
VRML

Virtual Reality Modeling Language

INTRODUÇÃO AO VRML

Carlos Guedes

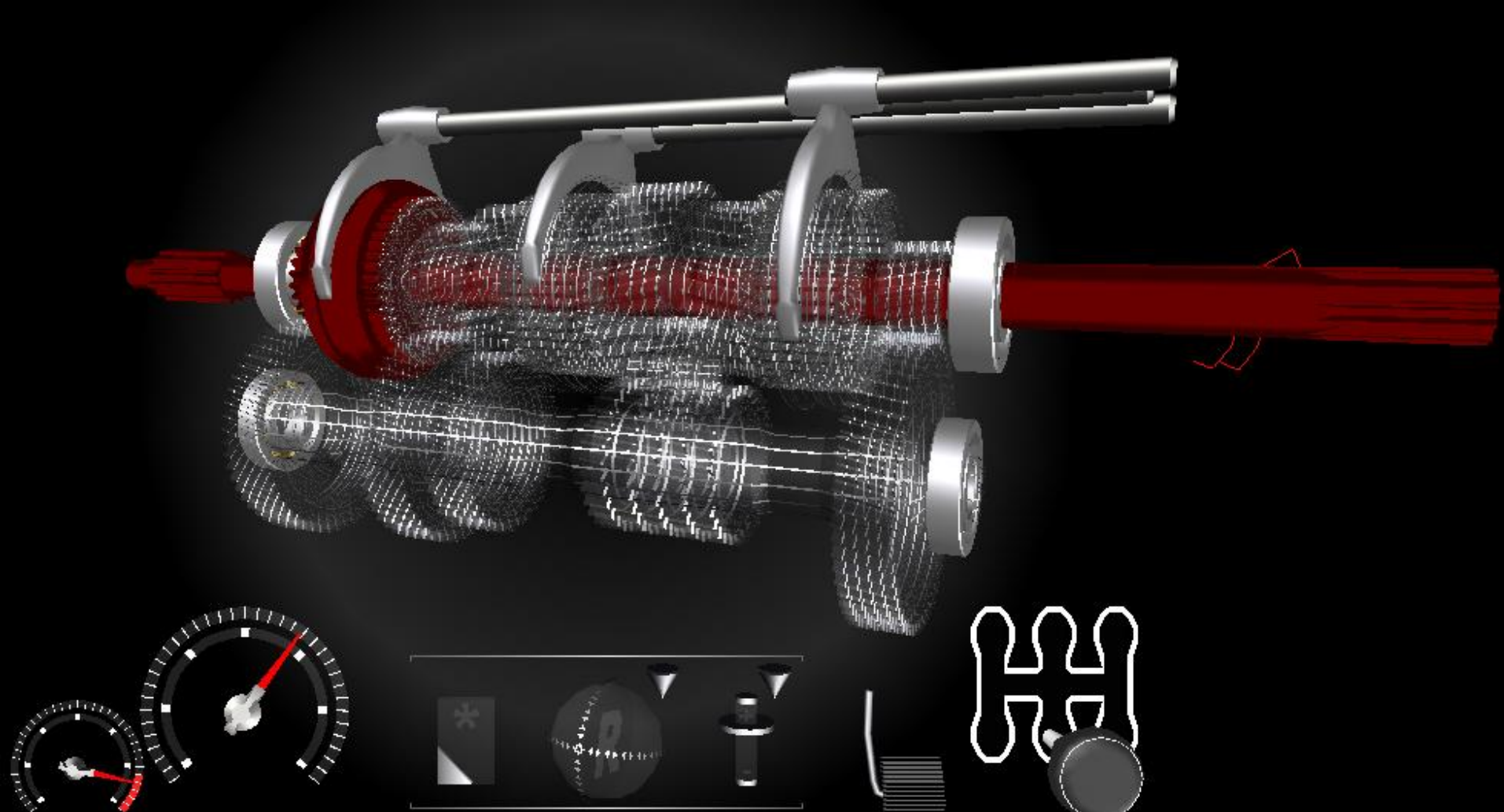
Baseado na apresentação de Márcio Bueno (masb@cin.ufpe.br), 09 de maio de 2005



Exemplo



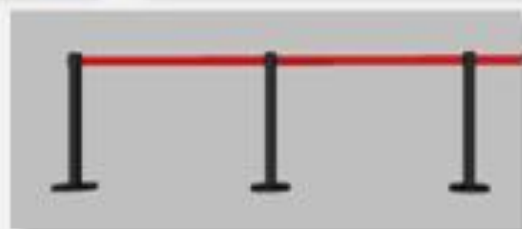
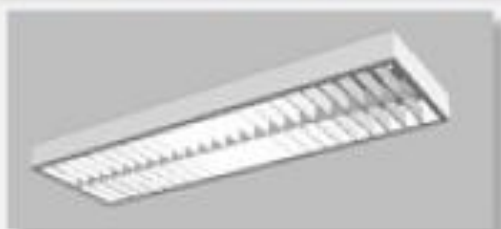
Exemplo



Trabalhos de CG (1s0607)

- **Objectivo:** Modelar cena do campus do ISEL, à escolha dos alunos
- Algumas das cenas modeladas
 - Átrio da secretaria
 - Átrio do DEETC
 - Auditório do DEEA
 - Biblioteca
 - Entrada do “Aqui estuda-se”
 - Pavilhão G
 - Sala LSI
 - Sala de Química
 - WCs do DEETC
- Cenas publicadas na exposição de informática de 2007 – [Sinfo07](http://www.deetc.isel.ipl.pt/sinfo07/)
(<http://www.deetc.isel.ipl.pt/sinfo07/>)

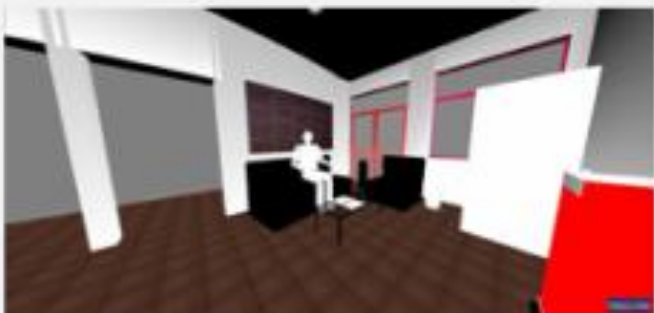
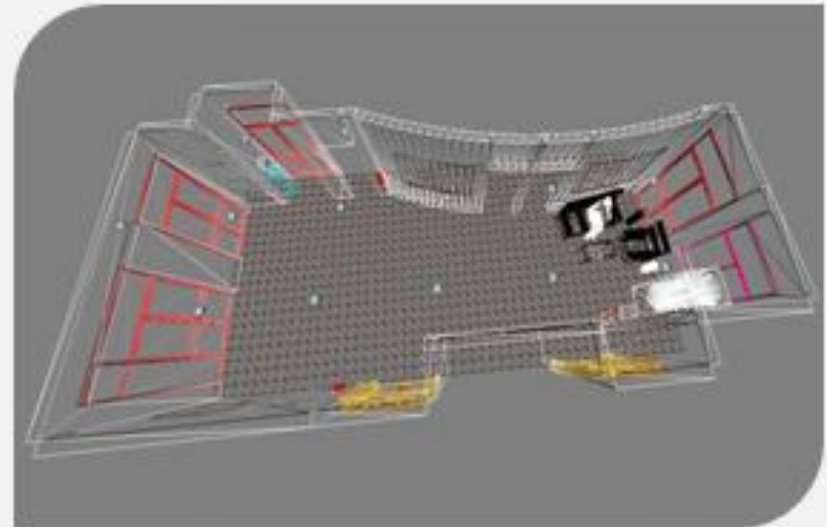
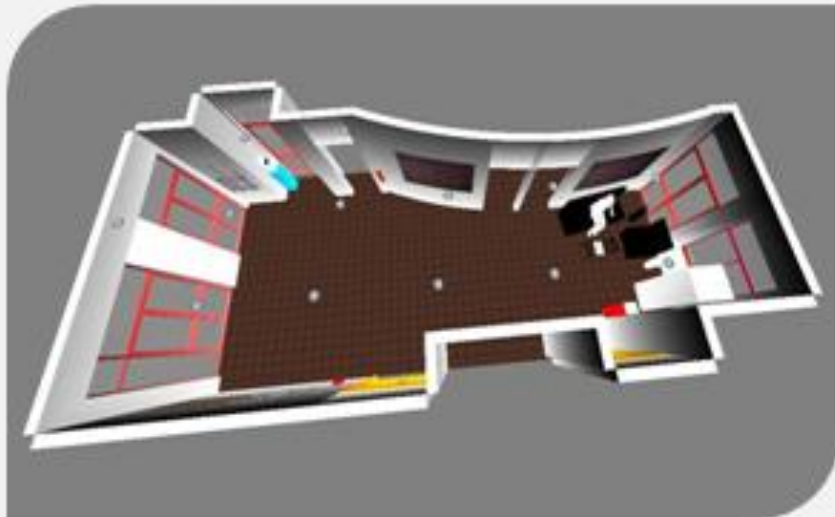
Átrio da Secretaria



Desenvolvido por:

Luis Gonçalves
Filipe Nascimento
João Cordeiro

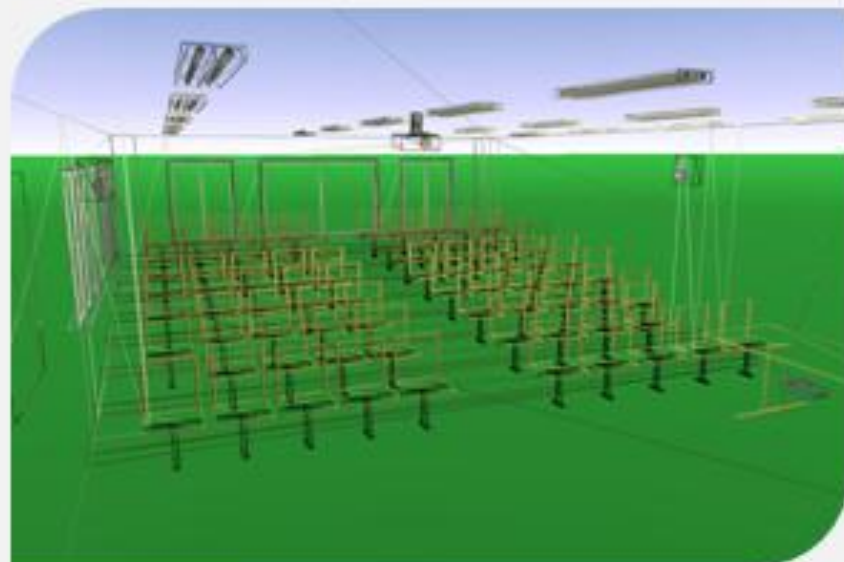
Átrio do DEETC



Desenvolvido por:

