Vážení zákazníci,

dovolujeme si Vás upozornit, že na tuto ukázku knihy se vztahují autorská práva, tzv. copyright.

To znamená, že ukázka má sloužit výhradně pro osobní potřebu potenciálního kupujícího (aby čtenář viděl, jakým způsobem je titul zpracován a mohl se také podle tohoto, jako jednoho z parametrů, rozhodnout, zda titul koupí či ne).

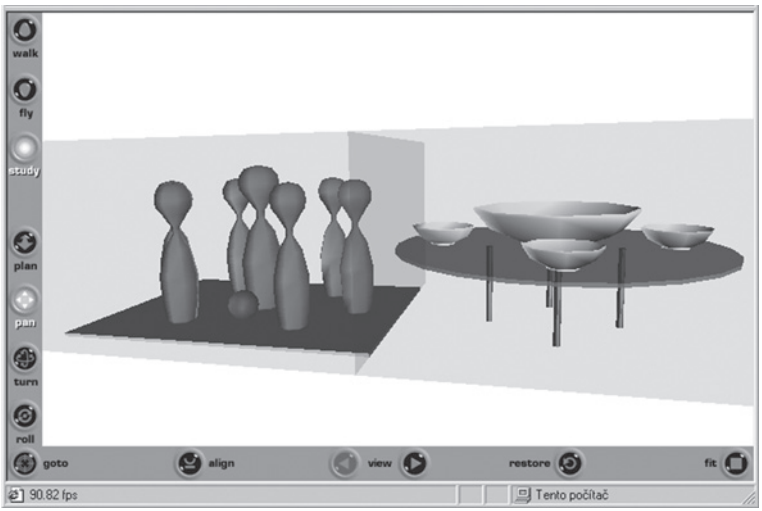
Z toho vyplývá, že není dovoleno tuto ukázku jakýmkoliv způsobem dále šířit, veřejně či neveřejně např. umisťováním na datová média, na jiné internetové stránky (ani prostřednictvím odkazů) apod.

redakce nakladatelství BEN – technická literatura
redakce@ben.cz

Část II

Jedna scéna nestačí

Vložení scény – Inline



Obr. 19 S19a.wrl

Co to ukazuje:

Použití dříve vytvořených scén jako stavebních prvků pro složitější scénu.

Jak to napsat:

```
Transform {
   translation  -10 -5.8 0
   children Inline   {
      url   "kuzelkyII.wrl"
   }
}
Transform {
   translation  10 -0.9 0
   children Inline   {
      url   "misky.wrl"
   }
}
```

Vysvětlení:

Z kódu je snad patrné, že uzel `Inline` povoluje vkládat na dané místo scény celé dříve vytvořené světy (VRML soubory).

Další příklad:

S019b.wrl scény obsahující mlhu

Poznámky:

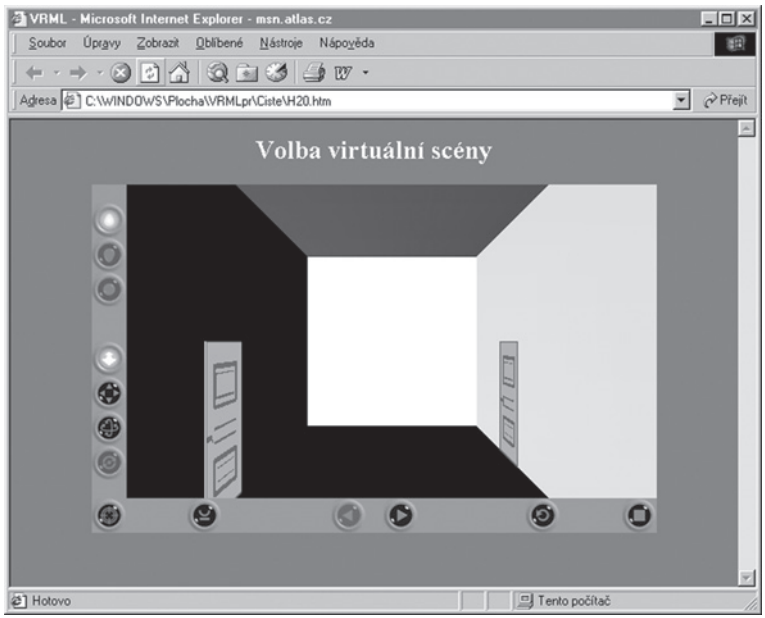
** Umístění vložených světů upravíme uzlem* `Transform`*.*

