PUC

ISSN 0103-9741

Monografias em Ciência da Computação
nº 11/04

Grafo de Cena e Realidade Virtual

Romano José Magacho da Silva
Alberto Barbosa Raposo
Marcelo Gattass

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22453-900

RIO DE JANEIRO - BRASIL

A versão integral deste relatório técnico se encontra em:
http://www.tecgraf.puc-rio.br/~abraposo/pubs/TechReports/MCC11-04.zip

Grafos de Cena

Grafos de cenas são ferramentas conceituais para representação de ambientes virtuais tridimensionais nas aplicações de computação gráfica [43]. Um ambiente virtual é uma representação de diversos aspectos do mundo real ou abstrato. Os aspectos que são considerados em uma aplicação de computação gráfica são: posição do objeto, forma, textura da superfície, iluminação, entre outros [14]. Cada um desses aspectos e seus atributos estão bastante detalhados na literatura, porém apresentamos aqui um resumo dos mais importantes:

- Descrição geométrica: A descrição geométrica é qualquer maneira de se representar a forma da entidade que pode ser processada para se obter uma imagem dessa entidade. A maneira mais comum é a representação aproximada por um conjunto de polígonos (mais especificamente por triângulos). O grau de complexidade da descrição geométrica é, em geral, diretamente proporcional à qualidade visual, porém inversamente proporcional à velocidade com que a imagem é gerada;
- Câmera: A câmera é a visão do mundo virtual. Geralmente ela é uma câmera de projeção perspectiva;
- Transformação: O objeto é posicionado no mundo virtual através de uma transformação geométrica. Ela transforma as coordenadas locais do objeto nas coordenadas do mundo virtual. As transformações são muito importantes para definir a hierarquia dos objetos;
- Aparência: Material, textura, transparência, sombra e reflexão estão entre os diversos atributos que definem a aparência de um objeto. Assim como a descrição geométrica, a aparência interfere diretamente na imagem final sendo gerada e na velocidade de geração;
- Comportamento: um objeto pode ser estático ou dinâmico. O objeto dinâmico é aquele que muda de posição, forma ou aparência entre um quadro e outro;
- Iluminação: várias fontes de luz podem ser adicionadas à cena e vários são os modelos de iluminação que podem ser empregados. O mais utilizado é o Gouraud [18] pois o seu cálculo é feito em hardware nas placas gráficas atuais, porém com o advento dos processadores gráficos programáveis, outros modelos podem ser usados, como o Phong [32] e o anisotrópico [34].

Cada um desses aspectos deve ser inserido em um grafo de cena para representar o ambiente virtual. O grafo de cena é formado, portanto, por nós conectados por arestas compondo um grafo acíclico direcionado. Cada nó possui um conjunto de atributos que podem, ou não, influenciar seus nós conectados. Os nós são organizados de uma maneira hierárquica correspondendo semântica e espacialmente com o mundo modelado.

Os nós podem ser divididos em três categorias: nó raiz, nós intermediários que são chamados de nós internos ou nós de agrupamento e os nós folha que estão localizados no final de um ramo. O nó raiz é o primeiro nó do grafo e todos os outros nós estão ligados a ele direta ou indiretamente. O nós internos possuem várias propriedades, sendo o uso mais comum o de representar transformações 3D (rotação, translação e escala). Os nós folha contêm, geralmente, a representação

geométrica de um objeto (ou dados de áudio, quando o grafo de cena possuir esse recurso).

Construindo um Grafo de Cena

Os nós devem ser conectados formando um grafo acíclico e direcionado. As conexões entre eles devem satisfazer as relações espaciais e semânticas existentes no mundo real. A Figura 1 mostra um exemplo de um grafo de cena representando uma casa. A casa é dividida em quartos e os quartos possuem diversos objetos (nós folha), como cama e cadeira, por exemplo. O grafo foi construído de maneira a manter a noção intuitiva dos relacionamentos entre os objetos citados.

