Modelação e registo da distribuição do Coeficiente de Reflexão Acústica num espaço utilizando o *Visualization Toolkit* (VTK)

Nuno Guimarães, Paulo Dias

***Resumo* – para a modelação e registo da distribuição do coeficiente de refracção acústica num espaço representado num ficheiro VRML, utilizando o Visualization Toolkit (VTK) e a linguagem C++. É discutida a metodologia utilizada e os resultados obtidos. Adicionalmente são apresentadas sugestões para futuro melhoramento da aplicação desenvolvida**

I. INTRODUÇÃO

No contexto das aulas de Laboratórios Integrados de Computação Gráfica, incluídas no Mestrado em Engenharia Electrónica e de Telecomunicações, foi dado o desafio de conceber uma aplicação em VTK que permitisse a modelação e registo da distribuição do Coeficiente de Reflexão acústica num espaço. Foi pedido que tal aplicação fosse capaz de obter os dados relativos a um modelo em VRML de uma sala, que atribuísse coeficientes de reflexão aos vários actores presentes na cena VRML lida, e que, finalmente, após a transformação desse modelo lido numa grelha estruturada de pontos, fossem registados num ficheiro os dados relativos à distribuição do coeficiente de reflexão nessa mesma grelha. Assim, obter-se-ia um modelo acústico da sala que poderia ser utilizado posteriormente noutras aplicações. Neste artigo é descrito o trabalho efectuado, bem como o método que utilizado para o fazer..

II. O VTK (VISUALIZATION TOOLKIT)

O VTK (Visualization Toolkit) é um sistema de software open-source, orientado a objectos e suportado em várias plataformas, para computação gráfica, visualização e processamento de imagem. Implementado em C++ o VTK também pode ser utilizado com as linguagens TCL, Python e Java permitindo assim o desenvolvimento de aplicações complexas, criação rápida de protótipos de aplicações e scripting. Embora não forneça componentes de interface de utilizador, pode ser integrado com bibliotecas como Tk ou X/Motif.

O VTK fornece uma variedade de representações de informação incluindo conjuntos desorganizados de pontos, informação poligonal, imagens, volumes e grelhas estruturadas, rectilíneas e não estruturadas. O

VTK tem também readers/importers e writers/exporters que permitem a troca de informação com outras aplicações. O VTK fornece vários filtros que permitem operar sobre os tipos de informação acima referidos [1].

Devido a todas estas características, o VTK foi escolhido como ferramenta a utilizar para atingir os objectivos do trabalho.

III A LINGUAGEM VRML (VIRTUAL REALITY MODELLING LANGUAGE)

A linguagem VRML permite a especificação dinâmica de cenas 3D nas quais um utilizador pode navegar utilizando um browser VRML. As cenas VRML podem ser distribuídas através da Internet e podem ser visualizadas em browsers utilizando plugins adequados. A linguagem VRML integra-se bem com a World Wide Web pois as cenas VRML podem ser ligadas a outras cenas VRML, utilizando URLs [2]..

IV. DESCRIÇÃO DO TRABALHO EFECTUADO

A Importação do ficheiro VRML que contém a cena

A primeira coisa a fazer para se conseguir uma representação da cena VRML utilizando VTK era de algum modo conseguir importar esse modelo. Ou seja, ter encapsulada num objecto VTK a informação relativa a actores, iluminação, câmaras, etc., do modelo. Para tal foi utilizado um dos vários importers disponibilizados no VTK, específico para o VRML, o vtkVRMLImporter. Esta classe tem métodos que permitem a leitura de um modelo VRML 2.0 e cria para esse modelo representações VTK do renderer associado, luzes, câmaras e actores [3]. O código utilizado foi:

```
vtkVRMLImporter
*importer=vtkVRMLImporter::New();
importer->SetFileName(vrmlFile);
importer->Read();
vtkRenderer *renderer;
renderer=
importer->GetRenderer();
```

