

Povray v počítačové grafice

Martin Mikulka

Bakalářská práce
2005



Univerzita Tomáše Bati ve Zlíně
Fakulta technologická

ABSTRAKT

Abstrakt česky

Tato práce je zaměřena na představení základních funkcí grafického programu POV-RAY. Program POV-RAY lze používat v souladu s podmínkami pro "OpenSource" software a je určen k vytváření 3D obrazů. Scéna je psána do zdrojového textového souboru, který je následně renderován za pomoci techniky zvané "Ray-tracing". Výstupem jsou vysoce kvalitní, až 48-bitové obrazy, jejichž kvalitu a rozměr lze libovolně nastavit.

Vysvětleny jsou základní objekty a modifikátory, které jsou pro běžné modelování zcela dostačující. Mezi základní objekty každé scény patří kamera, zdroje světla a modelovací objekty, na které jsou aplikovány materiály (textury). V práci je také stručně popsán programovací jazyk a základní nastavení scény. Nakonec je uvedeno několik příkladů demonstrujících možnosti tohoto programu.

Abstract in English

The aim of this thesis is to introduce basic functions of the graphical program POV-RAY. The program is distributed under an "OpenSource" license and is designed for creation of 3-D images. The scene is described in the source code and is then rendered using the "Ray-tracing" technique. The outcome are high quality up to 48-bit images, whose quality and size can be adjusted.

I explain the basic objects and modifiers which are sufficient for basic modelling. The basic objects of each scene are: camera, light sources and modelling objects on which different textures are applied. I also briefly describe the programming language and the basic setting of a scene. At the end I show a few examples demonstrating the possibilities of the program.

PODĚKOVÁNÍ

Tímto bych rád poděkoval vedoucímu mé bakalářské práce, panu Ing. Pavlu Pokornému, za uvedení do problému a za velmi cenné rady a připomínky v průběhu zpracování mé práce.

Martin Mikulka

OBSAH

OBSAH	4
ÚVOD.....	7
I. TEORETICKÁ ČÁST	8
1 PŘEDSTAVENÍ PROGRAMU	9
1.1 RAY-TRACING	9
1.2 HISTORIE PROGRAMU	10
1.3 PŘEHLED NEJVÝZNAMNĚJŠÍCH RYSŮ PROGRAMU.....	10
2 POPIS JAZYKA.....	12
2.1 KOMENTÁŘE	12
2.2 KLÍČOVÁ SLOVA (REZERVOVANÁ SLOVA)	12
2.3 IDENTIFIKÁTORY	13
2.4 ČÍSELNÉ VÝRAZY	13
2.5 VEKTOROVÉ VÝRAZY	13
2.6 ŘETĚZCE	13
2.7 DEFINICE BAREV	14
2.8 ARITMETICKÉ OPERÁTORY	15
2.9 POROVNÁVACÍ OPERÁTORY.....	15
2.10 LOGICKÉ OPERÁTORY.....	15
3 OBECNÝ POPIS SCÉNY.....	16
3.1 SOUŘADNÝ SYSTÉM.....	16
3.2 TRANSFORMACE.....	16
3.2.1 Scale	17
3.2.2 Rotate	17
3.2.3 Translate.....	17
3.2.4 Matrix.....	17
3.2.5 Pořadí transformací	18
3.2.6 Transformace textur a objektů	18
3.3 VKLÁDÁNÍ SOUBORŮ DO SCÉNY	18
4 OBJEKTY	19

4.1	KAMERA (CAMERA).....	19
4.1.1	Umístění a orientace kamery (location, look_at):.....	19
4.1.2	Naklonění kamery (sky):.....	19
4.1.3	Zorný úhel kamery (angle)	19
4.1.4	Typy projekce.....	20
4.1.5	Rozostření	21
4.2	ZDROJE SVĚTLA.....	21
4.2.1	Bodové všesměrové světlo (pointlight)	22
4.2.2	Svazek světla ohraničený kuželem (spotlight).....	22
4.2.3	Svazek světla ohraničený válcem (cylinder).....	24
4.2.4	Světlo s rovnoběžnými paprsky (parallel)	24
4.2.5	Plošný zdroj světla (area_light).....	24
4.2.6	Světlo, které nevrhá stín (shadowless).....	26
4.2.7	Přidružený objekt (looks_like).....	27
4.2.8	Promítací objekt (projected_through).....	27
4.3	KONEČNÉ PLNÉ PRIMITIVNÍ TVARY.....	27
4.3.1	Kapka (blob).....	27
4.3.2	Kvádr (box).....	29
4.3.3	Kužel (cone).....	29
4.3.4	Válec (cylinder).....	30
4.3.5	Výšková plocha (height_field)	30
4.3.6	Lathe.....	31
4.3.7	Hranol (prism).....	33
4.3.8	Koule (sphere).....	35
4.3.9	Sphere Sweep	35
4.3.10	Superquadric ellipsoid.....	36
4.3.11	Surface of revolution (sor).....	36
4.3.12	Text (text)	37
4.3.13	Anuloid (torus).....	38
4.4	NEKONEČNÉ PRIMITIVNÍ TVARY	38
4.4.1	Rovina (plane).....	38
4.5	CSG OPERACE	39
4.5.1	Sjednocení (union)	39
4.5.2	Průnik (intersection).....	39
4.5.3	Rozdíl (difference)	40
4.5.4	Sloučení (merge).....	40
4.6	MODIFIKÁTORY OBJEKTŮ	40
4.6.1	Clipped_by	41
4.6.2	Inverse	42
4.6.3	Hollow.....	42
4.6.4	No_shadow, no_image, no_reflection.....	42
4.6.5	Double_illuminate.....	43
4.6.6	Sturm.....	43
5	TEXTURY	44

5.1	PIGMENT	44
5.1.1	Jednobarevné pigmenty	44
5.1.2	Vzorované pigmenty	45
5.1.3	Barevné mapy	45
5.1.4	Pigmentové mapy	46
5.1.5	Obrazové mapy	46
5.2	NORMAL	47
5.2.1	Změna měřítka normál	48
5.3	FINISH	48
5.3.1	Okolní světlo (ambient)	48
5.3.2	Difúzní odraz (diffuse)	49
5.3.3	Odlesky (phong, specular)	49
5.3.4	Odraz (reflection)	49
II.	PRAKTICKÁ ČÁST	51
6	MODELY	52
6.1	HODINY	52
6.2	DÁMA	53
6.3	KANCELÁŘSKÝ STŮL	53
6.4	STOLNÍ LAMPA	54
6.5	MÍSTNOST	55
6.6	VYHODNOCENÍ VLIVU KVALITY VÝSTUPU A POUŽITÝCH OBJEKTŮ NA DOBU RENDEROVÁNÍ	56
	ZÁVĚR	58
	SEZNAM POUŽITÉ LITERATURY	59
	SEZNAM OBRÁZKŮ	60
	SEZNAM PŘÍLOH	61

ÚVOD

Tato práce je určena jako pomůcka pro studenty FT UTB ve Zlíně oboru Informační technologie, kteří v průběhu studia absolvují předmět "Počítačová grafika". Ve vyučovacích hodinách tohoto předmětu je program POV-RAY probírán. Dokumentace k programu v české verzi je však dosti málo, což bylo hlavním důvodem vzniku této práce.

Program POV-RAY lze používat v souladu s podmínkami pro "OpenSource" software a je určen k vytváření 3D obrazů. Scéna je psána do zdrojového textového souboru, který je následně renderován za pomoci techniky zvané "Ray-tracing". Výstupem jsou vysoce kvalitní, až 48-bitové obrazy, jejichž kvalitu a rozměr lze libovolně nastavit.

Vzhledem k tomu, že kompletní dokumentace k tomuto programu obsahuje více než 500 stran, práci jsem zaměřil zejména na základní objekty a modifikátory, které jsou pro běžné modelování zcela dostačující. Mezi základní objekty každé scény patří kamera, zdroje světla a modelovací objekty, na které jsou aplikovány materiály (textury). V práci je také stručně popsán programovací jazyk a základní nastavení scény.

V textu se vyskytuje mnoho anglických slov, protože program pracuje s anglickou syntaxí. Pro lepší čitelnost a orientaci jsou tyto části syntaxe, stejně jako všechna rezervovaná slova programu, psány jiným stylem. Na konci práce je uveden seznam použité literatury, který obsahuje množství odkazů na internetové stránky. Vzhledem k tomu, že nejsem správce těchto internetových stránek, nemohu zaručit funkčnost uvedených odkazů.

Martin Mikulka

I. TEORETICKÁ ČÁST

1 PŘEDSTAVENÍ PROGRAMU

Program POV-RAY vytváří tří-rozměrné, fotorealistické obrazy za použití renderovací techniky zvané "ray-tracing" (sledování paprsku). Program načítá ze zdrojového textového souboru informace popisující vlastnosti scény, objekty, zdroje světla, umístění kamery apod. a tyto informace následně zpracovává a vykresluje do obrazu. Ray-tracing není ve skutečnosti rychlá technika, avšak produkuje obrazy obrovské kvality se skutečným stínováním, odrazy světla, perspektivou a dalšími efekty. Program je volně šiřitelný v souladu s podmínkami pro "OpenSource" software.

1.1 Ray-tracing

Ray-tracing je renderovací technika (detailně popsána např. v [11]), která kalkuluje obraz tak, jakoby sledovala cestu jednotlivých světelných paprsků vyslaných ze všech definovaných světelných zdrojů. Ve skutečnosti jsou paprsky vysílány ze zdroje a osvětlují objekty, odráží se na nich, nebo jimi prostupují. Část odraženého světla nakonec směřuje k našim očím popř. do objektivu kamery. Jelikož je ale obrovské množství paprsků, které jsou odraženy mimo pozorovací médium, trvalo by velmi dlouho, než by byla tímto postupem zpracována a vykreslena výsledná scéna. Proto POV-RAY sleduje světelné paprsky směrem opačným, tedy od pozorovacího média ke zdroji světla.

Pro každý pixel výsledného obrazu je vyslán alespoň jeden paprsek směrem do scény. Každý paprsek tedy vychází z místa pozorovatele (z kamery) a prochází pozorovacím oknem (výsledný obraz).

Pokaždé, když je paprskem zasažen objekt, program vypočítá barvu osvětleného povrchu. Pro tento výpočet je paprsek vyslán z místa dopadu na objekt směrem ke každému světelnému zdroji, aby bylo možné zjistit intenzitu a barvu osvětlení bodu. Tyto "stínové" paprsky jsou testovány také proto, aby bylo jasné, zda uvažovaný bod na objektu je osvětlen, či zastíněn. Pokud je povrch objektu odrazivý nebo průsvitný, tak jsou z tohoto bodu vyslány a sledovány nové paprsky, které mohou interakcí s okolními objekty ještě ovlivnit výslednou barvu uvažovaného pixelu.

Definováním dalších např. atmosférických efektů (mlha, opar, oheň), radiosity a nebo odrazivých a průsvitných objektů dochází k dalšímu generování a sledování velkého

množství dodatečných paprsků pro každý pixel, což samozřejmě způsobuje zpomalení renderování.

1.2 Historie programu

Program POV-RAY byl vyvinut z programu DKBTrace 2.12 vývojovým týmem POV-TEAM, jehož členové na programu pracovali ve svém volném čase. Protože má POV-Ray bohatou tradici, neuškodí se trochu poohlédnout, jak vlastně vznikl.

V roce 1986 uvedl programátor David Kirk Buck svůj první Raytracer DKBTrace (David Kirk Buck's Trace) pro Amigu založený na volném zdrojovém kódu v C pro Unix. O něco později, v letech 1987-1988 začal spolupracovat s jiným programátorem Aaronem Collinsem a rozšířili DKBTrace i na platformu PC. Poslední verze DKBTrace nesla označení 2.12 (rok 1989).

Program se těšil velké popularitě, ale protože oba programátoři nestačili programovat nové verze dostatečně rychle, začalo se mluvit na diskuzních fórech CompuServe o napsání úplně nového raytraceru.

David Buck neváhal a poslal na fórum zprávu, že poskytne DKBTrace jako základ pro tento nový raytracer za splnění tří podmínek. Výsledný program musí být freeware s přístupným zdrojovým kódem (dnes bychom řekli OpenSource), musí být nadále vyvíjen pro různé platformy a musí mít nové jméno. A tak vznikl POV-Ray - Persistence Of Vision Raytracer jehož poslední verze má pořadové číslo 3.6. Program obsahuje detailní dokumentaci pro jeho používání a vytváření scén, dále obsahuje mnoho před-definovaných scén, objektů a materiálů.

1.3 Přehled nejvýznamnějších rysů programu

- jednoduchý jazyk pro popis scény podobný jazyku C
- obsáhlá knihovna vzorových příkladů, soubory obsahující definice objektů a materiálů pro vložení do scény
- velmi vysoká kvalita výstupu (až 48-bitové obrazy), možnost použití vyhlazování hran (antialiasing)
- mnoho výstupních formátů – bmp, png, tga ...

- mnoho typů kamer včetně perspektivních, pravoúhlých, fish-eye atd.
- mnoho typů světel – všesměrová, s rovnoběžnými paprsky, bodová, plošná ...
- atmosférické efekty (mlha, opar, duha), částicová média (oheň, pára, mraky, prach)
- základní modelovací tvary – kvádry, koule, válce, kužele, mnohoúhelníky, plochy...
- pokročilé modelovací tvary – prstence, výškové plochy ...
- kombinace tvarů (objektů) pomocí CSG (Constructive Solid Geometry)
- definice vlastností povrchu – barva, průhlednost, odlesk, hrbolatost ...
- definice vlastností interieru – index lomu ...
- mnoho před-definovaných barevných a normálových vzorů
- vytváření vlastních textur nebo použití textur před-definovaných – zlato, chrom, kámen, dřevo.
- kombinace několika polopropustných textur
- možnost definice proměnných, použití maker, cyklů apod. – parametrizování scény
- možnost přerušení renderingu a uložení aktuálního stavu s možností budoucího navázání na přerušenu akci

2 POPIS JAZYKA

Scény vytvořené a určené pro program POV-RAY jsou psány speciálním jazykem, podobným jazyku C. Zdrojový kód každé scény je uložen v textovém souboru, který je programem načten a zpracován. Na základě načtených informací si program vytvoří interní model, který následně renderuje.

