Yuri de Mello Villar

Projeto de um ASIC para extração de vértices de imagens em tempo real

Dissertação submetida ao corpo docente da Coordenação dos Cursos de Pós-Graduação em Engenharia Elétrica da Universidade Federal da Paraíba - Campus II como parte dos requisitos necessários para obtenção do grau de Mestre em Engenharia Elétrica.

Elmar Uwe Kurt Melcher, Dr.
Orientador

Campina Grande, Paraíba, Brasil

V719p Villar, Yuri de Mello
Projeto de um ASIC para extracao de vertices de imagens em tempo real / Yuri de Mello Villar. - Campina Grande, 1997.
69 f.

Dissertacao (Mestrado em Engenharia Eletrica). Universidade Federal da Paraiba, Centro de Ciencias e Tecnologia.

1. Analise de Imagem. 2. Procesamento Digital de Imagens - Engenharia Eletrica. 3. Imagens em Tempo Real. I. Melcher, Elmar Uwe Kurt, Prof. Dr. II. Universidade Federal da Paraiba, Campina Grande (PB)

CDU 621.397.31(043)

PROJETO DE UM ASIC PARA EXTRAÇÃO DE VÉRTICES DE IMAGENS EM TEMPO REAL

YURI DE MELLO VILLAR

Dissertação Aprovada em 12.12.1997

PROF. ELMAR UWE KURT MELCHER, Dr., UFPB
Orientador

PROF. JOÃO MARQUES DE CARVALHO, Ph.D., UFPB
Componente da Banca

PROF. PAULO CÉSAR CORTEZ, D.Sc., UFC
Componente da Banca

CAMPINA GRANDE - PB
Dezembro - 1997

Projeto de um ASIC para extração de vértices de imagens em tempo real

Yuri de Mello Villar

Dissertação de Mestrado aprovada em 12/12/1997

Elmar Uwe Kurt Melcher, Dr.
Orientador

Paulo César Cortez, Dr.
Membro da banca

João Marques de Carvalho, Ph.D.
Membro da banca

Campina Grande, Paraíba, Brasil, dezembro/1997

Resumo

A modelagem poligonal é uma ferramenta utilizada no processamento digital de imagens que, a partir dos vértices extraídos dos objetos de uma cena, fornece modelos que serão aplicados em tarefas como reconhecimento automático e compressão de vídeo. A partir do desenvolvimento de uma arquitetura paralela, com integração em VLSI, um algoritmo *data-flow* é implementado, de modo a extrair os vértices de contornos pré-processados e obter seus respectivos modelos poligonais. O algoritmo é baseado na transformada de Hough e detecta quando uma janela 8×8 de uma imagem contém ou não um vértice. O resultados de simulações da arquitetura produzida mostram que os modelos poligonais podem ser gerados a altas taxas de processamento o que a torna ideal para aplicações em tempo real.

Abstract

Polygonal modelling is a method used in digital image processing for object segmentation. Based on vertices extracted from a scene it produces models that can be applied in visual recognition tasks and in video compression. A data-flow algorithm for vertex extraction from pre-processed contour images was implemented in a VLSI circuit in order to obtain polygonal models. The algorithm is based on Hough Transform and detects if an 8x8 image window contains c vertex or not. Simulation results of the proposed arquitecture show that the polygonal models can be produced at high processing rate, turning the circuit ideal for real time aplications.

Agradecimentos

Agradeço às seguintes pessoas sem as quais este trabalho não seria possível: minha família e a família de Kedma, pela ajuda inestimável. Ao professor Elmar, pela oportunidade e por toda ajuda. Ao professor João Marques, pelas sugestões ao texto. À professora Aurizete, por esclarecer várias dúvidas. Aos colegas Marcos, Ricardo, Valteir e Fred, pelas discussões e trabalhos produzidos. E, principalmente, a minha esposa e filha por entenderem e suportarem minha ausência. Obrigado a todos.

Para Kedma e Ingrid. Amo vocês.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviações e Símbolos

ASIC - Circuito Integrado de Aplicação Específica.

Bit - Binary Digit; dígito binário.

Bits/pixel - quantidade de bits utilizados para representar um pixel.

CAD - Computer Aided Design; projeto auxiliado por computador.

CI - circuito integrado.

CMOS - Complementary MOSFET; tecnologia que utiliza transistores MOSFET complementares.

EPROM - Eareseble Programmable Read Only Memory; memória apenas de leitura apagável e programável.

FIFO - First In First Out; memória serial onde o primeiro dado a entrar é o primeiro a sair.

FPGA - Field Programmable Gate Array.

HDL - Hardware Description Language; linguagem de descrição de hardware.

MHz - megahertz.

mm^2 - milímetros quadrados.

MPEG - Motion Picture Expert Group; algoritmo para compressão de sinais de vídeo.

ns - nanossegundos.

Pixel - Picture Element; elemento da imagem.

Quadros/s - taxa de transmissão de uma imagem.

RAM - Random Acess Memory; memória de acesso aleatório.

VLSI - Very Large Scale Integration; integração em escala muito grande.

Capítulo 1

Introdução

O processamento digital de imagens é uma área de grande importância e que tem experimentado um acentuado crescimento nos dias atuais. Através dele, pode-se transformar uma dada imagem de entrada, de modo a obter outra de saída mais adequada para uma determinada finalidade. Dessa forma, estes métodos de processamento são utilizados em uma ampla variedade de problemas que, apesar de não relacionados, possuem a mesma necessidade de ferramentas, capazes de melhorar a informação pictorial para a interpretação humana, ou processar informações da cena para a percepção de máquinas autônomas. Exemplos típicos de aplicação são: diagnósticos médicos, inspeção industrial, robótica, entre outros.

O uso de modelos em processamento de imagens tem por objetivo permitir a classificação de objetos de um quadro em grupos (classes), através da medição e análise das características extraídas dos mesmos, obedecendo às propriedades de discriminação, confiabilidade e independência, apresentando também um número pequeno de características [GW87].

Dentre os diversos tipos, a modelagem poligonal tem sido, tradicionalmente, um dos métodos mais populares para a representação de formas e ainda é amplamente utilizada em aplicações em que o reconhecimento de objetos bidimensionais ou superfícies é necessário [SR92, SM92]. Os modelos poligonais têm a vantagem de ser uma representação local, ou seja, eles preservam as características locais, permitindo, portanto,

o reconhecimento de objetos mesmo na presença de oclusão parcial. Adicionalmente, eles podem ser realizados de forma insensível a rotações, translações e escalonamentos (uma necessidade para qualquer sistema prático de reconhecimento) e precisam de menos esforços computacionais que aproximações polinomiais de ordem mais elevada. Uma desvantagem desse método, é o fato da representação fornecida por modelos poligonais não ser normalmente tão compacta quanto aquelas baseadas em características globais como descritores de transformadas ou de momentos. Uma análise mais detalhada das questões envolvidas nestes e em outros modos de representação de formas foi feita por Pavladis [Pav78]. Detalhes adicionais sobre as vantagens da modelagem poligonal podem ser também encontradas na literatura [AF86, KD82].

Atualmente, muita atenção tem sido direcionada à compressão e codificação de vídeo, devido à importância associada a tais assuntos [SR92, SM92, Gha91]. Na verdade, a codificação de imagens, baseada em modelos, é uma técnica poderosa de compactação de cenas, dominadas por imagens de *cabeça e busto*, como acontece nas normas MPEG-4, impondo códigos com baixíssima taxa de bits. Para realizar este tipo de operação, a taxa de compressão e a qualidade de reconstrução são duas questões de grande importância. O uso de codificação adaptativa, baseada em modelos apropriados à imagem, é uma das aproximações mais eficientes para se atingir estes dois objetivos simultaneamente. Portanto, a modelagem poligonal é uma ferramenta importante para esses esquemas, porque o codificador procura em cada cena por alvos que são identificados de acordo com alguns modelos básicos de objetos [PKL92, eMS96, HAS89, GFGT95, ZLRG95]. Altas taxas de compressão são obtidas cada vez que um objeto é identificado, isso porque ele pode ser enviado através de um número de quadros em uma sequência e apenas as mudanças subsequentes nos parâmetros do modelo (forma, movimento, etc.) precisam ser transmitidas [PKL92].

A tarefa de modelagem poligonal consiste, essencialmente, em identificar, no contorno da forma, os pontos que serão os vértices da aproximação poligonal fechada.

A maioria dos trabalhos apresentados até agora, sobre algoritmos de extração de vértices de imagens digitais [Ram72, Cor95], admitem uma implementação posterior em software num computador de utilidade geral, ou num processador de sinal especializado.

No entanto, para aplicações em imagens em tempo real, é fundamental considerar a velocidade de processamento e a sua variação com o aumento da sequência de pontos, usada para a modelagem. Apesar de serem mais simples, os algoritmos sequenciais não representam uma boa escolha para este tipo de operação. Eles necessitam de que uma lista dos pontos de contorno do objeto esteja previamente disponível, a fim de realizar a detecção dos vértices para a aproximação poligonal. Porém, aplicações como sistemas de visão para robôs móveis ou compressão de vídeo podem exigir quadros de imagens de 512×512 pixels e 256 níveis de cinza (8 bits), a serem processados a uma taxa de 30 quadros por segundo. A operação em tempo real, usando esses valores, precisaria de um tempo de processamento de 120 nanossegundos para cada elemento de imagem. Este valor de performance torna-se claramente fora de alcance para qualquer computador de utilidade geral, quando se considera a quantidade de operações normalmente envolvida no cálculo de cada um dos pixels de saída. Portanto, para o processamento em tempo real, as características de performance dos algoritmos *data-flow* fazem deles a melhor escolha. Neles a cena é trabalhada ao mesmo tempo em que é adquirida, utilizando janelas retangulares de um determinado tamanho, 3×3 ou 5×5 pixels por exemplo. Os elementos da imagem são inseridos serialmente, um após o outro, uma linha após a outra. Para cada pixel de entrada, um algoritmo *data-flow* produz um pixel de saída, que é determinado como uma função do pixel de entrada correspondente e de um conjunto de pixels vizinhos a ele.

Dentro das considerações expostas anteriormente, pretenderemos com este trabalho desenvolver algoritmos e arquiteturas paralelas que implementarão as funções necessárias à extração de vértices dos objetos a serem modelados. Serão utilizadas ferramentas CAD para projetos VLSI, auxiliando o desenvolvimento, teste e síntese das estruturas e o resultado final será apresentado na forma de um *layout* que, posteriormente, será integrado em um ASIC, juntamente com as outras etapas do processo de obtenção dos modelos poligonais.

O restante do texto se encontra organizado da seguinte forma:

No capítulo 2, serão discutidos brevemente os processos utilizados para realizar

a segmentação e a descrição dos componentes das imagens, bem como as suas inter-relações.

No capítulo 3, o sistema de modelagem poligonal proposto será apresentado. Aqui, todas as partes envolvidas na tarefa da extração dos vértices serão detalhadas minunciosamente.

No capítulo 4, serão mostrados todos os passos envolvidos na implementação em hardware do algoritmo proposto.

O capítulo 5 apresentará os resultados dos testes obtidos e fará comparações com as simulações geradas por outros métodos de reconhecimento de formas. Os resultados da implementação em hardware também serão apresentados.

Finalizando o texto, apresentaremos as conclusões do trabalho.

Capítulo 2

Segmentação e descrição de imagens

Este capítulo apresenta o contexto no qual se insere este trabalho. Nele é dada uma maior atenção aos componentes da imagem e suas interrelações. Este tipo de abordagem é fundamental em áreas como análise automática de cenas e reconhecimento de padrões, em que se deseja utilizar a máquina para extrair informação que requer conhecimento detalhado dos componentes da imagem, a nível de objeto.

A discussão deste capítulo é organizada em quatro principais tópicos: segmentação, descritores regionais, descritores relacionais e descritores de similaridade. A figura 2.1 ilustra a ordem hierárquica destas operações.

2.1 Segmentação

O objetivo da segmentação é particionar a imagem em regiões significativas, sendo a definição de "região significativa" uma função do problema a ser considerado. Por exemplo, em uma imagem derivada de uma cena tridimensional, o propósito da segmentação pode ser o de identificar regiões correspondentes aos objetos presentes. Em aplicações de reconhecimento aéreo, por outro lado, a extração de áreas correspondentes a zonas residenciais, industriais, agrícolas ou de terreno natural, será a finalidade desejada.

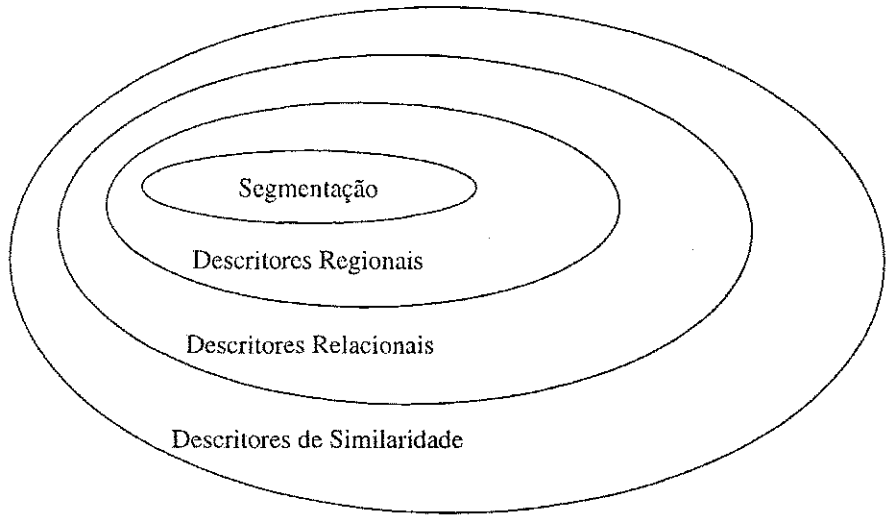


Figura 2.1: Ordem hierárquica dos procedimentos para descrição de imagens.

O problema da segmentação deve ser considerado como um processo dependente tanto de um ponto, como de uma região. A primeira categoria trata de métodos que examinam a imagem pixel a pixel. A segunda, utiliza técnicas que usam a informação contida em uma determinada vizinhança. Entretanto, em ambos os casos, deve-se ver a tarefa como um processo de tomada de decisão ou reconhecimento de padrões, cujo objetivo é o de estabelecer fronteiras entre as regiões.

Na sua forma mais simples, o processo de decisão utiliza uma única variável (por exemplo, a luminância de um pixel). Porém, aperfeiçoamentos podem ser conseguidos nesta etapa, considerando-se mais de uma variável, ao custo de uma maior complexidade envolvida. Tomando o conjunto das variáveis envolvidas como sendo $\{x_1, x_2, \ldots, x_n\}$, pode-se expressar esta informação na forma de um vetor X:

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \tag{2.1}$$

2.1.1 Operações pontuais

Limiar de nível de cinza

Uma técnica que é frequentemente utilizada para segmentar uma imagem, consiste em dividir a escala de níveis de cinza em bandas e usar limiares para determinar regiões ou obter pontos de fronteira.

Seja uma imagem $f(x, y)$, cujo histograma é mostrado na figura 2.2(a).

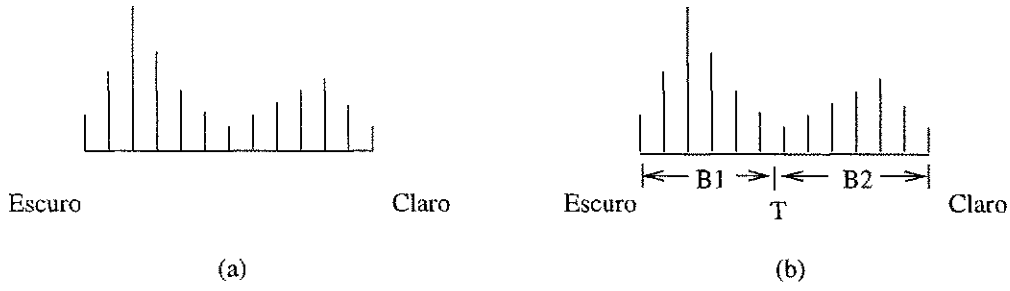


Figura 2.2: Limiarização de histogramas.

Este é um exemplo típico de uma cena de objetos superpostos a um fundo escuro. Para dar ênfase aos limites entre os elementos e o ambiente da imagem, o histograma é dividido em duas partes separadas pelo limiar de decisão T, como mostrado na figura 2.2 (b). O objetivo é selecionar T, de forma que a banda B_1 contenha, o mais próximo possível, os níveis de cinza do fundo da cena, enquanto que B_2 contenha os níveis dos objetos. Portanto, ao percorrer-se a imagem, uma mudança no nível de cinza de uma banda para outra denota a presença de uma fronteira. De forma a se obter tanto as fronteiras horizontais como as verticais, o procedimento pode ser o apresentado a seguir.

Para cada linha em $f(x, y)$ (com $x = 1, 2, \ldots, N-1$), uma linha correspondente deve ser criada em uma imagem intermediária $g_1(x, y)$, utilizando a seguinte relação para $y = 1, 2, \ldots, N-1$:

$$g_1(x,y) = \begin{cases} N_E & \text{se os níveis de } f(x,y) \text{ e } f(x,y-1) \text{ estão} \\ & \text{em diferentes bandas na escala de cinza} \\ N_F & \text{caso contrário} \end{cases} \tag{2.2}$$

na qual N_E e N_F são os níveis para a fronteira e o fundo da imagem, respectivamente.