Carlos Vicente
Eduardo Loureiro
Rui Ferrão

Auditório DEEA



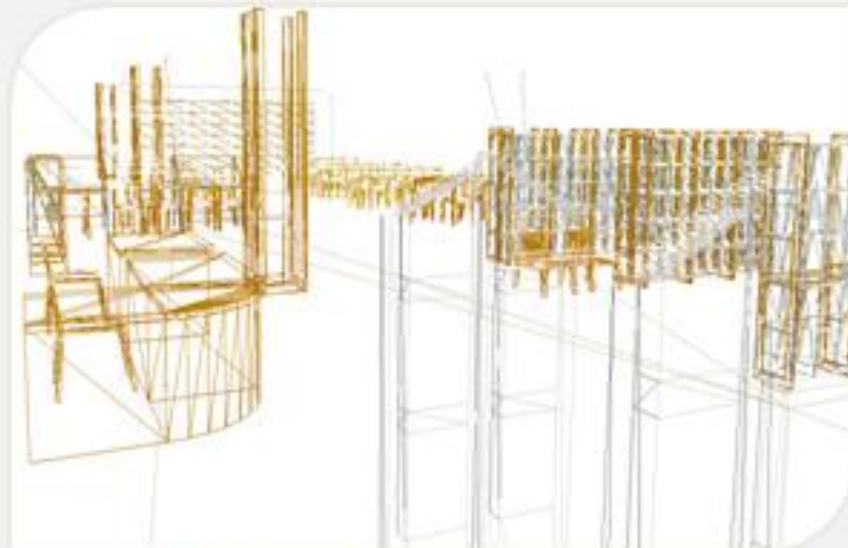
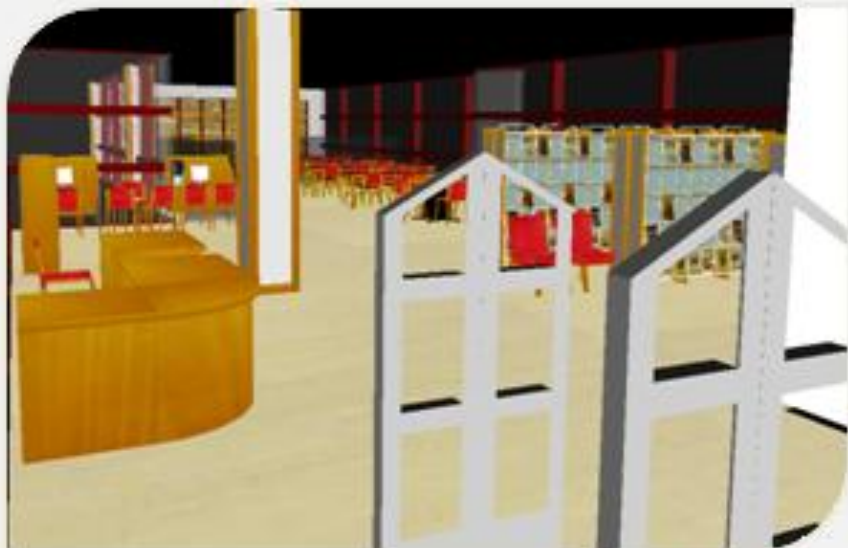
Desenvolvido por:

Luis Brás

Nuno Miguel

Vitor Chão

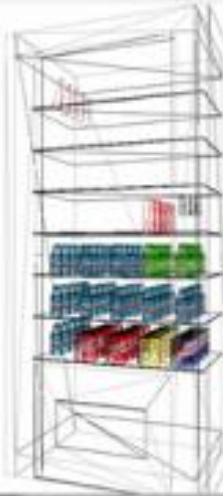
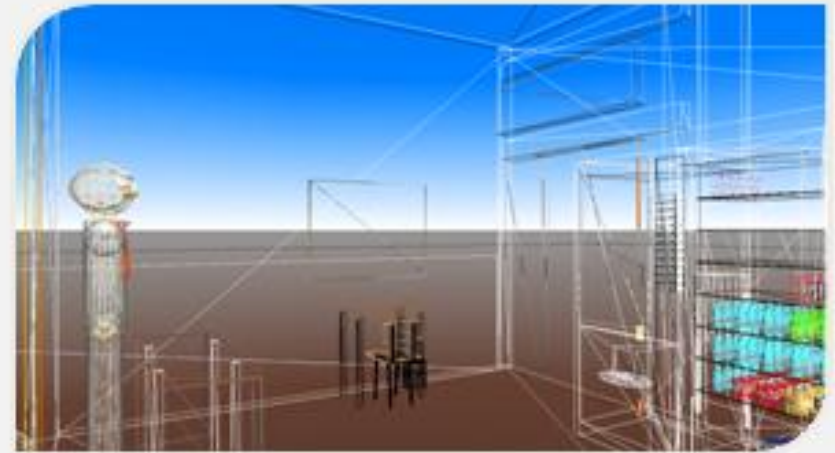
BIBLIOTECA



Desenvolvido por:

Virginia Ramalho
Tony Tam

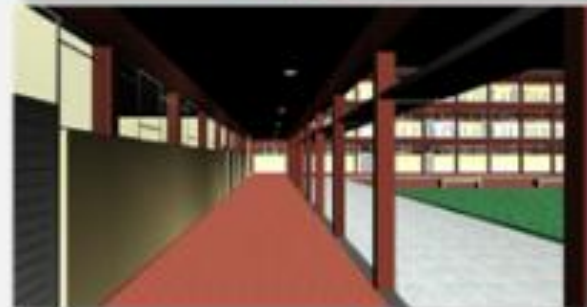
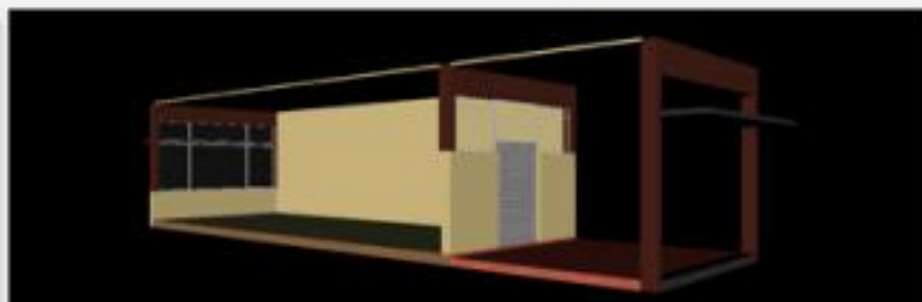
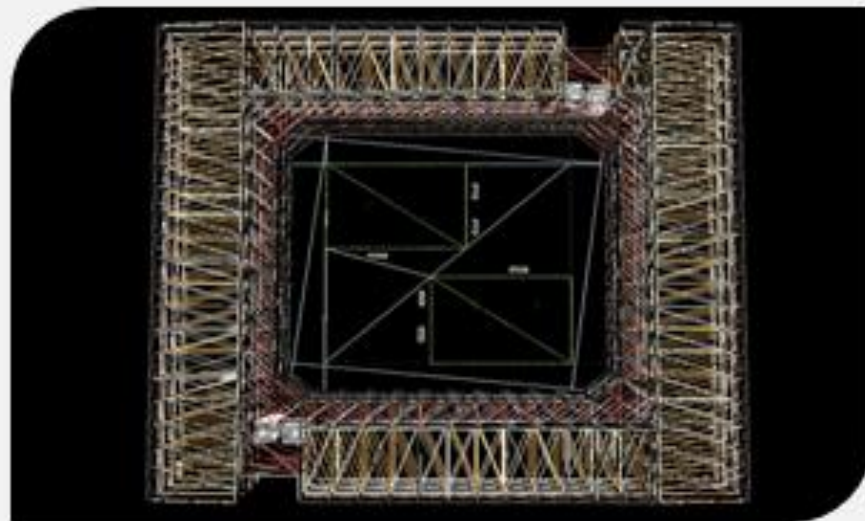
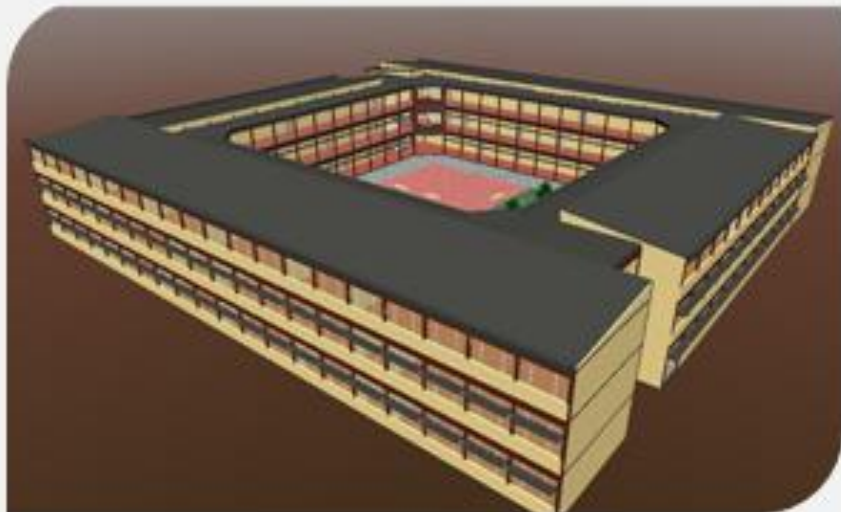
Entrada do “Aqui Estuda-se”



Desenvolvido por:

