RENDERIZADOR 3D PARA APLICAÇÕES GRÁFICAS UTILIZANDO VULKAN

Daniel Streck

Prof. Dalton Solano dos Reis - Orientador

1 INTRODUÇÃO

Pode-se afirmar que para criar a ilusão de imagem em movimento em vídeos ou aplicações gráficas, a taxa de quadros para que uma aplicação possa ser denominada de tempo-real começa a partir dos 15 quadros por segundo. Portanto há necessidade de garantir alta performance para que aplicações interativas se tornem agradáveis para os usuários (AKENINE-MÖLLER, HAINES, HOFFMAN, 2008, p.1). Para atingir altos níveis de performance APIs (Application Programming Interface) são comumente empregadas para programação de aplicações que utilizam o hardware com recursos de computação gráfica.

Uma das entidades que mantém tais APIs é o grupo Khronos, que tem como objetivo criar e manter especificações abertas de APIs para computação paralela, computação gráfica, mídias dinâmicas, visão computacional e processamento de sensores em uma gama variada de plataformas (KHRONOS, 2016c). Algumas das especificações mantidas pelo grupo são: o OpenGL (Open Graphics Library) (uma API gráfica utilizada em plataformas *desktop* e *consoles*) e o OpenGL ES (OpenGL for Embedded Systems) para *mobile*, começaram a evoluir de maneira separada. Devido a isso, inciativa OpenGL começou a se fragmentar e evoluir independentemente, o que causou falta de conformidade e compatibilidade entre aplicações do OpenGL ES e OpenGL (KHRONOS, 2016c).

Devido a esta fragmentação, desenvolvedores começaram a apontar outras inconsistências e possíveis melhorias para a API. Isto resultou em uma iniciativa sem precedentes a partir de desenvolvedores proeminentes das industrias de desenvolvimento de jogos, tanto de hardware quanto software, para especificação do futuro do OpenGL que veio a se tornar o Vulkan (KHRONOS, 2016c).

Algumas das vantagens propostas pela especificação do Vulkan em relação às APIs anteriores são a uniformidade entre plataformas, suporte ao envio de comandos assíncronos da CPU para GPU e a disponibilização explicita de suas funções com pouco *overhead*. Estas mudanças transferem maior responsabilidade para o desenvolvedor, para então utilizá-las de forma que atenda melhor a sua aplicação. Através dessas funções possibilita-se que API se comporte de maneira extremamente previsível (SAMSUNG, 2016).

Diante do exposto, se propõe um estudo exploratório da API Vulkan, realizando-se uma implementação de uma biblioteca para renderização de cenas 3D otimizadas para jogos digitais e análise da performance obtida em comparação com a API OpenGL.

1.1 OBJETIVOS

O objetivo deste trabalho é realizar um estudo exploratório da API Vulkan.

Os objetivos específicos são:

a) desenvolver uma biblioteca para renderização 3D utilizando Vulkan;
b) realizar uma comparação da biblioteca desenvolvida com Vulkan a uma análoga desenvolvida com OpenGL;
c) analisar a performance atingida nos testes propostos.

2 TRABALHOS CORRELATOS

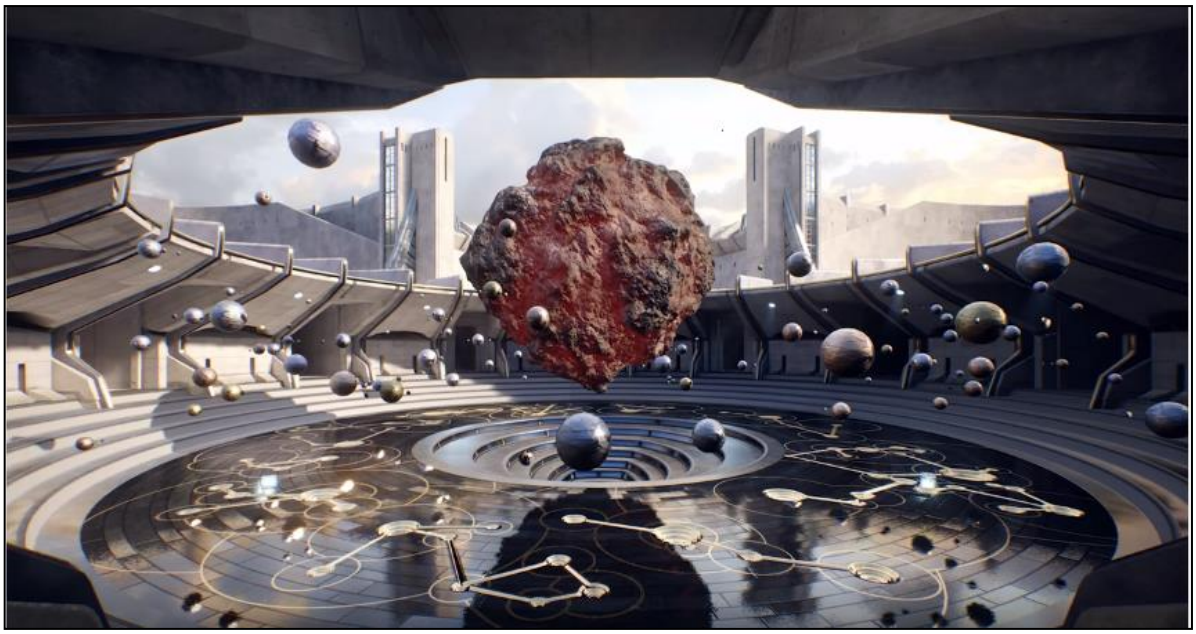
Serão introduzidos um produto e um trabalho acadêmico que implementam o objeto de estudo a ser explorado por este trabalho. O primeiro é o motor para aplicações multimídia com foco em jogos digitais Unreal Engine (EPIC GAMES, 2016b). O segundo é um trabalho que realiza um estudo exploratório de um renderizador 3D de alta performance, Vulkan Based Render Toolkit (MAINUŠ, 2016).