Para cada coluna em $f(x,y)$ (com $y = 1,2,\ldots,N-1$), cria-se uma coluna correspondente em uma imagem intermediária $g_2(x,y)$, usando a seguinte relação para $x = 1,2,\ldots,N-1$:

$$g_2(x,y) = \begin{cases} N_E & \text{se os níveis de } f(x,y) \text{ e } f(x-1,y) \text{ estão} \\ & \text{em diferentes bandas na escala de cinza} \\ N_F & \text{caso contrário} \end{cases} \tag{2.3}$$

A imagem desejada, consistindo dos pontos de contorno entre os objetos e o ambiente da cena (definidos por T), é obtida usando a relação para $x,y = 1,2,\ldots,N-1$:

$$g(x,y) = \begin{cases} N_E & \text{se } g_1(x,y) \text{ ou } g_2(x,y) \text{ é igual a } N_E \\ N_F & \text{caso contrário} \end{cases} \tag{2.4}$$

O procedimento acima é facilmente generalizado para um número maior de níveis de cinza, aumentando-se o poder da técnica de extração de fronteiras [GW77].

Limiarização ótima

Se é conhecido a priori que a imagem possui apenas dois níveis de cinza principais, o histograma de tal cena pode ser estimado através da função densidade de probabilidade da luminância. Esta função será a soma ou mistura de duas densidades unimodais, uma para a região clara e outra para a região escura. Os parâmetros desta mistura serão proporcionais às áreas de cada nível de cinza do quadro. Se as formas das densidades

são conhecidas ou assumidas, é possível determinar um limiar ótimo (em termos de um erro mínimo) para segmentar a imagem em duas regiões de luminância.

Considerando distribuições Gaussianas, o valor de T é obtido através da resolução da equação quadrática:

$$AT^2 + BT + C = 0 \tag{2.5}$$

na qual:

$A = \sigma_1^2 - \sigma_2^2$,

$B = 2(\mu_1\sigma_2^2 - \mu_2\sigma_1^2)$,

$C = \sigma_1^2\mu_2^2 - \sigma_2^2\mu_1^2 + \sigma_1^2\sigma_2^2 \ln(\sigma_1 P_1/\sigma_2 P_2)$,

μ_1 e μ_2: valores médios dos níveis de cinza,

σ_1 e σ_2: desvios padrões em torno das médias,

P_1 e P_2: probabilidade a priori dos dois níveis.

A possibilidade de duas soluções indica que podem ser necessários dois limiares para obter a solução ótima.

Se as variâncias forem iguais, $\sigma^2 = \sigma_1^2 = \sigma_2^2$, um único limiar será suficiente:

$$T = \frac{\mu_1 + \mu_2}{2} + \frac{\sigma^2}{\mu_1 - \mu_2} \ln\left(\frac{P_2}{P_1}\right) \tag{2.6}$$

Se as probabilidades P_1 e P_2 forem iguais, o resultado será apenas a média das médias μ_1 e μ_2. O cálculo do limiar ótimo pode ser realizado facilmente para outras densidades unimodais de forma conhecida, como a Raleigh e a log-normal [Pap65].

Uso de várias variáveis

Em alguns casos, um sensor pode usar mais de uma variável para caracterizar cada pixel na imagem. Um exemplo são as fotografias coloridas, nas quais as componentes vermelhas, verdes e azuis são utilizadas para compor as cores do quadro. Neste caso,

cada pixel é caracterizado por três valores e se torna possível construir um histograma tridimensional. O procedimento básico para a limiarização é o mesmo empregado para o caso de uma variável.

Considerando três imagens de 16 níveis correspondendo às componentes RGB, é formado um cubo com $16 \times 16 \times 16$ células. Em cada uma destas células é inserido o número de pixels cujas componentes têm intensidades correspondentes àquela célula em particular. O conceito de limiarização agora torna-se o de encontrar agrupamentos de pontos no espaço tridimensional. Se forem encontrados K agrupamentos significantes no histograma, a imagem pode ser segmentada, atribuindo-se K intensidades diferentes aos pixels, de acordo com a maior proximidade de suas componentes RGB a um determinado grupo [GW77].

Este processo é facilmente estendido a mais variáveis, resultando, porém, em um número maior de agrupamentos e em uma maior complexidade [GW77].

2.1.2 Procedimentos regionais

A alternativa às técnicas pontuais, para segmentar a imagem em regiões com propriedades semelhantes, é dada pelo uso de procedimentos regionais. Nestes, o processo de decisão envolvido consiste em atribuir cada ponto a uma região, de acordo com uma determinada característica de seleção. O método empregado mais frequentemente para estabelecer as propriedades das diversas regiões são as diferenças de níveis de cinza, porém a textura e as diferenças de cores também são utilizadas para este fim [RK76].

Processamento com máscaras

O conceito de janelas e máscaras tem encontrado larga aceitação em aplicações de segmentação, devido a sua simplicidade. Em imagens digitais, uma máscara é uma matriz designada para detectar alguma propriedade regional invariante [RK76].

Como um exemplo deste procedimento, será considerada uma imagem que contém uma intensidade de fundo constante e partículas isoladas (pontos), cujos níveis de

cinza são diferentes do fundo. Assumindo que as distâncias entre as partículas são maiores que $\left[(\Delta x)^2 + (\Delta y)^2\right]^{\frac{1}{2}}$, em que Δx e Δy são as distâncias de amostragem nas direções x e y, respectivamente. Estes pontos na imagem podem ser detectados, usando a máscara mostrada na figura 2.3, de acordo com o seguinte procedimento: o centro (marcado com o número 8) é movido em torno da imagem, pixel a pixel. Em cada posição, multiplicam-se os pontos da janela da imagem delimitada pela máscara, pelos valores dos coeficientes correspondentes da máscara. Por fim, os resultados são somados. Desta forma, se todos os pontos dentro da área delimitada tiverem o mesmo valor (do fundo da imagem), o módulo da soma será nulo. Porém, se o centro estiver sobre uma partícula, o módulo da soma será diferente de zero. Se o ponto ocupar uma posição que não seja a central, o resultado também será maior que zero, mas com uma magnitude inferior a do caso anterior. Estes módulos de valores reduzidos podem ser eliminados através da escolha de um limiar. Portanto, uma partícula é detectada, quando o resultado excede o critério de decisão escolhido (ou seja, quando ocupa a posição central).

-1	-1	-1
-1	8	-1
-1	-1	-1

Figura 2.3: Máscara para detecção de pontos.

A técnica descrita acima pode facilmente ser generalizada para uma janela $n \times n$, da forma mostrada a seguir.

Sejam $w_{11}, w_{12}, \ldots, w_{nn}$ os valores representando os pesos (coeficientes) da máscara e $x_{11}, x_{12}, \ldots, x_{nn}$, os níveis de cinza dos pixels dentro da janela. Como exposto anteriormente, é feito o produto interno dos vetores:

$$W = \begin{bmatrix} w_{11} & \ldots & w_{1n} \\ w_{21} & \ldots & w_{2n} \\ \vdots & \ddots & \vdots \\ w_{n1} & \ldots & w_{nn} \end{bmatrix}$$

e:

$$X = \begin{bmatrix} x_{11} & \dots & x_{1n} \\ x_{21} & \dots & x_{2n} \\ \vdots & \ddots & \vdots \\ x_{n1} & \dots & x_{nn} \end{bmatrix}$$

O produto interno de W e X é, portanto, definido por:

$$WX = w_{11}x_{11} + w_{12}x_{12} + \ldots + w_{nn}x_{nn} \tag{2.7}$$

cujo resultado pode então ser submetido ao critério de decisão desejado.

Um outro exemplo de processamento com máscaras consiste no gradiente de Sobel, aplicado em regiões com dimensões 3×3 da imagem, para a detecção de contornos [MC96]. As janelas utilizadas nesta operação são mostradas na figura 2.4.

-1	-2	-1
0	0	0
1	2	1

W_x

-1	0	1
-2	0	2
-1	0	1

W_y

Figura 2.4: Operadores Sobel.

Tomando uma determinada região 3×3, R, a componente do gradiente no pixel central, na direção x, G_x, é definido pelo produto interno:

$$G_x = W_x R \tag{2.8}$$

e na direção y, G_y é calculado por:

$$G_y = W_y R \tag{2.9}$$

A magnitude do gradiente no ponto central da área considerada será, então:

$$G = \left[G_x^2 + G_y^2\right]^{\frac{1}{2}} \tag{2.10}$$

Uma alternativa (aproximação) para este resultado, de implementação mais fácil por computadores, é obtida empregando-se valores absolutos:

$$G = |G_x| + |G_y| \tag{2.11}$$

Como estas operações de segmentação não necessitam que toda a imagem esteja disponível para os cálculos, o que é ideal para operações em tempo real, os algoritmos propostos no capítulo 3 realizam suas operações por meio do processamento e comparação de máscaras [MC96, MdCdB+95, MCC96].

Crescimento de regiões

Como o objetivo da segmentação é particionar uma imagem ou conjunto de imagens em regiões, uma aproximação direta para este problema é tentar uma divisão da cena em áreas que satisfaçam um critério de similaridade, como por exemplo, grupos de pontos em regiões. A vantagem desta abordagem é que ela não produz apenas pontos de fronteira entre os diversos componentes da imagem, mas também, desde que satisfeitas as condições, pontos internos aos contornos [GW87].

De forma a realizar o agrupamento de pontos, três questões fundamentais devem ser resolvidas:

1. O número de regiões deve ser determinado,
2. As propriedades ou características que distinguem uma região das outras devem ser escolhidas,
3. Finalmente, um critério de similaridade ajustável, que produza uma segmentação significativa, é especificado.

Uma técnica que utiliza esta abordagem é o crescimento de regiões. Suponha-se que o número de áreas e a posição de um único ponto interno de cada uma são conhecidos. Então, pode-se desenvolver um algoritmo que, começando pelos pontos conhecidos, agrupe a eles todos os pontos vizinhos similares em níveis de cinza, cor, textura, gradiente, ou outra propriedade, formando assim as regiões da imagem [GW87].

Agrupamento de regiões

O procedimento descrito na seção anterior assume que um ponto inicial, em cada região, encontra-se disponível. Este raramente é o caso, na prática, e torna-se necessário, portanto, procurar pelas regiões diretamente. Uma possibilidade é utilizar um algoritmo de agrupamento de regiões.

Outra possibilidade é usar conhecimento a priori sobre a cena para definir os pontos iniciais. Por exemplo, dados coletados no solo para imagens de satélite.

O problema de se agrupar regiões aplica-se diretamente à segmentação de imagens e difere das técnicas de agrupamento em geral, pela necessidade dos pontos internos a um grupo serem contínuos no plano da imagem e semelhantes em propriedades. O procedimento geral consiste em, inicialmente, examinar o conjunto de características da cena, a fim de determinar o número e o posicionamento dos grupos. Então, aplicam-se estas definições à cena para obter as regiões desejadas. O critério de decisão (textura, cor, etc) pode ser utilizado em ambas as etapas e, também, critérios diferentes podem ser requeridos para cada passo [GW87].

2.2 Descritores regionais

Uma vez que uma região tenha sido identificada, é interessante caracterizá-la por um conjunto de descritores insensíveis a variações, como mudanças de tamanho, rotações e translações. O uso de descritores regionais permite, além da redução da quantidade de dados utilizada na representação, enfatizar características que ajudarão a distinguir regiões com diferentes atributos [Pav78]. Nesta seção, serão consideradas duas abordagens típicas para este problema.

2.2.1 Descritores de Fourier

Com os pontos de fronteira de uma área tendo sido determinados, é, frequentemente, de interesse extrair informações destes pontos que irão ser úteis para discriminar formatos

de regiões diferentes. A transformada discreta de Fourier (DFT) pode ser usada para este propósito.

Suponha que uma função contínua $f(x,y)$ é discretizada em uma sequência $f(m,n) = f(x_0 + m\Delta x, y_0 + n\Delta y)$, para $m = 0,1,2,\ldots,M-1$ e $n = 0,1,2,\ldots,N-1$. Desta forma a transformada discreta de Fourier [GW77] é dada pelas equações:

$$F(u,v) = \frac{1}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f(m,n)\exp\left[-j2\pi\left(\frac{um}{M}+\frac{vn}{N}\right)\right] \tag{2.12}$$

para $u = 0,1,2,\ldots,M-1$, $v = 0,1,2,\ldots,N-1$, e:

$$f(m,n) = \sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F(u,v)\exp\left[j2\pi\left(\frac{um}{M}+\frac{vn}{N}\right)\right] \tag{2.13}$$

para $m = 0,1,2,\ldots,M-1$ e $n = 0,1,2,\ldots,N-1$.

Tomando M amostras do contorno de uma região, podemos iniciar por um ponto arbitrário e percorrer toda a fronteira, calculando, através da equação 2.12, uma sequência de valores complexos, que resultará no descritor de Fourier do contorno [GW77].

Como a DFT é uma transformação linear e reversível, não há informação ganha ou perdida por este processo. Entretanto, certas manipulações simples deste domínio de representação de formas podem eliminar a dependência da posição, tamanho e orientação [GW87]. Portanto, dado um descritor de Fourier arbitrário, sucessivos passos podem realizar sua normalização, de forma a conseguir uma igualdade em um conjunto teste de descritores, não importando sua escala original, translações e rotações.

2.2.2 Momentos invariantes

Uma outra abordagem bastante utilizada na descrição de uma região é a técnica dos momentos invariantes.

Dada uma função contínua bidimensional $f(x,y)$, o momento de ordem $(p+q)$ é definido pela relação [WH78]:

$$m_{pq} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} x^p y^q f(x,y)dxdy \tag{2.14}$$

para $p, q = 0, 1, 2, \ldots$.

O teorema da unicidade afirma que, se $f(x, y)$ é contínua por partes e tem valores diferentes de zero em uma porção finita do plano xy, então existem momentos de todas as ordens e a sequência dos momentos, m_{pq}, é unicamente determinada por $f(x, y)$. Inversamente, m_{pq} determina unicamente $f(x, y)$ [Pap65].

Os momentos centrais são expressos da seguinte forma:

$$\mu_{pq} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} (x - \overline{x})^p (y - \overline{y})^q f(x, y) dx dy \tag{2.15}$$

na qual:

$$\overline{x} = \frac{m_{10}}{m_{00}} \quad , \quad \overline{y} = \frac{m_{01}}{m_{00}}$$

Os valores $\overline{x}$ e $\overline{y}$ são as coordenadas do centro de gravidade de um objeto [WH78]. Para uma imagem digital, a equação 2.15 torna-se:

$$\mu_{pq} = \sum_m \sum_n (m - \overline{m})^p (n - \overline{n})^q f(m, n) \Delta n \Delta m \tag{2.16}$$

Estes momentos independem da posição e tamanhos originais.

2.3 Descritores relacionais

Uma vez que a imagem tenha sido segmentada em regiões ou componentes, o próximo nível de abstração, na tarefa de descrevê-la, é organizar estes elementos em uma estrutura relacional significativa [RK76]. A seguir, são mostradas algumas técnicas para implementar esta etapa.

2.3.1 Técnicas baseadas em conceitos gramaticais

Como uma introdução a este conceito, considere-se a estrutura simples em forma de escada, mostrada na figura 2.5(a). Assumindo que este contorno foi segmentado de

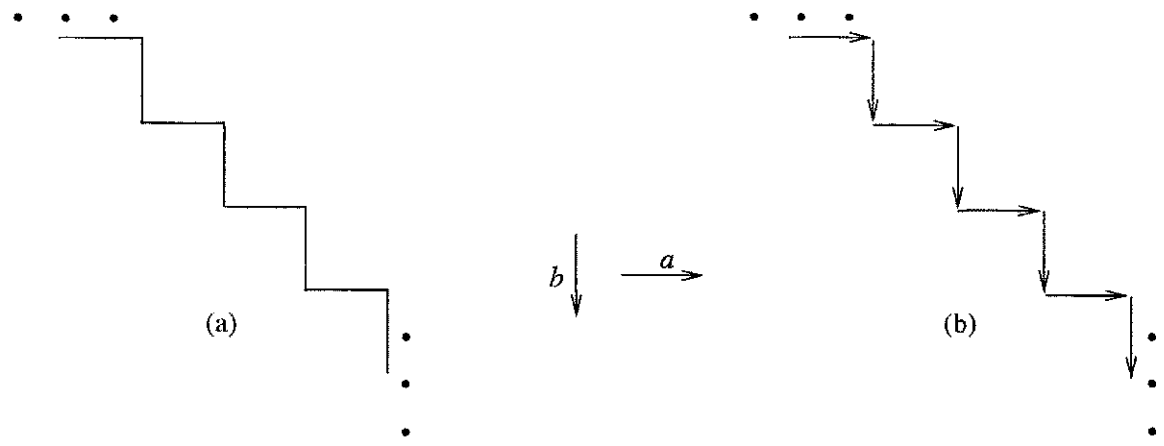


Figura 2.5: (a)Estrutura em forma de escada; (b)estrutura codificada.

uma imagem, deseja-se descrevê-lo de alguma maneira formal. Definindo dois elementos primitivos, a e b, a borda pode ser codificada na forma apresentada na figura 2.5(b).

