VRML
Virtual Reality Modeling Language

Luís Manuel Borges Gouveia

Fevereiro de 1998

Objectivo da apresentação:

- introduzir o VRML 2.0 como linguagem de especificação de cenas 3D

Pre-requisitos

- conhecimento de princípios de programação
- conhecimento do paradigma de programação orientada a objectos
- conhecimentos básicos de geometria analítica
- conhecimentos básicos sobre a Internet e HTML

Para mais informação sobre a especificação do VRML 2.0 consultar:

http://vag.vrml.org/VRML2.0/FINAL/

Luís Manuel Borges Gouveia

lmbg@ufp.pt

http://www.ufp.pt/staf/lmbg/

VRML
Virtual Reality Modeling Language

- especificação de um espaço tri-dimensional:
  - construção de cenários correspondentes a uma dada realidade;
  - cenários completamente imaginários;
  - representação de conceitos ou visualização
- o navegador VRML age como um interpretador dos ficheiros ASCII que especificam a cena VRML a gerar
- o espaço tri-dimensional é percorrido pelo utilizador, com o navegador VRML a regenerar os conteúdos da representação visual a tempo real, representando a cena do ponto de vista do utilizador.

O VRML - Virtual Reality Modeling Language - permite a especificação de descrição de um espaço tri-dimensional, possibilitando a construção de cenários correspondentes a uma dada realidade ou completamente imaginários, representando inclusive, conceitos.

Um navegador (browser) que entenda VRML age como um interpretador dos ficheiros ASCII que especificam a cena VRML a gerar. O navegador gera em tempo real a representação visual equivalente à descrição textual da cena.

À medida que o espaço tri-dimensional é percorrido pelo utilizador no interface (navegador VRML), este software deve regenerar os conteúdos da representação visual a tempo real de modo a representar a cena do ponto de vista do utilizador.

O VRML permite a construção de mundos tri-dimensionais usando um formato que pode ser considerado uma extensão gráfica da metodologia de programação introduzida em linguagens como o Smalltalk, o C++ e o JAVA.

VRML 2.0

- A identificação de ficheiros ASCII com formato VRML 2.0 é dada pela extensão *.wrl*
- ...e no seu conteúdo, deve possuir uma primeira linha, conhecida como cabeçalho do ficheiro, com o seguinte formato:

  #VRML V2.0 utf8

  utf8 indica a norma internacional unicode para a representação de caracteres da estrutura Text

  outras linhas que começem por #, são consideradas comentários e ignoradas pelo interpretador VRML

VRML 2.0

- Após o cabeçalho, uma ficheiro VRML pode conter uma qualquer número de vários elementos (“nodes”):
  - geometria
  - luz
  - agrupamento
  - outros nodos (descrições de elementos discretos da cena)
  - controlo de eventos (para criação de comportamentos e interacção)

um exemplo: esfera.wrl

```
#VRML V2.0 utf8
#esfera.wrl
Shape {
   appearance Appearance {
     material Material {
       emissiveColor  0 0 1
     }
   }
   geometry Sphere {
      radius 10
   }
}
```

O exemplo apresentado constroi uma esfera azul uniformemente iluminada. Notem-se os valores para emmisiveColor e radius

No caso do argumento para emmisiveColor, com o valor 0 0 1, define-se a forma de reflexão de cor do material da superficie da geometria especificada. Neste caso 0% de luz vermelha e verde, respectivamente e 100% de luz azul (a definição de cores é realizada no formato RGB, em percentagem).

O argumento 10, de radius fornece a dimensão da esfera, em unidade de radianos.

VRML como linguagem oo

- Possui uma definição de formato para a descrição de objectos num espaço 3D, por nodos (em Inglês *nodes*).
  - um nodo corresponde a um objecto em C++ ou JAVA. Pode-se considerar um nodo, uma classe de onde são derivados as geometrias Box, Sphere, Sound, SpotLight, entre outras.
  - cada nodo possui propriedades comuns como uma designação de tipo, valores por defeito de campos e a capacidade de enviar e receber mensagens (acontecimentos no VRML 2.0) que alteram valores de campos.
  - quando se instancia uma classe, podem-se alterar os valores por defeito do mesmo modo que se especificam os parâmetros para uma instância de classe no C++. *Um dos benefícios do VRML é que quando se instancia um nodo, geralmente é obtido um resultado visual tangível*

O VRML é uma linguagem orientada a objectos que oferece uma definição de formato de ficheiro para a descrição de objectos num espaço 3D, designados por nodos (em Inglês *nodes*).

Um nodo corresponde a um objecto em C++ ou JAVA. Pode-se considerar um nodo, uma classe de onde são derivados Box, Sphere, Sound, SpotLight, entre outras.

Cada objecto nodo possui propriedades comuns como uma designação de tipo, valores, por defeito, de campos e a capacidade de enviar e receber mensagens (acontecimentos no VRML 2.0) que alteram valores de campos.

Quando se instancia uma classe, podem-se alterar os valores por defeito do mesmo modo que se especificam os parâmetros para uma instância de classe no C++. Um dos benefícios do VRML é que quando se instancia um nodo, geralmente é obtido um resultado visual tangível

O VRML possui muitos nodos pré-definidos tais como uma biblioteca de objectos de onde os elementos de uma cena criada podem herdar características. Permite igualmente a derivação e utilização de nodos originais por prototipagem.

VRML como linguagem oo

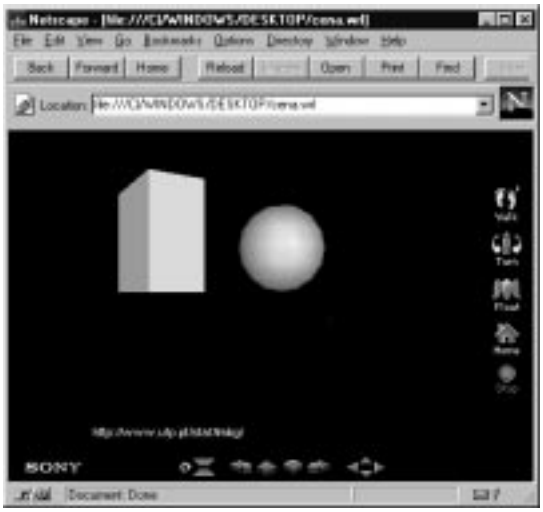
- O VRML possui muitos nodos pré-definidos tais como uma biblioteca de objectos de onde os elementos de uma cena criada pode herdar características.
- Permite a derivação e utilização de nodos originais por protótipagem

Agrupamento de nodos

- É possível agrupar os nodos num grafo de cena VRML - scene graph
  - permite organizar o “layout” e a funcionalidade do mundo virtual
  - hierarquia organizada em árvore com a raiz designada por *trunk*, e com os níveis seguintes organizados por ramos com os nodos designados por *children*
  - o nodo filho herda as propriedades como a posição e orientação do respectivo nodo pai numa ordem de hierarquia “objecto para o mundo” (grafo de cena)

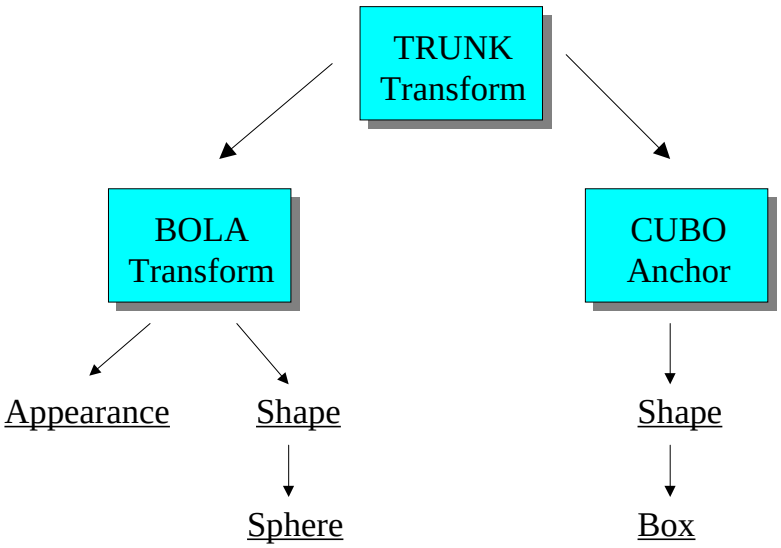
Exemplo de agrupamento de nodos

```
#VRML 2.0 utf8
DEF TRUNK Transform {
#nodo pai no grafo
      translation 0.0 1.0 0.0
      rotation 0.0 1.0 0.0 0.39
      children [
         DEF BOLA Transform {
#filho do nodo TRUNK Transform
            translation 3.0 0.0 0.0
             children [
              Shape {
          ....
                geometry Sphere {radius 1 }
          ....
          DEF CUBO Anchor {
              url [ "http://www.ufp.pt/…" ]
              children [
                 Shape {
                    geometry Box {
          ....
```



Comentários ao progrma VRML:
DEF é a palavra chave para adicionar o nome ao nodo (DEF BOLA e DEF CUBO permitem acrescentar a BOLA e o CUBO à raiz TRUNK).

