7
Referências bibliográficas

AMES, A. L.; NADEAU, D. R.; MORELAND, J. I. **The VRML 2.0 sourcebook**. 2nd. Edition. Canada: John Wiley & Sons, 1997.

ANSYS. **APDL Programmer's Guide.** 2000

BERNERS-LEE, T.; FIELDING, R.;FRYSTYK, H. **Hypertext Transfer Protocol**. 1996. Disponível em: http://www.w3.org/Protocols/HTTP/1.0/draft-ietf-http-spec.html. Acesso em: 20 dez. 2003.

BSISG; **Standard Template for Electronic Publishing**. 1998. Disponível em: http://www.crosswire.org/bsisg/. Acesso em: 20 dez. 2003

BUDGELL, P. C. **Finite Element Analysis and Optimization Introduction**. 1998. Disponível em: <http://www3.sympatico.ca/peter_budgell/FEA_intro.html>. Acesso em: 25 nov. 2003.

FELICÍSSIMO, C. H.; SALLES, M. A. V. **Um Ambiente Colaborativo para Desenvolvimento de Projetos de Engenharia de Grande Porte**. 29 f. Relatório técnico da Easycae, Rio de Janeiro, 2002.

GOODMAN, DANNY. **JavaScript Bible**, 4th Edition. John Wiley & Sons, 2001.

GRAHAM, IAN. **Introduction do HTML**. 2000. Disponível em: <http://www.utoronto.ca/webdocs/HTMLdocs/NewHTML/htmlindex.html>. Acesso em: 30 nov. 2003.

HARTMAN, J.; WERNECKE, J. **The VRML 2.0 Handbook—Building Moving Worlds on the Web**. Addison-Wesley, 1996.

KEMMERER, SHARON J.; **Initial Graphics Exchange Specification**. 2001. Disponível em: http://www.nist.gov/iges/. Acesso em: 20 dez. 2003.

KENNETH, H. et al. **The Finite Element Method for Enginneers**. 4th Edition. Interscience, 2001.

MARUDUR, K. S. **Concurent-Interactive Design and Analysis using the Internet and VRML**. 1998. 108 f. Master of Science Thesis - University of Oklahoma, Norman, Oklahoma, 1998.

MAGALHÂES, L. P.; RAPOSO, A. B.; TAMIOSSO, F. S. **VRML 2.0 – An Introductory View by Examples**. Disponível em: <http://www.dca.fee.unicamp.br/~leopini/tut-vrml/vrml-tut.html>. Acesso em: 24 out. 2003.

MSDN. **C++ Language Reference**. 2001. Disponível em: http://msdn.microsoft.com/library/en-us/vclang/html/_vclang_home.asp. Acesso em: 20 dez. 2003.

OBJECT MANAGEMENT GROUP. **C++ Language Mapping Specification**. 2003. Disponível em: http://www.omg.org/cgi-bin/doc?formal/03-06-03. Acesso em: 20 dez. 2003.

RAGGETT, DAVE. **HTML 4.01 Specification**. 1999. Disponível em: http://www.w3.org/TR/html401/. Acesso em: 20 dez. 2003.

RANGA, KARTHIK. **3-D Finite Element Analysis over the Internet using Java and VRML**. 2000. 151 f. Master of Science Thesis - School of Aerospace and Mechanical Engineering, University of Oklahoma, Norman, Oklahoma, 2000.

RAPOSO, A. B. et al. **Software Livre para Computação Gráfica e Animação por Computador**. 2000. 59 f. Tutorial – SIBGRAPI´2000, 2000.

ROEHL, BERNIE et al. **Late Night VRML 2.0 with Java**. Hightstown, NJ, USA: Ziff-Davis, 1997.

ROSENBLUM, L. et al. **Scientific Visualization: Advances and Challenges**. San Diego, CA: Academic Press, 1994.

WATT, A. **3D Computer Graphics**. Third Edition. USA: Addison-Wesley, 2000.

8
Apêndice A – VRML

8.1.
Estrutura hierárquica da cena VRML

O paradigma para a criação de mundos VRML é baseado em conceitos de nós, que são conjuntos de abstrações de objetos e de certas entidades do mundo real, tais como formas geométricas, luz e som. Nós, são os componentes fundamentais de uma cena VRML porque esta é constituída a partir da disposição, combinação e interação entre os nós.

Assim como em Java 3D, um mundo VRML é um grafo hierárquico em forma de árvore. As hierarquias são criadas através de nós de agrupamento, os quais contêm um campo chamado “children” que engloba uma lista de nós filhos. Há vários tipos de nós em VRML (Hartman & Wernecke, 1996).