A representação do modelo VRML original importado pode ser vista na Figura 1

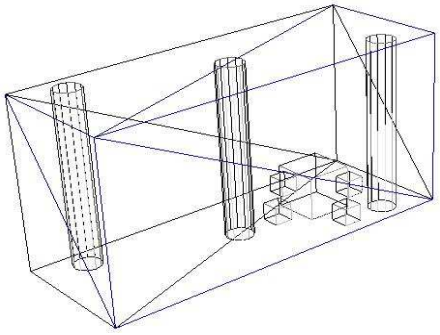

Figura 1 – Representação do modelo VRML importado

B. Cálculo das dimensões da grelha estruturada de pontos

Para se poder ter o modelo lido representado numa grelha estruturada de pontos de forma correcta era necessário obter não só as dimensões como a localização do modelo no espaço 3D. Com o código seguinte:

```
// variável que armazena os limites do modelo
// VRML

float  wholeBounds[6];

// cálculo desses limites

renderer->ComputeVisiblePropBounds
(wholeBounds);

// coordenadas calculadas acima

float globalXmin=wholeBounds[0];
float globalXmax=wholeBounds[1];
float globalYmin=wholeBounds[2];
float globalYmax=wholeBounds[3];
float globalZmin=wholeBounds[4];
float globalZmax=wholeBounds[5];
```

foi possível obter as coordenadas dos pontos que limitavam o modelo VRML e armazenar essas coordenadas numa variável para posterior utilização.

De seguida, foi efectuado um cálculo da translação necessária para trazer as coordenadas de valor mais pequeno em x, y e z do modelo para a origem, de modo a facilitar cálculos posteriores e um correcto acerto com as coordenadas da grelha estruturada de pontos:

```
// Exemplo para x
float xtransFactor=0;
// Cálculo do factor consoante
// coordenadas do modelo VRML
if(globalXmin<0)
 {
   xtransFactor=0-globalXmin;
   globalXmin=globalXmin+xtransFactor;
   globalXmax=globalXmax+xtransFactor;
 }
```

As dimensões da grelha estruturada de pontos teriam então de ser calculadas segundo as dimensões do modelo VRML original e o espaçamento desejado para a grelha de pontos. Utilizando os cálculos feitos anteriormente, fez-se:

```
// Cálculo das dimensões da grelha
estruturada
// consoante as dimensões do modelo VRML
original

int newXdim=(globalXmax -
globalXmin)/gridCellSpacing + 2;
int newYdim=(globalYmax -
globalYmin)/gridCellSpacing + 2;
int newZdim=(globalZmax -
globalZmin)/gridCellSpacing + 2;
```

Obtendo-se assim as dimensões newXdim, newYdim, newZdim para a grelha contando com o espaçamento dado por gridCellSpacing.

C. Construção da grelha estruturada vtkImageData e atribuição de escalares

A informação representada pela classe *vtkImageData* é regular quer em geometria quer em topologia. O tipo de dados desta classe é estruturado o que significa que a localização dos pontos na grelha é definida implicitamente utilizando apenas como parâmetros uma origem, um espaçamento e dimensões. Conceptualmente, o conjunto de dados *vtkImageData* é composto por células que representam *voxels* (*vtkVoxel*) ou *pixels* (*vtkPixels*). Contudo, a sua natureza estruturada permite-nos o armazenamento dos valores dos dados num simples *array* em contraste com a necessidade da criação explícita de células. Existem vários tipos de filtros no VTK capazes de converter informação *vtkImageData* em *vtkPolyData*, como, por exemplo o *vtkContourFilter* que é capaz de extrair contornos na forma de superfícies triangulares a partir do conjunto de dados da imagem [1]. Por todas estas características foi escolhido o *vtkImageData* como o objecto a considerar para encapsular a grelha estruturada de pontos e a informação escalar (valor do coeficiente de reflexão) associada a esses pontos.

