MARLON FERNANDES DE ALCANTARA

E-LAPIS – RECONHECIMENTO DE PRIMITIVAS GEOMÉTRICAS EM DESENHOS VETORIAIS

JOINVILLE – SC

2009

UNIVERSIDADE DO ESTADO DE SANTA CATARINA UDESC

CENTRO DE CIÊNCIAS TECNOLÓGICAS CCT

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

MARLON FERNANDES DE ALCANTARA

E-LAPIS – RECONHECIMENTO DE PRIMITIVAS GEOMÉTRICAS EM IMAGENS VETORIAIS

Trabalho de conclusão de curso submetido à Universidade do Estado de Santa Catarina como parte dos requisitos para a obtenção do grau de Bacharel em Ciência da Computação

Orientador: Alexandre Gonçalves Silva.

JOINVILLE – SC

2009

MARLON FERNANDES DE ALCANTARA

E-LAPIS – RECONHECIMENTO DE PRIMITIVAS GEOMÉTRICAS EM DESENHOS VETORIAIS

Este Trabalho de Conclusão de Curso foi julgado adequado para a obtenção do título de Bacharel em Ciência da Computação e aprovado em sua forma final pelo Curso de Ciência da Computação Integral do CCT/UDESC.

Banca Examinadora:

Orientador: _____

MSc, Alexandre Golçalves Silva
Departamento de Ciência da Computação – UDESC

Membro: _____

PhD, Daniela Gorski Trevisan
Departamento de Ciência da Computação – UDESC

Membro: _____

Dr, Claudio Cesar de Sá
Departamento de Ciência da Computação – UDESC

Joinville, 6 de novembro de 2009

AGRADECIMENTOS

Aos meus pais Ademir e Beatriz, e irmãos Maicon e Katianne, por todo apoio e confiança.

Ao meu orientador Alexandre, onde sem sua ajuda seria impossível a realização deste trabalho.

A todos os professores da UDESC pela formação que me foi dada.

A todos os meus amigos que estiveram presente durante esta jornada, e em especial aos amigos Ricardo, Guilherme, Silvio, Gabriela e Zuraica, que de alguma forma me forneceram um apoio pessoal ou técnico.

Nunca deixe que lhe digam que não vale a pena acreditar no sonho que se tem, ou que seus planos nunca vão dar certo, ou que você nunca vai ser alguém. Tem gente que machuca os outros, tem gente que não sabe amar. Mas eu sei que um dia a gente aprende, se você quiser alguém em quem confiar, confie em si mesmo. Quem acredita sempre alcança.

Renato Russo

RESUMO

Algumas imperfeições, em desenhos livres manuais produzidos por meio de dispositivos vetoriais (mouse, mesas digitalizadoras, telas sensíveis ao toque, etc), podem ser tratadas no sentido de promover uma melhor qualidade visual da imagem final. Com a transformação de pontos isolados em objetos vetoriais, uma representação concisa do desenho é obtida, possibilitando uma posterior edição simples e mais intuitiva. Este trabalho consiste na análise e interpretação de pontos no espaço bidimensional com o objetivo de classificá-los e descrevê-los em primitivas geométricas ou curvas paramétricas. Para isto, um levantamento de formas de aquisição, padrões vetoriais de imagens e métodos relacionados de interpretação de pontos devem ser detalhados, assim como projetada e implementada uma proposta de reconhecimento de elementos semânticos em traços de desenhos feitos à mão.

Palavras-chave: *Imagens vetoriais, interpretação de imagens, tratamento de pontos.*

ABSTRACT

Some imperfections in freehand drawing generated by vector devices (mouse, graphic tablets, touch-screens etc.), can be treated in order to promote a better visual quality of the final image. Using the transformation of isolated points in vector objects, a concise representation of the drawing can be obtained, allowing a simpler and more intuitive later edition.

This project consists of the analysis and interpretation of points in a two-dimensional space, in order to classify and describe them through geometric primitives or parametric curves. With that aim, a survey of acquisition forms, vector pattern of images, and related interpretation methods for point interpretation need to be detailed, as well as it is necessary to design and implement a proposal for recognition of semantic elements in traces from hand drawings.

Keywords: *Vector images, image interpretation, treatment of points.*

LISTA DE FIGURAS

LISTA DE SIGLAS E ABREVIATURAS

Sigla	**Descrição**
CDR	CorelDraw Record
EPS	Encapsuled PostScript
XML	eXtensible Markut Language
FIG	Formato da Ferramenta xFIG
PDF	Portable Document Format
POTRACE	A Polygon-based Tracing Algorithm
PS	PostScript
SVG	Scalable Vector Graphics
W3C	World Wide Web Consortium
WMF	Windows MetaFile

LISTA DE TABELAS

SUMÁRIO

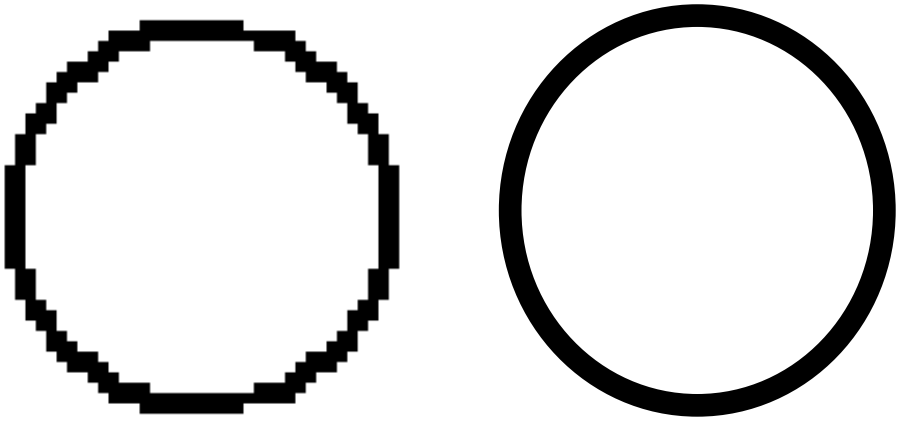
1. INTRODUÇÃO

As imagens nada mais são que representações visuais de algo que se quer expressar, que pode ser material como em uma fotografia ou abstrato como um desenho ou representação artística da realidade. O “processamento das imagens” tem por objetivo extrair informações das imagens ou transformá-las em outras imagens onde seja mais fácil a extração de informações. Como o processamento de imagens segue a mesma linha do “processamento de sinais”, onde ambos são suportes físicos que possuem alguma informação a ser transmitida, se esta informação se refere a uma medida, dizemos que a informação está associada a um fenômeno físico, caso esta associação seja a um nível cognitivo, chamamos de conhecimento [Albuquerque e Albuquerque, 2001; Rosário, 2008].

A “computação gráfica” segue a linha oposta ao “processamento de imagens”, quando a preocupação do “processamento de imagens” é extrair a informação, a “computação gráfica” busca transformar a informação em elementos visuais (gráficos). A computação gráfica faz uso de diversos métodos matemáticos a fim de conseguir uma representação visual para a informação a ser transmitida, o que não significa que o processamento de imagens não possa fazer uso dos mesmos métodos afim de conseguir o processo inverso, já que mesmo sendo trabalhos opostos, ambas estão ligadas no que se refere a área de trabalho [Albuquerque e Albuquerque, 2001].

Imagem matricial, ou bitmap (mapa de bits), é a representação de imagens através de uma matriz de pixels. Um pixel é a menor quantidade em uma imagem, ocupa uma posição na matriz e assume uma determinada cor, ou seja, a representação da imagem matricial depende da característica de cada pixel. Estes são dispostos de forma que sua união crie a aparência de uma figura. Por exigir que cada pixel contenha certo valor de cor e posição, as imagens matriciais podem tornar-se grandes para armazenamento. A mesma idéia de matriz de pontos é usada, por exemplo, para impressão de imagens e monitores de vídeo, algumas imagens capturadas por câmeras digitais e scanners também usam essa forma de representação de imagens [Derakhshani e Munn, 2008].

Os desenhos em formatos vetoriais possuem a capacidade de representar as suas imagens através de matrizes e fórmulas matemáticas, sendo dessa forma, muito mais compacto que imagens do tipo matricial. Dispensa-se nesse caso, explicitar as coordenadas de cores de cada pixel da imagem, sendo somente necessário mensurar as coordenadas de determinada primitiva geométrica, dessa forma, não é armazenada a informação visual, mas sim as instruções para criação das imagens. O formato vetorial permite o seu redimensionamento sem que com isso haja uma perda de qualidade, o que não ocorre no formato matricial, onde os pixels são reorganizados para ser efetuado o redimensionamento, onde em certo ponto pode perder o conjunto visual, ficando visíveis apenas os pixels em sua forma quadrangular, conforme pode ser visto na Figura 1. Outra vantagem seria na edição das imagens vetoriais, onde é possível manipular toda a sua forma ao mesmo tempo e não somente cada pixel por vez [Coelho e Fagundes, 2004].



Círculo Ampliado em imagem matricial Círculo Ampliado em imagem vetorial

Figura 1 – Comparação entre a ampliação de uma imagem matricial e uma imagem vetorial.

Para todo, qualquer formato vetorial baseia sua especificação em primitivas geométricas, cujo sua especificação se baseia em parâmetros que podem ser descritos de forma geométrica através de funções. As primitivas mais comuns nesse formato são: retângulo, círculo, linha, polígono, elipse e curvas. Ainda que algum formato possa suprimir ou acrescentar alguma outra primitiva. Todavia as imagens vetoriais não são indicadas para imagens naturais devido a não possibilidade de representá-la pelas primitivas geométricas, e para formas complexas, devido ao tempo que levaria a interpretação da imagem [Coelho e Fagundes, 2004].

O trabalho se restringirá nas imagens abstratas, mas precisamente em desenhos na forma binária onde existe cor, ou a sua ausência em determinado ponto da imagem. Observa-se que o trabalho de aquisição, o tratamento inicial, a transformação na forma binária e a especificação na representação vetorial do padrão SVG, já foram realizadas em um trabalho anterior [Rosário, 2008].

A detecção de formas é um método que consiste em, dado um conjunto de pontos, que podem ser pixels de uma imagem, dizer a que forma o dado conjunto mais se assemelha, e se é possível afirmar que o conjunto de pontos representa realmente aquela forma para uma tolerância especificada. Esta detecção pode ser realizada de duas maneiras, uma seria por análise de forma a partir de figuras geométricas de formulação simples ou critério morfológico, como por exemplo: linhas, círculos, retas, arcos, elipses, curvas. Outra maneira seria a detecção por casamento de padrões, quando a imagem a ser detectada possui uma forma complexa, por exemplo, um caractere ou outros símbolos, neste caso o método mais indicado seria por casamento de modelo da figura perfeita com a figura a ser reconhecida [Silva e Lotufo, 2006].

O reconhecimento de primitivas geométricas em desenhos vetoriais pode auxiliar em muitos aspectos. Primeiramente, é de alta complexidade para o usuário de um sistema desenhar uma primitiva geométrica de forma perfeita, desse modo, por mais que o usuário desenhe a forma desejada de maneira imperfeita, é possível, a partir da imagem gerada, interpretar a forma desejada e representá-la perfeita, mesmo que o usuário a tenha desenhado diferente. Nos desenhos a mão, principalmente se for desejado fazer de forma mais rápida ou sem a ajuda de equipamentos que auxiliem na precisão, se torna inviável a elaboração de figuras. Com a interpretação computacional da imagem, não é necessário a minuciosa preocupação com a forma ainda na etapa de desenho, e como a imagem final gerada, é na forma vetorial, torna ainda mais fácil o seu manuseio para uma eventual correção semântica manual. Coyette (2007), por exemplo, utiliza esta idéia no auxílio à construção de interfaces gráficas.

Outro aspecto seria a simplificação da especificação da imagem final, tornando-a mais simples e menos extensa. Formando dessa maneira a união de diversas primitivas que compõe um objeto em uma especificação única do objeto como um todo. Dessa forma, quando é desejado fazer a edição da

imagem, e a alteração de elementos visuais quaisquer, é possível manusear o objeto de forma completa ao invés de realizar o mesmo procedimento para as diversas partes.

Para imagens oriundas de dispositivos como câmeras digitais, ou webcams, seria possível a partir da imagem adquirida e seus eventuais ruídos, obter os pontos da imagem, e a partir deles, fazer a interpretação geométrica e a união dos pontos visuais da imagem, tornando uma imagem de especificação limpa, e facilmente manuseada pelo usuário. Também ajudaria a estruturar parte da imagem que ficou distorcida devido a perspectiva visual do equipamento de captura, ou por quaisquer outras pequenas falhas da imagem ou do desenho manual.

1.1. Objetivos

1.1.1. Objetivo geral

Acrescentar ao e-Lapis [Rosário, 2008] a funcionalidade de simplificação de imagens vetoriais a partir da interpretação da disposição espacial dos pontos gerados, e o reconhecimento de primitivas geométricas, de modo a produzir imagens com maior qualidade e com suporte à edição de novos objetos gráficos.

1.1.2. Objetivos específicos

1. Estabelecer métodos de aquisição de imagens vetoriais.
2. Compreensão de padrões vetoriais.
3. Tratamento dos pontos a partir de imagens vetoriais.
4. Definir o conjunto de primitivas geométricas a serem analisadas.
5. Estudo e escolha de métodos para a obtenção das primitivas a partir dos pontos dados.

1.2. Resultados Esperados

Espera-se ao final desde trabalho, obter uma aplicação que dados pontos no espaço provindos de um documento em texto puro, analise a

imagem e obtenhas as formas aproximadas que os pontos representam. E criando no formato vetorial a imagem que mais se assemelha a sequência de pontos dados como parâmetro e uma determinada tolerância para o reconhecimento das formas.

1.3. Estrutura do Trabalho

O trabalho foi estruturado de forma a primeiramente estabelecer a base conceitual utilizada para a elaboração dessa proposta, com isso dada a introdução que busca situar a área em que será desenvolvida a aplicação e o contexto em que se encontra, sendo também este primeiro capítulo responsável por esclarecer os objetivos do trabalho. O capitulo 2 em conjunto com o capítulo 3 busca fornecer a base necessária para o desenvolvimento do projeto final de reconhecimento das formas, para isso é levantado uma breve fundamentação teórica que exibe conceitos e dedução de cálculos que virão a ser usados na implementação final. Após isso parte-se para a exploração de trabalhos relacionados à aplicação, a fim de obter algoritmos que se assemelhem com parte da idéia final. Sendo obtida a base para a implementação final, explora-se uma sintaxe para a geração da saída da semântica obtida, dessa forma, o capítulo 4 exibe alguns padrões vetoriais, e se aprofunda na sintaxe daquele que será utilizado efetivamente para a geração da saída. O capítulo 5 por fim exibe uma proposta para a solução do trabalho, com base nos algoritmos levantados no capítulo 3, utilizando os conceitos matemáticos obtidos no capítulo 2. Explorando desde a entrada do sistema, até a sua gravação final na sintaxe levantada no capítulo 4. Ao final do trabalho são apresentadas as considerações finais gerais em relação ao trabalho, e por último as referências utilizadas.

2. FUNDAMENTAÇÃO TEÓRICA

Este capítulo busca esclarecer os conceitos que serão mencionados ao longo deste trabalho, bem como a fundamentação necessária para o entendimento e formulação dos métodos que visam realizar o objetivo desde deste. Os conceitos destacados referem-se basicamente a geometria analítica e computação.

2.1. Figuras Planas

É dito uma figura plana toda a figura que tem todo o seu perímetro descrito em um plano, e que dessa forma se pode definir uma área caso essa seja fechada. A geometria Euclidiana prevê formas planas definidas como primitivas. É dito um polígono toda a figura formado apenas por segmentos retos, e pode se dizer que o polígono é equiângulo quando possui todos os ângulos iguais, equilátero quando possui todos os lados iguais, e é dito um polígono regular quando os ângulos e os lados são iguais.

Uma circunferência é uma figura geométrica formada por infinitos pontos que são equidistantes de um ponto dito *ponto central*, essa distância do ponto central aos pontos que definem seu perímetro é dado o nome de raio da circunferência [SKIENA e REVILLA, 2003].

Um retângulo é uma figura geométrica que possui 4 lados e todos os ângulos são iguais, esses lados tem seus pares paralelos de medidas iguais. Quando todos esses lados são iguais, a figura caracteriza também um quadrado [SKIENA e REVILLA, 2003];

É dito um segmento de reta, a menor distância entre dois pontos, essa distância se prolongada, representa uma reta, um par de pontos é suficiente para definir uma. Uma reta também pode ser definida através de uma equação onde $y = mx + b$ onde m seria a inclinação, b o ponto onde a reta intercepta o eixo y, e (x, y) as coordenadas no espaço de determinado ponto da reta [SKIENA e REVILLA, 2003].

2.2. Curvas

É dito uma curva um segmento que liga dois pontos e tem sua forma descrita através de parâmetros (paramétrica) ou a uma constante (implícita), na primeira forma, é possível se verificar se um ponto está contido na curva se atribuindo valores aos parâmetros utilizados, um exemplo de curva paramétrica são as curvas de Bézier que serão utilizadas.

2.2.1. Curvas de Bézier

A idéia das curvas de Bézier consiste basicamente em sucessivas interpolações lineares em relação aos pontos $\{B_0, B_1, \dots B_n\}$ que são ditos pontos de controle. Para gerar os pontos tomam-se inicialmente as linhas que ligam B_n a B_{n+1} para cada $t \in [0,1]$. É definido então para cada um desses segmentos outro ponto $B_i^{(1)} = tB_i + (1-t)B_{i+1}$ por interpolação linear, dessa, forma é construído um novo polígono de controle $\{B_0^{(1)}, B_1^{(1)}, \dots B_{n-1}^{(1)}\}$, tendo assim $n-1$ segmentos a cada novo passo até que n seja igual a 1, e tenha apenas um único ponto [SEGURA, 2005].

As curvas de Bézier tem algumas propriedades:

- A curva está sempre totalmente interna ao polígono definido pelos pontos de controle.
- A curva começa no primeiro ponto de controle e termina no último, e não necessariamente toca os demais pontos.
- A curva não varia independente da transformação executada.
- Os vetores que tangem os pontos finais possuem a mesma direção do primeiro e do último segmento do polígono de controle.
- O grau da curva de Bézier é definido como um a menos que o número de pontos de controle.

As propriedades são válidas independente do sistema em que está sendo trabalhado (R^2, R^3) [SEGURA, 2005].

2.3. Distâncias

O estudo das distâncias está focado na distância de pontos a algum determinado objeto que é o que será utilizado neste trabalho. A idéia do ponto nesse caso pode ser abstraída para pixel, ou para algum nó de uma grade ou qualquer elemento que possua coordenadas cartesianas (valores que definem a posição no espaço de um objeto).

2.3.1. Distância entre dois pontos

Dado dois pontos A e B contidos em um plano, contendo coordenadas cartesianas respectivamente (x_1, y_1) e (x_2, y_2), pode-se definir a distância entre os dois pontos $d(A, B)$ se traçando as projeções dos pontos em relação aos eixos coordenados, obtendo assim um terceiro ponto C com coordenadas (x_1, y_2), e fazendo o cálculo utilizando o teorema de Pitágoras, dado que os valores dos catetos serão respectivamente $|x_2 - x_1|$ e $|y_2 - y_1|$. Dessa forma podemos dizer que a fórmula geral da distância entre dois pontos no plano será:

$$d(A, B) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Essa fórmula é valida para calcular a distância entre quaisquer pontos no plano [STEINBRUCH e WINTERLE, 1987].

2.3.2. Distância entre um ponto e uma reta

À distância $d(P, R)$ de um ponto qualquer P de coordenadas (x_0, y_0) a uma reta R de equação geral $Ax + By + C = 0$ pode ser definida pela projeção ortogonal de um vetor que parte de um ponto que pertence à reta R e segue ao ponto P. Dessa forma, qualquer que seja o ponto em R tomado a sua componente ortogonal será a distância entre o ponto e a reta. Dessa forma teríamos como fórmula geral:

$$d(P,R) = \frac{|Ax + By + C|}{\sqrt{A^2 + B^2}}$$

Analisando a fórmula é possível ver que é traçado a reta passando pelo ponto P de forma a analisar a componente ortogonal a reta R [STEINBRUCH e WINTERLE, 1987].

2.3.3. Distância entre um ponto e uma circunferência

Dada uma circunferência C de raio r e centro localizado no ponto c de coordenadas (x_c, y_c) é possível dizer que a distância da circunferência até um dado ponto P de coordenadas (x_0, y_0) se obtendo a distância do ponto P ao centro da circunferência c através e se subtraindo desta o raio r. Conforme a fórmula:

$$d(P,C) = \left|\sqrt{(x_c - x_0)^2 + (y_c - y_0)^2} - r\right|$$

Se tirarmos o módulo do cálculo é possível dizer que o ponto está interno a circunferência caso o valor seja negativo, está sobre a circunferência caso a distância seja 0, e está externo a circunferência caso esse valor seja positivo.

2.3.4. *City-Block* e *Chessboard*

A distância *city-block* é utilizada para quando se tem um espaço discretizado e há uma adjacência entre os pontos que representam uma imagem, nesta métrica de distância é considerada a adjacência somente com o ponto acima, abaixo, a esquerda e a direita do ponto dado. Dessa forma o número de pontos que representa o menor caminho de um ponto até outro é chamado de distância *City-Block*. Considerando dois pontos na imagem p e q com coordenadas respectivas (x_1, y_1) e (x_2, y_2) a fórmula da distância será:

$$D_{cb}(p,q) = |x_1 - x_2| + |y_1 - y_2|$$

Da mesma forma como o *City-Block* a métrica *Chessboard* é aplicada, se utilizando como adjacência todas as oito adjacências do ponto (acima,

abaixo, esquerda, direita, acima-esquerda, acima-direita, abaixo, esquerda, abaixo-direita), dessa forma se faz uso dos caminhos diagonais para se calcular a distância. Essa distância pode ser obtida se pegando o maior valor entre as diferenças entre as componentes x e y, conforme a fórmula:

$$D_{ch}(p,q) = \max\{|x_1 - x_2|, |y_1 - y_2|\}$$

Esses métodos de cálculo de distâncias são mais utilizados quando se tem o espaço discretizado em pontos. [GONZALEZ e WOODS, 2000].

Estes cálculos de distâncias são mais rápidos de serem calculados do que o método euclidiano, sendo mais indicado quando se deseja saber o número mínimo de pixels que estão no intervalo entre os dois pixels que se está calculando, e este valor é sempre preciso.

3. TRABALHOS RELACIONADOS

O estudo dos trabalhos relacionados, busca levantar o que já foi desenvolvido e estudado em áreas próximas, a fim de obter técnicas para o desenvolvimento da aplicação desejada. Foi procurado relatar trabalhos de diferentes objetivos para uma amplitude de embasamento maior. No contexto, algoritmos de vetorização, interpretação de formas, casamento de padrões e reconhecimento de gestos.