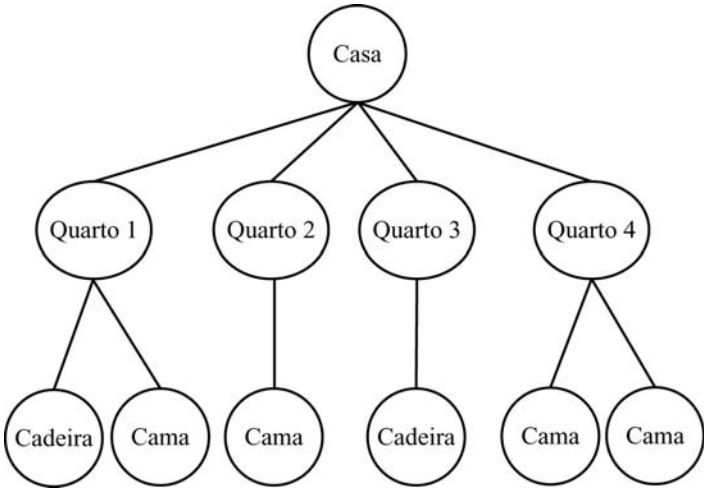

Figura 1: Grafo de cena bem construído de uma casa.

O mesmo grafo poderia ser construído de outra maneira, agrupando-se as cadeiras e camas em nós separados, logo, ao invés de uma casa com quatro quartos, teríamos uma casa e, logo abaixo, dois nós agrupando cadeiras e camas (Figura 2). Esse modelo está semanticamente relacionado com a casa, pois a mesma possui camas e cadeiras, porém os objetos geométricos não estão agrupados espacialmente. Esse resultado indesejado é contrário ao uso eficiente do grafo de cena, como veremos adiante.

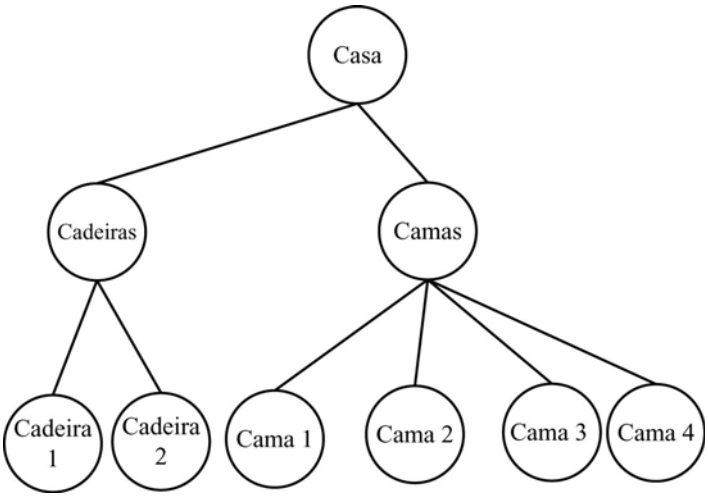

Figura 2: Estratégia de uso não eficiente de um grafo de cena.

Propriedades

Os grafos de cena implementam um princípio chamado de herança de estado. Os nós internos armazenam o estado do sistema, onde estado significa a posição e a orientação dos objetos no ambiente virtual e seus atributos de aparência. A herança de estado é uma propriedade dos grafos de cena que determina que cada nó deve herdar as propriedades de estado de todos os seus ancestrais no grafo até a raiz.

Analisemos, novamente, o modelo da casa e admitamos que seus objetos estejam modelados em relação à origem do sistema de coordenada. Em cada quarto acrescentamos uma translação que irá posicionar seus quartos corretamente em relação à casa. A casa pode ser ainda rotacionada, por exemplo, a fim de ficar voltada para uma determinada direção. Essa cena é ilustrada na Figura 3. Devido à herança de estado, todas as geometrias identificadas pelos nós raízes herdarão as propriedades dos seus ancestrais e serão posicionadas corretamente. A herança de estado é, portanto, uma ferramenta bastante útil para organização da cena 3D.

