PŘÍRODOVĚDECKÁ FAKULTA UNIVERZITY PALACKÉHO
KATEDRA INFORMATIKY

BAKALÁŘSKÁ PRÁCE

Fotorealistické zobrazování

2011 Ondřej Bahounek

Anotace

Techniky fotorealistického zobrazování (Monte Carlo, raycasting, raytracing, atd.) jsou podstatnou částí moderní počítačové grafiky. Práce se zaměřuje na metodu raytracing (zpětné sledování paprsku) a popis všech podstatných metod v něm využívaných (reprezentace objektů, světel, vržené stíny, odrazivost a lom paprsku). Na základě těchto znalostí byl vytvořen program RayTracer *schopný zobrazovat realistické 3D scény s nastavitelnou hloubkou rekurze a antialiasingem. Program umožňuje nastavit scénu ve vlastním editoru a exportování výstupu jak do obrázku ve zvoleném formátu, tak i do animace.*

Děkuji vedoucímu práce Mgr. Eduardu Bartlovi, Ph.D. za odborné rady, jež mi byly velmi nápomocny při realizaci práce.

Obsah

Seznam obrázků

Seznam tabulek

1. Úvod

Práce se snaží o vysvětlení základů fotorealistického zobrazování v počítačové grafice. Fotorealistické zobrazování je rozsáhlým oborem, který využívá zejména poznatky z analytické geometrie a optiky. Metod, jak co nejvěrohodněji dosáhnout požadovaného výsledku, je mnoho a výběr každé z nich závisí na konkrétní podobě zobrazované scény. Nejznámější metoda pro fotorealistického zobrazování se nazývá raytracing a právě jí se bude věnovat největší část textu. K demonstraci principů raytracingu byl vytvořen program RayTracer umožňující zobrazení všech základních světelných vlastností: odraz a lom světla, vržené a měkké stíny, skládání barev světla z různých světelných zdrojů.

Text je rozdělen do třech hlavních částí: Část (2.) poskytuje nutné matematické definice a operace využívané téměř ve všech následujících kapitolách a zároveň seznamuje se základními objekty, jejichž interakce budeme později zkoumat. Vždy je nejprve popsán objekt a předveden na ilustraci, načež jsou k němu představeny metody a vzorce pro výpočet potřebných prvků v raytracingu.

V další části (3.) se seznámíme s raytracingem, popíšeme jeho algoritmus, alternativy a modifikace. Současně k raytracingu je nutné zvolit osvětlovací model, kterých také existuje více druhů, avšak my se zde zaměříme pouze na nejrozšířenější Phongův osvětlovací model. Velmi zajímavou kapitolou jsou metody výpočtů vržených stínů a jejich alternativ. V závěru této části se lze dočíst o efektivnějších zobrazovacích metodách a složitějších reprezentací scény.

Poslední část (4.) je věnována programu RayTracer a popsání jeho struktury. Jsou přiloženy demonstrační obrázky pro snazší pochopení a ovládání programu současně s některými jeho zajímavými výstupy. Je zde také popsána i tvorba animací v programu RayTracer (některé lze nalézt na přiloženém DVD).

2. Základní pojmy a metody

K pochopení algoritmu raytracingu je potřeba porozumět některým pojmům z analytické geometrie. Uvedeme pouze ty základní, jež budou potřeba při výpočtech v raytracingu. Často se využívá parametrického vyjádření objektů pro efektivnější výpočty v počítači.

2.1. Základní pojmy

Pro lepší orientaci v textu uvedeme základní definice a pojmy používané v počítačové grafice.

2.1.1. Soustava souřadnic

Prostor, ve kterém se budeme pohybovat, je trojrozměrný s kartézskou soustavou souřadnic. Tj. 3 souřadné osy jsou na sebe vzájemně kolmé a jsou protnuty v jednom bodě – počátku soustavy. Osy jsou označeny x, y, z a jejich význam je následující: x – zleva doprava, y – probíhá shora dolů, z – probíhá odzadu dopředu.

Je potřeba podotknout, že se osy často označují jiným způsobem: y – zleva doprava, x – odzadu dopředu, z – shora dolů. My ovšem budeme používat první možnost vzhledem k tomu, že se běžně v počítačové grafice pracuje s osou y jako osou běžící od shora dolů.

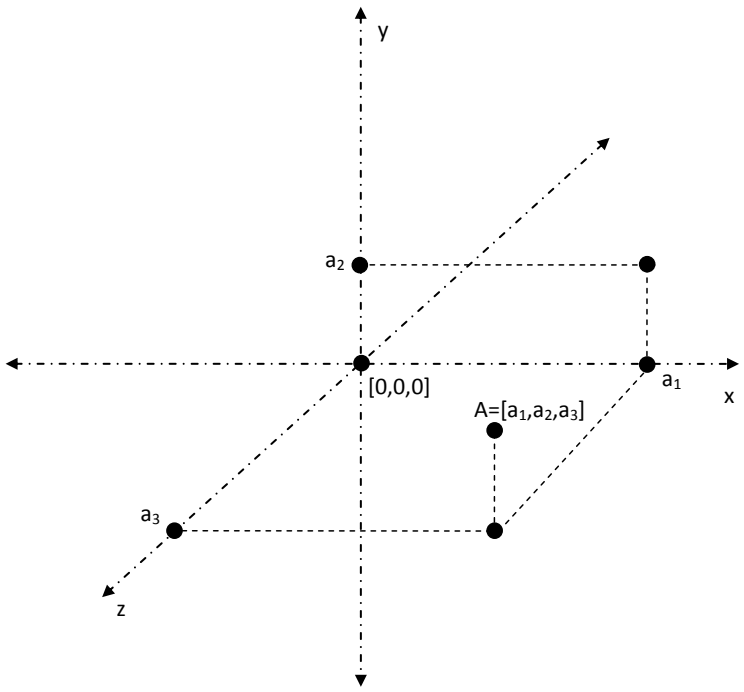
2.1.2. Bod a vektor

Jak bod, tak vektor jsou základní prvky pro reprezentaci objektů ve světě souřadnic. Je zadán třemi číslicemi určujících souřadnice na osách po sobě x, y, z. Např. bod A o souřadnicích $x = a_1, y = a_2, z = a_3$ se označuje $A = [a_1, a_2, a_3]$. Viz obrázek (1.).

Vektor bude opět v našem prostoru zadán třemi číslicemi, neboli souřadnicemi vektoru. Vektor označujeme $\vec{u} = (u_1, u_2, u_3)$. Například mějme dva body A a B. Chceme-li, aby měl vektor orientaci od bodu A do bodu B, vypočítáme jej jako rozdíl $B - A$. Pro opačný vektor (směr od B do A) bude výpočet $A - B$. Zde vidíme, že vektor lze definovat dvěma body a to právě dvěma způsoby lišícími se ve znaménku.

Důležitou vlastností vektoru je jeho velikost, neboli také délka. Jedná se o číslo, jež se pro vektor $\vec{u} = (u_1, u_2, u_3)$ spočítá jako: $\sqrt{u_1^2 + u_2^2 + u_3^2}$ a označuje se jako $\|\vec{u}\|$. Je-li velikost vektoru rovna 1, pak hovoříme o jednotkovém vektoru. Ke každému vektoru lze vytvořit jednotkový vektor. Tato operace se nazývá *normalizace* a daný vektor pak nazýváme normalizovaný. Máme-li tedy vektor $\vec{u} = (u_1, u_2, u_3)$, pak normalizovaný vektor bude

$$\vec{u}_{norm} = \frac{1}{\|\vec{u}\|} \cdot (u_1, u_2, u_3).$$



Obrázek 1.: Souřadnice bodu

Další důležitou operací s vektory je jejich skalární součin. Mějme dva vektory $\vec{u} = (u_1, u_2, u_3)$, $\vec{v} = (v_1, v_2, v_3)$. Jejich skalárním součinem nazveme reálné číslo

$$\vec{u} \cdot \vec{v} = u_1 \cdot v_1 + u_2 \cdot v_2 + u_3 \cdot v_3.$$

Alternativně se skalární součin definuje jako

$$\vec{u} \cdot \vec{v} = \|\vec{u}\| \cdot \|\vec{v}\| \cdot \cos\phi,$$

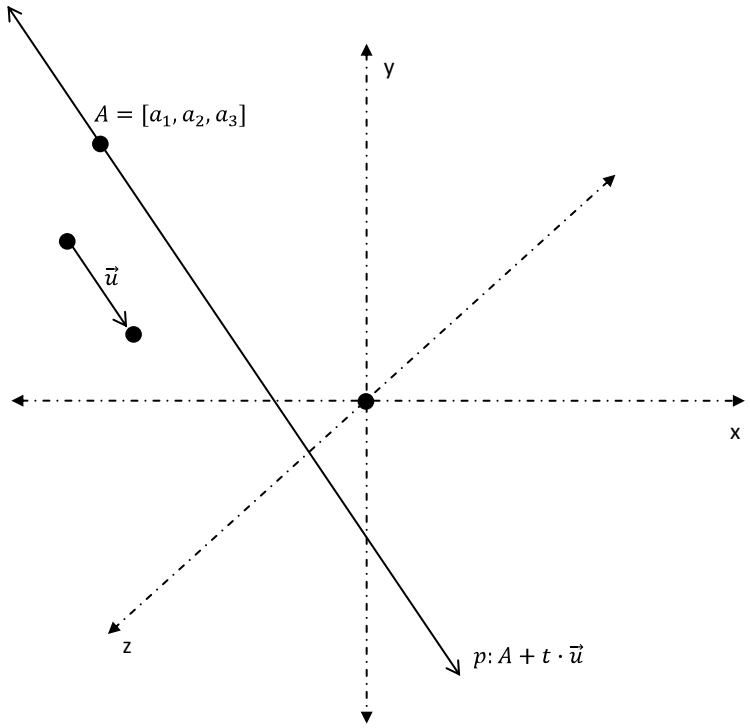
kde ϕ je úhel mezi vektory. Dva vektory jsou na sebe kolmé, právě když je jejich skalární součin roven nule.

2.1.3. Přímka a paprsek

Máme-li představu, co je bod a vektor, lze přejít k definici přímky. K popisu přímky a všech bodů na ní potřebujeme znát její počáteční bod a směr, kterým běží. K určení všech bodů ležících na přímce slouží parametr $t \in \mathcal{R}$. Definováním intervalu, přes který parametr běží, můžeme určit pevný úsek přímky (polopřímku, úsečku). Takové zadání přímky p nazýváme parametrické a má rovnici:

$$p : X = A + t \cdot \vec{u}.$$

Bod X označuje všechny body na přímce, A počáteční bod přímky, $\vec{u}$ její směrový vektor a $t \in \mathcal{R}$. Viz obrázek (2.).



Obrázek 2.: Přímka parametricky

Jeden z významných důsledků takového zadání přímky je, že když má přímka v tomto tvaru normalizovaný směrový vektor, pak parametr t přímo udává délku přímky, neboli vzdálenost mezi body X a A. Z tohoto důvodu budeme v dalším textu (pokud nebude řečeno jinak) uvažovat pouze normalizovaný směrový vektor.

Přímku budeme v další části textu často označovat jako **paprsek** (*paprsek světla*, nebo odražený paprsek zní mnohem lépe, než by vyzněla přímka světla respektive odražená přímka).

2.1.4. Rovina

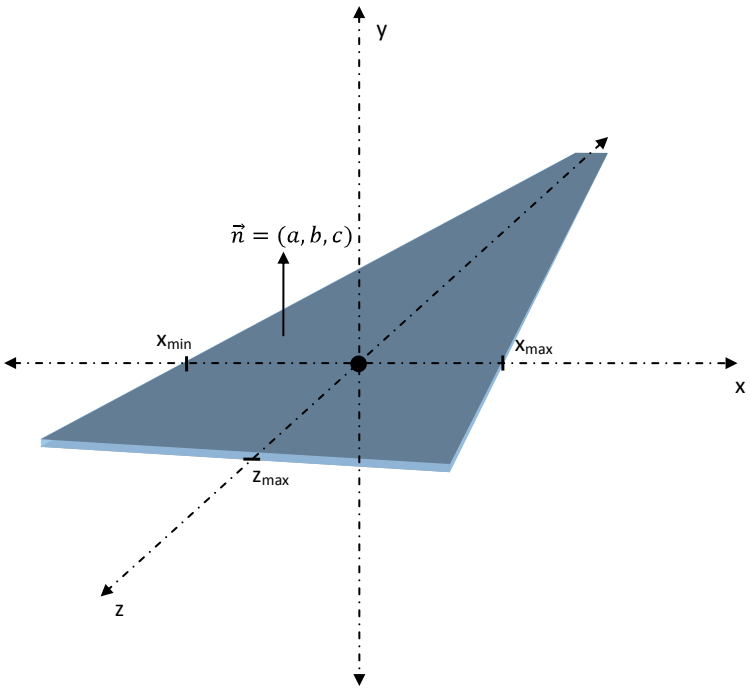
Další objekt, jehož reprezentace nás bude zajímat, je rovina. Rovina může být zadána opět několika způsoby (třemi body neležící na jedné přímce, dvěmi různoběžnými přímkami, přímkou a bodem mimo ní, . . .). Rovinu lze popsat opět jak parametrickou, tak obecnou rovnicí.

Parametrická rovnice má zadán bod A a dva nenulové vektory $\vec{u}$ a $\vec{v}$. Každý bod X je pak určen:

$$X = A + t \cdot \vec{u} + s \cdot \vec{v},$$

kde t,$s \in \mathcal{R}$. Vyloučením parametrů z parametrické rovnice dostaneme obecnou rovnici roviny:

$$a \cdot x + b \cdot y + c \cdot z + d = 0.$$



Obrázek 3.: Rovina

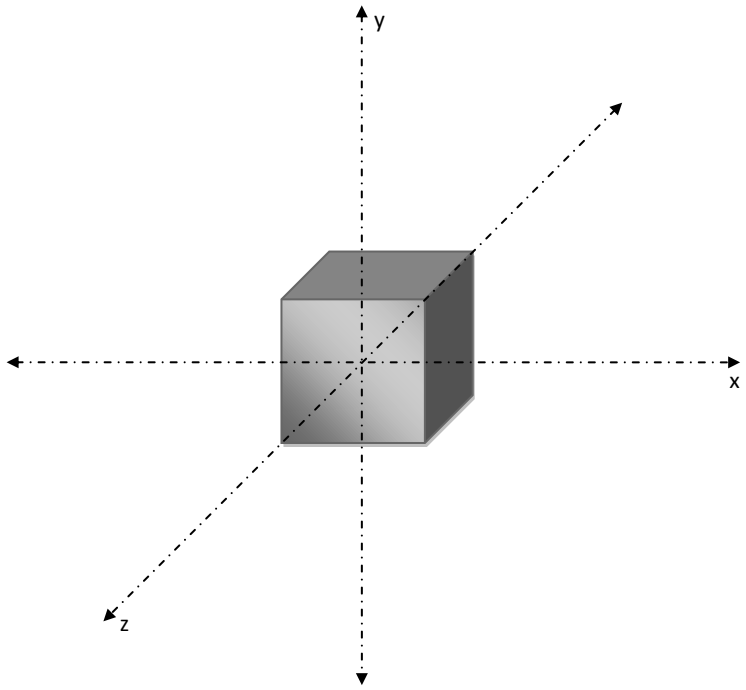
Souřadnice $[x, y, z]$ určují každý bod roviny, parametry a, b, c jsou reálná čísla (mají význam normálového vektoru roviny, tudíž $\vec{n} = (a, b, c) \neq (0, 0, 0)$) a parametr d určuje nejmenší vzdálenost mezi rovinou a počátkem soustavy souřadnic. Obecná rovnice má výhodu, že z ní můžeme pro zadaný bod lehce určit, zda se bod nachází na ní (pak tedy $a \cdot x + b \cdot y + c \cdot z + d = 0$), nebo před ní – ve směru normály ($a \cdot x + b \cdot y + c \cdot z + d > 0$) nebo za rovinou ($a \cdot x + b \cdot y + c \cdot z + d < 0$), což nám bude nápomocno později při dalších výpočtech.

Rovinu lze samozřejmě ohraničit na zadaných intervalech (rovnoběžných k osám soustavy souřadnic) nebo složitěji hraničními přímkami. Nám pro jednoduchost bude stačit první možnost, kdy pro daný bod ležící kdekoli na rovině rozhodneme, zda není již za hranicí. Viz obrázek (3.), kde pro jednoduchost uvádíme rovinu, jež prochází počátkem soustavy souřadnic.

2.1.5. Koule

Koule je těleso, jež má všechny body stejně vzdálené od jistého bodu. Daný bod se označuje středem koule a vzdálenost jejím poloměrem. Středovou rovnicí kružnice se středem $S = [s_1, s_2, s_3]$ a poloměrem r popíšeme všechny body $A = [a_1, a_2, a_3]$, ležící na povrchu koule. Její tvar je

$$(a_1 - s_1)^2 + (a_2 - s_2)^2 + (a_3 - s_3)^2 = r^2.$$



Obrázek 4.: Rovnoosá krychle

Každý bod na kouli lze vyjádřit též pomocí úhlů φ a θ:

$$\begin{aligned} a_1 &= s_1 + r \cdot \sin\theta \cdot \cos\phi, \\ a_2 &= s_2 + r \cdot \sin\theta \cdot \sin\phi, \\ a_3 &= s_3 + r \cdot \cos\theta, \end{aligned}$$

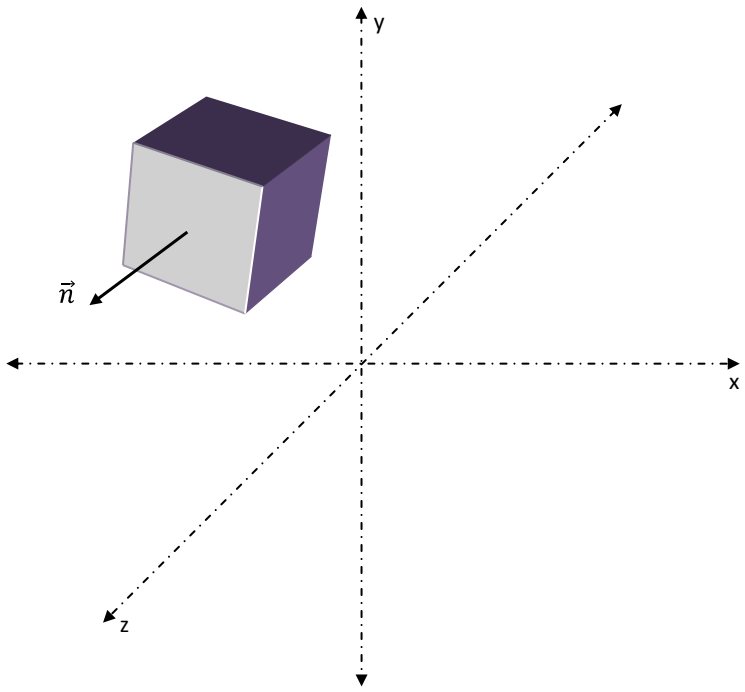
kde $\theta \in [0, \pi]$ a $\phi \in [0, 2\pi]$.

2.1.6. Rovnoosá krychle

Rovnoosá krychle je taková krychle, jež má každou stěnu rovnoběžnou s některou z os soustavy souřadnic. Je zadána svým středem a délkou hrany, viz obrázek (4.).

2.1.7. Obecně orientovaná krychle

Obecně orientovanou krychli definujeme jejím středem, délkou hran a normálou jedné z nich. Obecně se jedná o těleso složené ze šesti rovin, přičemž pro každou z nich platí, že právě jedna z dalších rovin je s ní rovnoběžná různá a ostatní jsou na ni kolmé. Viz obrázek (5.).