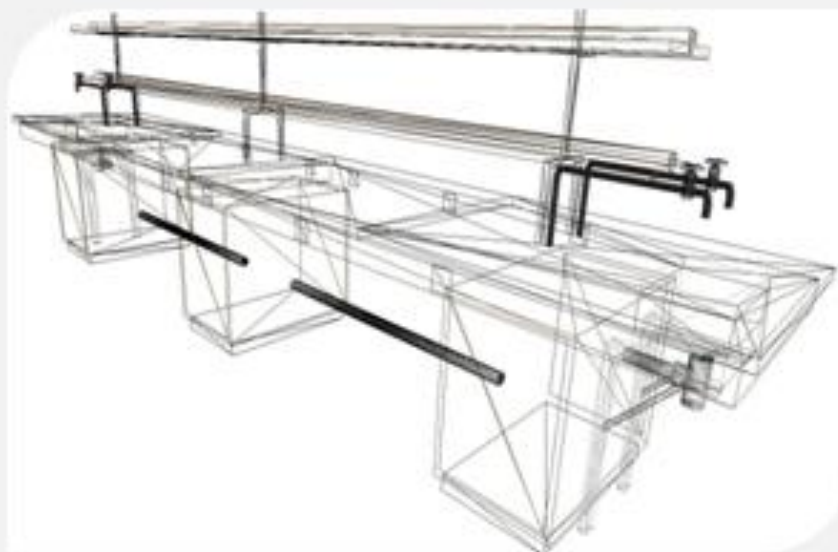
Ana Filipa Carvalho

Pavilhão G



Desenvolvido Por:
Luís Cunha
Tiago Ramalho
Emanuel Carvalho

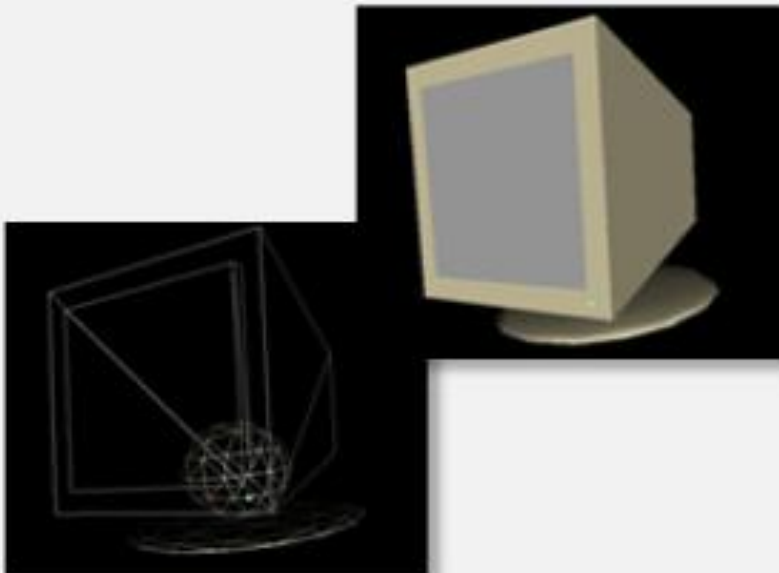
Sala de Química



Desenvolvido por:

João Neto
Manuel Felício
Tiago Freitas

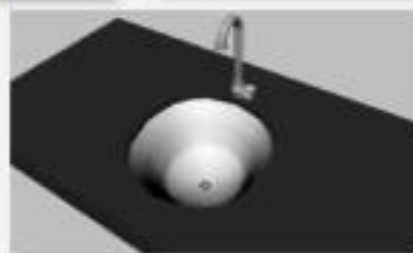
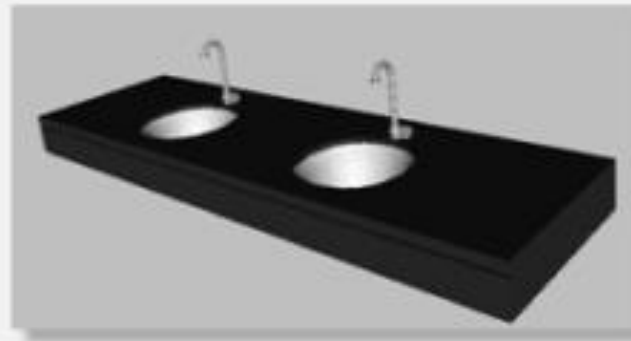
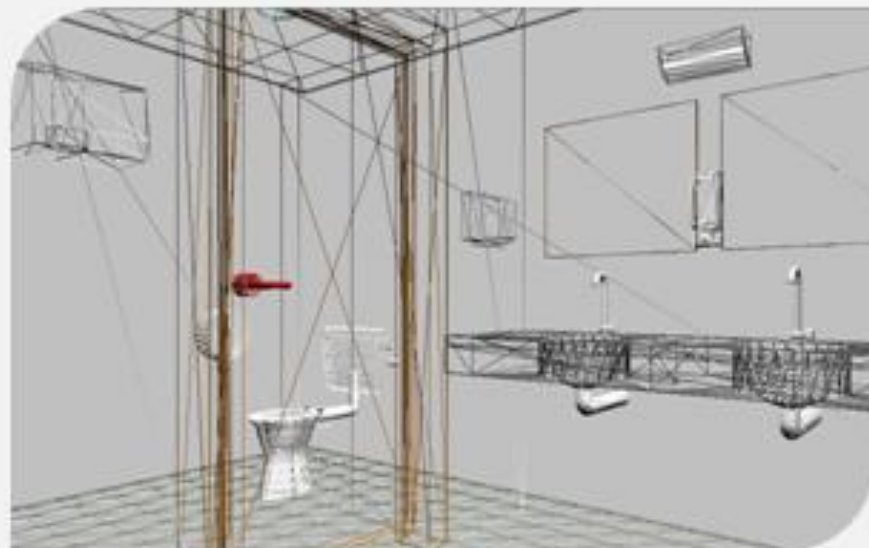
Sala LSI



Desenvolvido por:

David Vitorino,
Hélio Miranda,
Ricardo Santinho

WC do DEETC



Desenvolvido por:

Alexandre Serebuga
José Costa
João Antunes

Sumário

- O que é VRML
- Histórico
- Objectivos
- VRML, Internet e WWW
- Visão geral do VRML
- Estrutura de um documento VRML
- Conceitos-chave
- Semântica dos Nós
- Primitivas Básicas
- Primitivas Avançadas

O Que é VRML

■ Virtual Reality Modeling Language

→ Suporte para RV não-imersiva

→ VRML não suporta imersão

- Capacetes 3D, luvas digitais

■ Não é exactamente "Modeling Language"

→ Verdadeira linguagem de modelação 3D deve fornecer primitivas de modelação geométricas e mecanismos muito mais ricos

■ O que é então?

→ Um formato de intercâmbio para 3D

→ Publicação de páginas Web em 3D (~HTML 3D)

- Jogos, engenharia, visualização científica, experiências educacionais, arquitectura

→ Um modelo coerente para integração de 3D, 2D, texto e multimédia

Histórico

- **1989:** Silicon Graphics – projecto *Scenario*
- **1992:** *Iris Inventor*
- **1994:** *Open Inventor*
- **1994:** *Labyrinth*
- **1995:** Criação e ajustes
- **1996:** Proposta de VRML 2.0
- **1997:** Norma ISO
 - Especificação "*VRML97*" (aka VRML 2.0)
- **~2006:** VRML 3.0 ⇔ X3D

Objectivos da VRML

■ Modelação de objectos 3D

- Composição de "mundos" pela hierarquia de objectos
- Animação, interação, multimédia
- formato "aberto" (várias aplicações podem ler/gravar)

■ Internet e WWW

- VRML foi projectado para ser usado com a infraestrutura Internet/WWW existente

■ Simplicidade

- Suporte VRML deve ser facilmente adicionado às aplicações
 - *Vários browsers podem e devem dar suporte*

■ Facilidade de Composição

- Facilidade em se "montar o todo a partir das peças"
- Produtividade e trabalho em grupo

■ Escalabilidade

- O navegador VRML deve operar com "mundos" distribuídos na Internet
- Independente da potência da máquina
- Mundos ajustados ao desempenho da rede (14.4K ~ OC12)
- Especificação suporta adições futuras

■ Otimização do desempenho

- Algoritmos de manipulação 3D têm efeito nas ramificações agrupadas
- Se algo não puder ter desempenho, então não fará parte da especificação

■ Versões futuras

- Interação entre vários utilizadores
- Criaturas autónomas que podem sentir e reagir no ambiente

VRML, Internet e WWW

- Navegadores suportam VRML através de *plug-ins*
- Arquivos VRML podem referenciar vários outros formatos
 - ➔ JPEG, PNG, GIF, MPEG podem ser usados para compor a textura de objectos
 - ➔ WAV, MIDI podem ser usados para sons emitidos pelo ambiente ou por objectos
 - ➔ Arquivos com código Java ou JavaScript podem ser referenciados para implementar comportamento de objectos
- VRML permite *hyperlinks*

Visão Geral do VRML

■ Estrutura Gráfica da Cena

- VRML descreve "mundos" usando uma cena gráfica hierárquica
- Entidades na cena são chamados "nós" (*nodes*)
- VRML 2.0 define 54 tipos de nós:
 - Primitivas geométricas
 - Propriedades de aparência, som
 - Agrupamento de nós
- Nós armazenam seus dados (atributos) em "campos" (*fields*)
- VRML 2.0 define 20 tipos de campos (números, *arrays*)
- Nós podem conter outros nós
 - Alguns nós podem ter "filhos" (*children*)
 - Alguns nós podem ter mais de um "pai"
 - Um nó não se pode conter a si mesmo (recursividade).
 - Criação de "mundos" complexos ou objectos a partir de subpartes

■ Arquitectura de Eventos

- Mecanismo de geração de eventos suporta que os nós da cena comuniquem entre si
- Cada tipo de nó define os nomes e tipos de eventos que podem ser gerados ou recebidos

■ Sensores

- *Sensores* são as primitivas básicas de animação e interação com o utilizador
 - O nó "**TimeSensor**" gera eventos com o passar do tempo e é a base para todos os comportamentos animados
 - Outros sensores geram eventos quando o utilizador se movimenta através da cena ou quando interage através de algum dispositivo de entrada
- Sensores geram apenas eventos. Eles devem ser combinados com outros nós para ter qualquer efeito visual na cena

■ **Scripts e Interpoladores**

- *Scripts* permitem ao autor da cena definir comportamentos arbitrários, usando qualquer linguagem suportada. (No VRML2.0: *java* ou *javascript*)
- Nós "**Script**" podem ser inseridos entre geradores de eventos (sensores) e receptores.
- Interpoladores são *scripts* embutidos que executam animações. Eles podem ser combinados com "**TimeSensor**" e algum nó da cena para fazer objectos se moverem.

