Tecnologias CAVE-HOLLOWSPACE para a Mina do Lousal

Vasco Costa
IST/INESC-ID
Lisboa
vasco.costa@ist.utl.pt

João Madeiras Pereira
IST/INESC-ID
Lisboa
jap@inesc.pt

José Miguel Salles Dias
ADETTI
Lisboa
miguel.dias@iscte.pt

Abstract

In the context of project Ciência Viva da Mina do Lousal, it was decided to construct an immersive virtual reality environment in which a group of people, for example students, can do a virtual visit to a mine. This article discusses and rationalizes the diverse technological choices made to achieve this task.

Keywords

Distributed/Network Graphics, Graphics Packages, Virtual Reality.

1. INTRODUÇÃO

A mina do Lousal é localizada em pleno Alentejo, no concelho de Grândola, distrito de Setúbal. A mina foi explorada entre 1900 e 1988 [Tinoco01]. O seu encerramento conduziu a população local ao desemprego e esquecimento [Monteiro06].

No sentido de revitalizar o tecido económico e social da região, a empresa SAPEC, antiga proprietária das minas, e o município de Grândola conceberam um programa integrado de desenvolvimento. O museu mineiro do Lousal insere-se neste programa. Procura preservar a memória das gentes que viveram e trabalharam neste local e servir como pólo atractivo e energizador para esta região. Ainda em desenvolvimento, aberto ao público desde 2000, o museu recebeu cerca de 20 mil visitantes no ano de 2005. Este trabalho, uma colaboração entre diversas universidades e instituições (Fundação Fédéric Velge; FFCUL; IST, ISCTE; GPCG) com o grupo de trabalho localizado na ADETTI no ISCTE, insere-se na terceira e última fase do programa. Uma visita virtual com descida às minas, a ser localizada no edifício onde ficavam os balneários e laboratórios da antiga mina.

Pretende-se uma visita interactiva à mina feita para grupos, conduzidos por um guia. Deseja-se um ambiente que possa simular o interior de uma mina. Ao mesmo tempo o ambiente físico deve ter flexibilidade de modo a permitir outros usos futuros. Isto pode ser feito via *upgrades, patches* ou mudanças de *software.*

Para este trabalho foi dado um prazo de 5 meses. Dada a dimensão da tarefa a cumprir, recorreu-se a tecnologia disponível no mercado sempre que possível, tentado ir buscar o valor acrescentado à integração de componentes, focando em áreas específicas.

Este artigo foca-se nos problemas da visualização, em tempo real, de cenas complexas num ambiente deste tipo.

O capítulo 2 descreve o ambiente físico. De modo a introduzir os conceitos base para visualização distribuída, o capítulo 3 descreve taxionomias para sistemas gráficos. No capítulo 4 são analisados diversos sistemas gráficos de uso corrente, de acordo com a taxionomia. A arquitectura da aplicação é descrita no capítulo 5, e no capítulo 6 é dada ênfase especial às técnicas de aceleração e efeitos especiais.

2. AMBIENTE FÍSICO

Existe um largo espectro de tecnologias de visualização para realidade virtual [Stevens96]. O espectro vai desde as tecnologias de visualização focadas no indivíduo (e.g. óculos de realidade virtual), até às focadas em organizações com dezenas ou centenas de pessoas (e.g. *display walls* [Araújo05]).

Isto levou à escolha de uma tecnologia tipo CAVE-HOLLOWSPACE [Cruz-Neira93] para o ambiente físico de visualização (ver **Figura 1**).

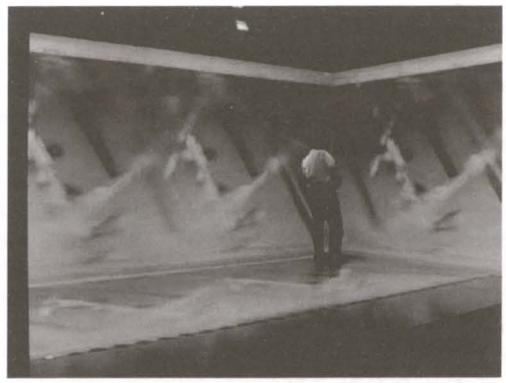

Figura 1: CAVE-HOLLOWSPACE Lousal

CAVE (*Cave Automatic Virtual Environment*) é um ambiente imerso de realidade virtual em que projectores são dirigidos para quatro, cinco ou seis paredes de um

cubo da dimensão de uma sala. O nome é também uma referência à alegoria da caverna na obra "República" por Platão em que um filósofo contempla a percepção, realidade e ilusão.

O ambiente escolhido possui quatro paredes. Frente, lados e chão. A frente e o chão têm projecção estéreo.

A parede frontal tem 5,6 metros de comprimento, as paredes laterais 3,6 metros. O sistema é composto por doze projectores.

De modo a dominar a complexidade, optámos por:

1. Utilizar um aglomerado para visualizar os gráficos. Por razões de desempenho e balanceamento optou-se por ter um aglomerado de seis nós gráficos com duas placas gráficas NVIDIA GeForce 8800 GTX cada. Uma placa gráfica, a gerar as imagens, por cada projector (ver **Figura 2**).
2. Reduzir a quantidade de informação a enviar para a placa gráfica (e.g. não enviando geometria que não é possível visualizar, ou simplificando a geometria enviada). Fazer o processamento visual na placa gráfica, não no CPU, se possível.

