

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

William Johnson dos Santos Okano

**Implementação de uma biblioteca gráfica
multiplataforma utilizando OpenGL e GLFW**

Uberlândia, Brasil

2018

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

William Johnson dos Santos Okano

**Implementação de uma biblioteca gráfica
multiplataforma utilizando OpenGL e GLFW**

Trabalho de conclusão de curso apresentado
à Faculdade de Computação da Universidade
Federal de Uberlândia, Minas Gerais, como
requisito exigido parcial à obtenção do grau
de Bacharel em Sistemas de Informação.

Orientador: Prof. Dr. André Ricardo Backes

Universidade Federal de Uberlândia – UFU

Faculdade de Ciência da Computação

Bacharelado em Sistemas de Informação

Uberlândia, Brasil

2018

William Johnson dos Santos Okano

Implementação de uma biblioteca gráfica multiplataforma utilizando OpenGL e GLFW

Trabalho de conclusão de curso apresentado
à Faculdade de Computação da Universidade
Federal de Uberlândia, Minas Gerais, como
requisito exigido parcial à obtenção do grau
de Bacharel em Sistemas de Informação.

Trabalho aprovado. Uberlândia, Brasil, 20 de Julho de 2018:

Prof. Dr. André Ricardo Backes
Orientador

Prof. Dr. Mauricio Cunha Escarpinati

**Prof. Me. William Chaves de Souza
Carvalho**

Uberlândia, Brasil
2018

Agradecimentos

Gostaria de agradecer ao meu orientador, Prof. Dr. André Ricardo Backes pela paciência em todos esses semestres.

*“O sucesso é a soma de pequenos
esforços repetidos dia após dia.”*
(Robert Collier)

Resumo

A dificuldade de aprendizagem de programação pode ser causada por várias razões, como a incapacidade de compreensão de algoritmos por este ser um conceito muito abstrato.

Várias são as formas utilizadas no ensino de algoritmos e estruturas de controle. Este trabalho apresenta uma proposta de biblioteca para auxiliar o ensino de programação, utilizando conceitos gráficos para facilitar o entendimento das estruturas de programação. A solução proposta tem como foco desenvolver uma biblioteca multiplataforma, com um conjunto de funções simples para gerenciamento de elementos gráficos, bem como a elaboração de um guia de utilização e sua respectiva documentação.

Palavras-chave: Biblioteca de funções, elementos gráficos, OpenGL.

Lista de ilustrações

Figura 1 – Linguagens de programação utilizadas primariamente no ensino de introdução à programação	16
Figura 2 – Arquitetura proposta para a biblioteca	19
Figura 3 – Exemplo inicialização da biblioteca	21
Figura 4 – Exemplos de polígonos regulares: Triângulo, quadrado, pentágono e hexágono	28
Figura 5 – Exemplo: Todas as formas implementadas	31
Figura 6 – Exemplo: Relógio analógico	32

Lista de abreviaturas e siglas

API	<i>Application Programming Interface</i>
FPS	<i>Frames per second (Quadros por segundo)</i>
GLFW	<i>Graphics Library Framework</i>
GLUT	<i>OpenGL Utility Toolkit</i>
IDE	<i>Integrated Development Environment (Ambiente de Desenvolvimento Integrado)</i>
RGB	<i>Red, Green, and Blue (Vermelho, Verde e Azul)</i>
SDL	<i>Simple DirectMedia Layer</i>
TI	<i>Tecnologia da Informação</i>

Sumário

1	INTRODUÇÃO	10
1.1	Objetivos	11
1.2	Funcionalidades previstas	11
2	TRABALHOS CORRELATOS	13
2.1	Borland Graphics Interface	13
2.2	GLUT e FreeGLUT	13
2.3	SDL	14
2.4	Ferramentas de técnica de visualização	14
3	DESENVOLVIMENTO	15
3.1	Tecnologias Empregadas	15
3.1.1	Linguagem de programação C	15
3.1.2	OpenGL	16
3.1.3	GLFW	17
3.1.4	TinyCThread	17
3.2	Atividades Desenvolvidas	18
3.2.1	A biblioteca gráfica	18
3.2.1.1	Iniciando a biblioteca	20
3.2.1.2	Limpando a tela	21
3.2.1.3	Desfazer último desenho	22
3.2.1.4	Refazer o último desenho	22
3.2.1.5	Alterar a cor de preenchimento de um objeto	22
3.2.1.6	Obter a cor de preenchimento do objeto	23
3.2.1.7	Definindo o tamanho de um objeto	24
3.2.1.8	Obtendo o tamanho de um objeto	24
3.2.1.9	Desenhando um polígono	25
3.2.1.10	Desenhando um ponto	26
3.2.1.11	Desenhando um triângulo	26
3.2.1.12	Desenhando um retângulo	27
3.2.1.13	Desenhando um quadrado	27
3.2.1.14	Desenhando um polígono regular	28
3.2.1.15	Desenhando um círculo	29
3.2.1.16	Desenhando um pentágono	30
3.2.1.17	Desenhando uma linha	30
3.3	Estado Atual do Desenvolvimento	31

4	CONSIDERAÇÕES FINAIS	34
4.1	Trabalhos Futuros	34
4.2	Dificuldades Encontradas	34
	Conclusão	36
	REFERÊNCIAS	37

1 Introdução

Segundo (SANTIAGO; DAZZI, 2004), um dos grandes problemas enfrentados em muitas instituições de ensino é relacionado ao baixo índice de assimilação dos estudantes nas disciplinas onde o conhecimento de programação seja um requisito. Uma das razões para a dificuldade no aprendizado é que as linguagens de programação possuem entidades muito abstratas, sendo difícil a visualização de como ocorre o fluxo das informações dentro de estruturas de controles, tais como laços de repetição e condicionais lógicas, ponteiros, *arrays*, etc.

Muitos alunos de cursos introdutórios utilizam-se das mais diversas técnicas de aprendizado para tentar assimilar o funcionamento de um programa ou algoritmo. Uma das técnicas é a utilização de fluxogramas, onde são desenhadas caixas, setas e direções de como o programa deve se comportar, ou então a execução passo à passo, onde anota-se o valor de cada variável em cada linha, laço de repetição ou condicional lógica executado. Alguns optam por utilizar ferramentas de auxílio visual, como o [Scratch \(2018\)](#) ou [Ruby Warrior \(2018\)](#). Apesar de serem excelentes ferramentas de ensino, estas ferramentas apresentam alguns pontos fracos. O Scratch, por exemplo, é uma linguagem de programação própria, enquanto RubyWarrior utiliza a linguagem de programação [Ruby \(1995\)](#). No entanto, em cursos de introdução à programação, geralmente, é utilizada a linguagem de programação C.

Com a introdução e popularização das linguagens [C++ \(1979\)](#) e [Java \(1995\)](#), vários programadores optaram por deixar a linguagem de programação C. Apesar de ela ainda ser considerada uma importante linguagem de programação, tanto na indústria da engenharia, quanto nas instituições de ensino, ela carece de ferramentas de visualização. A maioria das ferramentas de visualização de programação feitas para a linguagem C são focadas em algoritmos avançados, como algoritmos de ordenação, grafos, e não nas necessidades de um iniciante em programação, como declaração de variáveis, funções e ponteiros ([KIRBY; TOLAND; DEEGAN, 2010](#)).

De acordo com [Mayer \(1981\)](#), a aquisição de novo conhecimento pode ser obtido através do relacionamento da nova informação com as informações previamente adquiridas, utilizando as informações antigas como reforço de memória enquanto assimila as novas informações. Segundo [Souza e Dazzi \(2000\)](#), um grande motivador do processo de ensino aprendizagem é a utilização de alguma ferramenta computacional que permita os alunos confeccionar os seus algoritmos bem como testarem as suas soluções visualizando o resultado gerado por elas.

1.1 Objetivos

O objetivo geral deste trabalho é a implementação de uma biblioteca gráfica, baseada em OpenGL, que dê a possibilidade aos alunos de cursos introdutórios de programação compreender, de forma visual, os efeitos dos códigos ensinados em sala de aula. Os objetivos específicos do trabalho proposto são:

1. Criação de uma biblioteca gráfica que seja de fácil instalação;
2. Criação de uma biblioteca que possua uma API simples e objetiva;
3. Criação de um manual de utilização e exemplos da biblioteca.

1.2 Funcionalidades previstas

Apesar de ser uma abstração da biblioteca OpenGL, este trabalho não possui funções para a manipulação de janelas diretamente, com exceção da função que inicializa a biblioteca, na qual é automaticamente criada uma janela e um contexto OpenGL.

