Universidade de São Paulo
Instituto de Matemática e Estatística
Bachalerado em Ciência da Computação

Gianluca Takara Ciccarelli

Geração automática de quadros para animações utilizando Visão Computacional e Processamento de Imagens

São Paulo
Dezembro de 2017

Geração automática de quadros para animações utilizando Visão Computacional e Processamento de Imagens

Monografia final da disciplina
MAC0499 – Trabalho de Formatura Supervisionado.

Supervisor: Prof. Dr. Roberto Hirata Junior

São Paulo
Dezembro de 2017

Agradecimentos

Agradeço, primeiramente, a Deus, que me dá os desafios que me fizeram crescer, e me protege a todo momento.

A Meishu-Sama, que me mostrou o segredo da felicidade e o caminho da verdade e da salvação.

Aos meus pais, que são meus exemplos, e me deram todas as condições para viver e estudar.

Aos meus amigos, que me acompanham, me sustentam, e crescem junto comigo.

Ao meu orientador, e à responsável por esta matéria, que me ensinaram, e me auxiliaram no desenvolvimento deste trabalho.

A esta Universidade, que me disponibilizou toda a estrutura necessária para crescermos academicamente.

Aos meus professores, que fizeram parte da minha formação.

A meu chefe do trabalho, que foi paciente durante toda a minha graduação.

Enfim, agradeço a todos que fizeram parte da minha jornada até aqui.

Resumo

Na indústria de desenhos animados, uma das maiores dificuldades é a produção dos milhares de desenhos necessários para criar uma animação de boa qualidade. Apesar das diversas alternativas utilizadas para diminuir o trabalho, muitas afetam visivelmente na qualidade do produto final. Neste trabalho, será explorado um novo meio de gerar desenhos automaticamente, mas que mantém o aspecto natural de uma imagem desenhada por um artista. Isso será feito através de métodos em Visão Computacional e Processamento de Imagem, modificando as ilustrações originais para criar novas animações. Utilizando conceitos como detecção de *features*, costura de imagens, e transformações lineares, serão expostas novas utilidades para estas técnicas que já são tão utilizadas com imagens reais.

Palavras-chave: visão computacional, desenho animado, detecção de *features*.

Abstract

In the animation industry, one of the greatest challenges is to produce the thousands of drawings needed in order to create good quality animations. Despite the several alternatives applied in order to decrease work, many of them directly affect the visual quality of the final product. In this paper, a new method will be explored, one that creates drawings automatically, but retains the hand-drawn aspect of the original. This will be done using methods in Computer Vision and Image Processing, modifying the original illustrations to create new ones. Using concepts such as feature detection, image stitching and linear transformation, new uses will be discovered for these techniques that are popularly used with real images.

Keywords: computer vision, animation, feature detection.

Sumário

Capítulo 1

Introdução

O objetivo deste trabalho é estudar a aplicação da Visão Computacional e Processamento de Imagens na produção de desenhos animados.

À medida que a industria da animação cresce, a demanda por mais desenhistas também aumenta. No entanto, tal demanda não é acompanhada de boas condições de trabalho.

Devido ao grande número de imagens que precisam ser feitas para se produzir uma animação de boa qualidade, muitos animadores se veem forçados a passar noites no trabalho para cumprir prazos.

Uma das alternativas encontradas por estúdios de animação foi a utilização de Computação Gráfica para criar seus projetos. No entanto, nem todos adotaram esses métodos para substituir animações tradicionais, uma vez que há diferenças notáveis entre aquelas feitas manualmente e as geradas por Computação Gráfica.

Vale ressaltar que enquanto há empresas focadas em utilizar principalmente a Computação Gráfica, outras optam pelo desenho manual para criar suas animações, ambas buscando atender aos seus respectivos públicos.

Neste trabalho são estudadas alternativas para auxiliar na criação de desenhos animados, utilizando métodos nas áreas de Visão Computacional e Processamento de Imagens. Tais métodos utilizam transformações feitas sobre os desenhos do próprio artista. Assim, tem-se como objetivo facilitar a tarefa dos desenhistas, com a ressalva de que suas animações retenham a aparência adequada ao desenho original.

Capítulo 2

Processo de Animação

Neste capítulo será brevemente explicado o processo para criação de animações tradicionais e a sua indústria, que este trabalho tem como objetivo favorecer.

2.1 Definição

Uma animação é um conjunto de imagens que, dispostas em sequência uma após a outra, induzem a ilusão de movimento contínuo.

Essas imagens são chamadas de *frames*, ou quadros. A ilusão de movimento é criada graças à interpretação humana da sequência de imagens, mas não ocorre para qualquer conjunto de imagens. Para alcançar este efeito, cada par de *frames* adjacentes deve possuir informações o bastante para indicar qual é o movimento sendo realizado na transição de um para o outro.

Figura 2.1: *Nas figuras acima, é representada uma bailarina. No conjunto de cima, os frames não são o bastante para criar a ilusão de movimento. Ao serem expostos em forma de animação, eles aparentam ser apenas duas fotos sendo mostradas uma após a outra. O segundo conjunto, no entanto, já demonstra melhor uma ilusão de movimento, fazendo parecer que a bailarina está girando.*

Como demonstrado na Figura 2.1, mais *frames* criam uma melhor ilusão de movimento.

Pela lógica, é impossível criar uma quantidade suficiente de *frames* que descreva completamente um movimento contínuo, pois animações são discretas. Assim, não é necessário desenhar infinitos *frames* para descrever um movimento, basta criar *frames* o bastante para criar esta ilusão, e exibí-los em uma determinada frequência. Animações à 24 *frames*-por-segundo já produzem bons resultados.

Dentre as diferentes formas de criar animações, podemos destacar as Animações Computadorizadas (imagens são renderizadas a partir de modelos 2D ou 3D), Animações *Stop-Motion* (feitas utilizando fotos de cenas reais), e o estilo que será estudado mais a fundo neste trabalho, Animações Tradicionais.

2.2 Animação Tradicional

Animações tradicionais são as mais antigas dentre os três tipos mencionados. Ela consiste em desenhar manualmente cada *frame* da animação. Para explicar sua produção, serão utilizadas algumas imagens (Figura 2.2 e Figura 2.3) retiradas do documentário [*Inside Toei Animation* 2008].

Com a descoberta da animação computadorizada, em que modelos 2D poderiam ser criados pelo computador e movimentados livremente, este tipo de animação caiu um pouco em desuso. No entanto, hoje computadores são utilizados para auxiliar os próprios artistas a desenhar seus *frames* com mais ferramentas e mais apoio, o que ajudou muito na produção deste tipo de animação.

Figura 2.2: *Nesta figura, observamos um animador desenhando os primeiros traços de um frame.*

Como pode ser visto na Figura 2.2, animadores começam seus desenhos com traços mais simples, indicando a posição do personagem, assim como outros elementos pertinentes ao *frame*, como objetos que também estão em movimento. Após finalizarem os traços, o desenho ainda passa por outros processos, como colorimento, iluminação, e outros efeitos especiais.

2.3 Tipos de *Frames*

Os *frames* que compõem uma animação podem ser divididos em duas categorias principais:

- Quadros Chave (*Keyframes*)
- Quadros Intermediários (*Inbetweens*)

A diferença entre eles está em seu propósito, e quem os desenha.

Keyframes compõem a estrutura principal da animação. Cada par de *keyframes* adjacentes define um movimento, que pode conter o deslocamento de elementos na cena, mudanças de expressões faciais em personagens, e outros aspectos de uma cena capturados pontualmente em apenas um *frame*. Os artistas mais experientes são encarregados de desenhar *keyframes*, pois estes são mais notáveis para aqueles que assistem à animação, e também servem como base para a criação dos *inbetween frames* (explicados a seguir). Apesar de sua importância, *keyframes* descrevem apenas os momentos mais particulares de uma animação. Uma animação feita somente com *keyframes* possui pouca transição entre cada momento, enfraquecendo a ilusão do movimento contínuo. aparentando ser uma sequência de fotos tiradas em momentos diferentes, e não uma série de *frames* que tem como objetivo descrever um momento inteiro.

