UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CÂMPUS GUARAPUAVA
CURSO DE TECNOLOGIA EM SISTEMAS PARA INTERNET

RAPHAEL KSIASKIEWCZ CZOVNY

MOBILE SHADER EDITOR: UM EDITOR DE SHADERS PARA ANDROID

PROJETO DO TRABALHO DE CONCLUSÃO DE CURSO

GUARAPUAVA

2015

RAPHAEL KSIASKIEWCZ CZOVNY

MOBILE SHADER EDITOR: UM EDITOR DE SHADERS PARA ANDROID

Projeto do Trabalho de Conclusão de Curso apresentado à disciplina Trabalho de Conclusão de Curso II do Curso de Tecnologia em Sistemas para Internet da Universidade Tecnológica Federal do Paraná como requisito parcial para aprovação.

Orientador: Prof. Me. Andres Jessé Porfirio

GUARAPUAVA

2015

RESUMO

CZOVNY, Raphael Ksiaskiewcz. Mobile Shader Editor: um editor de Shaders para Android. 31 f. Projeto do Trabalho de Conclusão de Curso – Curso de Tecnologia em Sistemas para Internet, Universidade Tecnológica Federal do Paraná. Guarapuava, 2015.

Com a evolução tecnológica, os jogos acabaram evoluindo e sendo otimizados da mesma forma, ganhando efeitos cada vez mais realistas, parte disso se da ao fato da utilização de *Shaders*, que são fragmentos de programas que executam na Unidade de processamento gráfico (GPU). Quando é realizado o desenvolvimento de *Shaders* para dispositivos móveis, por exemplo para o sistema operacional Android, são encontrados problemas, em especial para sua depuração e testes. Isso se dá ao fato de existir certa carência das ferramentas voltadas para esse meio e, principalmente, da existência de limitações da versão da OpenGL utilizada nestes dispositivos (OpenGL ES). Por conta disto, o trabalho em questão terá como foco o desenvolvimento de uma aplicação para programação e depuração de *Shaders* da linguagem GLSL (linguagem de programação específica para *Shaders* OpenGL), com intuito principal de permitir que o desenvolvimento deste tipo de recurso seja mais prático e rápido do que o utilizado atualmente na maioria das ferramentas.

Palavras-chave: Shaders, GLSL, OpenGL ES, Android

ABSTRACT

CZOVNY, Raphael Ksiaskiewcz. Mobile Shader Editor: an editor of Shaders for Android. 31 f. Projeto do Trabalho de Conclusão de Curso – Curso de Tecnologia em Sistemas para Internet, Universidade Tecnológica Federal do Paraná. Guarapuava, 2015.

With technological progress, the game ended up being evolved and optimized in the same way, gaining more and more realistic effects, part of it is the fact of the use of Shaders, which are fragments of programs that run on the GPU. When done developing Shaders for mobile devices, for example the Android operating system, in particular problems for their production and testing are found. This is so the fact that there a certain lack of tools geared towards this medium and, especially, the existence of limitations on the version of OpenGL used in these devices (OpenGL ES). Because of this, the work in question will focus on the development of an application for programming and debugging of the GLSL Shaders (specific programming language for OpenGL Shaders), with the main goal of allowing the development of this resource type is more practical and faster than currently used in most tools.

Keywords: Shaders, GLSL, OpenGL ES, Android

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

Jogos eletrônicos vêm se tornando bastante presentes no dia a dia das pessoas. Um dos principais fatores se deve à evolução tecnológica, que está tornando cada vez mais acessível a utilização de *smarthphones*, notebooks e videogames, desta forma, aproximando cada vez mais a sociedade destes meios de entretenimento digital. Com isso, o mercado de jogos eletrônicos cresceu muito nos últimos anos, a produção massiva de jogos atinge todos os gêneros e idades. Devido a isso, este ramo fatura até mais que a própria indústria cinematográfica e, segundo a revista Reuters (EVANS, 2013), é considerada a indústria que mais fatura em todo o mundo.

Por conta disso, o desenvolvimento deste tipo de entretenimento se torna muito atraente para muitas empresas, proporcionando bons retornos financeiros. No entanto, resulta em um desenvolvimento complexo e exigente, pelo fato de contar com a junção de inúmeros componentes em sua criação, desde a apresentação artística, até um enredo interessante, efeitos sonoros e gráficos.

Contudo, ao longo do tempo foram criadas ferramentas para auxiliar nesses procedimentos, como um tipo especial delas, se destacam os motores de jogos, que nada mais são do que pacotes de ferramentas para desenvolvimento deste tipo de software. Em geral, um motor de jogos possui sistemas de simulações físicas de colisões, gravidade, renderização 3D, efeitos sonoros e métodos para tratamento de entrada e saída como teclado, mouse, *gamepad*, entre outros.

Mesmo com a existência dessas ferramentas, que tendem a auxiliar a produção de jogos, sempre existem obstáculos a serem enfrentados pelo desenvolvedor. Essas plataformas, em geral, tentam oferecer recursos genéricos para as mais variadas finalidades, em caso específico, pode-se destacar a utilização de *Shaders*. Um *Shader* pode ser definido como um conjunto de instruções que definem o comportamento da superfície dos objetos, ou seja, são os efeitos perceptíveis nos jogos, desde reflexos de objetos, curvaturas, movimentações da água, entre outros (PASSOS et al., 2009). Estes recursos deixam os jogos mais parecidos com a nossa realidade, tentando imitar os efeitos naturais do cotidiano vistos no mundo real.

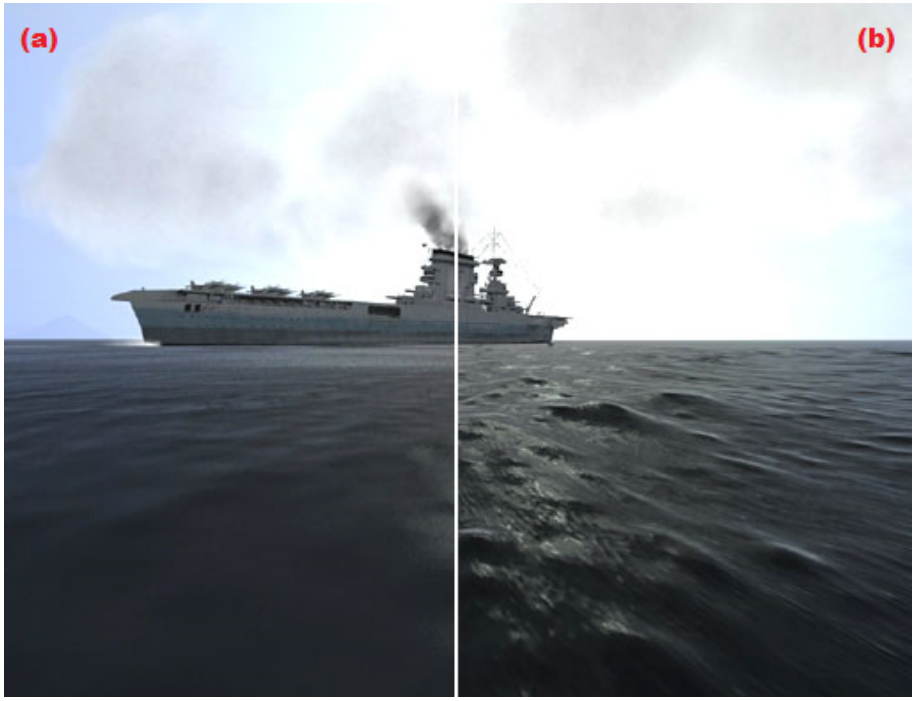

Figura 1: Exemplo da aplicação de Shaders em jogos.
Fonte: http://www.nvidia.com/object/usingvertextextures.html (NVIDIA, 2004)

Existem vários tipos de *Shaders*, como o *Vertex Shader*, *Fragment Shader* e *Geometry Shader*, que podem criar efeitos através de instruções. Um exemplo disso pode ser observado na Figura 1, que a partir da utilização de *Vertex Shaders* sobre um jogo (lado b), é possível observar as diferenças gráficas resultantes da aplicação do *Shader*. Onde a imagem da esquerda (a) não existe aplicação do *Shader*, assim destacando as mudanças gráficas depois de sua aplicação na imagem da direita (b).