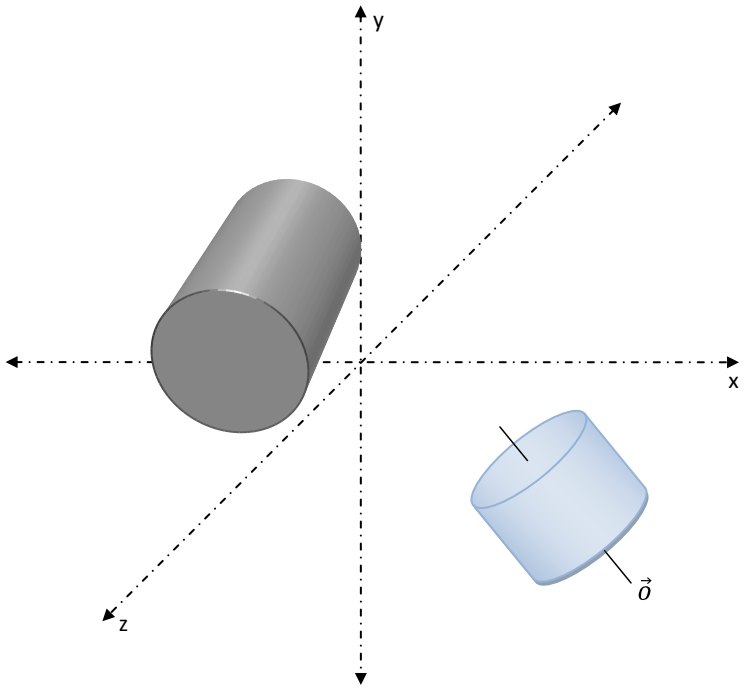
Obrázek 5.: Obecně orientovaná krychle

2.1.8. Válec

Válec neboli cylindr je již poměrně složitější objekt, jenž je zadán středem, směrem hlavní osy, poloměrem a výškou. Příklady válců viz obrázek (6.).

2.2. Základní metody

Dále se budeme věnovat výpočtům odraženého a lomeného paprsku a následně výpočtu průsečíku paprsku s objekty. Je zřejmé, že paprsek může objekt protnout na 0 místech (paprsek objekt neprotíná), na jednom místě (platí například pro rovinu), ve dvou bodech (koule, krychle, válec). Samozřejmě, že pro složitější objekty platí, že jej paprsek může protnout ve více než dvou bodech, avšak takové objekty jsou buď složené z výše uvedených základních objektů, nebo jsou mnohem složitěji definované a těmi se zde zabývat nebudeme. Při průniku paprsku s objektem nás vždy budou zajímat všechny body průniku, nikoli jen jeden – třeba nejbližší, nebo ten nejvzdálenější. Zajímat nás tedy budou všechny body, jež se střetnou se zkoumaným paprskem. Ostatní vlastnosti jako vzdálenost průniku od počátku paprsku se dají odvodit od každého bodu samostatně. Navíc kromě souřadnic bodů průniku, budeme chtít znát normálový vektor v místě dopadu a parametr určující vzdálenost na přímce od zdroje paprsku k místu dopadu.



Obrázek 6.: Dva libovolně orientované válce

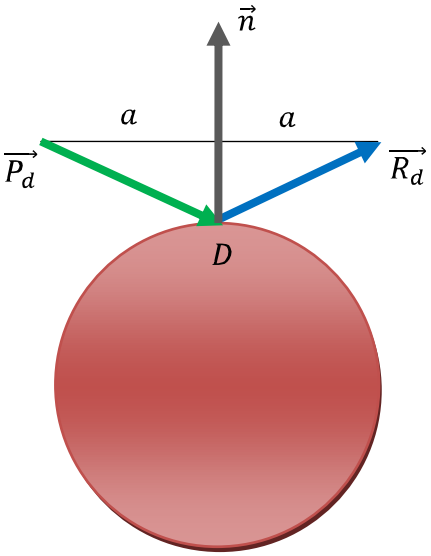
2.2.1. Výpočet odraženého paprsku

Výpočet odraženého paprsku patří k podstatným operacím v raytracingu. Jako typický příklad lze uvést zrcadlo, které je absolutní odrazivý materiál. V literatuře se uvádí termín *specular reflection*.

Běžně je potřeba se vypořádat se situací, kdy na povrch objektu dopadá paprsek a musí se zjistit, jakým směrem se odrazí. K takovému výpočtu je nutné znát i normálový vektor v místě dopadu.

Nechť tedy paprsek $p : P = P_0 + t \cdot \vec{P_d}$ s počátečním bodem P_0 a směrem $\vec{P_d}$ dopadne na objekt v místě dopadu D, k němuž známe normálu $\vec{n}$. Pak odražený paprsek $r : R = R_0 + s \cdot \vec{R_d}$ s počátečním bodem R_0 a směrem $\vec{R_d}$ vypočítáme následovně (viz obrázek 7.):

$$\begin{aligned} R_0 &= D, \\ \vec{R_d} &= \vec{n}(\vec{P_d} \cdot \vec{n}) + a, \\ \vec{P_d} + a &= \vec{n}(\vec{P_d} \cdot \vec{n}), \\ a &= \vec{n}(\vec{P_d} \cdot \vec{n}) - \vec{P_d}, \\ \vec{R_d} &= 2\vec{n}(\vec{P_d} \cdot \vec{n}) - \vec{P_d}. \end{aligned}$$



Obrázek 7.: Výpočet odraženého paprsku

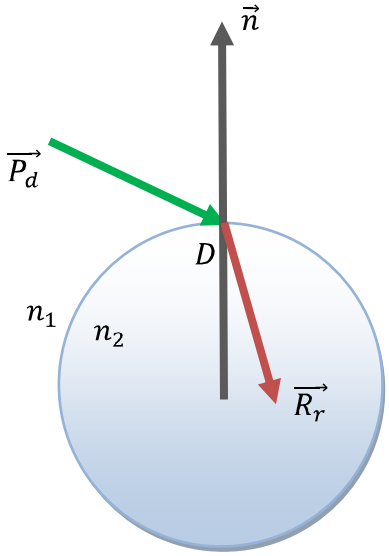
2.2.2. Výpočet lomeného paprsku

K lámání paprsku dojde třeba při průniku paprsku přes sklo. Hovoříme o *specular refraction*.

Abychom mohli přesně určit lomený paprsek, tak podle Snellových zákonů potřebujeme znát indexy lomů obou prostředí, přes která paprsek prochází a opět i normálový vektor v místě dopadu.

Mějme opět k dispozici obrázek (8.). Nechť paprsek $p : P = P_0 + t \cdot \vec{P}_d$ s počátečním bodem P_0 a směrem $\vec{P}_d$ dopadne z prostředí s indexem lomu n_1 na objekt s indexem lomu n_2 v místě dopadu D, k němuž známe normálu $\vec{n}$. Pak vypočítáme lomený paprsek $r : R = R_0 + s \cdot \vec{R}_r$ s počátečním bodem R_0 a směrem $\vec{R}_r$:

$$\begin{aligned} R_0 &= D, \\ n_{12} &= \frac{n_1}{n_2}, \\ c_1 &= -(\vec{n} \cdot \vec{P}_d), \\ c_2 &= \sqrt{1 - n_{12}^2(1 - c_1^2)}, \\ \vec{R}_r &= n_{12} \cdot \vec{P}_d + (n_{12} \cdot c_1 - c_2) \cdot \vec{n}. \end{aligned}$$



Obrázek 8.: Výpočet lomeného paprsku

2.2.3. Průsečík s rovinou

Paprsek může rovinu protnout maximálně v jednom bodě. Protne-li paprsek rovinu, ihned můžeme určit normálový vektor, který bude vždy stejný – bude to normálový vektor roviny, případně její opačný vektor, přichází-li paprsek z opačné strany. Výpočet bodu průniku paprsku p s rovinou ρ lze vzhledem k její reprezentaci v obecném tvaru popsat takto:

$$\begin{aligned} p &: P = P_0 + t \cdot \vec{P_d}, \\ \rho &: a \cdot x + b \cdot y + c \cdot z + d = 0, \end{aligned}$$

dosazením přímky do rovnice roviny:

$$a(P_{0_x} + t \cdot P_{d_x}) + b(P_{0_y} + t \cdot P_{d_y}) + c(P_{0_z} + t \cdot P_{d_z}) + d = 0,$$

$$t = \frac{-(a \cdot P_{0_x} + b \cdot P_{0_y} + c \cdot P_{0_z} + d)}{a \cdot P_{d_x} + b \cdot P_{d_y} + c \cdot P_{d_z}}.$$

Je-li jmenovatel rovný 0, pak je normála kolmá k paprsku a paprsek rovinu neprotíná v jednom bodě, ale splývá s ní, a tedy bychom dostali nekonečně mnoho bodů ležících jak na rovině, tak na paprsku. Souřadnice bodu průniku X nyní dostaneme

$$X = P_0 + t \cdot \vec{P_d}$$

a normálu

$$\vec{n} = (a, b, c).$$

2.2.4. Průsečík s koulí

Paprsek kouli protne maximálně ve dvou bodech. Přičemž jeden bod průniku znamená tečnu ke kružnici a tedy kolmici k normále. Mějme tedy kouli S se středem C a poloměrem r a paprsek p:

$$\begin{aligned} p &: P = P_0 + t \cdot \vec{P}_d, \\ S &: (x - C_x)^2 + (y - C_y)^2 + (z - C_z)^2 = r^2. \end{aligned}$$

Dosazením přímky do rovnice koule:

$$(P_{0_x} + t \cdot P_{d_x} - C_x)^2 + (P_{0_y} + t \cdot P_{d_y} - C_y)^2 + (P_{0_z} + t \cdot P_{d_z} - C_z)^2 = r^2$$

dostaneme kvadratickou rovnici tvaru:

$$k \cdot t^2 + l \cdot t + m = 0$$

s koeficienty k, l, m, kde

$$\begin{aligned} k &= P_{d_x}^2 + P_{d_y}^2 + P_{d_z}^2, \\ l &= 2(P_{d_x}(P_{0_x} - C_x) + P_{d_y}(P_{0_y} - C_y) + P_{d_z}(P_{0_z} - C_z)), \\ m &= (P_{0_x} - C_x)^2 + (P_{0_y} - C_y)^2 + (P_{0_z} - C_z)^2 - r^2. \end{aligned}$$

Máme-li směrový vektor $\vec{P}_d$ paprsku normalizován, je jeho velikost rovna 1 a tedy $k = 1$.

Nyní z výše uvedeného následuje vzorec pro výpočet parametrů t:

$$t_{1,2} = \frac{-l \pm \sqrt{l^2 - 4 \cdot m}}{2}.$$

Je-li diskriminant záporný, pak paprsek kouli neprotíná. Určíme body průniku:

$$\begin{aligned} P_1 &= P_0 + t_1 \cdot \vec{P}_d, \\ P_2 &= P_0 + t_2 \cdot \vec{P}_d. \end{aligned}$$

Blíže počátku paprsku z dvou získaných bodů je ten s nižší kladnou hodnotou parametru t.

K bodu průniku P vypočteme normálový vektor jako:

$$\vec{n} = \frac{C - P}{r}.$$

2.2.5. Průsečík s rovnoosou krychlí

Určení průsečíku paprsku s rovnoosou krychlí nevyjádříme přímým vzorcem, ale popíšeme postup. Prochází se vždy dvě rovnoběžné stěny krychle, které jsou tedy zároveň rovnoběžné s některou ze souřadnicových os a kolmá k jiné. Každou stěnu ohraničíme podle zadaného středu krychle a délky jejích stěn takzvanými extenty, neboli minimální a maximální hranicí. Následně testujeme průsečík paprsku pouze mezi danými hranicemi. Samozřejmě, že normála v bodu průniku bude vždy rovnoběžná se směrem osy, k níž je daná stěna kolmá.

2.2.6. Průsečík s obecně orientovanou krychlí

Krychli zadáme opět jejím středem, délkou stěny a navíc vektorem udávajícím směr jedné z os určující orientaci krychle. Opět neuvedeme přímo vzorec, jen naznačíme postup. Je zřejmé, že každý paprsek může protnout krychli podle místa dopadu v několika bodech. Avšak minimálně ve dvou, maximálně ve třech (narazí-li na jeden z rohů krychle). Nikoli však v jednom bodě, čehož využijeme následně. My ovšem neuvedeme žádný nový vzorec, jelikož ke spočítání průsečíku paprsku s krychlí nám bude stačit uvědomit si, že každá ze šesti stěn krychle je vlastně rovina. Tudíž nejprve spočítáme průnik paprsku s rovinami a jsou-li minimálně tyto průniky dva, máme kandidáta na průsečík paprsku s krychlí. Nejsme však u konce a potřebujeme zjistit, zda průsečík neleží mimo hranice některé z ostatních stěn. K tomu využijeme triku, že se můžeme „podívat“ ve směru do středu krychle o malou delta-změnu z bodu průniku X na bod X' a otestujeme jej, zda leží za všemi stěnami krychle. Připomínáme, že zjištění, zda bod $X' = [x'_1, x'_2, x'_3]$ leží za rovinou, se určí z obecné rovnice roviny, je-li

$$a \cdot x'_1 + b \cdot x'_2 + c \cdot x'_3 < 0.$$

Je potřeba ovšem podotknout, že daný výpočet není příliš efektivní, obsahuje spoustu početních operací, které navíc nejsou příliš přesné v počítačové reprezentaci, ale zato je tento způsob výpočtu značně intuitivní a snadno pochopitelný. Obrázek (9.) se pokusí osvětlit situaci.

2.2.7. Průsečík s válcem

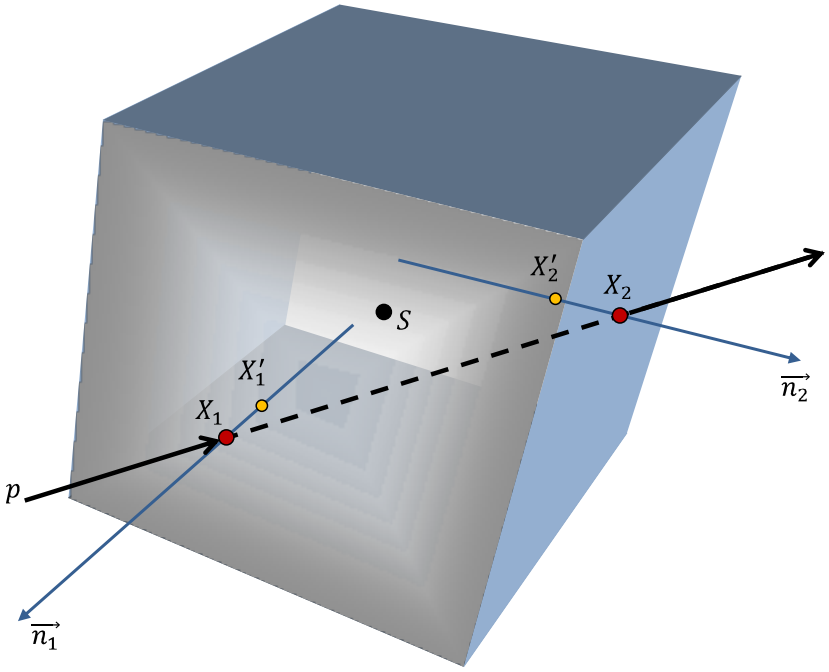
Vypočítat průsečík s válcem je komplikovanější. Je potřeba jej rozdělit na dva výpočty. Jedním se vypočítají případné body průniku na podstavách, druhým výpočtem body průniku pláště. Přičemž můžeme dostat 3 různé kombinace těchto bodů: Oba průniky v podstavách, oba v plášti a nakonec jeden průnik v podstavě a jeden v plášti.

Nejprve popíšeme jednodušší případ – průsečík paprsku s podstavou. Známe-li střed válce S, osu válce $\vec{o}$, výšku h a poloměr r, získáme obou podstav C_1 a C_2. Uvažujme nejprve střed podstavy C_1. Vytvoříme pak rovinu, jejíž normála je směr osy válce s bodem roviny C_1:

$$a \cdot C_{1_x} + b \cdot C_{1_y} + c \cdot C_{1_z} + d = 0.$$

Z rovnice vyjádříme jedinou neznámou D a získáme tak obecnou rovnici roviny obsahující podstavu. Nyní stačí použít známý výpočet průsečíku paprsku s rovinou a otestovat, zda je bod X do vzdálenosti r od středu podstavy C_1. Neboli ověříme, zdali platí nerovnost

$$\sqrt{(x_1 - C_{1_x})^2 + (x_2 - C_{1_y})^2 + (x_3 - C_{1_z})^2} \leq r^2.$$



Obrázek 9.: Průsečík paprsku s krychlí

Normála bodu průniku bude opět daný normálový vektor roviny, tj. tady směr osy válce.

Pro druhou podstavu válce provedeme výpočet analogicky, až na opačný směr osy válce.

Výpočet průsečíku s pláštěm je již složitější. Vychází z myšlenky, že body pláště se dají rozložit na množiny bodů na kružnicích okolo osy válce, tzn. množiny všech bodů stejně vzdálených od osy válce.

Je-li směrový vektor osy válce $\vec{o}$ normalizovaný, můžeme popsat všechny body pláště válce promítnutím na vektor $\vec{l} = C_2 + h \cdot \vec{o}$. Při průniku paprsku pláštěm stačí najít správnou kružnici a poté zjistit její průnik s paprskem promítnutém na rovinu danou kružnicí. Následující postup dokresluje obrázek (10.).

Hlavní idea:

Mějme průsečík X s pláštěm. Určíme vektory $\vec{v_1}$ a $\vec{v_2}$:

$$\begin{aligned} \vec{v_1} &= X - C_2 \quad \ldots \text{vektor z } C_2 \text{ do } X, \\ \vec{v_2} &= (\vec{v_1} \cdot \vec{l}) \cdot \vec{l} \quad \ldots \text{promítnutí } \vec{v_1} \text{ na osu } \vec{l}, \\ \vec{n_2} &= \vec{v_1} - \vec{v_2}. \end{aligned}$$

Potřebujeme tedy zjistit kružnici, jejíž střed leží na ose válce, o poloměru r a procházející bodem průniku X:

Nechť $\vec{o'}$ je normalizovaný směrový vektor osy válce a nechť bod C_2 je střed

podstavy. Střed C' kružnice určíme jako

$$C' = C_2 - (\vec{o'} \cdot C_2) \cdot \vec{o'}.$$

Nyní musíme převést i paprsek do roviny kružnice. Neboli jej do ní promítnout. Nechť tedy paprsek má tvar $p : P_0 + t \cdot \vec{P_d}$. Nový paprsek $p' : P'_0 + t \cdot \vec{P_d}'$ vytvoříme takto:

$$\begin{aligned} P'_0 &= P_0 - (\vec{o'} \cdot P_0) \cdot \vec{o'}, \\ \vec{P'_d} &= \vec{P_d} - (\vec{o'} \cdot \vec{P_d}) \cdot \vec{o'}. \end{aligned}$$

Nyní budeme hledat průsečík nového paprsku p' s novou kružnicí o středu C' a poloměrem r. Ze vztahů

$$|P'_0 + t \cdot \vec{P'_d} - C'| = r,$$

$$(P'_0 + t \cdot \vec{P'_d} - C') \cdot (P'_0 + t \cdot \vec{P'_d} - C') - r^2 = 0$$

vznikne kvadratická rovnice

$$k \cdot t^2 + l \cdot t + m = 0,$$

s koeficienty k, l, m, kde

$$\begin{aligned} k &= \vec{P'_d} \cdot \vec{P'_d}, \\ l &= P'_0 \cdot \vec{P'_d} - 2 \cdot \vec{P'_d} \cdot C', \\ m &= (P'_0 - C')^2 - r^2, \\ t_{1,2} &= \frac{-l \pm \sqrt{l^2 - 4 \cdot k \cdot m}}{2 \cdot k}. \end{aligned}$$

Určíme oba body

$$X_{1,2} = P_0 + t_{1,2} \cdot \vec{P_d}$$

a pro každý bod X provedeme jeho projekci na osu válce a zjistíme, zda promítnutý bod leží ve vzdálenosti menší než h:

$$\vec{v_1} = X - C_2,$$

$$\cos \alpha = \frac{\vec{o'} \cdot \vec{v_1}}{\|\vec{o'}\| \cdot \|\vec{v_1}\|},$$

$$C_3 = \vec{o'} \cdot \cos \alpha \cdot \|\vec{o'}\| \cdot \|\vec{v_1}\|.$$

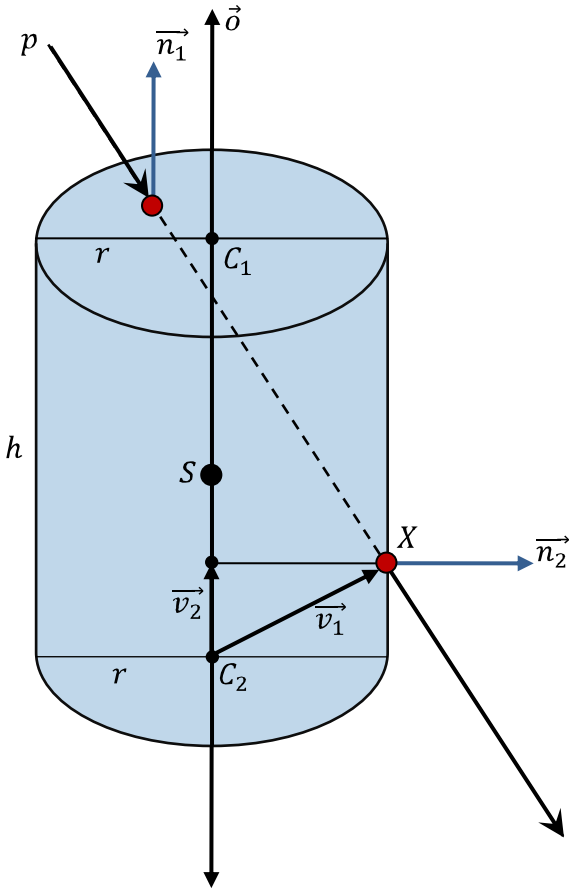
Aby bod ležel na plášti o zadané výšce h, musí platit

$$|C_3| \leq h \quad \text{a} \quad C_3 \cdot \vec{o'} > 0.$$

Normálu $\vec{n_2}$ pak tedy určíme, jak bylo popsáno výše:

$$\vec{v_2} = \vec{o'} \cdot (\vec{v_1} \cdot \vec{o'}),$$

$$\vec{n_2} = \vec{v_1} - \vec{v_2}.$$



Obrázek 10.: Průsečík paprsku s válcem

3. Jak na raytracing

Nyní je čas již představit samotnou podstatu této práce – algoritmus zpětného sledování paprsku – raytracing. Zjistíme, že tento algoritmus není příliš komplikovaný a zabere pouze několik řádků meta kódu, ale k jeho provedení je potřeba pochopit metody spolupracující s raytracingem. Tyto metody (výpočet stínů, osvětlovací model) nejsou závislé na raytracingu a jsou jen označením pro množinu algoritmů řešících daný problém. Podíváme-li se například na osvětlovací modely, zjistíme, že způsobů řešících tento problém je hned několik. Od spojitých empirických (Phongův model) přes fyzikální (Straussův model, Cook-Torrance model) až po konstantní modely. Nebo například problém vržených stínů lze řešit více způsoby. Klasický matematický (a přesný) model, mající však vysokou časovou složitost. Nebo opět empirický model, jenž je sice rychlý a dobře vypadající, ale realitě neodpovídající. Zde na těchto dvou příkladech problémů si můžeme všimnout jedné společné věci, a to, že pro oba existuje empirický model. Tento jev je v počítačové grafice poměrně častý a jeho existence má čistě pragmatické kořeny: přesné (fyzikální, matematické) modely jednak bývají o něco výpočetně složitější a jednak empirické modely jsou vytvořeny programátory pro použití programátorů. A programátor nemusí být fyzik, ani matematik...