Visualização do grafo de cena




```
#VRML 2.0 utf8
#cena.wrl - listagem completa
DEF TRUNK Transform {      #nodo pai no grafo
      translation 0.0 1.0 0.0      #todos os filhos são deslocados 1 metro em Y
      rotation 0.0 1.0 0.0 0.39    #todos os filhos são rodados 22.5 graus sobre Y
      children [
         DEF BOLA Transform {     #filho do nodo TRUNK Transform
               translation 3.0 0.0 0.0   #a esfera é movimentada 3 unidades em X
               children [
                    Shape {
                      appearance Appearance {
                          material Material {
                            emissiveColor 0 0 1
                          }
                       }
                       geometry Sphere {
                          radius 1
                       }
                    }
               ]
          }
         DEF CUBO Anchor {
              url [ "http://www.ufp.pt/staf/lmbg/" ]
              children [
                 Shape {
                    geometry Box {
                          size 1 2 2
                    }
                 }
              ]
         }
]
}
```

Os campos dos nodos

- parâmetros ou palavras chave com os seus valores a descreverem as propriedades de cada nodo
- divididos em duas categorias:
  - campos de atributos; permitem alterar as propriedades dos nodos instanciados (campo ambientIntensity de um Spotlight ou o radius de uma esfera);
  - campos de ligação que permitem a passagem de uma descrição de um nodo como um valor de propriedade.

Os campos dos nodos são parâmetros ou palavras chave com os seus valores a descreverem as propriedades de cada objecto nodo.

Os campos podem ser divididos em duas categorias:

- campos de atributos, que permitem alterar as propriedades dos nodos instanciados como, por exemplo, o campo ambientIntensity de um Spotlight ou o radius de uma esfera;
- campos de ligação que permitem a passagem de uma descrição de um nodo como um valor de propriedade. Um exemplo é o seguinte nodo:

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

…em que existem dois campos de ligação, o campo geometry, que aceita um nodo de geometria como argumento (Sphere, Box ou Cone) e o campo appearance que aceita a descrição do nodo Appearance.

Campos de ligação

Um exemplo é o seguinte nodo:

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

…em que existem dois campos de ligação, o campo *geometry*, que aceita um nodo de geometria como argumento (Sphere, Box ou Cone) e o campo *appearance* que aceita a descrição do nodo Appearance.

Alteração de valores dos campos

- Muitos dos campos, incluindo os de ligação, de um dado nodo são **expostos** de modo a ser possível a alteração dos seus valores no momento de visualização, através do disparo de eventos tais como sensores e scripts.
- Outros campos são **privados** e os seus valores apenas podem ser definidos na criação da cena 3D

Muitos dos campos, incluindo os de ligação, de um dado nodo são expostos de modo a ser possível a alteração dos seus valores na altura de visualização, através do disparo de eventos tais como sensores e scripts. Outros campos são privados e os seus valores apenas podem ser definidos na criação da cena 3D.

Unidades standard

- As unidades no VRML estão normalizadas…
  - distância linear, em metros
  - ângulos de rotação, em radianos

- criando um nodo shape do tipo box com uma unidade de largura, uma unidade de altura e uma unidade de profundidade, define-se um cubo virtual com um metro cúbico!
- se o cubo se encontra pousado no chão e se roda 45 graus de modo a ser visto de uma posição de canto, este é rodado sobre o eixo Y Pi/4 ou 3.14/4 ou 0,785 radianos

Unidades standard - observação

- Muitos dos programas de desenho e modelação 3D possuem filtros de exportação para VRML, realizando uma conversão de um para um, para metros de qualquer que seja a unidade utilizada no modelo original
  - Se se modelar um avatar humano com 175 centrimetros, o equivalente em VRML possui 175 metros!
- Na criação de modelos é aconselhável mudar a unidade utilizada para metros logo à partida

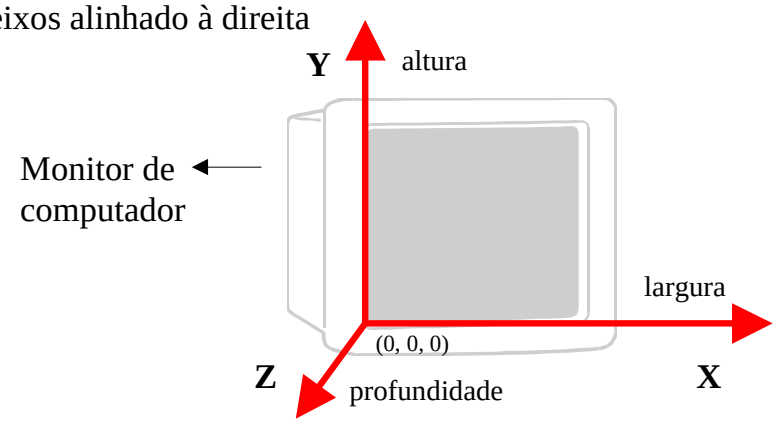
Posicionar objectos

- Num mundo VRML (cena 3D)
  - usar o campo de rotação no nodo Transform para definir a orientação do objecto

  a seguir
  - usar o campo translação para definir o número de unidades ao longo de cada eixo pretendidas para o objecto

Sistema de coordenadas

- O VRML organiza a representação bidimensional de um espaço tri-dimensional recorrendo a um sistema cartesiano de eixos alinhado à direita



Modo wireframe

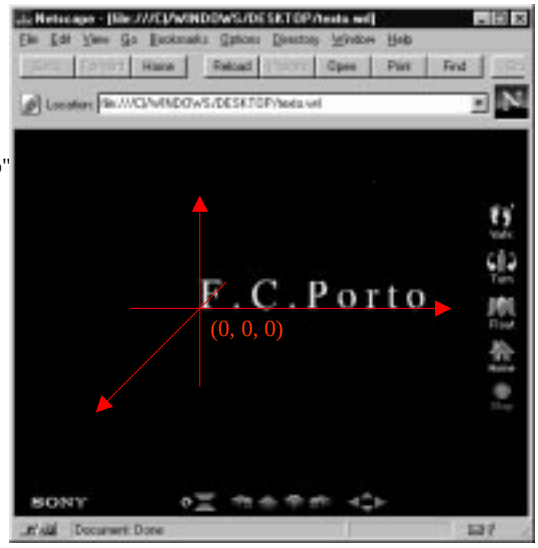
- a posição dos objectos é descrita no sistema de coordenadas, especificando pontos no espaço usando coordenadas tri-dimensionais
- estes pontos são ligados por linhas para formar uma malha, que descreve a superfície da geometria (modelo)
- representação no **modo wireframe**, em que se visualiza os pontos e linhas que descrevem o modelo

Colocação das geometrias

- sem especificar uma transformação, os nodos de geometria VRML são centrados, por defeito, nas coordenadas 0, 0, 0; correspondentes a (X, Y, Z)
- existe uma excepção que é a geometria Text: por defeito, o texto é alinhado à esquerda da posição (0, 0, 0) com cara caracter a ser colocado na direcção do eixo X, sentido positivo

Colocação de texto

```
#VRML V2.0 utf8
#texto.wrl
Shape {
geometry Text {
  exposedField  MFString string    ["F.C.Porto"
  exposedField  SFNode fontStyle  fontstyle {
              SFString family       "SANS"
              SFString language     ""
              SFFloat  size         1.0
              SFFloat  spacing    1.0
              SFString style        "BOLD"
            }
    exposedField  MFFloat  length    [5]
    }
}
```



Primitivas em VRML

- nodos base para construção de cenas 3D
  - exemplos são: **Box**, **Sphere**, **Cone**, **Cylinder**
- quando se constroi um modelo e se exporta este para VRML, o objecto aparece como um nodo **IndexedFaceSet**, isto é, como um conjunto de polígonos

