Universidade Estadual Júlio de Mesquita Filho
Faculdade de Ciências - Bauru

Victor Santos de Moura Cesario

Módulo de alinhamento rotacional de *blendshapes* direcionado a um pipeline de animação facial

Bauru
2017

Victor Santos de Moura Cesario

Módulo de alinhamento rotacional de *blendshapes* direcionado a um pipeline de animação facial

Monografia apresentada junto à disciplina Projeto e Implementação de Sistemas II, do curso de Bacharelado em Ciência da Computação, Faculdade de Ciências, Unesp, campus de Bauru, como parte do Trabalho de Conclusão de Curso.

Orientador: Prof. Adj. Antonio Carlos Sementille

Bauru
2017

Victor Santos de Moura Cesario

Módulo de alinhamento rotacional de *blendshapes* direcionado a um pipeline de animação facial

Monografia apresentada junto à disciplina Projeto e Implementação de Sistemas II, do curso de Bacharelado em Ciência da Computação, Faculdade de Ciências, Unesp, campus de Bauru, como parte do Trabalho de Conclusão de Curso.

BANCA EXAMINADORA

Prof. Dr. Antonio Carlos Sementille
Orientador
Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Departamento de Computação

Prof. Dr. Joao Fernando Marar
Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Departamento de Computação

Profª. Drª. Simone das Graças Domingues Prado
Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Departamento de Computação

Bauru, 7 de dezembro de 2017.

RESUMO

Recentemente, tem havido um aumento de interesse nas tecnologias de captura de movimentos humanos de corpo inteiro, assim como de expressões faciais. O barateamento de sensores de profundidade viabilizou a pesquisa e o desenvolvimento de diversos sistemas de animação facial, dentre eles os que se baseiam em performances de atores capturadas por câmeras RGB e RGB-D. Estes sistemas orientados a performance são oriundos da união de técnicas avançadas de animação facial e de captura de movimentos. Um método bastante utilizado para demonstrar expressões humanas digitalmente é o dos *blendshapes*, modelos deformáveis que definem um espaço linear de alcance dos movimentos faciais. O movimento rígido da cabeça pode ser interpretado diferentemente dependendo do contexto, tornando-se em alguns casos fator decisivo para determinar a natureza da mensagem transmitida. A identificação de rotação e translação da cabeça é crucial para a interação humana, e caracteriza um problema da captura de movimentos conhecido como estimativa da pose da cabeça. Considerando o cenário descrito, o presente trabalho teve como objetivo principal a estruturação, desenvolvimento e teste de um módulo de alinhamento de *blendshapes*, para ser utilizado sobre a saída de *pipelines* de animação facial baseados em performance.

Palavras-chave: Estimativa da pose da cabeça. Animação facial. Captura de movimentos.

ABSTRACT

In recent years, there has been an increasing interest on full-human body motion capture technology, as well as facial expressions. The decrease in cost of depth sensors have enabled the research and development of many facial animation systems, and among them, those that are based in human performances captured by RGB and RGB-D devices. These performance-driven systems stem from the union of advanced techniques in facial animation and motion capture. A very popular method for creating virtual human expressions is known as blendshapes, morphable models that define a linear space limiting the scope of a model's deformation. A head's rigid motion can be interpreted differently based on the context, making it crucial to determine the nature of a message in some cases. The identification of head rotation and translation is almost mandatory for human interaction, and outlines a problem in motion capture known as head pose estimation. Considering the described scenario, this work had as main objective the structuring, development and testing of a module for blendshape alignment, to be applied over the output of performance-driven facial animation pipelines.

Keywords: Head pose estimation. Facial animation. Motion capture.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

2D	Bidimensional
3D	Tridimensional
API	*Application Programming Interface*
FAPs	*Facial Animation Parameters*
FIP	*Face Identity-Preserving*
MPEG	*Moving Picture Experts Group*
OCR	*Optical Character Recognition*
OpenCV	*Open Source Computer Vision Library*
PCA	*Principal Component Analysis*
RGB	*Red, Green and Blue*
RGB-D	*Red, Green, Blue and Depth*

SUMÁRIO

1. INTRODUÇÃO

O ser humano possui a habilidade de detectar e interpretar a orientação e o movimento de uma cabeça, sem grandes esforços. A partir dessa detecção, é possível inferir as intenções das pessoas ao redor, tornando esta uma importante forma de comunicação.

Em muitos casos, a informação a respeito da orientação da cabeça se torna extremamente útil. Em uma aplicação de realidade virtual, o rastreamento da posição da cabeça pode ser utilizado para adaptar o ângulo de visualização do cenário corretamente (GOOGLE, 1999). Murphy-Chutorian, Doshi e Trivedi (2007) apresentam um sistema de auxílio a motoristas de carro capaz de detectar, através de câmeras internas, a posição da cabeça do motorista e determinar se ele está ou não prestando atenção na rodovia.

Ultimamente, tem-se observado um aumento de atividade no ramo da animação facial, que objetiva a reprodução realista e fiel das expressões humanas. Diversas técnicas são empregadas para acrescentar valor emocional em produtos como filmes para cinema e televisão, videoconferência baseada em avatares, na medicina com cirurgia facial e jogos eletrônicos.

Um fato que é normalmente negligenciado é a importância do movimento rígido da cabeça para a entoação de um sentimento que se deseja transmitir entre um ser humano e outro. O estudo de Gunes e Pantic (2010) apresenta uma forte correlação entre o movimento da cabeça e a percepção das mensagens e emoções. De acordo com o trabalho, estes movimentos marcam a estrutura do discurso sendo realizado, e são utilizados como referência para regular as interações sociais.

A estimativa da pose da cabeça (*head pose estimation*) é o nome dado à habilidade de inferir a orientação de uma cabeça humana em relação ao plano da câmera, ou mais precisamente, em relação a um sistema de coordenadas global. A literatura contém, atualmente, uma quantidade considerável de trabalhos sobre estimativa da pose da cabeça, que podem ser divididos grosseiramente em métodos que usam imagens 2D ou informação de profundidade (FANELLI et al., 2011). Murphy-Chutorian e Trivedi (2009) sugerem uma subdivisão ainda mais minuciosa em oito categorias, constando entre estas os métodos de rastreamento, de encaixe

múltiplo (*manifold embedding*) e os métodos híbridos com o maior número de publicações.

Os sistemas de animação facial baseados em performance representam a união entre o estado da arte dos métodos de animação digital do rosto humano e de captura de movimentos da cabeça. A composição de um sistema deste porte pode se utilizar do rastreamento de marcadores faciais, *blendshapes* e do algoritmo de minimização ordinal PCA (*Principal Component Analysis*), como no *pipeline* desenvolvido por Silva (2017). Este *pipeline* possui como saída um conjunto de *blendshapes* que representa as expressões faciais rastreadas. A adição de um módulo de alinhamento do *blendshape* resultante à captura, proposta deste trabalho, agrega ao *pipeline* uma forma mais apropriada de verificação da acurácia, melhorando a qualidade do sistema como um todo.

1.1 Motivação

Um grande problema da animação facial é a reprodução fidedigna e de alta qualidade das feições capturadas de um ator. As expressões realizadas por este ator podem atingir uma vasta ordem de intensidades, variando das mais acentuadas, como um largo sorriso aberto, às mais sutis, como um rápido piscar de olhos.

Sendo assim, é de interesse do criador de um sistema de animação o domínio das melhores formas de se contrapor o resultado gerado pelo sistema à captura do mundo real. Uma das formas mais primitivas de verificação de resultado é a comparação visual. Este trabalho focou, portanto, na acrescentação de um módulo para ser aplicado após um *pipeline* de animação facial sem funcionalidade de estimativa de pose de cabeça, similar ao de Silva (2017).

A melhoria de procedimentos simples como este possuem grande impacto na qualidade final das produções. Os *pipelines* devem estar constantemente buscando atingir o mais alto nível de eficiência, com algoritmos de visão computacional e deformação de malhas, a fim de se tornarem versáteis. Melhores *softwares* levam a redução do custo de operação, e consequentemente abaixam a barra de entrada para novos criadores de conteúdo. Por fim, pode-se aumentar a qualidade das produções de mídia, e assim levar a arte ao próximo patamar.

1.2 Objetivos

O objetivo principal deste trabalho foi complementar o *pipeline* de animação facial baseado em performance implementado por Silva (2017), por meio do desenvolvimento de um módulo com a finalidade de justaposição, com coerência de rotação, dos *blendshapes* gerados pelo *pipeline*, com relação aos *frames* do fluxo de vídeo provenientes da captura da face de um ator.

Os objetivos específicos deste trabalho foram:

- Estruturar um módulo para alinhamento de *blendshapes*, a fim de ser utilizado em um sistema de animação facial realista direcionado a performance;
- Implementar e testar este módulo, verificando a acurácia das rotações geradas por meio da comparação entre os quadros de frames do fluxo de captura e um modelo 3D da cabeça humana, simulando a saída do *pipeline* de Silva (2017).

1.3 Organização do trabalho

Esta monografia foi organizada da maneira que segue.

No capítulo 2, se encontra a fundamentação teórica necessária para o entendimento e a execução deste trabalho, abordando conceitos como visão computacional, captura de movimentos e estimativa de poses.

O capítulo 3 apresenta as etapas do desenvolvimento do projeto, elaboradas e implementadas para alcançar os objetivos.

O capítulo 4 apresenta uma análise sobre os resultados do *pipeline*, além de avaliação de sua capacidade por meio de experimentos.

Por fim, no capítulo 5, têm-se as considerações finais sobre o projeto e os trabalhos futuros.

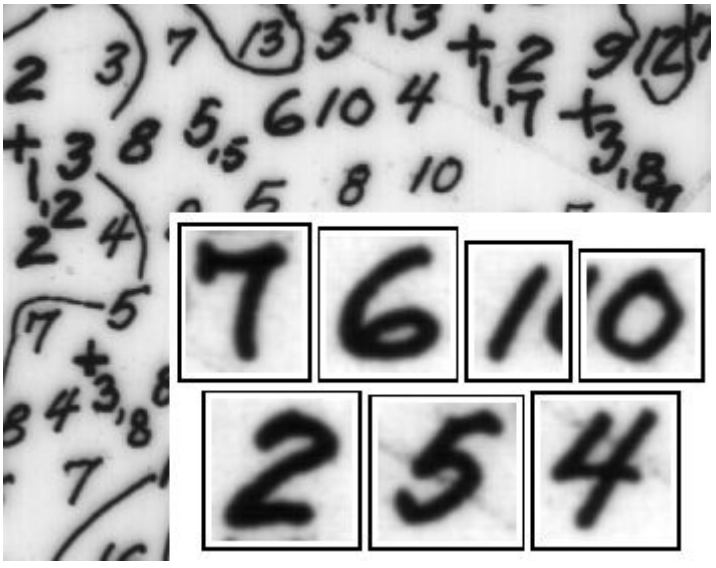
2. FUNDAMENTAÇÃO TEÓRICA

Um aspecto importante da animação facial que não é captado na detecção de expressões faciais é o movimento rígido da cabeça como um todo, elemento igualmente relevante para indicar a emoção que se deseja passar. Portanto, neste capítulo encontra-se uma breve introdução ao conceito de animação facial, além de explicações a respeito de alguns termos utilizados na visão computacional e computação gráfica, que serão utilizados ao longo deste trabalho.

