VRML 2.0

1. Introdução à VRML

A Internet é uma rede de computadores internacional que conecta Universidades, Empresas, Centros de Pesquisa, Lares e Departamentos do Governo. Pode-se pensar na Internet como a maior rede de computadores do mundo, e como uma poderosa ferramenta de comunicação. Atualmente um dos recursos mais utilizados da Internet é a **WWW** *(World Wide Web*), que provê uma forma de acesso e recuperação de informações através de *links* a documentos, onde um *link* corresponde a uma conexão entre pontos da rede que permite que sejam feitas referências a outros documentos, outras seções do próprio documento ou figuras nas páginas de um serviço de informações na WWW. Pode-se caracterizar WWW como um conjunto de informações distribuídas pela Internet, onde o usuário "navega" através dos diversos *sites*.
É possível "navegar" através da Internet usando uma grande variedade de aplicações, sendo que as mais comuns são os *Web Browsers*, ou navegadores. Este tipo de aplicação permite que o usuário acesse diversos endereços através da grande quantidade de informações disponíveis na *Web*, e visualize a grande quantidade de informação disponível. Entre os vários navegadores disponíveis, os mais utilizados são *Netscape Navigator* e *Microsoft Internet Explorer*. Todos os navegadores exibem documentos com textos formatados e imagens que estejam incluídas de maneira correta. A formatação do texto é controlada por uma linguagem chamada HTML (*HyperText Markup Language*). Através desta linguagem é possível formatar um documento através de comandos, ou seqüências especiais de caracteres (*tags*).
Dentro de um documento HTML é possível inserir *links* que conectem um documento a outros documentos na *Web*. Cada *link* é "ancorado", ou refere-se, a uma palavra, ou frase, ou linha de texto, ou figura no documento. A maioria dos navegadores exibem estes *links* sublinhando o texto "âncora" na página. Clicando-se no texto "âncora" o navegador é direcionado a seguir o *link* e recuperar o documento referenciado. Neste momento é verificado o tipo de informação que ele contém. Se for um texto HTML ou uma imagem, o navegador exibe o documento. Entretanto, para apresentar outros tipos de informações, tais como sons, animações, e o mundo VRML 3D, o navegador passa esta informação para aplicações *helper* ou para *plug-ins* dos navegadores. Um *helper* consiste em um programa que entende o conteúdo e o formato destes outros tipos de informações, e um *plug-in*, por sua vez, é um programa que permite visualizar informações que não sejam HTML dentro da janela do navegador.
Para visualizar documentos VRML, é necessário uma aplicação *helper* ou um *plug-in*. A lista de *helpers* e *plug-ins* para visualização de arquivos VRML está crescendo rapidamente. Os mais comuns são: *Silicon Graphics' CosmoPlayer*, *Sony's Community Place*, *Intervista's World View*, *VREAM's Wirl* e *Dimension X's Liquid*

Reality. A figura 1.1 mostra a imagem de um exemplo apresentado por Ames [AME 97] exibido através do *Silicon Graphics' CosmoPlayer*, um *plug-in* VRML que permite exibir o mundo VRML dentro da janela do navegador, no caso, o *Netscape*. Devido ao fato de que os navegadores HTML e VRML são, normalmente, adquiridos separadamente, deve-se configurar o navegador HTML de maneira que ele automaticamente inicie o navegador VRML quando um arquivo VRML for carregado.
Um *link* em um documento VRML ou HTML corresponde a um endereço da Web. Endereços de documentos Web são especificados através de URLs (*Uniform Resource Locators*). Uma URL é formada por três partes principais:

- O nome do protocolo de comunicação necessário para recuperar um arquivo;
- O nome do computador, ou *host*, na Internet;
- O caminho do diretório e do arquivo para o arquivo que está sendo recuperado do *Host*.

Por exemplo, a seguinte URL é o endereço Web para um documento HTML que contém um repositório VRML: http://www.sdsc.edu/vrml . Neste caso, "http" é o nome do protocolo de comunicação usado pela maioria dos navegadores (*hipertext transfer protocol*), "www.sdsc.edu" é o nome do *host* na Internet, e "/vrml" indica o nome do arquivo que deve ser exibido.

Nos últimos anos VRML tem cada vez mais sido aceito como uma tecnologia padrão da *Web* para exibição de conteúdo gráfico 3D. VRML representa um meio rico de expressão de idéias na *Web*, uma vez que é interativo e pode conter até mesmo animação e som. Consistindo num primeiro passo a caminho da *Web* 3D, imersiva e interativa, VRML é uma linguagem bastante simples e acessível.

2. Conceitos Importantes de VRML

Neste capítulo são apresentados os principais conceitos de VRML, necessários para a criação do "mundo virtual". Na primeira seção os componentes de um arquivo VRML são identificados, enquanto nas seções seguintes a construção de figuras, a definição do espaço de trabalho VRML e a descrição de eventos e rotas são descritos.

2.1 Arquivo VRML

Um arquivo VRML, que consiste em uma descrição do "mundo" VRML, contém textos que podem ser criados em qualquer editor ou processador. Este arquivo, que é um arquivo ASCII, também pode ser criado através da utilização de aplicações que permitem a edição em três dimensões ou de utilitários que traduzem outros formatos de arquivos gráficos para VRML. O arquivo VRML, identificado pela extensão ".wrl", descreve como construir *shapes* (= figuras ou

formas), onde colocá-las, que cores terão, e assim por diante. A expressão "mundo" VRML é usada para referenciar arquivos VRML, pois quando o navegador lê um arquivo VRML, ele constrói o "mundo" descrito no arquivo. Conforme o usuário move-se ao longo do "mundo", o navegador desenha, ou exibe, este "mundo".
Arquivos VRML podem conter quatro tipos principais de componentes:

- *Header*;
- *Prototypes*;
- *Shapes*, *Interpolators*, *Sensors*, *Scripts*;
- *Routes*;

Nem todos os arquivos contêm todos estes componentes. Na verdade o único item **obrigatório** em qualquer arquivo VRML é o *header*. Porém, sem pelo menos uma figura, o navegador não exibirá nada ao ler o arquivo. Outros componentes que um arquivo VRML também pode conter, são:

- *Comments*;
- *Nodes*;
- *Fields*, *field values*;
- *Defined node names*;
- *Used node names*;

A figura 2.1 mostra um exemplo de um "mundo" VRML com um *header* e um *group* que contém nós (*nodes*), campos e comentários. O arquivo VRML que deve ser carregado em um navegador para exibir esta figura, é apresentado a seguir, e o seu conteúdo será descrito nas próximas seções e capítulos.

Figura 2.1 - Primeiro arquivo VRML

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright (c) 1997
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
# A brown hut
Group {
   children [
      # Draw the hut walls
      Shape {
         appearance DEF Brown Appearance {
            material Material {
               diffuseColor 0.6 0.4 0.0
            }
```

```
            }
            geometry Cylinder {
                height 2.0
                radius 2.0
            }
        },
        # Draw the hut roof
        Transform {
            translation 0.0 2.0 0.0
            children Shape {
                appearance USE Brown
                geometry Cone {
                    height 2.0
                    bottomRadius 2.5
                }
            }
        }
    ]
}
```

Antes de começar a descrever o conteúdo do arquivo deste primeiro exemplo, torna-se importante salientar que os navegadores pulam, ou ignoram, espaços, vírgulas, tabulações e linhas em branco, entretanto eles diferenciam letras minúsculas de maiúsculas. Sendo assim, é fundamental que a sintaxe dos comandos VRML seja memorizada.

A primeira linha do arquivo do exemplo anterior contém o **cabeçalho** (*header*) que é obrigatório em qualquer arquivo VRML e deve aparecer na primeira linha exatamente como mostrado no exemplo. Ele descreve que é um arquivo VRML, da versão 2.0 que usa o conjunto de caracteres internacional UTF-8, que é uma maneira padrão de digitar caracteres em muitas linguagens. Já os **comentários** permitem a inclusão de informação extra que não afeta a aparência do "mundo" VRML Eles iniciam com o caracter "#" e terminam no final da linha.

Os **nós** são os blocos básicos de construção de VRML, e o seu nome ou tipo indica o objeto que ele descreve, que são basicamente *shapes* e suas propriedades. Nós individuais descrevem *shapes*, cores, luzes, como posicionar e orientar *shapes*, etc. Um nó geralmente contém:

- O tipo do nó (obrigatório), como *Shape*, *PointLight*, *Box*, etc.
- Um par de chaves (obrigatório).
- Campos (opcional), que definem os atributos dos nós e por sua vez possuem:
    - Nome;
    - Tipo, que define o tipo de valor permitido;
    - Valor *default*, que é utilizado caso não sejam especificados outros valores.

Exemplos de nós:

