_B &= y_B & y'_B &= y_B + S_y \times x_B \end{aligned} \quad (32)$$

v smere osi x v smere osi y

Samozrejme, že pri aplikovaní na jeden bod, výsledok transformácie nie je až taký zrejmy. Preto sa táto transformácia aplikuje na rozsiahlejšie dvojrozmerné objekty. Transformačné matice sú nasledovné:

$$\mathbf{T}_{Sx} = \begin{bmatrix} 1 & 0 & 0 \\ S_x & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{T}_{Sy} = \begin{bmatrix} 1 & S_y & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (33)$$

Parametre S_x a S_y sú príslušné koeficienty skosenia v príslušnej osi. Skosenie deformuje napr. štvorec na rovnobežník, v ktorom je každý bod posunutý v príslušnom smere o vzdialenosť, ktorá je úmerná koeficientu skosenia (pozri nasledujúci obrázok).

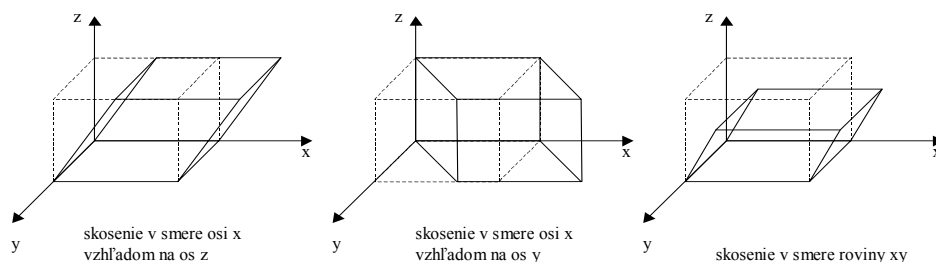


Obr. 41 Skosenie v 2D

SKOSENIE V 3D

Jedná sa o rozšírenie skosenia z 2D do trojrozmerného priestoru. Túto transformáciu môžeme v 3D vykonať v smere niektorej osi, niektorej roviny ich kombinácií alebo dokonca v smere všetkých osí. Pri skosení v smere niektorej z osí je nutné ešte určiť, vzhľadom na ktorú os sa bude skosenie definovať. Ak sa vyberie možnosť skosenia v smere dvoch osí, potom tieto osi určujú rovinu, v smere ktorej bude aplikované skosenie vzhľadom na zvyšnú tretiu os. Príklad ukazuje nasledujúci obrázok.

Poznámky:



Obr. 42 Skosenie v 3D

Príslušné transformačné vzťahy budú potom vyzerat' nasledovne:

$$\begin{aligned}
 x'_B &= x_B + S_{xoz} \times z_B & x'_B &= x_B + S_{xoy} \times y_B & x'_B &= x_B + S_{xoz} \times z_B \\
 y'_B &= y_B & y'_B &= y_B & y'_B &= y_B + S_{yoz} \times z_B \\
 z'_B &= z_B & z'_B &= z_B & z'_B &= z_B
 \end{aligned} \tag{34}$$

v smere osi x v smere osi x v smere roviny xy
 vzhľadom na os z vzhľadom na os y

K týmto vzťahom potom je možné definovať príslušné transformačné matice:

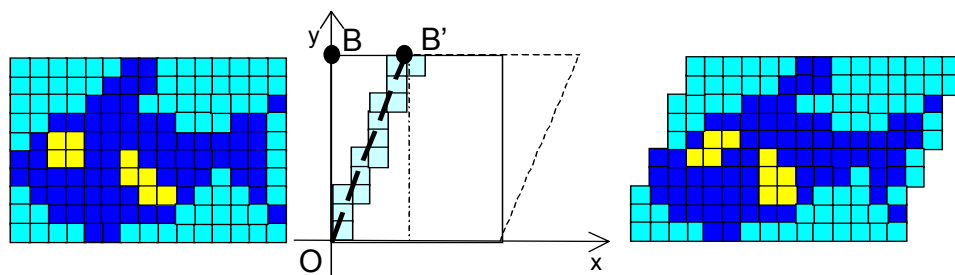
$$T_{Sxoz} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ S_{xoz} & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_{Sxoy} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ S_{xoy} & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_{Sxy} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ S_{xoz} & S_{yoz} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{35}$$

Ostatné vzťahy a transformačné matice je možné odvodiť analogicky.

SKOSENIE RASTROVÉHO OBJEKTU



Ako bolo uvedené, skosenie je transformácia, ktorá spôsobí vyklonenie objektu v príslušnej osi. Pri rastrovom objekte nakoľko sa tu jedná o 2D transformáciu, môžeme rastrový objekt skosiť v smere osi x (t.j. horizontálne), v smere osi y (t.j. vertikálne) alebo v oboch smeroch. Pri popise budeme vychádzať z Obr. 43. Úsečka OB predstavuje ľavú hranu rastrového objektu. Po skosení sa bod B transformuje do bodu B' . Potom úsečka OB' bude predstavovať ľavú hranu skoseného rastrového objektu. Potrebujeme vlastne zistiť o koľko bodov je potrebné, v tomto prípade vodorovne, poposúvať jednotlivé riadky (pri skosení v smere osi y jednotlivé stĺpce) v smere osi x . Tieto hodnoty získame jednoduchým spôsobom a to výpočtom rastrovej náhrady úsečky OB' (použitím napr. DDA alebo Bresenhamovho algoritmu). Jednotlivé diferencie rastrovej náhrady nám následne určia koľko riadkov a o koľko je nutné poposúvať, aby sme vo výsledku dostali skosený rastrový objekt. V prípade skosenia v oboch smeroch sa u rastrového objektu aplikuje postupná zložená transformácia v jednotlivých smeroch.



Obr. 43 Postup skosenia rastrového objektu

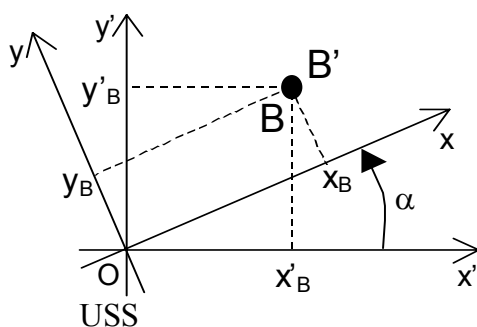
4.3.5 Otočenie

Transformácia objektu po kruhovej dráhe sa nazýva otáčanie (rotácia). Otočenie bodu B okolo počiatku súradnej sústavy O je dané orientovaným uhlom α (narastá proti smeru hodinových ručičiek). Nové súradnice x'_B, y'_B bodu B' sú:

$$\begin{aligned} x'_B &= x_B \times \cos(\alpha) - y_B \times \sin(\alpha) \\ y'_B &= x_B \times \sin(\alpha) + y_B \times \cos(\alpha) \end{aligned} \quad (36)$$

Transformačná matica je nasledujúca:

$$\mathbf{T}_O = \begin{bmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (37)$$



Obr. 44 Otočenie v 2D

Ak stred otáčania nie je počiatok súradnicovej sústavy, musí sa otočenie zložiť z posunutia do počiatku súradnicovej sústavy, otočenia a spätného posunutia:

$$\mathbf{T} = \mathbf{T}_P^{-1} * \mathbf{T}_O * \mathbf{T}_P \quad (38)$$

kde \mathbf{T}_P je matica transformácie posunutia, \mathbf{T}_O je transformácia otočenia o uhol α a \mathbf{T}_P^{-1} je spätná transformácia posunutia.

OTOČENIE V 3D

Otočenie v trojrozmernom priestore je zložitejšou transformáciou ako v 2D. Oproti 2D je možné objekt otáčať vzhľadom na niektorú z osí. Podľa toho, ktorá os to bude, potom nové hodnoty súradníc bodu B budú nasledujúce:

$$\begin{aligned} x'_B &= x_B \\ y'_B &= y_B \times \cos(\alpha) - z_B \times \sin(\alpha) \\ z'_B &= y_B \times \sin(\alpha) + z_B \times \cos(\alpha) \end{aligned} \quad \text{otočenie okolo osi } x \quad (39)$$

$$\begin{aligned} x'_B &= x_B \times \cos(\alpha) + z_B \times \sin(\alpha) \\ y'_B &= y_B \\ z'_B &= -x_B \times \sin(\alpha) + z_B \times \cos(\alpha) \end{aligned} \quad \text{otočenie okolo osi } y \quad (40)$$

$$\begin{aligned} x'_B &= x_B \times \cos(\alpha) - y_B \times \sin(\alpha) \\ y'_B &= x_B \times \sin(\alpha) + y_B \times \cos(\alpha) \\ z'_B &= z_B \end{aligned} \quad \text{otočenie okolo osi } z \quad (41)$$

Na tomto mieste ešte poznamenajme, že otočenie okolo osi z je vlastne len zovšeobecnením otočenia uvedenom pri otočení v 2D. Jednotlivé transformačné matice budú vyzerat' nasledovne (ak uhol otočenia je α):

$$\begin{aligned} T_{Ox} &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \alpha & \sin \alpha & 0 \\ 0 & -\sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} & T_{Oy} &= \begin{bmatrix} \cos \alpha & 0 & \sin \alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \alpha & 0 & \cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ T_{Oz} &= \begin{bmatrix} \cos \alpha & \sin \alpha & 0 & 0 \\ -\sin \alpha & \cos \alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \quad (42)$$

Pri otáčaní bodu okolo všeobecnej priamky p , sa použijú opäť len skôr uvedené transformácie otočenia, avšak táto operácia je zloženou transformáciou. Algoritmus je nasledovný:

1. Posunutie priamky p tak, aby prechádzala počiatkom O súradnicovej sústavy.
2. Vykonalenie transformácií otočenia tak, aby sa priamka p stotožnila s niektorou z osí.
3. Otočenie bodu známou transformáciou okolo príslušnej osi.
4. Spätné transformácie na základe bodov 2 a potom 1.

OTOČENIE RASTROVÉHO OBJEKTU



Pri nasledujúcom popise sa predpokladá, že stredom otočenia je vždy stred rastra.

Otočenie s interpoláciou medziľahých bodov

Ak je rastrový objekt reprezentovaný ako systém vodorovných úsečiek, je zrejmé že po transformácii môžu v rastri výsledného otočeného obrazu zostať body, ktoré nie sú prvkami žiadnej z otočených úsečiek, tzv. *diery*. Neznáme hodnoty v týchto bodoch sa určujú interpoláciou.

Interpolácia sa robí na základe hodnôt v najbližších štyroch symetricky umiestnených bodoch rastra $[j,k]$, $[j+1,k]$, $[j,k+1]$, $[j+1,k+1]$. Najjednoduchší typ interpolácie v rovine je interpolácia pomocou najbližšieho suseda, nazývaná aj interpolácia 0-tého rádu. Jadrom metódy je nájdenie bodu rastra $[j',k']$, ktorý je najbližšie k bodu $[j^*,k^*]$, v ktorom sa interpoluje. Interpolovaná hodnota sa určí jednoducho ako $g(j^*,k^*)=g(j',k')$. Metóda je veľmi rýchla, pretože nezahŕňa žiadne aritmetické operácie. Je však nepresná a dobré výsledky dáva zriedkavo.

Pri lineárnej interpolácii sa vychádza zo štyroch najbližších bodov rastra, ktoré všeobecne nemusia byť symetricky umiestnené. Pre tieto štyri body sa vypočítajú ich vzdialenosti $d1$, $d2$, $d3$, $d4$ od bodu $[j^*,k^*]$ rastra, v ktorom je hodnota neznáma. Lineárny interpolačný vzťah má tvar:

$$\begin{aligned} g(j^*, k^*) &= \left[\frac{1}{d1} + \frac{1}{d2} + \frac{1}{d3} + \frac{1}{d4} \right] \times z \\ z &= \left[\frac{1}{d1} \times g(j, k) + \frac{1}{d2} \times g(j, k+1) + \frac{1}{d3} \times g(j+1, k) + \frac{1}{d4} \times g(j+1, k+1) \right] \end{aligned} \quad (43)$$

Nevýhodou lineárnej interpolačnej metódy je výpočtová náročnosť.

Výpočtovú náročnosť lineárnej interpolácie v rovine zjednodušuje bilineárna interpolácia, ktorý pozostáva z dvoch krokov. V prvom kroku sa pri pevných

hodnotách jednej súradnice, napr. k , vypočítajú dve jednorozmerné interpolácie v smere druhej súradnicovej osi (j):

$$\begin{aligned} g1 &= [g(j, k)(j - j^* + 1) + g(j + 1, k)(j^* - j)] \\ g2 &= [g(j, k + 1)(j - j^* + 1) + g(j + 1, k + 1)(j^* - j)] \end{aligned} \quad (44)$$

V druhom kroku sa na základe získaných hodnôt $g1$, $g2$ vypočíta lineárna interpolácia $g3$ v smere osi k :

$$g3 = g1(k - k^* + 1) + g2(k^* - k) \quad (45)$$

a interpolovaná hodnota je:

$$gd(j^*, k^*) = g3 \quad (46)$$

Tento druh interpolácie sa v číslicovom spracovaní obrazov používa veľmi často.

Najpresnejšou, zároveň však výpočtovo najnáročnejšou metódou interpolácie je splineova interpolácia. Teóriu splineovej interpolácie možno v danej oblasti aplikovať rôzne. Najčastejšie je to dvojnásobné použitie jednorozmenej splineovej interpolácie alebo priamo interpolácia v rovine pomocou 2D splineov.

Otočenie s interpoláciou všetkých bodov

Tento spôsob otočenia obchádza nutnosť hľadania tzv. dier v otočenom rastrovom objekte. Pretože úsečka po otočení nemá rovnaký počet bodov ako pred otočením, treba vyriešiť problém určenia farieb otočených bodov.

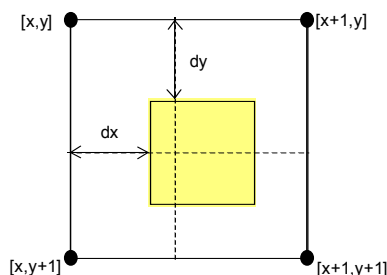
Na začiatku sa otočia len rohové body rastrového objektu zloženou transformáciou. Pre každé dva otočené body sú nájdené pomocou Bresenhamovho algoritmu, body úsečky, ktorých súradnice sú uchované do poľa napr. *Suradnice[]*. Súradnice bodov je nutné utriediť podľa y -ovej súradnice od najväčšieho po najmenší a v rámci tohto utriedenia zoradiť body podľa x -ovej súradnice. Týmto utriedením sa získajú krajné body úsečiek, ktoré vyplňajú otočený obdĺžnik (rastrový objekt).

Pre každý bod poľa *Suradnice[]* je potrebné určiť jeho farbu. Kvôli tomu sa bod spätne otočí, aby sa zistili jeho pôvodné súradnice. Tieto súradnice však môžu mať po transformácii reálne hodnoty, takže bod zasahuje aj do iných bodov - svojich susedov. Farba bodu sa určí váženým priemerom farieb týchto štyroch bodov v pôvodnom, neotočenom stave. Každý z bodov ovplyvňuje výslednú farbu váhou rovnou časti bodu, ktorá je zakrytá výsledným bodom. Podľa použitého farebného modelu sa vypočítajú jednotlivé zložky. Napr. ak sa použije RGB model potom sa musia vypočítať zložky R, G, B jednotlivo a z nich sa zloží výsledná farba:

$$\begin{aligned} R1 &= R(x, y) \cdot (1 - dx) \cdot (1 - dy) \\ R2 &= R(x + 1, y) \cdot dx \cdot (1 - dy) \\ R3 &= R(x, y + 1) \cdot (1 - dx) \cdot dy \\ R4 &= R(x + 1, y + 1) \cdot dx \cdot dy \\ R &= R1 + R2 + R3 + R4 \end{aligned} \quad (47)$$

Zložky G a B sa vypočítajú analogicky.

Poznámky:



Obr. 45 Určenie farby bodu pomocou najbližších susedov

Porovnanie interpolácií

Prvý spôsob interpolácie je teoreticky jednoduchší, ale pri praktickej realizácii je vyhľadávanie dier veľmi náročné. Je totiž nutné reprezentovať už vyplnené body, aby ich bolo možné odlíšiť od dier. Navyše týmto spôsobom môže pri určitých uhloch dochádzať ku deformácii objektov.

Druhý spôsob je síce časovo náročnejší kvôli tomu, že sa musí interpolovať každý bod rastra, ale praktická realizácia je oveľa jednoduchšia.

QUATERNIÓNŮ



Pri aplikácii transformačných matíc otočenia často vzniká problém s tzv. ich zablokovaním t.j., keď pokus o otočenie objektu zlyhá kvôli poradiu, v akom sa rotácie vykonávajú. Tento problém umožňuje odstrániť použitie *quaternionov*. Ich aplikácia v tomto prípade prináša aj implementáciu hladkej a súvislej rotácie. Skôr ako bude pojednané o quaternionoch, zopakujeme si niečo o komplexných číslach. Ako je známe, systém komplexných čísel je definovaný pomocou imaginárnej jednotky i , pričom $i * i = i^2 = -1$.

Keďže i nie je reálne číslo, môžeme písať komplexné čísla ako reálne čísla použitím reálnej a imaginárnej časti. Podobne ako u vektorov je možné definovať opačné číslo a veľkosť (absolútnu hodnotu) komplexných čísel.

$$\begin{aligned}
 z &= a + bi && \text{komplexné číslo } z \\
 z' &= a - bi && \text{opačné číslo k číslu } z \\
 |z| &= \sqrt{z * z'} = \sqrt{a^2 + b^2} && \text{veľkosť (absolútna hodnota) čísla } z
 \end{aligned} \tag{48}$$

Použitím predchádzajúcich vzťahov, môžeme definovať násobenie dvoch komplexných čísel

$$\left. \begin{aligned} z_1 &= a_1 + b_1 i \\ z_2 &= a_2 + b_2 i \end{aligned} \right\} z_1 * z_2 = (a_1 a_2 - b_1 b_2) + (a_1 b_2 + b_1 a_2) i \tag{49}$$

Quaterniony sú rozšírením komplexných čísel. Z hľadiska aplikácie v počítačovej grafike nás bude najviac zaujímať z ohľadom na transformáciu otočenia v trojrozmernom priestore. U quaternionov namiesto už spomínaného i tu máme k dispozícii tri rôzne čísla, ktorých ale druhé mocniny sa tiež rovnajú číslu -1 a sú označované (tieto čísla) ako i , j , k (vychádzajúc z jednotkových vektorov jednotlivých súradnicových osí).

$$i * i = -1 \quad j * j = -1 \quad k * k = -1 \tag{50}$$

Ak vynásobíme dve z týchto čísel medzi sebou, zistíme, že ich vynásobenie sa správa tak ako vynásobenie dvoch jednotkových vektorov, ktoré tvoria bázu.

$$\begin{aligned}
\mathbf{i} * \mathbf{j} &= -\mathbf{j} * \mathbf{i} = \mathbf{k} \\
\mathbf{j} * \mathbf{k} &= -\mathbf{k} * \mathbf{j} = \mathbf{i} \\
\mathbf{k} * \mathbf{i} &= -\mathbf{i} * \mathbf{k} = \mathbf{j}
\end{aligned} \tag{51}$$

Opačné číslo a veľkosť quaterniónu sa hľadá tak isto ako pri komplexných číslach (pozri vzťah (48)):

$$\begin{aligned}
\mathbf{q} &= w + x\mathbf{i} + y\mathbf{j} + z\mathbf{k} \\
\mathbf{q}' &= w - x\mathbf{i} - y\mathbf{j} - z\mathbf{k} \\
|\mathbf{q}| &= \sqrt{\mathbf{q} * \mathbf{q}'} = \sqrt{w^2 + x^2 + y^2 + z^2}
\end{aligned} \tag{52}$$

Inverzná hodnota (prevrátená hodnota) quaterniónu je definovaná podľa nasledujúceho vzťahu:

$$\mathbf{q}^{-1} = \mathbf{q}' / (\mathbf{q} * \mathbf{q}') \tag{53}$$

Ak quaternión \mathbf{q} má dĺžku 1, hovoríme že quaternión \mathbf{q} je *jednotkový quaternión*.

$$\text{Pre jednotkový quaternión platí: } |\mathbf{q}| = 1 \Rightarrow \mathbf{q}^{-1} = \mathbf{q}' \tag{54}$$

a teda inverzia jednotkového quaterniónu je rovná jeho opačnému číslu.

$$\text{Quaternióny sú asociatívne: } (\mathbf{q}_1 * \mathbf{q}_2) * \mathbf{q}_3 = \mathbf{q}_1 * (\mathbf{q}_2 * \mathbf{q}_3) \tag{55}$$

$$\text{Quaternióny nie sú komutatívne: } \mathbf{q}_1 * \mathbf{q}_2 \neq \mathbf{q}_2 * \mathbf{q}_1 \tag{56}$$

Quaternióny môžeme zapísať niekoľkými spôsobmi. A to:

1. ako lineárnu kombináciu 1, \mathbf{i} , \mathbf{j} a \mathbf{k}
2. ako vektor štyroch koeficientov v tejto lineárnej kombinácii
3. ako skalár pre koeficient 1 a vektor pre koeficienty z imaginárnej časti

Jednotlivé zápisy v príslušnom poradí potom vyzerajú nasledovne:

$$\mathbf{q} = w + x\mathbf{i} + y\mathbf{j} + z\mathbf{k} = [x, y, z, w] = (\mathbf{s}, \mathbf{v}) \text{ kde } \mathbf{s} = w \text{ a } \mathbf{v} = [x, y, z] \tag{57}$$

Ak máme potom dva vektory zapísané v tvare (\mathbf{s}, \mathbf{v}) , môžeme napísať výsledok ich násobenia na základe vzťahu (49) nasledovne:

$$\left. \begin{aligned} \mathbf{q}_1 &= (s_1, \mathbf{v}_1) \\ \mathbf{q}_2 &= (s_2, \mathbf{v}_2) \end{aligned} \right\} \quad \mathbf{q}_1 * \mathbf{q}_2 = (s_1 \cdot s_2 - \mathbf{v}_1 \cdot \mathbf{v}_2, s_1 \cdot \mathbf{v}_2 + s_2 \cdot \mathbf{v}_1 + \mathbf{v}_1 * \mathbf{v}_2) \tag{58}$$

Pri zápise druhým spôsobom po roznásobení dostaneme jednotlivé zložky:

$$\begin{aligned}
\mathbf{q} &= \mathbf{q}_1 * \mathbf{q}_2 = [(q_x, q_y, q_z), q_w] \\
q_x &= q_{1w} \cdot q_{2x} + q_{1x} \cdot q_{2w} + q_{1y} \cdot q_{2z} - q_{1z} \cdot q_{2y} \\
q_y &= q_{1w} \cdot q_{2y} - q_{1x} \cdot q_{2z} + q_{1y} \cdot q_{2w} + q_{1z} \cdot q_{2x} \\
q_z &= q_{1w} \cdot q_{2z} + q_{1x} \cdot q_{2y} - q_{1y} \cdot q_{2x} + q_{1z} \cdot q_{2w} \\
q_w &= q_{1w} \cdot q_{2w} - q_{1x} \cdot q_{2x} - q_{1y} \cdot q_{2y} - q_{1z} \cdot q_{2z}
\end{aligned} \tag{59}$$

Na základe toho, je možné interpretovať rotáciu pomocou quaterniónov. Môžeme vypočítať rotáciu okolo jednotkového vektora $\mathbf{u} = [u_x, u_y, u_z]$ o uhol α . Quaternión, ktorý vypočíta túto rotáciu sa zapíše takto:

$$\mathbf{q} = (\mathbf{s}, \mathbf{v}), \text{ kde } s = \cos\left(\frac{\alpha}{2}\right) \text{ a } \mathbf{v} = \mathbf{u} \cdot \sin\left(\frac{\alpha}{2}\right) \tag{60}$$

resp. jeho výsledný tvar (podľa druhého typu zápisu) vyzerá nasledovne:

$$\mathbf{q} = \left[\left(u_x \cdot \sin\left(\frac{\alpha}{2}\right), u_y \cdot \sin\left(\frac{\alpha}{2}\right), u_z \cdot \sin\left(\frac{\alpha}{2}\right) \right), \cos\left(\frac{\alpha}{2}\right) \right] \quad (61)$$

Bod $B[x_B, y_B, z_B]$ v priestore je možné reprezentovať pomocou quaterniónu $\mathbf{B} = (0, B)$. Potom požadovanú rotáciu vypočítame pomocou tejto rovnice:

$$\mathbf{B}_{\text{otočený}} = \mathbf{q} \mathbf{B} \mathbf{q}^{-1} \quad (62)$$

Ak vychádzame z druhého spôsobu zápisu (vzťah (57)), tak transformačná matica rotácie \mathbf{Q} quaternióna \mathbf{q} bude mať tvar podľa vzťahu (63) a aplikuje sa ako matica otočenia uvedená v kapitole otáčania v trojrozmernom priestore.

$$\mathbf{Q} = \begin{bmatrix} 1 - 2y^2 - 2z^2 & 2xy - 2zw & 2xz + 2yw & 0 \\ 2xy + 2zw & 1 - 2x^2 - 2z^2 & 2yz - 2xw & 0 \\ 2xz - 2yw & 2yz + 2xw & 1 - 2x^2 - 2y^2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (63)$$

Na záver si ešte ukážeme ako bude vyzerat' zret'azenie rotácií pri použití quaterniónov. Skúsme uskutočniť dve rotácie na tom istom objekte. Napríklad, ak máme prepojenie medzi myšou a objektom, čo znamená, že každý pohyb myši si vynucuje aj následný pohyb objektu (objekt „sleduje“ pohyby myši). Toto sa dá veľmi ľahko a numericky stabilne vyjadriť pomocou quaterniónov.

Predpokladajme, že \mathbf{q}_1 a \mathbf{q}_2 sú jednotkové quaternióny reprezentujúce dve rotácie. Chceme uskutočniť najprv \mathbf{q}_1 a potom \mathbf{q}_2 . Ak to takto chceme urobiť, tak pre \mathbf{q}_2 potrebujeme výsledok \mathbf{q}_1 , potom preskupením produktu použitím asociativity quaterniónov zistíme, že kompozitné rotácie sú reprezentované quaterniónom $\mathbf{q}_2 * \mathbf{q}_1$. Vychádzajúc zo vzťahu (62) je potom možné písať:

$$\mathbf{q}_2 * (\mathbf{q}_1 * \mathbf{B} * \mathbf{q}_1^{-1}) * \mathbf{q}_2^{-1} = (\mathbf{q}_2 * \mathbf{q}_1) * \mathbf{B} * (\mathbf{q}_1^{-1} * \mathbf{q}_2^{-1}) = (\mathbf{q}_2 * \mathbf{q}_1) * \mathbf{B} * (\mathbf{q}_2 * \mathbf{q}_1)^{-1} \quad (64)$$

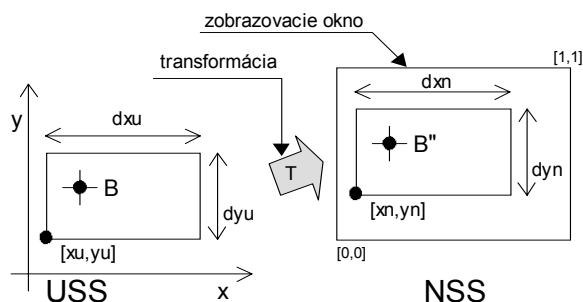
A tak čas, ktorý potrebujeme na vypočítanie matic je potrebný iba vtedy, ak chceme transformovať objekt. Pre ostatné operácie nám stačí sa iba pozrieť na quaternióny. Produkty matic potrebujú omnoho viac operácií ako produkty quaterniónov, takže môžeme ušetriť viac času napr. na väčšiu numerickú presnosť pomocou quaterniónov ako pomocou matic.

4.4 2D premietacie transformácie

Pri zobrazovaní objektov z *USS* do *NSS* sa zobrazuje len vymedzená spravidla pravouhlá (obdĺžniková) časť priestoru *USS*, ktorú chce používateľ zobrazit'. Tejto časti budeme hovoriť *okno* (window). Strany okna sú spravidla rovnobežné so súradnicovými osami *USS*. Okno však nemusí byť zobrazené na celý priestor rozsahu súradníc *NSS* (niekedy aj *SSZ*), ale býva zobrazené len do určenej pravouhlej (obdĺžnikovej) oblasti zobrazovacieho priestoru, do tzv. *výrezu* (viewport, pozri Obr. 46). Toto zobrazenie potom budeme nazývať *transformácia okno-výrez*. Pre príslušnú transformáciu platí nasledovné (bod B má súradnice v *USS* $B(x, y)$ a v *NSS* $B''(x'', y'')$):

$$\begin{aligned} x'' &= \left((x - x_u) \cdot \frac{dxn}{dxu} \right) + xn \\ y'' &= \left((y - y_u) \cdot \frac{dyn}{dyu} \right) + yn \end{aligned} \quad (65)$$

kde: $B(x,y)$ - je bod v USS
 $B''(x'',y'')$ - je bod v NSS
 x_u, y_u - sú súradnice ľavého dolného rohu okna v USS
 x_n, y_n - sú súradnice ľavého dolného rohu výrezu v NSS
 dx_u, dy_u - je rozmer výrezu v USS
 dx_n, dy_n - je rozmer výrezu v NSS

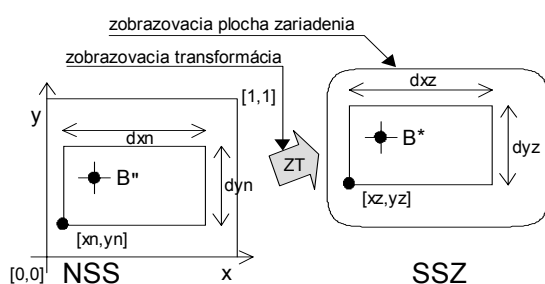


Obr. 46 Transformácia USS do NSS

Ak platí, že $\frac{dxn}{dxu} \neq \frac{dyn}{dyu}$, potom dôjde k deformácii objektu pri zobrazení. Tento princíp umožňuje zobrazovať rôzne okná do rôznych častí obrazovky t.j. do rôznych výrezov.

Ak je NSS použitá ako medzičlánok, potom sa musí ešte vykonať transformácia z NSS do SSZ. Ak chceme teda určitú časť (alebo celú) NSS zobraziť do výrezu v SSZ (pozri Obr. 47) potom platí (bod B má súradnice v NSS $B''(x'',y'')$ a v SSZ $B^*(x^*,y^*)$):

$$\begin{aligned} x^* &= xn \times dxz + xz \\ y^* &= yn \times dyz + yz \end{aligned} \quad (66)$$



Obr. 47 Transformácia NSS do SSZ

Sekundárnym problémom transformácie *okno-výrez* je problém *orezania* (*clipping*). Realizácia oboch algoritmov (zobrazovacej transformácie a orezania) býva v súčasnosti už často súčasťou technického vybavenia grafických zariadení. Toto potom umožňuje jednoducho a efektívne vykonávať napr. zväčšenie/zmenšenie okna, zmenu rozsahu používateľských súradníc tzv. *transfokáciu* (*zooming*) resp. posun okna po USS tzv. *panorámovanie* (*panning*) alebo *občerstvenie* zobrazenia obsahu okna (*refresh*).

4.4.1 Orezanie

K pojmu *orezanie*, za pomoci skôr definovaných pojmov súradnicových sústav, si pre ďalší výklad ešte definujeme nové pojmy. Najprv pojem *maximálna pracovná plocha*, ktorá predstavuje maximálne súradnice v obidvoch osiach, v rámci ktorých je možné pracovať (predstavuje teda *USS*). Pováčšine však rozsah aktuálne zobraziteľných súradníc býva menší ako maximálna pracovná plocha. Tejto ploche hovoríme aj *virtuálna* (alebo tiež *logická*) *pracovná plocha* a možnej zobraziteľnej ploche (dané napr. rozlišovacou schopnosťou zobrazovacieho zariadenia) zase *fyzická pracovná plocha* (*SSZ*). Spravidla fyzická pracovná plocha tvorí spomínané okno do virtuálnej pracovnej plochy. Vtedy sa môže stať, že časť kresby sa môže dostať mimo zobrazeného okna. Potom je nutné vykonať *orezanie*. Orezanie v praxi je možné realizovať dvomi spôsobmi:

- ❑ na úrovni grafického výstupného zariadenia, ktoré orezanie "inteligentne" vykoná pomocou vlastných technických alebo programových prostriedkov (súčasných takmer všetky).
- ❑ na úrovni vyššieho programového vybavenia, kde do zariadenia už idú informácie o orezanej kresbe.

OREZANIE ÚSEČIEK

Existuje niekoľko spôsobov ako orezanie vykonať. Najčastejšie sa používa tzv. *Cohen-Sutherlandov algoritmus*. Tento algoritmus je vhodný najmä pre orezanie úsečiek a predpokladá, že hranice zobrazovacieho okna sú rovnobežné s osami používaného súradnicového systému t.j. s *SSZ*. Tento algoritmus označuje koncové body úsečiek, ktoré sa zobrazujú pomocou kódov. Kódovanie je bitové a možné priradenie ukazuje nasledujúci obrázok.

vľavo hore 1001	hore 0001	vpravo hore 0101
vľavo 1000	Zobrazovacie okno 0000	vpravo 0100
vľavo dolu 1010	dolu 0010	vpravo dolu 0110

Obr. 48 Kódy oblastí Cohen-Sutherlandovho algoritmu

Z daného obrázku jasne vyplýva, že môžu nastať tri prípady:

1. kódy oboch koncových bodov sú nulové (prázdne) potom oba koncové body ležia vo vnútri zobrazovacieho okna a je možné úsečku vykresliť bez orezania.
2. jeden z kódov nenulový, potom je nutné orezanie, pretože časť úsečky určite leží mimo zobrazovacieho okna.
3. obidva kódy sú nenulové. Potom sú možné dva prípady:
 - ❑ celá úsečka je mimo zobrazovacieho okna a nevykreslí sa. Toto sa deje najmä ak obidva kódy dva príslušné bity rovnaké (napr. 1000 a 1010).
 - ❑ časť úsečky je v okne. Toto sa môže stať, ak obidva kódy nemajú dva príslušné bity rovnaké (napr. 1000 a 0100). Vtedy je nutné orezanie.

V prípade, že je nutné orezanie, musí sa nájsť priesečník úsečky s hranicou. Podľa kódu sa určí, s ktorou hranicou sa vyhladá priesečník. Ako príklad si uvedieme

príklad ľavej hranice. Pre určenie priesečníka potrebujeme zistiť už len príslušnú y-ovú súradnicu, pretože x-vá je daná hranicou. Potom už stačí dosadiť len do rovnice priamky, definovanej koncovými bodmi úsečky.

Nech sú teda súradnice koncových bodov (x_1, y_1) a (x_2, y_2) . Ľavá x-ová hranica nech je X_{LAVA} . Nech y_H je hľadaná y-ová súradnica priesečníka.

$$y_H = y_1 + \frac{y_2 - y_1}{x_2 - x_1} \times (X_{LAVA} - x_1) \quad (67)$$

Tento algoritmus používa pre výpočet priesečníka matematické operácie delenia a násobenia. Preto nie je vhodný celkom pre implementáciu vo VLSI grafických procesoroch. Preto sa hľadal vhodnejší postup. Algoritmus vychádzajúci z Cohen-Sutherlandovho algoritmu sa nazýva *Sproull-Sutherlandov algoritmus*. Tento na určenie priesečníka používa len iteračný spôsob delenia úsečky. Keďže obsahuje len delenie dvomi a sčítanie je vhodný práve pre implementácie tam, kde nebol vhodný Cohen-Sutherlandov algoritmus.

OREZANIE POLYGÓNOV



Pri orezávaní polygónov je možné v podstate použiť opäť *Cohen-Sutherlandov algoritmus*. Pri implementácii sa používa vlastnosť, že polygón sa vlastne skladá z úsečiek. Vytvorí sa pomocná pamäť, kde sa uložia jednotlivé úsečky, z ktorých sa polygón skladá. Potom sa vykoná klasický Cohen-Sutherlandov algoritmus, pre každú z úsečiek a v prípade orezania sa nové súradnice uložia do pomocnej pamäti. Potom sa takto orezaný polygón vykreslí. Nevýhoda takto implementovaného algoritmu je v nutnosti opakovania výpočtu (pre každý úsek) a vo vyššej pamäťovej náročnosti (pomocná pamäť). Pre menšie pamäťové nároky a pre určité možné zrýchlenie sa používa *Sutherland-Hodgmanov algoritmus*. Tento algoritmus postupne orezáva polygón podľa jednotlivých hraníc a je vhodný aj pre zretazené spracovanie (pipeline).

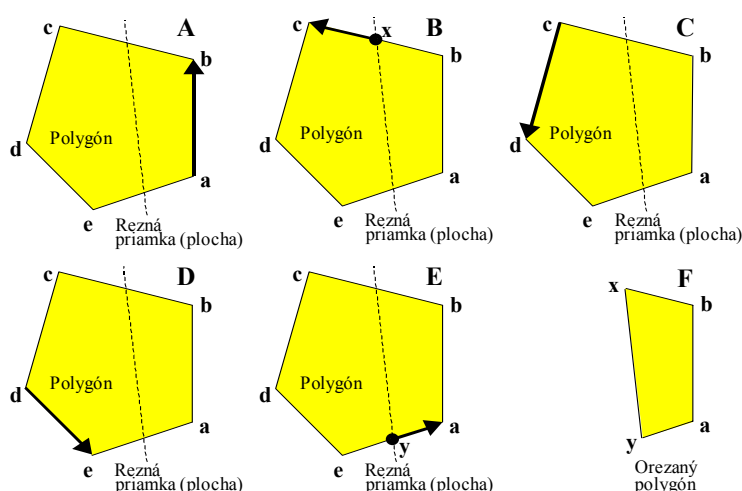
Predpokladajme konvexný polygón. V takomto prípade môžu nastať iba dva prieniky medzi reznou priamkou (v 3D reznou plochou) a hranami polygónu. Táto metóda je ilustrovaná na nasledujúcom obrázku.

V prípade, že je potrebné zachovať obidve časti rezaného polygónu, algoritmus je možné jednoducho rozšíriť. Princíp ostáva zachovaný aj pri orezávaní plochou v 3D napr. pri orezávaní na zorný ihlan (ihlan pohľadu, pozri neskôr). Formálne možno algoritmus zapísať nasledovne:

```

Začiatok(
Deklaruj    $v_1, v_2, v_3, \dots, v_n$       ako zoznam vrcholov v smere alebo proti smeru
                                                hodinových ručičiek
Deklaruj   P1                                ako prvú časť vzniknutého polygónu
           P2                                ako jeho druhú časť
Pre (i od 1 to n) vykonaj
  ( Ak (hrana z  $v_i$  do  $v_{i+1}$  pretína reznú priamku) Vyskoč z cyklu
    Koniec podmienky
  ) Opakuj cyklus
Pre (j od i to n) vykonaj
  ( Ak (hrana z  $v_j$  do  $v_{j+1}$  pretína reznú priamku) Vyskoč z cyklu
    Koniec podmienky
  ) Opakuj cyklus
Nech x = prienik hrany  $v_i-v_{i+1}$  s reznou priamkou.
Nech y = prienik hrany  $v_j-v_{j+1}$  s reznou priamkou.
Potom   ( P1 je:  $v_1, v_2, \dots, v_i, x, y, v_{j+1}, \dots, v_n$  )
        ( P2 je:  $x, v_{i+1}, v_{i+2}, \dots, v_j, y$  )
) Koniec

```



Obr. 49 Postup orezávania polygónov plochou

OREZANIE KRUHOVÝCH OBJEKTŮV

Pri orezávaní objektov kruhového typu je možné použiť podobné kódovanie ako v predošlom. Avšak je nutné určiť tri prípady:

- ❑ kruhový objekt nemá priesečník s hranicou, potom sa vykreslí celý objekt.
- ❑ kruhový objekt má jeden alebo dva, ale dotykové body. Potom sa opäť vykreslí celý objekt.
- ❑ kruhový objekt má dva priesečníky. Potom sa musí kruhový objekt rozložiť na kruhové oblúky a správne určiť štartovné a koncové body kruhových oblúkov podľa smeru vykresľovania.

4.4.2 Transfokácia

V prípade *transfokácie* (*zooming*) ide o transformáciu zmeny mierky. Zmena mierky je transformácia, pomocou ktorej sa dá meniť veľkosť objektov. Mierky môžu byť ľubovoľné kladné čísla, pričom hodnoty menšie ako 1 znižujú veľkosť a hodnoty väčšie ako 1 ju zväčšujú. Rozdiel oproti zmene mierky objektov v rámci globálnej transformácie je, že z hľadiska transformácií sa vždy bude jednať o transfokáciu celej scény t.j. všetkých zobrazených objektov až v rámci premietacej (zobrazovacej) transformácie. Tým pádom *transfokácia nie je geometrickou transformáciou a teda nemení geometriu objektov v scéne*. Podľa typu objektov v scéne (rastrové a vektorové) sa aj aplikuje transfokácia. Pri rastrových typoch sa transfokácia aplikuje na každý bod rastra. Pri vektorových typoch zase na každý objekt.

Transfokácia sa môže implementovať rôznym spôsobom:

- ❑ možnosťou priameho zadania mierky (napr. 1:100 alebo 5:1 a pod.) alebo veľkosti zväčšenia resp. zmenšenia
- ❑ možnosťou interaktívne definovať veľkosť a umiestnenie okna, v rámci ktorého bude transfokácia vykonaná. Spätný prechod sa spravidla definuje prechodom na najbližšiu definovanú mierku.
- ❑ implementáciou tzv. lupy, ktorá sa pohybuje nad pracovnou plochou. Slúži najmä na vypracovanie detailov. Spravidla sa definujú jednak pevné mierky zväčšenia a jednak veľkosti transfokačných okien.

4.4.3 Panorámovanie

Tejto možnosti sa anglicky hovorí *panning*. Poväčšine býva rozsah aktuálne zobraziteľných súradníc menší (jedná sa o SSZ) ako maximálna pracovná plocha (USS). Tejto ploche, z pohľadu panorámovania, budeme tiež hovoriť aj *virtuálna* (alebo tiež *logická*) *pracovná plocha* a možnej zobraziteľnej ploche, z pohľadu panorámovania, budeme hovoriť *fyzická pracovná plocha*. Tak, ako je vzťah medzi USS a SSZ, je aj vzťah medzi logickou a fyzickou pracovnou plochou t.j. fyzická tvorí vlastne akési okno do logickej. Vtedy sa môže stať, že časť kresby sa môže dostať mimo zobrazeného okna. Toto môže nastať aj pri nastavení určitej mierky zobrazenia pri transfokácii.

Pod panorámovaním budeme rozumieť vlastne posúvanie fyzického okna nad logickým. Podľa spôsobu posunu je možné panorámovanie implementovať nasledovne:

- *manuálne* - sa realizuje na želanie používateľa. Vtedy sú v aplikácii spravidla k dispozícii na okrajoch pracovnej plochy určité ovládacie prvky (napr. rolovacie lišty) a posúvaním posúvača rolovacej lišty alebo napr. používaním šípiek v smere želaného posunu, sa vykoná posun fyzickej plochy nad logickou.
- *automaticky* (autopanning) - sa realizuje automaticky. Vtedy sa v aplikácii musí sledovať umiestnenie kurzora a v prípade, že sa kurzor dostane do okrajovej zóny (danej nastavenou toleranciou) príslušnej strany fyzickej kresliacej plochy, potom dôjde k posunu o nastavený krok v želanom smere. Za určitých okolností sa automatické panorámovanie vypína a to vtedy ak posun plochy nie je žiaduci.

Pri implementácii panorámovania je nutné rozlišovať s akým typom objektov a scény sa bude pracovať.

- *Panorámovanie pri rastrových typoch*: v prípade rastrovej scény sa vykonáva panorámovanie pomerne jednoducho. Z logickej plochy je na fyzickú kopírovaný vlastne obdĺžnik s rozmerom daným fyzickou plochou o posun (offset) zväčša ľavého horného rohu oproti počiatku logickej plochy. Ak označíme rozmer logickej plochy LX a LY , rozmer fyzickej plochy FX a FY potom *posun (offset)* v jednotlivých osiach PX a PY bude:

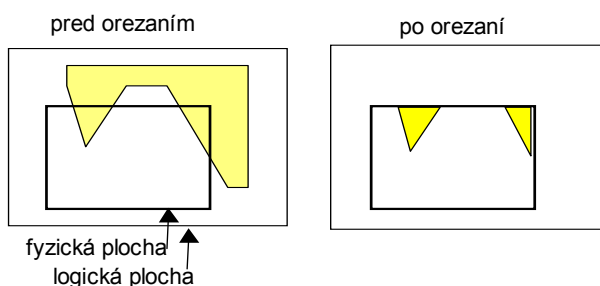
$$\begin{aligned} PX &\in \langle 0, (LX - FX) \rangle \\ PY &\in \langle 0, (LY - FY) \rangle \end{aligned} \quad (68)$$

Aj v prípade transfokácie je predchádzajúci vzťah platný, nakoľko transfokácia pracuje len s položkami FX, FY . Samozrejme, že aplikovanie sa musí vykonať po transfokácii. Keďže sa v podstate jedná o posun bloku rastra o príslušný krok, nepredstavuje tento typ v princípe žiadne implementačné riziko. Jediným úskalím je veľkosť alokovanej pamäti pre logickú pracovnú plochu, ktorá je spravidla neviditeľná.

- *Panorámovanie pri vektorových typoch*: v prípade vektorových objektov je potrebné toto starostlivo prepracovať. Aj keď platí v princípe vzťah (68), je už implementačne panorámovanie náročnejšie. Okrem toho, že sa takisto uplatní transfokácia, nesmie sa zabudnúť na orezania jednotlivých objektov. Oproti pamäťovej náročnosti u rastrovej implementácii, ktorá sa tu nevyskytuje (teda nie v zmysle nutnosti napr. udržiavať celú bitmapu v pamäti a pod.), je nutné riešiť práve spomínané problémy. Jedným z veľmi zaujímavých problémov je

Poznámky:

vyriešenie vyplnenia nekonvexnej plochy, z ktorej do fyzickej zobrazovacej plochy zasahujú dva a viac nespojitých fragmentov (pozri nasledujúci obrázok).



Obr. 50 Orežanie nekonvexného N-uholníka

Z druhej stránky má veľký význam aj rýchlosť panorámovania. Jednou zo zrýchľovacích techník je *metóda predikcie*, ktorá sa uplatní najmä pri automatickom panorámovaní. Vtedy sa musí sledovať aj predpokladaný smer pohybu kurzora. Na základe toho sa do pomocnej pamäti vypočíta zobrazenie o *krok* ďalej, ako je aktuálne zobrazenie na fyzickej ploche. V prípade, že dôjde k udalosti panorámovania, potom sa pomocné okno zobrazí a stane sa aktuálnym. Tým sa opticky okamžite dosiahne efekt posunu. Predchádzajúce aktívne okno sa stane pomocným, zneaktívni sa a zneviditeľní. Je možné zaviesť aj viackrokovú predikciu, ak je dostatok pamäti. Ak sa nezavedie žiadna zo zrýchľovacích panorámovacích techník a ani technickými prostriedkami sa to nepodporuje, potom sa posun musí vypočítat' a zobraziť v každom kroku. Najmä prekresľovanie môže spôsobovať nepríjemné "vlnenie" kresliacej plochy pre používateľa.

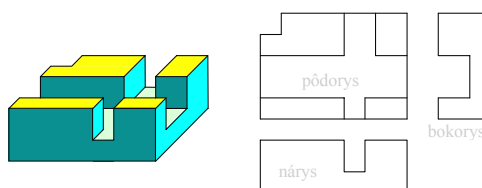
4.5 3D premietacie transformácie

Pri práci s trojrozmerným priestorom nesmieme zabúdať, že vykresľovanie bude vo väčšine prípadov do roviny (napr. obrazovka). Z tohto dôvodu sú premietacie transformácie dôležité ako základné transformácie priestoru do roviny t.j ako transformácie $USS \rightarrow SSC$ a následne $SSC \rightarrow SSZ$. Pre naše potreby si podrobnejšie uvedieme len niektoré. Medzi základné typy premietaní patria:

- ☐ kolmé
- ☐ axonometrické (axonometria)
- ☐ perspektívne (perspektíva)

4.5.1 Kolmé premietanie

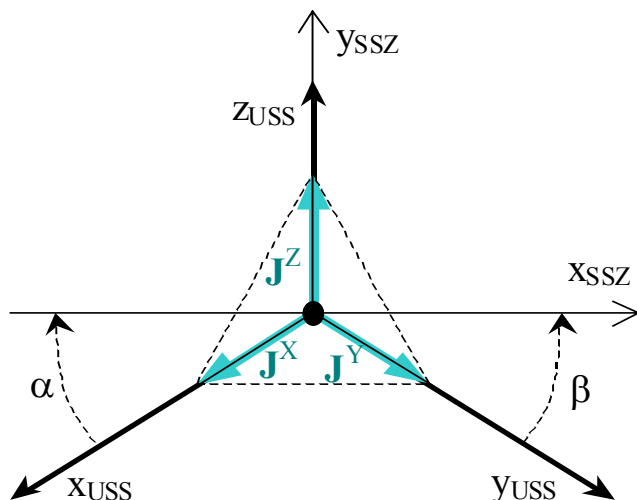
Patrí medzi typické technické premietanie vo forme *pôdorysu*, *nárys* a *bokorysu*. Pôdorys je daný rovinou XY ($z=0$), nárys je daný rovinou XZ ($y=0$) a bokorys rovinou YZ ($x=0$).



Obr. 51 Kolmé premietanie

4.5.2 Axonometria

Axonometrické premietanie sa používa pomerne často, pretože sa aj jednoducho implementuje. Hovoríme mu aj *názorný priemet*. Matematické vyjadrenie je získané z Pohlkeovej vety.



Obr. 52 Axonometrický kríž

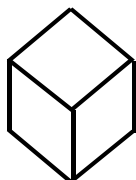
$$\begin{aligned} x_{SSZ} &= -\mathbf{J}^X \cdot \cos(\alpha) \cdot x_{USS} + \mathbf{J}^Y \cdot \cos(\beta) \cdot y_{USS} \\ y_{SSZ} &= -\mathbf{J}^X \cdot \sin(\alpha) \cdot x_{USS} - \mathbf{J}^Y \cdot \sin(\beta) \cdot y_{USS} + \mathbf{J}^Z \cdot z_{USS} \end{aligned} \quad (69)$$

kde: x_{SSZ}, y_{SSZ} sú premietacie súradnice v 2D (vypočítané súradnice v rámci SSZ)
 $\mathbf{J}^X, \mathbf{J}^Y, \mathbf{J}^Z$ sú jednotkové vektory axonometrie (najčastejšie = 1)
 α je uhol axonometrie osi x (reprezentuje odklon osi x od vodorovnej osi zobrazovacej roviny, kladný smer proti smeru hodinových ručičiek)
 β je uhol axonometrie osi y (reprezentuje odklon osi y od vodorovnej osi zobrazovacej roviny, kladný smer v smere hodinových ručičiek)

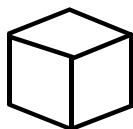
Podľa pomerov jednotkových vektorov a uhlov α a β rozoznávame nasledujúce druhy axonometrie:

- *izometria*: $\mathbf{J}^X = \mathbf{J}^Y = \mathbf{J}^Z$ a $\alpha = \beta$
- *dimetria*: $\mathbf{J}^X = \mathbf{J}^Y$ a $\alpha = \beta$
- *trimetria*: $\mathbf{J}^X \neq \mathbf{J}^Y \neq \mathbf{J}^Z$ a $\alpha \neq \beta$

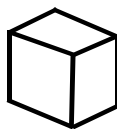
Veľmi často sa ešte používa tzv. *technická axonometria* ($\mathbf{J}^X = \mathbf{J}^Y = 1, \alpha = 45^\circ, \beta = 0^\circ$) Tomuto typu sa tiež hovorí aj *vojenská* alebo *kavalierna* axonometria.



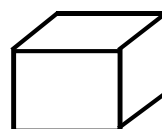
Izometria



Dimetria



Trimetria



Technická axonometria

Obr. 53 Typy axonometrie

4.5.3 Perspektíva

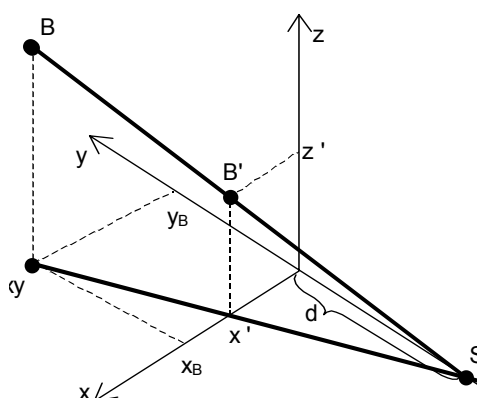
Perspektívne premietanie pôsobí najprirodzenejším dojmom. Častokrát sa toto zobrazovanie používa k zobrazeniu väčších technických celkov.