2.1 UNREAL ENGINE

A Unreal Engine é um conjunto de ferramentas para desenvolvimento de aplicações multimídia com alta fidelidade gráfica e alta performance mantidas pela EPIC Games (EPIC GAMES, 2016c). Foi criado em 1998 e foi originalmente apresentado ao público com o jogo digital Unreal (BLESZINSKI, 2010) e a partir da versão 4 a utilização do motor se tornou gratuita e com código aberto, sendo cobrados apenas 5% da receita a partir de 3 mil dólares (EPIC GAMES, 2016a).

A EPIC Games faz parte do grupo Khronos e participou ativamente na especificação da API Vulkan e portanto foi uma das primeiras *engines* a implementar o seu *renderer backend* utilizando Vulkan (SAMSUNG, 2016). A implementação da API Vulkan foi introduzida de forma experimental na versão 4.12 da Unreal Engine, permitindo assim a implementação de aplicações multimídia com suporte à Vulkan (EPIC GAMES, 2016b).

Figura 1 – Captura de tela da aplicação ProtoStar



Fonte: Epic Games (2016c).

Em conjunto com a Samsung, a Epic Games desenvolveu uma aplicação de demonstração para o Samsung Galaxy S7 chamada ProtoStar utilizando Vulkan, com o objetivo de demonstrar o potencial da API em plataformas mobile. Para atingir altos níveis de fidelidade gráfica que eles tinham como objetivo nessa aplicação foram implementadas diversas técnicas de computação gráfica como (EPIC GAMES, 2016a):

a) Reflexos planares dinâmicos (reflexos de alta qualidade para objetos dinâmicos);
b) Simulação de partículas na GPU;
c) *Temporal Anti-Aliasing* (TAA);
d) Compressão de texturas de alta qualidade;
e) *Chromatic aberration*;
f) Refração dinâmica de luz para plataformas *mobile*;
g) Suporte a cenas com milhares de objetos dinâmicos.

Um dos recursos disponíveis chama-se *command buffers*, o qual é um objeto utilizado para gravar comandos que podem subsequentemente ser enviados para a fila de um dispositivo para execução (KHRONOS, 2016c). Para manter a taxa de quadros aceitável no dispositivo Samsung Galaxy S7 a equipe da Unreal Engine empregou algumas técnicas, como a utilização de somente 3 grandes *command buffers*, sendo utilizado somente um por frame, e alternando-se entre eles utilizando a técnica de Round Robin.

Akenine-Möller, Haines e Hoffman (2008, p.711, tradução nossa) define GPU *instancing* como, o conceito de desenhar um objeto várias vezes com somente um *draw call* (comando de desenho). Foram apontadas também algumas das vantagens da reutilização dos *command buffers*, como em um caso de otimização simples o GPU *instancing*, e num caso de uso ótimo a renderização estereoscópica para Realidade Virtual (RV) (ARM, 2016).

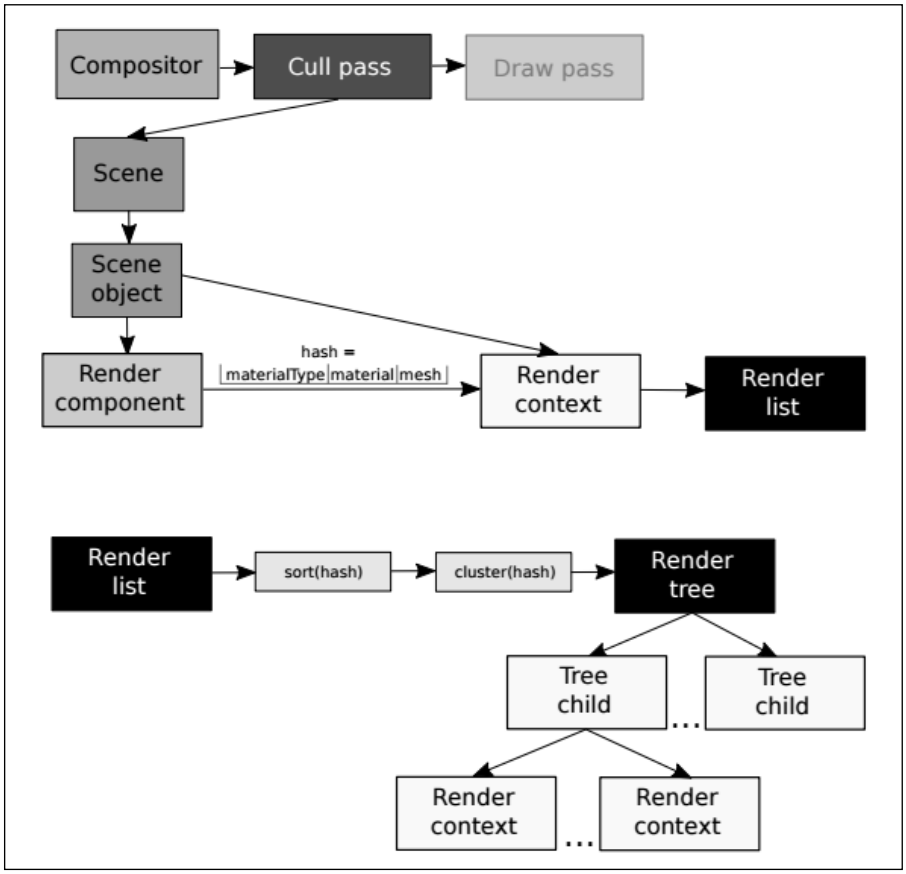
De acordo com Khronos Group (2016e, tradução nossa), Semáforos são utilizados para coordenar operações internas ou com filas externas a uma fila de comandos. *fences* podem ser utilizadas pelo dispositivo hospedeiro para determinar a completude da execução de uma fila de comandos. Para realizar a sincronização do envio dos comandos para a GPU, se utilizou Semáforos para a sincronização e ordenação de comandos na GPU, e *fences* para a sincronização na CPU, principalmente para verificação se um comando arbitrário já foi completado pela GPU. Em conclusão os autores afirmam ter atingido níveis de fidelidade gráfica e performance no mesmo nível de *consoles* da sétima geração (SAMSUNG, 2016).