A.Agrupamento

Nós de agrupamento criam a estrutura hierárquica da cena e permitem que operações sejam aplicadas a um conjunto de nós simultaneamente. Alguns exemplos desse tipo de nó são:

Anchor. É um nó de agrupamento que recupera o conteúdo de um URL quando o usuário ativa alguma geometria contida em algum de seus nós filhos (clica com o mouse sobre eles, por exemplo). É o nó que cria enlaces com outros mundos VRML, páginas HTML, ou qualquer outro tipo de documento presente no referido URL. Um Anchor é definido com os seguintes valores “default” para seus parâmetros:

```
Anchor {
        # objetos da cena que contém hyperlinks para outros arquivos.
        children [ ]
        description " "
        parameter [ ]
        url [ ] # url do arquivo a ser carregado
        bboxCenter 0 0 0
        bboxSize -1 -1 -1 }
```

Transform. É um nó de agrupamento que define um sistema de coordenadas para seus filhos que está relacionado com o sistema de

coordenadas de seus pais. Sobre este sistema de coordenadas podem ser realizadas operações de translação, rotação e escalonamento. Os seguintes valores “default” são definidos:

```
Transform {
        bboxCenter 0 0 0
        bboxSize -1 -1 -1
        translation 0 0 0
        rotation 0 0 1
        scale 1 1 1
        scaleOrientation 0 0 1
        center 0 0 0
        # nós filhos, que são afetados pelas transformações
        # especificadas neste nó.
        children [ ] }
```

Group. É um nó de agrupamento que contém um número qualquer de filhos. Ele é equivalente ao nó “Transform” sem os campos de transformação. A diferença básica entre o “Group” e o “Transform” é que o primeiro é usado quando se deseja criar um novo objeto constituído da combinação de outros. Quando se deseja agrupar espacialmente os objetos, ou seja, posicioná-los em certa região da cena, usa-se o nó “Transform”.

B.Geométrico

Define a forma e a aparência de um objeto do mundo. O nó “Shape”, em particular, possui dois parâmetros: “geometry”, que define a forma do objeto e “appearance”, que define as propriedades visuais dos objetos (material ou textura). Alguns exemplos de nós geométricos são:

Box. Este nó geométrico representa um sólido retangular (uma “caixa”) centrado na origem (0, 0, 0) do sistema de coordenadas local e alinhado com os eixos cartesianos. O campo “size” representa as dimensões do sólido nos eixos x, y e z. Exemplo:

```
Box { size 2 2 2 }
```

Cone. Representa um cone centrado na origem do sistema local de coordenadas cujo eixo central está alinhado com o eixo y.

Cylinder. Define um cilindro centrado na origem do sistema de coordenadas e com eixo central ao longo do eixo y.

Sphere. Especifica uma esfera centrada na origem.

Text. Desenha uma ou mais “strings” de texto em um estilo específico.

IndexedLineSet e ***IndexedFaceSet.*** Estes nós descrevem, respectivamente, objetos 3D a partir de segmentos de reta e polígonos cujos vértices estão localizados em coordenadas dadas.

C.Câmera

O nó “Viewpoint” define, entre outros parâmetros, a posição e orientação da câmera (ponto de vista do usuário). Este tipo de nó é chamado “bindable” porque apenas um pode estar ativo na cena.

D.Iluminação

Luzes em VRML não são como luzes no mundo real. No mundo real, luzes são objetos físicos que emitem luz e que podem ser vistos, assim como a luz por ele emitida. Em VRML, um nó de iluminação é descrito como parte do mundo mas não cria automaticamente uma geometria para representá-lo. Para que uma fonte de luz em uma cena seja um objeto visível é necessário criar uma geometria e colocá-la em um determinado local na cena. Existem três tipos de nós de iluminação em VRML:

DirectionalLight. Este nó define uma fonte de luz direcional com raios paralelos.

PointLight. Define uma fonte de luz pontual em um local 3D fixo. Este tipo de fonte ilumina igualmente em todas as direções.

SpotLight. Define um cone direcional de iluminação.

8.2. Propotipagem e reuso em VRML

Os mecanismos de prototipagem da linguagem VRML permitem definir um novo tipo de nó baseado na combinação de nós já existentes. É permitida inclusive, a criação de cenas distribuídas, pois o subgrafo (protótipo) pode ser definido em um arquivo remoto cujo URL é conhecido.

Atribuindo-se um nome a um nó através da palavra DEF, pode-se futuramente referenciá-lo através da palavra USE. Sendo assim, caso seja necessária a reutilização de um mesmo nó várias vezes em uma cena, é mais eficiente atribuir-lhe um nome na primeira vez que ele é descrito e posteriormente referenciá-lo por este nome. Essa técnica torna o arquivo menor e mais fácil de ler, diminuindo o tempo necessário para carregar a cena.

