INF01046 – Fundamentos de Processamento de Imagens
Semestre 2003.2 – Turma A

Prof. Manuel M. Oliveira

1º Trabalho de Implementação

Data: 06/10/03 Total de Pontos do Trabalho: 100
Data de Entrega: 20/10/03 às 8:30pm

OBJETIVO

O objetivo deste trabalho é familiarizar os estudantes com algumas operações importantes envolvendo imagens. Mais especificamente, ao completar este trabalho você terá aprendido a:

a) Ler e gravar arquivos de imagens;
b) Exibir o conteúdo de um arquivo de imagem;
c) Converter uma imagem colorida em uma imagem em tons de cinza;
d) Aplicar um esquema simples para quantização de imagens;
e) Usar a biblioteca GLUT para gerenciar os serviços de janela.

Parte I – Leitura e Gravação de Arquivos de Imagens (20 pontos)

1) Escreva um programa para ler um arquivo de imagem no formato JPEG e regrave-o com um outro nome. Esta tarefa simples tem o objetivo de familizarizá-lo com o uso de bibliotecas para leitura e gravação de arquivos. Para completar a Parte I deste trabalho, você precisará dos seguintes arquivos:

i) ***jpeg.lib***, uma biblioteca estática para leitura e gravação de arquivos JPEG;
ii) ***jconfig.h*** e ***jmorecfg.h***, arquivo de cabeçalho a serem utilizados com a bilbioteca;
iii) ***jpeg_api32.lib***, uma API para a biblioteca jpeg.lib que eu preparei para facilitar o trabalho de vocês;
iv) ***jpeg_api.h***, um arquivo de cabeçalho para a PAI.
v) ***jpeg_api.h***, a header file for the API.

Você deverá incluir os arquivos .lib na sua instalação e informar ao Visual C++ sobre eles na caixa de diálogo acessível via (*project->settings->link*).

Teste o seu programa com as imagens disponibilizadas para o trabalho. Após, verifique os tamanhos das imagens em cada par (original e arquivo gravado). Você percebe alguma diferença visual entre eles? Alguma diferença nos tamanhos dos arquivos? Caso haja diferença nos tamanhos de arquivos, faça uma pequena pesquisa na web sobre arquivos JPEG e tente explicar a diferença observada.

Parte II – Leitura, Exibição e Operações sobre Imagens (80 pontos)

1) Estenda o programa que você desenvolveu na Parte I para exibir as imagens lidas e utilize GLUT para fazer a gerência de janela para você. O seu programa deve exibir duas janelas separadas. Na janela da esquerda, mostre a imagem original; na janela da direita, mostre o resultado da operação realizada sobre a imagem.

Operações a serem implementadas:

a) (**25 pontos**) Espelhamento horizontal e vertical da imagem original (*neste caso, excepcionalmente, mostre o resultado na janela da esquerda*). Ao espelhar (verticalmente/ horizontalmente) a imagem um número par de vezes, você deverá obter novamente a imagem original. Procure implementar estas operações de modo eficiente (dica: considere o uso do comando *memcpy*) sempre que possível, ao invés de trocar um par de pixels por vez. Certifique-se de que a operação funciona para imagens tanto com número par como com número ímpar de linhas e colunas.

b) (**20 pontos**) Converta uma imagem colorida em tons de cinza (*luminância*). Uma imagem em tons de cinza pode ser obtida a partir de uma imagem colorida aplicando-se a seguinte fórmula para cada um dos pixels da imagem original:
$L = 0.299*R + 0.587*G + 0.114*B$, onde R, G e B são as componentes de cor do pixel original. Ao criar uma imagem a ser exibida em tons de cinza, faça $R_n = G_n = B_n = L$;

O seu programa deve permitir que a aplicação do cálculo de luminância um número arbitrário de vezes durante sua execução. Pergunta: o que acontecerá com uma imagem em tons de cinza ($R_n = G_n = B_n = L$) caso o cálculo de luminância seja aplicado repetidas vezes (*e.g.*, recursivamente)?

c) (**25 pontos**) Implemente um processo de quantização (de tons) sobre as imagens em tons de cinza. Note que, neste caso, como a dimensão do espaço de cor é 1, tal processo de quantização se torna bastante simples. Assim, o seu programa deve receber como entrada o número de tons a serem utilizados no processo de quantização.

d) (**10 points**) Salve a imagem resultante das operações realizadas em um arquivo JPEG. Provide an option to save the new image as a JPEG file.