As funcionalidades previstas para este trabalho podem ser divididas em 3 categorias: funções de gerenciamento da biblioteca; funções de configuração dos objetos; e funções de desenho.

As funções da biblioteca são divididas da seguinte forma:

- Funções de gerenciamento
 - Inicializar a biblioteca;
 - Limpar a tela;
 - Desfazer o último desenho;
 - Refazer o último desenho.
- Funções de configuração
 - Definir a cor de preenchimento do objeto;
 - Obter a cor de preenchimento do objeto;
 - Definir o tamanho de um ponto;
 - Obter o tamanho de um ponto;
 - Definir a largura de uma linha;
 - Obter a largura de uma linha.
- Funções de desenho

- Polígono;
- Ponto;
- Triângulo;
- Retângulo;
- Quadrado;
- Polígono regular;
- Círculo;
- Pentágono;
- Hexágono;
- Decágono;
- Dodecágono;
- Linha.

2 Trabalhos Correlatos

Nesta seção serão apresentadas algumas bibliotecas que fazem o gerenciamento de janelas e contexto OpenGL, bem como algumas ferramentas que auxiliam no ensino da programação fazendo uso de técnicas de visualização. Existem várias outras bibliotecas que fazem o gerenciamento de janelas OpenGL, assim como também existem várias outras ferramentas que ensinam programação através da técnica de visualização, porém, as bibliotecas e ferramentas apresentadas a seguir possuem semelhança com o objetivo deste trabalho.

2.1 Borland Graphics Interface

A biblioteca *Borland Graphics Interface*, também conhecida como BGI, é uma biblioteca gráfica para as linguagens C e C++ que é distribuída junto a vários compiladores da Borland para o sistema operacional DOS ([Freeware BGI Drivers, 2008](#)), e eventualmente com suporte a Windows, desde 1987. A biblioteca BGI é menos poderosa que outras bibliotecas, como a SDL ou OpenGL pois foi desenvolvida para a apresentação de gráficos e não para a criação de aplicações 3D baseadas em eventos. Entretanto, ela possui uma API muito simples de ser utilizada e que ajudou na sua popularização. O manual da BGI pode ser encontrado em [Borland Graphics Interface \(BGI\) for Windows \(2004\)](#)

2.2 GLUT e FreeGLUT

[GLUT \(2000\)](#) é o kit de ferramentas utilitárias para o desenvolvimento de programas OpenGL nas linguagens C e C++, utilizando um sistema de gerenciamento de janelas. Este kit de ferramentas faz com que seja consideravelmente mais fácil aprender e explorar a programação utilizando OpenGL. Além do mais, provê uma API portátil, permitindo que seja aproveitado o mesmo código OpenGL com GLUT entre vários sistemas operacionais.

Além disso, o GLUT é designado para construir aplicações de pequeno e médio porte já que, apesar de bem adequado tanto para o aprendizado, quanto para o desenvolvimento de aplicações simples usando OpenGL, ele não é um kit com todos os recursos necessários para se escrever aplicações de grande porte.

Já o [FreeGLUT \(1999\)](#) é uma versão alternativa de código-aberto à biblioteca GLUT, pois esta possui licença proprietária de autoria de Mark Kilgard. Foi desenvolvida em 1999 e liberada sob a licença *X-Consortium*. A biblioteca FreeGLUT possui todas as

funcionalidades da biblioteca GLUT, além de possuir algumas funcionalidades adicionais. Apesar de ser uma alternativa de código-aberto e mais atualizada em relação ao GLUT, ambas as bibliotecas estão defasadas. A última atualização da biblioteca GLUT ocorreu em 1998, com a versão 3.7. Atualmente ela não se encontra mais em desenvolvimento. Já a última atualização da biblioteca FreeGLUT ocorreu em março de 2015, com a versão 3.0.0. A biblioteca FreeGLUT pode ser encontrada em <http://freeglut.sourceforge.net/>.

2.3 SDL

[Simple DirectMedia Layer \(2018\)](#) (SDL), é uma biblioteca de desenvolvimento multiplataforma desenvolvida para prover acesso de baixo nível nos *hardwares* de áudio, teclado, mouse e *joystick* por meio da OpenGL e Direct3D.

SDL possui código-aberto, sob a licença [zlib \(2017\)](#). Além disso, possui suporte nativo oficial a Windows, Mac OS X, Linux, iOS e Android. Ela é escrita totalmente na linguagem C e possui compatibilidade com C++, além de possuir versões para outras linguagens de programação, como [Java \(1995\)](#), [C# \(2000\)](#) e [Python \(1989\)](#).

2.4 Ferramentas de técnica de visualização

Atualmente existem diversas ferramentas que auxiliam na aprendizagem utilizando técnicas de visualização. As ferramentas variam entre animações interativas, linguagem de programação visual ou programação de ações de personagens. Abaixo são apresentadas algumas ferramentas que fazem o uso dessa técnica utilizando as características citadas:

- [Scratch \(2018\)](#), trata-se de uma linguagem de programação visual onde é possível programar histórias, jogos e animações interativas. Scratch ajuda os jovens a pensar de forma criativa e raciocinar sistematicamente. Por se tratar de linguagem de programação visual, ao invés de escrever linhas de códigos, o usuário arrasta blocos e os conecta, desenvolvendo assim todo o fluxo da aplicação. Foi projetada com foco no público de pessoas entre 8 e 16 anos, mas é utilizada por pessoas de todas as idades.
- [Ruby Warrior \(2018\)](#), é um jogo onde o usuário, no controle de um guerreiro, deve completar as missões usando a linguagem de programação [Ruby \(1995\)](#) como método de controle do personagem;
- [Algoritmo Raft \(2018\)](#), é um website que possui uma simulação interativa do funcionamento do algoritmo de consensus Raft. O site permite várias configurações e ações, sendo possível derrubar um nó para observar o comportamento de eleição de novo líder.

3 Desenvolvimento

3.1 Tecnologias Empregadas

3.1.1 Linguagem de programação C

A linguagem de programação C (1972) foi criada por Dennis MacAlistair Ritchie entre 1969 e 1973 na Bell Labs (1972) e trata-se de uma linguagem de propósito geral que pode ser utilizada com os paradigmas imperativo, também conhecido como procedural, e o paradigma estruturado. Segundo o site The Free Dictionary (2018), uma linguagem de propósito geral é aquela que não é limitada apenas à algum tipo de hardware específico ou para uma aplicação especializada.

A linguagem de programação C foi escolhida por ser uma das linguagens mais utilizadas para o ensino de programação nas universidades brasileiras. Uma pesquisa conduzida online em um grupo de programação chamado A.P.D.A. Associação dos Programadores Depressivos Anônimos, um fórum que concentra mais de 71.000 programadores e entusiastas brasileiros, foi obtido um resultado de que aproximadamente 55% das universidades brasileiras escolhem C como a primeira, ou uma das primeiras, linguagens para o ensino de introdução à programação, como pode-se observar na Figura 1. Este dado representa 570 votos em uma pesquisa com 1036 respostas. A linguagem de programação Java, que se ficou em segundo lugar como mais utilizada, teve apenas 169 votos, representando 16,31% do total de votos. Esse resultado pode ser encontrada no link <<https://www.facebook.com/groups/osadpa/permalink/1508699765902212/>>.

Apesar do nome caricato do grupo onde foi conduzida a pesquisa, trata-se de um grupo sério sobre programação em geral, sendo utilizado por uma vasta gama de profissionais de TI.

