3
Linha de Produção Gráfica Distribuída

Neste capítulo, é proposta uma extensão à linha de produção gráfica convencional (*graphics pipeline*) destinada à renderização distribuída. Esta apresentação inclui as responsabilidades de cada estágio dessa linha de produção, a troca de mensagens entre os estágios e as implementações de algoritmos de renderização distribuída que foram feitas utilizando esse sistema.

PUC-Rio - Certificação Digital Nº 0310807/CA

3.1
Linha de Produção Utilizada

O maior desafio da área de renderização distribuída e paralela é desenvolver algoritmos escaláveis sob a largura de banda disponível da rede. Por este motivo, foi escolhido o modelo Mestre-Escravo para a distribuição de tarefas de renderização.

A Figura 3.1 ilustra a linha de produção gráfica convencional. Nela, a aplicação envia primitivas geométricas ao estágio da geometria, que transforma as primitivas para o espaço de *clipping*. Essas primitivas são passadas para o estágio de rasterização, que as rasteriza, gerando fragmentos. A cor final de cada fragmento é calculada tendo como base a cor da primitiva e texturas. Posteriormente, são realizadas operações como teste de *alpha*, teste de profundidade e *blending*.

A linha de produção gráfica foi ampliada para uma linha de produção distribuída, na qual é possível identificar três unidades lógicas: o mestre, as unidades de renderização e o compositor de resultados. O mestre tem como responsabilidades a distribuição de tarefas de renderização e a manutenção do sincronismo da aplicação nas unidades de renderização. As unidades de renderização são responsáveis pelo processamento contínuo de tarefas de renderização, fazendo após o término de cada tarefa a leitura, a compressão e o envio do seu *framebuffer* para o compositor de resultados. O compositor de resultados deve continuamente receber os *framebuffers* enviados pelas

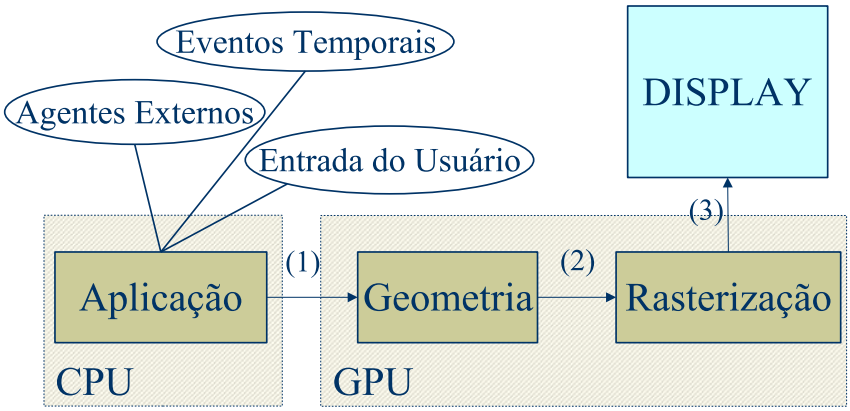

Figura 3.1: Linha de produção gráfica convencional: (1) primitivas do modelo; (2) primitivas no espaço de tela; (3) fragmentos.

unidades de renderização, descompactá-los e compô-los no dispositivo de visualização, realizando *alpha-blending* ou teste de profundidade no caso da visibilidade ser resolvida nessa unidade. Um diagrama da linha de produção distribuída encontra-se na Figura 3.2 No protótipo implementado, o mestre e o compositor de resultados foram implementados na mesma máquina.

As operações feitas com o *framebuffer*, como renderização, compactação, recebimento e envio, são independentes e formam os estágios da linha de produção distribuída. Na Figura 3.3 encontra-se um diagrama detalhado da linha de produção gráfica do nó mestre e dos nós escravos.

A implementação de cada estágio das linhas de produção do nó mestre e dos nós escravos foi feita em uma linha de execução própria (*thread*). Assim, as operações de entrada e saída (como o envio e o recebimento de dados pela rede) podem ocorrer em paralelo às operações feitas na CPU e na GPU. Isso também simplifica a implementação dos estágios da linha de execução, tornando-a mais modular, porém é necessário cuidar das condições de concorrência que emergem de tal modelo de programação.