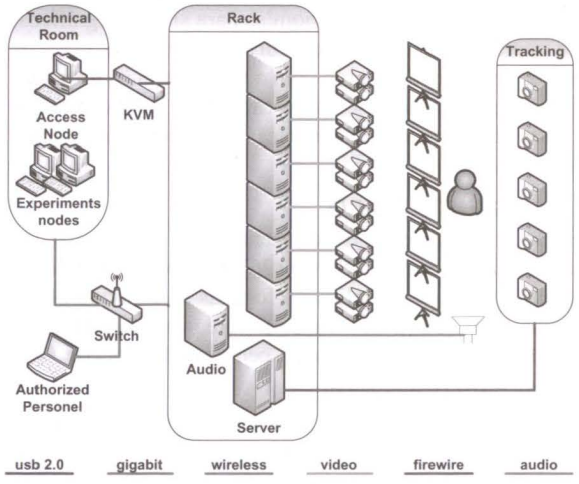

Figura 2: Hardware CAVE-HOLLOWSPACE Lousal

3. SISTEMAS GRÁFICOS

Dado que foi decidido utilizar um aglomerado para a visualização havia a saber que recursos de software já existentes poderiam ser utilizados. Existem diversos sistemas gráficos disponíveis no mercado para a visualização de cenas sob a forma de middleware.

Não se pode escolher uma solução devidamente sem perceber a sua arquitectura [Staadt03]. De modo a poder perceber quais os seus pontos fortes e fracos e analisar se esta terá um desempenho adequado para o problema em questão.

3.1 Arquitectura da aplicação

Pode-se classificar a arquitectura da aplicação de acordo com a localização da base de dados que contém a geometria da cena:

- Mestre escravo.
- Cliente servidor.

3.1.1 Mestre escravo

Na arquitectura mestre escravo a aplicação é executada nos vários nós do aglomerado. Um nó mestre encarrega-se de receber e distribuir as entradas e saídas de dados (e.g. teclado, rato, *tracking*) pelos diversos nós e sincronizar as mudanças de estado. A aplicação nos diversos nós tem de ser sincronizada para garantir a consistência.

3.1.2 Cliente servidor

Na arquitectura cliente servidor a aplicação é executada num só nó do aglomerado. A geometria da cena é depois distribuída por nós servidores de visualização, que simplesmente se encarregam de efectuar o rendering. Pode-se recorrer a diversas técnicas de visualização distribuída (ver **Capítulo 2.2**).

Existem dois modos de rendering: modo imediato, em que as primitivas são enviadas pela rede a cada imagem, e modo de retenção, em que as primitivas são armazenadas entre imagens.

No modo imediato o tempo de arranque da aplicação até ser gerada a primeira imagem é reduzido, contudo existe uma perda de desempenho após o arranque devido ao elevado uso de largura de banda a reenviar primitivas. Já no modo de retenção o tempo de arranque da aplicação até à primeira imagem é superior. Antes do arranque são previamente enviadas e armazenadas as primitivas num grafo de cena.

A vantagem do uso do grafo de cena é o menor tráfego na rede dado que as primitivas apenas são enviadas uma vez. Tendo em conta a largura de banda limitada das redes Ethernet actuais, esta solução costuma resultar num desempenho superior comparativamente ao modo imediato [Staadt03].

3.1.3 Mestre escravo versus Cliente servidor

A arquitectura cliente servidor tem como vantagens a facilidade de depuração de erros, uma vez que o estado actual da aplicação está localizado numa só máquina como habitual, e a inexistência de problemas de falta de coerência nos dados. Executar a aplicação num só nó pode, todavia, degradar o desempenho.

Uma vez que as entradas de dados na arquitectura mestre escravo são infrequentes ou requerem pouca largura de banda para transmitir, esta solução usualmente providencia um desempenho superior à arquitectura cliente servidor. Em contraste existem inconvenientes. É mais difícil de depurar os erros, visto que cada um dos nós tem o seu próprio estado. Por vezes não é óbvio para que informação é necessário assegurar a coerência. A geração de números aleatórios, o uso de temporizadores, bem como outros dados que influenciem o estado da aplicação, não devem ser esquecidos.

3.2 Técnicas de visualização distribuida

Existem diversos algoritmos para distribuir o rendering das primitivas da geometria num aglomerado. No entanto podem-se classificar estes algoritmos em três classes [Pereira96], [Molnar94]:

- Ordenação no início.
- Ordenação no meio.
- Ordenação no fim.

3.2.1 *Ordenação no início*

Na ordenação no início a área de visualização é subdividida por rectângulos. As primitivas são aleatoriamente distribuídas por nós, que efectuam as operações suficientes para saber que volumes de visualização intersectam. Depois são redistribuídas para os nós dedicados a esses volumes de visualização.

A fase inicial, em que as primitivas são distribuídas, usa bastante largura de banda. É possível reduzir esta largura de banda através de caches. Em cenas com primitivas grandes, ou que intersectem vários volumes de visualização, existe duplicação de trabalho a efectuar a transformação e rasterização da geometria em diversos nós.

3.2.2 *Ordenação no meio*

Na ordenação no meio as primitivas são aleatóriamente distribuídas pelos nós, que efectuam as transformações na geometria. A geometria já transformada é redistribuída para rasterização. Este tipo de ordenação não é vulgarmente utilizado em aglomerados de PCs.

3.2.3 *Ordenação no fim*

Na ordenação no fim as primitivas são aleatóriamente distribuídas pelos nós, que efectuam as transformações na geometria e a rasterização. Os fragmentos (e.g. R,G,B,A,Z) rasterizados por cada nó são unidos numa imagem final. Esta fase de composição da imagem final usa bastante largura de banda uma vez que os retalhos de cada um dos nós são enviados pela rede.

4. ANÁLISE DOS SISTEMAS GRÁFICOS

Decidiu-se escolher sistemas gráficos, com uma licença de software aberto, multiplataforma. Deste modo pensa-se estar a reduzir as probabilidades de ficar preso a uma solução com custo de aquisição elevado e problemas de suporte no futuro.