Modelos exportados para o VRML

- cada vértice contido na malha do conjunto de polígonos IndexedFaceSet é descrito por um ponto tridimensional no nodo filho Coordinate
- o nodo *coordIndex* de IndexedFaceSet define o agrupamento e a ordem do conjunto de pontos que definem a face

```
IndexedFaceSet {
  eventIn       MFInt32 set_colorIndex
  eventIn       MFInt32 set_coordIndex
  eventIn       MFInt32 set_normalIndex
  eventIn       MFInt32 set_texCoordIndex
  exposedField  SFNode  color          NULL
  exposedField  SFNode  coord          NULL #ver o nodo coordinate!
  exposedField  SFNode  normal         NULL
  exposedField  SFNode  texCoord       NULL
  field         SFBool  ccw            TRUE
  field         MFInt32 colorIndex     [ ]
  field         SFBool  colorPerVertex TRUE
  field         SFBool  convex         TRUE
  field         MFInt32 coordIndex     [ ]
  field         SFFloat creaseAngle    0
  field         MFInt32 normalIndex    [ ]
  field         SFBool  normalPerVertex TRUE
  field         SFBool  solid          TRUE
  field         MFInt32 texCoordIndex  [ ]
}
```

Nodo filho coordinate, que define um conjunto de coordenadas 3D

```
Coordinate {
  exposedField MFVec3f point  [ ]
}
```

Visualização dos objectos

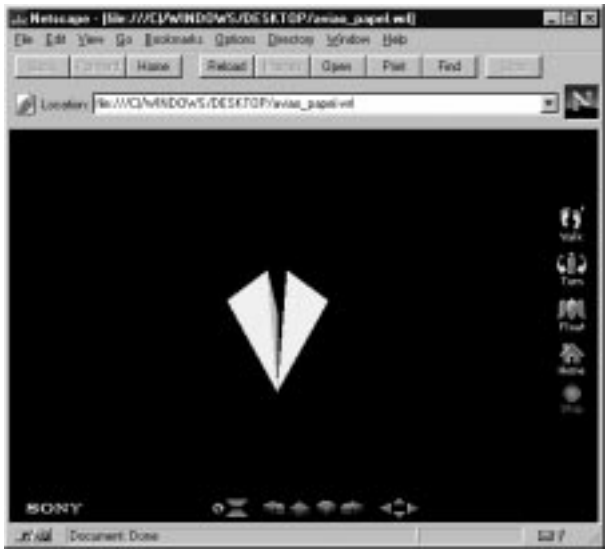
- normal: um vector que se estende da superfície de um polígono para o seu exterior
- o lado do polígono de onde a normal é projectada é a face que o navegador VRML visualiza; o lado oposto desta face não existe
- para determinar a direção da normal de um polígono, é verificado se os pontos listados no conjunto Coordinate aparecem na ordem contrária aos dos ponteiros do relógio de acordo com a ordem listada no coordIndex

Visualização dos objectos

- se se enrolar os dedos da mão direita ao redor dos pontos, na ordem em que estão listados no coordIndex, o dedo polegar aponta na direcção da face normal, isto é, a direcção na qual a luz é refletida pelo polígono
- se se encontrar uma face cuja normal é dirigida para o lado contrário, esse polígono não é visualizado no navegador
  - para corrigir o problema, alteram-se as normais, invertendo os valores no coordIndex para cada face afetada
  - se todas as normais das faces dos polígonos se encontram direccionados para o seu interior, então é mais simples colocar o campo ccw do nodo IndexedFaceSet com o valor FALSE

Exemplo de visualização

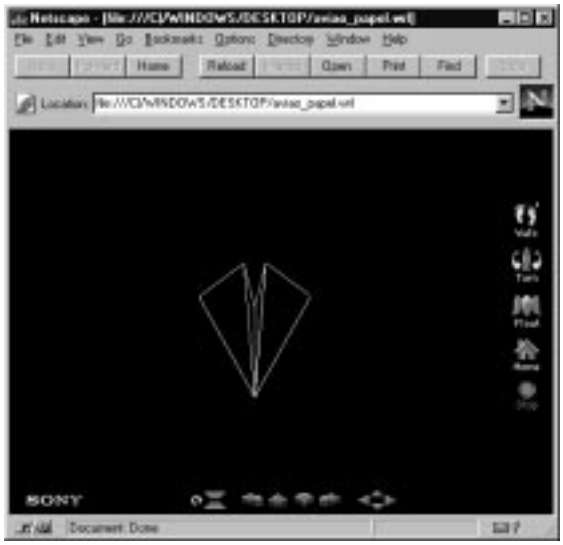
```
#VRML V2.0 utf8
Shape {
...
geometry IndexedFaceSet {
  coord Coordinate {
    point [ 0.0  0.0 -0.5,
           -0.5  0.1  0.5,
           -0.1  0.4  0.5,
            0.0 -0.8  0.5,
            0.1  0.4  0.5,
            0.5  0.1  0.5 ]
    }
    coordIndex [ 3, 1, 2, -1,
                 3, 2, 0, -1,
                 3, 4, 0, -1,
                 4, 5, 3, -1 ]
    normalPerVertex FALSE
    ccw FALSE
...
```




```
#VRML V2.0 utf8
#aviao_papel.wrl
Shape {
  appearance Appearance {
  material Material {
    diffuseColor  1.0 1.0 1.0
    emissiveColor 0.5 0.5 0.5
    shininess 5
    }
  }
  geometry IndexedFaceSet {
    coord Coordinate {
      point [ 0.0  0.0 -0.5,
            -0.5  0.1  0.5,
            -0.1  0.4  0.5,
             0.0 -0.8  0.5,
             0.1  0.4  0.5,
             0.5  0.1  0.5 ]
    }
    coordIndex [ 3, 1, 2, -1,
                 3, 2, 0, -1,     #esta face deve ser alterada para 0, 2, 3
                 3, 4, 0, -1,     #-1 é o delimitador
                 4, 5, 3, -1 ]
    normalPerVertex FALSE   #visualiza ângulos apertados entre faces
    ccw FALSE      #pontos coordindex listados na ordem dos ponteiros do relógio
    #solid FALSE   #as faces são definidas como possuindo dois lados, se activado
  }
}
```

Exemplo de visualização

- o exemplo anterior mostra que a face interior direita não foi visualizada
- com a opção *wireframe* do navegador VRML, é possível verificar que essa face está presente na cena 3D



Exemplo de visualização

- O problema é resolvido, realizando a modificação de:

  coordIndex [3, 1, 2, -1,
  3, 2, 0, -1,
  3, 4, 0, -1,
  4, 5, 3, -1]

- para

  coordIndex [3, 1, 2, -1,
  0, 2, 3, -1,
  3, 4, 0, -1,
  4, 5, 3, -1]

Translação linear em três dimensões e outras transformações

- um nodo de agrupamento geral que serve de encapsulamento para muitos outros objectos e é um elemento básico do VRML: **Transform**
  - possui grande funcionalidade
  - transformação é o termo genérico para movimento em computação gráfica (escalamento, rotação ou translação)
- um nodo Transform em VRML pode aplicar uma transformação a um grupo de objectos simultaneamente

O nodo Transform

- Listagem dos campos do nodo Transform com os respectivos valores por defeito

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

```
Transform {
    eventIn       MFNode     addChildren
    eventIn       MFNode     removeChildren
    exposedField  SFVec3f    center           0 0 0
    exposedField  MFNode     children         [ ]
    exposedField  SFRotation rotation         0 0 1  0
    exposedField  SFVec3f    scale            1 1 1
    exposedField  SFRotation scaleOrientation 0 0 1  0
    exposedField  SFVec3f    translation      0 0 0
    field         SFVec3f    bboxCenter       0 0 0
    field         SFVec3f    bboxSize         -1 -1 -1
}
```

Significado matemático do nodo Transform

- dado um ponto **P**, tri-dimensional e nodo Transform, P é transformado no ponto **P'** no sistema de coordenadas "herdado", numa série de transformações intermédias.
- utilizando uma notação de matrizes, com **C** (center), **SR** (scaleOrientation), **T** (translation), **R** (rotation), and **S** (scale) como matrizes de transformação equivalentes, temos a seguinte equação:

  P' = T . C . R . SR . S . -SR . -TC . P

  com P como vector coluna

O nodo transform:

```
Transform {
   center          C
   rotation        R
   scale           S
   scaleOrientation SR
   translation     T
   children        [...]
}
```

é equivalente à sequência de nodos indentados:

```
Transform { translation T
   Transform { translation C
      Transform { rotation R
         Transform { rotation SR
            Transform { scale S
               Transform { rotation -SR
                  Transform { translation -C
                     ...
}}}}}}}
```