■ **Encapsulamento e Reutilização**

- Geometria, propriedades, animações ou comportamento podem ser encapsulados separadamente ou junto da cena.
- Permite a definição de um novo tipo de nó a partir da combinação de tipos existentes de nós
 - Biblioteca de objectos 3D
 - Facilidade de uso, trabalho em equipa
 - Redução do tamanho de documentos

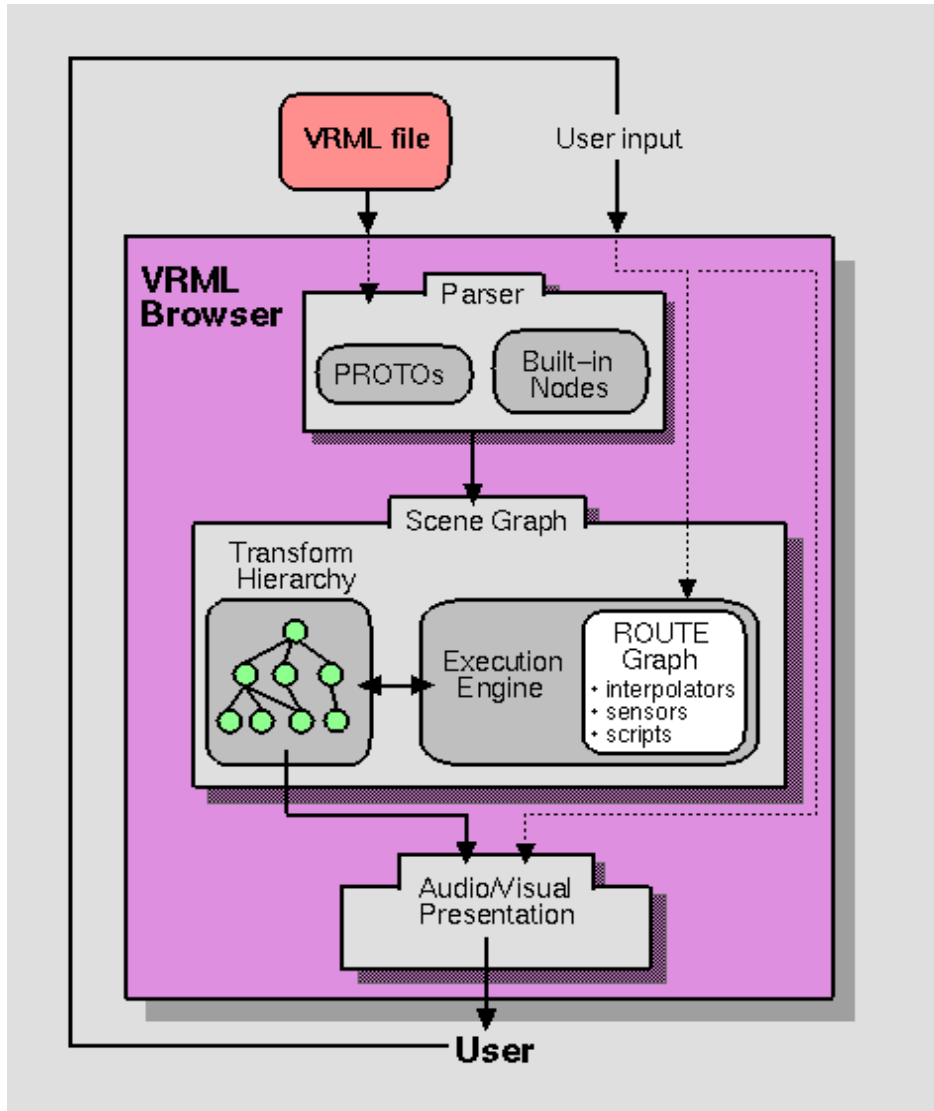
■ Cenas Distribuídas

- VRML 2.0 inclui duas primitivas para que a definição de uma cena possa ser distribuída
- O nó "***Inline***", para a inclusão de outra cena armazenada num documento externo (URL) – em qualquer lugar da Web
- O comando "***EXTERNPROTO***", para que definições de novos nós possam ser acedidos de qualquer lugar da Web. É o mecanismo básico de extensão o VRML.

Visão Geral do VRML

cont.

■ Modelo Conceitual de um Browser VRML



Estrutura dos documentos VRML

- Documentos VRML têm extensão **.wr1** (*world*)
- Cabeçalho (obrigatório)
- Cena
 - Definição dos objectos gráficos
 - Contém os nós (*nodes*) que descrevem objectos e suas propriedades
 - objectos podem ter geometria hierarquicamente agrupada para prover representações áudio -visuais complexas
- Protótipos
 - Extensões de nós VRML definidos externamente pelo utilizador
- Eventos
 - Definições dos eventos associados aos objectos

Estrutura dos documentos VRML

■ Cabeçalho

- Primeira linha do arquivo
- Identifica o documento como VRML e informa o tipo de codificação usada (forma de codificação dos caracteres)
 - **#VRML V2.0** <encoding type> [**comment**] <line terminator>
- Apenas um caractere de espaço entre os componentes do cabeçalho
- <encoding type> típico: **"utf8"**
 - Existem outras codificações autorizadas pelo VRML 2.0
 - **"utf8"** indica texto livre (padrão UTF-8 da ISO 10646-1, RFC2044)
 - Também conhecido como Unicode

```
#VRML V2.0 utf8 Gerado por ...
```

Estrutura dos documentos VRML

- # inicia um comentário
 - Apenas o 1º comentário tem significado (cabeçalho)
- Caracteres **NÃO** permitidos para nomes
 - Primeira letra: + - 0-9 " ' # , . [] \ { } 0x0-0x20
 - Restante: " ' # , . [] \ { } 0x0-0x20
- Palavras reservadas
 - DEF, EXTERNPROTO, FALSE, IS, NULL, PROTO, ROUTE
 - TO, TRUE, USE, eventIn, eventOut, exposedField, field
- VRML é sensível a maiúsculas e minúsculas
 - "Sphere" é diferente de "sphere"
 - "BEGIN" é diferente de "begin"

Conceitos-chave: **Unidades de Medida**

■ Unidades de medida

Categoria	Unidade
Distância linear	Metros
Ângulos	Radianos
Tempo	Segundos
Cores	RGB ([0., 1.], [0., 1.], [0., 1.])

- Radianos foi escolhido por compatibilidade com o padrão da linguagem C (*math library routines*)
- Conversão de graus para radianos: **Radianos** = $\pi \cdot \text{graus} / 180$
- Tempo é expresso como um *double-precision floating point* (*precisão de nano-segundos*)

Conceitos-chave: **Nó**

■ Sintaxe da declaração do Nó

```
[DEF <name>] <nodeType> { <body> }
```

- Nome <name> opcional serve como nome de variável
- Tipo do nó <nodeType>
- Corpo do nó <body>
- O **[DEF <name>]** permite usar o nó depois referenciando o nome com o comando **USE** (ex.: **USE <name>**)
- Não precisa de espaço separando { } e []
- Exemplo:
Box { size 2 2 2 }

Conceitos-chave: **Campo**

- Sintaxe da declaração de um campo:

```
<fieldName> <fieldValue>  
<fieldName> [ <fieldValues> ]
```

- Exemplos:

radius 0.1

height 20

colorIndex [0 1 2 3 4 5 6 7]

coordIndex [0 1 2 3 -1, 8 9 10 11 -1, 0 1 11 8 5 4 -1]

■ Tipos de campo

→ **SFBool** TRUE/FALSE

→ **SFColor** 3 valores *float* entre 0 e 1 (RGB). Ex.: 0 0.7 0.2

→ **MFCOLOR** lista de valores **SFColor**. Ex.: [0 0.5 0, 1 0 0]

→ **SFFloat** 1 valor *float*. Ex.: 1.7456

→ **MFFloat** lista de valores *float*. Ex.: [1.2, 3.5, 7, 9.5]

→ **SFInt32**, **MFInt32** inteiro de 32 *bits* e lista

→ **SFNode**, **MFNode** nó e lista de nós

→ **SFRotation** 4 valores *float*:

- 3 primeiros: vector-eixo de rotação
- 4º: ângulo de rotação em radianos

→ **MFRotation** lista de **SFRotation**

■ Tipos de campo

- **SFString** string de caracteres utf-8. Ex.: “Eu”
- **MFString** lista de SFStrings. Ex: [“Eu”, “Tu”, “Ele”]
- **SFVec2f** vector 2D. Ex.: 1.4 7.3
- **MFVec2f** lista de vectores 2D. Ex: [2.3 3.4, 4.5 5.6]
- **SFVec3f** vector 3D. Ex: 1.4 7.3 9.1
- **MFVec3f** lista de vectores 3D. Ex: [2.3 3.4 4.5, 5.0 6.7 7.8]

- **SFTime, MFTIME**: tempo (*float*) e uma lista de tempos
Ex: 1947582537.2323, [1000000000.1, 9.8]

■ Tipos de campo

→ **SFImage** imagem bidimensional colorida ou em tons de cinzento

- 2 inteiros: largura e altura da imagem
- 1 inteiro: número de componentes de cor:
 - 1 para tons de cinza
 - 2 para tons de cinza com transparência
 - 3 para RGB
 - 4 para RGB com transparência
- N números hexadecimais ($N = \text{largura da imagem} \times \text{altura da imagem}$) para a cor dos pixels. Ex.:
 - Tons de cinza: 0X00 (cor preta)
 - RGB com transparência: 0x00FF007F (verde semi-transparente)