```
Cylinder          # tipo do nó
{                 # abre chave
    height 2.0  # campo:
    radius 2.0  # campo:
}                 # fecha chave
```

```
Sphere {
   radius 2
}

Box { }
```

É importante salientar que o valor de um campo define atributos como cor, tamanho ou posição, e cada valor é de um tipo específico, que descreve o tipo de valor permitido naquele campo. Estes tipos possuem nomes tais como "*SFColor*" e "*SFImage*". A maioria dos tipos dos campos são: tipos de um único valor (por exemplo, só uma cor ou um número), cujos nomes começam por "*SF*"; e tipos com múltiplos valores (por exemplo uma lista de cores ou números), cujos nomes começam por "*MF*". Os tipos de campos VRML são: *SFBool*, *SFColor*, *MFColor*, *SFFloat*, *MFFloat*, *SFImage*, *SFInt32*, *MFInt32*, *SFNode*, *MFNode*, *SFRotation*, *MFRotation*, *SFString*, *MFString*, *SFTime*, *SFVec2f*, *MFVec2f*, *SFVec3f*, *MFVec3f*.
Também é possível definir um nome, que consiste numa seqüência de letras, números e sublinhas, para qualquer nó. Uma vez que um nó tem um nome, este nó pode ser usado posteriormente no arquivo. Por exemplo, pode-se especificar o nome "cadeira" para um nó ou grupo de nós usados na construção de uma cadeira. Depois, para colocar quatro ao redor de uma mesa, é possível simplesmente reutilizar este nó, sem tem que descrever toda cadeira novamente. O nó com o nome definido é chamado "nó original" e cada reutilização deste nó é chamada "instância". Para definir um nó para ser instanciado, basta colocar a palavra "**DEF**" antes do nome escolhido. E para usar este nó novamente dentro do arquivo basta colocar a palavra "**USE**" na frente do nome. A sintaxe destes comandos é a seguinte:

```
DEF   nome_do_nó   tipo_do_nó   { ... }
USE   nome_do_nó
```

2.2. Construção de Shapes em um Arquivo VRML

Em VRML, os blocos básicos de construção são *shapes*, descritos através de nós e seus campos. Um *shape* descreve a forma, ou geometria, da estrutura 3D do objeto, e a sua aparência, com base no material a partir do qual é feito e na sua textura, tal com madeira. Estes dois atributos, geometria e aparência, são especificados por campos dentro do nó ***Shape***. Sua sintaxe está descrita a seguir.

