VRML+: uma plataforma de desenvolvimento rápido de mundos VRML

António Ramires Fernandes
Dep. Informática
Universidade do Minho
af@di.uminho.pt

Hugo Castelo Pires
Dep. Informática
Universidade do Minho
hap@sim.di.uminho.pt

Sumário

O VRML é uma linguagem "simples" para a criação de modelos 3D [VrmlSpec, Carey97, Ames97, Fernandes97]. Contudo, esta simplicidade força à escrita de muitas linhas de código quando, por vezes, o que se pretende fazer é algo, algoritmicamente, tão simples como um ciclo. Neste contexto apresentamos uma extensão ao VRML, denominada VRML+, que procura eliminar estes problemas e introduzir novas potencialidades como a ligação a bases de dados.

Não se pretende criar uma nova linguagem de raiz e muito menos uma nova norma. O objectivo é apenas definir novas instruções e disponibilizar um pré-processador que converterá o código fonte VRML+ em código VRML. Evita-se assim o moroso processo de criação de novas ferramentas de desenvolvimento e visualização, inerentes ao processo de elaboração de uma nova linguagem [Elliot99].

Palavras-chave

VRML, pré-processador.

1. INTRODUÇÃO

Este artigo apresenta o VRML+, uma proposta para extensão à linguagem VRML. Para além do publicado em [Fernandes99], a "versão" que aqui se apresenta inclui a ligação a bases de dados relacionais e uma arquitectura para a geração dinâmica de mundos VRML via Internet.

O VRML+ é um pré-processador de VRML cuja sintaxe definida permite adicionar ao VRML conceitos básicos existentes numa qualquer linguagem de programação. Não se pretende com o VRML+ criar uma nova norma para modelação de mundos interactivos 3D, mas sim uma plataforma de desenvolvimento rápido desses mundos.

Se pensarmos no funcionamento de um pré-processador da linguagem C, o VRML+ é em tudo semelhante. Tal como um compilador, o *parser* é executado para transformar um ficheiro fonte VRML+ num ficheiro VRML. Neste artigo apresenta-se ainda uma arquitectura que possibilita a geração dinâmica de um ficheiro VRML, isto é, a pedido do utilizador podem ser gerados como resultado da execução do *parser* diferentes mundos VRML.

Sendo uma linguagem de modelação, o VRML não possui à partida a capacidade de operar com bases de dados. No entanto, muitas são as vezes em que o objectivo é a visualização tridimensional e interactiva de dados contidos numa BD. Desta forma, será de todo o interesse estender os comandos existentes em VRML+ e incluir a ligação e consultas a BDs. Neste artigo, recorremos ao JDBC [SunJDBC] – *Java DataBase Connectivity* – como protocolo de comunicação com motores de BD, para propor essa extensão.

2. VRML+

As características incluídas no VRML+ permitem a escrita de código de uma forma mais estruturada, compacta e legível [Fernandes99]. Como já foi referido, estas características advêm de conceitos típicos existentes em linguagens de programação imperativas. Incluem-se variáveis, ciclos e instruções condicionais.

Para uma leitura mais detalhada do que será apresentado nesta secção consulte [Fernandes99].

2.1 Variáveis

A noção de variável pode ser muito útil no contexto da construção de modelos VRML. A necessidade de, por vezes, se partilhar propriedades comuns em diferentes modelos ou reutilizar propriedades já definidas pode ser suprimida pela utilização deste conceito. No entanto, o conceito de variável não existe em VRML.

Apesar de existir no VRML a construção DEF-USE, as diferenças entre as duas abordagens tornam-as bastantes díspares:

- Ao contrário das variáveis, o DEF-USE não separa a definição de nodos da sua utilização;

- Em VRML+ a utilização de uma variável implica sempre a cópia do valor nela contido.
- A construção DEF-USE apenas permite a partilha de nodos no seu todo. Pelo contrário, as variáveis permitem não só a partilha de nodos completos, mas também a partilha de campos de nodos ou mesmo de partes desses campos.

