MARCOS FELIPE ALMEIDA DE SOUZA LEAL

FELIPE CUNHA DE ALBUQUERQUE

MODELAGEM 3D COM AUXÍLIO APENAS DE UMA CÂMERA

UNIVERSIDADE FEDERAL FLUMINENSE

CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

NITERÓI

2013

MARCOS FELIPE ALMEIDA DE SOUZA LEAL

FELIPE CUNHA DE ALBUQUERQUE

MODELAGEM 3D COM AUXÍLIO APENAS DE UMA CÂMERA

MONOGRAFIA: CIÊNCIA DA COMPUTAÇÃO

Monografia apresentada ao Departamento de Ciência da Computação da Universidade Federal Fluminense, como requisito parcial para obtenção de grau de bacharel em Ciência da Computação.

Orientado pela Profª. Helena Cristina da Gama Leitão

NITERÓI

2013

MARCOS FELIPE ALMEIDA DE SOUZA LEAL

FELIPE CUNHA DE ALBUQUERQUE

MODELAGEM 3D COM AUXÍLIO APENAS DE UMA CÂMERA

MONOGRAFIA: CIÊNCIA DA COMPUTAÇÃO

Monografia apresentada ao Departamento de Ciência da Computação da Universidade Federal Fluminense, como requisito parcial para obtenção de grau de bacharel em Ciência da Computação.

BANCA EXAMINADORA

_____

PROFª. DRª. HELENA CRISTINA DA GAMA LEITÃO
Universidade Federal Fluminense – UFF

_____

PROFª. DRª. ISABEL LEITE CAFEZEIRO
Universidade Federal Fluminense – UFF

_____

PROFª. DRª. ROSÂNGELA LOPES LIMA
Universidade Federal Fluminense – UFF

NITERÓI

2013

As nossas famílias:

Os maiores incentivadores para a nossa formação acadêmica.

E as pessoas que trabalham para a democratização da educação.

Agradecimentos

À Universidade Federal Fluminense onde encontramos um ambiente acolhedor.

A todos os professores que compartilharam conhecimento conosco e principalmente a nossa orientadora por nos dar grande auxílio na criação do projeto.

Aos amigos e familiares que estiveram juntos conosco desde o início da faculdade, nas derrotas e nas vitórias, sempre nos auxiliando e nos apoiando.

A todas as pessoas que nos ajudaram testando a ferramenta, dando sugestões, fazendo críticas e elogios.

Resumo

Câmeras de vídeo são um recurso amplamente disponível, visto que estão em praticamente todos os celulares. Escolhemos fazer uma ferramenta que utiliza este recurso em prol da modelagem 3D. Durante o estudo descobrimos que esta modelagem tem um alto custo computacional e focamos neste trabalho em torna-la mais acessível. Construímos em etapas os processos necessários para atingir nosso objetivo, modelagem de um objeto utilizando uma câmera , contudo, os passos finais não foram alcançados devido a complexidade do processo de modelagem. Este trabalho monográfico foi idealizado a fim de potencializar o processo de modelagem, reduzindo o tempo e a quantidade de equipamento utilizado neste processo. O resultado esperado é a criação de uma ferramenta de baixo custo que possa ser utilizada com facilidade pelos profissionais da área gerando um produto de qualidade. Neste trabalho também são descritas todas as etapas do projeto, desde pesquisas de métodos e tecnologias, passando pelo seu desenvolvimento e por fim a disponibilização dos resultados obtidos.

Palavras-chaves: modelagem, "câmera de vídeo".

Abstract

Video cameras are a feature widely available, seen there is one on every cell phone. We opted to make a tool that uses this feature in favor of 3D modeling. During the study we found that modeling has a high computational cost and this work focus in making it more accessible. We built several steps of the process necessary to achieve the objective of this study, however, the final steps were incomplete due to the complexity of the modeling process. This monograph is designed to enhance the modeling process, reducing the time and amount of equipment it uses. The expected outcome is the creation of a low-cost tool that can be easily used by field professionals creating a quality product. This paper also describes all the stages of the project, from research of methods and technologies, through its development and finally the release of the results.

Key words: modeling, video camera.

Sumário

1 Introdução

Nos últimos anos muito foi investido no desenvolvimento de novas formas de modelagem. Alguns anos atrás, a principal aplicação de modelagem 3D era navegação para robôs e inspeções. Atualmente, no entanto, a ênfase mudou pois há mais demanda para modelos 3D em computação gráfica, realidade virtual e comunicação. Isto resulta em uma mudança nos pré-requisitos da modelagem. A qualidade visual se torna o principal fator dos modelos. Existe uma importante demanda por métodos simples e flexíveis. Por isso qualquer tipo de configuração prévia deve estar ausente ou reduzida ao mínimo no processo. Várias novas aplicações também necessitam de equipamentos de aquisição robustos e de baixo de custo, o que estimula o uso de câmeras de celulares.

Nesta monografia apresentamos uma metodologia para obtermos o modelo virtual do objeto. O usuário captura as imagens mantendo a câmera parada e girando o objeto que se deseja mapear. Não é necessária qualquer forma de configuração e nem é necessário saber quais são as especificações do vídeo.

Outros métodos para a modelagem utilizando sequência de imagens já foram propostas por outros pesquisadores. A metodologia de TOMASI e KANADE (1992) inclui um método de fatoração para extrair um modelo 3D das sequências de imagens. Outra metodologia começa com um modelo inicial aproximado e refina o objeto utilizando as imagens capturadas, DEBEVEC et al (1996).

1.1 Motivação

A modelagem computacional em três dimensões é muito complexa e te-la de uma forma rápida, acessível e de fácil manipulação seria excelente para os profissionais que utilizam tal recurso. Maquetes poderiam ser mapeadas do modelo físico para modelos digitais em tempo real, pequenos objetos não mais compartilhados em fotos mas em estruturas em três dimensões, modelos tridimensionais poderiam ser exportados para outras aplicações.

No entanto percebemos que o processamento de imagens é muito custoso computacionalmente. Então focamos o trabalho em câmeras de baixo custo, com resoluções baixas. Primeiro porque o processamento de imagens é pesado e imagens com alta qualidade iria se tornar um gargalo no processamento. Teríamos que tratá-las, convertê-las e realizar os cálculos em uma grande quantidade de informação, e talvez não fizesse tanta diferença. E em seguida porque assim nosso material de trabalho poderia ser facilmente providenciado, e poderíamos realizar com uma câmera de notebook comum, como foi realizado. Nada impediria de usar imagens de alta qualidade, porém precisaríamos de processadores melhores.

Partindo desta premissa, este projeto busca entregar como produto tecnológico um sistema que gera modelos tridimensionais em tempo real. Para atingir este objetivo foram utilizados os conhecimentos adquiridos ao longo do curso de bacharelado em Ciência da Computação da Universidade Federal Fluminense, como nas disciplinas Programação I e II, Estruturas de Dados I e II entre outras.

1.2 Objetivo do trabalho

Nosso objetivo final era gerar modelos em três dimensões para rápida disponibilidade de análises e compartilhamento. Como veremos adiante, a teoria da modelagem consiste em basicamente de duas fases, coleta dos dados e uma lenta, e computacionalmente pesada, fase de reconstrução. Utilizaremos uma metodologia ágil para fazer a construção. A medida em que as informações sobre as faces do objeto a ser mapeado são capturadas modelos parciais são construídos. Entretanto a forma como a modelagem deve ser feita é muito passível de erros, e isto é um desafio, já que envolve passos delicados e custosos computacionalmente. Se a coleta não for boa e o resultado final não agradar ao usuário final, o processo de captura e a lenta reconstrução deverão ser repetidas. Logo um aplicativo capaz de fazer isso em tempo real é algo bastante promissor.

Com esse tema em mente, buscamos desenvolver esse projeto, cujo objetivo principal é construir um modelo digital de forma rápida de pequenos objetos. Porém os objetivos alcançados foram apenas uma das fases de construção de um modelo 3D, como será detalhado mais a frente.

Também é importante citar o objetivo geral deste trabalho que é contribuir para o aprofundamento em técnicas de edição de imagens.

1.3 Estrutura do trabalho

O texto será dividido da seguinte forma: No capítulo 2 veremos alguns exemplos de trabalhos na área. Usando suas metodologias eles apresentam soluções para problemas semelhantes

ao abordado nessa monografia. No capítulo 3 iremos descrever o problema melhor, citando as principais dificuldades encontradas e como pretendemos estudá-lo para chegar a solução desejada. Quais estruturas que faremos uso, descrições dos métodos pesquisados e utilizados e seus fundamentos matemáticos. No capítulo 4 descreveremos uma breve pesquisa sobre as possíveis formas de atacar o problema. No capítulo 5 falaremos com detalhes sobre os principais algoritmos e detalhamos suas implementações. No capítulo 6 mostraremos os detalhes do experimento. Seus resultados e comparações com as diversas abordagens utilizadas, as tentativas que fizemos dos variados métodos. No capítulo 7 mostramos as conclusões que chegamos. No capítulo 8 falamos dos trabalhos relacionados e concluímos com trabalhos futuros.

