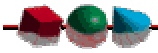


5. Shape - Formas Geométricas Avançadas

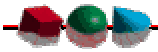
As formas geométricas avançadas que podem ser definidas no campo **geometry** do node **Shape** são **PointSet**, **IndexedLineSet, IndexedFaceSet, ElevationGrid, Extrusion** e **Text.** Essas geometrias são apresentadas no que se segue.

5.1. PointSet

A geometria **PointSet** especifica um conjunto de pontos 3D no sistema de coordenadas local e as cores associadas a cada ponto. PointSet contém dois campos: **coord**, que define as coordenadas de cada um dos pontos e **color**, que define a cor de cada um dos pontos. Deve haver tantas cores quanto forem os pontos.

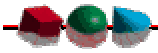
Sintaxe:

```
geometry PointSet
        { coord Coordinate
                { point [ □ □ □,
                          . . .
                          □ □ □
                        ]
                }
          color Color
                { color [ □ □ □,
                          . . .
                          □ □ □
                        ]
                }

        }
```



Exemplo:

```
#VRML V2.0 utf8
#PointSet.wrl
#M#VRML V2.0 utf8
#PointSet.wrl
#Mostra uma "chuva de pontos" através da geometria PointSet

Shape
{  appearance Appearance    {   }
   geometry PointSet
            { coord Coordinate
                    { point
                      [ #  Cooordenadas dos Pontos cor ciano
                        1.0 1.0 1.0,  0.5 1.0 1.0,
                        1.5 1.5 1.5,  1.0 0.5 1.0,
                        2.0 2.0 2.0,  1.0 1.0 0.5,
                        2.5 2.5 2.5,  1.5  2.0 2.0,
                        3.0 3.0 3.0,  2.0 1.5 2.0,
                        # Coordenadas dos Pontos cor Amarela
                        -1.0 -1.0 -1.0,  -0.5 -1.0 -1.0,
                        -1.5 -1.5 -1.5,  -1.0 -0.5 -1.0,
                        -2.0 -2.0 -2.0,  -1.0 -1.0 -0.5,
                        -2.5 -2.5 -2.5,  -1.5  -2.0 -2.0,
                        -3.0 -3.0 -3.0,  -2.0 -1.5 -2.0
                      ]
                    }
                color Color
                      { color
                        [ # cor ciano
                          0.0 1.0 1.0,   0.0 1.0 1.0,
                          0.0 1.0 1.0,   0.0 1.0 1.0,
                          0.0 1.0 1.0,   0.0 1.0 1.0,
                          0.0 1.0 1.0,   0.0 1.0 1.0,
                          0.0 1.0 1.0,   0.0 1.0 1.0,
                          # cor amarela
                          1.0 1.0 0.0,   1.0 1.0 0.0,
                          1.0 1.0 0.0,   1.0 1.0 0.0,
                          1.0 1.0 0.0,   1.0 1.0 0.0,
                          1.0 1.0 0.0,   1.0 1.0 0.0,
                          1.0 1.0 0.0,   1.0 1.0 0.0,
                        ]
                      }
              }
}
```



5.2. IndexedLineSet

A geometria **IndexedLineSet** especifica um conjunto de poli-linhas no sistema de coordenadas local e as cores associadas. Através da ligação das linhas, pode-se criar o contorno de qualquer polígono ou contornos de diferentes objetos, como uma estrela, uma casa, um cubo, uma pirâmide, etc. A geometria contém cinco campos:

coord: especifica as coordenadas dos pontos que serão conectados para formar as linhas; o primeiro ponto é o ponto 0, o segundo ponto 1 e assim por diante.

coordIndex: indica como os pontos serão ligados, formando as poli-linhas; quando uma poli-linha termina, ela é "partida" pelo número **–1**; para formar um polígono fechado é necessário repetir o primeiro ponto.

color: indica as cores que serão usadas para colorir as poli-linhas; a primeira cor é 0, a segunda 1 e assim por diante.

colorIndex: indica uma cor para cada uma das poli-linhas definidas no campo coordIndex ou cada um dos pontos definidos em coord.

colorPervertex: se **FALSE** indica que as cores serão aplicadas às poli-linhas; neste caso, deverá haver uma cor associada à cada poli-linha definida em coordIndex (as cores podem se repetir);

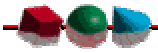
se **TRUE** indica que as cores serão aplicadas aos pontos (vértices): o resultado final é que cada linha começará com uma cor e terminará com outra, produzindo um efeito gradiente; neste caso deverá haver tantas cores (definidas no campo color) quanto o número de vértices (pontos) definidos em coord.

Se considerarmos as coordenadas dos vértices como sendo **ponto 0** (0 0 0), **ponto 1** (1 1 0), **ponto 2** (-1 0 0) e **ponto 3** (-1 -1 0), a figura 11 mostra o resultado de 3 ligações diferentes destes pontos que seriam definidos no campo coordIndex.