2.2 VULKAN BASED RENDER TOOLKIT

Este trabalho tem como objetivo realizar um estudo exploratório da API Vulkan demonstrando as novas funções disponíveis na mesma. Para tal o autor desenvolveu em C++ um *render toolkit* e uma aplicação para análise de performance (MAINUŠ 2016).

O render toolkit é formado por um módulo que trata do ciclo de vida da aplicação e o gerenciamento de eventos (denominado *core*), e um módulo para renderização. De acordo com Mainuš (2016, tradução nossa), o módulo de renderização cria o objeto "*compositor*", que é responsável por criar os "*render passes*" e por percorrer a árvore de objetos gráficos presentes no grafo de cena de uma cena para fazer uma triagem para determinar quais objetos atendem algumas restrições especificas para serem ordenadas na árvore. A ordenação é feita levando em consideração objetos com contextos de renderização similares, ou seja com características como material e malhas iguais. Este processo está ilustrado na figura 2.

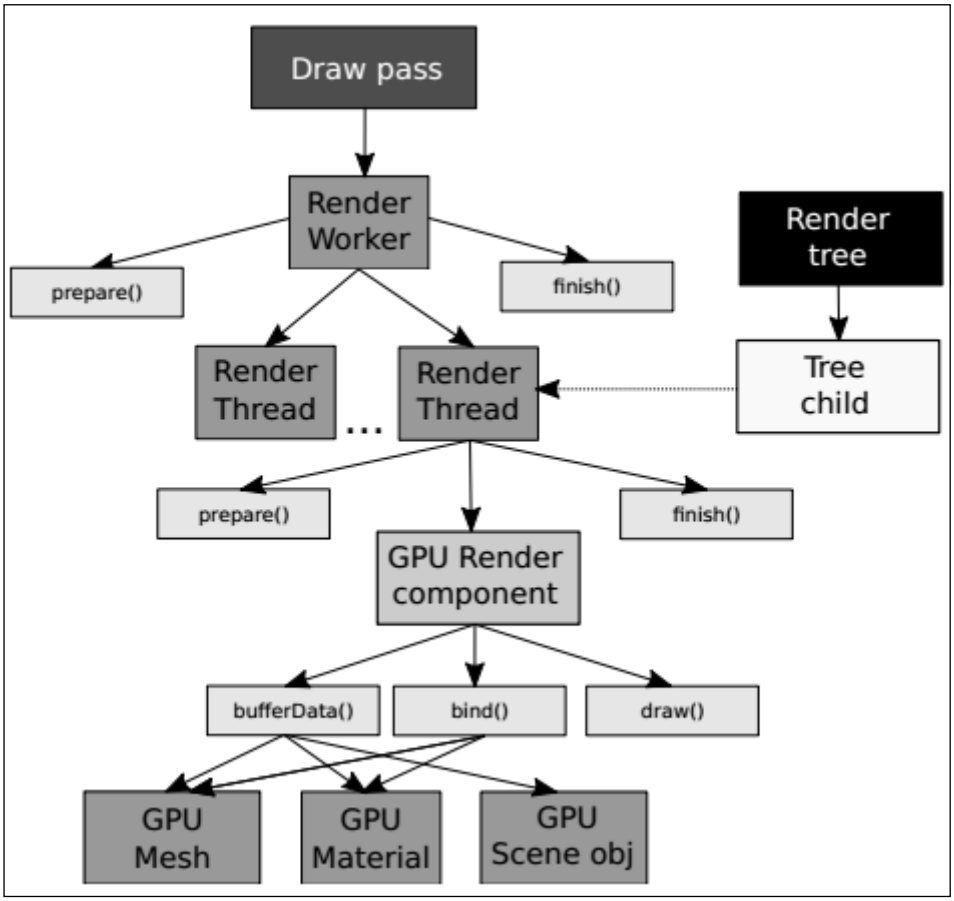
Figura 2 – Processo de criação da árvore de renderização (*Render tree*)



Fonte: Mainuš (2016).

Após o processo ilustrado na Figura 2 a "Render tree" é delegada para o "*Render worker*" que por sua vez divide as tarefas de renderização de maneira assíncrona através de diversas *threads*. Inicialmente cada *render component* tem os dados de inicialização preparados. Em seguida os atributos referentes a um *render component* são enviados para a memória da GPU. Na próxima etapa as informações de malha referentes aos materiais que serão utilizados pelos objetos são atribuídas ao pipeline para poderem ser reutilizados caso uma malha seja repetida (GPU *instancing*). Por último, dados referentes à cena são agregados ao pipeline. Este processo está ilustrado na Figura 3 (MAINUŠ, 2016).