3.1. Potrace

O algoritmo Potrace (SELINGER, 2003) tem por objetivo principal transformar uma imagem bitmap em um desenho vetorial, partindo no início, da obtenção de caminhos na imagem a partir da adjacência entre pixels, para através deles, obter os polígonos resultantes, para só então transformar em curvas e agregá-las, resultando na imagem final.

3.1.1. Decomposição em Caminhos

É assumido na imagem pixels pretos e brancos, e é dito que os cantos, e não os centros dos pixels possuem coordenadas em inteiros, e é assumido que o fundo da imagem é branco, e que o preto é a parte a ser descrita vetorialmente. É possível com isso definir um grafo direcionado com uma sequência a ser seguida, dado *p* um ponto de coordenadas inteiras, esse ponto, por ser um canto, tem 4 pixels adjacentes. Se os pixels adjacentes não são todos da mesma cor, esse ponto é então chamado de vértice. Todavia, se *v* e *w* são vértices, é dito que existe uma aresta que liga *v* a *w*, se a distancia Euclidiana entre eles é 1. Se um segmento separa um pixel branco de um pixel preto, de forma que o pixel preto está a esquerda, enquanto o pixel branco está a direita, é dado continuidade ao caminho, sempre tentando manter a mesma configuração dos pixels laterais [SELINGER, 2003].

Um caminho(*path)* é uma sequência de vértices $\{v_0, v_1, \dots, v_n\}$ sendo que existem arestas que vão de v_i até v_{i+1}, para todo $i = 0, \dots, n-1$. De forma que

todas essas arestas são distintas. Um caminho é considerado fechado se $v_0 = v_n$. O tamanho do caminho é o número de arestas que ele contém. O objetivo do algoritmo é decompor um dado grafo *G* em um grafo composto por caminhos fechados [SELINGER, 2003].

Para encontrar uma aresta que ocorra exatamente uma vez em um caminho, o Potrace utiliza um método simples. Inicialmente, encontre dois pixels adjacentes, que sejam diferentes, ou seja, um preto e um branco, e faça um segmento orientado de forma que o pixel preto fique a esquerda e o branco a direita, em seguida, conforme a Figura 2, então inicialmente temos um caminho de comprimento 1, continuar a aumentar esse caminho, de forma que sempre se tenha a mesma situação com o pixel esquerdo a esquerda e um branco a direita, e continua-se até chegar ao pixel que se estava inicialmente, quando se chegar nessa situação, logo, temos um caminho fechado. Cada vez que se fecha um caminho, é pego todos os pixels do seu interior que nesse caso são pretos, e se atribui a eles a cor branca, gerando uma nova imagem, e retorna a busca por um novo caminho recursivamente, até que não existem mais pixels pretos na imagem. Ao final dessa etapa, se tem toda a definição de todos os caminhos para serem trabalhados independentemente [SELINGER, 2003].

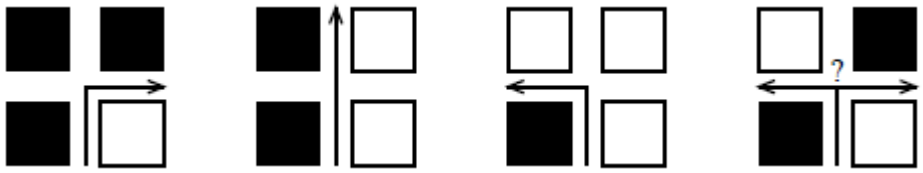


Figura 2 – Algoritmo para prolongar o caminho (SELINGER, 2003).

3.1.2. Polígonos

A segunda parte tem como entrada os caminhos fechados, e os relaciona a polígonos semelhantes. Esta etapa já tem como resultado, uma imagem de certa forma semelhante à original.

Retas

Para se gerar um polígono é necessário primeiramente encontrar os segmentos de reta que serão os lados desse polígono. Dados dois pontos

$z_0 = (x_0, y_0)$ e $z_1 = (x_1, y_1)$ em coordenadas em um plano que não precisam ser inteiros. A distância máxima é dado por $d(z_0, z_1) = \max\{|x_1 - x_0|, |y_1 - y_0|\}$, e se deve chegar a no máximo 1/2 a partir do ponto que é apenas para representar a distância até o centro do pixel [SELINGER, 2003].

Dado um caminho não fechado, podemos dizer que ele constitui uma reta aproximada se para um conjunto de pontos do caminho $p = \{v_0, \dots, v_n\}$ respeitar as seguintes condições, $d(v_0, a) \leq \frac{1}{2}$, $d(v_n, b) \leq \frac{1}{2}$ e pra cada ponto $i = 1, \dots, n-1$, existe pelo menos um ponto c_i sobre $\overline{ab}$ tal que $d(v_i, c_i) \leq \frac{1}{2}$. Todavia um caminho só é dito ser uma reta aproximada quando a regra das distâncias estiver de acordo, e também que não ocorra dodas as quatro possíveis direções no caminho, ou seja, não pode ocorrer simultaneamente desvios para baixo, esquerda, cima e direita. No caso da Figura 3, na parte 'c' é dito que não é reta pelas regras das distâncias, já no item 'e', não é dito reta, pois é feito uso das quatro direções para montar o caminho, os demais('a', 'b' e 'd') respeitam a regra das distancias, e ainda existe um ponto c sobre o segmento q que está interno a qualquer um dos pontos do *path* [SELINGER, 2003].

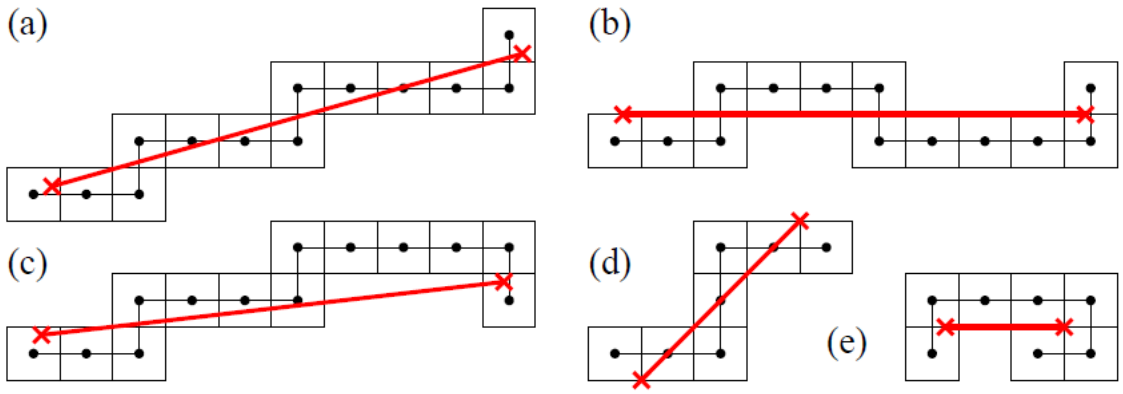


Figura 3 – Exemplos de aproximação por reta de um caminho (SELINGER, 2003).

Sub-caminhos de um polígono

Considerando um caminho fechado $p = \{v_0, \dots, v_n\}$, onde $v_n = v_0$para caracetizar um ciclo, podemos afirmar que para qualquer par de índices $i, j \in \{0, \dots, n-1\}$, p_{ij} definirá um sub-caminho desse ciclo. Esse subcaminho terá seu tamanho definido como $j - i$ caso $j \geq i$ ou $j - i + n$ caso $j < i$. Assim

pode-se dizer que o tamanho é a diferença cíclica entre i e j [SELINGER, 2003].

Para construir um polígono, a partir de um caminho fechado, se faz nesseário encontrar as retas aproximadas contidas no polígono, considera-se um segmento possível quando a diferença cíclica entra os pontos do segmento não ultrapasse a margem de $n - 3$. Então basta verifcar se o segmento $p_{i-1,j+1}$ é uma reta aproximada, caso seja verdade pode-se tentar extenter o segmento em ambos os lados e ainda assim, continuar sendo umas reta. Então o recorte do segmento pode ser feito simultaneamente em ambas as pontas da reta. Sempre alguns pontos poderão ser definidos como uma reta, pois nota-se se que qualquer caminho de tamanho 3 pode ser definido como uma reta que vai do primeiro ponto até o último ponto do segmento [SELINGER, 2003].

Por esta proposta, um polígono seria definido por um conjunto de índices onde sempre haveria um segmento que ligaria o ponto representado pelo índice k ao ponto definido pelo índice $k + 1$, o número de índices irá variar conforme a tolerância para o tamanho estabelecido e a complexidade da imagem de entrada, para a , há duas representações através de polígonos para a mesma imagem de entrada [SELINGER, 2003].

Escolha do melhor polígono

Dados os vários polígonos gerados a partir da mesma imagem, o melhor polígono seria aquele que tivesse o menos número de vértices, como no caso da Figura 4, a figura da esquerda possuim 14 vértices, contra 17 da segunda figura, nesse caso o da esquerda é um polígono melhor. Mas muitas vezes é obtidos diversos polígonos com a mesma quantidade de vértices, e se torna necessário decidir entre eles, para isso é adicionado um grau de desvantagem de um polígono em relação ao outro com base nas distâncias euclidianas entre os pontos do polígono. A penalidade de um segmento $\overline{v_i, v_j}$ é dada pela fórmula: $$P_{ij} = \left|v_i - v_j\right| \cdot \sqrt{\frac{1}{distancia\ ciclica(i,j)+1} \cdot \sum_{i+1}^{j} dist\left(v_k, \overline{v_i v_j}\right)^2}$$ [SELINGER, 2003].

O cálculo da penalidade é baseado na distancia enclidiana de uma linha a uma reta, logo, quanto maiores as distâncias, maior será a penalidade, pode-se dizer que terá maior penalidade o polígono que de maneira geral tiver pontos mais dispersos da forma que o está definindo. É escolhido para todos os efeitos, caso tenha o mesmo número de vértices, o polígono que tiver o menor grau de penalidade [SELINGER, 2003].

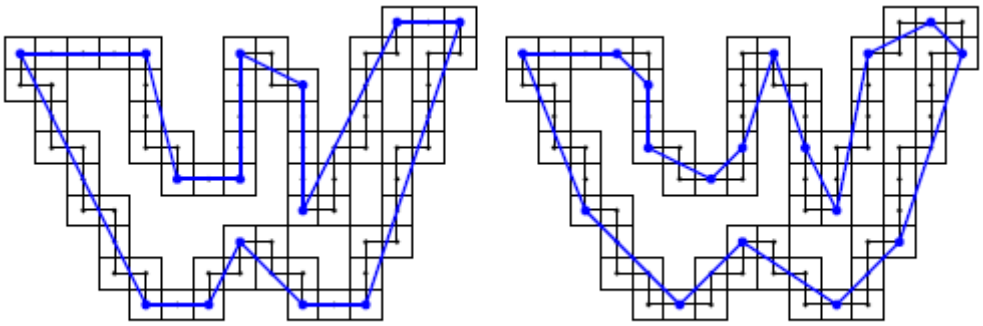


Figura 4 – Dois possíveis polígonos a partir da mesma imagem (SELINGER, 2003)

3.1.3. Montagem do esboço vetorial

Após obter o polígono, resta esboçá-lo ele na forma vetorial, para isso estabelecendo como que os pontos do polígono melhor se encaixam no *bitmap* da imagem original para isso ajustando a posição dos vértices, os ângulos e definindo curvas que melhor definem as retas com suas adjacentes.

Ajuste dos vértices

A saída da primeira fase do algoritmo é um polígono definido pelos vértices $\{i_0, \dots, i_{m-1}\}$ que descrevem um caminho fechado $\{v_0, \dots, v_n\}$. Dessa forma a tarefa então é associar os índices $i_0, \dots, i_{m-1}$ aos vértices, obtendo os valores $v_{i_0}, \dots, v_{i_{m-1}}$ e definindo-os como vértices do polígono. Realizando a associação de cada vértice i_k com um ponto a_k de coordenadas no plano não necessariamente inteiras, é dito que a_k é próximo à v_{i_k} e que, para qualquer dois pontos consecutivos do polígono i_k e i_{k+1} resulta num segmento $\overline{a_k a_{k+1}}$ que corresponde ao sub-caminho $\{v_{i_k}, \dots, v_{i_{k+1}}\}$ [SELINGER, 2003].

O seguinte algoritmo foi utilizado para situar os pontos a_k, para cada par de vértices i_k e i_{k+1} é calculada uma reta L_{kk+1} que melhor se aproxima dos

pontos $v_{i_k}, \ldots, v_{i_{k+1}}$ de forma a minimizar as distâncias euclidianas. Todavia se i_{k-1}, i_k e i_{k+1} são pontos consecutivos, o ponso a_k seria situado na intersecção entre $L_{k-1,k}$ e $L_{k,k+1}$. Porém como não se deseja que em alguns casos o ponto a_k fique muito longe do vértice v_{i_k} é definido uma distância máxima $d(a_k, v_{i_k}) \leq 1/2$. Em um caso particular, caso a intersecção entre $L_{k-1,k}$ e $L_{k,k+1}$ não gera uma intersecção o ponto a_k é colocado próximo do vértice v_{i_k} [SELINGER, 2003].

3.1.4. Conclusão da Seção

O trabalho Potrace trata toda a imagem como pontos e realiza o agrupamento deles. É considerado na imagem, todo aglomerado de de um ou mais pontos como uma componente que será representado através de uma curva. Sendo que ao final, qualquer forma é representado por um conjunto de curvas.

O grau de fidelidade do desenho obtido através das curvas, porém seria interessante o suporte a novos objetos gráficos, para o caso de se precisar editar a imagem. Sendo as imagens geradas de boa qualidade, porém sem o reconhecimento real da forma no objeto vetorial. Todavia a parte da obtenção do caminho que define um aglomerado de pontos é eficiente e pode ser usado em pararelo com alguma outra técnica afim de representar a imagem final.

3.2. *Passdoodles*

O trabalho Passdoodles (VARENHORST, 2004) realiza autenticações de usuários a um sistema por meio de uma análise de um desenho feito a mão instantaneamente. Observando para isso a velocidade com que o desenho foi feito, a identificação do desenho desenhado, e a dispersão dos pontos na grade de acordo com o desenho feito a mão pelo usuário utilizando algoritmos de reconhecimento de escrita ou desenho.

3.2.1. Idéia do sistema

A ideia do projeto é dada inicialmente uma assinatura criada, o sistema armazenaria os dados dessa assinatura, quanto à forma do desenho, distribuição dos pontos, e o tempo da escrita em casa ponto. Dadas as métricas, no momento da autenticação do usuário são verificadas as informações com as que foram armazenadas no sistema, e é dado ou não acesso ao usuário [VARENHORST, 2004].

Figura 5 – Exemplo de uma assinatura (VARENHROST, 2004)

Do ponto de vista da comparação com as demais assinaturas, o sistema possuiria uma limitação de usuários com assinatura que poderiam acessar o sistema por esse meio, sendo necessário dessa forma definir os limites do sistema [VARENHORST, 2004].

A assinatura até pode ser usada como único meio de identificação, porém, para manter a segurança do sistema não pode simplesmente autenticar um usuário como o utilizador registrado cuja assinatura seja muito simples, deve haver uma probabilidade e similaridade mínima a ser definida, para, dessa forma, evitar que um rabisco qualquer acabe identificando um usuário. As principais características para a autenticação pela assinatura nesse caso seriam a velocidade e a precisão com que a assinatura é feita, mas como verificar por essas métricas apenas tornaria computacionalmente complexo e demorado, a idéia do trabalho utiliza uma combinação de reconhecimento de desenho, velocidade e mapeamento de ponto [VARENHORST, 2004].

3.2.2. Métodos

O sistema criado deve permitir se fazer a distinção entre diferentes usuários, todavia, não somente isso baste, é necessário dizer, caso se

encontre, qual assinatura é mais semelhante a aquela inserida, e se, com determinada tolerância pode se dizer que é a mesma assinatura, dessa forma, a assinatura serviria de forma única, como usuário e como senha. No aspecto pelo número de diferentes autenticações a serem realizadas, já muito mais possibilidades de desenhos distintos num mapa do que a combinação de caracteres comuns, não contando ainda outros aspectos como precisão e velocidade [VARENHORST, 2004].

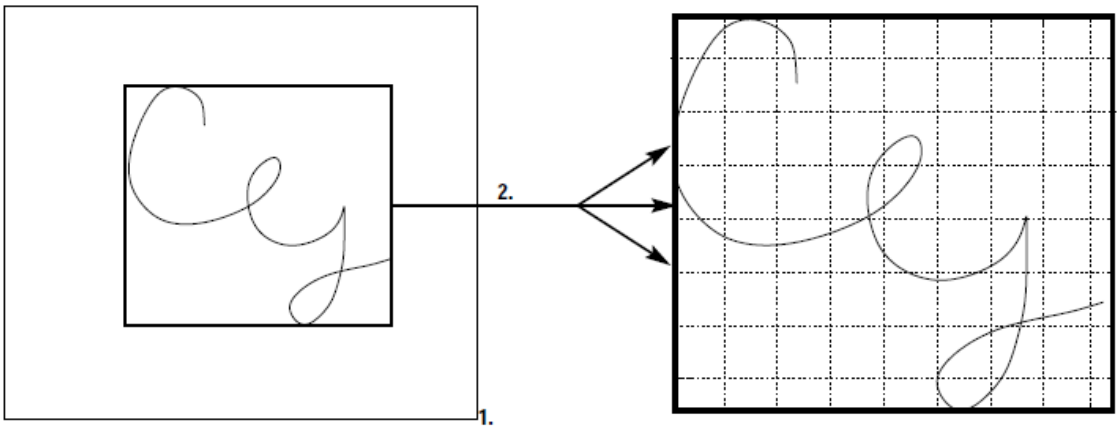
Após a análise da inserção de um usuário no sistema com sua assinatura escrita diversas vezes com pequena variação, e da inserção de assinaturas de diversos usuários no sistema, tendo dessa forma grandes variações, o trabalho apresentou três técnicas. Uma baseada na distribuição na grade, outro na velocidade instantânea, e outro na variação dos pontos através da grade. Esses três aspectos são analisados, e se espera que pelo menos dois deles possua alto grau de similaridade para então ser feita a autenticação [VARENHORST, 2004].

3.2.3. Assinatura

A assinatura pode ser estabelecida com um conjunto ordenado de pontos dispostos em uma grade tendo em adicional um registro do tempo do ponto a cada milissegundo a partir do primeiro ponto. O algoritmo faz então a ligação dos pontos obtidos dando a ilusão de um caminho pelo qual foi feito o rabisco. Fazendo o caminho, há uma fidelidade maior com a próxima assinatura, do que comparar apenas os pontos, já que o tempo pode mudar devido a qualquer fator externo [VARENHORST, 2004].

3.2.4. Distribuição na grade

Num sistema simples, é necessário um simples e direto método para a computação das comparações, para isso é feita uma definição de altos e baixos pontos, canonizando eles para a grade. Em seguida é feita várias amostras de uma mesma assinatura, e se obtém algo similar a Figura **6**. Cada assinatura tem o mesmo peso na junção, e nenhum pode dispersar mais de uma unidade na grade [VARENHORST, 2004].



1. Ler entrada do mouse
2. Escalar os pontos da assinatura para o grid
3. Comparar com as assinaturas armazenadas
   - Comparar distribuição no grid
   - Medir variação de pontos na distribuição no grid
   - Comparar a velocidade instantânea.
4. Se for confirmada a autenticidade, autenticar, caso contrário, repetir os testes para outra assinatura armazenada

Figura 6 – Algoritmo do *passdoodles.* Adaptado de (VARENHORST, 2004)

Em seguida, através dos dados obtidos é feito uma convolução gaussiana com variação de 1,6. A convolução, conforme pode ser visto na Figura 7, cria então sobre a imagem inúmeras variações que podem ser aceitas caso seja dado como entrada no sistema, qualquer assinatura que fique sobre o borrão é dado como aceita por esse teste [VARENHORST, 2004].

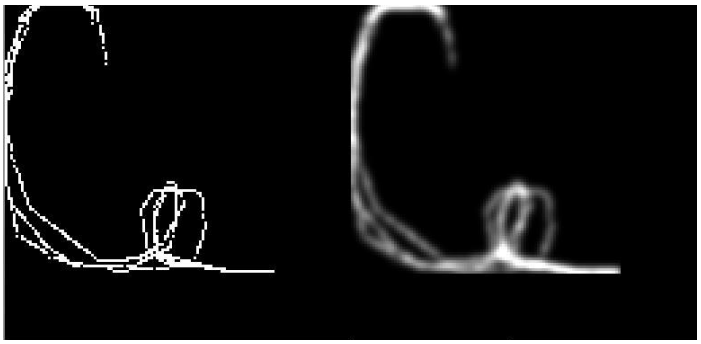


Figura 7 – Convolução Gaussiana (VARENHORST, 2004)

A assinatura a ser autenticada é processada com a distribuição na grade, cada ponto obtido da assinatura é usado para obter o ponto correspondente a distribuição da grade original. A dispersão da grade é então somada, sendo que cada ponto possui um grau de dispersão entre 0 e 1,

armazenado todos os dados da formação, se soma todos os casos, se for um valor baixo, a identidade do usuário é confirmada [VARENHORST, 2004].

3.2.5. Velocidade

A distribuição dos pontos com sua velocidade torna o mecanismo de identificação muito mais preciso, apenas o tempo em que cada ponto foi feito, já seria suficiente para fazer uma grande distinção entre diversos usuários, pois ainda que a forma seja semelhante, em determinado período da assinatura uma pessoa pode fazer mais rápida que outra [VARENHORST, 2004].

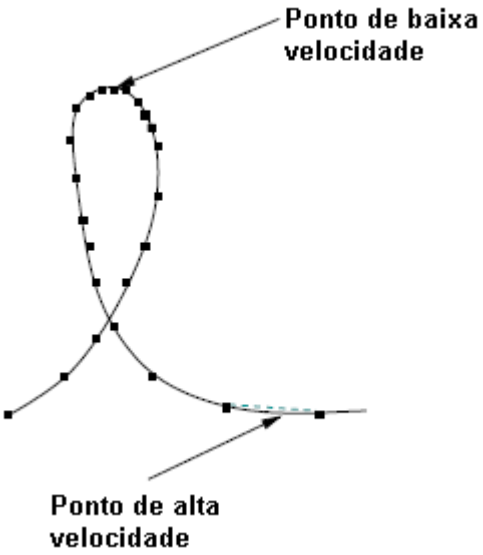


Figura 8 – Pontos de velocidade em milissegundos. Adaptado de (VARENHORST, 2004)

A técnica então consiste em dados os tempos em que cada ponto foi desenhado, é feito a comparação com o tempo do mesmo ponto que está armazenado em um vetor criado no momento em que foi cadastrada a assinatura. É obtida então a diferença entre o tempo dos pontos, se esse tempo extrapolar o limite inferior ou superior, a autenticação não é realizada. Uma assinatura, feita com um tempo idêntico, teria como resultado da diferença entre os pontos o valor 0 [VARENHORST, 2004].