Figura 11 – diferentes ligações de 4 pontos

Sintaxe:

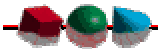
```
geometry IndexedLineSet
        { coord Coordinate
                { point [□ □ □, #ponto 0
                         ...
                         □ □ □
                        ]
                }

          coordIndex [□, .. , □, -1,..., □, .., □
                      ]

          color Color
                { color [□ □ □,  #cor 0
                         ...
                         □ □ □
                        ]
                }

          colorIndex [□, □,  ...,  □
                      ]

          colorPerVertex TRUE/FALSE
        }
```



Exemplo:

Veja também o programa **EstrelasComLinhas.wrl**.

```
#VRML V2.0 utf8
#IndexedLineSet.wrl
#Desenho das arestas de um cubo com#VRML V2.0 utf8
#IndexedLineSet.wrl
#Desenho das arestas de um cubo com 3 cores diferentes

Shape
{  appearance Appearance   {  }
   geometry IndexedLineSet
   {  coord Coordinate
      {  point
         [ # Cubo
             1.0  1.0  1.0,  # ponto 0
             1.0 -1.0  1.0,  # ponto 1
            -1.0 -1.0  1.0,  # ponto 2
            -1.0  1.0  1.0,  # ponto 3
             1.0  1.0 -1.0,  # ponto 4
             1.0 -1.0 -1.0,  # ponto 5
            -1.0 -1.0 -1.0,  # ponto 6
            -1.0  1.0 -1.0   # ponto 7
         ]
      }
      color Color
      {  color
         [1.0 0.0 0.0, 0.0 1.0 0.0, 0.0 0.0 1.0
          # apenas 3 cores 0=Vermelho, 1=Verde e 2=Azul
         ]
      }
      coordIndex
      [  # Linha vermelha (paralela ao eixo x)
         0, 3, -1, 1, 2, -1, 4, 7, -1, 5, 6, -1,
         # Linha verde (paralela ao eixo y)
         0, 1, -1, 2, 3, -1, 4, 5, -1, 6, 7, -1,
         # Linha azul (paralela ao eixo z)
         0, 4, -1, 1, 5, -1, 2, 6, -1, 3, 7, -1
      ]
      colorIndex
      [ 0, 0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 2 ]
      colorPerVertex FALSE
   }
}
```



5.3. IndexedFaceSet

A geometria **IndexedFaceSet** especifica um conjunto de faces no sistema de coordenadas local e as cores ou texturas associadas. Através da ligação das faces, pode-se criar qualquer polígono ou diferentes objetos com as faces preenchidas com cores sólidas, como uma estrela, uma casa, um cubo, uma pirâmide, etc. A geometria contém onze campos, que incluem **texCoord** e **texCoordIndex** que permitem a aplicação de texturas às faces, mas que não serão apresentados aqui. Os nove campos a serem abordados são:

coord: especifica as coordenadas dos pontos que serão conectados para formar as faces; o primeiro ponto é o ponto 0, o segundo ponto 1 e assim por diante.

coordIndex: indica como os pontos serão ligados, formando as faces; quando uma face termina, ela é "finalizada" pelo número **–1**. Não é necessário repetir o primeiro ponto, a face é sempre fechada.

color: indica as cores que serão usadas para colorir as faces; a primeira cor é 0, a segunda 1 e assim por diante.

colorIndex: indica uma cor para cada uma das faces definidas no campo coordIndex ou cada um dos pontos definidos em coord.

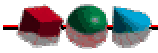
colorPervertex: similar ao definido na geometria indexedLineSet.

ccw: TRUE define se os pontos que delimitam as faces serão apresentados em sentido anti-horário; no sentido horário ou desconhecido, caso seja **FALSE**.

solid: FALSE determina se o *browser* deve desenhar ambos os lados das faces; **TRUE**, desenha só a frente.

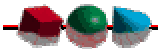
convex: deve ser **TRUE**, pois indica que as faces são convexas; FALSE indicaria que elas seriam côncavas, mas o VRML só trabalha com faces convexas.

creaseAngle: especifica o limiar de ângulo (em radianos); se duas faces adjacentes fazem um ângulo maior que o creaseAngle, é possível ver claramente que duas faces de encontram, pois o encontro delas fica abrupto, o ideal é que esse encontro seja suave.



Sintaxe:

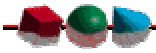
```
geometry IndexedLineSet
        { coord Coordinate
                { point [□ □ □, #ponto 0
                         ...
                         □ □ □
                        ]
                }


          coordIndex [□, .. , □, -1,..., □, .., □
                      ]

          color Color
                { color [□ □ □,  #cor 0
                         ...
                         □ □ □
                        ]
                }

          colorIndex [□, □,  ...,  □
                      ]

          colorPerVertex TRUE/FALSE

          solid TRUE/FALSE

          ccw TRUE/FALSE

          convex TRUE/FALSE

          creaseAngle □
        }
```



Exemplo:

Veja também o programa **EstrelaComFaces.wrl**.

```
#VRML V2.0 utf8
#IndexedFaceSet.wrl
#Construcao de uma piramide com faces de diferentes cores.

Shape
{  appearance Appearance
   {
      material Material {  }
   }

   geometry IndexedFaceSet
   {
      coord Coordinate
      {
         point
          [
             # Coordenadas dos pontos que formarao as faces da piramide
             -1  -1  0,
              1  -1  0,
              1   1  0,
             -1   1  0,
              0   0  1
          ]
      }
      coordIndex
      [
         0 1 4 -1,
         1 2 4 -1,
         2 3 4 -1,
         3 0 4
      ]
      color Color
      {
         color
         [
            1.0 0.0 0.0, 0.0 1.0 0.0, 0.0 0.0 1.0
            # apenas 3 cores 0=Vermelho, 1=Verde e 2=Azul
         ]
      }
      colorPerVertex   FALSE
      colorIndex
      [ 1 2 0 2 ]

      solid FALSE
      ccw TRUE
      convex TRUE
      creaseAngle 0
   }
}
```



