1. RESUMO

A operação denominada esqueletização, ou afinamento, remove todos os pixels redundantes em uma imagem produzindo uma nova imagem simplificada com largura de um único pixel. O problema para os algoritmos de afinamento é o de determinar quais pixels são redundantes em uma imagem. Por outro lado, o processo de afinamento é muito melhor que o processo de erosão, pois os pixels que devem ser removidos são primeiramente marcados e posteriormente removidos em um segundo. Este processo é repetido até que não existam mais pixels redundantes, até o ponto que os pixels remanescentes são aqueles que pertencem ao esqueleto do objeto. Este método é chamado de afinamento é caracterizado por sucessivas deleções e é um método bastante utilizado na prática. O esqueleto do objeto precisa permanecer intacto e deve respeitar as seguintes propriedades:

As regiões afinadas precisam ter um pixel de largura;

Os pixels que formam o esqueleto precisam permanecer próximos do centro da região de cruzamento de regiões.

É necessário que os pixels do esqueleto formem o mesmo número de regiões que a imagem original apresentava.

Vários métodos já foram implementados e testados antes do método *Zhang-Suen* (1984) o qual será abordado neste documento.

A idéia básica do método *Zhang-Suen* é decidir se um determinado pixel será eliminado olhando somente seus oito vizinhos. Existem duas regras para decidir se o pixel deve ou não ser removido.

A primeira regra diz que o pixel somente pode ser apagado o número de conectividade do mesmo for igual a um. Isto significa que o pixel é conectado somente a uma única região. Se um pixel possuir o número de conectividade igual a dois então, duas regiões conectadas poderão se separar e isto viola a terceira propriedade de esqueletização.

A segunda regra é que o pixel somente pode ser apagado se este tiver mais de um e menos de sete vizinhos. Esta regra assegura que os pixels resultantes foram retirados sucessivamente das bordas da região da imagem, e não de suas partes internas.

Para afinar uma região estas regras devem ser aplicadas para todos os pixels que pertencem a região, e os pixels que satisfazem as condições acima podem ser removidos.

2. INTRODUÇÃO

Para realizar a esqueletização de *Zhang-Suen* toma-se por base sempre a comparação do pixel que se está tentando eliminar sobre seus oito vizinhos. Existem quatro regras que devem ser aplicadas se e somente se as quatro forem satisfeitas, o pixel poderá ser eliminado. Tais regras asseguram que se o pixel em questão for eliminado não fará com que diferentes regiões ligadas por ele passem a ficar separadas. Também assegura que a eliminação de pixels sempre ocorrerá nas bordas do objeto.

3. ALGORITMO DE AFINAMENTO DE ZHANG-SUEN

3.1 DEFINIÇÃO TEÓRICA

O algoritmo é dividido em duas sub-iterações. Em uma sub-iteração, o pixel I(i,j) é eliminado (ou marcado para deleção) dependendo do valor verdade das seguintes condições:

1. O número de conectividade é 1;

P9	***P2***	***P3***
P8	***P1***	***P4***
P7	***P6***	***P5***

O número de conectividade é definido como sendo o número de transições de ***branco para preto***, nos pixels que circundam o pixel central (iniciando em *P2*, e terminando em P9) que deve ser ***exclusiva*** uma.

2. Existem ao menos dois pixels vizinhos pretos, e não mais do que seis;

P9	*P2*	*P3*
P8	***P1***	*P4*
P7	*P6*	*P5*

O número de vizinhos refere-se também aos pixels na faixa *P2*,...*P9* que não são fundo.

3. Ao menos um dos I(i,j+1), I(i-1,j) e I(i, j-1) são fundo da imagem (branco);

P9	***P2***	*P3*
P8	***P1***	***P4***
P7	*P6*	*P5*

A condição se refere aos pixels *P2*, *P4* e *P8*.

4. Ao menos um dos I(i-1,j), I(i+1,j) e I(i, j-1) são fundo da imagem (branco);

P9	***P2***	*P3*
P8	***P1***	*P4*
P7	***P6***	*P5*

A condição se refere aos pixels *P2*, *P6* e *P8*.