3.2.6. Variação

Uma forma pode ter uma variação dos pontos baixa, todavia pode haver uma diferença na definição da forma que caracteriza uma diferença

fundamental entre assinaturas distintas. A ideia é distinguir duas assinaturas que parecem iguais quando a imagem está fora de foco, mas que são diferentes, observando a definição da forma da assinatura. Essa será importante para realizar a interpretação dos pontos, como aproximação de um lugar geométrico [VARENHORST, 2004].

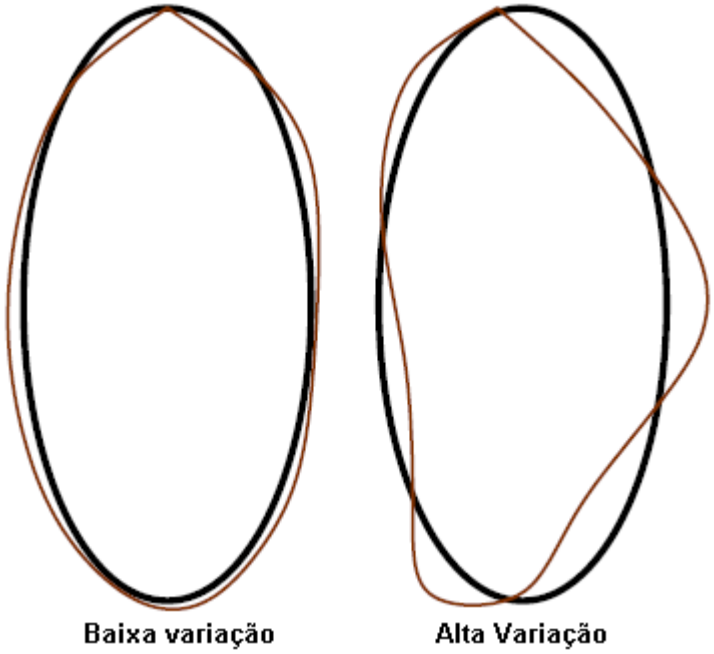


Figura 9 – Análise da variação (VARENHORST, 2004)

Esta etapa independe das etapas de variações anteriores, pois se estará realizando comparações com a forma como um todo, e não apenas com os pontos obtidos. A forma é obtida através da ligação dos pontos obtidos dois a dois ainda na etapa de aquisição da assinatura. Dessa forma, mesmo quando a imagem é autenticada no primeiro teste, pela sua forma estar contida no borrão que foi canonizado, a mesma imagem pode não ser autenticado nesta etapa caso haja muita diferença quanto à forma do desenho [VARENHORST, 2004].

3.2.7. Conclusão da Seção

O trabalho Passdoodle utiliza para o reconhecimento uma análise morfológica e de variação, bem como a velocidade que auxilia no processo de autenticação, mas que não se aplica a proposta deste trabalho. A análise de variação pode ser utilizado no processo decisório da forma, na parte de reconhecimento das formas. Porém a etapa de casamento de pontos com uma

grade faz com que o método fique restrito a uma forma inicial dada, dependendo inclusive do tamanho com o desenho foi feito.

3.3. *Sketching* de múltiplas fidelidades.

O trabalho *A Methodological Framework for Multi-Fidelity Sketching of User Interfaces* (COYETTE, 2007) busca facilitar o trabalho dos desenvolvedores de interfaces pro usuário, partindo do princípio que o melhor início do desenvolvimento de uma interface, começa com um esboço manual. A idéia consiste em implementar uma semântica para o reconhecimento desse esboço, de forma que ele possa diretamente levar ao código de desenvolvimento da interface, utilizando pra isso conceitos de reconhecimento de gestos.

A multi-plataforma proposta SketchiXML é uma ferramenta que facilita o trabalho do designer e incentiva a sua criatividade, podendo ser aplicado em diferentes tipos de contexto. Serve de apoio também na área de prototipagem e de metodologia de desenvolvimento. O trabalho estudado optou por usar no desenvolvimento da sua plataforma a linguagem UsiXML(UsiXML, 2009) devido a grande amplitude de elementos de interface para a apresentação dos dados e a grande facilidade da eventual adição de novos elementos a linguagem [COYETTE, 2007].

3.3.1. Arquitetura

O SketchiXML possibilita a realização de várias tarefas em simultâneo afim de realizar a interpretação das formas espaciais e manipulá-las para diversos tipos de entrada, gerando ao final um arquivo na sintaxe do UsiXML, e deixando em aberto o sistema para que possam ser anexadas mais ferramentas de manipulação e auxílio. A arquitetura deste trabalho pode ser dividida em sua parte geral, a arquitetura do mecanismo de reconhecimento de formas e a arquitetura do mecanismo de interpretação de formas [COYETTE, 2007].

A idéia da arquitetura geral do sistema é tê-la de forma robusta de flexível, para isso, distribuindo as tarefas internas do sistema pra agentes

distintos, porém cooperantes no objetivo final. A divisão em agentes é que torna a arquitetura mais flexível e robusta, uma vez que cada parte do sistema denominada crítica é tratado por um conjunto de agentes cooperativos a fim de realizar a tarefa imposta conforme os requisitos solicitados [COYETTE, 2007].

A arquitetura global do Sketchi XML é baseada em uma combinação da arquitetura de visão do controlador e sistemas multi-agentes. Dessa forma a arquitetura busca dividir o sistema em camadas separadas de aplicação. Então, reúne-se um determinado conjunto de tarefas, como reconhecimento de formas e implementação, e repassa essas informações ao conjunto de agentes responsável por essa tarefa. Dessa forma, quanto ao conjunto de agentes, pode-se dividir em dois módulos: o módulo para reconhecer formas, gestos e caligrafia, e o módulo que realizará a interpretação das formas, que já tem como entrada as definições do módulo anterior [COYETTE, 2007].

O mecanismo do SketchiXML é baseado em cooperativa de agentes para realizar tarefas, nesse módulo são necessários quatro agentes, três deles responsáveis por moldar as formas primitivas, caligrafia, e gestos, e um quarto agente que é para obter a saída dos três agentes e organizá-las [COYETTE, 2007].

Quando um agente mediador recebe um novo início desenho para reconhecer é emitida a informação para procurar quem seria o agente mais apropriado para realizar a tarefa com o desenho, por exemplo no caso da caneta do desenho, ser mantida pressionada, será indicado a um reconhecedor de gestos, caso contrário, seria enviado a um reconhecedor de formas. Nesse caso, existem três conjuntos de técnicas para realizar o reconhecimento: reconhecimento de formas, reconhecimento de gestos, e por último reconhecimento de escrita [COYETTE, 2007].

Reconhecimento de formas.

O SketchiXML utiliza para auxiliar no reconhecimento uma biblioteca chamada ‘Cali’(FONSECA, PIMENTEL e JORGE, 2002) que faz o reconhecimento baseado em algoritmos da lógica *Fuzzy*, e permite reconhecer múltiplos desenhos de formas geométricas e alguns comandos de gestos.

A biblioteca utilizada na aplicação SketchiXML usa adjacência temporal e propriedades de geometria de figuras para realizar o reconhecimento para um vocabulário simples de formas de desenho geométrico em diferentes estilos de linha como pode ser visto a esquerda da Figura 10. O reconhecimento da forma não é influenciado pela rotação ou pela escala da forma desenhada, levando a altas taxas de reconhecimento [COYETTE, 2007].

Também é possível atribuir uma semântica ao desenvolvimento contínuo pelo desenho, através do reconhecimento de gestos como pode ser visto a direita da Figura 10. Um desenho feito pode na verdade, ser uma ação a ser tomada pelo usuário, que efetuará modificações na próxima interface que está sendo desenvolvida por esboços [COYETTE, 2007].

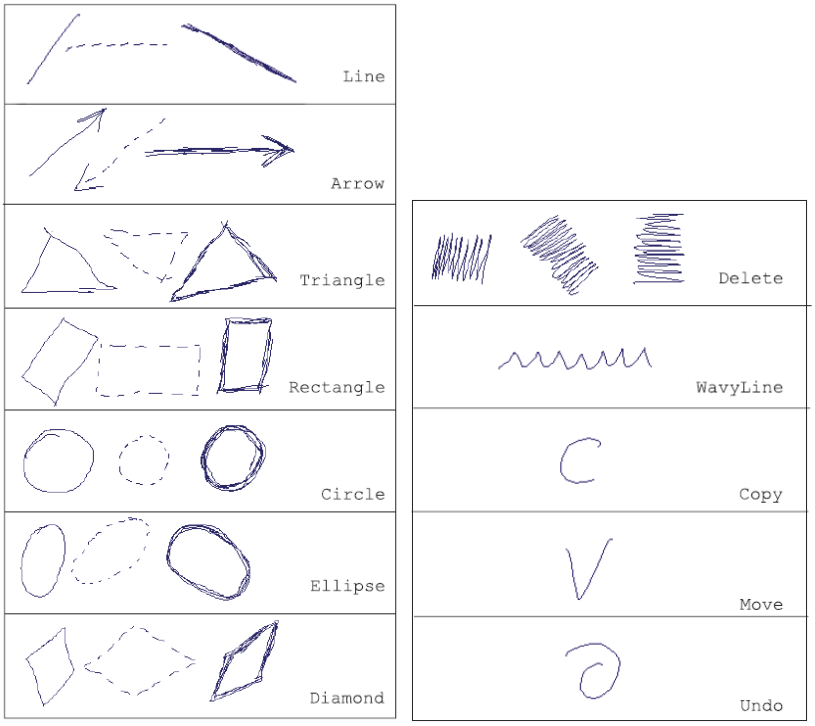


Figura 10 – Elementos de reconhecimento de forma e semântico (COYETTE, 2007).

Reconhecimento de gestos.

O princípio da idéia do reconhecimento de gestos se baseia na segmentação do traçado do gesto em segmentos de retas alinhados, para a interpretação ser feita em cima destes. Para o reconhecimento de um gesto, é levado em consideração 3 parâmetros para o sistema, (x, y, p) que definem respectivamente, a coordenada x e y da caneta, e p a pressão que ela está

exercendo sobre a superfície, que no caso do sistema analisado, é considerado como 0 pra quando a caneta não está tocando a superfície e 1 quando está tocando a superfície [COYETTE, 2007].

Dada a idéia, é possível dizer que um desenho plano pode ser definido como uma grade onde cada ponto da imagem é um nó dessa grade. Cada nó pode possuir oito nós adjacentes, e a cada par de nós adjacentes, existem oito possibilidades de direção entre eles. Assim, se numerarmos as adjacências de um nó, podemos estabelecer um padrão de legenda para cada uma das direções do espaço plano. O reconhecimento da sequência de nós adjacentes é a base para o reconhecimento do desenho, pelo tipo de representação curta e fácil interpretação. Para pontos esparsos de uma amostragem o ponto (x_i, y_i) onde $i \in \{1, \dots, n)$ para n pontos dados na entrada, o nó da grade que corresponderá ao ponto dado é calculado pela seguinte equação [COYETTE, 2007].

$$qx_i = arredondado(x_i / w)$$
$$qy_i = arredondado(y_i / w)$$

Onde w seria o tamanho da grade proposto. Se mais de um ponto acaba atribuindo o mesmo nó na grade, esse ponto aparecerá somente uma vez na sequência final. Dessa forma, alguns pontos acabam sendo suprimidos na definição. Dado uma vez montado a grade basta estabelecer a relação de adjacência entre os nós, de forma que todos os nós fiquem ligados [COYETTE, 2007].

O espaço entre o desenho de uma parte da figura pra outra, denominado *gap*, é utilizado como forma de reconhecimento, assumindo que a caneta está tomando uma determinada direção, dado um ponto atual (qx_j, qy_j) se existe uma mudança de direção para se chegar ao ponto $(qx_j + 1, qy_j + 1)$ o ângulo entre a direção que estava sendo tomada e a nova direção é usada como critério para a definição da forma. Dessa forma, pode-se definir momentos na definição dos gestos, traçados antes da mudança de sentido, e depois, para facilitar a identificação [COYETTE, 2007].

É possível, através do traçado, estabelecer uma cadeia de caracteres que o define através das direções conforme exibe a Figura 11, já que estas podem estar numeradas, onde cada direção constatada no desenho é passada para a cadeia. A partir dessa sequência é possível estabelecer uma diferença entre a cadeia obtida e a esperada para efetuar o reconhecimento. É dito como distância, o número de modificações necessárias para tornar a cadeia obtida na desejada. As modificações aqui ditas são inserção, remoção, ou alteração de algum elemento, pode ser atribuídos pesos a cada uma dessas alterações para aumentar mais ou menos o valor da diferença entre as cadeias [COYETTE, 2007].

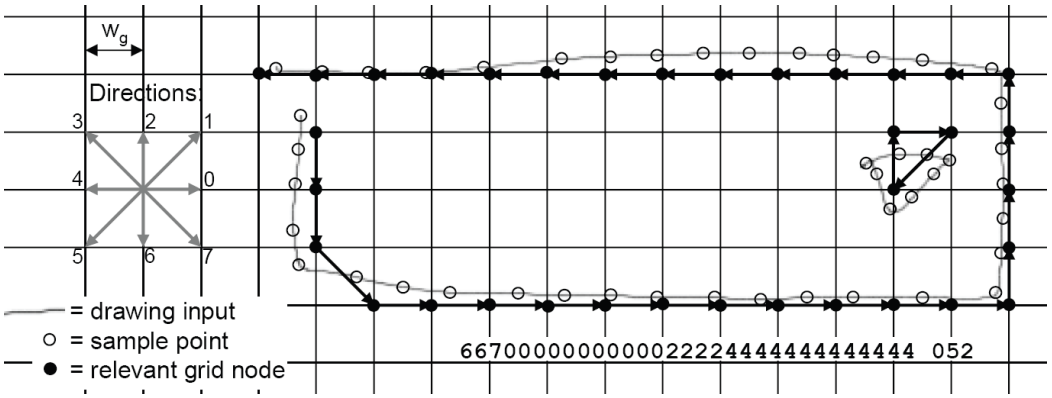


Figura 11 – Exemplo de um desenho de entrada mapeado em um grade (COYETTE, 2007)

Construção da gramática final

O processo de reconhecimento de forma é baseado na interpretação realizada por uma série de agentes, o número de agentes envolvidos em cada processo é variável. Todavia, existe um agente mediador que fará toda a interpretação das informações dos demais agentes envolvidos no processo. Novos agentes podem ser construídos ou destruídos, conforme a execução da gramática e sua expansão [COYETTE, 2007].

Ao início o agente mediador encaminha uma forma a todos os agentes que ele sabe que podem tratar com aquele tipo de forma, e cada um desses agentes fazer os seus testes para tentar reconhecer aquela forma, e caso haja um reconhecimento de forma incompleta, encaminham essa sub-parte a um outro agente por ele mesmo criado. Ao final, o mediador avalia conforme o

retorno do agente se a combinação dada por ele é válida ou não comparando com os níveis de fidelidades esperados pela aplicação [COYETTE, 2007].

Ao final, é gerada uma gramática que condiz com a semântica obtida ao longo do processo, que no caso da aplicação analisada é uma saída XML que será usado na construção de uma interface para Web.

3.3.2. Conclusão da Seção

O trabalho realiza a análise com os pontos atribuídos em uma grade, fazendo a distinção entra os tamanhos desenhados para a geração da interface, mas ainda assim, efetua o reconhecimento, independente do tamanho.

Possui um alto grau de fidelidade quanto a representação da forma esperada através da análise morfológica e da geração da cadeia direcional. Utiliza o reconhecimento de gestos como parte da criação e da interação do sistema, atribuindo a cada forma, uma operação a ser realizada. Gera a gramática final adquirida por meio de um documento XML que pode ser interpretado por outros sistemas.

3.4. Análise Hierárquica Aplicada a Detecção de Formas

O trabalho *Hierarchical Morphological Analysis for Generic Detection of Shapes in Grayscale Images* (SILVA e LOTUFO, 2006) apresenta um método de se obter as formas: linhas, retas, círculos e arcos de imagens em tons de cinza através da análise de uma árvore de componentes. De forma geral, realizar uma interpretação semântica da imagem, dividindo-a em regiões através do agrupamento de pixels, e organizando-as de forma hierárquica, para então poder realizar uma interpretação em cada parte da imagem de forma independente.

Desse modo, o trabalho propõe uma filtragem e detecção de objetos através da análise de sua árvore de informações, com o intuito de buscar a informação da forma representada na figura. A cada verificação de forma, se tem, além dos pontos que compõem a figura, uma tolerância mínima $(dmin)$ e

uma tolerância máxima$(dmax)$ dadas como parâmetro. Que represente o nível de fidelidade que se espera da figura com a forma a ser obtida.

3.4.1. Linhas

É dito ser uma linha qualquer componente conexa cuja distância entre os pixels e o contorno da linha não seja maior que a tolerância, e a distância interna entre cada pixel que gera a componente conexa não sejam maiores do que 1 [SILVA e LOTUFO, 2006].

3.4.2. Retas

Dados os intervalos mínimos e máximos que serão considerados como o nível de fidelidade da forma e dado dois pontos extremos de um componente conexo, pode-se dizer que existe uma reta com coeficiente angular dado por $(x_1 - x_2)/(y_1 - y_2)$ sendo que (x_1, y_1) e (x_2, y_2) são as coordenadas dos extremos do segmento respectivamente. Traçando uma reta com esse coeficiente ligando os dois extremos, pode se analisar a disperção dos demais pontos da reta traçada [SILVA e LOTUFO, 2006]..

Dessa forma, é feita uma análise, se $(x_1 - x_2) > (y_1 - y_2)$ então o termo y dos demais pontos da sequência são aumentados, caso contrário, os demais pontos x são aumentados. A cada incremento de ambos os lados, o valor do pixel pertencente a reta traçada é comparado com o pixel obtido, e é verificada a distância entre eles para todos os pixels do segmento da reta para os pixels gerados, se todos eles obtiverem uma distância entre $dmin$ e $dmax$ é dito que existe uma reta ligando os extremos dessa componente conexa, conforme a figura Figura 12 [SILVA e LOTUFO, 2006].

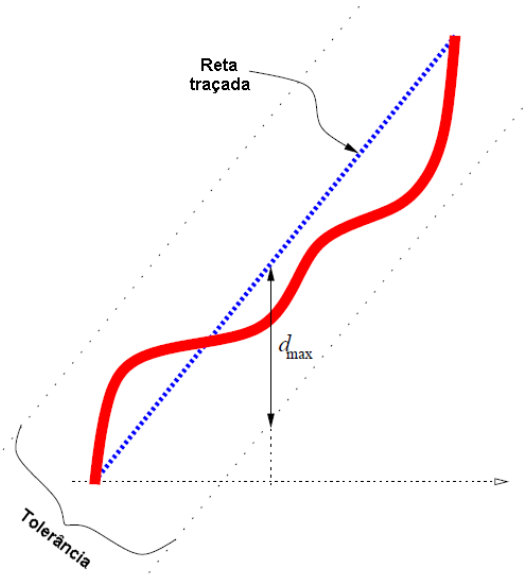


Figura 12 – Aproximação de uma reta por uma componente conexa. Adaptado de (SILVA e LOTUFO, 2006)

3.4.3. Círculos

A definição de um círculo já nos diz que um círculo é uma forma geométrica onde todos os pontos que compõem este têm a mesma distância de um ponto central. Dado um círculo como um componente conexo de vários pontos x_i que descrevem seu contorno, dizemos que a componente conexa gera um círculo se a distância entre o ponto do contorno x_i e o ponto central x_c então dentro de uma margem de tolerância $raio_{min} \leq dist(x_i, x_c) \leq raio_{max}$ para todo x entre 1 e a quantidade de elementos da borda, conforme a Figura 13 [SILVA e LOTUFO, 2006].

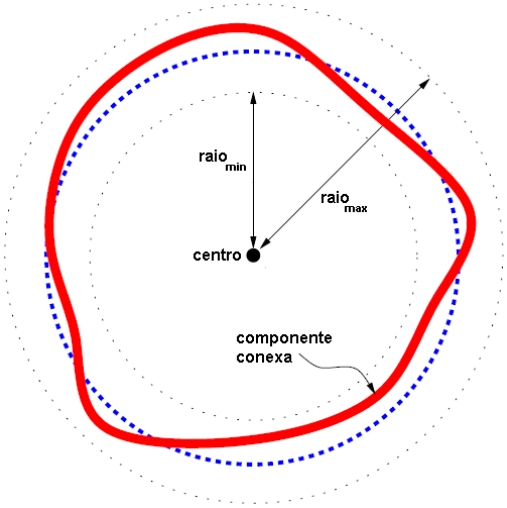


Figura 13 – Aproximação de um círculo por uma componente conexa. Adaptado de (SILVA e LOTUFO, 2006)

3.4.4. Arcos

Da mesma forma como no círculo, podemos definir se uma componente conexa pode ser aproximada de um arco. Para isso, deve-se encontrar um ponto x_c cujo este possua distâncias, onde dada a tolerância os pontos podem ser ditos equidistantes desse ponto central. Para isso, dado um ponto central x_c é pego um ponto da componente x_i, cuja a distância até o ponto x_c seja menor. Tendo esses dois pontos, é dito o ponto como novo centro y_c. Com isso, é traçada uma reta que passa por x_c e y_c de forma a buscar o centro global, então para cada ponto se partindo de y_c são verificadas as distâncias com os pontos que descrevem o contorno do arco, esperando que em algum ponto possa existir um ponto central que seja equidistante dada a tolerância $raio_{min}$ e $raio_{max}$ a todos os pontos que descrevem a borda. Se o ponto for encontrado, é obtido um arco aproximado tendo como o centro o ponto da reta obtido, essa definição está ilustrada na Figura 14 [SILVA e LOTUFO, 2006].

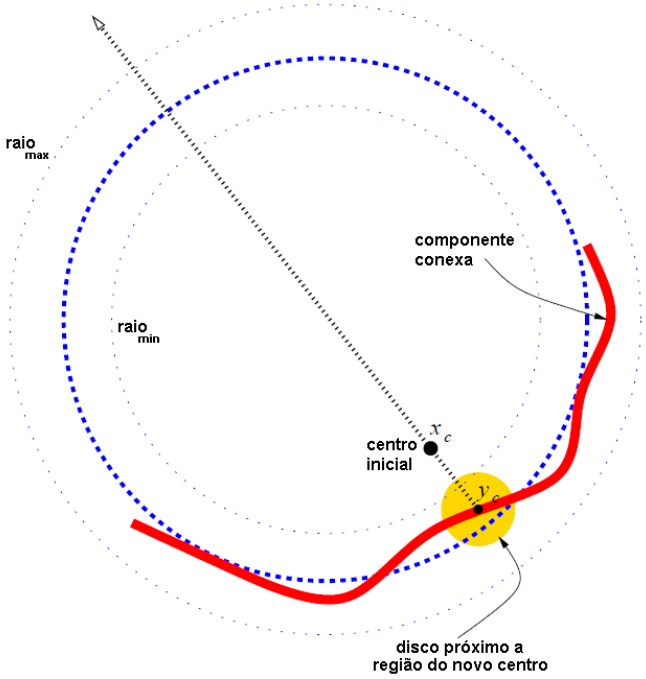


Figura 14 – Aproximação de um arco por uma componente conexa. Adaptado de (SILVA e LOTUFO, 2006)

3.4.5. Conclusão da Seção

O trabalho realiza a indenficação das figuras através de uma análise matemática de sua forma, assim, não existe distinção para a escala da figura, sendo que ela virá a ser reconhecida, independende do seu tamanho.