O nodo Group versus Transform

- Group é o nodo de agrupamento que oferece a mesma funcionalidade de encapsulamento que o nodo Transform, mas que não possui a capacidade de operar nos filhos
- pode-se alcançar a retenção de hierarquia do Group usando o nodo Transform e desta forma conseguir também a habilidade para adicionar facilmente uma transformação posterior; é assim mais benéfico agrupar as hierarquias de cena com Transform do que com Group

Agrupamento de nodos

- Um grafo de cena pode conter um número qualquer de nodos, que definam uma hierarquia válida para manipulação da cena
- pode-se agrupar Transform com outros nodos de grupo (um exemplo é a colocação de um sensor a um dado nível, na hierarquia de um objecto, mantendo a facilidade de manipular o grupo como um todo)

Relação de dependência

- Os valores dos campos *Transform* (como a *translation* ou *rotation*) são herdados por qualquer nodo *Shape* ou outro filho do campo *children* de *Transform*
- A colocação do campo *children*, assinala que os que os próximos nodos são subsidiários do nodo de agrupamento pai, estabelecendo desta forma uma relação de dependência

Um exemplo

- o exemplo, mostra um nodo *Shape* com a geometria a *box*, que representa um cubo
- a utilização do nodo *Transform* como pai, permite a translação, rotação e escala do cubo
- no exemplo, o cubo é movimentado da origem 3 metros para a esquerdo, 2 metros em relação à altura da origem e aproximado 1 metro (o nodo *appearance* apenas foi colocado para melhorar o aspecto do cubo)

```
#VRML V2.0 utf8
#movi_cubo.wrl
DEF transformarCUBO Transform {
      translation  -3.0  2.0  1.0
      children [
        DEF CUBO Shape {
          appearance Appearance {
          material Material {
             diffuseColor  0 0 1
             specularColor 0.9 0.9 0.9
          }
        }
        geometry Box { }
      }
  ]
}
```

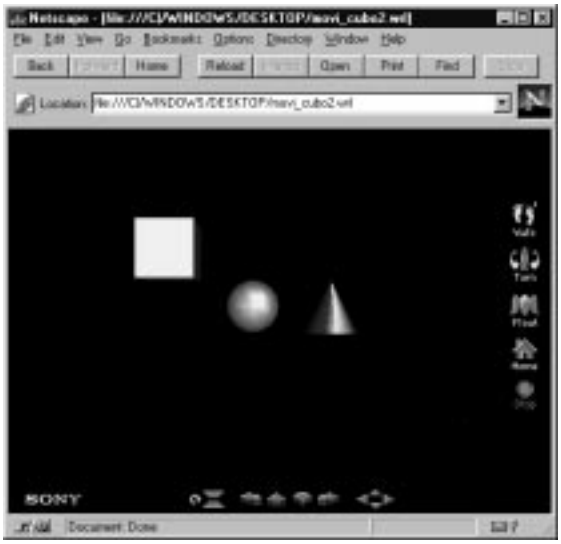
Visualização do exemplo apresentado…

Acerca das hierarquias

- sempre que se usa um nodo de grupo de qualquer tipo para organizar a cena 3D, é necessário especificar os conteúdos do nodo de grupo no campo *children*
- os nodos *children* herdam as transformações do nodo pai que define o espaço de coordenadas local para a hierarquia.
- o nodo de grupo pai, também herda as transformações dos nodos de nível superior; pelo que a translação, escalamento ou rotação locais são sempre relativas à última transformação executado no agrupamento de nível imediatamente acima, isto é, as transformações para nodos de grupo é cumulativa dentro da hierarquia

A hierarquia de transformações

- para demonstrar a hierarquia de transformações são adicionados ao exemplo anterior (movi_cubo.wrl) novas geometrias dentro e fora do nodo de grupo TransformarCubo
- o cone é definido fora da hierarquia, com os valores da translação referentes à origem
- a bola é definida dentro da hierarquia logo afectada pela posição do cubo



```
#VRML V2.0 utf8
#movi_cubo2.wrl
DEF transformarCUBO Transform {
 translation  -3.0  2.0  1.0
 children [
  DEF CUBO Shape {
   appearance Appearance {
    material Material {
     diffuseColor  1 0 0
     specularColor 0.9 0.9 0.9
    }
   }
   geometry Box { }
  }
  DEF transformarBOLA Transform {
   translation 3.0 -2.0 -1.0
   children [
    DEF BOLA Shape {
     appearance Appearance {
      material Material {
       diffuseColor  0 1 0
       specularColor 0.9 0.9 0.9
      }
     }
     geometry Sphere {}
    }
   ]
  }
  ]
}
```

```
#continuação do programa
DEF transformarCONE Transform {
 translation 3  0  0
 children [
  DEF CONE Shape {
   appearance Appearance {
    material Material {
     diffuseColor  0 0 1
     specularColor 0.9 0.9 0.9
    }
   }
   geometry Cone {}
  }
 ]
}
DirectionalLight {
 direction -1 -1 -5
 intensity 0.8
 }
```

A rotação de objectos

- a utilização de ângulos na colocação dos objectos permite a obtenção de perspectiva, fornecendo uma clara pista visual para a representação de uma cena em 3D
- o campo *rotation* aceita três valores no intervalo -1 a 1, que descrevem o vector nos eixos X, Y, Z, seguidos do valor da rotação em radianos
- os primeiros três valores do campo rotation definem um ponto no espaço 3D. Se se ligar esse ponto com a origem (do mundo ou da geometria - local) é definido um vector de rotação; o eixo em torno do qual é realizada a rotação especificada pelo quarto valor

Se se definir os três primeiros valores do campo rotation como 1, 0, 0, está a ser especificado o eixo de X como o vector de rotação para o objeto.

O valor de ângulo em radianos descreve então, em termos de Pi (aproximadamente 3.14, que é equivalente a uma rotação de 180 graus), o valor da rotação que é aplicada ao objecto no eixo anteriormente definido para rotação.

Para determinar se o ângulo de rotação deve ser positivo ou negativo, deve-se apontar seu dedo polegar da mão direita da origem para o ponto que define a direção do vector de rotação e enrolar os dedos ao redor do vector. Se os dedos enrolados apontam na direcção de rotação positiva, o valor do angulo é positivo caso contrário, se os dedos enrolados apontarem na direcção oposta, o valor em radianos é negativo.

Exemplo de rotação

- rotação do cubo com origem em <-3, 2, 1> e vector <-1, 1, 0.8>, de 45% (3,14/4 rad)
- colocação da bola for a da hierarquia do cubo e ajuste na sua translação para <0, 2, 1>
- rotação do cone com origem em <-3, 2, 1> e vector <0.5, 0.1, 0.75>, de 45% e ajuste na sua translação para <3, 2, 1>
- ajuste na direcção da luz para <-2, 1, -5>


```
#VRML V2.0 utf8
#movi_cubo3.wrl
DEF transformarCUBO Transform {
  translation  -3.0  2.0  1.0
  rotation -1, 1, 0.8, 0.785
  #eixo com origem <-3, 2, 1> e vector de
rotação <-1, 1, 0.8>
  children [
    DEF CUBO Shape {
      appearance Appearance {
        material Material {
          diffuseColor  1 0 0
          specularColor 0.9 0.9 0.9
        }
      }
      geometry Box { }
    }
  ]
}
DEF transformarBOLA Transform {
  translation 0.0 2.0 1.0
  children [
    DEF BOLA Shape {
      appearance Appearance {
      material Material {
        diffuseColor  0 1 0
        specularColor 0.9 0.9 0.9
      }
    }
    geometry Sphere {}
  }
  ]
}
```

```
DEF transformarCONE Transform {
  translation 3  2  1
  rotation 0.5 0.1 0.75 0.785
  children [
    DEF CONE Shape {
      appearance Appearance {
        material Material {
          diffuseColor  0 0 1
          specularColor 0.9 0.9 0.9
        }
      }
      geometry Cone {}
    }
  ]
}
DirectionalLight {
  direction -2 1 -5
  intensity 0.8
  }
```

A bold estão as modificações em relação a movi_cubo2.wrl

A BOLA está for a da hierarquia do CUBO

O escalamento de objectos

- na imagem do exemplo anterior, o cubo parece de maior dimensão que a esfera e o cone; algo que pode ser corrigido por escalamento
- o campo *scale* é usado para realizar uma operação de escalamento uniforme (escalamento aplicado igualmente a todos os eixos)
- também é usado para a integração de escalas de diferentes mundos numa só cena 3D (por exemplo, resultado da utilização de diferentes ferramentas de modelação)