Ve všech částech této dokumentace je použito jednotného stylu zápisu zdrojového kódu pro snadnou přehlednost. Povinné položky syntaxe v konstruktorech objektů jsou psány velkými písmeny např. **POLOZKA**, volitelné položky jsou psány velkými písmeny do hranatých závorek **[POLOZKA]**. Pokud je požadována jedna a více položek, jsou přidány za položku tři tečky **NEKOLIK_POVINNYCH_POLOZEK...**, více volitelných položek se obdobně zapisuje do hranatých závorek **[NEKOLIK_VOLITELNYCH_POLOZEK...]**. Reálné hodnoty nebo výrazy jsou psány znaky smíšené velikosti např. **Float_1**, zatímco vektory jsou psány do špičatých závorek např. **<Vektor_1>**. Volba parametru z různých hodnot je značena svislou čarou např. **Hodnota_1 | Hodnota_2**. V některých případech jsou také v syntaxi vyžadovány hranaté závorky. Např. při definici polí, seznamů apod. V těchto případech jsou mezi závorky a obsah vkládány mezery jako např. **[POLOZKA]**.

2.1 Komentáře

V programu jsou dostupné dva typy komentářů – jednořádkový a blokový (víceřádkový). Jednořádkový komentář je značen znaky **//**, vše co následuje tuto dvojici znaků na daném řádku bude ignorováno. Blokový komentář začíná dvojicí **/*** a končí dvojicí ***/**. Vše, co je uvedeno uvnitř tohoto bloku bude programem ignorováno.

2.2 Klíčová slova (rezervovaná slova)

V programu je definováno mnoho klíčových slov (např. **camera**, **color**, **error**, **object**, **scale** atd.), pomocí kterých je umožněno definovat jednotlivé objekty a vlastnosti. Názvy těchto klíčových slov jsou složeny pouze z malých písmen, příp. spojovníku **"_"**. Při použití identifikátoru se shodným názvem jako některého z klíčových slov, dojde k chybě. Proto se doporučuje v názvech identifikátorů používat alespoň jedno velké písmeno, aby nedocházelo ke konfliktům.

2.3 Identifikátory

V programu lze určité skupině objektů nebo modifikátorů přiřadit tzv. identifikátor. Použití identifikátorů vede ke zjednodušení zdrojového kódu scény a tudíž i ke zlepšení přehlednosti. Identifikátor lze přiřadit jakémukoliv objektu – např. `object`, `transform`, `texture`, `pigment` atd. Deklarace identifikátoru je následující:

```
#declare NAZEV_IDENTIFIKATORU = nazev_objektu{ MODIFIKATORY... }
```

kde `NAZEV_IDENTIFIKATORU` je až 40-ti místné jméno identifikátoru, které musí začínat písmenem. Jméno identifikátoru může obsahovat písmena smíšené velikosti, může obsahovat číslice a případně spojovník `"_"`. `MODIFIKATORY` jsou jakékoliv platné modifikátory pro daný objekt `nazev_objektu`.

2.4 Číselné výrazy

Program umí pracovat s desetinnými čísly, jako oddělovač se používá desetinná tečka. Pokud se jedná o logické nebo `bool` hodnoty, jakákoliv nenulová hodnota je považována za `TRUE`, nulová hodnota je považována za `FALSE`. Protože dochází k zaokrouhlovacím chybám, hodnoty velmi blízké nule (resp. hodnoty menší než implicitně nastavená hodnota $\epsilon \approx 10^{-10}$) jsou rovněž považovány za `FALSE`. Program podporuje vědecké značení velmi malých či velkých čísel (např. $6e-3 = 0,006$).

2.5 Vektorové výrazy

Vektory v programu POV-RAY mohou být složeny ze dvou až pěti složek. Většinou jsou ale používány 3-složkové vektory, které udávají x, y, z hodnoty odpovídající příslušným souřadnicovým osám. V některých případech je vyžadován 2-složkový vektor, nazývaný UV vektor. Fraktálové objekty vyžadují 4-složkové vektory, při definici barev jsou požadovány vektory 5-složkové, avšak akceptovány jsou i vektory 3-, 4-složkové a současně jsou použity výchozí hodnoty pro chybějící složky.

2.6 Řetězce

V některých případech je programem vyžadováno zadání textového řetězce. Např. při specifikaci jména souboru, nebo zadání textového řetězce pro objekt `text`. Pokud je

potřeba do řetězce zadat některý z nepovolených znaků, musíme před něj vložit tzv. "escape" sekvenci, tedy znak "\\".

2.7 Definice barev

Každá barva v programu je definována 5-složkovým vektorem. První tři složky **red**, **green**, **blue** udávají intenzitu jednotlivých primárních barev, ze kterých se skládá výsledná barva.

Čtvrtá složka **filter** udává množství filtrovaného světla procházejícího objektem. Příkladem může být barevné sklo. Světlo procházející takovými objekty je zbarveno do příslušného odstínu, protože některé frekvence světla jsou tímto materiálem absorbovány zatímco jiné frekvence procházejí. Barva objektu je dána množstvím a barvou světla prošlého skrz objekt.

Poslední pátá složka **transmit** udává množství nefiltrovaného světla procházejícího objektem. Příkladem může být třeba průsvitný papír, tenká látka, velmi jemná síť. V tomto případě všechny frekvence procházejí objektem, barva světla je tedy stejná, ale intenzita prošlého světla je snížena. V případě, že je tato složka nastavena na hodnotu 1, tak veškeré světlo prochází skrz, tzn. objekt je zcela průhledný, nevrhá žádný stín.

Každá složka v definici barvy by měla být vyjádřena číslem v intervalu $<0,1>$.

Tří-složková barva je dána klíčovým slovem **rgb**. Složky **filter** a **transmit** jsou automaticky nastaveny na nulu. Čtyř-složkové barvy jsou dány klíčovým slovem **rgba** nebo **rgbt**, pěti-složkové slovem **rgbft**. Za klíčovým slovem vždy následuje vektor, který udává hodnoty jednotlivých složek.

Příklad:

Čtyř-složková definice barvy. Pokud na objekt s definovanou barvou **rgba** $<1,0.5,0,1>$ (světelný filtr) dopadá světlo barvy **rgb** $<0.9,0.9,0.9>$, tak výsledné světlo procházející skrz filtr bude mít barvu $<0.9,0.9,0.9> * <1,0.5,0> = <0.9,0.45,0>$. To znamená, že filtr propustí veškerou intenzitu červené barvy, propustí 50% intenzity zelené barvy a kompletně pohltí modré světlo.

2.8 Aritmetické operátory

V programu lze pracovat s těmito aritmetickými operátory:

() +A -A !A A*B A/B A+B A-B

2.9 Porovnávací operátory

Podmínkou pro použití porovnávacích operátorů je uzavření výrazu do kulatých závorek.

Dostupné operátory jsou:

(A < B) (A <= B) (A = B) (A != B) (A >= B) (A > B)

Výstupními hodnotami výrazů jsou 1 ... TRUE, 0 ... FALSE. Při porovnávání hodnot se opět využívá hodnoty epsilon (např. $A = B$ pokud $\text{abs}(A-B) < \text{epsilon}$).

2.10 Logické operátory

Podmínkou pro použití logických operátorů je uzavření výrazu do kulatých závorek.

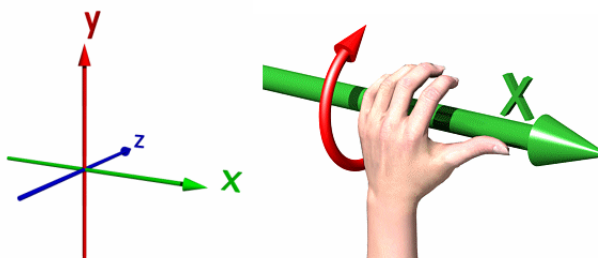
(A & B) (A | B)

První výraz je logický součin, druhý je logický součet.

3 OBECNÝ POPIS SCÉNY

3.1 Souřadný systém

Pro práci s tří-rozměrnými objekty je nezbytné definovat souřadný systém. Obvykle má souřadný systém v programu POV-RAY orientovánu kladnou část osy x směrem doprava, kladnou část osy y nahoru a kladnou část osy z směrem dopředu. Tento typ systému je nazýván levo-ruký, protože právě levou rukou (na rozdíl od pravé) lze demonstrovat směr jednotlivých os. Levým palcem ukážeme v kladném směru osy x, ukazováčkem ve směru osy y a prostředníčkem ve směru osy z. Levou ruku lze použít také ke stanovení směru otáčení. Palcem ukážeme v kladném směru osy otáčení. Ostatní zahnuté prsty ukazují kladný směr otáčení.



Obr. 1 – Levoruký souřadný systém v programu POV-RAY

Umístění objektů v tomto souřadném systému je většinou definováno tří-složkovým vektorem. Tyto tři složky odpovídají x, y a z směru souřadnicových os. Např. vektor $\langle 1, 2, 3 \rangle$ definuje bod, umístěný jednu jednotku doprava, dvě jednotky nahoru a tři jednotky dopředu vzhledem k počátku souřadného systému, tedy bodu $\langle 0, 0, 0 \rangle$.

Vektory však pokaždé nedefinují jen body. Mohou být použity k určení velikosti, posunu, rotaci objektů na scéně nebo textur na objektech. Změny velikosti, umístění, orientace a deformace objektů jsou v programu začleněny do kategorie modifikátorů nazvané transformace (**transformations**).

3.2 Transformace

Podporované transformace jsou rotace (**rotate**), změna velikosti (**scale**), posun (**translate**) a maticová transformace (**matrix**). Tyto transformace lze aplikovat na objekty či textury přiřazené objektům.

3.2.1 Scale

Změna velikosti objektů nebo textur se provádí modifikátorem **scale** následovaným vektorovým výrazem. Hodnoty ve vektoru, které jsou větší než 1 způsobí zvětšení (roztáhnutí) objektu v příslušném směru, hodnoty menší než 1 způsobí zmenšení. Hodnota rovna 1 nemění velikost objektu. Změna velikosti je relativní k původní velikosti objektu. Pokud je klíčové slovo **scale** následováno reálnou hodnotou, je tato hodnota použita pro všechny tři složky vektoru. Pokud by byla hodnota ve vektoru nastavena na 0, POV-RAY automaticky změní tuto hodnotu na 1 a vypíše varování.

3.2.2 Rotate

Orientaci objektu nebo textury lze změnit modifikátorem **rotate** následovaným vektorovým výrazem. Jednotlivé složky vektorů udávají úhel otočení ve směru příslušných os. Kladné směry otáčení byly vysvětleny v oddílu "Souřadný systém". Pořadí jednotlivých rotací má vliv na výslednou orientaci objektu. Rotace se provádí nejprve kolem osy x, dále kolem osy y a nakonec kolem osy z. Pokud tedy dochází k neočekávaným orientacím objektu, je možné aplikovat více transformací **rotate** a vždy uvést rotaci jen kolem jedné osy, čímž může dojít k záměně pořadí. Rotace se provádí kolem os souřadného systému, nikoli kolem souřadných os daného objektu.

3.2.3 Translate

Prvky mohou být posunuty přidáním modifikátoru **translate** následovaného vektorovým výrazem. Jednotlivé složky vektoru udávají posun ve směru příslušných souřadnicových os. Modifikátor posune objekt o zadanou vzdálenost vzhledem k jeho původní pozici. Pokud je klíčové slovo **translate** následováno reálnou hodnotou, je tato hodnota použita pro všechny tři složky vektoru.

3.2.4 Matrix

Modifikátor **matrix** následovaný seznamem dvanácti reálných čísel **c00**, **c01**, ..., **c32** uzavřeným do špičatých závorek "< >" lze použít k explicitní definici transformací použitých na objekt nebo texturu. Jakýkoliv myšlený bod $P = \langle p_x, p_y, p_z \rangle$ na tělese nebo textuře bude transformován na bod $Q = \langle q_x, q_y, q_z \rangle$ dle následujících vztahů:

$$q_x = c_{00} * p_x + c_{10} * p_y + c_{20} * p_z + c_{30}$$

$$qy = c01 * px + c11 * py + c21 * pz + c31$$

$$qz = c02 * px + c12 * py + c22 * pz + c32$$

Syntaxe:

```
matrix <
    c00, c01, c02,
    c10, c11, c12,
    c20, c21, c22,
    c30, c31, c32
>
```

3.2.5 Pořadí transformací

Pořadí použitých transformací má vliv na výslednou pozici a tvar objektu. Protože rotace a změna velikosti se vztahují k osám souřadného systému, je mnohokrát vhodné definovat objekty ohniskem v počátku, poté aplikovat změny velikosti a rotace a nakonec posunutí na příslušné místo. Běžnou chybou je nesprávné pořadí použitých transformací, jejichž výsledkem je v mnohých případech únik objektu ze zorného pole kamery. Doporučené pořadí transformací je tedy **scale**, **rotate**, **translate**.

3.2.6 Transformace textur a objektů

Umístění transformačních modifikátorů uvnitř konstrukturu objektu může mít vliv pouze na některé části tohoto objektu. Pokud umístíme transformace před definici textur, tyto transformace ovlivní pouze tvar objektu. Pokud umístíme transformace do definice textur, tyto transformace ovlivní pouze zobrazení textury. Pokud umístíme transformace až na konec konstrukturu objektu, tyto transformace ovlivní zobrazení jak objektu tak i textury. Transformace mohou být také aplikovány na modifikátory **pigment** a **normal**.

3.3 Vkládání souborů do scény

Do scény lze vkládat externí, tzv. "hlavičkové" soubory pomocí direktivy **#include**. Obsah vloženého souboru je chápán jako součást zdrojového kódu. Pro vkládané soubory se často používá přípona **.inc**. Některé soubory jsou dodávány s programem, jsou uloženy v podadresáři **include**, který se nachází v instalačním adresáři programu. Tyto soubory obsahují např. definice barev (**colors.inc**), vlastnosti povrchů (**finish.inc**), materiály (**glass.inc**, **metals.inc**, **stones.inc**, **woods.inc**), textury (**textures.inc**) apod.

4 OBJEKTY

4.1 Kamera (**camera**)

Definice kamery popisuje pozici, typ projekce a vlastnosti kamery, která zobrazuje scénu.