A medida da margem de tolerância das fórmulas de reconhecimento pode ser usada como parâmetro para flexibilizar o sistema para diversos tipos de aplicação.

Se faz necessário a implementação diferenciada para cada forma que se deseja obter, e a cada nova inserção de um objeto de reconhecimento, um estudo matemático de sua forma e comportamento espacial.

4. FORMATOS VETORIAIS

O trabalho tem a necessidade de estabelecer qual sintaxe será utilizada para representação das formas, dessa forma, é importante o estudo dos formatos de representação vetoriais. Desenhos vetoriais nada mais são do que a representação de uma imagem qualquer através de representações matemáticas da forma que a define. Em contrário a representação por bitmap, um formato vetorial não precisa descrever os *pixels* da imagem. Estes são interpretados de um arquivo vetorial e tem seus pixels gerados para a visualização. Assim sendo, nunca teria perda de qualidade quando se desejar mudar a escala da imagem [COELHO, 2004].

4.1. Arquivos Vetoriais

Os formatos PDF/PS/EPS criado pelo Adobe originalmente para representar arquivos que viriam a ser interpretados por impressoras, que possuem uma forma de armazenamento vetorial, sem representar pixel a pixel a imagem. Sendo usado na maior parte das vezes como um descritor das páginas a serem impressas, já que uma vez nessa forma, não é possível fazer a sua edição por pixels, ainda que existam mecanismos de realizar a sua edição por forma [ADOBE, 2009].

O formato FIG, fruto da ferramenta xFig, tem como seu formato nativo um arquivo somente texto, que faz a descrição das figuras vetoriais a serem desenhadas ou a incorporação do bitmap. E também permite que facilmente se possa editar no código os parâmetros que se deseja para criar uma figura [EBP, 2007].

Alguns formatos proprietários como CDR e o WMF possuem facilidades quando associados às ferramentas desenvolvidas para esse formato. Todavia, tanto a sintaxe do documento, quanto as ferramentas que trabalham com eles, são proprietárias, o que torna inviável o seu estudo para se realizar uma gravação nesse formato [MSDN, 2009 e COREL, 2009].

O SVG é um formato de arquivo vetorial que segue um padrão apoiado pela W3C, e tem uma sintaxe simples para a confecção de um arquivo no formato vetorial, e possui facilidades de especificação, e oferece suporte a

imagens *bitmap* e texto [W3C, 2009]. Por esses motivos ele foi escolhido como formato de saída para a aplicação a ser realizada.

4.2. SVG

O SVG foi lançado em 4 de setembro de 2001, e é uma linguagem de marcação assim como o XML, adotada como padrão pela W3C para a representação de imagens no formato vetorial, para dessa forma descrever gráficos bidimensionais e implementar aplicações gráficas de forma estática, dinâmica, ou animada. Essa linguagem é um padrão aberto e livre, e seu código pode ser facilmente interpretado e editado, interpretado pelo navegador Mozilla Firefox ou qualquer outra ferramenta de desenhos vetoriais que suporte o formato SVG, e editado por qualquer editor de textos, e este não suporta apenas descrições vetoriais, pode assumir imagens em bitmap ou até mesmo texto em sua especificação [W3C, 2009 e SVGBASICS, 2009].

4.2.1 Formas Básicas

Neste capítulo serão descritas apenas as formas que de alguma forma poderão ser utilizadas na aplicação final ou em sua análise. Dessa forma, as primitivas aqui serão representadas da forma como especificadas aqui ou de uma maneira similar utilizando outras primitivas.

Não há muito a explicar sobre as formas, retângulo e círculo. A forma retangular pode ser definida por dois parâmetros largura, e altura, que será a medida das linhas horizontais, e a medida das linhas verticais. Já o círculo pode ser definido apenas pelo tamanho de seu raio. Em ambos é necessário especificar a posição em que eles se encontram no espaço. E existem outros parâmetros para definir o *layout* da forma. Na figura Figura **15**, no item ‘a' pode se ver um exemplo de um retângulo definido em svg através do código:

```
<rect x="100" y="100" rx="50" ry="10" width="400" height="200"
fill="yellow" stroke="black" stroke-width="3" />
```

- rect – tag que define o elemento a ser desenhado.
- x, y – define a posição em relação o topo a esquerda.
- rx, ry – raios que definem o arredondamento dos cantos.

- width, height – definem a largura e a altura da forma.
- fill – define a cor a ser preenchido o interior da forma.
- stroke – define a cor da linha que contorna o desenho.
- stroke-width – define a espessura da linha que contorna o desenho.

Em seguida temos o exemplo de um círculo, note que alguns parâmetros são comuns independente da forma, na figura Figura 15, no item ‘b’ pode se ver um exemplo de um círculo definido em SVG através do código:

```
<circle cx="600" cy="200" r="100" fill="red" stroke="blue" stroke-width="10"/>
```

- circle – tag que define o elemento a ser desenhado.
- cx, cy – define a posição do centro do círculo.
- r – define o tamanho do raio
- rx, ry – definem o raio da elipse para o arredondamento dos cantos.

Assim como o círculo pode ser descrito se especificando um raio, uma elipse pode ser definida se especificando para ela dois raios, um em relação a cada eixo padrão definido. Um exemplo dessa especificação pode ser visto na Figura 15, no item ‘c’ é exibido uma elipse definida em SVG através do código:

```
<ellipse cx = "10" cy = "120" rx="200" ry="100" fill="none" stroke="blue" stroke-width="10"/>
```

- ellipse – tag que define o elemento a ser desenhado.
- rx – definem o raio da elipse em relação ao eixo x
- ry – define o raio da elipse em relação ao eixo y

A reta, todavia, tem uma notação um pouco diferente das outras formas geométricas é dado como parâmetro o ponto (x,y) onde ela se inicia, e o ponto (x,y) onde ela termina. O exemplo de sua exibição pode ser visto na Figura 15 item ‘d’ onde a imagem foi gerada no SVG através do código:

```
<line x1="320" y1="250" x2="10" y2="10" stroke-width="10" stroke="green"/>
```

- line – tag que define o elemento da reta.
- x1, y1 – Coordenadas do espaço em que se inicia o segmento.
- x2, y2 – Coordenadas do espaço em que se termina o segmento.

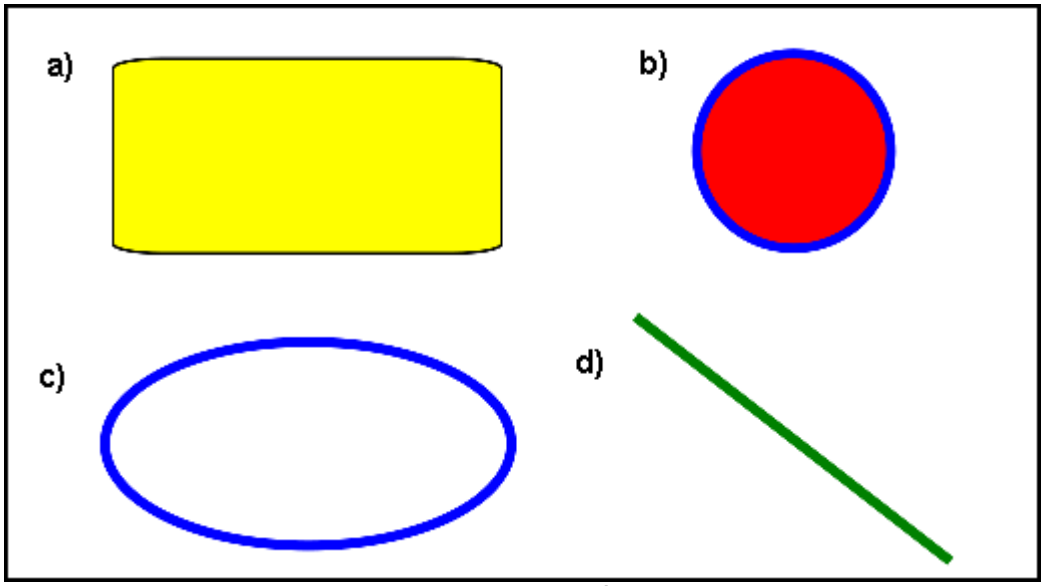


Figura 15 – Exemplos de primitivas básicas definidas no SVG

Um polígono aberto em SVG pode ser dito como a união de vários pontos, ou pode-se dizer de vários segmentos de reta. Internamente é feita a ligação do ponto *n* especificado com o ponto *n+1* de forma que gera um polígono aberto unindo todos os pontos especificados no atributo *points* da tag *polyline*, um exemplo de um polígono aberto definido em SVG pode ser visto na na parte inferior da Figura 16.

```
<polyline fill="none" stroke="blue" stroke-width="10" points="0,375
100,375 100,325 200,325 200,375 300,375 300,250 400,250 400,375
500,375 500,175 600,175 600,375 700,375"/>
```

- polyline – a tag que define a forma para um polígono aberto.
- fill – coloca preenchimento entre as linhas.
- points – define a sequência de vértices do polígono aberto.

Um polígono fechado no SVG é definido igualmente a um polígono aberto, só que realiza a junção do último vértice com o primeiro automaticamente, é equivalente a se colocar o primeiro vértice novamente no conjunto de pontos. Um polígono fechado pode ser visto na parte superior da Figura 16.

```
<polygon fill="red" stroke="blue" stroke-width="10" points="350,75
379,161 469,161 397,215 423,301 350,250 277,301 303,215 231,161
321,161" />
```

- polygon – a tag que define a forma para um polígono fechado.
- fill – preenche todo o conteúdo do polígono

- points – define a sequência de vértices do polígono fechado, faz a ligação.

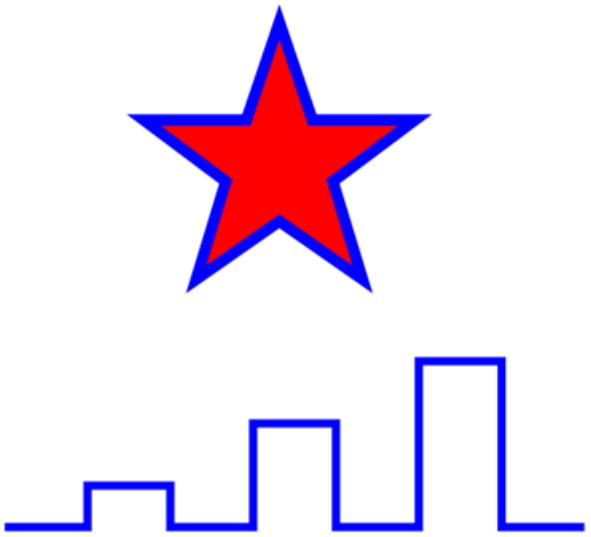


Figura 16 – Exemplo de polígonos definidos no SVG

4.2.2. Transformações Simples

As transformações podem ser usadas na interpretação de pontos quando se precisa realizar o casamento de algum padrão realizando uma sobreposição dos elementos, ou quando se deseja situar os pontos gerados o mais próximo possível de como era na imagem original.

As transformações mais comuns, que com eles se pode fazer qualquer coisa com uma imagem é a rotação, a translação e a escala. Que são realizadas dentro do sistema de coordenadas definido. A translação que pode ser vista na Figura 17 permite mover um objeto dentro do plano definido em relação ao eixo x ou ao eixo y, sua sintaxe é na forma de um atributo dentro da tag.

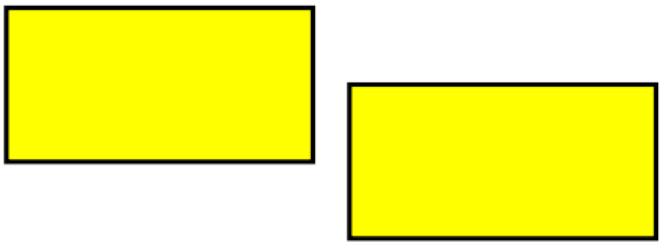


Figura original – Figura transladada em y
Figura 17 – Exemplo da operação de translação

```
<g transform="translate(50,50)">...</g>
```

Outra transformação que nesse caso modifica a forma como a imagem está sendo vista e não somente sua posição é a rotação, que pode ser vista na Figura 18, que no caso recebe como parâmetro os graus em rotação que serão dados em torno de z.

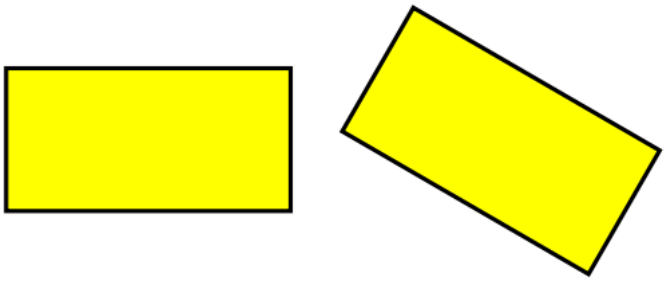


Figura original – Figura rotacionada 30 graus em z
Figura 18 – Exemplo de operação de rotação

```
<g transform="rotate(30)">...</g>
```

A outra transformação é a escala que muda o tamanho da imagem dentro de toda sua extensão igualmente é a escala, e pode ser vista na Figura 19. Recebe como parâmetro apenas a proporção da imagem que se quer representar.

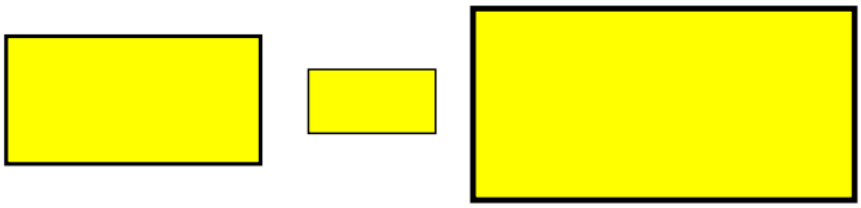


Figura original – Figura com escala em 0.5 – Figura com escala em 1.5
Figura 19 – Exemplo da operação de escala

```
<g transform="scale(1.5)">
```

Vale ressaltar que o atributo *transform* pode ser usado em outras tags como por exemplo as formas básicas do SVG.

4.2.3. Caminhos

Quando para gerar a forma for necessário a composição de diversos elementos, como retas e curvas, será necessário a aplicação dos

caminhos(*paths*) na especificação SVG. Onde com apenas uma tag pode se definir todo o conjunto de componentes.

Um caminho em um documento SVG pode especificar qualquer uma das outras formar que são descritas pelas outras primitivas, todavia, os caminhos(*paths*) possuem uma sintaxe mais complexa, parecida com qualquer outra linguagem de programação, esses comandos a serem desenhados, ficam dentro de um atribudo da tag *path* denominado atributo *d* e dentro dele é dado todos os comandos a serem desenhados, os seguintes operadores são utilizados na descrição dos comandos:

- M – realiza um movimento no objeto de desenho, ou seja, se movimenta de um ponto a outro e nesse trajeto, não deixa nenhum desenho no caminho, tem como parâmetros o ponto x e y que é para onde será realizado o movimento, ou seja, o ponto de destino.
- m – também realiza um movimento no objeto de desenho, todavia esse movimento é reativo ao ponto atual da figura, ou seja, o x e o y dados como parâmetros serão adicionados as coordenadas atuais.
- L – desenha uma linha que vai do ponto atual da figura até o ponto x, y dados como parâmetro, assuma o ponto x, y como novo ponto atual caracterizando juntamente um movimento.
- l – assim como o atributo L, o l faz uma linha entre o ponto atual e o ponto dado, todavia essa linha é feira de acordo com os parâmetros de forma relativa ao ponto atual, adicionando aos parâmetros as coordenadas atuais do caminho.
- H ou h – equivalente ao L(ou l), todavia só recebe como parâmetro o movimento em x, mantendo o y igual ao atual, o identificador maiúsculo representa a operação de forma absoluta, quanto o identificador minúsculo represente de forma relativa ao ponto anterior.
- V ou v – equivalente ao L(ou l), entrando em oposição ao H(ou h), recebe como parâmetro apenas a coordenada y.

O atributo *d* suporta ainda formas mais complexas de desenho, como curvas e arcos, que serão explicados em um tópico a parte. Sendo os presentes aqui, apenas as operações básicas. A Figura 20 ilustra um exemplo para os operadores mencionados para o seguinte código.

```
<path d="M 0 100 h 100 v 100 h -100 Z M 110 100 L 200 200 m 0 -50 L
110 50" fill="red" stroke="blue" stroke-width="3" />
```

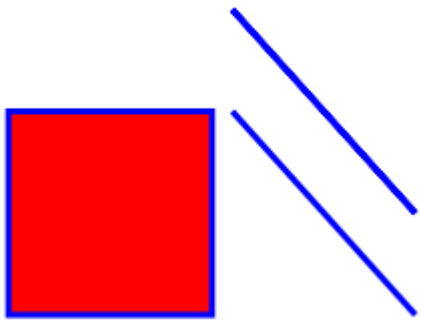


Figura 20 – Exemplo de um caminho(path) desenhado em SVG

O SVG suporta a definição de três tipos de curvas. As curvas cúbicas de Bezier, as curvas quadráticas de Bezier, e as curvas elípticas. As curvas Bezier cúbicas têm dois pontos de controle, enquanto as quadradas possuem apenas um ponto de controle, essa diferença é esperada quanto ao número de parâmetros que existe cada um dos atributos. Porém existem mecanismos para facilitar a especificação dos parâmetros, podendo omitir alguns deles. A parte de arco-elipse possui a sintaxe mais complexa, contudo uma maior quantidade de recursos.

Curvas Bezier Cúbicas

Uma curva Bezier cúbica, utiliza dentro da notação do caminho(*path)* a função C que recebe como parâmetro os dois pontos de controle da curva e o ponto onde termina a curva que será por consequência o novo ponto atual do desenho. Outra função semelhante é tida com a função S, todavia ela dispensa a mensuração das coordenadas do primeiro parâmetro de controle, ela assume esse parâmetro como sendo o último parâmetro da função C ou S usado anteriormente, caso não tenha nenhum, é assumido que os dois pontos de controle são por sua vez, coincidentes nessa especificação. A Figura 21 exibe alguns exemplos da aplicação das curvas C e S. Vale ressaltar que o C e o S podem também serem utilizados de forma minúscula e todos seus valores representados de forma relativa ao ponto atual.

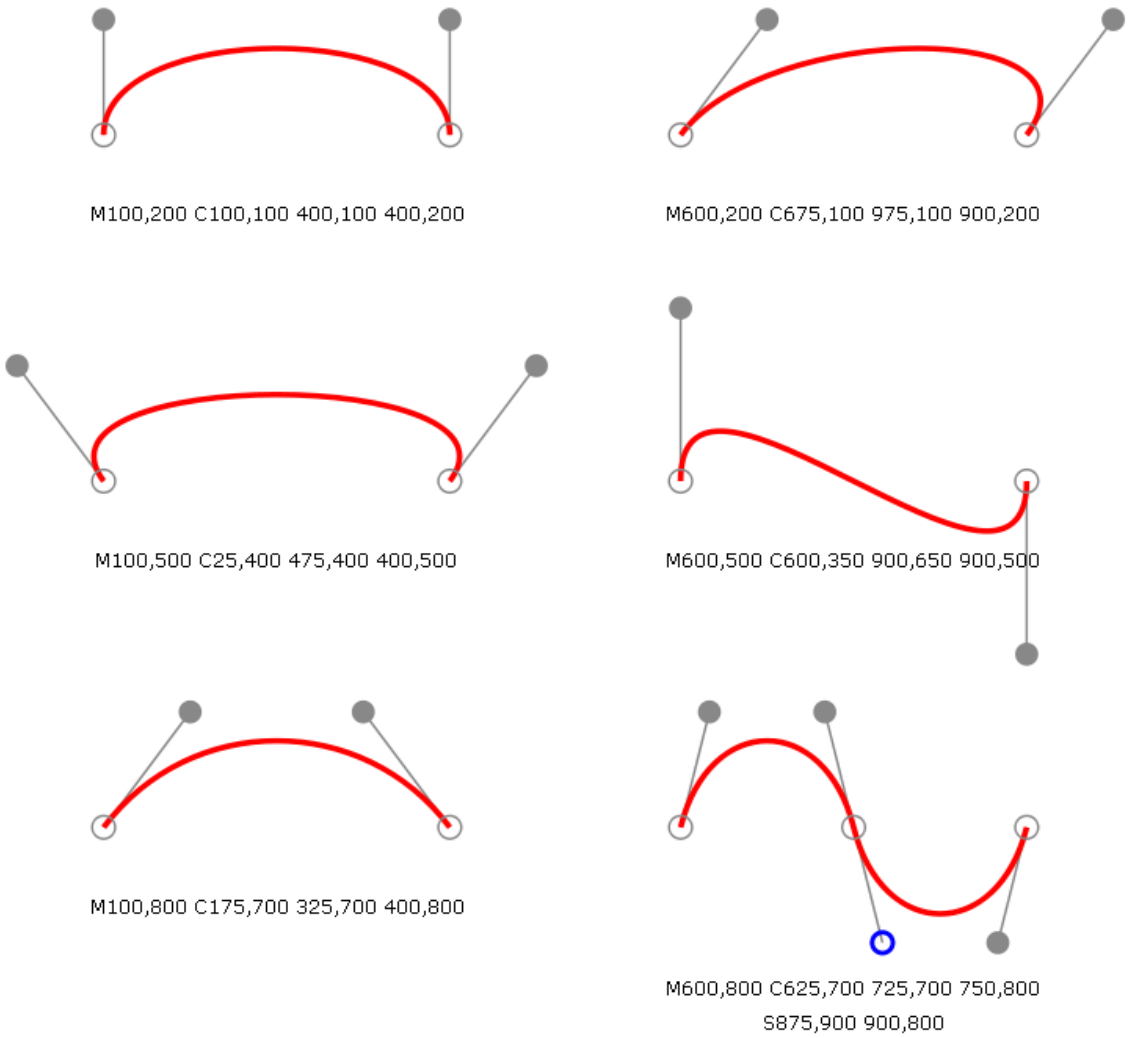


Figura 21 – Exemplos de curvas de Bezier para determinados parâmetros (SVGBASICS, 2009)

Curvas Bezier Quadráticas

Outra forma de curva possível no SVG é a curva de *Bezier* quadrática que pode ser definida se utilizando de apenas um ponto de controle, os parâmetros que o operador Q(ou q, de forma relativa) são, o ponto x,y que define a coordenada do ponto de controle, e o ponto x,y que define o final da curva. Um exemplo de uma curva Bezier descrita para o código abaixo pode ser visto na Figura 22.

```
<path d="M200,300 Q400,50 600,300 T1000,300"
fill="none" stroke="red" stroke-width="5"  />
<g fill="black" >
  <circle cx="200" cy="300" r="10"/>
  <circle cx="600" cy="300" r="10"/>
  <circle cx="1000" cy="300" r="10"/>
</g>
```

```
<g fill="gray" >
  <circle cx="400" cy="50" r="10"/>
  <circle cx="800" cy="550" r="10"/>
</g>
<path d="M200,300 L400,50 L600,300
         L800,550 L1000,300"
fill="none" stroke="gray" stroke-width="2" />
```

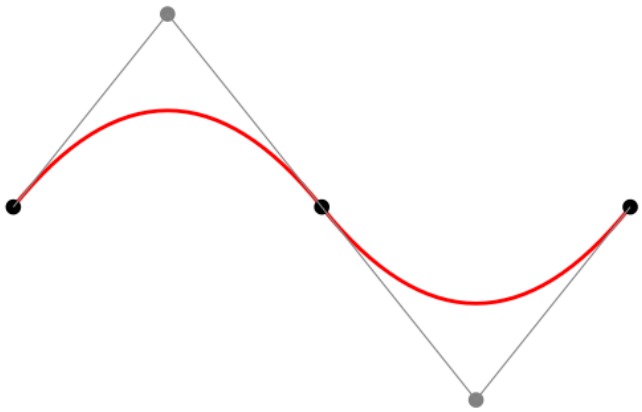


Figura 22 – Exemplo de uma curva de Bezier quadrática

Da mesma forma como nas curvas cúbicas de *Bezier*, nas curvas quadráticas também existe um modo para não precisar se especificar o ponto de controle, utilizando ao invés de Q, o identificador T dispensa o atributo, e assume como ponto de controle, o ponto espelhado em relação ao anterior, caso não exista um anterior, assume o ponto corrente do desenho como controle.