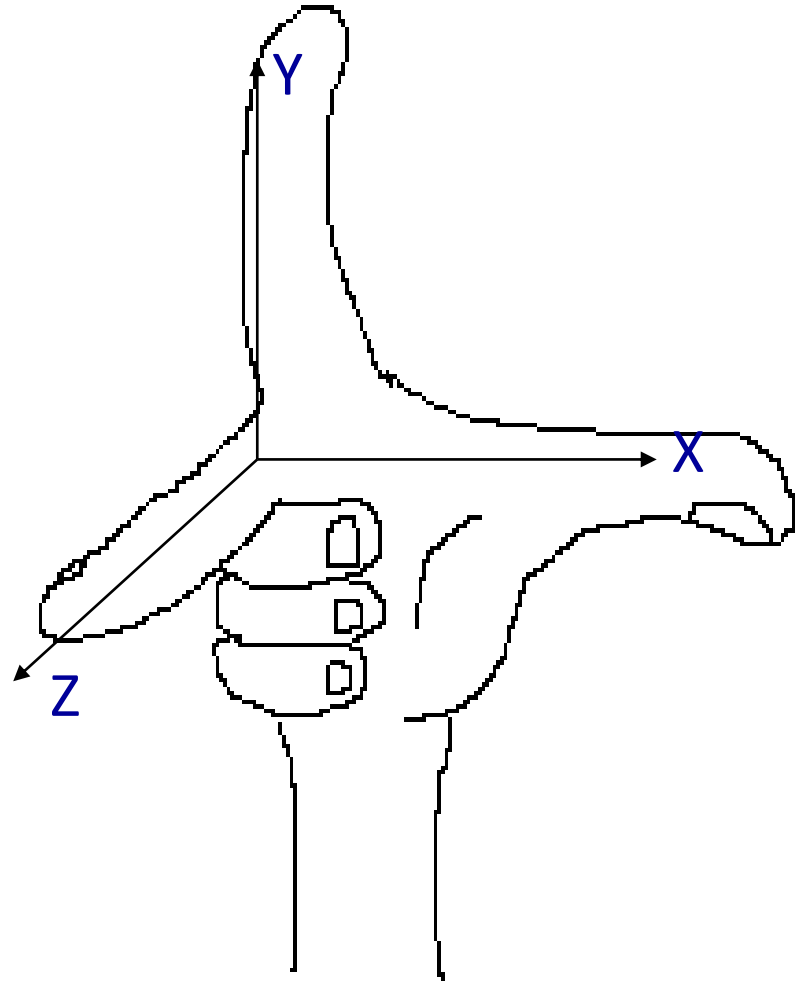
Conceitos-chave: **Sistema de Coordenadas**

■ Sistema de Coordenadas

→ VRML usa sistema Cartesiano 3D










→ Utiliza regra da mão direita (RMD)

- +X para a direita, -X para a esquerda
- +Y para cima, -Y para baixo
- +Z para perto, -Z para longe



Conceitos-chave: Cores

- Sistema RGB
- Cada cor básica é representada por número decimal de 0 até 1
- A cor é obtida através da combinação das três cores básicas
 - Red
 - Green
 - Blue

Nome	R	G	B	Cor
vermelho	1	0	0	
verde	0	1	0	
azul	0	0	1	
preto	0	0	0	
branco	1	1	1	
amarelo	1	1	0	
magenta	1	0	1	
ciano	0	1	1	
cinza médio	0.5	0.5	0.5	

Primitivas VRML

- Primitivas Básicas
- Primitivas Avançadas

Atenção:

Nas páginas a seguir, a especificação dos nós VRML não contém, necessariamente, todos os atributos e opções. Por motivos didáticos, apenas as principais características foram selecionadas para discussão. Para uma referência completa dos nós VRML, veja a especificação ISO em:

<http://www.web3d.org/x3d/specifications/vrml/>

Primitivas Básicas: Formas

■ Shape {

→ Geometry

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

→ Appearance

- material
- texture
- textureTransform

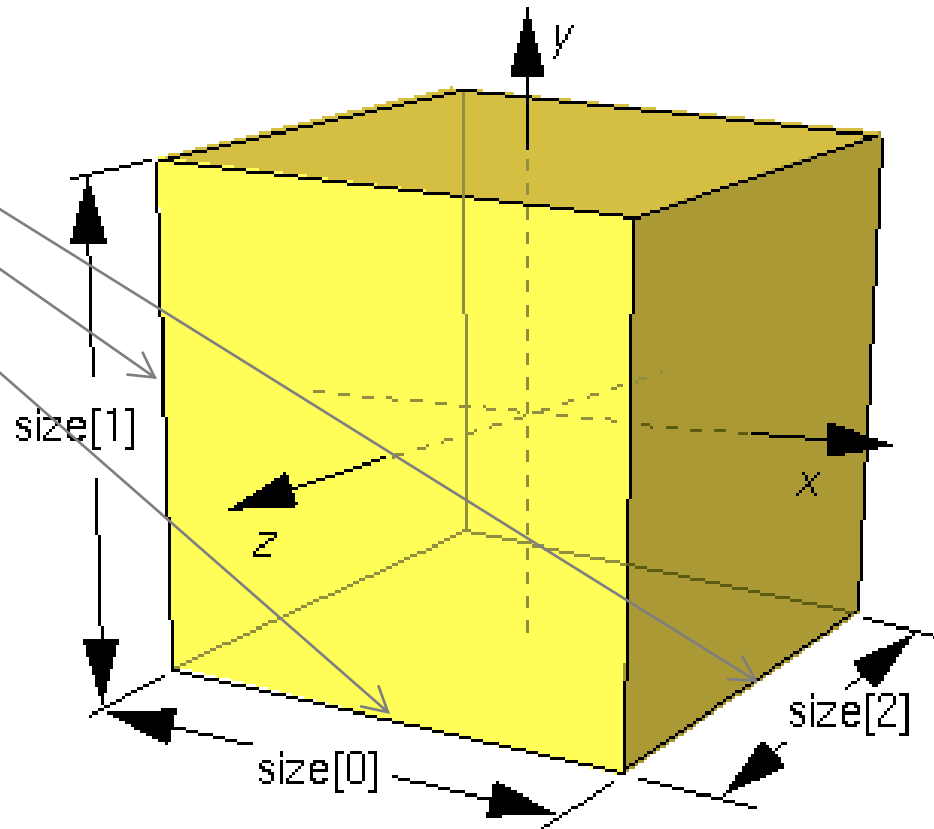
}

Caixa: Box

```
Box {  x  y  z  
      size 2 2 2  
}
```

Valores por omissão

Centro = (0, 0, 0)

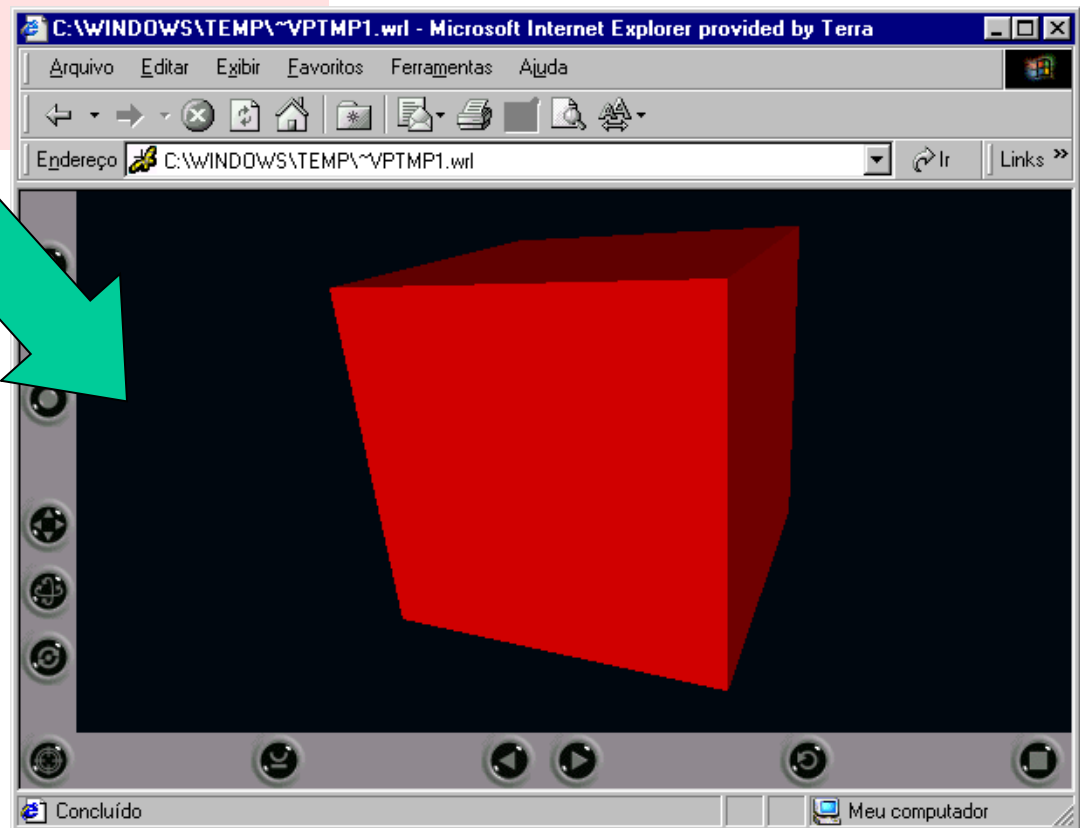


Caixa: Box

```
#VRML V2.0 utf8
Shape {
  geometry Box {}
  appearance Appearance {
    material Material {
      diffuseColor 1 0 0
    }
  }
}
```

