UNIVERSIDADE ESTADUAL DO PIAUÍ – UESPI
CAMPUS PROF. ALEXANDRE ALVES OLIVEIRA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

JOSUÉ MACHADO MOTA

VISÃO COMPUTACIONAL NA INTERAÇÃO HOMEM-MÁQUINA

PARNAÍBA – PI
2010

JOSUÉ MACHADO MOTA

VISÃO COMPUTACIONAL NA INTERAÇÃO HOMEM-MÁQUINA

Monografia apresentada à Universidade Estadual do Piauí como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Alessandro Saraiva Mendonça

PARNAÍBA

2010

FICHA CATALOGRÁFICA ELABORADA PELO BIBLIOTECÁRIO
HERNANDES ANDRADE SILVA CRB-3/936

M917v Mota, Josué Machado
Visão computacional na interação homem-máquina / Josué Machado Mota. – Parnaíba: 2010.
49f : il.

Monografia apresentada ao Curso de Bacharelado em Ciência da Computação, Universidade Estadual do Piauí - UESPI, Parnaíba - 2010.
Orientador: Prof. Alessandro Saraiva Mendonça.

1. Visão Computacional. 2. Computação Gráfica. 3. Interface Homem-Máquina. I. Título.

CDD – 004.77

GOVERNO DO ESTADO DO PIAUÍ
UNIVERSIDADE ESTADUAL DO PIAUÍ – UESPI
CAMPUS PROF. ALEXANDRE ALVES DE OLIVEIRA
COORDENAÇÃO DO CURSO DE CIÊNCIA DA COMPUTAÇÃO

Ata de Apresentação de Trabalho de Conclusão de Curso

Aos dez dias do mês de setembro de dois mil e dez, às 09h30, na Sala 201 do Campus Prof. Alexandre Alves Oliveira - UESPI, na presença da Banca Examinadora, presidida pelo Prof. Alessandro Saraiva Mendonça e composta pelos membros efetivos os professores Francisco das Chagas Rocha e Daniel Lima Sousa, o aluno **Josué Machado Mota** apresentou o Trabalho de Conclusão de Curso intitulado **Visão Computacional na Interação Homem-Máquina**, como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciência da Computação. Aberta a sessão pública, o candidato teve a oportunidade de expor o trabalho. Após a exposição, o aluno foi arguido oralmente e avaliado em sessão reservada pelos membros da Banca, nos termos do Regulamento Geral dos Trabalhos de Conclusão de Curso da Universidade Estadual do Piauí, tendo obtido nota 9,0 (nove pontos). A Banca concluiu pela **aprovação** do candidato, **sem restrições**, resultado este divulgado ao aluno e demais presentes. Nada mais havendo a tratar, eu, Prof. Alessandro Saraiva Mendonça lavrei a presente ata que será lida e assinada por mim, pelos membros da Banca Examinadora e pelo candidato. Parnaíba(PI), 10 de setembro de 2010.

Banca Examinadora

Prof. Esp Alessandro Saraiva Mendonça, UESPI
Orientador

Prof. MsC Francisco das Chagas Rocha, UESPI

Prof. MsC Daniel Lima Sousa, UFPI

Candidato

Josué Machado Mota

"O cavalo prepara-se para a batalha, mas do SENHOR vem a vitória." Pv 21.31

AGRADECIMENTOS

Agradeço ao meu Deus em primeiro lugar pelo seu grande amor para comigo. Morreu por mim, mas ressuscitou para que não só eu, mas todo aquele que Nele crê seja salvo e tenha a tão almejada vida eterna.

Em segundo lugar, agradeço a Deus pela vida de todos que de alguma forma puderam me ajudaram. Em especial, à minha mãe por ter acreditado em mim. Agradeço a Deus também pela vida dos amigos que ganhei ao longo dèsses quatro anos de aventura, meu orientador é um deles. Por falar em amigos, também agradeço a Deus pela vida da minha namorada por sempre me ajudar em tudo que me proponho a fazer.

RESUMO

Este trabalho apresenta a visão computacional como uma área de pesquisa sendo utilizada no desenvolvimento de novas interfaces homem-máquina. No intuito de demonstrar na prática essa interação com o usuário fazendo uso de ferramentas livres, foi criado um sistema de desenho chamado DesenhArte, uma contração de "Desenhando Arte".

O sistema é executado em modo *fullscreen* e deve ser visualizado por um projetor a fim de que a mesma tela projetada seja capturada por uma *webcam*. Fazendo uso da visão computacional, o sistema extrai informações do usuário a partir das imagens obtidas de sua interação. Essa interação é feita através de um *laser*, quando o sistema detecta a luz emitida pelo *laser* sobre a projeção, algumas partículas são desenhadas no local onde ocorreu a detecção. Alguns botões foram criados e também são acionados através do *laser*.

O presente trabalho não visa à substituição de dispositivos convencionais tais como monitor, teclado e/ou *mouse*. O foco do mesmo é utilizar a visão computacional como um recurso a mais para a criação de novas interfaces homem-máquina.

Palavras-chave: Interação Homem-Máquina. Visão Computacional.

ABSTRACT

This paper presents computer vision as a research area being used to develop new human-machine interfaces. In order to practically demonstrate that interaction with the user using free tools, created a drawing system called DesenhArte, a contraction of "Drawing Art".

The system runs in fullscreen mode and should be viewed with a projector so that the same screen designed to be captured by a webcam. Making use of computer vision, the system extracts user information from the images obtained from their interaction. This interaction is done through a laser, when the system detects the light emitted by the laser on the projection, some particles are drawn from the place where detection. Some buttons were created and are also triggered by the laser.

This paper is not intended to replace conventional devices such as monitor, keyboard and/or mouse. The focus of it is to use computer vision as one more resource to the creation of new human-machine interfaces.

Keywords: Human-Machine Interface. Computer Vision.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS E ABREVIATURAS

2D – Bidimensional

3D – Tridimensional

API – *Application Programming Interface*

CAD – *Computer Aided Design*

CPU – *Central Processing Unit*

CRT – *Cathode Ray Tube*

CMYK – *Cyan Magenta Yellow blacK*

ENIAC – *Electronic Numerical Integrator and Computer*

GLUT – *GL Utility Toolkit*

GUI – *User Interface Graphics*

HSV – *Hue Saturation Value*

IHC – Interação Homem-Computador

LASER – *Light Amplification by Stimulated Emission of Radiation*

LCD – *Liquid Crystal Display*

OPENCV – *Open Source Computer Vision Library*

OPENGL – *Open Graphics Library*

PC – *Personal Computer*

RGB – *Red, Green and Blue*

RUP – *Rational Unified Process*

UML – *Unified Modeling Language*

SUMÁRIO

1 INTRODUÇÃO

Desde 1946, quando J. Persper Eckert e John Mauchly criaram o primeiro computador considerado totalmente eletrônico, o ENIAC, a computação tem evoluído tanto em eficiência quanto em velocidade. De fato, o ENIAC foi um grande avanço tecnológico, mas possuía sérias desvantagens; uma delas era seu enorme tamanho que diferente de hoje, não era semelhante a um livro, mas sim a uma sala ou quarto. O sucesso da computação, hoje, está ligado a diversas áreas, dentre elas encontra-se a usabilidade (KIRKLAND, 2010).

Segundo Hendrik Witt (2008) a Interação Homem-Computador (IHC) é uma área de pesquisa que tem por objetivo tentar fazer a comunicação homem e máquina tão natural quanto à dos seres humanos. Ao observar as figuras 1 e 2 respectivamente temos um exemplo real das diferenças de usabilidade entre o ENIAC de 1946 e um computador portátil da atualidade.

Figura 1: ENIAC
Fonte: CERUZZI, 2003

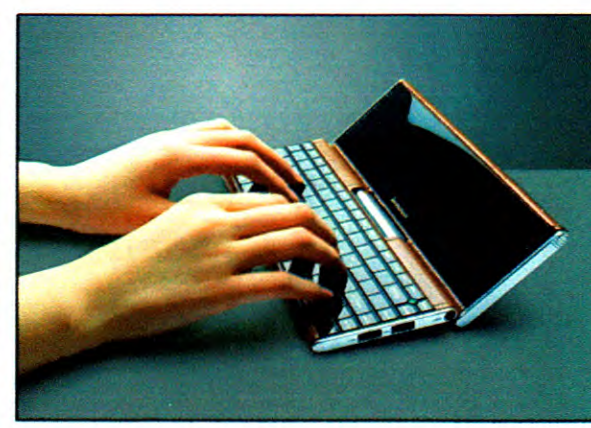

Figura 2: Netbook
Fonte: http://www.technobuzz.net/wp-content/uploads/2009/03/lenovopocketyoga-1.jpg

Diversos dispositivos foram criados a fim de facilitar essa interação com o usuário final; como exemplo temos o teclado, monitor e mouse que são tidos como dispositivos convencionais de interação pelo seu tempo de mercado. Hoje, dentre os mais recentes ficam as interfaces baseadas em gestos, rastreamento de objetos, reconhecimento de voz, fala sintetizada, etc. A criação de diversas interfaces de comunicação além de permitirem maior usabilidade, oferece acessibilidade a deficientes físicos, por exemplo. (TAN e NIJHOLT, 2010).

Uma interface amigável ou de fácil uso é tão importante hoje que é considerada uma etapa crucial no processo de desenvolvimento de qualquer novo *hardware* ou *software* de

mercado, afirma Hendrik Witt (2008). Facilidade de aprendizado, acesso rápido as funcionalidades e facilidade de lembrar como usar o sistema, são pontos chave dentre os objetivos da usabilidade (ROGERS, SHARP e PREECE, 2002).

Dentre os meios utilizados a fim de melhorar essa interação homem-máquina está à visão computacional – uma área de pesquisa que tem por principal objetivo à criação de máquinas que "vêem" (BRUNELLI, 2009). Sua base é o reconhecimento de padrões, ou seja, a partir de uma imagem obtida, a visão computacional tenta detectar padrões de cores, formas ou modelos pré-definidos buscando extrair algum conhecimento sobre a imagem e então poder tomar uma decisão adequada de forma automática.

1.1 OBJETIVOS E MOTIVAÇÃO

Este trabalho demonstra de forma prática e objetiva o uso da visão computacional como interação homem-máquina em aplicativos de tempo real. DesenhArte é um sistema de desenho, desenvolvido com o propósito de testar essas novas interfaces. Os principais objetivos são:

- Implementação de um sistema de desenho 2D.
- Implementação de um sistema de visão computacional.
- Integração dos sistemas/módulos de desenho e visão.

A principal motivação para a elaboração deste trabalho partiu de um estudo sobre ferramentas livres destinadas à visão computacional. O OpenCV não é o foco desde trabalho, entretanto é peça fundamental para a concretização do mesmo. A idéia é fazer uso da visão computacional de tal forma que a interação do usuário com o sistema computacional se torne mais amigável.

1.2 ESTRUTURA DO DOCUMENTO

O capítulo dois apresenta alguns trabalhos de pesquisa relacionados à visão computacional na interação homem-máquina, dentre eles está o projeto base na qual o presente trabalho foi desenvolvido. No terceiro capítulo é feita uma revisão de alguns conceitos básicos, porém importantes para o melhor entendimento e contextualização do mecanismo de interação proposto.

