Laskavý průvodce virtuálními světy

aneb

Praktická příručka jazyka VRML 97

Vážený čtenáři,
dostává se ti do rukou knížka, s níž nahlédneš do zákulisí systémů pro virtuální realitu, založených na používání jazyka VRML 97. S pomocí *Laskavého průvodce* se naučíš vytvářet vlastní virtuální světy a uvidíš, že s pomocí jednoduchých a šikovných postupů lze virtuální svět snadno rozhýbat a obohatit o nejrůznější dynamické efekty.

Jazyk VRML 97, který je mezinárodní normou ISO pro popis statických a dynamických světů (ISO/IEC 14772-1:1997), je oficiálně definován technickou specifikací v takovém rozsahu a podrobnostech, že pro běžného čtenáře není vůbec snadné se jím prokousat a pochopit jej. *Laskavý průvodce* je proto napsán tak, aby čtenáře citlivě vedl od prvních krůčků ve virtuálním světě až po profesionální návrh složitých a dynamických virtuálních scén.

Množství příkladů dokumentuje jednotlivé postupy při vytváření virtuálních světů, důraz je kladen na jednoduchost a srozumitelnost. Všechny příklady jsou plně funkční a mohou se tak stát základními kameny budoucích světů, vzniklých v myslích a počítačích čtenářů – tvůrců.

K přehlednosti a lepšímu pochopení napomáhá textové uspořádání *Laskavého průvodce*. Příklady, které lze přímo použít jako virtuální světy, jsou zapisovány do modrošedých okének. Klíčová slova a příkazy jazyka VRML 97 jsou v textu průběžně zvýrazňovány **`odlišným písmem`**. Tam, kde je nutno zavést nové pojmy a speciální hodnoty, nalezneme tabulky es žlutým podbarvením. Naprostou většinu černobílých obrázků z textu nalezneme v barevné podobě v obrazové příloze. S výjimkou schematických náčrtků představují všechny obrázky pohledy do virtuálních světů skrz okno obrazovky a uvidíme na nich právě ty světy, které jsou zapsány v příkladech. Na mnoha místech se pak objevují **TIPY** – doporučení a rady, které bychom těžko hledali v oficiálních specifikacích, protože pocházejí přímo od lidí, kteří mají s VRML 97 praktické zkušenosti.

Knížka je doplněna WWW stránkami, které jsou umístěny na adrese:

http://www.cgg.cvut.cz/LaskavyPruvodce

Na těchto stránkách jsou sdruženy všechny obrázky a příklady z knihy. Nechybí ani odkazy na kompletní oficiální specifikaci jazyka VRML 97 a na nejznámější programy pro prohlížení virtuálních světů – CosmoPlayer a WorldView.

Jiří Žára

(autor přednáší problematiku virtuální reality na Katedře počítačů FEL ČVUT v Praze a podílel se na vzniku mezinárodní normy ISO – jazyka VRML 97)

Obsah

1 Virtuální realita - Neskutečné skutečno!

Virtuální realita se velmi rychle dostává z oblasti vědecko-fantastických filmů na obrazovky počítačů vědeckých pracovníků, techniků i běžných uživatelů. Po počátečním překvapení z nového pojmu *virtuální realita* se většina z nás již ani nepozastavuje nad nesmyslným spojením dvou slov s opačným významem (*virtual* = fiktivní, neskutečný, zdánlivý; *real* = skutečný, pravý) a správně si vysvětlujeme virtuální realitu jako prostředí, které umožňuje práci v trojrozměrném prostoru, vymodelovaném v paměti počítače.

Základem virtuální reality jsou postupy, které nacházíme po řadu let v oboru počítačové grafiky. Jde zejména o tvorbu prostorových modelů a scén, manipulaci s nimi, pohyb v trojrozměrném prostoru a zobrazování v reálném čase. Tyto metody bývají umocněny použitím speciálních periferií, které zajišťují obrazovou, zvukovou a hmatovou (polohovou) interakci. Jde zejména o helmy se zabudovanými displeji, snímače polohy v prostoru, dotyková zařízení, simulační kabiny, apod. Pomocí takových zařízení přináší virtuální realita nevšední zážitky a nabízí nové a nečekané možnosti uplatnění výpočetní techniky.

Důležité přitom je, že virtuální realitu můžeme realizovat i bez speciálních zařízení, byť v pocitově méně intenzívní podobě. Normální obrazovka osobního počítače se může stát průzorem do virtuálního světa, v němž se pohybujeme pomocí klávesnice či myši.

Programy z oblasti virtuální reality se vyznačují následujícími rysy:

1. veškeré děje se provádějí **v reálném čase**, tedy pokud možno s okamžitou odezvou na vstupní aktivitu uživatele,
2. umělý svět a objekty v něm mají **trojrozměrný charakter** nebo alespoň vytvářejí jeho iluzi,
3. uživatel neprohlíží virtuální svět jenom zvenčí, ale **vstupuje do něj** a pohybuje se v něm po rozličných drahách (chodí, létá, skáče a mžikem se přesouvá – teleportuje se),
4. svět není statický, s jeho částmi uživatel manipuluje. Virtuální tělesa často působí jako samostatné bytosti - pohybují se po animačních křivkách, **interagují** s uživatelem i mezi sebou navzájem.

Největší důraz je kladen na první z těchto charakteristik. Zobrazování umělého (virtuálního) světa v *reálném čase* je natolik klíčové, že je mu podřizována i kvalita zobrazení. Proto nám mohou obrázky z některých systémů pro virtuální realitu připadat hrubé, nehezké. Překotný vývoj rychlosti procesorů a dostupnost kvalitních grafických karet pro prostorovou grafiku však mohou kvalitativně zlepšit vzhled virtuálních světů až nečekaně brzy. A jestliže éra široce používané virtuální reality dosud nenastala, lze bez obav říci, že právě čeká za dveřmi.

Tato kniha je věnována jazyku **VRML** (**Virtual Reality Modeling Language**), který je určen pro popis obsahu virtuálních světů a jejich chování. V této kapitole si nejprve řekneme, co všechno jazyk VRML dokáže a jaké jsou jeho základní vlastnosti. V kapitole 2 (*Ztraceni ve virtuálních světech?*) se seznámíme s tím, jak se pohybovat ve virtuálním prostředí. O tom, jak se popisují jednotlivé virtuální objekty, se dozvíme v kapitole 3 (*Jednoduché světy*), další speciální metody jsou uvedeny v kapitole 4 (*Chci mít originál*).

Jakmile budeme schopni vytvořit jednoduchý statický svět, zaměříme se na to, jak jej oživit. Kapitola 5 (*Události hýbou světem*) přináší přehled metod, zajišťujících animaci a interakci ve virtuálním prostředí. Nejdůmyslnější kousky ve virtuální realitě jsou pak uvedeny v kapitole 6 (*Jen pro opravdové programátory?*).

Praktické rady a doporučení pro zajištění efektivity jsou obsahem kapitoly 7 (*Ach, ty pomalé počítače!*). Předposlední kapitola 8 (*Kam kráčíš, virtuální realito?*) uvádí směry, kterými se virtuální realita bude dále rozvíjet. Poslední, poměrně rozsáhlá kapitola 9 (*Možná nudné, ale dozajista nutné*) shrnuje a vysvětluje vlastnosti všech prvků jazyka VRML v abecedním pořadí.

Pro první seznámení s virtuální realitou a jazykem VRML stačí přečtení prvních tří kapitol. Jakmile budeme chtít znát veškeré možnosti, které VRML nabízí, pustíme se dále až po osmou kapitolu. Praktičtí tvůrci virtuálních světů pak budou nejčastěji listovat kapitolou devátou.

1.1 Jazyk VRML: doma i na Internetu

Jazyk VRML definuje způsob zápisu virtuálních světů do souborů v textovém tvaru. Je tedy současně i tzv. *formátem*, předpisem pro zapisování informací určitého typu. V tomto smyslu ho můžeme připodobnit k formátům GIF, BMP či JPEG pro zápis obrázků nebo k formátům MPEG či AVI pro zápis videa. Jazyk VRML se přitom vyznačuje důležitou vlastností – nevznikl jako produkt jedné firmy, ale je výsledkem společného úsilí mnoha firem a odborníků z celého světa. Tím byl dán základní předpoklad pro jeho všeobecné přijímání jako univerzálního standardu pro virtuální realitu.

Základní vlastnosti jazyka VRML jsou:

- Virtuální světy tvořené prostorovými objekty jsou kombinovány s multimediálními prvky, jakými jsou obraz, video, zvuk.
- Při tvorbě virtuálních světů lze využívat prvky zapsané jak lokálně v souborech, tak kdekoliv v síti Internet. Stejně tak lze mezi různými světy plynule přecházet podobně jako přecházíme mezi stránkami WWW.
- Animace, interakce a manipulace s virtuálními objekty je zajištěna jednotným a přehledným způsobem. Stejné prostředky se používají pro popis statických i dynamických světů. Statické světy lze snadno rozšířit na dynamické a obráceně.
- Součástí jazyka jsou definice způsobů pohybu uživatele (chůze, let, zkoumání objektů), podpora automatické navigace ve virtuálním prostředí, popis reakce na chování uživatele.
- Virtuální světy lze vkládat do WWW stránek či rámů.
- Jazyk VRML umožňuje spolupráci s dalšími programovacími jazyky (Java, JavaScript) i aktivaci jiných programů, typicky prohlížečů WWW stránek.
- Popis virtuálních světů je ukládán pouze v textovém, tedy snadno čitelném tvaru. Velikost souborů je pak možno výrazně snížit kompresí pomocí programu `gzip`, aniž bychom se museli explicitně starat o jejich zpětné dekódování.

1.1.1 Krátká, ale dynamická historie

Počátky jazyka VRML můžeme najít v místech, kde se rodily systémy moderní prostorové grafiky, totiž ve firmě Silicon Graphics, Inc. Její programátoři navrhli koncem 80. let knihovnu pro práci s prostorovými objekty, nazvanou *Inventor*. Byla vybudována jako nadstavba nad známou a úspěšnou základní grafickou knihovou *GL*. Začátkem devadesátých let došlo k inovaci – vznikla nová základní grafická knihovna s názvem *OpenGL* a k ní nová aplikační knihovna *OpenInventor*. Právě formát, ve kterém se do souborů zapisovala tělesa a scény pro knihovnu *OpenInventor*, se stal základem jazyka VRML.

Za posledních několik let prodělal jazyk VRML obrovský vývoj. Jeho dynamika zjevně souvisí s rychlým rozvojem Internetu, růstem výkonnosti počítačů a voláním uživatelů po univerzálním prostředku pro popis prostorových dat určených pro zobrazování v reálném čase. Podívejme se na jednotlivé kroky vývoje VRML v krátkém časovém přehledu:

- Na podzim roku 1995 definuje firma Silicon Graphics formát **VRML 1.0**, který je rozšířením formátu OpenInventor o možnosti využívání prostorových dat ze sítě Internet, konkrétně v prostředí WWW.
- Současně se vznikem VRML 1.0 je založena nezávislá skupina programátorů a návrhářů, nazvaná *VAG* (***V**RML **A**rchitecture **G**roup*). Skupina stanovuje tři základní milníky potřebného budoucího vývoje jazyka VRML:
  1. prostředky pro popis statických světů (v dané chvíli pokryto formátem VRML 1.0)
  2. prostředky pro popis dynamických světů (nejbližší cíl)
  3. prostředky pro spolupráci více uživatelů ve virtuálním prostředí (vzdálenější cíl)

Skupina VAG oslovuje všechny významné tvůrce systémů pro virtuální realitu, jakož i internetovou veřejnost a odborníky na počítačovou grafiku s výzvou k vytvoření specifikace budoucího jazyka VRML 2.0.

- V dubnu 1996 je vybrána z osmi různých návrhů společná specifikace firem Silicon Graphics a Sony, která má pracovní název *Moving Worlds*. Tato specifikace se stává základem jazyka **VRML 2.0**. Je ve tvaru, který bude po dobu více než jednoho roku upřesňován, měněn a doplňován v otevřené diskusi probíhající průběžně na Internetu (`www-vrml@vrml.org`).

- Z neformální skupiny VAG se stává oficiální sdružení *VRML Consortium, Inc.*, které zahajuje spolupráci s mezinárodní standardizační organizací ISO na vzniku VRML v podobě mezinárodní normy. Ta nese pracovní název DIS 14772-1 a v dubnu 1997 získává jméno **VRML 97**. Po celý rok 1997 je upravována do podoby mezinárodně přijatelné normy ISO.

- Koncem roku je jazyk VRML oficiálně přijat za ISO standard s označením ISO/IEC 14772-1:1997.

Z výše uvedeného přehledu vidíme, že pod zkratkou VRML se můžeme setkat s několika verzemi, navzájem odlišnými. Tato nejednotnost je daní za rychlost, se kterou byl jazyk vyvíjen, a také za otevřenost při jeho upřesňování. Popis jazyka byl totiž neustále veřejně přístupný na Internetu a řada programátorských firem průběžně aktualizovala své prohlížeče nebo konvertory pro převod dat do formátu VRML. Soubory VRML 2.0 vzniklé v té době se liší v méně či více významných detailech.

Přijetím VRML jako standardu ISO skončilo období nejednotnosti formátu a byla zahájena doba jeho intenzivního využívání. Pod názvem **VRML** je nyní chápána jeho mezinárodní verze VRML 97. Setkáme-li se ještě s názvem VRML 2.0, jde pouze o programátorské označení VRML 97. Verze VRML 1.0 je pak z historických důvodů ponechávána po určitou dobu v platnosti, současné prohlížeče většinou obsahují automatický převodník z VRML 1.0 do VRML 97. Postupně bude verze VRML 1.0 zcela opuštěna.

V dalším textu budeme pod názvem VRML vždy mínit VRML 97.

1.1.2 K čemu se hodí VRML

Vznik VRML znamenal zásadní zlom v prezentaci prostorových dat na běžných počítačích. Zatímco donedávna bylo zpracování prostorových dat vyhrazeno speciálním a drahým systémům CAD nebo náročným programům pro triky a animace, s příchodem jazyka VRML se dostávají virtuální objekty a světy doslova na stůl každého uživatele osobního počítače.

Velikou výhodou VRML je jeho orientace na Internet, resp. WWW. Každý, kdo je na Internet připojen, může okamžitě vstupovat do virtuálních prostorů, zkoumat prostorová data, případně si je ukládat na svůj disk a dále je zpracovávat. Oba nejznámější producenti WWW prohlížečů (Netscape a jeho Navigator, Microsoft a jeho Internet Explorer) dodávají současně s prohlížečem WWW stránek i prohlížeč VRML světů. Nejeden uživatel je mile překvapen, když na své obrazovce spatří virtuální světy, aniž předtím musel instalovat jakýkoliv (drahý) program.

Snadná dostupnost dat (VRML souborů) a prohlížečů je předpokladem pro uplatnění VRML v dlouhé řadě aplikací. Počet lidí, kteří začnou přemýšlet „prostorově“ a budou pracovat s trojrozměrnými daty, poroste v příštích letech nevídaným tempem.

Teoreticky si můžeme představit uplatnění VRML v téměř každé počítačové aplikaci – někde je však jeho využití obzvlášť přirozené. V barevné příloze knihy jsou uvedeny příklady typických aplikačních oblastí, zahrnující architekturu, informační systémy, elektronické obchodování, reklamu, umění, vizualizaci dat aj. Tento výčet je přitom jen krátkou přehlídkou širokých možností, ve kterých nacházíme VRML. Sympatické je, že virtuální světy nemusí být omezeny konkrétní aplikací, ale mohou se snadno kombinovat a prolínat.

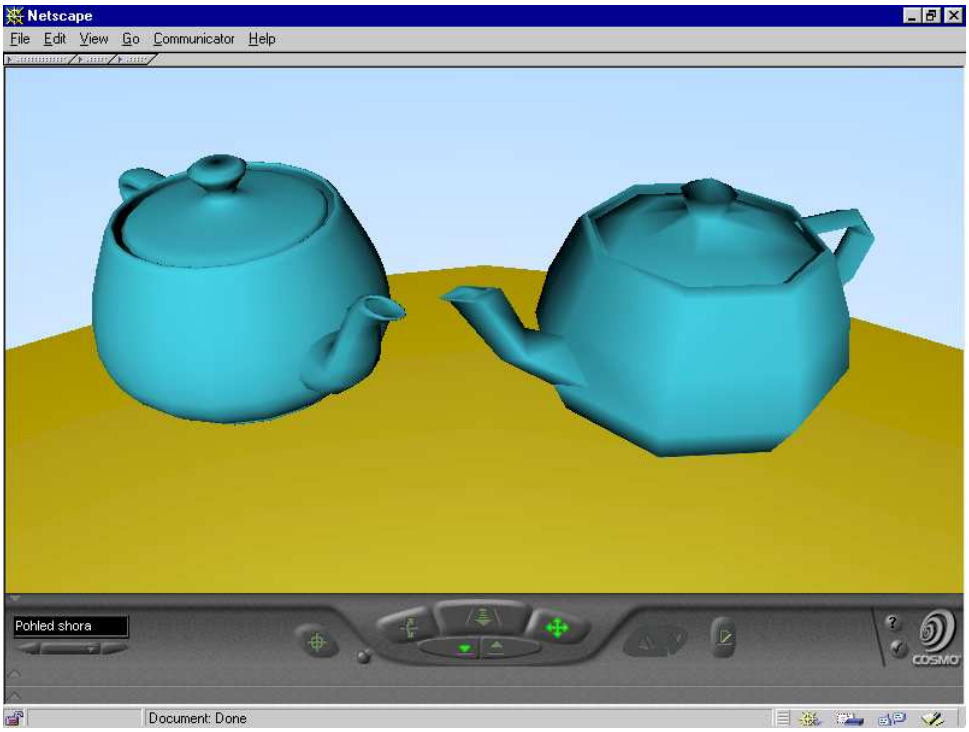
Snadno si potom dokážeme představit firmu, která na Internetu nabízí své výrobky v podobě prostorových modelů, případnému zájemci ukáže prostorový graf úspěšnosti prodeje daného výrobku či poklesu jeho ceny za minulé období a na závěr mu nabídne pobyt ve virtuální herně-obchodu, kde po proběhnutí všemi zákoutími plnými reklam na báječné výrobky nalezne v závěru poukázku na (virtuální?) slevu.

1.1.3 Jak si prohlédneme soubor VRML?

Prohlížeč VRML je program, který je schopen převést textový popis z VRML souboru do obrazu virtuálního světa. Navíc nám umožní pohyb v tomto světě a případnou interakci s virtuálními předměty. Vzhledem

k úzkému vztahu VRML a WWW je většina dostupných prohlížečů VRML souborů součástí prohlížečů WWW stránek. Samostatné prohlížeče jsou výjimkou, případně součástí programů pro vytváření VRML světů.

Vzhled a ovládání prohlížečů nejsou pochopitelně stejné. Většina z nich mívá v dolní části ovládací panel, s jehož pomocí se pohybujeme ve virtuálním prostoru. Přepínáním ovládacích prvků volíme, zda aktivita ukazovacího zařízení (myši) má být převáděna na pohyb v prostoru nebo na manipulaci s virtuálními objekty. Obrázek 1-1 je příkladem vzhledu prohlížeče CosmoPlayer (od firmy Silicon Graphics, Inc.) pro operační systém MS Windows, dodávaným s programem Netscape Communicator. Dalším dobrým prohlížečem je WorldView od firmy Intervista Software, Inc., který je dodáván spolu s programem Internet Explorer. Oba uvedené prohlížeče VRML lze přitom provozovat v rámci libovolného z obou zmíněných prohlížečů WWW stránek.



Obrázek 1-1: Vzhled prohlížeče CosmoPlayer pro MS Windows

1.2 Struktura souboru VRML

V každém souboru, obsahujícím popis světa v jazyce VRML, můžeme nalézt několik logicky odlišných částí, jak ukazuje tabulka T-1-1. Na prvním řádku je vždy umístěna hlavička souboru, jejíž tvar je neměnný. Podle ní rozpoznávají aplikační programy, o jaký typ souboru a jakou verzi jde. V hlavičce souborů VRML je dokonce zapsán i způsob kódování znaků tak, aby jeden soubor mohl obsahovat písmena z několika různých národních abeced současně. Zkratka **`utf8`** říká, že je použito kódování UTF-8 (jinak známé též pod názvem Unicode, viz [5]), které pro zápis základní sady znaků využívá běžným způsobem osm bitů, pro znaky národních abeced pak dvojice až šestice bytů pro jeden znak.

`#VRML V2.0 utf8`	Hlavička souboru VRML.
`WorldInfo { ... }` `Viewpoint { ... }`	Úvodní, všeobecné informace o virtuálním světě.

`Transform { ... }`
`Group { ... }`
`PositionInterpolator { ... }`
	Popis těles, jejich vlastností, definice prvků potřebných pro animace a interakci.

`ROUTE ... TO ...`	Propojení dynamických a statických prvků z předchozí části.

Tabulka T-1-1: Doporučené logické členění VRML souboru

Hned za hlavičkou nejčastěji následují údaje, které popisují celkové vlastnosti virtuálního světa. Bývají to informace o souboru a jeho tvůrci (**`WorldInfo`**), seznam zajímavých míst uvnitř světa (**`Viewpoint`**) a doporučený způsob procházení světem (**`NavigationInfo`**).

Třetí, většinou nejrozsáhlejší část souboru je tvořena popisem virtuálních těles, definicí jejich tvaru, barvy a dalších vlastností. Patří sem i zápis prvků, s jejichž pomocí bude statický virtuální svět „rozhýbán“. Do poslední části VRML souboru se pak zapisují propojení mezi aktivními a statickými prvky. Uvedené pořadí logických částí souboru není závazné, patří však mezi dobré programátorské a autorské zvyky.

TIP: Znak ‘#’ uvnitř souboru (s výjimkou prvního řádku) označuje tzv. komentář – vše za ním až do konce řádku je chápáno pouze jako poznámka, nikoliv jako součást definice virtuálního světa.

Soubory, které obsahují VRML světy, mají příponu `wrl` (z anglického *world*). Tuto příponu mají i v případě, že jejich obsah byl nejprve zkomprimován programem `gzip`. Prohlížeče jsou schopny podle obsahu souboru rozpoznat, zda jde o textový či komprimovaný tvar. Pokud chceme z komprimovaného tvaru získat tvar textový, je vhodné soubor nejprve přejmenovat (např. z **`'cosi.wrl'`** na **`'cosi.wrl.gz'`**) a poté jej předložit programu `gzip` či `winzip` k dekódování.

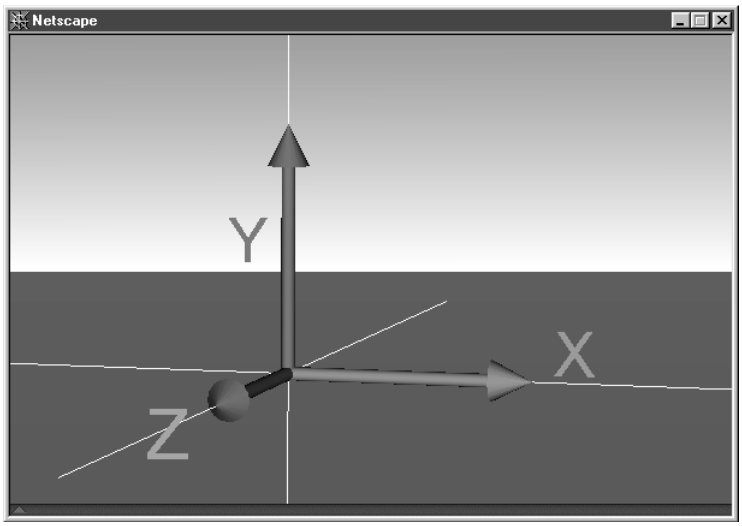
2 Ztraceni ve virtuálních světech?

Jedním z důvodů, proč neznalí lidé neradi navštěvují virtuální (rozuměj trojrozměrné) prostředí, je nejistota, se kterou se v něm člověk pohybuje a nezvyklé metody manipulace s prostorovými objekty. Zatímco při používání systému MS Windows či X-Window se lidé snadno ztotožní s představou, že na obrazovce vidí část pracovního stolu s různými dokumenty, které lze ovládat (přesouvat, aktivovat, měnit), virtuální realita představuje přístup zcela odlišný od výše uvedené symboliky. Člověk skrze obrazovku vstupuje do trojrozměrného prostoru (srovnej s MS Windows: „Cožpak lze vstoupit do hloubi dokumentu, tabulky, obrázku?“) a za pomoci myši mění svoji polohu a dokonce manipuluje s virtuálními tělesy („Jak se mohu myší položenou na ploché podložce přesouvat v trojrozměrném světě, jak s ní otočím, zvednu a zahodím nějakou věc?“).

Tyto principiálně nové způsoby práce s počítačem představují psychologickou bariéru pro první generaci uživatelů MS Windows, jsou však naštěstí běžné pro hráče počítačových her a tedy pro generaci nastupující. Přesto je ve VRML kladen důraz na to, aby se člověk ve virtuálním prostoru cítil pokud možno přirozeně a aby se s umělými světy pracovalo co nejjednodušeji.

2.1 Dejte mi pevný bod … (NavigationInfo, Viewpoint, WorldInfo)

V některých, zejména uměleckých a filmových aplikacích virtuální reality je chápáno jako obrovská výhoda, že se člověk ve virtuálním prostoru může snadno kamkoliv přesunout, procházet zdmi, být osvobozen od zemské gravitace a konečně tedy být „svobodný a volný jako pták“. Ve skutečnosti je však velmi dobré, když se návštěvník virtuálního světa může dobře orientovat, po případném romantickém letu přistát na pevné virtuální zemi a jednoznačně usoudit, zda zeď protějšího domu je postavena kolmo vzhůru nebo zda hrozí její zřícení. Stejně tak není marné vědět, že pokud při své cestě virtuálním prostorem dojdeme například ke stolu, budeme ho muset obejít, protože program nás skrze stůl nepustí. Představa, že procházíme skrz masivní dřevěnou desku, je přeci jen poněkud nezvyklá. Obecně ovšem lze průchod virtuální hmotou povolit a umožnit tak brodění v bahně nebo průchod křovím.



Obrázek 2-1: Orientace ve virtuálním prostoru

Systém prostorových souřadnic je ve VRML orientován podle obrázku 2-1. Směr vzhůru je totožný s kladným směrem osy *y*. Země či podlaha, po které se běžně pohybujeme, je rovnoběžná či přímo totožná s rovinou *xz*. Iniciálně míří kladná poloosa *x* doprava, kladná poloosa *z* směřuje k nám. Díváme se tedy ve směru záporné osy *z*.

Abychom mohli snadno navštívit všechna zajímavá místa daného umělého světa, mohou nám jeho tvůrci nabídnout seznam stanovišť (míst v trojrozměrném prostoru) včetně doporučených směrů pohledů a případných charakteristik pomyslné optiky, skrze kterou do světa hledíme. Každé takové stanoviště se nazývá **`Viewpoint`**. Při výběru libovolného nového stanoviště nás program automaticky dopraví ze stávajícího místa na místo nové, nejčastěji plynulým přeletem uvnitř virtuálního prostoru. Také směr pohledu se při přeletu postupně mění.

```
#VRML V2.0 utf8
Viewpoint {
  position        0 0 10
  orientation     0 0 1 0
  description     "Zepredu"
  fieldOfView     0.785398
}
Viewpoint { description "Zprava a zblizka"
            position 3 0 0 orientation 0 1 0 1.57
            fieldOfView 1.57
}
DEF CELKOVY Viewpoint
  { description "Zdaleka" position 0 0 123 }
```

Program P-2-1: Zadávání stanovišť

Soubor v ukázce P-2-1 definuje tři stanoviště. Jsou pojmenována **"Zepredu"**, **"Zprava a zblizka"** a **"Zdaleka"**. První z nich umisťuje návštěvníka do základní polohy na osu *z*, 10 metrů od počátku soustavy souřadnic. Jeho pohled míří vždy automaticky do záporné osy *z*, takže specifikace orientace pohledu není v tomto případě vůbec potřeba a je uvedena jen pro úplnost. Také šířka záběru (parametr **fieldOfView**) má zde implicitní hodnotu 45º, tedy π/4.

Druhé stanoviště je umístěno doprava, na osu *x*. Díváme se z něj do počátku soustavy souřadnic, proto je parametr **orientation** nastaven pomocí čtyř hodnot tak, aby se směr pohledu otočil kolem svislé osy *y*, tj. kolem vektoru [0 1 0] o úhel 90º, tj. zhruba 1,57 radiánů. Šířka záběru je zde větší, což zvýrazňuje výslednou perspektivní projekci. Při ní se jeví vzdálenější předměty menší a blízké větší.

Je důležité vědět, že konkrétní velikost objektů na obrazovce nelze přesně zadat. Šíře záběru je vždy přepočítávána s ohledem na velikost okna na obrazovce. Při změně velikosti okna se okamžitě přepočítá obraz, takže rozměry a poloha objektů na obrazovce se změní. Nezmění se však vzájemné proporce mezi objekty. To, co vidíme v malém okénku, spatříme v okně roztaženém na celou obrazovku zcela stejně, byť v jiném měřítku.

TIP:

1. **Větší šířka záběru (fieldOfView) umožní zobrazit více těles, ale na okrajích obrazovky způsobí výrazná perspektivní zkreslení.**
2. **Menší šířka záběru připomíná rovnoběžné promítání a dovolí zvětšit obraz těles, aniž bychom se k nim museli příliš přiblížit.**

Poslední, třetí stanoviště, určuje pohled z dálky. Ve velké míře využívá přednastavených hodnot parametrů a definuje pouze popisný text pro stanoviště a jeho polohu. Od předchozích stanovišť se dále odlišuje tím, že je pojmenováno. Zápisem **DEF** je mu přiřazeno jméno **CELKOVY**. Později uvidíme, že pojmenování je základní podmínkou potřebnou pro rozličné manipulace a animace částí virtuálního světa.

TIP:

1. **První stanoviště definované v souboru nastavuje iniciální pohled při vstupu do virtuálního světa**
2. **Jakmile pojmenujeme libovolné stanoviště příkazem DEF, lze zahajovat prohlídku světa též z něj, pokud se do něj přeneseme (teleportujeme) z jiných virtuálních světů.**

Jak je vidět na ukázce P-2-1, každý z prvků jazyka VRML má své jméno (např. **Viewpoint**) a několik parametrů (např. **position** či **description**). Od této chvíle budeme hlavní prvky jazyka VRML nazývat ***uzly*** (angl. *node*). Pro zápis jejich ***parametrů*** (angl. *field*) platí následující pravidla:

- Na pořadí parametrů uvnitř uzlu nezáleží
  (srovnej pořadí parametrů **position** či **description** v definici prvého a druhého stanoviště)
- Každý parametr má definovánu implicitní hodnotu danou mezinárodní specifikací ISO. Stačí tedy zapisovat pouze ty parametry, jejichž hodnoty jsou odlišné od implicitních. Zápis zcela všech parametrů a hodnot snižuje čitelnost a zbytečně zvyšuje velikost souboru.
- Počet mezer a řádků uvnitř definice uzlu není důležitý, hraje pouze estetickou roli
  (zápis na několika řádcích je považován za nejpřehlednější - viz stanoviště **"Zepredu"**)

- Názvy uzlů začínají velkým písmenem, názvy parametrů malým. Zbytek názvu pak může obsahovat malá i velká písmena.

- Každému *uzlu* lze přiřadit individuální jméno příkazem **DEF**. *Parametr* uzlu však pojmenován být nemůže.

TIP: Vzdálenosti se ve VRML zadávají v metrech, čas v sekundách a úhly v radiánech. Barva se popisuje trojicí hodnot RGB, každá je v rozsahu 0-1.

2.2 Co udělám s tímto fantem (tedy s myší)?

Většina počítačů je vybavena pouze dvěma zařízeními, umožňujícími zadávat vstupní údaje potřebné pro ovládání programů – klávesnicí a myší. Musíme tedy s nimi vystačit i pro práci ve virtuálním prostoru. Myš je přitom preferována, neboť s její pomocí se většinou dokážeme pohybovat na ploše obrazovky rychleji. Zkušení uživatelé pak rádi doplňují práci s myší rychlým přepínáním režimů pomocí klávesnice.

Podívejme se, co vše bychom s myší měli dokázat:

1. přesunout se v prostoru na nějaké místo,
2. podívat se libovolným směrem,
3. vybrat objekt, se kterým chceme pracovat,
4. manipulovat s vybraným objektem – otáčet jej, přesouvat, deformovat atd.

První dvě akce se týkají polohy návštěvníka virtuálního světa a pohyb myši představuje pohyb jeho nohou, resp. celého těla. Zbylé dvě aktivity jsou jednodušší – práce s myší při nich odpovídá již víceméně zavedenému „ručnímu“ ovládání objektů na obrazovce. Často se pracuje s trvale stisknutým tlačítkem myši. Je-li takto přesouván kurzor ve volném prostoru, mění se poloha a natočení návštěvníka.

Přesuneme-li volně kurzor nad obraz nějakého virtuálního objektu, může se tvar kurzoru změnit. Znamená to, že daný objekt je schopen reagovat na stisknutí myši vlastní akcí. Některé prohlížeče mění tvar kurzoru i podle toho, jakým způsobem se v prostoru přesouváme.

2.3 Avatarovo velitelské stanoviště

Neděste se, nadpis kapitoly není titulem ze sbírky vědecko-fantastických povídek. Pouze uvádí dva pojmy, které jsou typické pro práci ve virtuálním prostředí:

Avatar

je pojmenování virtuálního dvojníka[1], který představuje nás samé uvnitř virtuálního světa. Zatímco my sedíme před obrazovkou, pomyslný avatar se proplétá umělým světem. Lze říci, že okno prohlížeče je právě tím, co avatar vidí. V této abstrakci platí, že chceme-li se podívat doleva, natočíme doleva svého avatara.

Je přitom důležité, že avatar má určité rozměry, které mu brání procházet extrémně malými průchody. V interaktivně navržených světech lze detekovat, v jakém místě virtuálního světa avatar stojí, co vše vidí, zda naráží na překážku nebo zda vstupuje do nějaké hlídané zóny.

V dalším textu tedy můžeme bez obav zaměňovat pojmy *návštěvník* a *avatar*, protože jejich chování ve virtuálním prostředí je totožné.

[1] Jméno *avatar* pochází z hinduistické mytologie, kde označuje dočasnou tělesnou schránku, do které se vtěluje Bůh při své návštěvě Země.

Velitelské stanoviště

není nic jiného, než systém pro ovládání avatara. V prohlížeči je tento systém reprezentován prvky na ovládacím panelu aplikace.

Jak vlastnosti avatara, tak některé prvky velitelského stanoviště lze zadat pomocí uzlu, nazývaného **`NavigationInfo`**. Předdefinované hodnoty jeho parametrů definují avatara, jehož vlastnosti jsou podobné člověku. Proto bývá přímé použití tohoto uzlu s jinými parametry vzácné.

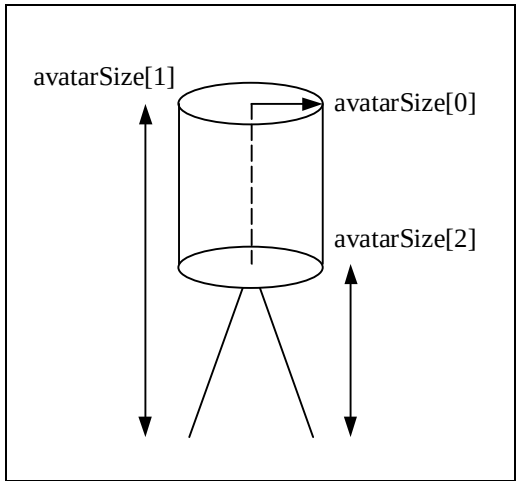
Je-li v souboru více navigačních uzlů, použije se právě první z nich. Pouze jeden takový uzel může být aktivní. Podle této vlastnosti říkáme, že uzel je *vázaný* na konkrétní stav prohlížeče. Přepnutí mezi vázanými uzly je možné a často bývá prováděno současně s přechodem na nové stanoviště. Dostaneme-li se například do podzemního světa trpaslíků, je nutno zmenšit avatara tak, aby mohl prolézt štolou. O tom, jak aktivovat určitý uzel nastavením jeho speciálního parametru **`set_bind`**, se dozvíme v kapitole 5 (Události hýbou světem).

parametr	iniciální hodnota	význam
`avatarSize`	**`[0.25, 1.6, 0.75]`**	rozměrové charakteristiky avatara
`headlight`	**`TRUE`**	zapnuté čelní světlo (baterka)
`visibilityLimit`	**`0.0`**	dohled v metrech, nulová hodnota značí nekonečno
`speed`	**`1.0`**	rychlost pohybu v m/s
`type`	**`["WALK", "ANY"]`**	nastavení způsobů pohybu pro velitelské stanoviště

Tabulka T-2-1: Parametry uzlu **`NavigationInfo`**

První čtyři parametry se týkají avatara. Trojice rozměrů **`avatarSize`** neurčuje míry přes prsa, pas a boky, nýbrž postupně:

1. Maximální povolenou vzdálenost, na kterou se smí avatar přiblížit k překážce.
   Jinými slovy – avatarovo tělo je tenká tyčka obklopená kolkolem bezpečnou zónou[2]. Běžný avatar tak projde dveřmi, které mají šířku nejméně 50 cm.
2. Výšku očí nad okolním terénem měřenou v ose *y*. Terén je to, na čem avatar stojí. Nemusí to být vždy jen základní rovina *xz*, ale jakékoliv virtuální těleso, na které se avatar může dostat – schody, jiné patro, stolička.
3. Maximální výšku překročitelné překážky. Avatar je schopen malou překážku překročit a na větší vystoupit. Tímto způsobem lze s avatarem bez jakékoliv námahy chodit po schodišti. Iniciální hodnota 75 cm zaručuje chůzi i po extrémně strmých schodech.



Obrázek 2-2: Geometrický význam parametru **`avatarSize`** *(složky jsou číslovány od nuly)*

Druhý parametr je důležitý pro zkoumání neosvětlených virtuálních světů. V řadě aplikací slouží VRML pouze pro modelování prostorových objektů a tvůrci se nezabývají jejich osvětlením. Kde není osvětlení, není nic vidět. Proto je každý avatar vybaven čelním světlem (angl. *headlight*) podobně, jako když mají horníci na helmách připevněny malé reflektory. Čelní baterka sleduje směr pohledu avatara a ozařuje bílým světlem vždy tu část světa, kterou právě prohlížíme. Tato sympatická baterka-čelovka má neomezený světelný dosah. Její vypnutí nastavením na hodnotu **`FALSE`** se používá tam, kde je umělý svět záměrně osvětlen dalšími zdroji světla, případně doplněn světelnými efekty a překvapeními.

[2] V jiném přirovnání je avatar reprezentován svislým válcem, postaveným na pomyslných nohách (viz též obrázek 2-2). První rozměr v parametru **`avatarSize`** určuje poloměr válce, druhý výškové umístění jeho horní podstavy, v jejímž středu jsou položeny oči. Třetí rozměr udává výškové umístění dolní podstavy válce, pod níž se nacházejí avatarovy nohy. Pomyslná chodidla stojí na zemi, tj. v rovině *y=0*. Nepříliš pohledná postava, že?

Zrakové schopnosti avatara jsou obecně dokonalé, teoreticky je schopen vidět do nekonečna. Z praktického hlediska je však vhodné nastavit parametr **visibilityLimit** na hodnotu několika desítek metrů, protože prohlížeč pak nemusí vzdálenější objekty vykreslovat.

Poslední charakteristikou avatara je jeho rychlost. I ta hraje svoji roli při orientaci v prostoru, protože podle času stráveného chůzí kolem virtuální stavby umíme odhadnout, jak je asi stavba velká.

TIP: Je-li nastavena rychlost na nulu, avatar se nepohne z místa, může se však otáčet jako holub na báni. Toho lze využít ve speciálních případech, kdy avatara postupně navádíme na zajímavá, avšak pouze stacionární stanoviště.

Posledním parametrem navigačního uzlu je seznam povolených metod řízení avatara z velitelského stanoviště – parametr **type**. Existují tři základní způsoby pohybu avatara doplněné dvěma dalšími možnostmi:

"WALK"	Běžná chůze, při které se avatar pohybuje po zemi či podložce. Je-li povrch pod ním zvlněný, avatar odpovídajícím způsobem klesá a stoupá. Působí na něj přitažlivost ve směru záporné poloosy *y*, tedy dolů. Je zapnuta detekce kolizí s objekty tak, aby se při pokusu o průchod objektem avatar zastavil.
"FLY"	Stejné chování jako při chůzi s tím, že je vyřazeno působení přitažlivosti.
"EXAMINE"	Způsob, při kterém je avatar nejméně omezován běžnými fyzikálními zákony. Je určen především ke zkoumání objektů (angl. *examine*). Proto je vypnuta přitažlivost a avatar může kolem objektu kroužit ze všech stran, shora i zdola. Rychlost pohybu nastavená parametrem **speed** je ignorována. Zpravidla je takto zkoumán objekt, který se nachází přímo před avatarem, uprostřed obrazovky. Je vypnuto testování kolizí, takže objekty jsou průchozí skrz naskrz.
"ANY"	Nejedná se o způsob ovládání, ale o doporučení prohlížeči, že může uživateli povolit přepínání mezi předchozími základními způsoby. Na implicitním nastavení parametru **type** vidíme, že při vstupu do virtuálního prostředí je uživatelův avatar připraven k chůzi, avšak lze jej přepnout do libovolného z dalších dvou režimů. Není-li hodnota **ANY** zadána, lze volit jen mezi těmi způsoby, které jsou v parametru **type** vyjmenovány.
"NONE"	Zatímco předchozí hodnoty bylo možno kombinovat, hodnota **NONE** se v parametru **type** objevuje samostatně a určuje, že prohlížeč má skrýt veškeré ovládací prvky. Současně přestane převádět pohyb myši na pohyb avatara. Tento způsob je vhodný pouze tehdy, pokud jsou ve virtuálním prostředí aktivní tělesa, schopná po doteku teleportovat avatara na další stanoviště.

Poznámka: Působení přitažlivosti a schopnost detekovat nárazy do objektů se týká pouze avatara. Podobné chování nelze očekávat u virtuálních objektů, protože toto vyhodnocení by bylo extrémně výpočetně náročné. Při nevhodném rozmístění objektů se tak můžeme setkat s podivně se protínajícími tělesy nebo s objekty, které se záhadně vznášejí ve vzduchu.

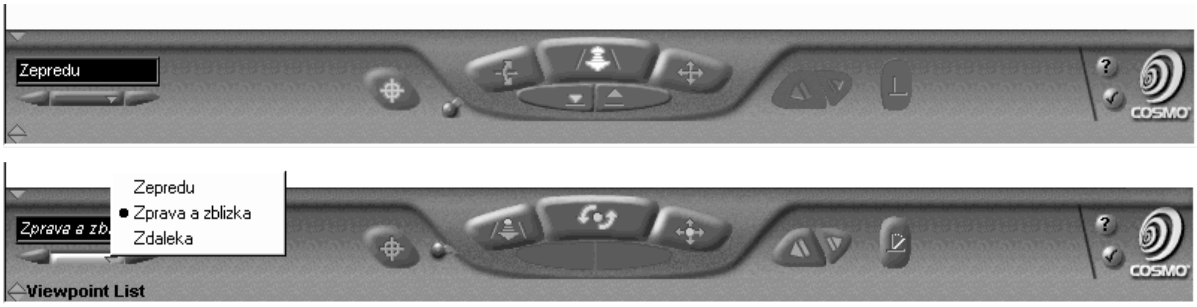
Krom výše uvedených způsobů prohlížení virtuálního světa obsahují mnohé prohlížeče i další prvky usnadňující pohyb a orientaci návštěvníka. V dalším textu popíšeme vzhled a způsoby ovládání prohlížeče CosmoPlayer pro MS Windows. Na obrázku 2-3 si všimneme postupně zleva těchto ovládacích prvků:

Nabídka stanovišť

Pod jménem aktuálního stanoviště jsou tři tlačítka pro přechod na jiná stanoviště. Krajními tlačítky přejdeme na předchozí a následující stanoviště, střední tlačítko nabídne k výběru ze seznamu všech stanovišť (dolní obrázek). Jakmile se z určitého stanoviště vydáme na samostatný průzkum a změníme tím tedy parametry pohledu, jméno stanoviště se zobrazí kurzívou. Při klepnutí na jméno nás prohlížeč vrátí do pozice určené tímto stanovištěm a současně zobrazí jeho jméno normálním písmem. Neměnná kurzíva se ve jménu stanoviště použije tehdy, pokud má stanoviště nastaveno směr pohledu jiný, než vodorovný – avatar je tedy předkloněn či zakloněn.

Zaměřovací kříž

Po stisknutí tlačítka se symbolem zaměřovacího kříže očekává prohlížeč, že v následujícím kroku vybereme libovolný objekt virtuálního světa. Prohlížeč nás poté plynule přenese do jeho blízkosti.



Obrázek 2–3: Dvě varianty ovládacích prvků prohlížeče CosmoPlayer pro MS Windows. Na horním obrázku režim chůze a letu, dole režim zkoumání objektů.

Ovládání pohybu

Poměrně složitá skupina ovládacích prvků uprostřed se přímo vztahuje k parametru **`type`** uzlu **`NavigationInfo`**. Tři větší tlačítka dovolují v režimu chůze a letu (WALK, FLY):

- otáčet se do stran, předklánět se a zaklánět se
- postupovat dopředu a zpět (zoom)
- dělat úkroky do stran (tzv. panorámování), stoupat a klesat.

K přepnutí mezi chůzí a letem slouží dvě malá dolní tlačítka se symboly gravitace (šipka směřující k podložce) a vzlétnutí. Navíc dochází k automatickému přechodu do režimu létání (je-li povolen), pokud při panorámování zahájíme stoupání vzhůru.

Změníme-li páčkou nalevo od středového bloku ovladačů režim procházení světa na režim zkoumání (EXAMINE), změní se současně trojice hlavních tlačítek (dolní obrázek). Zůstane tlačítko pro zvětšení a zmenšení zkoumaného objektu (zoom) a tlačítko pro posouvání objektu do stran (panorámování). Navíc se objeví tlačítko pro otáčení objektu. Zkoumaný objekt je v takovém případě obklopen pomyslnou neviditelnou koulí a pohyb kurzoru po obrazovce je transformován tak, jako kdybychom povrch této koule v libovolném místě uchopili a otáčeli s ním kolem středu koule. Souhlasně s touto akcí se natáčí i sledovaný objekt uvnitř.

TIP: Zkoumání objektu neznamená, že je objekt „vytažen“ z virtuálního světa ven. Ve skutečnosti jsou všechny akce s objektem převáděny na změny polohy avatara. Zvětšení objektu je pouhým přiblížením avatara, otáčení objektu je důsledkem toho, že jej avatar obchází. Nenechejme se tedy zmást tím, že při zkoumání jednoho objektu se hýbe celý okolní svět.

Návrat k předchozím pozicím

Prohlížeč zaznamenává historii našeho pohybu ve virtuálním světě a dovoluje nám pomocí další dvojice tlačítek vpravo od středového bloku postupně se vracet k předchozím pozicím avatara. To odpovídá použití známého principu operací UNDO/REDO, jak je známe z jiných počítačových programů. Seznam pozic avatara lze v tomto případě procházet dopředu i dozadu, změna je skoková. Na rozdíl od plynulých přeletů mezi stanovišti je tedy teleportace pomocí UNDO a REDO okamžitá.

Narovnání zad

Když ukončíme zkoumání nějakého objektu nebo let nad krajinou, je žádoucí narovnat avatara tak, aby stál kolmo vůči zemi, tedy aby jeho pohled směřoval vodorovně. K automatickému narovnání avatara dojde po stisku dalšího z ovládacích tlačítek se symbolem úhlu měřeného od kolmice (na dolním obrázku je prosvíceno).

Rady a další nastavení

Prohlížeč je podle očekávání schopen poradit s ovládáním (help) a umožňuje detailní nastavení různých parametrů – zhasnutí čelní svítilny, změna rychlosti pohybu, vypnutí detekce kolizí s překážkami apod. K tomu slouží poslední skupina ovládacích prvků, umístěná nejvíce vpravo.

Řídicí panely jiných prohlížečů mohou být uspořádány různými způsoby, případně doplněny o další možnosti a ikonky ovládacích prvků. V barevné příloze nalezneme vzhled nejznámějších prohlížečů.

Na závěr této kapitoly, která pojednávala o navigaci a globálních vlastnostech virtuálního prostředí, uvedeme nepříliš významný informační uzel **`WorldInfo`**. Ten se umísťuje do souboru vždy na začátek a slouží pro dokumentační účely. Do jeho parametru **`title`** se zapisuje jeden textový řetězec, který může, ale nemusí být prohlížečem zobrazován jako jméno daného světa. Parametr **`info`** může obsahovat libovolný počet textových řetězců, jejich význam není blíže určen.

```
#VRML V2.0 utf8
WorldInfo {
  title "Prazdny svet"
  info  ["Autor: Jiri Zara",
         "Datum: 1.4.1998"
  ]
}
```

Program P-2-2: Příklad použití uzlu **`WorldInfo`**

TIP:

1. **Textové řetězce se zapisují do uvozovek.**
2. **U každého parametru je stanoveno, zda do něj lze zapsat právě jednu nebo více hodnot. Je-li hodnot více, vkládají se do hranatých závorek a oddělují čárkami.**

3 Jednoduché světy
(aneb CO, KAM a JAK)

V této části se přesvědčíme, že není třeba mít obavy z modelování umělých světů. Při troše fantazie můžeme vytváření umělých těles a světů přirovnat k sestavování dílků stavebnice LEGO. Každý tvůrce prostorových objektů totiž postupuje přibližně stejným způsobem a i poměrně složité výrobky sestavuje z jednodušších částí. Není proto těžké jej napodobit. Následující přehled základních operací je zároveň stručným návodem, který bychom mohli nalézt v pomyslné (virtuální?) stavebnici *„Malý stavitel VRML“*.

Od malých krůčků k velkému dílu!	***Co k tomu potřebuji?***
1. Vymodeluj hrubý tvar objektu (tělesa) a stanov jeho velikost.	Tělesa a transformace
2. Vylepši povrch tělesa.	Barva, textura
3. Nejde-li použít základní těleso, udělej si vlastní.	Plochy, čáry, body
4. Přidej efekty pro zvýšení prostorového dojmu.	Světla, zvuky
5. Rozmísti tělesa do prostoru.	Skupinové uzly

Dříve, než se podíváme na jednotlivé kroky podrobněji, řekneme si informaci, která je pro stavbu virtuálních světů velmi důležitá:

V roli parametrů uzlů se mohou objevovat jiné uzly.

V tomto jednoduchém oznámení se skrývá mocný nástroj pro tzv. *hierarchickou stavbu*. Uzly totiž nemusí být kladeny do virtuálního prostředí pouze sekvenčně jeden za druhým jako v příkladu P-2-1, ale lze je sdružovat tak, aby vznikaly několikapatrové struktury, ne nepodobné stromům popisujícím například rodokmen. Však se také tyto hierarchické struktury skutečně nazývají *stromy* (*tree*) a podle vzájemné polohy uzlů ve stromu mluvíme o vztazích: *rodič*, *potomek*, *sourozenec*.

A podobně jako v rodině, rodič předává potomkům (tj. uzlům zapsaným jako příslušné parametry) určité vlastnosti. Ve virtuálním světě to může být poloha, schopnost stejné reakce na vnější podněty apod. Stromová struktura je používána velmi často a proto ji budeme v některých případech symbolicky kreslit současně s dále uváděnými příklady VRML souborů.

3.1 Základní tělesa a transformace
(Box, Cone, Cylinder, Sphere + Transform, Shape)

Pro stavitele jsou připravena pouze čtyři základní geometrická tělesa – koule, kvádr, kužel a válec. Všechna jsou iniciálně umístěna tak, aby jejich těžiště bylo v počátku souřadné soustavy. Výjimkou je kužel, který je situován tak, že v počátku souřadnic leží střed jeho osy.

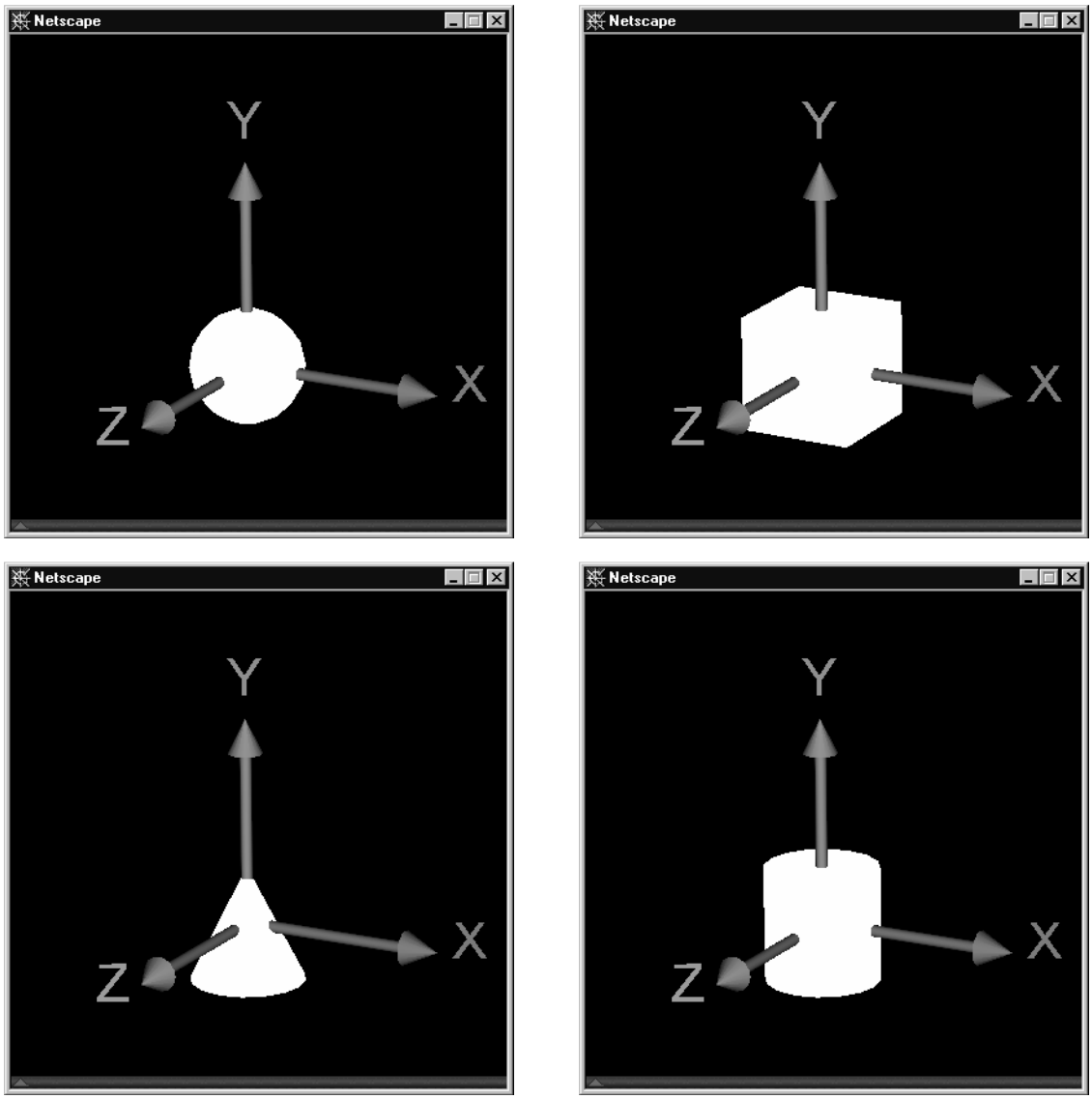
těleso	uzel	parametr	iniciální hodnota	význam
koule	`Sphere {` `}`	`radius`	`1`	poloměr
kvádr	`Box {` `}`	`size`	`2 2 2`	délky stran *x*, *y* a *z*

kužel	`Cone {`
	
	`}`
		`bottomRadius`
		`height`
			`1`
			`2`
				poloměr podstavy
				výška

válec	`Cylinder {`
	
	`}`
		`radius`
		`height`
			`1`
			`2`
				poloměr
				výška

Tabulka T-3-1: Rozměrové parametry základních těles

Tabulka T-3-1 určuje iniciální rozměry základních těles, jejich orientaci ukazuje obrázek 3-1. Všechna tělesa jsou v paměti počítače převedena do množiny povrchových plošek. Jejich přesný počet známe pouze u kvádru – má jich právě šest. U ostatních těles nelze stanovit počet a tedy přesnost, s jakou je oblý povrch tělesa nahrazen soustavou (trojúhelníkových) plošek. Tato vlastnost však není žádnou chybou, ba právě naopak – prohlížeč může dynamicky měnit kvalitu ploškové náhrady v závislosti na výkonu počítače. Menší počet ploch snižuje výpočetní nároky a uživatel se rád spokojí s „hranatou“ koulí za cenu vyšší rychlosti při práci ve virtuálním světě.



Obrázek 3-1: Umístění a orientace základních těles

Kužel a válec mají povrch tvořen rotačním pláštěm a kruhovými podstavami. Jsou-li tato tělesa postavena na podložce tak, že například jejich dolní podstava nemůže být nikdy vidět, je vhodné podstavu rovnou vypustit. Vhodným nastavením parametrů `side`, `bottom` (u válce ještě `top`) na logickou hodnotu **FALSE** vynecháme odpovídající části pláště, čímž snížíme počet zpracovávaných ploch a urychlíme zobrazování.

TIP: Odstraněním podstav válce nelze vyrobit trubici. Její odvrácená stěna je prohlížečem vždy ignorována kvůli úspoře času a výsledný útvar pak připomíná korýtko. Řešení přináší teprve uzel `Extrusion` (viz kapitola 3.4).

Abychom mohli do virtuálního světa vložit těleso a začít si je prohlížet, musíme se naučit pracovat se dvěma uzly, jejichž úkolem je vytvářet stromové struktury a shrnovat jednodušší objekty a jejich vlastnosti do jednoho celku:

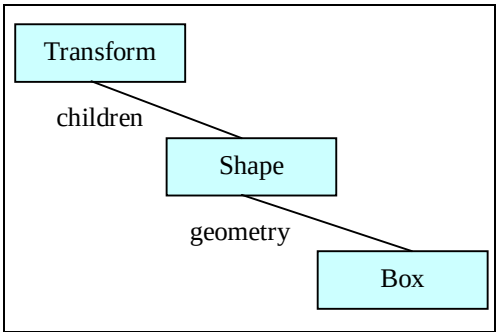
funkce uzlu	uzel	parametr	iniciální hodnota	význam
umístění	`Transform {`	`scale`	`1 1 1`	měřítko
		`rotation`	`0 0 1 0`	osa a úhel natočení
		`translation`	`0 0 0`	posunutí
		`children`	`prázdný uzel`	seznam potomků
	`}`			
svázání vlastností	`Shape {`	`geometry`	`prázdný uzel`	tvar povrchu
		`appearance`	`prázdný uzel`	vzhled povrchu
	`}`			

Tabulka T-3-2: Základní uzly stromové hierarchie

Typicky je **`Transform`** rodičovským uzlem, který zajišťuje pro všechny své potomky (zapsané do parametru **`children`**) totožné nastavení měřítka (**`scale`**), natočení (**`rotation`**) a posunutí (**`translation`**).

Následující příklad je typickou ukázkou základního tvaru stromu, definujícího polohu tělesa. Rodičovský uzel **`Transform`** má jediného potomka – uzel **`Shape`**. Ten má opět jednoho potomka, který určuje geometrii tělesa, v tomto případě kvádru. Parametr **`appearance`** je zatím nevyužit a proto je vzhled kvádru nepříliš zajímavý – těleso je matné a bílé, což odpovídá iniciálnímu nastavení barev objektů. Rodičovský uzel má nastaven parametr měřítka. Výsledný kvádr proto bude mít délky stran 6, 2 a 4 metry, neboť v iniciálním tvaru je reprezentován kostkou o délce hrany 2.

```
#VRML V2.0 utf8
Transform {
  scale 3 1 2
  children Shape { geometry Box {}
           }
}
```



Program P-3-1: Jednoduché umístění tělesa (vlevo) a odpovídající stromová struktura (vpravo)

Téhož tvaru kvádru ovšem docílíme i přímým zadáním délek stran do uzlu **`Box`**, jak ukazuje následující zápis:

```
#VRML V2.0 utf8
Transform {
  children Shape { geometry Box {size 6 2 4}
           }
}
```

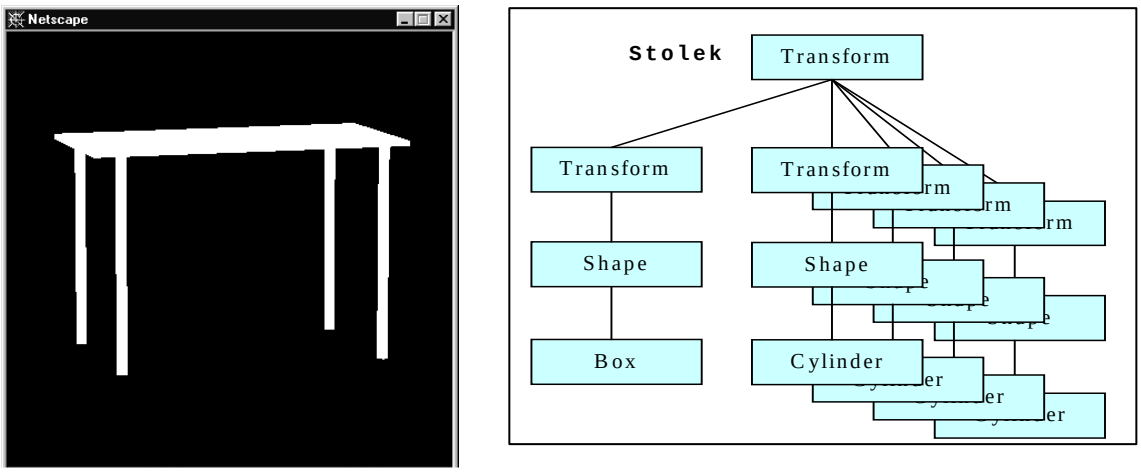
Program P-3-2: Lepší způsob zadání téhož tělesa

Tento druhý způsob je vhodnější, protože umožňuje pozdější dynamické změny měřítka a polohy při zachování neměnného poměru stran kvádru. Jak uvidíme v kapitole 5 (Události hýbou světem), všechny výše uvedené parametry uzlu **`Transform`** lze animovat, zatímco pevný parametr **`size`** kvádru nelze.

Rodičovský, neboli též *skupinový* uzel **`Transform`** dokáže sdružit několik těles dohromady, jak je vidět na obrázku 3-2. Výsledný stolek je tvořen pěti částmi. Všimněme si měrných jednotek, ve kterých je stůl definován. Rozměry jeho jednotlivých dílů naznačují zápis v centimetrech, vzhledem k oficiální jednotce *metr* je nutno celek upravit vhodnou změnou měřítka v rodičovském uzlu (viz zápis **`scale 0.01 0.01 0.01`**).

TIP: Ačkoliv lze uzel `Shape` zapsat samostatně, je vždy lepší jej zařadit jako potomka uzlu `Transform`, aby bylo možno definovat jeho polohu.

Pozn.: Pojmenování výsledného tělesa názvem **`Stolek`** v souboru nemá v tomto případě žádný význam.



Obrázek 3-2: Více těles sdružených do jednoho stromu

```
#VRML V2.0 utf8
DEF Stolek Transform {
  scale 0.01 0.01 0.01
  rotation 0 1 0 0.3
  children [
    Transform { # deska stolu
      translation 0 80 0
      children Shape { geometry Box {size 120 2 60}
               }
    }
    Transform { # noha
      translation -50 40 –25
      children Shape { geometry Cylinder {radius 2 height 80 top FALSE}
               }
    }
    Transform {
      translation -50 40 25
      children Shape { geometry Cylinder {radius 2 height 80 top FALSE}
               }
    }
    Transform {
      translation 50 40 –25
      children Shape { geometry Cylinder {radius 2 height 80 top FALSE}
               }
    }
    Transform {
      translation 50 40 25
      children Shape { geometry Cylinder {radius 2 height 80 top FALSE}
               }
    }
  ]
}
```

Program P-3-3: Uzel **`Transform`** *jako rodič několika potomků*

Transformace jsou v rámci jednoho uzlu prováděny vždy v následujícím pořadí:

1. úprava měřítka (**`scale`**)
2. natočení (**`rotation`**)
3. posunutí (**`translation`**)

Připomeňme si, že na pořadí velmi záleží. Kdybychom nejprve těleso posunuli a v jeho nové poloze otočili kolem středu souřadné soustavy, s největší pravděpodobností by se ocitlo na zcela jiném místě, než kdybychom je nejprve otočili a poté posunuli.

TIP:

1. **Chceme-li změnit pevně dané pořadí transformací (například nejprve těleso posunout a pak teprve natočit vůči počátku souřadnic), nezbývá než umístit více uzlů `Transform` do stromu nad sebe. Posunutí se provede v potomku, natočení v rodiči. Transformace se skládají od dětí k rodičům.**

2. **Uzel `Transform` obsahuje další „důmyslné“ parametry, které dovolují měnit měřítko tělesa (protahovat je) v libovolném směru nebo je otáčet kolem libovolného bodu. Detailní popis všech parametrů je uveden v kapitole 9.**

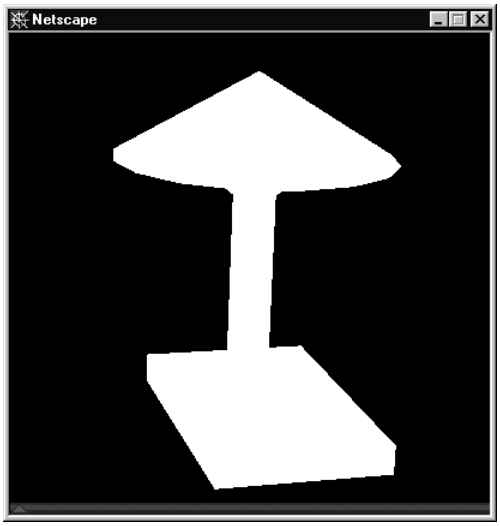
Natočení a posunutí jsou nedeformující operace, pouze nastavují polohu a orientaci tělesa. Pomocí změny měřítka pak dokážeme tělesa nejen zvětšovat a zmenšovat, ale i deformovat. Z koule tak může snadno vzniknout prostorový elipsoid. Platí však, že koeficienty měřítka musí být kladná čísla. Tím se možnosti uzlu `Transform` liší od obecných transformací, při nichž lze například docílit tzv. *zrcadlení* (symetrie) zadáním koeficientů měřítek o hodnotě minus jedna.

Krok za krokem I – Model lampičky

Představme si, že chceme vytvořit model virtuální lampičky. Bude stát na podstavci ve tvaru kvádru, její nožka bude válcová a stínítko bude mít tvar kuželu. S pomocí dosud popsaných těles dokážeme takovou lampičku snadno vymodelovat, ba dokonce do ní umístit i žárovku ve tvaru koule. Následující příklad definuje model na obrázku 3-3. Počátek souřadnic je umístěn do osy nožky lampičky, na dolní stěnu jejího podstavce.

```
#VRML V2.0 utf8
DEF Lampa-I Transform {
  children [
    Transform { # podstavec ve tvaru kvádru
      translation 0 0.01 0.04
      children Shape {
         geometry Box {size 0.12 0.02 0.2}
      }
    }
    Transform { # nožka ve tvaru válce
        translation 0 0.09 0
        children Shape {
           geometry Cylinder {
              bottom FALSE  top    FALSE
              radius 0.015  height 0.18
           }
        }
    }
    Transform { # žárovka ve tvaru koule
        translation 0 0.18 0
        children Shape {
           geometry Sphere {radius 0.03}
        }
    }
    Transform { # stínítko ve tvaru kuželu
        translation 0 0.21 0
        children Shape {
           geometry Cone {
              bottomRadius 0.1 height 0.06
           }
        }
    }
  ]
}
```

Program P-3-4: Geometrie jednoduché lampičky



Obrázek 3-3: Jednoduchá lampička

Nožka lampičky stojí na podstavci, na jejím vrcholu je pak baňka žárovky. Ani jedna z podstav válce představujícího nožku nemůže být nikdy vidět. Proto jsou parametry **`bottom`** a **`top`** nastaveny na **FALSE**. Ačkoliv tento zápis prodlouží soubor o jeden řádek, tj. přibližně o 40 bytů, ušetříme množství zobrazovaných dat a tedy cenný čas výpočtu při prohlížení.

Jak vidíme na obrázku 3-3, geometrie lampičky je poměrně uspokojivá, přesto je výsledný estetický dojem spíše zoufalý. To, co právě nyní potřebujeme, jsou barvy, které dodají povrchu jednotlivých částí pořádný vzhled.

3.2 Za povrch krásnější... (Appearance, ImageTexture, Material, MovieTexture, PixelTexture, TextureTransform)

V předchozí kapitole jsme zjistili, že uzel **Shape** má kromě parametru **geometry** ještě druhý parametr – **appearance**. Ten jsme dosud nevyužili, ačkoliv právě on má zásadní vliv na vzhled tělesa, jak ostatně sám jeho název napovídá.

Do parametru **appearance** se umísťuje pouze jediný možný uzel, který se nazývá příhodně **Appearance**. Dovoluje definovat dva možné vzhledy povrchu těles:

1. barvu (parametr **material**),
2. texturu (parametry **texture** a **textureTransform**).

Parametr **material** je určen k zadání barvy, která bude stejná na celém povrchu. Parametr **texture** pak dovoluje pokrýt (polepit) povrch libovolným obrázkem, čímž lze tělesu dodat velmi přirozený vzhled tak, aby připomínalo výrobek ze dřeva, kamene apod. Do žádného z parametrů uzlu **Appearance** se nepřiřazují číselné hodnoty, ale opět uzly. Jejich jména jsou odvozena ze jmen parametrů:

- do parametru **material** lze přiřadit pouze uzel **Material**,
- do parametru **texture** lze přiřadit jeden z uzlů **ImageTexture**, **PixelTexture** nebo **MovieTexture**,
- do parametru **textureTransform** lze přiřadit pouze uzel **TextureTransform**.

Jméno uzlu **Material** je trochu zavádějící. Normálně bychom si pod ním mohli představit takové pojmy, jako jsou například železo, dřevo, papír apod. Ve skutečnosti tento uzel označuje pouze barevné charakteristiky povrchu. Nutno přiznat, že jich není málo a že pro jejich plné pochopení je třeba proniknout hlouběji do problematiky barev a světel v trojrozměrném prostoru. Barevný vjem je totiž přímo ovlivňován světlem. Následující tabulka poskytuje sice úplný, ale vědomě zjednodušený popis parametrů uzlu **Material**. K jejich hlubšímu studiu doporučujeme odbornou literaturu, např. [1,8].

parametr	iniciální hodnota	význam
diffuseColor	**0.8 0.8 0.8**	základní složení barvy povrchu ve složkách RGB
ambientIntensity	**0.2**	jak je barva povrchu zesvětlována celkovým jasem prostoru
specularColor	**0 0 0**	jakou barvu dopadajícího světla povrch odráží
shininess	**0.2**	ostrost odrazu pro předchozí parametr
emissiveColor	**0 0 0**	fluorescenční (svítivá) barva povrchu
transparency	**0**	průhlednost (je-li rovna jedné, těleso zmizí)

Tabulka T-3-3: Parametry uzlu **Material**

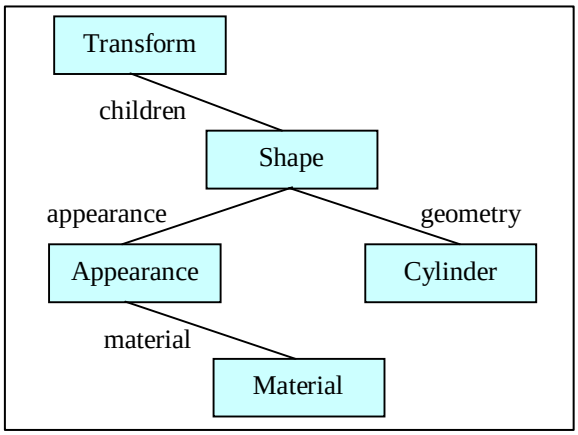
Tomu, jak do scény zavést různé zdroje světla, se budeme věnovat v samostatné kapitole 0. Zde jen poznamenejme, že všechny materiálové parametry mohou nabývat hodnot od nuly do jedné. Některé jsou tvořeny trojsložkovými barevnými vektory RGB, jiné pouze skalárními koeficienty. Všimněme si také, že přednastavené hodnoty parametrů určují právě takový vzhled těles, se kterým jsme se setkávali v dosud uváděných jednoduchých příkladech – matná, bílá tělesa, která neodrážejí světlo, sama nezáří a jsou neprůhledná.

TIP: Nenulové hodnoty parametru emissiveColor způsobí, že těleso začne zářit. Jeho jas však nemá žádný vliv na ostatní tělesa. Skutečné zdroje světla se definují samostatnými uzly (viz kapitola 0).

Pro většinu jednoduchých těles vystačíme s nastavením jediného parametru – **diffuseColor**. Ten definuje matný vzhled. Odstín barvy se mění podle úhlu dopadajícího světla. Přímo proti světlu je barva nejjasnější.

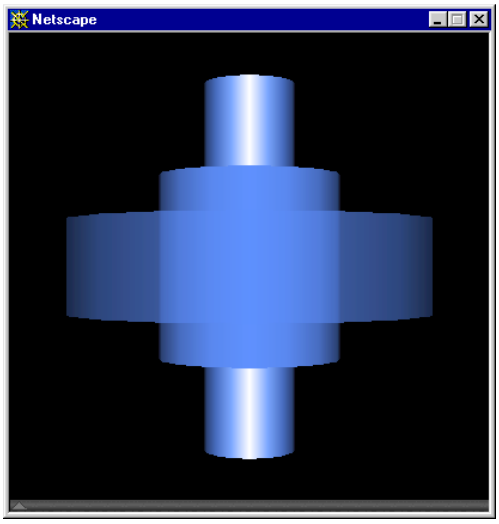
Typický zápis takto barevného tělesa může vypadat:

```
#VRML V2.0 utf8

Transform {
  children
    Shape {
        appearance Appearance {
          material Material {
            diffuseColor  0.5 0.5 1
          }
        }
        geometry Cylinder {}
    }
}
```



Program P-3-5: Matný modrý válec

TIP: Hierarchická (stromová) struktura je důsledně využívána nikoliv ke zmatení čtenářů a programátorů, ale k jasnému odlišení významu detailního popisu virtuálních těles. Zvlášť je řečeno, CO bude definováno (např. barva či geometrie) a zvlášť, JAK konkrétně daná veličina bude vypadat (např. složení barvy či rozměry válce).