Para ligar os *keyframes* e tornar a animação completa, são necessários os *inbetween frames*, ou *inbetweens*. Seu propósito é suavizar a transição entre dois *keyframes*, caso a diferença entre os dois seja grande demais para que um observador consiga inferir facilmente uma transição entre os dois. Na sequência de quadros numa animação, *inbetweens* são colocados entre pares de *keyframes*. Enquanto *keyframes* são desenhados independentemente entre eles próprios, *inbetweens* são sempre baseados em *keyframes* já existentes. Eles são desenhados por artistas iniciantes, ou equipes especializadas em sua criação, e por serem muito mais numerosos do que *keyframes*, requerem mais mão de obra e mais tempo de trabalho.

Para desenhar *inbetweens*, é comum que animadores sobreponham dois *keyframes* em seus computadores para observar suas diferenças, e então desenhar os *inbetweens*, utilizando sua própria percepção para saber quantos são necessários, como pode ser visto na Figura 2.3.

Ao serem usados em conjunto, *keyframes* e *inbetweens* formam uma animação completa, e esta divisão de *frames* inclusive ajuda na produção da obra. *Keyframes* estão em menor número, e descrevem apenas os momentos principais da animação. *Inbetweens* são feitos baseados em pares de *keyframes*, e descrevem uma transição suave entre eles.

2.4 Demanda

A indústria de animações japonesas é hoje a maior representante de animações tradicionais, portanto, é comum que mudanças neste local influenciem na produção de animações

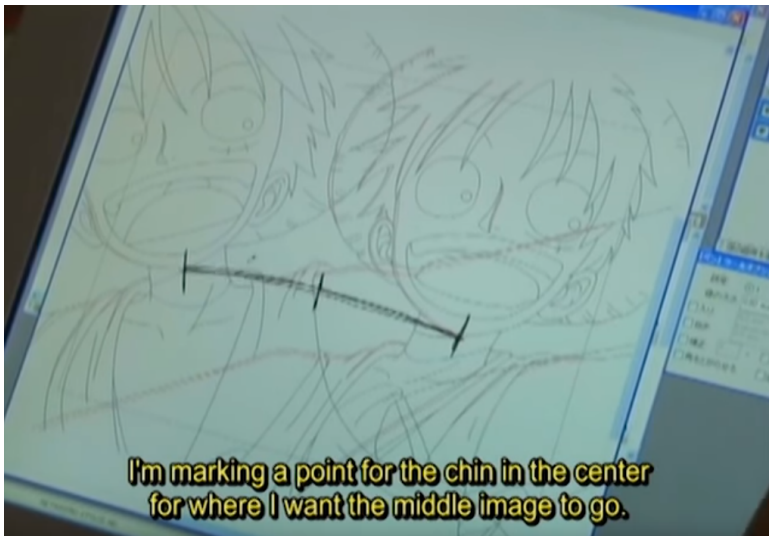

Figura 2.3: *Nesta figura, observamos um animador desenhando sobrepondo dois keysframes para compará-los, e então começar a desenhar o(s) inbetween frame(s) correspondentes.*

tradicionais em qualquer lugar do mundo. Ao longo dos últimos anos, houve um grande aumento na demanda por animações japonesas [Blair (2017)]. Isso ocorreu após este gênero, que já era popular mundialmente, ganhar ainda mais investimento através de empresas de transmissão de séries *online*, como *Netflix* e *Amazon*, e a criação de serviços especialmente criados para transmitir estas animações japonesas, como *Crunchyroll*. Esta mudança impulsionou a produção de animações, pois estas estavam apenas disponíveis através das transmissões na televisão, ou pela venda de DVDs e *Blu-Rays*.

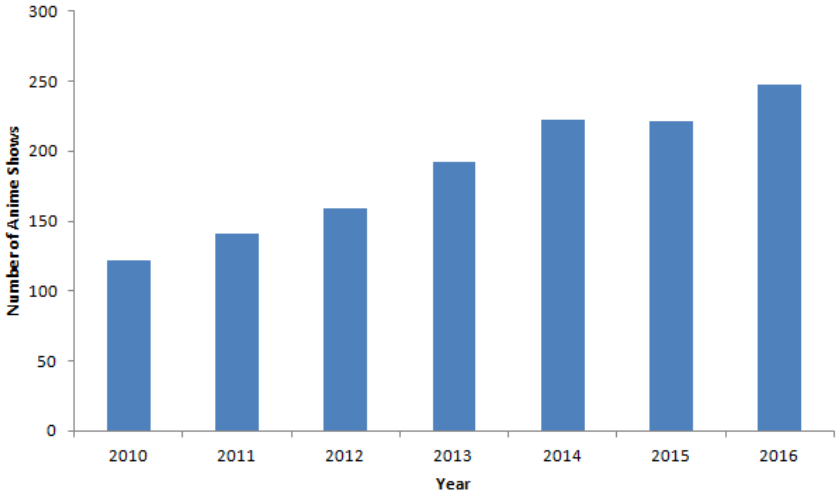

Figura 2.4: *Comparação de quantos animes foram lançados por ano [kVin (2017)].*

A resposta à demanda por novas animações pode ser vista na Figura 2.4, que mostra o aumento no número de animações japonesas a cada ano, sendo que o número quase dobrou em 2016 comparado a 2010. Isso está distribuído entre diversos estúdios, que também estão aumentando em número.

Com a demanda interna já em crescimento, em adição à demanda internacional, os ani-

madores começaram a passar por dificuldades para finalizar seus trabalhos. Muitos chegam a pernoitar nos estúdios para entregar a animação a tempo, e há estúdios que escolheram tercerizar a criação de seus desenhos, principalmente *inbetween frames*, para outros países como China e Coréia do Norte. Estes dois países também estão, aos poucos, investindo cada vez mais em suas próprias indústrias de animação, aumentando ainda mais a falta de mão de obra e a demanda.

Por conta disso, os estúdios estão buscando formas criativas de manter a qualidade, e diminuir a quantidade de trabalho dos animadores. Alguns utilizam Computação Gráfica para sanar essa necessidade, substituindo, em alguns casos, desenhos manuais por modelos 3D estilizados para terem a aparência de um desenho, como pode ser visto na Figura 2.5. No entanto, com poucas exceções, fica clara a diferença entre uma animação computadorizada e um desenho feito a mão. Projetos cuja premissa realmente é utilizar animações computadorizadas por conta do estilo e do publico alvo do produto, e não para substituir animações tradicionais, estão sendo bem recebidos. Porém, este é um mercado separado do de animações tradicionais.

Figura 2.5: *Berserk (2016) é uma produção japonesa que utiliza muitos elementos criados por Computação Gráfica. Nesta imagem é possível notar a diferença entre animações computadorizadas e animações tradicionais. Na esquerda, a animação original, que foi exibida na televisão, e na direita, a versão lançada em Blu-Ray meses depois, refeita usando desenhos manuais por terem mais tempo para finalizá-lo.*

2.5 Relação com este trabalho

O objetivo deste trabalho é justamente estudar um método de ajudar desenhistas em seus trabalhos. Seguindo tudo que foi mencionado na seção anterior, este método deve ser capaz de reduzir o trabalho manual realizado por eles, mas ao mesmo tempo ser capaz de gerar imagens visualmente similares ao que o próprio artista desenhou.

Visto que já estão aplicando modelos em computação gráfica aos desenhos (como visto na Figura 2.5), seria interessante utilizar outros conceitos para esta tarefa.