Figura 3 – Processo de renderização da *Render tree*



Fonte: Mainuš (2016).

Para verificação de funcionalidades o autor criou uma aplicação utilizando o *render toolkit* e realizou testes divididos em 5 níveis de otimização, sendo eles:

a) no primeiro nível não é realizado nenhum tipo de otimização;
b) no segundo nível os trabalhos de renderização foram paralelizados com 5 *Threads*;
c) no terceiro nível adicionou-se a técnica de utilização de *staging buffers*;
d) no quarto nível incorpora-se ao segundo nível o recurso de *memory pools* para pré-alocar e reutilizar os recursos;
e) o quinto nível implementa todas as otimizações propostas anteriormente mas faz com que ocorram o mínimo possível de trocas de estados do *pipeline*.

Nas tabelas 4 e 5 pode-se observar respectivamente o tempo de cada frame na CPU e GPU em relação à quantidade de objetos presentes na cena.

Tabela 4 – Tempo (milissegundos) de um frame na CPU (Central Processing Unit) em relação à quantidade de objetos

	1k	2k	3k	4k	5k	6k	7k
Level 0	6.2	9.1	14.8	16.7			
Level 1	4.9	6.2	11.6	13.3	13.1	16.7	
Level 2	6.5	7.4	27.9	49.7	46.9	42.0	37.0
Level 3	6.7	7.6	8.2	13.2	16.3	18.0	19.9
Level 4	8.8	8.7	11.9	16.9	19.8	22.9	25.3

Fonte: Mainuš (2016).

Tabela 5 – Tempo (milissegundos) de um frame na GPU (Graphics Processing Unit) em relação à quantidade de objetos

	1k	2k	3k	4k	5k	6k	7k
Level 0	48.3	61.0	68.1	66.3			
Level 1	49.4	61.2	67.8	64.3	63.7	63.2	
Level 2	5.7	7.7	9.7	10.5	11.7	13.1	14.0
Level 3	5.5	7.5	9.3	10.0	10.6	11.0	11.9
Level 4	5.2	7.0	8.6	9.4	10.0	10.8	11.4

Fonte: Mainuš (2016).

3 PROPOSTA DA BIBLIOTECA

Neste capítulo será apresentada a justificativa da biblioteca proposta para se realizar o estudo em questão. Um quadro comparativo entre os trabalhos correlatos será apresentado, o qual será discutido textualmente em seguida. Após será discutida a relevância do trabalho proposto e em seguida as contribuições que ela pode trazer. Na sequência, serão informados os requisitos para a implementação da biblioteca e pôr fim a metodologia proposta.

3.1 JUSTIFICATIVA

Conforme análise apresentada (Quadro 1), ambos trabalhos correlatos foram realizados utilizando a API Vulkan, ambos também implementam a técnica de reutilização de c*ommand buffers* previamente construídos para envio de comandos através da API para a GPU.

Quadro 1 – Comparativo entre os trabalhos correlatos

Características	Epic Games (2016a)	Mainuš (2016)
suporte à Vulkan	sim	sim
reutilização de *command buffers*	sim	sim
suporte à materiais	sim	sim
occlusion culling	sim	não
multi-threaded command submission	sim	sim

Fonte: elaborado pelo autor.

Ambos os trabalhos implementam o conceito de material. A Unreal Engine implementa a técnica de otimização *occlusion culling*, mas Vulkan Based Render Toolkit não. Observa-se também que ambos utilizam técnicas de programação concorrente para a criação de c*ommand buffers* (*Multi-threaded command submission*).

Devido à natureza recente do objeto de estudo a ser explorado, sendo o acesso à API Vulkan disponibilizado em fevereiro de 2016, ainda não foram realizados muitos estudos acadêmicos nem *softwares* desenvolvidos com os fins similares aos que este trabalho propõe. Portanto pretende-se através deste estudo explorar as técnicas de implementação de uma biblioteca para renderização de cenas em 3D de alta performance otimizada para jogos digitais, abrindo a possibilidade de trabalhos futuros abordarem outros aspectos além daqueles que são foco do trabalho proposto.

Este trabalho deverá trazer contribuições práticas para o tema e principalmente para futuras implementações de bibliotecas e aplicações que tiverem como objeto de estudo a API Vulkan. Devido à natureza exploratória do trabalho aqui proposto, o mesmo deve apontar detalhes de implementação que são adequados para os casos de uso que serão estudados.

3.2 REQUISITOS PRINCIPAIS DA BIBLIOTECA PROPOSTA

A biblioteca proposta deve permitir:

a) renderizar cenas compostas por diversos objetos gráficos em 3D (Requisito Funcional - RF);
b) ser implementada utilizando o pipeline gráfico programável para aplicação de *shaders* (RF);

c) ser implementada utilizando programação concorrente para enviar comandos para a GPU (RF);
d) implementar a técnica de *occlusion culling* (RF);
e) utilizar a API Vulkan (Requisito Não Funcional - RNF);
f) ser implementada utilizando a linguagem C++ (RNF);
g) utilizar a biblioteca OpenGL Mathmatics (GLM) para funções matemáticas (RNF).

