UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMÁTICA

Disciplina: [**GDSCO0051] Introdução à Computação Gráfica**
Professor: **Christian Azambuja Pagot (email: christian@ci.ufpb.br)**.
Data de entrega: **20/10/2020**.

Atividade Prática 2 – 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-se o acesso direto a 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://github.com/capagot/icg/tree/master/02_mygl_framework, disponibilizado pelo professor.

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

- `core.h`
- `frame_buffer.h`
- `main.c`
- `main.h`
- `Makefile`
- `mygl.c`
- `mygl.h`

O arquivo `core.h` contém macros que determinam as dimensões da janela em pixels. O arquivo `frame_buffer.h` declara o ponteiro `fb_ptr`, que aponta para o primeiro *byte* do *color buffer*. A posição apontada por `fb_ptr` 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.c` definem funções e variáveis necessárias a 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.c` 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 também os alunos deverão declarar as funções a serem desenvolvidas. O arquivo `mygl.c` 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.

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. As cores dos *pixels* ao longo da linha rasterizada devem ser obtidas por meio de *interpolação linear* das cores dos vértices. O algoritmo de rasterização de linha a ser implementado deve ser o **Algoritmo do Ponto Médio!**

- **`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. As cores dos *pixels* das arestas do triângulo devem ser obtidas através da *interpolação linear* das cores de seus 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.

Extras

As atividades descritas na Seção **Desenvolvimento** são o mínimo esperado nesta atividade. A implementação de recursos extras são livres e podem contribuir para uma pontuação extra no trabalho. Observa-se, entretanto, que o desenvolvimento de atividades extras deverá ser discutida previamente com o professor, e somente será considerada e avaliada caso as atividades descritas na Seção **Desenvolvimento** estejam completamente, e corretamente, implementadas. As atividades extras somarão, no máximo, **3 pontos** ao trabalho.

Entrega

Os trabalhos devem ser entregues até o dia **20/10/2020**, impreterivelmente até as **23 horas e 55 minutos**. O trabalho será considerado entregue assim que estiver acessível, de forma **pública**, em algum repositório do tipo **Git** (Github.com, Gitlab.com, etc.). **Este trabalho pode ser desenvolvido em duplas**.

O endereço do repositório deverá ser informado ao professor antes do *deadline* do exercício, através de tarefa específica no SIGAA. Os endereços dos repositórios serão divulgados a todos os outros alunos da disciplina por meio do SIGAA.

No repositório deverão constar os *printscreens* dos resultados (acompanhados de todos os parâmetros utilizados na sua geração) acompanhados de pequenos textos (frases ou parágrafos) explicativos. O repositório deve conter também o código fonte. Abaixo segue um detalhamento de cada um dos componentes que devem aparecer no *post*.

- ***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 melhoras.
  - 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!**

Abaixo segue um link de um repositório do Github que contém a postagem de um trabalho de computação gráfica. O trabalho não é sobre rasterização, mas serve para dar uma idéia de como se espera que seja o post deste trabalho: várias images de resultado, explicações sucintas e referências.

- https://github.com/Beskamir/Intro-to-Computer-Graphics

Avaliação

A avaliação dos trabalhos levará em consideração os seguintes critérios:

- Completude to 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:

Entregas atrasadas de trabalhos serão contabilziadas como faltas em aula.

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 8: 1) se o trabalho for entregue no prazo o aluno receberá nota 8; 2) se o trabalho for entregue com 1 dia de atraso, o aluno receberá nota 7, 2 (= 8 − 10%); se o trabalho for entregue com 2 dias de atraso, o aluno receberá nota 6,

4 (= (8 − 10%) − 10%); se o trabalho for entregue com 3 dias de atraso, o aluno receberá nota 5, 8 (= ((8 − 10%) − 10%) − 10%); e assim por diante.

Após 5 dias de atraso, o trabalho não será mais 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 implementem 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 a 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 `fb_ptr`, 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 `fb_ptr`). 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.c` 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:
    fb_ptr[0]  = 255; // componente R
    fb_ptr[1]  = 0;   // componente G
    fb_ptr[2]  = 0;   // componente B
    fb_ptr[3]  = 255; // componente A

    // Escreve um pixel verde na posicao (1,0) da tela:
    fb_ptr[4]  = 0;   // componente R
    fb_ptr[5]  = 255; // componente G
    fb_ptr[6]  = 0;   // componente B
    fb_ptr[7]  = 255; // componente A

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

```
    fb_ptr[8]  = 0;   // componente R
    fb_ptr[9]  = 0;   // componente G
    fb_ptr[10] = 255; // componente B
    fb_ptr[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) {
        fb_ptr[4*i + 4*i*IMAGE_WIDTH + 0]  = 255;
        fb_ptr[4*i + 4*i*IMAGE_WIDTH + 1]  = 0;
        fb_ptr[4*i + 4*i*IMAGE_WIDTH + 2]  = 255;
        fb_ptr[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.c`. A função `MyGlDraw()`, definida no arquivo `mygl.c`, 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

#include "core.h"
#include "frame_buffer.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:

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

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

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

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

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

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