```
Shape {
          appearance   ...
          geometry       ...
}
```

No exemplo da figura 2.1, são criados dois *shapes*, o primeiro é um cilindro e o segundo é um cone. Ambos definem a geometria do *shape* e compartilham uma aparência comum. VRML suporta vários tipos de geometrias primitivas, que são pré-definidas, incluindo *Sphere*, *Cylinder*, *Box*, *Cone*, bem como vários outros *shapes* avançados, tais como "*extruded shapes*" e "*elevation grids*". Além da

geometria, um *Shape* também possui propriedades, como material e textura, que definem seu aspecto. Estas propriedades são descritas no campo *appearance*. *Shapes* podem ser agrupados para construção de formas mais complexas. O cilindro e o cone do exemplo da figura 2.1 são agrupados através do nó *Group* para criar uma cabana, que por sua vez pode ser agrupada para criar uma cidade. O nó que agrupa *shapes* é chamado *parent*. Os *shapes* que fazem parte do grupo são chamados ***children***. Um grupo pode ter qualquer número de nós *children*, assim como podem ter outros grupos como *children*. Neste caso se diz que um grupo está aninhado (*nested*) em outro.

2.3. Espaço VRML

Os nós e campos em um arquivo VRML fornecem instruções de construção para criação de características de um mundo virtual. Tais instruções devem incluir distâncias e tamanhos precisos para controlar o tamanho e posicionamento dos *shapes* construídos no espaço 3D. Entretanto, torna-se importante atentar para o fato de que as unidades em VRML não são equivalentes a alguma unidade de medida do mundo real, tais como polegadas ou centímetros. Elas descrevem um tamanho ou uma distância dentro do contexto do mundo VRML.
Inicialmente, deve-se relembrar que um sistema de coordenadas é composto por dois eixos, X e Y, e pela origem (0.0, 0.0). Uma coordenada é formada pelo valor de X e de Y, que correspondem aos números ao longo dos eixos X e Y, respectivamente. Sendo assim, desenhar uma figura em duas dimensões torna-se bastante simples, basta dar a seqüência de coordenadas necessárias, e então imaginar que uma "caneta" irá ligar estes pontos para formar a figura final. Porém, a linguagem VRML é utilizada para desenhar figuras em três dimensões. Neste caso, é acrescentado um terceiro eixo ao sistema de coordenadas, o eixo Z. Os eixos X, Y e Z formam o sistema de coordenadas 3D, cuja origem consiste na coordenada espacial (0.0, 0.0, 0.0). Agora, uma "caneta virtual" pode ser movida para esquerda e para direita, para cima e para baixo e para frente e para trás. Para facilmente identificar como o eixo Z é posicionado em relação a X e Y pode-se utilizar a regra da mão direita para os eixos 3D. Nesta regra a mão direita deve ficar reta com o indicador apontando para direção positiva de Y (para cima), o polegar apontando para a direção positiva de X (para o lado) e com o dedo do meio apontando para a direção positiva de Z (para frente). A figura 2.2 mostra o "funcionamento" da regra da mão direita.

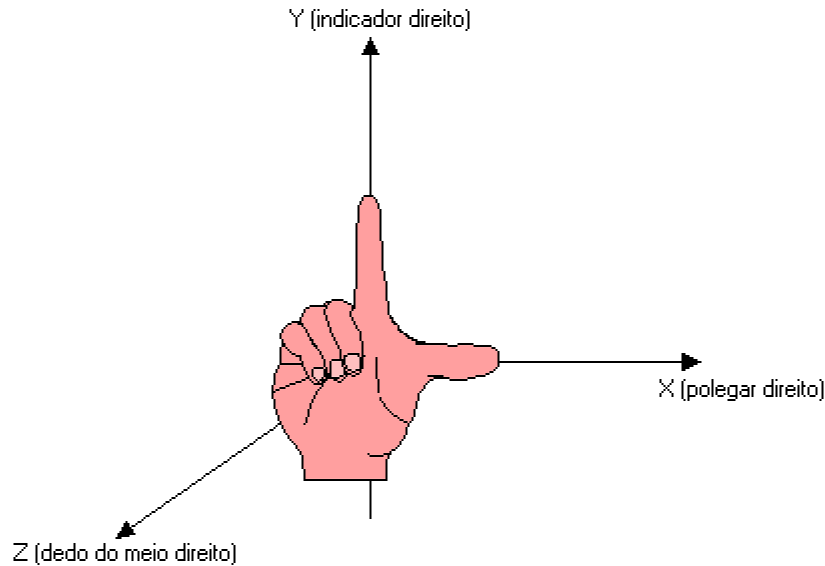

Figura 2.2 – Regra da mão direita para os eixos 3D

2.4. Eventos e Rotas

Eventos e rotas possibilitam a criação de instruções que permitem a interatividade e o dinamismo no "mundo VRML". Através da criação de uma rota entre dois nós, onde o primeiro envia eventos (mensagens) para o segundo, é possível, por exemplo, clicar num objeto e fazer com que uma luz acenda ou uma música toque. Esta ligação em VRML envolve:

- Um par de nós, que serão ligados;
- Uma rota (*route* ou *path*) entre os dois nós.

Uma vez que a rota é construída entre dois nós, o primeiro nó pode enviar mensagens ao segundo nó ao longo desta rota. Tal mensagem, chamada de evento (*event*), contém um valor similar ao valor de campos dentro dos nós. Valores típicos de eventos incluem ponto flutuante, cor ou coordenada 3D. Quando um nó recebe um evento, ele reage ligando a luz, tocando uma música, iniciando uma animação, etc., dependendo das características do nó. Ligando múltiplos nós é possível criar circuitos complexos para tornar o "mundo VRML" dinâmico.
A maioria dos nós podem ser ligados a um circuito. Cada nó possui pinos de entrada e de saída aos quais podem ser feitas ligações. Por exemplo, nós que produzem luz possuem um pino de entrada para ligá-la ou desligá-la. Um nó pode ter vários pinos de entrada e saída disponíveis. No caso da luz, ainda pode-se ter pinos de entrada para trocar sua cor, sua intensidade, etc. Alguns tipos de nós possuem pinos de entrada e saída, enquanto outros têm apenas um dos dois tipos. Um pino de entrada de um nó é chamado de *eventIn*. Um *eventIn* recebe eventos quando é conectado a uma rota e quando um evento é enviado para ele. Similarmente, um pino de saída é chamado de *eventOut*. Um *eventOut* envia

eventos quando é conectado a uma rota. Da mesma maneira que o campo de um nó, cada *eventIn* e *eventOut* para um nó também tem um tipo. Um *eventOut* do tipo *SFFloat*, por exemplo, produz como saída um valor de ponto flutuante quando conectado a um circuito. Um *eventOut* do tipo *SFColor* envia cores. Similarmente, um *eventIn* do tipo *SFFloat* pode receber valores de ponto flutuante, e *SFColor* pode receber cores. Torna-se importante salientar que no momento de especificar uma rota, através do comando *ROUTE*, os tipos dos *eventIn* e *eventOut* devem ser iguais.

Um circuito VRML é construído descrevendo-se a rota a partir do *eventOut* de um nó para o *eventIn* de outro nó. A rota do circuito permanece "parada" até que um evento é enviado do primeiro nó para o segundo ao longo da rota. Quando o primeiro nó ativa a rota enviando um evento, este "viaja" até o segundo nó que reage. O tipo de reação depende:

- Do tipo do nó que recebe o evento;
- Do pino de entrada ao qual foi feita a ligação;
- Dos valores contidos no evento;
- Das atividades correntes do nó.

Por exemplo, quando um evento contendo um valor *SFBool TRUE* é enviado para o *eventIn* "on" de um nó de luz, a luz é ligada. Da mesma forma, quando um evento contendo um valor de ponto flutuante é enviado para o *eventIn* "intesidade" de um nó de luz, o brilho da luz é alterado.

Resumindo: circuitos podem ser construídos através da construção de rotas entre nós; uma rota conecta o *eventOut* de um nó ao *eventIn* de outro nó; o primeiro nó pode então enviar eventos ao segundo, fazendo com que segundo nó reaja.

Para ilustrar, o exemplo a seguir descreve um cubo que gira em torno do seu próprio eixo quando o usuário move o cursor sobre ele. Neste caso, o *eventOut isOver* do nó *TouchSensor* é enviado para o nó *TimeSensor*. O nó *TimeSensor* é enviado para o nó *OrientationInterpolator* que direciona a rota para o nó *Transform*. Quando o cursor move-se sobre o cubo. O nó *TouchSensor* envia *TRUE* usando o *eventOut isOver* que permite que o nó *TimeSensor* direcione o nó *OrientationInterpolator* e gire o cubo. Quando o cursor sai de cima do cubo, *FALSE* é enviado usando o *eventOut isOver* do nó *TouchSensor*, desabilitando o nó *TimeSensor* e parando a animação.

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright (c) 1997
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
# A cube that spins when the viewer's cursor moves over it

Group{
	children [
		# Cubo que gira quando o usuário clica sobre ele
		DEF Cube Transform {
			children Shape {
				appearance Appearance {
```

```
                                        material Material { }
                                }
                                geometry Box { }
                        }
                },

                # Sensor, detecta quando o usuário clica em um objeto
                DEF Touch TouchSensor { },

                # relógio para controlar a animação
                DEF Clock TimeSensor {
                        enabled FALSE
                        cycleInterval 4.0
                        loop TRUE
                },

                # Caminho da animação (movimento)
                DEF CubePath OrientationInterpolator {
                        key [0.0, 0.50, 1.0]
                        keyValue [
                                0.0 1.0 0.0 0.0,
                                0.0 1.0 0.0 3.14,
                                0.0 1.0 0.0 6.28
                        ]
                }
        ]
}

# Rotas
ROUTE Touch.isOver           TO Clock.set_enabled
ROUTE Clock.fraction_changed TO CubePath.set_fraction
ROUTE CubePath.value_changed TO Cube.set_rotation
```

3. Construindo e Agrupando Shapes Pré-Definidos

O VRML possui algumas figuras pré-definidas, que são o cubo, o cone, o cilindro e a esfera, para construção do "mundo" VRML. Estas figuras são chamadas de figuras ou formas primitivas, ou simplesmente primitivas. Alterando estas figuras, isto é, trocando o valor de suas coordenadas (como, por exemplo, para "esticá-la" ou "achatá-la"), rotacionando-as ou agrupando-as, pode-se criar uma série de objetos.
Como já comentado, um *shape* VRML possui geometria e aparência, que são definidas no nó *Shape*. A aparência é descrita pelos nós *Appearance* e *Material*, que serão descritos posteriormente. As primitivas de geometria oferecidas no "mundo" VRML, e que podem ser usadas com o nó *Shape* para construir *shapes* primitivos, incluem: cubo, cone, cilindro e esfera. Cada nó de geometria primitiva possui um ou mais campos que permitem especificar atributos como as dimensões

do cubo, o raio da esfera ou a altura do cilindro e do cone. As primitivas são sempre construídas centralizadas na origem, e são consideradas como sólidos.
O nó *Shape*, usado para construção de todos os *shapes* VRML, tem a seguinte sintaxe:

```
Shape{
        appearance      NULL    # SFNode
        geometry NULL   # SFNode
}
```

O valor do campo *geometry* especifica um nó que define a forma 3D, ou geometria, do *shape*. Valores para este campo, tipicamente, incluem os nós *Box*, *Cone*, *Cylinder* e *Sphere*. O valor *default NULL* indica a falta de geometria. Já o valor do campo *appearance* especifica um nó que define a aparência do *shape*, incluindo sua cor e textura. Valores para este campo incluem o nó *Appearance* e o valor *default NULL*, que faz com que o objeto fique com uma cor branca.
A sintaxe do nó *Appearance*, usado no campo *appearance* do nó *Shape* descrito acima, é:

```
Appearance {
        material                NULL    # SFNode
        texture                         NULL    # SFNode
        textureTransform        NULL    # SFNode
}
```

Neste caso, o valor do campo *material* especifica um nó que define os atributos do material. Normalmente é utilizado o nó *Material* neste campo. O valor *default NULL*, determina um material branco. Os outros campos serão descritos posteriormente.
O nó *Material*, que por sua vez especifica os atributos do material que compõe o objeto, possui a sintaxe descrita a seguir. Os valores *default* para os campos deste nó, que serão detalhadamente descritos mais adiante, cria um *shape* branco com "sombras".

```
Material {
        ambientIntensity        0.2                     # SFFloat
        diffuseColor                    0.8 0.8 0.8             # SFColor
        emissiveColor                   0.0 0.0 0.0             # SFColor
        shininess               0.2                     # SFFloat
        specularColor                   0.0 0.0 0.0             # SFColor
        transparency                    0.0                     # SFFloat
}
```

O cubo, ou caixa, é criado através do nó *Box* e pode ser usado como valor do campo *geometry* no nó *Shape*. A sintaxe deste nó é a seguinte.

```
Box {
        size 2.0 2.0 2.0        # SFVec3f
}
```

O valor do campo *size* especifica o tamanho de uma caixa retangular tridimensional centrada na origem. O primeiro valor deste campo corresponde à largura da caixa na direção do eixo X, o segundo valor corresponde à altura na direção Y e terceiro valor corresponde à profundidade na direção Z. Todos estes três valores devem ser maior do que 0.0. Os valores de tamanho de campo *default* são 2.0.