Uma linguagem utilizada para criação de *Shaders* é a OpenGL Shading Language (GLSL), recurso da API OpenGL[1], geralmente utilizada para renderizações 3D em computadores e videogames, cujo poder de processamento pode ser considerado elevado em relação a determinados dispositivos. Aparelhos mais limitados como os dispositivos móveis fazem uso da API OpenGL ES, subconjunto da OpenGL.

Uma vez que se utilize recursos da OpenGL ES em dispositivos móveis, é necessário que se tenha meios de testar o resultado diretamente no aparelho, visto que a capacidade de processamento de um computador é superior e não reflete as limitações do dispositivo móvel. Um dos meios tradicionais de se realizar este teste é compilando o código especificamente para o aparelho desejado e, em seguida, após a instalação, observando seu resultado. Esse processo demanda tempo, desde a compilação, cópia do aplicativo para a memória do celular até sua

[1] API com recursos gráficos. Site oficial: www.opengl.org

execução.

Devido a esse problema, surge a dificuldade de realizar os testes dos códigos de *Shaders* diretamente no aparelho. Dessa forma, como objetivo desse trabalho, é proposto a criação de um *Mobile Shader Editor*, um editor de *Shaders* para Android, que será uma ferramenta de testes de programas GLSL. Com a depuração do código GLSL feita com o dispositivo móvel em conjunto com um PC, de modo que a escrita do *Shader* e a visualização dos possíveis registros de erros, sejam feitos no computador, sem a necessidade de recompilação do programa, mas apenas o recarregamento do *Shader* em cada teste.

O restante do texto é dividido conforme segue: O Subcapítulo 1.1 apresenta o objeto de estudo deste trabalho. O Subcapítulo 1.2 mostra os objetivos. No Subcapítulo 1.2 é demonstrado a justificativa sobre o tema. No Capítulo 2 é abordado o referencial teórico, com as tecnologias estudadas. No Capítulo 3 é apresentado o estado da arte, sobre as tecnologias correlatas. O Capítulo 4 relata a metodologia de desenvolvimento. O Capítulo 5 está o cronograma proposto para o trabalho.

1.1 OBJETO DE ESTUDO

O objeto de estudo deste trabalho constitui no desenvolvimento de uma aplicação contendo uma ferramenta com dois módulos, sendo o primeiro responsável pela escrita e depuração do código GLSL em um computador e o segundo dedicado ao teste deste código juntamente com visualização do seu resultado diretamente no dispositivo móvel.

O primeiro módulo consistirá em uma aplicação para *Desktop* representado em (a) na Figura 2, que será um editor de códigos em linguagem GLSL. Assim, possibilitando o desenvolvimento de *Shaders*, além deste desenvolvimento, a aplicação contará com os registros de erros emitidos pelo código sendo mostrados automaticamente. A partir disso, o segundo módulo será uma aplicação móvel representado em (b), com capacidade de execução de códigos GLSL e captura de informações de erro gerados pelo *Shader*.

A integração de ambas as ferramentas, que também pode ser observado na Figura 2, será feita através do ADB, uma ferramenta que possibilita o acesso do dispositivo através de um terminal, possibilitando integração entre o computador e o dispositivo móvel. Assim, através dele será possível enviar o código GLSL para memória do dispositivo e inclusive buscar os erros gerados para exibição no *Desktop*. Dessa forma, será possível a edição em tempo real do código GLSL, com sua visualização e depuração, sem a necessidade de recompilação da aplicação móvel.

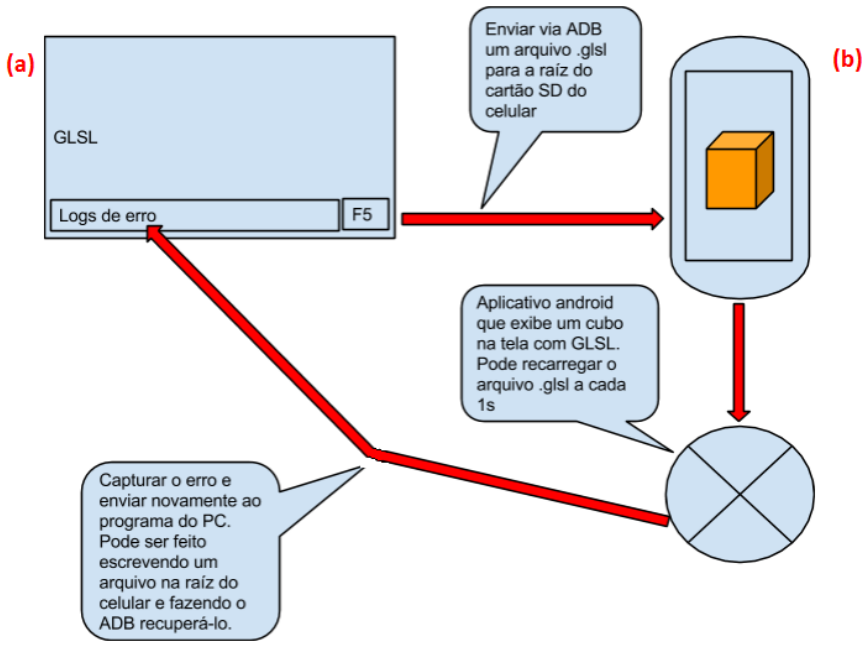

Figura 2: Fluxograma da ferramenta.
Fonte: O autor

1.2 OBJETIVOS

Nesta seção serão abordados os objetivos para concretização desse trabalho.

1.2.1 OBJETIVO GERAL

O presente trabalho tem como objetivo desenvolver uma ferramenta de programação e depuração de *Shaders* GLSL para dispositivos móveis na plataforma Android, possuindo a execução dos *Shaders* no aparelho e o registro de erros no computador, utilizando-se do ADB para comunicação entre ambos os *softwares*.

1.2.2 OBJETIVOS ESPECÍFICOS

São objetivos específicos desse trabalho:

- Pesquisar aplicações já existentes com suas capacidades e limitações;
- Construção de uma aplicação Android, capaz de ler um arquivo GLSL da memória do celular e executar o *Shader*;
- Construção de um programa para PC, capaz de fornecer uma interface para escrita de códigos GLSL e envio deste para memória do celular de modo imediato, através de ADB;

- Elaborar e aplicar uma metodologia de testes para validação da aplicação.

1.3 JUSTIFICATIVA

O mercado de celulares está crescendo cada vez mais. Lecheta (LECHETA, 2010) diz que hoje em dia mais de 3 bilhões de pessoas possuem um aparelho celular, correspondendo a cerca da metade da população mundial. Assim, devido à alta disponibilidade de dispositivos, a produção de *software* também cresceu. Também, os jogos eletrônicos se tornaram um objeto lucrativo e popular, de maneira que o mercado exige que este desenvolvimento faça uso de maneiras ágeis de produção e que tornem a criação destes meios de entretenimento mais simples.

Os consumidores de jogos eletrônicos estão cada vez mais exigentes, buscando mais interatividade e gráficos de qualidade. Um dos recursos que proporcionam a criação de efeitos visuais em jogos são os *Shaders*, que nada mais são do que pequenos programas que executam instruções relacionadas geralmente a imagens. Em dispositivos Android destaca-se o uso de *Shaders* GLSL, que é a linguagem de *Shaders* da OpenGL.

Quando se trata do desenvolvimento de efeitos gráficos para dispositivos móveis, são encontrados alguns problemas, existindo uma dificuldade de teste e depuração de programas GLSL. Isso ocorre, principalmente, pelo fato de que a GLSL possui especificações diferentes de acordo com a plataforma. Ao utilizar um computador pessoal, a linguagem de *Shaders* da OpenGL possui determinados recursos, no entanto para dispositivos móveis, ela utiliza de recursos da OpenGL ES, que é uma versão mais limitada do OpenGL, portanto nem todos os *Shaders* acabam sendo compatíveis. Existindo também a falta de suporte a *Geometry Shaders*, que realizam mudanças notáveis nos objetos, como pode ser observado na Figura 3, onde o lado esquerdo (a) não possui aplicação de *Geometry Shaders*, quando o mesmo é aplicado é notado as mudanças sobre o objeto no lado (b).