Em seguida são descritos os sistemas gráficos considerados.

4.1 Chromium

O sistema gráfico Chromium [Humphreys02] é o sucessor do WireGL, inicialmente desenvolvido por Greg Humphreys na Universidade de Stanford nos Estados Unidos da América. Esta solução proporciona suporte de aglomerado para aplicações OpenGL. Em princípio, basta trocar a biblioteca do OpenGL pela do Chromium e tem-se uma aplicação distribuída sem mais esforço de programação.

De modo a manter compatibilidade com o OpenGL não existe grafo de cena. Para diminuir o tráfego na rede, apenas são retransmitidas as alterações nas primitivas desde a última imagem. Isto é feito com recurso a uma cache, das primitivas básicas enviadas na última imagem, nos servidores de visualização.

As aplicações Chromium utilizam usualmente uma arquitectura cliente servidor, em que um ou mais nós de aplicação cliente 3D enviam comandos gráficos para um ou mais servidores de visualização remotos. Cada servidor de visualização está ligado a um écran. Podem-se ter vários servidores de visualização remotos num aglomerado para múltiplos écrans.

Opcionalmente pode ser utilizada uma arquitectura mestre-escravo utilizando programação paralela num aglomerado. Isto requer a invocação de chamadas específicas do Chromium pela aplicação.

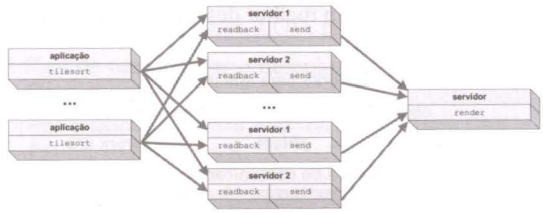

Figura 3: Aglomerado com ordenação no início usando ordenação no fim com recurso a um servidor compositor

São disponibilizadas primitivas de sincronização OpenGL:

- barreiras (criadas com `glBarrierCreateCR`)
- semáforos (criadas com `glSemaphoreCreateCR`).

A distribuição de inputs e outros eventos no cluster pode ser feita com recurso ao Cluster Rendering Utility Toolkit (CRUT).

Devido à flexibilidade da configuração do Chromium é possível construir aplicações com arquitecturas híbridas mestre-escravo/cliente-servidor (ver **Figura 3**).

O Chromium é uma solução multiplataforma que assenta sobre o OpenGL e TCP/IP. Possui um desempenho superior a outras soluções anteriores para visualização remota de aplicações OpenGL como o GLX do X Window System.

As técnicas de visualização distribuída utilizadas no modo cliente-servidor são a ordenação no início e/ou ordenação no fim.

Uma vez que não existe grafo de cena, a geometria é enviada imagem a imagem, é de esperar que seja usada muita largura de banda para cenas complexas com muita geometria. Ao fim e ao cabo a proximidade do Chromium ao OpenGL é uma espada de dois gumes. Esta proximidade leva a um interface de programação familiar que, contudo, não permite fazer certas optimizações porque a interface com o programador é de muito baixo nível. É a interface de uma biblioteca gráfica de modo imediato. O suporte OpenGL actual do Chromium também tem algumas limitações: falta o suporte para OpenGL 2.0 e o suporte para display lists é incompleto. Isto pode levar a problemas inesperados caso não sejam tomadas em conta as particularidades desta solução.

4.2 Syzygy

O sistema gráfico Syzygy [Schaeffer03] foi inicialmente desenvolvido por Benjamin Schaeffer na Universidade de Illinois nos Estados Unidos da América. Esta é uma solução completa para a implementação de aplicações de realidade virtual em aglomerados. Permite a programação em C++ ou Python em ambiente multiplataforma.

A arquitectura da aplicação é configurável. Pode ser utilizada uma topologia cliente servidor em que o grafo de cena Myriad é distribuído por diversos nós que executam o render através do serviço `szgrender`. Em alternativa pode ser usada uma arquitectura mestre escravo que permite um melhor desempenho à custa de

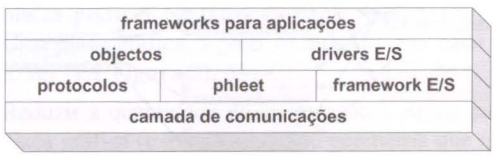

Figura 4: Plataforma Syzygy

complexidade adicional na programação. O grafo de cena distribuído Myriad possui algumas limitações em termos de funcionalidade. Estas limitações podem ser ultrapassadas via uso directo de OpenGL numa configuração mestre escravo. Note-se que o recurso a uma solução deste tipo implica a perda de funcionalidade do grafo de cena. Por exemplo o carregamento de objectos, armazenados em ficheiros OBJ ou 3DS, torna-se difícil. Existe suporte para a geração de áudio espacial 3D com recurso ao serviço SoundRender.

O Syzygy está implementado em cinco camadas (ver **Figura 4**). A camada de baixo nível abstrai as chamadas que variam de entre os diversos sistema operativos para um interface comum. Exemplos das funcionalidades suportadas a este nível incluem: threads, sincronização e sockets. Por cima desta camada existe uma infra-estrutura de comunicações genérica que funciona sobre TCP, UDP ou memória partilhada. Por cima desta infra-estrutura de comunicações são implementados protocolos para sincronização, distribuição do grafo de cena, som, e suporte para os dispositivos de entrada de dados. O aglomerado, ou Phleet, executa o serviço `szgserver` que proporciona o ambiente necessário para as aplicações Syzygy bem como a gestão de processos. A manipulação dos objectos de som e grafo de cena processa-se na camada superior. Por fim existe a *framework* para aplicações com suporte para as funcionalidades essenciais para uma aplicação de realidade virtual: grafo de cena, câmara, etc.