2.1 Visão computacional

Visão computacional consiste na utilização de técnicas matemáticas para reconstruir em um ambiente virtual as propriedades do mundo real, como forma, iluminação e distribuições de cor (SZELISKI, 2010). É uma tecnologia bastante abrangente, possuindo serventia em diversas áreas. OCR (*Optical Character Recognition*), por exemplo, é um termo referente aos métodos de reconhecimento ótico de caracteres, capazes de extrair informações textuais contidas em imagens de textos do mundo real, como no caso da Figura 1.

Figura 1. Reconhecimento ótico de caracteres sendo aplicado



Fonte: Trier, Jain e Taxt (1996).

Autonomia robótica e automotiva também têm ganhado bastante atenção recentemente devido ao aumento da eficácia dos algoritmos de detecção baseados em

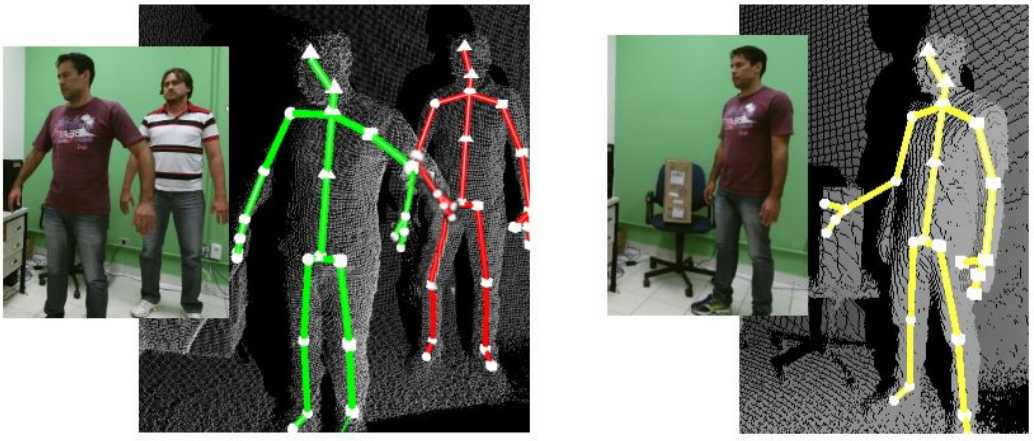
visão, permitindo que veículos possam detectar obstáculos inesperados como pedestres, sob condições onde radares e lidares não funcionam corretamente (SZELISKI, 2010).

A tecnologia de mocap (*motion capture*) pode ser definida como a captura de movimentos corporais - rotação de cabeça, braços, torso e pernas - em larga escala. No trabalho de Moeslund e Granum (2001), uma estrutura é inferida para as aplicações de mocap:

- **Inicialização.** Certificação de que o sistema inicia com uma interpretação correta da cena visível.
- **Rastreamento.** Segmentação e rastreamento dos seres humanos em um ou mais frames.
- **Estimativa de pose.** Presumir e quantificar a pose de um ser humano capturado em um ou mais frames.
- **Reconhecimento.** Reconhecimento de indivíduos e suas ações, atividades e comportamentos performados durante a captura.

Dentro desta estrutura e posteriores subdivisões, centenas de trabalhos a respeito do mocap foram desenvolvidos, com o intuito de explorar as extensões e utilidades da tecnologia. Dentre eles, pode-se destacar o trabalho de Motta (2016), que visou a reconstrução de um sistema offline de captura de movimentos a partir de um sensor RGB-D, visível na Figura 2.

Figura 2. Sistema de captura de movimentos offline



Fonte: Motta (2016).

2.2 Animação facial

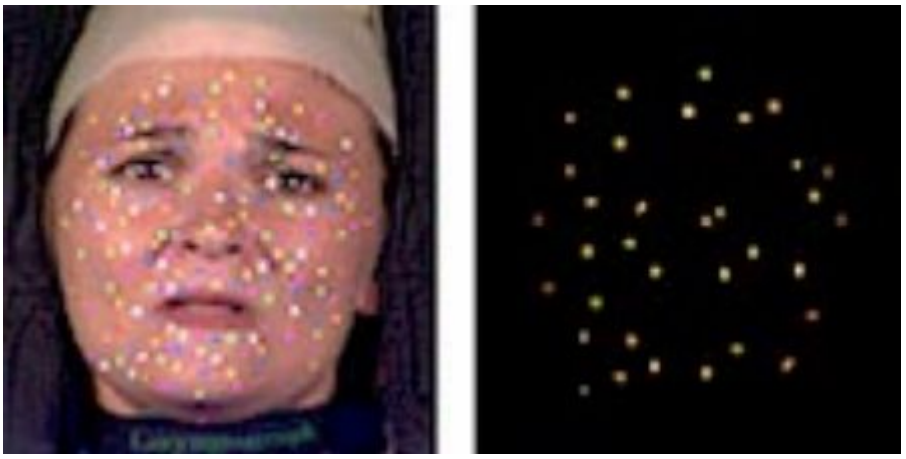
Dentro do campo da mocap, atualmente, técnicas de animação facial estão sendo utilizadas em larga escala para diversas finalidades. Existe um número crescente de aplicações que utilizam expressões faciais digitalizadas para transmitir as emoções humanas em um alto nível de fidelidade. Seu uso pode ser observado na biologia, acerca do controle de faces digitalizadas para gerar estímulos psicofisiológicos (NAPLES et al., 2014). O controle de faces digitais manualmente também pode ser visto na área da robótica (WITTIG; KLOOS; RÄTCH, 2015), e amplamente na indústria do entretenimento, com filmes e jogos eletrônicos em 3D.

Segundo Deng e Noh (2008), um sistema de captura de animação facial ideal deve ser capaz de criar animações realistas, operar em tempo real, ser automatizado e adaptar-se facilmente a qualquer face humana . São diversos os métodos de animação facial que podem ser empregados para atingir um resultado de alta qualidade. A técnica de animação baseada em física utiliza simulações de componentes reais do corpo humano, como pele, gordura, ossos e músculo (SIFAKIS; NEVEROV; FEDKIW, 2005). A Figura 3 contém um exemplo remoto de animação baseada em física. Guenter et al. (1998) utiliza um grande conjunto de pontos no rosto para rastrear precisamente as deformações faciais, exibido na Figura 4. Ao mesmo tempo, são capturados vídeos da face em alta resolução para gerar o mapa de textura que é renderizado em conjunto.

Figura 3. Animação baseada em física

Fonte: Terzopoulos e Waters (1990).

Figura 4. Animação facial com rastreamento de pontos



Fonte: Guenter et al. (1998).

Sistemas que dependem da inserção de marcadores faciais manualmente no rosto do ator, como o de Guenter et al. (1998) e Luo et al. (2014) são capazes de reproduzir as feições humanas com alta fidelidade e rápida performance. Em contrapartida, as etapas de aquisição para este tipo de *pipeline* acabam sendo mais complexas e manuais, além de possuírem uma natureza invasiva.

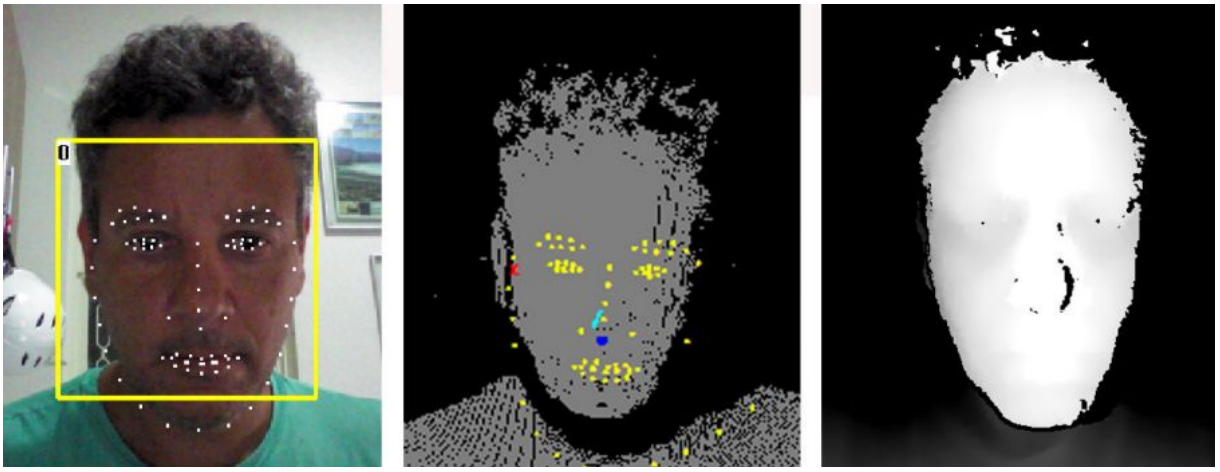
Uma técnica que se tornou bastante popular recentemente é a dos *blendshapes*. Originários da indústria, os *blendshapes* eventualmente se tornaram uma ferramenta de interesse acadêmico, devido a uma combinação de simplicidade, expressividade e interpretabilidade (LEWIS et al., 2014). Esta abordagem foi utilizada em obras cinematográficas bem-sucedidas como O Senhor dos Anéis, King Kong e O Curioso Caso de Benjamin Button. Silva (2017) propõe um método de animação facial baseada em performance através da captura de informações RGB e de profundidade. Uma das partes do seu *pipeline* pode ser observada na Figura 5.

2.3 Marcadores faciais

O MPEG (Moving Picture Experts Group) é um grupo de trabalho responsável pelo desenvolvimento de padrões internacionais para compressão, descompressão, processamento e representação codificada de vídeo, áudio e sua combinação. De acordo com Braun (2014), o padrão MPEG-4, criado pelo grupo MPEG, é utilizado na animação facial, e define um conjunto de 84 pontos faciais, que estão demonstrados

na Figura 6. Um subconjunto destes pontos serve de controle para 68 parâmetros de animação (FAPs), que também são definidos pelo padrão.

Figura 5. Detecção facial utilizando câmera RGB-D



Fonte: Silva (2017).

Figura 6. Conjunto de 84 pontos faciais em diversas imagens de rosto

Fonte: Zhang et al. (2014).

A detecção de marcadores faciais é um componente fundamental em diversas tarefas de análise facial. O trabalho de Zhu et al. (2014) demonstra que é possível reconstruir, a partir de fotos de cabeças em semi-perfil, a face frontal da pessoa

capturada. O método é conhecido como espaço facial com preservação de identidade e utiliza pontos-chave do rosto, denominados atributos FIP (*face-identity preserving*), em conjunto com uma rede neural para codificar uma imagem facial neutra a estes pontos. O resultado pode ser observado na Figura 7.