Por isto foi escolhido o campo da Visão Computacional, capaz de analizar imagens e transformá-las em dados, e do Processamento de Imagens, que transforma imagens em novas imagens. Através destas duas áreas, é resolvido o problema de gerar imagens visualmente similares às originais: é necessário apenas analisar alguns desenhos do artista, e utilizar estas

informações para gerar novos desenhos a fim de completar a animação. No capítulo seguinte, serão apresentados os passos deste método, exemplos de sua aplicação, e será mostrado que seu melhor uso é para a geração de *inbetween frames*.

Capítulo 3

Desenvolvimento

Neste capítulo será explicado todo o processo de desenvolvimento do método de geração automática de *frames*. Isso inclui algumas idéias descartadas, mas que trouxeram novos conceitos que foram usados no produto final, e explicações mais detalhadas sobre os algoritmos que são aplicados neste método.

3.1 Tecnologias Utilizadas

Toda o código escrito neste processo foi feito em Python 3.6, e está disponível no mesmo endereço deste documento[1]. Os algoritmos de detecção e descrição de *features* e descoberta de matriz de homografia foram aplicados utilizando a implementação disponível na biblioteca OpenCV [Garage *et al.* (2000–2017)].

3.2 Detecção de *Features*

Para criar *frames* novos, é necessário, primeiramente, descrever de alguma forma cada imagem para que seja possível transformá-la. Não há modelos matemáticos definidos contendo as características dos *frames*, pois são desenhados livremente pelo artista, então se vê necessário descrevê-los matematicamente utilizando Visão Computacional.

Dentre os diversos estudos realizadas na área, é possível encontrar vários voltados para a Detecção e Descrição de *Features*, ou características, de imagens [Tuytelaars e Mikolajczyk (2008)] [Krig (2014)]. Essas *features* são partes da imagem que podem ser facilmente distinguidas do resto. Apesar de estes métodos terem sido estudados e aplicados em imagens reais, como fotos, eles também funcionam com desenhos, como será estudado abaixo.

Adiante, serão estudadas boas formas de detectar estas *features*, mas já é possível definir que este é o primeiro passo do algoritmo.

[1] https://linux.ime.usp.br/~giantakara/mac0499/source.html (acessado em 27/11/2017)

Passo 1:

Encontrar e descrever as *features* de cada *frame*.

A aplicação destes descritores em imagens reais já foi e continua sendo muito estudada, mas pouco foi encontrado sobre sua eficácia com desenhos. Inicialmente, o problema deste trabalho foi explorado utilizando o descritor de cantos introduzido por Harris e Stephens [Harris e Stephens (1988)].

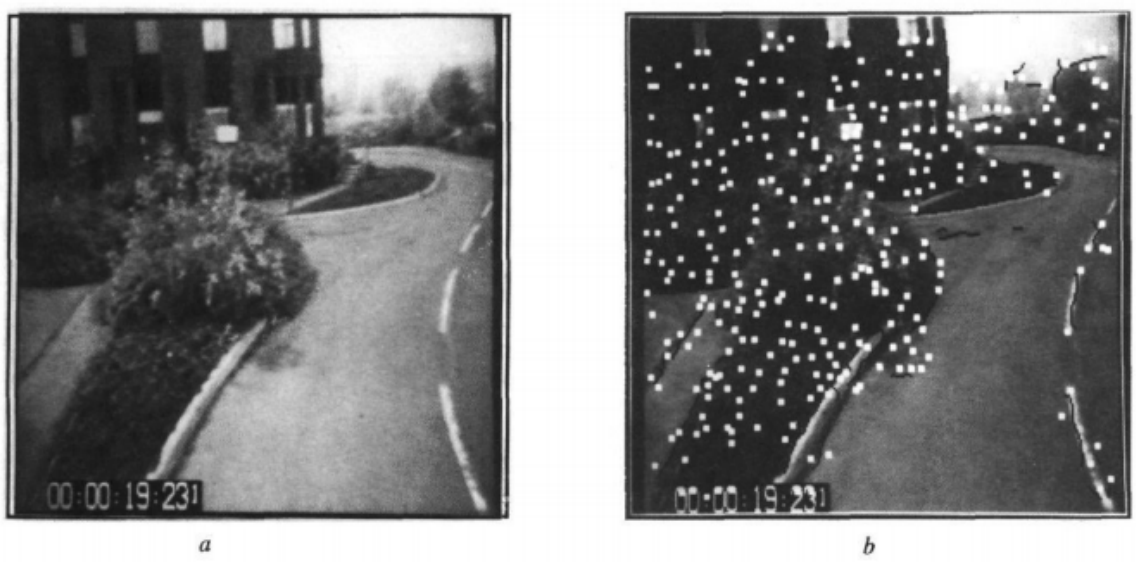

Figura 3.1: *Harris Corner Detector aplicado sobre uma imagem. Os pontos brancos indicam os vértices encontrados pelo algoritmo e marcados sobre a figura.*

Como pode ser visto na Figura 3.1, este descritor é capaz de detectar vértices em imagens. Apesar de sua eficácia, este algoritmo detecta apenas pontos, e não regiões. Em uma foto real, isso não é um grande problema, pois há vários tons de diferentes cores, o que facilita a descrição do algoritmo. Porém, em desenhos animados, grandes regiões dos objetos que se movem em uma cena são compostos por uma única cor, o que diminuiria o nível de descrição que o detector poderia gerar. Como visto na Figura 3.2, os pontos estão localizados justamente nas arestas da bola, não contemplando os espaços em branco e preto.

Por conta destes problemas, a opção escolhida foi realizar testes com outro descritor, o SIFT.

3.3 SIFT e *Keypoints*

O SIFT [Lowe (2004)] (*Scale-Invariant Feature Transform*) é um descritor de imagens desenvolvido por David G. Lowe, com o propósito de tornar a detecção de *features* em imagens mais robusta. Uma das principais características do SIFT em relação aos descritores que foram criados antes dele é a busca por regiões na imagem, e não apenas pontos.

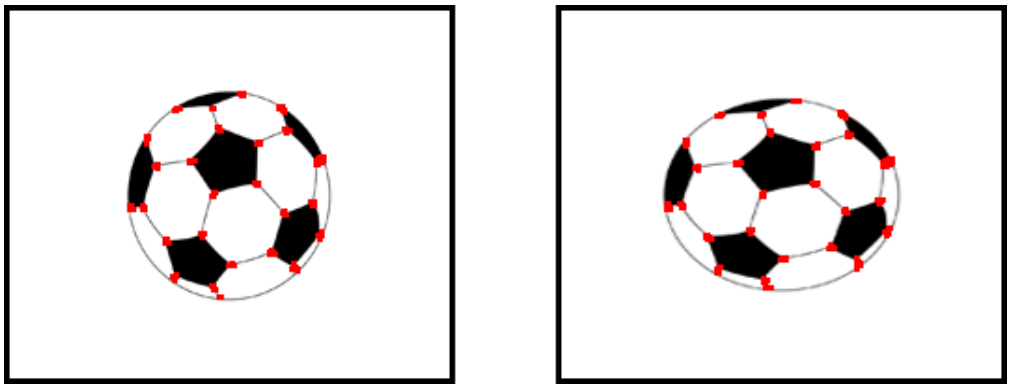

Figura 3.2: *Harris Corner Detector a consistência ao detectar vértices em uma mesma figura, antes e depois de ela sofrer uma transformação linear para alterar seu tamanho.*

Por conta do uso deste novo descritor, é necessário explicar o conceito de *Keypoints*. Um *Keypoint* é a representação gráfica da *feature* em questão, ou seja, é a parte da imagem correspondente à sua *feature*. Junto de cada *Keypoint* há também um Descritor, que representa numericamente diversos aspectos da *feature*. Tanto os *Keypoints* como os Descritores são utilizados pelo algoritmo SIFT.