5.4. ElevationGrid

A geometria ElevationGrid especifica uma matriz de pontos, cada qual com uma altura definida; é uma geometria útil para se construir um terreno acidentado ou malhas. A geometria é construída no plano XZ, como uma matriz, começando da origem e expandindo-se na direção positiva dos eixos. Os seguintes campos fazem parte de sua definição:

xDimension: define o número de pontos no eixo X

zDimension: define o número de pontos no eixo Z

xSpacing: define a distância entre dois pontos adjacentes no eixo X

zSpacing: define a distância entre dois pontos adjacentes no eixo Z

height: define uma lista de valores em ponto flutuante que especificam a altura de cada ponto da matriz; os pontos são ordenados da esquerda para a direita e de cima para baixo. Deverá haver xDimension multiplicado por zDimension pontos.

color: define a cor de cada um dos pontos da matriz; mas é opcional.

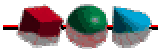
colorPerVertex: similar ao definido na geometria IndexedLineSet.

ccw: TRUE define se os espaços da matriz de pontos serão apresentados em sentido anti-horário; horário ou desconhecido, caso seja **FALSE**.

solid: FALSE determina se o *browser* deve desenhar ambos os lados da matriz de pontos; **TRUE**, desenha só a parte de cima.

convex: deve ser **TRUE**, pois indica que as faces são convexas; FALSE indicaria que elas seriam côncavas, mas o VRML só trabalha com faces convexas.

creaseAngle: especifica o limiar de ângulo (em radianos); se dois pontos adjacentes da matriz fazem um ângulo maior que o creaseAngle, é possível ver claramente que os dois se encontram, pois o encontro deles fica abrupto, o ideal é que esse encontro seja suave.



Sintaxe:

```
geometry ElevationGrid
            { xDimension □
              zDimension □
              xSpacing □
              zSpacing □
              height [ □ □ . . . □ □
                       . . . .
                       □ □ . . . □ □
                     ]
              color Color
                     { color [□ □ □, . . . , □ □ □,
                              . . .
                              □ □ □, . . . , □ □ □
                             ]
                     }
              colorPerVertex TRUE/FALSE
              solid TRUE/FALSE
              ccw TRUE/FALSE
              convex TRUE/FALSE
              creaseAngle □
            }
```

Exemplo:

Veja o exemplo **ElevationGrid.wrl**; como seu código é muito grande e complicado, não será reproduzido aqui. No seu lugar apresenta-se o programa **ElevationGridChess.wrl** que desenha um tabuleiro de xadrez. Edite o programa e visualize-o no Cortona.

```
#VRML V2.0 utf8
#ElevationGridChess.wrl
#Desenha um tabuleiro de xadrez com a geometria ElevationGrid

Shape
{  geometry ElevationGrid
   { #VRML V2.0 utf8
#ElevationGridChess.wrl
#Desenha um tabuleiro de xadrez com a geometria ElevationGrid

Shape
{

  geometry ElevationGrid
  {   xDimension 9
      zDimension 9
      xSpacing 1
      zSpacing 1
      height [    0 0 0 0 0 0 0 0 0
                  0 0 0 0 0 0 0 0 0
                  0 0 0 0 0 0 0 0 0
                  0 0 0 0 0 0 0 0 0
                  0 0 0 0 0 0 0 0 0
                  0 0 0 0 0 0 0 0 0
                  0 0 0 0 0 0 0 0 0
                  0 0 0 0 0 0 0 0 0
                  0 0 0 0 0 0 0 0 0
            ]
colorPerVertex FALSE
color Color
      { color [   0 0 0, 1 1 1, 0 0 0, 1 1 1, 0 0 0, 1 1 1, 0 0 0, 1 1 1,
                  1 1 1, 0 0 0, 1 1 1, 0 0 0, 1 1 1, 0 0 0, 1 1 1, 0 0 0,
                  0 0 0, 1 1 1, 0 0 0, 1 1 1, 0 0 0, 1 1 1, 0 0 0, 1 1 1,
                  1 1 1, 0 0 0, 1 1 1, 0 0 0, 1 1 1, 0 0 0, 1 1 1, 0 0 0,
                  0 0 0, 1 1 1, 0 0 0, 1 1 1, 0 0 0, 1 1 1, 0 0 0, 1 1 1,
                  1 1 1, 0 0 0, 1 1 1, 0 0 0, 1 1 1, 0 0 0, 1 1 1, 0 0 0,
                  0 0 0, 1 1 1, 0 0 0, 1 1 1, 0 0 0, 1 1 1, 0 0 0, 1 1 1,
                  1 1 1, 0 0 0, 1 1 1, 0 0 0, 1 1 1, 0 0 0, 1 1 1, 0 0 0,
              ]
      }
   }
}
```

5.5. Extrusion

A geometria Extrusion é muito poderosa, permitindo que se construam formas geométricas complexas usando apenas alguns pontos.

Sintaxe:

```
geometry Extrusion
        { crossSection [□ □, . . . ,□ □ ]
          orientation □ □ □ □
          scale □ □
          solid TRUE/FALSE
          spine [□ □ □, □ □ □, . . . , □ □ □ ]
          beginCap TRUE/FALSE
          endCap TRUE/FALSE
          ccw TRUE/FALSE
          convex TRUE/FALSE
          creaseAngle □
        }
```

A base de uma forma construída com a geometria Extrusion é uma seção 2D. Por exemplo, considerando um cubo, a base que forma sua seção 2D é um quadrado. As seções 2D são definidas no plano XZ e as coordenadas de seus pontos são definidas no campo **crossSection**. A figura 12 mostra uma seção 2D para um cubo.

