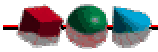
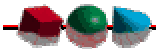
Curso Básico

Profa. Dra. Elisamara de Oliveira

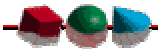


Índice



“Estude sempre. A renovação das ideias favorece a evolução do espírito”.

FCX



1. Introdução ao VRML

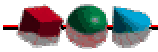
VRML é a abreviação de ***Virtual Reality Modeling Language***, ou Linguagem para Modelagem em Realidade Virtual. É uma linguagem independente de plataforma que permite a criação de cenários tridimensionais (3D) por onde se pode passear, visualizar objetos por ângulos diferentes e até interagir com eles. É uma linguagem textual para descrição de cenas e ambientes interativos em 3D; não é uma linguagem de programação.

Arquivos VRML têm extensão **.wrl** e não são compilados; são arquivos-texto escritos em ASCII que podem ser interpretados por um interpretador VRML. Um interpretador ou *browser* VRML permite a visualização de um programa VRML localmente ou através da internet. A primeira versão da linguagem não possibilitava muita iteração do usuário com o mundo virtual, mas versões recentes acrescentam características como animação, movimentos de corpos e iteração entre usuários. A última versão é a 2.0, chamada ***Moving Worlds VRML 2.0***. A "Especificação VRML" é a documentação que descreve todas as características da linguagem.

A linguagem VRML pode ser usada numa página *web* preenchendo a página, preenchendo apenas um retângulo da página ou preenchendo um *frame* ou parte do *frame.*

Apresentada pela primeira vez em 1994 na Primeira Conferência da *World Wide Web*, a linguagem tem como objetivo dar o suporte necessário para o desenvolvimento de mundos virtuais multi-usuários na internet, sem precisar de redes de alta velocidade. O código VRML é um subconjunto do formato de arquivo ASCII do Open Inventor, da Silicon Graphics, com características adicionais para navegação na *web*. Esta característica é equivalente aos *links* do HTML, ou seja, podem-se criar *links* em um mundo virtual que levem a outros mundos virtuais.

A linguagem trabalha com geometria 3D (possui as formas geométricas primitivas cubo, cone, cilindro e esfera e diversas outras geometrias avançadas) e suporta transformações (rotação, translação, escala), texturas, luz, sombreamento e animação.



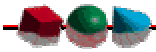
1.1. Histórico

No final da década de 1980, Tim Berners-Lee criou a *World Wide Web*, adicionando inovações à internet em termos de conectividade e de interface. Em maio de 1994, em Genebra, na 1ª Conferência da *WWW*, o grupo de discussão de Realidade Virtual decidiu desenvolver uma linguagem de descrição de cena que pudesse ser usada na *web.* Em maio de 1995, completou-se a especificação da **VRML 1.0** e em janeiro de 1996 foi lançada a versão **1.0c.** Em 1995, foi formado o *VRML Architecture Group* (VAG). Foi lançado um *call for proposals* (chamada para apresentação de propostas) para uma redefinição e extensão da linguagem para suportar animação e interação. A Silicon Graphics (SGI), Netscape e outras companhias criaram a **Moving Worlds**. Em março de 1996, o VAG decidiu por larga maioria adotar esta proposta como ponto de partida para o **VRML 2.0.** Em 1997 reescreveu-se a especificação para submissão à *International Standards Organization* (ISO), criando-se o **VRML 97**. O primeiro *browser* aderente a esta especificação foi o *Cortona* da SGI.

Os *browsers* para VRML 1.0 não permitem a exibição de VRML 2.0. A maioria dos *browsers* para VRML 2.0 também exibem VRML 1.0, e na maioria dos casos são capazes também de exibir VRML 97. Para navegar em mundos virtuais criados com a linguagem será necessário usar *plug-ins* que permitirão aos *browsers* suportarem VRML. Assim, ao invés de visitar *homepages*, o usuário visitará *homeworlds*. Na verdade, existem muitos *browsers* disponíveis que suportam diretamente a linguagem; os *browsers* tradicionais necessitam de software adicional (*plug-in*).

1.2. Aplicações do VRML

O VRML tem obtido crescente aceitação como tecnologia padrão da *web* para exibição de conteúdo gráfico 3D. O VRML tem se tornado um meio rico de expressão de idéias na *web*, pois o mundo VRML é interativo e pode conter animação, som e filmes. A linguagem é resultado de um processo de discussão e cooperação aberto, sintetizando o conhecimento e experiência de milhares de pessoas de modo simples e acessível. O VRML é o primeiro passo a caminho da *web* 3D, imersiva e interativa.