Nech rovina XZ je priemetňa a nech stred premietania S leží na osi Y . Vzdialenosť stredu premietania od priemetne budeme nazývať *dištanciou* perspektívy d . Potom platí, že $d > 0$ a súradnice stredu premietania sú $S[0, -d, 0]$. Majme opäť definovaný náš známy bod B . Označme jeho perspektívne zobrazenie B' . Toto perspektívne zobrazenie B' dostaneme ako priesečník priamky BS so zobrazovacou rovinou XZ . Označme súradnice priesečníka B' ako x', z' , čo už sú fakticky len rovinné súradnice vhodné na zobrazenie t.j. vlastne už v SSZ . Základom pre zobrazenie je zobrazovacia konštanta:

$$P = \frac{d}{d + y_B} \quad (70)$$

Samozrejme, že musí platiť aj $\frac{x'}{x_B} = \frac{z'}{z_B} = P$ (71)

Potom pre nové súradnice platí: $x' = P \cdot x_B$ (72)
 $z' = P \cdot z_B$



Obr. 54 Perspektíva

V súvislosti s týmito vzťahmi je nutné ešte definovať niektoré premietacie podmienky:

- dištanciu d je nutné voliť tak, aby ležala vo vnútri zomej kružnice.
- zorná kružnica má stred v počiatku súradnicovej sústavy a polomer rovný dištancii d .
- pre perspektívu všetkých bodov preto musí platiť:

$$x'^2 + z'^2 < \frac{d^2}{4} \quad (73)$$

potom útvary v rovinách rovnobežných s priemetňou sa v tomto premietaní len zväčšujú resp. zmenšujú.

4.6 Nelineárne premietacie transformácie

4.6.1 Rybie oko

Rybie oko sa čiastočne vymyká z rámca doteraz popisovaných transformácií. V zásade sa totiž jedná o určitý typ nelineárneho (v princípe panoramatického) zobrazenia. V princípe sa tu spája výpočtová technika, geometria a fotografia. Tento spôsob v podstate ukazuje, ako by objektív fotoaparátu zobrazil časť priestoru. Fotoaparáty známe pod menom "Rybie oko" zobrazia ešte väčšiu časť priestoru, ako obyčajné panoramatické prístroje a to celý polpriestor π pred fotoaparátom. Činnosť objektívu možno geometrizovať ako dvoje zobrazenie.

1. je priemet π polpriestoru π zo stredu O objektívu na polgul'ovú plochu η ,
 - ktorá má stred O ,
 - polomer f rovný ohniskovej vzdialenosti objektívu a
 - je obmedzená kružnicou m , v ktorej gul'ovú plochu pretína hraničná rovina polpriestoru π .
2. zobrazenie vykonáva priemet π do roviny filmu a závisí na optickom systéme objektívu.

Rovina filmu je dotykovou rovinou poglobule π v hlavnom bode H , t.j. v bode v ktorom optická os o objektívu pretína π .

Používajú sa objektívy, ktorých zobrazovacie rovnice sú :

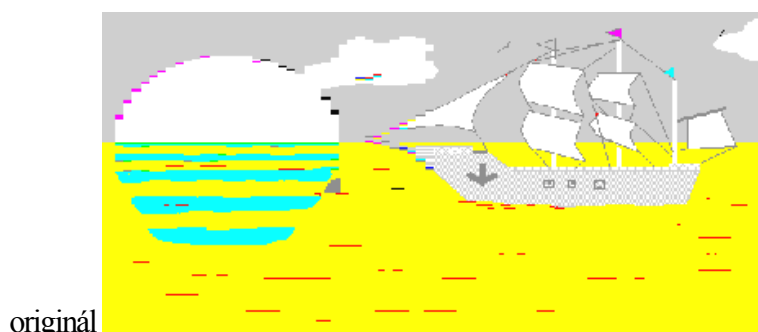
$$\begin{aligned} y &= f \cdot \sin(\beta) \\ y &= 2 \cdot f \cdot \sin\left(\frac{\beta}{2}\right) \\ y &= f \cdot \beta \end{aligned} \quad (74)$$

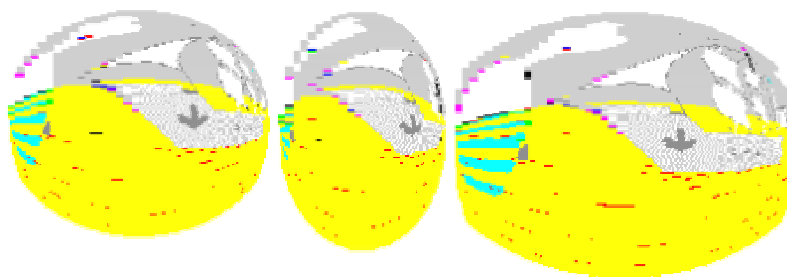
kde:

- y vzdialenosť snímku svetelného lúča a prechádzajúceho bodom O , od hlavného bodu H
- f ohnisková vzdialenosť objektívu
- β uhol, ktorý lúč a vytvára s optickou osou o objektívu.

Zobrazenia, ktoré vzniknú na základe týchto rovníc nazývame *ortografickým*, *rovnoplochým* a *ekvidistantným* zobrazením.

Nasledujúce obrázky ukazujú pôvodný obrázok a jeho zobrazenie jednotlivými typmi *rybieho oka*.





ortografické
zobrazenie

rovnoploché
zobrazenie

ekvidištantné
zobrazenie

Obr. 55 Zobrazenie pomocou „rybieho oka“



1. Vymenujte a v krátkosti popíšte súradnicové sústavy používané v počítačovej grafike
2. Charakterizujte transformačné zobrazovacie reťazce v rámci počítačovej grafiky
3. Charakterizujte geometrickú transformáciu posunutia
4. Charakterizujte geometrickú transformáciu zrkadlenia
5. Charakterizujte geometrickú transformáciu zmeny mierky
6. Charakterizujte geometrickú transformáciu skosenia
7. Charakterizujte geometrickú transformáciu otočenia
8. Charakterizujte quaterniony a ich použitie v rámci počítačovej grafiky
9. Vymenujte a v krátkosti popíšte 2D premietacie transformácie používané v počítačovej grafike
10. Popíšte princíp Cohen-Sutherlandovho algoritmu
11. Vymenujte a v krátkosti popíšte 3D premietacie transformácie používané v počítačovej grafike
12. Uveďte a popíšte aspoň jeden typ nelineárnej premietacej transformácie používanej v počítačovej grafike



V tejto kapitole sme sa naučili:

- ☐ Podľa rozmeru sa objekty v počítačovej grafike delia na jednorozmerné (1D), dvojrozmerné (2D), trojrozmerné (3D) a štvorrozmerné (4D).
- ☐ V počítačovej grafike sa používa niekoľko súradnicových sústav : USS (používateľská súr. sústava), NSS (normalizovaná súr. sústava), SSZ (súr. sústava zariadenia), SSO (súr. sústava objektu), SSC (súr. sústava kamery) a SST (súr. sústava textúry).
- ☐ V počítačovej grafike rozoznávame tri typy zobrazovacích transformácií: globálnu transformáciu, pohľadovú transformáciu a premietaciu transformáciu.
- ☐ Geometrické transformácie sa uplatňujú v rámci globálnej transformácie a môžu byť lineárne alebo nelineárne. Medzi základné lineárne geometrické transformácie patria: posunutie, zrkadlenie, zmena mierky, skosenie a otočenie.

- ☐ Pri geometrických transformáciách je výhodné používať maticové vyjadrenie a homogénne súradnice.
- ☐ Každú z transformácií je možné aplikovať na vektorový aj rastrový objekt a v 2D aj 3D priestore.
- ☐ Posunutie je transformácia, ktorá mení polohu objektu.
- ☐ Zrkadlenie je transformácia, ktorá vytvára zrkadlový obraz objektu vzhľadom na ťažisko zrkadlenia. Ťažiskom zrkadlenia môže byť objekt, ktorého rozmer je menší ako rozmer používanej súradnicovej sústavy.
- ☐ Zmena mierky je transformácia, pomocou ktorej sa dá meniť veľkosť objektu.
- ☐ Skosenie je transformácia, ktorá spôsobí deformáciu objektu vo forme vyklonenia v príslušnej osi resp. rovine.
- ☐ Transformácia objektu po kruhovej dráhe sa nazýva otočenie (rotácia)
- ☐ Pri otáčaní rastrového objektu sú možné dva postupy: otočenie s interpoláciou medziľahlých bodov a otočenie s interpoláciou všetkých bodov.
- ☐ Pri aplikácii transformačných matíc otočenia často vzniká problém s tzv. ich zablokovaním t.j., keď pokus o otočenie objektu zlyhá kvôli poradiu, v akom sa rotácie vykonávajú. Tento problém umožňuje odstrániť použitie quaterniónov.
- ☐ Quaternióny sú rozšírením komplexných čísel. Quaternióny sú asociatívne ale nie sú komutatívne.
- ☐ Pri 2D premietacích transformáciách sa najčastejšie používa orezanie, zmena mierky (transfokácia) a panorámovanie.
- ☐ Základným algoritmom pre orezanie je Cohen-Sutherlandov algoritmus. Tento algoritmus označuje koncové body úsečiek pomocou kódov a podľa nich sa volí orezanie.
- ☐ 3D premietacie transformácie rozoznávame lineárne a nelineárne. Medzi základné lineárne premietacie transformácie patria: kolmé premietanie, axonometria a perspektíva. Príkladom nelineárnej premietacej transformácie je transformácia Rybie oko.
- ☐ Kolmé premietanie patrí medzi typické technické premietanie vo forme pôdorysu, bokorysu a nárysu.
- ☐ Axonometria je definovaná axometrickým krížom (resp. axonometrickým trojuholníkom). Medzi základné parametre axonometrie patria: 3 jednotkové vektory a 2 uhly odklonu jednotlivých osí od vodorovnej osi zobrazovača. Podľa pomerov týchto parametrov rozoznávame axonometriu: izometriu, dimetriu, trimetriu a technickú axonometriu.
- ☐ Perspektíva pôsobí najprirodzenejším dojmom. Pre zobrazenie sa používa podobnosť trojuholníkov. Vzdialenosť stredu premietania od priemetne sa nazýva dištancia perspektívy.
- ☐ Podľa použitej rovnice objektívu poznáme nasledujúce typy Rybieho oka: ortografické, rovnoploché a ekvidištantné.



5 Krivky používané v počítačovej grafike

Cieľ: Študent po absolvovaní tejto kapitoly by mal mať schopnosť klasifikovať rôzne typy kriviek používaných v rámci počítačovej grafiky a oblasť ich použitia. Ďalej by mal získať prehľad o základných typoch kriviek z každej dôležitej kategórie a navyše teoretickú bázu pre implementáciu jednej najviac používanej krivky z každého typu (Fergusonova krivka a Beziéova kubická krivka).

Každá krivka musí mať svoj matematický popis, aby bolo možné vykonávať rôzne operácie s takouto krivkou alebo s jej časťou, ako sú napríklad posuny, rotácie, alebo zmena mierky.

Vzhľadom na to, ako sú rovinné krivky zadane a z hľadiska počítačovej grafiky rozdelíme krivky na:

- *krivky dané analytickým popisom*
- *interpolačné krivky*
- *aproximačné krivky*
- *interaktívne vytvárané krivky*

Skôr však ako sa dostaneme k samotnému popisu jednotlivých typov kriviek je nutné poznamenať, že snád' najväčšie ich uplatnenie je práve v systémoch CAD, kde je možné pomocou týchto kriviek naprojektovať rôzne tvary, ale s jedným veľkým plus. Tým je znalosť matematického vyjadrenie krivky, čo umožňuje napr. ľahko vypočítať dĺžku krivky, jednoducho ju tvarovať príp. vyrátať obsah plochy, ktorú potencionálne krivka uzatvára.

V publikácii budú uvedené (podobne ako plochy) len niektoré, nakoľko typov kriviek (aj plôch) a ich modifikácií je v súčasnej počítačovej grafike používaných niekoľko.

5.1 Krivky dané analytickým popisom

Ak je pre krivku daný analytický predpis, je známa jej rovnica, je ľahké realizovať jej grafický výstup. Pri rovinných krivkách sa najčastejšie používa parametrické vyjadrenie krivky:

$$\left. \begin{array}{l} x = f(t) \\ y = g(t) \end{array} \right\} t \in \langle \text{zac}, \text{kon} \rangle, \text{ kde } t \text{ je parameter funkcie.} \quad (75)$$

Realizovať grafický výstup takto uvedenej krivky je jednoduché. Stačí za t , ako parameter, dosadzovať hodnoty z jej definičného oboru a hodnoty x, y sú už vypočítané súradnice danej krivky pre parameter t . Takto vypočítané súradnice sú spravidla v USS. Pre zobrazenie je nutné vykonať ešte transformáciu USS \rightarrow SSZ. Práve u kriviek sa s výhodou používa aj normalizovaná súradnicová sústava NSS.

Napríklad rovnica pre úsečku je:

$$\left. \begin{array}{l} x = x_A + t \cdot (x_B - x_A) \\ y = y_A + t \cdot (y_B - y_A) \end{array} \right\} t \in \langle \text{zac}, \text{kon} \rangle \quad (76)$$

kde x_A, y_A sú súradnice začiatku, x_B, y_B sú súradnice konca úsečky,

alebo rovnica pre elipsu je:

$$\left. \begin{aligned} x &= x_s + a \cdot \cos(t) \\ y &= y_s + b \cdot \sin(t) \end{aligned} \right\} \quad t \in \langle \text{zac}, \text{kon} \rangle \quad (77)$$

kde x_s, y_s sú súradnice stredu, a, b sú poloosi elipsy, zac, kon sú začiatkový a koncový uhol elipsy uvedený v radiánoch.

5.2 Interpoláčn  krivky

Nech $X_1, X_2, X_3, \dots, X_n$ s  body v rovine. Interpol nou krivkou pre tieto body rozumieme krivku, ktor  prech dza t mito bodmi. Body $X_1, X_2, X_3, \dots, X_n$ budeme naz vať oporn  body. V tejto skupine si uvedieme tieto krivky:

- *krivka vytvoren  Lagrangeovou interpol ciou*
- *Fergusonova krivka*
- *krivka vytvoren  Akimovskou interpol ciou*
- *spline krivka*

5.2.1 Lagrangeova interpol cia



U Lagrangeovej met dy je krivka dan  bodmi $[x_i, y_i]$, $i=0..n$, pri om postupnos  $\{x_i\}$, $i=0..n$ tvor  rast cu postupnos . Krivka je dan  funkciou:

$$L(x) = \sum_{k=0}^n y_k \cdot \frac{\prod_{j=0, j \neq k}^n (x - x_j)}{\prod_{j=0, j \neq k}^n (x_k - x_j)} \quad (78)$$

Teda je to mnoho len stup ňa n . Nev hodou tejto funkcie pre interpol ciu je,  e pri v     ch po  toch oporn ch bodov sa v  po ty komplikuj  a krivka sa ne iad co oh ba. V hodou je v ak to,  e v celom defini nom obore m  krivka (funkcia) spojite deriv cie v etk ch r dov (d le it  najm  pre hladk  nav zovanie jednotliv ch segmentov).

5.2.2 Fergusonova krivka

Nech s  dan  polohov  vektory **G** a **H** bodov G a H a smerov  vektory **g** a **h** doty n c v t chto bodoch. Potom Fergusonova krivka je dan  rovnicou:

$$\mathbf{P}(v) = \mathbf{m} \cdot v^3 + \mathbf{n} \cdot v^2 + \mathbf{p} \cdot v + \mathbf{q} \quad (79)$$

kde $\mathbf{P}(v)$ je polohov  vektor bodu krivky, **m**, **n**, **p** a **q** s  vektory koeficientov a v je parameter tak ,  e $\mathbf{P}(0) = \mathbf{G}$ a $\mathbf{P}(1) = \mathbf{H}$.

Pr slu n m v  po tom dostaneme vyjadrenie vektorov **m**, **n**, **p**, **q**, teda:

$$\begin{aligned} \mathbf{m} &= 2 \cdot \mathbf{G} - 2 \cdot \mathbf{H} + \mathbf{g} + \mathbf{h} \\ \mathbf{n} &= -3 \cdot \mathbf{G} + 3 \cdot \mathbf{H} - 2 \cdot \mathbf{g} - \mathbf{h} \\ \mathbf{p} &= \mathbf{g} \\ \mathbf{q} &= \mathbf{G} \end{aligned} \quad (80)$$

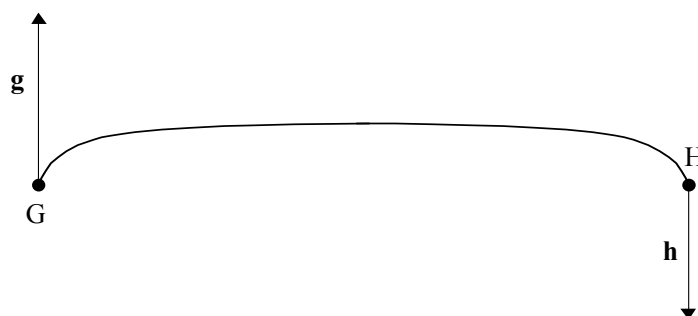
Fergusonovú krivku môžeme vyjadriť aj v tvare:

$$\mathbf{P}(v) = A(v) \cdot \mathbf{G} + B(v) \cdot \mathbf{H} + C(v) \cdot \mathbf{g} + D(v) \cdot \mathbf{h} \quad (81)$$

kde $A(v)$, $B(v)$, $C(v)$, $D(v)$ sú polynómy tretieho stupňa, pre ktoré platí:

$$\begin{aligned} A(v) &= 2 \cdot v^3 - 3 \cdot v^2 + 1 \\ B(v) &= -2 \cdot v^3 + 3 \cdot v^2 \\ C(v) &= v^3 - 2 \cdot v^2 + v \\ D(v) &= v^3 - v^2 \end{aligned} \quad (82)$$

Ak v týchto vzťahoch volíme v z intervalu $\langle 0, 1 \rangle$, tak dostaneme krivku začínajúcu v bode G a končiacu v bode H.



Obr. 56 Fergusonova krivka

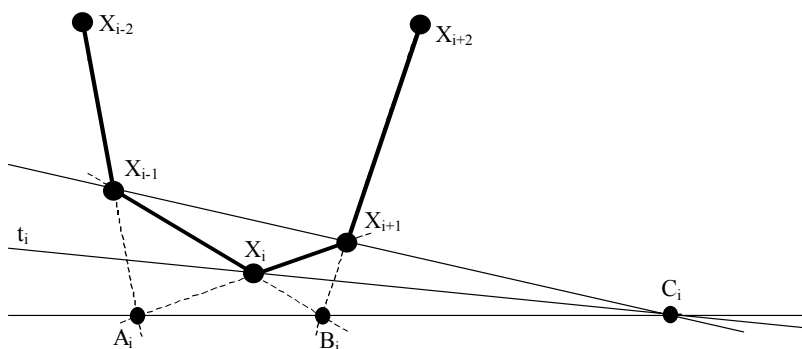
5.2.3 Akimovská interpolácia



Akimovská interpolácia pracuje s rovinnými Fergusonovými krivkami, smerové vektory sa určujú automaticky z konfigurácie oporných bodov. Jedna z možností je nasledujúca.

Smerový vektor so smernicou t_i v opornom bode X_i je určený bodmi X_i a C_i . Bod C_i je prienikom priamky vytvorenej bodmi X_{i-1} a X_{i+1} a priamkou vytvorenou bodmi A_i a B_i . Bod A_i je vytvorený prienikom priamky určenej bodmi X_{i-2} a X_{i-1} a priamkou určenou bodmi X_i a X_{i+1} . Bod B_i je určený prienikom priamky určenej bodmi X_{i-1} a X_i a priamkou určenou bodmi X_{i+1} a X_{i+2} .

Z uvedeného je zrejmé, že pokiaľ tri po sebe idúce oporné body X_{i-1} , X_i , X_{i+1} ležia na priamke, sú smerové vektory v bodoch X_{i-1} , X_i , X_{i+1} totožné s priamkou určenou bodmi X_{i-1} a X_{i+1} a z toho vyplýva, že oblúky určené bodmi X_{i-1} , X_i a X_i , X_{i+1} sú úsečky.



Obr. 57 Akimovská interpolácia

5.2.4 Spline krivka



Akimovská interpolácia nezaist'uje spojitú zmenu krivosti pozdĺž krivky. V niektorých technických aplikáciách je požiadavka spojit' zmeny krivosti dôležitá. Pri použití kubických funkcií túto vlastnosť majú spline krivky. Najprv si uvedieme definíciu spline funkcie.

Spline funkciou stupňa m pre daných $n+1$ bodov $X_i = (x_i, y_i)$, $i = 0..n$, $x_0 < x_1 < \dots < x_n$, nazývame funkciu $f(x)$, pre ktorú na intervale $\langle x_0, x_n \rangle$ platí:

- a) $f(x) = f_k(x)$ na intervale $\langle x_k, x_{k+1} \rangle$, kde f_k je polynóm stupňa m ,
- b) $f(x)$ má spojité derivácie $f^{(0)}, f^{(1)}, \dots, f^{(m-1)}$.

Najčastejšie sa používajú kubické spline funkcie ($m=3$), ktoré sú matematickým modelom pravička používaného lodiarskymi konštruktérmi. Často sa používa aj tzv. prirodzená spline funkcia, tiež kubická funkcia pre nulové hodnoty druhých derivácií v bodoch X_0 a X_n .

Pre krivky sa používa parametrická spline interpolácia, pričom každá zo zložiek je spline funkcia ako parameter. Platí:

1. Kubická spline funkcia je jednoznačne určená opornými bodmi a hodnotami prvých derivácií v bodoch X_0 a X_n .
2. Kubická spline funkcia je jednoznačne určená opornými bodmi a hodnotami druhých derivácií v bodoch X_0 a X_n .
3. Uzavretá spline krivka je jednoznačne určená opornými bodmi.

Ako bolo povedané je spline funkcia vlastne zložená z polynómov. Z bodov, cez ktoré má daná spline funkcia prechádzať a z podmienok spojitosti prvej a druhej derivácie dostaneme rovnice, z ktorých vieme vypočítať neznáme koeficienty výsledného polynómu.

Problematika spline kriviek je jednak pomerne rozsiahla a jednak patrí skôr do záujmovej oblasti matematiky. Z tohto dôvodu sme si tu uviedli len základné pojmy. V súčasných, najmä CAD systémoch sa vo veľkej miere používajú okrem iného β -spline, γ -spline a kv -spline. Vo všeobecnosti tieto krivky tu riešia problém tzv. geometrickej nadväznosti a spojitosti.

5.3 Interaktívne vytvárané krivky

Každá z interpolačných metód má isté výhody i nevýhody. Pomocou prostriedkov interaktívnej grafiky je možné, aby sa konštruktér spoluúčastnil vytvárania vhodnej krivky. Beziérová myšlienka vytvárania kriviek daného tvaru pomocou riadiaceho polygónu je základom teórie Beziérových kriviek a B-spline kriviek. Takže v tejto časti si uvedieme dve krivky:

- Beziérové krivky
- B-spline krivky

5.3.1 Beziérové krivky

$$\text{Nech } \left\{ \mathbf{P}_i \right\}_{i=0}^m, \quad m > 0 \quad (83)$$

sú polohové vektory vrcholov polygónu, ktorým sa určuje krivka. Beziérová krivka pre daný polygón je daná rovnicou:

$$\mathbf{R}(t) = \sum_{i=0}^m \mathbf{P}_i \cdot Be_{im}(t) \quad (84)$$

kde parameter $t \in \langle 0, 1 \rangle$, $\mathbf{R}(t)$ je polohový vektor bodu krivky a $Be_{im}(t)$ sú Bernsteinove polynómy, t. j.

$$Be_{im}(t) = \sum \binom{m}{i} \cdot t^i \cdot (1-t)^{m-i} \quad (85)$$

Za predpokladu, že $\binom{m}{0} = 1$ a $0^0 = 1$, určíme počiatočný a koncový bod oblúku

výslednej krivky. K výpočtu $\mathbf{R}(0)$ určíme, že $Be_{0m}(0) = 1$ a $Be_{im}(0) = 0$ pre $i = 1..m$. Vtedy $\mathbf{R}(0) = \mathbf{P}_0$. Pretože $Be_{im}(1) = 0$ pre $i = 0..m-1$ a $Be_{mm}(1) = 1$, dostávame, že $\mathbf{R}(1) = \mathbf{P}_m$.

Tým sme ukázali, že počiatočným a koncovým bodom Beziérovej krivky je počiatočný a koncový bod zadávajúceho polygónu.

Určíme aj smerové vektory Beziérovej krivky v počiatočnom a koncovom bode.

$$\mathbf{R}^{(1)}(0) = \sum_{i=0}^m \mathbf{P}_i \cdot Be_{im}^{(1)}(0) \quad (86)$$

$$\begin{aligned} Be_{0m}^{(1)}(0) &= \left[(1-t)^m \right]_{t=0}^{(1)} = -m \\ Be_{1m}^{(1)}(0) &= \left[m \cdot (1-t)^{m-1} \right]_{t=0}^{(1)} = m \\ Be_{im}^{(1)}(0) &= 0 \quad \text{pre } i = 2 \dots m \end{aligned} \quad (87)$$

plati $\mathbf{R}^{(1)}(0) = m \cdot (\mathbf{P}_1 - \mathbf{P}_0)$

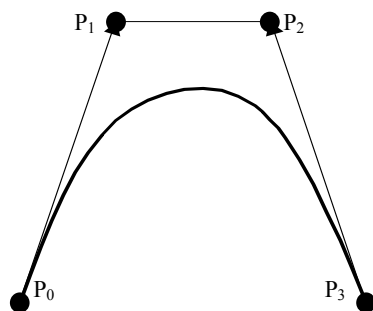
Tým sme ukázali, že strana P_0P_1 je dotyčnicou Beziérovej krivky v počiatočnom bode. Podobne sa dá zistiť, že dotyčnicou Beziérovej krivky v koncovom bode je strana P_mP_{m-1} riadiaceho polygónu.

Veľmi často používanou Beziérovou krivkou je Beziérová krivka 4-ho rádu alebo tiež Beziérová kubika, ktorá je určená štyrmi bodmi P_0-P_3 . Táto je určená vzťahom:

$$\mathbf{P}(t) = \mathbf{P}_0 \cdot B_0(t) + \mathbf{P}_1 \cdot B_1(t) + \mathbf{P}_2 \cdot B_2(t) + \mathbf{P}_3 \cdot B_3(t) \quad (88)$$

Kde $t \in \langle 0, 1 \rangle$ a B_0, B_1, B_2, B_3 sú kubické polynómy:

$$\begin{aligned} B_0(t) &= (1-t)^3 \\ B_1(t) &= 3 \cdot t \cdot (1-t)^2 \\ B_2(t) &= 3 \cdot t^2 \cdot (1-t) \\ B_3(t) &= t^3 \end{aligned} \quad (89)$$



Obr. 58 Beziérová krivka

Dve Beziérové krivky budú na seba nadväzovať ak sa zaistí identičnosť posledného bodu jednej krivky s prvým bodom krivky druhej. Navyše dve Beziérové krivky budú nadväzovať hladko na seba ak ešte navyše budú identické dotyčnicové vektory v koncovom bode prvej a štartovnom bode druhej krivky. Tieto vektory sú jednoznačne dané poslednými dvomi bodmi prvej krivky resp. prvými dvoma bodmi druhej krivky. Ak teda platí, že posledný bod prvej krivky je identický s prvým bodom druhej krivky, potom druhý bod druhej krivky je pre hladké nadväzovanie poslednými dvomi bodmi prvej krivky. Tento sa musí nachádzať na priamke, ktorá je daná týmito dvomi bodmi.

5.3.2 B-spline krivky



B-spline krivky sú zovšeobecnením Beziérových kriviek, miesto Bernsteinových polynómov sa používajú jednoduchšie funkcie. Uvedieme len veľmi špeciálnu B-spline krivku, tzv. kubický Coonsov B-spline.

Kubický Coonsov B-spline je určený vektorovou rovnicou:

$$\mathbf{R}(t) = \sum_{i=0}^3 \mathbf{P}_i \cdot Co_i(t), \quad t \in \langle 0,1 \rangle \quad (90)$$

kde

$$\begin{aligned} Co_0(t) &= -\frac{1}{6} \cdot t^3 + \frac{1}{2} \cdot t^2 - \frac{1}{2} \cdot t + \frac{1}{6} \\ Co_1(t) &= \frac{1}{2} \cdot t^3 - t^2 + \frac{2}{3} \\ Co_2(t) &= -\frac{1}{2} \cdot t^3 + \frac{1}{2} \cdot t^2 + \frac{1}{2} \cdot t + \frac{1}{6} \\ Co_3(t) &= \frac{1}{6} \cdot t^3 \end{aligned} \quad (91)$$

Určíme počiatkový a koncový bod oblúka B-spline krivky. Pre $t=0$ dostaneme:

$$\begin{aligned} Co_0(0) &= \frac{1}{6} \\ Co_1(0) &= \frac{2}{3} \\ Co_2(0) &= \frac{1}{6} \\ Co_3(0) &= 0 \end{aligned} \quad (92)$$

Poznámky:

,a teda

$$\mathbf{R}(0) = \frac{1}{6} \cdot (\mathbf{P}_0 + \mathbf{P}_2) + \frac{2}{3} \cdot \mathbf{P}_1 = \frac{1}{3} \cdot \left[\frac{1}{2} \cdot (\mathbf{P}_0 + \mathbf{P}_2) - \mathbf{P}_1 \right] + \mathbf{P}_1 \quad (93)$$

Počiatočným bodom oblúka B-spline krivky je bod $\mathbf{R}(0)$, antiŕažisko trojuholníka $\mathbf{P}_0, \mathbf{P}_1, \mathbf{P}_2$. Bod $\mathbf{R}(0)$ leží v $1/3$ dĺžky ŕažnice od bodu \mathbf{P}_1 . Pre $t=1$ platí:

$$\begin{aligned} Co_0(1) &= 0 \\ Co_1(1) &= \frac{1}{6} \\ Co_2(1) &= \frac{2}{3} \\ Co_3(1) &= \frac{1}{6} \end{aligned} \quad (94)$$

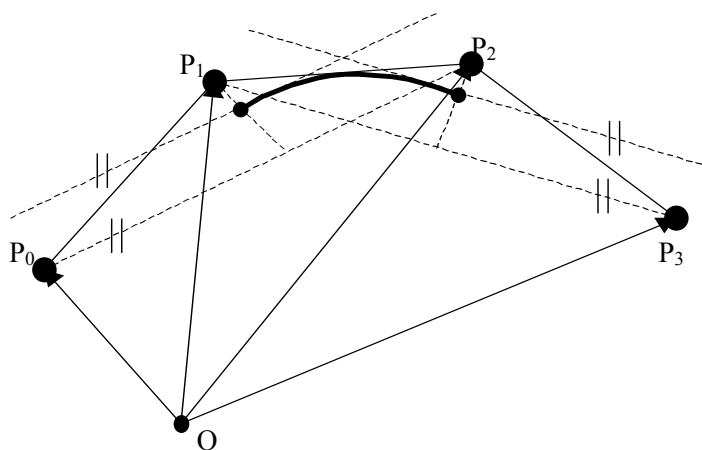
potom

$$\mathbf{R}(1) = \frac{1}{6} \cdot (\mathbf{P}_1 + \mathbf{P}_3) + \frac{2}{3} \cdot \mathbf{P}_2 = \frac{1}{3} \cdot \left[\frac{1}{2} \cdot (\mathbf{P}_1 + \mathbf{P}_3) - \mathbf{P}_2 \right] + \mathbf{P}_2 \quad (95)$$

Koncovým bodom oblúka je bod $\mathbf{R}(1)$, antiŕažisko trojuholníka $\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3$. Bod $\mathbf{R}(1)$ leží v $1/3$ dĺžky ŕažnice od bodu \mathbf{P}_2 .

Podobne vieme vypočítať aj dotyčnice v koncových bodoch :

$$\begin{aligned} \mathbf{R}^{(1)}(0) &= \frac{1}{2} \cdot (\mathbf{P}_2 - \mathbf{P}_0) \\ \mathbf{R}^{(1)}(1) &= \frac{1}{2} \cdot (\mathbf{P}_3 - \mathbf{P}_1) \end{aligned} \quad (96)$$



Obr. 59 B-spline krivka

Ak porovnáme Beziérové krivky a B-spline krivky pre daný polygón $\{\mathbf{P}_i\}$, $i=0..m$, Beziérová krivka je stupňa m , B-spline krivka je zložená z $m-1$ oblúkov (prvý oblúk je určený bodmi $\mathbf{P}_0, \mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3$ druhý oblúk bodmi $\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3, \mathbf{P}_4$ atď.), z ktorých každý je popísaný kubickou rovnicou. Nevýhodou B-spline kriviek je, že krivka nezačína a nekončí v začiatočnom a koncovom bode polygónu. Túto nevýhodu možno odstrániť tak, že použijeme náhradný polygón, alebo začiatočný a koncový bod krivky zdvojnásobíme.

1. Charakterizujte krivky používané v počítačovej grafike
2. Charakterizujte a popíšte Fergusonovu krivku
3. Charakterizujte a popíšte Beziérové krivky
4. Charakterizujte a popíšte B-spline krivky



V tejto kapitole sme sa naučili:



- ☐ Krivky v počítačovej grafike sa využívajú najmä v systémoch CAD a pri voľnej grafickej tvorbe.
- ☐ Podľa toho, ako sú krivky zadané, ich rozdeľujeme na: krivky dané analytickým popisom, interpolačné krivky, aproximačné krivky a interaktívne vytvárané krivky.
- ☐ Ak je pre krivku daný analytický predpis, je známa jej rovnica, je ľahké realizovať jej grafický výstup. Najčastejšie sa používa parametrické vyjadrenie krivky.
- ☐ Medzi interpolačné krivky môžeme zaradiť napr.: krivku vytvorenú Lagrangeovou interpoláciou, Fergusonovu krivku, krivku vytvorenú Akimovskou interpoláciou a spline krivky.
- ☐ Jednou z najčastejšie používaných interpolačných kriviek je Fergusonova krivka. Táto je definovaná štartovným a koncovým bodom a štartovným a koncovým radiacim vektorom, ktorý tvorí v príslušnom bode dotyčnicu krivky. Tvar krivky sa modeluje veľkosťou a smerom radiacích vektorov.
- ☐ Medzi najviac používané interaktívne vytvárané krivky patria: Beziérová krivka a B-spline krivka.
- ☐ Beziérová krivka je definovaná štartovným a koncovým bodom a množinou radiacích bodov, ktorými krivka neprechádza (iba ich aproximuje). Tvar krivky sa modeluje polohou radiacích bodov. Beziérová krivka je citlivejšia na zmenu radiaceho bodu ako Fergusonova krivka na zmenu radiaceho vektora.
- ☐ B-spline krivka je zovšeobecnením Beziérovej krivky. Jej výhodou je väčšia modelovacia schopnosť. Jej nevýhodou je, že krivka nezačína a nekončí v začiatočnom a koncovom bode radiaceho polygónu. Túto nevýhodu je možné odstrániť zdvojením začiatočného a koncového bodu v radiacom polygóne.



6 Plochy používané v počítačovej grafike

Ciel': Absolvovaním tejto kapitoly bude študent schopný klasifikovať rôzne typy plôch používaných v rámci počítačovej grafiky a oblast' ich použitia resp. aplikácie. Ďalej by mal získať prehľad o základných typoch plôch používaných v počítačovej grafike a tiež dostatočný teoretický základ nutný na implementáciu niektorých typov plôch (Bilinárna plocha a Beziérová bikubická plocha).

Každá plocha musí mať svoj matematický popis, aby bolo možné vykonávať rôzne operácie s takouto plochou alebo s jej časťou, ako sú napríklad posuny, rotácie, alebo zmena mierky. Pritom sa vždy ráta len s určitými ťažiskovými bodmi plochy.

Obdobne ako u kriviek je nutné poznamenať, že ich uplatnenie je tiež práve v systémoch CAD, kde je možné pomocou týchto plôch naprojektovať rôzne tvary s rovnakou výhodou ako u kriviek. Tou je znalosť matematického vyjadrenie plochy, čo umožňuje napr. obdobne ľahko vypočítať obsah plochy, jednoducho ju tvarovať príp. vyrátať objem priestoru, ktorú potencionálne plocha obopína.

V publikácii budú uvedené (podobne ako krivky) len niektoré, nakoľko typov plôch a ich modifikácií je v súčasnej počítačovej grafike používaných niekoľko. Pre vkročenie do ich problematiky však uvedené typy postačia.

6.1 Plochy dané analytickým popisom

Pre analytický popis plochy platia obdobné pravidlá ako pre analytický popis kriviek v 2D. Ak je pre plochu daný analytický predpis, je známa jej rovnica, je ľahké dostať jej grafický výstup. Pri plochách sa najčastejšie používa parametrické vyjadrenie plochy:

$$\left. \begin{aligned} x &= f(t) \\ y &= g(t) \\ z &= h(t) \end{aligned} \right\} t \in \langle zac, kon \rangle, \text{ kde } t \text{ je parameter funkcie.} \quad (97)$$

Realizovať grafický výstup takto uvedenej plochy je jednoduché. Stačí za t , ako parameter, dosadzovať hodnoty z jeho definičného oboru a hodnoty x, y, z sú už vypočítané súradnice ťažiskových bodov danej plochy pre parameter t . K vykresleniu postačuje už len pretransformovať 3D-súradnice na 2D t.j. priemet a vykonať lineárnu interpoláciu s predchádzajúcim ťažiskovým bodom. Pri základnom kreslení nie je nutné riešiť viditeľnosť, prípadne ju riešiť jednoduchým spôsobom vykresľovania odzadu dopredu.

6.2 Coonsove plochy

Patria medzi klasické plochy používané v počítačovej grafike. Coonsove plochy majú vlastnosti vhodné pre použitie vo výpočtovej technike: jednoduchú teóriu, nezvlnený tvar (minimum konvexných a konkávných častí) a jednoduchú možnosť implementácie na počítači. V praxi tvoria tieto plochy širokú triedu plôch, my si však uvedieme len najpoužívanejšie z nich.

6.2.1 Bilineárna Coonsova plocha

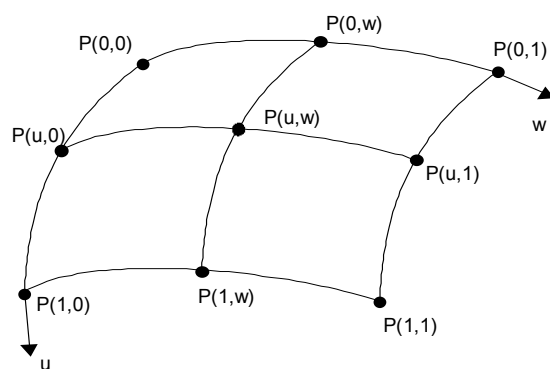
Je jednou z najjednoduchších typov Coonsových plôch. Základnou rovnicou popisujúcou túto plochu je:

$$[1-u, -1, u] * \mathbf{M} * [1-w, -1, w]^T = 0 \quad (98)$$

kde u a w sú parametre, pretože bilineárny plát je určený okrajom $P(u,0)$, $P(u,1)$, $P(0,w)$ a $P(1,w)$ ako ukazuje Obr. 60

Matica \mathbf{M} je štvorcová matica polohových vektorov a prvky v nej sú na rovnakých miestach ako na ploche.

$$\mathbf{M} = \begin{bmatrix} \mathbf{P}(0,0) & \mathbf{P}(0,w) & \mathbf{P}(0,1) \\ \mathbf{P}(u,0) & \mathbf{P}(u,w) & \mathbf{P}(u,1) \\ \mathbf{P}(1,0) & \mathbf{P}(1,w) & \mathbf{P}(1,1) \end{bmatrix} \quad (99)$$



Obr. 60 Bilineárna Coonsova plocha

Bod $P(u,w)$ resp. jeho polohový vektor uprostred matice \mathbf{M} je hľadaným riešením implicitnej rovnice pre určité hodnoty parametrov u a w a je to bod ležiaci na výslednej ploche. Pre lepšiu implementovateľnosť sa tento tvar prevedie na explicitný, čím dostaneme:

$$P(u,w) = [1-u, u] * [P(0,w), P(1,w)]^T + [P(u,0), P(u,1)] * [1-w, w]^T - [1-u, u] * Q * [1-w, w]$$

kde \mathbf{Q} je matica

$$\mathbf{Q} = \begin{bmatrix} \mathbf{P}(0,0) & \mathbf{P}(0,1) \\ \mathbf{P}(1,0) & \mathbf{P}(1,1) \end{bmatrix} \quad (100)$$

Pokiaľ protiahle strany budú úsečky, dostaneme tzv. *priamkovú (pravítkovú) plochu*. Coonsova bilineárna plocha je teda všeobecnejšia ako plocha priamková.

6.2.2 Bikubická Coonsova plocha



Je ďalším typom Coonsových plôch a predstavuje rozšírenie bilineárnej plochy, ale jej zadávanie a tvar je rovnaké. Rovnica popisujúca túto plochu je modifikáciou rovnice pre bilineárnu plochu.

$$[F_1(u), -1, F_2(u)] * \mathbf{M} * [F_1(w), -1, F_2(w)]^T = 0 \quad (101)$$

Kde \mathbf{M} je matica ako u bilineárnej plochy a F_1, F_2 sú známe Fergusonove polynómy:

$$\begin{aligned} F_1(t) &= 2 \cdot t^3 - 3 \cdot t^2 + 1 \\ F_2(t) &= -2 \cdot t^3 + 3 \cdot t^2 \end{aligned} \quad (102)$$

Pre implementovateľnosť túto rovnicu upravíme a dostaneme analogickú explicitnú rovnicu ako v prípade bilineárnej plochy.

Základným problémom Coonsových plôch tohto je veľmi ťažké vyjadrenie priamych dotyčnicových vektorov, či sa veľmi ťažko dosahuje hladké spojenie dvoch Coonsových plátov. Z tohto dôvodu sa Coonsove plochy tohto typu nepoužívajú na hladké spojenia.

6.2.3 Všeobecná Coonsova plocha



Doteraz uvedené typy mali jednu nevýhodu (ako už bolo spomenuté) a to neuvažoval sa priebeh priečných derivácií plochy pozdĺž okrajov. Možnosť konštruovať plochu s ohľadom na tieto priečne derivácie má význam pri tzv. plátovaní tzn. spajovaní plátov. Najčastejšou požiadavkou pri plátovaní je, aby pozdĺž spoločnej okrajovej krivky mali obe plochy (pláty) aj rovnaké priečne derivácie, čím sa dosiahne hladký prechod. Túto vlastnosť má všeobecná Coonsova plocha.

Táto plocha je určená okrajom, priečnymi deriváciami pozdĺž okraja a zmiešanými deriváciami v rohoch. Potom rovnica všeobecnej Coonsovej plochy je:

$$\mathbf{U} * \mathbf{C} * \mathbf{W} = 0 \quad (103)$$

kde

$$\begin{aligned} \mathbf{U} &= [A(u), -1, B(u), C(u), D(u)] \\ \mathbf{W} &= [A(w), -1, B(w), C(w), D(w)] \end{aligned} \quad (104)$$

a matica \mathbf{C} je nasledujúca:

$$\mathbf{C} = \begin{bmatrix} \mathbf{P}(0,0) & \mathbf{P}(0,w) & \mathbf{P}(0,1) & \mathbf{P}_w(0,0) & \mathbf{P}_w(0,1) \\ \mathbf{P}(u,0) & \mathbf{P}(u,w) & \mathbf{P}(u,1) & \mathbf{P}_w(u,0) & \mathbf{P}_w(u,1) \\ \mathbf{P}(1,0) & \mathbf{P}(1,w) & \mathbf{P}(1,1) & \mathbf{P}_w(1,0) & \mathbf{P}_w(1,1) \\ \mathbf{P}_u(0,0) & \mathbf{P}_u(0,w) & \mathbf{P}_u(0,1) & \mathbf{P}_{uw}(0,0) & \mathbf{P}_{uw}(0,1) \\ \mathbf{P}_u(1,0) & \mathbf{P}_u(1,w) & \mathbf{P}_u(1,1) & \mathbf{P}_{uw}(1,0) & \mathbf{P}_{uw}(1,1) \end{bmatrix} \quad (105)$$

Maticu \mathbf{C} môžeme rozdeliť do štyroch častí (z prava doľava a zhora nadol) na časť so známou maticou \mathbf{M} , časť dotyčnicových vektorov smerom w , časť dotyčnicových vektorov smerom u a časť rohových skrutov. Čiže prvok

- $\mathbf{P}_w(0,0)$ je dotyčnicový vektor v bode $(0,0)$ ku krivke $\mathbf{P}(0,w)$.
- $\mathbf{P}_u(0,0)$ je dotyčnicový vektor v bode $(0,0)$ ku krivke $\mathbf{P}(u,0)$.
- $\mathbf{P}_{uw}(0,0)$ je vektor určený zmiešanou deriváciou v bode $(0,0)$, *skrut* (twist).

6.3 Beziérové plochy

V šesťdesiatych rokoch sa používali v počítačovej grafike najmä už uvedené Coonsove plochy. Avšak s príchodom interaktívnych grafických pracovísk sa rozmohol nový štýl konštruovania, interaktívneho konštruovania na displeji. Prof. Beziér prišiel s novou metódou, ktorú použil pri konštruovaní karosérií áut značky Renault. Oproti skôr popisovaným plochám sa u Beziéra plocha neurčuje matematicky rovnicou, ale geometricky, pomocou priestorovej siete zadanej uzlami (bodmi, vrcholmi) tejto siete. Plocha definovaná takouto sieťou prechádza jej rohovými vrcholmi a jej okrajové krivky sa v rohoch dotýkajú rohových strán siete. Ostatné uzly plocha neobsahuje, ale prispôsobuje sa im svojim tvarom (napr. zvyšovaním resp. znižovaním tohto bodu sa zvyšuje resp. znižuje v okolí tohto bodu aj plocha). Konštruktér potom už len interaktívne "posúva" riadiace body tak, aby dostal želaný tvar.

6.3.1 Beziérova bikubická plocha

Základná Beziérova plocha je nazývaná aj Beziérova bikubická plocha a je daná maticou 4×4 bodov t.j. šesťnástimi bodmi. Plocha je teda daná 16-imi uzlami B_{ij} , $i, j=0, 1, 2, 3$. Plocha je následne definovaná explicitnou rovnicou:

$$z = [E(x), F(x), G(x), H(x)] * \mathbf{B} * [E(y), F(y), G(y), H(y)]^T \quad (106)$$

kde $E(t)$, $F(t)$, $G(t)$, $H(t)$ sú kubické polynómy definované nasledovne:

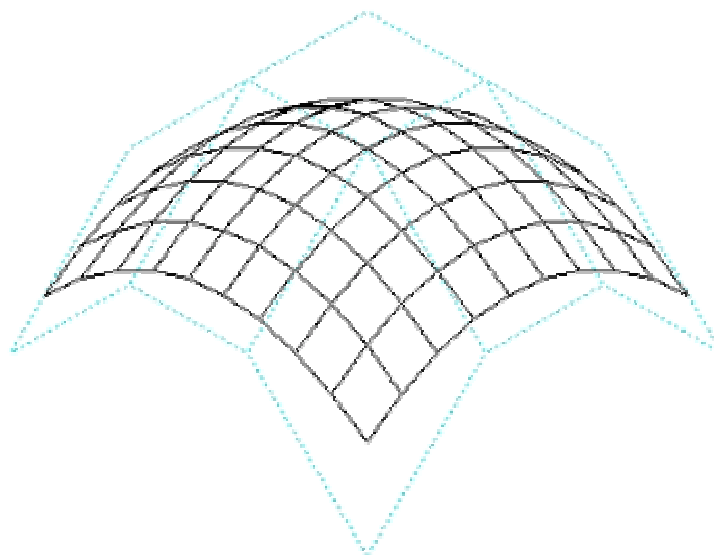
$$\begin{aligned} E(t) &= (1-t)^3 \\ F(t) &= 3 \cdot t \cdot (1-t)^2 \\ G(t) &= 3 \cdot t^2 \cdot (1-t) \\ H(t) &= t^3 \end{aligned} \quad (107)$$

a \mathbf{B} je matica kót vrcholov B_{ij} riadiacej siete:

$$\mathbf{B} = \begin{bmatrix} z_{00} & z_{01} & z_{02} & z_{03} \\ z_{10} & z_{11} & z_{12} & z_{13} \\ z_{20} & z_{21} & z_{22} & z_{23} \\ z_{30} & z_{31} & z_{32} & z_{33} \end{bmatrix} \quad (108)$$

Výhodou použitia takto definovaných plôch je, že môžeme meniť tvar plochy bez zmeny jej okraja (zmenou B_{11} , B_{12} , B_{21} a B_{22}). Pri naväzovaní jednotlivých plôch (resp. plátov) dosiahneme spojitú avšak nie hladkú naviazanosť, ak krajné body v mieste naviazania jednej plochy (riadok resp. stĺpec \mathbf{B}) sú identické s krajnými bodmi naväzujúcej plochy (plátu). Hladké spojenie navyše bude dosiahnuté identitou dotyčnicových vektorov na okraji plôch v smere ich spojenia. Identita dotyčnicových vektorov je zaručená, ak sú predposledné body oboch susedných plôch (plátov) v príslušnom riadku (resp. stĺpci) symetrické s bodmi okraja (je samozrejmé, že príslušné okrajové body musia byť identické). Identické spoločné okrajové body tvoria v podstate stred úsečiek, ktorých krajné body sú v podstate predposledné body v príslušných riadkoch (stĺpcoch) oboch plôch (plátov).

Pozorinky:



Obr. 61 Beziérová bikubická plocha

1. Charakterizujte plochy používané v počítačovej grafike
2. Charakterizujte a popíšte Coonsovu bilineárnu plochu
3. Charakterizujte a popíšte Coonsovu bikubickú a všeobecnú plochu
4. Charakterizujte a popíšte Beziérovú bikubickú plochu



V tejto kapitole sme sa naučili:



- ☐ Plochy v počítačovej grafike sa využívajú najmä v systémoch CAD t.j. v návrhových systémoch.
- ☐ Podľa toho, ako sú plochy zadané, ich rozdeľujeme na: plochy dané analytickým popisom, interpolačné plochy a aproximačné resp. interaktívne vytvárané plochy.
- ☐ Ak je pre plochu daný analytický predpis, je známa jej rovnica, je ľahké realizovať jej grafický výstup. Najčastejšie sa používa parametrické vyjadrenie plochy.
- ☐ Medzi základné plochy používané v počítačovej grafike patria priamkové (pravítkové) plochy.
- ☐ Všeobecnejšími plochami používanými v počítačovej grafike sú Coonsove plochy. Najjednoduchšou je Coonsova bilineárna plocha. Táto je definovaná maticou 3×3 riadiacich bodov. Rozšírením bilineárnej plochy je bikubická Coonsova plocha.
- ☐ Základným problémom Coonsových plôch tohto typu je veľmi ťažké vyjadrenie priamych dotyčnicových vektorov, čím sa veľmi ťažko dosahuje hladké spojenie dvoch Coonsových plôch.
- ☐ Medzi základné používané interaktívne modelované plochy patrí Beziérova bikubická plocha. Táto je definovaná maticou 4×4 riadiacich bodov a oproti popisovaným Coonsovým plochám umožňuje aj hladké naväzovanie plôch.



2. OKRUH

7 Vyplňovanie oblastí

Cieľ: V tejto kapitole je začiatok druhého, rozširujúceho okruhu. Po absolvovaní tejto kapitoly bude študent schopný definovať činnosti, ktoré spadajú pod pojem vyplňovanie v počítačovej grafike. Ďalej bude schopný kategorizovať základné algoritmy vyplňovania podľa toho, ako je definovaná hranica vyplňanej oblasti. V neposlednom rade bude schopný popísať a implementovať aspoň po jednom z algoritmov z každej oblasti (inverzné vyplňovanie a semienkové vyplňovanie). V rámci tejto kapitoly by mal študent takisto získať doplnkové rozširujúce informácie o ďalších typoch vyplňovania ako teoretické tak i praktické.

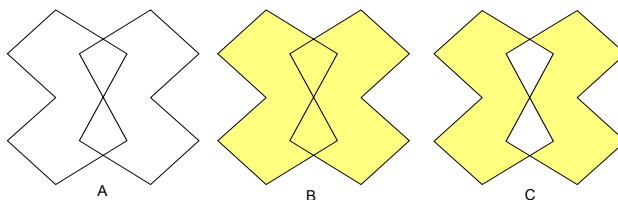
Pod pojem vyplňovanie oblasti v súčasnosti radíme nasledujúce činnosti:

- vyplnenie oblasti jedinou farbou
- vyšrafovanie oblasti
- vyplnenie farebným vzorom (v 3D textúrovanie).

Túto funkciu má v súčasnosti implementovanú takmer každý grafický systém. Funkcia c) je implementovateľná najmä na rastrových zariadeniach. Typickým triviálnym algoritmom na vyplnenie oblasti, je algoritmus, ktorý prechádza všetkými bodmi zobrazovacej plochy a v prípade, že bod je vo vnútri oblasti, potom daný bod sa vyplní vyplňovacou farbou. Inak sa prejde na ďalší bod. Ani nie je potrebné hovoriť, že daný algoritmus síce vedie k cieľu, ale je značne neefektívny.

Algoritmy pre vyplňovanie oblastí je možné rozdeliť napríklad podľa toho akým spôsobom je zadaná hranica vyplňanej oblasti:

- **hranica definovaná geometricky:** Najčastejším prípadom geometricky určenej hranice je mnohoúhelník. Mnohouhelník je definovaný postupnosťou koncových bodov lomenej čiary (polygónu), tvoriacej hranicu mnohoúhelníka. Mnohouhelník môže byť konvexný alebo nekonvexný. Na Obr. 62a) je uvedený príklad mnohoúhelníka. Ak hranica pretína sama seba, je potrebné určiť vnútro oblasti. Hranica oddeľuje nevyplnený priestor od vyplneného, preto nesprávne vyplnenie je na b) a správne na c).