3.3 METODOLOGIA

O trabalho será desenvolvido observando as seguintes etapas:

a) levantamento bibliográfico: buscar fontes bibliográficas relacionadas ao domínio do estudo a ser realizado, como fundamentação de Computação Gráfica, especificações da API Vulkan e trabalhos correlatos;
b) elicitação de requisitos: nesta etapa será realizado o refinamento dos requisitos, funcionais e não-funcionais, necessários para atender o escopo dos objetivos deste estudo proposto;
c) modelagem da biblioteca: esta etapa consiste da modelagem de classes da biblioteca proposta de acordo com os padrões da Unified Modeling Language™ (UML®), utilizando a ferramenta Enterprise Architect.
d) implementação: processo de implementação efetiva dos requisitos levantados e da modelagem realizada da biblioteca. A implementação será realizada em C++ utilizando a IDE Microsoft Visual Studio;
e) testes de validação: serão realizados testes unitários para verificar a conformidade da implementação realizada com o que foi proposto nos requisitos e modelagem;
f) análise dos resultados: nesta etapa será realizada a análise dos resultados de performance obtidos pela biblioteca proposta.

As etapas serão realizadas nos períodos relacionados no Quadro 2.

Quadro 2 – Cronograma

etapas / quinzenas	2017									
	fev.		mar.		abr.		maio		jun.	
	1	2	1	2	1	2	1	2	1	2
levantamento bibliográfico										
elicitação de requisitos										
modelagem da biblioteca										
implementação										
testes de validação										
análise dos resultados										

Fonte: elaborado pelo autor.

4 REVISÃO BIBLIOGRÁFICA

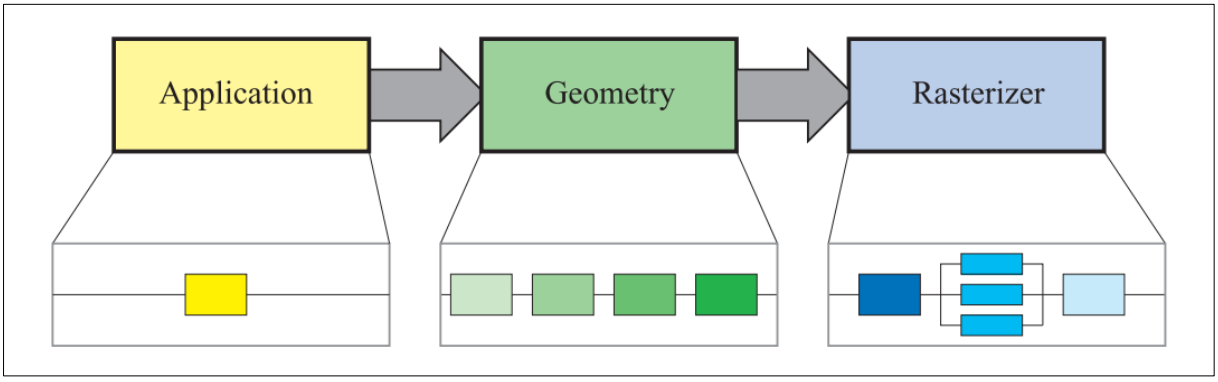
Este capítulo aborda assuntos tais como: fundamentação em Computação Gráfica, API Vulkan, API OpenGL e a biblioteca OpenGL Mathmatics (GLM).

4.1 FUNDAMENTAÇÃO EM COMPUTAÇÃO GRÁFICA

Para se gerar imagens através da Computação Gráfica, faz-se necessária a utilização de dispositivos de hardware com recurso do *pipeline* gráfico, geralmente GPUs (*Graphics Processing Unit*). Tais dispositivos podem receber comandos para executar diversas operações computacionais, que podem ter por resultado imagens rasterizadas ou simplesmente produtos computacionais.

Um *pipeline* gráfico de renderização tem como produto final uma imagem bidimensional de forma rasterizada. Para tal, diversos estágios são empregados (Figura 4), alguns são fixos e outros são programáveis ou configuráveis.

Figura 4: Estágios do *pipeline* gráfico



Fonte: Akenine-Möller, Haines e Hoffman (2008).

O *pipeline* é ativado com um *draw call*, que de acordo com Akenine-Möller, Haines e Hoffman (2008, p.31, tradução nossa), é uma chamada para a API gráfica para desenhar um

grupo de primitivas, causando a execução do *pipeline* gráfico. Após o comando de execução do *pipeline*, o primeiro estágio a ser executado é o estágio de geometria. Ele recebe os primitivas de renderização (pontos, linhas e triângulos) enviados a partir da aplicação, e é responsável pelas operações que ocorrem a cada vértice e polígono (AKENINE-MÖLLER, HAINES E HOFFMAN, 2008). Durante este estágio ocorrem algumas etapas que podem ser consideradas estágios separados ou não, que dependendo da implementação do hardware, podem ocorrer de forma paralela. Primeiramente ocorre a etapa de transformações geométricas, na qual as primitivas enviadas pela aplicação são transformadas para espaço global e espaço de câmera sintética respectivamente. Em seguida, de acordo Akenine-Möller, Haines e Hoffman (2008, p.17, tradução nossa) na etapa de *vertex shading* são executadas equações de *shading* que tem o propósito de definir o aspecto visual de um objeto através de atributos disponíveis por vértice, como sua localização no espaço universal, vetor normal, cores ou informações personalizadas. O produto dessa etapa é posteriormente utilizado com entrada para o estágio de rasterização. Após o *shading*, a próxima etapa é a de projeção, na qual o volume de visão da câmera sintética é transformado em projeção ortográfica ou perspectiva. Em seguida ocorre a etapa de *clipping*. Nesta etapa são descartados os objetos primitivas que não estão inteira ou parcialmente dentro do volume de visão da câmera sintética, e portando não precisam ser desenhados. Os objetos que estão parcialmente dentro do volume de visão passam pela operação de *clipping*, na qual novos vértices precisam ser determinados para as partes de objetos que estão parcialmente no volume de visão. A última etapa do estágio de geometria é o mapeamento para espaço de tela, na qual os primitivas, ainda representados em coordenadas tridimensionais, são convertidos para espaço de tela em duas dimensões.