Em resumo, VRML permite o encapsulamento (protótipos) e reutilização de subgrafos da cena. O exemplo a seguir mostra uma cena VRML simples (2 cones verdes), utilizando alguns dos conceitos apresentados até aqui (a primeira linha do arquivo deve ser sempre #VRML V2.0 utf8 e os comentários devem começar com #):

```
#VRML V2.0 utf8
# Posição da câmera (ou observador)
  Viewpoint { position 0 0 10  }
# Fonte de luz pontual
  PointLight {
        ambientIntensity 0
        attenuation 1 0 0
        color 1 1 1 # luz branca
        intensity .2
        location 0 0 2 # posição da fonte
        on TRUE # está "ligada"
        radius 20
        }
    Transform {
  # Um cone é definido e posicionado na origem (valor default = 0 0 0)
        children [
              DEF Cone_Verde Shape {
                    geometry Cone {} # cone
                    appearance Appearance
                          {       material Material
                                { diffuseColor 0 1 0 # cor verde }
                          }
              }
              # reutilização do cone
              Transform { translation 2 0 3 # posicionamento em (2 0 3)
                    children USE Cone_Verde }
        ]
    }
```

Na figura 57 é mostrado o resultado do código acima que descreve dois cones verdes, utilizando os conceitos de encapsulamento (protótipos) e reutilização de subgrafos.

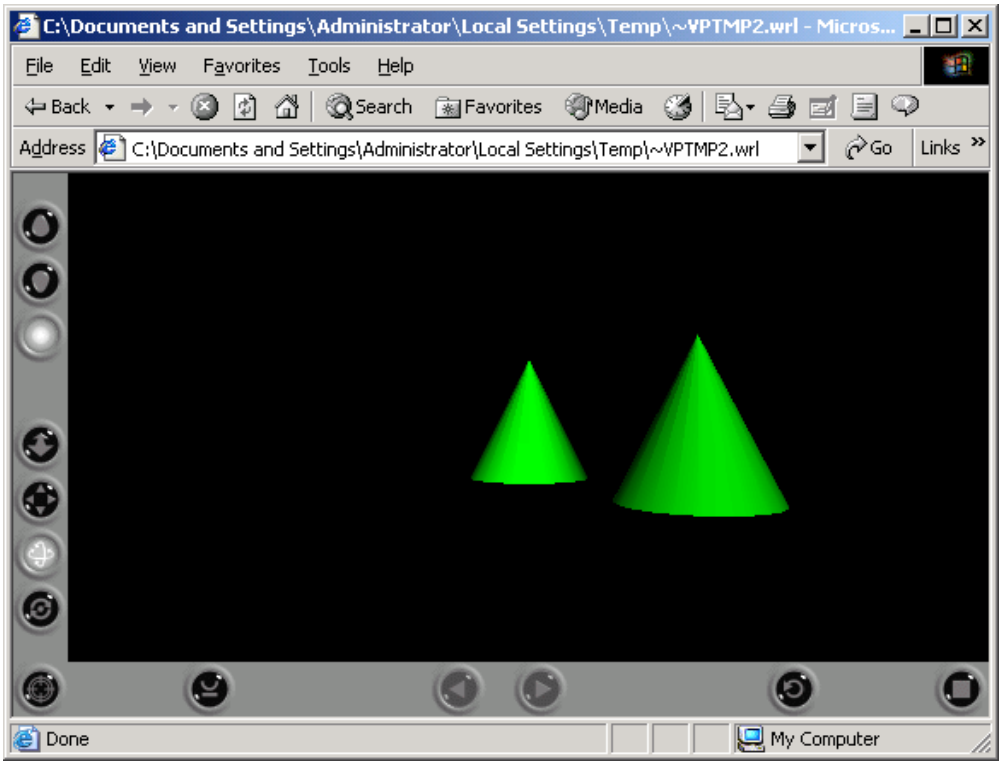

Figura 57 - Cena em VRML utilizando encapsulamento e reutilização de subgrafos.

8.3. Tipos de parâmetros e encaminhamento de eventos em VRML

Cada nó VRML define um nome, um tipo e um valor “default” para seus parâmetros. Estes parâmetros diferenciam um nó de outros de mesmo tipo (por exemplo, o raio de uma esfera a diferencia de outra que nós queremos ver diferente). Há dois tipos de parâmetros possíveis: campos (fields) e eventos (events).

Campos, podem ser modificáveis (exposedFields) ou não (fields). Um exposedField é, na verdade, uma composição de um evento de entrada, um evento de saída e o valor do campo (exposedField = eventIn + field + eventOut).

Os eventos podem ser enviados para outros nós por um parâmetro do tipo eventOut e recebidos por um eventIn.

Eventos sinalizam mudanças causadas por “estímulos externos” e podem ser propagados entre os nós da cena por meio de encaminhamentos (Routes) que conectam um eventOut de um nó a um eventIn de outro nó, desde que eles sejam eventos do mesmo tipo.