4.3 OpenSG

O sistema gráfico OpenSG [Voß02] foi concebido inicialmente na SIGGRAPH 99 por Dirk Reiners (Instituto Fraunhofer), Allen Bierbaum (VRAC da Universidade de Iowa) e Kent Watsen (NPS). Implementado em C++ multiplataforma, é um sistema gráfico de alto nível, tem como objectivo abstrair os problemas de baixo nível da programação OpenGL. Tem um grafo de cena distribuído com um nível de suporte avançado para *multi-threading* e aglomerados. Procura resolver os problemas de design do OpenInventor e Performer [Rohlf94] que ficaram em aberto após o cancelamento do projecto Fahrenheit.

Para suportar *multi-threading* o OpenSG possui suporte para aspectos. Um aspecto, neste contexto, é uma cópia do grafo de cena. Assim podemos ter várias *threads*, cada uma com a sua cópia, sem ter problemas de escritas concorrentes na mesma zona de memória. Quando não existem problemas com acessos concorrentes ao grafo de cena, por exemplo quando as *threads* limitam-se a ler o grafo, pode-se utilizar um mesmo aspecto para várias *threads* de modo a poupar memória. Existe suporte para sincronizar os diversos aspectos.

A arquitectura da aplicação suportada é cliente servidor. Uma vez que existe um grafo de cena, com uma cópia armazenada em cada um dos nós de visualização, não é necessário estar constantemente a reenviar geometria estática imagem a imagem pela rede. Apenas quando são efectuadas alterações na geometria existe comunicação. Isto leva a um consumo reduzido de largura de banda em cenas com modelos estáticos. Em cenas dinâmicas, o tráfego de rede aumenta consideravelmente. O OpenSG suporta as técnicas de visualização distribuída de ordenação no início e ordenação no fim.

4.4 VR Juggler

O VR Juggler [Bierbaum03] foi desenvolvido na Universidade de Iowa. É uma plataforma para o desenvolvimento de aplicações de realidade virtual, com suporte para aglomerados, de arquitectura mestre-escravo. A aplicação é executada em cada um dos nós do aglomerado, sendo um dos nós designado por mestre. Este nó mestre encarrega-se de sincronizar a execução com os nós escravos. Possui suporte para diversos sistemas operativos. Básicamente o VR Juggler é o meio que permite aos vários nós de um aglomerado comunicarem informação de estado, sincronizarem-se. Distribui as entradas de dados, possuindo suporte para vários dispositivos de entrada como: rato, teclado, joysticks e luvas. Também possui suporte para diversos dispositivos de saída tais como: monitores convencionais, capacetes de realidade virtual, dispositivos tipo PowerWall, ou dispositivos tipo CAVE. O VR Juggler contém suporte para a programação de aplicações sobre OpenGL ou VTK, bem como sobre os grafos de cena OpenGL Performer, OpenSG, OpenSceneGraph.

A plataforma base VR Juggler (ver **Figura 5**) possui diversos componentes que assentam sobre uma panóplia de tecnologias standard como o OpenGL (gráficos) e o CORBA (comunicações) e standards multiplataforma de facto como o OpenAL (som).

O **VRJuggler Portable Runtime**, vulgo VPR, é uma camada de abstração de diversas funcionalidades base do sistema operativo que fornece ao seu utilizador suporte

independente da plataforma para *threads*, *sockets* TCP/IP, e primitivas para interface série.

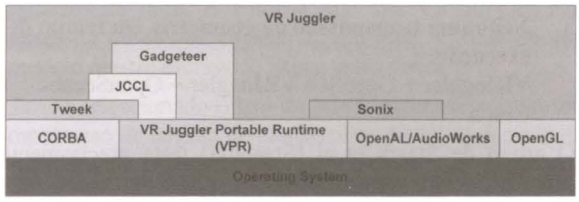

Figura 5: Plataforma VR Juggler

O **Sonix** é uma abstração do suporte áudio que permite suporte multiplataforma. O suporte desta funcionalidade pareceu-nos algo incipiente e levou-nos a considerar utilizar outras soluções multiplataforma para o som como o FMOD.

O **Tweek** é uma colecção de tecnologias C++ e Java que permitem a um interface Java comunicar com uma aplicação C++ via CORBA.

O **JCCL** é um sistema de configuração baseado na linguagem XML.

O **Gadgeteer** é um sistema de gestão de dispositivos de entrada. Possui um *Remote Input Manager* que partilha a informação dos dispositivos com as máquinas do sistema. É possível escrever drivers para novos dispositivos de entrada de dados caso não existam.

O **Maestro** é um componente adicional que permite o lançamento remoto de aplicações num aglomerado, bem como outras funcionalidades para gestão de um aglomerado. Por cada tipo de aplicação que pode ser lançado no aglomerado deve existir um ficheiro de configuração denominado *stanza* com a informação de configuração necessária.

O VR Juggler usa o módulo JCCL atrás mencionado para todas as configurações de sistema. Infelizmente a configuração do VR Juggler é verdadeiramente barroca, com inúmeros pequenos detalhes, tornando a configuração de um sistema deste tipo uma tarefa difícil. O interface gráfico de configuração VRJConfig, a alternativa a editar os ficheiros XML à mão, é pouco amigável.

4.5 OpenSceneGraph

O OpenSceneGraph, vulgo OSG, foi inicialmente criado por Don Burns em 1998 de modo a poder executar um simulador de asa delta, originalmente desenvolvido para o sistema operativo IRIX da SGI sobre o Performer, num PC com Linux. É um grafo de cena C++ multiplataforma que tenta utilizar os conceitos de padrões de desenho e aproveitar as funcionalidades da STL do ANSI C++.