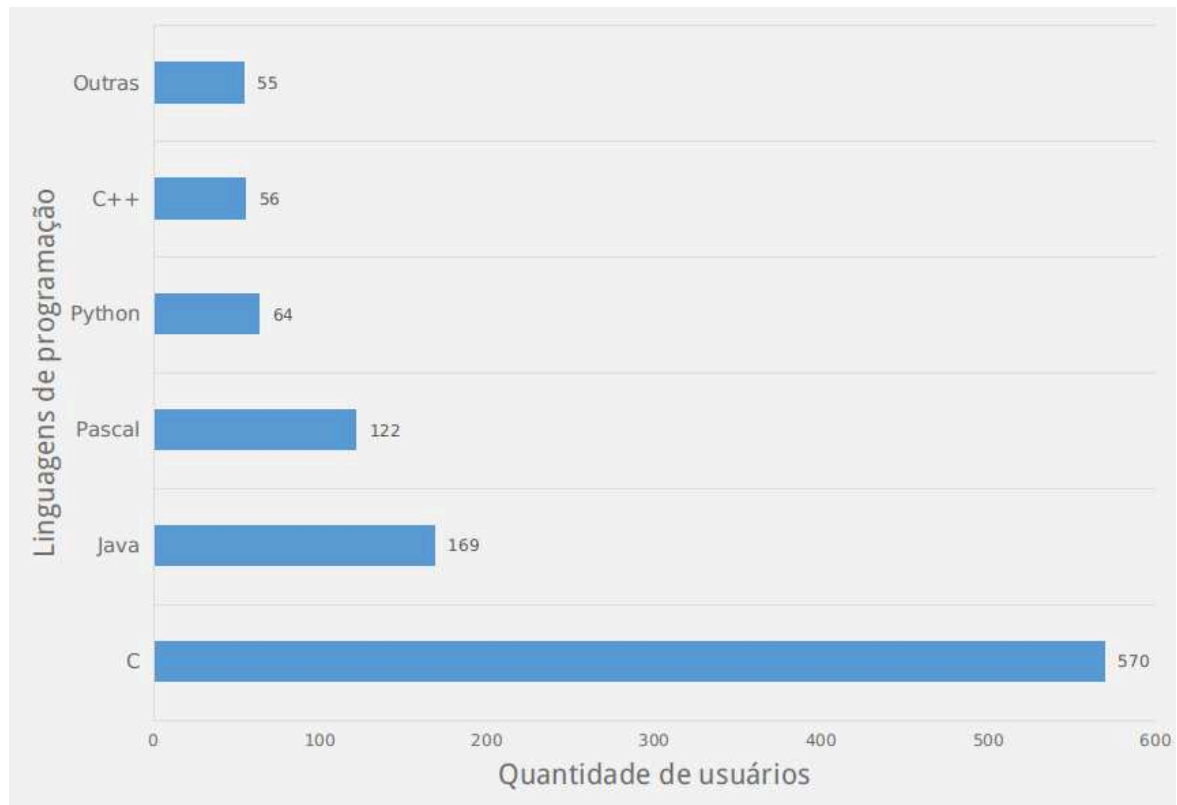


Figura 1 – Linguagens de programação utilizadas primariamente no ensino de introdução à programação

3.1.2 OpenGL

O [OpenGL \(1992\)](#) é uma biblioteca gráfica para desenvolvimento de aplicações interativas. Desde sua introdução, no ano de 1992, a biblioteca OpenGL se tornou a interface de programação de aplicação (API) mais utilizada com suporte a 2D e 3D pela indústria. O OpenGL foi escolhido como biblioteca gráfica por sua qualidade de ser altamente portátil estando presente em várias plataformas, tais como Windows, Linux, Unix, MacOS, entre outras. Outro motivo foi o fato de possuir um bom desempenho, e de poder ser utilizada em conjunto com várias linguagens de programação, dentre as quais foi escolhida a linguagem de programação [C \(1972\)](#).

A biblioteca gráfica [OpenGL \(1992\)](#) foi escolhida em detrimento de outras por ser mais simples, porém versátil, e suas já citadas qualidades: performance, largamente utilizada, interoperável entre plataformas e por ter fácil integração com a linguagem de programação C. Outros fatores que influenciaram a sua escolha foi a vasta documentação, como websites, livros e cursos online. Alguns sites que promovem a documentação de utilização do OpenGL são o próprio site do OpenGL <<https://opengl.org/>>, o website [Learn OpenGL \(2018\)](#) e o livro [OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.2](#), do autor [Woo et al. \(1999\)](#).

A forma que o OpenGL trabalha para renderizar os objetos na tela é baseada em uma máquina de estados. Cada item renderizado pelo OpenGL fica num buffer secundário e só é renderizado quando explicitamente solicitado. Ao renderizar o buffer ele então faz a troca do buffer primário pelo secundário, exibindo assim apenas o quadro renderizado por completo, evitando assim o *tearing*. *Tearing* é o efeito que ocorre quando parte do frame é renderizado utilizando informações antiga, exibindo assim parte do quadro com uma imagem e outra parte com os dados novos.

3.1.3 GLFW

De acordo com o site da [GLFW \(2018\)](#), GLFW é uma biblioteca gráfica de código-aberto, multiplataforma para desenvolvimento desktop, que utiliza OpenGL, OpenGL ES e Vulkan. Ela é responsável por criar janelas de forma unificada entre diferentes sistemas operacionais, contextos OpenGL e receber entradas de dados e eventos.

A biblioteca GLFW foi escrita em C e possui suporte nativo para Windows, macOS e vários sistemas Unix ([Unix Webiste, 1971](#)) que utilizam o sistema de janelas X Window System ([X Window System, 2018](#)), como por exemplo Linux ([Linux Website, 2018](#)) e FreeBSD ([FreeBSD, 2018](#)).

As motivações que levaram à escolha desta biblioteca foram a criação de janelas de forma transparente entre diversos sistemas operacionais, suporte para OpenGL, suporte à vários monitores e várias janelas simultâneas, suporte para mouse, teclado e joystick, além de ser a biblioteca mais atualizada, tendo vasta documentação e grande comunidade ativa. A documentação completa da biblioteca GLFW pode ser encontrada no website da biblioteca pelo endereço [<http://www.glfw.org/documentation.html>](http://www.glfw.org/documentation.html).

3.1.4 TinyCThread

A [TinyCThread \(2018\)](#) é uma biblioteca de código aberto escrita em C, multiplataforma capaz de criar e gerenciar *threads* de forma unificada e transparente, em diversos sistemas operacionais. Segundo [Tanenbaum \(2007\)](#), *thread* é uma forma de um processo dividir as tarefas a serem executadas de modo que possam ser executadas concorrentemente. Um processo que possui apenas uma *thread* para execução é chamado de *single threading*. Aplicações que possuem múltiplas *threads* são chamados de *multi threading*. *Threads* podem ser executadas em paralelo, desde que o processador possua mais de um núcleo, onde cada thread seria executada em um núcleo distinto.

O motivo da escolha da TinyCThread foi pois, apesar do padrão C11 implementar suporte nativo para criação de *threads* da mesma forma entre diferentes sistemas operacionais, os compiladores mais antigos, que ainda não implementam esse padrão, possuem formas distintas de criar e gerenciar as *threads*. Com o uso da biblioteca essa criação

fica padronizada, sendo possível compilar o mesmo código tanto para Linux quanto para Windows. A biblioteca também provê funcionalidades para gerenciamento de concorrência, como *locks* exclusivos, por exemplo, o *mutex*, e outros.

3.2 Atividades Desenvolvidas

Os artefatos gerados como resultado deste trabalho são os seguintes:

- A criação de uma biblioteca gráfica simplificada utilizando OpenGL e o framework GLFW
- A criação de uma aplicação de demonstração de utilização da biblioteca
- A documentação e manual de utilização

3.2.1 A biblioteca gráfica

A biblioteca gráfica descrita neste trabalho tem por intenção simplificar o uso da biblioteca OpenGL. Uma das formas abordadas para isto é a simplificação do gerenciamento da janela. Em uma aplicação OpenGL, o utilizador é responsável por gerenciar tanto a janela quanto o contexto, e decidir como será realizada a renderização dos objetos dentro da janela. Nesta biblioteca, apesar de ainda possuir uma janela, ela é oculta e gerenciada pela própria biblioteca. Tal abordagem reduz a flexibilização oferecida pela biblioteca OpenGL, entretanto, em troca disso, obtemos uma API bastante simplificada.

Para o correto funcionamento da biblioteca é necessário criar uma janela para realizar a renderização dos objetos; este é o padrão de funcionamento do OpenGL. Visando a simplicidade, todo o código de criação de janelas é abstraído em uma única função. Esta função realiza várias ações, como criar a janela do OpenGL, inicializar o *buffer* de objetos a serem renderizados, inicializar os *locks* de exclusão mútua do tipo *mutex* para gerenciamento de concorrência no *buffer* de objetos e a inicialização da *thread* que será responsável por toda a abstração de renderização do *buffer* de objetos na janela. Esta *thread* possui um laço de repetição onde é realizada a leitura do *buffer* de objetos contido na memória e então, renderiza os objetos de acordo com a configuração definida em cada objeto. A arquitetura proposta para a biblioteca pode ser vista na Figura 2.