A implementação das caracteristicas dadas pelas variáveis utilizando apenas mecanismos VRML reduz a legibilidade do código e aumenta a complexidade da solução. Utilizando variáveis é possível ter um código mais legível e limpo, mantendo também a facilidade de edição das propriedades dos modelos.

2.1.1 Sintaxe e Exemplos

As variáveis em VRML+ podem ter qualquer tipo definido em VRML. A sua definição deve ser feita no início do ficheiro e obedecer à seguinte sintaxe:

```
#tipoVRML nomeVariavel = valor(es)
```

Os seguintes exemplos ilustram a definição de variáveis.

```
#SFFloat numCasas = 5
#SFFloat largCasas = 10
#SFFloat altura = 3
#SFVec3f posicaoInicial = 0 altura 0
#SFVec3f posicaoFinal = (largCasas *
(numCasas - 1)) altura 0
```

Podemos observar que para além da utilização de valores, é possível definir uma variável à custa de outra previamente definida ou mesmo à custa do cálculo de uma expressão. Neste último caso é necessário garantir que a expressão esteja entre parêntesis.

A utilização destas variáveis no restante código VRML deve garantir que o tipo da variável seja igual ao tipo requerido na sua utilização e deve ser precedida pelo caracter `@`, para evitar colisão de nomes com palavras reservadas do VRML, nomes de protótipos e nodos, e para facilitar o *parsing* do código.

Por exemplo, uma utilização possível da variável `posicaoInicial` definida no exemplo de cima poderia ser:

```
Transform {
  translation @posicaoInicial
  children [ ... ]
}
```

No exemplo que se segue define-se a variável `a` com o valor de uma esfera, e denomina-se de `inst_a` o nodo cujo valor é igual ao de `a`. `@a` permite-nos criar cópias do nodo original e `USE inst_a` permite-nos criar instância de um nodo igual ao original definido por `a`.

Eis o código VRML+:

```
#SFNode a = Shape{geometry Sphere{}}
Transform {
  Children [
    DEF inst_a @a
    @a
    USE inst_a
  ]
}
```

Ao qual corresponde o código VRML:

```
Transform {
  children [
    DEF a Shape {geometry Sphere{}}
    Shape {geometry Sphere{}}
    USE a
  ]
}
```

2.2 Ciclos

As instruções cíclicas são fundamentais em qualquer linguagem de programação. Em VRML não existe qualquer mecanismo semelhante a instruções repetitivas, apesar da grande utilidade que estas teriam. Se, por exemplo, se quisesse colocar postes de iluminação ao longo de uma rua ou um conjunto de cadeiras em torno de uma mesa era necessário escrever vários nodos `Transform`, um para cada cadeira ou poste de iluminação. Para suprimir esta falta, o VRML+ inclui na sua sintaxe uma instrução cíclica que permite simplificar tarefas com estruturas tipicamente repetitivas.

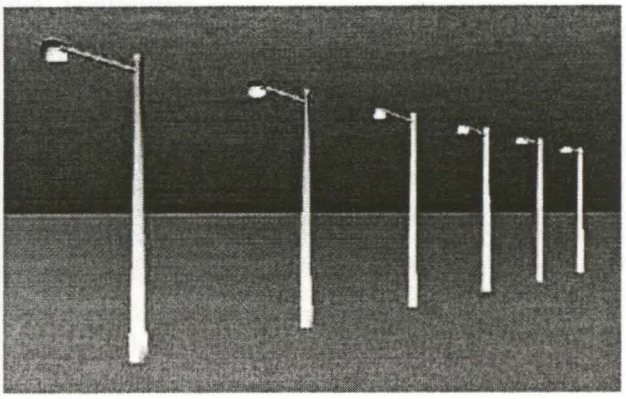
A sintaxe é a seguinte:

```
#REPEAT tipoVariavel var = arg1 TO
arg2 [TIMES arg3]
    vrml+ ou vrml utilizando var com a
    sintaxe definida no vrml+
#END
```