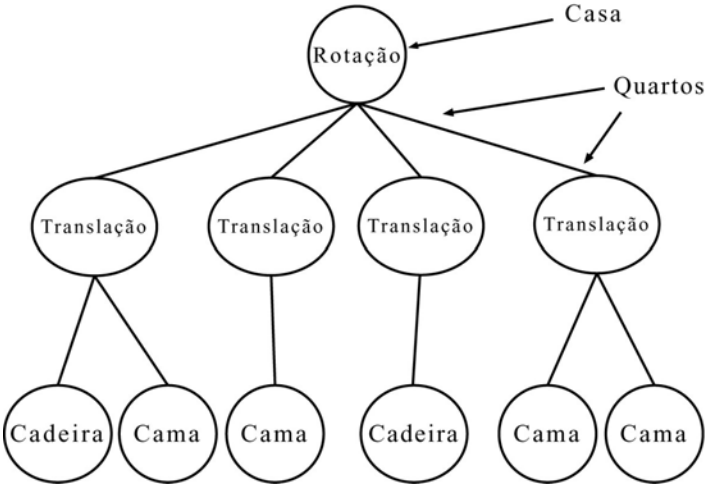

Figura 3: Usando transformações para posicionas os quartos e a casa.

Todos os nós de um grafo de cena podem possuir atributos como material, textura, transparência, entre outros. Todos esses atributos são herdados dos nós ancestrais. Um determinado nó pode sobrescrever um determinado atributo e, sendo assim, toda sua subárvore será modificada. A

Figura 4 ilustra um exemplo de um conjunto de objetos com um determinado material (cor) e alguns nós sobrescrevendo esse atributo.

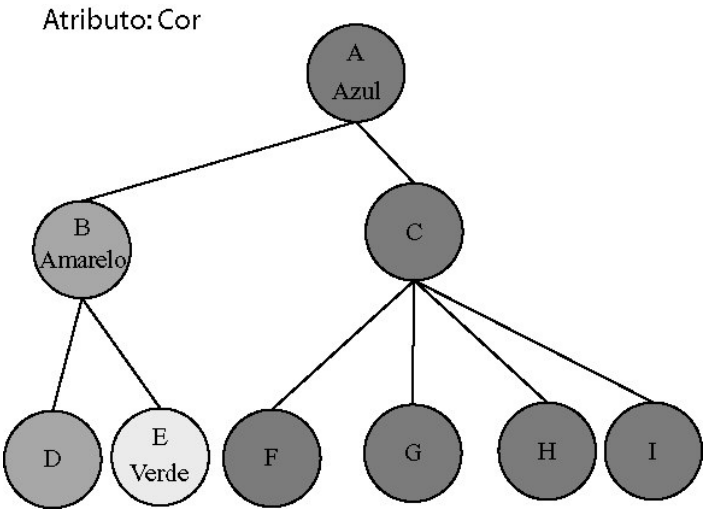

Figura 4: Materiais redefinidos ao longo do grafo. Os nós herdam a cor do nó ancestral.

Otimizações

As aplicações de computação gráfica possuem um *pipeline* de renderização responsável por transformar um vértice em coordenadas do mundo para as coordenadas da tela. Esse *pipeline* é composto por três etapas: *Aplicação*, *Geometria* e *Rasterização*.

Atualmente, as etapas de *Geometria* e *Rasterização* são realizadas em hardware. A primeira é responsável pela transformação e iluminação dos vértices que chegam à placa gráfica. A *Rasterização* recebe os *pixels* com seus valores de cor interpolados de acordo com o algoritmo de Gouraud e o valor da profundidade daquele *pixel*. Com esse valor, um teste de profundidade (Z-buffer [2]) é feito para saber se aquele *pixel* será renderizado ou descartado.

A etapa de *Aplicação* é responsável pelo envio de vértices para a placa gráfica. Essa etapa é totalmente controlada pelo usuário e é nela onde atua o grafo de cena.

Em geral, quanto menos vértices forem enviados para a placa gráfica, melhor será o desempenho geral da aplicação. O desempenho também é influenciado pela troca de estado da aplicação. Se a aplicação modificar várias vezes a textura ativa a ser usada em um modelo, a placa gráfica precisa ser informada. A cada troca de estado na placa gráfica, uma validação deve ser realizada e essa validação é uma etapa demorada. Portanto, é importante que a aplicação agrupe seus polígonos por estado, ao invés de mandá-los em ordem aleatória.

Uma boa implementação de um grafo de cena deve ser capaz de otimizar o número de vértices enviados para a placa gráfica, assim como o número de trocas de estado. Para isso, algumas técnicas devem ser empregadas.