Valores por omissão: 2 2 2

Aparência: cor vermelha

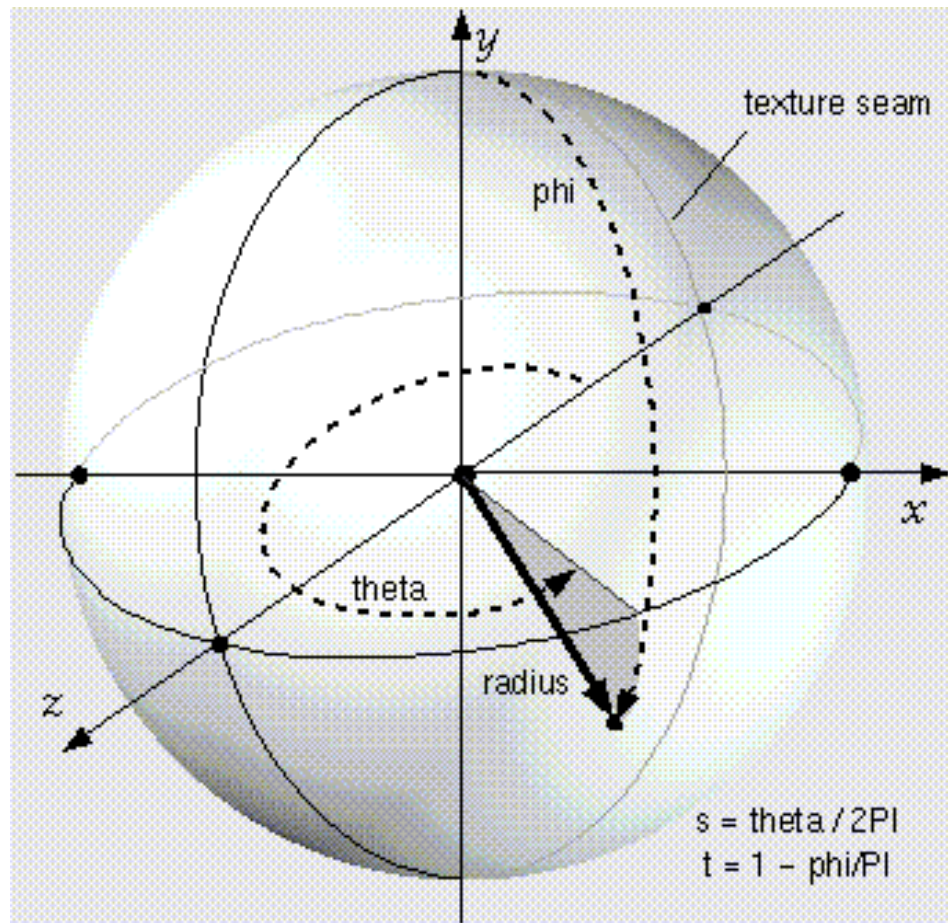


Esfera: Sphere

```
Sphere {  
    radius 1  
}
```

Valor por omissão

Centro = (0, 0, 0)

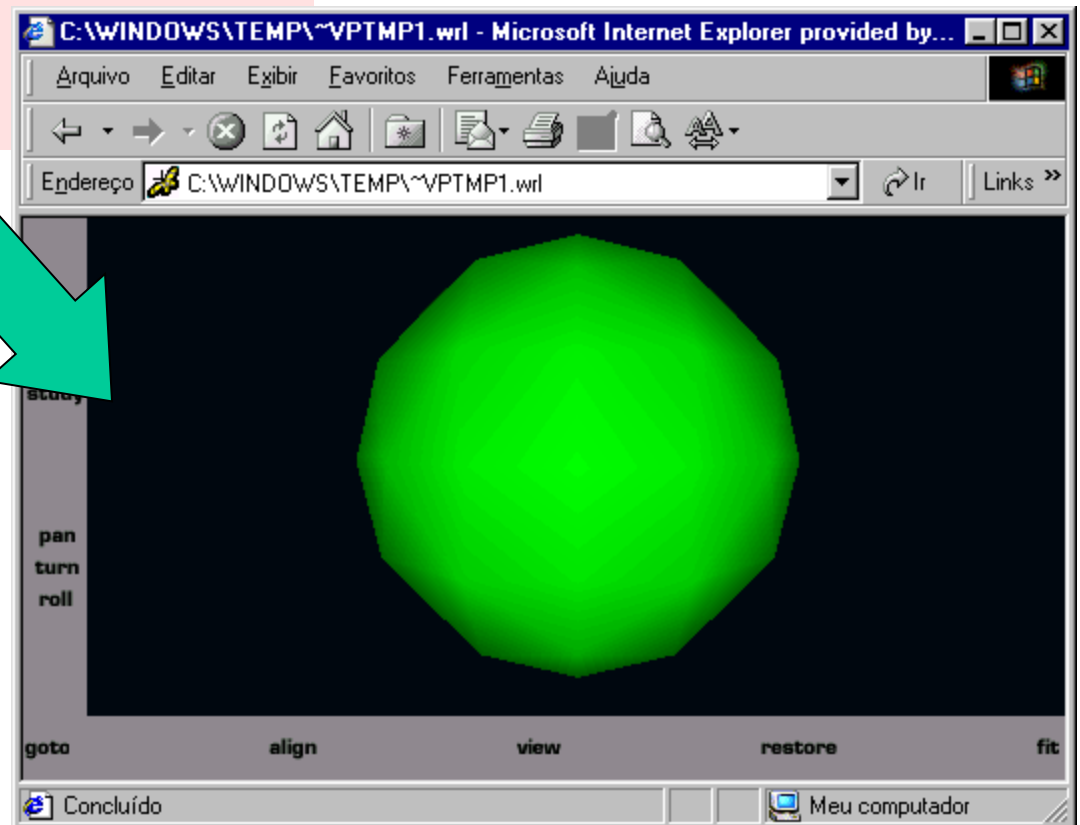


Esfera: Sphere

```
#VRML V2.0 utf8
Shape {
  geometry Sphere {}
  appearance Appearance {
    material Material {
      diffuseColor 0 1 0
    }
  }
}
```

Raio por omissão: 1

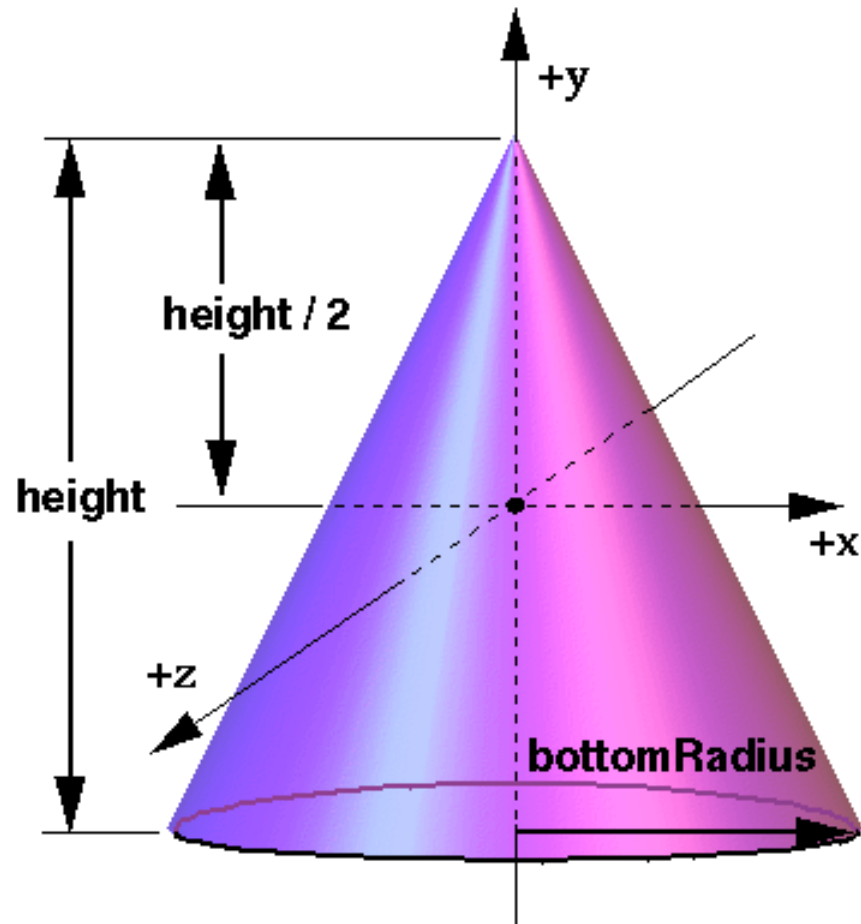
Aparência: cor verde



Cone: Cone

```
Cone {  
  bottomRadius 1  
  height 2  
  side TRUE  
  bottom TRUE  
}
```

Valores por omissão

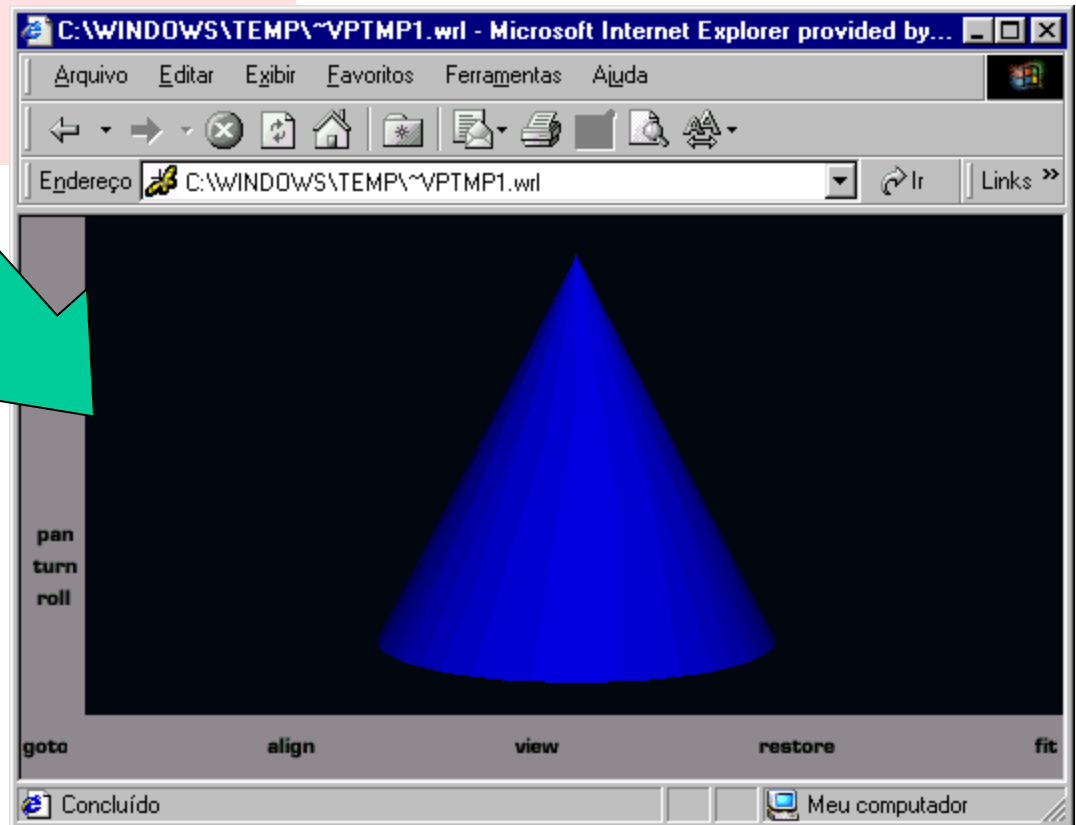


Cone: Cone

```
#VRML V2.0 utf8
Shape {
  geometry Cone {}
  appearance Appearance {
    material Material {
      diffuseColor 0 0 1
    }
  }
}
```

