	Disciplina	CGRAF
	Curso	SI
ALEXANDRE BELETTI FERREIRA	AULA – 10	

Realidade Virtual

Há um tempo atrás, surgiu a idéia de levar a Realidade Virtual para a Internet. Dessa idéia surgiu o VRML, que é a abreviação de ***Virtual Reality Modeling Language***, ou Linguagem para Modelagem em Realidade Virtual. VRML é uma linguagem independente de plataforma que permite a criação de ambientes virtuais por onde se pode passear, visualizar objetos por ângulos diferentes e até interagir com eles. A primeira versão da linguagem não possibilita muita iteração do usuário com o mundo virtual, mas versões recentes acrescentam características como animação, movimentos de corpos e interação 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.

Apresentada pela primeira vez em 1994 na Primeira Conferência sobre 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 às âncoras do HTML, ou seja, podem-se criar âncoras em um mundo virtual que levem a outros mundos virtuais.

A linguagem trabalha com geometria 3D (VRML 1.0 possui algumas primitivas : cubo, cone, cilindro e esfera) e suporta transformações (rotação, translação, escala), texturas, luz e sombreamento. Outra característica importante da linguagem é o Nível de Detalhe (*LOD, level of detail*) que disponibiliza a quantidade certa de dados para um objeto baseado na sua importância na cena. Isso torna rápida a visualização e possibilita ao usuário ajustar o nível de detalhe que lhe for melhor.

Para navegar em mundos virtuais criados com a linguagem você precisará usar browsers que suportem VRML. Assim, ao invés de visitar homepages, você visitará homeworlds. Existem muitos browsers disponíveis que suportam diretamente a linguagem. Outros browsers que não suportam necessitam de software adicional (*plug-in*).

Características Básicas

Tudo que se precisa para escrever um código VRML é um editor de textos. Uma vez editados, os arquivos são gravados em formato ASCII com a extensão .wrl. Na verdade, a linguagem apenas descreve como os ambientes tridimensionais serão representados pelo browser. O arquivo não precisa ser compilado e não é executado por ninguém. Pode-se, por exemplo, criar um cubo e gravá-lo em um arquivo chamado *cubo.wrl*. O código VRML para este cubo descreverá as características do ambiente, como coordenadas, luz, cores, sombreamento etc. Também pode-se colocar em um mundo objetos que estão localizados remotamente em outros lugares na internet, além de links que levam a outros homeworlds ou homepages.

Ficou definido, para fins de identificação, que todo arquivo VRML 2.0 tem que ter o cabeçalho :

#VRML V2.0 utf8

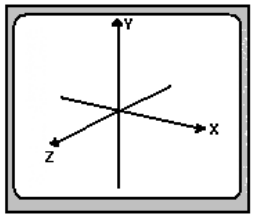
O caracter '#' também significa **comentário**. Toda linha que começa com # será ignorada pelo browser. É bom usar comentários no meio do código, isso facilita a compreensão e identificação de partes do cenário, como por exemplo:

```
# FALS - Computação Gráfica
```

Unidades de Medidas

VRML usa o sistema cartesiano 3D. A seqüência dos eixos é X,Y,Z e a unidade de medida para distâncias é **metros** e para ângulos, **radianos**. A medida para distância não é definida formalmente, mas adoutou-se o metro como unidade para que mundos diferentes feitos por usuários diferentes possam ser eventualmente unidos em um só mundo.

Como referência, utiliza-se 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, como mostra a figura a seguir:



Nodes ou Nó

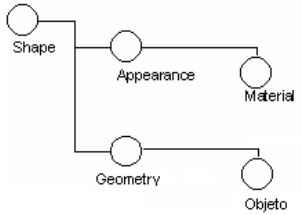
Basicamente, pode-se dizer que um ***node*** é um conjunto de especificações que determinam as características dos objetos contidos no cenário. Os nodes definem a hierarquia e as características individuais de cada objeto dentro do contexto geral.

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 tamanho de um cubo, diâmetro de uma esfera, intensidade da luz ambiente, cor etc. Não é necessário dar nome a um node, mas se isso for feito não se pode ter dois nodes com nomes iguais.