Následně budeme postupně popisovat obecné prostředky potřebné pro algoritmus raytracingu s uvedením konkrétních případů.

3.1. Kamera

Abychom byli schopni zachytit vizuální obsah scény, potřebujeme samozřejmě zařízení, jež nám bude reprezentovat náš pohled neboli naše oko. Tímto zařízením je kamera (tzv. okno do světa scény). Základní atributy kamery jsou její rozlišení, poměr stran (aspect ratio), umístění, směr pohledu a směr vertikály. Rozlišení nemusíme příliš vysvětlovat, stačí se zmínit, že rozlišení kamery bude výsledné rozlišení obrázku a poměr stran v rozlišení by měl odpovídat poměru stran kamery.

Pro ostatní atributy si kameru prostě představme jako naše oko. Je potřeba jej umístit na pozici, zadat směr našeho pohledu a jeho kolmici – směr pohledu nahoru.

Význam kamery jsme uvedli pro člověka prostřednictvím oka, avšak pro raytracing kamera slouží jako generátor paprsků. Je to tedy počátek každého výpočtu, kdy raytracing už jen pracuje s vygenerovaným paprskem. Paprsek se generuje na základě zadaného rozlišení a všech ostatních parametrů kamery.

Standardní poměr stran kamery je 4:3 nebo 16:9. Známe-li poměr stran a maximální úhel mezi směrem kamery a horizontální hranicí obrázku, můžeme dopočítat druhou hranici obrázku: Nechť θ je úhel mezi směrem pohledu kamery a x-ovou hranicí obrázku (*zobrazovací úhel*). Pak hraniční body promítacího

plátna jsou:

$$\begin{aligned} X_{max} &= \tan\left(\frac{\pi}{180} \cdot \frac{\theta}{2}\right), \\ X_{min} &= -X_{max}, \\ Y_{max} &= \frac{X_{max}}{ratio}, \\ Y_{min} &= \frac{X_{min}}{ratio}. \end{aligned}$$

Zobrazovací úhel θ se v praxi volí podle typu kamery, konkrétně jejího objektivu. Čím výše má objektiv nastavenu ohniskovou vzdálenost, tím menší je zobrazovací úhel. Následující tabulka (1.) uvádí typické hodnoty pro kamery. V programu nastavujeme zobrazovací úhel pro poměr stran obrázku 16:9 na 80 ° a pro ostatní poměry na 60 °.

Typ objektivu	**Úhel (°)**	**Poměr stran**	**Ohnisková vzdálenost (mm)**
širokoúhlý	64 – 84	16:9	24 – 35
normální	40 – 62	4:3	36 – 60

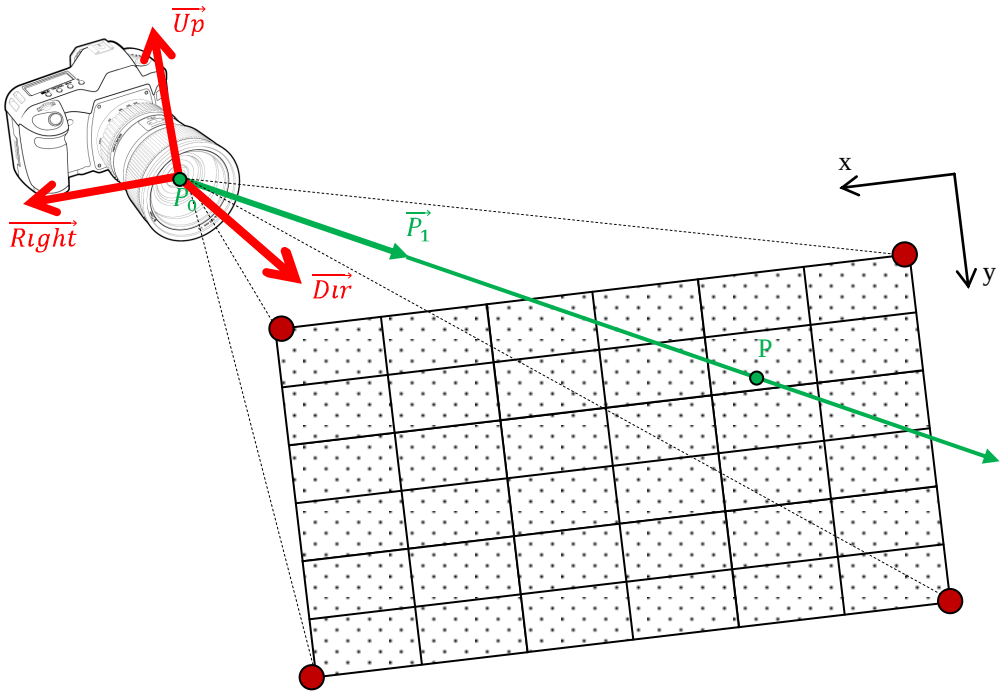
Tabulka 1.: Tabulka standardního nastavení objektivů kamery

Nyní známe ve světě scény hranice, v nichž generujeme paprsky. Zbývá zjistit složky onoho paprsku. Víme-li rozlišení obrázku (neboli roviny, na kterou promítáme), jež je v pixelech, potřebujeme jeho souřadnice převést do souřadnic světa, jenž je snímán kamerou. K tomu nám pomůžou 2 koeficienty pro každou souřadnici (aditivní a multiplikativní). Určíme je následovně:

$$\begin{aligned} xK &= \frac{\textit{šířka kamery}}{\textit{šířka obrázku}} = \frac{X_{max} - X_{min}}{width}, \\ xA &= X_{min}, \\ yK &= \frac{\textit{výška kamery}}{\textit{výška obrázku}} = \frac{Y_{max} - Y_{min}}{height}, \\ yA &= Y_{min}. \end{aligned}$$

Pro každý bod $[x, y]$ obrázku procházející od $[0, 0]$ po $[width, height]$ určíme body $[x', y']$ takové, že:

$$\begin{aligned} x' &= xA + xK \cdot x, \\ y' &= yA + yK \cdot y \end{aligned}$$



Obrázek 11.: Kamera – generátor paprsků

a vygenerujeme paprsek $p : P_0 + t \cdot \vec{P_1}$ z kamery o poloze C, jejím směru $\overrightarrow{Dir}$ a vektoru vzhůru $\overrightarrow{Up}$ následovně:

$$\begin{aligned} P_0 &= C, \\ \vec{P_1} &= \overrightarrow{Dir} + x' \cdot \overrightarrow{Right} + y' \cdot \overrightarrow{Down}, \end{aligned}$$

kde

$$\begin{aligned} \overrightarrow{Right} &= \overrightarrow{Up} \star \overrightarrow{Dir}, \\ \overrightarrow{Down} &= -\overrightarrow{Up}. \end{aligned}$$

Operátor $\star$ reprezentuje vektorový součin a operátor $\cdot$ násobení vektoru skalárem.

Pro ilustraci viz obrázek (11.).

3.2. Světlo

Fyzikálně je světlo rozděleno na dva druhy. Někteří jej zkoumají jako vlnové vyzařování, jiní jako částicové. Každé rozdělení je vhodné k popisu jiných jevů. Vlnové popisuje jevy jako je interference, difrakce apod. Částicové vidění světla zato popisuje odraz světla, chování v určitém optickém prostředí apod. To vše a mnoho jiného nám popisuje optika jako nauka o světle. Počítačovou grafiku však zajímají pouze dvě oblasti: *geometrická optika* a *fotonová optika*. Pomocí těchto modelů lze popsat většinu světelných jevů.

Nyní uvedeme několik zjednodušujících faktů o světle, s nimiž počítačová grafika pracuje:

1. světlo se šíří přímočaře,
2. rychlost světla je nekonečná – vše se jeví okamžitě,
3. světlo není ovlivněno gravitací, ani jinými čistě fyzikálními jevy.

K popisu jevů je potřeba sjednotit veličiny, s nimiž se bude pracovat. V optice je možno pro jeden jev použít více veličin, my však zvolíme jedny společné pro všechny, jelikož se z nich dají odvodit ostatní. Budeme tedy používat radiometrické veličiny.

Uvedeme některé základní radiometrické veličiny:

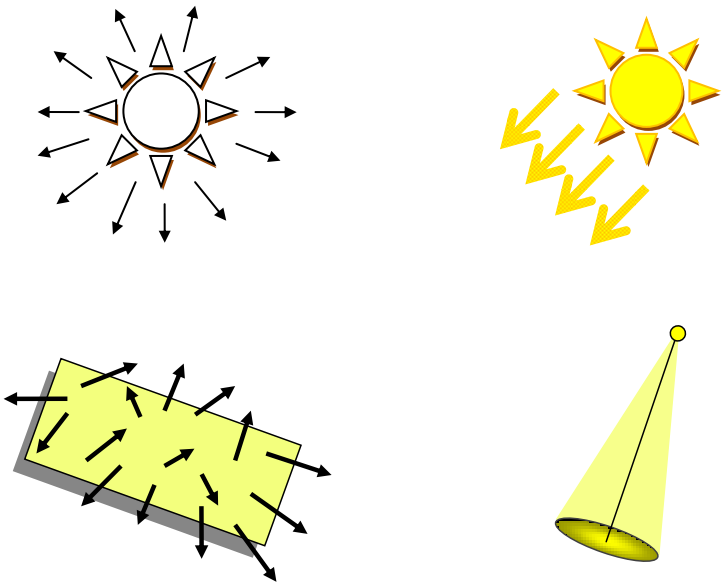
- iradiance – světelný tok dopadající na plochu,
- radiozita – světelný tok vyzářený plochou,
- zářivá intenzita – světelný tok proudící v určitém úhlu.

Avšak nejdůležitější radiometrickou veličinou je ***radiance***, s níž můžeme právě vyjádřit předešlé veličiny. Radiance je stručně řečeno to, co si představujeme (a co obrazovka počítače vnímá) pod pojmem barva paprsku. Víme, že lidské oko vnímá světlo z rozsahu určitých frekvencí. Tyto frekvence nazýváme úzké frekvenční pásmo elektromagnetického spektra a nachází se v oblasti 10^{14} Hz. Námi zkoumané světelné zdroje budou vyzařovat světlo v široké škále frekvencí. Takové světlo je pak okem interpretováno jako ***barva světla***.

Máme-li nyní představu o vztahu frekvence a barvy světla, můžeme se začít zabývat světelnými jevy obecně zkoumanými počítačovou grafikou. Zvláštní význam je zde pokládán chování světla při kontaktu s objektem. Po dopadu bílého světla na objekt se některé frekvence světla odrazí, jiné absorbují a další můžou projít tělesem skrz. Tyto 3 základní jevy se označují jako *odraz*, *difuze* a *lom*. Přitom barvou tělesa označujeme právě odražené frekvence světla. Typická reprezentace světla v počítačové grafice odpovídá jeho geometrickým a optickým vlastnostem, jež jsme si právě popsali. Světlo je tedy geometricky reprezentováno bodem a směrem, zatímco opticky trojskožkovým vektorem s hodnotami (většinou) modelu RGB.

Světlem ve scéně míníme přesně to, co se čtenáři vybaví pod světlem, s nímž se obyčejně setkává. Avšak typů světel je více, aniž by si to člověk uvědomoval. Máme světlo přímé (vyzařující od světelného zdroje) nebo světlo okolní (nezajímá nás, odkud se bere nebo kam dopadá, prostě existuje v každém bodě scény a všechny body osvětluje stejnou intenzitou).

Zatímco okolní neboli ambientní světlo je vždy právě jedno, světelných zdrojů může být více. Nejenom co do počtu, ale i co do typu. Nejjednodušším typem



Obrázek 12.: Typy světelných zdrojů. Horní řádek zleva: bodový a rovnoběžný. Spodní řádek: plošný a reflektor

světelného zdroje je bodové světlo. Je to světlo, jež svítí do všech směrů stejnou intenzitou a barvou a je určeno nekonečně malým bodem v prostoru udávajícím jeho polohu.

Jak již bylo zmíněno, u světla nás zajímá také jeho barva. Barvou myslíme trojsložkový vektor s hodnotami červená, zelená, modrá. Tato reprezentace se nazývá RGB-model.

Následuje výběr světel, u nichž má cenu se počítačovou grafikou zabývat (viz obrázek (12.)):

Bodové světlo jako jediné vždy vytváří ostré stíny (o stínech viz 3.5.).

Rovnoběžné světlo je rovnoběžný světelný zdroj sloužící pro reprezentaci velmi vzdálených (většinou umístěných v nekonečnu) světel jako je Slunce. Paprsky takového zdroje dopadají vždy rovnoběžně na každou plochu, tedy pod stejným úhlem.

Plošným zdrojem světla chápeme všechny paprsky vyzařující všemi směry z konečné roviny avšak pouze do jednoho poloprostoru. Takové světlo již může samo způsobit neostré stíny.

Reflektor je typ světla, jež je definováno jeho polohou a směrem záření. Přičemž největší intenzitu světlo vyzařuje právě na tomto vektoru a klesá expo-

nenciálně se vzdáleností ve směru kolmém na tuto hlavní osu. Geometricky lze reflektor popsat jako kužel. Typ reflektoru se dá kombinovat pro realističtější obrázky například umístěním dvou reflektorů do jednoho bodu, ale s různým poloměrem záření.

Ostatními reprezentacemi zdroje světel se nebudeme zabývat, jen uvedeme, že jsou to např. světelný zdroj definován tabulkou nebo zdroj typu obloha.

V dalším se bude využívat pouze bodový zdroj světla, jenž, jak již bylo řečeno, je nejjednodušší.

3.3. Objekty

O objektech ve scéně jsme již mluvili v první části. Byly popsány pouze 4 základní objekty, jež zkoumáme ve scéně. Jsou to koule, rovina, krychle a válec. Samozřejmě, že by se daly doplnit další objekty, ale jejich výpočty by byly o mnoho víc náročné a pro základní scénu 4 objekty bohatě stačí. I teď se dají postavit/poskládat zajímavá tělesa pomocí těchto stávajících, ale pro komplexnější objekty by bylo potřeba zavést zcela jinou reprezentaci, což je samotná oblast zájmu v počítačové grafice a lze se na ni zaměřit v případných dalších rozšířeních této práce.

Nyní využíváme matematickou reprezentaci objektů – jak již bylo ostatně předvedeno v první kapitole. Tedy každý bod objektu je definován matematickým vztahem.

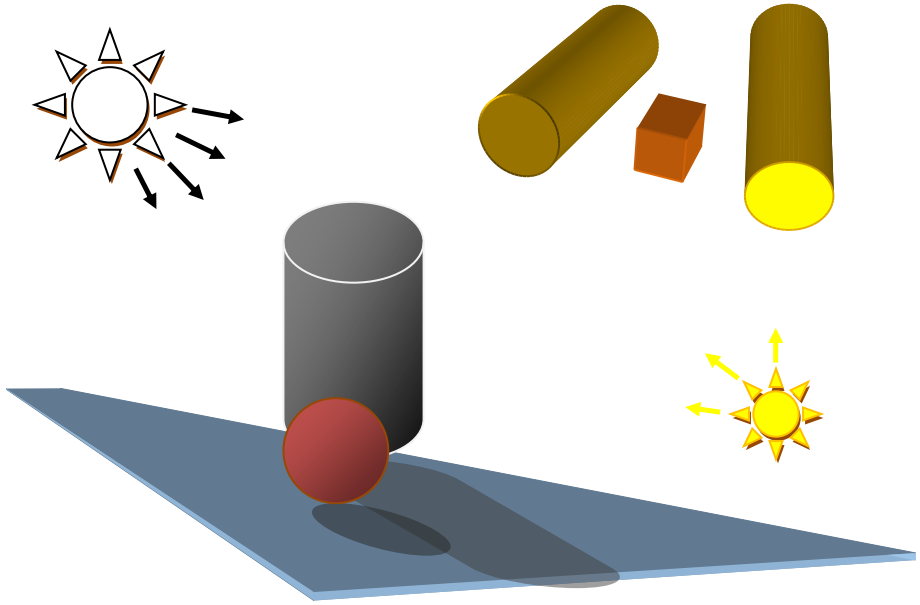
Zde uvedeme hlavní charakteristiky potřebné pro každý objekt. Ke každému objektu musí být známy jeho materiálové vlastnosti, jež jsou využívány při výpočtech v raytracingu. Všechny si vypíšeme a uvedeme později v této kapitole, avšak nyní již představíme základní vlastnost – barvu materiálu. Bez barvy by objekt nemohl být vůbec vykreslen a nešlo by na něm pozorovat jevy, jež nám raytracing zobrazuje.

Další velice významná vlastnost každého objektu: schopnost pro všechny jeho body ležící na povrchu objektu určit normálový vektor. Samozřejmě se výpočet normálového vektoru liší podle typu objektu. Dané výpočty normálových vektorů ke všem objektům ve scéně (koule, rovina, krychle, válec) lze najít v kapitole 2..

Ještě zmíníme jednu společnou vlastnost všech objektů: možnost výpočtu průsečíku s paprskem. Tento výpočet je jednou z nejdůležitějších součástí raytracingu a je potřebný v téměř každém jeho kroku. Opět byly všechny výpočty k objektům popsány v kapitole 2.2..