2 Trabalhos relacionados

A manipulação e modelagem de imagens em 3D nos últimos anos foi bastante popularizada.

A seguir apresentamos algumas aplicações onde diversas técnicas de modelagem e dispositivos que servem como fornecedores de dados para sistemas . GILBERTO CUNHA (2003) descreve que um dos apectos mais importantes da elaboração da modelagem é a relação entre simplicidade e fidelidade. Isso é um dos fatores para definir que tipo de equipamento usar e que qualidade de imagens se quer trabalhar.

O principal trabalho que foi apresentado pela orientadora, e o que contém a metodologia que nos baseamos, é o de QI PAN (2009), que tem o nome de PROForma. Ele não faz uso da abordagem tradicional das duas fases: captura e procesamento. Ele as mescla, fazendo a modelagem enquanto o vídeo de entrada é capturado. Resumindo o fluxo do ProFORMA, o usuário manipula um objeto, que é capturado pela câmera. Enquanto o usuário rotaciona o objeto em frente a câmera um modelo parcial é construído e exibido em tempo real. É feito seu rastreamento, reconstrução e modelagem e então são exibidos o modelo parcial e a captura da câmera do objeto. O fluxo se repete até que o usuário tenha rotacionado todo o objeto.

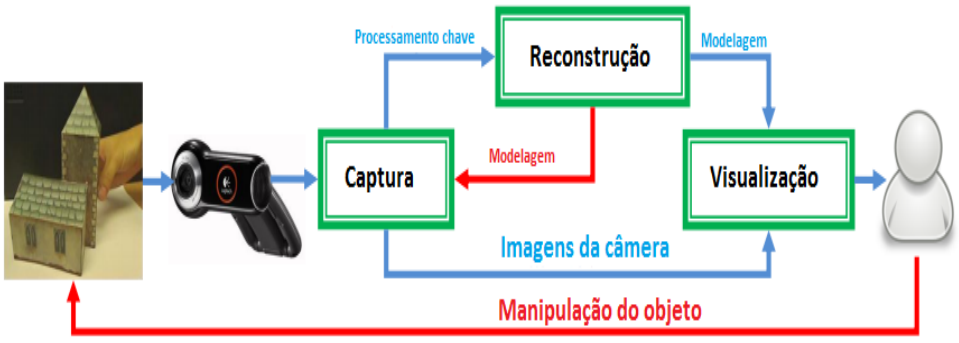

Figura 1 – Fluxo do processamento do projeto de Qi Pan

Fonte: Projeto PROForma

Tomamos a mesma iniciativa, precisamos que o usuário opere o sistema girando o objeto a ser capturado. Temos uma fase de rastreamento, onde podemos detectar o movimento do objeto. Essas informações são guardadas para processamento posterior, em que este é montado, seguido da exibição do resultado para o usuário.

Um exemplo onde equipamentos como lasers, scanners, projetores e câmeras são utilizadas para modelagem, não de objetos mas de interiores de construções, é o trabalho de AVIDEH ZAKHOR (2012).

Dotado de uma mochila com esses equipamentos citados, um controlador anda por corredores e túneis e é feita uma varredura do ambiente a sua volta. O objetivo principal de sua proposta é projetar, analisar e desenvolver arquitetura e algoritmos, bem como associar as estatística e cálculos matemáticos que contém o padrão da estrutura para um sistema portátil de modelagem de interiores em três dimensões, capaz de gerar modelos baseada em fotos de estruturas de vários andares de edifícios.

Uma das desvantagens do projeto de AVIDEH ZAKHOR, além do custo para se obter os materiais necessários para o experimento, é que caso a mostragem de imagens e dados não sejam boas o suficiente teria que se fazer uma outra varredura para um novo processamento. E como o processo que eles utilizam para captura dos dados é um operador com uma mochila pesada caminhando pelos corredores, isto torna-se um incômodo.

Outro exemplo de modelagem usando vídeos é o trabalho de LORENZO TORRESANI e AARON HERTZMANN (2004). Eles desenvolveram o projeto com a promessa de que reconstruções de modelos a partir de vídeos possam produzir resultados de alta qualidade para muitas aplicações como animações e análises.

Eles citam ainda duas das dificuldades encontradas quando se trabalha com imagens. Primeiro, os algoritmos existentes assumem que as cores das faces dos objetos permanecerão a mesma em todas as capturas que serão analisadas. O que é verdade, e acaba interferindo bastante na qualidade do projeto já que a constância de cor, como eles chamam, pode ser alterada por oclusões, mudança de iluminação, desfoque por movimento, entre outros. E segundo, mais relativo ao problema abordado por eles, é que recontruções de modelos não rígidos precisam de vários ajustes e regulagens para um bom resultado. Tais calibragens devem ser feitas manualmente, tornando em uma tarefa difícil e ocasionando muitas imprecisões. Caso opte-se, essa calibragem pode ser aprendida por uma outra entrada de dados, mas esta geralmente não está disponível ou é inapropriada para a ocasião.

Existem também programas no mercado que auxiliam na modelagem de objetos. O Autocad (2013), um dos mais famosos programas de desenho auxiliado por computador, é capaz de

construir e visualizar objetos e peças, modelos tridimensionais com muita qualidade. Além de ser um software extremamente caro e não trivial de usar, não é possível utilizar outros dados para gerar os modelos automaticamente.

Um programa criado recentemente e que tem o mesmo objetivo que o nosso é o Projeto Autodesk 123D Catch (2013), antigamente Autodesk Photofly. Seu funcionamento é baseado em fotos enviadas para seu servidor, onde serão processadas. Há um rascunho onde o usuário pode editar pequenas partes do projeto manualmente, e um breve vídeo do seu modelo gerado. Seus produtores incentivam a utilizar outras ferramentas da família 123D de modelagem como o 123D Make que transforma modelos 3D em um padrão de corte com instruções de montagem para fazer animações e o 123D Sculpt que facilita a esculpir e pintar formas realísticas em 3D, tendo o formato do aplicativo até para iPad.

3 O problema

Na interface do nosso sistema o usuário pode selecionar qual será a entrada de dados para o processamento, arquivos de vídeos previamente gravados no disco ou vídeos produzidos diretamente de uma câmera conectada ao computador.

As imagens estáticas são fotografias capturadas a partir de uma câmera. Os vídeos são compostos por diversas dessas imagens, que são exibidas uma após a outra em diversas frequências. Elas são, então, capturadas individualmente para análise em tempo real de execução do programa. Nessa captura, a imagem, já separada do vídeo, é chamada de "frame" e passa por alguns processos para identificação dos pontos críticos e detecção de movimento do objeto sendo rastreado.

O método de captura de dado seria girando o objeto em frente a câmera, mantendo esta estacionária, enquanto o programa captura imagens do objeto e realiza seu processamento. Enquanto é feita a captura de todas as faces do objeto, é montado o modelo onde o usuário pode ver o resultado e decidir se será enviado, armazenado, editado ou se gostaria de tirar outras amostragens. Também implementamos a opção de entrar com um vídeo gravado previamente para que fosse feita a mesma análise e modelagem. Assim permitimos uma forma rápida de repetimos o processamento para realização de testes.

4 Pesquisa para estruturar a solução

O projeto enfrentou uma série de dificuldades, como a necessidade de melhora do desempenho do algoritmo. Como descreveremos adiante durante os experimentos, inicialmente pegávamos todos os frames para realizarmos análise para a obtenção do modelo, mas o hardware necessário para esse tipo de processamento precisa ser muito potente e consequentemente muito caro.

A obtenção dos pontos relativos do objeto geraram grande problema. Para que não houvesse nenhuma interferência seria necessário utilizar um fundo de uma cor que se destaque do objeto, de preferência preto, e para manipular o objeto uma luva da mesma cor teria de ser usada, mas isto limita muito o ambiente onde o objeto pode ser modelado e não é nossa intenção utilizar esse tipo de técnica. Para obter os pontos somente do objeto que estamos analisando teríamos de isolar o fundo e recuperar somente os dados relativos ao objeto.

Como não conseguimos montar um ambiente isolado para a captura nas condições descritas anteriormente, fizemos nossos testes com uma gravação de uma estrutura de papel, figura 2, em um ambiente suficientemente iluminado, rotacionando o objeto de forma lenta e contínua.