Syntaxe:

```
camera{ [POLOZKY...] }

POLOZKY:
    TYP_KAMERY | VEKTOR_KAMERY | MODIFIKATORY_KAMERY |
    IDENTIFIKATOR_KAMERY
TYP_KAMERY:
    perspective | orthographic | fisheye | ultra_wide_angle | omnimax |
    panoramic | cylinder CylinderType | spherical
VEKTOR_KAMERY:
    location <Location> | right <Right> | up <Up> | direction
    <Direction> | sky <Sky>
MODIFIKATORY_KAMERY:
    look_at <Look_At> | angle HORIZONTAL [VERTICAL] | blur_samples
    Num_of_Samples | aperture Size | focal_point <Point> | confidence
    Blur_Confidence | variance Blur_Variance | NORMAL | TRANSFORMACE
```

4.1.1 Umístění a orientace kamery (**location**, **look_at**):

Ve většině případů lze pozici a orientaci kamery definovat pomocí dvou vektorů **<location>** a **<look_at>**, kde první z nich udává bod umístění kamery a druhý vektor udává bod, na který je kamera orientována. Vektor **<look_at>** by měl vždy být poslední položkou v konstruktoru kamery, aby bylo zaručeno, že kamera je orientována na daný bod a není ovlivněna jiným modifikátorem.

4.1.2 Naklonění kamery (**sky**):

Standartně se kamera nejdříve vodorovně otočí kolem svislé osy, aby byla orientována směrem na cílový bod. Poté se svisle nakloní tak, aby směřovala středem zorného pole přesně na cílový bod. Můžeme však chtít, aby kamera byla nakloněna do strany. Toho docílíme použitím modifikátoru **sky**, čímž změníme normálový vektor kamery.

4.1.3 Zorný úhel kamery (**angle**)

Klíčové slovo **angle** následované číselnou hodnotou udává vodorovný zorný úhel kamery ve stupních. Velikost zorného úhlu má nějaká omezení (v závislosti na typu použitým typu projekce), avšak obecně hodnoty větší než 360° způsobují opakování scény na průmětně.

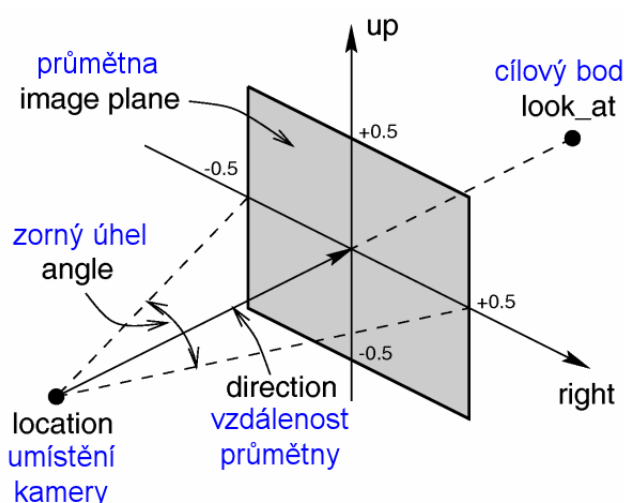
Pro sférický (**spherical**) typ kamery lze definovat i svislý zorný úhel. Pokud není definován, je použita hodnota odpovídající polovině vodorovného zorného úhlu.

4.1.4 Typy projekce

V programu existují různé typy projekce kamery. Modifikátor určující typ projekce by měl být uveden v konstruktoru kamery na prvním místě. Implicitně nastavený typ je perspektivní projekce.

4.1.4.1 Perspektivní projekce (**perspective**)

Tento typ projekce simuluje klasickou kameru. Paprsky se sbíhají do jediného bodu – ohniska. Zorný úhel kamery je určen buď z poměru délek vektorů **<direction>** a **<right>** nebo z modifikátoru **angle**, který udává zorný úhel přímo. Zorný úhel musí být větší než 0° a menší než 180° . Tento typ projekce, včetně modifikátorů je znázorněn na obrázku:



Obr. 2 – Schematické znázornění parametrů perspektivní projekce

4.1.4.2 Pravoúhlá projekce (**orthographic**)

Čistě pravoúhlé zobrazení. Tato projekce využívá rovnoběžných paprsků k vytvoření pohledu na scénu. Tato projekce může ovlivnit viditelnost některých objektů ve scéně. Objekty, které jsou v blízkosti umístění kamery se rovněž nemusejí zobrazit.

4.1.4.3 Sférická projekce (**fisheye**)

Při použití této projekce by měl vzniknout kulatý obraz (projekce na kouli), na kterém by byly zobrazeny objekty v závislosti na zorném úhlu. Pokud bychom uvedli hodnotu zorného úhlu 360°, zobrazily by se všechny objekty na scéně, včetně těch, které jsou umístěny za kamerou.

Ostatní typy projekce lze využít jen velmi zřídka, proto se jimi nebudeme zabývat.

4.1.5 Rozostření

Program POV-RAY dokáže simulovat rozostření vysláním určitého počtu paprsků do okolí uvažovaného bodu. Rozostření je pak průměrná hodnota stanovená z těchto paprsků.

Pokud chceme pracovat s rozostřením, musíme uvést v konstruktoru kamery modifikátor **aperture** následovaný číselným výrazem, který udává hloubku zaostřené oblasti. Veliké hodnoty tohoto modifikátoru způsobují velké rozostření, zatímco malé hodnoty způsobují velkou hloubku ostrosti.

Střed ostré oblasti se dá nastavit modifikátorem **<focal_point>**, jehož hodnotou je vektor ukazující na bod s maximální ostroší.

Dále je potřeba uvést modifikátor **blur_samples**, který udává počet vyslaných paprsků pro každý pixel obrazu. Čím vyšší je číselná hodnota tohoto modifikátoru, tím hladší bude rozostření, ale tím déle bude trvat vykreslení scény. Přestože modifikátor **blur_samples** udává maximální počet paprsků, skutečný počet může být ovlivněn adaptivním mechanismem, který ukončí vysílání paprsků, jakmile je dosaženo určitého stupně přesnosti daného modifikátorem **confidence**. Každý další paprsek by už nijak výrazně neovlivnil výsledek.

4.2 Zdroje světla

Zdroje světla jsou chápány také jako objekty, avšak nemají viditelný (zobrazitelný) tvar. Jsou to body vyzařující (emitující) světelné paprsky. Ve scéně může být použito více zdrojů světla a to i různých typů.

Obecná syntaxe pro nastavení zdroje světla:

```
light_source{  
    <Location>, COLOR
```

```

    [MODIFIKATORY_SVETLA...]
}

MODIFIKATORY_SVETLA:
    TYP_SVETLA | OBECNE_MODIFIKATORY_SVETLA | TRANSFORMACE
TYP_SVETLA:
    spotlight | cylinder | parallel | shadowless
Parametry modifikátoru SPOTLIGHT a CYLINDER:
    radius Radius | falloff Falloff | tightness Tightness | point_at
    <Spot>
Parametry modifikátoru PARALLEL:
    point_at <Spot>
Parametry modifikátoru AREA_LIGHT:
    area_light <Axis_1>, <Axis_2>, Size_1, Size_2 | adaptive Adaptive |
    jitter Jitter | circular | orient
OBECNE_MODIFIKATORY_SVETLA:
    looks_like { OBJECT } | fade_distance Fade_Distance | fade_power
    Fade_Power | media_attenuation [Bool] | media_interaction [Bool] |
    projected_through { OBJECT }

```

První dvě položky jsou dostupné pro všechny typy světel. Vektor **<Location>** udává umístění zdroje světla. Parametr **COLOR** udává barvu emitovaného světla.

Kategorie **OBECNE_MODIFIKATORY_SVETLA** je dostupná pro všechny typy světel.

Existují tři základní bodové typy světel. Pokud není uveden modifikátor **LIGHT_TYPE** jedná se o všesměrové bodové světlo (**pointlight**). Dalšími typy jsou **spotlight** a **cylinder**.

4.2.1 Bodové všesměrové světlo (**pointlight**)

Jedná se o nejjednodušší typ světla. Světelné paprsky jsou emitovány rovnoměrně do všech směrů. Tento typ světla vytváří ostré stíny. Při použití tohoto typu světla jsou vyžadovány pouze parametry **<Location>** a **COLOR**.

Syntaxe:

```

light_source {
    <Location>, COLOR
    [OBECNE_MODIFIKATORY_SVETLA]
}

```

4.2.2 Svazek světla ohraničený kuželem (**spotlight**)

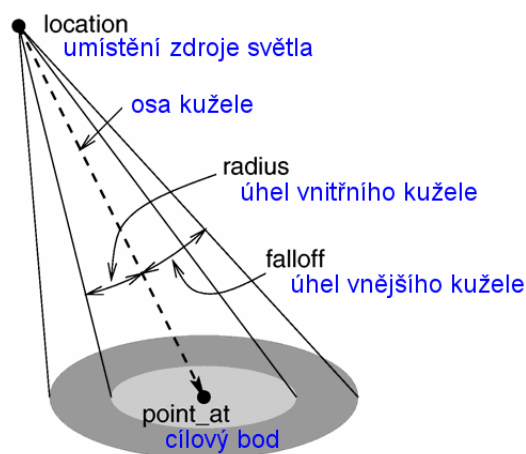
Jedná se také o bodový zdroj světla. Tento typ světla může být použit k vytvoření světelného kužele (např. reflektor). Světelný kužel je rozdělen na dvě části – vnitřní a vnější. Světelné paprsky ve vnitřním kuželi mají konstantní intenzitu a na dopadové ploše vytvářejí tzv. "hot-spot". Vnitřní kužel je určen parametrem **radius**, což je úhel mezi osou

a povrchem vnitřního kužele. Parametr **falloff** udává úhel mezi osou a povrchem vnějšího kužele. Paprsky procházející prostorem mezi povrchem vnitřního a vnějšího kužele postupně ztrácejí intenzitu, na povrchu vnějšího kužele je intenzita již nulová. Přestože intenzita slábne pozvolna a vzniká neostrý okraj kužele, tento typ světla vytváří ostré stíny.

Parametry **falloff**, **radius** a **tightness** přímo ovlivňují výsledný vzhled světelného kužele a jsou funkční pouze při použití typů **spotlight** nebo **cylinder**.

Parametr **tightness** udává rychlost poklesu intenzity ve vnitřním kuželu. Čím vyšší hodnota, tím rychlejší pokles intenzity. Pokud je nastavena hodnota 0, intenzita se nemění.

Vektor **<point_at>** udává cílový bod světelného záření. Úsečka mezi zdrojovým a cílovým bodem tvoří osu kužele. Následující obrázek pomáhá porozumět jednotlivým parametrům.



Obr. 3 – Kuželový svazek světla

Syntaxe:

```
light_source{
    <Location>, COLOR
    spotlight
    [OBECENE_MODIFIKATORY_SVETLA]
}
```

Parametry modifikátoru SPOTLIGHT a CYLINDER:

```
radius Radius | falloff Falloff | tightness Tightness | point_at
<Spot>
```

Čím jsou hodnoty parametrů **radius** a **falloff** bližší, tím rychleji klesá intenzita a tím se stává hrana vnějšího kužele ostřejší.

4.2.3 Svazek světla ohraničený válcem (**cylinder**)

Modifikátor **cylinder** vytváří válcový svazek světla, který je velmi vhodný pro simulování laserových paprsků. Tento typ světla pracuje velmi podobně jako kuželové světlo (**spotlight**), avšak paprsky tohoto světla jsou ohraničeny válcem. Stále se jedná o bodový typ světla, tzn. paprsky jsou emitovány z jednoho bodu a nejsou tedy rovnoběžné.

Vzhled výsledného světelného svazku opět ovlivňují parametry **point_at**, **radius**, **falloff** a **tightness**. Vysvětlení významu parametrů je v předchozím oddílu (**spotlight**).

Syntaxe:

```
light_source{  
    <Location>, COLOR  
    cylinder  
    [OBECENE_MODIFIKATORY_SVETLA]  
}
```

Parametry modifikátoru SPOTLIGHT a CYLINDER:

```
radius Radius | falloff Falloff | tightness Tightness | point_at  
<Spot>
```

4.2.4 Světlo s rovnoběžnými paprsky (**parallel**)

Modifikátor **parallel** může být použit u jakéhokoliv typu zdroje světla. Tyto světla jsou vhodná pro simulování velmi vzdálených zdrojů světla, např. sluneční svit. Jak už název napovídá, světelné paprsky jsou rovnoběžné.

Veškeré části objektů nad úrovní zdroje světla stále budou osvětleny v závislosti na směru světelného záření, avšak tyto části nebudou vrhat stín a ani na ně stín nebude dopadat.

Případné použití parametru **<point_at>** je přípustné až po použití modifikátoru **parallel**.

4.2.5 Plošný zdroj světla (**area_light**)

Plošné zdroje světla pokrývají omezenou (konečnou) dvourozměrnou plochu. Tento druh světla vytváří měkké stíny, protože světelné paprsky vycházejí z více bodových zdrojů. Díky tomu může nastat situace, kdy budou některé body na scéně osvětleny jen částečně. Tato situace však nemůže nastat u bodového zdroje světla, kdy je bod na scéně buďto zcela osvětlený, nebo zcela zakrytý.

Modifikátor **area_light** vytváří plošný zdroj světla, tedy pole bodových zdrojů. Tento fakt se však projevuje pouze na stínech, protože celková intenzita osvětlení scény bude

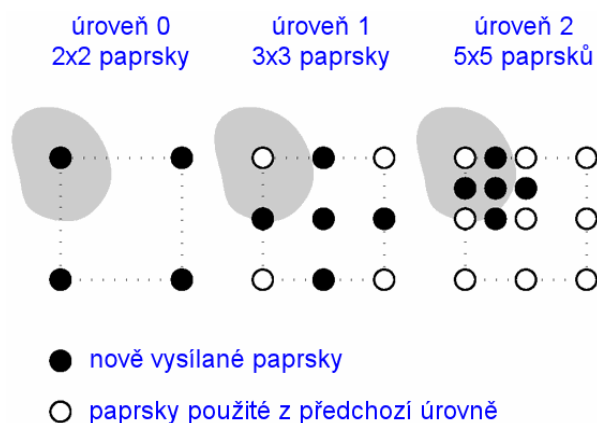
stejná, jako by bylo použito jednoho zdroje bodového. Modifikátor `area_light` lze použít na jakýkoliv typ zdroje světla.