Para ilustrar, o exemplo abaixo descreve a criação do cubo que aparece na figura 3.1.

```
#VRML V2.0 utf8
# Desenho de um cubo vermelho
Shape {
        appearance Appearance {
                material Material {
                        diffuseColor 1.0 0.0 0.0
         }
        }
        geometry Box {
                size 2.5 2.5 2.5
        }
}
```

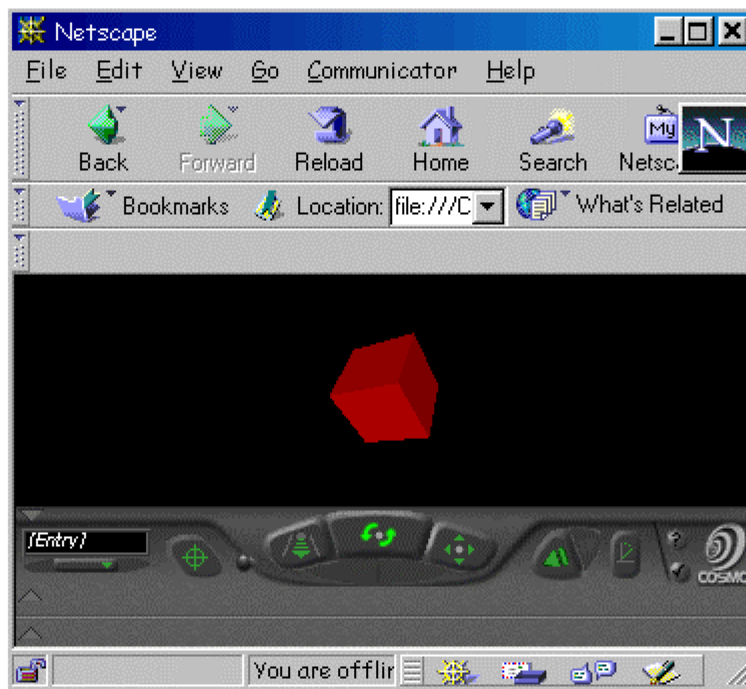

Figura 3.1 – Exemplo da criação de um cubo vermelho

De forma análoga ao nó *Box*, o nó *Cone* é usado para criar um cone. Os valores *default* para cada um dos campos deste nó são os apresentados na sintaxe descrita a seguir. Neste nó, o valor do campo *bottomRadius*, que deve ser maior do que 0.0, especifica o raio da base de um cone tridimensional centrado na origem cujo eixo cresce em direção ao eixo Y (o eixo de um cone corresponde a uma linha imaginária que vai do centro da base até o topo do cone).

```
Cone {
        bottomRadius    1.0     # SFFloat
        height          2.0     # SFFloat
        side            TRUE    # SFBool
        bottom          TRUE    # SFBool
}
```

O valor do campo *height*, que também deve ser maior do que 0.0, especifica a altura do cone na direção do eixo Y. O valor do campo *side* especifica quando ou não os lados do cone devem ser construídos. Se o valor for *TRUE* eles devem

aparecer, se for *FALSE* não. Da mesma foram, o valor do campo *bottom* especifica quando o círculo da base deve ser construído ou não. O próximo exemplo mostra o código para a criação do cone da figura 3.2.

```
#VRML V2.0 utf8
# Desenho de um cone amarelo
Shape {
          appearance Appearance {
                    material Material {
                              diffuseColor 1.0 1.0 0.0
           }
          }
          geometry Cone {
                    bottomRadius 1.0
                    height      5.0
                    side        TRUE
                    bottom      TRUE
          }
}
```

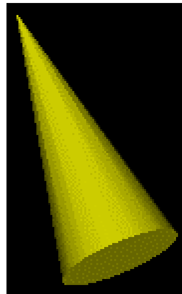

Figura 3.2 – Exemplo da criação de um cone amarelo

A sintaxe do nó *Cylinder*, usado para criar um cilindro, com os valores *default* para seus campos é a seguinte:

```
Cylinder {
          radius              1.0       # SFFloat
          height              2.0       # SFFloat
          side                TRUE      # SFBool
          top                 TRUE      # SFBool
          bottom              TRUE      # SFBool
}
```

Neste caso, o valor do campo *radius* especifica o raio do cilindro que é centrado na origem, e o valor do campo *height* especifica a sua altura. Ambos os valores devem ser maior do que 0.0. Assim como no nó *Cone*, os valores dos campos *side*, *top* e *bottom* determinam, respectivamente, se os lados, o topo e a base do cilindro vão ser desenhados ou não. O código que exemplifica a criação de um cilindro, com a sua respectiva visualização na figura 3.3, é apresentado a seguir.

```
#VRML V2.0 utf8
# Desenho de um cilindro cyan
Shape {
          appearance Appearance {
                    material Material {
                              diffuseColor 0.0 1.0 1.0
           }
```

```
        }
        geometry Cylinder {
                radius      4.0
                height      3.0
                side        TRUE
                top         TRUE
                bottom      TRUE
        }
}
```

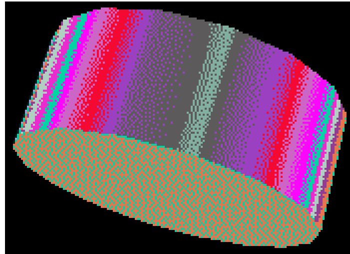

Figura 3.3 – Exemplo da criação de um cilindro *cyan*

O nó *Sphere*, usado para criação de esferas centradas na origem, possui um único campo, *radius*, que determina o seu raio e cuja valor *default* é 1.0. Sua sintaxe é:

```
Sphere {
        radius              1.0       # SFFloat
}
```

A figura 3.4 ilustra a visualização do exemplo a seguir:

```
#VRML V2.0 utf8
# Desenho de uma esfera verde
Shape {
        appearance Appearance {
                material Material {
                        diffuseColor 0.0 0.8 0.0
         }
        }
        geometry Sphere {
                radius      4.0
        }
}
```

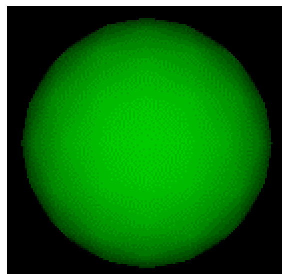

Figura 3.4 – Exemplo da criação de uma esfera verde

Como já comentado, é possível agrupar qualquer número de nós e manipular o grupo como uma única entidade. Este grupo pode ter qualquer número de membros, ou *children* (filhos), que podem também ser *shapes* ou outros grupos que contém *shapes* e grupos. O nó que contém os nós filho é chamado de *parent* (pai). Como grupos podem conter outros grupos, o pai de um grupo pode ser filho

de outro grupo de nível mais alto, e assim por diante. Então, o pai do grupo de nível mais alto é chamado de *root* (raiz).