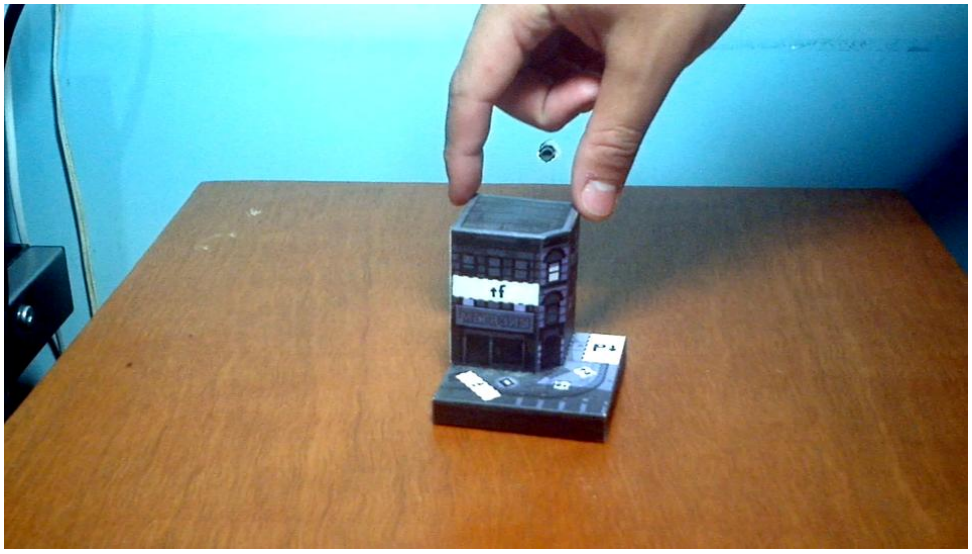

Figura 2 – Figura Frame original do vídeo com maior qualidade
Fonte Elaboração própria

5 Algoritmos e implementações

As imagens abaixo (WIKIPÉDIA, 2013) mostram um exemplo de processamento de detecção de borda. Primeiro a imagem original e em seguida processada pelo operador de Sobel.

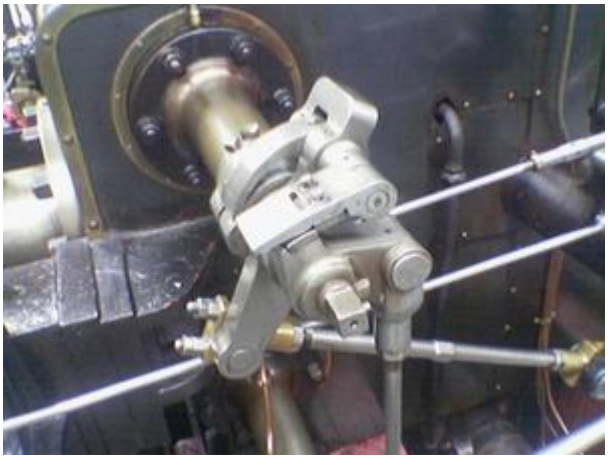

Figura 3 – Imagem sem filtros de uma válvula
Fonte: Adaptado de http://en.wikipedia.org/wiki/Sobel_operator

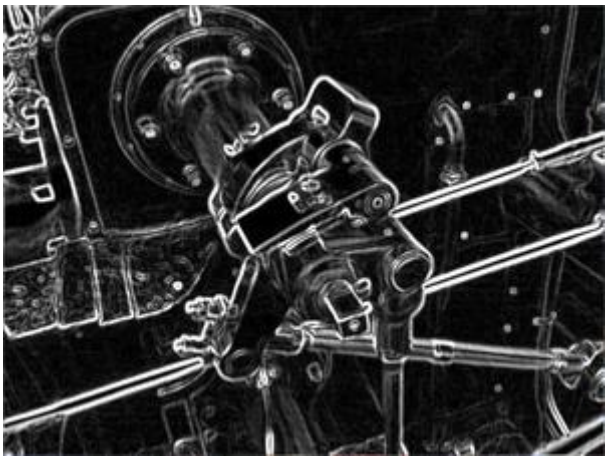

Figura 4 – Filtro de Sobel aplicado a figura 3
Fonte: Adaptado de http://en.wikipedia.org/wiki/Sobel_operator

5.1 Detecção de borda

Algoritmos de detecção de borda se referem a processos capazes de identificar descontinuidades em uma imagem. Estas são mudanças abruptas de cor, brilho, profundidade ou orientação caracterizando a borda dos objetos em uma cena. Uma representação de uma imagem em termos de borda é um forma compacta no sentido de que uma imagem com padrão de duas dimensões é representada por um conjuntos de curvas em uma dimensão (ENCYCLOPEDIA OF MATHEMATICS, 2013).

5.2 Variáveis de detecção de borda

Existem muitos algoritmos para realizar a detecção de borda e possuem algumas variavéis para se considerar como:

5.2.1 Orientação das bordas

Pode-se indicar ao operador qual é a direção mais sensivel do objeto em que se desejar identificar as bordas horizontal, vertical ou diagonal.

5.2.2 Ambientes com ruídos

Detectar bordas em ambientes muito granularizados é complicado pois apareceriam muitas bordas e a imagem ficaria borrada. Estes métodos geralmente trabalham com uma sensibilidade maior, resultando também em uma detecção menos precisa.

5.2.3 Estrutura da borda

Nem todas as bordas consistem em variações de intensidade. Efeitos como refração ou não focalizada podem definir objetos com uma mudança gradual na intensidade dos pixels. Então seu algoritmo precisa ser refinado para cada configuração.

5.3 Cálculo matemático

O filtro de Sobel, em termos técnicos, consiste num operador que calcula diferenciais finitas , dando uma aproximação do gradiente da intensidade dos pixels da imagem. Em cada ponto da imagem, o resultado da aplicação do filtro Sobel devolve o gradiente ou a norma deste vetor. O filtro Sobel calcula o gradiente da intensidade da imagem em cada ponto, dando a direção da maior variação de claro para escuro e a quantidade de variação nessa direcção. Assim, obtém-se uma noção de como varia a luminosidade em cada ponto, de forma mais suave ou abrupta.

Com isto consegue-se estimar a presença de uma transição entre uma parte clara e um escura e qual a orientação desta. Como as variações claro-escuro intensas correspondem a fronteiras bem definidas entre objetos, consegue-se fazer a detecção de contornos.

Matematicamente este operador utiliza duas matrizes 3×3 que sofrem convoluções com a imagem original para calcular aproximações das derivadas - uma para as variações horizontais e uma para as verticais. Sendo A a imagem inicial então, Gx e Gy serão duas imagens que em cada ponto contêm uma aproximação às derivadas horizontal e vertical de A.

$$G_x = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \qquad G_y = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Portanto a magnitude, **G** , e a direção, **Θ** , do gradiente são dados por:

$$\mathbf{G} = \sqrt{\mathbf{G_x}^2 + \mathbf{G_y}^2}$$

$$\mathbf{\Theta} = \arctan\left(\frac{\mathbf{G_y}}{\mathbf{G_x}}\right)$$

$$|\Delta I| = \sqrt{(G_x * I)^2 + \left(G_y * I\right)^2}$$

Como consequência da sua definição este operador pode ser implementado de forma simples: apenas oito pontos da imagem em volta de um ponto da imagem é necessario para computar o resultado correspondente, e somente a integral aritmética é necessária para computar o gradiente do vetor de aproximação. Sendo assim os filtros descritos acima podem ser separados em :

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$$

E as duas derivadas em:

$$\mathbf{G}_x = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} * \left(\begin{bmatrix} -1 & 0 & 1 \end{bmatrix} * \mathbf{A}\right) \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} * \left(\begin{bmatrix} 1 & 2 & 1 \end{bmatrix} * \mathbf{A}\right)$$

Em certas aplicações essa separação pode ser proveitosa pois implica em menos cálculos a serem computados para cada ponto da imagem [9].

5.4 Detecção de quina

Durante nossas pesquisas vimos uma ramificação da detecção de borda, que é a detecção de quina. Por definição a quina é o encontro de duas bordas ou ainda como o ponto onde duas bordas diferentes de uma mesma região se encontrariam. Esses algoritmos podem ser utilizados nos resultados dos algoritmos de detecção de borda, mas não nos aprofundamos neles pois não nos era interessante o aumento de refinamento proporcionado.

5.5 Tracking

Tracking ou rastreamento é o processo de localizar objetos que se movem durante a gravação de um vídeo. Esse processo possui ampla usabilidade, indo desde interação entre humanos e computador, segurança e vigilância, comunicação através de vídeos, realidade aumentada, controle de tráfego e edição de vídeos. Como já foi descrito, o processamento de imagem consome muitos recursos computacionais e o rastreamento não é diferente. Como queremos fazer um reconhecimento de objetos junto com o rastreamento, a complexidade aumenta com o número de variáveis envolvidadas no processo.

Objetivo do tracking é associar o objeto alvo em diferentes imagens, poder reconhecer o mesmo objeto em diferentes frames do vídeo. O movimento do objeto no vídeo pode ser uma transformação plana, quando move-se em um plano, ou giro, quando pode-se capturar suas faces.

Há uma série de complicadores na hora de realizar o tracking, por exemplo, objetos que se movam muito rápido não aparecem em muitos frames, ou quando o objeto muda sua orientação, e.g. de cabeça para baixo, etc. Nessas situações os sistemas de rastreamento geralmente implementam uma previsão de como o objeto vai estar, ou como vai mudar de acordo com seu movimento já executado.