Parametry modifikátoru `area_light` určují umístění, velikost a orientaci plošného zdroje. Vektor `<Location>` je středem obdélníku určeného vektory `<Axis_1>` a `<Axis_2>`. Parametry `Size_1`, `Size_2` určují počet bodových zdrojů v poli.

Při výchozím úplném vzorkování (bez použití parametru `adaptive`) se obrazový bod počítá tak, že se na něj vyšle světelný paprsek z každého bodového zdroje v poli. To však v některých scénách může být časově velmi náročné, obzvlášť pokud máme ve scéně použito velké množství světel. Proto existuje parametr `adaptive`, díky kterému se dá výpočet scény urychlit. Adaptivní vzorkování se snaží provést stejný výpočet, avšak s použitím minimálního počtu vyslaných paprsků. Čím vyšší je hodnota tohoto parametru, tím přesnější bude výpočet stínů, ale tím také bude výpočet déle trvat. Hodnoty parametru `adaptive` mohou být pouze celá nezáporná čísla.

Při výpočtu scény s adaptivním vzorkováním se nejprve vyšlou paprsky ze čtyř rohů plošného zdroje. Pokud je množství světla dopadajícího na uvažovaný bod ze všech čtyř zdrojů přibližně stejné, jedná se buď o zcela zakrytý nebo zcela osvětlený bod. Intenzita světla se pak vypočítá jako průměrná hodnota ze čtyř paprsků. Pokud se ale množství světla dopadající z jednotlivých zdrojů výrazně liší, jedná se o bod částečně zakrytý. V takovém případě se plocha světelného zdroje rozdělí na čtvrtiny (do rohů každé čtvrtiny se přidají další bodové zdroje) a výpočet se opakuje stejným způsobem.

Výhoda adaptivního vzorkování spočívá v tom, že konečné "rozparcelování" pole bodových zdrojů není rovnoměrné. K jemnějšímu dělení totiž dojde právě na hranicích viditelné a zakryté části pole plošného zdroje. V ideálním případě je tedy dosaženo stejně přesného výsledku jako při vzorkování úplném, přičemž počet vyslaných paprsků je podstatně menší.



Obr. 4 – Popis funkce adaptivního mechanismu

Množství vyslaných paprsků však nikdy nepřekročí hodnoty specifikované parametry **Size_1** a **Size_2**. Například **area_light x, y, 4, 4** udává, že se bude jednat o plošný zdroj o 4x4 bodových zdrojích. Pokud specifikujeme dále **adaptive 2**, bude to znamenat, že se plocha plošného zdroje bude moci 2x rozdělit na kvadranty, tzn. pro výpočet se bude moci použít až pole o 5x5 prvcích, což však převyšuje maximální dostupné množství bodových zdrojů. V tomto případě nebude použito adaptivní vzorkování a pro výpočet se použije pole o 4x4 prvcích.

Volitelný parametr **jitter** způsobuje, že pozice jednotlivých bodových zdrojů v síti plošného zdroje je náhodná, čímž se eliminuje případný vznik několika ostrých stínů. Rozmístění je náhodné pro každý výpočet scény, tzn. že tento parametr není vhodné používat při vytváření animací.

Syntaxe:

```
light_source{
  <Location>, COLOR
  area_light
  <AXIS_1>, <AXIS_2>, Size_1, Size_2
  adaptive Adaptive | jitter | circular | orient
  [OBECE_MODIFIKATORY_SVETLA]
}
```

4.2.6 Světlo, které nevrhá stín (**shadowless**)

Použitím modifikátoru **shadowless** zajistíme, že daný zdroj nebude způsobovat vrhání stínů. Jeho použití má význam pro přisvětlení scény. Tento modifikátor lze použít pro všechny typy zdrojů světla. Jediné omezení je, že modifikátor **shadowless** musí být před a nebo za všemi parametry modifikátorů **spotlight** a **cylinder**. Toto světlo bude osvětlovat všechny prvky na scéně v závislosti na jeho umístění, tzn. dojde k

nerovnoměrnému přisvětlení scény. Pokud bychom chtěli prosvětlit celou scénu rovnoměrně, můžeme využít globálního nastavení scény a modifikátoru `ambient_light`.

Syntaxe:

```
light_source{
    <Location>, COLOR
    shadowless
    [OBEČNE_MODIFIKATORY_SVETLA]
}
```

4.2.7 Přidružený objekt (`looks_like`)

Běžně zdroje světla nemají viditelný tvar. Světlo prostě vychází z neviditelného bodu nebo plochy. Zdroji však můžeme tvar přiřadit použitím modifikátoru `looks_like { OBJECT }`. Podmínkou je, aby použitý objekt již byl definován. Při použití tohoto modifikátoru je implicitně nastaveno, že objekt nebude vrhat stín, tím pádem světlo bude moci projít skrz něj. Přidružený objekt bude umístěn do bodu daného vektorem `<Location>`.

4.2.8 Promítací objekt (`projected_through`)

Modifikátor `projected_through { OBJECT }` může být použit s jakýmkoliv typem zdroje, včetně plošných zdrojů. Podmínkou je, aby použitý objekt již byl definován. Použití tohoto modifikátoru lze chápat jako opak stínění, tzn. pouze paprsky, které směřují skrz promítací objekt projdou dále do scény. Tímto modifikátorem lze vytvořit světelný svazek téměř neomezených tvarů. Žádný z objektů umístěný mezi zdroj a promítací objekt nebude vrhat stín vytvořený tímto světlem. Promítací objekt nebude na scéně zobrazen.

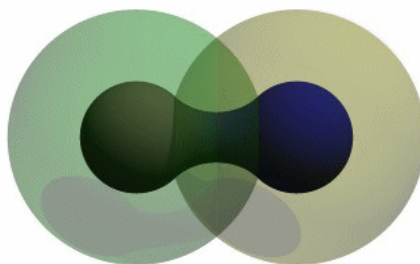
4.3 Konečné plné primitivní tvary

V programu je dostupných 14 konečných základních tvarů - `blob`, `box`, `cone`, `cylinder`, `height_field`, `Julia fractal`, `lathe`, `prism`, `sphere`, `sphere_sweep`, `superellipsoid`, `surface of revolution`, `text` objekt a `torus`. V této práci budou vysvětleny jen nejpoužívanější tvary.

4.3.1 Kapka (`blob`)

Představme si každou dílčí součást (komponent) objektu `blob` jako objekt vznášející se v prostoru. Komponent je tvořen pomyslným silovým polem, které má maximální hodnotu ve středu a zaniká na povrchu tohoto objektu. Silové pole všech komponent je sloučeno

dohromady a formuje tak celkový objekt **blob**. Program vyhledá v prostoru takové body, kde je silové pole rovno právě definované prahové hodnotě **threshold**. Tyto body vytváří povrch objektu **blob**. Body, které mají větší silovou hodnotu, než je definovaný práh jsou považovány jako vnitřní body objektu, body které mají nižší silovou hodnotu jsou vně objektu.



Obr. 5 – Objekt blob

Syntaxe:

```
blob {
    KOMPONENTY_OBJEKTU...
    [MODIFIKATORY_OBJEKTU_BLOB...]
}

KOMPONENTY_OBJEKTU:
    [component Strength, Radius, <Center>] |
    sphere{
        <stred>, Polomer, [strength] Strength
        [MODIFIKATORY_KOMPONENTU...]
    } |
    cylinder{
        <stred_1>, <stred_2>, Polomer, [strength] Strength
        [MODIFIKATORY_KOMPONENTU...]
    }
MODIFIKATORY_KOMPONENTU:
    TEXTURE | PIGMENT | NORMAL | FINISH | TRANSFORMATION
MODIFIKATORY_OBJEKTU_BLOB:
    threshold Float | hierarchy [Boolean] | sturm [Boolean] |
    MODIFIKATORY_OBJEKTU
```

Maximální síla pole je dána parametrem **strength** a je ve středu komponentu. Hodnota parametru může být kladná i záporná. Nulová hodnota není povolena. Pokud je hodnota kladná, bude tento komponent přitahovat ostatní komponenty. Pokud bude hodnota **strength** záporná, budou ostatní komponenty odpuzovány. Komponent se záporným parametrem **strength** nebude zobrazen. Komponenty z různých objektů **blob** se nijak neovlivňují.

Modifikátor `threshold` následovaný číselnou hodnotou udává sílu pole, kterou program vyhledává. Výchozí hodnota je `threshold 1.0`, hodnota tohoto modifikátoru musí být kladná. Komponent zmizí, pokud bude prahová hodnota větší, než je maximální síla tohoto objektu. Čím vyšší je prahová hodnota, tím blíže je povrch objektu `blob` ke středu každého komponentu, čím nižší je hodnota, tím blíže je povrch objektu `blob` k povrchu každého komponentu.

Jednotlivé komponenty mohou mít své vlastní modifikátory `texture`, `pigment`, `normal`, `finish`. Na komponenty také lze aplikovat jednotlivé transformace. Pokud chceme použít modifikátor `interior`, je to povoleno, pouze však pro celý objekt `blob`.

Modifikátor `component` je zde uveden jen z důvodů zpětné kompatibility. Dříve byl používán ke specifikaci jednotlivých komponent.

4.3.2 Kvádr (`box`)

Kvádr je definován dvojicí vektorů protilehlých vrcholů. Výsledný kvádr má stěny rovnoběžné s osami souřadnic. Pomocí objektových modifikátorů (`rotate`, `translate`, `scale`...) lze následně kvádr otočit, přemístit apod.

Syntaxe:

```
box{
    <vrchol1> , <vrchol2>
    [MODIFIKATORY_OBJEKTU]
}
```

4.3.3 Kužel (`cone`)

Kužel je definován dvojicí vektorů udávajících středy a dvojicí poloměrů jednotlivých podstav. Díky tomu můžeme vytvořit jak kužel se špičkou (horní poloměr nastavíme na 0), tak i komolý kužel s horní podstavou. Spojnice obou středů vytváří osu kužele a jednotlivé podstavy jsou na tuto osu kolmé. Pokud bychom chtěli kužel bez podstav, můžeme za definici druhého poloměru přidat modifikátor `open`.

Syntaxe:

```
cone{
    <střed1> , poloměr1
    <střed2> , poloměr2
    open | MODIFIKATORY_OBJEKTU
}
```

4.3.4 Válec (`cylinder`)

Válec je definován velmi podobně jako kužel, avšak zde stačí uvést jen jeden poloměr. Opět spojnice středů podstav tvoří osu válce a podstavy jsou na tuto osu kolmé. Rovněž zde můžeme uvést modifikátor `open` pro odstranění obou podstav.

Syntaxe:

```
cylinder{
    <střed1> , <střed2> , poloměr
    open | MODIFIKATORY_OBJEKTU
}
```

4.3.5 Výšková plocha (`height_field`)

Výškové plochy jsou objekty, které jsou používány zejména k vytváření hor nebo podobných zvlněných povrchů (terénů). Základem pro vytvoření tohoto objektu je zdrojový rastrový obrázek, jehož jednotlivé pixely tvoří uzlové body výškové plochy. Výška plochy nad daným pixelem je vypočítána z indexu barvy pixelu v barevné paletě obrázku. Mezi každou čtveřicí uzlových bodů jsou vytvořeny dva trojúhelníky. Kvalitu výškové plochy ovlivňují dva faktory – velikost zdrojového obrázku (počet pixelů v každém směru) a barevná hloubka. Čím větší je zdrojový obrázek, tím detailnější bude plocha. Počet barev ve zdrojovém obrázku udává počet výškových úrovní na ploše. Použijeme-li klasických grafických formátů (gif, bmp, jpeg ...), vytvoříme plochu se 256 výškovými úrovněmi. Pomocí některých grafických programů však lze vytvořit i speciální 16-bitové formáty (např. tga), díky kterým můžeme vytvořit výškovou plochu s cca 65 tisíci výškovými úrovněmi.

Syntaxe:

```
height_field{
    [TYP_HF]
    "zdrojovy_obrazek"
    smooth | water_level Float | hierarchy [Bool] |
    MODIFIKATORY_OBJEKTU
}
```

TYP_HF:
`gif | tga | pot | png | pgm | ppm | jpeg | tiff | sys | function`

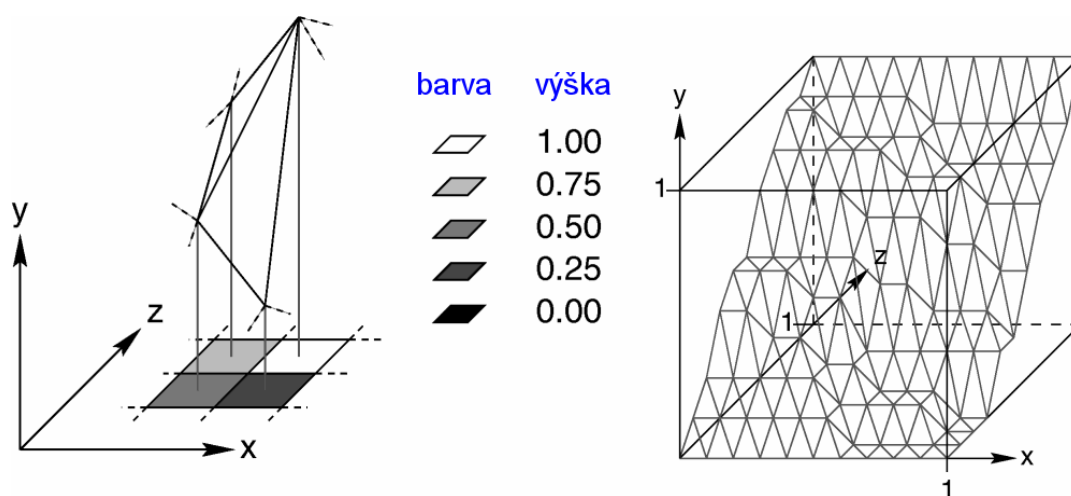
Výsledná výšková plocha bude mít rozměr 1x1x1, tzn. že rozměr zdrojového obrázku nijak neovlivní rozměr plochy. Velikost plochy a její umístění lze upravit použitím transformací.

Výška bodu plochy je závislá na indexu barvy v paletě. První barva v paletě (s indexem 0) odpovídá výšce 0, poslední barva v paletě odpovídá výšce 1, tzn. že barva daného pixelu nijak neovlivní výšku bodu plochy.

K obvyklým objektovým modifikátorům lze pro tento objekt doplnit další tři speciální – `water_level`, `smooth`, `hierarchy`.