A propriedade mais óbvia desta representação é a repetitividade dos símbolos a e b. Porém, uma abordagem para a descrição seria formular relações recursivas envolvendo estes elementos. Uma possibilidade é utilizar as seguintes regras de reescrita:

1. $S \rightarrow aA$,
2. $A \rightarrow bS$,
3. $A \rightarrow b$.

nas quais S (símbolo inicial) e A são variáveis e a e b os símbolos definidos anteriormente.

A primeira regra indica que S pode ser expressa pela primitiva a e variável A. Esta, por sua vez, pode ser reescrita por b e S ou por b apenas. Se A for expressa por bS, volta-se a primeira regra e o procedimento é repetido. Se A é substituída por b, o processo termina, pois nenhuma variável sobra na expressão. Nota-se que a relação entre a e b é preservada, pois estas regras forçam que o a seja sempre seguido por um b [GW77].

2.3.2 Código direcional

Sejam os pontos de um contorno fechado, mostrados na figura 2.6 (a). O procedimento de representação por código direcional toma um dos pontos da borda como ponto inicial e, a partir dele, indica as direções tomadas pelos demais pixels. As direções são mostradas na figura 2.6 (b).

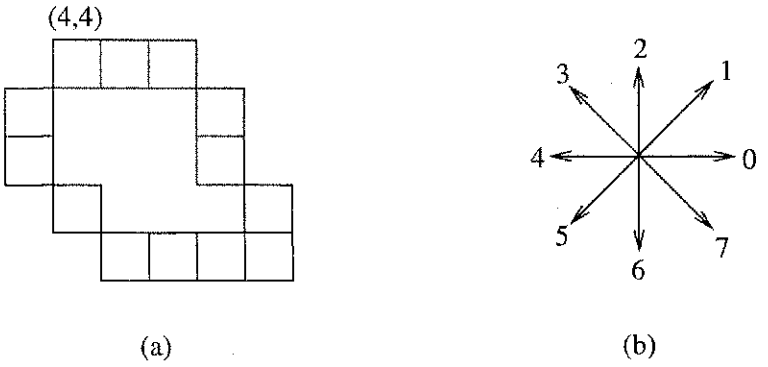


Figura 2.6: (a) e (b)Código direcional.

Portanto, para o contorno apresentado (tomando como ponto inicial o pixel com coordenadas (4,4)) a representação será dada por: $(4,4),0,0,7,6,7,6,4,4,4,3,3,2,1$.

Esta forma de descrição apresenta como principais vantagens a de ser compacta, poder ser gerada simultaneamente ao rastreamento do contorno e representar apenas o contorno da imagem.

2.3.3 Modelos poligonais

Seja a curva fechada C, mostrada na figura 2.7(a). Nela são apresentandos os pontos de borda segmentados de uma imagem, formando um conjunto com N elementos, $C = a_1, a_2, \ldots, a_N$, no qual $a_1 = a_N$. O objetivo da modelagem poligonal é, então, aproximar a curva C por um polígono P, formado a partir dos vértices extraídos da curva fechada, obedecendo a um determinado critério e limitado por um erro máximo de aproximação [Cor95].

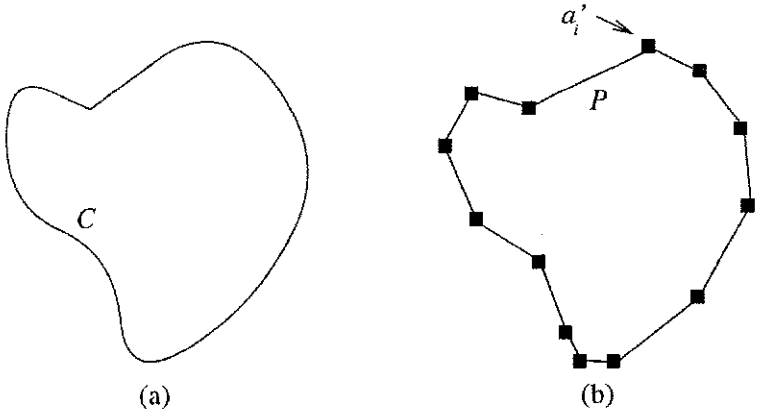


Figura 2.7: (a)Curva fechada C; (b)modelagem poligonal da curva C.

Portanto, considerando-se: um conjunto de vértices $V = a'_1, a'_2, \ldots, a'_{N'}$, no qual $V \subseteq C$ e $N' \leq N$, S_k um subconjunto de C cujos elementos pertencem ao intervalo $I_k = [a_i, a_j]$, com $a_i = a'_{k-1}, a_j = a'_k$ e $k = 1, \ldots, N'$ e o erro máximo $\in$, o critério de aproximação pode ser descrito pela função:

$$f(S_k) \leq \in \tag{2.17}$$

A definição de uma representação poligonal para a curva C implica que cada subconjunto S_k é aproximado por uma reta, como mostrado na figura 2.7(b), cuja distância máxima aos pontos S_k é dada por $f(S_k)$ [Cor95].

A modelagem poligonal foi o método escolhido para a geração dos modelos a partir dos vértices extraídos da imagem pois, como citado, são representações locais e também, como os métodos da seção anterior, pode ser realizada indiferentemente a rotações, translações e escalonamentos, facilitando o reconhecimento numa fase posterior [AF86].

2.4 Descritores de similaridade

Medidas de similaridade podem ser estabelecidas em qualquer nível de complexidade de uma imagem, variando desde o caso trivial da comparação entre dois pixels, até o problema complexo de determinar, de alguma maneira significativa, o grau de similaridade entre duas ou mais cenas.

2.4.1 Medidas de distâncias

Considere, por exemplo, os descritores de momentos da seção 2.2.2. Se os momentos para duas regiões são arranjados na forma de dois vetores, $\mathbf{x_1}$ e $\mathbf{x_2}$, a distância entre eles será dada por:

$$D(\mathbf{x_1}, \mathbf{x_2}) = \|\mathbf{x_1} - \mathbf{x_2}\| = \sqrt{(\mathbf{x_1} - \mathbf{x_2}) \bullet (\mathbf{x_1} - \mathbf{x_2})} \tag{2.18}$$

Este valor pode ser usado como medida de similaridade entre os dois conjuntos [GW87].

Isto se torna particularmente atrativo se for preciso comparar um dado descritor de origem desconhecida com outros cujas características foram previamente estabelecidas. Sabendo-se que os conhecidos são denotados por $\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_L}$ e o desconhecido por $\mathbf{x}$, então $\mathbf{x}$ será mais similar a x_i que a x_j se:

$$D(\mathbf{x}, \mathbf{x_i}) < D(\mathbf{x}, \mathbf{x_j}) \tag{2.19}$$

para $j = 1, 2, \ldots, L, j \neq i$. Esta abordagem pode ser usada com muitos descritores, dependendo apenas da possibilidade de serem expressos na forma vetorial.

2.4.2 Correlação

Dada uma imagem digital $f(m,n)$ de tamanho $M \times N$, pode-se determinar se ela contém uma região que é similar a outra região $w(m,n)$ de tamanho $J \times K$, onde $J < M$ e $K < N$. Um método utilizado para a solução deste problema é realizar a correlação entre $w(m,n)$ e $f(m,n)$ [RK76].

Na forma mais simples, a correlação entre estas funções é dada por:

$$R(u,v) = \sum_m \sum_n f(m,n) w(m-u, n-v) \tag{2.20}$$

onde $m = 0, 1, 2, \ldots, M-1, n = 0, 1, 2, \ldots, N-1$ e a soma é tomada sobre toda a região da imagem onde $w(m,n)$ é definida.

O procedimento é mostrado na figura 2.8. Para qualquer valor de (m,n), dentro de $f(m,n)$, aplica-se a equação 2.19 para obter um valor de R. Enquanto m e n são

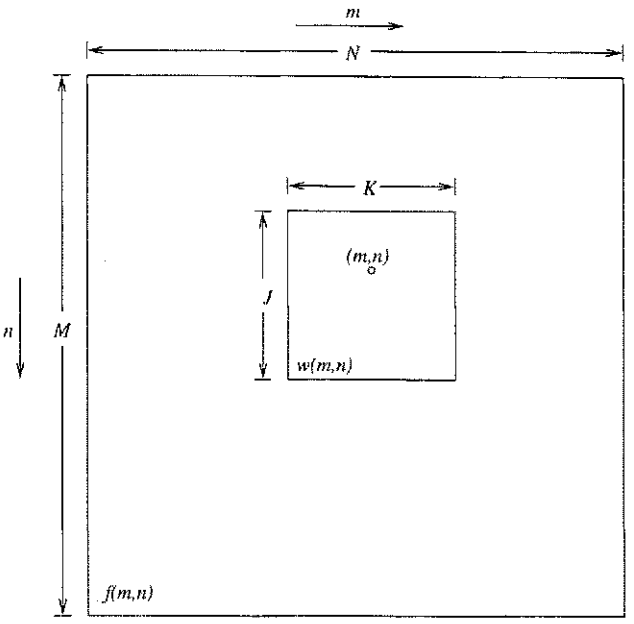


Figura 2.8: Correlação entre $f(m,n)$ e $w(m,n)$.

variados, $w(m,n)$ move-se sobre a imagem e a função $R(u,v)$ é determinada. O valor máximo de $R(u,v)$ indica a posição a que $w(m,n)$ melhor se aproxima de $f(m,n)$.

Outro procedimento, semelhante à correlação, é a técnica de *block matching* [KP89]. Esta operação, porém, em vez da equação 2.20, utiliza as diferenças absolutas entre a imagem e a janela para a estimação de movimentos.

2.4.3 Similaridades estruturais

Descrições de similaridades estruturais são, em geral, mais difíceis de formular e aplicar, devido ao fato de que o conceito de uma estrutura significativa é altamente dependente do problema a ser considerado. Se uma região fosse caracterizada pelo número de furos que contivesse, então uma figura de um número "8" seria mais similar a um "B" do que a um "A". Entretanto, este descritor poderia ser importante para estabelecer a similaridade entre imagens de faces humanas, através de uma caracterização apropriada [RK76].

Exemplos típicos que podem ser utilizados para medidas de semelhanças são comprimentos de segmentos retos, ângulos entre segmentos, características de luminância, áreas de regiões, posições de regiões na imagem em relação a outras, etc [RK76]. Porém, a extração automática de alguns destes componentes de uma cena torna-se a parte mais desafiadora do problema.

No próximo capítulo, apresentaremos os métodos de segmentação propostos para a extração dos vértices dos objetos presentes em uma cena.

Capítulo 3

Algoritmo Proposto

Como mencionado anteriormente, a implementação do processamento de imagens em hardware é bastante otimizada pela utilização de estruturas do tipo *data-flow*. Nelas, a performance e a capacidade de armazenamento de dados para realizar os cálculos, dependem diretamente do tamanho da janela usada, como mostrado na tabela 3.1. Os valores apresentados são válidos para cenas com 512×512 pixels, 5 bits/pixel e processados a uma taxa de 30 quadros/s. Os atrasos são muito pequenos e ideais para o uso em tempo real [MCC96].

tamanho da janela	**armazenamento** (bits)	**atraso da imagem** (μs)
3×3	1024	61.7
5×5	2048	123.2
7×7	3072	184.8
8×8	4096	246.2

Tabela 3.1: Performance dos algoritmos *data-flow*.

Um modelo geral do sistema proposto para a obtenção dos pontos para a modelagem poligonal é apresentado na figura 3.1.

O processo é dividido em três etapas:

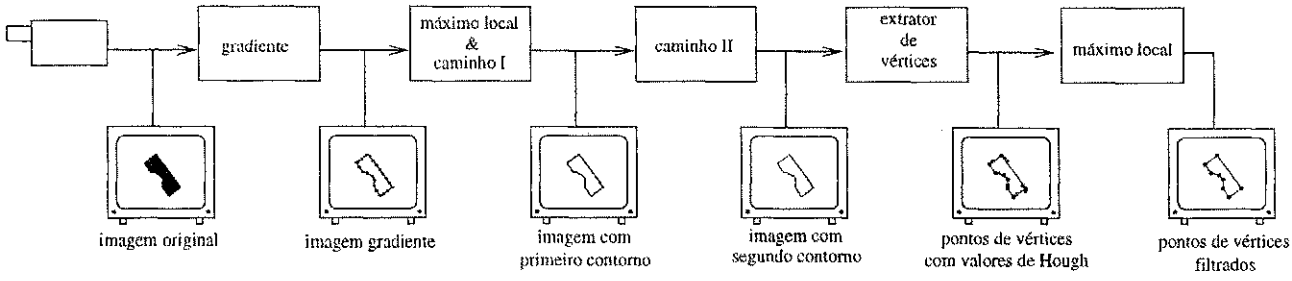


Figura 3.1: Algoritmos *data-flow* para a extração dos contornos e vértices.

- A aquisição e processamento das imagens da câmara para os algoritmos de extração de contornos e vértices,
- A extração dos contornos dos objetos presentes na cena,
- A extração dos vértices dos contornos processados.

O objetivo proposto por esse trabalho é o de implementar todas as etapas de processamento do terceiro item.

3.1 Extração de contornos

A obtenção dos contornos presentes na imagem é realizada pelas etapas descritas brevemente a seguir.

3.1.1 Gradiente Spline

O objetivo do cálculo do gradiente é o de extrair as bordas dos objetos presentes na cena. Este gradiente é calculado pela técnica do processamento por máscaras (seção 2.1.2), utilizando as máscaras mostradas na figura 3.2.

Portanto, o gradiente bidimensional é definido por [MNaMdC+97]:

$$g_p = 3\left(|g_{px}| + |g_{py}|\right) + 2\left(|g_{p1}| + |g_{p2}|\right) \tag{3.1}$$

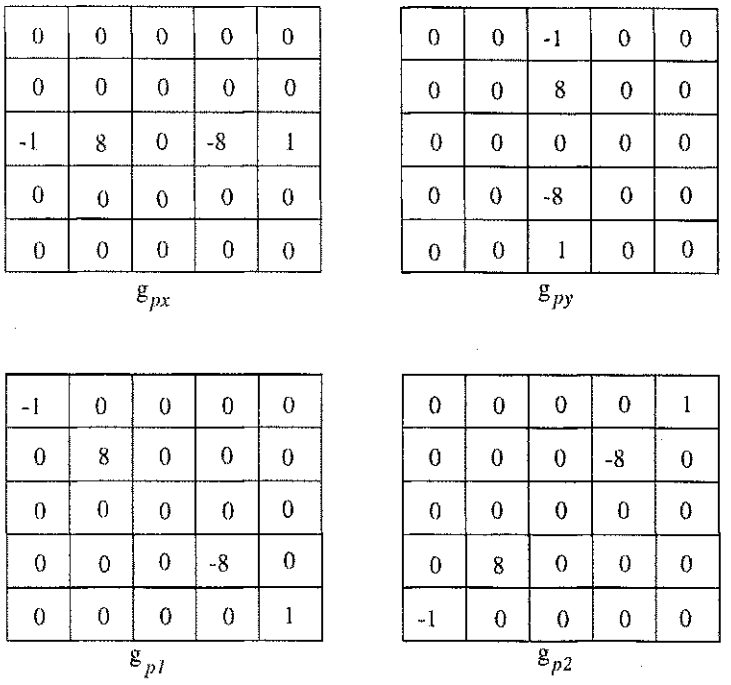


Figura 3.2: Máscaras utilizadas no cálculo do gradiente Spline.

Uma das principais vantagens deste operador gradiente é que ele é pouco sensível ao ruído. Isto se deve ao fato de o gradiente Spline utilizar uma janela 5×5 e determinar seu resultado a partir do valor médio das derivadas nas direções horizontal, vertical e diagonais.

Além disso, as bordas produzidas por este algoritmo são bastante precisas, fazendo com que os valores de gradiente dos pixels do contorno sejam significantemente maiores que aqueles na sua vizinhança, pois a interpolação através de splines cúbicas aproxima melhor a função original.

3.1.2 Máximo local e Caminho I

Como as bordas produzidas pelo gradiente Spline são muito "grossas", ou seja, possuem vários pixels de largura, são utilizadas algumas técnicas de refinamento de contornos, mostradas nesta e na próxima seção.

Nesta fase do processamento, a imagem gradiente obtida é submetida a dois critérios de avaliação. Se ambos forem validados, o pixel central da janela torna-se o pixel de saída, ou seja, é reconhecido como pertencente ao contorno. Caso contrário, o pixel de saída torna-se zero.

O critério do máximo local é baseado numa janela 5×5 [MNaMdC+97]. É definido um peso associado ao pixel central, de modo a ajudar na decisão a ser tomada:

$$W_m = \begin{cases} 0 & \text{, se } N_c = 0 \\ \\ 2N_s + N_e + p_c & \text{, caso contrário} \end{cases} \tag{3.2}$$

em que:

W_m =peso do pixel central,

N_s =número de pixels menores que o pixel central,

N_e =número de pixels iguais ao pixel central,

p_c =valor de nível de cinza do pixel central ($0 \leq p_c < 32$).

O máximo local é validado, então, se o valor do pixel central for maior que uma determinada porcentagem do valor máximo da escala de nível de cinza e se seu peso for maior que um limiar de decisão. Na implementação final, estes dois valores são transmitidos ao CI na forma de parâmetros ajustáveis.