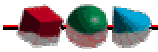


Apesar do VRML ser uma tecnologia relativamente nova, apresenta um enorme potencial em áreas como:

- ✓ Ciência (Medicina, Geociências, Engenharia, Arquitetura)
- ✓ Entretenimento (jogos, animação)
- ✓ Negócios (publicidade, manuais de produtos)
- ✓ Artes
- ✓ Ensino

O principal problema atual para aplicações na internet é a largura de banda. Quando mais usuários tiverem acesso às tecnologias de banda larga, mais e mais aplicações VRML serão utilizadas. A figura 1 traz um exemplo de um mundo virtual VRML.

Figura 1 – Exemplo de homeworld



1.3. Como começar a navegar em mundos VRML na internet

Antes de começar a navegar em *homeworlds* e mesmo programar em VRML é necessário fazer o *download* de um *browser* ou um *plug-in* VRML para que os sites e os programas possam ser visualizados. Como a maioria das pessoas possui os *browsers* Internet Explorer e Netscape, o melhor é utilizar um *plug-in*. O *plug-in* mais utilizado é o Cortona que se encontra no endereço:

http://www.cortona3d.com/Products/Cortona-3D-Viewer/install.aspx

Estando no site, faça o *download* do *plug-in* de acordo com o sistema operacional da sua máquina. Execute um programa e o Cortona se auto-instalará na sua máquina. Daí para a frente o Cortona será acionado automaticamente toda vez que uma página VRML ou arquivo **.wrl** for ativado.

Agora sua máquina está pronta para desvendar os *homeworlds*. Conecte-se à internet e faça uma busca por arquivos **.wrl**. Um excelente programa de busca é o google:

http://www.google.com.br

Esta busca foi feita e alguns resultados interessantes são listados a seguir. Caso algum desses *links* não mais exista, tente outros. Tudo muda muito rapidamente na www.

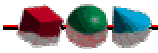
➔ Lista de sites em VRML

http://www.geom.umn.edu/~daeron/bin/legitlist.cgi

➔ Mundos VRML

http://www.amazing3d.com/free/free.html

http://www.frontiernet.net/~imaging/vrml_avatar.html



➔ Tutoriais/Cursos VRML

http://sim.di.uminho.pt/vrmltut/toc.html

http://mirror.impa.br/sibgrapi97/cursos/vrml/cap01/toc.htm

(em português)

http://www.lighthouse3d.com/vrml/tutorial/index.shtml?intro

(em inglês)

➔ Exemplos do livro *VRML 2.0 Sourcebook*

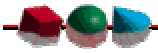
http://www.wiley.com/compbooks

➔ Especificações VRML

http://www.web3d.org/vrml/vrml.htm

http://vag.vrml.org

http://vrml.sgi.com/moving-worlds/spec



2. A Linguagem VRML

Ficou definido, para fins de identificação, que todo arquivo VRML, na versão 2, tem que ter o cabeçalho:

```
#VRML V2.0 utf8
```

O caracter '#' também significa **comentário**. Toda linha que começa com # será ignorada pelo *browser*. É bom usar comentários no meio do código, isso facilita a compreensão e identificação de partes do cenário, como por exemplo:

```
#Uma esfera
```

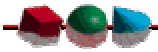
2.1. Primeiro programa em VRML

Um arquivo VRML é uma descrição textual do mundo VRML. Arquivos VRML têm extensão ***.wrl***. Um arquivo VRML é composto de:

- ✓ *header*
- ✓ comentários
- ✓ nós (*nodes*), contendo campos (*fields*) e valores (*values*)
- ✓ rotas e protótipos

```
#VRML V2.0 utf8
#esfera.wrl
#Uma esfera

Shape
{
      appearance Appearance
      { material Material
                  { diffuseColor 1 1 0
                  }
      }
      geometry Sphere
                  { radius 1.5
                  }
}
```



Tudo o que se precisa para escrever um código VRML é um editor de textos (por exemplo, o Bloco de Notas). Uma vez editado, o arquivo deve ser gravado em formato ASCII e ter a extensão **.wrl**.