O *buffer* de objetos em si também não pode ser gerenciado diretamente pelo usuário. As funções expostas pela biblioteca são responsáveis por gerenciar diretamente o *buffer*, minimizando assim a possibilidade de erros na hora de renderizar os objetos. Cada função que desenha um objeto adiciona no *buffer* de objetos a configuração correspondente ao que deve ser desenhado. Na sequência, a *thread* oculta faz o *lock* exclusivo do *buffer* para que, no ato do desenho, ele não seja alterado, e então, item a item, renderiza

todos os objetos na janela. Após o processo de renderização, é liberado o *lock* exclusivo, permitindo assim a manipulação do *buffer* de objetos. Este loop roda em torno de 60 vezes por segundo, podendo o número de execuções ser menor dependendo do hardware gráfico da máquina. Por se tratar de renderização de objetos muito básicos, espera-se que a média de quadros renderizados seja em torno de 60 FPS.

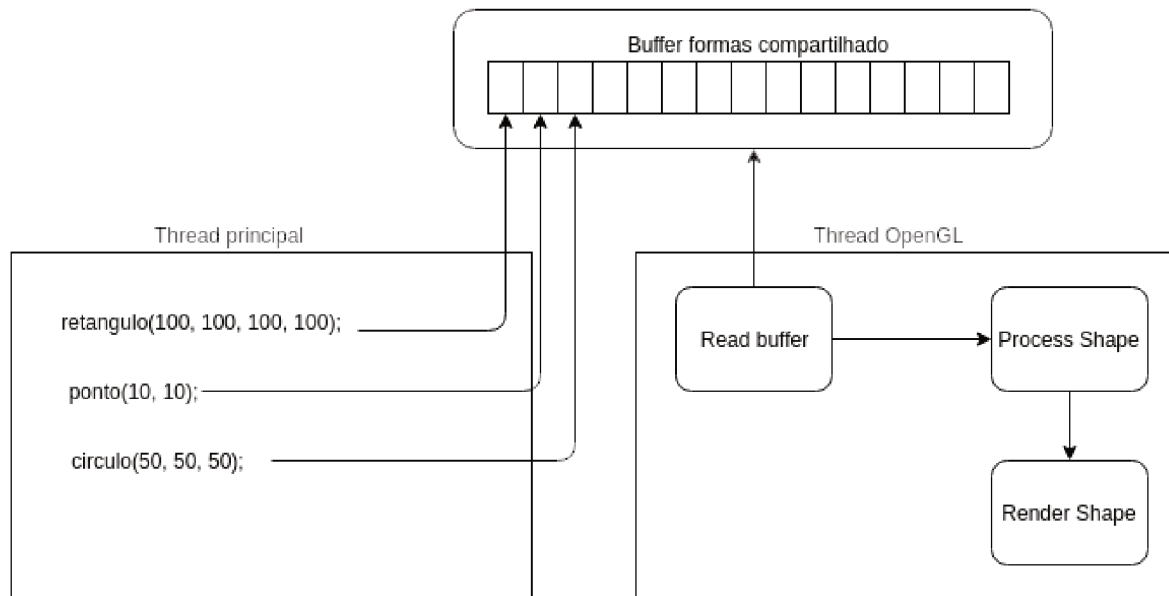


Figura 2 – Arquitetura proposta para a biblioteca

A biblioteca gráfica dispõe da API abaixo:

- `void inicializarBiblioteca(int largura, int altura);`
- `void limparTela();`
- `void desfazerUltimaForma();`
- `void refazerUltimaForma();`
- `void definirCor(int vermelho, int verde, int azul);`
- `int* obterCor();`
- `void definirEspessura(int espessura);`
- `int obterEspessura();`
- `void poligono(int numeroDeVertices, GLfloat* posicoes);`
- `void ponto(int posX, int posY);`
- `void triangulo(int posX1, int posY1, int posX2, int posY2, int posX3, int posY3);`

- `void retangulo(int posX, int posY, int largura, int altura);`
- `void quadrado(int posX, int posY, int tamanhoLado);`
- `void poligonoRegular(int posX, int posY, int raio, int faces);`
- `void circulo(int posX, int posY, int raio);`
- `void pentagono(int posX, int posY, int raio);`
- `void hexagono(int posX, int posY, int raio);`
- `void decagono(int posX, int posY, int raio);`
- `void dodecagono(int posX, int posY, int raio);`
- `void linha(int posX1, int posY1, int posX2, int posY2);`
- `void pausar(int time);`

As seções seguintes descrevem cada item da API bem como seus protótipos e o significado de cada parâmetro de entrada.

3.2.1.1 Iniciando a biblioteca

Apesar de não ter que gerenciar diretamente a janela e o laço de repetição de renderização do OpenGL manualmente, ainda é necessário inicializar a biblioteca manualmente, para disparar a *thread* secundária, responsável pela renderização dos objetos, e a criação da janela onde serão renderizadas os objetos.

A função que inicializa a biblioteca tem o seguinte protótipo:

```
void inicializarBiblioteca(int largura, int altura);
```

Os parâmetros de entrada são:

- `int largura` - a largura do tamanho da janela a ser criada.
- `int altura` - altura do tamanho da janela a ser criada.

O exemplo abaixo mostra como inicializar a biblioteca em uma aplicação C e o resultado pode ser observado na Figura 3:

```
#include <stdio.h>
#include <graphics.h>

int main() {
    inicializarBiblioteca(800, 600);

    retangulo(0, 0, 200, 100);
    limparTela();

    getchar();
    return 0;
}
```

Exemplo 1: Inicializando a biblioteca

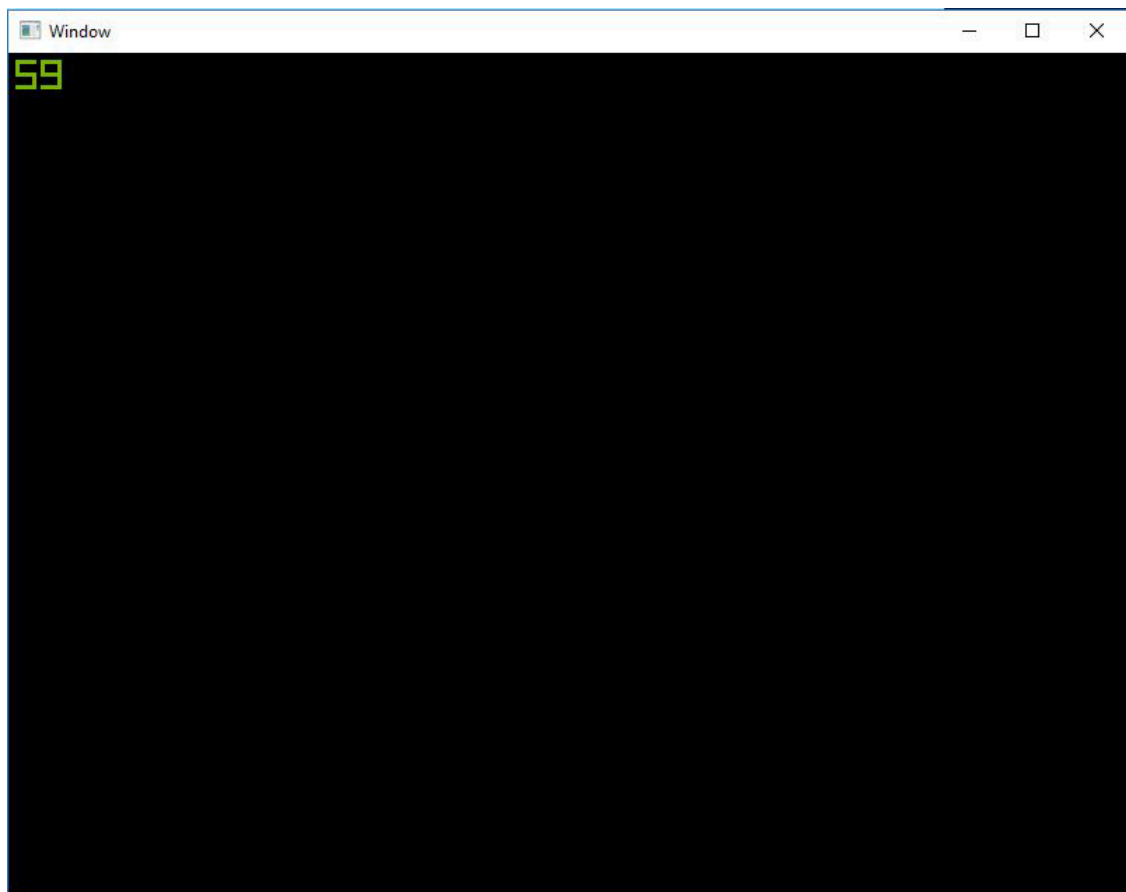


Figura 3 – Exemplo inicialização da biblioteca

3.2.1.2 Limpando a tela

Uma das funcionalidades propostas é que a janela seja limpa, removendo assim todos os objetos renderizados e reiniciando a janela para seu estado inicial.

