FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

URBANISMO E ESPAÇOS VIRTUAIS

Divulgação e Discussão na Comunidade

José Carlos Guedes dos Prazeres Miranda

Licenciado em Engenharia Informática pela Escola Superior de Tecnologia e Gestão do Instituto Politécnico da Guarda

Dissertação submetida para a satisfação parcial dos requisitos do grau de Mestre em Tecnologia Multimédia

Dissertação realizada sob a supervisão do Professor Doutor António Augusto de Sousa, do Departamento de Engenharia Electrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto

Porto, Março de 1999

À Nita,

Resumo

Um processo de transformação urbano é algo que nunca pode ser considerado definitivo, mas antes permanece sempre em devir. A verdadeira força motriz do desenvolvimento de uma cidade reside nos cidadãos, ao mesmo tempo sujeitos e destinatários de um processo que cada vez mais deve assentar na dinâmica criativa e na capacidade de participação de todos e de cada um.

No actual contexto legislativo e no que se refere concretamente às Autarquias Locais, a única forma existente de captar a opinião do cidadão sobre alguma intervenção de natureza urbana é o Inquérito, um processo de auscultação reconhecidamente moroso e pouco objectivo.

Uma Autarquia possui uma vasta experiência no planeamento e gestão do projecto urbano que poderá, ser interessante explorar e enriquecer através das novas tecnologias de informação (Internet, Multimédia, Realidade Virtual, etc.), de forma a tornar o processo de transformação da cidade mais divulgado e participado.

Neste contexto, foi desenvolvido um protótipo que oferece a possibilidade ao cidadão de aceder, via Internet, a um modelo virtual 3D do centro histórico da cidade da Guarda. Este modelo pode ser manipulado, sob o ponto de vista urbanístico, mediante a alteração e remoção dos diferentes elementos que o compõem, assim como pode beneficiar com a integração de novos elementos. Os resultados dessa manipulação poderão ser enviados, como sugestão, para a Autarquia que fornece o serviço, de forma a poder ser avaliado "na hora" o seu impacto urbano.

A actual tecnologia aponta para o VRML como sendo um suporte adequado para descrever ambientes virtuais interactivos e a três dimensões na Internet e, complementarmente, para o Java como sendo a linguagem apropriada à adição de comportamentos complexos aos objectos de uma cena VRML.

Com a conjugação das capacidades que as novas tecnologias de informação permitem, da experiência acumulada pelas Autarquias no planeamento e gestão de projectos urbanos e da motivação dos cidadãos em participarem na construção da sua cidade, pretende-se, assim, contribuir para o desenvolvimento e a dinamização do uso das tecnologias para melhor comunicar o trabalho de recuperação urbano, tornando, mais rico, o confronto de ideias e a participação de todos os intervenientes no processo de transformação da cidade: políticos, técnicos, agentes culturais, agentes comerciais, cidadãos.

Abstract

An urban transformation process is something that can never be considered as definite, as it is always changing. The actual main motive of the development of a city resides on the citizens, who at the same time are the senders and receivers of a process that is based more and more on the dynamic creativity and on the participation capacity of all and of each individual one.

In reference to the actual Local Authorities and in the current legislative context, the only existent way to hear the citizen's opinion about an urban intervention is with a questionnaire, which is a long process and not very objective.

Since a Local Authority has vast experience in planning and managing urban projects it would then be interested in improving by exploring new information technologies (Internet, Multimedia, Virtual Reality, etc.), in order to have more participation and disclosure in the city transformation process.

A prototype was developed in this context which offers the citizen the possibility to connect to a 3D virtual model of a historic centre in the city of Guarda via Internet. In an urbanistic point of view, this model can be manipulated by modifying and removing the elements that compose it, as well as integrating new elements. The results of this manipulation can then be sent to the Local Authority as a suggestion so that it can evaluate the urban impact at that moment.

The current technology considers VRML to be the adequate tool to describe interactive virtual and three dimensional environments on the Internet and it considers Java to be the appropriate language to add complex behaviours to the objects of a VRML scene.

By joining the new information technologies, the accumulated experience of Local Authorities in planning and managing urban projects and the citizens' motivation in building their city, it intends to contribute to the development and dynamization of the use of these technologies to improve the communication about urban recuperation work, therefore making the discussion of ideas and the participation of all the agents, (politicians, technicians, cultural agents, commercial agents, citizens) in the city transformation process more valuable.

Résumé

Le processus de transformation de l'aménagement urbain ne peut jamais être considéré comme étant définitif mais plutôt comme étant plongé dans une dynamique génératrice incessante. Ce sont les citoyens qui constituent la véritable force motrice du développement de la ville. Ils sont à la fois les agents et les destinataire d'un processus qui doit, de plus en plus, reposer sur la dynamique créative et sur la capacité de participation de tout un chacun.

Dans le contexte législatif actuel, et en ce qui concerne plus particulièrement la compétence des mairies, le questionnaire est la seule forme d'auscultation de l'opinion du citoyen quant à une éventuelle interventionsur l'urbanisation. Mais il s'agit d'un moyen lent et peu objectif.

Les mairies possèdent une large expérience au niveau de l'aménagement et de la gestion du projet urbain. Ces dernières peuvent alors éprouver l'intérêt d'explorer et d'enrichir, pour mieux divulguer et inciter à la participation de tous dans ce processus de métamorphose, par le biais de l'utilisation des nouvelles technologies d'information (Internet, le multimédia, la réalité virtuelle etc..),.

Ainsi, ai-je développé un prototype qui permet au citoyen de pouvoir accéder, *via* Internet, à un modèle virtuel en 3D du centre historique de Guarda. Du point de vue urbanistique, ce modèle peut être manipulé grâce à l'altération et l'élimination des différents éléments composant le modèle. L'intégration de nouveaux éléments est également possible. Les résultats de cette manipulation pourront être envoyés, comme

des suggestions, à la mairie promouvant ce service pour que l'impact urbain puisse être évalué «sur le champ».

La technologie actuelle a élu le VRML comme étant le support adéquat pour décrire les environnements virtuels interactifs et à 3D. Le Java, lui, semble être le langage le plus correct pour l'addition de nouveaux comportements complexes aux objets d'une scène VRML.

Il s'agit donc avec l'effort en conjoint de l'offerte des nouvelles technologies de l'information avec l'expérience accumulée par les mairies au niveau de l'aménagement et de la gestion des projets urbains et la motivation des citoyens à l'égard de la construction de leur ville de contribuer au développement et à la dynamisation de l'utilisation des technolgies nouvelles pour mieux communiquer à propos du travail de la récupération urbaine. Ainsi, l'échange et l'analyse des idées seront-ils enrichis et la participation de tous les intéressés (les politiciens, les techniques, les agents culturels, les agents commerciaux, les citoyens en général...) plus.

Agradecimentos

Entre as muitas pessoas e entidades inteiramente merecedoras dos meus sinceros agradecimentos pelo auxílio, compreensão e consideração que me dispensaram ao longo de todo este curso de Mestrado, cuja última etapa foi a elaboração desta dissertação, e sem querer menosprezar o contributo das omitidas, gostaria de destacar as seguintes.

Ao meu orientador, Professor Doutor António Augusto de Sousa, pela confiança em mim depositada e a colaboração prestada durante todo o tempo em que decorreu o curso de Mestrado, bem como por todos os conselhos preciosos que muito contribuiram para a dissertação apresentada.

Ao Presidente do Instituto Politécnico da Guarda, Professor Doutor José Augusto Alves, pelas facilidades concedidas e condições de trabalho proporcionadas.

Ao senhor Vereador Eng. Joaquim Valente e ao senhor Arquitecto Joaquim da Costa Gomes, da Câmara Municipal da Guarda, pela disponibilidade e interesse demonstrados em relação ao protótipo desenvolvido no âmbito desta dissertação.

À Professora Maria Clara Silveira e ao Eng. Carlos Costa, colegas de Departamento, pela disponibilidade, ajuda e amizade manifestadas.

Ao meu colega Carlos Reinas Caldeira, pelo auxílio prestado nas correcções da dissertação ao nível da expressão.

À Dra. Maria Manuela Coelho e à Lissa, pelo auxílio concedido nas traduções dos resumos desta dissertação.

Aos meus colegas de Departamento, o incentivo e muitos conselhos proporcionados.

A todos os que partilham informação através da Internet, sem a contribuição dos quais esta dissertação não teria sido realizada.

E finalmente, um agradecimento muito especial à Nita por todo o carinho, paciência e apoio demonstrados, que em muito me ajudaram a superar esta dura prova.

A todos, muito obrigado.

Índice

Índice de Figuras

Índice de Tabelas

capítulo 0.

Introdução

"A cidade não é uma obra de arte – um artefacto – mas sim qualquer coisa que constantemente se está fazendo e desfazendo" [Goitia92].

A maneira de articular a dupla operação de construção-destruição vai submeter esta virtualidade que as cidades têm de se desenvolverem harmoniosamente. Porém, o ideal é que a construção se faça com o mínimo de destruição possível e, sobretudo, que esta não seja senão uma readaptação inteligente às novas exigências. Poder-se-á dizer que as cidades beneficiam de um processo de transformação urbano, ao longo dos tempos.

A participação do cidadão neste processo de transformação urbano tem uma extrema importância, já que ninguém, como ele, conhece as características da cidade e está tão interessado na criação e no desenvolvimento de boas condições para a cidade. Dessa forma, ficará naturalmente empenhado na sua protecção e salvaguarda. Os espaços públicos deverão, tanto quanto possível, suscitar opiniões consensuais relativamente à maioria dos cidadãos dessa área.

A intervenção dos cidadãos num processo de transformação urbano pode ser factor de melhoria funcional, identificando-os com o "seu" exterior e corresponsabilizando-os na sua manutenção [Coelho88]. O cidadão precisa, assim, de participar e cooperar nas várias intervenções urbanas que vão sendo efectuadas na sua cidade.

0.1 Definição do Problema

Baseado em [Gomes99], "No actual contexto legislativo e no que se refere especificamente às Autarquias Locais, não existem disponíveis muitas formas sistematizadas de estas conhecerem a opinião dos seus munícipes, de maneira a que estes possam ser intervenientes activos no processo de tomada de decisões". Com efeito, quando pretende, de algum modo, conhecer o pulsar da população em relação a alguma intervenção urbana, as Autarquias recorrem normalmente ao Inquérito, um formato de auscultação que enferma de alguns defeitos e insuficiências, como sejam a pouca objectividade e a demora de apuramento de resultados.

Uma Autarquia possui um património de experiências no planeamento e gestão de projectos urbanos que poderão abrir perspectivas e suscitar o recurso a novas formas de exploração e desenvolvimento. As novas tecnologias de informação (Internet, Multimédia, Realidade Virtual, ...), com toda a sua força e dinâmica, assumem-se como o instrumento privilegiado para conferir um elevado grau de interacção à transformação do meio urbano, beneficiando da maior abertura dos canais de comunicação e promovendo, dessa forma, processos mais divulgados e participados.

Pretende-se, então, explorar as capacidades das novas tecnologias ao serviço da comunicação para demonstrar a possibilidade de qualquer cidadão comunicar, de uma forma rápida e eficiente, eventuais sugestões ou propostas de intervenção que pretenda realizar no meio urbano. O processo de planeamento e projecto na cidade poder-se-á tornar mais divulgado e concorrido, possibilitando uma maior abertura ao confronto de ideias entre todos os intervenientes num processo de transformação urbano: políticos, técnicos, agentes culturais, agentes comerciais e cidadãos.

Assim, os objectivos da presente dissertação são os a seguir enunciados:

1. Criar um ambiente virtual de um espaço urbano específico.

   Trata-se, muito concretamente, da possibilidade de uma Autarquia disponibilizar, através da Internet, modelos tridimensionais do espaço exterior que pretende submeter à apreciação do cidadão. Estes modelos devem permitir uma navegação 3D interactiva pelos diferentes quarteirões e a integração de informação multimédia, de forma a facultar um contacto mais realista com os seus elementos. A actual

tecnologia aponta para o VRML como um suporte adequado para descrever ambientes virtuais interactivos e a três dimensões na Internet.

2. Permitir a manipulação do espaço exterior, mediante a alteração e remoção dos diferentes elementos que compõem esse espaço, bem como a integração de novos elementos. De acordo com [Coelho85], os elementos caracterizadores de um espaço exterior são a iluminação, as texturas superficiais, a cor e as tonalidades, a componente verde e os elementos de equipamento. Entenda-se por elementos de equipamento tudo aquilo que "mobila" parcialmente as zonas exteriores "vazias", como sejam, as fontes, os postes de iluminação, os bancos de jardim, etc.

   Conceder ao cidadão comum a possibilidade de manipular o espaço exterior sob o ponto de vista urbanístico, poderá levantar algumas questões no que diz respeito à possível descaracterização daquilo que é a essência do próprio espaço. A solução para este tipo de situações passará pelo desenvolvimento de mecanismos que possam restringir, de algum modo, o tipo de operações que o cidadão possa efectuar na sua intervenção. Sob o ponto de vista informático, permitir a manipulação do espaço exterior implica adicionar comportamentos à cena virtual, normalmente obtidos à custa de programação externa. A actual tecnologia aponta para o Java como sendo a linguagem adequada para adicionar comportamentos complexos aos objectos de uma cena VRML.

3. Permitir que os resultados da manipulação do espaço sejam enviados, como sugestão, via Internet, para a Autarquia que fornece este serviço, por forma a poder ser avaliado o seu impacto urbanístico.

   O recurso aos meios informáticos e a fruição das vantagens proporcionadas pelos mesmos poderão facultar aos responsáveis pelo processo de decisão o conhecimento abalizado e actualizado do sentir da opinião pública em relação à gestão do espaço urbano submetido à sua apreciação. Torna-se assim necessário estabelecer uma comunicação entre o computador-cliente deste sistema e o computador-servidor existente na Autarquia que fornece o serviço.

A presente dissertação pretende assim aferir em que medida as novas tecnologias de informação abrem perspectivas para o desenvolvimento de novas estratégias em planeamento e projecto urbano.

0.2 Organização da dissertação

Esta dissertação está estruturada em 5 capítulos principais, para além deste e das Referências. No final de cada capítulo é feita uma síntese com o intuito de facilitar a leitura.

Um dos mais recentes desenvolvimentos na Internet foi a adopção do VRML *(Virtual Reality Modeling Language)*, como método *standard* para descrever cenas virtuais interactivas a três dimensões. No capítulo 1 descreve-se de uma forma geral a evolução desta linguagem ao longo dos tempos e efectua-se um enquadramento de vários assuntos relacionados com a criação e visualização de ambientes tridimensionais na Internet. É o Estado da Arte.

Apesar de ser uma linguagem conceptualmente simples, a utilização do VRML de forma eficiente exige uma visão clara das suas principais características. No capítulo 2 é feita uma introdução global aos conceitos da linguagem, tais como a estrutura dos dados, a hierarquia da cena e o modelo de execução, o qual está relacionado com o modo como os objectos trocam informação entre si, por forma a alterarem as suas propriedades.

A manipulação de uma cena VRML exige a adição de comportamentos complexos aos objectos de uma cena VRML: Torna-se assim necessário o recurso a uma linguagem de programação externa, sendo a linguagem Java, devido à sua portabilidade, a principal candidata. No capítulo 3 é feita uma descrição muito breve da linguagem Java e são abordadas as formas de integração desta linguagem com o VRML.

Porque se trata de aplicações destinadas a serem visualizadas através da Internet, os mundos VRML que não sejam cuidadosamente optimizados são, certamente, muito lentos para captar o interesse do utilizador. O capítulo 4 descreve as etapas envolvidas no processo de criação de um mundo VRML, dando especial atenção à Optimização e define algumas directrizes para aumentar a eficiência de um mundo virtual.

Como aplicação prática do problema definido nesta dissertação foi criado um protótipo. No capítulo 5 descreve-se, em pormenor, os requisitos deste protótipo e são abordadas todas as questões relacionadas com a sua implementação, nomeadamente a arquitectura do sistema, o interface da aplicação e, finalmente, os métodos aplicados na comunicação entre os componentes do interface, o browser VRML e o applet Java.

No capítulo 6, que encerra esta dissertação, são extraídas as respectivas conclusões e desenvolvem-se algumas linhas de orientação para trabalhos futuros.

capítulo 1.

Gráficos 3D na Internet

Um dos mais recentes desenvolvimentos na Internet foi a implantação do VRML (*Virtual Reality Modelling Language*) como uma linguagem de descrição de cenas virtuais interactivas e a três dimensões. O VRML pode ser encarado como o HTML para gráficos a três dimensões, já que proporciona um interface 3D com a WWW[1].

Neste capítulo são descritas, de forma breve, todas as versões que constituíram o desenvolvimento da linguagem VRML. Será também feito o enquadramento de vários assuntos relacionados com a criação e visualização de ambientes tridimensionais na *web*[2].

[1] WWW é a sigla para designar "*World Wide Web*" – um sistema de hipertexto com interface gráfica para procurar e aceder a informação na Internet.

[2] *Web* é outra designação, vulgarmente encontrada na literatura da especialidade para designar *World Wide Web*.

1.1 História

A Internet tem dado origem a uma crescente evolução de novas tecnologias e de novos *standards* no mundo da comunicação e dos computadores. Um dos mais notáveis destes *standards* é a linguagem HTML[3]. A necessidade de um *standard* equivalente ao HTML mas orientado para os gráficos a três dimensões foi entretanto reconhecida na 1ª Conferência Internacional da *World Wide Web* em Genebra, 1994 [WWW94].

Nessa conferência, Tim Berners-Lee e Dave Raggett (dois pioneiros colaboradores da WWW), organizaram uma sessão informal para discutir a criação de um interface 3D para a WWW. Seguiram-se apresentações de trabalhos neste campo e concluiu-se que qualquer progresso no futuro da *web* necessitaria de uma nova linguagem de descrição de cenas 3D interactivas. Surgia, assim, o VRML (*Virtual Reality Modeling Language*).

As pessoas envolvidas na concepção desta linguagem logo chegaram ao consenso de que a primeira versão do VRML deveria ser baseada numa linguagem de modelação 3D já existente. Isto evitaria "reinventar a roda", aumentando, assim, a velocidade de desenvolvimento da linguagem. Foram lançadas várias propostas, incluindo os formatos CDF (*Cyberspace Development Format*) da *Autodesk*, *OOGL (Object Orientated Geometry Language*) da Universidade de *Minnesota* e MSDL (*Manchester Scene Description Language*) da Universidade de *Manchester*. Contudo, e após algum debate, foi decidido que o VRML deveria ser baseado no formato *Open Inventor*, da *Silicon Graphics* [Inventor95]. As principais razões para esta decisão foram:

- O *Open Inventor* já ter sido largamente utilizado na Educação e Indústria para aplicações com visualização 3D em tempo real;
- Não ser uma linguagem que descreve somente a geometria 3D, mas também as propriedades dos objectos, tais como material, textura e iluminação, permitindo adicionar realismo a uma cena virtual;
- Ser escalável, isto é, possibilitar que novas características sejam facilmente adicionadas;

[3] HTML é a sigla de "*HyperText Markup Language*". A linguagem que serve para construir as páginas *web*. Não se trata propriamente de uma linguagem de programação, mas de simples "marcação" dos elementos da página.

- A *Silicon Graphics* ter concordado ceder o formato em domínio público.

Foi também decidido que a grande dificuldade de implementar interacções e comportamentos multi-participados seria deixado para mais tarde. Assim, a 1ª versão do VRML – VRML 1.0, lançada em 1995, iria somente descrever mundos estáticos.

A discussão continuava, entretanto, no sentido de se definir como "alargar" o *Open Inventor*, de forma a permitir plataformas independentes e ligações à WWW.

Na Tabela 1 apresenta-se um quadro-resumo que fornece uma abordagem rápida sobre a evolução do VRML ao longo dos tempos.

versão	resumo
VRML 1.0 Maio 1995	Nos finais de 1994 foi apresentado o primeiro esboço do VRML, na 2ª Conferência da WWW, em Chicago, ficando definido que a primeira versão desta linguagem seria baseada no *Open Inventor*, formato da *Silicon Graphics Inc.* (SGI). A especificação do VRML 1.0 [Bell95] foi finalizada em Maio de 1995 e incluía apenas o suporte para a construção de objectos 3D, aplicação de texturas e iluminação. No fim de 1995 havia já vários *browsers*[4] VRML disponíveis, embora muito poucos com a capacidade para suportar todas as características existentes na linguagem.
VRML 1.0c Janeiro 1996	À medida que se foram popularizando os *browsers* VRML, começaram a ser detectadas várias ambiguidades na linguagem. Este problema foi corrigido em janeiro de 1996, através de uma nova especificação designada por VRML 1.0c (*clarified*). Nesta correcção não foi adicionada nenhuma característica à linguagem.
VRML 1.1 cancelado	Nos finais de 1995, iniciaram-se algumas discussões sobre a possibilidade de melhorar algumas das características que

[4] *Browser* é o termo vulgarmente utilizado para designar o *software* que permite aceder aos recursos da Internet, no caso concreto, a visualização de Mundos Virtuais.

	tornavam a implementação do *browser* difícil ou ineficiente. Era a tentativa de criar uma versão mais completa do VRML, apelidada de VRML 1.1. As extensões que iriam ser acrescentadas a esta nova versão foram, mais tarde, aproveitadas na versão 2.0.
Moving Worlds Janeiro 1996	A versão 1.0 do VRML incluía características para a criação de mundos estáticos, inalteráveis, adequados para algumas aplicações de visualização científica e arquitectura. Para adicionar a esta linguagem suporte para animação e interacção, o grupo de arquitectura do VRML (VAG) colocou à discussão a possibilidade de alterar radicalmente a sua sintaxe. *Silicon Graphics, Netscape* e outros trabalharam em conjunto para criar a proposta *Moving Worlds*, apresentada em Janeiro de 1996. Esta proposta foi mais tarde aceite e tornou-se o ponto de partida para o desenvolvimento da segunda versão do VRML.
VRML 2.0 Agosto 1996	Em Agosto de 1996 e após sete meses de esforço intensivo por parte da comunidade do VRML, a proposta *Moving Worlds* deu, finalmente, lugar à especificação final do VRML 2.0 [Bell96]. Esta nova versão alterou radicalmente a sintaxe e adicionou um conjunto de novas características à linguagem, como, por exemplo, animação, interacção, som, fundos (*backgrounds*) e suporte para programação externa.
VRML 97 Dezembro de 1997	No início de 1997, o VAG começou a pensar na forma de apresentar a especificação do VRML 2.0 à *International Standards Organization* - ISO[5]. A versão ISO do VRML 2.0 foi revista e reescrita, de forma a clarificar alguns assuntos e foi realizado um pequeno número de alterações. A formalização ISO do VRML foi designada por VRML97 [VRML97]. Esta especificação foi ratificada pela Organização ISO em Dezembro de 1997 A maior parte dos *browsers* VRML 2.0 são, agora, *browsers* VRML 97.

Tabela 1 – Evolução do VRML, ao longo do tempo.

[5] ISO - Organização que superintende as mais importantes especificações de linguagens usadas na Comunidade Informática.

As versões VRML 1.0 e VRML 2.0 diferem radicalmente na sintaxe e nas suas características. Um *browser* VRML 1.0 não consegue visualizar um mundo criado de acordo com a versão 2.0 desta linguagem. Contudo, a maior parte dos *browsers* VRML 2.0 são capazes de visualizar mundos criados de acordo com a versão 1.0.

Quanto ao VRML 97, difere apenas em alguns pormenores da versão 2.0. Na maioria dos casos, um *browser* VRML 2.0 é capaz de visualizar correctamente ficheiros construídos em VRML 97.

1.2 Visualização de Cenas Virtuais

Quando um *web browser* acede a um ficheiro, a primeira tarefa que realiza é testar o tipo de dados que existe nesse ficheiro, ou seja, o seu MIME. Se esse ficheiro contém texto, HTML ou imagens, o *browser* visualiza o documento. Para apresentar outro tipo de informação, como áudio, vídeo ou VRML, são necessários pacotes de *software* adicionais, denominados por *plug-ins*. Um *plug-in* é um programa que "trabalha em cima" do *browser* e que permite ver informação não-HTML, aumentando desta forma as capacidades do *browser*.

Para se visualizar um mundo VRML na *web* é então necessário um *plug-in*, vulgarmente designado por *browser* VRML. O número de *browsers* VRML tem crescido rapidamente para diferentes sistemas e plataformas. Alguns dos mais populares, até ao momento em que é escrita esta dissertação, são:

- CosmoPlayer da Silicon Graphics;
- World View da Intervista;
- Liquid Reality da Dimension X;
- Community Place da Sony;

Em [Repository98] está disponível uma lista actualizada de *browsers* VRML. Em [Mitra99] e [Ressler98], podem ser encontrados testes e comparações entre os vários *browsers* VRML existentes.

A Figura 1 apresenta uma cena VRML a ser visualizada através do plug-in *CosmoPlayer*, instalado sobre o *browser* Internet Explorer 4.0.

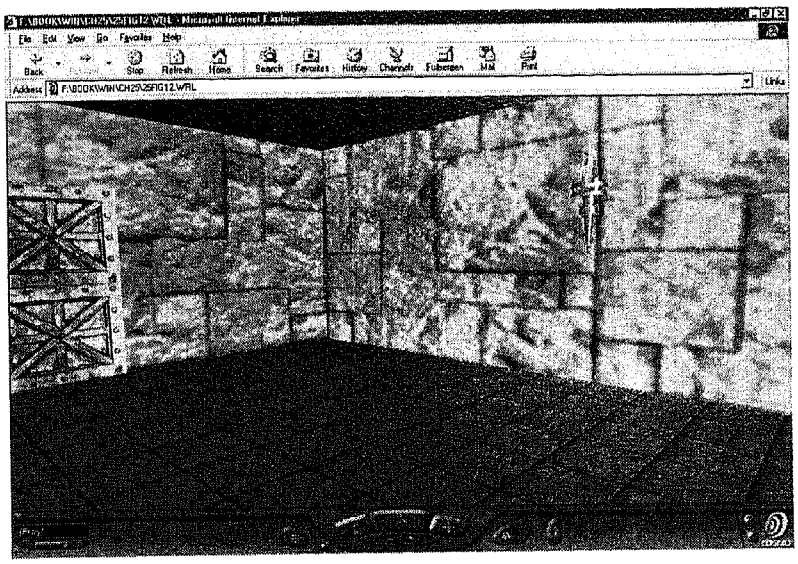


Figura 1 - Exemplo de uma cena VRML.

A especificação do VRML requer que todos os *browsers* suportem os formatos de imagem JPEG e PNG para texturas, e o formato MPEG para texturas em movimento. A maioria dos *browsers*, por razões óbvias, suportam adicionalmente o formato GIF e alguns aceitam igualmente o formato RGB da Silicon Graphics.

1.2.1 Navegação e Interacção

A maior parte dos *browsers* fornecem ao utilizador mais do que uma forma de visualizar cenas ou navegar através do mundo virtual. Embora os métodos de interacção sejam implementados de diferentes formas para cada *browser*, eles são similares. Normalmente, existe um modo de interacção que permite ao utilizador manipular o objecto 3D (rodar, mover, ..) e um modo de navegação que simula movimentos, com o rato ou teclado, através do mundo VRML. Alguns *browsers* implementam também um modo de visualização (*seek*), em que o utilizador aponta com o rato para um objecto da cena e o ponto de visão é automaticamente actualizado para esse objecto.

1.3 Exemplos de VRML na Internet

A possibilidade de criar ambientes tridimensionais na *web* proporciona oportunidades muito interessantes em diversas áreas, especialmente na arquitectura e no design. A modelação por computador é hoje largamente utilizada nestas áreas como uma

ferramenta para compreender a relação espacial entre os objectos. A importância do VRML advém do facto de permitir que estes modelos 3D possam ser visualizados por qualquer pessoa com acesso à rede, independentemente da plataforma que estiver a utilizar.

A característica multimédia adicional que permite que os objectos possam ser "ligados" a textos, sons e imagens veio tornar o VRML uma linguagem ainda mais poderosa.