O escalamento de objectos

- para demonstrar o escalamento, é adicionada ao exemplo anterior (movi_cubo3), depois da linha Transform, do cubo, o seguinte campo: ***scale*** **0.75 0.75 0.75**
- pode-se escalar uma geometria num nodo Transform por um factor de0 a infinito em cada um dos eixos individualmente
- animando um escalamento (via um TimeSensor ou um Touch Sensor) num ou em dois eixos, é possível obter efeitos de distorção geometricos interessantes


```
#VRML V2.0 utf8
#movi_cubo4.wrl
DEF transformarCUBO Transform {
 translation  -3.0  2.0  1.0
 rotation -1 1 0.8 0.758
 scale 0.75 0.75 0.75
#únioca alteração do exemplo 4 !
children [
 DEF CUBO Shape {
      appearance Appearance {
        material Material {
           diffuseColor  1 0 0
           specularColor 0.9 0.9 0.9

...
```

Ordem das operações

- quando se usam vários nodos Transform indentados, a ordem das operações de translação, escala e rotação é importante
- este aspecto é mais fácil de detectar se, por exemplo, os valores de escalamento não forem iguais para todos os eixos, uma vez que a rotação afecta a orientação do objecto com respeito aos eixos sobre os quais é realizado o escalamento

Exemplo da ordem das operações

- um cilíndro escalado 50% sobre o eixo X e que a seguir sofre uma rotação de -90 graus sobre o eixo Z
- um cilíndro que sofre uma rotação de -90 graus sobre o eixo Z que a seguir é escalado 50% sobre o eixo de X

Exemplo da ordem das operações

```
#VRML V2.0 utf8
#cilindro_1.wrl
Transform {
   rotation 0 0 1 -1.57
    children [
      Transform {
      scale 0.5 0.0 0.0
      children [
    Shape { geometry Cylinder {} }
       ]
      }
    ]
}
```

Exemplo da ordem das operações

```
#VRML V2.0 utf8
#cilindro_2.wrl
Transform {
  scale 0.5 0.0 0.0
  children [
    Transform {
      rotation 0 0 1 -1.57
      children [
   Shape {geometry Cylinder{}}
      ]
    }
  ]
}
```

São diferentes...!

A ordem de operações tem influência…

• ao efectuar uma **translação** positiva, de 100 metros, no eixo X do sistema de coordenadas local de um nodo filho

• e a seguir se executar uma **rotação** numa transformação de nível superior, em redor do eixo de Z,

• então, em lugar de o nodo filho girar em torno do seu centro, é operada a revolução do nodo filho em redor do centro do nodo pai (ou origem)

Os campos center e scaleOrientation do nodo Transform

- quando se executar uma transformação numa cena ou objecto, é possível prover um deslocamento usando o campo *center* para definir um ponto arbitrário a partir do qual as operações de translação, rotação e escala, sejam calculadas para os nodos filhos na hierarquia.
- a operação de escala permite fixar um valor de vector para o campo *scaleOrientation* do nodo Transform, para efectuar a rotação do sistema de coordenadas antes de efectuar escalamentos

Nodos VRML

- Podem ser divididos em duas categorias principais:
  - gráficos: em conjunto asseguram a construção da cena 3D; incluem as geometrias (Box, Sphere), os atributos dos nodos (Appearance, Material), nodos pai (Shape, Transform)
  - não gráficos: extendem a cena 3D, proporcionando os meios necessários para adicionar efeitos dinâmicos de som, monitorização de eventos e animação
  - existem ainda outros nodos, incluindo luzes, nods de agrupamente (além de Transform) e de ligação (bind)

Nodos gráficos

Grouping	Geometry	Attribute
Shape	Box	Appearance
Anchor	Cone	Color
Billboard	Cylinder	Coordinate
Collision	ElevationGrid	FontStyle
Group	Extrusion	ImageTexture
Transform	IndexedFaceSet	Material
Inline	IndexedLineSet	MovieTexture
LOD	PointSet	Normal
Switch	Sphere	PixelTexture
	Text	TextureCoordinate
		TextureTransform

Nodos não gráficos

Sound	Event triggers	Animation Data
AudioClip	CylinderSensor	ColorInterpolator
Sound	PlaneSensor	CoordinateInterpolator
	ProximitySensor	NormalInterpolator
	SphereSensor	OrientationInterpolator
	TimeSensor	PositionInterpolat
	TouchSensor	ScalarInterpolator
	VisibilitySensor	
	Script	

Outros nodos

Grouping nodes	Environmental nuances
Anchor	Viewpoint
Billboard	NavigationInfo
	BackGround
	Fog
	WorldInfo

Campos dos nodos

- permitem a especificação de atributos que distinguem um nodo de outro do mesmo tipo
- tipos mais comuns:
  - os que aceitam um valor único (iniciados por SF)
  - os que aceitam múltiplos valores de parâmetro (iniciado por MF); necessário especificar os parâmetros entre parentesis rectos “[“ e “]” separados por vírgulas “,” ou espaços

Exemplos da função de campos

- o campo *children* de nodos de agrupamento como *Transform* ou *Anchor* age como um envelope para um ou mais objectos; os membros da hierarquia do grupo de nodos
- o campo *url* do nodo *ImageTexture* aceita múltiplos endereços url para mapas de texturas e materiais; o que permite ao navegador VRML carregar quaisquer texturas localmente antes de o tentar fazer de um servidor
  - se se listar primeiro um drive local para um mapa de materiais ou ficheiro .wrl, o navegador tentará obter a cópia local, parando se o conseguir e prosseguindo para o próximo url caso não tenha ainda carregado o ficheiro em causa

Tipos de campos

Tipo de campo	Argumentos aceites
SFNode/MFNode	nodo VRML
SFBool	TRUE ou FALSE
SFColor/MFColor	Conjunto de 3 valores de 0.0 a 1.0 para (RGB)
SFFloat/MFFloat	número de vírgula flutuante
SFImage	descrição pixel a pixem de um imagemap
SFInt32/MFInt32	inteiro de 32 bits
SFRotation/MFRotation	4 valores; 3 primeiros para o vector de rotação e o último com o valor da rotação em radianos
SFString/MFString	cadeia de caracteres (em utf8, unicode)
SFTime/MFTime	número de segundos desde uma tomada de tempo, número em vírgula flutuante de dupla precisão
SFVec2f/MFVec2f	vector bidimesional de SSFloat
SFVec3f/MFVec3f	vector tridimensional de SSFloat

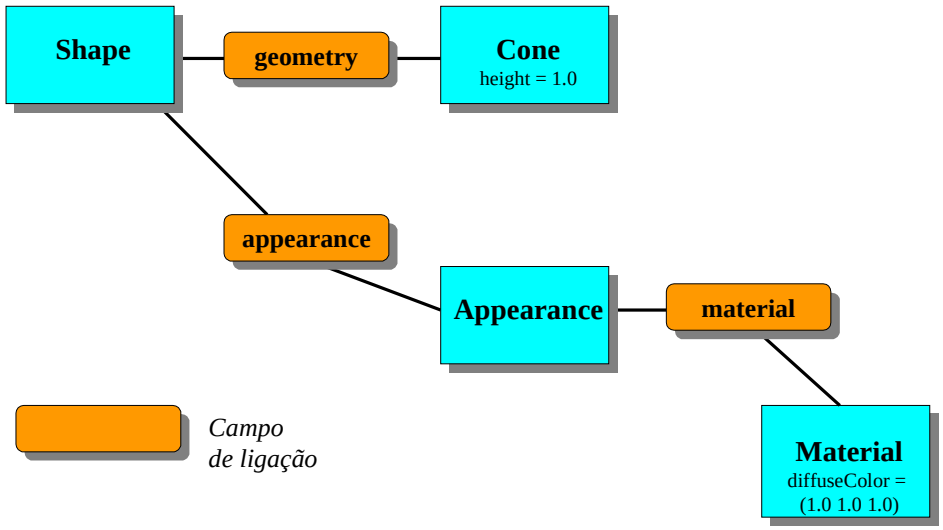
Campos de ligação

- todos os nodos contém um ou mais dos tipos de campos descritos
- ao mais alto nível de abstracção, os campos podem ser classificados como descrevendo atributos de nodos ou como campos de ligação
  - campo de ligação: define quais os nodos VRML que podem ser passados como argumento para o nodo de determinado tipo

Exemplo

```
#VRML V2.0 utf8
#campo_liga.wrl
Shape {
  geometry Cone {
    height 1.0
}
appearance Appearance {
  material Material {
    diffuseColor 1.0 1.0 1.0
  }
}
}
```

Diagrama visual do exemplo



Campos de ligação conectam nodos considerando que os campos de atributo descrevem instâncias de nodo.