O SIFT é de grande ajuda para esta pesquisa por dois motivos: sua capacidade de descrever e reconhecer uma mesma região cujo tamanho foi alterado (assim como o seu nome sugere, *Scale-Invariant*), e sua detalhada descrição, que auxilia no reconhecimento de *features* em *frames* diferentes. Além disso, por ser capaz de descrever regiões, este algoritmo se prova útil para a análise de desenhos, uma vez que várias regiões da imagem são compostas uniformemente por apenas uma cor. Estas, por sua vez, seriam ignoradas caso estivessemos buscando apenas por pontos.

Na Figura 3.3, pode se observar o SIFT em ação. Os círculos na imagem em preto-e-branco são os *keypoints* descritos. Os segmentos de reta dentro destes círculos representam a sua direção, um conceito que será aprofundado nas seções seguintes.

O SIFT considera cores em um intervalo $[0, 1]$, e seu funcionamento pode ser dividido em quatro etapas:

- Detecção de extremidades
- Localização de *keypoints*
- Associação de orientações
- Descrição dos *keypoints*

3.3.1 Detecção de extremidades

Para ter certeza de que o descritor consegue encontrar *keypoints* que se mantêm em variações de uma mesma imagem, independente de sua escala, ele analiza várias escalas ao mesmo tempo.

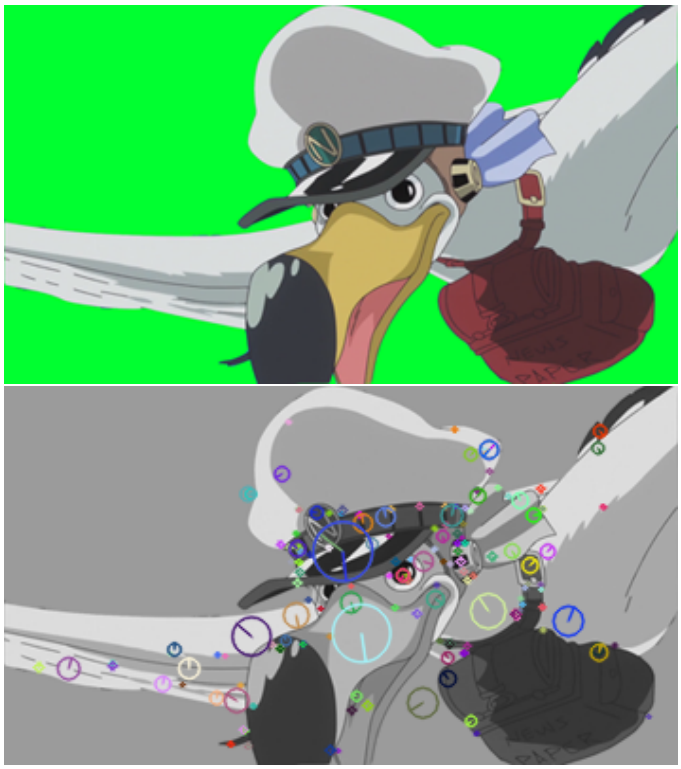

Figura 3.3: *Utilizando um frame retirado de uma animação, e removendo o plano de fundo para não afetar na análise, o algoritmo SIFT foi capaz de identificar os keypoints, resultando em sua representação visual na imagem de baixo.*

Isso é feito utilizando a função *Laplacian-of-Gaussian*. Primeiramente, uma certa escala gaussiana σ é aplicada em uma imagem $I(x, y)$, sendo x e y a localização do pixel sendo computado. Esta imagem é subtraída da mesma imagem, mas após ter outra escala gaussiana k-vezes menor aplicada a ela. Esta subtração é definida por $D(x, y, \sigma)$:

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}$$

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma)$$

Com a imagem analizada sob diferentes escalas, o algoritmo agora compara cada ponto com seus vizinhos, tanto laterais (numa área 3x3) quanto em escalas diferentes (1 acima e 1 abaixo), e marca os pontos que são maiores do que todos os seus vizinhos, ilustrado na Figura 3.4. Estes pontos são chamados de extremidades da imagem, e se tornam *keypoints* candidatos.

3.3.2 Localização de *keypoints*

Utilizando a Expansão de Taylor até o membro quadrático, a função da diferença entre duas escalas pode ser analizada com mais precisão, eliminando algumas extremidades que poderiam se tornar *keypoints* ruins para comparação por serem sensíveis a ruídos na imagem,

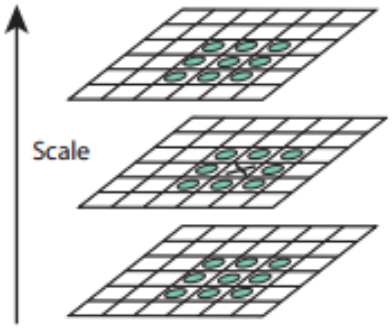

Figura 3.4: *Para encontrar os máximos e mínimos locais, o algoritmo compara a imagem numa escala σ com 1 escala acima e 1 abaixo, além de verificar os vizinhos numa área 3x3.*

ou por estarem mal-localizados.

$$D(X) = D + \frac{\partial D^T}{\partial X} + \frac{1}{2} X^T \frac{\partial^2 D}{\partial X^2} X$$

A expansão tem como origem o ponto sendo analisado, sendo X = $(x, y, \sigma)^T$ o quanto foi movido em relação ao ponto analisado. Busca-se então um extremo $\hat{X}$ para ser analisado na função diferencial de imagens. De acordo com o artigo original, se $|D(\hat{X})| < 0.03$, o ponto pode ser descartado com segurança.

$$\hat{X} = -\frac{\partial^2 D}{\partial X^2}^{-1} \frac{\partial D}{\partial X}$$

$$D(\hat{X}) = D + \frac{1}{2} \frac{\partial D}{\partial X}^T \hat{X}$$

3.3.3 Associação de orientações

A direção em que o *keypoint* aponta é calculada para facilitar a comparação entre *keypoints*. Alinhando ambos *keypoints* na mesma direção, é possível detectar sua equivalência, independente de rotação.

Para uma determinada escala Gaussiana, são calculados a magnitude (m) e orientação (θ) de cada ponto.

$$m(x, y) = \sqrt{(L(x+1, y) - L(x-1, y))^2 + (L(x, y+1) - L(x, y-1))^2}$$

$$\theta(x, y) = tan^{-1}((L(x, y+1) - L(x, y-1))/(L(x+1, y) - L(x-1, y)))$$

As operações são feitas utilizando subtração de pixels. Após todas as magnitudes e orientações terem sido calculadas, é montado um histograma em torno de cada *keypoint*. O valor de pico mais alto é tomado como valor de orientação do *keypoint*. Além disso, caso outros picos sejam encontrados que estejam próximos da orientação aceita, um novo *keypoint* é criado, com o mesmo local e escala, mas com orientação diferente. Isso pode ser observado na

Figura 3.5, que é uma imagem descrita pelo SIFT. Alguns pontos (como o que está próximo do canto inferior esquerdo), possuem duas flechas saindo da mesma origem.