3.4. Scéna

Scéna sama o sobě není nijak významný pojem, představuje pouze označení pro všechny objekty, jež jsme doposud popsali. Tedy součástí každé scény je kamera, jež ji snímá, světla, jež osvětlují objekty ve scéně a nakonec samotné objekty



Obrázek 13.: Typická scéna

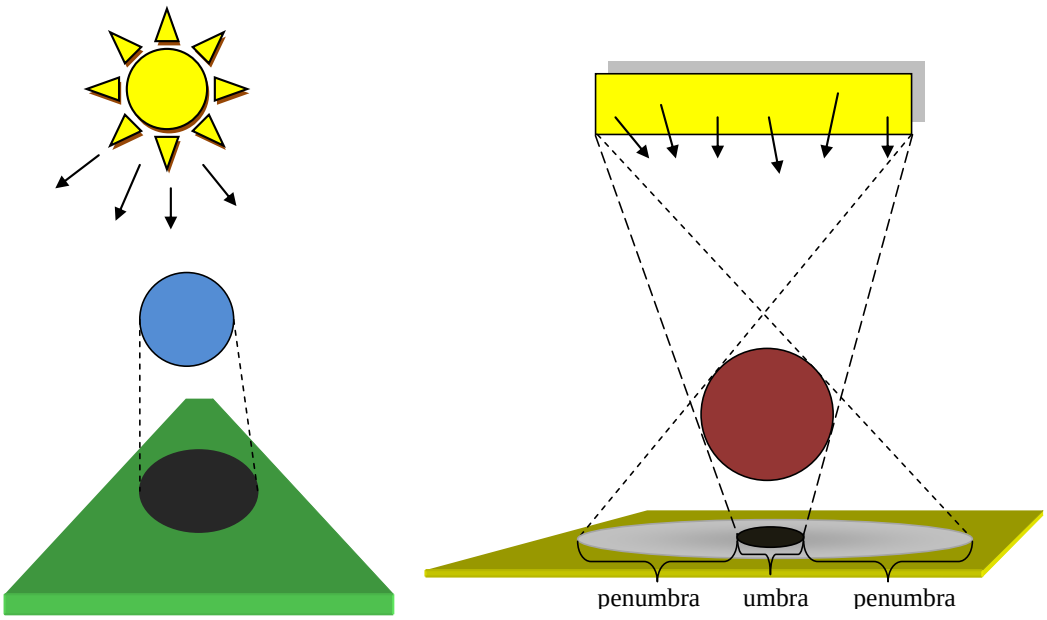
(koule, rovina, krychle, válec). Přičemž kamera je vždy právě jedna, zatímco množství světel a objektů jsou libovolné.

3.5. Stíny

Stíny neboli shadows je označením pro výpočet/vykreslení vrženého stínu. Jedná se o velice důležitý prvek při zobrazování scény, jelikož stíny poskytují detailnější přehled o prostorovém vjemu scény. Pomáhají při bližším pochopení rozmístění objektů ve scéně, jejich tvaru, rozměrů a také poloze světelných zdrojů.

Rozlišujeme dva základní druhy stínů – *vržené* a *vlastní*. Vržený stín vzniká, je-li jeden objekt zastíněn jiným objektem. Zato vlastní stín označuje zastínění objektu sebe samým (jeho součástí).

Tvar stínu závisí na stínícím objektu, zdroji světla a objektu, na který stín dopadá. Podle typu světelného zdroje můžou vznikat různé druhy stínů. Bodové světlo zapřičiňuje vzniku *ostrého stínu (hard shadow)*, zato plošné světelné zdroje způsobují tzv. *stín měkký (soft shadow)*. Ostré stíny ve skutečném světě neexistují a v počítačové grafice nepůsobí příliš věrohodně. Měkký stín se dělí na dvě složky: zcela zastíněná část neboli *umbra (hlavní stín)* a částenčně zastíněná část tzv. *penumbra (polostín)*. Polostín tvoří hranici přechodu mezi hlavní stínem a docela osvětlenou plochou. Se vzrůstající vzdáleností stínícího objektu od zastíněného bodu se velikost polostínu zvětšuje a velikost hlavního stínu zmenšuje



Obrázek 14.: Stíny: vlevo ostrý stín, vpravo měkký stín se zobrazenou umbrou i penumbrou

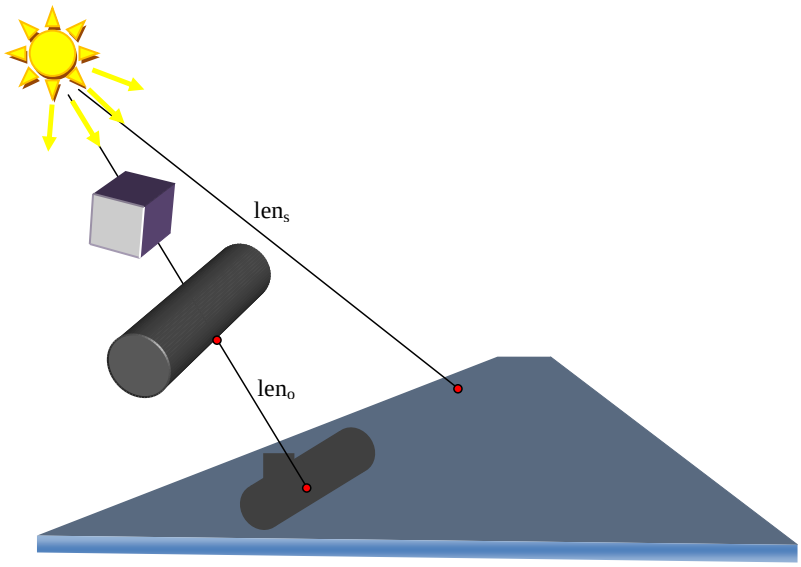
(jak je patrné z obrázku (14.)).

Nyní popíšeme získání bodového ostrého stínu. Základní metoda spočívá v tom, že víme, že každý objekt ležící mezi zdrojem světla a zkoumaným bodem, vrhá stín. Pro určení, zda je bod osvětlen nebo zastíněn, stačí, zda se od daného bodu podíváme směrem ke všem světlům ve scéně a pro každý tento směr zjistíme, zda se v něm nenachází nějaký objekt scény. Nestojí-li nic v cestě žádnému paprsku, nedopadá na bod stín a je tedy osvětlen daným světlem. V opačném případě se stín vypočítá podle zvoleného modelu.

Nejjednodušší případ – tzv. ostré stíny v přírodě (v reálném světě) se vůbec nevyskytující a vznikající pouze bodovým světlem – je prosté zastínění bodu přičtením barvy stínu (ambientní světlo) k bodu dopadu. Zjistí se vzdálenost zkoumaného bodu od zdroje světla (len_s) a vzdálenost zkoumaného bodu od nejbližšího stínícího objektu (len_o). Je-li potom $len_o < len_s$, pak je bod zastíněn. Pro představu situace je zde obrázek (15.).

V počítačové grafice existuje mnoho způsobů a algoritmů pro výpočet vržených i vlastních stínů. Jejich použití je omezeno reprezentací objektů ve scéně. Pro nejčastější reprezentaci objektů pomocí mikroplošek trojúhelníků nebo polygonů se výpočty stínů provádí algoritmy přes projekční metody (využívání transformací do roviny), algoritmus stínového tělesa a stínové paměti hloubky (podobný Z-bufferu). Těmito metodami pracujícími v objektovém prostoru se zde nebudeme zabývat, lze je najít např. v [1].

Ještě se podíváme na výpočty měkkých stínů. Existují dva typy těchto výpočtů: *single-pass* (pro jejž se používá i označení *single-sample*) a *multi-pass* (neboli *multi-sample*). Jak již bylo řečeno, vržené stíny jsou důležitou složkou



Obrázek 15.: Výpočet ostrého stínu

pro prostorovou orientaci ve scéně. Současně jejich výpočet nepatří mezi časově únosné složitosti. Obzvláště chceme-li, aby byl výsledek co nejrealističtější, což u měkkých stínů zcela jistě platí. Multi-pass algoritmy jsou zobecněním výše popsaného algoritmu pro vržený ostrý stín, avšak z každého světla vygenerujeme v jejich blízkém okolí množinu světel o stejné intenzitě. Existuje opět mnoho způsobů, jak světla vygenerovat, aby byl výsledek co nejrealističtější. Důležité je zvolit takový způsob, jenž rozdistribuuje světlo do svého okolí rovnoměrně s použitím co nejmenšího množství světel k tomu potřebných. Důvod je zřejmý: časová složitost. Jelikož pro každé nové světlo musíme zopakovat tentýž výpočet, neboli s ním jednat jako s plnohodnotným světlem. Nejlepší řešení se jeví normální rozdělení světelných zdrojů v okolí hlavního světla. Avšak zde přichází na řadu i empirické algoritmy distribuce světel:

Máme zadáno okolí světla (eps), do něhož generujeme světla, a počet světel, která chceme generovat (n). Uvádí se ([2]), že n by mělo být mocninou čísla 4 nebo 8.

Výpočet náhodných bodů provedeme pomocí sférických souřadnic:

1. vygenerujeme dvě reálná náhodná čísla u, v z intervalu $(0, 1)$,

2. pomocí nich určíme prostorové úhly ϕ a θ:

$$\begin{aligned} \phi &= 2 \cdot \pi \cdot u, \\ \theta &= \arccos(2 \cdot v - 1), \end{aligned}$$

alternativně $\theta = \arccos \sqrt{v}$,

3. určíme *delta* složky $[x, y, z]$ k původnímu světlu S o souřadnicích $[s_x, s_y, s_z]$:

$$\begin{aligned} x &= eps \cdot \cos\phi \cdot \sin\theta, \\ y &= eps \cdot \sin\phi \cdot \sin\theta, \\ z &= eps \cdot \cos\theta. \end{aligned}$$

Výsledné světlo S' bude mít souřadnice $[s_x + x, s_y + y, s_z + z]$. Ještě je potřeba původní intenzitu světla distribuovat mezi všechna nová. Tedy máme-li nyní $n+1$ světel, intenzita každého z nich bude mít velikost $\frac{1}{n+1}$ z intenzity původního světla.

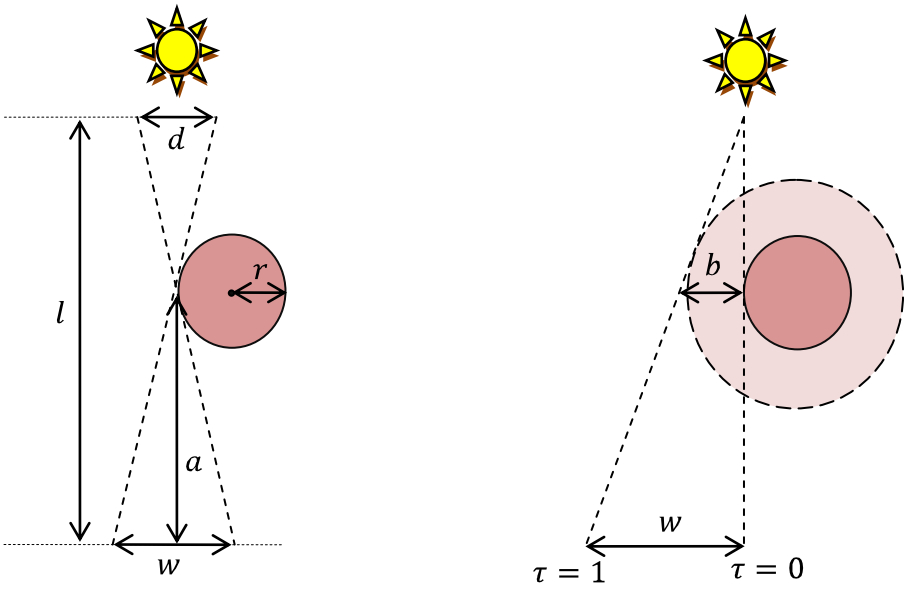
Nevýhodou tohoto algoritmu při výpočtu měkkých stínů je příliš vysoká časová složitost.

Metoda typu *single-pass* pracuje na zcela jiném principu. Jeho základem je empirické vypočítání velikosti složek penumbry i umbry vrženého stínu. Navíc tyto výpočty se liší podle stínícího objektu, takže existuje opět více způsobů určení stínu. Zde uvedeme výpočet single-pass pro kouli. Ještě před tím je třeba zmínit několik informací. Tak jak v předešlém algoritmu pro ostrý stín jsme potřebovali zjistit pouze nejbližší stínící objekt, jenž vrhá stín na zkoumaný bod, nyní potřebujeme v single-pass algoritmu zjistit všechny stínící objekty. Ke každému stínícímu objektu se vypočítá stínící hodnota a výsledkem bude vržený stín složený ze součtu stínových složek všech stínících objektů. Jak je z popisu patrné, samotný výpočet stínu se o něco málo zkomplikuje, jelikož bude potřeba vykonat větší počet numerických výpočtů, ale celkově je tento model výpočtu vrženého stínu efektivnější (uvádí se asi 20% navýšení časové složitosti oproti výpočtu ostrých stínů). Samozřejmě, že se mohou objevit situace, kdy zkoumaný bod zastíní více objektů, než by byl počet světel vygenerovaných v multi-pass algoritmu, a tedy by byl výpočet ještě pomalejší, ale to je pouze speciální velmi nepravěpodobný případ. Dále je třeba mít na paměti, že algoritmus je čistě empirický a nemusí odpovídat fyzikálním reáliím stínu.

Následuje stručný výpočet single-pass algoritmu pro vržený stín koule.

Kouli rozdělíme na dvě koule, jež budou vrhat stín. Vnitřní – původní koule – bude vrhat úplný stín. Vnější koule – zvětšená původní koule a od ní odečtená původní koule – bude vrhat částečný stín. Otázka je, jak určit velikost zvětšení koule. Obrázek (16.) osvětlí situaci.

Podle obrázku (16.) naše otázka, jak určit velikost neboli rozšíření (b) původní koule, závisí na šířce rozptylu světelného zdroje (d), vzdálenosti částečně zastíněného bodu od světelného zdroje (l) a vzdálenosti částečně zastíněného



Obrázek 16.: Výpočet měkkého stínu koule single-pass metodou

bodu od stínícího bodu objektu (a). Společným ukazatelem je hodnota w, udávající šíři částečného stínu. Tuto hodnotu lze aproximovat jako

$$w \approx \frac{a \cdot d}{l - a}.$$

Pak víme-li, že platí

$$\frac{b}{l - a} = \frac{w}{l},$$

vypočítáme b jako

$$b = \frac{a \cdot d}{l}.$$

Nutno ještě podotknout, že určením hodnoty b výpočet nekončí. Jak je na pravém obrázku (v 16.) znázorněno, je potřeba určit hodnotu $\tau \in \langle 0, 1 \rangle$, jež udává polohu zastíněného bodu v penumbře:

- $\tau = 1 \ldots$ bod již nezastíněn,
- $\tau = 0 \ldots$ bod v hlavním stínu.

Leží-li zastíněný bod v penumbra oblasti koule s poloměrem r, tj. platí-li:

$$r < d < r + b,$$

pak τ určíme jako

$$\tau = \frac{d - r}{b}.$$

Nakonec zbývá uvést tzv. *zeslabující funkci* s, jež nám určí onu intenzitu stínu. Tato funkce je sinusoida tvaru

$$s = \frac{1 + \sin(\pi \cdot \tau - \frac{\pi}{2})}{2},$$

jež se dá aproximovat Bernsteinovým interpolantem

$$s = 3\tau^2 - 2\tau^3,$$

který má stejné hodnoty i derivace v krajních bodech $\tau = 0$ i $\tau = 1$ jako uvedená sinusoida.

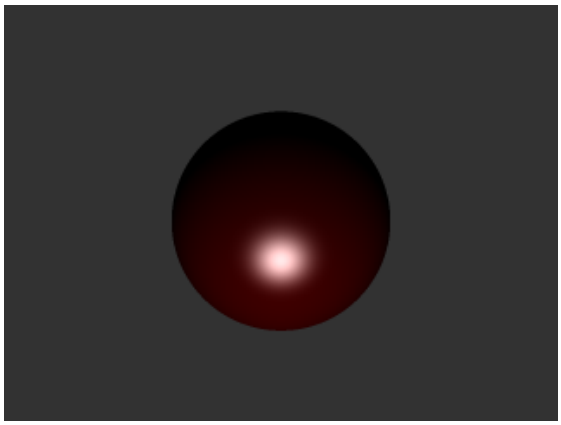
3.6. Osvětlovací model

Nyní je čas vysvětlit si metody a postup při výpočtu barvy nebo spíše osvětlení bodu na povrchu tělesa. K tomu budeme potřebovat znát některé základní pojmy z fyziky a optiky. Nebudeme zacházet do detailů, jen pro přehlednost uvedeme nejdůležitější definice. Nové poznatky si představíme na jediném zde popisovaném osvětlovacím modelu – Phongově. Ten je právě uváděn jako nejjednodušší a hlavně je empirický, takže nebude nutné zkoumat složité vzorce dopodrobna.

Nejdříve potřebujeme umět nějak obecně popsat jevy, jež vznikají po dopadu světla na povrch objektu. Zatím jsme se zhruba seznámili se světlem a jeho základními vlastnostmi. Nyní tyto poznatky využijeme při modelování případů interakce světla s objekty. Víme, že světlo z nějakého směru dopadá na bod a jiným směrem se od něj také odráží. To, zda se ovšem odráží a případně jak moc, budeme charakterizovat tzv. **Dvousměrovou odrazovou distribuční funkcí BRDF** (Bidirectional Reflectance Distribution Function). Ta definuje chování světla po dopadu na bod tělesa, respektive popisuje vlastnosti materiálu daného objektu. Nebudeme zde přesně definovat tuto funkci. Je poměrně složitá, ale popíšeme některé její vlastnosti:

- její hodnota je stejná po zaměnění vstupního a výstupního paprsku,
- funkce je vždy kladná nebo rovná 0,
- podléhá zákonu zachování energie – energie se nemůže sama vytvořit ani zaniknout, takže plocha nemůže odrazit více energie, než přijala,
- hodnota pro jeden vstupní úhel není ovlivněna hodnotou pro jiný vstupní úhel.

BRDF se definuje podle zvoleného modelu. Může být reprezentována buď tabulkou, nebo pomocí nějaké jednoduché funkce.



Obrázek 17.: Lesklý odraz na kouli

3.6.1. Lokální osvětlovací model

Nyní můžeme začít s metodami, jak určit výslednou barvu jednoho bodu na tělese po dopadu světla. Neboli zjišťujeme celkovou radianci (barvu), která vychází z bodu na základě všech vstupních radiancí. Tento problém řeší právě lokální osvětlovací model. Výsledný výpočet se často skládá ze součtů dílčích odrazů, jež vznikají po dopadu paprsku.
Rozlišujeme tyto základní druhy odrazu:

1. Okolní odraz (ambient)
   Vlastně se ani nejedná o odraz, ale o příspěvek barvy z okolního prostředí. Toto světlo si můžeme představit jako rozptýlené světlo všemi jevy, jež v okolí nastávají. Například vznikajícími neustálými odrazy světel. Často je reprezentováno konstantou a slouží především k zobrazení zastíněných prostorů jinou barvou než černou. Ke každému zkoumanému bodu přispívá tedy stejnou složkou.

2. Difúzní odraz (diffuse reflection)
   Difúzní povrch pohlcuje vstupní radianci a rozptyluje ji rovnoměrně do všech směrů. Hodnota odražené radiance není závislá na vstupním úhlu a je úměrná vstupní radianci. Tedy při pozorování difúzního povrchu bude ze všech úhlů vypadat pořád stejně. Výstupní radiance je to, co označujeme jako barva povrchu. Příklad absolutně difúzního materiálu jsou třeba guma nebo křída.

3. Zrcadlový odraz (specular reflection)

Tento odraz odpovídá dokonalému zrcadlovému obrazu. Tedy výstupní radiance bude souměrná podle normály v bodě dopadu se vstupní radiancí. Výstupní barva se může lišit od barvy tělesa, jelikož zde zobrazujeme odražené světlo. Příkladem buď sklo nebo lesklé povrchy kovů.

4. Lesklý odraz (glossy refleflection)
   Tento jev lze pozorovat na lesklých objektech a simuluje intenzitu světla pod daným úhlem ke světelnému zdroji. Je-li úhel nejmenší, je intenzita největší. S rostoucím úhlem intenzita klesá (viz obrázek (17.)).