O único tipo de campo de ligação que foi pre-definido para o VRML 2.0 foi o campo de tipo SFNode/MFNode. Cada campo SFNode/MFNode possui regras sobre que tipo de nodo pode ser conectado, dependendo do objecto nodo a que o campo pertence.

Foram já utilizadas algumas instancias de campos de ligação SFNode, incluindo o campo *appearance* do nodo *Shape*, que aceita o nodo *Appearance* como argumento e o campo *material* do nodo *Appearance* que aceita o nodo *Material* como argumento. Exemplos de campos MFNodes são o campo *children* de nodos de agrupamento como o nodo *Transform*, o campo *level* do nodo *LOD* e o campo *choice* do nodo *Switch*.

Sistema de cor no VRML

- o nodo *Appearance* permite a definição visual do aspecto de uma geometria VRML definindo as suas propriedades de reflexão à luz
  - forma de passagem ao navegador da informação para a construção de objectos e suas texturas
  - forma de definição das propriedades do material usado
- combinando as opções disponíveis no nodo *Material* incluindo brilho e transparência, é possível atingir um efeito mais realista e de maior impacto
  - a propriedade *emissiveColor* do material produz um efeito semelhante à auto-iluminação e é usado para objectos como lâmpadas incandescentes ou texto 3D

O exemplo esfera.wrl visualiza uma esfera azul uniformemente iluminada usando o campo *emissiveColor* do nodo *Material*. O nodo *Material* foi passado como argumento ao campo *appearance* do nodo *Appearance*. O nodo *Appearance* permite a definição visual do aspecto de uma geometria VRML definindo as suas propriedades de reflexão à luz (material) ou o mapa de textura. Esta foi a forma como se passou ao navegador a informação para a construção da esfera azul.

Quando não se atribui uma textura a uma geometria VRML é esta também a forma de definição das propriedades do material. No entanto, usando uma combinação das opções disponíveis no nodo *Material* incluindo brilho e por vezes transparência, é possível atingir um efeito mais realista e de maior impacto do que o realizado com a esfera azul. A propriedade *emissiveColor* do material produz um efeito semelhante à auto-iluminação e é usado assim, mais adequadamente, para objectos como lâmpadas incandescentes ou texto 3D.

A cor no VRML

- A superfície de um objecto do mundo real não é uniformemente iluminada;
  - geralmente reflecte luz em maior ou menor grau dependendo do ângulo de incidência da luz que atinge o objecto; o campo *diffuseColor* do nodo *Material* define a quantidade de cor da luz que é reflectida da superfície do objecto exposto directamente a uma fonte de luz
  - pode-se usar o campo *specularColor* do nodo *Material* para definir uma valor RGB para destaque das cores na superfície do objecto
  - o valor do brilho do objecto, *shininess*, do nodo *Material* é inversamente proporcional à dimensão e contraste das cores (um maior valor de brilho leva a um pequeno nível de contraste; menores valores em maiores contrastes, com melhor definição)

A superfície de um objecto solido colorido do mundo real não é uniformemente iluminada; geralmente reflecte luz em maior ou menor grau dependendo do ângulo de incidência da luz que atinge o objecto.

Ao passar os valores RGB no intervalo -1.0 a 1.0, o campo *diffuseColor* do nodo *Material*, é definida a quantidade de cor da luz que é reflectida da superfície do objecto exposto directamente a essa fonte de luz. O navegador VRML visualiza o objecto com os resultados da reflexão de luz (luz/sombra), baseado no ângulo de incidência para cada ponto da sua superfície.

De um modo semelhante, é possível usar o campo *specularColor* do nodo *Material* para definir uma valor RGB (normalmente mais luminoso) para destaque das cores na superfície do objecto. A propriedade *specular* da superfície está relacionada com o brilho do objecto - *shininess*, outro campo do nodo *Material* a que podem ser atribuidos valores entre 0.0 e 1.0. O seu valor é inversamente proporcional à dimensão e contraste das cores, isto é, um maior valor de brilho leva a um pequeno nível de contraste, e menores valores de brilho resultam em maiores contrastes, com melhor definição.

O navegador combinará os valores de cor definidos para os objectos com os atribuidos para as luzes colocadas na cena 3D, visualizando a superfície de cada objecto e criando a ilusão de três dimensões num dispositivo de visualização bidimensional - o monitor de computador.

Iluminação de uma cena 3D

```
#VRML V2.0 utf8
#caixa.wrl
DEF CUBO Shape {
  appearance Appearance {
    material Material {
      diffuseColor 0 1 0
      specularColor 0.6 0.6 0.6
      shininess 1.0
    }
  }
  geometry Box { }
}
DirectionalLight {
   direction -1 -1 -5
   intensity 0.8
   }
```

com opção *headlight* **ON**

com opção *headlight* **OFF**

No exemplo movi_cubo.wrl foi criado um cubo definindo a sua superfície como azul. No entanto, quando visualizado na janela do navegador o cubo apareceu com a cor brilhante branca. Tal efeito é devido à propriedade headlight do navegador estar activa por defeito e esta realizar o preenchimento da cena 3D com luz branca, “desbotando” qualquer cor difusa definida na geometria Box.

O interface do navegador permite desactivar o headlight no respectivo menu (também se pode alterar o seu valor no nodo NavigationInfo. No entanto, se não foram colocadas luzes na cena 3D, não há luz para o cubo refletir e não se visualizará nada (poder-sé-à falar, com justificação, de se estar às escuras!).

Iluminação de uma cena 3D

- O VRML possibilita três tipos de fontes de luz para ajudar a iluminar uma cena 3D.
  - **PointLight**; (luz omnidireccional), que representa a fonte de luz num único ponto no espaço, emitindo luz uniformemente em todas as direcções
  - **SpotLight**; que originada num ponto cria um cone de luz numa direcção especificada.
  - **DirectionalLight;** que parece vir de uma dada direcção e possui raios de luz paralelos, usada para simular uma fonte de luz distante como o sol (não suporta o efeito de atenuação, isto é, diminuição após uma dada distancia em radianos)

O VRML possibilita três tipos de fontes de luz para ajudar a iluminar uma cena 3D.

PointLight, designada em alguns sistemas 3D por luz omnidireccional, que representa uma fonte de luz como um único ponto no espaço emitindo luz uniformemente em todas as direcções de modo semelhante ao de uma lâmpada.

SpotLight, que originada num ponto cria um cone de luz numa direcção especificada.

DirectionalLight que parece vir de uma dada direcção e possui raios de luz paralelos (ao invés de raios conicamente difusos como o segundo tipo).

A luz direcionada é geralmente usada para simular uma fonte de luz distante como o sol. Ao contrário da luz pontual e de cone, as luzes direcionadas não suportam o efeito de atenuação, isto é, diminuição após uma dada distancia em radianos.

O exemplo anterior mostra o uso de uma luz direcionada alinhada com o vector <-1, -1, 5>, no espaço cartesiano. A intensidade de iluminação para todas as luzes pode ser definida numa escala de 0.0 a 1.0.