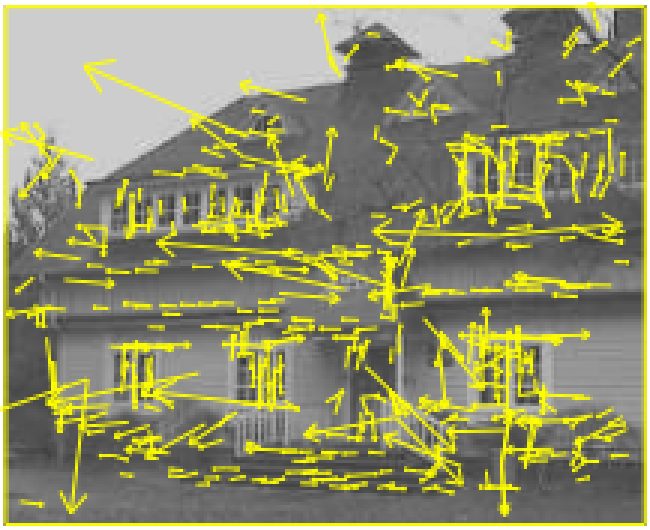

Figura 3.5: *Uma imagem, após passar pela detecção de extremos e pelo refinamento de keypoints. Flechas amarelas indicam as orientações dos keypoints, onde eles estão (sua origem) e magnitude (tamanho da flecha).*

3.3.4 Descrição dos *keypoints*

Com as operações anteriores, foram descobertos a localização, escala e orientação para cada *keypoint*. Agora, eles serão descritos com mais profundidade, para permitir comparações entre imagens que foram rotacionadas em movimentos 3D, mudanças de perspectiva, ou outros movimentos mais complexos. Todo este processo é melhor interpretado observando a Figura 3.6.

Uma vizinhança 16x16 é criada ao redor do *keypoint*, sendo divididas em 4 regiões, cada uma com área 4x4. Cada elemento é calculado utilizando a magnitude do ponto, e então modificados proporcionalmente em uma janela gaussiana, tendo o *keypoint* como origem. As magnitudes são somadas às magnitudes dos pontos com a mesma direção e que estão no mesmo intervalo 4x4.

Após isso, é formado um vetor com cada uma das informações calculadas. Este é normalizado para tamanho unitário. Isso é feito para que alterações na luminosidade não afetem a descrição dos pontos, pois uma mudança deste tipo implica num aumento igual a todos os pontos, e como os valores são calculados utilizando diferença entre pixels, eles permaneceriam os mesmos após a mudança na iluminação. Além disso, tratamentos adicionais são feitos para desconsiderar mudanças de iluminação não-lineares, como por exemplo, em modelos 3D cuja superficie é afetada diferentemente em cada um de seus pontos. Feito isso, é encerrada a busca e descrição de *keypoints* na imagem.

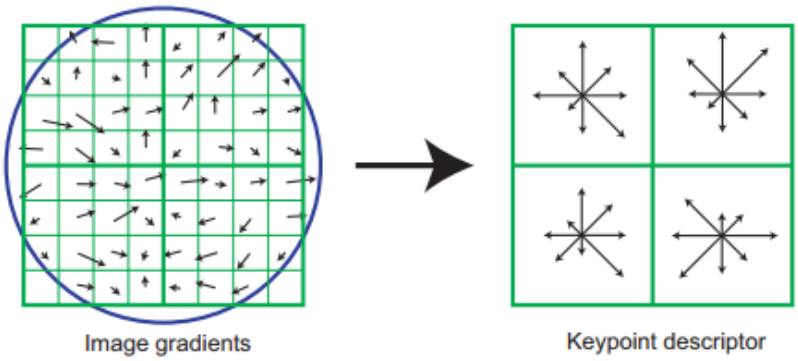

Figura 3.6: *Uma representação gráfica do descritor SIFT de keypoints. Na esquerda, uma vizinhança 16x16, cada elemento com sua respectiva direção e magnitude, envoltos de uma janela gaussiana. Na direita, um conjunto de 4 elementos, correspondendo à soma das magnitudes.*

3.4 Correspondência entre *Keypoints*

Com os *keypoints* de dois *frames* adjacentes descritos, será realizada a correspondência entre eles, o que significa encontrar um conjunto de *keypoints* que estejam presentes em ambos os *frames*.

Devido à descrição numérica dos *keypoints* feita no passo anterior, expressada em forma de vetor, é possível aplicar diversos meios para analisar a semelhança. Para este estudo, foi utilizada a Distância Euclideana dos elementos nos vetores de cada descritor, conforme recomendado pela documentação do OpenCV.

Passo 2:

Fazer a correspondência entre *keypoints* de dois *frame* adjacentes.

Como pode ser visto na Figura 3.7, o mesmo exemplo do pássaro agora passa pela correspondência entre *keypoints*, sendo que a grande maioria destas correspondências estão corretas.

Conforme pode ser visto, quando mais semelhanças entre os *frames* analisados, melhor a correspondência entre seus *keypoints*. Por isso, é recomendado utilizar este método entre dois *keyframes* adjacentes, por serem mais parecidos. E caso ainda não haja correspondências o bastante, é possível comparar um *keyframe* e um *inbetween* também, a fim de gerar mais *inbetweens*.

3.5 Homografia

Este algoritmo foi baseado em trabalhos feitos sobre Costura de Imagens, ou *Image Stitching* [Brown e Lowe (2007)], uma técnica que utiliza a correspondência entre *keypoints* para unir fotografias tiradas de um mesmo local, mas em diferentes posições ou ângulos.

A princípio, este assunto pode parecer não ter nenhuma relação com o problema de interpolar *frames*. Porém, os métodos nele utilizados são de grande utilidade. Ao costurar

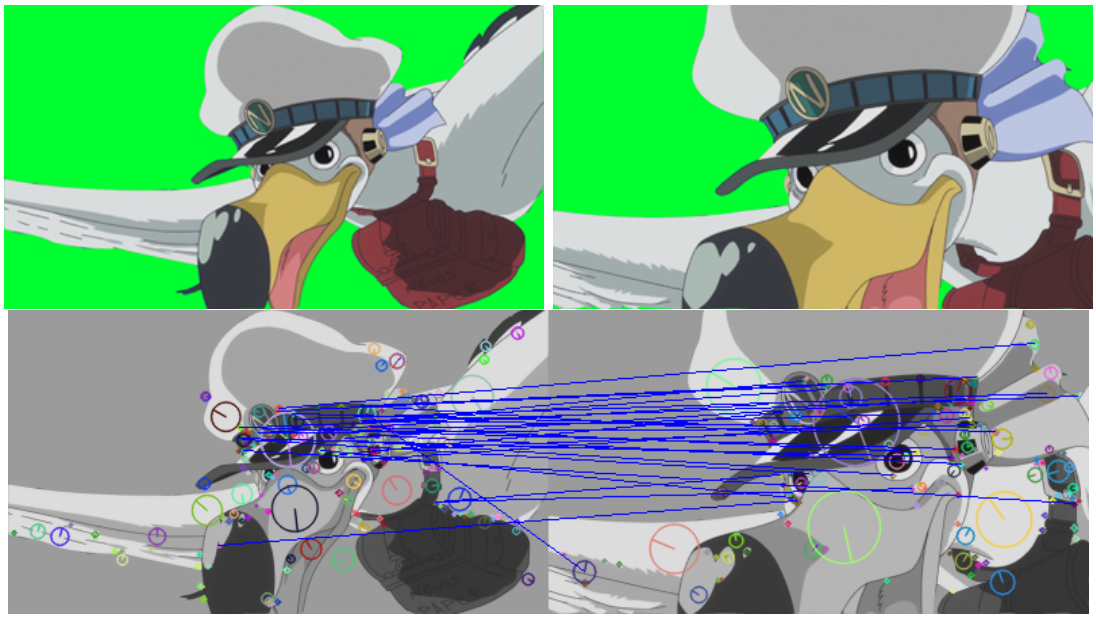

Figura 3.7: *O SIFT sendo aplicado em dois frames diferentes. Em cima, os desenhos originais, e em baixo, a representação visual dos keypoints (círculos), e a correspondência entre eles nos dois frames. É possível notar diversos outliers, keypoints correspondidos e com descrições semelhantes, mas que visualmente fica claro que não são a mesma feature.*

duas imagens, I_1 e I_2, o algoritmo está, em termos mais simples, buscando os *keypoints* mais parecidos, transformando I_1 para que ela se assemelhe a I_2 em ângulo e posição, e posicionando uma sobre a outra, obtendo como resultado uma única imagem, sendo que a transformação aplicada é expressada através de uma matriz de transformação.