A linha de produção do nó mestre foi dividida em 4 estágios: Controle, Recebimento, Descompressão e Composição. A troca de mensagens entre os estágios foi esquematizada na Figura 3.4.

O estágio de Controle é responsável por gerenciar todo o sistema de renderização distribuída. Ele deve receber as entradas do usuário e de sistemas externos, empacotá-las e enviá-las aos nós escravos, mantendo a câmera e o modelo 3D sincronizados. Para isto, foi desenvolvido um pequeno protocolo de comunicação de parâmetros, com o qual é possível empacotar parâmetros da aplicação, do sistema de renderização distribuída e da técnica de renderização distribuída utilizada.

PUC-Rio - Certificação Digital Nº 0310807/CA

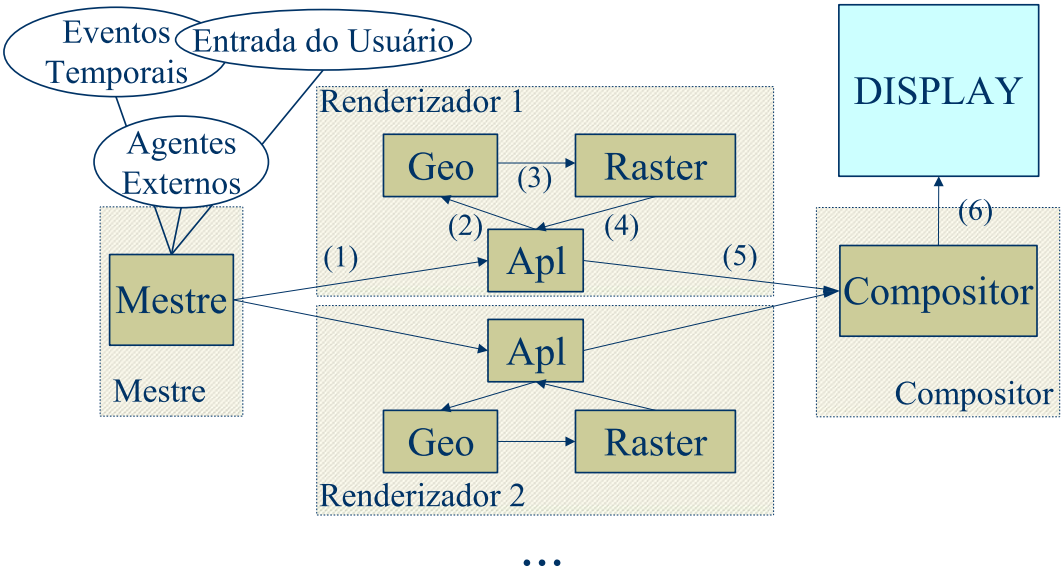

Figura 3.2: Linha de produção gráfica distribuída: (1) requisições de renderização; (2) primitivas do modelo; (3) primitivas no espaço de tela; (4) *framebuffer*; (5) *framebuffer* comprimido; (6) imagem final.

A partir da necessidade de se renderizar um quadro, são empacotados os parâmetros pertinentes, tais como as matrizes da câmera, o tempo de simulação e os parâmetros da aplicação e da técnica de renderização em mensagens de requisição de renderização que são enviadas a todos os nós escravos. No fim da renderização, cada nó escravo envia uma mensagem sinalizando o término de sua tarefa de renderização para o estágio de Controle, que pode, se desejável, enviar uma nova tarefa de renderização para este nó.

Conforme os nós enviam seus sinais de término de tarefa, o estágio de Controle pode lhes enviar novas requisições de renderização, possivelmente portadoras de novos parâmetros relacionados com a técnica de renderização, que visam balanceamento de carga, minimização do uso da rede e de recursos gráficos, entre outras coisas.

O estágio de Recebimento é responsável por receber os *framebuffers* compactados de cada escravo e pô-los numa fila de *framebuffers* a serem descomprimidos. O recebimento do *framebuffer* proveniente de um escravo só se inicia após o estágio de Controle receber um sinal de término de tarefa de renderização.