Nodos de agrupamento

- o VRML oferece, além do nodo Transform, outros nodos de agrupamento úteis
- este tipo de nodos permite a atribuição de propriedades a qualquer número de nodos filho, incluindo nodos de agrupamento filhos
- outros nodos de agrupamento: Anchor {}, Billboard {}, Inline {}

O nodo Anchor {}

- permite ligar a cena 3D com outros recursos da Internet, o que inclui páginas HTML e páginas VRML
- uma cena 3D que é carregada atraves de uma ligação deste tipo, substitui, no navegador, a cena que se encontrava a ser visualizada

O nodo Anchor {}, exemplo

```
#VRML V2.0 utf8
#salto.wrl
Transform {
 children [
  Anchor {
    description "Veja o interior da esfera!"
    url [ "texto.wrl" ]
    children [
     DEF ENTRADA Shape {
         geometry Sphere {}     }
      ]
    }
  ]
}
```

O nodo Billboard {}

- permite a criação de sprites na cena 3D
- o navegador de VRML orienta o eixo Z local de modo a apontar sempre directamente para o ponto de vista do utilizador
  - permite criar a ilusão de uma geometria 3D complexa, usando um poligono de uma só face, mapeado com uma imagem 3D (a usar!; produz bons efeitos e diminui a carga de processamento do navegador, para a visualização da cena 3D)

```
#VRML V2.0 utf8
#mistura.wrl
Transform {
 children [
  DEF FCP Inline {
   url [ "texto.wrl"]
  }
 children [
  Transform {
   translation -2 0 0
   children [
    Billboard {
     axisOfRotation  0 0 0
     children [
      Shape {
       appearance Appearance {
        material Material {
         emissiveColor  0 0 1
        }
       }
       geometry Sphere {
        radius 1.5
       }
      }
     ]
    }
   ]
  }
 ]
 ]

}
```

O nodo Billboard {}

- Seleciona-se um eixo de rotação para o nodo *Billboard* (*axisOfRotation*) definindo as coordenadas X, Y, Z de um ponto no espaço que forme o eixo de rotação, ligado com a origem local do nodo de agrupamento
  - este ponto é muitas vezes definido com *axisOfRotation* 0 1 0 para efectuar a rotação de um objecto de uma só face, em torno de Y. Reproduz, por exemplo, um candeeiro de iluminação de rua de um conjunto de candeeiros alinhados e com o mesmo plano de chão
  - para criar um *Billboard* que esteja sempre de frente para o utilizador, quaisquer que sejam os seus movimentos, define-se *axisOfRotation* 0 0 0; este tipo de *Billboard* é conhecido por sprite

Visualização da cena 3D do programa expo.wrl, mostrando o alinhamento da esfera com o utilizador.

O nodo Inline {}

- permite “importar” um ficheiro VRML .wrl, por exemplo, um avatar ou outra geometria, para a cena 3D corrente, especificando uma referência url para a sua localização
- usando nodos Inline (e mantendo uma escala de modelação consistente!) é possível criar uma biblioteca de objectos reutilizáveis a partir da qual se podem construir cenas 3D distintas

O nodo Inline {}, exemplo

```
#VRML V2.0 utf8
#mistura.wrl
Transform {
  children [
    DEF FCP Inline {
      url [ "texto.wrl"]
    }
  children [
    Transform {
      translation -2 0 0
      children [
        Shape {
          ...
          geometry Sphere {
            radius 1.5
          ...
```

Programa completo…

```
#VRML V2.0 utf8
#mistura.wrl
Transform {
  children [
    DEF FCP Inline {
      url [ "texto.wrl"]
    }
  children [
    Transform {
      translation -2 0 0
      children [
        Shape {
          appearance Appearance {
            material Material {
              emissiveColor  0 0 1
            }
          }
          geometry Sphere {
            radius 1.5
          }
        }
      ]
    }
  ]
  ]
}
```

No campo field do nodo Inline podem ser especificadas múltiplas referências url, de modo a permitir ao navegador verificar a existência de determinado ficheiro url num drive local e posteriormente na web.

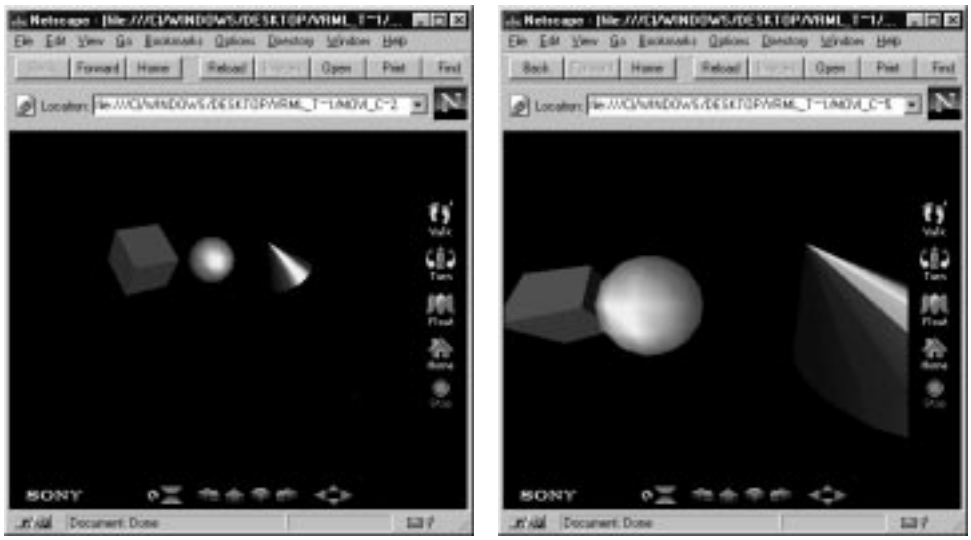
Nuances de ambiente da cena 3D

- o VRML oferece um conjunto de nodos de integração como *Viewpoint*, *Background*, *NavigationInfo*, *Fog* que permitem melhorar a experiência de navegação na cena 3D
  - lidam com aspectos de estética do mundo virtual
- só se pode utilizar um nodo descrito a cada dado momento, dos nodos acima descritos

O nodo Viewpoint {}

- Actua como uma câmera, especifica o ponto de vista do utilizador quando a cena 3D é carregada pelo navegador
  - ajustando o parâmetro fieldOfView (expresso em radianos) para um valor alto, é produzida uma imagem distorcida nos cantos da imagem visualizada pelo navegador, simulando uma lente de grande angular

O nodo Viewpoint {}, exemplo



```
#VRML V2.0 utf8
#movi_cubo5.wrl
DEF transformarCUBO Transform {
  translation  -3.0  2.0  1.0
  rotation -1, 1, 0.8, 0.785
  scale 0.75 0.75 0.75
  children [
    DEF CUBO Shape {
      appearance Appearance {
        material Material {
          diffuseColor  1 0 0
          specularColor 0.9 0.9 0.9
        }
      }
      geometry Box { }
    }
  ]
}
DEF transformarBOLA Transform {
  translation 0.0 2.0 1.0
  children [
    DEF BOLA Shape {
      appearance Appearance {
      material Material {
        diffuseColor  0 1 0
        specularColor 0.9 0.9 0.9
      }
    }
    geometry Sphere {}
  }
  ]
}
```

```
DEF transformarCONE Transform {
  translation 3  2  1
  rotation 0.5 0.1 0.75 0.785
  children [
    DEF CONE Shape {
      appearance Appearance {
        material Material {
          diffuseColor  0 0 1
          specularColor 0.9 0.9 0.9
        }
      }
      geometry Cone {}
    }
  ]
}
DirectionalLight {
  direction -2 1 -5
  intensity 0.8
  }
Viewpoint {
  fieldOfView 2.0
  position 1.75 2 3
  orientation 0 0.5 1 0.785
  description "Que ressaca..."
}
```

O nodo Viewpoint { }

- se não se define um *Viewpoint* para a cena 3D, a maioria dos navegadores vai visualizar a cena de uma posição do eixo de Z positivo para uma distância a partir da qual a cena de VRML inteira possa ser vista
  - este aspecto constituiu um comportamento perfeitamente funcional, forçando todos os objectos a "caber" dentro dos limites da janela de visualização, tal quebra as regras da boa composição
  - a definição de *Viewpoints* ajuda o utilizador a apreciar melhor a cena visualizada
  - o *Viewpoint* deve ser definido no ficheiro .wrl original e não pode ser carregado de um ficheiro obtido pelo nodo *Inline* nem referênciado por um URL

O nodo NavigationInfo {}

- permite definir o tamanho da presença do utilizador ou avatar na cena 3D
- usado pelo navegador para a detecção de colisões com os elementos da cena, determinando onde se deve e não deve represntar a luz de presença "*headlight*"
- determina a distância de visibilidade da cena para o utilizador
- determina a velocidade de travessia da cena 3D pelo utlizador

O nodo NavigationInfo {}

- Valores por defeito para os campos do nodo

```
NavigationInfo {
  avatarSize [ 0.25, 1.6, 0.75 ]
  headlight TRUE
  speed 1.0
  type “WALK”
  visibilityLimit 0.0
}
```

- qualquer transformação de escala da cena 3D que afecte o utilizador, automaticamente escala os valores dos campos do nodo *NavigationInfo*

O campo *avatarSize* define três valores para o avatar. O primeiro valor está relacionado com a distância entre o avatar e um objecto da cena em que é detectada uma colisão. O segundo valor determina a altura do ponto de vista do utilizador, acima do plano de “chão”. O terceiro valor foi projetado para permitir a avatars subir escadarias, calhas, ou outro pequeno obstáculo na cena, denotando uma altura aceitável para esse obstáculo; o valor por defeito é o recomendado pelo VRML Architecture Group (VAG).