Edite o programa descrito anteriormente. Salve o arquivo como **esfera.wrl**. Entre no seu *browser* e abra o arquivo editado (esfera.wrl). Não é preciso estar conectado à internet, pois o seu arquivo será exibido localmente. Nem é preciso fechar o arquivo no seu editor de textos. O Cortona entrará em ação e mostrará uma esfera amarela. Aproveite para se acostumar com os controles da barra inferior do Cortona, que permite a livre manipulação dos objetos no espaço tridimensional.

Caso não apareça a esfera, haverá luzes vermelhas piscando na barra inferior. Isso significa que há erro no arquivo. Clicando nas luzes, aparecerão as mensagens de erro. Volte ao editor, conserte os erros, salve o arquivo e volte ao *browser* novamente. Cuidado, pois o **VRML faz diferença entre letras maiúsculas e minúsculas**! Tudo muito simples e fácil, como mostra a figura 2.

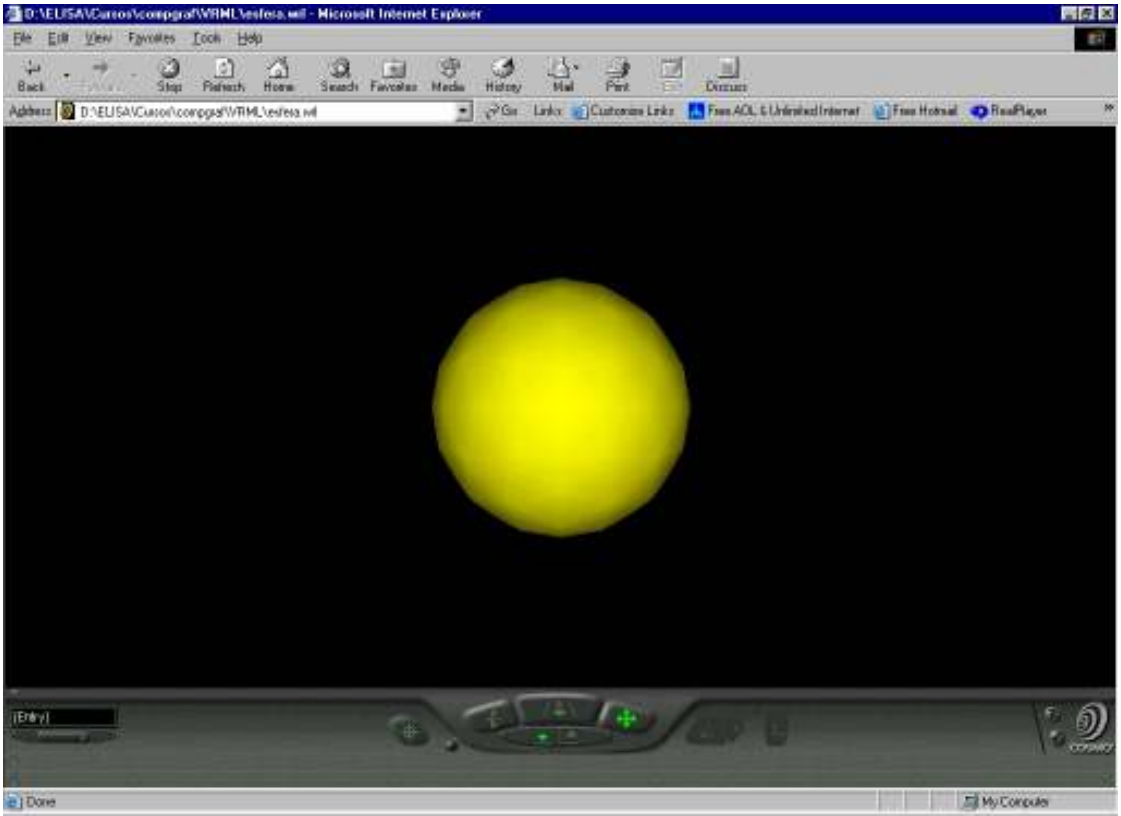
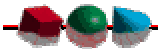


Figura 2 – Visualização do programa Esfera.wrl no Cortona



2.2. Cores em VRML

As cores em VRML são definidas pelo sistema **RGB** - **R**ed (vermelho) **G**reen (verde) **B**lue (azul). Utiliza-se um valor numérico associado a cada uma dessas cores para se obter uma gama maior delas, a partir de sua mistura. Veja a tabela 1. A tabela 1 traz 8 possibilidades de cores considerando a utilização de zeros e uns para se definir a presença (1) ou ausência (0) da cor na mistura.