O critério caminho I também trabalha com janelas 5×5 [MNaMdC+97]. Este critério determina se o pixel central da janela pertence a uma linha de contorno contínua, que passa por ela. Para realizar isso, todos os caminhos possíveis através de uma janela 3×3, mostrados na figura 3.3 dentro dos quadrados formados pelos pixels nas posições 1, junto a suas possíveis continuações, representadas pelas posições 2 e 3, devem ser checadas. Os caminhos possíveis são derivados dos primitivos apresentados na figura 3.3, por meio de rotinas de transposição e troca de linhas e colunas. Após a eliminação das repetições geradas por essas operações, um total de 44 caminhos diferentes são obtidos.

Caminho I será validado se o pixel central pertencer a dois dos oito caminhos com os maiores valores, determinados através da seguinte soma ponderada dos valores de

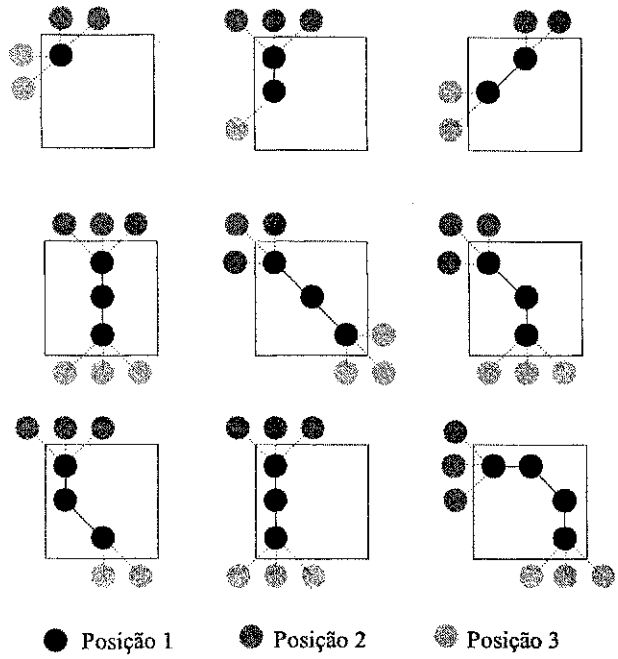


Figura 3.3: Segmentos utilizados em Caminho I.

seus pixels:

$$W_s = \begin{cases} \frac{M_2+M_3}{2} + \frac{12}{N_{p1}} \sum_{i=1}^{N_{p1}} p_i & \text{, se condição} \\ 0 & \text{, caso contrário} \end{cases} \tag{3.3}$$

na qual:

W_s = soma ponderada,

N_{p1} = número de posições 1 no caminho 3×3,

p_i = valor de nível de cinza do i-ésimo pixel na posição 1,

M_n = maior valor entre os pixels na posição n,

condição = p_i's e M_n's devem ser todos maiores que determinada porcentagem do valor máximo da escala de nível de cinza (parâmetro ajustável).

3.1.3 Caminho II

O algoritmo caminho II [MNaMdC+97] utiliza apenas caminhos contendo ao menos 3 pixels nas posições 1, como está mostrado na figura 3.4. Aplicando as mesmas técnicas usadas em caminho I, são obtidos 28 caminhos derivados.

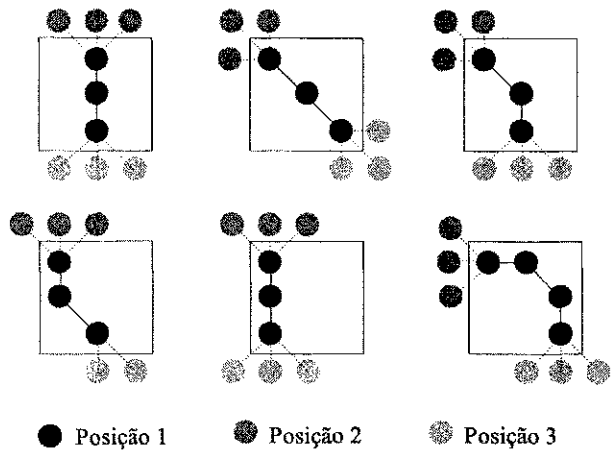


Figura 3.4: Segmentos utilizados em *Caminho II.*

A equação 3.3 é novamente usada para calcular a soma ponderada de cada caminho. Em sua saída, é obtido um contorno contínuo com um pixel de largura.

3.2 Extração de vértices

O processo proposto para a extração de vértices de uma imagem divide-se em três etapas:

- Geração dos vértices primitivos e valores de Hough,
- Extração dos vértices de uma cena,
- Filtro máximo local.

De maneira simplificada, a idéia é construir diversas matrizes contendo todos os vértices que se desejam identificar nos objetos da imagem. Estas máscaras, chamadas

"máscaras primitivas de vértices", percorrerão toda a imagem, linha por linha, realizando comparações para verificar a ocorrência de alguma delas no quadro. Cada comparação positiva resultará em um vértice na imagem de saída, de forma que, ao final do processamento, será obtida uma lista dos vértices dos objetos presentes na cena. A composição da imagem poligonal será realizada numa etapa posterior, através da ordenação dos pontos obtidos.

3.2.1 Geração dos vértices primitivos e valores de Hough

O primeiro passo para a criação de um algoritmo gerador de vértices primitivos é determinar o tamanho da máscara a ser utilizada e o número de pixels que cada primitiva possuirá. Estes dois fatores são fundamentais, pois influenciam diretamente na área final do CI e, consequentemente, o seu custo final.

A tabela 3.2 mostra a quantidade de vértices primitivos, deteminados pelo algoritmo implementado, com número de pixels fixo (n pixels) obtidos para máscaras de diversos tamanhos ($n \times n$).

tamanho da máscara ($n \times n$)	**vértices primitivos gerados**
3×3	2
4×4	7
5×5	17
6×6	54
7×7	127
8×8	273
9×9	971

Tabela 3.2: Vértices primitivos gerados em função do tamanho da máscara utilizada.

Os valores apresentados mostram um crescimento muito rápido na quantidade de primitivas geradas, dado um acréscimo de uma unidade nas dimensões da máscara

usada. Dessa forma, é estabelecida uma relação de compromisso, pois quanto maior o número de máscaras, maior será a área necessária para a integração, já que cada primitiva é implementada através de uma série de portas lógicas. Logo, decidiu-se utilizar máscaras 8×8 para a criação dos vértices primitivos. No gráfico apresentado na figura 3.5, os valores obtidos na tabela 3.2 (unidos pelas linhas tracejadas) são plotados junto a função $f(n) = 0.6\exp(0.765n)$, que aproxima os resultados obtidos.

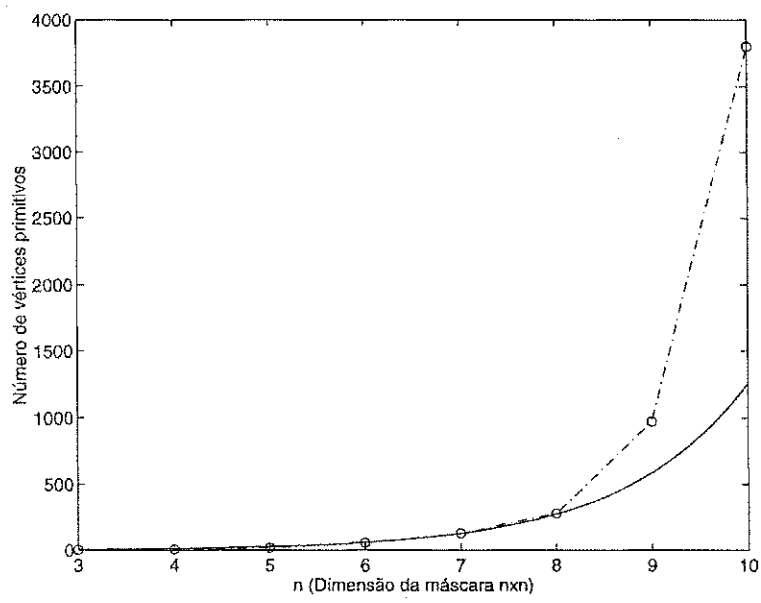


Figura 3.5: Comparação do número de vértices gerados com a função $f(n)$.

Aqui foi observado que a partir de $n = 8$, o número de vértices primitivos gerados apresenta um crescimento mais rápido que a função exponencial $f(n)$.

Da mesma forma, se o número de pixels do segmento de cada primitiva gerada poder variar - por exemplo, produzir vértices com 8, 9 ou 10 pixels - o conjunto de máscaras produzidas seria bastante elevado. Logo, foi utilizado um segmento com tamanho fixo de 8 pixels.

O algoritmo gerador de vértices primitivos é descrito a seguir.

Algoritmo gerador

Inicialmente, tentou-se utilizar o algoritmo de Bresenham [Cun87] para a geração dos vértices primitivos. Esse método consiste basicamente em traçar retas, através do fornecimento de seus pontos inicial e final. No nosso caso, o ponto inicial é o centro da máscara e o final é a sua borda (fim de um segmento que compõe o vértice). Não importa, para o algoritmo, quais pixels intermediários estarão presentes, mas apenas o primeiro e o último. Este algoritmo é bastante utilizado em programas onde necessita-se de elementos gráficos e geométricos como retas, quadrados, triângulos, etc. Um exemplo de um destes softwares que utilizam o algoritmo de Bresenham é o aplicativo PaintBrush do Windows.

Apesar da quantidade expressiva de vértices obtidos, cerca de 90(primitivos) e os resultados, quando testados com algumas imagens, terem sido razoavelmente bons, este método foi abandonado por não ser capaz de gerar todas as possibilidades de vértices desejadas.

A partir daí, decidiu-se desenvolver um método, baseado nos algoritmos de Cheng-Hsu [CH88] e Cortez [Cor95], de forma a obter todos os vértices possíveis numa máscara 8×8.

O método proposto por Cheng-Hsu utiliza o deslocamento ou passo de uma máscara 3×3, para frente e para trás sobre os pontos do contorno a ser modelado. Este deslocamento pode ser feito uma, duas, ou mais vezes, conforme desejado, resultando num ganho considerável de informação sobre a curvatura local, possibilitando, dessa forma, a localização dos vértices do modelo poligonal.

Primeiro dividiu-se a máscara em quatro quadrantes, ficando o ponto central determinado pelo interseção da 5ª linha com a 5ª coluna, como mostrado na figura 3.6. Então, o algoritmo passa a trabalhar basicamente utilizando duas funções recursivas chamadas *Passo1* (passo para a frente) e *Passo2* (passo para trás).

O objetivo destas duas rotinas é o de gerar segmentos de comprimento fixo (4 pixels cada), formando um vértice primitivo com 8 pixels. A partir do programa principal, *Passo1* é chamada e recebe como parâmetros as coordenadas do ponto onde começará o

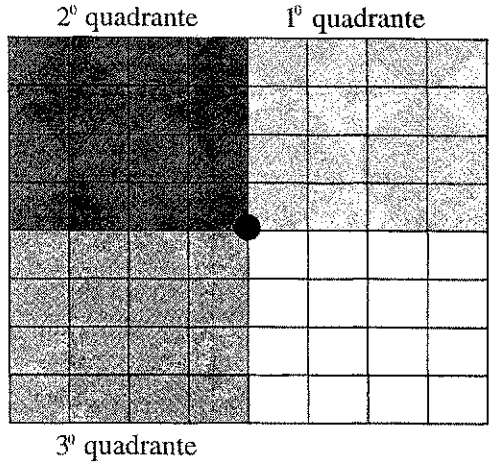


Figura 3.6: Máscara para geração dos vértices primitivos.

segmento que compõe a primitiva (*Passo1* sempre inicia pelo elemento (3,4)) e a direção que o segmento deve seguir. Este procedimento chama a si próprio três vezes e habilita 4 pixels do 1º quadrante por vez. Em seguida, ativa *Passo2* que, da mesma forma, cria todas as combinações possíveis de 4 elementos do 2º e 3º quadrantes. Esgotadas todas as possibilidades, *Passo1* é chamado novamente e o processo repetido até serem finalizados todos os segmentos possíveis no 1º quadrante. Nota-se que a mudança de posição de apenas um pixel em um segmento equivale a gerar outro diferente.

Todos os passos são gerados de tal forma que entre um pixel e outro do mesmo segmento ou entre os dois segmentos não existam ângulos retos [Cor92]. Após cada vértice ser gerado da maneira descrita anteriormente, é ativada uma rotina que faz operações de espelho e rotação, obtendo todas as possíveis "derivadas" desta primitiva. Estas máscaras - 1 primitiva mais 7 derivadas - são comparadas com os vértices gerados anteriormente. Se houver alguma repetição, implica que a máscara gerada não é uma primitiva e será descartada. Caso contrário o vértice gerado é primitivo e será armazenado. Aqui justifica-se o fato de não se usar os pixels do 4 quadrante para criar os segmentos, pois eles são obtidos nas derivações.

Exemplos de primitivas geradas são mostradas na figura 3.7.

Além da eliminação das redundâncias, citada anteriormente, foram retiradas do

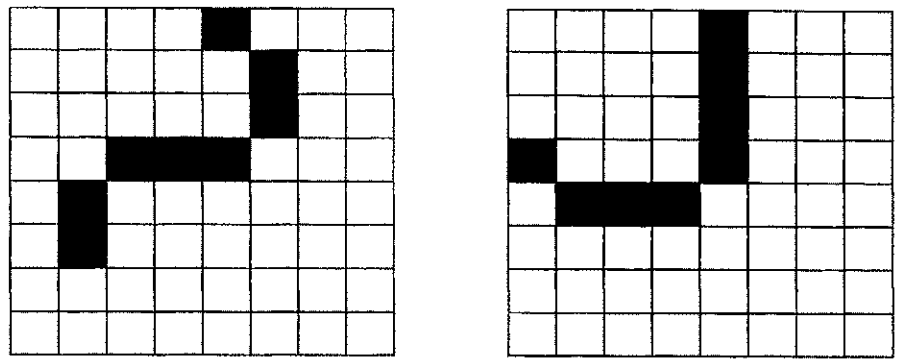


Figura 3.7: Exemplo de vértices primitivos gerados.

grupo das primitivas os vértices representando linhas retas (horizontais, verticais e diagonais) ou formas muito próximas a estas - por exemplo aqueles segmentos que diferem das retas por apenas um pixel. O resultado final apresenta um conjunto com 273 primitivas.

A etapa seguinte consiste no cálculo do valor de Hough para cada vértice gerado.

Transformada de Hough para detecção de linhas retas

A aplicação da transformada de Hough em imagens digitais consiste em mapear pontos de uma cena para um espaço de parâmetros, onde padrões contidos na imagem possam vir a ser evidenciados [Cor95].

A transformada de Hough se baseia na equação da reta no plano cartesiano xy. Considere um pixel $P_i(x_i, y_i)$ pertencendo a um determinado quadro. Infinitas linhas retas passam por este pixel cujas equações têm a forma:

$$y_i = mx_i + c \tag{3.4}$$

onde m é a declividade da reta e c é uma constante.

Para vários valores de m e c. Esta equação pode ser escrita como:

$$c = -x_i m + y_i \tag{3.5}$$

na qual agora, x_i e y_i são considerados constantes e m e c variáveis. Isto significa que o ponto P_i no plano xy é representado por uma reta no plano mc. Desta forma, pontos pertencentes a mesma reta, em xy, geram retas em mc que se interceptam num mesmo ponto. Do ponto de vista da aplicação em contornos, esta propriedade implica que pixels pertencentes a uma mesma reta, mesmo que esta sofra descontinuidade, são reconhecidos ou classificados como tal. Entretanto, isto também pode ocorrer com pixels pertencentes a segmentos retos distintos (pontos pertencentes a objetos isolados), mas que estejam alinhados na imagem, o que pode na prática levar a erros de classificação [MM92, GW87].

O mapeamento de um ponto pertencente ao plano xy para o plano mc nem sempre é possível. Isto ocorre, por exemplo, quando o ponto está contido em retas paralelas ao *eixo-y*, com m, coeficiente angular da reta, assumindo valor infinito. Este problema é resolvido pela parametrização da equação da reta [Cor95].

A equação paramétrica de uma reta r qualquer pode ser expressa da seguinte forma:

$$\rho = x_i \cos(\theta) + y_i sen(\theta) \tag{3.6}$$

onde ρ é o segmento reto formado pela origem e pelo ponto $P_i(x_i, y_i)$ e perpendicular a r neste ponto. A variável θ é o ângulo entre ρ e o *eixo-x* positivo, cujos valores podem variar de 0° a 180° com relação aos valores positivos de x [Cor95].

Qualquer ponto no plano xy pode agora ser representado por uma senóide no plano $\theta\rho$ (espaço de parâmetros). De forma análoga à situação anterior, pontos em xy sobre uma mesma reta geram senóides em $\theta\rho$ (pela equação 3.6) que se interceptam num mesmo ponto [MM92, GW87].