Obr. 62 Príklad geometricky určenej hranice

- **hranica nakreslená na zobrazovači:** Všetky informácie o hranici zadanej oblasti sú získané z obrazovej pamäte. Vstupom pre algoritmy sú súradnice vnútorného bodu oblasti a farba výplne. Niektoré algoritmy vyžadujú aj farbu hranice danej oblasti. Začínajúc zo zadaného bodu oblasti sa zisťuje, či susedné body majú farbu hranice. Ak nie, sú vyfarbené farbou výplne. Vyplňovanie je 4-spojité, t. j. v štyroch smeroch (sever, juh, východ, západ).

Iným kritériom je rozdelenie podľa techniky vyplňovania. Potom možno rozdeliť vyplňovacie algoritmy na:

- *riadková konverzia* (scan-conversion)
- *semienkové techniky* (seed-fill) (len pre raster)

Teraz si uvedieme predstaviteľov oboch typických techník.

7.1 Metóda riadkového rozkladu

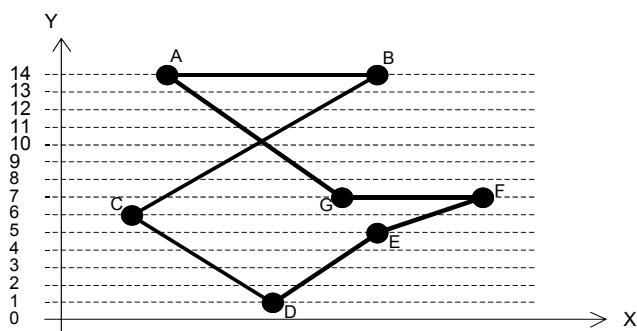
Tento algoritmus je typickým predstaviteľom techník *scan-conversion* a pre geometricky určenú hranicu. Tento algoritmus sa používa ako pre vektorové tak aj pre rastrové zariadenia

Princíp tejto metódy spočíva v postupnom vyplňaní oblasti mnohouholníka vodorovnými úsečkami, ktorých koncové body sú priesečníkmi vodorovnej priamky s hranicami oblasti. Táto priamka sa nazýva tiež rozkladový riadok.

Postupuje sa od najvyššieho vrcholu oblasti k najnižšiemu, smerom zľava doprava v každom riadku. Pre jednotlivé rozkladové riadky rovnobežné s osou X a s konštantnou súradnicou Y , klesajúcou s krokom -1 , sa nájdu priesečníky s hranicami vyplňanej oblasti. Vo výslednom zozname sú usporiadané priesečníky zľava doprava a vyfarbené úseky medzi nepárnymi a párnymi priesečníkmi. Preto je nutné aby počet priesečníkov bolo párne číslo.

Algoritmus musí obsahovať ešte niektoré úpravy:

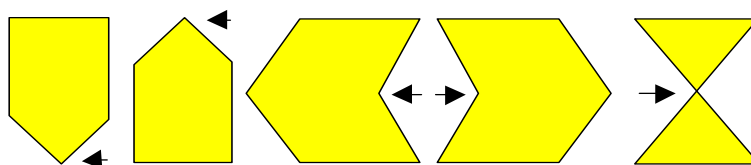
- v prípade, že rozkladový riadok prechádza vrcholom mnohouholníka alebo bodom, v ktorom sa pretínajú dve hrany, do zoznamu musia byť pridané dva vrcholy namiesto jedného, pretože rozkladový riadok pretína dve úsečky. Táto situácia nastane napr. pri výpočte priesečníkov s rozkladovým riadkom $Y=10$ a $Y=1$ na nasledujúcom obrázku.
- v prípade vodorovnej hrany je počet priesečníkov nekonečný, preto na začiatku algoritmu sú vodorovné hrany vyradené. Tieto hrany sa môžu hneď vykresliť.
- špeciálnym prípadom je stav, keď počet priesečníkov je nepárne číslo. Na nasledujúcom obrázku je takýto stav napr. pre $Y=6$, kde sú tri priesečníky: s úsečkou BC , CD a EF . Tieto sa nedajú pospájať do dvojíc a vykresliť. Preto v prípade, že ide o po sebe nasledujúce hrany monotónne stúpajúce alebo klesajúce, do zoznamu je pridaný len jeden priesečník. Aby sa monotónnosť hrán nemusela zakaždým kontrolovať, na začiatku algoritmu je prvá z dvojice takýchto hrán skrátaná o vzdialenosť zodpovedajúcu zmene súradnice Y o jeden bod. Na nasledujúcom obrázku by sa skrátili hrany CD a EF .



Obr. 63 Mnohouholník vyplňaný riadkovým rozkladom

7.1.1 Algoritmus riadkového rozkladu

1. Vykreslenie vodorovných hrán.
2. Skrátenie monotónne nadväzujúcich hrán a úprava ich orientácie zhora nadol.
3. Vytvorenie zoznamu hrán triedeného od najvyššej po najnižšiu vzhľadom na prvý bod hrany.
4. Nájdenie Y_{min} a Y_{max} .
5. Pre Y od Y_{min} po Y_{max} s krokom -1:
 - nájdenie priesečníkov hrán s rozkladovým riadkom
 - usporiadanie priesečníkov podľa súradnice X od najmenšieho po najväčší
 - vykreslenie úsekov medzi nepárnymi a párnymi priesečníkmi
6. Vykreslenie hranice mnohoúhelníka.



Obr. 64 Extrémne prípady pri vyplňovaní oblastí.

7.1.2 Nájdenie priesečníkov hrán s rozkladovým riadkom

Vychádza sa z parametrického vyjadrenia priamky:

$$\begin{aligned} x &= x_1 + t \cdot (x_2 - x_1) \\ y &= y_1 + t \cdot (y_2 - y_1) \end{aligned} \quad (109)$$

kde x_1, y_1 sú súradnice počiatočného bodu a x_2, y_2 súradnice koncového bodu. Ak úsečka tvoriaca hranu oblasti mnohoúhelníka je označená bodmi AB (definované súradnicami a_x, a_y a b_x, b_y) a rozkladový riadok bodmi CD, potom pre rozkladový riadok je parametrické vyjadrenie nasledovné:

$$\begin{aligned} x &= c_x + t \cdot (d_x - c_x) \\ y &= c_y \end{aligned} \quad (110)$$

kde c_x, c_y sú súradnice počiatočného a d_x, d_y súradnice koncového bodu. Rozkladový riadok je rovnobežný s osou x , preto y -ová súradnica je konštantá.

Z parametrického vyjadrenia úsečky a rozkladového riadku sa vypočíta priesečník z rovnosti x -ových a y -ových súradníc. Po úprave sú súradnice priesečníka nasledovné:

$$\begin{aligned} x &= a_x + \frac{c_y - a_y}{b_y - a_y} \cdot (b_x - a_x) \\ y &= c_y \end{aligned} \quad (111)$$

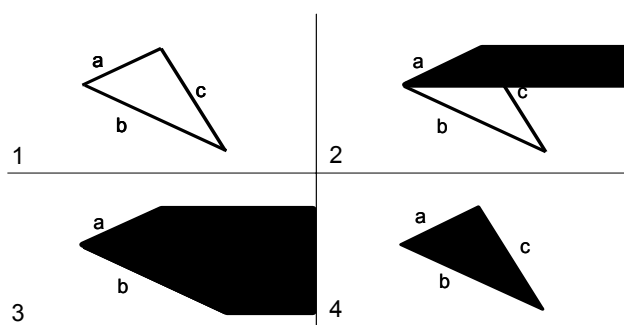
7.2 Inverzné vyplňovanie

Pod pojmom inverzia sa skrýva v podstate efekt funkcie XOR (Exclusive OR). Efekt vychádza z toho, že pri prvom aplikovaní XOR sa zmení (invertuje) jas príslušných pixelov. Pri druhom aplikovaní XOR je jas (farba) pôvodný, ako pred prvým XOR-om. Podľa toho v akom smere sa vykonáva inverzia, poznáme dve techniky inverzného vyplňovania:

- ❑ *vodorovné (riadkové) vyplňovanie*
- ❑ *zvislé (plotové) vyplňovanie*

Klasický riadkový algoritmus prechádza všetkými hraničnými úsečkami a pre každú úsečku a následne každý príslušný riadok, ktorý pretína príslušnú hraničnú úsečku invertuje všetky pixely ležiace vpravo o tejto hraničnej úsečky. Jednotlivé kroky algoritmu na jednoduchom príklade ukazuje nasledujúci obrázok:

Druhá technika t.j. plotové vyplňovanie je v podstate efektívnejšou modifikáciou predošlého algoritmu. V prvom kroku sa zvolí bázička vertikálna čiara tzv. plot. Plot by mal prechádzať niektorým vrcholom polygónu, ktorý ohraničuje vyplňovanú oblasť. Následne sa pre všetky hraničné úsečky vykoná skôr uvedené riadkové inverzné vyplňovanie. Avšak toto sa aplikuje na všetky riadky, ktoré pretínajú príslušnú hranu tak, že sa vypočíta priesečník s príslušnou hraničnou úsečkou a invertujú sa len pixely, ktoré ležia medzi týmto priesečníkom a plotom.



Obr. 65 Postup inverzného vyplňania

7.2.1 Algoritmus inverzného vyplňania

1. Skrátene monotónne nadväzujúcich hrán a úprava ich orientácie zhora nadol.
2. Vytvorenie zoznamu hrán triedeného od najvyššej po najnižšiu vzhľadom na prvý bod hrany.
3. Nájdenie X_{max} .
4. Pre každú nevodorovnú hranu:
 - ❑ nájdenie priesečníka hrany s rozkladovým riadkom
 - ❑ vykreslenie vodorovnej úsečky operáciou XOR vo výške Y medzi súradnicami X a X_{max}
5. Vykreslenie hranice mnohoúhelníka.

Nájdenie priesečníka je popísané v skôr uvedenej kapitole.

7.3 Šablónové vyplňovanie

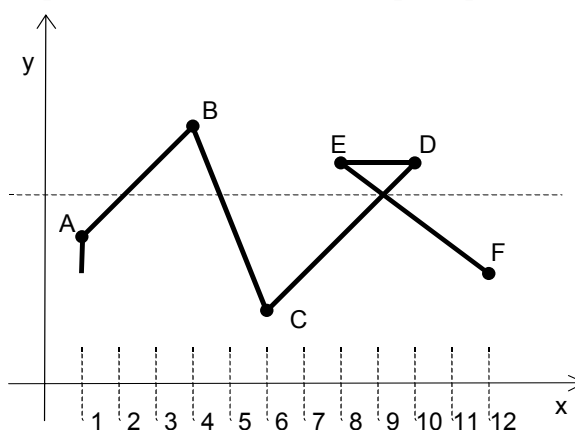


Pri tejto metóde je vyplňanie oblasti prevedené na vyplňanie šablóny. Šablóna je údajová štruktúra v operačnej pamäti, v ktorej sa vyplňa oblasť mnohoúhelníka. Údaje zo šablóny sú potom použité na výplň oblasti.

Veľkosť šablóny musí zodpovedať veľkosti oblasti mnohoúhelníka, preto je táto metóda náročná na pamäť. Údaje v šablóne sú dvojstavové (vyplnený/ nevyplnený). Vodorovné úseky, ktoré majú byť vyplnené, sú do šablóny zapisované len pomocou ich koncových bodov, takže prístup do šablóny je minimálny.

Postup je názorne ukázaný na príklade na nasledujúcom obrázku. Na začiatku je celá šablóna prázdna. Pre rozkladový riadok prechádzajúci hranami AB, BC, CD a EF je postup vyplnenia riadku šablóny v nasledujúcej tabuľke. Po spracovaní hrany CD sú v šablóne tri priesečníky - bity 2, 4, 9 nastavené na logickú 1. Po spracovaní hrany EF je deviaty bit opäť priesečníkom, je invertovaný a nadobúda opačnú hodnotu teda 0.

Negácia bitov zabezpečí, že v každom riadku bude párny počet nastavených bitov.



Obr. 66 Vyplnenie jedného riadku šablóny

krok \ stĺpec	1	2	3	4	5	6	7	8	9	10	11	12
štart	0	0	0	0	0	0	0	0	0	0	0	0
po AB	0	1	0	0	0	0	0	0	0	0	0	0
po BC	0	1	0	0	1	0	0	0	0	0	0	0
po CD	0	1	0	0	1	0	0	0	1	0	0	0
po EF	0	1	0	0	1	0	0	0	0	0	0	0

Tab. 2 Vyplnenie jedného riadku šablóny

7.3.1 Algoritmus vyplňania šablóny

1. Vyčistenie šablóny.
2. Skrátanie monotónne nadväzujúcich hrán.
3. Pre každú nevodorovnú hranu:
 - ❑ nájdenie riadkovým rozkladom priesečníkov hrany s rozkladovým riadkom
 - ❑ negácia zodpovedajúcich bitov v šablóne

4. Pre každý riadok v šablóne nájdenie pozícií bitov nastavených na logické jednotky a vykreslenie vodorovných úsekov medzi nepárnymi a párnymi bitmi v šablóne do obrazovej pamäte.
5. Vykreslenie hraníc oblasti mnohoúhelníka.

Nájdenie priesečníkov je popísané v kapitole o riadkovom rozklade. Kvôli náročnosti tohto algoritmu na pamäť môže byť algoritmus upravený. Body algoritmu 1, 2, 5 sú rovnaké. Veľkosť šablóny je však daná dĺžkou ($X_{max}-X_{min}$) a obsahuje len priesečníky spracúvaného riadku $Y=k$. Pri každom prechode na nový riadok $Y=k+1$ sa šablóna vyčistí, nájdu sa nové priesečníky a vykreslia sa vodorovné úseky medzi nastavenými nepárnymi a párnymi bitmi.

7.4 Porovnanie efektívnosti metód

Metóda riadkového rozkladu je najpomalšou metódou. Jej slabinou je nutnosť usporiadania priesečníkov na každom rozkladovom riadku a prechod celého zoznamu hrán pre každý rozkladový riadok.

Metóda inverznej výplne je veľmi rýchla, je lineárne závislá na počte hraničných úsečiek. Zoznam hrán stačí prejsť raz. Jej nevýhodou však je, že body vyplnenej oblasti neprekryjú to, čo bolo na obrazovke nakreslené skôr. Výhodnejšie je vyplniť najskôr oblasť v pomocnej pamäti a výsledok preniesť do obrazovej, tak ako je to pri výplni šablónou.

Metóda výplne šablónou je takisto rýchlejšia ako metóda riadkového rozkladu. Kvôli svojej jednoduchosti a efektívnosti je používaná v grafických procesoroch a akcelerátoroch.

7.5 Vypĺňanie iných oblastí



Niekedy sa vyžaduje vyplňanie oblastí ohraničených čiarou kombinovanou napr. z úsečiek, kruhových oblúkov, častí spline kriviek. Aby bol dodržaný princíp vykreslenia spojnic medzi jednotlivými priesečníkmi, musí mať každá krivka maximálne jeden priesečník s rozkladovým riadkom. Táto podmienka je porušená u kruhového oblúku. Krivku je nutné rozdeliť na menšie časti. V mnohých prípadoch je výhodnejšie aproximovať krivky lomenou čiarou a použiť niektorú z vyššie uvedených metód.

7.6 Výplň spektrom (prechod z jednej farby do druhej)



Pri tejto výplni sú vstupnými hodnotami dve farby F_1 a F_2 , geometricky daná hranica vo forme pol'a bodov a smer zmeny farby vo forme uhla α . Smer vykresľovania priamok tvoriacich výplň je kolmý na smer zmeny farby, teda $\beta = \alpha + 90^\circ$.

- Prvý spôsob výplne spektrom využíva metódu riadkového rozkladu popísanú skôr. Na začiatku sa musí otočiť mnohoúhelník tak, aby smer vykresľovania priamok bol rovnobežný s osou x , t.j. o uhol β a metódou riadkového rozkladu sa nájdu priesečníky. Pri vykresľovaní úsekov medzi priesečníkmi sa musia úseky spätne otočiť o uhol β . Rozdiel oproti riadkovej výplni je v tom, že oblasť nie je vyplňaná jednou farbou, ale farba postupne plynule prechádza z prvej farby do druhej.

Poznámky:

- Druhý spôsob využíva možnosť nastavenia orezávacej oblasti, ktorú poskytuje napr. grafické rozhranie MS Windows. Na začiatku sa nastaví orezávacia oblasť na celú oblasť mnohoúhelníka a vypočítajú sa súradnice X_{min} , X_{max} , Y_{min} , Y_{max} . Podľa veľkosti uhla β je vyplnený príslušný rovnobežník a orezávacia oblasť zabezpečí vyplnenie len v oblasti mnohoúhelníka.

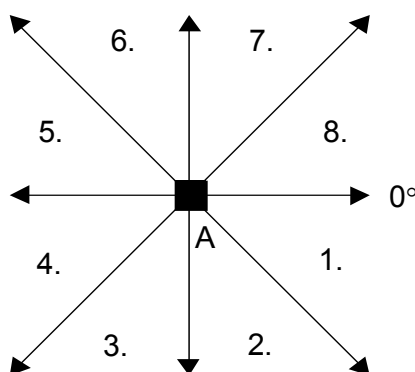
Prvý spôsob je pomalý a náročný na výpočet, ale dáva lepšie výsledky. Druhý spôsob je veľmi rýchly, pretože stačí určiť príslušný rovnobežník a vyplniť ho. V tomto prípade však môže dôjsť k čiastočnému orezaniu časti farebného spektra, ktorým danú oblasť vyplníme, ak oblasť mnohoúhelníka je oveľa menšia ako oblasť rovnobežníka.

- **Plynulý prechod z prvej farby do druhej**

1. Zistenie rozdielu jednotlivých RGB zložiek prvej a druhej farby.
2. Vypočítanie počtu úsečiek v rovnobežníku.
3. Vypočítanie prírastkov jednotlivých zložiek ako pomer rozdielu zložiek prvej a druhej farby a počtu úsečiek, napr. pre zložku R:

- **Určenie vyplňaného rovnobežníka na základe uhla β**

Ako vidno na nasledujúcom obrázku, uhol β môže byť z jedného z ôsmich oktantov. Oktanty 1-4 sú posunuté oproti oktantom 5-8 o 180° , preto stačí vyšetriť oktanty 1-4 a pre oktanty 5-8 vymeniť farby F_1 a F_2 .



Obr. 67 Rozmiestnenie oktantov

Ak veľkosť uhla β je z prvého oktantu $\langle 0^\circ - 45^\circ \rangle$, rovnobežník má tvar podľa Obr. 68a). Pri vykresľovaní sa najprv vypočíta Δy :

$$\Delta y = (X_{max} - X_{min}) \cdot \tan(\beta) \quad (112)$$

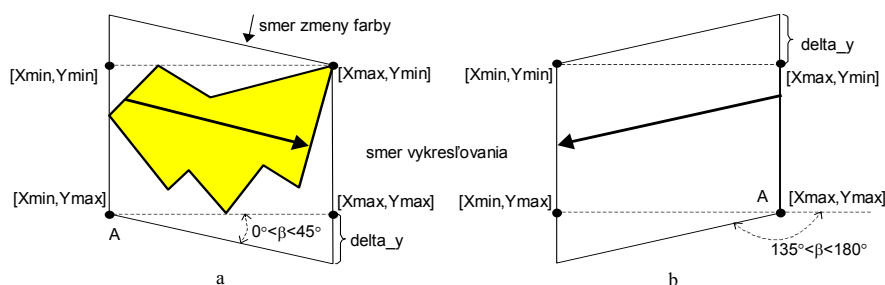
a postupne pre $i = (Y_{min} - \Delta y)$ po Y_{max} ,

$j = Y_{min}$ po $(Y_{max} + \Delta y)$ s krokom +1

sa vykresľujú úsečky z bodu (i, X_{min}) do bodu (j, X_{max}) . Po vykreslení všetkých úsečiek vznikne mnohoúhelník vyplnený spektrom. Štvrtý oktant je odvodený analogicky na Obr. 68b).

V tomto prípade:

$$\Delta y = (X_{max} - X_{min}) \cdot \frac{1}{\tan(\beta - 90^\circ)} \quad (113)$$



Obr. 68 Rovnobežníky pre oktanty 1 a 4

Rovnobežník pre druhý oktant je na

Obr. 69c). V tomto prípade sa delta_x vypočíta:

$$\text{delta_x} = (Y_{\max} - Y_{\min}) \cdot \frac{1}{\tan(\beta)} \quad (114)$$

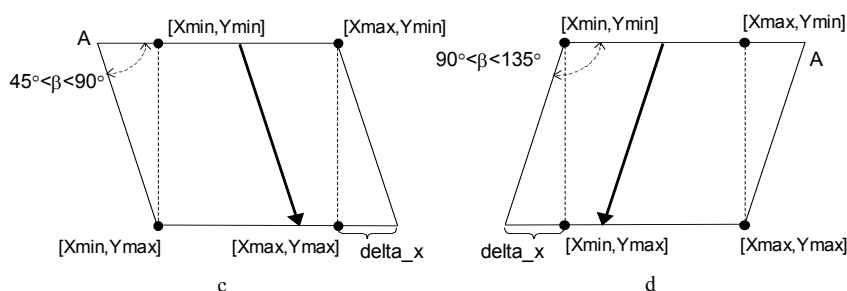
a postupne od $i = (X_{\min} - \text{delta_x})$ po X_{\max} ,

$j = X_{\min}$ po $(X_{\max} + \text{delta_x})$ s krokom +1

sa vykresľujú úsečky z bodu (i, Y_{\min}) do bodu (j, Y_{\max}) . Tretí oktant je odvodený analogicky na

Obr. 69d), v tomto prípade:

$$\text{delta_x} = (Y_{\max} - Y_{\min}) \cdot \tan(\beta - 90^\circ) \quad (115)$$



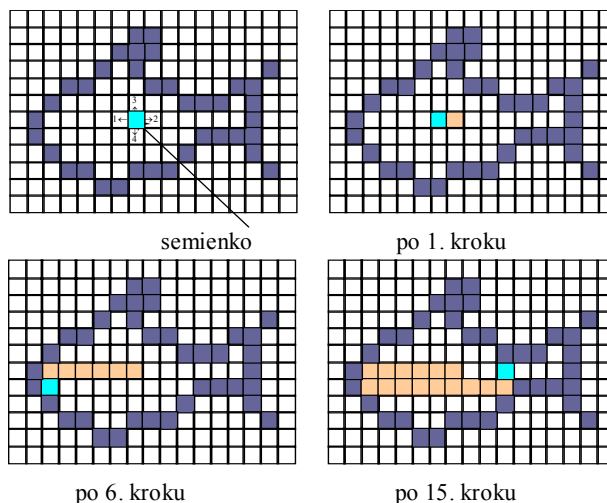
Obr. 69 Rovnobežníky pre oktanty 2 a 3

Na záver snáď len dodajme, že spektrálny prechod môžeme implementovať v rôznych variáciách ako napríklad v lineárnom prechode, tak napr. aj v radiálnom prechode.

7.7 Semienkové vyplňovanie

Vyplňovacie algoritmy tohto typu sa veľmi často používajú v rastrových zariadeniach a v interaktívnej grafike. Pre vyplnenie sa predpokladá, že je známy aspoň jeden vnútorný bod vyplňovanej oblasti (tzv. semienko). Najčastejšie ho definuje sám používateľ ukázaním do vnútra oblasti. Najzákladnejší algoritmus tohto typu má rekurzívny charakter. Pracuje sa priamo s bodmi rastra a testujú sa vždy susedné body semienka. Za susedné sa považujú len horný, dolný, pravý a ľavý bod, body po diagonálach sa neuvažujú. Pri vyšetrowaní sa zisťuje, či susedný bod je hraničný alebo nie a navyše či už je vyfarbený alebo nie. Ak bod nie je hraničný a nie je ani vyfarbený, vyfarbí sa a v novej úrovni rekurzcie sa ako semienko definuje tento bod. Rekurzia končí keď sa dosiahne hraničný bod resp. všetky susedné body sú už

vyfarbené. Takéto poňatie algoritmu má síce pomerne pekný algoritmickejší zápis avšak algoritmus je neefektívny a má vysoké pamäťové nároky (kvôli rekurzii). V súčasnosti sa používa nerekurzívna modifikácia tohto algoritmu. Tu sa pracuje po riadkoch a vyplňujú sa všetky body naľavo a napravo od semienka po hraničné body. Pokračovacie body (napr. pri nekonvexných útvaroch) sa ukladajú napríklad do zásobníka a postupne sú využívané.

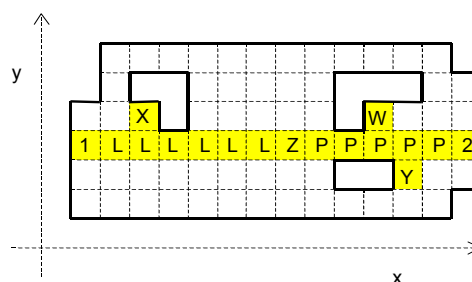


Obr. 70 Postup semienkového vyplňovania

7.7.1 Nerekurzívne semienkové vyplňovanie.

Toto vyplňovanie býva častokrát označované aj ako *floodfill* (záplavové vyplňovanie). Jedná sa v podstate o nerekurzívnu modifikáciu semienkového vyplňovania. Tým je odstránená náročnosť na veľkosť zásobníka. Funkcia si v priebehu vyplňovania ukladá informáciu o bodoch, z ktorých potom bude pokračovať. Informácia sa skladá zo súradníc bodu a príznaku smeru, ktorým sa má z daného bodu vyplňovať. Ešte pred vyplňovaním sa uloží počiatočný bod so súradnicami, ktoré sú vstupnými parametrami funkcie a s príznakom vyplňovania všetkými smermi (hore, dole, vľavo, vpravo). Ďalej sa v slučke, pokiaľ existuje uložený bod, opakuje táto činnosť:

1. nech posledne uložený bod má súradnice (x,y) a príznak p
 - ak príznak p obsahuje smer vľavo znižuj x až kým bod x,y nemá inú farbu ako tú, ktorú prekresľujeme. Zároveň vyšetruj body $(x,y-1)$ a $(x,y+1)$ s tým, že ak nastala zmena z inej farby na prekresľovanú medzi bodmi $(x,y-1)$ a $(x-1,y-1)$, alebo $(x,y+1)$ a $(x-1,y+1)$ tak bod $(x,y-1)$ príp. $(x,y+1)$ ulož ako ďalší bod vyplňovania.
 - ak príznak p obsahuje smer vpravo postup je obdobný x



Obr. 71 Algoritmus nerekurzívneho semienkového vyplňovania

kde:

4. Z začiatkový bod (x,y)
 5. L šírenie vľavo
 6. P šírenie vpravo
 7. X tento bod sa uloží pri šírení vľavo a príznak sa nastaví na smer vľavo a hore
 8. body Y a W sa uložia pri šírení vpravo a príznaky budú mať Y-vpravo a dole, W-vpravo a hore.
2. v ďalšom sa vykreslí horizontálna čiara v riadku y s krajnými bodmi 1 a 2.
- ☐ ak p obsahuje smer hore, uloží sa bod (x,y-1) s príznakom hore a vľavo a vpravo.
 - ☐ ak p obsahuje smer dole, uloží sa bod (x,y+1) s príznakom dole a vľavo a vpravo.
3. keďže boli prešetrované všetky smery šírenia sa z bodu Z, tento bod sa vymaže a pokračuje sa z posledne uloženým bodom.

Na zjednodušenie je možné upraviť algoritmus tak, že vstupnými parametrami sú štartovací bod a farba výplne a:

- ☐ za hraničnú farbu sa považuje farba, ktorá je odlišná od farby aktuálneho bodu
- ☐ za rovnaké sa považujú farby, ktoré sú v intervale (R-toleranciaR, R+toleranciaR), (G-toleranciaG, G+toleranciaG), (B-toleranciaB, B+toleranciaB)

1. Charakterizujte vyplňovanie oblastí používané v počítačovej grafike
2. Charakterizujte a popíšte algoritmus riadkového rozkladu pri vyplňovaní oblastí
3. Ako sa nájdu priesečníky hrán s rozkladovým riadkom pri algoritme riadkového rozkladu pri vyplňovaní oblastí
4. Charakterizujte a popíšte inverzné vyplňovanie
5. Charakterizujte a popíšte plotové vyplňovanie
6. Charakterizujte a popíšte vyplňovanie šablónou
7. Popíšte spôsob vytvorenia šablóny pri šablónovom vyplňaní
8. Popíšte základný postup pri vyplňovaní spektrom resp. prechodom jednej farby do druhej v rámci vyplňovania
9. Charakterizujte a popíšte rekurzívne semienkové vyplňovanie
10. Charakterizujte a popíšte nerekurzívne semienkové vyplňovanie





V tejto kapitole sme sa naučili:

- ☐ Pod pojem vyplňovanie oblasti v súčasnosti radíme nasledujúce činnosti: vyplnenie oblasti jedinou farbou, vyšrafovanie oblasti a vyplnenie farebným vzorom (v 3D textúrovanie).
- ☐ Algoritmy pre vyplňovanie oblastí sa rozdeľujú podľa toho akým spôsobom je zadaná hranica vyplňanej oblasti a to: hranica definovaná geometricky a hranica nakreslená na zobrazovači. Druhým kritériom delenia je technika vyplňovania. Podľa toho sa algoritmy vyplňovania delia na algoritmy riadkovej konverzie (scan conversion) a semienkové techniky.
- ☐ Typickým predstaviteľom techniky riadkovej konverzie a pre geometricky určenú hranicu je metóda riadkového rozkladu. Tento algoritmus sa používa ako pre vektorové tak pre rastrové zariadenia. Princíp spočíva v postupnom vyplňaní oblasti mnohoúhelníka vodorovnými úsečkami, ktorých koncové body sú priesečníkmi vodorovnej priamky s hranicami oblasti. Táto priamka sa nazýva tiež rozkladový riadok.
- ☐ Ďalším predstaviteľom riadkovej konverzie a geometricky definovanej hranice je inverzné vyplňovanie. Pri tomto algoritme sa využíva funkcia XOR (eXclusive OR). Klasický riadkový algoritmus prechádza všetkými hraničnými krivkami a pre každú hraničnú krivku a následne každý príslušný riadok, ktorý pretína príslušnú hraničnú krivku, invertuje všetky pixely ležiace vpravo od tejto hraničnej krivky.
- ☐ Ďalším predstaviteľom riadkovej konverzie a geometricky definovanej hranice je šablónové vyplňovanie. Pri tejto metóde je vyplňanie oblasti prevedené na vyplňanie šablóny. Šablóna je údajová štruktúra v operačnej pamäti, v ktorej sa vyplňa oblasť mnoho uholníka. Údaje zo šablóny sú potom použité na výplň oblasti. Veľkosť šablóny musí zodpovedať veľkosti oblasti mnohoúhelníka, preto je táto metóda náročná na pamäť.
- ☐ Pri výplni spektrom resp. prechodom z jednej farby do druhej sú vstupnými hodnotami dve farby (štartovná a koncová), geometricky daná hranica vo forme poľa bodov a smer zmeny farby vo forme uhla. Základnou metódou pri tomto postupe je modifikovaná metóda riadkového rozkladu.
- ☐ Typickým predstaviteľom semienkového algoritmu hranice definovanej na zobrazovači je semienkové vyplňovanie. Tento algoritmus sa používa u rastrových zariadení. Pre vyplnenie sa predpokladá, že je známy aspoň jeden vnútorný bod (semienko) vyplňovanej oblasti. Algoritmus má rekurzívny charakter. Pri vyplňovaní sa vyšetruje či susedný bod je hraničný alebo či už je vyfarbený vyplňovou farbou. Ak nie vyfarbí sa a rekurzívne sa pokračuje ďalším susedným bodom.
- ☐ Klasické semienkové vyplňovanie je náročné na pamäť. Preto sa používa aj nerekurzívna modifikácia semienkového vyplňovania. Tejto metóde sa hovorí aj záplavové vyplňovanie.

8 Riešenie viditeľnosti



Cieľ: Absolvovaním tejto kapitoly by mal študent získať vedomosti z vyšších technológií, ktoré sa v rámci počítačovej grafiky používajú. V prvom rade bude študent schopný charakterizovať problém riešenia viditeľnosti v počítačovej grafike a kategorizovať tento problém podľa rôznych kritérií. Ďalej bude schopný vymenovať a definovať základné typy algoritmov používaných v tejto oblasti ako aj definovať okruh ich nasadenia podľa riešeného problému počítačovej grafiky. V základnej miere by mal byť schopný rozpoznať a aplikovať niektoré urýchľujúce postupy, ktoré sa v tejto oblasti používajú.

Riešenie viditeľnosti v počítačovej grafike, ako je z predchádzajúcich riadkov zrejmé, nepatrí k triviálnym úlohám. V čom vlastne spočíva riešenie viditeľnosti (angl. visibility problem). Spočíva v odstránení (resp. odlišení) tých častí trojrozmerných objektov, ktoré pri danom premietaní do 2D nie sú z miesta pozorovateľa viditeľné. Tým sú tieto časti akože zakryté a dostávame jednoznačné priemety želaných telies.

Na riešenie problému viditeľnosti existuje niekoľko algoritmov. Rozdelenie algoritmov je možné podľa niekoľkých hľadísk. Základné delenie je podľa toho v akom tvare poskytujú výsledky na *vektorové* ((čiarové, hranové) výsledkom je množina kriviek predstavujúcich viditeľné hrany) a *rastrové* ((bodové) výsledkom je obraz predstavujúci jednotlivé body (pixels) s farbou príslušných viditeľných plôch s možnosťou ich osvetlenia príp. tieňovania). Ďalšie delenia sú možné:

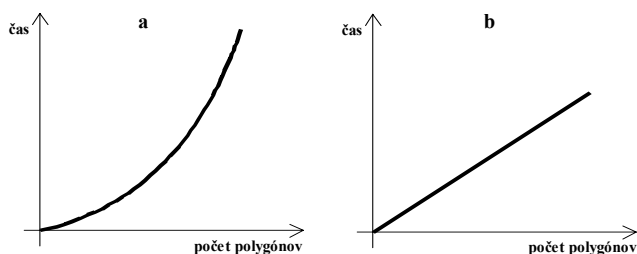
- *podľa priestoru, kde je viditeľnosť riešená:*
 - ☐ riešenie v 3D
 - ☐ riešenie v 2D priemete
- *podľa reprezentácie objektov, ktorých viditeľnosť riešime:*
 - ☐ objektovo orientované algoritmy (kde sa rieši, ktorá časť príslušného objektu je viditeľná)
 - ☐ obrazovo orientované algoritmy (kde sa rieši spätne pre každý obrazový bod, ktorý objekt je v ňom vidieť)
- *podľa toho či sa uvažuje aj osvetlenie telesa:*
 - ☐ riešenie bez osvetlenia (vyhodnotenie farieb je aplikované lokálne na každý objekt)
 - ☐ riešenie s osvetlením (medzi tieto radíme v podstate aj metódu sledovania lúča (raytracing) alebo vyžarovaciu metódu (radiosity), ktoré sa častokrát označujú aj ako globálne metódy riešenia viditeľnosti s globálnou aplikáciou farieb v rámci scény t.j. napr. aj odrazy)
- *podľa vplyvu možnej chyby pri vykonávaní (napr. použitím celočíselnej aritmetiky):*
 - ☐ s lokálnym vplyvom chyby na výsledok
 - ☐ s globálnym vplyvom chyby na výsledok
- *podľa času potrebného na riešenie viditeľnosti:*
 - ☐ riešenie mimo reálneho času
 - ☐ riešenie v reálnom čase

Poznámky:

Na vyriešenie viditeľnosti je potrebný čas T . Tento čas sa skladá z dvoch časov: času T_V a času T_K .

- T_V je čas potrebný na výber potenciálne viditeľných objektov (alebo neviditeľných) resp. ich utriedenie
- T_K je čas potrebný na vykreslenie utriedených objektov. Pritom musí platiť, že

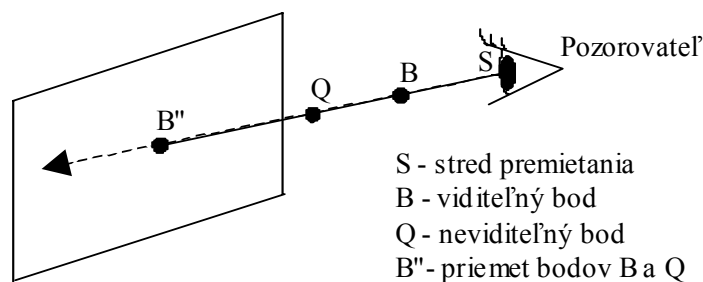
$$T = T_V + T_K \quad (116)$$



Obr. 72 Priebehy času T_V pri objektových a rastrových typoch algoritmov

Pre jednoduché scény sú obidva časy približne rovnaké. Pre zložitejšie scény sa časové kvantum zväčšuje v prospech jedného alebo druhého času, podľa použitého algoritmu. Pre vektorové typy algoritmov objektového typu je závislosť času T_V od počtu polygónov v scéne ukázaná na Obr. 72a. V tomto prípade je kvadratická závislosť od počtu polygónov (t.j. zložitosť úmerná n^2 kde n je počet polygónov). Pre niektoré vektorové typy najmä s potrebným predprocesingom napr. BSP, je zložitosť len $O(n)$ a platí Obr. 72b. Pre rastrové algoritmy, teda najmä obrazovo orientované typy, je závislosť času T_V od počtu polygónov v scéne ukázaná na Obr. 72b. Tu je lineárna závislosť od počtu polygónov (zložitosť je úmerná $n \cdot r$, kde n je počet polygónov a r je závislé od rozlíšenia výstupného obrazu ($r=x \cdot y$ (rozlíšenie v smere osi x a y) čo predstavuje počet požadovaných výstupných bodov (pixelov))).

V súčasnosti pri použití rýchlych grafických adaptérov a príp. aj akcelerátorov je čas T_K pomerne krátky a časové kvantum sa presúva do výberovej fázy. Preto má veľký význam skrátenie výberovej fázy, čo v konečnom dôsledku zníži čas potrebný na vyriešenie viditeľnosti a možnosť pracovať v reálnom čase.



Obr. 73 Viditeľnosť bodu

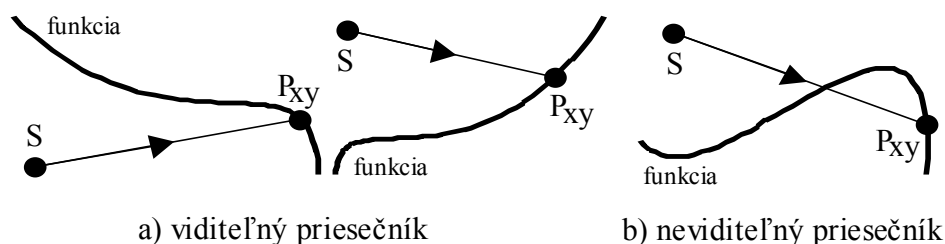
Teraz si definujeme, kedy je požadovaný bod (napr. B) viditeľný. Bod B je viditeľný, ak na priamke medzi bodom B a stredom premietania (poloha pozorovateľa) sa nenachádza žiadny iný bod objektu riešenej 3D scény. Ak sa takýto bod nachádza, potom je samozrejme bod B neviditeľný. V nadväznosti na tieto definície a následné vyhodnotenie uvedených podmienok, je nutné definovať však umiestnenie bodu B v rámci určitej súradnicovej sústavy.

8.1 Riešenie viditeľnosti grafov funkcií

8.1.1 Riešenie v priestore

Medzi klasické algoritmy patrí algoritmus pre zobrazenie grafu funkcií dvoch premenných $[f(x,y)]$. V prípade riešenia viditeľnosti tohto typu je vhodné zaradenie NSS ako medzisústavy, čím je možné dosiahnuť nezávislosť na rozlíšení SSZ, nakoľko výsledky budú normalizované do intervalu $<0,1>$. Graf je vlastne vytváraný ako aproximácia siete dvoch systémov kriviek, z ktorých jeden je rovnobežný s rovinou XZ a druhý s rovinou YZ . Jednotlivé krivky sa pretínajú v im odpovedajúcich bodoch, ktoré sú dané aj krokom aproximácie. Pri tomto postupe sa predpokladá, že vzniknutá plocha bude vykresľovaná vo forme drôtového modelu. V prípade povrchového modelu (použitie plátovania) sa použije iný algoritmus.

Priestorové riešenie je založené na testovaní priesečníkov spojnice medzi stredom SSC (stredom premietania) a príslušným priesečníkom siete, ktorý leží na ploche zobrazovanej funkcie. Na základe výsledku testu je možné rozdeliť priesečníky na viditeľné a neviditeľné. Po zistení viditeľnosti, sú spojené priesečníky rovnakého typu (viditeľné). Pri susedných priesečníkoch, ktoré sú rôzneho typu, je postupným delením a opakovaním algoritmu hľadaná hranica medzi viditeľnou a neviditeľnou časťou.

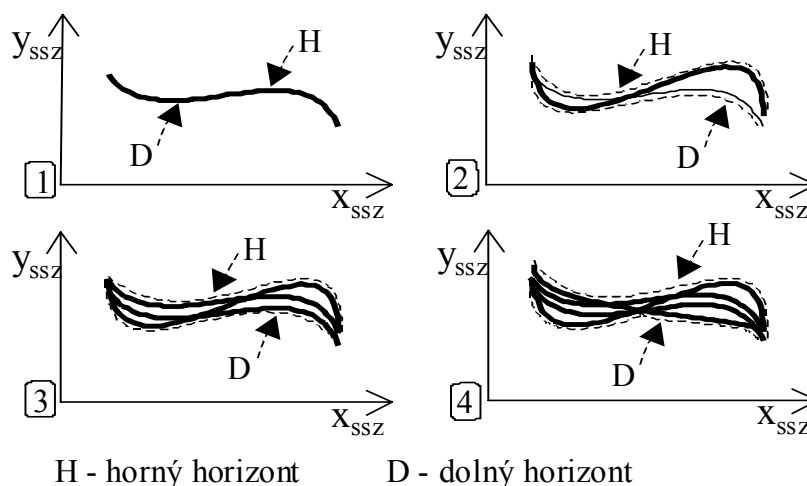


Obr. 74 Testovanie viditeľnosti bodov grafu funkcie

8.1.2 Riešenie v priemetni (Algoritmus plávajúceho horizontu)

Majme opäť klasickú funkciu dvoch premenných $f(x,y)$. Majme ju opäť aproximovanú sústavou rovnobežných kriviek. Potom pri danom premietaní sa najprv vyberie tá z nich, ktorá je najbližšie k stredu premietania (stred SSC). Táto je určite viditeľná a teda ju vykreslíme. Ďalej postupne zobrazujeme susedné krivky tak, že z nich zobrazíme len tie časti, ktoré sú "nad" (resp. "pod" podľa parametrov premietania t.j. vektora pohľadu) niektorou už zobrazenou krivkou (horizontom). Takto postupne vykresľované krivky vytvárajú tzv. "obalovú" krivku. "Pod" či "nad" ňou sa nezobrazí žiadna z nasledujúcich kriviek. Často sa tento algoritmus modifikuje zobrazením dvoch sústav kriviek. Testovanie "nad" resp. "pod" sa vykoná až v SSZ t.j. v priemetni a to najčastejšie podľa y -ovej súradnice.

Poznámky:



Obr. 75 Postup pri vytváraní horizontu v algoritme plávajúceho horizontu

8.2 Maliarov algoritmus riešenia viditeľnosti

Tento algoritmus pracuje tak, že objekty (napríklad jednotlivé polygóny) najďalej od pozorovateľa v rámci USS vykreslí ako prvé a objekty najbližšie ako posledné. Tým sa zabezpečí prekrytie (prekreslenie) vzdialenejších objektov bližšími a tým aj správna viditeľnosť.

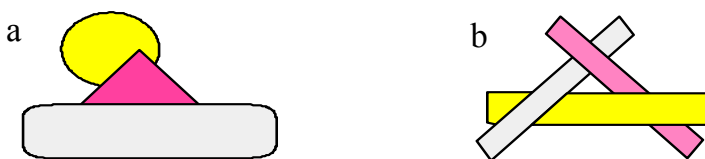


Obr. 76 Maliarov algoritmus

V scéne na Obr. 77a by sa podľa tohto algoritmu najprv kreslil kruh, potom trojuholník, ktorý by prekryl kruh a nakoniec obdĺžnik, ktorý je najbližšie k pozorovateľovi. Viditeľnosť je v tomto prípade vyriešená správne. Otázkou zostáva, na základe akého kritéria polygóny v 3D porovnávať, pretože vzdialenosť polygónu vo všeobecnosti nie je konštantná v každom jeho bode. K dispozícii sú v podstate tri možnosti, čo považovať za vzdialenosť polygónu od pozorovateľa. Môže to byť minimálna, maximálna, alebo priemerná vzdialenosť zo všetkých jeho bodov. Častokrát sa používa jedna priemerná súradnica, kvôli rýchlosti, pre celý polygón. Pre skvalitnenie je však možné postupovať nasledovne. Zistia sa krajné súradnice z každého polygónu a podľa menšej z nich polygóny utriedime (*Z-sort*). Tu však oproti spriemerovaniu nastáva kvalitatívny posun, pretože vzniknutý zoznam plôch ešte priamo nevykreslíme, nakoľko polygóny môžu mať i také polohy, pri ktorých polygón podľa zoznamu vzdialenejší, môže zakrývať určitý iný polygón. Najvzdialenejší polygón zo zoznamu je preto testovaný voči niekoľkým ďalším polygónom. Ak sa na základe týchto testov možno rozhodnúť, že polygón leží za všetkými ostatnými polygónmi, je vykreslený. V opačnom prípade dôjde k výmene s polygónom, u ktorého dopadli všetky testy prekryvania negatívne.

Niektoré testy sú založené na porovnávaní dvoch súradníc, iné vyžadujú náročnejšie výpočty. V nasledujúcich testoch prekryvania sú testované polygóny (vo všeobecnosti je možné použiť aj plochy) označené ako $P1$ a $P2$ a ich krajné súradnice $(z1_{min}, z1_{max})$ a $(z2_{min}, z2_{max})$. Testy sa vykonávajú postupne, pri splnení jedného sa môžu ostatné vynechať.

1. Ak $z1_{max} < z2_{min}$, potom je polygón $P1$ úplne za polygónom $P2$.
2. Ak sa neprekrývajú priemety polygónov v rovine priemetne (t.j. v SSZ), je poradie ich vykresľovania nezávislé a polygón $P1$ testu vyhovuje.
3. Ak sú všetky vrcholy polygónu $P1$ pod rovinou určenou polygónom $P2$, potom $P1$ leží za polygónom $P2$.
4. Ak sú všetky vrcholy polygónu $P2$ nad rovinou určenou polygónom $P1$, potom $P1$ leží za polygónom $P2$.
5. Ak sa poloha testov nedá vyhodnotiť podľa testov 1 až 4, je nutné nájsť v priemete ľubovoľný priesečník oboch plôch a podľa odpovedajúcich súradníc $z1$ a $z2$ tohto bodu určiť poradie vykresľovania plôch.



Obr. 77 Modelové scény pre maliarov algoritmus bez a s cyklickým prekryvom

Jediná výhoda maliarovho algoritmu spočíva v jeho jednoduchosti na pochopenie a naprogramovanie. Nevýhod je naproti tomu hneď niekoľko:

- je nutné zoradiť objekty podľa vzdialenosti od pozorovateľa (tzv. *Z-sort*), čo je zdĺhavý proces ($O(n \cdot \log_2 n)$ v najlepšom prípade) najmä ak je scéna väčšieho rozsahu (rádovo 10000 a viac polygónov). Navyše je niekedy problém nájsť kritérium pre porovnanie, ktorý objekt je bližšie a ktorý ďalej, o ktorom už bolo hovorené.
- je neefektívny, pretože vzdialené objekty sa často kreslia úplne zbytočne, keďže sú prekryté bližšími objektmi (nie sú viditeľné). Je ale možné toto odstrániť kreslením objektov spredu dozadu a vypočítané pixely už neprekresľovať.
- nie je korektný - v určitých prípadoch nedáva správne výsledky, pretože skôr uvedený rozbor testov nie je úplne postačujúci, nakoľko môžu nastať i zložitejšie prípady vzájomného prekryvania viacerých plôch. Tento fakt demonštrujú cyklicky prekryté rovnobežníky na Obr. 77b, kde nezáleží na poradí, v akom sa vykreslia, nikdy to nebude nakreslené správne. Tento stav je možné ľahko odhaliť, pretože vplyvom cyklického presunu sa na miesto testovaného polygónu dostane taký polygón, ktorý tam už raz bol. Situáciu je možné riešiť rozdelením polygónu na menšie časti, čím sa však znižuje efektívnosť algoritmu resp. tento nedostatok maliarovho algoritmu rieši napríklad BSP.

8.3 Warnockov algoritmus

S príchodom rastrových zobrazovačov sa do značnej miery rozšírili algoritmy, ktoré riešia viditeľnosť až v priemetni t.j. v SSZ resp. aj v NSS. Medzi klasického predstaviteľa tejto triedy patrí Warnockov algoritmus.

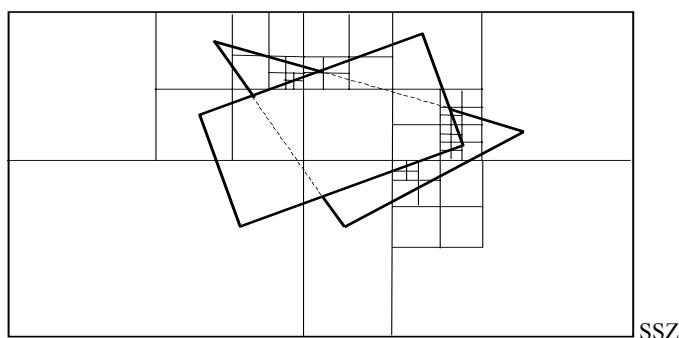
Warnockov algoritmus umožňuje zobraziť viditeľné časti telesa dvoma spôsobmi:

- *hrany* (drôtové zobrazenie)
- *steny* (plošné zobrazenie)

Algoritmus rieši viditeľnosť ľubovoľných, i nekonvexných telies s rovinnými stenami. Výpočtami podľa Warnockovho algoritmu sa určia tie časti hrán a plôch, ktoré sú viditeľné a len tie sa vykreslia na zobrazovaciu plochu.

Warnockov algoritmus pracuje v zobrazovacom priestore (teda v SSZ t.j. v priemetni), ktorý je tvaru obdĺžnika. Princíp spočíva v delení zobrazovacieho priestoru na 4 rovnaké časti. Ďalší postup je nasledovný:

1. Uloženie základného zobrazovacieho priestoru, teda jednej z tých 4.častí do zásobníka.
2. Výber zo zásobníka a sprístupnenie informácií, ktoré definujú veľkosť okna.
3. Ohodnotenie a rozhodnutie, vzhľadom na jednotlivé možnosti, ktoré môžu nastať pri kreslení telies po delení v príslušnom obdĺžniku (okne). Možnosti:
 - ❑ V okne sa nachádza mnohoúholník pokrývajúci daný obdĺžnik.
 - ❑ Mnohouholník je mimo obdĺžnika.
 - ❑ Ostatné prípady.
4. Keď sa jedná o možnosti a) alebo b), potom vieme jednoznačne rozhodnúť o viditeľnosti (výpočet vzdialenosti a rozhodnutie je v 3D). Nastáva vykreslenie.
5. Keď nevieme rozhodnúť o viditeľnosti, teda jedná sa o možnosť c), potom sa robí ďalšie (vlastne rekurzívne) delenie príslušného okna na 4 rovnaké časti. Vypočítajú sa súradnice jednotlivých okien a každé okno sa uloží do zásobníka.
6. Keď rozmery okna sú rovné jednej, tzn. že sa jedná o bod (pixel), dá sa jednoznačne rozhodnúť, ktorý bod je viditeľný a ten sa vykreslí.
7. Keď sa zásobník vyprázdni, ukončí sa celý algoritmus.
8. V opačnom prípade sa pokračuje v kroku 2.



Obr. 78 Postup delenia priestoru pri Warnockovom algoritme

Pri orezaní v tomto algoritme je možné použiť napr. Cohen-Shuterlandov algoritmus

8.4 Freeman-Loutrellov algoritmus

Patrí medzi klasické algoritmy, ktoré radíme medzi tie, čo riešia viditeľnosť ešte v 3D t.j. v rámci USS a SSC. Výborne sa hodí na riešenie viditeľnosti mnohostenov. Podstatou algoritmu je triedenie hrán na viditeľné a neviditeľné podľa polohy stien voči stredu premietania (hrana je vlastne priesečníkom príslušných stien). Algoritmus má pre konvexné telesá (uhol medzi dvomi stenami je menší ako 180°) dva kroky, pre nekonvexné telesá tri kroky. Tento algoritmus je časovo značne náročný a jeho náročnosť je daná zložitou riešenou 3D scénou.