Obrázek 3-4 ukazuje pro srovnání, jak vypadá těleso složené ze tří souosých válců. Prostřední z nich má materiál nastaven tak, jak je ukázáno výše v programu P-3-5. Tenký válec se od středního liší nastavením pouze jediného dalšího parametru. Je jím **`specularColor`**, jehož hodnota [**`1 1 1`**] indikuje, že dopadající světlo se bude projevovat bílými odlesky. Nejmenší a nejsilnější váleček má nastavenu průhlednost pomocí parametru **`transparency`** na 0.5, jinak je ze stejného materiálu jako jeho matný soused.

Obrázek 3-4: Odlišné materiálové parametry použité na tři tělesa s podobnou geometrií

TIP: Jen málokdy jsou nastaveny všechny materiálové parametry. Zejména má-li parametr fluorescence `emissiveColor` nenulovou hodnotu, je dobré všechny ostatní parametry vynulovat. Zjednoduší se tím mnoho výpočtů.

3.2.1 Textury aneb Za málo peněz hodně muziky

Ačkoliv použití barvy přináší výrazné zlepšení vzhledu těles, nejvěrnějšího vzhledu povrchu dosáhneme teprve využitím textur. Textura je obrazový vzorek, který je nanášen na povrch tělesa. Tomuto procesu se říká *mapování textury*. Je zřejmé, že mapování je doprovázeno rozličnými deformacemi, protože textura je definována jako dvojrozměrný (obdélníkový) obrázek, kdežto povrch těles je obecně zakřivený. Přesto se dá pomocí textur docílit velmi pěkných efektů – obyčejný válec se zdá být vyřezán ze dřeva, jednoduchý kvádr vypadá jako mramorová deska.

Pro výběr barevného vzoru lze použít několika možností, každá z nich je svázána se samostatným uzlem:

1. Obrázek uložený v samostatném souboru (uzel **`ImageTexture`**)
2. Opakující se kombinace barev zapsaná v uzlu (**`PixelTexture`**)
3. Přehrávaná video sekvence uložená v samostatném souboru (**`MovieTexture`**)

Výše uvedené tři druhy uzlů popisují zdroj dat, ze kterého budou získávána obrazová data použitá jako textura. Další uzel je pak určen k tomu, aby definoval způsob mapování textury na povrch. Tímto uzlem je **`TextureTransform`**, který popisuje posunutí, natočení a změnu měřítka textury. Uzel má proto následující parametry:

parametr	iniciální hodnota	význam
`translation`	**`0 0`**	posunutí textury
`center`	**`0 0`**	poloha vztažného bodu
`rotation`	**`0`**	úhel otočení kolem vztažného bodu
`scale`	**`1 1`**	změna velikosti vzhledem ke vztažnému bodu

Tabulka T-3-4: Parametry uzlu **`TextureTransform`**

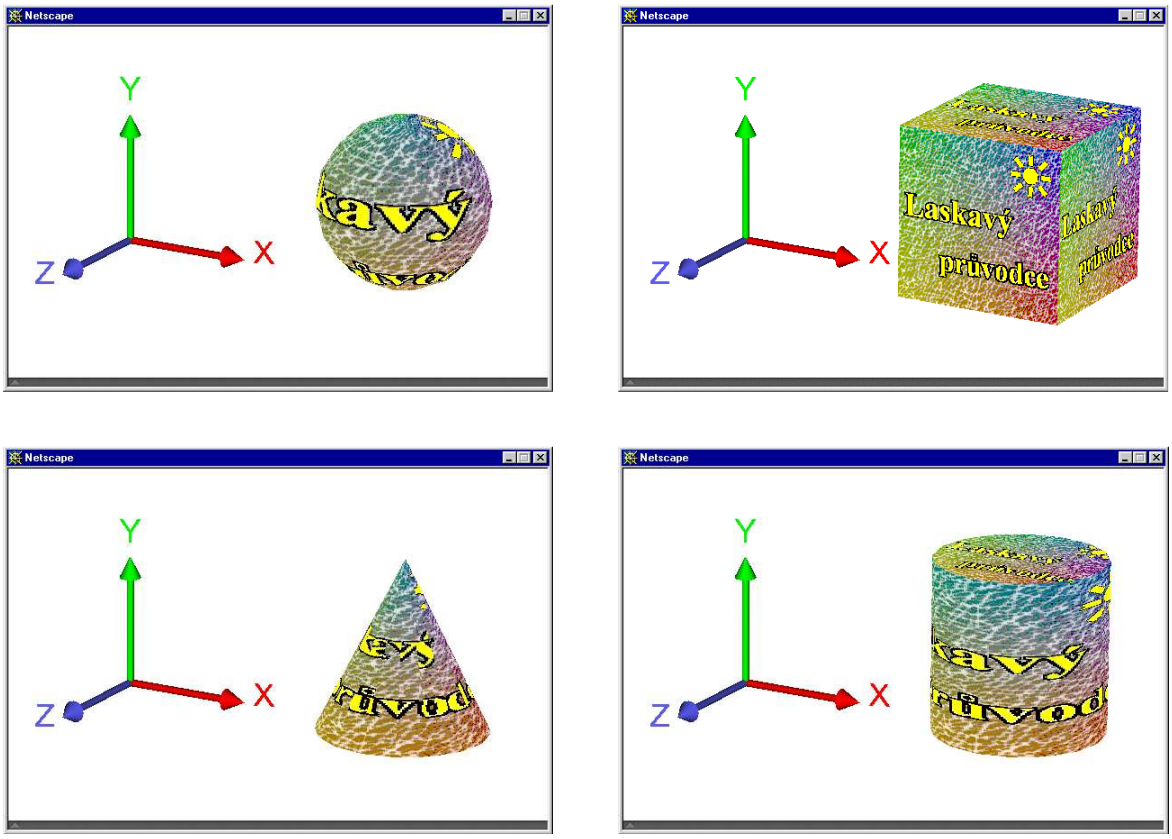
Hodnoty parametrů jsou poněkud nezvykle vztaženy vůči definičnímu rozsahu textury, nikoliv vůči tělesu, na který je textura mapována. V praxi to například znamená, že nastavení měřítka **`scale`** na hodnotu **`2 2`** nezvětší texturu ve směru *x* a *y* dvakrát, nýbrž ji zmenší a namapuje na těleso celkem ve čtyřech kopiích vedle sebe. Je to způsobeno tím, že při výpočtu se nepostupuje od textury k tělesu, ale obráceně. Ve výše uvedeném případě jsou nejprve nalezeny souřadnice bodu na povrchu tělesa, vynásobeny příslušnými měřítky (tj. zvětšeny) a poté hledány ve zdrojovém obrazu textury. Když přesáhnou rozměry textury, jejich hodnota je opakovaně snižována odečítáním velikosti textury (tzv. operací *modulo*) až do té doby, než je nalezena barva uvnitř textury.

```
#VRML V2.0 utf8

Transform {
  children
    Shape {
      geometry Sphere {}
      appearance Appearance {
        texture ImageTexture
          {url "obr-rgb.gif"}
      }
    }
}
```

Program P-3-6, který mapuje na kouli obrazovou texturu ze souboru ***`obr-rgb.gif`*** *(vpravo)*

Aniž bychom museli hlouběji proniknout k matematické podstatě problému, zapamatujeme si, že veškeré akce s texturou jsou vlastně prováděny obráceným postupem. Posun textury doprava zajistíme nastavením prvního parametru **`translation`** na zápornou hodnotu (tedy doleva), otočení proti směru hodinových ručiček (tzv. kladný směr otáčení) způsobí záporná hodnota parametru **`rotation`** a zvětšení textury docílíme zadáním měřítek menších než jedna.

Všechna čtyři základní tělesa mají jednoznačně určen způsob mapování textury, jak ukazují obrázky 3-5, kde jsou texturou pokryta tělesa v základní velikosti tak, jak je tomu v programu P-3-6. Obrazová textura definovaná uzlem **`ImageTexture`** se zapisuje do parametru **`texture`** uzlu **`Appearance`**.



Obrázek 3-5: Pokrytí základních těles texturou

Způsob nanesení textury je svázán s cílovým tělesem. Na kouli je textura nanesena pouze jednou tak, že ji „omotá“ kolem rovníku, na pólech dojde k výraznému zkreslení obrazu – jeho okraje se zhroutí do jediného bodu. Naopak na kvádr je textura mapována bez zkreslení, celkem v šesti kopiích – jedna na každou stěnu. Podobně je bez zkreslení nanášena textura na válec. Není-li pomocí uzlu **`TextureTransform`** zadáno jinak, je obraz právě jednou namapován na celý plášť a dále je nanesen na obě podstavy tak, že podstavy tvoří kruh vepsaný čtverci textury. Podobně je tomu i u kuželu, u jeho vrcholu však dochází ke zkreslení.

Jak je zřetelné na obrázcích 3-5, u rotačních těles je levý okraj textury nanášen od okraje se zápornou souřadnicí *x*.

TIP: Každý obraz, která slouží jako textura, je vnitřně převeden na čtverec o straně 1 a poté mapován. V důsledku toho nedokážeme nanést texturu bez deformace na kvádr se stěnami o různých délkách. Měřítko upravené pro jednu stěnu selže na jiné stěně. Řešení přináší teprve uzel `IndexedFaceSet` se samostatným mapováním textury pro každou stěnu (viz následující kapitola 3.4)

Každý z uzlů definujících texturu má své specifické parametry. Společné všem jsou právě dva:

parametr	iniciální hodnota	význam
`repeatS`	**`TRUE`**	povolení opakování textury ve vodorovném směru
`repeatT`	**`TRUE`**	povolení opakování textury ve svislém směru

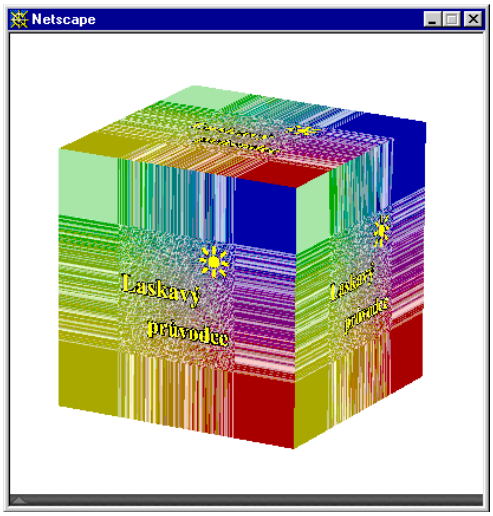
Tabulka T-3-5: Společné parametry všech tří uzlů definujících texturu

Nechceme-li, aby se textura opakovaně nanášela na celý povrch, nastavíme parametry opakování na hodnotu **FALSE**. Otázka, jak obarvit povrch tělesa mimo texturu, je řešena jednoduše – je zopakována barva okraje textury. Prostor vpravo od textury je tedy vyplněn těmi barvami, které se nacházejí na pravém okraji textury, podobně je tomu v ostatních směrech. Tím vzniká zajímavý, i když ne vždy žádoucí efekt vytvoření pruhů po

celém povrchu kombinovaných s jednobarevnými čtverci, jejichž barva je shodná s barvou rohového bodu textury (viz obrázek u příkladu P-3-7). Tohoto jevu lze využít v pozitivním smyslu, když obrázek textury předem upravíme v nějakém editoru tak, aby na jeho okraji vznikl tenký jednobarevný rámeček. Barva rámečku pak vyplní veškeré okolí textury na povrchu tělesa.

Ukázka použití textury bez opakování je uvedena[3] v příkladu P-3-7. Na něm si také všimneme již dříve zmiňovaného „obráceného“ způsobu zadávání parametrů transformace textury v uzlu **`TextureTransform`**. Obrázek chceme zmenšit na polovinu a proto zadáme dvojnásobná měřítka textury (**`scale 2 2`**). V tomto okamžiku by se textura umístila do levého dolního rohu každé stěny. K jejímu vystředění je potřeba ještě posunutí doprava nahoru, paradoxně pomocí hodnot se záporným znaménkem (**`-0.25`**). Velikost parametru posunutí (**`translation`**) je vztažena k základnímu rozměru textury, tj. čtverci o délce jedna.

```
#VRML V2.0 utf8
Transform {
  children
    Shape {
      geometry Box {}
      appearance Appearance {
        texture ImageTexture {
                url "obr-rgb.gif"
                repeatS FALSE
                repeatT FALSE
        }
        textureTransform
          TextureTransform {
                scale 2 2
                translation -0.25 -0.25
          }
      }
    }
}
```



Program P-3-7: Použití transformované textury bez opakování

TIP: **Porovnáme-li stejnojmenné parametry `scale` nebo `translation` u uzlů `Transform` a `TextureTransform`, zjistíme, že mají odlišný počet hodnot. V rovině se používají dvě souřadnice, v prostoru tři. Nelze se tedy spoléhat pouze na jméno parametru, ale je třeba brát do úvahy jeho konkrétní použití.**

3.2.2 Podivný parametr url

Tento parametr je zkratkou, používanou v síti Internet a označující adresu souboru či místa, přístupného po síti (URL – ***U**niform **R**esource **L**ocator*). Je prvním dokladem toho, že VRML dovoluje využívat pro tvorbu virtuálních světů prostředky dostupné po síti. Uzly **`ImageTexture`** a **`MovieTexture`** mají v tomto parametru zapsány údaje o umístění a jménu obrázku, resp. animace.

Sympatickou vlastností parametru **`url`** je možnost zápisu několika textových řetězců, které budou využity pro vyhledávání konkrétního souboru v případě, že předchozí adresy nejsou dostupné. Tento přístup ocení zejména zkušení uživatelé Internetu a WWW, kterým se nejednou stalo, že je hypertextový odkaz zavedl na WWW stránku, která byla v dané chvíli nedostupná, ať již kvůli přetížení sítě, změně adresy cílového počítače nebo jeho vypnutí.

Vytváříme-li například virtuální obrazovou galerii, nemusíme mít na svém počítači umístěny všechny obrazy, protože se na ně můžeme odkázat na jiné místo v Internetu. To mimochodem sníží zátěž našeho počítače, protože značné množství dat bude k uživatelům proudit z jiných serverů. Samozřejmě je vhodné mít připraveny

[3] Bílé pozadí obrázku je vytvořeno pomocí uzlu **`Background`**. O něm bude pojednáno v kapitole 3.5.

náhradní obrázky (v horší barevné kvalitě a nižším rozlišení) na našem počítači pro případ selhání sítě. Obsah parametru **`url`** v uzlu **`ImageTexture`** pak může vypadat takto:

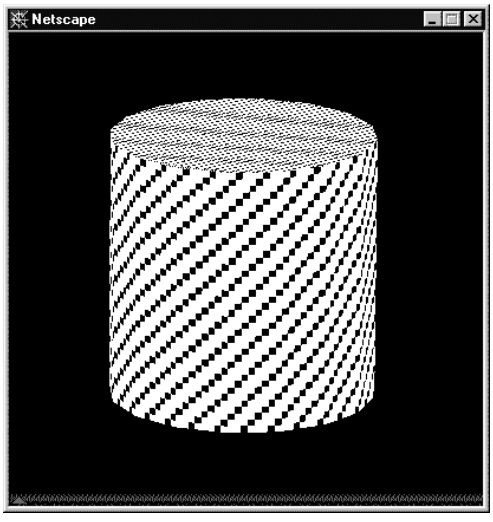
```
ImageTexture {
  url ["http://www.louvre.fr/da-vinci/mona-lisa.jpg",
       "http://www.archive.fr/images/mona-lisa.png",
       "moje-obrazky/nahradni.gif"
  ]
}
```

Uvedený příklad zároveň ukazuje, jaké typy obrázků mohou být použity pro textury. Formát ***JPEG*** (*jpg*) je vhodný pro kódování plnobarevných fotografií, vyznačuje se však tím, že zanedbává některé detaily a barevné přechody. Pro virtuální realitu je ovšem tento princip ztrátové komprese zcela přijatelný. Dalším formátem je ***GIF***, který uchovává obrazy o maximálně 256 barvách. Méně známým formátem je ***PNG***. Patří mezi nejmladší obrazové formáty a očekává se, že postupně nahradí GIF. Dokáže totiž efektivně kódovat plnobarevné obrazy včetně informací o průhlednosti, a to bezeztrátově.

Jako pohyblivou texturu (**`MovieTexture`**) lze použít soubor typu ***MPEG-1***. Některé prohlížeče akceptují i tzv. *animovaný GIF*, tedy jeden soubor GIF obsahující posloupnost několika obrázků. Na rozdíl od statického obrázku má uzel **`MovieTexture`** další parametry, týkající se dynamiky prohlížení, či spíše přehrávání sekvence pohyblivých obrazů. Logický parametr **`loop`** povoluje spouštění přehrávání v nekonečné smyčce, které je běžně vypnuté. Parametrem **`speed`** se pak nastavuje rychlost přehrávání. Je-li větší než 1, video sekvence se přehrává rychleji než původní rychlostí, hodnota menší než 1 znamená zpomalení. Zajímavým trikem je nastavení rychlosti na zápornou hodnotu, která způsobí přehrávání pozpátku. Nulová hodnota způsobí zobrazení jen prvního snímku z video sekvence. V případě použití animovaného GIFu však není korektní časování zaručeno.

Jako chudá příbuzná vypadá textura zvaná **`PixelTexture`**, přesto je velmi šikovná pro úsporné definování jednoduchých, opakujících se vzorků na povrchu těles. Neodkazuje se na žádný vnější zdroj obrazových dat, ale definuje vzorek přímým vypsáním barev do parametru **`image`**. Ukázka jejího použití je v programu P-3-8.

```
#VRML V2.0 utf8
Transform {
  children
    Shape {
      geometry Cylinder {}
      appearance Appearance {
        texture PixelTexture {
          image  4 4 1
                 0x00 0xFF 0xFF 0xFF
                 0xFF 0x00 0xFF 0xFF
                 0xFF 0xFF 0x00 0xFF
                 0xFF 0xFF 0xFF 0x00
        }
        textureTransform
          TextureTransform {scale 31 10}
      }
    }
}
```



Program P-3-8: Textura v uzlu **`PixelTexture`** *použitá ke šrafování válce*

Obsah parametru **`image`** má ze všech parametrů VRML nejsložitější vnitřní strukturu. První dvě čísla znamenají rozměry, tj. počet dále definovaných barev pixelů v osách *x* a *y*. Třetí číslo určuje způsob zápisu jednotlivých barev. Může nabývat následujících hodnot:

0 – žádná barva (tato hodnota je rezervována jen pro iniciální nastavení parametru `image: 0 0 0`)

1 – odstíny šedi v rozsahu 0 (černá) až 255 (bílá)

2 – odstíny šedi doplněné informací o průhlednosti (oboje v rozsahu 0-255)

3 – barva složená ze tří složek RGB (červené, modrá, zelená), každá v rozsahu 0-255

4 – barva tvořená třemi složkami RGB a doplněná informací o průhlednosti

V příkladu P-3-8 je textura definována čtvercovou oblastí o rozměrech 4 x 4 pixely, barvy jsou zadávány jako odstíny šedi. Jednotlivé barvy jsou potom zapisovány po řádcích. Začátečníka může zmást skutečnost, že řádky jsou chápány ve smyslu zdola nahoru. První řádek barev v souboru je proto nejnižším řádkem textury, což vyvolává optický dojem, že obraz textury je převrácený vůči definici v souboru.

Samotný zápis barev jednotlivých pixelů je určen spíše pro zkušenější programátory. Je preferováno vyjádření v hexadecimální (šestnáctkové) soustavě, kde **`0x00`** je rovno nule, **`0xFF`** znamená hodnotu 255. Podobně například zápis **`0xFF40FF80`** označuje pixel světle fialové barvy (RGB=[255, 64, 255]), který je z 50% průhledný. Ačkoliv pro jednoduchý zápis odstínů šedi můžeme použít přímo čísel od jedné do dvou set padesáti pěti, v praxi se používá hlavně hexadecimálního tvaru.

Na příkladu P-3-8 si dále všimneme zajímavých měřítek textury. Aby opakováním definovaného vzorku vzniklo šikmé šrafování pod úhlem 45°, je třeba vzít v úvahu, že rozvinutý plášť válce pokrytého texturou tvoří obdélník o výšce 2 a délce zhruba 6,28 (=2π). Měřítka, která zmenšují vzorek textury, původně roztažený na celý plášť, jsou proto zadána v poměru zhruba odpovídajícímu délce stran rozvinutého pláště.

3.2.3 Průhlednost aneb Děravá tělesa snadno a rychle

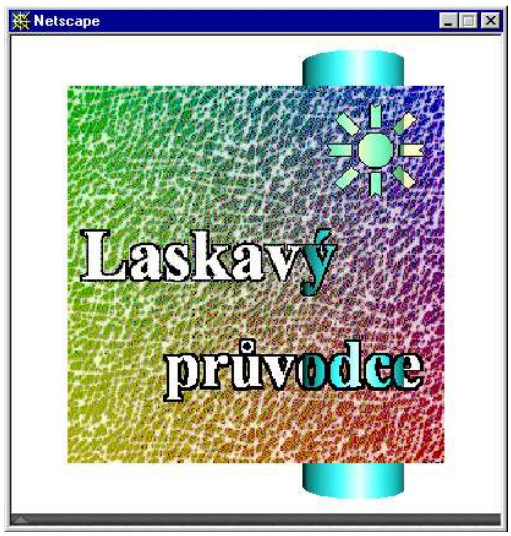
V předchozí části jsme zjistili, že textura může obsahovat informaci o průhlednosti. K čemu je taková věc dobrá, cožpak může být těleso v některých místech průhledné a jinde nikoliv? Ačkoliv se takové situace nevyskytují často, použití textury s průhledností je výhodné pro různé zajímavé efekty. Zcela průhledné části povrchu zmizí a je vidět skrze těleso na vzdálenější objekty. Zadní stěny tělesa přitom nepřekáží ve výhledu, protože prohlížeče se zadními stěnami nezabývají, není-li stanoveno jinak. Barva částečně průhledných částí textury je míchána s barvou zadních objektů, čímž lze vytvořit iluzi barevného skla, mraků či sloupce kouře.

Na obrázku 3-6 vidíme texturu z programu P-3-6, tentokrát doplněnou údaji o průhlednosti. Vnitřek nápisu má nastavenu plnou průhlednost, zatímco vnitřek sluníčka a jeho paprsků je částečně průhledný, konkrétně z jedné třetiny.

Pro lepší pochopení doporučujeme prohlédnout si odpovídající obrázek v barevné příloze.

Za kvádrem, na který je taková textura mapována, zřetelně vidíme další objekt – štíhlý, kovově lesklý válec. Všimněme si, jak je světle modrá barva válce míchána s původně sytě žlutou barvou slunce. Výsledná barva vzniká aditivním skládáním, vidíme tedy zelené odstíny jako výsledek „součtu“ odstínů modré a žluté, v daném případě v poměru 1:2.

Namísto bílého pozadí prosvítá symbolem slunce nevýrazná žluť.



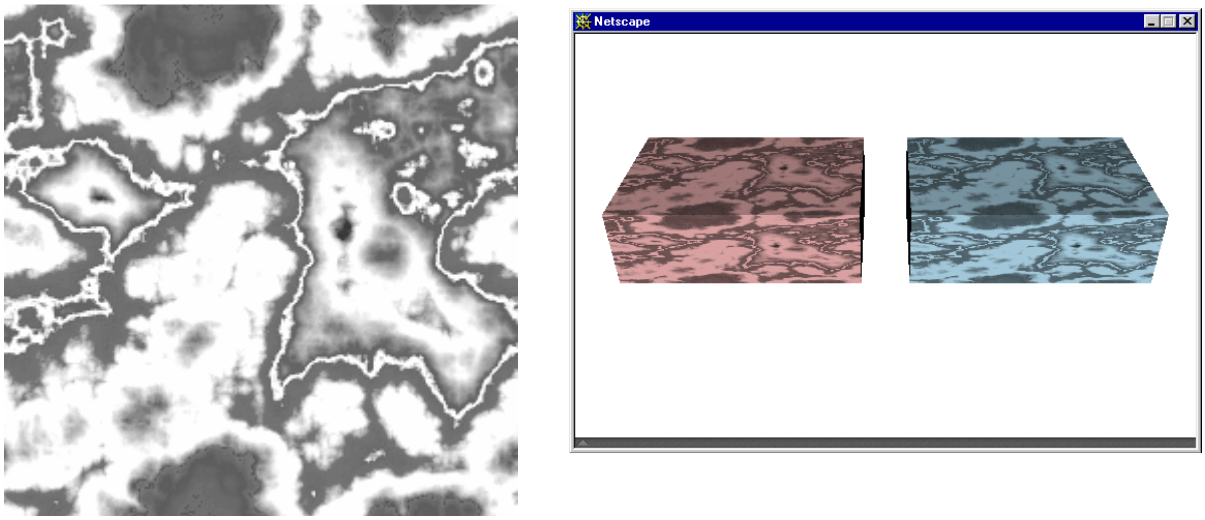
Obrázek 3-6: Použití průhledné a poloprůhledné textury

TIP: Pouze obrazový formát PNG dokáže ukládat informace o různém stupni průhlednosti. Formát GIF umí označit jen jedinou, zcela průhlednou barvu, formát JPG s průhledností nepracuje vůbec.

3.2.4 Materiál, textura nebo oboje?

Uzel **Appearance** dovoluje zadat jak materiál povrchu objektu, tak texturu, která jej pokrývá. Pokud to není nutné, je doporučováno nekombinovat obě informace dohromady, protože zobrazení takového tělesa je náročné. Parametr **material** přináší nutnost vypočítávat přesné osvětlení povrchu (odraz světla), parametr **texture** navíc přidává výpočty, které zajišťují mapování textury. Pro většinu případů je proto rozumné držet se dohody, že při použití textury je parametr **material** ponechán prázdný.

V některých případech ovšem kombinace obou parametrů může naopak zvýšit efektivitu. Je to tehdy, pokud je textura definována ve stupních šedi. Jednotlivé odstíny lze snadno modifikovat (obarvit) materiálem a s pomocí jediného šedotónového nebo černobílého obrázku vytvořit iluzi objektů vyrobených ze zcela odlišných materiálů. Dokladem tohoto přístupu je obrázek 3-7, na kterém je šedá textura mramoru mapována na dva kvádry. V kombinaci s difúzní barvou v uzlu **Material** vznikne jednou růžový, jindy modrý mramor.



*Obrázek 3-7: Šedá obrazová textura (vlevo) obarvená dále v uzlu **Material***

Kombinace barevného obrázku a materiálu má smysl jen ve výjimečných případech. Vzhledem k tomu, že barevná textura je chápána jako difúzní barva, nemá smysl vůbec použít parametr **diffuseColor** v materiálu. Do úvahy přicházejí pouze parametry **specularColor** a **shininess**, které na oblém povrchu vytvoří odlesky světla. Protože po použití materiálu textura okamžitě ztmavne, lze ji zesvětlit či dokonce mírně přebarvit citlivým nastavením parametru **emissiveColor**.

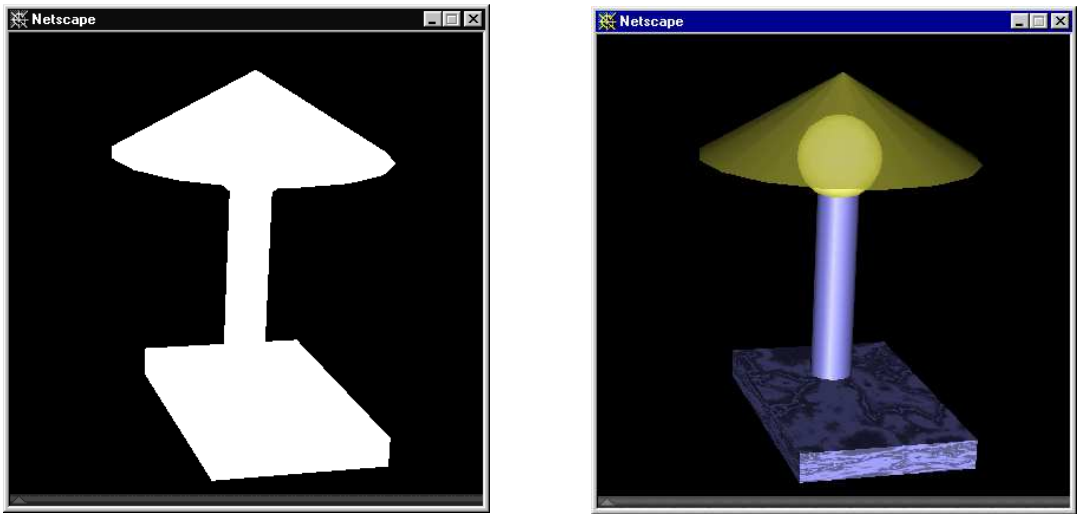
TIP: Současné použití barevné textury a barevného materiálu je nejenom výpočetně náročné, ale patří současně k jevům, které často zobrazují různé prohlížeče odlišně. Je proto lepší se takové kombinace vyvarovat.

Krok za krokem II – Materiál lampičky

V předchozí kapitole (3.1) jsme vytvořili model lampičky ze čtyř základních těles. Nyní již snadno zvládneme vylepšit její původně nezajímavý bílý vzhled přidáním barev a textur. Obrázek 3-8 pro srovnání ukazuje původní a nový vzhled poté, kdy podstavec byl vymodelován s pomocí textury mramoru, jejíž šedotónový obrázkový vzor byl obarven použitím materiálu modré difúzní barvy. Zbylé části lampičky již textury neobsahují, vzhled jejich povrch je ovlivňován pouze parametry uzlu **Material**. Nožka získala kovově lesklý vzhled pomocí parametru **specularColor**. Stínítko má žlutou difúzní barvu a je částečně průhledné, což zajišťuje parametr **transparency**. Koule, která představuje žárovku, dělá dojem, že vyzařuje světlo. Tento jev vznikl nastavením jediného parametru – **emissiveColor** na hodnotu žluté barvy.

Oproti původnímu modelu na obrázku 3-8 vlevo jsme dosáhli určitého optického zlepšení. V dalších částech se dozvíme, jak ještě lépe vytvarovat stínítko a jak konečně odstranit pochmurné černé pozadí.

Dříve se však v následující kapitole seznámíme s tím, jak efektivně zapisovat opakující se stejné části virtuálního světa.



Obrázek 3-8: Změna vzhledu modelu lampičky přidáním barev a textur

3.3 Proč se zbytečně opakovat? (DEF a USE)

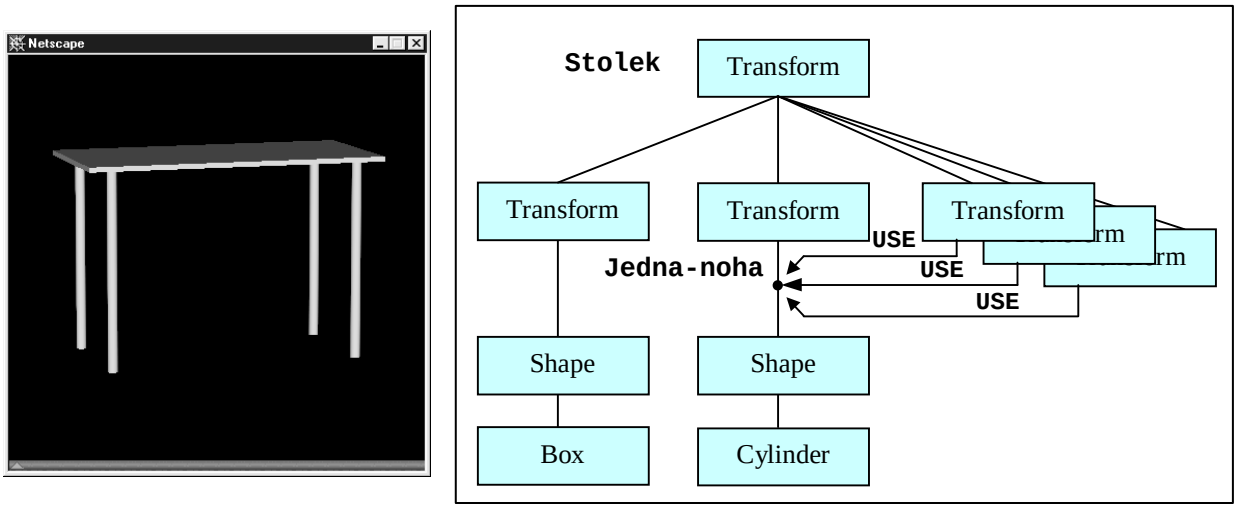
Ve světě kolem nás, ať skutečném či virtuálním, nacházíme mnoho věcí, které se sobě podobají tvarem nebo barvou a odlišují se pouze svojí polohou či velikostí. Mnohonásobné opakování popisu takto podobných objektů je z lidského hlediska pracné, z počítačového dokonce zbytečné. Existuje možnost, jak jednou zapsanou informaci opakovaně využít. Jakmile jednou pojmenujeme nějaký uzel (příkazem **DEF**), můžeme v další části souboru vytvořit jeho kopii příkazem **USE**. Vždy pochopitelně musíme uvést jméno, které jsme danému uzlu dříve přiřadili. Je sympatické, že tímto způsobem můžeme vytvářet kopie nejen samostatných uzlů (např. těles či materiálů), ale i celých stromových struktur.

Stolek z příkladu P-3-3 tak můžeme zapsat mnohem efektivněji. Jednu jeho nohu pojmenujeme (například **Jedna-noha)** a zbylé tři vytvoříme jako její kopii.

Stejně tak dobře využijeme tento přístup při zpracování barvy stolku. Pojmenovaný materiál (**Hnedy-povrch**) obarví desku stolu i jeho nohy. V ukázce jsou použitá jména zvýrazněna podtržením.

```
#VRML V2.0 utf8
DEF Stolek Transform {
  scale 0.01 0.01 0.01
  rotation 0 1 0 0.3
  children [
    Transform { # deska stolu
      translation 0 80 0
      children Shape {
        geometry Box {size 120 2 60}
        appearance
          DEF Hnedy-povrch Appearance {
            material Material {
              diffuseColor 1 0.6 0.2
            }
          }
      }
    }
    Transform {
      translation -50 40 –25
      children
        DEF Jedna-noha Shape
              { geometry Cylinder {radius 2 height 80 top FALSE}
                appearance USE Hnedy-povrch
              }
    }
    Transform {
      translation -50 40 25
      children USE Jedna-noha
    }
    Transform {
      translation 50 40 –25
      children USE Jedna-noha
    }
    Transform {
      translation 50 40 25
      children USE Jedna-noha
    }
  ]
}
```

Program P-3-9: Hnědý stolek s využitím **DEF** *a* **USE**



Obrázek 3-9: Model stolku z programu P-3-9. Vpravo je zobrazena vybraná část stromové struktury, která se týká pouze geometrie.

TIP: **Používání `DEF` a `USE` je všeobecně doporučováno, protože snižuje velikost souboru a může i urychlit zobrazování. Nic se však nesmí přehánět. Rutinní „množení“ uzlů by například v případě `Viewpoint` vedlo k úplnému zmatku – pod jedním jménem (viz parametr `description`) by bylo uživateli nabízeno několik, obecně různě umístěných stanovišť.**

Pro práci s **`DEF`** a **`USE`** platí několik pravidel:

1. Jméno, přiřazené uzlu příkazem **`DEF`**, je dostupné pouze v tomtéž souboru.
   Je-li tedy svět tvořen více soubory, nemůžeme příkazem **`USE`** vytvořit kopii objektu z jiného souboru.
   Lze to pouze speciálním příkazem **`PROTO`**, kterému je věnována samostatná kapitola 4.
   Výjimkou z tohoto pravidla jsou jména stanovišť (**`Viewpoint`**), která jediná jsou zvenčí „viditelná“.

2. V jednom souboru lze definovat stejné jméno víckrát.
   Příkaz **`USE JMÉNO`** je pak vztažen k nejbližšímu předchozímu výskytu **`DEF JMÉNO`**. Obecně je však vhodné dávat jména různá. V případě nedostatku jmen v zásobě lze doporučit systematickou tvorbu jmen typu `Material-00`, `Material-01`, … `Material-NN` apod.

3. Použít příkaz **`USE`** uvnitř stromu, ve kterém je tento strom teprve definován, je nesmysl. Zápis

```
#VRML V2.0 utf8
DEF A Transform {
  children [
    Shape { geometry Sphere {} }
    USE A
  ]
}
```

   je sice z formálního hlediska dobře, ve skutečnosti však způsobí chybu, v horším případě dokonce zacyklení prohlížeče, který se pokusí rekurzívním způsobem vytvořit strom o nekonečném počtu potomků. V něm bude teoreticky každá generace obsahovat jednu kouli a další nekonečný podstrom.

3.4 Obecnější tělesa a jiné zvláštnosti (ElevationGrid, Extrusion, Text, IndexedFaceSet, IndexedLineSet, PointSet + Color, Coordinate, FontStyle, Normal, TextureCoordinate)

Kdybychom měli tvořit virtuální světy jen s pomocí čtyř základních těles, příliš bychom fantazii rozvíjet nemohli. Je zřejmé, že pro popis obecnějších tvarů potřebujeme další uzly. Těchto doplňujících uzlů je celkem šest. První tři pracují s ploškami, ze kterých lze modelovat povrch objektů. Zbylé tři mají dvourozměrný charakter, ačkoliv jsou umísťovány do trojrozměrného prostoru:

1. Množina ploch (**`IndexedFaceSet`**)
2. Opláštění (**`Extrusion`**)
3. Výšková mapa (**`ElevationGrid`**)
4. Množina čar (**`IndexedLineSet`**)
5. Množina bodů (**`PointSet`**)
6. Nápis (**`Text`**)

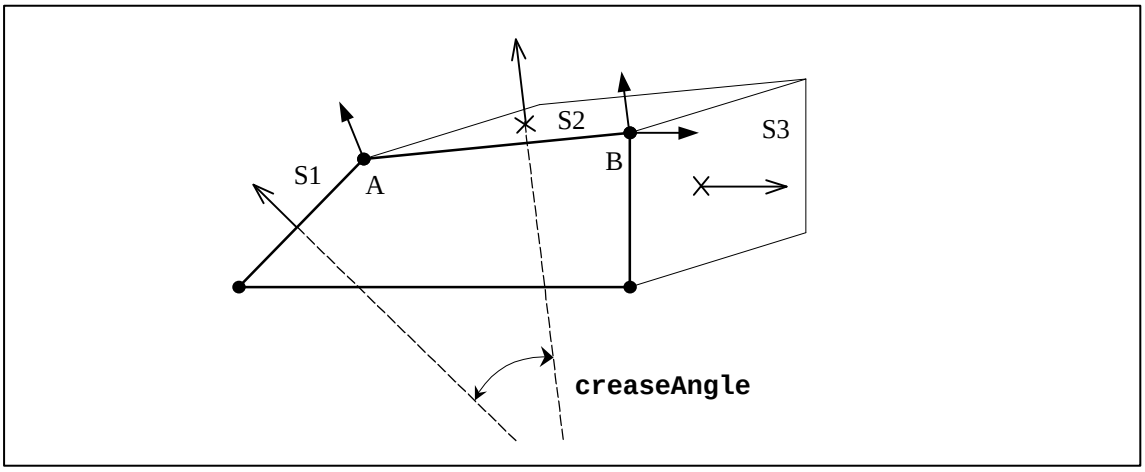
Na rozdíl od jednoduchých těles mají tyto uzly poměrně složitou strukturu. V mnoha případech jsou jejich parametry tvořeny dalšími, pomocnými uzly. Touto složitostí platíme za možnost téměř libovolného tvarování objektů.

Dříve, než se podíváme na jednotlivé uzly blíže, musíme se zaměřit na *vlastnosti ploch*. Ve virtuální realitě jsou téměř všechny objekty modelovány pomocí rovinných ploch, také zobrazovací systémy používají rovinnou plochu jako základní prvek zpracování. Nejinak je tomu u grafických karet s podporou 3D (prostorové) grafiky.

Plocha může být až překvapivě složitý útvar, vyznačující se řadou vlastností. Většina z nich má přitom přímý dopad na efektivitu zobrazení:

- **Rovinná** plocha má všechny své vrcholy položeny do téže roviny. Je-li plocha omezena třemi vrcholy, je vždy rovinná. Není-li rovinná, je nesnadné určit průběh jejího zakřivení či zalomení.
- **Konvexní** plocha se oproti nekonvexní dá lépe rozdělit na trojúhelníky a dále zpracovat jednoduchými metodami. Trojúhelníková plocha je vždy konvexní.
- **Jednostranná** plocha je zobrazována pouze z jedné (tzv. přivrácené) strany. Jakmile dojde k jejímu natočení tak, že na její přivrácenou stranu není vidět, lze celou plochu ze zobrazovacího systému dočasně vypustit. Povrch základních těles je tvořen jednostrannými plochami. **Oboustranné** plochy zvyšují výpočetní nároky na zobrazení.
- **Orientace vrcholů** plochy jde při jejich zadávání proti směru hodinových ručiček. Při splnění této dohody snadno určíme přivrácenou stranu podle pravidla pravé ruky – vrcholy jsou zapisovány tak, abychom je popořadě mohli pravou rukou „shrnout“ k sobě. Obrácená orientace určuje odvrácenou plochu.
- **Normála** k ploše je vektor o délce jedna, který určuje kolmici k povrchu v každém bodě plochy. Rovinná plocha má stejné normály po celém svém povrchu. Definováním individuálních normál pro jednotlivé vrcholy docílíme optického prohnutí rovinné plochy. Tento postup představuje základní trik při zobrazování oblých povrchů nahrazených množinou rovinných ploch. Rovné plochy se zdají být oblé tím, že jejich barevný odstín se plynule mění v závislosti na měnící se normále a tedy měnícím se úhlu dopadajícího světla.
- **Sevřený úhel** mezi sousedními dvěma plochami (lépe řečeno odchylka sousedních ploch) může určovat **ostrou hranu** nebo naopak opticky **hladké napojení**. Je-li odchylka rovna nule, plochy leží v téže rovině. Je-li odchylka malá, lze plochy považovat za část rovinné náhrady oblého povrchu a jejich společným vrcholům přiřadit vhodně nakloněné normály. Při větší odchylce jsou plochy považovány za rovinné, mají tedy stejnou normálu na celém povrchu a mezi nimi je výrazná, ostrá hrana.

- **Barva** na povrchu plochy je vypočítávána podle normál (a úhlu dopadajícího světla) nebo je zadána samostatně pro každý vrchol. Získá-li každý vrchol jinou barvu, vypočítávají se barevné odstíny vnitřních bodů plochy tzv. *stínováním* (odborně *bilineární interpolací*). Má-li však plocha jedinou společnou normálu, všechny její vnitřní body jsou vykresleny stejnou barvou.



Obrázek 3-10: Sousedící plochy a jejich normály

Některé z těchto vlastností ukazuje obrázek 3-10, na němž je nakreslen objekt s různě orientovanými plochami. Odchylka ploch S1 a S2, které sousedí s vrcholem A, je současně úhlem, který je svírán normálami těchto ploch. Právě tento úhel je porovnáván s hodnotou parametru **`creaseAngle`**, používaným při definici plošných objektů. Na obrázku 3-10 je tento úhel menší než zadaná mezní hodnota a proto je do vrcholu A přiřazen individuální normálový vektor, získaný jako průměr normálových vektorů ploch S1 a S2. U vrcholu B tomu tak není a proto si každá ze sousedních ploch zachová v tomto vrcholu svoji normálu. Ve výsledku je pak plocha S3 zobrazována jako rovinná, neboť je charakterizována jedinou normálou, platnou pro všechny její vrcholy. Plochy S1 a S2 se jeví oblé, protože mají ve svých vrcholech obecně různé normály.

Z hlediska rychlosti zobrazování je ideální, když je plocha rovinná, konvexní, orientovaná proti směru hodinových ručiček, jednostranná a s předem vypočítanými barvami ve vrcholech. Vhodnou plochou je zjevně trojúhelník. Z tohoto důvodu bývají veškeré ostatní plochy v počítači převáděny na množinu trojúhelníků. Abychom tento převod usnadnili a zajistili jeho správnost, můžeme v řadě geometrických uzlů VRML nastavit příslušné parametry a pomocné uzly, jak ukazuje tabulka T-3-6.

uzel	parametr	význam
`Coordinate { }`	**`point`**	seznam souřadnic vrcholů v trojrozměrném prostoru (3D)
`Normal { }`	**`vector`**	seznam normál v trojrozměrném prostoru (3D)
`Color { }`	**`color`**	seznam barev v barevném modelu RGB
`TextureCoordinate { }`	**`point`**	seznam bodů v rovině (2D), které slouží jako vztažné body pro nanesení textury

Tabulka T-3-6: Pomocné uzly pro zadávání ploch

Všechny čtyři pomocné uzly mají pouze jediný parametr, do kterého se ukládají číselné údaje. Zpravidla jsou to právě tyto pomocné uzly, které obsahují nejvíce dat a představují tak největší objem VRML souboru. Připomeňme si, že pokud je v parametru uloženo více hodnot, jsou uzavřeny do hranatých závorek a navzájem odděleny čárkami. Souřadnice protilehlých vrcholů jednotkové krychle bychom například zapsali do uzlu **`Coordinate`** takto:

```
Coordinate { point [ 0 0 0, 1 1 1] }
```

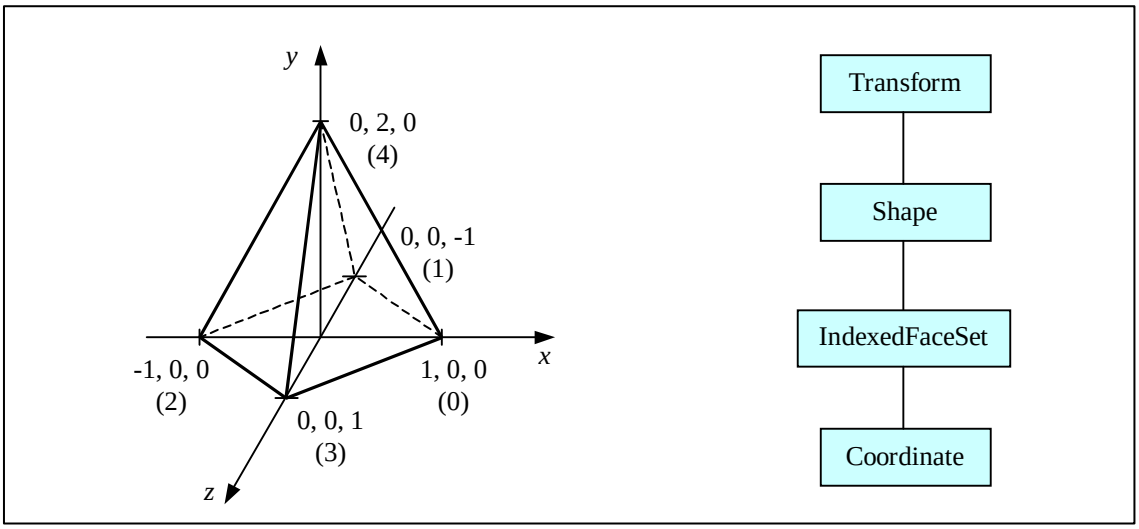
3.4.1 Množina ploch

Jakmile umíme zapsat základní stavební prvky, jakými jsou souřadnice bodů a hodnoty barev, můžeme z nich stavět složitější útvary – plochy. Nejvíce používaným uzlem je množina ploch (**IndexedFaceSet**). Dovoluje definovat zcela obecné těleso nebo libovolně zakřivenou prostorovou plochu tím, že popíše malé plošky, které daný objekt pokrývají. Název uzlu současně naznačuje, jakým způsobem je docíleno úspory paměti – každá plocha z množiny se definuje zapsáním indexů (pořadových čísel) vrcholů, nikoliv konkrétním uvedením jejich souřadnic. Když je jeden vrchol sdílen více plochami, jeho souřadnice stačí zapsat jen jednou a zapamatovat si jeho pořadí v seznamu vrcholů. U všech ploch, které takový vrchol použijí, se pro něj uvede jediné číslo – pořadí vrcholu.

```
Transform {
  children Shape {
    geometry IndexedFaceSet {
       coord Coordinate { point [
              1 0 0, 0 0 -1,       # body podstavy
             -1 0 0, 0 0 1,
              0 2 0 ]              # vrchol jehlanu
            }
       coordIndex [3 2 1 0 -1,     # podstava
             0 1 4 -1, 1 2 4 -1,   # stěny
             2 3 4 -1, 3 0 4 -1]
    }
  }
}
```

Program P-3-10: Čtyřboký jehlan definovaný uzlem **IndexedFaceSet**

Ukázka pravidelného čtyřbokého jehlanu v programu P-3-10 dokumentuje způsob, jakým se s vrcholy a plochami pracuje. Pořadová čísla vrcholů v parametru **coord** se počítají od nuly. Parametr **coordIndex** určuje, jakými vrcholy jsou jednotlivé plochy obklopeny. Protože plochy mohou mít více vrcholů, platí dohoda, že za posledním vrcholem každé plochy ze zapisuje číslo mínus jedna.



Obrázek 3-11: Vlevo umístění a souřadnice jehlanu, vpravo stromová struktura modelu

Pořadí, ve kterém se zadávají vrcholy, hraje roli při určení orientace plochy. Standardně jsou plochy chápány jako jednostranné a vrcholy jejich přední strany musí být uvedeny proti směru hodinových ručiček. Proto je podstava jehlanu v příkladu P-3-10 zadána tak, aby při pohledu zdola byla vidět. To vlastně znamená, že při pohledu shora, případně zevnitř jehlanu, je podstava zadána ve směru hodinových ručiček.

TIP: **Uzel IndexedFaceSet se podobně jako základní tělesa zapisuje do parametru geometry rodičovského uzlu Shape. Stejně tak je tomu i u dalších pěti uzlů uvedených v této kapitole.**

Ze všech uzlů má nejvíce parametrů právě **IndexedFaceSet**. Následující tabulka přináší jejich přehled:

parametr	iniciální hodnota	význam
coord	**prázdný uzel**	seznam vrcholů uložený v uzlu **Coordinate**
normal	**prázdný uzel**	seznam normál uložený v uzlu **Normal**
color	**prázdný uzel**	seznam barev uložený v uzlu **Color**
texCoord	**prázdný uzel**	seznam souřadnic bodů pro mapování textury uložený v uzlu **TextureCoordinate**
coordIndex	**[]**	posloupnosti indexů vrcholů jednotlivých ploch (zakončené číslem –1)
normalIndex	**[]**	posloupnosti indexů normál pro jednotlivé vrcholy či jednotlivé plochy
colorIndex	**[]**	posloupnosti indexů barev pro jednotlivé vrcholy či jednotlivé plochy
texCoordIndex	**[]**	posloupnosti indexů souřadnic textury pro jednotlivé vrcholy
colorPerVertex	**TRUE**	barvy v parametru **colorIndex** se vztahují na vrcholy, v opačném případě na plochy
normalPerVertex	**TRUE**	normály v parametru **normalIndex** se vztahují na vrcholy, v opačném případě na plochy
convex	**TRUE**	nápověda prohlížeči: všechny plochy jsou konvexní
ccw	**TRUE**	všechny plochy mají přivrácené strany zadávány posloupností vrcholů proti směru hodinových ručiček (*counterclockwise*)
solid	**TRUE**	plochy jsou jednostranné, tvoří povrch tělesa zobrazovaný pouze z vnější strany; v opačném případě jsou oboustranné
creaseAngle	**0**	mezní úhel, až do kterého jsou dvě sousední nerovinné plochy považovány za oblou náhradu zakřiveného povrchu (pozn.: *crease* je anglicky *puk na kalhotách*)

Tabulka T-3-7: Všech čtrnáct parametrů uzlu **IndexedFaceSet**. *Prázdný seznam je označen symbolem prázdných hranatých závorek* **[]**.

Velkým počtem parametrů se nenecháme zmást, protože si je rozdělíme do menších logických skupin. První skupina čtyř parametrů obsahuje jako potomky pomocné uzly uvedené v tabulce T-3-6. Jde o datové uzly se souřadnicemi, vektory a barvami, mezi kterými zatím nejsou definovány žádné vztahy či uspořádání.

Teprve další čtyři parametry dodají datům význam. Tyto parametry obsahují ve svých jménech slovo *Index*, pracují tedy pouze s celočíselnými hodnotami – pořadovými čísly údajů z dříve zmíněných datových uzlů. Parametry **colorIndex** a **normalIndex** mohou mít odlišný význam podle toho, zda tvůrce modelu přiřadil každému vrcholu vlastní barvu (o čemž vypovídá parametr **colorPerVertex**) nebo vlastní normálu (parametr **normalPerVertex**). *Současné použití* těchto dvou individuálních vlastností vrcholů bývá vzácné – buď je barva ve vrcholech již zadána a není ji nutno vypočítávat podle normál nebo je zadána pouze barva celé plochy a její odstíny se vyhodnotí podle normál ve vrcholech. Ve druhém případě se objekt často doplňuje uzlem **Material**, ve kterém jsou uvedeny parametry odrazu světla.

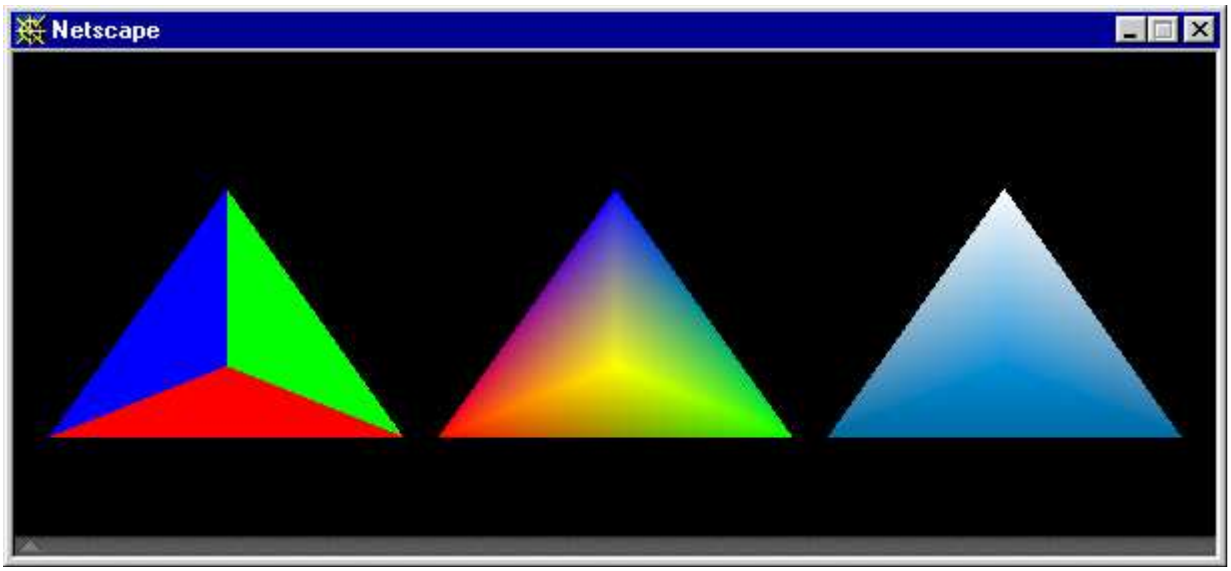
Program P-3-11 ukazuje různé způsoby nastavení parametrů uzlu **IndexedFaceSet** pro útvar, tvořený třemi trojúhelníky se společným vrcholem. Všechny trojúhelníky leží v jedné rovině, přesto lze pomocí barev vytvořit iluzi, že jejich společný vrchol poněkud vystupuje (viz obr. 3-12). Je tomu tak hlavně v případě individuálního nastavení hodnot ve vrcholech, tedy u prostředního a pravého obrázku. Všimněme si také, že pokud jsou pořadová čísla (indexy) vrcholů, barev či normál stejná pro všechny vrcholy, stačí ke správnému propojení těchto prvků uvést pouze parametr **coordIndex**.

```
Transform {
  children [
    Transform {               # každá plocha jinou barvou
      translation -2.2 0 0
      children Shape {
        geometry IndexedFaceSet {
          coord DEF VRCHOLY Coordinate {
                              point [-1 0 0,   1 0 0,
                                     0 1.4 0, 0 0.4 0]
          }
          color DEF BARVY Color {
              color [1 0 0, 0 1 0, 0 0 1, 1 1 0]
          }
          coordIndex [0 1 3 -1, 1 2 3 -1, 2 0 3 -1]
          colorPerVertex FALSE
        }
      }
    }
    Transform {                # každý vrchol jinou barvou
      children Shape {
        geometry IndexedFaceSet {
          coord USE VRCHOLY
          color USE BARVY
          coordIndex [0 1 3 -1, 1 2 3 -1, 2 0 3 -1]
        }
      }
    }
    Transform {                # každý vrchol má jinou normálu
      translation 2.2 0 0
      children Shape {
        appearance Appearance {
          material Material {
                diffuseColor 0 0.6 1
                specularColor 1 1 1
          }
        }
        geometry IndexedFaceSet {
          coord USE VRCHOLY
          normal Normal {vector [-.58 -.58 .58, .58 -.58 .58,
                                 0 0 1,       0 -0.6 0.8]}
          coordIndex [0 1 3 -1, 1 2 3 -1, 2 0 3 -1]
        }
      }
    }
  ]
}
```

Program P-3-11: Různé kombinace parametrů uzlu **`IndexedFaceSet`**

Na obrázku 3-12 vlevo je každý ze tří trojúhelníků obarven jedinou barvou. V souboru jsou sice u tohoto objektu uvedeny čtyři barvy, poslední je však nevyužita. Pojmenovaná skupina barev (jménem **BARVY**) bude totiž v dalším uzlu využita pro obarvení čtyř vrcholů za pomoci zápisu **USE**. Vidíme tedy, že vyšší počet barev není považován za chybu, zatímco menší počet by chybu znamenal.

Prostřední objekt je tvořen trojúhelníky, jejichž vrcholy mají přiřazeny individuální barvy. Vnitřní body trojúhelníků jsou automaticky obarvovány tak, aby vznikly plynulé barevné přechody. Vhodným přiřazením barev lze vyvolat iluzi zvlněného či lomeného povrchu tělesa.



Obrázek 3-12: Rovinné trojúhelníky se stejnou geometrií, ale s různými parametry. Zleva postupně plochy s jedinou barvou, plochy s barvami ve vrcholech, jednobarevné plochy s normálami ve vrcholech.

Velmi často používaným postupem při definování ploch je práce s normálami. Tehdy bývá povrchu přiřazena jediná barva, jak to odpovídá i reálnému světu. Podle úhlů, které svírají normálové vektory ve vrcholech se směrem paprsků dopadajícího světla, a podle směru pohledu avatara jsou automaticky vyhodnoceny odstíny barvy ve vrcholech. Vnitřní body ploch se dále vybarví stejně jako v předchozím případě[4]. Ukázka tohoto přístupu je na obr. 3-12 vpravo. Všechny tři trojúhelníky mají stejnou modrou barvu, která je kombinována s bílým světlem avatarovy čelní svítilny. Normála v horním vrcholu objektu směřuje proti iniciální pozici avatara, normála prostředního vrcholu je mírně skloněna dolů. Normály obou dolních vrcholů jsou pak skloněny ještě více.

TIP: Chceme-li, aby se odstíny barev vnitřních bodů plochy měnily podle jejího natočení, musíme přiřadit normály jednotlivým vrcholům.

Stejně jako na základní tělesa, také na množinu ploch můžeme nanést texturu. Platí přitom, že k jednomu uzlu **`IndexedFaceSet`** lze přiřadit právě jeden uzel, definující texturu. Pokud bychom chtěli mít na každé plošce jinou texturu, museli bychom plošky zapisovat jako samostatné uzly.

K přesnému umístění textury na povrch ploch slouží dvojice parametrů **`texCoord`** a **`texCoordIndex`**. První obsahuje uzel **`TextureCoordinate`** se souřadnicemi rovinných bodů textury, které mají být posléze uchyceny na jednotlivé prostorové vrcholy ploch. Snadno si tento postup představíme, když považujeme texturu za čtverec vyrobený z pružné, barevné látky. Na rozličné body této látky kápneme lepidlo a tato místa připevníme bodovým spojením na vrcholy prostorového objektu. Textura obklopí povrch objektu, přitom se různě roztáhne, případně výraznějším způsobem deformuje.

Tuto činnost dokumentuje program P-3-12 a jemu odpovídající obrázek. Již známá obrázková textura je mapována na tři stěny kvádru, popsaného uzlem **`IndexedFaceSet`** (levý dolní zadní roh tohoto objektu leží v počátku souřadnic). Kdyby bylo použito přímo těleso kvádr (**`Box`**), textura by se nanášela na všechny stěny a

[4] Metoda, při které se na základě známých barev ve vrcholech vypočítají barevné odstíny všech vnitřních bodů plochy, se odborně nazývá *Gouraudovo stínování*.

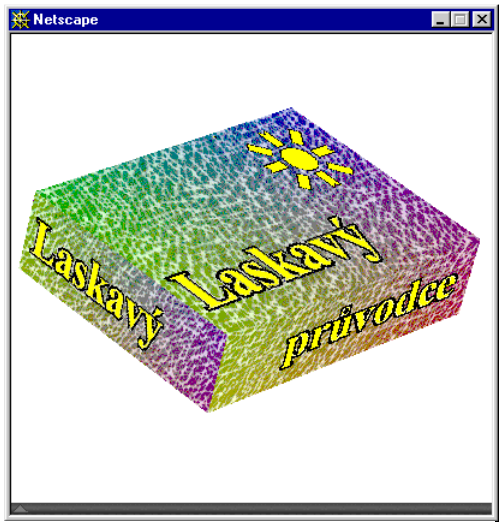
její měřítko by se měnilo podle rozměrů stěn. My však tentokrát požadujeme, aby textura plynule přecházela z horní na čelní stěnu, zatímco na levé boční stěně bude pouze výřez obrázku, obsahující část nápisu. Všechny požadavky snadno splníme vhodným výběrem bodů na textuře.

V ukázce je vybráno celkem osm bodů textury, které budou dále přichyceny na vrcholy ploch. Přichycení je dáno hodnotami parametru **`texCoordIndex`**. Je třeba dávat pozor na správné pořadí indexů. Při chybném zadání pořadí dochází k převrácení nebo zkroucení textury. Vhodným zápisem naopak docílíme velké škály různých transformací textury.

TIP:

1. **Při současném použití uzlů `Material` a `Color` nahrazují barvy z uzlu `Color` hodnoty parametru `diffuseColor` v materiálu.**
2. **Současné použití `Color` a textury nemá smysl, neboť textura přepíše barvy z uzlu `Color`.**

```
#VRML V2.0 utf8
Transform {
  children
    Shape {
      geometry IndexedFaceSet {
        coord Coordinate {
          point [0 1 3, 4 1 3,
                 4 1 0, 0 1 0,
                 0 0 3, 4 0 3, 0 0 0]
        }
        coordIndex [0 1 2 3 -1,
                    4 5 1 0 -1, 6 4 0 3 -1]
        texCoord TextureCoordinate {
          point [0 0.1, 1 0.1,
                 0 0.4, 1 0.4,
                 0 0.7, 1 0.7,
                 0 1,   1 1]
        }
        texCoordIndex
           [2 3 7 6 -1,   # horní stěna
            0 1 3 2 -1,   # čelní stěna
            2 3 5 4 -1]   # boční stěna
      }
      appearance Appearance {
        texture ImageTexture {
                url "obr-rgb.gif"
        }
      }
    }
}
```



Program P-3-12: Textura mapovaná různými způsoby na geometrii uzlu **`IndexedFaceSet`**

Velký počet parametrů uzlu **`IndexedFaceSet`** není jedinou nepříjemností, se kterou se při jeho používání setkáme. Ještě složitější je přesné pochopení významu parametrů, obsahujících indexy vrcholů, normál, barev a souřadnic textury. Hodnoty v těchto parametrech mohou podle situace označovat vlastnosti jedné každé plochy, jednoho každého vrcholu a dokonce přiřazovat jednomu vrcholu několik vlastností podle toho, k jaké ploše patří.

Vezměme si například barvy, které by teoreticky mohly být použity v objektu složeném ze tří trojúhelníkových ploch (viz program P-3-11):

- Pokud má každá plocha svoji barvu, obsahuje uzel **`Color`** umístěný do parametru **`color`** právě 3 barvy. Parametr **`colorPerVertex`** je nastaven na **`FALSE`**. Je-li parametr **`colorIndex`** prázdný, barvy jsou plochám přiřazovány ve stejném pořadí, v jakém byly zapsány do uzlu **`Color`**. Není-li **`colorIndex`** prázdný, obsahuje indexy barev pro jednotlivé plochy, přičemž v tomto případě může být počet barev v uzlu **`Color`** menší než počet ploch (tatáž barva se může použít pro více ploch).

- Pokud má každý vrchol vlastní barvu, přičemž vrchol sdílený více plochami si svoji barvu udržuje, potřebujeme pro 4 vrcholy zadat 4 barvy. Parametr **`colorPerVertex`** je tentokrát nastaven na **`TRUE`**, parametr **`colorIndex`** je prázdný. Barvy jsou uzlům přiřazovány ve stejném pořadí, v jakém byly zapsány do uzlu **`Color`**.

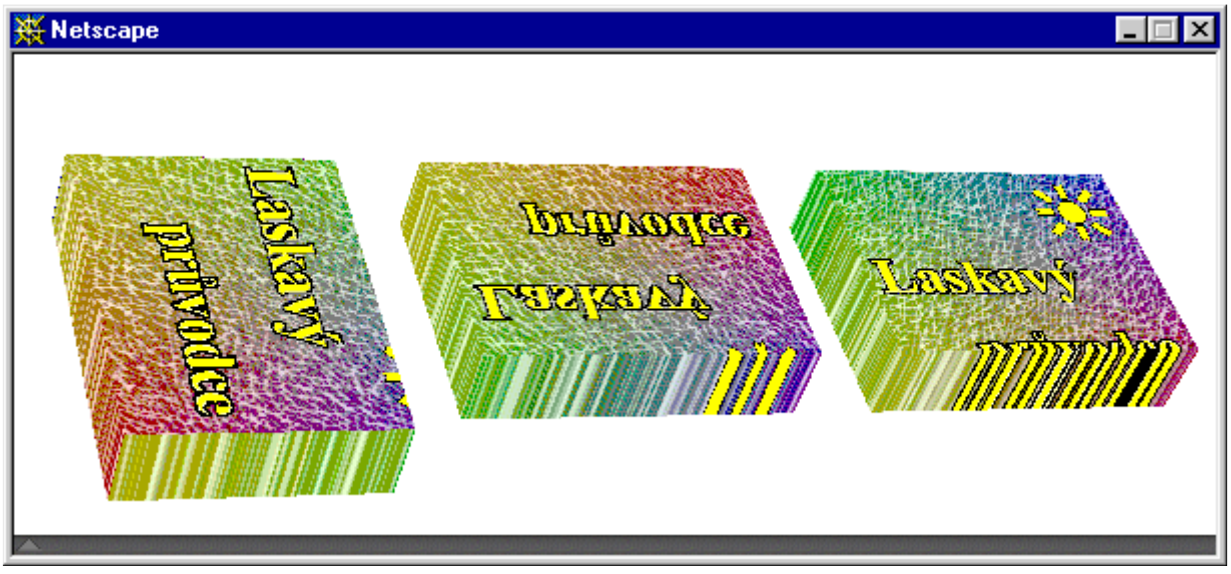
- Poslední možností je individuální obarvení vrcholů s ohledem na jejich příslušnost k plochám. Tehdy je v uzlu **`Color`** libovolný počet barev, parametr **`colorPerVertex`** je nastaven na **`TRUE`** a obsah parametru **`colorIndex`** je strukturován podobně jako v případě parametru **`coordIndex`**. Obsahuje posloupnosti barev jednotlivých vrcholů na obvodu plochy, zakončené hodnotou minus jedna. Praktickým příkladem takového zacházení s barvami je obarvení pravidelného n-bokého hranolu. Vrchol má jinou barvu, když je zařazen do seznamu vrcholů podstavy a jinou barvu, když je zařazen do posloupnosti ohraničující boční stěnu.

Zcela stejným způsobem se pracuje s normálami. Kombinací parametrů **`normal`**, **`normalPerVertex`** a **`normalIndex`** docílíme obdobných efektů s normálami jako kombinací parametrů **`Color`**, **`colorPerVertex`**, a **`colorIndex`** v případě barev.

Relativně nejjednodušší je situace se souřadnicemi textury. Ty nemohou být přiřazeny individuálně plochám, ale vždy se vážou k vrcholům:

- Obsahuje-li parametr **`texCoord`** uzel **`TextureCoordinate`** se seznamem souřadnic, pak parametr **`texCoordIndex`** má obdobnou strukturu jako **`coordIndex`** – viz též program P-3-12. Je-li ovšem **`texCoordIndex`** prázdný, souřadnice textury jsou uzlům přiřazovány ve stejném pořadí, v jakém byly zapsány do uzlu **`TextureCoordinate`**.

- Pokud je parametr **`texCoord`** prázdný, je použito *automatické mapování* textury. Největší rozměr objektu **`IndexedFaceSet`** v ose *x*, *y* nebo *z* je chápán jako vodorovná hrana mapované textury, druhý největší rozměr určuje svislou hranu textury. Měřítko textury je upraveno vždy tak, aby celá její šířka byla mapována na objekt, ze svislého rozměru textury je však použit jen takový díl, který odpovídá poměru druhého a prvního největšího rozměru objektu. Textura je kladena pouze jednou, bez opakování. Znamená to, že na plochách, které jsou kolmé k rovině textury, uvidíme jen pruhy vzniklé opakováním okrajů obrazu. Na ostatních plochách bude textura zobrazena normálně.



Obrázek 3-13: Při automaticky mapované textuře na geometrii uzlu **`IndexedFaceSet`** *jsou rozměry a orientace textury odvozeny z mezních rozměrů geometrie objektu*

Obrázek 3-13 ukazuje nevýhody automatického mapování textury. Kvádr vlevo má nejdelší rozměr v ose *z*, druhý největší rozměr je v ose *x*. Obrázek textury je proto orientován tak, aby jeho vodorovná hrana směřovala ve smyslu osy *z*. Ve zbylém směru jsou z obrázku použity jen tři čtvrtiny, jak odpovídá poměru hran horní stěny kvádru 3:4.

Na prostředním kvádru je textura převrácena, protože orientace os s prvním a druhým nejdelším rozměrem není proti směru hodinových ručiček. O nápravu se pokouší model vpravo, kde je do uzlu **`Appearance`** zařazen v předchozích objektech nepoužitý uzel **`TextureTransform`**. Jeho parametr měřítka textury má hodnotu **`scale 1 -1`**, čímž je převrácení obrázku napraveno. Obrázek je však mapován od své horní strany, zatímco v předchozích dvou případech byl mapován zdola.

TIP: Automatické mapování textury nedává vždy očekávané a dobré výsledky. Je proto lépe využít uzlu `TextureCoordinate`. Patrně nejlepší cestou je nastavení textury v jiném programu s následným exportem dat do formátu VRML.

3.4.2 Opláštění

Poté, co jsme zvládli záludnosti množiny ploch, snadno si poradíme s dalšími uzly. Často používaným uzlem, který pracuje s plochami, je uzel **`Extrusion`**. Ten je určen pro popis takových objektů, jejichž povrch vznikne postupným přesouváním dvourozměrného profilu po prostorové trajektorii. Profil je tvořen jedinou lomenou čárou, která může být při přesunu transformována. Počet jejích bodů se však nemění. Také trajektorie je definována jedinou lomenou čárou.

Odborně se takové objekty nazývají translační nebo rotační tělesa. Vždy je lze charakterizovat rovinným profilem, který je dále posouván a otáčen v prostoru. Příkladů nalezneme kolem sebe velké množství:

trup lodi	profil představuje žebra, trajektorie je kýlem. Profil i trajektorie jsou neuzavřené lomené čáry
váza	profilem je kružnice (dno), trajektorií svislá osa vázy. Uzavřená profilová křivka je na své trajektorii zvětšována a zmenšována, trajektorie je neuzavřená.
prstenec	kruhový profil (průřez prstence) je přesouván po kruhové trajektorii. Obě křivky jsou uzavřené

Počet parametrů uzlu **`Extrusion`** je oproti množině ploch nižší, některé z nich mají dokonce stejná jména i význam jako parametry množiny ploch:

parametr	iniciální hodnota	význam
`crossSection`	**`[1 1, 1 -1, -1 -1, -1 1, 1 1]`**	posloupnost bodů v rovině určující profil, tj. obrysovou křivku
`spine`	**`[0 0 0, 0 1 0]`**	posloupnost bodů v prostoru určující trajektorii
`scale`	**`[1 1]`**	seznam kladných měřítek pro každou další polohu obrysu
`orientation`	**`[0 0 1 0]`**	seznam otočení pro každou další polohu obrysu
`beginCap`	**`TRUE`**	povolení vykreslování dolní podstavy (profilu v počáteční poloze)
`endCap`	**`TRUE`**	povolení vykreslování horní podstavy (profilu v koncové poloze)
`convex`	**`TRUE`**	profil je konvexní
`ccw`	**`TRUE`**	profil je zadán proti směru hodinových ručiček
`solid`	**`TRUE`**	všechny plochy jsou jednostranné
`creaseAngle`	**`0`**	mezní úhel pro určení hladkého napojení sousedních ploch

Tabulka T-3-8: Parametry uzlu **`Extrusion`**

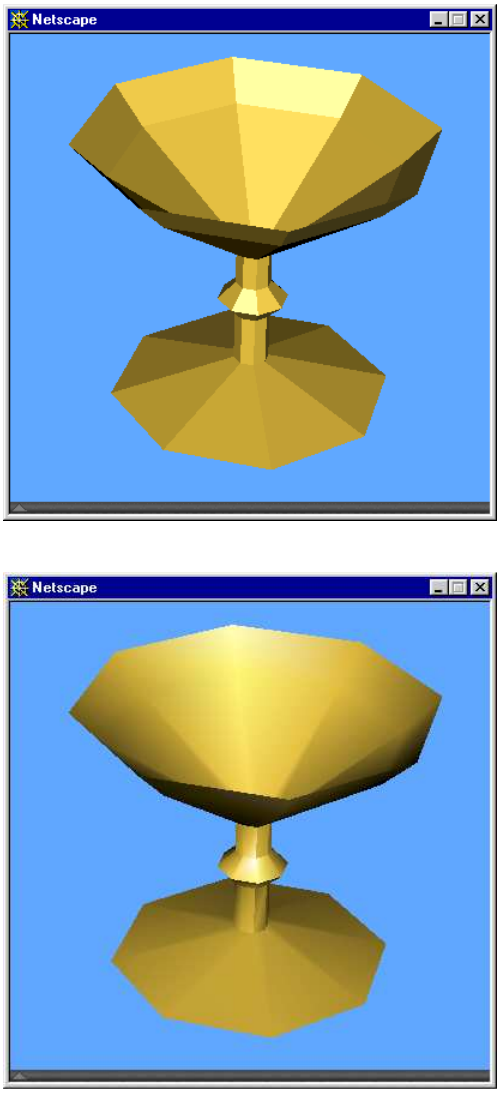
Jak je vidět z tabulky T-3-8, iniciální hodnoty definují jednoduché těleso se čtvercovým uzavřeným profilem (**`crossSection`**). Profil je opakovaně umísťován do prostoru podle poloh, které jsou uvedeny v parametru **`spine`**. Standardně je tedy čtverec umístěn nejprve do počátku souřadnic a poté o 1 m výše ve směru osy *y*. Opláštěním těchto dvou poloh profilu vzniknou čtyři boční plochy. S výjimkou první polohy profilu může být každý z následujících profilů zmenšen či zvětšen (**`scale`**) a navíc otočen (**`orientation`**). Je vidět, že počet

hodnot v seznamech měřítek a otočení je o jednu nižší než počet bodů trajektorie.Výjimku tvoří uvedení jediné hodnoty – ta potom platí pro všechny profily.