O desenvolvimento do sistema encontra-se com maiores detalhes no capítulo quatro e cinco. No capítulo de número quatro as ferramentas utilizadas no desenvolvimento são apresentadas destacando-se sua aplicabilidade no presente trabalho. O capítulo cinco apresenta a metodologia de software, a especificação e a implementação de cada protótipo desenvolvido. No sexto capítulo são feitas as considerações finais e alguns trabalhos futuros que podem ser desenvolvidos fazendo uso da visão computacional são sugeridos.

2 TRABALHOS RELACIONADOS

Nos últimos anos, muitos trabalhos relacionados à visão computacional e interação homem-máquina foram desenvolvidos. A PlayStation® atualmente, possui uma classificação destinada somente aos jogos que utilizam a visão computacional como meio de interação homem-máquina chamada PlayStation®Move (PLAYSTATION, 2010). Contudo, a idéia para a elaboração deste trabalho partiu de um projeto de Christopher Armenio, disponível em http://arsinio.googlepages.com/BTCP-FinalReport.pdf acessado em 29 de julho de 2010.

2.1 DESIGN OF A LASER CONTROLLED MOUSE USING OPENCV

Assim foi intitulado o projeto de Christopher. Seu projeto teve por base a criação de uma interface baseada em visão – ou seja, que faz uso da visão computacional – capaz de controlar o ponteiro do mouse utilizando apenas um *laser*, a figura a seguir demonstra o esquema físico de funcionamento de seu projeto.

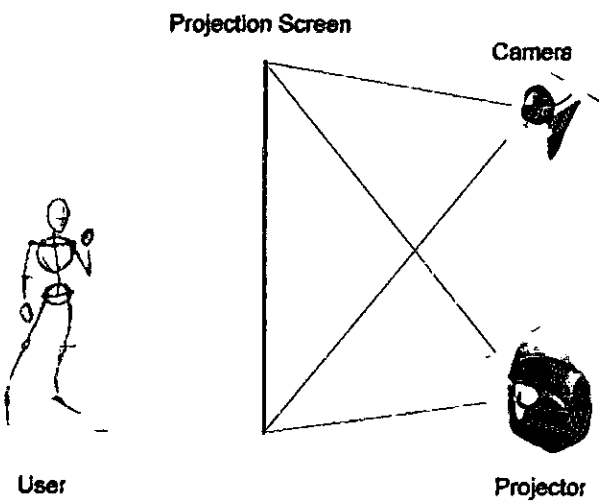

Figura 3: Modelo físico para utilização do sistema
Fonte: http://arsinio.googlepages.com/BTCP-FinalReport.pdf

Em seu sistema, o projetor exibe a tela de seu *desktop* e uma *webcam* é utilizada para capturar a imagem projetada. Uma vez capturada a imagem, o sistema faz uso de um filtro de processamento de imagem tendo em vista detectar a área atual onde o usuário está apontando o *laser*, ou seja, busca pelo ponto de luz emitida pelo *laser* e assim, após uma conversão de coordenadas entre a resolução da imagem capturada e a resolução utilizada no *desktop*, posiciona o *mouse* para o local onde foi detectado o ponto de luz.

Baseado no projeto de Christopher, DesenhArte também é um sistema baseado em visão que tem por base os mesmos equipamentos de *hardware* e a biblioteca OpenCV em *software*, entretanto difere em suas funcionalidades.

Christopher em sua implementação teve por objetivo detectar somente um ponto de luz emitido pelo *laser* – afinal, para cada *desktop* existe um único ponteiro (*mouse*) – ou seja, não permitia a interação de vários usuários ao mesmo tempo. No presente projeto a interação múltipla em tempo real é perfeitamente válida. Outra principal diferença é que DesenhArte além de ser um sistema de visão também é um sistema de computação gráfica, uma vez que a luz age como um "pincel mágico" sobre a tela projetada e o sistema de desenho cria partículas "vivas" onde for detectada a presença da luz emita pelo *laser*.

2.2 INTERAÇÃO HOMEM-MÁQUINA UTILIZANDO GESTOS

Como citado anteriormente, muitos são os trabalhos relacionados à visão computacional como mais uma forma de interação homem-máquina. No trabalho de Michel Alain (TRUYENQUE, 2005) intitulado de "Uma Aplicação de Visão Computacional que Utiliza Gestos da Mão para Interagir com o Computador" é apresentado um estudo teórico-prático sobre algoritmos de detecção de gestos e detalhes sobre os protótipos por ele implementado.

A visão computacional além de auxiliar na detecção de objetos torna possível a criação de dispositivos de interação com computadores mais intuitivos e rápidos. Uma das grandes vantagens encontrada nos sistemas de visão está na ausência de dispositivos de interação, seja ele mecânico, magnético ou óptico (TRUYENQUE, 2005). A idéia principal contida no projeto de Michel é a detecção da mão e do posicionamento de cada dedo a fim de serem representados como uma entrada de dados válida por seu sistema de visão desenvolvido.

3 FUNDAMENTAÇÃO TEÓRICA

Nesta parte do trabalho são descritos alguns conceitos essenciais encontrados na pesquisa sobre visão computacional utilizada como forma de interação homem-máquina.

3.1 IMAGEM DIGITAL

Na visão computacional, a imagem digital é a principal informação a ser analisada. Uma imagem digital nada mais é do que um vetor bidimensional de *pixels*. Estes podem ser exibidos em um monitor CRT ou LCD, copiados para a memória por um aplicativo ou armazenados em arquivo para futuramente serem manipulados (REYNOLDS e BLYTHE, 2005).

Importante ressaltar que a quantidade de *pixels* tanto na vertical quando na horizontal (resolução) devem ser observados com bastante atenção uma vez que todas as operações de baixo-nível na visão computacional são feitas neles. Outro fator importante é sua organização.

3.2 RGB

Existem alguns modelos na qual os *pixels* podem ser organizados, um deles é pelo RGB que representa um *pixel* pela adição de três componentes de cor: vermelho, verde e azul. Essas três cores são combinadas para produzir uma cor resultante. No caso de imagem digital em modelo RGB, cada componente de cor possui um único valor que varia de 0 à 255, normalmente. Assim, para obter um vermelho puro em RGB atribuímos o valor 255 para a componente vermelha e 0 (zero) para as demais, verde e azul (MONTABONE, 2010).

Existem outros modelos como, por exemplo, o CMYK e o HSV, porém de longe o mais utilizado em aplicativos gráficos e câmeras digitais é o RGB. A figura 4 mostra uma representação bastante utilizada onde cada componente equivale a um eixo 3D.

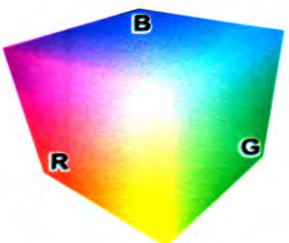

Figura 4: Cubo RGB
Fonte: primária

3.3 INTERAÇÃO HOMEM-MÁQUINA

Segundo Djeraba, Laback e Benabbas (2010), a IHC é o estudo de interações entre computadores e seus usuários. Nesta interação entre usuário e máquina é criada uma interface com o usuário (ou simplesmente, interface) que faz a comunicação entre eles através de dispositivos de entrada e saída de dados.

A grande maioria dos profissionais da área de computação ao iniciar qualquer projeto, um de seus objetivos é tornar a interface com o *software* transparente ao usuário, ou seja, tenta esconder toda a complexidade por trás do sistema e fazer essa comunicação mais fácil possível (REDONDO, BRAVO e ORTEGA, 2009).

3.4 INTERFACES MULTIMODAL

No mundo real, quando interagimos com um objeto, informações são transmitidas ao nosso cérebro simultaneamente por diferentes sentidos, o que podemos dividir em algumas modalidades: visual, auditiva, sensitiva, etc. Essa é a idéia básica da interface multimodal.

A interface de um sistema é considerada multimodal quando uma interação pode ser processada por duas ou mais interfaces e permitir uma interação multiusuário real. Uma das principais vantagens dessas interfaces é a coleta de certas informações que de maneira alguma poderiam ser obtidas por dispositivos convencionais como mouse ou teclado. Por exemplo, como saber se o usuário está próximo ou não do computador? Como saber se este está sozinho ou acompanhado por outras pessoas? Estas são algumas das situações possíveis graças às técnicas de visão computacional (REDONDO, BRAVO e ORTEGA, 2009).

Com a interface multimodal abre-se um novo mundo repleto de possibilidades onde idéias novas surgem a cada dia. Uma das promessas é tornar a informática acessível aos deficientes físicos, uma vez que essas interfaces se baseiam em várias modalidades de interação, o que certamente irá facilitar a vida de muitas pessoas com necessidades especiais, como por exemplo, um cego pode não utilizar um monitor, mas pode interagir com interfaces de fala sintética e de reconhecimento de fala e/ou gestos utilizando a visão computacional.

Um exemplo de interface multimodal que faz parte do nosso cotidiano são os aparelhos celulares que fazem discagem através de teclas ou por comandos de voz. Simples, porém de grande utilidade já que interagir com o teclado requer bem mais atenção do que simplesmente falar e próprio celular se encarregar de pesquisar o contato e fazer a ligação.

3.5 COMPUTAÇÃO GRÁFICA

A Computação Gráfica é uma ferramenta de concepção de arte, assim como um pincel ou instrumento musical. Porém, esta ferramenta proporciona um maior poder de abstração, ajudando na criação de imagens complexas e em muitos casos antes não imaginadas. Por definição, a computação gráfica trata-se de um conjunto de modelos, métodos e técnicas para transformar dados em imagens através de um dispositivo gráfico (VELHO, CARVALHO, FIGUEIREDO e GOMES, 2008).

Figura 5: Transformação de dados em imagens
Fonte: primária

Um exemplo simples e clássico da computação gráfica são os jogos eletrônicos. O ambiente de todo o jogo é armazenado em arquivos de dados que uma vez carregados para a memória principal fazem uso intenso da computação gráfica para que o mundo virtual 3D seja criado a partir dos dados armazenados. Hoje, graças à evolução tanto do *hardware* quanto do *software*, os resultados obtidos têm um grau de realismo cada vez maior.

3.6 VISÃO COMPUTACIONAL

Segundo Kirland (2010), a fabricação de máquinas que imitam certos aspectos da capacidade e inteligência humana é a idéia por trás da Ciência da Computação. Se tivermos a coleta de informações como área de maior interesse, um dos aspectos que devem ser observadas no ser humano é a visão, uma vez que 75% de todas as informações adquiridas ao longo do tempo de vida de uma pessoa são constituídas por informações visuais (MEDIONI e KANG, 2004).

Assim, a visão computacional tenta imitar essa rica fonte de dados, uma visão implementada não de forma restrita ou previamente configurada, mas de tal forma que seja capaz de interpretar o nosso mundo de maneira satisfatória. Porém, não é uma tarefa simples uma vez que ocorrem constantes mudanças físicas a cada dia em nosso mundo.