O próximo estágio do *pipeline* gráfico é o de rasterização que tem por finalidade converter os dados providos pelo estágio de geometria para *pixels* e determinar seus respectivos valores de cor (AKENINE-MÖLLER, HAINES E HOFFMAN, 2008). Este estágio é constituído também por diversas etapas que são: *triangle setup*, uma etapa não-programável, na qual dados que posteriormente serão utilizados para realizar o processo de *triangle traversal* são gerados e interpolados a partir dos dados providos pelo estágio de geometria. Após, no estágio de *triangle traversal* os *pixels* que possuem seu centro contido por um triângulo são marcados e um *fragment* é criado para eles. Em seguida ocorre o estágio de *pixel shading*. Neste estágio programável, que recebe como entrada dos dados das etapas anteriores interpolados, são executadas equações de *shading* nos *fragments*, que produzirão valores de cor para os *pixels* que serão encaminhados para etapas posteriores do *pipeline*. Nesta etapa são aplicadas técnicas como mapeamento de textura em um objeto gráfico. A última etapa do estágio de rasterização

é denominada *merging* e nela os fragmentos providos pela etapa anterior são combinados com os pixels presentes no *Color buffer* (um *array* retangular de valores de cor dos *pixels* a serem desenhados na tela). Nesta etapa ocorre também a resolução de visibilidade. Para tal geralmente se emprega o *Z-Buffer* (ou *depth buffer*), que é um *array* com as mesmas dimensões do *color buffer* e guarda o valor de coordenada Z de *pixel* em relação a câmera sintética.

O conceito de material é comumente empregado em aplicações gráficas, que de acordo com Akenine-Möller, Haines e Hoffman (2008, p.104, tradução nossa) tem a seguinte definição: a aparência de um objeto gráfico é representada por agregar materiais à modelos na cena. Cada material é associado com conjunto de *shaders*, texturas e outras propriedades para simular a interação de luz com objetos. Outra técnica muito empregada em renderizadores gráficos é o *occlusion culling*, definido por Akenine-Möller, Haines e Hoffman (2008, p.671, tradução nossa) como uma técnica de otimização utilizada para não renderizar objetos em uma cena que estão obstruídos por outros.

4.2 VULKAN

Vulkan é uma API para desenvolvimento de aplicações gráficas aceleradas por *hardware* mantida pelo grupo Khronos. Sua concepção se deu pela necessidade de uma API gráfica que atendesse melhor o ecossistema moderno do mercado de aplicações gráficas. Especificado em conjunto com os líderes da indústria da computação gráfica, para ser o novo padrão de API gráficas com especificação aberta (KHRONOS, 2016c).

Em sua concepção Vulkan é mais explícito do que seu antecessor (OpenGL), em que o desenvolvedor tem maior responsabilidade na utilização dos recursos de hardware, ao invés de haver certos comandos abstraídos nos *drivers* ou em comandos fixos disponíveis pela API. Como resultado, o Vulkan pode ser utilizado de forma que se adapta melhor a aplicação, possivelmente assim, rendendo maior desempenho (KHRONOS, 2016c).

Alguns benefícios da especificação do Vulkan são comentados de acordo com Khronos (2016c, tradução nossa): introdução do conceito de *command queue*, fila de *command buffers* para serem enviados para o dispositivo host (GPU – *Graphics Processing Unit*). A criação das *command queues* pode ser feita de forma paralela e assíncrona utilizando-se diversas *threads* o que otimiza a utilização da CPU (*Central Processing Unit*). Outra vantagem proposta em relação ao OpenGL é que especificação do Vulkan permite que aplicações sejam desenvolvidas para múltiplas plataformas sem que haja necessidade de alterações específicas para cada uma.

Standard Portable Intermediate Representation - V (SPIR-V) é uma linguagem intermediária desenvolvida pelo grupo Khronos para representação nativa de *shaders* gráficos e *kernels* computacionais. Permite que *shaders* possam ser compilados para formato binário antes da execução, o que gera os alguns benefícios como: depuração antes do tempo de execução, maior otimização e proteção de propriedade intelectual. Os programas (*shaders*) podem ser escritos utilizando a mesma linguagem utilizada no OpenGL, o OpenGL *Shading Language* (GLSL), mas pode-se também utilizar-se do SDK para desenvolver um compilador para linguagens de terceiros que ao final precisam estar no formato binário do SPIR-V (KHRONOS, 2016a).

O conceito de *Validation Layers* foi introduzido no *Sofware Development Kit* (SDK) do Vulkan para auxiliar no processo de depuração e verificação de conformidade com a especificação no desenvolvimento de aplicações que utilizam o mesmo (KHRONOS, 2016c). As *Validation Layers* podem ser empregadas durante o desenvolvimento de uma aplicação para reportar a utilização de forma incorreta da API Vulkan. Fornece também recursos como árvore de chamadas de funções da API para análise e depuração. Uma das vantagens propostas por essa ferramenta é que ela pode ser desabilitada completamente da aplicação quando ela estiver com o desenvolvimento concluído. Portanto removendo assim qualquer tipo de *overhead* que poderia ser causado pela API realizando validações internas para verificar a conformidade e validade das chamadas de função.