V prvom kroku sa najprv určia normály stien, ktoré tvoria scénu (resp. objekt). Tieto sa určia vektorovým súčinom dvoch vektorov ležiacich v rovine steny (napr. pri nasledujúcom obrázku pre stenu (polygón) $ABGF$ môžu tieto vektory definovať hrany AF a AB). Vektorový súčin dvoch vektorov $\mathbf{a}(a_x, a_y, a_z)$ a $\mathbf{b}(b_x, b_y, b_z)$ je určený nasledovne:

$$\mathbf{a} * \mathbf{b} = (a_y \cdot b_z - a_z \cdot b_y, a_z \cdot b_x - a_x \cdot b_z, a_x \cdot b_y - a_y \cdot b_x) \quad (117)$$

Na základe toho sa určí uhol, ktorý zvierá každá normála so smerom premietania. Pre určenie uhla dvoch vektorov je možné použiť napr. vzťah (ak uvažujeme spomínané vektory):

$$\varphi = \arccos \left(\frac{a_x \cdot b_x + a_y \cdot b_y + a_z \cdot b_z}{\sqrt{a_x^2 + a_y^2 + a_z^2} \cdot \sqrt{b_x^2 + b_y^2 + b_z^2}} \right) \quad (118)$$

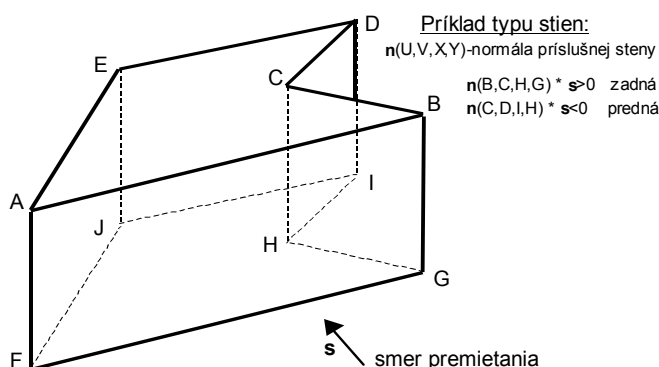
V praxi častokrát postačuje určovanie uhlov medzi priemetmi priamok, ktoré definujú vektor pohľadu pozorovateľa a normálový vektor steny (polygónu). Podľa charakteru definície scény postačí buď len priemet do pôdorysu, alebo ešte aj napr. do nárysu.

Na základe príslušných určených uhlov rozdelíme steny na "predné" a "zadné":

- *Predná stena*, potenciálne viditeľná, zvierá so smerom premietania tupý uhol.
- *Zadná stena*, ktorá je neviditeľná, zvierá so smerom premietania ostrý uhol.

Druhý krok zase triedi hrany. Hrany, podľa typov stien, ktoré ich vytvárajú, môžu byť v štyroch kategóriách:

- priesečník *Predná-Predná* je potenciálne viditeľná hrana
- priesečník *Predná-Zadná* a uhol medzi stenami (vo vnútri mnohoúhelníka) je menší ako 180° , potom je hrana opäť potenciálne viditeľná.
- priesečník *Predná-Zadná* a uhol medzi stenami (vo vnútri mnohoúhelníka) je väčší ako 180° , potom je hrana neviditeľná.
- priesečník *Zadná-Zadná* je neviditeľná hrana.



Obr. 79 Viditeľnosť mnohostenov

Tretí krok sa aplikuje len u nekonvexných mnohostenov. Totiž pre konvexné mnohosteny sú hrany typu 1 a 2 viditeľné. Pri nekonvexných telesách teda v tomto kroku ešte pracujeme s potenciálne viditeľnými hranami. Je potrebné zistiť, či niektorá stena plne alebo čiastočne nezakrýva skúmanú hranu. Toto nastáva vtedy ak:

- priemet hrany leží v priemete steny
- stena je medzi hranou a stredom premietania

Potom bod, v ktorom dochádza k zmene viditeľnosti je daný v priemete t.j. v SSZ priesečníkom skúmanej hrany a príslušnej hrany steny.

8.5 Algoritmus pamäte hĺbky (Z-buffer)

Tento algoritmus sa tiež nazýva aj *Depth-buffer* algoritmus alebo *Z-buffer algorithm*. Ako už názov napovedá, pri riešení viditeľnosti týmto algoritmom, sa jedná o tzv. hĺbkovú spojitosť. Ako metóda riešenia viditeľnosti je veľmi jednoduchá a robustná zároveň (z tohto dôvodu je najčastejšie hardvérovo implementovaná a v súčasnosti patrí k takmer štandardným vybaveniam grafických akcelerátorov samozrejme s požiadavkou dostatočnej pamäťovej kapacity umiestnenej na grafickom adaptéri, ktorú môže pre tento účel akcelerátor využiť). Pracuje na úrovni jednotlivých pixelov obrazovky. Pre každý pixel, ktorý sa ide kresliť, je nutné poznať jeho vzdialenosť od pozorovateľa. Ak je jeho vzdialenosť menšia ako vzdialenosť už nakresleného pixela na tom istom mieste obrazovky, daný pixel sa nakreslí a jeho vzdialenosť sa uloží do pamäte hĺbky pre ďalšie prípadné testovanie. Ak je jeho vzdialenosť väčšia, pixel sa nekreslí. Metóda vyžaduje mať v pamäti pole rozmerov ($maxX \ maxY$), kde sú uložené vzdialenosti jednotlivých (už nakreslených) pixelov na obrazovke.

Tento algoritmus je určený pre rastrové displeje a patrí do rovnakej triedy ako Warnockov algoritmus. Navyše mu je možné pripísať jednoduchosť a nezávislosť na zložení riešenej scény, tzn. že doba výpočtu je závislá len na počte objektov (najčastejšie mnohouholníkov) a nie je závislá na ich vzájomných vzťahoch. Veľkou nevýhodou tohto algoritmu je pomerne značná pamäťová náročnosť. Toto závisí od rozlišovacej schopnosti, s ktorou počítame napr. pri 640x480 pixelov je potrebné cca 600KB pamäti. Tento algoritmus je možné zefektívniť pri rovinných mnohouholníkoch a využiť znalosť o hĺbke v jednom bode mnohouholníka aj na určenie hĺbky ostatných bodov (vtedy hovoríme o tzv. hĺbkovej spojitosti). Samotný algoritmus je v podstate trojkrokový.

1. V prvom kroku sa deklarujú dve dvojrozmerné polia v rozsahu definovanom rozlišovacou schopnosťou, v ktorej viditeľnosť riešime (napr. 640x480). Príslušné súradnice X, Y riešeného pixelu sú dané príslušnými indexami I, J v rámci oboch polí. Prvé pole označme Z a príslušná položka Z_{IJ} obsahuje vzdialenosť najbližšieho bodu 3D scény zo všetkých bodov, ktoré sa do daného pixelu premietnu. Druhé pole je označované F a príslušná položka F_{IJ} obsahuje informácie o farbe (jase, príp. osvetlení) daného pixelu.
2. V druhom kroku sú všetky položky poľa Z nastavené na maximálnu hodnotu súradnice Z a položky poľa F na farbu pozadia. Týmto spôsobom je možné do pozadia dať rôzne podklady.
3. V poslednom treťom kroku sú pre každý mnohouholník mnohostenu t.j. polygón (príp. objekt) nájdené všetky pixely, do ktorých sa mnohosten (polygón, objekt) zobrazí. Potom sú určené hodnoty Z_{IJ} a F_{IJ} . Nasledovne sa pre každý pixel u každého mnohouholníka (polygónu) určí:
 - súradnica z t.j. priesečník premietacej priamky, ktorá prechádza pixelom a mnohouholníkom (polygónom). Pri tomto určovaní pre každý pixel je

možné použiť už spomínanú tzv. *hlbkovú spojitosť*. Ak je daný bod $X_0(x_0, y_0, z_0)$ a súradnice x_i, y_i iného bodu X_i určitého rovinného mnohouholníka (polygónu), potom na určenie zostávajúcej z-ovej súradnice je možné využiť spomínanú hĺbkovú spojitosť. Nech všeobecná rovnica roviny mnohouholníka je: $a.x + b.y + c.z + d = 0$.

Potom: $a.x_i + b.y_i + c.z_i + d = a.x_0 + b.y_0 + c.z_0 + d = 0$

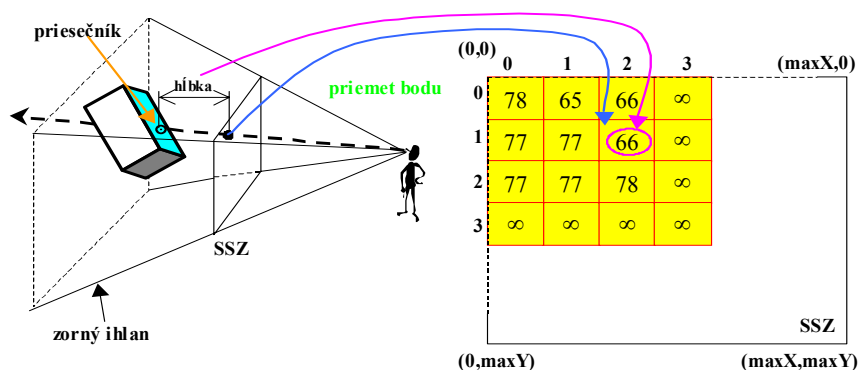
jedinou neznámou je z_i , ktoré je možné z predchádzajúcej rovnice vyjadriť:

$$z_i = \frac{-a}{c} \cdot (x_i - x_0) + \frac{-b}{c} \cdot (y_i - y_0) + z_0 \quad (119)$$

Na určenie z-ovej súradnice ľubovoľného bodu mnohouholníka (polygónu) stačí teda poznať súradnice x, y tohoto bodu, konštanty a, b, c všeobecnej rovnice roviny mnohouholníka (polygónu) a jeden bod (napr. vrchol), patriaci rovine mnohouholníka (polygónu).

- ak je mnohouholník (polygón) bližšie ako predtým skúmaný mnohouholník (polygón) ($z < Z_L$) potom sa $Z_L = z$ a určí sa farba (príp. jas, či osvetlenie) podľa nového pixela a uloží sa do F_L .

Po skončení činnosti nad všetkými mnohouholníkmi je potom v poli F úplná informácia o farbe (jase či osvetlení) všetkých mnohouholníkov a viditeľnosť je vyriešená.



Obr. 80 Príklad pre Z-buffer

Teraz si uvedieme príklad. Situáciu demonštruje predchádzajúci obrázok. Pixel na súradniciach $[2,1]$ je už nakreslený a je vo vzdialenosti 66. Pixel $[2,2]$ je zatiaľ prázdny a jeho vzdialenosť je inicializovaná na nekonečno. Algoritmus pamäte hĺbky zabezpečí, že sa na súradniciach $[2,1]$ nebude kresliť nič, čo je vo vzdialenosti väčšej ako 66. Dôsledkom je, že výsledná obrazovka obsahuje v každom bode pixel, ktorý je na danej súradnici najbližšie pozorovateľovi (čo je vlastne podstatou riešenia viditeľnosti). Pri aplikácii pamäte hĺbky v kombinácii s perspektívnou zobrazovacou transformáciou je potrebné pri kreslení polygónu dávať pozor na skreslenie, ktoré zobrazovacia transformácia prináša, pretože súradnica z sa po priamke na priemetni nemení lineárne, ale po hyperbole. Preto je nutné používať (a do pamäte ukladať) hodnotu $1/z$.

V nasledujúcom si zhrnieme výhody a nevýhody tohto algoritmu. Medzi výhody tohto algoritmu patria:

- korektne rieši viditeľnosť
- polygóny (mnohouholníky) sa môžu navzájom pretínať

Poznamky

- ❑ polygóny je možné kresliť v ľubovoľnom poradí (nevyžaduje žiadne predspracovanie ani triedenie polygónov)
- ❑ vhodný pre objekty s množstvom malých polygónov

Medzi nevýhody patria najmä:

- ❑ vysoké nároky na pamäť, minimálne ($2 \cdot \max X \cdot \max Y$) bajtov
- ❑ prekresľovanie
- ❑ je pomalý (málo efektívny) pri veľkých scénach (veľa veľkých polygónov), je ho možné však urýchliť použitím orezania na zorný ihlan.

Tento algoritmus je veľmi efektívny a nenáročný na programovanie. V praxi sa zvykne používať namiesto vyspelých algoritmov (napr. BSP) vtedy, ak by zložité výpočty prienikov polygónov alebo triedenie nepriniesli žiadne zrýchlenie, prípadne by celý proces riešenia viditeľnosti spomalili. Príkladom môže byť rozľahlá miestnosť (100 polygónov) a uprostred stojaci objekt (napríklad kvetináč) zložený z tisíc polygónov. Takéto prípady je jednoduchšie a častokrát rýchlejšie riešiť osobitne: metódou BSP pre miestnosť a metódou Z-buffer (v rámci SSO) pre kvetináč. Veľký počet polygónov v kvetináči môže totiž spôsobiť neúmerne delenie polygónov celej scény pri použití len BSP. Situácia sa ešte zhorší, ak by sa kvetináč pohyboval. V takom prípade by bolo značne neefektívne zakaždým vkladať do BSP stromu tisíc polygónov len preto, aby sa pixely neprekresľovali. Vtedy je Z-buffer výhodným riešením.

8.6 Algoritmus riadkového rozkladu



Tomuto algoritmu sa tiež hovorí aj *scan-line algoritmus* (v spolupráci s technikou s-bufferu sa niekedy označuje aj x-buffer). Jedná sa o pomerne rýchly algoritmus, ktorý vychádza z hĺbkového algoritmu. Tento algoritmus je asi 2-3-krát rýchlejší ako skôr popisovaný Warnockov algoritmus. Algoritmus je v podstate dvojkrokový. Rozdiel medzi hĺbkovým algoritmom je v tom, že tu sa v pamäti neudržiava informácia o viditeľnosti celej scény a tým pádom informácia o všetkých pixeloch, ale spracúvava sa len jeden riadok pixelov. Keď sa určí viditeľnosť v jednom riadku pixelov, je následne tento riadok vykreslený a začne sa spracovávať ďalší riadok. Pri určovaní viditeľnosti je možné využiť fakt, že viditeľnosť riadku bodov (pixelov) je možné určiť viditeľnosťou v jednom z daných pixelov. Tejto vlastnosti hovoríme aj *riadková súvislosť*. Na základe tejto skutočnosti je možné značne zrýchliť algoritmus. Celý algoritmus predpokladá, že všetky mnohoúhelníky (objekty) sú uložené vo forme tabuľky hrán, ktoré jednotlivé mnohoúhelníky tvoria. Túto tabuľku budeme nazývať *TH*. Každá hrana je premietnutá ako úsečka a tým pádom môže byť definovaná svojimi krajnými bodmi (x_1, y_1) a (x_2, y_2) s podmienkou, že $y_2 > y_1$. Označme:

$$K = \frac{x_2 - x_1}{y_2 - y_1} \quad (120)$$

potom rovnica priamky, na ktorej leží naša úsečka je:

$$y = y_1 + \frac{1}{K} \cdot (x - x_1) \quad (121)$$

V našom prípade je v podstate *y-ovou* súradnicou definovaný príslušný riadok (pri rastrových displejoch a ich rozlišovacej schopnosti). Označme potom riadok, ktorý pretína naša úsečka *i*. Dosadením do uvedeného vzťahu dostaneme:

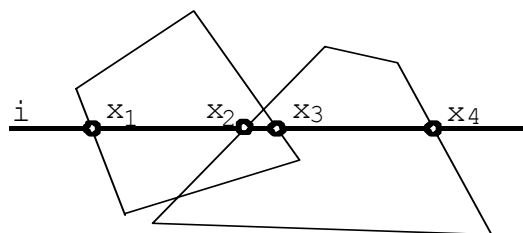
$$i = y_1 + \frac{1}{K} \cdot (x - x_1) \quad (122)$$

Pre *i+1* riadok platí obdobná rovnica. Úpravou dostaneme iterčný vzťah:

$$x_{i+1} = x_i + K \quad (123)$$

Týmto jednoduchým vzťahom je možné dosiahnuť určenie priesečníka priemetu príslušnej hrany a príslušného ďalšieho riadku len inkrementáciou (tzv. riadková súvislosť). Takýmto spôsobom určíme z rozsahov priemetov hrán rozsahy priemetov príslušných mnohoúholníkov. Častokrát sa údajová štruktúra tabuľky *TH* upravuje do komplexnejšieho tvaru, kde je ešte uložený aj predpočítaný koeficient *K* a prípadne aj smerník na mnohoúholník (polygón), ku ktorému hrana prislúcha. Týmto je v zásade definovaný prvý krok algoritmu.

Druhý krok je v podstate určovanie viditeľnosti v riadku pre jednotlivé úseky, ktoré sú ohraňované priesečníkmi hrán a riadku. Tieto sme určili v prvom kroku. Predpokladajme situáciu podľa nasledujúceho obrázku.



Obr. 81 Riadkový rozklad

Vidíme, že náš *i*-ty riadok pretínajú štyri priemety hrán (*h₁-h₄*) označené (*x₁, x₂, x₃, x₄*). Algoritmus postupne vyhodnocuje viditeľnosť v úsekoch *x₁x₂*, *x₂x₃* a *x₃x₄*. Vyberá a testuje len tie mnohoúholníky, ktoré sa do daného úseku premietajú a vyberie ten najbližší (je možné využiť hĺbokú spojitosť, pozri vzťah (119)). Ten je nakoniec viditeľný a vykreslený, pričom jeho sklon voči stredu premietania definuje jeho zobrazenie. Principiálne ide o jeden cyklus. Na začiatku definujeme údajovú štruktúru zásobník a označme ju *ZM*. Tento zásobník bude slúžiť na ukladanie všetkých mnohoúholníkov, ktoré sa do daného úseku zobrazujú. *ZM* sa začiatku vynuluje. Samotný cyklus prebieha napr. pre premennú *j*, ktorá postupne nadobúda hodnoty od 1 po *n-1*, kde *n* je počet priesečníkov priemetov hrán (v našom prípade *n=4*). Potom berieme postupne všetky úseky *x_j, x_{j+1}*. Určí sa hrana *h*, ktorá prislúcha priesečníku *x_j* a následne sa určí zodpovedajúci mnohoúholník (polygón). Ak daný mnohoúholník (polygón) nie je v *ZM*, zaradíme ho do *ZM*. Ak daný mnohoúholník (polygón) je v *ZM* potom ho odtiaľ vyradíme. Určí sa najbližší mnohoúholník (polygón) v bode *x_j* z mnohoúholníkov (polygónov) uložených v zásobníku *ZM*. Nakoniec sa zobrazí úsek (*x_j, x_{j+1}*) podľa určeného najbližšieho mnohoúholníka a cyklus pokračuje ďalším *j*. Inak je cyklus ukončený. Takýto cyklus vykonáme pre všetky riadky. Po prejdení všetkých riadkov je viditeľnosť vyriešená.

Záverom je nutné povedať, že daný algoritmus je vhodný okrem riešenia viditeľnosti objektov s rovinnými plochami aj pre objekty s plochami nerovinnými (napr. pri zobrazovaní plôch plátovaním).

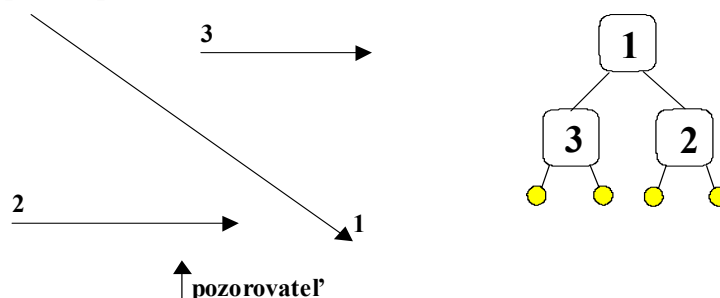
8.7 Metóda BSP stromov



Binary Space Partitioning (BSP) je metóda hierarchického rozkladu priestoru na konvexné podpriestory a je hojne využívaná pri vizualizácii pre svoje skvelé vlastnosti. BSP stromy sú štruktúry veľmi vhodné na použitie v počítačovej grafike, predovšetkým na rýchle triedenie, vyradovanie odvrátených plôch a raytracing. Tvorba BSP stromu je výpočtovo náročný proces a spočíva v delení podpriestoru vždy na dve časti, ktoré sa potom ďalej rekurzívne delia. Celá štruktúra vzniká v čase predspracovania a pri renderingu v reálnom čase sa (až na výnimky) nemení. Použitie BSP stromu výrazne urýchľuje riešenie viditeľnosti. Podľa toho koľko stupňov voľnosti pri vizualizácii poskytneme (6 alebo menej) v nepriamej úmERE ku časovej náročnosti je možné použiť 3D-BSP stromy (6 stupňov voľnosti t.j. neobmedzený pohyb pozorovateľa) alebo 2D-BSP stromy (menej ako 6 stupňov).

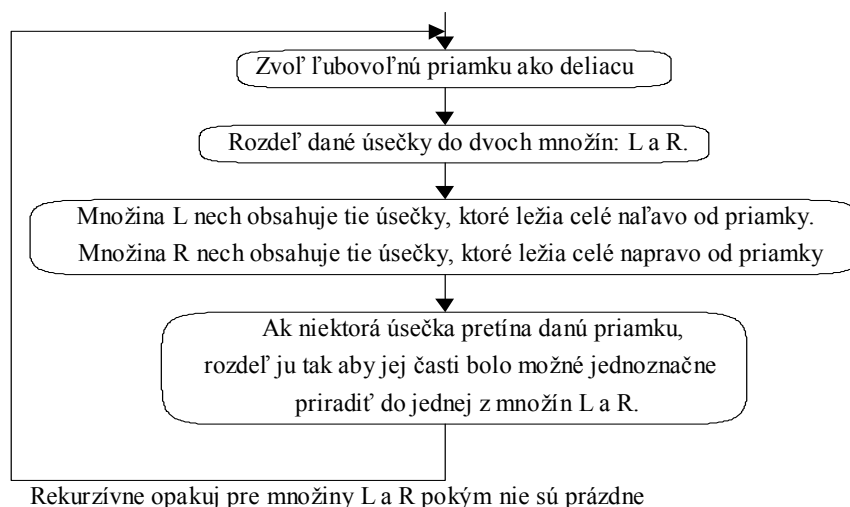
8.7.1 Tvorba a prechod BSP stromu

Konštrukciu BSP stromu je možné vysvetliť na nasledujúcom príklade riešenia viditeľnosti v rovine (2D-BSP). Majme tri orientované úsečky označené 1, 2 a 3 (viď. nasledujúci obrázok). Šípkou je naznačená poloha a smer pohľadu pozorovateľa na scénu (použitím maliarovho algoritmu by sa úsečky vykreslili zozadu dopredu v poradí 3,1,2).



Obr. 82 2D Scéna a jej BSP strom

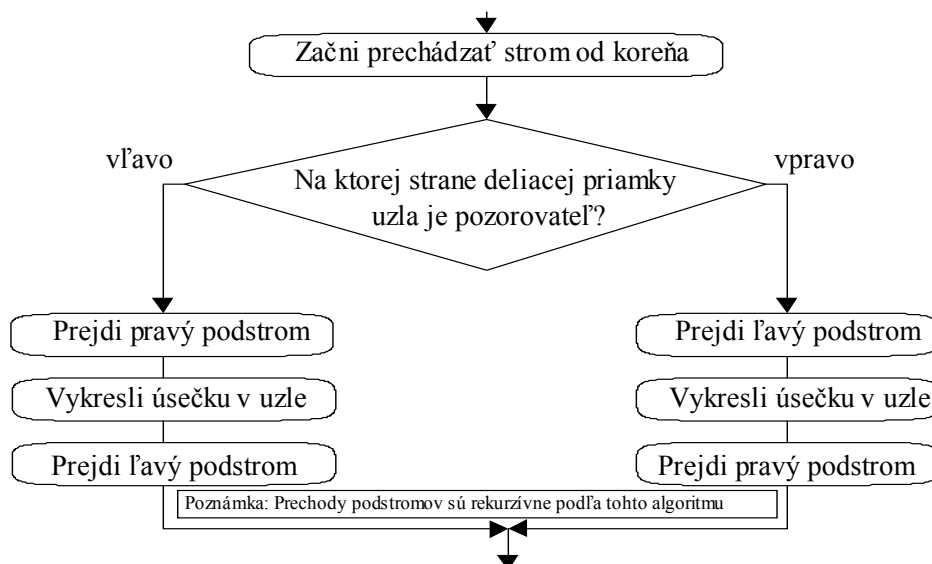
Pri tvorbe 2D BSP stromu je algoritmus nasledovný:



Obr. 83 Algoritmus tvorby BSP stromu

V prvom kroku je možné vo všeobecnosti voliť ľubovoľnú priamku, pre riešenie viditeľnosti je možné (a zvyčajne aj výhodné) zvoliť jednu z priamok zo scény. V ďalšom texte sa tento postup predpokladá.

Ak by sa zvolila za deliacu priamku úsečka 1, výsledný BSP strom scény na predchádzajúcom obrázku by bol taký, aký je naznačený na Obr. 82. Úsečka 2 leží celá vpravo od úsečky 1 a úsečka 3 leží celá vľavo od úsečky 1.



Obr. 84 Algoritmus prechodu BSP stromom zozadu dopredu

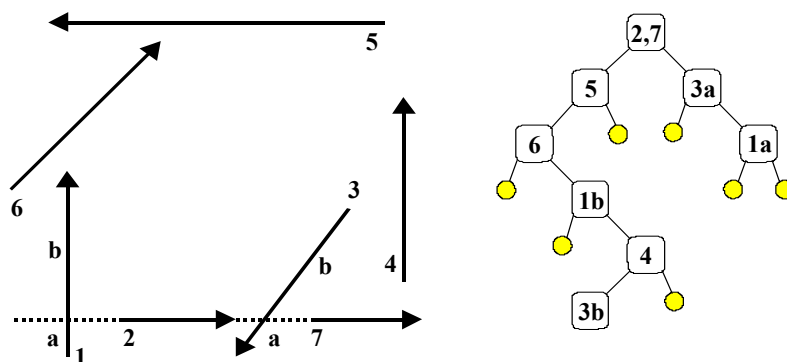
Tento strom sa teraz dá využiť napríklad na vyriešenie viditeľnosti v scéne na Obr. 82. Stačí iba podľa algoritmu na prechádzajúcom obrázku prejsť BSP stromom.

Prechod stromu scény je podľa algoritmu 3.1.2 čo presne súhlasí s postupom kreslenia scény pri maliarovom algoritme. BSP je pri riešení viditeľnosti vlastne rozšírením maliarovho algoritmu s tým, že nie je potrebné úsečky utriediť podľa vzdialenosti od pozorovateľa, ale poradie kreslenia je určené len prejdením BSP stromu. Tol'ko k podstate tvorby stromu, teraz ukážeme jeho ďalšie vlastnosti.

8.7.2 Vlastnosti BSP

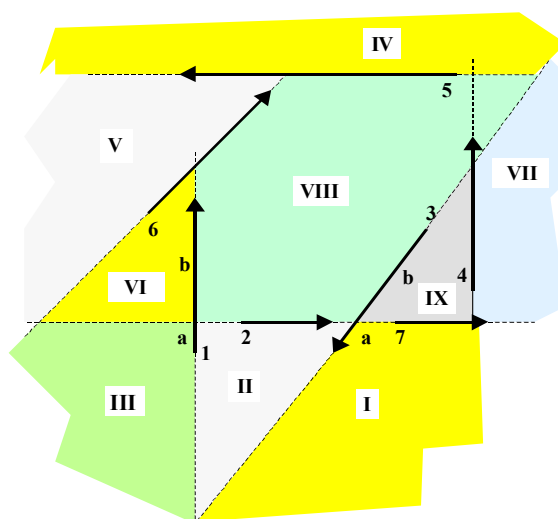
V scéne na nasledujúcom obrázku je 6 úsečiek a koreň stromu je úsečka 2. Keďže úsečky 1 a 3 neležia celé ani vľavo ani vpravo od úsečky 2, musia sa rozdeliť na dve časti, úsečka 1 na 1a 1b a úsečka 3 na 3a 3b. Týmto sa úsečky rozdelia na množiny $L = \{1b, 3b, 4, 5, 6\}$ a $R = \{1a, 3a\}$. Úsečka 7 je špeciálny prípad, lebo leží na tej istej priamke ako úsečka 2. Neleží ani napravo ani naľavo, leží "paralelne" s ňou. V takomto prípade je na ľubovôli tvorcu stromu, ako s úsečkou 7 naloží. Môže prehlásiť, že leží na jednej zo strán úsečky 2 a môže ju dokonca umiestniť do toho istého uzla stromu. Je to preto, lebo v uzloch sa nachádzajú deliace priamky a nie úsečky a úsečky 2 a 7 ležia na tej istej (deliacej) priamke. Ďalším delením množín L a R môže vzniknúť napríklad BSP strom napr. zobrazený na nasledujúcom obrázku.

Poznámky:



Obr. 85 Zložitejšia scéna a jej BSP strom

Ak sa pozorovateľ nachádza v strede scény, tak prechodom BSP stromu podľa algoritmu prechodu BSP stromom zozadu dopredu je postupnosť úsečiek 3a, 1a, 2, 7, 5, 6, 1b, 4, 3b. Je zrejmé, že sa takýmto poradím kreslenia úsečiek docielu prekrytie vzdialenejších úsečiek bližšími a tým aj správna viditeľnosť. Pritom nezávisí na smere pohľadu pozorovateľa! Ak je navyše úsečka viditeľná len z jednej strany (je jednostranná), tak prechodom BSP stromu sa nielenže vyrieši viditeľnosť, ale jeho prechod je vlastne "zadarmo", pretože testovanie, či je pozorovateľ na privrátenej alebo odvrátenej strane úsečky by sa muselo aj tak vykonať (výsledok tohto testu je priamo k dispozícii v kroku 2 algoritmu prechodu BSP stromom).

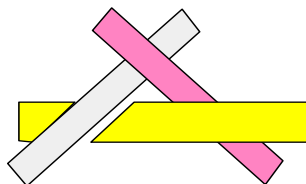


Obr. 86 Hierarchické rozdelenie priestoru

V BSP strome existujú 2 typy entít: *uzly* a *listy*. Uzly sú v diagramoch označené štvorčkami a listy krúžkami. Uzol predstavuje útvar, ktorým sa priestor delí a list predstavuje konvexný podpriestor. V prípade scény na obr. Obr. 85, keďže sa pracuje v rovine, obsahujú uzly deliace priamky a listy zasa konvexné polygóny, ktoré vznikli delením roviny pri tvorbe stromu. Situáciu znázorňuje predchádzajúci obrázok. V strome je 9 listov, teda rovina je rozdelená na 9 častí. Na samom začiatku je spomenuté "hierarchické" delenie priestoru a táto vlastnosť vyjde na povrch ak si uvedomíme, že uzol (2,7), ktorý je zároveň aj koreňom stromu delí rovinu na dve časti. Prvá časť je zjednotením konvexných útvarov z ľavej polovice stromu (polygóny V, VI, VII, VIII a IX) a druhú časť tvorí zjednotenie polygónov z pravej časti stromu (polygóny I, II a III). Teda uzol predstavuje (okrem deliaceho útvaru) aj podpriestor tvorený zjednotením podpriestorov v jeho poduzloch.

8.7.3 Použitie BSP

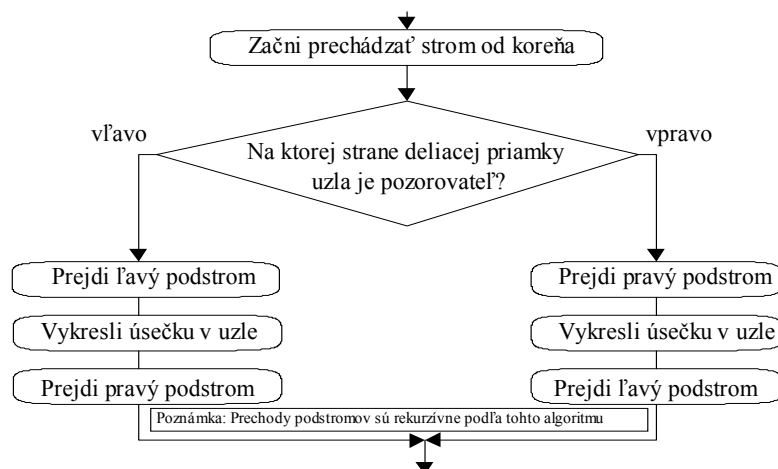
Je potrebné zdôrazniť, že na rozdiel od maliarovho algoritmu rieši BSP viditeľnosť vždy korektne, čo je možné demonštrovať na príklade rovnobežníkov v 3D priestore na Obr. 77b. BSP tento problém odstráni napríklad rozdelením spodného rovnobežníka na dve časti (ako je znázornené na nasledujúcom obrázku). Po rozdelení je už možné nájsť správne poradie kreslenia týchto (teraz už štyroch) útvarov.



Obr. 87 Scéna s cyklickým prekryvom po rozdelení pomocou BSP

BSP je veľmi vhodná technika na riešenie viditeľnosti v priestore. Pri BSP v 3D sa namiesto úsečiek použijú polygóny a namiesto deliacich priamok deliace roviny. Samotný algoritmus je v 3D rovnaký a vo všeobecnosti použiteľný pre N-rozmerný priestor s N-1 rozmernými deliacimi útvarmi.

Použitím algoritmu (prechod BSP stromom) sa dosiahne rovnaký efekt ako pri maliarovom algoritme, teda vzdialenejšie polygóny budú prekryté bližšími. Je však efektívnejšie, ak sa už nakreslené polygóny neprekresľujú a ak sa toto zabezpečí, je výhodnejšie použiť nasledujúci algoritmus.



Obr. 88 Algoritmus prechodu 3D BSP stromom spredu dozadu

Pri tomto postupe sa budú najprv kresliť polygóny najbližšie k pozorovateľovi. Ošetrenie neprekresľovania je síce práca navyše, ale v konečnom dôsledku stúpne výkon niekoľkonásobne, pretože strom nie je potrebné prejsť celý, ako pri algoritme na Obr. 84, a navyše sa každý pixel obrazovky kreslí iba raz, čo hrá veľkú úlohu pri náročnom výpočte farby pixelu napríklad pri textúrovaní. Samozrejmosťou je, že prechod stromom je výhodné čo najskôr zastaviť, akonáhle je nakreslená celá obrazovka.

Navyše pri prechode stromom je možné za určitých podmienok, vyplývajúcich z konkrétnej aplikácie algoritmu (napríklad použitím hraničných útvarov pre uzly), vynechať celé vetvy stromu naraz a tým algoritmus prechodu ešte zrýchliť.

V procese tvorby BSP stromu vystupuje do popredia problém voľby vhodnej deliacej roviny pričom v zásade existujú dve stratégie. Buď je potrebné mať strom čo najviac vyvážený, alebo je vhodné obmedziť počet delení polygónov na minimum. Sú to protichodné požiadavky a závisí od použitia BSP stromu pre konkrétne účely. Konštrukcia napríklad maximálne vyváženého stromu je však NP-úplný problém a preto je vo väčšine prípadov potrebné sa uspokojiť pri hľadaní najvhodnejšieho koreňa s rôznymi heuristikami a aproximáciami. Ak sa BSP využíva na riešenie viditeľnosti je výhodné minimalizovať počet delení polygónov na minimum, pretože často je potrebné prejsť veľkú časť stromu a navyše kreslenie štyroch či piatich polygónov (ktoré vznikli rozdelením jediného) je oveľa zdĺhavejšie ako kreslenie polygónu vcelku.

8.7.4 Výhody a nevýhody BSP

Výhody:

- korektne rieši viditeľnosť pri zložitosti $O(n)$ pri ľubovoľnej polohe pozorovateľa
- hierarchické rozdelenie priestoru na konvexné podpriestory, čo pomáha pri prechode stromom, kedy je možné vynechať naraz celý podstrom
- umožňuje kresliť zozadu dopredu aj spredu dozadu

Nevýhody:

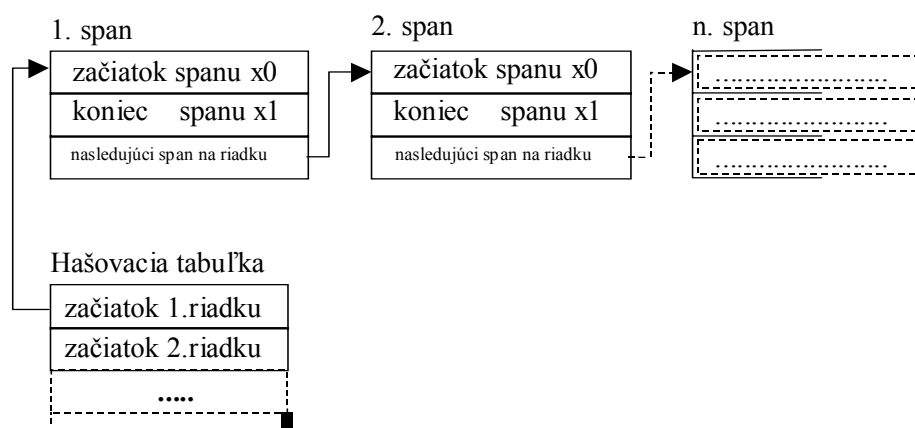
- takmer nulová flexibilita, pretože pri pohybe objektov (úsečiek, polygónov) sa mení ich vzájomná poloha a tým aj usporiadanie v strome, a preto je nutné odznova prepočítať celý BSP strom (prípadne jeho časť), čo pri systémoch pracujúcich v reálnom čase neprichádza do úvahy. Čiastočným riešením (pri veľkých, pohybujúcich sa objektoch) je vkladanie polygónov do stromu v reálnom čase. V predpočítanom strome sú vtedy len statické objekty.
- pri tvorbe stromu dochádza zákonite k zvýšeniu počtu objektov v scéne spôsobeného delením.
- náročný preprocessing.

8.8 S-buffer



S-buffer (angl. span-buffer) je jednou z techník, ako zabezpečiť neprekresľovanie už vykreslených polygónov v rámci SSZ, jedná sa teda o algoritmus, ktorý pracuje prioritne v 2D. Jej použitie sa teda obmedzuje na použitie v systémoch, ktoré kreslia scénu spredu dozadu. Údajové štruktúry, ktoré sa používajú pri S-buffri sú rôzne. Často sa však používa najmä spojkový zoznam, ako je znázornené na nasledujúcom obrázku. Ide teda o spojkový zoznam častí riadkov (angl. span), ktoré sú už vykreslené. Takýto zoznam je zostavený osobitne pre každý riadok obrazovky a smerníky na prvý span na každom riadku sú uložené v hašovacej tabuľke.

Pri kreslení polygónu (mnohouholníka) sa musí každý span podrobiť testu, či nie je prekrytý iným, už nakresleným spanom. Ak je prekrytý, t.j. leží celý v rozmedzí x_0 a x_1 nejakého spanu v danom riadku, tak sa nekreslí. V opačnom prípade je potrebné počítať prieniky spanov a vykresliť len tie časti, ktoré sú ešte nevyskreslené, t.j. postupne zaplňovať diery v S-buffri. Rutina na vkladanie spanu nie je vôbec jednoduchá a v značnej miere na jej efektívnosti závisí aj výkon vizualizačného systému.



Obr. 89 Štruktúra S-buffra a hašovacej tabuľky

Ako už bolo uvedené používané údajové štruktúry sú rôzne, tým sa aj líšia možnosťami efektívneho evidovania informácií pre riadok. Záverom si zhrňme v krátkosti možnosti reprezentácií:

- Pre každý riadok na obrazovke sa vytvára *spojkový zoznam* (môže byť alokovaný dopredu, v prípade potreby sa pridávajú ďalšie členy). Výhodou je pomerne malá pamäťová náročnosť, za ktorú sa platí pomalším spracovaním.
- Pre každý riadok na obrazovke sa v pamäti nachádza *pole konštantných rozmerov* (teda pre určený maximálny počet obsadených úsekov na riadok). Táto možnosť je jednoducho implementovateľné a rýchla, nevýhodou je trvale alokované pole. Ale pre väčšinu grafických systémov je rýchlosť dôležitejšia.
- Informácie o obsadenosti riadku sú uložené v *BSP strome*. V tomto prípade je to veľmi zjednodušený model 1-rozmerného BSP stromu. Nevýhodou je zložitejšia správa blokov (tvorba stromu), ale tá je vyvážená vysokou rýchlosťou spracovania najmä pri väčšom počte blokov obsadenia pre jeden riadok.
- Posledná uvedená metóda sa dá rozšíriť na *2D BSP strom*, pomocou ktorého je možné riešiť dvojrozmerné pokrývanie celej obrazovky, nie len konkrétneho riadku. Vhodnosť tohto prístupu je silne závislá na zobrazovanej scéne ale niekedy tento algoritmus môže dávať veľmi dobré výsledky

8.9 Urýchľovacie technológie riešenia viditeľnosti

Prvou úlohou pri riešení viditeľnosti je vybrať z celej scény tie objekty (polygóny), ktoré budú (prípadne s určitou pravdepodobnosťou môžu byť) z danej pozície kamery viditeľné t.j. výberová fáza (pozri Obr. 72). Tento proces vyradovania veľkých častí scény, ktoré určite nebudú viditeľné je veľmi dôležitou časťou najmä vizualizačných systémov v reálnom čase, pretože tvorí prvý filter, ktorý by mal vyradiť z ďalšieho spracovania čo najviac "neviditeľných" objektov. Predpokladá sa, že časová náročnosť algoritmov v procese vizualizácie postupne narastá. Dobre navrhnutý prvý filter môže výraznou mierou prispieť k vysokému a hlavne rovnomernému výkonu systému.

8.9.1 3D orezávanie na zorný ihlan

Zorný ihlan (ihlan pohľadu, pozri Obr. 80) je časť priestoru, ktorá sa po zobrazovacej transformácii ($SSC \rightarrow SSZ$) zobrazí do obdĺžnika na priemetni, ktorý reprezentuje tú jej časť, ktorá v konečnom dôsledku bude viditeľná na zobrazovači (obrazovke) t.j. zorný priestor.

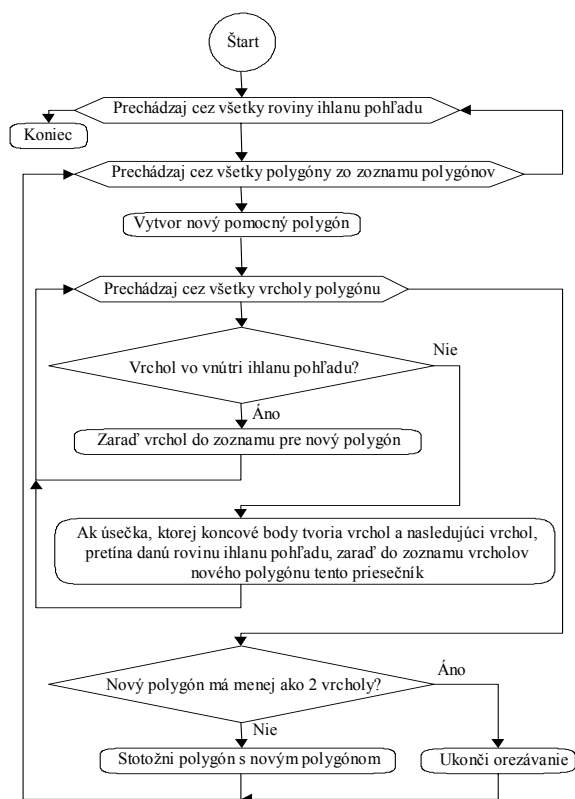
Pokiaľ je scéna, v ktorej sa rieši viditeľnosť, uzavretá (interiér stavby, bludisko, a pod.) je možné posunúť základňu ihlana do nekonečna, pretože je zaručené, že sa celá priemetňa "naplní". V prípade exteriérov je nutné si zvoliť určitú vzdialenosť a podstavu ihlana pohľadu umiestniť tam. Pri kreslení sa potom všetky polygóny a objekty za touto hraničnou vzdialenosťou automaticky vylúčia. Tým sa zabráni kresleniu polygónov "do nekonečna", čo by malo za následok neúmerne zvýšenie časovej náročnosti algoritmu.

Orezávanie je možné robiť buď v SSC alebo v USS . V prípade, že sa orezáva v SSC , je nutné najprv pretransformovať polygón do SSC (zorný ihlan v SSC je konštantný), v druhom prípade je zasa potrebné pretransformovať roviny samotného zorného ihlana do USS (statický polygón sa v USS nemení).

Roviny, ktorými je daný zorný ihlan (v SSC) sú:

- ❑ rovina rovnobežná s priemetňou vo vzdialenosti $Z = \varepsilon$ od kamery ($\varepsilon \rightarrow 0$)
- ❑ rovina rovnobežná s priemetňou vo vzdialenosti $Z = \max Z$ od kamery
- ❑ 4 roviny, ktoré prechádzajú vrcholom ihlana (okom pozorovateľa, kamerou) a spolu tvoria jeho plášť

Orezávanie polygónu prebieha postupne pre každú zo šiestich rovín podľa nasledovného algoritmu (platí obmedzene len pre konvexné polygóny).



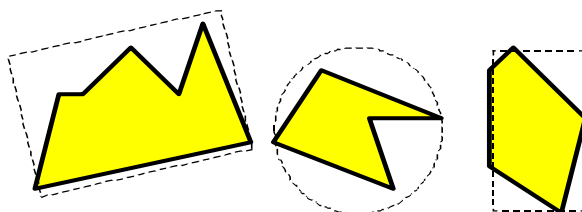
Obr. 90 Algoritmus orezávania polygónu na zorný ihlan

Pre odstránenie častí scény mimo zorný ihlan sa používa algoritmus orezávania polygónu plochou, ktorý je vlastne popísaný v kapitole 4.4.1. Pre tento algoritmus je potrebné vyjadriť plochy plášťa zorného ihlana a aplikovať orezanie scény každou z týchto plôch.

8.9.2 Ohraničujúce útvary

Použitie ohraničujúcich útvarov (angl. boundary volumes) je jednoduchou a často používanou metódou, ako rýchlo zistiť, či leží daný objekt vo viditeľnej časti priestoru (v ihlane pohľadu).

Každému objektu sa priradí ohraničujúci útvar a platí, že objekt celým svojím objemom leží vnútri neho. Ak je potrebné zistiť, či daný objekt leží v ihlane pohľadu, tak v prvom kroku sa otestuje jeho ohraničujúci útvar. Ak tento podmienku nespĺňa, celý objekt je možné z ďalšieho spracovania vylúčiť. V opačnom prípade je nutné sa objektom zaoberať podrobnejšie (test každého polygónu a pod.). Samozrejme sa predpokladá, že test s ohraničujúcim útvarom je oveľa jednoduchší ako test celého objektu, preto sa za ohraničujúce útvary volia základné 3D telesá ako guľa, kváder, pyramída a pod. Je zrejmé, že ich prínos narastá priamo úmerne so zložitou objektu, pretože jediným triviálnym testom je možné vyradiť z ďalšieho spracovania veľmi zložitý objekt. Príklad zostrojenia ohraničujúcich útvarov je na nasledujúcom obrázku. Každý z troch polygónov má svoju hranicu (obdĺžnik, prípadne kruh) a všetky sú ešte ohraničené ďalším obdĺžnikom. Teda ak padne do zorného uhla, testujú sa postupne hranice každého polygónu. Ohraničujúce útvary sú veľmi vhodné pri použití hierarchických štruktúr.



Obr. 91 Príklady ohraničujúcich útvarov

V tomto prípade k technológii ohraničujúcich útvarov je možné priradiť aj použitie oktantových stromov vlastne ako ohraničujúcich polpriestorov. 3D-scéna sa v rámci USS rozdelí pomocou troch rovín (XY, XZ a YZ) na osem polpriestorov. Tak ako sa u ohraničujúceho útvaru predpokladá, je potom jednoduchšie vykonať test na príslušný oktant. Niekedy je výhodné použitie aj viacúrovňových oktantových stromov (každý oktant je opäť delený na oktanty).

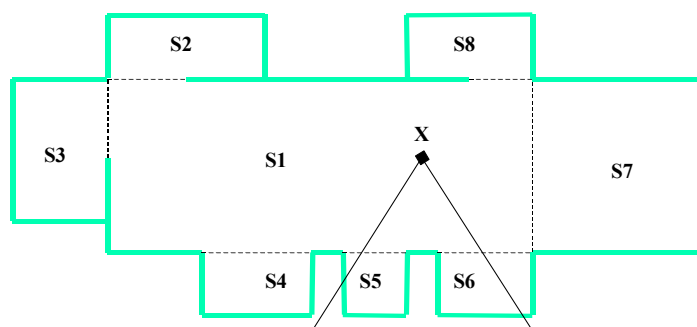
Z hľadiska ohraničujúcich útvarov je v popisovanej problematike výhodná aplikácia napr. na BSP stromy. Keďže BSP strom je prísne hierarchická štruktúra, ponúka sa možnosť výhodne spojiť vlastnosti oboch techník na zvýšenie výkonu vizualizačného programu. Každému uzlu stromu sa priradí jeden ohraničujúci útvar, ktorý v sebe obsahuje objekty zo všetkých jeho dcérskych uzlov. Výpočet rozmerov týchto útvarov sa robí zhora dole (od listov ku koreňu) a ohraničujúci útvar v každom uzle vznikne ako zjednotenie ohraničujúcich útvarov v jeho dvoch dcérskych uzloch.

Pri vizualizácii sa najprv skontroluje, či ohraničujúci útvar v uzle zasahuje do ihlana pohľadu. Môžu nastať tri prípady:

- ❑ Ohraničujúci útvar je celým svojím objemom v ihlane pohľadu. To znamená, že všetky objekty vo všetkých dcérskych uzloch daného uzla sú v ihlane pohľadu a nie je potrebné ich polohu (ani polohu samotných ohraničujúcich útvarov) vzhľadom na ihlan pohľadu ďalej skúmať.
- ❑ Ohraničujúci útvar je celým svojím objemom mimo ihlana pohľadu. V tom prípade sa prechod touto vetvou stromu končí.
- ❑ Ohraničujúci útvar je sčasti v ihlane pohľadu. Takýto výsledok neprináša žiadne zjednodušenie ďalších výpočtov.

8.9.3 Sektorovanie

Pri veľkom množstve objektov a polygónov scény, hlavne ak je to členitý interiér, zložený z množstva miestností, je výhodné ho rozdeliť na rad vzájomne prepojených sektorov. Ich použitie v prvej fáze vyradovania polygónov prispieva často k dramatickému zúženiu spektra všetkých objektov, ktorými sa má význam bližšie venovať. Príkladom môže byť situácia na Obr. 92. Miestnosť je rozdelená na sektory 1-8 (hranice sektorov sú znázornené čiarkovanými čiarami). Každý sektor obsahuje steny, ktoré sú v ňom (resp. tvoria jeho hranice) a ďalej špeciálne steny (nemusia byť viditeľné, často sú len imaginárne), ktoré tvoria hranicu medzi dvoma sektormi. Každá špeciálna stena má poznačené hranice, ktorých sektorov tvorí a ktorý sektor je na ktorej strane steny. Pri renderingu sa najprv zistí poloha pozorovateľa (na Obr. 92. označená písmenom X) a sektor, v ktorom sa nachádza. Steny z tohoto sektoru (ak sú pravda v ihlane pohľadu) sa zaradia do ďalšieho spracovania. Ak je niektorá z nich hranicou medzi sektormi, zoberú sa steny aj zo susedného sektoru a tak postupne pre všetky potenciálne viditeľné sektory. V prípade pozorovateľa X sú to sektory 1,5 a 6. Steny z ostatných sektorov sa do ďalšej fázy riešenia viditeľnosti nedostanú.



Obr. 92 Rozdelenie scény na sektory

Hranice sektorov a postup kreslenia musia spĺňať niektoré podmienky, aby celý postup fungoval správne:

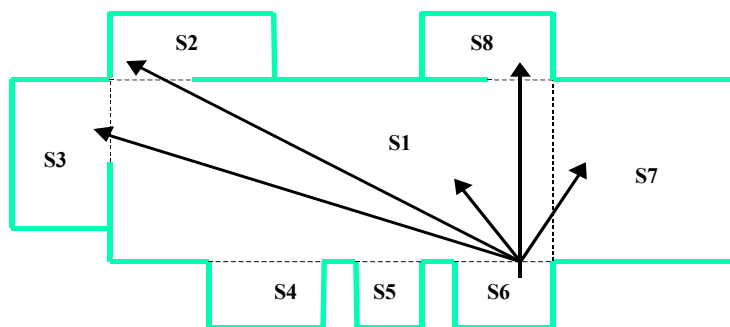
- ❑ Každá hranica je hranicou práve dvoch sektorov
- ❑ Platí, že zo sektoru A do sektoru B môže pozorovateľ vidieť iba cez hranicu sektorov A a B, t.j. každý sektor je uzavretý, a to hranicami s ostatnými sektormi a nepriehľadnými stenami.
- ❑ Sektory sú disjunktné
- ❑ Kreslenie stien z každého sektoru je orezané na výrez, ktorým pozorovateľ do neho vidí cez danú hranicu. To znamená, že na sektor sa pozorovateľ díva cez okno, vytvorené hranicou sektoru. Týmto sa zabráni problému v prípade, že pozorovateľ vidí naraz viac hraníc do toho istého sektoru.
- ❑ Každý sektor sa kreslí len raz (zabránenie zacyklenia)

Sektory sú lepším riešením ako napríklad BSP, pretože nevyžadujú delenie polygónov, ale sami osebe nestačia na úplné riešenie viditeľnosti (preto sú uvádzané ako urýchľujúca metóda) a je potrebná kombinácia s inými technikami. Možné riešenie by bolo vytvoriť pre každý sektor osobitný BSP strom.