Os algoritmos de rastreamento fazem basicamente a mesma coisa, analisam uma sequência de frames (vídeo) e retornam uma representação do movimento captado. Mas cada um dos algoritmos possuem prós e contras, e saber escolher qual utilizar baseado na aplicação é crucial. As aplicações podem ser divididas em duas categorias: localização e representação do alvo; e associação e filtragem de dados.

5.6 Descrição do fluxo de processamento

Para entrada do programa pode ser escolhido uma gravação de vídeo (experimentamos com WMV e MP4) ou um vídeo pode ser capturado no momento da transformação (varia de acordo com o software da câmera).

```
if (mode == 1)

    video = cvCaptureFromFile(filename.toLocal8Bit().data());
```

```
else

  video = cvCaptureFromCAM(CV_CAP_ANY);
```

É feita uma breve checagem do video para saber se está sem erros.

```
if (!video) {

  alert("ERRO: captura nula...\n");

  return;

}
```

São criadas as janelas em que serão exibidos os vídeos e resultados parciais do processamento.

```
cvNamedWindow("Original", CV_WINDOW_AUTOSIZE);

cvNamedWindow("Pontos", CV_WINDOW_AUTOSIZE);

cvNamedWindow("Motion", CV_WINDOW_AUTOSIZE);
```

Do vídeo é extraída a primeira imagem, frame, a ser tratada, esta serve de parâmetro para todas as imagens que serão criadas, pois serão baseadas em tamanho e formato da imagem original.

```
IplImage *frame = cvQueryFrame(video);
```

Em seguida são instanciadas as variáveis que servirão para receber os dados. Neste grupo estão inclusos a imagem de pontos-chave, a imagem que captura o movimento, a imagem cinza e a imagem com tratamento de bordas (Sobel).

```
// Imagem de pontos-chave
```

IplImage *modelo = cvCreateImage(cvGetSize(frame), IPL_DEPTH_8U, 1);

// Imagem de captura de movimento

IplImage* motion = cvCreateImage(cvGetSize(frame), IPL_DEPTH_8U, 3);

// Imagem em escala de cinza

IplImage *grey = cvCreateImage(cvGetSize(frame), IPL_DEPTH_8U, 1);

// Imagem de borda

IplImage *sobelImage = cvCreateImage(cvGetSize(frame), IPL_DEPTH_8U, 1);

Criamos um vetor para armazenar os chamados *key edge* points (pontos críticos da imagem de borda).

QVector<CvPoint2D32f> keyEdgePointMatches;

Para dar destaque aos pontos e linhas que aparecem no modelo o fundo desta imagem é colorido de branco e a imagem de captura de movimento é zerada para que não haja interferência.

// Pinta a imagem de pontos-chave de branco

cvSet(modelo,cvScalarAll(255));

// Zera a imagem de movimento

cvZero(motion);

A captura de movimento inicia no ponto em que o primeiro frame é capturado.

```
motion->origin = frame->origin;
```

Inicia a variável de contagem dos frames.

```
int frameCount = 0;
```

Inicio da repetição que verifica se ainda há frames a serem lidos no arquivo de entrada ou se o processamento foi interrompido pelo usuário, através da tecla ESC.

```
while ( ((cvWaitKey(10) & 255) != 27) && frame ) {

  …

}
```

É feito um teste para certificar que o frame foi capturado com sucesso.

```
if (!frame) {

  alert("ERRO: frame nulo...\n");

  return;

}
```

É chamado o método de atualização do movimento do objeto.

```
update_mhi(frame, motion, 30);
```

O contador de frames é incrementado.

```
frameCount++;
```

Para manter um desempenho aceitável, o processamento é feito a cada 20 frames e neste ponto é feita uma verificação do frame para executar ou não o processamento referido.

```
if ((frameCount % 20) == 0) {

  …

}
```

Dentro da condicional acima mencionada é feito o processamento principal do programa. Inicialmente é feita uma limpeza da janela de vizualização do modelo. Em seguida o frame original é copiado em escala de cinza para outra janela.

```
cvSet(modelo,cvScalarAll(255));

cvCvtColor(frame, grey, CV_RGB2GRAY);
```

Com a imagem em escala de cinza fazemos o tratamento para captura das bordas utilizando o método do Sobel.

```
sobelImage = fazSobel(grey);
```

Utilizando a imagem resultando do tratamento Sobel, é feito a busca por pontos importantes com o método SURF.

```
keyEdgePointMatches = findKeyPointsWithSURF(sobelImage);
```

Por fim é colocado na imagem modelo os pontos encontrados.

```
drawPoints(keyEdgePointMatches, &modelo, 2);
```

Ao final do condicional fazemos a exibição de todas as imagens relevantes e a captura de um novo frame.

cvShowImage("Original", frame);

cvShowImage("Pontos", modelo);

cvShowImage("Motion", motion);

frame = cvQueryFrame(video);

Ao termino do loop o vídeo é liberado da memória junto com as janelas criadas ao longo do processo.

cvReleaseCapture(&video);

cvDestroyWindow("Original");

cvDestroyWindow("Pontos");

cvDestroyWindow("Motion");

O método de atualização de movimento

Inicialmente verificamos se as imagens estão criadas e se não estiverem será alocada memória para a mesma e cria em seguida.

Nesse condicional verificamos de a imagem mhi está criada ou se seu tamanho é menor que o da imagem de entrada.

if(!mhi || mhi->width != size.width || mhi->height != size.height)
{

...

```
}
```

Verificamos de a imagem buf está criada e caso contrário é alocada memória para 4 imagens do tamanho de buf em seqüência.

```
if( buf == 0 ) {

  buf = (IplImage**)malloc(4*sizeof(buf[0]));

  memset( buf, 0, 4*sizeof(buf[0]));

}
```

Neste loop fazemos o preenchimento de buf como um array de imagens com 4 posições e em cada posição é colocada uma imagem replicada da imagem de entrada, para manter dimensões e tipos iguais, e zerada em seguida.

```
for( i = 0; i < 4; i++ ) {

  cvReleaseImage( &buf[i] );

  buf[i] = cvCreateImage( size, IPL_DEPTH_8U, 1 );

  cvZero( buf[i] );

}
```

As imagens são liberadas para que não fique nenhum lixo na memória.

```
cvReleaseImage( &mhi );

cvReleaseImage( &orient );

cvReleaseImage( &segmask );
```

cvReleaseImage(&mask);

A imagem mhi é criada a partir das dimensões da original e zerada em seguida. O mesmo é feito com as outras imagens auxiliares.

mhi = cvCreateImage(size, IPL_DEPTH_32F, 1);

cvZero(mhi); // clear MHI at the beginning

orient = cvCreateImage(size, IPL_DEPTH_32F, 1);

segmask = cvCreateImage(size, IPL_DEPTH_32F, 1);

mask = cvCreateImage(size, IPL_DEPTH_8U, 1);

Transformamos a imagem de entrada em cinza.

cvCvtColor(img, buf[last], CV_BGR2GRAY);

A variável idx2 recebe a próxima posição que ficará variando de 0 a 4 (as posições da variável buf) e está será a nova última posição.

idx2 = (last + 1) % 4;

last = idx2;

A variável silh recebe a última imagem capturada por motivo de inicialização.

silh = buf[idx2];

O próximo passo é pegar as diferenças entre os últimos frames e coloca-las em uma imagem, no caso na variável silh.

cvAbsDiff(buf[idx1], buf[idx2], silh);

Agora fazemos o threshold, armazenado em silh, e a atualização do movimento na imagem, armazenado em mhi.

cvThreshold(silh, silh, diff_threshold, 1, CV_THRESH_BINARY);

cvUpdateMotionHistory(silh, mhi, timestamp, 1);

Agora convertemos o resultado em uma imagem azul, zeramos a imagem de destino e fazemos a junção da imagem azul com a imagem de destino.

cvCvtScale(mhi, mask, 255./1, (1 - timestamp)*255./1);

cvZero(dst);

cvMerge(mask, 0, 0, 0, dst);

5.7 O método Sobel

Tivemos uma descrição completa, com seus embasamentos matemáticos na seção 5.3 deste capítulo.

Inicialmente são criadas duas imagens auxiliares. Uma para receber o resultado do processamento do método cvSobel e outra para receber a conversão deste resultado para um formato que seja melhor para exibir.

IplImage *dfx = cvCreateImage(cvGetSize(inputImage), IPL_DEPTH_16S, 1);

IplImage *dest_dfx = cvCreateImage(cvGetSize(inputImage), IPL_DEPTH_8U, 1);

É chamado o método cvSobel na imagem de entrada do método fazSobel e o resultado armazenado na variável dfx.

cvSobel(inputImage, dfx, 1, 0, 3);

Em seguida é feita a conversão para um formato que posso ser exibido e o resultado armazenado na variável dest_dfx, que será retornada no final do método.

