Počítačová grafika II

metody a nástroje zobrazování 3D scén

Eduard Sojka

Obsah

Úvod ke studiu

Tento text je určen jako materiál pro výuku předmětu „počítačová grafika II“ kurzu „počítačová grafika“ distančního vzdělávacího programu na VŠB-TUO. Obsahově je předmět „počítačová grafika II“ (a tedy i tento text) zaměřen na metody a nástroje pro zobrazování prostorových objektů a scén. Jedná se o prohloubení vybraných témat předmětu „počítačová grafika I“. Zvolené téma lze v počítačové grafice považovat za důležité a pro posluchače také atraktivní. Obsah jednotlivých kapitol je volen tak, aby posluchač získal informace dostatečně přesné. Podrobnost je záměrně taková, aby byl posluchač schopen i případné vlastní realizace jednotlivých postupů, a to alespoň v jejich základní variantě. Takových textů pociťuji nedostatek (na rozdíl od textů zaměřených spíše jen encyklopedicky).

K čemu tento text slouží?

Při zobrazování prostorových objektů máme náročný úkol, a to vykreslit ve dvojrozměrném obraze objekty, které jsou ve skutečnosti trojrozměrné. Pozorovatel výsledného uměle vytvořeného obrazu by při tom měl získat náležitý vjem, pokud možno blízký tomu, jaký by získal, kdyby scénu viděl sám nebo kdyby viděl její fotografii. K zobrazování trojrozměrných objektů se v praxi používá více postupů. Některé z nich kladou důraz na rychlost zobrazení a nepředpokládají, že by byla požadována špičková věrnost výsledného obrázku. Jiné naopak preferují věrnost a usilují o obrázky co nejpodobnější fotografiím, což je ale vždy za cenu rychlosti. V praxi jsou proto techniky obou uvedených základních typů potřebné a často se s oběma můžete setkat i v rámci jediného systému. Také v tomto učebním textu se seznámíte s oběma typy. Text zahrnuje zejména následující témata: a) standardní zobrazovací řetězec b) metodu rekurzivního sledování paprsku, c) vyzařovací (radiositní) metodu, d) programování v OpenGL.

Co je obsahem textu?

Standardní řetězec

Sledování paprsku

Vyzařovací metoda

Programování v OpenGL

„Standardní zobrazovací řetězec“ je technikou v praxi velmi rozšířenou. Pracuje rychle a kvalita vytvořených obrázků je při tom pro mnoho aplikací dostatečná. Setkáte se s ním v široké škále programů, jako jsou například CAD programy, počítačové hry a letecké simulátory. Tato technika zobrazování je realizována např. také v grafických standardech OpenGL a Direct3D a v posledních letech se jí dostává také intenzivní hardwarové podpory od výrobců grafických karet. Jestliže se žádá, aby vytvořený obrázek co nejvíce připomínal fotografii, pak se dnes obvykle nejčastěji používá metoda „rekurzivního sledování paprsku“ a také (možná poněkud méně často) „metoda vyzařovací“ (navíc někdy také kombinace obou metod). S obrázky produkovanými uvedenými technikami jste se už určitě také setkali. Jejich typickými odběrateli jsou architekti, návrháři a filmový a reklamní průmysl. Na obr. 1 až 3 jsou všechny tři zmíněné techniky ilustrovány svými typickými výstupy.

Všechny dříve vyjmenované techniky zobrazování trojrozměrných objektů jsou v textu probírány poměrně podrobně. Po přečtení textu budete jejich činnosti velmi dobře rozumět a také je budete schopni sami implementovat. Praktická implementace bude dokonce součástí výuky. Budete si moci zvolit mezi realizací

Co budete umět, až si text přečtete?

standardního zobrazovacího řetězce (celého nebo jeho vybraných částí) a realizací metody rekurzivního sledování paprsku. Při realizaci vás podrobně povedou vzorové programy, úkoly, které vám budou v textu průběžně ukládány, i výklad samotný, který je tomuto praktickému cíli podřízen. Realizovat někdy v životě alespoň jednu zobrazovací metodu vlastními silami je užitečné. Za svoji námahu budete odměněni tím, že o metodě a souvisejících problémech počítačové grafiky získáte představu tak přesnou, jak by pouhým teoretickým studiem bylo sotva možné. Tato podrobná představa vám později určitě přijde vhod, i když už možná budete jen používat dostupných systémů hotových. Budete dokonale rozumět jejich činnosti, jejich chování a zadávaným vstupním údajům. To si ostatně už v tomto předmětu hned sami také ověříte při studiu standardu OpenGL, který je rovněž součástí kurzu a který realizuje standardní zobrazovací řetězec. Po předchozí průpravě z vlastní realizace zobrazovacího řetězce proniknete do programování v OpenGL velmi snadno. Znalosti OpenGL můžete nepochybně přímo využít v praxi, a to při programování nejrůznějších grafických aplikací.

Ukázka 1 (*Standardní řetězec*)

Obr. 1. Obrázek získaný standardním zobrazovacím řetězcem (se svolením Martina Cvíčka, SPŠ stavební Ostrava).

Co byste měli znát, než text začnete číst?

Studium tohoto textu vyžaduje znalost základů počítačové grafiky, které prohlubuje. Potřebné znalosti byste měli získat studiem předmětu „počítačová grafika I“, který je také součástí kurzu. Z matematiky se předpokládá znalost základů maticového počtu v rozsahu ne větším, než se vyučuje na středních školách. Dále se předpokládá, že umíte programovat v některém programovacím jazyce. Svoji samostatnou práci můžete realizovat v jazyce C nebo Pascal. Vzorové

programy, na které se text odvolává, jsou ale připraveny pouze v jazyce C. Předpokládá se tedy, že jazyk ovládáte alespoň tak, abyste byli schopni číst programy v něm zapsané a případně v nich také provádět drobné změny. Příklady jsou ovšem vesměs poměrně krátké a jsou cíleně zapsány co nejjednodušeji.

Jak je předkládaný text organizován?

Předkládaný učební materiál je organizován do pěti kapitol. První kapitola je věnována afinním a projektivním prostorům a transformacím, které tvoří společný teoretický základ všech zobrazovacích postupů. Zbývající čtyři kapitoly jsou pak věnovány jednotlivým dříve uvedeným „hlavním“ tématům.

Jaká je forma textu?

S ohledem na to, že se jedná o materiál pro studium distanční, je text slovně poměrně velmi bohatý. Jinak je ale záměrně napsán klasicky ve formě dosti podobné knize. Takový způsob zpracování považuji na základě svých zkušeností za nejvýhodnější. Konkrétně lepší než text s nadbytkem různých piktogramů, rámečků, podtržení, řezů písma atd. souvisejících s organizací studia (jak se někdy v souvislosti s texty pro distanční vzdělávání doporučuje). Srozumitelnost textu je totiž určena zejména logickou strukturou výkladu, tedy obsahem sdělení, která jsou prezentována v jednotlivých kapitolách, podkapitolách, odstavcích a větách. Je-li výklad srozumitelný, pak jmenované „atrakce“ nepotřebuje (dokonce mu škodí). Výkladu nesrozumitelnému naopak stejně pomoci nemohou.

Ukázka 2 (sledování paprsku)

Obr. 2. Obrázek získaný metodou sledování paprsku (se svolením William Fawcett, The POV-Team).

Jednotlivé podkapitoly předkládaného textu lze současně považovat za základní „studijní bloky“. U každé podkapitoly je proto vyznačena předpokládaná doba studia, která se pohybuje zpravidla mezi 0,5 až 1,5 hod. Uvedené odhady je ale nutné považovat jen za velmi přibližné. Ve skutečnosti může být čas velmi individuální. Celková doba studia textu by měla činit přibližně 45 hodin. Realizace samostatné práce, o níž jsme v této kapitole mluvili již dříve, si vyžádá asi 25 hodin. Celkem si tedy studium předmětu vyžádá přibližně 70 hodin.

Celkem si studium vyžádá asi 70 hodin.

Ukázka 3 (vyzařovací metoda)

Obr. 3. Obrázek získaný vyzařovací metodou (se svolením studia „aade“ Thajsko).

Pro usnadnění studia je místo na okraji stránky vyhrazeno pro informace, které by vám při studiu měly pomoci (už i v této kapitole jste se s nimi ostatně setkali). Pro prezentaci těchto pomocných informací byla použita následující forma.

- Plný světle modrý obdélník s černým textem (nebo světle šedý obdélník při černobílém tisku) na okraji stránky vám přináší informace didaktické povahy. Takto jsou například vyznačeny cíle kapitol a podkapitol nebo jsou takto vyznačeny části, které mají povahu prohloubení učiva, a které

Jak jsou vyznačeny pokyny ke studiu?

proto můžete (chcete-li) přeskočit. Ukázku takového didaktického sdělení jste v této kapitole viděli již několikrát.

- Plný zelený obdélník s bílým textem (případně šedý obdélník) vyznačuje příklad nebo úkol. Většina příkladů je vyřešena, některé ale pouze obsahují nápovědu, jak byste je měli řešit. Úkoly zpravidla spočívají v tom, že si prohlédnete něco z přiložených doplňujících materiálů. Může se jednat např. o obrázek nebo o fragment zdrojového textu programu. Někdy také budete žádáni, abyste např. spustili program a prohlédli si výsledek jeho činnosti, případně, abyste program sami vytvořili. Příklady (zejména příklady řešené) ani úkoly nedoporučujeme přeskakovat. Jsou nedílnou součástí výkladu. Ukázku úkolu, který pravděpodobně rádi splníte, naleznete v závěru této kapitoly.

Jak jsou vyznačeny příklady a úkoly?

- Černý text v zeleném (šedém) rámečku na okraji stránky shrnuje hlavní myšlenky prezentované v odstavci vlevo. Ukázky jste v této kapitole také již viděli.

Jak jsou vyznačeny hlavní myšlenky textu?

- Plný žlutohnědý obdélník s bílým textem (nebo šedý obdélník) vyznačuje shrnutí, které bývá zpravidla na konci logických studijních celků. Příklad vidíte na konci této kapitoly.

Veškeré materiály, které budete ke studiu potřebovat (včetně učebního textu v elektronické podobě), naleznete na CD kurzu v adresáři „grafikaII“. Přečtěte si prosím soubor „čti_mne.txt“ v tomto adresáři, v němž naleznete pokyny pro instalaci a také podrobný popis obsahu jednotlivých podadresářů.

Úkol 1

Úkol 1. Chcete-li, můžete se nyní seznámit s obsahem CD (adresář „grafikaII“). Přinejmenším si můžete prohlédnout obrázky v adresářích „grafikaII\raytrac\obrázky“ „grafikaII\radiosita\obrázky“, „grafikaII\standard\obrázky“ (některé z nich naleznete také v úvodních částech jednotlivých kapitol). Věřím, že vás budou motivovat k dalšímu studiu. Vydržíte-li až do konce textu, budete umět sestrojit programy, které je kreslí, nebo přinejmenším velmi dobře rozumět jejich činnosti.

S H R N U T Í

Shrnutí: V tuto chvíli byste měli mít vcelku slušnou představu, čemu se pomocí textu a dalších souvisejících materiálů můžete naučit. Také už víte, co byste měli umět, než se do studia pustíte, a kolik úsilí vás bude studium stát.

Poděkování: Děkuji doc. Dr. Ing. Ivaně Kolingerové a B.c. Jiřímu Foltovi za pečlivé přečtení rukopisu a za cenné rady a připomínky. Dále děkuji Ing. Janu Plačkovi za vypracování vzorových programů ke kapitole 2 a 3 a svým dvěma synům Štěpánovi a Eduardovi za nakreslení většiny obrázků a za vypracování vzorových programů ke kapitole 5. Konečně také děkuji všem autorům ilustračních obrázků za to, že dali svolení k použití svých prací v tomto textu.

1 Afinní a projektivní prostory a jejich aplikace v grafických systémech

Čemu se v této kapitole naučíte?

Po prostudování této kapitoly porozumíte vybraným částem z teorie afinních a projektivních prostorů a transformací, a to v míře potřebné pro realizaci grafických systémů, pro které je tato teorie základním východiskem. Bez znalostí alespoň základů z této oblasti byste v žádném případě nemohli grafické systémy konstruovat a mnohdy ani efektivně používat. Po prostudování kapitoly budete schopni realizovat jakoukoli afinní a projektivní transformaci. Zejména transformace, které v grafických systémech realizují rovnoběžná nebo perspektivní zobrazení, a také transformace modelovací, které se používají při vytváření scény z jednotlivých objektů.

Nenechte se odradit teorií! Je základem všech grafických systémů.

Možná, že se vám kapitola může zdát poněkud teoretická. Ujišťuji vás ale, že všechny jevy, které jsou zde prezentovány, byly vybrány velmi pečlivě na základě toho, že je jejich znalost při praktické realizaci systémů nevyhnutelná. Všechny jevy také budete v následujících kapitolách využívat, jak se o tom ostatně sami přesvědčíte. Totéž plně platí i pro všechny zde uváděné řešené příklady, které jsou nedílnou součástí výkladu (často v nich naleznete důležité informace). Naopak všechny jevy, z nichž byste praktický užitek neměli, jsou potlačeny. Věřím proto, že překonáte případnou nedočkavost, nenecháte se odradit zdánlivě složitými matematickými vztahy a s chutí se do studia kapitoly pustíte. Celkem si studium této kapitoly vyžádá přibližně 10 hodin času.

1.1 Afinní prostor a afinní transformace

Doba studia asi 0,5 hod

Čemu se zde naučíte?

Afinní prostor a afinní transformace jsou základními pojmy, se kterými se lze při konstrukci grafických systémů setkat. V této podkapitole se s nimi seznámíte. Při studiu celého textu je pak budete téměř neustále potřebovat. Bylo by jistě možné začít výklad přesnými definicemi. S ohledem na účel tohoto textu však od takového postupu upustíme a spokojíme se s poněkud méně únavným (i když současně bohužel také poněkud méně přesným) úvodem vycházejícím z intuitivní představy a zkušenosti. Kdo by snad přece jen přesné definice postrádal, nalezne je shrnuty v podkapitole 1.10.

Co je afinní prostor?

Intuitivně za afinní prostor považujeme prostor obsahující body. Navíc musí být současně dán (nebo zvolen) nějaký přidružený vektorový prostor (tzv. vektorové zaměření) a nějaké zobrazení, které ke každé dvojici z prostoru bodů přiřazuje nějaký prvek onoho přidruženého prostoru vektorového. Místo dalšího zpřesňování této informace se zde spokojíme s konstatováním, že konstrukce afinního prostoru sleduje velmi praktický cíl, a to, aby bylo možné body v afinním prostoru jednoznačně specifikovat pomocí souřadnic (tedy pomocí prvků ze zmíněného přidruženého prostoru vektorového) a aby k manipulacím s body bylo možné využívat prostředků lineární algebry. Za afinní prostor je nutné, přísně

vzato, považovat nejen samotný prostor bodů, ale i onen zmíněný přidružený prostor vektorový a také zobrazení, které dvojicím bodů přiřazuje vektory. Dimenze přidruženého vektorového prostoru (který je využit k měření souřadnic) určuje také dimenzi afinního prostoru. Pro *n*-rozměrný afinní prostor budeme používat označení A_n. Nejčastěji budeme pracovat s afinními prostory trojrozměrnými. Řekněme, že v trojrozměrném afinním prostoru máme bod *X*. Jeho souřadnice vzhledem ke zvolené souřadné soustavě (přesněji je budeme nazývat afinními souřadnicemi) jsou popsány vektorem o třech prvcích $\mathbf{x} = (x_1, x_2, x_3)$. Vektor **x** je prvkem zmíněného přidruženého vektorového prostoru.

Co je euklidovský prostor?

Afinní prostor, v němž je zaveden skalární součin a norma jako odmocnina ze skalárního součinu, se nazývá *euklidovským prostorem*. Pro euklidovský *n*-rozměrný prostor budeme používat značení E_n. Zavedení skalárního součinu a normy je důležité, protože umožňuje měřit délku vektorů a úhly mezi vektory (to určitě znáte z matematiky). Nyní už asi shledáváte, že s euklidovskými a tedy i s afinními prostory máte již vlastně bohaté zkušenosti, protože jste s nimi v minulosti nepochybně nejednou při řešení geometrických úloh pracovali. Možná jste jen nebyli zvyklí zdůrazňovat, o jaké prostory se jedná.

Co je afinní transformace?

Afinní transformace je zobrazení bodů jednoho afinního prostoru do jiného afinního prostoru. Speciálním případem je zobrazení bodů afinního prostoru do téhož afinního prostoru, které je bijekcí. Tento případ se nazývá afinitou. (Bijekce je zobrazení, kdy dvěma různým vzorům vždy odpovídají dva různé obrazy. Zde tak konkrétně vylučujeme, aby se dva různé body zobrazily na bod jediný). Velmi často bude předmětem našeho zájmu afinní transformace mezi dvěma afinními prostory dimenze 3, případně afinita na prostoru dimenze 3. K matematickému popisu afinní transformace zavedeme následující vektory a matici (omezíme se zde na trojrozměrné prostory)

$$\mathbf{y} = (y_1, y_2, y_3), \quad \mathbf{x} = (x_1, x_2, x_3), \quad \mathbf{t} = (t_1, t_2, t_3),$$
$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}. \tag{1.1}$$

Jak lze afinní transformaci popsat matematicky?

Afinní transformaci lze pak popsat vztahem

$$\mathbf{y} = \mathbf{x}\mathbf{A} + \mathbf{t} \ . \tag{1.2}$$

Poznamenejme, že uvedený vztah je *důsledkem* toho, jak je afinní transformace definována (podkapitola 1.10). Pro naše potřeby je ale platnost vztahu (1.2) dostatečnou informací. V uvedeném vztahu matice **A** a vektor **t** popisují vlastnosti transformace. Vektor **y** reprezentuje bod, který je afinním obrazem bodu, který je reprezentován vektorem **x**. Jestliže přímo známe matici **A** a vektor **t**, pak lze transformaci podle předpisu (1.2) již jednoduše provést. Někdy je ale situace komplikovanější v tom smyslu, že je transformace zadána jinak než přímo hodnotami **A**, **t**. V takovém případě je obvyklé, že se hodnoty **A**, **t** na základě zadání určí a pak se opět využije vztahu (1.2). Typickými postupy, které vedou k určení potřebných hodnot, se budeme v dalším textu zabývat podrobněji.

1.2 Změna souřadné soustavy

Doba studia asi 0,5 hod

Čemu se zde naučíte?

Velmi častým případem afinní transformace mezi dvěma prostory stejné dimenze je změna souřadné soustavy a odpovídající transformace souřadnic bodů. Opět se zde omezíme na afinní (případně euklidovské) prostory trojrozměrné, protože ty budeme dále potřebovat v převážné většině případů. Zobecnění závěrů zde uvedených pro prostory s libovolným počtem dimenzí je ale snadné (máte-li chuť, můžete se o zobecnění po přečtení textu pokusit, není to ale nezbytně nutné).

Význam změny souřadné soustavy v grafických systémech

V grafických systémech se změna souřadné soustavy provádí dosti často. Někdy se jedná o vnitřní záležitost systému (pro uživatele neviditelnou), kdy je změna souřadné soustavy součástí nějakého výpočetního postupu nebo jeho odvození. Jindy může být manipulace se souřadnou soustavou patrná i pro uživatele. Jako příklad lze uvést tzv. modelovací transformace, s jejichž pomocí uživatel systému vytváří scénu (může pomocí ní zadávat např. umístění objektů ve scéně, jejich velikost atd.). Po prostudování této podkapitoly budete umět transformace souřadných soustav realizovat.

*Jak stanovit vektor **t**?*

Transformace souřadnic způsobená pouhým posunutím počátku souřadné soustavy je triviální. Nechť (p_x, p_y, p_z) jsou souřadnice nového počátku souřadné soustavy. Transformaci souřadnic z původní do nové souřadné soustavy pak popisuje vztah (1.2), v němž **A** je jednotková matice a vektor **t** je $\mathbf{t} = (-p_x, -p_y, -p_z)$.

*Jak stanovit matici **A**?*

Zabývejme se nyní poněkud komplikovanějším případem. Počátek nyní při změně souřadné soustavy ponecháme beze změny. Změníme ale osy soustavy. Předpokládejme, že původní souřadná soustava měla bázové vektory $\mathbf{v}_1$, $\mathbf{v}_2$, $\mathbf{v}_3$ (bázové vektory jsou vektory jednotkové délky udávající směry souřadných os). Bázové vektory nové souřadné soustavy jsou $\mathbf{w}_1$, $\mathbf{w}_2$, $\mathbf{w}_3$. Můžeme si myslet, že bázové vektory obou soustav vyjadřujeme vzhledem k nějaké další (už třetí) referenční souřadné soustavě. Řekněme, že máme bod X, který je v původní souřadné soustavě reprezentován vektorem $\mathbf{x} = (x_1, x_2, x_3)$. Úkolem je zjistit souřadnice bodu X vzhledem k nové souřadné soustavě. Tyto souřadnice budou vyjádřeny vektorem, který označíme $\tilde{\mathbf{x}}$. Polohový vektor bodu X (vzhledem k oné myšlené třetí referenční souřadné soustavě) můžeme vyjádřit jako součet

$$\sum_{i=1}^{3} x_i \mathbf{v}_i \,. \tag{1.3}$$

Vyjádřeme bázové vektory $\mathbf{v}_i$ souřadné soustavy původní pomocí bázových vektorů souřadné soustavy nové. Nepochybně musí být možné vyjádřit vektor $\mathbf{v}_i$ jako lineární kombinací vektorů $\mathbf{w}_1$, $\mathbf{w}_2$, $\mathbf{w}_3$ (tvoří-li trojice $\mathbf{w}_1$, $\mathbf{w}_2$, $\mathbf{w}_3$ bázi, pak lze každý vektor uvažovaného prostoru vyjádřit jako lineární kombinaci vektorů uvedené trojice, tedy i každý z vektorů $\mathbf{v}_i$). Máme proto

$$\mathbf{v}_i = \sum_{j=1}^{3} a_{ij} \mathbf{w}_j \,, \tag{1.4}$$

kde a_{ij} jsou koeficienty lineární kombinace. Dosazením vztahu (1.4) do vztahu (1.3) a dostaneme

$$\begin{aligned}\sum_{i=1}^{3} x_i \mathbf{v}_i &= \sum_{i=1}^{3} x_i \sum_{j=1}^{3} a_{ij} \mathbf{w}_j = \sum_{i=1}^{3}\sum_{j=1}^{3} x_i a_{ij} \mathbf{w}_j \\ &= (x_1 a_{11} + x_2 a_{21} + x_3 a_{31})\mathbf{w}_1 \\ &+ (x_1 a_{12} + x_2 a_{22} + x_3 a_{32})\mathbf{w}_2 \\ &+ (x_1 a_{13} + x_2 a_{23} + x_3 a_{33})\mathbf{w}_3 \ .\end{aligned} \tag{1.5}$$

Členy v závorkách v posledním ze vztahů (1.5) udávají souřadnice bodu X v nové souřadné soustavě. Uvažme, že podle našeho čekávání by měl transformaci popisovat maticový vztah $\tilde{\mathbf{x}} = \mathbf{xA}$ (protože se počátek při změně soustavy nezměnil, je ve vztahu (1.2) nutně $\mathbf{t} = \mathbf{0}$). Zjišťujeme, že když položíme

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}, \tag{1.6}$$

bude součin $\tilde{\mathbf{x}} = \mathbf{xA}$ (jednotlivé složky vektoru $\tilde{\mathbf{x}}$) vyjadřovat členy v závorkách v posledním ze vztahů (1.5), a tedy souřadnice bodu X v nové souřadné soustavě To je důležitý závěr. Vidíme, že prvky v i-tém řádku hledané matice **A** jsou souřadnice i-tého bázového vektoru původní souřadné soustavy vzhledem k souřadné soustavě nové (říká to vztah (1.4)). Tohoto závěru později mnohokrát využijeme v situaci, kdy bude zapotřebí matici **A** stanovit. Stojí za to zapamatovat si toto zjištění.

*Jak stanovit matici **A**? (důležitý závěr)*

1.3 Ortonormalita afinní transformace

Doba studia asi 0,5 hod

Mezi afinními transformacemi zaujímají významné místo transformace ortonormální. Ortonormální transformace jsou ty, při nichž zůstávají délky a úhly před a po transformaci nezměněny. Asi si vybavíte, že velmi obvyklé transformace posunutí a pootočení jsou příklady transformací ortonormálních. Často se při řešení praktických úloh vyplatí vědět o existenci této třídy transformací. Předem obvykle víte, zda zamýšlená transformace je či není ortonormální. Je-li tomu tak, pak matice **A** musí mít jisté speciální vlastnosti. Této znalosti můžete využít jednak při stanovení matice **A** a také při manipulaci s ní. Po přečtení této podkapitoly budete umět ortonormality transformací využívat.

Co je afinní ortonormální transformace?

Čemu se zde naučíte?

Jak délky, tak úhly (které mají být při ortonormální transformaci zachovány) souvisí se skalárním součinem. Uvažujme vektory **x**, **y**. Zopakujme, že skalární součin vektorů **x**, **y** (značíme jej zde $\langle \mathbf{x},\mathbf{y} \rangle$) zavádíme vztahem

$$\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}\mathbf{y}^{\mathrm{T}} \ . \tag{1.7}$$

Vlastnosti ortonormální transformace

Připomeňme dále dobře známé vztahy pro délku ($|\mathbf{x}|$) vektoru **x** a pro úhel (φ) dvou vektorů **x**, **y**. Víte, že platí

$$|\mathbf{x}| = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle} = \sqrt{\mathbf{x}\mathbf{x}^{\mathrm{T}}} \ , \tag{1.8}$$

$$\cos\varphi = \frac{\langle \mathbf{x}, \mathbf{y} \rangle}{|\mathbf{x}||\mathbf{y}|} = \frac{\mathbf{x}\mathbf{y}^{\mathrm{T}}}{|\mathbf{x}||\mathbf{y}|} \ . \tag{1.9}$$

Je zřejmé, že jestliže transformace nemění hodnotu skalárního součinu, pak také úhly a délky nejsou transformací změněny. Nechť $\mathbf{x}_1$, $\mathbf{x}_2$ jsou vektory reprezentující počáteční a koncový bod vektoru $\mathbf{x}$. Je tedy $\mathbf{x} = \mathbf{x}_2 - \mathbf{x}_1$. Afinní transformací (1.2) přejde vektor $\mathbf{x}$ ve vektor $(\mathbf{x}_2\mathbf{A}+\mathbf{t}) - (\mathbf{x}_1\mathbf{A}+\mathbf{t}) = (\mathbf{x}_2-\mathbf{x}_1)\mathbf{A} = \mathbf{x}\mathbf{A}$. Při transformaci vektoru se tedy translační složka transformace reprezentovaná vektorem $\mathbf{t}$ vůbec neuplatní (což jste ale asi intuitivně očekávali). Významná je naopak ta část transformace, která je reprezentována maticí $\mathbf{A}$. Podmínku zachování skalárního součinu vyjádříme rovnicí

$$\langle \mathbf{x}, \mathbf{x} \rangle = \langle \mathbf{x}\mathbf{A}, \mathbf{x}\mathbf{A} \rangle \ . \tag{1.10}$$

Odtud pak plyne požadavek

$$\mathbf{x}\mathbf{x}^{\mathbf{T}} = (\mathbf{x}\mathbf{A})(\mathbf{x}\mathbf{A})^{\mathrm{T}} = \mathbf{x}\mathbf{A}\mathbf{A}^{\mathrm{T}}\mathbf{x}^{\mathrm{T}} \ . \tag{1.11}$$

Z předchozího vztahu je již zřejmé, že afinní transformace bude zachovávat hodnotu skalárního součinu tehdy, jestliže bude splněna podmínka $\mathbf{A}\mathbf{A}^{\mathrm{T}}=\mathbf{I}$, kde $\mathbf{I}$ je jednotková matice. Uvedená podmínka je nutnou a postačující podmínkou ortonormality transformace. Z definice inverzní matice máme $\mathbf{A}^{-1}\mathbf{A}=\mathbf{I}$. Vidíme tedy, že dále musí platit

$$\mathbf{A}^{-1} = \mathbf{A}^{\mathrm{T}} \ . \tag{1.12}$$

*Jaké vlastnosti má matice **A** ortonormální transformace?*

Uvedené zjištění má praktický význam. Ukazuje, jak lze jednoduše vypočítat inverzní matici k matici ortonormální transformace. Inverzní matice se jednoduše rovná matici transponované. Ukažme konečně ještě jednu vlastnost matice ortonormální transformace, a to, že její determinant musí být roven hodnotě ±1. Zdůvodnění tohoto faktu není obtížné. Protože platí $\det(\mathbf{A}\mathbf{A}^{\mathrm{T}}) = \det(\mathbf{A})\det(\mathbf{A}^{\mathrm{T}}) = \det^2(\mathbf{A})$, a protože $\det(\mathbf{I})=1$, musí být také

$$\det^2(\mathbf{A}) = 1 \ . \tag{1.13}$$

Příklad.1.1

*Praktické stanovení matice **A** afinní transformace*

Příklad 1.1. V tomto příkladě shrneme dosud získané poznatky o afinní transformaci. Příklad je záměrně volen tak jednoduchý, abyste mohli porovnat výsledek získaný teoretickým řešením s výsledkem, který očekáváte. Uvažujte trojrozměrný afinní prostor. Nová soustava souřadnic je pootočena vůči původní soustavě o úhel φ kolem osy *z*. Nalezněte převodní vztah pro přepočet souřadnic bodů z původní souřadné soustavy do nové.

Řešení. Se záměrem připravit se na řešení pozdějších úloh budeme postupovat takto: Bázové vektory nové soustavy označíme $\mathbf{w}_1$, $\mathbf{w}_2$, $\mathbf{w}_3$. Jejich souřadnice vyjádříme vzhledem k původní souřadné soustavě. (Často je tento postup příjemnější, než vyjadřovat souřadnice bázových vektorů původní souřadné soustavy vzhledem k nové soustavě, jak bychom podle podkapitoly 1.2 měli učinit). Dostaneme

$$\mathbf{w}_1 = (\cos\varphi, \sin\varphi, 0), \quad \mathbf{w}_2 = (-\sin\varphi, \cos\varphi, 0), \quad \mathbf{w}_3 = (0, 0, 1) \ .$$

Na základě výsledku z podkapitoly 1.2 nyní již můžeme zapsat předpis, pomocí kterého je možné přepočítat souřadnice bodů z nové do původní souřadné soustavy (což je zatím ale naopak, než bylo požadováno v zadání). Máme

$$\mathbf{x}_{\text{old}} = \mathbf{x}_{\text{new}}\mathbf{A}\,, \quad \text{kde je} \quad \mathbf{A} = \begin{bmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Z předchozího vztahu a s přihlédnutím ke skutečnosti, že rotace je ortonormální transformací (protože nemění délky ani úhly) snadno dostaneme

$$\mathbf{x}_{\text{new}} = \mathbf{x}_{\text{old}}\mathbf{A}^{-1} = \mathbf{x}_{\text{old}}\mathbf{A}^{\mathrm{T}}\,. \qquad \square$$

Příklad 1.2

Příklad 1.2. Stanovte matici **A** pro transformaci spočívající v rotaci souřadné soustavy postupně kolem souřadných os *x*, *y*, *z* o úhly α, β, γ. (Rotace o úhel β se myslí okolo osy *y* soustavy, která vznikla dříve provedenou rotací původní soustavy o úhel α kolem její osy *x*. Analogicky se předpokládá, že před rotací okolo osy *z* o úhel γ byly již obě předchozí rotace kolem os *x* a *y* již provedeny.)

Stanovení matice **A** *pro složenou transformaci*

Řešení. Při řešení budeme postupovat tak, že nejprve převedeme souřadnice ze souřadné soustavy původní do souřadnic v souřadné soustavy pootočené kolem osy *x*, ty pak dále převedeme do souřadné soustavy pootočené kolem osy *y* a ty opět nakonec i do souřadné soustavy pootočené kolem osy *z*. Tuto posloupnost postupně prováděných transformací lze matematicky popsat vztahem

$$\mathbf{x}_{\text{new}} = \big((\mathbf{x}_{\text{old}}\mathbf{A}_x)\mathbf{A}_y\big)\mathbf{A}_z = \mathbf{x}_{\text{old}}\mathbf{A}_x\mathbf{A}_y\mathbf{A}_z\,.$$

Matice $\mathbf{A}_x$, $\mathbf{A}_y$, $\mathbf{A}_z$ odpovídají rotaci kolem jednotlivých souřadných os. S ohledem na výsledek předchozího příkladu, v němž jsme odvodili předpis pro matici $\mathbf{A}_z^{\mathrm{T}}$, je můžeme ihned zapsat. Máme

$$\mathbf{A}_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix},\ \mathbf{A}_y = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix},\ A_z = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

$\square$

Úkol 1.3

Úkol 1.3. Promyslete, kolik nezávislých hodnot obsahuje matice **A** afinní ortonormální transformace. Uvažujte trojrozměrný prostor.

Řešení. Třebaže matice **A** obsahuje celkem 9 prvků, jsou v případě ortonormální transformace jen tři z nich nezávislé. Ostatní prvky je totiž možné dopočítat ze vztahu $\mathbf{A}\mathbf{A}^{\mathrm{T}}=\mathbf{I}$, který poskytuje šest rovnic pro šest zbývajících prvků. Situaci již také ilustroval předchozí příklad. Rotace je ortonormální transformací. Nebylo proto divu, že k jejímu popisu stačily jen tři nezávislé hodnoty. V předchozím příkladě to byly úhly α, β, γ.

S H R N U T Í

Shrnutí: V tuto chvíli byste měli zejména znát, jak lze matematicky popsat afinní transformaci. Měli byste být schopni stanovit matici a vektor, které transformaci popisují. Měli byste vědět, co je ortonormální transformace a také byste měli umět těžit ze speciálních vlastností matice, která ji popisuje.

1.4 Projektivní prostor a projektivní transformace

Doba studia asi 2 hod

Při použití afinních prostorů v grafických systémech lze v některých situacích narazit na jisté problémy. Významným problémem je například to, že není jednoduše možné pracovat s body v nekonečnu (s tzv. nevlastními body). I když jsou praktické scény zpravidla konečné, mohou nevlastní body vzniknout při zobrazování scén (např. při velmi obvyklém středovém promítání na rovinnou průmětnu může průmět některých bodů scény padnout do nekonečna). Problém spočívá v tom, že vektor reprezentující nevlastní bod v afinním prostoru by měl mít jednu nebo více složek rovných hodnotě $\pm\infty$. Hodnotu ∞, jak víte, nelze ale v počítači jednoduše reprezentovat. Navíc by tato hodnota měla být i výsledkem aritmetických operací, což opět není jednoduše možné. Uvedené potíže lze překonat zavedením projektivního prostoru. Projektivní transformace, která se nad projektivními prostory zavádí, je také obecnější než transformace afinní. Například: Zatímco pomocí afinní transformace lze realizovat zobrazení pouze rovnoběžným promítáním, umožňuje projektivní transformace provádět promítání rovnoběžné i středové. V této podkapitole se projektivní prostor a projektivní transformaci naučíte používat.

Proč má smysl používat projektivní prostor?

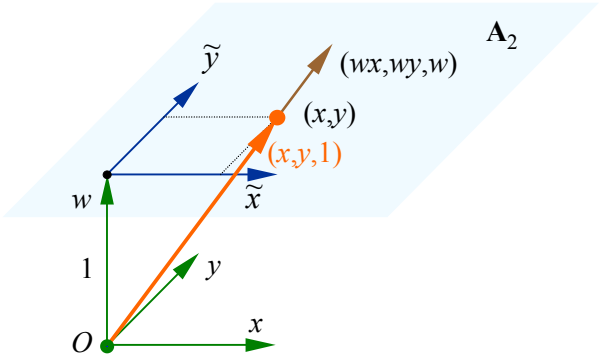
Čemu se zde naučíte?

Zavedení projektivního prostoru a tzv. homogenních souřadnic, které v projektivním prostoru jednotlivé body reprezentují, lze názorně vysvětlit ve dvojrozměrném případě (obr. 1.1). Uvažujme dvojrozměrný afinní prostor $\mathbf{A}_2$ a zaveďme nyní přidružený trojrozměrný prostor $\mathbf{V}_3$ tak, že všechny body prostoru $\mathbf{A}_2$ leží v rovině $w = 1$ tohoto přidruženého prostoru (obr. 1.1). Dvojrozměrný projektivní prostor $\mathbf{P}_2$ zavádíme jako množinu směrů ve $\mathbf{V}_3$. Vysvětleme blíže smysl takového počínání. V afinním prostoru $\mathbf{A}_2$ uvažujme bod, jehož afinní souřadnice jsou (x, y) (obr. 1.1). Homogenními souřadnicemi uvažovaného bodu, které jej reprezentují v právě zavedeném prostoru $\mathbf{P}_2$, je trojice (wx, wy, w), kde w je libovolné reálné číslo různé od nuly. V $\mathbf{P}_2$ je tedy uvažovaný bod reprezentován nekonečným počtem vektorů přidruženého vektorového prostoru $\mathbf{V}_3$, které jsou ovšem všechny navzájem kolineární. Speciálně můžeme také volit $w = 1$ (zvláštní význam této volby je patrný z obr. 1.1). Dostaneme tak trojici $(x, y, 1)$. Je-li naopak trojice (x, y, w) homogenními souřadnicemi nějakého bodu, pak také trojice $(x/w, y/w, 1)$ je homogenními souřadnicemi téhož bodu a dvojice $(x/w, y/w)$ jeho souřadnicemi afinními. Přechod od souřadnic (x, y, w) k souřadnicím $(x/w, y/w, 1)$ bývá nazýván normalizací.

Jak si lze projektivní prostor představit?

Co jsou homogenní souřadnice?

Převody mezi afinními a homogenními souřadnicemi



Obr. 1.1. Zavedení homogenních souřadnic.

Jak jsou v projektivním prostoru reprezentovány nevlastní body?

Shrneme-li to, co bylo dříve uvedeno, můžeme říci, že v projektivním prostoru je každý bod reprezentován nějakým vektorem, který k bodu směřuje. Je při tom jedno, jak dlouhý vektor je. Nyní je již zřejmé, proč nebudou s body v nekonečnu v projektivních prostorech potíže. I ony totiž mohou být reprezentovány vektory konečné délky. Uvažujme například bod ležící v obr. 1.1 v nekonečnu na souřadnicové ose $\tilde{x}$. Tento bod může být reprezentován např. trojicí (1, 0, 0).

Jak využívají projektivních prostorů grafické systémy?

... velmi brzy převádějí souřadnice afinní na homogenní.

... k afinním souřadnicím se vracejí až po ořezání zorným objemem.

Jestliže jsme se až doposud zabývali pouze dvojrozměrnými projektivními prostory, pak to bylo proto, že bylo snadné s pomocí obrázku ukázat zavedení takového prostoru. Grafické systémy ale obvykle pracují s prostory trojrozměrnými. Vektor homogenních souřadnic má v trojrozměrném projektivním prostoru čtyři složky. Je typické, že poloha bodů ve scéně bývá uživatelem prvotně popsána pomocí souřadnic afinních. Řekněme, že (x, y, z) jsou afinní souřadnice bodu X ve scéně, který neleží v nekonečnu. Grafické systémy při své činnosti obvykle velmi brzy převádějí souřadnice afinní na homogenní. Nejčastěji jednoduše tak, že jako homogenní souřadnice bodu berou čtveřici (x, y, z, 1). Tím je z nekonečně mnoha možných vektorů reprezentujících bod X v $\mathbf{P}_3$ vybrán jediný, a to ten, pro který platí $w = 1$. Zobrazení středovým promítáním se prakticky vždy řeší v homogenních souřadnicích. Před vykreslením výsledného obrazu (např. na obrazovku nebo do souboru) je ale zapotřebí vrátit se k souřadnicím afinním. To se provádí (jak již víte) normalizací. Před provedením normalizace bývá provedeno ořezání výsledného obrazu do zvoleného konečného zorného objemu, takže obraz již neobsahuje nevlastní body, které by jinak při provádění normalizace vedly k dělení nulou. Také rovnoběžné promítaní a modelovací transformace se v grafických systémech obvykle řeší s využitím homogenních souřadnic, a to přestože by to v tomto případě nebylo nezbytně nutné. Zde se jednoduše i pro řešení speciální úlohy využívá obecnějšího nástroje, protože jej systém zpravidla stejně tak či tak již obsahuje.

Příklad 1.4

Rovnice přímky v homogenních souřadnicích

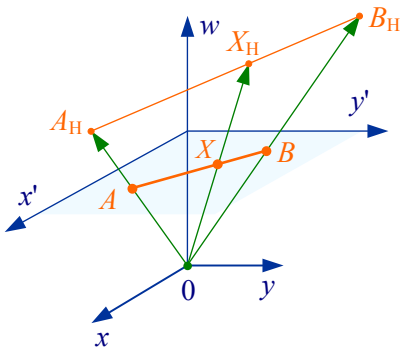
Příklad 1.4. Cílem tohoto příkladu je ukázat, že úlohy, které se čtenář ve středoškolské a vysokoškolské matematice již naučil řešit v prostorech afinních, lze stejně dobře a jednoduše řešit také v prostorech projektivních. Uvažujme následující úlohu: Určete v $\mathbf{P}_2$ rovnici přímky, která prochází body A, B o homogenních souřadnicích (x_A, y_A, w_A), (x_B, y_B, w_B). Řešte nejprve obecně, pak pro $A = (0, 0, 1)$, $B = (1, 0, 1)$ a pro $A = (1, 0, 1)$, $B = (0, 1, 1)$.

Řešení. Abychom co nejvíce ilustrovali příbuznost s již známými řešeními v afinním prostoru, ukážeme dvě řešení zadané úlohy.

1) Nechť $(\tilde{x}, \tilde{y})$ jsou afinní souřadnice bodu. Uvažujme dobře známou rovnici přímky ve tvaru $a\tilde{x} + b\tilde{y} + c = 0$, která platí pro souřadnice afinní. Nechť (x, y, w) jsou homogenní souřadnice téhož bodu. Víme, že afinní souřadnice lze získat normalizací. Máme $\tilde{x} = x / w$, $\tilde{y} = y / w$. Dosadíme-li hodnoty $\tilde{x}, \tilde{y}$ do dříve uvedené rovnice přímky, dostaneme po úpravě rovnici ve tvaru $ax + by + cw = 0$, která platí pro body zadané v homogenních souřadnicích. Předpokládejme na okamžik, že hodnoty a, b, c jsou známy. Násobíme-li rovnici reálným číslem $\lambda \neq 0$, pak dostaneme rovnici $\lambda ax + \lambda by + \lambda cw = 0$. Je zřejmé, že také rovnice $\hat{a}x + \hat{b}y + \hat{c}w = 0$, kde $\hat{a} = \lambda a$, $\hat{b} = \lambda b$, $\hat{c} = \lambda c$, je rovnicí téže přímky. Odtud vyplývá, že hodnoty a, b, c nebude možné jednoznačně stanovit pouze ze znalosti souřadnic bodů, jimiž přímka prochází. Bude zapotřebí použít vhodné doplňující

podmínky. (Na stejný problém bychom ovšem narazili i při řešení v prostoru afinním.) Pro stanovení koeficientů a, b, c tak máme rovnice $ax_A + by_A + cw_A = 0$, $ax_B + by_B + cw_B = 0$. „Univerzální“ doplňující podmínkou je např. podmínka ve tvaru $a^2 + b^2 + c^2 = 1$. Použití uvedené podmínky ovšem bohužel vede na systém, který je nelineární. Tuto potíž lze obejít např. použitím podmínky $a + b + c = 1$ nebo podmínky $a + b + c = 0$. Použije se ta varianta, pro niž má systém řešení. Uvedeným postupem získáme pro první a druhou dvojici zadaných bodů A, B rovnice přímek ve tvaru $x + y - w = 0$, $x - y = 0$.

2) Velmi jednoduše je možné rovnici přímky procházející body A, B zapsat ve tvaru $X = (1-\lambda)A + \lambda B$, kde λ je reálný parametr ($-\infty<\lambda<\infty$). Je známo, že vztah $X = (1-\lambda)A + \lambda B$ je parametrickou rovnicí přímky v afinních souřadnicích. Že je rovnicí přímky i při použití souřadnic homogenních ukazuje obr. 1.2. Body A, B jsou na obr. 1.2 reprezentovány vektory, jejichž koncové body jsou označeny jako A_H, B_H. Rovnice $X_H = (1-\lambda)A_H + \lambda B_H$ je rovnicí přímky ve $\mathbf{V}_3$. Protože body O, A, B, A_H, B_H leží v rovině, reprezentuje vektor s koncovým bodem v X_H bod X ležící na přímce spojující body A, B.



Obr. 1.2. K rovnici přímky ve tvaru $X = (1-\lambda)A + \lambda B$ v homogenních souřadnicích.

□

Co je projektivní transformace (kolineace)?

Zásadní úlohu v grafických systémech sehrává projektivní transformace (kolineace). Kolineace je zobrazením bodů jednoho projektivního prostoru na body téhož nebo jiného prostoru. Pro přesnou definici čtenáře odkazujeme na podkapitolu 1.10 (není však nezbytné ji studovat). Z hlediska praktické potřeby nám zde postačí konstatovat, že kolineaci lze matematicky popsat vztahem

$$\mathbf{y} = \mathbf{x}\mathbf{T} \ . \qquad (1.13)$$

Jak kolineaci reprezentovat matematicky?

Vektory **x** a **y** reprezentují body před resp. po transformaci. Matice **T** charakterizuje kolineaci. Grafické systémy pracují nejčastěji s trojrozměrným projektivním prostorem. Pak jsou **x**, **y** čtyřprvkové vektory homogenních souřadnic a **T** je matice rozměru 4×4. Podle tvaru matice **T** lze usuzovat na transformaci, kterou matice provádí. Je naopak také možné k požadované transformaci stanovit matici **T**. V následujícím přehledu uvedeme několik základních případů. Je užitečné se s nimi seznámit, a to mimo jiné také proto, že z nich lze sestavovat transformace složitější. Omezíme se na transformace v prostoru $\mathbf{P}_3$.

Základní projektivní transformace

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

Změna měřítka na jedné ose

Transformace popsaná uvedenou maticí provádí změnu měřítka na ose x. Můžete se o tom snadno přesvědčit, když transformaci maticí $\mathbf{T}$ aplikujete na bod o souřadnicích (x, y, z, 1). Dostanete $(x,y,z,1)\mathbf{T} = (s_x x, y, z, 1)$.

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

Změna měřítka na všech osách

Transformace provádí změnu měřítka na všech osách, což je patrné ze vztahu $(x,y,z,1)\mathbf{T} = (x,y,z,1/s) \rightarrow (sx, sy, sz, 1)$ (symbolem $\rightarrow$ jsme označili operaci normalizace homogenních souřadnic).

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

Zkosení

Transformace, která je popsána touto maticí, bývá obvykle nazývána zkosením. Je opět jednoduché se přesvědčit o oprávněnosti tohoto názvu. Provedením násobení dostáváme $(x,y,z,1)\mathbf{T} = (x, y + ax, z, 1)$. Vidíme, že rovina y = const transformací přejde v rovinu y = const + ax (tedy „zkosí se“; a je směrnice úhlu zkosení). Roviny x = const, z = const zůstanou uvedenou transformací nedotčeny. Matice popisující transformaci, která by „zkosila“ také tyto roviny, lze snadno zkonstruovat analogicky, což ponecháváme čtenáři jako drobné samostatné cvičení.

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

Posunutí

Jak vyplývá ze vztahu $(x,y,z,1)\mathbf{T} = (x + p_x, y, z, 1)$, popisuje tato matice posunutí bodu o hodnotu p_x ve směru souřadné osy x. Matice realizující translace ve směru souřadných os y, z jsou triviální modifikací matice zde uvedené.

$$\mathbf{T}=\begin{bmatrix} c & s & 0 & 0\\ -s & c & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1 \end{bmatrix},$$

kde $c^2 + s^2 = 1$

Rotace

Transformace popsaná uvedenou maticí je rotací kolem souřadnicové osy z. Symboly s, c značí hodnoty sinu, resp. kosinu úhlu rotace. Provedením maticového násobení dostáváme $(x,y,z,1)\mathbf{T} = (xc - ys, xs + yc, z, 1)$. Matice realizující rotaci kolem os x, y je možné zkonstruovat analogicky. Je užitečné, abyste se opět pokusili odpovídající matice zapsat.

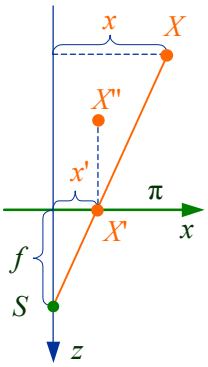
$$\mathbf{T}=\begin{bmatrix} 1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & -1/f\\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Středové promítání

Matice popisuje projekci ze středu o souřadnicích (0, 0, f) na rovinu z = 0 (tedy na rovinu xy). Ukážeme, jak je možné se o tom přesvědčit. Provedením transformace pro bod (x, y, z, 1) podle předpisu (1.13) máme

$$(x,y,z,1)\mathbf{T} = \left(x, y, z, \frac{f-z}{f}\right) \rightarrow \left(\frac{f}{f-z}x, \frac{f}{f-z}y, \frac{f}{f-z}z, 1\right).$$

Porovnejme tento výsledek s tím, co od dané projekce



Obr. 1.3. K ověření činnosti matice středového promítání.

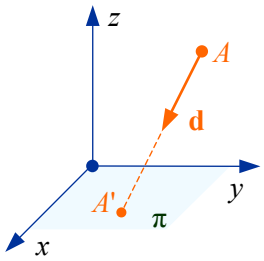
intuitivně očekáváme. Řekněme, že má být promítnut bod X o afinních souřadnicích x, y, z (obr. 1.3). Jeho průmět označme X'. Afinní souřadnice průmětu nechť jsou x', y', z'. Z podobnosti trojúhelníků (obr. 1.3) máme $x/(f–z) = x'/f$ (vezměte v úvahu, že souřadnice z bodu X na obr. 1.3 je záporná). Odtud plyne $x' = xf/(f–z)$. Podobně můžeme získat i souřadnici y'. Vyjde $y' = yf/(f–z)$. Souřadnici z' očekáváme $z' = 0$. Porovnáním uvedených hodnot s hodnotami získanými podle předpisu (1.13) zjišťujeme shodu u souřadnic x', y'. V souřadnici z' zjišťujeme rozdíl. Zatímco intuitivně očekávaná hodnota je $z' = 0$, projektivní transformace dává hodnotu $z' = zf/(f–z)$. Výsledkem projektivní transformace tedy není bod X', ale bod X'' (obr. 1.3). To ale nevadí. Spíše se jedná o přednost. K vykreslení průmětu (např. na obrazovku) se totiž použijí souřadnice x', y'. Nenulová souřadnice z' může být využita k řešení viditelnosti.

Rovnoběžné promítání

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{d_x}{d_z} & -\frac{d_y}{d_z} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Matice popisuje rovnoběžné promítání na rovinu xy. Směr promítacího paprsku je $\mathbf{d} = (d_x, d_y, d_z)$ (obr. 1.4). O správnosti matice se opět snadno přesvědčíme. Nechť A je bod, který má být promítán, a nechť $\mathbf{a} = (a_x, a_y, a_z)$ je vektor jeho afinních souřadnic. Určíme polohový vektor $\mathbf{a}' = (a'_x, a'_y, a'_z)$ průmětu bodu A'. Rovnice promítacího paprsku procházejícího bodem A je (t je parametr)

$$\mathbf{x} = \mathbf{a} + t\mathbf{d} \ .$$



Obr. 1.4. K ověření činnosti matice rovnoběžného promítání.

Z podmínky $a'_z = 0$ dostáváme

$$t = -\frac{a_z}{d_z} \ .$$

Odtud dále máme

$$a'_x = a_x - \frac{a_z}{d_z} d_x \,, \quad a'_y = a_y - \frac{a_z}{d_z} d_y \,.$$

Porovnáním právě získaných výsledků s výsledky, které poskytuje transformace popsaná maticí $\mathbf{T}$, zjišťujeme shodu v prvních dvou souřadnicích. Jednička na pozici 3, 3 v matici $\mathbf{T}$ zajišťuje, že souřadnice z průmětu je stejná jako u původního bodu. To je opět výhodné pro následné řešení viditelnosti.

Skládání transformací

Složitější transformace bývají běžně vytvářeny skládáním transformací jednodušších. Zabývejme se proto nyní i touto možností. Řekněme například, že jsme postupně provedli dvě transformace. Nejprve transformaci popsanou maticí $\mathbf{T}_1$, pak transformaci $\mathbf{T}_2$. Nechť $\mathbf{x}$ je vektor souřadnic nějakého bodu před provedením obou transformací. Přepočet souřadnic odpovídající první transformaci provedeme podle vztahu $\mathbf{x}\mathbf{T}_1$. Skutečnost, že dále pak následovala ještě transformace $\mathbf{T}_2$, zohledníme výpočtem podle vztahu $(\mathbf{x}\mathbf{T}_1)\mathbf{T}_2$. Protože platí $(\mathbf{x}\mathbf{T}_1)\mathbf{T}_2 = \mathbf{x}(\mathbf{T}_1\mathbf{T}_2)$, vidíme, že na uvedené dvě postupně prováděné transformace můžeme pohlížet jako na transformaci jedinou, která je popsána maticí $\mathbf{T}_1\mathbf{T}_2$. Právě

popsaný postup skládání transformací násobením matic, které je popisují, lze pochopitelně zobecnit na libovolný počet dílčích transformací, jak ostatně v následujících příkladech velmi brzy uvidíte.

Vztah mezi afinní a projektivní transformací

Z předchozího přehledu základních transformací vyplývá, že projektivní transformace je mocnější než transformace afinní. Afinní transformaci ze vztahu (1.2) lze realizovat jako speciální případ transformace projektivní (1.13) a tvůrci systémů to tak také velmi často dělají. Matice **T** má pak v takovém případě tvar (viz též vztah (1.2))

$$\mathbf{T} = \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{t} & 1 \end{bmatrix},$$

kde **0** je sloupcový vektor obsahující 3 nuly. Rozdíl mezi afinní a projektivní transformací lze slovně charakterizovat takto. Obě transformace jsou lineární. To znamená, že lineární objekty (přímky, roviny) jsou oběma transformacemi zobrazeny opět na objekty lineární. Na rozdíl od projektivní transformace zachovává transformace afinní rovnoběžnost. Přímky nebo roviny, které jsou rovnoběžné před transformací, zůstanou rovnoběžné i po afinní transformaci.

Příklad 1.5

Příklad 1.5. Zapište matici $\mathbf{T}_a$ projektivní transformace, která realizuje posunutí objektu o vektor (x, y, z), a dále matici $\mathbf{T}_b$, která realizuje rotaci objektu kolem osy x o úhel α a pak ještě rotaci kolem osy y o úhel β.

Řešení. Obě hledané matice snadno sestavíme s využitím předchozího přehledu, s využitím výsledku příkladu 1.2 a s uvážením, že na rozdíl od příkladu 1.2 zde provádíme transformaci objektu a nikoli transformaci souřadné soustavy (obě varianty se navzájem liší znaménkem složek translace i úhlů rotace). Bude proto

$$\mathbf{T}_a = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x & y & z & 1 \end{bmatrix}, \quad \mathbf{T}_b = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha & 0 \\ 0 & -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\beta & 0 & -\sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

□

Příklad 1.6

Příklad 1.6. Ukažte, že matice

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

Ještě jedna středová projekce

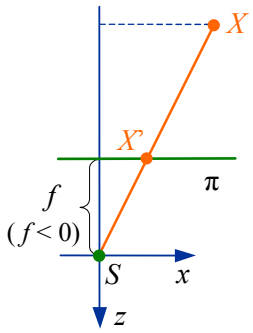
realizuje středové promítání ze středu (0, 0, 0) na rovinu $z = f$ (obr. 1.5).

Řešení. Snadno se o tom opět můžeme přesvědčit. Realizací transformace podle předpisu (1.13) máme

$$(x, y, z, 1)\mathbf{T} = \left(x, y, z, \frac{z}{f}\right) \rightarrow \left(\frac{f}{z}x, \frac{f}{z}y, f, 1\right).$$

Hodnoty, které od uvedené projekce očekáváme intuicí, lze opět snadno odvodit pomocí obr. 1.5, podobně jako jsme to provedli u středového promítání

z předchozího přehledu. Tentokrát zjistíme shodu u všech tří souřadnic (x, y, z), což prokazuje správnost dokazovaného tvrzení. Za povšimnutí stojí, že ke shodě nyní došlo i ve třetí souřadnici. Jak intuice, tak transformace popsaná maticí dávají shodně třetí souřadnici průmětu $z = f$. To ale činí uvedenou variantu poněkud méně praktickou, jestliže by měl následovat výpočet viditelnosti.



Obr. 1.5. K ověření činnosti matice středového promítání z příkladu 1.6.

□

Příklad 1.7

Příklad 1.7. Ukažte, že matic realizujících nějakou jedinou konkrétní projektivní transformaci je nekonečně mnoho.

Řešení. Předpokládejme, že $\mathbf{T}$ je matice popisující uvažovanou transformaci. Nechť X značí bod, který je transformací zobrazen na bod Y. Body X, Y nechť jsou reprezentovány takovými vektory $\mathbf{x}$, $\mathbf{y}$ svých homogenních souřadnic, že platí $\mathbf{y} = \mathbf{x}\mathbf{T}$. Vezměme nyní libovolné reálné $\lambda \neq 0$ a násobme jím uvedenou rovnici. Dostaneme vztah $\lambda\mathbf{y} = \mathbf{x}(\lambda\mathbf{T})$. Jestliže vektor $\mathbf{y}$ souřadnic reprezentoval bod Y (což jsme předpokládali), pak vektor $\lambda\mathbf{y}$ reprezentuje bod Y také (víme totiž, že v projektivním prostoru je každý bod reprezentován nekonečně mnoha kolineárními vektory). Shledáváme tedy, že jestliže $\mathbf{T}$ byla matice realizující požadovanou projekci, pak lze tutéž projekci realizovat i maticí $\lambda\mathbf{T}$.

K jediné projektivní transformaci existuje nekonečně mnoho matic T.

□

Příklad 1.8

Příklad 1.8. Odhadněte, jakou transformaci realizuje matice

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & -1/f \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Řešení. Na základě předchozího přehledu lze snadno odhadnout, že se jedná o matici realizující středové promítání na rovinu yz. Afinní souřadnice středu projekce jsou $(f, 0, 0)$.

□

S H R N U T Í

Shrnutí: V tuto chvíli byste měli zejména znát, jak lze matematicky popsat projektivní transformaci. Měli byste být schopni stanovit matici popisující jednoduché transformace nebo naopak rozhodnout, jakou transformaci zadaná (jednoduchá) matice provádí. Těchto znalostí využijeme v následujících příkladech, v nichž budeme z jednoduchých transformací skládat transformace složitější, které již budou velmi praktické.

1.5 Stanovení matice zobrazovací transformace – příklad A

Doba studia asi 0,5 hod

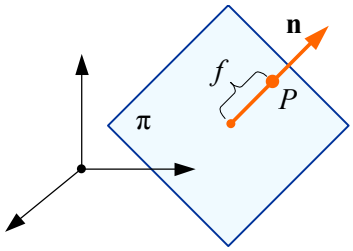
Příklad A

Tyto „velké“ příklady ukazují, jak řešit praktické úlohy.

V tomto příkladě ukážeme postup, jak lze dříve uvedených základních transformací a jim odpovídajících matic využít při sestavení matice realizující poněkud komplikovanější a prakticky již velmi užitečnou transformaci. Jedná se ve své podstatě o příklad, který je ale tak významný, že jej zařazujeme do textu jako samostatnou podkapitolu (v následujících podkapitolách budou prezentovány ještě další podobné příklady).

Úkolem je nalézt matici realizující středové promítání. Promítání je dáno středem $P = (p_x, p_y, p_z)$, normálou $\mathbf{n}$ zobrazovací roviny, $\mathbf{n} = (n_x, n_y, n_z)$, a ohniskovou vzdáleností f, tj. vzdáleností středu projekce od zobrazovací roviny (obr. 1.6). Předpokládáme, že délka vektoru $\mathbf{n}$ je $|\mathbf{n}|=1$ (tento předpoklad zjednoduší vztahy, které budeme později odvozovat).

*Projekce je zadána prvky P, **n**, f.*



Obr. 1.6. Prvky zadávající promítání.