VRML fornece vários tipos de nós de agrupamento. A seguir, está apresentada a sintaxe de um deles, o nó *Group*.

```
Group {
        children    [ ]                 # MFNode
        bboxCenter          0.0 0.0 0.0         # SFVec3f
        bboxSize  -1.0 -1.0 -1.0        # SFVec3f
        addChildren                             # MFNode
        removeChildren                          # MFNode
}
```

O valor do campo *children* especifica uma lista de nós *child* que deve ser incluída no grupo. Valores de campo típicos para *children* incluem o nó *Shape* e outros nós *Group*. Quando é exibido um arquivo com este nó, o navegador VRML constrói o grupo através da construção de cada um dos *Shapes* e *Groups* contidos no *Group*. O valor *default* para este campo é uma lista vazia de *children*. O próximo exemplo (figura 3.5) ilustra a utilização deste nó.

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright (c) 1997
# Andrea L. Ames, David R. Nadeau, and John L. Moreland

Group {
   children [
      Shape {
         appearance DEF White Appearance {
            material Material { }
         }
         geometry Box {
            size 10.0 10.0 10.0
         }
      },
      Shape {
         appearance USE White
         geometry Sphere {
            radius 7.0
         }
      },

      Shape {
         appearance USE White
         geometry Cylinder {
            radius 12.5
            height 0.5
         }
      },
      Shape {
         appearance USE White
         geometry Cylinder {
```

```
            radius 4.0
            height 20.0
         }
      },
      Shape {
         appearance USE White
         geometry Cylinder {
            radius 3.0
            height 30.0
         }
      },
      Shape {
         appearance USE White
         geometry Cylinder {
            radius 1.0
            height 60.0
         }
      }
   ]
}
```

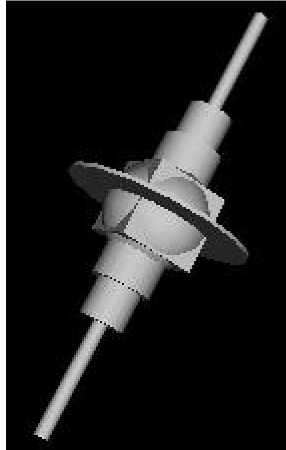

Figura 3.5 – Exemplo da utilização do nó *Group*

4. Construindo Shapes de Texto

Através do nó *Text*, pode-se adicionar *shapes* de texto no "mundo" VRML. Para cada texto, pode-se especificar uma lista de *strings* com seus respectivos tamanhos. E usando o nó *FontStyle*, pode-se controlar o tipo de fonte, seu estilo, tamanho, espaçamento, etc.
Assim como os nós *Box*, *Sphere*, etc. são usados para especificar a geometria dos *shapes* construídos através do nó *Shape*, o nó *Text* é usado para criar a geometria de textos no espaço. Os campos do nó *Text* permitem especificar os caracteres que devem ser construídos e os atributos que controlam como cada *string* é construído. Neste momento torna-se interessante comentar que, além dos nós *shape* e geometria, VRML suporta uma variedade de nós *property* que encapsulam

settings que podem ser usados várias vezes para controlar como os nós são construídos. No caso de textos, o nó *property* é o *FontStyle*.
O nó *Text*, que cria um texto e pode ser usado como o valor do campo *geometry* no nó *Shape*, tem a seguinte sintaxe:

```
Text{
        string          [ ]     # MFString
        length          [ ]     # MFFloat
        maxExtent       0.0     # SFFloat
        fontStyle   NULL        # SFNode
}
```

O valor do campo *string* especifica uma ou mais linhas de texto que serão exibidas. Cada linha ou coluna de texto aparece entre aspas e separada por vírgula, entre colchetes. O valor *default* para este campo é uma lista vazia de *strings*. O nó *Text* cria caracteres "planos", com a profundidade Z = 0.0. Por *default*, os caracteres são colocados lado a lado, ao longo do eixo X da esquerda para a direita. Linhas de *strings* consecutivos são colocadas uma embaixo da outra, descendo no eixo Y. Estes valores *default* podem ser trocados usando campos do nó *FontStyle* que são especificados como valores do campo *fontStyle*.
O campo *length* especifica o tamanho desejado, em unidades VRML, de cada linha de texto. Para respeitar este tamanho, linhas ou colunas de texto são comprimidas ou expandidas através da alteração do tamanho do caracter ou do espaço entre os caracteres. O valor 0.0 especifica que o texto será escrito no seu tamanho natural, sem comprimir ou expandir. Cada linha de texto tem seu próprio tamanho através da inclusão de múltiplos valores de *length* entre colchetes e separados por vírgula.
Já o valor do campo *maxExtent* especifica o tamanho máximo permitido, em unidades VRML, de qualquer linha ou coluna de texto, sendo que este valor deve maior ou igual a 0.0. Neste caso, linhas ou colunas mais longas do que a sua extensão máxima são comprimidas através da redução do tamanho ou espaçamento dos caracteres. Linhas ou colunas mais curtas não são afetadas. O valor *default* para este campo é 0.0, que indica que não há limite para ser usado.
As características que definem a aparência de um texto criado com o nó *Text* são especificadas através do valor do campo *fontStyle*. Tipicamente, o valor deste campo é o nó *FontStyle*, e o valor *default* NULL indica que será usado o estilo de fonte *default*, que instrui o nó *Text* a exibir o texto alinhado à esquerda, fonte *serif*, tamanho 1.0 unidade de altura, espaçamento entre linhas de 1.0 unidade, da esquerda para a direita horizontalmente e de cima para baixo verticalmente.
A seguir é apresentado um exemplo da utilização do nó *Text* com sua respectiva visualização na figura 4.1.

```
#VRML V2.0 utf8
# Exemplo de utilização do nó Text
Shape {
        appearance Appearance {
                material Material {
                        diffuseColor 0.0 0.0 0.8
         }
        }
        geometry Text {
```

```
                string [ "Linha Um", "Linha Dois" ]
        }
}
```

Figura 4.1 – Exemplo da utilização do nó Text

O nó *FontStyle*, que controla as características que definem a aparência do texto criado através do nó *Text*, geralmente é usado como valor do campo *fontStyle*. Sua sintaxe é:

```
FontStyle{
        family          "SERIF"         # SFString
        style           "PLAIN"         # SFString
        size            1.0             # SFFloat
        spacing         1.0             # SFFloat
        justify         "BEGIN"         # SFString
        horizontal TRUE         # SFBool
        leftToRight     TRUE            # SFBool
        topToBottom     TRUE            # SFBool
        language  ""            # SFString
}
```

O significado dos campos é o seguinte:

- *family*: especifica qual dos padrões de fonte VRML será usado, *SERIF* (semelhante à *Times Roman*) *SANS* (semelhante à *Helvetica*) ou *TYPEWRITER* (semelhante à *Courier*). O valor *default*, que também pode ser representado por "", é *SERIF*.
- *style*: indica o estilo de texto que será usado, *PLAIN* (*default*, plano/liso), *BOLD* (negrito), *ITALIC* (itálico) ou *BOLDITALIC* (negrito e itálico).
- *size*: determina a altura do caracter medida em unidades VRML (*default*=1.0).
- *spacing*: especifica o espaçamento de linha vertical em unidades VRML.
- *horizontal*: quando o valor é *TRUE* (*default*), cada *string* do campo *string* do nó *Text* constrói uma linha horizontal de texto; quando é *FALSE*, é construída uma coluna vertical.

- *leftToRight* e *topToBottom*: são usados em combinação com *horizontal*. Para um texto horizontal, se *leftToRight* for *TRUE*, caracteres consecutivos são colocados lado a lado da esquerda para a direita, se for *FALSE*, caracteres consecutivos são colocados da direita para a esquerda ao longo do eixo X negativo. Da mesma forma, o valor do campo *topToBottom* especifica como *strings* consecutivos são colocados ao longo da direção vertical (*TRUE*, um embaixo do outro, *FALSE*, um em cima do outro). Para um texto vertical, os efeitos destes campos são análogos. Se *topToBottom* é *TRUE*, os caracteres são colocados um embaixo do outro, se é *FALSE*, são colocados um cima do outro. Se *leftToRight* é *TRUE*, *strings* são colocados lado a lado da esquerda para a direita, se é *FALSE*, são colocados da direita para a esquerda. Os valores *default* para estes campos são *TRUE*.
- *justify*: determina a maneira na qual o bloco de texto é posicionado em relação aos eixos X e Y (centralizado, à direita ou à esquerda dos eixos). O valor do campo, que consiste numa lista de uma ou duas seleções, pode ser: *FIRST*, *BEGIN* (*default*), *MIDDLE* e *END*.
- *language*: valores para este campo baseiam-se em especificações locais esboçadas em vários padrões internacionais. Cada valor de campo contém o *language code* requerido seguido por um *underscore* opcional e um *territory code*. Exemplos de valores válidos são: *en* (Inglês), *en_US* (Inglês - Estados Unidos), *de* (Alemão) e *jp* (Japonês).

O exemplo a seguir é apresentado ilustra a utilização dos nós descritos (figura 4.2).