// Converte para o tipo de imagem correta para exibição

cvConvertScaleAbs(dfx, dest_dfx, 3, 0);

return dest_dfx;

5.8 O método SURF

Surf, acrônimo para Speeded Up Robust Features, é um algoritmo que extrai pontos chaves, aqui chamados de keypoints, únicos e descrições de uma imagem. Um conjunto deles pode calculado e usado para reconhecer a imagem de que foram extraidos posteriormente. Descrito em detalhe no artigo de HERBERT BAY, ANDREAS ESS, TINNE TUYTELAARS E LUC VAN GOOL (2008), ele é um método que em certos pontos é capaz de realizar suas verificações mais rápido do que seus antecessores. A chave para isso, dizem seus autores, consiste em 3 pontos: confiança nas imagens integrais, que consistem em imagens intermediárias que são computadas a partir da imagem original e são usadas para acelerar os cálculos em qualquer área retangular, para ser realizada as convoluções; construindo algoritmos de detecção e descritores mais

poderosos baseados nos já existentes; e ao mesmo tempo simplificando estes métodos para somente o seu essencial.

Sua detecção de keypoints é executada através da básica aproximação de matrizes Hessiana. Reduzindo drasticamente o tempo computacional , pois assim tende-se a usar as imagens integrais em, o que eles chamam de caixa de convoluções. As imagens integrais são calculadas somando-se as coordenadas x e y de uma imagem, tornando o tempo de computação invariável em função do tamanho. Outro ponto forte do SURF é que não necessita de treino como alguns detectores, por exemplo os baseados no classificador de HAAR (WIKIPEDIA, 2012). E como ele também invariante à rotação é possivel detectar objetos em qualquer orientação.

Para pegarmos pontos relativos temos de considerar um alcance das buscas e o intervalo é armazenado na variável params.

// Only images with a hessian greater than 500 are considered for keypoints

CvSURFParams params = cvSURFParams(500, 1);

É criada uma variável para guardar as informações necessárias para o processamento do método cvExtractSURF.

CvMemStorage* memBlock = cvCreateMemStorage();

São criadas duas variáveis para armazenar as seqüências de pontos-chave e descritores que são o resultado da execução do método.

CvSeq* edgeKeyPoints;

CvSeq* edgeDescriptors;

Nesse ponto podemos fazer a chamada ao método cvExtractSURF passando como parâmetro a imagem de entrada do método findKeyPointsWithSURF e armazenar as informações que queremos nas variáveis referenciadas acima e passadas por parâmetro como mostrado abaixo.

cvExtractSURF(inputImage, 0, &edgeKeyPoints, &edgeDescriptors, memBlock, params);

Criamos um vetor para armazenar os pontos que queremos.

QVector<CvPoint2D32f> keyPointMatches;

Aqui percorremos todos os descritores achados na imagem e acrecentamos os pontos ao vetor keyPointMatches e retornamos esta variável ao terminar o loop.

```
for (int i = 0; i < edgeDescriptors->total; i++) {

  keyPointMatches.append(((CvSURFPoint*)
cvGetSeqElem(edgeKeyPoints, i))->pt);

}

return keyPointMatches;
```

6 Experimentos

Após analisarmos o trabalho de QI PAN fomos orientados a primeiramente iniciar a pesquisa trabalhando sobre uma imagem transformada para preto e branco, pois assim seria mais facil de realizar os processamentos futuros.

No laboratório gravamos um video de um pequeno objeto de plástico para darmos início as pesquisas. Através de um comando do Linux convertemos o video inteiro para um formato que mostra somente sombras. Segue abaixo o exemplo do resultado de um frame:

Figura 5 – Frame original do primeiro vídeo testado
Fonte: Elaboração própria

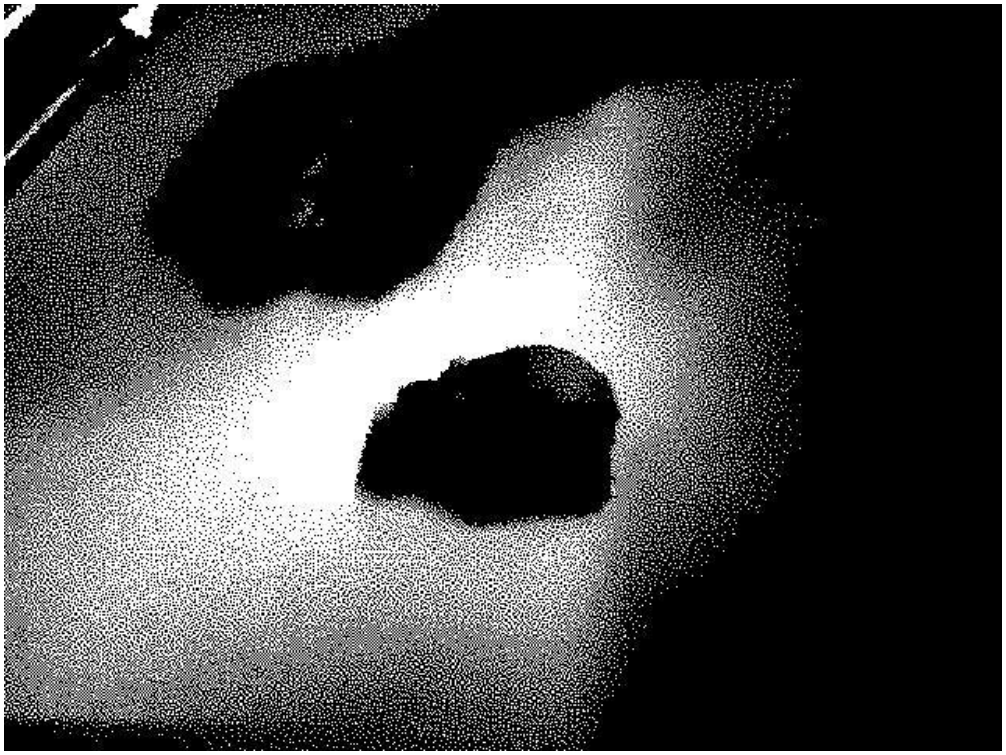

Figura 6 – Figura 5 transformada com um algoritmo de detecção limiar global.
Fonte: Elaboração própria

Podemos observar que as bordas do objeto foram detectados porém com um nível de detalhe muito baixo, pois ele se tratava de um algoritmo de limiar global, para identificar regiões. Logo nesse primeiro passo já observamos o quanto é lento trabalhar com imagens. Um vídeo de poucos minutos levou 3 vezes mais que a duração do arquivo para ser procesado.

Então iniciamos os estudos sobre algoritmos de detecção de borda. Mas antes tentamos utilizar da tecnica de segmentação.

Segmentação é o processo de particionar uma imagem digital em vários segmentos, ou conjuntos de pixels. O objetivo da segmentação é simplificar a representação de uma imagem para algo que seja mais fácil de analisar. Neste processo a imagem é dividida em várias partes que juntas formam o todo. Tipicamente usada para

localizar objetos e formas, como bordas e quinas, em imagens. O resultado da segmentação são pedaços que tenham pixels com uma certa característica em comum, como cor, intensidade ou textura.

Logo no ínicio de nossas pesquisas vimos os principais algoritmos de detecção de borda, que são os descritos anteriormente no texto. Para agilizar as pesquisas procuramos um framework que nos auxiliasse com esses algoritmos complexos, que já tivessem alguns métodos já implementados. Foi quando descobrimos o OpenCV.

Resolvemos fazer uma nova captura de vídeo para os nossos experimentos e para isso montamos um pequeno modelo de prédio e filmamos utilizando uma câmera de laptop para nos auxiliar com os testes no OpenCV.

Neste momento começamos a nos familiarizar com o framework e seu uso. Descobrimos que possuem muitos outros métodos de tratamento de imagens. Maiores detalhes podem ser encontrados em sua documentação (OPENCV, 2013).

Os principais algoritmos eram: Sobel, Canny e Frei-Chen.Tentamos utilizar para a detecção de bordas e cantos o algoritmo de FreiChen que teoricamente é melhor e vai além do algoritmo do Sobel, que faz detecção de bordas apenas, e faz detecção de cantos também. Mas assim que fizemos os primeiros testes percebemos que seus resultados tinham muito ruídos, o que atrapalhava a nossa implementação final, pois os pontos chaves recuperados eram distorcidos por este ruído.

Quando experimentamos o método de Canny tivemos algumas complicações para implementar. Apesar dos resultados obtidos no decorrer do experimento terem sidos promissores, não

tivemos muito sucesso ao mesclar com o método SURF, que escolhe os pontos chaves para realizar o rastreamento. Como vemos no exemplo a seguir os resultados da detecção foram consideravelmente melhores. Mais bordas foram detectadas e com uma definição melhor, a medida que alteramos a limitação do algoritmo o seu resultado é alterado drasticamente.