Standardně je dále vytvořena dolní podstava určená první polohou profilu (**beginCap**) a horní podstava určená poslední polohou profilu (**endCap**). Není-li stanoveno jinak, všechny vzniklé plochy jsou jednostranné (**solid**). Poslední čtyři parametry z tabulky T-3-8 mají stejný význam jako stejnojmenné parametry uzlu **IndexedFaceSet**.

Program P-3-13 ukazuje vytvoření jednoduchého poháru, který je definován uzavřeným profilem z osmi vrcholů a trajektorií o osmi bodech. Povrch je z materiálu připomínajícího zlato. Příslušné obrázky dokumentují použití parametru **creaseAngle**. Horní pohár má tuto hodnotu nastavenu na nulu, hrany mezi plochami jsou zřetelně ostré. Pohár dole má tento úhel nastaven na 0.78 (přibližně 45º), což způsobuje optické vyhlazení povrchu.

```
#VRML V2.0 utf8
Transform {
children  Shape {
    appearance Appearance {
      material Material {
         diffuseColor 1 0.8 0.2
         specularColor 1 1 1
      }
    }
    geometry Extrusion {
      crossSection [ 1 0, 0.7 0.7,
                     0 1, -0.7 0.7,
                    -1 0, -0.7 -0.7,
                     0 -1, 0.7 -0.7,
                     1 0]
      spine[ 0 0   0,
             0 0.2 0,
             0 0.5 0,
             0 0.6 0,
             0 0.7 0,
             0 0.9 0,
             0 1.4 0,
             0 1.6 0]
      scale[ 0.8 0.8
             0.1 0.1,
             0.1 0.1,
             0.2 0.2,
             0.1 0.1,
             0.1 0.1,
             0.8 0.8,
             1    1]
      solid FALSE
      endCap FALSE
      creaseAngle 0.78
    }
  }
}
```



Program P-3-13: Zlatavý pohár definovaný uzavřeným profilem z osmi vrcholů a trajektorií o osmi bodech. Povrch má zalomené nebo oblé plochy podle hodnoty parametru **creaseAngle**.

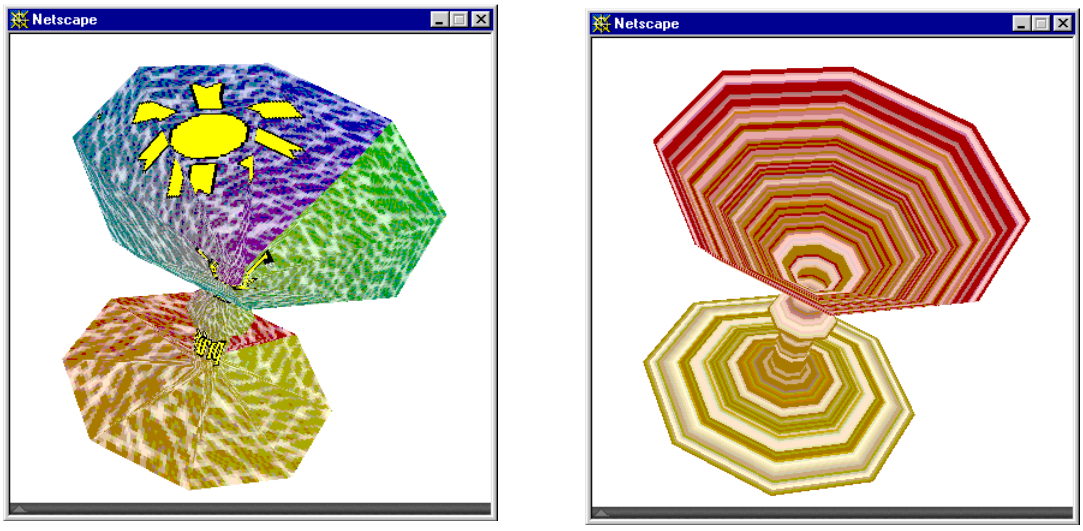
V ukázce je záměrně nastaven parametr **solid** na **FALSE**, aby u poháru byly zobrazovány i vnitřní plochy. Horní podstava (víko) je odstraněna hodnotou **FALSE** v parametru **endCap**.

Je zajímavé, že stejný pohár lze vygenerovat i jiným způsobem. Zatímco v programu P-3-13 je kruhový profil *vytažen* po svislé ose, na pohár můžeme také nahlížet jako na obrysovou křivku (v bokorysu), která je kolem osy poháru *otáčena*. Takové rotační těleso je popsáno programem P-3-14. Všimněme si, že seznam **spine**, představující trajektorii, je tvořen stále stejnými prostorovými souřadnicemi. Otáčený profil totiž není posouván, je pouze otáčen. Parametr **orientation** proto obsahuje seznam postupných otáčení kolem osy *z*. Výsledný objekt je nakonec „postaven“ otočením kolem osy *x* v rodičovském uzlu **Transform**.

Uzly **Extrusion** vygenerují v obou případech zcela stejnou geometrii poháru. Rozdíl mezi nimi se výrazně projeví, když na pohár naneseme texturu. Obrázek 3-14 ukazuje výsledky programů P-3-13 a P-3-14 s automaticky mapovanou texturou. Vlevo je pohár vzniklý posouváním profilu. Obrázková textura z dřívějších kapitol je sice zkreslená, ale stále rozpoznatelná. Obrázek je nanesen celkem třikrát - shora, zdola a dále ovinut kolem osy. Pohár vpravo vznikl otáčením profilu a také textura má výrazně rotační charakter. Nelze již rozpoznat původní obrázek.

```
#VRML V2.0 utf8
Transform {
  rotation 1 0 0 -1.57
  children  Shape {
    appearance Appearance {
      texture ImageTexture { url "obr-rgb.gif"}
    }
    geometry Extrusion {
      crossSection [ 0 0 ,    0.8 0,   0.1 0.2,
                     0.1 0.5, 0.2 0.6, 0.1 0.7,
                     0.1 0.9, 0.8 1.4, 1 1.6]
      spine[ 0 0 0, 0 0 0, 0 0 0,
             0 0 0, 0 0 0, 0 0 0,
             0 0 0, 0 0 0, 0 0 0]
      orientation [0 0 1 0,     0 0 1 0.78,  0 0 1 1.57,
                   0 0 1 2.35,  0 0 1 3.14,  0 0 1 -2.35,
                   0 0 1 -1.57, 0 0 1 -0.78, 0 0 1 0]
      solid FALSE
      endCap FALSE
      beginCap FALSE
      creaseAngle 0.78
    }
  }
}
```

Program P-3-14: Pohár popsaný otáčeným obrysem



Obrázek 3-14: Automaticky mapovaná obrazová textura na pohár, který vznikl posouváním profilu (vlevo) a otáčením profilu (vpravo).

TIP: Pro texturu na opláštění je vhodné volit obrázky s nezřetelnými, opakujícími se motivy, jako např. mramor, kámen, tepaný kov, dřevo.

Na závěr povídání o uzlu **`Extrusion`** se ještě zastavíme u transformací, které jsou prováděny s jednotlivými profily. Tyto transformace jsou složitější, než by se na první pohled mohlo zdát. Každý profil získává svoji vlastní souřadnicovou soustavu, na kterou jsou postupně uplatňovány transformace. Z hlediska jejich zpracování je nejdůležitější rovina, ve které profil leží. Tato rovina je určena předchozím a následujícím bodem trajektorie – *půlí úhel*, který svírají úsečky spojující aktuální bod trajektorie s jeho sousedy.

Leží-li všechny body trajektorie na jedné přímce, jako tomu bylo v programu P-3-13, je situace jednoduchá. Všechny profily leží v rovinách rovnoběžných s rovinou *xz*. Jakmile však přestanou body trajektorie ležet v přímce, dochází automaticky k odpovídajícímu natáčení rovin profilů, a to *bez ohledu* na hodnoty otočení uložené v parametru **`orientation`**.

Uvedený postup můžeme shrnout do následujících kroků, prováděných s profilem, resp. s rovinou, ve které profil leží:

1. nejprve je upraveno měřítko profilu v rovině profilu (**`scale`**),
2. rovina profilu je posunuta do příslušného bodu trajektorie (**`spine`**),
3. profil je příslušným způsobem otočen (**`orientation`**), přitom otáčení je vztaženo k rovině profilu, nikoliv k souřadnicové soustavě celého virtuálního světa.

3.4.3 Výšková mapa

Poslední ze speciálních uzlů pro manipulaci s plochami je uzel **`ElevationGrid`**. Slouží především k vytvoření základního tvaru krajiny, definované pomocí pravoúhlé sítě vrcholů s příslušnými výškami. Stejně tak dobře se dá využít k popisu plasticky zdeformovaného povrchu obdélníkové plochy. Přehled základních parametrů tohoto uzlu je uveden v tabulce T-3-9.

parametr	iniciální hodnota	význam
`xDimension`	**`0`**	počet vzorků (vrcholů sítě) v ose *x*
`zDimension`	**`0`**	počet vzorků (vrcholů sítě) v ose *z*
`xSpacing`	**`1.0`**	kladná vzdálenost mezi vzorky v ose *x*
`zSpacing`	**`1.0`**	kladná vzdálenost mezi vzorky v ose *z*
`height`	**`[]`**	pole výšek všech vrcholů sítě
`ccw`	**`TRUE`**	přivrácená strana mapy je vidět při pohledu shora (proti ose *y*)
`solid`	**`TRUE`**	mapa je jednostranná
`creaseAngle`	**`0`**	mezní úhel pro určení hladkého napojení sousedních ploch

Tabulka T-3-9: Základní parametry výškové mapy

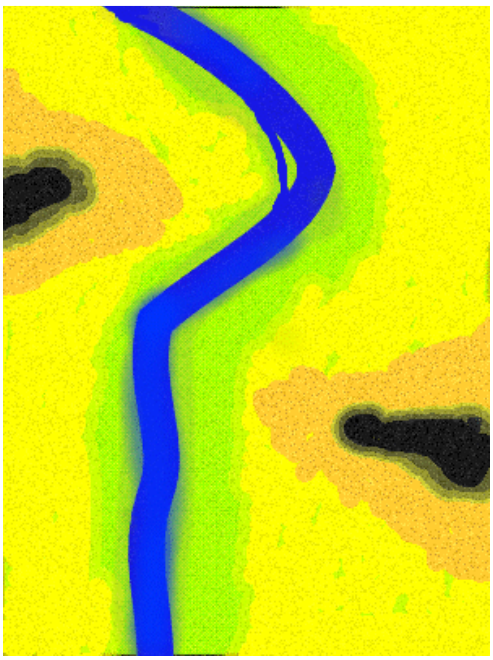
Jak je z tabulky vidět, mapa je standardně umístěna do roviny *xz*, tj. tam, kde je očekávána podložka či pevná zem. První čtyři parametry v tabulce T-3-9 ukazují, že počet vrcholů sítě může být odlišný ve směru osy *x* a *z* (**`xDimension`**, **`zDimension`**), stejně tak mohou být nastaveny jiné hodnoty pro pravidelné vzdálenosti řadami (**`zSpacing`**), resp. sloupci (**`xSpacing`**) vrcholů výškové mapy.

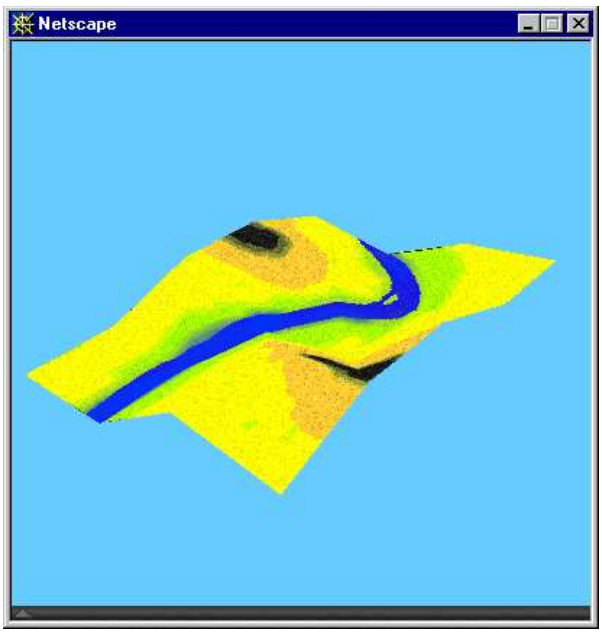
Seznam výšek, reprezentující souřadnice *y* jednotlivých vrcholů sítě, je zapsán v parametru **`height`**. Tento seznam je uspořádán po řádcích. Každý řádek proto obsahuje tolik výšek, kolik je hodnota zapsaná v parametru **`xDimension`**. Protože osa *z* míří při standardním uspořádání proti pozorovateli, lze říci, že řádky se zapisují od nejvzdálenějšího k nejbližšímu, jinak řečeno shora dolů.

Poslední tři parametry v tabulce T-3-9 jsou již známé specifikace charakteru plošek ležících mezi vrcholy sítě.

Výšková mapa se často kombinuje s obrazovou texturou, ať již ručně namalovanou nebo vzniklou z letecké fotografie. Jednoduchý příklad je ukázán na obrázku 3-15, kde je ručně nakreslený obraz krajiny nanesen na malou síť o rozměrech 5 x 4 vrcholy. Program P-3-15, pomocí kterého byla tato krajina zobrazena, obsahuje

potřebnou transformaci textury (převrácení ve svislém směru). Tato úprava je nutná kvůli způsobu zadávání bodů výškové mapy – zápis řádků shora dolů znamená automatické umístění počátečního bodu textury do levého *horního* rohu, zatímco vztažný bod obrázku se ve skutečnosti nachází v levém *dolním* rohu.





Obrázek 3-15: Obrázek vlevo je nanesen na výškovou mapu (vpravo)

```
#VRML V2.0 utf8
Transform {
  rotation -0.5 1 0 -0.8
  scale 1 0.3 1
  children  Shape {
    appearance Appearance {
      texture ImageTexture { url "mapa-rgb.gif"}
      textureTransform TextureTransform { scale 1 -1 }
    }
    geometry ElevationGrid {
      xDimension  4
      zDimension  5
      height [ 1 0 1 1, 2 1 0 1, 2 0 0 2,
               1 0.5 2 2, 1 0.5 2 1 ]
    }
  }
}
```

Program P-3-15: Výškové pole s texturou

Výše uvedené parametry uzlu **ElevationGrid** postačovaly pro jednoduché zadání výškové mapy. Podobně jako u množiny ploch, také zde můžeme geometrii mapy doplnit pomocnými údaji o barvách, normálách a souřadnicích textury pro jednotlivé vrcholy. Tabulka T-3-10 uvádí potřebné parametry. Do tří parametrů se přiřazují uzly se seznamem charakteristických hodnot (parametry **color**, **normal** a **texCoord**), další dva parametry určují, zda barvy a normály patří individuálním vrcholům nebo plochám vyplňujícím síť výškové mapy. Souřadnice textury jsou vždy vztaženy k jednotlivým vrcholům sítě.

parametr	iniciální hodnota	význam
color	**prázdný uzel**	seznam barev v uzlu **Color**

normal	**prázdný uzel**	seznam normál v uzlu **Normal**
texCoord	**prázdný uzel**	seznam souřadnic textury v uzlu **TextureCoordinate**
colorPerVertex	**TRUE**	barvy v parametru **color** se vztahují na vrcholy, jinak na plochy
normalPerVertex	**TRUE**	normály v parametru **normal** se vztahují na vrcholy, jinak na plochy

Tabulka T-3-10: Doplňkové parametry výškové mapy

3.4.4 Čáry a body

Někdy je vhodné doplnit prostorový virtuální svět i takovými geometrickými útvary, které sice nemají charakter těles, ale které dokáží elegantně a úsporně naznačit tvar nebo vlastnosti virtuálních objektů. Takovými uzly jsou množina čar (**IndexedLineSet**) a bodů (**PointSet**). Z možných příkladů jejich použití uveďme:

- lana, provazy, dráty (množina čar)
- viditelné světelné paprsky (množina čar)
- jednoduchý čárový obrázek či graf na povrchu tělesa (množina čar namísto textury)
- padající kapky vody (množina bodů)
- mák na housce, špína na povrchu tělesa (množina bodů namísto textury)

Nejjednodušší strukturu má množina bodů. Obsahuje pouze dva parametry (viz tabulka T-3-11), do nichž se zapisují uzly se souřadnicemi bodů a jejich barvami. Počet barev by měl být shodný s počtem bodů. Nechceme-li definovat barvy pro každý bod zvlášť, ponecháme parametr **color** prázdný a zadáme jedinou barvu materiálu do sourozeneckého uzlu **Appearance**.

parametr	iniciální hodnota	význam
coord	**prázdný uzel**	seznam bodů v uzlu **Coordinate**
color	**prázdný uzel**	seznam barev bodů v uzlu **Color**

*Tabulka T-3-11: Parametry uzlů **PointSet** a **IndexedLineSet***

Množina čar obsahuje oproti množině bodů navíc další tři parametry, které blíže určují význam použitých barev. V tomto smyslu množina čar připomíná množinu ploch. Kombinacemi hodnot parametrů **color**, **colorIndex** a **colorPerVertex** lze stanovit, zda se barvy přiřadí celým čárám, jednotlivým vrcholům samostatně či podle toho, kterou čáru zakončují, zda se barvy budou moci opakovat apod.

parametr	iniciální hodnota	význam
coordIndex	**[]**	posloupnosti indexů vrcholů jednotlivých čar (zakončené číslem –1)
colorIndex	**[]**	posloupnosti indexů barev pro jednotlivé vrcholy či celé čáry
colorPerVertex	**TRUE**	barvy v parametru **colorIndex** se vztahují na vrcholy, jinak na čáry

*Tabulka T-3-12: Další parametry uzlu **IndexedLineSet***

Souřadnice prostorových bodů v parametru **coord** jsou použity k tvorbě lomených čar tak, jak definuje parametr **coordIndex**. Ten obsahuje posloupnosti indexů vrcholů, každá posloupnost představuje jednu lomenou, tedy po částech navazující čáru.

TIP:

1. **Velikost bodů a tloušťku čar nelze stanovit. Tyto hodnoty bývají shodné s velikostí jednoho bodu na obrazovce. Při změně vzdálenosti u nich proto dochází k relativnímu zmenšení či zvětšení vůči okolním objektům – vzdálené množina bodů působí hustším dojmem, po přiblížení zřídne.**
2. **Čáry a body je třeba umisťovat těsně nad povrch jiných objektů (ploch), nikoliv přesně na ně. Vlivem zaokrouhlovacích chyb by při zobrazování mohlo dojít k jejich zakrytí.**

Množiny čar a bodů mohou dobře zastoupit textury. Nejsou pochopitelně vkládány do uzlu **Appearance**, nýbrž je s nimi zacházeno jako s geometrickými objekty. Příklad v programu P-3-16 ukazuje, jak za pomoci tří čar a šestnácti bodů vyrobit dopisní obálku na obrázku 3-16.

Body, které symbolizují místo pro známku, nemají v uzlu **PointSet** přiřazeny individuální barvy. Namísto toho je použita společná „luminiscenční“ červená barva v sourozeneckém uzlu **Appearance** u společného rodiče **Shape**.

Zelené čáry určující linky pro adresu jsou naopak obarveny barvou uvnitř uzlu **IndexedLineSet**.

Program P-3-16 je současně připraven k tomu, aby se daly namalovat tečkované rámečky pro případné směrovací číslo pomocí konstrukce **USE**. Tečkovaný obdélník je pojmenován slovem **RAMECEK** a doplněním několika uzlů **Transform** s mírným zmenšením a posunutím snadno připravíme prostor pro PSČ.

Obrázek 3-16: Tenký bílý kvádr je doplněn barevnými čárami a body

```
#VRML V2.0 utf8
Transform {
  children [
    Shape { geometry Box {size 12 8 0.1} }
    DEF RAMECEK Transform {
      translation 3.5 1.5 0.06
      children Shape { geometry
        PointSet { coord Coordinate {
              point [0 0 0,   0.3 0 0,   0.7 0 0,   1 0 0,
                     1 0.3 0, 1 0.6 0,   1 0.9 0,   1 1.2 0,
                     1 1.5 0, 0.3 1.5 0, 0.7 1.5 0, 0 1.5 0,
                     0 1.2 0, 0 0.9 0,   0 0.6 0,   0 0.3 0]
           }
        }
        appearance Appearance {material Material {emissiveColor 1 0 0}}
      }
    }
    Transform {               # vodorovné linky
      translation 0 -2 0.06
      children Shape { geometry
        IndexedLineSet { color Color {color 0 1 0.3}
          coord Coordinate {point [0 0 0, 5 0 0, 0 1 0, 5 1 0,
                                   0 2 0, 5 2 0]}
          coordIndex [0 1 -1, 2 3 -1, 4 5 -1]
          colorPerVertex FALSE
          colorIndex [0, 0, 0]
        }
      }
    }
  ]
}
```

Program P-3-16: Dopisní obálka doplněná množinou čar a bodů

TIP: **Body ani čáry nepodléhají kolizím s avatarem, pokrytí texturou a osvětlení ze zdrojů světla. Je vhodné je obarvovat materiálem, který má nastaven jediný nenulový parametr – `emissiveColor`.**

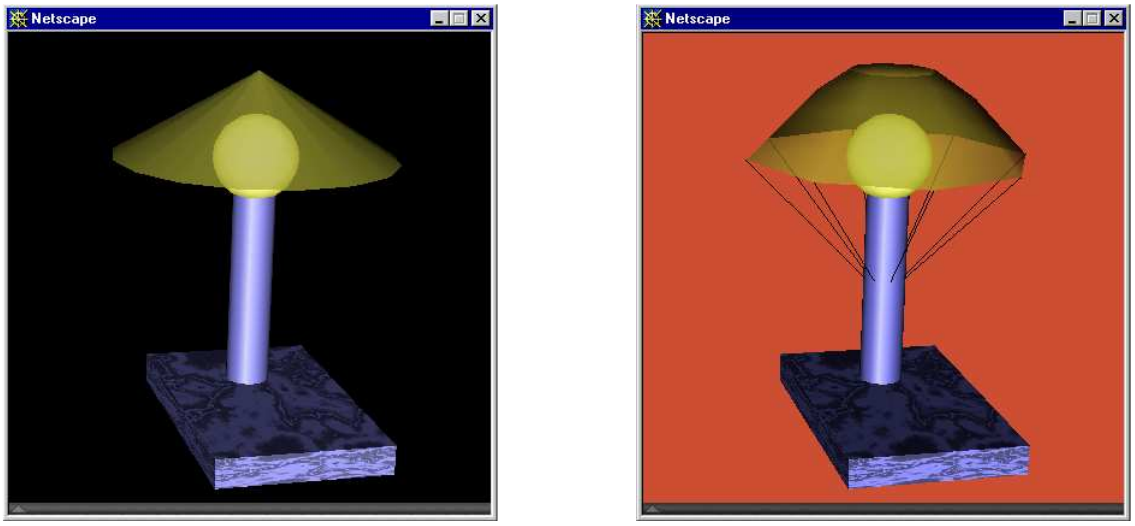
Krok za krokem III – Stínítko lampičky

Při postupném vylepšování vzhledu lampičky využijeme některých uzlů, se kterými jsme se seznámili v posledních kapitolách. Nahradíme stínítko lampičky, které bylo dosud vymodelováno kuželem, dokonalejším tvarem, vzniklým kombinací uzlů **`Extrusion`** a **`IndexedLineSet`**. Samotné stínítko vytvoříme opláštěním vytaženého kruhového profilu, navíc je doplníme drátěnými vzpěrami – množinou čar.

Na obrázku 3-17 je porovnání předchozího a nového tvaru lampičky, model stínítka je v programu P-3-17. Pozadí lampičky je obarveno na cihlově červenou pomocí uzlu **`Background`**, se kterým se seznámíme v kapitole 3.5.

```
#VRML V2.0 utf8
Transform {
  scale 0.1 0.1 0.1
  translation 0 0.18 0
  children [
    Shape {                              # stínítko
      appearance Appearance {
        material Material { diffuseColor 0.8 0.8 0.2
                            transparency 0.3 }
      }
      geometry    Extrusion {
        crossSection [ 1 0,  0.7  0.7, 0  1, -0.7 0.7,
                       -1 0, -0.7 -0.7, 0 -1,  0.7 -0.7, 1 0]
        spine [ 0 0.6 0,  0 0.4 0,  0 0.2 0,  0 0 0]
        scale [ 0.3 0.3,  0.6 0.6,  0.8 0.8,  1 1]
        creaseAngle 0.78
        solid  FALSE    endCap FALSE
      }
    }
    Shape {                              # drátky
      appearance Appearance {
        material Material {
          diffuseColor 0 0.8 1  specularColor    1 1 1
          shininess    0.2      ambientIntensity 1.0  }
      }
      geometry IndexedLineSet {
        coord  Coordinate {
          point [ 0 -1 0,     1 0 0,      0.7 0 0.7,
                  0 0 1,     -0.7 0 0.7, -1 0 0,
                 -0.7 0 -0.7, 0 0 -1,    0.7 0 -0.7]
        }
        coordIndex [ 1 0 2 -1,  3 0 4 -1,  5 0 6 -1,  7 0 8 -1]
      }
    }
  ]
}
```

Program P-3-17: Stínítko lampičky tvořené uzly **`Extrusion`** *a* **`IndexedLineSet`**



Obrázek 3-17: Vylepšení vzhledu lampičky změnou tvaru stínítka

3.4.5 Nápis

Ti, kteří očekávají, že do virtuálních světů budou moci snadno zabudovat rozličné prostorové, zprohýbané a jinak efektně upravené nápisy, budou zklamáni. Virtuální světy se snaží co nejvíce přiblížit světům skutečným, ve kterých jsou nápisy plošné, namalované většinou na rovinném podkladu. Takovým způsobem je zpracováván uzel **`Text`**, reprezentující plochý nápis umístěný do prostoru. Standardně je umísťován na pomyslnou rovinnou desku procházející počátkem souřadného systému, kolmou na osu *z*.

parametr	iniciální hodnota	Význam
`string`	**`[]`**	posloupnost textových řetězců
`fontStyle`	**`prázdný uzel`**	styl písma v uzlu **`FontStyle`**
`maxExtent`	**`0.0`**	nezáporná hodnota omezující maximální rozměr nápisu ve směru daném stylem písma; hodnota 0 ruší jakékoliv omezení
`length`	**`[]`**	seznam nezáporných délek pro každý z řetězců; hodnota 0 znamená libovolnou délku

*Tabulka T-3-13: Parametry uzlu **`Text`***

Jeho parametry jsou uvedeny v tabulce T-3-13. Texty, které mají být zobrazovány, se zapisují do parametru string. V něm může být několik textových řetězců (vět), každý z nich je vykreslen na novém řádku. Dále lze určit šířku, kterou nesmí text přesáhnout (**`maxExtent`**). Pokud *jediný* z textových řetězců tento rozměr přesáhne, jsou rovnoměrně stlačeny *všechny* nápisy v uzlu **`Text`**. Důmyslný parametr **`length`** pak dovoluje přiřadit každému z řetězců individuální délku, na kterou je řetězec roztažen či stlačen. Je-li počet délek v tomto parametru menší než počet řetězců, nadbytečné řetězce nejsou omezeny žádnými délkami.

TIP: Nápis je oboustranný, tj. pokud jej obejdeme, spatříme jej v téměř nečitelném zadním pohledu. Pro většinu aplikací je tento pohled nežádoucí a proto je nápis často umisťován na neprůhlednou podložku.

Je zřejmé, že parametry uzlu **`Text`** obsahují především základní údaje o obsahu textu a jeho vnějších rozměrech. Pro detailní určení stylu písma, ale i způsobu, jakým bude text nápisu vykreslován, slouží samostatný potomek – uzel **`FontStyle`**. Ten obsahuje celou řadu parametrů, jak ukazuje tabulka T-3-14.

parametr	iniciální hodnota	Význam
`size`	`1.0`	kladná výška písma
`spacing`	`1.0`	relativní vzdálenost mezi řádky
`family`	`"SERIF"`	seznam rodin písma
`style`	`"PLAIN"`	styl pro danou rodinu písma
`language`	`""`	dvojznaková zkratka použité abecedy
`horizontal`	`TRUE`	psaní ve vodorovném směru
`leftToRight`	`TRUE`	psaní na řádku zleva doprava
`topToBottom`	`TRUE`	psaní řádků shora dolů
`justify`	`"BEGIN"`	hlavní a vedlejší způsob umístění textu

Tabulka T-3-14: Parametry uzlu **`FontStyle`**

První dva parametry se týkají velikosti písma. Jsou udávány v metrech a určují jak výšku velkých písmen (**`size`**), tak relativní vzdálenost mezi řádky (**`spacing`**). Skutečně použitá vzdálenost mezi dvěma řádky se vypočítá jako součin **`spacing`** x **`size`**.

Další tři parametry uzlu **`FontStyle`** popisují tvary písmen tvořících nápis. Hlavní parametr **`family`** může obsahovat i několik jmen fontů, použit je pak první dostupný na daném počítači. Dva vedlejší parametry stanovují pro daný font jeho styl (**`style`**) a v případě použití národní abecedy jazyk[5], ve kterém je text kódován (**`language`**). Jazyk abecedy se zadává dvoupísmennou zkratkou, používanou v normě Unicode. Do zbylých parametrů se dosazují následující hodnoty:

font (**`family`**)

- **`"SERIF"`** pro patkové písmo (např. Times Roman),
- **`"SANS"`** pro bezpatkové písmo (např. Arial, Helvetica),
- **`"TYPEWRITER"`** pro písmo s pevnou šířkou znaků, jako je např. `Courier`,
- další libovolný název fontu tak, jak je pojmenován operačním systémem počítače.

styl (**`style`**)

- **`"PLAIN"`** – normální,
- **`"BOLD"`** – **polotučné**,
- **`"ITALIC"`** – *kurzíva*,
- **`"BOLDITALIC"`** – ***polotučná kurzíva***.

Jiné hodnoty pro styl, nežli výše definované čtyři, nelze použít. Naopak fontů můžeme zadat několik. Pokud na našem počítači není k dispozici ani jeden z požadovaných fontů, bude automaticky vybráno libovolné patkové písmo (**`"SERIF"`**).

Následující trojice logických parametrů určuje způsob kladení jednotlivých písmen a řádků nápisu. Mezi nimi je nejdůležitějším parametrem **`horizontal`**, který stanovuje tzv. *hlavní směr*, ve kterém je text vykreslován. Pokud je hlavní směr vodorovný, parametry **`leftToRight`** a **`topToBottom`** určují, zda písmena textového

[5] Vzhledem k tomu, že kódování UTF-8 (Unicode) není zatím příliš rozšířené, není vždy snadné zjistit, zda se nápisy se speciálními národními symboly nezobrazují správně z důvodu špatného kódování souboru, chybějícího fontu v systému nebo nefunkčnosti prohlížeče. V současnosti je tedy největší jistotou použití pouze základní sady znaků.

řetězce budou kladena zleva doprava či naopak a zda při výskytu více řádků se budou řádky vypisovat shora dolů či zdola nahoru. Je-li hlavní směr svislý, význam těchto dvou parametrů se vymění. Hodnota **`leftToRight`** říká, zda svislé sloupce textu budou umístěny zleva doprava, hodnota **`topToBottom`** specifikuje směr kladení písmen ve sloupcích.

Patrně nejsložitější postup se týká vykreslení nápisu vzhledem ke *vztažnému bodu*. Pomyslný vztažný bod je definován v počátku systému souřadnic, pokud nebyla jeho poloha změněna transformacemi v rodičovských uzlech. Mezi textovým řetězcem a vztažným bodem lze definovat mnoho různých vzájemných vztahů – tzv. *zarovnání*, a to vždy s ohledem na *hlavní směr* daný hodnotou parametru **`horizontal`**. Zarovnání textu vůči vztažnému bodu se definuje v parametru **`justify`**. Ten by měl obecně obsahovat dvojici hodnot, jednu pro hlavní směr, druhou pro vedlejší směr. Někdy je ovšem umístění v hlavním směru natolik jednoznačné, že hodnotu pro vedlejší směr není třeba zadávat a parametr **`justify`** obsahuje jen jedinou hodnotu.

Existují právě čtyři hodnoty, kterými se definuje zarovnání textu:

`"BEGIN"`, **`"FIRST"`**, **`"MIDDLE"`** a **`"END"`**.

Používají se jak pro hlavní, tak pro vedlejší směr. Jako příklad vybereme hlavní směr vodorovný. Tehdy první hodnota z dvojice **`justify`** říká, zda řádky nápisu budou podle souřadnice *x* vztažného bodu zarovnány doleva (**`"BEGIN"`** nebo **`"FIRST"`**), doprava (**`"END"`**) nebo budou vystředěny (**`"MIDDLE"`**). Hodnoty **`"BEGIN"`** a **`"FIRST"`** mají v tomto případě stejný význam. Druhá hodnota z dvojice **`justify`** se týká vedlejšího, v daném případě svislého směru, tj. umístění jednotlivých řádků. Je-li zadána hodnota **`"FIRST"`**, souřadnice *y* vztažného bodu je totožná s pomyslnou spodní linkou, na které leží první řádek nápisu. Při zadání **`"BEGIN"`** je tato souřadnice v místě horní linky, tj. vrcholků velkých písmen. Zbylé dvě hodnoty určují svislé vystředění řádků nápisu, resp. zarovnání řádek zdola.

Obdobně se s hodnotami parametru **`justify`** zachází, je-li hlavní směr svislý.

Ačkoliv lze pomocí parametrů pracujících se vztažným bodem definovat způsob vykreslování textu vysloveně „na přání“, z praktického hlediska je široká a místy nepřehledná škála možností spíše na závadu – většinou je nejsnazší zvolit předdefinované hodnoty a polohu textu doladit transformacemi v rodičovském uzlu.

TIP:

1. **Natočení roviny nesoucí nápis je zajištěno rodičovským uzlem `Transform`. Ten současně určuje nejen umístění vztažného bodu textu, ale mění i měřítka rozměrových parametrů uzlů `Text` a `FontStyle`.**
2. **Na nápis lze nanášet texturu, lze definovat jeho průhlednost, barvu, odraz světla. Uzly typu `Text` však nepodléhají kolizím s avatarem.**

V programu P-3-18 jsou ukázána dvě použití uzlu **`Text`**. První nápis je vykreslován co nejjednodušeji podle standardních hodnot všech parametrů. Druhý nápis obsahuje uzel **`FontStyle`**, s jehož pomocí jsou vykresleny dva sloupce řeckých písmen. Všimněme si, že tehdy je v parametru **`string`** uveden seznam textových řetězců. Písmo **`"Symbol"`** převádí v prostředí MS Windows řetězec `"abgd"` na řadu řeckých písmen "αβγδ", podobně řetěz `"SWPF"` na řadu "ΣΩΠΦ". Sklon písma je zajištěn použitím **`"ITALIC"`** v parametru **`style`**. Nastavením parametru **`horizontal`** je docílen směr psaní shora dolů, hodnota v parametru **`spacing`** v takovém případě určuje šířku mezery mezi sloupci.

```
#VRML V2.0 utf8
Transform {children [
  Transform {
    translation -1 1 0
    children Shape {
      appearance DEF MODRA Appearance {
        material Material {
          diffuseColor 0.2 1 1}
        }
      geometry Text {
        string "Jednoduchy text" }
    }
  }
  Transform {
    children Shape {
      geometry Text {
        string ["abgd", "SWPF"]
        fontStyle FontStyle {
          family "Symbol"  style "ITALIC"
          horizontal FALSE spacing 4
        }
      }
      appearance USE MODRA
    }
  }
]}
```

Program P-3-18: Obyčejný nápis a dvousloupcový text v řečtině[6] *specifikovaný uzlem* **`FontStyle`**

TIP: **Používejme uzel `Text` co nejméně. V prohlížeči je každé písmenko převáděno na mnoho trojúhelníků, což výrazně zpomaluje zobrazování. Často je lepší místo uzlu `Text` použít obrazovou texturu, byť její vzhled není tak pěkný.**

[6] Pozn.: uvedený obrázek byl vytvořen prohlížečem WorldView, neboť použitá verze prohlížeče CosmoPlayer chybně ignorovala parametr **`family`** při současném zadání parametru **`style`**.

3.5 Jak vykouzlit iluzi prostoru (Background, DirectionalLight, Fog, PointLight, SpotLight + AudioClip, Sound)

Trojrozměrný prostor vytvořený v paměti počítače by byl jen směsicí nevýrazných objektů, kdybychom do něj nevložili světlo a neumožnili povrchu objektů reagovat s ním, tj. odrážet je. V reálném světě je existence světla základní podmínkou našeho vidění a nejinak je tomu ve virtuální realitě. Světelné paprsky umocňují dojem prostoru, který je nám prezentován na ploché obrazovce. Základní tělesa a objekty, jejichž povrch je definován ploškami s normálovými vektory ve vrcholech, mění odstín barvy podle směru dopadajících paprsků, takže povrch vypadá plasticky a přesvědčivě.

Ve virtuálním prostoru existují čtyři druhy světelných zdrojů, tedy objektů, ze kterých se šíří světelné paprsky:

1. čelní svítilna avatara,
2. zdroj rovnoběžných paprsků (**`DirectionalLight`**),
3. bodový zdroj (**`PointLight`**).
4. reflektor, směrový zdroj (**`SpotLight`**).

O avatarově čelní svítilně jsme mluvili v souvislosti s uzlem **`NavigationInfo`** v kapitole 2.3. Protože ve virtuálním prostoru se pohybuje právě jeden avatar, existuje právě jedna čelní svítilna. Zbylé tři zdroje světla se mohou vyskytovat v mnoha exemplářích. V dalším textu se budeme věnovat pouze těmto zdrojům.

TIP: Po umístění a nastavení zdrojů světla je vhodné vypnout avatarovu čelní svítilnu. Její směrový svit potlačuje optické jevy vzniklé použitím individuálních světelných zdrojů.

Každý uzel, popisující světelný zdroj, má několik parametrů, určujících vlastnosti světelných paprsků (viz tabulka T-3-15). Paprsky mají vlastní barvu (**`color`**) o dané intenzitě (**`intensity`**). Světlený zdroj může být vypnut či zapnut (**`on`**) a lze stanovit, jak mnoho přispívá k celkovému projasnění virtuálního světa (**`ambientIntensity`**), tj. k množství nepřímého všesměrového světla – světelného šumu. Čím jasnější je svět, tím lépe vidíme plochy těles, které jsou odvrácené od zdrojů světla. Velké množství nepřímého světla však opticky potlačuje hloubku těles a prostoru.

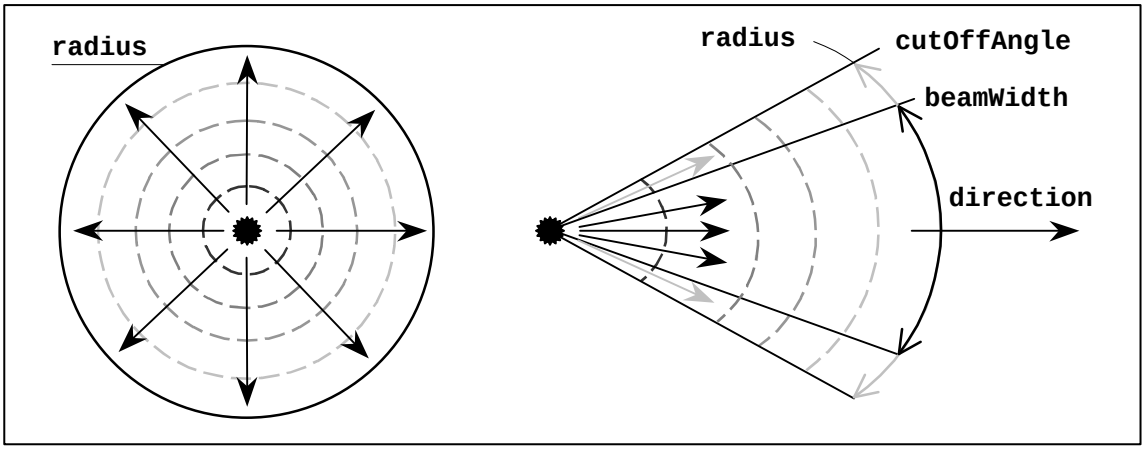
parametr	iniciální hodnota	Význam
`color`	**`1 1 1`**	Barva paprsků vyjádřená v RGB
`intensity`	**`1`**	Iniciální intenzita paprsků v rozsahu [0, 1]
`ambientIntensity`	**`0`**	Příspěvek zdroje k nepřímému osvětlení světa v rozsahu [0, 1]
`on`	**`TRUE`**	Zapnutí či vypnutí světelného zdroje

Tabulka T-3-15: Společné parametry světelných zdrojů

Další parametry se liší podle konkrétního typu zdroje světla. Nejjednodušší je uzel **`DirectionalLight`**, zbylé dva uzly mají složitější vlastnosti:

Zdroj rovnoběžných paprsků

- Má jediný parametr **`direction`**, který obsahuje vektor směru paprsků. Iniciální hodnota (**`0 0 -1`**) definuje paprsky ve směru záporné poloosy *z*. Paprsky nemají určen počátek, přicházejí jakoby z nekonečna.
- Osvětluje pouze sourozenecké objekty, tedy VRML stromy, které mají stejný rodičovský uzel jako tento zdroj světla. Záleží tedy na tom, do které části stromu uzel umístíme.
- Intenzita paprsků je konstantní, není ovlivňována žádnou vzdáleností.



Obrázek 3-19: Bodový zdroj (vlevo) a reflektor (vpravo) vysílají paprsky, jejichž intenzita klesá se vzdáleností.

Bodový zdroj a reflektor

- Mají definováno umístění v prostoru (parametr **`location`**), reflektor má navíc určenu osu světelného kužele (parametr **`direction`**).
- Osvětlují celý svět, avšak pouze ve vymezeném kulovém okolí – dosahu světla (parametr **`radius`**), jak symbolicky ukazuje obrázek 3-19. Na rozdíl od zdroje rovnoběžných paprsků nejsou omezeny na osvětlování sourozeneckých uzlů.
- Intenzita paprsků klesá se vzdáleností od zdroje až po nulovou hodnotu na okraji kulové hranice. Útlum je vyhodnocován výpočtem, ve kterém se použijí tři koeficienty, zapsané v parametru **`attenuation`**. Označíme-li tyto koeficienty jako $[a_0, a_1, a_2]$, pak je útlum světelného paprsku ve vzdálenosti d od zdroje světla vypočítán podle vzorce

  $$\frac{1}{a_0 + a_1 d + a_2 d^2}.$$

  Je-li výsledek větší než jedna, útlum je nastaven na jedna. Aby nedocházelo k dělení nulou, je trojice koeficientů [0, 0, 0] vždy chápána jako [1, 0, 0].

Společné parametry uzlů **`PointLight`** a **`SpotLight`**		
	iniciální hodnota	význam
`location`	**`0 0 0`**	umístění zdroje světla
`radius`	**`100`**	nezáporný dosah osvětlení
`attenuation`	**`1 0 0`**	trojice nezáporných koeficientů pro výpočet útlumu světla
Speciální parametry pouze pro uzel **`SpotLight`**		
`direction`	**`0 0 -1`**	směr osy světelného kužele
`beamWidth`	**`1.570796`**	kladný úhel (v rozsahu do π/2) mezi osou a površkou vnitřního kužele, v němž je plný světelný tok
`cutOffAngle`	**`0.785398`**	kladný úhel (v rozsahu do π/2) mezi osou a površkou vnějšího kužele, v němž klesá světelný tok

Tabulka T-3-16: Další parametry bodových a směrových zdrojů světla

- Reflektor si lze představit jako dvojici souosých kuželů se společným vrcholem (obrázek 3-19 vpravo). Tok paprsků ve vnitřním kuželu (určeném parametrem **`beamWidth`**) je homogenní, jejich intenzita klesá pouze s přímou vzdáleností od zdroje. Ve vnějším kuželu (určeném parametrem **`cutOffAngle`**) paprsky navíc „řídnou“ a jejich intenzita se snižuje i s rostoucím úhlem od osy kužele. Je-li vnější kužel menší než vnitřní, reflektor generuje ostře ohraničený svazek paprsků v prostoru menšího z kuželů.

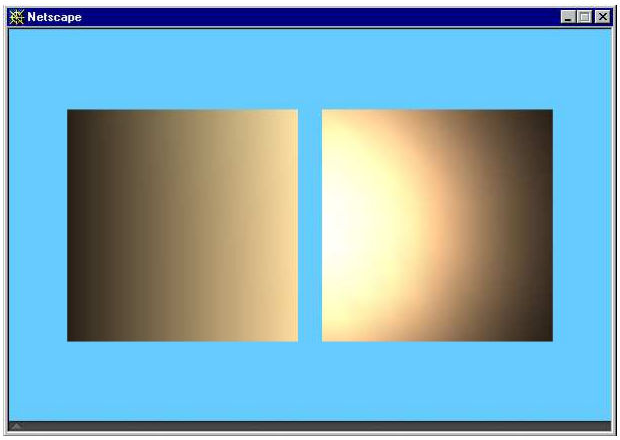
3.5.1 Není světlo jako světlo

Všechny zdroje světla jsou vždy jen *abstraktními* náhražkami skutečných světlených zdrojů. To mimo jiné znamená, že nemají vlastní geometrický tvar. Ten, kdo by očekával na místě bodového zdroje světla žárovku nebo alespoň malou jiskřičku, bude zklamán. Pokud jeho pohled při procházení umělým světem padne na místo, odkud jsou vysílány světelné paprsky, neuvidí žádné sluníčko, žádnou baterku. Je na tvůrci virtuálního světa, zda se rozhodne vymodelovat geometrii světelných zdrojů a umístit ji na patřičné místo.

Další důležitou a současně nemilou vlastností světelných paprsků je jejich schopnost procházet objekty. Žádný objekt totiž nemůže zastavit světelný paprsek na jeho dráze a zaclonit tak objekt jiný. Důsledkem tohoto chování je to, že tělesa *nevrhají stíny*. I malý objekt, který je zcela ukryt před světlem a schován za větší překážky, je jasně osvětlen ze strany, odkud přicházejí paprsky světla. Tvůrci virtuálních světů by na toto chování měli pamatovat a snažit se nastavit dosah světel tak, aby pokud možno co nejvíce odpovídal skutečnému šíření světla. Ti, kteří nutně potřebují zobrazit vržené stíny, je mohou vymodelovat pomocí poloprůhledných plošek (**`IndexedFaceSet`**) umístěných na povrchu ostatních těles. Jde přitom o poněkud náročný postup.

Co je však pro někoho nevýhodou, to jiný ocení. Skutečnost, že jeden objekt nemůže zaclonit druhý před světlem a způsobit tak vznik stínu, se zdá být splněným snem každého fotografa a divadelního či filmového osvětlovače. Chceme-li ve virtuálním prostředí zajistit kvalitní osvětlení pro množství různých objektů, nemusíme se bát, že si objekty budou navzájem překážet. Jeden vhodně umístěný zdroj světla je osvítí se stejnou kvalitou.

Chování světla ve virtuálním prostředí se od chování skutečného světla liší v další zásadní věci. Ta se tentokrát váže na vyhodnocení barevného odstínu povrchu podle *normál*. Jak již víme z kapitoly věnované tělesům pokrytým ploškami, barevný odstín se vypočítá pro každý vrchol podle orientace normály v tomto vrcholu, směru pohledu avatara a směru dopadajících paprsků. Vnitřní body ploch získají barevné odstíny, které plynule přecházejí mezi hodnotami barev v jednotlivých *vrcholech*. Protože vliv světla není pro individuální vnitřní body vůbec vyhodnocován, může se například stát, že světelný flíček, který očekáváme jako důsledek osvětlení plochy reflektorem, se na plošce vůbec neobjeví.



Obrázek 3-20: Dvě na první pohled stejné desky jsou osvětlovány jedním reflektorem. Přesvědčivý výsledek vzniká pouze na pravé desce, která je ve skutečnosti výškovou mapou s hustotou sítě 11 x 11 bodů.

Prakticky tento problém ilustruje obrázek 3-20. Na něm vidíme vlevo čtverec omezený čtyřmi vrcholy, vpravo je pak stejně velká síť (**`ElevationGrid`**), na níž je ovšem 11 x 11 vrcholů. Směrový zdroj světla (reflektor) svítí právě doprostřed mezi oba čtverce. Jeho vnější kužel paprsků má dvojnásobný rozměr oproti vnitřnímu kuželu. Očekávaný výsledek spatříme vpravo, kde vidíme jak téměř zcela bílou oblast plného osvětlení, tak oblast lineárního útlumu o dvojnásobném poloměru. Nedostatečná hustota vrcholů sítě způsobuje vznik nepřirozeně zubaté hranice mezi světlem a stínem. Úplně jiným dojmem působí levý čtverec, na němž pouze lineárně[7] přechází zprava doleva světlý odstín barvy až do černé. Všimněme si přitom, že navzájem si odpovídající rohy levého a pravého čtverce mají shodné barvy.

[7] Pokud má čtenář dojem, že i na čtverci vlevo vidí náznak světlejšího flíčku, jde o optický klam vyvolaný blízkostí čtverce vpravo.

Vidíme tedy, že světelné zdroje jsou ve virtuálních světech jen velmi přibližným vyjádřením skutečných světel a že jsou s nimi spojeny nečekané efekty. Na vysvětlenou uveďme, že většina nežádoucích vlastností virtuálních zdrojů světla má kořeny v nízké rychlosti počítačů, které osvětlení vyhodnocují. Metody, které zvládnou vypočítat osvětlení dokonaleji, jsou známy, ale jejich složitost nedovolí použití při vykreslování v reálném čase.

Na závěr části věnované zdrojům světla ještě zdůrazníme pravidlo, které má sice obecnou platnost, ale v oblasti světel platí dvojnásob:

Základní stavebnicové pravidlo

Každý virtuální svět má být konstruován tak, aby jeho případné vložení do dalšího světa neovlivnilo vlastnosti nadřazeného světa.

Pro světelné zdroje to znamená, že by neměly působit vně daného světa. Dosahy u světel typu **`PointLight`** a **`SpotLight`** by měly být stanoveny tak, aby nepřesáhly rozměry aktuálního světa. Stejně tak by neměly být umístěny zdroje rovnoběžných paprsků (**`DirectionalLight`**) na nejvyšší úroveň v souboru. Při vložení takto uspořádaného světa do rozsáhlejšího virtuálního prostředí by totiž mohly začít osvětlovat další objekty.

TIP:

1. **Geometrické vlastnosti zdrojů světla (umístění, směr, dosah) jsou ovlivňovány transformacemi v rodičovských uzlech.**
2. **Na interakci povrchu s konkrétním světlem mají největší vliv parametry `diffuseColor`, `specularColor` a `shininess` uzlu `Material`. Celkové projasnění světa se kombinuje s parametrem `ambientIntensity`.**

Milovníci matematických vzorců a přesných vyjádření naleznou úplný popis výpočtu barvy a osvětlení bodu hned za následující podkapitolou.

3.5.2 Staráme se o (životní) prostředí?

Lidé na Zemi dvacátého století nežijí v dokonale čistém prostředí, v krajině s dalekou viditelností a jasnými barvami. Tento neradostný fakt ovlivnil i tvůrce jazyka VRML, kteří zavedli uzel zaměřený na popis mlhy, kouře, zkrátka takového okolního prostředí, které opticky ovlivňuje celkový obraz pozorovaného virtuálního světa. Jmenuje se **`Fog`** a má pouhé tři parametry – barvu mlhy (**`color`**), způsob, jakým mlha houstne (**`fogType`**) a vzdálenost od avatara, za kterou jsou již všechny vzdálenější objekty zcela zakryty mlhou (**`visibilityRange`**).

parametr	iniciální hodnota	význam
`color`	**`1 1 1`**	barva mlhy ve složkách RGB
`fogType`	**`"LINEAR"`**	způsob houstnutí mlhy (**`"LINEAR"`** nebo **`"EXPONENTIAL"`**)
`visibilityRange`	**`0`**	nezáporná vzdálenost maximálního „barevného“ dohledu; hodnota 0 zcela odstraňuje mlhu

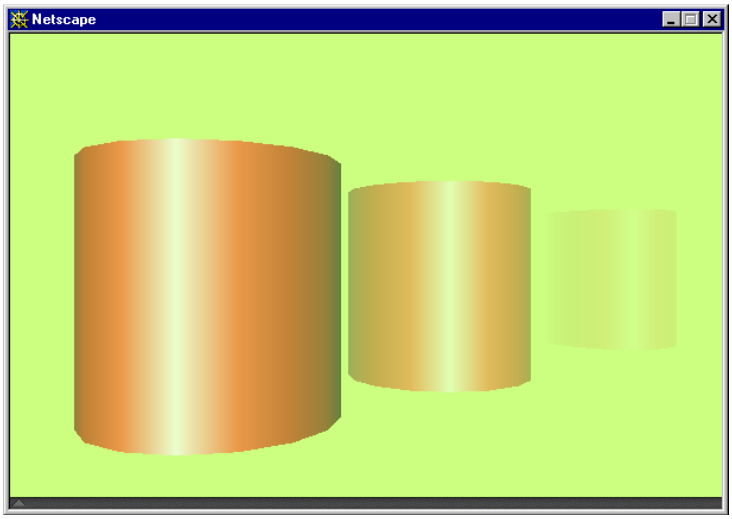
Tabulka T-3-17: Parametry uzlu **`Fog`**

Jak se projeví to, že objekty jsou v mlze? Tím, že jejich barva je smíchána s barvou mlhy. Jde vlastně o podobný jev, s jakým jsme se setkali u poloprůhledných textur. V případě mlhy se koeficient průhlednosti objektů mění dynamicky, s ohledem na aktuální vzdálenost avatara od objektů. Nejde přesně o průhlednost, jako spíše o koeficient určující, jak velké množství původní barvy si objekt uchová a jak velké množství bude nahrazeno barvou mlhy. V těsné blízkosti avatara je koeficient nulový, objekty mají pouze svoji barvu. Dále od avatara hodnota koeficientu roste, až konečně za danou mezí je jednotková – objekt je vykreslen pouze barvou mlhy. Podle hodnoty parametru **`fogType`** se koeficient mění lineárně nebo exponenciálně. Druhý způsob je více „agresivní“ – mlha houstne rychleji se vzdáleností.

TIP:

1. **Použití mlhy může urychlit výpočty, protože u vzdálených objektů není potom třeba vyhodnocovat barvy a textury.**
2. **Ačkoliv lze v jednom souboru definovat více uzlů `Fog`, vliv na zobrazení světa má vždy pouze jediný z nich.**

Program P-3-19 a jemu odpovídající obrázek 3-21 ukazují vzhled tří červených válců umístěných do prostředí se žlutozelenou mlhou. Příklad obsahuje i uzel **Background**, o kterém budeme mluvit vzápětí. Je vhodné, aby barva pozadí a barva mlhy byly totožné. Při normálním černém pozadí by totiž objekty nemizely v mlze, ale naopak by získávaly žlutozelenou barvu, jasně zářící na černém okolí.



Obrázek 3-21: Tři tělesa postupně se ztrácející v mlze

```
#VRML V2.0 utf8
Fog { color 0.8 1 0.5      visibilityRange 10 }

Background { skyColor 0.8 1 0.5}

DEF VALEC Transform {
  children Shape {
    appearance Appearance {
      material Material { diffuseColor  1 0.2 0  specularColor 1 1 1 }
    }
    geometry Cylinder {}
  }
}

Transform { translation 2 0 -2   children USE VALEC }

Transform { translation 5 0 –5   children USE VALEC }
```

Program P-3-19: Tři červené válce umístěné ve žlutozelené mlze

Zatímco mlha svým způsobem kazí vzhled virtuálního světa, další uzel, se kterým se seznámíme, dokáže obraz světa výrazným způsobem zlepšit a dodat mu skutečnou prostorovou hloubku. Takovým „zázračným“ uzlem je **Background**. Lze jej použít několika způsoby – od jednoduché výplně pozadí jedinou barvou (jako tomu bylo v některých předchozích obrázcích), přes škálu barev měnících se průběžně s výškou nad horizontem až po panoramatické obrázky obklopující ze všech stran virtuální svět. Parametry jsou uvedeny v tabulce T-3-18.

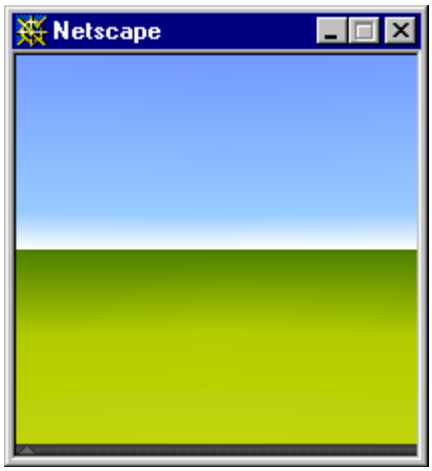
Nejprve si řekneme, že vše, co je definováno jako pozadí, je vykreslováno bez ohledu na světelné zdroje, mlhu či dohled avatara. Pozadí je prostě zcela samostatný prvek, který je do okénka prohlížeče nanesen jako prvý a teprve poté jsou před něj umisťovány virtuální objekty.

parametr	iniciální hodnota	význam
`skyColor`	`0 0 0`	seznam barev pro postupné přechody na sférické obloze
`skyAngle`	`[]`	rostoucí posloupnost úhlů pro barvy oblohy
`groundColor`	`[]`	seznam barev pro postupné přechody na sférické zemi (podlaze)
`groundAngle`	`[]`	rostoucí posloupnost úhlů pro barvy země
`frontUrl, leftUrl, rightUrl, backUrl, topUrl, bottomUrl`	`[]`	seznamy adres s umístěním obrázků pro jednotlivé stěny obklopující krychle

Tabulka T-3-18: Parametry pozadí

Z geometrického hlediska můžeme na pozadí pohlížet jako na těleso, obklopující celý virtuální svět. Avatar se vždy nachází uvnitř tohoto tělesa a nikdy nedojde k jeho hranicím. Je zajímavé, že uzel **`Background`** definuje hned dvě takováto obklopující tělesa – kouli a krychli. Každé z nich má jiné použití. Koule se používá pro jednodušší pozadí, na kterém mohou pouze přecházet barevné odstíny ve vodorovných pruzích. Krychle pak slouží jako podklad pro panoramatické obrázky, obklopující svět ze šesti stran. Jde vlastně o mapování obrazové textury na vnitřní stěny krychle. Jako zdroje obrázků můžeme použít stejné formáty souborů jako v případě uzlu **`ImageTexture`**.

```
#VRML V2.0 utf8

Background {
  skyColor    [ 0   0   1, # modrá
                0.6 0.8 1, # světle modrá
                1   1   1] # bílá
  skyAngle    [1.5, 1.57]
  groundColor [1   1 0.3,  # žlutá
               0.7 0.8 0,  # zelenkavá
               0.3 0.5 0]  # temně zelená
  groundAngle [1.4, 1.57]
}
```

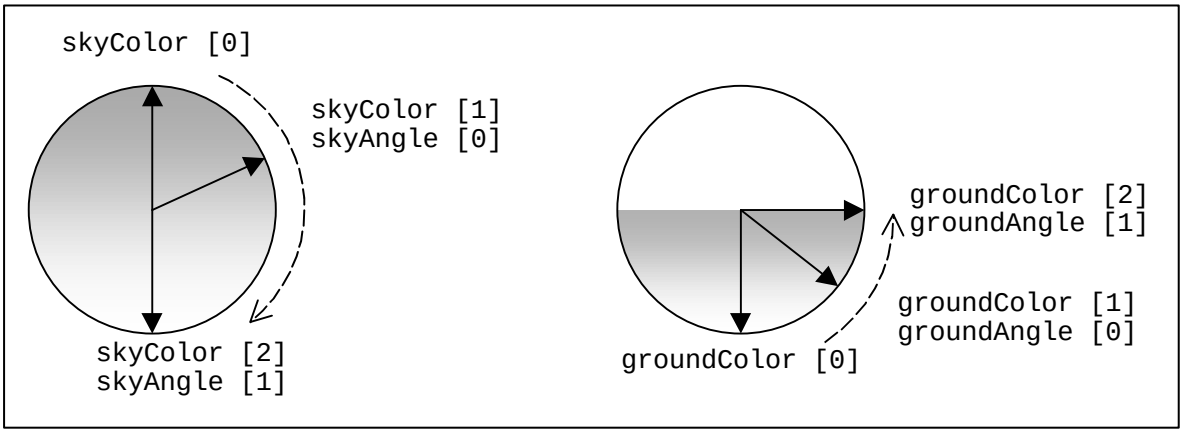


Program P-3-20: Pozadí v podobě koule dovoluje definovat pouze vodorovné barevné přechody

Pozadí v podobě koule

U koule rozlišujeme oblohu (*sky*) a zemi (*ground*). Obloha může pokrývat celou kouli, což je výhodné tehdy, pokud virtuální svět představuje vesmírný prostor. Je-li navíc definována země, její barva překryje v dolní polokouli barvu oblohy.

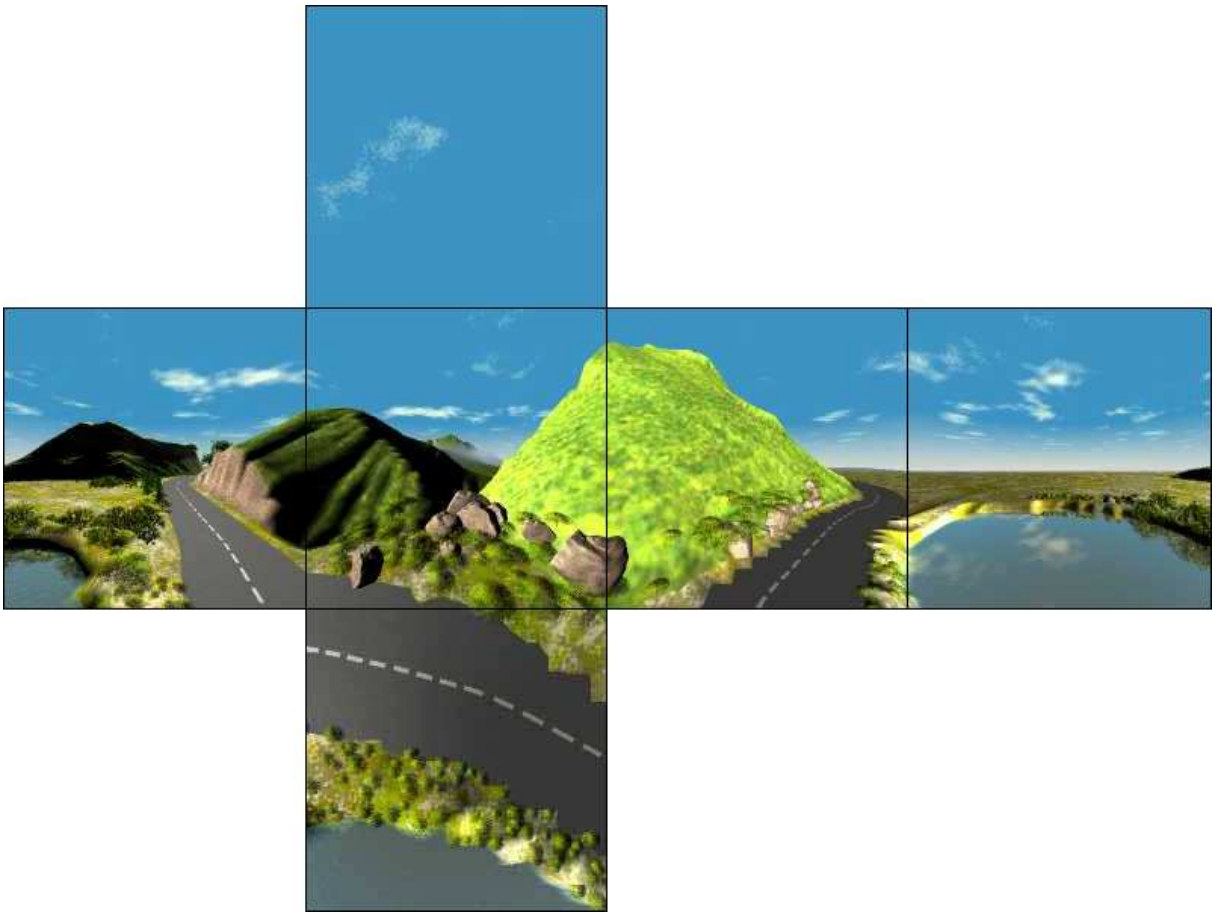
Parametr **`skyColor`** obsahuje posloupnost barev pro oblohu. Každé této barvě je přiřazen úhel v parametru **`skyAngle`**. Úhly se mění v intervalu od 0 do π a jsou odměřovány od svislé spojnice severního a jižního pólu za situace, kdy avatar stojí uprostřed kulového prostoru. Nulový úhel určuje barvu, kterou vidí nad sebou přímo v nadhlavníku (*zenitu*), nebo jinak řečeno na severním pólu obklopující koule. Pravý úhel znamená barvu na horizontu neboli na rovníku koule a konečně úhel o hodnotě π říká, která barva bude použita, dívá-li se avatar přímo dolů do podnožníku (*nadiru*), tj. na jižní pól obklopující koule. Prostor na pozadí mezi takto definovanými barvami a úhly je vyplněn plynulými barevnými přechody, jak je vidět na obrázku příslušejícím programu P-3-20. Počet barev musí být vždy o jednu vyšší, než počet úhlů, neboť první barva ze seznamu se použije pro místo, odkud jsou úhly odměřovány, tedy pro nadhlavník.



Obrázek 3-22: Zadávání posloupností úhlů a barev pro oblohu a pro zemi

Podobně jako s oblohou se pracuje i se zemí. Jedinou změnou je menší rozsah úhlů v parametru **groundAngle**. Úhly jsou odměřovány zdola od podnožníku k horizontu, tedy od nuly do pravého úhlu. Obdobně je nutno zadat o jednu barvu více (**groundColor**) vzhledem k použití první barvy jako barvy podnožníku. Vazby mezi úhly a barvami vysvětluje obrázek 3-22.

V iniciálním nastavení parametrů uzlu **Background** není definována žádná barva země a obloha je v celém rozsahu pokryta jedinou barvou – černou.



Obrázek 3-23: Šest obrázků pro panoramatické pozadí ve tvaru krychle

Pozadí v podobě krychle

Nejlepšího vzhledu pozadí docílíme při použití panoramatických obrázků nalepených na stěny pomyslné obklopující krychle. Obrázek 3-23 ukazuje, že příprava vhodných obrázků není snadná, protože musíme zaručit plynulou návaznost obrazů na sousedních stěnách. Je lépe použít specializovaný program pro modelování krajiny, který je schopen potřebných šest pohledů zapsat do souborů. Jména souborů se zapíší do parametrů `frontUrl` pro přední pohled, `leftUrl` pro pohled doleva, atd.

Ukázka použití pozadí v podobě krychle je v programu P-3-21.

Kombinace krychle a koule

Jakkoliv by se zdálo, že dvě pomocná tělesa, koule a krychle, se navzájem funkčně vylučují, opak je pravdou. Krychle je totiž vepsána do koule, která, ač bez jasně určeného rozměru, krychli zcela obklopuje. To pro praktické použití znamená, že můžeme vynechat například obrázek pro horní a dolní stěnu obklopující krychle a návštěvník při pohledu vzhůru, resp. dolů spatří kulový vrchlík s barevnými přechody tak, jak jsou definovány pro pozadí na kouli.

Další variantou je použití obrázků, které jsou poloprůhledné a prosvítají jimi barvy na kouli v pozadí. Snadno tak zkombinujeme například fotografické záběry města mapované na krychli (panorama Hradčan) s barevnými přechody na obloze. Jak se totiž dozvíme později, barvy na pozadí se dají dynamicky měnit a lze vytvářet iluzi zapadajícího či vycházejícího slunce. Jiným využitím kombinace krychle a koule na pozadí je zprůhlednění oken velkých místností tak, aby jimi bylo vidět na oblohu v pozadí.

```
#VRML V2.0 utf8

Background {

  frontUrl  "bg_ft.jpg"
  backUrl   "bg_bk.jpg"
  leftUrl   "bg_lf.jpg"
  rightUrl  "bg_rt.jpg"
  topUrl    "bg_up.jpg"
  bottomUrl "bg_dn.jpg"

}
```

Program P-3-21: Pozadí ze šesti obrazů zajišťuje úplný panoramatický pohled

TIP:

1. **Ačkoliv lze v jednom souboru definovat více uzlů `Background`, při prohlížení světa je použit vždy pouze jediný z nich.**
2. **Při současném použití pozadí a mlhy je vhodné barvu pozadí nastavit v oblasti horizontu na barvu mlhy. Další barva nad horizontem pak jakoby vystupuje z mlhy válející se při zemi.**

Krok za krokem IV – Lampička svítí

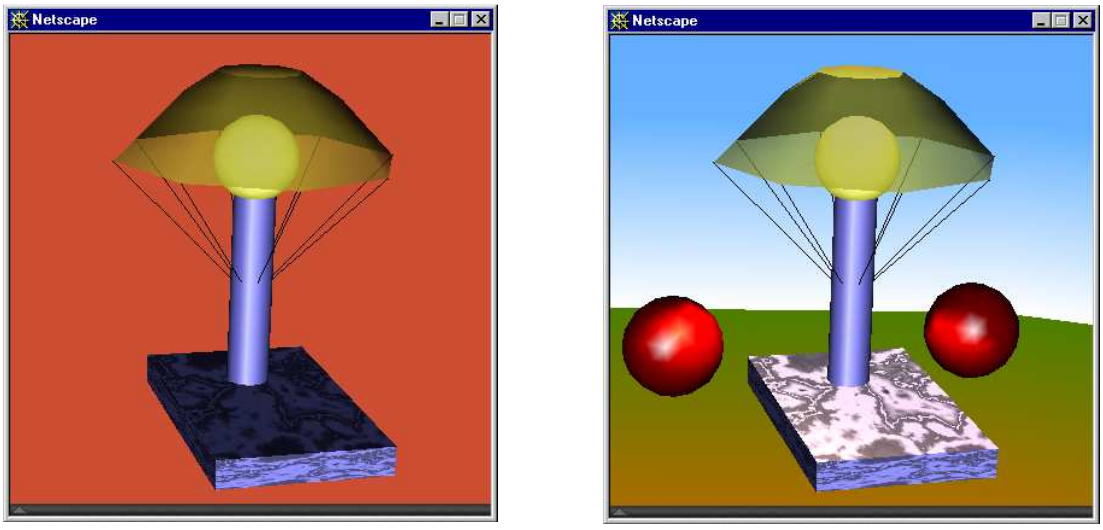
Co bychom to byli za tvůrce, kdybychom nedoplnili naši lampičku zdrojem světla? Vzhledem k tomu, že lampička má shora stínítko, nejvhodnějším zdrojem je reflektor svítící shora dolů. V programu P-3-22 je ukázáno správné nastavení polohy a směru tohoto světelného zdroje. Lampička bude vyzařovat žluté světlo (parametr **`color`**) a to až do vzdálenosti 50 cm (**`radius`**). Nízké koeficienty útlumu (**`attenuation`**) zajistí dostatečně silný světelný tok v celé oblasti dosahu světla.

```
#VRML V2.0 utf8

Background {
  skyColor    [0.2 0.3 1, 0.4 0.7 1, 1 1 1]   # modrá, světle modrá, bílá
  skyAngle    [1.75, 1.87]
  groundColor [1 1 0.3, 0.8 0.4 0, 0.3 0.5 0] # žlutá, rumělka, khaki
  groundAngle [1.1, 1.27]
}

SpotLight {    # žluté světlo
  color       1 0.8 0.5    radius      0.5
  location    0 0.3 0      direction   0 -1 0
  cutOffAngle 0.56         attenuation 0.5 0 0
}
```

Program P-3-22: Pozadí a zdroj světla doplňující model lampičky

Nejcitlivějším parametrem je úhel reflektoru (**`cutOffAngle`**). Na obrázku 3-23 můžeme ověřit, že jeho hodnota je správná. Světlé a tmavé plochy dvou červených kuliček jasně ukazují, kam směřuje svazek paprsků a jaká je jeho šíře. Všimněme si také bílých odlesků na červených kuličkách. Jsou to odrazy avatarovy (bodové) čelní svítilny. Na povrchu kuliček tak pozorujeme výsledky vlivu dvou zdrojů světla.