Alguns campos de aplicação do VRML:

- Arte;
- Arquitectura
- Comércio;
- Engenharia;
- Visualização Científica;
- Entretenimento;

Na Figura 2 apresentam-se alguns exemplos de modelos VRML:

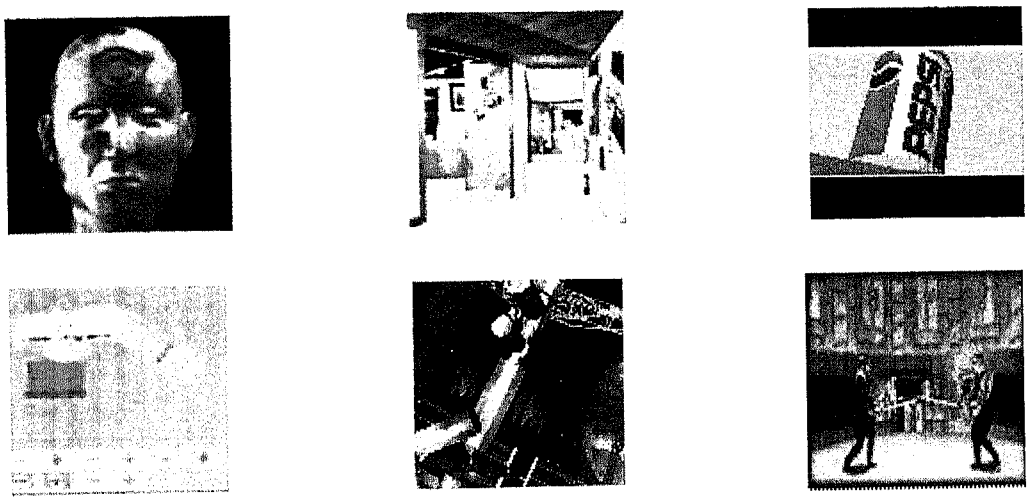


Figura 2 – Modelos VRML em diferentes áreas de aplicação.

Em [Galleries98] podem encontrar-se vários exemplos de mundos VRML, nas diferentes áreas de aplicação.

1.4 Criação de Mundos Virtuais

Para criar um mundo VRML pode utilizar-se um simples editor de texto ou ferramentas de modelação 3D. Estas ferramentas podem ser modeladores específicos de VRML ou simplesmente modeladores genéricos 3D. Neste último caso será necessário converter a informação do modelo para o formato VRML.

A forma de criar um mundo VRML está condicionada pela complexidade do modelo que se pretende construir. A seguir são apresentadas, de acordo com [Nadeau98], algumas vantagens e desvantagens na utilização de cada um dos processos definidos para a criação de um mundo VRML.

1.4.1 Editor de Texto

Dado que a linguagem VRML é composta de texto puro, a criação de um mundo VRML pode ser efectuada com um simples editor de texto. Esta situação tem como vantagens o facto de não ser necessário utilizar *software* específico e de suportar todas as características do VRML, à custa de um conhecimento profundo da linguagem por parte do programador, o que permite um controlo mais detalhado da eficiência do mundo. Mas tem desvantagens óbvias: é extremamente difícil criar objectos 3D com alguma complexidade e requer um grande domínio da linguagem VRML.

A Figura 3 apresenta um cubo e o respectivo código VRML, criado com um editor de texto.

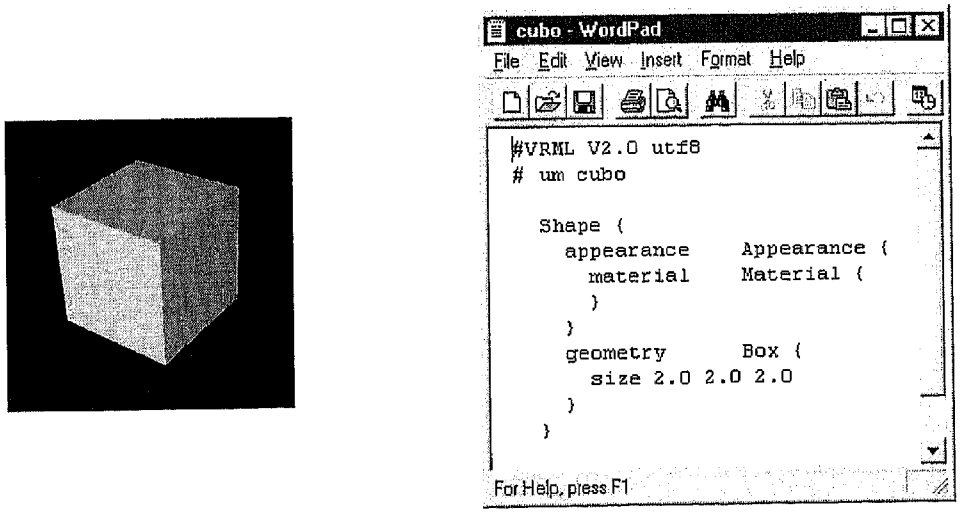


Figura 3 – Criação de uma cena VRML simples com um editor de texto.

Para a criação de cenas complexas, torna-se necessário *software* especializado de modelação. À medida que o VRML se foi tornando popular, o número de ferramentas para criar estes mundos cresceu rapidamente.

1.4.2 Modeladores de VRML

Um modelador de VRML, também designado por construtor de mundos, é uma aplicação de modelação 3D usada para criar cenas no formato VRML. O código VRML é gerado pelo próprio modelador.

Há já no mercado um variadissimo número deste tipo de modeladores. As suas vantagens são o interface "amigável" para modelar e animar objectos 3D e o facto de não requerer um grande domínio da linguagem VRML. Note-se no entanto que algumas implementações não suportam todas as características da linguagem, podendo produzir modelos pouco eficientes.

Na Figura 4 é apresentado o CosmoWorlds da Silicon Graphics, um dos mais poderosos e robustos modeladores de VRML [Schweber98] [Morrey98] [Polevoi98].

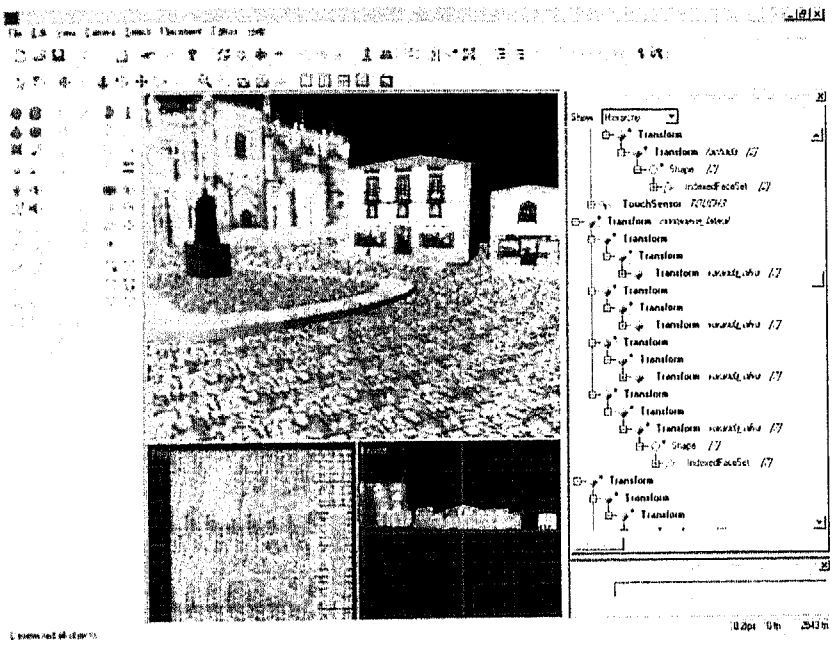


Figura 4 - Modelador de VRML - *Cosmo Worlds* da *Silicon Graphics*.

1.4.3 Modeladores Convencionais 3D

A base de um mundo VRML é a geometria 3D e há um grande número de ferramentas para converter modelos 3D de diferentes formatos (Autocad, 3D Studio, WaveFront,...) em formato VRML.

Tal facto permite que objectos complexos sejam modelados num ambiente poderoso de desenho e, posteriormente, exportados para o formato VRML. Algum *software* de modelação 3D possui mesmo capacidade intrínseca de exportação directa para o formato VRML. A vantagem de usar um modelador convencional 3D está no ambiente poderoso que normalmente possui. Como características negativas, há que referir o facto de não ser uma aplicação "desenhada" para o VRML, o que implica não suportar eventualmente todas as características da linguagem e, por consequência, produzir modelos pouco eficientes.

Em [Repository98] está disponível uma lista actualizada de *software* relacionado com a criação e visualização de mundos VRML.

1.5 Características dos Mundos VRML

A partir da versão 2.0 do VRML, os mundos 3D tornaram-se completamente dinâmicos, uma vez que acrescentou um conjunto de novas características, como sejam a animação, a interacção e o suporte para programação externa, o que tornou possível adicionar comportamentos aos objectos de uma cena. Na Tabela 2 são apresentadas algumas das características mais relevantes de um mundo VRML.

Interactividade	Os mundos VRML respondem às acções do utilizador: Portas que se abrem quando são tocadas; campainhas que tocam quando são premidas; luzes que se acendem quando o utilizador entra num novo ambiente, etc..
Animação	Objectos e criaturas que se movem; bolas que saltam; aves que voam; *avatars*[6] que passeiam, correm ou saltam, etc.
Áudio	Música de fundo, ruídos ambientes e efeitos sonoros, em geral.

[6] Representação humana num ambiente multiutilizador.

Scripts	Inclusão de programação externa, normalmente JAVA ou javascript, para adicionar comportamentos mais complexos aos objectos.
Protótipos	Criação de novos objectos que podem ser reutilizados.

Tabela 2 – Caracteristicas de um mundo VRML

1.6 Servidores de Mundos VRML

Como foi referido, uma das vantagens do VRML é a sua utilização na Internet. Tal como qualquer outro documento *web*, torna-se assim necessária a sua instalação num servidor. Algumas questões se colocam, então, ao nível da configuração e da própria natureza dos ficheiros.

Se o computador onde vai ser instalado o mundo VRML não estiver configurado para "servir" ficheiros VRML, é necessário adicionar o MIME respectivo: `model/vrml`. Contudo, existem ainda servidores com o MIME do tipo: `x-world/x-vrml`. Este foi o tipo original, enquanto a especificação do VRML não tinha ainda sido *standardizada*, ou seja, não era ainda uma versão oficial.

Uma das caracteristicas principais da natureza de um ficheiro VRML é a sua reduzida dimensão. Todavia, grande parte destes ficheiros antes de serem "lançados" na *web* precisam ainda de ser comprimidos.

1.7 Síntese

Após o aparecimento da www, depressa se chegou à conclusão que qualquer progresso no seu futuro passaria pela utilização de uma nova linguagem de descrição de cenas 3D. A 1ª versão do VRML surge em 1995 e é baseada no *Open Inventor*, formato da *Silicon Graphics*. Esta versão iria somente descrever cenas 3D estáticas. A 2ª versão do VRML adicionou suporte para animação, interacção e programação, o que alterou radicalmente a sintaxe e as capacidades da linguagem. A formalização ISO do VRML foi designada por VRML 97.

Para se visualizar um mundo VRML é necessário um *plug-in*, vulgarmente designado por *browser* VRML. Um *browser* VRML permite ao utilizador navegar através de um mundo virtual, simulando movimentos com o rato ou teclado, através da cena.

A possibilidade de criar ambientes tridimensionais na *web* proporciona oportunidades muito interessantes em diversas áreas, nomeadamente na arquitectura, arte, design, visualização científica, entretenimento, etc.. A importância do VRML é que permite que estes modelos 3D possam ser visualizados por qualquer pessoa com acesso à rede, independentemente da plataforma que estiver a utilizar.

Para criar um mundo VRML basta apenas um editor de texto. Contudo, para criar cenas mais complexas é conveniente usar *software* de modelação 3D adequado. Há duas formas possíveis de modelar cenas VRML: usar um modelador específico de VRML, vulgarmente designado por construtor de mundos, ou usar um modelador 3D qualquer e, posteriormente, converter a informação dos modelos finais para o formato VRML.

Grande parte dos ficheiros VRML, apesar de apresentarem um formato compacto, precisam ainda de ser comprimidos antes de serem "lançados" na *web*.

capítulo 2.

VRML - Virtual Reality Modeling Language

Apesar de o VRML ser uma linguagem conceptualmente simples, a sua utilização de forma eficiente exige uma visão clara das suas principais características

Neste capítulo faz-se uma introdução aos conceitos básicos do VRML, tais como a terminologia utilizada na linguagem e a estrutura hierárquica de uma cena. Também se aborda o Modelo de Execução do VRML.

2.1 Estruturação em Nós

A linguagem VRML é utilizada para descrever objectos de diferentes tipos (geometria 3D, aparência, áudio, etc.). Estes objectos são vulgarmente designados por nós e representam a estrutura básica de um ficheiro VRML. Segundo [Couch97] um nó é equivalente a uma classe, numa filosofia OO - Orientada a Objectos.

No momento em que é escrita esta dissertação, o VRML conta com 54 tipos diferentes de nós, que podem ser classificados em sete categorias, de acordo com a Tabela 3.

Tipos de nós	Objectivo
Geometria	Representam a geometria no mundo VRML.
Aparência	Usados para determinar o aspecto de um objecto.
Agrupamento	Servem para agrupar vários nós.
Animação e comportamentos	Fornecem suporte para animação, som e outras actividades baseadas no tempo, assim como para programação de comportamentos.
Sensores	Detectam os movimentos e as acções do utilizador.
Ambientes	Servem para criar ambientes em determinadas áreas do mundo.
Navegação	Afectam a navegação do utilizador através do mundo.

Tabela 3 – Divisão dos vários tipos de nós por categorias.

Em [VRML97] pode ser encontrada a especificação do VRML, com a definição completa de todos os nós.

2.1.1 Campos de um Nó

Um nó é constituído por campos que são utilizados para armazenar os valores que o caracterizam, como sejam o raio de uma esfera ou a cor de um material.

Da mesma forma que uma variável, numa qualquer linguagem de programação, os campos precisam de ter um identificador e um tipo de dados. Os tipos podem ser:

- Simples, como uma variável simples (ex:. SFVec3f, SFFloat, SFRotation, etc.)
- Múltiplo, como um array (ex:. MFVec3f, MFFloat, MFRotation, etc.)

No exemplo da Figura 5, a cor de um objecto é declarada pelo campo `diffuseColor` do nó `Material`.

```
Material {
 diffuseColor 0   0.2   0.1
}
```

Figura 5 – Declaração da cor de um objecto.

No exemplo dado, o valor do campo `diffuseColor` é guardado num vector do tipo *SFColor*. Os tipos são assumidos por defeito, pelo que não é necessário declarar o tipo do campo num ficheiro VRML.

Os campos que permitem o armazenamento dos valores que caracterizam o nó podem ser de dois tipos: privados e públicos. Um campo do tipo privado é declarado na linguagem pelo termo *field* e o seu valor nunca pode ser alterado por outros nós. Um campo do tipo público é declarado na linguagem como *exposedField* e pode ser alterado por outros nós (daí designar-se por propriedade pública de um nó).

2.1.2 Entradas e Saídas de um Nó

Para além dos campos que caracterizam um nó, este ainda apresenta uma outra propriedade que permite a comunicação entre os vários nós de uma cena VRML. Os *eventIns* e *eventOuts*[1]. Um nó pode receber e enviar eventos[2] através dos seus *eventIns* e *eventOuts*.

Um *eventIn* serve para um nó receber eventos que vão alterar as suas propriedades. A sintaxe é, normalmente, *set_*<evento>, por exemplo, *set_position*, *set_color*, *set_intensity*.

Um *eventOut* serve para um nó enviar eventos, o que sucede quando alguma alteração ocorre no nó. A sintaxe é, normalmente, <evento>*_changed;* por exemplo: *position_changed*, *color_changed*.

Os *eventIns* e *eventOuts* não contêm valores associados e devem ser vistos como as portas de um nó, uma vez que as suas funções são, unicamente, deixar passar informação para dentro ou para fora do nó.

Um campo público (*exposedField*) possui ambas as declarações de evento. Considere-se o exemplo da Figura 6.

```
exposedField  foo

é equivalente a :

field foo;
eventIn set_foo;
eventOut foo_changed;
```

Figura 6 – Declaração de um *exposedField*.

[1] Os termos originais - *EventIn* e *eventOut* - irão ser utilizados ao longo desta dissertação, por não se ter encontrado um termo equivalente em português.

[2] A definição de evento é abordada na secção seguinte – Modelo de Execução do VRML; para já, fica a ideia de que um evento é uma mensagem que contém informação para circular entre os objectos de uma cena.

No exemplo dado, o *eventIn* `set_foo` é gerado quando outro nó altera o campo `foo` . O *eventOut* `foo_changed` é gerado por este nó quando alguma propriedade do campo `foo` é alterada.

Na secção 2.4 apresenta-se com maior detalhe a forma de comunicação entre nós, através dos respectivos *eventIns* e *eventOuts*.

2.2 Hierarquia de uma Cena

Uma cena VRML organiza-se segundo uma estrutura hierárquica, vulgarmente designada por *SceneGraph* (grafo da cena), descrevendo a forma como os objectos VRML se relacionam uns com os outros. A Figura 7 tenta demonstrar esta hierarquia, fazendo uma analogia com algumas partes do corpo humano.

A estrutura hierárquica tem efeitos, nomeadamente ao nível da aplicação das transformações geométricas. Assim, se a "Perna" avançar, o "Pé" e os "Dedos" acompanham o movimento. Da mesma forma, se o "Corpo" se deslocar, as restantes partes do corpo também se deslocam. O que não significa que se a "Perna" se movimentar, o "Braço" ou a "Cabeça" se desloquem também.

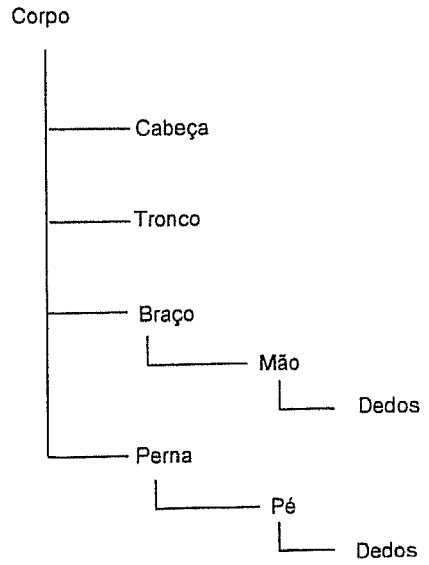


Figura 7 - Estrutura de algumas partes do corpo humano.

O VRML possui nós específicos para agrupar objectos – *Transform*, *Group*, etc. Estes nós, normalmente referidos por nós de agrupamento, possuem um campo designado por *children*, para a adição de novos nós.

Grupos de nós podem, desta forma, ser adicionados à cena, expandindo a estrutura hierárquica. Realça-se no entanto o facto de cada grupo possuir o seu próprio espaço de coordenadas, o que é essencial para a aplicação de transformações geométricas de forma estruturada [Couch97].

Na Figura 8 é apresentado um exemplo de um ficheiro VRML e a respectiva estrutura hierárquica da cena.

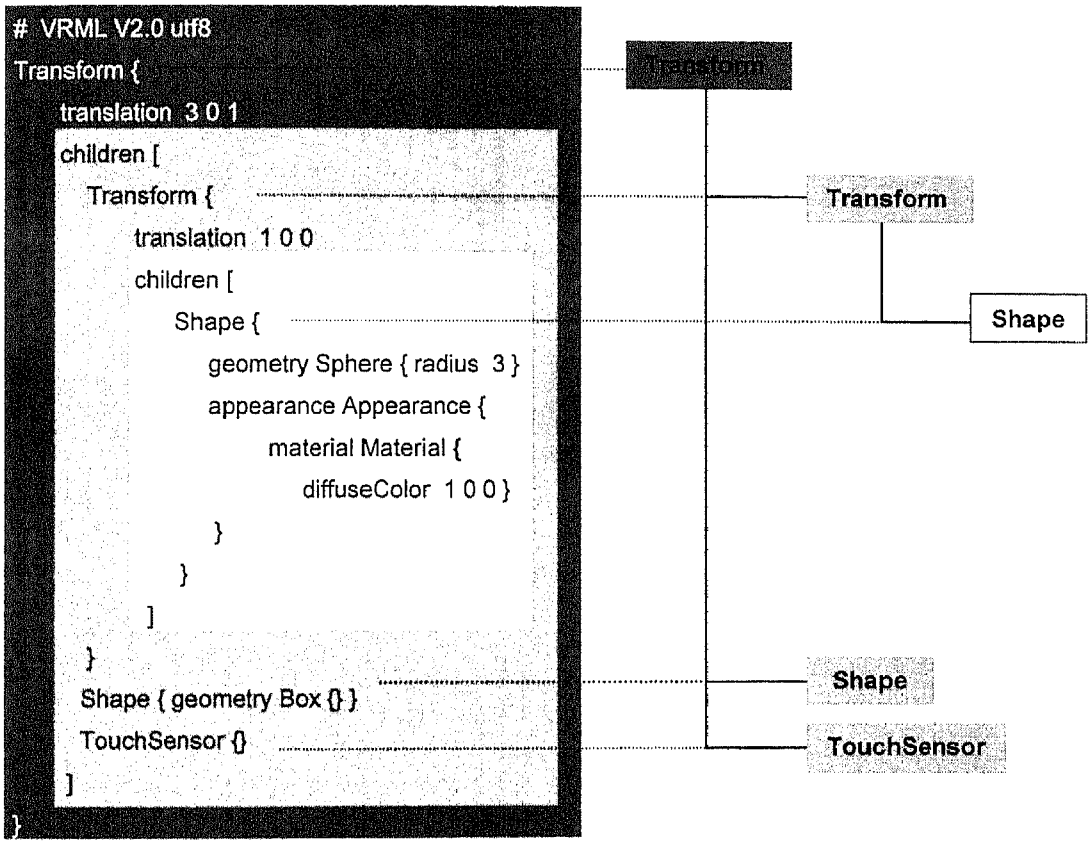


Figura 8 – Estrutura hierárquica de uma cena VRML.

De notar que no exemplo da figura foram adicionados à cena dois novos grupos de nós, através do campo *children*. O primeiro grupo assinalado a cinza-claro possui o seu próprio sistema de coordenadas, absolutamente independente. O segundo grupo assinalado a branco também possui o seu próprio espaço de coordenadas, embora seja relativo ao seu antecessor (cinza-claro).

O conceito de estruturação de uma cena VRML é bastante importante e deve estar sempre presente, uma vez que pode reduzir drasticamente a quantidade de esforço do *browser* para visualizar uma cena. Este tema é abordado, de uma forma mais cuidada, na secção 4.2 – Optimização.

2.3 Definição e Utilização de Nós

Uma das características mais interessantes e úteis do VRML é a capacidade de associar um nome a um nó e, posteriormente, poder usar essa definição na cena. Isto é particularmente importante quando se pretende criar animações e adicionar alguns comportamentos aos objectos. Efectivamente, para controlar o fluxo de eventos entre objectos de uma cena, torna-se necessário identificar os nós que fazem parte dessa interacção.

A associação de um identificador a um nó efectua-se por meio da instrução DEF, colocada antes do tipo do nó que está a ser declarado, como, por exemplo, para o objecto cubo: `DEF myBox Box {size 2 1 3}`

Posteriormente, é possível a instanciação daquele cubo através da instrução USE: `USE myBox`

A palavra reservada DEF só pode ser utilizada para associar um identificador a um nó que já exista na cena. É possível usar o mesmo DEF repetidas vezes num ficheiro, mas apenas é válida a última declaração antes da instrução USE.

Considere-se o exemplo da Figura 9. Como se pode ver, o nome *foo* é declarado duas vezes para o nó *Appearance*. Em [VRML97], "se for dado o mesmo nome a vários nós, a definição usada na instrução USE é a última DEF encontrada até então...". Desta forma, a aparência do último objecto do exemplo será uma textura, enquanto que a do segundo objecto será a cor vermelha.

Nós definidos através da palavra DEF podem ser acedidos em qualquer parte do ficheiro, imediatamente após a sua declaração. Não é possível aceder a nós declarados em ficheiros externos.

```
# Primeiro objecto
Shape {
       appearance DEF foo Appearance {
            material Material { emisiveColor 1 0 0 }
       }
       geometry Box {}
}
# Segundo objecto
Shape {
       appearance USE foo
       geometry Cylinder{}
}
# Terceiro objecto
Shape {
       appearance DEF foo Appearance {
              texture ImageTexture { url imagem.jpg }
       }
       geometry Cone {}
}
# Último objecto
Shape {
       appearance USE foo
       geometry Cylinder{}
}
```

Figura 9 – Exemplo de definição e utilização de nós (DEF/USE).

2.4 Modelo de Execução

A adição de comportamentos aos objectos de um mundo virtual realiza-se pela alteração dinâmica das suas propriedades, que pode ir da simples alteração da cor de uma esfera até comportamentos bastante complexos.

O modelo de execução do VRML implementa um mecanismo que permite aos objectos trocarem informação entre si, por forma a que as suas propriedades sejam dinamicamente alteradas.

Este mecanismo relaciona-se com a possibilidade de os nós poderem ser "ligados" uns aos outros, de forma a trocarem eventos através dos seus *eventIns* e *eventOuts*.

2.4.1 Eventos

Um evento em VRML tem um significado diferente de um evento noutra linguagem de programação, como, por exemplo, o JAVA.

Por definição, um evento é a ocorrência de um acontecimento. Em linguagem JAVA, por exemplo, quando ocorre um evento gerado por um toque num botão, é invocada uma função para tratamento desse evento, denominada *event handler* [Sun97]. Esta função contém todas as acções que devem ser executadas para aquele acontecimento. Em JAVA, um evento é apenas o "sinal" para serem executadas acções. Pelo contrário, o VRML não possui nenhum mecanismo que permita invocar uma função e, por esse motivo, é o próprio evento que contém o encapsulamento de todas as acções que devem ser executadas [Couch97].

Quando um nó pretende passar informação para outro nó, gera um evento. Este evento é uma mensagem que contém valores, similares aos valores dos campos. Assim, os valores de um evento podem ser de vários tipos: valores de cor, valores de coordenadas 3D, etc.

Um evento contém duas componentes distintas de informação: o seu valor e o *timestamp*. *O timestamp* é o instante em que o evento foi gerado na origem (não se refere ao instante em que o evento chega ao destino) e é extremamente importante para a programação de comportamentos. Normalmente, é usado para controlar algo que dependa do tempo, como por exemplo, uma animação.

É da responsabilidade do *browser* saber quando ocorreu um evento, de forma a manter a sequência correcta. Realmente, os eventos são executados pela ordem pela qual são gerados.

2.4.2 "Ligação" entre Nós

Como já foi referido anteriormente, o VRML não tem nenhum mecanismo que permita passar parâmetros, como, por exemplo, uma função. Para colmatar essa limitação, prevê uma "ligação" explícita entre o *eventOut* de um nó e o *eventIn* de outro nó, como se pode ver na Figura 10. A essa "ligação" dá-se o nome de *route*.

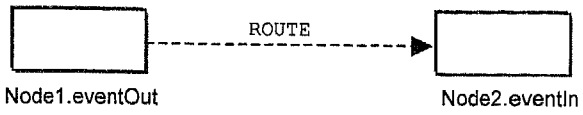


Figura 10 – "Ligação" entre dois nós.

Desta forma, é o programador quem decide qual o nó que gera um evento e qual o nó que o recebe. O programador tem, portanto, um controlo absoluto sobre o fluxo de eventos.

No entanto, existem algumas restrições de acesso às propriedades de cada nó. Por exemplo, não é possível criar ligações entre campos do tipo privado, uma vez que estes não podem ser acedidos, nem para leitura nem para escrita. São propriedade privada de um nó.

A Tabela 4 resume os tipos de acessos permitidos às propriedades de um nó.

PROPRIEDADES	TIPO DE ACESSO
campo privado	Não pode ser acedido por nós externos.
campo público	Pode ser acedido por outros nós para leitura e escrita.
eventIn	Pode ser acedido somente para escrita.
eventOut	Pode ser acedido somente leitura.

Tabela 4 – Tipos de acesso permitido às propriedades de um nó VRML

Para serem criadas ligações entre o *eventOut* de um nó e o *eventIn* de outro nó, estes precisam ter um identificador associado (secção anterior - Definição e Utilização de Nós). A "ligação" lógica entre os dois nós é criada através da instrução `route` :

```
ROUTE nóOrigem.eventOut TO nóDestino.eventIn
```

Quando é estabelecida uma *route* entre dois nós, o primeiro nó pode enviar eventos para o segundo, alterando, desta forma, as propriedades deste nó. Os dois campos, *eventOut* e *eventIn*, precisam ser do mesmo tipo, o que impede o estabelecimento de uma *route* entre campos de tipos diferentes, como sejam, um campo do tipo SFInt32 e outro do tipo MFInt32 ou SFBool e SFFloat, etc.

Quando um nó recebe um evento, reage, ora acendendo uma luz, ora iniciando uma animação, ora activando um som, etc. A acção depende do evento e, como é óbvio, das características do nó. Através do estabelecimento de *routes* entre vários nós da cena, podem ser criados circuitos complexos que, que conduzindo eventos vários, dão origem a mundos dinâmicos.

As routes podem ser colocadas em qualquer parte do ficheiro VRML, desde que tal aconteça posteriormente à definição dos nós que fazem parte da "ligação". Como foi dito na secção anterior, a definição de um nó através da sintaxe DEF está limitada ao ficheiro corrente, o que significa que as *routes* seguem a mesma regra. Não é possível "ligar" um nó do ficheiro corrente a um nó de um ficheiro externo.

2.4.3 Sensores

Os sensores são nós de um tipo especial, criados especificamente para detectarem as acções do utilizador e gerarem eventos para a cena VRML. Na Tabela 5 descrevem-se sucintamente os vários tipos de sensores existentes nesta linguagem.