8.4.
Sensores e interpoladores em VRML

Os nós sensores e interpoladores são especialmente importantes porque são os responsáveis pela interatividade e dinamismo dos mundos VRML.

Os sensores geram eventos baseados nas ações do usuário. O nó “TouchSensor”, por exemplo, detecta quando o usuário aciona o mouse sobre um determinado objeto (ou grupo de objetos) da cena. O “ProximitySensor”, por sua vez, detecta quando o usuário está navegando em uma região próxima ao objeto de interesse. Os sensores são responsáveis pela interação com o usuário, mas não estão restritos a gerar eventos a partir de ações dos mesmos. O “TimeSensor”, por exemplo, gera automaticamente um evento a cada pulso do relógio. Os eventOuts gerados pelos sensores podem ser ligados a outros eventIns da cena, iniciando uma animação, por exemplo.

A seguir alguns sensores são descritos:

TimeSensor. O nó TimeSensor gera eventos como passos de tempo e em conjunto com interpoladores pode produzir animações.

```
TimeSensor {    cycleInterval 1
                enabled TRUE
                loop FALSE
                startTime 0
                stopTime 0
            }
```

O campo cycleInterval determina a duração, em segundos, de cada ciclo, devendo ser maior que zero. Se o campo enabled for TRUE os eventos relacionados ao tempo são gerados caso outras condições sejam encontradas e se for FALSE os eventos relacionados ao tempo não são gerados. O campo loop indica que o sensor de tempo deve repetir indefinidamente ou deve parar depois de um ciclo. O campo startTime indica quando a geração de eventos se inicia e o campo stopTime quando a geração de eventos pára.

TouchSensor. O nó TouchSensor detecta quando um objeto do grupo do seu pai é ativado (clique do mouse, por exemplo). Esse sensor gera um evento de saída chamado touchTime que pode disparar um TimeSensor, iniciando uma animação.

```
TouchSensor {
enabled TRUE
            }
```

O campo enabled indica se o sensor está ou não habilitado para enviar e receber eventos.

ProximitySensor. O nó ProximitySensor gera eventos quando o usuário entra, sai e/ou se move em uma região do espaço. O sensor de proximidade é

habilitado ou desabilitado pelo envio de um evento enabled com o valor TRUE ou FALSE.

VisibilitySensor. O nó VisibilitySensor detecta quando certa parte do mundo (área ou objeto específico) torna-se visível ao usuário. Quando a área está visível, o sensor pode ativar um procedimento ou animação.

A forma mais comum de se criar animações é usando keyframes (quadros-chave), especificando os momentos-chave na seqüência da animação. Os quadros intermediários são obtidos através da interpolação dos quadros-chave. Nós interpoladores servem para definir este tipo de animação, associando valores-chave (de posição, cor, etc.) que serão linearmente interpolados. Alguns exemplos de interpoladores são:

PositionInterpolator. O nó PositionInterpolator permite realizar uma animação keyframe em uma localização no espaço 3D. Exemplo:

```
PositionInterpolator {
                key [0, .5, 1] # instantes de tempo
                keyValue [ 0 0 0, 0 10 0, 0 0 0 ]
                } # valores-chave associados aos instantes
```

CoordinateInterpolator, ColorInterpolator, OrientationInterpolator. De forma similar interpolam, respectivamente, uma lista de coordenadas, valores de cor e valores de rotação.

Os eventos gerados por sensores e interpoladores, ligados a nós geométricos, de iluminação ou de agrupamento, podem definir comportamentos dinâmicos para os elementos do ambiente. Na figura 58 há um exemplo típico, que é encaminhar um TouchSensor em um TimeSensor, causando o disparo do relógio quando o usuário clicar sobre um objeto. O TimeSensor, por sua vez, é encaminhado em um interpolador, enviando para ele valores de tempo para a função de interpolação. O interpolador então é encaminhado em um nó geométrico, definindo as alterações deste objeto.

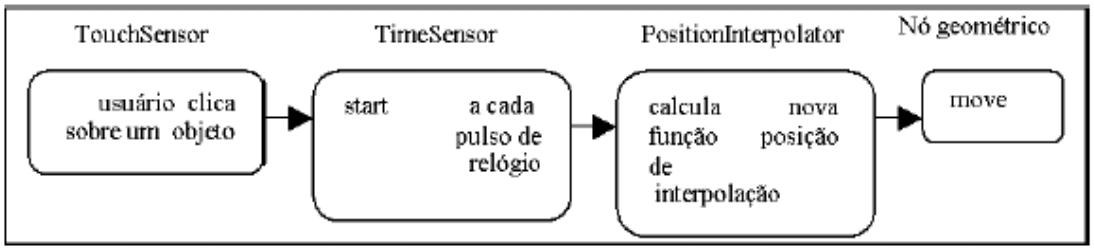

Figura 58 - Encaminhamento de eventos em uma interação típica de VRML.

