Fundamentos de Computação Gráfica INF 2608 - Trabalho 1 (Imagem)

Rafael Diniz
Lab. Telemídia, PUC-Rio
rafaeldiniz@telemidia.puc-rio.br

22 de junho de 2013

Sumário

1 Descrição do trabalho

O trabalho tem como objetivo segmentar os jogadores de futebol sobre o campo durante uma partida.

Para esse propósito foi feita a aplicação de filtros para processamento digital de imagens de forma a possibilitar um algorítmo simples que segmenta os jogadores de futebol do campo de futebol.

O arquivo de entrada é uma imagem de uma partida de futebol com jogadores e o campo de futebol ao fundo. Nessa imagem é aplicada primeiramente uma mudança do espaço de cor, de RGB para HSV, de forma a permitir que todas as operações sejam feitas somente utilizando-se a componente de luminância da imagem (no caso do HSV, o canal V). Após essa transformação, são aplicadas iterações de um filtro gaussiano e iterações do filtro da mediana, ambos com uma janela de 3x3 pixels, número de iterações especificadas como parâmetro para o software e operando somente na componente de luminância da imagem.

Após essa etapa é aplicado um filtro de detecção de bordas, no caso o filtro de Sobel. É importante ressaltar que o objetivo da aplicação dos dois filtros de redução de ruído, filtro da mediana e gaussiano, é de suavizar a textura do campo de futebol de forma a deixá-lo uniforme, fator que é importante para a eficiência do filtro de Sobel.

A imagem de saída após a aplicação do filtro de Sobel é então utilizada como uma máscara para o algorítmo de detecção dos jogadores, que copia os valores dos pixels da imagem original para a imagem de saída do software caso o valor do pixel da máscara seja superior a um limiar estabelecido.

2 Imagens testadas

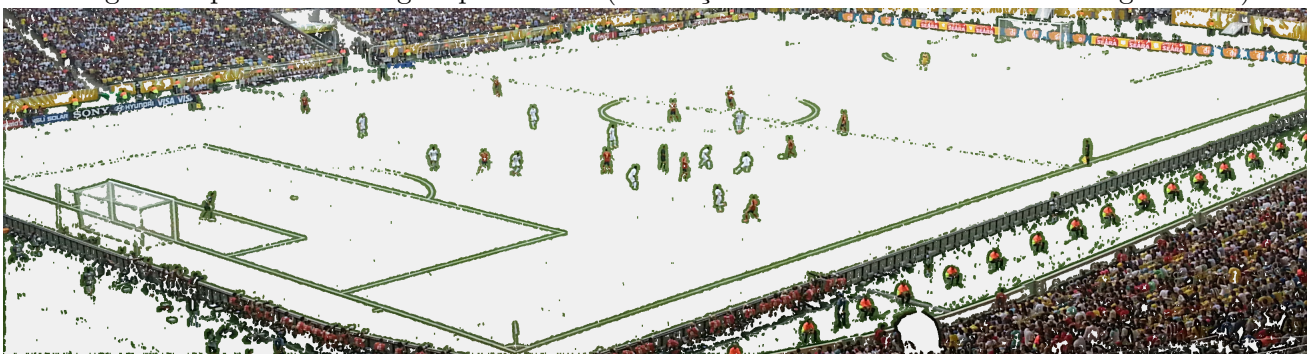
Segue a seguir uma lista de imagens, em sua versão original e após ser processada pelo software. As imagens estão em ordem crescente de resolução.

A figura 1 é uma imagem proveniente da transmissão digital da Globo em Standard Definition (480 linhas) do jogo Brasil e México da Copa das Confederações de 2013:

A figura 2 apresenta a imagem processada (1 iteração de filtro da mediana e 10 do gaussiano):

A figura 3 é uma imagem proveniente de câmera fotográfica digital em alta definição do jogo Espanha versus Tahiti que ocorreu no Maracanã:

A figura 4 apresenta a imagem processada (1 iteração de filtro da mediana e 20 do gaussiano):



A figura 5 é uma imagem de um jogo de futebol feminino retirada da internet em altíssima definição.