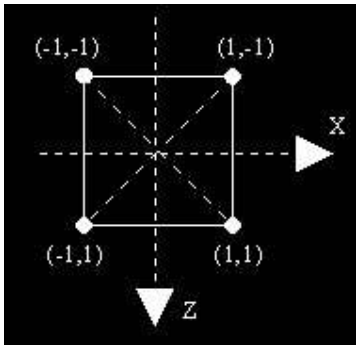


Figura 12 – base que forma uma seção 2D para um cubo

Outro conceito necessário para a construção de uma geometria Extrusion é o campo **spine**. A spine define o caminho que a seção 2D vai percorrer para criar a forma geométrica. Para construir um cubo a partir da seção 2D mostrada na figura 12, pode-se começar fazendo a seção 2D cruzar o plano em (0,-1,0) (direção negativa do eixo Y) em direção a (0,1,0) (direção positiva de Y). A figura 13 mostra a spine para o cubo e o caminho a ser percorrido pela seção 2D.

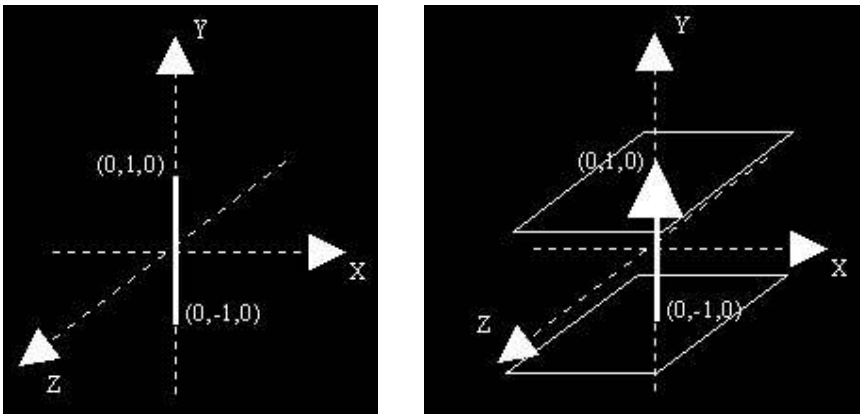


Figura 13 – spine para a seção 2D da figura 12

A spine da figura 13 é definida por dois pontos (0,-1,0) e (0,1,0), embora possam ter tantos pontos quanto o necessário para a definição da forma. A lista de passos que o *browser* segue para desenhar a geometria Extrusion a partir dos pontos de spine é:

1. Translada a seção 2D para o primeiro ponto da spine.
2. Reorienta a seção 2D definida no plano XZ de forma que o eixo Y coincida com a direção obtida com os dois pontos da spine (no exemplo da figura 13, esse passo não é necessário).
3. Move a seção 2D para o último ponto da spine.

Após a execução do último passo, o *browser* vai criar as paredes laterais do cubo. O resultado final é mostrado na figura 14.

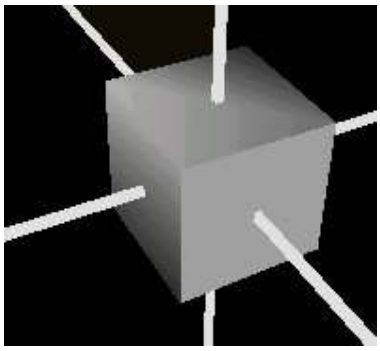


Figura 14 – cubo construído a partir da seção 2D da figura 12 e spine da figura 13

Os campos **beginCap** e **endCap** especificam se a forma será aberta ou fechada em suas extremidades. A figura 15(a) mostra o cubo da figura 14 quando os campos beginCap e endCap estão marcados como FALSE – o cubo está vazado em ambas as extremidades. Além disso, é possível ver somente as duas faces que estão viradas para os olhos do leitor, pois o campo **solid** está marcado como TRUE (ainda em 15a). Marcando o campo solid para FALSE, a visão do cubo torna-se a mostrada em 15(b).

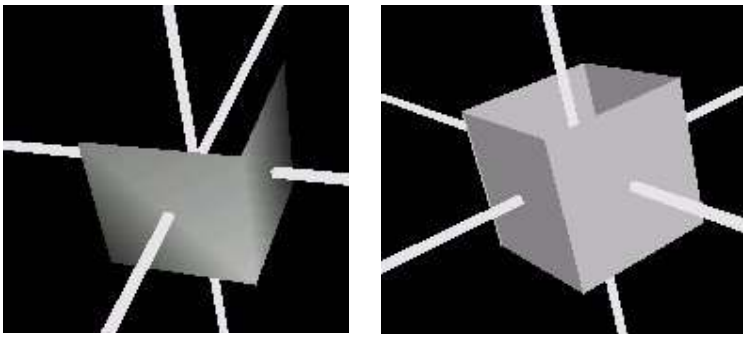
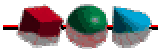


Figura 15 -(a) cubo com campos beginCap e endCap FALSE e solid TRUE
(b) campo solid FALSE