onde `tipoVariavel` define o tipo de dados VRML associado a `var`, `var` é uma variável de ciclo, e os valores `arg1` e `arg2` são os valores limites do intervalo

no qual var vai assumir um diferente valor a cada iteração. Estes podem ser também valores contidos em variáveis desde que o seu tipo seja igual ao definido em tipoVariavel. arg3 deve ser um valor ou variável pré-definida do tipo SFInt32 que define quantas vezes o ciclo vai ser iterado, ou seja, o intervalo entre arg1 e arg2 é divido em arg3 intervalos e var assume cada um dos limites desses intervalos. Se este argumento for omitido, e se o tipo da variável for SFInt32, assume-se um espaçamento unitário entre os limites.

Veja-se o seguinte exemplo no qual se colocam postes de iluminação em intervalos de 10 metros.



```
...
#SFVec3f inicioDaRua  0 0 100
#SFVec3f fimDaRua   0 0 150
...
#REPEAT SFVec3f trans = inicioDaRua TO
fimDaRua TIMES 6
  Transform {
    translation @trans
    children [
      código vrml para os postes
    ]
  }
#END
```

O código VRML equivalente seria:

```
Transform {
  translation 0 0 100
  children [
    código vrml para os postes
  ]
}
Transform {
  translation 0 0 110
  children [
    código vrml para os postes
  ]
}
... mais três transforms aqui
Transform {
  translation 0 0 150
  children [
    código vrml para os postes
  ]
}
```

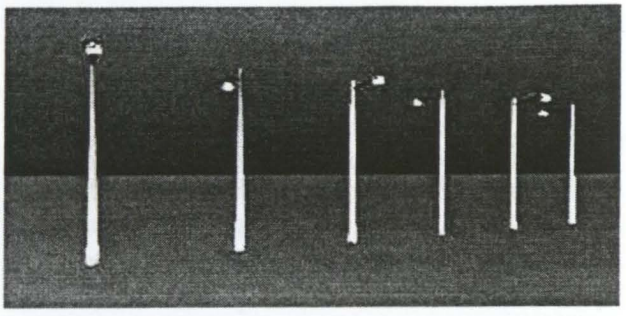
O ciclo REPEAT tem uma sintaxe alternativa que permite ao programador especificar directamente os valores a utilizar. A sintaxe é a seguinte:

```
#REPEAT tipoVariavel var IN [arg1, ...
argn]
    vrml+ ou vrml utilizando var com a
    sintaxe definida no vrml+
#END
```

Neste caso, var assume os n valores de arg1 a argn, iterativamente.

A instrução REPEAT permite ainda a combinação das duas sintaxes num só ciclo, podendo desta forma coexistir mais do que uma variável de ciclo. Consideremos novamente o exemplo da colocação de postes de iluminação ao longo de uma rua. Suponhamos agora que se pretende também alterar a orientação dos postes pares (o 2º, 4º e 6º) realizando uma rotação de 180º. Para tal escrever-se-ia:

```
...
#SFVec3f inicioDaRua  0 0 100
#SFVec3f fimDaRua     0 0 150
...
#REPEAT SFVec3f trans = inicioDaRua TO
fimDaRua TIMES 6 \
        SFFloat rot IN [0, 3.14]
  Transform {
    translation @trans
    rotation 0 1 0 @rot
    children [
      código vrml para os postes
    ]
}
#END
```



O caracter \ indica ao pré-processador que a instrução continua na linha seguinte. O número de iterações do ciclo, n, é definida pela primeira variável de ciclo (neste caso 6 vezes). As restantes variáveis, no caso do número de valores possíveis ser inferior a n, assumirão todos os valores ciclicamente. Assim, neste ultimo exemplo `rot` assumiria 6 valores, sendo o 1º valor `0`, o 2º `3.14`, o 3º novamente `0`, o 4º `3.14` e assim sucessivamente.

É ainda possível aninhar ciclos para aumentar o poder expressivo da linguagem.