O estágio de Descompressão retira os *framebuffers* compactados de sua fila e os descompacta. Após descompactado, cada *framebuffer* é posto em uma fila de *framebuffers* a serem compostos.

Por último, o estágio de Composição retira *framebuffers* descompac-

PUC-Rio - Certificação Digital Nº 0310807/CA

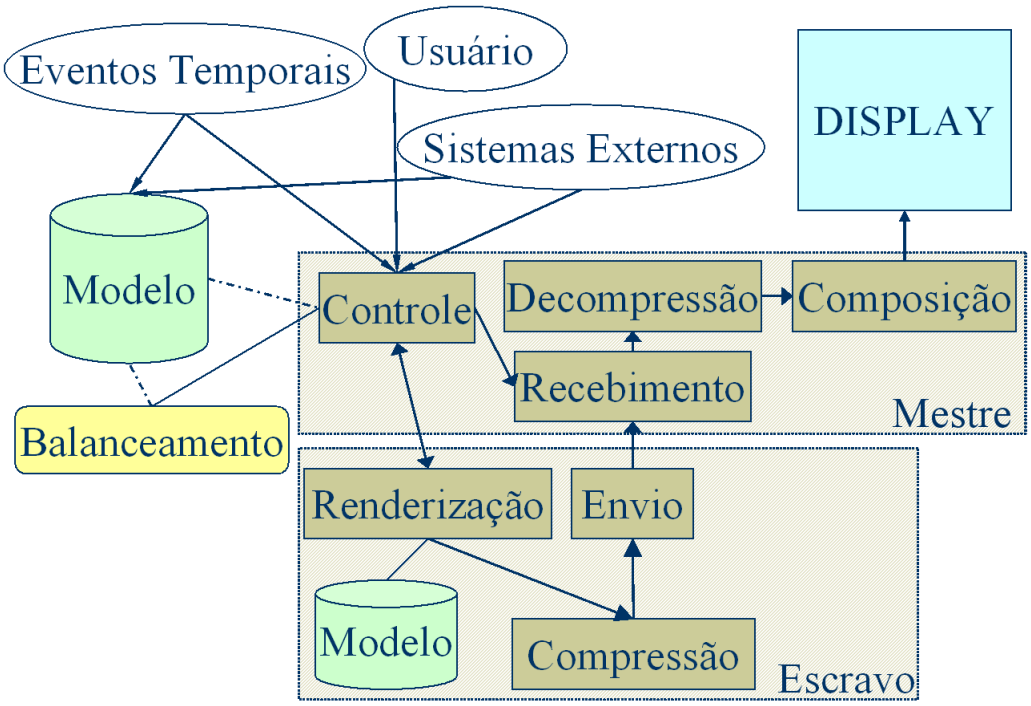

Figura 3.3: Linha de produção gráfica utilizada.

tados da fila e os compõe no dispositivo de visualização. O procedimento de composição é dependente da técnica de renderização distribuída utilizada.

A linha de produção de cada nó escravo foi dividida em 3 estágios: Renderização, Compressão e Envio.

O estágio de Renderização possui um canal de comunicação com o estágio de Controle do nó mestre, por onde recebe requisições de tarefas de renderização. Quando chega uma tarefa, ele a processa, lê o *framebuffer* resultante da renderização e o põe em uma fila de *framebuffers* a serem compactados. Então, esse estágio envia uma mensagem ao nó mestre sinalizando o término desta tarefa e fica livre para tratar uma nova requisição.

O estágio de Compressão aguarda por novos *framebuffers* a serem compactados. Quando um novo *framebuffer* é inserido na fila de *framebuffers* compactados, esse estágio o retira da fila e o compacta, pondo o resultado em uma fila de *framebuffers* a serem enviados. O algoritmo de compressão utilizado no protótipo implementado é o LZF [28]. Este foi escolhido por ter sido desenvolvido especificamene para aplicações que visam tempo real, aplicações estas que necessitam de uma compressão bastante rápida de seus dados.