Figura 7 – Resultado do método de Canny aplicado a figura 2.
Fonte: Elaboração própria

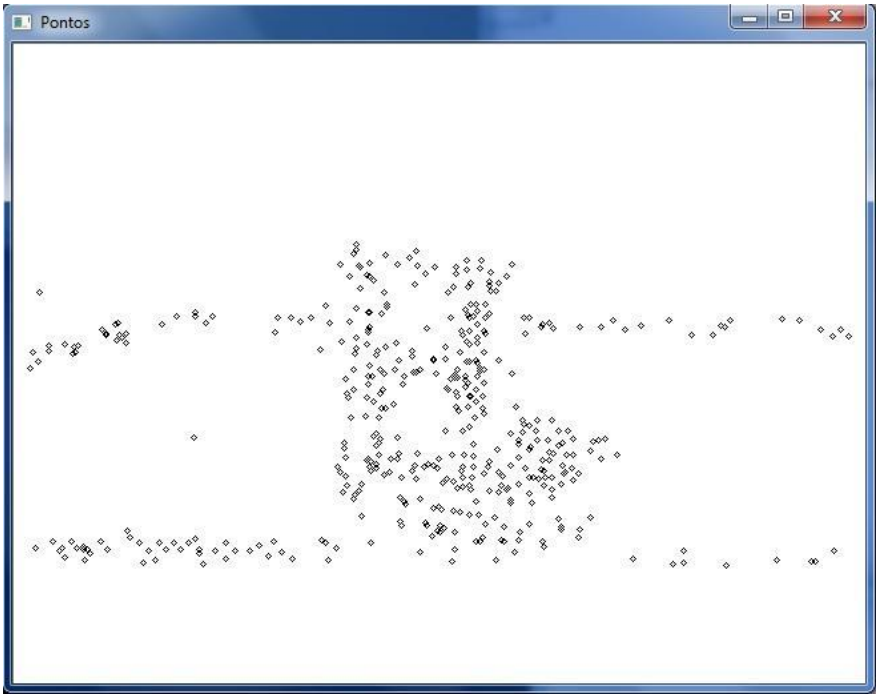

Figura 8 – Frame do nosso objeto de estudo com seus pontos chaves.
Fonte: Elaboração própria

Então experimentamos o método de Sobel. A figura 8 também é conhecida como janela Pontos. Nela é que são mostrados os pontos recuperados pelo método findKeyPointsWithSURF. Nele realizamos uma varredura para encontrar os pontos em diferentes frames, rastreando assim o movimento do objeto. As imagens utilizadas são as transformadas de Sobel, método fazSobel.

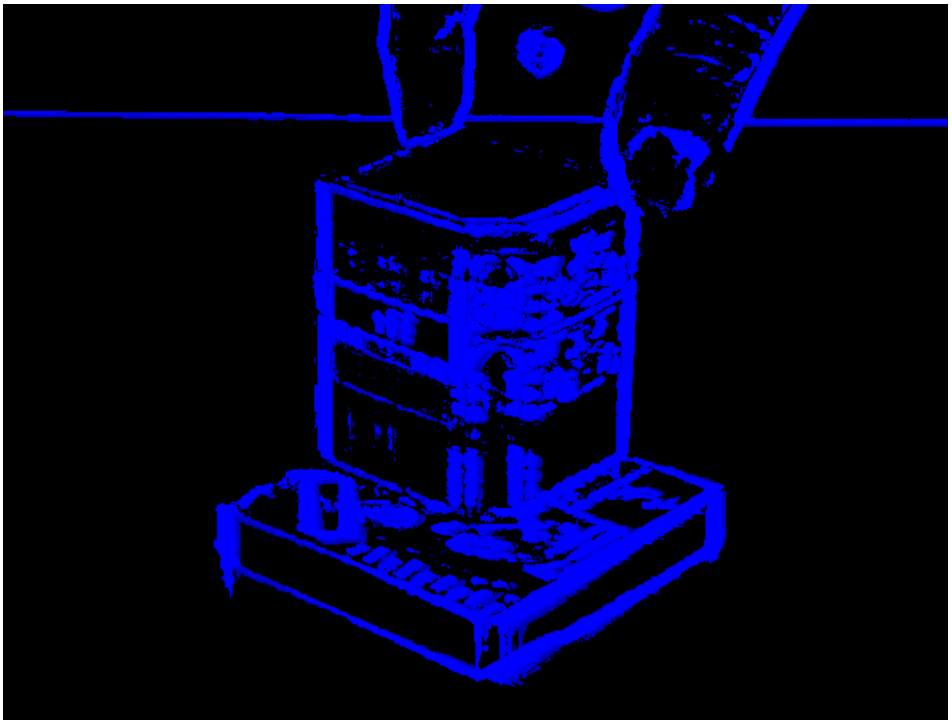

Figura 9 –Imagem de motion
Fonte: Elaboração própria

A imagem caracterizada por sua cor azul é chamada de Motion, pois essa foi a técnica utilizada para podermos prever o movimento e rotação do objeto. O motion é um evento que está contido na biblioteca que utilizamos para construção dos nossos métodos, o OpenCV.

A atual implementação já foi descrita no capitulo anterior. Este algoritmo ocasionou alguns problemas na sua implementação, pois são geradas duas imagens de retorno com os contornos horizontais e verticais e estas precisavam ser mescladas. No início mesclamos as duas imagens de retorno em uma só, mas vários problemas como capacidade de memória e desempenho do processador impossibilitaram a utilização de métodos de soma de imagens e por fim, ao aprendermos que poderíamos utilizar as variáveis de imagens

como matrizes, fizemos a aproximação do gradiente somando os dois gradientes direcionais obtidos com o método cvSobel.

Tendo obtido resultados favoráveis na detecção de bordas passamos a focar no tracking do objeto. Começamos por tentar fazer a utilização da técnica optical flow, mas ao nos aprofundarmos na pesquisa descobrimos que não seria útil localizar o movimento lateral do objeto mas sim o seu centro e para que lado o objeto está sendo rodado.

Passamos a utilizar o método SURF para fazer a detecção de movimentação dos pontos no vídeo, mas por fim não conseguimos realizar o cáculo da rotação do objeto através deste método e tivemos de seguir com nossa pesquisa.

Para sabermos para que lado o objeto está sendo rotacionado utilizamos a função update_mhi() que faz uso da função cvUpdateMotionHistory() para calcular a movimentação do objeto no frame e posteriormente fazer uso da função CalcMotionGradient().Passamos a recuperar os pontos-chave do objeto e o algoritmo do SURF é o responsável por essa tarefa. Passamos a reprocessar a parte de detecção dos pontos-chave para que pudéssemos achar os pontos mais importantes do objeto que serviria para a montagem do modelo gráfico.

Neste ponto geramos um novo vídeo com qualidade alta obtido de uma câmera melhor. Nosso algoritmo obteve um bom resultado achando mais pontos nas imagens mas o desempenho para processar as imagens caiu, o que era de se esperar devido ao tamanho da imagem (720p).

Nosso resultado final consta de três telas: uma com a imagem original sendo mostrada com do arquivo ou captura na hora sendo

processado (figura 2), uma contendo uma representação dos pontos que devemos considerar na hora de construir os polígonos para a modelagem (figura 8) e uma terceira imagem do objeto sendo rastreado pela técnica de motion (figura 9).

7 Considerações finais

Neste capítulo falaremos sobre as conclusões que chegamos com os resultados obtidos e analisados no capitulo anterior, bem como o quão satisfatório eles foram.

Devido a inexperiência com métodos de tratamento de imagem tivemos uma grande dificuldade para refinar os parâmetros dos algoritmos existentes para termos resultados que pudéssemos trabalhar. Depois de escolhido o método e feito seus ajustes, tentamos dar prosseguimento ao trabalho.

Com os resultados do tracking obtido pelo método SURF e o motion tinhamos agora o necessário para construir de fato o modelo. Fazendo uma analogia ao esquema apresentado por QI PAN, que foi nossa referência para o trabalho, estávamos no ponto onde ele consegue desenhar uma espécie de modelo somente com as retas que ligam os pontos chaves encontrados no seu algoritmo de tracking.

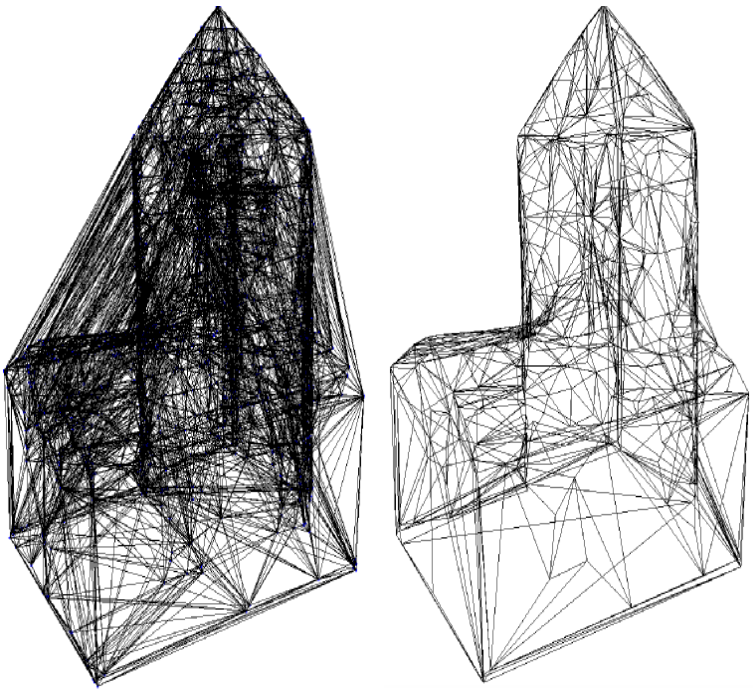

Figura 10 – Imagem de um frame do objeto estudado por Qi Pan, com seus pontos chaves interligados.
Fonte: Projeto PROForma

Chegamos nesse ponto, achamos os pontos chaves mas não conseguimos fazer a parte de construção e espelhamento das faces no objeto encontrado. Como mostra a imagem 8 vemos um exemplo de frame com os pontos encontrados.