Curva Arco Elipse

A última notação de curva é a curva elíptica. Por mais que pareça complexo traçar o arco elíptico passando exatamente pelos dois pontos definidos, o SVG dispensa que isso seja especificado no código. Fica assim como parâmetros para a definição da curva elíptica o raio em x e em y para definir o tamanho da elipse, um parâmetro que especifica a rotação em z, duas flags de controle e como último parâmetro tem-se o ponto final em que o arco termina, dado que o inicial é dito como o ponto corrente. Um exemplo de um caminho que faz uso de arcos elípticos pode ser visto na Figura 23, que representa a forma descrita pelo código:

```
<path d="M600,350 l 50,-25 a25,25 -30 0,1 50,-25 l 50,-25 a25,50 -30
0,1 50,-25 l 50,-25 a25,75 -30 0,1 50,-25 l 50,-25 a25,100 -30 0,1
50,-25 l 50,-25" fill="none" stroke="red" stroke-width="5"/>
```

Figura 23 – Exemplo de um caminho com curvas elípticas

As flags de controle, dizem por como se tratar de uma elipse, pode-se definir a escolha entre, o arco menor que passa por esses pontos, ou pelo arco maior, através de uma flag(*large-arc-flag*) que recebe 0 caso seja o arco menor, ou 1 caso se queira o arco maior. Pode ser feito a escolha também quanto a orientação do laço escolhido, que pode ser para a esquerda ou para a direita, nesse caso a flag(*sweep-flag*), nesse caso, deve-se enviar 0 caso seja para a esquerda, e 1 caso seja para a direita, um exemplo do esquema de flags pode ser visto na Figura 24.

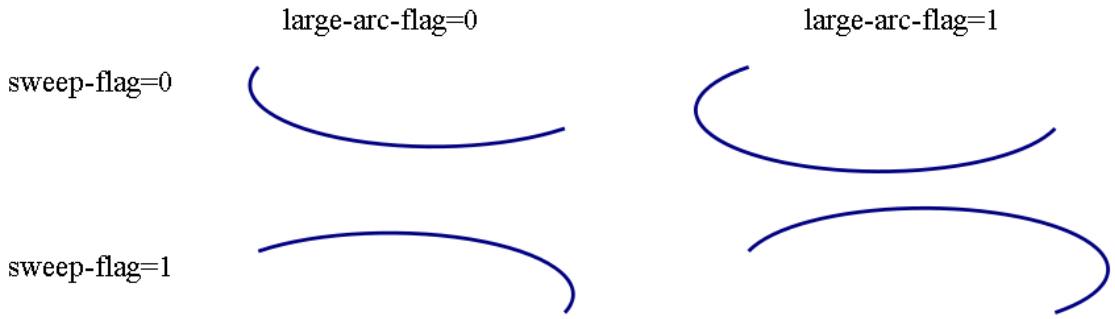


Figura 24 – Exemplo do uso das flags de controle do arco elipse

5. PROJETO DO SISTEMA

O projeto do sistema visa como entrada uma sequência de pontos provindos de um arquivo externo, a serem interpretados pelo sistema. Após a leitura e interpretação feita com base em um algoritmo elaborado com base em partes dos trabalhos relacionados.

A identificação de formas por ser obtida através de diferentes métodos, como por exemplo, o casamento de padrões, muito utilizado para o reconhecimento de caracteres, ou formas de definição complexas, se esperando uma imagem com um grau de similaridade mínina para os pontos que definem a figura, dessa forma dependendo da escalada da imagem. A abordagem utilizada por esse trabalho utiliza métodos matemáticos e uma dada tolerância utilizada sobre a fórmula que define a forma a ser obtida, independendo assim da escala. Todo o sistema executa em cima de uma margem de tolerância que será recebida como parâmetro, que pode ser definida conforme a necessidade de fidelidade que o sistema possua.

A saída, tendo visto a análise dos formalismos em formato vetorial, optou-se pelo padrão SVG apoiado pela W3C, podendo assim a saída do sistema ser reconhecida e já em seguida visualizada, sem que pra isso, precise de outro agente intermediador.

5.1. Linguagem

A linguagem escolhida para a implementação da aplicação foi à linguagem C. Como a aplicação receberá de entrada um arquivo de texto que representa o conjunto de pontos, e a saída final será um documento XML na sintaxe SVG, e esse tipo de documento também pode ser representado por texto puro, não se faz o uso de bibliotecas auxiliares quanto ao seu desenvolvimento. Dessa forma, como qualquer linguagem trabalha com texto puro, a linguagem C por ser a linguagem mais leve, com rápido processamento e compilada, se mostrou bastante adequada para o contexto desta aplicação.

5.2. Algoritmo

Elaborou-se neste projeto um algoritmo utilizando a abordagem geométrica levantada na fundamentação teórica, e os pontos positivos de cada trabalho relacionado, se preocupando com o isolamento de cada etapa do trabalho para produzir um código reusável (capaz de se reutilizar o mesmo código em outra aplicação) e destacadamente estruturado. O algoritmo foi dividido em 5 etapas conforme destaca a Figura 25.

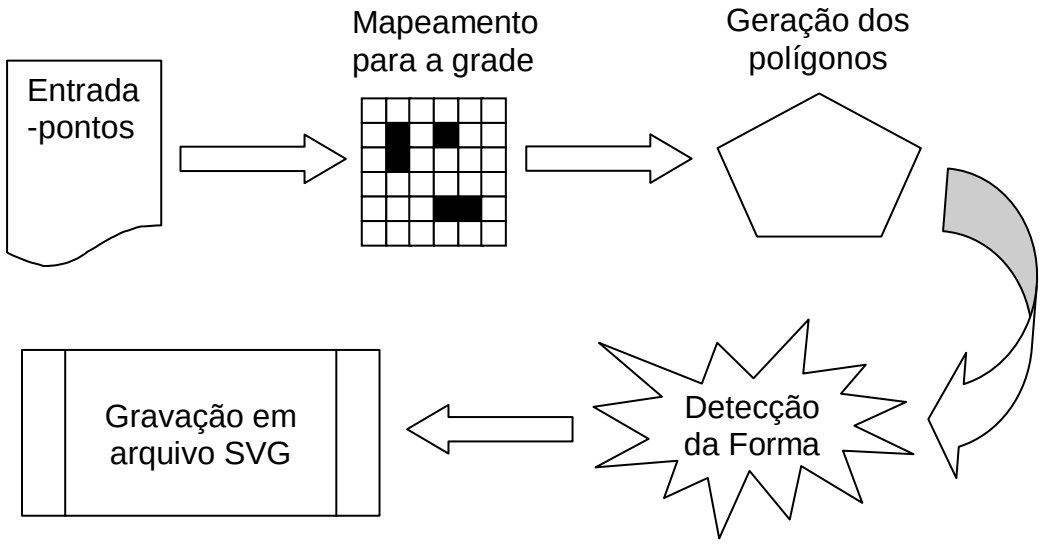


Figura 25 – Principais etapas do projeto do sistema

5.2.1. Entrada do sistema

Ao início do algoritmo, é dado entrada no sistema o conjunto de pontos obtidos de um arquivo de texto, e uma parâmetro de tolerância que é usado para se fazer a escolha do tamanho da grade a ser usada para o mapeamento, a escolha do tamanho da grade influencia na conectividade dos componentes da imagem, identificando se são um mesmo componente, ou componentes separados. Caso sejam separados, é obtida a forma para ambos de modo independente, onde podem resultar em figuras distintas que, todavia, serão representadas na mesma imagem final.

5.2.2. Tratamento Inicial

Dada a entrada inicial, foi necessária a discretização desses valores em uma grade, de forma a possibilitar a utilização de algoritmos ponto a ponto, e também descartar os pontos que estão muito próximos. Esse mapeamento poderia ser feito verificando quadrante a quadrante a adjacência do ponto dado, com o quadrante atual da grade, todavia, o algoritmo do *Sketchi* (COYETTE, 2007) apresenta um algoritmo eficiente pra esta tarefa, onde em tempo linear se pode atribuir um ponto a uma área da grade, dados assim pela fórmula.

$$qx_i = arredondado(x_i / w)$$

$$qy_i = arredondado(y_i / w)$$

Sendo w o tamanho da grade e i a ordem do ponto que está sendo lido para o caso do novo algoritmo. Pode ocorrer a supressão de pontos próximos mapeados em um mesmo quadrante da grade se tornando assim um ponto apenas. Um exemplo do mapeamento de pontos para uma grade, pode ser visualizado na Figura 26.

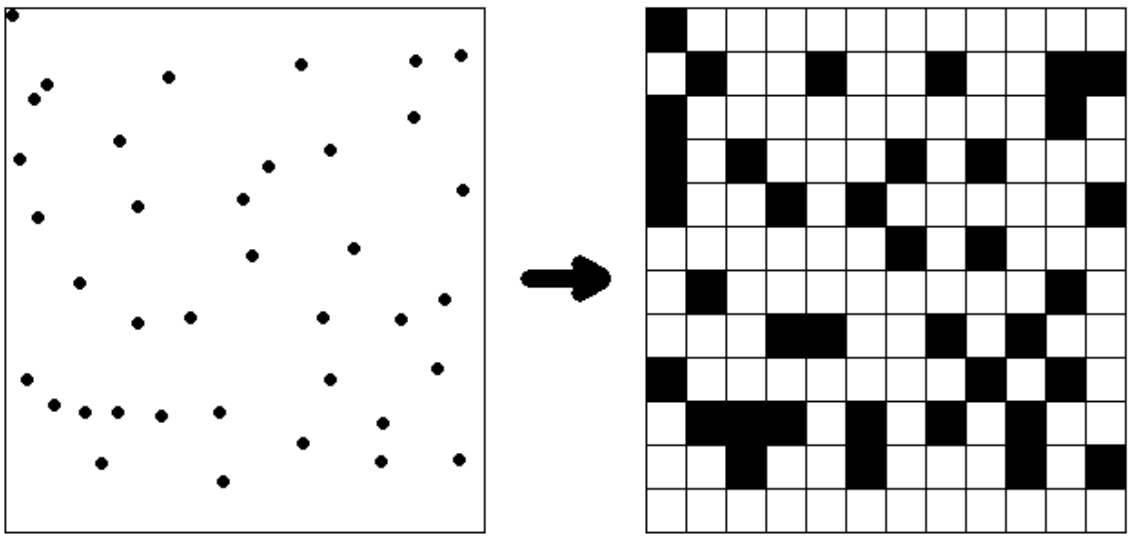


Figura 26 – Mapeamento de pontos para a grade.

5.2.3. Geração dos polígonos

Após o mapeamento dos pontos, tem-se como descrição algo similar a uma imagem binária, onde existiriam quadrantes da grade com cor, ou sua

ausência, sendo o reconhecimento feito apenas sobre os pixels com cor, possibilitando dessa forma, aplicar o algoritmo utilizado no trabalho Potrace(SELINGER, 2003). O algoritmo em questão utiliza as diversas componentes conexas da figura, dessa forma, se fosse dado como entrada a imagem binária vista à direita da Figura 26, poderia se obter diversas componentes conexas. Um exemplo de uma componente conexa obtida pode ser visto na Figura 27, onde com base no algoritmo do potrace para a geração do caminho, deve-se deve seguir sempre o caminho que mantém um pixel preenchido a direita.

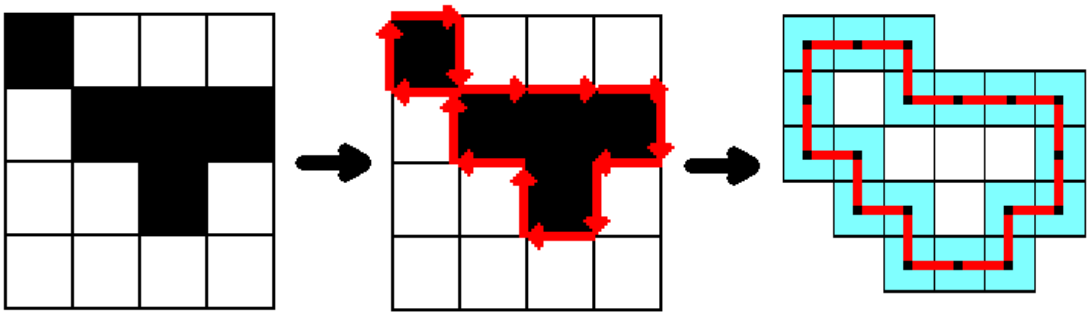


Figura 27 – Geração da componente conexa

Aplicando o algoritmo Potrace(SELINGER, 2003), cuja descrição da formação de um caminho encontra-se na Seção 3.1.1 é retornado um caminho que percorre a borda de um agrupamento de pontos. Dado este caminho é possível simplificar parte dele em segmentos de reta, gerando um polígono que pode ser interpretado neste projeto como uma componente conexa, conforme descrito aqui em 3.1.2.

5.2.4. Estabelecimento das Formas

Como ao final do algoritmo do Potrace(SELINGER, 2003), é representada uma forma através de curvas, fica sendo mais similar ao trabalho a ser desenvolvido, contudo, ainda é interessante, que se possa interpretar outras formas geométricas, como círculos, retângulos, e demais polígonos quaisquer.

Os polígonos podem ser facilmente interpretados a partir de múltiplos segmentos de reta, o polígono está praticamente pronto quando se assume

uma componente conexa, basta saber se existe algum tipo de curva nele, descaracterizando um polígono simples, e analisando se ele está fechado, ou não. Outras formas também podem ser reduzidas através da componente conexa gerada pelo Potrace, são essas descritas no trabalho de Detecção Genérica de Formas(SILVA e LOTUFO, 2006), porém, é interessante se estabelecer uma ordem de decições sobre uma componente dada, essa pode ser vista na Figura 28.

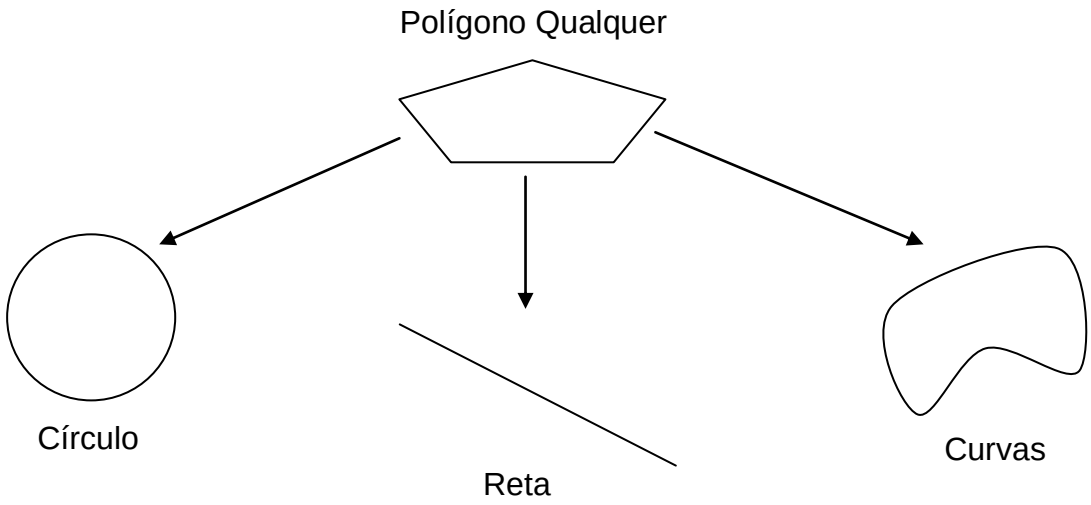


Figura 28 – Decisão entre as formas

Ainda que o algoritmo de detecção de formas já trabalhe com uma margem para aceitar se determinada sequência representa ou não uma forma, pode ser interessante nesse caso, ao invés de utilizar um algoritmo para descarte das formas, fazer uma análise de variação como é feito no trabalho Passdoodles(VARENHORST, 2004) apresentado aqui na Seção 3.2.6. Todavia análises como de velocidade ou tempo em que cada ponto foi dado não poderão ser usadas como critério para alguma decisão, já que a entrada se limita a coordenadas de pontos em um plano.

Após a semântica obtida, espera-se gerar uma imagem simples a partir dos pontos dados, tentando, conforme as margens de erro, reduzi-las às primitivas básicas, todavia, quando não for possível, será necessário representar a figura por uma sequência de retas e curvas. Pretende-se que o sistema fique aberto e permita a adição da possibilidade do reconhecimento de

novas formas por análise morfológica assim como faz o trabalho Sketchi(COYETTE, 2007), ou casamento dos pontos da grade conforme feito no trabalho Passdoodles (VARENHORST, 2004).

5.2.5. Formatação e Armazenamento das Saídas

Ao final do processo, dadas as formas já reconhecidas, basta gerar o arquivo com a especificação de todas as formas no padrão SVG que pode ser interpretado pelo navegador e outra ferramenta de desenho vetorial, utilizando a sintaxe apresentada na Seção 4.2. O arquivo gerado será apenas texto e poderá ter sua estrutura exibida por qualquer editor de textos, seguindo o esquema de tags XML.

Cada primitiva será associada a uma tag, quando um elemento obtido, for definido por uma sequência de outros elementos será utilizada a tag *path* do SVG para representar, onde é possível definir uma sequência de retas e curvas que compõem o desenho. Devido ao fato da ordem em que são descritas as primitivas não afetar a imagem final, uma ordem para sua gravação não foi estabelecida, logo será gravado na ordem de execução do programa.

5.3. Resultados

O sistema é composto de duas partes, o gerador de pontos e o interpretador da forma. O gerador de pontos visto na Figura 29, é um sistema que possui uma tela para desenho livre de pontos onde as coordenadas do mouse são diretamente gravadas em um arquivo caso o botão do mouse esteja pressionado, essa parte do sistema gera apenas um arquivo em texto contendo as coordenadas dos pontos descritos na interface, sem considerar a ordem em que esses pontos foram inseridos.

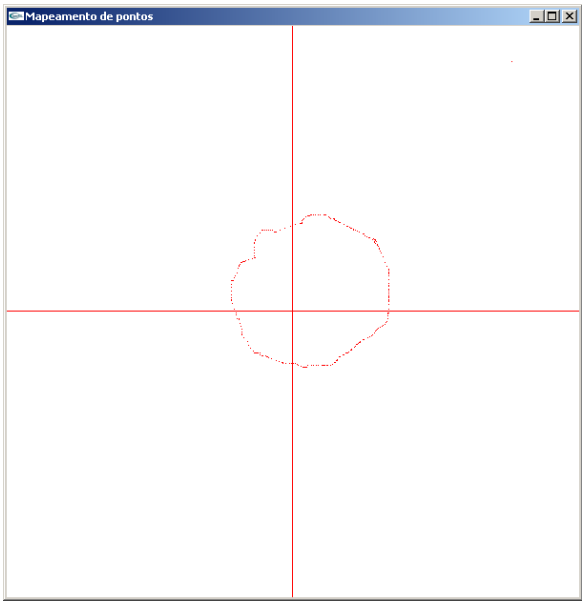


Figura 29 – Interface do gerador de pontos.

Com posse do arquivo gerado ou de outro arquivo oriundo de um outro sistema ou manualmente que siga os padrões esperados pelo reconhecedor, basta carregar o arquivo no sistema que este gerará dois arquivos SVG, um com os pontos originais, e outro com a forma interpretada conforme mostrado na Figura 30.

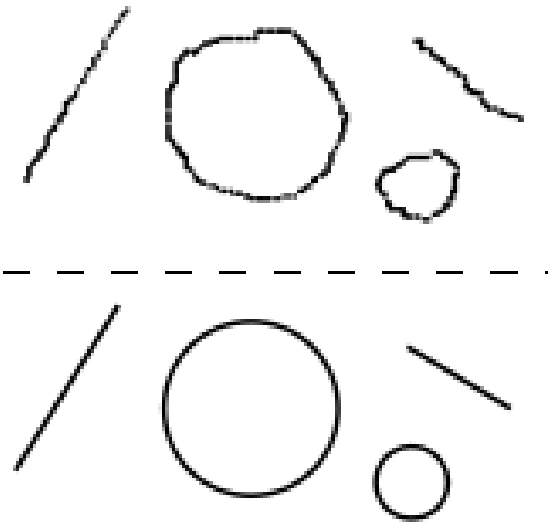


Figura 30 – Comparação entre figura desenhada(acima) e figura interpretada(abaixo)

Dadas as formas que o programa espera, são geradas internamente todas as figuras que o programa se dispõe a interpretar, exceto a figura que representa a forma por um conjunto de curvas. Esta é produzida apenas caso nenhuma forma gerada obtenha uma tolerância mínima. Com todas as formas

geradas, é representada a forma que tem a menor distância de cada ponto mapeado na grade até a figura obtida. Diversos exemplos de reconhecimento de imagens podem ser vistos no Apêndice C. O programa gera, então, para cada figura uma nota que representa, levando em consideração o tamanho da grade, o quão fiel a imagem gerada é em relação aos pontos originais dados. A nota é dada pela fórmula:

$$Nota = 10 - \frac{erro\ médio\ obtido}{tamanho\ da\ grade} * 10$$

$$tamanho\ da\ grade = altura\ da\ grade * largura\ da\ grade$$

$$erro\ médio\ obtido = \sum_{i=1}^{número\ de\ pontos} distância\,(ponto(i), forma)$$

Onde o erro médio é a média dos erros calculados de cada elemento de compõe a figura como um todo, o tamanho da grade corresponde a dimensão para onde foi mapeado os pontos na etapa inicial, ou seja a quantidade de quadrados nas horizontal multiplicado pelo número de quadrados na vertical.
Caso nenhuma imagem gerada tenha uma nota superior a 8,5(oito vírgula cinco), é feita uma estimativa da forma por meio de curvas Bézier quadráticas que tenha proximidade com os pontos. A nota 8,5 foi escolhida empiricamente observando a correspondência da imagem gerada com a desejada após testes exaustivos, sendo que esta pode ser deixada como parâmetro, dependendo do grau de fidelidade que se queira da imagem. Quando utilizada uma tolerância maior que esta, muitas formas serão definidas pelo conjunto de curvas, não representando a intenção do usuário, já uma nota menor que esta reduziria a semelhança da figura obtida.