Valores por omissão

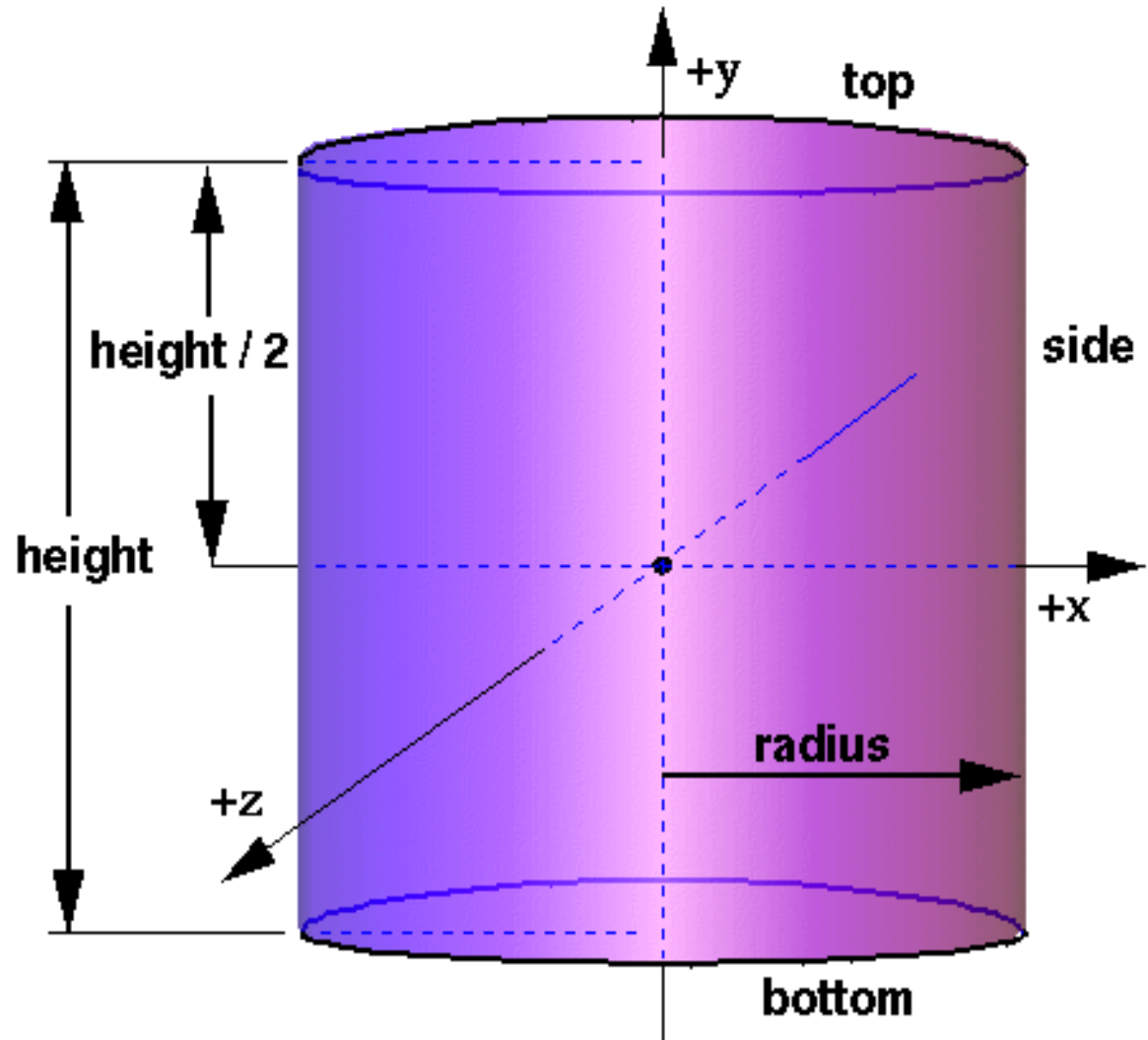
Aparência: cor azul



Cilindro: Cylinder

```
Cylinder {  
  bottom TRUE  
  height 2  
  radius 1  
  side TRUE  
  top TRUE  
}
```

Valores por omissão

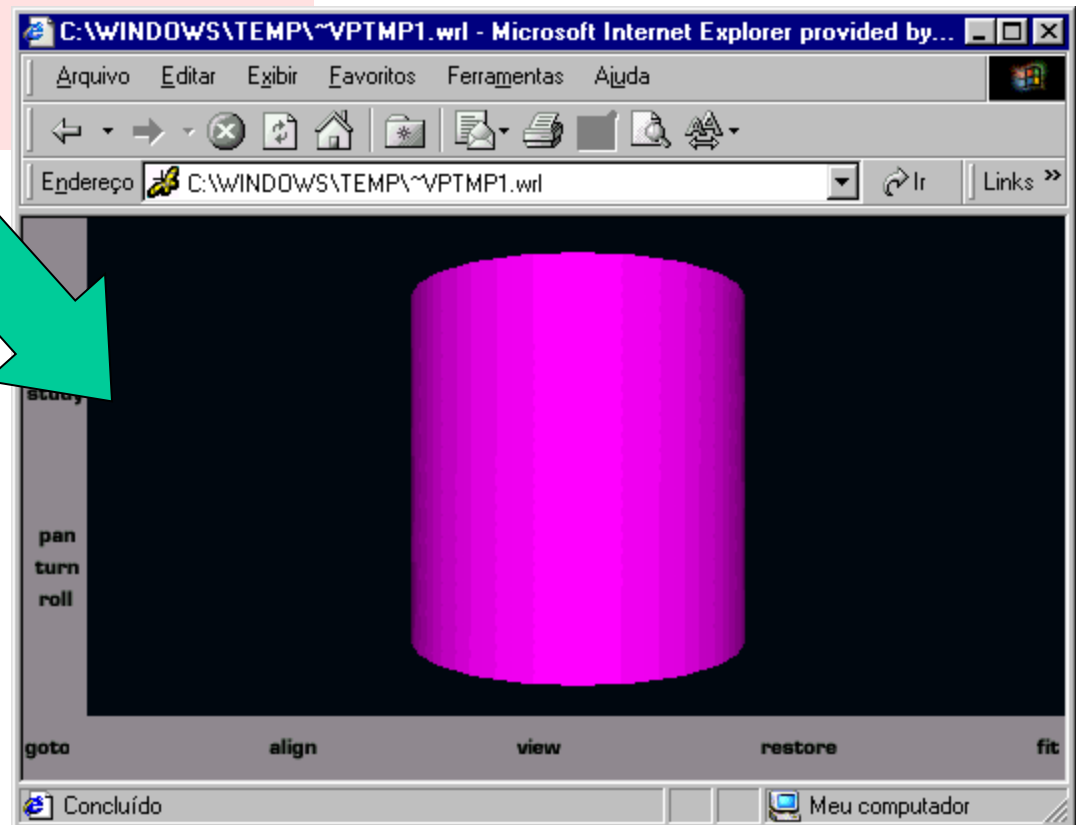
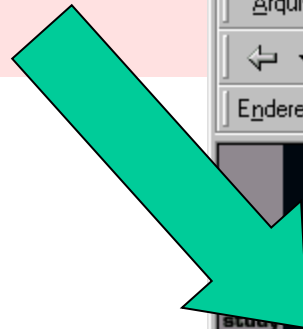


Cilindro: Cylinder

```
#VRML V2.0 utf8
Shape {
  geometry Cylinder {}
  appearance Appearance {
    material Material {
      diffuseColor 1 0 1
    }
  }
}
```

Valores default

Aparência: cor magenta



Texto: Text

```
Text {  
    string      []  
    fontStyle   NULL  
    length      []  
    maxExtent   0.0  
}
```

- Posição inicial $Z = 0$
- **Fontstyle**
 - Nó que especifica detalhes da fonte usada
- **Length**
 - indica o tamanho (pode encolher ou alargar o do texto)

maxExtent

m
a
x
E
x
t
e
n
t

Estilo da fonte: `fontStyle`

```
fontStyle {  
  family           "SERIF"  
  horizontal       TRUE  
  justify          "BEGIN"  
  language         ""  
  leftToRight      TRUE  
  topToBottom      TRUE  
  size             1.0  
  spacing          1.0  
  style            "PLAIN"  
}
```