Nevýhodou sektorov je ich takmer nemožná "algoritmizácia", čiže napísať program, ktorý by automaticky rozdelil scénu na vhodné sektory s vhodným rozmerom na vhodných miestach. Najlepšie je, ak už v procese editácie scény dotýčny návrhár scény rozdelí, resp. určí hranice sektorov, pretože na tom stojí a padá celá myšlienka a nakoniec aj výkon algoritmu.

8.9.4 Potenciál viditeľnosti

PVS je skratka z anglického Potentially Visible Set. Je to obdoba techniky sektorov a jej princíp bude vysvetlený na príklade použitia v kombinácii s BSP. V BSP strome tvorí každý list konvexný podpriestor, listy sú navzájom disjunktné a každý uzol (nie list) je zjednotenie podpriestorov v jeho dcérskych uzloch.



Obr. 93 Príklad PVS pre sektor S6 (z neho vidieť sektory S1, S2, S3, S7 a S8).

PVS sa počíta vždy pre určitú časť priestoru (v tomto prípade pre podpriestory v listoch stromu) a udáva množinu potencionálne viditeľných objektov z daného miesta. Rozšírenie BSP o PVS teda znamená mať pre každý list v strome zoznam všetkých listov, ktoré z daného listu môžu byť viditeľné. Už na prvý pohľad je to náročné na pamäť keď uvažíme, že už v malej scéne je v BSP strome okolo 100 listov. Z pohľadu programu je postup vizualizácie scény zo štruktúry BSP+PVS nasledovný:

1. Zisti, v ktorom liste sa nachádza pozorovateľ
2. Zober PVS pre daný list a spracuj polygóny vo všetkých listoch v PVS (ak sú splnené ďalšie požiadavky, ako prítomnosť v ihlane pohľadu a pod.)

Metóda PVS je veľmi náročná na predspracovanie, pretože vyžaduje určiť možnú viditeľnosť každého listu z každého listu v BSP strome, čo je $O(n^2)$, kde n je počet listov stromu. Samotné určenie nožnej viditeľnosti je tiež dosť náročný proces. Malé nie sú ani pamäťové nároky, pri 500 listoch v strome je potrebné uložiť $\frac{500^2}{2}$, čiže 125.000 pravdivostných hodnôt pre celý strom, prípadne iným spôsobom (napríklad zoznamom viditeľných listov) uložiť do každého listu jeho PVS.

Poznámky:



1. Charakterizujte problém riešenia viditeľnosti v rámci počítačovej grafiky
2. Popíšte riešenie viditeľnosti grafov funkcií
3. Uveďte postup pri získaní horizontu v rámci algoritmu plávajúceho horizontu
4. Charakterizujte a popíšte maliarov algoritmus riešenia viditeľnosti
5. Charakterizujte a popíšte Warnockov algoritmus riešenia viditeľnosti
6. Charakterizujte a popíšte Freeman-Lotrelov algoritmus riešenia viditeľnosti
7. Charakterizujte a popíšte algoritmus pamäte hĺbky (Z-buffer)
8. Charakterizujte a popíšte algoritmus riadkového rozkladu pri riešení viditeľnosti
9. Charakterizujte metódu BSP stromov pri riešení viditeľnosti v rámci počítačovej grafiky
10. Popíšte tvorbu a prechod BSP stromom pri metóde BSP stromov v rámci riešenia viditeľnosti v počítačovej grafike
11. Popíšte vlastnosti a použitie BSP stromov v rámci riešenia viditeľnosti v počítačovej grafike
12. Charakterizujte a popíšte S-buffer algoritmus
13. Vymenujte a v krátkosti popíšte urýchľovacie techniky pre riešenie viditeľnosti v počítačovej grafike
14. Popíšte spôsob orezávania na zorný ihlan pri riešení viditeľnosti v rámci počítačovej grafiky
15. Porovnajte technológiu sektorovania a potenciálu viditeľnosti pri urýchľovaní riešenia viditeľnosti v rámci počítačovej grafiky



V tejto kapitole sme sa naučili:

- ☐ Problém riešenia viditeľnosti v počítačovej grafike spočíva v odstránení resp. odlišení tých častí trojrozmerných objektov, ktoré pri danom premietaní do 2D nie sú z miesta pozorovateľa viditeľné. Tým sú tieto časti akože zakryté a dostávame jednoznačné priemety telies.
- ☐ Rozdelenie algoritmov riešenia viditeľnosti je možné podľa nasledujúcich hľadísk: v akom tvare poskytujú výsledky, podľa priestoru v ktorom je viditeľnosť riešená, podľa reprezentácie riešených objektov, podľa zahrnutia osvetlenia do riešenia, podľa vplyvu možnej chyby pri vykonávaní algoritmu a podľa času potrebného na vyriešenie viditeľnosti.
- ☐ Podľa toho v akom tvare sú poskytované výsledky rozdeľujeme algoritmy riešenia viditeľnosti na objektové (vektorové) a rastrové (bodové). Pri objektových je výsledkom množina kriviek resp. plôch predstavujúcich viditeľné elementy. Pri rastrových je výsledkom obraz predstavujúci jednotlivé body (pixely) s farbou príslušných viditeľných plôch.
- ☐ Bod B je viditeľný, ak na priamke medzi bodom B a stredom premietania (poloha pozorovateľa) sa nenachádza žiadny iný bod objektu riešenej 3D scény.

- ☐ Pri riešení viditeľnosti grafov funkcií je možné riešenie v priestore alebo v priemetni. V prípade riešenia viditeľnosti v priemetni sa používa algoritmus plávajúceho horizontu.
- ☐ Základným objektovo orientovaným algoritmom je maliarov algoritmus riešenia viditeľnosti. Jeho podstata spočíva v zoradení objektov (tzv. Z-sort) od najvzdialenejších po najbližšie a v následnom kreslení v tomto poradí na priemtnu, čím vlastne bližšie objekty kreslené neskoršie, prekreslia vzdialenejšie objekty a tým sa vyrieši viditeľnosť. Maliarov algoritmus nerieši viditeľnosť vždy stopercentne.
- ☐ Algoritmom, ktorý rieši viditeľnosť až v priemetni formou rozdeľovania priemetne na kvadranty, je Warnockov algoritmus. Rieši viditeľnosť ľubovoľných, i nekonvexných telies s rovinnými stenami. V každom kvadrante sa posudzuje možnosť rozhodnutia o viditeľnosti a v prípade, že to nie je možné, dochádza k rekurzívnemu deleniu kvadrantu na menšie kvadranty. Rekurzia končí ako v kvadrante nie je nič, je tam len jedna stena alebo pri delení sme dospeli k tomu, že rozmer vytváraného kvadrantu je rovný jednému bodu zobrazovacieho rastra.
- ☐ Medzi klasické objektovo orientované algoritmy patrí aj Freeman-Loutrellov algoritmus. Podstata algoritmu spočíva v rozdelení plôch na neviditeľné a potencionálne viditeľné na základe porovnania vektora normály vyhodnocovanej steny a vektora pohľadu pozorovateľa.
- ☐ K súčasnosti najviac používaným algoritmom patrí Z-buffer (algoritmus pamäte hĺbky) algoritmus. Patrí medzi rastrové typy algoritmov. Vyhodnocuje sa farba každého bodu priemetne tak, že sa ráta priesečník priamky definovanej okom pozorovateľa a bodom priemetne s objektami v scéne. Potom sa vypočítajú vzdialenosti (hĺbka) jednotlivých priesečníkov a bod priemetne sa vyfarbí farbou najbližšieho priesečníka. Tento algoritmus patrí dnes k najviac implementovaným algoritmom v grafických akceleračtoroch.
- ☐ Algoritmom, ktorý vychádza z algoritmu pamäte hĺbky je algoritmus riadkového rozkladu. Rozdiel medzi hĺbkovým algoritmom je v tom, že tu sa v pamäti neudržiava informácia o viditeľnosti celej scény a tým pádom informácia o všetkých pixeloch, ale spracúvava sa len jeden riadok pixelov. Keď sa určí viditeľnosť pre daný riadok, vykreslí sa a začne sa spracovávať ďalší riadok.
- ☐ BSP (Binary Space Partitioning) je metóda hierarchického rozkladu priestoru na konvexné podpriestory a je hojne využívaná pri vizualizácii v počítačovej grafike najmä z dôvodu rýchleho triedenia a vyradovania odvrátených plôch. Metóda je podobná maliarovmu algoritmu, ale v prípade nejednoznačnosti rozdelí problematický polygón. Metóda je stopercentná a aj napriek zložitejšiemu predprocesingu sa používa v systémoch riešenia viditeľnosti v reálnom čase. Jedná sa o objektovo-orientovaný algoritmus.
- ☐ S-buffer je jednou z techník, ktorá zabezpečuje neprekresľovanie už vykreslených polygónov v rámci priemetne. Najčastejšie sa pre každý riadok priemetne vytvorí spojkový zoznam už nakreslených intervalov.
- ☐ Medzi najčastejšie používané urýchľovacie technológie pri riešení viditeľnosti patria: orezanie na ihlan pohľadu, použitie ohraničujúcich útvarov, sektorovanie a použitie potenciálu viditeľnosti.



9 Realistické zobrazovanie

Cieľ: Táto kapitola má vzhľadom na svoj význam v súčasnej počítačovej grafike naozaj len základný informačný charakter. Študent po absolvovaní tejto kapitoly by mal vedieť charakterizovať problém tieňovania, osvetľovania a fotorealistického zobrazenia trojrozmerných scén prostriedkami počítačovej grafiky. Študent bude schopný vymenovať a popísať základné metódy tieňovania (konštantné, Gourardovo a Phongovo tieňovanie) používané v dnešnej počítačovej grafike. Takisto získa základné vedomosti o základnom fotorealistickom postupe, o metóde sledovania lúča – raytracing.

9.1 Tieňovanie

Pri práci v trojrozmernom priestore a s trojrozmernými objektami je pre reálnosť vnemu vhodné odlišiť aj určité zaoblenia resp. krivosti plôch. Tieňovanie samotné nerieši viditeľnosť a je to v podstate vykresľovanie farebných objektov rôznymi odtieňmi farieb. Pri týchto technikách sa častokrát používajú aj rozptyľovacie techniky ako *polotónovanie* a *rozptyľovanie*. Pomocou týchto techník dokážeme pomocou niekoľkých farieb vytvoriť dojem bohatej farebnej škály. Ako už bolo spomenuté, tieňovanie nerieši viditeľnosť, ale napomáha dostať očakávaný priestorový vnem. Pre realistický vnem sa zavádzajú tzv. modely svetelného odrazu, ktoré rozkladajú svetlo v každom bode viditeľného povrchu telies do troch oblastí. Výpočet odrazu svetla je veľmi časovo náročný. Z tohto dôvodu boli hľadané postupy ako určovať odraz svetla rýchlejšie, napr. pre celé plôšky (segmenty) riešených objektov.

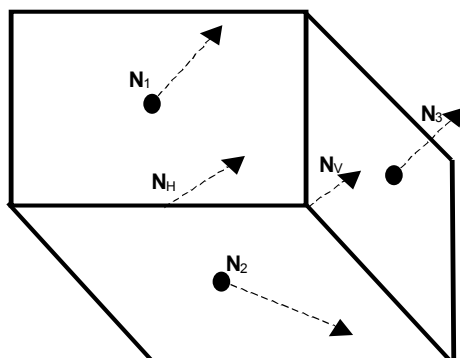
9.1.1 Konštantné tieňovanie

Konštantné tieňovanie (angl. *flat-shading*) patrí medzi základné metódy tieňovania. Princíp je veľmi jednoduchý. Určí sa hodnota odrazeného svetla v jednom ľubovoľnom bode skúmanej plochy daného objektu. Na základe tejto hodnoty sa vyfarbí celá plocha (platí aj pre úrovně šedi). Spôsob určovania je jednoduchý. Môže používať napr. princíp *Freeman-Loutrellovho algoritmu*. Rozdiel je v tom, že do zroja svetla sa "akože" umiestni pozorovateľ. Potom plochy zistené ako viditeľné sa vyfarbia svetlejšou farbou a plochy potencionálne viditeľné tmavšou farbou. Pre zvýšenie realistickosti je tu navyše potrebná aj veľkosť uhla medzi zdrojom svetla a plochou. Podľa tohto uhla sa potom nastaví intenzita farby.

9.1.2 Gouraudovo tieňovanie

Tejto metóde tieňovania hovoríme aj *kontinuálne tieňovanie* alebo tiež *tieňovanie interpoláciou farby*. Oproti konštantnému tieňovaniu sa tu uvažuje o tom, že zrakový vnem osvetlených objektov nie je celkom skokový na hranách objektov. V okolí hrán je odraz svetlejší (resp. tmavší) v závislosti na okolí. Samotný výpočet osvetlenia stien je vykonávaný z výpočtu osvetlenia hrán a vrcholov objektu. Riešenie pozostáva z niekoľkých krokov.

V prvom kroku sa určia "normály" riešených hrán pomocou výslednice normál stien, ktoré hranu vytvárajú. V druhom kroku sa určí to isté, ale pre riešené vrcholy. Pomocou určených normál sa určí osvetlenie hrán. Príklad ukazuje nasledujúci obrázok.



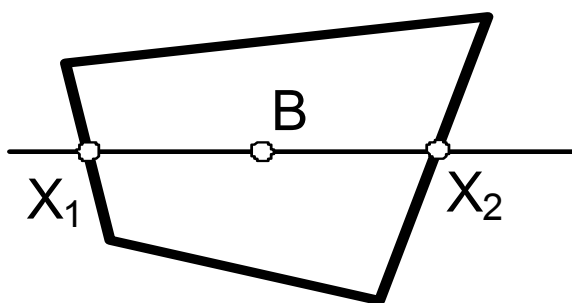
Obr. 94 Určenie normál pri Gouraudovom tieňovaní

$$\begin{aligned} \text{Potom } \mathbf{N}_V &= \mathbf{N}_1 + \mathbf{N}_2 + \mathbf{N}_3 \\ \mathbf{N}_H &= \mathbf{N}_1 + \mathbf{N}_2 \end{aligned} \quad (124)$$

Tretím krokom je určenie osvetlenia jednotlivých bodov steny. Toto je vypočítané už v priemete. Výpočet je interpoláciou medzi príslušnými známymi hodnotami osvetlenia hrán pozdĺž jednej zo súradníc. Príklad ukazuje Obr. 95.

Výpočet vykonáme napr. pozdĺž súradnice x . Body X_1 a X_2 sú body na hranách a ich intenzita osvetlenia I_{X1} a I_{X2} sú známe. Potrebujeme určiť intenzitu osvetlenia I_B v bode $B[x_B, y_B]$ riešenej plochy. I_B určíme interpolačne pomocou nasledujúceho vzťahu:

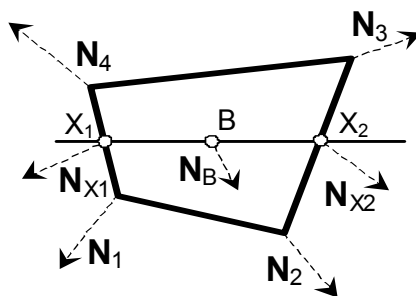
$$I_B = I_{X1} \cdot \frac{X_2 - X_B}{X_1 - X_2} + I_{X2} \cdot \frac{X_B - X_1}{X_2 - X_1} \quad (125)$$



Obr. 95 Určenie intenzity pri Gouraudovom tieňovaní

9.1.3 Phongovo tieňovanie

Tento algoritmus patrí medzi najviac používané algoritmy tieňovania. Je podobný Gouraudovmu algoritmu, ale určenie osvetlenia bodu plochy sa nevypočítava medzi dvomi bodmi interpolačne. Obdobne ako u "Gourauda" sa určia normály riešených hrán a vrcholov. Po určení normál hrán je určovaná interpolačne normála jednotlivých bodov plochy. Táto normála je určená ako výslednica normál hrán. Príklad ukazuje nasledujúci obrázok.



Obr. 96 Určenie normál v Phongovom tieňovaní

Keď predpokladáme určenie normály v strede, je:

$$\mathbf{N}_{X1} = \frac{\mathbf{N}_1 + \mathbf{N}_4}{2} \quad \mathbf{N}_{X2} = \frac{\mathbf{N}_2 + \mathbf{N}_3}{2} \quad (126)$$

potom výsledná normála:

$$\mathbf{N}_B = \frac{\mathbf{N}_{X1} + \mathbf{N}_{X2}}{2} \quad (127)$$

Tento výpočet je omnoho zložitejší a časovo náročnejší, avšak umožní zobrazenie aj možného zakrivenia plochy (napr. pri guli).

9.2 Fotorealistické zobrazovanie

Okrem skôr popísaných základných metód tieňovania 3D-scén sa v súčasnosti presadzujú najmä metódy umožňujúce fotorealistické zobrazenie 3D scén. V princípe existujú dve možné metódy ako dosiahnuť príslušný výsledok. Prvá metóda je lepšie algoritmizovateľná ale nevychádza celkom z fyzikálnej podstaty šírenia svetla. Druhá metóda je síce fyzikálne dobre podporená, avšak sa ťažšie algoritmizuje. Poznáme teda tieto základné metódy:

- **metóda sledovania lúča** (*raytracing*)
- **vyžarovacia metóda**. (*radiosity*)

Prvá metóda vychádza zo sledovania svetelného lúča od pozorovateľa (v našom prípade obrazovky) ku zdrojom svetla a potom spätné určenie farby. Druhá metóda je založená presne na opačnom princípe a to sledovanie svetla od zdroja k pozorovateľovi. V tomto princípe sa na svetlo "pozerá" ako na šíriacu sa energiu a viac sa prihliada na fyzikálnu podstatu týchto javov. Obe metódy sú pomerne náročné či už na výpočet alebo čas.

9.2.1 Porovnanie metód fotorealistického zobrazovania

Obe metódy majú svoje nesporné klady, z ktorých niektoré boli uvedené v predchádzajúcom. Vyžarovacia metóda síce vychádza z fyzikálnej podstaty šírenia energie, avšak vplyvom globálneho poňatia šírenia svetla nedokáže zobrazovať zrkadlové javy. Tento jav je totižto spojený s konkrétnymi lúčmi, ktoré prinášajú informáciu o farbe zrkadených predmetov. Na tomto základe nie je možné ani zobrazovať dobre ani priesvitné predmety. Táto metóda teda umožňuje pomerne rýchle hľadanie svetelných charakteristík, výsledkom čoho je vysokokvalitné zobrazenie s vernými a plynulými prechodmi. Toto všetko je však ideálne ak sa

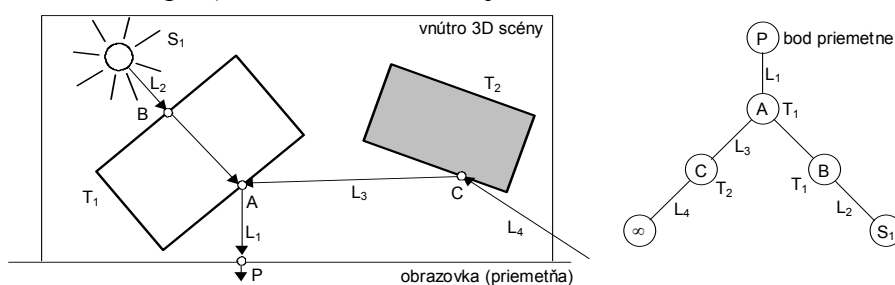
geometria scény nemení. Naproti tomu síce raytracing neposkytuje až takú kvalitu ako metóda vyžarovacia, avšak bezpečne zvládne zrkadlové a prechodové javy. Je možné ho lepšie algoritmizovať a je pomerne vhodná aj pre scény s meniacou sa geometriou.

Nové smery spôsobov verného zobrazovania priestorových scén vedú najmä ku kombinácii oboch metód. Samozrejme sú využívané klady oboch metód a zápory sú potláčané. Vždy je však určitá metóda dominantná. Napr. do metódy sledovania lúča sa difúzne zložky svetla vypočítavajú práve vyžarovacou metódou. Naopak sa zapracúvavajú výpočty konfiguračných faktorov vo vyžarovacej metóde práve metódou raytracing. Toto umožňuje nájdenie aj malých tieňov na veľkých plochách. Ďalej raytracing generuje pomerne ostré tieň. Pre získanie realistickejšieho zobrazenia sa zavádza tzv. **distributívny raytracing**. Princíp je v tom, že pri odraze alebo lome lúča sa negeneruje len jeden ďalší lúč, ale pomocou určitej distribučnej funkcie sa vygeneruje lučov viac. Spätné skladanie pre výpočet intenzity a farby sa deje zo všetkých takto vygenerovaných lúčov. Netreba však pripomínať, že obe metódy sú výpočtovo náročné najmä na čas. Keďže metódy majú prirodzený paralelizmus, vhodným zrýchlením je ich riešenie na paralelných výpočtových systémoch tak, že výpočty sa distribuuju na jednotlivé výpočtové uzly (**distribuuovaný raytracing**).

Keďže z hľadiska algoritmizácie je výhodnejšia prvá spomínaná metóda, popíšeme si ju v nasledujúcej kapitole.

9.2.2 Metóda sledovania lúča (raytracing)

Táto metóda umožňuje zobrazenie aj zrkadlových javov. Ako už bolo uvedené pri *raytracing-u* sa hľadajú svetelné lúče, ktoré po prechode scénou sa zobrazia na sietnici nášho oka. Keďže "oknom", ktorým sa pozeráme na scénu, je obrazovka, je farba lúčov zobrazovaná pomocou pixelov obrazovky. Určovanie farby pixelu (teda vlastne lúča, ktorý sa vyšle z daného pixelu) sa deje vlastne skladaním farieb lúčov, z ktorých sa výsledný lúč poskladá pri svojom prechode scénou (zdroje svetla, odrazivé telesá a pod.). Predstavme si nasledujúcu modelovú scénu:



Obr. 97 Modelová scéna a jej strom vzniknutý pri raytracing-u

Vyšleme lúč L_1 cez pixel P obrazovky do scény. Tento lúč L_1 dopadne na teleso T_1 umiestnené v scéne. Nech toto teleso čiastočne prepúšťa svetlo. Podľa známej fyzikálnej podstaty šírenia svetla, sa vytvoria dva lúče: L_2 je lúč lomeného svetla (prechádzajúci telesom) a L_3 je lúč odrazeného svetla. Lúč L_2 fakticky ide do(z) zdroja svetla S_1 . Lúč L_3 ďalej naráža na teleso T_2 , ktoré je nepriehľadné a tak vzniká len lúč L_4 , ktorý po odrazení od T_2 opúšťa scénu. Takýmto popisom dostaneme v podstate strom, pomocou ktorého je možné vysledovať lúč samozrejme spätne a určiť farbu pixelu P (pozri Obr. 97 vpravo).

Už z uvedeného popisu je jasné, že sa jedná o typicky rekurzívny algoritmus (po dopade vzniknú ďalšie lúče a na ich sledovanie sa vyvoláva znova tá istá procedúra). Do tohto je možné zahrnúť aj vzdialenosť medzi telesami, resp. pozorovateľom a telesami, čím je možné zapracovať aj určitý útlm intenzity lúča. Samotný algoritmus má dva kroky. V prvom sa prehľadá scéna pre príslušný lúč a vytvorí sa strom priesečníkov generovaných lúčov s telesami scény. Druhým krokom je spätný chod na vyhodnotenie svetelného modelu v uzloch stromu a zloženie výslednej farby. Tým, že to má rekurzívny charakter je možné algoritmus ukončiť, keď ide lúč mimo scény, alebo ide na horizont alebo "vošiel" do svetelného zdroja. Ďalším spôsobom je aj možnosť omedziť hĺbku rekurzívneho algoritmu, ale to vedie k deformáciám v zobrazení. Pre výpočet osvetľovacieho modelu použijeme nasledujúci vzťah:

$$I_V = I_Z + I_D + I_O + I_R + I_L \quad (128)$$

kde:

- I_V je výsledná intenzita
- I_Z je zrkadlová zložka (týka sa zdroja svetla)
- I_D je difúzna zložka
- I_O je ozložka okolitého osvetlenia
- I_R je zložka z odrazeného lúča (týka sa telesa)
- I_L je zložka z lomeného lúča

Pre zložky I_R a I_L je možné ešte zaviesť koeficient zrkadlového odrazu k_z a koeficient lomu k_L (charakterizujúci priehľadnosť telesa). Potom fakticky zložka I_R je daná nasledovne:

$$I_R = k_z \cdot I_r \quad (129)$$

Pre zložku lomeného lúča platí analogický vzťah. Nie je potrebné snáď hovoriť a spomenuté to už bolo, že metóda sledovania lúča je časovo náročná. Preto sa hľadali rôzne metódy ako ju urýchliť. V princípe je možné použiť podobné urýchľovacie techniky ako pri riešení viditeľnosti ved' v princípe sa častokrát raytracing aj radiosity označujú aj ako globálne technológie riešenia viditeľnosti. Možnými spôsobmi potom môže byť použitie:

- *ohraničujúcich telies*, kde telesá sa uzatvárajú do abstraktných telies, vzhľadom na ktoré je možné skôr zistiť, či lúč teleso pretína alebo nie. Najčastejšie sa takto používa guľa.
- *triedenia telies* podľa ohraničujúcich súradníc, kde pri zotriedení telies podľa niektorej zo súradníc je možné vylúčiť určitú skupinu telies, ktorými lúč neprechádza a nie je nutné testovať všetky telesá.
- *oktantového stromu*, kde sa scéna delí postupne na oktanty a tvorí sa strom, ktorý má v príslušných uzloch či listoch len tie telesá, ktoré do neho zasahujú. Pri hľadaní priesečníka sa prechádza scénou na základe tohto stromu. Prázdny uzol či listy (oktanty). Použitie tohto spôsobu pri väčšom počte telies spôsobí zrýchlenie metódy o jeden až dva rády.
- *BSP* (Binary Space partitioning) *stromov*, ktorých použitie je obdobné ako pri oktantových stromoch, ale deliaca rovina je vždy len jedna a tým sa delí priestor scény vždy len na dva polpriestory. Takto vytvorená štruktúra je pamäťovo nenáročná a pomerne rýchlo sa s ňou pracuje.

1. Charakterizujte a popíšte konštantné (flat) tieňovanie v rámci počítačovej grafiky
2. Charakterizujte a popíšte tieňovanie interpoláciou farby (Gourard) v rámci počítačovej grafiky
3. Charakterizujte a popíšte tieňovanie interpoláciou normály (Phong) v rámci počítačovej grafiky
4. Charakterizujte problém fotorealistického zobrazovania v rámci počítačovej grafiky
5. Popíšte metódu spätného sledovania lúča (raytracing), tvorbu a prechod definičným binárnym stromom raytracingu a základný spôsob výpočtu intenzity farby bodu pri tomto postupe.



V tejto kapitole sme sa naučili:



- ☐ Tieňovanie v počítačovej grafike sa používa pri práci v trojrozmernom priestore a s trojrozmernými objektami pre zvýšenie reálnosti vnemu po vyriešení viditeľnosti resp. na odlišenie zaoblenia alebo krivosti plôch. Používa vykresľovanie objektov pomocou rôznych odtieňov farieb.
- ☐ Medzi základné typy tieňovaní používaných v počítačovej grafike radíme: konštantné (flat) tieňovanie, tieňovanie interpoláciou farby (Gourardovo tieňovanie) a tieňovanie interpoláciou normály (Phongovo tieňovanie).
- ☐ Rozoznávame dva druhy fotorealistického zobrazovania scén: metóda spätného sledovania lúča (raytracing) a vyžarovacia metóda (radiosity).
- ☐ Pri metóde raytracing sa hľadajú svetelné lúče, ktoré po prechode scénou sa zobrazia na sietnici nášho oka. Metóda je podobná algoritmu pamäte hĺbky. Určovanie farby pixelu sa deje však skladaním farieb lúčov, z ktorých sa výsledný lúč poskladá po prechode scénou. Tento algoritmus je rekurzívny a náročný na čas.
- ☐ Urýchlenie metódy raytracing je možné použitím ohraničujúcich telies, triedením telies, delením priestoru na oktanty alebo pomocou technológie BSP stromov.



10 Farby v počítačovej grafike

Cieľ: Po absolvovaní tejto kapitoly by mal študent vedieť definovať problematiku používania farieb v počítačovej grafike na úrovni atribútov svetla a farebných modelov. Takisto bude schopný vymenovať a popísať základné farebné modely (RGB, CMY, HSB, HLS), ktoré sa používajú v rámci počítačovej grafiky. Ďalej študent získa základné vedomosti pre implementáciu prevodov medzi jednotlivými farebnými modelmi. Ako doplnkové pojmy bude študent po absolvovaní tejto kapitoly schopný definovať pojmy ako gama korekcia, alfa miešanie, rozptyľovanie a poltónovanie

Keďže jedným a takmer dominantným atribútom, ktorý sa používa pri spracovaní obrazov či pri práci s grafickými formátmi je farba obrazu resp. farba základného spracovávaného objektu (u rastrových je to pixel), pohovorme teda najprv o farbách. Aby sme však mohli hovoriť o farbách, musíme najprv povedať niečo o svetle. Z fyzikálneho hľadiska je ako svetlo chápané elektromagnetické vlnenie v oblasti 10^8 Hz. Z hľadiska farieb zodpovedá každá farba určitej frekvencii. Rozsah farieb je od červenej (4.3×10^8 Hz, mimo viditeľného spektra pokračuje do infračervenej oblasti) po fialovú (7.5×10^8 Hz, mimo viditeľného spektra pokračuje do ultrafialovej oblasti). V rámci viditeľného spektra je človek schopný rozlíšiť viac ako 4×10^5 rôznych farieb ich odtieňov. Podľa frekvencie, ktorú vysiela svetelný zdroj je možné svetlo rozdeliť na:

- *achromatické svetlo*. Tomuto svetlu sa hovorí tiež biele svetlo a obsahuje všetky farby (typický zdroj je slnko). Kombinácia frekvencií odrazených od telies vytvára v podstate farbu telies. Ak prevláda frekvencia z určitej oblasti spektra, hovoríme o dominantnej frekvencii.
- *monochromatické svetlo*. Je svetlo len jednej farby napr. červenej.

Svetlo je charakterizované niekoľkými svojimi atribútmi:

- *farba*, je základným atribútom svetla a závisí od už spomínanej frekvencie (resp. vlnovej dĺžky)
- *jas* odpovedá vlastne intenzite svetla. Jasnosť zdroja svetla je v priamej úmere s intenzitou
- *sýtosť* farby uvádza jej čistotu. Čím vyššia je sýtosť, tým užšie je spektrum frekvencií obsiahnutých vo svetle
- *svetlosť* je vlastne veľkosť achromatickej zložky vo svetle s určitou dominantnou frekvenciou.

Dôležitým faktorom je aj skladanie farieb. Je otázka, či existujú určité základné farby, pomocou skladania ktorých by sa vytvorili všetky ostatné. Existujú však tzv. komplementárne farby, ktorých kombináciou získame biele svetlo. Pre skladanie viacerých farieb bol vytvorený štandard vo forme chromatického diagramu. Avšak spomínaný chromatický diagram neurčuje z akých základných farieb sa ostatné budú skladať ani ich pomer. Toto určujú farebné modely, o ktorých bude pojednané ďalej.

10.1 Farebné modely

Ako už bolo povedané skôr, pri práci s farbami sú dôležité dve základné činnosti. Prvou je určenie základnej množiny farieb, z ktorou sa bude pracovať. Druhou činnosťou je určenie spôsobu ako sa budú kombinovať. Farbocit je pomerne značne subjektívna záležitosť a zmiešaním dvoch alebo viacerých farieb môže vzniknúť

rôzna predstava novej farby u rôznych ľudí. Poznáme dva základné spôsoby kombinácie (miešania) farieb:

- ❑ *aditívne miešanie* (každým pridaním určitej zložky vznikne svetlejšia farba. Pridaním všetkých vznikne biela, typický model RGB)
- ❑ *subtraktívne miešanie* (každým pridaním určitej zložky vznikne tmavšia farba. Pridaním všetkých vznikne čierna, typický model CMY)

Na základe tohto hovoríme aj o farebných modeloch. Tieto sú charakterizované:

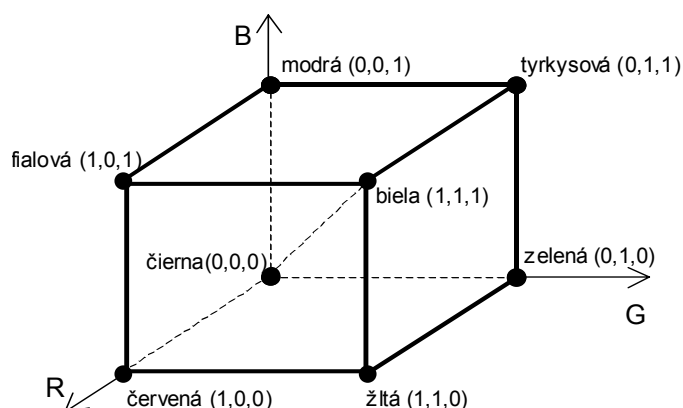
- ❑ množinou základných farieb,
- ❑ spôsobom ich miešania a
- ❑ pravidlami menenia farebných charakteristík.

Súčasnosti existuje niekoľko farebných modelov. Medzi základné patria: model RGB, model CMY(K), model HSB, model HLS a z hľadiska vnímania model UWB.

10.1.1 Model RGB

V tomto modeli sú farby vytvárané aditívnym spôsobom. Základné zložky sú: *R* - (Red) červená, *G* - (Green) zelená a *B* - (Blue) modrá. Pre tieto farby je príznačné práve to, že ľudské oko má najlepšiu citlivosť práve pre ich vlnové dĺžky (630nm, 530nm a 450nm). Intenzita základných farieb sa v tomto modeli pohybuje v intervale $<0,1>$. Pri technickej implementácii je tento rozsah prevádzaný do digitálnej formy. Najčastejšie je kódovaný na 8-bitov (t.j. 256 dielov). Pre ľudské oko by postačovalo aj delenie na 100 dielov. Pri praktickom nastavovaní sa preto častokrát používa percentuálne nastavenie jednotlivých zložiek.

Farebný model RGB svojim rozsahom sa najčastejšie reprezentuje ako jednotková kocka umiestnená v osiach *r,g,b*. Z toho aj vyplýva, že množina základných farieb obsahuje 8 farieb. Vrchol $[0,0,0]$ (t.j. stred súradnicového systému) odpovedá čiernej farbe (Black). Naproti tomu vrchol $[1,1,1]$ odpovedá bielej farbe (White). Farby ležiace na diagonále medzi týmito vrcholmi odpovedajú odtieňom šedej (Gray). Záverom dodajme snáď, že daný model sa najviac, oproti iným modelom, technicky orientovaný. Nasledujúci obrázok ukazuje jednotkovú kocku modelu RGB.



Obr. 98 Model RGB

Na základe toho je možné zostaviť tabuľku množiny základných farieb modelu RGB s príspevkami jednotlivých farebných zložiek:

zložka farba	R(ed) červená	G(reen) zelená	B(lue) modrá
čierna	0	0	0
modrá	0	0	1
zelená	0	1	0
tyrkysová	0	1	1
červená	1	0	0
fialová	1	0	1
žltá	1	1	0
biela	1	1	1

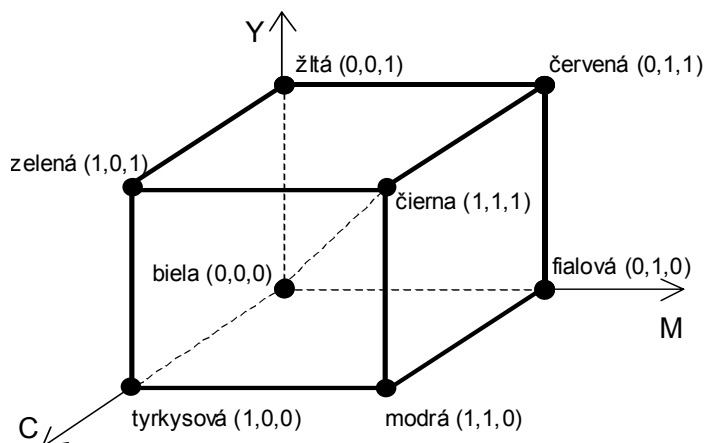
Tab. 3 Tabuľka základných kombinácií RGB

10.1.2 Model CMY

V tomto modeli sú farby vytvárané subtraktívnym spôsobom. Základné zložky sú: *C* - (Cyan) *tyrkysová*, *M* - (Magenta) *fialová* a *Y* - (Yellow) *žltá*. Pre tieto farby je príznačné práve to, že ľudská skúsenosť z miešaním farieb, najmä u maliarov, vychádza práve zo subtraktívneho miešania farieb. Preto je tento spôsob prirodzenejší. Tento model sa využíva aj v polygrafii pre reprodukciu farebných obrazov (aj fotografií). Výsledný farebný obraz dostaneme ako súťaž troch obrazov na báze jednotlivých zložiek. Tu sa ešte pridáva čierna zložka (black). Je to z toho dôvodu, že ak technologicky vznikne čierna zmiešaním všetkých troch zložiek, nedosahuje spravidla potrebnú kvalitu a preto sa čierne oblasti pretlačia zvlášť technologickou čiernou. Takýto model sa preto uvádza aj ako model **CMYK**. Moderné grafické systémy majú prostriedky na separáciu jednotlivých farebných zložiek obrazu na tvorbu spomínaných obrazov pre súťaž.

Farebný model CMY svojim rozsahom sa najčastejšie reprezentuje ako jednotková kocka umiestnená v osách *c,m,y*.

Z toho aj vyplýva, že množina základných farieb obsahuje opäť 8 farieb. Vrchol $[0,0,0]$ (t.j. stred súradnicového systému) odpovedá bielej farbe. Naproti tomu vrchol $[1,1,1]$ odpovedá čiernej farbe. Farby ležiace na diagonále medzi týmito vrcholmi, podobne ako u modelu RGB, odpovedajú odtieňom šedej s narastaním v opačnom smere. Nasledujúci obrázok ukazuje jednotkovú kocku modelu CMY.



Obr. 99 Model CMY

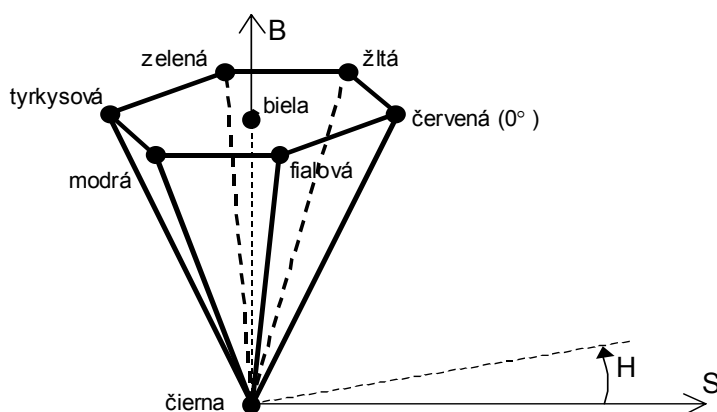
Podobne ako u modelu RGB je možné zostaviť tabuľku množiny základných farieb modelu CMY s príspevkami jednotlivých farebných zložiek:

zložka	C(yan)	M(agenta)	Y(ellow)
farba	tyrkysová	fialová	žltá
biela	0	0	0
žltá	0	0	1
fialová	0	1	0
červená	0	1	1
tyrkysová	1	0	0
zelená	1	0	1
modrá	1	1	0
čierna	1	1	1

Tab. 4 Tabuľka základných kombinácií CMY

10.1.3 Model HSB

Tento model oproti predchádzajúcim, skôr technickým, modelom je ešte bližší ľudskému chápaniu svetla, pretože zachytáva tie charakteristiky farby, ktoré sú bližšie k intuitívnemu popisu farieb človekom. Základné zložky sú: *H* - (Hue) *farebný tón*, *S* - (Saturation) *saturácia* alebo tiež *sýtosť* a *B* - (Brightness) *hodnota jas*. Farebný model HSB (niektoré zdroje ho udávajú ako **HSV**, kde *V* (Value) je hodnota jas) sa reprezentuje ako šesťboký ihlan, ktorého vrchol leží v počiatku súradnicovej sústavy. Súradnice *b* a *s* sa podobne ako u RGB (CMY) modelu menia od 0 do 1. Súradnica *h* je však uhlová z intervalu $\langle 0^\circ, 360^\circ \rangle$. Vrchol ihlanu v bode $[0,0,0]$ predstavuje čiernu farbu. Biela farba je v strede podstavy ihlanu. Jas klesá od podstavy k vrcholu. Sýtosť je daná vzdialenosťou od osi ihlana. V tomto aj spočíva určitý nedostatok tohto modelu. Pretože pri konštantnej hodnote *s* sa pri zmene farebného tónu (*h*) musíme pohybovať po šesťuholníkovej dráhe a nie po kruhovej, ktorá by bola prirodzená. Príslušné čisté farby (červená, žltá, zelená, tyrkysová, modrá a fialová) ležia na obvodě podstavy ihlana v príslušných vrchoch šesťuholníka. Z daného vyplýva aj poloha dominantných farieb a to na plášti ihlanu. Graficky znázornený model HSB prináša nasledujúci obrázok.



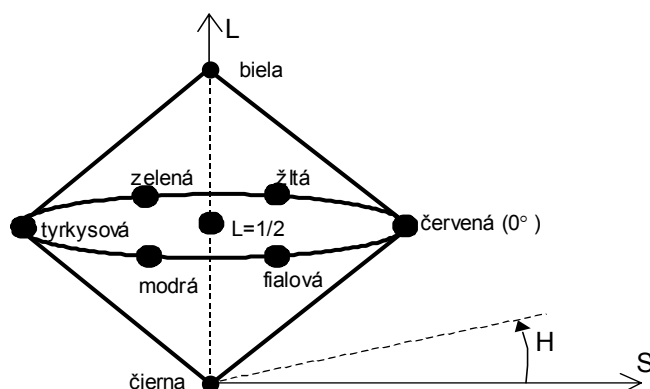
Obr. 100 Model HSB

10.1.4 Model HLS

Niektoré nedostatky predchádzajúceho modelu odstránil práve tento model. V tomto modeli je šesťboký ihlan nahradený dvojicou kužeľov. Základné zložky sú: *H* - (Hue) *farebný tón*, *L* - (Lightness) *svetlosť* a *S* - (Saturation) *saturácia* alebo tiež *sýtosť*. Farebný model HLS svojím rozsahom sa reprezentuje, ako bolo už

Poznámky:

spomenuté, dvojicou kuželov, ktoré majú spoločnú podstavu. Súradnice l a s sa podobne ako u RGB (CMY) modelu menia od 0 do 1. Súradnica h je opäť uhlová z intervalu $\langle 0^\circ, 360^\circ \rangle$. Vrchol jedného kužela v bode $[0,0,0]$ predstavuje čiernu farbu. Biela farba je naopak zase vo vrchole druhého kužela. Tento model asi najviac odpovedá skutočnosti, lebo najviac farieb je vnímaných práve pri strednej svetlosti (poloha spoločnej postavy kuželov, kde $L=0.5$) a vnímavosť klesá tak pri veľkom presvetlení ako aj stmavení. Príslušné základné farby (červená, žltá, zelená, tyrkysová, modrá a fialová) ležia opäť na obvodovej spoločnej podstavě kuželov, kde $s=l$ a $l=0.5$. Výhoda kruhovej podstavy spočíva práve v obiehaní okolo osi, kde už nie je nutný prechod po šesťuholníku ako u HSB, ale po ľahšej a prirodzenejšej kružnici. Obidva posledne definované modely umožňujú meniť jednotlivé farebné charakteristiky pri zachovaní ostatných typických vlastností farieb. Graficky znázornený model HLS prináša nasledujúci obrázok.



Obr. 101 Model HLS

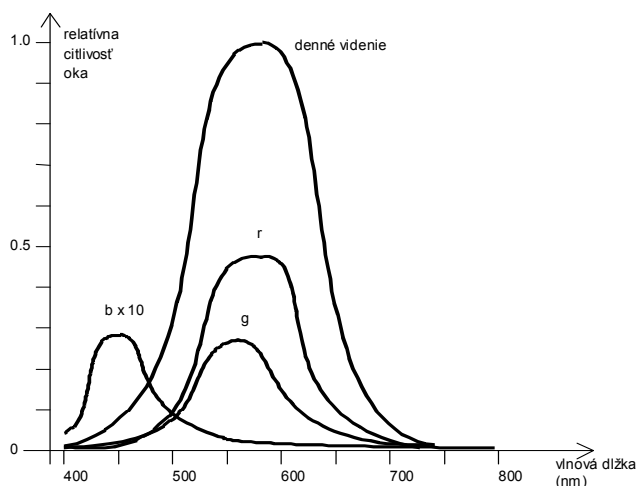
10.1.5 Model UWB



Tento model býva niekedy označovaný aj **YUV**. **UWB** je skratka od *Brightness* (Jas) a **UW** je označením pre dva farebné signály so pseudomenami **U** a **W**. Spomínané modely napr. **RGB** alebo **CMY** patria medzi klasické modely, ktoré sa pomerne dlho používajú. Existuje však mnoho iných modelov napr. **CIE-UVW**, **CIE-LAB** či **CIE-Luv** a pod. Tieto a aj spomínané modely definovala (počnúc rokom 1931) organizácia *Commission Internationale de l'Eclairage* (Medzinárodná organizácia pre osvetlenie angl. The International Commission on Illumination).

Okrem už teda definovaných typov existujú aj modely rešpektujúce spôsob vnímania farieb. Hlavným ľudským orgánom prijímania obrazových informácií je *oko*. Obraz je vytváraný na sietnici oka, kde je aj najviac nervových ukončení. Na sietnici sa nachádzajú dva druhy receptorov. Prvý druh (kužeľovité-cones, počet cca 6-7 mil.) sa nachádza v strede sietnice a je citlivý na farby. Senzitívnosť tohto druhu receptorov je možné rozdeliť ešte do dvoch skupín, ktoré rozlišujú farebný rozdiel spektra červená-zelená a modrá-žltá (RG-cones, BY-cones). Druhý druh (tyčinkovité-rods, počet cca 75-150 mil.) rovnomerne pokrýva celú sietnicu a umožňuje vnímať všeobecné obrazové informácie ako obrysy prípadne jas. Znamená to asi toľko, že kým pri bežnom dennom svetle je obraz prijímaný plnofarebne všetkými typmi receptorov, pri nasvetlení len monochromatickým svetlom sú aktivované len tyčinkovité-receptory.

Na základe týchto znalostí bol definovaný aj tento model. Všeobecne je teda možné povedať, že vlastne máme k dispozícii tri typy receptorov. Na základe toho je vnímaný jas a dva farebné signály pre odlišenie farieb v oblasti červeno-zelenej a oblasti modro-žltej. Nakoľko je takýto farebný priestor blízky televíznej norme PAL, častokrát sa potom označuje aj $YC_B C_R$. Kde je potom zjednodušene Y -jasová zložka, C_B -modrá zložka a C_R -červená zložka. Hodnota B (resp. Y) môže nadobúdať hodnoty z intervalu $\langle 0, 1 \rangle$, hodnoty farebných signálov U a W (C_B a C_R) sú spravidla z intervalu $\langle -0.5, 0.5 \rangle$. V prípade práce s odtienmi šedej sa pracuje len so zložkou B a zložky U, W sú ignorované.



Obr. 102 Relatívna citlivosť ľudského oka

10.2 Prevody farebných modelov



Častou funkciou grafických systémov býva aj možnosť zosvetlenia príp. stmavenia obrázku alebo jeho inverzné zobrazenie. Väčšinou sa využíva pri reprezentácii farieb podporovaný model RGB. Pre spomínané operácie je výhodnejšie však pracovať s modelmi HSB alebo HLS. Táto kapitola prinesie v krátkosti algoritmy prevodov medzi jednotlivými farebnými modelmi.

10.2.1 Prevody RGB a CMY

Prepočet z RGB do CMY je jednoduchý:

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad (130)$$

Opačný prepočet ako v predchádzajúcom prípade t.j. z CMY(K) do RGB:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} C \\ M \\ Y \end{bmatrix} \quad (131)$$

10.2.2 Prevod RGB do HSB

Pri prepočte sa vychádza z niektorých vlastností zmien súradníc RGB pri zmene polohy bodu v sústave HSB. Na plášti ihlanu je $S=1$, a aspoň jedna zo súradníc RGB je nulová. Pri zmene S ak H , B_{HSB} sú nemenné, zložky RGB sa menia lineárne. Pomer rozdielu menších zložiek a maximálnej zložky je pri zmene S konštantný. Veľkosť najväčšej súradnice RGB je zhodná s B_{HSB} a pri cyklickej zmene H sa mení len jedna zo súradníc RGB. Nakoľko model HSB sa častokrát označuje aj HSV, bude zložka B (doteraz označovaná B_{HSB}) tohto modelu ďalej označovaná V , aby sa odlišila od zložky B modelu RGB.

1. $maxim = \max(R, G, B)$
2. $minim = \min(R, G, B)$
3. $delta = max - min$
4. $V = max$
5. *ak* $max == 0$ $S = 0$ *inak* $S = (max - min) / max$
6. *ak* prevláda zložka R : $H = (G - B) / delta$
7. *ak* prevláda zložka G : $H = 2 + (B - R) / delta$
8. *ak* prevláda zložka B : $H = 4 + (R - G) / delta$
9. $H = H * 60$, *ak* H je záporné číslo, $H = H + 360$

10.2.3 Prevod HSB do RGB

V tomto prepočte sa najskôr zistí, v ktorej šestine kruhu sa bod nachádza. Potom sa na základe vzdialenosti od osi ihlanu vypočítajú hodnoty RGB, pričom prevládajúca farba nadobudne hodnotu V .

1. *ak* $S == 0$, $R = V$, $G = V$, $B = V$, koniec
2. *inak* $H = H / 60$
3. $i = \text{celá časť } H$, $f = \text{zvyšok po delení } H$
4. $p = V * (1 - S)$, $q = V * (1 - (S * f))$, $t = V * (1 - (S * (1 - f)))$
5. *ak* $i == 0$ $R = V$, $G = t$, $B = p$
6. *ak* $i == 1$ $R = q$, $G = V$, $B = p$
7. *ak* $i == 2$ $R = p$, $G = V$, $B = t$
8. *ak* $i == 3$ $R = p$, $G = q$, $B = V$
9. *ak* $i == 4$ $R = t$, $G = p$, $B = V$
10. *ak* $i == 5$ $R = V$, $G = p$, $B = q$

10.2.4 Prevod RGB do HLS

Na začiatku sa vypočíta hodnota L , ktorá určí, v ktorom kuželi sa bod nachádza. Ďalej nasledujú vypočty súradníc pomocného bodu (R_2, G_2, B_2) v rovine kruhovej podstavy z hodnôt L, R, G, B . Súradnica S sa určí na základe hodnoty L a minimálnej z hodnôt R, G, B . Podľa prevládajúcej farebnej zložky sa dá zistiť, v ktorej šestine kruhu sa bod nachádza a vypočíta sa hodnota H .

1. $minim = \min(R, G, B)$
2. $maxim = \max(R, G, B)$
3. $L = (minim + maxim) / 2$
4. *ak* $L == 1$, $H = 0$, $S = 0$, koniec //biela farba
5. *ak* $minim == maxim$, $S = 0$, $H = 0$, koniec //odtieň šedej

6. *inak* $S=(1-\min)/(1-L)-1$
7. *ak* $L > 0.5$
 $R2=1+(R-1)/(2-2L)$
 $G2=1+(G-1)/(2-2L)$
 $B2=1+(B-1)/(2-2L)$
8. *inak* $R2=R, G2=G, B2=B$
9. *ak prevláda zložka R*
ak $G2 > B2, H=G2/B2$
inak $H=6-B2/G2$
10. *ak prevláda zložka G*
ak $R2 > B2, H=2-R2/G2$
inak $H=2+B2/G2$
11. *ak prevláda zložka B*
ak $G2 > R2, H=4-G2/B2$
inak $H=4+R2/B2$
12. $H=H*60$
13. *ak* $H < 0, H=H+360$

10.2.5 Prevod HLS do RGB

Pri tomto prepočte sa určí na základe hodnoty L , v ktorom kuželi sa bod nachádza. Ďalej nasledujú výpočty súradníc pomocného bodu ($R2, G2, B2$) v rovine kruhovej podstavy z hodnôt L, S . Týmto bodom je vedená úsečka do vrcholu kužela a z podobnosti trojuholníkov takto vzniknutých sa vypočítajú súradnice RGB.