NOME	DESCRIÇÃO
TouchSensor	Detecta quando o utilizador selecciona um determinado objecto.
PlaneSensor	Desloca o objecto seleccionado num movimento ao longo do plano XY.
CylinderSensor	Desloca o objecto seleccionado num movimento cilíndrico.
SphereSensor	Desloca o objecto seleccionado num movimento esférico.
VisibilitySensor	Detecta se um determinado objecto é visível para o utilizador.
ProximitySensor	Detecta quando o utilizador se aproxima de um determinado objecto.
TimeSensor	Detecta o tempo corrente; usado para animações.

Tabela 5 – Sensores VRML

De acordo com [Couch97], os sensores existentes na linguagem VRML podem ser agrupados em três categorias:

- Sensores que detectam os *inputs* do utilizador;

- Sensores de visibilidade;
- Sensores de tempo;

Os **sensores que detectam os *inputs* do utilizador** são classificados em dois tipos:
- Sensores de "toque" - *TouchSensor*;
- Sensores de "arrasto" - *PlaneSensor*, *CylinderSensor* e *SphereSensor*;

Os *TouchSensors* geram um evento quando o utilizador toca num objecto. Este toque pode ser produzido com qualquer dispositivo apontador, como, por exemplo, o "rato" ou *dataglove* (luva de Realidade Virtual).

Os sensores *PlaneSensor*, *CylinderSensor* e *SphereSensor* são usados para deslocar determinados objectos da cena. Assim, quando o utilizador "arrasta" um destes objectos, os sensores respectivos convertem os movimentos do cursor numa translação ou rotação do objecto no seu sistema de coordenadas.

O *ProximitySensor* e *VisibilitySensor* fazem parte do grupo de **sensores de visibilidade**. O *ProximitySensor* detecta quando o utilizador entra, sai ou se move dentro de uma determinada região. De notar a utilidade de "ligar" um sensor deste tipo a um circuito de animação de, por exemplo, uma porta. Assim, quando o utilizador se aproxima da porta, esta abre e quando se afasta a porta fecha. Este sensor pode ser muito útil para fornecer informação da posição corrente do utilizador. Na secção 5.5.3 este assunto será focado novamente.

O *VisibilitySensor* detecta quando uma determinada região da cena é visível pelo utilizador e é bastante utilizada para controlar determinadas acções que devam somente ser efectuadas quando essa região é visível. Seja, por exemplo, o movimento de um carrossel num parque de diversões; enquanto o carrossel não for visível para o utilizador, não há necessidade de o colocar em movimento, o que diminui o esforço do *browser*.

O *TimeSensor* é um **sensor de tempo**. De acordo com [Ames97], este tipo de sensor pode ser visto como o relógio de uma cena VRML e é muito utilizado no controlo de animações, como se verá na secção 2.4.5.

Este sensor possui quatro propriedades:
- *startTime;*
- *isActive;*

- *stopTime;*
- *loop;*

Os nomes são sugestivos. *StartTime* e *stopTime* iniciam e terminam a ocorrência de algo. *Loop* é um *booleano* que indica quando o nó deve continuar em ciclo. *IsActive* indica se o nó está em execução naquele instante.

2.4.3.1 Sensores e Objectos

Um sensor não é um nó de agrupamento, pelo que não pode conter uma lista de objectos como seus *children*. Por este motivo, um sensor tem de ser colocado dentro de um nó de agrupamento, como, por exemplo, o nó *Transform* e os objectos que pretendemos "ligar" a este sensor devem fazer parte deste grupo [Couch97]. A ordem pela qual o sensor e os outros objectos são declarados não tem qualquer importância. Assim, um sensor pode ser declarado antes ou depois de um objecto.

Na Figura 11 apresenta-se um exemplo com a forma de "ligar" um *TouchSensor* a um cubo, resultando que qualquer acção do utilizador sobre o cubo gera eventos no sensor.

```
Transform {
    translation 3 -2 5
    children [
        DEF myTouch TouchSensor {}
        Shape{
            geometry Box { size 1 1 1 }
            appearance USE myAppearance
        }
    ]
}
```

Figura 11 – "Ligação" de um sensor a um cubo.

Um sensor está limitado a actuar unicamente sobre os objectos declarados no campo *children* do seu grupo. No exemplo anterior, o cubo é o único objecto que pode gerar eventos no sensor. Porém, se neste grupo existissem 2 cubos, qualquer um deles iria gerar eventos no sensor quando fossem seleccionados.

2.4.4 Interpoladores

Os interpoladores são muito usados na programação de animações. O método de interpolação depende do tipo de interpolador e está definido na especificação do VRML.

Todos os interpoladores apresentam a mesma estrutura. Para cada lista de *keys*[3] está associada uma lista de valores, correspondentes aos valores de saída (*keyValue*), como se pode ver na Figura 12.

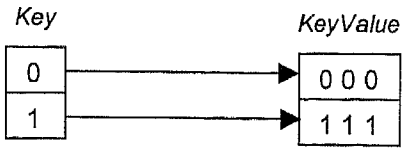


Figura 12 – Estrutura de um interpolador.

O princípio de funcionamento é o seguinte: O interpolador recebe um *input* através de um *eventIn* próprio designado *set_fraction*. Seguidamente, testa o valor recebido com os valores existentes na lista de *keys*, calcula o seu valor de saída e passa este valor para um *eventOut* próprio, designado por *value_changed*.

Para se entender um pouco melhor este processo, considere-se o código da Figura 13.

```
PositionInterpolator {
    Key [0 1]
       KeyValue [0 0 0, 1 1 1]
       set_fraction    #eventIn
       value_changed   #eventOut
}
```

Figura 13 – Exemplo de um interpolador de posição.

Neste exemplo, se o interpolador receber o valor 0.5 através do *eventIn set_fraction*, então o valor que vai gerar como *eventOut* será o ponto médio entre 0 0 0 e 1 1 1, que é, obviamente, 0.5 0.5 0.5.

[3] *Key* é o termo para designar os pontos-chave que servem de referência durante um processo de animação.

2.4.5 Animação

Animação é a alteração de algo ao longo do tempo. Por exemplo, a alteração da posição de um sistema de coordenadas implica que a posição dos objectos construídos nesse sistema seja modificado à medida que o tempo passa.

Segundo [Ames97], uma animação requer dois elementos fundamentais:

- Um relógio para controlar o *playback* da animação;
- Uma descrição de como as coisas mudam no decurso da animação;

Como foi referido anteriormente, o sensor *TimeSensor* cria um relógio que pode ser usado para controlar o tempo em que a animação decorre. Durante a animação, este sensor envia múltiplos eventos através de *eventOuts*. Durante cada ciclo, o sensor envia, através do *eventOut fraction_changed*, fracções de tempo entre 0.0 e 1.0, correspondentes a início e fim de ciclo.

Para descrever as alterações que ocorrem durante a animação, são usados os interpoladores descritos na secção anterior.

Para criar um fluxo de dados para uma animação, são usadas *routes*. É criada uma *route* entre a saída do sensor e a entrada do interpolador. Nos interpoladores são definidos os *keys* e os seus valores correspondentes nos campos *key* e *keyValue,* como já foi referido anteriormente. Cada *key* define uma fracção de tempo entre 0.0 e 1.0, à qual corresponde um valor de saída. Este valor pode corresponder, por exemplo, a uma nova posição, rotação ou factor de escala, que é então enviada pelo interpolador através de outra *route* para o nó *Transform*. No exemplo da Figura 14, foi criada uma translação do sistema de coordenadas.

```
ROUTE <TimeSensor>.fraction_changed TO <Interpolator>.set_fraction

ROUTE <Interpolator>.value_changed TO <Transform>.set_ translation
```

Figura 14– Routes necessárias para estabelecer um circuito de animação.

A partir do momento que as duas *routes* são estabelecidas, o relógio é iniciado e começa o fluxo de eventos do *TimeSensor* para o *Interpolator* e deste para o *Transform*,

causando, desta forma, uma animação do sistema de coordenadas, que pode ser eventualmente uma translação, rotação ou factor de escala. A Figura 15 tenta demonstrar, de uma forma esquemática, o fluxo de dados existente entre os nós *TimeSensor*, *Interpolator* e *Transform*.

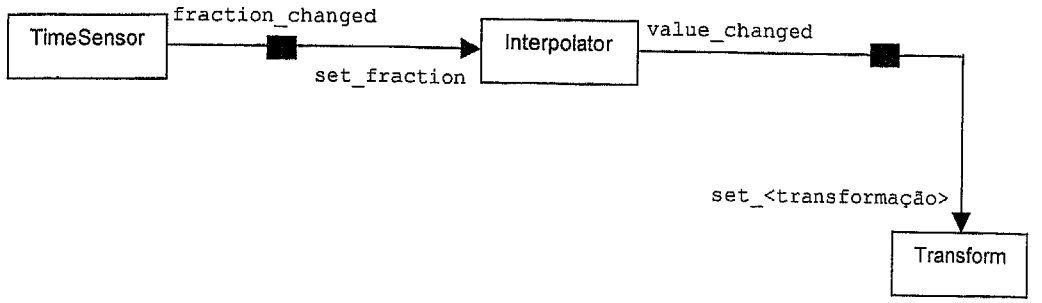


Figura 15 – Fluxo de dados numa animação constituída por uma transformação geométrica.

2.4.6 Situações Especiais

A partir do momento em que começam a ser gerados eventos na cena VRML, podem começar a surgir efeitos inesperados, como falhas na ordem de execução de eventos, *loops* e muitos outros, o que eventualmente deixa a cena fora de controlo.

A especificação não é muito flexível sobre a forma como conduzir o fluxo de uma cena. A ordem pela qual os eventos são executados é, unicamente, da responsabilidade do *browser* [Couch97]. Enquanto não existir nenhuma outra forma independente do *browser* que garanta uma determinada ordem na execução de eventos, há determinadas situações que merecem especial atenção.

2.4.6.1 Objectos com Múltiplas *Routes*

Estabelecer várias routes entre um mesmo objecto origem e vários objectos destino não apresenta qualquer problema e é normalmente referido como *Fan Out*. Na Figura 16 apresenta-se um exemplo desta técnica.

```
ROUTE interruptor1.isActive TO lamp1.turnOn
ROUTE interruptor1.isActive TO lamp2.turnOn
ROUTE interruptor1.isActive TO lamp3.turnOn
```

Figura 16 – Routes de um mesmo objecto para objectos diferentes.

Fan out é uma técnica muito utilizada para controlar vários objectos a partir da mesma fonte. Na Figura 17 é apresentado um exemplo muito simples de um interruptor ligar, ao mesmo tempo, várias luzes num quarto. Neste caso, um simples *eventOut* pode controlar vários objectos.

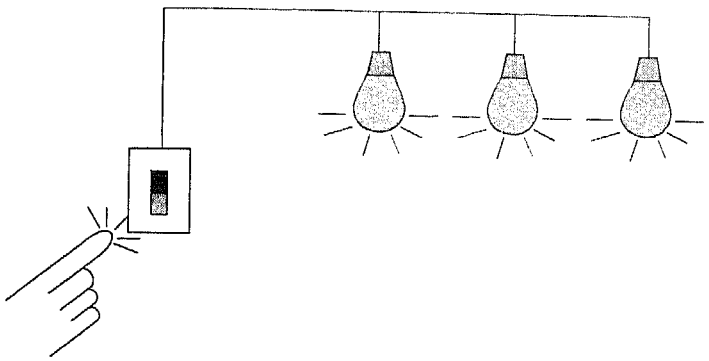


Figura 17 – Exemplo de *Fan Out*.

Uma técnica diferente, normalmente referida por *Fan In*, consiste em estabelecer várias *routes* entre diferentes objectos origem e um mesmo objecto destino, como se pode ver na Figura 18.

```
ROUTE interruptor1.isActive TO lamp1.turnOn
ROUTE interruptor2.isActive TO lamp1.turnOn
```

Figura 18 – Routes de objectos diferentes para um mesmo objecto.

Na Figura 19 apresenta-se um exemplo de *Fan In*, em que dois interruptores "ligam" em simultâneo uma luz num *hall* de entrada.

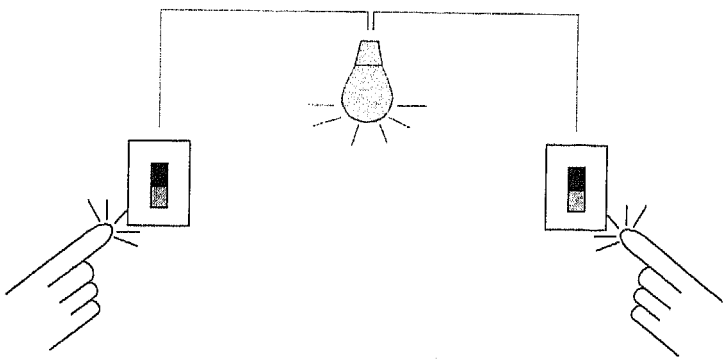


Figura 19 – Exemplo de *Fan In*.

Esta técnica pode ser útil em muitos casos, mas apresenta um grande problema para o *browser*. Se duas ou mais fontes geram um evento no mesmo instante de tempo, então acontece uma colisão, porque existem dois eventos com o mesmo *timestamp*. Quando isto acontece, o *browser* não reconhece o evento a executar em primeiro lugar e o resultado é indefinido, provocando o bloqueio do modelo de execução e, por consequência, do *browser*.

2.5 Síntese

Um nó é a estrutura básica de um ficheiro VRML e pode representar qualquer tipo de objecto (geometria, áudio, aparência, ...). As propriedades que definem um nó são os campos, os *eventIns* e os *eventOuts*.

Os campos são utilizados para armazenar os valores que caracterizam um nó e podem ser de dois tipos: privados e públicos.

Os *eventIns* e *eventOuts* de um nó servem para este receber e enviar eventos, respectivamente, permitindo a comunicação entre nós.

É possível associar um identificador a um nó, através da palavra-chave DEF e, posteriormente, usar essa definição na cena, através da instrução USE. Esta última cria instâncias do nó inicial.

O Modelo de Execução do VRML é o mecanismo que permite aos objectos trocarem informação entre si, por forma a que as suas propriedades sejam dinamicamente alteradas. Quando um nó pretende passar informação para outro nó, gera um evento. Um evento é uma mensagem que contém valores. Para um nó enviar um evento para outro nó é necessária uma "ligação" lógica entre os dois denominada *route*.

Um sensor é um nó que detecta as acções do utilizador, gerando os respectivos eventos para a cena. Desta forma, é possível detectar, por exemplo, o instante em que o utilizador selecciona um determinado objecto, ou se uma determinada região da cena é visível ou próxima do utilizador.

Em VRML, o processo de animação de um sistema de coordenadas consiste em estabelecer *routes* entre os nós *TimeSensor*, *Interpolator* e *Transform*.

Se entendermos que o VRML é uma linguagem desenhada para a Internet, então o seu objectivo fulcral será a programação de objectos 3D em "tempo real". Será, pois, de vital importância saber quais os objectos que aparecem na cena em determinado momento e a forma como os mesmos interagem. A ordem de execução de eventos é da responsabilidade do *browser* e é de extrema importância.

capítulo 3.

Programação Externa

A partir da versão 2.0, o VRML possui capacidade de adicionar alguns comportamentos aos objectos, tais como animação, detecção de acções do utilizador, execução de *clips* de áudio e de vídeo, etc..

Para atribuir comportamentos mais complexos aos objectos de uma cena é necessário recorrer a uma linguagem de programação externa. Para que tal seja possível, essa linguagem terá de possuir um interface de programação (API – *Application Programming Interface*) com o mundo VRML, ou seja, precisa de fornecer um conjunto de classes e de métodos adequados para interagir com os objectos da cena VRML.

No momento em que se escreve esta dissertação e baseado em [Couch97], as linguagens que possibilitam este interface de programação com o VRML são o Java, JavaScript e o VRMLScript, correspondendo este último a um subconjunto do JavaScript. Ao contrário do Java, o JavaScript e o VRMLScript são linguagens de texto, interpretadas directamente pelo *browser*. Estas linguagens estão mais orientadas para implementar comportamentos simples aos objectos de uma cena VRML, enquanto o Java pode já ser usado para implementar situações de grande complexidade.

Neste capítulo é feita uma abordagem genérica à linguagem Java e às formas de integração desta linguagem com o VRML.

3.1 JAVA

Java é uma linguagem de programação orientada por objectos, desenvolvida pela *Sun Microsystems*. Esta linguagem distingue-se da generalidade das restantes pelo facto de ser independente de plataformas concretas de *hardware* e de *software* [Chan97], pelo que é designada, por vários autores, como a "linguagem da Internet". Essa independência deriva de uma diferente concepção do processo de compilação de programas, baseado em máquinas virtuais.

3.1.1 Máquina Virtual JAVA

O compilador da linguagem Java cria código, não para um processador específico, mas para um processador "abstracto", que é designado por máquina virtual Java (JVM, *Java Virtual Machine*). É nesse processador que os programas são executados. O código fonte em Java dá, assim, origem a um conjunto de *bytes* formatados, designados vulgarmente por *bytecodes*, que contêm a especificação exacta das instruções que devem ser executadas na máquina virtual, como se pode constatar na Figura 20.

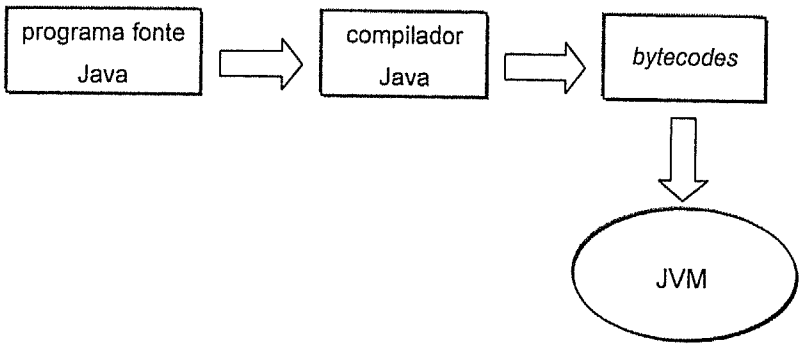


Figura 20 – Os vários passos da execução de um programa em Java.

3.1.2 Os Interpretadores JAVA

Os *bytecodes* não podem evidentemente ser executados directamente por um processador real, a não ser que o mesmo tivesse sido criado fisicamente de acordo com as especificações da "máquina virtual Java". Desta forma, para que os programas em Java, sob a forma de *bytecodes*, possam ser executados num processador real, torna-se necessária a intervenção de interpretadores. Um interpretador é o responsável pela

tradução dos programas no formato de b*ytecodes* para código nativo de um dado processador.

O facto de os programas em Java terem de ser interpretados torna a sua execução mais lenta. Como é evidente, um programa executável, contendo código nativo de um processador, executa muito mais rapidamente do que um programa constituído por *bytecodes*, que têm de ser submetidos a um processo de interpretação.

O código sob a forma de *bytecodes* perde em rapidez de execução, mas ganha em portabilidade. Praticamente todas as plataformas dispõem de interpretadores Java, o que garante a universalidade de um programa compilado em *bytecodes*. Assim, um programa não necessita de qualquer alteração para ser executado num PC, num Machintosh ou em praticamente qualquer outra plataforma. Qualquer *browser (Netscape, Internet Explorer,* etc.*)* tem integrado um interpretador de Java, o que permite a qualquer pessoa com acesso à rede visualizar páginas HTML que possuam programas nesta linguagem.

3.1.3 Os *Applets* JAVA e as Aplicações JAVA

A linguagem Java permite criar dois tipos de programas, aplicações e *applets.*

As aplicações Java possuem as características básicas de qualquer aplicação, podendo ser executadas num computador sem qualquer dependência de um *browser.*

Os *applets*, por seu turno, são programas escritos em Java que não possuem a autonomia de uma aplicação. São criados especificamente para serem executados num *browser* da *web e*, por esse motivo, precisam de ser carregados a partir de uma página HTML para o computador cliente. No entanto, no conjunto de ferramentas JAVA existe um programa – *appletviewer.exe* – que emula um *browser*, permitindo assim visualizar o resultado dos *applets*.

3.1.3.1 Ciclo de Vida de um *Applet*

Um *applet* tem vantagens significativas relativamente às aplicações, pelo facto de muitas das funções necessárias, durante a execução do *applet,* serem da responsabilidade do *browser* [Symantec97]. Assim, o ciclo de vida de um *applet* é composto por três métodos

principais - *init(), start() e stop()*, que são automaticamente invocados durante a sua execução.

O método *init( )* é invocado quando o *browser* carrega pela primeira vez o *applet* no computador cliente e é usado normalmente para inicializar as variáveis. Este método não é invocado todas as vezes que o *browser* abre a página HTML que contém o *applet*, mas somente a primeira vez.

O método *start( )* é imediatamente chamado pelo *browser* após a execução do método *init( )* e sempre que o utilizador abre a página que contém o *applet*. Quando o utilizador, por exemplo, acede a uma outra página e posteriormente regressa à página que contém o *applet*, o método *start( )* é de novo invocado.

O método *stop( )* é invocado pelo *browser* todas as vezes que o utilizador sai da página que contém o *applet* e acede a uma outra página HTML.

3.1.3.2 Limitações de um *Applet*

Pelo facto de um *applet* ser executado no lado do cliente, torna-se num óptimo veículo para a propagação de vírus na rede. Por esse motivo, foram impostas as seguintes limitações [Sun97] [Chan97] [Symantec97]:

- **Um *applet* não pode aceder ao disco local**, nem para leitura, nem para escrita; esta restrição evita a danificação de ficheiros e a propagação de vírus.

- **Um *applet* não pode executar programas no computador cliente.**

- **Um *applet* só permite estabelecer comunicação entre o computador cliente e o computador servidor do próprio *applet*.**

3.2 Integração de VRML e JAVA

A possibilidade de integração do VRML com Java veio resolver determinadas dificuldades na *web*. Efectivamente, enquanto o VRML permite criar objectos 3D num espaço tridimensional e adicionar comportamentos simples a esses objectos, o Java pode ser usado para lhes adicionar comportamentos complexos.

Nesta secção são abordadas duas formas de integrar VRML com Java, nomeadamente através de:

- *Scripting Authoring Interface;*
- *External Authoring Interface - EAI;*

3.2.1 Script Authoring Interface

A linguagem VRML possui um nó denominado *Script* que permite a comunicação de um mundo VRML com um programa externo, como se pode ver na Figura 21. As acções deste nó são definidas através de um programa escrito em qualquer linguagem de programação que o *browser* VRML suporte. As mais comuns são Java e JavaScript.

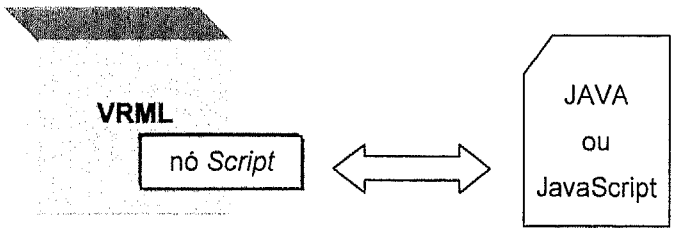


Figura 21– Comunicação entre uma cena VRML e um programa externo, através do nó *Script*.

O nó *Script* permite a uma aplicação Java ou JavaScript aceder aos nós de uma cena, usando, para o efeito, o modelo de execução do VRML, já referido na secção 2.4.

Neste modelo, um *eventOut* de um determinado nó pode ser "ligado" ao *eventIn* de outro nó, através de uma *route*. Quando o *eventOut* gera um evento, o *eventIn* é notificado e o nó processa esse evento. Adicionalmente, se o programa externo possuir um ponteiro para um determinado nó da cena, então pode enviar eventos directamente para qualquer *eventIn* desse nó e pode ler o último valor enviado por qualquer um dos seus *eventOuts* [Couch97]. Para criar um ponteiro entre uma cena VRML e um programa externo é necessário estabelecer um interface de comunicação entre os dois.

3.2.1.1 Definir um Interface para a Comunicação

Como qualquer nó VRML, o nó *Script* é composto por campos, *eventIns* e *eventOuts*. Baseado em [Ames97], são estes atributos que estabelecem o interface entre o

programa externo e o resto do mundo VRML e que criam, portanto, a possibilidade de estes se comunicarem entre si.

Na Figura 22, apresenta-se um exemplo com uma possível definição de interface para um nó *Script*:

```
Script {
   url "myScript.class"
   field        SFFloat    radius  1.0
   field        SFFloat    turn  1.0
   eventIn      SFFloat    set_fraction
   eventOut     SFVec3f    value_changed
}
```

Figura 22 – Definição de um interface para o nó *Script*.

No exemplo dado, quando o *eventIn* `set_fraction` for gerado, o programa externo indicado no campo *url* é executado e o novo valor processado é devolvido para a cena através do *eventOut* `value_changed`. Estão assim criados ponteiros entre a cena e o programa externo, através do *eventIn* `set_fraction`, e no sentido inverso, através do *eventOut* `value_changed`.

3.2.1.2 Execução do Programa Externo

O programa externo associado ao nó *Script* pode responder a três tipos de situações:

- **Inicialização**, que ocorre quando o programa externo é executado pela primeira vez. Este método é usado normalmente para configurar o estado inicial do mundo VRML a que está associado.

- ***Shutdown***, que ocorre quando o programa externo é terminado. Este método é habitualmente utilizado para o programa externo poder finalizar as acções antes de ser suspenso.

- **Distribuição de eventos**, que ocorre quando o programa externo recebe um novo evento. O princípio de funcionamento é o seguinte:

(i) O programa externo pode ter acesso a um novo evento através de um dos *eventIns* definidos no interface;

(ii) converte esse novo valor num tipo de dados apropriado à linguagem que está a utilizar, Java ou JavaScript;

(iii) processa esse dado;

(iv) envia um novo evento através de um dos *eventOuts* definidos no interface.

Na Figura 23 apresenta-se um exemplo em que uma função JavaScript é invocada a partir de um nó *Script*:

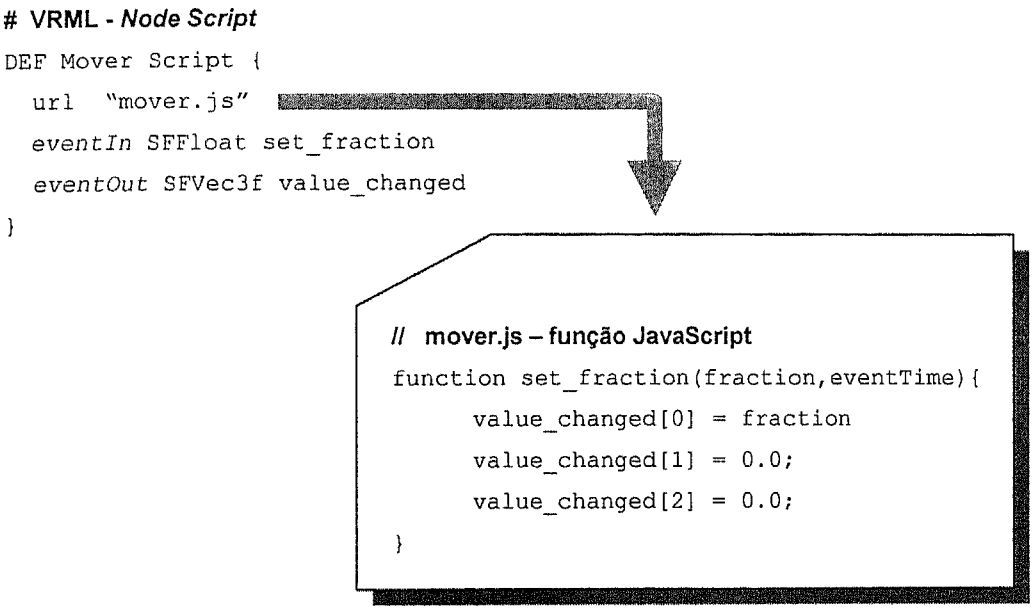


Figura 23 – Exemplo de uma função javascript executada a partir de um nó *Script*.

De notar que, no exemplo dado, o *eventIn* definido no interface vai corresponder a uma função JavaScript com o mesmo nome. Esta função é invocada com dois argumentos:

- valor do novo evento;
- *timestamp* do novo evento.

3.2.1.3 Controlar o Comportamento do *Script*

O nó *Script* possui um campo adicional para indicar se o programa externo deve ser executado sempre que um evento seja enviado para um dos seus *eventIns*. Quando o valor deste campo é TRUE, o programa externo é executado sempre que recebe um evento. Quando o valor deste campo é FALSE, a execução do programa pode ser

adiada enquanto o *browser* achar conveniente, o que pode gerar uma lista de eventos. O valor deste campo, por defeito, é FALSE.