Foram implementadas as identificações de retas, círculos e curvas Bézier. Sendo as duas primeiras processadas independentemente do conjunto de pontos para se definir qual seria a melhor. E apenas em último caso a forma é desenhada por uma composição de curvas de Bézier, independentemente do erro que seja gerado. Dessa forma, as curvas garantem que sempre se terá uma imagem correspondente para um dado conjunto de pontos dados.

O sistema implementado na linguagem C, cujo núcleo do código se encontra no apêndice D, gera um arquivo de texto que corresponde à sintaxe XML de um arquivo SVG, e não conterá mais a informação dos pontos originais dados, terá somente a especificação das formas que definem a imagem.

A implementação, dessa forma, seguiu os seguintes passos da Figura 31. Leitura do arquivo de pontos, criação do arquivo SVG contendo apenas os pontos originais, mapeamento de cada ponto para uma grade, identificação das componentes conexas da imagem, tentativa de identificação de cada componente conexa por círculo e reta, criação de uma possível forma para cada uma dessas formas, elaboração de uma nota, caso a nota, menor que 8,5(oito vírgula cinco), geração das curvas Bézier e atribuição da nota. Ao final é criado um arquivo onde são inseridos os códigos SVG das formas identificadas, sendo que cada componente conexa gera exatamente uma tag no documento. E este conjunto final de tags é uma imagem vetorial no formato SVG.

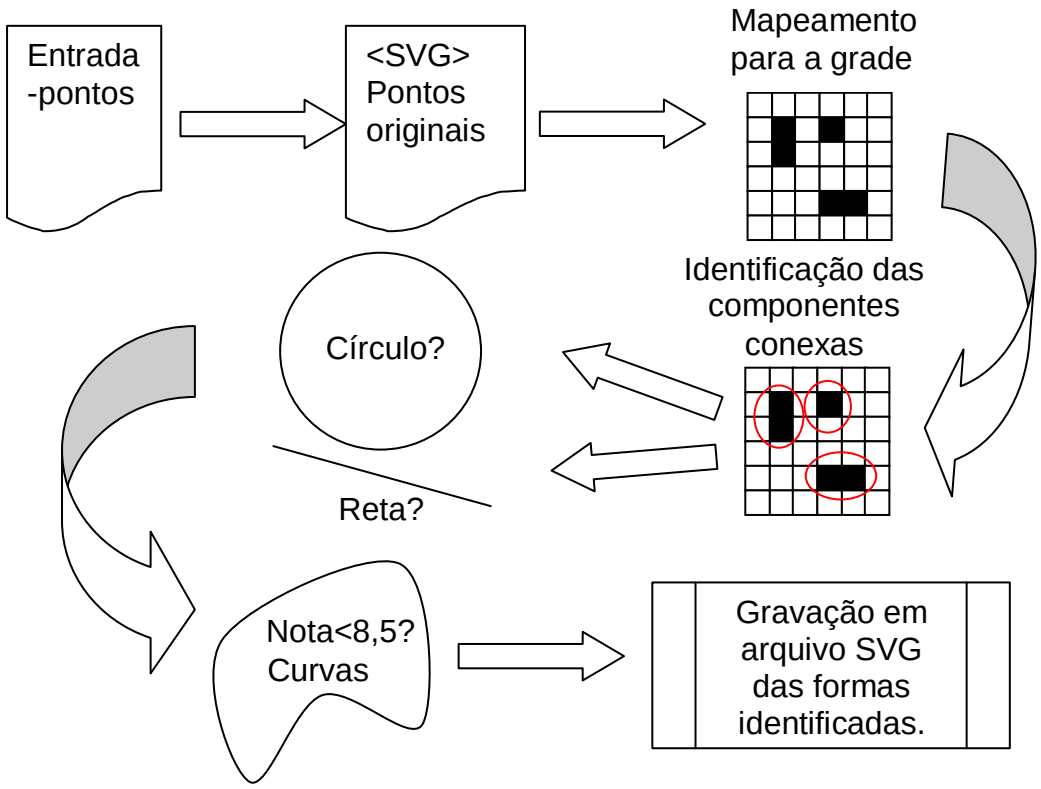


Figura 31 – Diagrama de fluxo do sistema

Conforme a Figura 31, os passos do sistema podem ser definidos da seguinte forma: dado a entrada de pontos, é gerado um arquivo com os pontos originais que é utilizado apenas na etapa de análise. É então feito o mapeamento para a grade e nesta a identificação das componentes conexas. Nestas é tentado a identificação com o círculo e a reta, caso nenhuma destas tenha uma nota maior ou igual a 8,5, é então gerada a imagem por um conjunto de curvas. Independente de qual forma foi reconhecida, esta é então gravada em um arquivo SVG com uma tag para cada elemento conexo da imagem.

Se a aplicação receber de entrada um arquivo com o conteúdo:

```
33 84
50 67
67 49
60 57
43 75
78 34
215 28
193 37
179 54
188 72
204 79
232 73
240 51
233 38
```

O arquivo de saída com base na identificação das formas será respectivamente Uma reta e um círculo, representados no documento SVG segundo as tags:

```
<line x1="64.000000" y1="32.000000" x2="32.000000" y2="80.000000" stroke="black" stroke-width="2"/>
<circle cx="204.000000" cy="46.000000" r="27.383966" fill="none" stroke="black" stroke-width="2"/>
```

Pode-se observar também uma diminuição do número de informações entre a entrada e a saída, enquanto a primeira apresenta a informação da coordenadas de 14(quatorze) pontos no espaço, a segunda obtinha apenas as especificação de 2(duas) formas. Independente do número de pontos, o arquivo final terá apenas o número de formas finais da imagem, não tendo mais a informação dor pontos.

6. AVALIAÇÃO

A avaliação da figura gerada e consequentemente a validação terão duas abordagens, uma objetiva, e outra subjetiva. A avaliação objetiva fará uso da nota que o sistema gera automaticamente escalada entre zero e dez. A avaliação subjetiva será feito segundo a análise de estudantes da Universidade do Estado de Santa Catarina.

6.1. Avaliação Objetiva

A avaliação objetiva fará uso da nota dada pelo sistema a cada forma. Uma nota do sistema pode variar de 0 a 10, onde quanto maior a nota, maior a fidelidade da imagem gerada com os pontos originais, esse fator depende além do algoritmo de identificação da forma, o quão correta, a forma foi desenhada, quanto maior essa nota, mais o desenho se assemelha à forma gerada. Como nota final da imagem, se tem a média dos conceitos atribuídos a cada uma das formas que a compõe, esta tabela de conceituação para dez imagens pode ser visto na Tabela 1.

Tabela 1 – Comparação entre figura desenhada (acima) e figura interpretada (abaixo)

Desenho	Elementos	Nota Elementos	Nota Final
1	A	9,65	9,65
2	A	9,63	9,63
3	A	9,65	9,78
	B	10,00	
	C	9,67	
	D	9,70	
	E	10,00	
	F	10,00	
	G	9,74	
	H	9,64	
	I	9,59	
4	A	9,65	9,67
	B	9,60	
	C	9,67	
	D	9,75	
5	A	10,00	9,89

	B	10,00	
	C	10,00	
	D	9,76	
	E	10,00	
6	A	9,56	9,56
7	A	9,91	9,91
8	A	9,99	9,87
	B	9,98	
	C	9,65	
9	A	9,98	9,98
10	A	9,99	9,99

Na Tabela 1 existem 4(quatro) colunas, a primeira apenas identifica a qual desenho cada linha se refere, cabe a segunda dizer a qual elemento, então a nota na terceira coluna é referente ao elemento da segunda. Por último é exibida uma média entre as notas da terceira coluna que é dito então como nota final da imagem.

6.2. Avaliação Subjetiva

A avaliação subjetiva foi feita com base nas respostas de 53 questionários aplicados aos acadêmicos do curso de Bacharelado em Ciência da Computação, que pode ser visto no Apêndice B, da Universidade do Estado de Santa Catarina, neste questionário, dez figuras foram dispostas com suas correspondentes interpretações geradas pelo sistema, onde deveriam avaliar a imagem, atribuindo-as um conceito entre zero e dez. Neste questionário buscou-se a variedade de imagens para uma validação geral do sistema. A pesquisa foi feita de modo anônimo, de forma a não induzir a resposta de nenhum entrevistado. A Tabela 2 apresenta os resultados da avaliação subjetiva, mostrando na primeira coluna a qual figura corresponde a nota da segunda coluna.

Tabela 2 – Resultados da Avaliação Subjetiva

Desenho	Nota
1	8,81
2	9,17
3	8,70

4	9,23
5	9,09
6	6,89
7	5,87
8	8,21
9	7,87
10	8,94

De um modo geral os desenhos foram bem avaliados, exceto aqueles cujas curvas complexas perdem muita informação no momento em que é mapeado para a grade, ou aqueles cuja tolerância para identificação superou a crítica visual da imagem. Normalmente elementos com nota computacional muito próxima a nota de corte(8,5). A Tabela 3 faz uma comparação quanto à nota computacional e a nota subjetiva.

Tabela 3 – Comparação entre a nota Objetiva e a Subjetiva

Desenho	Nota Subjetiva	Nota Objetiva	Diferença
1	8,81	9,65	0,84
2	9,17	9,63	0,46
3	8,70	9,78	1,08
4	9,23	9,67	0,44
5	9,09	9,89	0,79
6	6,89	9,56	2,67
7	5,87	9,91	4,04
8	8,21	9,87	1,67
9	7,87	9,98	2,11
10	8,94	9,99	1,05

As notas objetivas são relativamente maiores que as subjetivas como visto na Tabela 3, isto deve-se ao fato da nota ser calculada sobre os pontos mapeados para a grade e não sobre os pontos originais, como foi visto pelos avaliadores. Outro fator relevante foi à subjetividade simulada no processo, onde é difícil dizer quão ruim uma imagem deve ser para que seu conceito seja dito como zero. Uma diferença relativamente alta foi observada entre as notas atribuídas às imagens com curvas complexas, devido ao fato, que com o mapeamento para a grade, muita informação dessa curva foi perdida, o que impossibilitou a sua representação de forma mais fiel.

7. CONSIDERAÇÕES FINAIS

O estudo dos trabalhos relacionados, em seu embasamento teórico, levantou as principais etapas para o desenvolvimento de um sistema para reconhecimento de formas. Dentre essas, se observou que inicialmente é viável realizar um tratamento da entrada inicial para uma forma que se possam realizar as devidas análises de forma mais simplificada.

As principais abordagens iniciais vistas foram o mapeamento para uma grade, ou um cálculo matemático através dos pontos diretos no espaço. A abordagem matemática é mais complexa e está mais sujeita a pontos dispersos que dificultariam a identificação da forma, já o mapeamento para a grade além de descartar pontos muito próximos, facilita a identificação de componentes conexas que podem vir a resultar em uma forma.

Diversas semânticas são implementadas em cima do reconhecimento de formas, todavia é possível em muitos casos, isolar apenas a parte que cabe ao reconhecimento da forma, descartando-se a ação tomada após o seu reconhecimento, dessa forma, podem-se obter métodos de reconhecimento de formas de trabalhos com objetivos diferentes, e incorporá-los em um mesmo projeto.

A solução proposta neste trabalho envolveu as etapas de: mapeamento dos pontos para a grade, aquisição das componentes conexas, geração de polígonos através das componentes, reconhecimento das formas através dos polígonos, representação da forma obtida. Sendo cada etapa dependente apenas a sua etapa anterior, onde cada passo está isolado, e resulta em uma saída única, possibilitando assim a alteração de uma etapa do processo sem desencadear mudanças em todo o projeto.

Dentre as linguagens analisadas, o SVG foi escolhido para ser a linguagem cuja saída será gerada pelo sistema, essa escolha se deve ao seu suporte a todas as formas esperadas por esse trabalho, bem como a sintaxe simples e a recomendação da W3C. Sendo a escolha também pela facilidade de execução de um arquivo SVG diretamente no navegador, privando a aplicação de ter que efetuar também a representação gráfica das formas obtidas.

Pela implementação e análise dos resultados, foi possível perceber que

muita informação é perdida na etapa de mapeamento para a grade, todavia, este passo, torna possível a aplicação de outros algoritmos como a idenficação de componentes conexas, realizando também na maior parte dos casos, uma redução na quantidade de pontos a serem trabalhados, descartando aqueles que são de certa forma redundantes devido a proximidade. Porém fica visível essa perda de informação principalmente na imagem gerada pelo conjunto de curvas.

A avaliação subjetiva mostrou um conceito relativamente bom das imagens, sendo 9(nove) das 10(dez) imagens com conceito superior a 7(sete). Os conceitos foram maiores quando a nota objetiva da imagem foi maior para todos os casos exceto aqueles gerados por curvas. A tolerância aplicada no sistema foi aceitável quando se tenta identificar a intenção original do desenho, porém a forma projetada não fica muito semelhante a desenhada.

7.1. Avaliação Subjetiva

Tanto o trabalho como o sistema foi desenvolvido de modo a permitir sua ampliação, diversas aplicações podem ser desenvolvidas tendo por base o reconhecimento realizado por este trabalho.

O próprio núcleo que identifica as formas pode ser ampliado para realizar o reconhecimento de mais formas, onde para isso, se faz necessário a formulação geométrica da forma, um cálculo para elaboração da nota para a forma dado os pontos, e paralelizá-la com as demais formas já identificadas. Para o cálculo da nota, basta uma equação que defina a distância de um ponto qualquer a forma que se quer identificar.

Além da implementação de novas formas para o sistema, outros trabalhos futuros podem analisar quanto à implantação desse módulo a uma semântica, como por exemplo, reconhecimento de gestos, suporte à um mecanismo desenhador, desenvolvimento de interfaces ou projetos de engenharia.

O sistema e suas limitações podem ser tratados em trabalhos futuros, como por exemplo, a separação entre um mesmo componente conexo que referencia a composição de mais de uma forma que se sobrepõem em algum momento na imagem. Atualmente no sistema, para esses casos, é tentado

estabelecer apenas uma forma que defina a imagem como um todo.

Estabelecer um nível de fidelidade ideal e uma relação entre o tamanho da grade e aplicações diversas quanto ao tamanho que gera o melhor reconhecimento da imagem, dado que o tamanho da grade influencia na identificação dos componentes conexos e da imprecisão em relação aos pontos originais.

Implantação de novos mecanismos para definir uma imagem por um conjunto de curvas quando essa curva é formada por diversos pontos, e a implementação de diferentes modelos de curvas nesta identificação, atualmente é utilizado apenas as formas de curva Beziér quadráticas.

Cada etapa da aplicação depende apenas de sua etapa anterior, outro trabalho poderia sugerir a remoção, adição ou trocas de etapas do trabalho a fim de obter alguma melhoria com respeito à velocidade da aplicação ou suporte a outras formas, bem como uma saída diferente do padrão atual.

REFERÊNCIAS

ADOBE. **Post Script Language Reference.** 3ª edição, 2009. Disponível em: <http://www.adobe.com/products/postscript/pdfs/PLRM.pdf>

ALBUQUERQUE, M. P.; ALBUQUERQUE, M. P. **Processamento de imagens: métodos e análises**. Rio de Janeiro: FACET, 2001. Disponível em: <http://www.cbpf.br/cat/pdsi/pdf/ProcessamentoImagens.PDF>

BARBOSA, João Lucas Marques. **Geometria Euclidiana Plana.** 6ª edição. SBM, 2004.

COELHO, Alex; FAGUNDES, Fabiano. **Protótipo de um sistema de auxílio à localização no CEULP/ULBRA utilizando imagens vetoriais Scalable Vector Graphics.** Palmas – TO: ULBRA, 2004. Disponível em: <http://www.ulbra-to.br/ensino/43020/artigos/anais2004/anais/alexCoelhoSVGEncoinfo2004.pdf>

COREL. **About Corel.** Disponível em <http://www.corel.com/servlet/Satellite/us/en/Content/1152796555483>

COYETTE, Adrien.; **A Methodological Framework for Multi-Fidelity Sketching of User Interfaces**, Ph.D. thesis, Université catholique de Louvain, Louvain-la-Neuve, Belgium, 22 October 2007.

DERAKHSHANI Dariush; MUNN, Randi Lorene. **Introducing 3ds Max 2008**. John Wiley and Sons, 2008.

EBP, Energy Performance of Buildings Group, **xFig User Manual – Versão 3.2.5.** 2007. Disponível online em: <http://epb.lbl.gov/xfig/>

FONSECA M.J., PIMENTEL C. and JORGE J.A., CALI: **An Online Scribble Recognizer for Calligraphic Interfaces**, in Proceedings of the 2002 AAAI Spring Symposium – Sketch Understanding, Palo Alto, USA, 2002,

GONZALEZ, Rafael C.; WOODS, Richard E. **Processamento de Imagens Digitais**. São Paulo, Edgard Blücher, 2000. ISBN 8521202644.

MSDN, Microsoft Developer Network. **Windows Metafile Format Specification.** Disponível online: <http://msdn.microsoft.com/en-us/library/cc215212.aspx>

ROSÁRIO, Ricardo do. e-Lapis - **Gerador de Desenhos Vetoriais por Visão Computacional**. Joinville – SC: UDESC, 2008.

SEGURA, Vinicius. **Computação Gráfica - Evolução de Curvas e Superfícies**. PUC-RIO, 2005.

SELINGER, Peter. **Potrace: a polygon-based tracing algorithm.** Department of Mathematics and Statistics. Dalhousie University, 2003.

SILVA, A.G.; LOTUFO R. A. **Hierarchical Morphological Analysis for Generic Detection of Shapes in Grayscale Images**. Portugal, 2006. Pag 405-410.

SKIENA, Steven S; REVILLA, Miguel A. **Programming Challenges – The Programming Contest Training Manual.** Springer, 2003

STEINBRUCH, Alfredo; WINTERLE, Paulo. **Geometria Analítica.** Editora McGraw-Hill. Brasil, 1987.

SVGBASICS. SVGBasics tutorials. Disponível em: < http://www.svgbasics.com/index.html>

USIXML. User Interface Extensible Markup Language. Disponível em: http://www.usixml.org/.

VARENHORST; Christopher. **Passdoodles; a Lightweight Authentication Method.** Research Science Institute. Massachusetts Institute of Technology, 2004.

W3C. Scalable Vector Graphics (SVG) - XML Graphics for the Web. 2009. Disponível em: < http://www.w3.org/TR/SVG11/>

A – PLANO DE TRABALHO DE CONCLUSÃO DE CURSO

Plano de Trabalho de Conclusão de Curso

e-Lapis – Reconhecimento de primitivas geométricas em desenhos vetoriais

UDESC - Centro de Ciências Tecnológicas

Departamento de Ciência da Computação

Bacharelado em Ciência da Computação - Integral

Turma 2009/1 - Joinville – Santa Catarina

Marlon Fernandes de Alcantara – marlon_udesc@yahoo.com.br

Orientador: Alexandre Gonçalves Silva – alexandre@joinville.udesc.br

Resumo – Algumas imperfeições, em desenhos livres manuais produzidos por meio de dispositivos vetoriais (mouse, mesas digitalizadoras, telas sensíveis ao toque, etc), podem ser tratadas no sentido de promover uma melhor qualidade visual da imagem final. Com a transformação de pontos isolados em objetos vetoriais, uma representação concisa do desenho é obtida, possibilitando uma posterior edição simples e mais intuitiva. Este trabalho consiste na análise e interpretação de pontos no espaço bidimensional com o objetivo de classificá-los e descrevê-los em primitivas geométricas ou curvas paramétricas. Para isto, um levantamento de formas de aquisição, padrões vetoriais de imagens e métodos relacionados de interpretação de pontos devem ser detalhados, assim como projetada e implementada uma proposta de reconhecimento de elementos semânticos em traços de desenhos feitos à mão.

Palavras-chave: *imagens vetoriais, interpretação de imagens, tratamento de pontos.*

1. Introdução e Justificativa

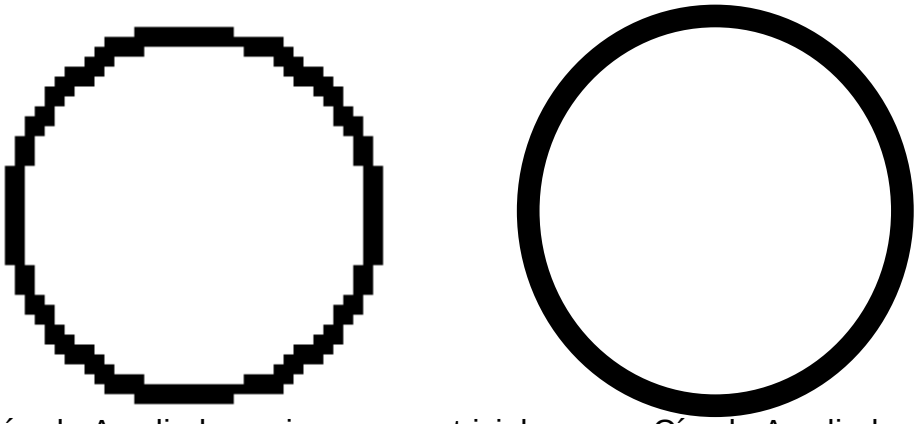
Imagens nada mais são que representações visuais de algo que se quer expressar, que pode ser material como em uma fotografia ou abstrato como um desenho ou representação artística da realidade. O “processamento das imagens” tem por objetivo extrair informações das imagens ou transformá-las em outras imagens onde seja mais fácil a extração de informações. Como o processamento de imagens segue a mesma linha do “processamento de sinais”, onde ambos são suportes físicos que possuem alguma informação a ser transmitida, se esta informação se refere a uma medida, dizemos que a informação está associada a um fenômeno físico, caso esta associação seja a um nível cognitivo, chamamos de conhecimento [Albuquerque e Albuquerque, 2001; Rosário, 2008].

A “computação gráfica” segue a linha oposta ao “processamento de imagens”, quando a preocupação do “processamento de imagens” é extrair a informação, a “computação gráfica” busca transformar a informação em elementos visuais (gráficos). A computação gráfica faz uso de diversos métodos matemáticos a fim de conseguir uma representação visual para a informação a ser transmitida, o que não significa que o processamento de imagens não possa fazer uso dos mesmos métodos afim de conseguir o processo inverso, já que mesmo sendo trabalhos opostos, ambas estão ligadas no que se refere a área de trabalho [Albuquerque e Albuquerque, 2001].