```
#VRML V2.0 utf8
# Exemplo de utilização do nó Text

Group {
   children [
          Shape {
                    appearance DEF White Appearance {
                              material Material { }
                    }
                    geometry Text {
                              string "First"
                              fontStyle FontStyle {
                                        family "SERIF"
                                        style "ITALIC"
                                        justify "END"
                                        size 1.0
                              }
                    }
          },
```

```
        Shape {
                appearance USE White
                geometry Text {
                        string "Second"
                        fontStyle FontStyle {
                                family "SANS"
                                style "BOLD"
                                justify "BEGIN"
                                size 1.0
                        }
                }
        }
   ]
}
```

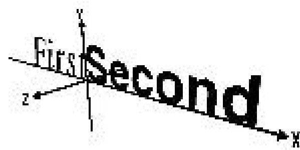

Figura 4.2 – Dois nós *Text* com diferentes formatações

5. Alterando o Aspecto de Shapes

Os *shapes* descritos até agora podem ser combinados e alterados para formar cenas mais complexas, como por exemplo a criação de uma cidade. Assim, neste capítulo é apresentado o nó de agrupamento (*group node*) *Transform*, que cria um sistema de coordenadas local para os nós filhos. A ordem de uma série de transformações definidas dentro de um mesmo nó *Transform* é fixa: mudança de escala em torno de um ponto central, rotação em torno de um ponto central, e translação relativa ao sistema de coordenadas pai. Para mudar a ordem das transformações, é necessário aninhar nós *Transform*. Nas próximas seções estas transformações são detalhadamente descritas.

5.1. Posicionando Shapes

Shapes podem ser posicionados em qualquer lugar do "mundo" VRML usando o nó de agrupamento *Transform* e seu campo *translation*. Como já comentado, o "mundo" VRML é constituído por um sistema de coordenadas 3D. Para

exemplificar, uma coordenada (3.0, 2.0, 5.0) está 3.0 unidades ao longo do eixo X, 2.0 unidades ao longo do eixo Y e 5.0 unidades ao longo do eixo Z. O nó S*hape*, assim como o *Text*, é construído na origem deste sistema de referência.
Em VRML é possível criar qualquer número de sistemas de coordenadas, sendo que cada um é posicionado, ou transladado, em relação à origem de outro sistema de coordenadas. Quando um sistema de coordenadas é relativo a outro, diz-se que o novo sistema de coordenadas é um "filho" (*child coordinate system*) que é aninhado (*nested*) dentro do sistema de coordenadas "pai" (*parent coordinate system*), que, por sua vez, também pode estar aninhado em outro sistema de coordenadas, e assim por diante. O pai mais externo é chamado de raiz (*root coordinate system*).
Nos exemplos apresentados até aqui os *shapes* eram centralizados na origem do sistema de coordenadas raiz. O nó *transform* cria novos sistemas de coordenadas relativos ao sistema raiz ou a qualquer outro. O sistema raiz, a partir do qual os outros sistemas dependem direta ou indiretamente, e ao qual o "mundo" VRML está associado, é freqüentemente chamado de sistema de coordenadas do mundo (*world coordinate system*). Todos os sistemas de coordenadas, incluindo *shapes* que são construídos nestes sistemas de coordenadas, são chamados de cena gráfica. Para ilustrar, a figura 5.1 mostra um sistema de coordenadas do universo representado pelos eixos X, Y e Z, e um sistema de coordenadas filho transladado -3.0 unidades ao longo do eixo X, 2.0 unidades ao longo do eixo Y e 2.0 unidades ao longo do eixo Z.

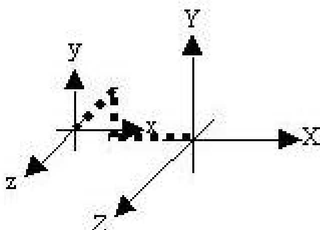

Figura 5.1 – Sistema de coordenadas do mundo e um filho

Cada vez que se cria um novo sistema de coordenadas com o nó *Transform*, especifica-se sua posição, ou translação, relativa ao sistema de coordenadas pai. A distância de translação é dada em unidades VRML nas direções de X, Y e Z e determina a localização de qualquer *shape* criado neste novo sistema de coordenadas. Torna-se importante salientar que, por *default*, o observador inicialmente está na posição (0, 0, 10) do sistema de coordenadas do mundo, olhando na direção do eixo Z negativo.
O nó *Transform* é um nó de agrupamento semelhante a *Group*, que cria um novo sistema de coordenadas relativo (transladado) ao sistema de coordenadas pai. Ele contém um lista de nós *child*, que podem ser nós *Shape*, *Group* ou *Transform*. *Shapes* criados como *children* do nó *Transform* são construídos em relação a nova origem do sistema de coordenadas. A sintaxe do nó *Transform* é a seguinte:

```
Transform{
        children    [ ]                 # MFNode
        translation        0.0 0.0 0.0          # SFVec3f
```

```
        rotation    0.0 0.0 1.0 0.0        # SFRotation
        scale                  1.0 1.0 1.0            # SFVec3f
        scaleOrientation 0.0 0.0 1.0 0.0            # SFRotation
        bboxCenter           0.0 0.0 0.0            # SFVec3f
        bboxSize  -1.0 -1.0 -1.0        # SFVec3f
        center                 0.0 0.0 0.0            # SFVec3f
        addChildren                                   # MFNode
        removeChildren                                # MFNode
}
```

O valor do campo *children* especifica uma lista de nós *child* que serão incluídos no grupo, tipicamente *Shape*, *Group* ou *Transform*. O valor *default* para este campo é uma lista vazia. Os valores de *translation* especificam as distâncias nas direções X, Y e Z, respectivamente, entre a origem do sistema de coordenadas pai e do filho. O valor *default* 0.0 não acarreta alteração de posição. Os outros campos deste nó serão descritos posteriormente. Um exemplo da utilização deste nó é apresentado abaixo e ilustrado na figura 5.2.

