UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMÁTICA

Disciplina: **Introdução à Computação Gráfica**
Professor: **Maelso B. Pacheco (email: maelso.bruno@gmail.com)**.
Data de entrega: **19/09/2022**.

Atividade Prática – Rasterizando Linhas

Objetivo

O objetivo deste trabalho é familiarizar os alunos com os algoritmos de rasterização utilizados em computação gráfica.

Atividade

Nesta atividade os alunos deverão implementar algoritmos para a rasterização de pontos e linhas. Triângulos deverão ser desenhados através da rasterização das linhas que compõem suas arestas. A rasterização destas primitivas será feita simulando o acesso direto à memória de vídeo. Como os sistemas operacionais atuais protegem a memória quanto ao acesso direto, os alunos utilizarão um *framework*, fornecido pelo professor, que simula o acesso à memória de vídeo.

O *Framework*

Estrutura

Este *framework* simula o acesso direto à memória de vídeo. Os seus arquivos podem ser acessados no repositório https://gitlab.com/maelso/icg/-/tree/master/my_gl_framework, disponibilizado pelo professor.

Abaixo segue a lista de arquivos que compõem este *framework*:

- `main.cpp`
- `main.h`
- definitions.h
- `Makefile`
- `mygl.h`
- `mygl.cpp`

O arquivo `definitions.h` contém macros que determinam as dimensões da janela em pixels e declara o ponteiro `FBptr`, que aponta para o primeiro *byte* do *color buffer*. A posição apontada por `FBptr` corresponde ao pixel de coordenadas (0,0), localizado no canto superior esquerdo da janela. Cada pixel possui 4 componentes de cor (*Red*, *Green*, *Blue*, *Alpha*), cada uma representada por 1 *byte* (`unsigned char`).

Os arquivos `main.h` e `main.cpp` definem funções e variáveis necessárias à simulação de acesso à memória de vídeo. Este *framework* é acompanhado também de um *script* `Makefile` que serve como sugestão de procedimento de compilação para sistemas Unix. A

compilação do *framework* é, de qualquer forma, responsabilidade do aluno.

Os arquivos `mygl.h` e `mygl.cpp` são os únicos arquivos que devem ser alterados durante a realização deste exercício. O arquivo `mygl.h` contém a declaração da função `MyGlDraw()`, responsável por invocar as funções de rasterização que os alunos irão desenvolver. É neste arquivo que os alunos também deverão declarar as funções a serem desenvolvidas. O arquivo `mygl.cpp` é o lugar onde serão definidas as funções de rasterização. Este arquivo contém também a definição da função `MyGlDraw()`, cujo corpo o aluno deverá alterar de forma que suas funções de rasterização sejam devidamente invocadas.

Dependências

A compilação do projeto exige que os cabeçalhos do OpenGL e a GLUT (*The OpenGL Toolkit*) estejam instalados. O documento `guia-de-instalacao.pdf`, que se encontra na pasta raíz do projeto, no repositório, pode ser usado como referência para instalação das dependências.

Desenvolvimento

Neste trabalho os alunos deverão desenvolver, ao menos, as três funções abaixo:

- **`PutPixel`**`(...)`: Rasteriza um ponto na memória de vídeo recebendo como parâmetros as coordenadas (x,y) do pixel na tela e sua cor (RGBA).

- **`DrawLine`**`(...)`: Rasteriza uma linha na tela, recebendo como parâmetros as coordenadas dos seus vértices inicial e final (representados respectivamente pelas tuplas (x0,y0) e (x1,y1)) e as cores (no formato RGBA) de cada vértice. O algoritmo de rasterização de linha a ser implementado deve ser o **Algoritmo de Bresenham!**

- **`DrawTriangle`**`(...)`: Função que desenha as arestas de um triângulo na tela, recebendo como parâmetros as posições dos três vértices (x0,y0), (x1,y1) e (x2,y2) bem como as cores (RGBA) de cada um dos vértices. **Não é necessário o preenchimento do triângulo!**

IMPORTANTE: As funções devem ser escritas em C++. Não é permitido o uso de nenhuma biblioteca externa a não ser as que já acompanham o projeto.

Dica

Os alunos são incentivados a utilizarem classes, ou *structs*, na descrição das primitivas (ponto, linha e triângulo). A utilização destas estruturas permite, por exemplo, uma melhor modularização do código e a redução do número de parâmetros a ser passado às funções.