Figura 7. Resultado do método do espaço facial com preservação de identidade

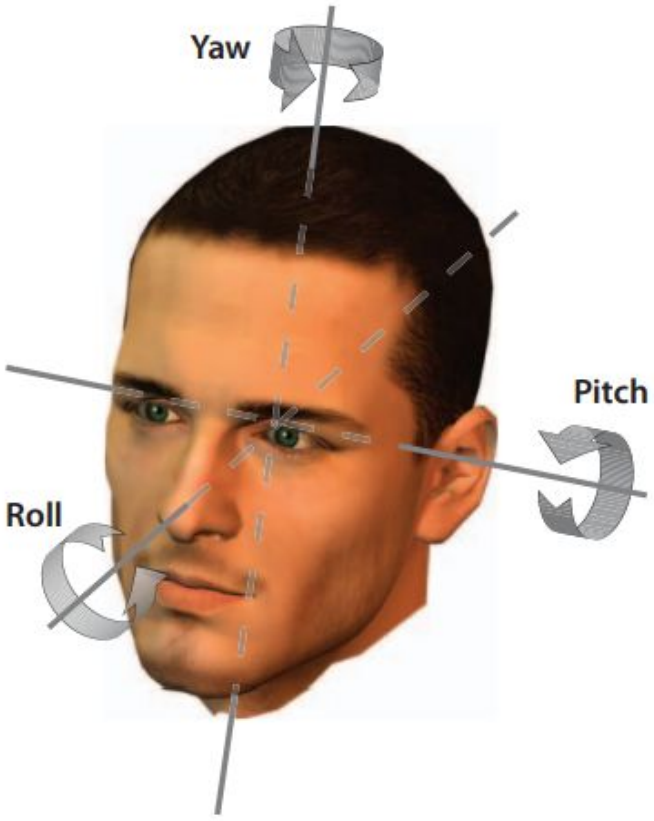
Fonte: Zhu et al. (2014).

2.4 Estimativa da pose da cabeça

Murphy-Chutorian e Trivedi (2009) definem a estimativa da pose da cabeça como sendo a habilidade de inferir a orientação de uma cabeça em relação a um sistema de coordenadas global, dado que se conheça os parâmetros intrínsecos da câmera como distância focal e coeficiente de distorção. Dentro deste cenário, a cabeça dispõe de três graus de liberdade para descrever sua pose. Estes graus são, como visto na Figura 8, conhecidos como *pitch*, *roll* e *yaw* (eixos lateral, longitudinal e vertical, respectivamente).

Os métodos para rastrear as coordenadas da cabeça podem ser baseados apenas em imagens 2D, ou contarem com informações 3D de profundidade. Os métodos 2D, apesar de apresentarem problemas, principalmente em relação a mudanças de iluminação e regiões faciais desprovidas de textura (FANELLI; GALL; GOOL, 2011), possuem o potencial de serem empregados mais facilmente, pois dispositivos RGB-D ainda não são tão difundidos quanto os dispositivos RGB, presentes em milhões de aparelhos de consumidor na atualidade.

Figura 8. Visualização dos eixos de *pitch, roll* e *yaw*



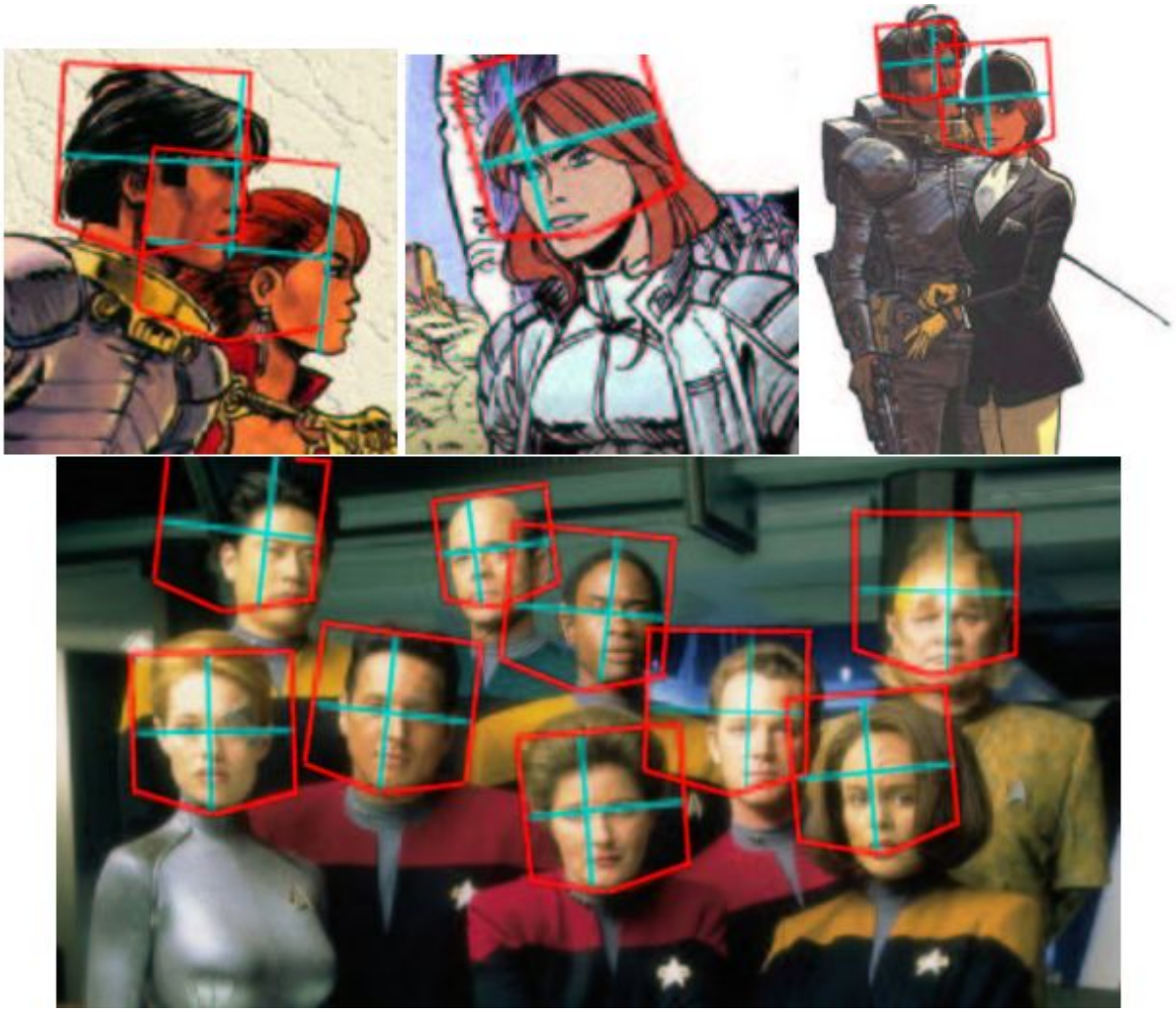
Fonte: Murphy-Chutorian e Trivedi (2009).

Fanelli, Gall e Gool (2011) classificam os métodos sobre imagens 2D em baseados em atributos (feature-based) e baseados em aparência (appearance-based).

Os métodos baseados em atributos exigem que o mesmo conjunto de atributos faciais esteja presente em todas as poses. Yang e Zhang (2002) exigem a entrada manual da posição dos pontos faciais para inicializar o sistema de rastreamento. Após esta inicialização, no entanto, o sistema elege outros pontos característicos para integrar o conjunto já determinado e aumentar a robustez da detecção.

Métodos baseados em aparência normalmente possuem um detector distinto para cada pose possível de cabeça (JONES; VIOLA, 2003). No entanto, King (2016) atesta que esta abordagem não é uma solução generalizada, além de aumentar desnecessariamente o tamanho da base. A utilização de redes neurais convolucionais, como proposta por Osadchy, Cun e Miller (2007), se mostra uma alternativa mais potente, capaz de detectar e estimar uma rotação aproximada de até mesmo rostos de perfil em páginas de quadrinhos, como mostra a Figura 9.

Figura 9. Resultado de estimativa da pose da cabeça utilizando redes convolucionais

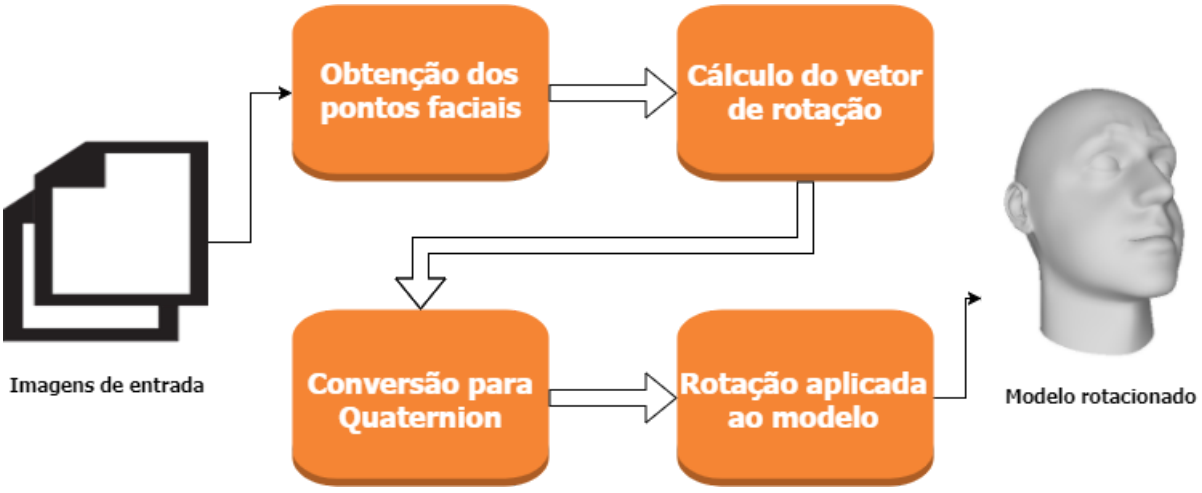


Fonte: Osadchy, Cun e Miller (2007).

3. DESENVOLVIMENTO

O desenvolvimento do trabalho foi realizado considerando o *pipeline* esquematizado na Figura 10. Cada uma das etapas está descrita nas seções a seguir, precedidas pela explicação da configuração do ambiente de desenvolvimento.

Figura 10. *Pipeline* elaborado para o projeto



Fonte: elaborada pelo autor.

3.1 Configuração do ambiente de desenvolvimento

O ambiente de desenvolvimento foi concebido especificamente para este projeto. Portanto, para realizar as capturas exigidas para o seu desenvolvimento, foram necessárias instalações de algumas bibliotecas públicas que serviram para executar procedimentos que não estão inclusos na proposta do projeto.