Řešení. Hledanou transformaci rozdělíme na čtyři transformace dílčí. Cílem prvních tří kroků bude provést takovou transformaci souřadnic, abychom obdrželi jeden ze speciálních případů projekce z předchozí podkapitoly, pro který již odpovídající matici známe. Provedeme následující: 1) Počátek souřadné soustavy, kterou máme ve scéně zavedenu, posuneme do středu projekce P. 2) Provedeme rotaci této posunuté souřadné soustavy tak, aby její osa z' splynula s normálou $\mathbf{n}$ zobrazovací roviny. 3) Posuneme počátek pootočené souřadné soustavy proti směru její osy z' o délku f tak, aby její počátek padl do zobrazovací roviny. (Po provedení dosud vyjmenovaných tří kroků leží rovina xy souřadné soustavy v rovině projekce. Střed projekce leží na ose z soustavy, a to v její kladné části ve vzdálenosti f od počátku.) 4) Provedeme projekci (nyní se již jedná o speciální případ uvedený v přehledu transformací v předchozí podkapitole, kdy střed projekce leží na ose z a zobrazovací rovina je $z = 0$). Označíme-li matice uvedených transformací postupně jako $\mathbf{T}_1$, $\mathbf{T}_2$, $\mathbf{T}_3$, $\mathbf{T}_4$, pak hledaná výsledná matice $\mathbf{T}$ bude jejich součinem $\mathbf{T} = \mathbf{T}_1\mathbf{T}_2\mathbf{T}_3\mathbf{T}_4$. S využitím výsledků z předchozí podkapitoly můžeme ihned psát vztahy pro matice $\mathbf{T}_1$, $\mathbf{T}_3$, $\mathbf{T}_4$. Máme

Hledanou transformaci sestavíme ze čtyř transformací jednoduchých.

$$\mathbf{T}_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -p_x & -p_y & -p_z & 1 \end{bmatrix}, \quad \mathbf{T}_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & f & 1 \end{bmatrix}, \quad \mathbf{T}_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1/f \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Ke stanovení zbývající matice rotace $\mathbf{T}_2$ použijeme postupu popsaného v podkapitole 1.2 a v příkladě 1.1. K tomu určíme souřadnice bázových vektorů souřadné soustavy po rotaci (nová souřadná soustava) vzhledem k souřadné soustavě platné před rotací (stará souřadná soustava). Bázové vektory staré souřadné soustavy označme **x**, **y**, **z**. Bázové vektory nové souřadné soustavy jsou **x'**, **y'**, **z'**. (Poznamenejme, že i když je souřadná soustava platná před rotací vůči původní soustavě ve scéně již posunuta, jsou její bázové vektory **x**, **y**, **z** stále tytéž jako u původní souřadné soustavy ve scéně.) Pro polohu souřadné soustavy po rotaci máme podmínku, aby osa *z*' byla kolmá k zobrazovací rovině (aby tedy měla směr zadaného vektoru **n**). Podmínka však nespecifikuje polohu nové souřadné soustavy jednoznačně (soustava může zatím rotovat kolem své osy *z*'). Zvolíme proto další doplňující podmínku. Požadujeme, aby směr osy *x*' soustavy po rotaci byl kolmý nejen k ose *z*', ale také k ose *z* souřadné soustavy před rotací, tedy k vektoru **z** = (0, 0, 1). Předpokládáme při tom, že vektory **n** a **z** nejsou kolineární. Z uvedených podmínek máme

$$\mathbf{z}' = \mathbf{n} \ ,$$

$$\mathbf{x}' = \frac{\mathbf{z}\times\mathbf{n}}{|\mathbf{z}\times\mathbf{n}|} = \frac{(-n_y, n_x, 0)}{\sqrt{n_x^2 + n_y^2}} \ ,$$

$$\mathbf{y}' = \mathbf{z}'\times\mathbf{x}' = (n_x, n_y, n_z)\times\frac{(-n_y, n_x, 0)}{\sqrt{n_x^2 + n_y^2}} = \frac{\left(-n_x n_z, -n_y n_z, n_x^2 + n_y^2\right)}{\sqrt{n_x^2 + n_y^2}} \ .$$

Zavedeme-li $\mathbf{x}' = (x'_x, x'_y, x'_z)$, $\mathbf{y}' = (y'_x, y'_y, y'_z)$, $\mathbf{z}' = (z'_x, z'_y, z'_z)$, pak s ohledem na ortonormalitu vyšetřované transformace (podkapitola 1.3, příklad 1.1) máme

$$\mathbf{T}_2 = \begin{bmatrix} x'_x & y'_x & z'_x & 0 \\ x'_y & y'_y & z'_y & 0 \\ x'_z & y'_z & z'_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} .$$

□

Úkol 1.9

Úkol 1.9. Spusťte si program „úkol1_9.exe“. Program předpokládá zadání zobrazení tak, jak bylo popsáno v této podkapitole, které čte ze souboru „úkol1_9.txt“. Program počítá a vypisuje dílčí matice $\mathbf{T}_1$ až $\mathbf{T}_4$ a také výslednou transformační matici, která je jejich součinem. Na konkrétních hodnotách si tak můžete ověřit, zda jste vše správně pochopili. V případě potřeby můžete zadání modifikovat (jedná se o textový soubor). □

SHRNUTÍ

Shrnutí: V této poněkud kratší podkapitole jste se naučili konstruovat matici realizující středové promítání, a to na základě takových zadávacích prvků, které jsou v grafických systémech v praxi obvyklé. V následujícím textu budeme pokračovat dalšími (složitějšími) případy.

1.6 Stanovení matice zobrazovací transformace – příklad B

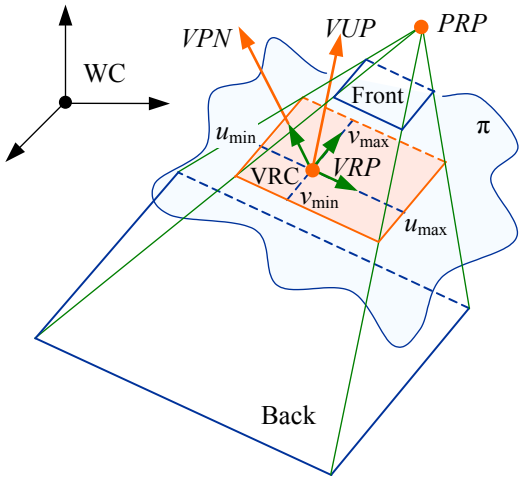
Doba studia asi 2 hod

Příklad B

Také tento příklad bude ještě věnován ukázce toho, jak lze složitější transformaci sestavit z transformací elementárních. Opět sestavíme matici středového promítání na základě velmi praktického zadání. Je dáno následující (obr. 1.7, 1.8).

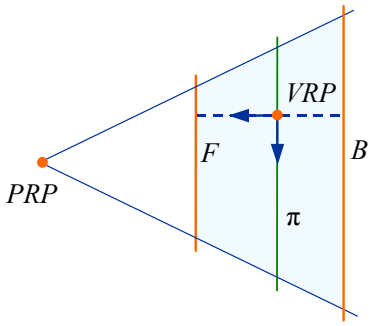
$VRP = (vrp_x, vrp_y, vrp_z)$	(View Reference Point,	souřadnice ve WC),
$VPN = (vpn_x, vpn_y, vpn_z)$	(View Plane Normal,	souřadnice ve WC),
$VUP = (vup_x, vup_y, vup_z)$	(View Up Vector,	souřadnice ve WC),
$PRP = (prp_x, prp_y, prp_z)$	(Projection Reference Point	souřadnice ve VRC),
$Window = (u_{min}, u_{max}, v_{min}, v_{max})$	(Zorné pole	souřadnice ve VRC),
F, B	(Front, Back: přední a zadní ořezávací rovina, jedná se o dvě vzdálenosti vzhledem k VRC).	

WC (world coordinate system) je souřadná soustava, která je zavedena v zobrazované scéně. Výše uvedeným zadáním je dále definována souřadná soustava obrazu VRC (view reference coordinate system), která umožňuje specifikaci dalších zadávacích prvků zobrazení. *VRP* je počátek souřadné soustavy VRC. Poloha počátku *VRP* je zadána ve WC. Bodem *VRP* také prochází průmětna, která splývá s rovinou *xy* soustavy VRC. Poloha průmětny (a tedy také roviny *xy* soustavy VRC) je dodefinována svojí normálou *VPN* (view plane normal). Souřadnice vektoru *VPN* se zadávají v souřadné soustavě WC. Osa *x* souřadné soustavy VRC je kolmá jednak k vektoru *VPN* (který udává směr osy *z* soustavy VRC) a dále k vektoru *VUP* (view up vector). Vektor *VUP* je vektor, který se má na obrázku jevit svisle (kolmo k ose *x* obrazu). Souřadnice vektoru *VUP* se zadávají ve WC. *PRP* (projection reference point) je střed projekce. Poloha středu se zadává v souřadné soustavě VRC. Hodnoty u_{min}, u_{max}, v_{min}, v_{max} na osách *x*, *y* souřadné soustavy VRC definují v průmětně obdélník omezující obraz (jen obraz ležící uvnitř tohoto obdélníka bude získán). Uvedený obdélník spolu s bodem *PRP* definuje zorný jehlan. Zorný jehlan je dále omezen rovinami „front“ a „back“. Roviny jsou zadány svými vzdálenostmi *F*, *B* (opatřenými znaménkem) od průmětny (obr. 1.8). Zorný jehlan je tedy jehlanem komolým (obr. 1.7). Pouze objekty (nebo jejich části) ležící uvnitř zorného jehlanu budou zobrazeny.



Projekce je zadána prvky VRP, VPN, VUP, PRP, u_{min}, u_{max}, v_{min}, v_{max}, F, B.

Obr. 1.7. Prvky zadávající uvažované středové promítání.



Obr. 1.8. Prvky zadávající promítání - dvojrozměrné schéma.
(Hodnoty F, B jsou včetně znaménka. Zde je $F > 0$, $B < 0$).

Je možné, že se vám právě popsaný způsob zadání bude zpočátku zdát dosti komplikovaný. Časem jej ale určitě shledáte velmi logickým. Je pravděpodobné, že pokud byste sami měli vymyslet způsob zadání obecné projekce, vymysleli byste něco velmi podobného. Není také nezajímavé říci, že popsaný způsob zadání je využíván grafickým standardem PHIGS.

Transformaci sestavíme ze šesti transformací jednoduchých.

Řešení. Hledanou transformaci sestavíme ze šesti elementárních transformací, jejichž transformační matice již umíme zapsat. Všechny dále podrobně popíšeme.

1) Posunutí počátku souřadné soustavy scény do *VRP*. Odpovídající matice má tvar

Posunutí počátku souřadné soustavy do VRP

$$\mathbf{T}_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -vrp_x & -vrp_y & -vrp_z & 1 \end{bmatrix}.$$

2) Natočení souřadné soustavy tak, aby osa z splynula s *VPN*. Vyjádříme bázové vektory **x'**, **y'**, **z'** soustavy po natočení vzhledem k soustavě před natočením. Dostaneme

Rotace

$$\mathbf{z}' = \left(z'_x, z'_y, z'_z\right) = \frac{VPN}{|VPN|}, \quad \mathbf{x}' = \left(x'_x, x'_y, x'_z\right) = \frac{VUP \times VPN}{|VUP \times VPN|},$$
$$y' = \left(y'_x, y'_y, y'_z\right) = \mathbf{z}' \times \mathbf{x}'.$$

Matice rotace pak má tvar

$$\mathbf{T}_2 = \begin{bmatrix} x'_x & y'_x & z'_x & 0 \\ x'_y & y'_y & z'_y & 0 \\ x'_z & y'_z & z'_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

3) Posunutí počátku souřadné soustavy z *VRP* do *PRP*. Matice popisující tuto transformaci má tvar (povšimněte si, že se při konstrukci matice hodí, že souřadnice středu projekce *PRP* známe v soustavě VRC)

Posunutí počátku do PRP

$$\mathbf{T}_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -prp_x & -prp_y & -prp_z & 1 \end{bmatrix}.$$

Zorný jehlan učiníme symetrickým.

4) Provedeme zkosení tak, aby zorný jehlan byl symetrický (obr. 1.9): Zaveďme bod *CV* (center of view – střed obrazu) a vektor *DOP* (direction of projection) takto (souřadnice *CV* a *PRP* zde měříme v soustavě VRC)

$$CV = \left(\frac{u_{\max} + u_{\min}}{2}, \frac{v_{\max} + v_{\min}}{2}, 0 \right), \quad DOP = \left(dop_x, dop_y, dop_z \right) = CV - PRP .$$

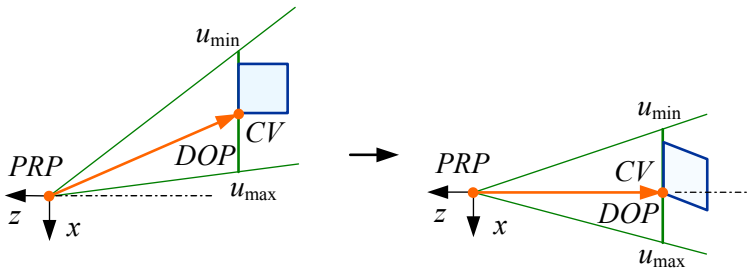
Vektor *DOP* lze interpretovat také jako afinní souřadnice bodu *CV* v souřadné soustavě mající počátek v bodě *PRP* zavedené předchozími třemi transformačními kroky). Matice realizující požadované zkosení má tvar

$$\mathbf{T}_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \frac{-dop_x}{dop_z} & \frac{-dop_y}{dop_z} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} .$$

Správnost matice $\mathbf{T}_4$ můžeme snadno ověřit. Transformujme maticí $\mathbf{T}_4$ vektor (dop_x, dop_y, dop_z, 1) (tedy bod *CV*). Dostaneme

$$(dop_x, dop_y, dop_z, 1)\mathbf{T}_4 = (0, 0, dop_z, 1) .$$

Vidíme, že po transformaci bod *CV* (střed obrazu) leží na ose *z*. Zorný jehlan je tedy nyní symetrický vzhledem k souřadnicovým rovinám *zx* i *yz*.



Obr. 1.9. K činnosti transformace popsané maticí $\mathbf{T}_4$.

Rozměry zorného jehlanu změníme na jednotkové.

5) Transformujeme velikost zorného jehlanu tak, aby měl jednotkové rozměry (obr. 1.10). Požadujeme, aby jeho větší podstava ležela v rovině $z = -1$ a aby délka její strany byla 2 (od −1 do +1 v obou směrech). Jedná se o změnu měřítka, a proto bude mít matice realizující uvedenou transformaci tvar

$$\mathbf{T}_5 = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} .$$

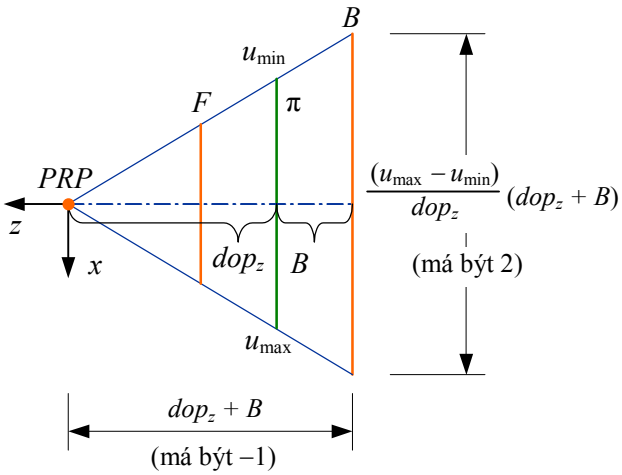
Hodnoty s_x, s_y, s_z stanovíme jednoduše na základě následujících skutečností: Vzdálenost mezi středem projekce a větší podstavou je nyní (dop_z+*B*), ale má být −1 (poznamenejme, že dop_z i *B* jsou obvykle záporné). Označme q_x, q_y rozměr

větší postavy jehlanu ve směru osy x, resp. y. Po transformaci mají být obě tyto šířky 2. Na základě podobnosti trojúhelníků máme (obr. 1.10)

$$q_x = \frac{u_{\max} - u_{\min}}{dop_z}(dop_z + B), \quad q_y = \frac{v_{\max} - v_{\min}}{dop_z}(dop_z + B) .$$

Nyní již snadno určíme

$$s_z = \frac{-1}{(dop_z + B)}, \quad s_x = \frac{2dop_z}{(u_{\max} - u_{\min})(dop_z + B)}, \quad s_y = \frac{2dop_z}{(v_{\max} - v_{\min})(dop_z + B)} .$$



Obr. 1.10. K činnosti transformace popsané maticí $\mathbf{T}_5$.

6) Nakonec provedeme projekci. Abychom usnadnili následující výpočty (zejména výpočet viditelnosti metodou „z-buffer“), provedeme projekci tak, že scénu transformujeme (deformujeme) tak, aby rovnoběžné promítání dalo tentýž výsledek jako původně zamýšlené promítání středové (obr. 1.11). Místo středového promítání pak použijeme promítání rovnoběžného s promítacím paprskem rovnoběžným s osou z, které lze realizovat velmi snadno. Uvedená transformace transformuje zorný jehlan na hranol. Zaveďme hodnotu

Středové promítání převedeme na rovnoběžné.

$$z_f = s_z(dop_z + F) .$$

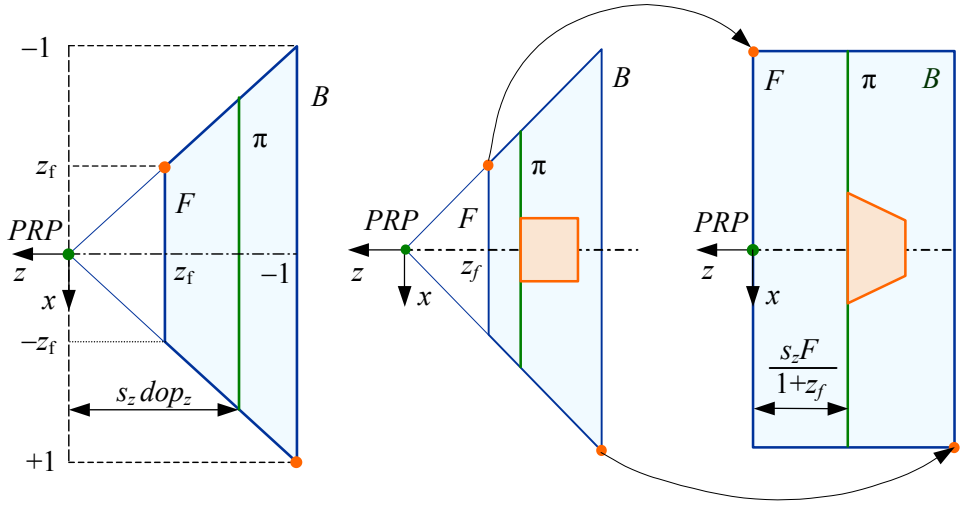
Dále zaveďme matici

$$\mathbf{T}_6 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \dfrac{1}{1+z_f} & -1 \\ 0 & 0 & \dfrac{-z_f}{1+z_f} & 0 \end{bmatrix} .$$

Snadno ověříme, že matice $\mathbf{T}_6$ skutečně transformuje zorný jehlan na hranol. Můžeme to provést tak, že transformaci aplikujeme na některé význačné body zorného jehlanu. Např. (promyslete, které body jehlanu jsme transformovali):

$$(z_f, z_f, z_f, 1)\mathbf{T}_6 = (z_f, z_f, \frac{z_f}{1+z_f} - \frac{z_f}{1+z_f}, -z_f) \to (-1,-1,0,1),$$

$$(1,1,-1,1)M_6 = (1,1,\frac{-1}{1+z_f} - \frac{z_f}{1+z_f}, 1) \to (1,1,-1,1).$$



Obr. 1.11. K činnosti transformace popsané maticí $\mathbf{T}_6$. Situace před transformací (obrázek vlevo). Transformace zorného jehlanu na kvádr (obrázek vpravo).

Nechť $\mathbf{x}$, $\tilde{\mathbf{x}}$ značí vektory souřadnic bodu před a po provedení zobrazovací transformace zadané v tomto příkladě. V prvním případě se tedy jedná o souřadnice ve „světové“ souřadné soustavě scény, ve druhém případě o souřadnice v jednotkovém zobrazovacím hranolu, ke kterému jsme právě popsaným postupem dospěli. Platí

$$\tilde{\mathbf{x}} = \mathbf{x}\mathbf{T}_1\mathbf{T}_2\mathbf{T}_3\mathbf{T}_4\mathbf{T}_5\mathbf{T}_6 = \mathbf{x}\mathbf{T} \ , \tag{1.14}$$

kde je $\mathbf{T} = \mathbf{T}_1\mathbf{T}_2\mathbf{T}_3\mathbf{T}_4\mathbf{T}_5\mathbf{T}_6$. Poznamenejme ještě, že transformace a matice $\mathbf{T}_1$, $\mathbf{T}_2$, $\mathbf{T}_3$ jsou ortonormální. Transformace $\mathbf{T}_4$, $\mathbf{T}_5$, $\mathbf{T}_6$ nikoli. □

Úkol 1.10

Úkol 1.10. Spusťte si program „úkol1_10.exe“. Program předpokládá zadání zobrazení tak, jak bylo popsáno v této podkapitole, které čte ze souboru „úkol1_10.txt“. Program počítá a vypisuje dílčí matice $\mathbf{T}_1$ až $\mathbf{T}_6$ a také výslednou transformační matici, která je jejich součinem. Na konkrétních hodnotách si tak můžete ověřit, zda jste vše správně pochopili. V případě potřeby můžete zadání modifikovat (jedná se o textový soubor). Protože tento způsob zadání promítání použijete ve své práci, můžete program použít také pro kontrolu výsledků. □

S H R N U T Í

Shrnutí: Seznámili jste se s dalším příkladem, v němž byla matice zobrazovací transformace sestavena z dílčích matic transformací elementárních. Postup popsaný v tomto příkladě je velmi praktický a bude vám současně sloužit jako návod pro sestavení matice projekce v programu, který budete realizovat jako svoji samostatnou práci. Ujistěte se proto, zda všem krokům dokonale rozumíte.

1.7 Stanovení matice zobrazovací transformace – příklad C

Doba studia asi 1 hod

příklad C

Zadávacích prvků z příkladu B zde použijeme k zadání rovnoběžného promítaní.

V tomto příkladě ukážeme, že velmi podobně, jako jsme v předchozím případě B realizovali promítání středové, lze realizovat i promítání rovnoběžné. Promítání bude opět realizováno rovnicí (1.13). Drobné odchylky budou pouze ve stanovení matice **T**. Provedeme následující: Prvky popsané v předchozím příkladě použijeme k zadání rovnoběžného promítání. Bod *PRP* spolu s bodem *CV* (viz. řešení příkladu B) v tomto případě určují směr promítacích paprsků. Význam ostatních prvků zůstává beze změny. Stanovíme matici realizující takto zadané promítání.

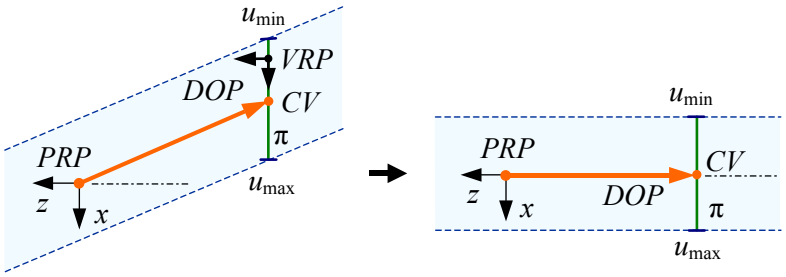
Řešení. Hledanou výslednou matici **T** opět stanovíme jako součin matic realizujících jednotlivé dílčí transformace. I v tomto případě bude

$$\mathbf{T} = \mathbf{T}_1\mathbf{T}_2\mathbf{T}_3\mathbf{T}_4\mathbf{T}_5\mathbf{T}_6 \ .$$

K vyřešení úlohy postačí jen drobné modifikace postupu z příkladu B.

Dílčí transformace 1) až 4) zůstávají zcela beze změny jako v příkladě B a beze změny zůstávají i jim odpovídající matice. Transformace 4) v tomto případě zajišťuje, aby promítací paprsky byly kolmé k průmětně (obr. 1.12). Transformace 5) převádí zorný kvádr na kvádr s jednotkovými rozměry. Jednotlivá měřítka v matici $\mathbf{T}_5$ se nyní stanoví takto (obr. 1.13)

$$s_x = \frac{2}{u_{\max} - u_{\min}}, \quad s_y = \frac{2}{v_{\max} - v_{\min}}, \quad s_z = -\frac{1}{dop_z + B} \ .$$



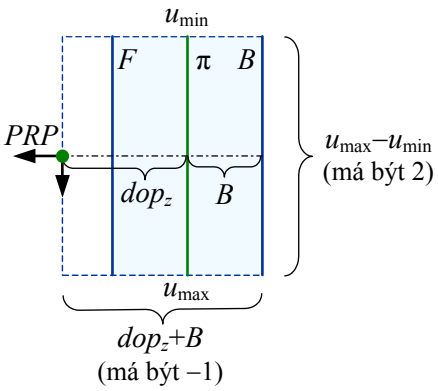
Obr. 1.12. K transformaci reprezentované maticí $\mathbf{T}_4$.

Transformace 6) zajišťuje, aby přední rovina jednotkového zobrazovacího hranolu ležela v rovině $z = 0$ (až dosud ležela v rovině $z = z_f$) (obr. 1.14). Matice $\mathbf{T}_6$ má tvar

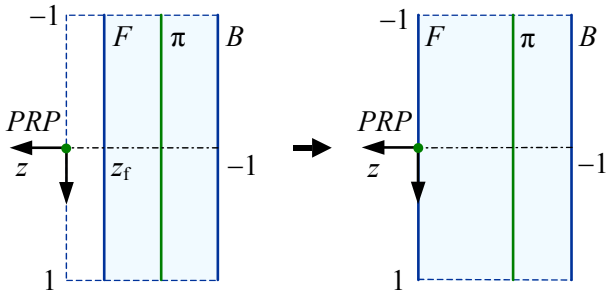
$$\mathbf{T}_6 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \dfrac{1}{1+z_f} & -1 \\ 0 & 0 & \dfrac{-z_f}{1+z_f} & 0 \end{bmatrix} .$$

Podobně jako v předchozím příkladě můžeme i nyní matici prověřit tak, že transformaci aplikujeme na některé významné body zorného kvádru. Snadno např. zjistíme, že je

$(1,1,z_f,1)\mathbf{T}_6 = (1,1,0,1)$ a také $(1,1,-1,1)\mathbf{T}_6 = (1,1,-1,1)$.



Obr. 1.13. K transformaci reprezentované maticí $\mathbf{T}_5$.



Obr. 1.14. K transformaci reprezentované maticí $\mathbf{T}_6$.

□

Příklad 1.11

Příklad 1.11. Sestavte matici realizující rovnoběžné promítání, které je zadáno bodem *P*, kterým prochází průmětna, a normálou **n** průmětny.

Řešení. Se znalostmi, které nyní již máte, by pro vás nalezení matice nemělo být problémem. Napovíme ale, že pomocí transformací popsaných maticemi $\mathbf{T}_1$, $\mathbf{T}_2$ z příkladu A v podkapitole 1.5 lze požadovanou projekci převést na projekci paprsky rovnoběžnými s osou *z* souřadné soustavy. □

SHRNUTÍ

Shrnutí: Oba příklady (příklad B a C), které jste právě vyřešili vám ukazují, že mezi realizací středového a rovnoběžného promítání není v grafických systémech zapotřebí činit žádných rozdílů. Oba případy lze realizovat vztahem (1.13) pro projektivní transformaci a liší se navzájem jen v konkrétních hodnotách matice **T**. Příklady nám také ukázaly, že postup stanovení matice **T** je si v obou případech velmi podobný.

1.8 Transformace na výstupní zařízení

Doba studia asi 0,5 hod

Při studiu příkladů B, C v předcházejících podkapitolách jste se mohli podivit, proč se zobrazovací transformace prováděla tak, aby souřadnice objektů, které mají být zobrazeny, padly do jednotkového zobrazovacího objemu. Mohli jste namítnout, že při vykreslování získaného obrazu na nějaké výstupní zařízení nebude pravděpodobně možné souřadnice z intervalu $\langle -1,1\rangle$ použít. Tyto pochybnosti byly na místě. Po zobrazovací transformaci do jednotkového zobrazovacího objemu totiž ještě následuje transformace na výstupní zařízení, kterou se v této podkapitole naučíte provádět.

K čemu transformace na výstupní zařízení slouží?

Čemu se zde naučíte?

Ačkoli je závěrečná transformace na výstupní zařízení z teoretického pohledu pouze jednoduchou afinní transformací, bývá často oddělena od předchozích transformačních kroků. Důvody jsou následující: 1) předpokládá se, že může být více výstupních zařízení. Všechna pak mohou použít téhož výpočtu zobrazení, jen v závěrečné fázi při transformaci na výstupní zařízení se výpočet pro jednotlivá zařízení liší. 2) Předpokládá se, že závěrečnou část zobrazení obsluha mění častěji než části předchozí (např. prováděním výřezu atd.). V takovém případě se pak opakuje (pokud možno) pouze výpočet závěrečné části zobrazovací transformace, který je jednoduchý, a proto rychlý. Pro pořádek ještě dodejme, že z použití normalizovaného zorného objemu vyplývají i jisté výhody při provádění ořezávání, jak ukážeme v kapitole 2.7.

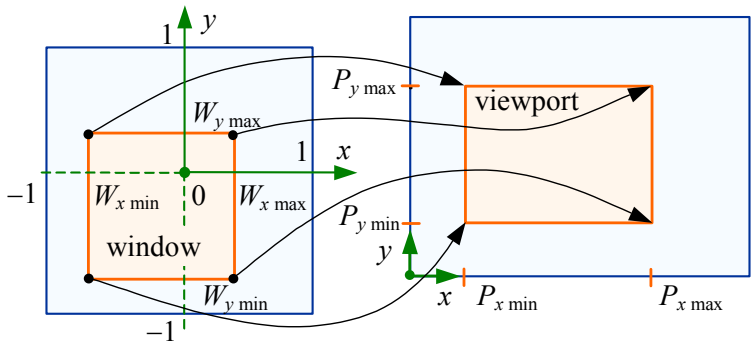
Teoreticky se jedná o afinní transformaci.

Bývá však oddělena od transformací předchozích.

Transformace na výstupní zařízení bývá zadána pomocí okna v normalizovaném zorném objemu. Toto okno říká, která část normalizovaného objemu se má zobrazit. (Poznamenejme, že v jednodušším případě lze zobrazit celý rozsah zorného objemu, okno pak není zapotřebí.) ViewPort naopak určuje oblast na výstupním zařízení, do které se má požadovaný obrázek umístit. Pro transformaci na výstupní zařízení tedy zpravidla známe následující hodnoty (obr. 1.15)

$$Window = \left(W_{x\min}, W_{y\min}, W_{x\max}, W_{y\max}\right),$$
$$ViewPort = \left(P_{x\min}, P_{y\min}, P_{x\max}, P_{y\max}\right).$$

Transformace bývá zadána pomocí okna a „viewportu“.



Obr. 1.15. K transformaci na výstupní zařízení.

Nechť x', y', z' jsou souřadnice na výstupním zařízení a $\tilde{x}, \tilde{y}, \tilde{z}$ souřadnice v normalizovaném zorném objemu. Pro transformaci souřadnic na výstupní zařízení pak na základě úměry jednoduše máme

$$x' = P_{x\min} + \frac{\tilde{x} - W_{x\min}}{W_{x\max} - W_{x\min}}\left(P_{x\max} - P_{x\min}\right),$$

$$y' = P_{y\min} + \frac{\tilde{y} - W_{y\min}}{W_{y\max} - W_{y\min}}\left(P_{y\max} - P_{y\min}\right),$$

$$z' = \tilde{z}\,.$$

Poznamenejme, že souřadnici z' potřebujeme, má-li být řešena viditelnost. Předpis pro její výpočet může být stanoven s jistou volností. Musí však zůstat zachováno pořadí bodů a objektů, jak se jeví ve směru od pozorovatele. Výše uvedený jednoduchý postup tomuto kriteriu vyhovuje. Dále poznamenejme, že někdy je požadováno nastavit okno tak, aby byly zobrazeny všechny objekty scény. V takovém případě je pak nutné nejprve vypočítat zobrazovací transformaci pro všechny objekty a pak nalézt extrémní hodnoty souřadnic x, y v normalizovaném zorném objemu, které pak budou použity pro specifikaci okna.

1.9 Inverze zobrazovací transformace – příklad D

Doba studia asi 0,5 hod

Provádět inverzní transformaci k transformaci zobrazovací je zapotřebí např. při výpočtu Phongova stínování nebo při nanášení textury. V těchto případech je totiž nutné pro daný bod na výstupním zařízení (pixel obrazu) zjistit polohu jeho vzoru ve scéně (nebo jinak: je zapotřebí zjistit, který bod „skutečného světa“ se promítl do daného bodu obrazu). Detaily použití této úlohy podrobněji popíšeme později v kapitolách 2.10, 2.11. Na tomto místě si pro pozdější použití připravíme inverzní transformaci k transformaci, která je složena ze zobrazovací transformace z podkapitoly 1.6 a z transformace na výstupní zařízení podle předchozí podkapitoly 1.8.

K čemu je inverzní transformace dobrá?

Pospícháte-li k praktickým aplikacím, můžete studium této podkapitoly zatím odložit. Vrátíte se k ní později.

Řešení. Nechť x', y', z' jsou souřadnice na výstupním zařízení a $\tilde{x}, \tilde{y}, \tilde{z}$ souřadnice v normalizovaném zobrazovacím hranolu zavedeném v podkapitole 1.6. Inverzí vztahů z podkapitoly 1.8 snadno získáme

$$\tilde{x} = W_{x\min} + \frac{x' - P_{x\min}}{P_{x\max} - P_{x\min}}\left(W_{x\max} - W_{x\min}\right),$$

$$\tilde{y} = W_{y\min} + \frac{y' - P_{y\min}}{P_{y\max} - P_{y\min}}\left(W_{y\max} - W_{y\min}\right),$$

$$\tilde{z} = z'\,.$$

Inverze k transformaci na zařízení

Nechť $\mathbf{x}$ je vektor souřadnic v souřadné soustavě scény. Inverzí předpisu pro zobrazovací transformaci z příkladu v podkapitole 1.6 dostaneme

$$\mathbf{x} = \tilde{\mathbf{x}}\mathbf{T}_6^{-1}\mathbf{T}_5^{-1}\mathbf{T}_4^{-1}\mathbf{T}_3^{-1}\mathbf{T}_2^{-1}\mathbf{T}_1^{-1}\,.$$

Inverze transformace ze vztahu (1.14)

Matice $\mathbf{T}_1$ až $\mathbf{T}_6$ byly již uvedeny v podkapitole 1.6. Inverzní matice mají tvar

$$\mathbf{T}_1^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ vrp_x & vrp_y & vrp_z & 1 \end{bmatrix}, \quad \mathbf{T}_2^{-1} = \mathbf{T}_2^{\mathrm{T}}, \quad \mathbf{T}_3^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ prp_x & prp_y & prp_z & 1 \end{bmatrix},$$

Inverze matic $\mathbf{T}_1$ *až* $\mathbf{T}_6$

$$\mathbf{T}_4^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \dfrac{dop_x}{dop_z} & \dfrac{dop_y}{dop_z} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{T}_5^{-1} = \begin{bmatrix} 1/s_x & 0 & 0 & 0 \\ 0 & 1/s_y & 0 & 0 \\ 0 & 0 & 1/s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$\mathbf{T}_6^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -\dfrac{1+z_f}{z_f} \\ 0 & 0 & -1 & -\dfrac{1}{z_f} \end{bmatrix}.$$

Poznamenejme, že ačkoli je uvedené matice možné získat „čistě mechanickou“ inverzí původních matic $\mathbf{T}_1$ až $\mathbf{T}_6$, může být často pohodlnější zapsat je jako matice provádějící jednotlivé dílčí transformační kroky „v opačném směru“, než byl směr transformací odpovídajících maticím původním. Například: Jestliže matice $\mathbf{T}_1$ popisuje posunutí počátku souřadné soustavy scény do bodu *VRP*, bude matice $\mathbf{T}_1^{-1}$ popisovat posunutí z bodu *VRP* zpět do původního počátku. Při praktické realizaci grafického systému byste samozřejmě také mohli invertovat až výslednou matici $\mathbf{T}$ ze vztahu (1.14). Použili byste k tomu obvyklých algoritmů pro výpočet inverzní matice, které určitě znáte z matematiky. □

Příklad 1.12

Příklad 1.12. Stanovte inverzní transformaci k zobrazovací transformaci z příkladu C v podkapitole 1.7.

Řešení. Využijeme výsledků z předchozího příkladu D. Všechny vztahy zůstávají v platnosti kromě matice $\mathbf{T}_6^{-1}$. Zapište tuto matici. □

SHRNUTÍ

Shrnutí: Na tomto místě už byste měli vědět vše, co z teorie afinních a projektivních prostorů a transformací budete ke konstrukci grafických systémů potřebovat. Víte, že větší systémy pracují velmi často tak, že nejprve provedou zobrazovací transformaci do jednotkového zorného objemu tak, jak jsme to i my provedli v příkladech B nebo C. Pak následuje transformace na nějaké konkrétní výstupní zařízení, jak jsme ukázali v podkapitole 1.8. Víte, že grafické systémy zobrazování realizují jako projektivní transformaci podle vztahu (1.13). Umíte již také sestavit matici, která požadovanou projektivní transformaci realizuje. Víte, že je obvyklé sestavit požadovanou transformaci z transformací dílčích. Matice výsledné hledané transformace je pak vypočtena jako součin matic, které popisují použité dílčí transformace. Pokud je vám toto vše jasné, můžete přistoupit ke studiu následující kapitoly, která se zabývá konstrukcí zobrazovacího řetězce.

1.10 Dodatky ke kapitole o afinních prostorech a afinních transformacích

Doba studia asi 1 hod

V této závěrečné podkapitole o afinních a projektivních prostorech a transformacích uvedeme nakonec ještě slíbené definice, kterým jsme se v předchozím textu záměrně pokusili vyhnout. Podkapitola je určena zejména pro teoretičtěji zaměřené čtenáře, kteří by snad mohli zjednodušení v předchozím textu pociťovat rušivě. Je také současně ale možné, že i pro ostatní by nyní mohlo být zajímavé vědět, jak jsou přesně definovány pojmy, s nimiž jsme v kapitole pracovali. Pokud ale takovou informaci nepostrádáte, můžete podkapitolu přeskočit.

Studium této podkapitoly není povinné

Afinní prostor (definice): Neprázdnou množinu $\mathbf{A}_n$ nazveme afinním prostorem dimenze n (prvky množiny $\mathbf{A}_n$ budeme nazývat body), jestliže je dán vektorový prostor $\mathbf{V}_n$ dimenze n a zobrazení $\varphi:\mathbf{A}_n\times\mathbf{A}_n \rightarrow \mathbf{V}_n$, které má následující dvě vlastnosti:

Afinní prostor (definice)

- Pro každý bod $A\in\mathbf{A}_n$ a pro každý vektor $\mathbf{v}\in\mathbf{V}_n$ existuje jediný bod $B\in\mathbf{A}_n$ tak, že $\varphi(A, B) = \mathbf{v}$.
- Pro každé tři body $A, B, C\in\mathbf{A}_n$ platí, že $\varphi(A, C) = \varphi(A, B) + \varphi(B, C)$.

Poznámka: Přesně vzato je afinní prostor uspořádaná trojice $(\mathbf{A}_n, \mathbf{V}_n, \varphi)$. Často se však používá pouze stručného zápisu $\mathbf{A}_n$. Vektorový prostor $\mathbf{V}_n$ bývá nazýván vektorovým zaměřením afinního prostoru $\mathbf{A}_n$.

Afinní zobrazení (definice): Nechť $\mathbf{A}_m$, $\mathbf{B}_n$ jsou dva afinní prostory dimenzí m, n a $\mathbf{V}_m$, $\mathbf{W}_n$ nechť jsou jejich vektorová zaměření. Zobrazení $f:\mathbf{A}_m\rightarrow\mathbf{B}_n$ nazveme afinním zobrazením, jestliže existuje takový lineární operátor $\mathcal{A}:\mathbf{V}_m\rightarrow\mathbf{W}_n$, že pro každý bod $M\in\mathbf{A}_m$ a pro každý vektor $\mathbf{v}\in\mathbf{V}_m$ platí, že $f(M+\mathbf{v})=f(M)+\mathcal{A}(\mathbf{v})$.

Afinní zobrazení (definice)

Projektivní prostor (definice): Nechť $\mathbf{V}_{n+1}$ je vektorový prostor dimenze $n+1$. Množinu všech směrů ve $\mathbf{V}_{n+1}$ nazveme projektivním prostorem dimenze n (a označíme $\langle\mathbf{V}_{n+1}\rangle$ nebo $\mathbf{P}_n$).

Projektivní prostor (definice)

Homogenní souřadnice: Homogenními souřadnicemi v projektivním prostoru $\mathbf{P}_n$ nazveme uspořádanou $(n+1)$-tici, která určuje směr v prostoru $\mathbf{V}_{n+1}$. Každá taková $(n+1)$-tice určuje jednoznačně nějaký bod v $\mathbf{P}_n$. Naopak každý bod může být reprezentován více (nekonečně mnoha) takovými $(n+1)$-ticemi. Nechť $\mathbf{x}\in\mathbf{V}_{n+1}$ je vektor reprezentující bod X, pak také vektor $\lambda\mathbf{x}$, $\lambda\neq0\in\mathbf{R}$, reprezentuje X.

Projektivní zobrazení (definice): Nechť $\mathbf{P}_n=\langle\mathbf{V}_{n+1}\rangle$, $\mathbf{P'}_n=\langle\mathbf{V'}_{n+1}\rangle$ jsou dva projektivní prostory a nechť f je izomorfismus $\mathbf{V}_{n+1}$ na $\mathbf{V'}_{n+1}$. Zobrazení $F:\mathbf{P}_n\rightarrow\mathbf{P'}_n$ je kolineací, jestliže pro každý bod $X\in\mathbf{P}_n$ a pro každého vektorového zástupce $\mathbf{x}\in\mathbf{V}_{n+1}$ tohoto bodu platí, že $F(X)$ je bod reprezentovaný vektorem $f(\mathbf{x})$.

Projektivní zobrazení (definice)

Úkol 1.13

Úkol 1.13. Nejsou-li vám definice úplně proti mysli, můžete se nyní např. zamyslet nad tím, proč je afinní prostor definován právě tak, jak jsme uvedli. Co by se stalo, kdyby nebyly splněny ony dvě vlastnosti zobrazení φ, které definice požaduje?

2 Standardní zobrazovací řetězec

Co je standardní zobrazovací řetězec?

V této kapitole popíšeme zobrazovací techniku, která klade důraz na rychlost. Posloupnost kroků, kterou je zobrazení v tomto případě realizováno, budeme nazývat „standardním zobrazovacím řetězcem“. Takový název je skutečně oprávněný. S popisovanou technikou se dnes totiž setkáte v široké škále programů jako jsou například CAD programy, počítačové hry a letecké simulátory. Tato technika je realizována např. také v grafických standardech OpenGL a Direct3D a v posledních letech se jí rovněž dostává intenzivní hardwarové podpory od výrobců grafických karet (např. tak, že karta podporuje standard OpenGL). Obr. 2.1 je ukázkou výstupu standardního řetězce.

Obr. 2.1. Ukázka obrázku vytvořeného standardním zobrazovacím řetězcem (se svolením Martina Cvíčka, SPŠ stavební Ostrava).

Čemu se v této kapitole naučíte?

Jak při studiu kapitoly postupovat?

Zobrazovací řetězec je v této kapitole popsán poměrně velmi podrobně. Po jejím prostudování budete umět řetězec prakticky realizovat. Praktická realizace je pro absolvování předmětu dokonce i vyžadována. Po dohodě s vyučujícím budete realizovat buď vybrané klíčové části řetězce nebo i řetězec celý (podle vašeho zájmu a možností). Studium kapitoly doporučujeme začít prohlídkou ukázek programů a obrázků. Jako ukázky jsou záměrně přiloženy dřívější práce studentské, které poskytují obraz o tom, jaké úrovně byste měli při studiu a také při pozdější realizaci samostatné práce dosáhnout. Po prohlédnutí ukázek se pusťte do studia.

Kapitola je rozdělena do 11 podkapitol s předpokládanou celkovou dobou studia přibližně 10 hodin. Kapitola vyžaduje, abyste již měli prostudovánu předchozí kapitolu o afinních a projektivních prostorech.

Jak realizovat samostatnou práci?

Chcete-li, pak můžete už během studia kapitoly také současně realizovat svoji samostatnou práci. Povedou vás úkoly v jednotlivých podkapitolách. Při realizaci samostatné práce doporučuji vyjít ze vzorového programu „shader“. S ohledem na způsob, jak jí budete používat, bude tato vzorová realizace zobrazovacího řetězce dále nazývána *šablonou*. Jedná se o fungující program, který můžete přeložit, sestavit a spustit. Vaše práce bude spočívat v tom, že vybrané dílčí úlohy, z nichž se zobrazovací řetězec skládá, budete postupně realizovat sami a v šabloně budete postupně nahrazovat její jednotlivé části svými produkty. Může tak nakonec vzniknout program, který jste vytvořili celý sami. Použití šablony je výhodné jednak proto, že vám usnadní ladění vašich produktů (prostě se jednoduše podíváte, zda šablona stále ještě funguje správně i po té, co jste původní řešení některého kroku nahradili svým vlastním řešením), a také proto, že vás šablona při práci velmi přesně povede, což by jinak, zejména při distančním studiu, nebylo možné. Šablona je k dispozici v jazyce C. Vaše samostatná práce na standardním zobrazovacím řetězci bude vyžadovat přibližně 25 hodin času.

Úkol 2.1

Úkol 2.1. Prohlédněte si programy a obrázky, které naleznete na přiloženém CD v adresáři „grafikaII\standard\obrázky” a „grafikaII\standard\student_programy”. Seznamte se také s obsahem adresáře „grafikaII\šablony”, kde v podadresáři „shader“ naleznete vzorový program (šablonu), s nímž budete později pracovat. □

2.1 Popis scény, osvětlení a požadovaného zobrazení

Doba studia asi 0,25 hod

Čemu se zde naučíte?

K tomu, abyste mohli vytvořit obrázek, musíte mít následující: 1) popis scény, která má být zobrazena, 2) popis osvětlení scény (jaká světla a kde jsou umístěna), 3) popis zobrazení (jak se na scénu chcete dívat). Po přečtení této podkapitoly získáte přibližnou představu, z čeho se jednotlivé uvedené části zadání skládají. Tato předběžná představa je nezbytná, abyste mohli sledovat další výklad. Podrobnější údaje a samozřejmě také popis samotného zobrazovacího postupu uvedeme v následujících podkapitolách.

Popis scény

Aproximace povrchů těles ploškami

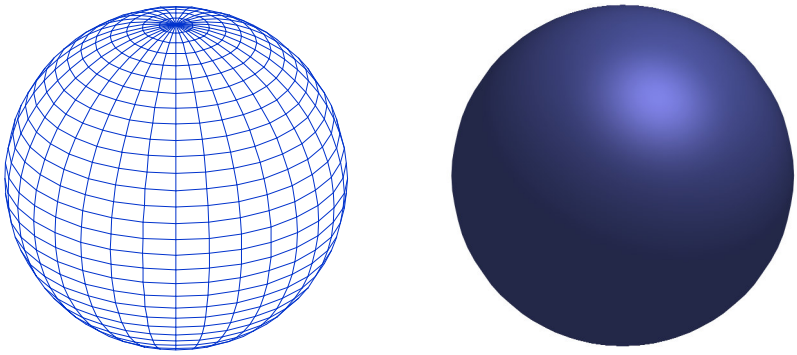
Scénou nazýváme množinu objektů v nějakém prostoru. Předpokládáme, že scénu máme v počítači vhodným způsobem popsánu. Problematice popisu scény se podrobněji věnuje odvětví počítačové grafiky nazývané „modelování těles“ (někdy také „objemové modelování“). Detaily však pro nás v tuto chvíli nejsou podstatné. Zde postačí, když uvedeme, že zobrazovací postup, o němž mluvíme, vyžaduje, aby bylo možné povrchy těles ve scéně popsat jako množinu rovinných plošek (obr. 2.2). To je obvykle možné. Jistě to není problém, obsahuje-li scéna tělesa s rovinnými stěnami. Je to ale možné i tehdy, když jsou stěny těles zakřivené, protože i ty lze přiměřeně velkými rovinnými ploškami s přijatelnou přesností aproximovat. Činnost zobrazovacího řetězce, který zde popisujeme, bude směřovat k tomu, aby fakt, že jsou zakřivené plochy takto aproximovány, před pozorovatelem zamaskoval. Není se proto nutné obávat toho, že by snad plošky použité k aproximaci musely být extrémně malé.

Zadání osvětlení

Bodový zdroj světla

Zdroj rovnoběžných paprsků

Ohledně osvětlení se na tomto místě spokojíme s konstatováním, že k tomu, aby na později vytvořeném obrázku bylo vůbec něco vidět, musí být scéna osvětlena. Za základní lze považovat tzv. bodový světelný zdroj, který si lze představit jako žárovku, která je umístěna v prostoru scény a která vyzařuje všemi směry světlo jisté barvy a intenzity. Dále se často také používají zdroje rovnoběžných paprsků. Takový zdroj si můžeme představit buď jako bodový zdroj umístěný v nekonečnu nebo jako rovinu (případně její část), z níž rovnoběžné paprsky vystupují (z praktického života víme, že zařízení realizující takový abstraktní model skutečně existují). Kromě uvedených dvou základních typů světelných zdrojů existují i zdroje poněkud speciálnější povahy (např. různé reflektory). O těch ale budeme podrobněji mluvit až později, v podkapitole o výpočtu osvětlení. Obvykle je možné scénu osvětlit z více světelných zdrojů. Lze také kombinovat různé typy zdrojů. Zadání osvětlení spočívá v tom, že se stanoví typ a umístění světelných zdrojů ve scéně, jejich barva a intenzita a případné další parametry, které jsou pro specifikaci zvoleného konkrétního světelného zdroje potřebné.



Obr. 2.2. Aproximace povrchu tělesa ve scéně ploškami ve tvaru čtyřúhelníků (vlevo) a výsledný obraz (vpravo).

Zadání požadovaného zobrazení

Na to, jak lze zadat požadavky na způsob zobrazení scény, jste se důkladně připravili již v předchozí kapitole. Řešené příklady v podkapitolách 1.5, 1.6 a 1.7 ukazují možné způsoby zadání. Pro každé zadání je v příkladech odvozen také odpovídající matematický předpis, pomocí něhož lze provést transformaci, kterou zadání definuje. Toho také v této kapitole využijeme.

2.2 Posloupnost kroků standardního řetězce

Doba studia asi 0,25 hod

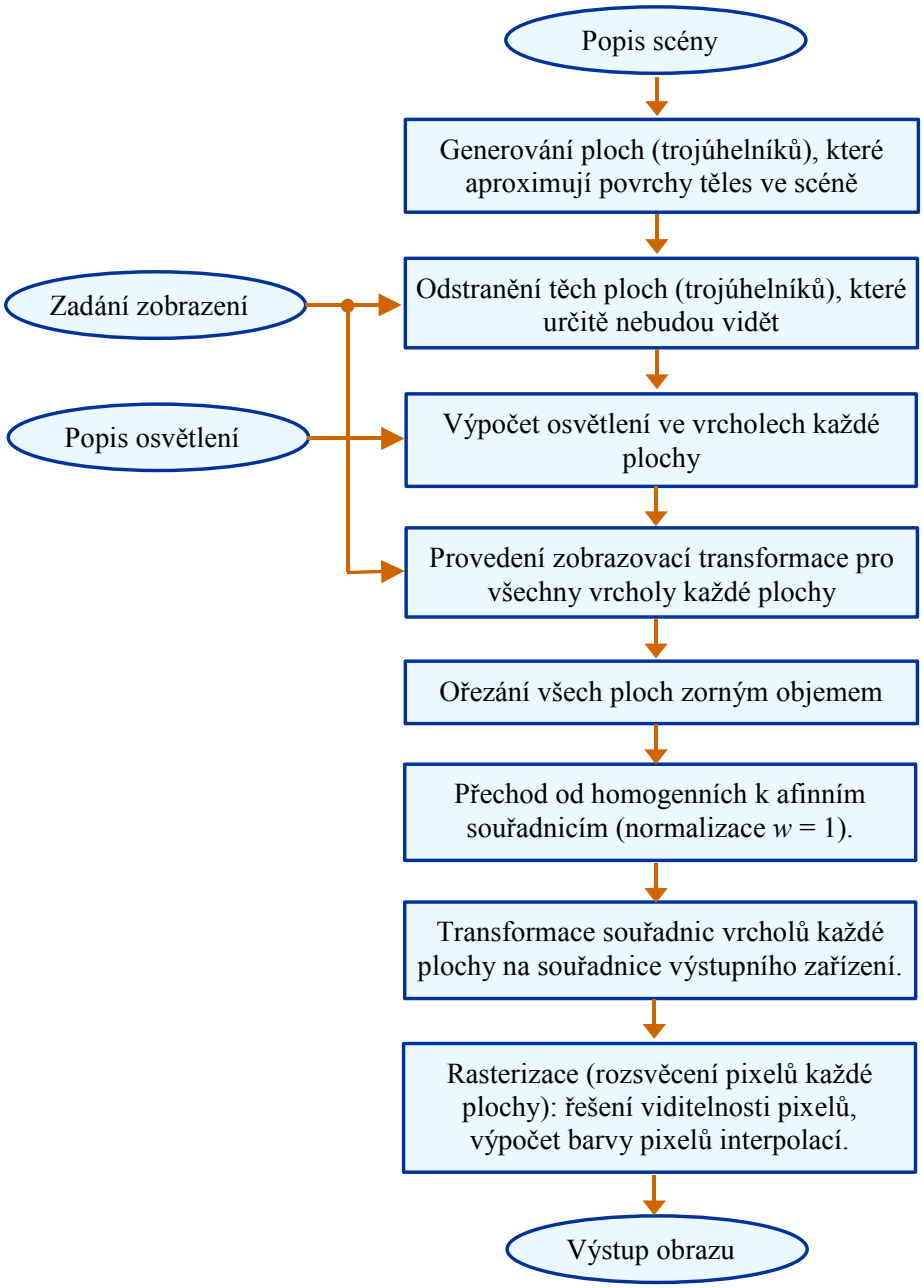
Čemu se zde naučíte?

V této podkapitole se dovíte, jaké kroky a v jakém pořadí zobrazovací řetězec provádí (ve stejném pořadí pak budeme jednotlivé kroky ve zbytku kapitoly postupně probírat).

Schéma zobrazovacího řetězce je uvedeno na obr. 2.3. Je nutné poznamenat, že se struktura řetězce může do jisté míry měnit podle toho, jaký typ stínování je při zobrazení požadován a s jakými plochami (případně s jakými jejich vyššími celky) řetězec pracuje. Schéma uvedené na obr. 2.3 předpokládá použití tzv. Gouraudova stínování, které se používá nejčastěji, protože je jednodušší a

probíhá rychle. Proto se nejprve zaměříme právě na tuto variantu. Modifikace řetězce, které jsou nutné při použití tzv. Phongova stínování, uvedeme později v podkapitole 2.10. (Při použití Phongova stínování sice trvá zobrazení déle, ale zato poskytuje hezčí obrázky. Pro pořádek poznamenejme, že obrázek 2.2. byl získán stínováním Phongovým. Obrázek získaný pro tentýž objekt stínováním Gouraudovým je na obr. 2.24.)

Nejprve se zaměříme na variantu s Gouraudovým stínováním.



Obr. 2.3. Schéma zobrazovacího řetězce při použití Gouraudova stínování a trojúhelníkových ploch pro aproximaci povrchů těles scény.

Úkol 2.2

Úkol 2.2. Otevřete si soubor „main.cpp“ v adresáři „grafikaII\šablony\shader“, který obsahuje zdrojový kód vzorového programu (šablony) realizující právě probíraný způsob zobrazování. Seznamte se se strukturou programu. Mělo by to být jednoduché, protože struktura přesně odpovídá schématu z obr. 2.3. Seznamte se s tím, jak je zadávána scéna, osvětlení a požadované zobrazení. Příklad naleznete např. v souboru „scene.dat“ v adresáři „grafikaII\šablony\bin“ (a také v dalších souborech s příponou „dat“). Můžete se ale také podívat, jak je zadání provedeno v ukázkových programech, které jste si prohlíželi v úkolu 2.1.

S H R N U T Í

Shrnutí: V tuto chvíli byste měli mít hrubou orientační představu o tom, co standardní zobrazovací řetězec dělá a jaké ke své činnosti potřebuje vstupní údaje. Je-li tomu tak, pak jste připraveni na to, abyste se pustili do detailního studia jednotlivých kroků řetězce.

2.3 Generování plošek aproximujících povrchy těles scény

Doba studia asi 0,5 hod

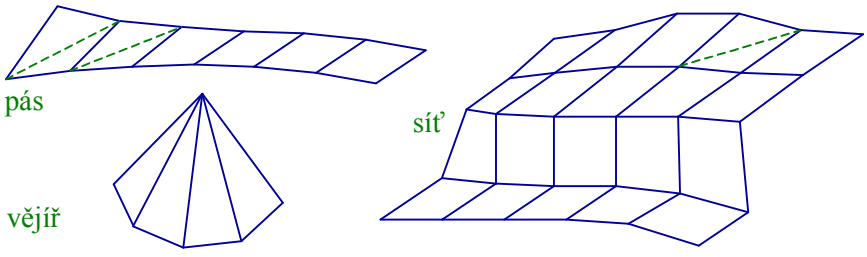
Čemu se zde naučíte?

V této podkapitole uvedeme detailnější informace o tom, jak se provádí aproximace povrchu tělesa ploškami. Na základě těchto informací budete pak také úlohu řešit ve své samostatné práci.

Reprezentace povrchů těles ploškami

Seskupování plošek do vyšších celků

Plošky, které jsou použity k aproximaci povrchů těles scény, jsou zpravidla trojúhelníky nebo čtyřúhelníky. My se při výkladu pro jednoduchost omezíme zejména na plošky trojúhelníkové. Někdy jsou plošky organizovány do vyšších celků (např. pásů, vějířů a sítí – obr. 2.4), což přináší úsporu paměti a také úsporu výpočetního času. V tomto textu se však (opět pro jednoduchost) této možnosti vzdáme a budeme předpokládat, že pro provedení zobrazení je povrch každého tělesa reprezentován seznamem trojúhelníků, které povrch aproximují. V této souvislosti poznamenáváme, že i v případě použití zmíněných vyšších celků se i ony na jednotlivé trojúhelníky nakonec vždy rozdělují, a to nejpozději před tzv. rasterizací (tj. „rozsvěcením“ jednotlivých obrazových bodů), často ale i dříve, např. před ořezáním zorným objemem (všechny právě použité termíny později podrobně vysvětlíme). Také poznamenáváme, že se k problému organizace plošek do vyšších celků ještě vrátíme v úkolu 2.16.

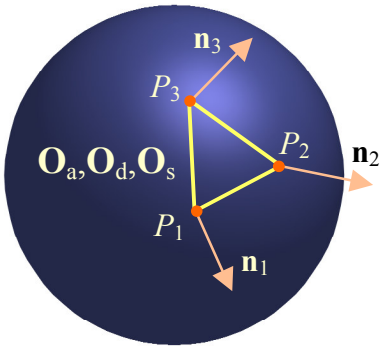


Obr. 2.4. Pás čtyřúhelníků, vějíř trojúhelníků a síť čtyřúhelníků. Čárkovaně je u některých čtyřúhelníků vyznačeno rozdělení na trojúhelníky, které se v systémech dříve či později vždy provádí. Kromě jiného se tak odstraní i problém spočívající v tom, že čtyřúhelníky nemusí být rovinné.

Informace potřebné pro zobrazení trojúhelníkové plošky

Příprava scény k zobrazení bude tedy v našem případě spočívat v tom, že pro každé těleso vygenerujeme množinu trojúhelníků, které povrch tělesa aproximují. Abychom pak plošky mohli zobrazit, budeme ke každému trojúhelníku potřebovat souřadnice jeho tří vrcholů. Vrcholy musí být specifikovány v jistém pořadí, které určuje směr normálového vektoru plošky (obr. 2.5) (Normálový vektor orientujeme zvoleným způsobem, např. tak, aby směřoval ven z tělesa. Pořadí vrcholů tomu pak musí odpovídat a musí být stejné u všech trojúhelníků). Z hlediska implementace je výhodné zadávat souřadnice vrcholů od samého začátku v homogenních souřadnicích. K tomu, jak víte z předchozí kapitoly, postačí doplnit známé hodnoty x, y, z souřadnic afinních čtvrtou souřadnicí $w = 1$. Dále budeme ve vrcholech trojúhelníka potřebovat normálové vektory původní plochy, kterou trojúhelník aproximuje (obr. 2.5). Normálové vektory budou reprezentovány tříprvkovými vektory (použití homogenních souřadnic není pro reprezentaci vektorů oprávněné). Konečně bude také nutné znát ke každému trojúhelníku i jeho vlastnosti (materiálové charakteristiky), které ve výsledném obrázku určují jeho vzhled. Vysvětlení, o jaké vlastnosti se jedná a jak se zadávají, ale ponecháme až do podkapitoly o výpočtu osvětlení.

Prvky popisující trojúhelníkovou plošku



Obr. 2.5. Každá trojúhelníková ploška aproximující povrch tělesa je popsána svými vrcholy P_1, P_2, P_3, normálovými vektory $\mathbf{n}_1$, $\mathbf{n}_2$, $\mathbf{n}_3$ aproximované plochy v místech vrcholů plošky, vektorem $\mathbf{O}_a$ koeficientů odrazu pro rozptýlené světlo a vektory $\mathbf{O}_d$, $\mathbf{O}_s$ koeficientů pro difúzní a zrcadlový odraz od světelných zdrojů. Vektory koeficientů odrazu jsou zpravidla stejné pro celé těleso (podrobněji vysvětlíme význam vektorů koeficientů odrazu v podkapitole 2.5).

Úkol 2.3

Úkol 2.3. Otevřete si soubor „solid.h“ v adresáři „šablony\shader“ a prohlédněte si, jaký záznam se pro reprezentaci trojúhelníka používá ve vzorovém programu (třída Triangle). Scéna je pak během zobrazování reprezentována jako seznam takových záznamů. Jednotlivé kroky zobrazovacího řetězce jsou realizovány jako operace s tímto seznamem.

Úkol 2.4

Úkol 2.4. Vytvořte funkci (metodu), která přečte popis scény ze souboru a vygeneruje seznam aproximujících trojúhelníkových plošek. Způsob popisu scény i způsob generování volte co nejjednodušší. Také repertoár možných těles volte malý (koule, válec, kužel). Nahraďte touto funkcí metodu „scene.triangulate“, která je použita v šabloně. Zkontrolujte, zda vaše funkce správně funguje.