** Spolu s vloženým světem jsou přístupné i jeho* `ViewPoint`*y (i se svými jmény).*

Pozor na světla ve vložených scénách – je třeba upravit jejich dosvit tak, aby nesvítila „do vedlejší místnosti". Prohlížeče by měly zobrazit vložené scény včetně jejich prostředí (mlha apod.).

** Parametr* `url` *vždy umožňuje určit několik cest k vložené scéně – pro případ nedostupnosti prvního souboru.*

** Uvedený příklad je demonstrační, obvykle neklademe celé světy takto vedle sebe.Uzel* `Inline` *použijeme s výhodou později – jako potomka uzlu* `LOD` *(LevelOfDetail).*



Obr. 20 H20.htm, S20a.wrl

Co to ukazuje:

Vložení VRML Scény do dokumentu HTML dokumentu a přechod (teleportaci) mezi scénami.

Jak napsat stránku:

```
<html>
<head>
<title> VRML </title>
</head>
<body text="white" bgcolor="green">
 <h2 align="center"> Volba virtuální scény </h2>
<center>
<embed src="S20a.wrl" width="80%" height="120%">
</center>
</body>
</html>
```

Vysvětlení:

Zobrazení scény z HTML dokumentu v okně prohlížeče umožní tag `embed` nebo `object`. `Url` adresa scény je uvedena v jeho parametru `src`. Prostý odkaz na VRML scénu obstará tag `<a href=url>…</a>`.

Přesun do jiné scény – teleportace

Co to ukazuje:

Přechod (teleportaci) mezi scénami.

Jak napsat teleportaci:

```
Anchor {
   url          "kII.wrl"
   children Shape {
      geometry IndexedFaceSet   {
      coord Coordinate {
           point [1.98 0 -3, 1.98 2 -3, 1.98 2 -4, 1.98 0 -4]
}
      coordIndex [0 1 2 3 -1]
      texCoord TextureCoordinate {point [1 0, 1 1, 0 1, 0 0]}
      }
      appearance Appearance {
           texture    ImageTexture {url "dverep.gif"}
      }
}}
Anchor {
   url     "kuzII.wrl"
   children Shape {
     geometry IndexedFaceSet    {
      coord Coordinate {
        point [-1.98 0 -1, -1.98 2 -1, -1.98 2 -2, -1.98 0 -2]}
      coordIndex [3 2 1 0 -1]
      texCoord TextureCoordinate {point [0 0, 0 1, 1 1, 1 0]}
      }
      appearance Appearance {
      texture    ImageTexture { url  "dverep.gif"}
      }
}}
```

Vysvětlení:

Stejně jako uvnitř párového HTML tagu `<a> Klikni </a>` (anchor) musí být zobrazitelný obsah, na nějž je možno kliknout, musí mít i VRML uzel `Anchor` v parametru `children` nějaké geometrické potomky (či potomky obsahující geometrii). Parametrem uzlu je `url` scény, do níž bude Avatar teleportován.

Další příklad:

S020b.wrl užití parametru `parameter` – otevření scény v novém okně

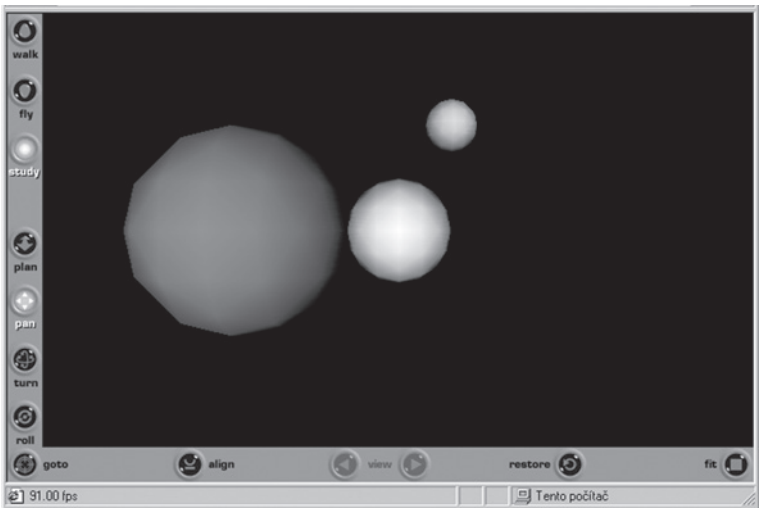
Poznámky:

** Ve scénách, do nichž budeme teleportováni, by měly být také uzly* `Anchor` *pro návrat zpět. Historie prohlížeče nemusí teleportaci zaznamenávat a tlačítko „Zpět“ v prohlížeči pak nepomůže. Pak zkuste znovu načíst stránku. Skupina scén tvořících virtuální svět by měla vzájemnou teleportaci podporovat – projdu-li dveřmi do jiné místnosti, měly by z ní vést dveře zpět ven.*

** Je možné teleportovat se i na jiné stanoviště (*`ViewPoint`*) téže scény:*

```
Anchor { url "#Vzadu" ...}
```

Vlastní uzly – prototypy I



Obr. 21 P01a.wrl

Co to ukazuje:

Vytvoření vlastního uzlu s požadovanými parametry. Kouli, u níž lze nastavit barvu a určit polohu bez vkládání do uzlu `Transform`.

Jak to napsat:

```
PROTO Mic [
   field SFFloat polomer 1
   field SFVec3f translation 0 0 0
   field SFColor barva 1 1 1
]
{
   Transform {
     translation IS translation
     children [
           Shape {
                geometry Sphere { radius IS polomer }
                appearance Appearance {
                     material Material {
                          diffuseColor IS    barva
                     }
                }
           }
     ]
   }
}

Mic {}
Mic {
   polomer 2
   translation  -3 0 0
   barva  1 0 0
}
Mic {
   polomer .5
   translation 1 2 0
   barva 0 1 0
}
```

Vysvětlení:

Konstrukce `DEF-USE` dovolovala „kopírovat“ ve scéně stejné, pojmenované uzly (stromy). Jejich vlastnosti nebylo možné měnit. Konstrukce `PROTO` umožňuje vytvářet vlastní prototypy plnohodnotných uzlů a dávat jim potřebné parametry. Tak může například námi definovaný geometrický uzel obsahovat parametry ovlivňující jeho vzhled a polohu – ač to není zcela obvyklé, protože geometrický uzel lze vždy vložit jako potomka do uzlu `Shape` a ten následně do uzlu `Transform`.

V definici prototypu následuje za hlavičkou `[seznam parametrů prototypu]`, kde se pro každý parametr uvede:`<třída><typ><jméno><iniciální_hodnota>`. Za seznamem parametrů následuje {popis prototypu}. {popis prototypu} připomíná scénu.

V `{popisu prototypu}` pak jako hodnoty zvolených parametrů některých uzlů použijeme jména parametrů prototypu pomocí konstrukce `IS`.

Další příklady:

P01b.wrl jednostranná a oboustranná deska
P01c.wrl prototyp souřadnicové osy z příkladu S04c.wrl

Poznámky:

** Parametry nově definovaného uzlu jsou typů MF(SF)xx (viz přílohu A) a navíc mohou být různých tříd (podle účelu, pro nějž je chceme použít – podrobněji viz kapitoly o komunikaci tj. předávání událostí mezi uzly). Prozatím se spokojíme s parametry třídy* `field`*. Pro různé druhy komunikace mezi uzly si později ukážeme ještě parametry tříd* `exposedField`*,* `eventIn` *a* `eventOut`*. Typem parametru zde tedy budeme rozumět typ jeho dat, třídou způsob jeho komunikace s okolními uzly. V literatuře se můžete setkat s jiným označením.*