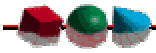
Exemplo:

```
#VRML V2.0 utf8
#ExtrusionCube.wrl
#Desenha um cubo a partir da geometria Extrusion

Transform
{ children [ Shape
                 { appearance Appearance { material Material {}
                                         }
                   geometry Extrusion
                            { crossSection [ -1 -1,
                                             -1  1,
                                              1  1,
                                              1 -1,
                                             -1 -1
                                           ]
                              spine [0 -1 0 , 0 1 0 ]
                              beginCap FALSE
                              endCap FALSE
                              solid FALSE
                            }
                 }
           ]
}
```



Veja também o exemplo **Extrusion.wrl**, apresentado a seguir:

```
#VRML V2.0 utf8
#Extrusion.wrl
#Construcao de um vaso com a geometria Extrusion

#Define ponto de vista inicial
Viewpoint
{  position 0.0 2.0 10.0
   description "Visao inicial"
}

Group
{  children
   [  Shape
      {  appearance Appearance
         {  material Material { diffuseColor 1.0 1.0 0.0 }
         }
         geometry Extrusion
         {  creaseAngle 1.57
            endCap FALSE
            solid  FALSE
            crossSection
            [  # Pontos em XZ que definem a seção 2D que e um circulo
               1.00  0.00,   0.92 -0.38,
               0.71 -0.71,   0.38 -0.92,
               0.00 -1.00,  -0.38 -0.92,
              -0.71 -0.71,  -0.92 -0.38,
              -1.00 -0.00,  -0.92  0.38,
              -0.71  0.71,  -0.38  0.92,
               0.00  1.00,   0.38  0.92,
               0.71  0.71,   0.92  0.38,
               1.00  0.00
            ]
            spine
            [  # Pontos que formam os planos que a seção 2D vai cruzar
               0.0 0.0 0.0,  0.0 0.6 0.0,
               0.0 1.0 0.0,  0.0 1.4 0.0,
               0.0 1.8 0.0,  0.0 2.2 0.0,
               0.0 2.6 0.0,  0.0 3.0 0.0,
               0.0 3.4 0.0,  0.0 3.8 0.0,
               0.0 4.2 0.0
            ]
            scale
            [  1.5  1.5,  1.95 1.95,
               2.0  2.0,  1.95 1.95
               1.8  1.8,  1.5  1.5
               1.2  1.2,  1.05 1.05,
               1.0  1.0,  1.05 1.05,
               1.3  1.3,
            ]
         }
      }
   ]
}
```



5.6. Text

O texto é tratado como uma geometria também, permitindo que palavras e frases sejam mostrados nos mundos VRML. Os campos que o definem são:

string: contém o texto a ser mostrado; pode ter uma ou mais linhas.

fontStyle: especifica um node à parte que permite que sejam definidos aspectos de como o texto será apresentado. É composto dos campos:

family: especifica o fonte; valores neste campo podem ser: "SERIF", "SANS", "TYPEWRITER".

style: especifica o estilo do fonte; valores neste campo podem ser: "PLAIN", "BOLD", "ITALIC", "BOLDITALIC".

horizontal: TRUE indica que o texto deve ser mostrado na horizontal, FALSE, na vertical.

leftToRight: TRUE indica que o texto deve ser escrito da esquerda para a direita; FALSE, da direita para a esquerda (modo árabe)

topToBottom: TRUE indica que o texto deve ser escrito de cima para baixo; FALSE, de baixo para cima (modo chinês).

justify: se horizontal é TRUE, a justificação principal é horizontal e a secundária é vertical; se horizontal for FALSE é o contrário. Há quatro valores para este campo: "FIRST", "BEGIN" (justificado à esquerda), "MIDDLE" (centralizado) e "END" (justificado à direita).

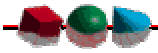
language: especifica o alfabeto: "en" para English,"en_US" para US english, "zh" para chinês, etc. Consulte o RFC 1766 para a lista completa.

size: especifica a altura dos caracteres em unidades VRML.

spacing: especifica o espaço entre linhas.

length: especifica o comprimento de cada string em unidades VRML, não em caracteres. Se o string for muito curto, é escalado, se for muito longo, é comprimido. Um valor zero indica que o string não deve ser nem escalado nem comprimido. Zero é o valor default.

maxExtent: limita, diminuindo se necessário, todos os strings.

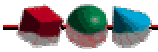


Sintaxe:

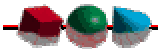
```
geometry Text
        {  string [ ]
           fontStyle { family ""
                       style “”
                       horizontal TRUE/FALSE
                       justify ""
                       language ""
                       leftToRight TRUE/FALSE
                       size □
                       spacing □
                       topToBottom TRUE/FALSE
                       leftToRight  TRUE/FALSE
                     }
           length [ ]
           maxExtent  □
}
```

Exemplo:

```
#VRML V2.0 utf8
#Text.wrl
#Sao mostrados tres textos com diferentes caracteristicas.

# Texto na posicao default
Shape
{  appearance Appearance
   {  material Material { diffuseColor 1.0 1.0 0.0 }
   }
   geometry Text
   {  string "Italic serif"
      fontStyle FontStyle
       {  size   1.7
         family "SERIF"
         style  "ITALIC"
         language "zh"
         leftToRight TRUE
      }
   }
}
```



```
# Texto centralizado
Transform
  {translation  0.0 3.0 0.0
   children
   [   Shape
       { appearance Appearance
         {  material Material { diffuseColor 0.0 1.0 0.0 }
         }
         geometry Text
         {  string ["Bold", "sans-serif"]
            fontStyle FontStyle
            {  size      1.4
               justify   "MIDDLE"
               family    "SANS"
               style     "BOLD"
               topToBottom TRUE
               horizontal TRUE
            }
         }
      }
   ]
}


# Texto alinhado pelo fim
Transform
{  translation -5.0 5.0 0.0
   children
   [  Shape
      {  appearance Appearance
         {  material Material{ diffuseColor 1.0 0.0 1.0 }
         }
         geometry Text
         {  string "Roman typewriter"
            fontStyle FontStyle
            {  size      0.9
               family    "TYPEWRITER"
               style     ""
               justify   "END"
               horizontal FALSE
            }
            maxExtent 42.5
         }
      }
   ]
}
```