Lampička je tentokrát umístěna do prostředí s barevným pozadím, jak je také vidět z ukázky P-3-22. Porovnání s předchozí verzí modelu lampičky (obr. 3-24 vlevo) přesvědčivě dokazuje, že vhodné pozadí má výrazný estetický dopad. Stejně tak je vidět, jak přidání zdroje světla zvýšilo jas mramorového podstavce lampičky.



*Obrázek 3-24: Doplnění lampičky zdrojem směrového světla (**`SpotLight`**) a pozadím. Kuličky jsou přidány pro kontrolu, že zdroj světla má nastaveny správně své charakteristiky.*

3.5.3 Vzorec pro přemýšlivé

V předchozí kapitole jsme čtenářům se zájmem o matematické vzorce slíbili, že uvedeme způsob vyhodnocení barvy na povrchu objektů s ohledem na světelné a barevné prostředí virtuálního světa. Nyní je čas tento výpočet uvést, protože již známe všechny uzly, které mají na výslednou barvu vliv:

- zdroje světla (**`DirectionalLight`**, **`PointLight`**, **`SpotLight`**), avatarova svítilna v uzlu **`NavigationInfo`**,
- materiál (**`Material`**), textury (**`ImageTexture`**, **`MovieTexture`**, **`PixelTexture`**), barvy v uzlu **`Color`**,
- normály ve vrcholech (**`Normal`**),
- mlha (**`Fog`**).

Protože se barva zadává pomocí tří složek R, G a B, je ve vzorci namísto znaménka obyčejného násobení použit symbol ×, který říká, že násobení je prováděno po jednotlivých složkách. Tedy červené složky spolu, obdobně zelené a modré složky.

Jiným způsobem se ovšem násobí geometrické vektory. Ty se ve vzorci objevují ve skalárním součinu, který je zároveň roven cosinu úhlu svíraného těmito vektory. Skalární součin budeme značit obyčejnou tečkou. Všechny vektory použité ve výpočtu musejí mít délku jedna.

Zavedeme následující označení:

$\mathbf{N}$	normálový vektor plochy ve vyhodnocovaném bodě
$\mathbf{V}$	vektor z vyhodnocovaného bodu směrem k avatarovi
$\mathbf{L}$	vektor z vyhodnocovaného bodu směrem k danému zdroji světla (směr proti paprskům)
f_0	koeficient působení mlhy na barvu objektu (hodnota 1 znamená, že mlha nepůsobí)
$d_{\mathbf{V}}$	vzdálenost vyhodnocovaného bodu od avatara
$d_{\mathbf{L}}$	vzdálenost vyhodnocovaného bodu od daného bodového zdroje světla nebo reflektoru

Koeficient působení mlhy f_0 se vypočítá uvnitř rozsahu **`visibilityRange`** jako

$\frac{\text{visibilityRange} - d_{\mathbf{V}}}{\text{visibilityRange}}$ pro lineární působení mlhy,

$e^{\frac{-d_{\mathbf{V}}}{\text{visibility Range}-d_{\mathbf{V}}}}$ pro exponenciální působení mlhy.

Mimo rozsah nabývá hodnot 0 nebo 1.

Výpočet barevného odstínu určitého vrcholu tělesa se provádí pro všechny zapnuté světelné zdroje, jejichž poloha uvnitř VRML stromu a vzdálenost od tohoto vrcholu má nějaký vliv na osvětlení vrcholu. Pro taková světla se vypočítá barevný odstín jako součet celkového osvětlení, difúzního a zrcadlového odrazu světla na povrchu objektu. Hodnota **`diffuseColor`** se získá z materiálu, textury nebo je tvořena barvou z uzlu **`Color`**.

a_i, tj. celkové osvětlení $= \text{ambientIntensity}_{\text{Light}} \times \text{ambientIntensity}_{\text{Material}} \times \text{diffuseColor}$

d_i, tj. difúzní odraz $= \text{intensity}_{\text{Light}} \times \text{diffuseColor} \times (\mathbf{N} \cdot \mathbf{L})$

z_i, tj. zrcadlový odraz $= \text{intensity}_{\text{Light}} \times \text{specularColor}_{\text{Material}} \times \left(\mathbf{N} \cdot \frac{\mathbf{L}+\mathbf{V}}{|\mathbf{L}+\mathbf{V}|} \right)^{\text{shininess} \times 128}$

Útlum se vypočítává na základě trojice hodnot (c_1, c_2, c_3) v parametru **`attenuation`** a vzdálenosti $d_{\mathbf{L}}$ bodu od světla:

$$\text{útlum}_i = \frac{1}{\mathbf{max}(c_1 + c_2 d_{\mathbf{L}} + c_3 d_{\mathbf{L}}^2, 1)} \times \text{spot}_i$$

kde funkce **max** je zařazena proto, aby dělitel nebyl větší než jedna. Hodnota spot_i se týká pouze reflektorů, u nichž je kužel vnějšího útlumu větší než kužel plného svazku paprsků. V oblasti mezi těmito kužely platí vztah

$$\text{spot}_i = \frac{\text{cutOffAngle} - \text{spotAngle}}{\text{cutOffAngle} - \text{beamWidth}},$$

kde spotAngle je úhel mezi osou světelného kuželu a vektorem **L**.

Výsledná barva I ve vrcholu s normálovým vektorem **N** se určí podle vzorce, v němž se výraz uvnitř sumy vyhodnocuje pro každý zapnutý zdroj světla:

$$I = \mathbf{color}_{\text{Fog}} \times (1 - f_0) + f_0 \times \left[\sum \left(\text{útlum}_i \times \mathbf{color}_{\text{Light}} \times (a_i + d_i + z_i)\right) + \mathbf{emissiveColor}_{\text{Material}}\right]$$

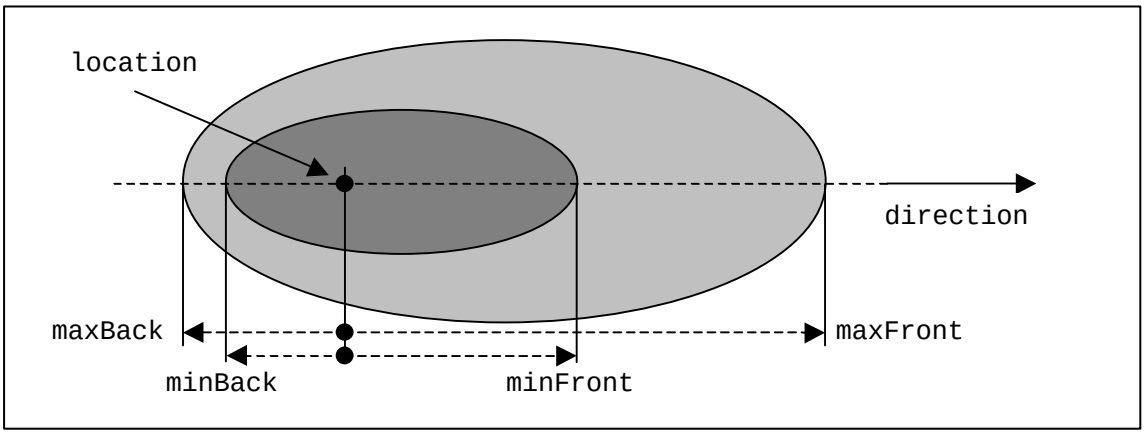
Důvod, proč jsme čtenáře obtěžovali vzorcem pro výpočet osvětlení jednoho bodu na povrchu prostorového objektu je jednoduchý. Nebylo tomu proto, že znalost vzorce považujeme za důležitou. Chtěli jsme však alespoň naznačit složitost výpočtů, které se skrývají za vytvořením barevného stínovaného obrázku. Snad bude nyní čtenář a návštěvník virtuálních světů shovívavější, když pohyb světem v reálném čase nebude při použití středně výkonného počítače zcela plynulý. Znalost výpočtu barevných odstínů současně přispěje k lepšímu porozumění významu parametrů ve zdrojích světla a v materiálu.

3.5.4 A nyní trochu hudby

Ke zvýšení dojmu reality přispívá značnou měrou i zvuk. Jednoduché zvuky ujišťují návštěvníka virtuálních světů o tom, že objekty se chovají stejně jako ve skutečnosti. Dveře při zavření hlasitě zaklapnou, zvonek při stisku zabzučí, sklenka při nárazu cinkne.

Další význam zvuků spočívá v tom, že usnadňují orientaci v trojrozměrném prostoru. Jen málokdo si uvědomuje, jak lidský mozek dokáže zužitkovat vzdálené a blízké zvuky k odhadnutí vzdálenosti a směru, a to podvědomě, bez jakékoliv cílené námahy. Zdroj zvuku ve virtuálním prostředí tak může působit jako maják. Je však třeba, aby zvuk měl prostorový charakter, tj.byl šířen nezávisle ze dvou reproduktorů. Tehdy je posluchač schopen určit směr, odkud zvuk přichází.

Do virtuálního prostředí můžeme umístit několik zdrojů zvuku. Umístění, směr a dosah šíření a další charakteristiky jednoho zdroje zvuku se definují v uzlu **Sound**. Ten má jediného potomka – uzel, který určuje soubor se zaznamenaným zvukem a časový průběh jeho přehrávání. Zvlášť jsou tedy popsány geometrické vlastnosti zvukového zdroje a zvlášť vlastní prezentace zvuku.



Obrázek 3-25: Geometrie šíření zvuku v prostoru.

Podobně jako světelné zdroje, také zdroj zvuku má definován dosah a útlum. K jejich popisu slouží dvojice prostorových elips (elipsoidů), se shodnou hlavní osou a jedním společným ohniskem, jak ukazuje obrázek 3-25. V prostoru menší, tedy vnitřní elipsy, se zvuk šíří bez jakéhokoliv útlumu. Má stále stejnou hlasitost. Teprve za hranicí vnitřní elipsy začíná slábnout, a to s lineárně odstupňovanou intenzitou (v dB). Za hranicí vnější elipsy není slyšet vůbec.

parametr	iniciální hodnota	význam
source	**žádný uzel**	uzel se zvukovým záznamem
location	**0 0 0**	umístění zdroje zvuku
direction	**0 0 1**	směr hlavní osy obou charakteristických prostorových elips
minBack	**1**	vzdálenost zadního vrcholu elipsy plné slyšitelnosti
minFront	**1**	vzdálenost předního vrcholu elipsy plné slyšitelnosti
maxBack	**10**	vzdálenost zadního vrcholu elipsy plného útlumu
maxFront	**10**	vzdálenost předního vrcholu elipsy plného útlumu
intensity	**1**	intenzita zvuku v rozsahu [0, 1]
spatialize	**TRUE**	povolení generování prostorového zvuku podle orientace avatara
priority	**0**	priorita mezi více zdroji zvuku v rozsahu [0, 1]

Tabulka T-3-19: Parametry uzlu **Sound**

K popisu geometrie obou elips slouží v uzlu **Sound** šest parametrů. Dva definují společné ohnisko obou elips (**location**) a směr hlavní osy (**direction**). V ohnisku leží současně zdroj zvuku. Vnitřní elipsa plné slyšitelnosti je dána vzdáleností předního a zadního vrcholu elipsy od zdroje zvuku (**minFront** a **minBack**), obdobně je zadána vnější elipsa, ve které dochází k postupnému útlumu (**maxFront**, **maxBack**). Jak je vidět z tabulky T-3-19, v iniciálním nastavení jsou obě prostorové elipsy totožné a jsou převedeny na koule, do jejichž středu je umístěn zdroj zvuku.

TIP:
1. **Geometrické charakteristiky zdroje zvuku jsou ovlivňovány transformacemi v rodičovských uzlech.**
2. **Správné nastavení velikosti charakteristických elips zajistí, aby zvuk z jedné virtuální místnosti nebyl slyšet v místnostech sousedních.**

Hlasitost, s jakou se zvuk ozývá v oblasti vnitřní elipsy, je zapsána v parametru **intensity**. Další parametr (**spatialize**) říká, zda při použití dvojice reproduktorů bude zvuk ovlivňován natočením avatara, tj. zda při otáčení avatara bude zvuk přecházet z jednoho reproduktoru do druhého.

Ve skutečném světě většinou slyšíme v každém okamžiku celou řadu zvuků. Některé splývají, jiné snadno rozlišujeme. Uzlů typu **Sound** proto můžeme definovat větší počet. Jestliže se elipsy jejich dosahu překrývají, zní v daném místě několik zvuků naráz. Současné přehrávání několika zvukových souborů klade vyšší nároky na počítač a jeho zvukovou kartu. Některé počítače dokážou zahrát jen velmi omezený počet zvuků (jeden, dva či tři). V takovém případě prohlížeč ocení nápovědu v podobě parametru **priority**. Ten přiřazuje uzlům relativní důležitost. Když se objeví technická nutnost některé zvuky vynechat, na řadu přijdou nejprve ty uzly, jejichž priorita je nejmenší.

TIP: Cyklicky se opakující zvuky (šplouchání přílivu, zpěv ptáků, dopravní ruch) by měly mít prioritu nastavenu na nulu, krátké jednorázové zvukové signály na jedničku.

Vraťme se nyní k prvnímu parametru z tabulky T-3-19, totiž ke zdroji zvukového signálu (**source**). Tento parametr obsahuje uzel, podle jehož parametrů lze vyhledat zvukový soubor. Existují právě dva takové uzly – **MovieTexture** a **AudioClip**. S prvním z nich jsme se setkali v kapitole věnované texturám. Pohyblivá textura (video sekvence) je zaznamenána v souboru ve formátu MPEG, který může obsahovat jak obrazová, tak zvuková data. Zvuk z takového souboru lze přiřadit do parametru **source**. Uzel **MovieTexture** obsahuje parametry potřebné pro přehrávání – čas zahájení a ukončení přehrávání, povolení opakování v nekonečné smyčce.

TIP: Když použijeme soubor MPEG současně jako zdroj zvuku i obrazové textury, je správné jej přiřadit patřičným rodičovským uzlům s pomocí konstrukce DEF a USE.

Mnohem častěji jsou pro přehrávání zvuku používány samostatné zvukové soubory. Ty mohou být ve formátu ***WAV***, většina prohlížečů pak přehrává i soubory ve formátu ***MIDI***. K jejich specifikaci slouží uzel **AudioClip**, jehož parametry obsahuje tabulka T-3-20. Stejně jako u obrazových textur lze do parametru **url** zadat několik adres souborů, prohlížeč použije první dostupnou.

parametr	iniciální hodnota	význam
url	**[]**	seznam adres s umístěním zvukového souboru
startTime	**0**	čas zahájení přehrávání
stopTime	**0**	čas ukončení přehrávání
pitch	**1.0**	rychlost přehrávání (tempo)
loop	**FALSE**	povolení přehrávání ve smyčce

Tabulka T-3-20: Parametry uzlu **AudioClip**

U zdrojů zvuku vystupuje do popředí otázka správného časování. Bylo by jistě smutné navštívit virtuální koncert a poté, kdy jsme se občerstvili v baru (alespoň pohledem, když ne doopravdy) a zabloudili ve foyeru, vstoupit do sálu, kde právě doznívají poslední tóny skladby. Z tohoto důvodu se do parametrů **startTime** a **stopTime** téměř nikdy nezapisují hodnoty absolutního času, ale pracuje se s časem relativním. Přehrávání je typicky zahájeno v okamžiku, kdy se avatar dostane do blízkosti zdroje zvukového signálu. O tom, jak to zajistit, si řekneme v kapitole 5, věnované zpracování dynamických událostí.

Uzel **AudioClip** dovoluje změnit původní tempo zaznamenaného zvuku (parametr **pitch**) a nastavit nekonečnou přehrávací smyčku (**loop**). Přehrávání ve smyčce je šikovným trikem, který „z mála udělá mnoho“. Opakování krátké sekvence vyvolává dojem trvalého děje ve virtuálním prostředí (vrčení motoru, svist větru), přičemž velikost potřebných zvukových souborů je velmi malá. Tímto způsobem současně šetříme paměť počítače. Dlouhé, jednorázově přehrávané soubory jsou proto ve virtuální realitě spíše výjimkou. Je dávána přednost krátkým sekvencím, např. hlasovému upozornění na nebezpečí, doprovodným zvukům (cink, klap, bum, šplouch) apod.

TIP: Jeden zdroj zvuku (AudioClip) může být slyšet v různých místech virtuálního světa podobně, jako je jeden zvukový záznam slyšet ve více reproduktorech, představovaných uzly Sound. Také v tomto případě je správné opakovaně definovat zdroj zvuku s pomocí konstrukce DEF a USE.

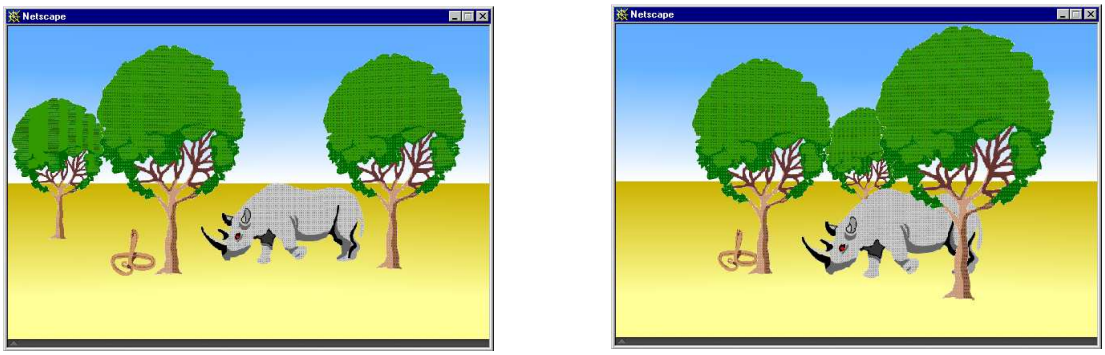
3.6 Co do toho dortu ještě dáme? (Anchor, Billboard, Group, Inline, LOD, Switch)

V předchozích kapitolách jsme se seznámili se všemi základními uzly VRML, které se používají ke stavbě virtuálních světů a tvoří tedy jakési základní kameny. Mezi uzly byly definovány vztahy rodič-potomek, které vytvářely v mnoha případech i několikastupňové stromy. V kořeni těchto stromů stál většinou jediný uzel – **Transform**. Jeho úkolem bylo seskupit potomky a dodat jim potřebné transformace.

Uzel **Transform** však není jediným uzlem, který je schopen sdružit několik potomků a zajistit jim určité společné chování. Je pouze prvním z několika tzv. *skupinových* uzlů. Jim je věnována tato kapitola.

Zcela nejjednodušším skupinovým uzlem je uzel **Group**. Má stejné parametry jako **Transform** kromě těch, které jsou zaměřeny na transformace. Lze říci, že tento uzel nemá jiný úkol, než být jakýmsi kontejnerem na uzly. S jeho pomocí uspořádáme uzly v souboru do přehlednějších skupin, každá z nich může být samozřejmě pojmenována konstrukcí **DEF**.

Daleko zajímavějším skupinovým uzlem je **Billboard**. Jeho úlohou je natáčet všechny potomky tak, aby k avatarovi byli vždy čelem. Natáčení probíhá dynamicky tak, jak se avatar pohybuje světem a mění svoji orientaci. Standardně jsou objekty otáčeny podle svislé osy, tj. osy určené vektorem (0 1 0). Její přesné umístění v prostoru je nutno zajistit rodičovským uzlem **Transform**. Ten může také změnit orientaci osy, lepší je však definovat osu přímo parametrem **axisOfRotation** uvnitř uzlu **Billboard**.



Obrázek 3-26: Použití billboardů pro modely stromů a zvířat

```
#VRML V2.0 utf8
Transform {
  children DEF STROM Billboard { children
    Shape {
      geometry IndexedFaceSet {
        coord Coordinate { point [ -1 0 0, 1 0 0, 1 3 0, -1 3 0] }
        coordIndex [ 0 1 2 3 -1]
        texCoord TextureCoordinate { point [0 0, 1 0, 1 1, 0 1] }
        texCoordIndex [0 1 2 3 -1]
      }
      appearance Appearance {
        texture ImageTexture { url "baobab.gif" }
      }
    }
  }
}

Transform { translation 3 0 –0.5  children USE STROM }

Transform { translation -3 0 –5   children USE STROM }
```

Program P-3-23:Tři stromy jako billboardy s poloprůhlednou texturou

Na první pohled se zdá, že automatické natáčení objektů vůči avatarovi je trikem, který se použije jen ve velmi speciálních případech – např. vševidoucí Boží oko nebo stále čitelný reklamní text. Ve skutečnosti je **`Billboard`** uzlem, který má nenahraditelnou funkci při zobrazování složitých přírodních objektů, jakými jsou stromy a keře, ale také lidské postavy, zvířata, ptáci. Takové objekty mívají natolik složitou geometrii, že její detailní popis by byl neúnosně rozsáhlý a v reálném čase těžko zpracovatelný. Mnohem lepší je použít obrázek, který naneseme jako texturu na rovinnou plochu a zařadíme ji jako potomka uzlu **`Billboard`**. Od tohoto okamžiku je daný obrázek zřetelně viditelný ze všech stran a snadno vznikne dojem, že namísto ploché fotografie se na daném místě vyskytuje prostorový objekt.

Příklad takového přístupu je na obrázku 3-26, kde vidíme dva různé pohledy na tentýž svět. Obrázek vpravo je získán ze stanoviště, které je umístěno vpravo vpředu od pozice avatara na levém obrázku. Stromy i zvířata jsou vždy natočena čelem k avatarovi podle svislé osy. I ze dvou pohledů si dokážeme představit, že jde o prostorové objekty. Jako tvůrci virtuálních světů si přitom uvědomíme, že osově symetrické obrázky jsou pro takový způsob zpracování ideální. Nesouměrný nosorožec jakoby nepřirozeně kráčel neustále doleva, ať se na něj díváme odkudkoliv. Část souboru s definicí stromů je ukázána v programu P-3-23. Všimněme si efektivního využití příkazů **`DEF`** a **`USE`**.

TIP:

1. **Obrázky, které používáme na billboardech, by měly vždy využívat průhlednost. Vhodnými formáty jsou GIF a PNG.**
2. **Pokud do parametru `axisOfRotation` zapíšeme hodnotu 0 0 0, změní se otáčení kolem *osy* na otáčení kolem *středu*. Potomci uzlu `Billboard` tak budou vyhlížet stejně nejen při jejich obcházení po kružnici, ale i při jejich zkoumání ze všech stran.**

Dalším skupinovým a současně i prvním z interaktivních uzlů je **`Anchor`** (česky *kotva*). V prostředí virtuální reality bychom mu spíše mohli říkat *teleportace* nebo *odkaz*. Všechny jeho potomci jsou citliví na aktivitu kurzoru (typicky stisk tlačítka myši). Jakmile uživatel aktivuje libovolný objekt-potomek, je přenesen do jiné části virtuálního světa. Reakce na aktivaci objektu může být dokonce trojího typu:

1. přechod na nové stanoviště (**`Viewpoint`**) v právě prohlíženém světě,
2. nahrazení aktuálního světa jiným s případným přechodem na určité stanoviště v novém světě,
3. aktivace hypertextového odkazu (*http*), nejčastěji s WWW stránkou.

Seznam parametrů tohoto uzlu je uveden v tabulce T-3-21. Podívejme se nejprve na to, jak přecházet mezi stanovišti virtuálních světů, ať již jde o různé světy nebo o právě zobrazovaný. Uzel **`Anchor`** obsahuje v parametru **`url`** seznam adres, na které se snaží prohlížeč přejít. Jakmile se mu to podaří, dále již seznam neprohledává. Adresy mají tvar podobný těm adresám, které známe z Internetu. Na konci mohou obsahovat rozšíření tvořené znakem **`'#'`** a jménem stanoviště. Jde o takové jméno, které bylo stanovišti přiřazeno příkazem **`DEF`**.

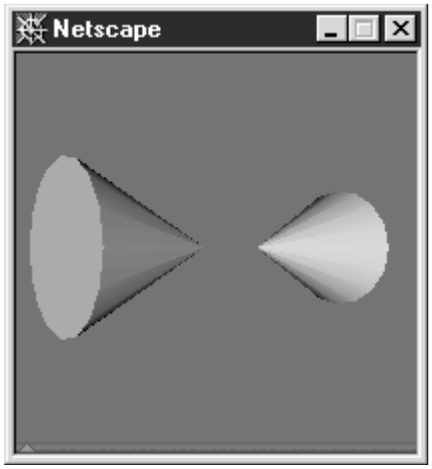
```
Anchor {
  url [ "http://www.world.org/nebesa.wrl#brana",
        "http://www.world.org/peklo.wrl",
        "#rezerva" ]

  children …
}
```

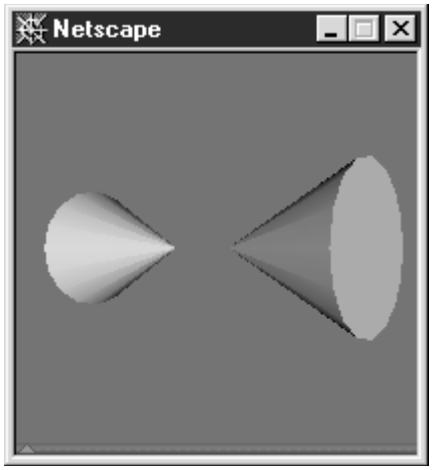
Bude-li výše uvedená část VRML souboru použita v nějakém virtuálním světě, pak se po aktivaci libovolného z potomků (**`children`**) uzlu **`Anchor`** pokusí prohlížeč o následující akce:

- Vyhledání souboru **`nebesa.wrl`** na počítači s adresou **`www.world.org`**. Tento soubor nahradí aktuální svět a avatar bude umístěn do pozice určené stanovištěm **`brana`**. Pokud stanoviště tohoto jména v souboru neexistuje, bude použito jako obyčejně první ze stanovišť v souboru **`nebesa.wrl`**.
- Pokud v předchozím kroku prohlížeč neuspěje, pokusí se vyhledat na stejné adrese soubor **`peklo.wrl`**. Protože v adrese není uvedeno jméno stanoviště, bude použito první ze souboru.
- Když se nepodaří žádný z předchozích kroků, avatar zůstane v aktuálním světě a bude přesunut na stanoviště jménem **`rezerva`**.

Ukázka v programu P-3-24 využívá uzlu **Anchor** pouze k přechodu mezi dvěma stanovišti. Jednoduchý svět je tvořen dvěma kužely. Levý ukazuje doprava a jakmile na něj uživatel ukáže kurzorem a aktivuje jej, je přesunut na stanoviště do blízkosti pravého kuželu. Odtud se obdobným způsobem může dostat zpět k levému kuželu. Doplňující obrázky k programu P-3-24 představují pohledy pojmenované **JSEM-VLEVO** a **JSEM-VPRAVO**.

```
#VRML V2.0 utf8
DEF JSEM-VLEVO Viewpoint {
  description "Predni"
  position -3 0 5
  orientation 0 1 0 -0.7
}
DEF JSEM-VPRAVO Viewpoint {
description "Bocni"
  position 6 0 5
  orientation 0 1 0 0.7
}
Anchor {
  url "#JSEM-VPRAVO"
  children Transform {
    rotation 0 0 1 -1.57
    children DEF KUZEL Shape {
      geometry Cone {}
      appearance Appearance {
        material Material {
          diffuseColor 1 0.75 0.4
        }
      }
    }
  }
}
Anchor {
  url "#JSEM-VLEVO"
  children Transform {
    translation 3 0 0
    rotation 0 0 1 1.57
    children USE KUZEL
  }
}
```





Program P-3-24: Pomocí uzlů **Anchor** *lze návštěvníka provádět od stanoviště ke stanovišti*

parametr	iniciální hodnota	význam
url	**[]**	seznam adres jiných světů, jmen stanovišť a WWW stránek
parameter	**[]**	seznam doplňujících parametrů předávaných prohlížeči po aktivaci potomka
children	**[]**	seznam potomků

Tabulka T-3-21: Parametry uzlu **Anchor**

TIP: Navigace v umělém, trojrozměrném prostředí bývá obtížná. Uzly Anchor, které realizují přesun na následující stanoviště (Viewpoint), jsou vynikajícím prostředkem pro řízenou prohlídku virtuálního světa. Je vhodné označit potomky takových uzlů Anchor jasným návěštím, např. přidat k nim uzel Text s popisnou informací. Další možností je zavedení intuitivně rozpoznatelných objektů – šipek dopředu a dozadu.

Uzel **Anchor** je současně i prostředkem pro kombinaci virtuálních světů a WWW stránek. Je jistě rozumné doplnit virtuální svět textovými informacemi zapsanými v podobě WWW stránek, tedy HTML dokumentů.

Pokud do parametru **url** zapíšeme adresu takové stránky, prohlížeč virtuálních světů (který je součástí internetového WWW prohlížeče) touto stránkou nahradí aktuální virtuální svět. Lepší je proto organizovat WWW stránky do rámů nebo dovolit otevření více oken na obrazovce. Specifikace, do kterého okna nebo rámce má být umístěna HTML stránka, se udává v parametru **parameter**. Ten je tvořen seznamem textových řetězců, které se všechny předají k úspěšně nalezené adrese z parametru **url**. Příklad použití vypadá takto:

```
Anchor {
  url       "http://www.world.org/vysvetlivky.html"
  parameter "target=jmeno_okna_nebo_ramce"

  children …
}
```

Pro úplnost uveďme, jakým způsobem se naopak vyvolá z HTML stránky virtuální svět, tedy soubor s příponou **wrl**. Nabízejí se dvě možnosti, jak odkaz na VRML soubor zapsat do HTML dokumentu:

1. Náhrada HTML dokumentu virtuálním světem:
   <A HREF=”soubor.wrl”>Informační text lákající k návštěvě virtuálního světa</A>

2. Vložení okénka s pohledem do virtuálního světa přímo do HTML stránky:
   <EMBED SRC="soubor.wrl">, případně s definovaným rozměrem a umístěním okénka
   <EMBED SRC="soubor.wrl" WIDTH=120 HEIGHT=90 ALIGN=”MIDDLE”>

Druhá varianta láká k tomu, aby autoři umístili do HTML stránky hned několik okének s virtuálními světy. Je doporučeno počet těchto okének co nejvíce omezit, neboť pro každé z nich vytváří prohlížeč poměrně rozsáhlé paměťové struktury a výrazně zatěžuje výpočetní systém.

3.6.1 Skupina versus jednotlivec

Skupinové uzly mohou obsahovat neomezený počet potomků, přičemž všichni potomci jsou si zcela rovnocenní a sdílejí vlastnost, kterou jim dodává jejich rodič – uzel typu **Transform**, **Group**, **Billboard** nebo **Anchor**. Zvláštním typem rodičovských uzlů jsou ty, které sice mají několik rovnocenných potomků, ale nikdy nejsou zobrazovány všechny, lépe řečeno je zobrazován maximálně jeden z nich.

Prvním uzlem, který se stará o několik potomků, aniž by dovolil jejich současné zobrazení, je **Switch**, česky *přepínač*. Ve svém parametru **whichChoice** obsahuje celočíselnou hodnotu – pořadí potomka, který má být zobrazen. Číslo minus jedna znamená, že všichni potomci jsou „skryti“, nezáporná hodnota vybírá potomka v pořadí, v jakém je zapsán do souboru. Potomci se zapisují do parametru **choice**, první z potomků má pořadí nula.

Existence více potomků v uzlu **Switch** by byla nesmyslná, kdybychom neměli možnost měnit hodnotu parametru **whichChoice**, tj. v průběhu prohlížení světa dynamicky nastavovat zobrazovaného potomka. Taková možnost naštěstí existuje a je často využívána při interaktivních akcích s objekty virtuálního světa. Podrobněji budeme o způsobech dynamického nastavování hodnot parametrů mluvit v kapitole 5 (*Události hýbou světem*).

TIP: Vzhledem k tomu, že uzel Switch je skupinový, musí obsahovat takové potomky, kteří by mohli stát samostatně ve VRML souboru, tj. nejčastěji Group nebo Transform. Nelze proto do uzlu Switch zařadit přímo textury, materiály, normály apod.

Posledním z rodičovských uzlů je uzel **LOD** (*Level Of Detail*), o kterém se dá s mírnou nadsázkou říci, že znamená převrat v klasickém pojetí modelů prostorových objektů. Až dosud jsme vycházeli z představy, že model jakéhokoliv tělesa by se měl svými vlastnostmi a tvarem co nejvíce blížit skutečnému objektu. Je pravda, že při prohlížení takového objektu zblízka by měly být vidět jeho veškeré detaily, jemná prohnutí povrchu, vzory textur, škála barevných odstínů vzniklá odrazem světla apod. Potřebujeme však mít takový objekt vymodelován se všemi podrobnostmi i tehdy, když jsme od něj vzdáleni desítky či stovky metrů? V takové vzdálenosti nejsme přece schopni tyto detaily rozpoznat. Navíc od určité vzdálenosti vnímáme objekty jen jako barevné skvrny, nic víc. Domyslíme-li tyto skutečnosti do důsledků, dojdeme k závěru, že z hlediska rychlosti zpracování virtuálního světa by bylo výhodné mít pro každý objekt v zásobě několik modelů (reprezentací), které by byly zobrazovány podle toho, jak daleko jsou v dané chvíli od avatara, resp. jak důležité jsou jejich detaily v závislosti na vzdálenosti od pozorovatele. Právě takový způsob modelování objektů zajišťuje uzel **LOD**.

parametr	iniciální hodnota	význam
level	**[]**	seznam reprezentací objektu s postupně klesající přesností (množstvím detailů)
center	**0 0 0**	bod, vůči kterému se měří vzdálenost objektu od avatara
range	**[]**	rostoucí posloupnost kladných vzdáleností indikujících přepnutí reprezentace

Tabulka T-3-22: Parametry uzlu **LOD**

Tento uzel se navenek tváří jako jediný objekt. Uvnitř však skrývá několik potomků – reprezentací téhož objektu v různých stupních přesnosti. Potomci se zapisují do parametru **level** (viz tabulka T-3-22) postupně od nejdokonaleji vymodelované reprezentace až po nejjednodušší. Mezní vzdálenosti, ve kterých dochází k přepnutí mezi dvěma reprezentacemi, se ukládají do parametru **range**. Objekt může mít velmi rozmanité tvary a rozměry. Je proto definován vztažný bod (nejčastěji v pomyslném těžišti objektu), vůči kterému se měří aktuální vzdálenost k avatarovi (parametr **center**).

Obrázek 3-27: Tři reprezentace téhož tělesa uzlem **LOD**.

```
#VRML V2.0 utf8
LOD {
  range [ 15, 30, 40]
  level [
    Transform {       # model 0 – kužel
      translation 0 1.5 0
      children Shape {
        appearance DEF MODRA Appearance {
          material Material {diffuseColor 0.2 0.3 1} }
        geometry Cone { bottomRadius 1  height 3 }
      }
    }
    Shape {           # model 1 - čtyřboký jehlan
      appearance USE MODRA
      geometry IndexedFaceSet {
        coord Coordinate { point [-1 0 1, 1 0 1, 1 0 -1, -1 0 -1, 0 3 0 ] }
        coordIndex [ 0 1 4 -1, 1 2 4 -1, 2 3 4 -1, 3 0 4 -1, 0 3 2 1 -1 ]
      }
    }
    Billboard {       # model 2 - trojúhelník na billboardu
      children Shape {
        geometry IndexedFaceSet {
          coord Coordinate { point [ 1 0 0, 0 3 0, -1 0 0 ] }
          coordIndex [ 0 1 2 ]
          colorPerVertex FALSE
          color Color { color 0.2 0.3 1 }
        }
      }
    }
    Group {}          # model 3 – nic
  ]
}
```

Program P-3-25: Odstupňování detailů modelu ve čtyřech krocích v uzlu **LOD**

Na obrázku 3-27 jsou vidět postupné změny tvaru tělesa při rostoucí vzdálenosti avatara od něj. Kužel je modelován uzlem **Cone** pouze v největší blízkosti (obrázek vlevo). Tehdy může být tvořen i několika desítkami povrchových trojúhelníků. Ve větší vzdálenosti (obrázek uprostřed) postačuje model reprezentovaný uzlem **IndexedFaceSet** v podobě čtyřbokého jehlanu. Ještě dále můžeme použít jediný trojúhelník vnořený jako potomek uzlu **Billboard**, aby působil ze všech stran stejným dojmem[8].

Jakmile vzdálenost přeroste jistou mez, může být zobrazování objektu zcela potlačeno. Taková situace již není na obrázku 3-27 znázorněna, je však ukázána v programu P-3-25. Posledním z potomků je prázdný uzel **Group**, takže poslední reprezentace nemá žádnou geometrii a není tudíž vůbec zobrazována.

Počet vzdáleností v seznamu **range** by měl být o jedničku nižší než je počet reprezentací v seznamu **level**. V příkladu P-3-25 vidíme, že ve vzdálenosti 0 až 15 m od avatara bude vykreslován objekt v podobě kuželu, v rozsahu 15-30 m v podobě jehlanu, ve vzdálenosti 30-40 m jako trojúhelník a jakmile vzdálenost přesáhne 40 m, objekt zmizí.

```
#VRML V2.0 utf8
LOD {
  range [ 30 ]   # jednoduché rozdělení na blízké a vzdálené modely
  level [
    LOD {        # automatický výběr ze dvou modelů pro pohled zblízka
      level [
        Transform {   # model 0a - kužel ...                    }
        Shape {       # model 1a - čtyřboký jehlan ...          }
      ]
    }
    LOD {        # automatický výběr ze dvou modelů pro pohled zdálky
      level [
        Billboard {   # model 0b - trojúhelník na billboardu ... }
        Group {}      # model 1b – nic
      ]
    }
  ]
}
```

Program P-3-26: Kombinace automatického a doporučeného přepínání mezi reprezentacemi v uzlu **LOD**

Šikovným řešením je ponechat parametr **range** prázdný. Prohlížeč v takovém případě není odkázán na předepsané vzdálenosti, ve kterých přepíná mezi reprezentacemi, nýbrž sám volí tu reprezentaci, kterou je schopen na konkrétním počítači vykreslit v dostatečně krátkém čase. Zdálo by se, že nejlepší je vždy ponechat seznam **range** prázdný, ať se prohlížeč rozhoduje automaticky. To by však na pomalých počítačích mohlo znamenat, že návštěvník virtuálního světa nikdy neuvidí objekty se všemi detaily. Nejrozumnější je kompromisní přístup, který je ukázán v programu P-3-26. V něm je pro jeden objekt použito několik uzlů **LOD** v hierarchickém uspořádání. Nejvyšší uzel **LOD** definuje jedinou přepínací vzdálenost. Pro objekt, nacházející se před touto hranicí, se vybere jeden ze dvou „lepších“ modelů – kužel nebo jehlan. Volba závisí na rozhodnutí prohlížeče, neboť odpovídající synovský uzel **LOD** má prázdný parametr **range**. Podobně pro objekt ve větší vzdálenosti se vybírá z dvojice „horších“ modelů – buď trojúhelník nebo nic. Tímto způsobem je zaručeno, že na rychlých počítačích bude podle vzdálenosti vidět kužel nebo trojúhelník, na velmi pomalých počítačích jehlan nebo nic.

TIP: Uzel LOD je základním prostředkem pro docílení vysoké rychlosti zobrazování světa. Měli bychom jej používat co nejčastěji, i když tím zvyšujeme celkovou paměťovou náročnost.

Již u tak jednoduchého tělesa, jakým je kužel, dokážeme pomocí uzlu **LOD** postupně snižovat počet zpracovávaných ploch z několika desítek na jednotky. Ještě výraznější je efekt uzlu **LOD** v kombinaci se složitými prostorovými útvary, jakými jsou například modely postav či detaily technických zařízení. Taková

[8] Základní tělesa, jakými jsou koule, válec a kužel, jsou ve většině prohlížečů zjednodušována automaticky. Použití uzlu **LOD** v příkladu P-3-25 tedy pouze ilustruje metodu, avšak pro zjednodušování kuželu nemá velký praktický význam.

tělesa obsahují nezřídka i několik set plošek a bez použití **LOD** bychom virtuální svět stěží zvládli procházet plynule v reálném čase.

TIP: Zkušený pozorovatel si všimne, že jednotlivé reprezentace použité v uzlu LOD se mění skokově, naráz. Výrazné je to u „nejhorší“ reprezentace, která se najednou objeví nebo najednou zmizí. Optické plynulosti docílíme kombinací s globálním uzlem Fog - z mlhy se hladce vynoří nejvzdálenější reprezentace.

3.6.2 Svět jako stavebnice z kostek

Jedním z principů, který je bohatě využíván v počítačové technice, je opakování a znovu využívání již hotových částí. Menší programy, které osvědčily svoji funkci, jsou vkládány do větších programů a pouze zcela nové části je třeba vytvářet od začátku. Tento postup je ve velké šíři používán i při konstruování virtuálních světů. Dobře navržené modely těles, které jsou samostatně zapsány v souborech, je možno vložit na libovolné místo do nového, vlastního světa. Lze dokonce spolupracovat v rámci Internetu a vytvářet kompozice složené z částí, které se nacházejí po celém světě.

K pospojování více virtuálních objektů a světů do většího celku slouží uzel **Inline**. Do jeho parametru s již známým jménem **url** zapíšeme seznam adres, na kterých prohlížeč hledá VRML soubory. Po úspěšném nalezení prvního souboru je jeho obsah vložen do toho místa ve stromové struktuře aktuálního světa, ve kterém je uzel **Inline** uveden. Vložený soubor je proto ovlivňován vlastnostmi případných rodičovských uzlů. Je-li některý z jeho rodičů uzlem **Transform**, bude vložený objekt příslušným způsobem transformován, je-li jeho rodičem **Billboard**, bude dynamicky natáčen podle polohy avatara atd.

```
#VRML V2.0 utf8
LOD { range [ 30 ]
      level [
        LOD { level [ Inline { url "kuzel.wrl" }
                      Inline { url "jehlan.wrl" }
                    ]
        }
        LOD { level [ Inline { url "trojuhelnik.wrl" }
                      Group  { }
                    ]
        }
      ]
}
```

*Program P-3-27: Doporučené použití uzlu **Inline** uvnitř uzlu **LOD***

Velmi šikovné je kombinovat vkládání objektů s násobnou reprezentací tělesa uzlem **LOD**. Program P-3-26 by s použitím uzlu **Inline** vypadal tak, jak je ukázáno v příkladu P-3-27. Nejenže je výsledný tvar souboru přehledný, ale navíc docílíme i lepšího chování prohlížeče při iniciálním načítání virtuálního světa. Když je totiž celý svět zapsán v jediném souboru, prohlížeč jej musí nejprve celý přečíst a teprve poté jej zobrazí. Tato akce může trvat i řadu sekund. Je-li naopak soubor malý a je kompozicí více samostatných souborů vkládaných pomocí **Inline** a **LOD**, prohlížeč rychle přečte základní soubor o malé velikosti a okamžitě se pokusí jej zobrazit. Mezitím hledá soubory určené ke vkládání a postupně jimi doplňuje virtuální svět. Návštěvníkovi se tak objekty doslova rodí před očima. Nejdůležitější však je, že některé objekty v uzlu **LOD** nemusí být načteny okamžitě – v příkladu P-3-27 stačí například načíst a zobrazit obsah souboru **"trojuhelnik.wrl"** (jsme-li dostatečně daleko) a teprve za nějakou chvíli zpracovat zbylé dva vkládané soubory. Jinak řečeno – abychom spatřili svět z programu P-3-26, musíme jej celý načíst. Abychom viděli tentýž svět, zapsaný přitom jiným způsobem v programu P-3-27, stačí přečíst zhruba třetinu dat. Pokud je dokonce daný objekt mimo zorný úhel našeho pohledu, nemusíme načítat vůbec žádná další data. Podobně jako u uzlu **LOD** platí, že uvedený postup je tím efektivnější, čím větší světy zpracováváme.

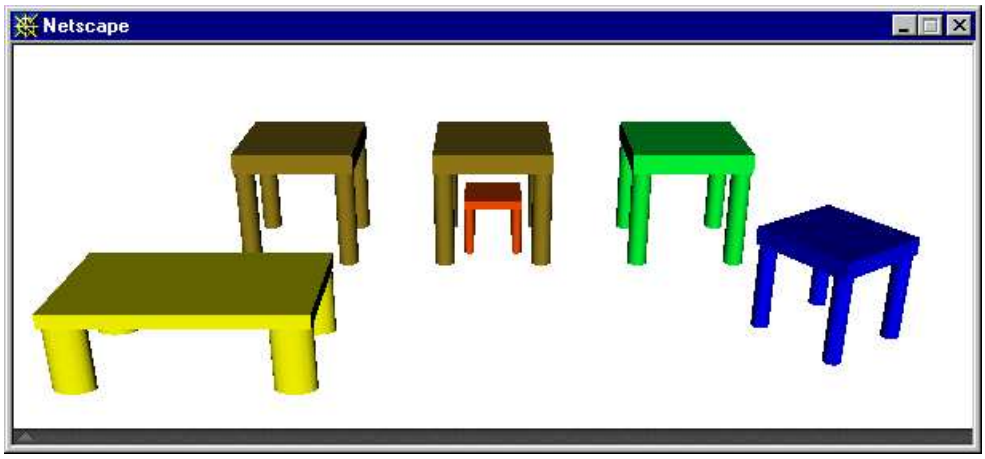
TIP: Svět, vkládaný uzlem Inline, musí být platným, samostatným VRML souborem. Nelze tedy vkládat například pouze uzel definující materiál.

Je zřejmé, že dokud prohlížeč nenačte celý obsah vkládaného světa, není schopen zjistit jeho umístění a rozměry v prostoru. Tyto údaje přitom mohou být poměrně důležité z hlediska určování viditelnosti a zákrytu těles,

detekce kolizí s avatarem a jeho navigace při průchodu světem. Uzel **`Inline`** proto obsahuje dva parametry, do kterých můžeme zapsat umístění a rozměry obálky ve tvaru pomyslného kvádru obklopujícího vkládaný svět, čímž poskytneme prohlížeči potřebnou nápovědu. Kvádr, představující obálku, má stěny rovnoběžné s rovinami systému souřadnic (je tzv. *osově orientovaný*) a zadává se dvojicí parametrů **`bboxSize`** (rozměry kvádru) a **`bboxCenter`** (poloha těžiště). Dokud prohlížeč nepřečte celý obsah vkládaného světa, může pro informaci návštěvníka zobrazovat na příslušném místě drátový model kvádru. Připomeňme, že obálka neslouží jako prostředek pro zmenšení a posunutí vkládaného světa. Takové transformace je nutno zajistit v rodičovském uzlu **`Transform`**.

4 Chci mít originál (Proč? – PROTO!)

Oblíbená odpověď rodičů na příliš dotěrné otázky dětí v podtitulu této kapitoly je současně jedním z nejmocnějších příkazů jazyka VRML. Je zkratkou anglického slova *prototype* a dovoluje definovat nové, vlastní uzly a stromové struktury, vytvářet knihovny specializovaných objektů. Pomocí konstrukce **PROTO** zkrátka dokážeme popsat to, co nám v jazyce VRML chybí, přičemž výsledek naší práce bude vypadat stejně, jako kterýkoliv standardní VRML uzel. Prototyp představuje vzor uzlu, který můžeme vkládat do stromové struktury VRML a modifikovat nastavením jeho parametrů.



Obrázek 4-1: Skupina těles vzniklá z jediného prototypu

Podívejme se nejprve na obrázek 4-1, na němž je několik stolků v různých velikostech, polohách a barvách. Tento svět byl vytvořen programem P-4-1 a všechny stolečky byly vygenerovány podle stejného vzoru – *prototypu*. Kdybychom je chtěli popsat pouze pomocí příkazů **DEF** a **USE**, těžko bychom dokázali měnit jejich barvy, protože informace o materiálu se nacházejí uvnitř stromu definujícího stůl jako celek. Mohli bychom je pouze zařadit k rodičům **Transform** a nastavit tak jejich polohu a velikost.

```
#VRML V2.0 utf8
EXTERNPROTO Stul
      [ field SFColor    barva
        field SFVec3f    posunuti
        field SFRotation natoceni
        field SFVec3f    meritko]
      "../knihovny/nabytek.wrl#MujStolek"

Group {
  children [ Stul { }
             Stul { posunuti -2 0 0 }
             Stul { barva    0 1 0.2     posunuti 2 0 0 }
             Stul { barva    1 0.3 0     meritko  0.5 0.5 0.5 }
             Stul { posunuti -2.5 0 2.5  meritko  1.8 0.6 1 barva 1 1 0 }
             Stul { posunuti 3 0 2       natoceni 0 1 0 0.6
                    meritko  0.8 0.8 0.8 barva    0 0 1 }
  ]
}
```

Program P-4-1:Použití prototypu definovaného v souboru **nabytek.wrl**.

Program P-4-1 je tvořen dvěma částmi. V první jsou popsány parametry stolku, ve druhé je tento stolek opakovaně vkládán do virtuálního světa jako běžný uzel se jménem **Stul**. Stejně jako u standardních uzlů, také zde má každý z jeho parametrů předdefinovanou hodnotu (ta zatím není vidět), takže při vkládání stolů do virtuálního světa stačí zapisovat jen ty jejich parametry, jejichž hodnoty se liší od základních. Toho si můžeme všimnout na posledních řádcích programu P-4-1, kde postupně vznikne šest stolů s různými vlastnostmi danými rozličnými kombinacemi nových a předdefinovaných hodnot parametrů.

Dříve, než uvedeme obsah souboru **nabytek.wrl** (v němž čtenář správně očekává důmyslný zápis definice prototypu stolu), všimneme si první části ukázky P-4-1. Hned za hlavičkou souboru jsou zapsány údaje, které blíže specifikují parametry stolku. Nejprve je uvedeno klíčové slovo **EXTERNPROTO**, které oznamuje, že následující prototyp bude definován v samostatném (externím) souboru a že v aktuálním souboru nenalezneme nic víc, než parametry tohoto prototypu. V rámci aktuálního souboru je prototypu přiřazeno uživatelské jméno **Stul** – toto jméno přitom nemusí být nutně shodné se jménem opravdového prototypu **MujStolek** zapsaného v adresáři **knihovny** v souboru **nabytek.wrl**.

V hranatých závorkách je pak uveden seznam parametrů. V ukázce je každý parametr charakterizován třemi slovy:

```
field        <typ dat>          <jméno parametru>
```

Slovo **field** patří přímo do jazyka VRML. Místo něj by se mohla objevit ještě další tři klíčová slova jazyka VRML, avšak o nich budeme mluvit podrobněji až v příští kapitole. V tuto chvíli vystačíme se zjednodušeným vysvětlením, že slovo **field** slouží k označení takového parametru, který má předdefinovanou hodnotu.

4.1 Typy dat

Každý parametr má jednoznačně určeno, jaké hodnoty do něj lze ukládat. Na výběr je několik datových typů, přičemž většina z nich se může vyskytovat ve dvou variantách. Buď v podobě jediné samotné hodnoty (předpona **SF** – *Single Field*) nebo jako seznam hodnot (předpona **MF** – *Multiple Field*), jehož délka obecně není omezena. Pod pojmem *jediná* hodnota rozumíme současně i hodnotu jednoho strukturovaného parametru, například barvy, která je ve skutečnosti dána třemi složkami R, G a B. Následující tabulka uvádí přehled datových typů, tj. těch klíčových slov, která určují povolený charakter dat parametru.

Datový typ (jednoduchý nebo seznam)		*význam*	*příklad hodnot*
SFBool	**---**	logická hodnota	TRUE, FALSE
SFInt32	**MFInt32**	celé číslo v rozsahu 32 bitů	-42, 0, 123456
SFFloat	**MFFloat**	číslo s desetinnou tečkou	-3.14, .001, 55
SFTime	**MFTime**	čas v sekundách	0.0, 60
SFVec2f	**MFVec2f**	vektor v rovině	3 –2, 1 1, -.5 1.7
SFVec3f	**MFVec3f**	vektor v prostoru	3 –2 5, 1 1 0, -.5 1.7 –3.33
SFColor	**MFColor**	barva ve složkách RGB, každá z intervalu <0, 1>	0 0 0, 1 1 1, .7 .5 0.3
SFRotation	**MFRotation**	osa (vektor v prostoru) a úhel rotace (v radiánech)	0 0 1 1.57, -1 0 0 -3.14
SFNode	**MFNode**	uzel VRML	
SFString	**MFString**	textový řetězec	“Dobrý den.“, “To je ale pěkná knížka!“
SFImage	**---**	strukturovaný vzor pixelů	viz uzel **PixelTexture** v kap. 3.2.1

Tabulka T-4-1: Typy dat

TIP: Čísla s desetinnou tečkou, která jsou v absolutní hodnotě menší než jedna, se mohou psát bez nuly na začátku.

Pozorný čtenář nyní tuší, že všechny parametry, o kterých jsme mluvili v předchozích kapitolách, musejí také mít stanoven svůj datový typ. Skutečně tomu tak je a ačkoliv se jména datových typů do VRML souboru běžně nezapisují, je dobré si uvědomit, že existují a že určují přesně povolený rozsah hodnot. Například všude, kde jsme uváděli, že parametr obsahuje *seznam* hodnot (parametr **`point`** uzlu **`Coordinate`** či parametr **`children`** uzlu **`Transform`**), šlo o datové typy s předponou **`MF`** (v tomto případě **`MFVec3f`** a **`MFNode`**). Je také dobré vědět, že stejnojmenné parametry **`SF`** a **`MF`** nelze navzájem přiřazovat. Není tedy možno např. zapsat seznam **`MFColor`** do parametru typu **`SFColor`**.

```
#VRML V2.0 utf8
PROTO MujStolek
      [ field SFColor    barva    .6 .5 .1
        field SFVec3f    posunuti 0 0 0
        field SFRotation natoceni 0 1 0 0
        field SFVec3f    meritko  1 1 1 ]
{
  Transform {
    translation IS posunuti
    rotation    IS natoceni
    scale       IS meritko
    children [
      Transform {              # deska stolu
        translation 0 1.1 0    # 110 cm nad zemí
        children
          Shape {
            appearance DEF BARVA Appearance {
              material Material { diffuseColor IS barva }
            }
            geometry Box { size 1.2  0.2 1.2 }
          }
      }
      Transform {   # první noha od stolu
        translation -.5 0.5 -.5
        children
          DEF NOHA Shape {
            appearance USE BARVA
            geometry Cylinder { height 1 radius .1 top FALSE}
          }
      }
      Transform {   # další noha od stolu
        translation .5 0.5 -.5
        children USE NOHA
      }
      Transform {   # další noha od stolu
        translation -.5 0.5 .5
        children USE NOHA
      }
      Transform {   # další noha od stolu
        translation .5 0.5 .5
        children USE NOHA
      }
    ]
  }
}
```

Program P-4-2:Obsah souboru **`nabytek.wrl`** *s prototypem stolku*

Nyní se již můžeme vrátit zpět k programu P-4-1 a pokračovat v rozboru jeho první části. Všechny čtyři parametry prototypu **`Stul`** jsou datovými typy obsahujícími jednu hodnotu (*single field*). Jejich jména jsou vymyšlena tvůrcem prototypu. Na konci seznamu parametrů je uvedena adresa souboru, ve které se nachází definice prototypu. V našem případě je to soubor **`nabytek.wrl`**. Obecně může být takových adres několik,

takže tvoří seznam uzavřený do hranatých závorek. Program s nimi pracuje stejně, jako se známým parametrem **url**. Postupně adresy prochází a jakmile najde první vyhovující, nalezený soubor použije. Soubor **nabytek.wrl** je zapsán v programu P-4-2, nově zavedené parametry jsou zvýrazněny podtržením.

Prototyp stolku je uvozen slovem **PROTO**, za nímž následuje jméno prototypu. Všimněme si, že právě toto jméno je vyhledáváno v adresách zapsaných v příkazu **EXTERNPROTO** (viz program P-4-2, zápis **"nabytek.wrl#MujStolek"**). Za jménem následuje seznam parametrů nově definovaného uzlu, tentokrát včetně iniciálních hodnot. Nakonec je ve složených závorkách vytvořen prototyp z uzlů VRML, ať již standardních nebo nově zavedených v nějaké dřívější konstrukci **PROTO**. V našem příkladu vznikne prototyp stolu z pěti těles – desky stolu a čtyř noh. Tělesa mají přiřazenu společnou barvu a všechna jsou potomky hlavního rodičovského uzlu **Transform**.

Programátorům tento zápis jistě připomíná deklaraci podprogramu (funkce) v běžných programovacích jazycích. Aby tento příměr byl na místě, je třeba zajistit předání parametrů nově deklarovaného uzlu do parametrů již existujících uzlů, jinými slovy je třeba zavést *přiřazovací příkaz*. Ten je realizován slovem **IS**, reprezentujícím rovnítko v jazyce C nebo výraz '**:=**' v Pascalu. Tímto způsobem jsou nové transformační parametry **posunuti**, **natoceni** a **meritko** zapsány do skutečných parametrů **translation**, **rotation** a **scale**. Podobně je parametr **barva** předán parametru **diffuseColor** uzlu **Material**. Je zajímavé, že jména nově zaváděných parametrů nemusí být odlišná od existujících. Můžeme se tak například setkat se zápisem

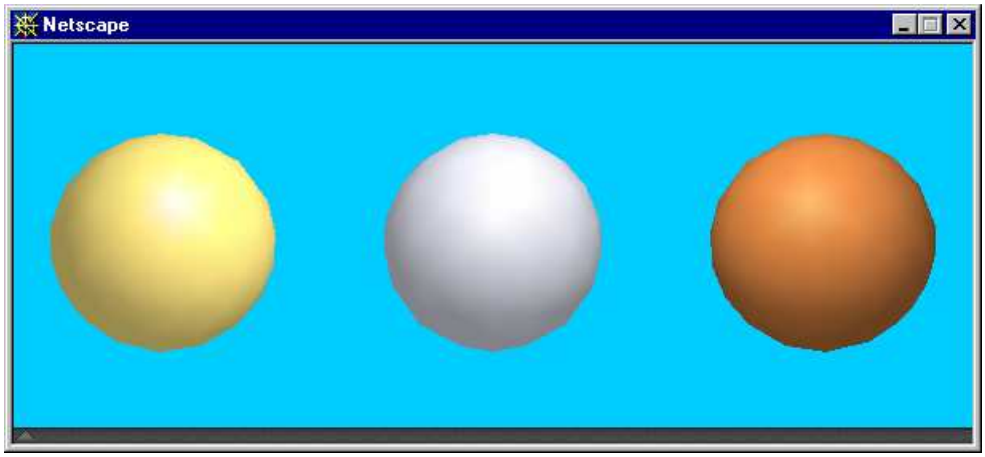
```
rotation IS rotation
```

který na první pohled vypadá nesmyslně, avšak ve skutečnosti správně přiřazuje hodnotu jednoho parametru do parametru jiného. Je dokonce doporučováno,aby jména nově zaváděných parametrů byla shodná s již existujícími, pokud vyjadřují stejnou vlastnost.

Shrňme tedy všechny příkazy, které se objevují v souvislosti s prototypy – **PROTO**, **EXTERNPROTO** a **IS**. Formálně se zapíší ve tvaru:

PROTO jméno **[** seznam deklarací parametrů **]** **{** stromové struktury uzlů **}**
<existující parametr> **IS** <nově definovaný parametr>
EXTERNPROTO jméno **[** seznam deklarací parametrů bez předdefinovaných hodnot **]** adresa prototypu nebo **[** seznam adres **]**

TIP: První uzel v definici prototypu (rodič prvního stromu) určuje druh celého prototypu a tedy i to, do kterých míst VRML stromu lze nově definovaný prototypovaný uzel zařadit. Je doporučeno definovat prototyp právě jedním stromem. Ostatní stromy jsou ignorovány až na speciální výjimky – senzory, interpolátory a skripty (viz kap. 4 až 6).



Obrázek 4-2: Použití knihovny prototypů materiálů

Ne vždy musí mít nově definovaný uzel parametry. Často se naopak zavádějí nové uzly proto, aby se zjednodušilo složité zadávání parametrů již existujících uzlů. Příkladem je knihovna materiálů, která pod jednoduchými názvy jako jsou **`Zlato`**, **`Stribro`** či **`Bronze`** skrývá množství hodnot pro difúzní a zrcadlový odraz světla. Program P-4-3 ukazuje tvar takové knihovny, program P-4-4 je příkladem jejího využití. Výsledky jsou vidět na obrázku 4-2. Všimněme si, jak lze zacházet se jmény prototypů v místě deklarace a v místě použití. V programu P-4-4 jsou dvě ze jmen materiálu přejmenována z anglických na česká, poslední jméno (**`Bronze`**) je ponecháno beze změny.

```
#VRML V2.0 utf8
PROTO Gold []
{ Material {  diffuseColor  1 .9 .5     ambientIntensity 1
              specularColor 1 1 1       shininess        1 }
}

PROTO Silver []
{ Material {  diffuseColor  .85 .85 .9  ambientIntensity 1
              specularColor 1 1 1       shininess        1 }
}

PROTO Bronze []
{ Material {  diffuseColor  1 .6 .3     ambientIntensity .7
              specularColor 1 1 1       shininess        .7 }
}
```

Program P-4-3: Obsah souboru **`kovy.wrl`** *s prototypy materiálů*

```
#VRML V2.0 utf8
Background     { skyColor 0 .8 1 }
PointLight     { location 1 5 3 intensity 0.6 ambientIntensity 0.6 }
NavigationInfo { headlight FALSE }

EXTERNPROTO Zlato   [] "../knihovny/kovy.wrl" # lépe však "kovy.wrl#Gold"
EXTERNPROTO Stribro [] "../knihovny/kovy.wrl#Silver"
EXTERNPROTO Bronze  [] "../knihovny/kovy.wrl#Bronze"

Group {
  children [
     Transform {
       translation -3 0 0
       children Shape {
         geometry DEF KOULE Sphere {}
         appearance Appearance { material Zlato {} }
       }
     }
     Transform {
       children Shape {
         geometry USE KOULE
         appearance Appearance { material Stribro {} }
       }
     }
     Transform {
       translation 3 0 0
       children Shape {
         geometry USE KOULE
         appearance Appearance { material Bronze {} }
       }
     }
  ]
}
```

Program P-4-4: Použití knihovny materiálů

Ukázka vytvoření a použití knihovny materiálů zároveň dokládá, že při tvorbě prototypů musíme myslet na to, které z parametrů již existujících uzlů zpřístupníme pomocí nově pojmenovaných parametrů a které naopak skryjeme. Knihovna materiálů ponechává uživateli dostatek prostoru pro kombinování nových materiálových uzlů s jinými uzly, například s texturou. To je důvodem, proč se uvnitř souboru **`kovy.wrl`** nevyskytuje přímo uzel **`Appearance`**, ale pouze jeho potomci – uzly **`Material`**. Naopak prototyp uzlu **`MujStolek`** z programu P-4-2 představuje přístup, při němž má mnoho uzlů nastaveny parametry neměnným způsobem (rozměry desky stolu a jeho noh). Jednoduchost použití stolku v tomto případě kontrastuje s omezenými možnostmi jeho variability.

TIP: V zápisu `EXTERNPROTO` je doporučeno skládat adresu prototypu ze jména souboru následovaného znakem '#' a jménem odpovídajícího příkazu `PROTO`. Objeví-li se pouze jméno souboru, je vybrán první prototyp, který se v tomto souboru nachází.

Dosud se zdálo, že uzel, nově definovaný konstrukcí **`PROTO`**, musí být zapsán v samostatném souboru. Ve skutečnosti se může vyskytovat kdekoliv, tedy uvnitř jakéhokoliv běžného VRML souboru. Pochopitelně v něm musí být zapsán dříve, než bude poprvé použit. Je-li nový uzel použit ve stejném souboru, v jakém byl definován, nepoužívá se zápis **`EXTERNPROTO`**. Využívání prototypů se tak objevuje ve dvou variantách, jak ukazuje následující příklad P-4-5.

```
#VRML V2.0 utf8
PROTO Stul_Original  [ … ] { … }
PROTO Zidle_Original [ … ] { … }
```

```
#VRML V2.0 utf8
EXTERNPROTO Stul [ … ]
  "nabytek.wrl#Stul_Original"
EXTERNPROTO Stul [ … ]
  "nabytek.wrl#Zidle_Original"

# normální obsah VRML souboru

Group {
  children [
    Stul  { … }
    Zidle { … }
  ]
}
```

```
#VRML V2.0 utf8
PROTO Stul_Original  [ … ] { … }
PROTO Zidle_Original [ … ] { … }

# …
# normální obsah VRML souboru
# …

Group {
  children [
    Stul_Original  { … }
    Zidle_Original { … }
  ]
}
```

Program P-4-5: Prototyp definovaný v samostatném souboru (vlevo) musí být zpřístupněn příkazem **`EXTERNPROTO`**. *Prototyp definovaný v aktuálním souboru (vpravo) je dostupný přímo, ale nelze jej přejmenovat.*

Přístup, uvedený v příkladu P-4-5 vpravo, nebrání použití prototypů v dalších souborech. Nově definované uzly lze „vytahovat“ příkazem **`EXTERNPROTO`** z jakéhokoliv souboru.

Práce s prototypy poskytuje nejvíce možností pro manipulace s uzly a jejich parametry. Dříve uvedené metody, jakými byly konstrukce **`DEF`** a **`USE`** nebo světy vkládané uzlem **`Inline`** byly pak výhodné svojí jednoduchostí. Tabulka T-4-2 obsahuje srovnání těchto příbuzných, avšak funkčností odlišných přístupů.

	`DEF` a **`USE`**	**`Inline`**	**`PROTO`**
vkládání uzlu na libovolné místo (k libovolnému rodiči)	**ano**	ne, pouze do skupinových uzlů	**ano**
využití externích (vzdálených) souborů	ne	**ano**	**ano**
možnost změny parametrů	ne	ne	**ano**

Tabulka T-4-2: Porovnání metod, vkládajících do virtuálního světa další uzly.

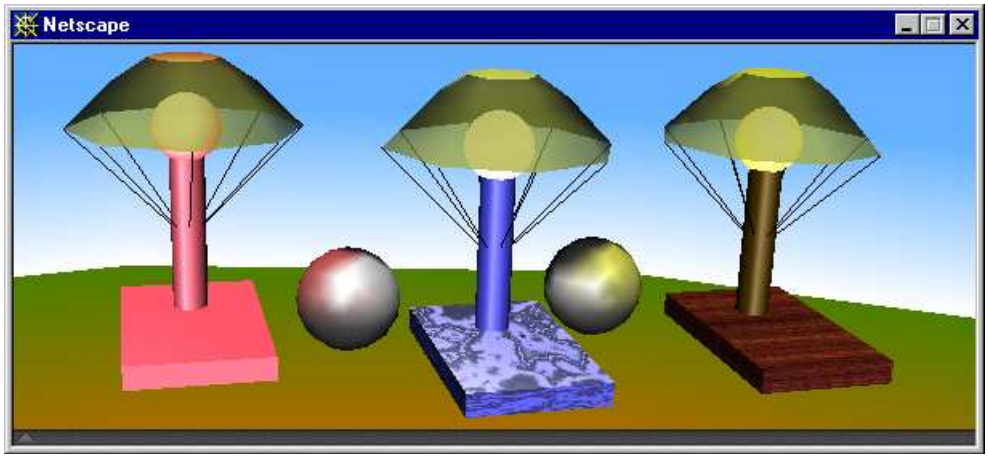
Krok za krokem V – Prototyp lampičky

Model lampy, jehož tvorba nás provází celou knihou, je výborným kandidátem pro použití prototypů. Snadno si představíme množství variant, které lze z jediného prototypu lampičky vygenerovat – různě zbarvená stínítka s odlišným stupněm průhlednosti, volitelná výška lampy a rozměrů podstavce, lesklost či naopak matnost povrchu apod. V následující ukázce jsme z mnoha možných parametrů vybrali pouze tři – barvu světla vyzařovaného lampou (parametr **`barvaSvetla`**), celkovou barvu, která je použita jak na obarvení povrchu nožky, tak podstavce (parametr **`barvaLampy`**) a nakonec texturu, která je nanesena na podstavec (parametr **`texturaPodstavce`**). Tyto parametry jsou v programu P-4-6 zvýrazněny podtržením.

```
#VRML V2.0 utf8

EXTERNPROTO Nozka-V [field SFColor color] "../knihovny/protonozka.wrl"

PROTO Lampa-V [field SFColor  barvaSvetla      1 1 1
               field SFColor  barvaLampy       0.3 0.3 1
               field MFString texturaPodstavce "../knihovny/mramor.gif" ]
{Group {
  children [
    SpotLight {    # žluté světlo
      color       IS barvaSvetla  radius      0.5
      location    0 0.3 0         direction   0 -1 0
      cutOffAngle 0.56            attenuation 0.5 0 0
    }
    Transform { # žárovka ve tvaru koule
      translation 0 0.18 0
      children Shape {
        geometry Sphere {radius 0.03}
        appearance Appearance {
          material Material { emissiveColor IS barvaSvetla }
        }
      }
    }
    Transform { # podstavec ve tvaru kvádru
      translation 0 0.01 0.04
      children Shape {
           geometry Box {size 0.12 0.02 0.2}
           appearance Appearance {
             material Material {
                diffuseColor  IS barvaLampy
                specularColor 1 1 1    ambientIntensity  1.0
              }
             texture ImageTexture { url IS texturaPodstavce }
           }
      }
    }
    Nozka-V { color IS barvaLampy }
    Inline  { url "../knihovny/stinitko.wrl"  }
  ]
 }
}
```

Program P-4-6: Obsah souboru **`protolampa.wrl`** *s prototypem lampičky*

Prototyp lampičky využívá dalšího prototypu – nožky. Předává do něj celkovou barvy lampy. Stínítko je naopak vkládáno pomocí uzlu **`Inline`** jako celek, takže jeho barva a průhlednost se nedají změnit.



Obrázek 4-3: Prototyp lampičky vkládaný s různě nastavenými parametry

Program P-4-7 je příkladem použití prototypu lampičky ve třech variantách, výsledek je na obrázku 4-3. Uprostřed je model, jehož parametry se shodují s lampičkou uvedenou v minulé kapitole. Má modře zbarvenou texturu podstavce a jeho světlo je bílé. Vlevo je lampička s růžovou barvou povrchu i světla. Zadáním tzv. prázdného řetězce (**""**) do parametru **texturaPodstavce** z ní odstraníme texturu. Vpravo je pak původní textura mramoru změněna na texturu dřeva, lampička vydává žluté světlo. Dvě stříbřité koule byly přidány do výsledného světa, abychom viděli vliv barevného světla na ostatní objekty.

```
#VRML V2.0 utf8
EXTERNPROTO Lampa
  [ field SFColor  barvaSvetla
    field SFColor  barvaLampy
    field MFString texturaPodstavce
  ] "../knihovny/protolampa.wrl#Lampa-V"

Group {
  children [
    Lampa {}                  # standardní lampička uprostřed
    Transform {               # růžová lampička vlevo
      translation -.25 0 -0.1
      children Lampa { barvaLampy 1 0.4 0.5  barvaSvetla 1 0.3 0.3
                       texturaPodstavce "" }
    }
    Transform {               # dřevěná lampička vpravo
      translation .25 0 0
      children Lampa { barvaLampy 0.3 0.2 0  barvaSvetla 1 1 0.2
                       texturaPodstavce "drevo.gif" }
    }
  ]
}
```

Program P-4-7: Trojí použití prototypu lampičky

5 Události hýbou světem

Virtuální světy, které jsme se naučili navrhovat v předchozích kapitolách, mají statický charakter. Jedinou dosud uvedenou interakcí je aktivace potomků uzlu **Anchor**, po níž následuje přesun na nové stanoviště nebo teleportace do dalšího světa. Abychom dokázali změnit statické světy na dynamické, musíme obohatit škálu našich prostředků o dva další prvky:

1. dynamické uzly, které reagují na chování uživatele, resp. avatara, a vysílají informace v podobě tzv. *událostí* (angl. *event*),
2. mechanismus předávání událostí mezi uzly – konstrukce **ROUTE**.

Událost je základním prostředkem, který umožňuje „rozhýbání“ statických světů. Můžeme si ji představit jako datový záznam, který je předáván mezi uzly v okamžiku, kdy z nějakého důvodu dojde ke změně hodnoty parametru uzlu. O uzlu, v němž událost na základě takového podnětu vznikne, říkáme, že *vyslal událost*. Podobně uzel, k němuž byla data o události dopravena k dalšímu zpracování, *událost přijal* a přijatá data uložil do svého parametru.

Abychom si dokázali tyto nové pojmy prakticky představit, použijeme jednoduchý příklad. Do pomyslné místnosti vložíme dynamický uzel, který detekuje přítomnost avatara. Uzel se jmenuje **ProximitySensor** a podrobněji se s ním seznámíme v kapitole 5.5. Jakmile avatar vstoupí do místnosti, tento dynamický uzel vyšle události dvěma dalším uzlům – zdroji světla (**SpotLight**) a uzlu specifikujícímu zvukový záznam (**AudioClip**). Původně vypnutý zdroj světla se rozzáří, protože jeho parametr **on** se nastaví na hodnotu **TRUE**, zvukový záznam spustí přehrávání hudby, neboť do jeho parametru **startTime** se zapíše čas, ve kterém avatar vstoupil do místnosti. Jakmile avatar místnost opustí, uzel **ProximitySensor** vyšle události, které způsobí vypnutí světla a zastavení přehrávání zvuku. Celá tato dynamická akce je zapsána v programu P-5-1.

```
Group {
  children [
    … # popis místnosti

    DEF SVETLO SpotLight { on FALSE }   # zpočátku světlo nesvítí
    Sound {
      source DEF HUDBA AudioClip { … }
    }
    DEF AKCE ProximitySensor { … }
  ]
}

ROUTE AKCE.isActive  TO SVETLO.on
ROUTE AKCE.enterTime TO HUDBA.startTime
ROUTE AKCE.exitTime  TO HUDBA.stopTime
```

Program P-5-1: Předávání událostí mezi parametry uzlů.

V programu P-5-1 je uveden nový příkaz **ROUTE … TO …**, který slouží k propojení vysílacích a přijímacích uzlů, lépe řečeno jejich parametrů. Právě ty parametry, které jsou takto propojeny, si mohou navzájem předávat události. Uzly, mezi jejichž parametry se události předávají, musejí být nejprve pojmenovány příkazem **DEF**. Do příkazu **ROUTE** se pak vždy zapisují dvojice, tvořené individuálním jménem uzlu a jeho parametru, spojené tečkou. První dvojice (hned za slovem **ROUTE**) představuje místo, odkud je událost vysílána, druhá dvojice (za slovem **TO**) označuje místo přijetí události.