** Pokud v definici prototypu obsahuje* `{popis prototypu}` *několik zobrazitelných uzlů (les), zobrazí se pouze první z nich. V takovém případě je potřeba vnořit vše, co se má zobrazit, do nějakého skupinového uzlu – např.* `Group`*. První uzel (strom) v* `{popisu prototypu}` *také určuje druh prototypu – geometrie, materiál apod. (a tím i pozici v grafu scény, kam lze prototyp umístit).*

`{popis prototypu}` *není strom scény! Samostatně v něm mohou být (a často jsou) i uzly, které v grafu scény nemohou být kořenovými uzly.*

** Každý parametr* `field` *prototypu musí mít v definici uvedenu iniciální hodnotu. Zařadíme-li uzel typu* `Mic` *z uvedeného příkladu do scény bez uvedení parametrů, bude to koule bílé barvy, s poloměrem 1, umístěná v počátku.*

** Definice prototypu je opravdu pouze definice. Neznamená automatické zařazení tohoto uzlu (s iniciálními hodnotami parametrů) do scény. Tam zařazujeme nově definovaný uzel zcela stejně jako standardní uzly jazyka VRML.*

** Nelekejte se přiřazení* `translation IS translation`*. Nejen, že není chybou, ale může zpřehlednit definici prototypu. Znamená: „parametr* `translation` *uzlu* `Transform` *bere svou hodnotu z hodnoty parametru* `translation` *prototypu.“*

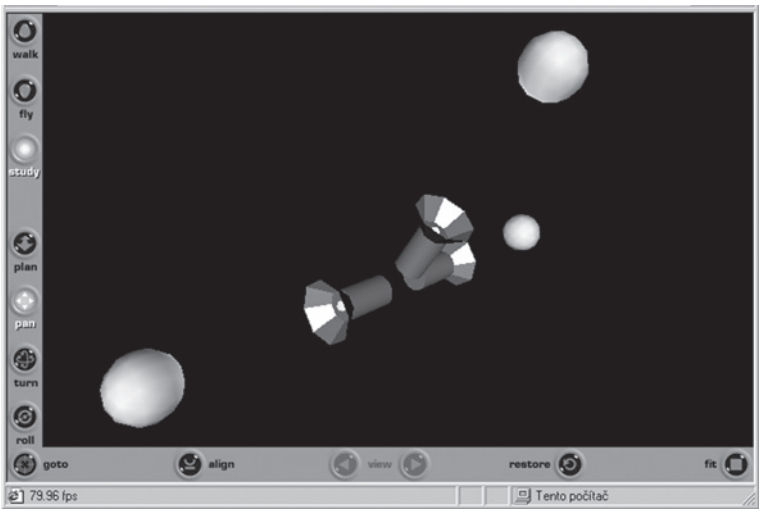
** Prototyp nemusí mít žádné parametry. Užitečná může být například konstrukce jednostranné obdélníkové desky:*

```
PROTO Deska [
] # povinné závorky
{
   IndexedFaceSet {
      coord Coordinate {point [0 0 0 , 1 0 0 ,1 1 0, 0 1 0]}
      coordIndex [0 1 2 3 -1]
   }
}
```

Tento prototyp zjednodušuje zadávání obdélníkových ploch a lze ho použít jako geometrii uzlu `Shape`*. Do scény uzel zařadíme zápisem* `Deska{}` *(se závorkami).*

** Vytvořené prototypy můžeme použít v dalších scénách – viz další kapitoly.*

Baterka – prototypy II



Obr. 22 P02.wrl

Co to ukazuje:

Parametr typu SFNode (uzel) a použití prototypu v prototypu.