O estágio de Envio retira *framebuffers* compactados da fila e os envia ao estágio de Recebimento do nó mestre, com o qual esse estágio possui outro canal de comunicação.

PUC-Rio - Certificação Digital Nº 0310807/CA

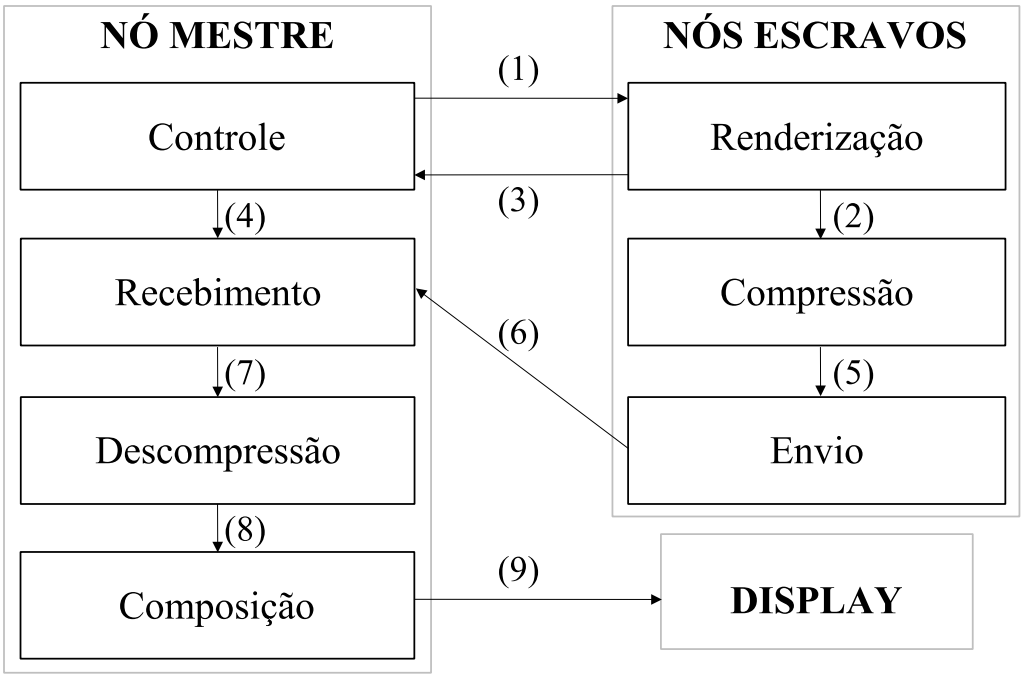

Figura 3.4: Troca de mensagens entre os *threads* do nó mestre e dos nós escravos: (1) tarefa de renderização; (2) *framebuffer* renderizado; (3) sinal de término; (4) sinal de *framebuffer* pronto para ser enviado; (5) *framebuffer* comprimido; (6) *framebuffer* enviado pela rede; (7) *framebuffer* recebido; (8) *framebuffer* descomprimido; (9) *framebuffer* escrito no *framebuffer* local.

PUC-Rio - Certificação Digital Nº 0310807/CA

3.2 Técnicas de Renderização Implementadas

Três técnicas de renderização distribuída foram implementadas utilizando a linha de produção distribuída descrita acima: um sistema de arquitetura com ordenação no início, um sistema de arquitetura híbrida com ordenação no início e no fim e um sistema de renderização volumétrica distribuída com ordenação no fim.

O sistema com ordenação no início utiliza o modelo 3D replicado em todos os nós do agrupamento. Este sistema divide a tela em ladrilhos disjuntos, os quais são distribuídos aos nós escravos, um ladrilho por nó. No caso de cenas com gargalo na geometria, é esperado que cada nó somente desenhe as primitivas cuja projeção intersecta o volume de visão determinado pelo seu ladrilho. Isto é normalmente feito com o auxílio de algoritmos de descarte hierárquico de objetos fora do volume de visão [2, 16]. Já no caso de cenas com gargalo na rasterização, a divisão do esforço total é automática, pois a tela está sendo dividida entre os nós escravos. Esse sistema inclui um algoritmo de balanceamento de carga que tenta calcular uma partição de tela em ladrilhos de igual carga, cujo procedimento será descrito no Capítulo 4. Este sistema especializa o estágio da Composição do