Além disso, houve um estudo para a criação de um plugin nativo, um tipo especial de biblioteca, para o *software* Unity 3D. Este plugin é o responsável por suportar a ponte entre as bibliotecas públicas instaladas, que estão na linguagem C++, e a API (*Application Programming Interface*) interna do Unity 3D, disponível na linguagem C#. As bibliotecas públicas utilizadas foram o Dlib e o OpenCV.

3.1.1 Dlib

A biblioteca Dlib consiste em um conjunto de ferramentas e algoritmos de aprendizado de máquina implementado em C++, e permite a criação de projetos de

software de alta complexidade em duas linguagens populares: C++ e Python. Devido ao fato de ser distribuída sobre a licença de código aberto, a biblioteca é largamente utilizada pela indústria e academia para resolver problemas de diversos domínios, incluindo robótica, sistemas embarcados, dispositivos móveis e ambientes de computação de alta performance.

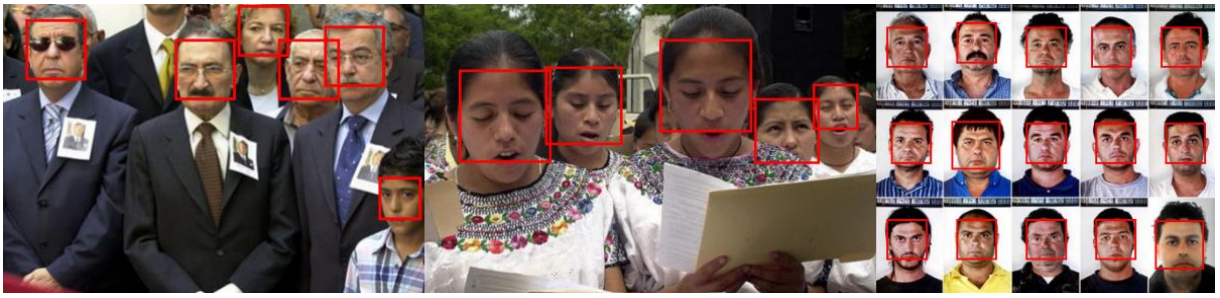
A biblioteca foi inicialmente concebida em 2002 por Davis King, e hoje em dia contém componentes que lidam com redes, processos concorrentes, interfaces gráficas, álgebra linear, processamento de imagens e outras tarefas.

De acordo com King (2009, tradução nossa),

> [...] Para possibilitar a fácil utilização destas ferramentas, a biblioteca foi desenvolvida com a metodologia de "programação por contrato", o que viabilizou uma documentação completa e precisa, além de ferramentas de depuração altamente capacitadas.
> [...] Isto significa que a biblioteca é, antes de tudo, um conjunto de componentes de *software* independentes, cada um acompanhado de extensa documentação e modos de depuração. Além disso, pretende-se que o conjunto seja útil tanto na área de pesquisa quanto na área comercial, e foi cuidadosamente projetado para facilitar a integração a uma aplicação em C++.

Desta biblioteca, foram utilizadas as funcionalidades de detecção de rostos e localização de seus pontos faciais. Para a detecção de rostos, o Dlib realiza uma combinação dos algoritmos de detecção de objetos HOG (Histogram of Oriented Gradients) de Dalal e Triggs (2005), e o MMOD (Max-Margin Object Detection), proposto pelo próprio King em seu artigo homônimo de 2015. Esta combinação permitiu a detecção de objetos em poses complexas e parcialmente oclusas, como exemplificado na Figura 11.

Figura 11. Resultado do componente de detecção de rostos do Dlib

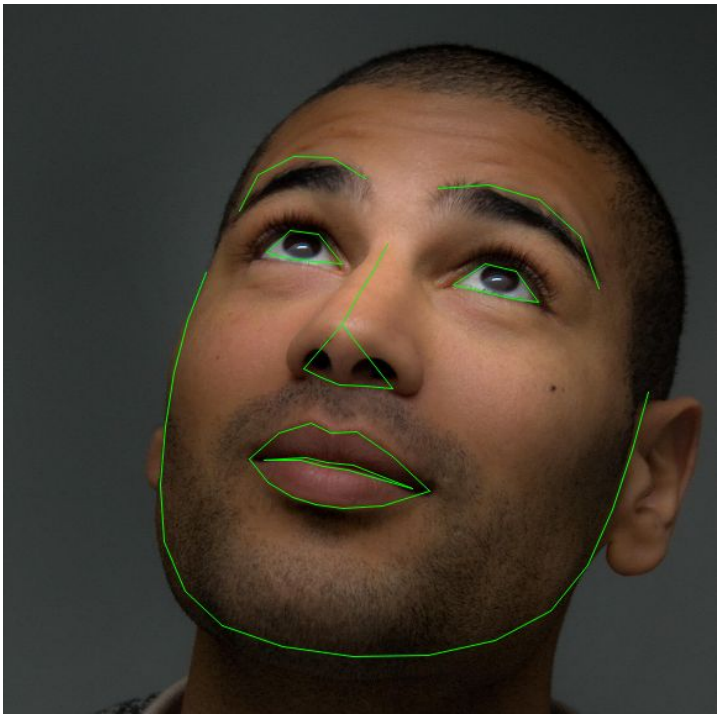


Fonte: King (2015).

Para a localização de pontos faciais dentro dos rostos detectados, a biblioteca possui uma implementação de um algoritmo de árvores de regressão, proposto por Kazemi e Sullivan (2014).

Como consequência destas tecnologias, a utilização do Dlib permite que se obtenha, a partir de uma foto de um rosto humano em um ângulo arbitrário e uma base de predição, a posição dos 68 pontos faciais humanos em tempo real e alta confiabilidade. Um exemplo de resultado do processo de localização pode ser observado na Figura 12.

Figura 12. Resultado da localização de pontos faciais do Dlib



Fonte: King (2014).

3.1.2 OpenCV

O OpenCV é uma biblioteca de visão computacional e aprendizado de máquina que possui licença de código aberto, assim como o Dlib. A biblioteca fornece acesso a mais de 2500 algoritmos de visão computacional e aprendizado de máquina, clássicos e de estado da arte. Com estes algoritmos, é possível detectar e reconhecer faces, rastrear objetos dinâmicos, extrair modelos tridimensionais de objetos, produzir nuvens de pontos tridimensionais a partir de câmeras estéreo, dentre outras aplicações. Dentre seus usuários, estão grandes empresas de tecnologia como Google,

Microsoft, Intel e IBM. Apesar de ser desenvolvida em C++, a biblioteca possui interfaces programáveis em C++, C, Python, Java e Matlab e executa nos 3 sistemas operacionais principais (Windows, Linux e MacOS), além do sistema mobile Android.

A biblioteca OpenCV se tornou essencial para este projeto por possuir implementações de algoritmos para solução de problemas PNP (Perspective-n-Point). Dentre os diversos algoritmos disponíveis, foi utilizado algoritmo de otimização Levenberg-Marquardt (ROWEIS, 2005) devido à característica dos dados de entrada. Utilizando esta funcionalidade, é possível obter um resultado como o visto na Figura 13.

Figura 13. Resultado da estimativa de pose de cabeça com Dlib e OpenCV

Fonte: Mallick (2016).

3.1.3 Plugins nativos para Unity 3D

Os plugins nativos para Unity 3D são DLLs que fornecem acesso a soluções externas àquelas presentes na API do *software*. De acordo com Zucconi (2015), são *softwares* compilados, similares a aplicações independentes. No entanto, não podem ser executadas diretamente pois são feitas para serem utilizadas por outras aplicações.

O Unity suporta dois tipos de plugins:

- **Plugins gerenciados.** São bibliotecas escritas em C#, e compiladas em CIL (Common Intermediate Language). Apesar de compartilhar da mesma linguagem utilizada internamente pelo Unity, a utilização de um plugin

gerenciado tem a vantagem de ser pré-compilado, economizando o tempo de recompilação do editor.

- **Plugins não-gerenciados.** São bibliotecas escritas em outra linguagem (na maioria das vezes, C++). Permitem acesso a bibliotecas públicas que não possuem implementação em C# de fácil acesso, como Dlib e OpenCV, no caso deste projeto.

3.1.4 Compilação

O projeto contou com duas etapas de compilação. Na primeira etapa, foi utilizado o CMake, ferramenta para a geração automatizada do workspace escolhido através de arquivos de configuração, para gerar os arquivos de extensão .dll e .lib da biblioteca Dlib. A biblioteca OpenCV fornece uma versão já compilada para download, e foi a versão utilizada neste projeto.

A segunda etapa consistiu na utilização do ambiente Microsoft Visual Studio 2015 para compilar o código escrito em C++ para a DLL. O código contém duas funções, *RastrearPontosFaciais()* apresentada na Figura 14, e *CalcularRotacao()* apresentada na Figura 15, embrulhadas em um cabeçalho estilo C, conforme diretiva *extern "C"* presente nos Apêndices A e B.

Após estas duas etapas, o plugin foi finalizado e estava pronto para ser utilizado por scripts do Unity.

Figura 14. Função *RastrearPontosFaciais()* acessando a biblioteca Dlib

```
bool RastrearPontosFaciais(char* dir_base_predicao, char* dir_img, int total_pontos,
    int * id_pontos, double** array_landmarks) {
    frontal_face_detector detector = get_frontal_face_detector();
    shape_predictor sp;
    deserialize(dir_base_predicao) >> sp;
    array2d<rgb_pixel> img;
    load_image(img, dir_img);
    std::vector<dlib::rectangle> dets = detector(img);
    if (dets.size() > 1)
        return false;
    full_object_detection shape = sp(img, dets[0]);

    double w = img.nc();
    double h = img.nr();

    memcpy(&(*array_landmarks)[0], &w, sizeof(double));
    memcpy(&(*array_landmarks)[1], &h, sizeof(double));

    for (int i = 0; i < total_pontos; i++)
    {
        double x = (double)(shape.part(id_pontos[i]).x());
        double y = (double)(shape.part(id_pontos[i]).y());
        memcpy(&(*array_landmarks)[2 * (i + 1)], &x, sizeof(double));
        memcpy(&(*array_landmarks)[2 * (i + 1) + 1], &y, sizeof(double));
    }

    return true;
}
```

Fonte: elaborada pelo autor.