6. Shape - Aparência

O node **Shape**, como foi definido no capítulo 3 e mostrado abaixo, possui dois campos: **appearance** (aparência) e **geometry** (geometria).

```
Shape
{   appearance Appearance # define a aparência
                {
                 ...
                }

   geometry   □   # define a geometria ou forma 3D
                 {
                   ....
                 }
}
```

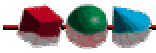
Nos capítulos 3 e 5 todas as dez formas geométricas que podem ser definidas no campo geometry foram estudadas. Neste capítulo será estudado o campo appearance.

Em appearance podem ser definidos os campos **material**, **texture** (que é composto dos campos **ImageTexture**, **MovieTexture** e **PixelTexture**) e **textureTransform**. Neste capítulo apenas três dos campos do appearance serão abordados, quais sejam:

material Material

texture ImageTexture

texture MovieTexture



6.1. Material

O campo **material Material** especifica cor, reflexão da luz e transparência das formas geométricas e só pode ser definido dentro do campo appearance do node Shape. Material tem seis campos:

diffuseColor: define a cor da geometria; este campo é ignorado quando alguma textura é usada.

emissiveColor: usado para definir objetos brilhantes.

ambientIntensity: especifica a quantidade de luz refletida pela geometria.

specularColor: define a cor dos “spots brilhantes” da geometria.

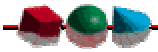
shininess: controla a intensidade do brilho dos “spots brilhantes”; pequenos valores representam brilho suave e valores altos definem brilhos intensos.

transparency: controla a trasparência da geoemtria; se um valor 0.0 é especificado, a geometria é totalmente opaca; o valor 1.0 indica que a geometria é transparente.

Os campos diffuseColor, emissiveColor e specularColor têm um valor RGB associado; os outros um valor em ponto flutuante entre 0.0 e 1.0.

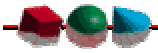
Sintaxe:

```
appearance Appearance
        { material Material
                { diffuseColor □□□
                  ambientIntensity □
                  emissiveColor □□□
                  specularColor □□□
                  shininess □
                  transparency □
                }
}
```

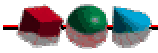


Exemplo:

```
#VRML V2.0 utf8
#Material.wrl
#Sao mostrados seis vasos com diferentes materiais.

#Definicao do ponto de vista inicial.
Viewpoint
{  position 0.0 3.0 15.0
   description "Vista inicial"
}

Group
{  children
   [  #Iluminacao
      PointLight
      {  location 0.0 10.0 -9.0
         ambientIntensity 0.2
      },
      PointLight
      {  location 0.0 10.0 9.0
         ambientIntensity 0.2
      },

      Shape
      {  appearance Appearance
         {  # Material - aluminio - vaso 1
            material Material
            {  ambientIntensity 0.3
               diffuseColor 0.30 0.30 0.50
               specularColor 0.70 0.70 0.80
               shininess 0.10
            }
         }
         #cria um vaso com a geometria Extrusion
         geometry  DEF vase Extrusion
         {  creaseAngle 1.57
            endCap FALSE
            solid  FALSE
            crossSection
            [  #Definicao do circulo de crossSection
               1.00  0.00,   0.92 -0.38,
               0.71 -0.71,   0.38 -0.92,
               0.00 -1.00,  -0.38 -0.92,
              -0.71 -0.71,  -0.92 -0.38,
              -1.00 -0.00,  -0.92  0.38,
              -0.71  0.71,  -0.38  0.92,
               0.00  1.00,   0.38  0.92,
               0.71  0.71,   0.92  0.38,
               1.00  0.00
            ]
            spine
            [  0.0 0.0 0.0,  0.0 0.6 0.0,
               0.0 1.0 0.0,  0.0 1.4 0.0,
               0.0 1.8 0.0,  0.0 2.2 0.0,
               0.0 2.6 0.0,  0.0 3.0 0.0,
               0.0 3.4 0.0,  0.0 3.8 0.0,
               0.0 4.2 0.0
            ]
```



```
      scale
      [  1.5  1.5,  1.95 1.95,
         2.0  2.0,  1.95 1.95
         1.8  1.8,  1.5  1.5
         1.2  1.2,  1.05 1.05,
         1.0  1.0,  1.05  1.05,
         1.3 1.3,
      ]
   }
},
Transform
{  translation -5.0 0.0 0.0
   children
   [  Shape
      {  appearance Appearance
         {  #Material - cobre - vaso 2
            material Material
            {  ambientIntensity 0.26
               diffuseColor 0.30 0.11 0.00
               specularColor 0.75 0.33 0.00
               shininess 0.08
            }
         }
         geometry USE vase
      }
   ]
},
Transform
{   translation 5.0 0.0 0.0
    children
    [   Shape
        { appearance Appearance
          {  #Material - ouro - vaso 3
             material Material
             {  ambientIntensity 0.4
                diffuseColor 0.22 0.15 0.00
                specularColor 0.71 0.70 0.56
                shininess 0.16
             }
          }
          geometry USE vase
       }
    ]
 }
 Transform
 {  translation -5.0 0.0 -5.0
    children
    [  Shape
       {  appearance Appearance
          {  # Material - vermelho metalico - vaso 4
             material Material
             {  ambientIntensity 0.15
                diffuseColor 0.27 0.0 0.0
                specularColor 0.61 0.13 0.18
                shininess 0.20
             }
          }
          geometry USE vase
       }
    ]
 }
```