> Em alguns casos, a extração e a interpretação dos dados se tornam bastantes complexas, necessitando diversas etapas intermediárias. Filtros podem ser aplicados para a remoção de ruído ou para o realce de determinadas características da imagem; técnicas de detecção de bordas e segmentação de imagens podem ser utilizadas para isolar objetos de interesse, facilitando a extração de propriedades relacionadas à forma, tamanho, posição e cor dos mesmos (SCHRAMM, 2006).

3.5.1 FILTROS

O tipo de informação a ser extraída de uma imagem diz respeito ao tipo de problema a ser solucionado da melhor forma possível (JÄHNE e HAUBECKER, 2000). Como citado anteriormente por Schramm, os filtros são algoritmos de processamento de imagem que visam justamente realizar operações sobre uma dada imagem para extrair alguns atributos contidos na mesma tais como: forma, cor ou textura de objetos contidos na cena.

Os filtros implementados na visão computacional podem ser aplicados em uma imagem de forma individual ou por seqüência de imagens. Um filtro para detecção de cor, por exemplo, é aplicado a cada imagem de forma individual, porém para a detecção de movimento, faz-se necessário no mínimo de duas imagens para que seja possível fazer a diferença entre os *pixels* de cada imagem e obter a diferença. Os filtros ocupam boa parte dos recursos implementados por uma biblioteca de visão e podem ser utilizados até mesmo para a criação de filtros novos.

3.5.2 ÁREAS RELACIONADAS

A visão computacional relaciona-se diretamente com três áreas: computação gráfica, processamento digital de imagens e processamento de dados. Na figura 6, trata-se de uma representação gráfica que demonstra bem essa relação. Brito (2009) comenta sobre essa relação afirmando que:

> A Visão Computacional, juntamente com a Computação Gráfica e o Processamento de Imagens constituem o tripé no qual se fundamentam os processos computacionais que envolvem imagens. Enquanto a Visão Computacional obtém modelos geométricos a partir de imagens digitais, a Computação Gráfica trata da transformação de modelos geométricos em imagens digitais. Ambas atuam com os mesmos tipos de dados, só que em direções opostas. Já o Processamento de Imagens lida com técnicas de transformações em imagens digitais.

Figura 6: Áreas relacionadas à visão computacional
Fonte: BRITO, 2009

Muitas outras áreas também estão envolvidas com a visão computacional, Jain e Kasturi (1995) já consideravam em sua época a inteligência artificial e o reconhecimento de padrões como áreas de interesse no desenvolvimento de sistemas de visão.

3.5.3 DESAFIOS

A maior dificuldade encontrada na implementação de sistemas baseados em visão é o fato de que uma imagem digital é a representação 2D de um espaço 3D, ou seja, a informação contida na imagem em sua grande maioria pode não ser restaurada em sua totalidade (JAIN e KASTURI, 1995). De certa forma, Chen e Wang (2005) confirmam o que foi dito afirmando que a detecção de objetos talvez seja a principal problemática da visão computacional.

Nós, seres humanos, temos a capacidade de reconhecer objetos sem muito esforço. Já para os sistemas de visão, o reconhecimento não é uma tarefa tão simples. Pela limitação do espaço tridimensional ser representado por um bidimensional, um mesmo objeto em repouso pode ter infinitas representações 2D dependendo do posicionamento do observador (ou câmera), uma vez que algumas transformações como rotação e translação, por exemplo, já resultam em incontáveis possibilidades de imagens diferentes.

O fato é que diversos fatores como iluminação e movimento dos objetos devem ser analisados para que a execução de um sistema baseado em visão seja bem sucedida no processo de reconhecimento.

3.6 INTERFACES DE PERCEPÇÃO

Surgida nos últimos anos, a Interface de Percepção visa integrar várias modalidades perceptivas (como visão computacional, processamento de fala, sensores de toque, etc.) para a interface do usuário. Para a tecnologia de visão computacional, em particular, o principal objetivo é usar a visão como uma modalidade de entrada efetiva na interação homem-máquina. Essas interfaces são consideradas altamente interativas que permitem interação multimodal de forma natural e eficiente entre o computador e o usuário final. De fato, a computação está presente em nossa vida diária em diversos momentos e lugares; visando facilitar essa comunicação comumente utilizada, as interfaces têm se tornado cada vez mais naturais, intuitivas, adaptáveis e discretas para uso mais eficiente e simples. Estes são os principais objetivos da pesquisa em interfaces de percepção, assim afirma Medioni e Kang (2004).

A visão computacional de fato possui um grande potencial na criação de novas interfaces centradas no usuário, já que muitas informações podem ser extraídas de imagens. Os sistemas baseados em visão possuem a grande vantagem de não necessitar de sensores ligando o usuário ao computador, como é o caso do mouse que normalmente é ligado à máquina por meio de um fio, o que força o usuário a dedicação de uma mão exclusiva para o uso do mesmo no local instalado.

Nas interfaces de percepção, a visão computacional possui um alto grau de importância uma vez que o usuário pode interagir de diversas formas com o sistema, seja por meio de gestos ou até mesmo um piscar de olhos. Qualquer movimento físico pode ser detectado para iniciar a execução de uma tarefa previamente configurada (AGHAJAN, DELGADO e AUGUSTO, 2010).

Alguns paradigmas de interface com o usuário tem se destacado ao longo da história da computação. O quadro 1 descreve uma visão geral dessa evolução.

Década	Implementação	Paradigma
1950's	Cartões Perfurados	Nenhum
1970's	Interface Linha de Comando	Máquina de Escrever
1980's	Interface Gráfica	Desktop
2000's	Interface de Percepção	Interação Natural

Quadro 1: Evolução dos paradigmas de interface com o usuário
Fonte: AGHAJAN, DELGADO e AUGUSTO, 2010

Nos primórdios da computação, não havia um modelo real de interação – os dados eram inseridos no computador através de chaves ou cartões perfurados e a saída era produzida algum tempo depois, através de cartões perfurados ou luzes. A segunda fase começou com a chegada de interfaces de linha de comando no início dos anos 1960. Na década de 1970 e 1980, a interface gráfica com o usuário e o ambiente desktop surgiram e atualmente, as interfaces de percepção originaram-se da tentativa de tornar a interface com o usuário mais natural possível, aproveitando as maneiras pelas quais as pessoas naturalmente interagem uns com os outros, seja de forma verbal ou não (MEDIONI e KANG, 2004).

4 FERRAMENTAS

Nesta seção todas as ferramentas utilizadas no desenvolvimento deste trabalho são apresentadas. A princípio, o sistema teve sua implementação sob a plataforma Windows, entretanto, pelo fato do projeto utilizar somente ferramentas livres e multiplataforma optou-se pela utilização de um sistema operacional também livre, no caso, o Linux. Uma breve descrição sobre cada ferramenta utilizada é feita de forma sucinta, tentando sempre destacar sua importância no presente trabalho.

4.1 LINGUAGEM C++

A linguagem de programação C++ é baseada na linguagem de programação C, por isso muitos a chamam de "C melhorado" uma vez que "++" é o operador de incremento na linguagem C. De fato, C++ possui diversas funcionalidades que não estavam presentes no C, por exemplo, a criação de dois novos paradigmas além do procedural, são esses à programação orientada a objetos e programação genérica (JUNEJA e SETH, 2009).

Desde sua criação nos anos 80 por Bjarne Stroustrup, a linguagem de programação C++ tem sido a escolhida quando o objetivo é a criação de programas velozes (SOLTER e KLEPER, 2005). A principal razão pela escolha do C++ como a linguagem de implementação de *software* segundo Gecks (2009) se dá pelo simples fato desta linguagem permitir a criação de *software* complexo mais eficiente do que utilizando qualquer outra linguagem de alto nível. Juneja e Seth (2009) fazem referência a uma lista de aplicações em que C++ é utilizado como linguagem principal, dentre elas estão os sistemas operacionais, programas CAD, aplicações de tempo real, jogos e animações 2D e 3D.

O presente trabalho apresenta animações que representam o desenho criado pelo usuário e ao mesmo tempo possui um controle em tempo real que constantemente busca identificar a presença de luz sobre cada imagem captura pela *webcam*. Levando em conta que aplicar um filtro na imagem capturada é um trabalho custoso, uma vez que mesmo tendo a resolução em 640x480 no modelo RGB teremos 921.600 componentes de cor para serem analisadas em cada imagem. Neste sentido, a linguagem C++ foi escolhida por proporcionar uma maior eficiência na execução do programa para que o usuário tenha realmente a sensação de está desenhando com o *laser*, ou seja, o tempo de resposta precisa ser rápido.

4.2 OPENCV

OpenCV pode ser interpretado no português brasileiro como "biblioteca de código aberto para visão computacional". A biblioteca foi desenvolvida originalmente pela Intel® no ano de 1999 em C e C++ visando eficiência e tendo por foco o desenvolvimento de aplicações de tempo real. Possui permissão de uso livre para aplicações de código aberto ou não. Hoje, o projeto é disponível pela *Source Forge*[1]. Além de ser uma ferramenta multiplataforma, ou seja, utilizada em Linux, Windows, Mac OS X e outros sistemas operacionais, há interfaces intermediárias para desenvolvimento em outras linguagens como: Python, Ruby e Java, por exemplo (BRADSKI e KAEHLER, 2008). Neste trabalho foi utilizada a interface nativa do OpenCV por ser mais rápida, uma vez que a comunicação é feita de forma direta com o C++.

Uma das idéias do OpenCV segundo Bradski e Kaehler (2008) é prover uma infra-estrutura de uso simples para visão computacional que auxilie no desenvolvimento rápido de aplicações baseadas em visão. Atualmente a biblioteca conta com mais de 500 rotinas de uso geral que atende todas as etapas de uma aplicação baseada em visão, incluindo comunicação com *webcam*, processamento de imagem por meio de filtros, componentes de interface gráfica com o usuário e até mesmo rotinas simples de desenho para escrever textos ou primitivas geométricas nas imagens manipuladas tais como: linhas, círculos, retângulos, etc. De fato, desde a entrada (comunicação com a *webcam*) até a saída de dados (interface com o usuário) o OpenCV se faz presente, isto demonstra o potencial desta rica biblioteca.

Neste trabalho a biblioteca OpenCV teve bastante utilidade no processo de comunicação do sistema DesenhArte com a *webcam* e na extração das imagens obtidas pela mesma. Medioni e Kang (2004) relatam que OpenCV também possui rotinas de baixo-nível a fim de possibilitar ao programador um maior controle até mesmo sobre os *pixels* de forma individual nas imagens. Tendo por objetivo uma maior eficiência foram utilizadas justamente essas rotinas de baixo-nível para obter todos os *pixels* de cor branca com grau de pureza entre 90% a 100%. Essa detecção sendo feita em baixo-nível é possível repassar no mesmo instante da detecção da luz do *laser* a localização do *pixel* encontrado ao OpenGL para que as partículas de desenho sejam criadas no local de detecção.

A utilização de um único filtro contendo todo o processamento é altamente recomendado quando se deseja eficiência em execução. De outra forma, utilizar vários filtros significa percorrer todos os *pixels* da mesma imagem várias vezes.