Todos os nós do grafo de cena possuem um atributo denominado volume envolvente. Esse atributo é um volume simples, geralmente uma caixa alinhada ou uma esfera, que engloba o conteúdo de todos os nós abaixo do nó em questão. O volume envolvente é utilizado em um grafo de cena para verificação de visibilidade e descarte. As técnicas empregadas para geração dos volumes envolventes estão detalhadas em [38].

As técnicas mais empregadas de descarte em um grafo de cena são:

- Descarte por volume de visão: o grafo de cena testa a interseção o volume envolvente do nó com o volume de visão do observador. Se o volume envolvente estiver completamente fora do campo de visão, o nó e toda a sua subárvore são descartados. Se o mesmo estiver completamente dentro do campo de visão, o nó e toda a sua subárvore são percorridos e caso a interseção seja parcial, o teste é refeito durante o percurso da subárvore;
- Descarte por oclusão: os algoritmos de descarte por oclusão têm por objetivo evitar a renderização de primitivas que estejam ocultas por outras partes da cena. A idéia por trás dos algoritmos de oclusão é realizar algum pré-processamento ou alguns testes durante a renderização para evitar que dados sejam enviados desnecessariamente para a placa. Os algoritmos mais conhecidos são o *Shadow Frusta* [21], *Hierarchical Z-Buffer (HZB)* [19] e *Hierarchical Occlusion Maps* [45]. O HZB está implementado em hardware nas placas gráficas atuais.
- Descarte de objetos pequenos: o volume envolvente dos nós é projetado na tela. Caso ele ocupe menos que um valor limite em *pixels*, esse objeto é descartado. A idéia desse descarte é a de que objetos muito pequenos em cenas complexas não influenciam muito na imagem final.

Com o descarte, o grafo de cena evita o envio de objetos que não estão visíveis para a placa gráfica, dessa forma reduzindo a quantidade de vértices a serem renderizados. É interessante notar ainda a característica hierárquica desses algoritmos. Se um nó for descartado, toda a sua subárvore será descartada. Por esse motivo, uma cena organizada espacialmente será muito mais eficiente do que uma cena que foi montada aleatoriamente.

Outra otimização que reduz a quantidade de vértices é chamada de Níveis de Detalhe (*LOD – Level of Detail*) [26]. A idéia por trás dessa técnica é a de que objetos muito distantes do observador podem ser renderizados com menor qualidade visual (menor número de polígonos) do que objetos mais próximos (Figura 5). Os níveis de detalhe podem ser estáticos, e portanto gerados uma única vez em alguma etapa de pré-processamento, ou dinâmicos, sendo gerados conforme a distância do observador ao objeto.

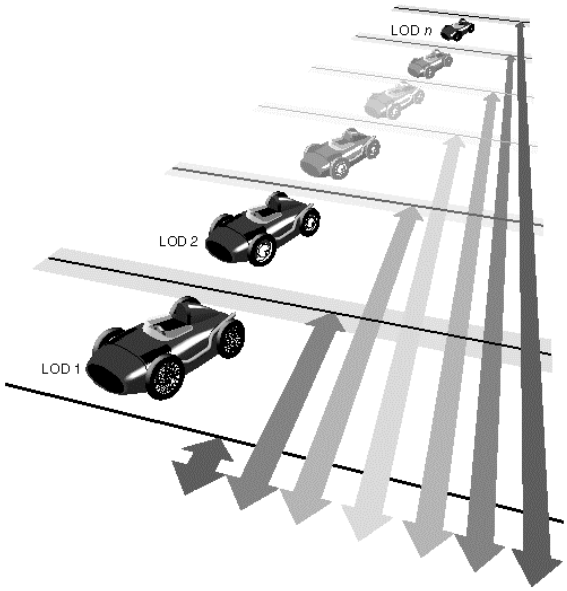

Figura 5: Níveis de Detalhes: objetos mais distantes são vistos com menos detalhe [44].

A fim de suportar a otimização de troca de estados, o grafo de cena deve ser capaz de armazenar todos os atributos que serão utilizados para renderizar antes de efetivamente enviar os vértices para placa de vídeo. De posse dessa lista de atributos, ele ordena os objetos geométricos por estado e depois os envia para a placa.