```
#VRML V2.0 utf8
# Exemplo de utilização do nó Transform

Transform {
   translation 2.0 1.0 -2.0
   children [
      Shape {
         appearance Appearance {
            material Material { }
         }
         geometry Cylinder { }
      }
   ]
}
```

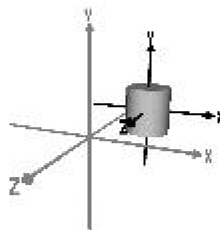

Figura 5.2 – Translação de um cilindro

5.2. Rotação de Shapes

Através dos campos *rotation* e *center* do nó *Transform*, torna-se possível rotacionar *shapes* e grupos de *shapes* sobre a origem do sistema de coordenadas ou sobre um ponto central especificado. Desta forma é possível, por exemplo, posicionar um cone com a ponta virada para baixo para formar um funil, e girar um cilindro para fazer a roda de um carro.

Um eixo de rotação é uma linha imaginária sobre a qual um sistema de coordenadas é rotacionado, que pode apontar para qualquer direção. Por exemplo, o eixo de rotação usado para girar o pneu de um carro aponta horizontalmente

para fora do centro do pneu. Quando se rotaciona um sistema de coordenadas em VRML, está se especificando um eixo de rotação e a direção para este eixo de rotação. Para se especificar a direção para um eixo de rotação, imagine o desenho de uma linha entre duas coordenadas no espaço. Uma coordenada está sempre na origem (0.0, 0.0, 0.0) e a outra está localizada em uma outra posição. A linha imaginária desenhada entre estas duas coordenadas cria um eixo de rotação. A distância entre as duas coordenadas não faz diferença, isto é, qualquer ponto na linha imaginária é válido. Por exemplo, para definir um eixo de rotação que aponta ao longo do eixo Y, (0.0, 2.0, 0.0), (0.0, 0.357, 0.0) ou (0.0, 1.4, 0.0) são todos equivalentes.
Além de se especificar um eixo de rotação, também deve-se indicar o quanto o novo sistema de coordenadas deve ser rotacionado em torno do eixo. A quantidade de rotação é especificada como um ângulo de rotação, medido em radianos. Os ângulos de rotação podem positivos ou negativos, fornecendo a opção de se rotacionar um objeto para a esquerda ou para direita.
Para entender melhor o resultado da rotação de acordo com os valores fornecidos, utiliza-se a regra da mão direita. Neste caso, imagina-se que uma pessoa "agarrando" um eixo com a mão direita, dobrando os dedos ao redor do eixo e eixo e apontando o polegar na direção positiva do eixo. Um ângulo de rotação positivo irá rotacionar um sistema de coordenadas em torno do eixo na mesma direção dos dedos dobrados, e um ângulo de rotação negativo rotaciona na direção oposta.
Através do campo *rotation* do nó *Transform*, apresentado na seção anterior, é possível especificar um eixo de rotação e a quantidade de rotação em torno deste eixo. Os primeiros três valores especificam a coordenada 3D (X, Y e Z) usada para criar a linha imaginária que especifica o eixo de rotação. O quarto valor especifica o ângulo de rotação em radianos.
O valor para o campo *center* especifica a coordenada 3D em um novo sistema de coordenadas transladado sobre o qual as rotações vão ocorrer. O centro de rotação *default*, uma vez que os *shapes* são construídos na origem, é a origem. Entretanto, quando se está construindo *shapes* articulados, como um robô, é mais natural usar como centro de rotação a ponta de um braço ao invés do seu centro. Utilizando, então, o campo *center*, pode-se especificar a localização do centro de rotação. O próximo exemplo mostra a base e o braço de uma lâmpada de mesa ajustável. Ele simula a existência de uma "junta" no meio da base ao qual a parte inferior do braço está conectada (figura 5.3).

```
#VRML V2.0 utf8

Group{
   children [
   # Base da lâmpada
      Shape {
         appearance DEF White Appearance {
            material Material { }
         }
         geometry Cylinder {
```

```
            radius 0.1
            height 0.01
         }
      },

      # Junta da base
      Transform {
         translation 0.0 0.15 0.0
         rotation    1.0 0.0 0.0  -0.7
         center      0.0 -0.15 0.0
         children [
            # Braço
            Shape {
               appearance USE White
               geometry Cylinder {
                  radius 0.01
                  height 0.3
               }
            }
         ]
      }
   ]
}
```

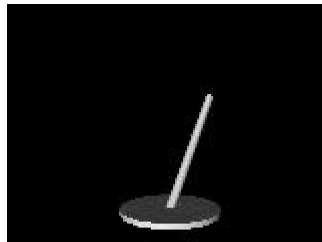

Figura 5.3 – Braço de uma lâmpada de mesa

5.3. Escala de Shapes

Translação, rotação e escala podem ser combinadas no mesmo nó *Transform*. *Shapes* podem ter qualquer tamanho em VRML e podem, então, ser combinados para criar novos objetos. Usando os campos *scale* e *scaleOrientation* do nó *Transform*, pode-se alterar a escala de *shapes* e grupos de *shapes* para qualquer tamanho. Assim, o sistema de coordenadas pode ter seu tamanho aumentado ou diminuído em relação ao sistema de coordenadas pai. Em outras palavras, o tamanho das unidades no mundo VRML é trocado e, por exemplo, as distâncias das translações posteriores são maiores ou menores dependendo do fator de escala. *Shapes* criados em um sistema de coordenadas onde foi aplicada uma escala, são construídos neste novo sistema de coordenadas.
A diferença de tamanho quando se aumenta ou diminui um objeto é chamada de fator de escala e corresponde a um fator de multiplicação. Por exemplo, para construir um carro com a metade de seu tamanho original, utiliza-se um fator de

escala 0.5, e com o dobro do seu tamanho, utiliza-se um fator de escala 2.0. O campo *scale* usa três fatores de escala, um valor para escala na direção X, um para a direção Y e outro para direção Z. Para aumentar e diminuir a escala de um sistema de coordenadas sem deformá-lo, deve-se especificar o mesmo fator de escala nas direções X, Y e Z.
Como já comentado. os fatores de escala X, Y e Z do nó *Transform* permitem aumentar ou diminuir o tamanho do sistema de coordenadas diferentemente nas posições X, Y e Z. Entretanto, pode haver situações onde é necessário "esticar" um *shape* ao longo de uma posição arbitrária. Este efeito pode ser alcançado em VRML através da orientação da escala. Em outras palavras, somente a direção da escala é alterada, e o sistema de coordenadas como um todo não é orientado novamente.
Para orientar uma escala, deve-se utilizar o campo *scaleOrientation* e fornecer um eixo de rotação e um ângulo, como no campo *rotation*. A rotação da orientação da escala é aplicada ao novo sistema de coordenadas antes da escala, o que permite "esticar" o sistema de coordenadas em qualquer direção sem deixá-lo rotacionado.
Por *default*, o centro da escala (*scale center*) é a origem do sistema de coordenadas. Porém, é possível especificar-se um centro de escala em qualquer lugar do sistema de coordenadas usando o campo *center*. Assim, o fator de escala utilizado faz com que a escala seja relativa a este ponto ao invés da origem.
Torna-se importante comentar que os valores para o campo *center* são também usados como centro da rotação, permitindo que se faça uma rotação e uma escala sobre o mesmo ponto ao mesmo tempo.
Os valores *default* para o campo *scale* do nó *Transform* são 1.0 para X, Y e Z. Fatores de escala podem ter valores positivos ou negativos. Valores entre 0.0 e 1.0 reduzem o tamanho do novo sistema de coordenadas, e valores maior do que 1.0 aumentam. Os valores do campo *scaleOrientation* correspondem aos valores X, Y e Z da coordenada 3D e ao ângulo em radianos, determinando assim o eixo de rotação e o ângulo.
O exemplo apresentado a seguir, e ilustrado na figura 5.4, mostra a maneira na qual os campos *scale*, *scaleOrientation* e *center* podem ser usados.