Entrega

Os trabalhos devem ser entregues até o dia **19/09/2022**, impreterivelmente até as **23**

horas e 55 minutos. **Este trabalho pode ser desenvolvido em grupos de 3 alunos**.

Deve ser enviado o código fonte do trabalho e um relatório das atividades e resultados.

Exclusivamente para esta atividade, o envio deverá ser feito por e-mail para maelso.bruno@gmail.com. Emails não enviados, salvos como rascunhos, caixa de saída, etc, serão considerados como atrasos.

No relatório deverá constar os *printscreens* dos resultados (acompanhados de todos os parâmetros utilizados na sua geração) acompanhados de pequenos textos (frases ou parágrafos) explicativos. Abaixo segue um detalhamento de cada um dos componentes que devem aparecer no relatório.

- ***Printscreens***: Devem demonstrar a evolução do processo de implementação, incluindo eventuais problemas encontrados, e os resultados obtidos após suas correções.

- **Texto**:
  - Deve iniciar com um parágrafo que descreva a atividade desenvolvida.
  - Deve explicar brevemente as estratégias adotadas pelo aluno na resolução da atividade (estruturas de dados utilizadas e funções desenvolvidas).
  - Breve discussão sobre os resultados gerados, dificuldades e possíveis melhorias.
  - Listar as referência bibliográficas consultadas durante o desenvolvimento do trabalho (livros, artigos, endereços web), se for o caso.
  - **OBS: A inclusão de trechos de código no corpo do post deve se limitar ao mínimo indispensável para o correto entendimento do texto!**

Avaliação

A avaliação dos trabalhos levará em consideração os seguintes critérios:

- Completude do trabalho (se atendeu tudo o que foi solicitado).
- Clareza, organização, linguagem e capacidade de síntese do texto.
- Embasamento técnico dos argumentos apresentados.
- Eficiência e eficácia das soluções apresentadas.

Importante:

Cada dia de atraso na entrega do trabalho acarretará em um desconto de 10% na nota atual do trabalho. Por exemplo, considerando-se um trabalho avaliado inicialmente com nota 10:

1. Se o trabalho for entregue no prazo, o aluno receberá nota 10;
2. Se o trabalho for entregue com 1 dia de atraso, os(as) integrantes receberá(ão) nota 9 (= 10−10%);
3. Se o trabalho for entregue com 2 dias de atraso, os(as) integrantes receberão nota 8,1 (= (10 − 10%) − 10%);
4. Se o trabalho for entregue com 3 dias de atraso, os(as) integrantes receberão nota 7,3 (= ((10− 10%) − 10%) − 10%);

5. Após 3 dias de atraso, o trabalho não será avaliado, e a nota será zero.

APÊNDICE – Exemplos de Utilização do Framework

O objetivo deste trabalho prático é que os alunos desenvolvam alguns algoritmos fundamentais utilizados em computação gráfica. No caso deste trabalho, são os algoritmos de rasterização de pontos e linhas.

Embora existam atualmente diversas APIs que implementam estes algoritmos de forma bastante eficiente, dentre elas o OpenGL e o Direct3D, o propósito deste trabalho é que os alunos implementem estes algoritmos a partir do zero. Desta forma, os algoritmos desenvolvidos pelos alunos devem ser escritos em C, sem o uso de nenhuma biblioteca adicional. A escrita dos pixels deve ser feita diretamente na memória, byte a byte (um byte para cada componente de cor do pixel – RGBA).

Os sistemas operacionais modernos, tais como o Linux ou Windows, não permitem que o usuário tenha acesso direto à memória. O acesso à memória é normalmente feito por meio de uma API. Desta forma, para que o trabalho pudesse ser desenvolvido, o professor implementou um *framework* para a simulação de acesso direto à memória de vídeo. Embora este simulador seja implementado com o OpenGL, os alunos devem escrever seus algoritmos apenas em C, fazendo a escrita na memória utilizando apenas o ponteiro `FBptr`, que aponta para o primeiro byte da memória de vídeo simulada.

Como visto em aula, cada pixel possui 4 componentes de cor (RGBA), cada um ocupando um byte. Isto significa que cada componente pode assumir um valor, inteiro, no intervalo [0, 255]. A seção a seguir exemplifica como, através do uso do *framework*, se podem escrever pixels na tela.