A eficiência de um *browser* está dependente das tarefas (*workload*[1]) que este tem de realizar. Depende, em parte, da quantidade de programas que precisam ser executados num determinado instante de tempo. Usar um valor TRUE para este campo aumenta o *workload* do *browser* e, no caso de um mundo com muitos *scripts*, pode causar uma diminuição na velocidade com que a cena é visualizada, com consequente perda de interactividade [Ames97].

3.2.2 EAI - *External Authoring Interface*

Outra forma de o Java controlar um mundo VRML é usar um interface denominado *External Authoring Interface - EAI*. Este interface define um conjunto de funções que o ambiente externo pode executar de forma a afectar o mundo VRML.

O diagrama da Figura 24 mostra a forma como esta comunicação é possível, apresentando o EAI como a "ponte" que permite a comunicação entre um *applet* Java e um mundo VRML.

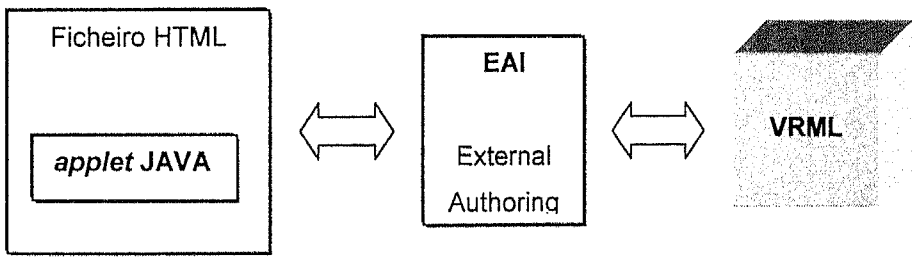


Figura 24 – Comunicação entre um *applet* Java e um mundo VRML.

A especificação final deste interface já está completa, embora se trate ainda de uma versão provisória. Foi submetida ao Grupo de Revisão do VRML – VRB, em 18 de Janeiro de 1999, para formalização ISO. Espera-se que a versão oficial esteja aprovada por volta de Janeiro/Fevereiro de 2000.

[1] *Workload* é o vocábulo original para designar carga computacional, neste caso concreto carga do *browser*.

Em [EAI99] pode-se encontrar informação completa sobre a especificação deste interface.

3.2.2.1 Conceitos Fundamentais do EAI

O EAI permite a um *applet* Java aceder aos nós de uma cena VRML, usando para o efeito o modelo de execução do VRML, já referido na secção 2.4.

O EAI permite quatro tipos de acesso a uma cena VRML [EAI99]:

1. Aceder às funcionalidades do *Browser*;
2. Enviar eventos para a cena;
3. Receber eventos da cena;
4. Ser notificado da ocorrência de eventos na cena.

Este interface externo foi desenhado posteriormente ao *Script Authoring Interface* – o interface usado no nó *Script*, descrito na secção anterior.

Quais os objectos de uma cena VRML que podem ser acedidos por um *applet* Java?

Como foi referido na secção 2.3, é possível atribuir um nome a qualquer nó da cena VRML, através da instrução DEF. Só os nós marcados com a instrução DEF podem ser acedidos pelo *applet*. Uma vez obtido um ponteiro entre o *applet* e um nó, é possível aceder aos *eventIn*s e *eventOut*s desse nó. Como os campos públicos (*exposedFields)* contêm implícitos um *eventIn* e um *eventOut*, podem igualmente ser acedidos, atribuindo-lhes o respectivo nome (*set_*<nome> ou <nome>*_changed* ou <nome>). Os campos privados de um nó não podem ser acedidos pelo *applet*.

3.2.2.2 Descrição do EAI

O EAI permite a comunicação entre um *applet* Java e um mundo VRML. As classes que permitem este interface externo estão especificadas no *package vrml.external.** .

vrml.external.Browser Classe *Browser*. Esta classe representa o mundo VRML no *applet* Java.

vrml.external.Node Classe *Node*. Esta classe representa um nó VRML.

vrml.external.field.* Classe *Field*. Esta classe representa todos os tipos de dados existentes na linguagem VRML.

vrml.external.exception.* Classes de excepção para tratamento de erros.

Acesso ao *Browser*

Para comunicar com um mundo VRML, um *applet* Java precisa, em primeiro lugar, de obter uma instância da classe *Browser*. Esta classe representa o mundo VRML dentro do *applet*. Tal é possível através do método ***getBrowser( )***, o qual devolve uma referência do *Browser*.

Na Figura 25, apresenta-se um exemplo típico em que um *applet* Java obtém uma referência do *Browser* :

`static public Browser getBrowser (Applet applet)`

método que devolve uma instância da classe Browser, devolve o primeiro *plugin* embutido no *frame* corrente.

***applet* Java**

```
...
public class MyClass extends Applet
{
        Browser browser = null;
        Public void start ()
        {
                ...
                browser = Browser.getBrowser (this);
                ...
        }
}
...
```

Figura 25 - Exemplo típico em que um *applet* obtém uma referência do *browser*

Acesso aos Nós

Para um *applet* Java ter acesso a um determinado nó necessita obter uma instância da classe *Node* (esta classe representa um nó da cena VRML). Tal é possibilitado pelo método ***getNode( )***, o qual devolve uma referência do nó.

Na Figura 26 apresenta-se um exemplo típico em que um *applet* obtém uma referência de um nó de uma cena VRML. Neste exemplo, foi definido um ponteiro para o nó SENSOR, tornando-o, assim, num nó acessível para o *applet*.

```
public Node getNode (String nome)
```

ficheiro VRML

```
...
DEF SENSOR TouchSensor {};
...
```

***applet* Java**

```
...
Node clicker  = browser.getNode ("SENSOR");
...
```

Figura 26 - Exemplo típico de um *applet* aceder a um nó da cena VRML.

A partir do momento em que se define um ponteiro entre o *applet* e um determinado nó, os *eventIns* e os *eventOuts* desse nó podem ser acedidos através dos métodos *getEventIn( )* e *getEventOut( )* da classe *Node*.

Enviar Eventos para a Cena VRML

Para um *applet* enviar um evento para a cena VRML precisa de obter uma instância da classe *EventIn*. Esta situação é possível através da execução do método *getEventIn( )*, o qual devolve uma referência do *eventIn*. Posteriormente pode ser enviado um novo evento para este *eventIn*, através do método *setValue*( ).

A classe *EventIn* representa o interface base para todos os tipos de *EventIns* (*EventInSFVec3f, EventInSFColor, EventInSFBool*, etc.) e cada uma destas sub-classes

implementa o método *setValue( )* para o correspondente tipo. Um *eventIn* serve para um nó receber eventos. Estes eventos são usados para configurar o estado actual dos nós da cena VRML, como se pode ver na Figura 27 .

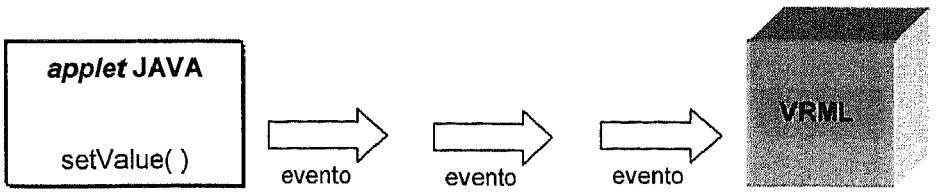


Figura 27– Enviar eventos para a cena VRML.

Na Figura 28 apresenta-se a situação típico em que um *applet* Java altera a posição de uma esfera para (3, 4, 5).

```
public EventIn getEventIn (String nome)
```

ficheiro VRML

```
DEF ESFERA Transform
{
      translation  1, 2, 3
      children [ Shape { geometry Sphere  { }  } ]
}
```

***applet* Java**

```
...
Node esfera = browser.getNode ("ESFERA");
EventInSFVec3f posição = (EventInSFVec3f) esfera.getEventIn
("set_translation");
Float [ ] nova_posição = { 3, 4, 5 };
posição.setValue (nova_posição);
 ...
```

Figura 28 – Exemplo típico em que um *applet* envia eventos para a cena VRML.

Neste exemplo, foi enviado um evento, com a nova posição da esfera, para o *eventIn set_translation* do nó ESFERA. De notar que o campo *translation* é um *exposedField* [secção 2.1.1].

Receber Eventos da Cena VRML

Para um *applet* receber eventos da cena VRML precisa de obter uma instância da classe *EventOut*. Tal é possível através da execução do método *getEventOut( )*, o qual devolve uma referência do *eventOut*.

A classe *EventOut* representa o interface base para todos os tipos de *EventOuts (EventOutSFVec3f, EventOutSFColor, EventOutSFBool*, etc.) e cada uma destas sub-classes implementa o método *getValue*( ) para o correspondente tipo. Um *eventOut* serve para um nó enviar eventos e estes são usados para duas situações:

- Acesso do *applet* ao seu valor corrente, através do método *getValue*( ), com se pode ver na Figura 29;
- Notificar o *applet* sempre que ocorra um *eventOut*, através do método *advise( )*;

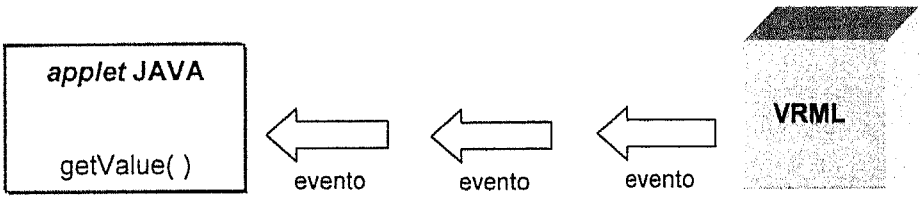


Figura 29– Receber eventos da cena VRML.

Aceder ao valor corrente de um *eventOut*

Na Figura 30 apresenta-se um exemplo típico em que um *applet* Java acede à posição da esfera anterior.

```
public EventOut getEventOut (String nome)
```

***applet* Java**

```
...
EventOutSFVec3f  posição = (EventOutSFVec3f)  esfera.getEventOut
("translation_changed");
Float [ ] nova_posição = posição.getValue();
...
```

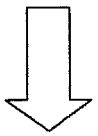
Figura 30 – Exemplo típico em que um *applet* acede a eventos da cena VRML.

Notificar o *applet* sempre que ocorre um *eventOut*

Para um *applet* ser notificado de que foi gerado um *eventOut* na cena VRML, precisa de, em primeiro lugar, implementar um interface da classe *EventOutObserver*. Pela própria filosofia do Java, o *applet* herda automaticamente todas as variáveis e métodos deste interface, tornando-se assim num "observador" de *eventOuts*.

Todas as vezes que um nó gera um *eventOut* na cena, o método *advise*() da classe *EventOut* informa o "observador", que, por sua vez, executa o método *callback*( ).

```
nóX.getEventOut( ).advise(observador,objecto)
```



```
callback(valor do evento, timestamp, objecto)
```

O processamento do evento gerado é efectuado através do método *callback( )*, da classe *EventOutObserver*. Este método recebe três argumentos: valor do evento, *timestamp* e um objecto (enviado pelo método *advise*()). Este objecto definido pelo programador serve para identificar o nó que gerou o evento e desta forma permite-se que um único "observador" processe eventos de múltiplas fontes.

Considere-se o exemplo da Figura 31: Numa cena VRML existem vários cubos, cada um deles "ligado" a um sensor diferente. Quando o utilizador selecciona um desses cubos é gerado um evento no respectivo sensor. A forma de o *applet* saber qual foi o sensor que gerou o evento é pela identificação do objecto, enviado como argumento, pelo método *advise( )*.

O exemplo dado demonstra algo que não podia ser feito através do nó *Script*, uma vez que contraria o modelo de execução do VRML. Realmente, foi criada uma "ligação" entre a cena VRML e o *applet*, através do Observador. Se existem vários nós a enviar informação para o método *callback( )*, então é criado *Fan-In* para um "nó", onde se torna possível identificar o "remetente", ao contrário do modelo de execução do VRML, que se traduzia numa colisão e num problema indefinido para o *browser* [Couch97].

ficheiro VRML

```
...
DEF sensor1 TouchSensor {}
DEF sensor2 TouchSensor {}
DEF sensor3 TouchSensor {}
...
```

***applet* Java**

```
public class MyClass extends Applet implements EventOutObserver
{
...
   public void start ( )
   {
      ...
      MyClass observador = new MyClass;
      sensor1.getEventOut("touchTime").advise(observador, new Integer(1));
      sensor2.getEventOut("touchTime").advise(observador, new Integer(2));
      sensor3.getEventOut("touchTime").advise(observador, new Integer(3));
      ...
      public void callback (EventOut valor, double timeStamp, Object obj)
      {
      // Se obj=1 então sensor1 gerou evento
      // Se obj=2 então sensor2 gerou evento
      // Se obj=3 então sensor3 gerou evento
      }
   }
}
```

Figura 31 - Exemplo típico em que um *applet* é notificado sempre que ocorre um *eventOut*.

3.3 Síntese

VRML e Java. Ambas as tecnologias foram "desenhadas" para a Internet. Porém, cada uma delas persegue objectivos diferentes. O Java tornou-se popular e derivou numa ferramenta poderosa para controlar o fluxo de uma apresentação *web*. O VRML deu a possibilidade de adicionar a essa apresentação conteúdos 3D. Java e VRML são, assim, um complemento perfeito.

- *Scripting Authoring Interface* (nó *Script*);
- *External Authoring Interface - EAI;*

A linguagem VRML possui um nó específico – *Script,* onde pode ser incluído um programa escrito em Java (também pode ser javascript ou vrmlscript). Esse programa Java comunica com o mundo VRML através de um interface denominado *Script Authoring Interface*. Através deste interface, o Java pode enviar eventos para outros nós, criar novos componentes e aceder às características da cena VRML. O programa Java recebe eventos vindos da cena VRML, processa esses eventos e envia o resultado desse processamento para outros nós da cena. Isto dá a possibilidade ao Java de adicionar comportamentos complexos aos objectos de um mundo VRML.

Outra forma de o Java controlar um mundo VRML é usando um interface denominado *External Authoring Interface - EAI*. Este interface possibilita que um *applet* Java controle uma cena VRML, como controla qualquer outro elemento da matriz multimédia: áudio vídeo, etc. Da mesma forma que pressionando um botão se pode executar um vídeo, pode igualmente enviar-se um evento para a cena, de forma a alterar a posição ou a cor de, por exemplo, uma esfera.

Ambas as formas facultam ao Java a possibilidade de controlar uma cena VRML, de uma forma flexível e poderosa. Na maioria dos casos podem ser usados em conjunto. Mas há diferenças significativas:

Quando usar o nó *Script* ?

Se o objectivo for controlar o fluxo de eventos sem sair fora do *browser* VRML, pode ser uma boa opção utilizar o nó *Script*, para adicionar comportamentos aos objectos. O nó Script aplica-se a situações que o *browser* sabe tratar, ou seja, comportamentos puramente dentro do *browser*. Não há forma de um *script* comunicar com código externo, como, por exemplo, um *applet*.

Quando usar EAI?

Se o que se pretende é controlar a cena VRML através de componentes AWT[2] - *package* Java que contém todas as classes para criar o interface gráfico com o utilizador -, torna-se necessário aceder a informação que é externa ao *browser* VRML. Numa situação

[2] AWT é a sigla para designar "Abstract Window Toolkitt" - *package* que contém todas as classes necessárias para criar o interface gráfico com o utilizador.

destas, o interface gráfico com o utilizador (botões, barras de deslocamento, etc.) deve ser definido num *applet* Java. A forma de um *applet* comunicar com um mundo VRML é através do EAI. Também é verdade que a única forma de usar o EAI é através de um *applet*.

capítulo 4.

Criação de um Mundo VRML

Dadas as limitações impostas pela actual tecnologia, um mundo virtual que não seja cuidadosamente optimizado torna-se necessariamente demasiado lento para captar o interesse do utilizador.

Neste capítulo são descritas todas as etapas envolvidas no processo de criação de um mundo VRML, dando especial atenção à sua optimização. São definidas directrizes concretas, no intuito de aumentar a interactividade de um mundo virtual.

4.1 Processo de Planificação

A Figura 32 descreve todas as etapas envolvidas no processo de criação de um mundo virtual em VRML.

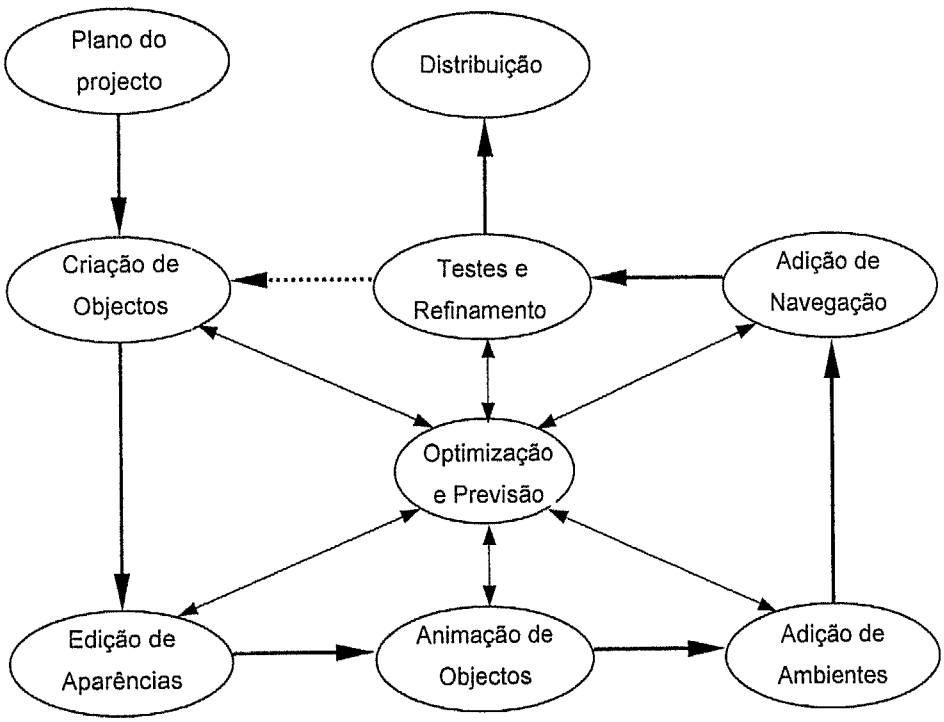


Figura 32 – Planificação do processo de criação de um mundo VRML.

Segundo [Hartman98], a construção de um mundo consiste em atravessar ciclicamente uma série de etapas. Os ciclos devem seguir uma determinada ordem e ser repetidos sempre que necessário. Na prática, porém, são dados saltos entre as várias etapas, especialmente após a primeira vez que o ciclo é percorrido. A construção envolve criar alguns objectos, posicioná-los na cena, adicionar-lhes animação, optimizar ligeiramente, criar mais alguns objectos, incluir algumas texturas para modificar a aparência de alguns objectos, optimizar mais um pouco, adicionar alguns pontos de visão, etc., intercalando com várias previsões entre as etapas.

A planificação consiste, assim, em oito etapas principais, mais duas operações suplementares, que devem ser realizadas como parte integrante de cada etapa (como a

optimização e a previsão). Nas secções seguintes discutem-se essa etapas com maior detalhe.

4.1.1 Plano do Projecto

Para construir um mundo virtual, a primeira fase consiste em decidir qual é o seu objectivo.

- Apresentar dados em 3D?
- Contar uma história?
- Apresentar um protótipo virtual de uma tecnologia que ainda não existe?
- Demonstrar técnicas médicas avançadas?
- Simular a zona histórica de uma cidade?

Nesta fase deve ser considerado que tipo de audiência vai ter este mundo e em que plataformas será executado, que tipos de interacção devem existir entre o mundo e o visitante, que tipos de comportamento devem ter os objectos – animação, com ou sem programação –, quais os conteúdos multimédia que devem ser adicionados – áudio, vídeo, texturas, páginas HTML associadas com o mundo, etc.

4.1.2 Criação de Objectos

Assim que o plano do projecto esteja bem definido, dá-se início à modelação dos objectos. A modelação de objectos para um mundo VRML é feita com base em primitivas geométricas 3D: cubos, cones, esferas, cilindros. Quando se criam novos objectos para um mundo VRML, devem usar-se modeladores que permitam controlar a geometria ao nível dos vértices, arestas e polígonos. Apenas desta forma será possível criar um mundo eficiente[1].

Uma regra básica a ter em consideração na modelação é que os objectos devem ser construídos com o menor número de polígonos possível. Uma cena VRML pode conter objectos cujas faces nunca vão ser visíveis, seja qual for o ponto de visão. Estas faces denominam-se *backfaces* e devem ser removidas, dentro do possível, logo a seguir à criação do objecto.

[1] Este assunto é abordado com maior detalhe na secção 4.2 - Optimização.

A Figura 33 mostra um exemplo com dois cubos, cujas duas faces em contacto nunca serão visíveis.

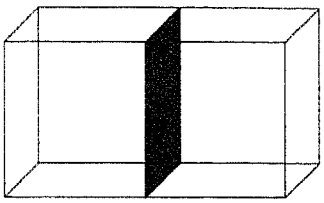


Figura 33 – Exemplo de *backfaces*.

Quando os objectos estão concluídos, é necessário posicioná-los no mundo. Aqui torna-se necessário usar variadas ferramentas de manipulação, de forma a alterar as posições relativas dos objectos, os seus tamanhos, orientação, etc.

Quando se importam objectos criados noutro tipo de *software* 3D (por exemplo, 3D *Studio*) para um modelador de VRML, deve ter-se o cuidado de os optimizar previamente de forma a serem utilizados em aplicações de "tempo real".

4.1.3 Edição de Aparências

Nesta fase deve-se dar aos objectos o aspecto que se pretende, adicionando materiais e texturas. A aparência de um objecto é a combinação do seu material, que inclui cor, brilho e transparência, com a textura que lhe é aplicada.

4.1.3.1 Materiais

Se um objecto VRML não tem qualquer material atribuído, significa que este objecto não tem capacidade de reflectir a luz e, portanto, é visualizado na cena apenas com a componente de iluminação ambiente, apresentando-se sem qualquer sombreamento e com uma cor constante ao longo de todas as facetas. Em VRML esta cor é, por defeito, um branco brilhante.

A Figura 34(a) representa um cubo sem material atribuído. A Figura 34(b) representa o mesmo cubo, agora com um tom cinzento claro, atribuído ao respectivo nó Material. De notar que é o sombreamento que dá a um objecto a sensação de tridimensionalidade.

(a) (b)

Figura 34 – Exemplo de um cubo, (a) sem material atribuído, (b) com material atribuído.

4.1.3.2 Texturas

Texturas são imagens 2D que são aplicadas aos objectos, de forma a aumentar o detalhe e o realismo das suas superfícies. Segundo [Foley94], esta técnica foi iniciada por [Catm74] e posteriormente refinada por [Blin76].

Quando é aplicada uma textura ao cubo da figura anterior, este passa a apresentar maior profundidade e realismo, especialmente quando simultaneamente tem um material atribuído, como se pode ver na Figura 35.

(a) (b)

Figura 35 – Exemplo de um cubo com uma textura atribuída e, (a) sem material definido, (b) com material definido.

Quando é aplicada uma textura a um objecto, a informação de cor que a textura contém sobrepõe-se à informação de cor do material associado a esse objecto. Por exemplo, o cubo da Figura 35(b) apresentaria o mesmo aspecto, ainda que o material que está associado ao objecto tivesse cor amarela ou verde.

O mesmo não acontece quando a textura é uma imagem *grayscale*[2]. A informação de cinzentos que estas imagens contêm não se sobrepõe à informação de cor que está associada ao material do objecto. Em vez disso, o valor de cor associado ao material vai ser multiplicado pelo valor de cinzentos da própria textura. Isto significa que, se for atribuída uma imagem *grayscale* a um material vermelho, o resultado será o equivalente a uma textura vermelha com vários tons de claro e escuro. Desta forma, a textura ocupa apenas um terço do tamanho que ocuparia a mesma imagem se tivesse as três componentes de cor - RGB[3]. Esta é uma excelente técnica para criar efeitos nas texturas (com o aspecto de envelhecido ou sujo) sem usar muita largura de banda.

A Figura 36 apresenta um modelo criado na aplicação a que esta dissertação se refere e no qual a técnica referida foi utilizada para aplicar as várias tonalidades visíveis na fachada.

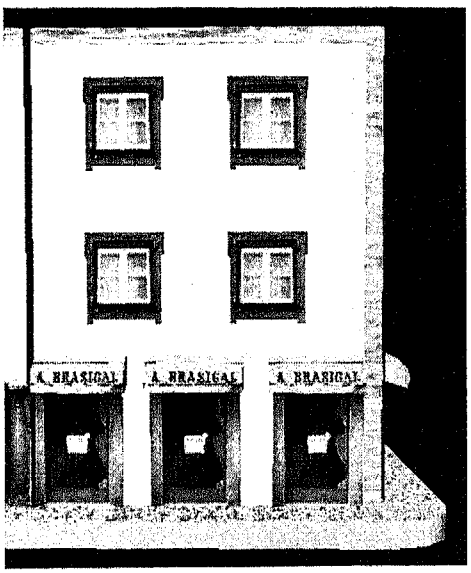


Figura 36 – Modelo criado no CosmoWorlds. Efeito de uma parede suja e envelhecida, através da combinação de um material branco com uma textura *grayscale*.

Sistemas de Coordenadas em Texturas

O sistema de coordenadas utilizado para descrever as posições de uma imagem usada como textura é denominado espaço da textura. Este sistema usa as coordenadas u e v

[2] Imagens *grayscale* são aquelas que apresentam apenas uma componente de cor.

[3] RGB (*Red, Green, Blue*) – Iniciais, em inglês, de vermelho, verde e azul. O RGB é um modelo de cores baseado nessas três componentes primárias e utilizado como padrão nos monitores.

para definir qualquer posição na textura. Como se pode ver na Figura 37, u representa a componente horizontal e v representa a componente vertical, ambas normalizadas.

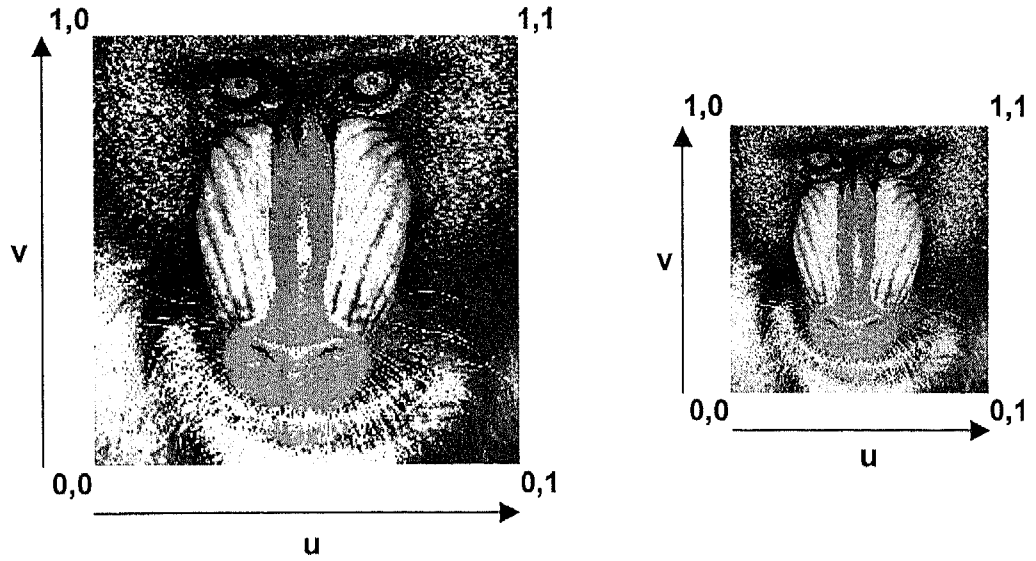
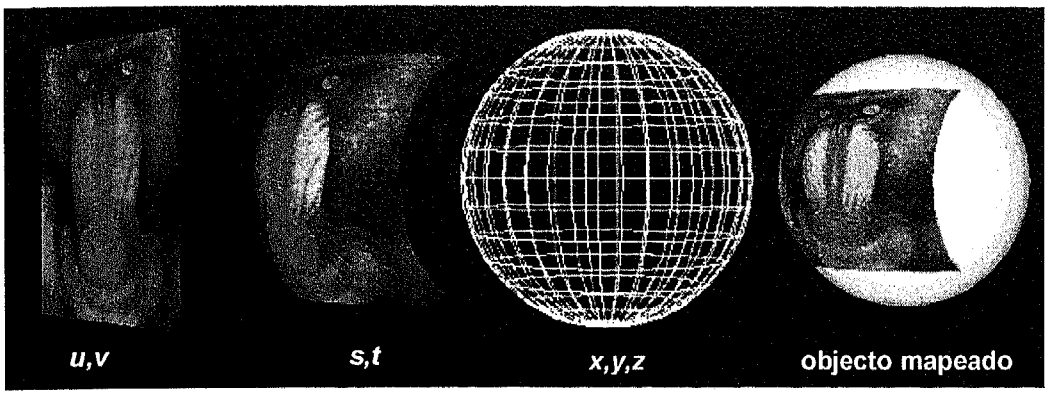


Figura 37 – Espaço de uma textura: sistema de coordenadas u, v.