2.4 Předběžné odstranění určitě neviditelných ploch

Doba studia asi 0,5 hod

Čemu se zde naučíte?

Plochy ohraničující nějaké těleso lze často pozorovat jen z jedné strany, a to z vnějšku tělesa, protože se předpokládá, že těleso je plné, tvořené neprůhledným materiálem. Pro úsporu času potřebného pro provedení zobrazení lze pak ty plochy povrchu tělesa, které jsou odvrácené od pozorovatele, předem ze zpracování vyloučit. Úspora času, kterou provedení tohoto kroku přináší, není zdaleka zanedbatelná. V obvyklých případech lze takto předem vyloučit přibližně polovinu ploch. Po prostudování této podkapitoly budete vědět, jak to provést.

Rozhoduje znaménko skalárního součinu $\mathbf{n}\cdot\mathbf{v}$.

Určit plochy, které mají být vyloučeny, je snadné (často se uvedený krok nazývá „*trivial rejection*“). Předpokládejme, že hranice každého trojúhelníka je zadána jako orientovaná (jak jsme již ostatně uvedli v předchozí podkapitole). Tím myslíme, že vrcholy jsou zadávány v jistém pořadí. Řekněme např. tak, že posloupnost P_1, P_2, P_3 při pohledu z vnějšku tělesa „točí“ kladně (proti směru hodinových ručiček). Střed projekce nechť je C a X nechť je libovolný bod uvnitř nebo na hranici uvažovaného trojúhelníka. O vyloučení plochy lze snadno rozhodnout na základě znaménka skalárního součinu normály $\mathbf{n}$ plochy a směru $\mathbf{v}$ paprsku vedeného ze středu projekce C k bodu X. Máme (obr. 2.6)

$$\operatorname{sgn}(\mathbf{n}\cdot\mathbf{v}) = \operatorname{sgn}([(P_2 - P_1)\times(P_3 - P_1)]\cdot(X - C)) \,. \qquad (2.1)$$

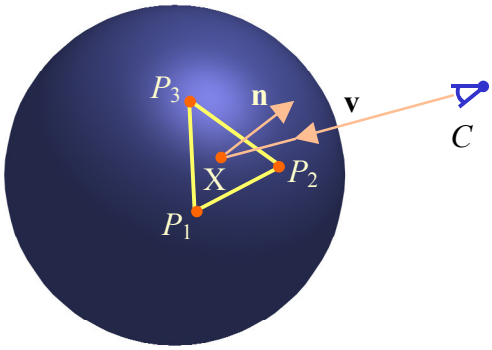
Vyšetřovanou plochu lze z dalšího zpracování vyloučit tehdy, jestliže je znaménko součinu kladné. Protože má normála plochy v tomto případě stejný směr jako je směr „pozorovacího paprsku“, musel by paprsek projít tělesem, což nepředpokládáme. Výsledek testu nezáleží na volbě bodu X. Bod X může být libovolný bod uvnitř nebo na obvodu trojúhelníka. Výpočetně je nejsnazší, když je bodem X některý (libovolný) z jeho vrcholů. Možné je také použít těžiště trojúhelníka, jehož poloha je dána vztahem $T = (P_1 + P_2 + P_3)/3$. To lze však doporučit spíše jen pro představu. Výpočetně se jedná o zbytečnou komplikaci. Na závěr ještě poznamenejme, že v případě zadání zobrazení podle příkladu z podkapitoly 1.6, které je také použito ve vzorovém programu, je zapotřebí dát pozor na to, že scéna je popsána v souřadné soustavě scény, zatímco střed projekce je zadán v souřadné soustavě obrazu. Je tedy proto nejprve nutné určit jeho souřadnice v souřadné soustavě scény. Mělo by pro vás být jednoduchým opakováním ověřit, že pro přepočet platí vztah $\mathbf{x} = \tilde{\mathbf{x}}\mathbf{T}_2^{-1}\mathbf{T}_1^{-1}$, kde $\mathbf{x}$ je vektor souřadnic v souřadné soustavě scény, $\tilde{\mathbf{x}}$ vektor souřadnic v souřadné soustavě obrazu a matice $\mathbf{T}_1$, $\mathbf{T}_2$ jsou matice zavedené v příkladu 1.6 (úkol 2.5).

Úkol 2.5

Úkol 2.5. Uvažujte zadání zobrazení z příkladu 1.6. Nechť $\mathbf{x}$ je vektor souřadnic v souřadné soustavě scény, $\tilde{\mathbf{x}}$ vektor souřadnic v souřadné soustavě obrazu a $\mathbf{T}_1$, $\mathbf{T}_2$ nechť jsou matice zavedené v příkladu 1.6. Zdůvodněte, že pro přepočet souřadnic ze souřadné soustavy scény do souřadné soustavy obrazu platí vztah $\tilde{\mathbf{x}} = \mathbf{x}\mathbf{T}_1\mathbf{T}_2$ a pro přepočet ve směru opačném vztah $\mathbf{x} = \tilde{\mathbf{x}}\mathbf{T}_2^{-1}\mathbf{T}_1^{-1}$.

Úkol 2.6

Úkol 2.6. Realizujte funkci, která provádí předběžné odstranění ploch, a nahraďte touto funkcí původní funkci použitou v šabloně. Šablonu sestavte, spusťte a přesvědčete se, zda vaše funkce funguje správně.



Obr. 2.6. K předběžnému odstranění neviditelných ploch. Jestliže plocha aproximuje hranici neprůhledného tělesa, pak může být vidět jedině tehdy, jestliže směr pozorování **v** a směr normály **n** plochy „jdou proti sobě“. V opačném případě prochází „pozorovací paprsek“ nejprve vnitřkem tělesa, což se u neprůhledných těles nepovažuje za možné.

2.5 Výpočet osvětlení

Doba studia asi 1,5 hod

Čemu se zde naučíte?

Úkolem výpočtu osvětlení je stanovit, jak se jednotlivé body scény jeví pozorovateli (jedná se o barevný vjem) a jak by tedy proto měly být vykresleny ve vytvářeném obraze. V této podkapitole se dovíte, jak výpočet provést.

Phongův model osvětlení

Základní vztah Phongova modelu

Vjem pozorovatele závisí na způsobu osvětlení scény a na optických vlastnostech povrchů těles. V případě, který nyní podrobně diskutujeme, tedy při použití tzv. Gouraudova stínování se výpočet osvětlení provádí pouze ve vrcholech trojúhelníků. V praxi je nejčastěji používanou technikou výpočtu osvětlení výpočet podle tzv. Phongova modelu (nezaměňovat s Phongovým stínováním, o němž budeme mluvit později). Phongův model osvětlení sice nerespektuje příliš přesně fyzikální zákony šíření a odrazu světla, je ale jednoduchý a obrazy, které poskytuje, vypadají velmi přijatelně, což platí zejména tehdy, jestliže mají zobrazovaná tělesa matné hladké povrchy. Model je založen na představě, že je-li scéna osvětlena, dopadají paprsky vycházející ze světelných zdrojů do jednotlivých bodů povrchů těles scény a odtud se odrážejí. Odražené paprsky pak dávají vzniknout vjemu pozorovatele. Intenzita $\hat{I}_\lambda$ vjemu se počítá pomocí vztahu

$$\hat{I}_\lambda = I_{a\lambda} O_{a\lambda} + \sum_i I_{\lambda_i} f_{att_i} \left(O_{d\lambda} \cos\varphi_i + O_{s\lambda} \cos^n \alpha_i \right). \qquad (2.2)$$

Výpočet obvykle probíhá v barevných složkách R, G, B.

Jednotlivé členy v tomto vztahu postupně probereme. Nejprve ale vysvětleme, že hodnoty opatřené indexy λ jsou závislé na vlnové délce světla (λ značí vlnovou délku). Prakticky výpočet obvykle probíhá pro tři základní barevné složky R, G, B. Potřebné vztahy lehce získáme přepsáním vztahu (2.2). Pro výpočet červené složky $\hat{I}_R$ vjemu (pro červenou složku budeme místo λ psát R) například dostáváme (pro zbývající barevné složky vztahy již nepíšeme, protože je očividné, že je lze snadno získat pouhou záměnou indexů)

$$\hat{I}_{\mathrm{R}} = I_{\mathrm{aR}} O_{\mathrm{aR}} + \sum_i I_{\mathrm{R}_i} f_{\mathrm{att}_i} \left(O_{\mathrm{dR}} \cos\varphi_i + O_{\mathrm{sR}} \cos^n \alpha_i \right). \qquad (2.3)$$

Popis barvy a intenzity světla a vlastností povrchů těles

Povšimněte si, že všechny členy v sumách na pravé straně vztahů (2.2), (2.3) obsahují součin obecného tvaru $I_\lambda O_\lambda$. Vysvětleme nejprve obecně, proč tomu tak je. Hodnota I_λ je intenzita světelného zdroje. Např. trojice $I_\mathrm{R} = 10$, $I_\mathrm{G} = 10$, $I_\mathrm{B} = 10$ definuje bílé světlo intenzity 10. Vjem pozorovatele ale závisí na tom, kolik světla (z množství, které na dané místo povrchu dopadlo ze světelných zdrojů) se odráží z povrchu tělesa zpět do prostoru. Míra odrazu povrchu tělesa je popsána hodnotou O_λ, která se nazývá koeficient odrazu. Je samozřejmé předpokládat, že platí $0 \le O_\lambda \le 1$. Koeficient odrazu lze považovat za charakteristiku materiálu povrchu tělesa. Pomocí koeficientů odrazu je zadána barva povrchu tělesa. Jako příklad uveďme, že trojice $O_\mathrm{R} = 0$, $O_\mathrm{G} = 0$, $O_\mathrm{B} = 1$ definuje povrch barvy modré a trojice $O_\mathrm{R} = 1$, $O_\mathrm{G} = 1$, $O_\mathrm{B} = 1$ definuje povrch barvy bílé (uvedené barvy vnímáme, jestliže je scéna osvětlena bílým světlem). Trojice hodnot popisujících intenzitu světelných zdrojů a koeficienty odrazu můžeme pro stručnost uspořádat také do vektorů. Příklady uvedené dříve v tomto odstavci tak můžeme zapsat stručně ve tvaru $\mathbf{I} = (10, 10, 10)$, $\mathbf{O} = (0, 0, 1)$ nebo $\mathbf{O} = (1, 1, 1)$.

Význam hodnot $I_{\mathrm{a}\lambda}$, $O_{\mathrm{a}\lambda}$

Po všeobecném zdůvodnění tvaru jednotlivých členů ve vztahu (2.2) diskutujme nyní jednotlivé veličiny ve vztahu podrobně. $I_{a\lambda}$ je intenzita světla rozptýleného ve scéně (ambient light), které považujeme za všudypřítomné a svítící všemi směry. Tento fiktivní světelný zdroj a jemu odpovídající první člen ve vztahu (2.2) jsou zavedeny proto, aby v místech, kam přímo nesvítí ostatní světelné zdroje, nedával výpočet nulovou hodnotu vjemu pozorovatele. Taková místa by se pak v obraze jevila jako zcela černá. Právě zavedený fiktivní světelný zdroj současně také neodporuje naší každodenní zkušenosti. Ta říká, že když scénu (řekněme vnitřek místnosti) osvětlíme světelným zdrojem (např. žárovkou) budou do jisté míry vždy osvětlena i ta místa, kam žárovka přímo nesvítí. To je způsobeno mnohočetnými odrazy světelných paprsků od stěn místnosti a také od dalších předmětů v ní. Phongův osvětlovací model nebere tento jev v úvahu tak, že by přesně sledoval, jak se paprsky vycházející z jednotlivých světelných zdrojů ve scéně odrážejí. Místo toho zavádí právě diskutovanou intenzitu „všeobecného“ osvětlení $I_{\mathrm{a}\lambda}$, která se při zobrazování scény zadává jako vstupní údaj. Hodnota $I_{\mathrm{a}\lambda}$ se zpravidla jednoduše „uhodne“ a podle potřeby pak případně poopraví tak, aby výsledný obrázek vypadal pěkně. Hodnota $O_{\mathrm{a}\lambda}$ je odpovídající koeficient odrazu pro toto osvětlení. Dosti často se bere $O_{\mathrm{a}\lambda} = O_{\mathrm{d}\lambda}$.

Význam hodnot $I_{\lambda i}, f_{\mathrm{att}i}$

Difúzní a zrcadlový odraz

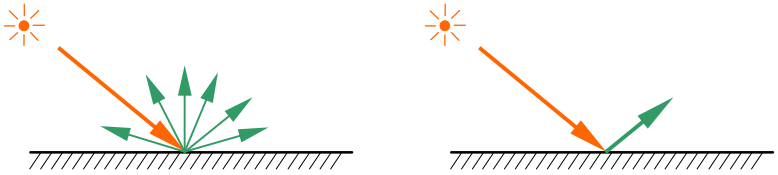
Význam hodnot $O_{\mathrm{d}\lambda}$, $O_{\mathrm{s}\lambda}$, φ_i, α_i, n

Suma ve vztahu (2.2) sčítá účinek všech světelných zdrojů, které jsou ve scéně umístěny. $I_{\lambda i}$ je intenzita i-tého světelného zdroje. Součinitel $f_{\mathrm{att}i}$ vyjadřuje zeslabení intenzity i-tého světelného zdroje v závislosti na vzdálenosti zdroje od místa, v němž je intenzita osvětlení počítána. Předpokládá se, že platí $0 \le f_{\mathrm{att}i} \le 1$. Phongův model osvětlení předpokládá dva druhy odrazu paprsků, které na povrch ze světelných zdrojů dopadly. Jedná se jednak o odraz difúzní, kdy se světelný paprsek po dopadu na povrch tělesa odráží všemi směry (obr. 2.7), a dále o odraz zrcadlový, kdy se dopadnuvší paprsek odrazí podle zákona odrazu známého z fyziky (úhel dopadu je roven úhlu odrazu). Dvěma druhům odrazu odpovídají také dva členy v závorce na pravé straně výrazu (2.2). První počítá vjem způsobený odrazem difúzním, druhý vjem způsobený odrazem zrcadlovým. Hodnoty $O_{\mathrm{d}\lambda}$, $O_{\mathrm{s}\lambda}$ jsou součinitele difúzního resp. zrcadlového odrazu. Opět se jedná o hodnoty, které charakterizují vlastnost povrchu tělesa v místě, kde se osvětlení počítá. Úhel φ_i je

úhel mezi paprskem dopadajícím od *i*-tého světelného zdroje a normálou povrchu; α_i je úhel mezi odraženým paprskem a směrem k pozorovateli (obr. 2.8). Hodnota n určuje „velikost plošky zrcadlového odlesku“ (obr. 2.11). Volí se zpravidla v rozmezí 1 až 100.

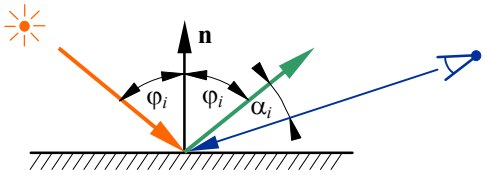
Zrekapitulujme, že z veličin ve vztahu (2.2) jsou hodnoty $I_{a\lambda}$, $I_{\lambda i}$, $O_{a\lambda}$, $O_{d\lambda}$, $O_{s\lambda}$, n zvoleny zadáním. Hodnoty f_{atti}, φ_i, α_i musí být naopak během zobrazení scény vypočítány (způsob výpočtu ukážeme později). Vliv koeficientů odrazu a hodnoty n na výsledný obraz ilustrují obr. 2.9 až 2.11. U Gouraudova stínování se výpočet osvětlení podle vztahů (2.2), (2.3) provádí ve vrcholech plochy (v našem případě trojúhelníka). Při zobrazování plochy jsou ovšem zapotřebí i hodnoty v bodech mezilehlých a ty se získávají interpolací (podkapitola 2.9).

Difúzní a zrcadlový odraz



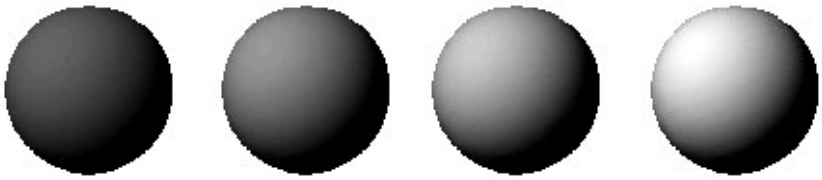
Obr. 2.7. Difúzní (vlevo) a zrcadlový (vpravo) odraz světla.

Význam úhlů φ_i, α_i

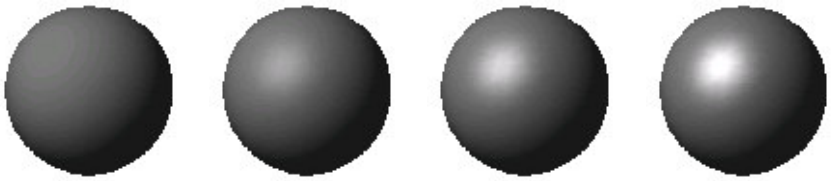


Obr. 2.8. K výpočtu osvětlení: φ_i je úhel, který svírá směr dopadajícího resp. odraženého paprsku s normálou povrchu v místě výpočtu, α_i je úhel, který svírá směr odraženého paprsku se směrem pozorování.

Vliv volby hodnoty $\mathbf{O}_d$ na výsledný obraz

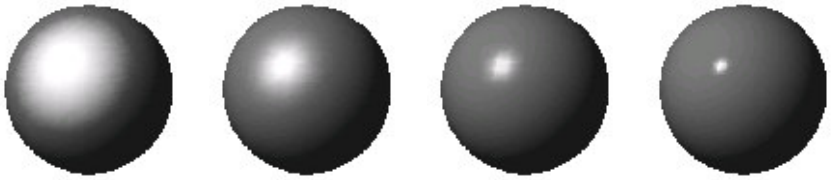


Obr. 2.9. Vliv hodnoty $\mathbf{O}_d$ na výsledný obraz. Hodnoty $\mathbf{I}_a$, $\mathbf{I}_1$, $\mathbf{O}_s$ jsou $\mathbf{I}_a = (0, 0, 0)$, $\mathbf{I}_1 = (1, 1, 1)$ (tj. ve scéně je jediný světelný zdroj intenzity $\mathbf{I}_1$), $\mathbf{O}_s = (0, 0, 0)$. Hodnoty $\mathbf{O}_d$ jsou (zleva doprava) $\mathbf{O}_d = (0.4, 0.4, 0.4)$, $\mathbf{O}_d = (0.6, 0.6, 0.6)$, $\mathbf{O}_d = (0.8, 0.8, 0.8)$, $\mathbf{O}_d = (1, 1, 1)$. (Mělo by být zřejmé, že hodnoty $\mathbf{O}_d$ mění pouze jas objektu.)



Vliv volby $\mathbf{O}_s$ *na výsledný obraz*

Obr. 2.10. Vliv hodnoty $\mathbf{O}_s$ na výsledný obraz. Hodnoty $\mathbf{I}_a$, $\mathbf{I}_1$, $\mathbf{O}_a$, $\mathbf{O}_d$, n jsou $\mathbf{I}_a = (0.1, 0.1, 0.1)$, $\mathbf{I}_1 = (1, 1, 1)$, $\mathbf{O}_a = \mathbf{O}_d = (0.4, 0.4, 0.4)$, $n = 5$. Hodnoty $\mathbf{O}_s$ jsou postupně (zleva doprava) $\mathbf{O}_s = (0, 0, 0)$, $\mathbf{O}_s = (0.2, 0.2, 0.2)$, $\mathbf{O}_s = (0.4, 0.4, 0.4)$, $\mathbf{O}_s = (0.6, 0.6, 0.6)$.



Vliv volby n na výsledný obraz

Obr. 2.11. Vliv hodnoty n na výsledný obraz. Hodnoty $\mathbf{I}_a$, $\mathbf{I}_1$, $\mathbf{O}_a$, $\mathbf{O}_d$, $\mathbf{O}_s$, jsou $\mathbf{I}_a = (0.1, 0.1, 0.1)$, $\mathbf{I}_1 = (1, 1, 1)$, $\mathbf{O}_a = \mathbf{O}_d = (0.4, 0.4, 0.4)$, $\mathbf{O}_s = (0.6, 0.6, 0.6)$. Hodnoty n jsou postupně (zleva doprava) $n = 1$, $n = 5$, $n = 20$, $n = 100$.

Výpočet koeficientu $f_{\mathrm{att}i}$

Doplňme nyní, jak lze spočítat hodnotu koeficientu $f_{\mathrm{att}i}$ zeslabení intenzity i-tého světelného zdroje v závislosti na vzdálenosti. Nechť d_i označuje vzdálenost světelného zdroje od místa, v němž osvětlení počítáme. U bodového světelného zdroje je význam a způsob výpočtu hodnoty d_i zcela zřejmý. Pokud by mělo být zeslabení počítáno také u zdroje rovnoběžných paprsků, lze si takový zdroj představit (jak jsme již uvedli v úvodu této kapitoly) jako rovinu umístěnou v prostoru, která paprsky vyzařuje. Vzdáleností d_i by pak bylo možné rozumět vzdálenost místa, kde osvětlení počítáme, od této roviny. Součinitel zeslabení intenzity světelného zdroje se obvykle uvažuje ve tvaru

$$f_{\mathrm{att}_i} = \min\left\{\frac{1}{c_0 + c_1 d_i + c_2 d_i^2},\quad 1\right\}, \tag{2.4}$$

kde c_0, c_1, c_2 jsou konstanty, které popisují úbytek intenzity světla v uvažovaném prostředí. Jejich hodnotu je nutné zadat. Konstanty jsou zpravidla tytéž pro všechny světelné zdroje, jak ostatně vztah (2.4) naznačuje. To má také rozumnou interpretaci, protože konstanty vlastně popisují vlastnost prostředí („atmosféry“), v němž je scéna umístěna. To je obvykle jediné a pro všechny světelné zdroje stejné (přesto je ale např. v OpenGL možné zadat konstanty pro každý světelný zdroj jiné). Na základě znalostí z fyziky byste možná očekávali, že intenzity bude ubývat nejspíše se čtvercem vzdálenosti d_i. V praxi se však ukázalo, že tak rychlý úbytek obvykle nevede k pěkným obrázkům. Vztah (2.4) byl proto stanoven do značné míry empiricky. Kromě teoreticky očekávaného kvadratického členu byly do jmenovatele výrazu doplněny také člen prostý a lineární. V praxi má právě lineární člen největší význam. Zbývající dva bývají dokonce často zanedbávány, což odpovídá volbě $c_0 = c_2 = 0$. Výpočet minima ve vztahu (2.4) zajišťuje, aby i při malých hodnotách d_i vyšel koeficient $f_{\mathrm{att}i}$ vždy nikoli větší než 1. Někdy se také výpočet zeslabení vůbec nemusí provádět, což odpovídá volbě $f_{\mathrm{att}i} = 1$.

Výpočet úhlů φ_i, α_i

Při praktické aplikaci vztahů (2.2), (2.3) musí být stanoveny velikosti úhlů φ_i, α_i. Ukážeme, že řešení tohoto úkolu není obtížné. Nechť $\mathbf{n}$ je vektor normály plochy, $\mathbf{l}_i$ nechť je vektor směřující od místa, v němž osvětlení počítáme, k i-tému světelnému zdroji a $\mathbf{v}$ nechť je vektor směřující k pozorovateli (obr. 2.12). Předpokládáme, že všechny vektory jsou normalizovány na jednotkovou délku, tj. $|\mathbf{n}| = |\mathbf{l}_i| = |\mathbf{v}| = 1$. To zjednoduší zápis následujících vztahů. Je okamžitě zřejmé, že pro úhel φ_i platí

$$\cos\varphi_i = \mathbf{n}\cdot\mathbf{l}_i \ . \tag{2.5}$$

Dále nechť $\mathbf{q}_i$ je vektor, který je průmětem vektoru $\mathbf{l}_i$ na vektor $\mathbf{n}$ (obr. 2.12). Máme

$$\mathbf{q}_i = \mathbf{n}\cos\varphi_i \ . \tag{2.6}$$

Zaveďme dále vektor

$$\mathbf{s}_i = \mathbf{q}_i - \mathbf{l}_i = \mathbf{n}\cos\varphi_i - \mathbf{l}_i \ . \tag{2.7}$$

Pro směr $\mathbf{r}_i$ odraženého paprsku pak vychází (obr. 2.12)

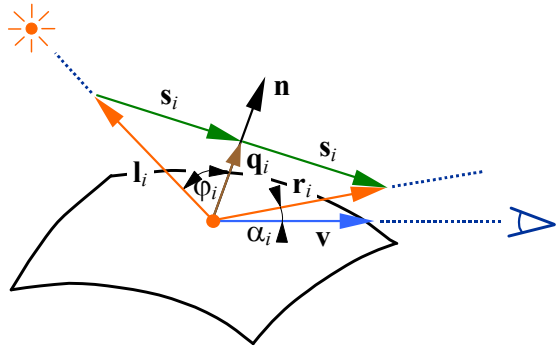
$$\mathbf{r}_i = \mathbf{q}_i + \mathbf{s}_i = 2\mathbf{n}\cos\varphi_i - \mathbf{l}_i = 2\mathbf{n}(\mathbf{n}\cdot\mathbf{l}_i) - \mathbf{l}_i \ . \tag{2.8}$$

Pro hledaný úhel α_i tak konečně můžeme psát

$$\cos\alpha_i = \mathbf{v}\cdot\mathbf{r}_i \ . \tag{2.9}$$

Světelné efekty

Na závěr této podkapitoly o výpočtu osvětlení uvedeme několik poněkud speciálnějších efektů, které lze ale v zobrazovacím řetězci popisovaném v této kapitole realizovat překvapivě snadno. Jedná se o následující: 1) změnu barevného odstínu vlivem atmosféry, 2) realizaci reflektoru vrhajícího do scény kužel světla, 3) realizaci reflektoru vrhajícího do scény „hranol světla“.



Obr. 2.12. K výpočtu úhlů α_i, φ_i.

Vliv vzdálenosti, mlha

Změna barevného odstínu vlivem atmosféry (atmospheric attenuation, depth cueing): Jedná se o modelování dobře známého jevu, že objekty v dálce mají poněkud jinou barvu než objekty blízké. Změnu barvy v závislosti na vzdálenosti od pozorovatele lze provést tak, že se barva $\hat{I}_\lambda$ vypočtená v daném bodě podle vztahu (2.2) v jistém poměru míchá s barvou $I_{DC\lambda}$, což je barva zvolená pro znázornění velmi vzdálených objektů. Teprve barva $\tilde{I}_\lambda$ vzniklá mícháním se použije pro zobrazení bodu. Míchání barev se obvykle provádí podle vztahu (index λ opět naznačuje, že se výpočet provádí po jednotlivých barevných složkách)

$$\tilde{I}_\lambda = \mu\hat{I}_\lambda + (1-\mu)I_{DC\lambda} \ . \tag{2.10}$$

V nejjednodušším případě závisí „poměr míchání“ μ ($0 \le \mu \le 1$) lineárně na vzdálenosti od pozorovatele k bodu, v němž se osvětlení počítá. Typickou situaci ukazuje obr. 2.13. Při malých vzdálenostech mezi pozorovatelem a objektem (menších než hodnota z_f) se míchání vůbec neuplatní, protože hodnota μ je pro tyto vzdálenosti $\mu = 1$, a vychází proto $\tilde{I}_\lambda = \hat{I}_\lambda$. K největšímu uplatnění vlivu atmosféry dochází naproti tomu při vzdálenostech větších než z_b. Pro tyto vzdálenosti totiž μ nabývá nějaké malé kladné hodnoty μ_b, řekněme např. $\mu_b = 0.1$ (při $\mu_b = 0$ by objekty ve vzdálenosti $\ge z_b$ „zcela zmizely“, což zpravidla není žádoucí). Pro vzdálenosti $z_f \le z \le z_b$ lze hodnotu μ získat interpolací pomocí vztahu (obr. 2.13)

$$\mu = \mu_b + \frac{z - z_b}{z_f - z_b}\left(1 - \mu_b\right) . \tag{2.11}$$

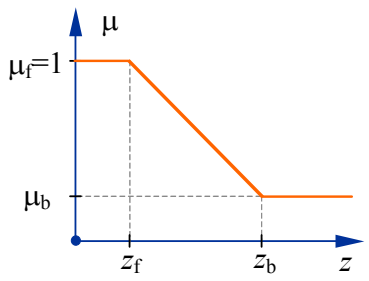
Poznamenejme, že ne vždy ale musí koeficient μ záviset na vzdálenosti lineárně. V OpenGL lze například volit mezi průběhem lineárním a průběhem exponenciálním.

Reflektor vyzařující kužel světla

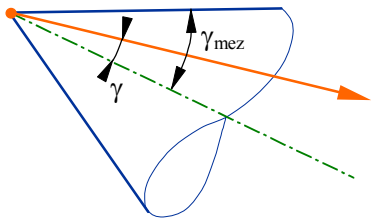
Reflektor vyzařující kužel světla: Reflektor lze velmi jednoduše realizovat tak, že intenzita vyzařovaného paprsku závisí na úhlu γ mezi paprskem a optickou osou reflektoru (obr. 2.14). Lze volit např. jednu z následujících funkcí

$$I_\lambda(\gamma) = \begin{cases} I_\lambda, & \gamma \le \gamma_{mez} \\ 0, & \text{jinak} \end{cases} \quad \text{nebo} \quad I_\lambda(\gamma) = I_\lambda \cos^n \gamma \ . \tag{2.12}$$

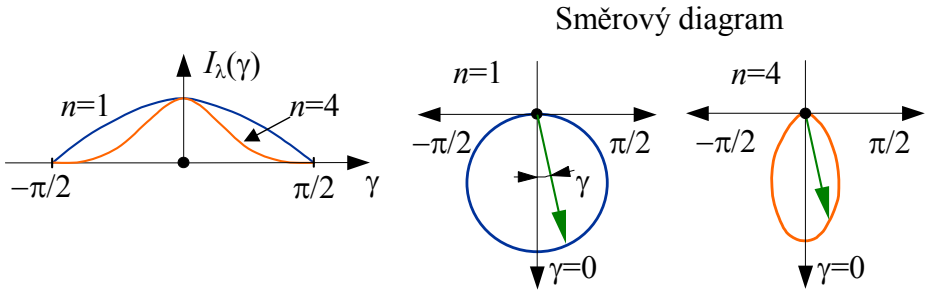
V prvním případě je intenzita všech vyzařovaných paprsků stejná, a to I_λ. Reflektor ale vyzařuje pouze paprsky svírající s osou úhel menší než mezní úhel γ_{mez}. Ve druhém případě se intenzita vyzařovaných paprsků mění. Reflektor září nejsilněji ve směru své osy. S rostoucím úhlem γ intenzity záření ubývá. Rychlost úbytku lze nastavit hodnotou n. Při malých hodnotách n klesá hodnota intenzity (v závislosti na γ) pomalu a reflektor tedy vyzařuje široký kužel světla. Při velkých hodnotách n je kužel naopak úzký. Situaci ilustruje obr. 2.15. Hodnoty n se obvykle pohybují od 1 do přibližně 50.



Obr. 2.13. K výpočtu koeficientu μ.



Obr. 2.14. K realizaci reflektoru.



Obr. 2.15. Vliv n ve funkci $I_\lambda(\gamma) = I_\lambda \cos^n \gamma$ na směrovou charakteristiku reflektoru. Hodnoty $I_\lambda(\gamma)$ v závislosti na γ (obrázek vlevo) pro dvě různé hodnoty n. Tzv. směrové diagramy reflektoru (obrázek vpravo). Délka znázorněných vektorů ukazuje intenzitu paprsku v daném směru.

Reflektor vyzařující hranol světla

Reflektor vyzařující „hranol světla“

Tento reflektor nemusíte studovat, jestliže nechcete.

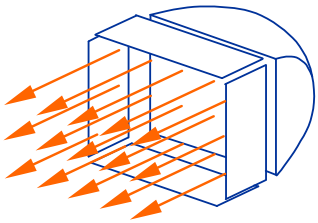
Reflektor vyzařující hranol světla: Nakonec ještě ukážeme příklad modelování speciálních reflektorů, které lze někdy vidět ve fotografických ateliérech. Prostor, který má být osvětlen, lze u takových reflektorů volit pomocí stavitelných „bočnic“ reflektoru (obr. 2.16). Zde se omezíme na případ, kdy reflektor osvětluje prostor ve tvaru hranolu (připusťme takovou idealizaci). Předpokládáme, že je reflektor zadán svým umístěním P (obr. 2.17), směrem vyzařování *DIR*, vektorem *UP* (součin *UP*×*DIR* určuje směr osy x reflektoru), rozměry $x_{\min}$, $x_{\max}$, $y_{\min}$, $y_{\max}$, $z_{\max}$ světelného hranolu a hodnotami barevných složek svého světla. Jediným problémem při výpočtu osvětlení takovým reflektorem je rozhodnout, zda je či není vyšetřovaný bod reflektorem osvětlen. Problém lze snadno řešit transformací souřadnic vyšetřovaného bodu ze souřadné soustavy scény do souřadné soustavy reflektoru. Nechť $\mathbf{x}$ je vektor souřadnic bodu v souřadné soustavě scény. Vektor $\tilde{\mathbf{x}}$ souřadnic v souřadné soustavě reflektoru získáme pomocí vztahu

$$\tilde{\mathbf{x}} = \mathbf{x}\mathbf{T}_1\mathbf{T}_2 \ . \tag{2.13}$$

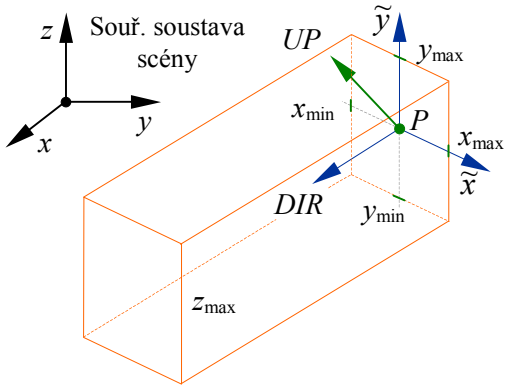
Matice $\mathbf{T}_1$ realizuje posunutí počátku souřadné soustavy scény do bodu P. Matice $\mathbf{T}_2$ realizuje rotaci takto posunuté souřadné soustavy do souřadné soustavy reflektoru. Detailní odvození matic přenecháváme čtenáři jako cvičení, protože se jedná jen o opakování postupu, kterým jsme odvodili matice $\mathbf{T}_1$, $\mathbf{T}_2$ v řešeném příkladu z podkapitoly 1.6. Jestliže obě matice známe, můžeme výpočet podle vztahu (2.13) realizovat. Předpokládejme, že $\tilde{\mathbf{x}} \equiv (\tilde{x}, \tilde{y}, \tilde{z}, 1)$. Je zřejmé, že vyšetřovaný bod leží uvnitř osvětleného hranolu, jestliže platí

$$(x_{\min} \le \tilde{x} \le x_{\max}) \wedge (y_{\min} \le \tilde{y} \le y_{\max}) \wedge (0 \le \tilde{z} \le z_{\max}) \ . \tag{2.14}$$

Výpočet osvětlení pak již spočívá pouze v tom, že na body ležící uvnitř osvětleného hranolu světelný zdroj aplikujeme a na body ležící vně nikoli.



Obr. 2.16. Reflektor vyzařující hranol světla.



Obr. 2.17. Geometrické prvky definující „světelný hranol“. *P* je referenční bod, pomocí něhož je reflektor do scény umístěn, *DIR* je směr paprsků, které reflektor vyzařuje, součin *UP*×*DIR* určuje směr osy *x* reflektoru, hodnoty x_{min}, x_{max}, y_{min}, y_{max}, z_{max} určují rozměry světelného hranolu.

Úkol 2.7

Úkol 2.7. Otevřete si soubor „light.h“ v adresáři „šablony\shader“ a prohlédněte si, jakým způsobem jsou uvnitř programu popsány světelné zdroje. Záznamy pro jednotlivé zdroje jsou organizovány do seznamu, což dovoluje ve scéně definovat teoreticky neomezený počet světelných zdrojů.

Úkol 2.8

Úkol 2.8. Realizujte funkci, která provádí výpočet osvětlení ve vrcholech trojúhelníků a nahraďte touto funkcí původní funkci použitou v šabloně. Přesvědčete se, zda vaše funkce funguje správně.

SHRNUTÍ

Shrnutí: V tuto chvíli byste měli vědět, jak se v jednotlivých bodech na povrchu těles scény počítá hodnota, o níž se předpokládá, že popisuje vjem pozorovatele, a které se často stručně (ale poněkud méně výstižně) říká „hodnota osvětlení“. Měli byste také vědět, že v případě Gouraudova stínování, na které jsme se zatím omezili, se výpočet provádí ve vrcholech ploch, kterými jsou povrchy scény aproximovány.

2.6 Promítání

Doba studia asi 1 hod

Na řešení promítání jste již dobře připraveni z předchozí kapitoly. Ta směřovala právě k řešení úlohy o promítání (kvůli rozsáhlosti a důležitosti tohoto tématu mu ale byla vyhrazena celá samostatná kapitola). Zde proto postačí jen připomenout, že úloha se obvykle řeší jednoduše tak, že se na základě prvků zadání vypočítá matice, která promítání realizuje (zopakujme, že rozměr matice je 4×4). Promítání samotné se pak provádí násobením této matice a čtyřprvkového vektoru homogenních souřadnic promítaného bodu podle vztahu (1.13). V samostatné práci byste měli použít zadání projekce z příkladu B z podkapitoly 1.6.

Úkol 2.9

Úkol 2.9. Připomeňte si příklad B z podkapitoly 1.6. Poznamenáváme, že se k příkladu váže i program na přiloženém CD (viz úkol 1.10). Program poskytuje dílčí a výsledné transformační matice pro zobrazení, které zadáte. Bude vám později sloužit ke kontrole, zda jste transformační matice sestavili správně.

Úkol 2.10

Úkol 2.10. Otevřete si soubor „camera.h“ v adresáři „šablony\shader“ a prohlédněte si, jakým způsobem je uvnitř programu popsáno zobrazení. Měli byste zjistit, že si program jednak pamatuje hodnoty specifikované zadáním a dále, že vypočítává výslednou matici projekce, kterou si rovněž pamatuje. Úschovu hodnot a výpočet matice zobrazovací transformace provádí metody „set“ a „create“ třídy „camera“.

Úkol 2.11

Úkol 2.11. Realizujte funkci, která provádí výpočet matice zobrazovací transformace postupem podle příkladu B z podkapitoly 1.6. Pomocí programu z úkolu 1.10. si nejprve zkontrolujte veškeré dílčí matice i transformační matici výslednou. Pak v šabloně nahraďte metodu „create“ nově vytvořenou funkcí (metodou) a zkontrolujte funkčnost programu. Vytvořte také svoji vlastní variantu funkce, která zajišťuje zapamatování si zadávacích prvků zobrazení v datové struktuře, kterou jste si prohlédli v předchozím úkolu.

2.7 Ořezání zorným objemem

Doba studia asi 1 hod

Protože vytvářené obrázky mají nejčastěji simulovat vjem člověka, je také zorné pole obvykle specifikováno tak, aby bylo blízké zornému poli vidění lidského. Cílem ořezání je odstranit ty plochy nebo jejich části, které padly mimo specifikované zorné pole. Po přečtení této podkapitoly budete umět ořezání v programu realizovat.

Čemu se zde naučíte?

Připomeňme si, že při středovém promítání je zorné pole (zorný objem) nejčastěji tvořeno komolým jehlanem, jak jsme již dříve ukázali v řešeném příkladě B v podkapitole 1.6. Obvyklé také je, že se zobrazovací transformace konstruuje tak, aby komolý jehlan transformací přešel v jednotkový zorný kvádr. I to jsme v podkapitole 1.6 již také provedli. Z hlediska praktického provádění ořezání je zorný objem ve tvaru jednotkového kvádru výhodný, protože se pak ořezávání děje speciálními rovinami. V příkladu z podkapitoly 1.6 se konkrétně jedná o roviny $x = \pm 1$, $y = \pm 1$, $z = 0$, $z = -1$. Ořezání speciálními rovinami může být provedeno rychleji než ořezání rovinami obecnými.

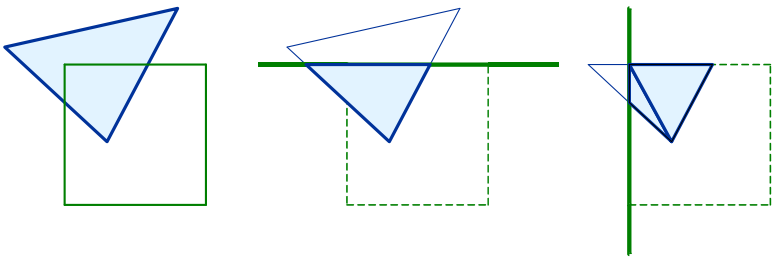
Nejjednodušší způsob ořezání, který lze použít pro zorný objem ve tvaru konvexního tělesa, spočívá v tom, že se všechny roviny, které zorný objem ohraničují, postupně použijí k ořezání všech plošek aproximujících povrchy těles ve scéně (již víme, že plošky mohou být organizovány například v seznamu). I přes svoji jednoduchost se jedná o postup praktický. Podobně jako pro ořezávaní rovinných útvarů (zejména úseček) oknem ve tvaru obdélníka (2D ořezávání) byly i pro ořezávání prostorových útvarů zorným kvádrem (3D ořezávání) publikovány speciální důmyslnější a rychlejší postupy. Jejich význam však není tak velký jako u dvojrozměrné varianty ořezávání. Je to jednoduše proto, že ořezávání zorným objemem je vždy součástí poměrně komplikovaného zobrazovacího řetězce. I dost významná úspora při ořezávání by se proto mohla promítnout vždy jen do poměrně malé úspory času potřebného pro generování celého obrazu. Navíc zde ořezáváme plochy (ořezávání úseček je jednodušší) a konečně i zde platí, že se úspora dosažená různými speciálními postupy zpravidla snižuje s rostoucí dimenzí prostoru, v němž úlohu řešíme.

Poznámka k algoritmům 3D ořezávání

Dohodněme se tedy, že ve svém programu provedete ořezávání skutečně tak, že jednotlivé roviny, které zorný objem ohraničují, použijete postupně k ořezání množiny trojúhelníků aproximujících povrchy těles ve scéně. Při ořezávání každou rovinou ponecháte z trojúhelníků vždy jen ty části, které leží na stejné straně roviny jako zorný objem. Postup ořezání zorným objemem ve tvaru kvádru ilustruje obr. 2.18. Je zřejmé, že základní úlohou, kterou je nutné řešit, je ořezání trojúhelníka rovinou (poloprostorem). Přestože se úloha zdá být jednoduchá, zaměřme se na ni poněkud podrobněji, protože je nezbytné řešit ji v homogenních souřadnicích. Projekcí scény mohly totiž v obraze vzniknout body ležící v nekonečnu. Protože zorný objem je konečný, budou ořezáním takové body (a části ploch) odstraněny. Přechod od homogenních k afinním souřadnicím je proto „bezpečný“ teprve až je ořezání provedeno (připomeňme, že zatímco homogenní souřadnice umožňují reprezentovat body v nekonečnu bez jakýchkoli problémů, u afinních souřadnic tomu tak není).

Ořezávání rovinami ohraničujícími zorný objem

Ořezání budeme řešit v homogenních souřadnicích.



Obr. 2.18. K ořezávání ploch stěnami normalizovaného zorného objemu. V levém obrázku je zorný objem schematicky znázorněn v rovině jako čtverec. Ořezávání lze provést tak, že se roviny, které zorný objem ohraničují, postupně použijí k ořezání všech ploch (trojúhelníků) aproximujících povrchy scény. Na obrázku je znázorněno řezání jediného trojúhelníka dvěma rovinami. Čtyřúhelník vzniklý při řezání druhou rovinou je nutné rozdělit, protože předpokládáme, že program zpracovává pouze trojúhelníky.

Řešme nejprve podrobněji problém jak určit, na které straně od zadané roviny leží zadaný bod *X*. Nechť (n_x, n_y, n_z) je normálový vektor roviny a ρ nechť je vzdálenost roviny od počátku souřadné soustavy (leží-li počátek na opačné straně od roviny, než kam směřuje její normála, budeme hodnotu ρ považovat za kladnou, jinak ji budeme považovat za zápornou). Normálový vektor nechť má jednotkovou délku a nechť směřuje ven ze zorného objemu. Bod *X* je popsán svými homogenními souřadnicemi $\mathbf{x} = (x, y, z, w)$. Mohou nastat následující případy: 1) bod *X* leží mimo zadanou rovinu, a to na straně, kam ukazuje normálový vektor (tedy na opačné straně, než je zorný objem), 2) bod *X* leží v zadané rovině, 3) bod *X* leží mimo rovinu na stejné straně jako zorný objem. Uvedené případy lze snadno rozlišit podle toho, které z relačních znamének platí v následujícím vztahu

$$\left(n_x, n_y, n_z\right)\cdot\left(\frac{x}{w}, \frac{y}{w}, \frac{z}{w}\right) - \rho \begin{matrix} > \\ = \\ < \end{matrix} 0 \ . \tag{2.15}$$

Levá strana vztahu (2.15) je známým výrazem pro odchylku bodu od roviny, který jsme jen upravili pro použití homogenních souřadnic. Zaveďme označení $(n_x, n_y, n_z, -\rho) \equiv \mathbf{n}$. Předchozí vztah pak snadno můžeme upravit na tvar

$$\left(n_x, n_y, n_z, -\rho\right)\cdot\left(x, y, z, w\right) = \mathbf{n}\cdot\mathbf{x} \begin{matrix} > \\ = \\ < \end{matrix} 0 \ . \tag{2.16}$$

Na které straně od roviny leží bod X?

Při ořezání normalizovaným zorným kvádrem mají ořezávací roviny speciální tvar. Uvažujme např. rovinu $x = 1$. Pro tuto rovinu máme $\mathbf{n} = (1, 0, 0, -1)$. Ze vztahu (2.16) pak např. snadno zjistíme, že bod *X* leží na stejné straně roviny jako zorný kvádr, jestliže platí podmínka $x < w$. Pro zbývající roviny normalizovaného zorného kvádru z příkladu v podkapitole 1.6 lze stejně snadno odvodit podobné vztahy. Uveďme pro tyto roviny hodnoty vektorů **n** a také výslednou podmínku potvrzující, že bod leží na stejné straně roviny jako zorný kvádr, alespoň ve stručném přehledu. Máme: $\mathbf{n} = (-1, 0, 0, -1)$, $-x < w$ (rovina $x = -1$); $\mathbf{n} = (0, \pm 1, 0, -1)$, $\pm y < w$ (roviny $y = \pm 1$); $\mathbf{n} = (0, 0, 1, 0)$, $z < 0$ (rovina $z = 0$); $\mathbf{n} = (0, 0, -1, -1)$, $-z < w$ (rovina $z = -1$).

Při ořezávání trojúhelníka rovinou stěny zorného objemu mohou nastat následující případy: 1) Všechny vrcholy leží na opačné straně roviny než zorný objem. Celý trojúhelník tedy leží nepochybně mimo zorný objem a je z dalšího zpracování vyloučen (je jednoduše vypuštěn ze seznamu trojúhelníků aproximujících hranici objektů scény). 2) Všechny vrcholy trojúhelníka leží na stejné straně roviny jako zorný objem. V tomto případě je celý trojúhelník ponechán beze změny. 3) Vrcholy trojúhelníka leží po obou stranách řezové roviny. Tento případ bude vyžadovat „rozřezání“ trojúhelníka. V dalším textu postup popíšeme podrobněji.

Průsečík hrany trojúhelníka s rovinou stěny zorného objemu

Leží-li koncové body některé hrany trojúhelníka na opačných stranách řezové roviny, pak je zapotřebí nalézt průsečík hrany s řezovou rovinou. Nechť vektory $\mathbf{p}_1$, $\mathbf{p}_2$ reprezentují koncové body takové hrany. Pro nalezení průsečíku s výhodou použijeme rovnice přímky diskutované v závěru příkladu 1.4 a rovnice roviny odvozené v předchozím vztahu. Máme tedy rovnice

$$\mathbf{x} = \left(1-\lambda\right)\mathbf{p}_1 + \lambda\mathbf{p}_2 , \quad \mathbf{n}\cdot\mathbf{x} = 0 \ . \tag{2.17}$$

Řešením snadno získáme

$$\lambda = \frac{\mathbf{n} \cdot \mathbf{p}_1}{\mathbf{n} \cdot (\mathbf{p}_1 - \mathbf{p}_2)} \,. \tag{2.18}$$

Jestliže hrana řezovou rovinu protíná, pak musí být $\mathbf{n} \cdot (\mathbf{p}_2 - \mathbf{p}_1) \neq 0$, $0 \leq \lambda \leq 1$ (jestliže by platilo $\mathbf{n} \cdot (\mathbf{p}_2 - \mathbf{p}_1) = 0$, pak by byla hrana s řezovou rovinou rovnoběžná). Ořezáním trojúhelníka rovinou může vzniknout buď trojúhelník nebo čtyřúhelník (obr. 2.18). Často se připouští pouze trojúhelníky. I my jsme se tak dohodli, a proto je zapotřebí čtyřúhelník rozdělit na dva trojúhelníky. V každém případě (ať již je výsledkem ořezání trojúhelníka rovinou trojúhelník jediný nebo dvojice trojúhelníků) je v nově vzniklých vrcholech trojúhelníků (průsečících hran původního trojúhelníka s ořezávací rovinou) zapotřebí dopočítat ty hodnoty, které jsou k vrcholům přidruženy a které budou později ještě používány. V případě Gouraudova stínování se jedná o intenzitu osvětlení a v případě stínování Phongova o normálu plochy. V obou případech lze výpočet provést interpolací mezi hodnotami v koncových bodech původních hran.

Příklad 2.1

Příklad 2.1. Najděte průsečík úsečky s rovinou $z = 1$. Koncové body úsečky mají homogenní souřadnice $\mathbf{p}_1 = (1, 1, 0, 1)$, $\mathbf{p}_2 = (3, 1, 2, 1)$.

Řešení. Máme $\mathbf{n} = (0, 0, 1, -1)$. Ze vztahu (2.18) vyjde $\lambda = 1/2$. Souřadnice průsečíku jsou $(2, 1, 1, 1)$. □

Úkol 2.12

Úkol 2.12. Realizujte funkci, která právě popsaným postupem provádí ořezání trojúhelníků aproximujících povrch těles ve scéně. Nahraďte touto funkcí původní funkci v šabloně a přesvědčete se, zda vaše funkce funguje správně.

2.8 Přechod od homogenních k afinním souřadnicím a transformace do souřadné soustavy zařízení

Doba studia asi 0,5 hod

Také na řešení obou úloh uvedených v nadpisu této podkapitoly jste se detailně připravili již v kapitole předchozí. Přechod od homogenních k souřadnicím afinním byl popsán v podkapitole 1.4 a transformace z jednotkového zobrazovacího objemu do souřadné soustavy výstupního zařízení byla podrobně vysvětlena v podkapitole 1.8. Obě úlohy zde uvádíme jen pro úplnost, protože jsou nedílnou součástí popisovaného řetězce (obr. 2.3) a protože jsme slíbili, že budeme řetězec probírat podle pořadí, jak jednotlivé kroky provádí.

Úkol 2.13

Úkol 2.13. Realizujte funkci, která provádí přechod od homogenních k afinním souřadnicím, a funkci, která realizuje transformaci na výstupní zařízení. Oběma funkcemi postupně nahraďte původní funkce v šabloně a zkontrolujte, zda vaše nové funkce fungují správně.

2.9 Vykreslení na rastrové výstupní zařízení a řešení viditelnosti

Doba studia asi 1,5 hod

Čemu se zde naučíte?

Nyní jsme v situaci, kdy mají být jednotlivé plochy ve tvaru trojúhelníka vykresleny na rastrové výstupní zařízení. Často se jedná o displej nebo o vykreslení do souboru. V této kapitole se naučíte, jak při vykreslování postupovat při použití

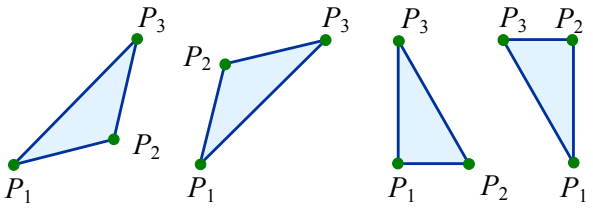
Gouraudova stínování (rozdíly vyplývající z použití stínování Phongova uvedeme v následující podkapitole 2.10).

Výchozí údaje

Připomeňme si nejprve okolnosti, za nichž úlohu řešíme. Afinní souřadnice $\mathbf{p}_i = (x_i, y_i, z_i)$, $i \in \{1, 2, 3\}$ vrcholů trojúhelníka v souřadné soustavě zařízení jsou známy z předchozího kroku. V každém z vrcholů jsme již také dříve (postupem popsaným v podkapitole 2.5) vypočetli barvu (r_i, g_i, b_i). Nyní je úkolem náležitou barvou „rozsvítit" jednotlivé body obrazu (pixely), na které se obrazy ploch aproximujících povrchy objektů scény promítají. Požaduje se, aby při tom byla respektována viditelnost. Neviditelné části ploch (skryté pozorovateli za jinými plochami) nemají být vykresleny.

Změna pořadí vrcholů trojúhelníka

Před vykreslováním je výhodné změnit pořadí vrcholů trojúhelníka (přeskládat je v datové struktuře, která trojúhelník reprezentuje). Pořadí vrcholů až dosud určovalo směr normály trojúhelníkové plošky, normálu už ale dále potřebovat nebudeme. Přeskládání má význam při praktické realizaci zobrazovacího řetězce. Lze jím totiž dosáhnout toho, že program nebude muset rozlišovat tolik případů jako v případě, kdyby přeskládání provedeno nebylo. Přeskládání lze provést např. tak, aby nastal jeden z případů uvedených na obr. 2.20, kde je použito pravidla, aby vrchol P_1 měl ze všech vrcholů nejmenší souřadnici y a vrchol P_3 největší.



Obr. 2.20. Přeskládání vrcholů trojúhelníka podle pravidla, že vrchol P_1 má nejmenší souřadnici y a vrchol P_3 největší. Napravo jsou dva speciální případy kdy $y_1 = y_2$, $y_2 = y_3$.

Rasterizace cyklem přes řádky plochy

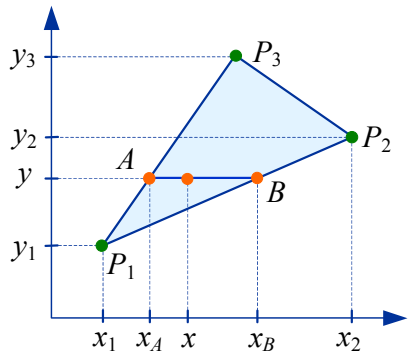
Ve výkladu se dále podrobněji zaměříme pouze na první případ z obrázku 2.20. Řešení ostatních je velmi podobné. Při vykreslování plochy postupně vykreslujeme jednotlivé řady obrazových bodů (pixelů). Řekněme, že pracujeme po řádcích. Řádek je charakterizován svojí celočíselnou souřadnicí y na výstupním zařízení. Vykreslení trojúhelníkové plochy tedy můžeme realizovat ve dvou cyklech přes souřadnici y, a to v cyklu $y_1 \le y \le y_2$, a v cyklu $y_2 < y \le y_3$. Pro každý řádek y je nutné stanovit místa, kde řada obrazových bodů, které mají být rozsvíceny, začíná a kde končí. Tato místa jsou specifikována souřadnicemi x_A, x_B (obr. 2. 21). Zaměřme se podrobněji na případ, kdy platí $y_1 \le y \le y_2$. Pomocí úměry pro souřadnice x_A, x_B snadno získáme vztahy

Stanovení začátku a konce řádku

$$x_A = x_1 + \frac{y - y_1}{y_3 - y_1}(x_3 - x_1), \quad x_B = x_1 + \frac{y - y_1}{y_2 - y_1}(x_2 - x_1) \,. \tag{2.19}$$

Jsou li hodnoty x_A, x_B, známy, lze již odpovídající část řádku obrazových bodů plochy vykreslit. Lze opět postupovat v cyklu, v němž jsou pro všechny celočíselné

hodnoty x, Round(x_A) ≤ x ≤ Round(x_B) postupně zpracovávány jednotlivé obrazové body (x, y).



Obr. 2.21. Interpolace na ploše trojúhelníka.

Zpracování obrazového bodu v řádku

Určení souřadnice z

Určení barvy

Interpolace reálné hodnoty po ploše trojúhelníka

Při zpracování obrazového bodu je rozumné nejprve rozhodnout, zda je odpovídající bod zobrazované plochy potenciálně viditelný či neviditelný. Ke stanovení viditelnosti metodou „z-buffer“, kterou v této podkapitole později podrobně popíšeme, je zapotřebí určit pro každý obrazový bod souřadnici z toho bodu plochy, který se do vyšetřovaného obrazového bodu promítá. Protože i během transformace na výstupní zařízení je vrcholům trojúhelníka stále ještě ponechána také souřadnice z (podkapitoly 1.8, 2.8), lze souřadnici z určit jednoduše lineární interpolací. V případě, že je zobrazovaný bod plochy rozhodnut jako viditelný, je dále zapotřebí spočítat jeho barvu a touto barvou „rozsvítit“ odpovídající (právě zpracovávaný) bod obrazu. V případě Gouraudova stínování se barva obrazového bodu stanoví lineární interpolací mezi dříve vypočtenými hodnotami barev ve vrcholech trojúhelníka. Vidíme tedy, že v každém obrazovém bodě bude zapotřebí provádět interpolaci až pro celkem 4 reálné veličiny, kterými jsou souřadnice z, a barevné složky R, G, B (jestliže je ale bod plochy rozhodnut jako neviditelný, pak hodnoty barevných složek nejsou potřebné). Abychom zde neopakovali vícekrát téměř stejné a navíc jednoduché vzorce, řekněme obecně, že ve vrcholech trojúhelníka máme nějaké reálné hodnoty v_1, v_2, v_3 a že hledáme hodnotu v v bodě o souřadnicích (x, y) uvnitř trojúhelníka. Hledanou hodnotu lze snadno zjistit interpolací pomocí vzorců

$$v_A = v_1 + \frac{y - y_1}{y_3 - y_1}(v_3 - v_1), \quad v_B = v_1 + \frac{y - y_1}{y_2 - y_1}(v_2 - v_1),$$
$$v = v_A + \frac{x - x_A}{x_B - x_A}(v_B - v_A), \tag{2.20}$$

kde v_A, v_B značí hodnoty v bodech o souřadnicích (x_A, y), (x_B, y) (obr. 2.21). Poznamenáváme důrazně, že vztahy (2.20) jsou zapsány tak, aby bylo co nejlépe patrné, jak byly odvozeny. Praktická implementace interpolace je dosti kritická. Vztahy jsou sice jednoduché, ale provádí se mnohokrát. Implementaci je proto zapotřebí provést racionálně. Jednoduše řečeno: V cyklech je nutné vyhnout se

výpočtu týchž výrazů nebo jejich částí. Vše co lze, je nutné připravit před prováděním cyklů. Rozhodně se nedoporučuje spoléhat jen na optimalizaci překladačem.

Řešení viditelnosti metodou „z-buffer“

Řešení viditelnosti se dnes velmi často provádí algoritmem, který obvykle bývá v žargónu stručně označován jako „*z*-buffer“ (česky by bylo možné říci „algoritmus využívající paměti hloubky“). Jedná se o algoritmus jednoduchý, ale účinný, který je určen pro generování obrazů složených z bodů (pixelů). K jeho činnosti jsou nutná čtyři pole. Počet prvků každého z nich je stejný jako počet bodů vytvářeného obrazu. V jednotlivých polích se pro každý obrazový bod uchovávají barevné složky R, G, B a dále souřadnice *z* toho bodu plochy, který se na daný bod obrazu promítá a který je nejblíže pozorovateli (je viditelný). Můžete si jednoduše myslet, že hodnoty R, G, B, *z* jsou reprezentovány jako reálná čísla (každé např. na 4 bajtech). Taková reprezentace je při praktické realizaci systému nejpohodlnější, neboť není zapotřebí příliš „hlídat“ rozsahy ani přesnost jednotlivých hodnot. Různé úsporné (dosti často celočíselné) reprezentace sice přinášejí úsporu paměti a snižují požadavky na výpočetní výkon, ale z hlediska praktické implementace zobrazovacího řetězce jsou spíše komplikací. Nepřesná reprezentace hodnoty *z* může navíc při běhu programu vést až k chybnému určení viditelnosti v některých obrazových bodech. Za maximálně úsporný lze považovat případ, kdy je každá z barevných složek uchovávána na jednom bajtu a hodnota *z* na dvou bajtech.

Roviny R,G,B, *z a jejich realizace*

Popis práce algoritmu „z-buffer“

Popišme nyní algoritmus „*z*-buffer“ podrobněji. Dříve, než vytváření obrazu začne, jsou pole R, G, B naplněna hodnotami reprezentujícími zvolenou barvu pozadí (např. černou barvu) a pole obsahující souřadnici *z* je naplněno hodnotou $z_{\min}$. Hodnota $z_{\min}$ je taková hodnota souřadnice *z*, která není překročena ani na plochách, které jsou od pozorovatele vzdáleny nejvíce. Předpokládáme, že jsme zobrazovací transformaci provedli podle příkladu z podkapitoly 1.6 a transformaci na výstupní zařízení podle podkapitoly 1.8. Pak souřadnice *z* vrcholů trojúhelníků a také všech jiných zobrazovaných bodů nabývá hodnot $-1 \le z \le 0$. Body se souřadnicí z = −1 jsou od pozorovatele nejvzdálenější. Jako $z_{\min}$ lze proto v našem případě zvolit libovolnou hodnotu pro kterou platí $z_{\min} < -1$. Plochy aproximující povrch tělesa lze algoritmem „*z*-buffer“ zpracovávat v libovolném pořadí. Během výpočtu se v paměti hloubky udržují hodnoty souřadnice *z* těch bodů ploch, které již byly zpracovány a které jsou zatím pozorovateli nejblíže.