A construção da grelha estruturada foi então efectuada utilizando a classe *vtkImageData* e a informação de dimensão do modelo VRML original:

```
// Construção da grelha estruturada
vtkImageData

vtkImageData *imgData=vtkImageData::New();
imgData->SetSpacing(gridCellSpacing,
gridCellSpacing,gridCellSpacing);
imgData->SetDimensions(newXdim,
newYdim,newZdim);
```

Adicionalmente foi efectuada uma inicialização de todos valores dos escalares associados aos pontos da grelha para o valor 1.0, valor máximo do coeficiente de reflexão acústica:

```
for ( float x =globalXmin ;
x < globalXmax + gridCellSpacing ;
x = x + gridCellSpacing )
{
 vtkIdType id=imgData->FindPoint(x,y,z);
 if(id>=0)
 {
    imgData->GetPointData()->GetScalars()
    ->InsertTuple1(id,1.0);
 }
}
```

Figura 2 – Grelha estruturada obtida.

D. Criação de actores da cena e atribuição de coeficientes a esses actores

No intuito de permitir ao utilizador poder interagir com a representação do modelo "clicando" sobre os actores e obter assim informação sobre qual o coeficiente associado a cada um, foi implementada uma nova classe *vtkMyOpenGLActor*, que herda de *vtkOpenGLActor*, mas que tem como atributo adicional o coeficiente de reflexão acústico. Este novo tipo de actor destinou-se a fazer parte de um modelo poligonal que permite o *picking* dos actores e que tem associado um *Observer* que escreve na *prompt* os dados relativos ao actor escolhido, nomeadamente o coeficiente de reflexão respectivo.

À medida que são detectados os actores presentes no modelo VRML original, são construídos actores do novo tipo *vtkMyOpenGLActor* com um dado coeficiente associado e é garantido que estes actores se irão situar correctamente no novo modelo poligonal que é obtido:

```
// Aplicar transformação que garanta que
// novos actores criados
// vtkMyOpenGLActor estarão localizados
// correctamente na cena
vtkPolyData *poly=vtkPolyData::New();
vtkPolyDataMapper *newMapper =
vtkPolyDataMapper::New();
vtkTransform *aTransform = vtkTransform::New();
vtkTransformPolyDataFilter  *aFilter =
vtkTransformPolyDataFilter::New();
aTransform->Identity();
aTransform->SetMatrix(Act->GetMatrix());
aFilter->SetTransform(aTransform);
aFilter->SetInput((vtkPolyData *)
Act->GetMapper()->GetInput());
poly = aFilter->GetOutput();
newMapper->SetInput(poly);

( … )
// Criação de objecto vtkMyOpenGLActor
// que é a minha implementação de
// vtkOpenGLActor
// e que permite a associação do atributo
// coeficiente de reflexão a cada actor

vtkMyOpenGLActor
*newActor=vtkMyOpenGLActor::New();
newActor->SetRefractionCoefficient
(1.0/auxiliar);
newActor->SetActorID(auxiliar);
newActor->SetMapper(newMapper);
```

E. Preenchimento na grelha de pontos dos valores escalares respectivos aos coeficientes de cada actor

À medida que se vão obtendo os novos actores, é necessário preencher a grelha de pontos de forma coerente com o modelo original atribuindo os escalares correspondentes ao coeficiente de reflexão de modo correcto. Para tal, são obtidas as coordenadas do actor no modelo VRML original e ele sofre uma translação coerente com aquela que tinha sido descrita anteriormente no ponto b) aquando da atribuição de dimensões e localização à grelha estruturada. Após este acerto pode-se então utilizar mais uma vez o método *FindPoint* da classe *vtkImageData* e atribuir o valor do coeficiente aos vários pontos correspondentes à localização do actor corrente.