Tabela 1 – Cores básicas em VRML no sistema RGB

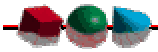
R (Vermelho)	**G** (Verde)	**B** (azul)	**Cor Resultante**
0	**0**	**0**	**preto**
0	**0**	**1**	**azul**
0	**1**	**0**	**verde**
0	**1**	**1**	**ciano**
1	**0**	**0**	**vermelho**
1	**0**	**1**	**roxo**
1	**1**	**0**	**amarelo**
1	**1**	**1**	**branco**

No programa apresentado na seção 2.1, a cor da esfera foi assim definida:

```
appearance Appearance
{
      material Material
            { diffuseColor 1 1 0
            }
}
```

ou seja, a mistura RGB resultou no amarelo. Mas, não existem apenas as 8 cores mostradas na tabela 1. **Os valores associados aos componentes RGB podem ser quaisquer valores entre 0 e 1! Isso dá uma mistura praticamente infinita de cores.**

Quando nenhuma cor é definida, ou seja, quando o campo **diffuseColor** é omitido, a cor assumida é o branco.



Assim, poder-se-ia definir uma cor como:

```
appearance Appearance
{
      material Material
            { diffuseColor 0.57 0.25 0.89
            }
}
```

Que cor foi criada? Modifique o programa **esfera.wrl** e veja o resultado. Faça outros testes também. Não há limite para a criação de cores.

2.3. Unidades de medida e sistema de coordenadas

O VRML usa o sistema cartesiano 3D. A seqüência dos eixos é X,Y,Z; a unidade de medida para distâncias é **metros** e para ângulos **radianos**.

O sistema cartesiano 3D define o espaço com um sistema de 3 eixos:

- **Eixo X** (análogo à largura)
- **Eixo Y** (análogo à altura)
- **Eixo Z** (análogo à profundidade)

Assumindo que o cruzamento dos 3 eixos é o ponto central (0,0,0), a descrição dos objetos no espaço 3D pode ser expressa através de coordenadas relativas ao ponto central. Cada eixo tem uma direção positiva e negativa, estendendo-se do ponto central da cena.

Tomando esta página como referência, o eixo-X positivo está para a direita, o eixo-Y positivo está para cima e o eixo-Z positivo está perpendicular aos dois anteriores, saindo da página na direção do leitor, conforme mostra a figura 3.

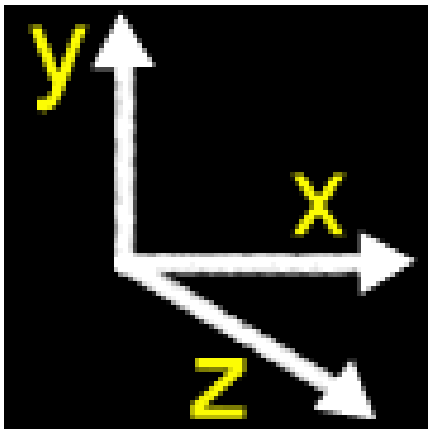


Figura 3 – sistema de coordenadas 3D

Os objetos dentro da cena (e a própria cena) podem ser rotacionados mudando a sua orientação de 3 formas (conforme mostra a figura 4):

- **Yaw**, rotação em torno do eixo Y
- **Pitch**, rotação em torno do eixo X
- **Roll**, rotação em torno do eixo Z

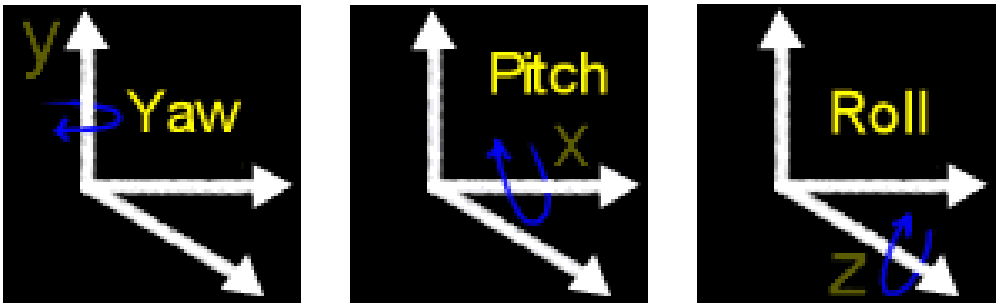


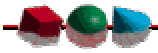
Figura 4 – três formas de rotação dos objetos e das cenas

Os 3 eixos e as 3 rotações são chamados de 6 graus de liberdade, que determinam a localização e orientação dos objetos no espaço 3D.