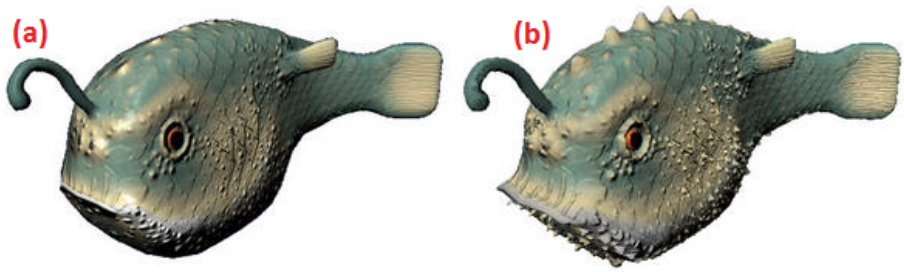

Figura 3: Exemplo da utilização de Geometry Shaders.
Fonte:
http://www.pc-infopratique.com/article-82-2-geforce-8800-la-gamme-complete-en-test.html

Os testes de *Shaders* tendem a ser demorados, pois é necessário a compilação, copia da aplicação para memória do celular, instalação da aplicação e, apenas no final, sua execução, para assim, ocorrer a avaliação de teste do *Shader*. Além disso, em algumas ferramentas se torna difícil detectar erros, por exemplo na ferramenta Unity, quando o *Shader* não compila, apenas a superfície fica com uma tonalidade rosa, não existindo um retorno claro ao desenvolvedor. Outro exemplo é na utilização da JPCT, que é uma engine gráfica 3D para implementação em linguagem Java, onde apenas é lançado uma exceção da OpenGL, sem muitos detalhes.

Através dessas dificuldades, é proposto o desenvolvimento de um editor de *Shaders*, capaz de permitir que o programador desenvolva o código GLSL em um PC e execute-o diretamente no celular sem a necessidade de recompilação do aplicativo móvel. Com isso, tendendo a favorecer o desenvolvimento de programas GLSL, com a possibilidade de realizar testes e depurações com maior agilidade.

2 REFERENCIAL TEÓRICO

Neste Capítulo será abordado o referencial teórico, detalhando sobre cada tecnologia estudada. Na Seção 2.1 é detalha-se sobre o Android. A Seção 2.2 refere-se ao referencial sobre Jogos. Na Seção 2.3 menciona-se sobre os *Shaders*. Na Seção 2.4 será abordado a OpenGL. A Seção 2.5 apresenta sobre a GLSL. E a Seção 2.6 contém o referencial sobre a Unity 3D.

2.1 ANDROID

Para acompanhar a evolução da tecnologia no âmbito móvel, iniciou-se uma corrida realizada pelas maiores empresas do mundo na área, afim de conquistar esse nicho de mercado. Desta forma, o Android foi a resposta da Google para ocupar esse espaço, consistindo em uma nova plataforma de desenvolvimento para aplicativos móveis, com diversas aplicações pré instaladas e, ainda, um ambiente de desenvolvimento bastante poderoso, ousado e flexível (LECHETA, 2010).

Desta maneira, logo quando foi anunciado o surgimento do Android, houve um grande impacto. Pode-se dizer que isso ocorreu porque por trás do Android está a Google, uma grande empresa. Porém, ela não foi a única que iniciou esse processo, mas sim um grupo formado por empresas líderes do mercado de telefonia como a Motorola, LG, Samsung, Sony Ericsson e muitas outras. Este grupo, chamado de Open Handset Alliance (OHA). Foi criado com a intenção de padronizar uma plataforma de código aberto e livre para celulares, com objetivo de atender a todas as expectativas e tendências do mercado atual (LECHETA, 2010).

A partir dessa parceria da Google com as outras empresas, lançou-se o desafio de criar uma plataforma em que todos os participantes da aliança pudessem utilizar em seus *hardwares* e que fossem de acesso economicamente viável a população, surgindo assim o Android. Um sistema operacional móvel para dispositivos, que busca trazer uma plataforma completa para satisfazer o usuário ao desenvolvedor de aplicações, compatibilidade com aparelhos de pouca capacidade a aparelhos grande capacidade, além de ser uma plataforma padrão e gratuita aos membros da aliança (GIROLLETE, 2012).

O sistema operacional do Android foi baseado no kernel 2.6 do Linux e é responsável por gerenciar a memória, os processos, *threads*[1] e a segurança dos arquivos. Além de redes e seus *drivers*, a segurança do sistema é baseada na mesma segurança do Linux, onde no Android cada aplicação é executada em único processo e cada processo por sua vez possui uma *thread* dedicada, sem existir qualquer interferência entre processos (LECHETA, 2010).

O desenvolvimento de aplicações para o Android é realizado de uma maneira simples, sem nem mesmo necessitar ter acesso a um aparelho com Android, sendo necessário apenas um computador, onde é utilizado o Android SDK (BURNETTE, 2010), que é uma ferramenta própria para esse desenvolvimento, que tem um emulador para simular o celular, ferramentas utilitárias e uma API completa para a linguagem Java, possuindo todas as classes necessárias para o desenvolvimento de aplicações. Além das aplicações tradicionais, o Android também possui suporte a gráficos 3D, baseados na especificações OpenGL ES e, dessa maneira, é possível o desenvolvimento de jogos com uma excelente qualidade e resolução (LECHETA, 2010).

2.2 JOGOS

A palavra jogo origina-se do vocábulo latino *Ludus*, que significa diversão, brincadeira, são elementos que sempre estiveram presentes na vida de todos os sujeitos, desde os tempos primitivos até os dias atuais. Os jogos são atividades primordiais na infância, por meio do ato de jogar que a criança tem a oportunidade de desenvolver, descobrir, inventar, exercitar e aprender com facilidade. A partir dos jogos que se estimula a curiosidade, a iniciativa e a autoconfiança (ROSADO, 2006).

Quando se pensa na história dos videogames, nota-se sua rápida evolução, passando de equipamentos rudimentares de baixa qualidade gráfica até chegar nos dias de hoje, em que grandes empresas disputam um mercado milionário e oferecem produtos que incorporam rapidamente os novos recursos tecnológicos (BARACHO et al., 2012). Os jogos acabaram se subdividindo em várias categorias com o tempo: jogos comuns, jogos de regras, jogos de faz de conta e finalmente aos jogos eletrônicos. Tornando-se fruto do desenvolvimento tecnológico, ganhando diferentes espaços e se tornando uma industria tão lucrativa quando Hollywood. Hoje é encontrado os mais variados tipos de jogos, jogos de simulação, RPGs eletrônicos, jogos de raciocínio, jogos de aventura, jogos de xadrez e dama online, jogos de estratégia entre outras variações (MOURA, 2008).

[1] Pequeno programa que trabalha como um subsistema, sendo uma forma de um processo se autodividir em duas ou mais tarefas.

Um jogo virtual 3D é formado por um conjunto de texturas, efeitos 3D, cenários, física nos objetos, inteligência artificial e sons que, ao se integrarem, coordenam o funcionamento de um jogo. Cada um desses itens deve ser desenvolvido nos mínimos detalhes e a união desses motores é responsável pelo controle do que acontece no jogo (MORIBE, 2012).

Após várias gerações de videogames, os jogos começaram a ganhar efeitos gráficos impressionantes, tornando-se cada vez mais realistas (LOPES et al., 2008). Porém isso só se tornou possível com a utilização de *Shaders*. Por meio deles e das unidades gráficas de processamento (GPU), é possível ter os efeitos encontrados nos jogos atuais (BRANCO et al., 2013).

2.3 SHADERS

Shaders são pequenos programas executados dentro de GPU e são utilizados para manipular uma imagem antes dela ser desenhada na tela. São esses pequenos programas que permitem os vários tipos de efeitos de renderização de imagem, que vão desde a adição de sombreamento a determinado objeto ou até mesmo a adição de uma animação (HERGAARDEN, 2011).