Volitelný modifikátor `water_level`, který se uvádí za název zdrojového souboru, říká programu, aby při renderování ignoroval pixely s výškou pod úrovní hodnoty tohoto modifikátoru. Výchozí hodnota je 0, přípustné jsou hodnoty v intervalu $<0,1>$. Tento modifikátor simuluje funkci vodní hladiny, kdy jsou části objektu pod určitou úrovní neviditelné.

Standartně mají výškové plochy hrubý, zubatý vzhled protože jsou vytvořeny z mnoha plochých trojúhelníků. Přidáním modifikátoru `smooth` za název zdrojového souboru dojde k tomu, že tvar trojúhelníků bude přizpůsoben tak, aby plocha vypadala hladce.



Obr. 6 – Konstrukce výškové plochy ze zdrojového obrázku

4.3.6 Lathe

Tento objekt vznikne rotací dvourozměrné křivky x-y kolem svislé osy. Křivka je definována uzlovými body, které jsou spojeny lineárními, kvadratickými, kubickými nebo Bezierovými křivkami.

Syntaxe:

```
lathe{
    [TYP_KŘIVKY]
    počet_uzlovych_bodu,
```

```
<bod_1> , <bod_2> , ... , <bod_n>
sturm | MODIFIKATORY_OBJEKTU
}
```

TYP_KŘIVKY:

```
linear_spline | quadratic_spline | cubic_spline | bezier_spline
```

První položkou v konstruktoru objektu je typ použité křivky. Povinný celočíselný parametr `pocet_uzlovych_bodu` dále udává počet uzlových bodů křivky. Poté musí být uvedeny polohové vektory `<x,y>` jednotlivých uzlových bodů. Křivka není automaticky uzavřena. Pokud budeme požadovat, aby křivka byla uzavřená, je potřeba nastavit poslední bod totožný s bodem prvním.

Výchozí typ `linear_spline` je nejjednodušší, jde o spojení dvou následných uzlových bodů úsečkou. Ve výpočtu daného segmentu se tedy nezohledňují žádné další body. Segmenty jsou rovné, přechody mezi jednotlivými segmenty jsou ostré.

Modifikátor `quadratic_spline` vytváří mezi jednotlivými dvojicemi uzlových bodů kvadratickou křivku. K vykreslení segmentu kvadratické křivky jsou potřeba tři body. Bod 1 je pomocný pro výpočet sklonu křivky v koncových bodech 2 a 3. Vzhledem k tomu, že nelze vypočítat sklon křivky v bodě 1 (předchozí bod na křivce neexistuje), dojde k vykreslení křivky až od druhého bodu. Křivky mezi body sice jsou nelineární, ale přechody jednotlivých segmentů nejsou hladké, protože sklony křivek pod a nad daným koncovým bodem jsou různé.

Modifikátor `cubic_spline` vytváří kubickou křivku, která netrpí problémem s hladkostí přechodů. Problém je odstraněn použitím čtyř bodů pro výpočet segmentu křivky. Body 1 a 4 jsou pomocné pro výpočet sklonu křivky v koncových bodech 2 a 3. Mezi tyto koncové body se vykreslí daný segment. Protože jsou první a poslední definovaný bod pomocné, nedojde k jejich vykreslení. Jak úseky tak i přechody jsou hladké.

Posledním modifikátorem je `bezier_spline` což je alternativa ke kubické křivce. Body 1 a 4 jsou koncové body segmentu, body 2 a 3 jsou pomocné, které udávají sklon křivky v koncových bodech. Každý segment Bezierovy křivky je tedy tvořen čtyřmi body. Segmenty tohoto typu křivky jsou hladké, přechody mohou být jak hladké, tak i ostré (závisí na volbě pomocných bodů).

Počet vykreslených segmentů křivky závisí na použitém typu. Pokud máme definováno `n` bodů, a lineární typ křivky, tak dostaneme `n-1` segmentů. Kvadratická křivka bude mít `n-2`

segmenty, kubická křivka bude obsahovat $n-3$ segmenty, Bezierova křivka bude obsahovat $n/4$ segmentů, protože pro každý segment je potřeba čtyř bodů.

Pokud máme lineární typ křivky, použití modifikátoru `sturm` nemá v konstrukturu objektu význam.

4.3.7 Hranol (`prism`)

Hranol je objekt vytvořený vytáhnutím uzavřené jedno- nebo dvourozměrné x-z křivky podél osy y. Křivky jsou definovány uzlovými body, které jsou spojeny lineárními, kvadratickými, kubickými nebo Bezierovými křivkami.

Syntaxe:

```
prism{
    [PRISM_ITEM...]
    vyska_1, vyska_2
    pocet_uzlovych_bodu,
    <bod_11> , <bod_12> , <bod_13> , ... , <bod_1m>
    <bod_21> , <bod_22> , <bod_23> , ... , <bod_2m>
    ...
    <bod_n1> , <bod_n2> , <bod_n3> , ... , <bod_nm>
    open | sturm | MODIFIKATORY_OBJEKTU
}

PRISM_ITEM:
    linear_spline | quadratic_spline | cubic_spline | bezier_spline |
    linear_sweep | conic_sweep
```

Prvními položkami v konstrukturu objektu jsou typ křivky a typ vytáhnutí. Výchozími hodnotami jsou `linear_spline` a `linear_sweep`. Dále pak následují dvě číselné hodnoty `vyska_1` a `vyska_2`, definující y souřadnici jednotlivých podstav objektu. Celočíselná hodnota `pocet_uzlovych_bodu` definuje počet uzlových bodů křivky a je následována polohovými vektory `<x,z>` jednotlivých uzlových bodů.

U tohoto objektu je povoleno použít jakéhokoliv množství pod-objektů (se stejným typem křivky a vytáhnutí).

Interpretace uzlových bodů závisí na typu použité křivky. V případě typu `linear_spline` bude první definovaný bod počátečním bodem první křivky. Následující body budou patřit k první křivce do té doby, než bude definován koncový bod totožný s bodem počátečním. Tím dojde k uzavření křivky. Další definovaný bod pak bude počátečním bodem další křivky. Každá křivka v tomto objektu musí být uzavřena.

V případě typu `quadratic_spline` každý pod-objekt vyžaduje navíc jeden pomocný bod pro určení sklonu křivky v počátečním bodu. První definovaný bod je pomocný, který ve skutečnosti není součástí křivky. Počátečním bodem křivky je tedy bod druhý. Křivka bude uzavřena definováním koncového bodu totožného s bodem počátečním. Následující definovaný bod bude pomocný bod druhé křivky, další pak bude počátečním bodem druhé křivky.

Příklad:

```
prism {
    quadratic_spline
    0, 40,
    12
    <1,-1> , <0,0> , <6,0> , <6,8> , <0,8> , <0,0>
    //vnejsi ram; <1,-1> je pomocny bod a <0,0> je pocatek křivky

    <2,0> , <1,1> , <5,1> , <5,7> , <1,7> , <1,1>
    //vnejsi ram; <2,0> je pomocny bod a <1,1> je pocatek křivky

    pigment{rgb <1,0.5,0>}
}
```

V případě typu `cubic_spline` každý pod-objekt vyžaduje navíc dva pomocné body. Jeden na začátku pro určení sklonu křivky v počátečním bodě. Počátek křivky je tedy druhý definovaný bod, křivka je uzavřena jakmile je definován koncový bod totožný s bodem počátečním. Za tímto koncovým bodem ještě následuje druhý pomocný bod pro určení sklonu křivky v koncovém bodě. Další definovaný bod pak je počáteční pomocný bod, za ním následuje počáteční bod druhé křivky.

Typ `bezier_spline` je alternativa kubické křivky. Body 1 a 4 jsou koncové body segmentu, body 2 a 3 jsou pomocné, které udávají sklon křivky v koncových bodech. Každý segment Bezierovy křivky je tedy tvořen čtyřmi body.

Výchozím modifikátorem je lineární vytáhnutí (`linear_sweep`), tzn. že všechny stěny objektu jsou kolmé na rovinu x-z. Můžeme však také použít modifikátor `conic_sweep`, čímž docílíme toho, že se stěny budou sbíhat do jednoho bodu.

Stejně jako u válců, je tento objekt standartně uzavřen. Podstavy lze odstranit použitím modifikátoru `open`.

Pokud máme lineární typ křivky, použití modifikátoru `sturm` nemá v konstrukturu objektu význam.

4.3.8 Koule (**sphere**)

Koule je definována jejím středem a poloměrem. Poloměr musí být nenulová hodnota, pokud je záporná, znaménko bude ignorováno.

Syntaxe:

```
sphere{  
    <střed> , poloměr  
    [MODIFIKATORY_OBJEKTU]  
}
```

4.3.9 Sphere Sweep

Pomocí tohoto objektu lze vymodelovat tvar odpovídající prostoru, kterým prochází koule při pohybu po křivce. Při definici objektu je potřeba zadat celočíselný parametr **Pocet_uzlovych_pozic**. Objekt je dále definován seznamem vektorů uzlových pozic a poloměrů jednotlivých koulí. Výsledný objekt je následně interpolován.

Syntaxe:

```
sphere_sweep {  
    linear_spline | b_spline | cubic_spline  
    Pocet_uzlovych_pozic,  
    <stred_1>, polomer_1,  
    <stred_2>, polomer_2,  
    ...  
    <stred_n>, polomer_n,  
    tolerance DEPTH_TOLERANCE | MODIFIKATORY_OBJEKTU  
}
```

Program podporuje tři různé druhy interpolace:

linear_spline (lineární křivka) – mezi jednotlivé uzlové body se vkládají lineární segmenty. Koule v uzlových bodech jsou tedy spojeny rovnými objekty kuželovitého tvaru.

b_spline, **cubic_spline** – mezi jednotlivé uzlové body se vkládají segmenty tvořené kubickými nebo Bezierovými křivkami. Koule v uzlových bodech jsou spojeny objektem zakřiveného tvaru odpovídajícím tvaru křivky.

Počet uzlových pozic koulí není omezen. Pro objekt s interpolací typu **linear_spline** je potřeba definovat minimálně 2 uzlové pozice, pro interpolaci typů **b_spline** nebo **cubic_spline** je potřeba definovat minimálně 4 uzlové pozice.

Volitelný modifikátor **tolerance** lze použít pro zadání přesnosti výpočtů průniků objektů. Výchozí hodnota je 10^{-6} a měla by být dostačující pro většinu těchto objektů.

Pokud by se na výsledném objektu vyskytovaly černé oblasti, způsobené početními chybami, nastavte hodnotu tohoto modifikátoru na větší číslo, např. 10^{-4} .

4.3.10 Superquadric ellipsoid

Tento objekt se dá použít k vytvoření kvádrů a válců se zaoblenými hranami a jiných zajímavých tvarů. Matematicky je tvar definován rovnicí

$$f(x, y, z) = \left(\left| x \right|^{\left(\frac{2}{e} \right)} + \left| y \right|^{\left(\frac{2}{e} \right)} \right)^{\left(\frac{e}{n} \right)} + \left| z \right|^{\left(\frac{2}{n} \right)} - 1 = 0$$

kde hodnoty **e** a **n**, nazývané východo-západní a severo-jížní koeficienty, udávají tvar elipsoidu. Oba koeficienty musí být z intervalu $(0,1>$. Pokud jsou oba koeficienty rovny 1, výsledný objekt bude koule.

Syntaxe:

```
superellipsoid{
    <Value_E, Value_N>
    [MODIFIKATORY_OBJEKTU...]
}
```

Výsledný tvar bude umístěn středem do počátku a bude vykreslen do prostoru ohraničeného krychlí **<-1,-1,-1>**, **<1,1,1>**. Dvou-složkový vektor udává koeficienty pro rovnici uvedenou výše.

Velmi malé hodnoty mohou dělat problémy kořenovému řešiteli. Pro tento objekt nelze použít modifikátoru **sturm**.

4.3.11 Surface of revolution (sor)

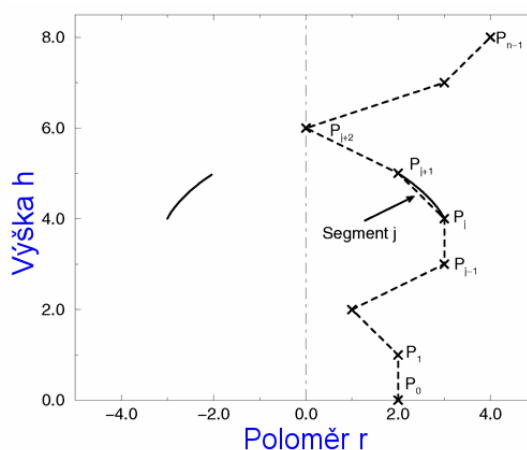
Tento objekt vznikne rotací 2D grafu funkce kolem osy y. Tento objekt je vhodný pro vytváření různých váz, sklenic apod.

Syntaxe:

```
sor{
    Pocet_uzlovych_bodu, <Bod_1>, <Bod_2>, ... <Bod_n>
    open | sturm | MODIFIKATORY_OBJEKTU
}
```

Celočíselný parametr `Pocet_uzlovych_bodu` udává počet dvou-složkových vektorů, které jej následují. Jednotlivé složky vektorů jsou poloměr (vzdálenost od osy otáčení) a výška. Uzlové body jsou propojeny hladkou křivkou. Křivka je následně rotována kolem osy y a formuje výsledný `sor` objekt. První a poslední definované body jsou jen pomocné a neleží na křivce, určují sklon křivky v koncových bodech. Tento objekt je ve své podstatě velmi podobný objektu `lathe`. Rozdíl je v tom, že objekt `sor` podléhá matematickým omezením pro definici funkce, tzn. že pro každou hodnotu na ose rotace může být definována maximálně jedna funkční hodnota (poloměr). Nelze tedy použít uzavřenou křivku, také je potřeba si dávat pozor, aby funkce neprocházela přes osu rotace.

Volitelný modifikátor `open` umožňuje odstranit plné postavy objektu.



Obr. 7 – Konstrukce objektu `sor` rotací 2D grafu funkce kolem osy y

4.3.12 Text (`text`)

Tento objekt vytváří třírozměrný text vytáhnutím jednotlivých písmen ve směru osy z. V současné době jsou implementovány pouze truetype písma (ttf) a truetype kolekce (ttc).