Não possui suporte funcional para aglomerados. De modo a suportar um ambiente do tipo de um aglomerado pode-se recorrer ao auxílio do Chromium ou VRJuggler.

Eventualmente cresceu e atraiu uma comunidade de desenvolvimento. O aumento da procura e interesse nesta solução levou um dos autores, Robert Osfield, a criar uma empresa de suporte. Tem existido um esforço contínuo de melhoramento desta solução, sendo lançadas regularmente novas versões com mais funcionalidades, bem como suporte para as novas plataformas 64-bit para PCs da arquitectura X86-64.

Este grafo de cena possui um suporte avançado para o carregamento de diversos tipos de modelos (3D Studio, Lightwave, Wavefront OBJ, COLLADA, VRML, etc) bem como imagens (PNG, JPEG, TGA, TIFF, GIF, BMP, etc) através de um sistema de plugins.

Possui suporte base para hierarquias de volumes envolventes, remoção de objectos fora do volume de visualização, impostores, LOD, paginação, e portais.

É fácil de personalizar o desenho de objectos através da redefinição do método de desenho. O processamento no OpenScenegraph possui três fases. *Update*, *Cull* e por fim *Draw* (ver **Figura 6**).

Na fase de *Update* é actualizada a cena. Nesta fase podem, por exemplo, ser executadas simulações de física ou efectuados movimentos de objectos dinâmicos. Na fase de *Cull* é efectuada a remoção de primitivas que estão fora do volume de visualização ou são muito pequenas (de dimensão inferior a um pixel). Por fim é efectuada a fase de *Draw* e a cena é desenhada.

Figura 6: Processamento do OpenSceneGraph

A plataforma base contém várias bibliotecas. Exemplos:

- **osg**
  Biblioteca para o grafo de cena.
- **osgUtil**
  Biblioteca utilitária com funcionalidades como triangulação de Delaunay, cálculo do ponto de intersecção de um raio com a cena, conversão de geometria para *strips* de triângulos, etc.
- **osgDB**
  Biblioteca que permite a leitura e escrita de grafos de cena.
- **osgParticle**
  Biblioteca que possui suporte base para partículas de fogo, chuva, explosões, etc.

A biblioteca *osg* do OpenSceneGraph possui várias classes auxiliares para gestão de memória (alocação de memória com contagem de referências e smart pointers) e operações matemáticas (operações de vectores a duas, três e quatro dimensões, quaterniões, operações para matrizes, conversão de ângulos em graus de e para radianos, etc).

5. PLATAFORMA DE SUPORTE

Descrevem-se, em seguida, as opções tomadas quanto à plataforma de software para suporte à aplicação.

5.1 Sistema gráfico

De entre os sistemas gráficos analisados consideram-se as seguintes opções como alternativas credíveis, tendo sido testadas por diversos projectos bem sucedidos, para efectuar visualização num aglomerado:

- Chromium
- OpenSG
- VR Juggler + OpenSG
- VR Juggler + OpenSceneGraph

Tendo em conta o que se sabe, sobre a previamente mencionada arquitectura, destas soluções espera-se à priori que:

O Chromium tenha problemas de desempenho em cenas com muitas primitivas. Não possui grafo de cena, transmite primitivas a cada imagem, a largura de banda Ethernet a 1000Mbps que se tenciona utilizar será usada por seis nós gráficos com duas placas gráficas cada. Para além disto, as suas funcionalidades são basicamente um subconjunto das do OpenGL, existirá trabalho adicional a implementar funcionalidades tais como remoção de primitivas fora do volume de visualização. Ou usar um sistema gráfico extra de alto nível sobre o Chromium que contenha essas funcionalidades e seja compatível com o Chromium. Como o OpenSG, OpenSceneGraph ou outro grafo de cena.

O OpenSG usa menos largura de banda que o Chromium. Também possui uma gama de funcionalidades mais rica para o programador. Em caso de geometria dinâmica na cena, como água em movimento, destruição de partes da cena irá haver transmissão de geometria que irá congestionar a rede.

As soluções com VR Juggler possuem suporte para diversos dispositivos de entrada. Como usam o modelo mestre-escravo terão um ritmo de visualização mais consistente visto que não será transmitida geometria pela rede que leve a reduções do desempenho. A aplicação é executada em todos os nós, cada um tem a base de dados da cena completa. Apenas serão transmitidos dados posicionais e outra informação reduzida, o mínimo essencial para sincronizar os diversos nós.

O modelo cliente-servidor no Chromium e OpenSG leva a menos problemas de sincronização dos dados já que a aplicação é executada numa só máquina. Já nas soluções com VR Juggler que usam o modelo mestre-escravo será necessário ter mais cuidado na sincronização das aplicações dos vários nós.

Podem-se separar estas quatro alternativas, de grosso modo, em três grupos de acordo com o modo como a base de dados da cena e a geometria é transmitida durante a execução da aplicação:

1. **Geometria estática e dinâmica transmitida imagem a imagem:**
   Chromium
2. **Geometria dinâmica transmitida imagem a imagem:**
   OpenSG
3. **Nenhuma transmissão de geometria em tempo de execução:**
   VRJuggler + OpenSG, VRJuggler + OpenSceneGraph

O artigo de Staadt et al [Staadt03] testa precisamente soluções nestes três grupos. O sistema testado é uma Wall 2x2 com quatro nós servidores (ou escravos), cada nó com uma placa gráfica ligada a um projector. Existe um nó cliente (ou mestre) adicional que executa a aplicação.

Os nós estão ligados a um switch Ethernet a 100 Mbps. O sistema para o Lousal, ainda por adquirir, possuirá 10x mais largura de banda, contudo também possuirá 4x mais placas gráficas com um desempenho mais elevado a uma resolução superior. Espera-se que estes resultados tenham um bom correlacionamento com os do sistema para o Lousal.