Considerando-se quatro pontos $P_0(0,0)$, $P_1(2,0)$, $P_2(0,2)$ e $P_3(2,2)$. As curvas representando estes pontos, obtidas pela equação 3.6 , estão mostradas na figura 3.8. As linhas retas definidas por dois quaisquer destes pontos, no plano xy, formam as direções indicadas na figura 3.9. Estas direções são dadas pelos valores das abcissas das interseções entre as correspondentes senóides, que representam aqueles pontos, no plano $\theta\rho$. Estas interseções estão marcadas com círculos na figura 3.8 e seus valores expressos em função de π.

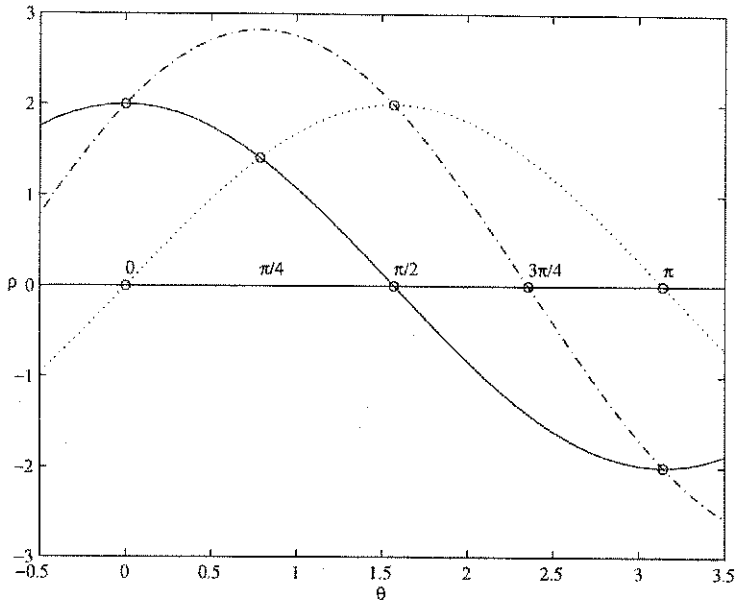


Figura 3.8: Representação dos quatro pontos no espaço de parâmetros.

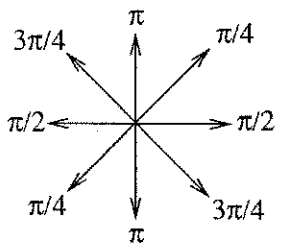


Figura 3.9: Direções das retas geradas pelos quatro pontos.

Tradicionalmente, a transformada de Hough se utiliza, no espaço de parâmetros, de elementos chamados de acumuladores, os quais evidenciam algum padrão (reta, círculo, etc) contido na imagem digital. É neste enfoque que a maioria das investigações, até agora relatadas, estão concentradas.

Cálculo dos valores de Hough para os vértices primitivos

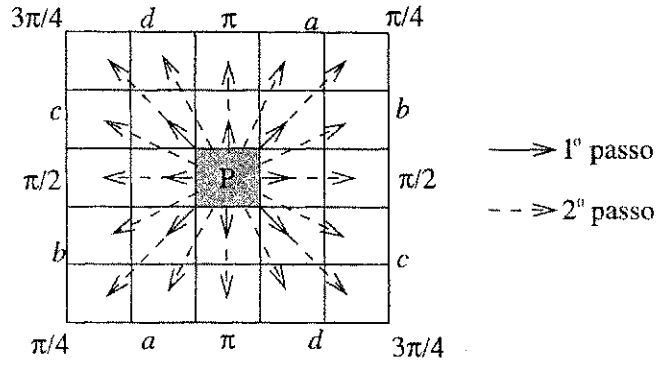


Figura 3.10: Principais direções da transformada de Hough em torno do ponto P.

Considere o ponto central de uma máscara 5×5, P, mostrado na figura 3.10. Em torno dele será calculada a curvatura local. Suas possíveis direções para o *Passo-1*(dadas pelos oito pixels ao redor de P) são π, $\pi/4$, $\pi/2$ e $3\pi/4$. Para o *Passo-2*, além daquelas do passo anterior, as direções podem ser a, b, c e d. Para a máscara 8×8, utilizada pelo algoritmo proposto (seção 3.2.1), todas as direções foram previamente calculadas e mostradas, em graus, na figura 3.11.

Portanto, seja uma sequência $S = \{P_p, P_{1+p}, \ldots, P_{7+p}\}$ de 8 pontos, representando um vértice primitivo gerado e sejam os pontos P_{i-p}, P_i e P_{i+p}, cujos parâmetros $\theta\rho$ são $(\theta_{i-p}, \rho_{i-p})$, (θ_i, ρ_i) e $(\theta_{i+p}, \rho_{i+p})$, respectivamente. A variável p representa a quantidade de passos dados, a partir do centro da máscara, P_i (interseção da 5^a linha com a 5^a coluna), de modo a chegar ao ponto $P_{i\pm p}$. Os pontos P_{i-p} e P_{i+p} estão associados aos p passos para trás e para frente, a partir de P_i, respectivamente. A diferença de direção D_p, para o passo p é definida por [CH88]:

$$D_p = |If_p - Ib_p| \qquad p = 1, 2, 3, 4 \tag{3.7}$$

135	145	157	172	9	23	36	45
126	135	150	169	11	32	45	54
113	120	135	162	19	45	60	66
99	101	109	135	45	72	80	82
82	80	72	45	135	109	101	99
66	60	45	19	162	135	120	113
54	45	32	11	169	150	135	126
45	36	23	9	172	157	145	135

Figura 3.11: Valores de Hough para a máscara 8×8.

onde, If_p é o ângulo correspondente ao ponto de interseção entre as senóides que mapeiam os pontos P_i e P_{i+p} em $\theta\rho$ e Ib_p entre as senóides que mapeiam os pontos P_i e P_{i-p} no mesmo plano. Os valores D_p, portanto, representam a variação absoluta de direção entre os segmentos $\overline{P_{i-p}P_i}$ e $\overline{P_iP_{i+p}}$ no plano xy.

Após o cálculo das diferenças de direção de cada passo, para frente e para trás, a variação total é calculada por:

$$D_t = \sum_{i=1}^{4} D_i \tag{3.8}$$

onde observa-se que, quanto maior o valor de D_t obtido, maior a variação angular do segmento gerado e, consequentemente, mais significativo será o vértice.

Os valores de Hough de cada vértice primitivo gerado são então armazenados em uma memória RAM ou EPROM e serão utilizados pela etapa de extração de vértices da imagem.

3.2.2 Extração dos vértices de uma cena

O extrator de vértices recebe continuamente os pixels do contorno do objeto processados. A cada pixel de entrada, o que corresponde a inserção de uma nova janela 8×8, o extrator realiza inicialmente operações de rotação, transposição e "espelho", de forma

a obter todas as derivações possíveis da nova janela da imagem de entrada. Estas operações, nesta etapa, são realizadas através de multiplexação e roteamento da janela dentro do circuito integrado, sem a necessidade de operadores especiais. De posse da janela de entrada e de todas as suas derivações, o extrator de vértices realiza comparações destas com os vértices primitivos. Estas comparações são implementadas via lógica combinacional, de forma que todo o conjunto de primitivas pode ser verificado simultaneamente, com grande velocidade. Caso seja determinada alguma igualdade, a saída será o endereço da memória EPROM que contém o número de Hough da primitiva encontrada, caso contrário, será o endereço da memória que contém o valor zero.

3.2.3 Filtro máximo local

A última etapa da extração de vértices é implementada pelo filtro máximo local. A entrada desta etapa é dada pelos valores de Hough provenientes da memória EPROM.

Este filtro trabalha com dois critérios de decisão. No primeiro, se o número de Hough correspondente ao pixel central de uma janela 5×5 for o maior valor da janela, ele será submetido ao segundo critério, caso contrário a saída torna-se zero. No segundo, o valor de Hough é comparado a um limiar de decisão (na forma de um parâmetro ajustável via software), se for maior ou igual, a saída do sistema será 1, caso contrário, 0.

Portanto, na saída do *máximo local*, teremos um bit que será acrescentado ao endereço fornecido pela etapa anterior, de forma a validá-lo (caso seja 1) ou não (caso seja 0).

No próximo capítulo, apresentaremos a implementação em hardware para o algoritmo proposto para extração de vértices de uma imagem.

Capítulo 4

Implementação em hardware

Neste capítulo encontram-se todos os passos envolvidos no desenvolvimento, síntese e verificação da implementação em hardware para a extração de vértices de imagens digitais. Aqui serão descritas a metodologia utilizada no projeto, as arquiteturas do sistema e do ASIC, a implementação das funções determinadas pelo algoritmo proposto e os resultados obtidos ao final do processo.

4.1 Metodologia de projeto

Várias são as etapas de concepção de um circuito integrado, que podem ser mais ou menos complexas, em função das especificações fornecidas. A seguir são mostrados os passos adotados no projeto.

Especificação do sistema

Foram determinadas as especificações do sistema quanto às entradas, funções a serem realizadas e saídas requeridas, sem detalhes quanto ao seu funcionamento interno. Neste ponto, todo o sistema é visto como uma caixa preta.

Algoritmo e simulação do sistema

Teve como finalidade descrever o sistema, sem levar em consideração a implementação, e descobrir possíveis problemas de desempenho e erros de funcionamento do algoritmo. Todos os algoritmos propostos foram simulados utilizando a linguagem de programação C.

Partição hardware/software

Aqui é tomada a decisão sobre quais partes do sistema serão implementadas em hardware e quais as funcionalidades que serão implementadas em software. Além disso, determinou-se também o número de pinos que seriam usados por cada uma das entradas e saídas do ASIC, o que dependeu de fatores como o tipo dos dados (síncronos ou assíncronos), suas formas de entrada no CI (paralela ou serial) e a utilização ou não de multiplexação. Esta etapa foi bastante facilitada, como será mostrado, pela flexibilidade do projeto.

Escrita da descrição em linguagem para hardware

O comportamento e/ou estrutura de um sistema pode ser determinado em uma linguagem de descrição para hardware (HDL). Estas linguagens diferem das de alto nível usadas em softwares pela inclusão de conceitos como paralelismos e simulação de atrasos de sinais e eventos e são responsáveis, no projeto, pela definição dos seguintes atributos:

- hierarquia
- operações sequenciais e paralelas
- comprimento das palavras e vetores de bits
- especificação e alocação de registradores
- operações lógicas, aritméticas e de comparação

- fluxo de controle

Portanto, tendo visualizado todo o componente, pensou-se então numa arquitetura que realizasse a função desejada, levando-se em conta a implementação. Várias linguagens diferentes poderiam ter sido adotadas como VHDL, ELLA, Verilog, etc, porém, a escolha recaiu sobre Verilog devido ao seu fácil entendimento e agilidade no desenvolvimento de projetos, além do alto grau de integração no aplicativo *Cadence* [INC94].

Adicionalmente, também foi usada a metodologia de geradores de descrição sintetizável. Um gerador é um programa capaz de gerar uma descrição de uma função predeterminada e parametrizável, como por exemplo geradores de RAM, de multiplicadores, de FIFOs, etc. Existem geradores que trabalham a nível de *layout* [WE92] e outros que produzem listas de portas lógicas e suas conexões. No nosso caso, a metodologia foi empregada para gerar código Verilog sintetizável, a partir das rotinas de simulação de sistemas em linguagem C (seção 4.3).

Escrita do Testbench

Aqui foram gerados conjuntos de vetores de simulação, utilizando imagens de teste de uma biblioteca, para o algoritmo, cujas respostas ao sistema eram previamente conhecidas, através da simulação do sistema em C. Seu objetivo foi, então, o de detectar possíveis falhas de funcionamento através da observação das saídas produzidas.

Simulação do componente

A simulação foi realizada injetando-se as entradas do *testbench* no algoritmo e analisando-se as saídas geradas.

Síntese

A síntese corresponde a etapa onde a descrição proposta é implementada a nível de lista de portas lógicas. Portanto, o sintetizador compilou a descrição comportamental, selecionando registros auxiliares utilizados para implementar a função necessária,

otimizando dois parâmetros muito importantes, o *custo* e o *timing*. Como já mencionado, o custo é diretamente proporcional a área ocupada pelo circuito na pastilha. O *timing* está relacionado ao tempo total gasto pelo hardware para gerar as suas saídas, independentemente dos vetores de entrada aplicados. Este tempo é determinado através do cálculo de todos os *caminhos críticos*, ou seja, aqueles caminhos mais longos e que, por isso mesmo, causam um maior atraso no circuito. Após realizadas tais optimizações, o sintetizador gera um novo código Verilog que usa células padrão de uma biblioteca (somadores, registros, portas NAND, etc.).

Simulação pós-síntese

Como feito para a descrição comportamental, o novo código gerado pela síntese também foi simulado, usando-se o mesmo *testbench*. Neste ponto, não interessam apenas se as saídas geradas são corretas ou não, mas também os tempos de atraso do circuito.

Síntese de Layout

Nesta etapa foram criadas as máscaras do *layout* do esquemático (circuito) obtido através do sintetizador. Todas as células de portas lógicas e registradores são organizadas e roteadas (interligadas) e, também, são inseridos os PADs, que são contatos metálicos que interligam o circuito interno aos pinos de entrada e saída do encapsulamento.

Simulação pós-layout

Mais uma vez, os resultados foram simulados para a observação das saídas e análise de *timing*, porém, além disso, foi realizado um confronto entre o *layout* e o esquemático que o gerou, de forma a garantir que se tratavam do mesmo circuito.

Vale a pena citar que, durante as etapas de descrição do algoritmo, utilização dos

geradores de descrição sintetizável, síntese lógica e geração do *layout*, foi usada a abordagem *Top-down*. Nesta técnica, partimos de uma descrição de alto nível que, através de uma série de métodos de refinamento, faz a adaptação à tecnologia empregada, resultando em uma descrição em baixo nível, que pode ser dada na forma de FPGAs, células programáveis, *standard cells* ou *full custom* (a nível de transistores). No projeto, a síntese estrutural foi implementada em células padrão, utilizando a tecnologia ECPD 07 - transistores CMOS com $0.7\mu m$ de comprimento mínimo do canal.

Nas etapas de simulação e análise de *timing*, a abordagem inversa, *Bottom-up*, foi empregada.

4.2 Arquiteturas sistêmicas

4.2.1 Arquitetura do sistema dedicado

Um diagrama do sistema com ASIC é mostrado na figura 4.1.

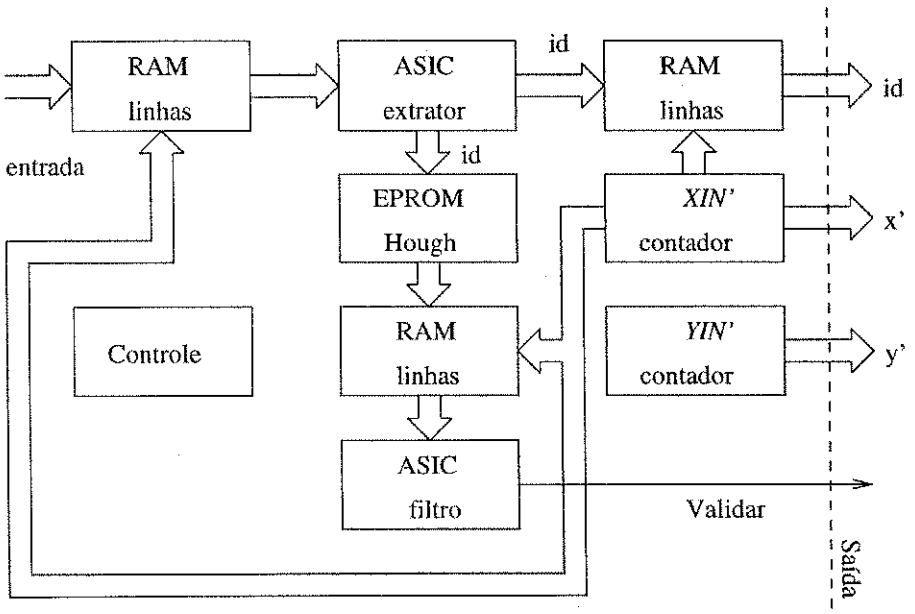


Figura 4.1: Diagrama do sistema com ASIC.

Cada ASIC é conectado a uma memória RAM de linhas, controlada pelo contador *XIN'*. O ASIC que realiza a extração recebe os dados referentes ao contorno processado e fornece a identificação dos vértices primitivos, contidos na cena, para a memória EPROM, que armazena os valores de Hough de cada primitiva, e para a memória de linha da saída. O ASIC que realiza a filtragem recebe os valores de Hough de sua memória de linha e decide se o valor central de uma janela 5×5 é, ou não, um vértice do modelo poligonal. A saída do filtro é dada por um bit que validará, ou não, o número de identificação final, formado pelos valores dos contadores *XIN'* e *YIN'* e a identificação fornecida pelo ASIC extrator.

A pinagem do ASIC é mostrada na figura 4.2.