Para apresentar os resultados de renderização para uma superfície ou tela utilizando-se Vulkan faz-se necessário a utilização do objeto *swapchain* (KHRONOS, 2016f). Este objeto, fornecido através de uma extensão, é uma abstração de um *array* com imagens associadas a uma tela prontas para apresentação. A aplicação renderiza uma imagem que resulta em um objeto VkImage, o qual é adicionado à fila do *swapchain* para futuramente ser apresentada pelo *presentation engine*, que é responsável por ordenar quais imagens podem ser adquiridas pela aplicação.

4.3 OPENGL E OPENGL MATHMATICS (GLM)

OpenGL é uma API para o desenvolvimento de aplicações gráficas introduzida originalmente em 1992, e tornou-se a API gráfica com especificação aberta mais utilizada em uma alta gama de aplicações gráficas (KHRONOS, 2016b). Em sua concepção OpenGL é agnóstico em relação à plataforma e linguagem de programação.

Em sua especificação OpenGL segue um modelo de estados globais, no qual qualquer objeto (*frame buffer, handle* de texturas) pode ser adquirido por uma aplicação em tempo de execução (KHRONOS, 2016b). O que torna a API mais acessível para desenvolvedores, uma vez que funções providas pela API, como transformações geométricas por exemplo, não precisam ser implementadas pelos mesmos.

OpenGL Mathmatics é uma biblioteca matemática em forma *header* de C++ para softwares gráficos baseados no OpenGL *Shading Language* (GLSL) (G-TRUC CREATION, 2016). São fornecidas na biblioteca funções matemáticas referentes a *quaternions*, transformações geométricas, matemática vetorial, entre outros.

REFERÊNCIAS

AKENINE-MÖLLER, Tomas; HAINES, Eric; HOFFMAN, Naty. **Real-Time Rendering.** 3. ed. Natick: A K Peters/CRC Press, 2008. 1045 p.

ARM. **Vulkan on Mobile.** San Francisco: Vulkan On Mobile, 2016. Color. Disponível em: <http://malideveloper.arm.com/downloads/Presentations/GDC 2016/Sponsored/Vulkan on Mobile with Unreal Engine 4 Case Study.pdf>. Acesso em: 10 set. 2016.

BLESZINSKI, Cliff. **HISTORY OF THE UNREAL ENGINE.** 2010. Disponível em: <http://www.ign.com/articles/2010/02/23/history-of-the-unreal-engine>. Acesso em: 02 set. 2016.

EPIC GAMES: **Protostar: Pushing Mobile Graphics with Unreal Engine 4 and Vulkan API – Mobile Congress 2016**. 2016. Disponível em: <https://www.youtube.com/watch?v=lIdNoSB69PI>. Acesso em: 10 set. 2016.

_____. **Vulkan:** How to use Vulkan in Unreal Engine 4. 2016. Disponível em: <https://wiki.unrealengine.com/Vulkan>. Acesso em: 03 set. 2016.

_____. **WHAT IS UNREAL ENGINE 4?** 2016. Disponível em: <https://www.unrealengine.com/what-is-unreal-engine-4>. Acesso em: 01 set. 2016.

G-TRUC CREATION. **OpenGL Mathematics**, 2016. Disponível em: <http://glm.g-truc.net/0.9.7/index.html>. Acesso em: 04 set. 2016.

KHRONOS GROUP. **An Introduction to SPIR-V**, 2016. Disponível em: <https://www.khronos.org/registry/spir-v/papers/WhitePaper.pdf>. Acesso em: 01 nov. 2016.

_____. **OpenGL Overview**, 2016. Disponível em: <https://www.opengl.org/about/>. Acesso em: 06 nov. 2016.

_____. **The Khronos Group Inc**, 2016. Disponível em: <https://www.khronos.org/>. Acesso em: 12 set. 2016.

_____. **Vulkan™ Overview February 2016**. Khronos Group, 2016. Color. Disponível em: <https://www.khronos.org/assets/uploads/developers/library/overview/vulkan-overview.pdf>. Acesso em: 11 set. 2016.

_____. **Vulkan 1.0.25 - A Specification**, 2016**.** Disponível em: <https://www.khronos.org/registry/vulkan/specs/1.0/xhtml/vkspec.html>. Acesso em: 01 set. 2016.

_____. **Vulkan 1.0.32 - A Specification (with KHR extensions)**, 2016. Disponível em: <https://www.khronos.org/registry/vulkan/specs/1.0-wsi_extensions/xhtml/vkspec.html#_wsi_swapchain>. Acesso em: 05 nov. 2016.

MAINUŠ, Matěj. Vulkan based render toolkit. In: EXCEL@FIT, 2016, Brno. **Excel@FIT 2016.** Brno: Brno University Of Technology, 2016. p. 1 - 5. Disponível em: <http://excel.fit.vutbr.cz/submissions/2016/040/40.pdf>. Acesso em: 01 set. 2016.

SAMSUNG (Org.). **SDC 2016 Session: Developing Console Games with Vulkan and Unreal.** Disponível em: <https://www.youtube.com/watch?v=NyGsMr2tcks>. Acesso em: 01 set. 2016.