Vantagens

Uma vantagem direta que o grafo de cena traz com todas as otimizações que ele implementa é a melhoria do desempenho da aplicação. Mas essa não é a única vantagem, como mostramos na lista a seguir:

- **Produtividade**: Os grafos de cena diminuem o trabalho de se desenvolver aplicações gráficas de alto desempenho. O grafo de cena gerencia toda a parte gráfica, reduzindo as várias linhas de código que seriam necessárias para implementar a mesma funcionalidade utilizando uma interface de programação baixo nível, como a OpenGL. Além disso, vários conceitos avançados de programação orientada a objetos, como o uso de padrões de projeto [16], tornam o grafo de cena uma aplicação flexível e de fácil reuso. Além disso, um grafo de cena geralmente é acompanhado de outras bibliotecas responsáveis por gerência de janelas, carregamento de modelos 3D e imagens. Tudo isso faz com que o usuário possa desenvolver uma aplicação com poucas linhas de código;
- **Portabilidade**: Os grafos de cena encapsulam todas as tarefas de baixo nível necessárias para renderizar a cena e ler e escrever arquivos, reduzindo, ou até mesmo extinguindo, a quantidade de código que é específica de alguma plataforma a ser inserido na aplicação. Sendo assim, se o grafo de cena for portável, imediatamente toda a aplicação será portável, sendo necessário apenas uma nova compilação ao se mudar de plataforma;

- **Escalabilidade**: os grafos de cena são feitos para funcionar em configurações simples baseadas em computadores de mesa e placas gráficas aceleradores convencionais ou em hardware complexos, tais como *clusters* de máquinas gráficas, ou sistemas multiprocessados/*multipipe*. O desenvolvedor não se preocupa com a configuração em que ele irá rodar a aplicação, podendo estar focado exclusivamente no seu desenvolvimento.

Ferramentas disponíveis

Várias bibliotecas de grafo de cena existem há algum tempo. A lista a seguir não é, de maneira alguma, completa, mas apresenta os mais utilizados:

- SGI OpenGL Performer (http://www.sgi.com/products/performer)
- OpenSceneGraph (http://www.openscenegraph.org)
- OpenSG (http://www.opensg.org)
- Open Inventor (http://oss.sgi.com/projects/inventor)
- PLIB (http://plib.sf.net)
- SGL (http://sgl.sf.net)
- OpenRM (http://openrm.sf.net)

Apenas as duas primeiras da lista serão detalhadas a seguir.

SGI OpenGL Performer

O OpenGL Performer, da empresa Silicon Graphics, é uma interface de programação para o desenvolvimento de aplicações gráficas 3D em tempo real. Ela funciona em plataformas IRIX, Linux e Windows baseada na biblioteca gráfica OpenGL [29]. Além do grafo de cena, ela possui uma série de outros módulos que permitem o melhor desempenho da aplicação. A Figura 6 ilustra a organização das bibliotecas que compõem a interface.

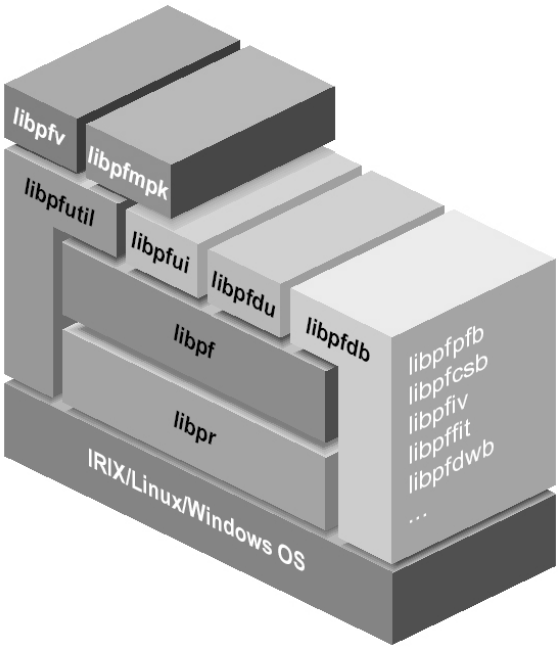

Figura 6: Hierarquia de módulos do OpenGL Performer [30].

A biblioteca *libpr* é a biblioteca base e a responsável pelo desempenho de renderização. Ela é uma biblioteca desenvolvida em C e com uma interface C++, orientada a objeto, que fornece funções de renderização otimizadas, controle de troca de estados eficiente e outras funcionalidades gráficas independentes de aplicação (rotinas de matemática, detecção de colisão, criação e gerência de janelas, etc).