A seguir é mostrado um exemplo de utilização de sensores e interpoladores.

```
#VRML V2.0 utf8
# Uso de sensores e interpoladores.
# Quando a bola for tocada (clique do mouse sobre ela) o texto irá mover-se na
# horizontal e quando a caixa for tocada o texto irá mover-se na vertical.
Viewpoint { position 0 0 50 } # observador
Group {
children [
        Transform {
          translation -4 8 0
                children [
                  Shape {
                        geometry Sphere { radius 1.5 } # bola
                                appearance Appearance {
                                        material Material {
                                        diffuseColor .73 .56 .56
                                        }
                                }
                        }
                  DEF STOUCH TouchSensor {} # sensor da bola
          ]
        }
  Transform {
          translation 4 8 0
          children [
                  Shape {
                        geometry Box { size 2 2 2 } # caixa
                                appearance Appearance {
                                        material Material {
                                        diffuseColor 0 1 0
                                        }
                                }
                        }
                  DEF BTOUCH TouchSensor{} # sensor da caixa
                ]
  }
  # Sensores de tempo
  DEF XTIMERH TimeSensor { cycleInterval 2 }
  DEF XTIMERV TimeSensor { cycleInterval 2 }

  # Interpoladores
  # Horizontal
  DEF ANIMAH PositionInterpolator {
                                        key [ 0, .25, .5, .75, 1 ]
                                        keyValue [ 0 0 0, 8 0 0, 16 0 0, -8 0 0, 0 0 0 ]
                                        }
        # Vertical
        DEF ANIMAV PositionInterpolator {
                                        key [ 0, .25, .5, .75, 1 ]
                                        keyValue [ 0 0 0, 0 -8 0, 0 -16 0, 0 -8 0, 0 0 0 ]
                                        }
  # Texto
  DEF SFORM Transform {
                  children Shape {
                        geometry Text {
                        string ["Virtual" ]
                        fontStyle FontStyle {
                                style "BOLD"
                                justify "MIDDLE"
                        }
                        length[7]
                        maxExtent 20
                        }
```

```
                    }
                }
            ]  # fecha priemiro children
        }  # fecha primeiro Group
        # Toque na esfera inicia relógio
          ROUTE STOUCH.touchTime TO XTIMERH.set_startTime
        # Valores do relógio entram no interpolador horizontal
          ROUTE XTIMERH.fraction_changed TO ANIMAH.set_fraction
        # Valores gerados pelo interpolador transladam texto horizontalmente
          ROUTE ANIMAH.value_changed TO SFORM.set_translation
        # Procedimento similar para a caixa
          ROUTE BTOUCH.touchTime TO XTIMERV.set_startTime
          ROUTE XTIMERV.fraction_changed TO ANIMAV.set_fraction
          ROUTE ANIMAV.value_changed TO SFORM.set_translation
```

Na figura 59 é mostrado o resultado do código acima que descreve a utilização em VRML do encaminhamento de eventos, utilizando sensores e interpoladores. A interação real não poderá ser visualizada no papel, mas poderia ser explorada completamente copiando o código acima em um arquivo com extensão wrl e logo abri-lo em algum browser de Internet. O efeito vai ser o seguinte: quando se faz click na bola, logo o texto Virtual vai se movimentar horizontalmente e quando se faça click no cubo, logo o texto Virtual vai se movimentar verticalmente, ambos os casos por um segundo de tempo.