Uzel **ProximitySensor**, nazvaný **AKCE**, vysílá v příkladu dvě události při vstupu avatara do určité oblasti a dvě při avatarově odchodu. Všimněme si, že propojení ke světlu je v tomto případě použito jak k jeho zapnutí, tak k vypnutí, neboť parametr **isActive** uzlu **ProximitySensor** mění svoji hodnotu, tj. vysílá událost, právě ve dvou okamžicích – při avatarově příchodu (**enterTime**) a odchodu (**exitTime**).

Z příkladu je dále vidět, že události obsahují data různých typů. Pro zapnutí a vypnutí světla je třeba přenášet údaje **TRUE** a **FALSE**, tedy typu **SFBool**. K přehrávání zvuku je nutno dodávat časové údaje, tj. data typu **SFTime**. Uzel, který přijímá událost, proto smí zapisovat dodávané údaje pouze do těch parametrů, jejichž datový typ je shodný s datovým typem konkrétní události.

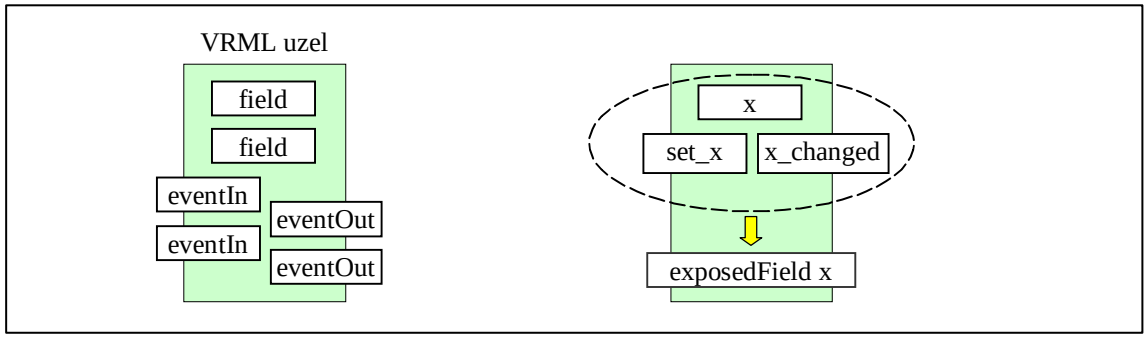
5.1 Co jsme zatajili

Až dosud se zdálo, že každý parametr uzlu je plně popsán svým datovým typem (viz kap. **4.1**) a iniciální hodnotou. Alespoň pro práci se statickými světy nám tyto údaje postačovaly. V okamžiku, kdy začínáme pracovat s dynamickými událostmi, musíme parametry zkoumat i z hlediska jejich schopnosti přijímat a vysílat události. Hodnoty některých parametrů totiž nemohou být změněny zasláním události, naopak ty parametry, jejichž úkolem je pouze události přijímat či vysílat, nemají žádnou iniciální hodnotu.

Parametry jsou tedy kromě svého datového typu dále charakterizovány svým vztahem k událostem. Existují tři základní třídy parametrů:

`field`	*statická* veličina s iniciální hodnotou. Lze ji změnit pouze zapsáním nové hodnoty ve VRML souboru, nikoliv však dynamicky.
`eventIn`	parametr schopný *přijmout událost* daného datového typu. Hodnota parametru je změněna přijetím události.
`eventOut`	parametr schopný *vyslat událost* v okamžiku, kdy dojde ke změně jeho hodnoty. Tato změna je nejčastěji vyvolána chováním avatara nebo je důsledkem zpracování jiných událostí.

Zobrazíme-li symbolicky jeden uzel VRML jako obdélník na obrázku 5-1 vlevo, statické parametry třídy **`field`** jsou umístěny zcela uvnitř, nedostupné pro dynamické akce. Parametry **`eventIn`** pro příjem událostí jsou posunuty doleva, parametry **`eventOut`** pro vyslání událostí doprava. Tuto symboliku budeme používat i v následujících částech textu.



Obrázek 5-1: Třídy parametrů VRML uzlů

U mnoha uzlů je žádoucí, aby nějaký parametr měl nejen svoji iniciální hodnotu, ale aby současně dokázal přijmout událost, která jeho hodnotu změní a aby tuto změnu vyslal v podobě události dalším uzlům. Proto byla zavedena ještě čtvrtá, pomocná třída parametrů, nazvaná **`exposedField`**. Každý parametr této třídy v sobě zahrnuje schopnosti všech tří základních tříd, jak symbolicky ukazuje obrázek 5-1 vpravo. Na obrázcích budeme takové parametry kreslit do širšího rámečku, přesahujícího zleva i zprava symbol uzlu.

`exposedField`	parametr s iniciální hodnotou, který je schopný přijmout události měnící jeho hodnotu a také po změně své hodnoty události vysílat.

Zajímavé je, jakým způsobem se zachází se jmény parametrů. Pokud se parametr třídy **`exposedField`** jmenuje například **`poloha`**, pak ve skutečnosti představuje tři parametry základních tříd, z nichž každý má svoje pojmenování a lze s ním i samostatně pracovat:

1. **`field`** **`poloha`** <iniciální hodnota>
2. **`eventIn`** **`set_poloha`**
3. **`eventOut`** **`poloha_changed`**

Předpona "**`set_`**", resp. přípona "**`_changed`**", je často používána i ve jménech parametrů, které události pouze přijímají, resp. vysílají. Používání této dohody usnadní návrháři práci, protože podle jména lze snadno odhadnout charakter parametru.

V kapitolách 2, 3 a 4 jsme tedy před čtenářem tajili, že parametry spadají do výše uvedených čtyř tříd a předkládali jsme mu je bez detailního rozlišení. Ve většině případů šlo o parametry třídy **field** a **exposedField**. Detailní popis parametrů s uvedením jejich tříd nalezne čtenář v kapitole 9. Zde si pouze řekneme, že statické parametry třídy **field** se vyskytují v menším počtu. Jsou to parametry všech čtyř základních těles (kvádr, koule, kužel a válec) a uzlů **FontStyle** a **WorldInfo**. Všechny ostatní uzly obsahují ve velkém počtu parametry třídy **exposedField** s tím, že jsou někdy doplněny několika málo parametry třídy **field**.

Parametry třídy **eventIn** a **eventOut** jsou vlastní především dynamickým uzlům a proto se s nimi v dosud uvedených (statických) uzlech setkáme jen výjimečně. Následující přehled uvádí ty statické uzly, které zpracovávají události speciálními parametry určenými pouze k vysílání či pouze k přijímání událostí:

parametr třídy **eventIn**	činnost	výskyt v uzlech
SFBool set_bind	aktivování uzlů, z nichž pouze jeden z každého druhu může být aktivní	Background, Fog, NavigationInfo, Viewpoint
MFNode addChildren, **MFNode removeChildren**	přidání a odebrání seznamu potomků skupinovým uzlům	Anchor, Billboard, Group, Transform

parametr třídy **eventOut**	činnost	výskyt v uzlech
SFBool isBound	informace o tom, že daný uzel byl aktivován nebo deaktivován	Background, Fog, NavigationInfo, Viewpoint
SFBool isActive	informace o tom, že nahrávka je přehrávána nebo ukončena	AudioClip, MovieTexture
SFTime duration_changed	informace o délce nahrávky po jejím načtení ze souboru	

Tabulka T-5-1: Přehled parametrů statických uzlů, používaných výhradně pro vysílání a příjem událostí

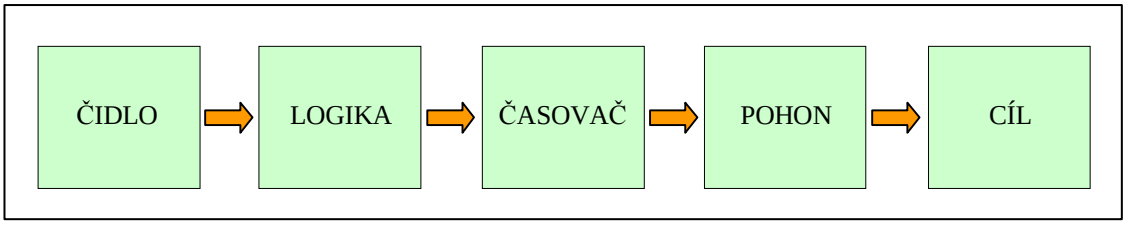
Se znalostmi o třídách uzlů můžeme nyní přehledně shrnout vše, co víme o zpracování událostí:

- Událost je vysílána z parametru **eventOut** nebo **exposedField**, jehož datový typ je shodný s datovým typem události. K vyslání dojde v okamžiku, kdy se změní hodnota tohoto parametru (např. parametr **isActive** v programu P-5-1).
- Událost je přijímána parametrem **eventIn** nebo **exposedField**, jehož datový typ je shodný s přenášeným typem události. Příjem události znamená zapsání nového údaje do daného parametru (např. zapsání času do parametrů **startTime** a **stopTime** v programu P-5-1)
- Uzly, které přijímají a vysílají události, musejí být pojmenovány příkazem **DEF**, případně **USE**.
- Jedna událost může být zaslána několika uzlům, resp. parametrům (takto lze například rozsvítit několik světel naráz).
- Událost je vnitřně tvořena dvěma datovými položkami – přenášeným *údajem*, jehož datový typ je jedním z datových typů uvedených v kapitole **4.1**, a *časem* svého vzniku (typu **SFTime**). Datový typ přenášeného údaje definuje celkový datový typ události. Čas vzniku události není parametrům uzlů obecně přístupný, avšak lze jej zpracovávat ve speciálním uzlu **Script** (viz kap. 6).

Příkaz **ROUTE** zasílá události mezi pojmenovanými uzly. Velmi důležitá je skutečnost, že pokud přiřadíme příkazem **DEF** jméno nějakému uzlu a následujícími příkazy **USE** vytvoříme nové uzly, jediný příkaz **ROUTE** zajistí zaslání události všem těmto stejně pojmenovaným uzlům. V příkladu z předchozí kapitoly bychom tedy mohli definovat opakovaným zápisem **USE SVETLO** (s vhodnými transformacemi) několik světel a rozsvítit je jedinou událostí.

Bez nadsázky lze říci, že události jsou tím, co rozhýbe statický svět, dodá mu dynamiku, animace, schopnost reakce na avatarovo jednání. Abychom se snáze orientovali v tom, jak s událostmi zacházet, podíváme se na obrázek 5-2. Na něm je schematicky naznačen průběh libovolné dynamické akce. Toto schéma je velmi obecné a neznamená, že každá akce či animace musí nutně obsahovala všechny uvedené prvky nebo přesně dodržovat

jejich pořadí. Mnoho dynamických akcí je tvořeno jen třemi či čtyřmi prvky. Přesto je vhodné se s nimi nejprve detailněji seznámit a při návrhu interaktivních virtuálních světů z nich pak vybírat menší skupiny, případně je podle potřeby kombinovat.



Obrázek 5-2: Logické schéma obecné dynamické akce

Na obrázku 5-2 je pět uzlů, které si postupně předávají události. Ty mohou být různého typu, při jejich cestě dynamickým řetězcem dokonce často dochází k několika změnám typu předávané události.

1. Na počátku každé aktivity musí stát prvek, který detekuje příčinu dynamické akce, ať je to vstup avatara do místnosti, aktivace kurzoru nad obrazem virtuálního objektu nebo jiná událost. Takovému prvku můžeme říkat ČIDLO. Ve VRML máme k dispozici celou řadu rozličných čidel.

2. Za čidlem často následuje prvek, který rozhoduje o tom, zda byly splněny veškeré podmínky pro skutečné zahájení dynamické akce. Příkladem je virtuální video přehrávač, který nezahájí činnost dříve, než jej avatar zapne do elektrické sítě. Jiným příkladem je otevření dveří trezoru až po nastavení správné číselné kombinace. Prvek, zvaný LOGIKA, obsahuje většinou lokální paměť realizovanou uzlem třídy **`field`**. Ke složitějšímu vyhodnocování podmínek slouží uzel **`Script`**, který bude důkladně popsán v kap. 6.

3. ČASOVAČ je zodpovědný za časový průběh akce. Nemusí se starat pouze o správný čas zahájení a ukončení, ale může vhodným způsobem měnit dynamiku děje. Otevírané dveře se například mohou nejprve prudce „rozletět“ a poté krátce dobrzdit v místě úplného otevření. Špatně namířená kulička na kulečníku naopak bude svoji iniciální vysokou rychlost snižovat postupně až do úplného zastavení.

4. Viditelná dynamická akce je vždy realizována postupnou změnou hodnot některého parametru, ať již jde o polohu, rozměr či natočení objektu, změnu jeho barvy, rozsvícení světla apod. Prvek, nazvaný POHON, dokáže na základě předem definovaných počátečních a koncových hodnot průběžně vypočítávat nové hodnoty, a to v souladu s dynamikou dodávanou časovačem.

5. Na konci řetězce událostí je CÍL, na němž je dynamická akce viditelná. Bývá to běžný statický uzel, do jehož parametrů (nejčastěji třídy **`exposedField`**) se zasílají hodnoty způsobující změnu jeho vzhledu. Jednoduše lze říci, že všechny čtyři předchozí prvky sloužily k tomu, aby se s cílovým prvkem „něco stalo“, a to v souladu s představami tvůrce dynamického virtuálního světa.

Na závěr této části se ještě zaměříme na časové souvislosti. Uvedli jsme, že každá událost s sebou nese údaj o čase, ve kterém vznikla. Tomuto údaji se říká *časové razítko*. Čas je v něm zapsán *absolutně*, a to počtem sekund, které uplynuly od půlnoci 1. ledna 1970. Čas přitom může obsahovat i necelá čísla a popisovat tak děje v rozsahu milisekund apod. Přestože je čas udáván absolutně, ve virtuálních světech se pracuje s časem *relativním*, tedy vztaženým k nějaké události, typicky k aktivitě avatara. Nemá valný smysl požadovat, aby nějaký děj začal přesně v 9:25, když nevíme, zda v tu dobu vůbec někdo navštíví náš virtuální svět. Lepší je děj zahájit teprve poté, kdy avatar vstoupí do určitého prostoru.

TIP: Rozeslat tutéž událost do několika parametrů je zcela regulérní. Takové *rozvětvení* (angl. *fan-out*) vede na několik dynamických akcí v různých částech světa. Naopak *sloučení* (angl. *fan-in*) více událostí (například ukončení několika dynamických akcí) do jednoho parametru vede na nejednoznačné výsledné chování tohoto parametru.

Časová razítka mají velký význam při větvení událostí, tj. v takových situacích, kdy jedna událost vyvolá celý řetěz následných událostí. Aby byl co nejvíce potlačen vliv různých časových prodlev způsobených zatížením počítače, časové razítko původní události je beze změny okopírováno do všech bezprostředně následujících událostí. Zapíše-li tedy například jedna událost novou hodnotu do parametru **set_poloha**, jenž je třídy **exposedField**, je vygenerována další událost vyslaná z parametru s formálním názvem **poloha_changed**, jejíž časové razítko je totožné s razítkem původní události, ač mezitím jistě uběhl čas nenulové délky.

Tímto způsobem je zajištěno, že možný programátorův omyl při propojování událostí nezpůsobí zacyklení. Pokud by řetěz událostí vedl k tomu, že bude změněna hodnota parametru, který tento řetězec vyvolal, zpracování událostí bude v tomto parametru zastaveno.

Při znalosti pravidel zpracování událostí a jejich časových razítek dokážeme elegantně spojit dva samostatné uzly **A** a **B** tak, aby aktivace libovolného z nich způsobila okamžitou aktivaci druhého. Nemusíme přitom vědět, který ze dvou uzlů je původcem děje – důležité je, že na aktivaci zareagují oba současně. V následující ukázce předpokládáme, že parametry **xxx** v uzlu **A** a **yyy** v uzlu **B** jsou třídy **exposedField**:

```
ROUTE A.xxx_changed TO B.set_yyy

ROUTE B.yyy_changed TO A.set_xxx
```

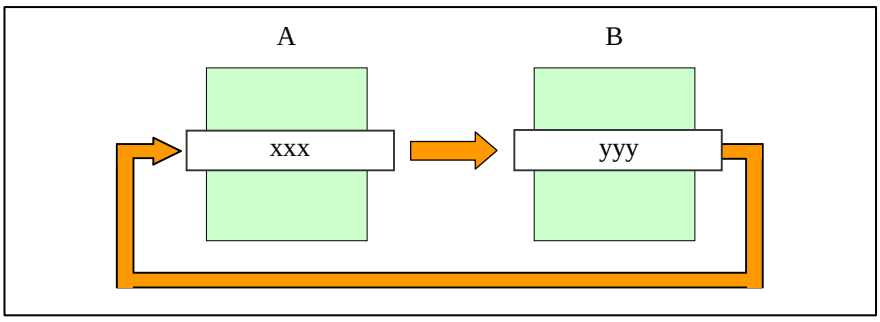
Uvedený zápis skutečně připomíná nekonečný cyklus. Ve skutečnosti vždy dojde k zaslání a tedy i zpracování právě dvou událostí tak, jak jsou zapsány výše. Případné cyklické zapisování do parametru **xxx** nebo **yyy** je zablokováno právě časovým razítkem, které putuje s událostmi. Událost **xxx_changed** si nemůže sama sobě způsobit (byť zprostředkovaně) nastavení **set_xxx**.

TIP: U parametrů třídy exposedField není třeba v příkazu ROUTE uvádět jména s předponou set_ a příponou _changed, protože tyto přípony jsou k nim doplněny automaticky podle toho, zda parametr událost vysílá či přijímá. Jejich plné uvedení však zvyšuje čitelnost souboru.

Protože parametry **xxx** a **yyy** v diskutovaném příkladu jsou třídy **exposedField**, můžeme propojení uzlů zapsat i v takto zkrácené podobě:

```
ROUTE A.xxx TO B.yyy

ROUTE B.yyy TO A.xxx
```

Schéma tohoto propojení je na obrázku 5-3.



Obrázek 5-3: Události, které synchronizují hodnoty dvou parametrů

5.2 Neviditelná ruka avatarova
(CylinderSensor, PlaneSensor, SphereSensor, TouchSensor)

Zatímco v předchozí části jsme se seznámili s dynamickými akcemi z teoretického hlediska, nyní ukážeme jejich praktické použití. Často se přitom budeme odvolávat na obrázek 5-2, neboť na něm uvedené schéma je dobrým pomocníkem.

První skupinu dynamických uzlů budeme nazývat *manipulátory*. Umožňují návštěvníkovi virtuálního světa změnit polohu vybraného objektu nebo skupiny objektů. Lze říci, že avatar s jejich pomocí dokáže svojí virtuální rukou uchopit objekt a změnit jeho vlastnosti popsatelné transformacemi v prostoru. V okamžiku, kdy je na obrazovce přesunut kurzor nad objekt, který je pod vlivem manipulátoru, lze aktivováním kurzoru (stisknutím tlačítka myši) s objektem manipulovat – měnit jeho polohu, orientaci či měřítko. Prohlížeče usnadňují práci s takovými objekty tím, že nad nimi změní tvar kurzoru a upozorní návštěvníka na možnost manipulace.

Ve smyslu obrázku 5-2 jsou manipulátory takovými prvky, které v sobě zahrnují současně jak funkci *čidla* (když uživatel aktivuje kurzor nad objektem), tak *pohonu* (generují nové hodnoty prostorových transformací). Prvek *logiky* v tomto případě není uplatněn, protože manipulaci nelze podmiňovat dalšími podmínkami. Také *časování* je jednoduché – akce probíhá tak dlouho, dokud návštěvník drží tlačítko myši stisknuté. *Cílem* dynamické akce jsou většinou transformační parametry v uzlu **`Transform`**.

Existují celkem tři manipulátory:

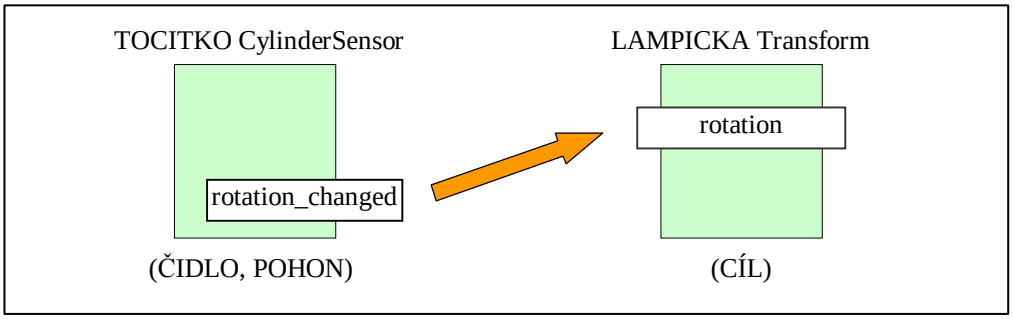
1. **`SphereSensor`** (kulový manipulátor)
   převádí pohyb kurzoru na osu a úhel otočení dané pohybem po povrchu pomyslné koule.
2. **`PlaneSensor`** (rovinný manipulátor)
   převádí pohyb kurzoru na posun po povrchu pomyslné rovinné desky kolmé na osu *z*.
3. **`CylinderSensor`** (válcový manipulátor)
   převádí pohyb kurzoru na úhel otočení daného pohybem po povrchu pomyslného válce, jehož osa je rovnoběžná s osou *y*.

Manipulátory se navzájem liší nejen ve způsobu převodu pohybu aktivovaného kurzoru na výpočet nových polohových vlastností objektů, ale i tím, jaká výsledná data poskytují. Všechny tři manipulátory však mají několik společných vlastnosti, resp. parametrů, které jsou uvedeny v tabulce T-5-2. Všimněme si, že od této chvíle jsou v přehledu parametrů uváděny také jejich vztahy k událostem. Parametry tříd **`eventIn`** a **`eventOut`** nemají v tabulkách uvedeny iniciální hodnoty.

úplná specifikace parametru		jméno parametru	iniciální hodnota		význam
`exposedField`	`SFBool`	`enabled`	`TRUE`		povolení práce manipulátoru
	`SFRotation`		`0 1 0 0`		základní hodnota, která se vždy přičte k nově vypočítané hodnotě
`exposedField`	`SFVec3f`	`offset`	`0 0 0`		
	`SFFloat`		`0`		
`exposedField`	`SFBool`	`autoOffset`	`TRUE`		povolení automatické aktualizace parametru **`offset`** po skončení činnosti manipulátoru
`eventOut`	`SFBool`	`isActive`		-	zahájení a ukončení činnosti manipulátoru
`eventOut`	`SFVec3f`	`trackPoint_changed`		-	měnící se poloha bodu na pomyslném objektu (desce, kouli, válci), na který ukazuje kurzor při práci s manipulátorem
	`SFRotation`	`rotation_changed`			průběžně vypočítávaná hodnota odvozená z pohybu kurzoru po povrchu pomyslného objektu
`eventOut`				-	
	`SFVec3f`	`translation_changed`			

Tabulka T-5-2: Společné a funkčně podobné parametry manipulátorů

Dříve, než podrobně vysvětlíme funkce jednotlivých parametrů, podíváme se na příklad použití válcového manipulátoru. Vytvoříme jednoduchý svět, v němž bude stát malý stolek a na něm lampička. Avatarovi dovolíme, aby lampičkou otáčel podle její osy tehdy, pokud nad jejím obrázkem aktivuje kurzor. K tomu použijeme válcový manipulátor nazvaný **TOCITKO**. Obrázek 5-4 ukazuje, že k zajištění takto jednoduché manipulace vystačíme pouze s jedinou událostí. Ta je typu **SFRotation** a je předávána do uzlu **Transform**. Potomkem tohoto uzlu je pochopitelně ona lampička.



Obrázek 5-4: Vyslání události manipulátorem

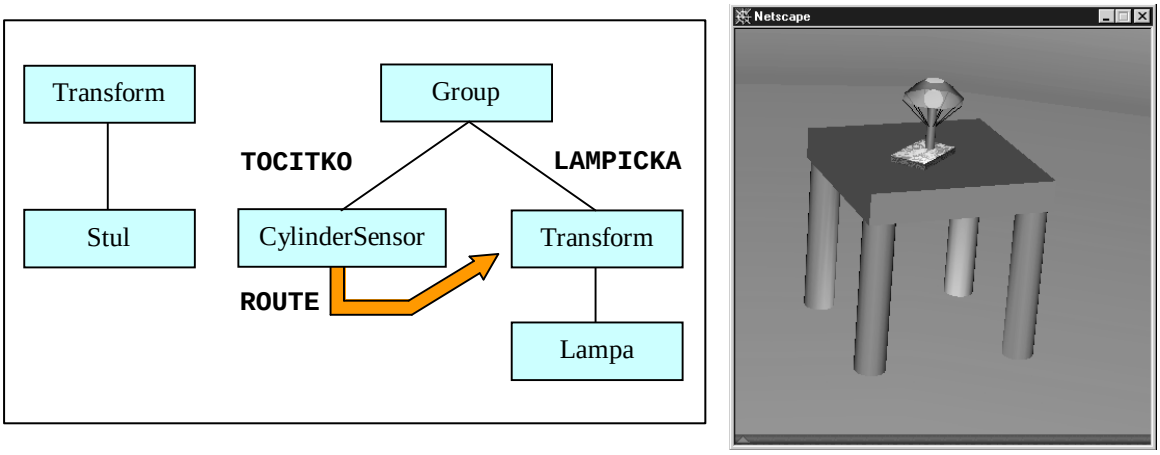
Ukázka v programu P-5-2 přesně odpovídá schématu na obrázku 5-4. Uzly, které se účastní zpracování události, jsou pojmenovány (**TOCITKO**, **LAMPICKA**) a v programu jsou zvýrazněny podtržením. Pro definice stolu i lampy jsou použity prototypy z kapitoly 4. Stolek je zmenšen na polovinu, takže jeho horní deska je ve výši 60 cm. Do této výšky je také umístěna lampa.

Válcový senzor vysílá události z parametru **rotation_changed** (třídy **eventOut**) do parametru **rotation** (třídy **exposedField**). Oba parametry musí být stejného datového typu – v tomto případě jde o rotační koeficienty (**SFRotation**).

```
#VRML V2.0 utf8
EXTERNPROTO Stul  [ … ] " ../knihovny/nabytek.wrl#MujStolek"
EXTERNPROTO Lampa [ … ] " ../knihovny/protolampa.wrl#Lampa-V"

Transform { scale .5 .5 .5
            children Stul {}
}

Group {
  children [
     DEF TOCITKO  CylinderSensor {}
     DEF LAMPICKA Transform {
       translation 0 0.6 0
       children Lampa {}
     }
  ]
}

ROUTE TOCITKO.rotation_changed TO LAMPICKA.set_rotation
```

Program P-5-2: Ovládání natočení lampičky pomocí válcového manipulátoru

Umístění uzlu **Lampa** v souboru VRML si zaslouží naši pozornost. Aby se mohl obrázek lampy stát místem na obrazovce, na který reaguje kurzor (lépe řečeno manipulátor), je třeba zapsat uzel **Lampa** do správné části VRML stromu. Platí totiž, že manipulátor reaguje na aktivaci těch objektů, kteří jsou jeho *sourozenci* ve stromové hierarchii. Současně chceme, aby se měnila transformace lampy. Z toho plyne požadavek, aby lampa byla potomkem transformačního uzlu. Výsledkem je stromová struktura v programu P-5-2, ve které mají válcový manipulátor a transformační uzel společného rodiče. Tvar této struktury přibližuje obrázek 5-5 vlevo. Vpravo pak vidíme stolek, na němž je lampa již pootočena.



Obrázek 5-5: Stromová struktura uzlů s manipulátorem (vlevo) a výsledný svět (vpravo)

V uvedeném příkladu byl použit pouze jediný z parametrů manipulátoru. Tabulka T-5-2 uvádí celou řadu dalších. Nejjednodušším z nich je parametr **`enabled`**, který dovoluje nebo naopak zakazuje činnost konkrétního manipulátoru. Vzhledem k tomu, že jde o parametr třídy **`exposedField`**, jeho nastavování probíhá v rámci zasílání událostí a tedy takových dynamických akcí, ve kterých se objevují události typu **`SFBool`**. O nich se dozvíme v dalších kapitolách.

Všechny tři manipulátory obsahují parametr **`offset`**. Ten má jediný význam – uchovává počáteční hodnotu, která je připočítávána k nově vyhodnoceným transformačním údajům dříve, než jsou z manipulátoru odeslány. Je zřejmé, že každý typ manipulátoru bude mít **`offset`** jiného datového typu. Kulový manipulátor generuje koeficienty otočení a proto je v jeho parametru **`offset`** uložena hodnota typu **`SFRotation`**. Rovinný manipulátor zasílá událost s koeficienty pro posunutí a jeho **`offset`** je typu **`SFVec3f`**. Mírně odlišná je situace u válcového manipulátoru. Ten sice generuje koeficienty otočení podobně jako kulový manipulátor, avšak vždy jde o otáčení kolem osy *y*. Do parametru **`offset`** stačí v tomto případě ukládat namísto všech koeficientů otočení pouze počáteční úhel, tedy hodnotu typu **`SFFloat`**.

Pokud by hodnota uložená v parametru **`offset`** zůstávala neměnná, každá nová manipulace s objektem by znamenala, že řízený objekt na začátku manipulace „poskočí“ do iniciální polohy a teprve poté jsou na něj aplikovány další transformace. Abychom tomuto nezvyklému chování zabránili, je třeba po ukončení jedné manipulace uložit do parametru **`offset`** naposledy dosaženou transformační hodnotu. Tato aktualizace může být prováděna automaticky, a to podle nastavení parametru **`autoOffset`**. Má-li hodnotu **TRUE**, údaje v parametru **`offset`** se aktualizují po každém ukončení manipulace (je přitom vyslána událost z parametru **`offset_changed`**), při hodnotě **FALSE** zůstávají neměnné.

Manipulátory mohou vysílat několik různých typů událostí. Parametr **`isActive`** třídy **`eventOut`** změní svoji hodnotu, tj. vyšle událost vždy při zahájení a při ukončení manipulace. Vysílaná hodnota je typu **`SFBool`** a lze ji využít například k zapnutí a vypnutí pomocného zdroje světla po dobu manipulování s objektem. Stačí jen zaslat tuto událost do parametru **`on`** patřičného světelného zdroje.

Nejdůležitějšími událostmi, které manipulátor vysílá, jsou ty, které nesou údaje použitelné pro transformaci dalších objektů. Jak již bylo uvedeno, každý z manipulátorů vysílá událost odlišného typu. Rovinný manipulátor vysílá trojici souřadnic z parametru **`translation_changed`**, zbylé dva manipulátory generují události typu **`SFRotation`** vysílané z parametru **`rotation_changed`**. Jejich vysílání probíhá průběžně po celou dobu, po kterou avatarova ruka pracuje s objektem citlivým na manipulaci.

Posledním parametrem, který je společný všem třem manipulátorům, je **`trackPoint_changed`**. Je třídy **`eventOut`** a průběžně generuje události obsahující měnící se polohu bodu na pomyslném prostorovém objektu, na který ukazuje avatarova ruka. Pomyslným objektem je právě koule, rovinná deska nebo válec podle toho, který typ manipulátoru je použit. Tento parametr není příliš často využíván. Poskytuje korektní výsledky pouze tehdy, pokud je kurzor přesně nad obrazem aktivovaného objektu. Kurzor ovšem může zůstat aktivní i po přesunu mimo objekt (když zůstalo stisknuté tlačítko myši). Tehdy zjevně nelze určit bod na pomyslném manipulačním objektu a parametr **`trackPoint_changed`** poskytuje údaje o pochybné vypovídací hodnotě.

Většinou je pouze přičítána aktuální souřadnice kurzoru k naposledy vyhodnocené souřadnici a je tedy generována posloupnost poloh v rovině namísto posloupnosti bodů na povrchu pomyslného manipulačního tělesa.

Jednotlivých manipulátorů si nyní všimneme podrobněji a zdůrazníme rozdíly mezi nimi:

Kulový manipulátor

- neobsahuje již žádné další parametry kromě těch, které byly uvedeny v tabulce T-5-2.

- Velikost pomyslné koule, po které se pohybuje avatarova ruka, je automaticky nastavena tak, aby koule obklopila geometrii sourozeneckých uzlů, tedy objektů citlivých na avatarovu ruku.

Rovinný manipulátor

- má navíc další dva parametry, které omezují pohyb na jinak nekonečně velké pomyslné manipulační desce umístěné v rovině *xy (z=0)*. Vzhledem k tomu, že jde o pohyb v rovině, parametry obsahují souřadnice pouze ve dvourozměrném prostoru:

úplná specifikace parametru	jméno parametru	iniciální hodnota	význam
exposedField SFVec2f	**minPosition**	**0 0**	dolní mezní poloha
exposedField SFVec2f	**maxPosition**	**-1 -1**	horní mezní poloha

  Je-li libovolná ze souřadnic **maxPosition** menší než odpovídající souřadnice **minPosition**, vypočítávaná posunutí nejsou omezena. To je také případ iniciálního nastavení. V opačném případě zasílá manipulátor události, obsahující příslušné mezní hodnoty, když dojde k posunutí kurzoru mimo stanovené meze.

- V případě rovnosti jedné ze souřadnic **maxPosition** a **minPosition** převádí manipulátor pohyb kurzoru na pohyb po úsečce. Z uzlu **PlaneSensor** se tak stává jakýsi „**LineSensor**“, tedy generátor poloh na vodorovné nebo svislé úsečce.

Válcový manipulátor

- Je nejsložitějším manipulátorem, protože musí zajišťovat pohyb avatarovy ruky jak po pomyslném rotačním plášti válce, tak po jeho podstavách. Obsahuje proto několik speciálních parametrů:

úplná specifikace parametru	jméno parametru	iniciální hodnota	význam
exposedField SFFloat	**diskAngle**	**0.262**	úhel, rozlišující mezi podstavou a pláštěm
exposedField SFFloat	**minAngle**	**0**	dolní mezní úhel otočení
exposedField SFFloat	**maxAngle**	**-1**	horní mezní úhel otočení

- Parametr **diskAngle** je důležitý právě v situaci, kdy na pomyslný válec hledíme z takového místa, odkud jsou dosažitelné současně plášť i podstava. Válcová manipulátor je totiž schopen převádět pohyb kurzoru pouze na *jedinou* z těchto geometricky odlišných částí povrchu válce. Je-li k avatarovi pomyslný válec natočen podstavou, otáčení cílového objektu docílíme krouživým pohybem kurzoru, jako kdybychom mleli mlýnkem na kávu. Je-li naopak válec natočen bokem, stačí s kurzorem přejíždět po obrazovce zleva doprava a manipulátor tento pohyb převádí na otáčení stejně, jako když otáčíme na nádraží válcovými plochami s vylepenými jízdními řády.
  Rozlišení, zda pro manipulaci vybrat pomyslnou podstavu nebo plášť, je dáno úhlem, který svírá osa válce vůči natažené avatarově ruce. Ruka je v tomto případě představována úsečkou spojující avatarovy oči s bodem na povrchu válce, na který avatar ukazuje. Pokud je úhel mezi rukou a osou válce menší než hodnota parametru **diskAngle**, pohyb kurzoru je chápán jako pohyb po kruhu. V opačném případě je pohyb převáděn na válcový plášť.

- Úhly **minAngle** a **maxAngle** mají podobný omezující význam jako mají parametry **minPosition** a **maxPosition** u rovinného manipulátoru. Definují interval, ve kterém se mohou vyskytovat

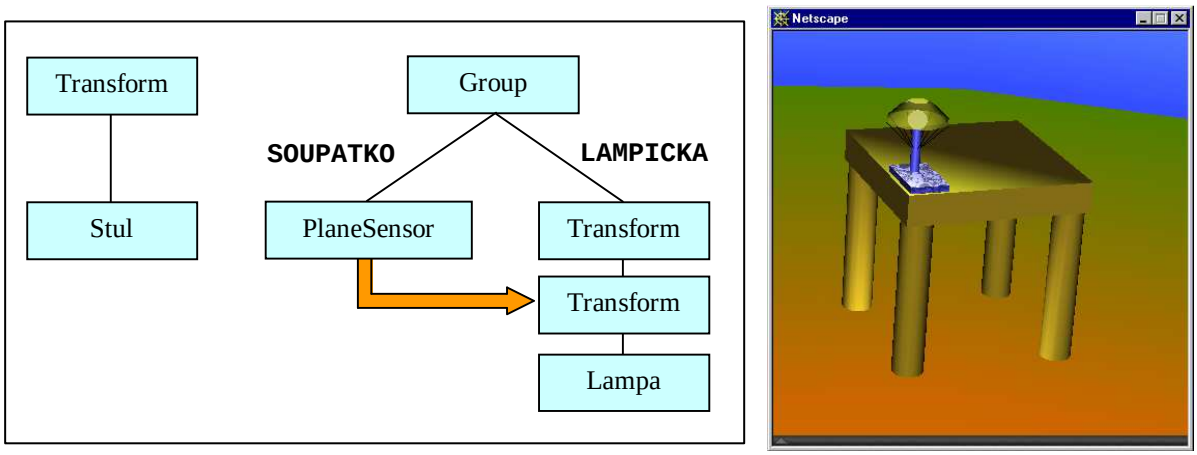
hodnoty vypočítávané válcovým manipulátorem. Tento interval může být tvořen až dvěma úplnými otáčkami, tedy celkovým rozsahem 4π. Pokud je hodnota **`maxAngle`** menší než **`minAngle`**, neplatí žádné omezení.

- Velikost pomyslného válce, po kterém se pohybuje avatarova ruka, je automaticky nastavena tak, aby válec obklopil geometrii sourozeneckých uzlů.

TIP: Při pohledu na svět shora dolů (půdorys) se špatně ovládá rovinný manipulátor, protože pracuje vždy s rovinnou *xy*. Podobně válcový manipulátor má válec umístěný vždy kolmo na rovinu *xz*. Řešením je zařazení manipulátorů jako potomků uzlu `Transform` s otočením o π/2 nebo ještě lépe jako potomků uzlu `Billboard`. Možnosti manipulace se výrazně zlepší, manipulátory přitom budou vypočítávat hodnoty, jako kdyby byly v původní poloze.

Autoři virtuálních světů se musejí vyrovnat se skutečností, že rovinný a válcový manipulátor poskytují vypočítané hodnoty vždy tak, jako kdyby pomyslné manipulační objekty byly v základní, neměnné poloze. To například znamená, že rovinný manipulátor dokáže měnit polohu cílových objektů jen v rovině *xy*. Snadno tedy jakýkoliv virtuální předmět zvedneme nebo přesuneme zleva doprava, ale již se nám nepodaří přitáhnout si jej k sobě ve směru osy *z*.

Ten, kdo je trochu zkušený v oblasti prostorové geometrie, si s problémem pevné orientace rovinného manipulátoru dokáže poradit. Použije k tomu několik transformačních uzlů, které převedou údaje poskytované rovinným manipulátorem na údaje v požadované cílové orientaci. Tento postup ukážeme na známém příkladu lampičky, která stojí na stole. Budeme chtít, aby avatar pohyboval lampičkou po desce stolu, tj. po ploše rovnoběžné s rovinou *xz*. Postup, který zvolíme, vypadá na první pohled, jako když se někdo drbá levou rukou za pravým uchem. Je to však jediný možný přístup pro takovouto situaci.



Obrázek 5-6: Stromová struktura uzlů s rovinným manipulátorem (vlevo) a výsledný svět (vpravo)

Vytvoříme stromovou strukturu, ve které budou nad sebou dva uzly **`Transform`** a pod nimi bude umístěn uzel **`Lampa`**, jak ukazuje obrázek 5-6 vlevo. Nejbližší rodič uzlu **`Lampa`** otočí dočasně lampičku do stejné roviny, ve které generuje rovinný manipulátor souřadnice, tedy do roviny *xy*. Na tento transformační uzel proto budou směřovány události z manipulátoru. Transformační uzel, který je umístěn ještě výše, provede otočení lampičky zpět do původní roviny. Výsledná kombinace transformací zajistí požadované chování lampičky.

```
#VRML V2.0 utf8
EXTERNPROTO Stul  [ … ] " ../knihovny/nabytek.wrl#MujStolek"
EXTERNPROTO Lampa [ … ] " ../knihovny/protolampa.wrl#Lampa-V"

Transform { scale .5 .5 .5  children Stul {} }

Group {
  children [
     DEF SOUPATKO PlaneSensor { maxPosition 0.46 0.38 }
     Transform {
       rotation 1 0 0 -1.57          # otočení zpět do roviny xz
       translation -0.23 0.6 0.15    # iniciální poloha lampy na stole
       children
         DEF LAMPICKA Transform {
           rotation 1 0 0 1.57       # dočasné otočení do roviny xy
           children Lampa {}
         }
      }
  ]
}

ROUTE SOUPATKO.translation_changed TO LAMPICKA.translation
```

Program P-5-3: Posouvání lampičky v horizontální rovině

V programu P-5-3, který obsahuje zápis výše uvedeného postupu, si také všimneme toho, jak omezit manévrovací prostor rovinného manipulátoru tak, aby lampička nemohl být posunuta mimo desku stolu. Nastavení parametru **`maxPosition`** omezuje pohyb lampičky na 46 cm v jednom a 38 cm v druhém směru. Původní délka a šířka desky stolku 120 cm jsou i po zmenšení na polovinu stále ještě dostatečně velké na to, aby posouvaná lampička desku neopustila.

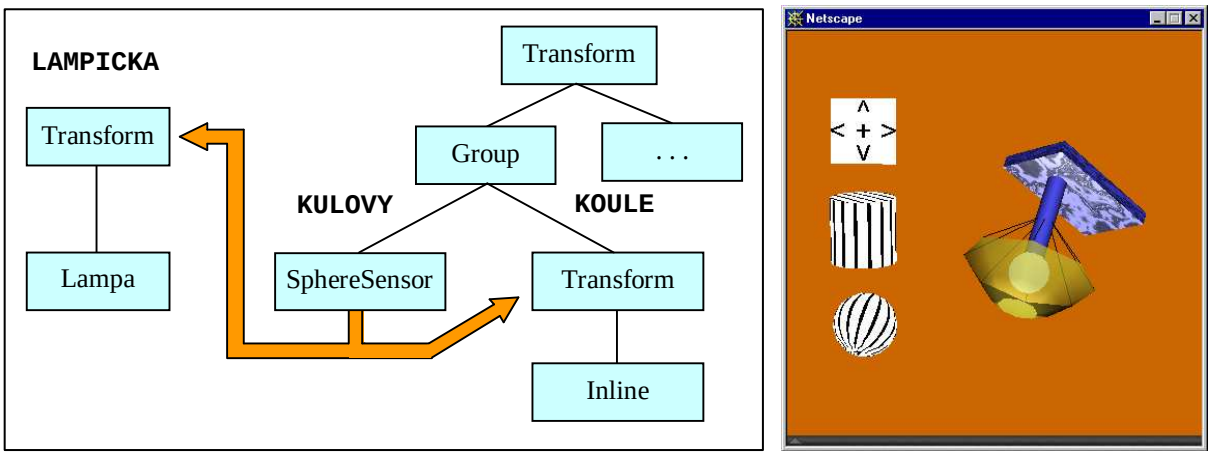
Další zajímavostí je pojmenování parametrů v příkazu **`ROUTE`**. Namísto úplného zápisu **`set_translation`** u uzlu **`LAMPICKA`** jsme použili zkrácenou variantu **`translation`**. U manipulátoru se jménem **`SOUPATKO`** je nutno použít celého jména parametru **`translation_changed`**. Nejde totiž o parametr třídy **`exposedField`**, nýbrž **`eventOut`**.

Na obrázku 5-6 vpravo si současně prohlédněme, jak světlo lampičky ovlivňuje okolní předměty. Dvě nohy stolku jsou výrazně osvětleny, ačkoliv ve skutečném světě by na ně světlo nemohlo dopadat – neprošlo by deskou stolu. Obrázek je tedy připomínkou toho, že ve virtuálním prostředí se předměty navzájem nezastiňují. Současně nás usvědčuje z mírného prohřešku proti základnímu stavebnicovému pravidlu – definovali jsme příliš velký dosah světla lampy. Zmenšením hodnoty parametru **`radius`** ve zdroji světla bychom tuto chybu napravili a nohy stolu by zůstaly v souladu se skutečností tmavé[9].

TIP: Zasílání událostí od manipulátorů k transformacím v jejich rodičovských uzlech vede při manipulaci ke zmatenému chování. Správné je propojit manipulátory se sourozenci, případně dále s uzly ve zcela jiných částech světa (tzv. dálkové ovládání).

Jedním manipulátorem lze ovlivňovat transformační parametry více uzlů, stejně tak lze měnit jeden transformační uzel více manipulátory. Program P-5-4 je ukázkou, ve které je lampička ovlivňována celkem třemi různými manipulátory (na obrázku 5-7 jsou vlevo od lampičky). Manipulátory jsou symbolizovány jednoduše kvádrem, válcem a koulí. Jakmile avatar začne pracovat s válcem nebo koulí, manipulátor začne vysílat události nejen k manipulovanému tělesu, ale i k lampičce. Pouze kvádr zůstává nehybný.

[9] Některé prohlížeče však hodnotu parametru **`radius`** ignorují, takže je nutno zajistit správný útlumu pečlivým nastavením parametru **`attenuation`**.



Obrázek 5-7: Při vysílání jedné události více uzlům můžeme manipulovat s několika objekty naráz. Schéma vlevo zachycuje ze stromové struktury jen tu část, která se týká kulového manipulátoru.

```
#VRML V2.0 utf8
EXTERNPROTO Stul  [ … ] " ../knihovny/nabytek.wrl#MujStolek"
EXTERNPROTO Lampa [ … ] " ../knihovny/protolampa.wrl#Lampa-V"

DEF LAMPICKA Transform { children Lampa {} }

Transform {
  translation –0.3 0.1 0
  scale 0.05 0.05 0.05
  children [
    Group { children [
          DEF ROVINNY PlaneSensor { maxPosition 0.7 0.3 }
          Billboard {
            axisOfRotation 0 0 0
            children DEF ROVINA Transform {
              translation 0 3 0
              children Inline { url "../knihovny/ovladac-deska.wrl" }
            }
          }
    ]}
    Group { children [
          DEF VALCOVY CylinderSensor { }
          DEF VALEC Transform {
              children Inline { url "../knihovny/ovladac-valec.wrl" }
            }
    ]}
    Group { children [
          DEF KULOVY SphereSensor { }
          DEF KOULE Transform {
              translation 0 -3 0
              children Inline { url "../knihovny/ovladac-koule.wrl" }
            }
    ]}
  ]
}

ROUTE ROVINNY.translation_changed TO LAMPICKA.translation
ROUTE VALCOVY.rotation_changed    TO LAMPICKA.rotation
ROUTE VALCOVY.rotation_changed    TO VALEC.rotation
ROUTE KULOVY.rotation_changed     TO LAMPICKA.rotation
ROUTE KULOVY.rotation_changed     TO KOULE.rotation
```

Program P-5-4: Ovládání lampičky s pomocí více manipulátorů

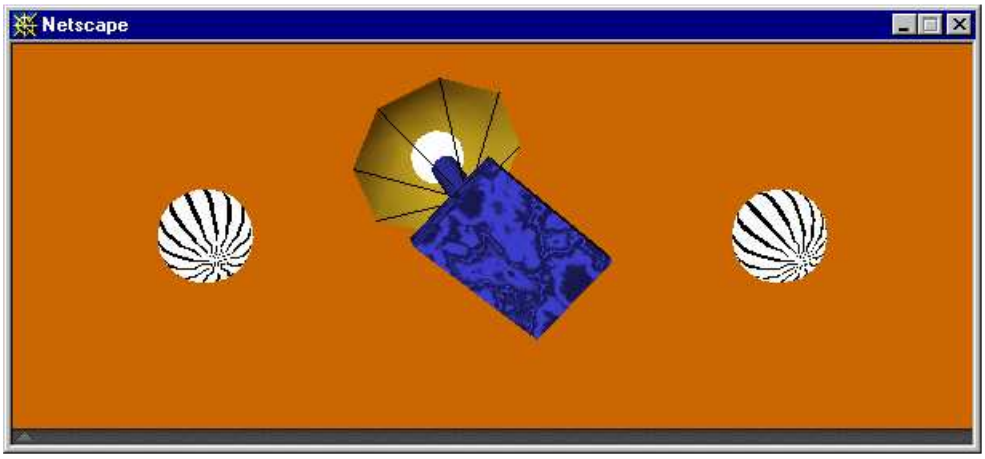
Na válcovém a kulovém manipulátoru je nanesena jednoduchá textura definovaná uzlem **PixelTexture**, která při manipulaci zvýrazňuje natočení ovládacího tělesa. Válec i koule jsou tedy propojeny s příslušným manipulátorem tak, jak ukazuje schéma na obrázku 5-7 vlevo. Bílá deska, která slouží jako další ovládací objekt, má na sobě symboly posunutí vytvořené uzlem **Text**. Tato deska není propojena s rovinným manipulátorem a zůstává proto při manipulaci na místě. Natáčí se ovšem stále k avatarovi, protože je potomkem uzlu **Billboard**. Snadno si dovedeme představit vylepšení spočívající v tom, že na statickou bílou desku umístíme jednu pohyblivou značku (propojenou

s rovinným manipulátorem), jejíž poloha v rámci desky bude zároveň ukazovat polohu řízeného tělesa, tj. lampičky.

Práce s manipulátory je první z mnoha možností, jak statický virtuální svět rozhýbat. Začátečníci, kteří se teprve seznamují s virtuálním prostředím, ocení, že mohou ovládat polohu vybraných předmětů a postupně se naučit, jak s pomocí myši pracovat v trojrozměrném prostoru. Manipulátory nacházejí své uplatnění zejména tam, kde chceme návštěvníkům ukázat nějaký výrobek ze všech stran – například ve virtuálním obchodním domě. Použití většího počtu manipulátorů dokonce dovolí modelovat skládanky tvořené více částmi.

Někdy je zapotřebí ovládat jedno těleso *z různých míst*, a to pomocí manipulátorů stejného typu. Představme si rozsáhlou místnost se čtyřmi vchody a s velkou uměleckou plastikou uprostřed. Vedle každého vchodu je umístěn jeden kulový manipulátor. Přejeme si, aby avatar mohl pomocí libovolného manipulátoru otáčet středovým objektem a prohlížet si jej ze všech stran. Naivním řešením je zavedení čtyř uzlů **SphereSensor** a použití čtyř příkazů **ROUTE** pro propojení manipulátorů s cílovým objektem. Tento způsob má jednu nevýhodu, která je způsobena existencí čtyř různých hodnot v parametrech **offset** příslušných manipulátorů. Když totiž začneme pracovat s jedním manipulátorem, jeho **offset** se po skončení aktualizuje tak, aby byl v souladu s natočením cílového objektu. Jakmile zahájíme práci s jiným manipulátorem, cílový objekt se skokem natočí podle parametru **offset** tohoto manipulátoru. Ovládání cílového objektu pak nejenže není plynulé, ale ani neodpovídá našemu původnímu záměru, protože manipulátory se chovají jako zcela samostatné prvky, nikoliv jako synchronizované dálkové ovladače. Zlepšení chování docílíme teprve zavedením dalších čtyř příkazů **ROUTE**, které zajistí vzájemnou aktualizaci hodnot **offset** u všech manipulátorů.

Správné řešení je mnohem snazší a přehlednější. Definujeme nejprve jediný manipulátor, který propojíme s cílovým objektem běžným způsobem. Zbylé tři manipulátory potom zapíšeme do souboru příkazem **USE** se jménem prvního manipulátoru. Kombinace příkazů **DEF** a **USE** v tomto případě zajistí společné, tedy synchronizované chování všech manipulátorů. Jednoduchost tohoto řešení je dokumentována obrázkem 5-8, na němž je použita dvojice samostatných, avšak příkazem **USE** správně synchronizovaných kulových manipulátorů. Program P-5-5 pak obsahuje příslušný zápis.



Obrázek 5-8: Dva kulové manipulátory, které synchronně ovládají jeden cíl

TIP: Příkaz USE sloužil ve statických světech k vytváření kopií. Teprve u dynamických uzlů vidíme skutečnou sílu jeho použití – sdílení vlastností stejně pojmenovaných uzlů či dokonce celých stromových struktur.

```
#VRML V2.0 utf8
EXTERNPROTO Stul  [ … ] " ../knihovny/nabytek.wrl#MujStolek"
EXTERNPROTO Lampa [ … ] " ../knihovny/protolampa.wrl#Lampa-V"

DEF LAMPICKA Transform { children Lampa {} }

Transform {                    # základní manipulátor (levý)
  translation -0.3 0 0
  children [
    DEF KULOVY SphereSensor { }
    DEF KOULE  Transform {
               scale 0.05 0.05 0.05
               children Inline { url "../knihovny/ovladac-koule.wrl" }
    }
  ]
}

Transform {                    # synchronizovaný manipulátor (pravý)
  translation 0.3 0 0
  children [
    USE KULOVY
    USE KOULE
  ]
}

ROUTE KULOVY.rotation_changed  TO LAMPICKA.rotation
ROUTE KULOVY.rotation_changed  TO KOULE.rotation
```

Program P-5-5: Ovládání lampičky dvojicí synchronizovaných manipulátorů

TouchSensor

Trojici manipulátorů doplňuje dynamický uzel **`TouchSensor`**, který je podobně citlivý na aktivitu avatarovy ruky, avšak negeneruje žádné transformace. Je schopen pracovat jen jako *detektor dotyku* – zjišťovat, zda avatarova ruka nad nějakým objektem pouze „přejela“ (přesunutí kurzoru) či zda se avatar pokusil objekt „uchopit“ (aktivace kurzoru). Jakmile avatar ukáže na objekt, který je sourozencem uzlu **`TouchSensor`**, je průběžně vysíláno poměrně velké množství událostí, týkajících se bodu, na který bylo ukázáno. Všechny parametry detektoru dotyku jsou uvedeny v tabulce T-5-3.

specifikace parametru		jméno	iniciální hodnota		význam
`exposedField`	**`SFBool`**	**`enabled`**	**`TRUE`**		povolení práce detektoru
`eventOut`	**`SFBool`**	**`isActive`**		-	změna stavu tlačítka pro aktivaci kurzoru
`eventOut`	**`SFBool`**	**`isOver`**		-	kurzor se dostal nad objekt nebo jej opustil
`eventOut`	**`SFTime`**	**`touchTime`**		-	čas uvolnění tlačítka pro aktivaci kurzoru
`eventOut`	**`SFVec3f`**	**`hitPoint_changed`**		-	souřadnice bodu na povrchu objektu, na který ukazuje kurzor
`eventOut`	**`SFVec3f`**	**`hitNormal_changed`**		-	normála v bodě, který odpovídá parametru **`hitPoint_changed`**
`eventOut`	**`SFVec2f`**	**`hitTexCoord_changed`**		-	souřadnice textury v bodě, který odpovídá parametru **`hitPoint_changed`**

Tabulka T-5-3: Parametry detektoru dotyku

Z tabulky je vidět podobnost s manipulátory. Parametr **enabled** povoluje či zakazuje práci detektoru, podobně parametr **isActive** vysílá události při stisku a uvolnění tlačítka myši.

S výjimkou prvního parametru jsou všechny ostatní parametry třídy **eventOut**. Znamená to, že detektor dokáže o daném objektu zjišťovat mnoho údajů a předávat je dále. Jakmile se avatarova ruka, tj. kurzor, přesune nad obraz objektu na obrazovce, je vyslána jedna událost z parametru **isOver** a zahájeno průběžné vysílání událostí z parametrů **hitPoint_changed**, **hitNormal_changed** a **hitTexCoord_changed**. Hodnoty těchto tří parametrů se mění tak, jak avatarova ruka přejíždí nad objektem. Vždy lze určit souřadnice bodu dotyku, normálový vektor k povrchu v tomto bodě a dvourozměrné souřadnice používané pro nanášení textury na tento bod. Tyto detailní geometrické informace jsou však určeny spíše pro náročnější aplikace.
V jednoduchých virtuálních světech se s jejich použitím setkáme jen vzácně.

Jakmile kurzor opustí objekt, průběžné vysílání výše uvedených geometrických dat se zastaví a vyšle se jediná událost z parametru **isOver** o hodnotě **FALSE**.

Všechny tyto děje probíhají bez ohledu na to, zda kurzor byl aktivován, tj. zda bylo stisknuto tlačítko myši. Aktivitu tlačítka totiž sleduje samostatný parametr **isActive**. Ten vyšle jednu událost o hodnotě **TRUE** právě tehdy, když se kurzor nachází nad objektem a dojde ke stisku tlačítka myši. Druhá událost je z tohoto parametru (hodnota **FALSE**) vyslána teprve tehdy, když je tlačítko uvolněno. Pokud mezitím nebyl kurzor přesunut mimo objekt, dojde i k vyslání času uvolnění tlačítka. Tento čas je zapsán do parametru **touchTime**. Uvedený postup je velmi benevolentní k avatarovi. Nerozhodný avatar totiž může stisknout tlačítko myši (dotknout se objektu), ale pokud se nechce pouštět do žádné další samostatné akce, přesune kurzor mimo objekt a tam teprve tlačítko myši uvolní. Detektor dotyku v tomto případě žádný čas do parametru **touchTime** nezapíše. Stejně tak se detektor zachová, pokud aktivovaný kurzor opustí objekt a poté se nad něj zase vrátí.

Program P-5-6 je ukázkou propojení detektoru dotyku se světelnými zdroji. Virtuální svět obsahuje jediné těleso – stříbřitou kouli. Ta může být osvětlena dvěma bodovými zdroji světla, které jsou zpočátku vypnuty (jejich parametr **on** nastavíme na hodnotu **FALSE**). Detektor dotyku zapsaný jako sourozenecký uzel stříbřité koule zapne jeden zdroj světla (například horní, červený) v okamžiku, kdy se avatarova ruka přiblíží ke kouli. Tuto akci zajistíme vysláním události z parametru **isOver**. Druhé světlo (dolní, zelené) se zapne teprve tehdy, když se avatar koule dotkne. To provedeme vysláním události z parametru **isActive**.

```
#VRML V2.0 utf8
EXTERNPROTO Stribro [] "../knihovny/kovy.wrl#Silver"

Group {
  children [
    DEF DOTYK TouchSensor {}
    DEF KOULE Shape {
      appearance Appearance { material Stribro {} }
      geometry Sphere {}
    }
  ]
}

DEF HORNI-SVETLO PointLight {location 0  2 1 on FALSE color 1 0 0}
DEF DOLNI-SVETLO PointLight {location 0 -2 1 on FALSE color 0 1 0}

ROUTE DOTYK.isOver   TO HORNI-SVETLO.on
ROUTE DOTYK.isActive TO DOLNI-SVETLO.on
```

Program P-5-6: Propojení zdrojů světla s detektorem dotyku

K tomuto příkladu neuvádíme příslušný obrázek virtuálního světa, protože spoléháme na čtenářovu představivost. Současně si může čtenář zapřemýšlet o tom, zda je možné, aby se na stříbřité kouli zeleně lesklo pouze světlo z dolního zdroje světla a červené světlo nesvítilo[10].

TIP: Detektor dotyku lze použít současně s manipulátorem. Oba uzly však musejí být ve stromové struktuře zapsány jako sourozenci. Jinak bude aktivní jen ten, který je ve stromu hlouběji, v lidské terminologii „ten nejmladší“.

Detektor dotyku **`TouchSensor`** je velmi často používaným dynamickým uzlem. Jsou využívány zejména ty jeho parametry, které vysílají události časového charakteru (**`touchTime`**) nebo booleovské hodnoty **`TRUE`** a **`FALSE`** (**`isActive`**, **`isOver`**). Z hlediska obecného schématu dynamických akcí na obrázku 5-2 sehrává detektor dotyku roli čidla.

Pro úplnost dodejme, že uzel **`Anchor`**, který jsme uvedli v části věnované statickým uzlům, je vlastně také detektorem dotyku. Má však značně omezené možnosti reakce - na dotyk reaguje jedinou aktivitou, totiž přenesením avatara do jiné části virtuálního prostoru. Pokud by se jednalo o teleportaci na stanoviště v témže světě, bylo by teoreticky možno uzel **`Anchor`** obejít a zajistit takovou akci vysláním události z detektoru dotyku do parametru **`set_bind`** daného stanoviště:

```
ROUTE TOUCHSENSOR.isActive TO VIEWPOINT.set_bind
```

Prakticky je však takové řešení nepoužitelné, protože propojení mezi uzly pojmenovanými v tomto případě **`TOUCHSENSOR`** a **`VIEWPOINT`** přenese i událost s hodnotou **`FALSE`** v okamžiku, kdy kurzor přestane být aktivní. Znamená to, že také stanoviště **`VIEWPOINT`** přestane být aktivní a prohlížeč vrátí avatara na předchozí stanoviště. Náhrada uzlu **`Anchor`** uzlem **`TouchSensor`** tedy není plnohodnotná. Představuje jen dočasnou změnu stanoviště, kterou lze využít např. jako lupu (stanoviště s jinými optickými parametry) po dobu aktivace kurzoru.

[10] Odpověď zní: ANO. Je-li kurzor aktivovaný nad koulí přesunut mimo kouli, červené světlo zhasne, ale zelené svítí do té doby, než je kurzor deaktivován uvolním tlačítka myši.

5.3 Jak plyne čas … (TimeSensor)

Významnou roli v každé interaktivní akci hraje čas. Jen málo dějů ve skutečném světě kolem nás probíhá naráz, skokovou změnou. Většina dynamických jevů má naopak zřetelný časový rozměr – v určitém čase je děj zahájen, z počátečního klidového stavu se postupně dostáváme do pohybu, který po určitou dobu trvá, a nakonec dochází k ukončení děje přechodem do klidového stavu. Časování takových akcí zajišťuje uzel **`TimeSensor`**, který ve schématu na obrázku hraje jednoznačnou roli časovače. Vzhledem k důležitému postavení časovače mezi ostatními uzly nás u něj nepřekvapí poměrně vysoký počet parametrů v tabulce T-5-4.

specifikace parametru		jméno	iniciální hodnota		význam
`exposedField`	**`SFBool`**	**`enabled`**	**`TRUE`**		povolení práce časovače
`exposedField`	**`SFTime`**	**`startTime`**	**`0`**		čas zahájení činnosti časovače
`exposedField`	**`SFTime`**	**`stopTime`**	**`0`**		čas ukončení činnosti časovače
`exposedField`	**`SFTime`**	**`cycleInterval`**	**`1`**		nezáporná délka časové smyčky
`exposedField`	**`SFBool`**	**`loop`**	**`FALSE`**		povolení generování časových událostí v nekonečné smyčce
`eventOut`	**`SFBool`**	**`isActive`**		-	zahájení a ukončení práce časovače
`eventOut`	**`SFTime`**	**`time`**		-	plynule generovaný absolutní čas
`eventOut`	**`SFTime`**	**`cycleTime`**		-	čas zahájení další časové smyčky
`eventOut`	**`SFFloat`**	**`fraction_changed`**		-	plynule generovaná poměrná hodnota uplynulého času v rámci jedné smyčky

Tabulka T-5-4: Parametry časovače

Na časovač můžeme pohlížet jako na zařízení, které je schopno průběžně vysílat události – časové impulsy. Frekvenci vysílání těchto impulsů nemůžeme ovlivnit, protože závisí na výkonu počítače. Na rychlejších počítačích je vysíláno mnohem více událostí než na počítači pomalém. Z toho také vyplývá, že nemá příliš velký smysl ptát se po *přesném času*, ale spíše se zajímat o to, zda nějaký děj předchází jinému ději, zda určitá akce následuje po jiné či zda je několik dějů zahájeno ve stejný okamžik.

Parametry časovače uchovávají několik klíčových časových hodnot. Je-li práce časovače povolena (parametr **`enabled`**), události jsou vysílány v době určené hodnotami parametrů **`startTime`** a **`stopTime`**. Krom tohoto jednoduchého přístupu lze dále zpracování času chápat jako cyklickou, opakující se aktivitu a zavést pojem *délka časového intervalu* – **`cycleInterval`**. Podle této délky a rozdílu časů **`startTime`** a **`stopTime`** rozlišujeme, kdy časovač ukončí svoji aktivitu:

1. Je-li **`cycleInterval`** delší než rozdíl koncového a počátečního času, časovač se zastaví po dosažení času **`stopTime`**, tedy bez ohledu na délku intervalu.

2. Je-li délka časového intervalu menší než rozdíl mezních časů, využije se další parametr – **`loop`** (česky *cyklus*). Jeho hodnota **FALSE** říká, že časovač se zastaví okamžitě po dosažení času, který odpovídá uplynutí jednoho cyklu od začátku práce časovače. Hodnota **TRUE** pak povoluje cyklické opakování s tím, že časovač je zastaven po dosažení času **`stopTime`**, tedy po uplynutí několika intervalů a případně uvnitř posledního z nich (není-li rozdíl mezních časů násobkem délky intervalu).

Tato na první pohled složitá strategie pokrývá ve skutečnosti většinu praktických požadavků na zpracování časových událostí. Jedním časovačem lze například řídit budíček, který silně zazvoní v přesně určenou dobu (vyšle událost z parametru **`isActive`**), ale současně se každou minutu připomene krátkým cinknutím, když při hodnotě 60 s uložené v parametru **`cycleInterval`** vysílá pravidelně události z parametru **`cycleTime`**. Průběžně lze přitom zjišťovat aktuální čas přijímáním událostí z parametru **`time`**.

Také další náměty ukazují, jak bohatou škálu dynamických akcí může časovač obsloužit:

- řízení animací a plynulých simulací (například pohyb kabiny výtahu)
- vysílání periodicky se opakujících událostí (rozsvícení nápisu „Rrrračte vstoupit!“ každých 15 sekund)
- vyvolání jednorázové akce v určitou dobu (pád hvězdy)

Mezi nejčastěji kladené otázky na činnost časovače patří:

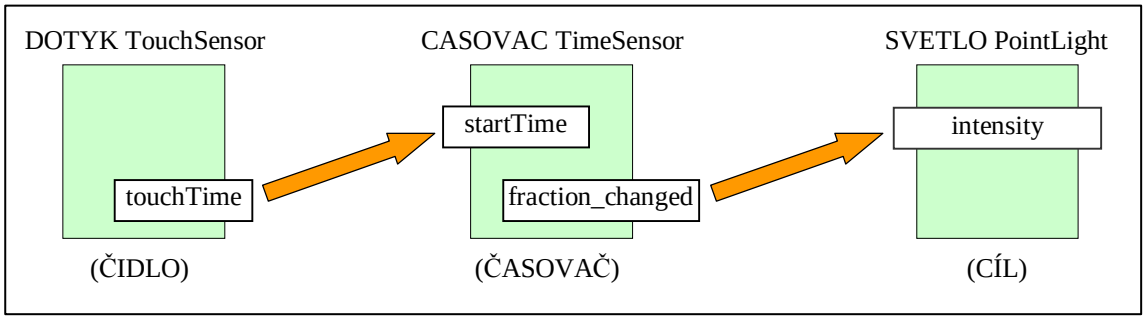
1. Lze vysílat časové impulsy nekonečně dlouho?
2. Lze měnit parametry časovače, když je časovač aktivní?
3. Jaké časy se to vlastně zapisují do parametrů **`startTime`** a **`stopTime`**?

Odpovědi na první dvě otázky jsou kladné. Nekonečné vysílání časových údajů se zajistí poměrně snadno tím, že parametr **`loop`** nastavíme na hodnotu **TRUE** (aby se časovač nezastavil na konci prvního intervalu) a do parametru **`startTime`** zapíšeme hodnotu, která *není menší* než hodnota **`stopTime`**. Tato podmínka indikuje nikdy nekončící generování časových impulsů. Prakticky se toho dociluje zasláním téže události do obou parametrů s mezními časy. I samotná rovnost hodnot postačuje pro nekonečnou činnost časovače.

TIP: Má-li se časovač spustit okamžitě po vstupu do virtuálního světa, stačí uvést v jeho definici v parametru `loop` hodnotu TRUE.

Jakmile jsme zjistili, že je možno spustit časovač na neomezenou dobu, je na místě se ptát na jeho případné zastavení, tedy na reakci na změnu parametrů „za chodu“. Zpracování času je ve virtuální realitě klíčové, vždyť celá virtuální realita je založena na práci v tzv. reálném čase. S časem je tedy třeba zacházet s rozmyslem. Aktivní časovač proto některé události, které mění jeho parametry, sice formálně přijímá, ale na tyto změny nereaguje. Ignoruje především změnu délky časového intervalu (**`cycleTime`**) a pokusy o nastavení hodnot mezních časů (**`startTime`** a **`stopTime`**). Výjimkou je událost měnící parametr **`stopTime`** na hodnotu, která není nižší než aktuální čas. Tato událost je zpracována korektně a může tedy ovlivnit čas, kdy časovač ukončí svoji aktivitu. Nejlepší metodou zastavení běžícího časovače je zaslání události o hodnotě **FALSE** do parametru **`enabled`**. Tuto událost časovač správně přijme, přejde do pasivního stavu a umožní nastavení všech svých parametrů.

Práci s časovačem v nekonečné smyčce ukazuje příklad v programu P-5-7. V něm je nad stříbřitou koulí z předchozího příkladu P-5-6 umístěn jeden zdroj červeného světla. Když se avatar dotkne koule, světlo se začne periodicky rozsvěcet. Jak je vidět na schematickém obrázku 5-9, tato dynamická akce již vyžaduje spolupráci tří uzlů.



Obrázek 5-9: Dynamická akce s detektorem dotyku a časovačem

V příkladu se také dostává na řadu zatím opominutý parametr časovače – **`fraction_changed`**. Je určen k tomu, aby vysílal události obsahující poměrnou hodnotu charakterizující čas, který uplynul v rámci jednoho časového intervalu. Je šikovné využít právě tento parametr k postupnému rozsvěcení světla, protože rozsah jeho hodnot (od 0 do 1) je totožný s rozsahem hodnot, jichž může nabýt intenzita zdroje světla.

Oproti příkladu P-5-6 je tentokrát světelný zdroj zapnut již od začátku, avšak jeho počáteční intenzita je nastavena na nulu. Časovač je připraven pro práci v nekonečném cyklu s časovými intervaly o délce dvě

sekundy (viz parametry **loop** a **cycleInterval**). Záporná hodnota parametru **startTime** zakazuje automatické spuštění časovače po vstupu avatara do virtuálního světa. Jakmile je časovač odstartován přijetím události z detektoru dotyku, rozběhne se v nekonečném počtu opakujících se intervalů, přičemž řídí intenzitu zdroje světla. Určitou nedokonalostí je okamžité zhasínání světla vždy po dosažení konce jednoho intervalu. Očekávané pozvolné zhasnutí nelze totiž realizovat jen s pomocí časovače. Potřebujeme k němu ještě další uzly, s nimiž se seznámíme v následující kapitole.

```
#VRML V2.0 utf8
EXTERNPROTO Stribro [] "../knihovny/kovy.wrl#Silver"

Group {
  children [
    DEF DOTYK TouchSensor {}
    DEF KOULE Shape {
      appearance Appearance { material Stribro {} }
      geometry Sphere {}
    }
  ]
}

DEF SVETLO PointLight {location 0 2 1  intensity 0  color 1 0 0}

DEF CASOVAC TimeSensor {
  loop TRUE
  startTime -1
  cycleInterval 2
}

ROUTE DOTYK.touchTime          TO CASOVAC.startTime
ROUTE CASOVAC.fraction_changed TO SVETLO.intensity
```

Program P-5-7: Animace světla po dotyku objektu

Při práci s časem se musíme vyrovnat se skutečností, že veškerý čas je odměřován *absolutně* (počtem sekund, které uplynuly od půlnoci 1. ledna 1970). Pro praktické účely je však vhodné, když se čas vyjadřuje *relativně* – např. „za minutu po vstupu avatara do místnosti“ nebo „dvě vteřiny po zaťukání na dveře“. Abychom splnili takové požadavky, musíme zařazovat několik časovačů do řady. První z nich působí jako zpožďovač, který sice zahájí svoji činnost okamžitě po detekci nějaké avatarovy aktivity, ale ve skutečnosti pouze nechává plynout čas. Teprve po ukončení práce tohoto prvního, pomocného časovače může dojít k zahájení skutečné dynamické akce řízené druhým časovačem. Je však nepříjemné, že časovou návaznost nelze snadno zabezpečit například tím, že bychom z parametru **stopTime** prvního časovače zaslali událost do parametru **startTime** druhého časovače. Ukončení činnosti časovače je totiž detekováno pouze parametrem **isActive**, který však není vhodný pro přímé odstartování dalších časovačů, protože nevysílá údaj o čase. Řešení nalezneme až v kapitole 6, ve které popíšeme uzel **Script**.

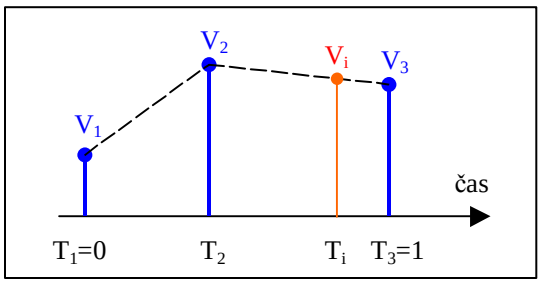
Je vidět, že v případě časovače zapisujeme do souborů konkrétní hodnoty nejčastěji do parametru **cycleInterval**, případně **loop**. Naopak velmi vzácně se setkáme s přímým zápisem číselných hodnot do parametrů **startTime** a **stopTime**. Hodnoty se do těchto parametrů totiž dostávají událostmi, které jsou přijímány od jiných uzlů – typicky čidel. Tato skutečnost je tedy současně odpovědí na třetí otázku položenou v této kapitole.

TIP: Je zcela jedno, do které části stromové struktury zapíšeme časovač. Nemá žádný vztah k sourozencům ani k rodičům. Pouze z důvodu přehlednosti se snažíme zapisovat tento „asociální“ uzel do míst, se kterými bude později propojen příkazem ROUTE.

Uzel **TimeSensor** je neodmyslitelnou součástí většiny dynamických akcí. Budeme se s ním setkávat v téměř všech příkladech uvedených v následujících kapitolách.