2.4. Representação de objetos 3D

Os objetos 3D podem ser representados de 4 formas distintas, conforme mostra a figura 5:

Figura 5 – representação de objetos 3 D



3. Shape- Formas Geométricas 3D Primitivas

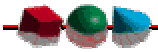
Basicamente, pode-se dizer que um **node** é um conjunto de especificações que determinam as características dos objetos contidos no cenário. Os nodes definem a hierarquia e as características individuais de cada objeto dentro do contexto geral.

O node descreve o tipo do objeto, que pode ser uma esfera, um cilindro, uma transformação, uma definição de luz ou textura, etc. Também define as características de cada um, como altura de um cone, diâmetro de uma esfera, intensidade da luz ambiente, cor, etc.

Para se definir qualquer forma geométrica 3D em VRML, seja ela básica ou avançada, utiliza-se o node **Shape**.

Os objetos tridimensionais em VRML são chamados de Shapes. Um Shape possui, em geral dois atributos, a **aparência** e a **geometria**. A **aparência** define a cor do objeto, sua textura, dentre outros atributos. A **geometria** define qual objeto ou forma 3D deve ser exibida. O formato de um arquivo VRML que utiliza apenas um node para definir uma forma geométrica é:

```
#VRML V2.0 utf8
#Nome do arquivo
#Comentários
Shape
{   appearance Appearance # define a aparência
                {
                  ...
                }
    geometry   □   # define a geometria ou forma 3D
                  {
                    ....
                  }
}
```



Neste curso será adotado como padrão colocar o nome do arquivo na segunda linha dos programas apresentados, na forma de comentário. Assim o aluno poderá localizar o programa (que se encontra no disquete que acompanha o curso) e visualizá-lo no Cortona.

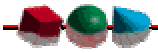
As formas geométricas que podem ser definidas no campo **`geometry`** são:

`Box`
`Cone`
`Cylinder`
`Sphere`
`ElevationGrid`
`Extrusion`
`IndexedFaceSet`
`IndexedLineSet`
`PointSet`
`Text`

As quatro primeiras são formas geométricas primitivas: **box** (cubo), **cone** (cone), **cylinder** (cilindro) e **sphere** (esfera) e serão definidas no que se segue. As outras são consideradas formas de geometria avançada e serão tratadas no capítulo 5.

3.1. Box

Para definir um cubo (ou um paralelepípedo) usa-se a geometria **Box** do node **Shape**. O centro padrão de um Box é (0,0,0). A geometria tem apenas um campo **size** composto de 3 valores em ponto flutuante que correspondem às dimensões do cubo nas direções X (largura), Y (altura) e Z (profundidade).



Sintaxe:

```
geometry Box { size □ □ □
               }
```

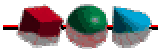
Exemplo:

```
#VRML V2.0 utf8
#Programa Box.wrl
#Um paralelepípedo azul

Shape
{   appearance Appearance
    {   material Material
                { diffuseColor 0.0 0.0 1.0
                }
    }
    geometry Box {   size 2.5 2.5 5.0
                  }
}
```

3.2. Cone

Para definir um cone (chapéu de palhaço) usa-se a geometria **Cone** do node **Shape**. A geometria tem os campos **bottomRadius** e **height** que definem o raio da base e a altura, e dois campos opcionais **side** e **bottom** que, se omitidos, são assinalados para TRUE.



Sintaxe:

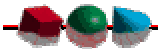
```
geometry Cone { bottomRadius □      #raio da base
                height □            #altura
                side TRUE/FALSE     #tem a lateral ?
                bottom TRUE/FALSE   #tem a base ou é vazado?
              }
```

Exemplo:

```
#VRML V2.0 utf8
#Programa Cone.wrl
#Um cone vermelho com lateral e base

Shape
{  appearance Appearance
   {  material Material
              { diffuseColor 1 0 0
              }
   }
   geometry Cone {  bottomRadius 3
                    height 5
                    side TRUE
                    bottom TRUE
                 }
}
```

Seria um exercício interessante modificar os atributos **side** e **bottom** de TRUE para FALSE para se entender melhor a sua função. O campo bottom FALSE permite que se construa um “chapéu de palhaço” vazado no centro.