Essa fase chamada de reconstrução possui etapas de ajuste do conteudo a ser analisado, recuperação da superficie do objeto e, o que consideramos as partes mais dificies de se reproduzir, triangularização de Delaunay com a lapidação do modelo.

Não conseguimos juntar todos os pedaços parciais do modelo obtido pelo tracking e com isso não fomos capazes de chegar nesse nível de modelagem. Reproduzindo os pontos chaves rapidamente, como fizemos no motion, conseguimos ter uma boa visão do que está sendo modelado. Porém sem os detalhes das curvas do objeto, e

também não sendo possivel navegar pelo modelo como se estivéssemos manuseando-o.

O retorno, ou feedback, que o usuário tem com o resultado do nosso projeto não é completo mas é bastante promissor pois conseguimos de forma rápida recuperar os pontos críticos da imagem e junta-los de uma forma dentro do programa que pode ser trabalhado mais adiante .

O resultado com várias telas retornando vários exemplos de implementações dos métodos testados nos auxiliou e muito a ajustar e escolher qual resultado estava satisfatório. Isso não incomodou muito na performance final então resolvemos deixar para efeito de apresentação. QI PAN faz uma junção dessas imagens como mostra em seu video de exemplo do ProFORMA.

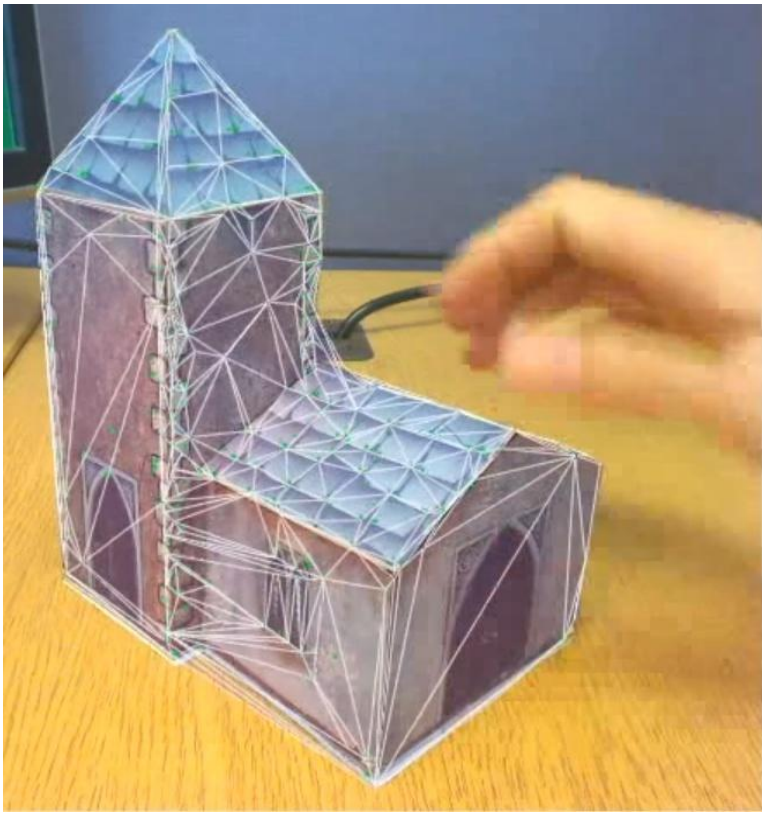

Figura 11 – Frame do nosso objeto de estudo com seus pontos chaves.
Fonte: Projeto PROForma

8 Trabalhos futuros

No capítulo um vimos uma série de trabalhos relacionados na área de modelagem, porém seus problemas em especifico geralmente precisavam de outros equipamentos e ações mais específicas da parte do usuário. Porem as técnicas aqui apresentadas poderiam ser testadas para verificar se há uma mudança significativa na qualidade dos dados obtidos do objeto que se deseja modelar.

Como constatamos, a modelagem em si não ficou completa, somente as partes críticas das imagens dos frames do vídeo do objeto foram capturadas e podem ser armazenadas para trabalhos futuros.

Para melhorar o projeto é preciso melhorar alguns pontos base. É necessário, para que o desempenho possa melhorar, fazer uma busca no frame e localizar o centro do objeto, assim excluindo os pontos que estão separados dos próximos do centro do objeto. Uma outra melhora em performance pode ser feita na busca de pontos na imagem, atualmente é feita uma busca em toda a imagem como um único objeto e poderia ser feito uma busca na imagem separada em pedaços menores e até mesmo excluir pedaços que não tenham nenhuma importancia nas buscas.

No projeto foi feita a estimativa de movimento capturando todos os pontos que se movem na imagem e isso tem o intuito de ser usado para mapear o objeto excluindo o fundo da imagem, pois no vídeo o objeto é a única parte que se move e nunca sai do foco.

Tendo armazenado todos os pontos encontrados do objeto, é preciso fazer uma junção dos pontos para que seja formada a geometria do objeto em questão e isso não deve desconsiderar a

captura de movimento, pois é este movimento que vai ditar a forma que os pontos serão ligado entre si. Após essa fase seria necessário rever o modelo e aparar as pontas que estão fora dos frame-chave, que seria o frame capturado no momento da captura dos pontos-chave.

Tendo o formato base da estrutura o próximo ponto seria passar o modelo para um formato de estrutura gráfica 3D e colar as fases, retiradas dos frame-chave, sobre a estrutura 3D(PUC-RIO, 2013).

9 Referências bibliográficas

[1] TOMASI, C. e KANADE, T. **Shape and motion from image streams under orthography: A factorization approach. International Journal of Computer Vision**. 1992.

[2] DEBEVEC, P., TAYLOR, C., and MALIK, J. **Modeling and rendering architecture from photographs: A hybrid geometry-and imagebased approach. In Proc. SIGGRAPH'96**. 1996.

[3] Página do INPE.br. Disponível em: <http://mtc-m05.sid.inpe.br/col/sid.inpe.br/jeferson/2003/07.08.08.27/doc/INPE%20-%209665%20-%20NTC.pdf>. Acesso em: 07 mar. 2013.

[4] Q. Pan, **ProFORMA: Probabilistic Feature-based On-line Rapid Model Acquisition**, 2009.

[5] Página da berkeley.edu. Disponível em: <http://www-video.eecs.berkeley.edu/research/indoor/>. Acesso em: 07 mar. 2013.

[6] Página da stanford.edu. Disponível em: <http://movement.stanford.edu/automatic-nr-modeling/TorrHertz-eccv04.pdf>. Acesso em: 07 mar. 2013.

[7] Página da autodesk.com.br. Disponível em: <http://www.autodesk.com.br/adsk/servlet/pc/index?id=15933484&siteID=1003425&mktvar004=463857>. Acesso em: 07 mar. 2013.

[8] Página da 123dapp.com. Disponível em: <http://www.123dapp.com/catch>. Acesso em: 07 mar. 2013.

[9] WIKIPÉDIA. Desenvolvido pela Wikimedia Foundation. Apresenta conteúdo enciclopédico. **Operador Sobel.** Disponível em: <http://en.wikipedia.org/wiki/Sobel_operator>. Acesso em: : 07 mar. 2013.

[10] Página da encyclopediaofmath.org. Disponível em: <http://www.encyclopediaofmath.org/index.php?title=Edge_detection>. Acesso em: 07 mar. 2013.

[11] Página da ethz.ch. Disponível em: <ftp://ftp.vision.ee.ethz.ch/publications/articles/eth_biwi_00517.pdf>. Acesso em: 07 mar. 2013.

[12] WIKIPÉDIA. Desenvolvido pela Wikimedia Foundation. Apresenta conteúdo enciclopédico. **Haar-like features.** Disponível em: < http://en.wikipedia.org/wiki/Haar-like_features>. Acesso em: 15 jun. 2012.

[13] Página da openvc.org. Disponível em: <http://docs.opencv.org/>. Acesso em: 07 mar. 2013.

[14] Página da puc-rio.br. Disponível em: <http://www.maxwell.lambda.ele.puc-rio.br/9828/9828_3.PDF>. Acesso em: 07 mar. 2013.

10 Apêndice

10.1 Formatos de imagens e mídias digitais

Aqui falaremos sobre termos específicos do problema e vários formatos usados normalmente.