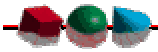


```
        Transform
        {  translation 0.0 0.0 -5.0
           children
            [ Shape
              {  appearance Appearance
                 {  # Material - plastico azul - vaso 5
                    material Material
                    {  ambientIntensity 0.10
                       diffuseColor 0.20 0.20 0.71
                       specularColor 0.83 0.83 0.83
                       shininess 0.12
                    }
                 }
                 geometry USE vase
              }
           ]
        },
        Transform
        {  translation 5.0 0.0 -5.0
           children
           [  Shape
              {  appearance Appearance
                 {  # Material - transparencia - vaso 6
                    material Material
                    {  ambientIntensity 0.5
                       diffuseColor 0.0 0.0 0.2
                       specularColor 1.0 1.0 1.0
                       shininess 0.50
                       transparency 0.5
                    }
                 }
                 geometry USE vase
              }
           ]
        }
    ]
}
```

6.2. ImageTexture

O campo **texture ImageTexture** especifica a localização da imagem que será utilizada para texturizar a geometria, assim como se a imagem será repetida verticalmente ou horizontalmente ao longo das faces da forma. Os campos presentes são:

url: especifica a localização da imagem; os formatos aceitos são JPG/JPEG - *Joint Photographic Experts Group* , GIF- *Graphical Interchange Format* e PNG- *Portable Network Graphics* . Podem ser definidas múltiplas localizações e o *browser* irá procurar pelos dados em ordem decrescente de endereços.



repeatS: TRUE indica que a imagem deve ser repetida verticalmente.

repeatT: TRUE indica que a imagem deve ser repetida horizontalmente.

Todos os campos são opcionais e valores default são aplicados quando os campos não são referenciados. Se a imagem não for localizada, nenhuma textura é aplicada.

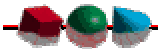
Quando ambos os campos repeatS e repeatT são TRUE a imagem é repetida duas vezes em cada direção.

Sintaxe:

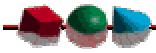
```
appearance Appearance
        { texture ImageTexture
            {  url [ ]
               repeatS TRUE/FALSE
               repeatT TRUE/FALSE
            }
        }
```

Exemplo:

```
#VRML V2.0 utf8
#ImageTexture.wrl
#Criacao de um vaso com flores. O vaso e mapeado com uma
#textura colorida e a textura das flores possui transparencia.

#Definicao do ponto de vista inicial
Viewpoint
{  position 0.0 5.0 18.0
   description "Vista inicial"
}

Group
{  children
   [  #Cria o vaso com a geometria Extrusion
      Shape
      {  appearance Appearance
         {  texture ImageTexture { url "imporsol.jpg"}
         }
         geometry Extrusion
         {  creaseAngle 1.57
            endCap FALSE
```



```
            solid  FALSE
            crossSection
            [  1.00  0.00,   0.92 -0.38,
               0.71 -0.71,   0.38 -0.92,
               0.00 -1.00,  -0.38 -0.92,
              -0.71 -0.71,  -0.92 -0.38,
              -1.00 -0.00,  -0.92  0.38,
              -0.71  0.71,  -0.38  0.92,
               0.00  1.00,   0.38  0.92,
               0.71  0.71,   0.92  0.38,
               1.00  0.00
            ]
            spine
            [  0.0 0.0 0.0,  0.0 0.6 0.0,
               0.0 1.0 0.0,  0.0 1.4 0.0,
               0.0 1.8 0.0,  0.0 2.2 0.0,
               0.0 2.6 0.0,  0.0 3.0 0.0,
               0.0 3.4 0.0,  0.0 3.8 0.0,
               0.0 4.2 0.0
            ]
            scale
            [  1.5  1.5,  1.95 1.95,
               2.0  2.0,  1.95 1.95
               1.8  1.8,  1.5  1.5
               1.2  1.2,  1.05 1.05,
               1.0  1.0,  1.05  1.05,
               1.3 1.3,
            ]
         }
      },

      # Cria um cubo e aplica uma imagem com tulipas,
      # simulando as flores do vaso
      Transform
      {  translation 0.0 6.8 0.0
         children
            Shape
            {  appearance Appearance
               {  #Textura com transparencia
                  texture DEF flowers ImageTexture
                  {   url "imtulipas.jpg"
                  }
               }
               geometry  Box{ size 5.0 5.0 5.0 }
            }
      }

      #Cria mais um cubo com as flores
      Transform
      {  translation 0.0 7.8 0.0
         rotation     0.0 1.0 0.0 -0.75
            children
               Shape
               {  appearance Appearance
                  {  texture USE flowers
                  }
               geometry  Box { size 3.0 5.0 3.0 }
            }
       }
   ]
}
```