5.4 Čím že se to pohání? No přeci interpolátorem! (ColorInterpolator, CoordinateInterpolator, NormalInterpolator, OrientationInterpolator, PositionInterpolator, ScalarInterpolator)

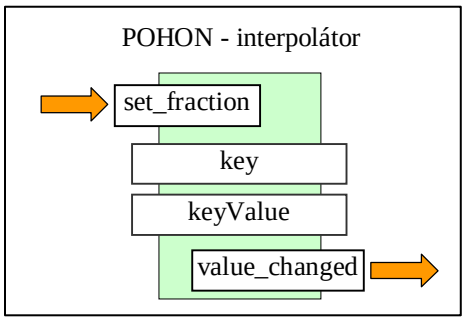
Podíváme-li se znovu na schéma obecné dynamické akce na obrázku 5-2, uvidíme, že cílový objekt nebývá řízen přímo časovačem, nýbrž prvkem nazvaným *pohon*. Jeho úkolem je přijímat časové události a v souladu s nimi generovat události měnící parametry cílových objektů. Do jisté míry se pohon stará i o konverzi dat. Časovač je totiž schopen plynule vysílat události pouze dvou typů – **`SFTime`** a **`SFFloat`**. S takto omezenou škálou dat bychom samozřejmě nedokázali nastavovat u cílových objektů například barvy, normálové vektory a další složité parametry.



Obrázek 5-10: Princip lineární interpolace

Jako pohon slouží ve VRML skupina uzlů nazvaných *interpolátory*. Dříve, než je popíšeme, zastavíme se u pojmu *interpolace*. Tímto slovem se označuje proces vyhledávání mezilehlých hodnot na základě předem daných konstant, nazývaných *klíčové hodnoty*. Na obrázku 5-10 jsou klíčové hodnoty označeny modrými body. Je vidět, že každá klíčová hodnota se vztahuje k nějakému času. Také nově vypočítávaná hodnota je určena časem (T_i) a lze ji proto snadno nalézt mezi dvojicí sousedních klíčových hodnot. Vypočítávaná veličina (označena červeně jako V_i) leží v grafu na spojnici klíčových hodnot, tedy na přímce. Proto se tomuto typu interpolace říká *lineární*.

Lineární interpolace umožňuje na základě malého množství klíčových hodnot generovat velké množství nových hodnot. To je velmi příhodné pro animace. Chceme-li například pohybovat nějakým tělesem po prostorové dráze, stačí zvolit interpolátor pro výpočty prostorových souřadnic a zadat mu ty polohy tělesa, v nichž se dráha výrazněji zakřivuje. Interpolátor pak dopočítá souřadnice všech ostatních poloh.



Obrázek 5-11: Parametry interpolátorů

Existuje celkem šest různých uzlů, které provádějí interpolaci. Každý z nich je zaměřen na jiný typ vypočítávaných hodnot V_i. Všechny mají společné to, že první z klíčových hodnot se vztahuje k počátečnímu času označenému jako nula a poslední ke koncovému času označenému jako jedna (viz obrázek 5-10). Interpolátory jsou dynamické uzly a jsou řízeny jedinou vstupní událostí, která je přijímána parametrem **`set_fraction`** (viz obrázek 5-11). Hodnoty zasílané do tohoto parametru by měly být v intervalu od nuly do jedné a jsou tedy typu **`SFFloat`**. Zde je zřetelná návaznost na parametr **`fraction_changed`** uzlu **`TimeSensor`**.

Nová hodnota je interpolátorem vypočítána podle klíčových hodnot, uložených v seznamu **`keyValue`**. K nim příslušné časové údaje se nacházejí v parametru **`key`**.

specifikace parametru		jméno	iniciální hodnota		význam
`exposedField`	**`MFFloat`**	**`key`**	**`[]`**		neklesající posloupnost klíčových časů
`exposedField`	**`MF…`**	**`keyValue`**	**`[]`**		seznam klíčových hodnot (podle typu uzlu)
`eventIn`	**`SFFloat`**	**`set_fraction`**		**`-`**	vstupní řídicí čas (poměrná hodnota)
`eventOut`	**`SF…, MF…`**	**`value_changed`**		**`-`**	výsledná interpolovaná veličina

Tabulka T-5-5: Parametry společné všem interpolátorům

Tabulka T-5-5 ukazuje, že jména parametrů jsou u všech interpolátorů stejná, odlišnosti nalezneme u datových typů parametrů. Následující přehled uvádí jména a vlastnosti interpolátorů. Druhý sloupec vždy obsahuje typ dat, která jsou interpolátorem vypočítávána a vysílána z parametru **`value_changed`**:

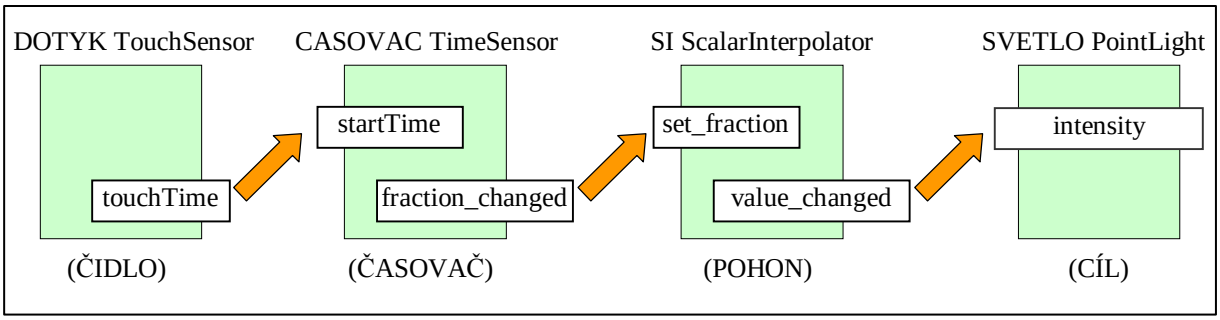
ScalarInterpolator	**SFFloat**	přepočet času na číslo s desetinnou tečkou, univerzální interpolátor pro řízení parametrů v mnoha uzlech
PositionInterpolator	**SFVec3f**	výpočet polohy jednoho bodu v prostoru, vhodné pro určení trajektorie pohybu objektů
ColorInterpolator	**SFColor**	výpočet hodnoty jedné barvy, typicky v materiálovém uzlu nebo ve světelném zdroji
OrientationInterpolator	**SFRotation**	určení osy a úhlu natočení, nejčastěji pro uzel **Transform**, ale třeba i pro stanoviště
CoordinateInterpolator	**MFVec3f**	generování množin souřadnic, které lze použít v uzlech **Extrusion** nebo **Coordinate**
NormalInterpolator	**MFVec3f**	generování množin normálových vektorů, které se použijí v uzlu **Normal**

Tabulka T-5-6: Přehled interpolátorů

Jednotlivé interpolátory přiblížíme v několika příkladech. Stejně jako u časovače platí, že interpolátory nemusejí být umístěny v žádné stromové struktuře. Pro lepší čitelnost je však budeme zapisovat poblíž cílových objektů.

5.4.1 Interpolátor čísla

První příklad představuje vylepšení programu P-5-7. V něm bylo po dotyku objektu zahájeno cyklické rozsvěcování zdroje barevného světla. Zhasnutí však probíhalo naráz. Když mezi časovač a cílový zdroj světla zařadíme interpolátor čísel s vhodnými klíčovými hodnotami, dokážeme v jednom časovém intervalu světlo plynule zapínat i vypínat. Schéma na obrázku 5-12 již připomíná dynamickou akci s téměř všemi dílčími prvky.



Obrázek 5-12: Dynamická akce s detektorem dotyku, časovačem a interpolátorem

Hlavní trik spočívá ve správném nastavení parametrů **key** a **keyValue**. Jak je vidět v programu P-5-8, v rámci jednoho intervalu jsou definovány tři důležité časové momenty zapsané do parametru **key** – počátek, střed a konec intervalu. Jim odpovídají tři klíčové hodnoty v parametru **keyValue**. Pro začátek i konec intervalu je zadána hodnota nula (intenzita světelného zdroje bude nulová). Uprostřed intervalu je klíčová hodnota rovna jedné, což odpovídá plné intenzitě zdroje světla.

Délka intervalu je v časovači nastavena na dvě sekundy. Časový rozsah poměrných klíčových hodnot v parametru **key** interpolátoru bude tedy přepočítáván do rozsahu dvou sekund. Znamená to, že pro postupné zapínání je vymezena jedna sekunda, stejně jako pro následné plynulé zhasnutí.

```
#VRML V2.0 utf8
EXTERNPROTO Stribro [] "../knihovny/kovy.wrl#Silver"

Group {
  children [
    DEF KOULE Shape {
      appearance Appearance { material Stribro {} }
      geometry Sphere {}
    }
    DEF DOTYK TouchSensor  {}
    DEF CASOVAC TimeSensor {loop TRUE  startTime -1 cycleInterval 2 }
    DEF SVETLO PointLight  {location 0 2 1  intensity 0  color 1 0 0}
    DEF SI ScalarInterpolator {
      key [0.0, 0.5, 1.0]
      keyValue [0 1 0]
    }
  ]
}

ROUTE DOTYK.touchTime          TO CASOVAC.startTime
ROUTE CASOVAC.fraction_changed TO SI.set_fraction
ROUTE SI.value_changed         TO SVETLO.intensity
```

Program P-5-8: Animace plynulého rozsvěcování a zhasínání světla po dotyku objektu

5.4.2 Interpolátor barvy

Tento interpolátor vysílá událost obsahující právě jednu barvu, tedy údaj typu **SFColor**. Tím je rozsah možných aplikací omezen na interpolaci barev světelných zdrojů a jednotlivých parametrů uzlu **Material**. Posledním parametrem, který lze takto interpolovat, je barva mlhy. Nelze naopak interpolovat barvy v uzlu **Color**, který je používán na obarvování bodů, úseček a ploch, a který ukládá barvy do parametru typu **MFColor**. Stejně tak nelze přímo interpolovat barvy pozadí.

Program P-5-9 je příkladem použití interpolátoru barvy v barevném nápisu "**Neon**". Pro jednoduchost není zaveden žádný uzel v roli čidla a barva nápisu se mění v nekonečném cyklu trvajícím vždy šest sekund. Interpolátor barvy je propojen s parametrem **emissiveColor** a postupně generuje barvy od červené přes zelenou k modré a opět na červenou. Barevný vjem je zlepšen vypnutím čelní avatarovy svítilny.

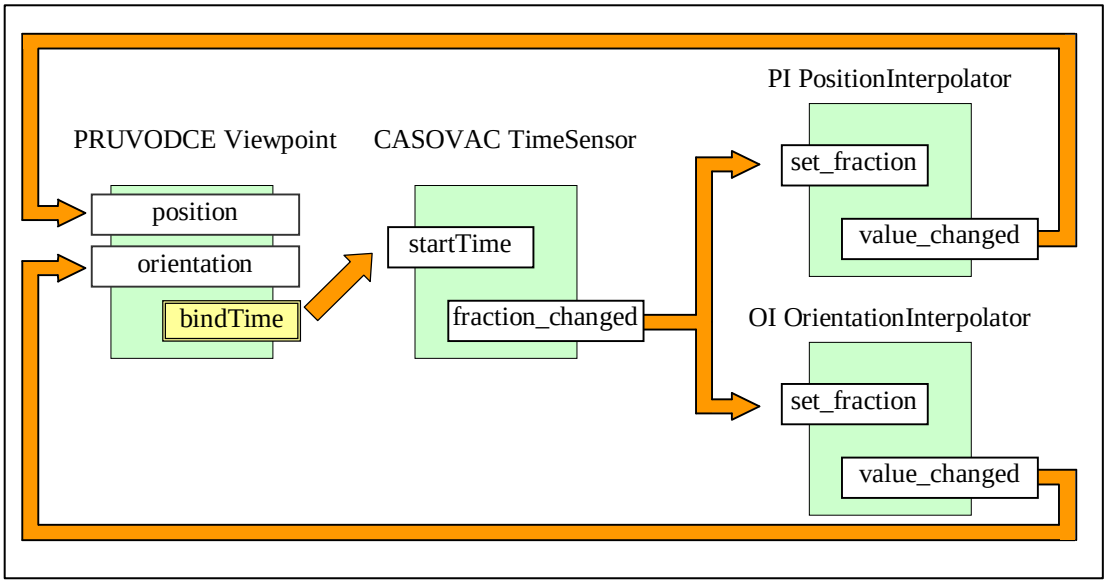
```
#VRML V2.0 utf8

Group {
  children [
    NavigationInfo { headlight FALSE }
    Shape {
      appearance Appearance { material DEF BARVA Material {} }
      geometry   Text {string "Neon"}
    }
    DEF CASOVAC TimeSensor   {loop TRUE cycleInterval 6 }
    DEF CI ColorInterpolator {
      key      [0.0,   0.333, 0.666, 1.0  ]
      keyValue [1 0 0, 0 1 0, 0 0 1, 1 0 0]
    }
  ]
}

ROUTE CASOVAC.fraction_changed TO CI.set_fraction
ROUTE CI.value_changed         TO BARVA.emissiveColor
```

Program P-5-9: Barevné proměny reklamního nápisu

5.4.3 Interpolátor polohy a orientace

Uzly **PositionInterpolator** a **OrientationInterpolator** se často používají společně, protože pohyb objektu bývá mnohdy doprovázen i změnou jeho orientace. Tak kupříkladu letící míč se v letu otáčí, ryba plovoucí v akváriu se vlní ze strany do strany, vývrtka se současně zvedá i otáčí apod.

Kromě virtuálních objektů lze animovat i pohyb avatara. V příkladu P-5-10 jsou události ze dvou interpolátorů zasílány do uzlu **Viewpoint** s cílem zavést avatara do pravého rohu virtuální místnosti s několika stolky z příkladu P-4-1. Čidlem bude v tomto případě přímo stanoviště pojmenované **PRUVODCE**. Jakmile si návštěvník toto stanoviště vybere, jeho parametr **bindTime** vyšle událost, která odstartuje animaci polohy a orientace avatara. Schéma dynamické akce na obrázku 5-13 má tentokrát zajímavý tvar. Časovač řídí dva interpolátory, oba pak zasílají události uzlu, který akci vyvolal. Čidlo (stanoviště) se v tomto případě stává i cílem akce.

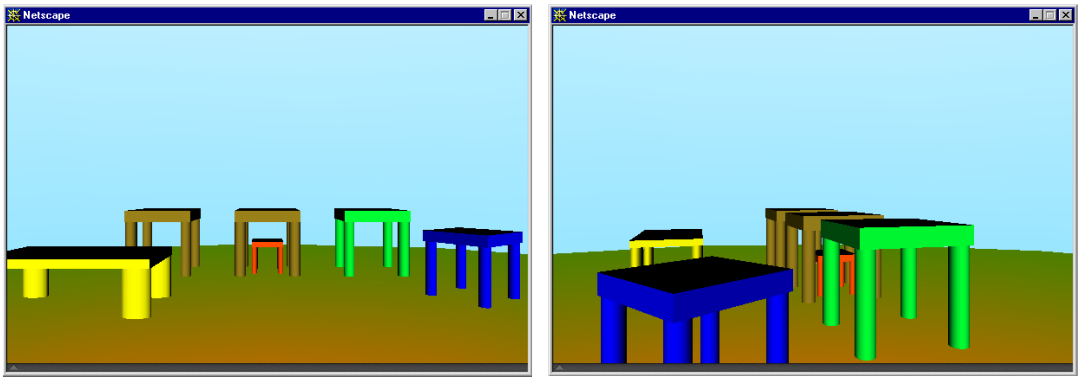


Obrázek 5-13: Řízená prohlídka virtuálního světa pomocí interpolátoru polohy a orientace

```
#VRML V2.0 utf8

Inline {url "../kap4/p-4-1.wrl"}        # svět s několika stolky

DEF PRUVODCE Viewpoint { description "Pruvodce" position 0 1.5 8}
DEF CASOVAC TimeSensor { cycleInterval 10 }
DEF PI PositionInterpolator {
       key      [0.0,     0.3,     0.9,     1.0]
       keyValue [0 1.5 8, 0 1.5 8, 6 1.5 2, 6 1.5 2]
}
DEF OI OrientationInterpolator {
       key      [0.0,     0.08,       0.12,       0.3, 0.8, 0.9, 1.0]
       keyValue [0 1 0 0, 0 1 0 0.87, 0 1 0 0.87, 0 1 0 -0.43,
                 0 1 0 0.87, 0 1 0 1.1, 0 1 0 1.3]

}

ROUTE PRUVODCE.bindTime        TO CASOVAC.startTime
ROUTE CASOVAC.fraction_changed TO PI.set_fraction
ROUTE CASOVAC.fraction_changed TO OI.set_fraction
ROUTE PI.value_changed         TO PRUVODCE.position
ROUTE OI.value_changed         TO PRUVODCE.orientation
```

Program P-5-10: Animovaná procházka s pomocí interpolátoru polohy a orientace

Všimněme si, že k vytvoření iluze přirozeného pohybu ve virtuálním prostředí je třeba s rozmyslem navrhovat časování interpolátorů. V programu P-5-10 je celá animovaná procházka rozvržena do deseti sekund. V prvních třech sekundách necháváme avatara stát na místě a pouze natáčíme jeho pohled v první vteřině doleva a v následujících dvou vteřinách doprava. Má-li být toto rozhlížení podobné opravdovému, je třeba zpomalit pohyb při změně otáčení avatarovy hlavy. Proto je časování v interpolátoru orientace jemnější, než v interpolátoru polohy (viz klíče **`0.08`** a **`0.12`** pro druhý časový úsek).

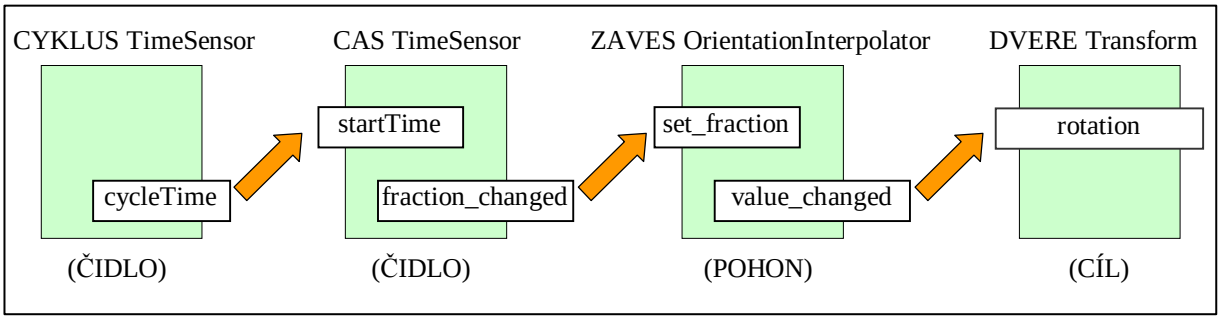


Obrázek 5-14: Pohledy na virtuální svět na začátku a na konci řízené procházky

Poté, co se avatar rozhlédne ze strany na stranu, vydá se šikmo dopředu a doprava. Cestou mírně natáčí svůj pohled doleva, aby viděl na prostor se stolky. Nakonec se zastaví napravo od stolků a dokončí otáčení hlavy doleva. Pro pomalejší počítače je deset sekund k takovéto animaci poměrně krátká doba. Výsledkem je trhaný pohyb avatara. Řešením je prodloužení času na animaci, například na dvacet sekund.

TIP: Otočení v uzlu `OrientationInterpolator` o více než 180º je třeba provést postupně, za pomoci mezilehlého úhlu. Interpolátor by jinak zvolil cestu přes doplňkový, tedy menší úhel. Pomocný mezilehlý úhel použijeme i tehdy, když chceme zajistit, aby se virtuální vojáci otáčeli při povelu „Čelem vzad!“ shodně přes levou stranu.

Dalším příkladem použití interpolátoru orientace je definice dveří, které se pravidelně otevírají a zavírají. Zavedeme pro ně dva časovače. První, nazvaný **CYKLUS**, bude pouze každých dvacet sekund opakovaně startovat dynamickou akci. Vlastní otevírání dveří bude řízeno druhým časovačem, pojmenovaným **CAS**. Jeho úkolem bude po dobu šesti sekund generovat události pro interpolátor orientace. Schéma akce je ukázáno na obrázku 5-15.



Obrázek 5-15: Dveře, které se cyklicky otevírají a zavírají

Program P-5-11 je téměř doslovným přepisem schématu z obrázku 5-15. Všimneme si, že geometrie dveří musí být posunuta o poloviny šířky dveří, jinak by se dveře otáčely kolem středové osy namísto svislé boční osy. Pozornost si zaslouží i dynamika rotačního pohybu. Aby bylo otevírání dveří dostatečně realistické, je jejich otevírání rozděleno do dvou fází. V první z nich se dveře otevřou dokořán (o úhel 1.6 radiánů), přičemž tento pohyb je vykonán poměrně velmi rychle – je pro něj vymezeno jen 20 % z celkové doby animace šest sekund (viz hodnota 0.2 v parametru **`key`** interpolátoru). Ve druhé fázi se dveře ještě pomalu pootevřou o malý úhel 0.2 rad, což vyvolá správný dojem zbrždění pohybu. Po určité době klidu se dveře obdobným způsobem zavřou.

```
#VRML V2.0 utf8

Group {
  children [
    DEF CYKLUS TimeSensor { cycleInterval 20 loop TRUE }
    DEF CAS    TimeSensor { cycleInterval 6 }      # délka animace
    DEF ZAVES  OrientationInterpolator {
                 key [0, 0.2, 0.3, 0.7, 0.9, 1]
                 keyValue [0 1 0 0,    0 1 0 -1.6, 0 1 0 -1.8,
                           0 1 0 -1.8, 0 1 0 -0.2, 0 1 0 0]
    }
    DEF DVERE Transform {           # pro otáčení
        children Transform {        # pro posunutí svislé osy dveří
          translation 0.4 0 0
          children Shape {          # tvar dveří
                     geometry Box { size 0.8 1.8 0.1}
          }
        }
    }
]}

ROUTE CYKLUS.cycleTime      TO CAS.startTime
ROUTE CAS.fraction_changed  TO ZAVES.set_fraction
ROUTE ZAVES.value_changed   TO DVERE.rotation
```

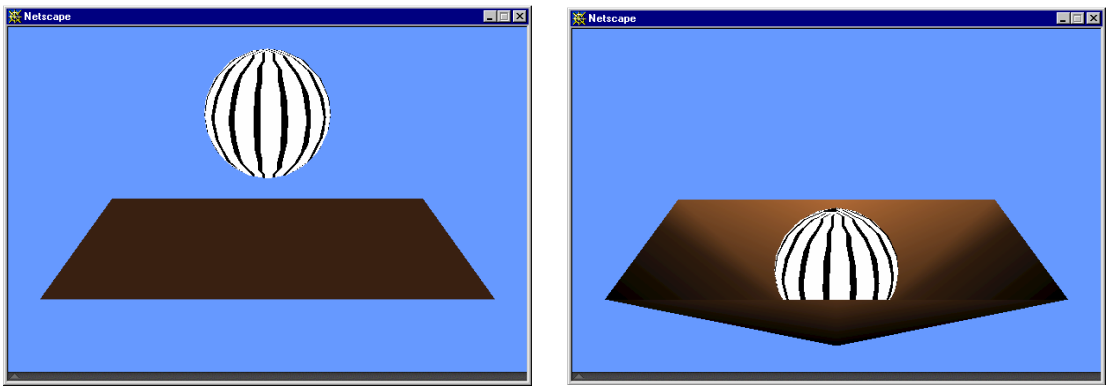
Program P-5-11: Otevírání dveří s pomocí interpolátoru orientace

5.4.4 Interpolátor souřadnic a normálových vektorů

Poslední dva interpolátory, **`CoordinateInterpolator`** a **`NormalInterpolator`**, se od předchozích odlišují tím, že namísto jediné hodnoty poskytují ve svém parametru **`value_changed`** celou množinu, lépe řečeno posloupnost hodnot. Interpolují tedy mezi zadanými klíčovými posloupnostmi, a to s ohledem na pořadí. První hodnoty v klíčových posloupnostech slouží k získání nové první hodnoty ve výsledné posloupnosti, podobně se z druhých hodnot stanovuje druhá hodnota ve vyhodnocené posloupnosti apod.

Použití těchto dvou interpolátorů je poměrně náročné, neboť je pro ně potřeba zadat mnohem více dat, než pro předchozí jednoduché interpolátory. „Ruční“ zadávání hodnot téměř nepřichází do úvahy, doporučuje se využít programů pro modelování těles a z nich potřebná data exportovat do souboru VRML. Oba interpolátory bývají často připojeny ke stejnému virtuálnímu tělesu, typicky množině ploch (**`IndexedFaceSet`**) nebo obecnému opláštění (**`Extrusion`**). Umožňují totiž velmi realisticky a téměř libovolně deformovat tvar tělesa. Interpolace souřadnic zajišťuje geometrické změny tvaru, interpolace normál umocňuje výsledný optický dojem.

Samozřejmě existují situace, ve kterých není potřeba použít oba tyto interpolátory najednou. Chceme-li například vyvolat zdání, že virtuální závěs na stěně se mírně vlní, není nutno měnit jeho souřadnice, ale stačí vhodným způsobem naklánět normály v jeho vrcholech doleva a doprava. Pochopitelně je třeba, aby závěs nebyl modelován jedinou velkou plochou, ale sítí několika menších plošek, například výškovou mapou (**`ElevationGrid`**). Podobně dokážeme vytvořit iluzi vzdálené vlající vlajky.

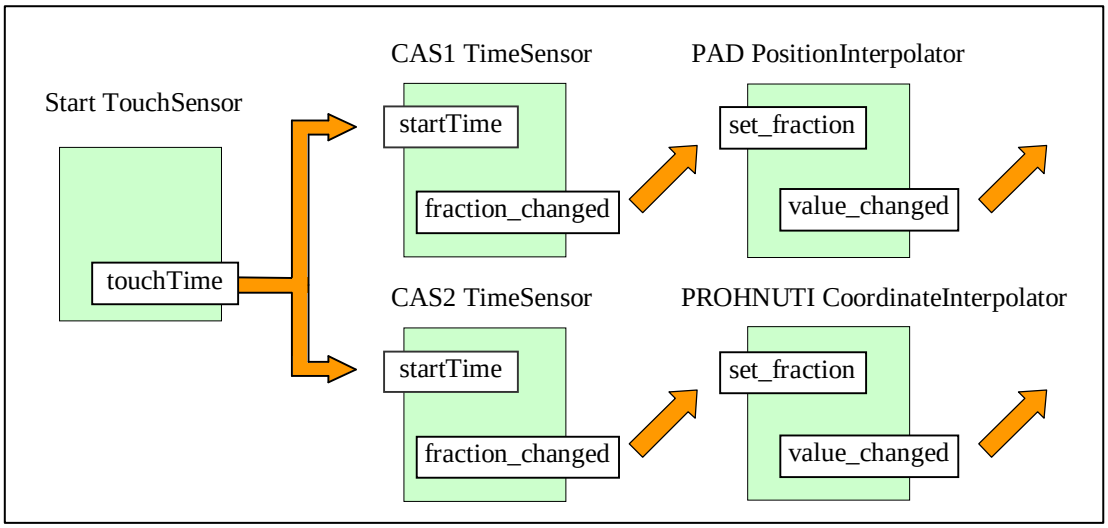


Obrázek 5-16: Deformace jednoduché trampolíny pomocí interpolátoru souřadnic

Tam, kde si může avatar prohlížet objekty zblízka, samotná interpolace normál nepostačuje a je třeba použít interpolátor souřadnic. Je-li přitom interpolace dostatečně jemná a je-li vhodně nastaven parametr **`creaseAngle`** v cílovém uzlu **`IndexedFaceSet`**, odpovídající výpočet normálových vektorů je prováděn prohlížečem automaticky a není třeba zavádět interpolátor normál. V uzlech **`IndexedLineSet`** a **`PointSet`** se dokonce normály neuplatní vůbec.

Následující příklad je ukázkou použití interpolátoru souřadnic. Jednoduchá trampolína na obrázku 5-16 je definována sítí čtyř menších ploch, tj. čtvercem třikrát tří vrcholů. Jakmile se avatar dotkne míče nad trampolínou, míč spadne a trampolína se prohne. Její deformace docílíme interpolací souřadnic, v tomto jednoduchém případě dojde ke změně souřadnice jediného vnitřního vrcholu trampolíny.

Schéma na obrázku 5-17 ukazuje, že jeden detektor dotyku spojený s koulí zahajuje současně dvě animace. První představuje pád koule do trampolíny rozdělený do dvou fází – volného a brzděného pádu. Druhá animace prohýbá trampolínu. Odpovídající interpolátor souřadnic má nastaveny klíčové hodnoty tak, aby vlastní prohnutí bylo zřetelné teprve od okamžiku dopadu koule na povrch trampolíny.



Obrázek 5-17: Schéma animace dopadu koule na trampolínu

Podíváme-li se do programu P-5-12, nalezneme v interpolátoru souřadnic pojmenovaném **`PROHNUTI`** celkem tři skupiny souřadnic. První dvě reprezentují trampolínu v klidu a těsně před dopadem koule. Teprve třetí, poslední posloupnost hodnot obsahuje souřadnice vrcholů deformované trampolíny. Zbytečné zdvojení dat pro první a druhou posloupnost souřadnic je způsobeno nešikovně zadanou synchronizací animačních dějů. Časovač **`CAS2`** zodpovědný za animaci trampolíny je totiž odstartován okamžitě po zahájení pádu koule, namísto po dopadu koule na trampolínu. Lepší časové synchronizace a tedy i úspory dat docílíme až použitím uzlu **`Script`** (viz další kapitola).

Je zajímavé, že posloupnosti hodnot v parametru **`keyValue`** interpolátoru souřadnic **`PROHNUTI`** nejsou navzájem odděleny žádnými zvláštními symboly, například minus jedničkou nebo hranatými závorkami. Parametr v našem případě obsahuje dvacet sedm souřadnic vrcholů, které jsou chápány jako tři skupiny dat o devíti prvcích. Tento význam dat je interpolátoru dodán teprve propojením s uzlem **`VRCHOLY`**, který vyžaduje devět souřadnic vrcholů. Kdyby na jeho místě stál například uzel vyžadující pouze tři souřadnice, náš interpolátor by chápal zadaná data jako devět posloupností o třech prvcích.

Na závěr ještě připomeňme, že trampolína v programu P-5-12 se ve skutečnosti nechová jako pružná plocha, ale jen jako měkké plátno. Koule není odražena zpět do prostoru, ale zůstane ležet na povrchu prohnuté trampolíny. Na čtenářově fantazii ponecháváme návrh scény, ve které bude koule na trampolíně skákat nahoru a dolů, ať již v nekonečném cyklu či dokonce s pozvolným útlumem.

```
#VRML V2.0 utf8

EXTERNPROTO Bronz  [] "../knihovny/kovy.wrl#Bronze"

Group { children [
    DEF KOULE Transform {
        translation 0 2 0
        children Inline {url "../knihovny/ovladac-koule.wrl"}
    }
    DEF START TouchSensor {}
]}

Group {
  children
    Shape {
      geometry IndexedFaceSet {
        coord DEF VRCHOLY Coordinate {
          point [-3 0 -3, 0 0 -3, 3 0 -3,
                 -3 0 -1, 0 0  0, 3 0 -1,
                 -3 0  3, 0 0  3, 3 0  3]
        }
        coordIndex [3 4 0 -1, 0 4 1 -1, 4 5 2 -1, 4 2 1 -1,
                    4 3 6 -1, 4 6 7 -1, 4 7 8 -1, 8 5 4 -1]
        solid       FALSE
        creaseAngle 1.5
      }
      appearance Appearance { material Bronz {} }
    }
}

DEF CAS1 TimeSensor {cycleInterval 3 }
DEF CAS2 TimeSensor {cycleInterval 3 }
DEF PAD PositionInterpolator {
          key [0 0.3 1]
          keyValue [0 2 0, 0 1 0, 0 –0.5 0]
}
DEF PROHNUTI CoordinateInterpolator {
          key [0 0.3 1]
          keyValue [ -3 0 -3, 0 0 -3, 3 0 -3,
                     -3 0 -1, 0 0  0, 3 0 -1,
                     -3 0  3, 0 0  3, 3 0  3,
                   -3 0 -3, 0 0 -3, 3 0 -3,
                   -3 0 -1, 0 0  0, 3 0 -1,
                   -3 0  3, 0 0  3, 3 0  3,
                 -3 0 -3, 0  0  -3, 3 0 -3,
                 -3 0 -1, 0 -1.8 0, 3 0 -1,
                 -3 0  3, 0  0   3, 3 0  3,
          ]
}

ROUTE START.touchTime        TO CAS1.startTime
ROUTE CAS1.fraction_changed  TO PAD.set_fraction
ROUTE PAD.value_changed      TO KOULE.translation

ROUTE START.touchTime        TO CAS2.startTime
ROUTE CAS2.fraction_changed  TO PROHNUTI.set_fraction
ROUTE PROHNUTI.value_changed TO VRCHOLY.point
```

Program P-5-12: Trampolína deformovaná po dopadu koule

5.5 Avatare! – Velký Bratr tě vidí … (Collision, ProximitySensor, VisibilitySensor)

Virtuální světy jsou nazývány interaktivními především proto, že jejich jednotlivé části mohou reagovat na avatarovu aktivitu. Manipulátory umožňují měnit transformační koeficienty objektů, detektor dotyku slouží jako čidlo, po jehož aktivaci se mohou rozběhnout rozličné animace. Ve všech těchto případech hraje avatar aktivní roli – jeho ruka uchopí manipulátor nebo se dotkne objektu a teprve poté je dynamická akce zahájena. Připomeňme, že pomyslná avatarova ruka je nejčastěji reprezentována kombinací kurzoru a tlačítek myši.

Ve virtuálních světech lze provádět dynamické akce, aniž by avatar musel kamkoliv ukázat nebo se čehokoliv dotknout. To je výhodné zejména tehdy, je-li avatar nerozhodný či přímo bojácný. Představme si, že máme pro návštěvníka virtuálního světa přichystáno mnoho animací, efektů, dynamických dějů. Pokud bychom čekali, až se avatarova ruka dotkne nějakého tělesa, nemuseli bychom se dočkat nikdy a avatar by nakonec zklamán opustil náš svět se slovy: „Tady se nic neděje.“ Je zřejmé, že potřebujeme takové uzly, které budou schopny zahájit dynamické akce i tehdy, pokud avatar naším světem pouze prochází. Takové uzly nebudou čekat na dotek avatarovy ruky, ale budou si všímat avatara jako celku, tj. jeho přítomnosti a chování v určitém prostoru.

V roli „skryté kamery“ vystupují celkem tři dynamické uzly:

1. **`Collision`** (detektor nárazu)
   zjišťuje, zda avatarova postava narazila do určitého objektu

2. **`ProximitySensor`** (detektor přítomnosti)
   sleduje avatarovu polohu a orientaci v prostoru omezeném pomyslným kvádrem

3. **`VisibilitySensor`** (detektor viditelnosti)
   určuje, zda se daná oblast obklopená pomyslným kvádrem dostala do avatarova zorného prostoru a zda je alespoň část této oblasti viditelná.

Všechny tyto uzly slouží jako čidla. Jakmile avatar při své procházce dorazí na místo hlídané některým z těchto detektorů, může svým chováním, tj. pohybem, vyvolat vyslání události s příslušnou informací. Avatar tedy nemusí aktivně ukázat na nějaký objekt, protože namísto činnosti jeho ruky je sledována činnost jeho postavy – do čeho naráží, v jakém prostoru se nachází a co vidí.

Patrně nejsložitějším uzlem z této trojice je uzel **`Collision`**. Zjišťování kolizí mezi avatarem a tělesy je sice prováděno automaticky prohlížečem virtuálního světa, ale pouze uzel **`Collision`** může o takových situacích informovat další uzly. Dokáže jim zaslat jedinou událost ze svého parametru **`collideTime`**, tedy čas, kdy avatar při svém pohybu virtuálním prostředím narazil[11] na nějakou překážku.

Objekty, které jsou citlivé na avatarův náraz, se zapisují jako potomci detektoru do parametru **`children`**. Protože výpočty kolizí jsou poměrně náročné, pro zrychlení vyhodnocovacího systému se zavádějí další parametry:

- **Pomocná obálka** ve tvaru kvádru (angl. *bounding box*, *bbox*)
  slouží pouze ke zrychlení výpočtů. Má tvar kvádru, jehož hrany jsou rovnoběžné se souřadnicovými osami. Měla by zcela obklopit geometrii všech potomků. Obálka není zobrazována a avatar do ní může bez problémů vstoupit. Je-li však mimo ni, složité výpočty kolizí jsou dočasně vyřazeny. Obálka je určena svými rozměry (parametr **`bboxSize`**) a polohou těžiště (**`bboxCenter`**). Rozměry musí být kladné, výjimkou je pouze trojice délek **`-1 -1 -1`** indikující, že prohlížeč si má obálku určit sám.

- **Náhradní tělesa**
  jsou určitou obdobou jednoduché reprezentace v uzlu **`LOD`**, byť s odlišnou funkcí. Dovolují definovat jednoduchou, avšak nezobrazovanou geometrii, která v testech na kolize nahradí původní složité tvary vlastních potomků detektoru nárazu. Náhradní těleso nebo skupina náhradních těles se zapisuje do parametru **`proxy`**.

[11] Vzpomeňme si, že avatarovo tělo je představováno „válcem na nožičkách“ (viz kapitola 2.3). Jde tedy o zjištění, zda došlo k nárazu tohoto válce na nějaký virtuální předmět.

TIP: Náhradní tělesa sice nejsou vidět, působí však jako neprostupná zeď. S jejich pomocí zajistíme, aby avatar nevešel tam, kam nemá, například za okraj propasti nebo balkónu.

Je třeba zdůraznit rozdíl mezi pomocnou obálkou a náhradními tělesy. Obálka je tím nejobyčejnějším prostředkem pro zrychlení výpočtů, protože dokáže pouze výpočty vypnout, je-li avatar dostatečně daleko od testovaných objektů. Náhradní tělesa reprezentují mnohem důmyslnější strategii. Jakmile jsou definována, detektor nárazu se při avatarově pohybu nevěnuje *původní* geometrii potomků, ale pouze náhradním tělesům. To je velmi výhodné zejména tehdy, pokud je původní geometrie definována desítkami a stovkami plošek. Příkladem může být model křišťálového lustru tvořeného desítkami drobných sklíček. I tak jednoduché náhradní těleso, jakým je koule nebo kostka, dokáže detekovat kolizi s avatarovým tělem, po níž následuje rozbití lustru na tisíce střepů (reprezentovaných vhodnou texturou na podlaze) nebo alespoň jeho varovné rozhoupání.

TIP: S pomocí náhradních těles lze vytvořit druhý, neviditelný svět. Milovníci tajemství mohou nechat avatara procházet zdí, když v její náhradní reprezentaci ponechají tajný otvor.

Náhradní tělesa nemusí pouze urychlovat výpočty, ale použijeme je i tehdy, pokud nutně potřebujeme testovat náraz s takovými uzly, u nichž se standardně kolize nevyhodnocují. Jde o uzly **`IndexedLineSet`**, **`PointSet`** a **`Text`**. Pokud se takové uzly objeví v parametru **`children`**, je nutno pro ně zavést náhradní popis pomocí těles. Tímto způsobem lze například pavučinu definovanou jako množina čar (**`IndexedLineSet`**) nahradit z hlediska kolizí jednou plochou (**`IndexedFaceSet`**) a po nárazu nechat vykouknout zvídavého pavouka.

specifikace parametru		jméno	iniciální hodnota		význam
`exposedField`	`SFBool`	`collide`	`TRUE`		povolení činnosti detektoru
`exposedField`	`MFNode`	`children`	`[]`		seznam potomků
`field`	`SFNode`	`proxy`	`žádný`		uzel nebo strom s náhradními tělesy
`field`	`SFVec3f`	`bboxSize`	`-1 -1 -1`		velikost obálky ve tvaru kvádru
`field`	`SFVec3f`	`bboxCenter`	`0 0 0`		střed obálky
`eventIn`	`MFNode`	`addChildren`		`-`	seznam připojovaných potomků
`eventIn`	`MFNode`	`removeChildren`		`-`	seznam odstraňovaných potomků
`eventOut`	`SFTime`	`collideTime`		`-`	čas nárazu avatara

Tabulka T-5-7: Parametry detektoru nárazu

Parametry uzlu **`Collision`** jsou uvedeny v tabulce T-5-7. V ní si kromě parametru **`collide`**, který povoluje či zakazuje práci detektoru, všimneme dvou parametrů třídy **`eventIn`**. Jde o parametry, s jejichž pomocí lze dynamicky měnit počet potomků uzlu. Zasláním vhodných událostí se seznamem uzlů do parametrů **`addChildren`** a **`removeChildren`** lze přidávat, resp. odebírat uzly zapsané v parametru **`children`**. Žádný s dosud uvedených uzlů však nedokáže takové události zasílat a proto je tato možnost ponechána speciálnímu uzlu **`Script`**, se kterým se seznámíme až v následující kapitole.

Je zajímavé, že parametry **`addChildren`** a **`removeChildren`** se vyskytují ve všech skupinových uzlech. Detektor nárazu je tedy skupinovým uzlem stejně, jakou jsou jím uzly **`Anchor`**, **`Billboard`**, **`Group`** a **`Transform`**.

TIP: Kolize jsou vyhodnocovány i tehdy, pokud avatar zkoumá svět v režimech `"EXAMINE"` či `"ANY"` (viz uzel `NavigationInfo`), při nichž se jinak kolize vypínají. Jediný způsob, jak detektor nárazu vypnout, spočívá v nastavení jeho parametru `collide`.

Teoreticky je možno zapsat mezi potomky jednoho detektoru další detektor nárazu. Tím dokážeme detekovat náraz ve více stupních (detailech), například „avatar narazil do auta, konkrétně do předního nárazníku“. Pro takovou situaci musíme zapsat detektor příslušný nárazníku mezi potomky detektoru, který je zodpovědný za celé auto. Vypnutí činnosti rodičovského detektoru (parametr **`collide`**) má však za následek vyřazení detekce nárazu i pro všechny potomky. Navíc nesmíme v rodičovském detektoru použít náhradní tělesa, protože jejich zavedení indikuje, že skuteční potomci se testování kolize nezúčastňují.

Příklad v programu P-5-13 využívá dvou detektorů nárazu. Virtuální svět obsahuje pouze dva válce – levý je menší a tlustší, pravý je tenčí a vyšší. Mezi nimi je mezera. Pokud avatar při průchodu světem narazí do pravého válce, ozve se zvuk přehraný ze souboru "**smich.wav**". Po nárazu do levého válce se ozve tentýž zvuk, avšak zpomalený na polovinu, tedy hlubší (viz parametr **pitch** v uzlu **AudioClip**).

```
#VRML V2.0 utf8

DEF LEVY Collision {
 children [
   Transform {
     translation -3 0 0           # posunutí doleva
     scale 2 1 2
     children DEF VALEC Inline {url "../knihovny/ovladac-valec.wrl"}
   }
]}

DEF PRAVY Collision {
 children [
   Transform {
     translation 3 0 0            # posunutí doprava
     scale 1 2 1
     children USE VALEC
   }
]}

Sound { source DEF ZVUK1 AudioClip
          { url "../knihovny/smich.wav"    pitch 0.5 }
}
Sound { source DEF ZVUK2 AudioClip
          { url "../knihovny/smich.wav"}
}

ROUTE LEVY.collideTime  TO ZVUK1.startTime
ROUTE PRAVY.collideTime TO ZVUK2.startTime
```

Program P-5-13: Detekce kolize u dvou těles

Vzhledem k jednoduchosti příkladu neuvádíme schéma dynamické akce. Všimneme si pouze, že na rozdíl od detektoru dotyku nejsou cílová tělesa zapsána jako sourozenci uzlu **Collision**, ale jsou přímo jeho potomky. V příkladu také nejsou použita náhradní tělesa ani pomocná obálka. Pokud bychom například chtěli avatarovi zakázat průchod mezerou mezi válci, zapsali bychom oba válce jako potomky jediného detektoru nárazu a navíc definovali vhodné náhradní těleso, které by obklopilo válce i prostor mezi nimi. Každý avatarův pokus o průchod by pak způsobil stejnou zvukovou odezvu.

Co avatar právě dělá? Co vidí?

Zbylé dva uzly, s jejichž pomocí lze sledovat avatara stejně, jako tajní agenti sledují své nic netušící oběti, jsou poměrně jednoduché. Vždy pracují s nezobrazovaným, pomocným kvádrem, ať již se v něm avatar pohybuje (uzel **ProximitySensor**) nebo je schopen jej zahlédnout (**VisibilitySensor**). Kvádr může být transformován podle parametrů rodičovských uzlů. Na rozdíl od detektoru pohybu, který vysílal jedinou událost danou nárazem do objektu, tyto detektory vysílají i události informující o zahájení a ukončení činnosti avatara, týkající se daného kvádru. Jejich společné parametry jsou uvedeny v tabulce T-5-8.

specifikace parametru		jméno	iniciální hodnota	význam
exposedField	**SFVec3f**	**center**	**0 0 0**	těžiště kvádru vymezujícího sledovanou oblast
exposedField	**SFVec3f**	**size**	**0 0 0**	nezáporné rozměry kvádru
exposedField	**SFBool**	**enabled**	**TRUE**	povolení činnosti detektoru
eventOut	**SFBool**	**isActive**	-	avatar se pohybuje v oblasti, resp. ji vidí
eventOut	**SFTime**	**enterTime**	-	čas, kdy parametr **isActive** nabyl hodnoty **TRUE**
eventOut	**SFTime**	**exitTime**	-	čas, kdy parametr **isActive** nabyl hodnoty **FALSE**

Tabulka T-5-8: Společné parametry detektoru přítomnosti a viditelnosti

Detektor viditelnosti **VisibilitySensor** má právě ty parametry, které jsou uvedeny v tabulce T-5-8. Jakmile se avatar dostane do pozice, ze které vidí alespoň část testovaného kvádru, z parametru **isActive** je vyslána událost o hodnotě **TRUE**. Další událost je pak z tohoto parametru vyslána v okamžiku, kdy se kvádr ocitne mimo zorný úhel avatara nebo kdy je zcela zakryt jinými částmi světa. Spolu se změnami hodnoty parametru **isActive** dochází k vyslání odpovídajících časových událostí z parametru **enterTime**, resp. **exitTime**. Parametr **isActive** nemusí být vyhodnocován všemi prohlížečích stejně. Někdy je vyslána událost o hodnotě **TRUE** hned, jakmile se testovaný kvádr dostane do zorného úhlu avatara, jindy teprve tehdy, pokud může být kvádr skutečně spatřen, tj. pokud není zakryt jinými objekty.

Detektor přítomnosti je navíc oproti detektoru viditelnosti schopen informovat další uzly o tom, jak se mění poloha a orientace avatara uvnitř sledované oblasti. Obsahuje dva parametry třídy **eventOut**, které při pohybu avatara průběžně vysílají události s jeho aktuální polohou (parametr **position_changed**) a natočením (**orientation_changed**). Takové události lze přijímat v uzlu **Transform** a v dané oblasti například definovat ptáčka, který před blížícím se avatarem vlétne vzhůru nebo se k němu neustále otáčí zadečkem[12]. Z větší vzdálenosti (mimo sledovanou oblast) si přitom lze ptáčka prohlédnout v klidu ze všech stran.

specifikace parametru		jméno	význam
eventOut	**SFVec3f**	**position_changed**	poloha avatara uvnitř oblasti se změnila
eventOut	**SFRotation**	**orientation_changed**	orientace avatara uvnitř oblasti se změnila

Tabulka T-5-9: Speciální parametry detektoru přítomnosti

TIP: Detektory přítomnosti a viditelnosti jsou ideálními čidly pro zahájení animací. Je zbytečné a také výpočetně náročné provádět dynamické akce, když je avatar v jiné místnosti nebo když je k animovanému ději otočen zády.

Detektory přítomnosti a viditelnosti můžeme zařazovat do virtuálního světa ve větším počtu. Platí přitom následující pravidla:

1. Existuje-li několik různých detektorů, jejichž sledované oblasti se překrývají, je aktivita avatara detekována současně všemi těmito uzly.

2. Je-li jeden detektor násobně vložen na různá místa pomocí příkazů **DEF** a **USE**, vznikne násobná oblast jako sjednocení jednotlivých kvádrů. Kvádry se v tomto případě nesmějí překrývat.

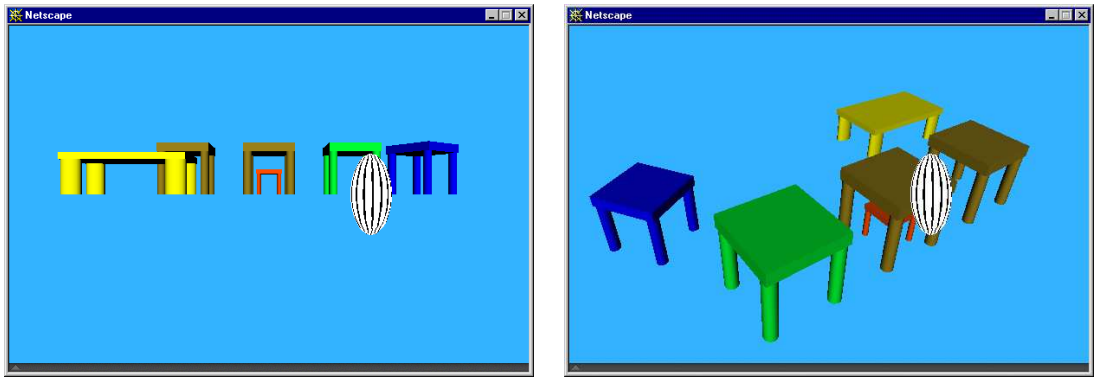
Zajímavým využitím detektoru přítomnosti je zavedení tzv. *průvodců* – objektů, které avatara provázejí při každém jeho pohybu a jsou tedy jakoby přilepeny k obrazovce. Na obrázcích 5-18 má jeden takový průvodce tvar elipsoidu, praktické je však definovat průvodců několik. Aby jejich existence nebyla samoúčelná, je vhodné je provázat s různými detektory a průvodcům přiřadit různé grafické symboly vyjadřující například přechod na

[12] Vzlétnutí nelze bohužel realizovat přímo, protože je nutno převádět měnící se pozici avatara na výšku ptáčka. Tento výpočet lze provést pouze ve speciálním uzlu **Script** (viz další kapitola). Otáčení ptáčka je proti tomu hračkou – stačí propojit parametr **orientation_changed** s parametrem **rotation** správného transformačního uzlu.

iniciální stanoviště, vyvolání HTML stránky s nápovědou, spuštění či zastavení doprovodné hudby apod. Pokud si tvůrce virtuálního světa přeje z nějakého důvodu zakázat používání ovládacích prvků prohlížeče (nastavením parametru **type** v uzlu **NavigationInfo** na hodnotu **"NONE"**), může pomocí průvodců definovat vlastní způsoby řízení pohybu avatara.

V příkladu P-5-14 je průvodce tvaru elipsoidu propojen s uzlem **Anchor**, který po dotyku zajistí přesun avatara na určené stanoviště. Toto stanoviště, pojmenované **DOMA**, je součástí definice daného světa.V programu P-5-14 je vidět, že detektor přítomnosti má nastaveny takové iniciální rozměry, aby by se avatar okamžitě po vstupu nacházel uvnitř sledované oblasti. Každá změna polohy a orientace avatara je pak zasílána transformačnímu uzlu, který přísluší průvodci. Důležitá je také aktualizace polohy sledované oblasti, která je zajištěna posledním z příkazů **ROUTE**. V něm si detektor přítomnosti sám sobě zasílá událost, která správně nastaví umístění citlivé oblasti.

Jak je vidět na obrázcích 5-18, poloha a velikost průvodce na obrazovce zůstávají nezměněny při jakémkoliv avatarově pohybu. Ke změně může dojít pouze tehdy, pokud uživatel použije standardní ovládací prvky prohlížeče a zvolí nové stanoviště. Tehdy prohlížeč teleportuje avatara na jiné stanoviště, aniž by zaslal potřebné události s geometrickými údaji směrem k průvodci.



Obrázek 5-18: Průvodce zachovává svoji polohu i vzhled bez ohledu na pohyb avatara

```
#VRML V2.0 utf8

Inline { url "../kap4/p-4-1.wrl" }        # svět s několika stolečky

DEF DOMA Viewpoint    { description "Normalni" }
DEF P ProximitySensor { size 100 100 100 }

DEF PRUVODCE Transform {
  children Transform {
    translation 1 0 -4  scale 0.2 0.4 0.2
    children Anchor {
      children Inline { url "../knihovny/ovladac-koule.wrl" }
      url "#DOMA"
    }
  }
}

ROUTE P.orientation_changed TO PRUVODCE.rotation
ROUTE P.position_changed    TO PRUVODCE.translation
ROUTE P.position_changed    TO P.center
```

Program P-5-14: Definice tělesa v roli průvodce s pomocí detektoru přítomnosti

6 Jen pro opravdové programátory?

V předchozí kapitole jsme uvedli všechny uzly, které umožňují definovat a provádět jednoduché dynamické akce. V několika příkladech jsme však viděli, že i tyto uzly mají svá omezení a že důmyslnější interaktivní práce vyžaduje ještě další prostředky. Vždy jsme se v takovém případě odvolali na speciální uzel, který nese název **`Script`** (česky *skript*). Jemu je věnována celá tato kapitola.

Co je to tedy za uzel, který umí to, co jiné uzly nedokáží? Co je to uzel, k němuž se obracejí tvůrci virtuálních světů pokaždé, když jim nestačí některý z padesáti „běžných“ uzlů VRML? V čem spočívají jeho výjimečné schopnosti?

Odpověď na tyto otázky je jednoduchá - uzel **`Script`** není ničím jiným, než schránkou na funkce napsané v nějakém programovacím jazyce. Můžeme tedy do něj vložit program, s jehož pomocí rozšíříme chybějící vlastnosti běžných uzlů a který zajistí mnohem bohatší škálu interaktivních možností. Skript je tak dobrý, jak dobrý je program v něm obsažený.

Do uzlu **`Script`** pochopitelně nelze vkládat programy v libovolném programovacím jazyku. V současné době jsou povoleny pouze dva jazyky. Prvním z nich je **Java** – moderní, rozsáhlý, objektově orientovaný jazyk, který je svými schopnostmi srovnatelný s klasickými programovacími jazyky, jakým je například C++. Druhým jazykem, jímž se definují funkce v uzlu **`Script`**, je jazyk **ECMAScript**. Toto jméno je oficiálním názvem jazyka Netscape JavaScript v podobě mezinárodní normy ISO [7]. Znají jej zejména tvůrci WWW stránek. Jde o jednoduchý, přímo interpretovaný jazyk, který se v některých rysech podobá jazyku Java, avšak dovoluje snadnou a intuitivní práci, byť s omezenou škálou prostředků. Krátké, jednodušší programy se proto zapisují v jazyku ECMAScript, zatímco důmyslné a rozsáhlejší funkce je nutno definovat jazykem Java.

Tabulka T-6-1 ukazuje, že skript má pouze tři základní parametry:

specifikace parametru		jméno	iniciální hodnota	význam
`exposedField`	**`MFString`**	**`url`**	**`[]`**	adresy souboru s obslužnými funkcemi či jejich přímý zápis v uzlu
`field`	**`SFBool`**	**`directOutput`**	**`FALSE`**	funkce mohou přímo zasílat události jiným uzlům
`field`	**`SFBool`**	**`mustEvaluate`**	**`FALSE`**	okamžité vyvolání funkcí po přijetí události uzlem

Tabulka T-6-1: Parametry uzlu **`Script`**

Nejdůležitějším parametrem je **`url`**, který obsahuje adresu souboru, v němž jsou zapsány potřebné funkce uzlu. Jako obvykle lze do tohoto parametru zapsat více adres pro případ nedostupnosti některé z nich. Jakmile jsou získána data z první dostupné adresy, zbylé adresy se ignorují. Častým případem je zapsání všech funkcí přímo do parametru **`url`**. To však lze jen v případě jazyka ECMAScript, který lze na rozdíl od jazyka Java interpretovat, tj. provádět rovnou, bez nutnosti překladu do souboru.

Síla skriptu spočívá v možnosti zavedení libovolného počtu *dalších parametrů*. V tomto smyslu se uzel podobá konstrukci **`PROTO`** z kapitoly 4, která však má jen omezené možnosti práce se svými parametry. Parametry, které se vyskytují v příkazu **`PROTO`**, jsou pouze přiřazovány příkazem **`IS`** parametrům dalších uzlů, a to bez jakékoliv změny. V uzlu **`Script`** jsou pak parametry zpracovávány vnitřními funkcemi. Platí tato pravidla:

1. parametr třídy **`field`** musí mít definovánu *iniciální hodnotu*. Funkce skriptu mohou tuto hodnotu měnit, parametr tedy slouží jako tzv. lokální proměnná.

2. ke každému parametru třídy **`eventIn`** musí existovat *stejnojmenná* funkce. Tato funkce se provede pokaždé, když parametr získá novou hodnotu, tj. když přijme událost. Každá taková funkce má proto dva vstupní parametry – přijímanou hodnotu a čas vzniku události.

3. z parametru třídy **eventOut** je vyslána událost pokaždé, když je přiřazena nová hodnota do *stejnojmenné* proměnné uvnitř libovolné z funkcí. Je-li uvnitř funkce přiřazena hodnota do téže proměnné několikrát, vyšle se z odpovídajícího parametru jen jediná událost s poslední hodnotou[13].

4. skript *nesmí* obsahovat parametr třídy **exposedField**. Toto omezení lze překonat zavedením trojice parametrů **field**, **eventIn** a **eventOut** s odpovídajícími názvy **x**, **set_x** a **x_changed**, jak bylo uvedeno na začátku kapitoly 5.

K prvnímu praktickému seznámení se skriptem slouží následující příklad P-6-1. Skript má jedinou schopnost – převést vstupní hodnotu typu **SFBool** (tj. **TRUE** nebo **FALSE**) na čísla 1 či 0. Zavedeme v něm tedy parametr třídy **eventIn** a pojmenujeme jej **set_bool**. Jméno parametru může být sice libovolné, ale je vhodné dodržovat zavedené konvence VRML a výběrem jména naznačit význam parametru. Parametr třídy **eventOut** proto podle této zvyklosti pojmenujeme **value_changed**.

```
Script {
      eventIn  SFBool   set_bool
      eventOut SFInt32  value_changed

      url "javascript:
          function set_bool (hodnota, cas)
          { if (hodnota) value_changed = 1; else value_changed = 0; }
      "
}
```

Program P-6-1: Skript jako konvertor datových typů

Podle výše uvedených pravidel je potřeba ke každému vstupnímu parametru napsat stejnojmennou funkci. V našem případě potřebujeme jen jedinou funkci, s názvem **set_bool**. Její definici umístíme přímo do skriptu a proto musí parametr **url** obsahovat text začínající klíčovým slovem **javascript**. Toto pravidlo má historické kořeny a paradoxně platí i přes skutečnost, že oficiální název jazyka je ECMAScript.

Funkce **set_bool** má ve svém záhlaví zapsány dva parametry, jejichž typ se neuvádí. Je totiž zřejmé, že typ prvního parametru **hodnota** je shodný s typem parametru **set_bool**, neboť jde o hodnotu vstupní události. Druhý parametr **cas** je vždy typu **SFTime** a obsahuje čas, ve kterém událost vznikla. Uvnitř funkce je pak přiřazována hodnota do proměnné **value_changed**, což způsobí vyslání události těm uzlům, které jsou se stejnojmenným parametrem skriptu propojeny konstrukcí **ROUTE**.

Vlastní funkce je jednoduchá a bude patrně pochopitelná i čtenáři, který není zběhlý v programování. Nyní je také zřejmé, že jméno této kapitoly je nadsázkou. Práce se skriptem není v žádném případě vyhrazena pouze odborníkům, ale je přístupná i těm, kdo mají s programováním jen základní zkušenosti.

Velkou výhodou jazyka ECMAScript je značná volnost při zápisu funkcí. Není třeba přísně dodržovat všechna formální pravidla, jazyk navíc automaticky provádí převody mezi proměnnými různých typů. Výše uvedenou funkci tak lze zapsat nejen bez nepoužívaného parametru **cas**, ale i v mnohem jednodušším tvaru:

```
function set_bool (hodnota)
            { value_changed = hodnota; }
```

V dalším textu se zaměříme výhradně na používání jazyka ECMAScript. Jeho jednoduchost a možnost vkládání funkcí přímo do těla uzlu **Script** nám dovolí ukázat řadu příkladů bez nutnosti podrobného vysvětlování pravidel a syntaxe celého jazyka. Naopak se zcela vyhneme práci s jazykem Java. Jeho použití již vyžaduje skutečné programátorské znalosti, tvůrce funkcí musí mít na počítači instalován překladač zdrojového jazyka do mezikódu. Práce s tímto jazykem přesahuje rámec této knihy a čtenáře odkazujeme na specializované publikace věnované jazyku Java.

[13] Čas vyslání události má hodnotu *totožnou* s časem události, která způsobila vyvolání funkce. Délka výpočtu uvnitř funkce je tedy považována ideálně za nulovou, byť ve skutečnosti každý výpočet trvá nenulovou dobu a ke skutečnému vyslání události dojde určitě se zpožděním.

Pro úplnost ukážeme, jak by vypadalo použití skriptu, jehož funkce jsou zapsány v samostatném souboru. Program P-6-1 by mohl vypadat takto:

```
Script { eventIn  SFBool   set_bool
         eventOut SFInt32  value_changed
         url [ "../ECMAScript/Bool2Int.js"
               "../java/Bool2Int.class" ]
}
```

Soubory, obsahující funkce v jazyce ECMAScript mívají příponu "**`js`**", přeložené funkce v jazyce Java příponu "**`class`**".

Jednoduché skripty, které navzájem převádějí data různých typů, jsou praktickým doplňkem pro řadu animací a interaktivních akcí. Jednu z nich ukazuje následující příklad P-6-2. V něm chceme, aby se vzhled virtuálního těleso zvýraznil pokaždé, když nad ním přejede kurzor. Ke správnému provedení této operace (nazývané anglicky *highlighting*) potřebujeme detektor dotyku a dvě reprezentace tělesa – normální a zvýrazněnou. Na základě informací dodávaných detektorem dotyku budeme přepínat mezi dvěma reprezentacemi tělesa.

Pokud bychom neměli k dispozici skript, nedokázali bychom požadovanou akci vůbec realizovat. Uzel **`Switch`**, který dovoluje přepínání reprezentací, dokáže přijmout vstupní události pouze typu **`SFInt32`**, tedy celočíselná data. Naproti tomu detektor dotyku poskytuje údaje časové a geometrické. Jediná cesta ke spolupráci těchto dvou uzlů spočívá v použití konverzního skriptu z příkladu P-6-1.

```
#VRML V2.0 utf8

Group { children [
    DEF DOTYK  TouchSensor {}
    DEF TELESO Switch {
      whichChoice 0
      choice [
        Shape {                  # normální vzhled
          geometry DEF KOULE Sphere {}
          appearance Appearance {material Material {diffuseColor 0 0 1 }}
        }
        Shape {                  # zvýrazněný vzhled
          geometry USE KOULE
          appearance Appearance {material Material {emissiveColor 0 0 1}}
        }
      ]
    }
    DEF PREVOD Script {
      eventIn  SFBool   set_bool
      eventOut SFInt32  value_changed
      url "javascript: function set_bool (hodnota)
                         { value_changed = hodnota; }
    "}
]}

ROUTE DOTYK.isOver         TO PREVOD.set_bool
ROUTE PREVOD.value_changed TO TELESO.whichChoice
```

Program P-6-2: Zvýraznění vzhledu tělesa s pomocí převodního skriptu

TIP: Umístění skriptu ve stromové hierarchii není důležité. Skript je schopen pracovat i tehdy, pokud je zařazen do nezobrazované větve stromu, např. k neaktivnímu potomku uzlu `Switch`.

6.1 Další parametry uzlu Script

Tabulka T-6-1 obsahuje dva parametry, které dosud nebyly blíže vysvětleny. Slouží k zajištění efektivity při zpracování skriptů. Je-li parametr **mustEvaluate** nastaven na hodnotu **FALSE**, prohlížeč nemusí reagovat na každou vstupní událost vyhodnocováním funkcí uvnitř skriptu. Může provést jen ty funkce, které zapisují hodnoty do výstupních parametrů propojených konstrukcí **ROUTE** s jinými uzly. Taková situace může nastat, když do jednoduchého virtuálního světa převezmeme skript s mnoha parametry třídy **eventOut**, původně navržený pro složité operace v jiných světech. Pokud budeme využívat jen některé z jeho výstupních parametrů, lze ušetřit čas potlačením výpočtů těch funkcí, které pracují s nepoužitými parametry. Riskujeme však, že vynecháním některých výpočtů nedojde například ke správnému nastavení lokálních proměnných uvnitř skriptu. Hodnota **TRUE** parametru **mustEvaluate** zajistí vždy korektní chování skriptu, ovšem za cenu možného zpomalení výpočtů.

Druhým parametrem ovlivňujícím efektivitu je **directOutput**. Používá se tehdy, pokud skript zasílá události dalším uzlům. Existují dva základní způsoby předávání dat. První způsob je realizován již známou konstrukcí **ROUTE**, která propojuje parametr skriptu s parametrem jiného uzlu (viz poslední řádek v programu P-6-2). V tomto případě nemá skript možnost ovlivnit jiné parametry uzlu nežli ty, se kterými je výslovně definováno propojení. Při druhém způsobu se uzel uvede mezi parametry skriptu, tedy jako parametr typu **SFNode** nebo **MFNode**. Skript pak může přistupovat k jeho parametrům nejen pomocí konstrukce **ROUTE**, ale i přímým zápisem v programovacím jazyku. Parametr **directOutput** určuje, zda je tento přímý zápis povolen nebo ne. Hodnota **TRUE** umožňuje přímou manipulaci se všemi parametry daného uzlu, ale od prohlížeče vyžaduje provádění časově náročnějších kontrol. Jde totiž o určité „obcházení“ zavedených postupů založených na konstrukci **ROUTE**, které by v některých případech mohlo způsobit chyby ve struktuře virtuálního světa.

Přímý zápis hodnot do parametrů zjednodušuje dynamické akce, jak ukazuje příklad P-6-3. V něm je realizován stejný virtuální svět jako v programu P-6-2, avšak bez zavedení přepínače a dvojí reprezentace potřebné pro zvýraznění vzhledu tělesa. Skript má v tomto případě definován parametr **objekt** typu **SFNode**, kterému je přiřazeno dané těleso konstrukcí **USE**. Jediná vnitřní funkce využívá lokální proměnné **zjasneni**, do které si podle hodnoty vstupní události připraví barvu pro zvýraznění tělesa. V normální situaci bude tato barva, hrající roli barevného světla vyzařované tělesem, obsahovat samé nuly. V případě zvýraznění se do ní dosadí barevné složky s totožnými hodnotami 0.3.

```
#VRML V2.0 utf8

Group { children [
  DEF DOTYK  TouchSensor {}
  DEF TELESO Shape {
       geometry Sphere {}
       appearance Appearance {material Material {diffuseColor 0 0 1 }}
  }
  DEF ROVNOU Script {
      field   SFNode  objekt    USE TELESO
      eventIn SFBool  set_highlight
      field   SFColor zjasneni  0 0 0
      directOutput TRUE
      url "javascript:
        function set_highlight (hodnota) {
          if (hodnota) zjasneni[0]=zjasneni[1]=zjasneni[2]= 0.3;
                  else zjasneni.r =zjasneni.g =zjasneni.b = 0;
          objekt.appearance.material.emissiveColor = zjasneni;
        }"
  }
]}

ROUTE DOTYK.isOver TO ROVNOU.set_highlight
```

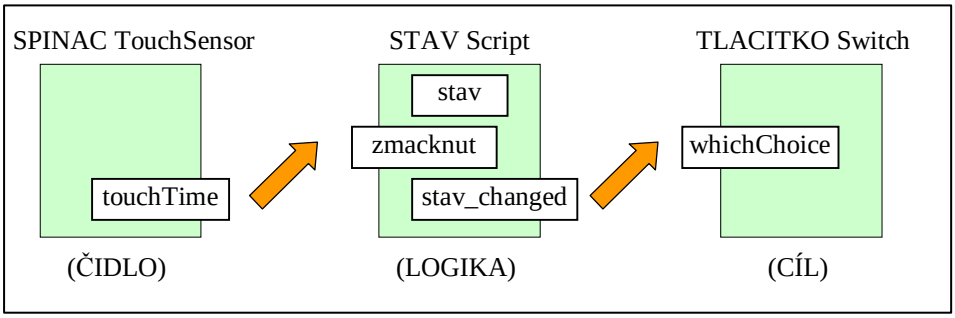
Program P-6-3: Zvýraznění vzhledu tělesa přímou změnou jeho parametrů

Nejzajímavější je poslední řádek ve funkci **set_highlight**. Ten je právě oním přímým zápisem hodnoty do parametru jiného uzlu. Všimněme si, jak je tento zápis mocný – ačkoliv skript **ROVNOU** má ve svém parametru zapsán pouze uzel **TELESO**, hodnota **zjasneni** se přiřadí do parametru uzlu **Material**, který je potomkem

uzlu **TELESO** „přes jednu generaci“. Poslední řádek funkce obsahuje tzv. násobnou tečkovou notaci, s jejíž pomocí se v objektových programovacích jazycích přistupuje k dílčím položkám objektové proměnné.

Pro zajímavost také ukazujeme, jakými způsoby se pracuje s proměnnou typu **SFColor**. Její tři barevní složky lze buď indexovat (od nuly do dvou) nebo k nim přistupovat přes tečkovou notaci a symbolická jména základních barevných složek.

Další příklad ukáže roli skriptu jako logického prvku ve schématu dynamických akcí. Vytvoříme spínací tlačítko, které při prvním stisku změní svoji barvu a polohu, při druhém stisku dojde k obnovení původního stavu. Je zřejmé, že k základnímu detektoru dotyku potřebujeme přidat pomocnou paměť, obsahující informaci o tom, zda je tlačítko zapnuté či vypnuté. Tato paměť musí být realizována skriptem. Ten vybere ze dvou reprezentací tlačítka tu, která odpovídá jeho aktuálnímu stavu.



Obrázek 6-1: Skript s lokální proměnnou

Dynamickému schématu práce s tlačítkem na obrázku 6-1 odpovídá program P-6-4. V něm si také všimneme, že v jazyce ECMAScript se k označení komentářů používá dvojice lomítek. Pokud bychom použili známé značky **'#'**, interpret jazyka ECMAScript by ohlásil chybu.

```
#VRML V2.0 utf8

Group {
  children [
    DEF SPINAC TouchSensor { }
    DEF TLACITKO Switch {
        whichChoice 0
        choice [ Inline {url "../knihovny/tlacitko0.wrl" }
                 Inline {url "../knihovny/tlacitko1.wrl" }
    ]}
    DEF STAV Script {
      eventIn  SFTime  zmacknut
      eventOut SFInt32 stav_changed
      field    SFBool  stav FALSE        # inicialně vypnuto
      url "javascript:
          function zmacknut() {
                   if (stav) stav = 0;   // vypnutí
                   else stav = 1;        // zapnutí
                   stav_changed = stav;  // vyslání události
          }"
    }
]}

ROUTE SPINAC.touchTime  TO STAV.zmacknut
ROUTE STAV.stav_changed TO TLACITKO.whichChoice
```

Program P-6-4: Tlačítko se dvěma stavy

Jména parametrů **stav** a **stav_changed** připomínají odvozená jména jediného parametru třídy **exposedField**. Aby však skutečně mohla označovat jeden kombinovaný parametr, musela by být doplněna vstupním parametrem **set_stav** a navíc by všechny tři parametry musely být stejného typu.

Uzly **Script** bývají velmi často používány ve spojení s konstrukcí **PROTO**. Nový uzel-prototyp obsahuje jen ty parametry, které jsou přímo potřebné pro komunikaci s jinými uzly, zatímco jeho vnitřní skript zajišťuje odpovídající zpracování událostí a používá rozsáhlejší množinu proměnných. Dříve, než tento přístup ukážeme na příkladu, seznámíme se s několika dalšími funkcemi jazyka ECMAScript svázanými se skripty.

6.2 Speciální funkce ECMAScriptu

Funkce uvnitř skriptů obsluhují buď jednotlivé vstupní události (tehdy mají název shodný se jménem příslušného parametru třídy **eventIn**) nebo slouží jako pomocné funkce pro dílčí výpočty. Navíc však existují tři funkce s předem definovanými jmény a vlastnostmi, které mohou být zahrnuty do libovolného skriptu:

1) **initialize ()** — Funkce se provede právě jednou, těsně před tím, než je svět prezentován uživateli.

   Jejím úkolem je nastavit iniciální hodnoty, provést případnou iniciální posloupnost příkazů.

2) **shutdown ()** — Funkce se provede právě jednou, po skončení prohlížení světa, tj. před přechodem do dalšího světa či WWW stránky.

   Existence funkce umožňuje, aby například byla zaznamenána poslední poloha avatara ve virtuálním světě a obnovena při jeho další návštěvě.

3) **eventsProcessed ()** — Funkce se provádí pokaždé, když je skriptu zaslána nějaká událost.

   Slouží ke snížení počtu dalších vysílaných událostí, protože může „akumulovat“ dílčí vstupní události různých typů a vyslat jednu výstupní událost až při dosažení určitého stavu, splnění podmínek. Kupříkladu trezor se třemi zámky nemusí vysílat informaci o svém stavu po každém zasunutí jednotlivého klíče, ale až po úplném odemčení.

Prázdné závorky za jmény funkcí říkají, že funkce nemají žádné parametry. Práce s těmito funkcemi je charakteristická pro složitější světy vytvářené zkušenějšími programátory. Pro běžné účely slouží především funkce **initialize**, které se budeme věnovat podrobněji.

Jestliže virtuální svět obsahuje několik skriptů s vlastními inicializačními funkcemi, nelze zjistit pořadí jejich provedení, protože je záležitostí konkrétního prohlížeče. Pokud je pro nás pořadí vyvolání skriptů důležité, můžeme definovat skript, který má jedinou funkci – inicializační. Ta zašle událost prvnímu uzlu v žádané posloupnosti skriptů propojených konstrukcí **ROUTE**, čímž dojde k provedení skriptů ve správném pořadí.