1. *ak* $S==0, R=G=B=L$, koniec
2. *inak* $H=H/60, a=\text{celá časť}, f=\text{zvyšok po delení}$
3. *ak* $a==0, R1=1, G1=f, B1=0$
4. *ak* $a==1, R1=1-f, G1=1, B1=0$
5. *ak* $a==2, R1=0, G1=1, B1=f$
6. *ak* $a==3, R1=0, G1=1-f, B1=1$
7. *ak* $a==4, R1=f, G1=0, B1=1$
8. *ak* $a==5, R1=1, G1=0, B1=1-f$
9. $R2=0.5+S*(R1-0.5)$
10. $G2=0.5+S*(G1-0.5)$
11. $B2=0.5+S*(B1-0.5)$
12. *ak* $L < 0.5$
 $R=R2*L/0.5$
 $G=G2*L/0.5$
 $B=B2*L/0.5$, koniec
13. *ak* $L > 0.5$
 $R=R2*(2-2*L)+(2*L-1)$
 $G=G2*(2-2*L)+(2*L-1)$
 $B=B2*(2-2*L)+(2*L-1)$, koniec
14. *ak* $L==0.5$
 $R=R2$
 $G=G2$
 $B=B2$

10.2.6 Prevod na úrovne šedej

Najjednoduchším spôsobom redukcie farieb obrazu je prepočet na odtiene šedej. Farebné odtiene môžu byť na odtiene šedej prevedené veľmi jednoducho podľa vzorca :

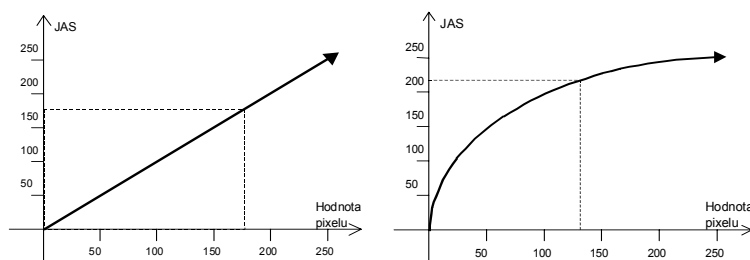
$$I = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \quad (132)$$

kde: I je výsledná intenzita (úroveň šedej)
 R, G, B sú základné farebné zložky pôvodnej farby

Prevod obrazu na úrovne šedej spočíva v postupnom prepočítaní všetkých bodov obrazu podľa vyššie uvedeného vzorca.

10.3 Gama-korekcia

Ako primárne výstupné zariadenie sa spravidla pre počítač používa monitor. V prípade CRT monitora tento zobrazuje farby najčastejšie pomocou fosforu vybudeného tokom elektrónov. Na základe toho začne fosfor žiarieť. V ideálnom prípade by bola závislosť hodnoty pixelu a jeho jas (intenzita) lineárna ako ukazuje nasledujúci obrázok vľavo. V praxi je však tento nedosiahnuteľný. Objavuje sa určitá nelinearita v chovaní sa fosforu pri jeho vybudení elektrónovým lúčom. Túto závislosť približne zobrazuje nasledujúci obrázok vpravo.

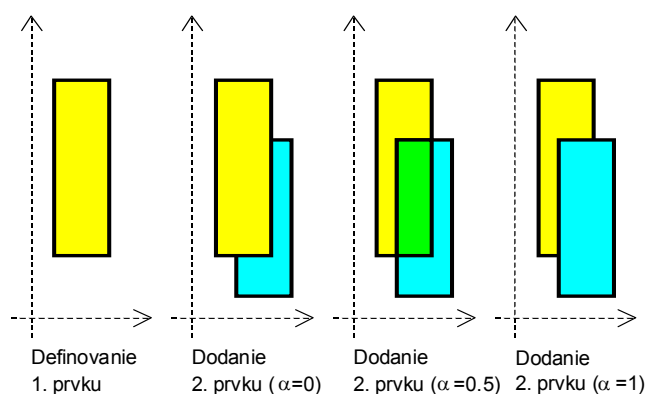


Obr. 103 Ideálna a reálna závislosť hodnoty pixelu a jeho jas.

Z tohto dôvodu je nutné brať do úvahy túto skutočnosť a podľa možnosti korigovať odchýlku hodnoty pixelu od skutočného jas pixelu na monitore. Tejto odchýlkovej hodnote sa hovorí gamma faktor a snahe o jej korigovanie potom gamma korekcia. Samozrejme, že všetky monitory nie sú rovnaké a preto napr. pre počítačové monitory je iná hodnota gamma korekcie ako by bola napr. pre klasické televízne obrazovky. Niektoré grafické systémy obsahujú funkciu na nastavenie gama korekcie.

10.4 Alfa-miešanie

Alfa-miešanie (α -blending) je funkcia pre zobrazovanie polopriehľadných plôch. Je možné ju podporovať ako hardvérovo, tak aj softvérovo. Koeficient priehľadnosti sa označuje ako *alfa-zložka*. Definuje sa buď ako číslo v pohyblivej rádovej čiarke z intervalu $\langle 0,1 \rangle$, alebo ako číslo (kód) v pevnej rádovej čiarke napr. 0-255, kde 0 odpovedá 0 a 255 odpovedá 0.99. Potom sa definuje úplná priehľadnosť pixelu pri hodnote *alfa-zložky* = 0. Ak má byť pixel nepriehľadný, potom je *alfa-zložka* = 1. Princíp alfa-miešania je ukázaný na nasledujúcom obrázku.



Obr. 104 Alfa-miešanie

10.5 Rozptyľovanie

Znížením počtu farieb obrazu často dochádza k významnému zhoršeniu vizuálnej kvality výsledného obrazu. Zobrazované farebné odtiene je síce možné vybrať z niekoľko tisíc farebných odtieňov, ale súčasne ich môže byť zvolených len obmedzený počet, úplným extrémom sú konverzie obrazov na čierno-biele (napríklad kvôli výstupu na čierno-bielu tlačiareň, niekedy dôležitá schopnosť grafického editora).

Aby bolo možné aj pri takýchto obmedzeniach reprodukovať obrazy obsahujúce mnoho farebných odtieňov, boli vyvinuté metódy, ktoré sa snažia preklenúť rozdiel medzi požadovanou kvalitou výsledných obrazov a obmedzenými možnosťami. Metódy, ktoré dokážu z niekoľkých farieb vytvoriť ilúziu bohatej farebnej palety sa nazývajú *rozptyľovanie* (*dithering*) a *poltónovanie* (*halftoning*).

Rozptyľovacie metódy využívajú tú vlastnosť ľudského oka, že z farieb niekoľkých blízkych bodov vytvára dojem jediného bodu, ktorého farba je daná aditívnym zlúčením farieb pôvodných bodov. Napríklad ak zariadenie nemá možnosť zobrazit' oranžovú, týmto postupom ju nahradíme napr. optickým zmiešaním červenej a žltej pomocou matice v tvare dominovej päťky.

Nasledujúce metódy sú kvôli väčšej názornosti popisované pre prevod obrazu so 16 úrovňami šedej na binárny obraz. Vo všetkých nasledujúcich metódach označuje intenzitu jasu (farbu) vstupného bodu I_{IN} a výstupného bodu I_{OUT} . Predpokladá sa že $I_{IN} \in \langle 0, 15 \rangle$ a $I_{OUT} \in \langle 0, 1 \rangle$.

10.5.1 Náhodné rozptyľovanie

Metóda náhodného rozptyľovania využíva pre generovanie rôznych odtieňov šedej generátor náhodných čísel. Ak bude vstupná intenzita I_{IN} rovná maximálnej intenzite (v našom prípade 15), nadobudne výstupná hodnota vždy najvyššiu hodnotu (tj. 1). V opačnom prípade rozhodne o výstupnej intenzite porovnanie vstupnej intenzity s náhodne vygenerovaným číslom menším ako maximálna vstupná intenzita.

Výsledný efekt metódy bude taký, že sa napríklad súvislá plocha s intenzitou 8 zobrazí ako plocha zložená z nepravidelne sa striedajúcich bodov s intenzitou 0 a 1. Pomer intenzít bude približne 1/2, čo zodpovedá pomeru 8/16.

Metóda náhodného rozptyľovania je jednou z najrýchlejších metód. Touto metódou zostanú pôvodné jasové pomery v obraze zachované.

10.5.2 Maticové rozptyľovanie

Metóda maticového rozptyľovania používa pre generovanie rôznych odtieňov šedej pravidelné vzorky zložené z bodov s intenzitou 0 a 1. Ak sú vzorky dobre navrhnuté, vyzerá výsledný obraz ako jemne vyšrafovaný. Táto metóda bola pôvodne navrhnutá pre takú modifikáciu obrazov, pri ktorej je jeden bod (prvok) vstupného obrazu nahradený maticou bodov vo výslednom obraze. Pri aplikovaní pôvodného algoritmu teda dochádza k zväčšeniu obrazu. Ak na výstupe pracujeme len s dvomi hodnotami prvkov matice potom sa jedná o *halftoning* (poltónovanie). Ak sa na výstupe pracuje s viacerými hodnotami prvkov matice potom sa jedná o *dithering*. Poltónovaniu sa zvlášť budeme venovať v samostatnej kapitole.

Na nasledujúcom obrázku sú zobrazené príklady vzoriek použitých v prípade že intenzity vstupného obrazu sú z intervalu $\langle 0,4 \rangle$. Pre $I_{IN}=0$ sa vyberie prvá vzorka, pre $I_{IN}=1$ druhá, atď.

<table><tr><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td></tr></table>	0	0	0	0	<table><tr><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td></tr></table>	0	0	0	1	<table><tr><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td></tr></table>	0	0	1	1	<table><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td></tr></table>	0	1	1	1	<table><tr><td>1</td><td>1</td></tr><tr><td>1</td><td>1</td></tr></table>	1	1	1	1
0	0																							
0	0																							
0	0																							
0	1																							
0	0																							
1	1																							
0	1																							
1	1																							
1	1																							
1	1																							
$I_{IN}=0$	$I_{IN}=1$	$I_{IN}=2$	$I_{IN}=3$	$I_{IN}=4$																				

Obr. 105 Rozptyľovacie matice pre $I_{IN} = \langle 0, 4 \rangle$

Pre väčší rozsah intenzít je nutné použiť matice vyšších rádoov. Pri vytváraní nových matic je nutné vždy pridávať jednotky na jednu novú pozíciu a je nutné zabrániť permutáciám v rámci matice, ktoré by v prípade určitých kombinácií odtieňov vstupného obrazu viedli k nesprávnemu zosvetleniu alebo stmaveniu obrazu.

V prípade dodržania uvedenej zásady nie je nutné v pamäti udržiavať matice pre všetky možné vstupné intenzity, ale je možné použiť jednu maticu, ktorej hodnoty udávajú pri akej minimálnej veľkosti vstupnej intenzity sa na výstupe v danom bode objaví jednotka.

$M_d =$	<table border="1"><tr><td>0</td><td>12</td><td>3</td><td>15</td></tr><tr><td>8</td><td>4</td><td>11</td><td>7</td></tr><tr><td>2</td><td>14</td><td>1</td><td>13</td></tr><tr><td>10</td><td>6</td><td>9</td><td>5</td></tr></table>	0	12	3	15	8	4	11	7	2	14	1	13	10	6	9	5	$M_p =$	<table border="1"><tr><td>1</td><td>5</td><td>9</td><td>2</td></tr><tr><td>8</td><td>12</td><td>13</td><td>6</td></tr><tr><td>4</td><td>15</td><td>14</td><td>10</td></tr><tr><td>0</td><td>11</td><td>7</td><td>3</td></tr></table>	1	5	9	2	8	12	13	6	4	15	14	10	0	11	7	3
0	12	3	15																																
8	4	11	7																																
2	14	1	13																																
10	6	9	5																																
1	5	9	2																																
8	12	13	6																																
4	15	14	10																																
0	11	7	3																																

Obr. 106 Rôzne rozptyľovacie matice pre $I_{IN} = \langle 0, 16 \rangle$

Pomocou matic rozmeru 4×4 na je možné vygenerovať 17 rôznych vzoriek, čo pri vstupných obrazoch so 16 úrovňami šedej nie je vyhovujúce. Táto nevýhoda sa rieši tak, že niektoré hodnoty zo stredu intervalu sa zopakujú v matici viackrát. Pre dosiahnutie 16 intenzít sa v praxi najčastejšie používa hodnota 8.

$M_d =$	<table border="1"><tr><td>0</td><td>11</td><td>3</td><td>14</td></tr><tr><td>8</td><td>4</td><td>10</td><td>7</td></tr><tr><td>2</td><td>13</td><td>1</td><td>2</td></tr><tr><td>9</td><td>6</td><td>8</td><td>5</td></tr></table>	0	11	3	14	8	4	10	7	2	13	1	2	9	6	8	5	$M_p =$	<table border="1"><tr><td>1</td><td>5</td><td>8</td><td>2</td></tr><tr><td>8</td><td>11</td><td>12</td><td>6</td></tr><tr><td>4</td><td>14</td><td>13</td><td>9</td></tr><tr><td>0</td><td>10</td><td>7</td><td>3</td></tr></table>	1	5	8	2	8	11	12	6	4	14	13	9	0	10	7	3
0	11	3	14																																
8	4	10	7																																
2	13	1	2																																
9	6	8	5																																
1	5	8	2																																
8	11	12	6																																
4	14	13	9																																
0	10	7	3																																

Obr. 107 Rôzne rozptyľovacie matice pre $I_{IN} = \langle 0, 15 \rangle$

Jednotky a nuly môžu byť v rozptyľovacích maticiach usporiadené rôznymi spôsobmi. V praxi sa najvhodnejšia matica väčšinou vyberá podľa zobrazovacieho zariadenia, pre ktoré je výsledný obraz určený. Matica M_d vytvára krížový tieňovací

vzor a je najčastejšie používaný pre výstup na obrazovku, kým matica \mathbf{M}_p vytvára bodové tieňovanie a je najčastejšie používaná pre výstup na tlačiareň.

Matice vyšších rádov môžu byť taktiež vytvorené podľa vzorca :

$$\mathbf{M}_{[4,4]} = \begin{bmatrix} \mathbf{M}_0 & \mathbf{M}_3 \\ \mathbf{M}_2 & \mathbf{M}_1 \end{bmatrix} \quad (133)$$

Kde : $\mathbf{M}_i = 4 \times \mathbf{M}_{[2,2]} + i$

$$\mathbf{M}_{[2,2]} = \begin{bmatrix} 0 & 3 \\ 2 & 1 \end{bmatrix}$$

V prípade, že výsledný obraz oproti pôvodnému nemá byť zväčšený, použije sa namiesto celej matice len jediný jej prvok, ktorého index v matici sa získa ako zvyšok po celočíselnom delení súradníc bodu rozmerom matice.

Maticové rozptyľovanie dáva veľmi dobré výsledky pre väčšinu obrazov. Pravidelné striedanie odtieňov však pri spracovaní fotografií vytvára dojem umelo vytvorených obrazov.

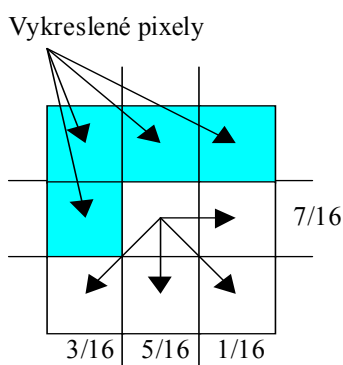
10.5.3 Distribúcia chyby



Intenzita vstupného prvku obrazu (bodu) je na výstupe postupne nahradzovaná jednotkou alebo nulou, čím nutne dochádza k strate informácie. Rozptyľovanie na základe distribúcie chyby sa snaží vstupnú informáciu v čo najväčšej miere zúžitkovať.

Vstupná intenzita I_{IN} je použitá nielen k nájdeniu najbližšej zodpovedajúcej intenzity I_{OUT} , ale zanedbaná hodnota, vznikajúca zaokrúhlením I_{IN} z väčšieho rozsahu do rozsahu $\langle 0, 1 \rangle$ je využitá na modifikáciu hodnôt susedných prvkov obrazu.

Ak má byť napríklad spracovaný prvok obrazu so vstupnou hodnotou 4, výstupnému prvku obrazu sa priradí hodnota 0 a zanedbaná hodnota 4 sa pripočíta k hodnotám susedných prvkov obrazu. Distribuovaná chyba je rozdelená medzi susedné prvky obrazu v závislosti od použitej metódy.



Obr. 108 Spôsob distribúcie chyby podľa Floyd-Steinberga

Pri distribúcii chyby musia byť, na rozdiel od predchádzajúcich metód, prvky obrazu spracúvané postupne po riadkoch. Pre aktuálny spracúvaný riadok a riadky za ním nasledujúce (ich počet závisí od veľkosti matice určujúcej koeficienty rozdelenia chyby) musí byť v pamäti vymedzené pole - pamäť chyby.

Pred vykreslením každého prvku obrazu je ku vstupnej hodnote I_{IN} pripočítaná doteraz získaná chyba (z pamäte chyby) a až potom je výsledná hodnota prevedená do výstupného rozsahu. Zanedbaná hodnota sa opäť rozdelí a pripočíta do pamäte chyby.

Vzhľadom na celočíselný charakter spracúvaných hodnôt jasových úrovní a neceločíselný charakter podielov chyby distribuovaných do jednotlivých susedných prvkov obrazu je bezpodmienečne nutné zabezpečiť, aby nedochádzalo k strate celočíselným delením. Jedným z možných riešení je, že posledný koeficient chyby sa vypočíta ako rozdiel celej chyby a súčtu všetkých predchádzajúcich koeficientov tejto chyby.

Najčastejšie používanou metódou pre distribúciu chyby je použitie koeficientov podľa *Floyda-Steinberga*. Je to veľmi rýchla metóda, pretože pri nej dochádza k deleniu jednotlivých koeficientov šestnástimi, čo je možné veľmi jednoducho implementovať bitovou rotáciou. Výsledky dosahované touto metódou sú pomerne dobré. Okrem tejto distribúcie sa používajú aj iné, ktoré si tu ale neuvedieme.

	x	7
3	5	1

Obr. 109 Distribúcia chyby - Floyd-Steinberg

V praxi sa často vyskytuje situácia, pri ktorej je nutné farebný obraz zobraziť na zariadení, ktoré má k dispozícii nižší počet farieb ako je v zdrojovom obraze. Uvedené rozptyľovacie metódy popísané skôr je možné po niekoľkých úpravách použiť aj pre farebné obrazy. Postup pre použitie rozptyľovacích metód pre farebné obrazy je nasledujúci :

1. Pre daný obraz sa vytvorí paleta podľa požiadaviek niektorou z metód popísaných vyššie a algoritmus sa nastaví na prvý bod obrazu.
2. Vyberie sa bod a nájde sa k nemu najbližšia farebná hodnota vo vytvorenej palete.
3. Pre daný bod sa určí zanedbaná hodnota, ktorá vznikne zaokrúhlením pôvodnej farby na farbu z palety.
4. Na základe zanedbanej hodnoty a použitej metódy sa rozhodne o prípadnom zvýšení výslednej intenzity farby výsledného bodu.
5. Ak ešte nie je spracovaný celý obraz, posunie sa na ďalší bod a pokračuje sa ďalej krokom 2.

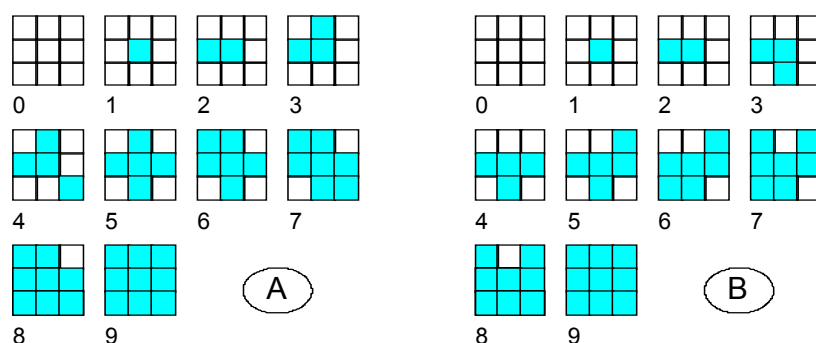
Kroky 2 až 5 sa vykonávajú postupne pre jednotlivé farebné zložky obrazu.

10.6 Poltónovanie (halftoning)



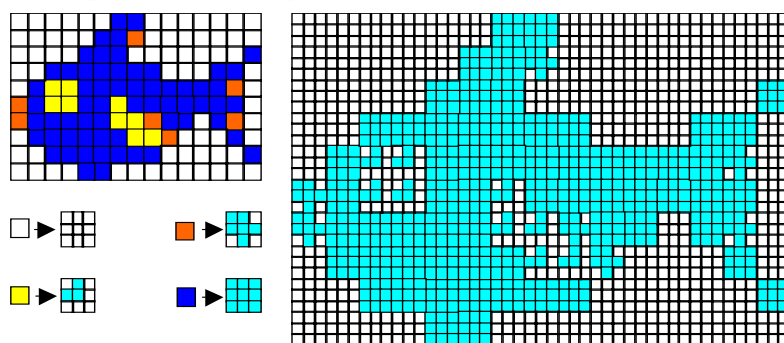
Pri poltónovaní, tak ako už bolo naznačené skôr, dochádza k náhrade jedného bodu (pixelu) pôvodného obrazu (spravidla s väčším počtom farieb) na maticu bodov výsledného obrazu (s monochromatickým výstupom) najčastejšie pomocou určitej nahradzovacej masky. Je jasné, že dôjde k nadvzorkovaniu, t.j. že výsledný obraz bude väčší ako pôvodný. Techniky poltónovania boli vlastne vyvinuté pre transformáciu z obrazu s viac úrovňami šedi na čierno-bielu paletu, pričom je po

transformácii požadovaný približne rovnaký vnem. Existuje niekoľko techník poltónovania. Najčastejšie sa však používa priama metóda náhrady pomocou spomínaných nahradzovacích masiek. Na nasledujúcom obrázku budú ukázané dve rozličné sady nahradzovacích masiek s rozmerom nahradzovacej matice 3x3.



Obr. 110 Príklady poltónovacích masiek 3x3

S poltónovaním sa stretávame napr. pri výstupe na monochromatickú maticovú tlačiareň alebo laserovú tlačiareň, prosté všade tam, kde máme k dispozícii len jednu farbu. Ukážku prevodu obrázku poltónovaním ukazuje nasledujúci obrázok.

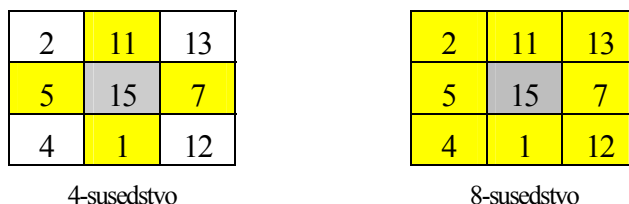


Obr. 111 Ukážka zobrazenia obrázu poltónovaním

10.7 Medián filter



Tento filter sa používa najmä v technikách spracovania obrázu na odstránenie impulzného šumu. Pri aplikovaní na originál však môže mať ešte jeden efekt. Okrem toho, že odstráni impulzný šum, spôsobí na "zdravých" hranách degradáciu, čo má vizuálne efekt mierneho rozmazania (roztostrenia) a teda aj akéhosi akvarelového efektu (akokeby ste kreslili vodovými farbami). Princíp efektu (filtra) je jednoduchý. V praxi sa pracuje buď so 4-susedstvom alebo 8-susedstvom (častejšie) ako ukazuje nasledujúci obrázok:



Obr. 112 Masky pre určenie medián filtra

Poznámky:

Potom sa zoberú indexy farieb (intenzity) riešeného pixelu a príslušných susedov. Napríklad pre uvedené osemsusedstvo dostaneme zoznam (2,11,13,5,15,7,4,1,12). Potom tento zoznam napr. vzostupne zotriedime, čím dostaneme (1,2,4,5,7,11,12,13,15). Následne sa pre stredný, riešený pixel, zvolí nová hodnota farby ako stredná hodnota (medián) z tohto zoznamu t.j. 7.

10.8 Spriemernenie farieb



Tento postup je podobný ako predchádzajúci medián filter a za určitých okolností dá na riešenom pixely rovnaký výsledok. Rozdiel je však v nasledovnom. Aj keď vychádzame z rovnakých základov ako v predchádzajúcom prípade, po získaní príslušného zoznamu sa hodnoty sčítajú a vypočíta sa priemer (teda neurčuje sa z existujúcich). V tomto prípade je súčet=70 a priemer=70/9=8. Pre prácu s paletou platia obdobné odporúčania ako pre medián filter.



1. Charakterizujte problém používania a spracovania farieb v rámci počítačovej grafiky
2. Charakterizujte a popíšte model farebný model RGB
3. Charakterizujte a popíšte model farebný model CMY
4. Charakterizujte a popíšte model farebný model HSB
5. Charakterizujte a popíšte model farebný model HLS
6. Charakterizujte a popíšte model farebný model UWB
7. Popíšte v krátkosti základné prevody medzi niektorými farebnými modelmi a prevod na odtiene šedej
8. Charakterizujte gama korekciu a popíšte alfa-miešanie
9. Charakterizujte problém rozptyľovania (dithering) farieb v rámci počítačovej grafiky
10. Charakterizujte problém poltónovania (halftoning) v rámci počítačovej grafiky



V tejto kapitole sme sa naučili:

- ☐ Podľa frekvencie, ktorú vysiela svetelný zdroj je možné svetlo rozdeliť na achromatické a monochromatické. Svetlo je charakterizované niekoľkými svojimi atribútmi: farba, jas, sýtosť a svetlosť.
- ☐ Poznáme dva základné spôsoby kombinácie (miešania) farieb: aditívny a subtraktívny.
- ☐ Farebný model je charakterizovaný množinou základných farieb, spôsobom ich miešania a pravidlami menenia farebných charakteristík.

- ☐ Medzi najviac používané farebné modely v počítačovej grafike patria: RGB, CMY(K), HSB a HLS.
- ☐ Základné farby, s ktorými sa pracuje v počítačovej grafike sú: čierna, modrá, zelená, tyrkysová, červená, fialová, žltá a biela.
- ☐ Jeden z farebných modelov, ktoré vychádzajú z fyziologického vnímania farieb a je veľmi v súčasnosti používaný je model UWB ($Y C_B C_R$).
- ☐ V prípade potreby je možné jednotlivé modely medzi sebou prevádzať, prípadne previesť na odtiene šedej.
- ☐ Odchýlke hodnoty pixelu od skutočného jas pixelu na obrazovke hovoríme gama faktor a snahe o jej korigovanie potom gama korekcia.
- ☐ Alfa miešanie je funkcia pre obrazy polopriehľadných plôch. Koeficient priehľadnosti sa označuje ako alfa-zložka a najčastejšie sa udáva z intervalu $\langle 0,1 \rangle$.
- ☐ Rozptyľovanie (dithering) je metóda, ktorá sa snaží na zariadení s menším počtom farieb zobraziť opticky obraz, ktorý sa zdá s väčším počtom farieb. Rozptyľovacie metódy využívajú tú vlastnosť ľudského oka, že z farieb niekoľkých blízkych bodov vytvárajú dojem jediného bodu, ktorého farba je daná aditívnym zlúčením farieb pôvodných bodov.
- ☐ Najčastejšie je používané náhodné rozptyľovanie, maticové rozptyľovanie a rozptyľovanie s distribúciou chyby.
- ☐ Poltónovanie (halftoning) je špeciálnym prípadom rozptyľovania, kedy dochádza k náhrade farby pixelu len do monochromatickej bázy najčastejšie do čierno-bielej.



11 Záver

Záverom si uveďme aspoň v krátkosti niektoré ďalšie odvetvia, dnes už tak mohutného odboru akým počítačová grafika je.

V súčasnosti sa do popredia v počítačovej grafike dostávajú rôzne netradičné postupy, ktoré umožňujú používateľom vyrábať rôzne efektné kresby a dosahovať zaujímavé výsledky. Medzi tieto „netradičnosti“ môžeme zaradiť aj **fraktály**. Tieto sa používajú na generovanie rôznych, najmä abstraktných podkladov a tvarov, prípadne na rôzne zaujímavé rozptyľovacie technológie, ktoré môžu po aplikovaní na digitálny obraz vyvolať dojem, že obraz je napr. za sklom alebo v hmle. Fraktálmi sa zaoberá fraktálna geometria. Táto, ako vedná disciplína, sa rozvíja zhruba od 60-tich rokov nášho storočia. V tejto dobe sa niekoľko vedcov nezávisl na sebe pokúšalo objaviť popis chaosu prírody. Väčšina objektov v prírode je z tohto hľadiska chápania celkom chaotická, ale na druhej strane je zrejmé, že tento chaos nie je absolútny. Objekty v prírode sú síce rozložené náhodne, ale ani táto náhoda nie je celkom bez pravidiel. Násť tieto pravidlá namenalo vytvoriť aparát, ktorý by popisoval chaotické javy a dovoľoval tak človeku ich exaktné pochopenie. Ústredný pojmom celej fraktálnej geometrie je sebepodobnosť. Typickým sebepodobným objektom je kameň. Či sa pozrieme na malý kamienok alebo na veľký kameň, tak jediné, čím sa líšia, je práve ich veľkosť (matematicky sa táto vlastnosť nazýva invariancia voči zmene mierky). A práve preto sa fraktály okrem iného hodia na reprezentáciu prírodných útvarov v počítačovej grafike.

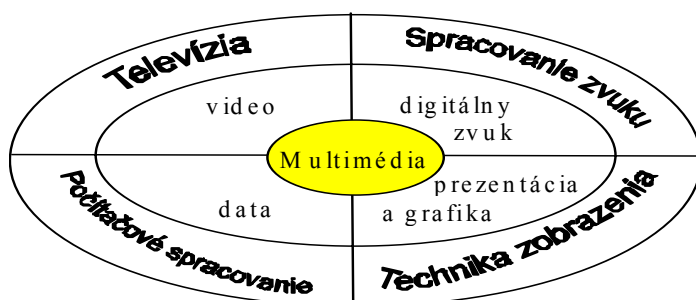


Obr. 113 Ukážky prírodných útvarov definovaných pomocou fraktálov

Ďalšou nezanedbateľnou oblasťou, kde sa výrazne uplatňujú prvky počítačovej grafiky je reklama a prezentácie firiem a organizácií. Je zrejmé, že od kvality nástrojov na prezentáciu závisí aj celková kvalita prezentácie. Ako nástroj prezentácie môže poslúžiť samotný tovar (primárny nástroj), alebo rôzne iné predmety, popisujúce tovar (sekundárny nástroj). Za sekundárne nástroje prezentácie možno považovať diaprojektory, rôzne tabule s nákresmi, či spätný projektor. To že prezentácia pomocou počítačovej grafiky nie je len okrajová oblasť počítačového priemyslu dokazuje aj skutočnosť, že priamo pre potreby počítačovej prezentácie bolo vytvorených niekoľko špecializovaných periférnych zariadení (projekčné LCD panely či veľkoplošné projektory).

Aj keď generovanie zvuku priamo s grafickou prezentáciou na počítači nesúvisí, určite má v nej svoje nezastupiteľné miesto. Ak sa k obrazu dodá kvalitný zvuk napr. pomocou kvalitnej zvukovej karty, celkový dojem z prezentácie sa tým ešte znásobí.

Prezentácia, ktorá združuje zvuk a obraz, či už statický, alebo pohyblivý sa nazýva multimediálna prezentácia. Počítačová grafika dneška je neodmysliteľne spätá s multimédiami. **Multimédiá** prenikajú do našich domovov a preto navždy zmenia spôsob, ktorým komunikujeme. Čo sú to vlastne multimédiá? Podľa firmy IBM pod tento pojem zahŕňame grafiku, prezentáciu, audio, video, animáciu, interakciu a "netradičné zariadenia". Z uvedeného vyplýva, že pojem multimédiá zastrešujú množstvo nezávislých, pomerne atraktívnych činností, ktoré až doposiaľ nemali veľa spoločného. Od tej istej firmy pochádza aj zjednodušenie tohto pojmu, ktoré graficky možno vyjadriť takto:



Obr. 114 Definícia multimédií podľa IBM

Okrem pojmu multimédií je v súčasnosti veľmi často skloňovaný aj pojem **virtuálna realita** (virtual reality). *Virtuálno-reálny systém predstavuje interaktívny počítačový systém, vytvárajúci ilúziu v danom čase neexistujúceho len syntetizovaného priestoru alebo ešte presnejšie môžeme hovoriť o tzv. dokonalej simulácii v prostredí tesného spojenia človek-počítač.* Ako mnoho iných lukratívnych odborov v oblasti počítačov aj virtuálna realita má svoje základy v počítačovej grafike. Koncom šesťdesiatych a začiatkom sedemdesiatych rokov boli predvedené prvé vstupné technické prostriedky pre interaktívnu grafiku vyvinuté prof. Brooksom. Najprv to bol prilbový displej s možnosťou trojrozmerného videnia a snímania pohybu hlavy s následnou interakciou obrazu. Druhým následníkom bola kontaktná ruka. Všetky tieto pokusy viacmenej stroskotávali na jednak rýchlosti vtedajšej výpočtovej techniky, jednak jej dosť neohrabanosti a tiež cene. Jej oživenie nastalo najmä vďaka NASA začiatkom osemdesiatych rokov a komerčnú explóziu začal svojím prilbovým displejom a kontaktnou rukavicou v roku 1989 Jaron Lanier.

Virtuálna realita aj keď už má zopár rokov za sebou, predsa je možné povedať, že k jej dokonalému nasadeniu ešte niečo bráni a to najmä: cena, pohodlnosť a pomerne slabšia grafika, aj keď súčasný rozvoj možností grafických adaptérov je veľký.

Pred nasledujúcim delením virtuálno-reálnych systémov je nutné uviesť, čo všetko vlastne spadá do oblasti VR. Okrem ďalej uvedených typických stupňov delenia je možné do VR zahrnúť aj *telerobotiku* resp. iné typy *teleprezencií* (t.j. účasť na vzdialenom deji, forma virtuálnej výuky alebo forma virtuálnej konštrukčnej kancelárie).

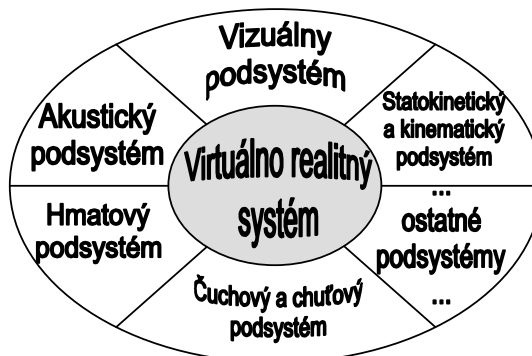
Najmä podľa technických, čiastočne aj podľa programových prostriedkov potrebných na dosiahnutie príslušných VR výsledkov môžeme deliť systémy na:

- *Vstupná VR* predstavuje v podstate len "inteligentnejšie" 3D modely a ich zobrazenie na obrazovku počítača. Spravidla je to podporené zvukom zo zvukovej karty. Interakcia s generovaným svetom je na úrovni 2D ovládačov. V podstate sa môže jednať o niektoré dobre spracované hry.

Poznámky:

- ❑ *Základná (nízka) VR*, kde základného 3D-efektu je dosahované pomocou monitorov a dvojfarebných okuliarov s použitím príslušných filtrov (napr. červený a zelený filter). Nevýhoda je menšie "vtiahnutie" používateľa do generovaného sveta a najmä čiernobiely pohľad. Interakcia s generovaným svetom je stále na úrovni 2D ovládačov (napr. myš, trackball alebo joystick).
- ❑ *Stredná VR*, pri ktorej dochádza k podstatnému vylepšeniu komunikácie používateľa s počítačom. Sú prebudované najmä dokonalejšie zobrazovacie metódy, ale aj dokonalejšie vstupné ovládacie prvky. Trojrozmerný obraz je dosahovaný buď okuliarmi s polarizačnými filtermi alebo LCD okuliarmi, čo už umožňuje vytvoriť farebný pohľad do virtuálneho sveta. Pohyb resp. interakcia smerom od používateľa je dosahovaná pomocou ovládačov s viacerými stupňami voľnosti (napr. lietajúca myš (FLYING MOUSE), spaceball, 3D joystick a pod.). Najlepším prvkom na tejto úrovni je kontaktná rukavica. Charakteristickou črtou VR-systémov na tejto úrovni je možnosť ich prevádzky aj na osobných počítačoch s príslušným vybavením, najmä dobrou a výkonnou grafickou kartou.
- ❑ *Úplné (immersive) systémy VR* umožňujú kvalitatívne inú interakciu používateľa s počítačovým svetom. Samozrejme najmä na úrovni vizuálnej, zvukovej a pohybovej. Vizuálneho 3D efektu je dosahovaného použitím špeciálnej kontaktnej prilby so vstavanými displejmi (Head mounted display - HMD). Tieto displeje môžu mať vzájomne posunutý obraz tak, aby vznikol dojem priestoru. Prilba je navyše vybavená príslušným audio systémom pre generovanie 3D zvukových efektov. Zo vstupných zariadení sem môžeme zaradiť ďalšie iné snímače polohy hlavy, rúk či nôh, simulátory fyzického odporu resp. dotyku, prípadne generátory a simulátory pôsobiace na iné ľudské receptory (napr. čuch, chuť a pod.). Pre používateľa VR-systémov tejto triedy sa začína používať označenie "cybernaut" alebo tiež vnorený aktér. Osobnosť, ktorú tento používateľ predstavuje vo virtuálnom svete sa tiež nazýva *avatar*. Samozrejme výpočtovou náročnosťou spadá tento stupeň do oblasti pracovných staníc (v súčasnosti aj na platforme PC s procesormi Pentium II a vyššie), ale najmä paralelných počítačov príp. vyšších špecializovaných počítačov.

Kategorizácia subsystémov VR je daná hlavne podľa zmyslov, na ktoré jednotlivé časti VR-systému pôsobia: *Vizuálny subsystém, Akustický subsystém, Kinematický a statokinetický subsystém, Hmatový a dotykový subsystém a Iné vnemy* (napr. vnemy čuchové, chuťové, citlivosť na feromóny, citlivosť pri chorobe, bolesť, spánok či myšlienky). Potom z *hľadiska implementácie* VR systému sa uvažuje niekoľko skôr spomínaných subsystémov. Jednotlivé dôležité zložky ukazuje nasledujúci obrázok.



Obr. 115 Podsystemy virtuálno-realitného systému

Z hľadiska počítačovej grafiky grafiky má pri virtuálno-reálnom systéme dominantné postavenie podpora vizuálneho podsystemu. Nesmieme zabúdať, že všetky činnosti spojené s vizualizáciou virtuálneho sveta musia prebiehať v reálnom čase, čo so sebou prináša mimoriadne nároky ako po stránke technického vybavenia, tak po stránke programového vybavenia. To platí aj pre implementáciu niektorých algoritmov (najmä geometrických a premietacích transformácií, definovania modelu virtuálneho sveta na úrovni rôznych kriviek a plôch, vyplňovania, riešenia viditeľnosti a realistického zobrazovania), ktoré boli obsahom tejto publikácie. Verím, že aplikáciou poznatkov načerpaných z tejto publikácie sa Vám podarí preniknúť do tajov počítačovej grafiky, aby ste potom, keď Vás uchváti tento krásny odbor, mohli siahnuť aj na jeho vrchol, aj napríklad prostredníctvom virtuálnej reality



12 Prílohy

12.1 GDI – Graphics Device Interface MS Windows

Táto príloha popisuje základné grafické funkcie MS Windows, ktoré sú nezávislé na grafických zariadeniach. Funkcie umožňujú vytváranie širokého spektra výstupov riadkov, textov a bitových máp na mnohých zariadeniach. Tieto funkcie spoločne vytvárajú Rozhranie Grafických Zariadení (GDI) vo Windows.

Táto príloha nie je referenčnou príručkou GDI, keďže táto problematika je veľmi široká. Popisuje základné princípy pri práci s paletami, perami, štetcami, bitovými mapami, textom, fontami, čiarovými a mnohoúhelníkovými funkciami.

12.1.1 Kontext (Trieda=Ikontext)

Funkcie kontextu zariadenia vytvárajú, rušia a obnovujú kontexty zariadení (DC). Kontext zariadenia je spojenie medzi aplikáciou Windows, ovládačom zariadenia a výstupným zariadením (napr. Tlačiareň alebo ploter).

Každá aplikácia môže používať funkcie GDI pre prístup na výstupné zariadenie. GDI predáva volania (ktoré sú nezávislé na výstupnom zariadení) z aplikácie do ovládača zariadenia. Tento ovladač zariadenia potom prevádza tieto volania na operácie, závislé na konkrétnom type tohoto zariadenia.

Atribúty kontextu zariadenia popisujú zvolené kresliace objekty, zvolený font, a jeho farbu, spôsob kreslenia objektov, oblasti prístupné pre výstup (orezávacie oblasti) a ďalšie dôležité informácie.

Niekedy je potrebné uchovať kontext zariadenia, aby sme mali pôvodne nastavené atribúty k dispozícii na obnovenie. Na to je možné použiť funkcie **SaveDC** a **RestoreDC**.

Odstránenie kontextu zariadenia funkciou **DeleteDC** zaisťuje, že zdieľané prostriedky nie sú odstránené skôr ako posledný kontext. Ovladač zariadenia je zdieľaný prostriedok.

12.1.1.1 GetDC (Trieda=IPolyPolygon)

Syntax: HDC GetDC(hwnd)

Táto funkcia zistí madlo kontextu zariadenia pre užívateľskú oblasť daného okna. Funkcia zistí obecný, triedny alebo súkromný kontext zariadenia v závislosti na štýle triedy, pre zvolené okno. Pri obecných kontextoch zariadenia GetDC priradzuje implicitné atribúty kontextu. U triednych alebo súkromných kontextov ponecháva predtým priradené atribúty nezmenené.

Parametre: hwnd - HWND Identifikuje okno, ktorého kontext zobrazenia má byť zistený.

Vrátená hodnota: V prípade, že funkcia prebehla úspešne, vrátená hodnota indikuje kontext zariadenia pre užívateľskú oblasť. V opačnom prípade vráti NULL.

Poznámky: Po kreslení s obecným kontextom sa musí volať funkcia **ReleaseDC**, aby bol kontext uvoľnený. Triedne a súkromné kontexty zobrazenia nemusia byť uvoľnené. Súčasne je možné použiť len päť obecných kontextov, preto nedodržanie tejto zásady môže zabrániť iným aplikáciám v prístupe ku kontextu.

Pozri tiež: HDC HWND ReleaseDC

12.1.1.2 ReleaseDC (Trieda=ILineTo)

Syntax: int ReleaseDC(hwnd,DC)

Táto funkcia uvoľňuje kontext zariadenia, aby mohol byť použitý ďalšími aplikáciami. Účinok funkcie závisí od typu kontextu zariadenia. Uvoľňuje sa iba obecný kontext zariadenia. Funkcia nemá vplyv na triedne a súkromné kontexty zariadení.

Parametre: hWnd - HWND Identifikuje okno, ktorého zariadenie sa má odstrániť.
hDC - HDC Identifikuje kontext zariadenia, ktorý sa má uvoľniť.

Vrátená hodnota: Vrátená hodnota určuje, či kontext zariadenia bol uvoľnený. V opačnom prípade je vrátená nula.

Poznámky: Aplikácia musí volať funkciu **ReleaseDC** ku každému volaniu funkcie **GetWindowDC** a **GetDC**, ktorá získa obecný kontext zariadenia.

Pozri tiež: HDC HWND GetDC

12.1.1.3 Ďalšie funkcie kontextu (Trieda=IPolyLine)

Odporúča sa preštudovať si tiež nasledujúce funkcie:

CreateCompatibleDC	- vytvára pamäťový kontext zariadenia
CreateDC	- vytvára kontext zariadenia
CreateIC	- vytvára informačný kontext
GetDCOrg	- zisťuje počiatok špecifikovaného kontextu zariadenia
RestoreDC	- obnovuje kontext zariadenia
SaveDC	- uchováva aktuálny kontext zariadenia
GetWindowDC	- vracia kontext zariadenia pre dané okno, zahrňujúc lištu, menu a rolovacie lišty. Kontext umožňuje kresliť kdekoľvek v okne.

12.1.2 Palety (Trieda=Ipaleta)

Množstvo farebných obrazoviek môže zobrazovať široké spektrum farieb. Vo väčšine prípadov je počet farieb, ktoré môžu byť zobrazené naraz obmedzený. V takýchto prípadoch ovladač obrazovky pracuje s farebnou paletou. V prípade, že ak aplikácia požaduje farbu, ktorá nieje aktuálne zobrazená, potom ovladač pridá túto farbu do palety. Vo chvíli, keď počet farieb v palete dosiahne maxima daného množstvom obrazovky, musí nová farba zaujať miesto farby už existujúcej. Výsledkom toho je, že pokiaľ celkový počet farieb, požadovaný oknami dosiahne maxima, potom farby už zobrazené nebudú správne.

Farebné palety poskytujú metódu, nezávislú na zariadení. Zariadenia, ktoré môžu zobraziť viac ako 256 farieb, túto paletu používajú. Aplikácia využíva systémovú paletu pre vytváranie a používanie jednej alebo viacerých logických paliet. Každá položka v palete obsahuje špecifikovanú farbu.

Logickú paletu môže používať viacej aplikácií, preto sa môže stať, že celkový počet požadovaných farieb je väčší ako je kapacita zariadenia.

Ak okno vyžaduje, aby boli farby zobrazené v jeho log. paletе, Windows najprv porovnajú položky v log. paletе s položkami v systémovej paletе. Pokiaľ sa niektorá farba log. palety v sys. nenajde, Windows nastaví nepoužívanú položku na túto hodnotu. Ak sú všetky položky v systémovej paletе obsadené Windows zoberú tie položky v log. paletе, ktoré neodpovedajú položkám systémovej palety a nastaví ich na čo najbližšiu hodnotu v syst. paletе. Na pomoc tomuto priradovaniu si Windows nechávajú 20 statických farieb (implicitná paleta).

Windows vždy zabezpečia, že požiadavky na farbu okna, ktoré je v popredí, ú vybavované pred ostatnými. V ostatných oknách najprv vybaví požiadavky okna, ktoré boli naposledy vo vstupnom ohnisku atd.

12.1.2.1 CreatePalette (Trieda=ICreatePalette)

Syntax: HPALETTE CreatePalette(lpLogPalette)

Táto funkcia vytvorí logickú paletu farieb.

Parametre: lpLogPalette - const LOGPALETTE FAR*
LPLOGPALETTE ukazuje na štruktúru LOGPALETTE, ktorá obsahuje informácie o farbách v logickej paletе.

Poznámky:

Vrátená hodnota: V prípade, že funkcia prebehla úspešne, vráti identifikátor logickej palety. Inak je rovná NULL.

Pozri tiež: SelectPalette RealizePalette

12.1.2.2 SelectPalette (Trieda=ICreatePalette)

Syntax: HPALETTE SelectPalette(hDC, hPalette, fPalBack)

Táto funkcia ukazuje na log. paletu, určenú parametrom hPalette, ktorý je zvolený ako paletový objekt kontextu zariadenia, identifikovaného parametrom hDC. Nová paleta sa stane paletovým objektom, GDI tento objekt použije k regulácii farieb a nahradí nim predchádzajúcu paletu.

Parametre:

- hdc - HDC Identifikuje kontext zariadenia
- hPalette - HPALETTE Identifikuje logickú paletu, ktorá sa má zvolit. Log. paleta sa tvorí pomocou funkcie CreatePalette.
- fPalBack - BOOL Určuje, či logická paleta je vynútenou paletou pozadia. V prípade, že parameter je nenulový, zvolená paleta je vždy paleta pozadia, bez ohľadu na to, či je okno vo vstupnom ohnisku.

Vrátená hodnota: V prípade, že funkcia prebehla úspešne, vráti identifikátor logickej palety, ktorá sa má nahradit' paletou určenou parametrom hPalette. Ak sa vyskytne chyba je vrátené NULL.

Poznámky: Aplikácia môže zvolit' paletu do viacerých kontextov zariadení. Potom však zmeny log. palety ovplyvnia všetky kontexty zariadení. Ak aplikácia zvolí objekt do viacerých kontextov zariadení, tieto kontexty musia patriť jednému fyzickému zariadeniu.

Pozri tiež: CreatePalette RealizePalette

12.1.2.3 RealizePalette (Trieda=ICreatePalette)

Syntax: UINT RealizePalette(hdc)

Táto funkcia mapuje do systémovej palety položky v log. palety, ktorá je aktuálne zvolená v kontexte zariadenia. Log. paleta pôsobí ako vyrovnávacia pamäť medzi aplikáciou pracujúcou s farbami a systémom. Pokiaľ má okno vstupné ohnisko a volá RealizePalette, Windows zaistujú, že budú zobrazené všetky požadované farby, až do max. počtu farieb zobraziteľných na zariadení.

Parametre: hdc - HDC Identifikuje kontext zariadenia

Vrátená hodnota: Vrátená hodnota určuje, koľko položiek v log. palety bolo mapovaných do položiek v syst. palety.

Pozri tiež: CreatePalette SelectPalette

12.1.2.4 Ďalšie funkcie pre prácu s paletou (Trieda=ICreatePalette)

AnimatePalette	Mení položky v logickej palety, pričom nové položky sú zaradené do systémovej palety okamžite.
GetNearestPaletteIndex	Zistí index položky v logickej palety, ktorá je najbližšia špecifikovanej RGB hodnote.
GetPaletteEntries	Vytáha položky z log. palety.
GetSystemPaletteEntries	Zistí rozsah položiek syst. palety.
GetSystemPaletteUse	Určuje, či má aplikácia prístup k celej syst. palety.
SetPaletteEntries	Pridáva nové položky do log. palety, pričom Windows nemapujú tieto položky na syst. paletu, kým aplikácia paletu nerealizuje.
SetSystemPaletteUse	Presúva farby bod po bode do systémovej palety. Takto môže neaktívne okno zmeniť svoje farby do tvaru, vyhovujúceho aktuálnej palety, bez nutnosti prekreslenia zákaznickej oblasti.

12.1.3 Štetce (Trieda=Istetec)

Aplikácia Windows môže pri vytváraní výstupu používať ktorýkoľvek z nástrojov (bitová mapa, štetec alebo pero). Aplikácia použije pero pre nakreslenie obrysov oblasti a vyplní ju definovaným štetcom.

V GDI existuje sedem dopredu predefinovaných štetcov. Jeden z nich si aplikácia vyberá funkciou **GetStockObject**. Následujúci zoznam uvádza zoznam týchto štetcov: čierny, tmavo šedý, šedý, svetlo šedý, nulový a biely.

Taktiež existuje šesť šrafovaných vzorkov štetcov. Aplikácia si vyberá jeden z nich funkciou **CreateHatchBrush**. Preddefinované sú nasledovné šrafované vzorky: zpäťne diagonálny, diagonálny, krížový, vodorovný krížový diagonálny a sivý.

12.1.3.1 CreateBrushIndirect

Syntax: **HBRUSH CreateBrushIndirect(lpLogBrush)**

Táto funkcia vytvorí logický štetec, ktorý má štýl, farbu a daný vzor v dátovej štruktúre, na ktorý ukazuje parameter lpLogBrush. Štetec môže používať akekoľvek zariadenie zvolené za aktuálne.

Parametre: lpLogBrush - LOGBRUSH FAR* Ukazuje na štruktúru LOGBRUSH, ktorá obsahuje informácie o štetci.

Vrátená hodnota: V prípade, že funkcia prebehla úspešne, vráti identifikátor logického štetca. Inak je rovná NULL.

Poznámky: Štetec vytvorený s použitím monochromatickej bitmapy, je nakreslený s použitím aktuálnych farieb textu a pozadia. Body predstavované bitom nastaveným na 0, budú nakreslené textovou farbou a body predstavované bitom nastaveným na 1, budú nakreslené farbou pozadia.

Pozri tiež: GetDC ReleaseDC CreateHatchBrush CreatePatternBrush CreateSolidBrush
GetStockObject SelectObject DeleteObject

12.1.3.2 CreateHatchBrush

Syntax: **HBRUSH CreateHatchBrush(hIndex, crColor)**

Táto funkcia vytvorí logický štetec so špecifikovaným šrafovaným vzorom a farbou. Tento štetec môže používať akekoľvek zariadenie zvolené za aktuálne.

Parametre: nIndex - int Určuje štýl šrafovania. Môže mať jednu z nasledujúcich hodnôt:
HS_BDIAGONAL 45-stupňové šrafovanie (zl'ava do prava)
HS_CROSS Horizontalne a verticalne šrafovanie krížom
HS_DIAGCROSS 45-stupňové šrafovanie krížom
HS_FDIAGONAL 45-stupňové šrafovanie zostupne (zl'ava do prava)
HS_HORIZONTAL Horizontalne šrafovanie
HS_VERTICAL Vertikálne šrafovanie
crColor - COLORREF Určuje farbu popredia štetca.

Vrátená hodnota: Ak funkcia prebehla úspešne, vrátená hodnota indikuje logický štetec.

Pozri tiež: GetDC ReleaseDC CreateBrushIndirect CreatePatternBrush CreateSolidBrush
GetStockObject SelectObject DeleteObject

12.1.3.3 CreatePatternBrush

Syntax: **HBRUSH CreatePatternBrush(hBitmap)**

Táto funkcia vytvorí logický štetec so vzorom, špecifikovaným parametrom hBitmap. Štetec môže používať akekoľvek zariadenie zvolené za aktuálne.

Poznámky:

Parametre: hBitmap - HBITMAP Identifikuje bitovú mapu. Predpokladá sa, že bola vytvorená funkciou **CreateBitmap**, **CreateBitmapIndirect** alebo **CreateCompatibleBitmap**. Minimálna veľkosť bitovej mapy je 8x8.

Vrátená hodnota: Ak funkcia prebehla úspešne, vrátená hodnota identifikuje logický štetec, v opačnom prípade NULL.

Poznámky: Bitmapy sa používajú ako vzor 8 x 8 bodov. V prípade, že bitmapa je širšia, Windows použijú iba prvých 8 x 8 bitov z ľavého horného rohu bitmapy (v našom príklade bola použitá bitmapa 64 x 30 bodov so vzorom šachovnice).

Štetec so vzorom môže byť zrušený funkciou DeleteObject, čo neovplyvní združení bitovú mapu.