Vlastní výpočet probíhá takto: Postupně se zpracovávají všechny plochy, kterými byly povrchy těles aproximovány a které zbyly po provedení předchozích kroků zobrazovacího řetězce. Na pořadí zpracování nezáleží. Pro každou zpracovávanou plochu se pokoušíme „rozsvítit“ všechny obrazové body (pixely) jejího průmětu. U každého zpracovávaného obrazového bodu stanovíme interpolací souřadnici *z* odpovídajícího bodu na zpracovávané ploše a porovnáme ji s hodnotou zapsanou na odpovídajícím místě paměti hloubky (obr. 2.22). Je-li vypočítaná hodnota menší než hodnota zapsaná v paměti hloubky, znamená to, že je odpovídající bod plochy od pozorovatele více vzdálen než bod některé jiné plochy, která již byla zpracována dříve. Bod plochy tedy nebude viditelný, není nutné pro něj počítat barvu a lze přejít na zpracování dalšího obrazového bodu. Jestliže je naopak hodnota souřadnice *z* bodu plochy větší než hodnota zapsaná v paměti hloubky, znamená to, že je bod plochy pozorovateli blíže než body všech ploch, které byly až dosud zpracovány a které se do právě prověřovaného bodu obrazu zobrazily. V prověřovaném bodě obrazu „bude tedy zatím vidět“ právě zpracovávanou plochu. Hodnota souřadnice *z* odpovídajícího bodu zpracovávané

plochy bude proto zapsána do paměti hloubky (pole z). Dále bude vypočtena barva plochy v prověřovaném bodě a její složky budou zapsány do polí R, G, B. Slovní spojení „zatím vidět“ jsme použili proto, že viditelnost byla stanovena jen na základě dosud zpracovaných ploch. Bod plochy, který byl v nějakém okamžiku stanoven jako viditelný, může být překryt plochou, která bude zpracovávána později. Algoritmus z-buffer lze shrnout takto:

Algoritmus „z-buffer“

Shrnutí algoritmu „z-buffer“

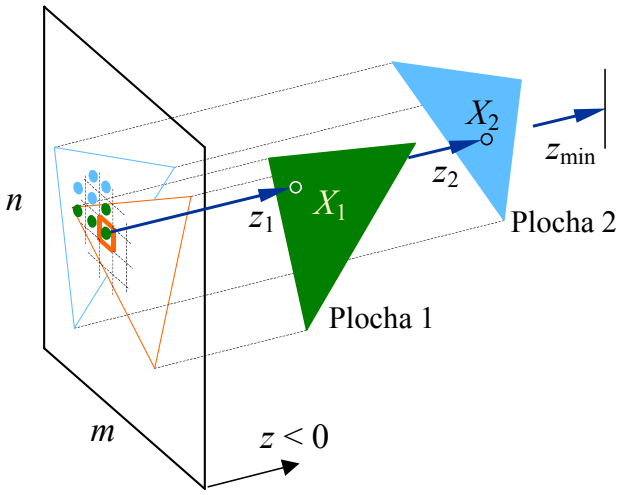
```
Nechť m × n jsou rozměry obrazu zadané v počtu obrazových bodů.
Vytvoř pole R, G, B (obrazovou paměť) a pole z (paměť hloubky)
každé o rozměru m × n.
Do všech pozic obrazové paměti zapiš barvu pozadí (např. černou).
Do všech pozic paměti hloubky zapiš hodnotu z_min (např. −∞).

Probírej seznam trojúhelníků aproximujících povrch scény.
Pro každý trojúhelník dělej:

    Generuj všechny pixely trojúhelníka.
    Pro každý pixel (x,y) dělej:

        Vypočítej v místě (x,y) souřadnici z zpracovávané plochy.
        Je-li z > Paměť hloubky[x,y] pak proveď:
            Zapiš zjištěnou souřadnici z do paměti hloubky na
            pozici [x,y].
            Vypočítej barvu plochy v místě (x,y,z) a zapiš ji do
            obrazové paměti na pozici [x,y].

Po zpracování všech trojúhelníků je v obrazové paměti výsledný
obraz.
```



Obr. 2.22. K výpočtu viditelnosti metodou „z-buffer“. Obrazovému bodu zvýrazněnému rámečkem odpovídají na plochách 1, 2 body X_1, X_2. Jsou-li plochy zpracovávány v pořadí plocha 1, plocha 2, bude nejprve testováno, zda platí $z_{min} < z_1$. Protože je podmínka splněna (předpokládáme, že všechny hodnoty z na obrázku jsou záporné), bude do obrazového bodu zapsána barva bodu X_1. Při vykreslování druhé plochy se bod X_2 vyhodnotí již jako neviditelný, protože platí $z_2 < z_1$.

Časová složitost vykreslování

Čas potřebný k vykreslení obrazu na rastrové výstupní zařízení právě popsaným postupem s využitím algoritmu „*z*-buffer“ roste lineárně s úhrnnou velikostí průmětů ploch scény (čím větší průměty ploch jsou a čím je ploch více, tím více obrazových bodů je nutné generovat, prověřovat a případně také „rozsvěcet“). Časová složitost může také záviset na pořadí, v němž jsou plochy zpracovávány (úkol 2.14).

Úkol 2.13

Úkol 2.13. Zapište vztahy 2.19, 2.20 v takovém tvaru, který by umožňoval jejich efektivní provádění v cyklu přes řádky a přes sloupce (tj. tak, jak jsme uvažovali během předchozího výkladu).

Úkol 2.14

Úkol 2.14. Rychlost provádění algoritmu „*z*-buffer“ může záviset na pořadí, v němž jsou plochy zpracovávány. Promyslete, jaké pořadí může být pro algoritmus nejpříznivější a naopak nejméně příznivé. Upřesněte okolnosti, za jakých jsou vaše úvahy platné.

Úkol 2.15

Úkol 2.15. Realizujte funkci, která právě popsaným postupem provádí vykreslení na rastrové výstupní zařízení a řešení viditelnosti. Nahraďte touto funkcí původní funkci v šabloně a přesvědčete se, zda vaše funkce funguje správně.

Úkol 2.16

Úkol 2.16. Promyslete, k jakým úsporám paměti a výpočetního času může dojít použitím sítí místo jednotlivých (např. trojúhelníkových plošek). Promyslete také, v kterém okamžiku je pak ale vhodné síť na jednotlivé plošky rozdělit.

S H R N U T Í

Shrnutí: V tuto chvíli byste měli vědět, jak standardní zobrazovací řetězec provádí závěrečné vykreslení na výstupní zařízení, tedy jak „rozsvěcuje“ jednotlivé body obrazu (pixely). Měli byste vědět, jak se určuje viditelnost a barva bodů v obraze.

2.10 Realizace řetězce při použití Phongova stínování

Doba studia asi 0,5 hod

Gouraudovo stínování má občas problémy, které řeší stínování Phongovo.

Čemu se zde naučíte?

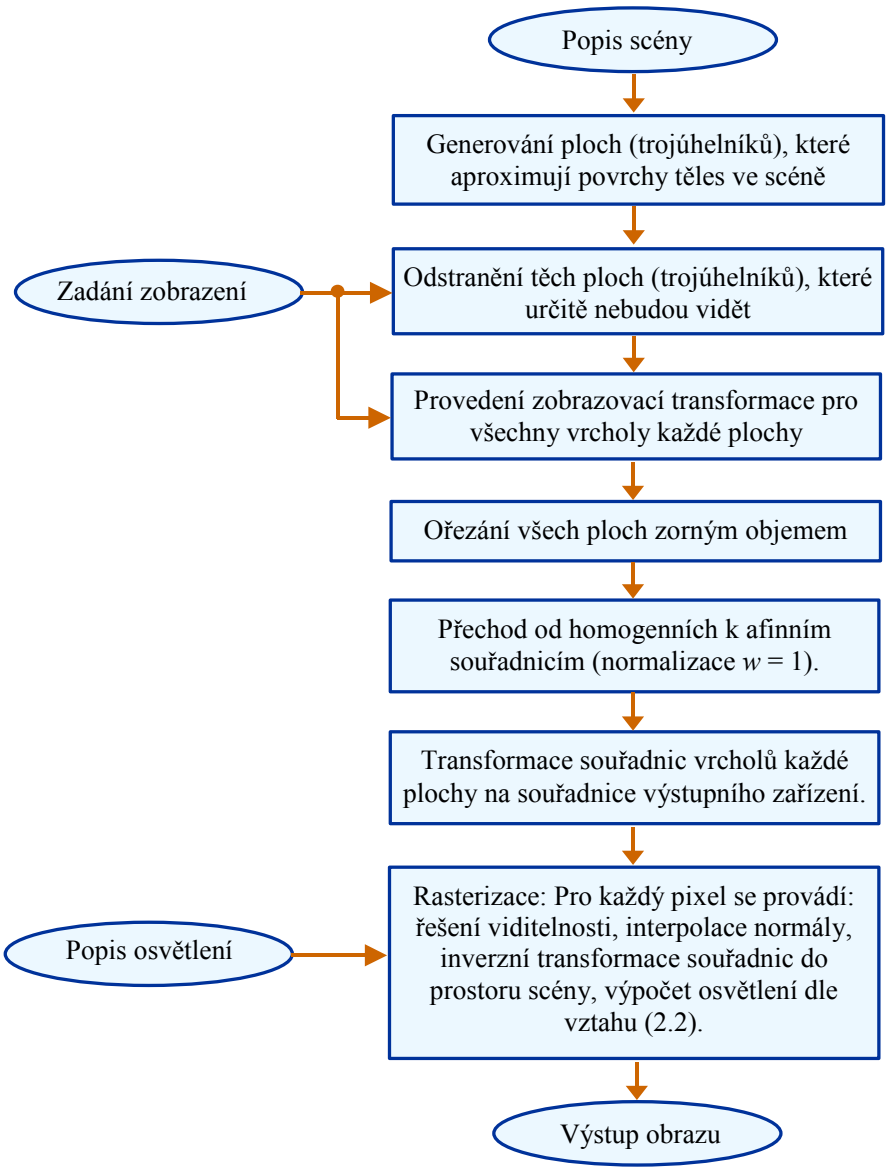
Jestliže jste již vytvořili program realizující Gouraudovo stínování a jestliže jste jej už také použili k zobrazování různých scén (nebo pokud jste totéž prováděli s nějakým programem hotovým), nemuseli jste být vždy s výsledným obrazem zcela spokojeni. Problémy mohly nastat např. tehdy, jestliže jste v obraze očekávali vznik malých a jasných plošek zrcadlového odrazu (obr. 2.24). Nyní, když už víte, že Gouraudovo stínování na trojúhelnících barvu interpoluje, je určitě jasné, že pěkného výsledku by v takovém případě mohlo být dosaženo jen tehdy, když by trojúhelníky byly velmi malé. To je zase ale nevýhodné z hlediska spotřeby jak paměti tak i výpočetního času. Diskutovaný nedostatek Gouraudova stínování odstraňuje stínování Phongovo. Rozdílnost výsledků obou postupů ilustruje obr. 2.24. V této podkapitole se dovíte, v čem Phongovo stínování spočívá, a po přečtení podkapitoly jej také budete umět prakticky realizovat.

U Phongova stínování se celý výpočet osvětlení realizuje až v závěrečné fázi vytváření obrazu, tedy při rasterizaci (odpadá výpočet osvětlení ve vrcholech ploch, který jsme u Gouraudova stínování prováděli ještě před výpočtem zobrazovací transformace, obr. 2.23). Phongovo stínování vychází ze znalosti

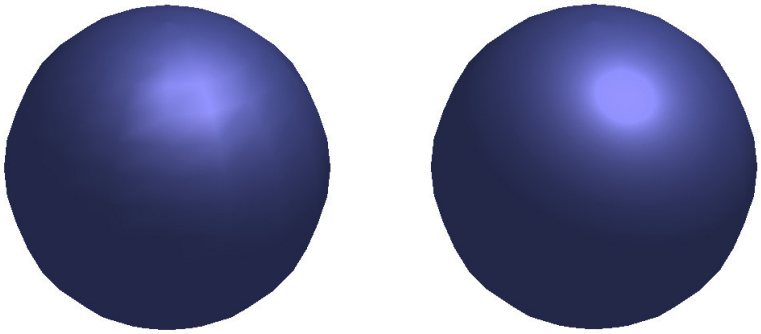
normál ve vrcholech a při rasterizaci interpolací určuje směr normály v každém pixelu. Na základě takto určené normály se pak pro každý obrazový bod osvětlení počítá podle vzorce (2.2). Protože jsou k výpočtu osvětlení zapotřebí nezkreslené úhly a případně i délky, je ale nejprve nutné provést zpětnou transformaci polohy obrazového bodu do prostoru scény. To se provede postupem, který jsme probrali v kapitole 1.9. Je zřejmé, že vyšší kvalita obrázků produkovaných Phongovým stínováním je zaplacena podstatně složitějšími výpočty, které se pro každý obrazový bod musí provádět. Proto obzvláště v souvislosti s touto metodou stojí za to zdůraznit, že algoritmus „z-buffer“ lze modifikovat tak, že se osvětlení počítá jen v bodech, které skutečně budou nakonec vidět. Modifikovaný algoritmus běží ve dvou průchodech: V prvním se řeší viditelnost. Místo výpočtu barev si algoritmus pouze zapamatuje, které plochy jsou v jednotlivých bodech obrazu vidět. Výpočet barev se provede až ve druhém průchodu.

Jak Phongovo stínování pracuje?

Užitečná modifikace algoritmu „z-buffer“.



Obr. 2.23. Schéma zobrazovacího řetězce při použití Phongova stínování.



Obr. 2.24. Objekt stínovaný Gouraudovým (vlevo) a Phongovým (vpravo) stínováním.

Úkol 2.17

Úkol 2.17. Promyslete, jak těžké by bylo doplnit Phongovo stínování do programu, který souběžně s výkladem realizujete. Máte-li chuť, můžete si vyzkoušet i jeho praktickou realizaci. Pokud jste program realizovali promyšleně („pěkně“ z pohledu programátora), pak by to nemělo být těžké. Přibude zejména zpětná transformace ze zobrazovacího zařízení do prostoru scény. Vše ostatní jsou jen drobnosti.

2.11 Nanášení textury

Doba studia asi 1,5 hod

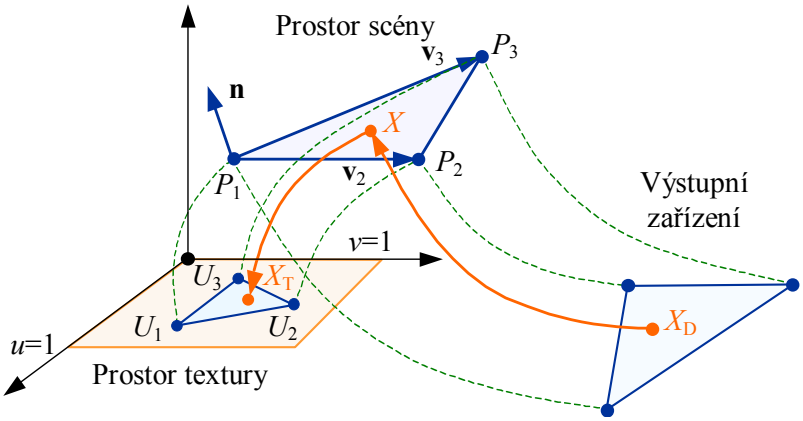
Čemu se zde naučíte?

Účinným nástrojem k získávání věrně vypadajících obrázků je nanášení textury. Pokud jste se ve čtení tohoto textu dostali až sem, pak už jistě víte, o co se jedná. Obvykle se na stěny těles, která scénu vytvářejí, texturou „nanáší vzhled materiálu“ (obr. 3.1) nebo různé nálepky, nápisy atd. V této podkapitole se naučíte nanášení textury v programech realizovat. Seznámíte se zde s postupem založeným na použití transformací. (V některých systémech se můžete také setkat s postupy založenými na interpolaci. Ty pracují rychleji, ale jejich výsledky jsou horší).

Jak je zadán způsob přiložení textury?

Předpokládejme, že povrch objektu vytvářejí trojúhelníkové plochy. Uvažujme jednu z nich. Její vrcholy nechť jsou P_1, P_2, P_3 (obr. 2.25). V souřadné soustavě scény nechť jsou uvedené vrcholy reprezentovány vektory $\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{p}_3$. Na uvažovanou plochu je „přiložena“ textura. Předpokládejme, že textura je popsána diskrétní funkcí definující barvu. Prakticky se obvykle jedná o dvojrozměrné pole, v němž jsou pro každý bod uvedeny jeho barevné složky (někdy se pro body textury užívá také názvu „texely“). Dále je zaveden prostor a souřadný systém textury. Výklad se zjednoduší, když si budete myslet, že textura leží přímo v rovině *xy* souřadné soustavy scény (obr. 2.25). Způsob přiložení textury na uvažovanou plochu je specifikován tak, že jsou v rovině *xy* textury zadány „lícovací body“ U_1, U_2, U_3. Poloha těchto bodů je popsána vektory $\mathbf{u}_1$, $\mathbf{u}_2$, $\mathbf{u}_3$ vztaženými k souřadnému systému textury. Textura je při přiložení na plochu podrobena afinní transformaci tak, aby body U_1, U_2, U_3 padly do bodů P_1, P_2, P_3. Pro námi uvažovanou plochu jsou tedy k dispozici následující informace

$$\begin{aligned} &P_1\colon \mathbf{p}_1 = (x_1, y_1, z_1), \quad P_2\colon \mathbf{p}_2 = (x_2, y_2, z_2), \quad P_3\colon \mathbf{p}_3 = (x_3, y_3, z_3), \\ &U_1\colon \mathbf{u}_1 = (u_1, v_1), \quad U_2\colon \mathbf{u}_2 = (u_2, v_2), \quad U_3\colon \mathbf{u}_3 = (u_3, v_3)\ . \end{aligned} \tag{2.21}$$



Obr. 2.25. Nanášení textury.

Jak se nanášení textury provádí?

K nanášení textury je zapotřebí stanovit transformaci z prostoru scény do prostoru textury. V našem případě se jedná o transformaci afinní provádějící zobrazení $P_1 \to U_1$, $P_2 \to U_2$, $P_3 \to U_3$. Tím je hledaná transformace jednoznačně určena. Jestliže transformaci známe, pak praktická realizace nanášení textury probíhá takto: Víte, že při rasterizaci plochy se rozsvěcují její jednotlivé obrazové body (pixely). Řekněme, že právě rozsvěcujete bod X_D (obr. 2.25). Inverzní transformací k transformaci zobrazovací (kapitola 1.9) zjistíte, kde leží vzor tohoto bodu ve skutečné scéně (bod X na obr. 2.25). Transformací bodu z prostoru scény do prostoru textury zjistíte, do kterého místa textury se bod X zobrazí (na obr. 2.25 je tento bod označen jako X_T). Nakonec zjistíte barvu textury v místě X_T a použijete ji k vykreslení bodu X_D vytvářeného obrazu. (Existují ovšem i důmyslnější možnosti, jak s informací přečtenou z textury naložit. Jejich podrobnější diskusi ale ponecháme až do kapitoly 5.)

Transformace z prostoru scény do prostoru textury

Nechť $\mathbf{x}$ značí vektor souřadnic bodu ve scéně a $\mathbf{x}_T$ vektor souřadnic jemu odpovídajícího bodu v textuře. Použijeme homogenních souřadnic, a proto se v obou případech jedná o vektory o čtyřech složkách. (Použití homogenních souřadnic sice není v uvažovaném případě nezbytně nutné, ale je obvyklé, protože se jednoduše pouze využívá prostředků, které v programu stejně musí být k realizaci jiných transformací). Hledanou transformaci sestavíme ze tří transformací dílčích postupem, který znáte z kapitoly 1. Máme tedy

Transformaci sestavíme ze tří transformací dílčích.

$$\mathbf{x}_T = \mathbf{x}\mathbf{T}_1\mathbf{T}_2\mathbf{T}_3 \ . \tag{2.22}$$

V prvním kroku posuneme trojúhelník tak, aby jeho vrchol P_1 přešel do počátku souřadné soustavy scény (tedy také do počátku textury). Z kapitoly 1 víte, že matice realizující toto posunutí má tvar

$$\mathbf{T}_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -x_1 & -y_1 & -z_1 & 1 \end{bmatrix} . \tag{2.23}$$

Ve druhém kroku provedeme rotaci roviny posunutého trojúhelníka tak, aby splynula s rovinou xy a hrana P_1P_2 trojúhelníka aby splynula s osou x. Ke stanovení matice popisující tuto transformaci zavedeme vektory

$$\mathbf{v}_2 = P_2 - P_1, \quad \mathbf{v}_3 = P_3 - P_1, \quad \mathbf{n} = \mathbf{v}_2 \times \mathbf{v}_3 ,$$
$$\hat{\mathbf{z}} = \frac{\mathbf{n}}{|\mathbf{n}|}, \quad \hat{\mathbf{x}} = \frac{\mathbf{v}_2}{|\mathbf{v}_2|}, \quad \hat{\mathbf{y}} = \hat{\mathbf{z}} \times \hat{\mathbf{x}} . \tag{2.24}$$

Nechť je

$$\hat{\mathbf{x}} = (\hat{x}_x, \hat{x}_y, \hat{x}_z), \quad \hat{\mathbf{y}} = (\hat{y}_x, \hat{y}_y, \hat{y}_z), \quad \hat{\mathbf{z}} = (\hat{z}_x, \hat{z}_y, \hat{z}_z) . \tag{2.25}$$

S využitím závěrů z kapitoly 1 (zejména podkapitoly 1.2) pak pro matici $\mathbf{T}_2$ máme

$$\mathbf{T}_2 = \begin{bmatrix} \hat{x}_x & \hat{y}_x & \hat{z}_x & 0 \\ \hat{x}_y & \hat{y}_y & \hat{z}_y & 0 \\ \hat{x}_z & \hat{y}_z & \hat{z}_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} . \tag{2.26}$$

Ve třetím kroku provedeme afinní transformaci v rovině xy (v ní už trojúhelník po provedení předchozích dvou kroků leží) tak, aby vrcholy P_1, P_2, P_3 nabyly souřadnic zadaných vektory $\mathbf{u}_1$, $\mathbf{u}_2$, $\mathbf{u}_3$. Požadovanou transformaci ukazuje obr. 2.26. Matici $\mathbf{T}_3$ hledáme ve tvaru (protože se jedná o afinní transformaci v rovině, předem víme, že na některých místech matice budou hodnoty 0 resp. 1)

Tohoto postupu stanovení transformační matice používáme poprvé!

$$\mathbf{T}_3 = \begin{bmatrix} a_{11} & a_{12} & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ b_1 & b_2 & 0 & 1 \end{bmatrix} . \tag{2.27}$$

Položme dále

$$\hat{\mathbf{p}}_i = \mathbf{p}_i \mathbf{T}_1 \mathbf{T}_2, \quad \hat{\mathbf{u}}_i = (u_i, v_i, 0, 1), \quad i = 1,2,3 . \tag{2.28}$$

Vidíme, že hodnoty $\hat{\mathbf{p}}_i$ znamenají souřadnice bodu po provedení prvních dvou kroků transformace a že hodnoty $\hat{\mathbf{u}}_i$ jsou jednoduše pouze rozšířením souřadnic zadaných v textuře o třetí a čtvrtou složku. S ohledem na to, že od třetího transformačního kroku požadujeme, aby provedl zobrazení $\hat{\mathbf{p}}_i \to \hat{\mathbf{u}}_i$, máme pro hledané členy a_{11}, a_{12}, a_{21}, a_{22}, b_1, b_2 systém rovnic

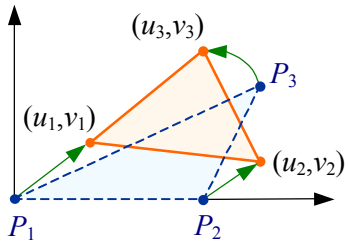
$$\hat{\mathbf{u}}_i = \hat{\mathbf{p}}_i \mathbf{T}_3 . \tag{2.29}$$

Z předchozího maticového vztahu získáme pro každou hodnotu i dvě rovnice (jedná se o rovnice pro první dvě souřadnice; pro třetí a čtvrtou souřadnici bychom získali pro řešení nezajímavé rovnice 0 = 0, 1 = 1). Pro i = 1, 2, 3 tak máme celkem šest rovnic pro celkem šest neznámých hodnot a_{11}, a_{12}, a_{21}, a_{22}, b_1, b_2. Prvky matice $\mathbf{T}_3$ tedy můžeme stanovit řešením právě odvozené soustavy rovnic.

Při zadávání vektorů $\mathbf{u}_1$, $\mathbf{u}_2$, $\mathbf{u}_3$ se dosti často předpokládá rozměr textury normalizovaný, tj. 1 v obou směrech. Nechť u, v ($u,v \in \langle 0,1 \rangle$) jsou souřadnice v normalizovaném prostoru textury a N_u, N_v nechť jsou skutečné rozměry textury zadané počtem řad obrazových bodů (texelů). Skutečné souřadnice v textuře měřené v obrazových bodech (označme je $\tilde{u}, \tilde{v}$) pak snadno získáme pomocí vztahů

Normalizované a skutečné souřadnice v textuře

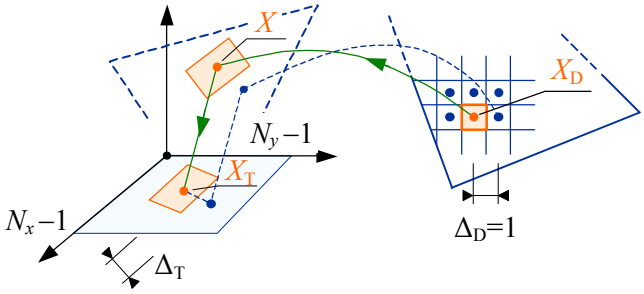
$$\tilde{u} = u(N_u - 1), \quad \tilde{v} = v(N_v - 1) . \tag{2.30}$$



Obr. 2.26. K transformaci realizované maticí $\mathbf{T}_3$.

Řešení problémů s podrobností textury

Závažnou je při nanášení otázka přiměřené podrobnosti textury. Problém je ilustrován na obr. 2.27. Představte si, že je uvažovaná trojúhelníková plocha vykreslována např. na displej. Při vykreslování reprezentuje jediný pixel na výstupním zařízení čtvercovou plošku. Transformujme tuto plošku postupně do prostoru scény a odtud dále do prostoru textury. V prostoru textury je obrazem plošky čtyřúhelník (obr. 2.27). Jestliže do tohoto čtyřúhelníka padne více bodů textury, nastává situace hodná zamyšlení: Kterou hodnotu bychom měli použít pro nakreslení bodu výsledného obrazu? Správným řešením je použít hodnotu získanou průměrováním textury nad zmíněným čtyřúhelníkem. (Při praktické realizaci je výhodné nahradit obecný čtyřúhelník čtvercem se středem v bodě X_T a stranou stanovenou tak, že se do prostoru textury transformuje vzdálenost mezi dvěma sousedními pixely výsledného obrazu.) Jestliže do čtyřúhelníka (resp. čtverce) padne velký počet bodů textury, je výpočet průměru časově náročný, což je závažný problém.



Obr. 2.27. K problému volby podrobnosti textury.

Interpolace v textuře a mipmapping

V případech, kdy jsou rozměry čtyřúhelníka (resp. čtverce) přibližně stejně velké jako vzdálenost bodů textury, lze od počítání průměrů upustit a barvu stanovit jednoduše jako barvu, kterou má textura právě v bodě X_T. Souřadnice bodu X_T nejsou ale obvykle celočíselné. Jednoduše lze hodnotu barvy v místě X_T stanovit např. tak, že se použije hodnota z toho bodu textury, který je místu X_T nejbližší (často se pro tento postup používá názvu „metoda nejbližšího souseda“). Pěknější výsledky ale dává bilineární interpolace mezi hodnotami ve všech čtyřech nejbližších sousedech bodu X_T. Některé systémy (např. OpenGL) řeší nanášení textury tak, že umožňují zadání jedné a téže textury v několika úrovních podrobnosti. Systém pak podle potřeby vybírá popis textury v té podrobnosti, kde

lze postupovat tak, jak jsme právě popsali v tomto odstavci (kde tedy není zapotřebí počítat průměry). Tato technika bývá označována jako „mipmapping“.

Úkol 2.18

Úkol 2.18. Promyslete, jak těžké by bylo doplnit nanášení textury do programu, který souběžně s výkladem realizujete. Máte-li chuť, můžete si vyzkoušet i jeho praktickou realizaci.

S H R N U T Í

Shrnutí: V tuto chvíli byste měli velmi dobře vědět, jak standardní zobrazovací řetězec funguje a jaké obrazy produkuje. Vaše znalost je nyní také podepřena jeho praktickou realizací. I když později možná dáte přednost použití nějakého profesionálního produktu (např. implementaci OpenGL nebo Direct3D), vaše námaha určitě nebyla zbytečná. Pro průpravě, kterou jste absolvovali, budete velmi dobře rozumět jak způsobu použití, tak i chování takových nástrojů. Pokud byste se chtěli problémem zabývat hlouběji, lze doporučit např. knihy

- Thomas Akenine-Moller, Eric Haines, Real-Time Rendering (2nd Edition), *A K Peters Ltd*, 2002, 900 stran, ISBN 1568811829.
- James D. Foley, Andries van Dam, Steven K. Feiner, John F. Hughes, Computer Graphics: Principles and Practice in C (2nd Edition), *Addison-Wesley Pub Co*, 1995, 1200 stran, ISBN 0-20184-840-6.

Druhá kniha má dosti široký záběr. Pokrývá prakticky celou počítačovou grafiku a lze ji považovat za velmi kvalitní učebnici. Zajímavé informace lze ale také nalézt v publikacích zabývajících se programováním počítačových her, kterých v poslední době vyšel větší počet.

3 Zobrazování metodou rekurzivního sledování paprsku

Čemu se v této kapitole naučíte?

Sledování paprsku je metodou zobrazování scén, která klade důraz na kvalitu výsledného obrázku. Kvality je dosaženo tím, že metoda bere v úvahu i složité interakce světla, ke kterým může ve scéně docházet. V úvahu jsou brány například vícenásobné odrazy světla od povrchů těles, vržené stíny, průhlednost a neprůhlednost těles a další. Výsledné obrazy mohou překvapit svojí věrností. Na obrázcích 3.1 až 3.3 jsou ukázky obrázků, jaké metoda produkuje (další ukázku jsme prezentovali již dříve na obrázku 2 v úvodu). Možná byste očekávali, že metoda schopná vytvářet tak realisticky vypadající obrázky bude komplikovaná. Opak je ale pravdou. Metoda zpětného sledování paprsku je ve své podstatě velmi jednoduchá, nepochybně jednodušší než standardní zobrazovací řetězec, který jste právě prostudovali v předchozí kapitole. Po přečtení této kapitoly budete podrobně vědět, jak metoda pracuje. To vám přinejmenším umožní kvalifikovaně používat programy, které na principu rekurzivního sledování paprsku pracují. Budete ale také schopni metodu případně i sami implementovat.

Proč by vás mohla metoda sledování paprsku zajímat?

Jaké obrázky metoda produkuje?

Ukázka 1

Obr. 3.1. Obraz generovaný metodou sledování paprsku (se svolením André Kirschner, The Internet Ray Tracing Competition, 2003).

Název „sledování paprsku“ vychází z toho, že se metoda skutečně snaží vyšetřovat „běh“ světelných paprsků ve scéně. Bude užitečné vysvětlit nejprve méně obvyklou, ale svojí podstatou intuitivně velmi dobře přijatelnou variantu této metody. Představme si světlo jako paprsky, které jsou do scény vyzařovány světelnými zdroji a putují prostorem scény. Některé paprsky mohou při své cestě dopadnout na povrchy těles ve scéně. Jiné mohou prostor scény opustit

Ukázka 2

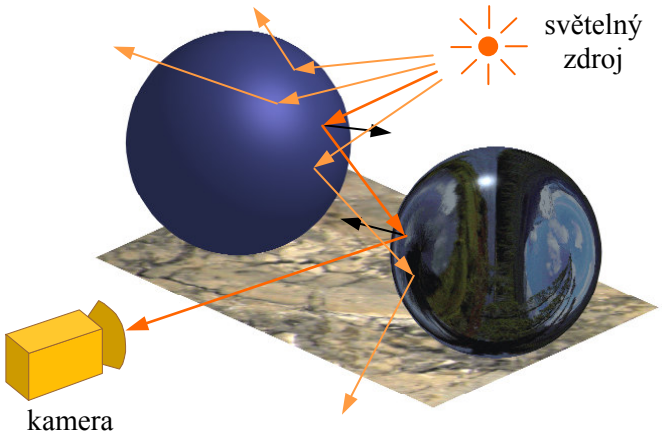
Obr. 3.2. Obraz generovaný metodou sledování paprsku (se svolením Norbert Kern, The Internet Ray Tracing Competition, 2001).

Ukázka 3

Obr. 3.3. Obraz generovaný metodou sledování paprsku (se svolením Yves Roux, The Internet Ray Tracing Competition, 2001).

(jednoduše „odlétnou“ do nekonečna). Paprsek, který dopadl na povrch tělesa, se od něj může odrazit (podle zákona odrazu) zpět do prostoru scény. Je-li těleso průhledné, pak se paprsek dopadnuvší na jeho povrch může také „zlomit“ (podle zákona lomu) a pokračovat v cestě vnitřkem tělesa. Jak paprsky odražené zpět do prostoru scény, tak paprsky procházející průhlednými tělesy mohou po čase opět dopadnout na další povrchy, kde se děj opakuje. Myšlenka, na níž je generování obrazu založeno, spočívá v tom, že se do scény vyšle velké množství paprsků ze světelných zdrojů a sleduje se jejich chod scénou. Předmětem zájmu jsou ty paprsky, které prošly objektivem myšlené kamery a dopadly na průmětnu (obr. 3.4). Za předpokladu, že máme takových paprsků dostatečný počet, pak na průmětně „nakreslí“ hledaný obraz scény. Myšlenková přijatelnost naznačeného postupu spočívá v tom, že neodporuje fyzikální představě (byť jednoduché, v níž na světlo pohlížíme jako na paprsky). Podobně si ostatně také představujeme činnost fotografického přístroje nebo kamery.

Varianta sledování paprsků od světelných zdrojů



Obr. 3.4. Běh paprsků od světelného zdroje ke kameře.

Třebaže se varianta popsaná v předchozím odstavci zdála jasná a fyzikálně v podstatě podložená, nepoužívá se jí v praxi často a je snad spíše jen předmětem experimentování. Důvodem je její obtížná realizace. Jistě vás už při prohlížení obrázku 3.4 napadlo, že pravděpodobnost toho, že se paprsek vyslaný ze světelného zdroje při své cestě prostorem scény „trefí“ právě do objektivu kamery, zasáhne průmětnu a přispěje tak ke tvorbě obrazu je velmi nízká. To je základní problém popsané varianty. Ze světelných zdrojů by bylo zapotřebí vyslat obrovské množství paprsků. Jen nepatrná část z nich by ale nakonec přispěla ke tvorbě obrazu. Průchod většiny paprsků scénou by byl sledován zcela zbytečně.

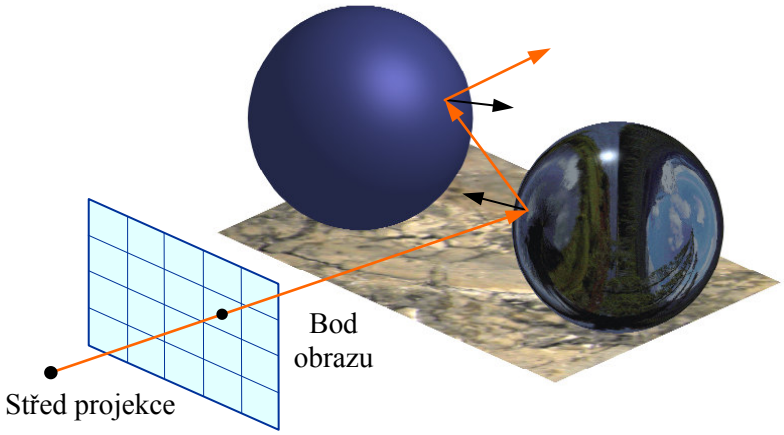
V čem je problém sledování od světelných zdrojů?

Problém naznačený v předchozím odstavci může být, zdá se, úspěšně vyřešen tak, že se chod paprsků obrátí. Předpokládá se, že obraz, který má vzniknout, je složen z obrazových bodů (pixelů). Paprsky jsou v tomto případě do scény vysílány ze středu projekce (oka) přes jednotlivé body obrazu (obr. 3.5). I tyto paprsky se při dopadu na povrch tělesa odráží zpět do prostoru scény nebo

Sledování paprsků ve směru od kamery

lámou a pokračují vnitřkem tělesa podobně, jak to již bylo popsáno v předchozí variantě. Na první pohled se takový postup může zdát ničím nepodložený. Vždyť oko ani kamera či fotografický aparát žádné podobné paprsky nevysílají. Jisté racionální vysvětlení postupu však existuje. Na vysílané paprsky se můžeme dívat tak, že se pomocí nich pokoušíme vytvořit cestu mezi středem projekce a světelným zdrojem. Jestliže se takovou cestu podaří vytvořit, pak se v opačném směru, tedy od světelného zdroje směrem ke kameře, vyhodnotí barva a intenzita světla (a proti postupu od světelného zdroje přece nic nenamítáme). Zajímavé jsou proto v tomto případě zejména takové paprsky, které při své cestě od kamery zasáhnou nějaký světelný zdroj.

Přísně vzato, mohli byste nyní namítnout, že pravděpodobnost jevu, že paprsek vyslaný od kamery zasáhne světelný zdroj, bude podobně malá jako pravděpodobnost jevu, že paprsek vyslaný od světelného zdroje proletí objektivem a dopadne na průmětnu kamery. Teoreticky máte pravdu. Prakticky ale dává sledování paprsku od kamery větší prostor k různým velmi přijatelným zjednodušením, která nakonec vedou k tomu, že paprsek světelný zdroj přímo zasáhnout nemusí. Nejběžnější postup výpočtu podrobně ukážeme v následující podkapitole. (Už zde si ale můžete alespoň předběžně světelný zdroj představit jako dostatečně velkou zářící plochu, kterou pak není tak obtížné zasáhnout.)



Obr. 3.5. Sledování paprsků ve směru od kamery.

Úkol 3.1

Úkol 3.1. Prohlédněte si obrázky, které naleznete na přiloženém CD v adresáři „grafikaII\raytrac\obrázky". Uvidíte, co od metody sledování paprsku můžete očekávat. Seznamte se také s obsahem adresáře „grafikaII\raytrac\student_programy", kde naleznete ukázkové studentské programy přibližně takové úrovně, jaké byste sami mohli vcelku snadno dosáhnout, pokud byste se během studia rozhodli pro realizaci vlastního jednoduchého výukového programu pro sledování paprsku. □

3.1 Algoritmus sledování paprsku

Doba studia asi 1,5 hod

V této podkapitole se seznámíte se základními principy algoritmu sledování paprsků ve směru od kamery. Ty pak budou v následujících podkapitolách doplněny dalšími detaily. Algoritmus je poměrně jednoduchý, a proto můžeme ihned přejít k jeho popisu. Předpokládáme, že scéna je zadána jako seznam objektů. Objekty mohou být vcelku libovolné. Důležité ale je, že musíte být schopni vypočítat průsečík (průsečíky) objektu s paprskem (tj. s přímkou). V místě průsečíku bude také potřebné stanovit normálu povrchu. Uvedené výpočty jsou nejsnazší pro objekty ve tvaru koule. Kromě popisu scény je zapotřebí znát také popis osvětlení scény (seznam světelných zdrojů) a popis zobrazení (jak se na scénu chcete dívat). Generování paprsků se zjednoduší, provedeme-li transformaci souřadné soustavy tak, aby se střed promítání stal počátkem souřadné soustavy a zobrazovací rovina aby byla kolmá k ose *z*. K realizaci uvedené transformace postačí translace a rotace. Přesný postup provedení transformace by měl být jasný, protože s transformacemi již máte bohaté zkušenosti z předchozích dvou kapitol. Zde proto budeme jednoduše předpokládat, že transformace je již provedena. Algoritmus lze nyní formulovat následovně (obr. 3.5).

Čemu se zde naučíte?

```
{Origin a Direction určují počátek a směr paprsku.}
Origin ← Střed promítání;
for each  image point (x,y)  do
   Direction ← směr paprsku z Origin do pixelu (x,y);
   Image(x,y) ← Ray_Trac( Origin, Direction, 0 );
konec;
```

Přes každý obrazový bod se do scény vyšle paprsek.

Z výše uvedeného kódu vidíte, že algoritmus jednoduše postupně do scény vysílá paprsky vedené ze středu projekce ve směru přes obrazové body (*x*, *y*). Tyto paprsky obvykle bývají nazývány primárními. Každý paprsek je popsán svým počátečním bodem (Origin) a směrem (Direction). U primárního paprsku je jeho počátkem střed projekce. Směrem je směr k obrazovému bodu (*x*, *y*). Cesta vyslaného paprsku ve scéně je sledována funkcí Ray_Trac. Ke sledování funkce potřebuje seznam objektů scény a také seznam světelných zdrojů. Jako funkční hodnotu funkce vrací barvu (obvykle barevné složky *r*, *g*, *b*), kterou se má obrazový bod (*x*, *y*) „rozsvítit“. Je zřejmé, že většinu práce algoritmu vykoná právě funkce Ray_Trac. I tuto funkci můžeme ale formulovat poměrně jednoduše.

Cesta paprsku ve scéně se analyzuje funkcí Ray_Trac.

```
{TCoord3 je vektor souřadnic o třech složkách.}
{TRGB je vektor barevných složek r, g, b.}
{Origin a Direction určují počátek a směr paprsku.}
{Nest omezuje počet dopadů/odrazů – viz následující text.}

function Ray_Trac( Origin, Direction: TCoord3;
                   Nest: Integer): TRGB;

   Jestliže (Nest > MaxNest), vrať barvu (0,0,0) a skonči;
   Najdi průsečíky paprsku (Origin, Direction) s objekty;
   Nebyl-li nalezen žádný průsečík, vrať (0,0,0) a skonči;
   X ← průsečík nejbližší bodu Origin (ve směru paprsku);
   Reflected_Dir ← směr paprsku odraženého v X;
   Ir = Ray_Trac( X, Reflected_Dir, Nest+1 );
```

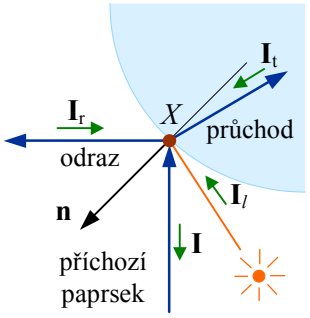
```
    Transmitted_Dir ← směr paprsku procházejícího v X;
    It = Ray_Trac( X, Transmitted_Dir, Nest+1 );
    Il = intenzita v bodě X od světelných zdrojů;
    vrať barvu (Il + kr Ir + kt It);
konec;
```

Funkce Ray_Trac hledá průsečíky paprsku zadaného svým počátkem (Origin) a směrem (Direction) s objekty scény, jejichž seznam jí proto musí být přístupný. Jestliže není nalezen žádný průsečík, znamená to, že vyšetřovaný paprsek opustil prostor scény. Nebude proto nijak přispívat k výslednému obrazu. Tomu také odpovídá hodnota (0, 0, 0) barevných složek, kterou funkce v tomto případě vrací. V případě, že naopak existuje více průsečíků paprsku s povrchy těles, je nutné vybrat průsečík, který je po směru paprsku nejbližší jeho počátku (průsečíky, k nimž by bylo nutné od počátku paprsku postupovat proti jeho směru, se neuvažují). Nechť *X* označuje takový průsečík. Procedura Ray_Trac počítá v bodě *X* směr odraženého paprsku (Reflected_Dir) a vyšle odražený paprsek znovu do scény rekurzivním voláním sama sebe. Jestliže bod *X* leží na povrchu tělesa, které je průhledné nebo alespoň částečně průhledné, pak procedura Ray_Trac spočítá také směr procházejícího paprsku (Transmitted_Dir) a také i ten vyšle znovu do scény, a to opět rekurzivním voláním sama sebe. Počátkem pro oba nově vyslané paprsky je bod *X*. Běh paprsku ve scéně je ilustrován na obr 3.6, 3.7.

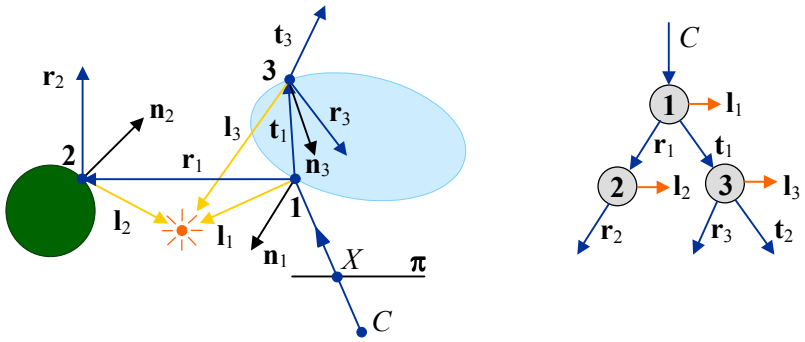
Po dopadu paprsku na povrch tělesa je vyslán paprsek odražený a procházející.

Přemýšlejme, jak se proces neustálého rekurzivního generování dalších a dalších paprsků zastaví. Jednu možnost již známe, a to, že paprsek opustí prostor scény. Jen to by ovšem obecně nemohlo zajistit, že se generování paprsků vůbec někdy ukončí. Aby bylo možné proces generování paprsků ovládat, je funkce Ray_Trac vybavena parametrem Nest, který počítá úroveň rekurzivního zanoření. Úroveň zanoření je omezena hodnotou MaxNest. Je-li této úrovně dosaženo, pak se již další paprsky (ani odražené ani procházející) negenerují. To má také docela rozumnou praktickou interpretaci. Jestliže by počet odrazů (lomů) paprsku měl být větší než hodnota MaxNest, pak nepředpokládáme, že by po tak složité (tolikrát zalomené) cestě mohl do vyšetřovaného místa obrazu přicházet významný příspěvek intenzit barevných složek.

Počet odrazů nebo lomů paprsku je omezen.



Obr. 3.6. K činnosti funkce Ray_Tac: Paprsek dopadá na povrch tělesa v bodě *X*. V tomto místě je pak generován paprsek odražený a paprsek procházející tělesem (je-li těleso průhledné). Hodnoty $\mathbf{I}_r$, $\mathbf{I}_t$ jsou intenzity „vracející se“ ze směru odraženého resp. procházejícího paprsku. Hodnota $\mathbf{I}_l$ je lokální složka osvětlení.



Obr. 3.7. Průchod paprsku scénou (vlevo) a znázornění průchodu schématem ve tvaru stromu (vpravo). *C* je střed projekce a *X* je bod obrazu. Body 1, 2, 3 jsou body dopadu paprsků. Veličiny $\mathbf{n}_i$, $\mathbf{r}_i$, $\mathbf{t}_i$, $\mathbf{l}_i$ jsou postupně normála povrchu, odražený a procházející paprsek a směr ke světelnému zdroji.

Výpočet osvětlení

Zabývejme se konečně ještě tím, jak se v místech každého dopadu paprsku počítají parametry osvětlení. Uvedli jsme již dříve, že na smysl vysílání paprsků od kamery můžeme pohlížet tak, že je pomocí nich hledána možná cesta ke zdrojům světla. Je-li cesta nalezena, pak teprve probíhá výpočet „kolik energie“ po této cestě od zdrojů do vyšetřovaného místa obrazu přichází. Ve funkci Ray_Trac je to prakticky zajištěno tím, že se intenzity počítají až v okamžiku, když dochází k „vynořování“ z rekurze. K vynoření dojde tehdy, jestliže je již nějaká část cesty paprsku ve směru od zdroje vyřešena. Uvažujme situaci na obr. 3.6. Do bodu *X* dopadl paprsek. Z bodu *X* byl proto vyslán paprsek odražený a paprsek procházející. Funkce Ray_Trac „počká“, až se vyřeší cesty těchto vyslaných paprsků a jejich potomků ve scéně. Jakmile jsou vyřešeny, je zřejmě možné spočítat, jaká je barva a intenzita světla, které po nalezených cestách putuje prostřednictvím obou vyslaných paprsků zpět do bodu *X*. Intuitivně ovšem cítíme, že osvětlení v bodě *X* bude dáno nejen osvětlením přicházejícím podél odraženého a procházejícího paprsku. Jestliže jsou ve scéně světelné zdroje, které na bod *X* svítí, pak se také ony mohou na osvětlení bodu spolupodílet. Můžeme nyní shrnout, že barvu a intenzitu světla, které cestuje z bodu *X* zpět po paprsku, který do něj dopadl, se zdá rozumné počítat podle vztahu

Osvětlení v bodě X je součtem příspěvku od světelných zdrojů a příspěvků od odraženého a procházejícího paprsku.

$$\mathbf{I} = \mathbf{I}_l + k_\mathrm{r}\mathbf{I}_\mathrm{r} + k_\mathrm{t}\mathbf{I}_\mathrm{t} \ . \tag{3.1}$$

Význam hodnot v uvedeném vztahu je následující: **I** popisuje výsledné osvětlení, které se vrací zpět po paprsku dopadnuvším do bodu *X* a které bude později sloužit pro výpočet osvětlení v bodě, odkud byl paprsek do bodu *X* vyslán. Hodnota **I** spočítaná v místě dopadu primárního paprsku se použije pro „rozsvícení“ odpovídajícího obrazového bodu. $\mathbf{I}_l$ popisuje přímé osvětlení bodu *X* od světelných zdrojů. $\mathbf{I}_\mathrm{r}$, $\mathbf{I}_\mathrm{t}$ popisují osvětlení, které se do bodu *X* „vrátilo“ ze směru odraženého paprsku resp. paprsku procházejícího. Hodnoty k_r, k_t jsou koeficienty beroucí v úvahu ztrátu, ke které dojde při odrazu paprsku nebo při jeho přechodu z jednoho prostředí do druhého. Jejich hodnoty musí být zadány ($0 \le k_\mathrm{r}, k_\mathrm{t} \le 1$). Výpočet osvětlení se provádí po barevných složkách *r*, *g*, *b*, a proto jsou **I**, $\mathbf{I}_l$, $\mathbf{I}_\mathrm{r}$, $\mathbf{I}_\mathrm{t}$ tříprvkové vektory.

Zbývá uvést, jak se v bodě X počítá osvětlení od světelných zdrojů. Velmi často se i v tomto případě používá postupu, který jsme již popsali v podkapitole 2.5. Můžeme proto ihned zapsat výsledný vztah. Máme

Výpočet lokálního příspěvku od světelných zdrojů

$$\mathbf{I}_l = \mathbf{I}_\mathrm{a}\mathbf{O}_\mathrm{a} + \sum_i S_i f_{att_i} \mathbf{I}_i \left(\mathbf{O}_\mathrm{d} \cos\varphi_i + \mathbf{O}_\mathrm{s} \cos^n \alpha_i\right). \qquad (3.2)$$

Význam symbolů $\mathbf{I}_\mathrm{a}$, $\mathbf{O}_\mathrm{a}$, $f_{\mathrm{att}i}$, $\mathbf{I}_i$, $\mathbf{O}_\mathrm{d}$, φ_i a $\mathbf{O}_\mathrm{s}$ jsme vysvětlili již v podkapitole 2.5. Na rozdíl od podkapitoly 2.5 jsme zde ale pro stručnost použili vektorového zápisu. Součiny vektorů (např. $\mathbf{I}_\mathrm{a}\mathbf{O}_\mathrm{a}$) je zde zapotřebí chápat tak, že se násobí odpovídající si složky vektorů. Hodnotu S_i jsme nyní do výpočtu osvětlení zavedli nově. S_i nabývá hodnoty 1 nebo 0 podle toho, zda je i-tý světelný zdroj z bodu X viditelný či nikoli. Není-li viditelný, pak na bod X nezáří, a jeho účinek se proto nebere v úvahu. Při praktické implementaci nepředstavuje zjištění hodnoty S_i velkou komplikaci, protože vyžaduje jen ty nástroje, které již v programu stejně jsou pro řešení cesty paprsků. Ty se v tomto případě jednoduše použijí tak, že se vyšle paprsek z bodu X ke světelnému zdroji a nalezne se jeho průsečík s nejbližším neprůhledným tělesem. Jestliže průsečík existuje a leží-li uvnitř úsečky spojující X se světelným zdrojem, pak zdroj na bod X nezáří a hodnota S_i je $S_i = 0$. (Nyní by mělo být jasné, proč jsme koeficient S_i nepoužívali i v podkapitole 2.5. Při jeho použití současně s Phongovým stínováním by sice zajistil vytváření pěkných vržených stínů, ale současně by bohužel také dramaticky narostla časová složitost výpočtu. Při použití se stínováním Gouraudovým by vržené stíny navíc stejně nejspíše vůbec ani nebyly pěkné, protože se barva po ploše interpoluje.)

Úkol 3.2

Úkol 3.2. Prohlédněte si internetové stránky www.povray.org, které prezentují velmi známý a úspěšný program POV-Ray pracující technikou sledování paprsku (navštivte v této souvislosti také stránky megapov.inetart.net). Můžete také navštívit stránky www.irtc.org, kde najdete mnoho pěkných obrázků. □

3.2 Řešení dílčích úloh při sledování paprsku

Doba studia asi 1,5 hod

V této podkapitole se seznámíte s tím, jak lze řešit úlohy, jejichž existence byla naznačena v předchozí podkapitole při popisu algoritmu jako celku. Bude se jednat zejména o úlohu nalezení průsečíků paprsku s tělesy a o úlohu nalezení směru odraženého a procházejícího paprsku. Tato podkapitola vás dostatečně teoreticky vybaví k realizaci vlastního cvičného programu, pokud se pro ni rozhodnete. Je ale užitečné si podkapitolu přečíst i v případě, že hodláte používat pouze programy hotové. Ukáže vám, jaké problémy musí program řešit, a vy pak budete velmi přesně vědět, proč generování obrazu vyžaduje tolik času.

Čemu se zde naučíte?

K řešení úlohy o protínání paprsku s tělesy si nejprve připravme rovnici paprsku. Nechť $\mathbf{x}_0$ je vektor popisující polohu počátku paprsku (v předchozích pseudokódech algoritmu jsme jej nazývali poněkud zdlouhavě Origin) a $\Delta\mathbf{x}$ nechť je vektor určující směr paprsku (Direction); t nechť je parametr. Předpokládejme, že platí $|\Delta\mathbf{x}| = 1$, což zjednoduší vztahy, které budou následovat. Rovnici paprsku můžeme napsat ve tvaru (zapíšeme nejprve vektorově a pak po složkách)

Rovnice paprsku

$$\mathbf{x} = \mathbf{x}_0 + t\Delta\mathbf{x}\ , \qquad (3.3)$$

$$x = x_0 + t\Delta x\,, \quad y = y_0 + t\Delta y\,, \quad z = z_0 + t\Delta z\ .$$

Průsečík paprsku s kulovou plochou

Začněme řešením průsečíku paprsku s kulovou plochou, protože to je nejjednodušší případ. Předpokládejme, že střed plochy leží v bodě o souřadnicích (a, b, c) a rovnici plochy zapišme ve tvaru

$$(x-a)^2 + (y-b)^2 + (z-c)^2 - r^2 = 0\ . \tag{3.4}$$

Dosazením vztahů (3.3) do rovnice (3.4) dostaneme pro parametr t rovnici

$$At^2 + Bt + C = 0\ , \tag{3.5}$$

v níž A, B, C jsou následující hodnoty

$$\begin{aligned} A &= \Delta x^2 + \Delta y^2 + \Delta z^2 = 1\ , \\ B &= 2\left[\Delta x(x_0 - a) + \Delta y(y_0 - b) + \Delta z(z_0 - c)\right]\ , \\ C &= (x_0 - a)^2 + (y_0 - b)^2 + (z_0 - c)^2 - r^2\ . \end{aligned} \tag{3.6}$$

Hledanou hodnotu t snadno určíme užitím formule pro řešení kvadratických rovnic, kterou znáte už ze střední školy. Máme

$$D \equiv B^2 - 4AC\,, \quad t = \frac{-B \pm \sqrt{D}}{2A}\ . \tag{3.7}$$

Nalezení nejbližšího průsečíku

V závislosti na hodnotě diskriminantu D můžeme rozlišit tři případy. 1) $D < 0$: Paprsek plochu neprotíná. Tento případ bude nastávat velmi často. Zpracování plochy tímto zjištěním končí. 2) $D = 0$: paprsek se plochy dotýká v jednom bodě. Také v tomto případě lze zpracování plochy ukončit. 3) $D > 0$: paprsek plochu protíná ve dvou bodech. Zajímavý je jen ten bod, pro nějž je t kladné a menší z obou hodnot. Tato hodnota odpovídá tomu průsečíku, který je bližší počátku paprsku (záporná hodnota t by znamenala, že cesta do průsečíku vede proti směru paprsku, a proto ji vylučujeme). Normálu v místě průsečíku, kterou budeme potřebovat k výpočtu osvětlení, získáme pro kulovou plochu jednoduše jako rozdíl souřadnic průsečíku a středu plochy.

Průsečík polygonu s tělesy ve tvaru mnohostěnu

Zaměřme se nyní na řešení úlohy o nalezení průsečíků paprsku s tělesy ve tvaru mnohostěnů s rovinnými stěnami. Zjistíme (možná trochu překvapivě), že řešení není jednodušší než u kulové plochy. Spíše naopak. Předpokládáme, že tělesa máme popsána jejich povrchem tak, že pro každou stěnu tělesa můžeme z popisu získat její rovnici. Dále můžeme získat posloupnost vrcholů definující polygon, který stěnu ohraničuje. Průsečík paprsku s tělesem hledáme tak, že prověřujeme všechny stěny tělesa. Předpokládejme, že rovina stěny má tvar

$$Ax + By + Cz + D = 0\ . \tag{3.8}$$

Dosazením vztahů (3.3) do rovnice (3.8) snadno pro t dostaneme řešení

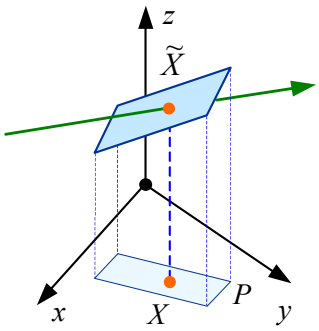
$$t = -\frac{Ax_0 + By_0 + Cz_0 + D}{A\Delta x + B\Delta y + C\Delta z}\ . \tag{3.9}$$

Nejprve se najde průsečík paprsku s rovinou stěny.

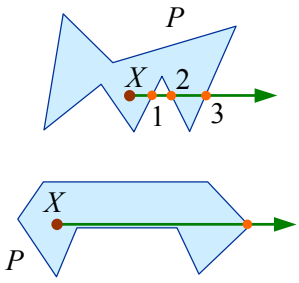
Vyjde-li jmenovatel výrazu na pravé straně rovnice nula, pak je paprsek se stěnou rovnoběžný. Jinak rovinu stěny protíná. V tom případě je pak ale ještě nutné určit, zda průsečík paprsku s rovinou leží uvnitř nebo vně polygonu, který stěnu ohraničuje, a zda tedy je či není skutečným průsečíkem s tělesem. Úlohu můžeme

převést na dvojrozměrnou tak, že vynecháme souřadnici, která odpovídá největší z hodnot $|A|$, $|B|$, $|C|$. Vynecháním souřadnice provedeme projekci úlohy do některé ze souřadnicových rovin (obr. 3.8). Po provedení projekce zbývá řešit následující úlohu: Je dán jednoduchý polygon P v rovině a bod X. Leží X uvnitř P? Obecné řešení je následující: Zvolíme libovolnou polopřímku tak, aby bod X byl jejím koncovým bodem. Vypočítáme průsečíky polopřímky s hranicí polygonu. Jestliže je počet průsečíků lichý, pak X leží uvnitř polygonu (obr. 3.9). V opačném případě leží vně. Tento postup je poněkud nepříjemný jednak proto, že hrozí velkou časovou složitostí (časová složitost nejhoršího případu a také časová složitost střední závisí lineárně na počtu vrcholů polygonu), a také proto, že je při praktické implementaci nutné pamatovat na možnost vzniku jistých speciálních situací, které naznačuje obr 3.9. Lepší algoritmus ovšem neexistuje a ani existovat nemůže (ve smyslu časové složitosti). Jedinou možností, jak lze problém řešit jednodušeji, je slevit z předpokladů na obecnost polygonu. Pro konvexní polygony lze např. řešení najít v čase logaritmicky závislém na počtu vrcholů (podrobný popis by ale přesáhl rámec tohoto textu). Jednoduché řešení lze také najít pro speciální polygony, jako jsou například trojúhelníky nebo obdélníky. V příkladě 3.3 ukážeme, jak lze postupovat v případě trojúhelníků. Výpočet normály stěny je jednoduchý. Vypočteme ji pomocí vektorového součinu hran polygonu ohraničujícího stěnu.

Ověří se, zda průsečík leží uvnitř polygonu, který stěnu ohraničuje.



Obr. 3.8. K hledání průsečíku paprsku s mnohostěnem.



Obr. 3.9. K určení vzájemné polohy bodu a polygonu.