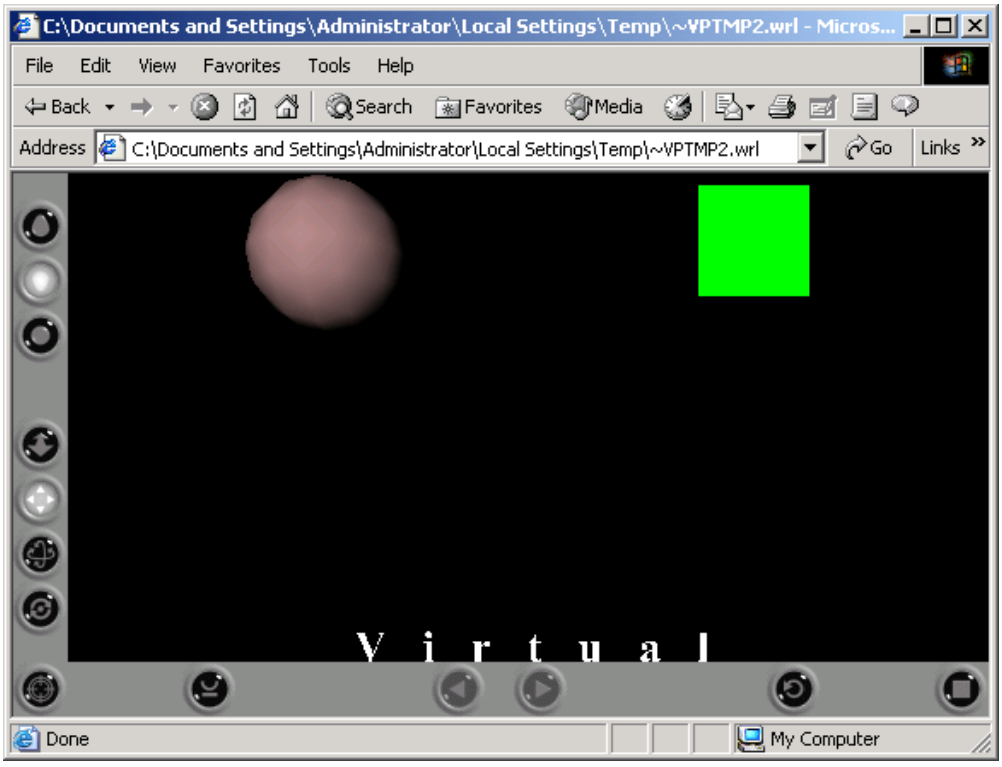

Figura 59 - Cena em VRML com encaminhamento de eventos, sensores e interpoladores.

8.5.
O Nó Script em VRML

Apesar de ser um recurso poderoso, o encaminhamento de eventos entre os nós não é suficiente para o tratamento de várias classes de comportamento. Por exemplo, não é possível escolher entre duas trajetórias pré-definidas (lógica de decisão). Para superar esta limitação, VRML define um nó especial chamado Script, que permite conectar o mundo VRML a programas externos, onde os eventos podem ser processados. Este programa externo, teoricamente, pode ser escrito em qualquer linguagem de programação, desde que o browser a suporte. Na prática, apenas Java e JavaScript são usadas.

O nó Script é ativado pelo recebimento de um evento. Quando isso ocorre, o browser inicia a execução do programa definido no campo url do nó Script. Este programa é capaz de receber, processar e gerar eventos que controlam o comportamento do mundo virtual. Por meio do nó Script é possível usar técnicas bem mais sofisticadas que a intepolação linear para a geração de animações.

```
Script {
        ExposedField MFString url [name.class ]     # url do programa
        Field SFBool mustEvaluate FALSE             # browser atrasa o envio dos
                                                            # eventos ao script até serem
                                                            # necessários ao browser
        Field SFBool DirectOutput FALSE             # script tem acesso a
                                                            # informação via seus eventos
        }
```

A seguir é dado um exemplo utilizando o nó Script, cujo encaminhamento é representado graficamente na figura 60.

```
#VRML V2.0 utf8
# Este exemplo define um cubo e uma esfera. O cubo muda de cor com um toque. A esfera
# está continuamente rotacionando a menos que o cubo tenha mais de 50% de cor vermelha
        Group {
          children [
                    # Esfera
                    DEF Sph Transform {
                              children Shape {
                                        geometry Sphere {}
                                        appearance Appearance {
                                                  material Material { diffuseColor 1 0 0 }
                                        }
                              }
                    }
          Transform {
                              translation -2.4 .2 1
                    rotation 0 1 1 .9
                    children [
                              # Cubo
                              Shape {
                                        geometry Box {}
                                        appearance Appearance {
```

```
                                material DEF MATERIAL Material {}
                        }
                }
                DEF TS TouchSensor {} # sensor associado ao cubo
        ]
  }
  # Arquivo script associado
        DEF SC Script {
                url "extouchcube.class"
                field SFColor currentColor 0 0 0
                eventIn SFColor colorIn
                eventOut SFBool isRed
        }
  ]
}

DEF myColor ColorInterpolator {
                keys [0.0, 0.3, 0.6, 1.0 ]
                values [1 0 0 , 0 1 0, 1 0 0, 0 0 1]     # red, green, red, blue
                        }

DEF myClock TimeSensor {
                cycleInterval 10 # 10 segundos de animação red -> green -> red -> blue
                }

DEF XTIMER TimeSensor {
                        loop TRUE
                cycleInterval 5
                }
DEF ENGINE OrientationInterpolator {
                keys [ 0, .5, 1]
                values [ 0 1 0 0, 0 1 0 3.14, 0 1 0 6.28]
                }
# Toque no cubo inicia alteração de cor no mesmo
# sensor -> relógio -> interpolador -> cor do cubo
  ROUTE TS.touchTime TO myClock.set_startTime
  ROUTE myClock.fraction TO myColor.set_fraction
  ROUTE myColor.value_changed TO MATERIAL.diffuseColor
# Alteração de cor do cubo enviada para Script
  ROUTE myColor.value_changed TO SC.colorIn
# Evento gerado pelo nó Script habilita ou desabilita relógio
  ROUTE SC.isRed TO XTIMER.enabled
# Relógio determina rotação da esfera
# relógio -> interpolador -> movimento da esfera
  ROUTE XTIMER.fraction TO ENGINE.set_fraction
  ROUTE ENGINE.value_changed TO Sph.set_rotation
```