Figura 15. Função *CalcularRotacao()*

```
bool CalcularRotacao(char* dir_img, int* coords, double** array_rotacao) {
    cv::Mat im = cv::imread(dir_img);

    if (im.data == NULL)
        return false;

    std::vector<cv::Point2d> pontos_img_2d;
    pontos_img_2d.push_back(cv::Point2d(coords[0], coords[1]));     // Nariz
    pontos_img_2d.push_back(cv::Point2d(coords[2], coords[3]));     // Queixo
    pontos_img_2d.push_back(cv::Point2d(coords[4], coords[5]));     // Olho esquerdo
    pontos_img_2d.push_back(cv::Point2d(coords[6], coords[7]));     // Olho direito
    pontos_img_2d.push_back(cv::Point2d(coords[8], coords[9]));     // Boca (esquerdo)
    pontos_img_2d.push_back(cv::Point2d(coords[10], coords[11]));   // Boca (direito)

    std::vector<cv::Point3d> pontos_modelo_3d;
    pontos_modelo_3d.push_back(cv::Point3d(0.0f, 0.0f, 0.0f));          // Nariz
    pontos_modelo_3d.push_back(cv::Point3d(0.0f, -330.0f, -65.0f));     // Queixo
    pontos_modelo_3d.push_back(cv::Point3d(-225.0f, 170.0f, -135.0f));  // Olho esquerdo
    pontos_modelo_3d.push_back(cv::Point3d(225.0f, 170.0f, -135.0f));   // Olho direito
    pontos_modelo_3d.push_back(cv::Point3d(-150.0f, -150.0f, -125.0f)); // Boca (esquerdo)
    pontos_modelo_3d.push_back(cv::Point3d(150.0f, -150.0f, -125.0f));  // Boca (direito)
    double dist_focal = im.cols;
    Point2d centro = cv::Point2d(im.cols / 2, im.rows / 2);
    cv::Mat matriz_camera = (cv::Mat_<double>(3, 3) << dist_focal, 0, centro.x, 0, dist_focal, centro.y, 0, 0, 1);
    cv::Mat coef_dist = cv::Mat::zeros(4, 1, cv::DataType<double>::type);
    cv::Mat vetor_rotacao;
    cv::Mat vetor_translacao;
    cv::Mat rotacao3x3;

    cv::solvePnP(pontos_modelo_3d, pontos_img_2d, matriz_camera, coef_dist, vetor_rotacao, vetor_translacao);
    cv::Rodrigues(vetor_rotacao, rotacao3x3);

    Vec3d axis{ 0, 0, -1 };
    cv::Mat direcao = rotacao3x3 * cv::Mat(axis, false);

    double dirX = direcao.at<double>(0);
    double dirY = -direcao.at<double>(1);
    double dirZ = direcao.at<double>(2);
    double len = sqrt(dirX*dirX + dirY*dirY + dirZ*dirZ);
    dirX /= len;
    dirY /= len;
    dirZ /= len;

    memcpy(&(*array_rotacao)[0], &(dirX), sizeof(double));
    memcpy(&(*array_rotacao)[1], &(dirY), sizeof(double));
    memcpy(&(*array_rotacao)[2], &(dirZ), sizeof(double));
    memcpy(&(*array_rotacao)[3], &(len), sizeof(double));

    return true;
}
```

Fonte: elaborada pelo autor.

3.2 Execução do *pipeline*

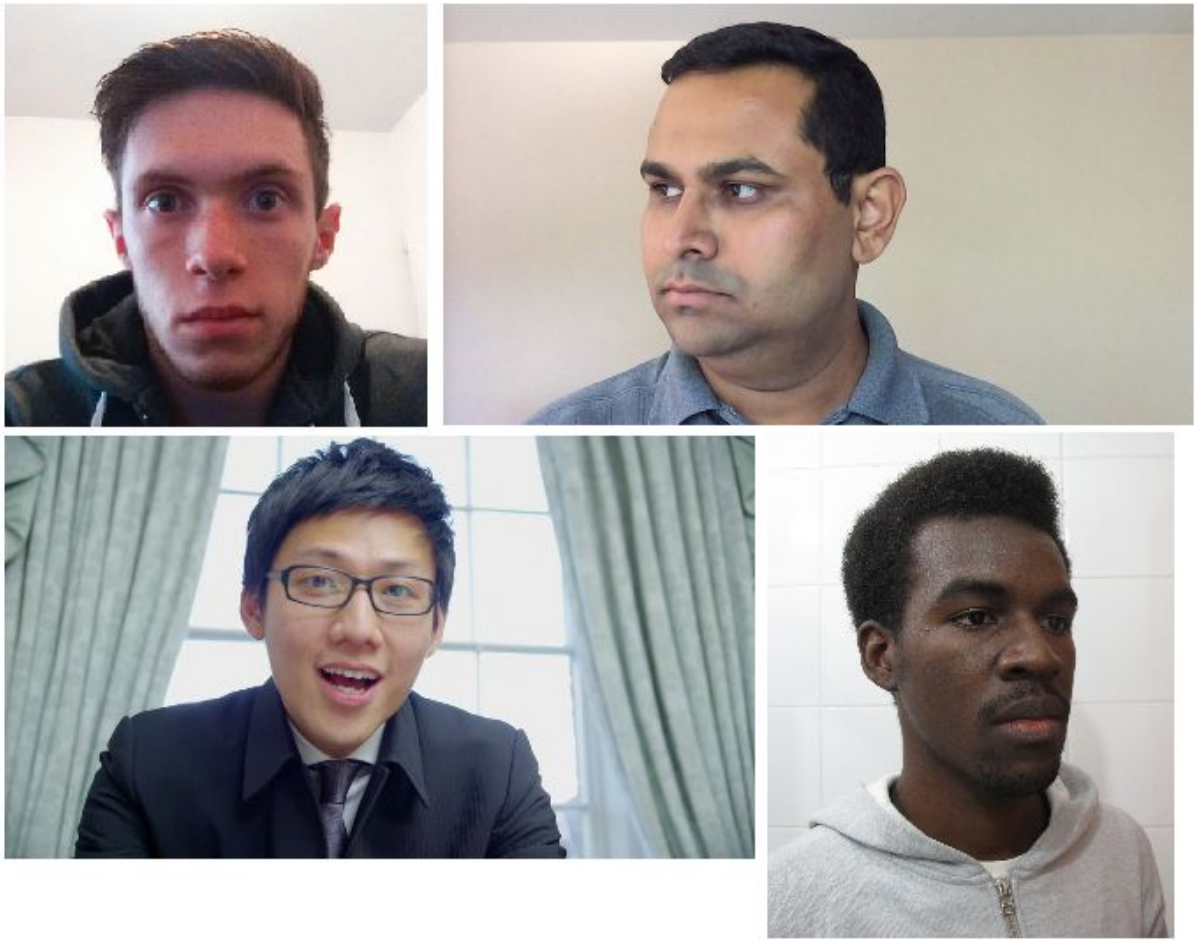
Nesta seção, estão explicadas as etapas do *pipeline* utilizado para obter o movimento de rotação a partir de imagens de entrada.

3.2.1 Imagens de entrada

As imagens de entrada consistem nas fotos de referência que são utilizadas para controlar a rotação do modelo tridimensional. Portanto, foram consideradas apenas fotos contendo um rosto humano (ou seja, dois olhos, um nariz, uma boca e

um queixo aparecendo). Alguns exemplos de imagens adequadas podem ser conferidos na Figura 16.

Figura 16. Exemplo de imagens de entrada para o *pipeline*

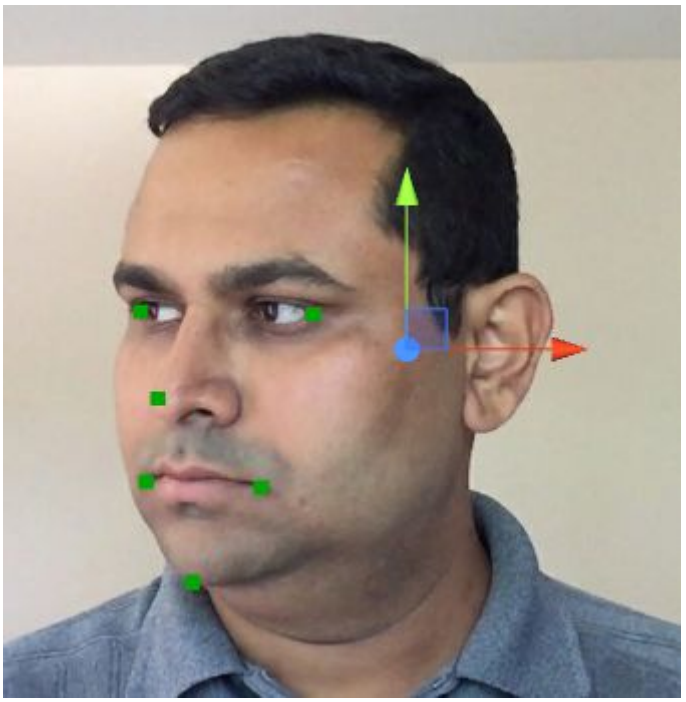


Fonte: elaborada pelo autor.

3.2.2 Obtenção dos pontos faciais

As imagens de entrada são, em seguida, passadas como parâmetro para a função *RastrearPontosFaciais()*, do plugin nativo, junto com a base de predição e os pontos faciais desejados. Segundo Mallick (2016), são necessários apenas 6 dos 68 pontos faciais para determinar a pose do rosto. São estes: a ponta do nariz, o queixo, o canto externo do olho esquerdo, o canto externo do olho direito e os cantos esquerdo e direito da boca. A função então acessa o Dlib e retorna as coordenadas em pixel dos pontos exigidos, conforme exemplificado na Figura 17. Também é importante ressaltar que o sistema de coordenadas do Dlib e do OpenCV reconhecem a origem (o ponto [0,0]) como sendo o pixel superior esquerdo.

Figura 17. Imagem de entrada com pontos localizados pelo Dlib



Fonte: elaborada pelo autor.

3.2.3 Cálculo do vetor de rotação

Para calcular o vetor de rotação do rosto, a função *CalcularRotacao()* do plugin nativo recebe como parâmetros de entrada a imagem e as coordenadas dos pontos faciais. A implementação de solução do PnP presente na biblioteca OpenCV também requer como parâmetro algumas características intrínsecas da câmera (distância focal e coeficiente de distorção), além de também exigir a localização de pontos faciais em um modelo 3D do rosto capturado na imagem. No entanto, como sugerido por Mallick (2016), estes parâmetros podem ser aproximados da seguinte maneira:

- **Distância focal da câmera.** Esta medida pode ser aproximada como sendo o centro da imagem de entrada fornecida.
- **Coeficiente de distorção.** Esta medida indica a distorção radial da câmera conforme o pixel se aproxima das bordas. Para o caso desta aplicação, a distorção foi considerada nula.
- **Pontos faciais de um modelo 3D.** Para que o algoritmo PnP realize uma projeção correta dos pontos 3D em relação aos pontos 2D fornecidos, é

necessário que estes representem um modelo de topologia equivalente à do usuário capturado. Porém, um modelo 3D genérico pode, segundo Mallick (2016), resultar em uma rotação aproximada satisfatória. Foram considerados os pontos mostrados na Figura 18.

Figura 18. Pontos faciais em um modelo 3D genérico

```
std::vector<cv::Point3d> pontos_modelo_3d;
pontos_modelo_3d.push_back(cv::Point3d(0.0f, 0.0f, 0.0f));          // Nariz
pontos_modelo_3d.push_back(cv::Point3d(0.0f, -330.0f, -65.0f));     // Queixo
pontos_modelo_3d.push_back(cv::Point3d(-225.0f, 170.0f, -135.0f)); // Olho esquerdo
pontos_modelo_3d.push_back(cv::Point3d(225.0f, 170.0f, -135.0f));  // Olho direito
pontos_modelo_3d.push_back(cv::Point3d(-150.0f, -150.0f, -125.0f)); // Boca (esquerdo)
pontos_modelo_3d.push_back(cv::Point3d(150.0f, -150.0f, -125.0f));  // Boca (direito)
```

Fonte: elaborada pelo autor.