A função que limpa a janela já foi utilizada anteriormente e pode ser encontrada no Exemplo 1.

3.2.1.3 Desfazer último desenho

Em uma aplicação, é comum se arrepender de sua última ação e portanto, querer desfaze-la. A biblioteca aqui implementada possui uma função específica para esta ação. Ela pode ser utilizada múltiplas vezes e a cada vez um objeto é removido da tela.

A função que desfaz o último desenho tem o seguinte protótipo:

```
void desfazerUltimaForma();
```

3.2.1.4 Refazer o último desenho

Refazer o último desenho é uma opção da mesma forma que desfazer o último desenho. Sendo assim, é possível se arrepender e desfazer a opção de desfazer. A cada vez que a função for chamada, um desenho será restaurado. A ordem de restauração é baseada na estrutura de dados pilha, portanto o último objeto removido será o primeiro objeto restaurado. Essa função só pode ser executada caso invocada imediatamente após a função de desfazer, pois caso seja inserida um novo objeto após ter desfeito o último desenho, a pilha de ações será descartada.

A função que refaz o último desenho tem o seguinte protótipo:

```
void refazerUltimaForma();
```

3.2.1.5 Alterar a cor de preenchimento de um objeto

A grande maioria, senão todas, as bibliotecas citadas nos trabalhos correlatos permitem a definição de uma cor de preenchimento para o objeto a ser desenhado.

A função que permite alterar a cor de um objeto tem o seguinte protótipo:

```
void definirCor(int vermelho, int verde, int azul);
```

Os parâmetros de entrada são:

- int vermelho - um inteiro de 0 a 255 representando a tonalidade de vermelho do padrão RGB.
- int verde - um inteiro de 0 a 255 representando a tonalidade de verde do padrão RGB.
- int azul - um inteiro de 0 a 255 representando a tonalidade de azul do padrão RGB.

3.2.1.6 Obter a cor de preenchimento do objeto

Assim como podemos definir a cor de preenchimento de um objeto, também é possível obtermos a cor de preenchimento atual. Esta função se faz útil quando, por exemplo, queremos criar um objeto de uma nova cor mas queremos manter a cor antiga, porém não é conhecido a cor de preenchimento atual. Assim como visto no item 3.2.1.5, como a função *definirCor* recebe 3 argumentos de entrada, a saída é um ponteiro representando um array de inteiros de tamanho 3, representando as cores RGB.

A função que obtém a cor atual de uma forma tem o seguinte protótipo:

```
int* obterCor();
```

A saída do método é um ponteiro para um array contendo 3 posições e seus valores são descritos conforme a lista abaixo:

- int cor[0] - um inteiro de 0 a 255 representando a tonalidade de vermelho do padrão RGB.
- int cor[1] - um inteiro de 0 a 255 representando a tonalidade de verde do padrão RGB.
- int cor[2] - um inteiro de 0 a 255 representando a tonalidade de azul do padrão RGB.

O exemplo abaixo mostra como obter a cor atual de uma forma em uma aplicação C:

```
#include <stdio.h>
#include <graphics.h>

int main() {
    inicializarBiblioteca(800, 600);

    definirCor(255, 0, 127);
    retangulo(0, 0, 200, 100);

    int* corAtual = obterCor();

    getchar();
    return 0;
}
```

3.2.1.7 Definindo o tamanho de um objeto

Alguns objetos podem ter seu tamanho alterado. Para que tenha efeito, a função *definirTamanho* deve ser invocada anteriormente a uma função de desenho de forma. Após chamado o método, todos os objetos desenhados após a chamada serão desenhados com o novo tamanho definido. Atualmente apenas o objeto *ponto* pode ter seu tamanho alterado.

A função que define o tamanho de uma forma tem o seguinte protótipo:

```
void definirTamanho(int tamanho);
```

O exemplo abaixo mostra como definir o tamanho de uma forma em uma aplicação C:

```
#include <stdio.h>
#include <graphics.h>

int main() {
    inicializarBiblioteca(800, 600);

    definirTamanho(50);
    ponto(100, 100);

    getchar();
    return 0;
}
```

3.2.1.8 Obtendo o tamanho de um objeto

Da mesma forma que é possível definir o tamanho de um objeto, esta função permite obter o valor atual do tamanho de desenho de objetos. Esta função é útil quando se quer desenhar um objeto em novo tamanho e deseja retornar o tamanho anterior para os próximos objetos e não existe a informação de qual o valor atual do tamanho da forma.

A função de como obter o tamanho atual de uma forma tem o seguinte protótipo:

```
int obterTamanho();
```

O exemplo abaixo mostra como obter o tamanho atual de uma forma em uma aplicação C:

```
#include <stdio.h>
#include <graphics.h>
```

```
int main() {
    inicializarBiblioteca(800, 600);

    definirTamanho(50);
    ponto(100, 100);

    int tamanhoAtual = obterTamanho();

    getchar();
    return 0;
}
```

3.2.1.9 Desenhando um polígono

Um polígono é uma figura geométrica composta por 3 ou mais vértices. Um polígono pode ser desenhado de forma livre e a função que desenha um polígono tem o seguinte protótipo:

```
void poligono(int numeroDeVertices, GLfloat* posicoes);
```

Os parâmetros de entrada são:

- int numeroDeVertices - número de vértices que o seu polígono possui
- GLfloat* posicoes - um array contendo as posições de cada vértice. O array deve possuir um tamanho par definido pela fórmula $tamanho = 2 \times numeroDeVertices$, pois as posições pares representam as posições X dos vértices e as posições ímpares representam as posições Y, par a par. Os pares devem ser definidos sequencialmente. Para desenhar um triângulo, seria então necessário um array de 6 posições. Utilizando um triângulo como exemplo, vamos utilizar os pontos (0, 0), (200, 0) e (100, 200). O array de posições então seria `GLfloat posicoes[6] = {0, 0, 200, 0, 100, 200};`.

O exemplo abaixo mostra como desenhando um polígono em uma aplicação C:

```
#include <stdio.h>
#include <stdlib.h>
#include <graphics.h>

int main() {
    inicializarBiblioteca(800, 600);
```

```
GLfloat posicoes[] = {  
    320, 100,  
    400, 300,  
    800, 50,  
    520, 10,  
    340, 40  
};  
poligono(5, posicoes);  
  
getchar();  
return 0;  
}
```

3.2.1.10 Desenhando um ponto

A função que desenha um ponto tem o seguinte protótipo:

```
void ponto(int posX, int posY);
```

Os parâmetros de entrada são:

- int posX - a posição X do ponto.
- int posY - a posição Y do ponto.

3.2.1.11 Desenhando um triângulo

Triângulos são uma das formas mais básicas e utilizadas, pois todos os polígonos convexos pode ser decompostos em vários triângulos. A função que desenha um triângulo tem o seguinte protótipo:

```
void triangulo(  
    int posX1, int posY1,  
    int posX2, int posY2,  
    int posX3, int posY3  
);
```

Os parâmetros de entrada são:

- int posX1 - a posição X da vértice 1 do triângulo.

- int posY1 - a posição Y da vértice 1 do triângulo.
- int posX2 - a posição X da vértice 2 do triângulo.
- int posY2 - a posição Y da vértice 2 do triângulo.
- int posX3 - a posição X da vértice 3 do triângulo.
- int posY3 - a posição Y da vértice 3 do triângulo.

3.2.1.12 Desenhando um retângulo

Retângulos são formas geométricas que possuem 4 ângulos retos. Por serem uma das mais básicas formas, são bastante utilizados

A função que desenha um retângulo tem o seguinte protótipo:

```
void retangulo(int posX, int posY, int largura, int altura);
```

Os parâmetros de entrada são:

- int posX - posição X do retângulo. Representa o vértice inferior esquerdo do retângulo.
- int posY - posição y do retângulo. Representa o vértice inferior esquerdo do retângulo.
- int altura - altura do triângulo.
- int largura - largura do triângulo.

3.2.1.13 Desenhando um quadrado

Um quadrado é um polígono considerado especial, pois trata-se de um retângulo que possui largura e altura de mesmo tamanho.