5. Lom (specular refraction)
   Výpočet lomu paprsku jsme již předvedli v první části. Nyní popíšeme, jak k němu zhruba dochází. Víme, že lom paprsku vzniká při přechodu mezi prostředími s rozdílnou optickou hustotou. Paprsek je při dopadu na jejich hranici rozdělen na dva paprsky. Jeden se odráží již zmíněným postupem, druhý je lámán. Přičemž úhel lomu závisí na indexech lomu v obou prostředích a řídí se Snellovým zákonem. Index lomu lze chápat jako poměr rychlosti šíření světla ve vakuu a v daném prostředí. Tedy vzduch mívá hodnotu 1, sklo asi 1.6 a například diamant přes 2.0. Takové hodnoty větší než jedna jsou interpretovány jako šíření paprsku z prostředí řidšího do hustšího. Pro opačný případ bývá poměr menší než jedna a je třeba výsledný úhel omezit hraničním úhlem. Přes tuto hranici již neprojde žádné světlo a tedy se již neláme žádný paprsek. Dochází k takzvanému totálnímu odrazu. Pro hraniční úhel θ_{max} s indexem lomu původního prostředí μ_1 a nového prostředí μ_2 pak platí, že
   $$\sin\theta_{max} = \frac{\mu_1}{\mu_2}.$$

3.6.2. Phongův osvětlovací model

Nyní můžeme aplikovat již poznané jevy na konkrétní model. Tímto modelem bude již několikrát zmiňovaný empirický Phongův model. Jeho hlavní výhodou je, že je jednoduchý. K výpočtu osvětlení daného bodu budeme potřebovat znát dopadající paprsek, odražený paprsek, směr k pozorovateli a normálu v daném bodě. Všechny tyto položky jsou nutné znát ke třem hlavním výpočtům ve Phongově osvětlovacím modelu. Tyto výpočty slouží k popsání difúzního, zrcadlového a ambientního odrazu. Uvedeme označení složek při výpočtech:

- $\vec{u}$ ... směr vstupního paprsku světla,
- $I_{světla}$ ... barva zdroje světla pro vstupní paprsek,
- $\vec{v}$ ... směr od místa dopadu paprsku k pozorovateli,
- $\vec{r}$ ... odražený vektor,

- $\vec{n}$ ... normála v místě dopadu,
- ks ... koeficient zrcadlového odrazu ($ks \in \langle 0, 1 \rangle$),
- kd ... koeficient difúzního odrazu ($kd \in \langle 0, 1 \rangle$),
- ka ... koeficient ambientního odrazu ($ka \in \langle 0, 1 \rangle$),
- h ... odrazivý exponent ($h \in \mathbb{N}$).

1. Zrcadlový odraz I_s

   Barevná složka, jež zastupuje odraženou barvu, se vypočítá jako

   $$I_s = I_{světla} \cdot ks \cdot (\vec{v} \cdot \vec{r})^h,$$

   kde koeficienty ks i h jsou zadány v materiálu objektu, na němž zkoumáme intenzitu světla pro příchozí paprsek ze světelného zdroje. Exponent h vyjadřuje ostrost zrcadlového odrazu, viz obrázek koule (17.). S jeho hodnotou roste ostrost zrcadlového odlesku.

   Dále ověřujeme, zda může pozorovatel odraz světla vůbec vidět, tj. zda není ve stejném prostoru jako zdroj světla. Tedy pokud $(\vec{v} \cdot \vec{r}) < 0$, tak výsledná zrcadlová složka I_s bude rovna 0.

   Nakonec zdůrazníme, že množství zrcadlového odrazu je nepřímo úměrné odchylce mezi odrazem $\vec{r}$ a pohledem $\vec{v}$. Tedy čím blíže je odraz světla k pohledu pozorovatele, tím je zrcadlový odraz intenzivnější.

2. Difúzní odraz I_d

   Tento odraz je to, co vnímáme pod pojmem barva tělesa. Vztah

   $$I_d = I_{světla} \cdot kd \cdot (\vec{u} \cdot \vec{n})$$

   je vyjádřením Lambertova zákona, tedy že výsledná intenzita světla difúzního odrazu je nepřímo úměrná odchylce směru dopadu světla $\vec{u}$ a normály $\vec{n}$. A je-li $\vec{u} \cdot \vec{n} \leqq 0$, pak má výsledná difúzní složka I_d hodnotu také 0.

3. Ambientní odraz I_a

   Již jsme uvedli, že tato složka udává konstantní okolní světlo, jež simuluje například neustálé šíření světelných paprsků po scéně a tedy tzv. rozptýlené světlo I_A. Toto světlo bývá většinou bílé a konstantní pro celou scénu. Ambientní odraz potom vypočítáme jako

   $$I_a = I_A \cdot ka.$$

Koeficient ka mívá stejnou velikost jako koeficient kd a určuje, jak je těleso schopné odrážet okolní světlo.

Existence ambientního odrazu spočívá v tom, aby body objektů, na něž nedopadá žádné světlo, nebyly totálně tmavé a alespoň udávaly barvu tělesa se sníženou intenzitou.

Výslednou hodnotu celkové barvy světla I v místě dopadu získáme součtem:

$$I = I_s + I_d + I_a.$$

Tento výpočet popisuje základní metodu k výpočtu lokálního osvětlovacího modelu, kdy tedy neřešíme ani vzájemnou interakci objektů mezi sebou (stíny), ani odražené nebo lomené paprsky. Jak vypočítat hodnotu odraženého I_r a lomeného I_t paprsku se dozvíme v následující kapitole (3.7.3.), ale rozšířený Phongův vzorec intenzity barvy můžeme uvést již nyní. Budeme k tomu potřebovat znát další materiálové vlastnosti:

- kt ... koeficient lomu tělesa,
- n ... index lomu tělesa neboli nového prostředí pro paprsek.

Teď již můžeme vypočítat další dvě barevné složky:

4. Odražená složka I_r

   Jedná se o barvu, jež přinesl **odražený** paprsek od místa dopadu a byl vyslán směrem do scény. Složku I_r získáme jednoduše takto:

   $$I_r = ks \cdot I_R,$$

   kde ks je již známý koeficient odrazu a I_R intenzita barvy odraženého paprsku.

5. Lomená složka I_t

   Podobně jak byl vyslán do scény nový odražený paprsek a jeho výsledná barva použita do odražené složky, tak provedeme i s **lomeným** paprskem. Pomocí indexu lomu určíme lomený paprsek a vyšleme jej opět do scény. Ze získané intenzity I_T lomeného paprsku vypočítáme lomenou složku I_t:

   $$I_t = kt \cdot I_T.$$

   Koeficient lomu kt je opět reálné číslo z intervalu $\langle 0, 1\rangle$ udávající, jak moc těleso láme paprsky:

   - $kt = 0$ ... vůbec neláme,
   - $kt = 0.5$ ... láme napůl,

- $kt = 1 \ldots$ láme úplně.

Vzorec pro celkovou hodnotu barvy v místě dopadu ve Phongově modelu bude po přidání nových složek vypadat takto:

$$I = I_s + I_d + I_a + I_r + I_t.$$

3.7. Zobrazovací metody

Nyní, máme-li představu o zobrazení samostatných objektů ve scéně a metodách k tomu sloužících, je načase se podívat na metody, jež nám zobrazí celou scénu co možná nejrealističtěji. Tedy budeme hovořit o metodách (respektive jejich hledání a pozorováních) fotorealistického zobrazování, jejichž cílem je poskytnout co možná nejpřesnější počítačem vygenerovaný obraz, se snahou od skutečné fotografie co nejmenší rozlišitelnosti. Kromě samotného výsledného obrazu je požadována i co možná nejrychlejší metoda, jež poskytne simulaci reálných jevů v reálném čase.

Budeme vždy zvažovat situaci, v níž scéna obsahuje několik navzájem se ovlivňujících objektů (ať už světla nebo další tělesa s rozdílným materiálem). Pro popis vzájemné interakce objektů nebudeme používat formální zobrazovací rovnice, jejíchž řešení je v obecném případě analyticky nemožné. Pouze zmíníme, že dále uvedené metody k řešení osvětlovací rovnice konvergují.

Nejprve se seznámíme se základními optickými jevy, jež vznikají při přenosu světla a které tvoří více či méně podstatnou část k vytvoření fotorealistického obrazu.

1. Přímé osvětlení
   Nazýváno jako lokální osvětlovací model, kdy pozorujeme světlo po jednom odrazu od objektu. K jeho simulace je potřeba znát odrazivé vlastnosti materiálu společně s jeho barvou a dále charakteristiku světelného zdroje. S přímým osvětlením jsme se již setkali u Phongova osvětlovacího modelu.

2. Stín
   Již bylo dříve řečeno, že stín je důležitou složkou potřebnou k prostorovému vnímání scény. Bez stínů prakticky nejde vytvořit fotorealistický obraz, proto je na něj kladen velmi vysoký důraz. Pro vznik stínu (vrženého) jsou potřeba alespoň dva objekty ve scéně, kdy jeden leží v cestě paprskům dopadajícím ze světelného zdroje na druhý objekt.

3. Odrazy
   Na základě vlastností materiálů může vzniknout několik druhů odrazů světla. Každý z nich je potřeba pro realističtější výsledek zvážit zvlášť, proto uvedeme jejich typy:

(a) Zrcadlový odraz
vzniká u lesklých materiálů typu zrcadlo, ne zcela průhledné sklo a podobně.

(b) Difúzní nepřímý odraz
z Phongova modelu pro vysokou časovou náročnost simulace znám jako ambientní koeficient, jež je právě aproximací difúzního odrazu. Projevuje se pomalou změnou intenzity světla na vysoce difúzních površích.

(c) Difúzní přenos barvy
odraz z difúzního materiálu přenese část své barvy na blízké objekty, jež tímto jevem změní svou barvu přimícháním barvy odraženého světla.

(d) Kaustika
známá jako „prasátka“ vznikající při průchodu nebo odrazu paprsku zakřivenou plochou (i s odlišným indexem lomu) a projevující se jasnými shluky těchto paprsků ve stejných oblastech. Známé kaustiky vznikají například průnikem světla sklenicí vody nebo na dnech bazénů.

4. Opticky aktivní média
Jejich simulace patří k nejsložitějším a stále vyvíjeným metodám. Přenos světla je v těchto prostředích značně komplikovaný. Jako příklad uvedeme simulaci dýmu, kouře, mlhy apod.

Nyní se už můžeme podrobněji podívat na různé metody pozorování celé scény. Tyto metody se dělí na základní dva druhy podle typu pohledu. Nezávisí-li zobrazení na pohledu, jímž scénu pozorujeme, nazýváme metody **pohledově nezávislé** a jejich výsledkem je výpočet osvětlení všech ploch scény. V opačném případě při pozorování scény pouze z určitého směru pohledu hovoříme o **pohledově závislých** metodách.

Další metody globálního zobrazování se dají rozdělit podle toho, zda scénu během výpočtu pouze čtou, nebo ji zároveň i mění.

3.7.1. Metody Monte Carlo

Jak již bylo zmíněno v úvodu kapitoly, cílem globálních osvětlovacích metod je nalezení přesného řešení obecné osvětlovací rovnice, avšak analytické řešení je v naprosté většině případů nemožné. Hledáme tedy taková řešení, jež k přesnému řešení konvergují. Tyto metody souhrnně nazýváme metody *Monte Carlo* a pracují na bázi náhodného vzorkování, kdy řešením zadaného problému je střední hodnota náhodné proměnné. Tento postup využívá počítačová grafika v mnoha převážně fyzikálních modelech.

Vlastnosti Monte Carlo metod:

- jedná se o zástupce pohledově závislých řešení,
- lze je použít pro geometrické metody sledování paprsku,
- lze pracovat s celými objekty místo jejich dělení na menší části,
- snadný výpočet zrcadlového odrazu,
- nízká paměťová náročnost,
- přesnost zobrazení je úměrná velikosti rozlišení obrazu.

Nevýhodou Monte Carlo metod je jejich pomalá konvergence k přesnému řešení, kdy přesnost jejich řešení roste pro počet vzorků/paprsků n v pixelu pouze $\sqrt{n}$. Existuje mnoho způsobů, jak výpočet zefektivnit. Jedním z nejjednodušších je algoritmus Ruská ruleta, pracující na generování náhodných čísel v rozsahu $\langle 0, 1\rangle$ s rovnoměrným rozložením. Výpočet odraženého paprsku se následně řídí podle koeficientu odrazivosti objektu a vygenerovaného náhodného čísla. Je-li náhodné číslo větší než odrazivost objektu, výpočet paprsku vůbec neprovedeme.

3.7.2. Zobrazovací metody vycházející od světelného zdroje

Pro představu těchto metod uvedeme, že tyto metody bývají rovněž označovány jako metody vystřelující energii. Jak z názvu vyplývá, počátek vycházejících paprsků sledujících scénu bude umístěn ve světelných zdrojích. Po vypuštění paprsku ze světelného zdroje náhodně vybraným směrem se rekurzivně postupuje následovně:

1. sleduj trajektorii paprsku,
2. narazí-li paprsek na objekt, pak v průsečíku urči intenzitu jeho osvětlení a pošli množství světla směrem k pozorovateli,
3. pošli dál odražený (částečně náhodně) a případně i lomený paprsek.

Popíšeme, co se přesněji děje v kroku 2. Paprsek s intenzitou světla je poslán směrem k pozorovateli. Lépe řečeno směrem k promítací rovině, jež bude protnuta paprskem v nějakém bodě – definujícím výsledný pixel obrázku. Každý pixel si bude pamatovat počet příchodů takových paprsků a zároveň sčítat jejich nosnou barvu. Výslednou barvu pixelu zprůměrujeme ze všech příchozích paprsků. Tedy poslední hodnotu pixelu vydělíme počtem příchozích vzorků.

Nevýhodou této metody je velký počet vypočítaných vzorků, jež se ani nedostanou k pozorovateli (nějaký objekt jim stojí v cestě). A obecně problém těchto metod je vznik šumu, jelikož metoda nezaručuje, že se ke každému pixelu dostaví potřebný počet vzorků. Tudíž je potřeba generovat velké množství paprsků, což není příliš efektivní.

Snadnou modifikací metody sledování světla je v kroku 2. zjistit, zda je bod pro pozorovatele viditelný ještě před tím, než je vysláno světlo na projekční rovinu směrem k pozorovateli. Tedy určit, zda mezi bodem a pozorovatelem neleží v cestě jiný objekt, nebo zda není mimo pozorovací úhel. Není-li viditelný, vynechává se přispění intenzity barvy příslušnému pixelu.

Další úprava v Monte Carlo metodě je počítání útlumu paprsku po každém odrazu. Je-li intenzita paprsku již zanedbatelná, výpočet pro daný paprsek nepokračuje dál a skončí.

3.7.3. Zobrazovací metody vycházející od pozorovatele

Předchozí metoda sledovala paprsek od počátku ze zdroje světla, avšak nyní bude počátek trajektorie paprsků umístěn v pozorovateli. Můžeme tedy tuto metodu označit jako zpětné sledování světla nebo častěji se používá označení *zpětné sledování paprsku*. Paprsek při své cestě akumuluje světlo získané při odrazu a průniku objektů ve scéně, s nimiž se protne. Výhoda oproti předešlé metodě spočívá v tom, že zkoumáme pouze paprsky, jež dorazí k pozorovateli. U paprsků začínajících ve zdroji světla jich mnoho bylo zbytečně zkoumáno, aniž by se k pozorovateli vůbec dostaly.

Tedy základní algoritmus pro metody začínající u pozorovatele pro jeden vzorek v každém pixelu lze pozorovat v (Algoritmus 1) (opět, chceme-li ke každému pixelu určit více vzorků, pak výsledná barva pixelu se vypočítá jako průměr ze všech vzorků).

Algoritmus 1 Zobrazovací metoda směrem od pozorovatele

Inicializace: nastav hodnotu všech pixelů na barvu pozadí

Cyklus přes všechny pixely:

1. vypočti paprsek procházející pixelem od pozorovatele do scény,
2. sleduj paprsek a zjisti jeho výslednou barvu (algoritmem 2),
3. přiřaď výslednou barvu danému pixelu.

Mezi metody vycházející od pozorovatele patří především metoda sledování paprsku a metoda sledování cesty. Obě metody jsou si docela podobné. Metodě sledování paprsku neboli raytracingu se budeme věnovat v samostatné kapitole (3.7.5.). Zde uvedeme pouze stručně metodu sledování cesty a popíšeme její výhody a odlišnosti od sledování paprsku.

Zobrazovací metoda vycházející od pozorovatele způsobem sledování cesty neboli *pathtracing* používající vzorkování Monte Carlo se označuje taktéž jako

Monte Carlo sledování cesty. Výpočet řešení zobrazovací rovnice je navzdory vysokému výpočetnímu výkonu obohacen o metody, s nimiž se obyčejný raytracing nedokáže vyrovnat. Jedná se o výpočet při osvětlení jinými než bodovými světly nebo například zobrazení kaustik.

Algoritmus sledování cesty vysílá paprsky zcela náhodně od pozorovatele a sleduje jejich trajektorie:

Algoritmus 2 Sledování cesty paprsku

Vstup: paprsek, scéna

1. zjisti první průsečík paprsku ve scéně,
   (a) není-li žádný, vrať barvu pozadí scény;
2. v průsečíku vypočti stín a intenzitu barvy podle osvětlovacího modelu,
3. urči nový směr paprsku po odrazu a rekurzivně zjisti jeho barvu,
4. je-li materiál průhledný, vypočti lomený paprsek a rekurzivně zjisti jeho barvu,
5. vrať intenzitu barvy jako součet barev z kroků 2. 3. a 4.

Jelikož tato metoda vzorkuje na základě Monte Carlo, je trajektorie každého paprsku vždy odlišná a je potřeba vyslat pro každý pixel větší množství vzorků, jejichž průměrná hodnota intenzity barvy bude výsledná barva pixelu. Je zřejmé, že touto metodou vzniká šum, jenž se dá snížit jen vyšším počtem paprsků pro každý pixel. Z toho vyplývá opět vysoká časová složitost kvalitnějších obrázků. Navíc zde jistě nemá na šum vliv hloubka rekurze, ale počet vzorků vyslaných každým pixelem. Větší hloubka rekurze nemá přitom ani podstatný význam pro kvalitu vzorků. Uvádí se, že hloubka rekurze poskytuje kvalitní výsledky pro 5 až 10 odrazů ve složitých scénách. U jednodušších scén postačují poloviční hodnoty.

Ze způsobu výpočtu intenzity světla je zřejmé, že například jevy typu kaustika budou velice náročné co do výpočetního výkonu. Jelikož každá trajektorie paprsku je víceméně náhodná, bude nutné vygenerovat vysoké množství paprsků, které se náhodně trefí do stejného bodu a následně po odrazu do bodu světelného zdroje. U bodových světelných zdrojů je to téměř nemožné.

3.7.4. Dvousměrové metody

Právě jsme popsali dva přístupy, jimiž můžeme zkoumat scénu. Buď jsme sledovali světlo od jeho zdroje a při každém odrazu počítali intenzitu světla pro příslušný pixel (metoda vycházející od světelného zdroje), nebo jsme paprsek

vyslali od pozorovatele a postupně dle hloubky rekurze upřesňovali hodnotu barvy pro původní pixel (metoda vycházející od pozorovatele). Obě tyto metody mají své výhody a nevýhody a každá z nich se hodí pro jiný typ scén. Takže zůstává otázka, zda by se nedaly obě zkombinovat. Odpověď lze nalézt v dvousměrových metodách.

Tyto metody souběžně sledují dvě cesty – jednu od pozorovatele, druhou od světelného zdroje. Obě po vygenerování své dráhy pro jeden paprsek spojí tzv. *deterministickým krokem* navzájem všechny průsečíky obou cest, čímž se určí vzájemné příspěvky mezi danými kroky. Ovšem každá cesta nese jinou energetickou informaci. Směrem od pozorovatele zjišťujeme radianci a od světelného zdroje výkon. Je potřeba tedy převést jednu informaci na druhou a obráceně. Výhodou těchto metod je reálnější interpretace obrazu, načež samozřejmě roste výpočetní náročnost.

Takto popsaná metoda se nazývá *Dvousměrové sledování cesty*. Jako další metodu lze uvést *Fotonové mapy*, jež je navíc paměťově náročnější, ale zvláště se věnuje metodě na výpočet kaustik a podobných jevů.

3.7.5. Raytracing

Nyní se dostáváme k hlavní části této práce, algoritmu raytracingu. Vlastně podle výše uvedeného se následující algoritmus nebude příliš vymykat očekávání. Algoritmus patří do metod sledování paprsku vycházejícího od pozorovatele neboli takzvaného zpětného sledování paprsku. Celý výpočet nebude obsahovat žádná kouzla, ale pouze popis základních optických jevů jako je odraz, lom, atd. Již víme, jak vypočítat odražený i lomený paprsek, teď tyto výpočty využijeme.