O plugin em seguida faz uma chamada ao algoritmo de solução do PnP, e recebe como resultado dois vetores: translação e rotação. O vetor de rotação é retornado em uma representação conhecida como eixo-ângulo (*axis-angle representation*). Através da documentação, sabe-se que o vetor de rotação resultante da projeção na forma comprimida. Portanto, deve-se realizar a equação para extrair o ângulo e normalizar o vetor. Esta operação está demonstrada na Figura 19.

Figura 19. Código da extração do ângulo e normalização sobre o vetor de rotação

```
double dirX = direcao.at<double>(0);
double dirY = -direcao.at<double>(1);
double dirZ = direcao.at<double>(2);
double len = sqrt(dirX*dirX + dirY*dirY + dirZ*dirZ);
dirX /= len;
dirY /= len;
dirZ /= len;
```

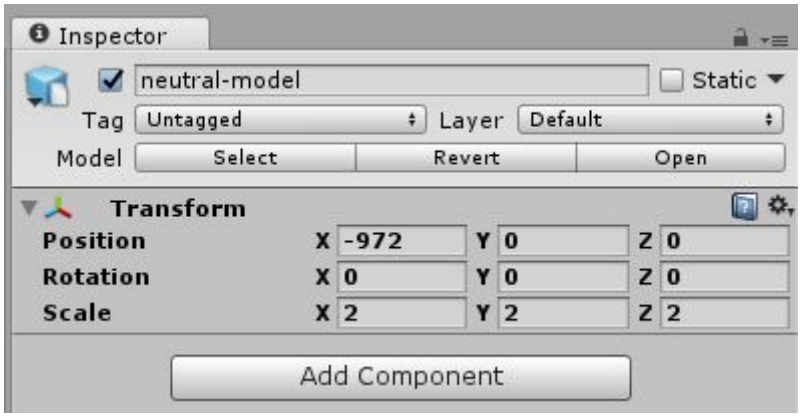
Fonte: elaborada pelo autor.

Por fim, o plugin retorna uma array contendo o vetor de rotação normalizado em conjunto com o ângulo a ser rotacionado para o script do Unity.

3.2.4 Conversão do vetor de rotação para Quaternion

Para representar os GameObjects, nomenclatura dada aos objetos dentro de uma cena, o Unity atribui a eles um componente conhecido como Transform, responsável por armazenar e aplicar as três deformações de transformada rígidas (escala, rotação e translação), e apresentado na Figura 20.

Figura 20. Componente Transform de um GameObject no Unity



Fonte: elaborada pelo autor.

Apesar das coordenadas de rotação estarem apresentadas na representação de ângulos Euler, o Unity utiliza internamente a estrutura de quaternions para lidar com movimento rotacional dos objetos na cena. Devido a uma característica adicional presente nos Quaternions, eles são menos ambíguos do que a representação em ângulos de Euler.

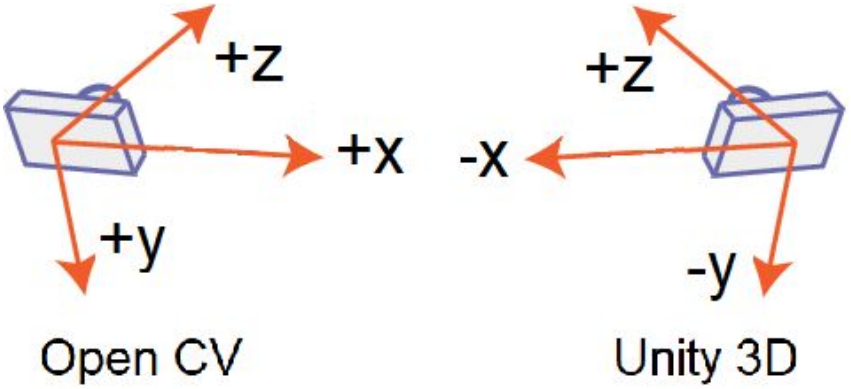
O vetor de rotação recebido da etapa anterior deve ser convertido antes de ser aplicado ao GameObject. Portanto, foi utilizada a função *Quaternion.AxisAngle()*, disponibilizada pela API do Unity, que recebe como parâmetros de entrada o vetor de rotação normalizado e o ângulo de rotação, e retorna o valor em quaternion equivalente. Este valor pode ser atribuído diretamente ao Transform do GameObject.

3.2.5 Rotação aplicada ao modelo

Como dito anteriormente, o sistema de coordenadas utilizado pelo plugin nativo possui sua origem no canto superior esquerdo. O eixo X é positivo para a direita, o eixo Y é positivo para baixo e o eixo Z é positivo em direção à tela (regra da mão direita). Já o sistema de coordenadas do Unity possui os eixos X e Z similares

aos do plugin, mas o eixo Y é positivo para cima (regra da mão esquerda). A comparação entre os eixos pode ser verificada na Figura 21.

Figura 21. Comparação entre os sistemas de coordenada do plugin nativo e do Unity



Fonte: elaborada pelo autor.

Isto significa que, para que a rotação seja aplicada corretamente, deve-se espelhar o valor do eixo Y, multiplicando-o por -1. No caso desta aplicação, opta-se por multiplicado o eixo Z também por -1, para que a rotação do rosto resultante esteja virada para a tela após a aplicação ao Transform. Estas multiplicações já foram realizadas dentro do plugin, como demonstra a Figura 22.

Figura 22. Código que espelha os eixos Y e Z no plugin nativo

```
Vec3d axis{ 0, 0, -1 };
cv::Mat direcao = rotacao3x3 * cv::Mat(axis, false);
double dirY = -direcao.at<double>(1);
```

Fonte: elaborada pelo autor.

4. TESTES E RESULTADO

Neste capítulo, estão descritos os experimentos realizados ao longo do desenvolvimento do framework e as respectivas conclusões, junto com uma especificação dos equipamentos e *softwares* utilizados.

4.1 Materiais

Para a realização dos experimentos, foram necessários os seguintes itens:

4.1.1 Componentes de *hardware*

- Notebook ASUS X550CA; processador i5-3317U (1.7GHz); 6.0GB DDR3 de Memória RAM; placa de vídeo integrada Intel HD Graphics 4000;
- Câmera integrada USB2.0 HD UVC WebCam.

4.1.2 Componentes de *software*

- Sistema Operacional Windows 10 64-Bits;
- Bibliotecas: Unity 3D; Dlib; OpenCV;
- Ambientes de desenvolvimento: Microsoft Visual Studio 2015.

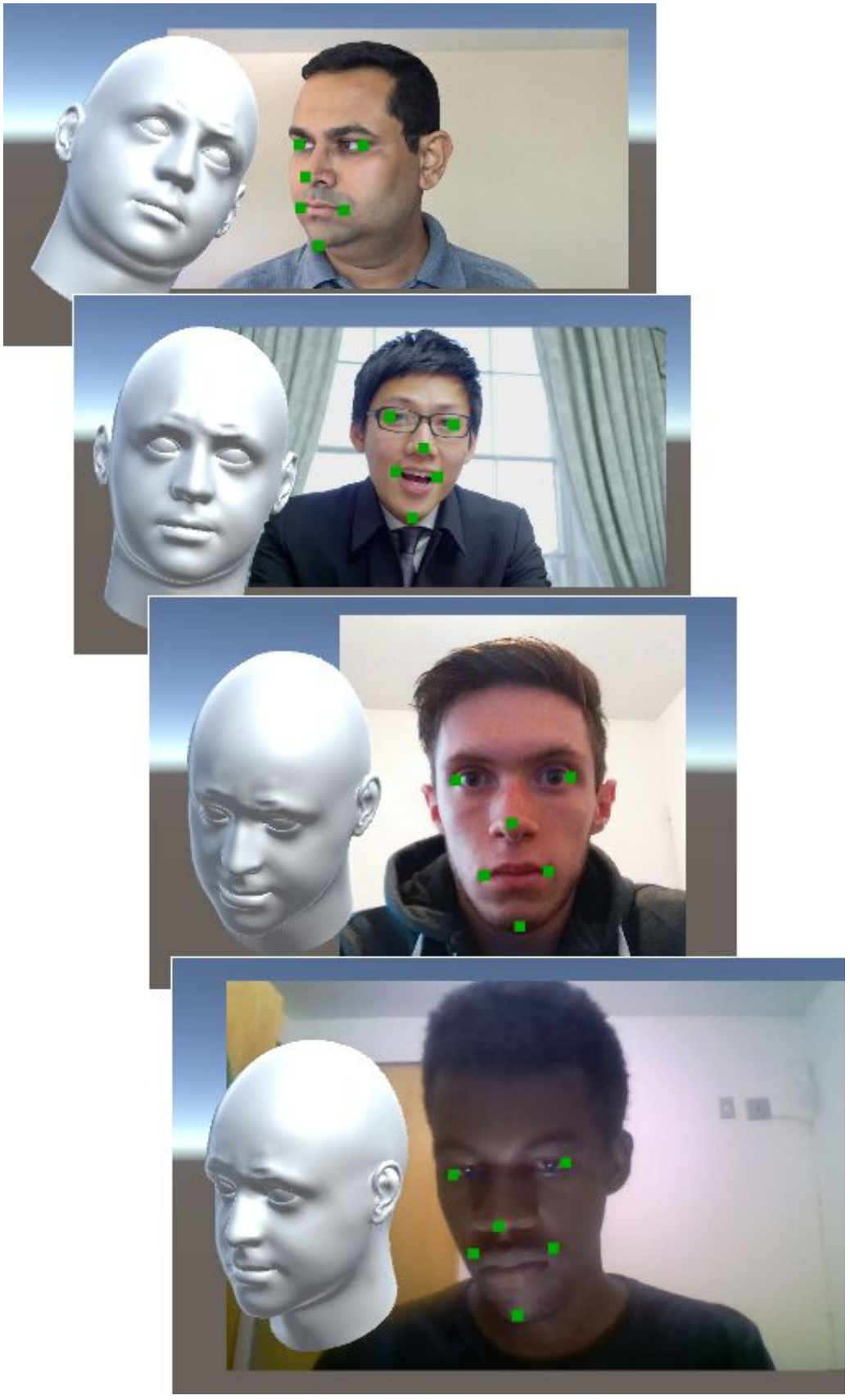
4.2 Experimentos e análise de resultados

Nesta seção, estão descritos os experimentos que foram realizados utilizando o *pipeline* proposto.