No final desta sub-iteração os pixels marcados são eliminados. A próxima sub-iteração é a mesma exceto para os passos 3 e 4, logo:

1. O número de conectividade é 1;

P9	***P2***	***P3***
P8	***P1***	***P4***
P7	***P6***	***P5***

O número de conectividade é definido como sendo o número de transições de ***branco para preto***, nos pixels que circundam o pixel central (iniciando em *P2*, e terminando em P9) que deve ser ***exclusivamente*** uma.

2. Existem ao menos dois pixels vizinhos pretos, e não mais do que seis;

P9	*P2*	*P3*
P8	***P1***	*P4*
P7	*P6*	*P5*

O número de vizinhos refere-se também aos pixels na faixa *P2*,...*P9* que não são fundo.

3. Ao menos um dos I(i-1,j), I(i,j+1) e I(i+1, j) são fundo da imagem (branco);

P9	***P2***	*P3*
P8	***P1***	***P4***
P7	***P6***	*P5*

A condição se refere aos pixels *P2*, *P4* e *P6*.

4. Ao menos um dos I(i,j+1), I(i+1,j) e I(i, j-1) são fundo da imagem (branco);

P9	*P2*	*P3*
P8	***P1***	***P4***
P7	***P6***	*P5*

A condição se refere aos pixels *P4*, *P6* e *P8*.

E novamente, cada um dos pixels marcados são deletados.

Observação importante:

É importante ressaltar que em ambas as iterações, os pixels só devem ser eliminados ***no final*** da iteração. Se no final da segunda iteração não existem pixels para serem eliminados, então a esqueletização está completa e o programa para.

As figuras abaixo, mostram os esqueletos encontrados pelo algoritmo *Zhang-Suen* aplicado às quatro imagens vistas a seguir: **T**, **X**, **V** e **8**. O esqueleto **T** (fig. 1) é excepcionalmente bom, e o esqueleto **V** não mostra sinais de cauda (houve perda do vértice implícito em **V** (fig. 2), agora substituído por uma espécie de curva). O esqueleto **X** (fig. 3) mostra um alongamento no vértice principal, e o esqueleto **8** apresenta uma linha nebulosa (linha distoante com relação a imagem). Pode-se eliminar estes problemas com a aplicação de um pré-processamento (pré-processamento sugerido por Stentiford).

Fig. 1 – Afinamento de **T** – resultado satisfatório.

Fig 2. Afinamento de **V** – perda de vértice.

*

Fig 3. Afinamento de **X** – alongamento do vértice principal.

Fig 4. Afinamento de **8** – presença de linha destoante (*).

4. EXEMPLO DA APLICAÇÃO DO MÉTODO

Por exemplo, supondo que tenhamos a seguinte imagem de origem:

Fig. 5 – Imagem do número 1.

Fazendo um *zoom* da região em vermelho, obteremos o seguinte grid de pixels:

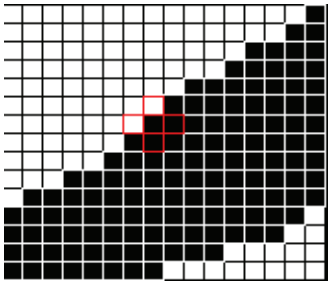

Fig. 6 – Grid de pixels de uma região da imagem

Observe que se for aplicado o método de afinamento de *Zhang-Suen* na região em vermelho, teremos o seguinte grid de pixels:

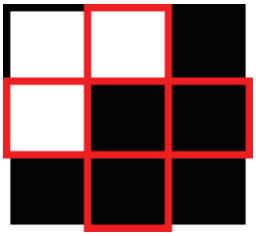

Fig. 7 – Grid de pixels da região para afinamento

Neste caso, verificando as quatro regras de aplicação do algoritmo no primeiro passo temos:

- 1ª Regra **é válida** pois a conectividade do pixel é igual a 1, pois temos somente uma transição de branco para preto;