Pozri tiež: GetDC ReleaseDC CreateBrushIndirect CreateHatchBrush CreateSolidBrush GetStockObject SelectObject DeleteObject

12.1.3.4 CreateSolidBrush

Syntax: HBRUSH CreateSolidBrush(crColor)

Táto funkcia vytvorí logický štetec so špecifikovanou farbou. Štetec môže používať akekoľvek zariadenie zvolené za aktuálne.

Parametre: crColor - COLORREF Určuje farbu štetca.

Vrátená hodnota: Ak funkcia prebehla úspešne, vrátená hodnota identifikuje logický štetec, v opačnom prípade NULL.

Pozri tiež: GetDC ReleaseDC CreateBrushIndirect CreateHatchBrush CreatePatternBrush GetStockObject SelectObject DeleteObject

12.1.3.5 GetStockObject

Syntax: HANDLE GetStockObject(nIndex)

Táto funkcia vracia madlo jedného z preddefinovaných kmeňových pier, štetcov alebo fontov.

Parametre: nIndex - int Špecifikuje typ požadovaného kmeňového objektu. Môže to byť jedna z hodnôt:

Hodnota Význam

BLACK_BRUSH	Tmavý štetec
DKGRAY_BRUSH	Tmavo šedý štetec
GRAY_BRUSH	Šedý štetec
HOLLOW_BRUSH	Prázdny štetec
LTGRAY_BRUSH	Svetlošedý štetec
NULL_BRUSH	Nulový štetec
WHITE_BRUSH	Biely štetec
ANSI_FIXED_FONT	Pevný systémový ANSI font
ANSI_VAR_FONT	Variabilný systémový ANSI font
DEVICE_DEFAULT_FONT	Font závislý na zariadení
OEM_FIXED_FONT	Pevný font závislý na OEM
SYSTEM_FONT	Systémový font. Implicitne Windows používa systémový font pre vykresľovanie menu, ovládačov dialógových okien a iného textu.
SYSTEM_FIXED_FONT	Systémový font s pevnou šírkou, používaný v predchádzajúcich verziách Windows.
DEFAULT_PALETTE	Implicitná paleta farieb. Táto paleta obsahuje 20 statických farieb vždy prítomných v systémovej

palette pre porovnanie farieb v logických paletách okien v pozadí.

Vrátená hodnota: Ak bola funkcia prevedená úspešne, vráti hodnotu identifikujúcu požadovaný objekt. Inak je to NULL.

Poznámky: Objekty DKGRAY_BRUSH, GRAY_BRUSH a LTGRAY_BRASH by sa mali používať ako štetce pozadia, príp. iné účely v okne, ktorého trieda štýly neurčuje štýly CS_HREDRAW a CS_VREDRAW. Použitie šedého kmeňového štetca v týchto oknách môže viesť k zlému zarovnávaní vzorku štetca pri presunoch alebo zmene veľkosti okna.

Pozri tiež: RGB COLORREF HPEN LOGPEN SelectObject LineTo

12.1.3.6 SelectObject

Syntax: **HGDIOBJ** *SelectObject(hdc, hobject)*

SelectObject ukazuje na log. objekt, určený parametrom hObject. Nový objekt nahradí predchádzajúci objekt rovnakého typu. Kontext zariadenia môže mať šesť zvolených objektov (pero, štetec, font bitovú mapu, oblasť a logickú paletu. Súčasne však môže byť iba jeden objekt daného typu. Funkcia SelectObject nezvolí log. paletu, na to je potrebné použiť funkciu SelectPalette.

Parametre: hdc - HDC Identifikuje kontext zariadenia.

hObject - HANDLE Identifikuje objekt, ktorý sa má zvoliť. Môže to byť jeden z nasledujúcich objektov a musel byť vytvorený pomocou nasledujúcich funkcií:

<i>Bitová mapa</i>	CreateBitmap,	CreateBitmapIndirect,
	CreateCompatibleBitmap,	CreateDIBitmap
<i>Štetec</i>	CreateBrushIndirect,	CreateDIBPatternBrush,
	CreateHatchBrush,	CreatePatternBrush,
	CreateSolidBrush	
<i>Font</i>	CreateFont,	CreateFontIndirect

Vrátená hodnota: Identifikuje objekt, ktorý sa nahrádza iným objektom, určeným parametrom **hObject**. Pokiaľ sa vyskytne nejaká chyba, je vrátená hodnota NULL.

Poznámky: V prípade, že zvolíte font, pero alebo štetec použitím funkcie **SelectObject**, potom GDI prideli tomuto zvolnému objektu priestor vo svojom dátovom segmente. Tento priestor je však obmedzený. Preto je potrebné zmazať každý objekt, ktorý už nepotrebuje, pomocou funkcie **DeleteObject**.

Pozri tiež: GetDC ReleaseDC CreateBrushIndirect GetStockObject SelectObject DeleteObject

12.1.3.7 DeleteObject

Syntax: **BOOL** *DeleteObject(hObject)*

Táto funkcia zruší z pamäti logické pero, štetec, font, bitovú mapu, oblasť alebo paletu, uvoľnením všetkej systémovej pamäti združenej z objektom. Akonáhle je objekt zrušený, madlo **hObject** nieje platné.

Parametre: hObject - HANDLE Identifikuje madlo pre rušený objekt.

Vrátená hodnota: Určuje, či bol špecifikovaný objekt zrušený.

Poznámky: Objekt, ktorý má byť zrušený, by nemal byť aktuálne zvolený do kontextu zariadenia

Pozri tiež: GetDC ReleaseDC CreateBrushIndirect GetStockObject SelectObject DeleteObject

12.1.3.8 Ďalšie funkcie pre prácu so štetcom

GetBrushOrg	Funkcia zistí začiatok aktívneho štetca pre daný kontext zariadenia
GetBrushOrgEx	Funkcia zistí začiatok aktívneho štetca pre daný kontext zariadenia
GetStockObject	Funkcia zistí madlo jedného z preddefinovaných kmeňových pier, štetcov, fontov, alebo palet.
SetBrushOrg	Nastaví počiatok všetkých štetcov, vybraných do daného kontextu zariadenia.
LOGPEN	štruktúra definujúca štýl, šírku a farbu pera. <pre>typedef struct tagLOGPEN { UINT lopnStyle; POINT lopnWidth; COLORREF lopnColor; } LOGPEN;</pre>
COLORREF	32-bitová hodnota, používa sa ako hodnota farby
Makro RGB	zvolí farbu na základe špecifikovaných zložiek a schopností výstupného zariadenia. <pre>COLORREF RGB(cRed, cGreen, cBlue) BYTE cRed; /* červená zložka farby */ BYTE cGreen; /* zelená zložka farby */ BYTE cBlue; /* modrá zložka farby */</pre>

12.1.4 Perá (Trieda=Ipero)

Aplikácia Windows môže pri vytváraní výstupu používať ktorýkoľvek z nástrojov (bitová mapa, štetec alebo pero). Aplikácia použije pero pre nakreslenie obrysov oblasti a vyplní ju definovaným štetcom.

12.1.4.1 CreatePen

Syntax: **HPEN CreatePen(nPenStyle, nWidth, crColor)**

Táto funkcia vytvára logické pero so špecifikovaným štýlom, šírkou a farbou. Takto vytvorené pero môže byť vybrané ako aktuálne pre kreslenie na ľubovoľné zariadenie.

Parametre: nPenStyle - int Určuje štýl pera. Môže to byť jedna z nasledujúcich hodnôt:

Štýl pera	Hodnota
PS_SOLID	0
PS_DASH	1
PS_DOT	2
PS_DASHDOT	3
PS_DASHDOTDASH	4
PS_NULL	5
PS_INSIDEFRAME	6

Pokiaľ je šírka pera väčšia než 1 a štýl pera je **PS_INSIDEFRAME**, čiara bude kreslená vnútom rámčeka všetkých grafických prvkov okrem mnohoholníkov a lomených čiar.

nWidth - int Určuje šírku pera (v logických jednotkách).

crColor - COLORREF Určuje farbu pera.

Vrátená hodnota: V prípade, že funkcia prebehla úspešne, vrátená hodnota identifikuje logické pero. V opačnom prípade je rovna NULL.

Poznámky: Pera s fyzickou šírkou väčšou než jeden budú vždy nakreslené štýlom PS_SOLID.

Pozri tiež: RGB COLORREF HPEN LOGPEN SelectObject GetStockObject CreatePenIndirect

12.1.4.2 CreatePenIndirect

Syntax: `HPEN CreatePenIndirect(lpLogPen)`

Táto funkcie logické pero, ktoré má štýl, šírku a farbu dané v štruktúre, na ktorú ukazuje parameter lpLogPen.

Parametre: lpLogPen - LOGPEN FAR * Ukazuje na datovú štruktúru LOGPEN, ktorá obsahuje informácie o logickom pere.

Vrátená hodnota: Identifikuje objekt logického pera, ak prebehla funkcia úspešne. V opačnom prípade NULL.

Poznámky: Perá s fyzickou šírkou väčšou než jeden budú vždy nakreslené štýlom PS_SOLID.

Pozri tiež: RGB COLORREF LOGPEN SelectObject GetStockObject LineTo CreatePen

12.1.4.3 Ďalšie funkcie pre prácu s perom

GetStockObject	Funkcia zistí madlo jedného z preddefinovaných kmeňových pier, štetcov, fontov, alebo palet.
SelectObject	Vyberie objekt za aktuálny.
DeleteObject	Odstráni objekt.
LOGPEN	štruktúra definujúca štýl, šírku a farbu pera. <pre>typedef struct tagLOGPEN { /* lgpn */ UINT lopnStyle; POINT lopnWidth; COLORREF lopnColor; } LOGPEN;</pre>
COLORREF	32-bitová hodnota, používa sa ako hodnota farby
Makro RGB	zvolí farbu na základe špecifikovaných zložiek a schopností výstupného zariadenia. <pre>COLORREF RGB(cRed, cGreen, cBlue) BYTE cRed; /* červená zložka farby */ BYTE cGreen; /* zelená zložka farby */ BYTE cBlue; /* modrá zložka farby */</pre>

12.1.5 Atribúty

Tieto funkcie vplyvajú na vzhľad výstupu z Windows.

Režim a farba pozadia

Čiarový výstup môže byť plný alebo prerušovaný. Pokiaľ je prerušovaný, priestor medzi prerušením môže byť vyplnený nastavením režimu pozadia na **OPAQUE** a následným vyberom farby. Nastavením režimu pozadia na **TRANSPARENT** bude priestor medzi prerušeniami zachovaný v pôvodnom stave. Toto nastavenie vykonávajú funkcie **SetBkMode** a **SetBkColor**.

Windows vytvárajú štetce na displeji, kombinujú farbu existujúcu na displeji s farbou štetca. Táto operácia sa nazýva rastrovacia. Pokiaľ implicitné rastrovacie operácie nevystačia požiadavkom aplikácie, je možné použiť funkciu **SetROP2**.

Režim rozširovania

Pokiaľ aplikácia kopíruje bitovú mapu na zariadenie a je nutné túto kópiu zväčšiť, príp. zmenšiť ešte pred vykreslením môžeme riadiť efekt funkcií **StretchBlt** a **StretchDIBlt**, ktoré tieto zmeny vykonávajú volaním funkcie **SetStretchBltMode**. Táto funkcia nastavuje režim rozširovania pre kontext zariadenia.

Farba textu

Vzhľad textového výstupu je limitovaný iba počtom fontov a farieb, ktoré sú k dispozícii na danom zariadení. Funkcia **SetBkColor** nastavuje farbu pozadia textu a funkcia **SetTextColor** nastavuje farbu vlastného znaku.

12.1.5.1 GetBkColor

Syntax: COLORREF GetBkColor(hdc)

Funkcia vracia aktuálnu farbu pozadia určeného zariadenia.

Parametre: hdc HDC - Identifikuje kontext zariadenia

Vrátená hodnota: Určuje farebnú hodnotu RGB, ktorá identifikuje aktuálnu farbu pozadia.

Pozri tiež: GetDC ReleaseDC CreateSolidBrush

12.1.5.2 GetBkMode

Syntax: int GetBkMode(hDC)

Funkcia vracia režim pozadia určeného zariadenia. Režim pozadia je používaný s textom, štetcom so šrafovaním a perom, ktoré nemá štýl plnej čiary.

Parametre: hDC - HDC Identifikuje kontext zariadenia

Vrátená hodnota: Určuje aktuálny režim pozadia. Môže to byť **OPAQUE** alebo **TRANSPARENT**.

Pozri tiež: SetBkMode GetBkColor SetBkColor GetTextColor

12.1.5.3 GetTextColor

Syntax: DWORD GetTextColor(hdc)

Funkcia vráti aktuálnu farbu textu. Farba textu definuje popredie textu písaného funkciami **TextOut** alebo **ExtTextOut**.

Parametre: hDC - HDC Identifikuje kontext zariadenia.

Vrátená hodnota: Vrátená hodnota špecifikuje akt. farbu textu ako farebnú hodnotu RGB.

Pozri tiež: SetBkMode getBkMode GetBkColor SetBkColor

12.1.5.4 SetBkColor

Syntax: DWORD SetBkColor(hdc, crColor)

Funkcia nastavuje aktuálnu farbu pozadia na farbu určenú parametrom **crColor**, alebo na najbližšiu zobraziteľnú farbu. Pokiaľ je nastavený režim pozadia **OPAQUE**, GDI použije farbu pozadia k vyplneniu medzier medzi riadkami písmen, medzi šrafovanými čiarami v štetci a na vyplnenie buniek znakov.

Parametre:	hDC	- HDC Identifikuje kontext zariadenia
	crColor	- COLORREF Určuje novú farbu pozadia.

Vrátená hodnota: Vrátená hodnota určuje predchádzajúcu farbu pozadia ako farebnú hodnotu RGB. V prípade chyby, je vrátená hodnota 0x80000000.

Pozri tiež: SetBkMode GetBkColor SetTextColor GetTextColor

12.1.5.5 SetTextColor

Syntax: DWORD SetTextColor(hdc, crColor)

Funkcia nastavuje aktuálnu farbu textu určenú parametrom **crColor**, prípadne najbližšiu fyzicky zobraziteľnú farbu. Farba textu definuje popredie textu písaného funkciami **TextOut** alebo **ExtTextOut**.

Parametre:	hDC	- HDC Identifikuje kontext zariadenia.
	crColor	- COLORREF Určuje farbu textu.

Vrátená hodnota: Vrátená hodnota špecifikuje predchádzajúcu farbu textu ako hodnotu RGB.

Pozri tiež: SetBkMode GetBkColor GetTextColor TextOut

12.1.5.6 Ďalšie funkcie pre atribúty

GetPolyFillMode	Zisťuje aktuálny režim vyplňovania mnohoúhelníkov.
GetROP2	Zisťuje akt. režim kreslenia.
GetStretchBltMode	Zisťuje akt. režim rozširovania.
SetPolyFillMode	Nastavuje režim vyplňovania.
SetROP2	Nastavuje režim kreslenia.
SetStretchBltMode	Nastavuje režim rozširovania.

12.1.6 Čiary

Funkcie čiarového výstupu vyžadujú súradnice v logických jednotkách, ktoré GDI používa na kreslenie čiar v logickom priestore. Použitie logických jednotiek zabezpečuje nezávislosť aplikácii na zariadení. GDI mapuje túto čiaru z logického priestoru do fyzického na zariadenie. Počet logických jednotiek, ktoré GDI mapuje na jeden bod zariadenia, závisí na aktuálnom mapovacom režime. Pri kreslení čiar sa nevyskresľuje posledný bod čiar.

Pokiaľ aplikácia kreslí čiaru a nevytvorí pred tým pero, GDI použije implicitné pero. Je to pero čiernej farby, šírky jedného bodu s mapovacím režimom **MM_TEXT**. Iné než implicitné pero sa vytvára pomocou funkcie **CreatePen**. Hneď po vytvorení pera, sa môže pero vybrať do kontextu zariadenia funkciou **SelectObject**.

12.1.6.1 Arc

Syntax: BOOL Arc(hdc, X1, Y1, X2, Y2, X3, Y3, X4, Y4)

Funkcia nakreslí eliptický oblúk. Stred oblúka je stredom ohraničujúceho pravouholníka, určeného bodmi (X1,Y1) a (X2,Y2). Oblúk začína v bode (X3,Y3) a končí v bode (X4,Y4). Pri kreslení oblúka sa pero pohybuje proti smeru hodinových ručičiek. Keďže oblúk nedefinuje uzavretú oblasť, nie je vyplnený.

Parametre:

- hDC - HDC Identifikuje kontext zariadenia
- X1 - int Logická súradnica x ľavého horného rohu pravouholníka
- Y1 - int Logická súradnica y ľavého horného rohu pravouholníka
- X2 - int Logická súradnica x pravého dolného rohu pravouholníka
- Y2 - int Logická súradnica y pravého dolného rohu pravouholníka
- X3 - int Logická súradnica x počiatočného bodu oblúka
- Y3 - int Logická súradnica y počiatočného bodu oblúka
- X4 - int Logická súradnica x koncového bodu oblúka
- Y4 - int Logická súradnica y koncového bodu oblúka

Vrátená hodnota: Určuje, či je oblúk nakreslený. Ak áno, má nenulovú hodnotu.

Pozri tiež: GetDC ReleaseDC CreateSolidBrush SelectObject DeleteObject MoveTo

12.1.6.2 LineDDA

Syntax: void LineDDA(X1, Y1, X2, Y2, lpLineFunc, lpData)

Funkcia vypočítava postupne body čiar, ktorá začína bodom (X1, Y1). Koncový bod nie je súčasťou čiar. Pre každý bod čiar, funkcia **LineDDA** volá funkciu, danú aplikáciou, ktorá je určená parametrom **lpLineFunc**, predáva jej súradnice aktuálneho bodu a parameter **lpData**.

Poznámky:

Parametre:	X1	- int Logická súradnica x počiatočného bodu
	Y1	- int Logická súradnica y počiatočného bodu
	X2	- int Logická súradnica x koncového bodu
	Y2	- int Logická súradnica y koncového bodu
	lpLineFunc	- FARPROC Je adresa inštancie funkcie, dodanej aplikácii
	lpData	- LPSTR Ukazuje na dáta dodané aplikácii

Vrátená hodnota: Nieje**Poznámky:** Adresa, ktorá sa predáva pomocou parametra lpLineFunc, musí byť vytvorená použitím funkcie **MakeProcInstance**.*Pozri tiež:* GetDC ReleaseDC CreateSolidBrush SelectObject DeleteObject MoveTo LineTo

12.1.6.3 LineTo

Syntax: BOOL LineTo(hDC, X, Y)

Funkcia kreslí čiaru z akt. pozície až k bodu, určenému parametrami X a Y. Tento bod sa do čiar nezahŕňa. Pokiaľ nevznikne žiadna chyba, pozícia sa nastaví na (X,Y).

Parametre:	hDC - HDC Určuje kontext zariadenia
	X - int Určuje logickú x-súradnicu koncového bodu čiar
	Y - int Určuje logickú y-súradnicu koncového bodu čiar

Vrátená hodnota: Určuje, či čiara je alebo nie je nakreslená. Je nenulová, ak je čiara nakreslená.*Pozri tiež:* GetDC ReleaseDC CreateSolidBrush SelectObject DeleteObject MoveTo PolyLine

12.1.6.4 MoveTo

Syntax: DWORD MoveTo(hDC, X, Y)

Funkcia presúva aktuálnu pozíciu do bodu, určeného parametrami X a Y.

Parametre:	hDC - HDC Identifikuje kontext zariadenia
	X - int Určuje logickú x-súradnicu nového bodu
	Y - int Určuje logickú y-súradnicu nového bodu

Vrátená hodnota: Vrátená hodnota určuje x a y-súradnice predchádzajúcej pozície. Y- súradnica sa nachádza vo vyššom slove, x-súradnica sa nachádza v nižšom slove.**Poznámka:** Funkcia **MoveTo** vo Visual C++ je nahradená funkciou **MoveToEx**.*Pozri tiež:* GetDC ReleaseDC CreateSolidBrush SelectObject DeleteObject LineTo PolyLine

12.1.6.5 PolyLine

*Syntax: BOOL PolyLine(hDC, lpPoints, nCount)*Funkcia kreslí mnohoúholník pozostávajúci z dvoch alebo viacerých bodov. Mnohouholník sa vyplní aktuálnym vyplňovacím režimom. Popis mnohoúholníkového vyplňovacieho režimu nájdete pri funkcii **SetPolyFillMode**. Pokiaľ je to nutné mnohoúholník je automaticky uzavretý čiarou, ktorá spája posledný a prvý vrchol.

Parametre:	hDC	- HDC identifikuje kontext zariadenia
	lpPoints	- LPPOINTS Ukazuje na pole bodov, ktoré určujú vrcholy mnohoúholníka. Každý bod tohoto poľa je štruktúra POINT.
	nCount	- int Určuje počet vrcholov, daných polí.

Vrátená hodnota: Je nenulová, ak funkcia prebehla úspešne.**Poznámky:** Táto funkcia nevyužíva ani neaktualizuje akt. pozíciu. Akt. vyplňovací režim je možné zistiť a nastaviť funkciami GetPolyFillMode a SetPolyFillMode.*Pozri tiež:* GetDC ReleaseDC CreateSolidBrush SelectObject DeleteObject MoveTo LineTo

12.1.7 Elipsy

Elipsy a mnohoúhelníky sú kreslené vybraným perom a vnútro je vyplnené vybraným štetcom. Aj tieto funkcie vyžadujú súradnice v logických jednotkách. Použitie logických jednotiek zabezpečuje nezávislosť aplikácii na zariadení.

Namiesto polomeru alebo inej špecifikácie, funkcie **Chord**, **Ellipse** a **Pie** používajú ohraničujúci pravouholník, ktorým definujú veľkosť vytváraného objektu. Tento pravouholník je skrytý.

12.1.7.1 Chord

Syntax: BOOL Chord(hDC, X1, Y1, X2, Y2, X3, Y3, X4, Y4)

Funkcia nakreslí tetivu. Parametre (X1,Y1) a (X2,Y2) určujú ľavý horný a pravý dolný roh pravouholníka, ohraničujúceho elipsu. Parametre (X3,Y3) a (X4,Y4) určujú koncové body čiary, ktorá elipsu pretína. Tetiva je nakreslená zvoleným perom a vyplnená zvoleným štetcom.

Parametre:

- hDC - HDC identifikuje kontext zariadenia
- X1 - int Určuje súradnice x ľavého horného rohu pravouholníka.
- Y1 - int Určuje súradnice y ľavého horného rohu pravouholníka.
- X2 - int Určuje súradnice x pravého dolného rohu pravouholníka.
- Y2 - int Určuje súradnice y pravého dolného rohu pravouholníka.
- X3 - int určuje súradnice x jedného konca čiarového segmentu.
- Y3 - int určuje súradnice y jedného konca čiarového segmentu.
- X4 - int určuje súradnice x druhého konca čiarového segmentu.
- Y4 - int určuje súradnice y druhého konca čiarového segmentu.

Vrátená hodnota: Ak je oblúk nakreslený, vrátená hodnota je nenulová.

Pozri tiež: GetDC ReleaseDC CreateHatchBrush CreatePatternBrush CreateSolidBrush Ellipse Pie Polygon RoundRect

12.1.7.2 Ellipse

Syntax: BOOL Ellipse(hDC, X1, Y1, X2, Y2)

Funkcia nakreslí elipsu. Stred elipsy je stredom ohraničujúceho pravouholníka, určeného parametrami X1,Y1,X2,Y2.

Parametre:

- hDC - HDC Identifikuje kontext zariadenia
- X1 - int Určuje súradnice x ľavého horného rohu pravouholníka.
- Y1 - int Určuje súradnice y ľavého horného rohu pravouholníka.
- X2 - int Určuje súradnice x pravého dolného rohu pravouholníka.
- Y2 - int Určuje súradnice y pravého dolného rohu pravouholníka.

Vrátená hodnota: Vrátená hodnota je nenulová, ak je elipsa nakreslená.

Poznámky: Šírka pravouholníka určeného hodnotou X2-X1, nesmie prekročiť 32 767 jednotiek. Toto obmedzenie sa vzťahuje takisto na výšku pravouholníka.

Pozri tiež: GetDC ReleaseDC CreateHatchBrush CreatePatternBrush CreateSolidBrush Chord Pie Polygon RoundRect

12.1.7.3 Pie

Syntax: BOOL Pie(X1, Y1, X2, Y2, X3, Y3, X4, Y4)

Funkcia kreslí koláčový výsek tak, že nakreslí eliptický oblúk a jeho stred sa spája s oboma koncovými bodmi oblúka. Koláčový výsek sa vyplní zvoleným štetcom.

Poznámky:

Parametre: hDC - HDC Identifikuje kontext zariadenia

X1 - int Log. súradnica x ľavého horného rohu pravouholníka

Y1 - int Log. súradnica y ľavého horného rohu pravouholníka

X2 - int Log. súradnica x pravého dolného rohu pravouholníka

Y2 - int Log. súradnica y pravého dolného rohu pravouholníka

X3 - int Log. súradnica x počiatočného bodu oblúka. Tento bod nemusí ležať presne na oblúku.

Y3 - int Log. súradnica y počiatočného bodu oblúka. Tento bod nemusí ležať presne na oblúku.

X4 - int Log. súradnica x koncového bodu oblúka. Tento bod nemusí ležať presne na oblúku.

Y4 - int Log. súradnica y koncového bodu oblúka. Tento bod nemusí ležať presne na oblúku.

Vrátená hodnota: Je nenulová, ak je koláčový výsek nakreslený.

Poznámky: Šírka pravouholníka určeného hodnotou X2-X1, nesmie prekročiť 32 767 jednotiek. Toto obmedzenie sa vzťahuje takisto na výšku pravouholníka.

Pozri tiež: GetDC ReleaseDC CreateHatchBrush CreatePatternBrush CreateSolidBrush Chord Ellipse Polygon Rectangle

12.1.7.4 Polygon

Syntax: BOOL Polygon(hDC, lpPoints, nCount)

Funkcia kreslí mnohoúhelník pozostávajúci z dvoch alebo viacerých bodov. Mnohouhelník sa vyplní aktuálnym vyplňovacím režimom. Popis mnohoúhelníkového vyplňovacieho režimu nájdete pri funkcii SetPolyFillMode. Pokiaľ je to nutné mnohoúhelník je automaticky uzavretý čiarou, ktorá spája posledný a prvý vrchol.

Parametre: hDC - HDC identifikuje kontext zariadenia

lpPoints - LPPOINTS Ukazuje na pole bodov, ktoré určujú vrcholy mnohoúhelníka. Každý bod tohoto poľa je štruktúra POINT.

nCount - int Určuje počet vrcholov, daných poľi.

Vrátená hodnota: Je nenulová, ak funkcia prebehla úspešne.

Poznámky: Táto funkcia nevyužíva ani neaktualizuje akt. pozíciu. Akt. vyplňovací režim je možné zistiť a nastaviť funkciami GetPolyFillMode a SetPolyFillMode.

Pozri tiež: GetDC ReleaseDC CreateHatchBrush CreatePatternBrush CreateSolidBrush Ellipse Pie Polygon Chord PolyPolygon

12.1.7.5 PolyPolygon

Syntax: BOOL PolyPolygon(hDC, lpPoints, lpPolyCounts, nCount)

Funkcia kreslí sled uzavretých mnohoúhelníkov. Mnohouhelníky sa môžu, ale nemusia prekrývať.

Parametre: hDC - HDC Identifikuje kontext zariadenia

lpPoints - LPPOINT Ukazuje na maticu datových štruktúr POINT, ktoré definujú vrcholy mnohoúhelníkov. Každý mnohoúhelník musí byť uzavretý.

lpPolyCounts - LPINT Ukazuje na pole celých čísel. Každé číslo určuje počet bodov v jednom mnohoúhelníku.

nCount - int Určuje počet mnohoúhelníkov

Vrátená hodnota: Je nenulová, ak boli mnohoúhelníky nakreslené.

Pozri tiež: GetDC ReleaseDC CreateHatchBrush CreateSolidBrush Ellipse Pie Polygon Chord PolyPolygon RoundRect

12.1.7.6 Rectangle

Syntax: BOOL Rectangle(hDC, X1, Y1, X2, Y2)

Funkcia kreslí pravouholník. Vnútro pravouholníka sa vyplní zvoleným štetcom a hranice zvoleným perom.

Parametre:

- hDC - HDC Identifikuje kontext zariadenia
- X1 - int Logická súradnica x ľavého horného rohu pravouholníka
- Y1 - int Logická súradnica y ľavého horného rohu pravouholníka
- X2 - int Logická súradnica x pravého dolného rohu pravouholníka
- Y2 - int Logická súradnica y pravého dolného rohu pravouholníka

Vrátená hodnota: Je nenulová, ak bol pravouholník nakreslený.

Pozri tiež: GetDC ReleaseDC CreateHatchBrush CreatePatternBrush CreateSolidBrush Ellipse Pie Polygon RoundRect

12.1.7.7 RoundRect

Syntax: BOOL RoundRect(hDC, X1, Y1, X2, Y2, X3, Y3)

Funkcia kreslí pravouholník so zaoblenými rohmi. Vnútro pravouholníka sa vyplní zvoleným štetcom a hranice zvoleným perom.

Parametre:

- hDC - HDC Identifikuje kontext zariadenia
- X1 - int Logická súradnica x ľavého horného rohu pravouholníka
- Y1 - int Logická súradnica y ľavého horného rohu pravouholníka
- X2 - int Logická súradnica x pravého dolného rohu pravouholníka
- Y2 - int Logická súradnica y pravého dolného rohu pravouholníka

Vrátená hodnota: Je nenulová, ak bol pravouholník nakreslený.

Poznámky: Šírka pravouholníka určeného hodnotou X2-X1, nesmie prekročiť 32 767 jednotiek. Toto obmedzenie sa vzťahuje takisto na výšku pravouholníka.

Pozri tiež: GetDC ReleaseDC CreateHatchBrush CreatePatternBrush CreateSolidBrush Ellipse Pie Polygon Chord PolyPolygon

12.1.8 Bitmapy

Funkcie bitových map zobrazujú bitové mapy. Bitová mapa je matica bitov v pamäti, ktorá pokiaľ je prekopírovaná na zariadenie, definuje farbu, a vzor odpovedajúcich bodov na povrchu displeja. Vzťah medzi bitovými mapami v pamäti a bodmi na zariadení závisí na druhu tohto zariadenia. V Microsoft Windows je implementovaná skupina funkcií, ktoré vytvárajú abitoové mapy a manipulujú s nimi tak, že tieto bitové mapy môžu byť zobrazené na akomkoľvek zariadení.

Bitové mapy, nezávislé na zariadení sa skladajú z dvoch častí:

1. Dátová štruktúra BITMAPINFO, ktorá definuje formát bitovej mapy a prípadné tiež tab. farieb, používaných v tejto bitmape.
2. Pole bajtov, ktoré obsahuje jednotlivé bity bitmapy.

V závislosti na hodnotách, obsiahnutých v informačnej dátovej štruktúre, bity v bitovej mape môžu určovať explicitnú farbu (RGB) alebo indexy tabuľky farieb. Tabuľka farieb môže takisto obsahovať indexy aktuálne realizovanej farebnej palety alebo explicitné farebné hodnoty. Je dôležité vedieť, že počiatok súradnicového systému v DIB je v spodnom dolnom rohu.

12.1.8.1 BitBlt

Syntax: BOOL

BitBlt(hDestDC, X, Y, nWidth, nHeight, hSrcDC, XSrc, YSrc, dwRop)

Poznámky:

Funkcia presunie bitovú mapu zo zdrojového zariadenia, daného parametrom **hSrcDC**, na cieľové zariadenie, dané parametrom **hDestDC**. Parametre **XSrc** a **YSrc** špecifikujú počiatok na zdrojovom zariadení bitovej mapy, ktoré budú presunuté. Parametre **X**, **Y**, **nWidth** a **nHeight** špecifikujú počiatok, šírku a výšku pravouholníka na cieľovom zariadení. Parameter **dwRop** definuje, ako sú kombinované bity zdroja a cieľa.

Parametre:

- hDestDC** - HDC Identifikuje kontext zariadenia, ktoré má prijať bitovú mapu.
- X** - int Určuje log. x súradnicu ľavého rohu cieľového pravouholníka
- Y** - int Určuje log. y súradnicu ľavého rohu cieľového pravouholníka
- nWidth** - int Určuje šírku cieľového pravouholníka a zdrojovej bitovej mapy.
- nHeight** - int Určuje výšku cieľového pravouholníka a zdrojovej bitovej mapy.
- hSrcDC** - HDC Identifikuje kontext zariadenia, z ktorého bude kopírovaná bitová mapa.
- XSrc** - int Určuje log. x súradnicu ľavého rohu zdrojového pravouholníka
- YSrc** - int Určuje log. y súradnicu ľavého rohu zdrojového pravouholníka
- dwRop** - DWORD Určuje rastrovú operáciu, ktorá má byť prevedená.

Vrátená hodnota: Je nenulová, ak je bitová mapa nakreslená.

Pozri tiež: CreateBitmap LoadBitmap PatBlt StretchBlt

12.1.8.2 CreateBitmap

Syntax: HBITMAP

CreateBitmap(nWidth,nHeight,nPlanes,nBitCount,lpBits)

Funkcia vytvorí na zariadení závislú pamäťovú bitovú mapu o určenej šírke, výške a bitovom vzore. Bitová mapa môže byť použitím funkcie **SelectObject** následne zvolená ako akt. bitová mapa.

Hoci bitová mapa nemôže byť kopírovaná priamo do zobr. zariadenia, môže ju funkcia **BitBlt** kopírovať z pamäťového kontextu zariadenia do akejkol'vek kompatibilného zariadenia.

Parametre:

- nWidth** - int Určuje šírku mapy v obrazovkových bodoch.
- nHeight** - int Určuje výšku mapy v obrazovkových bodoch.
- nPlanes** - BYTE Určuje počet farebných vrstiev v bitovej mape. Každá vrstva má nWidth x nHeight x nBitCount bitov.
- nBitCount** - BYTE Určuje počet bitov farieb na obr. bod zobrazenia.
- lpBits** - LPSTR Ukazuje na plochu typu krátkych celých čísel, ktoré obsahuje počiatočné bitové hodnoty bitovej mapy. Ak je NULL, je bitová mapa neinicializovaná.

Vrátená hodnota: Identifikuje bitovú mapu, ak prebehla funkcia úspešne. Inak NULL.

Pozri tiež: BitBlt LoadBitmap PatBlt StretchBlt

12.1.8.3 LoadBitmap

Syntax: HBITMAP LoadBitmap(hInstance, lpBitmapName)

Funkcia zavádza prostriedok bitovej mapy, označený parametrom **lpBitmapName** zo spustiteľného súboru modulov, ktorý je určený parametrom **hInstance**.

Parametre:

- hInstance** - HANDLE Ukazuje na inštanciu modulu, ktorý obsahuje bitovú mapu
- lpBitmapName** - LPSTR Ukazuje na znakový reťazec, ktorý pomenováva bitovú mapu. Reťazec musí byť ukončený nulou.

Vrátená hodnota: Určuje bitovú mapu. Ak takáto bitová mapa neexistuje, vrátená hodnota je NULL.

Poznámky: Aplikácia musí volať funkciu **DeleteObject**, aby zmazala každé madlo bitovej mapy, vrátené funkciou **LoadBitmap**. Funkciou **LoadBitmap** je možné tiež použiť pre prístup k predefinovaným bitovým mapám. parameter **hInstance** musí byť nastavený na NULL a parameter **lpBitmapName** musí obsahovať jednu z hodnôt:

OBM_BTNCORNERS OBM_OLD_RESTORE OBM_BTSIZE OBM_OLD_RGARROW
 OBM_CHECK OBM_OLD_UPARROW OBM_CHECKBOXES OBM_OLD_ZOOM
 OBM_CLOSE OBM_REDUCE OBM_COMBO OBM_REDUCED OBM_DNARROW
 OBM_RESTORE OBM_DNARROWD OBM_RESTORED OBM_DNARROWI
 OBM_RGARROW OBM_LFARROW OBM_RGARROWD OBM_LFARROWD
 OBM_RGARROWI OBM_LFARROWI OBM_SIZE OBM_MNARROW
 OBM_UPARROW OBM_OLD_CLOSE OBM_UPARROWD OBM_OLD_DNARROW
 OBM_UPARROWI OBM_OLD_LFARROW OBM_ZOOM OBM_OLD_REDUCE
 OBM_ZOOMD

Parameter **lpBitmapName** môže byť tiež naplnený hodnotou vytvorenou makrom **MAKEINTERSOURCE**. V tom prípade musí byť identifikátor uložený v nižšom slove parametra **lpBitmapName** a vyššie slovo musí obsahovať nulu.

Pozri tiež: BitBlt CreateBitmap PatBlt StretchBlt

12.1.8.4 PatBlt

Syntax: **BOOL PatBlt(hDC, X, Y, nWidth, nHeight, dwRop)**

Funkcia vytvorí bitový vzor na určenom zariadení. Vzor je kombináciou vzoru, ktorý je už na zariadení a zvoleného štetca. Rastrový operačný kód, určený parametrom **dwRop**, definuje ako sa majú vzory kombinovať.

Parametre:

- hDC** - HDC Identifikuje kontext zariadenia.
- X** - int Určuje log. x-súradnicu ľavého horného rohu pravouholníka, ktorý má vzor prijať.
- Y** - int Určuje log. y-súradnicu ľavého horného rohu pravouholníka, ktorý má vzor prijať.
- nWidth** - int Určuje šírku pravouholníka (v log. jednotkách), ktorý má vzor prijať.
- nHeight** - int Určuje výšku pravouholníka (v log. jednotkách), ktorý má vzor prijať.
- dwRop** - DWORD Určuje rastrový operačný kód. Rastrové operačné kódy definujú, ako má GDI kombinovať farby vo výstupných operáciách, ktoré zahŕňujú aktuálny štetec, možnú zdrojovú bitmapu a cieľovú bitmapu.

Vrátená hodnota: Je nenulová, ak bitový vzor je nakreslený.

Pozri tiež: BitBlt CreateBitmap LoadBitmap StretchBlt

12.1.8.5 StretchBlt

Syntax: **BOOL StretchBlt(hDC, X, Y, nWidth, nHeight, hSrcDC, XSrc, YSrc, nSrcWidth, nSrcHeight, dwRop)**

Funkcia premiestňuje bitmapu zo zdrojového do cieľového pravouholníka. Pritom túto bitmapu rozširuje alebo zmenšuje. Funkcia **StretchBlt** používa rozširovací režim cieľového pravouholníka (nastavené funkciou **SetStretchBltMode**) na zistenie ako rozšíriť alebo zmenšiť bitmapu.

Funkcia premiestňuje bitmapu zo zdrojového zariadenia, daného parametrom **hSrcDC**, do cieľového zariadenia, daného **hDestDC**. Parametre **XSrc**, **YSrc**, **nSrcWidth** a **nSrcHeight** definujú počiatok a rozmery cieľového pravouholníka. Rastrová operácia definuje, ako sa zkombinuje zdrojová bitmapa a bity prítomné na cieľovom zariadení.

Poznámky:

Parametre:

- hDC - HDC Identifikuje kontext zariadenia.
- X - int Určuje log. x-súradnicu ľavého horného rohu pravouholníka, ktorý má vzor prijať.
- Y - int Určuje log. y-súradnicu ľavého horného rohu pravouholníka, ktorý má vzor prijať.
- nWidth- int Určuje šírku pravouholníka (v log. jednotkách), ktorý má vzor prijať.
- nHeight- int Určuje výšku pravouholníka (v log. jednotkách), ktorý má vzor prijať.
- hSrcDC - HDC Určuje kontext zariadenia, obsahujúci zdrojovú bitmapu.
- XSrc - int Určuje log. súradnicu x-ľavého horného rohu zdrojového pravouholníka.
- YSrc - int Určuje log. súradnicu y-ľavého horného rohu zdrojového pravouholníka.
- nSrcWidth - int Určuje šírku (v log. jednotkách) zdrojového pravouholníka.
- nSrcHeight - int Určuje výšku (v log. jednotkách) zdrojového pravouholníka.
- dwRop - DWORD Určuje rastrový operačný kód. Rastrové operačné kódy definujú, ako má GDI kombinovať farby vo výstupných operáciách, ktoré zahŕňajú aktuálny štetec, možnú zdrojovú bitmapu a cieľovú bitmapu.

Vrátená hodnota: Je nenulová, ak bitový vzor je nakreslený.

Pozri tiež: BitBlt CreateBitmap LoadBitmap

12.1.8.6 Ďalšie funkcie pre prácu s bitmapou

CreateBitmapIndirect - vytvára bitovú mapu popísanú v datovej štruktúre

CreateDiscardableBitmap - vytvára nahraditeľnú bitmapu, ktorá je kompatibilná so špecifikovaným zariadením.

FloodFill - vyplní ohraničenú plochu displeja.

GetBitmapBits - zisťuje bity špecifikovanej bitmapy v pamäti.

GetBitmapDimension - zisťuje rozmery bitmapy.

GetPixel - zisťuje RGB hodnotu daného bodu.

SetBitmapBits - nastavuje bity bitmapy.

SetBitmapDimension - nastavuje výšku a šírku bitmapy.

SetPixel - nastavuje RGB hodnotu daného bodu.

StretchBlt - kopíruje bitmapu zo zdrojového zariadenia na cieľové.

V Microsoft Windows od verzie 3.0 je implementovaná skupina funkcií, ktoré vytvárajú bitmapy a manipulujú s nimi tak, že tieto bitmapy môžu byť zobrazené na akomkoľvek zariadení daného rozlíšenia, nezávisle na metóde, ktorú toto zariadenie používa na vyjedenie farby. Medzi tieto funkcie patrí:

CreateDIBitmap - vytvorí bitmapu závislú na zariadení, z špecifikovanej bitmapy, nezávislej na zariadení (DIB), a voliteľne môže inicializovať bity tejto mapy.

GetDIBits - zisťuje stav bitov v bitmape, nezávisle na zariadení.

SetDIBits - nastavuje bity mapy v pamäti z DIB.

SetDIBitsToDevice - nastavuje bity na ploche zariadenia priamo podľa DIB.

StretchDIBits - Presúva bitmapu nezávislú na zariadení zo zdrojového pravouholníka do cieľového pravouholníka. Prevádza zväčšenie alebo zmenšenie, pokiaľ je to nutné.

12.1.9 Texty (Trieda=Itexty)

Textové funkcie zisťujú informácie o texte, menia zarovnávanie a usporiadanie textu a píšú text na zariadenie. Pre textový výstup sa používa aktuálny font.

12.1.9.1 GetTextExtent

Syntax: DWORD GetTextExtent(hDC, lpString, nCount)

Funkcia vypočíta šírku a výšku textového riadku, na ktorý ukazuje parameter **lpString**. Funkcia **GetTextExtent** používa pre výpočet rozmerov reťazca aktuálne zvolený font. Šírka a výška sú počítané bez uvažovania aktuálnych orezovacích oblastí.

Parametre:
hDC - HDC Identifikuje kontext zariadenia.
lpString - LPSTR Ukazuje na textový reťazec.
nCount - int Špecifikuje počet znakov v textovom reťazci.

Vrátená hodnota: Špecifikuje rozmery reťazca. Výška je vo vyššom slove, šírka je v nižšom slove.

Poznámka: Funkcia **GetTextExtent** vo Visual C++ je nahradená funkciou **GetTextExtentPoint32**.

Pozri tiež: GetTextFace SetTextJustification TextOut

12.1.9.2 GetTextFace

Syntax: int GetTextFace(hDC, nCount, lpFacename)

Funkcia kopíruje názov typu písma vybraného fontu do vyrovnávacej pamäti, na ktorú ukazuje parameter lpFacename. Meno typu písma je zkopírované ako nulou zakončený reťazec. Parameter nCount špecifikuje max. počet znakov, ktorý sa má kopírovať.

Parametre:
hDC - HDC Identifikuje kontext zariadenia
nCount - int Špecifikuje veľkosť vyr. pamäti v bajtoch.
lpFacename - LPSTR Ukazuje na vyr. pamäť, ktorá dostane názov typu písma.

Vrátená hodnota: špecifikuje skutočný počet bajtov, zkopírovaných do vyrovnávacej pamäti. Ak sa vyskytla chyba, vráti sa nula.

Pozri tiež: GetTextExtent SetTextJustification TextOut

12.1.9.3 SetTextJustification

Syntax: int SetTextJustification(hDC, nBreakExtra, nBreakCount)

Funkcia pripravuje GDI na zarovnávanie textu použitím zarovnávacích parametrov, ktoré sú určené parametrom **nBreakExtra** a **nBreakCount**. Na zarovnávanie textu pridáva GDI zvláštne obrazkové body medzi oddeľovacie znaky v textovom riadku písanom funkciou **TextOut**.

Funkcia **SetTextJustification** pripravuje zarovnávanie definovaním veľkosti medzery, ktorá sa má pridať. Parameter **nBreakExtra** určuje celkovú veľkosť medzery, ktorá sa pridá k riadku. Parameter **nBreakCount** určuje, koľko oddeľovacích znakov je na riadku.

S funkciou **SetTextJustification** sa vždy používa aj funkcia **GetTextExtent**. Funkcia **GetTextExtent** spočíta šírku daného riadka pred zarovnávaním. Táto šírka musí byť známa pred tým, než sa určí hodnota **nBreakExtra**.

Funkciu **SetTextJustification** je možné použiť aj na zarovnávanie riadkov, ktoré obsahujú viacej častí v rôznych fontoch.

Parametre:
hDC - HDC identifikuje kontext zariadenia.
nBreakExtra - int Určuje celkovú zvláštnu medzeru, ktorá sa pridáva k textovému riadku.
nBreakCount - int Určuje počet oddeľovacích znakov na riadku.

Vrátená hodnota: Ak funkcia prebehla úspešne, vráti 1.

Pozri tiež: GetTextExtent GetTextFace TextOut

12.1.9.4 TextOut

Syntax: BOOL TextOut(hDC, X, Y, lpString, cCount)

Funkcia zapisuje znakový reťazec na určenej obrazovke s použitím aktuálne zvoleného textu.

Parametre: hDC - HDC Identifikuje kontext zariadenia.
 X - int Určuje log. súradnicu x poč. bodu reťazca.
 Y - int Určuje log. súradnicu y poč. bodu reťazca.
 lpString - LPSTR Ukazuje na reťazec znakov, ktoré sa majú nakresliť.
 nCount - int Určuje počet znakov v reťazci.

Vrátená hodnota: Je nenulová, ak sa reťazec nakreslil. Inak je nulová.

Pozri tiež: GetTextExtent GetTextFace SetTextJustification

12.1.9.5 Ďalšie funkcie pre prácu s textom

ExtTextOut - píše reťazec znakov do danej pravouhlovej oblasti. Táto oblasť môže byť matná a môže to tiež byť orezávaná oblasť.

GetTabbedTextExtent - vypočíta šírku a výšku riadku textu, ktorý obsahuje tabulátory.

GetTextAlign - vracia masku príznaku usporiadania textu.

GetTextMetrics - vyrovnávaciu pamäť vyplní metrickými informáciami o vybranom fonte.

SetTextAlign - umiestni reťazec textu na display alebo na zariadenie.

TabbedTextOut - píše reťazec znakov so znakmi tabulátora, nahradenými znakom medzera, v akt. fonte.

12.1.10 Fonty (Trieda=Ifonty)

Funkcie fontu vyberajú, vytvárajú a odstraňujú fonty a zisťujú informácie o nich. Font je podmnožina príslušného typu písma, čo je sada znakov, ktoré majú podobný vzhľad.

GDI obsahuje množstvo fontov. Niektoré zariadenia obsahujú navyše tzv. hardwarové fonty. GDI dovoľuje vyberať fonty tak, že aplikácia najprv popíše charakteristiky fontu, ktoré vyžaduje a potom si vyberie font skutočne existujúci, ktorý tomuto popisu najviac vyhovuje. Font, ktorý aplikácia popíše, je logický font. Proce s výberu fyzického fontu, ktorý najviac odpovedá logickému fontu sa nazýva mapovanie fontu.

12.1.10.1 CreateFont

Syntax: HFONT CreateFont(nHeight, nWidth, nEscapement, nOrientation, fnWeight, fbItalic, fbUnderline, fbStrikeOut, fbCharSet, fbOutputPrecision, fbClipPrecision, fbQuality, fbPitchAndFamily, lpszFace)

Táto funkcia vytvorí log. font s určenými vlastnosťami. Logický font môže byť následne zvolený ako font pre ktorékoľvek zariadenie.

Parametre: nHeight - int Určuje žiadanú výšku (v log. jednotkách) fontu.
 nWidth - int Určuje priemernú šírku (v log. jednotkách) znakov fontu.
 nEscapement - int Určuje uhol každého riadku textu napísaného vo fonte.
 nOrientation - int Určuje uhol (v desatinách stupňov) základného učiaria.

<code>fnWeight</code>	- <code>int</code> Určuje požadovanú váhu fontu v rozsahu od 0 do 1000
<code>fbItalic</code>	- <code>BYTE</code> Určuje, či je font kurzíva.
<code>fbUnderline</code>	- <code>BYTE</code> Určuje, či je font podčiarknutý.
<code>fbStrikeOut</code>	- <code>BYTE</code> Určuje, či je font preškrtnutý.
<code>fbCharSet</code>	- <code>BYTE</code> Určuje požadovanú znakovú sadu.
<code>fbOutputPrecision</code>	- <code>BYTE</code> Určuje požadovanú úresnosť výstupu.
<code>fbClipPrecision</code>	- <code>BYTE</code> Určuje presnosť orezávania.
<code>fbQuality</code>	- <code>BYTE</code> Určuje požadovanú kvalitu výstupu.
<code>fbPitchAndFamily</code>	- <code>BYTE</code> Určuje vzdialenosť medzi dvoma znakmi.
<code>lpszFace</code>	- <code>LPSTR</code> Ukazuje na reťazec s nulovým ukončovacím znakom.

Vrátená hodnota: Ak funkcia prebehla úspešne, identifikuje log. font. Inak `NULL`.

Pozri tiež: `EnumFonts` `GetCharWidth` `GetDC` `ReleaseDC` `TextOut` `SetTextColor`

12.1.10.2 EnumFonts

Syntax: `int EnumFonts(hDC, lpFacename, lpFontFunc, lpData)`

Funkcia vymenuje fonty, ktoré sú dostupné na danom zariadení. Pre každý font, ktorý má názov typu určený parametrom **lpFacename**, funkcia **EnumFonts** zistí informácie o tomto fonte a prenesie ich do funkcie, na ktorú ukazuje parameter **lpFontFunc**. Vymenovávanie je ukončené, pokiaľ už neexistujú žiadne fonty.

Parametre:	<code>hDC</code>	- <code>HDC</code> Identifikuje kontext zariadení.
	<code>lpFacename</code>	- <code>LPSTR</code> Ukazuje na reťazec, ktorý určuje názov typov požadovaných fontov.
	<code>lpFontFunc</code>	- <code>FARPROC</code> Je adresa inštalácie funkcie pre spätný dotaz.
	<code>lpData</code>	- <code>LPSTR</code> Ukazuje na dáta dodané aplikáciou.

Vrátená hodnota: Určuje poslednú hodnotu vrátenú spätnou funkciou. Zmysel je definovaný užívateľom.

Pozri tiež: `CreateFont` `GetCharWidth` `GetDC` `ReleaseDC` `TextOut` `SetTextColor`

12.1.10.3 GetCharWidth

Syntax: `BOOL GetCharWidth(hDC, wFirstChar, wlastChar, lpBuffer)`

Funkcia zistí šírku jednotlivých znakov v sekvenčnej skupine znakov z aktuálneho fontu. Napr. ak **wFirstChar** identifikuje písmeno **a** a parameter **wlastChar** identifikuje písmeno **wLastChar**, tak funkcia zistí šírku všetkých malých písmen. Funkcia ukladá hodnoty do vyrovnávacej pamäti, na ktorú ukazuje parameter **lpBuffer**.

Parametre:	<code>hDC</code>	- <code>HDC</code> Identifikuje kontext zariadenia.
	<code>wFirstChar</code>	- <code>WORD</code> Určuje prvý znak v sekvenčnej skupine znakov
	<code>wLastChar</code>	- <code>WORD</code> Určuje posledný znak v sekvenčnej skupine znakov
	<code>lpBuffer</code>	- <code>LPINT</code> Ukazuje na vyrovnávaciu pamäť, ktorá prijíma hodnoty široké pre sekvenčnú skupinu znakov v aktuálnom fonte.

Vrátená hodnota: Je nenulová, ak funkcia prebehla úspešne.

Pozri tiež: `CreateFont` `EnumFonts` `GetDC` `ReleaseDC` `TextOut` `SetTextColor`

12.2 Vytvorenie aplikácie pomocou WinAPI funkcií.

V menu MS DEV C++ sa zvolí **FILE ⇒ NEW**.

V okne New je potrebné vybrať voľbu **Project ⇒ Win32 Application**. Vyplní sa názov projektu a nastaví sa vhodná cesta na disku. Keď máme vytvorený prázdny projekt, potrebujeme pridať doňho príslušné zdrojové súbory.

Z menu sa vyberie voľba **Project ⇒ Add To Project ⇒ New**. Dostaneme sa opäť do okna New, ale tentokrát si vyberieme **Files ⇒ C++ Source File**. Vyplní sa názov súboru (napr. okno), cesta by mala byť nastavená do nášho projektu.