VRML – FORMAS BÁSICAS Geométricas

Um node é a construção fundamental de um arquivo VRML. Alguns nós são objetos – Box, Cylinder, Cone, SpotLight, etc... Outros nós são usados como *containers* que armazenam nós com alguma relação lógica. Um nó **Shape**, por exemplo, contém um nó (**geometry**) que indica a forma do objeto e outro (**appearance**) que controla a sua aparência. Estes nós podem, por sua vez, conter outros, e assim por diante.

Veja o grafo da cena:



Shape é a representação mais básica de um objeto em VRML. Ele encapsula um nó que representa a aparência do objeto e outro que representa a sua forma.

Appearance é um dos tipos especiais de nós que aparecem na Figura 4.2. Ele só aparece como o valor do campo *appearance* de um nó Shape, nunca como um nó indepenednte.

Se o campo *material* é nulo (NULL), a geometria associada com Appearance não é mostrada pelo browser:

appearance Appearance { }

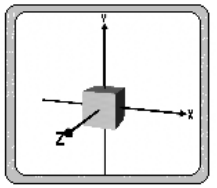
No entanto, se um nó Material vazio é especificado como o valor de *material*, o objeto é renderizado pelo browser utilizando um valores de um material padrão:

appearance Appearance { material Material { } }

geometry contém um nó de geometria, como Box, Cone, IndexedFaceSet e outros.

Com estes conceitos podemos criar a nossa primeira cena, como mostra a codificação a seguir:

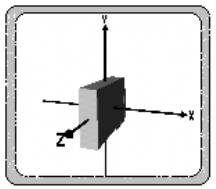
```
#VRML V2.0 utf8
Shape {
   appearance Appearance {
      material Material { }
   }
   geometry Box { }
}
```



Agora iremos manipular o tamanho de um cubo, para tanto, veja as seguintes linhas de código:

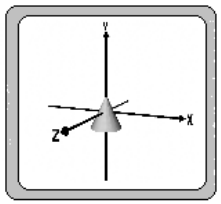
```
#VRML V2.0 utf8
Shape {
        appearance Appearance {
                material Material { }
        }
        geometry Box {
                size 1.0 3.0 5.0
                }
        }
```

X,Y,Z



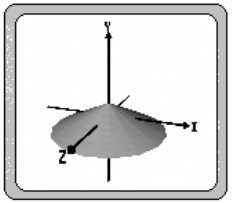
Vejamos como construir um cone padrão:

```
#VRML V2.0 utf8
Shape {
 appearance Appearance {
        material Material { }
        }
 geometry Cone { }
 }
```



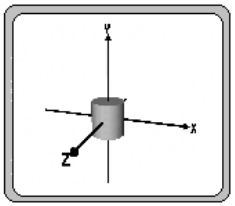
Manipulando o tamanho do cone, temos então a seguinte linha de código:

```
#VRML V2.0 utf8
Shape {
   appearance Appearance {
      material Material { }
   }
   geometry Cone {
      bottomRadius 3.5
      height 1.5
   }
```



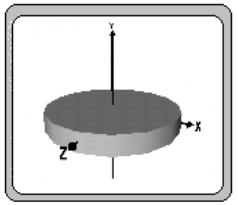
Para construir um cilindro, devem-se criar as seguintes linhas de código:

```
#VRML V2.0 utf8
Shape {
   appearance Appearance {
      material Material { }
   }
   geometry Cylinder { }
}
```



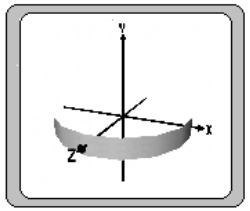
Manipulando o tamanho do cilindro, terão as seguintes linhas de código:

```
#VRML V2.0 utf8
Shape {
   appearance Appearance {
      material Material { }
   }
   geometry Cylinder {
      radius 4.0
      height 1.0
   }
}
```