- 2ª Regrá **é válida** pois o pixel central possui seis (6) vizinhos e isto é válido para a regra que cita que o pixel deve ter mais de um (1) e menos de sete (7) vizinhos;
- 3º Regra **é válida** pois, I(i,j+1) é preto I(i-1,j) é branco I(i,j-1) é branco e a regra diz que ao devemos ter ao menos um branco nestas transições, como temos dois logo é válida;
- 4º Regra **é válida** pois, I(i-1,j) é branco I(i+1,j) é preto I(i,j-1) é branco e a regra diz que ao devemos ter ao menos um branco nestas transições, como temos dois logo é válida;

Como as quatro regras são verificadas, o pixel central é apagado.

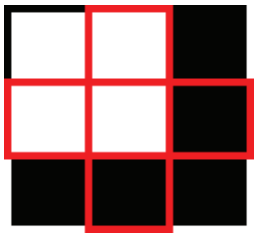

Fig. 8 – Grid de pixels já aplicado o primeiro passo do afinamento

5. RESULTADO DA APLICAÇÃO DO ALGORITMO

Foi executado o algoritmo de *Zhang-Suen* na Imagem manuscrita do número **1** (Figura 5 - pág. 6), e o resultado obtido foi a seguinte imagem:

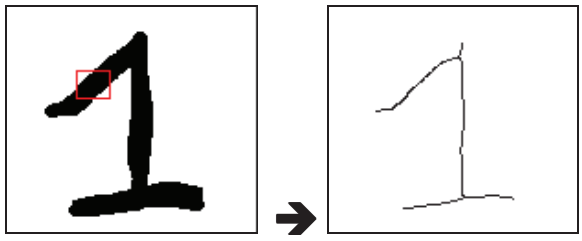

Fig. 9 – Imagem resultante do afinamento (Fig. 5) utilizando o algoritmo Zhang-Suen

Referência:

[ZHAN-84] Zhang S. and Fu K.S, “A Thinning Algorithm for discrete Binary Images”, in “Algorithms for Image Processing and Computer Vision”, JR Parker- John Wiley &Sons, Inc , 1997.