2.3 Instruções Condicionais

O uso de instruções condicionais permite ao programador criar modelos dependentes do resultado de uma expressão. A sintaxe é a seguinte:

```
#IF expressão
   código vrml
[#ELSE
   código vrml]
#END
```

onde `expressão` é qualquer expressão do tipo SFBool e o componente `ELSE` é opcional. No código VRML final apenas um dos blocos de código aparecerá dependendo do valor da expressão calculada.

Como é obvio, é possível conjugar as instruções condicionais com as instruções cíclicas definidas na secção anterior.

O seguinte exemplo ilustra a utilização da instrução condicional para efeitos de depuração. Mais detalhes em [Fernandes99]:

```
#SFBool debug=TRUE
...
Material {
#IF debug
   transparency 0.5
#ELSE
   transparency 1
#END
}
...
```

O código VRML correspondente é,

```
...
Material {
   transparency 0.5
}
...
```

3. LIGAÇÃO A BASES DE DADOS

A ligação a bases de dados é uma característica importante num ambiente ou linguagem de programação/especificação. Para permitir a ligação com um grande número de bases de dados e porque o protocolo é amplamente utilizado, escolheu-se o JDBC como protocolo de comunicação com as BDs.

Assim, utilizar informação guardada numa base de dados é um processo com três fases:

1. identificar e estabelecer a ligação com a base de dados;
2. efectuar consultas e utilizar os resultados;
3. terminar a ligação.

Note-se que a ligação à BD só é efectuada durante o pré-processamento e, por conseguinte, durante a visualização do mundo VRML não existe nenhuma ligação aberta.

Todo o processo de consulta à BD está escondido do utilizador que irá visualizar o mundo VRML. Estes utilizadores não têm qualquer acesso à BD. Sendo assim, este sistema não implica nenhuma quebra de segurança da BD.

3.1 Estabelecimento da Ligação

Mantendo a semelhança com a sintaxe definida, a ligação com uma base de dados é realiza com o comando

```
#CONNECT TO 'url' AS user : passwd
```

no qual `url` representa, à semelhança do JDBC, o `url` da base de dados cuja forma é `protocolo:nome`[1], `user` o utilizador de acesso à BD e `passwd` a respectiva senha.

Por exemplo, o estabelecimento da ligação com a base de dados Fornecedores, via ODBC, utilizando o utilizador hap com a senha xpto32 seria realizado à custa do comando

[1] Não é necessário incluir no protocolo o JDBC. Este será incluído automaticamente pelo *parser* de VRML+.

```
#CONNECT TO 'odbc:Fornecedores' AS
hap:xpto32
```

Se, no entanto, a ligação à BD não exigir a utilização de utilizador e senha, uma versão simplificada deste comando, obtida omitindo a parte final, pode ser utilizada. Utilizaríamos então:

```
#CONNECT TO 'url'
```

3.2 Consultas

Após o estabelecimento da ligação o sistema está pronto a responder a consultas até que se feche da ligação. Estas consultas são realizadas utilizando SQL.

Para implementar o comportamento de uma consulta SQL criou-se um novo ciclo: o ciclo `QUERY`. Regra geral, as consultas devolvem a sequência dos vários registos que satisfazem o pedido. No ciclo `QUERY`, cada um destes registos é instanciado a cada iteração do ciclo.

Por exemplo, se existir uma tabela de alunos de uma turma numa BD, uma consulta pedindo os alunos cuja idade seja superior a 18 anos resulta na sequência de alunos que obedecem à condição "maiores de 18 anos". Em cada iteração do ciclo QUERY é instanciado um e um só desses alunos.

Esta instanciação é conseguida à custa de variáveis de ciclo. Estas servem para representar cada um dos campos do registo instanciado. No exemplo apresentado poderíamos ter uma variavel de ciclo nome, na qual era instanciado o nome do aluno da iteração corrente, uma variável de ciclo número para representar o número do aluno instanciado, e assim sucessivamente. Como cada variável de ciclo corresponde a um campo do registo devolvido pela consulta deve garantir-se que o número de campos devolvidos pela consulta é igual ao número de variáveis de ciclo. O conjunto das variáveis de ciclo pode ser visto como um registo de vários campos.