Originalmente os *Shaders* realizavam operações em um único pixel, sendo que hoje esse processo é chamado de Fragment Shader (HERGAARDEN, 2011). Além desse método, existem, essencialmente, outros dois tipos de *Shaders*, o *Vertex Shader* e o *Geometry Shader* (BRANCO et al., 2013). Eles atuam diretamente sobre os Vértices e a Geometria de um polígono 3D respectivamente.

Logo, com a existência dos três tipos, existem características para diferenciá-los. É notado que o *Fragment Shader* recebe como entrada um vértice e retorna uma cor associada a esse vértice. Já o *Vertex Shader*, recebe como entrada um vértice e retorna uma nova posição, resultado de alguma transformação. O *Geometry Shader* é outro tipo de *Shader*, onde com ele é possível modificar o vértice, possuindo como saída um novo número de vértices, múltiplos do original (ENGEL et al., 2008).

Além disso, através da Figura 4, é notável que existem variáveis na programação de *Shaders*, como o *Attribute*, que é uma variável global, herdada da OpenGL, usada nos *Vertex Shaders*. Existe ainda a variável global *Uniform*, também herdada da OpenGL, onde é utilizada nos três tipos de *Shaders*, sendo uma variável apenas de leitura. Além dela, existe também a *Varying*, que é usada para passar dados de um *Vertex Shader* para um *Geometry Shader* ou ainda de um *Geometry Shader* para um *Fragment Shader* (LABORATORY, 2010).

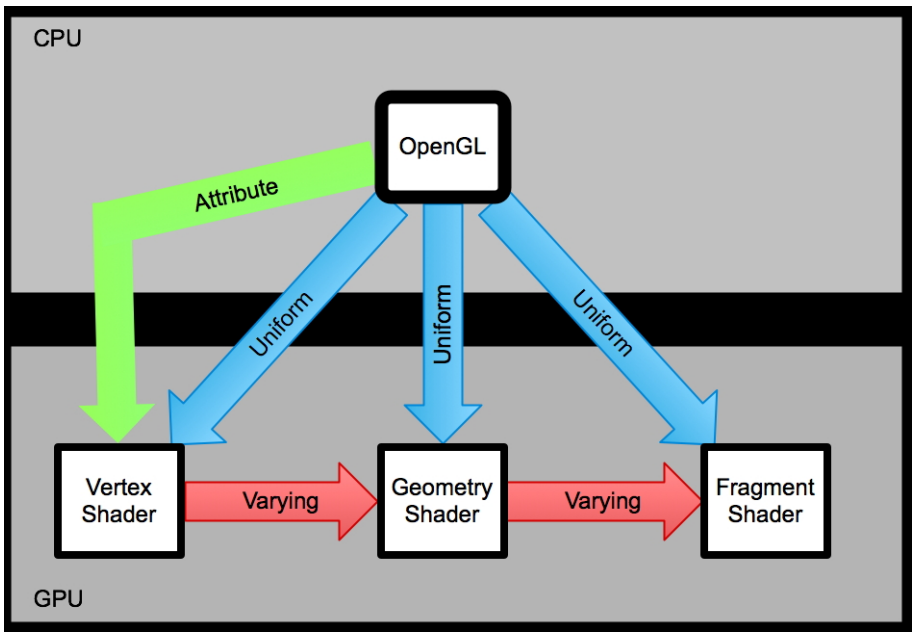

Figura 4: Exemplo dos tipos de Shaders e variáveis da OpenGL.
Fonte: http://www.evl.uic.edu/aej/525/lecture02.html

2.4 OPENGL

O sistema gráfico OpenGL é uma interface de *software* de código aberto criada para o *hardware* gráfico. Em seu nome possui a sigla GL, que significa *Graphics Library*, ou seja, uma biblioteca gráfica para acessar os recursos em *hardware* gráfico. Sendo criada pelo grupo Khronos Group [2], que foi fundado em Janeiro de 2000. Que é focado na criação de padrões de código aberto de APIs livre de *royalties*, para dispositivos embarcados (MUNSHI et al., 2008). A OpenGL permite a criação de programas interativos contendo imagens, cores e objetos tridimensionais. A partir dela, é possível controlar a tecnologia da computação gráfica, afim de produzir imagens realistas ou até mesmos imagens que parte de origem cognitiva ou imaginárias (MUNSHI et al., 2008).

A OpenGL é designada como uma API, possuindo um grande número de capacidades embutidas, que incluem remoção de superfícies, transparência, suavização, mapeamento de texturas, operações de pixel, visualização e modelagem de transformações e efeitos atmosféricos (nevoeiro, fumaça e neblina). Com a OpenGL é possível especificar um conjunto de comandos ou instruções que são imediatamente executados, onde cada comando dirige uma ação de desenho ou efeito especial. A lista desses comandos pode ser criada de forma repetitiva, assim criando efeitos repetitivos (KESSENICH et al., 2009). Efeitos gráficos na OpenGl como *Shaders*, podem ser codificados por meio da implementação da códigos OpenGL Shading Language

[2]Site oficial: www.khronos.org

(GLSL).

Como a OpenGL é uma API extensa, ela acabou sendo dividida em uma outra API, chamada de OpenGL ES. Seu propósito é definir uma API apropriada para dispositivos limitados, onde foi eliminada qualquer redundância existente da OpenGL, ou seja, onde existia mais de uma forma de se realizar a mesma operação, o método mais utilizado foi mantido. Nesse sentido, grande parte das funções foram removidas. Como por exemplo as especificações de geometria, que um aplicativo OpenGL pode usar o modo imediato, lista de exibição, ou matrizes de vértices. Na OpenGL ES, existem apenas matrizes de vértices, os outros métodos foram removidos, nesse sentido a OpenGL ES se tornou limitada (MUNSHI et al., 2008).

Com o surgimento da OpenGL ES, ela se tornou uma interface de programação de aplicações para gráficos 3D avançados. Sendo voltada principalmente para dispositivos portáteis e embarcados, como celulares, assistentes pessoais, veículos e aeronaves (MUNSHI et al., 2008).

2.5 GLSL

A GLSL é uma linguagem em alto nível, que utiliza *Shaders* para construir efeitos gráficos na OpenGL. A partir de instruções em GLSL, o desenvolvedor é capaz de expressar graficamente o processamento que ocorre em forma de pixels, vértices e geometrias, utilizando funções que alteram cor, forma e posição de geometrias (SOUZA, 2012).

A linguagem GLSL foi baseada na linguagem de programação C, com muitas de suas propriedades mantidas. A menos quando existia problemas de performance ou implementação, algumas características do C++ foram implementadas como herança e sobrecarga. Desta forma, como pode ser observado na (Figura 5), o programador não precisa mais escrever seus *Shaders* como antigamente, em Assembly, mas sim codificar seus próprios, criando seus *Vertex Shaders* e *Fragment Shaders* em GLSL, além de que existem atualmente diversos *Shaders* prontos disponíveis gratuitamente (SOUZA, 2012).

A principal vantagem da GLSL é que ela é totalmente integrada com a OpenGL, possuindo compatibilidade com diferentes plataformas, incluindo Windows, Linux e Mac OS. Além disso, os *Shaders* feitos em GLSL podem ser executados em qualquer *hardware* com suporte acima das versões da OpenGL 1.3. Com isso, grande parte dos fabricantes inclui um compilador de GLSL nos drivers de suas placas de vídeo, sendo até mesmo possível otimizar o código GLSL para um *hardware* específico (VIANA, 2013).