Esta matriz de transformação H, ou Homografia, descreve como transformar uma imagem para que os *keypoints* correspondidos fiquem na mesma posição, ou o mais próximo possível, dos mesmos *keypoints* na outra imagem. Portanto, se fosse feita uma interpolação linear, começando de uma matriz de transformação H_0 que não modifica a imagem, ou seja, transforma I_1 em I_1, e terminando na matriz H, que transforma I_1 em I_2, seria possível obter inúmeras matrizes intermediárias, que por sua vez transformariam I_1 em $I_{1,1}$, $I_{1,2}$ ... e assim em diante, cada vez ficando menos parecido com I_1, e mais parecido com I_2. Esta é a mesma definição de *inbetween frame*, um conjunto de novos *frames* que servem de transição entre dois *keyframes*.

Ao invés de duas imagens quaisquer, I_1 e I_2, agora serão utilizados dois *frames*, f_x e f_{x+1}, sendo este um par de *frames* adjacentes de uma mesma cena.

Passo 3:

Encontrar Matriz de Homografia de f_x para f_{x+1}.

Primeiramente, é necessário encontrar a matriz H. Para isso, conforme Brown e Lowe (2007) mencionam em seu artigo, é utilizado o algoritmo RANSAC (*Random Sample Consensus*), desenvolvido por Fischler e Bolles (1980), e a implementação do OpenCV.

Este algoritmo utiliza um conjunto de amostras para estimar um modelo matemático que consiga melhor descrevê-la, sendo que ele espera que dentre essas amostras existam *outliers*, que são elementos que não pertecem ao modelo, e que os elementos que realmente pertencem estão sujeitos a ruídos.

Aplicando este algoritmo, é possível estimar a matriz de homografia H que melhor descreve a transformação necessária.

Figura 3.8: *No topo, os dois frames originais. Em seguida, o primeiro frame após ser transformado pela homografia H em relação aos keypoints correspondidos entre os dois frames. Por último, o segundo frame sobreposto na imagem transformada.*

Utilizando novamente o exemplo da animação do pássaro, visto na Figura 3.8, são realizados os mesmos procedimentos de *stitching*: Fazer a correspondência de *keypoints*, encontrar H, aplicar a transformação, e sobrepor as imagens. Na imagem resultante, é possível ver que houve um bom encaixe entre as imagens na região da bolsa, porém a região do bico está desalinhada. Esta é uma das consequências de utilizar transformações lineares, a imagem inteira é transformada da mesma forma, então há encaixes bons e outros ruins. Uma possível alternativa é discutida na seção 4.1, mas para o objetivo deste trabalho, esse tipo de trans-

formação é o suficiente, e serão utilizados exemplos mais símples que podem ser expressados apenas por transformações lineares.

A matriz H descoberta para este exemplo foi a seguinte:

$$H = \begin{pmatrix} 1.1016 & -0.2469 & -29.4403 \\ 0.0005 & 1.1310 & -38.1937 \\ -0.0005 & -0.0007 & 1.0000 \end{pmatrix}$$

Sua composição, e o método para interpolar novas matrizes, serão estudados na seguinte seção.

3.6 Decomposição da Transformação

Uma vez definida a Matriz de Transformação Linear H de um *frame* f_x para o *frame* seguinte, f_{x+1}, é necessário criar novas matrizes que descrevem transformações lineares para interpolar *frames* intermediários. Nesta seção será definida a composição desta matriz, explicitando todas as variáveis envolvidas, e como é possível montar novas matriz a partir delas.

Passo 4:

Decompor a Matriz de Homografia.

A H é composta por outras matrizes[2] da seguinte forma:

$$H = AR$$

Sendo A a Matriz de Posicionamento da Câmera, e R a Matriz de Rotação. Apesar de se tratar de imagens 2D, são utilizadas matrizes capazes de fazer rotações em 3 eixos, para que os efeitos de distorção na imagem sejam atingidos.

Primeiramente, a matriz R rotaciona a imagem original nos planos X, Y e Z. Cada uma destas três rotações possui sua própria matriz, definidas da seguinte forma:

$$R = ZYX$$

$$X = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\theta_x) & -sin(\theta_x) \\ 0 & sin(\theta_x) & cos(\theta_x) \end{bmatrix}$$

[2] https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html?highlight=findhomography (acessado em 16/11/2017)

$$Y = \begin{bmatrix} cos(\theta_y) & 0 & sin(\theta_y) \\ 0 & 1 & 0 \\ -sin(\theta_y) & 0 & cos(\theta_y) \end{bmatrix}$$

$$Z = \begin{bmatrix} cos(\theta_z) & -sin(\theta_z) & 0 \\ sin(\theta_z) & cos(\theta_z) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Sendo θ_z, θ_y e θ_x os ângulos de rotação em Radianos nos eixos X, Y e Z, respectivamente. Na documentação do OpenCV, e no código criado para este trabalho, a matriz R na verdade é uma matriz 3x4, cuja última coluna é composta por valores que transladam a imagem. O uso destes valores é desnecessário pois a translação já é feita através da matriz seguinte, e portanto sempre são inicializados como 0.

A matriz A é definida por:

$$A = \begin{bmatrix} e_x & s & c_x \\ 0 & e_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Na documentação do OpenCV, esta matriz descreve as propriedades da câmera, como ponto focal e posição. Porém, ela também pode ser definida como uma matriz de transformação linear que altera o tamanho da imagem, baseado nos valores e_x e e_y, faz a translação dela, baseado em c_x e c_y, e aplica o efeito de *shearing* baseado no valor s. Além disso, na documentação do OpenCV os valores de translação são referidos como f_x e f_y, porém opta-se por chamá-los de e_x e e_y neste trabalho pois a notação f já é utilizada pela numeração dos *frames*.

Com a matriz H em mãos, foi utilizada a função decomposeProjectionMatrix do OpenCV para decompô-la nas matrizes R e A. Então, foram utilizados os valores de R para descobrir os três ângulos de rotação, e a matriz A não precisa der decomposta, pois as variáveis necessárias já estão explícitas nela. Além disso, como fica claro nas Figuras 3.9 e 3.10, aplicar cada matriz separadamente, ou aplicar diretamente a matriz H, têm o mesmo resultado, cuja única diferença é a queda de resolução ao aplicar matrizes separadamente, por conta das diversas transformações que a imagem sofre.

Finalmente, foram obtidos os valores de todas as variáveis existentes no problema: θ_x, θ_y, θ_z, e_x, e_y, c_x, c_y e s. Estes são os valores que realizam a transformação do *frame* f_x em f_{x+1}. Sabe-se também como cada matriz que compõe H é construída, então é possível criar novas matrizes com os valores desejados durante a interpolação.