A cada iteração as variáveis de ciclo definidas assumem um novo valor obtido a partir do registo seguinte devolvido pela consulta. Se o resultado dessa consulta for apenas um registo, o ciclo terá apenas uma iteração, pelo que as variáveis de ciclo assumem um único valor.

A sintaxe é:

```
#QUERY 'sql' TO var1, var2, ..., varn
    vrml+ ou vrml utilizando varx com a
    sintaxe definida no vrml+
#END
```

Voltando ao exemplo dos alunos, se da tabela de alunos se pretende-se obter o número, nome e idade dos alunos maiores de 18 anos, escrever-se-ia:

```
#QUERY 'SELECT numero, nome, idade
FROM tabAlunos WHERE tabAlunos.idade >
18' TO vnumero, vnome, vidade
    ...
#END
```

Desta forma, nas variáveis `vnumero`, `vnome` e `vidade` estariam, respectivamente, a cada iteração do ciclo, o número, nome e idade de um dos alunos da tabela `tabAlunos` que obedecem a condição de serem maiores de 18 anos.

Dentro do `sql` é ainda possível a utilização de variáveis definidas quer no ficheiro quer na linha de comando aquando da execução do *parser*. Isto permite parametrizar a consulta e obter mundos diferentes a cada invocação do *parser* sem necessidade de alteração do código. É sobre esta geração dinâmica de mundos que falaremos de seguida na secção 4.

3.3 Terminar a Ligação

Após a execução de todas as consultas é possibilitado ao programador a terminação explícita da ligação com a BD de forma a libertar recursos durante o pré-processamento.

No entanto, e por razões de segurança, todas as ligações serão terminadas automaticamente após o pré-processamento.

A sintaxe para o termino da ligação é:

```
# DISCONNECT FROM 'url' AS user
```

na qual se identificam o utilizador (`user`) utilizado no estabelecimento da ligação `url`.

No exemplo apresentado na secção 3.1 terminaríamos a ligação efectuada com o utilizador `hap` à BD `Fornecedores`, protocolo `odbc`, utilizando o comando

```
#DISCONNECT FROM 'odbc:Fornecedores'
AS hap
```

Se no estabelecimento de uma ligação não se definir o utilizador dever-se-á terminar a ligação utilizando a seguinte sintaxe simplificada:

```
#DISCONNECT FROM 'url'
```

4. GERAÇÃO DINÂMICA DE MUNDOS NA WEB

A possibilidade de criar, a pedido, um mundo diferente é um aspecto da maior importância. A flexibilidade dada por esta abordagem pode ser garantida com a arquitectura que se apresenta a seguir. Esta é baseada na utilização do *parser* via Internet.

Para que o dinamismo seja garantido é necessário impor a seguinte condição: o *parser* VRML+ deve poder receber parâmetros aquando da sua execução, parâmetros esses que reflectem variáveis e os seus respectivos valores.

A utilização dessas variáveis no ficheiro VRML+ seria em tudo idêntica a utilização de uma qualquer variável definida dentro desse ficheiro.

Assim, a invocação do *parser* seria algo do tipo:

```
parser [<var1> <valor1> ... <varn>
<valorn>] <ficheiro entrada VRML+>
```

na qual `varx` é o identificador de uma variável e `valorx` o seu valor.

Esta imagem mostra a arquitectura que se propõe. No contexto da Internet, o recurso a uma CGI permitiria invocar, a partir de um *browser*, o *parser* VRML+ criando variáveis no seu ambiente e atribuindo-lhes valores. Utilizando essas variáveis, o *parser* processaria o ficheiro VRML+ de entrada e, acedendo a BDs se necessário, geraria um ficheiro VRML que seria devolvido à CGI e, por conseguinte, ao *browser*.

A utilização destas variáveis em instruções condicionais e em ciclos permite, a cada execução, a geração de um ficheiro VRML diferente a ser devolvido ao browser para visualização.