O campo *speed* é expresso em metros por segundo, e determina a rapidez com que é realizada a travessia de uma cena 3D. Se o utilizador se encontra estacionário, este campo denota a velocidade de “panning” da cena.

O campo *type*, que pode ser usado com sensores de proximidade ou outros, para despoletar uma navegação do tipo EXAMINE (o utilizador pode tomar e rodar um objecto, dependendo da funcionalidade do navegador utilizado). No mínimo, todos os navegadores suportam o tipo de navegação WALK e a maioria suporta alguma forma de FLY (não transmite ao utilizador a noção de gravidade).

O campo *visibilityLimit* pode ser usado cem conjunto com o campo *visibilityRange*, do nodo *Fog*, para assegurar que os objectos a partir de uma dada distância se esvanecem no fundo da cena.

Note-se que qualquer transformação na escala na cena 3D que afecte o utilizador, automaticamente escala os valores dos campos *avatarSize*, *speed*, e *visibilityLimit* do nodo *NavigationInfo*.

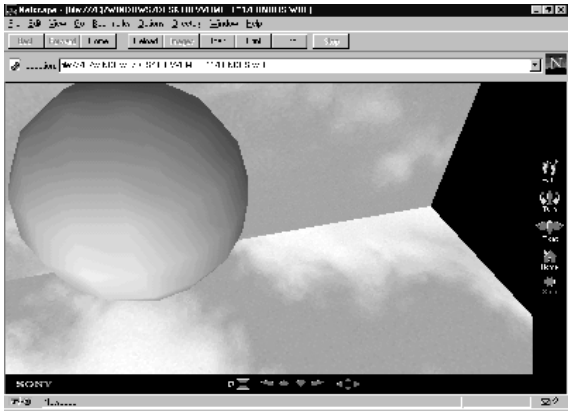
O nodo Background {}

- permite especificar uma cor de fundo, gradiente, ou textura como pano de fundo da cena 3D em vez do valor por defeito (cor preta)
- podem-se usar os campos do nodo *Background* para atribuir diferentes texturas para cada uma das 6 “paredes”, tomando a cena 3D como contida num cubo
- o nodo *Background* não é afectado pelo nodo *Fog,* o que permite desvanecer a visualização dos objectos na cor de nevoeiro com a distância ao utilizador e assim proporcionar a ilusão e de percepção de distâncias na cena 3D

```
Background {
   frontUrl “textura1.jpg“
   backUrl “textura2.jpg“
   bottomUrl “textura3.jpg“
   leftUrl “textura4.jpg“
   rightUrl “textura5.jpg“
   topUrl “textura6.jpg“
}
```

O nodo Background {}, exemplo

```
#VRML V2.0 utf8
#fundos.wrl
Shape {
   appearance Appearance {
     material Material {
       emissiveColor  0 0 1
     }
   }
   geometry Sphere {
      radius 10
   }
}
Background {
  frontUrl "ceu.jpg"
  bottomUrl "ceu.jpg"
  topUrl "ceu.jpg"
}
```



```
Background {
  eventIn          SFBool   set_bind
  exposedField  MFFloat  groundAngle  [ ]
  exposedfield   MFColor  groundColor  [ ]
  exposedField  MFString backUrl       [ ]
  exposedField  MFString bottomUrl   [ ]
  exposedField  MFString frontUrl      [ ]
  exposedField  MFString leftUrl       [ ]
  exposedField  MFString rightUrl      [ ]
  exposedField  MFString topUrl        [ ]
  exposedField  MFFloat  skyAngle     [ ]
  exposedField  MFColor  skyColor     [ 0 0 0 ]
  eventOut        SFBool   isBound
}
```

O nodo Fog {}

- permite a criação de pistas para a percepção de distâncias e efeitos atmosféricos, desvanecendo os objectos da cena 3D, na cor do nevoeiro, de acordo com a distância ao ponto de vista do utilizador
- valores por defeito do nodo *Fog*

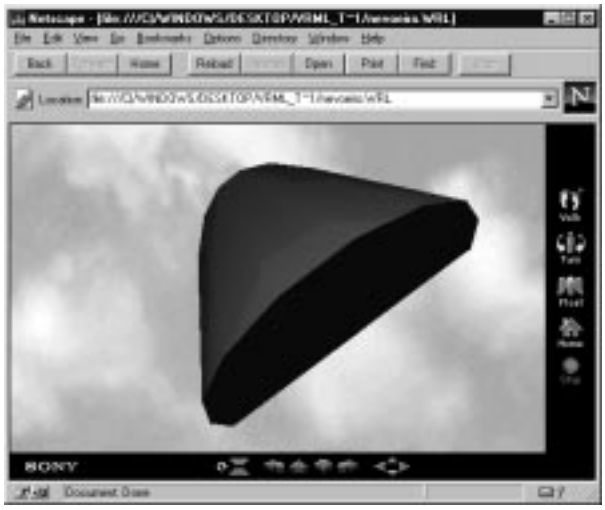
```
Fog  {
  color 1 1 1
  fogType “LINEAR”
  visibilityRange 0
}
```

O nodo Fog {}

- o campo *color* permite a atribuição de cor com a qual os objectos mais distantes são misturados para criar a ilusão do nevoeiro atmosférico
- o campo *fogType* aceita como argumentos os valores *LINEAR* e *EXPONENTIAL*, dependendo se se pretende simular distancias ou tempo nebuloso, respectivamente
- o campo *visibilityRange* expecifica a distancia, em metros, desde a origem do nodo *Fog*, a partir da qual os objectos são completamente “tapados” (para produzir efeitos de nevoeiro, o valor atribuido tem de ser maior que 0)

O nodo Fog {}, exemplo

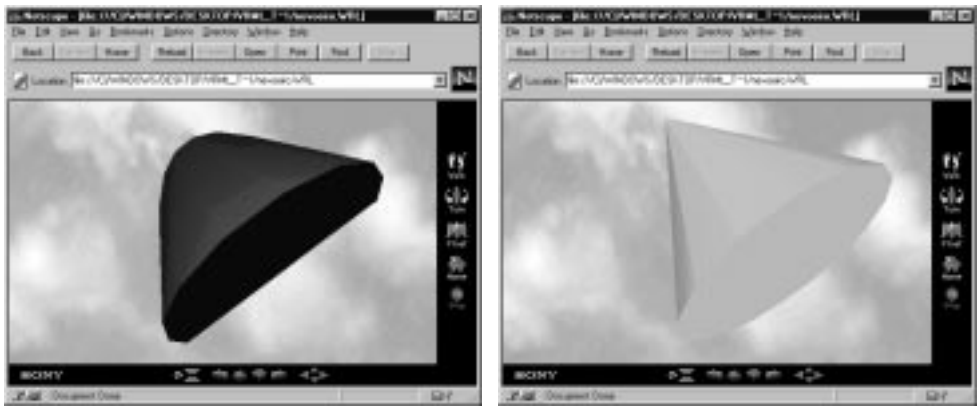
```
#VRML V2.0 utf8
#nevoeiro.wrl
Transform {
  rotation -0.9 0 1 0.785
  ...
     geometry Cone {
       bottomRadius 1.6
       height 3.5
     }
  ...
Fog {
  color 1 0 0
  fogType "EXPONENTIAL"
  visibilityRange 4.2
}
Background {
  frontUrl "ceu.jpg"
}
```



Programa completo…

```
#VRML V2.0 utf8
#nevoeiro.wrl
Transform {
  rotation -0.9 0 1 0.785
  children [
    Shape {
      appearance Appearance {
        material Material {
          emissiveColor  0 1 0
        }
      }
      geometry Cone {
      bottomRadius 1.6
      height 3.5
      }
    }
  ]
}
Fog {
   color 1 0 0
   fogType "EXPONENTIAL"
   visibilityRange 4.2
}
Background {
   frontUrl "ceu.jpg"
}
```

O nodo Fog {}, exemplo



Camparação com efeito do nodo *Fog* e sem esse efeito…
visibilityRange = 4.2 ou *visibilityRange* = 0

O nodo WorldInfo {}

- além dos comentários contidos no programa VRML é possível documentar a cena 3D com o respectivo nome (campo *title*) e com informação sobre os direitos de autor (campo *info*)

```
#VRML V2.0 utf8
WorldInfo {
info [ “Mundo criado por MMIG/CEREM” ]
title “UFP vista_norte”
}
```

- a colocação destes elementos como comentários pode ser retirada pelo navegador como forma de optimizar o tempo de visualização da cena 3D