Original

$$X = \begin{pmatrix} 1.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.9999 & 0.0007 \\ 0.0000 & -0.0007 & 0.9999 \end{pmatrix}$$

$$Y = \begin{pmatrix} 0.9999 & 0.0000 & 0.0005 \\ 0.0000 & 1.0000 & 0.0000 \\ -0.0005 & 0.0000 & 0.9999 \end{pmatrix}$$

$$Z = \begin{pmatrix} 0.9998 & 0.0000 & 0.0005 \\ 0.0000 & 1.0000 & 0.0000 \\ -0.0005 & 0.0000 & 0.9999 \end{pmatrix}$$

$$A = \begin{pmatrix} 1.0814 & -0.2876 & -29.4407 \\ 0.0000 & 1.1030 & -38.1945 \\ 0.0000 & 0.0000 & 1.0000 \end{pmatrix}$$

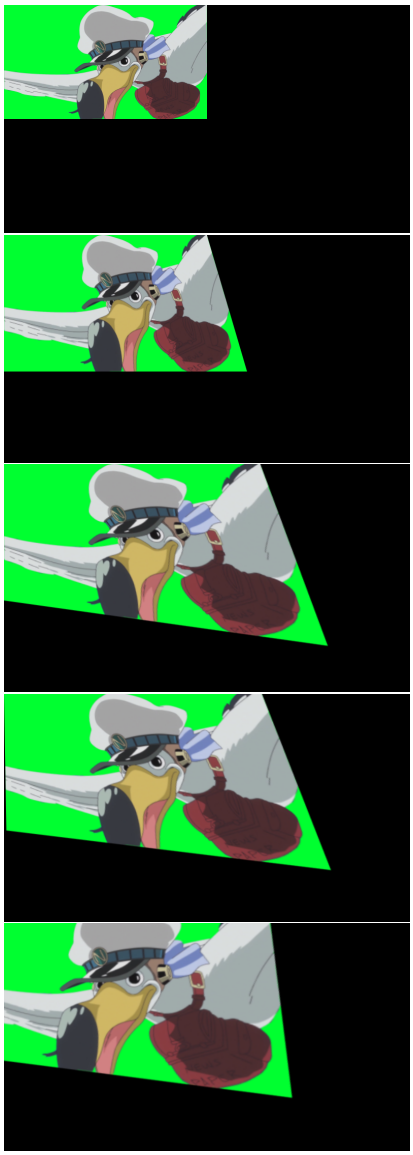

Figura 3.9: *A sequência das matrizes que compõem H, sendo aplicadas uma de cada vez na imagem original. Isso demonstra que a reconstrução da matriz H a partir das variáveis é possível. (Valores aproximados até a 4^a casa decimal.)*

3.7 Interpolação de Matrizes

Define-se n como sendo o número de matrizes de transformação que serão aplicadas a cada *frame* original, sendo que tem-se uma matriz para transformar um certo *frame* f_x em f_{x+1}, e todas as outras $n-1$ matrizes para criar os *frames* intermediários. A partir de agora, define-se como $H_{x,n}$ a matriz que transforma o *frame* f_x em f_{x+1}, e $H_{x,1}$ ... $H_{x,n-1}$ como as matrizes que transformam f_x nos *frames* intermediários $f_{x,1}$ ... $f_{x,n-1}$. Estes são os *frames* que devem ser gerados. Assim, para um dado n, há 2 *frames* já existentes (f_x e f_{x+1}), e $n-1$ *frames* que devem ser gerados.

Original	Interpolado	Original
f_x	$f_{x,1}\ f_{x,2}\ \cdots\ f_{x,n-1}$	f_{x+1}

Original

$$H = \begin{pmatrix} 1.1016 & -0.2469 & -29.4403 \\ 0.0005 & 1.1310 & -38.1937 \\ -0.0005 & -0.0007 & 1.0000 \end{pmatrix}$$

Figura 3.10: *A matriz H aplicada diretamente na imagem, para ser comparada à figura 3.9. (Valores aproximados até a 4ª casa decimal.)*

Passo 5:

Interpolar e aplicar as novas Matrizes de Transformação ao *frame* f_x.

Retomando as váriaveis da matríz $H_{x,n}$: $\theta_{x,n}$, $\theta_{y,n}$, $\theta_{z,n}$, $e_{x,n}$, $e_{y,n}$, $c_{x,n}$, $c_{x,n}$ e s_n. Busca-se um valor nulo (ou seja, que não altere a imagem) para que seja possível iniciar a interpolação.

$\theta_{x,n}$, $\theta_{y,n}$ e $\theta_{z,n}$ são ângulos de rotação. Portanto, o valor inicial destes ângulos $\theta_{x,0}$, $\theta_{y,0}$ e $\theta_{z,0}$ será 0, pois a imagem não deve ser rotacionada.

$e_{x,n}$ e $e_{y,n}$ representam a proporção do escalonamento da imagem. Inicialmente, a imagem deve se manter com o mesmo tamanho, o que equivale a ter seu tamanho multiplicado por 1. Assim, $e_{x,0}$ e $e_{y,0}$ são 1.

$c_{x,n}$ e $c_{y,n}$ representam a translação da imagem. A imagem deve permanecer no mesmo local, portanto, $c_{x,0}$ e $c_{y,0}$ são 0.

Finalmente, s_n é o valor de *shearing*, ou inclinação da imagem. Devido à forma como matrizes de transformação linear são aplicadas, este valor na verdade é igual a $cotg(\theta_s)$, sendo θ_s o ângulo de inclinação desejado. Este ângulo é medido em relação ao eixo x, por tanto, é necessário um ângulo de 90º, para que a figura permaneça erguida. Assim, $s_0 = cotg(90º) = 0$.

Agora as variáveis de $H_{x,0}$ e $H_{x,n}$ são conhecidas. Utilizando-as como valores inicial e final, é possível interpolar linearmente $n - 1$ matrizes intermediárias.

3.8 Resultados

Conforme mencionado anteriormente, será utilizada uma animação simples de uma bola pingando no chão para demonstrar o resultado deste programa. O resultado final está ilustrado abaixo na Figura 3.11, mas para melhor visualização, recomenda-se visitar a página

com a animação[3].

Aplicando cada uma das matrizes $H_{x,1}$ ... H_{n-1} a um *frame* original f_x da bola caindo, obtêm-se bons resultados. É possível notar que o movimento da bola segue uma trajetória adequada, indo sempre em direção ao ponto onde ela está no *frame* seguinte. Além disso, a deformação que a bola sofre ao cair no chão também está de acordo com o esperado, pois ela não está deformada em um quadro, e aos poucos se deforma até chegar ao estado que está no quadro seguinte.