Syntaxe:

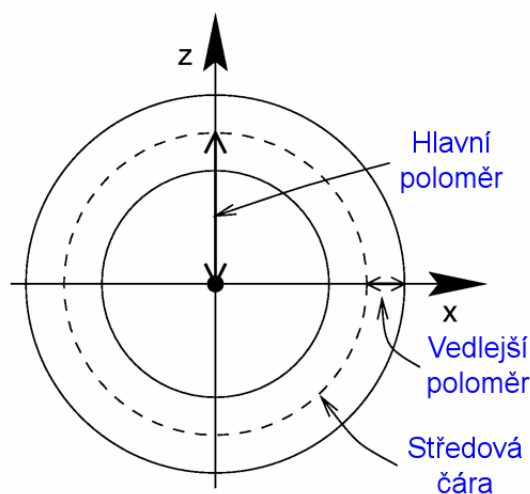
```
text {
    ttf "nazev_pisma.ttf/ttc" "Textovy retezec na vypsani"
    Thickness
    <Offset>
    [MODIFIKATORY_OBJEKTU...]
}
```

Text bude vypsán ve směru osy x, lícová strana písma je umístěna v x-y rovině. Číselný parametr `Thickness` udává tloušťku písma (ve směru osy z).

Vzdálenost mezi jednotlivými znaky lze změnit povinným vektorovým parametrem `<Offset>`. Znaky budou odsazeny v závislosti na hodnotách příslušných jednotlivým směrům ve vektoru.

4.3.13 Anuloid (`torus`)

Velmi užitečným objektem je anuloid, který je definován dvojicí číselných parametrů – poloměrů. Význam jednotlivých poloměrů je znázorněn na obrázku.



Obr. 8 – Objekt torus a jeho parametry

Syntaxe:

```
torus{
    Hlavni, Vedlejsi
    sturm | MODIFIKATORY_OBJEKTU
}
```

4.4 Nekonečné primitivní tvary

Do této kategorie patří objekty `plane`, `poly`, `cubic`, `quartic`, `quadric`, `parametric`, `isosurface`.

4.4.1 Rovina (`plane`)

Rovina je definována jejím normálovým vektorem a kolmou vzdáleností od počátku. V programu nelze definovat velikost roviny, jedná se tedy o nekonečný objekt. Např. vodorovná rovina umístěná 5 jednotek pod počátkem má následující zápis:

```
plane {
    <0,1,0> , -5
    [MODIFIKATORY_OBJEKTU...]
```

```
}
```

Protože jsou vektory $\langle 1, 0, 0 \rangle$, $\langle 0, 1, 0 \rangle$ a $\langle 0, 0, 1 \rangle$ používány často, byly v programu zavedeny tři klíčová slova **x**, **y**, **z**, která mohou být použita jako zkrácená verze zápisu. Alternativně lze tedy psát zápis takto:

```
plane {  
    y , -5  
}
```

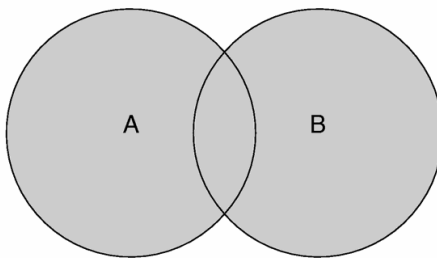
4.5 CSG operace

CSG (Constructive Solid Geometry) je mocná technika pro vytváření nových těles (objektů) zkombinováním několika primitivních tvarů. V programu jsou dostupné čtyři typy CSG operací – **union**, **intersection**, **difference** a **merge**. Syntaxe je následující:

```
nazev_operace{  
    OBJEKTY...  
    [MODIFIKATORY_OBJEKTU...]  
}
```

4.5.1 Sjednocení (**union**)

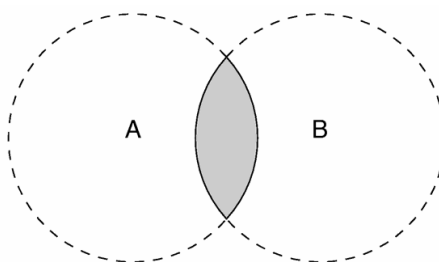
Uniony se používají ke sjednocení dvou a více objektů. Povrchy těles uvnitř unionu jsou zachovány, tzn. může docházet k nesrovnalostem při použití průsvitných materiálů.



Obr. 9 – CSG operace - sjednocení

4.5.2 Průnik (**intersection**)

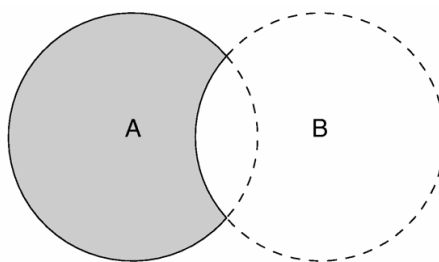
Intersection vytváří objekt odpovídající prostoru společnému pro všechny uvažované objekty (průnik). Pokud se objekty nepřekrývají, nedojde k zobrazení průniku ani objektů samotných.



Obr. 10 – CSG operace - průnik

4.5.3 Rozdíl ([difference](#))

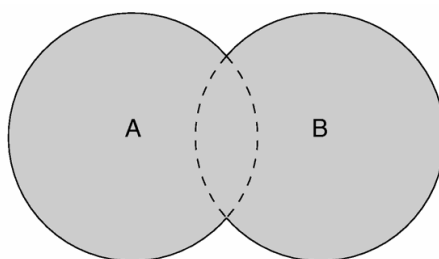
Difference se používá pro vytvoření rozdílu dvou a více tvarů. Jako základ se uvažuje prvně definovaný tvar. Od něj se odečítají všechny ostatní tvary. Pokud je základní tvar zcela uvnitř některého z později definovaných, nedojde k zobrazení žádného tvaru.



Obr. 11 – CSG operace - rozdíl

4.5.4 Sloučení ([merge](#))

Merge funguje velmi podobně jako union. Merge sloučí uvažované tvary v jeden celek, ale také odstraní povrchy tvarů, které jsou uvnitř výsledného objektu.



Obr. 12 – CSG operace - sloučení

4.6 Modifikátory objektů

U objektů lze použít velkého množství modifikátorů. Následující modifikátory mohou být použity u všech objektů:

```
clipped_by { NETEXTUROVANY_PLNY_OBJEKT... } | clipped_by { bounded_by } |
bounded_by { NETEXTUROVANY_PLNY_OBJEKT... } | bounded_by { clipped_by } |
no_shadow | no_image [Bool] | no_reflection [Bool] | hollow [Bool] |
inverse | sturm [Bool] | hierarchy [Bool] | double_illuminate [Bool] |
material {[IDENTIFIKATOR_MATERIALU][POLOZKY]} | interior_texture {TELO} |
texture {TELO} | pigment {TELO} | normal {TELO} | finish {POLOZKY...} |
interior {POLOZKY...} | photons {POLOZKY...}
TRANSFORMACE
```

Některé modifikátory (`texture`, `pigment`, `normal`, `finish`, `interior`) jsou vysvětleny v jiných částech této práce. Méně důležité modifikátory nebudou vysvětleny vůbec.

4.6.1 Clipped_by

Výsledkem použití modifikátoru `clipped_by` je objekt podobný objektu získanému CSG operací `intersection`.

Syntaxe:

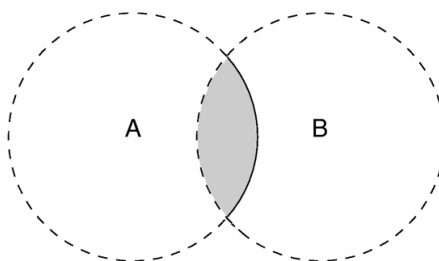
```
clipped_by { NETEXTUROVANY_PLNY_OBJEKT... } | clipped_by { bounded_by }
```

kde `NETEXTUROVANY_PLNY_OBJEKT` je jeden nebo více plných objektů, na které není aplikována textura.

Příklad:

```
object {
    Objekt_A
    clipped_by{Objekt_B}
}
```

Každá část objektu `Objekt_A`, která je uvnitř objektu `Objekt_B` je vykreslena, zatímco ostatní části jsou "odstřiženy". Rozdíl od CSG operace `intersection` je v tom, že tento objekt nemá jednu část povrchu. Následující obrázek ilustruje výsledný objekt.



Obr. 13 – Modifikátor Clipped_by

4.6.2 Inverse

Při použití CSG operací je mnohdy užitečné zaměnit vnitřní a vnější stranu objektu použitím modifikátoru **inverse**. Tato operace nijak neovlivní vzhled objektu, změní se jen způsob, jak program objekt vnímá.

Rozdíl mezi vnitřní a vnější stranou objektu je rovněž důležitý při aplikaci interieru na objekt zejména pokud je současně použit modifikátor **media**. Atmosférická media jako mlha se nezobrazují správně, pokud je kamera umístěna uvnitř objektu. Tento problém lze odstranit modifikátorem **inverse**.

Syntaxe:

```
object{  
    Muj_Objekt inverse  
}
```

4.6.3 Hollow

Ve výchozím nastavení program předpokládá, že objekty jsou plné, tzn. jsou tvořeny materiálem, který zcela vyplní vnitřní část objektu (interiér). Přidáním modifikátoru **hollow** lze vytvořit objekt prázdný, tzn. objekt bude mít jen "skořápku" tvořící jeho povrch. Použití tohoto modifikátoru je užitečné v případech, kdy chceme, aby uvnitř objektu bylo možné použít např. atmosférické efekty apod.

Syntaxe:

```
object{  
    Muj_Objekt hollow  
}
```

4.6.4 No_shadow, no_image, no_reflection

Tyto modifikátory jsou užitečné pro změnu viditelnosti objektů. Použitím těchto tří modifikátorů lze vytvořit zajímavé efekty, jako např. kouli, která vrhá stín ve tvaru krychle nebo kostku, která bude mít v zrcadle odraz tvaru kužele apod.

Modifikátorem **no_shadow** můžeme zajistit, aby objekt nevrhal žádný stín. Pokud použijeme modifikátor **no_image**, objekt nebude viditelný pro kameru. Přesto bude vrhat stín a objeví se v odrazech. Při použití modifikátoru **no_reflection** se objekt nebude zobrazovat v odrazech. Bude však viditelný pro kameru a skrz průhledné objekty a bude také vrhat stín.

Syntaxe:

```
object{
    Muj_Objekt no_shadow | no_image | no_reflection
}
```

4.6.5 Double_illuminate

Každý povrch má dvě strany – obvykle jednu orientovanou naproti světelnému zdroji a tudíž i osvětlenou, zatímco druhá strana zůstává zastíněná. Pokud použijeme modifikátoru `double_illuminate`, obě strany povrchu budou osvětleny. Modifikátor je vhodný pro simulování efektů jako např. průsvitnost (list papíru, stínítko u lampy apod.)

Syntaxe:

```
object{
    Muj_Objekt double_illuminate
}
```

4.6.6 Sturm

U některých objektů je možné si zvolit mezi rychlým, ale mnohdy nepřesným kořenovým řešitelem a nebo řešitelem Sturmiánským, který je sice pomalejší, ale přesnější. Hledání kořenů nastává u všech objektů, které vyžadují řešení polynomů.

Řešení polynomů nižších řádů je jednoduché, zatímco řešení polynomů vyšších řádů vyžaduje iterační algoritmy. Jeden z těchto algoritmů je Sturmiánský kořenový řešitel.

Modifikátor `sturm` lze použít pro tyto objekty – `blob`, `cubic`, `lathe` (pouze s kvadratickými křivkami), `poly`, `prism` (pouze s kubickými křivkami), `quartic`, `sor`.

5 TEXTURY

Vzhled povrchu objektu (materiál) popisuje modifikátor `texture`. Textura je kombinace modifikátorů `pigment`, `normal` a `finish`. Modifikátor `pigment` určuje barvu materiálu. Prostorové efekty na textuře lze ovlivnit modifikátorem `normal`. Modifikátor `finish` popisuje povrchovou úpravu materiálu (odrazivost apod.). Základním typem textury je `plain` (prostá), z tohoto typu lze definovat složitější typy – `patterned` (vzorovaná), `layered` (vrstvená).

5.1 Pigment

Barva nebo barevný vzor materiálu je definován modifikátorem `pigment`. Všechny prosté textury musí obsahovat tento modifikátor, pokud není definován, tak program použije výchozí, implicitně nastavenou hodnotu. Barva objektu bude odpovídat definované barvě v případě, že bude objekt plně osvětlen.

Syntaxe:

```
pigment {
    [IDENTIFIKATOR_PIGMENTU]
    [TYP_PIGMENTU]
    [MODIFIKATORY_PIGMENTU...]
}

TYP_PIGMENTU:
    PATTERN_TYPE | COLOR |
    image_map {
        BITMAP_TYPE "bitmap.ext" [IMAGE_MAP_MODS...]
    }

MODIFIKATORY_PIGMENTU:
    PATTERN_MODIFIER | COLOR_LIST | PIGMENT_LIST | color_map
    {COLOR_MAP_BODY} | colour_map { COLOR_MAP_BODY } | pigment_map
    {PIGMENT_MAP_BODY} | quick_color COLOR | quick_colour COLOR
```

Modifikátor `pigment` je součástí modifikátoru `texture`. Tento modifikátor však lze použít i bez uvedení modifikátoru `texture`, pokud chceme nastavit pouze barvu objektu. V konstruktoru `pigment` lze použít tyto speciální modifikátory – `color_map`, `pigment_map`, `quick_color`.

5.1.1 Jednobarevné pigmenty

Nejjednodušší typ pigmentu je jednotná barva. K definici jednotné barvy materiálu jednoduše přidáme specifikaci barvy uvnitř modifikátoru `pigment`. Specifikace barvy

sestává z nepovinného klíčového slova `color` a dále buďto z identifikátoru barvy nebo z vektoru určujícího množství RGB složek a případně filtrované a nefiltrované průsvitnosti. Jakékoliv modifikátory vzoru jsou v tomto případě ignorovány, protože zde není žádný vzor k modifikaci.

5.1.2 Vzorované pigmenty

Dostupné jsou čtyři barevné vzory – `checker`, `hexagon`, `brick` a `object`. Výstupem je vzor tvořený jednotnými barvami, jejichž přechodové hrany jsou ostré. Všechny specifikace barev by měly být odděleny čárkou, případně klíčovým slovem `color`. Vzory `brick` a `checker` jsou dvoubarevné, vzor `hexagon` je třibarevný. Pokud není specifikováno potřebné množství barev, jsou použity barvy implicitně nastavené.