2) Escreva um relatório com tamanho equivalente a uma ou duas páginas descrevendo de forma ilustrada a sua implementação. No relatório, para cada etapa do trabalho, indique se você a completou satisfatoriamente. Em caso de não tê-la completado, explique porque não conseguiu fazê-lo. Além disso, liste as dificuldades que você enfrentou e, em retrospecto, indique o que você faria diferente de modo a minimizar ou evitar as dificuldades experimentadas.

IMPORTANTE: O seu relatório ilustrado (i.e., contendo imagens mostrando os resultados obtidos) deverá ser disponibilizado até a data/hora de entrega do trabalho por meio de uma web-page a ser criada por cada estudante para a disciplina.

3) Para o seu programa, é desejável construir uma interface intuitiva utilizando o seu "toolkit" preferido *(e.*g., FLTK, GLUI, etc.), mas certifique-se de que a sua interface é intuitiva. Neste caso, coloque uma imagem da interface no seu relatório na web. Para este primeiro trabalho de implementação, o seu programa poderá utilizar uma interface via teclado. Os próximos trabalhos, entretanto, já deverão incorporar o uso de uma interface gráfica. Portanto, é uma boa idéia já começar a tentar utilizar uma desde já.

GLUI é uma interface gráfica extremamente fácil de utilizar e que se encontra disponível para download na página de Paul Rademacher (http://www.cs.unc.edu/~rademach). Fontes, exemplos e manual de utilização estão disponíveis on-line.

DISCAS PARA COMPLETAR O TRABALHO

Importante: Antes de começar a implementação, leia atentamente estas dicas e consulte algum livro sobre OpenGL (ou pesquise na web) sempre que você encontrar algum comando de OpenGL, GLUT ou GLU que não lhe seja familiar. Em particular, recomendo o uso do OpenGL Red Book.

1) Como Utilizar Multiplas Janelas com GLUT

Você pode criar tantas janelas quantas quiser utilizando o comando *glutCreateWindow*. Esta função retorna um identificador único (um número inteiro) para cada janela criada. No caso de várias janelas terem sido criadas, você deverá indicar explicitamente a janela que será utilizada utilizando o comando *glutSetWindow*. É uma boa prática salvar os identificadores retornados pelo comando *glutCreateWindow* em um array e utilizar constants (*defines*) ao se referir a cada janela, de modo a melhorar a legibilidade do código. Por exemplo, veja o fragmento de código a seguir que ilustra o uso deste procedimento.

```
#include <GL/glut.h>
#define SOURCE 0
#define TARGET 1
int win_id[2];
```

e o utilize com o seguinte modelo de função *main* (observe que eu estarei usando notação de C++):

```
//*****************************************************************************
//
// main function for controlling the implementation of your assignment
//
//*****************************************************************************
int main(int argc, char *argv[])
{
//
// read the image file
//
    <your command to read the source image goes here>
//
// Initialize two windows.
// Make sure you understand the meaning of the parameters used with each command. In particular,
// make sure you understand the meaning of the parameters used with glutInitDisplayMode and understand
// its relationship to the glutSwapBuffers command. You can find detailed explanations about them in the
// OpenGL red book
//
// First, initialize source window
//
```

```
	glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
	glutInitWindowPosition(0, 0);
	glutInitWindowSize(<source image width>, <source image height>);
	win_id[SOURCE] = glutCreateWindow("Original Image");
	glutDisplayFunc(<name of your function for displaying in the SOURCE window >);
	glutReshapeFunc(sourceReshape);
	glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
	glClearColor(0.0, 0.0, 0.0, 0);
//
// Now, initialize target window
//
	glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
	glutInitWindowPosition((<source image width>+50, 0);
	glutInitWindowSize(<source image width>, <source image height>);
	win_id[TARGET] = glutCreateWindow("New Image");
	glutDisplayFunc(<name of your function for displaying in the TARGET window >);
	glutReshapeFunc(targetReshape)
	glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
	glClearColor(0.0, 0.0, 0.0, 0);
//
// call your function for initializing your user interface
//
	<your function for initializing your user interface>;
//
// call glutMainLoop()
//
	glutMainLoop();
	return 0;
}
```