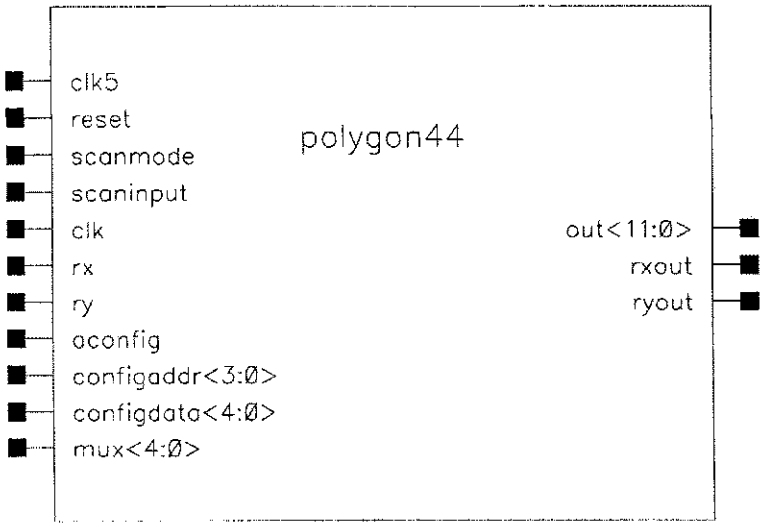


Figura 4.2: Pinagem do ASIC.

Cada um dos pinos é descrito da seguinte forma:

Entradas:

- clk5: entrada do relógio do sistema para todos os registros;
- reset: reset do sistema para todos os registros, ativo em nível baixo;

- scanmode: modo de deslocamento no *scanpath* para todos os registros;
- clk: entrada do relógio de pixel;
- aconfig: entrada que ativa o modo de configuração;
- rx: entrada de sincronismo de linhas;
- scaninput: entrada no *scanpath*;
- ry: entrada de sincronismo de *frame*;
- configaddr(3:0): entrada que fornece o endereço de acesso aos registradores de parâmetros;
- configdata(4:0): entrada que fornece os dados para os registradores de parâmetros;
- mux(4:0): entrada serial dos pixels da imagem.

Saídas:

- rxout: sinal de sincronismo de linha para o próximo estágio. Quando *scanmode*=1, é a saida do *scanpath*;
- ryout: sinal de sincronismo de *frame* para o próximo estágio;
- out(11:0): saída dos pixels processados.

Os valores entre " ( )" correspondem a dimensão dos vetores de bits. O *scanpath* é um caminho formado pela interligação em série de todos os registradores do ASIC, onde se pode injetar dados para testes.

4.2.2 Arquitetura de um sistema programável

Um digrama do sistema proposto para a emulação do circuito é mostrado na figura 4.3.

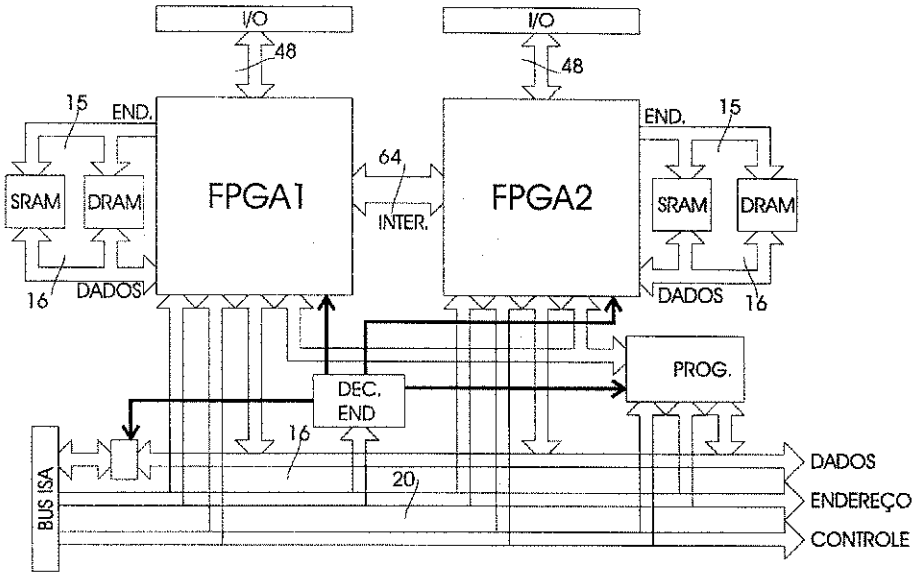


Figura 4.3: Arquitetura da placa utilizando FPGAs.

A arquitetura proposta tem por objetivo permitir a emulação dos algoritmos implementados. São utilizadas memórias RAM para armazenar as linhas da imagem, conectores para inserir as imagens digitalizadas, por meio de uma câmera *Quick Cam*, um barramento ISA, suficiente para transmitir a lista de vértices na taxa de operação empregada e duas FPGAs, sendo uma para realizar a extração de contornos e outra para obter os vértices que serão usados na modelagem poligonal.

A arquitetura interna ao CI, para a extração de vértices é mostrada na figura 4.4 e cada uma das suas funções é tratada na próxima seção.

4.3 Funções implementadas

Para a tarefa de extração dos vértices de uma imagem, foram implementadas as seguintes funções:

- *Gera_jan8*
- *Primit*

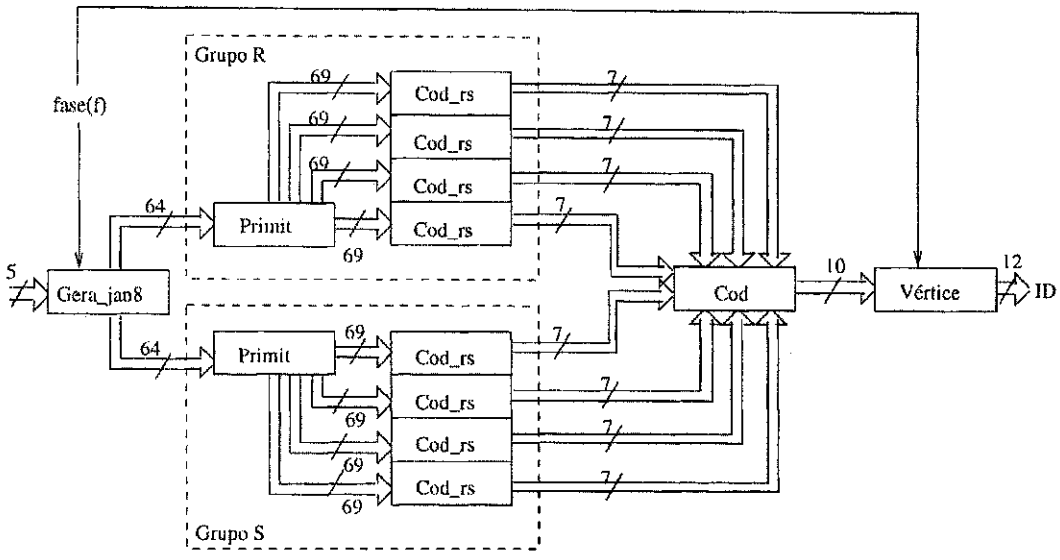


Figura 4.4: Arquitetura do CI para extração dos vértices.

- *Cod_rs*
- *Cod*
- *Vértice*

A seguir, cada uma delas é descrita.

Gera_jan8

Esta função é responsável pela organização dos dados de entrada em forma de janelas 8×8, pelo apagamento das células da máscara que estiverem fora dos limites da imagem e pelas operações de rotação e "espelho" da matriz de entrada para geração de derivadas (ver seção 3.2.1).

Os pixels processados do contorno, provenientes de uma memória RAM, são inseridos em um conjunto de registradores dispostos como mostrado na figura 4.5.

A memória RAM é capaz de armazenar sete linhas da imagem do contorno, com 1024 pixels por linha e 1 bit por pixel (contorno binário). O registrador $Col0$, tem a

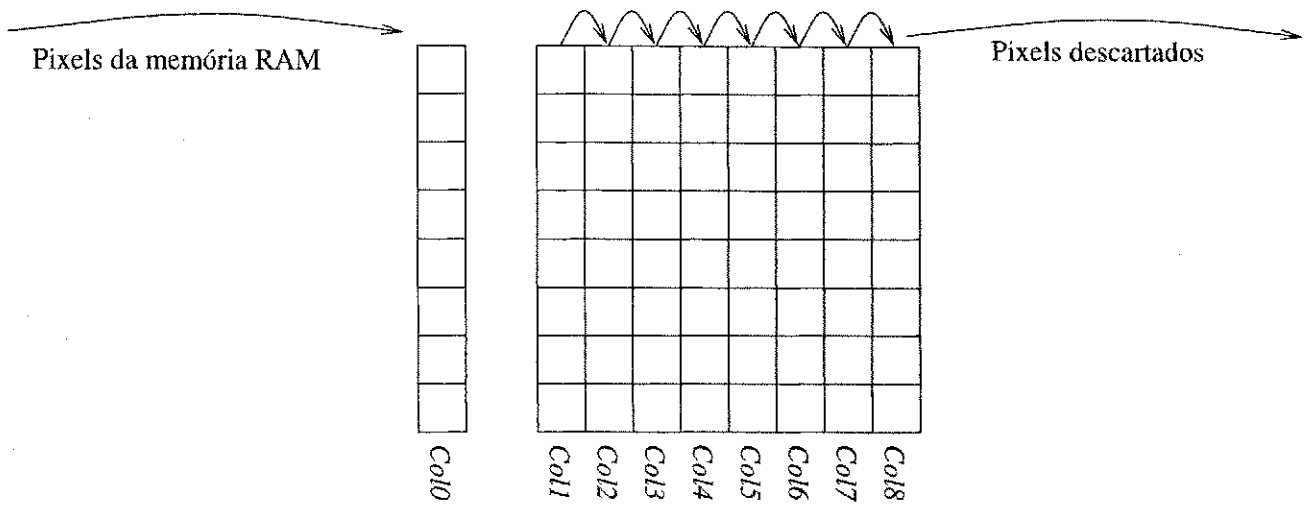


Figura 4.5: Formação da janela 8×8 no CI.

função de acumulador temporário para os dados da memória, enquanto que os demais, $Col1, \ldots, Col8$, formam a janela para as operações. O deslocamento do quadro é dado pela transferência do conteúdo dos registradores no sentido indicado na figura.

Internamente ao módulo, são implementados contadores (*XIN* e *YIN*) que indicam em que lugar da imagem está o centro da janela e cujos valores servem para indicar se algum pixel da janela 8×8 se encontra sobre a zona de apagamento, fora da imagem e, portanto, deve ter seu valor igual a 0. Um exemplo é mostrado na figura 4.6.

Uma amostra dos comandos em Verilog empregados para o apagamento é apresentado a seguir:

```
wire l15=(xin==nc-1||yin==0||yin==1||yin==2)?0:col5[7];

wire l25=(xin==nc-1||yin==0||yin==1)?0:col5[6];
```

Aqui, os valores dos contadores são comparados e, dependendo dos resultados, o valor dos pixels analisados será anulado ou não. As linhas de comando mostradas referem-se aos elementos da primeira linha e quinta coluna e segunda linha e quinta coluna. Os valores de *XIN* e *YIN* variam de 0 a $nc-1$ e de 0 a $nl-1$, respectivamente, onde nc é o número de colunas da imagem e nl o número de linhas. Estas dimensões da imagem são fornecidas durante a inicialização do CI, na forma de parâmetros.

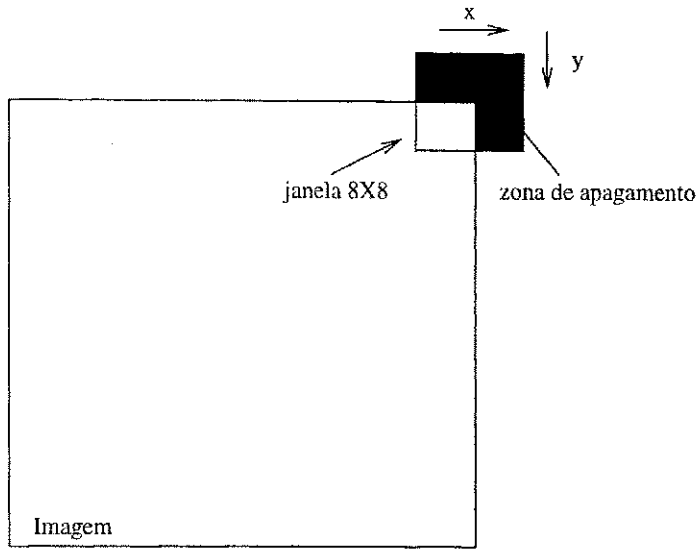


Figura 4.6: Zona de apagamento da janela 8×8.

Como mencionado no capítulo 3, cada vértice primitivo possui 7 derivados. Um exemplo para uma imagem na forma da letra "P" é mostrado na figura 4.7.

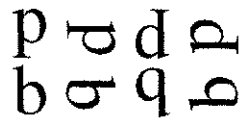


Figura 4.7: Possíveis derivadas de uma imagem na forma da letra "P".

Como cada primitiva deve ser construída utilizando lógica combinacional, é mais econômico realizar operações de rotação e "espelho" da janela de dados de entrada que implementar todas as derivadas no CI, na forma de portas lógicas. Dessa maneira, a tarefa torna-se apenas um problema de multiplexação. Alguns comandos para estas tarefas são apresentados a seguir:

```
wire r11=(f==3)?l18:(f==1)?l81:(f==2)?l88:(f==0)?l11:l11;

wire r12=(f==3)?l17:(f==1)?l82:(f==2)?l87:(f==0)?l21:l12;
```

No exemplo apresentado, os pixels *r11* e *r12* têm seus valores modificados de acordo com o sinal de controle *f* (fase), produzindo assim, a cada instante de tempo, uma das

derivações necessárias.

A fase seleciona qual rotação cada grupo está realizando, de forma que, a cada instante, temos na saída do *Gera_jan8* duas janelas 8×8. Portanto, as oito derivadas são produzidas para dois grupos que realizam quatro operações cada, chamados R e S.

A tarefa de geração de derivadas poderia ser implementada por oito grupos que realizassem apenas uma operação cada. Isto seria mais vantajoso em relação ao *timing*, porém, como veremos a seguir, seria utilizada uma área muito maior para a integração, pois cada máscara entregue necessita de um módulo *Primit* e quatro módulos *Cod_rs*.

Primit

Nesta etapa do projeto foi empregada a técnica de geração de descrição sintetizável. Foi criado um programa em C que, a partir de um arquivo contendo todos os vértices primitivos gerados na forma de matrizes de 0's e 1's, produzisse um código Verilog capaz de implementá-los. Com isto foi conseguida uma imunidade a possíveis erros na escrita da descrição Verilog e também uma maior agilidade no projeto. Uma parte do código gerado é mostrado a seguir, onde cada linha corresponde a uma primitiva:

```
r15 && r26 && r36 && r43 && r44 && r45 && r52 && r61,
r15 && r26 && r36 && r43 && r44 && r45 && r51 && r52,
r15 && r26 && r36 && r42 && r43 && r44 && r45 && r51,
```

Este módulo é um bloco combinacional que recebe uma janela 8×8 na entrada e coloca na saída um vetor com 276 posições. Estas posições são formadas pelas 273 primitivas mais o vetor $(0, 0, 0)$. A razão da inclusão destes elementos se tornará aparente no próximo bloco. Desta forma, se alguma máscara contendo um vértice primitivo for detectada em *Primit*, a posição correspondente no vetor será 1 - o resultado das operações AND na linha de código. São instanciados dois blocos *Primit*, um para o grupo S e outro para o grupo R.

Cod_rs

A idéia inicial para este módulo era o de criar um codificador de 273 para 9 bits, de modo a fornecer para cada grupo (R e S) um número de identificação preliminar. Porém, ocorreram problemas no processo de síntese deste bloco, devido a limitações do sintetizador - ou foram geradas áreas grandes demais ou criadas situações onde o tempo de síntese era excessivo - e decidiu-se pela divisão do bloco, inicialmente em duas partes e, depois, em quatro partes. Assim, ao conjunto inicial de 273 bits foram acrescentados mais três nulos, para se trabalhar com quatro codificadores iguais de 69 para 7 bits.Um total de 8 módulos *Cod_rs* são instanciados, 4 para o grupo S e 4 para o R. No caso de vários acertos simultâneos no vetor, o primeiro acerto detectado é codificado.

Cod

Aqui é realizada uma nova codificação, dessa vez de 8 para 3, das saídas dos blocos *Cod_rs*. Dessa forma, é atribuído um novo valor de identificação com 10 bits.

Vértice

A etapa final do processo consiste em verificar se existe um valor de identificação válido, gerado pelos blocos anteriores (valor não nulo). Se confirmado, aos bits de Cod são acrescentados os da fase (sinal f) que os gerou, produzindo um número de identificação final de 12 bits, caso contrário, a saída é nula. Esse número será o endereço de uma memória EPROM externa ao ASIC, que conterá todos os valores de Hough (seção 3.2.1). No próximo capítulo serão apresentados os resultados obtidos pela simulação

do algoritmo proposto e pelo processo de síntese do hardware projetado.

Capítulo 5

Resultados

Neste capítulo são apresentados os resultados obtidos pela simulação do algoritmo proposto e sua implementação em hardware.

5.1 Simulações do algoritmo proposto

Os resultados aqui apresentados foram obtidos a partir de imagens reais com 32 níveis de cinza. Foram selecionadas duas imagens - *Arco* e *Chave de Fenda* - para a extração tanto dos contornos como dos vértices. Os resultados obtidos são comparados aos produzidos pelos métodos de modelagem poligonal desenvolvidos por Cortez [Cor95], Ramer [Ram72] e o método Split-Merge.