---

[1] http://sourceforge.net/projects/opencvlibrary/

4.3 OPENGL

Assim como o OpenCV, o OpenGL também é uma biblioteca aberta escrita na linguagem de programação C porém, aberta não em código fonte, mas em arquitetura (especificação) e licença de uso. Em essência, OpenGL é uma API gráfica 3D de baixo-nível multiplataforma para o desenvolvimento de aplicações 3D de tempo real, tais como jogos eletrônicos e simuladores (WRIGHT e LIPCHAK, 2004).

> OpenGL também é definida como uma 'interface para hardware gráfico'; é uma biblioteca de rotinas gráficas e de modelagem 2D e 3D extremamente portável e rápida. Ela permite o desenvolvimento de aplicativos 3D com um alto grau de realismo. Entretanto, a sua maior vantagem é a velocidade, uma vez que incorpora vários algoritmos otimizados [...] (COHEN e MANSSOUR, 2006)

Para garantir toda essa portabilidade em *hardware*, dentre todas suas rotinas gráficas (mais de 700 rotinas distintas) nenhuma foi projetada para o controle de janelas ou tratamento de eventos, isto tem grande importância já que o sistema de janelas está intimamente ligado ao sistema operacional (SHREINER, 2009). Hoje, o OpenGL é tido como um padrão gráfico, isso quer dizer que em qualquer sistema, por padrão, o OpenGL está presente; até mesmo no Windows que possui sua própria biblioteca gráfica, o Direct X. Semelhantemente ao OpenCV, OpenGL também possui interfaces intermediárias para desenvolvimento em outras linguagens como Python, Java, Ruby, etc. Neste trabalho, foi utilizada sua interface nativa por ser mais rápida. Na verdade, a linguagem C++ foi escolhida por dois motivos: C++ é uma linguagem rápida e as bibliotecas OpenCV e OpenGL são escritas em C/C++ o que torna a integração entre as mesmas mais fácil e eficiente.

4.3.1 GLUT

A biblioteca GLUT é um conjunto de ferramentas utilitárias para o OpenGL responsável pelo gerenciamento de janelas e tratamento de eventos. É bastante utilizada em aplicações de pequeno e médio porte, não por fazer parte do padrão OpenGL pois não faz, mas por seguir as convenções de nomenclatura do OpenGL, ser multiplataforma e de fácil aprendizado. Com a utilização da GLUT é possível escrever código elegante com ótima portabilidade para manipulação de janelas e tratamento de eventos de forma fácil para diversos sistemas operacionais.

> OpenGL trata da exibição de objetos gráficos, não possuindo, entretanto, rotinas específicas para o tratamento de eventos de teclado, ou de mouse, gerenciamento de janelas gráficas e outras funções não-ligadas diretamente à exibição das imagens. Com isso, a integração entre o OpenGL e os diversos sistemas operacionais, nos quais esta poderia ser utilizada, acabava reduzindo a portabilidade dos programas [...] (COHEN e MANSSOUR, 2006)

De fato, o OpenGL é uma biblioteca ou interface de programação (API) especializada somente em "desenhar". Todos os resultados de seus processos são armazenados na memória RAM e para exibir esses dados é necessário uma janela utilizando um contexto OpenGL para que esses dados armazenados sejam vinculados à janela e exibidos. Na prática, esse processo é trabalhoso, pois devem ser feitas algumas chamadas de sistema com diversos parâmetros para que o sistema operacional crie e configure esse contexto OpenGL em uma determinada janela. A GLUT abstrai todas essas chamadas de sistema, uma vez que todas as janelas criadas por ela já são previamente configuradas para comunicar-se com o OpenGL. Na figura 7 é mostrado um pequeno código de exemplo que cria a janela demonstrada na figura 8.

```
 1: #include <GL/glut.h>
 2:
 3: void draw() {
 4:    glClear(GL_COLOR_BUFFER_BIT);
 5:    /* aqui as rotinas de desenho do OpenGL*/
 6:    glFlush();
 7: }
 8:
 9: int main(int a, char **b) {
10:    glutInit(&a, b);
11:    glutCreateWindow("Uma Janela com  GLUT");
12:    glutDisplayFunc(draw);
13:    glutMainLoop();
14:    return 0;
15: }
```

Figura 7: Código para uma janela com GLUT
Fonte: primária

Figura 8: Janela GLUT
Fonte: primária

A janela exibida na figura 8 possui nada mais do que um plano de fundo preto, porém, a mesma encontra-se pronta para exibir qualquer rotina gráfica do OpenGL. Explicando um pouco o código da figura 7 temos: na linha 10 a inicialização da GLUT, linha 11 a janela é criada, já na linha 12 é registrado o endereço da função responsável por executar as rotinas de desenho e na linha 13 a GLUT entra em um *loop* "infinito" e sempre que precisar redesenhar a janela chama a função registrada.

5 IMPLEMENTAÇÃO

Nesta parte do trabalho é apresentada a metodologia, especificação e a implementação do sistema desenvolvido.

5.1 METODOLOGIA DE SOFTWARE

A metodologia utiliza para o desenvolvimento do sistema foi o RUP, também bastante conhecida como Processo Unificado. Visando uma forma organizada de trabalhar, o sistema foi desenvolvido tendo como base as quatro fases do processo unificado: concepção, elaboração, construção e transição.

> A fase de concepção incorpora o estudo de viabilidade e uma breve análise de requisitos é feita tentando conhecer o problema como um todo. A fase de elaboração incorpora a maior parte da análise de requisitos, a análise de domínio e o projeto. A fase de construção corresponde à programação e testes, e a fase de transição consiste na instalação e manutenção do sistema desenvolvido (WAZLAWICK, 2004)

Importante ressaltar que na fase de elaboração e construção são feitos ciclos interativos para cada funcionalidade do sistema, sendo feita uma análise de requisitos mais aprofundada seguida de implementação e testes. Na figura 9 temos as fases do RUP destacando seus ciclos interativos.

Figura 9: Ciclos interativos do RUP
Fonte: primária

A idéia de ciclos interativos é bastante útil uma vez que o trabalho pode ser facilmente dividido entre equipes e na presença de erros, os mesmos são imediatamente detectados e corrigidos. Isto faz com que o programa em sua fase final (transição), que se inicia logo após o último ciclo interativo, tenha uma maior qualidade (WAZLAWICK, 2004).

O sistema desenvolvido seguiu uma metodologia orientada a protótipos incrementais, ou seja, para a criação de um novo protótipo, partia-se de um protótipo anterior como base e incrementava-se com novas funcionalidades. Essa metodologia foi adotada devido à equipe de desenvolvimento ser constituída de apenas um desenvolvedor e pelo fato do sistema ser de caráter inovador.

No primeiro protótipo foi implementada a comunicação entre o sistema de desenho e o filtro de luz (detecção da luz emitida pelo *laser*). No segundo, foi implementada o desenho de pontos verdes sobre o local detectado pelo filtro. No terceiro, os pontos verdes foram substituídos por partículas que se movimentam. Na quarta fase, foram criados dois botões que também são utilizados através do *laser* e finalmente, no quinto e último protótipo, a imagem de fundo foi substituída por um plano azulado e a imagem gerada é exibida no mesmo espaço capturado pela *webcam*, este modelo físico de montagem dos equipamentos são apresentados com detalhes a seguir na especificação do sistema.

5.2 ESPECIFICAÇÃO

Neste tópico é feita uma apresentação geral do sistema, abordando seus aspectos técnicos e funcionais, sua arquitetura e alguns diagramas UML contendo informações relevantes sobre o sistema.

5.2.1 DESCRIÇÃO DO SISTEMA

O sistema desenvolvido utiliza a visão computacional como interface base de comunicação entre o usuário e a tela de desenho. Na verdade, as interações do usuário são reconhecidas através de qualquer fonte de luz, o *laser* foi escolhido por ser uma fonte de luz direcional com boa precisão e boa popularidade. O usuário aponta o *laser* sobre a tela projetada do sistema e através de uma *webcam* a qual é controlada pela biblioteca de visão computacional é utilizado um filtro de cor a fim de capturar todos os pontos considerados brancos sobre a imagem, ou seja, a região supostamente iluminada pelo *laser*.

A computação gráfica é utilizada para desenhar as partículas "vivas" que se movimentam no local onde o ponto de luz foi encontrado. As partículas de desenho se movimentam sobre o ponto de luz onde elas foram originalmente detectadas dando um efeito artístico, assim o *laser* para o usuário passa a ser uma espécie de "pincel mágico". Por esse motivo, o sistema foi batizado de DesenhArte, uma contração de "Desenhando Arte".

Tendo por objetivo lembrar uma interface gráfica real com o usuário, também foi implementado um componente bastante comum em qualquer GUI, o botão. Semelhantemente à forma de desenhar, o usuário deve fazer uso do *laser* para acionar o botão. Para cada imagem capturada é aplicado o filtro de cor o qual encontrando a luz sobre o botão

incrementa-se uma variável interna e decrementa-se esta caso a luz não seja detectada. Assim, o usuário ao fixar o *laser* sobre o botão, a variável é incrementada continuamente e ao atingir uma pontuação máxima pré-definida o botão é acionado. Como exemplos foram criados dois botões, sendo um para limpar a tela de desenho e outro para fechar a aplicação.

5.2.2 REQUISITOS DE HARDWARE

O computador que executará o sistema desenvolvido deve possuir os seguintes requisitos mínimos de *hardware*:

- *Notebook* (ou PC) possuindo memória de 512 MB e CPU de 2 GHz
- *Laser* convencional (vermelho)
- *Webcam* com resolução 640x480
- Projetor do tipo *datashow*
- Tripé para a *webcam* (opcional)
- Tela para *datashow* (opcional)

A organização física desses equipamentos é extremamente importante para funcionamento adequado do sistema uma vez que a área capturada pela *webcam* deve ser a mesma ocupada pela projeção do *datashow*. A figura 10 apresenta com detalhes essa organização.