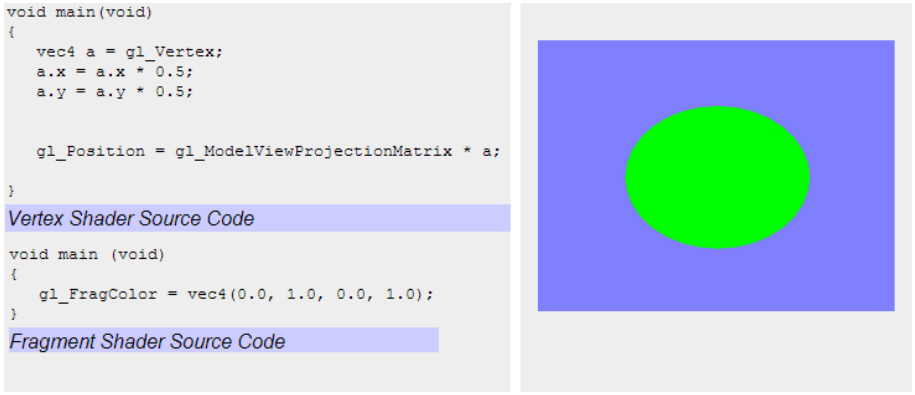

Figura 5: Exemplo de código GLSL e seu respectivo resultado.
Fonte: O autor

2.6 UNITY 3D

A Unity 3D é um motor de jogos (*game engine*), que conta com um ambiente integrado de desenvolvimento. Ela utiliza como linguagens de programação para produção das aplicações as linguagens Boo, JavaScript e C#, sendo compatível com diversos navegadores *web* e também com diversos programas de modelagem tridimensional. Além disso, a Unity 3D possui interface gráfica, sistemas de colisões, iluminação e som. Outro ponto forte é o fato dela ser multiplataforma, sendo desenvolvida aplicações tanto para *desktop*, *mobile* e *web*, bastando algumas configurações (CARVALHO, 2012).

A Unity 3D, apesar de exigir licença comercial na versão completa, possui uma versão gratuita que contém a maioria dos recursos e funcionalidades necessários para o desenvolvimento de uma aplicação. Ela possui sistema de importação de arquivos 2D e 3D, suportando extensões jpg e png para 2D, blend (Blender 3D) e max (Autodesk) para 3D, além de ser possível a utilização de *Shaders* com facilidade (MORIBE, 2012).

Possuindo um editor completo como visto (Figura 6), a Unity 3D consegue realizar o posicionamento de objetos, importar arquivos e outras tarefas. Também é possível a realização de testes da aplicação sem a necessidade da compilação da versão definitiva da aplicação, no entanto, não se aplica quando se referido a dispositivos móveis. A Unity 3D se destaca principalmente na velocidade de renderização dos gráficos, funcionando tanto para DirectX como para OpenGL. O motor gráfico presente nela é de alta qualidade, possuindo um bom sistema de partículas e animação de modelos, além de também possuir ferramentas que auxiliam a utilização de rede, áudio e vídeo (MORIBE, 2012).

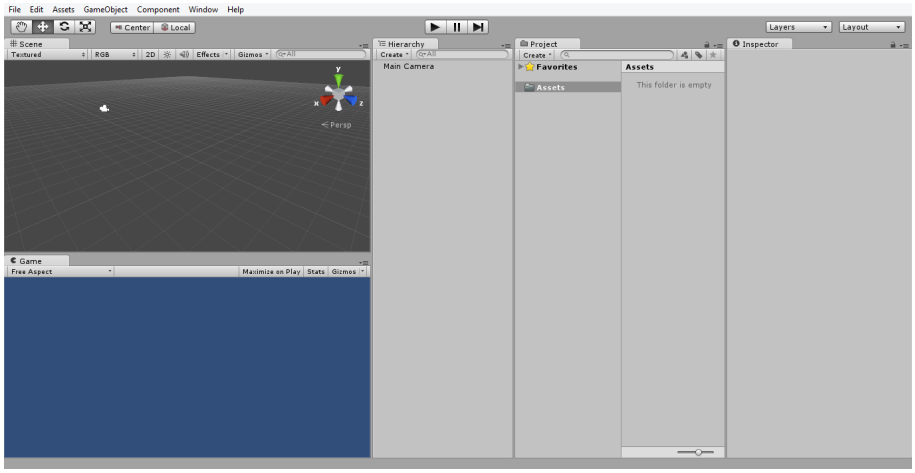

Figura 6: Interface da Unity 3D.
Fonte: O autor

3 ESTADO DA ARTE

Existem várias ferramentas que têm como intuito o processo de codificação e testes de *Shaders*, no entanto, grande parte dos mesmas não atendem completamente as necessidades já mencionadas nesse trabalho. Serão destacadas a seguir, algumas dessas ferramentas, mostrando seus recursos.

Inicialmente foi testada a glslDevil Figura 7, que é uma ferramenta gratuita de depuração e análise de *Shaders* OpenGL. Ela possui como característica principal a depuração de *Shaders* GLSL em programas OpenGL, sem a necessidade de recompilar ou ter o código fonte do programa específico, bastando apenas o executável da aplicação que será analisada. Com isso o glslDevil apenas intercepta as chamadas OpenGL e permite a depuração do código (STRENGERT et al., 2007). Apesar dessas características, um dos seus problemas é a falta de recursos a respeito da OpenGL ES, não suportando esse tipo de tecnologia.

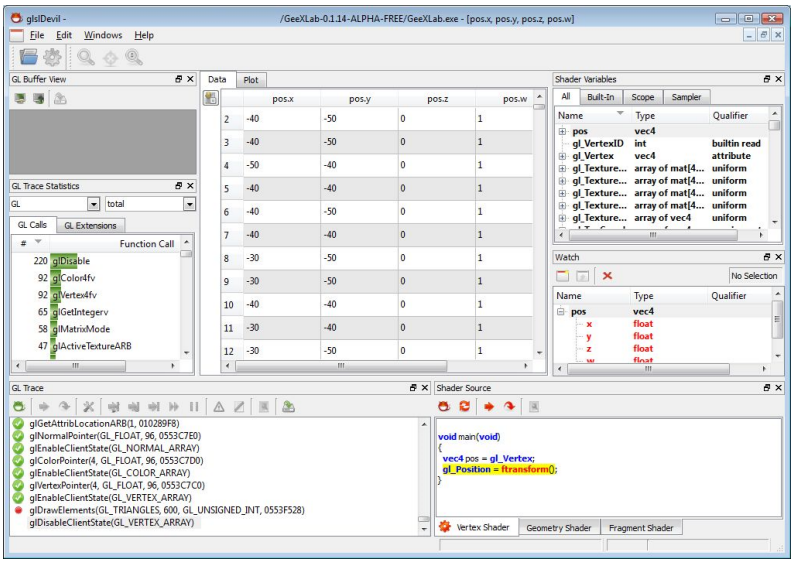

Figura 7: Interface da glslDevil.
Fonte: http://www.geeks3d.com/20091124/debug-your-glsl-shaders-with-glsldevil/

Uma outra ferramenta também conhecida é o ShaderDesigner Figura 8, que é uma IDE para programação e teste de *Shaders* OpenGL. Foi desenvolvido pela Typhoon Labs até 2004

e atualmente é um projeto praticamente descontinuado (VIANNA; GOMES, 2012). Possui como funcionalidade o suporte a *Vertex Shaders* e *Fragmet Shaders*, gerenciamento de múltiplas texturas e *highlight* para códigos GLSL. Porém ele não possui capacidade de testes dos *Shaders* diretamente nos smarthphones.

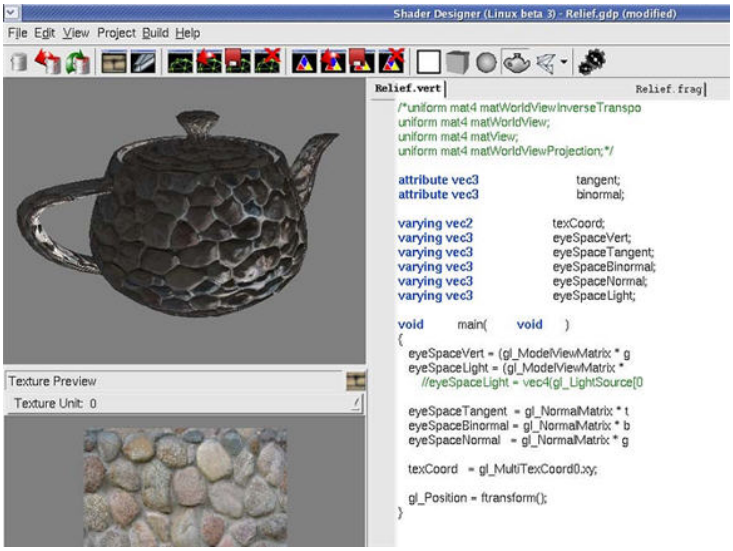

Figura 8: Interface da ShaderDesigner.
Fonte: http://createdigitalmotion.com/2006/10/free-opengl-shader-language-course-learn-to-build-3d-image-effects/

Existe também a IDE ShaderLabs Figura 9, que provê as principais funcionalidades para o aprendizado de programação de *Shaders*, como *Vertex*, *Geometry* e *Fragment Shader* (VIANNA; GOMES, 2012). No entanto, a visualizações dos *Shaders* apenas é possível no computador.