De modo a eliminar diferenças de desempenho a carregar modelos, e outras que pudessem causar perturbação nos resultados, Staadt et al utilizaram OpenSG sobre Chromium, OpenSG, e OpenSG sobre VR Juggler.

Os testes de cenas, com geometria estática, consistiram em visualizar modelos com diferentes níveis de complexi-dade, e efectuar translações e rotações destes no espaço da Wall.

Estes testes permitem determinar problemas de sincronização entre nós, balanceamento de carga no *rendering*, distribuição da geometria, impacto de objectos posicionados na fronteira entre écrans.

A solução Chromium acabou por ter um ritmo de visualização muito reduzido em comparação com as soluções OpenSG e VR Juggler. Isto deve-se ao elevado tráfego de rede gerado pelo Chromium. Mesmo com o modelo estático, se as primitivas ao serem enviadas não estiverem organizadas por proximidade, o *buffering* do Chromium não é eficaz. O VR Juggler demora mais tempo até estabelecer a sincronização inicial para a primeira imagem que o Open SG e em especial o Chromium que não tem de efectuar sincronização entre nós, contudo as diferenças de desempenho em todos os outros testes, durante a execução da aplicação, são evidentes.

O Chromium não consegue obter ritmos de visualização interactivos a mais de 25 imagens por segundo para muitas das cenas. As soluções OpenSG e VR Juggler são boas para cenas com geometria complexa como as cenas que se desejam visualizar para o Lousal. O Chromium apenas pode ser usado em cenas simples ou existe muita largura de banda disponível. Testes independentes efectuados pelo Sérgio Miguel e pela Dora Rita [Cabrita06] na LEMeWall do Tagus [Araújo05] confirmam a grande diferença de desempenho entre o Chromium e as outras soluções com menos requisitos de largura de banda.

Em cenas dinâmicas o VR Juggler possui um desempenho superior ao OpenSG. Uma vez mais a solução vencedora em termos de desempenho é aquela que usa menos recursos de rede. A solução que usa VR Juggler não necessita de enviar geometria pela rede, virtude do seu modelo mestre escravo, e ganha estes testes.

Tendo seleccionado o uso do VR Juggler, devido ao seu menor uso de largura de banda, resta decidir que grafo de cena usar. O OpenSceneGraph 1.2, disponível na data de início do trabalho, possui funcionalidades de remoção de primitivas não visíveis mais completo que a última versão estável do OpenSG 1.6 então disponível. Também possui mais funcionalidades como sistemas de partículas. O facto de muitos dos membros de equipa, nomeadamente no ISCTE, possuírem prática no uso do OpenSceneGraph também deverá facilitar a curva de aprendizagem e ajudar a reduzir o prazo de desenvolvimento.

5.2 Arquitectura da aplicação

A aplicação (ver **Figuras 7, 8**) utiliza diversos componentes sobre uma plataforma Microsoft Windows.

Figura 7: Aplicação em modo de projecção única (OsgApp.exe)

Segue-se uma breve descrição de alguns destes componentes:

- **VR Juggler**

  Utilizado para a configuração dos volumes de visualização.

- **OpenSceneGraph (inclui osgFX, osgParticle)**

  Grafo de cena.

- **Cal3D e OsgCal**

  Bibliotecas para a animação de personagens.

- **AGEIA PhysX**

  Biblioteca para efeitos de física, como colisões entre personagem, objectos e a cena. Suporta objectos estáticos e dinâmicos.

- **FMOD**

  Bibliotecas usada para o áudio espacial 3D.

- **Lua e OsgLua**

  Bibliotecas para a linguagem de *scripting*.

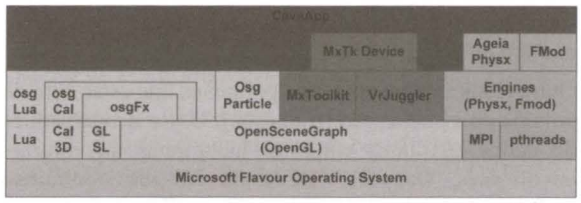

Figura 8: Plataforma de suporte

Também foram utilizados outros componentes de apoio, como uma implementação da API de **POSIX Threads** para a plataforma Windows, e a implementação **MPICH2** da API de comunicações por mensagem para aglomerado MPI. A implementação do suporte de comunicações é bastante elaborada, merecedora de um artigo por si só, portanto fora do âmbito deste artigo.

5.3 Edição de mapas

Para a edição de mapas de cena da aplicação é utilizada a ferramenta **GtkRadiant** (ver **Figura 9**). Esta é conhecida pela comunidade de jogos Quake III, e foi configurada para os fins da plataforma desenvolvida. A ferramenta suporta a leitura ficheiros de formato `.map`. Os modelos 3D têm sido concebidos na aplicação 3ds Max.

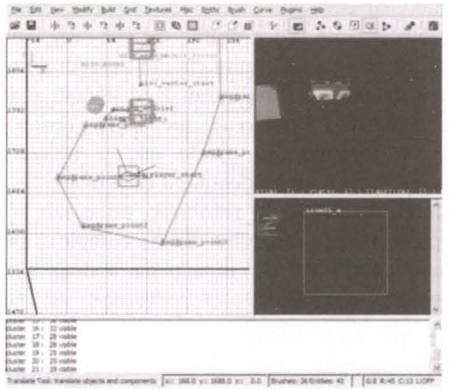

Figura 9: Editor de mapas GtkRadiant

6. TÉCNICAS DE ACELERAÇÃO

Pretende-se visualizar cenas complexas 3D com um número elevado de triângulos nos modelos. De modo a não saturar a placa gráfica com a geometria da cena, serão utilizadas técnicas que evitem enviar geometria que não se irá visualizar (i.e. **técnicas de oclusão**) e, se necessário, técnicas que simplifiquem a geometria da cena (i.e. **técnicas de nível de detalhe**).