De acordo com [Ballreich97], a grande diferença entre as coordenadas de uma textura (u, v) e as coordenadas geométricas (x, y, z) definidas no espaço 3D reside no seu significado. As coordenadas geométricas (x, y, z) definem um sistema de referência para posicionar geometria no espaço tridimensional (um objecto existe num determinado sistema de coordenadas 3D). Por outro lado, as coordenadas (u, v) referenciam somente o espaço 2D de uma textura e não a sua posição no espaço 3D, ou a sua posição no objecto. Esta é a razão pela qual as coordenadas u, v são normalizadas entre 0 e 1.

O processo de colocar uma textura sobre a superfície de um objecto levanta novas questões. Como o espaço de uma textura é bidimensional e as superfícies onde as texturas são aplicadas são, por vezes, de objectos 3D complexos, é necessário um outro sistema de coordenadas para estabelecer a ligação entre o mapa da textura e a superfície geométrica. Esta ligação é feita através do espaço da superfície. Este espaço é definido através de um novo sistema de coordenadas *s* e *t*, sendo *s* a componente horizontal e *t* a componente vertical.

A Figura 38 demonstra o processo de mapeamento de uma textura 2D sobre uma superfície de um objecto 3D.



(a) (b) (c) (d)

Figura 38 – Mapeamento de uma textura. (a) Imagem 2D definida no espaço da textura, (b) textura definida no espaço da superfície, (c) espaço 3D do objecto, (d) textura mapeada no objecto.

As coordenadas (s, t) definem repetições das coordenadas (u, v) (Figura 39). Sem repetição, as coordenadas (*s,t*) são exactamente equivalentes às coordenadas (*u,v*). Com repetições, torna-se possível repetir uma textura sobre uma superfície maior – processo mosaísta ou *tile*.

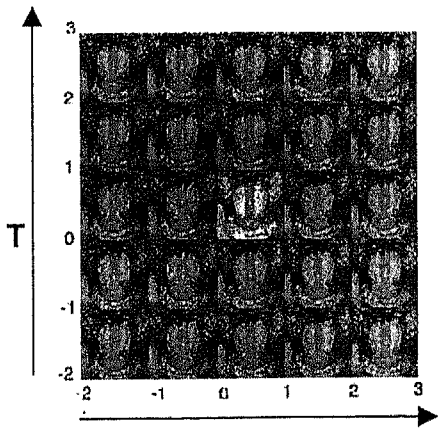
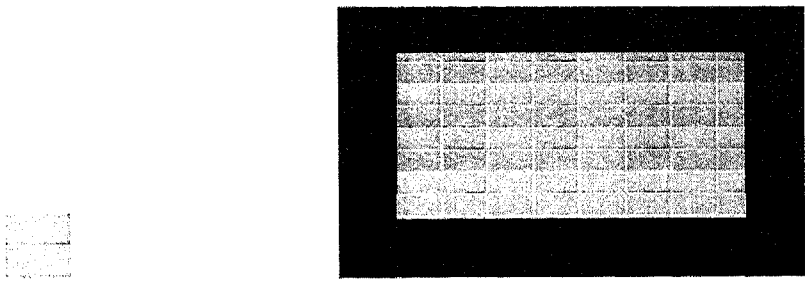


Figura 39 – Repetição de uma textura no espaço (*s,t*) – *tile*

Cobrir grandes áreas com texturas pode ser um desafio. Uma textura com uma grande resolução pode ter muito bom aspecto, mas torna-se de difícil tratamento porque leva demasiado tempo a ser visualizada e, muitas vezes, bloqueia o *browser*. Uma textura pequena é visualizada mais rapidamente mas, quando é escalada para cobrir uma grande área, a imagem degrada-se por inclusão de ruído e de *aliasing*.

Em algumas situações, é possível resolver o problema anterior por recurso ao processo mosaísta. Um bom exemplo deste processo é cobrir um chão com azulejo ou uma parede com tijolo, como se pode observar na Figura 40.



(a) (b)

Figura 40 – *Tiling* de uma textura. (a) pequena textura - 64x64 *pixels*, (b) repetição mosaísta da textura no espaço (s,t), de forma a cobrir uma grande área.

Tamanho de um Textura

Um dos principais factores a considerar na criação de uma textura é o seu tamanho. O número de *pixels*[4] de uma textura pode afectar muitos aspectos de uma cena VRML, como, por exemplo, a eficiência do modelo.

O primeiro e mais óbvio efeito que o tamanho de uma textura pode provocar num mundo VRML é o tempo de *download*[5] da cena. Uma textura grande implica um ficheiro VRML maior e, consequentemente, um maior tempo de *download*.

Uma outra grande questão reside na forma como as texturas são processadas pelo *browser* VRML. Segundo [Ballreich97], uma textura deve obedecer a algumas regras, de modo a não criar problemas quando é efectuado o *rendering*[6] da cena:

- ser o mais pequena possível;
- ser quadrada (64x64, 128x128, ...);
- a sua resolução ser uma potência de dois (16, 32, 64, 128, 256, ...).

[4] *Pixels* são os pontos que compõem uma imagem no écrã. O termo original *"pixel"* é uma corruptela da expressão inglesa *"picture element"*.

[5] *Download* consiste em transferir ficheiros para o computador através da rede.

[6] *Rendering* é o processo pelo qual um computador calcula uma imagem final a partir de um modelo tridimensional, desenhando as superfícies, as texturas e a iluminação.

Formatos Gráficos para Texturas

Um outro factor importante a considerar na criação de texturas é o formato gráfico das imagens. Os métodos de armazenamento de uma imagem num ficheiro são inúmeros, sendo alguns mais eficientes do que outros. Os formatos gráficos mais usados, na representação de texturas para VRML são os seguintes:

- *GIF (Graphics Interchange Format)* [GIF90];
- *JPEG (Joint Photographic Experts Group)* [Hamilton92];
- *PNG (Portable Network Graphics)* [Adler99]

A especificação do VRML requer que todos os *browsers* suportem os formatos gráficos *JPEG* e *PNG* e, em complemento, recomenda que todos os *browsers* suportem o formato *GIF*. Na prática, todos os *browsers* suportam os formatos *JPEG* e *GIF* e apenas os mais populares suportam o formato *PNG*, por este ser recente.

Em adição aos três formatos mencionados, a maioria dos *browsers* VRML suportam igualmente outros tipos de imagens. Por exemplo, o CosmoPlayer suporta o formato RGB da Silicon Graphics.

A não ser que surjam necessidades muito específicas, devem ser evitados outros formatos gráficos para o armazenamento de texturas que não os *GIF*, *PNG* e *JPEG* [Ballreich97].

4.1.4 Animação de Objectos

O objectivo da componente animação em VRML é dar movimento e vida ao mundo, mas é limitada nas suas capacidades [secção 2.4.5]. A adição de comportamentos complexos aos objectos pode tornar um mundo rico em interactividade, mas requer a utilização de programação externa, usando Java ou JavaScript [secção 3.2].

4.1.5 Ambientes

Para acrescentar algum realismo a um mundo virtual é possível adicionar fontes de luz e sons, criar terrenos, incluir *backgrounds* e nevoeiro. A inclusão de iluminação é um ponto fundamental nomeadamente ao nível das aplicações para Arquitectura e Urbanismo [Coelho85], temas com que esta dissertação se relaciona e, por isso, merece uma atenção especial.

4.1.5.1 Iluminação de Cenas

Por defeito, um *browser* adiciona automaticamente uma fonte de luz a qualquer cena VRML. À medida que o utilizador se move na cena, essa luz acompanha o seu movimento e ilumina todos os objectos visíveis para aquele ponto de visão. Pode-se imaginar que esta fonte de luz está "agarrada" à cabeça do utilizador, como se pode observar na Figura 41, pelo que se denomina *headlight*[7].

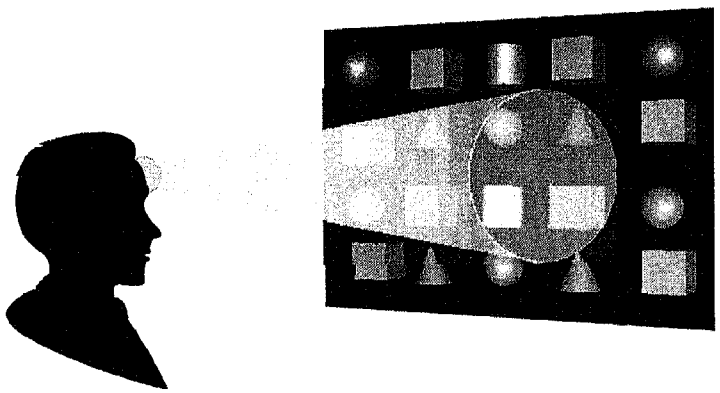


Figura 41 – Luz criada automaticamente pelo *browser* – *headlight* (como uma lanterna na cabeça de um mineiro).

Além da *headlight*, o VRML prevê outros tipos de fontes de luz:

- Luz Direccional *(DirectionalLight);*
- Luz Omnidireccional *(PointLight);*
- Holofote *(SpotLight);*

Uma fonte de luz direccional permite dirigir raios luminosos, paralelos e numa determinada direcção, a partir de uma fonte de luz infinitamente afastada, como o sol. A posição da cena onde uma fonte de luz deste tipo é colocada é irrelevante, uma vez que ela ilumina da mesma forma todos os objectos da cena (Figura 42).

[7] *Headlight* é o termo que, em português, corresponde a "farol dianteiro", numa tradução livre. Contudo, é suposto que o mesmo perde algum significado relativamente ao termo original e, por conseguinte, este será utilizado no decurso do texto.

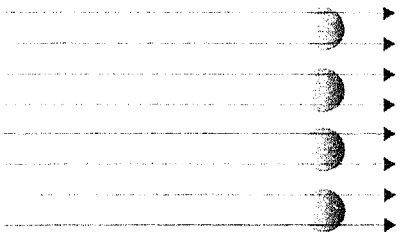


Figura 42 – Luz Direccional: os raios de luz são projectados em direcções paralelas.

Uma fonte de luz omnidireccional possui uma posição específica na cena. Como se pode ver na Figura 43, os raios de luz partem desta posição em todas as direcções, como se de uma lâmpada incandescente se tratasse.

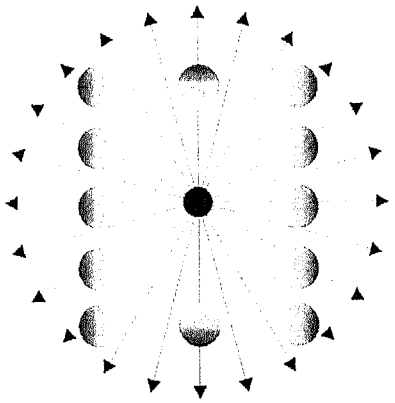


Figura 43 – Luz Omnidireccional: os raios de luz são projectados em todas as direcções.

Uma fonte de luz do tipo holofote possui também uma posição específica na cena, a partir da qual irradia, numa determinada direcção, um cone de luz (Figura 44)

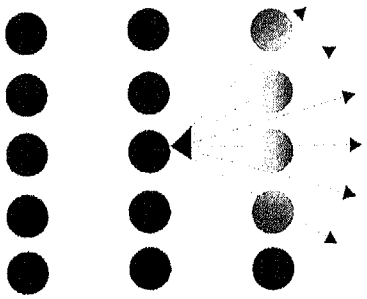


Figura 44 – Holofote: os raios de luz são projectados numa área definida por um cone.

Devem ser tidos alguns cuidados na utilização de luzes em cenas VRML, uma vez que aumentam drasticamente o *workload* do *browser*. Este assunto é referido na secção 4.2 - Optimização.

4.1.6 Navegação

Um mundo virtual, por mais realista que seja, só atrai a atenção do visitante se for de fácil navegação. Assim, mundos demasiado extensos devem incluir ajudas de navegação, por exemplo diferentes pontos de visão predefinidos.

Por outro lado, um mundo virtual deve dar ao visitante a sensação de presença física, pelo que devem ser previstas detecções de colisões que evitem, por exemplo, a transposição de paredes. Esta é uma ajuda à navegação na cena, dado que mantém o utilizador dentro do espaço navegável.

4.1.7 Testes e Refinamentos

Nesta etapa de criação de um mundo VRML, deve ser considerado que a plataforma usada para criar o mundo não é normalmente a mesma que é usada para a sua visualização. Um ponto a ter em atenção relativamente à plataforma destino é a sua capacidade de processamento gráfico e do acesso à rede.

Nesse sentido, devem ser feitos testes em máquinas diferentes de forma a avaliar a eficiência do modelo e, se necessário, voltar às etapas anteriores para que este seja novamente optimizado. Esquecer esta etapa pode implicar ficheiros demasiado grandes, complexos e em geral ineficientes para serem visualizados em plataformas de grande divulgação, vulgarmente caracterizadas por um menor desempenho.

4.1.8 Distribuição

Para um mundo VRML ficar acessível na Internet é necessário colocar num servidor todos os ficheiros que o compõem. Grande parte dos ficheiros VRML têm de ser comprimidos antes de serem disponibilizados na *web*.

Do que foi descrito anteriormente conclui-se que durante todo o ciclo de criação de um mundo, composto pelas oito etapas anteriores, a eficiência é uma questão fundamental.

Controlar a eficiência de uma cena é um processo iterativo que não pode ser deixado para o final, antes pelo contrário, deve estar sempre presente e continuamente sujeito a testes e melhoramentos.

Na secção seguinte abordam-se algumas das técnicas de optimização usadas no VRML para aumentarem a eficiência de um mundo.

4.2 Optimização de um Mundo VRML

Na construção de um mundo VRML, a optimização é uma parte essencial. Dadas as limitações impostas pela actual tecnologia, nomeadamente a largura de banda das redes de telecomunicações e a velocidade de *rendering de um PC*, um mundo virtual que não seja cuidadosamente optimizado é certamente muito lento para captar o interesse do utilizador.

4.2.1 Avaliação da Eficiência

Há dois aspectos a considerar aquando da avaliação da eficiência de um mundo VRML:

- tempo de *download*;
- velocidade de *rendering*;

O tempo de *download* depende directamente do tamanho dos ficheiros que compõem o mundo, pelo que estes devem apresentar, dentro do possível, um tamanho reduzido.

A velocidade de *rendering* tem dependências de vários outros factores, que vão desde a complexidade da geometria, passando pela representação de texturas até à iluminação.

A dificuldade existente em avaliar os dois factores é real e deve-se ao facto de o potencial utilizador poder usar um leque variadíssimo de plataformas, com diferentes capacidades de *rendering* e velocidades de acesso à rede. Este leque pode ir desde um PC sem aceleração gráfica por *hardware* e com um *modem* a 14,4 Kbps até ao mais avançado sistema gráfico, como seja uma *Onyx2* da *Silicon Graphics* com *InfiniteReality* e uma ligação directa à Internet.

Na secção seguinte apresentam-se algumas directrizes que podem contribuir para o aumento da eficiência de um mundo e desta forma contrariar, de algum modo, as

referidas limitações. No entanto, o utilizador de um sistema com poucos recursos encontrará sempre alguns problemas de desempenho na visualização e navegação de um mundo VRML.

4.2.2 Directrizes para Aumentar a Eficiência

Algumas considerações para produzir ficheiros VRML pequenos com *renderings* rápidos são:

- Medir frequentemente a eficiência do mundo em diferentes plataformas. Deve existir um compromisso entre os conteúdos da cena e a eficiência.
- Usar o menor número de polígonos possíveis.
- Recorrer a texturas, materiais e luzes para criar o efeito de superfícies mais complexas, sem usar muitos polígonos. Limitar o uso de texturas e luzes para manter o tempo de *download* e a velocidade de *rendering* em níveis razoáveis.
- Estruturar cuidadosamente a hierarquia dos objectos que compõem a cena, por forma a permitir que os *browsers* a visualizem rapidamente.
- Reutilizar texturas e sons sempre que possível, e com tamanhos o mais reduzidos possível.
- Aplicar as características do VRML que ajudam a aumentar a eficiência, como por exemplo, diferentes níveis de detalhe (LOD – *level of detail*), *Billboards*, *Inlines* e Detecção de Colisões. As características são abordadas seguidamente.

Para proporcionar ao utilizador uma experiência imersiva e interactiva, o *rendering* deve ter uma taxa de transferência de, pelo menos, 15 imagens por segundo. Se esta taxa cair abaixo das 8 imagens por segundo, a interactividade torna-se lenta, pesada e incómoda [Hartman98].

4.2.2.1 Redução do Número de Polígonos

Quanto maior for o número de polígonos maior será o tamanho do ficheiro VRML, e quantos mais polígonos forem visíveis a cada instante mais lento será o *rendering* da cena. Por este motivo, um bom método para melhorar, quer o tempo de *download*, quer

a velocidade de *rendering*, é reduzir na medida do possível o número de polígonos na cena. Regra geral, para criar o efeito de superfícies mais complexas usam-se texturas com pouca resolução (uma textura com grande resolução aumenta drasticamente o tempo de *download*). A remoção de *backfaces* é também uma técnica possível mas nem sempre utilizável.

É boa regra que mundos planeados para serem vistos na Internet por um PC não tenham mais do que, aproximadamente, 1000 triângulos visíveis a cada instante e não mais do que um par de milhares de triângulos totais na cena [Hartman98].

4.2.2.2 Controlar o Detalhe

Quanto mais realista for um mundo VRML, tanto mais tempo demora o *browser* a desenhar esse mundo, o que acarreta uma perda na interactividade. Para manter um grande nível de realismo sem uma grande perda de interactividade é necessário controlar o nível de detalhe com que o *browser* constrói os objectos. Tal é possível através de um nó do tipo nível de detalhe - *LOD*.

Esta característica implica que sejam criadas múltiplas versões de diferentes complexidades de um mesmo objecto, normalmente um modelo de grande detalhe, um de médio detalhe e outro de baixo detalhe. O *browser* selecciona o modelo que é visualizado em determinado instante, baseado na distância entre o ponto de visão e o objecto. A Figura 45 tenta demonstrar este processo:

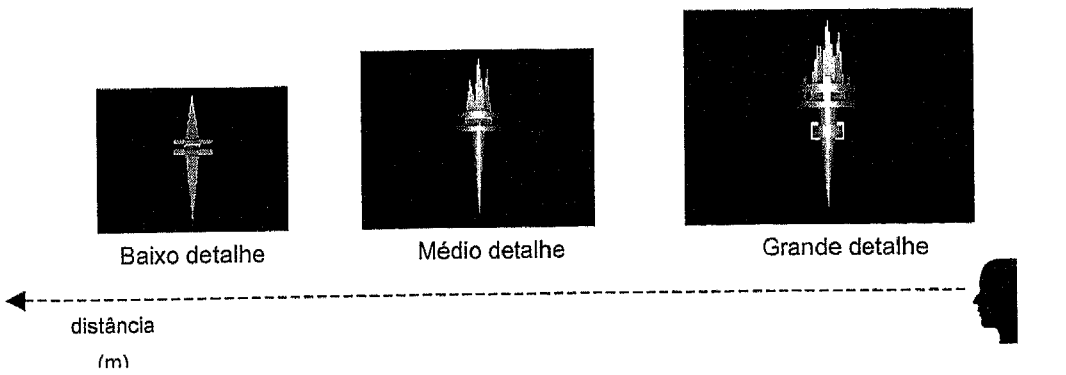


Figura 45– Três níveis de detalhe para o mesmo objecto.

No exemplo dado, o modelo que o *browser* vai desenhar depende da distância entre o ponto de visão e o objecto. A versão de grande detalhe é utilizada quando o ponto de

visão está próximo do objecto, incluindo toda a minúcia necessária para fazer com que o mesmo pareça realista. Na versão de médio detalhe são eliminados alguns pormenores suplementares, por forma a simplificar o objecto; esta versão é utilizada quando o ponto de visão está a meia distância. A versão de baixo detalhe é utilizada quando o ponto de visão está longe e por isso os objectos são o mais simplificados possível.

4.2.2.3 Objectos *Inline*

Quando uma cena VRML é demasiado complexa deve ser "partida" em mundos mais pequenos, de modo a facilitar o *download*.

Esta técnica permite reduzir o tamanho do ficheiro VRML principal, uma vez que deixa de incluir alguma geometria, normalmente aquela que não é visível no plano inicial, para dar lugar a uma referência, um apontador para um ficheiro externo, diminuindo, assim, o tempo de *download* da vista inicial.

Ao estruturar um mundo complexo, uma boa técnica é construir um ficheiro principal com um tamanho reduzido, para que o *download* seja rápido. O mesmo poderá conter *inlines*, ou seja, referências para ficheiros externos que irão conter as restantes cenas que compõem o mundo.

Na Figura 46 apresenta-se a organização espacial dos objectos que fazem parte do modelo criado no protótipo que foi implementado no âmbito desta dissertação. Os objectos assinalados não fazem parte do ficheiro principal, são objectos *inline*.

4.2.2.4 Níveis de Detalhe com Objectos *Inline*

Uma forma expedita de optimização é usar modelos de baixo detalhe no ficheiro VRML principal e modelos com maior complexidade como objectos *inline*. Assim, o tempo de *download* é rápido, uma vez que são visualizados os modelos menos complexos. Os mais complexos são visualizados quando o *browser* achar conveniente, com base na distância entre o objecto e o ponto de visão, como foi referido anteriormente.

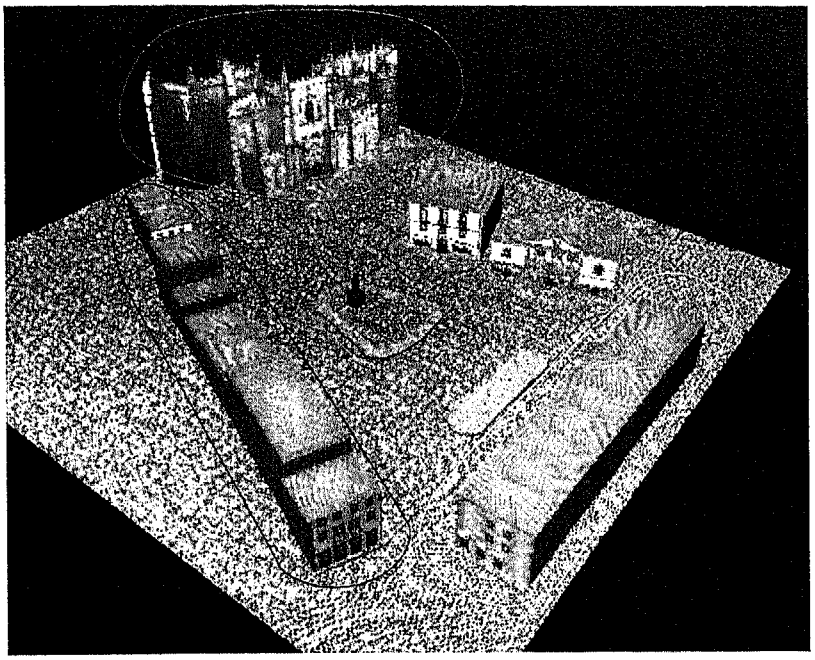


Figura 46 – Organização espacial dos objectos.

4.2.2.5 Billboard

Ao ser aplicado a um objecto, o nó *Billboard* cria um novo sistema de coordenadas e define um eixo de rotação para esse objecto. Esse sistema de coordenadas é automaticamente rodado sobre o eixo definido, de forma a que o objecto esteja sempre orientado para o ponto de visão e para que, dessa maneira, o utilizador o veja sempre de frente. A Figura 47 tenta demonstrar esse processo.

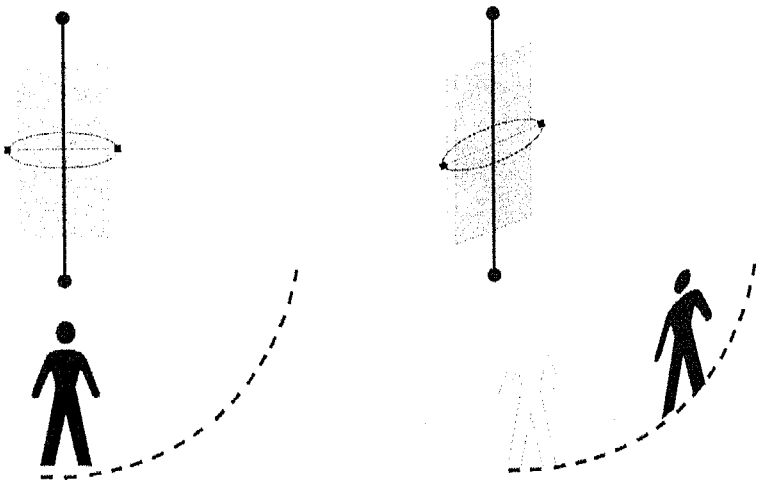


Figura 47 – Ilusão da tridimensionalidade através de um objecto *Billboard*.

Se o utilizador rodar em torno da superfície, esta acompanha a rotação e mantém-se sempre de frente para o utilizador, iludindo a tridimensionalidade.

Esta técnica é muito eficiente para criar efeitos 3D usando superfícies planas bidimensionais. Um bom exemplo de aplicação é o caso de árvores. Neste caso, basta agrupar um nó *Billboard* com uma superfície 2D e atribuir uma textura com a imagem de uma árvore a essa superfície.

Os *Billboards* reduzem fortemente o número de polígonos, o que contribui para um melhor desempenho.

4.2.2.6 Detecção de Colisões

Por defeito, o *browser* considera todos os objectos sólidos. Desta forma, não é possível passar através dos objectos, originando colisões.

Em termos de recursos, é muito dispendioso calcular as colisões de todos os objectos que compõem a cena. Uma boa solução é informar o *browser* para que não faça este teste sobre aqueles objectos que não seja provável serem colididos, como o tecto de um quarto ou um telhado de uma casa.

Desta forma, há uma diminuição dos testes de colisão e um consequente aumento na eficiência.

4.2.3 Estrutura de uma Cena VRML

Para determinar os objectos envolvidos no cálculo de visibilidade, o *browser* executa um procedimento denominado *culling test*. Este teste consiste em comparar a *bounding box*[8] de um objecto com o volume do campo de visão (ver Figura 48). Se estes dois volumes não se intersectam, então o objecto não é visível neste ponto de visão, podendo ser ignorado durante o *rendering*. Sempre que houver deslocamento do ponto de visão ou movimento do objecto, o *browser* tem de fazer um novo teste.

A estruturação da cena é fundamental, como se pode ver pela análise do exemplo seguinte. Se o modelo consiste numa parede, duas janelas e uma porta - portanto,

[8] *Bounding box* significa, em computação gráfica, o volume envolvente de um objecto.

quatro objectos -, o *browser* necessita de fazer quatro *culling tests*, um para cada objecto. Uma boa técnica será agrupar esses objectos num único. Deste modo, o *browser* fará apenas um *culling test* no caso de a *bounding box* exterior não intersectar o volume do campo de visão. Objectos que estão fisicamente próximos devem ser sempre agrupados, por forma a diminuir o cálculo de visibilidade.

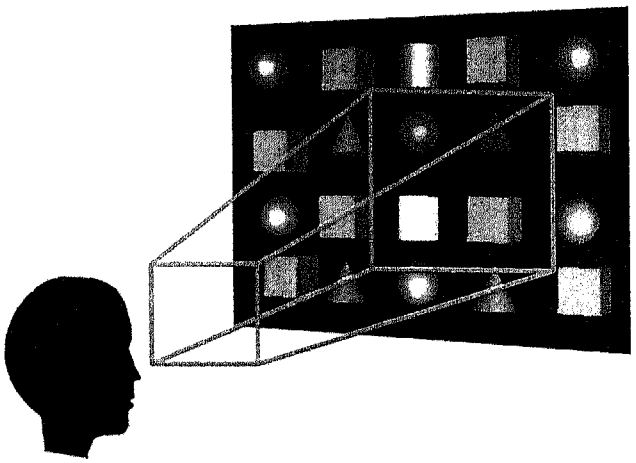


Figura 48 – Volume do campo de visão.

Por outro lado, objectos que se encontram fisicamente afastados não devem fazer parte do mesmo grupo de objectos, senão corre-se o risco de *renderings* desnecessários (sempre que a *bounding box* do grupo de objectos intersecte o volume do campo de visão, o que vai ocorrer muitas vezes).

Objectos muito grandes devem, sempre que possível, ser "partidos" em objectos mais pequenos. Por exemplo, um terreno muito extenso deve ser dividido em várias partes. Desta forma, sempre que uma parte desse terreno não faça parte do volume do campo de visão pode ser ignorada no *rendering*.

É importante ter-se em conta que, quando é adicionado mais um objecto à estrutura da cena VRML, este vai afectar o cálculo de visibilidade e consequentemente a velocidade de *rendering*.