ASSINATURAS

(Atenção: todas as folhas devem estar rubricadas)

Assinatura do(a) Aluno(a): _____

Assinatura do(a) Orientador(a): _____

Assinatura do(a) Coorientador(a) (se houver): _____

Observações do orientador em relação a itens não atendidos do pré-projeto (se houver):

FORMULÁRIO DE AVALIAÇÃO (PROJETO) – PROFESSOR TCC I

Acadêmico(a): _____

Avaliador(a): _____

	ASPECTOS AVALIADOS[1]	atende	atende parcialmente	não atende
ASPECTOS TÉCNICOS	1. INTRODUÇÃO O tema de pesquisa está devidamente contextualizado/delimitado?			
	O problema está claramente formulado?			
	2. OBJETIVOS O objetivo principal está claramente definido e é passível de ser alcançado?			
	Os objetivos específicos são coerentes com o objetivo principal?			
	3. TRABALHOS CORRELATOS São apresentados trabalhos correlatos, bem como descritas as principais funcionalidades e os pontos fortes e fracos?			
	4. JUSTIFICATIVA Foi apresentado e discutido um quadro relacionando os trabalhos correlatos e suas principais funcionalidades com a proposta apresentada?			
	São apresentados argumentos científicos, técnicos ou metodológicos que justificam a proposta?			
	São apresentadas as contribuições teóricas, práticas ou sociais que justificam a proposta?			
	5. REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO Os requisitos funcionais e não funcionais foram claramente descritos?			
	6. METODOLOGIA Foram relacionadas todas as etapas necessárias para o desenvolvimento do TCC?			
	Os métodos, recursos e o cronograma estão devidamente apresentados e são compatíveis com a metodologia proposta?			
	7. REVISÃO BIBLIOGRÁFICA Os assuntos apresentados são suficientes e têm relação com o tema do TCC?			
	As referências contemplam adequadamente os assuntos abordados (são indicadas obras atualizadas e as mais importantes da área)?			
ASPECTOS METODOLÓGICOS	8. LINGUAGEM USADA (redação) O texto completo é coerente e redigido corretamente em língua portuguesa, usando linguagem formal/científica?			
	A exposição do assunto é ordenada (as ideias estão bem encadeadas e a linguagem utilizada é clara)?			
	9. ORGANIZAÇÃO E APRESENTAÇÃO GRÁFICA DO TEXTO A organização e apresentação dos capítulos, seções, subseções e parágrafos estão de acordo com o modelo estabelecido?			
	10. ILUSTRAÇÕES (figuras, quadros, tabelas) As ilustrações são legíveis e obedecem às normas da ABNT?			
	11. REFERÊNCIAS E CITAÇÕES As referências obedecem às normas da ABNT?			
	As citações obedecem às normas da ABNT?			
	Todos os documentos citados foram referenciados e vice-versa, isto é, as citações e referências são consistentes?			

PARECER – PROFESSOR DE TCC I OU COORDENADOR DE TCC:

O projeto de TCC será reprovado se:

- qualquer um dos itens tiver resposta NÃO ATENDE;
- pelo menos **4 (quatro)** itens dos **ASPECTOS TÉCNICOS** tiverem resposta ATENDE PARCIALMENTE; ou
- pelo menos **4 (quatro)** itens dos **ASPECTOS METODOLÓGICOS** tiverem resposta ATENDE PARCIALMENTE.

PARECER: ( ) APROVADO ( ) REPROVADO

Assinatura: _____ Data: _____

[1] Quando o avaliador marcar algum item como atende parcialmente ou não atende, deve obrigatoriamente indicar os motivos no texto, para que o aluno saiba o porquê da avaliação.

FORMULÁRIO DE AVALIAÇÃO (PROJETO) – PROFESSOR AVALIADOR

Acadêmico(a): _____

Avaliador(a): _____

	ASPECTOS AVALIADOS[1]	atende	atende parcialmente	não atende
ASPECTOS TÉCNICOS	1. INTRODUÇÃO O tema de pesquisa está devidamente contextualizado/delimitado?			
	O problema está claramente formulado?			
	2. OBJETIVOS O objetivo principal está claramente definido e é passível de ser alcançado?			
	Os objetivos específicos são coerentes com o objetivo principal?			
	3. TRABALHOS CORRELATOS São apresentados trabalhos correlatos, bem como descritas as principais funcionalidades e os pontos fortes e fracos?			
	4. JUSTIFICATIVA Foi apresentado e discutido um quadro relacionando os trabalhos correlatos e suas principais funcionalidades com a proposta apresentada?			
	São apresentados argumentos científicos, técnicos ou metodológicos que justificam a proposta?			
	São apresentadas as contribuições teóricas, práticas ou sociais que justificam a proposta?			
	5. REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO Os requisitos funcionais e não funcionais foram claramente descritos?			
	6. METODOLOGIA Foram relacionadas todas as etapas necessárias para o desenvolvimento do TCC?			
	Os métodos, recursos e o cronograma estão devidamente apresentados e são compatíveis com a metodologia proposta?			
	7. REVISÃO BIBLIOGRÁFICA Os assuntos apresentados são suficientes e têm relação com o tema do TCC?			
	As referências contemplam adequadamente os assuntos abordados (são indicadas obras atualizadas e as mais importantes da área)?			
ASPECTOS METODOLÓGICOS	8. LINGUAGEM USADA (redação) O texto completo é coerente e redigido corretamente em língua portuguesa, usando linguagem formal/científica?			
	A exposição do assunto é ordenada (as ideias estão bem encadeadas e a linguagem utilizada é clara)?			

PARECER – PROFESSOR AVALIADOR:

O projeto de TCC serdeverá ser revisado, isto é, necessita de complementação, se:

- qualquer um dos itens tiver resposta NÃO ATENDE;
- pelo menos **5 (cinco)** tiverem resposta ATENDE PARCIALMENTE.

PARECER: ( ) APROVADO ( ) REPROVADO

Assinatura: _____ Data: _____

[1] Quando o avaliador marcar algum item como atende parcialmente ou não atende, deve obrigatoriamente indicar os motivos no texto, para que o aluno saiba o porquê da avaliação.