nó mestre, que retira os *framebuffers* renderizados pelos nós escravos da fila de *framebuffers* descomprimidos e os compõe no dispositivo de visualização utilizado. Como nas arquiteturas com ordenação no início o problema da visibilidade é resolvido no momento em que a tela é dividida em ladrilhos, o estágio de Composição nessa arquitetura é bastante simples: basta desenhar os *framebuffers* recebidos dos nós escravos. Uma foto de tela do protótipo desse sistema encontra-se na Figura 3.5

Figura 3.5: Foto de tela do sistema com ordenação no início implementado. A cena em questão possui diversas plataformas de petróleo, que são compostas por milhões de polígonos.

Foi implementada uma arquitetura híbrida com ordenação no início e no fim que será descrita em maior detalhe no Capítulo 4. Esta arquitetura também utiliza o modelo 3D replicado nos nós do agrupamento. A cada quadro é calculada uma divisão da tela em ladrilhos de igual carga, utilizando-se o mesmo algoritmo da arquitetura com ordenação no início. Cada nó escravo, assim como na arquitetura com ordenação no início, descarta a parte do modelo que está fora do volume de visão determinado pelo ladrilho que está desenhando. Então, ele lê a cor da parte da tela em que desenhou objetos e a profundidade das áreas da tela em que há interseção entre as partes do modelo desenhadas por diferentes nós escravos. O estágio

PUC-Rio - Certificação Digital Nº 0310807/CA

de Composição na arquitetura híbrida é mais complexo: os servidores de renderização enviam *framebuffers* contendo informações de cor e de profundidade (*Z-Buffer*), que são desenhados com o teste de profundidade ligado. Na Figura 3.6 encontra-se uma foto de tela do protótipo implementado.