A figura 6 apresenta a imagem processada (1 iteração de filtro da mediana e 70 do gaussiano):

3 Resultados obtidos

Através da aplicação do filtro de Sobel aliado à aplicação de filtros de redução de ruído conseguiu-se um bom resultado na detecção dos jogadores.

As linhas do campo (linha do meio do campo, linha da área e da lateral), pelo fato de se apresentarem como um sinal de alta frequência ficaram presentes na imagem de saída. As linhas poderiam ser removidas utilizando-se um algorítmo que busca pelo elemento conexo das linhas do campo que apresentam um padrão bem característico.

Pelo fato do algorítmo de detecção de borda não pintar o interior às bordas dos jogadores, fica evidenciado pela figura 6, na qual o tamanho das jogadoras é grande, a parte do interior delas não fica exposto.

4 Análise dos dados

Pelo fato das imagens utilizadas não apresentarem uma quantidade relevante de ruído impulsivo, o filtro da mediana não apresentou grande importância na operação software.

O filtro gaussiano teve grande relevância na eficiência do filtro de Sobel no que diz respeito a detecção dos jogadores. A textura da grama do campo de futebol foi na prática removida como se fosse um ruído e transformada em um espaço de cor homogêneo, fato que permitiu ao filtro de Sobel identificar os jogadores com sucesso.

Abaixo vemos a importância da aplicação dos filtros e temos uma análise sobre o número de iterações do filtro que foram aplicadas à imagem. Na figura 7 é apresentada a imagem do jogo da Espanha e Tahiti processada pelo software, mas sem a aplicação de nenhum filtro de redução de ruído:

Na figura 8 abaixo são aplicadas 5 iterações do filtro da mediana, nenhuma iteração do filtro da gaussiano:

Na figura 9 são aplicadas 10 iteraçãos do filtro da mediana. Vemos que a melhoria com relação a 5 iterações não é grande:

Na figura 10 abaixo o filtro gaussiano é iterado 5 vezes (e o filtro da mediana não é aplicado). Vemos que a detecção dos jogadores é mais eficiente do que na figura 8, na qual somente o filtro da mediana é aplicado:

Na figura 11 o filtro gaussiano é iterado 10 vezes. Vemos claramente que ao aumentar o número de iterações estamos convergindo para o resultado esperado.

Pelo motivo da aplicação iterada do filtro da mediana não convergir para o resultado esperado, somente uma iteração do filtro da mediana foi aplicado em cada imagem. Este resultado pode ser considerado como esperado visto que uma vez que as componentes de alta frequência são removidas pelo filtro, não existe motivo de ficar aplicando-o mais vezes.

No caso do filtro gaussiano, podemos tirar duas conclusões à respeito de seu uso. O primeiro é que ao aplicarmos o filtro de forma iterativa, temos uma melhoria no resultado final. Outra conclusão é que quanto maior a resolução da imagem, maior o número de iterações necessárias para ser obtido um resultado bom. Esse fato é esperado visto que quanto maior a resolução da imagem, podemos considerar que a grama do campo é representada com uma maior fidelidade, que podemos interpretar como um ruído de maior intensidade, portanto, dado que a janela de operação do filtro é fixa em 3x3 px, um número maior de iterações do filtro são necessárias para se convergir em um resultado satisfatório.

5 Código Fonte

O código fonte do trabalho deverá estar anexo, sendo que a lista dos arquivos em código C (ISO C99) é:

- color.c: código C. Contém a implementação das funções de conversão de espaço de cor.
- color.h: código C. Cabeçalho do color.c.
- common.c: código C. Contém funções auxiliares, para depuração e alocação/desalocação de memória.
- common.h: código C. Cabeçalho do common.c.
- image.c: código C. Contém funções para leitura e escrita de imagens no formato PPM P6 (raw). Expõe todas as funções de apoio utilizadas no trabalho.
- image.h: código C. Cabeçalho do image.c.
- t1.c: código C. Arquivo mais importante que contém o código que implementa a proposta do trabalho. Nele estão implementados os filtros gaussiano, da mediana e Sobel e a lógica para identificar os jogadores. Contém a função main().
- Makefile: arquivo Makefile ajustado pra utilizar o compilador GCC. Deve ter suas variáveis CC, CFLAGS e LDFLAGS alteradas em caso de outro compilador.

