Aplicação de um Filtro Mediano em C++ para Remoção de Ruídos em Vídeos Digitais

Aluno, Fernando Felix do Nascimento Junior[1]; Professor, Danilo Regis[1]

[1] Instituto Federal de Educação, Ciência e Tecnologia da Paraiba – Campina Grande – Brasil

Trabalho de aplicação de filtro para remoção de ruídos em vídeos digitais corrompidos para ser apresentada as disciplinas de processamento digital de sinais e comunicações digitais no IFPB - Campus Campina Grande, como parte dos requisitos exigidos para aprovação das mesmas. Primeiro período de 2011.

SUMÁRIO

1. Introdução

Scratches e outros tipos de ruídos impulsivos geralmente degradam a qualidade do vídeo. Um exemplo comum esta relacionado a um receptor de satélite que produz *scratches* no sinal do vídeo quando a antena não esta exatamente direcionada para o satélite. Esses *scratches* constituem um distorção que é muito desengradável para o espectador. Portanto, existe uma demanda de filtros para remoção de ruido de vídeo e que possam preservar bordas, texturas e detalhes pequenos nas imagens processadas (ABBAS & DOMANSKI).

Os filtros medianos e suas variantes podem ser considerados os tipos de filtros mais populares para a questão do ruído em imagens e vídeos. Contudo, na maioria dos casos, filtros medianos e suas variantes tendem a remover detalhes bons e destruir texturas boas. Para solucionar as limitações dos filtros medianos, sistemas com processamento de predição de erro para processamento de imagem e vídeo foram elaborados (ABBAS & DOMANSKI).

Nesse trabalho, é apresentado uma aplicação genérica, em C++, de um filtro mediano para remoção de ruido em vídeos digitais com cores, onde o mesmo proporciona flexibilidade para implementações futuras com algum sistema de predição de erro ou outro tipo de filtro.

Na segunda seção é apresentado o conceito de filtro mediano. Na terceira seção é apresentado o algoritmo de pesquisa do valor mediano em um *array* utilizado na aplicação. Na quarta seção é apresentado o algoritmo de filtro mediano genérico. Na quinta seção é apresentado a aplicação que foi implementada para processar sinais digitais de vídeos, com foco no filtro mediano de ruídos. Na sexta seção é apresentado o resultado da aplicação. Na sétima seção é apresentada a conclusão.

2. Filtro Mediano

O filtro mediano é um filtro local não linear. Em processamento de imagem, um filtro mediano é computado através de uma convolução com um kernel (2N+1,2N+1). Para cada pixel em um *frame* de entrada, o pixel na mesma posição no *frame* de saída é substituída pela mediana dos valores de pixeis no *kernel (window)*. Desde que o valor mediano seja robusto para *outliers*, o filtro é geralmente utilizado para redução de ruídos impulsivos (LUKIN).

Devillard (1998) justifica o uso da mediana para rejeitar outliers, afirmando:

> "*When you have a signal distribution that is smooth enough but contains crazy outliers, you might get into trouble with fitting routines or statistical tools. That is the case for astronomical detectors hit by a cosmic ray for example, but there are many cases where you want to get an estimate of the average signal value without bothering about outlier rejection. A median can often be a fast and useful answer* " (DEVILLARD, 1998) .

As definições para a mediana de N valores numéricos podem ser dadas por:

– A mediana de uma lista de N valores é encontrada pela ordenação de um array

de entrada em ordem crescente, e pegando o valor do meio;

– A mediana de uma lista de N valores tem a propriedade de que na lista existe tanto o valor maior quanto valor menor do que este elemento .

Exemplo:

Valores de entrada: 19 10 84 11 23

Mediana: 19

A primeira definição é facilmente de se compreender, mas é dada através de um algoritmo (ordena o *array*, pega o valor central), na qual não é o melhor caminho para definição de um conceito. A ultima definição é mais interessante desde que a mesma deixe de fora a escolha do algoritmo. Intuitivamente, pode-se notar que a ordenação de todo o *array* para achar a mediana é muito trabalhosa, tendo em vista que se necessita apenas o valor central e não todo o *array* ordenado DEVILLA.

Aplicações para pesquisar o valor mediano em um *array* são mutias. Em processamento de sinal digital, isto pode ser feito removendo *outliers* em uma distribuição harmoniosa de valores. Ele pode também ser utilizado para estimar a média de uma lista de valores numéricos, independentemente de *strong outliers* DEVILLA.