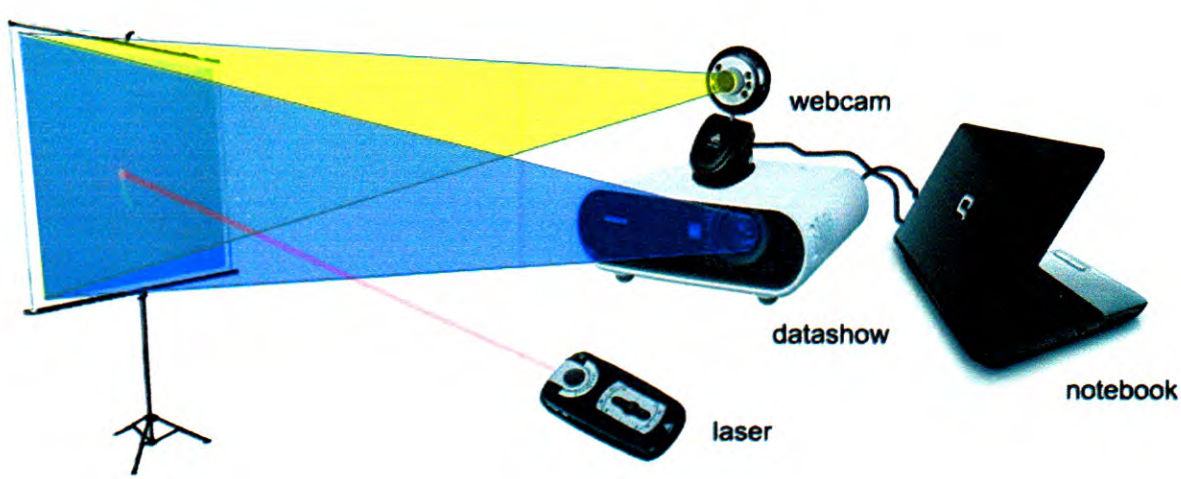

Figura 10: Organização física dos equipamentos
Fonte: primária

5.2.3 REQUISITOS DE SOFTWARE

O sistema operacional utilizado para o desenvolvimento deste trabalho foi o Ubuntu[2]. A única dependência de *software* são as bibliotecas utilizadas (OpenCV, OpenGL e GLUT) que devem está presente na máquina que se deseja executar o sistema.

5.2.4 ARQUITETURA DO SISTEMA

O sistema pode ser dividido em dois módulos principais, um relacionado à detecção de luz por meio da visão computacional e outro com o sistema de desenho utilizando a computação gráfica. A figura 11 ilustra melhor essa divisão:

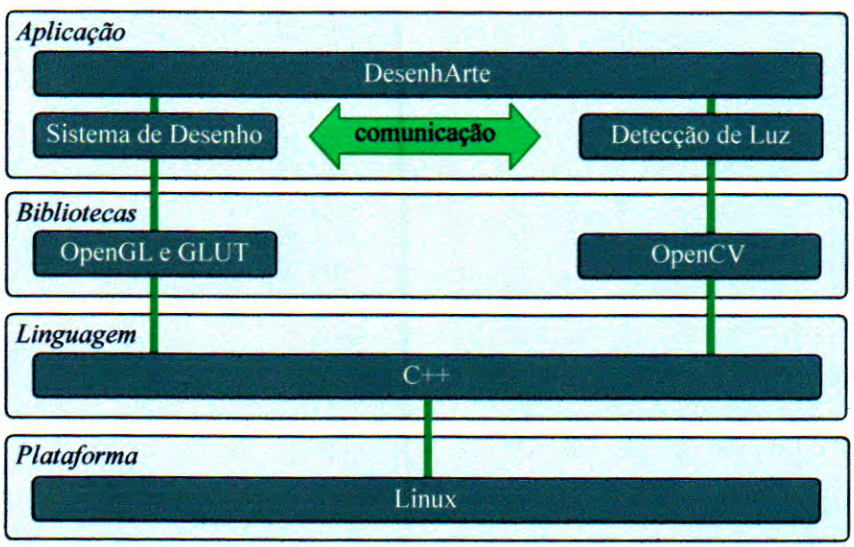

Figura 11: Diagrama da Arquitetura do Sistema
Fonte: primária

No diagrama da figura 11, observa-se que o sistema é claramente dividido em quatro camadas. Na primeira temos a plataforma na qual o sistema foi desenvolvido, porém é perfeitamente válido outro sistema operacional uma vez que todas as bibliotecas utilizadas são livres e multiplataforma. A camada dois temos a linguagem C++, aqui também é permitido o uso de outras alternativas como a linguagem Python, Ruby ou Java, por exemplo. A terceira camada refere-se às bibliotecas, no caso, OpenGL e OpenCV que são utilizadas em diversos projetos livres e escrita em C e C++. No topo temos a camada da aplicação que é constituída pelos módulos de desenho, detecção de luz e rotinas extras utilizadas para fazer a comunicação entre os dois módulos.

[2] http://www.ubuntu.com

5.2.5 DIAGRAMAS

Como forma de documentar a implementação realizada no presente trabalho foram criados dois diagramas UML buscando apresentar sua modelagem e fluxo de funcionamento interno do sistema da forma mais clara possível.

Apesar da linguagem C++ não ser uma linguagem totalmente orientada a objetos, optou-se pela utilização desse paradigma como alicerce na criação do sistema. De fato, a simplicidade na escrita/leitura de código foi o motivo de tal escolha. Na figura 12 é apresentado os detalhes da modelagem utilizada por meio de um diagrama de classe.

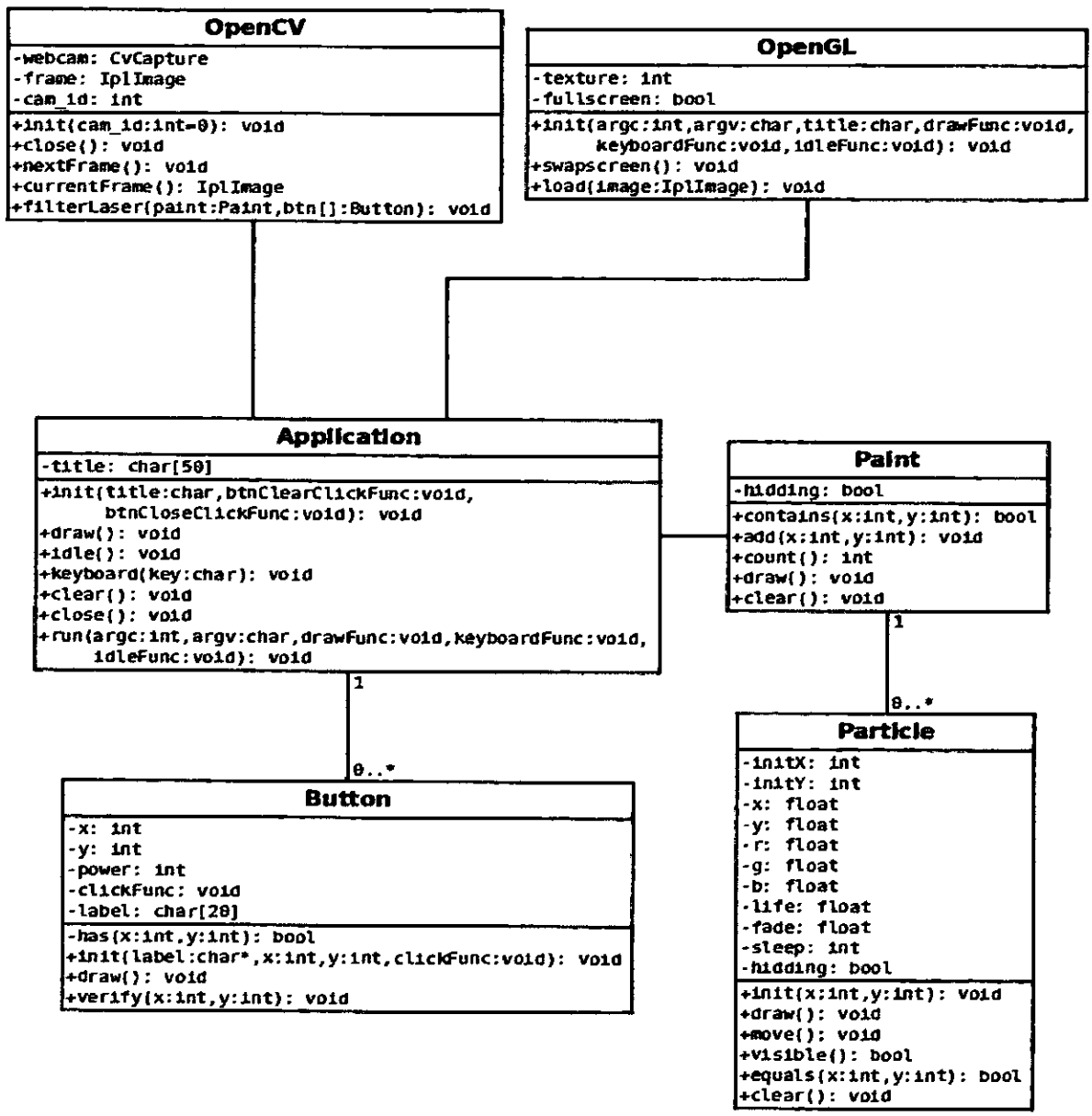

Figura 12: Diagrama de Classe
Fonte: primária

Já na figura 13 são apresentadas todas as operações realizadas pelo sistema desde seu início até seu processo de finalização. O diagrama de atividade é de importância fundamental uma vez que demonstra todos os processos envolvidos no ciclo de vida da aplicação, bem como a ordem em que os mesmos são executados.