Celý proces začíná u pozorovatele a promítací roviny – algoritmus 3. Pro každý pixel roviny vypočítáme paprsek vycházející od pozorovatele a procházející daným pixelem (námi již popsaný popis kamery alias funkce generátoru paprsků). Získaný paprsek vypustíme do scény a zkoumáme, zda mu bude stát v cestě nějaké těleso či ne a podle toho určíme výslednou barvu pixelu – algoritmus 4. Přičemž u každého tělesa ve scéně je potřeba znát jeho povrchové vlastnosti neboli tzv. materiál tělesa. Tyto vlastnosti jsou rozhodující při výpočtu další trajektorie paprsku. Uvedeme pro přehlednost dalšího textu základní pojmenování:

1. primární paprsek – paprsek vyslaný generátorem paprsků před prvním dopadem na těleso ve scéně,

2. sekundární paprsek – každý již odražený nebo lomený paprsek,

3. stínový paprsek – paprsek z místa průsečíku primárního nebo sekundárního paprsku s tělesem scény do světelného zdroje.

Po každém novém průsečíku primárního nebo sekundární paprsku s tělesem vznikají vždy maximálně 2 další sekundární paprsky – zrcadlově odražený

a lomený. Oba závisí na povrchových vlastnostech materiálu místa dopadu. Každý takto nově vytvořený paprsek pak pokračuje ve sledování scény, pokud nepřesáhl povolenou maximální hloubku rekurze. Ta bývá nastavena na hodnoty 3 až 5 (někdy až 7) podle složitosti scény. Avšak čím vyšší hloubka rekurze, tím samozřejmě roste časová náročnost výpočtu každého primárního paprsku a tedy i výsledného obrazu. Nastavení větší hloubky rekurze má význam, pokud scéna obsahuje spoustu zrcadlových nebo průhledných těles, kdy lze rozeznat detaily v odrazech. Počet detailů sice s hloubku roste, ale jejich velikost se zároveň snižuje a stávají se pro lidské oko již nepozorovatelné. Víme-li ještě, že hloubka rekurze zvyšuje čas výpočtu, tak ji není vážně nutné nastavovat příliš vysoko.

Stínový paprsek slouží k určení množství zastínění bodu. Tento výpočet probíhá tak, že od místa dopadu paprsku na těleso určíme stínové paprsky ke všem světelným zdrojům a zjistíme, které z nich bod přímo osvětlují (nestojí mezi bodem dopadu a světelným zdrojem žádné další těleso). K tomuto výpočtu je opět nutno použít algoritmus průsečíku paprsku se tělesem. Stačí nám přitom najít jediné takové těleso a můžeme světelný zdroj označit jako zakrytý. V tomto postupu se vyskytuje jeden docela podstatný nedostatek a vada základního raytracingu a to taková, že při testování existence stínícího tělesa se nebere v potaz, zda je těleso průhledné nebo jen poloprůhledné. Tedy vzniká efekt, kdy může skleněné těleso docela zastínit bod. K vyřešení tohoto problému existují rozšiřující modifikace základního algoritmu raytracingu, ale těmi se zde nebudeme zabývat.

Podobný problém vzniká u materiálů typu zrcadlo, jež se ve své podstatě chovají jako další zdroj světla, směřuje-li od zrcadla odražený paprsek přímo do světelného zdroje. Tento případ algoritmus raytracingu opět není schopen pochytit v jeho základní podobě.

Popsali jsme nyní slovně základní princip algoritmu zpětného sledování paprsku, nyní se můžeme podívat na metakód:

Algoritmus 3 Renderování obrázku raytracingem

Vstup: rozměry obrázku, maximální hloubka rekurze
Výstup: barvy všech pixelů obrázku
Cyklus přes všechny pixely:

1. pro daný pixel obrázku vypočti z generátoru paprsků paprsek *in*,
2. urči barvu pixelu metodou *Raytrace(in, 0)*.

Pokusíme si osvětlit situaci také obrázkem (18.).

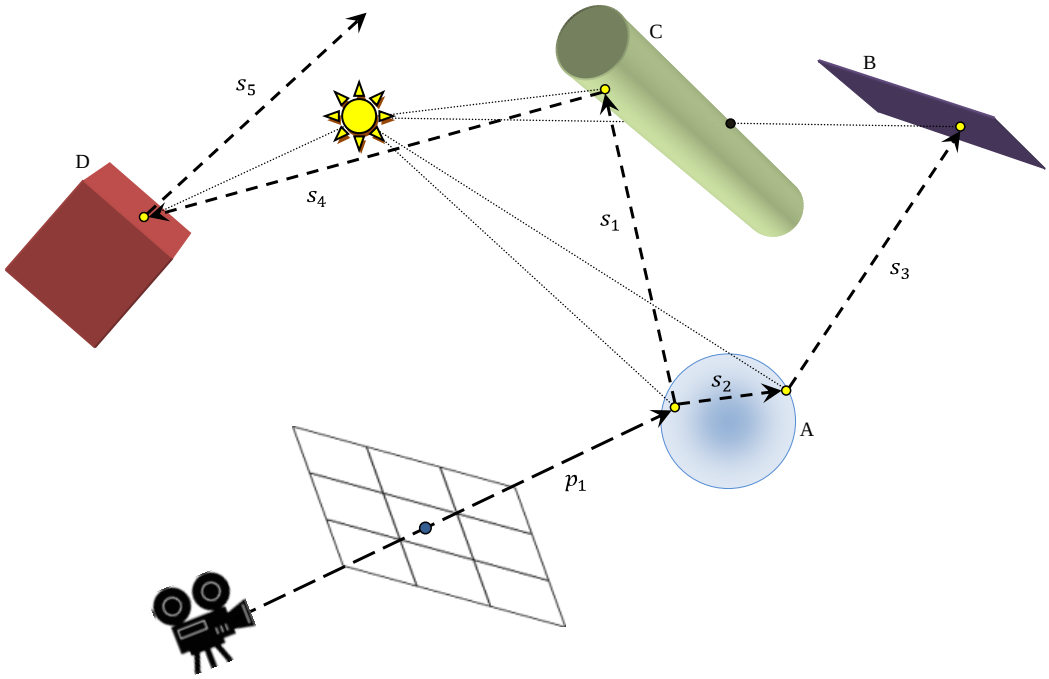
Nechť je nastavena maximální hloubka rekurze 5.

Vidíme, že kamera skrz pixel projekční roviny vygenerovala primární paprsek p_1, jenž nejdříve narazil na těleso A. Z prvního protnutého bodu najdeme všechny přímo dosažitelné světelné zdroje a na základě osvětlovacího modelu spočítáme

Algoritmus 4 Raytracing

Vstup: sledovaný paprsek in, aktuální hloubka rekurze h
Výstup: spočítaná barva paprsku
Raytrace(paprsek in, hloubka h):

1. vypočti průsečík P paprsku in s nejbližším tělesem ve scéně,
   (a) není-li průsečík, vrať barvu pozadí scény;
2. pomocí stínových paprsků zjisti všechny světelné zdroje L_i přímo osvětlující průsečík P,
3. ze všech L_i vypočti barvu b_1 lokálním osvětlovacím modelem v bodě P,
4. jestliže hloubka h nepřekročila maximální hloubku rekurze:
   (a) připouští-li to materiál, spočti odražený paprsek R_r a vyšli ho: $b_2 = Raytrace(R_r, h+1)$
   (b) připouští-li to materiál, spočti lomený paprsek R_t a vyšli ho: $b_3 = Raytrace(R_t, h+1)$
   (c) přičti k barvě b_1 také b_2 a b_3: $b_1 = b_1 + b_2 + b_3$
5. vrať barvu b_1



Obrázek 18.: Sledování paprsku od pozorovatele – raytracing

příspěvek barvy. Jelikož se nacházíme teprve ve hloubce rekurze 0, zkusíme vypočítat na základě materiálu objektu A nové paprsky. Toto těleso ovšem simuluje skleněnou kouli, takže příchozí paprsek se rozdělil na dva sekundární: odražený s_1 a lámaný s_2. Pro oba nové paprsky spustíme rekurzivně metodu na sledování paprsku s hloubkou 1, jejichž výsledky přičteme k prvně spočítané hodnotě barvy.

Nyní sledujme sekundární paprsek s_3. Vidíme, že protne objekt B simulující rovinu. Vypustíme-li stínový paprsek, zjistíme, že mu v trajektorii stojí objekt C. Tedy B leží ve stínu objektu C a žádné další světelné zdroje ve scéně nejsou. Budeme tedy dále počítat barvu v místě dopadu bez přispění světelného zdroje.

Nakonec ještě popíšeme případ pro sledování paprsku s_4. Tento narazí nejdříve na objekt D. V místě dopadu opět vyšleme stínový paprsek, vypočítáme osvětlení daného bodu a zkusíme vygenerovat odražený a lámaný paprsek. Materiál objektu D není průhledný, takže generujeme pouze odražený paprsek – s_5. Hloubka rekurze je nyní rovna 2, takže můžeme paprsek vyslat. Budeme-li jej nyní sledovat, zjistíme, že mu žádný objekt ve scéně nestojí v cestě a tedy metoda jeho sledování vrátí barvu pozadí scény. Byla-li by maximální hloubka rekurze nastavena na hodnotu 2, pak paprsek s_5 ani nevyšleme a rovnou vrátíme hodnotu barvy v dané hloubce.

Na závěr této kapitoly ještě zmíníme jedno rozdělení metody sledování paprsku. Představíme-li si právě popsanou metodu pro nastavení s maximální

hloubkou rekurze 0, máme metodu jež se nazývá *raycasting*. Jedná se o tzv. vržení paprsku a následné sledování paprsku prvního řádu. Tato metoda tedy pro každý paprsek buď nalezne průsečík s prvním tělesem a k němu spočítá výslednou intenzitu, načež skončí, nebo vrátí barvu pozadí scény. Kdežto algoritmus raytracingu se naproti tomu označuje jako sledování paprsku vyššího řádu.

3.8. Optimalizační metody

Hrubý algoritmus raytracingu, který jsme si právě popsali, je ve své podstatě velice časově náročný. To je způsobeno velkým množstvím operací, jež jsou potřeba v každém kroku vykonat. Uvádí se ale, že největší zátěž algoritmu přináší výpočty průsečíku paprsku s objekty, jež jsou používány ve dvou krocích algoritmu. Jednak při zjištění zkoumaného bodu, pro nějž budeme počítat osvětlení, a jednak při zkoumání stínového paprsku. Tedy body 1. a 2. algoritmu. Sice v druhém kroku nemusíme prohledat všechny tělesa, stačí nám první stínící. Uvádí se ([1, 6]), že tyto výpočty v základní podobě zaberou algoritmu 90% času. Dále se budeme zabývat způsoby, jež mohou pomoci urychlit výpočty prováděné při sledování paprsku. Zvýšit rychlost algoritmu je docela nasnadě, protože když uvážíme, kolik početních operací je potřeba pro každý pixel s hloubkou rekurze 5, tak se pro obrázky se složitější scénou, větším rozlišením a navíc s antialiasingem dostáváme k nežádoucímu času výpočtu. Avšak zavedením urychlovacích technik do algoritmu můžeme časovou složitost až řádově snížit. Rozlišujeme několik typů metod, jak algoritmus urychlit:

1. zrychlíme a zefektivníme výpočty průsečíků paprsků s tělesy:
   (a) rychlejší výpočty průsečíků,
   (b) menší počet výpočtů průsečíků,
2. snížení počtu sledovaných paprsků,
3. sledování více paprsků naráz.

Metody pro rychlejší výpočty paprsku s tělesem sice přinášejí úsporu času, avšak nijak zvlášť uspokojivou. Pracují na bázi, že než se začne skutečně analyticky počítat průnik paprsku s tělesem, provede se několik předvýpočtů a testů, které pomohou odhalit, zda-li paprsek má šanci těleso vůbec protnout. Tyto metody se liší podle typu objektu a jejich přispění do zrychlení v algoritmu je zřejmé, ale bohužel ne tak podstatné.

Mnohem efektivnější jsou ovšem metody na snížení počtu výpočtů při zkoumání bodu průniku paprsku s tělesem. Tato skupina metod je velice rozsáhlá a zabrala by samostatnou práci, proto jen nastíníme základní princip. Testování průsečíku tělesa, jak jsme jej zatím popsali, zůstane, avšak předtím budeme testovat paprsek s jednoduchým objektem, jenž ohraničuje každé těleso a jehož výpočet

průsečíku s paprskem bude velmi jednoduchý. Tato metoda bývá nazývána jako metoda obálek, avšak přesnější je anglický termín *bounding volume*. Nejprve pro každé těleso ve scéně vytvoří obálku, do níž jej uloží. Typicky se jako obálka volí rovnoosá krychle (*Axis-Aligned Bounding Box*), již jsme uváděli v první kapitole a jejíž výpočet průsečíku je opravdu velmi jednoduchý, obsahující jen pár početních operací sčítání a násobení a několik porovnání, nikoli žádné goniometrické funkce, jež jsou, jak víme, pro procesor nejsložitější. Máme-li každý objekt v takové obálce, testujeme nejprve pro daný paprsek, jestli neprotíná obálku. Když ano, otestujeme i těleso v ní. Je potřeba ještě podotknout, že obálka typu kvádr vůbec není jediná. Může být vytvořen podobně jednoduchý typ obálky pro každý druh objektů ve scéně, ke kterým se více bude obálka hodit. Obálka se ovšem musí k danému tělesu aspoň trošku přizpůsobit, aby zbytečně navíc neobklopovala i velké množství prázdného prostoru.

Vytvořením a použitím obálek, jak bylo popsáno, vzniká optimalizace algoritmu raytracingu, avšak ne taková, abychom již byli spokojeni. Máme-li pro každé těleso obálku, tak při testování průniku paprsku procházíme seznam obálek lineárně, což moc efektivitu nezvyšuje. Lepší způsob bude, když sestrojíme z obálek binární strom, pokud možno dokonale vyvážený. Strom lze konstruovat zdola i shora, my však popíšeme první variantu. V listech stromu se budou nacházet samotné jednotlivé obálky. Pak každý uzel vznikne sjednocením ze dvou uzlů nebo listů, jejichž obálky se buď překrývají, nebo jsou velmi blízko sebe. Tímto sjednocením vznikne nová obálka náležící novému uzlu stromu. Takto postupujeme dál, dokud nesestrojíme kořen stromu, jenž bude obsahovat jako své potomky postupně všechny již sestrojené obálky. Testování průsečíku paprsku s tělesem nyní bude probíhat procházením stromu od kořene a zjišťováním, s kterou obálkou příslušného uzlu se paprsek protíná. Máme již logaritmickou složitost testování paprsku s obálkami, oproti lineární.

Jiný přístup na zvýšení efektivity je rozdělení prostoru scény na menší části. Těchto rozdělení je několik druhů, nejčastější jsou dělení pravidelnou mřížkou, oktálovým stromem, BSP stromem a kD-stromem. Každé z těchto dělení má své výhody, jimiž se zde nebudeme detailně zabývat. Princip těchto algoritmů je nalezení všech prostorových buněk, jež jsou protnuty paprskem a tedy vrácení všech objektů, k těmto buňkám náležícím. Těmto metodám a jejich efektivní implementaci bylo věnováno značné výzkumné úsilí a proto patři k vysoce ceněným optimalizačním technikám. My zde nebudeme uvádět jejich podrobnosti.

Další způsob, jak snížit počet výpočtů průsečíků paprsku s tělesy, je metoda zvaná koherence paprsků, neboli využívání již dříve získaných informací v předchozích výpočtech. Pracuje tak, že vyšleme paprsek dvěma ne příliš od sebe vzdálenými pixely a podle jejich vypočítané barvy určit (interpolovat) i barvu pixelů mezi nimi. Přitom liší-li se hodnoty krajních pixelů víc, než je povoleno, zkoumáme prostor i mezi nimi. Tato metoda se nazývá *Pixel Selected Ray Tracing* a je tedy založena na záměrném snížení vzorkovací frekvence. Metoda se pro svou rychlost často používá při náhledu scény, kdy pozorovatele nezajímají

její detaily, ale například nastavuje kameru nebo tělesa ve scéně. Tento způsob optimalizace výpočtů může být použit i pro zpřesnění výsledků, při opačném způsobu jednání, neboli pro zvýšení kvality vyšším vzorkování. Zjistíme-li pro dva sousední pixely jejich výrazné barevné rozdíly, zvýšíme mezi nimi vzorkovací frekvenci a zpřesníme tím mezi nimi přechod barev. Tato metoda se nazývá *Adaptive Antialiasing*.

Uvedeme poslední metodu snížení časové složitosti raytracingu, jež nebude obsahovat žádné nové pojmy a navíc bude do značné míry intuitivní a přišel by na ní jistě každý sám. Popíšeme metodu řízení hloubky rekurze. Jak víme, paprsek se na základě povrchu tělesa odrazí nebo bude pohlcen. Takové chování závisí na zrcadlovém koeficientu materiálu tělesa. Je-li nenulové, paprsek se vždy odrazí. Je dobré teď přemýšlet nad tím, jak dlouho se může takový paprsek odrážet, nemáme-li příliš nízkou hloubku rekurze. Potenciálně takovým způsobem se paprsek může odrážet donekonečna tak dlouho, dokud bude narážet na odrazivá tělesa. Což ovšem ani moc neodpovídá realitě, zvážíme-li, že paprsek může postupně slábnout až vymizet úplně. Jak víme, paprsek s sebou nese intenzitu barvy. Když ji ovšem budeme po každém odrazu snižovat a testovat, zda má pro nás stále ještě nějakou informační hodnotu, značně tím můžeme snížit počet sledovaných sekundárních paprsků. Každá nová intenzita paprsku při odrazu se vynásobí zrcadlovým koeficientem (o němž víme, že je vždy v intervalu $\langle 0, 1 \rangle$). Obdobně můžeme postupovat i při lámání paprsku, čímž simulujeme útlum prostředí, do něhož paprsek přechází. Řízená hloubka rekurze není samozřejmě vhodná pro scény s velkým počtem zrcadlových materiálů.

Právě jsme popsali několik způsobů, jak zefektivnit algoritmus raytracingu. Vybrané metody lze mezi sebou samozřejmě kombinovat a dosáhnout tím velice příznivých výsledků oproti původnímu řešení. Největší příspěvek na zrychlení algoritmu lze přičíst metodám na dělení scény, ovšem za obětování více paměti při výpočtu.

3.9. Antialiasing

Při získávání hodnot barvy pixelů vysláním každým z nich právě jeden vzorek paprsku vzniká nepříjemný efekt pro nerovné povrchy – tzv. alias. Metod na odstranění aliasu (jež se označuje jako antialiasing) je spousta, my zde popíšeme pouze jednu základní, kdy přesuneme alias k vyšším frekvencím. Těmto metodám antialiasingu se říká *supersampling* a zde uvedeme jednu z nich – *FSAA* (Full Screen AntiAliasing). Supersampling pracuje tak, že daným pixelem tedy nepošleme jeden vzorek, ale více a výslednou hodnotu pixelu určíme podle všech těchto vzorků. Takto samozřejmě roste výpočetní výkon celého procesu generování obrázku. Volba rozmístění vzorků daného pixelu může být různá. Od pravidelného, přes náhodný až po jejich nejefektivnější kombinaci – tzv. *jittering*.

FSAA pracuje tak, že každý pixel rozdělíme na tzv. *superpixely*. Jejich rozlišení

může být různé (1x2, 2x2, 3x3) a tedy podle nich získáme i obraz v tolikrát větším rozlišení. Hodnotu každého pixelu pak určíme jako průměr z hodnot těchto jeho superpixelů. Tímto vznikne místo ostrého rušivého přechodu v obraze jeho rozmazání. Při počítání průměru přitom nemusíme všem superpixelům přiřazovat stejnou hodnotu, ale můžeme klást větší důraz třeba na prostřední superpixel.

4. Představení programu RayTracer

Nyní máme všechny potřebné vědomosti a informace k tomu, abychom mohli začít implementovat raytracing. Program s názvem RayTracer byl napsán v jazyce C# pod platformou .NET 3.5 ve vývojovém prostředí MS Visual Studio 2008. Program je rozdělen na dvě části – knihovnu implementující samotný raytracing (soubror *RayTracerLib.dll*) a formulář pro interakci programu s uživatelem.

Nejprve představíme základní UML diagramy programu a poté popíšeme nejdůležitější nebo nejzajímavější metody v programu. Samozřejmě přidáme pro ilustraci i výstupy z programu demonstrující vybrané metody z kapitol 2. a 3..