4.2.1 Experimento 1

Este experimento visou observar a coerência das rotações resultantes, comparando imagens de entrada de fontes diversas, obtidas na internet. Como descrito anteriormente, as características das imagens foram limitadas a fotos e ilustrações de cabeça contendo rostos inteiros à mostra, em ângulos distintos. As imagens utilizadas foram demonstradas na página 28. A avaliação dos resultados foi feita de forma qualitativa, equiparando a rotação aplicada ao modelo 3D com a foto de origem. Alguns resultados podem ser verificados na Figura 23.

Figura 23. Resultados do Experimento 1



Fonte: elaborada pelo autor.

Como observado, as rotações resultantes coincidiram com as imagens de entrada em metade dos casos. Existem alguns fatores que contribuíram para a incoerência dos resultados. São estes:

- **Aproximação dos valores intrínsecos da câmera.** Os valores de distância focal e coeficiente de distorção da câmera fornecidos no plugin nativo são generalizações de dispositivos reais. Em um cenário como o experimento atual, no qual a câmera (ou o equivalente para o caso das ilustrações) difere de foto para foto, as variações de resultado entre os exemplos se tornam evidentes.
- **Modelo 3D genérico.** Os pontos faciais do modelo 3D utilizados como entrada no plugin nativo são referentes a um modelo genérico. Características importantes em uma foto como tamanho do nariz e distância entre os pontos dos olhos podem levar a interpretações errôneas de projeção, caso o modelo 3D utilizado seja topologicamente diferente do usuário mostrado na imagem.

4.2.2 Experimento 2

Este experimento visou observar a variação da qualidade aparente das rotações resultantes, comparando imagens de uma mesma fonte, porém com dois tipos diferentes de iluminação.

Como imagens de entrada, foram realizadas duas gravações de um usuário, com a disposição da câmera colocada à altura da cabeça do usuário. Na primeira gravação, foram realizadas rotações de cabeça em diversos ângulos, sob a iluminação da luz do dia. Na segunda gravação, tentou-se reproduzir fielmente o mesmo conjunto de movimentos, porém sob iluminação não natural. Pode-se verificar a comparação entre os dois cenários na Figura 24.

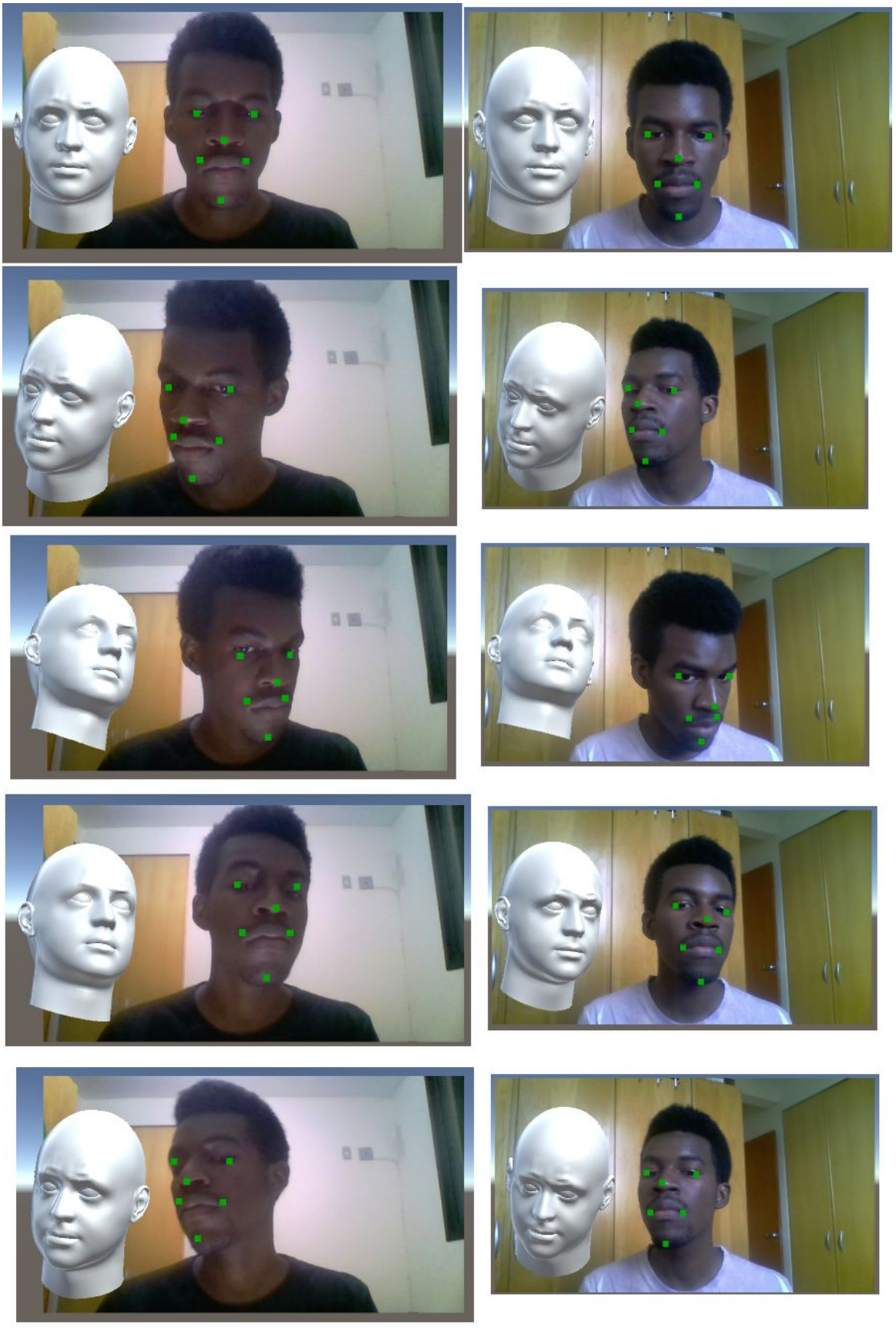
A avaliação dos resultados foi feita de forma similar à do Experimento 1, e pode-se conferir algumas comparações na Figuras 25.

A Figura 26 contém os resultados para as imagens com iluminação natural, porém com suas rotações aplicadas a um modelo 3D específico de usuário gerado pelo sistema elaborado em Cesario (2017).

Figura 24. Configuração de iluminação das duas gravações. À esquerda, iluminação artificial não natural. À direita, iluminação natural, sob a luz do dia.

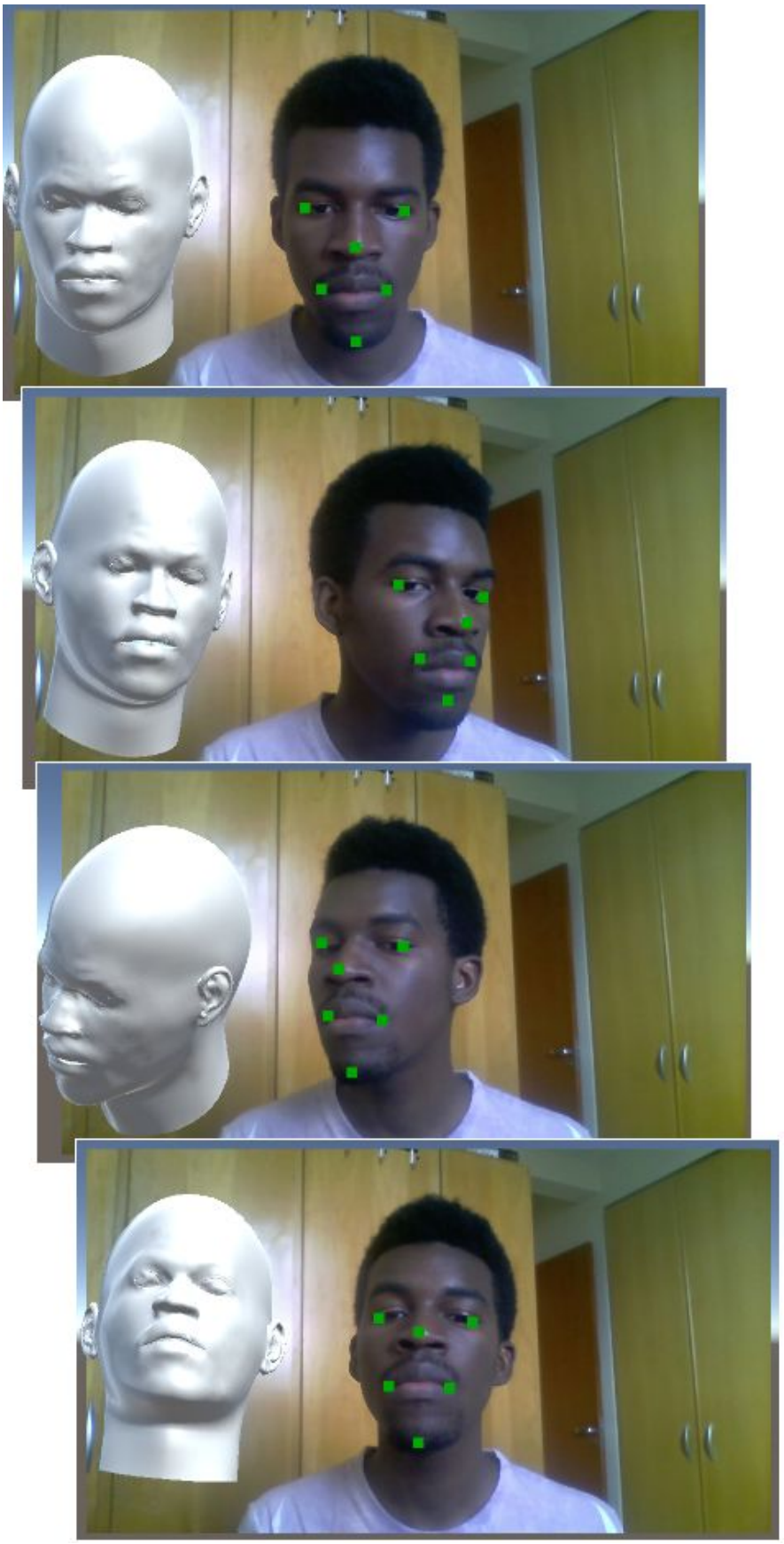
Fonte: elaborada pelo autor.

Figura 25. Resultados do Experimento 2



Fonte: elaborada pelo autor.