Imagem matricial, ou bitmap (mapa de bits), é a representação de imagens através de uma matriz de pixels. Um pixel é a menor quantidade em uma imagem, ocupa uma posição na matriz e assume uma determinada cor, ou seja, a representação da imagem matricial depende da característica de cada pixel. Estes são dispostos de forma que sua união crie a aparência de uma figura. Por exigir que cada pixel contenha certo valor de cor e posição, as imagens matriciais podem tornar-se grandes para armazenamento. A mesma idéia de matriz de pontos é usada, por exemplo, para impressão de imagens e monitores de vídeo, algumas imagens capturadas por câmeras digitais e

scanners também usam essa forma de representação de imagens [Derakhshani e Munn, 2008].

Os desenhos em formatos vetoriais possuem a capacidade de representar as suas imagens através de matrizes e fórmulas matemáticas, sendo dessa forma, muito mais compacto que imagens do tipo matricial, já que se dispensa explicitar as coordenadas de cores de cada pixel da imagem, sendo somente necessário mensurar as coordenadas de determinada primitiva geométrica, dessa forma, não é armazenada a informação visual, mas sim as instruções para criação das imagens. O formato vetorial permite o seu redimensionamento sem que com isso haja uma perda de qualidade, o que não ocorre no formato matricial, onde os pixels são reorganizados para ser efetuado o redimensionamento, onde em certo ponto pode perder o conjunto visual, ficando visíveis apenas os pixels em sua forma quadrangular, conforme pode ser visto na figura 1, outra vantagem seria na edição das imagens vetoriais, onde é possível manipular toda a sua forma ao mesmo tempo e não somente cada pixel por vez [Coelho e Fagundes, 2004].



Círculo Ampliado em imagem matricial Círculo Ampliado em imagem vetorial

Figura 1: Comparação entre a ampliação de uma imagem matricial e uma imagem vetorial.

Para todo, qualquer formato vetorial baseia sua especificação em primitivas geométricas, cujo sua especificação se baseia em parâmetros que podem ser descritos de forma geométrica através de funções. As primitivas mais comuns nesse formato são: retângulo, círculo, linha, polígono, elipse e curvas. Ainda que algum formato possa suprimir ou acrescentar alguma outra

primitiva. Todavia as imagens vetoriais não são indicadas para imagens naturais devido a não possibilidade de representá-la pelas primitivas geométricas, e para formas complexas, devido ao tempo que levaria a interpretação da imagem [Coelho e Fagundes, 2004].

O trabalho se restringirá nas imagens abstratas, mas precisamente em desenhos na forma binária onde existe cor, ou a sua ausência em determinado ponto da imagem. Observa-se que o trabalho de aquisição, o tratamento inicial, a transformação na forma binária e a especificação na representação vetorial do padrão SVG, já foram realizadas em um trabalho anterior [Rosário, 2008].

A detecção de formas é um método que consiste em, dado um conjunto de pontos, que podem ser pixels de uma imagem, dizer a que forma o dado conjunto mais se assemelha, e se é possível afirmar que o conjunto de pontos representa realmente aquela forma para uma tolerância especificada. Esta detecção pode ser realizada de duas maneiras, uma seria por análise de forma a partir de figuras geométricas de formulação simples ou critério morfológico, como por exemplo: linhas, círculos, retas, arcos, elipses, curvas. Outra maneira seria a detecção por casamento de padrões, quando a imagem a ser detectada possui uma forma complexa, por exemplo um caractere ou outros símbolos, neste caso o método mais indicado seria por casamento de modelo da figura perfeita com a figura a ser reconhecida [Silva e Lotufo, 2006].

O reconhecimento de primitivas geométricas em desenhos vetoriais pode auxiliar em muitos aspectos. Primeiramente, é de alta complexidade para o usuário de um sistema desenhar uma primitiva geométrica de forma perfeita, dessa forma, por mais que o usuário desenhe a forma desejada de forma imperfeita, é possível, a partir da imagem gerada, interpretar a forma desejada e representá-la perfeita, mesmo que o usuário a tenha desenhado de forma diferente. Nos desenhos a mão, principalmente se for desejado fazer de forma mais rápida ou sem a ajuda de equipamentos que auxiliem na precisão, se torna inviável a elaboração de figuras. Com a interpretação computacional da imagem, não é necessário a minuciosa preocupação com a forma ainda na etapa de desenho, e como a imagem final gerada, é na forma vetorial, torna ainda mais fácil o seu manuseio para uma eventual correção semântica manual. Coyette (2007), por exemplo, utiliza esta ideia no auxílio a construção de interfaces gráficas.

Outro aspecto seria a simplificação da especificação da imagem final, tornando-a mais simples e menos extensa. Formando dessa maneira a união de diversas primitivas que compõe um objeto em uma especificação única do objeto como um todo. Dessa forma, quando é desejado fazer a edição da imagem, e a alteração de elementos visuais quaisquer, é possível manusear o objeto de forma completa ao invés de realizar o mesmo procedimento para as diversas partes.

Para imagens oriundas de dispositivos como câmeras digitais, ou *webcams,* seria possível a partir da imagem adquirida e seus eventuais ruídos, obter os pontos da imagem, e a partir deles, fazer a interpretação geométrica e a união dos pontos visuais da imagem, tornando uma imagem de especificação limpa, e facilmente manuseada pelo usuário. Também ajudaria a estruturar parte da imagem que ficou distorcida devido a perspectiva visual do equipamento de captura, ou por quaisquer outras pequenas falhas da imagem ou do desenho manual.

2. Objetivos

Objetivo Geral: acrescentar ao e-Lapis [Rosário, 2008] a funcionalidade de simplificação de imagens vetoriais a partir da interpretação da disposição espacial dos pontos gerados, e o reconhecimento de primitivas geométricas, de modo a produzir imagens com maior qualidade e com suporte à edição de novos objetos gráficos.

Objetivos Específicos:

- Estabelecer métodos de aquisição de imagens vetoriais.
- Compreensão de padrões vetoriais.
- Tratamento dos pontos a partir de imagens vetoriais.
- Definir o conjunto de primitivas geométricas a serem analisadas.
- Estudo e escolha de métodos para a obtenção das primitivas a partir dos pontos dados.

3. Metodologia

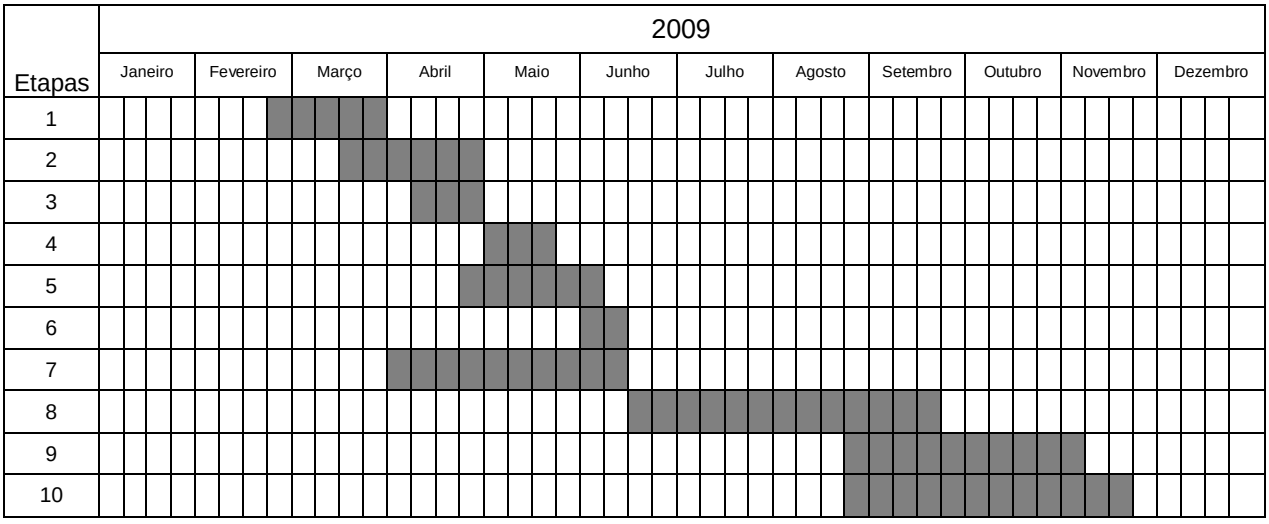
O trabalho será desenvolvido com base em um levantamento dos métodos já utilizados para trabalhar com imagens no formato vetorial, e realizar o tratamento de pontos proposto neste. Para isso, será realizada uma filtragem dentre esses métodos, onde serão vistos quais são úteis e viáveis para a realização deste trabalho. São pontos fundamentais desse trabalho, as seguintes etapas:

1. Fundamentação teórica na parte de imagens e processamento delas, modelos de representação de arquivo.
2. Levantamento de técnicas já utilizadas para processamento de pontos no formato vetorial.
3. Detalhamento de características de padrões de arquivo no formato vetorial e aprofundamento no funcionamento e na sintaxe do padrão escolhido.
4. Estudo matemático das primitivas básicas a que serão reduzidos os pontos.
5. Elaboração da proposta de simplificação de imagens vetoriais, unindo um conjunto de pontos em especificações de primitivas básicas do padrão adotado.
6. Escolha da linguagem e exploração de possíveis bibliotecas livres para auxiliar na implementação.
7. Escrita da Monografia I
8. Implementação
9. Análise dos Resultados
10. Escrita da Monografia II

Ao término das etapas, se espera ter todo o embasamento necessário para desenvolver uma ferramenta capaz de realizar todas as transformações descritas e gravação da imagem no formato de arquivo padrão vetorial. Para dessa forma, obter um produto final que dado uma imagem qualquer de

qualquer fonte, se possa descrevê-la no formato vetorial como o mínimo de formas primitivas possível sem grande perda de informação visual.

4. Cronograma Proposto



5. Linha e Grupo de Pesquisa

A linha de pesquisa deste trabalho de conclusão de curso é o estudo de técnicas para o reconhecimento de primitivas geométricas em desenhos vetoriais. O trabalho faz parte do grupo de pesquisa LARVA (Laboratório de Realidade Virtual Aplicada).

6. Forma de Acompanhamento/Orientação

Para este trabalho, o acompanhamento e orientação serão feitos através de encontros semanais, realizados em horários definidos pelo orientador e aluno, e

também com a utilização de correio eletrônico. No decorrer do trabalho, se uma das partes achar conveniente ou necessário, poderá ser acrescentado outras tarefas e horários de encontro.

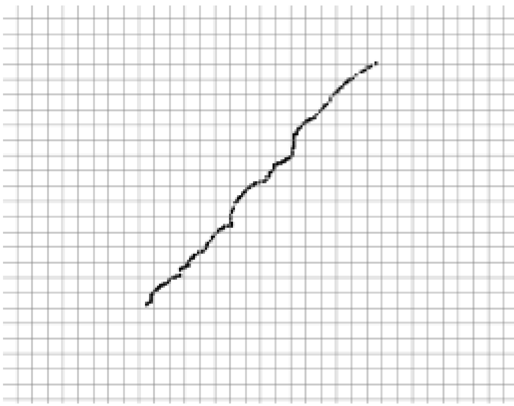
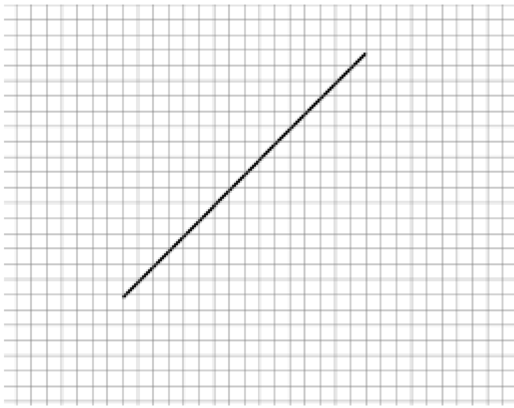
B – FORMULÁRIO DE AVALIAÇÃO SUBJETIVA

RECONHECIMENTO DE PRIMITIVAS GEOMÉTRICAS EM DESENHOS VETORIAIS

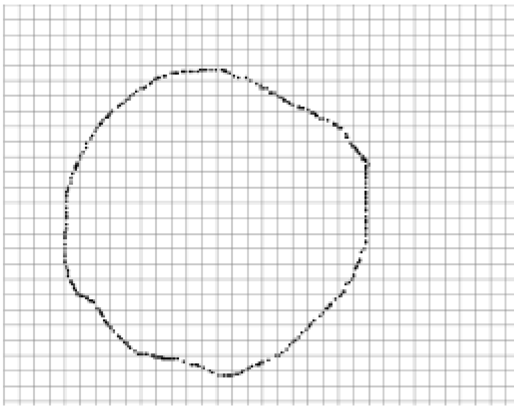
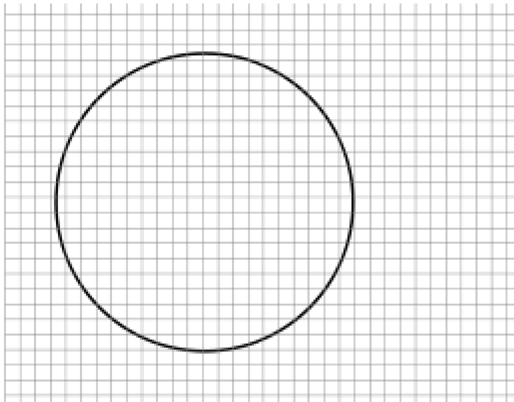
- Essa é uma pesquisa anônima, portanto, não assine.

- As imagens à esquerda correspondem a um conjunto de pontos gerados manualmente e, à direita, representam a interpretação computacional destes pontos como um objeto geométrico dentre três possíveis (reta, circunferência ou curvas).

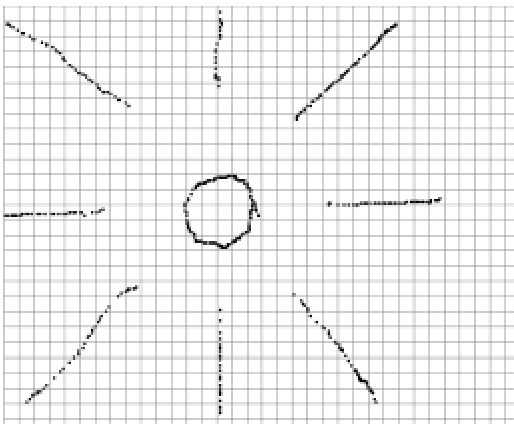
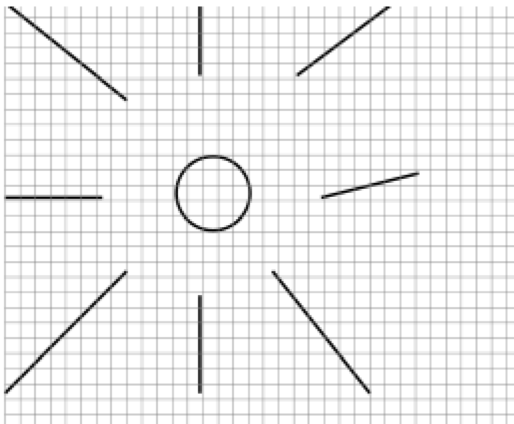
- Dê uma nota de 0 (zero) a 10 (dez) sobre as decisões tomadas automaticamente pelo algoritmo desenvolvido e a fidelidade das formas geradas em relação aos pontos iniciais em cada caso.

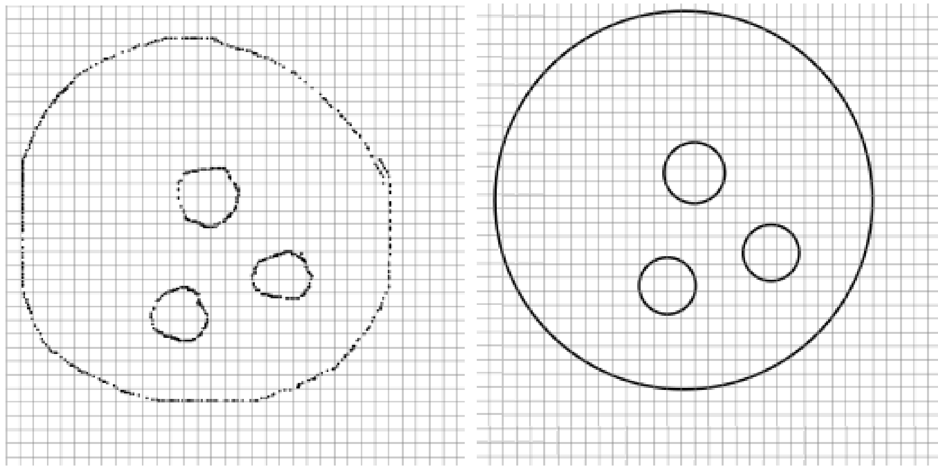
0 □ 1 □ 2 □ 3 □ 4 □ 5 □ 6 □ 7 □ 8 □ 9 □ 10□

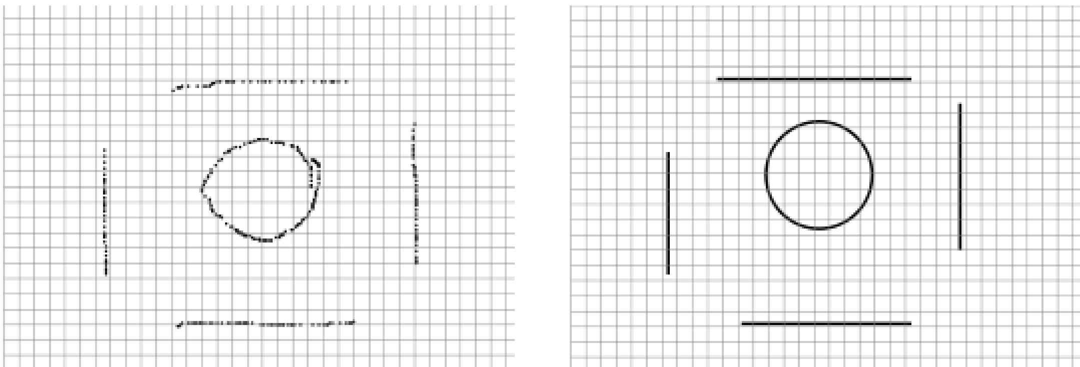
0 □ 1 □ 2 □ 3 □ 4 □ 5 □ 6 □ 7 □ 8 □ 9 □ 10□

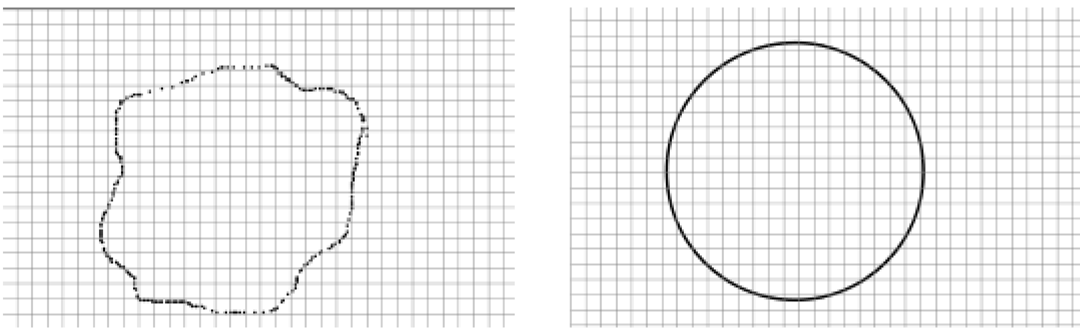
0 □ 1 □ 2 □ 3 □ 4 □ 5 □ 6 □ 7 □ 8 □ 9 □ 10□



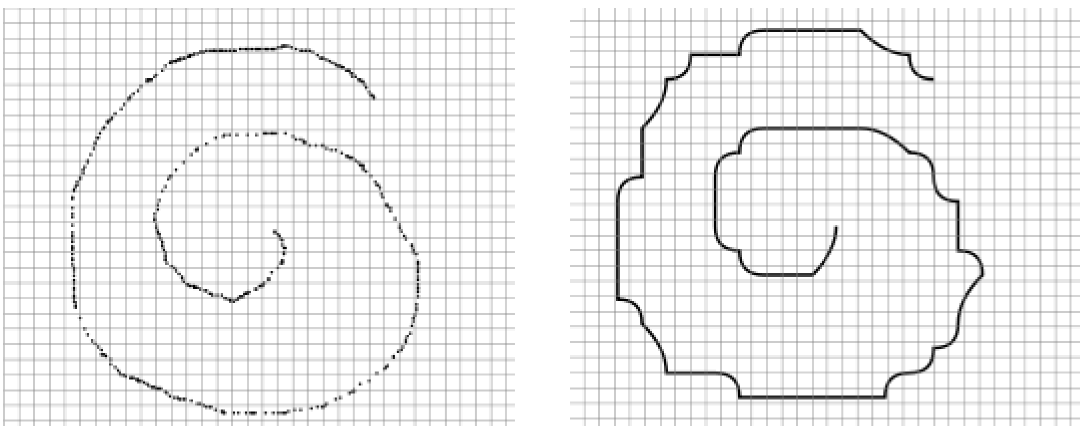
0 ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐ 8 ☐ 9 ☐ 10☐



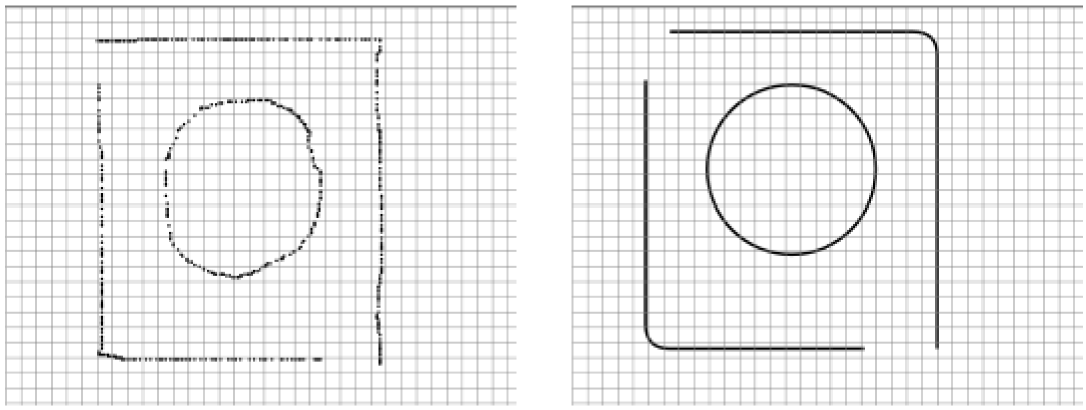
0 ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐ 8 ☐ 9 ☐ 10☐