A função que desenha um quadrado tem o seguinte protótipo:

```
void quadrado(int posX, int posY, int tamanhoLado);
```

Os parâmetros de entrada são:

- int posX - posição X do quadrado. Representa o vértice inferior esquerdo do quadrado.

- `int posY` - posição Y do quadrado. Representa o vértice inferior esquerdo do quadrado.
- `int tamanhoLado` - tamanho do lado do quadrado.

3.2.1.14 Desenhando um polígono regular

Um polígono regular é um polígono convexo com n faces onde cada face possui o mesmo tamanho e todos os ângulos internos são iguais. Alguns exemplos de polígonos regulares são triângulos retângulos, quadrados, pentágonos, hexágonos, dentre outros. Algumas dessas formas você pode ver na Figura 4.

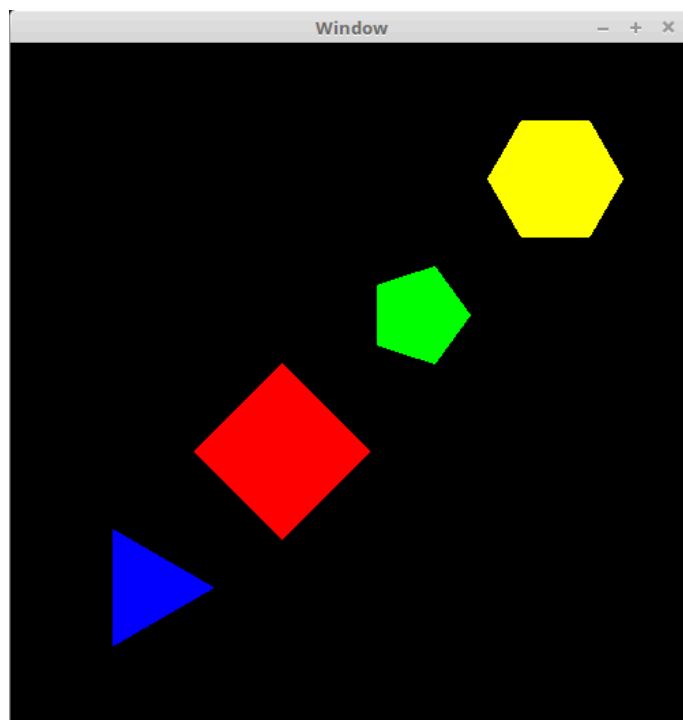


Figura 4 – Exemplos de polígonos regulares: Triângulo, quadrado, pentágono e hexágono

Segundo [Silva \(2018\)](#), “um polígono é convexo quando todos os pontos de um segmento de reta que possui as extremidades no interior do polígono também estão dentro dele. Sendo assim, se for possível encontrar pelo menos um segmento de reta que possui as extremidades dentro do polígono e, ao mesmo tempo, um ponto fora dele, esse polígono não será convexo”.

A função que desenha um polígono regular tem o seguinte protótipo:

```
void poligonoRegular(int posX, int posY, int raio, int faces);
```

Os parâmetros de entrada são:

- int posX - posição X onde do polígono regular. Representa o centro de onde será desenhado o objeto.
- int posY - posição Y onde do polígono regular. Representa o centro de onde será desenhado o objeto.
- int raio - tamanho do raio que do polígono regular.
- int faces - o número de faces que do polígono regular.

O exemplo abaixo mostra como desenhar um polígono regular azul em uma aplicação C:

```
#include <stdio.h>
#include <stdlib.h>
#include <graphics.h>

int main() {
    inicializarBiblioteca(800, 600);

    definirCor(0, 0, 255);
    poligonoRegular(300, 300, 100, 7);

    getchar();
    return 0;
}
```

3.2.1.15 Desenhando um círculo

O círculo é uma forma geométrica que possui virtualmente infinitos vértices.

A função que desenha um círculo tem o seguinte protótipo:

```
void circulo(int posX, int posY, int raio);
```

Os parâmetros de entrada são:

- int posX - posição X do centro do círculo.
- int posY - posição Y do centro do círculo.
- int raio - raio do círculo.

3.2.1.16 Desenhando um pentágono

Um pentágono é um polígono regular composto por 5 vértices.

A função que desenha um pentágono tem o seguinte protótipo:

```
void pentagono(int posX, int posY, int raio);
```

Os parâmetros de entrada são:

- int posX - posição X do centro do pentágono.
- int posY - posição Y do centro do pentágono.
- int raio - raio do pentágono.

3.2.1.17 Desenhando uma linha

Dado 2 pontos A e B, uma linha é um segmento de reta que liga os pontos A e B.

A função que desenha uma linha tem o seguinte protótipo:

```
void linha(int posX1, int posY1, int posX2, int posY2);
```

Os parâmetros de entrada são:

- int posX1 - posição do X1 da linha.
- int posY1 - posição do Y1 da linha.
- int posX2 - posição do X2 da linha.
- int posY2 - posição do Y2 da linha.

O exemplo abaixo mostra como desenhar uma linha que corta a janela de desenho verticalmente em uma aplicação C:

```
#include <stdio.h>
#include <graphics.h>

int main() {
    inicializarBiblioteca(1366, 768);

    linha(
```

```
    0, 0,  
    1366, 768  
);  
  
getchar();  
return 0;  
}
```

3.3 Estado Atual do Desenvolvimento

No estado atual do desenvolvimento todas as funcionalidades previstas foram implementadas e funcionam de forma portátil e multiplataforma. Todas as formas implementadas podem ser observadas na Figura 5.

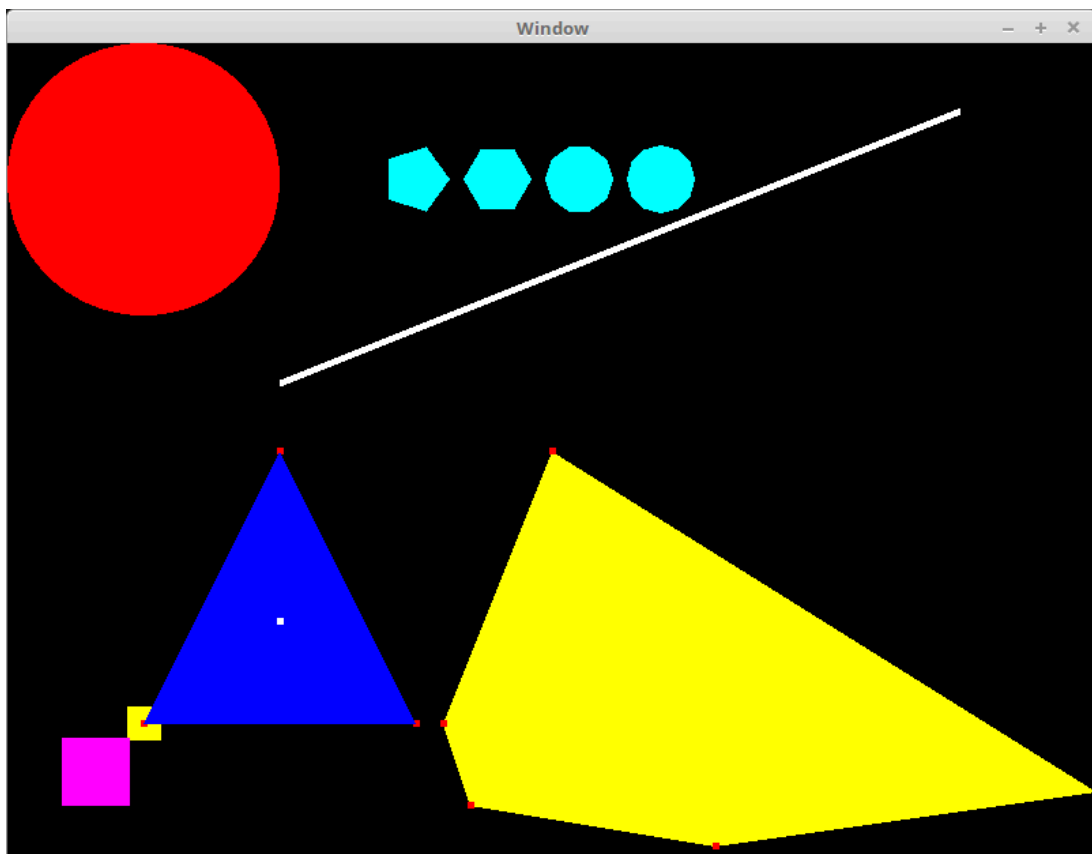


Figura 5 – Exemplo: Todas as formas implementadas

Como todas as funcionalidades foram implementadas, os usuários são capazes de desenhar objetos e cenas que sejam baseados nas formas implementadas. Um exemplo de uso mostrando como desenhar um relógio funcional utilizando a biblioteca desenvolvida neste projeto, utilizando linguagem de programação C, pode ser visto na Figura 6.