Už stačí len napísať príslušný kód.

Okno .cpp

```
#include <windows.h>

long FAR PASCAL WndProc(HWND hwnd,           // smerník na okno
                          UINT message,       // naplnená štruktúra správy
                          UINT wParam,        // ďalšie parametre
                          LONG lParam)

{
    HDC      hdc; // smerník na kontext zariadenia, na GDI objekt {pero}
    HPEN      hpen, hpenOld; // smerník na GDI objekt pera
    PAINTSTRUCT ps; // štruktúra obsahuje inf. pre kreslenie do oblasti okna
    RECT      rect; // štruktúra obdĺžnika

    switch (message) {
        case WM_PAINT: // obsluha požiadavky na prekreslenie ok

            hdc = BeginPaint ( hwnd, &ps); // zisti smerník na kontext zariadenia
            GetClientRect ( hwnd, &rect); // zisti súradnice oblasti okna
            hpen = CreatePen(PS_SOLID, 6, RGB(0,0, 255)); // vytvor GDI objekt pero,
            penOld = (HPEN) SelectObject (hdc, hpen); // pridaj vytvorene pero ku
                // kontextu a uschovaj vrátený smerník starého pera
                // pomocou pera nakresli do kontextu obdĺžnik
            Rectangle (hdc, rect.left + 10, rect.top + 10,
                rect.right - 10, rect.bottom - 10); // do stredu okna napíš text
            DrawText (hdc, "Toto je moje oramované okienko",-1,
                &rect, DT_SINGLELINE | DT_CENTER | DT_VCENTER);
            SelectObject (hdc, hpenOld); // vráť kontextu staré pero
            DeleteObject(hpen); // zmaž nepotrebný GDI objekt pera
            EndPaint (hwnd, &ps); // ukončenie kreslenia do kontextu
            return 0;
        case WM_DESTROY: //požiadavka na ukončenie aplikácie, zatvorenie okna
            PostQuitMessage(0);
            return 0;
        }
    }
    return DefWindowProc(hwnd, message, wParam, lParam);
}
```

Aplikácia pomocou tejto funkcie vykreslí modrý obdĺžnik plnou čiarou a vypíše na stred okna „Toto je moje orámované okienko“.

```

//*****
//** Hlavná funkcia ,ktorá sa spúšťa ako prvá pri každej aplikácii **//
//*****

int PASCAL WinMain (HINSTANCE hInstance, // smerník na inštanciu spustenej aplikácie
                   HINSTANCE hPrevInstance, // smerník na predchádzajúcu aplikáciu
                   LPSTR lpszCmdParam, // smerník na reťazec z príkazového riadku
                   Int nCmdShow ) // množina konštánt oddelených symbolom „|“

{
    static char szAppName[] = "Moje okienko"; // reťazec
    HWND hwnd; // smerník na okno
    MSG msg; // štruktúra správ
    WNDCLASS wndclass; // štruktúra triedy okna

    // !!!!! trieda okna nie je to iste ako trieda v C++ !!!!!

    //zistíme či nie je spustená iná inštancia našej aplikácie, ak nie
    zaregistrujeme novu triedu okna.//
    if (!hPrevInstance) {
        wndclass.style = CS_HREDRAW | CS_VREDRAW; //špecifikujeme štýl okna
        wndclass.lpfnWndProc = WndProc; //meno obslužnej procedúry okna
        wndclass.cbClsExtra = 0;
        wndclass.cbWndExtra = 0;
        wndclass.hInstance = (HINSTANCE)hInstance; //smerník inštalácie
        wndclass.hIcon = LoadIcon(NULL, IDI_APPLICATION); //nahratie ikony
        wndclass.hCursor = LoadCursor(NULL, IDC_ARROW); // nahratie kurzora
        wndclass.hbrBackground = GetStockObject(WHITE_BRUSH); // farba pozadia
        wndclass.lpszMenuName = NULL;
        wndclass.lpszClassName = szAppName; // meno triedy okna

        RegisterClass( &wndclass); // zaregistrovanie novej triedy okna
    }

    // vytvoríme nové okno
    hwnd = CreateWindow(szAppName, // meno triedy okna
                       "Program moje okienko", // meno okna, vypisuje v modrom pruhu okna
                       WS_OVERLAPPEDWINDOW, // štýl okna
                       CW_USEDEFAULT, // začiatková pozícia X
                       CW_USEDEFAULT, // začiatková pozícia Y
                       CW_USEDEFAULT, // začiatková veľkosť X
                       CW_USEDEFAULT, // začiatková veľkosť Y
                       NULL, // smerník na rodiča
                       NULL, // smerník na menu
                       (HINSTANCE)hInstance, // SMERNÍK NA INSTANCIU PROGRAMU
                       NULL); // ďalšie parametre programu

    ShowWindow( hwnd,nCmdShow); // zobrazenie okna
    UpdateWindow(hwnd);

    // teraz spúšťame hlavnú slučku správ, tento cyklus skončí až keď funkcia
    WinMain dostane správu WM_QUIT

    while( GetMessage ( &msg,NULL, 0, 0)) {
        TranslateMessage( &msg);
        DispatchMessage ( &msg);
    }
    return msg.wParam;
}

```

12.3 Architektúra MFC aplikácie

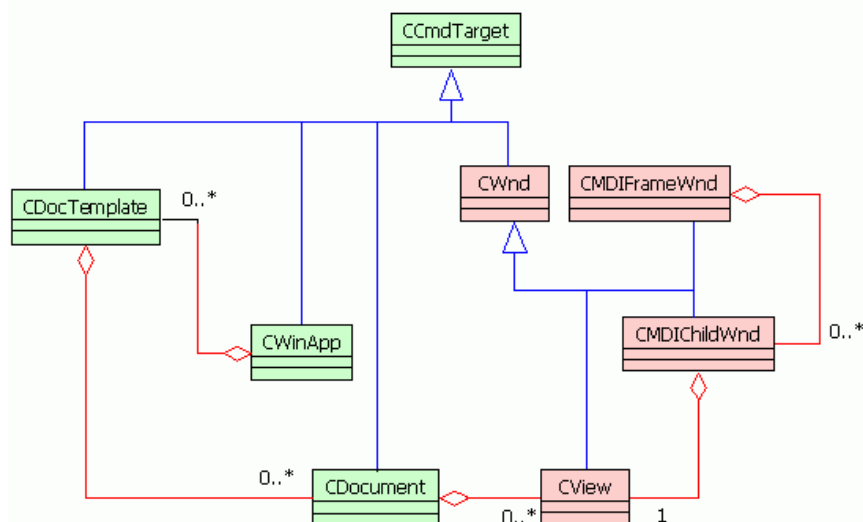
Veľmi užitočnou vlastnosťou MFC aplikácií je ich pomerne pevne daná architektúra, teda vzťahy medzi inštanciami istých tried. Ideálnym príkladom na preskúmanie je štandardne vygenerovaná MDI alebo SDI aplikácia. Ak necháme vygenerovať MDI aplikáciu s názvom *Test*, tak získame nasledujúce triedy.

Vygenerovaná trieda	Predok	Popis
CTestApp	CWinApp	Aplikácia
CTestDoc	CDocument	Dokument aplikácie - obsahuje údaje.
CTestView	CView	Pohľad nad dokumentom - v tejto triede sa vykonáva samotné vykresľovanie.
CMainFrame	CMDIFrameWnd	Hlavný rámec aplikácie.
CChildFrame	CMDIChildWnd	Rámec okolo detských okien.
CAboutDlg	CDialog	Dialóg o aplikácii.

Tab. 5 Tabuľka tried príkladovej MDI aplikácie *Test*

Pri popise architektúry sa ešte stretneme s triedou *CMultiDocTemplate*, prípadne *CDocTemplate*, ktoré predstavujú šablóny dokumentu v MDI a v SDI aplikáciách. Tieto triedy sa väčšinou používajú tak, ako sú, a preto sa z nich nedeďí. (nenachádzajú sa v hornej tabuľke.) Naopak v tabuľke je jeden riadok navyše, ktorým je trieda *CAboutDlg*, ktorá predstavuje dialóg *O aplikácii*, a ktorý je z hľadiska architektúry aplikácie bezvýznamný a ďalej sa s ním nebudeme zaoberať.

Na nasledujúcom obrázku je zjednodušený UML návrh architektúry aplikácie. Ak vám pojem UML nič nehovorí, tak obdĺžnik je trieda, prázdna šípka predstavuje dedičnosť (modrá), kosoštvorec kompozíciu (červená), pri ktorej je uvedený počet inštancií (hviezdička = n)



Obr. 116 UML návrh architektúry MFC aplikácie

Ako vidno na predchádzajúcom obrázku, niektoré triedy dedia z triedy *CWnd*, čo znamená, že sa jedná o okná (vyznačené červenou farbou), zatiaľ čo iné triedy dedia z *CcmdTarget* a nejedná sa o okná - z hľadiska aplikácie sa jedná o neviditeľné triedy (vyznačené zelenou farbou). Všetci potomkovia triedy *CcmdTarget* môžu reagovať na príkazy z menu, aj keď sa nejedná o okná. To je prvá významná vlastnosť MFC architektúry.

Ďalšia vec je kompozícia, ktorá stojí za pozornosť. Kompozícia je vzťah, keď jedna inštancia spravuje alebo vlastní inú inštanciu. Z obrázku je patrné, že aplikácia obsahuje zoznam šablón dokumentov, kde každá šablóna dokumentu obsahuje zoznam dokumentov, kde každý dokument obsahuje zoznam pohľadov (views).

Ku pohľadu existuje aj iná cesta, kde hlavný rámec (*CMDIFrameWnd*) obsahuje zoznam detských rámcov (*CMDIChildWnd*) a každý z nich typicky obsahuje práve jednu inštanciu pohľadu. Tým je možné sa dostať ku každému pohľadu dvomi cestami (zelenou alebo červenou).

MFC v skutočnosti často iba obaluje API, čo v prípade okien znamená, že okolo každého okna (*HWND*) je vytvorená inštancia nejakej triedy odvodennej z *CWnd*. API pozná pojem rodič a potomok, čo znamená, že nejaké okno vlastní iné okná - teda taká kompozícia v API (v UML návrhu nie je tato skutočnosť uvedená, iba obyčajná C++ kompozícia). Tým je teda daná "červená" cesta. Na druhej strane máme v C++ možnosť vytvárať zoznamy smerníkov na inštancie, v našom prípade na inštancie okien - a to je "zelená" cesta. Obe cesty majú svoj význam.

12.3.1 CWnd a HWND

Trieda *CWnd* predstavuje okno v MFC aplikácii a ako členskú premennú obsahuje handle (smerník) skutočného okna. V takomto vzťahu hovoríme o triede ako o obálke, ktorá obaluje API prvok. Vďaka tomu, že môžeme pracovať s C++ inštanciou, máme možnosť využívať objektové programovanie, ale nič nám nebráni pracovať priamo s API pomocou handle (smerníka) okna.

Keď vytvárame nové okno, musíme to urobiť v dvoch krokoch, ktorých poradie je pevne dané:

1. Najprv musíme vytvoriť C++ obálku, teda inštanciu triedy *CWnd* alebo jej potomka.
2. V druhom kroku potom pre túto inštanciu vytvoríme skutočné Windows okno, ktoré je určené svojím handle.

12.3.2 Aplikácia - CWinApp

Hneď na úvod si uvedme, že na aplikácii sú najdôležitejšie dve veci:

- jedná o singleton,
- obsahuje slučku správ.

12.3.2.1 Aplikácia ako singleton

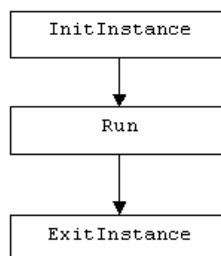
To, že trieda aplikácie je singleton znamená, že existuje **práve** jedna inštancia tejto triedy v aplikácii. V našej vygenerovanej aplikácii *Test*, nájdeme v zdrojovom kóde nasledujúci riadok:

```
////////////////////////////////////
// Jeden a len CTestApp object
CTestApp theApp;
```

Tu sa vytvára spomínaná inštancia a jedná sa o jedinú globálnu premennú v programe. Globálna premenná je prístupná z akejkoľvek inej inštancie alebo funkcie. Ak sa pozrieme na triedu aplikácie údajovo, môžu v nej byť uložené všetky globálne údaje, pretože sú zovšadiaľ prístupné.

12.3.2.2 Životný cyklus aplikácie

V okamžiku, keď spustíme program vo Windows, je vyvolaná funkcia *WinMain*, ktorá je štandardným vstupným bodom. Po skončení funkcie *WinMain*, je ukončený aj program. V MFC táto funkcia volá metódy triedy aplikácie (smerník na inštanciu získava volaním *AfxGetApp*), a to podľa nasledujúcej zjednodušenej schémy.



Obr. 117 Zjednodušená schéma životného cyklu MFC aplikácie

Najprv je vyvolaná metóda *InitInstance*. Tato metóda je vždy v našej triede prekrytá a okrem iného, vytvára hlavné okno aplikácie, prípadne otvára dokument a podobne. Ďalej sa volá metóda *Run*, ktorá obsahuje slučku správ. Metóda *Run* sa väčšinou v potomkoch neprekrýva, pretože jej chovanie je väčšinou dostačujúce. Nakoniec je zavolaná metóda *ExitInstance*, ktorá by mala uvoľniť zdroje, ktoré boli alokované v *InitInstance*. Metóda sa prekrýva iba v prípade, že sme nejaké takéto zdroje alokovali, čo štandardne neplatí.

12.3.2.3 Aplikácia a slučka správ

Každý program vo Windows musí mať slučku správ, čo je kód, ktorý vyberá správy z frontu správ a odovzdáva ich pocietru okna na spracovanie (správy sú do frontu vkladané systémom zakaždým, keď dôjde nad oknom programu k nejakej udalosti, napríklad ku stlačeniu tlačidla myši atď.) Pokiaľ by sme programovali v API, tak by sme museli slučku správ uvádzať znovu a znovu v každom programe. Nie však ale s MFC. V MFC je slučka správ implementovaná v metóde *Run* (viď vyššie). Metóda teda neskončí, pokiaľ nie je zaradená do frontu správ správa *WM_QUIT*, ktorá je poslaná po zatvorení hlavného okna aplikácie.

12.3.3 Dokument - CDocument

Ak by sme chceli popísať dokument niekoľkými slovami, mohli by sme povedať, že dokument sú údaje aplikácie. Táto trieda zabezpečuje perzistenciu údajov.

12.3.3.1 Perzistencia údajov

V MFC aplikácii sa o perzistenciu údajov stará práve dokument, teda inštancia triedy *CDocument* alebo jej potomkov. Pokiaľ si chceme predstaviť dokument, môžeme ho vidieť ako pamäťový obraz diskových údajov, teda nejakého súboru. Vždy, keď sa používateľ pokúsi otvoriť súbor alebo uložiť údaje do súboru, vyvolá sa virtuálna metóda *Serialize*, v ktorej zaistíme načítanie alebo uloženie údajov. Pozrime sa ako vyzerá štandardne vygenerovaná metóda

```

void TestDoc::Serialize(CArchive& ar)
{
    if (ar.IsStoring())
    {
        // vložiť kód na uloženie
    }
    else
    {
        // vložiť kód na načítanie
    }
}

```

Ako parameter dostaneme referenciu na inštanciu triedy *CArchive*, ktorá predstavuje archív, do ktorého uložíme naše údaje, alebo ich z neho načítame. Žiadne trápenie s menom súboru, právami, atď. Našou úlohou bolo zabezpečiť uloženie hodnôt a nie práca so súborom. O samotný súbor sa postarajú metódy dokumentu, konkrétne triedy *CDocument*, jediné čo je potrebné urobiť, je prekryť virtuálnu metódu *Serialize*.

12.3.4 CCmdTarget

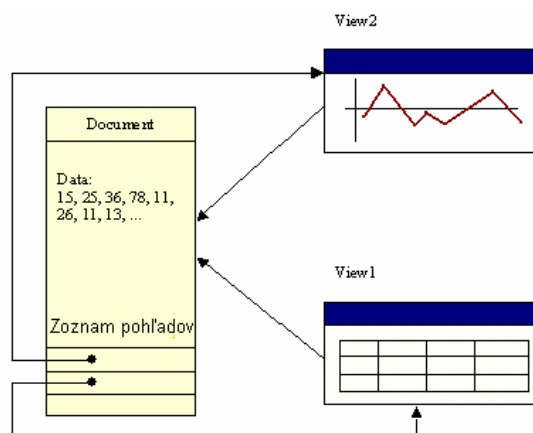
Trieda *CCmdTarget* je rozhraním, ktoré umožňuje potomkom reagovať na udalosti z menu. MFC distribuuje príkaz z menu cez celú aplikáciu, presnejšie medzi potomkov *CCmdTarget* a hľadá niekoho, kto sa príkazu ujme a spracuje ho.

Významné je, že týmto príjemcom nemusí nutne byť okno, ale ktokoľvek, kto dedí z *CCmdTarget*, teda aj aplikácia a dokument. Tým je umožnené aj "neviditeľným" objektom, aby reagovali na správy z menu.

V prípade dokumentu sa teda jedná o výnimku, keď sa porušuje pravidlo o tom, že používateľ nemôže s dokumentom komunikovať priamo, ale vždy cez nejaký *pohľad* (okno). Táto vlastnosť je v MFC kvôli tomu, aby sa niektoré akcie zjednodušili. Pokiaľ máme v menu položku *Vymaž dokument*, tak je asi jasné, že bude vhodné, aby bol tento povel spracovaný priamo dokumentom.

12.3.5 CView - pohľad

Trieda *CView* a jej potomkovia predstavujú používateľské rozhranie nad údajmi - dokumentom. V nasledujúcom texte budeme o *view* hovoriť ako o *pohľade*. *Pohľad* sa dá prirovnať ku okuliarom, cez ktoré sa pozeráme na dokument. *Pohľad* typicky neobsahuje údaje. Cez rôzne okuliare vidíme tie isté údaje rôzne.

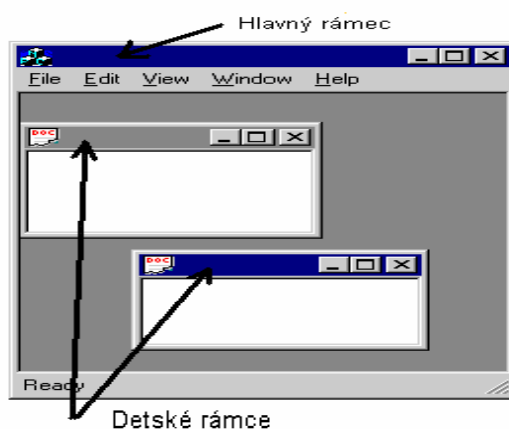


Obr. 118 Zjednodušená architektúra Dokument/Pohľad

Na Obr. 118 je vidieť, že súbor čísel je možné reprezentovať ako graf alebo ako tabuľku. Tým, že sme oddelili údaje (dokument) od ich zobrazenia (pohľad), získali sme možnosť pridávať ďalšie rôzne pohľady, podľa potreby. Takýto vzťah medzi pohľadom a dokumentom sa označuje ako architektúra dokument/pohľad (document/view architecture).

12.3.6 Hlavný rámec MDI aplikácie

V MDI aplikácii máme hlavné okno, ktoré obsahuje ďalšie detské okná. MDI skratka predstavuje *Multiple Document Interface*, čo by malo znamenať, že pomocou jednej aplikácie môžeme upravovať viacero dokumentov naraz. V mnohých prípadoch sa ale MDI aplikácia používa pre editovanie jedného dokumentu s tým, že sa používa viacero druhov okien. Typickou ukážkou MDI aplikácie je MS-Word.



Obr. 119 Príklad MDI aplikácie

Z hľadiska architektúry aplikácie je dôležité, že hlavný rámec je predstavovaný triedou *CMDIFrameWnd* a spravuje zoznam detských okien. Tento zoznam nie je reprezentovaný nejakým C++ zoznamom a členskou premennou, ale jedná sa o internú záležitosť Windows.

Pri pohľade do vygenerovanej aplikácie zistíme, že žiadna trieda *CMDIFrameWnd* tu neexistuje. Je to kvôli tomu, že prostredie pre nás rovno vygenerovalo potomka, ktorý sa volá *CMainFrame*.

Hlavný rámec je podobne ako aplikácia *singleton*, čo znamená, že v dobe behu programu existuje práve jedna inštancia. K tejto inštancii potom majú prístup všetky ďalšie triedy pomocou globálnej funkcie *AfxGetMainWnd*. Hlavný rámec alebo tiež hlavné okno aplikácie je prístupné zovšadiaľ.

12.3.6.1 Detský MDI rámec

Oveľa zaujímavejšie sú detské rámce, ktoré sú tvorené inštanciami triedy *CMDIChildWnd*. Vo vygenerovanej aplikácii opäť túto triedu nájdeme ako predka, tentoraz z nej dedí trieda *CChildFrame*.

Veľmi dôležitou vlastnosťou detských rámcov je skutočnosť, že sa veľmi často nepoužívajú tak ako sú, ale sa do nich vložia ďalšie detské (podriadené) okná. Týmito oknami sú väčšinou pohľady.

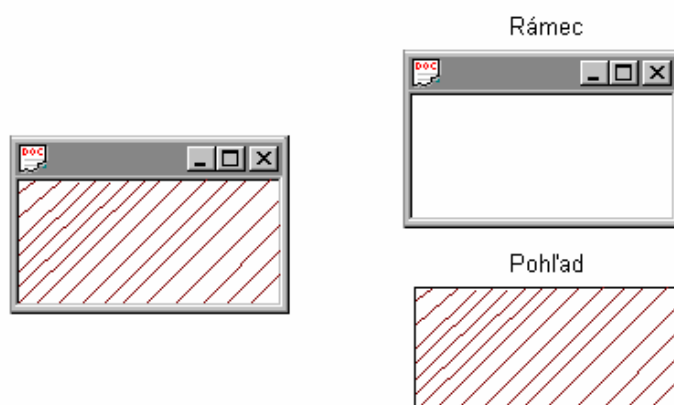
12.3.7 Hlavný rámec SDI aplikácie

Posledným druhom rámca je hlavný rámec SDI aplikácie a môžeme sa naň pozerat' ako na akýsi hybrid medzi hlavným rámcom MDI aplikácie a detským rámcom MDI aplikácie. Typickým príkladom SDI aplikácie je Poznámkový blok (Notepad) z Windows. Rovnako ako v prípade hlavného rámca v MDI aplikácii ide o singleton, ale rovnako ako detský MDI rámec priamo obsahuje ďalšie okno - pohľad.

Hlavný SDI rámec je reprezentovaný triedou *CFrameWnd* a vo vygenerovanej SDI aplikácii by sme ho našli ako predka triedy *CMainFrame* (zhoda mena triedy s MDI aplikáciou nič neznamená).

12.3.7.1 Pohľady v rámcoch

Typicky sa do rámcov (hlavný SDI, alebo detský) vkladá *pohľad* (view). *Pohľad* sa chová tak, že neustále zakrýva klientskú plochu okna rámca a automaticky prispôbuje svoju veľkosť v prípade zmeny veľkosti okna rámca tak, aby tato podmienka bola stále splnená. *Pohľad* je ďalej nastavený tak, že sa jedná o okno, ktoré nemá záhlavie, tlačidlá, menu ani rámec. (viď Obr. 120)



Obr. 120 Zloženie okna: trieda rámca a trieda pohľadu

Z toho vyplýva, že pohľad je vložený do nejakého rámca, na ktorý sa môže ľahko odvolať pomocou metódy *GetParentFrame*. Rámec ale môže obsahovať viacero pohľadov, a preto nie je nájdenie pohľadu také triviálne. Vzťah medzi pohľadom a rámcom je daný vzťahom podriadené/nadriadené okno z Windows a teda nie je predstavovaný nejakými členskými premennými, ale jedná sa o internú záležitosť Windows.

12.3.7.2 Šablóna dokumentu

Poslednou triedou, ktorá sa vyskytuje v architektúre MFC aplikácie je šablóna dokumentu. Pozor, jej názov je mierne zavádzajúci, pretože sa nejedná o C++ šablónu, ale o bežnú C++ triedu.

Šablóna dokumentu slúži na správu dokumentov, čo znamená, že nové dokumenty sa vytvárajú práve pomocou tejto šablóny, ktorá si drží zoznam existujúcich dokumentov, atď. Ako je vidno, tak sa jedná o pomerne dôležitú triedu, ktorá býva často prehliadaná, a to najskôr preto, že sa z nej bežne nededí, ale používa sa viac-menej automaticky.

Samotné šablóny sú spravované aplikáciou. Aplikácia teda neobsahuje priamo dokumenty, ale obsahuje šablóny, ktoré ďalej obsahujú dokumenty.

Zjednodušene sa dá povedať, že každá šablóna je spojená s nejakou súborovou príponou. Pri otvorení dokumentu aplikácie, sa prehládajú šablóny dokumentov a zistí sa, ktorá z nich pracuje so súborom, identifikovaným danou príponou. Tým je jednoducho a automaticky zabezpečené, že aplikácia môže pracovať s viacerými druhmi dokumentov.

Súčasťou šablóny dokumentu je aj väzba medzi *rámcom* a *pohľadom*. Šablóna dokumentu obsahuje vzor (preto tá šablóna v názve), podľa ktorého je možné vytvoriť nový dokument, *rámec* a *pohľad*, vložený do rámca. Taktiež sa pri vytváraní dokumentu automaticky zabezpečí väzba medzi dokumentom a pohľadom.

Ukážku kódu štandardne vygenerovanej MDI aplikácie, kde sa vytvára nová šablóna dokumentu:

```
// Vytvorenie novej šablóny
// -----
CMultiDocTemplate* pDocTemplate;
pDocTemplate = new CMultiDocTemplate(
    IDR_PRO_COTYPE,                // ID resource
    RUNTIME_CLASS(PRO_CommonMDIDoc), // Trieda dokumentu
    RUNTIME_CLASS(CChildFrame),    // Trieda rámca
    RUNTIME_CLASS(PRO_CommonMDIView)); // Trieda pohľadu

// Pridanie novo vytvorenej šablóny do aplikácie
AddDocTemplate(pDocTemplate);
```

MDI aplikácie používajú pre šablóny dokumentu triedu *CMultiDocTemplate* a SDI používajú *CSingleDocTemplate*. Obe majú spoločného predka, ktorým je trieda *CDocTemplate*.

Konštruktor očakáva štyri parametre. Prvý preskočíme a budeme sa venovať ďalším trom. Na týchto parametroch je zvláštna jedna vec - jedná sa o triedy. Z princípu C++ nemôže byť parametrom metódy trieda, pretože tie počas prekladu zanikajú. MFC toto obchádza tak, že vytvára špeciálne štruktúry a premenné, ktoré akoby predstavovali triedy. Odkaz na tieto špeciálne údaje sa potom vykonáva napríklad pomocou komplikovaného makra *RUNTIME_CLASS*. Pokiaľ sa vrátíme k parametrom a prijmeme skutočnosť, že sú to triedy, zistíme, že sa jedná o triedy, ktoré tvoria dokument, rámec a pohľad. Šablóna potom na základe odovzdaných parametrov dokáže dynamicky vytvoriť inštancie týchto tried.

Prvým parametrom konštruktoru šablóny je odkaz do zdrojov (*resource*) aplikácie a odkazuje sa do tabuľky reťazcov. Pod uvedeným identifikátorom musí byť zvláštny reťazec, ktorý popisuje ďalšie vlastnosti dokumentu. Tento reťazec je tvorený niekoľkými časťami, kde každá z nich je oddelená pomocou "\n".

Názov parametru	Popis parametru
windowTitle	Titulok hlavného okna. Používa sa iba v SDI aplikáciách.
docName	Meno dokumentu. Toto meno spoločne s poradovým číslom sa objaví v rámci MDI pohľadu. Taktiež sa s ním stretneme pri uložení dokumentu.
fileNewName	Meno šablóny dokumentov. Dôležité pokiaľ používame viacero šablón dokumentov a chceme vytvoriť nový dokument. MFC zabezpečí, že sa zobrazí dialóg, v ktorom máme vybrať typ dokumentu, aký chceme vytvoriť.
filterName	Názov filtra. Pri otváraní/uložení dokumentov sa v dialógu pre výber súboru objaví ako filter. Napríklad: "Moje súbory (*.dat)"
filterExt	Prípona, podľa ktorej sa vykonáva filtrovanie. Napr. ".dat"
regFileTypeId	Pod týmto identifikátorom je uložená väzba na dokument v databáze registrov .
regFileName	Meno, ktoré popisuje väzbu.

Tab. 6 Tabuľka parametrov konštruktora šablóny

Príklad z vygenerovanej aplikácie môže vyzerat' takto:

```
WinTitle\nDocName\nFileNewName\nMoje soubory (*.moje)\n.moje
\nCommonMDI.Document\nPRO_Co Document
```

12.4 Vytvorenie projektu pomocou MFC AppWizardu.

V menu MS DEV C++ sa zvolí **FILE** ⇒ **NEW**.

Vyberie sa v okne **New** voľba **Project** ⇒ **MFC AppWizard (exe)**. Vyplní sa názov projektu a nastaví sa vhodná cesta na disku. Nasleduje 6 krokov na vyplnenie parametrov MFC projektu:

- V 1.kroku si zvolíme **single document**.
- 2. a 3.krok preskočíme, lebo nie sú pre nás momentálne zaujímavé.
- Vo 4.kroku sa vyberá menu a celkový vzhľad okna.
- V 5.kroku si môžeme pridať do vygenerovaného kódu aj komentáre, samozrejme v anglickom jazyku.
- V 6.kroku nám ponúkne mená tried a súborov, ktoré bude generovať a môžeme ich ešte zmeniť.

Stlačí sa tlačidlo **Finish**, vysvieti sa okno so súhrnnými informáciami o vytváranom projekte a ak sú v poriadku tak ich je potrebné potvrdiť.

Bolo vygenerované 5 tried, 5 zdrojových a 6 hlavičkových súborov, ktorých názvy sú odvodené od názvu projektu. Ak sme projekt nazvali **Skúška** tak by sme mali mať tieto triedy (pozri predchádzajúcu kapitolu):

CAboutDlg, CMainFrame, CSkuskaApp, CSkuskaDoc, CSkuskaView

A tieto súbory:

MainFrm.cpp, Skuska.cpp, SkuskaDoc.cpp, SkuskaView.cpp a Stdafx.cpp

MainFrm.h, Skuska.h, SkuskaDoc.h, SkuskaView.h a Stdafx.h, Resource.h

Ak teraz projekt skompilujeme a zlinkujeme tak dostaneme spustiteľnú aplikáciu.

O to, čo bude zobrazené v okne sa stará trieda View. V našom prípade odvodená trieda CSkuskaView. Nájdeme jej funkciu CSkuskaView::OnDraw a doplníme o tento kód.

```
void CSkuskaView::OnDraw(CDC* pDC)
{
    CSkuskaDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    LOGBRUSH lb;
    lb.lbStyle = BS_SOLID;           // štýl pera "plná čiara"
    lb.lbColor = RGB(255, 0, 0);    // farba pera "červená"
    lb.lbHatch = 0;
    CPen oPero(PS_DASH, 5, &lb); // PS_GEOMETRIC|
    CPen* poPredchPero = pDC->SelectObject(&oPero);
    pDC->MoveTo(150, 100);           // kreslenie čiar
    pDC->LineTo(200, 100);
    pDC->LineTo(300, 300);
    pDC->LineTo(400, 100);
    pDC->LineTo(450, 100);
    pDC->SelectObject(poPredchPero); // vrátime kontextu povodne pero
}
```

Môžeme skompilovať a spustiť. Aplikácia vykreslí plnou čiarou červené lomené V.



12.5 Programové úlohy

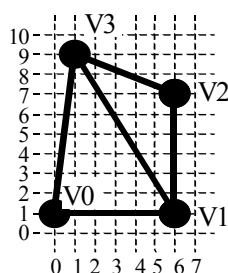


Úloha č. 1	Vo vyššom programovacom jazyku jednoducho realizujte vykreslenie úsečky DDA algoritmom. Vstupné údaje budú súradnice štartovného a koncového bodu úsečky. Chod programu overte pomocou jednoduchých vstupných údajov resp. parametrov.
Úloha č. 2	Vo vyššom programovacom jazyku jednoducho realizujte vykreslenie úsečky Bresenhamovým algoritmom. Vstupné údaje budú súradnice štartovného a koncového bodu úsečky. Chod programu overte pomocou jednoduchých vstupných údajov resp. parametrov.
Úloha č. 3	Vo vyššom programovacom jazyku jednoducho realizujte vykreslenie kružnice a elipsy pomocou algoritmu s predikciou chyby. Vstupné údaje budú súradnice stredu kružnice a polomer resp. súradnice stredu elipsy a jej polosi a,b. Chod programu overte pomocou jednoduchých vstupných údajov resp. parametrov.
Úloha č. 4	Vo vyššom programovacom jazyku jednoducho realizujte transformáciu posunutia v 2D priestore. Vstupom bude 2D teleso a vektor posunutia v 2D. Chod programu overte pomocou jednoduchých vstupných údajov resp. parametrov. Štruktúra telesa vid' poznámka.
Úloha č. 5	Vo vyššom programovacom jazyku jednoducho realizujte transformáciu posunutia v 3D priestore. Vstupom bude 3D teleso a vektor posunutia v 3D. Chod programu overte pomocou jednoduchých vstupných údajov resp. parametrov. Štruktúra telesa vid' poznámka. Na premietanie scény použite axonometriu.
Úloha č. 6	Vo vyššom programovacom jazyku jednoducho realizujte transformáciu zrkadlenia v 2D priestore. Vstupom bude 2D teleso a ťažisko zrkadlenia. Chod programu overte pomocou jednoduchých vstupných údajov resp. parametrov. Štruktúra telesa vid' poznámka.
Úloha č. 7	Vo vyššom programovacom jazyku jednoducho realizujte transformáciu zrkadlenia rastrového objektu v 2D priestore. Vstupom bude jednoduchý rastrový objekt nakreslený na obrazovke a ťažisko zrkadlenia. Chod programu overte pomocou jednoduchých vstupných údajov resp. parametrov.
Úloha č. 8	Vo vyššom programovacom jazyku jednoducho realizujte transformáciu posunutia v 3D priestore. Vstupom bude 3D teleso a ťažisko zrkadlenia. Chod programu overte pomocou jednoduchých vstupných údajov resp. parametrov. Štruktúra telesa vid' poznámka. Na premietanie telies použite axonometriu.
Úloha č. 9	Vo vyššom programovacom jazyku jednoducho realizujte transformáciu zmeny mierky v 2D priestore. Vstupom bude 2D teleso a koeficient zmeny mierky v 2D. Chod programu overte pomocou jednoduchých vstupných údajov resp. parametrov. Štruktúra telesa vid' poznámka.
Úloha č. 10	Vo vyššom programovacom jazyku jednoducho realizujte transformáciu zmeny mierky v 3D priestore. Vstupom bude 3D teleso a koeficient zmeny mierky v 3D. Chod programu overte pomocou jednoduchých vstupných údajov resp. parametrov. Štruktúra telesa vid' poznámka. Na premietanie scény použite axonometriu.
Úloha č. 11	Vo vyššom programovacom jazyku jednoducho realizujte transformáciu zmeny mierky – zväčšenia, rastrového objektu v 2D priestore. Vstupom bude jednoduchý rastrový objekt nakreslený na obrazovke a neceločíselný koeficient zväčšenia. Chod programu overte pomocou jednoduchých vstupných údajov resp. parametrov.
Úloha č. 12	Vo vyššom programovacom jazyku jednoducho realizujte transformáciu skosenia v 2D priestore. Vstupom bude 2D teleso a koeficient skosenia v príslušnej osi. Chod programu overte pomocou jednoduchých vstupných údajov resp. parametrov. Štruktúra telesa vid' poznámka.
Úloha č. 13	Vo vyššom programovacom jazyku jednoducho realizujte transformáciu zmeny mierky v 3D priestore. Vstupom bude 3D teleso a koeficient skosenia v príslušnej osi alebo roviny. Chod programu overte pomocou jednoduchých vstupných údajov resp. parametrov. Štruktúra telesa vid' poznámka. Na premietanie scény použite axonometriu.
Úloha č. 14	Vo vyššom programovacom jazyku jednoducho realizujte transformáciu otočenia v 2D priestore. Vstupom bude 2D teleso a uhol otočenia v stupňoch. Chod programu overte pomocou jednoduchých vstupných údajov resp. parametrov. Štruktúra telesa vid' poznámka.
Úloha č. 15	Vo vyššom programovacom jazyku jednoducho realizujte transformáciu otočenia v 3D priestore. Vstupom bude 3D teleso, uhol otočenia v stupňoch a os, okolo ktorej sa otočenie zrealizuje. Chod programu overte pomocou jednoduchých vstupných údajov resp. parametrov. Štruktúra telesa vid' poznámka. Na premietanie scény použite axonometriu.

Poznámky:

Úloha č. 16	Vo vyššom programovacom jazyku jednoducho realizujte Cohen-Sutherlandov algoritmus orezania. Chod programu overte pomocou jednoduchých vstupných údajov resp. parametrov.
Úloha č. 17	Vo vyššom programovacom jazyku jednoducho realizujte základné typy axonometrického premietania 3D scény. Vstupom bude 3D teleso. Chod programu overte pomocou jednoduchých vstupných údajov resp. parametrov. Štruktúra telesa vid' poznámka.
Úloha č. 18	Vo vyššom programovacom jazyku jednoducho realizujte perspektívne premietanie 3D scény. Vstupom bude 3D teleso. Chod programu overte pomocou jednoduchých vstupných údajov resp. parametrov. Štruktúra telesa vid' poznámka.
Úloha č. 19	Vo vyššom programovacom jazyku jednoducho realizujte vykreslenie Fergusonovej krivky. Vstupom budú štartovný a koncový bod a riadiace vektory v štartovnom a koncovom bode. Chod programu overte pomocou jednoduchých vstupných údajov resp. parametrov.
Úloha č. 20	Vo vyššom programovacom jazyku jednoducho realizujte vykreslenie Beziérovej krivky. Vstupom bude usporiadaná množina štyroch riadiacich bodov. Chod programu overte pomocou jednoduchých vstupných údajov resp. parametrov.
Úloha č. 21	Vo vyššom programovacom jazyku jednoducho realizujte vykreslenie Bilineárnej Coonsovej plochy bez riešenia viditeľnosti. Vstupom bude matica riadiacich bodov. Chod programu overte pomocou jednoduchých vstupných údajov resp. parametrov. Na premietanie scény použite axometriu.
Úloha č. 22	Vo vyššom programovacom jazyku jednoducho realizujte vykreslenie Beziérovej bikubickej plochy bez riešenia viditeľnosti. Vstupom bude matica riadiacich bodov. Chod programu overte pomocou jednoduchých vstupných údajov resp. parametrov. Na premietanie scény použite axometriu.
Úloha č. 23	Vo vyššom programovacom jazyku jednoducho realizujte algoritmus vyplňovania metódou riadkového rozkladu. Vstupom je usporiadaná, uzavretá množina hrán, definujúcich vyplňovanú oblasť a farba výplne. Chod programu overte pomocou jednoduchých vstupných údajov resp. parametrov.
Úloha č. 24	Vo vyššom programovacom jazyku jednoducho realizujte algoritmus inverzného vyplňovania. Vstupom je usporiadaná, uzavretá množina hrán, definujúcich vyplňovanú oblasť a farba výplne. Chod programu overte pomocou jednoduchých vstupných údajov resp. parametrov.
Úloha č. 25	Vo vyššom programovacom jazyku jednoducho realizujte algoritmus semienkového vyplňovania. Vstupom bude vyplňovaná oblasť nakreslená na obrazovke a poloha a farba semienka. Chod programu overte pomocou jednoduchých vstupných údajov resp. parametrov.

Poznámka: štruktúra telesa je daná zoznamom vrcholov (s 2 alebo 3 súradnicami) a maticou susednosti. Prvok matice susednosti $[i,j]$ je rôzny od 0, ak medzi vrcholmi i a j existuje hrana. Príklad:



Zoznam vrcholov:

V0 = [0,1]

V1 = [6,1]

V2 = [6,7]

V3 = [1,9]

Matica susednosti:

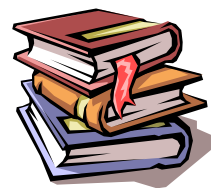
0 1 0 1

1 0 1 1

0 1 0 1

1 1 1 1

13 Odporúčané rozširujúce zdroje



- [1] Sobota B.: Počítačová grafika a jazyk C; KOPP České Budějovice, 1995, pp.278, ISBN 80-85828-52-9
- [2] Sobota, B.- Milián, J.: Grafické formáty, KOPP České Budějovice, 1996, pp.160, ISBN 80-85828-58-8
- [3] Sobota, B. - Milián, J. - Miliánová, L. : Grafické editory; KOPP České Budějovice, České Budějovice 1997, ps. 230, ISBN 80-85828-79-0
- [4] Sobota, B.: Počítačová grafika; skriptum, ELFA Košice, Košice 1997, ps. 188., ISBN 80-88786-57-6
- [5] Žára J. – Beneš B. – Felkel P.: Moderní počítačová grafika; Computer Press Praha, Praha 1998, ps. 448, ISBN 80-7226-049-9
- [6] Sobota, B.: Počítačová grafika - riešenie viditeľnosti; ANIMA Košice, Košice 1999, ps. 48, ISBN 80-968167-6-4

Ďalšie podklady k štúdiu je možné získať pomocou Internetu:

http://hornad.fei.tuke.sk/predmety/pg

14 Register

A		
abakus	9	
AGP	33	
Akimovská interpolácia	86	
alfa-miešanie	146	
alfa-zložka	146	
algoritmus pamäte hĺbky	116	
algoritmus spájajúceho horizontu	111	
algoritmus riadkového rozkladu	118	
alias	51	
antialiasing	46, 51, 52	
ATC	31	
autopanring	77	
avatar	156	
axonometria	79	
kavaliema	79	
technická	79	
vojenská	79	
B		
Bernsteinove polynómy	88	
Bezierova bikubická plocha	95	
Bezierové krivky	88	
nadväzovanie	89	
bikubická Coonsova plocha	93	
bilinéárna Coonsova plocha	93	
BIOS		
EGA (VGA)	31	
bod	45	
atribúty	45	
bokorys	78	
Bresenhamov algoritmus	47, 66	
brightness	141, 142	
BSP	120, 136	
nevýhody	124	
použitie	123	
vlastnosti	121	
výhody	124	
zložitosť	124	
BSP strom	120, 125, 127, 136	
2D	120	
3D	120	
listy	122	
prechod	120, 123	
tvorba	120	
uzly	122	
B-spline krivky	89	
C		
CAD	10, 11, 13, 19	
CGI	11, 37	
clipping	73	
CMY	140, 143	
CMYK	140	
Cohen-Shuterlandov algoritmus	114	
Cohen-Sutherlandov algoritmus	74	
Coonsove plochy	92	
CRT	23, 27	
CRTC	31	
CSG	44	
cuter	23	
cybemaut	156	
D		
DDA	46, 66	
depth-buffer algoritmus	116	
digitizér	19	
dimetria	79	
DIRECTX	33	
displej	23	
distribúcia chyby	149	
Floyd-Steinberg	149	
dithering	41, 147, 148	
DPI	16, 20	
DTP	19	
E		
ekvidistančné zobrazenie	81	
elipsa	49	
kreslenie parametrickým vyjadrením	49	
kreslenie s predikciou chyby	50	
event	37	
F		
farba	138	
farebná hĺbka	28	
farebný model	138	
farebný tón	141	
Fergusonova krivka	85	
flat-shading	132	
floodfill	106	
font	26	
fotoploter	23	
fotorealistické zobrazovanie	134	
fraktál	154	
fraktálna geometria	154	
Freeman-Loutrellov algoritmus	114, 132	
G		
gama-korekcia	146	
gamepad	18	
gamma faktor	146	
GDC	31	
geometrické modelovanie	44	
CSG	44	
drôtový model	44	
hraničná reprezentácia	45	
objemový model	44	
povrchový model	44	
geometrické transformácie	57	
GKS	37	
GKS-3D	37	
grafické zariadenia		
aktívne	15	
inteligentné	16	
neinteligentné	16	
pasívne	15	
rastrové	15	
s dočasným záznamom	15	
s trvalým záznamom	15	
vektorové	14	
vstupné	14, 16	
výstupné	14, 20	
grafický akcelerátor	33	
grafický element	41	
atribúty	42	
grafický objekt		
rastrový	42, 54	
vektorový	42, 54	
grafický procesor	28	
grafický prvok	41	
atribúty	42	
grafický urýchľovač	33	
GUI	36, 37	
H		
halfioning	147, 148, 150	
Highcolor	33	
hlbková spojitosť	117	

HLS	144, 145
HMD	156
HPGL	23
hranica vyplňovanej oblasti	
definovaná geometricky	98, 108
definovaná na zobrazovači	98
hraničná reprezentácia	45
HSB	141, 144
HSV	141
hue	141
HW Registre	
EGA (VGA)	31

C

chaos	154
choice	36

I

ihlan pohľadu	75, 126
inverzné vyplňovanie	101
ISA	33
izometria	79

J

jas	138, 141, 142
joystick	17
Joystick	17, 18
analogový	17
digitálne	17

K

keystick	18
kolmé premietanie	78
krivka	41
analytická	84
aproximačná	84
Beziérová	88
B-spline	89
Fergusonova	85
interaktívne vytváraná	84, 87
interpoláčná	84, 85
spline	87
kruhový oblúk	50
kruhový odsek	50
kruhový výsek	50
kružnica	48
kreslenie parametrickým vyjadrením	49
kreslenie s predikciou chyby	49
k _v -spline	87

L

Lagrangeova interpolácia	85
LCD	25, 27, 154
MIM	26
STN	25
TFT	26
LED	16
lightness	141
locator	36
lokátor	36
lomená čiara	41
lúč svetla	135

M

malať algoritmus	112, 123
medián filter	64, 151
metóda BSP stromov	120
metóda riadkového rozkladu	99
metóda sledovania lúča	134, 135
miešanie	
aditívne	139
alfa	146
subtraktívne	139
mód obrazový	
grafický	26
textový	26

model CMY	140
model HLS	141
model HSB	141
model RGB	139
model UWB	142
monitor	23, 146
elektroluminiscenčný	24
farebný	26
LCD	25
monochromatický	26
plazmový	24
s katódovou obrazovkou	23
multimédiá	13, 155
myš	10, 16

N

nárys	78
názomý priemet	79
neviditeľná hrana	115
NSS	55

O

občerstvenie	73
obrazová pamäť	30
kapacita	28
OCR	19
offset	77
ohraničujúce telesá	136
ohraničujúce útvary	127
okno	72
oktantové stromy	127
oktantový strom	136
OpenGL	33, 39
základné operácie	40
optické pero	20
orezanie	73, 74
kruhových objektov	76
polygónov	75
úsečiek	74
orezávanie	
3D na zomý ihlan	126
ortografické zobrazenie	81
osvetľovací model	136
osvitová jednotka	20
otočenie	57, 67
quaternióny	70
s interpoláciou medziahľých bodov	68
s interpoláciou všetkých bodov	69
v 3D	67
ovládač	17

P

palec	16
panning	77
panorámovanie	73, 77
pri rastrových typoch	77
pri vektorových typoch	77
paralelný port	16
PCI	33
PCL	20, 22
perspektíva	80
dištanca	80
PHIGS	38
pick	36
pixel	45
plát	93, 94
plátovanie	94
plávajúci horizont	111
plocha	41
analytická	92
Beziérová bikubická	95
bikubická Coonsova	93
bilinéarna	93
právková	93
priamková	93
všeobecná Coonsova	94

Poznámky:

ploter	10, 22
analogový	23
digitálny	23
fixový	23
s guľičkovým perom	23
stolový	23
tryskový	23
tušový	23
valcový	23
plotové vyplňovanie	101
počítačová grafika	10
počítačové hry	12
Pohlkeova veta	79
poltónovanie	147, 148, 150
polyline	41, 48
polymarker	41, 45
relácie	45
port	16
PostScript	20, 22
posunutie	57, 59
v 3D	59
potenciál viditeľnosti	129
potenciálne viditeľná hrana	115
pôdorys	78
pracovná plocha	
fyzická	74, 77
logická	74, 77
maximálna	74
virtuálna	74, 77
pravítková plocha	93
premietacie 2D transformácie	72
premietacie 3D transformácie	78
axonometrické	79
kolmé	78
perspektíva	80
premietacie lineárne transformácie	81
prevod	
HLS do RGB	145
HSB do RGB	144
na úroveň šedej	146
RGB a CMY	143
RGB do HLS	144
RGB do HSB	144
priamková plocha	93
priepustnosť zbernice	33
PS/2	16
PVS	129
Q	
quaternióny	70
R	
radiosity	134
raytracing	134, 135
distribúovaný	135
distributívny	135
strom	135
refresh	73
request	37
RGB	139, 143, 144, 145
riadková súvislosť	118
riadkový rozklad	99
nájdienie priesečníkov hrán	100
riešenie viditeľnosti	
grafov funkcií	111
úrychľovacie technológie	125
zložitosť	110
rotácia	67
rovnoploché zobrazenie	81
rozlišovacia schopnosť	16, 28
rozptyľovacia matica	148
rozptyľovanie	43, 147
maticové	148
náhodné	147

rybie oko	81
ekvidištantné zobrazenie	81
ortografické zobrazenie	81
rovnoploché zobrazenie	81

S

sample	37
saturácia	141
s-buffer	118, 124
scan-conversion	99
scan-line algoritmus	118
scanner	19
plošný	19
ručný	19
sektorovanie	128
Sekvencér	31
semienkové vyplňovanie	
nerekurzívne	106
rekurzívne	105
sériový port	16
simulácia	13
skosenie	57, 65
rastového objektu	66
v 3D	65
skrut	94
sled bodov	41, 45
relácie	45
span-buffer	124
Spline krivka	87
spojenie	36
spojkový zoznam	125
sprememenie farieb	64, 152
Sprout-Sutherlandov algoritmus	75
SSC	55
SSO	55
SST	55
SSZ	55
stroke	36
súradnice	
homogénne	58
karteziánske	55
polárne	55
súradnicová sústava	54
2D	54
3D	54
atribúty	55
globálna	55
kamery	55
normalizovaná	37, 55
objektu	55
používateľská	55
rozmer	54
textúry	55
typ súradnic	54
univerzálna	55
zariadenia	55
súradnicový zapisovač	15, 22
susedstvo	
4-susedstvo	151
8-susedstvo	151
Sutherland-Hodgmanov algoritmus	75
svetelný lúč	135
svetlo	138
monochromatické	138
svetlosť	138, 141
sýtosť	138, 141
Š	
šablóna	102
šablónové vyplňovanie	102
šrafovanie	98, 108
T	
tablet	18
teleprezencia	155
telerobotika	155

texel	45
text	
grafický	41
textúrovanie	98, 108
tieňovanie	132
Gouraudovo	132
interpoláciou farby	132
konštantné	132
kontinuálne	132
Phongovo	133
tixel	45
tlačiareň	15, 20
atramentová	21
bublinová	20
ihličková	20, 22
laserová	20, 22
LED	16, 20, 22
maticová	20, 21, 22
piezoelektrická	20
s typovým kolieskom	20
sublimačná	20
tepelná	20
trysková	20, 21, 22
trackball	17
trackpad	17
trackpoint	17
transfokácia	73, 76
transformácia	
afinná	57
geometrická	56, 57
globálna	56
okno-výrez	72
pohľadová	56
premietacia	56, 72, 78, 81
zobrazovacia	56
transformácie	
normalizačné	37
transformačná matica	58
trimetria	79
Truecolor	33
twist	94
U	
udalosť	37
určenie	36
úsečka	46
algoritmus DDA	46
alias	46
Bresenhamov algoritmus	47
USS	55
UWB	142
V	
valuator	36
VESA	32
VGA	31, 33
režimy	32
videopamäť	28, 30
viditeľnosť bodu	110
viewport	72
virtual reality	13, 155
virtuálna realita	13, 155
vnorený aktér	156
voxel	45

všeobecná Coonsova plocha	94
všeobecný grafický prvok	41
výber	36
vyhladzovanie	43, 46, 51
vyplnená oblasť	41
vyplňovanie	98
vyplňovanie spektróm	103
výplňový vzor	41
mapovanie	45
výrez	72
vzorkovanie	37

W

Wamockov algoritmus	113
---------------------	-----

X

x-buffér	118
----------	-----

Y

YCbCr	143
YUV	142

Z

záplavové vyplňovanie	106
zbemica	33
Z-buffér algoritmus	116
zmena mierky	57, 61
v 3D	61
zmenšenie rastrových objektov	64
zväčšenie rastrových objektov	62
zobrazovací adaptér	29
CGA	29
EGA	29, 30
grafický	27, 28
Hercules	29
HGC	29
MDA	29
umiestnenie	27
VGA	29, 31
zobrazovací reťazec	
2D	57
3D	57
zobrazovacia jednotka	26
zobrazovacia karta	27
zobrazovač	23
2D	54
3D	54
elektroluminiscenčný	24
farebný	26
monochromatický	26
plazmový	24
s katódovou obrazovkou	23
z tekutých kryštálov	25
zooming	73, 76
zomý ihlan	75, 117, 126
zoznam bodov	41
zrkadlenie	57, 59
rastrových objektov	60
v 3D	60
zrkadlové javy	135
Z-sort	112

Ž

žiadosť	37
---------	----