Velmi často se inicializační funkce používá k ošetření statických hodnot parametrů. V příkladu P-6-4 je takovým parametrem **stav**. Jeho iniciální hodnota **FALSE** říká, že použité tlačítko je vypnuté a je tedy reprezentováno prvním uzlem přepínače **TLACITKO**. Pokud by parametr **stav** měl mít hodnotu **TRUE**, museli bychom změnit i hodnotu parametru **whichChoice** přepínače. Tuto akci lze s výhodou provést automaticky v inicializační funkci, která reaguje na skutečně zapsanou hodnotu statického parametru **stav**.

Program P-6-5 ukazuje nejen použití inicializační funkce, ale definuje i prototyp univerzálního vypínače, pojmenovaný **Button**. V něm se předpokládá, že teprve při konkrétním použití prototypu budou zapsány dvě různé reprezentace vypínače jako potomci uzlu **Button**.

Prototyp vypínače obsahuje celkem tři uzly – detektor dotyku, přepínač a skript. Detektor dotyku je základem funkce vypínače. Do přepínače se ukládají příslušné geometrické reprezentace. Skript pak obsahuje několik parametrů. Trojice parametrů, jejichž jména jsou odvozena od názvu **zapnuto**, zajišťuje iniciální nastavení polohy vypínače (**zapnuto**), přepnutí do konkrétního stavu (**set_zapnuto**) a vyslání události informující o změně stavu vypínače (**zapnuto_changed**). Samostatný vstupní parametr **prepnuti** reaguje na události zasílané detektorem dotyku.

V programu P-6-5 jsou zavedena stejná jména pro parametry prototypu i pro parametry skriptu. Navzájem jsou provázána konstrukcí **IS**.

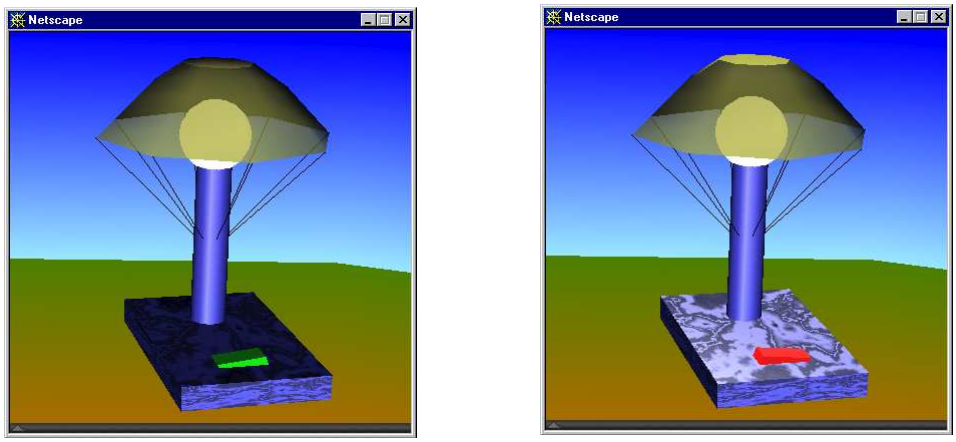
```
#VRML V2.0 utf8

PROTO Button [ field    MFNode tvary           []
               field    SFBool zapnuto         FALSE
               eventIn  SFBool set_zapnuto
               eventOut SFBool zapnuto_changed
]
{ Group {
  children [
    DEF STISK  TouchSensor {}
    DEF SWITCH Switch { choice IS tvary }
    DEF SCRIPT Script {
      directOutput TRUE
      field    SFNode objekt      USE SWITCH
      field    SFBool zapnuto     IS zapnuto     # aktuální stav vypínače
      eventIn  SFBool set_zapnuto IS set_zapnuto # nastavení vypínače
      eventOut SFBool zapnuto_changed IS zapnuto_changed # změna stavu
      eventIn  SFTime prepnuti                    # přepnutí vypínače
      url "javascript:
        function prepnuti () {
                 zapnuto = !zapnuto;            // změna aktuálního stavu
                 set_zapnuto ();                // změna tvaru
                 zapnuto_changed = zapnuto;     // výstup události
        }
        function set_zapnuto () { objekt.whichChoice = zapnuto; }
        function initialize  () { set_zapnuto (); }
      "
    }
  ]}
  ROUTE STISK.touchTime TO SCRIPT.prepnuti
}
```

Program P-6-5: Soubor **`protobutton.wrl`** *s prototypem dvoustavového vypínače*

Skript obsahuje tři jednoduché funkce, jejichž činnost je patrně zřejmá na první pohled. Funkce jsou většinou tvořeny jediným příkazem, s výjimkou funkce **`prepnuti`**, která změní stav přepínače na opačnou hodnotu (symbol vykřičník označuje negaci), vyvolá funkci **`set_zapnuto`** pro změnu geometrie a vyšle odpovídající výstupní událost.

Krok za krokem VI. – Lampička s vypínačem

Model lampičky, který od počátku této knihy postupně vylepšujeme, získává zavedením skriptu interaktivní vlastnosti. Prototyp vypínače z programu P-6-5 lze snadno přidat k prototypu lampičky a jeho výstupní parametr **zapnuto_changed** propojit s parametrem **on** světelného zdroje.



Obrázek 6-2: Zhasnutá a rozsvícená lampička s vypínačem

```
#VRML V2.0 utf8

EXTERNPROTO Button [ field    MFNode tvary
                     field    SFBool zapnuto
                     eventIn  SFBool set_zapnuto
                     eventOut SFBool zapnuto_changed
  ] "../knihovny/protobutton.wrl"

PROTO Lampa-VI [field SFColor  barvaSvetla      1 1 1
                field SFColor  barvaLampy       0.3 0.3 1
                field MFString texturaPodstavce "../knihovny/mramor.gif"]

{Group {
  children [
    Inline  { url "../knihovny/stinitko.wrl"  }
    ... # další geometrie lampičky

    DEF SVETLO SpotLight {        # žluté světlo
      color       IS barvaSvetla  radius      0.5
      location    0 0.3 0         direction   0 -1 0
      cutOffAngle 0.56            attenuation 0.5 0 0
      on FALSE
    }
    Transform {
      translation 0 0.015 0.1
      scale 0.02 0.01 0.01
      children DEF VYPINAC Button {
          tvary [ Inline {url "../knihovny/tlacitko0.wrl" }
                  Inline {url "../knihovny/tlacitko1.wrl" } ]
          zapnuto FALSE
      }
    }
  ]}
  ROUTE VYPINAC.zapnuto TO SVETLO.on
}
```

Program P-6-6: Prototyp lampičky s vypínačem

Na obrázku 6-2 vlevo je vidět vzhled lampičky při zhasnutém stavu, kdy má vypínač zelenou barvu. Vpravo je lampička rozsvícena, vypínač změnil svoji barvu i sklon. Nový prototyp lampičky je zapsán v programu P-6-6. Všimněme si, jak jednoduchým způsobem se přidají potomci k uzlu **Button** do parametru **tvary**. Prototyp lampičky zůstal téměř stejný jako dosud, pouze světlo bylo pojmenováno a propojeno jedinou konstrukcí **ROUTE** s vypínačem. Je sympatické, že odkazem na prototyp a jediným krátkým příkazem **Lampa** získáme poměrně realistický dynamický model schopný podle přání uživatele rozsvěcet a zhasínat světlo (viz program P-6-7).

```
#VRML V2.0 utf8

EXTERNPROTO Lampa [ field SFColor  barvaSvetla
                    field SFColor  barvaLampy
                    field MFString texturaPodstavce
  ] "../knihovny/protolampa.wrl#Lampa-VI"

Lampa {}
```

Program P-6-7: Použití prototypu lampičky

6.3 Další příklady skriptů

Je zřejmé, že šíře a rozmanitost použití skriptů je obrovská. Nemá jistě smysl předvádět desítky nových příkladů, protože by stejně nepokryly všechny požadavky tvůrců virtuálních světů. Na závěr této kapitoly proto uvedeme jen tři vybrané ukázky, v nichž se pracuje se skripty v různých rolích.

Generátor pohybu (po orbitě)

Jedna z typických aplikací skriptu spadá do oblasti animací. Pokud se ve virtuálních světech pohybují rozmanité předměty po geometricky jednoduchých drahách, je rozumné připravit si několik prototypů, které poslouží jako generátory určitého pohybu. Když potom přidáme k takovému prototypu objekt na místě potomka, objekt se začne pohybovat po předepsané trajektorii. Tímto způsobem lze definovat kyvadlový pohyb, přesuny po úsečce, rotační pohyby apod. Příklad P-6-8 představuje generátor pohybu po kruhové orbitě s volbou poloměru a rychlosti oběhu.

Je zajímavé, že skript zde má pouze jediný úkol – vypočítat iniciální vektor posunutí, který je posléze zapsán na vhodné místo do dvouúrovňové hierarchie transformací. Dvě transformace nad sebou jsou důležité pro správné provádění rotačního pohybu na kruhové dráze. Horní transformace **T1** je propojena s výstupním parametrem interpolátoru orientace, dolní transformace **T2** zajišťuje posunutí. V inicializační funkci skriptu vidíme, jak příkazem **new** vzniká nová proměnná určitého typu. Tato proměnná má obdobné vlastnosti jako parametr skriptu, avšak její rozsah a existence jsou omezeny jen na funkci, uvnitř které je definována.

Jednoduchý prototyp by jistě bylo možno vylepšovat. Zavedením časových parametrů bychom umožnili zahájit rotační pohyb zasláním události do časovače, kruhový tvar orbity bychom dokázali změnit na eliptický. V takovém případě bychom nevystačili jen s interpolátorem orientace, ale museli bychom průběžně měnit i vzdálenost tělesa od středu elipsy. Potřebný skript by obsahoval náročnější geometrické výpočty. Namísto do interpolátorů by proto bylo lepší zasílat události z časovače (parametr **fraction_changed**) přímo do výpočetního skriptu. Ten by vypočítával a vysílal události typu **SFRotation** a **SFVec3f**, obsahující výsledné natočení a posunutí objektu na orbitě.

```
#VRML V2.0 utf8

PROTO Kruhovy [ field SFTime  doba_obehu 10
                field SFFloat radius      2
                field MFNode  objekt     []
 ] {
     DEF T1 Transform { children
        DEF T2 Transform { children IS objekt }
     }
     DEF CASOVAC TimeSensor {
         cycleInterval IS doba_obehu
         loop          TRUE               # nekonečná smyčka
     }
     DEF SKRIPT Script {
         field    SFFloat radius IS radius
         eventOut SFVec3f translation
         url "javascript:
            function initialize() {
              posun       = new SFVec3f (radius, 0, 0);
              translation = posun;
         }"
     }
     DEF ROTACE OrientationInterpolator {
         key      [ 0, 0.25, 0.5, 0.75, 1 ]
         keyValue [ 0 0 1 0, 0 0 1 1.57, 0 0 1 3.14,
                    0 0 1 -1.57, 0 0 1 0 ]
     }
     ROUTE SKRIPT.translation       TO T2.translation
     ROUTE CASOVAC.fraction_changed TO ROTACE.set_fraction
     ROUTE ROTACE.value_changed     TO T1.rotation
}
```

Program P-6-8: Generátor pohybu v podobě prototypu se skriptem

Synchronizace časovačů (v semaforu)

V kapitole 5.3, věnované časovači, jsme uvedli, že jednoduchý požadavek na odstartování časovače po ukončení činnosti časovače předchozího není snadno realizovatelný. Časovače totiž po skončení své činnosti vysílají událost typu **`SFBool`** z parametru **`isActive`**, kterou nelze přímo použít jako startovní impuls pro další časovač. Vhodným řešením je opět skript, který je schopen z této události získat časový údaj a ten zaslat dále. Časový údaj je totiž druhým parametrem ve vnitřních funkcích obsluhujících vstupní události.

Ukázkový program P-6-9 obsahuje skript, který ovládá dokonce několik časovačů současně. Každý časovač řídí rozsvícení a zhasnutí jednoho ze tří barevných světel virtuálního semaforu. Každé z těchto světel je definováno prototypem, dovolujícím stanovit polohu a barvu světla a pomocí jednoho parametrem světlo rozsvítit či zhasnout.

Pro zajištění realistického chodu semaforu je třeba navrhnout takové časování, na jaké jsou zvyklí motoristé. Při změně červené barvy na zelenou svítí oranžové světlo po určitou dobu současně s červeným. Při návratu ze zelené na červenou však svítí oranžové světlo samostatně. Z tohoto důvodu potřebujeme zavést čtyři časovače, které budou synchronně pracovat v cyklu o celkové délce 8 vteřin. Na počátku rozsvítí první časovač červené světlo. Ve třetí vteřině se k němu přidá oranžové světlo spouštěné druhým časovačem. Ve čtvrté vteřině obě rozsvícená světla zhasnou a na tři vteřiny se rozsvítí zelené světlo sepnuté třetím časovačem. Poslední časovač pak rozsvítí na jednu vteřinu samotné oranžové světlo[14].

Skript přijímá událost na začátku každého hlavního cyklu. Podle přijatého času vypočítá startovní časy jednotlivých časovačů přičtením konstant a odešle je časovačům.

[14] Zdálo by se, že pro ovládání oranžového světla vystačíme s jediným časovačem s délkou cyklu jedna vteřina. Protože však vypočítáváme všechny startovní časy naráz, čas prvního rozsvícení oranžového světla by v takovém případě byl okamžitě přepsán časem druhého rozsvícení. Čtyři časovače jsou tedy v našem řešení zcela na místě.

Poznamenejme, že uvedené řešení není jedinou možností. Lze například navrhnout skript řízený jen jediným časovačem, který podle uplynulého času (parametr **`fraction_changed`**) bude sám přímo zapínat a vypínat jednotlivá světla.

```
#VRML V2.0 utf8

EXTERNPROTO Svetlo [ exposedField SFVec3f position
                     exposedField SFColor color
                     eventIn      SFBool  svit
  ] "../knihovny/svetlo-semafor.wrl"

Shape { geometry Box {size 2.5 7 0.5 }}         # rám semaforu
DEF CERVENA  Svetlo { position 0  2.2 0  color 1  0  0 }
DEF ORANZOVA Svetlo { position 0  0   0  color 1 .7 .3 }
DEF ZELENA   Svetlo { position 0 -2.2 0  color 0  1  0 }

DEF T  TimeSensor { cycleInterval 8 loop TRUE } # celkový cyklus
DEF T1 TimeSensor { cycleInterval 4 }           # červené světlo
DEF T2 TimeSensor { cycleInterval 1 }           # první oranžová
DEF T3 TimeSensor { cycleInterval 3 }           # zelené světlo
DEF T4 TimeSensor { cycleInterval 1 }           # druhá oranžová

DEF S Script {
    eventIn  SFTime cyklus
    eventOut SFTime cas2
    eventOut SFTime cas3
    eventOut SFTime cas4
    url "javascript:
      function cyklus (cas)
        { cas2 = cas + 3; cas3 = cas + 4; cas4 = cas + 7; }
    "
}

ROUTE T.cycleTime TO S.cyklus      # propojení skriptu s časovači
ROUTE T.cycleTime TO T1.startTime
ROUTE S.cas2      TO T2.startTime
ROUTE S.cas3      TO T3.startTime
ROUTE S.cas4      TO T4.startTime

ROUTE T1.isActive TO CERVENA.svit  # propojení časovačů se světly
ROUTE T2.isActive TO ORANZOVA.svit
ROUTE T3.isActive TO ZELENA.svit
ROUTE T4.isActive TO ORANZOVA.svit
```

Program P-6-9: Spouštění několika časovačů skriptem

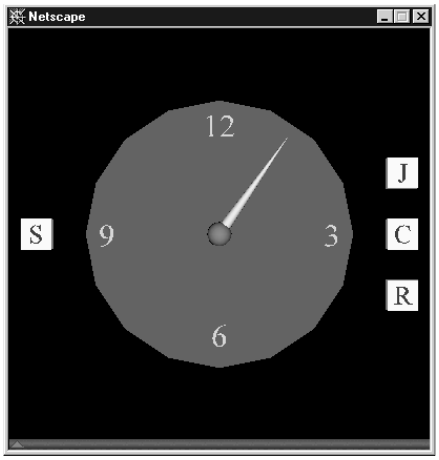
Snadno si dovedeme představit využití semaforu ve virtuální dopravní síti, kde červené světlo blokuje pohyb avatara (nastavením rychlosti jeho pohybu na nulu – viz parametr **`speed`** v uzlu **`NavigationInfo`**). Jen jako perličku pak uveďme myšlenku skriptu, který měří rychlost, se kterou avatar prochází virtuální svět. Pokud je avatar příliš rychlý a namísto důkladné prohlídky světem pouze prolétne, dostane pokutu. K realizaci takového měřiče stačí dva detektory přítomnosti na dvou vzdálených místech a skript, který vyhodnotí časové rozdíly mezi událostmi přijatými od obou detektorů.

Řízení animací

Také tato úloha je zaměřena na spolupráci skriptu a časovače. Ukazuje řešení často se vyskytující situace, v níž je třeba pozastavit a znovu rozběhnout déle trvající animaci, dynamickou akci. V mnoha případech potřebujeme mít k dispozici podobnou skupinu ovladačů, s jakými se setkáváme v audio-přehrávačích, tedy tlačítka pro zastavení (*stop*), spuštění z naposledy dosaženého místa (*play*) a vrácení na začátek (*rewind*).

V některých případech se setkáváme ještě s náročnějšími požadavky. Virtuální hodiny či stopky mohou obsahovat kromě tlačítka pro zastavení (*stop*) a vynulování (*reset*) dvě odlišná tlačítka pro další spuštění. První

z nich (*jump*) dopraví ručičku na místo, které odpovídá skutečnému času, který neustále běží na pozadí. Tímto způsobem se například měří tzv. *průběžné časy* při sledování mezivýsledků běžce-vytrvalce po každém kole. Druhé tlačítko pak rozběhne ručičku stopek z místa, kde se naposledy zastavila (*continue*). Tak se realizuje tzv. *přírůstkové měření* času.



Obrázek 6-3: Řízení stopek

Obrázek 6-3 ukazuje jednoduché stopky, jejichž ručičku budeme uvedenými způsoby ovládat. Vlastní model stopek je tvořen modrým válcem se čtyřmi popiskami v uzlech **`Text`** a dále středovou kuličkou. Kolem středu se otáčí jediná pohyblivá část stopek – ručička ve tvaru kuželu.

K řízení slouží čtyři tlačítka (malé kvádry s textovými popiskami). Levé tlačítko **S** zastavuje ručičku. Pravá tlačítka jsou určena k jejímu dalšímu rozběhnutí. Tlačítko **J** slouží k měření průběžného času, tlačítko **C** k měření přírůstkového času a tlačítko **R** představuje vynulování, tj. start nového měření od začátku.

Ovládací tlačítka jsou definována prototypem, obsahujícím kromě geometrických uzlů také jeden detektor dotyku. Časová událost z jeho parametru **`touchTime`** je předávána řídicímu skriptu.

Virtuální svět obsahuje jediný časovač, nazvaný **`MINUTKA`** (v programu P-6-10). Má nastaven nekonečně se opakující cyklus 60 vteřin a řídí interpolátor orientace, který otáčí ručičkou. Veškeré akce, týkající se pohybu ručičky, se proto týkají časovače. Jakmile se časovač zastaví, zastaví se i ručička. Časovač je ovládán hlavním řídicím skriptem, který zasílá události do jeho parametrů **`startTime`** a **`stopTime`**.

Skript **`RIZENI`** musí reagovat na pět různých vstupních událostí:

Inicializace **`initialize()`**	Nastavení času, od kterého se odvozuje chod stopek. Hodnota nula, použitá v našem příkladu, říká, že stopky byly spuštěny v 00:00, 1. ledna 1970. Tento čas je uložen v lokální proměnné **`pocatek`**. Prohlížeč vyhodnocuje aktuální stav stopek podle skutečného času zaznamenaného v počítači. Virtuální stopky tedy mohou ukazovat reálný čas.
Zastavení **`set_stop (kdy)`**	Do lokální proměnné **`zastaveno`** se zapíše čas, v němž avatar zastavil stopky. Tento čas získáme z parametru **`kdy`**. Dále vyšleme událost z parametru **`stop_changed`**, která způsobí zastavení časovače **`MINUTKA`**.
Vynulování **`set_reset (kdy)`**	Čas **`pocatek`**, od kterého se odvozuje chod stopek, změníme na hodnotu aktuálního času z parametru **`kdy`**. Současně tento čas vyšleme z parametru **`start_changed`** a zahájíme práci zastaveného časovače zasláním hodnoty minus jedna do parametru **`stop_changed`**.
Průběžný čas **`set_jump (kdy)`**	Vzhledem k tomu, že skutečný čas neustále plyne, stačí jen povolit práci zastaveného časovače **`MINUTKA`** zasláním hodnoty minus jedna do parametru **`stop_changed`**.
Přírůstkový čas **`set_continue (kdy)`**	Tato operace vyžaduje takovou změnu hodnoty proměnné **`pocatek`**, aby poloha stojící ručičky odpovídala aktuálnímu času počítače. K původní hodnotě **`pocatek`** je třeba přičíst délku prodlevy, po kterou byly stopky zastaveny. Velikost prodlevy získáme jako rozdíl aktuálního času (parametr **`kdy`**) a času zastavení stopek, který je uložen v proměnné **`zastaveno`**. Po provedení výpočtu zahájíme chod stopek podobně, jako tomu bylo u vynulování, tj. zasláním příslušných časových událostí z parametrů **`start_changed`** a **`stop_changed`**.

Vzhledem k většímu rozsahu definice virtuálních stopek uvádíme v programu P-6-10 jen vybrané části, které se týkají dynamických akcí.

```
DEF MINUTKA TimeSensor { loop TRUE      startTime 0
                         stopTime -1    cycleInterval 60
}
DEF OTACENI OrientationInterpolator {
      key [0, 0.5, 1]
      keyValue [0 0 -1 0, 0 0 -1 3.14, 0 0 -1 6.28]
}
DEF STOP Tlacitko { napis "S" translation -3  0 0 }
DEF RUNJ Tlacitko { napis "J" translation  3  1 0 }
DEF RUNC Tlacitko { napis "C" translation  3  0 0 }
DEF RUNR Tlacitko { napis "R" translation  3 -1 0 }

DEF RIZENI Script { eventIn  SFTime set_stop
                    eventIn  SFTime set_jump
                    eventIn  SFTime set_continue
                    eventIn  SFTime set_reset
                    eventOut SFTime start_changed
                    eventOut SFTime stop_changed
                    field    SFBool pracuje   TRUE
                    field    SFTime pocatek   0
                    field    SFTime zastaveno 0
  url "javascript:
  function initialize () { stop_changed = -1; start_changed = pocatek; }
  function set_stop (kdy)     // pozastaví chod ručičky
    { if (pracuje) { pracuje       = FALSE;
                     zastaveno     = kdy;
                     stop_changed  = kdy; }
    }
  function set_jump (kdy)     // ručička skočí na aktuální čas
    { if (!pracuje) { pracuje = TRUE;
                      stop_changed  = -1; }
    }
  function set_continue (kdy) // ručička pokračuje z místa zastavení
    { if (!pracuje) { pracuje       = TRUE;
                      pocatek       = pocatek + (kdy - zastaveno);
                      start_changed = pocatek;
                      stop_changed  = -1; }
    }
  function set_reset (kdy)    // ručička skočí na 12 hodin
    { if (!pracuje) { pracuje       = TRUE;
                      pocatek       = kdy;
                      start_changed = pocatek;
                      stop_changed  = -1; }
    }
"}
########## ovládání skriptu #########
ROUTE STOP.touchTime TO RIZENI.set_stop
ROUTE RUNJ.touchTime TO RIZENI.set_jump
ROUTE RUNC.touchTime TO RIZENI.set_continue
ROUTE RUNR.touchTime TO RIZENI.set_reset
########## řízení ručičky ########
ROUTE RIZENI.start_changed     TO MINUTKA.startTime
ROUTE RIZENI.stop_changed      TO MINUTKA.stopTime
ROUTE MINUTKA.fraction_changed TO OTACENI.set_fraction
ROUTE OTACENI.value_changed    TO RUCICKA.rotation
```

Program P-6-10: Složitější řízení animace skriptem

Zastavování a opětné spouštění animací má ve virtuálním prostředí velký význam. Prochází-li například avatar po ulici, po níž se blíží ničivý parní válec, avatar může vstoupit do nejbližšího domu a počkat, až válec přejede. Abychom nezatěžovali počítač zbytečnými výpočty, můžeme animaci pohybu válce zastavit po dobu, během níž je avatar v jiné části virtuálního světa. Po návratu avatara na ulici dokončíme animaci parního válce v souladu s uplynulým reálným časem.

Jindy naopak požadujeme, aby avatar spatřil celou animaci, například sekvenci reklam na virtuálním projekčním plátně. Tehdy můžeme detektorem viditelnosti zjistit, zda avatar odvrátil svůj zrak od reklamní plochy či zda odešel do jiné oblasti. Po návratu avatara pokračujeme v animaci tam, kde byla předtím zastavena.

7 Ach, ty pomalé počítače!

Jen málo lidí je plně spokojeno s výkonem svého počítače. Většina uživatelů si pod pojmem počítač představuje něco neuvěřitelně dokonalého, a proto jim čekání na výsledek výpočtu či na nalezení potřebného souboru přesahující jednu vteřinu přináší nemilé rozladění, téměř srovnatelné s nalezením housenky v bramborovém salátu. Naivní uživatelé se domnívají, že dnešní Počítač (psáno s velkým **P**) prostě zvládne všechno a to okamžitě. Pokud tomu tak není, počítač získává nálepku pomalého krámu, patřícího do starého železa.

Vstup do světa virtuální reality může být pro mnoho uživatelů podobným traumatem. Dlouhé čekání na načtení souboru, nepěkný vzhled hrubě vymodelovaných objektů, trhavé pohyby při animacích a při chůzi ve virtuálních světech – s tím vším se můžeme při troše smůly setkat. Ne vždy musí být důvodem malý výkon počítače. Často je příčinou neefektivně vytvořený virtuální svět. Aby uživatele neodradila hned první návštěva virtuálního světa, je dobré přemýšlet o tom, jak snížit potřebu výpočetního výkonu a jak docílit kvalitního vzhledu světa s minimem prostředků. Tato kapitola je proto věnována metodám, které zajišťují efektivní zpracování virtuálních světů a pomáhají rychlejšímu zobrazování.

Jak bylo uvedeno, při úvahách o efektivitě se můžeme zaměřit na několik kriterií:

1. zkrácení doby načítání souboru, snížení množství dat,
2. zrychlení zobrazování a manipulace ve virtuálním prostoru,
3. zlepšení vizuální kvality.

Tyto aktivity bývají často v protikladu. Lepšího vzhledu tělesa například docílíme zvětšením počtu plošek na jeho povrchu, čímž však prodloužíme dobu načítání a zpomalíme zobrazování. Přesto lze virtuální světy navrhovat s ohledem na všechna výše uvedená kriteria. Čas věnovaný úvahám o efektivitě se bohatě vrátí při práci s rozsáhlými virtuálními světy.

7.1 Načítání souborů

Zdálo by se, že v této oblasti se nedá příliš mnoho zlepšit – jestliže virtuální svět obsahuje dvě stovky těles, musíme tyto dvě stovky v každém případě načíst a poté zobrazit. Lze tedy vůbec nějak ušetřit čas? Odpověď je naštěstí kladná a co více, cest k úspoře času při načítání je hned několik. Některé skutečně snižují množství přenášených dat, jiné zkracují dobu mezi načtením souboru a vznikem prvního obrázku v okně. V té době probíhají v prohlížeči iniciální výpočty a předzpracování dat, obojí lze ovlivňovat použitím správných technik při tvorbě virtuálního světa.

Využívání prototypů i konstrukcí `DEF` a `USE`

Oba tyto přístupy jsou základním prostředkem pro snižování objemu přenášených dat. Práce s prototypy navíc nenápadně podporuje autory v tom, aby definovali univerzální prototypy s možností násobného použití ve více virtuálních světech. Tím jim šetří budoucí práci.

Používání `DEF` a `USE` je nezbytné při opakovaném použití obrazových textur, zvukových souborů a video sekvencí, stejně dobře se hodí při definování materiálů, souřadnic vrcholů a normálových vektorů.

Rozdělení světa do několika souborů

Pokud bude návštěvník vstupovat do virtuálního prostoru vždy v jednom místě, je vhodné definovat v hlavním souboru pouze objekty v nejbližším okolí vstupního stanoviště a vzdálenější části světa vkládat pomocí uzlů `Inline`. Návštěvník tak velmi rychle spatří první objekty světa a zatímco se jim věnuje, postupně se načítají další.

Moderní prohlížeče jsou schopny načítat několik souborů najednou. To má velký význam při čtení dat z poměrně pomalé sítě Internet. Současné čtení více souborů bývá rychlejší než jejich postupné zpracování. Počet souborů, které tvoří jeden virtuální svět, by však neměl přesáhnout několik desítek.

Využívání násobné reprezentace pomocí `LOD`

Pro každý jen trochu složitější objekt je správné definovat několik reprezentací v různých detailech zapsaných v uzlu `LOD`. Když detailní reprezentace vkládáme uzlem `Inline`, prohlížeč může odložit jejich

přečtení až do doby, kdy má více času. Nejjednodušší reprezentace by však měla být obsažena přímo v hlavním souboru, aby byl objekt vidět okamžitě po vstupu do virtuálního světa.

Definování více reprezentací stojí autora čas a samozřejmě zvyšuje množství celkově přenášených dat. Přináší však urychlení, a to jak při načítání, tak zejména při prohlížení světů. Z tohoto důvodu se použití uzlů **`LOD`** vždy vyplatí.

Důsledné definování pomocných obálek (bbox)

Rodičovské uzly, které mohou mít velké množství potomků, obsahují parametry **`bboxSize`** a **`bboxCenter`** pro definování obálky ve tvaru kvádru. Tato obálka obklopuje geometrii všech potomků a je jedním z důležitých prostředků pro urychlení zobrazování. Jakmile se totiž avatar natočí tak, že mu obálka zmizí ze zorného úhlu, prohlížeč se okamžitě přestane věnovat desítkám či stovkám geometrických objektů uvnitř této obálky.

Prohlížeče proto automaticky vyhodnocují rozměry potomků a výsledné hodnoty zapisují do obálky v rodičovském uzlu. Tyto výpočty probíhají ve fázi načítání souborů. Když velikost obálky sami definujeme, ušetříme prohlížeči čas. Správně definovaná obálka v uzlu **`Inline`** je výhodná i pro návštěvníka virtuálního světa – pokud prohlížeč ještě nenačetl potřebný objekt, zobrazí dočasně na jeho místě hrany kvádru reprezentujícího obálku. Návštěvník tak získá informaci o tom, že v daném místě zakrátko přibude do scény objekt o dané velikosti.

Existuje celkem šest uzlů, obsahujících parametry pro definici obálky: **`Anchor`**, **`Billboard`**, **`Collision`**, **`Group`**, **`Inline`**, **`Transform`**.

Malá velikost textur

Obrazové textury v uzlu **`ImageTexture`** představují vždy velké množství dat. Pokud to je možné, používejme takové textury, které lze nanášet opakovaně. Jejich základní velikost by v takovém případě neměla přesáhnout rozměry 128 `x` 128 pixelů.

Namísto rozsáhlé textury **`PixelTexture`** je možno s výhodou použít skript, který texturu vytvoří algoritmem a to teprve tehdy, když je potřeba ji zobrazit. Algoritmus většinou zabere v souboru méně místa než úplná textura. Typickým příkladem jsou textury šachovnice, šrafování, písku apod.

Správná definice plošných objektů

Základní tělesa, jakými jsou koule, kvádry, kužele a válce, lze použít jen pro modelování jednoduchých objektů. Složitější tělesa se definují povrchovými ploškami, jichž bývají desítky až stovky. Proto je třeba věnovat pozornost jejich správnému zápisu. Ten by měl splňovat iniciální nastavení parametrů **`solid`**, **`convex`** a **`ccw`** na hodnotu **`TRUE`**. Tehdy prohlížeč data přímo načte do svých vnitřních datových struktur. Při jiném nastavení parametrů musí prohlížeč plošky upravovat tak, aby jejich orientace a další vlastnosti byly shodné s ostatními. Tyto přepočty se provádějí při načítání objektů.

Velkou úsporou je definování plošek jako konvexních útvarů, ba ještě lépe výhradní použití *trojúhelníků*. Rozdělení plošek na trojúhelníky přímo v souboru sice zvyšuje množství načítaných dat, ale celkově znamená úsporu. Plochy obecného tvaru totiž prohlížeč v iniciální fázi vždy rozděluje na trojúhelníky. Tento postup je časově náročný, navíc nemusí vždy vést na optimální tvar vzniklých trojúhelníků. Někdy vznikají úzké, dlouhé trojúhelníky, které zpomalují zobrazování.

Pokud to není vysloveně nutné, je lépe do souboru nezapisovat normálové vektory a ponechat na prohlížeči, aby podle hodnoty parametru **`creaseAngle`** vypočítal normály sám. Tím sice prodloužíme iniciální fázi, ale významně snížíme velikost souboru, v němž normálové vektory tvořené trojicemi čísel s desetinnou tečkou mohou zabírat až třetinu celkového objemu dat.

Všechna tato doporučení se týkají uzlů **`ElevationGrid`**, **`Extrusion`** a **`IndexedFaceSet`** (u výškové mapy ovšem nenalezneme parametr **`convex`**).

Krátké zvukové soubory

Iluzi virtuálního prostředí výrazně umocňuje používání rozličných zvuků. Namísto dlouhých zvukových záznamů je správné co nejvíce využívat jednorázových zvuků doprovázejících konkrétní akce (cink, bum, klap) nebo kratší zvuky opakovat (zpěv ptáků na zahradě, šplouchání vody v bazénu, hukot dopravy na ulici).

Používání komprese

Po úspěšném navržení virtuálního světa bychom vždy měli výsledné soubory zkomprimovat programem **gzip**. Je schopen zmenšit velikost souboru typicky pětkrát až desetkrát a významně tak přispět ke zrychlení načítání souborů ze sítě.

7.2 Rychlost zobrazování a pohybu

Dříve, než uvedeme jednotlivé postupy a doporučení, zastavíme se u rychlosti zobrazování. Tato rychlost je udávána počtem snímků (nových obrázků) za vteřinu, anglickou zkratkou je **fps** (*frames per second*) Pro lidské oko se pohyb jeví plynulým, je-li rychlost větší než 24 fps. Znamená to, že všechny děje ve virtuálním prostoru by měly být zobrazovány touto, případně vyšší frekvencí. Obyčejné počítače dosahují frekvence 24 fps jen stěží, naštěstí i rychlost kolem 10 fps je ještě přijatelná.

Rychlost zobrazování nelze u všech prohlížečů snadno zjistit. Často jsme závislí jen na subjektivním odhadu. Příjemnou výjimkou je prohlížeč CosmoPlayer, který po stisku klávesy **'='** začne rychlost průběžně vypisovat do levého dolního rohu zobrazovacího okna. Dobrý autor virtuálních světů proto sleduje, s jakou rychlostí jsou jeho výtvory zobrazovány a snaží se docílit co největšího počtu snímků za vteřinu. Připomeňme, že rychlost závisí nejen na kvalitě procesoru, ale také na schopnostech grafické karty a na velikosti zobrazovacího okna.

Následující přehled uvádí metody, které jsou používány profesionálními návrháři virtuálních světů. Jejich dodržováním lze výrazně zvýšit rychlost zpracování světa.

Šetření počtem ploch

Začátečníci rádi využívají základních těles i tam, kde by bylo na místě definovat množinu ploch. Typickou chybou je zápis geometrie billboardu pomocí kvádru namísto obdélníku určeného uzlem **IndexedFaceSet**. Je pravda, že z hlediska psaní je definice kvádru úsporná, ale ve skutečnosti obsahuje šestkrát více ploch.

Jinou chybou je povolení tvorby podstav u kuželu a válce v případě, že tato tělesa stojí na podložce a avatar podstavy nikdy nemůže spatřit. Příkladem je sloup antického chrámu stojící na kamenném podstavci a podpírající střechu. Pokud je parametrům **bottom** a **top** ponechána implicitní hodnota **TRUE**, prohlížeč zbytečně vytváří obě podstavy, tvořené až desítkami trojúhelníků. Správným nastavením parametrů na hodnotu **FALSE** lze uspořit téměř polovinu z celkového počtu trojúhelníků.

Omezení používání textu

Uzly **Text** znamenají pro prohlížeč náročnou práci, protože každé písmenko je převedeno na mnoho malých trojúhelníků. Pokud je to možné, je lepší všechny nápisy nahradit texturami.

Musejí-li být do virtuálního světa zařazeny uzly **Text** (například z důvodu dynamických změn textových řetězců), pak by měly vždy být potomky uzlu **LOD**. Z určité vzdálenosti se každý text stane nečitelným a může být nahrazen jednoduchou texturou nebo linkou.

Obarvujeme-li nápisy, pak ze všech materiálových parametrů je dobré definovat právě jediný – **emissiveColor**. Nápis se stane jasným a dobře čitelným bez ohledu na světelné podmínky.

Práce se světly a materiály

Čím více je definováno parametrů v uzlu **Material**, tím složitější výpočty se provádějí pro každý světelný zdroj osvětlující povrch daného tělesa. Pokud je na povrch mapována textura, je lepší materiál zcela vynechat, což má za následek vypuštění výpočtů osvětlení a tedy zrychlení zobrazování. Kdybychom však materiál v souboru uvedli prázdný, tj. pouze s přednastavenými parametry, odraz světla na povrchu by se vyhodnocoval. V tomto případě tedy není vhodné „myslet na zadní kolečka“ a ke každé nové geometrii ihned zapisovat materiál s tím, že později jej nějak nastavíme.

Ve zdroji světla typu reflektor (**SpotLight**) způsobí nastavení parametru **beamWidth** na větší hodnotu než **cutOffAngle** vytvoření jasně ohraničeného světelného kuželu a přinese úsporu složitých výpočtů útlumu jasu paprsků.

Množství výpočtů úzce souvisí s počtem světelných zdrojů. Působnost zdroje rovnoběžného světla (**DirectionalLight**) je naštěstí omezena jen na jeho sourozence, ovšem pro zbylé dva zdroje (**PointLight** a **SpotLight**) jsou prováděny výpočty při zobrazování povrchu všech těles virtuálního

světa. Je proto správné nejen pečlivě nastavovat dosah světelných paprsků, ale především zdroje světla vypínat (parametr **on**) ihned, jakmile se od nich avatar vzdálí.

Vypínání animací

Animace, které se odehrávají mimo zorný úhel avatara nebo ve větší vzdálenosti, jsou značnými „žrouty“ výpočetního výkonu. Proto je správné každou animaci doplnit detektorem viditelnosti nebo přítomnosti (**ProximitySensor**, **VisibilitySensor**) a ve správnou chvíli ji zastavit. Zkušený tvůrce virtuálních světů zařadí detektor viditelnosti automaticky ke každému složitějšímu prototypu, takže nově definovaný uzel se bude samostatně vypínat ihned, jakmile zmizí ze zorného pole avatara.

Práce se stupněm detailu

Jak bylo řečeno v kapitole 3, uzel **LOD** je základním prostředkem pro docílení vysoké rychlosti zpracování. Při zadávání vzdáleností, ve kterých dochází k přepnutí reprezentací (parametr **range**), je vhodné umožnit prohlížeči automatickou volbu vzdálenosti tak, jak je to uvedeno v příkladu P-3-26.

Efektivita skriptů

Parametry **directOutput** a **mustEvaluate** ve skriptech by měly být co nejčastěji nastaveny na **FALSE**, v opačném případě se může výrazně zhoršit rychlost zpracování a prohlížení virtuálního světa.

Nastavení zdrojů zvuku

Není-li bezpodmínečně nutné používat prostorový zvuk, je lépe nastavit v uzlu **Sound** parametr **spatialize** na **FALSE** a obě charakteristické elipsy ztotožnit a nastavit do tvaru koule. To je nanejvýš vhodné, když zavádíme cyklicky se opakující zvuky v pozadí.

7.3 Vizuální kvalita

Přes veškeré snahy o co nejmenší délku souborů a nejrychlejší zpracování nesmíme zapomínat na to, že návštěvníci našich světů očekávají příjemné virtuální prostředí, ve kterém se většinou tělesa podobají skutečným předmětům a také se tak chovají a pohybují. Následující postupy směřují k tomu, aby se návštěvníci ve virtuálních světech dobře cítili a nemuseli se pracně dohadovat, co asi mohou znamenat ty hranaté, nepřirozeně poskakující věci. K dobrému pocitu přitom patří i poskytnutí správné navigační podpory.

Objekty v několika reprezentacích

Ani v této části si neodpustíme zmínku o uzlu **LOD**. Používání tohoto uzlu by se mělo stát zcela přirozeným pro každého autora rozsáhlejších virtuálních světů. Pouze díky jemu lze tvořit světy složené z desítek či stovek složitých objektů, jejichž reprezentace je automaticky zjednodušována s ohledem na vzdálenost od avatara.

Zákaz vstupu do nevhodných prostor

Je správné obklopit těleso popsané mnoha ploškami pomocnou geometrií zapsanou do parametru **proxy** v uzlu **Collision** a urychlit tak vyhodnocování kolizí s tímto tělesem. Vhodné je i zavedení zvukových efektů (**AudioClip**) při nárazu do takto definovaných překážek propojením parametru **collideTime** s parametrem **startTime**. Takovým způsobem také dokážeme odradit návštěvníka od zkoumání těch detailů světa, které nejsou určeny k podrobnému prohlížení. Nedovolíme mu ani dostat se do úzkých zákoutí, ze kterých by se těžko vracel na hlavní trasu.

Zavedení navigačních prvků

V příkladu P-5-14 jsme ukázali, jak definovat průvodce, tj. objekty, která provázejí avatara uvnitř světa a dovolují mu v každém okamžiku vyvolat nápovědu či přesunout se na určité stanoviště. Pokud nebudeme používat průvodce, je správné umístit do rozsáhlejších světů cedulky (šipky, nápisy) s uzlem **Anchor** směřující na následující a předchozí stanoviště. Z každého stanoviště by měla být vidět alespoň jedna taková aktivní směrovka.

Informační nápisy mohou být pro lepší čitelnost umístěny na billboardech tak, aby je návštěvník viděl vždy v kolmém pohledu. Na zvláště zajímavá místa nebo objekty je vhodné upozornit samostatným poutačem, blikajícím světlem apod.

Aby avatar nezapomněl aktivovat tlačítka zahajující dynamické akce, použijeme postup z příkladů P-6-2 nebo P-6-3, v nichž dojde k okamžitému zvýraznění tlačítka, jakmile přes jeho obraz přejede kurzor. Podobně můžeme zvýrazňovat důležité objekty, když se k nim avatar přiblíží (viz detektor přítomnosti).

Přirozený pohyb v animacích

Dobře rozmyšlená dynamika pohybu v dynamických akcích dokáže zásadním způsobem zlepšit dojem z virtuálního světa. Obecně platí, že kvalitní animace potlačí špatný dojem z nekvalitního vzhledu modelovaného tělesa, avšak obráceně to neplatí. Tvary objektů mohou být jen symbolické (hlava v podobě koule, trup jako kvádr, nohy a ruce jako válce či množiny čar), avšak pohyb by měl být realistický.

Jednorázové pohyby, např. otevírání dveří a oken, by nikdy neměly mít konstantní rychlost. Přirozené je zavést zrychlení na začátku a zpomalení na konci pohybu. U pružných těles je efektním a přitom snadno proveditelným trikem deformace (parametr **`scale`** rodičovského uzlu **`Transform`**) při odrazu a dopadu.

Pohybem naznačíme i hmotnost virtuálních objektů. Těžká tělesa se pohybují pomalu, lehká rychle s případnou změnou dráhy.

Živé objekty (zvířata, ptáci, umělé příšerky) téměř nikdy nestojí bez hnutí na místě, vždy vykazují mírný pohyb – kývají se, natáčejí se, koulejí očima apod.

Opatrné zacházení s parametry uzlů

Začátečníkům se při některých animacích stává, že interpolátory vygenerují takové číselné hodnoty, které nejsou pro určité parametry povoleny. Příkladem je pokus o zmenšení objektu až na úplné „nic“ pomocí postupného snižování hodnot parametru **`scale`** v uzlu **`Transform`** až na nulu. Tento postup je špatný hned ze dvou důvodů. Prvním je skutečnost, že prohlížeč se stále pokouší takto minimalizovaný objekt zobrazovat a podle polohy avatara přepočítává jeho souřadnice, čímž vykonává zcela zbytečnou práci. Druhý důvod je ještě zřetelnější – většina prohlížečů ohlásí uživateli, že došlo k chybnému nastavení parametru. Kvalita virtuálního světa tím v uživatelových očích jistě rázem poklesne.

Správné řešení takové situace spočívá například v zavedení skriptu, který řídí jak interpolátor, tak uzel **`Switch`**. Interpolátor zmenšuje těleso až do velmi malé, avšak nenulové velikosti. Po ukončení interpolace zajistí skript přepnutí přepínače na prázdnou reprezentaci zasláním hodnoty minus jedna do jeho parametru **`whichChoice`**.

8 Kam kráčíš, virtuální realito?

Jazyk VRML je prvním vážným pokusem o uchopení mladé a dynamicky se rozvíjející aplikace výpočetní techniky. Virtuální realita překonává svá dětská léta, prochází zákonitými obdobími nekritického obdivu i skeptického odmítání a postupně nachází své uplatnění v mnoha oblastech. V takové situaci je velmi obtížné předpovídat budoucnost, protože ta většinou překoná i nejsmělejší odhady, zatímco jindy naopak nesplní mnohá očekávání. Přesto se můžeme pokusit odhadnout, jak bude vypadat virtuální realita v nejbližší budoucnosti. Podkladem k těmto odhadům jsou jednak aktivity vědeckých pracovišť, jednak úsilí programátorských a vývojových týmů velkých firem.

V této kapitole nejprve uvedeme očekávané hlavní směry dalšího vývoje jazyka VRML, poté se zmíníme o jiných přístupech k popisu virtuální reality a na závěr popíšeme různé typy větších systémů pro virtuální realitu.

8.1 Další vývoj jazyka VRML

Specifikace jazyka VRML vznikla jako společné dílo řady špičkových firem, byť hlavním impulsem byl návrh firmy Silicon Graphics s názvem Moving Worlds. Dalšímu rozvoji jazyka se v současné době věnuje otevřené a nezávislé sdružení *VRML Consortium*, složené ze zástupců desítek firem i jednotlivců. Sdružení úzce spolupracuje s mezinárodní standardizační organizací ISO.

Pod pojmem další vývoj jazyka si nemusíme představovat vznik nových verzí. Takový vývoj by byl jistě nežádoucí, protože by znamenal přepracování existujících dat do nového tvaru. Namísto toho jsou připravována rozšíření ve smyslu nových uzlů pro speciální aplikace, zavedení dalších způsobů přenosu, reprezentace a vizualizace dat. Z programátorského hlediska lze říci, že další vývoj přinese vznik nových vrstev nad základním jazykem VRML. Brzy proto můžeme očekávat rozšíření v následujících oblastech:

Proudový přenos dat

Jakmile je virtuální svět načten do prohlížeče, nelze jej měnit jinak, nežli dynamickými akcemi, které jsou v něm obsaženy. Některé aplikace pracující v reálném čase vyžadují, aby data virtuálního světa (polohy objektů, souřadnice vrcholů a normál, vzhled textur apod.) byla měněny i poté, kdy je svět načten. Taková dat mohou být například snímána na jiném místě Internetu ze skutečného světa nebo vypočítávána samostatnými simulačními programy. K tomu je třeba zavést proudový přenos dat (angl. *streaming*) a definovat způsoby, jakými měnit a doplňovat existující virtuální prostředí.

S dalším použitím proudového přenosu dat VRML se brzy setkáme ve standardu MPEG-4. Zkratka MPEG je všeobecně známa jako název formátu pro uchování a přenos číslicového televizního signálu - obrazových a zvukových dat. Ve formátu MPEG se zapisují filmy na CD-ROM a DVD. Verze 4 bude obsahovat další vrstvy, v nichž budou ukládány symbolické informace o rovinných a prostorových objektech. Na obrazovce budeme moci sledovat filmy tvořené kombinací video snímků a virtuálních objektů VRML.

Jiný pohled na proudový přenos dat se týká efektivity uzlu **LOD**. V posledních letech byly vyvinuty rychlé metody, které umožňují průběžné zjemňování detailů na povrchu těles v reálném čase. V uzlu **LOD** tedy nebude muset být několik různých reprezentací, ale postačí jediná univerzální množina vrcholů a ploch, z níž se bude odvozovat vzhled tělesa podle vzdálenosti. I zde lze využít proudového přenosu dat – iniciální množina ploch je načtena ihned a podrobnější informace se přenášejí pouze tehdy, je-li to potřeba, tj. když se avatar přiblíží k danému objektu.

S proudovým přenosem dat je úzce spjata otázka efektivní komprese a binárního kódování dat, zejména souřadnic vrcholů, které představují až 90 % objemu souborů. Existující textový popis a univerzální komprese programem *gzip* by bylo možno zlepšit zhruba dvakrát zavedením speciálních komprimačních postupů. Jejich zavedení by ovšem znamenalo doplnit existující prohlížeče o nové dekodéry.

Víceuživatelské prostředí

Jakmile dovolíme více uživatelům vstupovat do téhož virtuálního prostředí, musím věnovat pozornost jejich vzájemné komunikaci. Pomineme nyní technickou stránku problému průběžné aktualizace polohy všech avatarů ve všech prohlížečích zobrazujících stejný svět a zaměříme se na další stránky této zajímavé aplikace virtuální reality, nazývané anglicky *distributed multi-user virtual reality*.

Z uživatelského hlediska je velmi lákavá představa, že virtuální svět je místem, ve kterém se mohou setkávat lidé propojení Internetem. Ve světě existuje již několik virtuálních měst, kde lidé stavějí své virtuální domy z prefabrikovaných dílů, vyvěšují billboardy zvoucí k návštěvě a s pomocí mikrofonu či dokonce kamery připojené k počítači si povídají s dalšími podobně „postiženými“ uživateli. Není to nic jiného, než setkávání známe pod názvem *chat*. Původně textová komunikace se vyvinula přes tzv. internetovou telefonii a video konference až po kontakty v počítačovém trojrozměrném prostředí.

Aktivity, týkající se víceuživatelského prostředí a VRML, však jdou ještě dál. Avataři, kteří se ve virtuálním prostředí setkávají, musejí mít jednotně definovány základní reakce a měli by používat mezinárodně srozumitelná gesta, např. pro projevení zájmu o rozhovor, souhlas, nesouhlas apod. Reakce jsou vždy reprezentovány změnami geometrie řízenými skriptem. Od skriptu je jen krůček k rozsáhlejším inteligentním programům, které dovolí avatarovi zůstat ve virtuálním světě i poté, kdy se lidský uživatel odhlásil z Internetu či si jen odskočil na oběd. Samostatný avatar se může aktivně pohybovat ve virtuálním prostředí jako robot, zjišťovat různé informace a své poznatky později sdělit skutečnému uživateli. Může také vykonávat určité akce, například sbírat vzorky hornin na virtuální planetě. Teoreticky může uživatel vyslat do virtuálního světa i několik takových robotů.

Je zřejmé, že zde virtuální realita významným způsobem překračuje hranici počítačových her a dává prostor pro aplikovaný výzkum v oblastech robotiky, umělé inteligence, sociálního chování apod. Dává také vzniknout zcela novým pojmům, které byly donedávna doménou autorů příběhů science fiction. Následující tabulka uvádí některé z těch, o kterých brzy budeme slýchat častěji:

agent, *bot*	inteligentní stvoření, které vykonává určitý úkol nebo zjišťuje informace pro uživatele. Někdy se slovem *bot* označuje samotný inteligentní počítačový program, zatímco *agent* je jeho vizuální reprezentací ve virtuálním prostředí. Pojem *bot* je zkratkou slova *robot*.
biot	objekt s prvky umělého života a umělé inteligence, ať již představující zvíře, hmyz či rostlinu. Někdy se takové objekty chovají zcela samostatně, jindy pouze v omezeném rozsahu interagují s nejbližším avatarem. Zvláštním případem je *virus*, tj. *biot* s destruktivním chováním.
klon	avatarova kopie zanechaná v nějaké části virtuálního světa

Studiem víceuživatelského virtuálního prostředí se zabývá mnoho firem a výzkumných pracovišť. Ostatně jde o směr, který byl původně stanoven jako další krok v rozvoji jazyka VRML. Jestliže první generace jazyka umožňovala popisovat statické virtuální světy, druhá generace světy dynamické, pohyblivé *(Moving Worlds*), pak třetí generace by měla definovat živoucí, obydlené světy (angl. *Living Worlds*). Proto se někdy tento směr vývoje neoficiálně nazývá VRML 3.0.

Podpora programovacích jazyků

Uzel **`Script`** dovoluje, aby bylo chování virtuálních objektů obohaceno o programy psané v jazycích Java a ECMAScript. Stále však virtuální svět zůstává samostatným, bez možnosti užší spolupráce s jinými programy. Snahou vývojářů je definovat programátorské rozhraní, které by umožnilo zasílat události virtuálnímu světu z jiných programů, typicky psaných v jazyce Java. Na základě takové komunikace by bylo možno upravovat existující světy a vytvářet nové objekty externími programy. Tím by se usnadnil vznik jednoduchých editorů VRML, jejichž nedostatek je v současné době citelný, neboť mnoho autorů je při tvorbě světů odkázáno na nejjednodušší textový editor. Problém spolupráce s jinými programy je velmi aktuální a jeho řešení bude obsahem nejbližšího dalšího standardu ISO týkajícího se VRML. Standard s názvem *External Authoring Interface* bude tvořit druhý díl normy VRML 97 a měl by být publikován v roce 1999.

Další aktivity jsou věnovány spolupráci a konverzi dat mezi VRML a programátorskou knihovnou Java 3D.

Jiné doplňky

Z různých aplikačních oblastí přicházejí rozličné požadavky na to, jak by měl být jazyk VRML rozšířen o další speciální prvky. Čím bohatší však bude jazyk VRML, tím složitěji se bude zpracovávat. Proto jsou uvedené snahy brzděny a nové prvky budou zaváděny teprve tehdy, pokud získají podporu u dostatečně velké skupiny uživatelů. Z možných směrů jmenujme například aktivity věnované modelování lidských bytostí, jednotnému ovládání všech prohlížečů, zavedení netradičních vstupních zařízení, zavedení uzlů pro popis geografických a geologických dat.

8.2 Jiné přístupy k popisu virtuální reality

Jazyk VRML není jediným prostředkem pro popis umělých objektů a světů. Mnohé z velkých, profesionálních systémů pro virtuální realitu používají své vlastní formáty dat, ať již jsou odvozeny z VRML nebo jsou zcela odlišné. Oproti těmto formátům má VRML velkou výhodu v tom, že je univerzální a zaměřený na Internet. Nevýhodou je pak to, že nezahrnuje všechny speciální rysy velkých systémů. V současné době je většina systémů pro virtuální realitu schopna převádět svá data do formátu VRML (byť obecně s omezením některých vlastností) a zpřístupnit tak složité umělé světy laickým návštěvníkům připojeným k Internetu.

V rámci Internetu se můžeme setkat se dvěma dalšími způsoby popisu virtuální reality. Prvním z nich je formát *Super VR*, druhým *QuickTime VR*. O obou se krátce zmíníme.

Formát *Super VR* (zkráceně *SVR*) vytvořila britská firma Superscape, průkopník virtuální reality na osobních počítačích. Formát je v principu podobný VRML, obsahuje však více možností pro animace a interakce. Nevýhodou je to, že prohlížeče pro formát SVR jsou určeny jen pro osobní počítače s operačním systémem MS Windows. Přestože se prohlížeče vyznačují vysokou rychlostí, z dlouhodobého hlediska se SVR patrně neprosadí. Svědčí o tom i skutečnost, že firma Superscape již vyvinula vlastní prohlížeč světů zapsaných ve VRML.

Mnohem zajímavější přístup k popisu virtuální reality představuje formát *QuickTime VR*. Ten vychází z formátu Apple QuickTime pro video sekvence a jeho domácím prostředím jsou počítače Apple Macintosh. Je však dostupný i na dalších počítačích a operačních systémech a tvoří sympatický doplněk jazyka VRML. Filosofie formátu QuickTime VR vychází z představy, že virtuální prostředí si uživatel chce především prohlížet, a to z různých stran a z různých vzdáleností. Zaznamenáme-li možné panoramatické pohledy na virtuální svět do jednoho souboru, získáme zvláštní video záznam, který lze uživateli přehrávat dopředu a dozadu, případně obraz zvětšovat a zmenšovat. Směr přehrávání odpovídá otáčení avatarovy hlavy doprava a doleva, změna velikosti obrazu koresponduje s tím, jak avatar postupuje dopředu či dozadu.

Virtuální svět je ve formátu QuickTime VR představován soustavou válcových ploch, na nichž jsou naneseny panoramatické záběry. Stojí-li avatar uvnitř virtuální místnosti, stojí ve skutečnosti v pomyslném válci. Zde je vidět jistá podoba s uzlem `Background` jazyka VRML. Virtuální tělesa, která si avatar prohlíží, jsou opět představována sérií panoramatických pohledů, tentokrát promítaných zvenčí na pomyslný válce nebo pomyslnou kouli obklopující těleso.

Formát QuickTime VR má menší možnosti interakcí a dynamických akcí. Uživatel si může pouze ze všech stran prohlížet umělá tělesa nebo dotykem vyvolat vstup do další virtuální oblasti. I přes tuto nevýhodu je formát poměrně oblíben, protože dokáže zprostředkovat vizuálně velmi pěkné prohlídky, a to nejen umělých, ale i reálných světů. S pomocí speciálního programového vybavení a dobře nastavenou kamerou lze připravit procházku jakýmkoliv skutečným prostředím a umožnit tak například vzdálenou návštěvu Technického muzea, Paříže, New Yorku nebo Mayských pyramid.

Poměrně velká potřeba obrazových dat je vyvážena extrémně rychlým a plynulým prohlížením, o jakém si většina prohlížečů VRML může zatím nechat jen zdát. Formát QuickTime VR bude patrně existovat souběžně s formátem VRML, neboť pro určité aplikace se jeví jako velmi výhodný.

8.3 Systémy pro virtuální realitu

V předchozí kapitole jsme zmínili existenci velkých systémů s vlastními formáty dat a vlastním řešením interaktivních akcí. Rozsah této knihy nedovoluje, abychom se o těchto systémech podrobněji rozepsali. Uvedeme proto jen jejich základní rysy, aby čtenář získal představu o tom, co vše může virtuální realita zahrnovat.

Pro profesionální systémy virtuální reality je typické používání speciálních periferních zařízení. Velký důraz je kladen na *optická zařízení*, neboť pocit, že se člověk nachází v trojrozměrném prostoru, je nejvíce ovlivněn zrakem. Proto je potřeba vytvářet samostatný obraz pro každé oko uživatele. Škála takových zobrazovacích zařízení je velmi široká. Nejdražší systémy obsahují helmy (*HMD*, angl. *Head Mounted Display*), v nichž jsou nejen dvě malé obrazovky pro obě oči, ale i reproduktory pro generování prostorového zvuku. Levnější přístup je založena na brýlích, jejichž zorníky se střídavě zatmívají. V každém okamžiku vidí obrazovku jen jediné oko. Jakmile je na ní vykreslen obraz pro dané oko, synchronizační impuls změní viditelnost zorníků a na obrazovce dojde k vykreslení obrazu pro druhé oko, který je mírně prostorově posunutý. Složením obou obrazů v mozku vzniká přesvědčivý prostorový vjem. Při použití tohoto poměrně jednoduchého zařízení však musí návštěvník sedět v určité vzdálenosti před obrazovkou a příliš se nehýbat.

Aby se návštěvník ve virtuálním světě cítil přirozeně, je třeba mu umožnit běžný pohyb. K ovládání pohybu avatara se proto nepoužívá myš, ale odvozuje se ze skutečných pohybů návštěvníka. Snímače polohy, které detekují směr návštěvníkova pohledu, sklon hlavy a jeho přesun z místa na místo, využívají různých fyzikálních principů. Většinou jsou tvořeny několika sondami, které se umístí na tělo uživatele. Centrální systém pak vyhodnocuje jeho polohu v prostoru. Existují i systémy, které snímají pohyb zorniček a přesně určují směr pohledu.

Další kvalitu získávají virtuální systémy při použití hmatových periferních zařízení. Ta mají za úkol zjišťovat, zda uživatel virtuální předmět drží, tlačí na něj, přejíždí rukou po jeho povrchu, deformuje jej. Zařízení jsou tvořena elektromechanickou či pneumatickou soustavou, do které se vkládá ruka. Sada detektorů reaguje na aktivitu ruky a naopak sada pohyblivých prvků generuje hmatové podněty. Tím uživatel získává velmi přesvědčivé informace o tvaru, povrchu a vlastnostech zkoumaného virtuálního předmětu.

Pokud se bude čtenář zabývat virtuální realitou ve větším rozsahu, setká se s různými hledisky dělení systémů virtuální reality. V současné době můžeme rozlišit zhruba čtyři základní typy aplikací:

Virtuální realita (VR)

Pod pojmem VR bez bližšího označení můžeme nalézt celou řadu aplikací od počítačových her po simulace pohybu ve vesmíru či životu nebezpečných oblastech. K iluzi práce nebo pohybu ve virtuálním světě poslouží obyčejná obrazovka, představující okno do trojrozměrného prostoru. Stereofonní reproduktory dodají zdání prostorového zvuku, myš nahradí složité ukazovací a uchopovací zařízení.

Do této obecné kategorie patří virtuální světy VRML. Zařazujeme sem však i systémy obsahující různá speciální zařízení, často určená pro výzkumné a experimentální účely, pokud nezapadají do následujících skupin.

Pohlcující virtuální realita (angl. Immersive VR)

Cílem systémů je v co největší míře oprostit uživatele od vjemů venkovního, skutečného světa a dodat mu zdání, že je zcela ponořen do světa umělého. Mezi typická periferní zařízení patří helma se stereoskopickými brýlemi a sluchátky, snímače detekující prostorovou polohu uživatele, datová rukavice nahrazující jednodušší vstupní zařízení. Často je uživatel umístěn v simulátoru - kabině, která se naklání a vyvolává pocit pádu či odstředivé síly.

Pohlcující virtuální realita nachází uplatnění nejen v herních centrech a rozličných trenažérech, ale i jako prostředek terapeutický. Uvádí se například úspěšné léčení panického strachu z výšek.

Rozšiřující virtuální realita (angl. Augmented VR)

Informace ze skutečného, okolního světa jsou kombinovány s doplněnými prvky virtuální reality. Příkladem je použití ve vojenských letadlech a vozidlech, kde je venkovní obraz snímaný kamerou přenášen na obrazovku a okamžitě doplněn výraznými symboly pro nepřátelské a spřátelené objekty. Někdy je míchání reálného a umělého světa realizováno na obyčejné obrazovce, jindy se používají speciální průhledné brýle, na něž počítač dokresluje další obrazy a symboly.

Někteří odborníci se domnívají, že právě rozšiřující virtuální realitě patří budoucnost, zejména v případě, kdy systém nevyžaduje takové periferie, které by člověka omezovaly v normální práci. Jako příklad se uvádí kombinace běžné kanceláře doplněné soustavou projektorů umístěných na stropě. Obrazy virtuálních předmětů se mohou promítat nejen na stěnu, ale například i na rýsovací prkno či jen papír držený ve vzduchu.

Distribuovaná virtuální realita (angl. Distributed VR)

S rozvojem rychlých počítačových sítí se virtuální realita stává prostředkem pro komunikaci mezi lidmi. Je možné hrát hry ve virtuálním světě se spoluhráči sedícími v danou chvíli na opačných koncích planety, stejně tak je možno posuzovat v distribuovaném lékařském týmu nádor či nemocný orgán reprezentovaný virtuálním modelem získaným z naměřených dat pacienta.

Z uvedeného přehledu je zřejmé, že jazyk VRML tvoří jen malou, byť důležitou součást toho, co nazýváme virtuální realitou. Další verze a rozšíření jazyka budou patrně postupně pronikat do všech výše uvedených oblastí a nezbývá než doufat, že napomohou žádoucímu sjednocení dosud mnohdy roztříštěných systémů.

9 Možná nudné, ale dozajista nutné

V předchozích kapitolách jsme čtenáři v některých případech vědomě předkládali pouze některé z parametrů VRML uzlů, protože jejich úplný seznam by ho spíše mátl a odrazoval od dalšího čtení. Ten, kdo chce plně využít všech možností jazyka VRML, se však neobejde bez kompletní specifikace všech parametrů. Tato kapitola proto obsahuje abecedně řazený přehled všech VRML uzlů, jejich parametrů včetně iniciálních hodnot a stručný výklad významu těchto parametrů. Úplný a rozsáhlý popis pak čtenář nalezne v oficiální specifikaci normy.

Nejprve znovu shrňme, do jakých logických skupin můžeme uzly rozdělit:

- Informační - NavigationInfo, Viewpoint, WorldInfo
- Geometrie (tvar)
  - Tělesa - Box, Cone, Cylinder, Sphere
  - Plochy - ElevationGrid, Extrusion, IndexedFaceSet
  - Ostatní - IndexedLineSet, PointSet, Text
  - Pomocné - Shape, FontStyle
  - Datové - Coordinate, Normal, TextureCoordinate
- Vzhled povrchu - Appearance, Color, ImageTexture, Material, MovieTexture, PixelTexture, TextureTransform
- Prostředí - AudioClip, Background, DirectionalLight, Fog, PointLight, Sound, SpotLight
- Skupinové - Anchor, Billboard, Group, Inline, LOD, Switch, Transform
- Speciální - Script, TimeSensor
- Manipulátory - CylinderSensor, PlaneSensor, SphereSensor
- Interpolátory - ColorInterpolator, CoordinateInterpolator, NormalInterpolator, OrientationInterpolator, PositionInterpolator, ScalarInterpolator
- Detektory - Collision, ProximitySensor, TouchSensor, VisibilitySensor

Uzly v rámečku patří mezi vázané, tedy ty, z nichž v každém okamžiku je platný (aktivní) maximálně jeden toho jména.

Kromě dělení uzlů do logických skupin podle funkce lze uzly členit i podle toho, v jakých částech stromové struktury se vyskytují. *Skupinové uzly* vystupují v roli rodičů dalších uzlů. Mívají několik potomků a samozřejmě také bývají kořeny stromu. V následujícím přehledu je u každého ze skupinových uzlů uvedeno v závorce jméno parametru, do kterého se zapisují potomci.

Skupinové uzly

Anchor	(children)	Inline	(url)
Billboard	(children)	LOD	(level)
Collision	(children, proxy)	Switch	(choice)
Group	(children)	Transform	(children)

Protikladem ke skupinový uzlům jsou uzly, které mají přesně stanovené místo ve stromové struktuře. Jsou vždy umístěny do konkrétního parametru rodičovského uzlu, jak také ukazuje obrázek 9-1. Nemohou nikdy tvořit kořen stromu, protože musejí mít rodiče.

Pevně umístěné uzly

Appearance	Cylinder	IndexedLineSet	Sphere
AudioClip	ElevationGrid	Material	Text
Box	Extrusion	MovieTexture	TextureCoordinate
Color	FontStyle	Normal	TextureTransform
Cone	ImageTexture	PixelTexture	
Coordinate	IndexedFaceSet	PointSet	

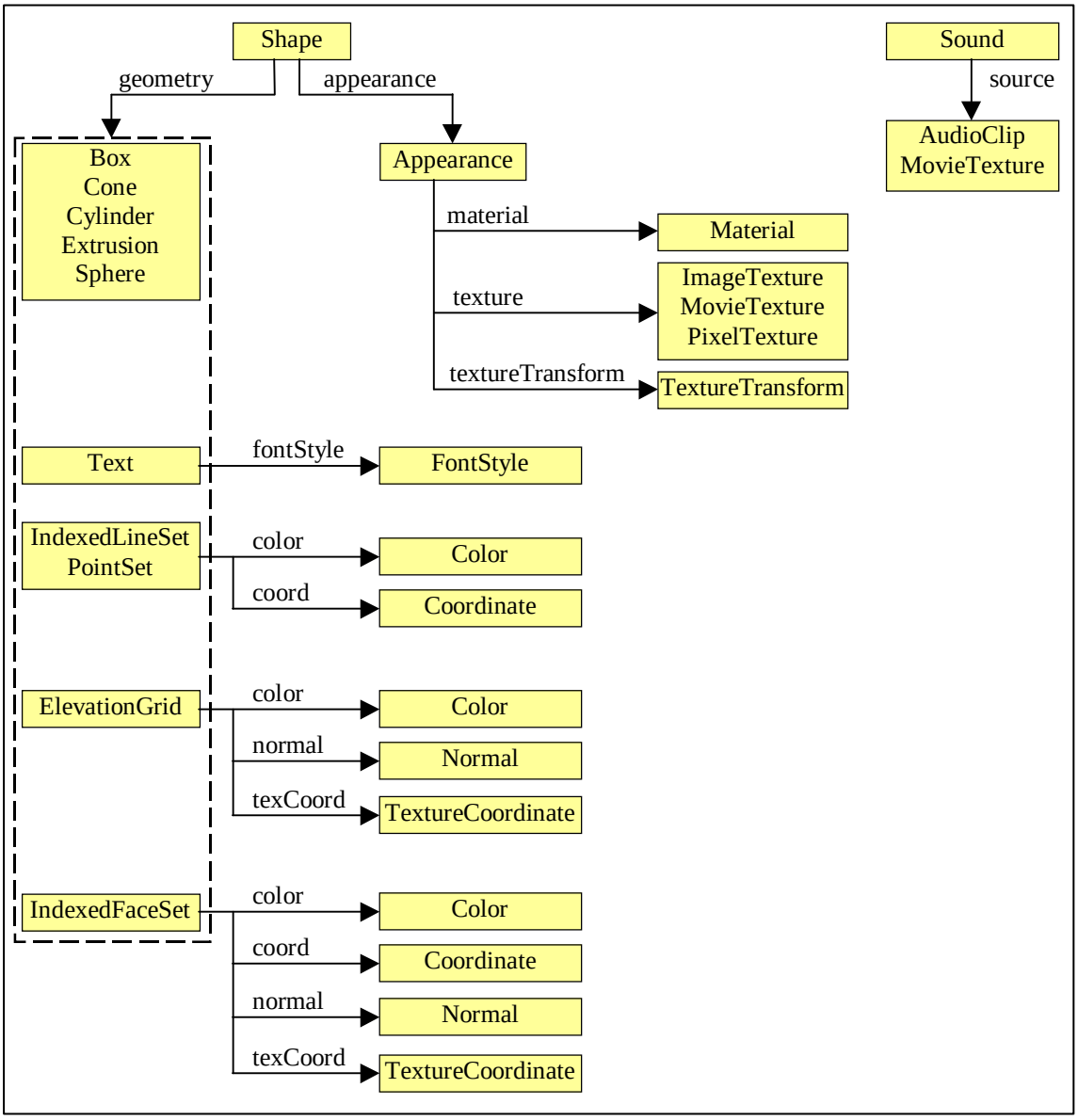
Zbylé uzly patří do kategorie takových uzlů, které se objevují v roli potomků skupinových uzlů. Je zřejmé, že také každý skupinový uzel může být potomkem dalšího skupinového uzlu. Správně by tedy do následujícího přehledu patřily i skupinové uzly.

Současně je pravda, že některé uzly mohou být zapsány do souboru i bez rodiče, samostatně. Jde především o vázané uzly (např. Background), informační uzly (např. NavigationInfo) a některé dynamické uzly (např. TimeSensor). Vzhledem k tomu, že v praxi se takové uzly často zapisují jako potomci uzlu **`Group`**, uvádíme je i do tohoto přehledu.

Potomci skupinových uzlů

Background	NavigationInfo	ProximitySensor	SphereSensor
ColorInterpolator	NormalInterpolator	ScalarInterpolator	TimeSensor
CoordinateInterpolator	OrientationInterpolator	Script	TouchSensor
CylinderSensor	PlaneSensor	Shape	Viewpoint
DirectionalLight	PointLight	Sound	VisibilitySensor
Fog	PositionInterpolator	SpotLight	WorldInfo

Nově definované uzly popsané příkazem **`PROTO`** spadají do výše uvedených kategorií podle toho, jaký je konkrétní obsah nového prototypu, tj. jaký je druh uzlu uvedeného v těle prototypu na prvním místě.



Obrázek 9-1: Vztahy mezi uzly a jejich parametry

Obrázek 9-1 schematicky ukazuje, jaké pevné rodičovské vztahy mohou existovat mezi uzly. Vzhledem k vysokému počtu uzlů není tento obrázek použitelný pro rychlé vyhledávání vztahů, proto je informace o možných rodičích a potomcích navíc uvedena zvlášť i u každého uzlu v dalším textu.

Na každé z následujících stránek nalezneme vždy jeden uzel se všemi jeho parametry a příslušným popisem. Nejprve jsou uvedeny univerzální parametry **`exposedField`** a statické parametry **`field`**. Za nimi následují parametry pro přijímání a odesílání událostí – **`eventIn`** a **`eventOut`**.

Vedle číselných hodnot se objevují v parametrech seznamy, uzly, textové řetězce. Pro ně je použito následující značení:

`""`	prázdný textový řetězec
`NULL`	žádný uzel
`[]`	prázdný seznam

Anchor (Teleportace, odkaz)

Sdružení uzlů do skupiny, po jejíž aktivaci je avatar přenesen (teleportován) do dalšího světa, případně do jiné části právě prohlíženého světa. Variantou je vyvolání nového okna prohlížeče, typicky s WWW stránkou.

typ parametru	typ dat	název	iniciální hodnota	význam
exposedField	MFString	`url`	`[]`	seznam adres jiných světů, jmen stanovišť a WWW stránek
	MFString	`parameter`	`[]`	seznam doplňujících parametrů předávaných prohlížeči po aktivaci
	SFString	`description`	`""`	informační text
	MFNode	`children`	`[]`	seznam potomků

field	SFVec3f	`bboxSize`	`-1 -1 -1`	kladné délky stran pomocné obálky ve tvaru kvádru; výjimka [-1, -1, -1] indikuje dosud nespecifikovanou velikost kvádru
	SFVec3f	`bboxCenter`	`0 0 0`	souřadnice středu pomocné obálky ve tvaru kvádru

eventIn	MFNode	`addChildren`	seznam připojovaných uzlů – potomků
	MFNode	`removeChildren`	seznam odstraňovaných uzlů – potomků

Rodič: žádný nebo libovolný skupinový uzel

Potomek: libovolný

Poznámky:

1. Jméno cílového stanoviště (uzlu **`Viewpoint`**) se v parametru **`url`** zapisuje za adresu odděleno znakem **`'#'`**. V cílovém souboru musí být jméno stanoviště zapsáno pomocí konstrukce **`DEF`**.
2. Zda a jak bude zobrazen informační text (**`description`**), je ponecháno na prohlížeči.