Outros dois contornos - *Chave* e *Garotos* - foram simulados apenas para a extração de vértices e os resultados obtidos são comparados aos produzidos pelo método de Young [YSN97], que utiliza redes neurais de Hopfield para a tarefa de reconhecimento de objetos (imagem *Chave*), e pelo método de codificação de formas baseado em vértices, desenvolvido por O 'Connell [O'C97] (imagem *Garotos*).

Para efeito de comparação de desempenho dos diversos métodos citados, foram considerados o número de lados do modelo produzido (l), o tempo de processamento (t), a taxa de compactação obtida (τ), dada pelo quociente do número de pixels do

contorno sobre o número de lados l, e o erro médio quadrático (ϵ), dado em pixels, da aproximação, calculado a partir da distância de cada ponto a_i da borda do objeto ao correspondente segmento de reta do modelo poligonal.

As imagens originais e simuladas para a imagem *Arco* são mostradas na figura 5.1.

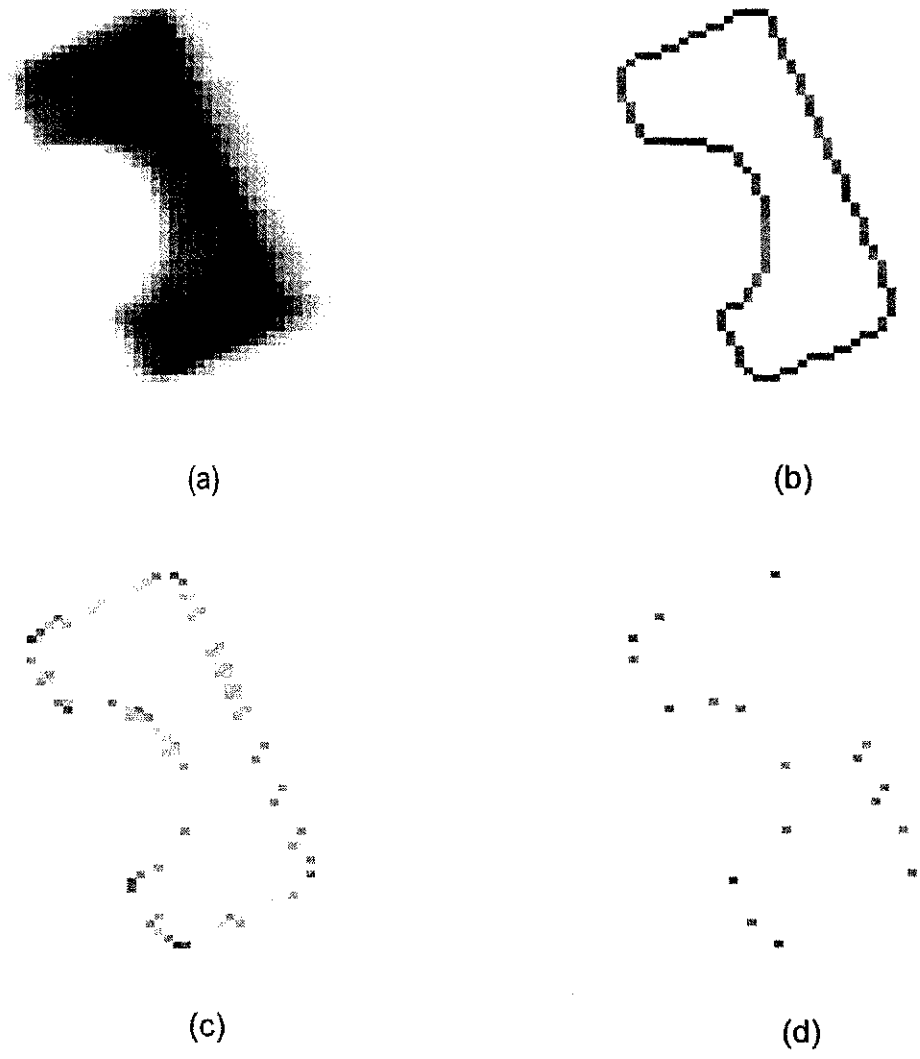


Figura 5.1: (a)Imagem original; (b)contorno; (c)vértices; (d)vértices filtrados.

A tabela 5.1 compara os resultados obtidos pelo método proposto com os algoritmos

de Cortez, Ramer e o método Split-Merge. Na comparação, deve-se observar que o método proposto só fornece os vértices, não os lados da aproximação poligonal.

Método	t	l	τ	ϵ
Cortez	115 ms	11	11.45	0.66
Ramer	115 ms	12	10.5	0.4
Split-Merge	135 ms	12	10.5	0.4
Proposto	31.2 μs	18	7	0.28

Tabela 5.1: Resultados para a imagem *Arco*.

As imagens originais e simuladas para a imagem *Chave de Fenda* são mostradas nas figuras 5.2 e 5.3.

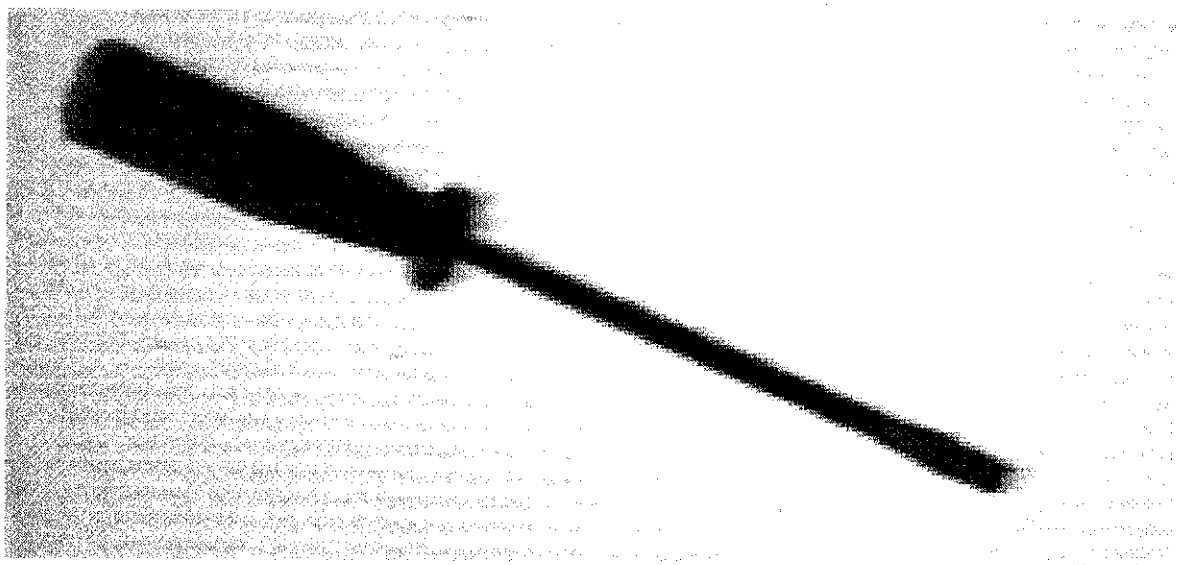
Na tabela 5.2 são apresentados os resultados produzidos pelos mesmos algoritmos usados em *Arco*.

Método	t	l	τ	ϵ
Cortez	292 ms	20	19.35	0.46
Ramer	270 ms	18	21.5	0.56
Split-Merge	305 ms	14	27.64	0.55
Proposto	212.2 μs	34	11.47	0.39

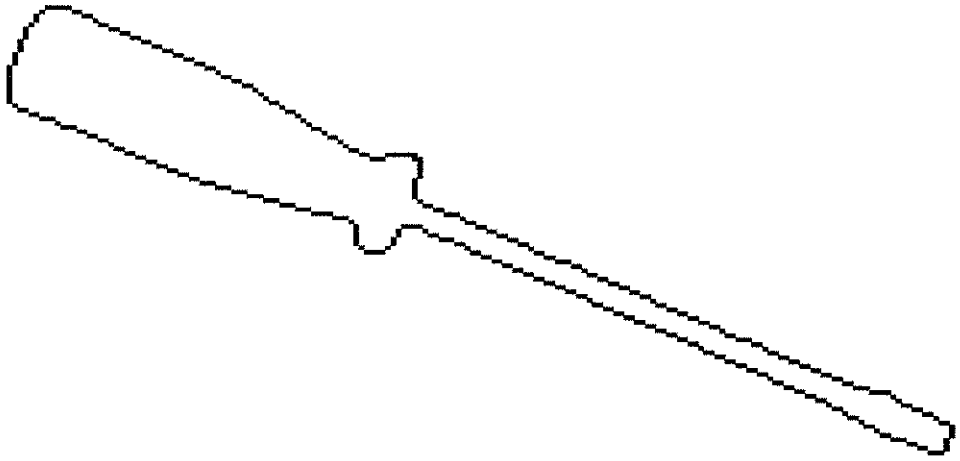
Tabela 5.2: Resultados para a imagem *Chave de Fenda*.

Os contornos das imagens *Chave* e *Garotos* e o conjunto de vértices extraídos são mostrados nas figuras 5.4, 5.5 e 5.6 respectivamente.

A tabelas 5.3 e 5.4 mostram o desempenho dos métodos de Young e O 'Connell comparados ao proposto, com relação ao número de lados produzidos e taxa de compactação.



(a)



(b)

Figura 5.2: (a)Imagem original; (b)contorno.

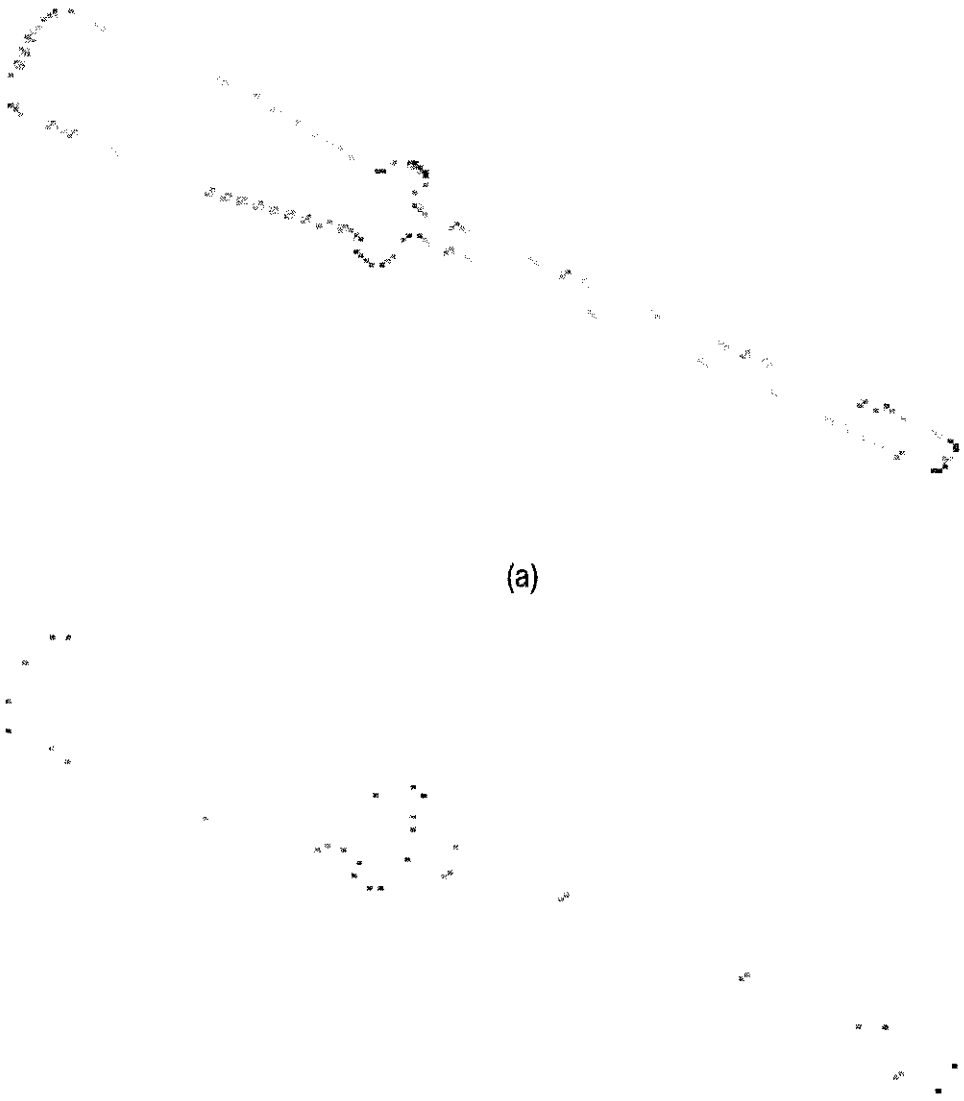


Figura 5.3: (a)Vértices; (b)vértices filtrados.

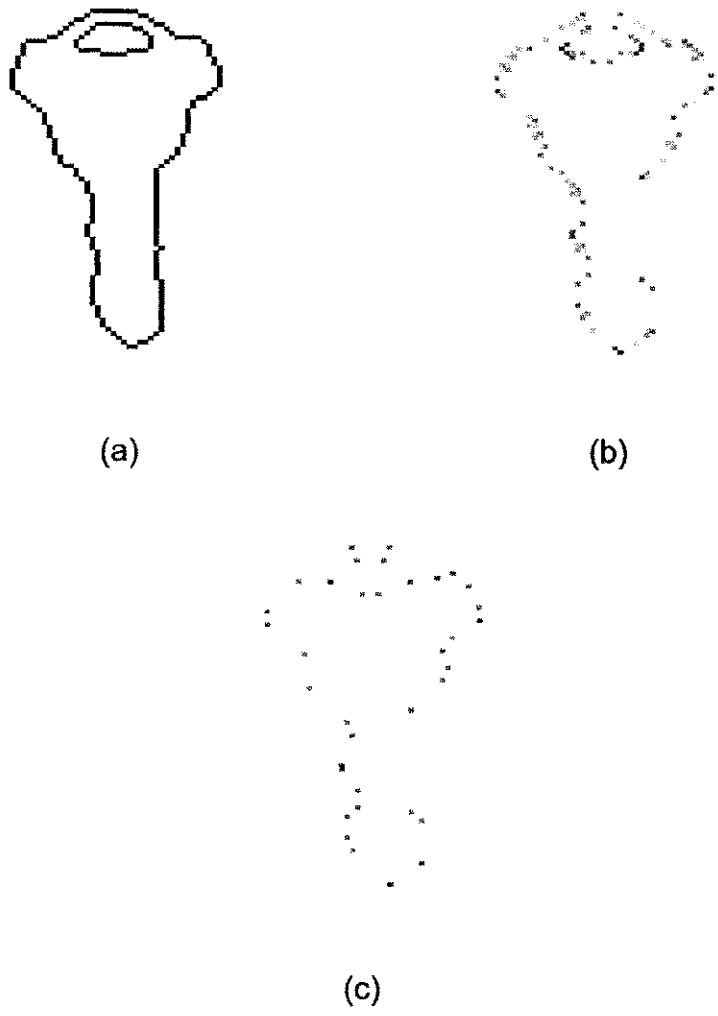
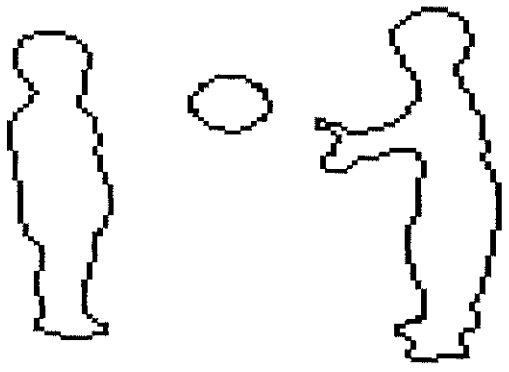


Figura 5.4: *Chave*: (a)contorno; (b)vértices; (c)vértices filtrados.

Método	l	τ
Young	10	17.9
Proposto	36	5.83

Tabela 5.3: Resultados para a imagem *Chave*.



(a)

(b)

Figura 5.5: *Garotos*: (a)contorno; (b)vértices.

Método	l	τ
O 'Connell	82	5.2
Proposto	86	4.95

Tabela 5.4: Resultados para a imagem *Garotos*.

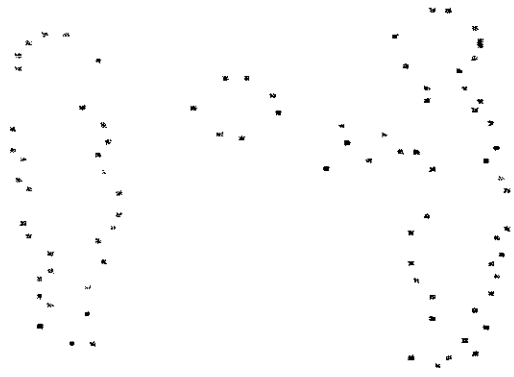


Figura 5.6: Vértices filtrados da imagem *Garotos.*

5.2 Hardware implementado

Os resultados obtidos no processo de síntese quanto ao números de transistores e células padrão, para cada um dos módulos descritos anteriormente, são mostrados na tabela 5.5.

módulo	**células padrão**	**transistores**
Gera_jan8	918	8568
Primit	427	1696
Cod_rs	323	1550
Cod	74	470
Vértice	82	1246

Tabela 5.5: Transistores e células padrão utilizados na síntese.