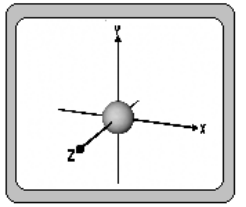
Alterando determinadas características do cilindro para criar efeitos especiais, veja as seguintes linhas de códigos:

```
#VRML V2.0 utf8
Shape {
   appearance Appearance {
      material Material { }
   }
   geometry Cylinder {
      radius 4.0
      height 1.0
      top    FALSE
      bottom FALSE
   }
}
```



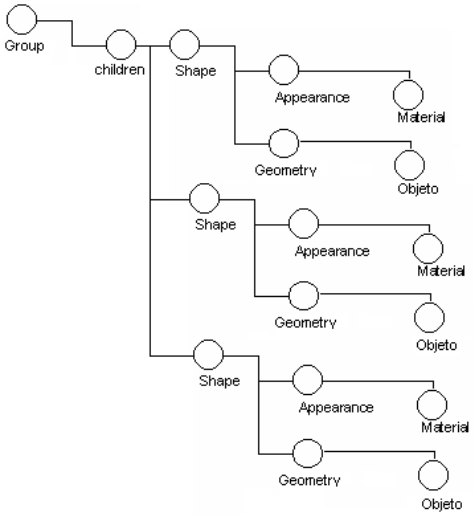
Para construir uma esfera, digite as seguintes linhas de códigos:

```
#VRML V2.0 utf8
Shape {
   appearance Appearance {
      material Material { }
   }
   geometry Sphere { }
}
```



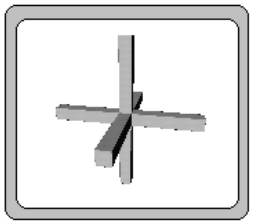
Para efetuar construções combinando elementos geométricos, devemos discutir os seguintes conceitos:

Precisamos criar um nó de agrupamento, para tanto, utiliza-se a nó **Group**.
Os nós de agrupamento são usados para criar grafos de transformações gráficas hierárquicas. Eles têm um campo **children** que contém uma lista de nos que são os descentes da transformação do grupo. Cada nó de agrupamento define um espaço de coordenadas para seus filhos, que é relativo ao espaço de coordenadas do seu nó-pai, ou seja, as transformações são cumulativas no sentido da descendência dos nos na hierarquia do grafo de cena, como mostra a figura a seguir:



Vamos construir então apoiado nestes conceitos um sinal de + em 3D, para tanto, digite as seguintes linhas de códigos:

```
#VRML V2.0 utf8
Group {
   children [
      Shape {
         appearance DEF Padrao Appearance {
            material Material { }
         }
         geometry Box {
            size 25.0 2.0 2.0
         }
      },
      Shape {
         appearance USE Padrao
         geometry Box {
            size 2.0 25.0 2.0
         }
      },
      Shape {
         appearance USE Padrao
         geometry Box {
            size 2.0 2.0 25.0
         }
      }
   ]
}
```



Façamos agora mais uma construção com os elementros geometry, porém permitindo que eles sofram deslocamento em relação à origem para tanto devemos ampliar os nossos conhecimentos do nó children que é o grupo de nós-filhos que são afetados pelas transformações especificadas neste nó **Transform**.

Transform é um nó de agrupamento que define um sistema de coordenadas para seus filhos que é relativo aos sistemas de coordenadas de seus antecessores no grafo de cena.

Um dos campos do Transform é o **translation** que define uma transformação geométrica 3D, em relação ao eixo (X,Y,Z)

Esta aplicação irá construir uma cabana aonde definiremos uma cor utilizando o nó material com o campo **diffuseColor** que utiliza o padrão RGB com valores compreendidos no intervalo de 0 a 1.

Para tanto, veja as linhas de códigos a seguir:

```
#VRML V2.0 utf8
Group {
   children [
   # Desenha as paredes da cabana
      Shape {
         appearance DEF Cor Appearance {
            material Material {
               diffuseColor 0.6 0.4 0.0
            }
         }
         geometry Cylinder {
            height 2.0
            radius 2.0
         }
      },
   #Desenha o Teto da cabana
      Transform {
         translation 0.0 2.0 0.0
         children Shape {
            appearance USE Cor
            geometry Cone {
               height 2.0
               bottomRadius 2.5
            }
         }
      }
   ]
}
```