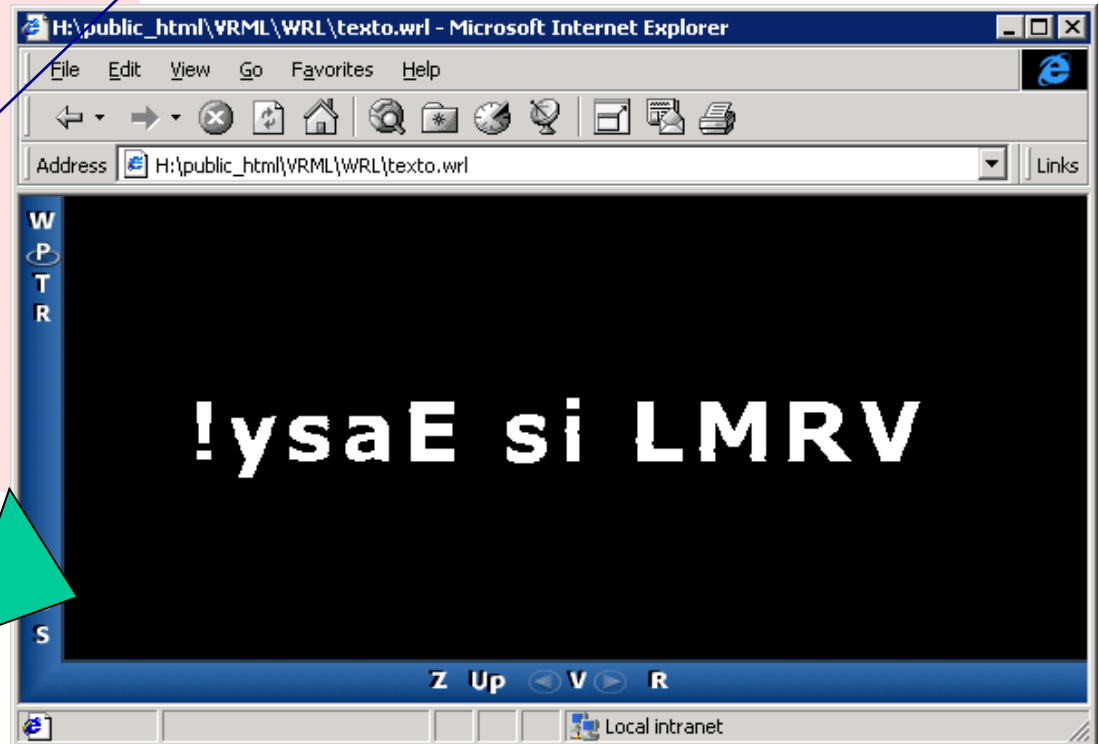
Texto: Text

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

```
Shape{  
  geometry Text {  
    string      "VRML is Easy!"  
    length      30  
    fontStyle FontStyle {  
      family      "VERDANA"  
      horizontal   TRUE  
      justify      "BEGIN"  
      language     ""  
      leftToRight  FALSE  
      size         4.0  
      spacing      1.0  
      style        "BOLD"  
      topToBottom  TRUE  
    }  
  }  
}
```

fontStyle Opcional

Esquerda para a direita



Primitivas Avançadas

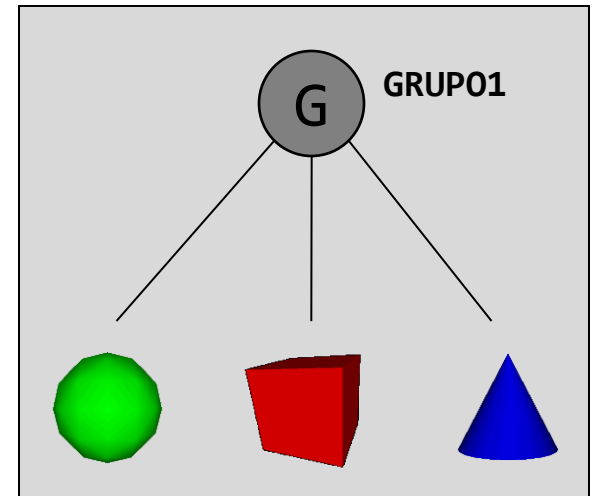
- **Group**
- **Transform**
- Viewpoint
- Achor
- Inline
- WorldInfo
- Background

Grupo: Group

```
Group {  
    children [ nós internos ]  
}
```

- Agrupa nós formando uma hierarquia
- Podemos usar **DEF** para nomear e usar depois (**USE**)

```
#VRML V2.0 utf8  
DEF GRUPO1 Group {  
    children [  
        Shape { geometry Sphere {...} }  
        Shape { geometry Box {...} }  
        Shape { geometry Cone {...} }  
    ]  
    bboxSize -1.0 -1.0 -1.0  
    bboxCenter 0.0 0.0 0.0  
    addChildren  
    removeChildren  
}
```



Transformações Geométricas: Transform

```
Transform {  
  children      [ ]  
  rotation      0 0 1 0  
  scale         1 1 1  
  translation   0 0 0  
}
```

- children pode conter vários nós
 - Transformações afectam todos os filhos
- rotation indica o eixo e o ângulo (em radianos)
 - Ex: 1 0 0 0.785 (rotação em torno do eixo do XX de $\pi/4$ graus)
- scale define a escala em cada eixo
- translation é usado para mover objectos
 - translation 20 0 0 $X=X+20$
 - translation 0 -1 -10 $X=X, Y=Y-1, Z=Z-10$

Transformações Geométricas: Transform

■ Criar objectos em determinado lugar

→ Usar `no transform`

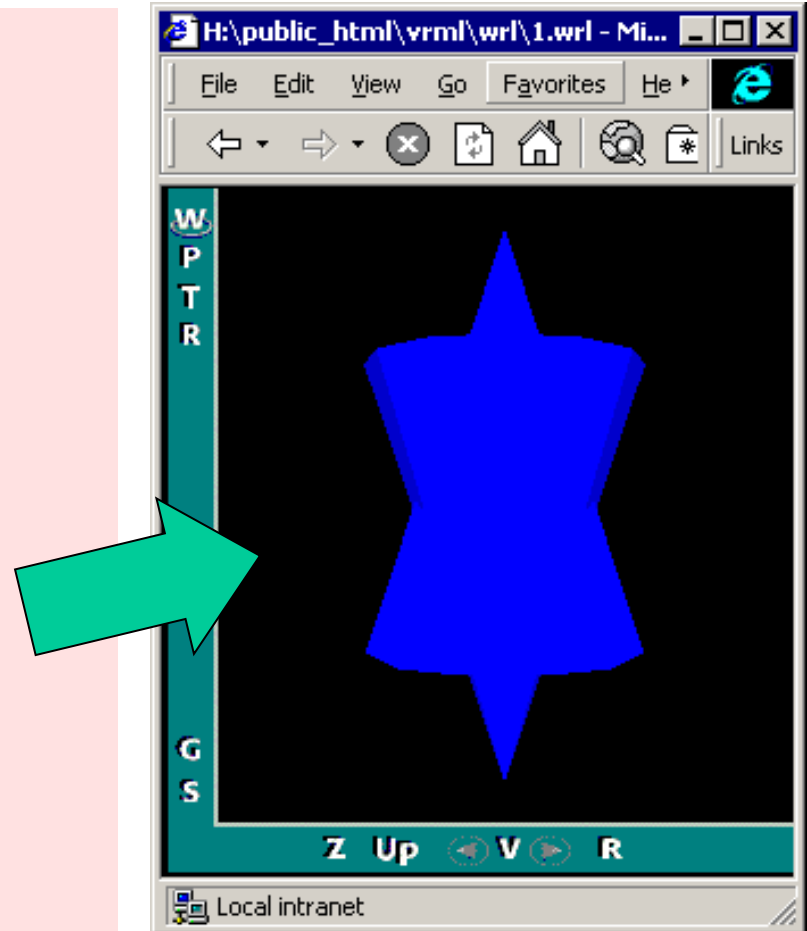
```
#VRML V2.0 utf8
Transform {
  translation 10 10 10
  children [
    Shape { geometry Sphere {} }
  ]
}
```

Transformações Geométricas: Transform

■ Rodar objectos

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

```
DEF CONE Shape {  
  geometry Cone { height 3 }  
  appearance Appearance {  
    material Material {  
      diffuseColor 0 0 1  
    }  
  }  
}  
  
Transform {  
  # 180 graus em X  
  rotation 1 0 0 3.14159  
  translation 0 -1 0  
  children USE CONE  
}
```



Definição de novos objectos

■ Usa-se o nó **PROTO**

```
PROTO nome
[
  tipoAcesso tipo parâmetro1 valor
]
{
  objectoPrototipado
  {
    campoParametrizado IS parâmetro1
  }
}
```

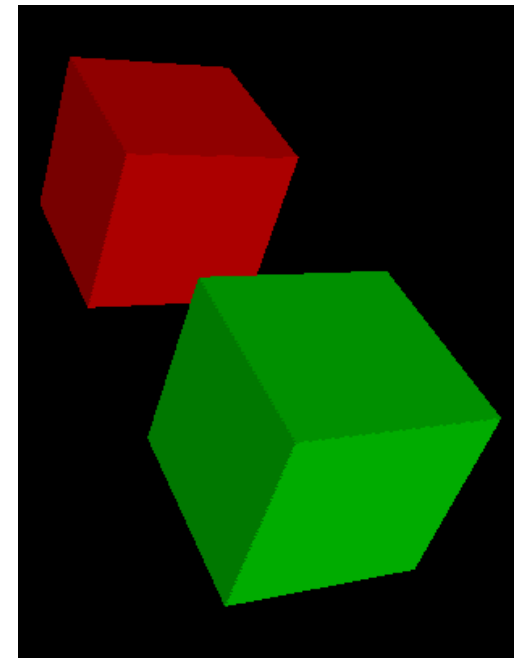
- Incorpora objectos já construídos e faz modificações através da passagem de parâmetro a usar na cópia
- **NOTA:** Para agrupar nós tem que se usar o nó **Group**

Reutilizar e modificar objectos

cont.

```
#VRML V2.0 utf8
#Criação de protótipo
PROTO myBox [ field SFCOLOR boxColour 1 0 0 ] {
  Shape {
    appearance Appearance {
      material Material {
        diffuseColor IS boxColour
      }
    }
    geometry Box { }
  }
}
#Caixa vermelha: usar valor por omissão
Transform {
  translation -2 0 0
  children myBox { }
}
#Caixa com nova cor passada como parâmetro
Transform {
  translation 2 0 0
  children myBox { boxColour 0 1 0 }
}
```

■ Exemplo



Referências

- **Norma VRML97 (VRML2.0)**

- ➔ <http://www.web3d.org/x3d/specifications/vrml/ISO-IEC-14772-VRML97/>

- **Introduction to VRML 97 (David R. Nadeau)**

- ➔ http://www.siggraph.org/education/materials/siggraph_courses/S98/18/vrml97/slides/mt0000.htm

- **Floppy's VRML 97 Tutorial**

- ➔ <http://web3d.vapourtech.com/tutorials/vrml97/>

- **Web 3D Consortium – VRML Archives**

- ➔ <http://www.web3d.org/x3d/vrml/index.html>

- **IDE: VRMLPad**

- ➔ <http://www.parallelgraphics.com/products/vrmlpad/>

- **Cliente: *plugin* para o *Browser (cortona)***

- ➔ <http://www.parallelgraphics.com/products/cortona/>