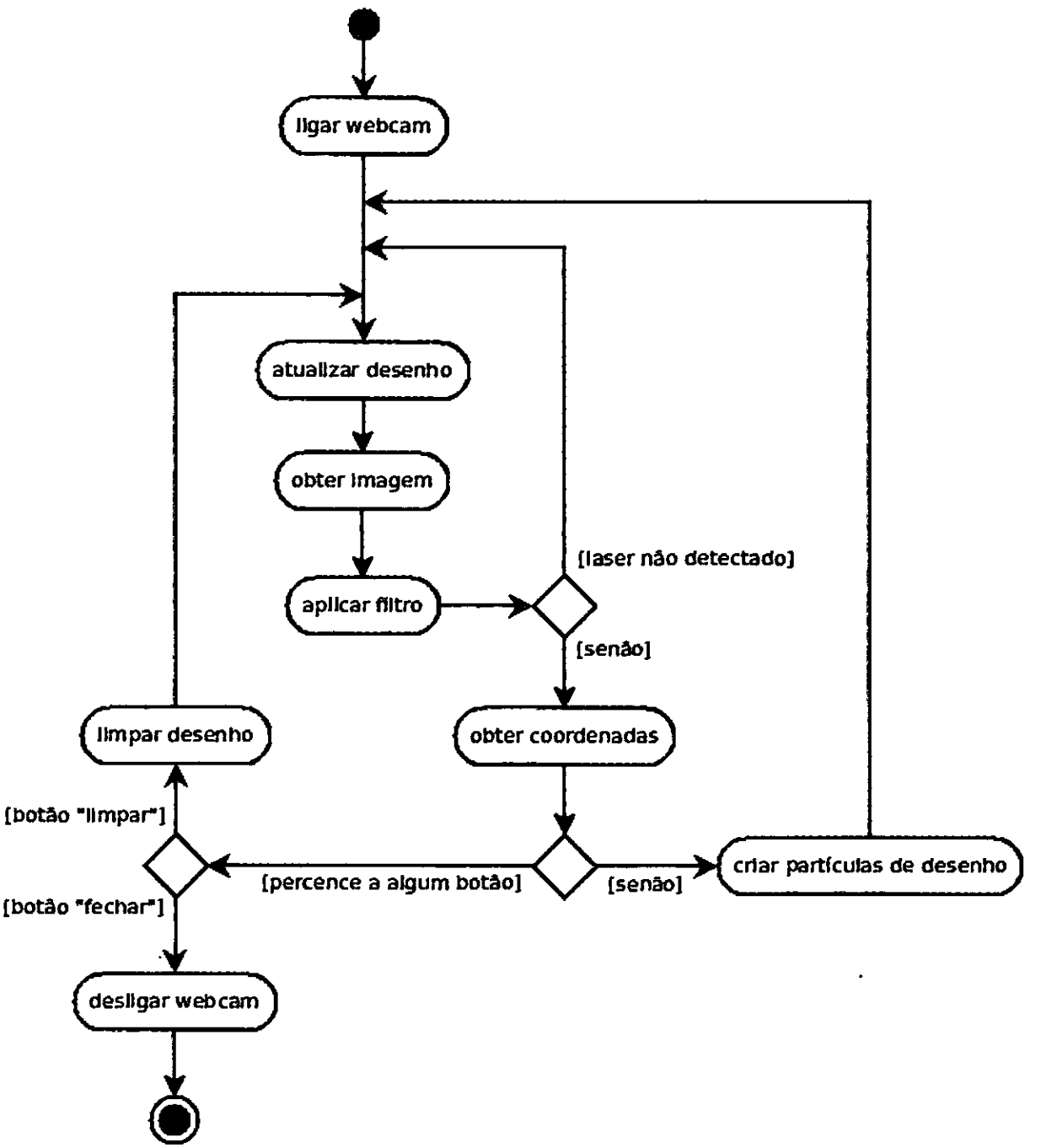

Figura 13: Diagrama de Atividade
Fonte: primária

5.3 IMPLEMENTAÇÃO

Nesta seção serão apresentados os protótipos criados para a implementação das funcionalidades do sistema desenvolvido. Os protótipos foram criados na seqüência em que são apresentados, para cada um deles será detalhado seus objetivos e ilustrações dos resultados obtidos.

5.3.1 PROTÓTIPO 1

A dificuldade inicial foi a comunicação entre o OpenCV e o OpenGL. De fato, a solução ideal seria utilizar o próprio OpenCV para criar a interface com o usuário (janelas, botões, etc.) e desenhos gerados pelo OpenGL serem exibidas na própria interface do OpenCV não necessitando assim da GLUT por exemplo, mas não foi encontrada nenhuma referência sobre essa integração na documentação do OpenCV. Portanto, tornou-se obrigatório o uso da GLUT. O primeiro protótipo teve por base a utilização do OpenGL e GLUT para exibir as imagens capturas da *webcam* pelo OpenCV, ou seja, este protótipo se assemelha à uma filmadora.

A *webcam* está direcionada para uma parede na qual o usuário está emitindo a luz do *laser* sobre esta. O resultado obtido por este protótipo foi de grande valia para a implementação do filtro de luz do *laser*, pois aos nossos olhos a luz é vermelha em sua totalidade, mas aos "olhos" da máquina, a luz é branca sendo vermelha apenas em suas extremidades. A figura 14 apresenta algumas imagens capturadas em modo seqüencial do vídeo produzido.

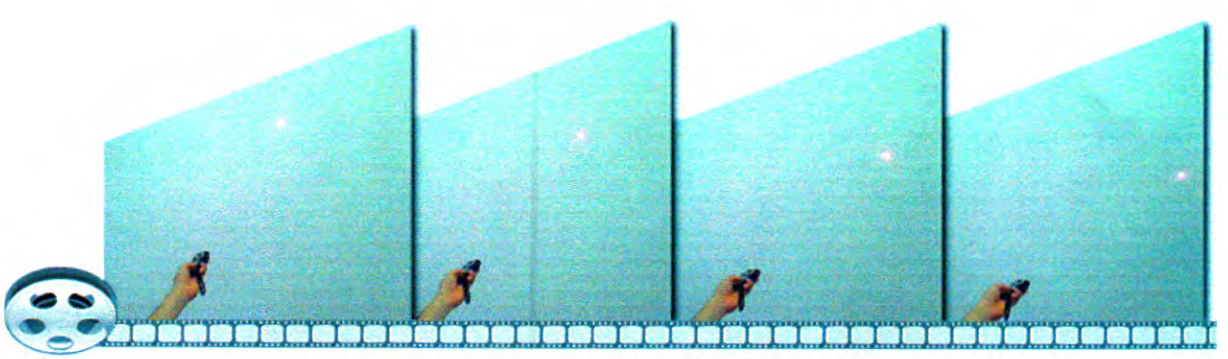

Figura 14: imagens geradas pelo protótipo 1
Fonte: primária

5.3.2 PROTÓTIPO 2

O segundo protótipo concentra-se ainda mais no processo de comunicação entre o OpenGL e OpenCV. Nesta fase foi implementada a detecção de luz sobre cada uma das imagens capturas pela *webcam*. Sendo todos os protótipos desenvolvidos de forma incremental, no momento inicial desta fase a *webcam* continua sendo semelhante a uma filmadora. Na implementação da detecção de luz do *laser* os *pixels* procurados não eram vermelhos, mas de cor branca, por motivo apresentado no protótipo 1. Sendo a imagem capturada em modelo RGB, a busca ficou entre os *pixels* que tivessem de 90% a 100% de brancura, ou seja, cada componente do modelo RGB tinham que possuir um valor aproximado maior que 230. Após a classificação dos *pixels* que atendessem a esse critério, a posição dos mesmos deveriam ser passadas para o OpenGL no intuito de ser desenhado pontos de cor verde sobre as regiões que segundo o algoritmo, eram caracterizados como pontos de luz. A figura 15 demonstra um exemplo de uso.

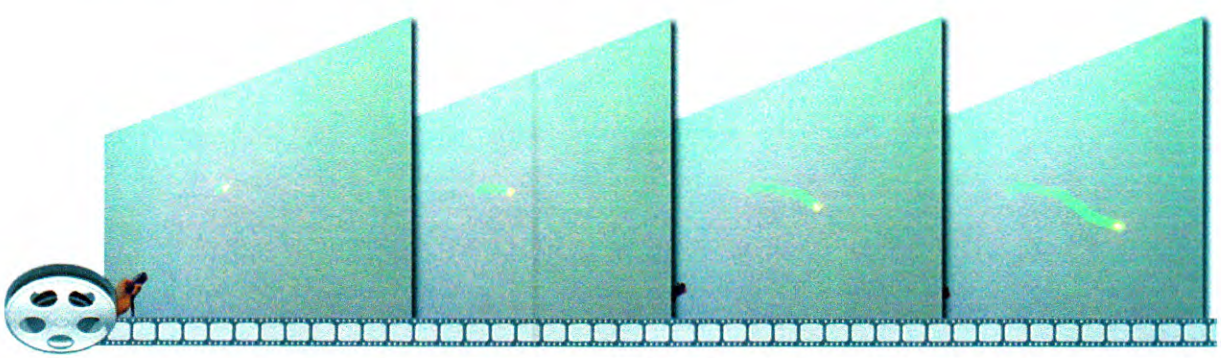

Figura 15: Imagens geradas pelo protótipo 2
Fonte: primária

5.3.3 PROTÓTIPO 3

No protótipo 3 foi implementado um sistema de partículas em OpenGL para substituir os pontos verdes criado no protótipo 2 por partículas que se movimentam em torno de uma posição pré-definida passada pelo OpenCV, obtida anteriormente no protótipo 2 pela detecção da luz emitida pelo *laser*. Pelo fato dessas partículas possuírem movimento próprio, o *laser* foi denominado de "pincel mágico" na interação com o usuário. A figura 16 demonstra o efeito criado pelas partículas implementadas neste protótipo.

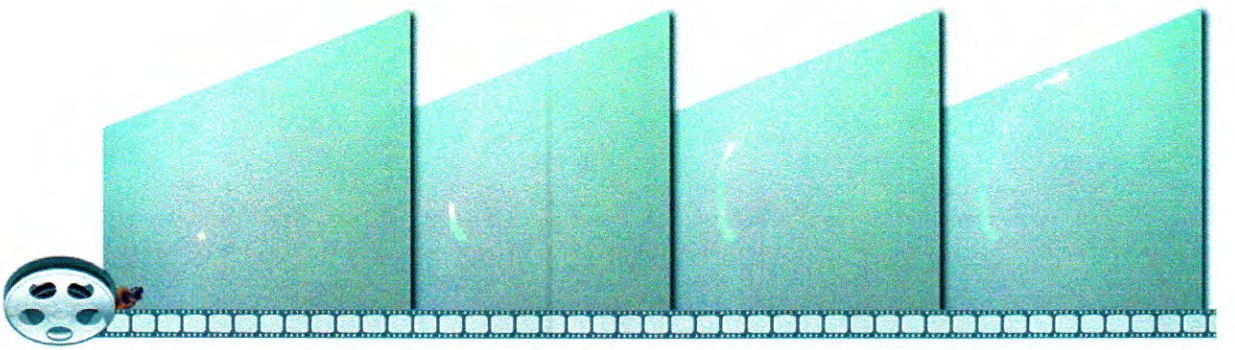

Figura 16: imagens geradas pelo protótipo 3
Fonte: primária

5.3.4 PROTÓTIPO 4

O protótipo de número 4 assim como o 3 foi implementado usando OpenGL, ou seja, está ligada à parte gráfica da aplicação. O objetivo deste protótipo é a criação de botões para que o usuário possa acioná-los fazendo uso somente do *laser*. O botão é desenhado numa camada acima da imagem extraída pela *webcam* para que o usuário possa ver o botão sempre em primeiro plano e a filmagem em segundo plano ou no plano de fundo. Internamente, cada botão possui uma variável que incrementa seu valor todas as vezes que a luz está sobre este e decrementa caso contrário. A variável sempre está entre 0 e 30 então, posicionando a luz do *laser* sobre o botão a variável é incrementa continuamente e ao atingir o valor máximo, no caso 30, o botão é acionado e seu valor é setado para 0 (zero). Seu princípio de funcionamento é simples, mas serve para demonstrar que é perfeitamente possível utilizar componentes gráficos de janela também em sistemas baseados em visão. A figura 17 demonstra o botão limpar sendo acionado pela luz do *laser* e o desenho sendo apagado.

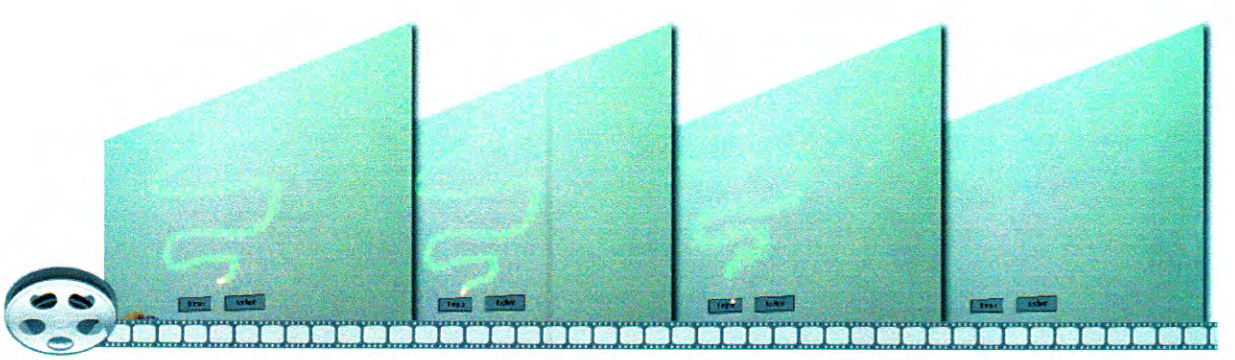

Figura 17: imagens geradas pelo protótipo 4
Fonte: primária

5.3.5 PROTÓTIPO 5

Finalmente, o quinto e último protótipo. Até o momento, o usuário está direcionando o *laser* para a parede e todo o resultado do processamento é exibido no computador. Isto não é ideal para o usuário uma vez que ele não tem precisão quando está desenhando e muito menos é agradável, pois o usuário tem que observar para onde ele está apontando o *laser* e ao mesmo tempo olhar o resultado na tela do computador.