A melhor maneira para estruturar uma cena VRML é considerar de uma forma muito lúcida a hierarquia dos objectos. Assim, devem ser respeitadas, na medida do possível, as seguintes regras:

- Objectos fisicamente próximos devem pertencer ao mesmo grupo (evitar *culling tests* desnecessários);
- Objectos fisicamente afastados não devem pertencer ao mesmo grupo (evitar *renderings* desnecessários);
- Objectos demasiado grandes devem ser "partidos " em objectos mais pequenos (evitar *renderings* desnecessários).

4.3 Síntese

Construir um mundo VRML consiste em atravessar uma série de etapas ciclicamente. Estes ciclos devem seguir uma determinada ordem e ser repetidos sempre que necessário. Na prática, são dados saltos entre as várias etapas, especialmente após a primeira vez que o ciclo é percorrido. Esta planificação pode envolver as seguintes etapas:

1. Definir qual o plano do projecto;
2. Criar objectos com base em primitivas 3D e com o menor número de polígonos possível;
3. Trabalhar na aparência dos objectos, adicionando para o efeito materiais e texturas (reduzidas) para criar o efeito de superfícies complexas;
4. Criar animações (não muito pesadas) e acrescentar programação externa (Java e/ou javascript), de forma a tornar o modelo mais interactivo;
5. Atribuir algum realismo à cena virtual, através de luzes, nevoeiro, backgrounds, etc.
6. Adicionar ao modelo informação de navegação;
7. Testar o modelo em várias máquinas, de modo a poder ser avaliada a sua eficiência e se necessário voltar às etapas anteriores para optimização.
8. Colocar todos os ficheiros utilizados na criação do modelo num servidor *web*, de modo a este ficar acessível na rede.

Durante todo este ciclo, composto pelas oito etapas anteriores, é necessário controlar, de uma forma contínua, a eficiência do modelo. Esquecer esta operação pode implicar ficheiros demasiado grandes, complexos ou desagradáveis para serem visualizados por um PC através da rede. Dadas as limitações da actual tecnologia, nomeadamente a largura de banda das redes de telecomunicações e a velocidade de *rendering* de um PC,

um mundo virtual que não seja cuidadosamente optimizado é certamente muito lento para captar o interesse do utilizador.

O VRML possui uma série de características (LOD, *billboard*, *inline*, detecção de colisões) que, bem utilizadas, podem contribuir para o aumento da eficiência de um mundo e contrariar, de algum modo, as ditas limitações. Contudo, um utilizador de um sistema com poucos recursos vai sempre encontrar alguns problemas para manter alguma *performance* na visualização de um mundo VRML.

Para diminuir o esforço do *browser* VRML e consequentemente aumentar a velocidade de processamento, a hierarquia dos objectos deve ser cuidadosamente estruturada.

capítulo 5.

Implementação de um Protótipo

Podiam ser várias as aplicações práticas do problema definido nesta dissertação. Contudo, o problema foi dirigido para uma área concreta, a remodelação de um centro histórico, tendo sido criado um protótipo como exemplo de aplicação. Este protótipo está disponível na rede, com o endereço **http://caligula.ipg.pt/ideias/virtual.html** e pode ser visualizado como um complemento desta dissertação.

Neste capítulo são apresentados todos os requisitos exigidos por este protótipo e abordadas todas as questões relacionadas com a sua implementação, nomeadamente a arquitectura do sistema, o interface da aplicação e, finalmente, os métodos aplicados na comunicação entre as componentes do interface: o *browser* VRML e o *applet* Java.

5.1 Especificação

Como foi referido na Introdução, o objectivo desta dissertação é demonstrar a possibilidade de os vários intervenientes num processo de transformação urbano, desde que disponham de uma ligação à Internet, poderem comunicar, de um forma rápida e eficiente, as várias propostas de intervenção urbana que pretendem realizar num espaço exterior.

Para o protótipo desenvolvido no âmbito desta dissertação foram definidos os seguintes requisitos:

1. Criar um ambiente virtual de um centro histórico, no caso concreto, da cidade da Guarda: a "Praça Velha";

2. Permitir uma navegação 3D interactiva pelos diferentes quarteirões e a integração de informação multimédia, de forma a possibilitar um contacto mais realista com os seus elementos;

3. Permitir a manipulação da aparência dos elementos que caracterizam o espaço exterior. De acordo com [Coelho85] e no caso concreto do espaço exterior definido para esta aplicação, os elementos que interessa aqui reter são texturas superficiais, nomeadamente fachadas, janelas, portas e pavimentos.

4. Permitir a integração de novos elementos no espaço exterior, nomeadamente elementos de equipamento predefinidos, existentes numa galeria de objectos 3D, e novos objectos com base em primitivas 3D.

   A Galeria de objectos 3D deve ser constituída por objectos que podem ser escolhidos para "mobilar" a zona exterior e, por esse motivo, deverão estar dentro do contexto visual da zona que se pretende remodelar. Cabe aos técnicos (arquitectos e paisagistas) decidir quais os objectos que deverão estar disponíveis nesta Galeria.

   Deve ser, também, incorporado um modelador que permita adicionar à cena novos objectos com base em primitivas 3D. Estes novos objectos podem ser pintados e também se lhes pode aplicar texturas. Desta forma, qualquer utilizador sem quaisquer conhecimentos de VRML tem a possibilidade de adicionar à cena novos objectos, criados por si próprio.

5. Permitir a manipulação e remoção da geometria dos novos elementos que foram integrados no espaço exterior e de alguns já existentes, através da aplicação de transformações geométricas 3D.

   Como foi referido na Introdução, conceder a possibilidade a um utilizador de intervir em alguns elementos já existentes no espaço exterior poderá levantar algumas questões no que diz respeito à possível descaracterização daquilo que é a essência do próprio espaço. A solução para este tipo de situações passará pelo desenvolvimento de mecanismos que possam restringir, de algum modo, o tipo de operação que o cidadão possa efectuar na sua intervenção.

6. Permitir que todas estas intervenções urbanas sejam enviadas, via internet, como sugestão para a Autarquia que fornece este serviço, por forma a poder ser avaliado o seu impacto urbanístico, tornando, desta forma, mais rico o confronto de ideias e a participação de todos os intervenientes no processo de transformação urbano. Esta participação poderá ficar ao nível técnico ou descer até ao nível do cidadão comum.

O protótipo deve ser projectado tendo em vista utilizadores com dois níveis distintos de conhecimento: um utilizador sem quaisquer conhecimentos de VRML e um outro que poderá ser até um especialista nesta linguagem. Este último poderá pretender adicionar objectos criados por si, ao modelo existente, usando, para o efeito, o seu próprio modelador de VRML. Como já foi referido na secção 3.1.3, não é permitido, a um *applet*[1], aceder ao disco local. Por este motivo, a forma de um utilizador deste nível poder adicionar os seus próprios modelos à cena, é enviar os ficheiros VRML correspondentes, por correio electrónico, para a Autarquia. Os novos modelos podem muito facilmente ser adicionados à Galeria e, posteriormente, integrados na cena. É obvio que caberá aos técnicos (arquitectos e paisagistas) decidir se o modelo enviado pelo utilizador está ou não adequado ao contexto visual da zona que se pretende remodelar. No caso afirmativo o modelo é adicionado à galeria.

[1] O programa que permite manipular os objectos do modelo VRML é um *applet* Java. Este assunto é referido a seguir.

5.2 Arquitectura

De acordo com a especificação efectuada na secção 5.1, planeou-se desenvolver uma arquitectura baseada numa comunicação cliente-servidor, como se pode ver na Figura 49.

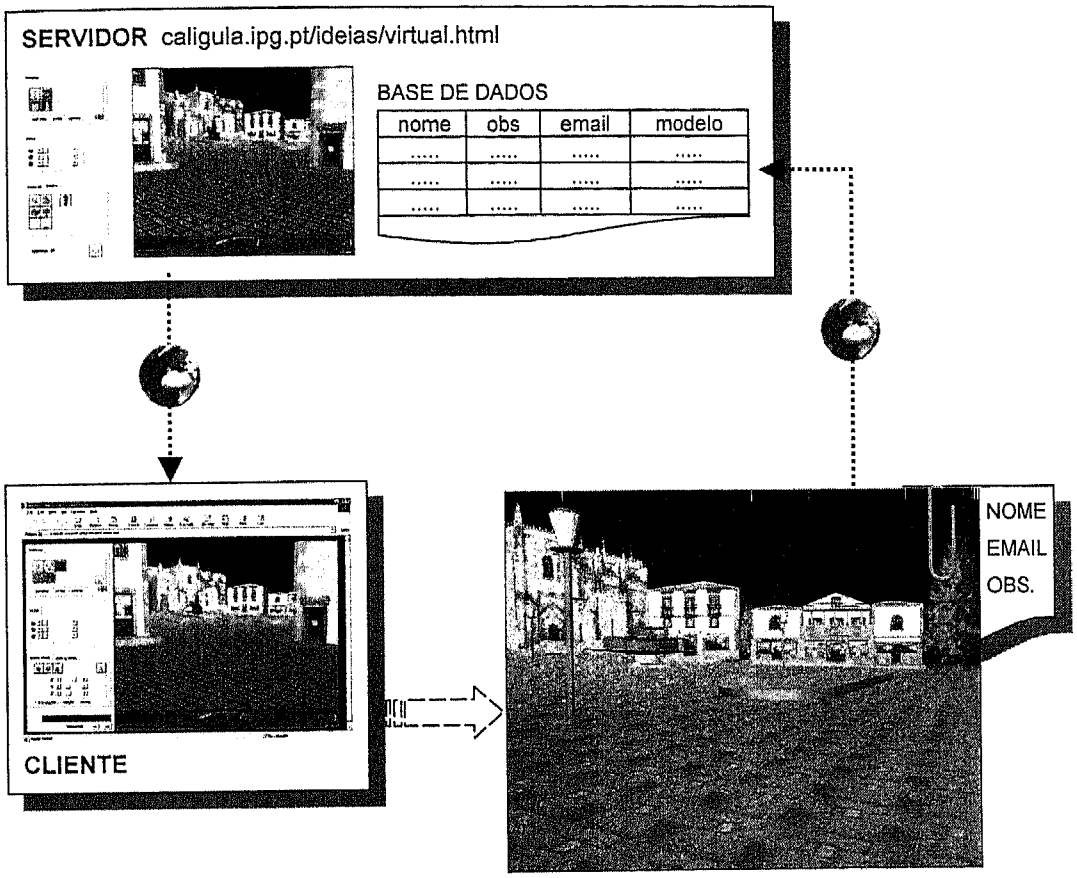


Figura 49 – Arquitectura do sistema baseada numa comunicação cliente-servidor.

Esta arquitectura permite estabelecer uma comunicação entre o potencial utilizador deste sistema (cliente) e a Autarquia que fornece este serviço (servidor).

O cliente é o programa que "corre" do lado do utilizador e constitui-se de uma página HTML composta de um mundo VRML e um *applet* Java. O mundo VRML é o modelo da zona que se pretende transformar – a "Praça Velha" e o *applet* é um programa Java que comunica com o modelo VRML, de forma a permitir a sua manipulação.

O servidor, por seu turno, é um programa que "corre" no computador da Autarquia e que, para além de fornecer o conteúdo solicitado pelo cliente, espera passivamente que esse

e/ou outros clientes lhe enviem propostas de intervenção devidamente identificadas, que armazenará numa base de dados.

Na Figura 50, apresenta-se uma hipotética base de dados para esta aplicação. De notar, que cada uma das sugestões existentes na base de dados está perfeitamente identificada com o interveniente no processo de transformação.

nome	email	observações	modelo
André Sarmento	asarm@mail.telepac.pt	Penso que a estátua do D.Sancho deve ficar no enfiamento da Rua Direita com a Rua Couceiro. Desta forma, não tapa a vista geral da Sé	sugestao0000
José Silva	js@mail.isoterica.pt		sugestao0001

Figura 50– Base de dados, com a sugestão de cada um dos interveniente no processo de transformação urbano.

5.2.1 Modelo de Comunicação

Uma comunicação cliente-servidor pode ser estabelecida através de *sockets* [Sun97] que são implementados na linguagem Java com as classes do *package java.net.* O sistema de *sockets* permite estabelecer ligações entre dois programas que estão a ser executados em dois computadores distintos, ligados por uma rede. A cada *socket* estão associados dois *streams*: um *InputStream*, usado para receber dados e um *OutputStream* usado para enviar dados.

No diagrama da Figura 51, descreve-se o que acontece do lado do cliente e do lado do servidor. De uma forma esquemática, os passos necessários a uma comunicação cliente-servidor baseada em *sockets*, são os seguintes:

(i) O servidor indica o número da porta através da qual se pode estabelecer a ligação.

(ii) O cliente tenta estabelecer uma ligação com a máquina remota indicando o `host` e o número da porta `port#`.

(iii) Quando o cliente "faz uma chamada" para esse "número", o servidor cria um *socket* para a comunicação entre os dois.

(iv) Ambos os lados, cliente e servidor, comunicam através do *`InputStream`* e do *`OutputStream`*, do *socket* aberto para essa comunicação.

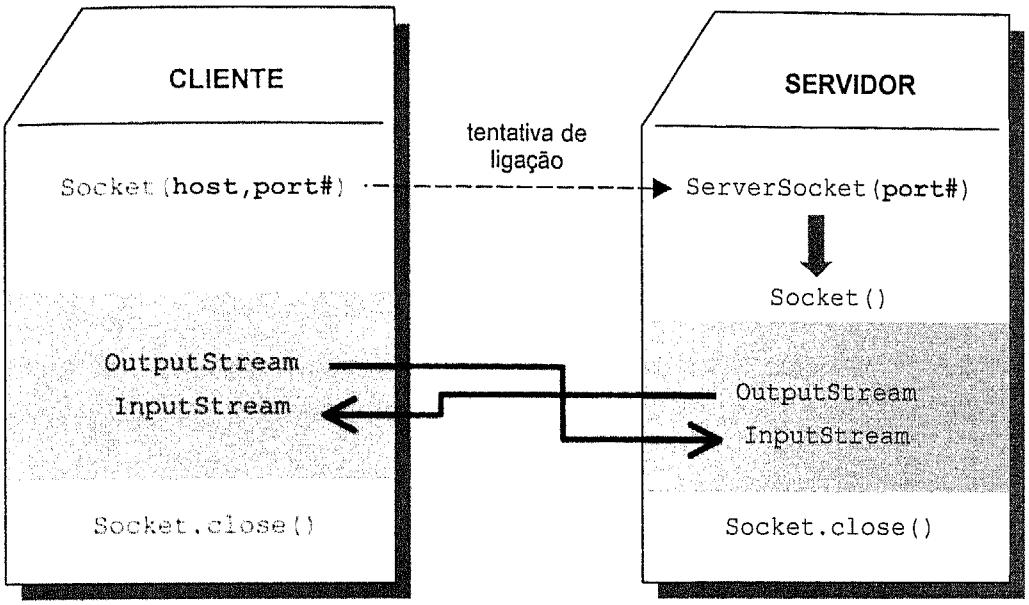


Figura 51 - Modelo de Comunicação Cliente-Servidor, através de *sockets* TCP/IP.

A partir do momento em que é estabelecida a ligação, os dois programas podem começar a trocar informação nos dois sentidos.

No caso concreto desta aplicação, o programa cliente deverá enviar dados para o programa servidor. Para tanto, basta escrever no *OutputStream* associado ao *socket* usado nessa ligação, como se pode ver na Figura 52.

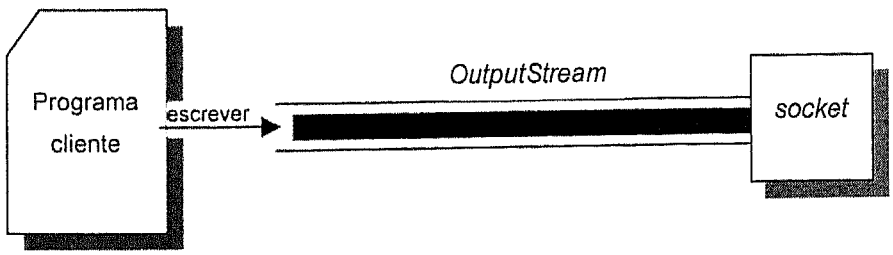


Figura 52 – Escrever dados num *OutputStream*.

Do outro lado, o programa servidor deverá receber os dados enviados. Para isso, basta ler o *InputStream* associado a esse *socket*, como se pode ver na Figura 53.

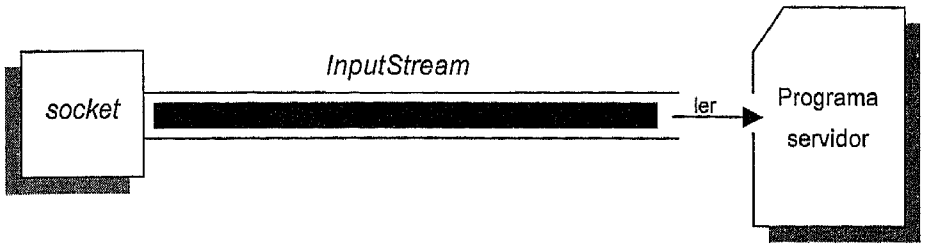


Figura 53 – Ler dados de um *InputStream*.

A informação que é enviada/recebida através de *streams* pode ser de qualquer tipo: objectos, caracteres, imagens, som. O *package java.io* possui um conjunto de classes que permitem ler e escrever dados de diferentes tipos em *streams*.

Como se viu anteriormente, para além de ser necessário enviar para o servidor os campos textuais como o nome, e-mail e observações, é igualmente necessário enviar as alterações introduzidas no modelo VRML. Trata-se, então, de enviar nós VRML, isto é, objectos. Para que seja possível um objecto ser escrito num *stream*, é necessário serializar esse objecto [Chan97] [Sun97], o que consiste em converter um objecto em bytes, de forma a poder ser escrito num *stream*. Para ler um objecto de um *stream* é necessário fazer a operação inversa, isto é, a regeneração do objecto para a sua classe original.

Os dados recebidos no servidor, referentes às alterações efectuadas no modelo e à identificação do interveniente neste processo, podem ser escritos numa base de dados relacional, por JDBC[2] [Chan97].

5.2.2 Configuração do Ambiente de Trabalho

Para a produção do protótipo foram usados o *hardware* e *software* apresentados na Tabela 6 e Tabela 7, respectivamente.

[2] JDBC – Java Database Connectivity é um API desenvolvido pela Sun que possibilita a um programa Java aceder a um "servidor" de base de dados – DBMS (DataBase Management System).

HARDWARE	DESCRIÇÃO
PC *Micron*	• Dois processadores *Pentium* II a 300MHz • 128 MB SDRAM
Placa gráfica - *ELSA Gloria*	• 16 MB VRAM + 24MB EDO-DRAM • aceleração gráfica por *hardware* - GLINT
Máquina fotográfica digital – *EPSON* 600	• 1024x768 *pixels*

Tabela 6 – *Hardware* utilizado na produção do protótipo

SOFTWARE	DESCRIÇÃO
Browser HTML – IE 4.0 da *Microsoft.*	Para visualizar páginas HTML.
Plug'in VRML – *CosmoPlayer* 2.1 da *Silicon Graphics.*	Para visualizar mundos VRML.
Modelador de VRML – *Cosmo Worlds* 2.0 da *Silicon Graphics.*	Para criar o modelo VRML.
Tratamento de Imagem - *Photoshop* 4.0 da *Adobe.*	Para tratamento das texturas.
Ambiente JAVA – *Visual Café* da *Symantec.*	Para adicionar programação externa ao modelo VRML.

Tabela 7 – *Software* utilizado na produção do protótipo.

5.3 Criação do Modelo Utilizado

Como foi referido anteriormente, este protótipo implica a criação de um ambiente virtual de um espaço de grande importância para a cidade da Guarda, a "Praça Velha". A Figura 54 apresenta fotografias da zona em questão.

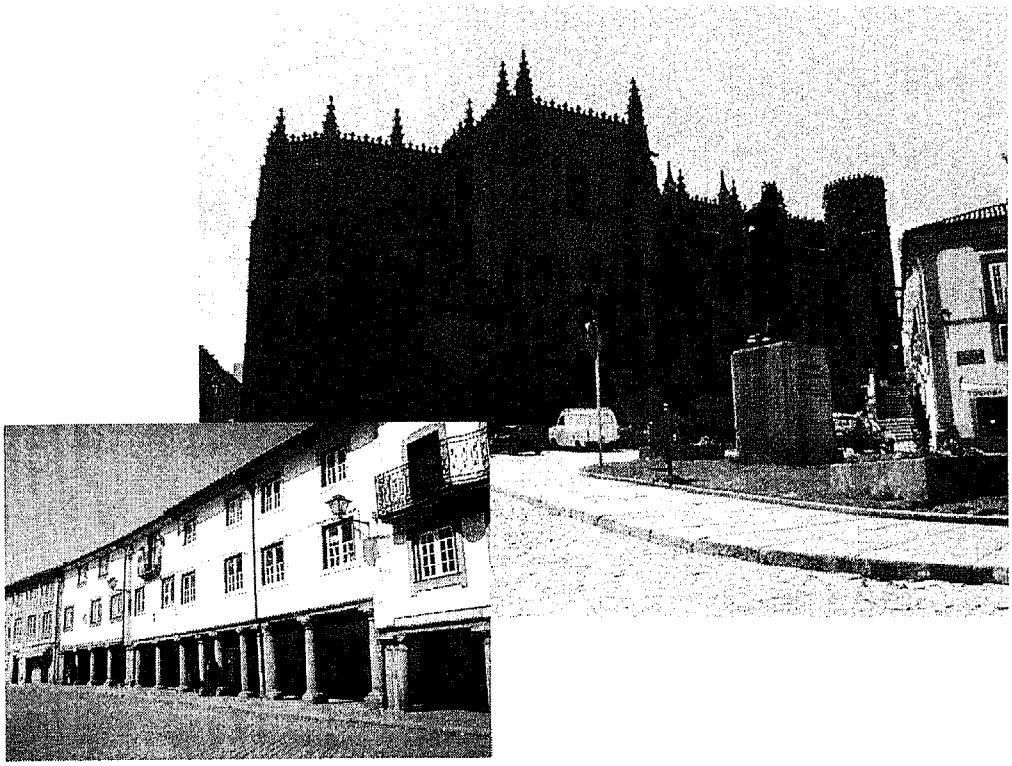


Figura 54 - Fotografias do centro histórico da Guarda

Na Figura 55 é apresentada uma planta da zona que se pretende modelar.

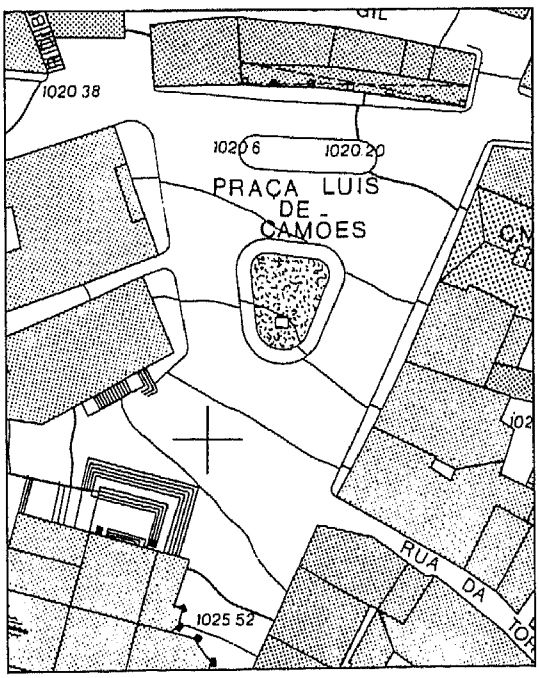


Figura 55 – Planta da zona que se pretende modelar (escala 1:1000).

Um modelo deste centro histórico foi criado, no âmbito do protótipo, de acordo com a teoria referida no capítulo 4 – Criação de um Mundo VRML. Além disso, foi necessário ter em consideração vários outros aspectos que se descrevem seguidamente.

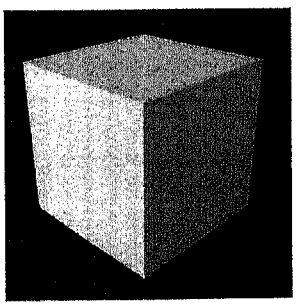
Uma questão importante que teve de ser tida em atenção na modelação foi a determinação de elementos que, posteriormente, durante o processo de transformação do centro histórico, poderiam sofrer alterações. De acordo com a especificação feita na secção 5.1, os elementos que interessa aqui reter são as fachadas, as janelas, as portas e a pavimentação.

Para que os elementos assim definidos possam ser alterados terão de ser modelados como objectos independentes, isto é, se o protótipo prevê a possibilidade de ser alterado o tipo de uma janela ou a cor de uma fachada, então essa janela ou essa fachada deverá ser um objecto independente.

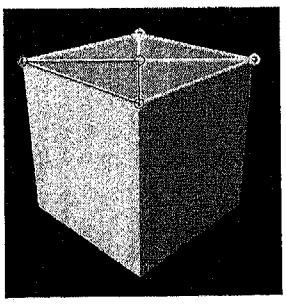
No exemplo concreto de uma casa, os elementos exteriores que poderão ser posteriormente alterados são aqueles que causam maior impacto visual, ou seja, as fachadas, as janelas, as portas e o telhado, o que, por esse motivo, devem ser modelados como objectos independentes.

Na Figura 56, é apresentado um exemplo básico do processo seguido na modelação de uma casa, atendendo a esses pressupostos.

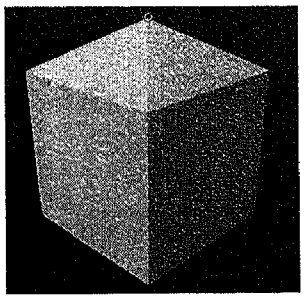
(i) Criação de um cubo (nó Box)



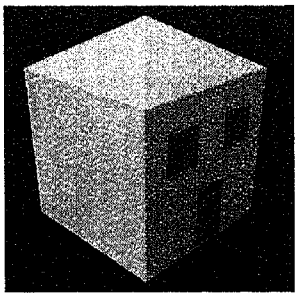
(ii) Divisão da face superior – SPLIT



(iii) Extrusão do vértice central



(iv) Criação de 3 faces adicionais



(v) Aplicação de texturas

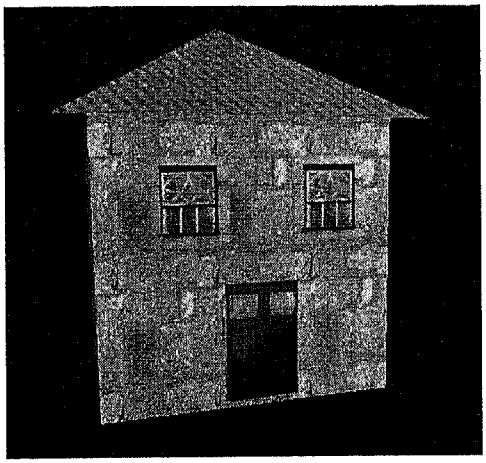
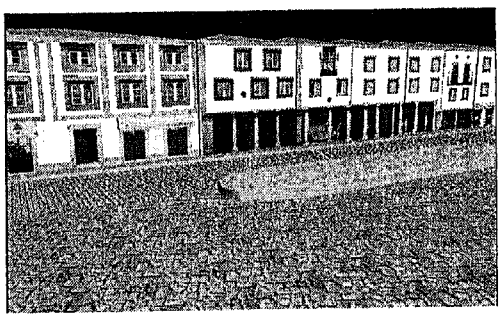


Figura 56 – Processo de criação de uma casa. (i) construção das paredes, (ii) e (iii) construção do telhado, (iv) construção de duas janelas e uma porta, (v) aplicação das respectivas texturas.

Foram modelados cinco objectos independentes que poderão, posteriormente, ser manipulados individualmente. Todos os objectos que fazem parte do modelo foram modelados segundo os mesmos critérios.

Na Figura 57 são apresentadas algumas imagens do modelo virtual, já no estado final.



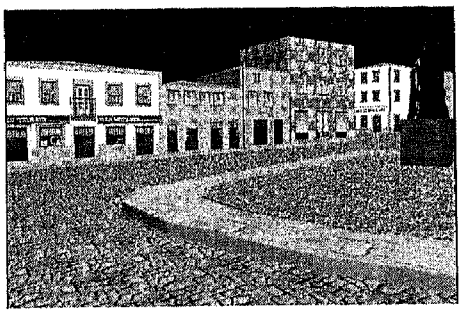


Figura 57 – Imagens do modelo VRML, construídas no *CosmoWorlds* da *Silicon Graphics*.

O protótipo desenvolvido, ao tentar simular um ambiente real, não pode perder as características mínimas de velocidade necessárias à desejável sensação de imersividade.

A natural complexidade do modelo original tem portanto de ser simplificada, desde que sem grandes perdas de realismo, por recurso à utilização das técnicas descritas na secção 4.2 – Optimização de um mundo VRML, nomeadamente a imposição de texturas.