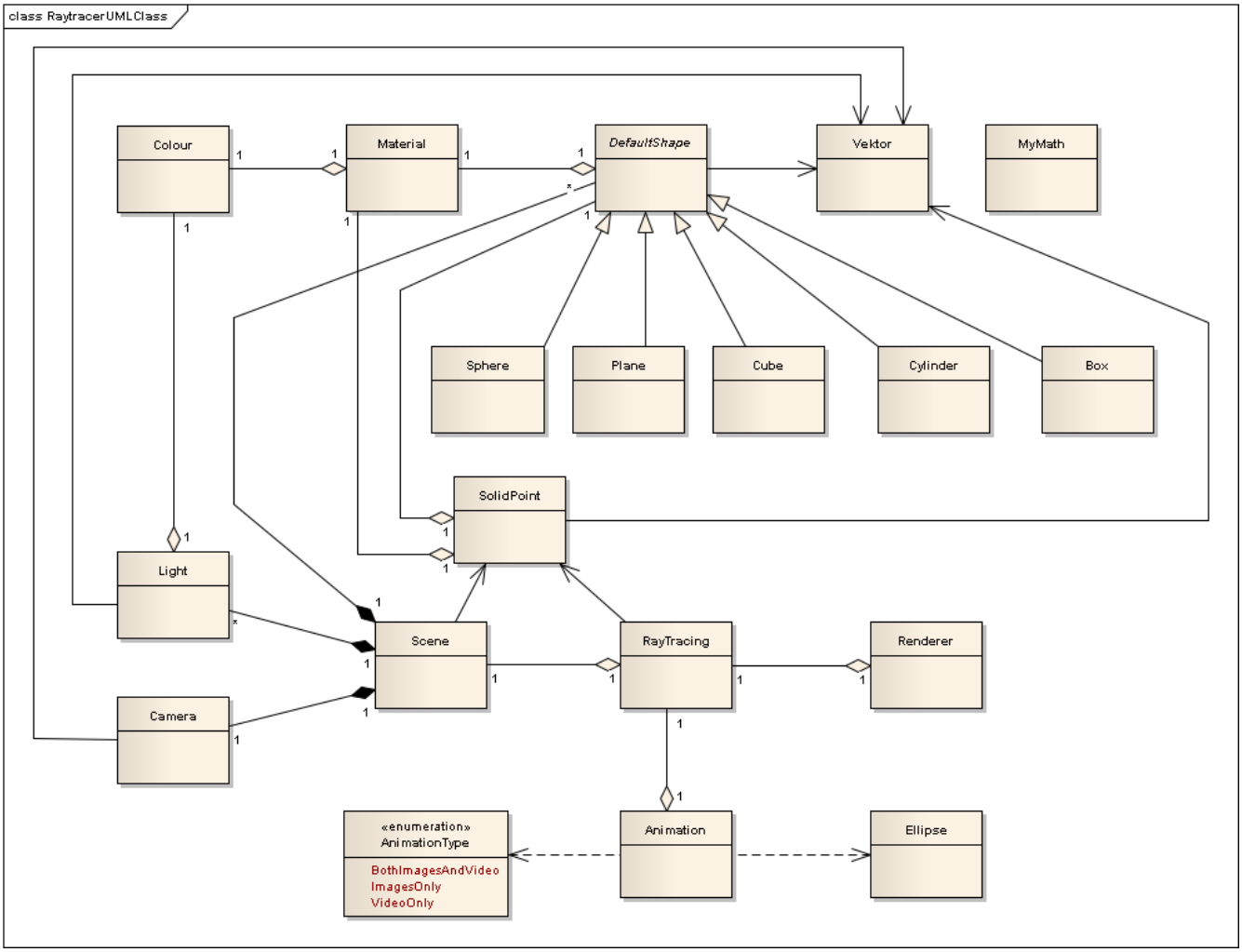
4.1. Struktura kódu

Prohlédneme-li si obrázek diagramu tříd na obrázku (19.) a připomeneme-li si názvy podkapitol z druhé části práce, zjistíme ne příliš překvapivé informace. Tedy že jsme se snažili každou podkapitolou reprezentovat jednu třídu z programu. Dále následuje popis tříd a jejich metod ve struktuře programu zdola nahoru.

1. *MyMath*
   Statická třída definující základní výpočty určené pouze pro raytracing. Počítá odražený i lomený paprsek, vzálenosti bodů mezi sebou a definuje nejmenší hodnotu EPSILON, jež se ještě nepočítá jako nula.

2. *Vektor*
   Je patrné, že když raytracing pracuje ve trojrozměrném světě, potřebujeme mít také takovou reprezentaci bodů z tohoto světa. Základní takovým prostředkem je třída *Vektor* se složkami X,Y,Z a implementující operace jako skalární součin, velikost vektoru, normalizace vektoru, vynásobení vektoru skalárem nebo přetypované operátory pro součet, rozdíl, vektorový součin (*cross product*) a (ne)rovnosti vektorů mezi sebou.

3. *Colour*
   Barvu objektu ve scéně definujeme opět vlastní třídou (všimněte si její britský název oproti jiné implementaci barvy v C#, jež má americký název barvy *Color*) se třemi hlavními složkami R,G,B a jednou zatím nevyužívanou Alfa složkou. Tedy, jak již bylo řečeno, pracujeme v RGB(A) modelu s každou složkou definovanou reálným číslem s hodnotami mezi 0 (černá) až 1 (bílá). Hlavní metody třídy jsou přetypované operátory pro sčítání barev mezi sebou. Dále je v programu často potřeba právě onen převod mezi objektem barvy z Raytraceru do barvy *System.Color*, takže



Obrázek 19.: UML diagram tříd

třída implementuje jak statickou metodu `ColourCreate()` pro konverzi objektu *Color* na objekt typu *Colour*, tak veřejnou metodu `SystemColor()` pro zpětný převod objektu na systémovou barvu.

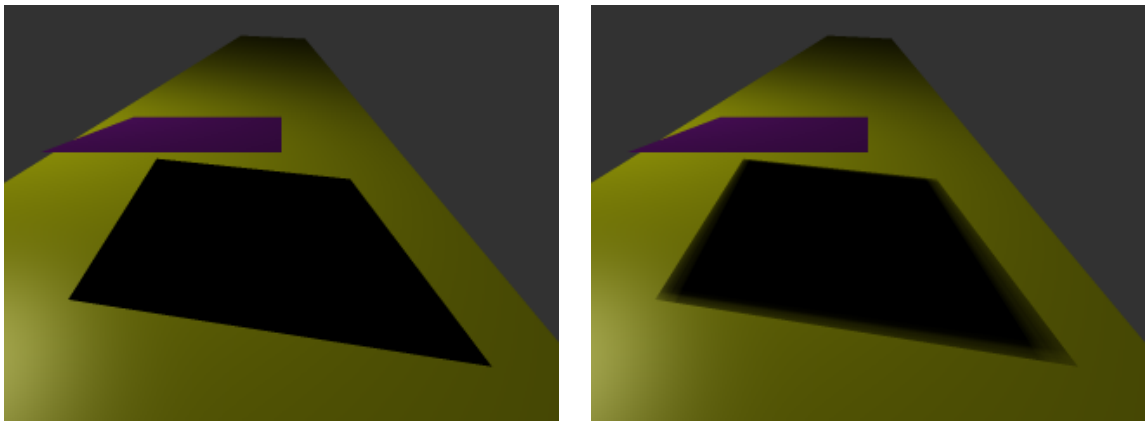
4. *Material*
   Udává materiálové vlastnosti každého objektu pro implementovaný osvětlovací model v raytracingu – Phongův model. Do materiálu patří jak barva, tak odrazivý koeficient (`Ks`), difúzní koeficient (`Kd`) a ambientní koeficient (`Ka`). Všechny tyto koeficienty by měly mít hodnotu v intervalu $\langle 0, 1 \rangle$, přičemž, jak víme z Phongova modelu, čím vyšší hodnota koeficientu, tím vyšší příspěvek v daném odrazu. Mezi další koeficienty patří celé nezáporné číslo `SpecularExponent` udávají lesklost materiálu a pak koeficienty pro lom: `KT` – z intervalu $\langle 0, 1 \rangle$ udávající, jak moc bude materiál lámat paprsek ($0 \ldots$ žádné lámání, $1 \ldots$ maximální lom) a index lomu `N`. Ve třídě jsou předdefinované vlastnosti materiálu pro nejtenčí sklo, zrcadlo a pro gumu.

5. *DefaultShape*
   Abstraktní třída pro reprezentaci vykreslovaných objektů ve scéně. Každá odvozená třída musí definovat vlastní materiál, zda bude ve scéně aktivní (tedy zobrazitelná) a operaci *Intersects* určující, zda a ve kterých bodech vstupní paprsek objekt protíná. Od této třídy jsou v současné verzi programu odvozeny objekty Koule, Roviny, Krychle a Válce, jejichž implementace operace `Intersects()` jsou podle metod uvedených v první části práce.

6. *SolidPoint*
   Popisuje bod průniku paprsku s objektem. Kromě přesných souřadnic také definuje parametr (neboli také vzdálenost od počátku paprsku), normálu v bodě průniku a jeho materiál. Navíc jsou ke třídě implementovány operace pro porovnání dvou takových bodů a vybrání aktuálně vyhovujícího.

7. *Camera*
   V druhé části dopodrobna popsaný generátor paprsků je přesně tak i implementován. Potřebuje být zadán bodem umístění, dopředným směrovým vektorem a vektorem určujícím směr nahoru. Dále obsahuje metodu na nastavení úhlu pohledu, poměru stran obrázku (neboli pro nastavení hraničních bodů projekční roviny) `SetAspectParams()` a hlavně generuje paprsek pro souřadnice projekční roviny v metodě `TryGetCameraRay()`.

8. *Light*
   Třída reprezentující jedno světlo. Přičemž toto světlo může být nastaveno buď jako obyčejné bodové nebo měkké. Měkké světlo zde bylo přidáno pouze pro experimentální účely a v dalších verzích a pokračování projektu je zamýšleno oddělit tyto dva typy světel od sebe zděděním od hlavního typu světla. Proto aktuální implementace světla obsahuje větší počet a složitěji



(a) Ostrý stín (b) Měkký stín multi-pass

Obrázek 20.: Vržené základní stíny a porovnání ostrého stínu s měkkým

definovaných metod, než by bylo pro klasické bodové světlo potřeba. Navíc implementace pro měkká světla jsou přítomny dvě. Jedna je klasická, kdy rovnoměrným rozložením vytvoříme z jednoho světla v okolí více světel a tedy při výpočtu osvětlení zvažujeme každé zvlášť – tzv. multi-pass metoda. Tento způsob výpočtu je značně neefektivní a už pro jednoduché scény je časově náročný. Pro srovnání na obrázku (20.) uvádíme 2 obrázky. Jeden (20.a) s klasickým bodovým světlem. Druhý (20.b)s měkkým světlem s nastavením: počet světel: 64, epsilon okolí: 0.5. Vygenerování obrázku s měkkým stínem trvalo přibližně 6 krát déle než prvního.

Tato metoda výpočtu měkkého stínu je sice časově náročnější, ale můžeme o ní říci, že je spolehlivá pro všechny scény nastavitelné v programu.

Druhá metoda na generování měkkých stínů – tzv. single-pass metoda – pracuje naprosto rozdílným způsobem. Nerozděluje jedno světlo na více světel, ale sama dopočítává intenzitu stínu, jak jsme se dočetli v kapitole 3.5.. Tato metoda je značně rychlejší, ale aktuální implementace počítá stín jen pro nejblížší těleso ke zkoumanému bodu, přičemž stín je potřeba spočítat ke všem stínícím objektům a výsledný stín z nich sečíst. Navíc se tento stín musí ke každému typu objektu počítat jiným způsobem. Tento výpočet překračuje stanovený rozsah práce a je ponechán na další zpracování. V současné verzi programu je single-pass výpočet měkkého stínu implementován pouze pro objekty typu koule. Pro ostatní objekty je nahrazen multi-pass metodou. Opět uvedeme na obrázku 21. porovnání s hrubým

stínem s parametrem epsilon okolí: 0.5 a pro multi-pass metodu se stejným epsilon okolím a parametr počtu světel je roven 64. Zde výpočet obrázku (21.b) s měkkým stínem metodou single-pass trval téměř stejně dlouho jako původní ostrý stín (21.a), zato měkký stín (21.c) multi-pass metodou přibližně 9 krát déle. Navíc je vidět, že single-pass metoda má jemnější přechody a i když je empirická, používá se častěji nejen pro zrychlení výpočtu.

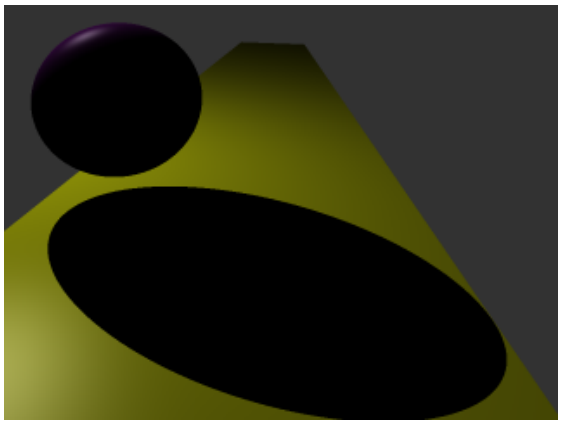
Nahoře obrázek základního vrženého stínu pro bodové světlo. Dole vlevo měkký stín single-pass metodou, dole vpravo měkký stín multi-pass metodou.

Základní světlo máme tedy v programu reprezentováno jeho souřadnicemi polohy, barvou a opět indikátorem `IsActive` aktivity ve scéně. Rozšíření pro měkká světla přidává ještě indikátor, zda je světlo měkké a jakou používá metodu: `IsSoftLight`, `IsSinglePass`. Dále pak potřebné atributy pro počet světel k multi-pass a epsilon okolí.

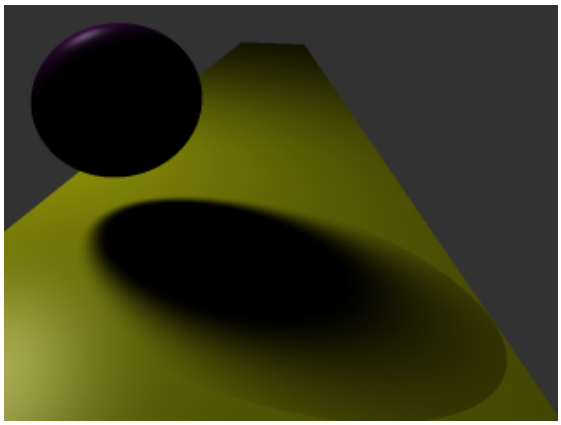
Nejdůležitější operací u objektů světla je zjištění, zda a případně jak moc je vstupní bod osvětlen. To vykonává metoda `Lightning()`, jež má několik verzí závislých na typu světla.

9. *Scene*
   Reprezentuje vykreslovanou scénu. Obsahuje seznam všech *DefaultShape* objektů ve scéně, seznam světel a kameru, jež scénu snímá. Navíc definuje i barvu pozadí scény a také její popisek pro zobrazení „děje“ scény v editoru. Její hlavní metody jsou vypočítání průniku paprsku s objekty ve scéně *SolidPoint* `GetIntersectPoint()` a zjištění osvětlení (nebo stínu) pro bod některého objektu: `Light[] GetAllLightningsToPoint()`.

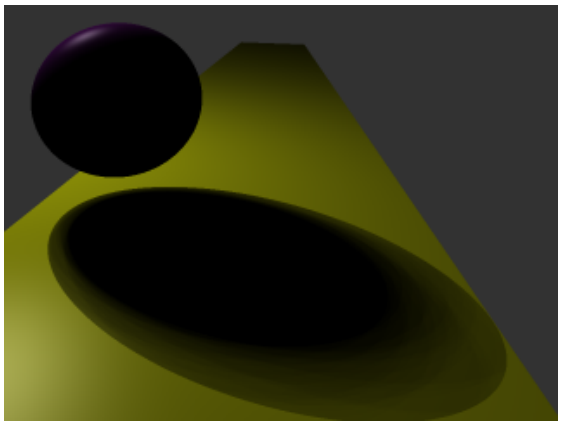
10. *Raytracing*
    Hlavní třída celého programu počítající již samotný raytracing. Ke své činnosti potřebuje znát hlavně scénu, s níž pracuje – atribut `Scene RScene`. Kromě nastavení projekční roviny a jejích okrajových bodů metodou `SetBoundValues()`, počítá výslednou hodnotu barvy pro zadaný pixel obrázku. Vzhledem k postupu vývoje aplikace jsou zde hlavní metody `Colour RayCast(double x, double y)` a `Colour RayTrace(int maxDepth, double x, double y)`. Jejich činnost je jistě zřejmá. Obě na začátku pro souřadnice pixelu obrázku vypočítají kamerou hlavní paprsek pro jeho sledování. Přičemž metoda pro raycasting nejprve ze scény zjistí nejbližší průsečík paprsku s objekty a všechna světla, jež na něj svítí. Pak z hodnot intenzit těchto světel vrátí pouze barvu osvětlení v tomto bodě. K tomu je určena metoda `Colour lightSum(SolidPoint sp, Vektor Pd, Light[] lights)`, jež pro bod, směr pohledu do kamery a světla na něj svítící podle osvětlovacího modelu spočítá hodnotu barvy.



(a) Ostrý stín



(b) Měkký stín single-pass



(c) Měkký stín multi-pass

Obrázek 21.: Porovnání všech metod vržených stínů

Metoda pro raytracing dělá zpočátku to samé, ale navíc přidává do výsledné barvy také intenzitu odraženého a případného lomeného paprsku, jež jsou zjištěny rekurzivním voláním metody `Colour DoRayTracing(int depth, ...)`.

11. *Renderer*
    Třída s funkcí pro vykreslení jednoho obrázku. Její činnost spočívá ve vyvolání raytracingu v novém vlákně na pozadí, jež může vracet průběžné výsledky vykreslování a vše podobně očekávatelné od procesu běžícího na pozadí.

    V konstruktoru *Renderer* dostane odkaz na objekt *Raytracing*, rozměry obrázku a indikátory pro hloubku rekurze a antialiasing. Start výpočtu provádí veřejná metoda `RenderAsyncImage()`, jež volá soukromou metodu `RenderImage()` vykonávající celý raytracing. Je-li zapnut antialiasing, pak jej metoda `AntializeArrayCols()` provede. V současné implementaci se pro antialiasing využívá supersampling 3x3 s váženým průměrem. Pro předčasné ukončení výpočtu slouží metoda `StopRendering()`.

    Na konci celého výpočtu je ve veřejné vlastnosti `Bitmap Bitmap` uložen výsledný obrázek. Ten je ovšem také předán v události vyvolávající dokončení výpočtu.

12. *Animation*
    Jako takový bonus k celé aplikaci je přítomna možnost vytvořit ze scény animaci. Jedná se o základní typ animace, kdy se po scéně pohybuje pouze kamera. Její dráha je definována provizorním způsobem po obvodu elipsoidu. Avšak zdaleka ne všemi směry. Kromě definování cesty kamery je samozřejmě potřeba zadat délku animace a požadovaný počet snímků za sekundu (fps). Výstup animace je ve formátu *avi*, přičemž lze nastavit i generování jednotlivých snímků z animace.

    Všechnu tuhle činnost zajišťuje třída *Animation*. Ta má v sobě vnořenou třídu pro definování dráhy, zde tedy zatím pouze třída *Ellipse*. Animace běží samozřejmě na pozadí a poskytuje možnost zobrazení průběžných snímků videa a odhadovanou dobu k dokončení procesu.

    Nejprve uvedeme, že v programu jsou přítomny dvě třídy pro animaci: *OneThreadAnimation* a *Animation*. Obě poskytují tentýž výsledek, avšak pro značné časové nároky základního raytracingu bylo nutné se vypořádat s výkonnostními obtížemi a náročným testováním, k čemuž slouží právě *OneThreadAnimation*. Původní třída *Animation* využívala k vykreslování jednotlivých obrázků třídu *Renderer* a pouze ukládala její výsledky do videa. Avšak, jak již bylo popsáno, *Renderer* pracuje na pozadí v novém vlákně, takže po spuštění animace již v hloubce vláken 3, což při testování a ladění není nejvhodnější. Takže třída *OneThreadAnimation* obsahuje

metody z třídy *Renderer* a všechny výpočty k animaci jsou prováděny pouze v jednom vlákně na pozadí.