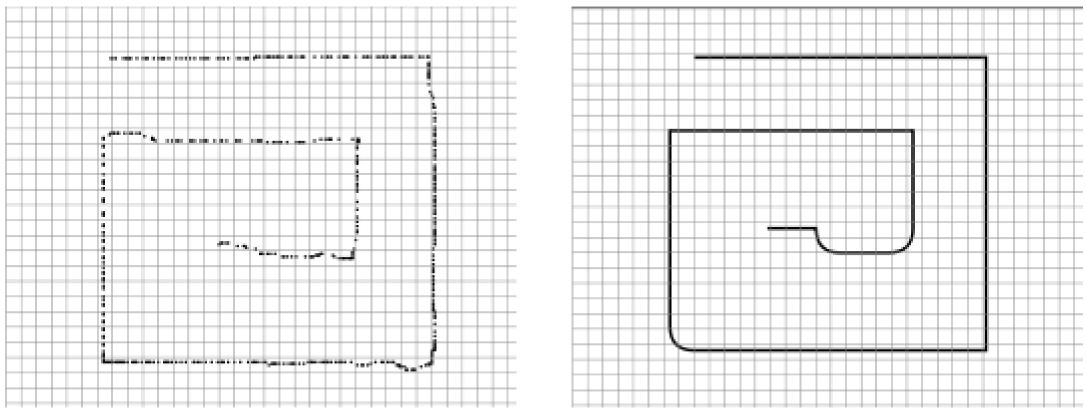
0 ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐ 8 ☐ 9 ☐ 10☐



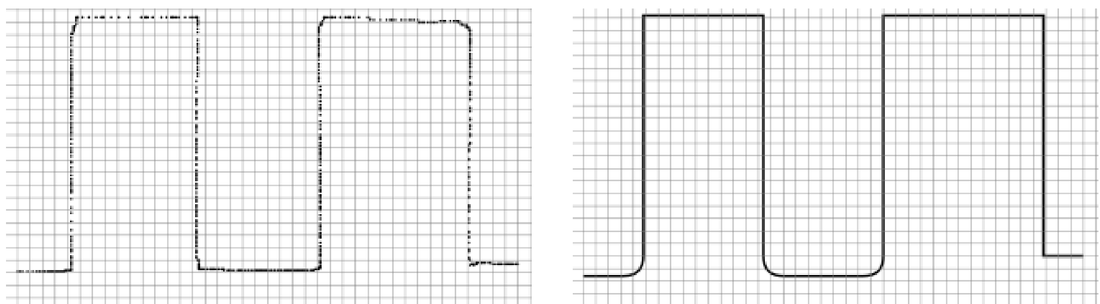
0 ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐ 8 ☐ 9 ☐ 10☐



0 □ 1 □ 2 □ 3 □ 4 □ 5 □ 6 □ 7 □ 8 □ 9 □ 10□



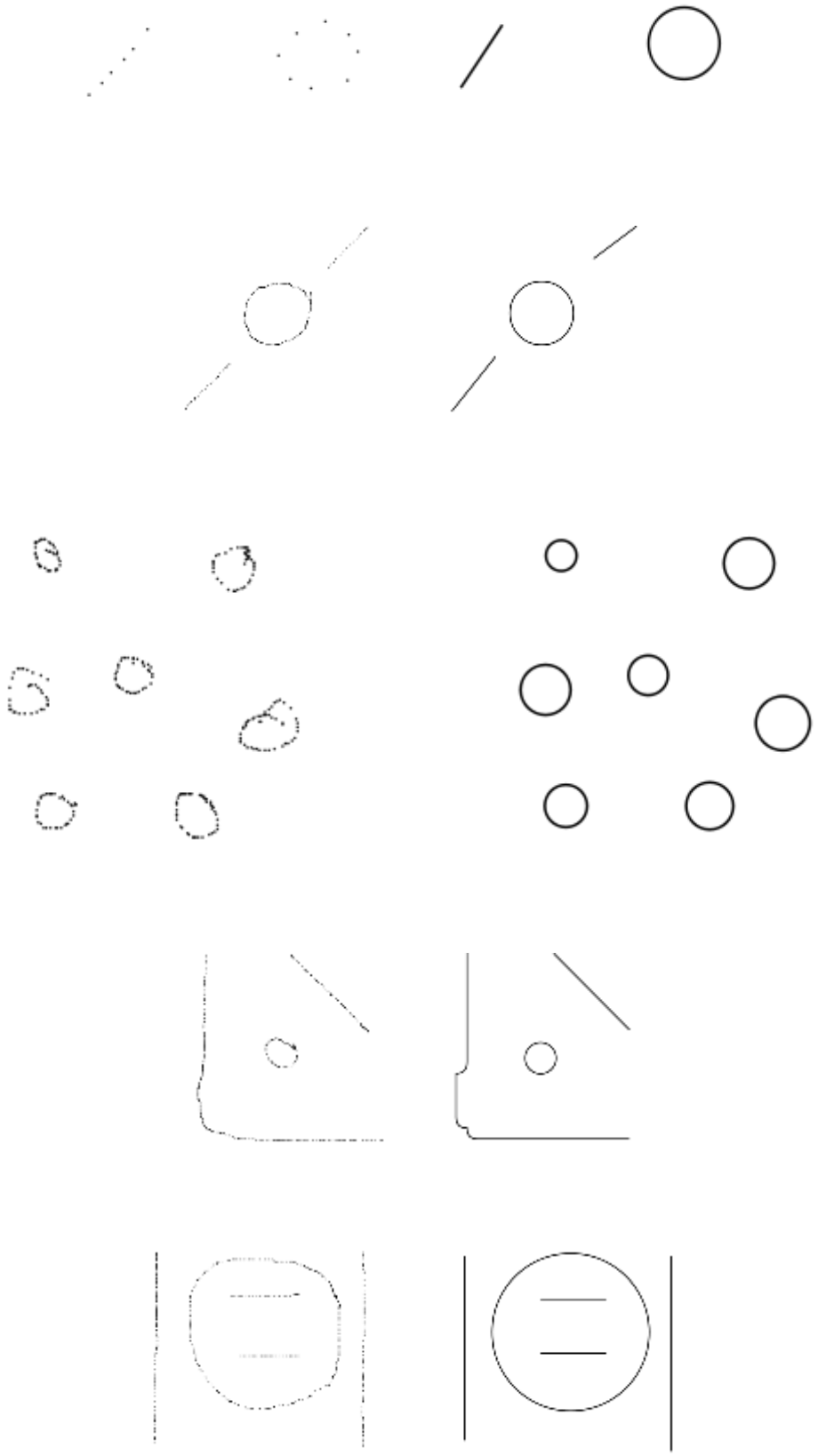
0 □ 1 □ 2 □ 3 □ 4 □ 5 □ 6 □ 7 □ 8 □ 9 □ 10□

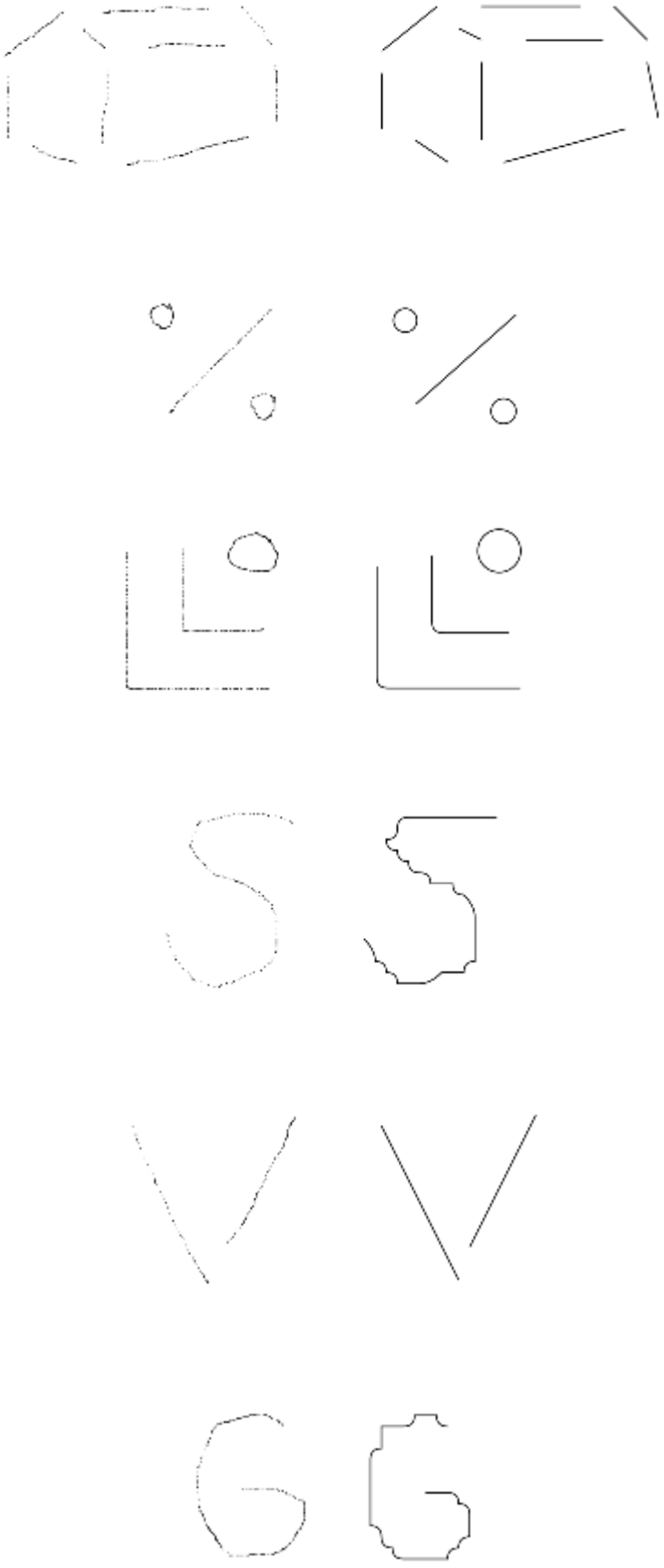


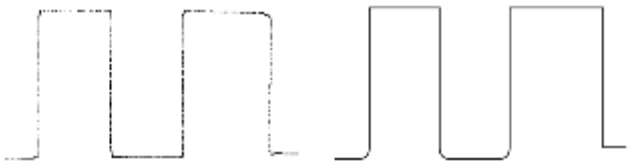
0 □ 1 □ 2 □ 3 □ 4 □ 5 □ 6 □ 7 □ 8 □ 9 □ 10□

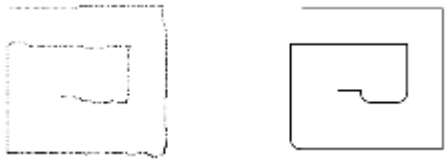
Obrigado por responder a essa avaliação!

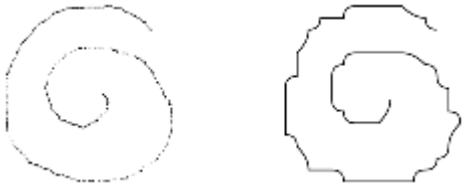
C – EXEMPLOS DE ENTRADA E SAÍDA DO SISTEMA

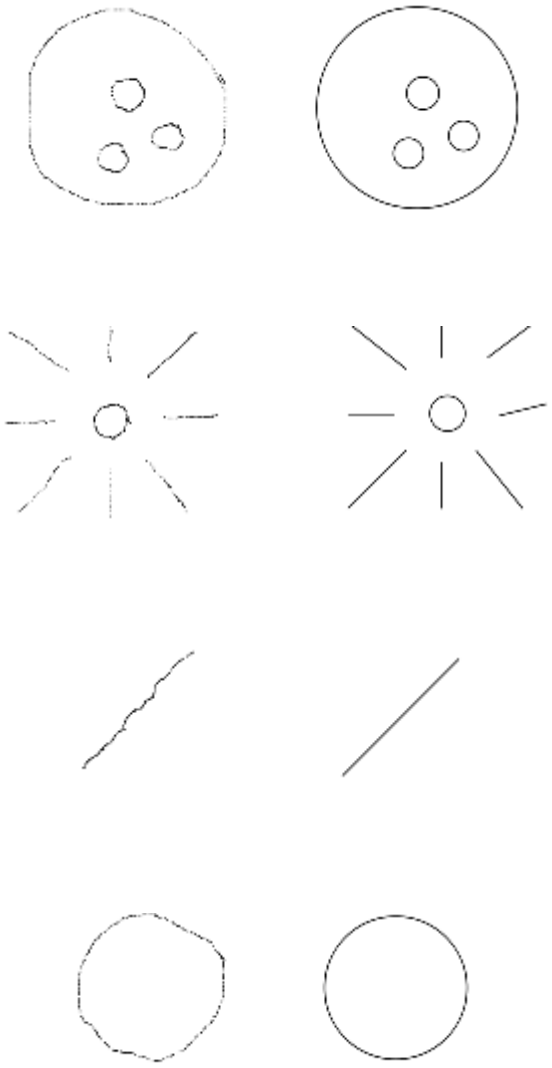












D – CÓDIGO FONTE DO NÚCLEO DA APLICAÇÃO EM LINGUAGEM C

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#define GRID 50
#define GRIDVID 800/GRID

typedef struct{
    float x, y;
}Ponto;

Ponto p[100000];
char pontos[100];
char grid[GRID+1][GRID+1];
char g[GRID+1][GRID+1];
int pos;
FILE *arq;
float erroreta;
float errocirc;
float errocurv;
char saidareta[200];
char saidacirculo[200];
float pc, pr, pcur;
Ponto aux[100000];
int paux;
int inter;

float distancia(Ponto a, Ponto b){
    return sqrt(pow((a.x-b.x), 2) + pow((a.y-b.y), 2));
}

float distanciapontoreta(float a, float b, float c, Ponto pt){
    return(fabs(a*pt.x + b*pt.y + c)/sqrt(a*a+b*b));
}

void montacomponente(int i, int j){
    if(i<0 || j<0 || i>= GRID || j>=GRID || !grid[i][j]) return;
    grid[i][j]=0;
    p[pos].x = i*GRIDVID;
    p[pos].y = j*GRIDVID;
    pos++;
    montacomponente(i+1, j); //BAIXO
    montacomponente(i, j+1); //DIREITA
    montacomponente(i-1, j); //CIMA
    montacomponente(i, j-1); //ESQUERDA
    /*ATÉ AQUI CONECTIVIDADE 4*/
    montacomponente(i+1, j+1); //BAIXO-DIREITA
    montacomponente(i+1, j-1); //BAIXO-ESQUERDA
    montacomponente(i-1, j+1); //CIMA-DIREITA
    montacomponente(i-1, j-1); //CIMA-ESQUERDA
    /*ATÉ AQUI CONECTIVIDADE 8*/
}

void desenhareta(){
    float maiorx, maiory, menorx, menory;
```

```
    int idmaiorx, idmaiory, idmenorx, idmenory;
    int i;
    maiorx = menorx = p[0].x;
    maiory = menory = p[0].y;
    idmaiorx=idmaiory=idmenorx=idmenory=0;
    for(i=1; i<pos; i++){
        if(p[i].x>maiorx){
            maiorx = p[i].x;
            idmaiorx = i;
        }
        if(p[i].x<menorx){
            menorx = p[i].x;
            idmenorx = i;
        }
        if(p[i].y>maiory){
            maiory = p[i].y;
            idmaiory = i;
        }
        if(p[i].y<menory){
            menory = p[i].y;
            idmenory = i;
        }
    }
    float dy, dx, dxy, dyx;
    dy = distancia(p[idmenory], p[idmaiory]);
    dx = distancia(p[idmenorx], p[idmaiorx]);
    dxy = distancia(p[idmenorx], p[idmaiory]);
    dyx = distancia(p[idmenory], p[idmaiorx]);
    int pi, pf;

    if(dx >= dy && dx >= dxy && dx >= dyx){
        pi = idmenorx;
        pf = idmaiorx;
    }else if(dy >= dx && dy >= dxy && dy >= dyx){
        pi = idmenory;
        pf = idmaiory;
    }else if(dxy >= dy && dxy >= dx && dxy >= dyx){
        pi = idmenory;
        pf = idmaiory;
    }else if(dyx >= dy && dyx >= dxy && dyx >= dx){
        pi = idmenory;
        pf = idmaiory;
    }
    float a, b, c;
    a = p[pi].y-p[pf].y;
    b = p[pf].x-p[pi].x;
    c = p[pi].x*p[pf].y-(p[pi].y*p[pf].x);
    erroreta=0;
    for(i=0; i<pos; i++){
        erroreta += distanciapontoreta(a, b, c, p[i]);
    }
    pr = 100-((erroreta/pos)/((GRID*GRID)/(GRIDVID))*100);
    printf("PROBABILIDADE DE SER RETA: %.2f%%\n", pr);
    sprintf(saidareta, "<line x1=\"%f\" y1=\"%f\" x2=\"%f\" y2=\"%f\"
stroke=\"black\" stroke-width=\"2\"/>\n", p[pi].x, p[pi].y, p[pf].x,
p[pf].y);
}

void desenhacircunferencia(){
    //CALCULANDO O CENTROIDE
```

```
    float sx, sy, sd, raio; //soma dos x, soma dos y, soma das
distancias
    Ponto centroide;
    int i;
    sx=sy=0;
    for(i=0; i<pos; i++){
        sx+=p[i].x;
        sy+=p[i].y;
    }
    centroide.x = sx/pos;
    centroide.y = sy/pos;
    sd=0;
    for(i=0; i<pos; i++){
        sd+=distancia(centroide, p[i]);
    }
    raio = sd/pos;
    errocirc = 0;
    for(i=0; i<pos; i++){
        errocirc += fabs(distancia(p[i], centroide) - raio);
    }
    pc = 100-((errocirc/pos)/((GRID*GRID)/(GRIDVID))*100);
    printf("PROBABILIDADE DE SER CIR.: %.2f%%\n", pc);
    sprintf(saidacirculo, "<circle cx=\"%f\" cy=\"%f\" r=\"%f\"
fill=\"none\" stroke=\"black\" stroke-width=\"2\"/>\n", centroide.x,
centroide.y, raio);
}

void extremo(int i, int j){
    g[i][j]=0;
    aux[paux].x = i*GRIDVID;
    aux[paux].y = j*GRIDVID;
    paux++;
    if(i+1<GRID && g[i+1][j]){
        extremo(i+1, j);
    }
    if(j+1<GRID && g[i][j+1]){
        extremo(i, j+1);
    }
    if(i-1>=0 && g[i-1][j]){
        extremo(i-1, j);
    }
    if(j-1>=0 && g[i][j-1]){
        extremo(i, j-1);
    }
    if(i+1 < GRID && j+1 < GRID && g[i+1][j+1]){
        extremo(i+1, j+1);
    }
    if(i-1 >= 0   && j-1 >=0    && g[i-1][j-1]){
        extremo(i-1, j-1);
    }
    if(i+1 < GRID && j-1 >=0    && g[i+1][j-1]){
        extremo(i+1, j-1);
    }
    if(i-1 >=0    && j+1 < GRID && g[i-1][j+1]){
        extremo(i-1, j+1);
    }
    if(!inter) inter = paux;

}

void generico(){
```

```
    int i, j;
    memset(g, 0, sizeof(grid));
    for(i=0; i<pos; i++){
        g[(int)p[i].x/(GRIDVID)][(int)p[i].y/(GRIDVID)] = 1;
    }
    /*for(i=0; i<GRID; i++){
        puts("");
        for(j=0; j<GRID; j++){
            if(g[j][i]) printf("#");
            else printf(" ");
        }
    }*/
    for(i=0; i<GRID; i++){
        for(j=0; j<GRID; j++){
            if(g[i][j]){
                paux = 0;
                inter = 0;
                extremo(i, j);
                goto sai;
            }
        }

    }
    sai:;
    pos = 0;
    p[pos] = aux[inter-1];
    pos++;

    for(i=inter-2; i>=0; i--){
        p[pos] = aux[i];
        pos++;
    }
    for(i=inter; i<paux; i++){
        p[pos] = aux[i];
        pos++;
        //fprintf(arq, "L %f %f ", aux[i].x, aux[i].y);
    }
    fprintf(arq, "<path d = \"M ");
    fprintf(arq, "%f %f ", p[0].x, p[0].y);
    Ponto m1, m2;
    errocurv=0;
    for(i=1; i+1<pos; i+=2){
        m1.x=(p[i-1].x+p[i].x)/2.0;
        m1.y=(p[i-1].y+p[i].y)/2.0;
        m2.x=(p[i].x+p[i+1].x)/2.0;
        m2.y=(p[i].y+p[i+1].y)/2.0;
        float a, b, c;
        a = m1.y-m2.y;
        b = m2.x-m1.x;
        c = m1.x*m2.y-(m1.y*m2.x);
        errocurv += distanciapontoreta(a, b, c, p[i]);
        //printf("ERRO CURV %f\n", errocurv);
        fprintf(arq, "Q %f,%f %f,%f ", p[i].x, p[i].y, p[i+1].x,
p[i+1].y);
    }
    fprintf(arq, "\" stroke = \"black\" stroke-width = \"2\" fill =
\"none\"/>");
    pcur = 100-((errocurv/pos)/((GRID*GRID)/(GRIDVID))*100);
}

int main(){
```

```
    int i, j;
    pos=0;
    printf("Digite o nome do arquivo txt que contem o conjunto de
pontos:\n:.");
    do{
        gets(pontos);
        arq = fopen(pontos, "r");
        //arq = fopen("an.txt", "r");
        if(!arq)
            printf("Arquivo nao encontrado!\nDigite novamente o nome
do arquivo que contem o conjunto de pontos:\n:.");
    }while(!arq);
    while(fscanf(arq, "%f %f", &p[pos].x, &p[pos].y) != EOF) pos++;
    fclose(arq);
    arq = fopen("pontos.svg", "w");
    fprintf(arq, "<svg viewBox=\"0 0 800 800\"
xmlns=\"http://www.w3.org/2000/svg\" version=\"1.1\">\n\n");
    for(i=0; i<pos; i++){
        fprintf(arq, "<circle cx=\"%f\" cy=\"%f\" r=\"1\"
fill=\"black\"/>\n", p[i].x, p[i].y);
    }
    fprintf(arq, "\n</svg>");
    fclose(arq);
    //MONTANDO O GRID
    memset(grid, 0, sizeof(grid));
    for(i=0; i<pos; i++){
        grid[(int)p[i].x/(GRIDVID)][(int)p[i].y/(GRIDVID)] = 1;
    }
    arq = fopen("saida.svg", "w");
    fprintf(arq, "<svg viewBox=\"0 0 800 800\"
xmlns=\"http://www.w3.org/2000/svg\" version=\"1.1\">\n\n");
    for(i=0; i<=GRID; i++){
        for(j=0; j<=GRID; j++){
            if(grid[i][j]){
                pos = 0;
                montacomponente(i, j);
                desenhacircunferencia();
                desenhareta();
                if(erroreta<errocirc && pr>85){
                     fprintf(arq, saidareta);
                }
                else if(pc>85){
                     fprintf(arq, saidacirculo);
                }else{
                     generico();
                }
                printf("ERRO RETA %.2f\n", erroreta);
                printf("ERRO CIRC %.2f\n", errocirc);
                printf("ERRO CURV %.2f\n", errocurv);
                printf("Nota Reta %.2f\n", pr/10);
                printf("Nota Circ %.2f\n", pc/10);
                printf("Nova Curv %.2f\n", pcur/10);
            }
        }
    }
    fprintf(arq, "\n</svg>");
    fclose(arq);
    getch();
    return 0;
}
```