```
#VRML V2.0 utf8

Group {
   children [
   # Chão
      Shape {
         appearance DEF White Appearance {
            material Material { }
         }
         geometry Box {
            size 12.0 0.1 12.0
         }
      },

   # Árvore
      Transform {
```

```
            translation 0.0 1.0 0.0
            scale       1.0 2.0 1.0
            scaleOrientation 0.0 0.0 1.0 -0.785
            center      0.0 -1.0 0.0
            children [

            # Tronco
                Shape {
                    appearance USE White
                    geometry Cylinder {
                        radius 0.5
                        height 2.0
                    }
                },

            # Galhos
                Transform {
                    translation 0.0 3.0 0.0
                    children Shape {
                        appearance USE White
                        geometry Cone {
                            bottomRadius 2.0
                            height 4.0
                        }
                    }
                }
            ]
        }
    ]
}
```

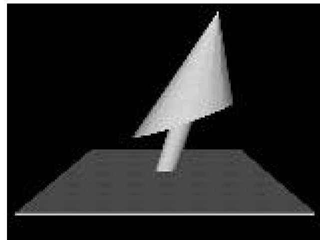

Figura 5.4 – Árvore com escala ao longo do eixo diagonal usando orientação de escala

6. Controlando a Aparência de Shapes

É possível controlar a aparência de qualquer *shape* especificando-se atributos do material a partir do qual ele é feito. Atributos do material incluem cor, quando ele brilha e qual é a cor do brilho, quando e o quanto ele é transparente, etc. Usando os nós *Appearance* e *Material* é possível controlar estes atributos. Como já visto, *shapes* VRML são definidos através de dois campos, geometria (*geometry*) e aparência (*appearance*), onde o valor utilizado é o nó *Appearance*. Similarmente,

os atributos deste nó são separados nos nós *Material* e de textura. A sintaxe destes nós, cujos campos são descritos neste capítulo, está apresentada a seguir.

```
Appearance {
	material		NULL	# SFNode
	texture			NULL	# SFNode
	textureTransform	NULL	# SFNode
}
Material {
	ambientIntensity	0.2		# SFFloat
	diffuseColor		0.8 0.8 0.8	# SFColor
	emissiveColor		0.0 0.0 0.0	# SFColor
	shininess		0.2		# SFFloat
	specularColor		0.0 0.0 0.0	# SFColor
	transparency		0.0		# SFFloat
}
```

As cores em VRML são descritas de uma maneira mais precisa, combinando porções de vermelho, verde e azul (ou RGB). Cores RGB contêm três valores de ponto flutuante que variam de 0.0 (não possui a cor) a 1.0 (cor máxima). A cor refletida pelo objeto é especificada através do campo *diffuseColor*. Já o campo *emissiveColor*, por sua vez, possibilita a obtenção de efeitos de brilho, pois através dele é possível especificar-se a cor emitida pelo objeto. É importante salientar que através deste campo não é possível iluminar *shapes* próximas, ele só indica que o *shape* é mais brilhante do que o usual.

O valor do campo *transparency* determina um fator de transparência entre 0.0 (opaco) e 1.0 (transparente), e *ambientIntensity* determina como o material é afetado pelo nível de luz ambiente do "mundo" VRML. Quanto mais baixo for o valor, menor é o efeito da luz sobre o material. *specularColor* especifica a cor da luz refletida pelo *shape*, isto é, a cor do brilho. O valor *default* para este campo é preto, que desabilita a reflexão especular. Finalmente, *shininess* controla a intensidade do brilho, sendo que 0.0 torna-o mais escuro. O efeito visual para este campo consiste na redução do tamanho da reflexão especular. É importante salientar que os navegadores VRML incluem automaticamente uma *headlight* que move-se de acordo com a posição do observador.

A textura na verdade nada mais é do que uma imagem 2D mapeada em um objeto. O valor do campo *texture* do nó *Material* consiste num nó que especifica a imagem da textura a ser aplicada a um *shape*. Tal nó pode ser *ImageTexture*, *PixelTexture* ou *MovieTexture*.

```
ImageTexture {
	url		[ ]	# MFString
	repeatS		TRUE	# SFBool
	repeatT		TRUE	# SFBool
}
```

O campo *url* especifica uma lista de URLs ordenadas por prioridade. O navegador tenta abrir a primeira URL especificada na lista, se não encontrar, ele pega a próxima e assim por diante. Quando uma URL pode ser aberta, o arquivo é lido e mapeado como textura do *shape*. O arquivo que contém a textura pode ter formato JPEG ou PNG. Alguns navegadores também aceitam o formato GIF. Os

campos *repeatS* e *repeatT* podem ter os valores *TRUE* (a textura é repetida dentro do sistema de coordenadas da textura) ou *FALSE* (textura não é repetida).

```
PixelTexture {
        image           0 0 0     # SFImage
        repeatS         TRUE      # SFBool
        repeatT         TRUE      # SFBool
}
```

Este nó especifica uma imagem, ou atributos de mapeamento de textura, no próprio arquivo VRML. O valor para o campo *image* determina o tamanho da imagem e valores de *pixel* para uma textura de imagem. Os primeiros três valores inteiros são a largura da imagem em pixels, a altura da imagem também é pixels e o número de bytes para cada pixel. Os campos *repeatS* e *repeatT* são iguais ao nó anterior.

```
MovieTexture {
        url             [ ]       # MFString
        loop            FALSE     # SFBool
        speed           1.0       # SFFloat
        startTime  0.0  # SFTime
        stopTime  0.0   # SFTime
        repeatS         TRUE      # SFBool
        repeatT         TRUE      # SFBool
        isActive        # SFBool
        duration_changed    # SFBool
}
```

O nó *MovieTexture* permite mapear uma textura animada para um objeto. Os formatos suportados são: MPEG1-Systems e MPEG1-Video. Além de campos para especificar a textura a ser mapeada, o nó possui campos que permitem controlar a exibição do filme: velocidade, horas de início, hora de fim e indicador de laço da exibição.

7. Adicionando Links e Arquivos

Talvez uma das características mais interessante de VRML é a sua habilidade de *linkar* "mundos" VRML no WWW. Os *links* em VRML são ligações, ou âncoras, feitas em figuras específicas da cena. Qualquer figura descrita através de nós VRML pode ser uma âncora. Também pode-se usar *links* para unir cenas ou figuras que estão em arquivos diferentes em um único arquivo VRML. Esta técnica, conhecida como *inlining*, permite construir, por exemplo, uma sala usando as paredes de um arquivo VRML e a mobília de vários outros.
O nó de agrupamento *Anchor* engloba um grupo de *shapes* e cria um *hyperlink* com outras mídias (página HTML, imagem, etc.), isto é, quando o usuário clica nestes *shapes* o navegador visualiza um novo arquivo. A sintaxe deste nó é a seguinte:

```
Anchor {
        children    [ ]           # MFNode
```

```
        bboxCenter          0.0 0.0 0.0         # SFVec3f
        bboxSize   -1.0 -1.0 -1.0        # SFVec3f
        url                 [ ]                 # MFString
        parameter [ ]                    # MFString
        description         ""                  # SFString
        addChildren                             # MFNode
        removeChildren                          # MFNode
}
```

Através dos campos deste nó pode-se especificar: um grupo *children*; o tamanho do envelope (*bounding box*) que contém todos os *shapes* do grupo; uma URL (*Universal Resource Locator*) que determina um endereço ou um arquivo na *Web*; e uma descrição (*description*).

Um objeto, ou um conjunto de objetos, pode ser criado em um único arquivo VRML. Porém, unindo-se vários objetos, para criar objetos ou cenas mais complexas, o arquivo VRML pode se tornar muito grande e difícil de gerenciar. O nó *Inline* consiste numa técnica de construção de cenas que permite manter cada uma das pequenas partes da cena em um arquivo diferente. Para construir uma cena que utiliza estes arquivos, deve-se criar um arquivo VRML que lista todos estes arquivos. Isto é feito através do nó *Inline*, cujos os nomes dos arquivos especificados são nomes de arquivos locais que serão incluídos na cena.

A sintaxe do nó *Inline* é:

```
Inline {
        url                 [ ]                 # MFString
        bboxCenter          0.0 0.0 0.0         # SFVec3f
        bboxSize   -1.0 -1.0 -1.0        # SFVec3f
```

Bibliografia

[AME 97] AMES, Andrea L.; NADEAU, David R.; MORELAND, John L. **VRML 2.0 Sourcebook**. 2nd ed. New York: John Wiley, 1997. 654 p.