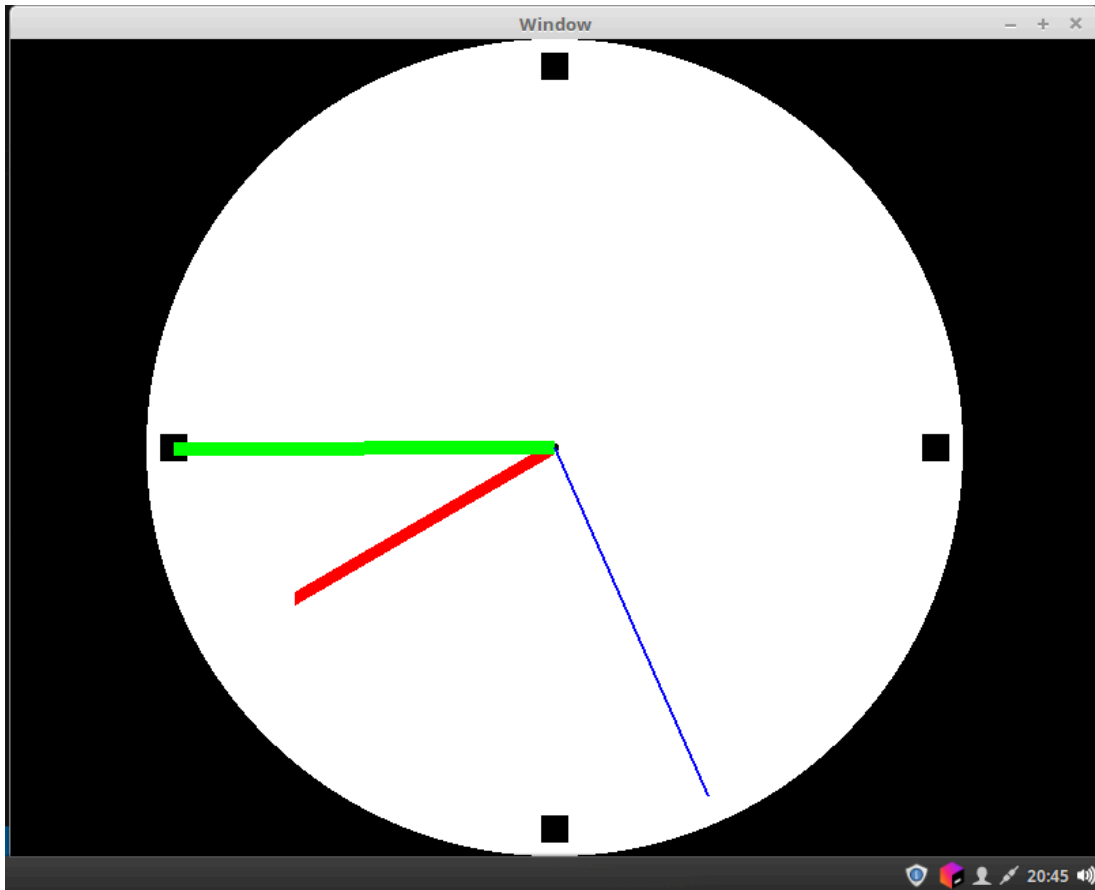


Figura 6 – Exemplo: Relógio analógico

Um ponto muito importante é a ausência de testes automatizados. Toda alteração em softwares que não possuem testes automatizados correm o risco de quebrar a compatibilidade com versões anteriores ou em evoluções de manutenção corretiva. Logo, os testes devem ser realizados pelo usuário, no projeto que estiver sendo desenvolvido utilizando esta biblioteca.

O código-fonte da biblioteca descrita neste projeto, bem como a documentação, o guia de uso e alguns exemplos de utilização podem ser encontrados na lista abaixo:

- <https://github.com/williamokano/tcc/tree/master/src>: Contém o código-fonte da biblioteca. Este código deve ser compilado para gerar o arquivo biblioteca a ser importado em seu projeto;
- <https://github.com/williamokano/tcc/tree/master/headers>: Contém os arquivos cabeçalhos para utilização em conjunto com a biblioteca gráfica;
- <https://github.com/williamokano/tcc/tree/master/documentacao>: Contém os arquivos de documentação da biblioteca, onde é explicado cada função e seus parâmetros;

- <https://github.com/williamokano/tcc/tree/master/guia>: Contém um guia ilustrado mostrando como utilizar a biblioteca em ambientes Windows e Linux;
- <https://github.com/williamokano/tcc/tree/master/exemplos>: Contém alguns exemplos criados utilizando a biblioteca. Os exemplos incluem, mas não se limitam a, uma figura se movendo pela tela, um relógio, desenhos na tela utilizando entradas fornecidas pelo usuário.

4 Considerações Finais

4.1 Trabalhos Futuros

Como trabalho futuro, um dos principais pontos a ser trabalho é o uso da biblioteca aqui descrita em salas de aula. Elaborar algum método de avaliação capaz de medir a assimilação do conteúdo pela utilização da biblioteca e verificar se ela realmente gera os resultados esperados.

Como dito na Seção 3.1.1, os dados obtidos para esta pesquisa foram provenientes de um grupo de programação online. Assim, uma sugestão para trabalhos futuros seria refazer esta mesma pesquisa utilizando dados de faculdades e universidades brasileiras de ensino de Ciência da Computação, Sistemas de Informação ou cursos similares. Com esta abordagem, pode-se obter valores mais precisos sobre a real utilização da linguagem C como linguagem de cursos introdutórios de programação.

Outro ponto interessante seria a adição de novas funcionalidades de desenho, como elipses, arcos, linhas curvas e textos. Com essas novas funcionalidades o usuário teria um novo leque de funcionalidades como, por exemplo, a criação, de forma bem simplificada, de gráficos de barras e de linhas. A habilidade de poder se expressar por texto, além das formas geométricas, seria uma excelente adição para os próximos trabalhos.

Com o intuito de deixar a biblioteca mais segura, o desenvolvimento de testes automatizados podem ser incluídos, a fim de garantir que a compatibilidade não seja quebrada entre versões ou que, caso isso ocorra, que tenha sido proposital.

Outra possibilidade seria evoluir a biblioteca de forma que deixe ser apenas para formas geométricas e possa começar a importar recursos externos, como por exemplo imagens e sons. Com essa evolução, a biblioteca ficaria mais dinâmica e possibilitaria um novo leque de possibilidades ao usuário, como o desenvolvimento de jogos simples.

4.2 Dificuldades Encontradas

Inicialmente uma das dificuldades encontradas foi conseguir compilar a biblioteca de uma forma que fosse simples de utilizar tanto no sistema operacional Linux quanto no Windows. A primeira abordagem foi utilizar o CMake (2000), pois ele possui o mesmo suporte de macros que a linguagem C, portanto deveria ser fácil a compilação multi-plataforma. Entretanto, algumas macros, como a `#__WIN32`, que indica se o sistema operacional é Windows, nem sempre é adicionada no momento da compilação. Com isso, a possibilidade de utilizar CMake como ferramenta de compilação foi excluída.

A ideia inicial deste projeto era criar uma biblioteca que, além de ser simples, não necessita-se de outras bibliotecas complementares para o gerenciamento de janelas e contextos do OpenGL. Entretanto, mostrou-se muito difícil realizar essas implementações, pois o sistema de janelas dos sistemas operacionais Windows e Linux são completamente diferentes. No sistema operacional Windows, para o gerenciamento de janelas, é utilizado a API do Windows, através da biblioteca win32.dll. Já no sistema operacional Linux, todo o sistema de gerenciamento de janelas é realizado através da biblioteca X11. Essa gestão de janelas é muito complexa, mesmo quando estamos desenvolvendo exclusivamente para um sistema operacional, portanto foi utilizado a biblioteca GLFW para gerenciar as janelas de forma portátil e multiplataforma.

Mesmo utilizando a biblioteca GLFW, a compilação multiplataforma não foi direta. No sistema operacional Windows a biblioteca GLFW é compilada antes da biblioteca descrita neste trabalho, e logo após é realizado o link estático entre as duas bibliotecas. A estratégia utilizada no sistema operacional Linux foi instalar a biblioteca GLFW através de um gerenciador de pacotes e então realizar o link compartilhado no ato da compilação da biblioteca deste projeto.