Jak to napsat:

```
PROTO Reflektor [
]
{
    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 1 0]
      scale  [1 1, 2 2]
      beginCap FALSE
      endCap FALSE
      solid FALSE
    }
}

PROTO Baterka [
   field SFNode appearance NULL
   field SFColor barva_svetla 1 1 1
   field SFFloat dosvit 10
]
{
Transform {
    translation  0 .5 0
    children [
      Shape {
            geometry Cylinder {
                           radius .3
                           height 1
            }
            appearance IS appearance
      }
      Transform {
            translation     0 .5 0
            scale .3 .4 .3
            children  Shape {
                  geometry Reflektor {}
                  appearance IS appearance
            }
      }
      Transform {
            translation     0 0.5 0
            children  Shape {
                  geometry Sphere { radius .1}
                  appearance Appearance {
                        material Material {
                                    emissiveColor IS barva_svetla
                        }
                  }
```

```
            }
        }
        SpotLight {
            location 0 0.6 0
            radius IS dosvit
            color IS barva_svetla
            direction 0 1 0
            cutOffAngle .9
        }
    ]
}
} # konec prototypu

# vlastní scéna - první baterka

Transform {
    scale 2 2 2
    translation 0 1 0
    children
      Baterka   {
            appearance Appearance  {
                material Material {
                                diffuseColor .4 .4 .4
                                specularColor .7 .9 .9
                                shininess .13
                }
            }
            barva_svetla 1 1 0
      }
}

# . . . další baterky
```

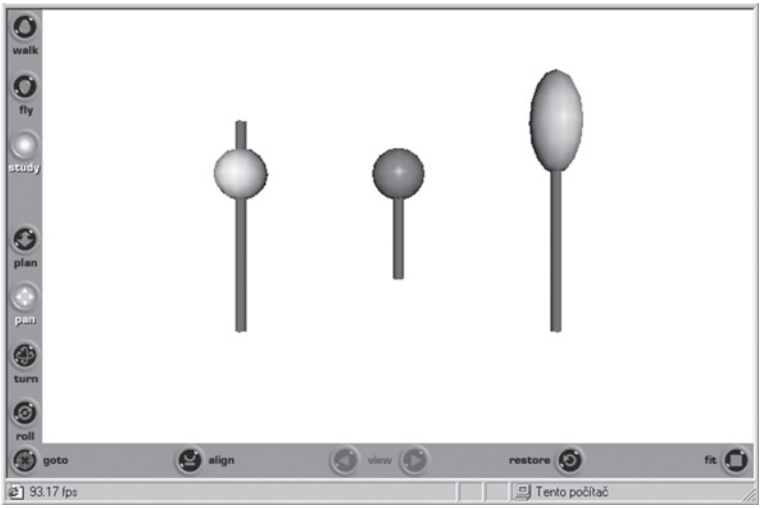
Vysvětlení:

Prototyp baterky sestává z válce, žárovky a kuželového reflektoru – uzel `Cone` však vytváří plochu zevnitř neviditelnou a proto si vytvoříme vlastní prototyp `Reflektor`, který vhodnou plochu vytvoří. Třeba pomocí uzlu `Extrusion` s parametrem `solid` hodnoty `FALSE`. Nový uzel (prototyp) použijeme stejně jako ostatní geometrické uzly při definici prototypu baterky.

Poznámka:

** Iniciální hodnota parametru* `appearance Baterky` *je prázdná – hodnota* `NULL`*. Vzhled takové baterky je tedy takový jako v příkladu S01.wrl.*

Lízátko – prototypy III



Obr. 23 P03.wrl

Co to ukazuje:

Pokus o vytvoření geometrie tvaru „lízátka“ s nastavitelnou délkou špejle. Pokus o něco, co nejde udělat bez programování.

Jak to NEpsat:

```
PROTO Lizatko [
    field SFFloat delka_spejle 2
    field SFFloat sirka_spejle .1
    field SFFloat polomer .5
    field SFColor barva 1 0 0
]{
Group {
    children [
      Shape {
           geometry Cylinder {
                     radius IS sirka_spejle
                     height IS delka_spejle
           }
           appearance Appearance {
                material Material {
                     diffuseColor .5 .3 0
           }}
      }
      Transform {
           translation 0 1 0
           children  Shape {
                geometry Sphere { radius IS polomer }
                appearance Appearance {
                     material Material {
                          diffuseColor IS barva
                          specularColor .8 .8 .8
                          ambientIntensity .5
                          shininess .8
                     }}

           }}]
}}
Lizatko {} # Lizatko s inicialnimi hodnotami parametru

Transform {
    translation  -3 0 0
    children Lizatko  {
                delka_spejle 4
                barva 1 1 0
           }
}
Transform {
    translation  3 0 0
    scale 1 2 1
    children Lizatko { barva 0 1 0 }
}
```