10.1.1 Codecs

É o acrônimo de codificador e decodificador, podendo ser um dispoitivo ou um programa que trabalha em cima do fluxo de dados digitais. A codificação é feita para transmissões, armazenagem ou criptografia, e a decodificação é feita para reprodução e edição. Eles são a raiz de aplicações que usam videos.

10.1.2 MPEG

Formato de video bastante popular, padronizado pela Moving Picture Experts Group. Utiliza as compressões MPEG-1 e MPEG-2. Encontrado em abundancia nos videos da internet.

10.1.3 MP4

Para o uso da compressão MPEG-4 foi feito um outro format de video. A conversão dele é feita separando o video do audio. O audio é compremido com o modelo ACC, um tipo especifico para audio. O MP4 é bastante utilizado em tocadores portateis.

10.1.4 MOV

Utiliza um algoritmo de compressão proprietario criado pela Apple, utilizado para salvar videos. Bastante encontrado em plataformas Macintosh.

10.1.5 RMVB

Compressão da RealMedia, que utiliza um taxa de bit, oq eu determina a qualidade da imagem, variavel. Popularizado por ter um balanço bom entre espaço utilizado do arquivo comprimido e qualidade de imagem.

10.1.6 3GP

Desenvolvido pela 3rd Generation Partnership é um container que suporta video e audio. Foi feito para ser o formato multimedia para transmissão de audio e video entre celulares pela rede 3G, comunmente usado nesses aparelho somente. Foi atualizado para uma nova versão o 3G2.

10.1.7 AVI

A Microsoft criou esse formato de video para que contivesse menos compressão que o MPEG e o MOV. Ele também pode ser seu decodificado por varios codecs, podendo rodar em multiplos tocadores, mas estes precisam ter suporte ao codec utilizado.

10.1.8 WMV

Arquivo baseado no sistema de sistemas avanaçados da Mircrosoft (ASF), utiliza o compressor Windows Media.

10.1.9 MKV

MatroskaVideo, similamenter ao AVI, ASF e MOV, suport avarios tipos de Codecs. Popularizado por poder conter também legendas embutidas e ter alta qualidade de reprodução.

10.1.10 DIVX

Alta qualidade, alta compressão. Usado para distribução de midias como filmes em DVD. Suporta videos em alta resolução como a 1080 HD.

10.2 Qualidades de videos

10.2.1 SD(Standard Definition)

O termo se refere a vídeos com qualidade padrão, onde o aspecto da imagem na maioria das vezes obedece a razão de 4 para 3 (4:3), tendo dois tamanhos de tela disponiveis: 576 pixels e 480 pixels de altura. Onde um pixel é o ponto mínimo das imagens.

10.2.2 HD(High Definition)

Video em alta definição. Comumente relacionado com imagens com a resolução fixa de 1280×720 pixels (720p) ou 1920×1080 pixels (1080i/1080p), os números entre parênteses signifcam a altura ou o número de linhas na tela. A letra i e p após sua categorização significa o tipo de scaneamento que deverá ser feito da imagem. No caso do i a imagem está entrelaçada e os aparelhos devem converter a fonte para um scan progressivo. Basicamente a imagem “i” é feita a partir de 2 imagens de qualidade pior que são sobrepostas, para dar a impressão de alta qualidade os aparelhos transformam essas imagens em um fluxo único, como já construido o arquivo 1080p.

O trabalho visa captar imagens de câmeras de baixa qualidade e com framerate baixo. Nossa intenção é utilizando uma câmera de baixo custo poder modelar um objeto em tempo real. A utilização de câmeras com alta qualidade não impede o funcionamento do projeto. Esta mudança apenas reflete a qualidade final do modelo que pode ser altera utilizando um programa de edição de modelos gráficos.

10.3 Conversão de imagens

Percebemos que há muitos formatos de imagens existentes para armazenar e reproduzir videos e imagens. Consequentemente há um grande número de problemas relacionados com a conversão entre diferentes formatos, dentre eles, o principal problema é a perda de detalhes nas imagens.

Muitos desses formatos são proprietários, sendo assim, podem não haver programas que façam a conversão desses formatos para outros e vice-versa. Assim a compatibilidade dos arquivos com os softwares diminui bastante e muitas vezes estes só abrem em programas específicos. Quando disponíveis, essas conversões muitas vezes causam perda na qualidade da imagem, o que geralmente não acontece em transformações de imagens com maior qualidade para menor qualidade.

Essa operação de conversão pode ser relacionada com a transformação de qualquer arquivo em qualquer formato, mas de acordo com o substrato trabalhado não é possível realiza-lo com a mesma precisão. Muitas compressões podem ocorrer sem perda, como de arquivos texto onde algoritmos conseguem diminuir o tamanho e recupera-lo posteriormente com todas as informações intactas. O mesmo é muito dificil de ser atingido quando trabalhamos com certos tipos de dados virtuais como sons e imagens.

Uma das operações mais simples é a remostragem, que consiste em exibir a mesma imagem em tamanhos variados. Por isso que reproduzir uma imagem gravada em um celular, que encontra-se em um formato e tamanho planejado para tal aparelho, em um dispositivo com tela maior pode gerar resultados muito distorcidos, causando o efeito de aliasing. Este consiste no resultado da imagem que sofreu a interpolação, reconstrução, para um espaço onde a resolução de pixels não era compatível com a que foi gerada inicialmente. Esse é um dos motivos para que nós focássemos em câmeras de baixo custo, como descreveremos mais a frente.

10.4 Tecnoligias utilizadas

Neste capítulo serão abordados conceitos e fundamentos das tecnologias e padrões utilizados durante o projeto. Fizemos a adição desse capitulo para que o leitor que não esteja a par dessas tecnologias possa ter o completo entendimento sobre as soluções abordadas.

10.4.1 C++

C++ é um linguagem de programação com tipagem estática, multiparadigma e compilada. Desenvolvida por Bjarne Stroustrup é uma mesclagem de linguagens de

baixo e alto nível. Umas das linguagens mais populares, veio como melhorias para o C, sua origem, adicionando classes, funções virtuais, sobrecarga de operadores, herança múltipla, templates e tratamento de exeções entre outros.

Na escolha da linguagem as opções disponíveis eram Python e C++, que são as linguagens nas quais o framework OpenCV é implementado. O C++ foi escolhido pois os membros deste artigo possuem experiência com a mesma.

10.4.2 Visual Studio

Visual Studio é um ambiente integrado de desenvolvimento (IDE) da Microsoft. Este ambiente permite a organização do projeto e é um ambiente familiar aos membros deste artigo e possui integração com o Qt, framework utilizado para auxiliar o desenvolvimento. Ela é usada para desenvolver aplicações desktop e de interface gráfica, juntamente com aplicações Windows Forms, web sites, aplicações web e serviços web em ambos código nativo juntamente com código gerenciado para todas as plataformas suportadas pelo Microsoft Windows, Windows Mobile, Windows CE,.NET Framework, .NET Compact Framework e Microsoft Silverlight.

Utilizamos a versão 2008, pois temos acesso fornecido pela UFF e ele já englobava muitas funcionalidades.

10.4.3 Open CV

OpenCV é uma biblioteca de código aberto para processamento em tempo real de imagens. Inicialmente escrita em C, possui diversas interfaces para as liguagens Python, C#, Ruby e Java. Seu fácil uso, qualidade nos resultados e quantidade de opções de processamento que nos levaram a escolher o OpenCV.

Lançado em 1999 em um projeto de pesquisas da Intel como iniciativa para avançar em projetos que tinham um processamento intenso. Seus objetivos eram descritos como:

- Avançar na pesquisa de imagens fornecendo não somente um código otimizado mas também que fosse aberto.
- Disseminar o conhecimento de imagens provendo uma infraestrutura que os desenvolvedores pudessem construir sobre, tornando os códigos altamentes plugáveis.

- Tivesse aplicação comercial, para isso as licenças utilizadas nos códigos, portáteis, otimizados e de livre acesso, não precisavam ser livre ou aberta.

Este framework possui muitas funções de auxilio a atividade desenvolvida como as funções de captura de imagem da câmera e de vídeos, captura e tratamento de frame, busca de bordas e cantos, busca e comparação de pontos semelhantes e desenho em imagens. E também é suportado pelo Android, um dos maiores sistemas operacionais do mercado, tornando nossa idéia de aplicar o produto em celulares ainda mais viável.

10.4.4 Qt

Qt é um framework multiplataforma para desenvolvimento em C++ criado pela empresa norueguesa Trolltech. Com ele é possível desenvolver aplicativos e bibliotecas uma única vez e compilá-los para diversas plataformas sem que seja necessário alterar o código fonte. Qt é utilizado largamente no ambiente desktop livre KDE e na nova plataforma para dispositivos móveis da Nokia, Maemo (também livre).

Usa C++ padrão mas faz extenso uso do compilador de meta objetos [19], e diversos macros para enriquecer a linguagem. Pode ser usada para desenvolver em outras linguagens além do C++, funciona nos sistemas operacionais mais famosos e em algumas plataformas mobile.