Hledání průsečíku paprsku s tělesem, jehož povrch je tvořen obecnou plochou popsanou implicitní rovnicí tvaru $f(x, y, z) = 0$, je snadné, jestliže plocha není stupně vyššího než třetího (řešení pro kulovou plochu, kterým jsme tuto podkapitolu začali, je ostatně speciálním případem této úlohy). Dosazením výrazu (3.3) do rovnice povrchu snadno získáme rovnici $F(t) = 0$, kterou řešíme pro t. Kořeny polynomu do třetího stupně včetně lze snadno získat analyticky. Nalezení kořenů polynomu druhého stupně je všeobecně známé. Pro nalezení kořenů polynomů třetího stupně lze pak využít tzv. Cardanových vzorců (naleznete je v podrobnějších učebnicích matematiky). Normála plochy se vypočte jako vektor derivací $(\partial f/\partial x, \partial f/\partial y, \partial f/\partial z)$ v místě průsečíku. Hledání průsečíku paprsku s plochou vyššího než třetího stupně lze považovat za problém, protože v tomto případě je obecně nutné použít numerického iteračního řešení.

Hledání průsečíku s plochou popsanou rovnicí tvaru $f(x, y, z) = 0$

Nalezení průsečíku s plochou popsanou parametricky ve tvaru $\mathbf{f}(u, v)$ lze provést řešením soustavy tří rovnic tvaru $\mathbf{x}_0 + t\Delta\mathbf{x} = \mathbf{f}(u, v)$ pro hodnoty t, u, v. Praktická schůdnost postupu závisí na konkrétním tvaru funkce $\mathbf{f}(u, v)$. V mnoha

Hledání průsečíku s plochou popsanou rovnicí tvaru **x=f**(u, v)

praktických případech je hodnota parametrů u, v omezená, např. podmínkou $0 \le u, v \le 1$ (plocha je tedy ohraničená). V takovém případě je pak zapotřebí zkontrolovat, zda nalezené souřadnice u, v průsečíku leží v předepsaných mezích a zda tedy paprsek plát plochy vůbec protíná. Normálu v místě průsečíku lze spočítat pomocí vektorového součinu $\partial\mathbf{f}(u, v)/\partial u \times \partial\mathbf{f}(u, v)/\partial v$.

Implementaci provádějte s ohledem na rychlost!

K výpočtu průsečíků na závěr ještě dvě praktické poznámky: Scéna zpravidla obsahuje více těles. Z hlediska rychlosti výpočtu, která je u metody sledování paprsku kritická, je vhodné nejprve vypočítat hodnoty t pro všechna tělesa. Pak vybrat nejmenší kladnou z nich (tedy najít průsečík, který je nejblíže počátku paprsku) a pak teprve podle rovnice (3.3) počítat souřadnice průsečíku a dále normálu k ploše. 2) Jestliže se rozhodnete pro realizaci vlastního cvičného programu, možná se setkáte s problémem, že vám algoritmus bude jako nejbližší průsečík vybírat samotný bod, ze kterého byl paprsek vyslán. Na vině tomu budou numerické nepřesnosti, které mohou způsobit, že se pro počátek paprsku zjistí hodnota t větší než nula. Problém můžete vyřešit jednoduše tak, že budete požadovat, aby t bylo větší než nějaká malá kladná hodnota ε.

Výpočet směru odraženého paprsku

Abyste o řešení dílčích úloh, které při sledování paprsku vyvstávají, byli informování kompletně, musíme se ještě zmínit o výpočtu směru odraženého a procházejícího paprsku. Nechť **n** je normála povrchu v místě dopadu paprsku, nechť **a** značí směr dopadajícího paprsku a nechť **r**, **t** jsou směry odraženého resp. procházejícího paprsku. Předpokládejme, že $|\mathbf{n}| = |\mathbf{a}| = 1$. Máme pak (obr. 3.10)

$$\mathbf{q} = \mathbf{n}\cos\varphi_1, \quad \mathbf{s}_1 = \mathbf{q} + \mathbf{a}, \quad \mathbf{r} = \mathbf{q} + \mathbf{s}_1 = 2\mathbf{n}\cos\varphi_1 + \mathbf{a} \ . \tag{3.10}$$

Výpočet směru procházejícího paprsku

Pro výpočet směru procházejícího paprsku vyjdeme ze zákona lomu známého z fyziky. Nechť η_1, η_2 jsou indexy lomu světla (připomeňme, že indexy lomu jsou poměrem rychlosti světla ve vakuu ku rychlosti v uvažovaném materiálu). Platí

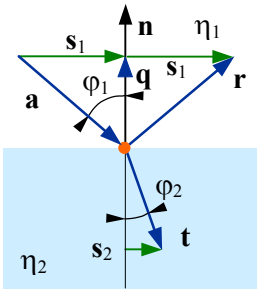
$$\frac{\sin\varphi_1}{\sin\varphi_2} = \frac{\eta_2}{\eta_1} \ . \tag{3.11}$$

Pro určení směru procházejícího paprsku vyjdeme z rovnice $\mathbf{s}_1/|\mathbf{s}_1| = \mathbf{s}_2/|\mathbf{s}_2|$ (obr. 3.10) a z požadavku $|\mathbf{t}| = 1$. Rozepsáním dostaneme

$$\frac{(\mathbf{n}\cos\varphi_1 + \mathbf{a})}{\sin\varphi_1} = \frac{(\mathbf{n}\cos\varphi_2 + \mathbf{t})}{\sin\varphi_2} \ . \tag{3.12}$$

Řešíme-li rovnici pro hledaný směr **t** procházejícího paprsku, dostáváme

$$\mathbf{t} = \frac{\sin\varphi_2}{\sin\varphi_1}(\mathbf{n}\cos\varphi_1 + \mathbf{a}) - \mathbf{n}\cos\varphi_2 = \frac{\eta_1}{\eta_2}(\mathbf{n}\cos\varphi_1 + \mathbf{a}) - \mathbf{n}\cos\varphi_2 \ . \tag{3.13}$$

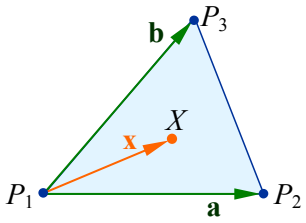


Obr. 3.10. K určení směru odraženého a procházejícího paprsku.

Příklad 3.3

Příklad 3.3. Navrhněte postup umožňující rozhodnout, zda zadaný bod X leží uvnitř či vně trojúhelníka popsaného svými vrcholy P_1, P_2, P_3.

Řešení. Ačkoli nás při sledování paprsku zajímá trojrozměrná varianta uvedené úlohy, postupem uvedeným v předchozím výkladu můžeme přejít k variantě dvojrozměrné. Na její řešení se zaměříme. Zaveďme vektory $\mathbf{a} = P_2 - P_1$, $\mathbf{b} = P_3 - P_1$, $\mathbf{x} = X - P_1$, jak je nakresleno na následujícím obrázku.



Vektor $\mathbf{x}$ můžeme vyjádřit ve tvaru $\mathbf{x} = \alpha\mathbf{a} + \beta\mathbf{b}$. Bod X leží uvnitř trojúhelníka právě tehdy, když platí $\alpha \geq 0$, $\beta \geq 0$ a současně $\alpha + \beta \leq 1$. Předpokládejme, že je $\mathbf{a} = (a_1, a_2)$, $\mathbf{b} = (b_1, b_2)$, $\mathbf{x} = (x_1, x_2)$. Hodnoty α, β pak stanovíme z výše uvedené rovnice $\mathbf{x} = \alpha\mathbf{a} + \beta\mathbf{b}$ (jedná se o soustavu dvou rovnic zapsaných ve vektorovém tvaru). Řešením soustavy snadno zjistíme, že platí

$$\alpha = \frac{b_2 x_1 - b_1 x_2}{a_1 b_2 - a_2 b_1}, \quad \beta = \frac{a_1 x_2 - a_2 x_1}{a_1 b_2 - a_2 b_1}.$$

□

Příklad 3.4

Příklad 3.4. Navrhněte postup umožňující rozhodnout, zda zadaný bod X leží uvnitř či vně konvexního polygonu zadaného svými vrcholy P_1, P_2, … , P_n.

Řešení. Opět se zaměříme na dvojrozměrnou variantu úlohy. Konvexní n-úhelník lze vždy jednoduše rozdělit na $n - 2$ trojúhelníků (nakreslete si obrázek). Úlohu proto můžeme řešit tak, že všechny tyto trojúhelníky postupně prověřujeme postupem popsaným v předchozím úkolu. Časová složitost tohoto řešení závisí na n lineárně. Pro úplnost dodejme, že existuje i řešení se závislostí logaritmickou. Jeho podrobný popis by ale přesáhl rámec tohoto textu. □

S H R N U T Í

Shrnutí: V tuto chvíli by vám měl být jasný jak princip metody sledování paprsku, tak i hlavní detaily potřebné k její realizaci. Chybí vám už jen informace, jak zajistit, aby algoritmus běžel co možná nejrychleji, a informace o tom, jak bojovat s nepříjemným jevem, tzv. aliasingem, který snižuje kvalitu obrázku.

3.3 Metody urychlování algoritmu sledování paprsku

Doba studia asi 1,5 hod

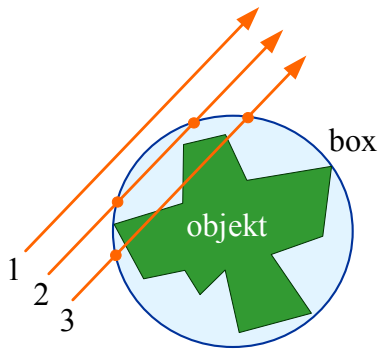
Čemu se zde naučíte?

Metoda sledování paprsku je náročná na výpočetní výkon. Nejvíce času je zapotřebí pro hledání průsečíků paprsků s objekty scény. Výsledkem výpočtu je při tom často pouze zjištění, že se paprsek a objekt neprotínají. Právě hledání průsečíků je proto žádoucí zefektivnit. V této podkapitole se dovíte, jak to udělat. Základními postupy jsou: 1) použití ohraničujících ploch („boxů“) (případně včetně jejich organizování do hierarchických struktur), 2) dělení prostoru na podprostory. Uvedené metody dále postupně popíšeme. Kromě toho uvedeme i některé postupy další.

Použití ohraničujících ploch

Nejjednodušším opatřením směřujícím ke zvýšení rychlosti metody je využití ohraničujících ploch („bounding boxů“). Ohraničující plocha se vytvoří ke každému objektu ve scéně. Vhodná je taková plocha, která má následující vlastnosti: 1) Objekt vždy leží celý uvnitř své ohraničující plochy, ale plocha současně objekt obepíná co nejtěsněji (obr. 3.11). 2) O existenci průsečíků paprsku s ohraničující plochou musí být možně rozhodnout co nejjednodušším výpočtem. 3) Plochu musí být možné pro jednotlivé objekty dostatečně jednoduše nalézt. Konkrétním případem vhodné ohraničující plochy je plocha kulová. O existenci průsečíků lze v tomto případě rozhodnout, jak víme z předchozí podkapitoly, velmi jednoduše. Jistou nevýhodou ale je, že kulová plocha ne vždy může objekt obepínat těsně (představte si například, že je objekt protáhlý). Také nalezení středu a poloměru plochy pro každý objekt si vyžádá něco málo programátorského a později i výpočetního času. Jinou ohraničující plochou může být povrch kvádru se stěnami rovnoběžnými se souřadnicovými rovinami. V tomto případě je ale rozhodnutí existence průsečíku s paprskem o něco pracnější než u plochy kulové (příklad 3.5). Ani v tomto případě není vždy možné, aby ohraničující plocha obepínala objekt těsně (představte si protáhlý objekt umístěný ve scéně „na šikmo“). Nalezení ohraničující plochy je však v tomto případě velmi jednoduché. Pokud je objektem mnohostěn, stačí projít jeho vrcholy a nalézt minimální a maximální hodnoty jednotlivých souřadnic.

Vhodné plochy jsou např.: kulová plocha, povrch kvádru.

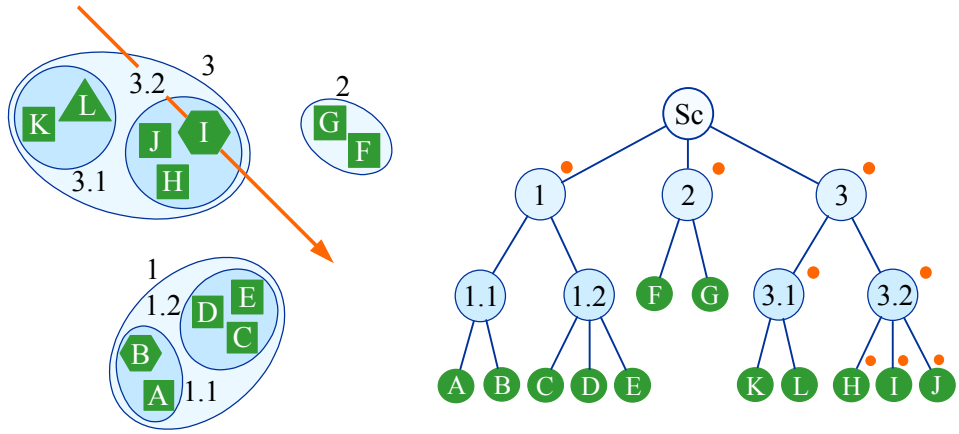


Obr. 3.11. Využití ohraničující plochy: Protože paprsek 1 ohraničující plochu neprotíná, určitě nemůže protínat ani objekt ležící uvnitř. Paprsky 2, 3 ohraničující plochu protínají, a vyžadují proto podrobné řešení průsečíků se samotným objektem.

Myšlenka použití ohraničujících ploch spočívá v následujícím. Kdykoli má být počítán průsečík paprsku s objektem, pak se nejprve zjišťuje, zda vůbec protíná ohraničující plochu. Jestliže ne, pak je jisté, že nemůže protínat ani objekt, který leží uvnitř. Průsečíky paprsku s objektem proto není zapotřebí počítat. K tomu, že paprsek vyšetřovaný objekt neprotíná, dochází obecně velmi často. Včasné odhalení této situace je podstatou urychlení při použití ohraničujících ploch. Jestliže naopak paprsek ohraničující plochu protíná, pak je zapotřebí vypočítat i průsečíky se samotným objektem ležícím uvnitř ohraničující plochy. Ty mohou, ale také nemusí existovat. Paprsek 2 na obr. 3.11, který protíná ohraničující plochu, ale nikoli objekt ležící uvnitř, ukazuje, proč je žádoucí, aby ohraničující plocha obepínala objekt pokud možno těsně.

Hierarchicky uspořádané ohraničující plochy

Myšlenku ohraničujících ploch lze dále zdokonalit jejich organizováním do hierarchických struktur (obr. 3.12). Ať se již jedná o ohraničující plochu kterékoli hierarchické úrovně, zůstává základní myšlenka stále tatáž. Zjistí-li se, že paprsek ohraničující plochu neprotíná, pak nejsou hledány ani průsečíky s žádnými jejími synovskými útvary (dalšími ohraničujícími plochami nebo objekty). Činnost metody je ilustrována na obrázku 3.12, kde je také dobře vidět, jak k urychlení dochází. Skutečnost, že lze při počítání průsečíků paprsku s objekty scény vynechávat celé skupiny objektů, vypadá slibně. Nedostatkem metody ale je, že obecně není jednoduché hierarchickou strukturu umožňující efektivní výpočet automatizovaně nalézt.



Obr. 3.12. Hierarchická struktura ohraničujících ploch a její reprezentace stromem (vpravo). Písmeny jsou označeny objekty scény, čísly ohraničující plochy. Tečky u uzlů stromu ukazují, se kterými ohraničujícími plochami a se kterými tělesy bylo zapotřebí počítat průsečíky znázorněného paprsku. Kořen stromu (označený jako Sc) odpovídá celé scéně.

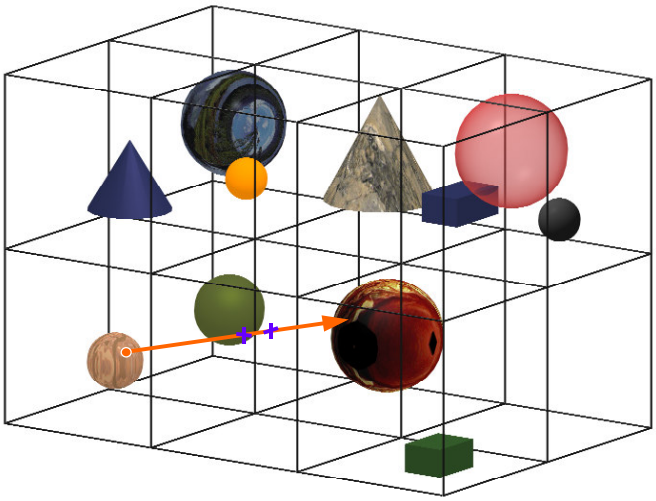
Dělení prostoru scény na podprostory

Metodou, která se pro urychlení sledování paprsku používá velmi často, je metoda dělení prostoru scény na podprostory. Obvykle se prostor dělí rovinami rovnoběžnými s rovinami *xy*, *yz*, *zx* souřadné soustavy. Objemové elementy prostoru vzniklé takovým dělením mají tvar kvádrů. Dělení lze provést tak, že jsou elementy buď stejně nebo různě velké. Princip metody ukážeme na jednodušší variantě, kdy je velikost elementů stejná (obr. 3.13). Základní myšlenka je tato: Zatímco u neurychleného sledování paprsku byly všechny objekty scény organizovány v jednom jediném „velkém“ seznamu, bude nyní zřízeno tolik seznamů, kolik je objemových elementů vzniklých dělením prostoru. Každý element bude mít „svůj“ seznam, který bude obsahovat objekty, které do elementu padnou buď celé nebo alespoň nějakou svojí částí (objekt zasahující do více elementů je v seznamech všech elementů, kam zasahuje).

Jak metoda dělení na podprostory funguje?

Řekněme nyní, že máme zadán paprsek. Hledání průsečíků s objekty scény začíná v tom elementu prostoru, kde leží počátek paprsku. Dále se (podle potřeby) postupuje podél paprsku do elementů dalších. V případě, že paprsek některý element prostoru opouští, je snadné rozhodnout, do kterého ze sousedních elementů přechází. Pro každý element prostoru, kterým paprsek prochází, jsou řešeny

průsečíky s objekty obsaženými v seznamu objektů elementu. Jestliže je v některém elementu průsečík nalezen, pak zpracování končí. Není-li nalezen, pak se končí tehdy, až paprsek opustí prostor scény. Metoda je účinná, jestliže jsou objekty rozděleny do jednotlivých dílčích seznamů pokud možno rovnoměrně a jestliže se tytéž objekty v příliš mnoha dílčích seznamech neopakují. (Opakovaným výpočtům průsečíku paprsku a tělesa zařazeného do více seznamů se lze ovšem vyhnout i vcelku jednoduchým programátorským opatřením. Pokuste se jej navrhnout). Časová úspora metody spočívá v tom, že se průsečíky hledají s menším počtem objektů. Metodu lze kombinovat s metodou ohraničujících objemů.



Obr. 3.13. Dělení prostoru scény na objemové elementy stejné velikosti. Vyznačený paprsek prochází třemi elementy prostoru a k nalezení jeho průsečíku v tomto případě stačilo prověřit jen tři objekty.

Kolik času lze dělením prostoru na elementy ušetřit?

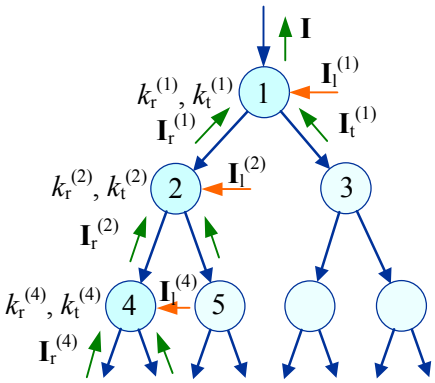
Ukažme jednoduchý pokus o odhad urychlení, kterého lze metodou dělení na elementy prostoru dosáhnout. Řekněme, že scéna obsahuje M objektů a že dělení jsme provedli tak, že podél každé ze souřadných os máme n elementů prostoru. Celkem tedy máme prostor scény rozdělen na n^3 elementů. Předpokládejme dále, že objekty jsou ve scéně rozloženy rovnoměrně a že jsou dost malé, takže se každý objekt vyskytuje vždy jen v jediném elementu. Do každého elementu proto padne M / n^3 objektů. Libovolný paprsek, který může být ve scéně veden, prochází vždy ne více než řádově n elementy. Při zpracování jednoho paprsku bude proto zapotřebí počítat průsečíky s ne více než řádově $n\ M / n^3 = M / n^2$ objekty. To je i při malých hodnotách n podstatně méně než u metody bez urychlení, která počítá průsečíky se všemi M objekty nebo alespoň s jejich ohraničujícími plochami. Prakticky dosahované hodnoty urychlení jsou ale menší, než právě provedený výpočet naznačuje. Je to proto, že předpoklady, které jsme zavedli, jsou příliš optimistické. Tělesa ve scéně často nebývají rozložena rovnoměrně a ani nebývají tak malá, aby vždy padla jen do jediného elementu prostoru. Zejména s rostoucím n budou v praktických scénách použité předpoklady platit stále méně a méně. Je také třeba uvážit, že metoda s urychlením musí řešit

přecházení paprsku z jednoho elementu prostoru do elementu sousedního. Při zpracování jediného paprsku je takových přechodů řádově až n. I tak lze ovšem v každém případě metodu považovat za účinnou. Varianta, kdy je prostor scény rozdělen na elementy různě velké, bere v úvahu možné nerovnoměrné rozložení objektů ve scéně. Jednoduše řečeno: Tam, kde je objektů málo, mohou být elementy prostoru velké. Malé elementy jsou naopak výhodné v místech, kde je mnoho malých objektů. Příkladem této techniky je dělení pomocí oktantových stromů.

Varianta dělení s různě velkými elementy

Až dosud jsme se zabývali úlohou, jak provést výpočty pro paprsek vyslaný do prostoru scény co nejrychleji. Existuje ale ještě jedna účinná a při tom velmi jednoduchá možnost urychlení, a to některé paprsky vůbec ani nevyslat. V podkapitole popisující algoritmus jsme uvedli, že proces generování „synovských“ paprsků končí, jestliže zpracovávaný paprsek opustil prostor scény nebo jestliže hloubka rekurze (tj. počet odrazů a průchodů) dosáhla jisté pevně zvolené hodnoty. To však může vést k tomu, že jsou zbytečně sledovány i paprsky, jejichž vliv na výslednou intenzitu ve zpracovávaném bodě obrazu je nepatrný. Urychlení metodou adaptivní hloubky rekurze spočívá v tom, že se odhadne, zda je paprsek pro stanovení intenzity ve vyšetřovaném obrazovém bodě dostatečně užitečný. Paprsky, které nemohou výsledek významně ovlivnit, se negenerují.

Adaptivní hloubka rekurze



Obr. 3.14. K principu adaptivní hloubky rekurze. Například: Odražený paprsek vyslaný z bodu 4 přispívá k výsledné intenzitě ve vyšetřovaném bodě obrazu hodnotou $k_r^{(1)}k_r^{(2)}k_r^{(4)}\mathbf{I}_r^{(4)}$. Jsou-li hodnoty koeficientů $k_r^{(.)}$ malé, může být příspěvek zanedbatelný.

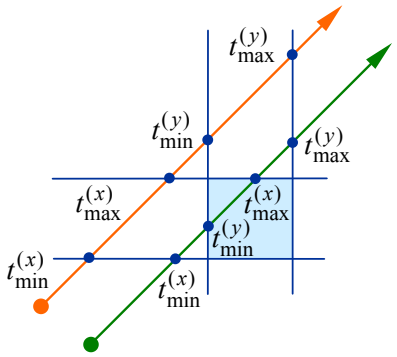
Užitečnost či neužitečnost paprsku lze stanovit tak, že se při vytváření stromu paprsků, které jsou potomky primárního paprsku vyslaného pro právě zpracovávaný bod obrazu, průběžně na cestě od kořene sledují hodnoty koeficientů k_r, k_t ze vztahu (3.1). Podrobněji to ukážeme na obr. 3.14, kde je uvedeno schéma generování potomků jednoho primárního paprsku vyslaného do scény. Zaměřme se např. na uzel označený číslem 4 (název „bod 4“ použijeme pro bod scény korespondující s uzlem 4 stromu) a promysleme, zda má ještě cenu dále vysílat z bodu 4 např. odražený paprsek. Řekněme, že bychom to skutečně udělali a že intenzita, kterou by paprsek zpět do bodu 4 přinesl, je $\mathbf{I}_r^{(4)}$. Z této intenzity se

ovšem jenom její část, a to $k_r^{(4)}\mathbf{I}_r^{(4)}$, dostane do bodu 2. Podobně se redukce koeficientem k_r opakuje ještě i v bodech 2 a 1. Pokud bychom tedy z bodu 4 do scény odražený paprsek skutečně vyslali, mohl by k intenzitě právě řešeného obrazového bodu přispět pouze hodnotou $k_r^{(1)}k_r^{(2)}k_r^{(4)}\mathbf{I}_r^{(4)}$. Jsou-li hodnoty koeficientů malé (víme, že vždy leží v intervalu $\langle 0, 1\rangle$), pak může být příspěvek paprsku zanedbatelný a paprsek není zapotřebí vůbec vysílat. Ačkoli jsme při vysvětlení použili jen paprsků odražených, platí totéž samozřejmě i pro paprsky procházející a jejich eventuální kombinace. Hodnoty součinu koeficientů lze při rekurzivním zanořování funkce Ray_Trac jednoduše sledovat, takže praktická implementace této urychlovací metody je snadná. Na závěr poznamenejme, že v tuto chvíli je již určitě jasné, proč se metoda označuje termínem „adaptivní hloubka rekurze“ (v protikladu k hloubce „fixní“, kterou jsme použili při vysvětlení algoritmu).

Příklad 3.5

Příklad 3.5. Navrhněte, jak rozhodnout, zda paprsek protíná ohraničující plochu ve tvaru kvádru se stěnami rovnoběžnými se souřadnicovými rovinami.

Řešení. Návod, jak lze úlohu řešit, v podstatě podává Liang-Barského algoritmus trojrozměrného ořezávání úseček. Jeho myšlenku zde připomeneme jen neformálně pomocí následujícího obrázku (úloha je sice trojrozměrná, ale obrázek pro jednoduchost znázorňuje jen dvojrozměrné schéma).



Algoritmus pracuje tak, že se paprskem protínají vždy dvojice rovnoběžných rovin a zjišťuje se hodnota parametru t průsečíku (rovnice (3.1)). Menší z obou hodnot je v obrázku označena indexem min, větší indexem max. Horní index označuje, o kterou dvojici rovin se jedná. Protínání se provede pro všechny tři dvojice rovin. Skutečnost, že paprsek kvádr neprotíná, je signalizována tím, že největší ze všech hodnot $t_{\min}$ je větší než nejmenší z hodnot $t_{\max}$ (zkontrolujte na obrázku). □

3.4 Vznik aliasingu a jeho potlačení

Doba studia asi 1,5 hod

Čemu se zde naučíte?

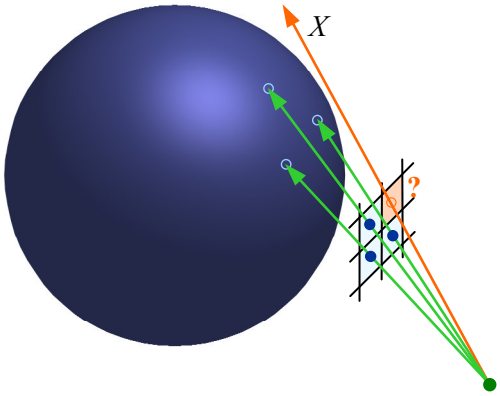
Aliasing je problém zcela obecný, který vzniká při digitálním zpracování obrazů a také v počítačové grafice. Jeho příčinou je použití nižší frekvence vzorkování (nižšího počtu vzorků), než by bylo s ohledem na obsah obrazu (jeho „členitost“) žádoucí. Jevy způsobené aliasingem vznikají vždy, když se používá diskrétní reprezentace obrazu pomocí vzorků. Největší pozornost je jim ale v počítačové grafice pochopitelně věnována v souvislosti s metodami majícími ambice produkovat realistické obrazy. Proto se o aliasingu na tomto místě zmíníme i my. Hlubší teoretické vysvětlení podstaty vzniku aliasingu se obvykle podává

v digitálním zpracování obrazu a neobejde se bez použití odpovídajících teoretických nástrojů, jako je např. Fourierova transformace. Takto náročně zde ovšem postupovat nechceme. Úloha, na níž vznik aliasingu studujeme (tedy zobrazování scény metodou sledování paprsků), nám dovoluje velmi názorné a při tom jednodušší vysvětlení. Čemu se tedy studiem této podkapitoly naučíte? Pochopíte, jak aliasing při generování obrazů vzniká. Také budete vědět, jaká opatření provést, abyste jej co nejvíce potlačili.

Co je aliasing?

Vznik aliasingu můžeme pozorovat na obr. 3.15. Řekněme, že metodou sledování paprsku zobrazujeme kouli. Zatímco tři vyobrazené paprsky se s kulovou plochou proťaly, paprsek označený *X* ji těsně minul a opustil prostor scény. Na výsledném obraze proto bude odpovídající obrazový bod znázorněn barvou pozadí. Okamžitě vidíme, že takto vytvořený obraz koule bude nepříjemně „zubatý“. Kromě toho si ještě můžeme povšimnout i jisté nekorektnosti našeho postupu. Přestože mluvíme o obrazovém bodu, nebudou jeho rozměry (např. na obrazovce) nakonec nekonečně malé, ale bude se jednat o plošku jistých rozměrů. Plošce nenulových rozměrů v obraze by nutně i v prostoru scény na povrchu tělesa měla odpovídat ploška, nikoli bod. Pokud bychom to vzali v úvahu, zjistili bychom, že diskutovaný bod obrazu by sice měl zčásti reprezentovat pozadí, ale zčásti také i kouli. To jsme ovšem jednoduchým postupem, při němž jsme pro každý obrazový bod použili pouze jediného paprsku, zjistit nemohli. Typické praktické projevy aliasingu jsou dva (obr. 3.16): 1) Vyobrazené objekty se jeví jako „zubaté“. 2) V místech, kde je obraz členitý (bohatý na vysoké frekvence) mohou vzniknout falešné a pro lidské vnímání velmi nápadné útvary.

Jak se aliasing v obrazech projevuje?



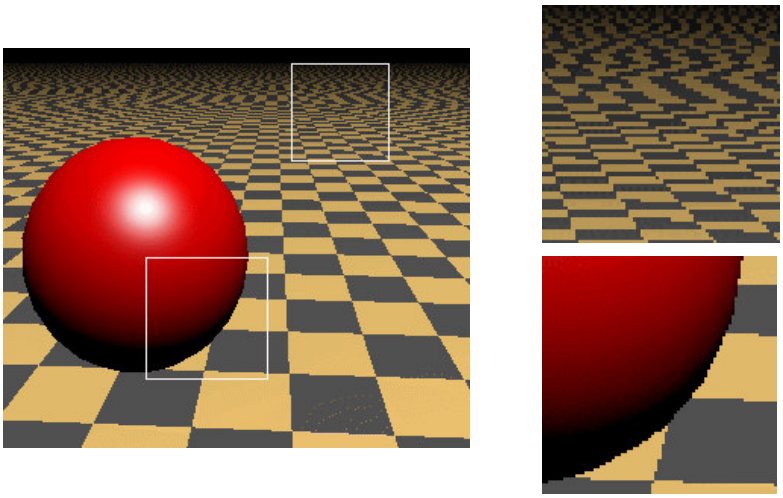
Obr. 3.15. Vznik aliasingu při zobrazování scény.

Programy realizující zobrazení metodou sledování paprsku bývají nástroji pro boj s aliasingem vybaveny. Některé z takových nástrojů zde ukážeme. Jistého zmírnění jevů způsobených aliasingem lze dosáhnout tak, že se paprsky nevysílají v mřížce zcela pravidelné. Místo toho jsou teoretické polohy bodů v obraze modifikovány malými náhodnými hodnotami, čímž dochází k „třesení“ paprsků (obr. 3.17). Podstata problému sice odstraněna není, ale efekty vzniklé aliasingem pak již nejsou pro lidské oko tolik nápadné.

Jak s aliasingem bojovat?

Supersampling

Důkladnějším, ale také časově náročnějším řešením je použití většího počtu vzorků (primárních paprsků). To bývá často označováno jako „supersampling“. Metoda je velmi jednoduchá. Obraz se generuje stejně, jak bylo popsáno v předchozích podkapitolách, ale primárních paprsků je do scény vysláno více, než je požadovaný počet bodů výsledného obrazu. Můžete si například představit, že hodnota pro jeden obrazový bod je získána průměrováním hodnot získaných pomocí čtyř až šestnácti paprsků. Supersampling lze jednoduše realizovat i pomocí takových programů, které jej přímo nenabízí. Obraz jednoduše „necháte vyrobit“ větší (v obou směrech např. 2 až 4–krát) a pak jeho rozměry zmenšíte převzorkováním v nějakém programu pro zpracování obrazu (např. Photoshop). Nevýhodou systematického zvětšení počtu použitých vzorků je dosti podstatné prodloužení výpočetního času, protože ten roste s počtem vzorků lineárně.



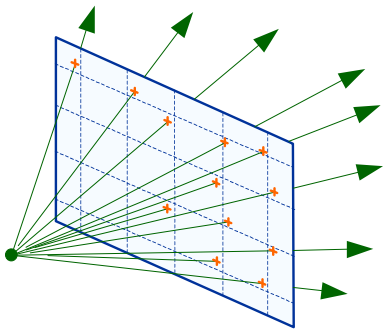
Obr. 3.16. Praktické projevy aliasingu v obrazech: „zubaté“ okraje objektů (detail dole), vznik falešných obrazců (detail nahoře).

Adaptivní supersampling

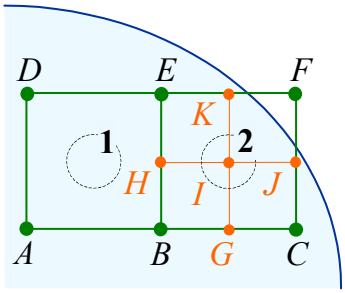
Časově úspornější je metoda, kde se počet vzorků zvětšuje podle potřeby. Na nekomplikovaných místech obrazu je vzorků méně. V komplikovaných je tomu naopak. Jedna z dnes již klasických metod pracuje tak, že jsou body obrazu (pixely) považovány za čtverečky. Pro každý obrazový jsou nejprve vyslány čtyři paprsky vedoucí přes rohy čtverečku (obr. 3.18). Jestliže je diference vypočtených intenzit menší než předen zvolený práh, pak je průměr použit jako intenzita obrazového bodu. V opačném případě je čtvereček rozdělen na čtvrtiny a výpočet se pro každou z nich opakuje. Proces dělení probíhá rekurzivně tak dlouho, dokud není dosaženo dostatečné shody intenzit v rozích vzniklých čtverečků nebo dokud nebylo dosaženo maximální předem zvolené úrovně dělení. Výsledná intenzita v obrazovém bodě se pak spočítá jako vážený průměr všech hodnot, které jsou pro obrazový bod k dispozici. Váhou je podíl na ploše pixelu, ke kterému se zjištěná dílčí hodnota váže. Na obr. 3.18 bylo další dělení potřebné v obrazovém bodě 2.

Řešení problému aliasingu bylo v minulosti věnováno poměrně hodně pozornosti. Kromě uvedených a vcelku přímočarých metod spočívajících ve

zvětšování počtu vyslaných paprsků byly publikovány i metody, v nichž paprsek není považován za polopřímku, ale za jehlan, kužel nebo hranol. Nevýhodou takových postupů je ovšem jejich obtížná teoretická i praktická zvládnutelnost.



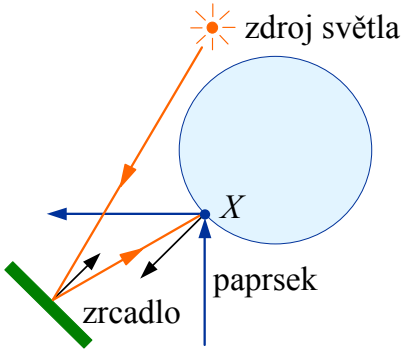
Obr. 3.17. Třesení paprsku („jittering“).



Obr. 3.18. Adaptivní „supersampling“: Ke stanovení barvy v bodě 2 obrazu byly navíc dopočteny vzorky *G*–*K*.

Úkol 3.6

Úkol 3.6. Abyste snad nenabyli dojmu, že metoda sledování paprsku „dokáže úplně všechno“, prohlédněte si následující obrázek. Ukazuje situaci, kdy má metoda (alespoň ve své základní variantě, jejíž popis jsme právě dokončili) jisté potíže. Předpokládáme, že těleso je neprůhledné. Do bodu *X* na jeho povrchu dopadají paprsky ze světelného zdroje odrazem od zrcadla. Navzdory tomuto očekávání ale metoda nebere v bodě *X* vliv světelného zdroje vůbec v úvahu.



Úkol 3.7

Úkol 3.7. Odražený paprsek zřejmě modeluje zrcadlové odrazy na površích těles. Promyslete, zda (a případně i jak) bere metoda v úvahu odrazy difúzní.

Úkol 3.8

Úkol 3.8. Zvažte, zda vás metoda sledování paprsku zaujala natolik, že byste si své zatím spíše teoretické znalosti chtěli prohloubit i její praktickou realizací. Pokud budete předpokládat jen omezený počet typů těles a pokud byste, řekněme, ani nerealizovali urychlování, pak se jedná o vcelku snadnou záležitost. V adresáři

„grafikaII\šablony\raytrac“ máte vzorový program včetně zdrojového kódu, který můžete použít jako východisko podobně, jako tomu bylo u programu „shader“ z předchozí kapitoly. Pokud se vám spíše jedná o vytváření pěkných obrázků, pak určitě raději použijte programu již hotového. Program POV-Ray je kvalitní a při tom volně dostupný. Vězte ale, že ani vytvoření pěkného obrázku není vůbec lehké. Nebuďte zklamáni prvními neúspěchy.

SHRNUTÍ

Shrnutí: V tuto chvíli by vám měl být jasný jak princip metody sledování paprsku, tak i mnoho jejích detailů. Informace, které máte, by vám měly bohatě stačit pro kvalifikovanou obsluhu programů, vytváření obrázků a případně také i k vytváření programů vlastních. Pokud byste se chtěli problémem zabývat hlouběji, lze doporučit např. následující knihy:

Doporučená literatura

- Peter Shirley, Realistic Ray Tracing, *A K Peters Ltd*, 2000, 184 stran, ISBN 1-56881-110-1,
- Andrew S. Glassner (Editor), An introduction to ray tracing, *Academic Press*, 1989, ISBN 0-12286-160-4.

Ačkoli je druhá z knih již poněkud starší, lze ji stále považovat za v celku slušný úvod do problematiky. Nejčerstvější informace je pochopitelně nutné hledat v časopiseckých článcích. Těch je k metodě sledování paprsku nepřeberné množství. Mnoho je také bibliografických přehledů. Jako příklad lze uvést přehled na http://www.acm.org/tog/resources/bib .

4 Zobrazování vyzařovací metodou

Vyzařovací metoda je dalším postupem usilujícím o vytváření realistických obrázků. Její vznik se datuje do doby asi před dvaceti lety. Mohli byste namítnout, že již metoda rekurzivního sledování paprsku se vám zdála dost dobrá, a že tedy proto vlastně není další metody zapotřebí. Není tomu tak. Již v úkolech 3.6 a 3.7 jsme poukázali na některé nedostatky metody sledování paprsku. Stručně lze říci, že sledování paprsku příliš preferuje zrcadlový odraz paprsků na površích objektů scény na úkor odrazu difúzního. Nejpěkněji proto zobrazuje scény obsahující objekty s více či méně lesklými povrchy, které jsou dobře osvětleny, a to zejména bodovými zdroji světla. Vlastnosti vyzařovací metody jsou do značné míry protikladné. Zaměřuje se zejména na difúzní odrazy světla (alespoň ve své základní variantě, kterou zde popíšeme). Je proto vhodná pro zobrazování scén obsahujících objekty s matnými povrchy a osvětlené rozptýleným světlem. Velmi se například hodí pro zobrazování architektury, zejména interiérů. Protože poměrně přesně počítá osvětlení v každém místě scény, může být použita i při návrhu rozmístění a intenzity světelných zdrojů. Podstatný rozdíl v myšlenkovém základu obou metod lze ilustrovat i dosti odlišným vzhledem obrázků, které metody produkují (obr. 4.1, 4.2). Po přečtení této kapitoly budete podrobně vědět, jak vyzařovací metoda pracuje. To vám přinejmenším umožní kvalifikovaně používat existující programy, které vyzařovací metodu k zobrazování scény využívají. Budete také schopni sledovat vývoj v dané oblasti a metodu případně i sami implementovat.

Proč by vás mohla vyzařovací metoda zajímat?

Čemu se v této kapitole naučíte?

Typické je použití metody při zobrazování interiérů.

Obr. 4.1. Obraz generovaný vyzařovací metodou (se svolením Advanced Interfaces Group, Department of Computer Science, University of Manchester).

Obr. 4.2. Obraz vytvořený vyzařovací metodou (model vytvořen v AutoCADu R14, zobrazen v AccuRenderu 3, se svolením Scott Davidson, Robert McNeel & Associates).

Úkol 4.1

Úkol 4.1. Prohlédněte si obrázky, které naleznete na přiloženém CD v adresáři „grafikaII\radiosita\obrazky". Uvidíte, co můžete od zobrazování vyzařovací metodou očekávat. Můžete také navštívit internetové prezentace programových produktů, které zobrazování vyzařovací metodou provádí. Např. AccuRender (http://www.accurender.com), Lightscape (http://www.autodesk.com) Helios (http://www.helios32.com). □

4.1 Princip vyzařovací metody

Doba studia asi 1,5 hod

Čemu se zde naučíte?

V této podkapitole se seznámíte se základními myšlenkami vyzařovací metody. Ty pak budou v následujících podkapitolách doplněny dalšími detaily.

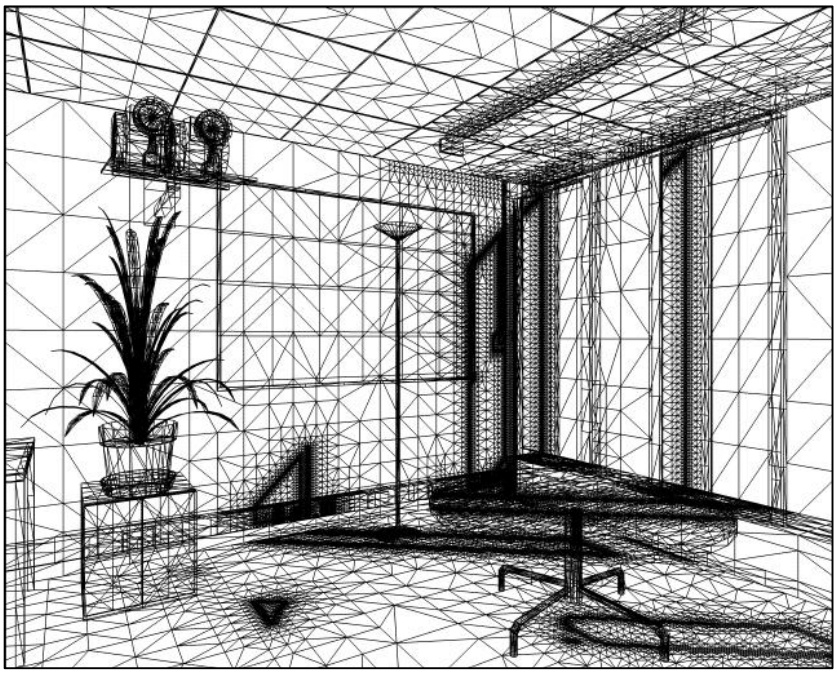
Scéna se pokryje ploškami.

Vypočte se záření plošek.

Hodnota záření se použije k zobrazení plošky.

Ústřední krok vyzařovací metody spočívá v tom, že se vypočítá, jak jsou osvětlena jednotlivá místa scény. K tomu se povrchy těles ve scéně pokryjí sítí plošek, např. tak, jak je znázorněno na obr. 4.3. Pro každou plošku sítě metoda vypočítá světelný výkon, který ploška vyzařuje zpět do prostoru scény. Obrázek scény se pak vytváří tak, že se vykreslují jednotlivé plošky sítě. Hodnoty vyzařování plošek jsou při tom použity pro výpočet intenzit bodů v obraze. Je zajímavé zdůraznit, že výsledek výpočtu osvětlení scény (tedy vyzařování plošek) závisí jen na geometrii scény, na materiálech povrchů a na intenzitě a rozmístění světelných zdrojů. Je naopak nezávislý na zobrazení. Jestliže již byl výpočet osvětlení jednou proveden, lze výsledku použít k vytváření různých obrazů scény. To je důležité, protože výpočet osvětlení, který je časově náročný, často stačí provést jen jednou. Samotné zobrazování pak už může být velmi rychlé.

Příklad pokrytí scény ploškami



Obr. 4.3. Síť plošek pokrývajících povrchy scény z obr. 4.1. Různá podrobnost dělení je v různých místech nezbytná. V místech s komplikovanějším průběhem osvětlení musí být síť hustá. (Se svolením Advanced Interfaces Group, Department of Computer Science, University of Manchester.)

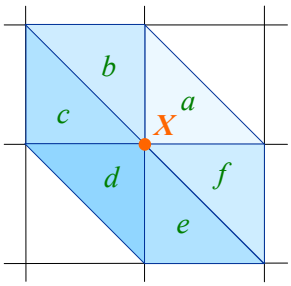
Výpočet probíhá po složkách r, g, b.

Prakticky jsou při výpočtu osvětlení pro každou plošku spočítány tři hodnoty udávající její vyzařování na vlnových délkách odpovídajících základním barevným složkám *r*, *g*, *b*. Je tedy zřejmé, že proto již nelze dále rozlišovat, zda některá část plošky vyzařuje silněji či slaběji, ale předpokládá se vyzařování konstantní po celém povrchu plošky.

Jak se záření plošek použije k jejich zobrazení?

Představme si na okamžik, že jsme již výpočet osvětlení provedli a že vyzařování každé plošky už známe. Je užitečné hned na tomto místě říci, jak se hodnoty vyzařování plošek použijí k zobrazení scény, i když se z hlediska metody jedná až o její závěrečný krok. Na obr 4.4 je znázorněn fragment sítě. Vypočítaná intenzita vyzařování plošek je zde znázorněna jejich různým odstínem. Je ale ihned jasné, že podobně jednoduchého postupu nebude možné při vykreslování skutečného obrázku použít. Protože vyzařování jednotlivých (byť vedle sebe ležících) plošek může vyjít znatelně různé, staly by se plošky ve výsledném obraze viditelnými, což je jistě nežádoucí. Dělení na plošky je zapotřebí „zamaskovat“. Tradiční postup je tento: Nejprve se hodnoty intenzit vyzařování „přenesou“ do uzlů sítě. To se jednoduše provede tak, že se jako intenzita v uzlu vezme průměr intenzit vyzařování ploch, které k uzlu přiléhají. Uvažujme uzel *X* z obr. 4.4, o němž předpokládáme, že neleží ani na hraně ani ve vrcholu žádného z objektů scény. Hodnotu intenzity vyzařování v tomto uzlu lze pak jednoduše spočítat jako průměr intenzit ze šesti k němu přiléhajících ploch *a* až *f*. Pro uzly ležící na hranách nebo ve vrcholech těles je situace poněkud jiná, protože skutečné hrany a vrcholy objektů interpolací naopak zamaskovány být nemají. Pro každý takový uzel je tolik různých hodnot vyzařování, kolik různých stěn tělesa k němu přiléhá. To je ale snadno pochopitelné. Tutéž situaci již ostatně znáte také z Gouraudova stínování. Ukázku výpočtu vyzařování pro uzly na hranách a ve vrcholech objektů scény ponecháme do úkolu 4.3. Zde budeme předpokládat, že hodnoty intenzit vyzařování ve všech uzlech sítě už známe, a že proto známe také intenzity ve všech uzlech na obvodu každé plošky. Při zobrazování plošky (tedy při rozsvěcení jejích jednotlivých bodů v obraze) lze pak postupovat tak, že se mezi hodnotami v uzlech na jejím obvodu interpoluje.

Přenesení intenzity záření do uzlů sítě.



Obr. 4.4. Výpočet intenzity vyzařování v uzlu *X* sítě. Intenzita se spočítá jednoduše jako průměr intenzit od všech ploch, které k uzlu *X* přiléhají. Zde konkrétně ploch *a* až *f*.

Hodnoty v mezilehlých bodech se získají interpolací.

Nejjednodušším možným způsobem provedení interpolace a zobrazení scény je použití Gouraudova stínování, s nímž jsme se seznámili v kapitole 2. Pro pořádek musíme ovšem upozornit na to, že to ale není postup teoreticky úplně správný. Osvětlení bylo vypočítáno v prostoru scény a tam by se také měla

provádět interpolace. Gouraudovo stínování interpoluje v prostoru obrazu. Problém je v tom, že středová projekce, která bude pro zobrazení scény nejspíš použita, nezachovává dělicí poměr, a proto se výsledky interpolace v prostoru obrazu budou poněkud lišit od výsledků, které bychom dostali v prostoru scény. K dalším odchylkám může navíc dojít, pokud by plochy nebyly rovinné. I přes uvedené výhrady se ale na tomto místě Gouraudova stínování používá vcelku dost často, zejména tehdy, má-li být zobrazení scény rychlé. I my se s tímto postupem zatím spokojíme. Také v ostatních krocích zobrazení (projekce, ořezání, řešení viditelnosti) lze postupovat beze zbytku tak, jak bylo popsáno v kapitole 2.

Jaká jsou pravidla pro generování sítě plošek?

Při prohlížení obrázku 4.3 jste se možná podivili tomu, jak je pokrytí povrchů těles ploškami provedeno. Asi jste si všimli, že někde je dělení řídké (plošky jsou velké), jinde je tomu naopak. V tuto chvíli by již mělo být jasné, proč tomu tak je. Protože se předpokládá, že je vyzařování po celém povrchu jedné plošky konstantní, musí být dělení husté v místech, kde dochází v osvětlení k nějakým složitějším jevům. Je zřejmé, že při velkých ploškách by např. náhlý přechod mezi světlem a stínem nemohl být zobrazen správně, protože by jej interpolace „zamaskovala“. V takových místech proto musí být dělení velmi husté. V místech, kde naopak k žádným komplikovaným světelným jevům nedochází, může být dělení řidší. Používání proměnné hustoty dělení je žádoucí. Síť všude řídká by vedla k velkým chybám ve výpočtu osvětlení. Síť naopak všude hustá by ale také nebyla vhodná. Vedla by na tak rozsáhlé výpočty, že by je nebylo možné prakticky realizovat. Určitým problémem je, že v době, kdy se síť vytváří, máme o světelných poměrech ve scéně zpravidla jen přibližnou předběžnou představu, a nevíme proto přesně, jak hustého či řídkého dělení bude na jednotlivých místech zapotřebí. Je proto obvyklé, že se nalezení vhodné sítě provádí v několika iteračních krocích. Nejprve se použije hustoty odhadnuté intuitivně. Pak se provede výpočet osvětlení a podle získaných výsledků se síť upraví (zpravidla zhustí). Tento postup se opakuje tak dlouho, dokud není dosaženo vyhovujícího dělení. Proces vytváření sítě a jejího zjemňování bývá zpravidla automatizován.

Hustota dělení musí odpovídat složitosti světelných poměrů!

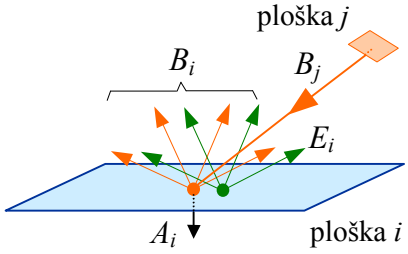
Jak se spočítají hodnoty vyzařování jednotlivých plošek?

Nyní již konečně můžeme přistoupit k vysvětlení, jak se hodnoty vyzařování jednotlivých plošek spočítají. Uvažujme *i*-tou plošku ve scéně. Na tuto plošku „září“ jiné plošky a k plošce uvažované tak vysílají jistý vyzářený výkon (na obr. 4.5 je znázorněna jediná taková ploška, která je označena jako ploška *j*). Výkon B_j vyzářený ploškou *j* a dopadající na plošku *i* je zčásti ploškou *i* absorbován, ale zčásti také difúzně odražen zpět do prostoru scény. Míru odrazu popisuje koeficient ρ_i, který tak popisuje optické vlastnosti materiálu plošky. Kromě toho, že ploška *i* vysílá zpět do prostoru výkon odražený, může být sama také aktivním zdrojem energie. Tato možnost je na obrázku 4.5 vyznačena hodnotou E_i, což je hodnota výkonu vlastního vyzařování plošky. Podobně jako u odrazu, předpokládá se i zde, že se vyzařování děje difúzně, tj. všemi směry.

Princip spočívá v dosažení energetické rovnováhy na každé z plošek.

Základní myšlenkou vyzařovací metody je předpoklad, že se na všech ploškách scény ustaví energetická rovnováha spočívající v tom, že součet výkonu ploškou vyzařovaného a výkonu absorbovaného je roven součtu výkonu na plošku dopadajícího od jiných ploch a výkonu, který ploška sama aktivně vyzařuje. Uvedenou myšlenku lze matematicky zapsat pomocí rovnice (*n* značí počet plošek)

$$B_i = E_i + \rho_i \sum_{j=1}^{n} B_j F_{j \to i} \frac{A_j}{A_i} \ . \qquad (4.1)$$



Vyzařování plošky i do prostoru scény

Obr. 4.5. Záření plošky *j* na plošku *i*: Výkon B_j přicházející od plošky *j* ploška *i* částečně absorbuje (A_i) a částečně odráží zpět do prostoru. Ploška *i* může být také sama zdrojem energie (E_i). Výsledkem všech uvedených jevů je, že ploška *i* difúzně vyzařuje do prostoru výkon B_i.

Význam členů B_i, E_i, ρ_i jsme vysvětlili již dříve. Zde jen upřesníme, že výkony B_i a E_i jsou vztaženy na jednotku velikosti plochy (jedná se tedy o měrný výkon, jinak také o energii / čas / velikost plochy). Hodnota $F_{j\to i}$ je tzv. konfigurační koeficient zohledňující velikosti a vzájemné geometrické uspořádání *i*-té a *j*-té plošky. Konkrétně konfigurační koeficient $F_{j\to i}$ říká, jaká část z celkového výkonu, který do prostoru vyzářila ploška *j*, dopadne na plošku *i*. Odtud vidíme, že se hodnoty konfiguračních koeficientů nutně musí pohybovat v intervalu $\langle 0,1\rangle$ a že záleží na pořadí indexů ($j\to i$ není totéž co $i\to j$). Hodnota $F_{j\to i} = 0$ platí pro případ, že plocha *j* na plochu *i* nezáří, např. proto, že je mezi nimi neprůhledná překážka. Bližší podrobnosti o konfiguračních koeficientech a jejich výpočtu uvedeme v následující podkapitole. Hodnoty A_i, A_j jsou velikosti plošek *i*, *j* (tj. jejich plošný obsah).

Nyní by již konstrukce vztahu (4.1) měla být jasná. Hodnota B_jA_j udává celkový výkon, který do prostoru vyzařuje celá ploška *j*. Z této hodnoty ovšem na plošku *i* dopadne jen výkon $F_{j\to i}B_jA_j$. Z této hodnoty je opět jen část, a to $\rho_iF_{j\to i}B_jA_j$, *i*-tou ploškou odražena zpět do prostoru scény. Konečně dělení plochou A_i je ve vztahu (4.1) proto, že se bilancování energetické rovnováhy provádí pomocí měrného výkonu na jednotku plochy a také hodnoty B_i, E_i v rovnici (4.1) mají tento rozměr. Poznamenejme ještě, že hodnoty výkonů i hodnoty koeficientů odrazu ρ_i jsou závislé na vlnové délce. Prakticky se postupuje tak, že se výpočet, jak jste již ostatně zvyklí, provádí pro tři základní barevné složky. Pro stručnost už tuto skutečnost nebudeme dále nijak zdůrazňovat. Vztahy, které budou následovat, můžete brát jako platné pro jednu (kteroukoli) barevnou složku. Pro pořádek dodejme, že konfigurační koeficienty na vlnové délce nezávisí.

Pro úpravu rovnice (4.1) se používá tzv. principu reciprocity

$$A_iF_{i\to j} = A_jF_{j\to i}\ . \tag{4.2}$$

Je dost možné, že důvody, proč by měla rovnost (4.2) platit, v tuto chvíli ještě nevidíte. Její platnost dokážeme až v následující podkapitole. Předpokládejte proto zatím, že skutečně platí. Dosazením vztahu (4.2) do rovnice (4.1) dostaneme

$$B_i = E_i + \rho_i\sum_{j=1}^{n} B_jF_{i\to j}\ . \tag{4.3}$$

Přeskládáním členů v rovnici (4.3) pak dále máme

Rovnice výkonové rovnováhy pro i-tou plošku

$$B_i - \rho_i \sum_{j=1}^{n} B_j F_{i \to j} = E_i \,. \tag{4.4}$$

Připomeňme, že rovnice (4.4) vyjadřuje energetickou rovnováhu pro *i*-tou plošku sítě. Rovnováha však platí pro všechny plošky. Aplikací principu energetické rovnováhy pro všech *n* plošek dostaneme soustavu lineárních rovnic, kterou lze zapsat v maticovém tvaru takto

Rovnice výkonové rovnováhy pro všechny plošky

$$\begin{bmatrix} 1-\rho_1 F_{1\to1} & -\rho_1 F_{1\to2} & \dots & -\rho_1 F_{1\to n} \\ -\rho_2 F_{2\to1} & 1-\rho_2 F_{2\to2} & \dots & -\rho_2 F_{2\to n} \\ \dots & \dots & \dots & \dots \\ -\rho_n F_{n\to1} & -\rho_n F_{n\to2} & \dots & 1-\rho_n F_{n\to n} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ \dots \\ B_n \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \dots \\ E_n \end{bmatrix} . \tag{4.5}$$

V uvedené soustavě jsou neznámými hodnoty B_1, B_2, … , B_n vyzařování jednotlivých plošek. Matice soustavy obsahuje jen hodnoty konfiguračních koeficientů a koeficientů odrazu. Hodnoty konfiguračních koeficientů se spočítají postupem popsaným v následující podkapitole. Hodnoty koeficientů odrazu musí být zadány. Zadány musí být také hodnoty E_1, E_2, … , E_n tvořící vektor pravých stran. Soustavu bude většinou zapotřebí řešit třikrát pro jednotlivé barevné složky (*r*, *g* ,*b*) vyzařování ploch. Pro všechna tři řešení ale v matici soustavy zůstávají nezměněny hodnoty konfiguračních koeficientů. Hodnoty koeficientů odrazu se mohou měnit. Zjištěné hodnoty vyzařování ploch se, jak již víme, použijí k zobrazení scény.

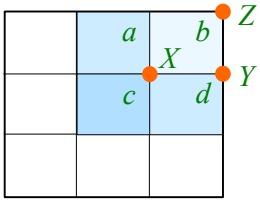
Řešení soustavy rovnic (4.5)

Vlastní řešení soustavy (4.5) je poněkud nepříjemné. Je to proto, že ploch ve scéně bývá velké množství (řádově až desetitisíce či statisíce) a matice soustavy obecně může být plná. Nejčastěji se v tomto kontextu uplatňují metody iterační, jakou je např. Gauss-Seidelova metoda relaxační (úkol 4.4). Výhodou iteračních metod je to, že mohou velmi brzy poskytnout alespoň přibližné řešení. To je obvykle užitečné. Zatímco uživatel tráví čas prohlížením prvních předběžných obrázků, může se řešení soustavy postupnými iteracemi dále zpřesňovat. Při tom také může být podle potřeby průběžně prováděno zjemňování dělení sítě (podkapitola 4.3). Vlastnosti matice soustavy ze vztahu (4.5) jsou pro řešení relaxační technikou výhodné (úkol 4.4).

Úkol 4.2

Úkol 4.2. Promyslete, jak by bylo možné spočítat hodnoty vyzařování v uzlech ležících na hraně nebo ve vrcholu objektu scény.

Řešení. Jak jsme již uvedli v textu podkapitoly, je v uzlech ležících na hranách nebo ve vrcholech objektů scény zapotřebí tolik hodnot vyzařování, kolik stěn tělesa k uzlu přiléhá. Zde budeme uvažovat jen jednu přiléhající stěnu, která dá vzniknout právě jedné z hledaných hodnot (pro zbývající stěny je postup stejný). Řekněme např., že následující obrázek znázorňuje stěnu kvádru, která je pokryta sítí plošek. Pro jednoduchost jsme tentokrát zvolili plošky obdélníkového tvaru.



$$B_X = (B_a+B_b+B_c+B_d)/4$$
$$B_Y = B_b+B_d-B_X$$
$$B_Z = 2B_d-B_X$$

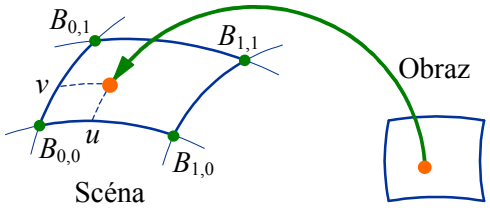
Výpočet vyzařování v uzlu X je zřejmý, protože odpovídá postupu, který jsme již v textu dříve vysvětlili. Pro vyzařování v uzlech Y, Z byste nyní možná očekávali hodnoty $B_Y = (B_b + B_d)/2$, $B_Z = B_b$. Častěji se ale používá hodnot uvedených v obrázku. Jedná se o extrapolaci, jak se o tom rozepsáním uvedených vztahů můžete přesvědčit. Vychází

$$B_Y = B_b + B_d - B_X = \frac{1}{2}(B_b + B_d) + \frac{1}{4}\left[(B_b - B_a) + (B_d - B_c)\right],$$
$$B_Z = 2B_d - B_X = B_b + \frac{1}{4}\left[(B_b - B_a) + (B_b - B_c) + (B_b - B_d)\right].$$ □

Úkol 4.3. Promyslete, jak by bylo možné realizovat interpolaci v prostoru scény. (Místo interpolace v prostoru obrazu, kterou provádí Gouraudovo stínování. Vzpomeňte si, že jsme byli v předchozím textu k interpolaci v prostoru obrazu poněkud kritičtí.)