Třída *Ellipse* pro popis dráhy kamery obsahuje pouze atributy elipsoidu: Střed a osy A,B,C. V metodě `List<Vektor> GetEllipsePoints(int count)` vrátí rovnoměrně od sebe vzdálený zadaný počet bodů na elipse – budoucích souřadnic polohy kamery. Bod $P = [x, y, z]$ na elipse se určí:

$$\begin{aligned} x &= Center_x + (A \cdot \cos\alpha \cdot \cos\beta - B \cdot \sin\alpha \cdot \sin\beta), \\ y &= Center_y + (B \cdot \cos\alpha \cdot \sin\beta - B \cdot \sin\alpha \cdot \cos\beta), \\ z &= Center_z + (C \cdot \sin\alpha), \end{aligned}$$

kde postupně posouvaný úhel $\alpha \in \langle 0, 2\pi \rangle$ a β je počáteční úhel. Potřebný počet bodů na elipse zjistí metoda `int ComputeNumberOfPoints(double fps, double time)` využívající vzorce pro obvod elipsy. Takto zjistíme všecha umístění kamery, která se předají animaci. Zbývá určit směr pohledu kamery, který budeme počítat jako směr z bodu na elipse do středu elipsy. Tento výpočet i zároveň posunutí do následujícího bodu kamery provádí metoda `bool MoveToNextImage()`. Ta, dokud kamera neobešla celou elipsu dokola, počítá její novou polohu a směr pohledu.

Animaci vytvoříme konstruktorem předáním odkazu na raytracer a dalších parametrů stejných jako v konstruktoru třídy *Renderer*. Navíc ale musíme nastavit atributy animace: `SetPath(Ellipse elipsa, double fps, double time)`. Pak již můžeme animaci spustit: `void StartAnimation(...)` zadáním rozměrů obrázku, hloubky rekurze, indikátoru antialiasingu, názvu výstupní animace a typem animace, jež je buď `BothImagesAndVideo`, `ImagesOnly`, `VideoOnly`. Animace se předčasně ukončuje metodou `StopAnimation()`.

Animace generuje postupně obrázky a přidává je do videa. K tomuto účelu bylo nutné importovat do programu knihovnu *Splicer.dll* využívající knihovnu *DirectShowLib-2005.dll*. Obě tyto knihovny musí být přítomny v hlavním adresáři programu. Avšak je zde jedno malé omezení pro rozlišení animace. Knihovna umožňuje vytvořit video pouze v určitých rozměrech. Ověřené velikosti jsou přednastavené v editoru, ale pro vlastní nastavení se doporučuje zvolit šířku videa na násobek čísla 4 a výšku na sudé číslo. Tato omezení jsou dána kodekem *DivX*.

Na závěr k animaci podotkneme, že i když se jedná o bonus celé aplikace, tak její vyřešení patřilo k těm nejkomplikovanějším úlohám celé práce.

4.2. Činnost programu

Program je formulářovou aplikací využívající knihovnu raytraceru popsanou v předešlé části. Samotný formulář je poměrně rozsáhlý, což je způsobeno

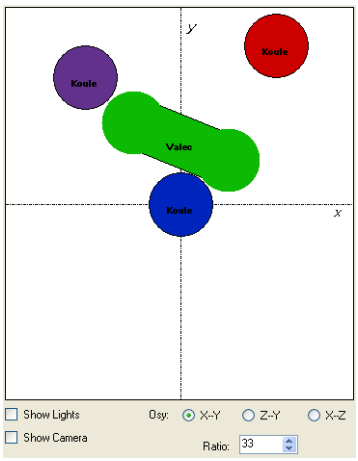
značným množstvím parametrů, jež je potřeba k raytracingu nastavit. Z toho důvodu bylo cílem implementovat uživatelské rozhraní co možná nejjednodušeji a nejintuitivněji. K tomuto účelu vznikl editor scény. Jedná se o klasický editor poskytující stručný pohled na scénu. Editor umožňuje zobrazit 3 úhly pohledu ve směru všech os, jak můžeme vidět na obrázku (22.). Tedy pohled zepředu (vidíme osy X a Y) lze pozorovat na obrázku (22.a), pohled z boku (vidíme osy Z a Y) viz obrázek (22.b) a pro pohled shora (vidíme osy X a Z) viz obrázek (22.c). Obrázek (22.d) ilustruje výsledek takto nastavené scény.

V editoru jsou zobrazitelné všechny aktivní objekty, kromě rovin. Zobrazení kamery či světel je volitelné. Kamera navíc zobrazuje šipku ze svého středu reprezentující směr pohledu. Příklad zobrazení kamery a světel v editoru viz obrázek (23.).

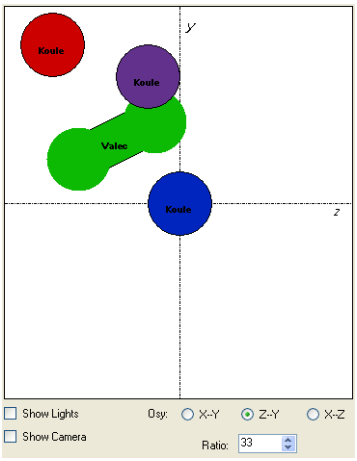
Editor navíc ještě umožňuje:

- s každým objektem lze pohybovat myší nebo přímo v nastavení přes seznam objektů,
- v obrázcích editoru vidíme i *Ratio* pro přibližování nebo oddalování pohledu,
- editor obsahuje několik přednastavených scén. Je to zvolený kompromis vzhledem k tomu, že nelze do programu zatím importovat externí scénu. Tato funkčnost je naplánována v pozdějších verzích pro formát XML. Uživatel si tedy bude moci vytvořit vlastní scénu, uložit ji a později ji kdykoli opět načíst.

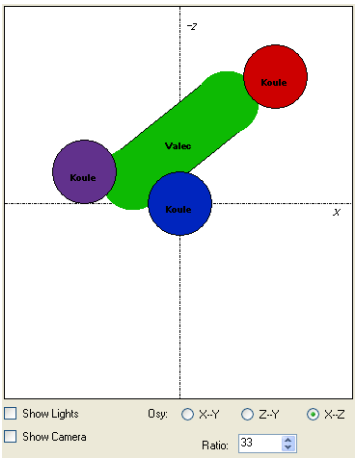
Pro představu funkcí uživatelského rozhraní následuje Use case diagram programu (obrázky 24. a 25.).



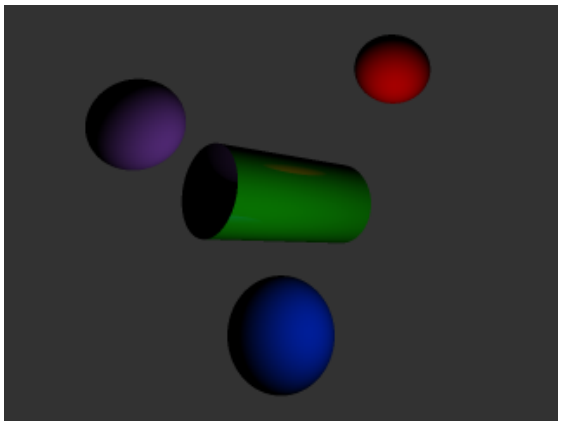
(a) Editor os X-Y



(b) Editor os Z-Y

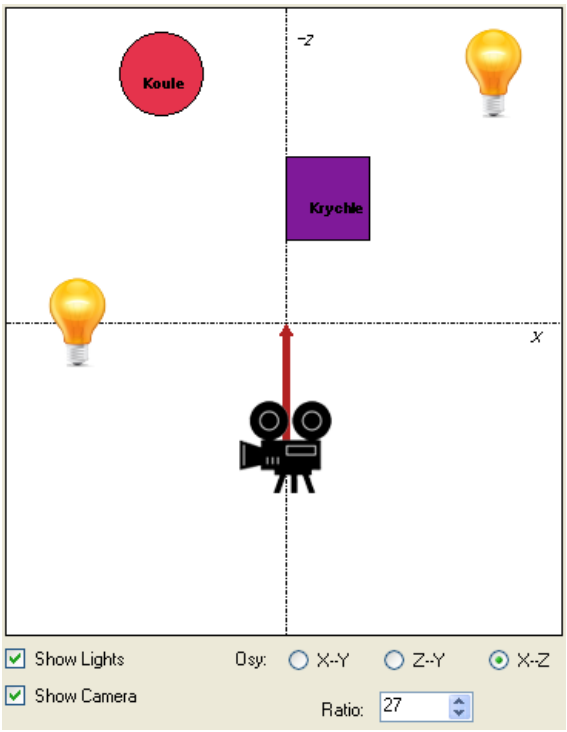


(c) Editor os X-Z

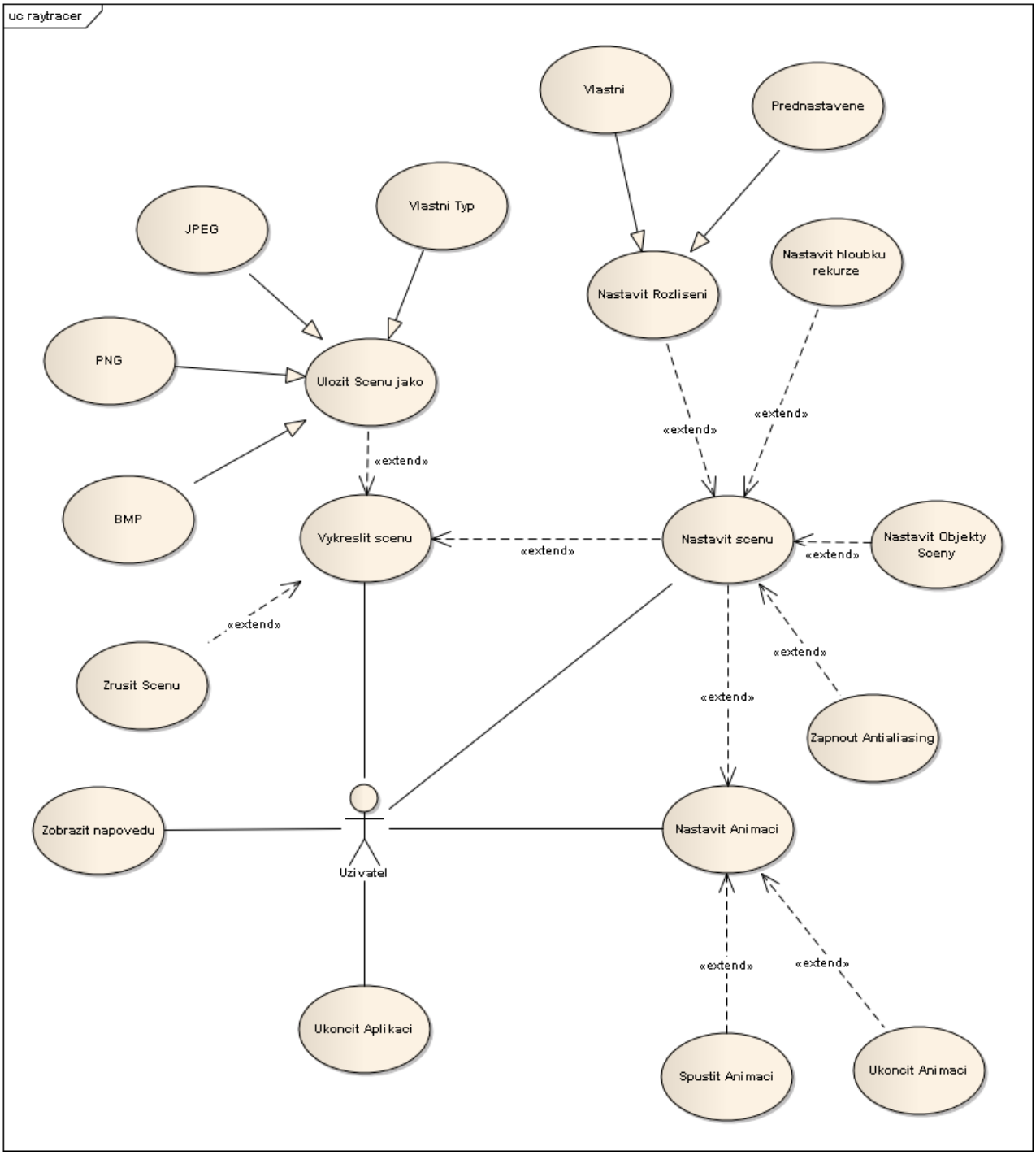


(d) Výstupní obrázek nasnímaný kamerou ve směru osy Z

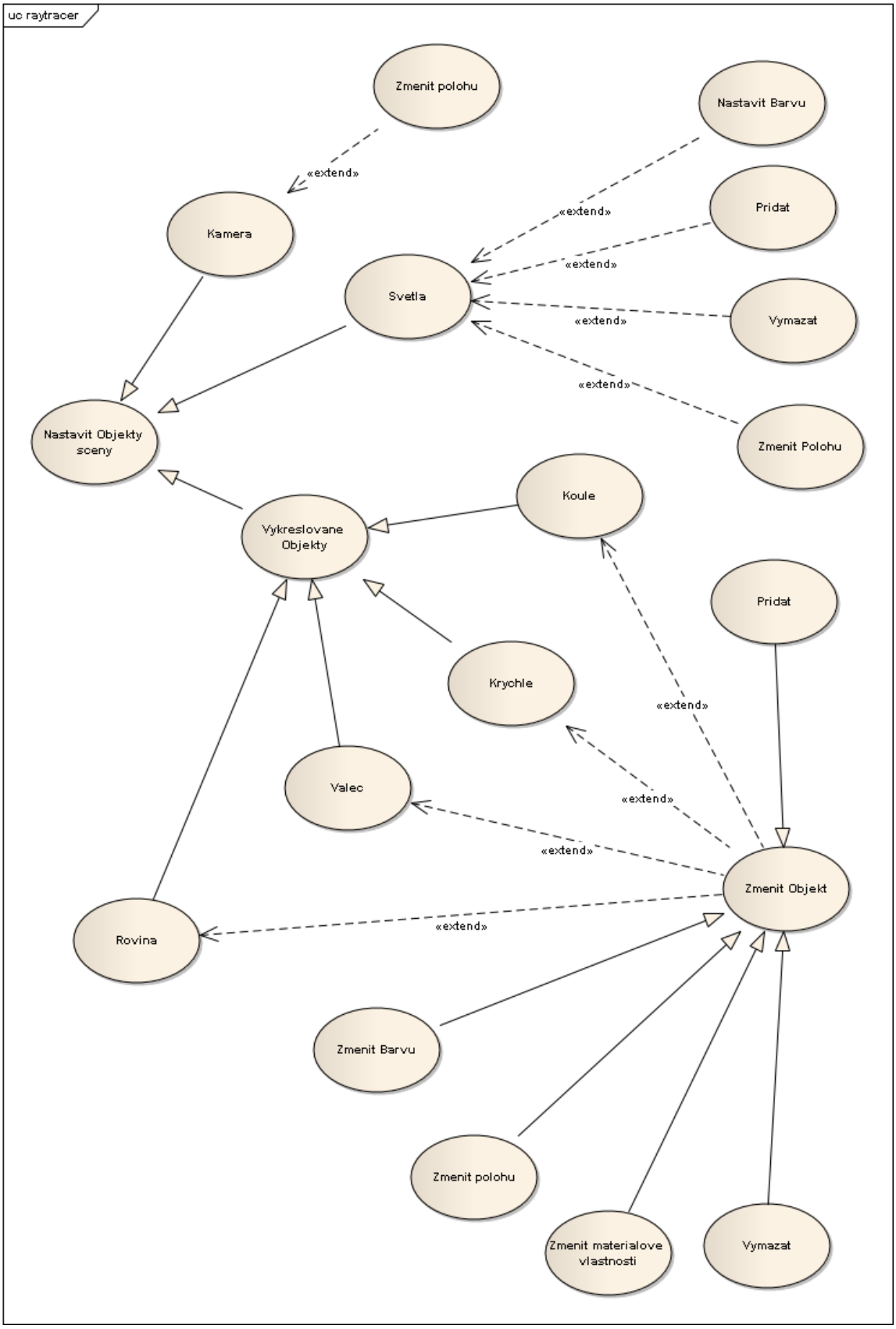
Obrázek 22.: Editor scény programu RayTracer s výstupem



Obrázek 23.: Editor s kamerou i světly



Obrázek 24.: Use case diagram 1.



Obrázek 25.: Use case diagram 2.

Závěr

Cílem práce bylo představit základní techniky fotorealistického zobrazování a implementovat získané poznatky v programu. Rozsah tohoto tématu však zabraňuje kompletní analýze metod a proto byly většinou uvedeny ve stručné podobě. Stejně tak i výsledný program RayTracer obsahuje implementace jen malého množství těchto metod. Nabízí se tedy otázka, zdali se věnovat tomuto tématu i během mého dalšího studia.

Conclusions

The aim of the bachelor thesis was to show basic techniques of fotorealistic rendering and to implement obtained knowledge in a program. Due to extensiveness of this topic, it was almost impossible to describe all methods in detail, therefore we mostly introduced them in a very brief form. Also the resulting program RayTracer just provides a short range of these methods. Hence, there is a question on a continuation of the topic in my further study.

Reference

[1] J. Žára, B. Beneš, J. Sochor, P. Felkel: *Moderní počítačová grafika*, Computer Press, Brno 2004, ISBN 80-251-0454-0.

[2] R. S. Ferguson: *Practical Algorithms for 3D Computer Graphics*, A. K. Peters, Ltd., 2001, ISBN 1-56881-154-3.

[3] D. H. Eberly: *3D Game Engine Design: A Practical Approach to Real-Time Computer Graphics*, Morgan Kaumann Publishers, Academic Press, 2000, ISBN 1558605932.

[4] F. Dunn, I. Parberry: *3D Math Primer for graphics nad Game Development*, Wordware Publishing, 2002, ISBN 1-55622-911-9.

[5] D. Martíšek: *Matematické principy grafických systémů*, Littera, 2002, ISBN 80-85763-19-2.

[6] *Ray Tracing*
Last changed July 20, 1999, G. Scott Owen.

[7] *Aliasing Problems and Anti-AliasingTechniques*
Last changed October 04, 1999, G. Scott Owen.

[8] *Illumination Models*
Last changed September 06, 1999, G. Scott Owen.

[9] *Lighting in Computer Graphics*
Last changed November 02, 1998, G. Scott Owen.

[10] *Monte Carlo Methods for Computer Graphics*
Last changed April 01, 1998, G. Scott Owen.

[11] P. Bourke: *Geometry, Surfaces, Curves, Polyhedra*.

[12] SoftSurfer.com *Algorithms*.

[13] www.raytracingnews.org *Ray Tracing News Guide*.

[14] S. Parker, P. Shirley, B. Smits: University of Utah, 1998, *Single Sample Soft Shadows*.

[15] *The Online POV-Ray Tutorial*.

[16] C. Balázs: METDST 1997 *Monte Carlo Light Tracing*.

[17] *Splicer for .Net*.

A. Obsah přiloženého DVD

V samotném závěru práce je uveden stručný popis obsahu přiloženého DVD, tj. závazné adresářové struktury, důležitých souborů apod.

`bin/`
: Instalátor programu RayTracer a další programy spustitelné přímo z CD/DVD. Adresář obsahuje i všechny potřebné knihovny a další soubory pro bezproblémové spuštění programu.

`doc/`
: Dokumentace práce ve formátu PDF, vytvořená dle závazného stylu KI PřF pro diplomové práce, včetně všech příloh, a všechny soubory nutné pro bezproblémové vygenerování PDF souboru dokumentace (v ZIP archivu), tj. zdrojový text dokumentace, vložené obrázky, apod.

`src/`
: Kompletní zdrojové texty programu RayTracer se všemi potřebnými (převzatými) zdrojovými texty, knihovnami a dalšími soubory pro bezproblémové vytvoření spustitelných verzí programu / adresářové struktury pro zkopírování na webový server (v ZIP archivu).

`readme.txt`
: Instrukce pro instalaci a spuštění programu RayTracer, včetně požadavků pro jeho provoz.

Navíc DVD obsahuje:

`data/`
: Ukázková a testovací data použitá v práci a pro potřeby obhajoby práce. Data vytvořená programem RayTracer pro demonstraci výsledků. Obsahuje obrázky i videa.

`install/`
: Instalátory aplikací, knihoven a jiných souborů nutných pro provoz programu / webové aplikace, které nejsou standardní součástí operačního systému.

`literature/`
: Některé položky literatury odkazované z dokumentace práce.

U veškerých odjinud převzatých materiálů obsažených na DVD jejich zahrnutí dovolují podmínky pro jejich šíření nebo přiložený souhlas držitele copyrightu. Pro materiály, u kterých toto není splněno, je uveden jejich zdroj (webová adresa) v textu dokumentace práce nebo v souboru `readme.txt`.