6.3. MovieTexture

O campo **texture MovieTexture** especifica a localização de um filme que será utilizado para texturizar a geometria. O filme deve estar no formato MPEG e pode ser repetido verticalmente ou horizontalmente ao longo das faces da forma. Os campos presentes são:

loop: especifica se o filme deve ser apresentado repedidamente, sem parar.

speed: especifica o quão rápido o filme deve ser apresentado; por exemplo, se a velocidade (speed) for 2 o filme será duas vezes mais rápido; valores negativos fazem com que o filme seja apresentado de trás para a frente.

startTime: especifica o tempo de início da apresentação do filme em segundos; o valor deste campo é o número de segundos desde a meia-noite de 1º de janeiro de 1970.

stopTime: especifica o tempo do final da apresentação do filme em segundos; o valor deste campo é o número de segundos desde a meia-noite de 1º de janeiro de 1970.

url: especifica a localização do filme; podem ser definidas múltiplas localizações e o browser irá procurar pelos dados em ordem decrescente de endereços.

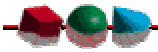
repeatS: TRUE indica que o filme deve ser repetido verticalmente.

repeatT: TRUE indica que o filme deve ser repetido horizontalmente.

Em VRML o mundo foi criado à meia-noite de 1º de janeiro de 1970. Alguns dizem que a razão da escolha desta data foi para coincidir com o nascimento do sistema operacional UNIX.

Se o loop é marcado como TRUE e o startTime for maior ou igual a stopTime, o filme vai ser repetido para sempre. Entretanto, se o startTime for menor que o stopTime, o filme vai parar assim que o stopTime for alcançado.

Se o startTime for maior ou igual a stopTime, então o filme vai começar tão logo o startTime for alcançado. Note que alguns *browsers* somente começam o filme se o startTime for maior que o stopTime.



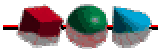
Todos os campos são opcionais e valores default são aplicados quando os campos não são referenciados. Se o filme não for localizado, nenhuma textura é aplicada.

Sintaxe:

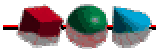
```
appearance Appearance
        { texture MovieTexture
                { loop TRUE/FALSE
                  speed □
                  startTime □
                  stopTime □
                  url [ ]
                  repeatS TRUE/FALSE
                  repeatT TRUE/FALSE
                }
        }
```

Exemplo:

```
#VRML V2.0 utf8
#MovieTexture.wrl
#Usa o vaso com flores criado em ImageTexture.wrl,
#mas no lugar das flores apresenta um filme

#Definicao do ponto de vista inicial
Viewpoint
{  position 0.0 5.0 18.0
   description "Vista inicial"
}


Group
{  children
   [  Shape
      {       appearance Appearance
              {
                   texture ImageTexture
                   {  url "imporsol.jpg"
                   }
              }
```



```
         geometry Extrusion
         {  creaseAngle 1.57
            endCap FALSE
            solid  FALSE
            crossSection
            [  1.00  0.00,   0.92 -0.38,
               0.71 -0.71,   0.38 -0.92,
               0.00 -1.00,  -0.38 -0.92,
              -0.71 -0.71,  -0.92 -0.38,
              -1.00 -0.00,  -0.92  0.38,
              -0.71  0.71,  -0.38  0.92,
               0.00  1.00,   0.38  0.92,
               0.71  0.71,   0.92  0.38,
               1.00  0.00
            ]
            spine
            [  0.0 0.0 0.0,  0.0 0.6 0.0,
               0.0 1.0 0.0,  0.0 1.4 0.0,
               0.0 1.8 0.0,  0.0 2.2 0.0,
               0.0 2.6 0.0,  0.0 3.0 0.0,
               0.0 3.4 0.0,  0.0 3.8 0.0,
               0.0 4.2 0.0
            ]
            scale
            [  1.5  1.5,  1.95 1.95,
               2.0  2.0,  1.95 1.95
               1.8  1.8,  1.5  1.5
               1.2  1.2,  1.05 1.05,
               1.0  1.0,  1.05  1.05,
               1.3 1.3,
            ]
         }
      },

      #Cria um cubo em cima do vaso e passa o filme em suas faces
      Transform
      {  translation 0.0 6.8 0.0
         children
            Shape
            { appearance Appearance
                        { texture MovieTexture
                          { url "usuario.mpeg"
                            loop TRUE
                            speed 1
                          }
                        }

               geometry  Box{ size 5.0 5.0 5.0 }
            }
      }

   ]
}
```



Bibliografia Consultada

JAMSA, K, SCHMAUDER, P., YEE, N. **VRML biblioteca do programador.** Makron Books, 1999.

AMES, A. L., NADEAU, D. R., MORELAND, J. L. **VRML 2.0 sourcebook**. John Wiley & Sons, Inc., 1997.

HARTMAN, J., WERNECKE, J. **The VRML 2.0 handbook - building moving worlds on the web**. Addison-Wesley Publishing Company, 1996.

MARRIN, C., CAMPBELL, B. **Teach yourself VRML 2 in 21 days**, Sams.net Publishing, 1997.

PESCE, M. **VRML browsing and building cyberspace**. New Riders Publishing, 1995.

LEA, R., MATSUDA, K., MIYASHITA, K. **Java for 3D and VRML worlds**. New Riders Publishing, 1996.

http://www.terravista.pt/enseada/1527/vrml_ok.htm

http://www.di.ufpe.br/~if291/documentos/vrmlsibgrapi97/toc.htm

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

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

http://www.inf.pucrs.br/~pinho/CG/Aulas/Vrml/Vrml2/vrml2Pinho.htm