Outra dificuldade encontrada foi a compilação da biblioteca TinyCThread no sistema operacional Windows. Inicialmente o compilador utilizado era o mingw, um kit de ferramentas para Windows contento o compilador GCC e algumas bibliotecas previamente instalas. Este compilador foi instalado automaticamente junto com a IDE [Code::Blocks \(2005\)](#). Entretanto a versão instalada com essa IDE não é suportada pela TinyCThread. A versão recomendada do mingw para compilar a biblioteca TinyCThread no sistema operacional Windows é a versão mingw64. Após instalar essa versão, o problema foi sanado.

Conclusão

Este trabalho apresentou uma biblioteca para auxiliar o aluno a assimilar melhor os códigos ensinados em cursos introdutórios de programação, utilizando técnicas de visualização. Os problemas causados pela falta de assimilação dos conceitos ensinados nos cursos de programação pode levar o aluno a ter um atraso em outras disciplinas onde o conhecimento de programação é requisito, como por exemplo, na disciplina de estrutura de dados. Além do mais, alunos que não conseguem assimilar o conhecimento introdutório de programação podem acabar se frustrando e abandonando o curso, aumentando assim o índice de evasão.

Para poder realizar este trabalho, foi necessário uma base teórica multidisciplinar, onde a maioria do conhecimento foi obtido através de disciplinas oferecidas pelo curso de Sistemas de Informação. As disciplinas de Modelagem de Software, Engenharia de Software, Sistemas Operacionais e Estrutura de Dados foram primordiais para que fosse desenvolvida uma arquitetura simples e eficiente. Conceitos como *multi threading*, *locks* de exclusão mútua, ensinados em Sistemas Operacionais, foram essenciais para o desenvolvimento desta biblioteca. Os conhecimentos ensinados em Estrutura de Dados, como listas e pilhas, também foram utilizados para o desenvolvimento da biblioteca, seja para o armazenamento dos objetos em um *buffer*, que é uma lista, ou então a utilização de uma pilha, para que pudesse ser implementado o conceito de desfazer e refazer uma ação. Além do mais, os conhecimentos obtidos em Modelagem de Software e Engenharia de Software foram de grande ajuda para que fosse possível entender o problema e modelar a arquitetura para que fosse obtida uma API simples e de fácil utilização.

Referências

Algoritmo Raft. *Algoritmo Raft*. 2018. Disponível em: <<https://raft.github.io/>>. Acesso em: 12/07/2018. Citado na página 14.

Bell Labs. *Bell Labs*. 1972. Disponível em: <<https://www.bell-labs.com/>>. Acesso em: 15/06/2018. Citado na página 15.

Borland Graphics Interface (BGI) for Windows. *Borland Graphics Interface (BGI) for Windows*. 2004. Disponível em: <<https://www.cs.colorado.edu/~main/bgi/doc/>>. Acesso em: 12/07/2018. Citado na página 13.

C. *C Programming Language*. 1972. Disponível em: <[https://en.wikipedia.org/wiki/C_\(programming_language\)](https://en.wikipedia.org/wiki/C_(programming_language))>. Acesso em: 15/06/2018. Citado 2 vezes nas páginas 15 e 16.

C++. *About C++*. 1979. Disponível em: <<http://www.cplusplus.com/info/history/>>. Acesso em: 13/07/2018. Citado na página 10.

C#. *About C#*. 2000. Disponível em: <<https://docs.microsoft.com/pt-br/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework>>. Acesso em: 12/07/2018. Citado na página 14.

CMake. *CMake*. 2000. Disponível em: <<https://cmake.org/overview/>>. Acesso em: 13/07/2018. Citado na página 34.

Code::Blocks. *Code::Blocks*. 2005. Disponível em: <<http://www.codeblocks.org/>>. Acesso em: 13/07/2018. Citado na página 35.

FreeBSD. *FreeBSD*. 2018. Disponível em: <<https://www.freebsd.org/about.html>>. Acesso em: 15/06/2018. Citado na página 17.

FreeGLUT. *The FreeGLUT project*. 1999. Disponível em: <<http://freeglut.sourceforge.net/>>. Acesso em: 12/07/2018. Citado na página 13.

Freeware BGI Drivers. *Freeware BGI Drivers*. 2008. Disponível em: <<https://www.von-bassewitz.de/uz/bgi.php>>. Acesso em: 12/07/2018. Citado na página 13.

GLFW. *GLFW*. 2018. Disponível em: <<http://www.glfw.org/>>. Acesso em: 15/06/2018. Citado na página 17.

GLUT. *GLUT - The OpenGL Utility Toolkit*. 2000. Disponível em: <<https://www.opengl.org/resources/libraries/glut/>>. Acesso em: 12/07/2018. Citado na página 13.

Java. *About Java*. 1995. Disponível em: <https://www.java.com/pt_BR/about/>. Acesso em: 12/07/2018. Citado 2 vezes nas páginas 10 e 14.

KIRBY, S.; TOLAND, B.; DEEGAN, C. Program visualization tool for teaching programming in c. In: . [S.l.: s.n.], 2010. Citado na página 10.

- Learn OpenGL. *Learn OpenGL*. 2018. Disponível em: <<https://learnopengl.com/>>. Acesso em: 22/06/2018. Citado na página 16.
- Linux Website. *Linux Website*. 2018. Disponível em: <<https://www.linux.org/articles/>>. Acesso em: 15/06/2018. Citado na página 17.
- MAYER, R. E. The psychology of how novices learn computer programming. *ACM Computing Surveys (CSUR)*, ACM, v. 13, n. 1, 1981. Citado na página 10.
- OPENGL. *About OpenGL*. 1992. Disponível em: <<https://opengl.org/about/>>. Acesso em: 15/06/2018. Citado na página 16.
- Python. *About Python*. 1989. Disponível em: <<https://www.python.org/about/>>. Acesso em: 12/07/2018. Citado na página 14.
- Ruby. *About Ruby*. 1995. Disponível em: <<https://www.ruby-lang.org/pt/about/>>. Acesso em: 12/07/2018. Citado 2 vezes nas páginas 10 e 14.
- Ruby Warrior. *Ruby Warrior*. 2018. Disponível em: <https://www.bloc.io/ruby-warrior#>. Acesso em: 12/07/2018. Citado 2 vezes nas páginas 10 e 14.
- SANTIAGO, R. de; DAZZI, R. L. S. Ferramenta de apoio ao ensino de algoritmos. 2004. Citado na página 10.
- Scratch. *Scratch*. 2018. Disponível em: <<https://scratch.mit.edu/about/>>. Acesso em: 12/07/2018. Citado 2 vezes nas páginas 10 e 14.
- SILVA, L. P. M. "Polígonos convexos"; *Brasil Escola*. 2018. Disponível em: <<https://brasilecola.uol.com.br/matematica/poligonos-convexos.htm>>. Acesso em: 28/06/2018. Citado na página 28.
- Simple DirectMedia Layer. *Simple DirectMedia Layer*. 2018. Disponível em: <<https://www.libsdl.org/>>. Acesso em: 12/07/2018. Citado na página 14.
- SOUZA, E. M. S. d.; DAZZI, R. L. S. Sistema de apoio ao ensino de lógica e algoritmo. In: *VI Congreso Argentino de Ciencias de la Computación*. [S.l.: s.n.], 2000. Citado na página 10.
- TANENBAUM, A. S. *Modern Operating Systems*. 3rd. ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2007. ISBN 9780136006633. Citado na página 17.
- The Free Dictionary. *Linguagem de propósito geral*. 2018. Disponível em: <<https://encyclopedia2.thefreedictionary.com/general-purpose+language>>. Acesso em: 15/06/2018. Citado na página 15.
- TinyCThread. *TinyCThread*. 2018. Disponível em: <<https://tinycthread.github.io/>>. Acesso em: 19/06/2018. Citado na página 17.
- Unix Webiste. *Unix*. 1971. Disponível em: <<http://www.unix.org/>>. Acesso em: 15/06/2018. Citado na página 17.
- WOO, M. et al. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.2*. 3rd. ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999. ISBN 0201604582. Citado na página 16.

X Window System. *X Window System*. 2018. Disponível em: <<https://www.x.org/wiki/>>. Acesso em: 15/06/2018. Citado na página 17.

zlib. *zlib licence*. 2017. Disponível em: <https://www.zlib.net/zlib_license.html>. Acesso em: 12/07/2018. Citado na página 14.