F. Aplicação de filtro para visualização da grelha estruturada de pontos

Na figura 2 vimos o resultado do *rendering* directo da grelha estruturada obtida. Nessa figura não é possível detectar se os valores dos coeficientes estão efectivamente atribuídos nem qual a sua distribuição sobre a grelha, pois trata-se de uma representação directa do objecto *vtkImageData* resultante do ponto f), e, como tal, é um *array* regular de *voxels* (células) preenchendo todo o volume. Sendo assim, foi necessário utilizar um filtro, que já foi referido no ponto c) e que é chamado de *vtkContourFilter*. Este filtro aplica um algoritmo do tipo *marching cubes* aos seus dados de entrada, ou seja, associa uma tabela *contour case* a cada tipo de célula de modo que cada célula gera primitivas de contorno adequadas. Por exemplo, uma célula do tipo tetraedro implementa o algoritmo *marching tetrahedron* e irá gerar primitivas triangulares. Ou seja, o filtro *vtkContourFilter* gera primitivas de contorno como pontos, linhas ou superfícies consoante a combinação dos tipos de células de entrada [3]. No nosso caso o filtro gera superfícies correspondentes a valores iguais do coeficiente de reflexão e permite-nos assim uma posterior visualização da distribuição do coeficiente no espaço do modelo representado na Figura 3.

```
// Definição do filter que detecta os
// contornos
// que permitirão a visualização "filtrada"
// da grelha estruturada

vtkContourFilter
*cFilter=vtkContourFilter::New();
cFilter->SetInput(imgData);
cFilter->SetValue(0, 1.0);
cFilter->Update();

( … )

vtkDataSetMapper *imgMapper =
vtkDataSetMapper::New();
imgMapper->SetInput(cFilter->GetOutput());
```

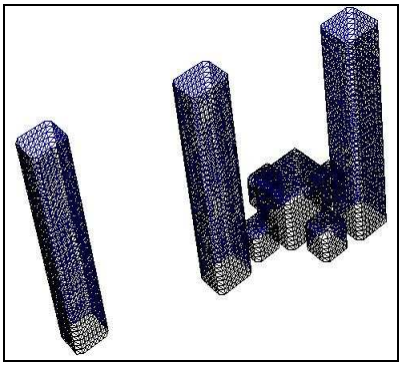

Figura 3 – Grelha de pontos filtrada

g) Escrita para ficheiro da grelha obtida

De modo a se poder posteriormente utilizar a atribuição de coeficientes de reflexão efectuada ao modelo, a informação relativa à grelha estruturada de pontos é escrita em dois ficheiros:

- Um ficheiro de texto de leitura humana fácil:

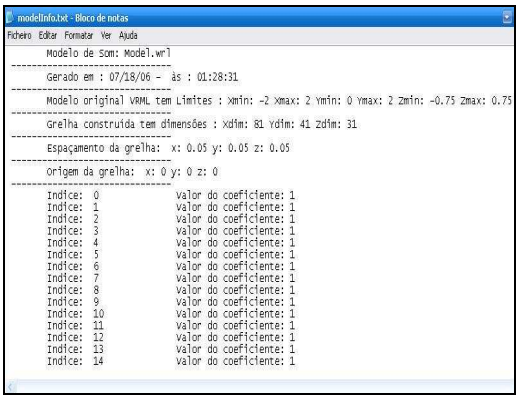

Figura 4 – Ficheiro de texto

- Um ficheiro VTI, que é um ficheiro XML mas com terminação .vti, que permite um posterior carregamento da grelha estruturada para outro objecto *vtkImageData* garantindo assim a portabilidade dos dados.

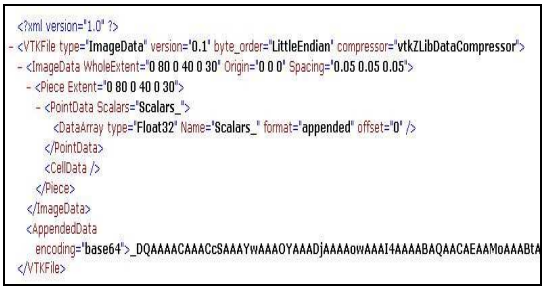

Figura 5 – Ficheiro XML (terminação .vti)