5.1.3 Barevné mapy

Většina barevných vzorů nepoužívá ostré barevné přechody (jako např. u vzorů `checker`, `brick` ...), ale hladké přechody, které se pozvolna mění z jedné barvy na další. Barvy jsou definovány v modifikátoru pigmentu zvaném `color_map`, který popisuje také barevné přechody jednotlivých barev.

Syntaxe:

```
color_map { COLOR_MAP_BODY }

COLOR_MAP_BODY:
    COLOR_MAP_IDENTIFIER | COLOR_MAP_ENTRY...
COLOR_MAP_ENTRY:
    [ Value COLOR ] | [ Value_1, Value_2 color COLOR_1 color COLOR_2 ]
```

Reálné hodnoty `value_n` mohou být v rozsahu $<0,1>$ a `COLOR_n` jsou specifikace barev. Barevná mapa může obsahovat 2 až 256 barev.

Modifikátor `color_map` lze použít pro všechny vzory kromě `brick`, `checker`, `hexagon`, `object` a `image_map`. Seznam vzorů je v nápovědě k programu POV-RAY v oddílu "3.5.11 Patterns".

5.1.4 Pigmentové mapy

Obdobně jako při definování barevných přechodů pomocí barevných map lze vytvořit pigmentové mapy pomocí modifikátoru `pigment_map`. Syntaxe je identická se syntaxí pro barevné mapy, avšak specifikujeme pigmenty namísto barev.

Syntaxe:

```
pigment_map { PIGMENT_MAP_BODY }  
  
PIGMENT_MAP_BODY:  
    PIGMENT_MAP_IDENTIFIER | [Value PIGMENT_BODY]
```

kde `Value` je číselná hodnota v intervalu $<0,1>$ a položka `PIGMENT_BODY` je cokoliv, co může být uvedeno uvnitř modifikátoru `pigment`. Pigmentová mapa může obsahovat 2 až 256 záznamů.

5.1.5 Obrazové mapy

Pokud ani jedna z výše uvedených možností nesplňuje naše požadavky, lze použít obrazovou mapu `image_map` k rozprostření 2D obrázku kolem 3D objektu.

Syntaxe:

```
pigment{  
    image_map{  
        [TYP_BITMAPY] "bitmap[.ext]"  
        [MODIFIKATORY_IMAGE_MAP...]  
    }  
    [MODIFIKATORY_PIGMENT...]  
}  
  
TYP_BITMAPY:  
    gif | tga | iff | ppm | pgm | png | jpeg | tiff | sys  
MODIFIKATORY_IMAGE_MAP:  
    map_type Type | once | interpolate Type | filter Palette, Amount |  
    filter all Amount | transmit Palette, Amount | transmit all Amount
```

Za volitelným klíčovým slovem `TYP_BITMAPY` následuje řetězec obsahující název obrázku již definovaného typu. Soubory uvedené v modifikátorech `image_map` budou nejdříve hledány v domovském adresáři programu, poté budou hledány v adresářích nastavených v `Library_path` v konfiguračním `.ini` souboru.

Ve výchozím nastavení se obrázek mapuje v rovině x-y. Obrázek pokryje čtvercovou plochu danou protilehlými vrcholy (0,0) a (1,1) nezávisle na původní velikosti obrázku. Pokud chceme výchozí nastavení změnit, můžeme použít transformací.

Název souboru může být následován `MODIFIKATORY_BITMAPY` nebo modifikátory specifickými pro `image_map`. Jsou to `filter`, `filter all`, `transmit`, `transmit all`. Tyto modifikátory jsou dostupné jen pro obrazové mapy `image_map`. Dále lze použít bitmapových modifikátorů `map_type`, `once`, `interpolate`.

Pokud bychom chtěli, aby část obrázku byla průhledná, můžeme uvést klíčové slovo `filter` nebo `transmit` za uvedeným názvem obrázku. Klíčové slovo je následováno dvěma čísly. První celočíselná hodnota je index barvy v paletě. Druhá reálná hodnota z intervalu $<0,1>$ je požadovaná průhlednost. Tyto hodnoty by měly být odděleny čárkou.

Pokud bychom chtěli, aby byl celý obrázek průhledný, lze použít modifikátorů `filter all` nebo `transmit all` s požadovanými průhlednostmi.

Rozdíl mezi modifikátory `filter` a `transmit` je vysvětlen v kapitole "Definice barev".

5.2 Normal

Strukturu povrchu objektu by ve skutečnosti bylo velmi složité matematicky vymodelovat. Prostorové efekty na povrchu objektu však lze simulovat pomocí modifikátoru `normal`. Výpočet odrazu světla je závislý na normálovém vektoru v každém bodě povrchu. Tento vektor je kolmý na povrch a je orientován směrem od objektu. Deformací normálových vektorů lze docílit toho, že povrch objektu vypadá, jakoby měl určitou strukturu. Tento modifikátor nijak neovlivní tvar objektu, ovlivní však způsob, jakým se na povrchu láme či odráží světlo.

Syntaxe:

```
normal { [NORMAL_IDENTIFIER] [NORMAL_TYPE] [NORMAL_MODIFIER...] }
```

NORMAL_TYPE:

```
PATTERN_TYPE Amount | bump_map { BITMAP_TYPE "bitmap.ext"
[BUMP_MAP_MODS...]}
```

NORMAL_MODIFIER:

```
PATTERN_MODIFIER | NORMAL_LIST | normal_map { NORMAL_MAP_BODY } |
slope_map{ SLOPE_MAP_BODY } | bump_size Amount | no_bump_scale Bool
| accuracy Float
```

Modifikátor `normal` je součástí modifikátoru `texture`. Tento modifikátor však lze použít i bez uvedení modifikátoru `texture`, pokud chceme nastavit pouze strukturu povrchu objektu.

Modifikátor `PATTERN_TYPE` udává typ vzoru, může být následován číselnou hodnotou, která udává hloubku struktury. Seznam vzorů je v nápovědě k programu POV-RAY v oddílu "3.5.11 Patterns". V konstruktoru `normal` lze použít tyto tři speciální modifikátory – `slope_map`, `normal_map`, `bump_size`.

5.2.1 Změna měřítka normál

Pokud bychom z nějakého důvodu chtěli zamezit tomu, aby se měnilo měřítko nerovností, resp. normál, při aplikaci transformací, lze to provést použitím modifikátoru `no_bump_scale`.

5.3 Finish

Vzhled povrchu lze výrazně ovlivnit modifikátorem `finish`. Pomocí tohoto modifikátoru lze vytvořit potřebné efekty jako jsou odlesky, odraz okolních objektů apod.

Syntaxe:

```
finish { [FINISH_IDENTIFIER] [FINISH_ITEMS...] }

FINISH_ITEMS:
    ambient COLOR | diffuse Amount | brilliance Amount |
    phong Amount | phong_size Amount | specular Amount |
    roughness Amount | metallic [Amount] | reflection COLOR |
    crand Amount | conserve_energy BOOL_ON_OFF |
    reflection { Color_Reflecting_Min [REFLECTION_ITEMS...] } |
    irid { Irid_Amount [IRID_ITEMS...] }

REFLECTION_ITEMS:
    COLOR_REFLECTION_MAX | fresnel BOOL_ON_OFF |
    falloff FLOAT_FALLOFF | exponent FLOAT_EXPONENT |
    metallic FLOAT_METALLIC

IRID_ITEMS:
    thickness Amount | turbulence Amount
```

5.3.1 Okolní světlo (`ambient`)

Modifikátor `ambient` udává množství okolního světla dopadajícího na materiál. Výchozí hodnota je `ambient 0.1`, povolené hodnoty jsou v intervalu $\langle 0,1 \rangle$. Lze také použít vektorovou hodnotu, která specifikuje barvu okolního světla. Tento modifikátor ovlivní

osvícené i zastíněné části materiálu. Prosvětlení celé scény lze provést modifikátorem `ambient_light` v globálním nastavení.

5.3.2 Difúzní odraz (`diffuse`)

Při dopadu světla na povrch, se každý světelný paprsek odrazí jiným směrem, v závislosti na nerovnosti povrchu. Modifikátor `diffuse` ovlivňuje množství difúzně odraženého světla z povrchu. Výchozí hodnota je `diffuse 0.6`, povolené hodnoty jsou v intervalu $<0,1>$.

5.3.3 Odlesky (`phong`, `specular`)

Modifikátor `phong` ovlivňuje odlesk světla odraženého na povrchu objektu. Způsobí zobrazení odlesku v barvě odraženého světla. Výchozí hodnota je `phong 0`, povolené hodnoty jsou v intervalu $<0,1>$, kde maximální odlesk je číslo 1. Velikost odlesku lze ovlivnit modifikátorem `phong_size`. Čím vyšší hodnota tohoto modifikátoru, tím menší ale zářivější bude odlesk (materiál bude vypadat hladší). Hodnota modifikátoru nemá omezení, obvyklé hodnoty jsou v intervalu $<1,250>$. Výchozí hodnota je 40.

Modifikátor `specular` je velmi podobnou alternativou k modifikátoru `phong`. Používá rozdílného modelu, který vytváří důvěryhodnější odrazy. Hodnoty tohoto modifikátoru jsou v intervalu $<0,1>$. Velikost odlesku je ovlivněna modifikátorem `roughness`. Výchozí hodnota je `roughness 0.05`, obvyklé hodnoty jsou v intervalu $<0.0001,1>$. Modifikátor `metallic` lze použít u obou typů odlesků. Tento modifikátor přizpůsobí barvu odlesku tak, jakoby byl materiál z kovu.

5.3.4 Odraz (`reflection`)

Pokud se odražené světlo nerozptýlí, vznikne na povrchu zřetelný odraz. Intenzitu odrazu lze ovlivnit modifikátorem `reflection{...}`. Výchozí hodnota je `reflection{0}`, povolené hodnoty jsou v intervalu $<0,1>$. Hodnota `reflection{1}` znamená úplný odraz, je proto velmi vhodná, současně s modifikátorem `diffuse` nastaveným na 0, k vytvoření povrchu zrcadla.

V modifikátoru lze také uvést minimální hodnotu odrazu, např. `reflection{0.05, 1}` udává minimální odraz 0.05 a maximální odraz 1 (některé materiály jako např. voda odráží různé množství světla při různých úhlech dopadu).

II. PRAKTICKÁ ČÁST

6 MODEL

6.1 Hodiny

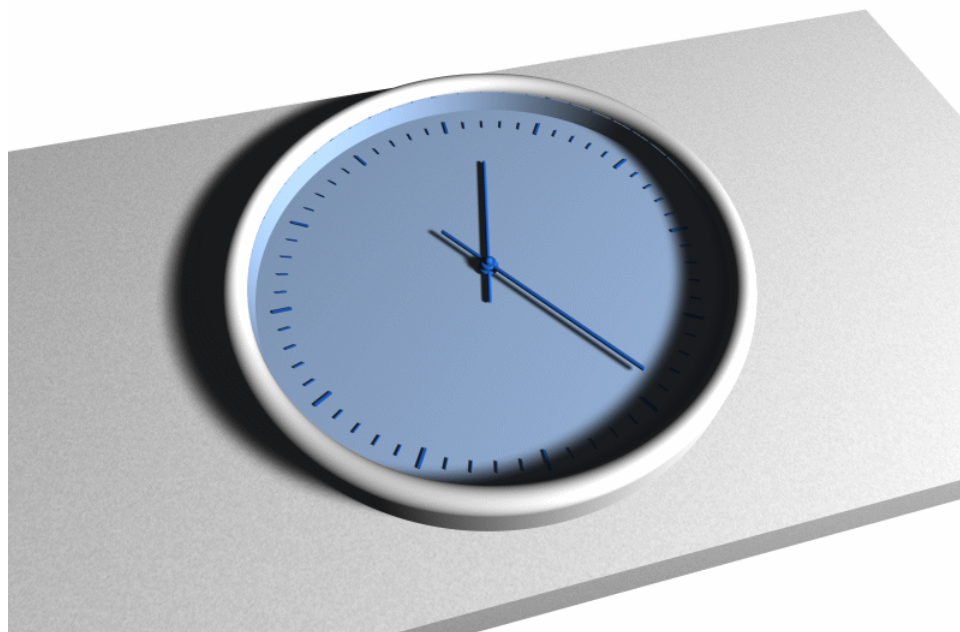
Tento model zahrnuje základní modelovací objekty a použití jednoduchých textur. Ve scéně je definován plošný zdroj světla, který vytváří hladké stíny. Hodiny jsou vytvořeny sjednocením anuloidu, který vytváří zaoblenou hranu, a rozdílu několika válců. Na tento sjednocený objekt je použit jednoduchý materiál, který je definován pouze konstruktorem `pigment`. Sklo hodin je vytvořeno válcem, který má definovanou barvu včetně filtrované průsvitnosti, tedy pomocí klíčového slova `rgba`. Ručičky jsou vytvořeny sjednocením několika koulí, anuloidů a válců, čímž byl vytvořen po všech stranách oblý tvar. Pro značku minut je deklarován identifikátor, protože vykreslení této značky bylo provedeno pomocí cyklu `while`.