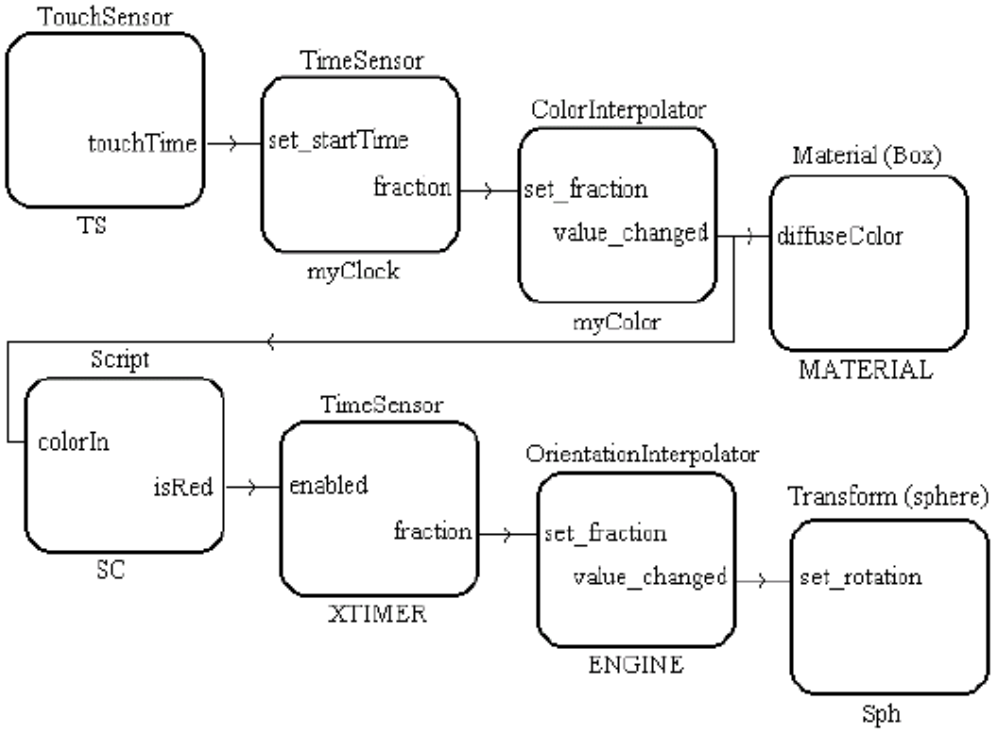

Figura 60 - Encaminhamento de eventos para o exemplo usando o nó Script.

O programa Java que implementa a funcionalidade do nó Script do exemplo acima (extouchcube.java) é mostrado a seguir.

```
import vrml.*;
import vrml.field.*;
import vrml.node.*;
public class extouchcube extends Script
{                                       // declarações dos campos e eventOuts usados
  private SFColor currentColor;
  private SFBool isRed;
  public void initialize() {            // alocação dos campos e eventos
            currentColor = (SFColor) getField("currentColor");
            isRed = (SFBool) getEventOut("isRed");
          }
                                        // método chamado no recebimento do evento colorIn
  public void processEvent(Event e) {
            currentColor.setValue((ConstSFColor)e.getValue());
  }

  public void eventsProcessed() {
                  // componente vermelho da cor maior que 50% -> envio de isRed =
            FALSE,
            // o que desabilita o relógio que controla a rotação da esfera da cena
  if (currentColor.getRed() >= 0.5)
  isRed.setValue(false);
  else isRed.setValue(true);
  }
}
```

9
Apêndice B - Macros na linguagem APDL do ANSYS

A macro relatório.mac:

Macro que chama todas as outras macros, que geram os diferentes arquivos texto, que são entradas do Gerador.

```
!! Macro que chama as outras macros de relatorios
 flag2=1              !! Seta flag para 1
 !*IF,ARG1,EQ,0,THEN        !! Se não for dado o argumento, da uma
                            !! mensagem de erro e seta flag para zero.
 !
 ! *MSG,ERROR
 ! You have to pass an argument to this macro. Options are 3 or 4.
 !
 ! flag2=0
 !
 ! *ENDIF
 *IF,flag2,EQ,1,THEN
 allele,telem                   !! Lista elementos
 allnodes             !! Lista todos os nos
 kpoints                !! Lista keypoints
 lines                          !! Lista nodes por linha
 areas                          !! Lista elementos por area
 alldof             !! Deslocamentos
 res                            !! Lista 3 colunas de resultados
 *ENDIF
```