6. CÓDIGO FONTE

O código correspondente à implementação do algoritmo está descrito a seguir:

```
typedef struct tagPoint{
      int Px,Py;
}PixelPoint;


BOOL CPdiBase::ThiningZhangSuen(void)
{

      #define P1(MAT,X,Y) (MAT.GetAt(X))->GetAt(Y)
      #define P2(MAT,X,Y) (MAT.GetAt(X-1))->GetAt(Y)
      #define P3(MAT,X,Y) (MAT.GetAt(X-1))->GetAt(Y+1)
      #define P4(MAT,X,Y) (MAT.GetAt(X))->GetAt(Y+1)
      #define P5(MAT,X,Y) (MAT.GetAt(X+1))->GetAt(Y+1)
      #define P6(MAT,X,Y) (MAT.GetAt(X+1))->GetAt(Y)
      #define P7(MAT,X,Y) (MAT.GetAt(X+1))->GetAt(Y-1)
      #define P8(MAT,X,Y) (MAT.GetAt(X))->GetAt(Y-1)
      #define P9(MAT,X,Y) (MAT.GetAt(X-1))->GetAt(Y-1)
      #define Delete(MAT,X,Y) (MAT.GetAt(X))->SetAt(Y,0)

      //     P9      P2     P3
      //     P8      P1     P4
      //     P7      P6  P5

      BOOL ThiningContinue=TRUE;
      int line, col, counter;
      PixelPoint ActualPixel;
      BYTE Conectivity=0, Neighboors=0;
      CArray <PixelPoint,PixelPoint> RemPoints;
      CArray<CArray<BYTE, BYTE> *,CArray<BYTE, BYTE> *> Iteraction;

      for (line = 0; line < (int) GetHeight(); line++)
      {
            CArray<BYTE, BYTE> *ptrLine = new CArray<BYTE, BYTE>;

            for (col = 0; col < (int) GetWidth(); col++)
            {
                  (int)  GetPixelBW(col,line)      ?    ptrLine->Add(0)  :
ptrLine->Add(1);
            }
            Iteraction.Add(ptrLine);
      }

      while(ThiningContinue)
      {
            ThiningContinue = FALSE;
```

```
                        // First Sub-Iteraction
                        for (line = 1; line < (int) GetHeight()-1; line++)
                        {
                                for (col = 1; col < (int) GetWidth()-1; col++)
                                {
                                        Neighboors = 0;
                                        Conectivity= 0;

                                        // Pixel must be black
                                        if( P1(Iteraction,line,col) == 0 )
                                                continue;

                                        // Connectivity number must be 1;
                                        Conectivity = (P2(Iteraction,line,col) ==
0 && P3(Iteraction,line,col)== 1) ? 1 : 0;
                                        Conectivity  +=  (P3(Iteraction,line,col)
== 0 && P4(Iteraction,line,col)== 1) ? 1 : 0;
                                        Conectivity  +=  (P4(Iteraction,line,col)
== 0 && P5(Iteraction,line,col)== 1) ? 1 : 0;
                                        Conectivity  +=  (P5(Iteraction,line,col)
== 0 && P6(Iteraction,line,col)== 1) ? 1 : 0;
                                        Conectivity  +=  (P6(Iteraction,line,col)
== 0 && P7(Iteraction,line,col)== 1) ? 1 : 0;
                                        Conectivity  +=  (P7(Iteraction,line,col)
== 0 && P8(Iteraction,line,col)== 1) ? 1 : 0;
                                        Conectivity  +=  (P8(Iteraction,line,col)
== 0 && P9(Iteraction,line,col)== 1) ? 1 : 0;
                                        Conectivity  +=  (P9(Iteraction,line,col)
== 0 && P2(Iteraction,line,col)== 1) ? 1 : 0;

                                        if (Conectivity != 1)
                                                continue;

                                        // 2 <= BlackNeighboors <= 6
                                        Neighboors  =  P2(Iteraction,line,col)  +
P3(Iteraction,line,col) + P4(Iteraction,line,col) + P5(Iteraction,line,col) +
                                                P6(Iteraction,line,col)          +
P7(Iteraction,line,col) + P8(Iteraction,line,col) + P9(Iteraction,line,col);


                                        if (Neighboors < 2 || Neighboors > 6)
                                                continue;

                                        // At least one of P2, P4 and P8 are
background

                                        Neighboors = 0;
                                        Neighboors  =   P2(Iteraction,line,col)  *
P4(Iteraction,line,col) * P8(Iteraction,line,col);

                                        if (Neighboors != 0)
                                                continue;
```

```
                                        // At least one of P2, P6 and P8 are
background
                                        Neighboors = 0;
                                        Neighboors =   P2(Iteraction,line,col) *
P6(Iteraction,line,col) * P8(Iteraction,line,col);

                                        if (Neighboors != 0)
                                                continue;

                                        // Actual Pixel was deleted
                                        ThiningContinue = TRUE;
                                        ActualPixel.Px = line;
                                        ActualPixel.Py = col;
                                        RemPoints.Add(ActualPixel);
                                }
                        }

                        for(counter=0;counter<RemPoints.GetSize();counter++)

        Delete(Iteraction,RemPoints[counter].Px,RemPoints[counter].Py);

                        RemPoints.RemoveAll();