Úkol 4.3

Řešení. Abychom postup ukázali co nejjednodušeji a při tom abychom současně byli náležitě obecní, budeme předpokládat, že plošky, kterými jsou povrchy scény pokryty, jsou popsány parametricky. Nechť u, v jsou parametry, $0 \le u, v \le 1$. Řekněme, že pro každý vykreslovaný bod v obraze bude možné stanovit hodnotu parametrů u, v pro odpovídající bod plošky ve scéně. Nechť $B_{0,0}$, $B_{1,0}$, $B_{0,1}$, $B_{1,1}$ jsou vyzařování ve vrcholech plošky.



Bilineární interpolací v prostoru scény lze pak hledanou hodnotu vyzařování stanovit podle vztahu.

$$B(u,v) = (1-u)(1-v)B_{0,0} + u(1-v)B_{1,0} + (1-u)vB_{0,1} + uvB_{1,1}.$$ □

Úkol 4.4. Připomeňte si Gauss-Seidelovu relaxační metodu řešení soustavy lineárních rovnic. Pokud jste se s ní snad ještě nesetkali, nevadí. Vše by mělo být jasné z následujícího řešení.

Úkol 4.4

Řešení. Uvažujme soustavu $\mathbf{Ax} = \mathbf{b}$ n rovnic. Horní index (k) nechť vyznačuje, že hodnota byla nalezena po k-té iteraci. Nejprve je nutné odhadnout počáteční hodnotu $\mathbf{x}^{(0)}$ vektoru neznámých. Jednotlivé iterace jsou pak prováděny podle předpisu

$$x_i^{(k)} = \frac{1}{a_{ii}}\left[b_i - \sum_{j<i} a_{ij} x_j^{(k)} - \sum_{j>i} a_{ij} x_j^{(k-1)} \right], \quad (1 \le i \le n).$$

Předpis říká, že se jednotlivé rovnice řeší každá zvlášť tak, že se z *i*-té rovnice vypočítá prvek x_i. Při tom se při výpočtu využijí „nejčerstvější“ (tedy naposledy zjištěné) hodnoty všech ostatních prvků x_j. Iterační proces konverguje, jestliže je matice **A** striktně diagonálně dominantní, což znamená, že pro každé *i* platí $|a_{ii}| > \sum_{j \neq i} |a_{ij}|$. Jestliže je $\rho_i < 1$, pak matice soustavy (4.5) skutečně má tuto vlastnost (úkol 4.6). Jako počáteční odhad lze vzít hodnotu $\mathbf{x}^{(0)} = (E_1, E_2, \ldots, E_n)^{\mathrm{T}}$. □

Úkol 4.5

Úkol 4.5. Abyste si ověřili, zda jste činnosti vyzařovací metody správně porozuměli, nakreslete schéma posloupnosti jejích jednotlivých kroků od zadání popisu scény až po generování výsledného obrazu. Předpokládejte, že se interpolace provádí v prostoru obrazu. (Řešení je uvedeno na konci kapitoly.) □

S H R N U T Í

Shrnutí: V tuto chvíli by už vám měl být princip základní varianty vyzařovací metody jasný. Chybí vám už jen informace, jak vypočítat hodnotu konfiguračních koeficientů. Tomuto problému se budeme věnovat v následující podkapitole.

4.2 Výpočet konfiguračních koeficientů

Doba studia asi 1,5 hod

Čemu se zde naučíte?

Jak jste se dověděli v předchozí podkapitole, hrají při zobrazování vyzařovací metodou konfigurační koeficienty zásadní úlohu. V této podkapitole ukážeme, jak je lze stanovit. Ukáže se při tom, že to není tak úplně jednoduché. Výpočet konfiguračních koeficientů proto tvoří značnou část času, který je pro výpočet osvětlení potřebný.

Kosinový zářič

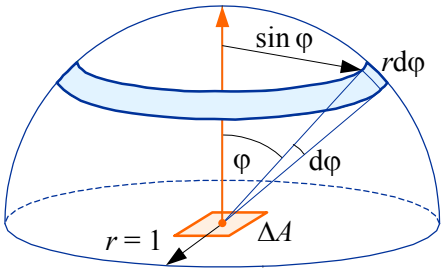
Než se do výpočtu konfiguračních koeficientů pustíme, musíme upřesnit, že si každou plošku představujeme jako tzv. kosinový zářič (někdy se také používá termínu „Lambertův“ zářič). U kosinového zářiče se intenzita vyzařování mění v závislosti na úhlu, který svírá vyzářený paprsek s normálou vyzařující plošky. Označme tento úhel φ a zaveďme hodnotu $\tilde{B}(\varphi)$, která popisuje intenzitu záření do prostorového úhlu jednoho steradiánu v závislosti na φ (jeden steradián je takový úhel, který na kulové ploše o poloměru *r* vytíná plochu r^2). Pro kosinový zářič platí vztah $\tilde{B}(\varphi) = B\cos\varphi$, kde *B* je intenzita záření ve směru normály. Předpokládejme, že uvažovaný kosinový zářič má tvar malé plošky velikosti ΔA. Obklopme jej polokulovou plochou, jejíž poloměr zvolíme pro jednoduchost jednotkový (obr. 4.6). Celková velikost záření, které polokulovou plochou prochází a kterou tedy zářič „vysílá“ do prostoru je (obr. 4.6)

$$\int_0^{\pi/2} \tilde{B}(\varphi)\Delta A\, 2\pi\sin(\varphi)\,\mathrm{d}\varphi = 2\pi B\Delta A \int_0^{\pi/2} \cos(\varphi)\sin(\varphi)\,\mathrm{d}\varphi = \pi B\Delta A \ . \qquad (4.6)$$

Výpočet koeficientů pro „malé“ zářící plochy a malé plochy dopadu.

Při výpočtu konfiguračních koeficientů vyjdeme z definice zavedené v předchozí podkapitole, a to, že konfigurační koeficient $F_{i\to j}$ říká, jaká část z celkového výkonu, který do prostoru vyzářila ploška *i*, dopadne na plošku *j*. Začneme případem, kdy jsou obě plošky vzhledem ke vzdálenosti mezi nimi zanedbatelně malé. Nechť $\mathrm{d}A_i$, $\mathrm{d}A_j$ značí jejich velikost a *r* vzdálenost (obr. 4.7).

Vyzařování kosinového zářiče do prostoru

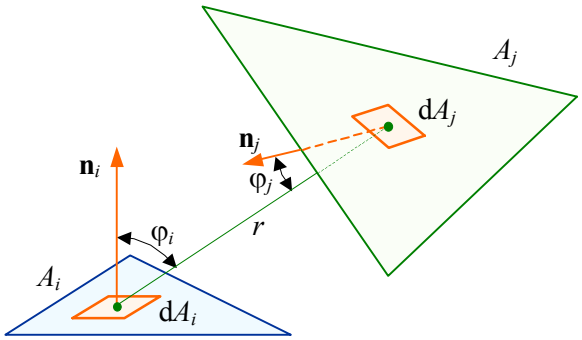


Obr. 4.6. K výpočtu vyzařování kosinového zářiče.

Mysleme si, že intenzita vyzařování zářící plošky je B_i. Z předchozího již víme, že ploška pak do prostoru vyzáří výkon $\pi B_i \mathrm{d}A_i$. Z tohoto výkonu na plošku j dopadne jen jeho část velikosti $(B_i \mathrm{d}A_i \cos\varphi_i)(\mathrm{d}A_j \cos\varphi_j / r^2)$. Člen v první závorce udává velikost záření, které ploška i vysílá do jednoho steradiánu ve směru k plošce j. Člen v druhé závorce pak počítá prostorový úhel, v němž se ploška j při pohledu z místa plošky i jeví. Berouce v úvahu dříve vyslovenou definici konfiguračních koeficientů jako poměrů výkonů, docházíme k následujícímu vztahu

$$F_{\mathrm{d}i\to\mathrm{d}j} = \frac{\cos\varphi_i \cos\varphi_j}{\pi r^2} H_{ij} \mathrm{d}A_j \ . \qquad (4.7)$$

Indexu di→dj jsme použili, abychom zdůraznili, že se jedná o plošky malé. Navíc k dosud provedeným úvahám jsme zavedli hodnotu H_{ij} beroucí v úvahu, zda plošky i, j „na sebe vidí“ (H_{ij} = 1) nebo nikoli (tj. je mezi nimi neprůhledná překážka bránící průchodu záření, H_{ij} = 0).



Obr. 4.7. K výpočtu konfiguračních koeficientů.

Výpočet konfiguračního koeficientu pro „velké“ plochy dopadu

S využitím výsledku ze vztahu (4.7) můžeme řešit i další případy. Jestliže lze vyzařující plošku stále považovat za malou, ale plochu, na níž záření dopadá, nikoli, získáme hodnotu konfiguračního koeficientu integrací podle vztahu

$$F_{\mathrm{d}i\to j} = \int_{A_j} \frac{\cos\varphi_i \cos\varphi_j}{\pi r^2} H_{ij}\,\mathrm{d}A_j \ . \tag{4.8}$$

Výpočet pro „velké" zářící plochy

Také v případě, že ani zářící plošku nelze považovat za malou, získáme hodnotu konfiguračního koeficientu integrací. Význam integrování je zde ale jiný než ve vztahu (4.8). Povšimněte si totiž, že hodnota $F_{\mathrm{d}i\to j}$ na velikosti zářící plošky již nezávisí, což naznačuje, že by další integrace nemělo být zapotřebí. Smysl dalšího integrování zde spočívá v následujícím: Představme si zářící plochu pokrytou malými ploškami. Pro každou z těchto malých plošek lze konfigurační koeficient „malá plocha → plocha *j*" vypočítat podle vztahu (4.8). Pro každou z nich může ale obecně vyjít jiná hodnota. Z předchozí podkapitoly ovšem víme, že metoda předpokládá pro každou dvojici ploch, byť velkých, koeficient právě jediný. Možným řešením, jak toho dosáhnout, je vzít hodnotu koeficientu $F_{i\to j}$ jako střední hodnotu koeficientů $F_{\mathrm{d}i\to j}$ vypočtených pro uvažované malé dílčí plošky. Výpočtu střední hodnoty konfiguračního koeficientu nad plochou A_i pak odpovídá vztah

$$F_{i\to j} = \frac{1}{A_i} \int_{A_i}\int_{A_j} \frac{\cos\varphi_i \cos\varphi_j}{\pi r^2} H_{ij}\,\mathrm{d}A_j\,\mathrm{d}A_i \ . \tag{4.9}$$

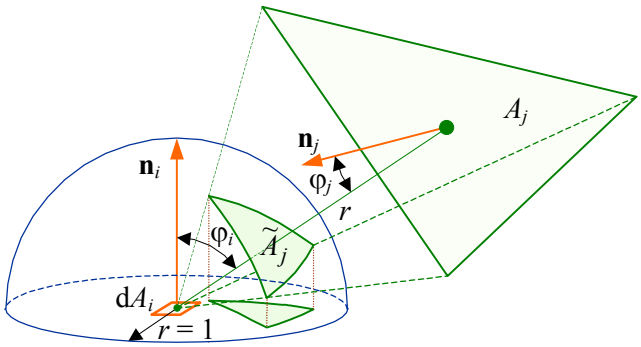
Princip reciprocity

Vztah (4.9) již podává návod, byť zatím poněkud teoretický, jak konfigurační koeficienty vypočítat. Navíc již s využitím vztahu (4.9) také dokážete ověřit princip reciprocity (4.2), který jsme použili v předchozí podkapitole (je asi nyní jasné, proč nebylo možné jeho platnost už v předchozí podkapitole také prokázat). Než se pustíme do problematiky praktického výpočtu konfiguračních koeficientů, uveďme ještě dvě pozorování, která jednak doplňují pohled teoretický, ale která současně také mohou být využita i při praktických výpočtech. 1) Konfigurační koeficienty pro všechny dvojice ploch ležící na povrchu jediného konvexního tělesa vždy nabývají hodnoty nula (protože plochy na povrchu konvexního tělesa na sebe navzájem „nevidí"). 2) Ve scéně ohraničené ze všech stran plochami musí být $\sum_j F_{i\to j} = 1$ (jednoduše proto, že všechen výkon vyzářený plochou *i* musí dopadnout na jiné plochy scény).

Praktický postup výpočtu konfiguračních koeficientů

Nepožadujete-li hlubší znalosti, můžete zbytek podkapitoly přeskočit.

Vztahy (4.8) a (4.9) sice již jednoznačně ukazují, jak konfigurační koeficienty počítat, ale jejich praktická realizace v programech by byla nesnadná. V praxi se proto používá různých zjednodušení. I my zde některá ukážeme. Zaměříme se nejprve na případ, kdy lze zářící plošku *i* považovat za velmi malou a rovinnou, tedy na případ, pro který platí vztah (4.8). Plošku obklopíme polokulovou plochou jednotkového poloměru tak, že rovina plošky tvoří její rovníkovou rovinu. Na ploše *j* uvažujme malou elementární plošku $\mathrm{d}A_j$. Z předchozích teoretických úvah již víme, že velikost energie vyzářená z plošky *i* a dopadnuvší na elementární plošku $\mathrm{d}A_j$ závisí na prostorovém úhlu, jímž se ploška $\mathrm{d}A_j$ jeví z místa plošky *i*. Je očividné, že se prostorový úhel pro plošku $\mathrm{d}A_j$ shoduje s prostorovým úhlem pro plošku $\mathrm{d}\widetilde{A}_j$, která vznikne průmětem plošky $\mathrm{d}A_j$ na polokouli (obr. 4.8). Protože je ploška *i* kosinovým zářičem, je záření dopadající na plošku $\mathrm{d}\widetilde{A}_j$ (a také na plošku $\mathrm{d}A_j$) závislé na kosinu úhlu φ sevřeného normálou plošky *i* a směrem k plošce $\mathrm{d}\widetilde{A}_j$. Hodnotu součinu $\cos\varphi\,\mathrm{d}\widetilde{A}_j$ lze interpretovat jako velikost průmětu elementu $\mathrm{d}\widetilde{A}_j$ do rovníkové roviny kulové plochy.



Obr. 4.8. Nusseltův pohled na výpočet konfiguračních koeficientů.

Nusseltův přístup

Vidíme, že konfigurační koeficient $F_{di \to dj}$ pro dvojici plošek i, dA_j by bylo možné spočítat tak, že vypočteme průmět plošky dA_j na polokouli a dále pak do rovníkové roviny polokoule. Poměr velikosti průmětu plošky v rovníkové rovině ku velikosti plochy celého „rovníkového kruhu“ dává hledaný konfigurační koeficient (celá polokoule, a proto i celý kruh totiž nutně odpovídají celkovému záření, které ploška i do prostoru vysílá). Hodnotu koeficientu pro celou plochu j snadno získáme „integrací“. Dojdeme tak k závěru, že konfigurační koeficient $F_{di \to j}$ lze vypočítat tak, že ty části plochy j, které jsou z plochy i viditelné, promítneme na jednotkovou polokouli a odtud dále do její rovníkové roviny. Poměr velikosti průmětu v rovníkové rovině ku velikosti celého rovníkového kruhu udává hledanou hodnotu konfiguračního koeficientu. Právě uvedenou geometrickou interpretaci vztahů pro výpočet konfiguračních koeficientů, která současně může být i návodem pro jejich výpočet, poprvé formuloval Nusselt. Bývá proto v literatuře spojována s jeho jménem, a to jako Nusseltova analogie. Nusseltova analogie sice ukazuje geometrický význam vztahů, které jsme dříve zavedli, ale ani ona ještě nevede k výpočetnímu postupu, který by byl zcela praktický.

Postup využívající poloviny krychle

Konečně ještě ukážeme postup, který je již z hlediska praktické použitelnosti velmi zajímavý a který je očividně inspirován Nusseltovou analogií. Rozdíl spočívá v tom, že se místo polokoule nyní využívá poloviny krychle (obr. 4.9). I v tomto případě se předpokládá, že vyzařující ploška i je malá. Znovu se tedy jedná o výpočet koeficientu, pro který jsme dříve zavedli označení $F_{di \to j}$ a odvodili vztah (4.8). Základní myšlenka se opět opírá o pozorování, že při stanovení výkonu záření, které dopadá na plochu j, záleží na velikosti prostorového úhlu, pod kterým se plocha j při pohledu z místa plochy i jeví. Prostorový úhel můžeme vyšetřovat tak, že plochu j promítneme na povrch poloviny krychle opsané zářící plošce (obr. 4.9). Povrch krychle předpokládáme pokryt malými elementárními ploškami. Výkon záření dopadající na plochu j pak můžeme stanovit jako součet výkonů vycházejících z krychle ven těmi elementárními ploškami, které jsou součástí průmětu plochy j. Tomuto odpovídá konfigurační koeficient ve tvaru součtu

$$F_{di \to j} = \sum_{p \in \tilde{A}_j} \Delta F_p \ , \tag{4.10}$$

kde ΔF_p je konfigurační koeficient mezi zářící ploškou i a p-tou elementární ploškou na povrchu krychle. Sčítání se v sumě (4.10) provádí přes ty plošky p, které jsou součástí průmětu $\tilde{A}_j$ plochy j na povrch poloviny krychle.

Předpokládáme, že plošky na povrchu krychle jsou dostatečně malé (plochu každé z nich označíme ΔA), takže pro ně lze použít vztahu (4.7). Pro hodnotu ΔF_p pak máme (obr. 4.10)

$$\Delta F_p = \frac{\cos\varphi_i \cos\varphi_p}{\pi r^2} \Delta A \ . \tag{4.11}$$

Při praktickém výpočtu hodnoty ΔF_p je zapotřebí rozlišovat mezi ploškami na horní stěně poloviny krychle a ploškami na stěnách bočních. Pro plošky na horní stěně krychle vychází (obr. 4.10)

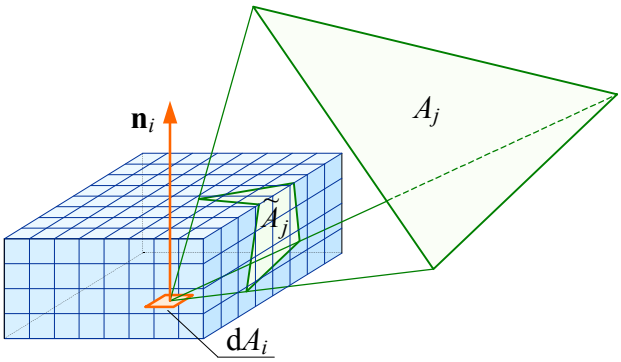
$$r = \sqrt{x_p^2 + y_p^2 + 1} \ , \quad \cos\varphi_i = \cos\varphi_p = \frac{1}{r} \ ,$$
$$\Delta F_p = \frac{\Delta A}{\pi\left(x_p^2 + y_p^2 + 1\right)^2} \ . \tag{4.12}$$

Pro plošky na bocích $x = \pm 1$ máme (pro boky $y = \pm 1$ platí vztahy analogické)

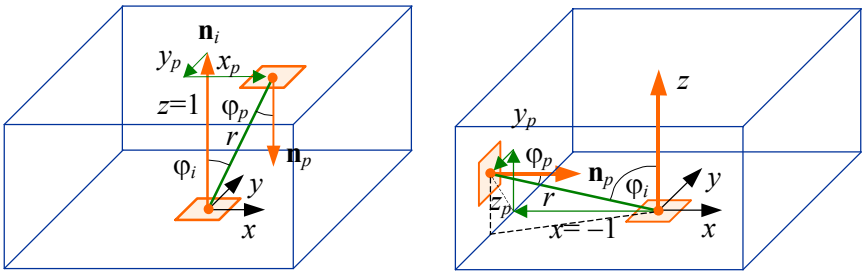
$$r = \sqrt{y_p^2 + z_p^2 + 1} \ , \quad \cos\varphi_i = \frac{z_p}{r} , \quad \cos\varphi_p = \frac{1}{r} \ ,$$
$$\Delta F_p = \frac{z_p \Delta A}{\pi\left(y_p^2 + z_p^2 + 1\right)^2} \ . \tag{4.13}$$

Některé další detaily přístupu využívajícího poloviny krychle

I v tomto případě připomínáme, že se na povrch krychle promítají jen ty části plochy *j*, které jsou z plochy *i* viditelné. Na neviditelné části plocha *i* nezáří, a proto se nepodílejí ani na výpočtu konfiguračních koeficientů. Právě při stanovení vzájemné viditelnosti ploch je popisovaná metoda výhodná. Všechny plochy viditelné ze zářící plochy *i* lze stanovit najednou pomocí algoritmu z-buffer. Pro každou z pěti stěn, které polovinu krychle ohraničují, je zřízena paměť hloubky a paměť obsahující odkaz na plochu, která je viditelná. Každé elementární plošce na povrchu krychle odpovídá jedna pozice v každé z obou pamětí. Zpracování probíhá postupem, který jsme popsali v kapitole 2. Plocha *i* (nějaký její bod) je středem projekce, roviny ohraničující polovinu krychle jsou průmětnami. Barva se v tomto případě nepočítá. Místo toho se pamatuje pouze odkaz na plochu, která je viditelná. Po zpracování všech ploch scény zůstanou k dispozici odkazy na plochy, které jsou ze zářící plochy *i* skrz jednotlivé elementární plošky viditelné. Této informace lze pak využít k hromadnému stanovení všech konfiguračních koeficientů, v nichž je plocha *i* plochou zářící. Řešení je ovšem třeba postupně provést pro všechna *i*, protože každá plocha ve scéně je současně také zářící plochou. Hodnoty ΔF_p ale stačí spočítat pouze jednou, protože jsou pro všechny polohy krychle (tedy pro všechny plochy *i*) stejné. Přesnost výpočtu konfiguračních koeficientů (ale současně bohužel také rychlost výpočtu) jsou samozřejmě závislé na jemnosti dělení povrchu krychle na dílčí plošky. Pro získání alespoň hrubé orientační představy lze říci, že délka hrany elementární plošky se obvykle pohybuje v rozmezí od 1/10 do 1/50, což odpovídá 20 až 100 ploškám podél hrany krychle.



Obr. 4.9. Výpočet konfiguračních koeficientů pomocí poloviny krychle.



Obr. 4.10. K výpočtu hodnoty ΔF_p pro jednu elementární plošku na horní stěně krychle (vlevo) a na boční stěně $x = -1$ (vpravo).

Výpočet koeficientů pro „velké“ zářící plochy

Pozorný čtenář má jistě stále na paměti, že jak Nusseltova metoda, tak i metoda využívající poloviny krychle předpokládají, že vyzařující ploška je malá. Obě tedy realizují výpočet vztahu (4.8). Postup, kterého by bylo možné použít pro plochy velké, ukazuje vztah (4.9) a komentář k němu. Vyzařující plocha by se jednoduše rozdělila na několik plošek dílčích tak, aby vzniklé plošky již bylo možné považovat za malé. Konfigurační koeficienty původní velké plochy by se pak získaly jako střední hodnota koeficientů vypočítaných pro plochy dílčí. Podrobněji tento postup popíšeme v následující podkapitole v souvislosti s hierarchickým zjemňováním dělení povrchů objektů na plochy.

Úkol 4.6

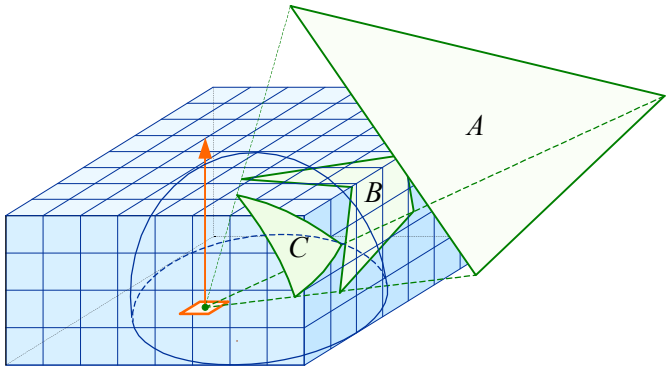
Úkol 4.6. Ukažte že pro $\rho_i < 1$ $(1 \le i \le n)$ je matice soustavy ze vztahu (4.5) skutečně striktně diagonálně dominantní (navazujeme zde na úkol 4.4).

Řešení. Nyní máte dobrou příležitost, abyste si připomenuli definici konfiguračních koeficientů. Protože předpokládáme, že všechen zářící výkon plošky i dopadne na nějaké plošky ve scéně, musí být $\sum_j F_{i\to j} = 1$. Odtud pak dále máme $\rho_i \sum_j F_{i\to j} < 1$ a konečně také $\rho_i \sum_{j\neq i} F_{i\to j} < 1 - \rho_i F_{i\to i}$, což již striktně diagonální dominanci dokazuje (viz úkol 4.4), protože hodnoty konfiguračních koeficientů i koeficientů odrazu jsou vždy kladné. □

Úkol 4.7. Ukažte, že postupy využívající k výpočtu konfiguračních koeficientů polokulové plochy a povrchu poloviny krychle jsou skutečně ekvivalentní.

Úkol 4.7

Řešení. Ekvivalenci lze jednoduše ukázat pomocí následujícího obrázku. Protože se plochy v obrázku označené jako A, B, C jeví z místa zářící plochy pod stejným prostorovým úhlem, dopadá na ně stejný zářící výkon, a konfigurační koeficienty proto jsou pro všechny tři plochy (vzhledem k ploše zářící) stejné.



□

Shrnutí: Nyní již znáte nejobvyklejší postup výpočtu konfiguračních koeficientů. (Kromě toho ovšem existují i další více či méně praktické metody, jejichž popis se ale vymyká z rámce tohoto textu.)

SHRNUTÍ

4.3 Adaptivní hierarchické dělení povrchů na plošky

Doba studia asi 1,5 hod

Čemu se zde naučíte?

Základní variantu vyzařovací metody jsme již v předchozích dvou podkapitolách probrali. V této podkapitole se již budeme věnovat pouze některým zajímavým a užitečným detailům, které se v souvislosti s vyzařovací metodou v průběhu času objevily. Zaměříme se zejména na adaptivní hierarchický přístup k dělení povrchu scény na plošky.

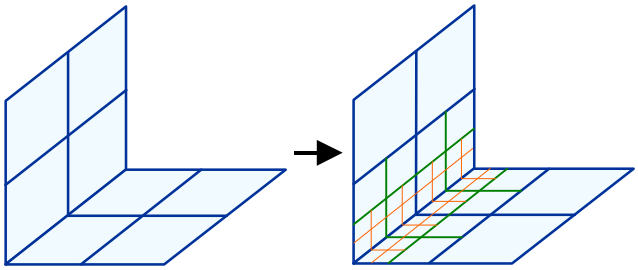
Nepožadujete-li hlubší znalosti, můžete podkapitolu přeskočit.

Proč má smysl adaptivního dělení používat?

V čem hierarchické adaptivní dělení spočívá?

Výpočet osvětlení vyzařovací metodou je potenciálně časově náročný. Největším problémem je v tomto ohledu výpočet velkého počtu konfiguračních koeficientů a řešení velkých soustav rovnic, na které metoda může vést. Často používaným postupem, který se pokouší udržet časovou složitost na přijatelné úrovni, je využití hierarchického adaptivního dělení povrchu scény na plošky. Postup spočívá v tom, že se jemnost dělení lokálně přizpůsobuje požadavkům vyzařovací metody, tak aby bylo dosaženo vyhovující přesnosti při co nejmenším celkovém počtu plošek. Termínem hierarchické se zdůrazňuje, že zjemňování proběhlo tak, že na místě jedné plošky, která se ukázala jako příliš velká, vzniklo několik plošek menších. Prakticky může proces hierarchického adaptivního dělení vypadat takto. Povrchy objektů ve scéně se nejprve pokryjí ploškami, které jsou zpravidla dosti velké (obr. 4.11). Ploška nevhodně velká může být rozpoznána ve dvou okamžicích: 1) Jestliže při výpočtu konfiguračních koeficientů vyjde hodnota některého koeficientu (řekněme koeficientu $F_{i\to j}$) velká (např. blízká jedné), pak to znamená, že téměř všechen výkon, který ploška i do prostoru vyzáří, dopadne

právě jen na plošku j. Už po vypočtení konfiguračního koeficientu $F_{i\to j}$ tak lze usuzovat, že je plocha j „podezřele“ velká. 2) Později lze nevhodně velké plošky odhalit po vyřešení systému rovnic ze vztahu (4.5). Vyjdou-li rozdíly vyzařování vedle sebe ležících plošek větší než nějaká mezní předem zvolená hodnota (gradient intenzity vyzařování je v uvažovaném místě velký), pak to signalizuje, že poměry osvětlení jsou v daném místě natolik složité, že současná hustota dělení zde nevyhovuje a plošky by měly být zmenšeny (zpravidla se jedná o místa, kudy prochází hranice vrženého stínu nebo vrženého světla (obr. 4.1, 4.3)). Obr. 4.11 ukazuje zahuštění dělení sítě (např. v rohu místnosti) postupným dělením plošek na čtvrtiny.



Obr. 4.11. Obrázek vlevo ukazuje hrubé počáteční dělení povrchu scény, např. v rohu místnosti. Vpravo je pak znázorněn výsledek adaptivního zjemnění dělení. Zahuštění dělení v rozích místností je obvykle žádoucí, protože rohy bývají tmavé.

Zajímavé důsledky hierarchického dělení

Hierarchické dělení ploch na plochy dílčí vykazuje ve vztahu k řešení soustavy rovnic (4.5) velmi zajímavou a prakticky užitečnou vlastnost, které lze využít ke snížení časové složitosti výpočtu. Uvažujme případ, kdy byla rozdělena právě jediná plocha i, a to na Q ploch dílčích, které označíme i_1, i_2,… , i_Q (obr. 4.12). Zapišme soustavu (4.5) pro situaci, která rozdělením vznikla. Matice soustavy, vektor hledaných vyzařování a vektor pravých stran mají nyní tvar

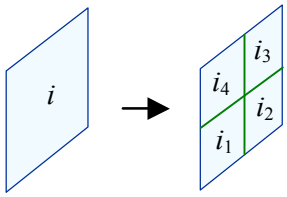
$$\begin{bmatrix}
1-\rho_1 F_{1\to 1} & \cdots & -\rho_1 F_{1\to i_1} & -\rho_1 F_{1\to i_2} & \cdots & -\rho_1 F_{1\to i_Q} & \cdots & -\rho_1 F_{1\to n} \\
\vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
-\rho_i F_{i_1\to 1} & \cdots & 1-\rho_i F_{i_1\to i_1} & -\rho_i F_{i_1\to i_2} & \cdots & -\rho_i F_{i_1\to i_Q} & \cdots & -\rho_i F_{i_1\to n} \\
-\rho_i F_{i_2\to 1} & \cdots & -\rho_i F_{i_2\to i_1} & 1-\rho_i F_{i_2\to i_2} & \cdots & -\rho_i F_{i_2\to i_Q} & \cdots & -\rho_i F_{i_2\to n} \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
-\rho_i F_{i_Q\to 1} & \cdots & -\rho_i F_{i_Q\to i_1} & -\rho_i F_{i_Q\to i_2} & \cdots & 1-\rho_i F_{i_Q\to i_Q} & \cdots & -\rho_i F_{i_Q\to n} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
-\rho_n F_{n\to 1} & \cdots & -\rho_n F_{n\to i_1} & -\rho_n F_{n\to i_2} & \cdots & -\rho_n F_{n\to i_Q} & \cdots & -\rho_n F_{n\to n}
\end{bmatrix},$$

$$\left(B_1,\dots,B_{i_1},B_{i_2},\dots,B_{i_Q},\dots,B_n\right)^{\mathrm{T}},\quad \left(E_1,\dots,E_{i_1},E_{i_2},\dots,E_{i_Q},\dots,E_n\right)^{\mathrm{T}}. \quad (4.14)$$

Nechť A_i značí velikost původní plochy a $A_{i1}, A_{i2}, \ldots, A_{iQ}$ nechť je velikost ploch dělením nově vzniklých. Proveďme nyní kondenzaci soustavy tak, že řádek odpovídající dílčí plošce i_q násobíme hodnotou A_{iq}/A_i a všechny řádky odpovídající dílčím ploškám vzniklým dělením plochy i sečtěme. Ve sloupci j matice soustavy, ve vektoru neznámých a ve vektoru pravých stran dostaneme popsaným postupem následující hodnoty

$$\frac{1}{A_i}\sum_{q=1}^{Q} A_{i_q} F_{i_q \to j} = F_{i \to j} \;, \quad \frac{1}{A_i}\sum_{q=1}^{Q} A_{i_q} B_{i_q} = B_i \;, \quad \frac{1}{A_i}\sum_{q=1}^{Q} A_{i_q} E_{i_q} = E_i \;. \tag{4.15}$$

Zdůvodnit oprávněnost zápisu rovností ve vztazích (4.15) není těžké. Jestliže jsme plochu i rozdělili, pak první ze vztahů (4.15) počítá vážený průměr konfiguračních koeficientů dílčích ploch vzniklých dělením (váhou je podíl velikosti plochy dílčí na velikosti původní plochy před rozdělením). To ale odpovídá postupu, kterým by měl být počítán konfigurační koeficient $F_{i \to j}$ pro velké zářící plochy, a také vztahu (4.9) z předchozí podkapitoly (z předchozí podkapitoly si připomeňte zdůvodnění tohoto vztahu). Podobně vážený průměr vyzařovaní dílčích plošek dává hodnotu vyzařování celé plochy i. To zdůvodňuje obě zbývající rovnosti.



Obr. 4.12. Hierachické dělení plošky i na plošky i_1, i_2, i_3, i_4.

Proveďme dále kondenzaci soustavy sečtením sloupců odpovídajících nově zavedeným plochám vzniklým dělením plochy i. Sečtením dostáváme (výsledek zde zapisujeme pro j-tý řádek)

$$\rho_j \sum_{q=1}^{Q} F_{j \to i_q} B_{i_q} = \frac{\rho_j}{A_j} \sum_{q=1}^{Q} F_{i_q \to j} A_{i_q} B_{i_q} = \rho_j F_{j \to i} B_i \;, \tag{4.16}$$

Při úpravě vztahu (4.16) jsme nejprve použili principu reciprocity. Pak jsme si povšimli, že výraz $(\Sigma_q\, F_{iq \to j}\, A_{iq}\, B_{iq}) \,/\, A_j$ počítá intenzitu záření, které od všech dílčích částí plochy i dopadne na jednotku plochy j. Jak již víme z podkapitoly 4.1, je ale také možné tuto intenzitu vyjádřit ve tvaru $F_{i \to j}\, A_i\, R_i \,/\, A_j$ nebo podle principu reciprocity také ve tvaru $F_{j \to i} R_i$ (připomeňte si konstrukci vztahů (4.1), (4.3)).

Hiearchické dělení nezvyšuje počet rovnic!

Nyní konečně ona slíbená zajímavá a užitečná vlastnost, kterou jsme prostřednictvím vztahů (4.15) a (4.16) zjistili: Při hierarchickém rozdělení ploch na plochy dílčí není nutné zvětšovat počet rovnic (my jsme to ve vztahu (4.14) udělali jen proto, abychom tuto vlastnost ukázali). Z hlediska scény jako celku a tedy i

z hlediska soustavy rovnic (4.5) lze rozdělení plochy považovat za postup, který vede pouze k přesnějšímu výpočtu konfiguračních koeficientů dělené plochy. Zvyšovat počet rovnic není nutné. I v případě, že dojde k rozdělení plochy (plocha i je rozdělena na Q částí), může ji v systému rovnic zastoupit pouze jediná hodnota záření B_i a jediná n-tice konfiguračních koeficientů $F_{i\to j}$ spočítaných podle prvního ze vztahů (4.15). Jestliže je soustava rovnic pro „velké“ plochy vyřešena (dělením ploch byly zatím pouze zpřesněny konfigurační koeficienty), lze přistoupit k dopočítání hodnot vyzařování dílčích plošek vzniklých dělením. Hodnoty vyzařování B_{i1}, B_{i2}, … , B_{iQ} získáme řešením soustavy Q rovnic, která vznikne ze soustavy (4.14) tak, že řádky odpovídající dílčím plochám i_1, i_2,… , i_Q tentokrát naopak ponecháme. Protože kromě hodnot B_{i1} až B_{iQ} jsou již nyní ostatní hodnoty vyzařování známé, jedná se skutečně o soustavu Q rovnic pro záření dílčích částí plochy i. Tento zdánlivě komplikovaný postup je výhodný z hlediska časové složitosti výpočtu osvětlení (úkol 4.8).

Úkol 4.8

Úkol 4.8. Porovnejte časovou složitost algoritmu bez a s využitím hierarchického dělení na dílčí plochy. Pro jednoduchost předpokládejte, že scéna má M ploch a že každá z nich vyžaduje rozdělení na Q ploch dílčích.

Řešení. Pokud byste soustavu (4.5) sestavili bez ohledu na skutečnost, že dělení je hierarchické, pak by měla MQ rovnic a bylo by zapotřebí stanovit $(MQ)^2$ konfiguračních koeficientů. Pokud naopak vlastností hierarchického dělení využijete, bude mít soustava jen M rovnic. Konfiguračních koeficientů bude jen M^2, ale výpočet každého z nich bude pracnější, protože bude vyžadovat výpočet Q koeficientů dílčích, z nichž teprve bude výsledný koeficient počítán jako průměr podle prvního ze vztahů (4.15). Po vyřešení soustavy bude v tomto případě zapotřebí ještě dále dopočítat záření jednotlivých dílčích plošek vzniklých dělením. To si pro každou z původních M ploch scény vyžádá čas QM. Celkem tedy bude dopočet záření pro všechny dílčí plošky vyžadovat čas QM^2. □

Úkol 4.9

Úkol 4.9..Promyslete, zda a jak by bylo možné kombinovat vyzařovací metodu zobrazování s metodou sledování paprsku.

Řešení. Různé kombinace obou metod jsou skutečně časté. Jak jsme už vysvětlili v úvodu této kapitoly, akcentuje vyzařovací metoda odrazy difúzní a metoda sledování paprsků odrazy zrcadlové. Lze proto očekávat, že by spojením mohla vzniknout metoda, která bere v úvahu oba typy odrazu vyváženě. Za nejpřímočařejší kombinaci obou technik lze považovat postup, kdy se vyzařovací metodou spočítá osvětlení scény. Scéna se pak zobrazuje metodou sledování paprsku. Osvětlení vypočtené vyzařovací metodou se použije místo intenzity $\mathbf{I}_a$ okolního osvětlení v prvním členu vztahu (3.2). □

S H R N U T Í

Shrnutí: V tuto chvíli byste měli vcelku dobře znát vyzařovací metodu v její základní a nejobvyklejší variantě. To by vám mělo umožnit, abyste dokázali kvalifikovaně obsluhovat programy, které ji využívají, a mohli tak např. modelovat vlastní scény a vytvářet vlastní obrázky. Jestliže byste se snad chtěli pustit i do vlastní implementace, pak je nutné poznamenat, že metoda je dodnes rozvíjena a neustále se objevují její nové více či méně praktické varianty, se kterými byste se v takovém případě měli rovněž seznámit. Seznámit byste se pak samozřejmě měli i s již existujícími komerčními produkty. Z knižních publikací pojednávajících výlučně o vyzařovací metodě lze doporučit následující:

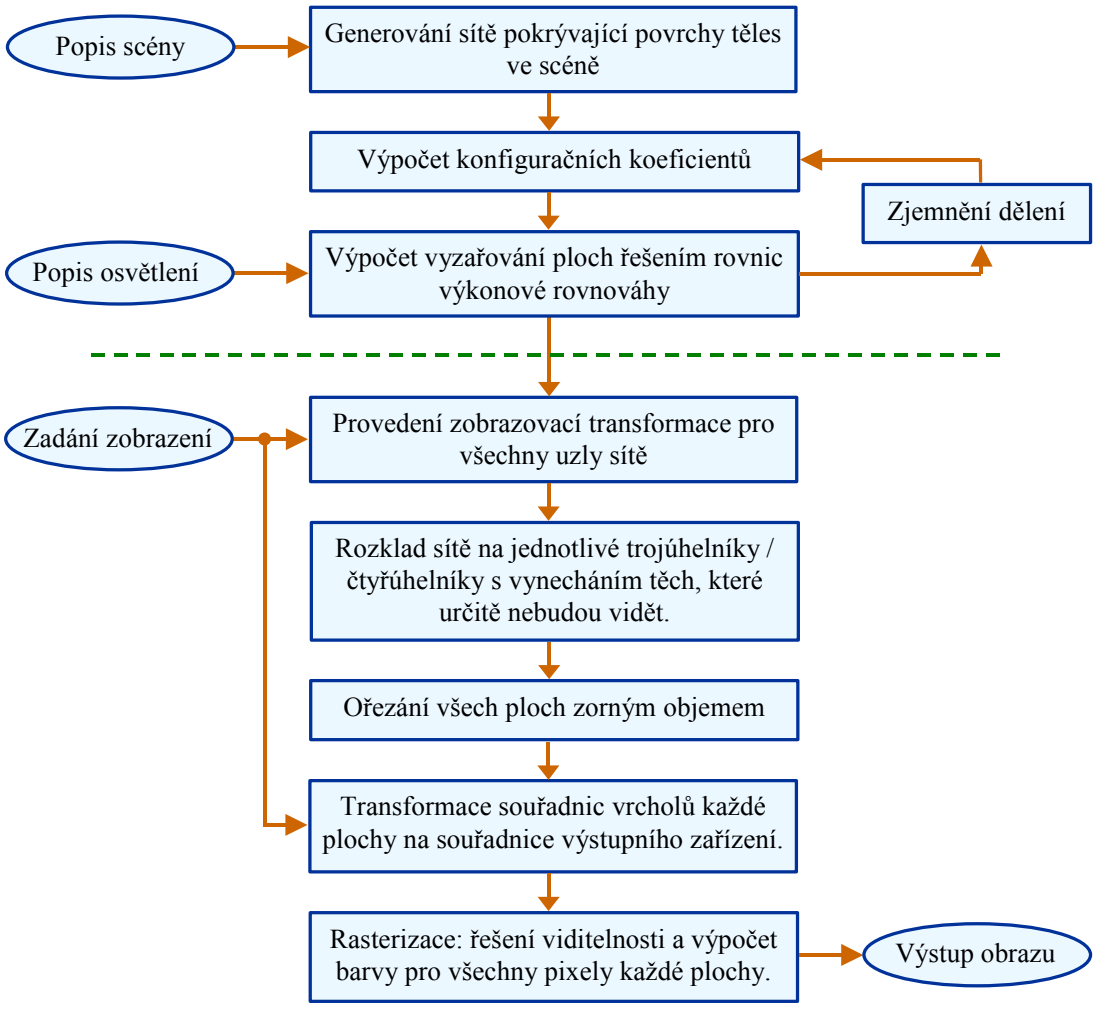
Doporučená literatura

- François X. Sillion and Claude Puech, Radiosity and Global Illumination, *Morgan Kaufman Publishers*, San Francisco, 1994, 251 pages, ISBN 1-55860-277-1,
- Ian Ashdown, Radiosity: A Programmer's Perspective, *John Wiley & Sons*, 1994, 497 pages, ISBN 0-471-30444-1.

Obě již jsou, jak je vidět, poněkud staršího data, ale jiné zatím nevyšly. Jednotlivých článků v časopisech lze naproti tomu nalézt nepřeberné množství. Čerstvý (2003) a velmi obsáhlý bibliografický přehled (kolem tří set článků) lze nalézt na http://www.helios32.com.

Řešení úkolu 4.5

Jako řešení úkolu 4.5 nakonec uvádíme schéma postupu kroků vyzařovací metody.



5 Programování v OpenGL

Proč by vás mohlo rozhraní OpenGL zajímat?

OpenGL je grafická knihovna určená pro zobrazování 2D a 3D objektů vyvinutá na počátku devadesátých let společností Silicon Graphics, a to původně pro operační systém IRIX. V průběhu času byla knihovna exportována do mnoha dalších operačních systémů a s postupně rostoucím výkonem osobních počítačů našla své uplatnění i zde. V současné době lze OpenGL považovat za všeobecně uznávaný standard. Svědčí o tom mimo jiné i široká technická podpora, které se OpenGL dostává od výrobců grafických karet. V tomto případě lze knihovnu OpenGL současně také chápat jako standardní programové rozhraní pro technická zařízení (hardware) od různých výrobců. Za optimálních podmínek, tj. při maximální technické podpoře, OpenGL více či méně pouze vyvolává příslušné funkce realizované na technické úrovni. Pokud některá funkce na technické úrovni podporována není, je realizována programově. Tento postup umožňuje dosáhnout maximálního výkonu při současném zachování technické nezávislosti programů. Knihovnu lze ovšem samozřejmě používat i bez jakékoli technické podpory. Všechny funkce knihovny jsou pak realizovány programově.

Použití OpenGL lze určitě doporučit!

Standard OpenGL definuje množinu funkcí, které se volají z uživatelského programu. Definice funkcí vyhovuje normě ANSI C. Objektově orientovaného programování není využito. Více než desetiletý vývoj ukazuje, že lze standard považovat za velmi úspěšný. Pokud byste měli realizovat program s grafickými výstupy, lze použití OpenGL jednoznačně doporučit. Typické výhody, které tím získáte, jsou následující: 1) Standard je vcelku jednoduchý (snadno jej bez přílišné námahy zvládnete). 2) I přes svoji jednoduchost standard nabízí dosti široké a dostatečné možnosti. 3) Použitím standardu získáte zcela automaticky přístup k urychlování grafichých operací technickými prostředky, a to bez toho, abyste se starali o to, v jaké míře jsou tyto prostředky (či zda vůbec jsou) na cílovém počítači přítomny. 4) Váš program bude snadno přenositelný mezi různými operačními systémy (např. MS Windows, UNIX / LINUX).

Čemu se v této kapitole naučíte?

V této kapitole se seznámíte se základy programování v OpenGL. Kapitola je pojata tak, abyste se na co nejmenším prostoru seznámili s maximem toho, co lze v OpenGL považovat za podstatné. Po prostudování kapitoly si učiníte poměrně přesný názor na to, jaké možnosti OpenGL má. Budete samozřejmě také schopni vytvářet vlastní programy, které standardu OpenGL využívají. Výslovně poznamenáváme, že cílem kapitoly určitě není prezentovat úplné (a úmorné) přehledy všech funkcí, konstant atd., které v OpenGL existují. Takový přístup by byl pro začátečníka sotva vhodný. Věříme ale, že po přečtení kapitoly vám k doplnění detailních informací v mnoha případech postačí už jen soubor s nápovědou, který se k OpenGL dodává. Při výkladu budeme vycházet z verze 1.1, se kterou se setkáte asi nejsnáze, protože je dostupná v MS Windows.

Úkol 5.1

Úkol 5.1. Navštivte internetové stránky http://www.opengl.org a také stránky http://www.mesa3d.org (softwarový produkt „Mesa“ je rovněž implementací OpenGL). Je možné, že „atmosféra“ stránek bude pro vás dalším impulsem ke studiu standardu.

5.1 Syntaxe příkazů OpenGL (a jiné)

Doba studia asi 0,5 hod

Čemu se zde naučíte?

V této podkapitole uvedeme několik všeobecných informací týkajících se zejména formální stránky zápisu příkazů (funkcí) OpenGL. Její prostudování vám umožní lépe sledovat výklad v následujících podkapitolách. Doporučuji tuto kapitolu při studiu nepřeskakovat. Funkcí OpenGL je hodně, a každá pomůcka, která může pomoci v orientaci, je proto jistě vítána.

Pojmenování funkcí OpenGL: předpony a přípony

Příkazy OpenGL začínají předponou **gl** a každé slovo tvořící název příkazu (funkce) pak začíná velkým písmenem (např. **glClearColor**). Konstanty definované v OpenGL začínají předponou GL_. Všechny znaky jsou pak psány velkými písmeny a k oddělení jednotlivých slov v názvu konstanty je použit znak „_“ (např. GL_COLOR_BUFFER_BIT). Názvy některých příkazů mají příponu, která se skládá z číslice a z jednoho nebo dvou znaků (např. **glColor3f**). Číslice udává počet předávaných parametrů a znaky udávají jejich typ. Podrobnosti vysvětluje tabulka 5.1. Některé příkazy, se kterými se zde setkáte, budou začínat nejen předponou **gl**, ale také předponami **glu**, případně **glaux**. Předponou glu budou začínat některé „pěkné“ a užitečné příkazy, které však, přísně vzato, nejsou v OpenGL příkazy základními. Jedná se o tzv. „utility“. Předponou glaux bude začínat relativně malý počet příkazů z rozšiřující knihovny, která řeší zejména vazby na konkrétní platformu, na níž OpenGL běží.

Tabulka. 5.1. Přípony jmen funkcí a odpovídající datové typy parametrů.

Přípony a odpovídající datové typy parametrů

Pří-pona	Datový typ parametrů	Korespondující typ jazyka C	Definice v OpenGL
b	8-bitové celé číslo	signed char	GLbyte
s	16-bitové celé číslo	short	GLshort
i	32-bitové celé číslo	long	GLint, GLsizei
f	32-bitové číslo s plovoucí čárkou	float	GLfloat, GLclampf
d	64-bitové číslo s plovoucí čárkou	double	GLdouble, GLclampd
ub	8-bitové celé číslo bez znaménka	unsigned char	GLubyte, GLboolean
us	16-bitové celé číslo bez znaménka	unsigned short	GLushort
ui	32-bitové celé číslo bez znaménka	unsigned long	GLuint, GLenum, GLbitfield

„Vektorová a nevektorová“ varianta funkcí

Jména části funkcí OpenGL končí znakem **v**. Znamená to, že parametrem funkce je ukazatel na pole (vektor) hodnot, místo množiny individuálních parametrů. Mnoho funkcí má vektorovou i nevektorovou variantu, některé funkce však akceptují pouze individuální argumenty nebo naopak zase jen parametr ve formě vektoru. V následujícím příkladě jsou na příkazu pro nastavení barvy kreslení ukázány obě varianty (obě dělají totéž).

```
glColor3f(1.0, 0.0, 0.0);   /*varianta se třemi parametry*/
float color_array[] = {1.0, 0.0, 0.0};
glColor3fv(color_array);    /*varianta s vektorem*/
```

OpenGL jako stavový stroj

OpenGL se chová jako stavový stroj. Pokud je uveden do nějakého stavu, pak v tomto stavu setrvává do té doby, než je vyžádána změna stavu. Například barva, která byla nastavena v předcházejícím příkladě, platí pro všechnu dále

prováděnou kresbu až do doby, než je nastavena barva jiná. OpenGL udržuje velké množství stavových proměnných. Aktuální barva je jednou z nich. S několika dalšími se seznámíte postupně v průběhu studia. Některé stavové proměnné se vztahují k tomu, zda je nějaká možnost OpenGL zapnuta nebo vypnuta. Zapnutí nebo vypnutí se provádí voláním funkcí **glEnable** a **glDisable**. Parametr říká, co se má zapnout nebo vypnout. Následující příklad ukazuje zapnutí a pak vypnutí řešení viditelnosti (v OpenGL se provádí metodou z-buffer).

Povolení a zakázání funkcí

```
glEnable(GL_DEPTH_TEST);     /*povoluje používání z-bufferu*/
glDisable(GL_DEPTH_TEST);    /*zakazuje používání z-bufferu*/
```

Každá stavová proměnná má nějakou svoji přednastavenou hodnotu. Na právě platnou hodnotu většiny proměnných se může programátor kdykoli zeptat. Lze k tomu použít jednu z funkcí **glGetBooleanv**, **glGetIntegerv**, **glGetFloatv** nebo **glGetDoublev**. Která z nich to bude, záleží na typu proměnné, jejíž hodnota se zjišťuje, a na požadovaném typu výsledku. Všechny uvedené funkce mají dva parametry. První je konstanta specifikující proměnnou, jejíž hodnota má být zjištěna (přípustné konstanty jsou v OpenGL předdefinovány). Druhý parametr je adresa místa v paměti, kam má být zjištěná hodnota uložena. Následující příklad ukazuje, jak je možné zjistit následující: 1) Zda je povoleno používání z-bufferu (řešení viditelnosti). 2) Zda je povolen výpočet osvětlení scény. 3) Jaký je maximální počet světel, kterého lze v dané implementaci OpenGL použít. 4) Jaká tloušťka čáry je právě nastavena pro kreslení. 5) Jaká je v dané implementaci OpenGL maximální možná tloušťka čáry. 6) Jaká je právě nastavená barva.

Zjišťování hodnoty stavových proměnných

```
glGetBooleanv(GL_DEPTH_TEST, povoleno_reseni_viditelnosti);
glGetBooleanv(GL_LIGHTING, povoleno_osvetlovani);
glGetIntegerv(GL_MAX_LIGHTS, max_pocet_svetel);
glGetFloatv(GL_LINE_WIDTH, prave_nastavena_tloustka_cary);
glGetFloatv(GL_LINE_WIDTH_RANGE, maximalni_tloustka_cary);
float prave_platna_barva[4];
glGetFloatv(GL_CURRENT_COLOR, prave_platna_barva);
```

Možností, na co se lze v OpenGL zeptat, je mnoho. Pro podrobnější informaci by vám ale nyní už měla postačit nápověda, která bývá s OpenGL dodávána.

Pro pořádek ještě poznamenejme, že pro přístup k některým proměnným existují také funkce speciální. Důležitá je např. funkce **glGetError** umožňující zjistit kód chyby, která v dříve prováděných krocích eventuálně nastala. Funkce nemá parametr a kód chyby vrací jako funkční hodnotu. Vyvoláním funkce se také současně záznam o chybách nuluje (nastavuje se na hodnotu GL_NO_ERROR). Následuje praktický příklad, jak byste funkci mohli využít ve svých programech.

Speciální dotazovací funkce

```
if (glGetError() != GL_NO_ERROR){
   printf("Nekde v predchozim se stala chyba!");
   return -1;
}
...
if (glGetError() == GL_OUT_OF_MEMORY){
   printf("Neni dost pameti!");
   return -1;
}
```

Další speciální dotazovací funkcí je např. funkce **glIsEnabled**. Ta svojí funkční hodnotou (boolean) říká, zda je v programu právě dovoleno to, co je specifikováno pomocí parametru funkce (tedy zda to programátor dříve dovolil).

5.2 Kreslení základních geometrických útvarů

Doba studia asi 1,5 hod

V této podkapitole ukážeme, jaké základní geometrické útvary lze v OpenGL kreslit a jak lze nastavit vlastnosti kresby. Množina základních útvarů se vám bude možná zdát poněkud chudá. To je ale do značné míry záměrné. Standard OpenGL byl totiž navržen tak jednoduchý, aby byla možná podpora jeho funkcí technickými prostředky. (Později se dovíte, že kromě základních útvarů popsaných v této podkapitole umí OpenGL kreslit ještě Bèzierovy a NURBS křivky a plochy.)

Čemu se zde naučíte?

Při kreslení geometrických útvarů se samozřejmě nelze obejít bez zadávání souřadnic bodů. Chceme-li např. nakreslit trojúhelník, pak bude určitě nutné zadat souřadnice jeho vrcholů. K zadávání polohy bodů (vrcholů) má OpenGL funkci **glVertex**. Tu probereme nejdříve, protože se využívá při kreslení kteréhokoli ze základních útvarů. Funkce má následující tvar:

void **glVertex** {234}{sifd}[v] (TYPE *coords*);

Funkce **glVertex** slouží k zadání souřadnic jednoho bodu. Procvičme ještě jednou čtení výše uvedeného zápisu syntaxe (vysvětlili jsme jej už v minulé podkapitole). O funkci jsme zatím pro stručnost mluvili jako o funkci **glVertex**. Nyní ale můžeme upřesnit, že funkce má povinnou příponu. Přípona se skládá z jednoho čísla (buď **2** nebo **3** nebo **4**). Číslo udává, kolik souřadnic bude pro bod zadáno a tím také určuje počet parametrů (s výjimkou vektorové varianty zadání, kterou popíšeme dále). Složenými závorkami {.} se vyznačuje, že vždy musí být vybrána právě jedna položka z množiny symbolů uvnitř. Možnými variantami, které lze vybrat, jsou tedy buď dvojice (x, y), nebo trojice (x, y, z) a nebo čtveřice (x, y, z, w) souřadnic. Druhá část přípony je tvořena buď znakem **s**, nebo **i**, nebo **f** a nebo znakem **d**. Znak určuje, jakého typu budou proměnné použité jako parametry. Vysvětlení podává tabulka 5.1. Konečně třetí část přípony je nepovinná (nepovinnost vyznačují hranaté závorky []). Jako nepovinná část se v případě funkce glVertex může objevit pouze znak **v**. Ten říká, že jsou hodnoty parametrů (souřadnic bodu) připraveny v poli (vektoru). Funkci glVertex bude proto pak už předávána pouze adresa tohoto pole. Předpokládá se, že jednotlivé souřadnice leží v poli za sebou. Slovo „TYPE“ v deklaraci prametru *coords* říká, že parametr bude takového typu, jaký vyplývá z použitých přípon. Následující příklad ukazuje několik typických variant použití funkce glVertex.

Zadání souřadnic pomocí funkce glVertex

```
glVertex2s(2, 3);
glVertex3f(0.0, 0.0, 3.14);
glVertex4d(2.3, 1.0, -2.2, 2.0);
float coordinates[] = {1.0, 0.0, 0.0, 1.0};
glVertex4fv(coordinates);
```

Je užitečné vědět, že souřadnice jsou v OpenGL vždy uchovávány jako souřadnice o čtyřech složkách. Jestliže si programátor přeje složek méně, pak jsou chybějící hodnoty w a případně z automaticky doplňovány na hodnoty $w = 1$ a $z = 0$. Pro zajímavost ještě konečně poznamenejme, že funkce **glVertex** je v OpenGL explicitně definována ve všech čtyřiadvaceti variantách, které mohou být kombinacemi znaků v příponě vytvořeny (což jste ostatně asi ale čekali).

Nyní, když už víme, jak se specifikují vrcholy, můžeme přistoupit ke slíbenému kreslení elementárních geometrických útvarů. Kreslení všech útvarů se provádí v podstatě stejně, a to takto: Kreslení útvaru je uvozeno voláním funkce

Kreslení základních útvarů

glBegin. Funkce má jediný parametr, který říká, jaký útvar se má nakreslit. Typickou akcí, která se provádí po zahájení kresby, je zadávání souřadnic vrcholů pomocí funkce **glVertex**. Kreslení útvaru je ukončeno voláním funkce **glEnd**. Poněkud formálnější pohled nabízí následující definice syntaxe.

```
void glBegin(GLenum mode);
void glEnd(void);
```

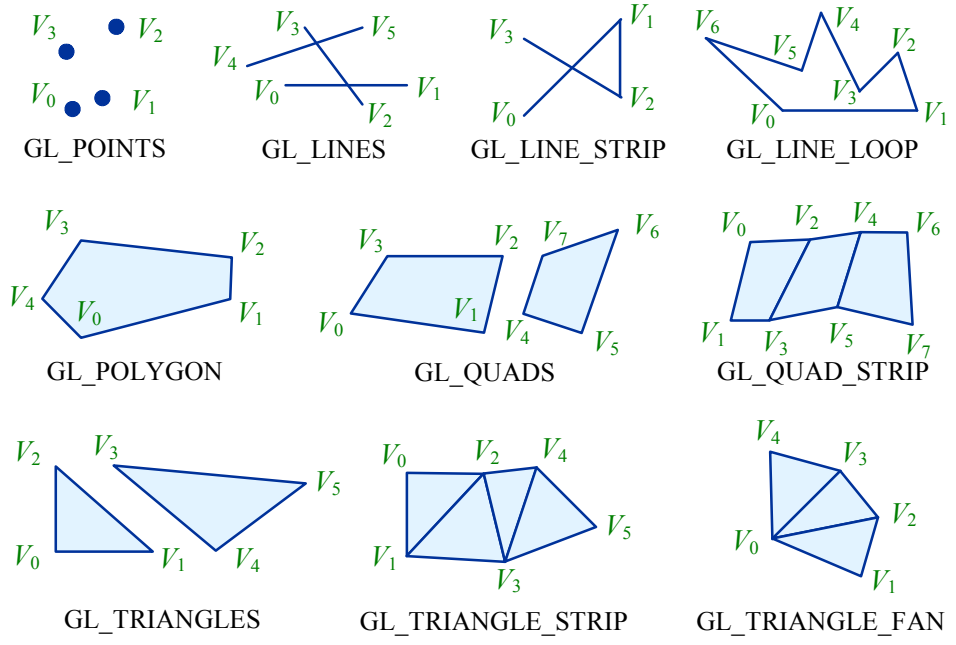
Funkce glBegin a glEnd

Možné hodnoty parametru *mode*, který udává, co se má nakreslit, jsou uvedeny v tabulce 5.2. Odpovídající základní geometrické útvary jsou pak také znázorněny na obr. 5.1.

Tabulka 5.2. Seznam základních geometrických útvarů v OpenGL.

Hodnota *mode*	Význam
GL_POINTS	jednotlivé body
GL_LINES	jednotlivé úsečky
GL_POLYGON	jednoduchý konvexní polygon
GL_TRIANGLES	jednotlivé trojúhelníky
GL_QUADS	jednotlivé čtyřúhelníky
GL_LINE_STRIP	posloupnost na sebe navazujících úseček
GL_LINE_LOOP	uzavřená posloupnost na sebe navazujících úseček
GL_TRIANGLE_STRIP	množina trojúhelníků tvořících pás
GL_TRIANGLE_FAN	množina trojúhelníků tvořících vějíř
GL_QUAD_STRIP	množina čtyřúhelníků tvořících pás

Co lze v OpenGL nakreslit?