5. COMPARAÇÃO COM OUTROS TRABALHOS

O X-VRML [Walczak99] é o único trabalho semelhante ao VRML+. Em termos funcionais ambos os trabalhos têm muitos aspectos em comum. A principal diferença reside na sintaxe.

A sintaxe do X-VRML é mais complexa e extensa, sendo baseada no XML. A implementação de variáveis, ciclos e instruções condicionais é realizada a custa de *tags* XML.

O VRML+ utiliza uma sintaxe baseada em linguagens de programação imperativas.

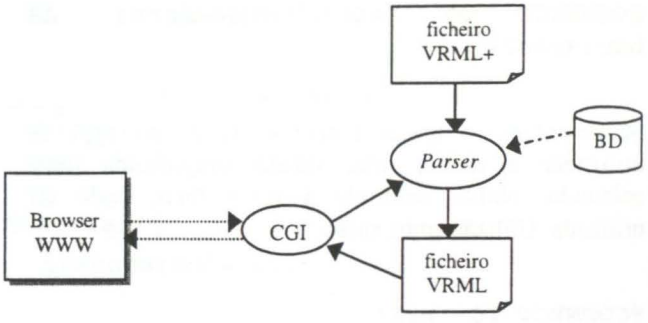
No nosso ponto de vista, o tipo de instruções disponibilizadas adequam-se mais à sintaxe do VRML+ do que a definida no X-VRML.

É de relevo notar que estes dois trabalhos foram desenvolvidos independentemente e sem conhecimento prévio um do outro, tendo sido apresentados pela primeira vez em VSMM99.

6. EXEMPLO

Nesta secção apresenta-se um exemplo no qual se aplicam todos os conceitos apresentados:

Consideremos uma página HTML que apresenta um gráfico VRML de cotações na bolsa de acções.



Esse gráfico, relativo às acções de uma empresa durante uma semana, deverá assinalar com a cor vermelha todas as cotações dessa semana superiores ao valor predefinido *aAssinalar*, e é construído pela execução de uma CGI que integra uma arquitectura como a apresentada na secção 4 para geração dinâmica de mundos.

A pedido do utilizador, esta CGI deve pode ser executada por várias vezes para apresentar graficamente cotações de diferentes acções. Assim, tem de ser passada à CGI o nome da empresa da qual se pretende obter o comportamento na bolsa. A invocação da CGI seria:

```
http://<servidor>/cgi.bat?empresa=<nom
e_empresa>
```

Supondo que a CGI efectua o *parsing* dos parâmetros e coloca na variável `nome` o nome da empresa, esta deve chamar o *parser* executando o seguinte comando:

```
parser nomeempresa nome grafico.wrl+
```

Este comando cria a variável `nomeempresa` no ambiente do processador com o valor `nome`, e processa o ficheiro VRML+ `grafico.wrl+`. O ficheiro VRML de saída é produzido para o *stdout* e enviado ao *browser* para visualização.

Se os dados das cotações na bolsa de acções existem numa base da dados *bdaccoes*, cujo acesso via ODBC dispensa utilizador e senha de acesso, o código VRML+, do ficheiro `grafico`, para resolver o problema será:

```
...
#SFInt32 x=0
#SFInt32 aAssinalar=10

#CONNECT TO 'odbc:bdaccoes'
#QUERY 'SELECT valor FROM accoes,
empresas WHERE
accoes.codEmpresa=empresas.codEmpresa
AND empresa=nomeempresa' TO valor
  #SFInt32 posY=valor/2
```