A macro allele.mac:

Macro que gera o arquivo texto ALLELEM.txt

```
!! Lista elementos com atributos e elementos adjacentes
csys,0                          ! Seta o sistema de coordenadas para zero
asel,all                                ! Seleciona todas as areas
ALLSEL,ALL,ALL                  ! seleciona tudo abaixo das areas
*get,numele,elem,0,count                ! Conta o numero de elementos
*CFOPEN,ALLELEM,txt                     ! Abre o arquivo texto
*CFWRITE,ELEM,MATERIAL,TYP,REL,NODES
index=1
*DO,rep,1,numele,1                  ! Loop que percorre todos os elementos
   _indext_='%index%'
*GET,mat,ELEM,index,attr,mat
*GET,type,ELEM,index,attr,type            ! Pega o tipo    *GET,real,ELEM,index,attr,real
*cfwrite,_indext_,mat,type,real,nelem(index,1),nelem(index,2),nelem(index,3),nelem(index,4)
   index=1+index
*ENDDO
```

A macro allenodes.mac:

Macro que gera o arquivo texto ALLNODES.txt

```
!! Lista todos os nos com respectivas coordenadas

*CFOPEN,ALLNODES,txt                          ! Abre arquivo texto
csys,0                                        ! Seta sistema de coordenadas para padrão
asel,all                                      ! Seleciona todas as areas
ALLSEL,ALL,ALL                                ! Seleciona tudo abaixo das areas
*get,numnode,node,0,count                     ! Pega o numero de nos total
*DO,reps,1,numnode,1                          ! Loop que percorre todos os nos
  _reps_='%reps%'
  *cfwrite,_reps_,nx(reps),ny(reps),nz(reps)  ! Escreve no e suas coordenadas
*ENDDO
*CFCLOSE
```

A macro areas.mac:

Macro que gera o arquivo texto AREAS.txt

```
!!!!!!!!!!   Lista os elementos contidos em uma area   !!!!!!!!!!!!!!!

*CFOPEN,AREAS,txt
ALLSEL,ALL,ALL
asel,all
*get,anum,area,0,count
*DO,area,1,anum,1
  *VWRITE
Area,lenght
        *VWRITE
Components

  asel,s,area,,area
  esla,s,1
  *get,minode,elem,0,num,min
  *get,maxnode,elem,0,num,max
  *get,numele,elem,0,count

  _area_='%area%'
  *CFWRITE,_area_,numele
    element=minode
    *DO,reps,1,numele,1
        *IF,element,LE,maxnode,THEN
            _element_='%element%'
            *cfwrite,_element_
            element=elnext(element)
        *ENDIF
  *ENDDO
*ENDDO
*CFCLOSE
```

A macro kpoints.mac:

Macro que gera o arquivo texto KLIST.txt

```
ksel,all                                ! Seleciona todos os keypoint
csys,0
*CFOPEN,KPLIST,txt
*CFWRITE,KP,COORDINATES
*get,minkp,KP,0,NUM,MIN
*get,maxkp,KP,0,NUM,MAX
flag=1
```

```
*DO,i,minkp,maxkp,1
  it='%i%'
  *If,flag,EQ,1,THEN
  *CFWRITE,it,KX(i),KY(i),KZ(i)
  *IF,KPNEXT(i),EQ,0,THEN
        flag=0
  *ENDIF
  *ENDIF
*ENDDO
```

A macro lines.mac:

Macro que gera o arquivo texto LINES.txt

```
!!!!!!!!!!   Lista os elementos contidos em uma linha   !!!!!!!!!!!!!!!
lsel,all
*get,lnum,line,0,count
*CFOPEN,LINES,txt
*DO,linha,1,lnum,1
  *VWRITE
Line,lenght
        *VWRITE
Components
  lsel,s,line,,linha
  nsll,s,1
  *get,minode,node,0,num,min
  *get,maxnode,node,0,num,max
  *get,numnode,node,0,count
  _linha_='%linha%'
  *CFWRITE,_linha_,numnode
     noden=minode
    *DO,reps,1,numnode,1
        *IF,noden,LE,maxnode,THEN
        _noden_='%noden%'
            *cfwrite,_noden_
            noden=ndnext(noden)
        *ENDIF
  *ENDDO
*ENDDO
*CFCLOSE
```

A macro alldof.mac:

Macro que gera o arquivo texto ALLDOF.txt

```
*CFOPEN,ALLDOF,txt
csys,0
asel,all
ALLSEL,all,all
*get,numnode,node,0,count
*DO,reps,1,numnode,1
  _reps_='%reps%'
  *cfwrite,_reps_,ux(reps),uy(reps),uz(reps)
*ENDDO
*CFCLOSE
```