3.3. Cylinder

Para definir um cilindro (canudo de refrigerante) usa-se a geometria **Cylinder** do node **Shape**. A geometria tem os campos **radius** e **height** que definem o raio das duas bases e a altura, e três campos opcionais **side, top** e **bottom** que, se omitidos, são assinalados para TRUE.

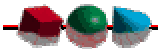
Sintaxe:

```
geometry Cylinder { radius □    #raio
                    height □    #altura
                    side TRUE/FALSE     #tem a lateral?
                    top TRUE/FALSE      #tem a parte de cima?
                    bottom TRUE/FALSE   #tem a parte de baixo?
                  }
```

Exemplo:

```
#VRML V2.0 utf8
#Programa Cylinder.wrl
#Um canudinho verde com lateral, vazado no centro

Shape
{   appearance Appearance
    {   material Material
                { diffuseColor 0 1 0
                }
    }
    geometry Cylinder {   radius 0.5
                          height 4
                          side TRUE
                          top FALSE
                          bottom FALSE
                      }
}
```



3.4. Sphere

Para definir uma esfera (bola) usa-se a geometria **Sphere** do node **Shape**. O centro padrão de uma Sphere é (0,0,0). A geometria tem apenas um campo **radius** que define o seu raio.

Sintaxe:

```
geometry Sphere { radius □
                  }
```

Exemplo:

```
#VRML V2.0 utf8
#Programa Sphere.wrl
#Uma esfera amarela

Shape
{   appearance Appearance
    {   material Material
                { diffuseColor 1 1 0
                }
    }
    geometry Sphere {   radius 1.89
                    }
}
```

3.5. Unindo formas primitivas

As formas primitivas podem ser unidas num único programa VRML para se construir objetos formados a partir delas. No programa **Shapes.wrl**, que se segue, o objeto apresentado na figura 6 é construído utilizando-se apenas as formas primitivas Box, Cylinder e Sphere. Note que o centro de todas as formas primitivas é o ponto (0,0,0).

```
#VRML V2.0 utf8
#Shapes.wrl
#Demonstra o uso de multiplos shapes no mesmo
#arquivo
#Space Station by David R. Nadeau
#Alterado por Elisamara de Oliveira em  04/2001

Shape
{   appearance Appearance
    {  material Material {  }   }

    geometry Box  { size 1.0 1.0 1.0 }
}

Shape
{   appearance Appearance
    { material Material { diffuseColor 1 1 0 } }

    geometry Sphere { radius 0.7  }
}

Shape
{   appearance Appearance
    { material Material { diffuseColor 0 1 1} }

    geometry Cylinder {  radius 1.25
                         height 0.1
                      }
}

Shape
{   appearance Appearance
    { material Material { } }

    geometry Cylinder { radius 0.4
                        height 2.0
                      }
}

Shape
{   appearance Appearance
    {  material Material { }   }
    geometry Cylinder { radius 0.3
                        height 3.0
                      }
}

Shape
{   appearance Appearance
    {  material Material { diffuseColor 0 1 1 }
    }
    geometry Cylinder { radius 0.1
                        height 6.0
                      }
}
```

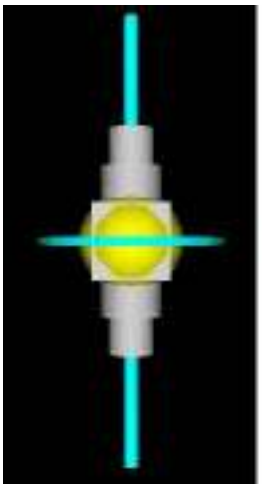
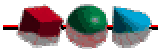


Figura 6 – Objeto construído a partir de formas primitivas pelo programa Shapes.wrl



4. Transform- Transformações Geométricas

No programa **Shapes.wrl** apresentado na seção 3.5, todas as formas primitivas foram desenhadas a partir do centro (0,0,0). Mas é possível deslocar (transladar) essas formas no espaço tridimensional e mesmo rotacioná-las ou aumentar ou diminuir seus tamanhos originais a partir de operações de translação, rotação e escala. Essas operações são realizadas a partir do node **Transform**.