Acima da *libpr* está localizada a *libpf*. Essa biblioteca fornece um ambiente de visualização em tempo real através de um grafo de cena, além de um sistema de renderização multiprocessado. Esse sistema multiprocessado aproveita o máximo das arquiteturas SGI.

As outras bibliotecas são responsáveis por leitura de arquivo (*libpfdb*), criação de manipuladores e componentes de interface com usuário (*libpfui*) e configuração de renderização multiprocessada e multicanal, além de interface com usuário (*libpfutil*).

O grafo de cena contém a informação que define o mundo virtual. Ele inclui descrição da geometria e a sua aparência, informação espacial, tais como transformação, animação, níveis de detalhe, efeitos especiais entre outros. O OpenGL Performer atua no grafo de cena para realizar a renderização e o descarte. Para isso, entram em ação dois conceitos: *channel* e *pipe* (Figura 7).

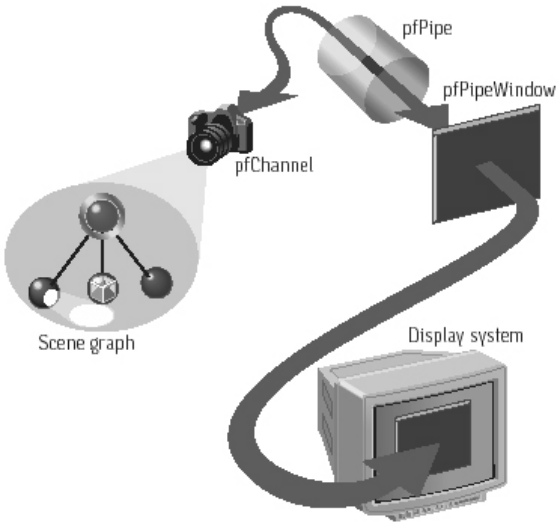

Figura 7: O caminho do grafo de cena [30].

Um *channel* é equivalente a uma câmera movendo-se dentro da cena. Cada *channel* está associado a uma área da configuração final de visualização. O *pipe* é o coração de todo processamento feito pelo OpenGL Performer. Ele renderiza cada *channel* na sua área de projeção na tela. Nos equipamentos da SGI, cada *pipe* pode ser associado a um processador gráfico. Essas máquinas são chamadas *multi-piped*.

O OpenGL Performer é uma excelente solução de grafo de cena e renderização em tempo real. Porém, ele é uma solução comercial de alto custo, desenvolvida especialmente para os equipamentos SGI. Em outros equipamentos, boa parte das otimizações que são extremamente importantes, como o conceito de *multi-pipe*, não está presente.

OpenSceneGraph

O *OpenSceneGraph* é uma interface de programação construída sobre a biblioteca gráfica OpenGL [29]. Ele é responsável pela gerência do grafo de cena e otimizações gráficas. Ele é multi-plataforma, gratuito e o seu código é aberto. A Figura 8 ilustra os módulos da interface.