Também será feito processamento de efeitos na placa gráfica, em alternativa ao CPU, recorrendo a **shaders**.

6.1 Técnicas de oclusão

As APIs gráficas utilizadas hoje em dia, como o OpenGL [Woo03], já implementam remoção de superfícies ocultas. Basicamente, numa malha fechada, os polígonos, cuja face não está orientada na direcção da câmara, não são processados sendo feita a sua rejeição cedo no *pipeline* de visualização. A principal limitação desta técnica, como implementada no OpenGL, é que a

sua complexidade do algoritmo varia linearmente com o número de polígonos na cena.

Outra técnica é não enviar os polígonos que estão fora do volume de visualização [Haines02]. O uso de estructuras de dados auxiliares como as hierarquias de volumes envolventes, Octrees, BSPs ou Kd-trees, permitem trivialmente rejeitar um conjunto de polígonos de uma só vez.

Por fim, não enviar os polígonos que estão a ser obscurecidos por outros polígonos. Um exemplo de técnicas deste tipo é o uso de *Potentially Visible Sets* (PVS) numa BSP para uma cena interior. Deste modo podem ser determinadas quais são as salas adjacentes visíveis a partir da sala actual e trivialmente rejeitar aqueles que não são visíveis de qualquer posição na sala.

Dado que uma mina no subsolo constitui uma cena interior, neste caso, o uso de BSPs com PVS é a solução pretendida. Foi implementada uma solução tendo por base as BSPs do jogo Quake III. Deste modo foi possível aproveitar trabalho já feito no compilador de BSP.

6.2 Técnicas de nível de detalhe

Para um objecto próximo da câmara o utilizador consegue visualizar exaustivamente os polígonos de uma malha. No entanto, se a malha estiver suficientemente afastada, os polígonos diminuem até ficarem do tamanho de píxeis no écran. No limite da distância a malha é representada por um ponto até que por fim desaparece do volume de visualização. Dado que o utilizador não consegue discernir tanto detalhe a essa distância, devido aos limites inerentes da resolução do écran e da sua visão, pode-se substituir uma malha complexa por outra mais simples, ou até por uma placa a duas dimensões [Haines02].

Em princípio serão usadas cenas interiores, em salas de dimensão limitada, com elevado nível de oclusões. Espera-se que estas técnicas tenham um impacto mais reduzido no aumento do desempenho nos ambientes fechados. Tenciona-se usar estas técnicas para objectos ou personagens no interior da cena. Para este efeito foi implementada uma solução de nível de detalhe discreto.

6.3 Shaders

Utilizando *shaders* é possível construir imagens complexas através da combinação de outras imagens e elementos mais simples. Por exemplo pode-se querer visualizar uma laranja com uma textura rugosa. O efeito de rugosidade da superfície pode ser simulado utilizando shaders que combinem texturas com a cor, sombras e clarões.

Os *shaders* expõem funcionalidades dos processadores de vértices e fragmentos do GPU ao programador.

Uma vez que se tenciona utilizar tecnologias baseadas em OpenGL, sobre placas NVIDIA, existem duas opções:

Cg [Fernando03] é uma linguagem específica a produtos NVIDIA com sintaxe similar ao C. Funciona para programas OpenGL e DirectX.

GLSL [Kessenich04] é uma linguagem similar ao Cg da NVIDIA. Contudo é extensão ARB no OpenGL 1.5 e vem standard com o OpenGL 2.0. É suportada por outros fornecedores, como a ATI, para além da já mencionada NVIDIA.

A GLSL expõe dois pontos do pipeline OpenGL ao programador: processador de vértices, processador de fragmentos.

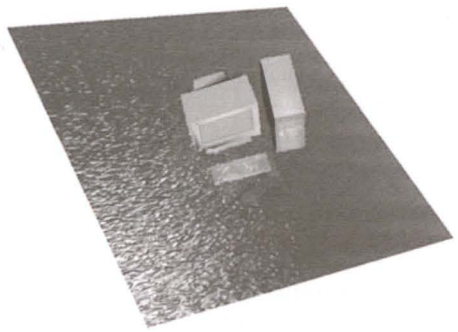

Figura 10 – Água animada com bump-mapping, reflexões e refracções

De modo a evitar problemas de portabilidade do código desenvolvido no futuro optou-se pela linguagem padrão OpenGL GLSL. Deste modo espera-se que seja possível aplicar o código desenvolvido e o conhecimento ganho em múltiplos sistemas operativos e diverso *hardware* no futuro.

Os *shaders* são utilizados para implementar certos efeitos gráficos. Por exemplo água em movimento (ver **Figura 10**), certas funcionalidades das partículas do grafo de cena, texturas volumétricas baseadas em *Perlin noise* (ver **Figura 11**), sombras limitadas a objectos específicos da cena.

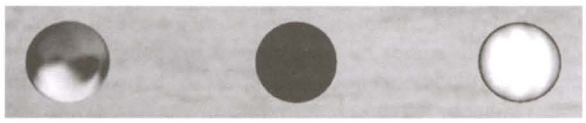

Figura 11: Detalhe de texturas volumétricas na aplicação

No caso das sombras, dada a complexidade do desenvolvimento de suporte genérico de sombras num motor gráfico [Kuehl07], foi feito um *backport* do suporte de sombras do OpenSceneGraph versão 2.0 para a versão 1.2. Esta versão, lançada posteriormente ao início do trabalho, possui uma implementação de *shadow maps* bem como uma pré-implementação incompleta de *shadow volumes*. Infelizmente foram detectados diversos problemas, durante a fase de prototipagem, no suporte de *shadow maps*: differenças entre a projecção da luz e da câmara, *aliasing*, devido à resolução limitada da textura de sombra, problemas de ruído no *shadow map* quando aplicado a grafos de grande dimensão devido à precisão limitada dos valores de Z no *Z-buffer*. Estes problemas têm sido factores impeditivos para o uso de sombras na globalidade do grafo de cena neste trabalho. Quando a

técnica é aplicada a um modelo específica da cena, e não ao grafo de cena completo, estes problemas são minimizados (ver **Figura 12**).