3. Algoritmo de pesquisa de Mediana

Para fornecer um melhor performance para encontrar o valor mediano em um *array*, o algoritmo *Quick Select* foi utilizado na aplicação.

Este algoritmo foi publicado em (DEVILLA, 1998, apud Numerical Recipes in C, 1992). Velozmente sábio, ele tem um laco estreito com o método de Wirth. Entretanto, esse primeiro é mais rápido. Abaixo encontra-se o algoritmo implementado em ANSI-C:

```
/*
 * This Quickselect routine is based on the algorithm described in
 * "Numerical recipes in C", Second Edition,
 * Cambridge University Press, 1992, Section 8.5, ISBN 0-521-43108-5
 * This code by Nicolas Devillard - 1998. Public domain.
 */
#define ELEM_SWAP(a,b) { register elem_type t=(a);(a)=(b);(b)=t; }
elem_type quick_select(elem_type arr[], int n) {
        int low, high;
        int median;
        int middle, ll, hh;
        low = 0;
        high = n - 1;
        median = (low + high) / 2;
        for (;;) {
                if (high <= low) /* One element only */
                        return arr[median];
                if (high == low + 1) { /* Two elements only */
                        if (arr[low] > arr[high])
                                ELEM_SWAP(arr[low], arr[high]);
                        return arr[median];
                }
                /* Find median of low, middle and high items; swap into position low */
                middle = (low + high) / 2;
                if (arr[middle] > arr[high])
```

```
                ELEM_SWAP(arr[middle], arr[high]);
        if (arr[low] > arr[high])
                ELEM_SWAP(arr[low], arr[high]);
        if (arr[middle] > arr[low])
                ELEM_SWAP(arr[middle], arr[low]);
        /* Swap low item (now in position middle) into position (low+1) */
        ELEM_SWAP(arr[middle], arr[low+1]);
        /* Nibble from each end towards middle, swapping items when stuck */
        ll = low + 1;
        hh = high;
        for (;;) {
                do
                        ll++;
                while (arr[low] > arr[ll]);
                do
                        hh--;
                while (arr[hh] > arr[low]);
                if (hh < ll)
                        break;
                ELEM_SWAP(arr[ll], arr[hh]);
        }
        /* Swap middle item (in position low) back into correct position */
        ELEM_SWAP(arr[low], arr[hh]);
        /* Re-set active partition */
        if (hh <= median)
                low = ll;
        if (hh >= median)
                high = hh - 1;
    }
}
#undef ELEM_SWAP
```

4. Algoritmo de filtro mediano genérico

Considerando que V denota um vídeo corrompido por ruido, para cada pixel do *frame* V(X(l,w,h)), o seguinte algoritmo de filtro mediano genérico é aplicado:

Let X (l, w, h) = corrupted frame

Let Y (l, w, h) = restored frame

Let V (X) = corrupted video

Let W (Y) = restored video

Let size = filter window size

for f = 1 V number of Frames in a video

read X(l,w,h) from v(f)

for l = 1 X number of Layers in a frame

for w = 1 X number of Rows in a frame

for h = 1 X number of Columns in a frame

if (1 > w < number of Rows) and (1 > h < number of Columns)

Let P(size) = pixel elements in the filter window, x(l,w,h) based;

medianPixel = Median(P);

```
            y(l,w,h) = medianPixel;
          end if
          if (w = 1 or w = width) or (h = 1 or h = height)
            y(l,w,h) = x(l,w,h)
          end if
        end for
      end for
    end for
    write Y(l,w,h) in v(f)
end for
```

5. Aplicação em C++

A aplicação foi feita utilizando, de modo a ser flexível, o paradigma de orientação a objeto na linguagem de programação C++, utilizando o compilador g++. No entanto, para melhorar a performance de memoria, foi utilizado o conceito antigo de ponteiros da linguagem C para alocação dinâmica na memoria dos *frames* dos vídeos de entrada e saída. A figura 1 ilustra o diagrama de classes da aplicação como um todo. A seguir estão as etapas que foram realizadas para a construção da aplicação genérica.