Outra ferramenta é a RenderMonkey Figura 10, que é uma IDE desenvolvida pela AMD para o desenvolvimento de efeitos visuais usando *Shaders*. Como características, possui a área de renderização dos resultados e editores para programação GLSL. No entanto, ela não é mais mantida mais pela AMD, pois seu projeto foi descontinuado e atualmente é disponibilizada gratuitamente na internet, sem atualizações (VIANNA; GOMES, 2012). É disponibilizada apenas para testes de *Shaders* no sistema operacional Windows, sem a possibilidade de realizar o teste imediato em um dispositivo móvel.

Um outro exemplo é o Shader Forge [1] Figura 11, que é um editor de *Shaders* para Unity. O editor possibilita a visualização em tempo real dos *Shaders* e a criação deles sem que seja necessário sua codificação, no entanto é uma ferramenta paga e que tem apenas seu funcionamento unificado a plataforma Unity.

[1] Site oficial: www.unity3d.com

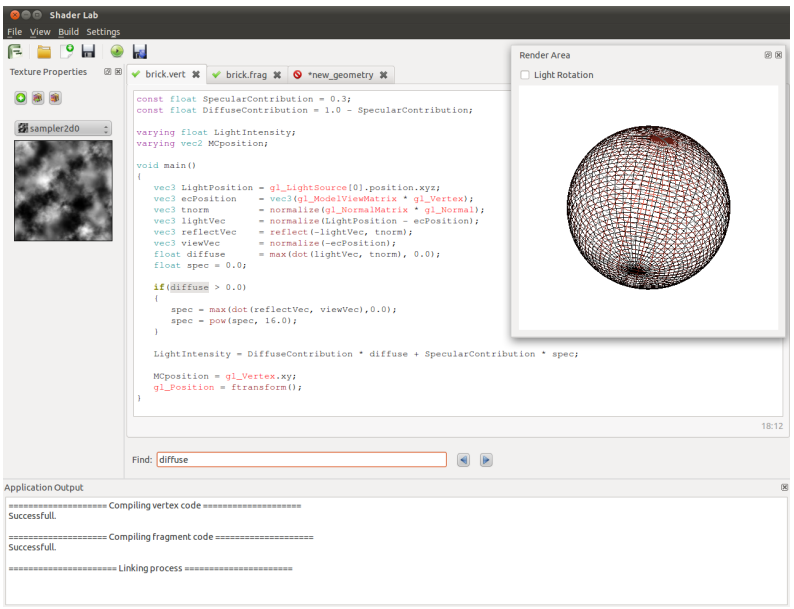

Figura 9: Interface da ShaderLabs.
Fonte: http://www.dcc.ufrj.br/ shaderlabs/Shaderlabs

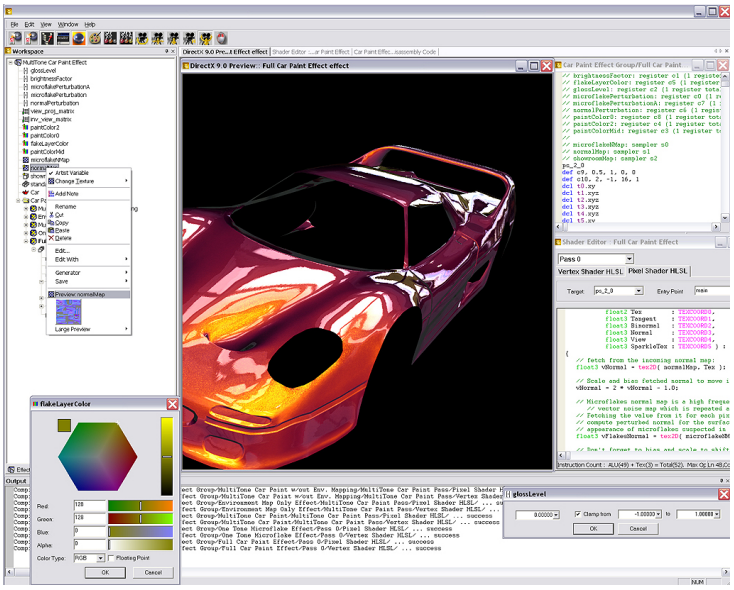

Figura 10: Interface da RenderMonkey.
Fonte: http://developer.amd.com/tools-and-sdks/archive/legacy-cpu-gpu-tools/rendermonkey-toolsuite/rendermonkey-toolsuite-ide-features/

A única aplicação móvel encontrada com intuito semelhante ás necessidades abordados por esse trabalho foi a ShaderToy Figura 12. Nela é possível a realização de testes de *Shaders* diretamente no aparelho (SHADERTOY, 2009), no entanto, não deixa de necessitar que os programas GLSL precisem ser transferidos para o aparelho e apenas assim abertos pela aplicação, tornando-se uma operação lenta. Destaca-se que em testes realizados com o dispositivo Samsung Galaxy Mega 6.3 (Android versão 4.2.2), a ferramenta apresentou constantes

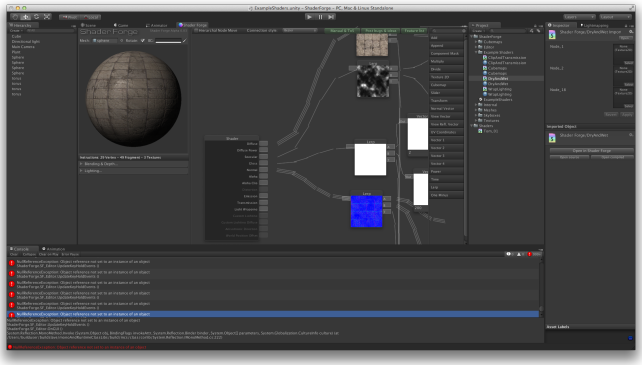

Figura 11: Interface da Shader Forge.
Fonte: http://acegikmo.com/shaderforge/

travamentos, contudo, o motivo deste defeito não foi investigado em detalhes.

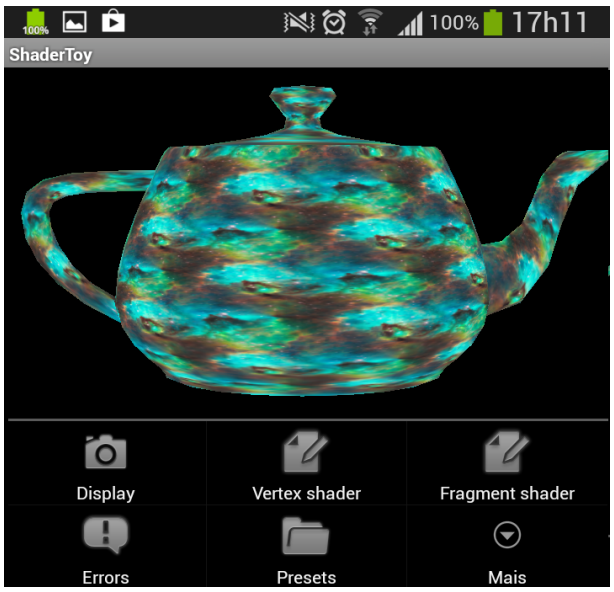

Figura 12: Interface da ShaderToy.
Fonte: O autor

3.1 COMPARATIVO

É notável que as ferramentas citadas não atendem as os objetivos mencionados neste trabalho. Pode-se perceber peoa Quadro 1 como principal deficiência das ferramentas mencionadas é a realização dos testes diretamente no dispositivo móvel. Elas possuem apenas a visualização dos *Shaders* no computador e dependendo da ferramenta como o Shader Forge, possuem licença comercial. Desta maneira, a necessidade da criação de um software que consiga cumprir essas necessidades é visível. Cita-se ainda, que a ferramenta ShaderToy permite a visualização e edição de GLSL diretamente no dispositivo, entretanto, seu uso se torna descon-

fortável devido a sua utilização apenas pelo teclado virtual do dispositivo, além de constantes travamentos e, de uma interface nada prática, pois é sempre necessário realizar a troca de suas abas para manipulação da aplicação.