A principal idéia desta fase é fazer com que o usuário tenha a parede como um quadro de desenho e para isto fez-se necessário a utilização de um projetor, para então exibir o resultado obtido pelo processamento na mesma área capturada pela *webcam*. Entretanto, outro problema foi criado já que o resultado exibido pelo *datashow* também está servindo como um meio de entrada de dados. Sendo assim, o resultado foi uma imagem recursiva à medida que o tempo passava. Para solucionar o problema, a camada da filmagem exibida pelo OpenGL foi substituída por um fundo de cor azulado, desta forma o protótipo não mais se comporta como uma filmadora.

Um ponto chave que deve ser observado é o posicionamento da *webcam*. Antes de utilizar o sistema de desenho é importante utilizar o protótipo 1 desenvolvido neste trabalho (ou outro visualizador/filmadora) para que seja possível alinhar o espaço de captura da *webcam* com o espaço utilizado pelo projetor. Algumas imagens do sistema implementado em sua fase final podem ser visualizadas na figura 18.

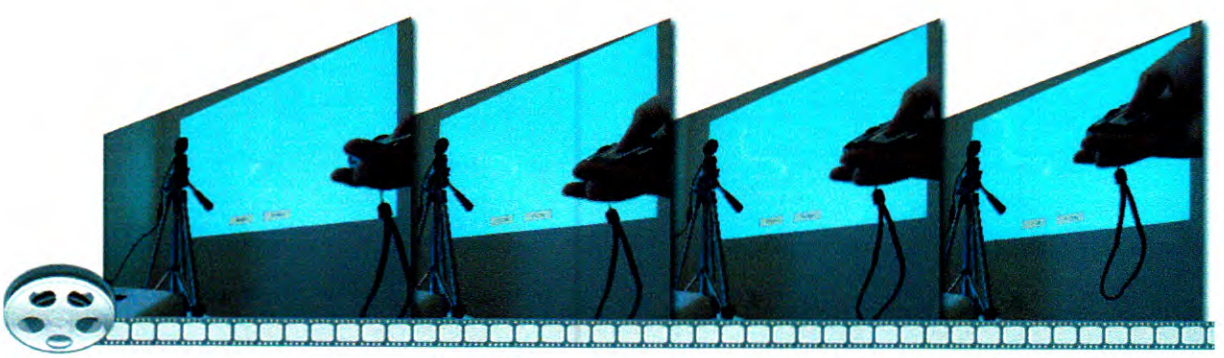

Figura 18: imagens geradas pelo protótipo 5
Fonte: primária

6 CONCLUSÃO

A finalidade deste trabalho foi demonstrar na prática como a visão computacional pode ser utilizada para criação de novas interfaces baseadas em visão, tornando essa interação homem-máquina mais natural e intuitiva. O objetivo do trabalho foi definitivamente alcançado, o sistema DesenhArte é a prova real desta afirmação.

Existem inúmeras aplicações que podem ser desenvolvida baseada na visão computacional. Fazendo uso das mesmas ferramentas aqui apresentadas é possível desenvolver um aplicativo de apresentação de slide que ao "ver" o usuário se movimentando ou fazendo algum gesto previamente definido, o programa é capaz de "entender" e pular para o próximo slide. Outro exemplo seria implementar um sistema de visão para reconhecer alguns gestos da linguagem de sinais utilizados pelos deficientes físicos.

De fato, hoje, a implementação de sistemas baseados em visão é possível graças aos avanços tanto em *hardware* quanto em *software*. Contudo, a visão computacional ainda não é tão rápida e eficiente como a visão humana devido usar tecnologias ainda tidas como em desenvolvimento ou em recente desenvolvimento, porém nas últimas décadas essa área tem crescido bastante. Espera-se que as interfaces homem-máquina realmente se tornem algo natural em nosso cotidiano e que em trabalhos futuros possam surgir novas idéias de como utilizar a visão computacional na interação homem-máquina.

REFERÊNCIAS BIBLIOGRÁFICAS

AGHAJAN, Hamid; DELGADO, Ramón; AUGUSTO, Juan. **Human-Centric Interfaces for Ambient Intelligence**. Elsevier, 2010.

BRADSKI, Gary; KAEHLER, Adrian. **Learning OpenCV**. O'Reilly, 2008.

BRUNELLI, Roberto; **Template Matching Techniques in Computer Vision.** Wiley, 2009.

CERUZZI, PAUL E. **A History of Modern Computing**. 2ª ed. MIT Press, 2003.

CHEN, C. H; WANG, P. S. P. **Handbook of Pattern Recognition and Computer Vision**. 3ª ed. World Scientific, 2005.

CHEN, J.; WEGMAN, E. **Foundations of 3D Graphics Programming**. Springer, 2006.

COHEN, Marcelo; MANSSOUR, Isabel Harb. **OpenGL: Uma Abordagem Prática e Objetiva**. Novatec, 2006.

DJERABA, Chaabane; LABLACK, Adel; BENABBAS, Yassine. **Multi-Modal User Interactions in Controlled Environments**. Springer, 2010.

GECKS, Harriet. **Optimizing C++**. Global Media, 2009.

JÄHNE, Bernd; HAUBECKER, Horst. **Computer Vision and Applications: A Guide for Students and Practitioners**. Academic Press, 2000.

JAIN, R.; KASTURI, R.; SHUNCK, B. G. **Machine Vision**. McGraw, 1995.

JUNEJA, B. L; SETH, Anita. **Programming with C++**. New Age, 2009.

KIRKLAND, Kyle. **Computer Sciense: Notable Research and Discoveries.** Facts on File, 2010.

MONTABONE, Sebastian. **Beginning Digital Image Processing: Using Free Tools for Photographers**. Apress, 2010.

MEDIONI, Gérard; KANG, Sing Bing. **Emerging Topics in Computer Vision**. Prentice Hall, 2004.

TAN, Desney S; NIJHOLT, Anton. **Humam-Computer Interaction Series: Brain-Computer Interfaces.** Springer, 2010.

REDONDO, Miguel; BRAVO, Crescencio; ORTEGA, Manuel. **Engineering the User Interface: From Research to Practice**. Springer, 2009.

REYNOLDS, Tom; BLYTHE, David. **Advanced Graphics Programming Using OpenGL**. Elsevier, 2005.

ROGERS, Yvonne; SHARP, Helen; PREECE, Jenny. **Interaction Design: beyond Human-Computer Interaction.** Wiley, 2002.

SHREINER, Dave. **OpenGL Programming Guide.** 7ª ed. Person Education, 2009.

SOLTER, Nicholas A; KLEPER, Scott J. **Professional** C++. Wiley, 2005.

VELHO, L.; CARVALHO, P. C. P.; FIGUEIREDO, L. H.; GOMES, J. **Mathematical Optimization in Computer Graphics and Vision**. Elsevier, 2008.

WAZLAWICK, Raul Sidnei. **Análise e Projeto de Sistemas de Informação Orientados a Objetos**. Campus, 2004.

WITT, Hendrik. **User Interfaces for Wearble Computers: Development and Evalnation.** MRC, 2008.

WRIGHT, Richard; LIPCHAK, Benjamin. **OpenGL SuperBIble**. 3ª ed. Sams, 2004.

BRITO, Jailson A. **Novo Aeon: Um Ambiente de Programação para Visão Computacional.** Universidade Federal da Bahia – UFBA. Disponível em <https://disciplinas.dcc.ufba.br/pub/MATA67/TrabalhosSemestre20091/monografia_jailson_final.pdf> acessado em 30 de julho de 2010.

PLAYSTATION. **Get Moving with PlayStation Move.** Disponível em <http://us.playstation.com/ps3/playstation-move/index.htm> acessado em 03 de agosto de 2010.

SCHRAMM, Rodrigo. **Sistema de Visão Computacional para Auxílio na Aprendizagem da Marcação de Compassos em Música**. Universidade do Vale do Rio dos Sinos, 2006. Disponível em <https://www.unisinos.br/inf/files/melhores_tccs/2006_2/tcc_rodrigoschramm.pdf> acessado em 21 de julho de 2010.

TRUYENQUE, Michel Alain Quintana. **Uma Aplicação de Visão Computacional que Utiliza Gestos da Mão para Interagir com o Computador**. Pontifícia Universidade Católica do Rio de Janeiro, 2005. Disponível em <http://www.tecgraf.puc-rio.br/publications/diss_2005_alain_aplicacao_visao_computacional.pdf> acessado em 29 de julho de 2010.