Para carregar a grelha noutra aplicação VTK qualquer basta fazer:

```
vtkImageData *testData=vtkImageData::New();
vtkXMLImageDataReader
*reader=vtkXMLImageDataReader::New();
reader->SetFileName(vtiFile);
reader->Update();
testData=reader->GetOutput();
```

V. CONCLUSÕES

A aplicação obtida, como já foi descrito, é capaz de carregar um modelo VRML 2.0, analisá-lo, atribuir coeficientes de reflexão aos diversos actores, construir uma grelha de pontos estruturada com esses coeficientes, e gravar em ficheiro a grelha obtida, quer num formato facilmente lido por humanos (texto) quer num formato

que garante a sua portabilidade (VTI). Adicionalmente, o utilizador pode visualizar o modelo quer num formato poligonal em que os actores são seleccionáveis e é indicado na *prompt* o coeficiente do actor seleccionado (Figuras 6 e 7), quer num formato poligonal resultante duma filtragem que obtém contornos consoante os coeficientes de reflexão (Figura 3)

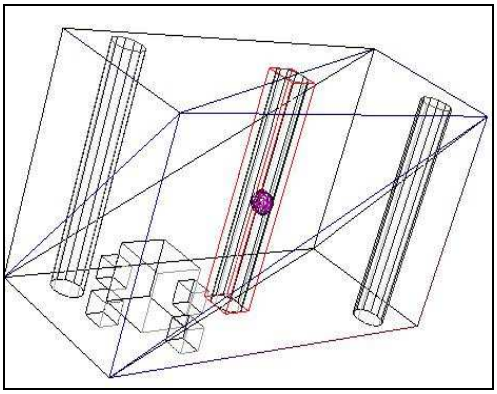

Figura 6 – Modelo poligonal com *picking*

```
Coords: x:0.802785 y:0.926432 z:0.000000
 ID do Actor: 2
 Coeficiente do Actor: 0.500000
 Coords: x:0.509804 y:0.058564 z:0.000000
 ID do Actor: 7
 Coeficiente do Actor: 0.142857
 Coords: x:0.468756 y:0.928743 z:0.000000
 ID do Actor: 8
 Coeficiente do Actor: 0.125000
 Coords: x:0.632797 y:0.733943 z:0.000000
 ID do Actor: 2
 Coeficiente do Actor: 0.500000
 Coords: x:0.483439 y:0.362034 z:0.000000
 ID do Actor: 7
 Coeficiente do Actor: 0.142857
```

Figura 7 – Indicaço na *prompt* de coeficientes e actores

Contudo, faltou implementar algumas funcionalidades que são importantes mas que podem estender o que já foi feito, nomeadamente:

- Associar um interface gráfico que permita ao utilizador indicar sobre os actores da cena qual o coeficiente de reflexão associado ou utilizar o formato que surgiu da evolução do VRML para XML, que é o X3D (a explorar, pois não chegou a ser investigado se o VTK conseguiria ler atributos adicionais colocados no XML, tal como o coeficiente de reflexão);

- Atribuição de cores diferentes consoante o coeficiente de reflexão aos pontos do modelo poligonal resultante da filtragem da grelha.

- Uma lacuna importante da aplicação desenvolvida é que só para modelos VRML cujas cenas são constituídas exclusivamente por volumes rectangulares. Ou seja, só nesse caso é que o modelo 3D preenchido obtido se assemelha de forma quase exacta ao modelo VRML original. Assim, um trabalho futuro será o de permitir que a aplicação efectue o preenchimento não só de volumes rectangulares, mas também de volumes com outras formas mais complexas.

REFERÊNCIAS

[1] Lisa S. Ávila et Al. , “The VTK User’s Guide”, Kitware, 2004

[2] Daniel K. Schneider et Al., “VRML Primer & Tutorial” , University of Geneva, 1998

[3] Will Schroeder et Al., ”The Visualization Toolkit 2nd. Edition”, Prentice Hall, 1998