Appearance (Vzhled povrchu)

Přiřazení barevných vlastností a textury, případně jejich kombinace, geometrickému objektu, který je sourozencem tohoto uzlu.

typ parametru	typ dat	název	iniciální hodnota	význam
exposedField	SFNode	**material**	**NULL**	barevné vlastnosti povrchu
	SFNode	**texture**	**NULL**	textura na povrchu
	SFNode	**textureTransform**	**NULL**	umístění a natočení textury

Rodič: **Shape** (parametr **appearance**)

Potomek: v parametru **material** uzel **Material**

v parametru **texture** uzel **ImageTexture**, **MovieTexture** nebo **PixelTexture**

v parametru **textureTransform** uzel **TextureTransform**

AudioClip (Zvukový soubor)

Výběr zvukového souboru a určení způsobu jeho přehrávání.

typ parametru	typ dat	název	iniciální hodnota	význam
exposedField	MFString	**`url`**	**`[]`**	seznam adres s umístěním zvukového souboru
	SFString	**`description`**	**`""`**	informační text
	SFTime	**`startTime`**	**`0`**	čas zahájení přehrávání
	SFTime	**`stopTime`**	**`0`**	čas ukončení přehrávání
	SFFloat	**`pitch`**	**`1.0`**	kladná rychlost přehrávání (tempo)
	SFBool	**`loop`**	**`FALSE`**	povolení přehrávání ve smyčce

eventOut	SFTime	**`duration_changed`**	původní délka zvuku v sekundách; je vyslána po načtení souboru do paměti
	SFBool	**`isActive`**	bylo zahájeno nebo ukončeno přehrávání

Rodič: **`Sound`** (parametr **`source`**)

Potomek: žádný

Poznámky:

1. Zda a jak bude zobrazen informační text (**`description`**), je ponecháno na prohlížeči.
2. Povolený typ zvukového souboru je WAV (nekomprimovaný formát PCM), většina prohlížečů přehrává také soubory MIDI.

Background (Pozadí, panorama)

Definice pozadí reprezentovaného vnitřkem krychle s panoramatickým obrazem či vnitřkem koule s vodorovnými plynulými barevnými přechody, které obklopují ze všech stran virtuální svět.

typ parametru	typ dat	název	iniciální hodnota	význam
exposedField	MFString	**backUrl**	**[]**	seznam adres s umístěním obrázku pro zadní stěnu obklopující krychle
	MFString	**bottomUrl**	**[]**	seznam adres s umístěním obrázku pro dolní stěnu obklopující krychle
	MFString	**frontUrl**	**[]**	seznam adres s umístěním obrázku pro čelní stěnu obklopující krychle
	MFString	**leftUrl**	**[]**	seznam adres s umístěním obrázku pro levou stěnu obklopující krychle
	MFString	**rightUrl**	**[]**	seznam adres s umístěním obrázku pro pravou stěnu obklopující krychle
	MFString	**topUrl**	**[]**	seznam adres s umístěním obrázku pro horní stěnu obklopující krychle
	MFColor	**skyColor**	**0 0 0**	seznam barev pro postupné přechody na sférické obloze
	MFFloat	**skyAngle**	**[]**	rostoucí posloupnost nezáporných úhlů, pro které jsou dány barvy oblohy
	MFColor	**groundColor**	**[]**	seznam barev pro postupné přechody na sférické zemi (podlaze)
	MFFloat	**groundAngle**	**[]**	rostoucí posloupnost nezáporných úhlů, pro které jsou dány barvy země

eventIn	SFBool	**set_bind**	aktivování pozadí

eventOut	SFBool	**isBound**	pozadí se stalo aktivním nebo naopak bylo nahrazeno jiným

Rodič: žádný nebo skupinový uzel (parametr **children**)

Potomek: žádný

Zvláštnost: V každém okamžiku je z více uzlů tohoto typu aktivní právě jeden

Poznámky:

1. Úhly pro barevné přechody oblohy (**skyAngle**) jsou v rozsahu od 0 do π (shora dolů), aby bylo možno modelovat zcela volný prostor. Úhly pro barevné přechody země (**groundAngle**) jsou pouze do pravého úhlu (od podhlavníku k horizontu).
2. Počet barev oblohy a země (**skyColor**, **groundColor**) musí být o jedničku vyšší než počet odpovídajících úhlů, neboť první barva v seznamu určuje barvu nadhlavníku, resp. podhlavníku.
3. Na vzhled pozadí nemá vliv ani použití mlhy ani zdroje světla.
4. Jako zdroje obrázků lze použít stejné formáty souborů jako v uzlu **ImageTexture**.

Billboard

Sdružení uzlů do skupiny, která je automaticky otáčena s ohledem na aktuální polohu avatara.

typ parametru	typ dat	název	iniciální hodnota	význam
exposedField	MFNode	**children**	**[]**	seznam potomků
	SFVec3f	**axisOfRotation**	**0 1 0**	osa otáčení

field	SFVec3f	**bboxSize**	**-1 -1 -1**	kladné délky stran pomocné obálky ve tvaru kvádru; výjimka [-1, -1, -1] indikuje dosud nespecifikovanou velikost kvádru
	SFVec3f	**bboxCenter**	**0 0 0**	souřadnice středu pomocné obálky ve tvaru kvádru

eventIn	MFNode	**addChildren**	seznam připojovaných uzlů – potomků
	MFNode	**removeChildren**	seznam odstraňovaných uzlů – potomků

Rodič: žádný nebo libovolný skupinový uzel

Potomek: libovolný

Poznámky:

1. Speciální hodnota **0 0 0** pro parametr **axisOfRotation** znamená otáčení nikoliv kolem osy, ale kolem středu.
2. Geometrické parametry jsou ovlivňovány transformacemi případného rodičovského uzlu **Transform**.

Box (Kvádr)

Šest jednostranných ploch pokrývajících osově orientovaný kvádr s těžištěm v počátku systému souřadnic.

typ parametru	typ dat	název	iniciální hodnota	význam
field	SFVec3f	**`size`**	**`2 2 2`**	nezáporné rozměry kvádru

Rodič: **`Shape`** (parametr **`geometry`**)

Potomek: žádný

Collision (Detekce nárazu)

Sdružení uzlů do skupiny a povolení či zakázání výpočtů kolize avatara s objekty této skupiny. Složitější skupina může být pro výpočty kolizí nahrazena nezobrazovaným náhradním tělesem či skupinou těles.

typ parametru	typ dat	název	iniciální hodnota	význam
exposedField	MFNode	**children**	**[]**	seznam potomků
	SFBool	**collide**	**TRUE**	povolení detekování kolizí s avatarem
field	SFVec3f	**bboxSize**	**-1 -1 -1**	kladné délky stran pomocné obálky ve tvaru kvádru; výjimka [-1, -1, -1] indikuje dosud nespecifikovanou velikost kvádru
	SFVec3f	**bboxCenter**	**0 0 0**	souřadnice středu pomocné obálky ve tvaru kvádru
	SFNode	**proxy**	**NULL**	uzel nebo strom s náhradní geometrií, která se použije pro detekce kolize
eventIn	MFNode	**addChildren**		seznam připojovaných uzlů - potomků
	MFNode	**removeChildren**		seznam odstraňovaných uzlů - potomků
eventOut	SFTime	**collideTime**		čas nárazu avatara

Rodič: žádný nebo libovolný skupinový uzel

Potomek: libovolné stromy s geometrií, ať již v parametru **children** či **proxy**

Poznámky:

1. Je-li definován náhradní objekt pro výpočet kolizí (**proxy**), žádný z potomků (**children**) není na kolizi testován. Náhradní objekt není pochopitelně nikdy zobrazován.
2. Obálka ve tvaru kvádru (*bbox*) neslouží jako náhradní reprezentace, pouze jako doplňující informace.

Color (Barvy)

Definice difúzních barev používaných k obarvení prvků v obecných geometrických objektech.

typ parametru	typ dat	název	iniciální hodnota	význam
exposedField	MFColor	**`color`**	**`[]`**	seznam barev v modelu RGB

Rodič: **`ElevationGrid`**, **`IndexedFaceSet`**, **`IndexedLineSet`** a **`PointSet`** (parametr **`color`**)

Potomek: žádný

ColorInterpolator (Interpolace barvy)

Uzel, který generuje barvu postupnou interpolací mezi dvěma po sobě následujícími klíčovými hodnotami barev. Výběr hodnot pro interpolaci je určen vstupní řídicí hodnotou.

typ parametru	typ dat	název	iniciální hodnota	význam
exposedField	MFFloat	**key**	**[]**	neklesající posloupnost řídicích klíčů
	MFColor	**keyValue**	**[]**	seznam barev RGB – hodnot pro odpovídající řídicí klíče

eventIn	SFFloat	**set_fraction**	aktuální řídicí hodnota

eventOut	SFColor	**value_changed**	interpolovaná barva

Rodič: žádný nebo libovolný skupinový uzel

Potomek: žádný

Poznámky:

1. Uzel generuje pouze jedinou barvu, nikoliv seznam barev.
2. Počet hodnot v seznamech **key** a **keyValue** by měl být shodný.

Cone (Kužel)

Síť jednostranných ploch pokrývajících kužel s osou rovnoběžnou s osou *y*, střed osy kužele leží v počátku systému souřadnic.

typ parametru	typ dat	název	iniciální hodnota	význam
field	SFFloat	**bottomRadius**	**1**	nezáporný poloměr podstavy kuželu
	SFFloat	**height**	**2**	nezáporná výška
	SFBool	**side**	**TRUE**	povolení vykreslování pláště
	SFBool	**bottom**	**TRUE**	povolení vykreslování podstavy

Rodič: **Shape** (parametr **geometry**)

Potomek: žádný

Coordinate (Souřadnice prostorových bodů)

Definice souřadnic bodů v prostoru, používaných jako vrcholy v obecných geometrických objektech.

typ parametru	typ dat	název	iniciální hodnota	význam
exposedField	MFVec3f	**`point`**	**`[]`**	seznam bodů

Rodič: **`IndexedFaceSet`**, **`IndexedLineSet`** a **`PointSet`** (parametr **`coord`**)

Potomek: žádný

CoordinateInterpolator (Interpolace souřadnic)

Uzel, který generuje seznam souřadnic postupnou interpolací mezi dvěma po sobě následujícími klíčovými seznamy souřadnic. Výběr hodnot pro interpolaci je určen vstupní řídicí hodnotou.

typ parametru	typ dat	název	iniciální hodnota	význam
exposedField	MFFloat	**key**	**[]**	neklesající posloupnost řídicích klíčů
	MFVec3f	**keyValue**	**[]**	pole seznamů souřadnic – hodnot pro odpovídající řídicí klíče

eventIn	SFFloat	**set_fraction**	aktuální řídicí hodnota

eventOut	MFVec3f	**value_changed**	seznam interpolovaných souřadnic

Rodič: žádný nebo libovolný skupinový uzel

Potomek: žádný

Poznámka:

Protože uzel generuje najednou celý seznam souřadnic, počet jednotlivých souřadnic v seznamu **keyValue** by měl být N x M, kde M je požadovaná délka výstupního seznamu souřadnic a N počet hodnot v parametru **key**.

Cylinder (Válec)

Síť jednostranných ploch pokrývajících válec s těžištěm v počátku systému souřadnic a osou rovnoběžnou s osou *y*.

typ parametru	typ dat	název	iniciální hodnota	význam
field	SFFloat	**height**	**2**	nezáporná výška válce
	SFFloat	**radius**	**1**	nezáporný poloměr
	SFBool	**side**	**TRUE**	povolení vykreslování pláště
	SFBool	**bottom**	**TRUE**	povolení vykreslování dolní podstavy
	SFBool	**top**	**TRUE**	povolení vykreslování horní podstavy

Rodič: **Shape** (parametr **geometry**)

Potomek: žádný

CylinderSensor (Válcový manipulátor)

Převod údajů ze vstupního ukazovacího zařízení (myši) na úhel otočení daného pohybem po povrchu pomyslného válce, jehož osa je rovnoběžná s osou *y*. Umístění a rozměry pomyslného válce jsou dány geometrií sourozeneckých uzlů.

typ parametru	typ dat	název	iniciální hodnota	význam
exposedField	SFBool	**enabled**	**TRUE**	povolení práce manipulátoru
	SFFloat	**offset**	**0**	základní úhel, který je vždy přičítán k nově vyhodnocenému úhlu
	SFBool	**autoOffset**	**TRUE**	povolení automatické aktualizace **offset** po skončení činnosti manipulátoru
	SFFloat	**minAngle**	**0**	dolní mez vyhodnoceného úhlu otočení; je z intervalu [-2π, 2π]
	SFFloat	**maxAngle**	**-1**	horní mez vyhodnoceného úhlu otočení; je z intervalu [-2π, 2π]
	SFFloat	**diskAngle**	**0.262**	kladný úhel menší než π/2, rozlišující, zda z pomyslného válce se použije podstava nebo plášť.

eventOut	SFBool	**isActive**	zahájení a ukončení činnosti manipulátoru
	SFVec3f	**trackPoint_changed**	měnící se poloha bodu na pomyslném válci, na který ukazuje kurzor při práci s manipulátorem
	SFRotation	**rotation_changed**	měnící se hodnota úhlu odvozená z pohybu vstupního zařízení kolem osy válce

Rodič: skupinový uzel, typicky **Transform** nebo **Group** (parametr **children**)

Potomek: žádný

Zvláštnost: objekty, které jsou citlivé na vstupní ukazovací zařízení, jsou sourozenci tohoto uzlu

Poznámky:

1. Manipulátor sám o sobě nemění transformace žádného uzlu, pouze pro ně vypočítává vhodné parametry.
2. Je-li nastaven parametr **autoOffset** na **TRUE**, je po skončení činnosti manipulátoru aktualizována hodnota parametru **offset** a vyslána událost **offset_changed**.
3. Parametr **diskAngle** určuje, zda pohyb kurzoru bude manipulátorem chápán jako pohyb po plášti pomyslného válce (např. přesun kurzoru zleva doprava, pokud je válec „vidět“ v iniciální poloze) či jako pohyb po kruhové podstavě pomyslného válce (např. při pohledu na pomyslný válec shora). Vektor určený aktuální polohou avatara a bodem na povrchu pomyslného válce svírá s osou pomyslného válce úhel, jehož hodnota je porovnávána s velikostí parametru **diskAngle**. Je-li svíraný úhel menší, pracuje se s podstavou válce a ze strany uživatele je očekáván kruhový pohyb kurzoru. V opačném případě se pracuje s pláštěm a uživatel může generovat otáčení pouze translačním pohybem kurzoru.
4. Je-li hodnota **maxAngle** menší než **minAngle**, vypočítávané úhly otočení nejsou omezeny. V opačném případě je pohyb kurzoru, který by generoval otočení mimo meze, ignorován a je vysílán odpovídající mezní úhel.

DirectionalLight (Směrový zdroj světla)

Světelná charakteristika zdroje, vysílajícího svazky rovnoběžných paprsků.

typ parametru	typ dat	název	iniciální hodnota	význam
exposedField	SFVec3f	**direction**	**0 0 -1**	směr světelných paprsků
	SFColor	**color**	**1 1 1**	barva paprsků
	SFFloat	**intensity**	**1**	iniciální intenzita paprsků v rozsahu [0, 1]
	SFFloat	**ambientIntensity**	**0**	příspěvek zdroje k nepřímému osvětlení virtuálního světa v rozsahu [0, 1]
	SFBool	**on**	**TRUE**	zapnutí či vypnutí světelného zdroje

Rodič: žádný, nejlépe však skupinový uzel (parametr **children**)

Potomek: žádný

Poznámky:

1. Tento světelný zdroj osvětluje pouze objekty se stejným rodičovským uzlem, tedy sourozenecké stromy.
2. Směr světelných paprsků (**direction**) je ovlivňován případnou transformací rodičovského uzlu **Transform**.

ElevationGrid (Výšková mapa)

Komplexní definice sítě ploch pokrývající terén.

typ parametru	typ dat	název	iniciální hodnota	význam
exposedField	SFNode	`color`	`NULL`	seznam barev v uzlu `Color`
	SFNode	`normal`	`NULL`	seznam normál v uzlu `Normal`
	SFNode	`texCoord`	`NULL`	seznam souřadnic textury v uzlu `TextureCoordinate`

field	SFInt32	`xDimension`	`0`	počet vzorků (vrcholů sítě) v ose *x*
	SFInt32	`zDimension`	`0`	počet vzorků (vrcholů sítě) v ose *z*
	SFFloat	`xSpacing`	`1.0`	kladná vzdálenost mezi vzorky v ose *x*
	SFFloat	`zSpacing`	`1.0`	kladná vzdálenost mezi vzorky v ose *z*
	MFFloat	`height`	`[]`	pole výšek všech vrcholů sítě
	SFBool	`colorPerVertex`	`TRUE`	barvy v parametru `color` se vztahují na vrcholy, jinak na celé plochy sítě
	SFBool	`normalPerVertex`	`TRUE`	normály v parametru `normal` se vztahují na vrcholy, jinak na plochy
	SFBool	`ccw`	`TRUE`	přivrácená strana mapy je vidět při pohledu shora (proti ose *y*)
	SFBool	`solid`	`TRUE`	mapa je jednostranná
	SFFloat	`creaseAngle`	`0`	nezáporný úhel, až do kterého jsou dvě sousední plochy považovány za oblé

eventIn	MFFloat	`set_height`	změna parametru `height`

Rodič: `Shape` (parametr `geometry`)

Potomek: uzly `Color`, `Normal` a `TextureCoordinate` v příslušných parametrech

Extrusion (Opláštění)

Plochy pokrývající těleso vniklé tažením a otáčením rovinného obrysu po trajektorii určené lomenou čárou.

typ parametru	typ dat	název	iniciální hodnota	význam
field	MFVec2f	**crossSection**	**[1 1, 1 -1, -1 -1, -1 1, 1 1]**	posloupnost bodů v rovině určující obrysovou křivku (obecně neuzavřenou)
	MFVec3f	**spine**	**[0 0 0, 0 1 0]**	posloupnost bodů v prostoru určující trajektorii, po které je tažen obrys
	MFVec2f	**scale**	**1 1**	seznam kladných měřítek (v rovině *xz*) pro každou polohu obrysu
	MFRotation	**orientation**	**0 0 1 0**	seznam otočení pro každou polohu obrysu
	SFBool	**beginCap**	**TRUE**	povolení vykreslování dolní podstavy
	SFBool	**endCap**	**TRUE**	povolení vykreslování horní podstavy
	SFBool	**convex**	**TRUE**	obrys je konvexní
	SFBool	**ccw**	**TRUE**	obrys je zadán proti směru hodinových ručiček
	SFBool	**solid**	**TRUE**	všechny plochy jsou jednostranné
	SFFloat	**creaseAngle**	**0**	nezáporný úhel, až do kterého jsou dvě sousední plochy považovány za oblé

eventIn	MFVec2f	**set_crossSection**	změna parametru **crossSection**
	MFVec3f	**set_spine**	změna parametru **spine**
	MFVec2f	**set_scale**	změna parametru **scale**
	MFRotation	**set_orientation**	změna parametru **orientation**

Rodič: **Shape** (parametr **geometry**)

Potomek: žádný

Poznámky:

1. U daného obrysu je nejprve upraveno měřítko (**scale**), poté je posunut do příslušného bodu trajektorie (**spine**) a na závěr otočen (**orientation**). Otočení je vztaženo k rovině, která dělí úhel dvou po sobě následujících bodů trajektorie.
2. Iniciální hodnoty uzlu definují kvádr s hranami délky 2 a výškou 1, s dolní podstavou v rovině *xz* (*y*=0).
3. Počet údajů v parametrech **scale** a **orientation** musí být roven počtu bodů trajektorie **spine**. Výjimečně lze uvést jen jedinou hodnotu, která pak platí i pro všechny ostatní.
4. Je-li obrys tvořen neuzavřenou křivkou, horní a dolní podstavy automaticky vzniknou spojením prvního a posledního bodu obrysu.

Fog (Mlha)

Definice barvy mlhy, která je míchána s barvou objektů podle rostoucí vzdálenosti od avatara.

typ parametru	typ dat	název	iniciální hodnota	význam
exposedField	SFColor	**color**	**1 1 1**	barva mlhy ve složkách RGB
	SFString	**fogType**	**"LINEAR"**	způsob houstnutí mlhy ("LINEAR" nebo "EXPONENTIAL")
	SFFloat	**visibilityRange**	**0**	nezáporná vzdálenost maximálního „barevného“ dohledu; hodnota 0 zcela odstraňuje mlhu

eventIn	SFBool	**set_bind**	aktivování mlhy

eventOut	SFBool	**isBound**	mlha se stala aktivní nebo naopak byla nahrazena jinou mlhou

Rodič: žádný nebo skupinový uzel (parametr **children**)

Potomek: žádný

Zvláštnost: V každém okamžiku je z více uzlů tohoto typu aktivní právě jeden

Poznámky:

1. Barevný dohled (**visibilityRange**) definuje mez, za kterou jsou vzdálenější objekty zcela překresleny barvou mlhy, čímž většinou zmizí (splynou). Je ovlivněn měřítkem případného rodičovského uzlu **Transform**.
2. Barva mlhy jí míchána pouze s objekty, které mají geometrii. Nemá žádný vliv na pozadí.

FontStyle (Styl písma)

Komplexní popis písma podle zadaného fontu, jeho velikosti, umístění a směru psaní.

typ parametru	typ dat	název	iniciální hodnota	význam
field	SFString	`language`	`""`	dvojznaková zkratka použité abecedy
	MFString	`family`	`"SERIF"`	seznam rodin písma
	SFString	`style`	`"PLAIN"`	styl pro danou rodinu písma
	SFFloat	`size`	`1.0`	kladná výška písma
	SFFloat	`spacing`	`1.0`	nezáporná hodnota, definující vzdálenost mezi řádky jako `spacing` x `size`
	SFBool	`horizontal`	`TRUE`	psaní ve vodorovném směru
	SFBool	`leftToRight`	`TRUE`	psaní na řádku zleva doprava
	SFBool	`topToBottom`	`TRUE`	psaní řádků shora dolů
	MFString	`justify`	`"BEGIN"`	hlavní a vedlejší způsob umístění textu

Rodič: `Text` (parametr `fontStyle`)

Potomek: žádný

Poznámky:

1. Jsou definovány tři základní rodiny písem použitelné v parametru `family` - **`"SERIF"`** pro patkové písmo (např. Times Roman), **`"SANS"`** pro bezpatkové (např. Arial, Helvetica) a **`"TYPEWRITER"`** pro písmo s pevnou šířkou znaků, jako je např. Courier.
2. Parametr `style` obsahuje jeden ze čtyř stylů písma - normální **`"PLAIN"`**, polotučné **`"BOLD"`**, kurzívu **`"ITALIC"`** a polotučnou kurzívu **`"BOLDITALIC"`**.
3. Do parametru `justify` se přiřazuje dvojice určující tzv. *hlavní* a *vedlejší* způsob umístění, přičemž některé hlavní způsoby není třeba doplňovat vedlejšími. Podle nastavení parametru `horizontal` je hlavní způsob vztažen k vodorovnému směru a vedlejší ke svislému, respektive naopak. Možné hodnoty pro způsoby umístění jsou **`"FIRST"`**, **`"BEGIN"`**, **`"MIDDLE"`** a **`"END"`**.
4. Na výšku písma (`size`) se vztahují transformace případného rodičovského uzlu.

Group (Skupina)

Sdružení více uzlů do jednoho stromu.

typ parametru	typ dat	název	iniciální hodnota	význam
exposedField	MFNode	**children**	**[]**	seznam potomků

field	SFVec3f	**bboxSize**	**-1 -1 -1**	kladné délky stran pomocné obálky ve tvaru kvádru; výjimka [-1, -1, -1] indikuje dosud nespecifikovanou velikost kvádru
	SFVec3f	**bboxCenter**	**0 0 0**	souřadnice středu pomocné obálky ve tvaru kvádru

eventIn	MFNode	**addChildren**	seznam připojovaných uzlů - potomků
	MFNode	**removeChildren**	seznam odstraňovaných uzlů - potomků

Rodič: žádný nebo skupinový uzel (parametr **children**, případně **choice**, **level** či **proxy**)

Potomek: libovolný

Poznámka:

Uzel **Group** je totožný s uzlem **Transform**, avšak neobsahuje žádné transformace. Je vhodný pro seskupování negeometrických uzlů, např. senzorů a interpolátorů.

ImageTexture (Textura určená obrázkem)

Určení obrázku a jeho případného opakovaného nanášení jako textury na povrchu.

typ parametru	typ dat	název	iniciální hodnota	význam
exposedField	MFString	`url`	`[]`	seznam adres s umístěním obrázku

field	SFBool	`repeatS`	`TRUE`	povolení opakování textury ve vodorovném směru
	SFBool	`repeatT`	`TRUE`	povolení opakování textury ve svislém směru

Rodič: `Appearance` (parametr `texture`)

Potomek: žádný

Poznámka:

Povolené formáty obrazových souborů jsou JPEG a PNG, podporován je také GIF a v některých prohlížečích vektorový formát CGM.

IndexedFaceSet (Množina ploch)

Komplexní definice geometrie a vzhledu skupiny ploch.

typ parametru	typ dat	název	iniciální hodnota	význam
exposedField	SFNode	**coord**	**NULL**	seznam vrcholů v uzlu **Coordinate**
	SFNode	**normal**	**NULL**	seznam normál v uzlu **Normal**
	SFNode	**color**	**NULL**	seznam barev v uzlu **Color**
	SFNode	**texCoord**	**NULL**	seznam souřadnic textury v uzlu **TextureCoordinate**

field	MFInt32	**coordIndex**	**[]**	posloupnosti indexů vrcholů jednotlivých ploch (zakončené číslem –1)
	MFInt32	**normalIndex**	**[]**	posloupnosti indexů normál pro jednotlivé vrcholy či jednotlivé plochy
	MFInt32	**colorIndex**	**[]**	posloupnosti indexů barev pro jednotlivé vrcholy či jednotlivé plochy
	MFInt32	**texCoordIndex**	**[]**	posloupnosti indexů souřadnic textury pro jednotlivé vrcholy
	SFBool	**colorPerVertex**	**TRUE**	barvy v parametru **colorIndex** se vztahují na vrcholy, jinak na plochy
	SFBool	**normalPerVertex**	**TRUE**	normály v parametru **normalIndex** se vztahují na vrcholy, jinak na plochy
	SFBool	**convex**	**TRUE**	všechny plochy jsou konvexní
	SFBool	**ccw**	**TRUE**	přivrácené strany všech ploch jsou zadávány proti směru hodinových ručiček
	SFBool	**solid**	**TRUE**	všechny plochy jsou jednostranné
	SFFloat	**creaseAngle**	**0**	nezáporný úhel, až do kterého jsou dvě sousední plochy považovány za oblé

eventIn	MFInt32	**set_coordIndex**	změna parametru **coordIndex**
	MFInt32	**set_normalIndex**	změna parametru **normalIndex**
	MFInt32	**set_colorIndex**	změna parametru **colorIndex**
	MFInt32	**set_texCoordIndex**	změna parametru **texCoordIndex**

Rodič: **Shape** (parametr **geometry**)

Potomek: uzly **Coordinate**, **Normal**, **Color** a **TextureCoordinate** v příslušných parametrech

IndexedLineSet (Množina čar)

Definice skupiny barevných lomených čar v prostoru.

typ parametru	typ dat	název	iniciální hodnota	význam
exposedField	SFNode	**coord**	**NULL**	seznam vrcholů v uzlu **Coordinate**
	SFNode	**color**	**NULL**	seznam barev v uzlu **Color**

field	MFInt32	**coordIndex**	**[]**	posloupnosti indexů vrcholů jednotlivých čar (zakončené číslem –1)
	MFInt32	**colorIndex**	**[]**	posloupnosti indexů barev pro jednotlivé vrcholy či celé čáry
	SFBool	**colorPerVertex**	**TRUE**	barvy v parametru **colorIndex** se vztahují na vrcholy, jinak na celé čáry

eventIn	MFInt32	**set_coordIndex**	změna parametru **coordIndex**
	MFInt32	**set_colorIndex**	změna parametru **colorIndex**

Rodič: **Shape** (parametr **geometry**)

Potomek: v parametru **coord** uzel **Coordinate**

v parametru **color** uzel **Color**

Poznámky:

1. Tloušťka čar není definována, typicky bývá rovna velikosti jednoho pixelu na obrazovce.
2. Čáry nepodléhají kolizím s avatarem, pokrytí texturou a osvětlení ze zdrojů světla.

Inline (Vložení)

Vložení dalšího světa nebo objektu z vnějšího souboru do aktuálního světa.

typ parametru	typ dat	název	iniciální hodnota	význam
exposedField	MFString	`url`	`[]`	seznam adres virtuálního světa či objektu

field	SFVec3f	`bboxSize`	`-1 -1 -1`	kladné délky stran pomocné obálky ve tvaru kvádru; výjimka [-1, -1, -1] indikuje dosud nespecifikovanou velikost kvádru
	SFVec3f	`bboxCenter`	`0 0 0`	souřadnice středu pomocné obálky ve tvaru kvádru obklopujícího vkládaný svět

Rodič: žádný nebo libovolný skupinový uzel

Potomek: žádný

Poznámky:

1. Z možných adres virtuálních světů nebo objektů (`url`) je vybrán pouze první dostupný soubor.
2. Vkládaný svět by neměl přesáhnout pomocnou obálku (*bbox*), je-li definována. Obálka neslouží jako prostředek pro zmenšení a posunutí vkládaného světa. Takové transformace je nutno zajistit v rodičovském uzlu `Transform`.
3. Vkládaný svět musí být platným, samostatným VRML souborem. Nelze tedy vkládat například pouze uzel definující barvu, zvuk apod.

LOD (Stupeň detailu)

Variantní reprezentace jednoho objektu v různých detailech (přesnosti).

typ parametru	typ dat	název	iniciální hodnota	význam
exposedField	MFNode	`level`	`[]`	seznam reprezentací objektu s postupně klesající přesností (množstvím detailů)

field	SFVec3f	`center`	`0 0 0`	bod, vůči kterému se měří vzdálenost objektu od avatara
	MFFloat	`range`	`[]`	rostoucí posloupnost kladných vzdáleností indikujících přepnutí reprezentace

Rodič: žádný nebo libovolný skupinový uzel

Potomek: libovolné podstromy, které by mohly být samostatně umístěny ve VRML souboru

Poznámky:

1. Počet reprezentací v seznamu `level` by měl být o jednu větší než počet vzdáleností v seznamu `range`, jinak je pro chybějící reprezentace použita ta poslední, nejjednodušší.
2. Prázdný seznam `range` indikuje, že prohlížeč bude automaticky vybírat vhodnou reprezentaci s ohledem na aktuální rychlost jejího zobrazování.

Material (Barevné vlastnosti)

Nastavení barevných charakteristik povrchu podle schopnosti odrazu barevných složek nepřímého a přímého světla. Nastavení průhlednosti nebo naopak vlastní zářivosti.

typ parametru	typ dat	název	iniciální hodnota	význam
exposedField	SFFloat	**ambientIntensity**	**0.2**	*světlost povrchu* získaná odrazem od okolního světla, tedy bez ohledu na úhel dopadu světla na povrch
	SFColor	**diffuseColor**	**0.8 0.8 0.8**	barva *povrchu* ovlivňovaná úhlem dopadu světla na povrch
	SFColor	**specularColor**	**0 0 0**	barva *světla* odraženého od přímých zdrojů světla
	SFFloat	**shininess**	**0.2**	rozmazané (0.0) nebo ostré (1.0) *odlesky* od přímých zdrojů světla
	SFColor	**emissiveColor**	**0 0 0**	*vyzařovaná* barva, luminiscence; není ovlivňována žádnými zdroji světla
	SFFloat	**transparency**	**0**	průhlednost

Rodič: **Appearance** (parametr **material**)

Potomek: žádný

Poznámky:

1. Hodnoty všech parametrů jsou v rozsahu [0, 1].
2. Současné nastavení všech parametrů nemá smysl. Většinou jsou některé z nich vynulovány.
3. Výsledná barva bodů na povrchu se vypočítává podle vzorce, který bere do úvahy všechny dostupné zdroje světla. Vzorec je odvozen z tzv. Phongova osvětlovacího modelu (viz [2, 8]).
4. Barva je ovlivňována čelním světlem avatara, světelnými zdroji a barvou mlhy.

MovieTexture (Pohyblivá textura)

Určení souboru s animací nebo video sekvencí a jeho případného opakovaného nanášení jako textury na povrch. Druhou možností je využití samotné zvukové stopy video záznamu jako zdroje zvuku.

typ parametru	typ dat	název	iniciální hodnota	význam
exposedField	MFString	**`url`**	**`[]`**	seznam adres s umístěním video souboru
	SFTime	**`startTime`**	**`0`**	čas zahájení přehrávání
	SFTime	**`stopTime`**	**`0`**	čas ukončení přehrávání
	SFFloat	**`speed`**	**`1.0`**	rychlost přehrávání; záporná hodnota znamená přehrávání pozpátku
	SFBool	**`loop`**	**`FALSE`**	povolení přehrávání ve smyčce
field	SFBool	**`repeatS`**	**`TRUE`**	povolení opakování textury ve vodorovném směru
	SFBool	**`repeatT`**	**`TRUE`**	povolení opakování textury ve svislém směru
eventOut	SFTime	**`duration_changed`**		původní délka sekvence v sekundách; je vyslána po načtení souboru do paměti
	SFBool	**`isActive`**		bylo zahájeno nebo ukončeno přehrávání sekvence

Rodič: **`Appearance`** (parametr **`texture`**) nebo **`Sound`** (parametr **`source`**)

Potomek: žádný

Poznámka:

Povolený formát video souboru je MPEG, některé prohlížeče podporují také animovaný GIF.

NavigationInfo (Ovládání avatara)

Geometrické, světelné a pohybové charakteristiky avatara, volba ovládacích prvků prohlížeče.

typ parametru	typ dat	název	iniciální hodnota	význam
exposedField	MFFloat	**avatarSize**	**[0.25, 1.6, 0.75]**	trojice kladných čísel, určujících rozměry avatara: šířku (resp. její polovinu), výšku očí a výšku kolen (překonatelné překážky)
	SFBool	**headlight**	**TRUE**	zapnutí svítilny na pomyslném čele avatara; svítilna vyzařuje plné bílé světlo, ale nepřispívá k ambientnímu osvětlení světa
	SFFloat	**speed**	**1.0**	rychlost pohybu avatara v m/s; nezáporné číslo
	MFString	**type**	**["WALK", "ANY"]**	seznam povolených způsobů přesunu avatara v prostoru; možnosti: "ANY", "WALK", "EXAMINE", "FLY", "NONE"
	SFFloat	**visibilityLimit**	**0.0**	dohled avatara; nezáporné číslo

eventIn	SFBool	**set_bind**	aktivování vlastností avatara a způsobu jeho ovládání

eventOut	SFBool	**isBound**	uzel se stal aktivním nebo naopak byl nahrazen jiným

Rodič: žádný

Potomek: žádný

Zvláštnost: v každém okamžiku je z více uzlů tohoto typu aktivní právě jeden

Poznámky:

1. Má-li parametr **speed** nulovou hodnotu, avatar nemůže chodit, ale může se rozhlížet.
2. Nastavení hodnoty “NONE“ do parametru **type** způsobí vypnutí ovládacích prvků prohlížeče. Velikost rychlosti pohybu avatara pak ztratí smysl.
3. Má-li parametr **visibilityLimit** nulovou hodnotu, avatar má nekonečně daleký dohled.
4. Parametry **avatarSize**, **speed** a **visibilityLimit** jsou ovlivňovány měřítkem souřadné soustavy aktuálního stanoviště (uzlu **Viewpoint**), pokud je takové stanoviště potomkem uzlu **Transform**.

Normal (Normálové vektory)

Definice normál v trojrozměrném prostoru, doplňujících informace v ploškových geometrických objektech.

typ parametru	typ dat	název	iniciální hodnota	význam
exposedField	MFVec3f	**vector**	**[]**	seznam normálových vrcholů

Rodič: **ElevationGrid** a **IndexedFaceSet** (parametr **normal**)

Potomek: žádný

Poznámka:

Všechny vektory by měly mít jednotkovou délku. Mnoho prohlížečů však pro jistotu veškeré normálové vektory při načítání normalizuje, tj.upravuje jejich délku na jedna.

NormalInterpolator (Interpolace normál)

Uzel, který generuje seznam normálových vektorů postupnou interpolací mezi dvěma po sobě následujícími klíčovými seznamy normál. Výběr hodnot pro interpolaci je určen vstupní řídicí hodnotou.

typ parametru	typ dat	název	iniciální hodnota	význam
exposedField	MFFloat	**key**	**[]**	neklesající posloupnost řídicích klíčů
	MFVec3f	**keyValue**	**[]**	pole seznamů normál – hodnot pro odpovídající řídicí klíče

eventIn	SFFloat	**set_fraction**	aktuální řídicí hodnota

eventOut	MFVec3f	**value_changed**	seznam interpolovaných normál

Rodič: žádný nebo libovolný skupinový uzel

Potomek: žádný

Poznámka:

Protože uzel generuje najednou celý seznam normálových vektorů, počet jednotlivých normál v seznamu **keyValue** by měl být N x M, kde M je požadovaná délka výstupního seznamu normál a N počet hodnot v parametru **key**.

OrientationInterpolator (Interpolace orientace, tj. natočení)

Uzel, který generuje orientaci postupnou interpolací mezi dvěma po sobě následujícími klíčovými orientacemi. Výběr hodnot pro interpolaci je určen vstupní řídicí hodnotou.

typ parametru	typ dat	název	iniciální hodnota	význam
exposedField	MFFloat	`key`	`[]`	neklesající posloupnost řídicích klíčů
	MFRotation	`keyValue`	`[]`	seznam orientací – hodnot pro odpovídající řídicí klíče

eventIn	SFFloat	`set_fraction`	aktuální řídicí hodnota

eventOut	SFRotation	`value_changed`	interpolovaná orientace

Rodič: žádný nebo libovolný skupinový uzel

Potomek: žádný

Poznámky:

1. Uzel generuje pouze jedinou orientaci, nikoliv seznam. Orientace je dána čtveřicí čísel – první tři určují osu rotace, čtvrté úhel natočení.
2. Počet hodnot v seznamech `key` a `keyValue` by měl být shodný.
3. Svírají-li dvě po sobě následující orientace úhel větší než π, interpolace probíhá po úhlu, který je doplňkem do celé kružnice, tedy po úhlu menším.

PixelTexture (Textura určená vzorkem)

Popis obdélníkového vzorku barevných bodů a jejich případného opakovaného nanášení jako textury na povrchu.

typ parametru	typ dat	název	iniciální hodnota	význam
exposedField	SFImage	**image**	**0 0 0**	typ, šířka, výška a definice vzorku barevných bodů

field	SFBool	**repeatS**	**TRUE**	povolení opakování textury ve vodorovném směru
	SFBool	**repeatT**	**TRUE**	povolení opakování textury ve svislém směru

Rodič: **Appearance** (parametr **texture**)

Potomek: žádný

Poznámky:

1. První hodnota strukturovaného parametru **image** definuje typ barevných bodů tvořících vzorek. Typ 0 znamená žádný vzorek, typ 1 znamená odstíny šedi, typ 2 odstíny šedi a průhlednost, typ 3 znamená barevné body v modelu RGB a typ 4 je určen pro barevné body RGB doplněné průhledností.
2. Všechny barevné složky, odstíny šedi a průhlednost se zadávají v rozsahu 0 – 255. Ve smyslu průhlednosti znamená 255 zcela neprůhledný bod, 0 zcela průhledný. Pro vícesložkové body se používá hexadecimální zápis bodu ve tvaru 0xAABB…, kde AA je vyjádření první složky v šestnáctkové soustavě, BB vyjádření druhé složky atd.
3. Body vzorku se zapisují v řádcích zleva doprava, zdola nahoru.

PlaneSensor (Rovinný manipulátor)

Převod údajů ze vstupního ukazovacího zařízení (myši) na posun po povrchu pomyslné rovinné desky, která je kolmá na osu *z*.

typ parametru	typ dat	název	iniciální hodnota	význam
exposedField	SFBool	**enabled**	**TRUE**	povolení práce manipulátoru
	SFVec3f	**offset**	**0 0 0**	základní posunutí, který je vždy přičítáno k nově vyhodnocenému posunutí
	SFBool	**autoOffset**	**TRUE**	povolení automatické aktualizace **offset** po skončení činnosti manipulátoru
	SFVec2f	**minPosition**	**0 0**	dolní mez vyhodnoceného posunutí
	SFVec2f	**maxPosition**	**-1 -1**	horní mez vyhodnoceného posunutí

eventOut	SFBool	**isActive**	zahájení a ukončení činnosti manipulátoru
	SFVec3f	**trackPoint_changed**	měnící se poloha bodu na pomyslné desce, na který ukazuje kurzor při práci s manipulátorem
	SFVec3f	**translation_changed**	měnící se hodnota posunutí odvozená z pohybu vstupního zařízení

Rodič: skupinový uzel, typicky **Transform** nebo **Group** (parametr **children**)

Potomek: žádný

Zvláštnost: objekty, které jsou citlivé na vstupní ukazovací zařízení, jsou sourozenci tohoto uzlu

Poznámky:

1. Manipulátor sám o sobě nemění transformace žádného uzlu, pouze pro ně vypočítává vhodné parametry.
2. Je-li nastaven parametr **autoOffset** na **TRUE**, je po skončení činnosti manipulátoru aktualizována hodnota parametru **offset** a vyslána událost **offset_changed**.
3. Je-li libovolná ze souřadnic **maxPosition** menší než odpovídající souřadnice **minPosition**, vypočítávaná posunutí nejsou omezena. V opačném případě je pohyb kurzoru, který by generoval posunutí mimo meze, v odpovídajícím směru ignorován. V případě rovnosti jedné ze souřadnic **maxPosition** a **minPosition** převádí manipulátor pohyb kurzoru na pohyb po úsečce.

PointLight (Bodový zdroj světla)

Charakteristika světelného zdroje, vysílajícího z jednoho bodu paprsky všemi směry, jejichž intenzita klesá se vzdáleností v rámci zadaného dosahu.

typ parametru	typ dat	název	iniciální hodnota	význam
exposedField	SFVec3f	**location**	**0 0 0**	umístění bodového zdroje světla
	SFFloat	**radius**	**100**	nezáporný dosah osvětlení
	SFVec3f	**attenuation**	**1 0 0**	trojice nezáporných koeficientů pro výpočet útlumu světla
	SFColor	**color**	**1 1 1**	barva paprsků
	SFFloat	**intensity**	**1**	iniciální intenzita paprsků v rozsahu [0, 1]
	SFFloat	**ambientIntensity**	**0**	příspěvek zdroje k nepřímému osvětlení virtuálního světa v rozsahu [0, 1]
	SFBool	**on**	**TRUE**	zapnutí či vypnutí světelného zdroje

Rodič: žádný nebo skupinový uzel (parametr **children**)

Potomek: žádný

Poznámky:

1. Tento světelný zdroj osvětluje celý virtuální svět, tedy nejen sourozenecké stromy.
2. Parametry **location** a **radius** jsou ovlivňovány případnými transformacemi rodičovského uzlu **Transform**.
3. Označíme-li trojici koeficientů útlumu (**attenuation**) jako $[a_0, a_1, a_2]$, pak je celkový útlum světelného paprsku ve vzdálenosti d od zdroje světla vypočítán podle vzorce $1/(a_0 + a_1 d + a_2 d^2)$. Je-li výsledek větší než jedna, celkový útlum je nastaven na jedna. Aby nedocházelo k dělení nulou, je trojice koeficientů [0, 0, 0] vždy chápána jako [1, 0, 0].

PointSet (Množina bodů)

Definice skupiny barevných bodů v prostoru.

typ parametru	typ dat	název	iniciální hodnota	význam
exposedField	SFNode	**coord**	**NULL**	seznam bodů v uzlu **Coordinate**
	SFNode	**color**	**NULL**	seznam barev bodů v uzlu **Color**

Rodič: **Shape** (parametr **geometry**)

Potomek: v parametru **coord** uzel **Coordinate**

v parametru **color** uzel **Color**

Poznámky:

1. Velikost bodů není definována, typicky bývá rovna velikosti jednoho pixelu na obrazovce.
2. Body nepodléhají kolizím s avatarem, pokrytí texturou a osvětlení ze zdrojů světla.

PositionInterpolator (Interpolace polohy)

Uzel, který generuje prostorové souřadnice jednoho bodu postupnou interpolací mezi dvěma po sobě následujícími klíčovými hodnotami souřadnic. Výběr hodnot pro interpolaci je určen vstupní řídicí hodnotou.

typ parametru	typ dat	název	iniciální hodnota	význam
exposedField	MFFloat	**`key`**	**`[]`**	neklesající posloupnost řídicích klíčů
	MFVec3f	**`keyValue`**	**`[]`**	seznam prostorových souřadnic – hodnot pro odpovídající řídicí klíče

eventIn	SFFloat	**`set_fraction`**	aktuální řídicí hodnota

eventOut	SFVec3f	**`value_changed`**	interpolované souřadnice bodu

Rodič: žádný nebo libovolný skupinový uzel

Potomek: žádný

Poznámky:

1. Uzel generuje souřadnice pouze jednoho bodu, pro interpolaci seznamu bodů je určen uzel **`CoordinateInterpolator`**.
2. Počet hodnot v seznamech **`key`** a **`keyValue`** by měl být shodný.

ProximitySensor (Detektor přítomnosti)

Zjištění přítomnosti avatara a jeho pohybu uvnitř pomyslného kvádru.

typ parametru	typ dat	název	iniciální hodnota	význam
exposedField	SFVec3f	**center**	**0 0 0**	těžiště kvádru vymezujícího sledovanou oblast
	SFVec3f	**size**	**0 0 0**	nezáporné rozměry kvádru
	SFBool	**enabled**	**TRUE**	povolení práce detektoru

eventOut	SFBool	**isActive**	vstup a výstup avatara do, resp. ze sledované oblasti
	SFTime	**enterTime**	čas vstupu avatara do sledované oblasti
	SFTime	**exitTime**	čas opuštění sledované oblasti avatarem
	SFVec3f	**position_changed**	poloha avatara uvnitř oblasti se změnila
	SFRotation	**orientation_changed**	orientace avatara uvnitř oblasti se změnila

Rodič: žádný nebo libovolný skupinový uzel

Potomek: žádný

Poznámky:

1. Rozměry sledované oblasti jsou transformovány podle parametrů případného rodičovského uzlu **Transform**.
2. Existuje-li několik různých uzlů **ProximitySensor**, jejichž sledované oblasti se překrývají, mohou být přítomnost a pohyb avatara detekovány současně několika uzly.
3. Je-li jeden uzel **ProximitySensor** násobně vložen na různá místa pomocí konstrukcí **DEF** a **USE**, vznikne násobná oblast jako sjednocení jednotlivých kvádrů. Kvádry se však v tomto případě nesmějí překrývat.

ScalarInterpolator (Interpolace čísla)

Uzel, který generuje neceločíselnou skalární hodnotu postupnou interpolací mezi dvěma po sobě následujícími klíčovými čísly. Výběr hodnot pro interpolaci je určen vstupní řídicí hodnotou.

typ parametru	typ dat	název	iniciální hodnota	význam
exposedField	MFFloat	**key**	**[]**	neklesající posloupnost řídicích klíčů
	MFFloat	**keyValue**	**[]**	seznam čísel – hodnot pro odpovídající řídicí klíče

eventIn	SFFloat	**set_fraction**	aktuální řídicí hodnota

eventOut	SFFloat	**value_changed**	interpolované číslo

Rodič: žádný nebo libovolný skupinový uzel

Potomek: žádný

Poznámka:

Počet hodnot v seznamech **key** a **keyValue** by měl být shodný.

Script (Skript)

Uzel, který zpracovává události pomocí podprogramů (skriptů) v programovacím jazyce Java či ECMAScript (dříve JavaScript).

typ parametru	typ dat	název	iniciální hodnota	význam
exposedField	MFString	**`url`**	**`[]`**	adresy souboru s obslužnými funkcemi či jejich přímý zápis v uzlu

field	SFBool	**`directOutput`**	**`FALSE`**	funkce mohou přímo zasílat události jiným uzlům
	SFBool	**`mustEvaluate`**	**`FALSE`**	okamžité vyvolání funkcí po přijetí události uzlem

+ libovolný počet parametrů typu:

field	typ dat	**`jméno`**	**`inic. hodnota`**
eventIn	typ dat	**`jméno`**	
eventOut	typ dat	**`jméno`**	

Rodič: žádný nebo libovolný skupinový uzel, případně konstrukce **`PROTO`**

Potomek: podle vlastních parametrů datového typu **`SFNode`** či **`MFNode`**

Poznámky:

1. Z možných adres programů (**`url`**) je vybrán pouze první dostupný program.
2. Každá vstupní událost (parametru typu **`eventIn`**) musí být obsloužena funkcí. Vstupní událost je reprezentována dvojicí dat – zasílanou hodnotou a časem události.
3. Nové parametry typu **`exposedField`** nejsou povoleny.
4. Je-li uzel propojen s jinými uzly (přes vstupní událost nebo ve formě vlastních potomků uzlu **`Script`**), může přímo číst hodnoty jejich parametrů typu **`exposedField`**. Je-li navíc nastaven parametr **`directOutput`** na **TRUE**, může uzel měnit parametry takových uzlů dynamicky vzniklým zasláním události.
5. Parametry **`directOutput`** a **`mustEvaluate`** by měly být co nejčastěji nastaveny na **FALSE**, v opačném případě se může výrazně zhoršit rychlost zpracování a prohlížení virtuálního světa.
6. Uzel je schopen pracovat, i když je zařazen do nezobrazované větve VRML stromu (např. k neaktivnímu potomku uzlu **`Switch`**).

Shape (Zobrazitelný objekt)

Provázání definice geometrického tvaru a vlastností povrchu jednoho objektu.

typ parametru	typ dat	název	iniciální hodnota	význam
exposedField	SFNode	**appearance**	**NULL**	vzhled povrchu objektu
	SFNode	**geometry**	**NULL**	geometrie objektu

Rodič: buď žádný nebo libovolný skupinový uzel, nejčastěji **Transform** (parametr **children**)

Potomek: v parametru **appearance** uzel **Appearance**

v parametru **geometry** základní tělesa **Box**, **Cone**, **Cylinder**, **Sphere** a obecné geometrické objekty **ElevationGrid**, **Extrusion**, **Text**, **IndexedFaceSet**, **IndexedLineSet**, **PointSet**

Sound (Zdroj zvuku)

Definice polohy a charakteristik zdroje zvuku, který je plně slyšitelný v oblasti dané prostorovým elipsoidem a ve větší vzdálenosti je tlumen.

typ parametru	typ dat	název	iniciální hodnota	význam
exposedField	SFNode	**source**	**NULL**	uzel se zvukovým záznamem
	SFVec3f	**location**	**0 0 0**	umístění zdroje zvuku
	SFVec3f	**direction**	**0 0 1**	směr vysílaného zvuku (hlavní osa obou charakteristických eliptických oblastí)
	SFFloat	**minBack**	**1**	nezáporná vzdálenost zadního vrcholu elipsy plné slyšitelnosti
	SFFloat	**minFront**	**1**	nezáporná vzdálenost předního vrcholu elipsy plné slyšitelnosti
	SFFloat	**maxBack**	**10**	nezáporná vzdálenost zadního vrcholu elipsy plného útlumu
	SFFloat	**maxFront**	**10**	nezáporná vzdálenost předního vrcholu elipsy plného útlumu
	SFFloat	**intensity**	**1**	intenzita zvuku v rozsahu [0, 1]
	SFFloat	**priority**	**0**	priorita mezi více zdroji zvuku v rozsahu [0, 1]

field	SFBool	**spatialize**	**TRUE**	povolení generování prostorového zvuku podle orientace avatara

Rodič: buď žádný nebo libovolný skupinový uzel, nejčastěji **Transform** (parametr **children**)

Potomek: v parametru **source** uzel **AudioClip** nebo **MovieTexture**

Poznámky:

1. Uvnitř eliptické oblasti plné slyšitelnosti (**minBack**, **minFront**) intenzita zvuku neklesá, v další vzdálenosti až do eliptické hranice plného útlumu (**maxBack**, **maxFront**) klesá lineárně (v dB). Zdroj zvuku je přitom umístěn ve společném ohnisku těchto eliptických oblastí. V iniciálním nastavení uzlu mají obě eliptické oblasti tvar koule.
2. Všechny geometrické vlastnosti zdroje zvuku jsou ovlivňovány transformacemi případného rodičovského uzlu **Transform**.
3. Je-li jako zdroj zvukového záznamu vybrána zvuková stopa z již použité pohyblivé textury (souboru MPEG), je správné namísto dvojího zápisu uzlu **MovieTexture** použít konstrukci **DEF** a **USE**.
4. Priorita je použita v případě, že prohlížeč není schopen přehrávat více zvuků současně. Tehdy jsou zvuky přehrávány podle nastavené priority s tím, že nově spouštěným zvukům může být priorita automaticky dočasně zvýšena.
5. Prostorový zvuk (**spatialize**) nepředstavuje skutečný stereo signál, ale pouze zesílení či zeslabení zvuku v jednotlivých reproduktorech (typicky levém a pravém) podle umístění zdroje vůči natočení avatara v prostoru.

Sphere (Koule)

Síť jednostranných ploch pokrývajících povrch koule se středem v počátku systému souřadnic.

typ parametru	typ dat	název	iniciální hodnota	význam
field	SFFloat	`radius`	`1`	nezáporný poloměr koule

Rodič: **`Shape`** (parametr **`geometry`**)

Potomek: žádný

SphereSensor (Kulový manipulátor)

Převod údajů ze vstupního ukazovacího zařízení (myši) na osu a úhel otočení dané pohybem po povrchu pomyslné koule. Umístění a rozměry pomyslné koule jsou dány geometrií sourozeneckých uzlů.

typ parametru	typ dat	název	iniciální hodnota	význam
exposedField	SFBool	**enabled**	**TRUE**	povolení práce manipulátoru
	SFRotation	**offset**	**0 1 0 0**	základní osa a úhel otočení, se kterými jsou vždy složeny nově vyhodnocené hodnoty
	SFBool	**autoOffset**	**TRUE**	povolení automatické aktualizace **offset** po skončení činnosti manipulátoru

eventOut	SFBool	**isActive**	zahájení a ukončení činnosti manipulátoru
	SFVec3f	**trackPoint_changed**	měnící se poloha bodu na pomyslné kouli, na který ukazuje kurzor při práci s manipulátorem
	SFRotation	**rotation_changed**	měnící se osa a otočení odvozené z pohybu vstupního zařízení

Rodič: skupinový uzel, typicky **Transform** nebo **Group** (parametr **children**)

Potomek: žádný

Zvláštnost: objekty, které jsou citlivé na vstupní ukazovací zařízení, jsou sourozenci tohoto uzlu

Poznámky:

1. Manipulátor sám o sobě nemění transformace žádného uzlu, pouze pro ně vypočítává vhodné parametry.
2. Je-li nastaven parametr **autoOffset** na **TRUE**, je po skončení činnosti manipulátoru aktualizována hodnota parametru **offset** a vyslána událost **offset_changed**.

SpotLight (Reflektor)

Charakteristika světelného zdroje, vysílajícího z jednoho bodu určeným směrem kužel paprsků. Reflektor je představován dvojicí souosých kuželů s totožným vrcholem v místě světelného zdroje. Uvnitř vnitřního kužele intenzita klesá pouze se vzdáleností od vrcholu, v oblasti vnějšího kužele klesá navíc i se zvyšujícím se úhlem od osy kužele.

typ parametru	typ dat	název	iniciální hodnota	význam
exposedField	SFVec3f	**location**	**0 0 0**	umístění vrcholu světelného kužele
	SFVec3f	**direction**	**0 0 –1**	směr osy světelného kužele
	SFFloat	**beamWidth**	**1.570796**	kladný úhel (v rozsahu do π/2) mezi osou a površkou vnitřního kužele, v němž je plný světelný tok
	SFFloat	**cutOffAngle**	**0.785398**	kladný úhel (v rozsahu do π/2) mezi osou a površkou vnějšího kužele, v němž klesá světelný tok
	SFFloat	**radius**	**100**	nezáporný dosah osvětlení
	SFVec3f	**attenuation**	**1 0 0**	trojice nezáporných koeficientů pro výpočet útlumu světla
	SFColor	**color**	**1 1 1**	barva paprsků
	SFFloat	**intensity**	**1**	iniciální intenzita paprsků v rozsahu [0, 1]
	SFFloat	**ambientIntensity**	**0**	příspěvek zdroje k nepřímému osvětlení virtuálního světa v rozsahu [0, 1]
	SFBool	**on**	**TRUE**	zapnutí či vypnutí světelného zdroje

Rodič: žádný nebo skupinový uzel (parametr **children**)

Potomek: žádný

Poznámky:

1. Reflektor osvětluje celý virtuální svět, tedy nejen sourozenecké stromy.
2. Geometrické parametry uzlu jsou ovlivňovány transformacemi případného rodičovského uzlu **Transform**.
3. Úhlový útlum v oblasti mezi vnitřním a vnějším kuželem je lineární. Pro útlum vztažený ke vzdálenosti od zdroje světla (**attenuation**) platí stejný vztah, který byl uveden v uzlu **PointLight**.
4. Nastavení parametru **beamWidth** na hodnotu větší než **cutOffAngle** způsobí ostré ořezání svazku paprsků podle úhlu **cutOffAngle**.

Switch (Přepínač, volba)

Uzel umožňující zobrazit maximálně jednoho ze svých potomků.

typ parametru	typ dat	název	iniciální hodnota	význam
exposedField	MFNode	`choice`	`[]`	seznam potomků – možností k výběru
	SFInt32	`whichChoice`	`-1`	číslo zobrazovaného potomka; hodnota –1 znamená žádný výběr

Rodič: žádný nebo skupinový uzel (parametr `children`)

Potomek: libovolný uzel, který by mohl stát na nejvyšší úrovni ve VRML souboru

Poznámky:

1. Výběr zobrazovaného potomka nepotlačuje případné aktivity ostatních, nevybraných potomků (přijímání a vysílání událostí).
2. Potomci jsou číslování od nuly.

Text (Nápis)

Definice několikařádkového oboustranného nápisu a jeho umístění v prostoru.

typ parametru	typ dat	název	iniciální hodnota	význam
exposedField	SFNode	**fontStyle**	**NULL**	styl písma v uzlu **FontStyle**
	MFString	**string**	**[]**	posloupnost textových řetězců
	SFFloat	**maxExtent**	**0.0**	nezáporná hodnota omezující maximální rozměr nápisu ve směru daném stylem písma; hodnota 0 ruší jakékoliv omezení
	MFFloat	**length**	**[]**	seznam nezáporných délek pro každý z řetězců; hodnota 0 znamená libovolnou délku

Rodič: **Shape** (parametr **geometry**)

Potomek: v parametru **fontStyle** uzel **FontStyle**

Poznámky:

1. Přesáhne-li jakýkoliv z textových řetězců rozměr **maxExtent**, je stlačen celý nápis.
2. Při zadání konkrétních délek (**length**) jsou odpovídající řetězce stlačovány či roztahovány individuálně.

TextureCoordinate (Souřadnice textury)

Definice bodů v rovině, které slouží jako vztažné body pro nanesení textury v ploškových geometrických uzlech.

typ parametru	typ dat	název	iniciální hodnota	význam
exposedField	MFVec2f	**`point`**	**`[]`**	seznam bodů v rovině

Rodič: **`ElevationGrid`** a **`IndexedFaceSet`** (parametr **`texCoord`**)

Potomek: žádný

TextureTransform (Transformace rovinné textury)

Nastavení posunu, otočení a měřítka rovinné textury.

typ parametru	typ dat	název	iniciální hodnota	význam
exposedField	SFVec2f	**translation**	**0 0**	posunutí
	SFFloat	**rotation**	**0**	úhel otočení vzhledem ke vztažnému bodu
	SFVec2f	**scale**	**1 1**	změna měřítka vzhledem ke vztažnému bodu
	SFVec2f	**center**	**0 0**	poloha vztažného bodu

Rodič: **Appearance** (parametr **textureTransform**)

Potomek: žádný

Poznámky:

1. Transformace jsou prováděny v pevném pořadí. Nejprve posunutí (**translation**), poté otočení (**rotation**) a na závěr změna měřítka (**scale**). Otočení a změna měřítka jsou prováděny vzhledem ke společnému vztažnému bodu (**center**).
2. Transformace se nevztahují přímo k textuře, nýbrž k jejímu definičnímu oboru mapovanému do povrchové oblasti o rozměru 1x1. To se projevuje obrácením důsledku transformačních koeficientů. Měřítko **scale** o velikosti [2, 2] způsobí zmenšení výsledné textury na polovinu, posunutí o 3 m doprava způsobí skutečný posun textury o 3 m doleva apod.

TimeSensor (Časovač)

Generování událostí v různých časech.

typ parametru	typ dat	název	iniciální hodnota	význam
exposedField	SFTime	**startTime**	**0**	absolutní čas, po jehož dosažení zahájí časovač práci
	SFTime	**stopTime**	**0**	absolutní čas, po jehož dosažení ukončí časovač práci.
	SFTime	**cycleInterval**	**1**	nezáporná délka časové smyčky; v sekundách
	SFBool	**loop**	**FALSE**	povolení generování časových událostí v nekonečné smyčce
	SFBool	**enabled**	**TRUE**	povolení práce časovače

eventOut	SFBool	**isActive**	bylo zahájeno či ukončeno generování časových událostí časovačem
	SFTime	**time**	plynule generovaný absolutní čas
	SFTime	**cycleTime**	absolutní čas zahájení další časové smyčky
	SFFloat	**fraction_changed**	plynulé generovaná poměrná hodnota uplynulého času v rámci jedné smyčky, tj. v rozsahu [0, 1]

Rodič: žádný

Potomek: žádný

Poznámky:

1. Absolutní čas je měřen v sekundách, počínaje 1. lednem 1970, nikoliv tedy od vstupu uživatele do virtuálního prostoru.
2. Je-li povolena časová smyčka, nabývá parametr **fraction_changed** nulové hodnoty pouze jedinkrát, na úplném počátku. Veškeré další průchody začátkem nových časových smyčky jsou ohodnoceny číslem 1, tj. jako ukončení průchodu předchozí časovou smyčkou.
3. Pokud je hodnota **stopTime** menší než hodnota **startTime**, časovač může pracovat neustále v časové smyčce. V opačném případě ukončí svoji práci po dosažení času **stopTime**, když předtím proběhlo libovolné množství časových smyček.
4. Množství generování časových událostí (frekvence) závisí na rychlosti zpracování virtuálního světa. Vždy je však zajištěno korektní generování času na začátku a na konci práce časovače.
5. V průběhu práce ignoruje časovač vstupní události, které mění délku smyčky (**cycleInterval**) a startovní čas (**startTime**), případně koncový čas (**stopTime**), pokud je jeho hodnota menší než aktuální skutečný čas.

TouchSensor (Detektor dotyku)

Zjištění geometrických údajů o objektech, nad kterým se pohybuje kurzor. Objekty jsou umístěny v sourozeneckých uzlech.

typ parametru	typ dat	název	iniciální hodnota	význam
exposedField	SFBool	**enabled**	**TRUE**	povolení práce detektoru

eventOut	SFBool	**isOver**	kurzor se dostal nad objekt nebo jej opustil
	SFBool	**isActive**	stav tlačítka pro aktivaci kurzoru se změnil
	SFTime	**touchTime**	čas uvolnění tlačítka pro aktivaci kurzoru
	SFVec3f	**hitPoint_changed**	bod na povrchu objektu, na který ukazuje kurzor, se změnil
	SFVec3f	**hitNormal_changed**	normála v bodě, který odpovídá parametru **hitPoint_changed**
	SFVec2f	**hitTexCoord_changed**	souřadnice textury v bodě, který odpovídá parametru **hitPoint_changed**

Rodič: skupinový uzel, typicky **Transform** nebo **Group** (parametr **children**)

Potomek: žádný

Zvláštnost: objekty, které jsou citlivé na polohu a aktivitu kurzoru, jsou sourozenci tohoto uzlu

Poznámky:

1. Pokud uživatel přesunuje kurzor nad objektem, jsou vysílány události ze všech parametrů typu **eventOut** s výjimkou parametrů **isActive** a **touchTime**. Ty jsou významné teprve v situaci, kdy uživatel aktivuje kurzor, tj. v případě použití myši stiskne tlačítko.
2. Parametr **touchTime** vysílá událost tehdy, pokud uživatel nejprve aktivuje kurzor nad objektem (stiskne tlačítko) a v okamžiku uvolnění tlačítka se kurzor stále nachází nad tímtéž objektem.
3. Vyskytuje-li se ve stromové struktuře více uzlů **TouchSensor** nad sebou, reaguje na aktivity kurzoru pouze nejnižší z nich, tj. „nejmladší“ potomek. Pokud jsou objekty svázány s různými detektory a manipulátory, reagují tyto uzly současně.

Transform (Umístění skupiny)

Nastavení společné polohy, měřítka a otočení potomků.

typ parametru	typ dat	název	iniciální hodnota	význam
exposedField	SFVec3f	**scale**	**1 1 1**	změna měřítka ve směru os *x*, *y* a *z*
	SFRotation	**scaleOrientation**	**0 0 1 0**	osa a úhel natočení provedeného před změnou měřítka
	SFRotation	**rotation**	**0 0 1 0**	osa a úhel otočení
	SFVec3f	**center**	**0 0 0**	vztažný bod, vůči kterému se provádí otočení
	SFVec3f	**translation**	**0 0 0**	posunutí
	MFNode	**children**	**[]**	seznam potomků

field	SFVec3f	**bboxSize**	**-1 -1 -1**	kladné délky stran pomocné obálky ve tvaru kvádru; výjimka [-1, -1, -1] indikuje dosud nespecifikovanou velikost kvádru
	SFVec3f	**bboxCenter**	**0 0 0**	souřadnice středu pomocné obálky ve tvaru kvádru

eventIn	MFNode	**addChildren**	seznam připojovaných uzlů - potomků
	MFNode	**removeChildren**	seznam odstraňovaných uzlů - potomků

Rodič: žádný nebo skupinový uzel (parametr **children**, případně **choice**, **level** či **proxy**)

Potomek: libovolný

Poznámky:

1. Transformace se provádějí v pevném pořadí. Nejprve změna měřítka (**scale**) podle daného směru (**scaleOrientation**), poté otočení (**rotation**) vzhledem ke vztažnému bodu (**center**) a nakonec posunutí (**translation**).
2. Parametr **scale** může nabývat pouze kladných hodnot. Nelze proto docílit například zrcadlení nastavením koeficientů měřítka na mínus jedna.

Viewpoint (Stanoviště)

Umístění a pohledové charakteristiky stanoviště ve virtuálním světě.

typ parametru	typ dat	název	iniciální hodnota	význam
exposedField	SFFloat	`fieldOfView`	`0.785398`	zorný úhel v rozsahu (0,π)
	SFVec3f	`position`	`0 0 10`	poloha avatara v libovolných souřadnicích
	SFRotation	`orientation`	`0 0 1 0`	natočení avatara podle osy a úhlu; osa se zadává vektorem s hodnotami v rozsahu [-1,1], úhel je libovolný
	SFBool	`jump`	`TRUE`	povolení plynulého přechodu na stanoviště
field	SFString	`description`	`""`	jméno stanoviště uváděné prohlížečem
eventIn	SFBool	`set_bind`		aktivování stanoviště, tj.umístění avatara do dané polohy a orientace
eventOut	SFTime	`bindTime`		čas aktivace stanoviště
	SFBool	`isBound`		stanoviště se stalo aktivním nebo naopak bylo nahrazeno jiným stanovištěm

Rodič: žádný nebo skupinový uzel (parametr `children`)

Potomek: žádný

Zvláštnost: v každém okamžiku je z více uzlů tohoto typu aktivní právě jeden

Poznámky:

1. Je-li parametr `jump` nastaven na **FALSE**, pak skok na toto stanoviště není plynule animován. Parametry nového stanoviště jsou pouze připraveny k použití a stanou se aktivními teprve tehdy, je-li programově (například zasláním události) změněna poloha nebo orientace tohoto stanoviště. Některé prohlížeče nastavení parametru `jump` ignorují.
2. Stanoviště může být potomkem uzlu `Transform`. V takovém případě je ovlivňováno jeho transformačními parametry.

VisibilitySensor (Detektor viditelnosti)

Zjištění, zda je na obrazovce vidět alespoň část pomyslného kvádru.

typ parametru	typ dat	název	iniciální hodnota	význam
exposedField	SFVec3f	**center**	**0 0 0**	těžiště kvádru, jehož viditelnost je sledována
	SFVec3f	**size**	**0 0 0**	nezáporné rozměry kvádru
	SFBool	**enabled**	**TRUE**	povolení práce detektoru

eventOut	SFBool	**isActive**	část kvádru se objevila na obrazovce, resp. kvádr přestal být viděn (zobrazován)
	SFTime	**enterTime**	čas, ve kterém nabude parametr **isActive** hodnoty **TRUE**
	SFTime	**exitTime**	čas, ve kterém nabude parametr **isActive** hodnoty **FALSE**

Rodič: žádný nebo libovolný skupinový uzel

Potomek: žádný

Poznámky:

1. Rozměry sledované oblasti jsou transformovány podle parametrů případného rodičovského uzlu **Transform**.
2. Existuje-li několik různých uzlů **VisibilitySensor**, jejichž sledované oblasti se překrývají, může být jejich viditelnost detekována současně několika uzly.
3. Je-li jeden uzel **VisibilitySensor** násobně vložen na různá místa pomocí konstrukcí **DEF** a **USE**, vznikne násobná oblast jako sjednocení jednotlivých kvádrů. Kvádry se však v tomto případě nesmějí překrývat.

WorldInfo (Globální informace)

Textové údaje vztahující se nejčastěji k autorovi virtuálního světa, datu vzniku souboru, použitém programu.

typ parametru	typ dat	název	iniciální hodnota	význam
field	SFString	`title`	`""`	jméno virtuálního světa
	MFString	`info`	`[]`	posloupnost řetězců s libovolným významem

Rodič: žádný

Potomek: žádný

Literatura

1. R. Carey, G. Bell – *The Annotated VRML97 Reference Manual*, Addison Wesley, 1997, ISBN: 0-201-41974-2.
2. J. D. Foley, A. van Dam, S. Feiner, J. F. Hughes - *Computer Graphics - Principles and Practice*, Addison Wesley, 1990.
3. J. Hartman, J. Wernecke – *The VRML Handbook*, Addison Wesley, 1996, ISBN: 0-201-47944-3.
4. http://www.web3d.org/ - dříve *VRML, nyní Web3D Consortium.*
5. ISO/IEC 10646-1:1993 - I*nformation technology - Universal Multiple-Octet Coded Character Set (UCS) - Part 1: Architecture and Basic Multilingual Plane*, 1993.
6. ISO/IEC 14772-1:1997 - *Information technology - Computer graphics and image processing - The Virtual Reality Modeling Language (VRML) – Part 1: Functional specification*, 1997.
7. ISO/IEC DIS 16262 - *Information technology - ECMAScript: A general purpose, cross-platform programming language.*
8. J. Žára, B. Beneš, P. Felkel - *Moderní počítačová grafika*, Computer Press, 1998, ISBN: 80-7226-049-9.

Rejstřík

Anchor (teleportace, odkaz)
Appearance (vzhled povrchu)
AudioClip (zvukový soubor)
Background (pozadí, panorama)
Billboard
Box (kvádr)
Collision (detekce nárazu)
Color (barvy)
ColorInterpolator (interpolátor barvy, interpolace barvy)
Cone (kužel)
Coordinate (souřadnice prostorových bodů)
CoordinateInterpolator (interpolátor souřadnic, interpolace souřadnic)
Cylinder (válce)
CylinderSensor (válcový manipulátor)
DirectionalLight (směrový zdroj světla)
ElevationGrid (výšková mapa)
Extrusion (opláštění)
Fog (mlha)
FontStyle (styl písma)
Group (skupina)
ImageTexture (obrazová textura, textura určená obrázkem)
IndexedFaceSet (množina ploch)
IndexedLineSet (množina čar)
Inline (vložení)
LOD (stupeň detailu)
Material (barevné vlastnosti)
MovieTexture (pohyblivá textura)
NavigationInfo (ovládání avatara)
Normal (normálové vektory)
NormalInterpolator (interpolátor normál, interpolace normál)
OrientationInterpolator (interpolátor orientace, interpolace orientace)
PixelTexture (textura určená vzorkem)
PlaneSensor (rovinný manipulátor)
PointLight (bodový zdroj světla)
PointSet (množina bodů)
PositionInterpolator (interpolátor polohy, interpolace polohy)
ProximitySensor (detektor přítomnosti)
ScalarInterpolator (interpolátor čísla, interpolace čísla)
Script (skript)
Shape (zobrazitelný objekt)
Sound (zdroj zvuku)
Sphere (koule)
SphereSensor (kulový manipulátor)
SpotLight (reflektor)
Switch (přepínač, volba)
Text (nápis)
TextureCoordinate (souřadnice textury)
TextureTransform (transformace rovinné textury)
TimeSensor (časovač)
TouchSensor (detektor dotyku)
Transform (umístění skupiny, transformace)
Viewpoint (stanoviště)
VisibilitySensor (detektor viditelnosti)
WorldInfo (globální informace)