Tabela 1: Comparação das ferramentas testadas.

	Características		
Ferramentas	**Codificação de Shaders**	**Teste no dispositivo Móvel**	**Descontinuada**
glslDevil			
ShaderDesigner	**X**		**X**
ShaderLabs	**X**		
RenderMonkey	**X**		**X**
ShaderForge	**X**		
ShaderToy	**X**	**X**	

4 PROCEDIMENTOS METODOLÓGICOS

Este Capítulo descreve os passos para o desenvolvimento do *Mobile Shader Editor*: um editor de *Shaders* para Android. Segue em itens os passos metodológicos sugeridos para sua execução.

- Estudo das tecnologias: Estudo sobre as tecnologias empregadas na construção da ferramenta, entendendo o funcionamento do sistema operacional móvel Android, situação e carências a respeito da OpenGL ES, codificação de *Shaders* GLSL e desenvolvimento de aplicações na Unity 3D;

- Construção de uma aplicação Android capaz de executar *Shaders* GLSL: Construção da aplicação móvel, nela sendo possível a execução dos códigos GLSL e captura dos registros de erros;

- Construção de um editor GLSL para plataforma PC: Construção da aplicação para *Desktop*, que consistirá em um editor de *Shaders* GLSL e visualizador dos registros de erros. Esta aplicação será construída inicialmente com o uso da plataforma Unity 3D, não atuando como um componente da mesma, mas sim como uma ferramenta de apoio;

- Integração das ferramentas desenvolvidas: Por meio da utilização do ADB, será possível a integração das ferramentas, onde a primeira enviará o código GLSL para a aplicação móvel. Esta além de executar ele, gerará registros de erros sobre o código, que serão enviados para a visualização na aplicação *Desktop*;

- Validação da ferramenta: Desenvolvimento de uma metodologia de testes para validação da ferramenta, onde será possível obter as conclusões se a mesma cumpre as expectativas e necessidades, que seriam de ser uma ferramenta intuitiva e de ágil desenvolvimento, além de também dela ser superior as ferramentas existentes;

- Escrita da monografia: Escrita da monografia, contendo todo o material abstraído para o desenvolvimento e as possíveis conclusões sobre a ferramenta.

5 CRONOGRAMA

Neste Capítulo é apresentado o cronograma para realização deste trabalho.

1. Estudo das tecnologias;
2. Construção de uma aplicação Android capaz de executar *Shaders* GLSL;
3. Construção de um editor GLSL para plataforma PC;
4. Integração das ferramentas desenvolvidas;
5. Validação da ferramenta;
6. Escrita da monografia.

Atividade	Maio	Junho	Julho	Agosto	Setembro	Outubro	Novembro	Dezembro
1	**X**	**X**						
2		**X**	**X**	**X**	**X**	**X**		
3		**X**	**X**	**X**	**X**	**X**		
4		**X**	**X**	**X**	**X**	**X**		
5		**X**	**X**	**X**	**X**	**X**		
6				**X**	**X**	**X**	**X**	**X**

6 DESENVOLVIMENTO

Neste Capítulo será apresentado o desenvolvimento do *Mobile Shader Editor*: um editor de *Shaders* para Android proposto nesse trabalho.

6.1 CONSTRUÇÃO DA APLICAÇÃO ANDROID

6.1.1 UNITY 3D

Foi desenvolvido uma aplicação no motor de jogos Unity 3D Figura 13 (destacada em amarelo), para a plataforma Android com intuito de possibilitar a leitura e compilação dos códigos GLSL diretamente no dispositivo móvel. Sendo possível a aplicação do *Shader* de modo instantâneo ao objeto. Em resumo, o fluxo de execução consiste em enviar um código GLSL do Desktop para o dispositivo móvel e, em seguida, realizar a leitura e execução deste por meio da aplicação desenvolvida com a Unity 3D, assim permitindo a visualização do *Shader* no Android.

No entanto, ao realizar testes no dispositivo móvel real, percebeu-se que a execução não ocorria com os mesmos efeitos resultantes da execução dentro do editor da Unity. Com base em informações da documentação e *website* da Unity 3D e testes, foi notado que a mesma possui um compilador do código GLSL interno. Assim, quando ocorre a troca de informações de um código GLSL no editor da Unity 3D, a ferramenta pré-compila este código antes de aplicar o resultado do *Shader*, porém, após exportada para a plataforma Android, a aplicação não apresentou o mesmo comportamento, resultando em falha na substituição do *Shader* no dispositivo móvel, toda a região referente ao cubo 3D recebe cor branca sem qualquer efeito do código GLSL.

A partir disso, iniciou-se uma tentativa sem sucesso de utilizar o mesmo compilador de códigos GLSL da Unity 3D como um arquivo interno do projeto. Através da pesquisa realizada, detectou-se que o arquivo responsável pela pré-compilação do código GLSL se chamava ”Cg-Batch.exe”. Porém, a partir da versão 4.5 da Unity 3D esse compilador se tornou inacessível

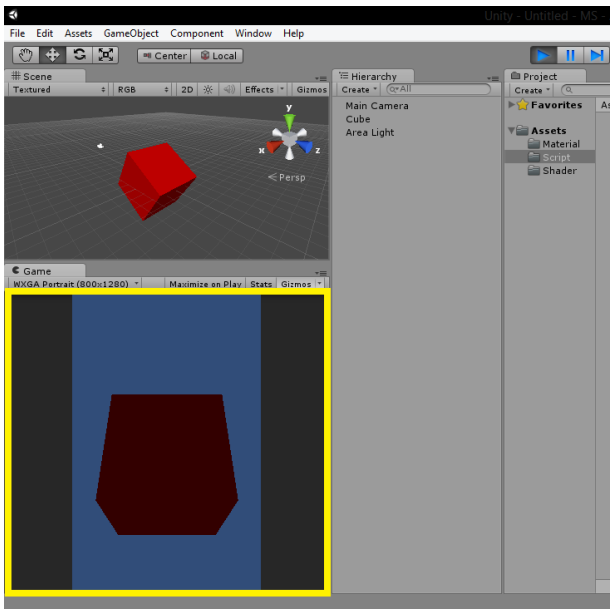

Figura 13: Aplicação desenvolvida na Unity 3D executando no Ambiente de Emulação da Ferramenta.

Fonte: O autor

e não foi encontrada nenhuma informação sobre a existência deste pré-compilador nas novas versões da ferramenta.

Dessa forma, por conta das dificuldades encontradas, a Unity 3D acabou sendo abandonada. Assim, para continuar o desenvolvimento optou-se por utilizar a JPCT-AE.

6.1.2 JPCT-AE

A partir das dificuldades encontradas com a Unity 3D, o desenvolvimento da aplicação passou a ser realizado na engine 3D JPCT-AE. Esta engine é uma versão na JPCT, funcionando como uma API, programada na linguagem Java, com funcionamento no sistema operacional Android. Ela possui ferramentas de renderização, manipulação de objetos 3D e possibilidade da utilização de *Shaders* GLSL. Com a utilização dessa engine o programador acaba tendo mais liberdade, pois tudo através dela é feito via programação, possuindo grande diferença com a Unity 3D, que acaba sendo mais padronizada, fechada no seu ambiente de desenvolvimento.

Por meio da JPCT-AE foi possível implementar a leitura e compilação do código GLSL de forma instantânea e dinâmica. Nela é realizada a troca do *Shader* em tempo real diretamente no dispositivo, como visto na Figura 14, que mostra o *Shader* compilado no objeto, sem ocorrer problemas ou travamentos. A substituição do *Shader* é realizada toda vez que a aplicação percebe alteração nos arquivos enviados pela aplicação Desktop.