Duas partes relevantes do código fonte a se ressaltar. Uma é a função main(), que embute toda a semântica do trabalho realizado:

```
int main(int argc, char *argv[])
{
    CImage  *img_rgb, *img_hsv;
    int median_pass = 1;
    int gauss_pass = 2;

    if (argc < 3)
    {
```

```
        fprintf(stderr, "usage: %s input_image.ppm output_image.ppm
            <median_iterations> <gaussian_iterations> \n", argv[0]);
        fprintf(stderr,"\n");
        exit(-1);
    }
    if (argc > 3)
        median_pass = atoi(argv[3]);
    if (argc > 4)
        gauss_pass = atoi(argv[4]);

    img_rgb   = ReadCImage(argv[1]);
    img_hsv   = CImageRGBtoHSV(img_rgb);

    CImage *gauss_img = img_hsv;
    for (int i = 0; i < gauss_pass; i++)
        gauss_img = gaussiano(gauss_img, 1);

    CImage *median_img = gauss_img;
    for (int i = 0; i < median_pass; i++)
        median_img = mediana(median_img, 1);

    CImage *sobel_img = sobel(median_img, 1);

    CImage *output_img = detect_players(sobel_img, img_rgb, 1);

    WriteCImage(output_img, argv[2]);
    DestroyCImage(&output_img);

    return 0;
}
```

E a função que detecta os jogadores. Nela a imagem de saída é primeiramente preenchida com os valores RGB 240, 240, 240, posteriormente, para cada valor de luminância da máscara (imagem após a aplicação do filtro de Sobel), em caso dele ser maior que 6 (limiar escolhido experimentalmente), o pixel correspondente da imagem original (antes dos processamentos) é copiado para a imagem de saída.

```
CImage *detect_players(CImage *mask, CImage *src, int destroy_src)
{
    int cols = src->C[2]->ncols;
    int lines = src->C[2]->nrows;

    CImage *out_img;

    fprintf(stderr, "Algoritmo que acha os jogadores em imagem %dx%d\n", cols, lines);

    out_img = CreateCImage(cols, lines);

    int n = cols * lines;

    for (int i = 0; i < n; i++)
    {
        out_img->C[0]->val[i] = 240;
        out_img->C[1]->val[i] = 240;
```

```
            out_img->C[2]->val[i] = 240;
        }


        for (int i = cols + 1; i < (n - cols - 1); i++)
        {
            if (mask->C[2]->val[i] > 6)
            {
                for (int j = 0; j < 3; j++)
                {
                    out_img->C[j]->val[i - cols - 1] = src->C[j]->val[i - cols - 1];
                    out_img->C[j]->val[i - cols] = src->C[j]->val[i - cols];
                    out_img->C[j]->val[i - cols + 1] = src->C[j]->val[i - cols + 1];
                    out_img->C[j]->val[i - 1] = src->C[j]->val[i - 1];
                    out_img->C[j]->val[i] = src->C[j]->val[i];
                    out_img->C[j]->val[i + 1] = src->C[j]->val[i + 1];
                    out_img->C[j]->val[i + cols - 1] = src->C[j]->val[i + cols - 1];
                    out_img->C[j]->val[i + cols] = src->C[j]->val[i + cols];
                    out_img->C[j]->val[i + cols + 1] = src->C[j]->val[i + cols + 1];
                }

            }

        }

        if (destroy_src)
        {
            DestroyCImage(&mask);
            DestroyCImage(&src);
        }
        return out_img;

}
```

6 Instruções de compilação e execução

```
Para compilar:
$ make

Para executar:
$ ./t1 entrada.ppm saida.ppm <numero_iteracoes_mediana> <numero_iteracoes_gaussiano>
```

Para converter imagens para o formato PPM foi utilizado o software *Gimp*. Deve ser selecionada a opção "Exportar", colocada a extensão .ppm no nome da imagem, e escolher a opção "Crú" quando aparecer a janela perguntando a respeito da Formatação dos dados a ser utilizada.