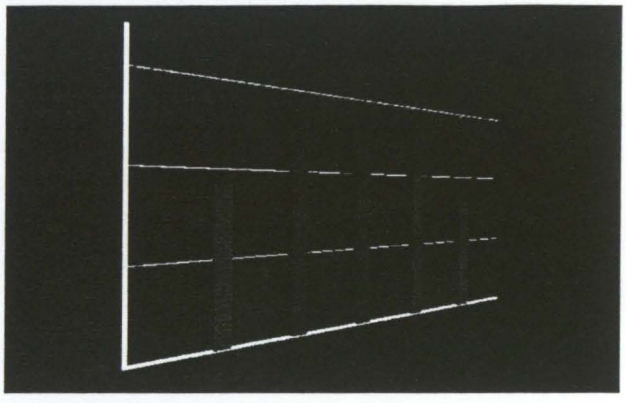
```
  Transform {
    translation @x @posY 0
    children [
      Shape {
        appearance Appearance {
          material Material {

            #IF valor>aAssinalar
              diffuseColor 1 0 0
            #ELSE
              diffuseColor .6 .2 .2
            #END
          }
        }
        geometry Box {
          size 1 @valor 1
        }
      }
    ]
  }
  #SFInt x=x+5
#END
```

Supondo as cotações 9, 11, 12, 10 e 8 das acções da empresa a visualizar, o código gerado pelo pré-processador de VRML+ seria:

```
...
Transform {
  translation 0 4.5 0
  children [
    Shape {
      appearance Appearance {
        material Material {
          diffuseColor .6 .2 .2
        }
      }
      geometry Box { size 1 9 1 }
    }
  ]
}
Transform {
  translation 5 5.5 0
  children [
    Shape {
      appearance Appearance {
        material Material {
          diffuseColor 1 0 0
        }
      }
      geometry Box { size 1 11 1 }
    }
  ]
}
Transform {
  translation 10 6 0
  children [
    Shape {
      appearance Appearance {
        material Material {
          diffuseColor 1 0 0
        }
      }
      geometry Box { size 1 12 1 }
    }
  ]
}
Transform {
  translation 15 5 0
  children [
    Shape {
      appearance Appearance {
        material Material {
          diffuseColor .6 .2 .2
        }
      }
      geometry Box { size 1 10 1 }
    }
  ]
}
Transform {
  translation 20 4 0
  children [
    Shape {
      appearance Appearance {
        material Material {
          diffuseColor .6 .2 .2
        }
      }
      geometry Box { size 1 8 1 }
    }
  ]
}
```

Às quais correspondem a imagem...



7. CONCLUSÃO

Este artigo apresenta o VRML+, uma extensão ao VRML que permite, escrevendo código de uma forma mais estruturada, compacta e legível, diminuir o tempo necessário à criação de mundos VRML.

A abordagem seguida foi a de importar noções comuns em linguagens de programação imperativas, tais como variáveis, ciclos e instruções condicionais. Inclui-se ainda a ligação a BDs e a possibilidade de geração dinâmica de mundos por forma a aumentar o poder expressivo do VRML.

Sendo um pré-processador, o VRML+ não obriga à construção de novas ferramentas de desenvolvimento e visualização. A conversão para ficheiros VRML garante a utilidade e versatilidade do VRML+.

8. REFERÊNCIAS

[Fernandes99] Fernandes, António Ramires, Pires, Hugo Castelo. VRML+. *Proceedings da Conferência VSMM99*, Setembro 1999, Dundee, Escócia.

[Elliot99] Elliot, Conal. Modeling Interactive 3D and Multimedia Animation with an Embedded Language. Microsoft Research, Outubro 1997. http://www.research.microsoft.com/~conal/papers/dsl97/dsl97.html.

[Walczak99] Walczak, Krzysztof, Cellary, Wojciech. XML-based Dynamic Generation of Virtual Scenes. *Proceedings da Conferência VSMM99*, Setembro 1999, Dundee, Escócia.

[VrmlSpec] The Official VRML Specification http://www.vrml.org/technicalinfo/specifications/vrml97/index.htm

[Carey97] Carey, Bell. The Annotated VRML 2.0 Reference Manual. Addison Wesley, 1997.

[Ames97] Ames, Nadeau, Moreland. VRML.SourceBook. John Wiley & Sons, 1997

[Fernandes97] Fernandes, António Ramires. The Online Interactive VRML Tutorial. http://sim.di.uminho.pt/vrml.

[SunJDBC] Sun Microsystems. JDBC Data Access API. http://java.sun.com/products/jdbc/