- Primeiro, foram criados novos tipos (ou classes) para melhorar o entendimento do código: *Pixel, DefaultFrame e LayeredFrame*. O tipo *Pixel* representa um *unsigned char*; O tipo *DefaultFrame* representa um *frame* padrão, na qual, é um conjunto de pixeis; O *LayeredFrame* representa um *DefaultFrame* que possui mais de uma camada de cores, como no caso de *frames* de um vídeo em formato YUV;
- Segundo, foi criado uma classe *Video*, na qual, contem métodos e atributos para manipulação de *frames* de vídeos digitais;
- Terceiro, foi criado uma classe estendida de *Video*, a classe *MedianFilter*, para manipular um vídeo de entrada corrompido de modo que o mesmo seja filtrado para gerar um vídeo de saída regenerado, utilizando o algoritmo de pesquisa mediano *Quick Sort* e o algoritmo de filtro mediano genérico, através de uma janela de filtro 3x3;
- Quarto, por questão de organização, as operações matemáticas necessárias para manipulação de vídeos digitais, foi criado a classe *Math*;
- Quinto, foi gerado uma documentação em formato HTML utilizando o JavaDoc, através de uma adaptação do código da aplicação em C++ para Java.

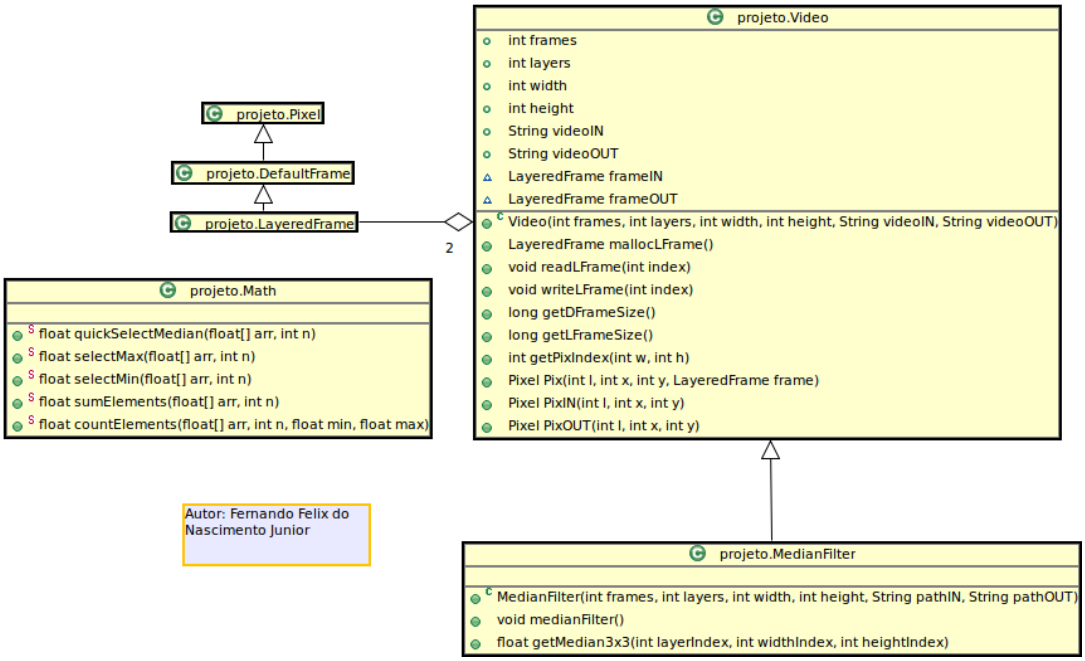

Figura 1: Diagrama de Classes, UML

6. Resultado

O resultado da aplicação do filtro mediano para remoção de ruídos em vídeos digitais é ilustrado nas figuras 2 e 3, onde (2) representa o vídeo corrompido com ruido *Salt & Papper* e (3) o vídeo regenerado pelo filtro.

Figura 2: Video corrompido

Figura 3: Video filtrado com o algoritmo mediano genérico

As especificações para a filtragem do vídeo corrompido da figura 2 foram:

- Formato do vídeo: YUV;
- Numero de camadas de cores: 3;
- Largura: 352/2;
- Altura: 288/2;
- Numero de frames: 300.

A duração média para a filtragem, para 10 testes com o mesmo vídeo ilustrado

pela figura 2, foi de aproximadamente 7 segundos.

7. Conclusão

Portanto, pode-se concluir que, mesmo sendo genérico, o algoritmo de filtro não linear mediano implementado reduz consideravelmente os ruídos em um vídeo corrompido, como foi ilustrado na figura 3.

Referências

ABBAS, K, J. DOMANSKI, M. Rejection of scratches from video by use of filters with predction error processing. Proceedings of Signal Processing X: Theories and Applications, Eusipco 2000, Tampere 2000.

LUKIN, A. Tips & Tricks: Fast Image Filtering Algorithms. GraphiCon'2007. Russia, Moscow, Junho, 2007.

DEVILLA, N. Fast median search: an ANSI C implementation. Julho, 1998.