O Transform é um **group node**. Um group node permite que se defina um conjunto de nodes como um único objeto. Mas o principal propósito do node Transform é definir um sistema de coordenadas local para os nodes pertencentes ao grupo.

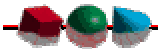
Todos os nodes dentro de um grupo Transform são afetados pelas transformações geométricas. Os principais campos (os outros campos: **scaleOrientation**, **center**, **bboxCenter** e **bboxSize** não serão aqui abordados) que estão presentes no node Transform são:

scale: especifica uma transformação de escalamento 3D. São passados 3 valores em ponto flutuante: o primeiro se refere ao escalamento do objeto na direção X, o segundo na direção Y e o terceiro na direção Z.

rotation: define uma rotação em torno de um eixo. A rotação é definida por um vetor (x,y,z) e um ângulo em radianos.

translation: define a origem do sistema de coordenadas local.

children: contém todos os nodes incluídos no group.



Sintaxe:

```
Transform
{  scale □ □ □
   rotation □ □ □ □
   translation □ □ □
   children []
 }
```

Exemplo:

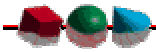
```
#VRML V2.0 utf8
#TransRotScale.wrl
#Exemplo de Translação, rotação e escala juntos

Shape
{ appearance Appearance
  {  material Material
             { diffuseColor 0 0 1
             }
  }
  geometry Cylinder { height     4
                      radius     0.5
                    }
}

Transform
{     translation 2 2 2
      rotation 3 2.0 2.0 3.14
      scale 1 1.2 1
      children [   Shape
                    {  appearance Appearance
                       { material Material
                                  { diffuseColor 0 1 1
                                  }
                       }

                       geometry Cylinder { height     4
                                           radius     0.5
                                         }
                    }

              ]
}
```



O programa **TransRotScale.wrl** mostra um cilindro azul centrado no ponto (0,0,0) e o mesmo cilindro com a cor alterada para ciano (repetido no campo children do node Transform), após sofrer as operações de translação, rotação e escala, conforme ilustra a figura 7.

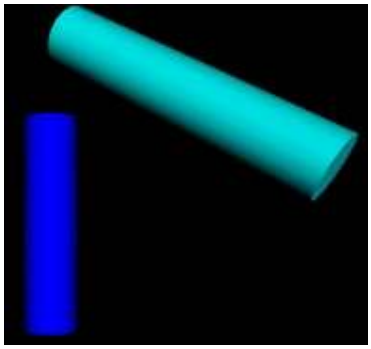


Figura 7 – Resultado do programa TransRotScale.wrl

4.1. USE DEF e Inline

O programa TransRotScale.wrl descreve o node Cylinder duas vezes: a primeira para mostrar o cilindro original e a segunda, dentro do node Transform, no campo children, para mostrá-lo após as transformações geométricas nele aplicadas. No entanto, esta repetição pode ser evitada através dos comandos **USE DEF** e **Inline** conforme mostram os exemplos abaixo.

```
#VRML V2.0 utf8
#Inline.wrl
#Exemplo do uso de Inline para evitar repetições do Nó Cylinder

Inline {url "cylinder.wrl"}

Transform
{
     translation 0.8 1.8 1.8
     rotation 1.8 3.8 1.8 2.8
     scale 0.8 1.8 2.8
     children [ Inline { url "cylinder.wrl"} ]
}
```

```
#VRML V2.0 #VRML V2.0 utf8
#UseDef.wrl
#Exemplo do uso de USE e DEF para evitar repetições do Nó Cylinder

DEF Cilindro Shape
              { appearance Appearance
                {
                    material Material { diffuseColor 1 0 1  }
                }
                geometry Cylinder { height     4
                                    radius     0.5
                                  }
              }

Transform
{
     translation 0.8 1.8 1.8
     rotation 1.8 3.8 1.8 2.8
     scale 0.8 1.8 2.8
     children   USE Cilindro

}
```

4.2. Scale

A operação de escala permite modificar o tamanho de uma forma geométrica. A forma pode ser aumentada ou reduzida em qualquer uma das três dimensões X (largura), Y (altura) e Z (profundidade). O fator de escala deve ser positivo. A figura 8 mostra um **box** sem escalamento, depois escalado em x por 0.2, em seguida escalado em y também por 0.2 e por último escalado ao mesmo tempo em x e em y por 0.2.