APÊNDICES

APÊNDICE A: CÓDIGO FONTE

Para a compilação do código fonte deve ser feita algumas lincagens com bibliotecas externas, no caso:

```
g++ main.cpp -lglut -lcv -lcxcore -lhighgui -o main
```

```
//*************************************************************************
//** Sistema: DesenhArte ***************************************************
//** Autor:   Josueh Machado - 2010 ****************************************
//*************************************************************************
//** Arquivo: main.cpp *****************************************************
//*************************************************************************

#include "lib/application.h"

Application app;

void draw() {
    app.draw();
}
void idle() {
    app.idle();
}
void keyboard(unsigned char key, int x, int y) {
    app.keyboard(key);
}

void btnClearClick() {
    app.clear();
}
void btnCloseClick() {
    app.close();
}

int main(int argc, char **argv) {
    app.init("DesenhArte - Josueh Machado", btnClearClick, btnCloseClick);
    app.run(argc, argv, draw, keyboard, idle);
    return 0;
}
```

```
//*************************************************************************
//** Arquivo: application.h ************************************************
//*************************************************************************

#ifndef _APPLICATION_H
#define _APPLICATION_H

#include <stdio.h>
#include <opencv/cv.h>
#include <opencv/highgui.h>
#include <GL/gl.h>
#include <GL/glut.h>
#include "utils.h"
```

```
#include "opengl.h"
#include "opencv.h"
#include "button.h"
#include "paint.h"
#include "particle.h"

class Application {
private :
    OpenGL opengl;
    OpenCV opencv;
    Paint  paint;
    Button btn[2];
    char   title[50];

public :

    void draw() {
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glColor4f(1,1,1,1);
        /*** descomente caso queira usar a webcam como filmadora
        glEnable(GL_TEXTURE_2D);
        glBegin(GL_QUADS);
            glTexCoord2f(0.0,1.0); glVertex2f(    0,     0);
            glTexCoord2f(1.0,1.0); glVertex2f(WIDTH,     0);
            glTexCoord2f(1.0,0.0); glVertex2f(WIDTH,HEIGHT);
            glTexCoord2f(0.0,0.0); glVertex2f(    0,HEIGHT);
        glEnd();
        glDisable(GL_TEXTURE_2D);
        ***/
        paint.draw();
        btn[0].draw();
        btn[1].draw();
        glutSwapBuffers();
    }
    void idle() {
        opencv.nextFrame();
        opengl.load(opencv.currentFrame());
        opencv.filterLaser(&paint, btn);
        glutPostRedisplay();
    }
    void keyboard(const char key) {
        if (key == 27) {
            close();
        } else if (key ==  8) {
            paint.clear();
        } else if (key == 32) {
            opengl.swapscreen();
        }
        glutPostRedisplay();
    }
    void clear() {
        paint.clear();
    }
    void close() {
        opencv.close();
        exit(0);
    }
    void init(const char *title, void (*btnClearClick)(void), void
(*btnCloseClick)(void)) {
        randomize();
        strcpy(this->title, title);
```

```
        btn[0].init("limpar", 220, 30, btnClearClick);
        btn[1].init("fechar", 330, 30, btnCloseClick);
    }
    void run(int argc, char **argv,
             void (*draw)(void),
             void (*keyboard)(unsigned char,int,int),
             void (*idle)(void)) {
        opencv.init();
        opengl.init(argc, argv, title, draw, keyboard, idle);
    }
};
#endif


//*************************************************************************
//** Arquivo: button.h ****************************************************
//*************************************************************************

#ifndef _BUTTON_H
#define _BUTTON_H

#define BUTTON_HEIGHT 30
#define BUTTON_WIDTH  80

#define BUTTON_MAX_POWER 30

class Button {
private :
    int  x, y;
    int  power;
    void (*func)(void);
    char label[20];

    bool has(int x, int y) {
        bool r = true;
        r &= x > this->x;
        r &= x < this->x + BUTTON_WIDTH;
        r &= y > this->y;
        r &= y < this->y + BUTTON_HEIGHT;
        return r;
    }

public :

    void init(const char *label, int x, int y, void (*func)(void)) {
        strcpy(this->label, label);
        this->func = func;
        this->x = x;
        this->y = y;
        power   = 0;
    }
    void draw() {
        int h = BUTTON_HEIGHT;
        int w = BUTTON_WIDTH;
        glBegin(GL_QUADS);
            glColor4f(0.3,0.3,0.3,1);
            glVertex3f(x+0, y+0, 0.2);
            glVertex3f(x+w, y+0, 0.2);
            glVertex3f(x+w, y+h, 0.2);
            glVertex3f(x+0, y+h, 0.2);
        glEnd();
```

```
        glBegin(GL_QUADS);
            glColor4f(0.5,0.5,0.5,1);
            glVertex3f(x+0+2, y+0+2, 0.21);
            glVertex3f(x+w-2, y+0+2, 0.21);
            glVertex3f(x+w-2, y+h-2, 0.21);
            glVertex3f(x+0+2, y+h-2, 0.21);
        glEnd();
        glColor4f(0.0,0.0,0.0,1);
        glRasterPos3f(x + 24, y + 11, 0.22);
        for (int i = 0; label[i]; i++)
            glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, label[i]);
        if (power > 0)
            power--;
    }
    void verify(int x, int y) {
        if (!has(x,y))
            return;
        power++;
        if (power == BUTTON_MAX_POWER) {
            func();
            power = 0;
        }
    }
};
#endif


//*************************************************************************
//** Arquivo: opencv.h ****************************************************
//*************************************************************************

#ifndef _OPENCV_H
#define _OPENCV_H

#include "paint.h"
#include "button.h"

class OpenCV {
private :
    CvCapture *webcam;
    IplImage  *frame;
    int       cam_id;

public :

    void init(int cam_id = 0) {
        this->cam_id = cam_id;
        webcam = cvCaptureFromCAM(cam_id);
        if (!webcam) {
            printf("ERRO! Erro ao inicializar webcam %d!\n", cam_id);
            exit(-1);
        }
    }
    void close() {
        cvReleaseCapture(&webcam);
    }
    void nextFrame() {
        if (frame) {
            cvReleaseImageHeader(&frame);
            cvReleaseImage(&frame);
        }
```

```
        frame = cvQueryFrame(webcam);
    }

    IplImage* currentFrame() {
        return frame;
    }
    void filterLaser(Paint *paint, Button *btn) {
        IplImage *image = frame;
        int qtd = 0;
        int r, g, b;
        CvScalar pixel;
        for (int x = 0; x < 640; x++) {
            for (int y = 0; y < 480; y++) {
                pixel = cvGetAt(image,y,x);
                b = pixel.val[0];
                g = pixel.val[1];
                r = pixel.val[2];
                if (r > 190 && g > 210 && b > 230) {
                    if (qtd++ > 1000)
                        return;
                    paint->add(x,480-y);
                    btn[0].verify(x,480-y);
                    btn[1].verify(x,480-y);
                }
            }
        }
    }
);
#endif


//*************************************************************************
//** Arquivo: opengl.h ****************************************************
//*************************************************************************

#ifndef _OPENGL_H
#define _OPENGL_H

#define HEIGHT 480
#define WIDTH  640

class OpenGL {
private :
    GLuint texture;
    bool   fullscreen;

public :

    void init(int argc, char **argv,
              const char *title,
              void (*draw)(void),
              void (*keyboard)(unsigned char,int,int),
              void (*idle)(void)) {
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
        glutInitWindowSize(400, 300);
        glutCreateWindow(title);

        glClearColor(0.41, 0.64, 0.71, 1.0);
        glOrtho(0, WIDTH, 0, HEIGHT, -1, 1);
        glEnable(GL_TEXTURE_2D);
```

```
        glEnable(GL_DEPTH_TEST);
        glDepthFunc(GL_LEQUAL);
        glFrontFace(GL_CCW);
        swapscreen();

        glutSetCursor(GLUT_CURSOR_NONE);
        glutDisplayFunc(draw);
        glutKeyboardFunc(keyboard);
        glutIdleFunc(idle);
        glutMainLoop();
    }
    void swapscreen() {
        fullscreen = !fullscreen;
        if (!fullscreen) {
            glutReshapeWindow(600, 400);
            glutInitWindowPosition(250, 50);
        } else {
            glutFullScreen();
        }
    }
    void load(IplImage *image) {
        if (texture != 0) {
            glDeleteTextures(1, &texture);
        }
        glGenTextures(1, &texture);
        glBindTexture(GL_TEXTURE_2D, texture);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
        glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB,
                     image->width, image->height, 0,
                     GL_BGR, GL_UNSIGNED_BYTE, image->imageData);
    }
};
#endif


//*************************************************************************
//** Arquivo: paint.h *****************************************************
//*************************************************************************

#ifndef _PAINT_H
#define _PAINT_H

#include "particle.h"

#define MAX_PARTICLE 25000

class Paint {
private:
    Particle p[MAX_PARTICLE];
    int      last;
    bool     hidding;

public:

    bool contains(int x, int y) {
        for (int i = 0; i < last; i++)
            if (p[i].equals(x,y)) return true;
        return false;
    }
```

```
    void add(int x, int y) {
        if (hidding) {
            int i = count();
            if (i == 0) {
                last = 0;
                hidding = false;
            }
            return;
        }
        if (last == MAX_PARTICLE || contains(x,y))
            return;
        p[last++].init(x,y);
    }
    void draw() {
        glDisable(GL_DEPTH_TEST);
        glBlendFunc(GL_SRC_ALPHA, GL_ONE);
        for (int i = 0; i < last; i++)
            p[i].draw();
        glEnable(GL_BLEND);
        glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    }
    int count() {
        int c = 0;
        for (int i = 0; i < last; i++)
            if (p[i].visible()) c++;
        return c;
    }
    void clear() {
        hidding = true;
        for (int i = 0; i < last; i++)
            p[i].clear();
    }
};
#endif


//*************************************************************************
//** Arquivo: particle.h ***************************************************
//*************************************************************************

#ifndef _PARTICLE_H
#define _PARTICLE_H

class Particle {
private:
    int   initx, inity;
    float x, y;
    float vx, vy;
    float r, g, b;
    float life;
    float fade;
    int   sleep;
    bool  hidding;

public:

    void init(int x, int y) {
        this->x = x;
        this->y = y;
        initx = x;
        inity = y;
```

```
    int angle = random(360);
    vx    = cos(angle) * 2;
    vy    = sin(angle) * 2;
    r     = randomBetween( 0, 10) / 100.0;
    g     = randomBetween(30, 40) / 100.0;
    b     = randomBetween( 0, 10) / 100.0;
    life  = randomBetween( 20, 100) / 100.0;
    fade  = randomBetween( 50,  90) / 100.0;
    sleep = 0;
    hidding = false;
}
void draw() {
    if (sleep) {
        sleep--;
        return;
    }
    if (!visible()) {
        if (hidding)
            return;
        init(initx, inity);
        return;
    }
    float x = this->x + vx;
    float y = this->y + vy;
    glBegin(GL_TRIANGLE_FAN);
        glColor4f(r, g, b, life);
        glVertex3f(x, y, 0.1);
        glColor4f(0,0,0,0);
        glVertex3f(x+5, y+0, 0.1);
        glVertex3f(x+3.5, y+3.5, 0.1);
        glVertex3f(x+0, y+5, 0.1);
        glVertex3f(x-3.5, y+3.5, 0.1);
        glVertex3f(x-5, y+0, 0.1);
        glVertex3f(x-3.5, y-3.5, 0.1);
        glVertex3f(x+0, y-5, 0.1);
        glVertex3f(x+3.5, y-3.5, 0.1);
        glVertex3f(x+5, y+0, 0.1);
    glEnd();
    move();
}
void move() {
    vx *= fade;
    vy *= fade;
    x  += vx;
    y  += vy;
    life *= fade;
    if (hidding)
        life *= fade;
}
bool visible() {
    return life > 0.05;
}
bool equals(int x, int y) {
    return this->initx == x && this->inity == y;
}
void clear() {
    hidding = true;
    fade = 0.9 + randomBetween(0, 70) / 1000.0;
    vx = randomBetween(-100,100) / randomBetween(5,20);
    vy = randomBetween(10,30);
    if (sleep) {
```

```
            sleep = 0;
            life  = 0.0;
        }
    }
};
#endif


//*************************************************************************
//** Arquivo: utils.h *****************************************************
//*************************************************************************

#ifndef _UTILS_H
#define _UTILS_H

#include <GL/glut.h>
#include <time.h>
#include <math.h>

#ifndef GL_BGR
#define GL_BGR 0x80E0
#endif

#define randomize()   (srand(time(NULL)))
#define random(n)     (rand()%(n))

#define randomBetween(n,m) (random(m-n+1)+n)

#define cos(x)    (cos((M_PI*x)/180.0))
#define sin(x)    (sin((M_PI*x)/180.0))

#endif
```