No caso de objectos que apresentam superfícies complexas, como é o caso da catedral integrada no modelo, é essencial reduzir a sua geometria, tendo-se recorrido para o efeito à imposição de texturas com transparências sobre objectos de geometria simples. Desta forma, o número de polígonos é reduzido, na tentativa de manter o tempo de *download* e a velocidade de *rendering* em níveis razoáveis.

As texturas revelam-se assim de grande importância, pois é com elas que os elementos alteráveis, referidos anteriormente, serão modificados. Além disso, são responsáveis pelo aspecto realista que esses ou outros elementos do modelo devem apresentar (calçadas, jardins, fachadas, etc.).

As texturas utilizadas no modelo do protótipo foram obtidas a partir de fotografias do local, uma vez que apresentam melhor qualidade do que as texturas baseadas em vídeo [Bernardes96]. Estas fotografias foram captadas durante várias semanas e a diferentes

horas do dia e, por esse motivo, tiveram de ser manipuladas, de forma a minimizar as diferenças de luminosidade existentes entre elas. Foi usado o *Photoshop* da *Adobe* para fazer todo o tratamento de imagem necessário.

O formato PNG foi escolhido para armazenar as texturas usadas no modelo, uma vez que este formato gráfico permite tamanhos reduzidos, uma boa qualidade cromática (48 *bits* por *pixel*) e, sobretudo, um grande nível de transparência (8 *bits*, o que permite suportar gradações suaves de transparência). A maior parte destas imagens possui uma resolução de 128x128 *pixels* e as que são usadas para cobrir grandes áreas e precisam de ser repetidas (*tile*), têm uma resolução de 64x64 *pixels,* que, como foi referido no capítulo 4, é uma óptima resolução para este tipo de situações. Esta técnica mosaísta, foi largamente utilizada no modelo, com o objectivo de reduzir o mais possível o tamanho das texturas.

Na Figura 58, podem ver-se exemplos de objectos que utilizam texturas pequenas para cobrir grandes áreas, como calçadas, estradas e passeios.

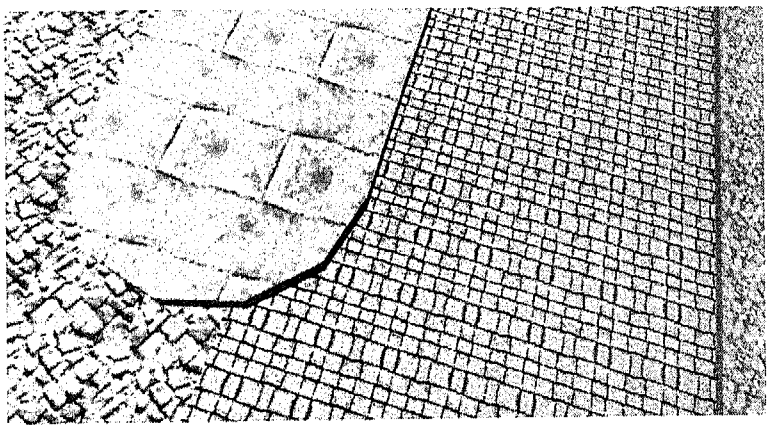


Figura 58 – *Tiling* de texturas.

A técnica usada de imposição de texturas não dá uma sensação profunda de foto-realismo. No entanto, com a actual tecnologia, um mundo com um aspecto mais próximo do real seria necessariamente um mundo pouco imersivo e lento. Na construção de um mundo virtual é essencial estabelecer um compromisso entre os conteúdos e a eficiência da cena.

5.4 Interface do Protótipo

O interface do protótipo (Figura 59) é composto por duas componentes:

- o *browser* VRML, que permite uma navegação 3D no modelo criado;
- o *applet* JAVA, que permite a manipulação dos objectos do modelo, sob o ponto de vista urbanístico;

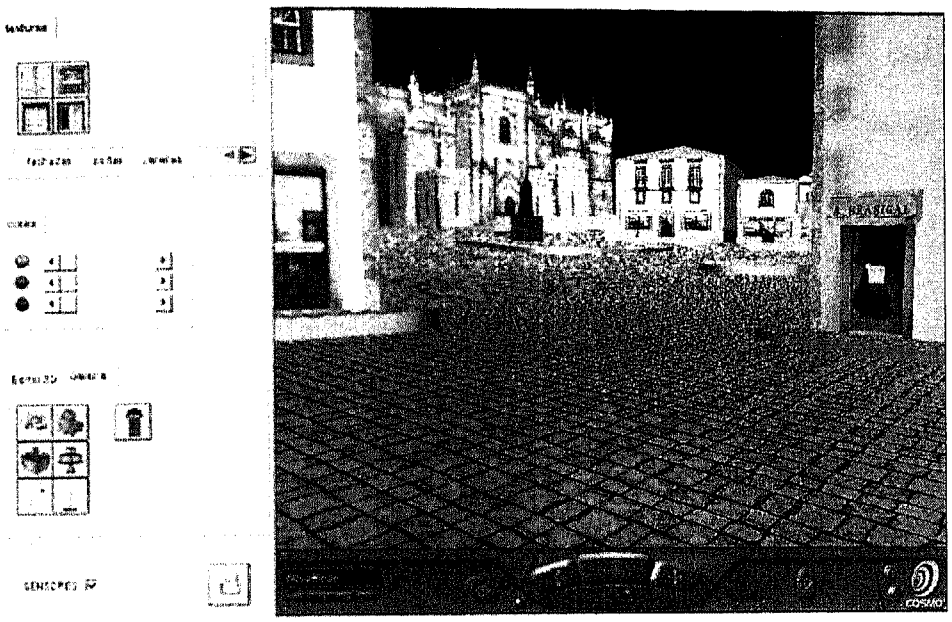


Figura 59 – Interface do protótipo

O *applet* JAVA, simples e intuitivo, permite ao utilizador a manipulação do mundo VRML, sob o ponto de vista urbanístico. Interessa, aqui, disponibilizar ferramentas que conduzam o utilizador a comunicar as várias intervenções urbanas que pretende realizar no modelo virtual e, desta forma, avaliar o seu impacto urbanístico.

O *browser* VRML permite uma navegação 3D pelo modelo virtual e a selecção dos objectos que vão ser alvo de transformação.

5.4.1 Organização do Interface

Um dos aspectos a ter em conta no desenvolvimento de um interface gráfico é a simplicidade e a coerência. Um interface deve ser capaz de possibilitar ao utilizador interagir com uma aplicação de uma forma natural e espontânea. Além disso, os

elementos que constituem um interface devem ser reduzidos ao essencial, isto é, só os elementos absolutamente necessários devem dele fazer parte. De acordo com [Hix93], um interface simples é mais facilmente apreendido e assimilado pelo utilizador, uma vez que contém menos informação visual.

Os elementos que constituem um interface devem também ser uniformizados, de forma a simplificar e estruturar a interacção do utilizador. Isto significa que deve ser mantida uma determinada coerência entre os elementos que compõe o interface, nomeadamente no seu tamanho, forma, cor, textura, orientação, alinhamento e espaçamento. Sempre que possível, os elementos de um interface devem seguir determinados padrões. Este processo facilita a percepção humana porque repete os elementos segundo um determinado ritmo [Mullet95].

De acordo com [Hix93] e [Mullet95] e com os princípios enumerados na secção 5.1, acerca dos elementos caracterizadores de um espaço exterior, o interface foi estruturado de forma a permitir a intervenção do utilizador a três níveis: **aparência**, **geometria** e **sistema**, como se pode ver na Figura 60.

As intervenções ao nível da aparência prendem-se com a possibilidade de alterar o aspecto visual dos objectos, isto é, a sua cor e textura. As operações fundamentais são:

- Alterar a cor da fachada de uma casa;
- Alterar o tipo de uma fachada, janela, porta ou pavimento, com base em texturas;

Ao nível da geometria, as intervenções têm a ver com a possibilidade de adicionar à cena novos objectos tridimensionais, bem como aplicar transformações geométricas a determinados objectos do modelo. Neste caso, as operações fundamentais são:

- Inserção de elementos de equipamento predefinidos, existentes numa galeria de objectos 3D;
- Criação de novos objectos com base em primitivas 3D;
- Aplicação de transformações geométricas 3D a determinados objectos do modelo;

Por último, as intervenções ao nível do sistema têm a ver com questões de navegação no modelo virtual, e com a comunicação com a Autarquia que fornece o serviço.

Assim, o interface desenvolvido permite, através de simples botões e barras de deslocamento, executar uma série de intervenções que tornam possível transformar o

modelo VRML, de uma forma rápida e simples, e avaliar, desta forma, o impacto urbanístico causado.

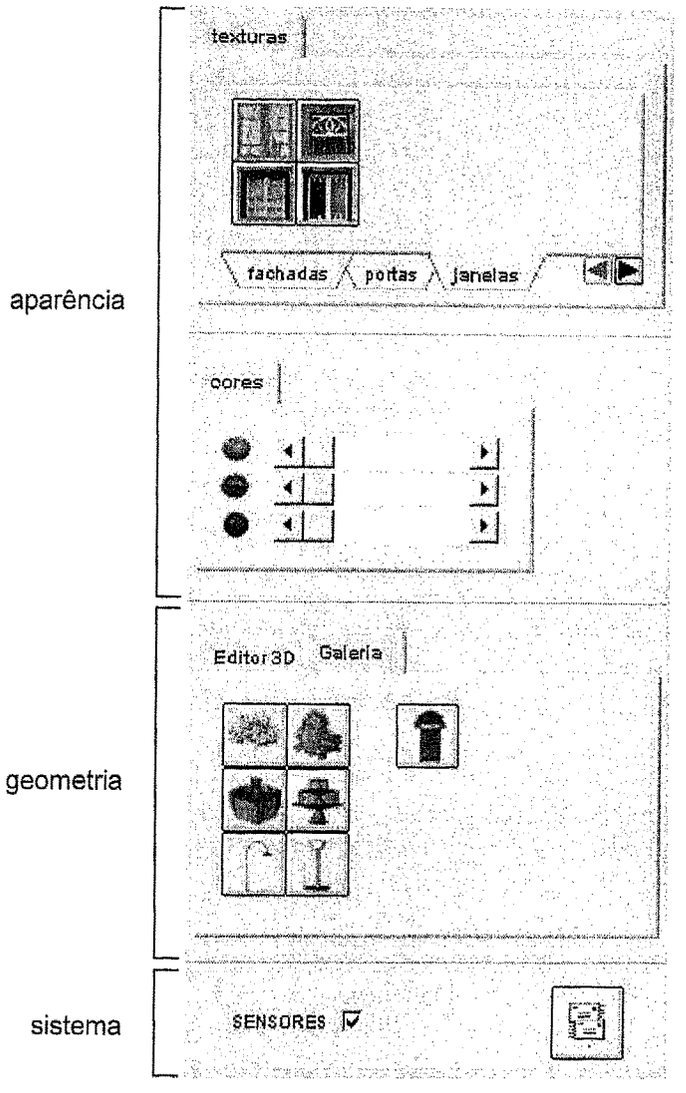
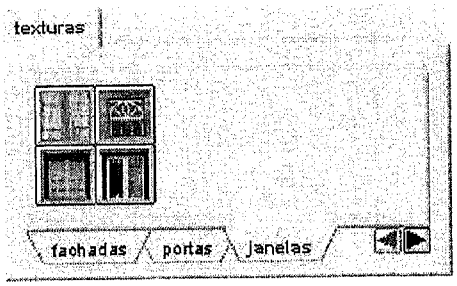


Figura 60 – Organização do Interface.

5.4.2 Descrição do Interface

Apresentam-se de seguida todos os elementos que fazem parte do interface criado em Java.

PALETES DE TEXTURAS



Foram criadas várias *paletes*, com texturas para:

- tipos de fachada;
- tipos de porta;
- tipos de janela;
- tipos de pavimento (calçada romana, paralelo, relva,...)

Os elementos destas *paletes* podem ser atribuídos a objectos do mesmo tipo presentes na cena. Desta forma, o utilizador pode experimentar diferentes tipos de fachada, janelas, portas ou pavimentos e avaliar, assim, o impacto causado.

Foram introduzidas restrições, de modo a não possibilitar, por exemplo, a introdução da textura de uma janela no objecto porta (como se pode ver na Figura 61), ou a textura de uma fachada no objecto pavimento.

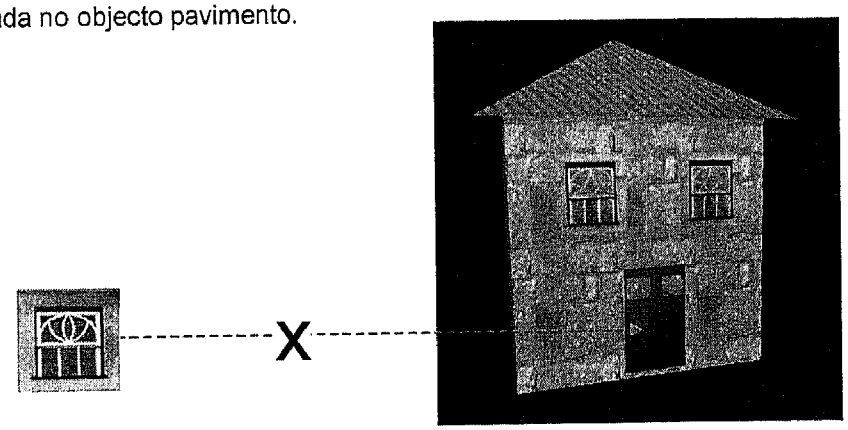
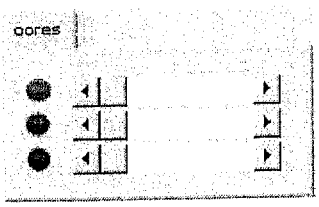


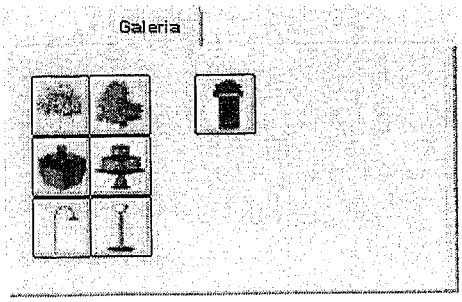
Figura 61 – Restrição na introdução de janelas.

CONTROLE DE COR



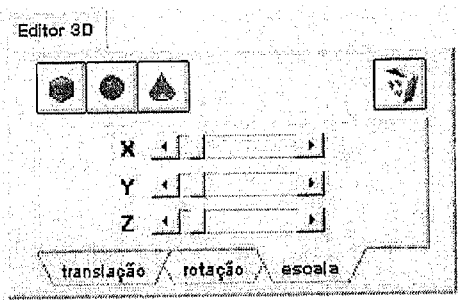
Através de 3 barras de deslocamento RGB, o utilizador pode ajustar a cor da fachada de uma casa, visualizando de imediato o efeito causado.

GALERIA



Através da Galeria, o utilizador tem a possibilidade de juntar, ao modelo existente, novos elementos de equipamento, previamente modelados como objectos VRML (árvores, fontes, postes de iluminação, etc). Estes objectos são aqueles que irão "mobilar" a zona exterior e, por isso, deverão estar dentro do contexto visual da zona que se pretende remodelar. Como já foi referido, cabe aos técnicos decidir quais os objectos que deverão estar disponíveis nesta galeria.

EDITOR 3D



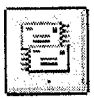
Este modelador permite criar novos objectos com base em primitivas 3D. Após a sua criação, os objectos podem ser pintados, com recurso às barras de deslocamento RGB e podem receber texturas existentes nas *paletes*.

Nesta fase de protótipo, este modelador permitirá ao utilizador criar, por exemplo, um cubo e aplicar-lhe uma textura de pedra. De seguida, poderá criar outro cubo e aplicar-lhe uma textura com um tipo de janela e por aí adiante, seguindo o processo de criação de uma casa, referido na secção 5.3.

As transformações geométricas 3D (translação, rotação e factor de escala) foram implementadas permitindo mover, rodar ou escalar um objecto, nos três eixos X, Y e Z. Estas transformações geométricas podem ser aplicadas, não só aos novos objectos, criados através do modelador ou inseridos através da galeria, mas também a determinados objectos já existentes no mundo original, como por exemplo, a estátua ou o jardim. Para evitar que um utilizador coloque – suponhamos - uma estátua em cima de um telhado ou que lhe aplique uma rotação que não faz sentido, foram introduzidas algumas restrições no tipo de operação que o utilizador pode efectuar. Com restrições semelhantes também é possível remover determinados objectos do modelo.

Por vezes torna-se necessário desligar os sensores dos objectos por forma a facilitar a navegação do utilizador no mundo virtual. Como será explicado, o mundo original possui vários sensores ligados aos diferentes objectos que se pretende manipular. Assim, quando o utilizador passa com o cursor por cima de um desses objectos, o mesmo assume a forma de interacção, o que, por vezes, torna a navegação complicada. A possibilidade do utilizador desligar e ligar os sensores existentes na cena facilita o processo de navegação.

COMUNICAÇÃO COM A AUTARQUIA



Após ter efectuado todas as alterações pretendidas, o utilizador deve enviar a sua sugestão para uma base de dados, existente num servidor, na Autarquia que fornece este serviço. Assim, quando o utilizador toca neste botão, é aberto um formulário de dados (Figura 62), onde terá de indicar o seu NOME, E-MAIL e OBSERVAÇÕES que eventualmente queira referir. Estes dados acompanham o modelo VRML alterado, de acordo com a arquitectura definida na secção 5.2.

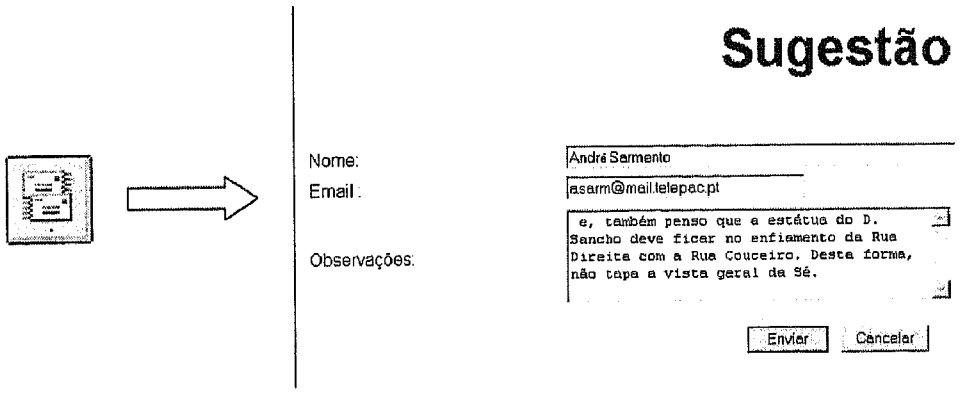


Figura 62 - Formulário de dados, com a identificação do utilizador que participou no processo de transformação urbano.

5.5 Comunicação entre os Componentes da Aplicação

Como referido na secção anterior, os componentes do interface gráfico desta aplicação são o *browser* VRML e o *applet* Java. Enquanto o *browser* permite ao utilizador navegar num ambiente tridimensional, o *applet* oferece um conjunto de ferramentas que possibilitam a manipulação dos objectos desse ambiente 3D.

Segundo a secção 3.2.2, a comunicação de um *applet* com um mundo VRML realiza-se através do *External Authoring Interface* – EAI. Este API possui um conjunto de classes e métodos que dão a possibilidade de um *applet* aceder às funcionalidades de um *browser* VRML, e também de comunicar com os objectos da cena.

Para um *applet* comunicar com os objectos de uma cena VRML necessita obter "ligações" para:
(i) o *browser* VRML;
(ii) os nós que pretende manipular;
(iii) os respectivos *eventIns* e *eventOuts* desses nós.

O processo de estabelecer estas "ligações" encontra-se descrito na secção 3.2.2.

A partir do momento em que nós, *eventIns* e *eventOuts* estão acessíveis no *applet*, torna-se possível trocar informação entre o *applet* e a cena VRML e, consequentemente, iniciar o processo de manipulação dos objectos. Este processo baseia-se, fundamentalmente, em receber eventos da cena e enviar eventos para a cena VRML.

Quando o utilizador pretende manipular um objecto, isto é, alterar as suas propriedades (a cor de uma fachada, o tipo de uma janela, a posição de uma estátua, etc.) deve "ligar" um sensor a esse objecto. Assim, quando o utilizador selecciona o objecto, o sensor que lhe está associado gera imediatamente um evento na cena. A forma de o *applet* ser notificado da ocorrência desse evento e de reconhecer qual o objecto que foi seleccionado pelo utilizador foi já descrita na secção 3.2.2.2. É com este objecto corrente que o *applet* vai trocar informação. A secção seguinte descreve os dois tipos de acesso permitidos a um objecto corrente.

5.5.1 Acesso às Propriedades de um Objecto Corrente

Um *applet* pode aceder às propriedades de um objecto corrente para duas situações:

- **Leitura**, de forma a que o *applet* seja notificado do estado actual do objecto corrente e, por consequência actualize as propriedades dos componentes gráficos do *applet*, como por exemplo, as barras de deslocamento.
- **Escrita**, de forma ao *applet* alterar as propriedades do objecto corrente, como por exemplo, o tipo de uma porta, a cor de uma fachada, a posição de uma estátua ou o tamanho de um jardim.

Para o *applet* ser notificado de, por exemplo, a cor da fachada de uma casa, o processo é ler a informação de cor, associada ao *eventOut diffuseColor* do nó *Material,* dessa fachada (Figura 63). Neste caso, quando o utilizador selecciona a fachada, as barras de deslocamento existentes no *applet* são imediatamente actualizadas com a informação de cor dessa fachada.

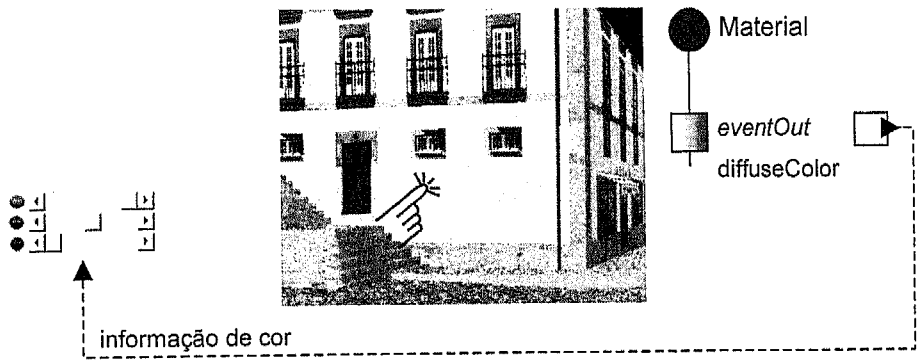


Figura 63 – Leitura das propriedades da fachada de uma casa.

Se o utilizador pretender, agora, alterar a cor dessa mesma fachada, o processo inverte-se, enviando eventos, com a informação de cor respectiva, para o *eventIn diffuseColor* do nó *Material* que está associado a essa fachada. A Figura 64 descreve este processo.

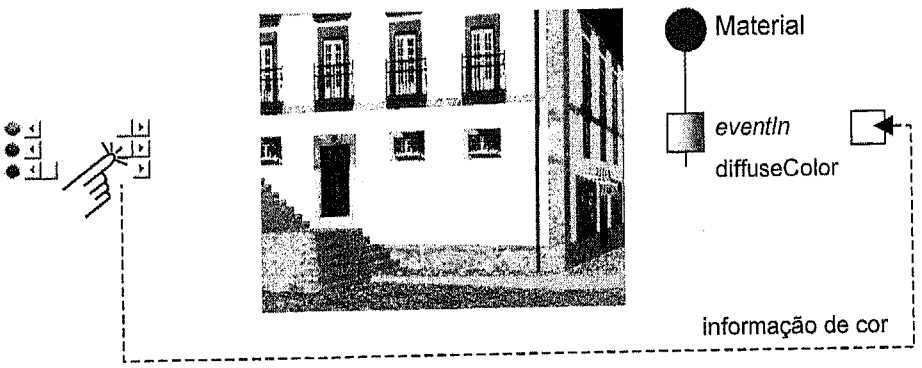


Figura 64 – Enviar eventos para o objecto corrente.

De notar que o campo *diffuseColor* é do tipo *exposedField* e, por esse motivo, tem ambas as declarações de *eventIn* e *eventOut*.

Os exemplos apresentados na Figura 63 e Figura 64 dizem respeito à cor. No entanto, qualquer tipo de informação (textura, posição, rotação, etc.) que seja necessário trocar entre o *applet* e o mundo VRML segue o mesmo processo de troca de eventos.

5.5.2 Criação de Novos Objectos com base em Primitivas 3D

Através do editor 3D desenvolvido neste protótipo, é possível criar objectos simples com base em primitivas 3D (cubos, esferas e cones). De notar que o que se pretende com esta facilidade é permitir a criação dinâmica de objectos VRML. O método `createVrmlFromString(String vrmlSyntax)` da classe *Browser* serve precisamente para este tipo de situações.

Este método informa o *browser* no sentido de converter uma *string* no objecto VRML correspondente ao código escrito como argumento. Sendo assim, o nó *Box* é criado da seguinte maneira: `createVrmlFromString("Box {}")`

Com este método é possível criar, a partir de um *applet*, qualquer tipo de nó VRML: nós de geometria, aparência, agrupamento, sensores, etc.

Para que os novos objectos adicionados à cena possam sofrer uma série de transformações, geométricas e de aparência, é necessário que tenham associado um sensor, como referido anteriormente. Assim, o novo objecto e o respectivo sensor são agrupados a um nó receptor[3], criando o seu próprio grupo e respectivo sistema de coordenadas. Além disso, para que se tornem visíveis na cena, é necessário agrupá-los, de novo, a um nó já existente na cena VRML.

No exemplo da Figura 65, esse nó chama-se *ROOT* e não deve conter qualquer geometria, dado que a sua função é apenas tornar visíveis todos os objectos que são adicionados à cena existente. Este é um nó de agrupamento e, por conseguinte, recebe os novos nós como seus *children*. Assim, os nós RECEPTOR1 e RECEPTOR2 são *children* do ROOT. Os nós CUBO e sensorCUBO são *children* de RECEPTOR1 e, finalmente, os nós ESFERA e sensorESFERA são *children* do RECEPTOR2.

[3] Receptor, no sentido de ser um nó que serve para receber outros nós.

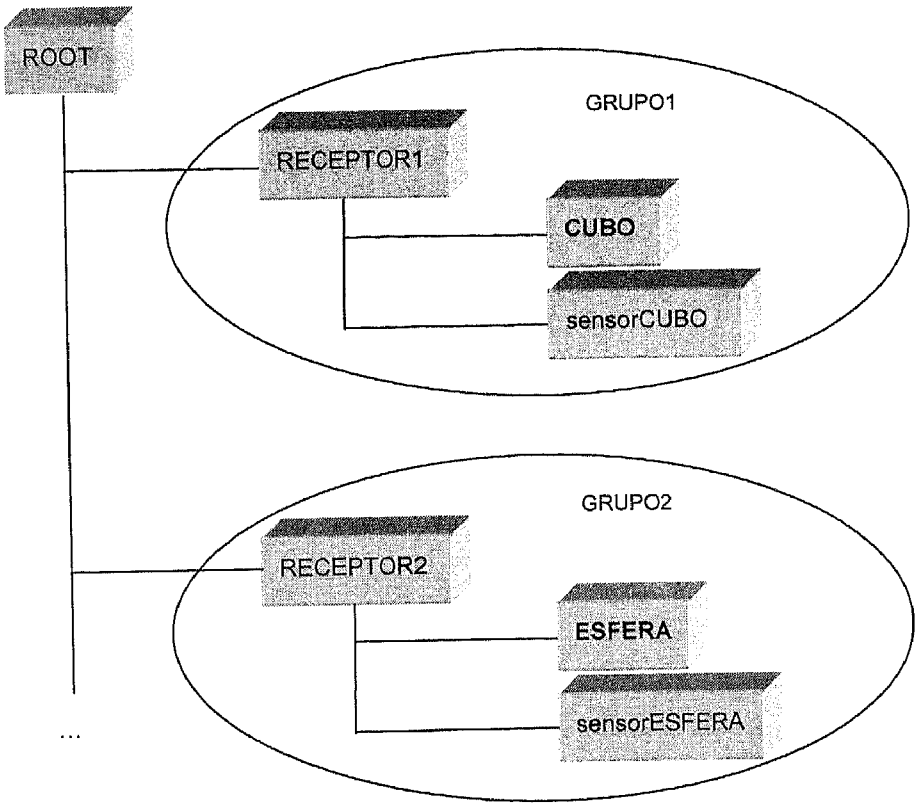


Figura 65– Estrutura hierárquica de dois novos objectos e respectivos sensores adicionados a uma cena VRML existente.

Neste exemplo, foram adicionados dois novos objectos à cena, um CUBO e uma ESFERA. De notar que cada um destes objectos possui um sensor no seu grupo, o que os torna sensíveis ao toque do utilizador e, por conseguinte, passíveis de ser manipulados.