Základní deometrické útvary v OpenGL

Obr. 5.1. Základní geometrické útvary v OpenGL.

Následují dvě ukázky kreslení v OpenGL. V první jsou nakresleny dvě úsečky, ve druhé vějíř trojúhelníků. V obou případech předpokládáme, že jsou nastaveny nějaké rozumné parametry kreslení, jako je např. barva, tloušťka a typ čáry atd. (zatím je nastavit neumíte, ale ještě v této podkapitole se to naučíte).

Příklad kreslení úseček

```
glBegin(GL_LINES);
   glVertex2f(0.0, 0.0);
   glVertex2f(5.0, 0.0);
   glVertex3f(1.0, 1.0, 1.0);
   glVertex3f(5.0, 5.0, 5.0);
glEnd();
```

Příklad kreslení vějíře trojúhelníků

```
glBegin(GL_TRIANGLE_FAN);
   glVertex3f( 0.0, 0.0, 0.0);
   glVertex3f( 1.4, 0.0, 0.0);
   glVertex3f( 1.0, 1.0, 0.1);
   glVertex3f( 0.0, 1.4, 0.2);
   glVertex3f(-1.0, 1.0, 0.3);
glEnd();
```

Ačkoli je funkce glVertex nejtypičtější funkcí, jejíž volání se mezi dvojicí glBegin, glEnd vyskytuje, zdaleka to není funkce jediná možná. K dokreslení pohledu na možnosti kreslení v OpenGL napomůže, když uvedeme, že se mezi dvojicí glBegin, glEnd smí vyskytnout následující: 1) příkazy jazyka C, 2) volání funkcí OpenGL uvedených v tabulce 5.3. Užitečnost první varianty je zřejmá. Ilustruje ji následující příklad, v němž se pomocí uzavřené posloupnosti úseček kreslí kružnice (jak již víme, kružnice v nabídce základních útvarů OpenGL není). K nakreslení je ovšem zapotřebí spočítat souřadnice bodů na kružnici. K tomu je výhodné použít prostředků programovacího jazyka.

Co lze provádět mezi glBegin a glEnd?

Používat konstrukce jazyka C!

```
#define PI 3.1415926535897
GLint circle_points = 100;
glBegin(GL_LINE_LOOP);
   for (i=0; i < circle_points; i++ ){
      angle = 2.0*PI*i / circle_points;
      glVertex2f(cos(angle),sin(angle));
   }
glEnd();
```

Také možnost volat funkce uvedené v tabulce 5.3 je užitečná. Třebaže se ve většině případů jedná o funkce, které jsme zatím systematicky neprobrali, je dost pravděpodobné, že na základě svých znalostí z předchozích kapitol významu mnoha z nich alespoň rámcově rozumíte. Opět uvedeme příklad. Řekněme, že chcete nakreslit trojúhelník, který bude vystínován a pokryt texturou. Víme, že k výpočtu osvětlení je zapotřebí specifikovat normály ve vrcholech trojúhelníka (kapitola 2.5). K nanesení textury pak bude zase zapotřebí pro každý vrchol trojúhelníka stanovit jeho souřadnice v textuře (kapitola 2.11). Část programu, v níž je trojúhelník popsán, by pak mohla vypadat např. takto:

Mezi glBegin a glEnd lze také volat funkce z tabulky 5.3!

```
glBegin(GL_TRIANGLES);
   /* Následující tři řádky se vztahují k prvnímu vrcholu.*/
   glNormal3f(0.0, 0.0, 1.0);      /* nastavení normály   */
   glTexCoord2f(0.0, 0.0);         /* souřadnice v textuře*/
   glVertex3f(2.0, 0.0, 0.0);      /*.souřadnice vrcholu  */
   /* Podobné trojice řádků budou i pro zbývající vrcholy.*/
   ...
glEnd();
```

Tabulka 5.3. Funkce OpenGL, které lze volat mezi glBegin a glEnd.

Funkce	Co funkce provádí
glVertex	nastavení souřadnic vrcholu
glColor	nastavení aktuální barvy
glIndex	nastavení aktuálního barevného indexu
glNormal	nastavení normálového vektoru
glEvalCoord	generování souřadnic (podkapitola 5.9)
glCallList, glCallLists	provedení zobrazovacího seznamu (podkapitola 5.5)
glTexCoord	nastavení souřadnic textury (podkapitola 5.8)
glEdgeFlag	řízení vykreslování hran (tato podkapitola)
glMaterial	nastavení materiálových vlastností (podkapitola 5.6)

Funkce přípustné mezi glBegin / glEnd

Ve zbytku této podkapitoly popíšeme některé významnější funkce, které nastavují parametry kresby. Začneme nastavením barvy.

void **glColor3**{b s i f d ub us ui} (TYPE *r*, TYPE *g*, TYPE *b*);
void **glColor4**{b s i f d ub us ui} (TYPE *r*, TYPE *g*, TYPE *b*, TYPE *a*);
void **glColor3**{b s i f d ub us ui}**v** (const TYPE **v*);
void **glColor4**{b s i f d ub us ui}**v** (const TYPE **v*);

Nastavení barvy

Funkce **glColor** slouží k nastavení barvy. Nastavená barva platí pro následně kreslené objekty tak dlouho, dokud není zvolena barva jiná. Varianta se čtyřmi složkami předpokládá zadání barevných složek *r*, *g*, *b* a hodnoty tzv. „alfa kanálu“ (hodnota *a*). Alfa kanál zpravidla slouží k míchání obrazů (příklad jeho použití uvidíte v souvislosti s nanášením textury). Pokud je použita verze příkazu se třemi parametry, je hodnota alfa kanálu automaticky nastavena na hodnotu 1.0. Tuto hodnotu lze považovat za nejběžnější (naznačuje, že žádné míchání obrazů nejspíše nebude prováděno). Vedle zadávání barvy jejími složkami *r*, *g*, *b*, lze barvu specifikovat také pomocí jejího indexu v paletě, a to pomocí funkce **glIndex**. Tuto možnost zde ale podrobněji rozebírat nebudeme, protože se dnes již zdá poněkud méně častá než zadání pomocí barevných složek.

void **glPointSize**(GLfloat *size*);

Nastavení velikosti značek bodů

Funkce **glPointSize** nastavuje velikost bodů kreslených pomocí GL_POINTS. Velikost se zadává jako reálné číslo (větší než nula) udávající velikost v obrazových bodech (pixelech). Přednastavená velikost je jeden bod.

void **glLineWidth**(GLfloat *width*);

Nastavení tloušťky čáry

Funkce **glLineWidth** nastavuje šířku kreslených čar. Šířka se opět zadává jako reálné číslo (větší než nula) znamenající šířku čáry v obrazových bodech. Přednastavená tloušťka je jeden obrazový bod. Maximální tloušťka čáry bývá omezená. Stejně tak nelze pochopitelně očekávat, že se na výsledném obraze bude tloušťka čáry měnit zcela spojitě, jak by to snad zadání tloušťky reálným číslem mohlo naznačovat. Jaké vlastnosti má v tomto ohledu vaše implementace OpenGL, můžete zjistit pomocí funkce **glGet** (např. **glGetFloatv**) s parametry GL_LINE_WIDTH_RANGE, GL_LINE_WIDTH_GRANULARITY.

void **glLineStipple**(GLint *factor*, GLushort *pattern*);

Nastavení typu čáry

Funkce **glLineStipple** nastavuje způsob kreslení čar (např. plná, čárkovaná, tečkovaná atd). Parametr *pattern* svými 16 bity definuje vzor, kterým je čára

kreslena. Jedničkový bit znamená, že se „kousek“ čáry nakreslí, nulový bit, že se naopak nekreslí. Např. hodnota 1010101010101010 způsobí kreslení „husté“ tečkované čáry. Parametr *factor* se používá jako měřítko pro vzor. Jestliže by např. pro výše uvedený vzor byla jeho hodnota zadána 2, pak by to bylo totéž, jako by byl zadán vzor 11001100… . Dříve, než začnete čáry s využitím vzoru vykreslovat, musíte tuto možnost kreslení povolit, což se provede voláním funkce **glEnable** s parametrem GL_LINE_STIPPLE. Budete-li chtít později tuto možnost naopak zase zakázat, provedete to voláním funkce **glDisable** se stejným parametrem. Následující příklad ukazuje nastavení tloušťky a typu čáry.

Příklad nastavení čáry

```
glLineWidth(2.0);
glLineStipple(1, 0x3F07);
glEnable(GL_LINE_STIPPLE);
```

void **glPolygonMode**(GLenum *face*, GLenum *mode*);

Nastavení způsobu kreslení polygonu

Voláním funkce **glPolygonMode** lze řídit způsob zobrazování polygonů. Polygony se předpokládají rovinné a konvexní. Rovina polygonu dělí prostor na dvě části. Z každé z obou částí prostoru lze vidět jen jednu stranu polygonu (je to totéž, jako kdybyste polygon vystřihli z papíru, také bude mít dvě strany). OpenGL pojmenovává strany polygonů „přední“ (FRONT) a „zadní“ (BACK). Přední strana je ta, která je přivrácená k pozorovateli. Jak OpenGL pozná, která strana je přední a která zadní, ukážeme v souvislosti s následující funkcí **glFrontFace**. Parametr *face* říká, kterých polygonů se právě specifikovaný požadavek na zobrazování polygonů týká. Může nabývat hodnoty GL_FRONT (specifikace platí jen pro přivrácené polygony) nebo hodnoty GL_BACK (jen pro odvrácené polygony) nebo GL_FRONT_AND_BACK (pro polygony přivrácené i odvrácené). Parametr *mode* pak konečně určuje, jakým způsobem se polygon zobrazí. Mohou se vykreslit pouze vrcholy polygonu (GL_POINT), nebo jeho hrany (GL_LINE) nebo může být polygon vyplněn (GL_FILL).

void **glFrontFace**(GLenum *mode*);

Určení „přední“ strany polygonu

Voláním funkce **glFrontFace** lze OpenGL sdělit, jak má stanovit, která plocha polygonu je přivrácená k pozorovateli (přední). Parametr *mode* může nabývat buď hodnoty GL_CCW (plocha je přivrácená, když se její vrcholy jeví uspořádány proti směru hodinových ručiček) nebo hodnoty GL_CW (po směru ručiček).

void **glCullFace**(GLenum *mode*);

Řízení odstraňování určitě neviditelné strany polygonu

Voláním funkce **glCullFace** se stanoví, které strany polygonů budou při zobrazování okamžitě odstraněny jako určitě neviditelné. Je možný jeden ze dvou módů: GL_BACK (budou odstraněny zadní strany polygonů) nebo GL_FRONT (přední strany). Přednastavená hodnota je GL_BACK. Odstraňování je zapotřebí povolit voláním funkce **glEnable** s parametrem GL_CULL_FACE.

void **glPolygonStipple**(const GLubyte **mask*);

Nastavení způsobu vyplňování polygonu

Funkce **glPolygonStipple** definuje aktuální vzor pro vyplňování polygonů. Argument *mask* je ukazatel na bitovou mapu o velikosti 32×32 bitů. Jedničkový bit v mapě znamená, že korespondující pixel polygonu bude vykreslen. Pro jeho vykreslení se použije aktuálně nastavená barva. Nulový bit naopak znamená, že odpovídající pixel polygonu vykreslen nebude. Pořadí bitů v mapě odpovídá vykreslování bloků polygonu po řádcích. Předpokládá se, že se mapa v obou směrech podle potřeby periodicky opakuje. Vyplňování polygonů s využitím vzoru je zapotřebí povolit voláním funkce **glEnable** s parametrem GL_POLYGON_

STIPPLE. K zakázání lze použít volání funkce **glDisable** se stejným parametrem. Použití funkce **glPolygonStipple** ukazuje následující příklad.

Příklad vyplňování polygonu

```
/* Nejprve se definuje mapa mající 32x32 bitů, tj. 2^7 bajtů */
GLubyte map[] = {
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x03, 0x80, 0x01, 0xC0, 0x06, 0xC0, 0x03, 0x60,
. . .
0x10, 0x63, 0xC6, 0x08, 0x10, 0x30, 0x0C, 0x08,
0x10, 0x18, 0x18, 0x18, 0x18, 0x18, 0x10, 0x00};
glEnable(GL_POLYGON_STIPPLE);
glPolygonStipple(map);
/* V tomto místě pak už konečně budete kreslit polygon. */
```

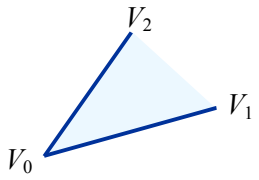
void **glEdgeFlag**(GLboolean *flag*);
void **glEdgeFlagv**(const GLboolean **flag*);

Pomocí volání funkce **glEdgeFlag** lze žádat, aby některá jednotlivá hrana plochy (trojúhelníka, čtyřúhelníka nebo polygonu) byla či nebyla zobrazena. Budete-li skládat např. nějakou složitější plochu z trojúhelníků, pak můžete chtít, aby byly zobrazeny právě jen hrany oné původní plochy a nikoli vnitřní hrany vzniklé triangulací. Volání funkce glEdgeFlag s parametrem GL_TRUE, způsobí, že všechny hrany mající počátek v následně specifikovaných vrcholech se budou kreslit. Po vyvolání funkce glEdgeFlag s parametrem GL_FALSE se naopak kreslit nebudou. Použití funkce glEdgeFlag ilustruje následující příklad a obr. 5.2.

Řízení vykreslování hran polygonu

Příklad řízení vykreslování hran

```
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
glBegin(GL_POLYGON);
   glEdgeFlag(GL_TRUE);  glVertex3fv(V0);
   glEdgeFlag(GL_FALSE); glVertex3fv(V1);
   glEdgeFlag(GL_TRUE);  glVertex3fv(V2);
glEnd();
```



Obr. 5.2. K příkladu použití funkce glEdgeFlag.

void **glNormal3**{bsidf} (TYPE *nx*, TYPE *ny*, TYPE *nz*);
void **glNormal3**{bsidf}**v** (const TYPE **v*);

Voláním funkce **glNormal3** se nastavuje normálový vektor ve vrcholech, které budou následně kresleny pomocí volání funkce glVertex. Normálový vektor je zapotřebí nastavit zejména tehdy, jestliže má být prováděn výpočet osvětlení. Příklad použití funkce glNormal3 jsme uvedli již dříve, když jsme vysvětlovali, které funkce smí být volány mezi glBegin a glEnd.

Nastavení normály ve vrcholu

S H R N U T Í

Shrnutí: V tuto chvíli byste měli vědět, jaké základní útvary lze v OpenGL kreslit a jaké vlastnosti kresby lze pro ně nastavovat. Buďte, prosím, ještě chvíli trpěliví, velmi brzy začnete v OpenGL sami také programovat.

5.3 Geometrické transformace

Doba studia asi 1,5 hod

Důležitou činností, kterou grafické systémy provádějí velmi často, jsou geometrické transformace. Jejich cílem je následující: 1) Jsou významným krokem při zobrazování scény (často se pak mluví o transformacích zobrazovacích). 2) Mohou být nápomocny i při vytváření scény (pak se často označují jako transformace modelovací). V této kapitole se dovíte, jak jsou transformace realizovány v OpenGL. Po přečtení kapitoly budete v OpenGL umět prakticky realizovat transformace zobrazovací i modelovací.

Čemu se zde naučíte?

Je obvyklé, že geometrické transformace bývají v grafických systémech realizovány jako transformace projektivní. Víme už totiž z první kapitoly tohoto textu, že projektivní transformace je nejobecnější lineární transformací (afinní transformaci lze považovat za speciální případ transformace projektivní). Také víme, že projektivní transformaci lze popsat maticovým zápisem

Všechny lineární transformace bývají obvykle realizovány jako transformace projektivní.

$$\mathbf{y} = \mathbf{M}\mathbf{x} \ , \tag{5.1}$$

kde $\mathbf{x}$ je vektor souřadnic před transformací, $\mathbf{M}$ je matice popisující transformaci a $\mathbf{y}$ je vektor souřadnic po transformaci. Protože grafické systémy pracují nejčastěji s prostorem trojrozměrným, je typické, že vektory $\mathbf{x}$, $\mathbf{y}$ jsou rozměru 4 a matice $\mathbf{M}$ je rozměru 4×4. Pokud se snad podivujete tomu, že jsme v první kapitole pro projektivní transformaci používali vztahu $\mathbf{y}=\mathbf{x}\mathbf{T}$ a nyní uvádíme vztah poněkud jiný, vězte, že jsou oba ekvivalentní a jeden lze obdržet transpozicí druhého. V první kapitole jsme vektory $\mathbf{x}$, $\mathbf{y}$ předpokládali řádkové, nyní je předpokládáme sloupcové. Mezi oběma variantami matice popisující transformaci platí jednoduchý vztah $\mathbf{M}=\mathbf{T}^T$.

Projektivní transformace ve 3D je popsána maticí rozměru 4×4.

Vše, co jsme až doposud ohledně transformací popisovali jako typické, naleznete také v OpenGL. Lze tedy shrnout: V OpenGL je realizována jak transformace zobrazovací, tak transformace modelovací. Obě se provádějí podle vztahu (5.1). V každém okamžiku jsou proto v OpenGL k dispozici dvě transformační matice. Jedna (v tomto textu ji označíme $\mathbf{M}_P$) popisuje transformaci, která bude prováděna vždy při zobrazování scény. Druhá (označíme ji $\mathbf{M}_M$) popisuje modelovací transformaci. Modelovací transformace je aplikována na všechny zadávané souřadnice. Jako příklad řekněme, že jsme při kreslení trojúhelníka jako souřadnice jeho vrcholů zadali prostřednictvím funkce glVertex vektory $\mathbf{v}_1$, $\mathbf{v}_2$, $\mathbf{v}_3$. Ve skutečnosti ale OpenGL vytvoří trojúhelník s vrcholy o souřadnicích $\mathbf{M}_M\mathbf{v}_1$, $\mathbf{M}_M\mathbf{v}_2$, $\mathbf{M}_M\mathbf{v}_3$. To proto, že na zadávané souřadnice vždy aplikuje právě platnou modelovací transformaci. Vidíme, že jak při zobrazování scény, tak při používání modelovací transformace je zásadní naplnit transformační matice $\mathbf{M}_P$, $\mathbf{M}_M$ správnými hodnotami. V podstatě právě jen tomuto problému bude věnován zbytek podkapitoly. Jsou-li totiž matice správně naplněny, pak již dál téměř není co řešit. Poznamenejme, že, přesně vzato, existuje v OpenGL ještě jedna transformace a jí odpovídající matice (nazveme ji $\mathbf{M}_T$), kterou se transformují souřadnice v textuře. Tou se ale zatím podrobněji zabývat nebudeme.

V OpenGL je transformace zobrazovací i transformace modelovací.

Zásadní: Obě matice musí být naplněny správnými hodnotami!

void **glMatrixMode**(GLenum *mode*);

Hodnoty ve všech transformačních maticích ($\mathbf{M}_M$, $\mathbf{M}_P$, $\mathbf{M}_T$) jsou nastavovány jedinou množinou funkcí OpenGL. Než programátor začne tyto funkce používat, musí sdělit pomocí volání funkce **glMatrixMode**, ke které matici se hodlá dále

Volba aktuální matice

obracet (kterou hodlá modifikovat). Budeme říkat, že programátor učiní některou z matic maticí aktuální. K aktuální matici se pak vztahují všechny následně volané funkce, které obsah transformačních matic mění. Parametr *mode* funkce glMatrixMode může nabývat hodnot GL_MODELVIEW, GL_PROJECTION nebo GL_TEXTURE, což odpovídá tomu, že se aktuální maticí stane matice $\mathbf{M}_M$, $\mathbf{M}_P$, nebo matice $\mathbf{M}_T$. Jméno první konstanty se může zdát poněkud překvapivé. Možná, že byste spíše očekávali pojmenování „GL_MODEL“. Pojmenování GL_MODELVIEW není ovšem nahodilé. Vyjadřuje, že součástí transformace modelovací se v OpenGL obvykle stává i jistá část transformace zobrazovací. Podrobnosti uvedeme později.

void **glLoadIdentity**(void);

Nastavení jednotkové matice

Voláním funkce **glLoadIdentity** se aktuální matice nastavuje na matici jednotkovou. Funkce se obvykle používá pro nastavení počáteční hodnoty transformačních matic.

void **glLoadMatrix**{fd}(const TYPE **m*);

Zapsání hodnot do aktuální matice

Funkce **glLoadMatrix** přepisuje hodnoty všech šestnácti prvků aktuální transformační matice hodnotami novými. Nové hodnoty musí být nachystány po sloupcích v poli, na které ukazuje parametr *m*. Obvykle se hodnoty připraví postupem, kterému jste se naučili v kapitole 1 tototo textu.

void **glMultMatrix**{fd}(const TYPE **m*);

Kompozice zadané matice s aktuální maticí

Funkce **glMultMatrix** násobí aktuální matici maticí, která je funkci předána jako parametr (pole, na nějž ukazuje parametr *m*, obsahuje 16 prvků této matice uspořádaných po sloupcích). Výsledek násobení je uložen do aktuální matice. Uvedený postup odpovídá skládání transformací (v kapitole 1 jsme ukázali, že skládání transformací odpovídá násobení transformačních matic a naopak). Ukázku použití dosud probraných funkcí přináší následující příklad.

```
glMatrixMode(GL_MODELVIEW); /* Aktuální bude matice M_M.     */
glLoadIdentity()            /* Nastavíme M_M = I.            */
glMultMatrix(M1);           /* Matice M1,M2 jsme, řekněme,   */
glMultMatrix(M2);           /* připravili už dříve.          */
glBegin(GL_POINTS);         /* Nyní je M_M = M1M2.           */
   glVertex3f(v);           /* Konečné souřadnice tohoto     */
...                         /* vrcholu budou M1M2v.          */
glEnd;
```

Použití následujících funkcí **glTranslate**, **glRotate** a **glScale** je typické zejména pro nastavování hodnoty matice $\mathbf{M}_M$ (použití uvedených funkcí pro nastavení zbývajících matic je sice možné, ale není příliš obvyklé). Funkce se tak zpravidla stávají součástí specifikace modelovací transformace.

void **glTranslate**{fd}(TYPE *x*, TYPE *y*, TYPE *z*);

Specifikace posunutí

Funkce **glTranslate** násobí aktuální matici maticí, která provádí posun objektu o vektor (*x*, *y*, *z*), a výsledek násobení zapisuje do aktuální matice. Jak matice realizující takový posuv vypadá, už víte z kapitoly 1 (to bude platit i pro matice realizující další transformace z této podkapitoly, což již dále nebudeme připomínat). Poznamenejme, že na uvedený posuv objektu můžete také pohlížet jako na posun souřadné soustavy o vektor (–*x*, –*y*, –*z*).

void **glRotate**{fd}(TYPE *angle*, TYPE *x*, TYPE *y*, TYPE *z*);

Specifikace rotace

Funkce **glRotate** násobí aktuální matici maticí, která provádí rotaci objektu o úhel *angle* proti směru hodinových ručiček okolo přímky spojující počátek souřadné

soustavy s bodem o souřadnicích (*x*, *y*, *z*). Hodnota parametru *angle* se zadává ve stupních. Výsledek násobení je zapsán do aktuální matice.

void **glScale**{fd}(TYPE *sx*, TYPE *sy*, TYPE *sz*);

Nastavení měřítka

Funkce **glScale** násobí aktuální matici maticí, která provádí změnu měřítka a případně zrcadlení. Hodnoty *sx*, *sy*, *sz* udávají měřítka na jednotlivých osách. Je-li měřítko záporné, jedná se o zrcadlení.

Poněkud speciální (ale užitečná) je funkce **gluLookAt**. Je typické, že funkce gluLookAt opět modifikuje matici $\mathbf{M}_M$, ačkoli souvisí se zobrazením. Je to proto, že se v OpenGL předpokládá, že se scéna pozoruje z počátku souřadné soustavy ve směru proti ose *z*. Aby byl získán požadovaný obraz scény, je proto nutné objekty během zobrazení vhodně posunout pod střed projekce a vhodně natočit. Tato část zobrazovací transformace bývá v OpenGL obvykle připojena k transformaci modelovací. K výpočtu odpovídající složky transformace lze použít funkci gluLookAt (povšimněte si, že podle předpony se v tomto případě již nejedná o základní funkci OpenGL, ale o funkci rozšiřující).

void **gluLookAt**(GLdouble *eyex*, GLdouble *eyey*, GLdouble *eyez*,
GLdouble *centerx*, GLdouble *centery*, GLdouble *centerz*,
GLdouble *upx*, GLdouble *upy*, GLdouble *upz*);

Nastavení středu projekce a ohniska pozornosti

Funkce **gluLookAt** vytvoří matici „pohledové transformace“ a násobí ji s aktuální maticí. Výsledek je uložen do aktuální matice. Poloha pozorovatele je specifikována pomocí parametrů *eyex*, *eyey*, *eyez*. Parametry *centerx*, *centery*, *centerz* definují souřadnice bodu, do něhož se pozorovatel dívá (zpravidla „střed“ scény). Parametry *upx*, *upy*, *upz* definují směr, který se v obraze bude jevit svisle.

Pro následující funkce **glFrustum**, **gluPerspective**, **glOtho**, **glOrtho2D** je naopak typické, že se používají k modifikaci matice $\mathbf{M}_P$ (jejich použití k modifikaci obsahu jiných matic je sice teoreticky možné, bylo by ale dost neobvyklé). Funkce slouží k definici středového nebo rovnoběžného promítání.

void **glFrustum**(GLdouble *left*, GLdouble *right*, GLdouble *bottom*,
GLdouble *top*, GLdouble *near*, GLdouble *far*);

Nastavení středového promítání

Funkce **glFrustum** vytváří matici perspektivní projekce a násobí touto maticí matici aktuální. Výsledek násobení je uložen opět do aktuální matice. Trojice (*left*, *bottom*, –*near*) a (*right*, *top*, –*near*) určují souřadnice levého dolního a pravého horního rohu „zobrazovacího okna“. Polopřímky vedené ze středu promítání do bodů na obvodě tohoto okna vytvářejí plášť zorného jehlanu (obr. 5.3). Parametry *near* a *far* určují vzdálenost přední a zadní ořezávací roviny od středu projekce (opět obr. 5.3). Zadávají se s kladným znaménkem. Kladná hodnota *near* specifikuje rovinu $z = -near$ (OpenGL se tak snaží zamaskovat fakt, že prakticky zajímavé hodnoty souřadnice *z* jsou obvykle všechny záporné).

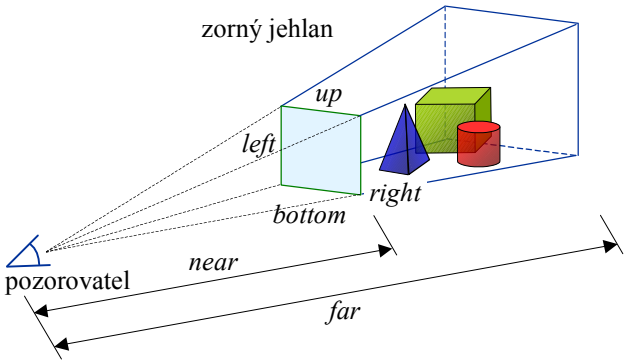
void **gluPerspective**(GLdouble *fovy*, GLdouble *aspect*,
GLdouble *near*, GLdouble *far*);

Jednodušší varianta zadání středového promítání

Funkce **gluPerspective** vytváří matici „symetrické“ perspektivní projekce a násobí touto maticí matici aktuální. Výsledek násobení je uložen opět do aktuální matice. Termínem „symetrická projekce“ se rozumí, že zorný jehlan je symetrický podle souřadnicových rovin *zx*, *yz*. Parametr *fovy* je úhel rozevření zorného jehlanu měřený v rovině *zx*. Jeho hodnota musí být v intervalu ⟨0, 180⟩ stupňů. Parametr *aspect* udává poměr šířky ku výšce zobrazovacího okna. Šířka se měří v rovině *zx*, výška v rovině *yz*. Hodnoty *near* a *far* jsou vzdálenosti mezi středem promítání a

přední, resp. zadní ořezávací rovinou. Platí pro ně totéž, co bylo uvedeno v popisu předchozí funkce.

Specifikace středového promítání



Obr. 5.3. Údaje zadávající středové promítání ve funkci glFrustum.

void **glOrtho**(GLdouble *left*, GLdouble *right*, GLdouble *bottom*, GLdouble *top*,
GLdouble *near*, GLdouble *far*);

Nastavení rovnoběžného promítání

Funkce **glOrtho** vytváří matici rovnoběžného pravoúhlého promítání a násobí touto maticí matici aktuální. Výsledek násobení je uložen do aktuální matice. Zorný objem má tvar kvádru. Průnik přední ořezávací roviny a zorného kvádru je tvořen obdélníkem, jehož jeden roh má souřadnice (*left*, *bottom*, –*near*). Souřadnice rohu protilehlého jsou (*right*, *top*, –*near*). Podobné platí i pro zadní ořezávací rovinu. Zde má dvojice rohů souřadnice (*left*, *bottom*, –*far*) a (*right*, *top*, –*far*). Parametry *near* i *far* mohou být v tomto případě kladné i záporné.

void **gluOrtho2D**(GLdouble *left*, GLdouble *right*,
GLdouble *bottom*, GLdouble *top*);

2D varianta rovnoběžného promítání

Funkce **gluOrtho2D** je jednodušší dvojrozměrnou variantou funkce předchozí. Předpokládá se, že má být zobrazeno to, co je nakresleno v rovině *xy* (proto 2D). Na základě toho systém nastaví hodnoty *near* a *far* sám na hodnoty −1.0, 1.0.

Zásobníky pro uschování transformačních matic

Pro uschovávání a obnovu matic $\mathbf{M}_M$, $\mathbf{M}_P$ a $\mathbf{M}_T$ jsou v OpenGL k dispozici zásobníky. Zásobníky jsou celkem tři, protože každá z uvedených matic „má svůj zásobník“. Možnost uschování hodnoty transformační matice je zvláště užitečná u matice $\mathbf{M}_M$, tedy u modelovací transformace. Typické je, že už, když hodnotu matice $\mathbf{M}_M$ měníte, víte, že se k její původní hodnotě budete později chtít vrátit. V takovém případě původní hodnotu matice uložíte do zásobníku. K manipulaci s maticovými zásobníky jsou k dispozici příkazy **glPushMatrix** a **glPopMatrix**.

void **glPushMatrix**(void);

Funkce **glPushMatrix** zapíše na vrchol zásobníku hodnotu aktuální transformační matice. Použije se ten zásobník, který odpovídá nastavené aktuální matici.

void **glPopMatrix**(void);

Funkce **glPopMatrix** přepíše aktuální transformační matici maticí přečtenou z vrcholu zásobníku odpovídajícího nastavené aktuální matici. Přečtená matice je ze zásobníku odstraněna.

Protože se v praktických příkladech s využitím maticových zásobníků často setkáte, uvedeme již nyní jednoduchý příklad. Řekněme, že kreslíte nějaké objekty. Kresba se provádí ve třech částech programu, které nazveme Kresba1, Kresba2, Kresba3. Řekněme, že pro popis objektů v části Kresba2 je výhodné změnit souřadnou soustavu. Provedete tedy odpovídající modelovací transformaci. Řekněme dále, že nová souřadná soustava již ale není výhodná pro popis objektů kreslených v části Kresba3, a proto se před kreslením objektů v této části vrátíte k souřadné soustavě původní. Popsaná část programu může vypadat např. takto:

Příklad použití zásobníku

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
Kresba1;
glPushMatrix();
glRotatef(30.0, 0.0, 0.0, 1.0);
glTranslatef(3.0, 1.0, 0.0);
Kresba2;
glPopMatrix();
Kresba3;
```

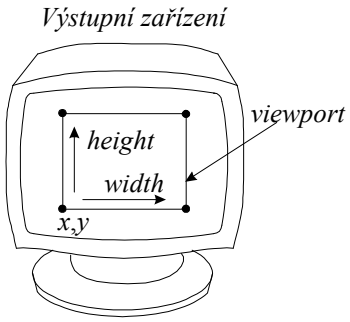
Na konec kapitoly o transformacích jsme ponechali transformaci na výstupní zařízení. Transformace provádí převod souřadnic z normalizovaného zobrazovacího objemu na celočíselné souřadnice obrazových bodů. Transformace se z rámce transformací dosud popsaných vymyká tím, že nemodifikuje žádnou z dříve zmiňovaných matic. S ohledem na svoji jednoduchost se totiž neprovádí maticovým výpočtem. Teorii transformace na výstupní zařízení již dobře znáte z podkapitoly 1.8. Transformace je zde dokonce jednodušší, než jsme ji dříve probrali my. OpenGL totiž neumožňuje zadání výřezu z normalizovaného zorného objemu, ale vždy zobrazuje objem celý. Zadání transformace se tak redukuje na pouhé zadání oblasti, do níž má být kreslení provedeno. To se provede pomocí funkce **glViewport**.

Transformace na výstupní zařízení

void **glViewport**(GLint *x*, GLint *y*, GLsizei *width*, Glsizei *height*);

Volání funkce **glViewport** definuje obdélníkovou oblast (viewport), do níž bude výstupní obraz vykreslen. Parametry *x* a *y* určují levý dolní roh oblasti, *width* a *height* jsou její rozměry (obr. 5.4). Implicitně je $x = 0$, $y = 0$, *width* a *height* jsou rozměry celého okna.

Nastavení oblasti pro kresbu



Obr. 5.4. Nastavení velikosti viewportu.

S H R N U T Í

Shrnutí: V tuto chvíli byste měli vcelku podrobně vědět, jak OpenGL provádí modelovací a zobrazovací transformace. Probrali jsme téměř všechny funkce, které OpenGL v tomto ohledu nabízí.

5.4 První program v OpenGL

Doba studia asi 1 hod

Nyní již konečně máte dostatek informací k tomu, abyste mohli sestavit váš první program v OpenGL. Najdete jej v souboru „první_program.cpp“ na přiloženém CD. Protože k němu bude zapotřebí učinit několik poznámek, následuje jeho výpis. Jedná se o program vemi jednoduchý. Nakreslí pouze dvě úsečky.

Čemu se zde naučíte?

První program v OpenGL

```
/*============================================================*/
#include <stdio.h>
#include <gl\glaux.h>
/*------------------------------------------------------------*/
/* Funkce my_display se volá, když se má překreslit obsah okna.   */
void my_display(){
  GLfloat a[3]={ 0.0, 0.0, 0.0};     /* vektor souřadnic          */
  glClearColor(0.0, 0.0, 0.0, 1.0);  /* barva mazání              */
  glClear(GL_COLOR_BUFFER_BIT);      /* Smaže obsah okna.         */
  glBegin(GL_LINES);                 /* začátek kreslení úseček   */
    glColor3f(0.0,1.0,0.0);          /* první barva kreslení      */
    glVertex3fv(a);                  /* počáteční bod             */
    glVertex3f(-9.0, 0.0, 0.0);      /* koncový bod               */
    glColor3f(1.0, 0.0, 1.0);        /* druhá barva kreslení      */
    glVertex3fv(a);                  /* počáteční bod             */
    glVertex3f(9.0, 9.0, 0.0);       /* koncový bod               */
  glEnd();                           /* konec kreslení            */
  glFlush();                         /* vyprázdnění bufferů       */
}
/*------------------------------------------------------------*/
/* Funkce my_reshape se volá, když se změní rozměry okna.         */
/* V parametrech w a h jí systém předá nové rozměry okna.         */
void my_reshape(GLsizei w, GLsizei h){
  glViewport(0,0,w,h);               /* Nastavíme oblast kresby.  */
  glMatrixMode(GL_PROJECTION);       /* Matici projekce nastavíme */
  glLoadIdentity();                  /* na jednotkovou matici.    */
  if (w<=h)  gluOrtho2D( -10.0, 10.0, -10.0*(GLfloat)h/(GLfloat)w,
                         10.0*(GLfloat)h/(GLfloat)w);
      else   gluOrtho2D( -10.0*(GLfloat)w/(GLfloat)h,
                         10.0*(GLfloat)w/(GLfloat)h, -10.0, 10.0);
  glMatrixMode(GL_MODELVIEW);        /* Aktivní bude matice model- */
  glLoadIdentity();                  /* view. Bude jednotková.    */
}
/*------------------------------------------------------------*/
int main(void){
  auxInitDisplayMode(AUX_SINGLE | AUX_RGBA);
     /* jednoduché bufferování, RGBA barevný model              */
  auxInitPosition(10,10,300,300);
     /* specifikace poč. umístění a velikosti okna pro kresbu   */
  auxInitWindow("Okno kresby");
     /* vytvoření okna podle předchozích parametrů              */
  auxReshapeFunc((void(_stdcall*)(long,long))my_reshape);
     /* stanovení, která funkce se volá při změně velikosti okna */
  auxMainLoop((void(_stdcall*)(void))my_display);
     /* stanovení, která funkce zajišťuje překreslení okna      */
  return 0;
}
/*============================================================*/
```

Přestože je program vcelku bohatě komentován, považujeme za potřebné objasnit některé záležitosti podrobněji, protože ne vše, co je v programu použito, jsme zatím probrali.

Funkcí, které jsme zatím neprobrali, naleznete nejvíce v hlavním programu. Funkce **auxInitDisplayMode** specifikuje podrobnosti o paměti obrazu, kterou v programu budete používat. Konstanta AUX_SINGLE říká, že paměť má být jediná. Do ní bude OpenGL nově vznikající obraz zapisovat a současně bude obsah paměti také zobrazovat. Pokud budete mít složitější obraz a pomalejší počítač, může se stát, že budete na obrazovce vidět, jak se do obrazu postupně kreslí jednotlivé grafické objekty. To je někdy nežádoucí. V takovém případě lze pak místo konstanty AUX_SINGLE použít konstanty AUX_DOUBLE. Ta říká, že si pro obraz přejete dvě paměti. Jednu paměť, v níž je již hotový obraz, OpenGL zobrazuje. Nový obraz vytváří do druhé paměti, takže postup vykreslování není vidět. Nový obraz OpenGL zobrazí teprve tehdy, až je celý hotov. V programátorském žargónu se pro popsané varianty používá termínů jednoduché nebo dvojité „bufferování“. Konstanta AUX_RGBA říká, že si barvy přejeme zadávat v *r*, *g*, *b* složkách a že si také přejeme zřízení tzv. alfa kanálu.

Specifikace počtu obrazových pamětí

Funkce **auxInitPosition** slouží ke specifikaci počátečního umístění a počátečních rozměrů okna pro kresbu (po vytvoření lze ale okno přesouvat a měnit jeho rozměry). V programu jsme požadovali umístění levého horního rohu okna na pozici 10, 10 pixelů. Počáteční šířka i výška okna jsou obě 300 pixelů. Funkce auxInitPosition sama ale okno kresby nevytvoří. To provede teprve až následující funkce **auxInitWindow**. Jediným parametrem funkce je nápis, který se má objevit v horní liště okna.

Vytvoření okna pro kresbu

Programy v OpenGL jsou řízené událostmi. Systému je zapotřebí sdělit, které funkce mají být volány, když některá z možných událostí nastane. V našem programu předpokládáme jen dvě události. 1) Systém rozpozná, že obsah okna je zapotřebí překreslit (např. proto, že bylo celé nebo zčásti skryto a nyní má být opět vidět). 2) Sytém rozpozná, že jste změnili rozměry okna, a proto je kresbu v okně zapotřebí aktualizovat. Funkce **auxMainLoop** definuje, která funkce bude vyvolána při požadavku na překreslení okna a jaké má parametry. V našem případě má být volána funkce my_display, která je bez parametrů. Voláním funkce **auxReshapeFunc** podobně definujeme, že při změně velikosti okna má být v našem případě volána funkce my_reshape, která má dva parametry typu „long“. Pro úplnost uveďme, že při změně velikosti okna se volá také i funkce pro překreslení okna. V našem případě by se tedy nejprve vyvolala funkce my_reshape a pak ještě i funkce my_display.

Specifikace funkcí volaných při požadavku na překreslení a změnu velikosti okna kresby

Činnosti funkce my_display byste měli již vcelku dobře rozumět. Neprobrali jsme jen funkci **glClearColor**, která nastavuje barvu, kterou je pak následně funkcí **glClear** vymazána paměť obrazu. V našem případě paměť obrazu mažeme černou barvou ($r = g = b = 0$). Hodnotu alfa kanálu nastavujeme na 1. Alfa kanál, popravdě řečeno, v našem úvodním programu ani nepotřebujeme. Zavedli jsme jej jen proto, abyste si na tento užitečný a současně jednoduchý nástroj postupně zvykali. Volání funkce **glFlush** způsobí, že se všechny příkazy, které jste až dosud v OpenGL vydali, provedou a že tedy v okně pro kresbu uvidíte odpovídající obraz. Pokud funkci nevyvoláte, může se stát, že příkazy budou čekat na „vhodnější“ (z hlediska systému) dobu pro své provedení v nejrůznějších frontách. Zbytku funkce my_display už určitě rozumíte a shledáváte, že náš úvodní program nakreslí jen dvě úsečky.

Co dělá naše funkce my_display?

Funkci my_reshape, která je, jak již víme, volána při změně velikosti okna kresby, jsou systémem předány jako parametry dvě hodnoty *w*, *h*, což je nová

hodnota šířky a výšky okna. Voláním funkce **glViewport** nastavujeme velikost oblasti, do níž se bude kreslit, na celý nový rozměr okna. Voláním funkce **glMatrixMode** s parametrem GL_PROJECTION říkáme, že se v dalším budeme obracet k matici specifikující projekci. Funkce **glLoadIdentity** matici projekce nastaví na matici jednotkovou. K této, zatím triviální projekční transformaci, je dále připojena transformace specifikovaná funkcí **gluOrtho2D**. Konstrukce, která je při volání uvedené funkce v našem programu použita, konkrétně zajišťuje, abychom ve směru osy x i y viděli vždy alespoň rozsah $\langle -10,10\rangle$ a aby měřítko na obou osách bylo stejné. V závěru funkce my_reshape činíme aktuální maticí matici modelview ($\mathbf{M}_M$) a zapisujeme do ní matici jednotkovou. Tím se chystáme na kreslení, které bude následovat, až bude vyvolána funkce my_display.

Co dělá naše funkce my_reshape?

Úkol 5.2. Přeložte a sestavte program probraný v této podkapitole. Program se pro vás stane východiskem ke tvorbě dalších složitějších programů. Můžete se také pokusit o nějaké své vlastní modifikace programu.

Úkol 5.2

Úkol 5.3. Vytvořte program, v němž nakreslíte po jednom od každého základního útvaru, který lze v OpenGL nakreslit. Kreslete v rovině. Vzorové řešení naleznete v souboru „úkol5_3.cpp“ na přiloženém CD.

Úkol 5.3

Úkol 5.4. Vytvořte program, který pomocí pásů čtyřúhelníků zobrazí průběh funkce $\cos(2\pi ax)\cos(2\pi by)$. Funkci zobrazte ve vhodném středovém promítání s řešením viditelnosti. Vzorové řešení naleznete v souboru „úkol5_4.cpp“ na přiloženém CD. Uvidíte zde mimo jiné, jak se prakticky nastaví zobrazení trojrozměrného objektu.

Úkol 5.4

5.5 Zobrazovací seznamy

Doba studia asi 1 hod

Zobrazovací seznam (display list) je skupina příkazů OpenGL, které jsou připraveny k pozdějšímu použití (vykonání). Zobrazovací seznam lze přirovnat k proceduře v programovacích jazycích (nejprve je nachystáte a pak je voláte). Ve fázi přípravy (kompilace) OpenGL připraví zobrazovací seznam do podoby, v níž ho lze efektivně provádět. Když je pak zobrazovací seznam vyvolán, jsou postupně prováděny jednotlivé jeho příkazy v pořadí, jak byly v seznamu uvedeny. Je dovoleno vnořování zobrazovacích seznamů. V této podkapitole se zobrazovací seznamy naučíte používat.

Čemu se zde naučíte?

K čemu jsou zobrazovací seznamy dobré?

Zobrazovací seznamy jsou identifikovány celými kladnými čísly. Definice seznamu se provádí tak, že se posloupnost požadovaných příkazů, které mají seznam tvořit, uzavře dvojicí volání **glNewList** a **glEndList**. Ne všechny funkce OpenGL však mohou být při deklaraci zobrazovacího seznamu použity. Obecně lze říci, že se při vytváření seznamů nesmí použít funkce, kterým se předávají parametry adresou nebo které vrací nějaké funkční hodnoty. Po svém vytvoření se zobrazovací seznam volá příkazem **glCallList**. Pokud je seznam jednou vytvořen, pak již nemůže být modifikován (může být ale zrušen). Dále uvádíme podrobnější specifikaci trojice dosud jmenovaných funkcí.

Jak se seznamy vytváří a volají?

void **glNewList**(GLuint *list*, GLenum *mode*);

Voláním funkce **glNewList** se specifikuje začátek nového zobrazovacího seznamu. Parametr *list* je jedinečné celé kladné číslo, které zobrazovací seznam identifikuje.

Vytvoření seznamu

Možné hodnoty parametru *mode* jsou GL_COMPILE_AND_EXECUTE (zobrazovací seznam se připraví pro pozdější použití a ihned se také jednou provede) a nebo GL_COMPILE (jen se připraví, ale neprovede).

void **glEndList**(void);

Volání funkce **glEndList** ukončuje dříve zahájený zobrazovací seznam.

void **glCallList**(GLuint *list*);

Volání zobrazovacího seznamu

Voláním funkce **glCallList** se provede zobrazovací seznam specifikovaný parametrem *list*.

Vnořování seznamů

Volání zobrazovacích seznamů je možné vnořovat. Volání funkce glCallList je tedy možné používat i uvnitř definice nového zobrazovacího seznamu. Není ovšem povoleno volání rekurzivní. Maximální hloubka vnoření smí být přinejmenším 64 (to znamená, že každá implementace OpenGL musí zajistit alespoň tuto hodnotu, může se ale stát, že je v konkrétní implementaci dovolena i hodnota vyšší). Následující příklad ukazuje definici a použití zobrazovacího seznamu.

Příklad vytvoření a použití zobrazovacího seznamu

```
glNewList(1, GL_COMPILE);/* Tuto posloupnost příkazů   */
  glBegin(GL_TRIANGLES); /* musíte ve svém programu    */
    glVertex3fv(v1);     /* provést dříve, než seznam  */
    glVertex3fv(v2);     /* číslo 1 začnete používat.  */
    glVertex3fv(v3);
  glEnd();
glEndList();

void my_display(){
  …
  glCallList(1);
  glTranslate(5.0f, 0.0f, 0.0f);
  glCallList(1);
  …
}
```

Správa čísel seznamů

Identifikace zobrazovacích seznamů pomocí čísel staví před programátora jisté problémy. V rozsáhlejších programech si programátor zejména nemusí být jist, která čísla už byla jako identifikátory zobrazovacích seznamů použita a která ještě zůstávají volná. V takovém případě mohou pomoci následující dvě funkce **glIsList** a **glGenLists**, sloužící pro správu čísel zobrazovacích seznamů.

GLboolean **glIsList**(GLuint *list*);

Funkce **glIsList** vrací hodnotu TRUE, jestliže číslo specifikované argumentem *list* je již použito jako identifikátor zobrazovacího seznamu. Jinak vrací FALSE.

GLuint **glGenLists**(GLsizei *range*);

Funkci **glGenLists** programátor použije v případě, že chce zřídit větší počet zobrazovacích seznamů tak, že jejich identifikační čísla jdou po sobě. Parametr *range* určuje počet zřizovaných seznamů. Funkce zjistí, kde se v prostoru přirozených čísel nachází souvislý blok (velikosti *range*) hodnot, které jako identifikátory zobrazovacích seznamů zatím nebyly použity. Jako funkční hodnota se vrací hodnota prvního (nejmenšího) čísla z nalezeného bloku. Pokud by snad blok požadované délky nebyl nalezen, vrací se hodnota 0.

Zobrazovací seznamy, které již nejsou v programu potřebné, lze vypustit pomocí funkce **glDeleteLists**.

void **glDeleteLists**(GLuint *list*, GLsizei *range*);

Funkce **glDeleteLists** ruší skupinu zobrazovacích seznamů. Identifikátor prvního rušeného seznamu je dán parametrem *list*. Parametr *range* je počet rušených zobrazovacích seznamů (ruší se seznamy po sobě jdoucí, počínaje seznamem *list*).

Rušení seznamů

OpenGL poskytuje účinný postup pro vykonání několika zobrazovacích seznamů „v sérii“. K tomu je zapotřebí uložit identifikátory jednotlivých zobrazovacích seznamů, které mají být vykonány, do pole. Vykonání se pak provede pomocí funkce **glCallLists**. S funkcí glCallList může spolupracovat funkce **glListBase**.

void **glCallLists**(GLsizei *n*, GLenum *type*, const GLvoid **lists*);

Funkce vykonává *n* zobrazovacích seznamů. Identifikátory seznamů, které budou vykonány, jsou dány součtem báze specifikované pomocí následující funkce **glListBase** a hodnot uložených v poli, na které ukazuje parametr *lists*. Argument *type* udává datový typ prvků pole *lists*. Dovolena je jedna z následujících konstant: GL_BYTE, GL_UNSIGNED_BYTE, GL_SHORT, GL_UNSIGNED_SHORT, GL_INT, GL_UNSIGNED_INT, GL_FLOAT nebo GL_2_BYTES, GL_3_BYTES, GL_4_BYTES.

Hromadné volání seznamů

void **glListBase**(GLuint *base*);

Funkce **glListBase** určuje hodnotu (bázi), která je přičtena k hodnotám všech identifikátorů zobrazovacích seznamů při jejich hromadném provádění pomocí funkce glCallLists. Přednastavená hodnota báze je 0. Na funkce glCallList a glNewList nemá hodnota báze žádný vliv.

Hromadné provádění zobrazovacích seznamů je obvyklé např. při psaní textů. Následující příklad naznačuje definici vektorového fontu. Každé písmenko je připraveno jako jeden zobrazovací seznam. Pro font alokujeme celkem 128 identifikátorů zobrazovacích seznamů (proto volání glGenLists(128)). Při vykreslování textu se použije hromadného vykonání zobrazovacích seznamů. Jako identifikátory seznamů, které mají být provedeny, se použijí ordinální hodnoty znaků řetězce + hodnota báze přidělená systémem pro zobrazovací seznamy fontu.

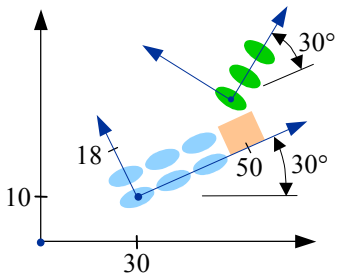
```
GLuint font1_base;
/* Následující funkce definuje použitý font. */
void initFont(void){
  font1_base = glGenLists(128);
  /* Následuje zobrazovací seznam pro písmeno A. */
  glNewList(font1_base+'A',GL_COMPILE);
    glBegin(GL_LINE_STRIP);
      glVertex2f(0.0, 0.0);
      glVertex2f(0.4, 1.0);
      glVertex2f(0.8, 0.0);
    glEnd();
    glBegin(GL_LINES);
      glVertex2f(0.133, 0.333);
      glVertex2f(0.533, 0.333);
    glEnd();
    glTranslate(1.0, 0.0, 0.0);
  glEndList();
```

Příklad definice vektorového fontu a psaní textu

```
  /* Tady se doplní zobrazovací seznamy pro ostatní znaky. */
}
/* A takto jednoduše lze pak později vypsat řetězec s. */
glListBase(font1_base);
glCallLists(strlen(s), GL_BYTE, s);
```

Úkol 5.5

Úkol 5.5. Vytvořte zobrazovací seznam kreslící plný kruh jednotkového poloměru se středem v počátku souřadné soustavy. Vytvořte program, který pomocí zobrazovacího seznamu a pomocí modelovacích transformací nakreslí dvě pole elips dle obrázku. Pak nakreslí i čtverec. Rozměry, které nejsou vyznačeny, vhodně zvolte. Nakonec nechť program také vypíše nějaký text. K tomu využijte nástrojů připravených v souboru text.h. Vzorové řešení naleznete v souboru „úkol5_5.cpp“ na přiloženém CD.



□

SHRNUTÍ

Shrnutí: Po předchozím výkladu a po úspěšném provedení úkolu 5.5 by nyní vše podstatné o zobrazovacích seznamech mělo být jasné.

5.6 Definice osvětlení a materiálů

Doba studia asi 1,5 hod

Chcete-li v OpenGL zobrazovat trojrozměrné scény, musíte provést následující: 1) Vytvořit a aktivovat jeden nebo více světelných zdrojů. 2) Určit některé další podrobnosti osvětlování (OpenGL to nazývá specifikací osvětlovacího modelu). 3) Definovat materiálové vlastnosti objektů scény. 4) Při vytváření objektů scény nesmíte zapomenout na stanovení normál ve vrcholech plošek, které povrchy objektů aproximují (z kapitoly 2 víte, že normály jsou potřebné pro výpočet úhlů dopadu a odrazu paprsků od světelného zdroje). Všechny uvedené činnosti v této kapitole podrobně popíšeme.

Jak postupvat při zobrazování 3D scén?

Čemu se zde naučíte?

Začneme definicí světelných zdrojů. Každá implementace OpenGL musí umožňovat použití alespoň osmi světelných zdrojů. Zdroje jsou identifikovány konstantami GL_LIGHT0, GL_LIGHT1, … , GL_LIGHT7. K nastavení vlastností světelného zdroje slouží v OpenGL funkce **glLight**.

void **glLight**{if}[v](GLenum *light*, GLenum *pname*, TYPE *param*);

Nastavení vlastností světelného zdroje

Parametr *light* je identifikátor světelného zdroje (GL_LIGHT0, … , GL_LIGHT7). Vlastnosti zvoleného zdroje se nastavují tak, že se parametrem *pname* určí, která vlastnost má být nastavena, a parametrem *param* se zadává její hodnota. Přehled

vlastností, které lze nastavovat, uvádí tabulka 5.4. Nevektorová varianta příkazu smí být použita pouze při nastavování „jednohodnotových“ vlastností.

Co všechno lze pro světelný zdroj nastavit?

Tabulka 5.4. Vlastnosti světelných zdrojů, které lze nastavit funkcí glLight.

Hodnota parametru *pname*	Přednastavená hodnota	Co se nastavuje
GL_AMBIENT	(0.0, 0.0, 0.0, 1.0)	intenzita rozptýlené složky světla (R, G, B, A)
GL_DIFFUSE	(1.0, 1.0, 1.0, 1.0)	intenzita difúzní složky světla (R, G, B, A)
GL_SPECULAR	(1.0, 1.0, 1.0, 1.0)	intenzita zrcadlové složky světla (R, G, B, A)
GL_POSITION	(0.0, 0.0, 1.0, 0.0)	poloha světla (x, y, z, w)
GL_SPOT_DIRECTION	(0.0, 0.0, -1.0)	směr osy reflektoru (x,y,z)
GL_SPOT_EXPONENT	0.0	exponent rozložení intenzity světla uvnitř kuželu reflektoru
GL_SPOT_CUTOFF	180	úhel rozevření kuželu reflektoru
GL_CONSTANT_ATTENUATION	1.0	konstantní koeficient útlumu
GL_LINEAR_ATTENUATION	0.0	lineární koeficient útlumu
GL_QUADRATIC_ATTENUATION	0.0	kvadratický koeficient útlumu

Předvolené hodnoty vlastností uvedené v předchozí tabulce platí pouze pro světelný zdroj 0. U zbývajících zdrojů musíte všechny vlastnosti nastavit sami. Ukázku použití funkce glLight uvádíme v následujícím příkladu.

Příklad nastavení vlastností zdroje

```
GLfloat ambient[]  = (0.1, 0.1, 0.1, 1.0);
GLfloat diffuse[]  = (1.0, 1.0, 0.0, 1.0);
GLfloat specular[] = (1.0, 1.0, 0.0, 1.0);
GLfloat position[] = (5.0, 5.0, 5.0, 1.0);

glLightfv(GL_LIGHT0, GL_AMBIENT,  ambient);
glLightfv(GL_LIGHT0, GL_DIFFUSE,  diffuse);
glLightfv(GL_LIGHT0, GL_SPECULAR, specular);
glLightfv(GL_LIGHT0, GL_POSITION, position);
```

Útlum intenzity v závislosti na vzdálenosti

Jednou z vlastností světelných zdrojů, kterou lze pomocí funkce glLight také definovat, je útlum světla v závislosti na vzdálenosti od zdroje. Předpokládá se útlum popsaný vztahem

$$\frac{1}{k_c + k_l d + k_q d^2},$$

kde d je vzdálenost mezi světelným zdrojem a bodem, v němž se osvětlení počítá. Uvedený vztah pro útlum intenzity zdroje již znáte z kapitoly 2.5. Parametry k_c, k_l, k_q, které útlum popisují, lze nastavit pomocí funkce glLight. Parametr *pname* při tom nabývá hodnot GL_CONSTANT_ATTENUATION, GL_LINEAR_ATTENUATION, GL_QUADRATIC_ATTENUATION (tabulka 5.4).

Vytvoření reflektoru

Kromě bodového zdroje je možné pomocí funkce glLight jednoduše vytvořit i světelný zdroj typu reflektor (paprsky vyzařované reflektorem tvoří kužel). Stačí k tomu definovat směr osy reflektoru (GL_SPOT_DIRECTION), a úhel rozevření kužele paprsků (GL_SPOT_CUTOFF). Tento úhel musí být z intevalu ⟨0,

90⟩ stupňů (jedná se o úhel, který jsme v kapitole 2.5 nazývali γ_{mez}). Jedinou výjimkou je přednastavená hodnota 180° platící pro bodový zdroj světla (bodový zdroj vyzařuje do všech směrů). U reflektorů lze definovat i rozložení světla uvnitř kužele. Největší intenzitu má paprsek vyzařující v ose kužele. Směrem od osy kužele k jeho plášti se intenzita světla snižuje. Snížení je úměrné kosinu úhlu sevřeného osou kužele a přímkou vedenou z místa reflektoru k osvětlovanému bodu. Kosinus je umocněn na hodnotu zadanou pomocí GL_SPOT_EXPONENT. Tuto techniku realizace reflektoru již ale opět znáte z kapitoly 2.5.

Osvětlování i světelné zdroje musíte povolit!

Při vytváření programů musíte mít na paměti, že v OpenGL musíte osvětlování explicitně povolit voláním funkce **glEnable** s parametrem GL_LIGHTING. Kromě toho musíte povolit i jednotlivé světelné zdroje. K tomu opět použijete funkci glEnable, a to s parametrem GL_LIGHT0 až GL_LIGHT7. Je-li to potřebné, můžete jednotlivé světelné zdroje i osvětlování vůbec naopak zakázat voláním funkce **glDisable** s týmiž parametry.

Specifikací osvětlovacího modelu se v OpenGL rozumí, že lze provést následující: 1) Stanovit intenzitu okolního rozptýleného (ambientního) světla ve scéně. 2) Říci, zda lze umístění pozorovatele (středu projekce) považovat za blízké nebo vzdálené od objektů scény (při vzdálené poloze je možné výpočet osvětlení zjednodušit a také urychlit). 3) Určit, zda se mají výpočty osvětlení provádět pro obě strany stěn objektů. K zadání uvedených údajů slouží funkce **glLightModel**.

void **glLightModel**{if}[v](GLenum *pname*, TYPE *param*);

Nastavení rozptýleného světla ve scéně

Parametr *pname* udává název údaje, který se má nastavit, a parametr *param* jeho požadovanou hodnotu. Parametr *pname* může nabývat hodnoty buď GL_LIGHT_MODEL_AMBIENT nebo GL_LIGHT_MODEL_LOCAL_VIEWER a nebo GL_LIGHT_MODEL_TWO_SIDE. Hodnoty GL_LIGHT_MODEL_AMBIENT se použije pro nastavení rozptýleného světla ve scéně. Použití ukazuje příklad:

```
GLfloat ambient[] = {0.1, 0.1, 0.2, 1.0};
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, ambient);
```

Povolení / zákaz zjednodušování výpočtu osvětlení

Specifikace polohy pozorovatele vzhledem k objektům scény má vliv na výpočet zrcadlové složky osvětlení (kapitola 2.5, vztah (2.2)). Pokud je pozorovatel velmi vzdálen (teoreticky v nekonečnu), potom může být výpočet úhlu α ve vztahu (2.2) jednodušší, protože směr k pozorovateli zůstává pro všechny body scény tentýž. Následující příklad ukazuje specifikaci, kdy pozorovatele za velmi vzdáleného nepovažujeme a požadujeme tedy přesnější a o něco málo zdlouhavější výpočet (přednastavená je varianta GL_FALSE).