Figura 12 – Sombras baseadas em shadow mapping, aplicadas a um modelo, na aplicação

7. CONCLUSÕES E TRABALHO FUTURO

A construção de um sistema multi-écran com os gráficos fornecidos por um aglomerado é difícil. Existem diversos obstáculos a resolver, do lado da visualização, como os problemas da sincronização e do desempenho.

Escolheu-se uma solução envolvendo VR Juggler para distribuição e OpenSceneGraph como grafo de cena. À volta desta solução foi criada uma plataforma de suporte para aplicações na CAVE-HOLLOWSPACE da Mina do Lousal. Está a ser criado um primeiro conteúdo para esta plataforma. A plataforma é extensível, podendo ser utilizada para outros conteúdos.

Na execução deste trabalho foi dada ênfase à visualização realista de cenas com elevada complexidade em arquitecturas paralelas. Deste ponto de vista existe muito ainda por fazer. Sombras, inclusivé com penumbras, reflexões, visualização de modelos 3D com nível de detalhe contínuo, são exemplos de problemas deixados em aberto. Expera-se abordar estes temas em mais profundidade numa futura tese de doutoramento.

8. AGRADECIMENTOS

Agradecemos ao Bruno Araújo e ao Luciano Soares, que já trilharam estes caminhos dos sistemas multi-projector, pela informação disponibilizada. Por fim não podemos esquecer o Rafael Bastos, que orientou a equipa nesta fase final, bem como o Nelson Carvalho, Francisco Pires e Rui Silvestre, sem os quais este não teria sido possível.

9. REFERÊNCIAS

[Tinoco01] A. Tinoco, A. M. C. Matos, I. M. Ribeiro, M. L. Santos, M. Plácido et al: A Valorização do Património Geológico e Mineiro do Lousal. Congresso Internacional sobre Património Geológico e Mineiro, Beja, (Outubro 2001).

[Monteiro06] C. Monteiro: Lousal vai ter Centro de Ciência Viva. notícia no jornal regional Setúbal na Rede (Agosto 2006).

[Stevens96] R. Stevens: The engines that Drive Collaboration. Argonne National Laboratory. (1996).

[Araújo05] B. R. Araújo, T. Guerreiro, R. Jota, J. A. Jorge, J. M. Pereira: LEMe Wall: Desenvolvendo um Sistema de Multi-Projecção (2005).

[Cruz-Neira93] C. Cruz-Neira, D. J. Sandin, T. A. Defanti: Surround-screen projection-based virtual reality: the design and implementation of the CAVE. Proceedings of the 20th annual conference on Computer graphics and interactive techniques (1993).

[Staadt03] O. G. Staadt, J. Walker, C. Nuber, e B. Hamann: A Survey and Performance Analysis of Software Platforms for Interactive Cluster-Based Multi-Screen Rendering. Eurographics Workshop on Virtual Environments (2003).

[Pereira96] J. A. M. Pereira.: Cinerealismo em Arquitecturas Paralelas de Uso Geral. Tese de Doutoramento (1996).

[Molnar94] S. Molnar, M. Cox, D. Ellsworth, e H. Fuchs: A Sorting Classification of Parallel Rendering. IEEE Computer Graphics and Applications (1994).

[Humphreys02] G. Humphreys, M. Houston, R. NG, R. Frank, S. Ahern, P. Kirchner, E J. T. Klosowski: Chromium: A Stream-Processing Framework for Interactive Rendering on Clusters. SIGGRAPH 2002.

[Voß02] G. Voß, J. Behr, D. Reiners and M. Roth: A Multi-thread Safe Foundation for Scene Graphs and its Extension to Clusters. Fourth Eurographics Workshop on Parallel Graphics and Visualization (2002).

[Rohlf94] J. Rohlf, J. Helman: IRIS Performer: A High Performance Multiprocessing Toolkit for Real-Time 3D Graphics. 21st International Conference on Computer Graphics and Interactive Techniques.

[Schaeffer03] B. Schaeffer, C. Goudeseune: Syzygy: Native PC Cluster VR. Virtual Reality 2003.

[Bierbaum03] A. Bierbaum, C. Just, P. Hartling, K. Meinert, A. Baker, C. Cruz-Neira: VR Juggler: A Virtual Platform for Virtual Reality Application Development. Virtual Reality (2001).

[Woo03] M. Woo, J. Neider, T. Davis et al: OpenGL Programming Guide, Fourth Edition, Addison-Wesley, 2003.

[Cabrita06] S. M. E. Cabrita, D. R. M. Esteves: Plataformas de Suporte para Visualização Interactiva no Sistema Display Wall do Tagus. Trabalho Final de Curso da LEIC no IST (2006).

[Haines02] E. Haines, T. A. Möller: Real-Time Rendering, A.K. Peters Ltd., 2002.

[Fernando03] R. Fernando, M. Kilgard: The Cg Tutorial. Addison-Wesley, 2003.

[Kessenich04] J. Kessenich, D. Baldwin, R. Rost: The Open GL Shading Language. 3Dlabs, 2004.

[Kuehl07] B. Kuehl, K. J. Blom, S. Beckhaus: Generation of Shadows in Scene Graph based VR. WSCG 2007.