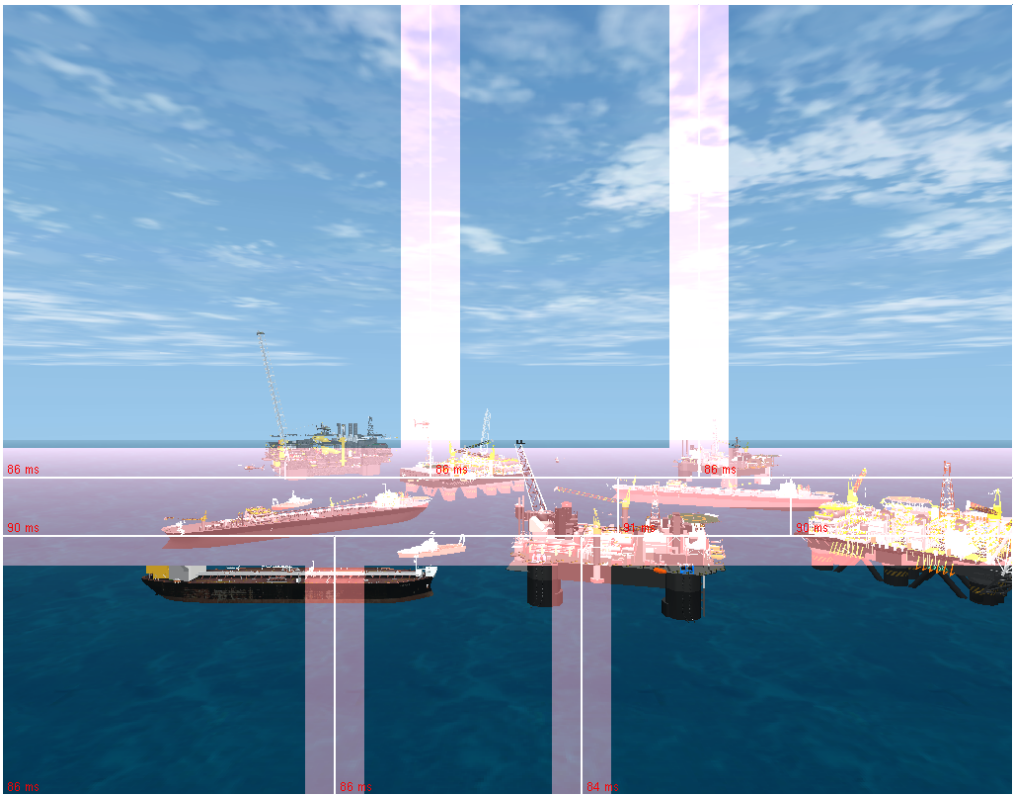

Figura 3.6: Foto de tela do sistema híbrido implementado. Em vermelho, as áreas em que foram transmitidas informações de visibilidade.

Um sistema de renderização volumétrica distribuída com ordenação no fim também foi implementado utilizando a linha de produção descrita na seção anterior. Hoje em dia muitos sistemas de renderização volumétrica codificam os dados volumétricos em texturas 3D. Como a quantidade de memória nas placas gráficas atuais é limitada, o tamanho dos dados também é bastante limitado. Na tentativa de possibilitar a visualização interativa de dados maiores, foi feita uma implementação que divide o dado volumétrico em sub-volumes de igual tamanho. Cada nó escravo do agrupamento fica responsável pela renderização de um sub-volume. O nó mestre calcula o espaço de tela que cada sub-volume ocupará e empacota esses dados nas requisições de renderização a serem enviadas para os nós escravos. Estes renderizam seus sub-volumes e lêem as componentes de cor e *alpha* do *framebuffer*. A componente *alpha* é necessária para a composição *alpha-blending* que será feita no nó mestre. A técnica de renderização de volumes

PUC-Rio - Certificação Digital Nº 0310807/CA

utiliza o operador *over* [4, 10] para a composição de camadas da imagem, tornando a composição dos resultados parciais bastante simples: o estágio de Composição utiliza esse operador, desenhando os *framebuffers* parciais na ordem correta (de trás para a frente) com o *alpha-blending* ligado. Uma foto de tela dessa implementação encontra-se na Figura 3.7.

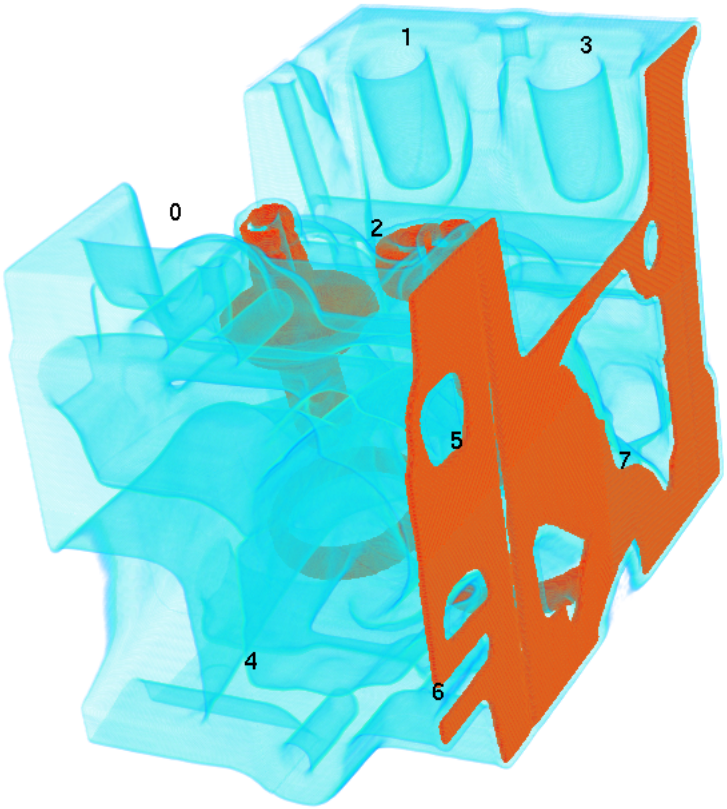

Figura 3.7: Foto de tela do sistema de renderização volumétrica distribuída implementado. Os números representam os índices dos sub-volumes em questão.

PUC-Rio - Certificação Digital Nº 0310807/CA