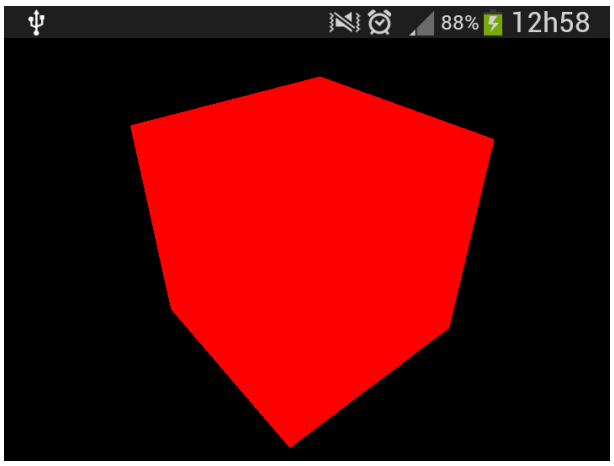

Figura 14: Aplicação desenvolvida na JPCT.
Fonte: O autor

Como dificuldades encontradas até o momento, podem ser mencionadas a interpretação e captura dos registros de erros, gerados quando um código GLSL incorreto é compilado. Quando o *Shader* é compilado com erros, ele fecha a aplicação, dessa forma, é necessário implementar alguma ferramenta que analise se o código está incorreto antes de executá-lo.

A partir disso, é importante ressaltar que é possível monitorar os erros do código GLSL utilizando da SDK do Android como visto na Figura 15, onde é possível capturar as *exceptions* geradas pela JPCT-AE, quando um código GLSL incorreto é compilado, assim, possibilitando que sejam exibidos na interface da aplicação *Desktop*.

```
com.exam...   jPCT-AE          Compiling shader program!
com.exam...   jPCT-AE          Could not compile shader 35632: 0(4) : error C0000: syntax error, u ↵
                               nexpected '}', expecting ',' or ';' at token "}"
com.exam...   jPCT-AE          [ 1406056490201 ] - ERROR: Failed to load and compile fragment shad ↵
                               ers!
```

Figura 15: Registro de erros gerados por um código GLSL
Fonte: O autor

6.2 CONSTRUÇÃO DA APLICAÇÃO DESKTOP

Foi desenvolvida a aplicação para Desktop como visto na Figura 16, ela é um editor para programação de *Shaders GLSL*, que possui destaque da sintaxe da linguagem. Foi desenvolvido com a da linguagem de programação Java, utilizando os componentes *Swing*[1] para moldar sua interface.

A aplicação Desktop possui a integração com a versão móvel, realizada por meio do

[1]Biblioteca nativa do Java, utilizada para criação de interfaces gráficas.

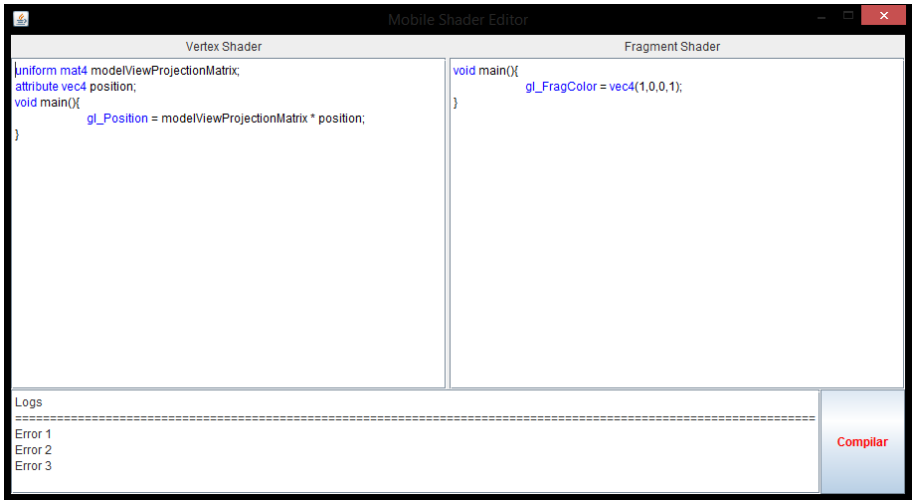

Figura 16: Editor desenvolvido para programação de Shaders.
Fonte: O autor

ADB em sua programação. Assim, é possível enviar os códigos GLSL para o dispositivo móvel e conseguir sua visualização no mesmo instante.

REFERÊNCIAS

BARACHO, A.; GRIPP, F.; LIMA, M. Os exergames ea educação física escolar na cultura digital. **Revista Brasileira Ciências do Esporte**, SciELO Brasil, p. 111–126, 2012.

BRANCO, M.; MALFATTI, S.; LAMAR, M. V. **Jogos Eletrônicos na Prática: Livro de Tutoriais**. [S.l.]: XI Simpósio Brasileiro de Games e Entretenimento Digital, Editora Feevale, 2013.

BURNETTE, E. **Hello, Android : introducing Google's Mobile Development Plataform**. Third edition. [S.l.]: Pragmatic Programmers, 2010.

CARVALHO, D. E. Desenvolvimentos do jogo robogol para navegadores web utilizando o motor de jogos unity 3d. **ICMC USR**, 2012.

ENGEL, W. et al. **Programming Vertex, Geometry, and Pixel Shaders**. [S.l.]: Cengage Delmar Learning, 2008.

EVANS, S. H. **The do good profit motive**. jan 2013. Disponível em: http://www.evl.uic.edu/aej/525/lecture02.html. Acesso em: 24/03/2014.

GIROLLETE, R. B.-H. A. **ANDROID: Visão Geral**. 1. ed. Cascavel PR, 2012.

HERGAARDEN, M. **Graphics shaders**. 1. ed. VU Amsterdam, jan 2011.

KESSENICH, J.; BALDWIN, D.; ROST, R. **The OpenGL® Shading Language**. Third edition. [S.l.]: Addison-Wesley Professional, 2009.

LABORATORY, E. V. **GLSL language study**. 2010. Disponível em: http://www.evl.uic.edu/aej/525/lecture02.html. Acesso em: 25/04/2014.

LECHETA, R. R. **Google Android-Aprenda A Criar Aplicaçoes: Para Dispositivos Moveis Com O Android SDK**. 2. ed. [S.l.]: Novatec Editora, 2010.

LOPES, R. O.; SORIANO, T. G.; OLIVEIRA, Y. G. **Evolução e Inovação no Mercado de Jogos Eletrônicos**. 1. ed. Rio de Janeiro, fev 2008.

MORIBE, V. A. Jogo para android com unity3d. **Reverte-Revista de Estudos e Reflexões Tecnológicas da Faculdade de Indaiatuba**, n. 10, 2012.

MOURA, J. S., 2008. jogos eletrônicos e professores: primeiras aproximações. **Seminário Jogos eletrônicos, educação e comunicação: construindo novas trilhas, IV**, 2008.

MUNSHI, A.; GINSBURG, D.; SHREINER, D. **OpenGL ES 2.0 programming guide**. [S.l.]: Pearson Education, 2008.

NVIDIA. **Vertex Texture Fetch**. 2004. Disponível em: http://www.nvidia.com/object/usingvertextextures.html. Acesso em: 19/04/2014.

PASSOS, E. B. et al. Tutorial: Desenvolvimento de jogos com unity3d. **VIII Brazilian Symposium on Games and Digital Entertainment**, Rio de Janeiro, oct 2009.

ROSADO, J. dos R. História do jogo e o game na aprendizagem. **Universidade do estado da Bahia - UNEB**, Bahia, oct 2006.

SHADERTOY. **The application to test Shaders**. 2009. Disponível em: https://play.google.com/store/apps/details?id=com.simongreen.shadertoy. Acesso em: 19/04/2014.

SOUZA, L. F. R. T. Paralelização em cuda/glsl do algoritmo sift para reconhecimento de íris. **IBICT**, Rio de Janeiro, fev 2012.

STRENGERT, M.; KLEIN, T.; ERTL, T. A hardware-aware debugger for the opengl shading language. **Graphics Hardware**, p. 81–88, 2007.

VIANA, J. R. M. Programação em gpu: Passado, presente e futuro. **Universidade Federal do Rio de Janeiro**, Rio de Janeiro, oct 2013.

VIANNA, F. V.; GOMES, T. E. Shaderlabs: Desenvolvimento agil de uma ide para opengl shaders. **Universidade Federal do Rio de Janeiro**, Rio de Janeiro, fev 2012.