Exemplo 1

O código abaixo escreve três pixels coloridos nas 12 primeiras posições da memória de vídeo (apontada por `FBptr`). Estes 3 pixels estão localizados no canto superior esquerdo da tela. Observe que este código utiliza somente C.

Para executar este exemplo, abra o arquivo `mygl.cpp` e altere a função `MyGlDraw()` da seguinte forma:

```
void MyGlDraw(void) {

 //
 // >>> Chame aqui as funções que você implementou <<<
 //

 // Escreve um pixel vermelho na posicao (0,0) da tela:
 FBptr[0] = 255; // componente R
 FBptr[1] = 0; // componente G
 FBptr[2] = 0; // componente B
 FBptr[3] = 255; // componente A

 // Escreve um pixel verde na posicao (1,0) da tela:
 FBptr[4] = 0; // componente R
 FBptr[5] = 255; // componente G
 FBptr[6] = 0; // componente B
```

```
 FBptr[7] = 255; // componente A

 // Escreve um pixel azul na posicao (2,0) da tela:
 FBptr[8] = 0; // componente R
 FBptr[9] = 0; // componente G
 FBptr[10] = 255; // componente B
 FBptr[11] = 255; // componente A
}
```

Exemplo 2

O código abaixo desenha uma linha lilás, e diagonal, com comprimento de 250 *pixels*.

```
void MyGlDraw(void) {

 //
 // >>> Chame aqui as funções que você implementou <<<
 //

 for (int i=0; i<250; ++i) {
 FBptr[4*i + 4*i*IMAGE_WIDTH + 0] = 255;
 FBptr[4*i + 4*i*IMAGE_WIDTH + 1] = 0;
 FBptr[4*i + 4*i*IMAGE_WIDTH + 2] = 255;
 FBptr[4*i + 4*i*IMAGE_WIDTH + 3] = 255;
 }
}
```

Exemplo 3

Esta atividade prática exige que os códigos que realizam a rasterização das primitivas sejam encapsulados em funções, e que estas sejam declaradas no arquivo `mygl.h` *e definidas no* arquivo `mygl.cpp`. A função `MyGlDraw()`, definida no arquivo `mygl.cpp`, deve ser utilizada apenas para chamar as funções desenvolvidas.

Sendo assim, o exemplo a seguir encapsula os códigos dos exemplos anteriores em funções, definidas nos arquivos `mygl.h` e `mygl.c`, e as chama de dentro da função `MyGlDraw()`, no arquivo `mygl.c`. Seguem abaixo as implementações.

Arquivo `mygl.h`:

```
#ifndef MYGL_H
#define MYGL_H

// Declaração da função que chamará as funções implementadas pelo aluno
void MyGlDraw(void);

//
// >>> Declare aqui as funções que você implementar <<<
//

void DesenhaPixels(void);

void DesenhaLinha(void);

#endif // MYGL_H
```

Arquivo `mygl.c`:

```
#include "mygl.h"

//
// >>> Defina aqui as funções que você implementar <<<  //

void DesenhaPixels(void) {
 // Escreve um pixel vermelho na posicao (0,0) da tela:

 FBptr[0] = 255; // componente R
 FBptr[1] = 0; // componente G
 FBptr[2] = 0; // componente B
 FBptr[3] = 255; // componente A

 // Escreve um pixel verde na posicao (1,0) da tela:

 FBptr[4] = 0; // componente R
 FBptr[5] = 255; // componente G
 FBptr[6] = 0; // componente B
 FBptr[7] = 255; // componente A

 // Escreve um pixel azul na posicao (2,0) da tela:

 FBptr[8] = 0; // componente R
 FBptr[9] = 0; // componente G
 FBptr[10] = 255; // componente B
 FBptr[11] = 255; // componente A
}

void DesenhaLinha(void) {
 for (unsigned int i=0; i<250; ++i) {
 FBptr[4*i + 4*i*IMAGE_WIDTH + 0] = 255;
 FBptr[4*i + 4*i*IMAGE_WIDTH + 1] = 0;
 FBptr[4*i + 4*i*IMAGE_WIDTH + 2] = 255;
 FBptr[4*i + 4*i*IMAGE_WIDTH + 3] = 255;
 }
}

// Definição da função que chamará as funções implementadas pelo aluno void
MyGlDraw(void) {

 //
 // >>> Chame aqui as funções que você implementou <<<  //

 DesenhaLinha();
 DesenhaPixels();
}
```