```
glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE);
```

Výpočet pro odvrácené strany ploch

Pokud předpokládáte, že ve vytvářeném obraze mohou být vidět také odvrácené strany polygonů (stane se to např. tehdy, když část tělesa odříznete a vidíte dovnitř), musíte na to OpenGL explicitně upozornit takto:

```
glLightModeli(LIGHT_MODEL_TWO_SIDE, GL_TRUE);
```

OpenGL pak při výpočtu osvětlení odvrácených stran polygonů převrací normálu, aby i na odvrácených stranách byly úhly potřebné k výpočtu osvětlení spočítány správně.

Materiálové vlastnosti povrchů objektů scény se definují voláním funkce glMaterial. Nastavené vlastnosti platí pro všechny následně vytvářené objekty tak dlouho, dokud nejsou změněny opětovným voláním funkce glMaterial.

void **glMaterial**{if}[v](GLenum *face*, GLenum *pname*, TYPE *param*);

Nastavení materiálových vlastností

Funkce **glMaterial** specifikuje vlastnosti materiálu, které jsou použity při výpočtu osvětlení všech následně vytvářených objektů. Parametr *face* může mít hodnotu GL_FRONT, GL_BACK nebo GL_FRONT_AND_BACK. Určuje, na jakou stranu objektu mají být právě zadávané materiálové vlastnosti aplikovány. Jednotlivé zadávané materiálové vlastnosti jsou identifikovány parametrem *pname* a jejich požadované hodnoty jsou předány parametrem *param* (jedná se buď o jednu hodnotu a nebo o ukazatel na pole hodnot). Seznam všech materiálových vlastností, jejichž hodnoty lze zadávat, je uveden v tabulce 5.5.

Tabulka 5.5. Materiálové vlastnosti, které lze nastavit funkcí glMaterial.

Co všechno lze pro materiál nastavit?

Hodnota parametru *pname*	Přednastavená hodnota	Co se nastavuje
GL_AMBIENT	(0.2, 0.2, 0.2, 1.0)	koeficienty odrazu pro rozptýlené (ambientní) světlo
GL_DIFFUSE	(0.8, 0.8, 0.8, 1.0)	koeficienty difúzního odrazu
GL_SPECULAR	(0.0, 0.0, 0.0, 1.0)	koeficienty zrcadlového odrazu
GL_AMBIENT_AND_DIFFUSE		jako GL_AMBIENT a GL_DIFFUSE současně
GL_SHININESS	0.0	exponent n ve členu $\cos^n(\alpha_i)$ ze vztahu (2.2)
GL_EMISSION	(0.0, 0.0, 0.0, 1.0)	intenzita vlastního vyzařování

Typickou ukázku použití funkce glMaterial k definici materiálu platného pro přední i zadní strany ploch ukazuje následující příklad.

Příklad nastavení materiálových vlastností

```
GLfloat ambient[]  = {0.1, 0.1, 0.1, 1.0};
GLfloat diffuse[]  = {0.6, 0.6, 0.5, 1.0};
GLfloat specular[] = {0.8, 0.8, 0.7, 1.0};
GLfloat shininess  = 30.0;
glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, ambient);
glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, diffuse);
glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, specular);
glMaterialf (GL_FRONT_AND_BACK, GL_SHININESS, shininess);
```

Úkol 5.6

Úkol 5.6. Sestavte program, který ve zvoleném středovém promítání zobrazí scénu obsahující několik těles. Povrchy těles můžete vytvořit ze základních útvarů dostupných v OpenGL. Můžete také využít předdefinovaných povrchů z knihovny glaux (jejich přehled naleznete ke konci souboru glaux.h, prohlédněte si jej). K povrchům zadejte materiálové vlastnosti. Ve scéně také definujte světelné zdroje. Scénu zobrazte Gouraudovým stínováním (**glShadeModel**(GL_SMOOTH)). Vzorové řešení naleznete v souboru „úkol5_6.cpp“ na přiloženém CD. Vzorové řešení si nejprve prostudujte. Pak se pokuste o vlastní experimenty tím, že jej budete modifikovat.

Úkol 5.7

Úkol 5.7. Sestavte program realizující animaci znázorňující přiblížení se ke scéně z předchozího příkladu z dálky. Vzorové řešení naleznete v souboru „úkol5_7.cpp“ na přiloženém CD (věnujte pozornost funkci **auxIdleFunc**).

S H R N U T Í

Shrnutí: Nyní byste už měli být velmi dobře schopni vytvářet programy zobrazující osvětlené scény obsahující objekty, u nichž umíte zadat materiálové vlastnosti jejich povrchů.

5.7 Kreslení rastrových objektů

Doba studia asi 0,5 hod

Čemu se zde naučíte?

Zatím jsme se zabývali kreslením „vektorových“ geometrických objektů, jako jsou čáry, trojúhelníky, polygony atd. Pomocí OpenGL lze však zobrazovat také objekty rastrové, jako jsou např. bitové mapy a rastrové obrazy. V této podkapitole se dovíte, jak se to dělá.

Rozdíl mezi bitovou mapou a rastrovým obrazem spočívá v tom, že bitová mapa má pro každý bod obrazu pouze jeden informační bit, zatímco rastrový obraz má pro každý obrazový bod informací více (např. barevné složky *r*, *g*, *b*, *a*). Při zobrazování bitových map i rastrových obrazů musíte určit, kam se ve vytvářeném obraze mají tyto objekty umístit. K zadání polohy slouží funkce **glRasterPos**. Nastavená poloha je platná pro následně prováděnou kresbu.

void **glRasterPos**{234}{sifd}[v](TYPE *x*, TYPE *y*, TYPE *z*, TYPE *w*);

Nastavení polohy pro kresbu

Funkce **glRasterPos** nastavuje polohu pro kreslení bitových map a rastrových obrazů. Pro stručnost budeme dále jednoduše říkat, že funkce *nastavuje pozici v rastru* (na mysli při tom máme pozici ve vytvářeném výstupním obraze). Parametry *x*, *y*, *z*, *w* určují souřadnice polohy, a to v právě platné souřadné soustavě scény (tedy nikoli v obrazových bodech). Zadané souřadnice jsou podrobeny modelovací a zobrazovací transformaci a teprve po ní a po zaokrouhlení udávají novou aktuální pozici v rastru. Pokud nejsou hodnoty *z* a *w* zadány výslovně, pak se předpokládá, že je $z = 0$ a $w = 1$. Je-li naopak zapotřebí zjistit, jaká pozice v rastru je právě nastavena, lze k tomu použít funkce **glGetFloatv**. Jejím prvním parametrem bude konstanta GL_CURRENT_RASTER_POSITION, druhým parametrem bude ukazatel na pole, kam funkce zapíše zjištěné souřadnice *x*, *y*, *z*, *w* právě platné pozice.

void **glBitmap**(GLsizei *width*, GLsizei *height*, GLfloat x_{b0}, GLfloat y_{b0},
GLfloat x_{bi}, GLfloat y_{bi}, const GLubyte **bitmap*);

Kreslení bitové mapy

Funkce **glBitmap** vykreslí bitovou mapu specifikovanou v poli, na které odkazuje ukazatel *bitmap*. Kreslení se provádí právě platnou barvou. Jednotlivé bity v poli *bitmap* určují, zda se obrazový bod kreslí nebo ne. Parametry *width* a *height* určují šířku a výšku bitové mapy v obrazových bodech. Hodnota parametru *width* musí být násobkem osmi. Jednotlivé bity v poli *bitmap* popisují kresbu po řádcích. Parametry x_{b0} a y_{b0} definují v bitové mapě bod, který je považován za její počátek. Bitová mapa se kreslí do výsledného obrazu tak, že je její počátek umístěn do právě platné pozice v rastru výsledného obrazu specifikované pomocí funkce glRasterPos. Parametry x_{bi} a y_{bi} určují přírůstek, který bude k platné pozici v rastru připočten po vykreslení bitmapy (tato možnost modifikovat aktuální pozici v rastru je výhodná např. při vykreslování jednotlivých znaků dohromady tvořících textový řetězec). Hodnoty parametrů x_{b0} a y_{b0}, x_{bi} a y_{bi} se zadávají v obrazových bodech.

K manipulaci s rastrovými obrazy OpenGL nabízí funkce **glReadPixels**, **glDrawPixels**, **glCopyPixels**.

void **glReadPixels**(GLint *x*, GLint *y*, GLsizei *width*, GLsizei *height*,
GLenum *format*, GLenum *type*, GLvoid **pixels*);

Získání hodnot z obrazové paměti

Funkce **glReadPixels** čte hodnoty z obdélníkové oblasti obrazové paměti, jejíž levý dolní roh je zadán hodnotou parametů *x, y* a rozměr parametry *width* a *height*. Hodnoty uvedených parametrů se zadávají v obrazových bodech. Hodnoty přečtené z obrazové paměti jsou uloženy do pole, na které ukazuje parametr *pixels*. Parametr *format* určuje, jaká data se z obrazové paměti mají číst. Možné hodnoty parametru a jejich význam ukazuje tabulka 5.6. Parametr *type* definuje datový typ, v němž budou hodnoty uloženy do pole pixels (tabulka 5.7).

Tabulka 5.6. Možné hodnoty parametru *format* a jejich význam.

Význam parametru format

Hodnota parametru *format*	Informace přečtená z obrazové paměti
GL_COLOR_INDEX	barevný index
GL_RGB	barevné složky *R*, *G*, *B* (v udaném pořadí)
GL_RGBA	barevné složky *R*, *G*, *B*, *A* (v udaném pořadí)
GL_RED	červená barevná složka
GL_GREEN	zelená barevná složka
GL_BLUE	modrá barevná složka
GL_ALPHA	alfa kanál
GL_LUMINANCE	jas
GL_LUMINANCE_ALPHA	jas a alfa kanál
GL_STENCIL_INDEX	index v masce (viz. help nebo literatura)
GL_DEPTH_COMPONENT	hloubka

Tabulka 5.7. Možné hodnoty parametru *type* a jejich význam.

Význam parametru type

Hodnota parametru *type*	Datový typ
GL_UNSIGNED_BYTE	8-bitové celé číslo bez znaménka
GL_BYTE	8-bitové celé číslo
GL_BITMAP	jednotlivé bity v 8-bitovém celém čísle
GL_UNSIGNED_SHORT	16-bitové celé číslo bez znaménka
GL_SHORT	16-bitové celé číslo
GL_UNSIGNED_INT	32-bitové celé číslo bez znaménka
GL_INT	32-bitové celé číslo
GL_FLOAT	číslo v plovoucí řádové čárce

void **glDrawPixels**(GLsizei *width*, GLsizei *height*, GLenum *format*,
GLenum *type*, GLvoid **pixels*);

Zápis hodnot do obrazové paměti

Funkce **glDrawPixels** vykresluje obdélníkovou oblast, jejíž rozměry jsou specifikovány pomocí parametrů *width* a *height*. Levý dolní roh oblasti je umístěn na aktuální pozici v rastru. Pole, na které ukazuje parametr *pixels*, obsahuje data, která budou vykreslena. Obsah pole *pixels* a způsob uložení dat je popsán pomocí parametrů *format* a *type*. Možné hodnoty a jejich význam jsou stejné jako v případě funkce glReadPixels (tabulky 5.6, 5.7).

```
void glCopyPixels( GLint x, GLint y, GLsizei width, GLsizei height,
                   GLenum type);
```

Kopírovaní oblasti v obrazové paměti

Funkce **glCopyPixels** kopíruje obdélníkovou oblast obrazové paměti, jejíž levý dolní roh a rozměry jsou specifikovány pomocí parametrů *x, y*, *width* a *height* na jiné místo v obrazové paměti. Levý dolní roh kopie oblasti je umístěn na aktuální pozici v rastru. Hodnoty parametrů *x, y*, *width* a *height* se udávají v obrazových bodech.

5.8 Nanášení textury

Doba studia asi 1,5 hod

Čemu se zde naučíte?

Nanášení textur je jednou z metod, pomocí níž lze podstatně zvýšit věrnost zobrazení scény. Chcete-li v OpenGL textury používat, musíte provést následující kroky: 1) Specifikovat, jakou texturu chcete používat. 2) Rozhodnout, jakým způsobem má být textura na povrch nanesena (může být na povrch objektu např. jednoduše pouze nakopírována nebo může být použita k modulaci vypočtených hodnot jasu atd.). 3) Musíte nanášení textur povolit. 4) Při vytváření objektů nesmíte zapomenout přiřadit k vrcholům stěn objektů jejich souřadnice v textuře (připomeňme, že teorii nanášení textur jsme probrali již v kapitole 2.11). V této podkapitole ukážeme, jak lze v OpenGL všechny uvedené kroky realizovat.

Co musíte pro nanášení textury udělat?

Jednoduše lze říci, že textura je nějaký rastrový obraz, který zpravidla bývá přečten ze souboru, někdy je také vytvořen přímo programem samotným. Nejčastěji bývají textury dvojrozměrné (mají plochu), mohou ale být i textury jednorozměrné nebo trojrozměrné. K definování textury slouží funkce glTexImage.

```
void glTexImage2D( GLenum target, GLint level, GLint components,
                   GLsizei width, GLsizei height, GLint border,
                   GLenum format, GLenum type, const GLvoid *pixels);
```

Definice dvojrozměrné textury

Funkce **glTexImage2D** definuje dvojrozměrnou texturu. Specifikovaná textura platí pro následně kreslené objekty tak dlouho, dokud není specifikována textura jiná nebo dokud není používání textur zakázáno. Parametr *target* je určen k budoucímu použití. Ve verzi OpenGL 1.1 musí povinně obsahovat konstantu GL_TEXTURE_2D. Parametr *level* určuje úroveň podrobnosti textury. Hodnota *level* = 0 znamená největší podrobnost. Hodnoty *level* = 1, 2,… znamenají texturu stále menší a menší podrobnosti. Specifikovat více úrovní podrobnosti jedné textury je sice pracné, ale užitečné. Systém si pak může vybrat tu podrobnost, která se za dané situace nejlépe hodí (z kapitoly 2.11 víte, že při nanášení textury hrozí vznik aliasingu, proti němuž lze použitím různých úrovní podrobnosti textury bojovat). Parametr *components* je celé číslo určující, kolika složkami je každý pixel textury popsán. Možné hodnoty jsou 1 (pro každý pixel je jen jas *L*), 2 (*L*, *A*), 3 (*R*, *G*, *B*) a 4 (*R*, *G*, *B*, *A*) (*A* značí hodnotu alfa kanálu). Jak jednotlivé varianty fungují, ukážeme později v tabulce 5.8 v souvislosti s vysvětlením funkce **glTexEnv**. Parametry *width* a *height* určují rozměr textury; *border* je šířka okraje textury. V tomto textu se jednoduše spokojíme s konstatováním, že šířka okraje bývá většinou nulová. Parametry *width* a *height* musí mít hodnotu vyhovující vztahu 2^m+2b, kde *m* je celé číslo a *b* je šířka okraje textury. Maximální velikost textury závisí na implementaci OpenGL, musí však být alespoň 64×64 (bez okraje). Parametr *pixels* ukazuje na místo v paměti, kde je přichystána textura. Parametry *format* a *type* popisují formát a datový typ použitý pro data v poli *pixels*. Význam těchto parametrů je stejný jako u funkcí glReadPixels, glDrawPixels z předchozí podkapitoly (tabulky 5.6, 5.7). Jesliže je pomocí funkce glTexImage2D

zadáváno více úrovní podrobnosti textury, pak musí být funkce volána pro každou podrobnost textury znovu (hodnota parametru *level* při tom roste po 1). Při snížení podrobnosti textury o jeden stupeň se její rozměr v obou směrech zmenší na polovinu. Je typické, že pro jednu texturu je tak zapotřebí připravit její obrazy o velikosti 64×64, 32×32, 16×16, 8×8, 4×4, 2×2 a 1×1. Pro přípravu obrazů textury různých podrobností i pro jejich předání systému je možné a výhodné použít funkce **gluBuild2DMipmaps** (úkol 5.8).

void **glTexImage1D**(GLenum *target*, GLint *level*, GLint *components*,
GLsizei *width*, GLint *border*, GLenum *format*,
GLenum *type*, const GLvoid **pixels*);

Definice jednorozměrné textury

Funkce **glTexImage1D** definuje tzv. jednorozměrnou texturu. Tím se v OpenGL rozumí textura, která se mění jen v jednom směru a ve druhém směru zůstává konstatní. Pole pixels proto obsahuje pouze jednu řadu obrazových bodů.Význam zbývajících parametrů je stejný jako v případě předchozí funkce glTexImage2D.

Povolení textur se provádí voláním **glEnable** s parametrem GL_TEXTURE_2D v případě dvojrozměrné textury nebo s parametrem GL_TEXTURE_1D v případě textury jednorozměrné. Způsob, jakým se textura na povrch objektů nanáší, lze řídit funkcí glTexEnv. Textura totiž nemusí definovat barvu ve vyšetřovaném bodě obrazu přímo, ale může modulovat původní barvu objektu, případně se s ní může určitým způsobem mísit.

void **glTexEnv**{if}[v](GLenum *target*, GLenum *pname*, TYPE *param*);

Nastavení módu nanášení textury

Funkce **glTexEnv** nastavuje způsob určování výsledné barvy v jednotlivých bodech obrazu objektu pokrývaného texturou. Parametr *target* musí mít povinně hodnotu GL_TEXTURE_ENV. Jestliže má parametr *pname* hodnotu GL_TEXTURE_ENV_MODE, pak může parametr *param* nabývat hodnot GL_DECAL, GL_MODULATE nebo GL_BLEND. V módu DECAL barva textury přímo určuje barvu ve výsledném obraze. V ostatních módech je výsledná barva kombinací barvy textury a původní barvy objektu. Možnosti ukazuje tabulka 5.8. Jestliže má argument *pname* hodnotu GL_TEXTURE_ENV_COLOR, pak to znamená, že je funkce použita k definici barvy označené v tabulce 5.8 jako C_C a parametr *param* pak musí ukazovat na pole obsahující čtveřici hodnot, které definují barevné složky R, G, B, A barvy C_C.

Tabulka 5.8. Možné módy nanášení textury.

Počet složek textury	Mód „decal“	Mód „modulate“	Mód „blend“
1	nedefinováno	$C = L_tC_o$, $A = A_o$	$C = (1–L_t)C_o+L_tC_c$, $A = A_o$
2	nedefinováno	$C = L_tC_o$, $A = A_tA_o$	$C = (1–L_t)C_o+L_tC_c$, $A = A_tA_o$
3	$C = C_t$, $A = A_o$	$C = C_tC_o$, $A = A_o$	nedefinováno
4	$C = (1–A_t)C_o+A_tC_t$, $A = A_o$	$C = C_tC_o$, $A = A_tA_o$	nedefinováno

Jaké módy nanášení lze zvolit?

Význam symbolů z tabulky 5.8 je následující: L znamená jas, C barvu a A hodnotu alfa kanálu. Výsledné hodnoty v obraze jsou bez indexu. Index „t“ znamená, že se jedná o hodnoty zjištěné z textury, index „o“ znamená, že se jedná o aktuální hodnoty vypočítané při kreslení objektu, a konečně C_c je barva definovaná příkazem glTexEnv.

Pro každý vrchol plochy, na kterou má být nanesena textura, musí být určeny jeho souřadnice v prostoru textury (teorii nanášení textur jsme již probrali v kapitole 2.11). Ke stanovení souřadnic v textuře slouží funkce glTexCoord.

void **glTexCoord**{1234}[v](TYPE *coords*);

Stanovení souřadnic v textuře

Funkce **glTexCoord** nastavuje aktuální souřadnice v textuře. Nastavené souřadnice platí pro všechny vrcholy následně zadávané voláním funkce glVertex tak dlouho, dokud nejsou v textuře stanoveny souřadnice nové. Vektor souřadnic textury může obsahovat jednu, dvě, tři nebo i čtyři složky, které v OpenGL bývají často pojmenovávány s, t, r, q. Pro jednorozměrné textury se používá souřadnice s, pro dvojrozměrné textury souřadnic s a t (tento případ je nejběžnější). Využití souřadnice r má smysl pouze v souvislosti s trojrozměrnou texturou, kterou ale verze 1.1 OpenGL nepodporuje. Souřadnice q se používá podobně jako složka w u homogenních souřadnic. Její typická hodnota je 1. Bez ohledu na skutečný rozměr textury v obrazových bodech se souřadnice v textuře zadávají normalizované tak, že interval $\langle 0,1 \rangle$ v každém z obou směrů odpovídá celému rozměru textury.

Zvětšování a zmenšování textury

Je jistě jasné (přinejmenším z výkladu v podkapitole 2.11 by tomu tak být mělo), že bodu ve výstupním obraze může v prostoru textury odpovídat bod se zcela libovolnými souřadnicemi, tj. bod, pro který hodnota barevných složek není v textuře k dispozici přímo (víte, že textura je reprezentována diskrétně, což znamená, že jsou k dispozici hodnoty jen ve vybraných bodech). Pro řešení tohoto problému jsou v OpenGL možné dva postupy: 1) Použije se hodnota z nejbližšího bodu textury, pro který je explicitně uchovávána. 2) Provede se bilineární interpolace, při níž se v textuře využije čtyř nejbližších bodů (vytvářejí v textuře čtverec 2×2 body). Považujme vzdálenost dvou sousedních bodů ve výstupním obraze za rovnou hodnotě $d_o = 1$ a transformujme ji do prostoru textury, kde ji označmme d_t. Mohou nastat dva případy, a to $d_t < 1$ nebo $d_t > 1$ (rovnost jsme pro jednoduchost výkladu vyloučili). OpenGL pro tyto případy používá termínu, že se textura „zvětšuje“ nebo „zmenšuje“. Jestliže je hodnota d_t podstatněji větší než 1, je nutné počítat s nebezpečím vzniku aliasingu (kapitoly 2.11 a 3.4). K boji s aliasingem OpenGL nabízí možnost specifikovat texturu v různých úrovních podrobnosti. Je-li více úrovní k dispozici, může pak OpenGL při nanášení textury postupovat dvěma způsoby: 1) Vybere tu úroveň podrobnosti textury, při níž je d_t nejbližší hodnotě 1. 2) Vybere dvě podrobnosti s hodnotou d_t nejbližší hodnotě 1 (pro jednu bude platit $d_t < 1$, pro druhou $d_t > 1$), zjistí požadovanou informaci z textur obou podrobností a mezi oběma získamými hodnotami interpoluje. Poznamenejme, že více úrovní podrobnosti se může uplatnit i při vykreslování jedné jediné plochy, protože vzdálenost d_t obecně není pro různé dvojice sousedních obrazových bodů konstantní. Pokud k pokrytí plochy nestačí jeden exemplář textury, může se textura opakovat. Opačně lze ale také opakování textury zakázat. Ke specifikaci vlastností, které jsme zatím uvedli slouží funkce glTexParameter.

Opakování textury

void **glTexParameter**{if}[v](GLenum *target*, GLenum *pname*, TYPE *param*);

Funkce **glTexParameter** říká, jak se má postupovat při zvětšování a zmenšování textury. Stanovuje také, zda se má textura opakovat. První parametr je vždy buď konstanta GL_TEXTURE_2D nebo konstanta GL_TEXTURE_1D podle toho, zda mají být ovlivněny vlastnosti textury dvojrozměrné nebo jednorozměrné. Parametr *pname* může nabývat hodnot GL_TEXTURE_MAG_FILTER (specifikuje postup při zvětšování) GL_TEXTURE_MIN_FILTER (specifikuje postup při zmenšování), GL_TEXTURE_WRAP_S (specifikuje, zda se textura opakuje ve směru *s*) nebo GL_TEXTURE_WRAP_T (specifikuje opakování ve směru *t*). Možné hodnoty třetího parametru *param* jsou uvedeny v tabulce 5.9. Volba GL_NEAREST způsobí, že se při zvětšování nebo zmenšování vybere z textury bod, který je nejbližší požadované vypočtené teoretické poloze v textuře. Při volbě GL_LINEAR se provádí bilineární interpolace mezi 2×2 nejbližšími body, které jsou v textuře k dispozici. Stejný význam má také první část jména konstanty, je-li použito více úrovní podrobnosti textury. Druhá část jména je pak MIPMAP_NEAREST nebo MIPMAP_LINEAR. Volba MIPMAP_NEAREST způsobí, že je v každém okamžiku kreslení výstupního obrazu použita pouze jediná (nejvhodnější) úroveň podrobnosti textury. Volba MIPMAP_LINEAR specifikuje, že mají být v každém okamžiku použity dvě nejvhodnější úrovně ($d_t < 1$, $d_t > 1$). Mezi hodnotami získanými z obou úrovní se pak interpoluje. Je zřejmé, že volba GL_LINEAR_MIPMAP_LINEAR dává nejkvalitnější výsledky, ale je současně také časově nejnáročnější. Hodnota GL_REPEAT říká, že se textura v daném směru smí opakovat. Při volbě GL_CLAMP naopak dochází k oříznutí souřadnic vypočtených při převodu z prostoru obrazu do prostoru textury na hodnotu z intervalu ⟨0,1⟩ (souřadnice < 0 jsou ořízntuty na hodnotu 0 a souřadnice > 1 jsou ořízntuty na hodnotu 1). Praktickým důsledkem volby GL_CLAMP je, že se v případě potřeby (nestačí-li na pokrytí plochy jediný exemplář textury) opakují krajní řady bodů textury.

Nastavení zvětšování, zmenšování a opakování

Nejbližší soused nebo bilineární interpolace

Tabulka 5.9. Možné hodnoty parametrů *pname* a *param* funkce glTexParameter.

Hodnota *pname*	Možná hodnota *param*
GL_TEXTURE_MAG_FILTER	GL_NEAREST nebo GL_LINEAR
GL_TEXTURE_MIN_FILTER	GL_NEAREST, GL_LINEAR, GL_NEAREST_MIPMAP_NEAREST, GL_NEAREST_MIPMAP_LINEAR, GL_LINEAR_MIPMAP_NEAREST nebo GL_LINEAR_MIPMAP_LINEAR
GL_TEXTURE_WRAP_S	GL_CLAMP, GL_REPEAT
GL_TEXTURE_WRAP_T	GL_CLAMP, GL_REPEAT

Jaké hodnoty lze pro zvětšování, zmenšování a opakování nastavit?

Nanášení textury je zapotřebí povolit voláním funkce **glEnable** s parametrem GL_TEXTURE_2D nebo GL_TEXTURE_1D (podle toho, zda chcete používat jednorozměrnou či dvojrozměrnou variantu textury). Je-li to potřebné, můžete nanášení textury voláním funkce **glDisable** naopak také zakázat. Ukázku nanášení textury přináší následující příklad ilustrující všechny kroky, které musíte při nanášení textury provést. V příkladu je použita jediná úroveň textury. Čtení příkladu by vám nemělo činit potíže, protože všechny funkce, které jsou v něm

Nanášení textury musíte povolit!

použity už byste měli znát. Jedinou výjimkou je funkce **glPixelStorei**, kterou jsme neprobrali. Jejím voláním zde sdělujeme, že si OpenGL má informace o jednotlivých pixelech textury, jejíž popis mu předáváme, v paměti zarovnávat na jednotlivé bajty. Jedná se o paměťově nejúspornější variantu, kdy jsou trojice bajtů představujících informaci pro jeden pixel řazeny v paměti bez mezer těsně za sebou (paměťově méně úspornou, ale možná o něco rychlejší variantou by bylo zarovnávání např. na čtyřbajty nebo dokonce na osmibajty).

Příklad nanášení textury

Tady se textura vytvoří.

Tady se specifikují požadované vlastnosti textury.

Tady se textura nanáší.

```
/*================================================================*/
#include <stdio.h>
#include <gl\glaux.h>
#define checkImageWidth  64
#define checkImageHeight 64

GLubyte checkImage[checkImageWidth][checkImageHeight][3];

void makeCheckImage(void){
  int i,j,c;
  for (i=0; i<checkImageWidth; i++){
    for (j=0; j<checkImageHeight; j++){
      c = ( ((i&0x8)==0)^((j&0x8)==0) )*255;
      checkImage[i][j][0] = (GLubyte) c;
      checkImage[i][j][1] = (GLubyte) c;
      checkImage[i][j][2] = (GLubyte) c;
    }
  }
}

void myInit(void){
  glClearColor(0.0, 0.0, 0.0, 1.0);
  glEnable(GL_DEPTH_TEST);
  glDepthFunc(GL_LEQUAL);
  makeCheckImage();
  glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
  glTexImage2D(GL_TEXTURE_2D, 0,3,checkImageWidth, checkImageHeight,
               0, GL_RGB, GL_UNSIGNED_BYTE, &checkImage[0][0][0]);
  glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
  glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
  glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
  glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
  glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
  glEnable(GL_TEXTURE_2D);
  glShadeModel(GL_FLAT);
}

void myDisplay(void){
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glBegin(GL_QUADS);
    glTexCoord2f(0.0, 0.0); glVertex3f(-2.0, -1.0, 0.0);
    glTexCoord2f(0.0, 1.0); glVertex3f(-2.0,  1.0, 0.0);
    glTexCoord2f(1.0, 1.0); glVertex3f( 0.0,  1.0, 0.0);
    glTexCoord2f(1.0, 0.0); glVertex3f( 0.0, -1.0, 0.0);

    glTexCoord2f(0.0, 0.0); glVertex3f( 1.0, -1.0, 0.0);
    glTexCoord2f(0.0, 1.0); glVertex3f( 1.0,  1.0, 0.0);
    glTexCoord2f(1.0, 1.0); glVertex3f( 2.41421,  1.0, -1.41421);
    glTexCoord2f(1.0, 0.0); glVertex3f( 2.41421, -1.0, -1.41421);
  glEnd();
  glFlush();
}
```

```
void myReshape(GLsizei w, GLsizei h){
  glViewport(0, 0, w, h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(60.0, 1.0*(GLfloat)w/(GLfloat)h, 1.0, 30.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glTranslatef(0.0, 0.0, -3.6);
}

/*----------------------------------------------------------------*/
int main(void){
  auxInitDisplayMode(AUX_SINGLE | AUX_RGBA);
  auxInitPosition(0, 0, 400, 400);
  auxInitWindow("Texture Mapping");
  myInit();
  auxReshapeFunc((void ( _stdcall *)(long,long))myReshape);
  auxMainLoop((void ( _stdcall *)(void))myDisplay);
  return 0;
}
```

Úkol 5.8

Úkol 5.8. Doplňte program z úkolu 5.4 tak, že na některé povrchy scény nanesete texturu. Proveďte nejprve ve variantě s jednou úrovní podrobností textury, pak s více úrovněmi. Vyzkoušejte různé módy nanášení textury.

Rešení. Vzorová řešení naleznete v souborech „úkol5_8a.cpp“ a „úkol5_8b.cpp“ na přiloženém CD. V prvním případě je použita pouze jediná úroveň podrobnosti textury. (Spusťte si program a povšimněte si vzniku aliasingu na podložce pod tělesy.) Ve druhém případě je využito více úrovní podrobnosti. K zadání textury je zde použito funkce **gluBuild2DMipmaps**. Ve druhém případě si rovněž povšimněte použití funkce **glHint**, jejímž voláním s parametry GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST se zde ujišťujeme, že se při nanášení textury bude uvažovat perspektivní projekce přesně (některé implementace OpenGL by jinak mohly souřadnice v prostoru textury počítat interpolací). □

SHRNUTÍ

Shrnutí: Po prostudování textu této podkapitoly a po vyřešení úkolů byste měli být schopni vytvářet vlastní programy, které nanášení textur provádí. Možnosti nanášení textur nyní znáte v rozsahu, jak je definuje verze 1.1 OpenGL, která je k dispozici ve všech verzích MS Windows. Pokud budete mít k dispozici verzi vyšší, seznamte se s rozšířeními, která ohledně nanášení textur nabízí. Vyšší verze totiž nabízí rozšíření právě zde. Je možné, že některá z nich byste rádi využili. Funkce, které jsme zde probrali my, ovšem ale platí stále.

5.9 Evaluátory

Doba studia asi 1 hod

Čemu se zde naučíte?

Název „evaluátory“ OpenGL používá pro nástroje umožňující zejména vykreslování Bèzierových křivek a ploch (jak ale brzy ukážeme, jsou možnosti použití evaluátorů poněkud širší). V této podkapitole se dovíte, o jaké nástroje se jedná a jak je používat.

Co je jednorozměrný evaluátor?

Jednorozměrný evaluátor je nástrojem pro vyhodnocování výrazů tvaru

$$C(u)=\sum_{i=0}^{n} B_i^n(u)P_i \quad , \quad \text{kde} \quad B_i^n(u)=\binom{n}{i}u^i(1-u)^{n-i} \qquad (5.2)$$

a P_i jsou vektory souřadnic tzv. kontrolních bodů a kde n je zvolený stupeň polynomu. Rozepišme uvedené vztahy podrobně pro $n = 1$ a $n = 3$. Dostaneme

Evaluátor prvního a třetího stupně

$$C(u) = (1-u)P_0 + uP_1 \,, \tag{5.3}$$

$$C(u) = (1-u)^3 P_0 + \frac{1}{3}u(1-u)^2 P_1 + \frac{1}{3}u^2(1-u)P_2 + u^3 P_3 \,. \tag{5.4}$$

Určitě ihned vidíte, že vztah (5.3) provádí lineární interpolaci souřadnic kontrolních bodů. Měli byste také vědět, že pro $u \in \langle 0,1 \rangle$ je vztah (5.3) rovnicí úsečky, která spojuje body P_0, P_1. Z předmětu „počítačová grafika I“ byste také měli znát, že vztah (5.4) je rovnicí oblouku Bèzierovy křivky třetího stupně. Jednorozměrný evaluátor je nástroj, který slouží k vyčíslování hodnoty $C(u)$ definované výrazem (5.2) a případně také derivací této hodnoty. Před použitím evaluátoru je nejprve nutné specifikovat jeho vlastnosti.

void **glMap1**{fd}(GLenum *target*, TYPE *u1*, TYPE *u2*, GLint *stride*,
GLint *order*, const TYPE **points*);

Vytvoření jednorozměrného evaluátoru

Funkce **glMap1** definuje jednorozměrný evaluátor. Parametr *target* určuje, jaký praktický význam mají v programu zadané kontrolní body a také, k čemu má být použita vypočtená hodnota $C(u)$. Možné hodnoty parametru *target* ukazuje tabulka 5.10. Z tabulky je vidět, že hodnoty $C(u)$ zdaleka nemusí být interpretovány jen jako geometrické souřadnice bodů ležících na křivce. Mohou být chápany také např. jako souřadnice barevné, souřadnice v prostoru textury, složky normály atd. Parametry u_1 a u_2 určují rozsah, v němž se bude pohybovat souřadnice u. Parametr *points* je ukazatel na pole obsahující souřadnice kontrolních bodů. Parametrem *stride* se systému sděluje, o kolik hodnot souřadnic (tj. o kolik paměťových pozic float nebo double) má OpenGL v poli *points* postoupit, když chce přejít od jednoho kontrolního bodu k následujícímu. Konečně parametr *order* určuje počet kontrolních bodů (je roven stupni polynomu plus 1). Pro každou hodnotu parametru *target* je možné v programu definovat nezávislý evaluátor. V programu tak může existovat v jednom okamžitu nejvýše po jednom evaluátoru ke každé možné hodnotě parametru *target*.

Nastavení parametrů evaluátoru

Tabulka 5.10. Možné hodnoty parametru *target* funkce glMap1 a jejich význam.

Jaké evaluátory můžete vytvořit?

Hodnota parametru *target*	Jaký význam mají kontrolní body a hodnota $C(u)$
GL_MAP1_VERTEX_3	souřadnice x, y, z bodu
GL_MAP1_VERTEX_4	souřadnice x, y, z, w bodu
GL_MAP1_INDEX	barevný index
GL_MAP1_COLOR_4	barevné složky R, G, B, A
GL_MAP1_NORMAL	souřadnice x, y, z normály
GL_MAP1_TEXTURE_COORD_1	souřadnice s textury
GL_MAP1_TEXTURE_COORD_2	souřadnice s, t textury
GL_MAP1_TEXTURE_COORD_3	souřadnice s, t, r textury
GL_MAP1_TEXTURE_COORD_4	souřadnice s, t, r, q textury

Každý nadefinovaný evaluátor je před jeho použitím nezbytné povolit. To se provádí, jak je v OpenGL obvyklé, pomocí volání funkce **glEnable**, přičemž se jako parametr použije odpovídající konstanta z levého sloupce tabulky 5.10. Je-li to potřebné, lze použití dříve povoleného evaluátoru naopak také zakázat, a to voláním funkce **glDisable** se stejným parametrem jako při povolení.

Jestliže jste již evaluátor nadefinovali a povolili, můžete jej v programu používat. Základní varianta použití spočívá v tom, že kdykoli to potřebujete, můžete evaluátor požádat, aby pro vámi zadanou konkrétní hodnotu *u* spočítal hodnotu $C(u)$ z výrazu (5.2). K tomu slouží následující funkce.

void **glEvalCoord1**{fd}[v](TYPE *u*);

Funkce **glEvalCoord1** vyhodnocuje pro hodnotu *u* zadanou jako parametr hodnotu $C(u)$ ve všech nadefinovaných a povolených jednorozměrných evaluátorech. Jak se hodnoty vypočtené jednotlivými evaluátory použijí, závisí na typu evaluátoru (na tom, jaká byla hodnota parametru *target* při jeho zřizování). Jestliže se např. jedná o evaluátor produkující souřadnice bodů (při zřizování evaluátoru měl parametr *target* hodnotu např. GL_MAP1_VERTEX_3), pak si můžete představit, že funkce glEvalCoord1 nakonec ještě volá funkci **glVertex** a tím zajistí, že je vypočítaná hodnota $C(u)$ předána na místo, kde se očekává a kde má smysl. (Pro jiné hodnoty parametru *target* si podobně představte, že glEvalCoord1 volá funkce **glColor**, **glNormal**, **glTextCoord** a hodnotu $C(u)$ jim předává.) Příklad definice a použití evaluátoru ke kreslení oblouku Bèzierovy křivky třetího stupně podává následující příklad. S využitím funkce glEvalCoord1 je zde požadovaná křivka nakreslena jednoduše jako posloupnost dostatečně krátkých úseček.

Jak se evaluátoru používá?

```
/*================================================================*/
#include <stdio.h>
#include <gl\glaux.h>
GLfloat ctrlpoints[4][3]={{-4.0, -4.0, 0.0},{-2.0, 4.0, 0.0},
                          { 2.0, -4.0, 0.0},{ 4.0, 4.0, 0.0}};
void myInit(void){
  glClearColor(0.0, 0.0, 0.0, 1.0);
  glMap1f(GL_MAP1_VERTEX_3, 0.0, 1.0, 3, 4, &ctrlpoints[0][0]);
  glEnable(GL_MAP1_VERTEX_3);
  glShadeModel(GL_FLAT);
}
void myDisplay(void){
  int i;
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(1.0, 1.0, 1.0);
  glBegin(GL_LINE_STRIP);
    for (i=0; i<=30; i++) glEvalCoord1f((GLfloat) i/30.0);
  glEnd();
  glFlush();
}
void myReshape(GLsizei w, GLsizei h){
  glViewport(0,0,w,h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  if (w <= h) glOrtho(-5.0, 5.0, -5.0*(GLfloat)h/(GLfloat)w,
                      5.0*(GLfloat)h/(GLfloat)w, -1.0, 1.0);
      else    glOrtho(-5.0*(GLfloat)h/(GLfloat)w,
                      5.0*(GLfloat)h/(GLfloat)w,
```

Tady se evaluátor definuje ...

... a tady použije.

```
                      -5.0, 5.0, -1.0, 1.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
}
/*-------------------------------------------------------------------*/
int main(void){
  auxInitDisplayMode(AUX_SINGLE | AUX_RGBA);
  auxInitPosition(0, 0, 400, 400);
  auxInitWindow("Bezierova krivka");
  myInit();
  auxReshapeFunc((void ( _stdcall *)(long,long))myReshape);
  auxMainLoop((void ( _stdcall *)(void))myDisplay);
  return 0;
}
/*===================================================================*/
```

Postup ukázaný v předchozím případě, kdy jsme sice k výpočtu souřadnic použili evaluátor, ale vlastní kreslení jsme již prováděli více či méně sami, se může zdát nepohodlný. OpenGL nabízí prostředky umožňující dosáhnout téhož výsledku jednodušeji. Jedná se o funkce **glMapGrid1** a **glEvalMesh1**.

void **glMapGrid1**{fd}(GLint *n*, TYPE *u1*, TYPE *u2*);

Definování posloupnosti hodnot parametru

Funkce **glMapGrid1** definuje v intervalu $\langle u_1,u_2\rangle$ posloupnost hodnot parametru u o n pravidelných krocích. Délka kroku a i-tý prvek posloupnosti jsou $(u_2-u_1)/n$, resp. $i(u_2-u_1)/n + u_1$.

void **glEvalMesh1**(GLenum *mode*, GLint *p1*, GLint *p2*);

Využití posloupnosti hodnot parametru ke kreslení

Funkce **glEvalMesh1** použije posloupnosti parametrů definované pomocí funkce **glMapGrid1** pro výpočet ve všech definovaných a povolených evaluátorech. (Pro názornost si můžete představit evaluátor, jehož výsledkem jsou souřadnice bodů.) Výsledek je také vykreslen, a to způsobem specifikovaným pomocí parametru *mode*. Je-li hodnota parametru GL_POINT, pak jsou vykresleny pouze jednotlivé body, jejichž souřadnice evaluátor pro postupně vzrůstající hodnoty parametru u vypočítal. Je-li hodnota GL_LINE, jsou body navíc pospojovány úsečkami. Kreslení začíná parametrem hodnoty u_{p1} a končí parametrem hodnoty u_{p2}.

Podobné jako použití evaluátorů jednorozměrných je i použití evaluátorů dvojrozměrných. Dvojrozměrné evaluátory jsou nástroje pro výpočet hodnot funkcí tvaru

$$S(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} B_i^n(u)B_j^m(v)P_{ij} \quad . \tag{5.5}$$

Měli byste vědět, že vztah (5.5) je rovnicí plochy. Pro $m = n = 1$ a $u,v\in\langle 0,1\rangle$ se např. jedná o plát bilineární plochy specifikovaný čtyřmi body v jeho rozích , pro $m = n = 3$ o se jedná o plát Bèzierovy plochy třetího stupně, který je specifikován šestnácti kontrolními body. Ačkoli jsou možné i jiné varianty, využijete dvojrozměrné evaluátory pravděpodobně nejčastěji ke kreslení ploch. Prostředky, které OpenGL pro práci s dvojrozměrnými evaluátory nabízí, jsou zcela analogické prostředkům pro práci s evaluátory jednorozměrnými.

void **glMap2**{fd}(GLenum *target*,
TYPE *u1*, TYPE *u2*, GLint *ustride*, GLint *uorder*,
TYPE *v1*, TYPE *v2*, GLint *vstride*, GLint *vorder*,
TYPE **points*);

Definování dvojrozměrných evaluátorů

Funkce **glMap2** definuje dvojrozměrný evaluátor. Pro hodnotu parametru *target* opět platí tabulka 5.10 s tím, že v názvu konstanty je místo podřetězce MAP1 podřetězec MAP2. Takto modifikované názvy konstant se použijí jako parametr i pro funkce **glEnable** a **glDisable**, kterými se činnost dvojrozměrných evaluátorů povoluje, případně zakazuje. Význam každé z obou čtveřic (u_1, u_2, *ustride*, *uorder*), (v_1, v_2, *vstride*, *vorder*) parametrů je stejný jako význam čtveřice (u_1, u_2, *stride* a *order*) pro funkci glMap1. Každá z obou čtveřic pro funkci glMap2 popisuje vlastnosti v jednom ze dvou možných parametrických směrů (vidíme tedy, že vlastnosti obecně nemusí být v obou směrech stejné).

void **glEvalCoord2**{fd}[v](TYPE *u*, TYPE *v*);

Získání hodnoty dvojrozměrných evaluátorů

Funkce **glEvalCoord2** vyhodnocuje nadefinovaný a povolený evaluátor pro hodnoty *u* a *v* zadané jako parametr. Evaluátory vypočítávající souřadnice bodů (jedná se o evaluátory, při jejichž zřizování parametr *target* nabýval hodnoty GL_MAP2_VERTEX_3 nebo GL_MAP2_VERTEX_4) mohou automaticky počítat také normály k povrchu a asociovat je s generovanými vrcholy. Uvedená možnost se povolí voláním funkce glEnable s parametrem GL_AUTO_NORMAL. Možnost je to určitě vítaná, protože si nepochybně uvědomujete, že normály jsou nezbytné k výpočtu osvětlení.

void **glMapGrid2**{fd}(GLint *nu*, TYPE *u1*, TYPE *u2*,
GLint *nv*, TYPE *v1*, TYPE *v2*);

Definování sítě hodnot v prostoru parametrů

Funkce **glMapGrid2** definuje množinu dvojic hodnot parametrů *u*, *v* tak, že interval $\langle u_1, u_2\rangle$ je rozdělen na n_u kroků a interval $\langle v_1, v_2\rangle$ je rozdělen na n_v kroků, a to způsobem, který jsme popsali již dříve v souvislosti s funkcí glMapGrid1. Dělením obou intervalů jsou definovány hodnoty u_i, v_j. Funkce glMapGrid2 definuje v prostoru parametrů *u*, *v* síť bodů zahrnující všechny možné dvojice u_i, v_j.

void **glEvalMesh2**(GLenum *mode*, GLint *p1*, GLint *p2*, GLint *q1*, GLint *q2*);

Vykreslení plochy

Funkce **glEvalMesh2** provede pro všechny uzly sítě, která byla definována dřívějším voláním funkce **glMapGrid2**, výpočet všech povolených evaluátorů. (Pro názornost si opět můžete představit evaluátor, jehož výsledkem jsou souřadnice bodů.) Parametr *mode* může mít kromě hodnot GL_POINT a GL_LINE, které již znáte z jednorozměrné varianty, nyní také hodnotu GL_FILL. Použití této hodnoty způsobí, že se plocha vykreslí pomocí vyplněných čtyřúhelníků. Význam dvojic (p_1, p_2) a (q_1, q_2) parametrů je stejný jako význam dvojice (p_1, p_2) ve funkci glEvalMesh1. Dvojice (p_1, p_2) platí ve směru parametru *u* a dvojice (q_1, q_2) ve směru parametru *v*. Příklad definice dvojrozměrného evaluátoru a jeho použití ke kreslení plátu Bèzierovy plochy pomocí funkce glEvalMesh2 ukazuje následující příklad:

Příklad použití dvojrozměrného evaluátoru

```
/*============================================================*/
#include <stdio.h>
#include <gl\glaux.h>
#include <math.h>

GLfloat ctrlpoints[4][4][3] = {
{{-1.5,-1.5,4.0},{-0.5,-1.5,2.0},{0.5,-1.5,-1.0},{1.5,-1.5,2.0}},
{{-1.5,-0.5,1.0},{-0.5,-0.5,3.0},{0.5,-0.5, 0.0},{1.5,-0.5,-1.0}},
{{-1.5,0.5,4.0},{-0.5,0.5,0.0},{0.5,0.5,3.0},{1.5,0.5,4.0}},
{{-1.5,1.5,-2.0},{-0.5,1.5,-2.0},{0.5,1.5,0.0},{1.5,1.5,-1.0}}};

void initlights(void){
  GLfloat ambient[]       = {0.2, 0.2, 0.2, 1.0};
```

```
  GLfloat position[]      = {0.0, 0.0, 2.0, 1.0};
  GLfloat mat_diffuse[]   = {0.6, 0.6, 0.6, 1.0};
  GLfloat mat_specular[]  = {1.0, 1.0, 1.0, 1.0};
  GLfloat mat_shininess[] = {50.0};
  glEnable(GL_LIGHTING);
  glEnable(GL_LIGHT0);
  glLightfv(GL_LIGHT0, GL_AMBIENT, ambient);
  glLightfv(GL_LIGHT0, GL_POSITION, position);
  glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
  glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
  glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
}
```

Definice osvětlení a materiálu

```
void myDisplay(void){
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glPushMatrix();
  glRotatef(85.0, 1.0, 1.0, 1.0);
  glEvalMesh2(GL_FILL, 0, 20, 0, 20);
  glPopMatrix();
  glFlush();
}
```

Vykreslení plochy

```
void myInit(void){
  glClearColor (0.0, 0.0, 0.0, 1.0);
  glEnable (GL_DEPTH_TEST);
  glMap2f(GL_MAP2_VERTEX_3, 0, 1, 3, 4,
          0, 1, 12, 4, &ctrlpoints[0][0][0]);
  glEnable(GL_MAP2_VERTEX_3);
  glEnable(GL_AUTO_NORMAL);
  glEnable(GL_NORMALIZE);
  glMapGrid2f(20, 0.0, 1.0, 20, 0.0, 1.0);
  initlights();
}
```

Definice evaluátoru a sítě bodů

```
void myReshape(GLsizei w, GLsizei h){
  glViewport(0, 0, w, h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  if (w <= h) glOrtho(-4.0, 4.0, -4.0*(GLfloat)h/(GLfloat)w,
                      4.0*(GLfloat)h/(GLfloat)w, -4.0, 4.0);
       else   glOrtho(-4.0*(GLfloat)w/(GLfloat)h,
                      4.0*(GLfloat)w/(GLfloat)h, -4.0, 4.0,
                     -4.0, 4.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
}
/*----------------------------------------------------------------*/
```

Definice zobrazení

```
int main(void){
  auxInitDisplayMode(AUX_SINGLE | AUX_RGBA | AUX_DEPTH );
  auxInitPosition(0, 0, 400, 400);
  auxInitWindow("Bezierova plocha");
  myInit();
  auxReshapeFunc((void ( _stdcall *)(long,long))myReshape);
  auxMainLoop((void ( _stdcall *)(void))myDisplay);
  return 0;
}
/*================================================================*/
```

„Běžný“ hlavní program

Úkol 5.9

Úkol 5.9. Sestavte program, který zobrazí plochu složenou z několika Bèzierových plátů. Polohu kontrolních bodů můžete zadat přímo v programu. Vzorové řešení naleznete v souboru „úkol5_9.cpp“ na přiloženém CD.

SHRNUTÍ

Shrnutí: V tuto chvíli znáte o evaluátorech to nejpodstatnější a měli byste být schopni jich používat. Pro pořádek musíme ale poznamenat, že kromě kreslení Bèzierových křivek a ploch pomocí evaluátorů existuje v OpenGL i možnost kreslit NURBS křivky a plochy. Tuto možnost jsme zde ale vynechali. Pokud byste chtěli NURBS křivky nebo plochy kreslit, zkuste nejprve, zda nevystačíte s informacemi uvedenými v souboru nápovědy k OpenGL. Pokud ne, tak se obraťte na literaturu uvedenou v závěru této kapitoly (zejména na první z uvedených knih).

5.10 Výběr objektů

Doba studia asi 1,5 hod

V mnoha grafických systémech je obvyklé, že s objekty, které byly již dříve na obrazovku vykresleny, je zapotřebí nějak manipulovat. Můžete chtít objekty např. vymazat, posunout, pootočit atd. K tomu je zapotřebí, abyste objekty, které mají být žádané akci podrobeny, uměli vybrat (např. tak, že na ně ukážete pomocí kurzoru). Protože lze výběr objektů považovat v grafických systémech za jednu ze základních operací, je pochopitelné, že i OpenGL její provádění umožňuje. Nástroje, které OpenGL pro výběr objektů nabízí, se v této podkapitole naučíte používat.

Čemu se zde naučíte?

K tomu, abyste mohli objekty vybírat, musíte provést následující: 1) Musíte objektům přidělovat jména (po provedení výběru vám OpenGL vrátí jména objektů, které jste vybrali). 2) Musíte specifikovat místo v paměti, kam vám má OpenGL vracet své odpovědi na vámi požadované výběry. 3) Musíte provést výběr samotný. Všechny uvedené kroky popíšeme dále podrobněji.

Co musíte k provádění výběru udělat.

OpenGL pojmenovává objekty posloupností celých čísel. K vytváření jmen OpenGL používá zásobník („last in, first out“), který smí obsahovat celá čísla. Aktuální obsah zásobníku se považuje za jméno objektu. Jméno specifikované jistým konkrétním obsahem zásobníku platí pro všechny následně kreslené objekty tak dlouho, dokud obsah zásobníku nezměníte (pokud chcete, aby objekty měly jména jedinečná, musíte před kreslením každého objektu obsah zásobníku změnit). Použití zásobníku je při vytváření jmen účelné. Vykreslované objekty totiž obvykle mají hierarchickou strukturu. Pomocí jména lze pak jednoduše zjistit, kde se ve struktuře objektu nachází ta část, která byla obsluhou vybrána. Obsluha zásobníku se děje pomocí funkcí **glInitNames**, **glPushName**, **glPopName** a **glLoadName**.

Vytváření jmen objektů

```
void glInitNames (void);
void glPushName (GLuint name);
void glPopName (void);
void glLoadName (GLuint name);
```

Funkce **glInitNames** slouží k vytvoření prázdného zásobníku. Pomocí funkcí **glPushName** / **glPopName** se na vrchol zásobníku číslo vloží, resp. se z vrcholu zásobníku vyjme. Funkce **glLoadName** přepíše číslo na vrcholu zásobníku číslem, které je zadáno jako parametr *name* (určitě víte, že funkci glLoadName by bylo možné nahradit dvojicí glPopName a glPushName).

Místo v paměti, kam má OpenGL vracet své odpovědi na provedené výběry, se specifikuje pomocí funkce **glSelectBuffer**.

void **glSelectBuffer** (Glsizei *size*, GLuint **buffer*);

Specifikace místa pro výsledek výběru

Parametr *buffer* udává adresu místa, které jste pro odpovědi rezervovali a parametr *size* jeho velikost. Délka odpovědi systému závisí na tom, kolik objektů jste vybrali a jak dlouhá jména mají (blíže o tom v následujícím textu). Délku budoucích odpovědí dopředu neznáte. Nezbývá proto, než abyste místo pro odpovědi přichystali s dostatečnou rezervou.

Provádění výběru

Provedení vlastního výběru se může zdát na první pohled poněkud nezvyklé (po jisté praxi ale postup pravděpodobně shledáte velmi racionálním). Za zásadní lze považovat následující dva rysy: 1) OpenGL neudržuje žádnou datovou strukturu obsahující informaci o objektech, které již byly nakresleny. V okamžiku výběru proto musíte popis množiny objektů, nad níž má být výběr proveden, OpenGL předat tak, jako byste je chtěli znovu nakreslit. Při provádění výběru se ale objekty nevykreslují. Je jen na vás, aby množina objektů, jejichž popis v okamžiku výběru předáváte, souhlasila s množinou objektů, které jsou právě vidět na obrazovce. 2) Vybírají se ty objekty, které nějakou svojí částí padnou do aktuálně platného zorného objemu (specifikovat zorný objem jste se už naučili v podkapitole 5.3). Při výběru objektů žádný obraz nevzniká, a proto zde může mít zorný objem tento nový význam. Za typickou lze považovat situaci, kdy jsou vybírány objekty padnoucí nějakou svojí části do zorného objemu ve tvaru „malého“ kvádru. Je na vás, abyste zorný objem pro účely výběru objektů patřičně nastavili. Podrobnější informace uvedeme dále.

GLint **glRenderMode** (GLenum *mode*);

Přepínání módu kreslení a módu výběru

Funkce **glRenderMode** přepíná mezi módem vykreslování (který jsme používali až doposud a který je nastaven implicitně) a módem vybírání objektů. Uvedeným dvěma módům činnosti OpenGL odpovídají hodnoty parametru *mode* GL_RENDER a GL_SELECT (přesně vzato existuje ještě mód GL_FEEDBACK, o němž se zde ale dále podrobněji zmiňovat nebudeme). Při provádění výběru objektů musíte nejprve pomocí funkce glSelectBuffer specifikovat paměť pro odpovědi. Teprve pak se můžete přepnout do módu výběru pomocí volání funkce glRenderMode(GL_SELECT). Po provedení výběru se pomocí volání funkce glRenderMode(GL_RENDER) musíte přepnout zpět do módu vykreslování. Při tomto druhém přepnutí funkce glRenderMode vrací jako funkční hodnotu počet objektů, které byly vybrány. Další podrobnosti jsou k dipozici na místě specifikovaném pomocí funkce glSelectBuffer (detaily uvedeme později).

Příklad

Následující příklad shrnuje to, co jsme zatím o výběru objektů probrali. Funkce *pick* má parametry *x*, *y*, které specifikují požadované místo výběru. Jak je zřejmé z volání funkce gluOrtho2D, vybírají se zde objekty, které padnou do čtverečku se středem *x*, *y* o straně 0.2. Hodnoty *x*, *y* i velikost čtverečku jsou zde měřeny v souřadné soustavě scény. Použití funkce gluOrtho2D rovněž naznačuje, že je úloha zřejmě rovinná (řekněme, že pro body všech objektů platí $z = 0$). Povšimněte si, že obsah matice projekce zde uschováváme pomocí volání funkce glPushMatrix a nakonec opět obnovujeme pomocí glPopMatrix. To je proto, že tato matice původně popisovala zobrazení a o tuto informaci nechceme během provádění výběru přijít (i později budeme zřejmě chtít scénu opět zobrazovat). Jako funkční hodnotu funkce *pick* vrací počet vybraných objektů. Bližší údaje jsou v poli *selectBuf*.

```
GLuint selectBuf[512];
void pick(GLdouble x, GLdouble y){
  glSelectBuffer (512, selectBuf);
  (void) glRenderMode (GL_SELECT);
  glInitNames();
  glMatrixMode (GL_PROJECTION);
  glPushMatrix ();
  glLoadIdentity ();
  gluOrtho2D (x-0.1, x+0.1, y-0.1, y+0.1);
  glPushName (1);
  tady se nakreslí objekt pojmenovaný "1";
  glLoadName (2);
  tady se nakreslí objekt pojmenovaný "2";
  glPushName (1);
  tady se nakreslí objekt pojmenovaný "2,1";
  glPopMatrix ();
  return glRenderMode (GL_RENDER);
}
```

Struktura informace o vybraném objektu

Po provedení výběru lze bližší informace zjistit analýzou odpovědi předané v dohodnutém místě paměti. Pro každý vybraný objekt odpověď obsahuje jeden záznam. Počet vybraných objektů je, jak víme, navrácen jako funkční hodnota při volání glRenderMode(GL_RENDER), tedy při opětovném přepnutí do módu vykreslování. Záznam pro každý vybraný objekt obsahuje následující údaje: 1) počet čísel, které tvoří jméno objektu. 2) minimální a maximální hodnota souřadnice *z* všech vrcholů objektu, který byl výběrovým objemem zasažen (obě tyto hodnoty se zjišťují v normalizovaném zorném objemu, a leží proto v rozsahu ⟨0,1⟩; násobením hodnotou $2^{32} - 1$ a zaokrouhlením jsou ale převedeny na celé číslo bez znaménka). 3) Konečně jsou pro každý objekt uvedena všechna čísla, která tvoří jeho jméno (výčet začíná ode dna zásobníku).

Zadaní výběrového objemu

Pro usnadnění zadání výběrového objemu lze v praktických úlohach použít následující funkce **gluPickMatrix**.

void **gluPickMatrix** (GLdouble *x*, GLdouble *y*,
GLdouble *width*, GLdouble *height*, GLint *viewport*[4]);

Funkce zadává výběrový objem pomocí velmi často používaných prvků, a to pomocí souřadnic *x*, *y* v okně kresby na výstupním zařízení (může se jednat např. o souřadnice kurzoru) a pomocí požadovaných rozměrů *width* a *height* výběrové plošky. Parametr *viewport* udává polohu a velikost okna kresby na výstupním zařízení. Jeho hodnotu lze získat voláním funkce glGetIntegerv(GL_VIEWPORT, GLint **viewport*). Typické použití funkce gluPickMatrix ukazuje následující příklad.

```
/* Hodnota x, y je poloha kurzoru - získá se od systému.*/
/* Hodnoty width, height jsou rozměry výběrové plošky.*/
/* Hodnotu viewport získáte od OpenGL pomocí glGetIntegerv.*/

glMatrixMode(GL_PROJECTION);
glLoadIdentity( );
gluPickMatrix(x, y, width, height, viewport);
gluPerspective(. . .); /* Tatáž perspektiva jako při kresbě*/
glMatrixMode(GL_MODELVIEW);
/* Tady bude následovat zadávání jmen a kreslení objektů.*/
```

Poznamenejme, že v příkladě uvedeném na předchozí straně nahoře jsme se zadání využívajícímu souřadnice na výstupním zařízení záměrně vyhnuli tak, že jsme výběrový objem zadávali pomocí hodnot měřených v souřadné soustavě scény.

Úkol 5.10

Úkol 5.10. Sestavte program, který kreslí úsečky. Počáteční a koncový bod je zadán pomocí levého tlačítka na myši. Pomocí pravého tlačítka se úsečka, na níž ukazuje kurzor, z kresby odstraní. Vzorové řešení naleznete v souboru „úkol5_10.cpp“ na přiloženém CD.

S H R N U T Í

Shrnutí: V tuto chvíli byste měli mít o OpenGL už vcelku velmi slušnou představu. Připomínám, že jsme se omezili na verzi 1.1, která je dostupná v MS Windows. Pokud byste pracovali s verzí vyšší, určitě už pro vás nebude problém seznámit se s rozšířeními, která nabízí. Pokud vás programování v OpenGl zajímá podrobněji, můžete se obrátit na vcelku bohatou literaturu. Z knižních publikací pojednávajících o programování v OpenGL lze doporučit následující:

Doporučená literatura

- Mason Woo, Jackie Neider, Tom Davis, Dave Shreiner, OpenGL Architecture Review Board, OpenGL(R) Programming Guide: The Official Guide to Learning OpenGL, Version 1.2 (3rd Edition), *Addison-Wesley Pub Co*, 1999, 784 pages, ISBN 0201604582.
- Richard S. Wright Jr., Michael R. Sweet, OpenGL SuperBible (2nd Edition), *Waite Group Pr*, 1999, 696 pages, ISBN 1571691642.
- Dave Astle, Kevin Hawkins, Andre LaMothe, OpenGL Game Programming, Premier Press, Inc., 2002, 808 pages ISBN 0761533303.

Pro programátora, který s OpenGL teprve začíná, bude pravděpodobně nejvhodnější první z uvedených knih (je ovšem také možné, že se vám bude zdát až příliš „upovídaná“). Její třetí vydání, které v přehledu uvádíme, popisuje verzi 1.2 OpenGL (starší popisovala verzi 1.1). Kromě knižních publikací lze podrobné informace čerpat také ze specifikací jednotlivých verzí OpenGL. Naleznete je na http://www.opengl.org. Pro první kroky specifikace ovšem příliš vhodné nejsou.