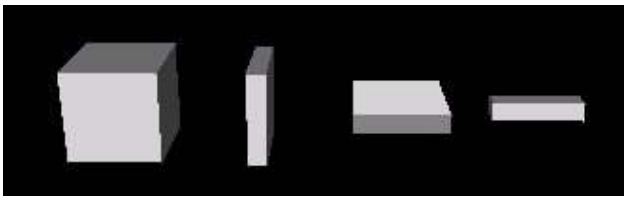
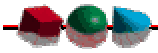


Figura 8 – Box escalado em diferentes dimensões



Exemplo:

```
#VRML V2.0 utf8
#Scale.wrl


Inline {url "cylinder.wrl"}

Transform
{      scale 2 1 1
       translation 2 1 2
       children
       [      Shape
              {  appearance Appearance
                 {  material Material { diffuseColor 0 1 1  }
                 }
                 geometry Cylinder
                 { height      4
                   radius      0.5
                 }
              }

       ]
}
```

4.3. Rotation

A rotação é definida por um vetor (x,y,z) e um ângulo em radianos. O vetor define o eixo de rotação enquanto o ângulo define o quanto que o objeto será rotacionado no sentido horário.

Na figura 9, o primeiro box não está rotacionado; o segundo box está rotacionado por um ângulo de 45 graus em torno de Y; o terceiro está rotacionado em 45^{o} em torno de X; o último está rotacionado em 45^{o} usando o vetor (1 1 0).

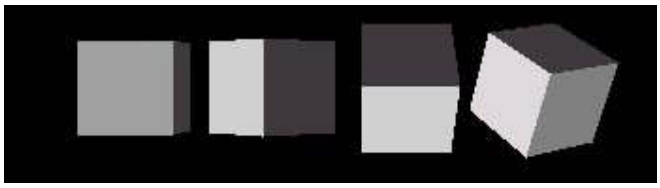
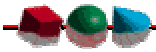


Figura 9 – Box rotacionado em diferentes eixos



Exemplo:

```
#VRML V2.0 utf8
#Rotation.wrl


Inline {url "cylinder.wrl"}

Transform
{      rotation 2 2 1 3
       children [    Shape
                     {  appearance Appearance
                        {    material Material { diffuseColor 0 1 1   }
                        }
                        geometry Cylinder
                        { height      4
                          radius      0.5
                        }
                      }
                ]
}
```

4.4. Translation

A translação perimite que se desloque o objeto para o lugar que se queira no espaço tridimensional. A figura 10 ilustra este conceito. As linhas tracejadas representam o sistema de coordenadas usado fora do node Transform. As linhas sólidas reprsentam o sistema de coordenadas local dentro do node Transform definido pela seta que aponta para a nova posição (x,y,z). Na figura 10, a translação é para (1,1,0).

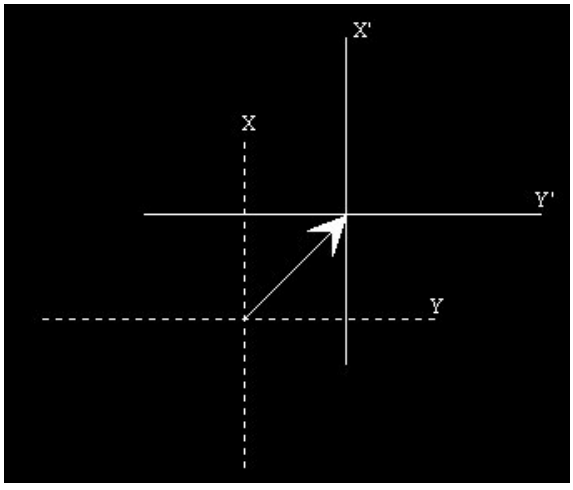
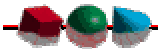


Figura 10 – Novo eixo a partir de uma translação



Exemplo:

```
#VRML V2.0 utf8
#Translation.wrl

#ESTE É O CILINDRO ORIGINAL
Inline {url "cylinder.wrl"}

#ESTE É O CILINDRO TRANSLADADO
Transform
{     translation 2 1 2
      children
      [   Shape
          {  appearance Appearance
                        {  material Material { diffuseColor 0 1 1  }
                        }
             geometry Cylinder
                      { height     4
                        radius     0.5
                      }
           }

       ]
}
```