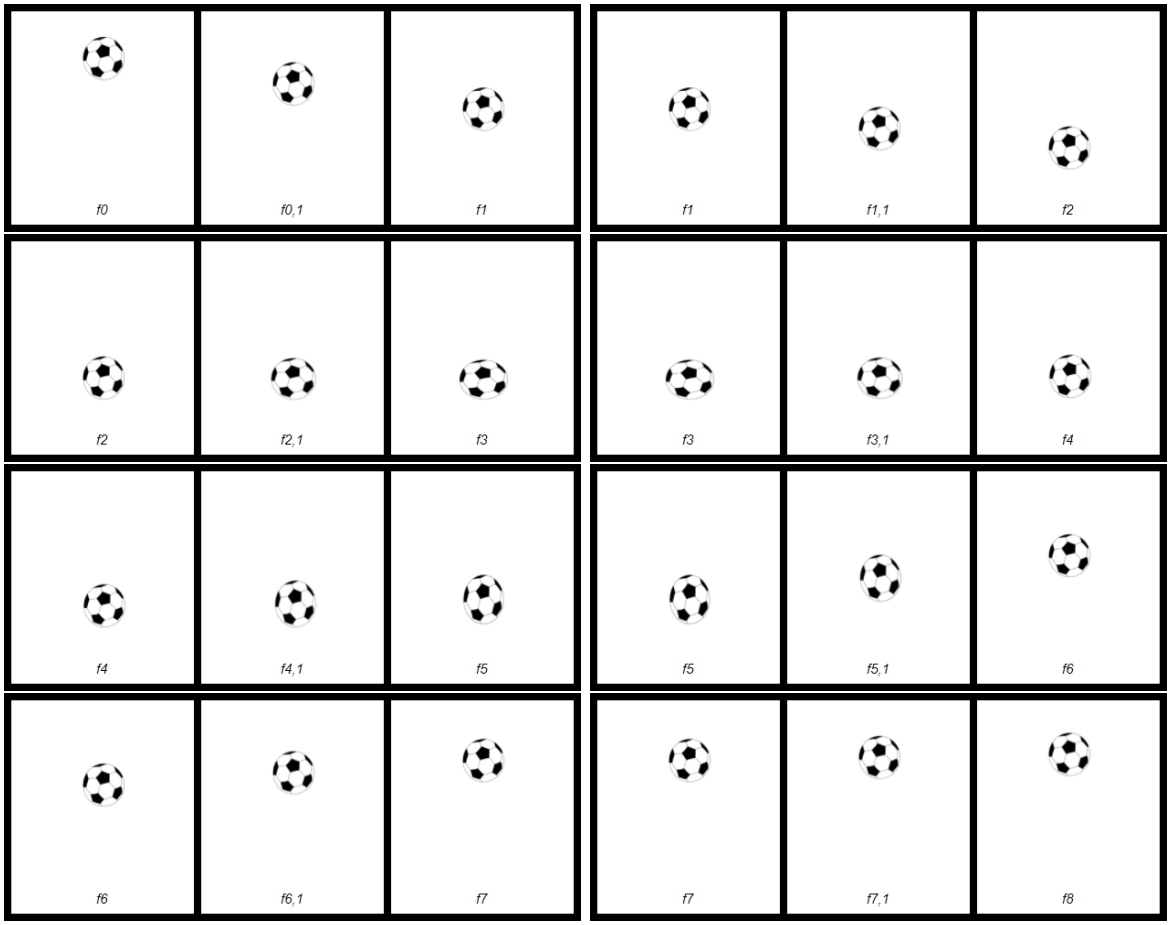

Figura 3.11: *Utilizando $n = 2$, é interpolado 1 frame intermediário entre cada par de frames originais na animação da bola.*

[3] https://linux.ime.usp.br/~giantakara/mac0499/examples/inbetween/page.html (acessado em 27/11/2017)

Capítulo 4

Conclusões

Neste trabalho foram discutidos diversos aspectos da produção de desenhos animados, e como sua popularidade vem influenciando seu futuro, tanto positiva quanto negativamente. Não há expectativas de sua demanda cair, portanto, não somente este trabalho, mas também muitas outras iniciativas estão sendo criadas para que o público consiga a arte de qualidade que tanto admiram, e para que seus criadores possam se dedicar de forma saudável ao seu trabalho.

O uso de *keypoints* através do SIFT funcionou muito bem, mesmo sendo feito em desenhos, algo que foge ao propósito original dessa tecnologia. O mesmo pode ser dito do algoritmo de *stitching*, que originalmente foi feito para unir duas imagens, e que neste trabalho foi utilizado para estimar transformações entre *frames*.

E como foi demonstrado, a combinação dessas tecnologias resultou em bons resultados, capazes de revelar muito sobre aspectos que desenhos têm de diferente em relação a imagens reais. Estes resultados permitem continuar estudos na área de produção de desenhos utilizando Visão Computacional e Processamento de Imagens.

4.1 Trabalhos Futuros

Como pôde ser visto no desenvolvimento do projeto, esta lógica funciona bem quando deseja-se que uma imagem inteira seja transformada linearmente. Porém, há casos em que uma imagem possui diferentes elementos que precisam ser transformados de formas diferentes. Isso foi verificado com os dois exemplos usados neste trabalho: Para bola, que compõe inteiramente a imagem, uma transformação que afeta todos os pontos da mesma forma funciona bem, porém, para o exemplo do pássaro que se aproxima da câmera, seria desejável que alguns elementos se movessem mais do que outros para que os últimos *frames* interpolados ficassem mais similares ao *frame* seguinte original.

Este último caso que não foi resolvido é o mais comum na indústria de desenhos animados. Os artistas possuem a liberdade de criarem seus desenhos da forma como acharem melhor, por isso, uma lógica mais flexível que possa incluir estes casos é necessária para de fato

auxiliar estes artistas.

Uma alternativa que não foi explorada a fundo durante este trabalho é a deformação de imagens utilizando quadrados mínimos [Schaefer *et al.* (2006)]. Com este método, é possível aplicar diferentes transformações em diferentes pontos da imagem, não mais necessitando aplicar uma única transformação à imagem inteira. No artigo, é demonstrado o uso deste método utilizando certos pontos na imagem, visto na Figura 4.1, algo que é possível substituir com os *keypoints* detectados neste trabalho.

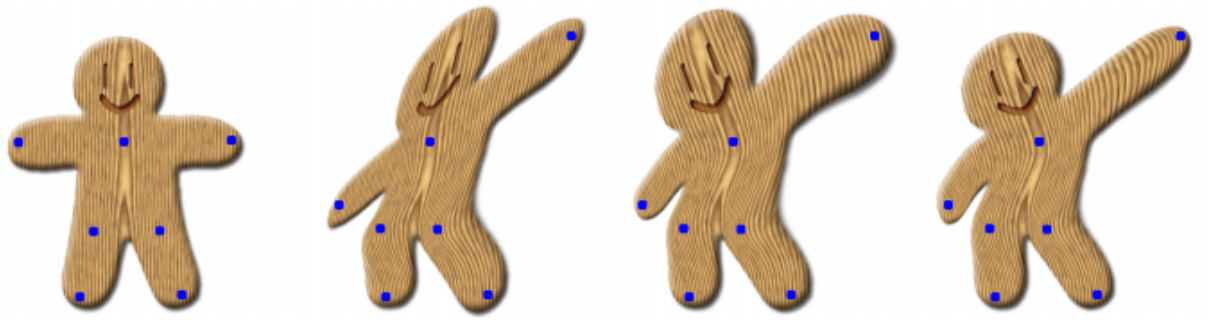

Figura 4.1: *A deformação por quadrados mínimos em ação. É importante notar como cada ponto pode se mover de uma forma diferente dos outros, e que ainda sim é mantida a relação entre eles, mantendo a imagem contínua.*

Outro fator necessário seria a possibilidade de um desenhista que viesse a utilizar este método poder escolher manualmente os pontos na imagem. Poderiam também ser feitas associações entre *keypoints* manualmente, para utilizar a percepção do artista para fazer a correspondência correta, criando resultados que podem ser até melhores do que aqueles obtidos através de algoritmos de correspondência automáticos. Deixando mais espaço para o artista, é possível diminuir erros em casos que necessitam da percepção humana para identificar certos aspectos de animações.

Finalmente, seria interessante que uma funcionalidade como esta estivesse disponível em programas próprios para criação de animações. Após concluído, considera-se submeter este trabalho para desenvolvedores de aplicações deste tipo, como o OpenToonz[1].

[1] https://opentoonz.github.io/e/ (acessado em 20/11/2017)

Referências Bibliográficas

Blair(2017) G. Blair. Hollywood Reporter, Japan's Anime Industry Grows to Record $17.7b, Boosted by 'Your Name' and Exports, 2017. Cited in page 5

Brown e Lowe(2007) M. Brown e D. Lowe. Automatic Panoramic Image Stitching using Invariant Features, 2007. Cited in page 14, 15

Fischler e Bolles(1980) M. Fischler e R. Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography, 1980. Cited in page 15

Garage *et al.***(2000–2017)** W. Garage, Intel Corporation e Itseez. OpenCV: Open Source Computer Vision Library. https://opencv.org/, 2000–2017. Cited in page 8

Harris e Stephens(1988) C. Harris e M. Stephens. A Combined Corner and Edge Detector. Cited in page 9

Krig(2014) S. Krig. Interest point detector and feature descriptor survey, in computer vision metrics, 2014. Cited in page 8

kVin(2017) kVin. Sakugabooru, ANIME FINANCIALS 2016, 2017. Cited in page 5

Lowe(2004) D. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. Cited in page 9

Macdonald e Sevakis(2008) C. Macdonald e J. Sevakis. Inside Toei Animation, 2008. Cited in page 3

Schaefer *et al.***(2006)** S. Schaefer, T. McPhail e J. Warren. Image Deformation Using Moving Least Squares. Cited in page 23

Tuytelaars e Mikolajczyk(2008) T. Tuytelaars e K. Mikolajczyk. Local invariant feature detectors: a survey, 2008. Cited in page 8