Quando o utilizador manipular um destes novos objectos, são enviados eventos para os *eventIns* respectivos dos nós receptores. Considere-se, por exemplo, que o utilizador pretende rodar o CUBO. Serão enviados eventos, com informação da rotação correspondente, para o *eventIn rotation* do nó RECEPTOR1. Se, por outro lado, pretender mover a ESFERA, serão enviados eventos, com a translação correspondente, para o *eventIn translation* do nó RECEPTOR2. Isto significa que a edição de um novo objecto é levada a efeito ao nível do nó RECEPTOR. Desta forma, todos os novos objectos são absolutamente independentes, uma vez que possuem o seu próprio grupo e, por conseguinte, os seus próprios sistemas de coordenadas. O mesmo não estava garantido se a edição fosse feita ao nível do nó ROOT. Neste caso, a edição de um novo objecto

implicaria a edição de todos os outros, uma vez que todos os novos grupos são *children* da ROOT e, portanto, partilham o mesmo sistema de coordenadas.

Um problema adicional relacionado com a inserção de novos objectos, é a escolha da posição da cena em que a inserção se efectua. Assim, deve ser garantido que um novo objecto seja inserido numa posição visível ao utilizador, seja qual for a sua posição na cena, o que sugere uma posição frontal à posição do utilizador (Figura 66).

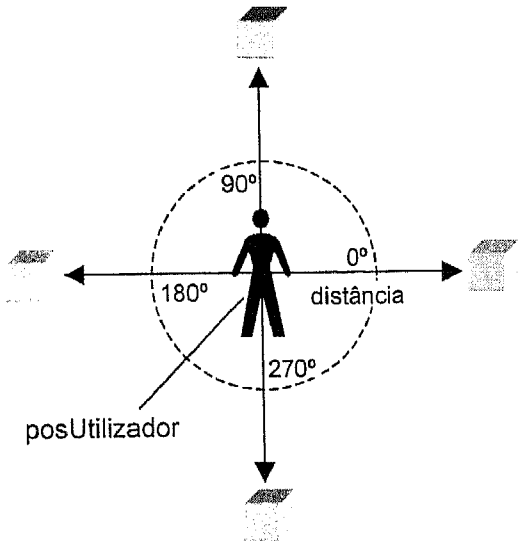


Figura 66 – Posição de inserção frontal ao utilizador.

A determinação dessa posição exige o conhecimento da posição do utilizador. Para o efeito, foi "ligado" um sensor de proximidade que "cobre" toda a envolvente do modelo. Desta forma, sempre que o utilizador se move dentro desta região, o sensor gera eventos com as coordenadas correntes do utilizador, nomeadamente a sua posição e orientação.

Conhecendo a posição (`posUtilizador`) e orientação (`ang`) correntes do utilizador, a posição de inserção do novo objecto é calculado da seguinte forma:

```
PosInserçãoX = posUtilizadorX + distância * cos(ang);
PosInserçãoY = posUtilizadorY + distância * sin(ang);
```

A posição de inserção calculada anteriormente é então enviada para o *eventIn translation*, dos nós de agrupamento receptores, dos novos objectos.

5.5.3 Inserção de Objectos da Galeria

Os objectos da Galeria são modelos VRML existentes em ficheiros externos que precisam ser adicionados dinamicamente à cena existente. O método *createVrmlFromUrl( )* permite adicionar a uma cena VRML existente um objecto VRML externo.

Este método apresenta a seguinte sintaxe:

```
public void createVrmlFromURL(String[] url, Node receptor, String evento)
```

O primeiro argumento indica o `url` do ficheiro VRML externo. O segundo argumento indica o nó de agrupamento que vai receber o modelo externo, como seu *children*. No terceiro argumento é definido um *eventIn* deste nó receptor.

No exemplo seguinte, o modelo `galeriaVRML/fonte.wrl` é adicionado ao nó de agrupamento `receptor` através do *eventIn* `addChildren`:

```
createVrmlFromURL("galeriaVRML/fonte.wrl",receptor,"addChildren")
```

A edição e a inserção destes objectos na cena segue uma filosofia semelhante à que foi descrita na secção 5.5.2 para os novos objectos com base em primitivas 3D.

5.6 Síntese

No protótipo realizado, como exemplo de aplicação desta dissertação, foi criado um modelo virtual de um espaço de grande importância para a cidade da Guarda – a "Praça Velha".

O objectivo deste protótipo é proporcionar ao utilizador a comunicação de eventuais sugestões ou propostas de intervenção que pretende realizar neste espaço urbano. Para o efeito, deve permitir a manipulação da aparência dos elementos que compõe este espaço (alterar e pintar fachadas, alterar tipos de janelas, portas e pavimentos) e a integração de novos elementos, como sejam, novos objectos 3D criados por si próprio através de um simples modelador, e também elementos de equipamento predefinidos, existentes numa galeria de objectos 3D disponibilizada pela própria Autarquia. Complementarmente, deve permitir manipular a geometria não só destes novos objectos

mas também de alguns previamente existentes no modelo original. Não quer isto dizer que o utilizador tem a possibilidade de intervir livremente sobre todo e qualquer elemento do espaço em questão. Para salvaguardar esta situação, foram desenvolvidos mecanismos para restringir, de algum modo, as operações que o utilizador possa efectuar na sua intervenção.

Para permitir que os intervenientes neste processo de transformação urbano possam fazer chegar à Autarquia as suas sugestões e propostas de intervenção precisa ser estabelecida uma comunicação cliente-servidor, baseada em *sockets*.

O interface construído para facultar ao utilizador deste sistema interagir com a aplicação é constituído por duas componentes: o *browser* VRML, que permite uma navegação 3D no modelo criado, e o *applet* Java, que permite manipular os objectos desse modelo, sob o ponto de vista urbanístico. Este interface foi estruturado de forma a tornar a intervenção do utilizador simples e intuitiva, fornecendo-lhe um conjunto de ferramentas (botões, barras de deslocamento, painéis) que se repetem segundo um determinado ritmo, com o que facilita a percepção do utilizador.

Como foi referido na secção 3.2.2, para um *applet* Java comunicar com um mundo VRML é necessário um interface de programação entre os dois – EAI. A partir do momento em que nós, *eventIns* e *eventOuts* estão acessíveis no *applet*, torna-se possível trocar informação entre este e a cena VRML e, consequentemente, iniciar o processo de manipulação dos objectos. Este processo baseia-se, fundamentalmente, em receber eventos da cena e enviar eventos para a cena VRML.

Para adicionar novos objectos à cena, o EAI possui dois métodos da classe Browser, *createVrmlFromString* e *createVrmlFromURL*, que permitem adicionar dinamicamente, isto é, durante a execução da aplicação, novos objectos à cena 3D. No caso dos objectos criados com base em primitivas 3D foi usado o primeiro método, que indica ao *browser* para converter uma *string* num objecto VRML. O segundo método indica ao *browser* para inserir no mundo actual objectos VRML, armazenados em ficheiros externos, como é o caso dos elementos predefinidos existentes na galeria 3D.

capítulo 6.

Conclusões da Dissertação

O problema apresentado nesta dissertação foi dirigido para as áreas da Arquitectura e do Urbanismo, onde a modelação por computador é hoje largamente utilizada como uma ferramenta para compreender a relação espacial entre os objectos. Pretendeu-se aferir em que medida as novas tecnologias (Internet, Multimédia, VRML) permitiriam o desenvolvimento de novas estratégias em planeamento e projecto urbano.

O VRML é um dos mais recentes desenvolvimentos da Internet para descrever ambientes virtuais tridimensionais e interactivos e tem tido uma expansão e aceitação incontestáveis. Esta tecnologia pode proporcionar oportunidades muito interessantes em diversas áreas, nomeadamente na Arquitectura e no Urbanismo.

A vantagem da linguagem VRML é que permite que os ambientes virtuais possam ser visualizados por qualquer pessoa com acesso à Internet, independentemente da plataforma que estiver a utilizar (windows 95/NT, unix, linux, etc.). As características multimédia adicionais que possui e que permitem integrar texto, som e vídeo a ambientes 3D, assim como a capacidade de recorrer a linguagens de programação externa para adicionar comportamentos específicos aos objectos desses ambientes, vieram tornar o VRML ainda mais poderoso. Java é a linguagem de programação externa adequada, uma vez que tem a vantagem de ser, também ela, independente de plataformas concretas de *hardware* e *software*.

Dadas as limitações impostas pela actual tecnologia, nomeadamente a largura de banda das redes de telecomunicações e a velocidade de processamento gráfico de um PC, um mundo virtual precisa de ser cuidadosamente optimizado para captar o interesse do utilizador, ainda mais quando se trata de simular ambientes que implicam um nível de realismo apreciável, como é o caso do modelo virtual desenvolvido no protótipo a que se refere esta dissertação. Na construção de modelos tridimensionais para a Internet, a optimização é uma questão essencial a ter em conta.

No âmbito dos trabalhos conducentes a esta dissertação foi elaborado um protótipo, recorrendo às novas tecnologias de informação, como suporte para validar, ou não, a tese que se propunha demonstrar. O protótipo em causa foi aplicado a um espaço de inestimável valor e importância para a cidade da Guarda – a "Praça Velha", já por diversas vezes - e ao longo da sua história - palco de intervenções que nem sempre se revelaram as mais adequadas e pacíficas junto da opinião pública em geral.

Este protótipo foi apresentado ao Departamento Técnico de Obras e Urbanismo da Câmara Municipal da Guarda, com o intuito de o mesmo ser avaliado por alguém com responsabilidades e conhecimento na área do planeamento e gestão do projecto urbano. Com efeito, e na sequência desta apresentação, foi elaborado um parecer que se apresenta em anexo a esta dissertação e que serviu de referência para grande parte das conclusões aqui expressas.

Como conclusão principal, considera-se que da utilização das novas tecnologias de informação actualmente disponíveis, decorre todo um novo horizonte de estratégias e projectos no plano urbanístico. Os novos meios tecnológicos concorrem, de maneira eficaz e definitiva, para o enriquecimento das experiências de planeamento urbano que integram o espólio de uma Autarquia ao proporem uma forma atraente e "simpática" de captar o interesse e a motivação dos cidadãos na resolução de problemas relacionados com a transformação ou o crescimento urbanístico da sua terra, levando-os a tornarem-se um interveniente precioso, um factor influenciador vivo e actuante de um processo de tomada de decisões rápido e eficiente, ao mesmo tempo que se sentem como parte integrante de um projecto abrangente de construção da cidade em que vivem, num espírito de renovada abertura e participação que só os novos canais de comunicação podem proporcionar.

Este sistema de auscultar a opinião dos munícipes mediante recursos totalmente tecnológicos incorpora, além de outras vantagens, a mais-valia do conhecimento "na

hora" e de maneira inequívoca do sentir da comunidade para com o espaço urbano em que se move, suscitando o tratamento estatístico imediato das propostas consideradas válidas para o efeito. Em relação à forma tradicional de registar a opinião dos cidadãos, o "inquérito", estes novos processos transportam-nos já em direcção a um futuro de novas e risonhas realidades.

No caso concreto da Arquitectura, pelas suas especificidades, pelo grau de ciência e de arte que engloba, nem todas as sugestões que emanam da vontade de participação dos munícipes poderão ser consideradas, por nem todas se revelarem as mais correctas e ajustadas na hora de decidir. Por esse motivo, uma aplicação desta natureza deve naturalmente ser possuidora de mecanismos que possam restringir a intervenção dos cidadãos, de forma a balizar com rigor o que em cada caso concreto se encontra realmente submetido à sua apreciação.

No limiar de um novo século e de um novo milénio, com a emergência de novos conceitos, ditados pela vaga a que chamam Revolução Tecnológica e com o afloramento de novos comportamentos e atitudes, é necessário um reenquadramento do Homem na sociedade, com mútuas dádivas e obrigações, em que aos novos desafios seja dada uma resposta cabal e responsável, sempre na perspectiva de uma crescente qualidade de vida e dentro do primado da pessoa humana como centro de todas as actividades. Neste contexto, nunca serão demasiados os fluxos de comunicação e os mecanismos de intervenção, grupal ou pessoal, dos cidadãos na construção da sociedade.

6.1 Perspectivas de trabalho futuro

A aplicação correspondente ao protótipo desenvolvido, no âmbito das ideias enunciadas, foi estruturada de forma modular, baseada numa metodologia OO (Orientada a Objectos), pelo que muito do seu código pode ser reutilizado para o rápido desenvolvimento de outras aplicações desta natureza. As aplicações que assentam em metodologias OO tornam-se muito mais robustas e de manutenção mais simples do que as que derivam de metodologias funcionais. Em aplicações inovadoras, como é o caso do protótipo a que se refere esta dissertação, tal característica é fundamental, já que possibilita alterações de estratégia ao longo do seu desenvolvimento.

Alguns aspectos são neste momento enunciáveis como sendo importantes para futuros desenvolvimentos deste protótipo.

Um problema que se pode prever nesta aplicação é uma enorme afluência de propostas de intervenção por parte do grande público, o que irá certamente dificultar a consulta das mesmas na base de dados respectiva. Assim sendo e de modo a facilitar esta consulta, deveriam ser desenvolvidos mecanismos automáticos destinados a tipificar o tipo de intervenções dos utilizadores no espaço exterior, por exemplo, gerando automaticamente terminologia predefinida a integrar na base de dados. Desta forma, as diferentes propostas poderiam ser devidamente agrupadas pelo tipo de intervenção, com o que facilitaria aos responsáveis do serviço o acesso selectivo e eficiente de tipos muito concretos de intervenção.

De referir que este tipo de aplicações, que no caso concreto do protótipo desenvolvido visou um espectro largo de intervenientes (desde o cidadão comum até ao político), pode ser restrita à participação de apenas um grupo específico de trabalho. Neste contexto, uma solução complementar seria a criação de ambientes virtuais multiparticipados. Caberia, então, a possibilidade de grupos de profissionais trabalharem cooperativamente no mesmo projecto.

Outros aspectos que também se podem considerar importantes no desenvolvimento de uma aplicação final seriam, a identificação de um maior número de elementos arquitectónicos passíveis de ser manipulados pelo utilizador, bem como a disponibilização de uma palete de cores predefinida e seleccionada de acordo com os padrões exigidos pelos técnicos da Autarquia.

A concepção de ambientes tridimensionais é um trabalho criativo e vital para comunicar ideias. Os conceitos associados à criação de modelos de realidade virtual na Internet podem ser de grande utilidade em diversas áreas de aplicação. Fica assim aberta a perspectiva de um "leque" variado de opções a aplicar em projectos futuros, em qualquer área cujo objectivo principal seja comunicar ideias de uma forma eficaz, sem a distância e a plataforma como obstáculo.

Referências

[Adler99] Mark Adler, Thomas Boutell, et al, PNG (Portable Network Graphics) Specification, Version 1.1, http://www.cdrom.com/pub/png/spec/, 15 February 1999.

[Ames97] Andrea L. Ames, David R. Nadeau, John L. Moreland, "VRML 2.0 Source book - Second Edition", John Wiley & Sons, Inc., 1997.

[Ballreich97] Cindy Reed-Ballreich, "Chapter 10 - Texture Map Creation", Late Night VRML 2.0 with Java, Ziff-Davis Press, an Macmillan Computer Publishing, 1997.

[Bell95] Gavin Bell, Anthony Parisi, Mark Pesce, The VRML – Version 1.0 Specification, http://www.vrml.org/VRML1.0/vrml10c.html, 5-Nov-95.

[Bell96] Gavin Bell, Rikk Carcy, Chris Marrin, The VRML Specification – Version 2.0, http://www.vrml.org/technicalinfo/specifications/vrml2.0/index.htm, August 4, 1996.

[Bernardes96] Paulo Bernardes, Laurentina Soares, Sérgio Coelho, Rui Sérgio Rodrigues, António Diogo, José C. Teixeira, "Virtual Reality, Urban Planning and Historical Town Centers", COMPUTER GRAPHIK topics, Volume 8, No. 4/96, August 1996.

[Blin76] J.F. Blinn, and M.E. Newell, "Texture and Reflection In Computer Generated Images," *CACM*, 19(10), October 1976.

[Catm74] E. Catmull, A Subdivision Alghorithm for Computer Display of Curved Surfaces, Ph.D. Thesis, Report UTEC-CSc-74-133, Computer Science Department, University of Utah, Salt Lake City, UT, December 1974.

[Chan97] Mark C. Chan, Steven W. Griffith, Anthony F. Iasi, "1001 Java Programmer's Tips", Jamsa Press, 1997.

[Coelho85] António J. M. Baptista Coelho, António M. Reis Cabrita, "Os espaços exteriores de uma nova área residencial e o comportamento humano", Documento - Base 2, Estudos sobre espaços exteriores em novas áreas residenciais, Laboratório Nacional de Engenharia Civil, Lisboa 1985.

[Coelho88] António J. M. Baptista Coelho, António M. Reis Cabrita, "Exigências e Critérios para Projecto", Documento - Base 5, Estudos sobre espaços exteriores em novas áreas residenciais, Laboratório Nacional de Engenharia Civil, Lisboa 1988.

[Couch97] Justin Couch, "Part 1 - An Introduction to VRML", Late Night VRML 2.0 with Java, Ziff-Davis Press, an Macmillan Computer Publishing, 1997.

[EAI99] Interface between VRML worlds and an external environment, External Authoring Interface Working Group, http://www.vrml.org/WorkingGroups/vrml-eai, 1999.

[Foley94] James D. Foley, Andries van Dam, Steven K. Feiner, John F. Hughes, Richard L. Phillips, "Introduction to Computer Graphics", Addison-Wesley Publishing Company, Inc., 1994.

[Galleries98] Cosmo Galleries, http://cosmosoftware.com/galleries/, PLATINUM technology, inc., 1998.

[GIF90] GRAPHICS INTERCHANGE FORMAT Version 89a, http://www.w3.org/Graphics/GIF/spec-gif89a.txt, 31 July 1990.

[Goitia92] Fernando Chueca Goitia, "Breve História do Urbanismo", Editorial Presença, Lisboa, 1992.

[Gomes99] Joaquim Luís da Costa Gomes, "Parecer sobre Trabalho desenvolvido no âmbito de uma Tese de Mestrado", Departamento Técnico de Obras e Urbanismo, Câmara Municipal da Guarda, Março 1999. (em anexo).

[Hamilton92] Eric Hamilton, JPEG File Interchange Format, http://www.w3.org/Graphics/JPEG/jfif.txt, September 1, 1992.

[Hartman98] Jed Hartman, Wendy Vogt, "Cosmo Worlds 2.0 User's Guide", Silicon Graphics, Inc. 1998.

[Hix93] Deborah Hix, H. Rex Hartson, "Developing User Interfaces - Ensuring Usability Through Product & Process", John Wiley & Sons, Inc., 1993.

[Inventor95] Open Inventor(TM) - an object-oriented toolkit for developing interactive 3D graphics applications, http://www.sgi.com/Technology/Inventor/, Silicon Graphics, Inc, 1995.

[Mitra99] VRML Browsers testing,http://earth.path.net/vrml2tests/, Mitra VRML 2.0 tests.

[Morrey98] Brad Morrey, "Review: Cosmo Worlds 2.0 gives 3-D Web authoring a boost", http://www.infoworld.com/cgi-bin/displayArchive.pl?/98/20/i12-20.81.htm, Intranets & I-Commerce, Vol. 20, Issue 20, May 18, 1998.

[Mullet95] Kevin Mullet, Darrell Sano, "Designing Visual Interfaces - Communication Oriented Techniques", Sun Microsystems, Inc., 1995.

[Nadeau98] David R. Nadeau, "Introduction to VRML 97", EG'98 Tutorial, EUROGRAPHICS'98, 19th Annual Conference of the European Association for Computer Graphics, August - September 1998.

[Polevoi98] Robert Polevoi, "VRML 97--Using Authoring Tools - Part 2", http://www.webreference.com/3d/lesson32/part2.html, 1998.

[Repository98] The VRML Repository, http://www.sdsc.edu/vrml, SDSC Inc., San Diego Supercomputer Center, 1997.

[Ressler98] Sandy Ressler, Feature comparison of all known VRML97 Browsers, http://vrml.miningco.com/library/weekly/aa070698.htm, MiningCo.com, Inc., 1998.

[Schweber98] Linda Von Schweber, Erik Von Schweber, "Cosmo Software: Cosmo Worlds", http://www.zdnet.com/pcmag/firstlooks/9805/f980515a2.htm, PC Magazine, June 1998.

[Sun97] "Java Programming SL-276 - Student Guide", Sun Microsystems, Inc., 1997.

[Symantec97] "Symantec Visual Café for Java User's Guide", Symantec Corporation, 1997.

[VRML97] VRML97 – International Standard ISSO/IEC 14772-1:1997, http://www.vrml.org/technicalinfo/specifications/vrml97/index.htm, 1997.

[WWW94] "First International Conference on the World-Wide Web", http://pigeon.elsevier.nl/cgi-bin/ID/WWW94, 25-27 May 1994, CERN, Geneva (Switzerland).

Anexo

PARECER

Assunto: Parecer sobre Trabalho desenvolvido no âmbito de uma Tese de Mestrado pelo docente do Instituto Politécnico da Guarda, José Carlos Miranda

Na qualidade de Arquitecto e Técnico Superior da Câmara Municipal da Guarda, actualmente com responsabilidades na área da Gestão Urbanística, procedi a uma análise do protótipo que me foi para o efeito apresentado, merecendo-me o mesmo as seguintes considerações:

- No actual contexto legislativo e no que se refere especificamente às Autarquias locais, não existem disponíveis muitas formas sistematizadas de estas conhecerem a opinião dos seus munícipes, de maneira a que estes possam ser intervenientes activos no processo da tomada de decisões.

- Com efeito, actualmente os mecanismos ao dispor das Autarquias para auscultarem a opinião dos cidadãos sobre os inúmeros assuntos que lhe dizem directamente respeito no âmbito de intervenções urbanísticas (nomeadamente sobre os Planos Municipais de Ordenamento do Território) passam sobretudo pela figura do "Inquérito", instrumento de trabalho muito usado, quer na fase de elaboração das propostas de intervenção, tendo em vista a obtenção de um conhecimento mais profundo da realidade onde se pretende intervir, quer posteriormente, já como "Inquérito Público", para auscultar a opinião pública e os directamente interessados relativamente à validade das propostas submetidas à sua apreciação.

- Trata-se de um instrumento de trabalho que tem sobretudo em vista disponibilizar aos vários intervenientes nos processos urbanísticos uma forma de poderem participar activamente na escolha das várias opções e alternativas colocadas à sua disposição.

- Bem utilizado, é sem dúvida uma forma válida de proceder, pois disponibiliza uma grande quantidade de informação que, se devidamente tratada, pode ser de grande utilidade aqueles que têm a responsabilidade de tomar a decisão final, fornecendo com um grande aproximação qual a opinião e o impacto junto dos munícipes das várias opções tomadas. Em muitos casos, conduz mesmo à alteração das propostas submetidas à consideração dos munícipes no sentido estas de irem ao encontro daquilo que um maior número de intervenientes consideram ser as opções mais correctas para as decisões a tomar pela administração local.

- No entanto, a prática tem demonstrado que nem sempre este método se revela viável, sobretudo pela morosidade de que o processo em si mesmo se reveste, o que leva naturalmente a que a sua utilização seja, salvo raras excepções, restringida às situações expressamente previstas na legislação em vigor.

- No que se refere à situação concreta do protótipo elaborado no âmbito da dissertação em causa, que para o efeito escolheu como aplicação prática um espaço de grande importância para a Cidade da Guarda, a "Praça Velha", já por diversas vezes ao longo da sua história palco de intervenções que nem sempre se revelaram as mais adequadas e pacíficas junto da opinião pública em geral, situação que mesmo recentemente se verificou quando da apresentação das últimas propostas de intervenção para este espaço público, constata-se que o mesmo se poderá revelar de grande utilidade, nomeadamente no que se refere ao recolher junto da opinião pública em geral o seu sentir sobre as intervenções que se possam propor para o local.

- É claro que, no caso concreto da Arquitectura e das intervenções de natureza urbanística, nem sempre a vontade ou a opinião, mesmo que expressa e quantificada da maioria dos cidadãos, se revelam como as mais correctas e fundamentadas para a escolha das opções a tomar, dado que, por natureza, este tipo de intervenções devem ter sempre necessariamente subjacente um conjunto de conhecimentos de carácter técnico e erudito que, por questões de (de)formação sensibilidade e cultura, dificilmente se pode encontrar no cidadão comum, como diariamente se constata em muitas das alturas em que este participa activamente na definição e gestão do espaço urbano.

- No entanto, o protótipo apresentado não deixa de surpreender positivamente pela inovação e imaginação que lhe estão subjacentes, podendo sem sombra de dúvida e com as necessárias adaptações, vir a revelar-se de grande utilidade, exactamente por proporcionar um nova e atractiva maneira de os cidadãos em geral participarem activamente no processo de tomada de decisões sobre assuntos de natureza urbanística que lhes dizem directamente respeito, ao possibilitar-lhes o conhecimento das intervenções que se propõem para um determinado espaço, permitido-lhes ao mesmo tempo intervirem eles próprios através da apresentação das suas propostas, cuja validade podem em tempo real aferir, daí tirando conclusões próprias que, de forma eficaz e rápida, conseguem fazer chegar junto dos centros de decisão.

- O protótipo parece ainda revelar-se extremamente vantajoso, em relação aos métodos tradicionalmente utilizados por, ao recorrer exclusivamente a meios informáticos, poder usufruir de forma directa das vantagens que os mesmos proporcionam, permitindo que os responsáveis pelo processo de decisão conheçam em qualquer altura qual o sentir da opinião pública em geral relativamente ao entendimento que esta faz das propostas submetidas à sua apreciação, possibilitando ainda e imediatamente o tratamento estatístico das sugestões que estes considerem adequadas para manifestar junto dos seus representantes eleitos e dos vários técnicos intervenientes nos processos de natureza urbanística.

- Considero ainda importante chamar a atenção para uma outra valência que, não estando á partida declarada como intenção expressa na concepção do protótipo, parece-me no entanto, poder vir a revelar-se de grande utilidade. Refiro-me concretamente à possibilidade de a tecnologia e os conceitos associados à criação de modelos em realidade virtual, bem como a aplicação informática desenvolvida no âmbito do trabalho apresentado, quando utilizados com o rigor indispensável, poderem ser de grande utilidade para os vários profissionais intervenientes nos processos de natureza urbanística, designadamente os Arquitectos, mas não só. Ao permitir de uma forma expedita a manipulação dos vários elementos que conformam o espaço urbano, poderá enriquecer os métodos de trabalho já correntemente usados por estes profissionais, permitindo-lhes uma nova forma de não só reflectirem sobre o impacto que as suas propostas inevitavelmente têm subjacente, mas também, dada a grande versatilidade que os modelos criados em realidade virtual parecem revelar, de as poderem mais facilmente fazer chegar e compreender junto daqueles a quem se destinam.

- Do protótipo elaborado, tendo em consideração a forma de o mesmo ser utilizado para o fim a que se destina, talvez a principal limitação que se possa referir venha de ter como finalidade a sua utilização na Internet, por isso mesmo tecnicamente limitado às possibilidades de comunicação que esta ainda revela, nomeadamente no que se refere á velocidade de transmissão da informação. No entanto, dado estarmos perante um modelo meramente experimental e obviamente de natureza meramente académica, não me parece que este facto possa para já ser considerado relevante nem pôr em causa a validade que se considera que os conceitos aplicados e desenvolvidos possam vir futuramente a ter no âmbito da sua aplicação enquanto forma de enriquecer o debate de ideias e a participação dos vários intervenientes nos processos de natureza urbanística.

- Gostaria ainda de referir um aspecto que, enquanto Arquitecto, me parece de ter em consideração no desenvolvimento futuro da aplicação prática do protótipo em causa, dado que o mesmo teve por base um espaço existente, por isso mesmo condicionante das intervenções que sobre ele se venham a propor. Mais concretamente, a existência de um excessivo número de variáveis em presença, ou seja, a possibilidade de se intervir mesmo naquilo que, sendo a essência do próprio espaço, terá obviamente que ficar resguardado, sob pena de este poder ser irremediavelmente

descaracterizado por propostas cuja validade deixe de ser poder ser considerada. Parece-me que a solução para esta situação, que no entanto permitiu neste caso acentuar a versatilidade do protótipo e da aplicação desenvolvida, passará sempre pelo desenvolvimento de mecanismos que possam restringir a intervenção dos cidadãos aquilo que em cada caso concreto venha a estar efectivamente submetido à sua apreciação.

Para finalizar realça-se mais uma vez o interesse que se reconhece no trabalho desenvolvido, bem como as potencialidades que o mesmo parece sem sombra de dúvida revelar, e porque não dize-lo também, a curiosidade e expectativa que a sua apresentação despertou.

Câmara Municipal da Guarda, 16 de Março de 1999

Joaquim Luís da Costa Gomes, Arq.