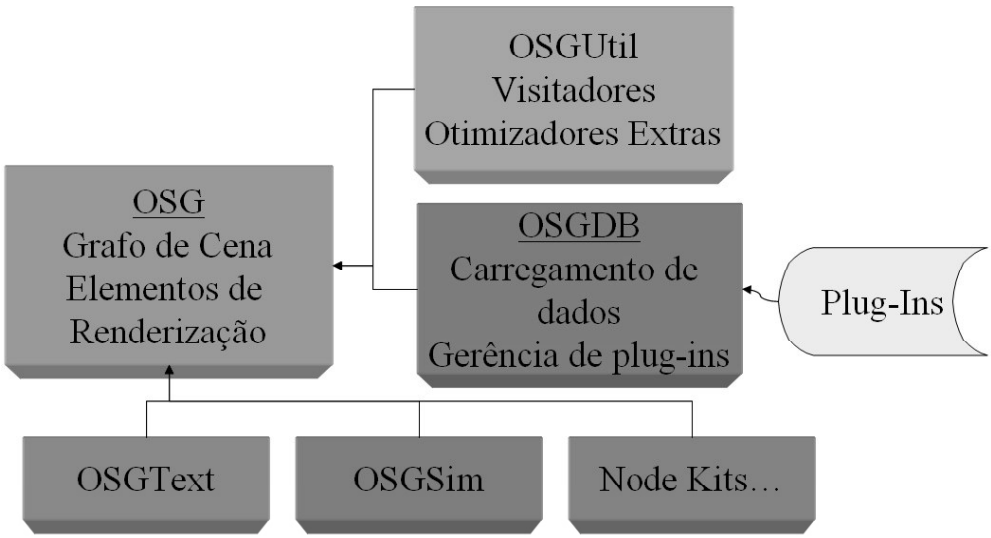

Figura 8: Módulos da interface de programação *OpenSceneGraph*.

O módulo *osg* é responsável pelo gerenciamento do grafo de cena. Ele possui as otimizações básicas, como descarte hierárquico por campo de visão e por oclusão, níveis de detalhe e gerência de troca de estados. O módulo *osgUtil* possui a câmera da cena e outras rotinas de otimização, como a remoção de transformações estáticas, fusão de nós do mesmo estado, partição espacial da cena para melhor descarte, geração de geometrias otimizadas para renderização, e outras.

Os **Visitadores** existentes no módulo *osgUtil* são responsáveis pelo percurso no grafo. Os **Visitadores** implementam o padrão de projeto de mesmo nome [16]. A Figura 9 ilustra o diagrama de classes que implementam esse padrão. Existem, geralmente, 2 visitadores padrão: o visitador de atualização e o de descarte.

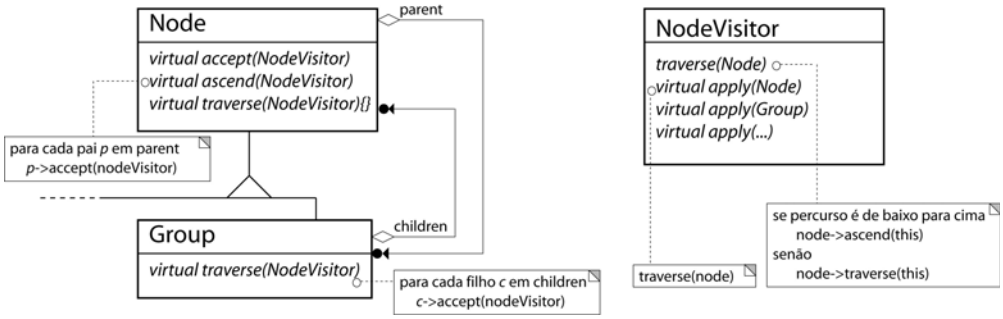

Figura 9: Padrão de projeto visitadores utilizado para percorrer o grafo de cena.

O visitador de atualização é responsável por atualizar os objetos dinâmicos do grafo. O visitador de descarte percorre o grafo e elimina todos os objetos passíveis de exclusão e retorna uma lista de objetos renderizáveis. A etapa de renderização é feita com o resultado obtido pelo visitador de descarte. Durante a renderização a troca de estado é feita por demanda (*lazy-state change*).

O módulo *osgDB* é responsável por implementar uma cadeia de responsabilidade [16] para o carregamento de arquivos de modelos tridimensionais e de imagens. Com isso, o usuário não precisa se preocupar com a leitura em baixo nível dos arquivos, tornando sua aplicação mais portável. Os

leitores de arquivo são implementados na forma de *plugins*, tornando o *osgDB* bastante flexível e extensível. Os módulos *osgText* e *osgSim* são extensões do módulo *osg* para suportar a renderização de texto e outros nós não-específicos, como, por exemplo, pontos de luz.

O *OpenSceneGraph* possui a vantagem de ser uma interface de programação disponível gratuitamente e de código aberto. Apesar de não possuir muita documentação disponível, a quantidade de exemplos e o próprio código fonte auxiliam a sua compreensão.

Por ser orientado a objetos, ele é bastante extensível, permitindo ao usuário a criação de novas funcionalidades sem a necessidade de modificação do seu código fonte. A comunidade existente em torno do OpenSceneGraph é bastante numerosa e existem diversas outras bibliotecas que criaram uma interface para ele, como é o caso do motor de física Vortex da CMLabs [31] e a biblioteca de terrenos Demeter [13].