2) Demais funções que você precisará para implementar o trabalho

```
//*******************************************************
//
// Display image
//
//*******************************************************
void
Image::Display(void)
{
	glClear(GL_COLOR_BUFFER_BIT);
	glRasterPos2i(0, 0);
	glDrawPixels(<image width>, < image height>, GL_RGB, GL_UNSIGNED_BYTE,<pointer to the 1-D
array of pixels of the image>);
	glutSwapBuffers();
}

//*******************************************************
//
// Display function for SOURCE
//
//*******************************************************
void sourceDisplay(void)
{
	Source.Display();
}

//*****************************************************************************
//
// Reshape function for SOURCE
//
//*****************************************************************************
```

```
void sourceReshape(int w, int h)
{
    glViewport(0, 0, (GLsizei) w, (GLsizei) h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, w, 0.0, h);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

//*****************************************************
//
// Display function for TARGET
//
//*****************************************************
void targetDisplay(void)
{
    Target.Display();
}

//*****************************************************************************
//
// Reshape function for TARGET
//
//*****************************************************************************
void sourceReshape(int w, int h)
{
        glViewport(0, 0, (GLsizei) w, (GLsizei) h);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0.0, w, 0.0, h);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
}
```

3) Sincronização dos conteúdos mostrados em todas as janelas

Uma forma simples de garantir o sincronismo do que é exibido em todas as janelas é utilizar um laço que percorre todas as janelas de interesse e chamar a função *glutPostRedisplay()* para cada uma delas. O fragmento de código a seguir ilustra isso para
A simple way to guarantee synchronized renderings in all windows is to loop through all windows and call *glutPostRedisplay()* for each one of them. The next code fragment illustrates this for the case of changing rendering mode and for changing of field of view.

```
switch (Option) {
case LUMINANCE: {
    Source.ComputeLuminance(&Target);
    for (i=0; i<2; i++) {
       glutSetWindow(win_id[i]);              // seleciona a janela
       glutPostRedisplay();                   // força a atualização do conteúdo da janela
     }
  break;
  }
}
```

4) **Leitura do arquivo de entrada (imagem)**

This code is showed here to illustrate how to use the jpeg_api library.

```
// Examples of how to use the read and write functions in your program
//
#include "jpeg_api.h"
```

```
//
int
main(int argc, char *argv[])
{
	int w, // image widht (# of columns) in pixels
	h, // image height (# of rows) in pixels
	c; // # of color channels. (e.g., 1 if grayscale image,
	// 3 if RGB image,
	// 4 if RGBA image
	//
	unsigned char *pixels; // pointer to a linear array that will
	// // contain the pixel information for the
	// // image.
	// // If image RGB -> pixels = RGBRGB...RGB
	// // If image RGBA-> pixels = RGBARGBA...RGBA
	// // If image grayscale -> pixels = LL...L (luminance)
	//
	if (!ReadJPEG(argv[1], &pixels, &w, &h, &c)) // argv[1] contains the input file name
	{ // pixels: returns the pixel data
	   // ... manipulate the image data // w, h, c: retun width, height and
	   // // # of channels, respectivelly
	   // // Now, suppose you would like to write back
	   // // the image using a different file name
	   // // (say, "output_name.jpg")
	   //
	   WriteJPEG("output_name.jpg", pixels, w, h, c);
	   //
	}
	else
	   // < your error message: "Not a valid JPEG file" goes here>
	}
```

5) Dica Final

Não esqueça de chamar o comando *glutSwapBuffers* após ter executado o commando *glDrawPixels*, conforme mostrado no procedimento Display.

Boa Sorte!