UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMÁTICA

Disciplina: [**1107190] Introdução à Computação Gráfica - Turma 01**.
Professor: **Christian Azambuja Pagot (email: christian@ci.ufpb.br)**.
Data de início: **05/03/2018**. Data de entrega: **19/03/2018**.

Trabalho I

Objetivo

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

Atividade

Nesta primeira 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 através da escrita direta na memória de vídeo. Como os sistemas operacionais atuais protegem a memória quanto ao acesso direto, os alunos deverão utilizar um *framework* que simule o acesso à memória de vídeo.

O Framework

Estrutura

Este *framework* simula o acesso direto à memória. Todos os seus arquivos encontram-se comprimidos no arquivo **cg_framework.tar.bz2**.

Os arquivos que compõe o *framework* são:

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

O arquivo *definitions.h* contém a declaração das constantes que determinam a dimensão da tela (resolução) e o ponteiro (*FBptr*) para o início da memória de vídeo (mais especificamente, o início da região do *framebuffer*). A primeira posição de memória, apontada por *FBptr*, corresponde ao pixel da posição (0,0), canto superior esquerdo da tela. Cada pixel possui 4 componentes de cor (RGBA), cada uma representada por 1 byte (*unsigned char*).

O arquivo *main.cpp* contém o programa principal e a declaração da função *MyGlDraw()*, de onde as funções de rasterização a serem criadas devem ser chamadas.

O arquivo *main.h* contém a definição de algumas funções de auxílio na simulação do acesso à memória de vídeo e ao seu *scan*.

O arquivo *Makefile* que acompanha o *framework* é um script para compilação do sistema no ambiente Unix. A conversão do código para compilação em ambiente Windows é responsabilidade do aluno.

IMPORTANTE: Os únicos arquivos que deverão ser modificados para a execução do projeto são o *mygl.h*, onde serão definidas as funções de rasterização, e o arquivo *main.cpp*, onde estas funções deverão ser chamadas!

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, pelo menos, 3 funções. São elas:

- ***PutPixel***: Função que rasteriza um ponto na memória de vídeo, recebendo como parâmetros a posição do pixel na tela (x,y) e sua cor (RGBA).

- ***DrawLine***: Função que rasteriza uma linha na tela, recebendo como parâmetros os 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 através da *interpolação linear* das cores dos vértices. O algoritmo de rasterização 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 (xa,ya), (xb,yb) e (xc,yc) 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/C++. Não é permitido o uso de nenhuma biblioteca externa, a não ser: GLUT/GLEW/GLFW/SDL/glm e o OpenGL!

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 **19/03/2018**, impreterivelmente até as **23 horas e 55 minutos**. O trabalho será considerado entregue assim que estiver acessível no repositório *do Github* criado pelos alunos para a disciplina.

O endereço do repositório deverá ser informado ao professor antes do *deadline* do exercício, através de tarefa específica no SIGAA.

No repositório deverão constar os *printscreens* dos resultados (acompanhados de todos os parâmetros utilizados na sua geração) e textos explicativos. A postagem do código fonte é opcional. Abaixo segue um detalhamento de cada um dos componentes que devem aparecer no *post*.

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

- **Texto**:
  - Deve iniciar com um parágrafo que introduza a atividade a ser desenvolvida.
  - Deve explicar brevemente as estratégias adotadas pelo aluno na resolução da atividade (estruturas de dados utilizadas e funções desenvolvidas). O aluno pode ilustrar estas explanações através da apresentação de pequenos trechos de código/algoritmo.
  - Breve discussão sobre os resultados gerados e possíveis melhoras.
  - Citar eventuais dificuldades encontradas.
  - Listar as referência bibliográficas (livros, artigos, endereços web).
  - **OBS: A inclusão de trechos de código no corpo do blog deve se limitar ao mínimo indispensável para o correto entendimento do texto!**

- **DICA:** Abaixo segue um link de uma página que contém a postagem de um trabalho feito por um aluno de uma universidade do exterior. Como se pode observar, a página apresenta muitas imagens de **resultado** e textos **curtos** descrevendo como os resultados foram obtidos:

  - http://www.dgp.toronto.edu/~jacky/raytracer.html

Avaliação

A avaliação dos trabalhos levará em consideração os seguintes critérios:

- Aderência ao tema proposto.
- Clareza, organização, corretude, legibilidade e capacidade de síntese do texto.
- Relevância dos elementos apresentados (imagens, trechos de código, etc.) no contexto das discussões.
- Embasamento técnico dos argumentos.
- Eficiência e eficácia das soluções apresentadas.
- Capacidade de análise dos resultados e autocrítica.

Importante:

- Não serão aceitos trabalhos atrasados ou que apresentem indícios de plágio.
- Não serão aceitos trabalhos enviados por qualquer outro meio que não o repositório informado previamente pelos alunos.

APÊNDICE – Exemplos de Utilização do Framework

O objetivo do trabalho prático T1 é que os alunos desenvolvam alguns algoritmos fundamentais utilizados em computação gráfica. No caso deste trabalho, são os algoritmos de rasterização de primitivas.

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/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 de APIs.

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/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/C++.

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

```
void MyGlDraw(void)
{
	//*************************************************************************
	// Chame aqui as funções do mygl.h
	//*************************************************************************

	// 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, em diagonal, com comprimento de 250 *pixels*.

```
void MyGlDraw(void)
{
	//*************************************************************************
	// Chame aqui as funções do mygl.h
	//*************************************************************************

	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;
	}
}
```

Exemplo 3

Esta atividade prática exige que os códigos que realizam a rasterização das primitivas sejam encapsulados em funções, e que estas funções devem ser definidas obrigatoriamente no arquivo *mygl.h*. A função `MyGlDraw()`, definida no arquivo *main.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 no arquivo mygl.h, e chama estas funções de dentro da função `MyGlDraw()`, no arquivo *main.cpp*. No arquivo *mygl.h* as funções são definidas:

```
#ifndef _MYGL_H_
#define _MYGL_H_

#include "definitions.h"
#include "math.h"
#include <vector>

//*****************************************************************************
// Defina aqui as suas funções gráficas
//*****************************************************************************

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;
	}
}

#endif // _MYGL_H_
```

No arquivo *main.cpp* as funções definidas previamente são chamadas para execução:

```
void MyGlDraw(void)
{
	//*************************************************************************
	// Chame aqui as funções do mygl.h
	//*************************************************************************

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