Com relação ao *timing* e a *área* utilizados, os valores são mostrados na tabela 5.6. A frequência máxima de operação de cada bloco é dada pelo inverso do maior valor de *delay* obtido. Um fato importante nestes resultados é que, como o relógio mestre do CI pode ter uma frequência de operação de até $41.67MHz$ (período de $24ns$), o

módulo	**maior delay** (ns)	**área sem roteamento**(mm^2)
Gera_jan8	10.57	0.62
Primit	2.44	0.16
Cod_rs	3.2	0.14
Cod	4.71	0.04
Vértice	1.79	0.08

Tabela 5.6: Área e *timing* obtidos na síntese.

conjunto dos três módulos combinacionais, *Primit*, *Cod_rs* e *Cod*, são implementados sem a necessidade de *pipeline* entre eles.

Levando-se em conta todo o processo de extração de vértices, com 2 blocos *Primit* e 8 blocos *Cod-rs*, os valores totais são mostrados na tabela 5.7.

	células padrão	**transistores**	**delay** (ns)	**área sem roteamento** (mm^2)
Total	4512	26076	22.71	2.18

Tabela 5.7: Valores totais obtidos.

A área total mostrada refere-se ao espaço da pastilha ocupada apenas pelos transistores. Incluindo os espaços utilizados para o roteamento das células e para a inserção dos PADs, a área necessária para a integração é de $12.09mm^2$ ($3.1mm \times 3.9mm$). O *layout* gerado é mostrado na figura 5.7.

É importante citar que o projeto foi desenvolvido de maneira bastante flexível, podendo ser adaptado facilmente a diferentes configurações como: multiplexação ou não dos dados de entrada e saída, implementação *on-chip* da memória RAM de linha e integração em conjunto com a extração de contornos no mesmo CI, podendo usar encapsulamentos de 28 a 82 pinos. A escolha de uma das configurações é feita levando-se em conta as disponibilidades econômicas para o projeto.

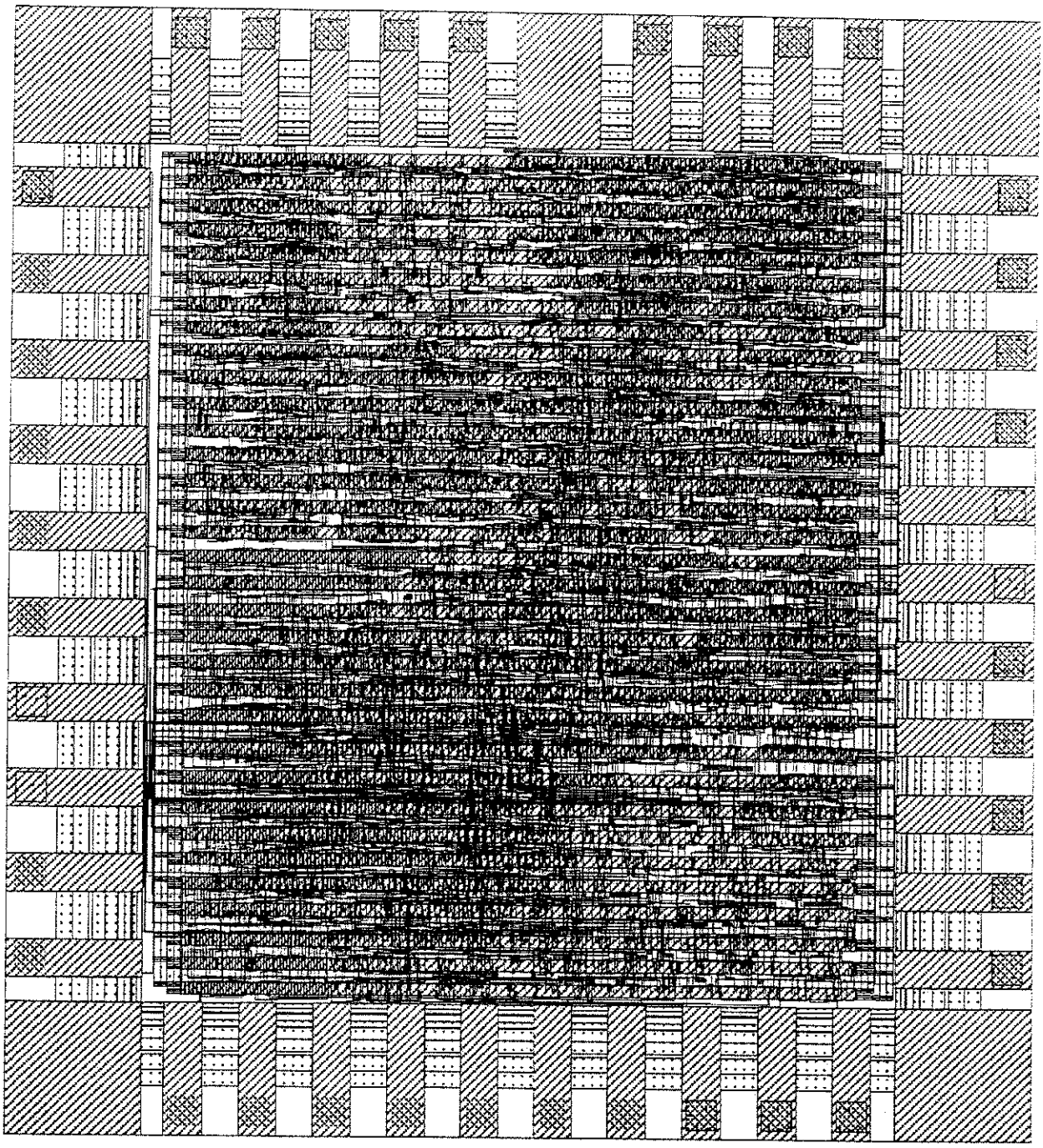


Figura 5.7: *Layout* para extração de vértices.

Um *layout* para a integração conjunta de vértices e contornos também foi gerado e é mostrado na figura 5.8. A área necessária é de $43.33mm^2$.

Para o caso da implementação interna da memória de linha, é usada uma RAM de 7×1024 bits para a extração de vértices, ocupando uma área adicional de $2.2mm^2$. Para a extração conjunta, a RAM necessária é de 25×1024 bits e área extra de $6.2mm^2$.

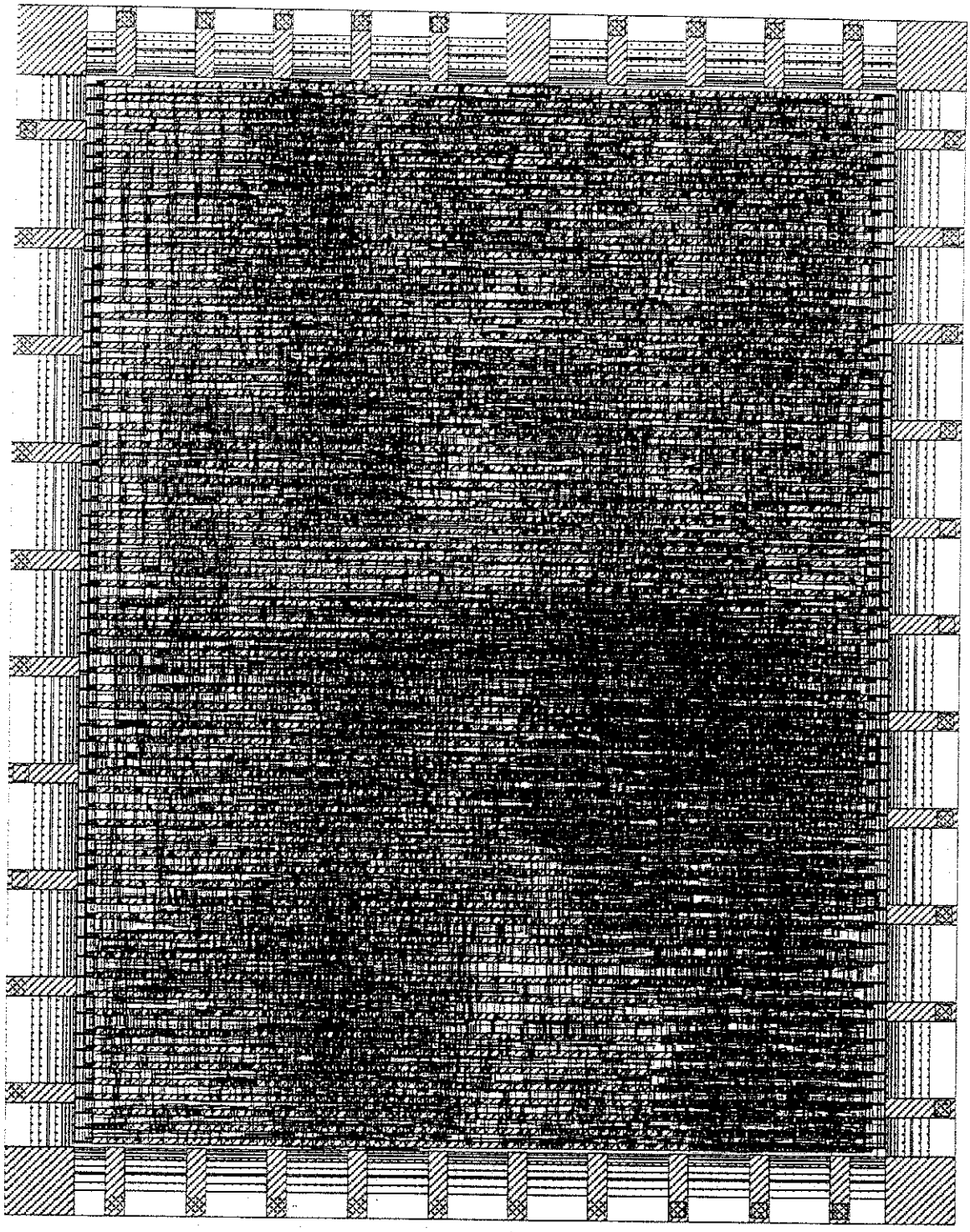


Figura 5.8: *Layout* para extração de contornos e vértices.

Capítulo 6

Conclusões e trabalhos futuros

Pelos resultados simulados, através do algoritmo proposto, podemos perceber que os modelos poligonais produzidos, quando comparado aos demais métodos apresentados, resultam em modelos com maior número de lados e, consequentemente, uma menor taxa de compactação. Por outro lado, os erros médios quadráticos são menores que os produzido pelos outros métodos de modelagem poligonal, o que significa que os modelos poligonais produzidos aproximam melhor as curvas testadas. Uma etapa de processamento posterior poderia eliminar alguns dos vértices gerados para realizar uma compactação maior dos resultados, ao custo de um aumento do erro.

O principal ponto a favor do método implementado são as velocidades de processamento obtidas. O circuito sintetizado foi simulados com o relógio mestre operando a uma frequência de aproximadamente $40MHz$, valor suficiente para garantir o funcionamento do sistema mesmo com imagens de 512×512 pixels, inseridas a uma taxa de 30 quadros por segundo.

Citamos os seguintes trabalhos que podem ser desenvolvidos futuramente.

- A realização de um estudo aprofundado para verificar se existem máscaras primitivas geradas que correspondam a vértices não significativos. Com isto será possível obter-se modelos poligonais ainda melhores.

- O desenvolvimento de um software capaz de ordenar os pontos fornecidos pelo circuito integrado e, com isto, fornecer o modelo poligonal definitivo dos objetos presentes em uma cena.

- O desenvolvimento de um sistema de compressão de vídeo, a partir dos modelos poligonais produzidos, que poderá ser operado a altas taxas de compressão.

- A implementação das funções de extração de vértices em uma FPGA e o desenvolvimento de uma placa para a emulação do sistema, a fim de avaliar o comportamento do algoritmo em condições reais de operação.

- Após realizar os testes da implementação em FPGA, os *layouts* poderão ser enviados para a fundição onde serão integrados em chips.

Referências

[AF86] N. Ayache and O. D. Faugeras. "HYPER: A new approach for the recognition and positioning of two-dimensional objects". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8:44–54, 1 1986.

[CH88] F. Cheng and W. Hsu. Parallel algorithm for corner finding on digital curves. *Pattern Recognition Letters*, 8:47–53, 1988.

[Cor92] P. C. Cortez. Object recognition by polygonal modeling. Master's thesis, Federal University of Paraiba - Brazil, LAPS - DEE - UFPb, August 1992. Published in Portuguese.

[Cor95] Cortez, P. C. e Carvalho, J. M. "Modelagem Poligonal de Contornos 2d Usando a Transformada de Hough". 13^{o} *Simpósio Brasileiro de Telecomunicações, Águas de Lindóia-SP*, Setembro 1995.

[Cun87] G.J. Cunha. *"Computação Gráfica"*. Atlas S.A., 1987.

[eMS96] T. Y. Tian e M. Shah. "Motion estimation and segmentation". *Machine Vision and Applications*, 9:32–42, 1996.

[GFGT95] M. Gheari, S. De Faria, I. N. Goh, and K. T. Tan. "Motion compensation for very low bitrate video". *Signal Processing: Image Communication*, 7:567–580, 1995.

[Gha91] H. Gharavi. "Multilayer subbe-based video coding". *IEEE Trans. Commun.*, 39:1288–1291, 1991.

[GW77] R. C. Gonzalez and P. Wintz. *"Digital Image Processing"*. Addison-Wesley Publishing Company, 1977.

[GW87] R. C. Gonzalez and P. Wintz. *Digital Image Processing.* Addison Wesley, New York, 1987.

[HAS89] H. Harashima, K. Aizawa, and T. Sailton. "Model-based analysis synthesis coding of videophone images - conception and basic study of intelligent image coding". *The Trans. of IEICLE*, page 72, 452-45 1989.

[INC94] CADENCE DESIGN SYSTEMS INC. *"Verilog-XL Reference Manual 2.0"*. 1994.

[KD82] Y. Kurozumi and W. A. Davis. "Polygonal approximation by the minimax method". *Computer Graphics e Image Processing*, 19:248–264, 1982.

[KP89] T. Komarek and P. Pirsch. "Array architectures for block-matching algorithms". *IEEE Trans. on Circuits and Systems*, pages 1301–1308, 1989.

[MC96] E. Melcher and J.M. et al Carvalho. A novel gradient operator suited for vlsi implmentation of 2d shape recognition. In *Anais do SBCCI 96*, March 1996. Published in Portuguese.

[MCC96] E. Melcher, J.M. Carvalho, and P.C. et al Cortez. A vertex detection algorithm for vlsi implementation. In *Anais do SIBGRAPI XI*, Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens, pages 367–368, 1996. Published in Portuguese.

[MdCdB+95] Elmar Melcher, João Marques de Carvalho, Marcelo de Barros, Lírida Naviner, Jean François Naviner, Valteir R. da Silva, Fred Henrique Souza Paes, and Ricardo A. S. Moreira Jeanne E. de P. Braquehais. "A 2D Shape Boundary Detection Algorithm for VLSI Implementation". *Proc. of VIII Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens*, Oct. 1995.

[MM92] A.D Marshall and R.R. Martin. *Computer Vision, Models and Inspection.* World Scientific, London, U.K., 1992.

[MNaMdC+97] Elmar Melcher, Lírida Naviner, Jo ao Marques de Carvalho, Jean François Naviner, Ricardo A. S. Moreira, Yuri de Mello Villar, and Marcos Morais. "VLSI Implementation of Contour Extraction from Real Time Image Sequences". *VLSI*, 1997.

[O'C97] K.J. O'Connell. "Object-adaptive Vertex-based Shape Coding Method". *IEEE Trans. on Circuit and Systems for Video Technology*, 7, 1 1997.

[Pap65] A. Papoulis. *"Probability, Random Variables and Stochastic Processes"*. McGraw-Hill, 1965.

[Pav78] T. Pavlidis. "A review of algorithms for shape analysis". *Computer Graphics and Image Processing*, 7:243–258, 1978.

[PKL92] H. Paek, R. C. Kim, and S. U. Lee. "On the motion compensated transform coding technique employing sub-be decomposition". *SPIE Proc. Series*, pages 253–264, 1992.

[Ram72] U. Ramer. An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing*, 1:224–256, 1972.

[RK76] A. Rosenfeld and A. C. Kak. *Digital Picture Processing.* Academic Press, New York, 1976.

[SM92] F. Stein and G. Medioni. "Structural indexing: Efficient 2-d object recognition". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14:1198–1204, 12 1992.

[SR92] I. K. Sethi and N. Ramesh. "Local association based recognition of two-dimensional objects". *Machine Vision and Applications*, 5:265–276, 1992.

[WE92] Neil H.E. Weste and Kamran Eshraghian. *Principles of CMOS VLSI Design: a Systems Perspective.* Addison-Wesley Publishing Company, 2nd edition, 1992.

[WH78] R.Y. Wong and E.L. Hall. Scene matching with invariant moments. *Computer Graphics and Image Processing*, 8:16–24, 1978.

[YSN97] S.S. Young, P.D. Scott, and N.M. Nasrabi. "Object Recognition Using Multilayer Hopfield Neural Network". *IEEE Trans. on Image Processing*, 6, 3 1997.

[ZLRG95] S. Zhang, M. Liang, J. A. Robinson, and G. L. Greg. "Motion coding of image primitives". *Signal Processing: Image Communication*, 7:457–469, 1995.