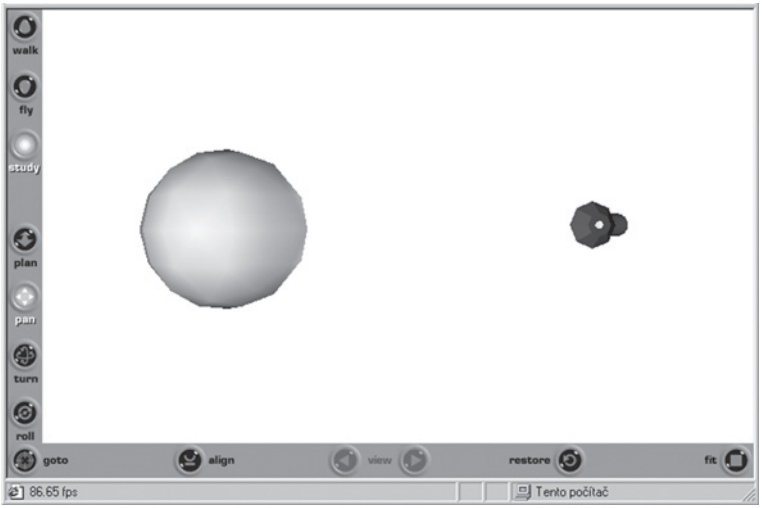
Vysvětlení:

Problém tohoto prototypu spočívá v posunutí kuličky. Druhá souřadnice posunutí závisí na hodnotě parametru `delka_spejle`. Chyba je na řádce označené ♦. Místo posunutí `0 1 0`, které je při iniciální hodnotě parametru `delka_spejle` správné, bychom měli uvést obecnější posunutí `0 delka_spejle/2 0`. VRML však dovoluje pouze přiřazení parametrům ve tvaru `parametr IS parametr_téhož_typu`. Nedovoluje žádné výpočty, ani „rozebírání" strukturovaných parametrů. Potřebného posunutí docílíme jedině výpočtem prostřednictvím kódu uzlu `Script`. (Řešení najdete ve scéně SC03d.wrl.)

Poznámka:

** Ani pokus řešit problém změnou měřítka nevede k cíli. Změna měřítka sice vysune střed kuličky do správné polohy, zároveň však kuličku zdeformuje.*

EXTERNPROTO



Obr. 24 P04.wrl

Co to ukazuje:

Použití uzlů, jejichž prototypy byly definovány v jiných souborech, jejich zařazení do scény.

Jak to napsat:

```
EXTERNPROTO Svitilna [
   field SFColor barva_svetla
   field SFNode appearance
] "P02.wrl#Baterka"
EXTERNPROTO     Mic [
   field SFFloat polomer
   field SFVec3f translation
   field SFColor barva
] "P01a.wrl#Mic"

Mic {
   translation -6 0 0
   barva 1 1 0
}
Transform {
   translation 5 0 0
   rotation 0 0 1 1.57
   children Svitilna {
      barva_svetla 0 1 0
      appearance Appearance {
           material Material {diffuseColor .3 .2 0}
      }}
}
```

Vysvětlení:

Konstrukce `EXTERNPROTO` zprostředkovává přístup k prototypům již definovaným konstrukcí `PROTO` v jiných VRML souborech. Obsahuje seznam těch parametrů prototypu, které chceme v nové scéně nastavovat a uplatňovat (ne nutně všech) bez uvedení jejich iniciálních hodnot. Za seznamem parametrů musí být uvedena cesta (seznam cest) k souboru a prototypu ve tvaru <url_souboru_s_prototypem>#<původní_jméno_prototypu>. Uzel můžeme v nové scéně přejmenovat.

Poznámky:

** Není-li v cestě k prototypu* uvedeno `původní_jméno_prototypu`*, bere se první prototyp v uvedeném souboru.*

** Parametry uvedené v* `EXTERNPROTO` *vybereme z parametrů zdroje. Musí se shodovat s původními parametry prototypu v třídě, typu i jméně.*