Zdrojové soubory:

hodiny.pov

hodiny_materials.inc

Vyrenderovaná scéna:



6.2 Dáma

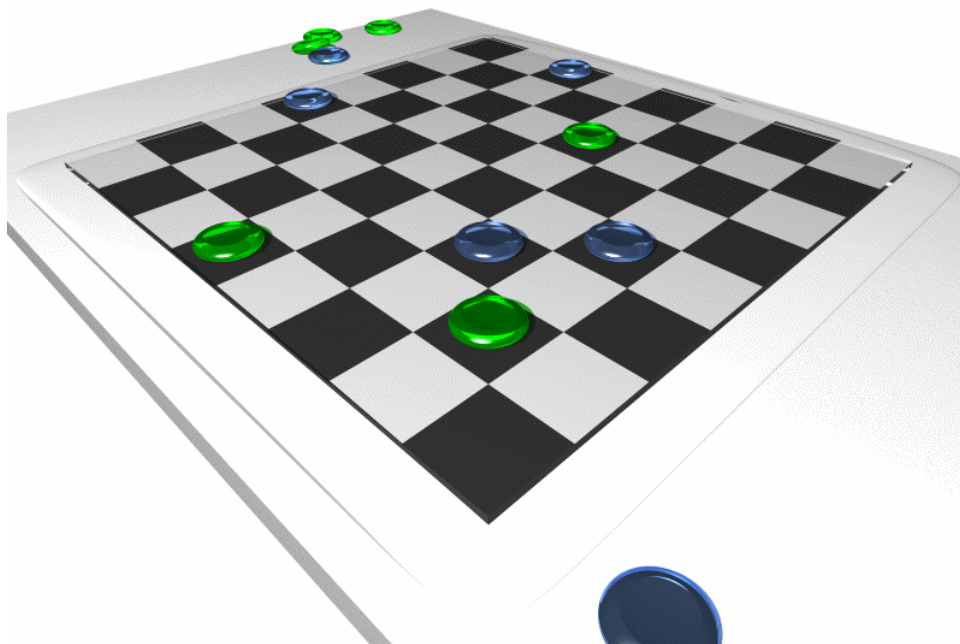
Tento model je vytvořen kombinací superkvadratického elipsoidu a několika kvádrů, které jsou zkombinovány pomocí CSG operací. Podklad šachovnice je vytvořen kvádrem, na který je aplikována textura tvořená dvoubarevným vzorem **checker**. Dotyková vrstva šachovnice je vytvořena skleněnou deskou, stejně jako jsou ze skla vytvořeny kameny. Materiál odpovídající sklu je definován čtyřsložkovou barvou, tedy včetně filtrované průsvitnosti. Dále je v definici uveden modifikátor **ior**, tedy lom světla, pro sklo zcela nezbytný. Vizuální efekty jsou vylepšeny díky použití konstruktoru **finish** a modifikátoru **specular**, který vytváří na materiálu odlesk. Kameny jsou vytvořeny rozdílem dvou koulí a kvádrů. Tento rozdíl je nakonec asymetricky zvětšen.

Zdrojové soubory:

dama.pov

dama_materials.inc

Vyrenderovaná scéna:



6.3 Kancelářský stůl

Pro tento model bylo nutné vytvořit materiály reprezentující dva druhy dřeva. Dřevěný vzhled byl vytvořen použitím barevného vzoru **wood**, který byl dále upraven modifikátorem **warp** a jeho parametrem **turbulence**. Vzhled materiálu byl podpořen

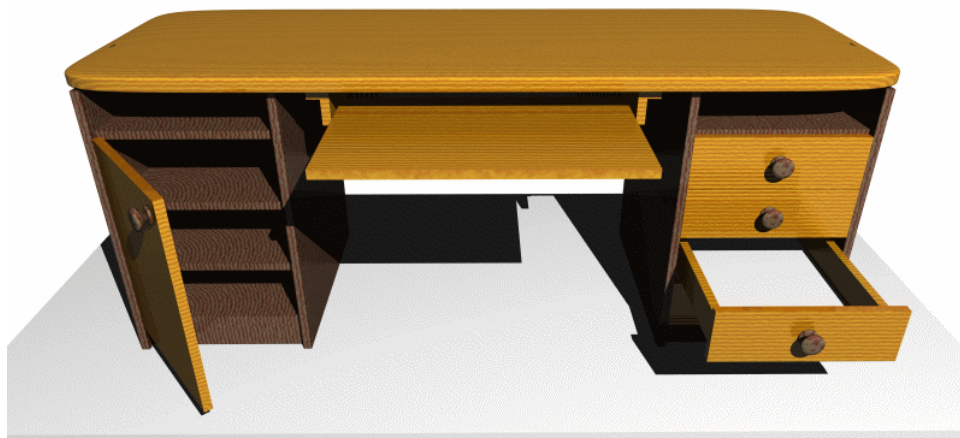
uvedením také normálového vzoru `wood`. Při aplikaci tohoto materiálu na objekt je potřeba dbát na jeho orientaci. Stůl je velmi jednoduchý model, sestavený z několika kvádrů. Pracovní plocha je vytvořena ze superkvadratického elipsoidu, díky kterému bylo možné vytvořit oblé rohy. Úchytky na šuplících byly vymodelovány objektem `sor`. Zde bylo použito bodového zdroje světla, který vytváří ostré stíny.

Zdrojové soubory:

stul.pov

stul_materials.inc

Vyrenderovaná scéna:



6.4 Stolní lampa

U tohoto modelu je dobré si všimnout různého pořadí použitých transformací, které ovlivňuje výsledný tvar objektů. Podstavec lampy je vytvořen z asymetricky zvětšené koule, která byla ze spodní strany oříznuta kvádrem. V materiálu je použit normálový vzor `bumps`, který vytváří hrbolatý vzhled. Stopka tvaru vysouvací antény je vytvořena objektem `sphere_sweep`, který je pro tento tvar velmi vhodný. Materiál stopky je definován jako chrom, tedy bezbarvý, zcela odrazivý povrch. Tvar svítlny je totožný s podstavcem, jen v menším měřítku a lehce pootočený. Dále byl definován objekt `zarovka`, který má tvar válce. Materiál tohoto objektu je upraven modifikátorem `ambient`, který

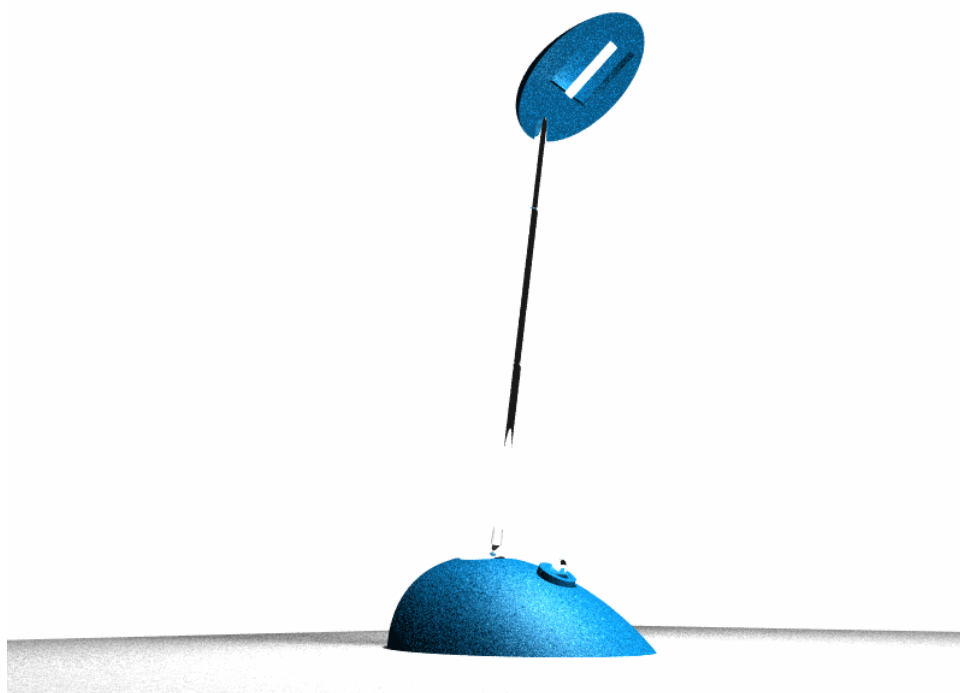
zajistí, že objekt září, i když není osvětlen. Tím byl vytvořen efekt, svítící žárovky. V modelu byl definován plošný zdroj světla, který je znázorněn přidruženým objektem – již zmíněnou žárovkou.

Zdrojové soubory:

lampa.pov

lampa_materials.inc

Vyrenderovaná scéna:



6.5 Místnost

Tento model je tvořen místností, do které jsou zakomponovány všechny výše uvedené modely. Jsou zde použity dva zdroje světla. Jednak světlo definované v modelu Stolní lampa a také bodové světlo, které slouží k prosvětlení scény. Toto světlo nevrhá stín, je zde totiž použit modifikátor `shadowless`. V tomto modelu je definováno několik kamer (při renderování však lze použít vždy max. jednu), díky kterým lze zobrazit detaily některých prvků scény. Dále je zde použit před-definovaný objekt `sky_sphere`, kterým lze simulovat nebe včetně mraků. Jednotlivé modely jsou do této scény vloženy pomocí direktivy `#include`.

Zdrojové soubory:

scene.pov

mistnost.pov

mistnost_materials.inc

stresni_okno.pov

stresni_okno_materials.inc

Vyrenderovaná scéna:



6.6 Vyhodnocení vlivu kvality výstupu a použitých objektů na dobu renderování

V průběhu vytváření modelů jsem zkoušel různá nastavení renderování. Při použití vyhlazování hran (antialiasing) byly výstupní obrazy sice velmi kvalitní, avšak renderování trvalo velmi dlouhou dobu. Vyhlazování hran bylo použito u modelů:

scena.pov – plošný zdroj světla o 5x10 prvcích, bodový zdroj světla nevrhající stín

hodiny.pov – plošný zdroj světla o 5x5 prvcích

dama.pov – bodový zdroj světla, průsvitné materiály

stul.pov – bodový zdroj světla

U modelů **scena.pov** a **hodiny.pov** bylo použito plošného zdroje světla, který je na výpočet scény dosti náročný. V modelu **dama.pov** je hodně průsvitných materiálů, které způsobují emisi nových testovacích paprsků, což rovněž ovlivňuje dobu renderingu. Model

stul.pov neobsahuje žádné průsvitné materiály ani plošný zdroj světla. Obě varianty obrazů lze srovnat na přiloženém CD.

	vyhlazování hran zapnuto	vyhlazování hran vypnuto
scena.pov	3 hod 9 min	2 min 58 sec
hodiny.pov	1 hod 8 min	1 min 9 sec
dama.pov	30 min	30 sec
stul.pov	6 min	7 sec

Tab. 1 – Porovnání délky doby renderování

Nastavení výstupu při zapnutém vyhlazování hran:

Width=800

Height=600

Antialias=On

Antialias_Threshold=0.0

Sampling_Method=2

Antialias_Depth=3

ZÁVĚR

Program POV-RAY je velmi kvalitní renderovací nástroj. Jeho obrovskou výhodou je, že je zdarma a lze jej používat v souladu s podmínkami pro "OpenSource" software. Díky tomu, že program vytváří velmi kvalitní obrazy, byly v jiných, často i komerčních, programech implementovány funkce pro export scény právě do programu POV-RAY. V programu lze ve scéně definovat proměnné, používat makra, cykly apod., což v je v mnoha případech velkou výhodou. Drobnou překážkou v použití programu může být absence grafického rozhraní a tudíž nutnost znalosti syntaxe, avšak ta není nijak složitá a zvládne ji i začátečník.

V teoretické části jsem shrnul informace o vzniku programu, jeho možnostech a dále jsem se snažil popsat základní nastavení a prvky scény tak, aby byl každý student schopen vytvořit svůj vlastní model.

V praktické části jsem připravil několik jednoduchých modelů, ve kterých byly, alespoň z části, demonstrovány možnosti tohoto programu. Poté jsem vytvořil scénu, do které jsou jednotlivé modely zakomponovány. Nakonec jsem naznačil závislost délky doby renderování na použitých materiálech, zdrojích světla a nastavením scény. Zdrojové kódy jednotlivých modelů, včetně vyrenderovaných obrazů, jsou dostupné na přiloženém CD, na kterém se nachází také digitální verze této práce.

Doufám, že tato práce pomůže studentům pochopit základy, které jsou nutné k úspěšnému používání programu POV-RAY.

Martin Mikulka

SEZNAM POUŽITÉ LITERATURY

- [1] Grafika On-line [online]. [cit. 2005-06-05]. Dostupné z URL:
<<http://www.grafika.cz/>>
- [2] POKORNÝ, P., *Cvičení z počítačové grafiky – BLENDER, POV-Ray*.
Skripta FT UTB ve Zlíně, Academia centrum Zlín, 2005, 96s, ISBN 80-7318-264-5
- [3] POV-Ray : Documentation [online]. [cit. 2005-06-05]. Dostupné z URL:
<<http://www.povray.org/documentation/>>
- [4] PovRay [online]. [cit. 2005-06-05]. Dostupné z URL:
<<http://astronomy.swin.edu.au/~pbourke/povray/>>
- [5] POV-Ray 2.2 [online]. [cit. 2005-06-05]. Dostupné z URL:
<<http://home.pf.jcu.cz/~edpo/povray/povray.html>>
- [6] Povray Resources [online]. [cit. 2005-06-05]. Dostupné z URL:
<<http://povray.unzworld.com/>>
- [7] The Online POV-Ray Tutorial [online]. [cit. 2005-06-05]. Dostupné z URL:
<<http://library.thinkquest.org/3285/>>
- [8] The POV-Ray Include Files Page [online]. [cit. 2005-06-05]. Dostupné z URL:
<<http://www.geocities.com/SiliconValley/Lakes/1434/>>
- [9] The World of POV-Ray [online]. [cit. 2005-06-05]. Dostupné z URL:
<<http://www.povworld.org/>>
- [10] Úvod do programu POV-Ray [online]. [cit. 2005-06-05]. Dostupné z URL:
<<http://mdg.vsb.cz/jdolezal/Pgrafika/Prednaska/UvodPOVRay.html>>
- [11] ŽÁRA, J., BENEŠ, B., FELKEL, P., *Moderní počítačová grafika*.
1. vydání, Computer Press Praha, 1998, 448s, ISBN 80-7226-049-9

SEZNAM OBRÁZKŮ

Obr. 1 – Levoruký souřadný systém v programu POV-RAY	16
Obr. 2 – Schematické znázornění parametrů perspektivní projekce	20
Obr. 3 – Kuželový svazek světla	23
Obr. 4 – Popis funkce adaptivního mechanismu	26
Obr. 5 – Objekt blob	28
Obr. 6 – Konstrukce výškové plochy ze zdrojového obrázku	31
Obr. 7 – Konstrukce objektu sor rotací 2D grafu funkce kolem osy y	37
Obr. 8 – Objekt torus a jeho parametry	38
Obr. 9 – CSG operace - sjednocení.....	39
Obr. 10 – CSG operace - průnik	40
Obr. 11 – CSG operace - rozdíl	40
Obr. 12 – CSG operace - sloučení	40
Obr. 13 – Modifikátor Clipped_by	41

SEZNAM PŘÍLOH

- P I Dokumentační CD obsahující digitální formu této práce. Dále obsahuje zdrojové kódy jednotlivých modelů a vyrenderované obrazy.