                        // Second Sub-Iteraction

                        for (line = 1; line < (int) GetHeight()-1; line++)
                        {
                                for (col = 1; col < (int) GetWidth()-1; col++)
                                {
                                        Neighboors = 0;
                                        Conectivity= 0;

                                        // Pixel must be black
                                        if( P1(Iteraction,line,col) == 0 )
                                                continue;

                                        // Connectivity number must be 1;
                                        Conectivity    =  (P2(Iteraction,line,col)
== 0 && P3(Iteraction,line,col)== 1) ? 1 : 0;
                                        Conectivity  +=  (P3(Iteraction,line,col)
== 0 && P4(Iteraction,line,col)== 1) ? 1 : 0;
                                        Conectivity  +=  (P4(Iteraction,line,col)
== 0 && P5(Iteraction,line,col)== 1) ? 1 : 0;
                                        Conectivity  +=  (P5(Iteraction,line,col)
== 0 && P6(Iteraction,line,col)== 1) ? 1 : 0;
                                        Conectivity  +=  (P6(Iteraction,line,col)
== 0 && P7(Iteraction,line,col)== 1) ? 1 : 0;
                                        Conectivity  +=  (P7(Iteraction,line,col)
== 0 && P8(Iteraction,line,col)== 1) ? 1 : 0;
```

```
                                        Conectivity  +=  (P8(Iteraction,line,col)
== 0 && P9(Iteraction,line,col)== 1) ? 1 : 0;
                                        Conectivity  +=  (P9(Iteraction,line,col)
== 0 && P2(Iteraction,line,col)== 1) ? 1 : 0;

                                        if (Conectivity != 1)
                                                continue;

                                        // 2 <= BlackNeighboors <= 6

                                        Neighboors  =  P2(Iteraction,line,col)  +
P3(Iteraction,line,col) + P4(Iteraction,line,col) + P5(Iteraction,line,col) +
                                                P6(Iteraction,line,col)          +
P7(Iteraction,line,col) + P8(Iteraction,line,col) + P9(Iteraction,line,col);

                                        if (Neighboors < 2 || Neighboors > 6)
                                                continue;

                                        // At least one of P2, P4 and P6 are
background
                                        Neighboors = 0;
                                        Neighboors  =   P2(Iteraction,line,col)  *
P4(Iteraction,line,col) * P6(Iteraction,line,col);

                                        if (Neighboors != 0)
                                                continue;

                                        // At least one of P2, P6 and P8 are
background
                                        Neighboors = 0;
                                        Neighboors  =   P4(Iteraction,line,col)  *
P6(Iteraction,line,col) * P8(Iteraction,line,col);

                                        if (Neighboors != 0)
                                                continue;

                                        // Actual Pixel was deleted
                                        ThiningContinue = TRUE;
                                        ActualPixel.Px = line;
                                        ActualPixel.Py = col;
                                        RemPoints.Add(ActualPixel);
                                }
                        }


                        for(counter=0;counter<RemPoints.GetSize();counter++)

        Delete(Iteraction,RemPoints[counter].Px,RemPoints[counter].Py);

                        RemPoints.RemoveAll();

                }
```

```
		for (line = 0; line < (int) GetHeight(); line++)
		{
			for (col = 0; col < (int) GetWidth(); col++)
			{
				if(P1(Iteraction,line,col) == 0)
					SetPixel((DWORD)		col,		(DWORD)
line,RGB(255,255,255));
				else
					SetPixel((DWORD) col, (DWORD) line,RGB(0,0,0));
			}
		}

		return TRUE;
	}
```

7. CONCLUSÃO

O método de *Zhang-Suen* torna-se uma ferramenta necessária na maioria dos projetos de processamento de imagens. Este método tem sido utilizado como base de comparação entre métodos de afinamento por muitos anos, sendo de fácil entendimento e de simples implementação. Este método possui a funcionalidade de rodar suas fases em paralelo, uma vez que a partir da primeira sub-iteração todas as outras são aplicadas sobre o seu resultado, portanto se possuirmos uma CPU por pixel pode-se rodar todas as sub-iterações subsequentes de uma única vez.

É importante ressaltar a importância de se atualizar o buffer que corresponde à imagem apenas no final de cada iteração (como pode ser visto no código fonte). A deleção dos pixels no instante de seu processamento altera o resultado do algoritmo, pois é importante que a imagem não se altere no decorrer de cada iteração.

Nota-se que algumas imagens após a aplicação do afinamento apresentam algumas distorções. Para reduzir tais distorções apresentadas pode-se utilizar a técnicas de pré-processamento, Stentiford, ou a técnica de Holt 1987, a qual é um melhoramento da técnica de afinamento de *Zhang-Suen* .