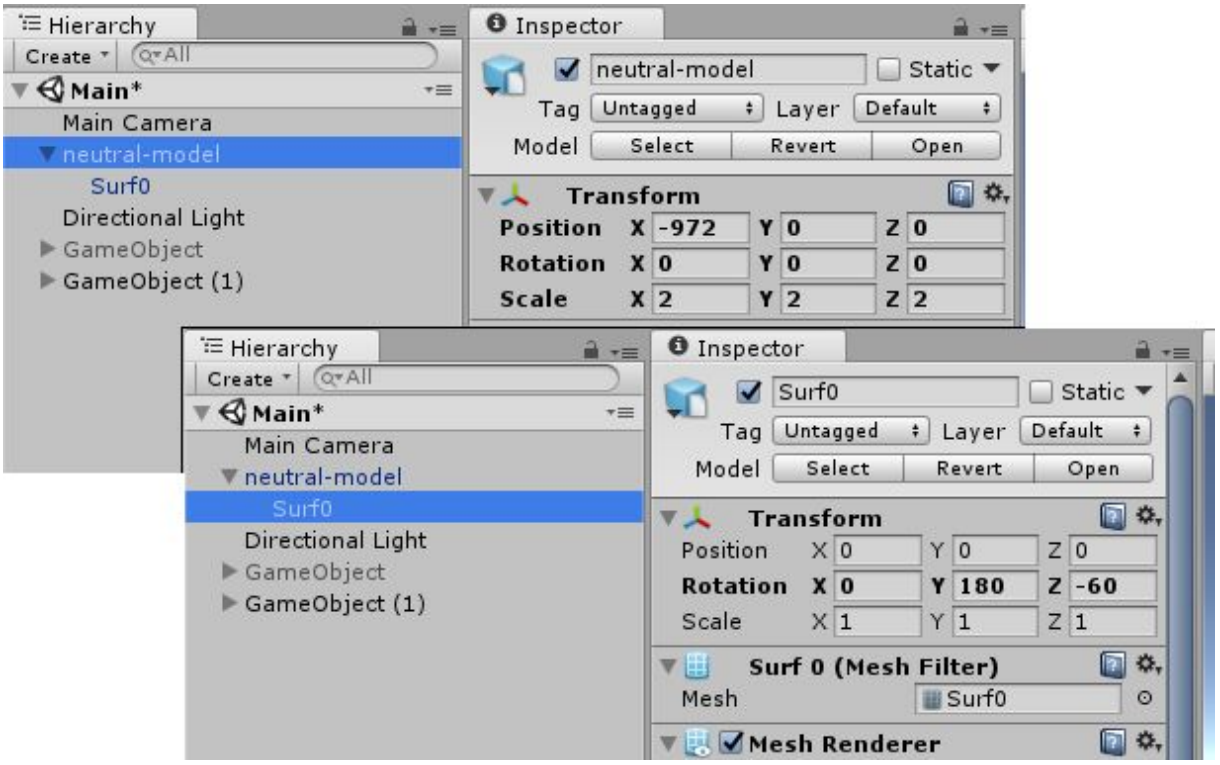
Figura 26. Resultados do Experimento 2 com modelo específico



Fonte: elaborada pelo autor.

As rotações resultantes destas imagens de entrada foram mais consistentes em relação às do Experimento 1, como esperado. A iluminação natural produziu uma localização de pontos visualmente mais coerente do que a segunda iluminação, porém a diferença não afetou as rotações drasticamente, de um modo geral. É possível inferir que a aplicação funciona de forma semelhante em ambientes com iluminação variada. É importante ressaltar que, para este experimento, foi necessário um segundo processo de rotação para contrabalancear o deslocamento proveniente do plugin nativo e fazer com que o modelo 3D ficasse com o rosto virado para a câmera. Portanto, o GameObject do modelo 3D continha uma rotação aplicada manualmente em seu Transform, mas era descendente hierárquico de um segundo GameObject, ao qual foi aplicada a rotação do plugin nativo. Este esquema está demonstrado na Figura 27.

Figura 27. Esquema hierárquico do modelo 3D para o Experimento 2



Fonte: elaborada pelo autor.

5. CONCLUSÃO

Este trabalho objetivou o desenvolvimento de um módulo de alinhamento de *blendshapes* à captura do usuário, abrangendo as etapas de estruturação, implementação e teste. O projeto se utilizou das bibliotecas de código aberto Dlib e OpenCV, além da tecnologia de plugins nativos fornecida pelo Unity para interligar a API do *software* com estas bibliotecas.

A utilização das bibliotecas de código aberto foi essencial, pois elas fornecem gratuitamente algoritmos de aprendizado de máquina rápidos e eficientes. O Dlib possui, além de uma documentação bem explicativa, blocos de código de exemplo para demonstrar a utilização das funções disponíveis, o que se tornou uma referência bastante útil para a escrita do plugin nativo.

Não foi possível obter resultados satisfatórios em todos os casos exemplificados pelos experimentos devido ao fato de terem sido usados dados numéricos aproximados. A falta de informações corretas como um modelo 3D específico de usuário e os valores perdidos na conversão do tipo *double* para o tipo *float* causou incoerências bastante perceptíveis em alguns casos.

O Unity se apresentou como um ambiente de desenvolvimento capaz de lidar com manipulação de objetos 3D, proporcionando aplicação de transformações rígidas com o mouse de forma bastante intuitiva. O *software* possui uma interface programável para alteração de variáveis, o que possibilita um feedback mais rápido e conveniente para a aplicação desenvolvida. Sua interação com os plugins nativos permite a utilização de diversas bibliotecas externas, que podem estender a funcionalidade do editor de maneira prática.

O *pipeline* proposto se mostrou adequado para atingir o resultado final a partir das informações de entrada. A manipulação de informações de rotação e translação dentro do Unity, apesar de inconsistente em alguns casos, validou o procedimento. O conceito apresentado no capítulo 3, e mais precisamente na página 21, é genérico o suficiente para ser replicado em outras plataformas, utilizando tecnologias diferentes de captura e análise de imagens.

Para trabalhos posteriores, visa-se a compatibilidade da aplicação com um fluxo de vídeo constante como entrada. A inicialização da base de predição toma um

tempo considerável do processamento atual, e pode ser reduzido com a adição de funções no plugin nativo específicas para inicialização, referência e remoção de um ponteiro contendo a base de predição. Esta operação separa o processo de carregar a base na memória do processo de detecção dos pontos faciais, o que aumentaria consideravelmente a velocidade do processamento.

Um outro recurso que pode aprimorar a coerência das rotações é o de inserir modelos 3D com topologia mais próxima à do rosto do usuário capturado. Uma forma de gerar modelos 3D a partir de capturas faciais contendo dados de RGB e profundidade foi proposta por Cesario (2017). Para a entrada dos pontos faciais no modelo, Cesario (2017) sugere a utilização de um algoritmo detector de características faciais em modelos 3D (PERAKIS et al., 2010) em conjunto com o Dlib. Estas melhorias podem tornar o *pipeline* mais robusto e confiável, e aperfeiçoar os métodos de verificação qualitativa em um sistema de animação facial.

REFERÊNCIAS

BRAUN, A.. **Aprendizado e utilização do estilo de movimento facial na animação de avatares**. 2014. 124 f. Tese (Doutorado) - Curso de Pós-graduação em Ciência da Computação, Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, 2014. Cap. 3.

CESARIO, V. S. M.. **Um framework para a geração de blendshapes de expressões faciais específicas de usuário direcionado a um sistema de animação facial realista.** 2017. 28 f. Universidade Estadual Paulista Júlio de Mesquita Filho, Bauru, 2017.

DALAL, N.; TRIGGS, B.. Histograms of Oriented Gradients for Human Detection. **2005 IEEE Computer Society Conference On Computer Vision And Pattern Recognition (CVPR'05),** [s.l.], p.886-893, jun. 2005. IEEE. http://dx.doi.org/10.1109/cvpr.2005.177.

DENG, Z.; NOH, J.. **Computer Facial Animation**: A Survey. In: DENG, Z.; NEUMANN, U.. Data-Driven 3D Facial Animation. Londres: Springer-verlag, 2008. Cap. 1. p. 1-28.

FANELLI, G.; WEISE, T.; GALL, J.; GOOL, L. V.. Real Time Head Pose Estimation from Consumer Depth Cameras. In: ANNUAL SYMPOSIUM OF THE GERMAN ASSOCIATION FOR PATTERN RECOGNITION, 33., 2011, Frankfurt. **Anais... .** Frankfurt: DAGM, 2011. 8p.

FANELLI, G.; GALL, J.; VAN GOOL, L.. Real Time Head Pose Estimation with Random Regression Forests. In: COMPUTER VISION AND PATTERN RECOGNITION, 1., 2011, Colorado Springs. **Anais... .** Colorado Springs: Ieee, 2011. 8p.

GOOGLE (Estados Unidos). Digital Equipment Corporation. Sing Bing Kang. **Hands-free interface to a virtual reality environment using head tracking**. US nº 6009210 A, 5 mar. 1997, 28 dez. 1999. 1999.

GUENTER, B.; GRIMM, C.; WOOD, D.; MALVAR, H.; PIGHIN, F.. **Making faces**. In: SIGGRAPH 98, 25., 1998, Orlando. Proceedings... .Orlando: ACM Transactions On Graphics, 1998. p. 55 - 66.

GUNES, H.; PANTIC, M.. **Dimensional Emotion Prediction from Spontaneous Head Gestures for Interaction with Sensitive Artificial Listeners.** Londres: Springer-verlag Berlin Heidelberg, 2010. 7 p.

JONES, M.; VIOLA, P.. **Fast Multi-view Face Detection.** Cambridge: Mitsubishi Electric Research Laboratories, 2003. 11 p.

KAZEMI, V.; SULLIVAN, J.. One Millisecond Face Alignment with an Ensemble of Regression Tree. In: THE IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, 2014, Columbus. **Proceedings... .** Columbus: IEEE, 2014. p. 1867 - 1874.

KING, D. E.. Dlib-ml: A Machine Learning Toolkit. **Journal Of Machine Learning Research.** Estados Unidos, p. 1755-1758. jul. 2009.

KING, D. E.. **Easily Create High Quality Object Detectors with Deep Learning.** 2016. Disponível em: <http://blog.dlib.net/2016/10/easily-create-high-quality-object.html>. Acesso em: 25 nov. 2017.

KING, D. E.. **Max-Margin Object Detection.** 2015. Disponível em: <https://arxiv.org/abs/1502.00046>. Acesso em: 25 nov. 2017.

KING, D. E.. **Real-Time Face Pose Estimation.** 2014. Disponível em: <http://blog.dlib.net/2014/08/real-time-face-pose-estimation.html>. Acesso em: 25 nov. 2017.

LEWIS, J. P.; ANJYO, K.; RHEE, T.; ZHANG, M.; PIGHIN, F.; DENG, Z.. **Practice and theory of blendshape facial models**. In: CITESEER. Eurographics (State of the Art Reports). [S.l.], 2014. p. 1–3.

LUO, C.; YU, J.; JIANG, C.; LI, R.; WANG, Z. **Real-time control of 3d facial animation**. In: Multimedia and Expo (ICME), 2014 IEEE International Conference on. [S.l.: s.n.], 2014. p. 1–6.

MALLICK, S. **Head Pose Estimation using OpenCV and Dlib.** 2016. Disponível em <https://www.learnopencv.com/head-pose-estimation-using-opencv-and-dlib/>. Acesso em: 25 nov. 2017.

MOESLUND, T. B.; GRANUM, E.. A Survey of Computer Vision-Based Human Motion Capture. **Computer Vision And Image Understanding,** [s.l.], v. 81, n. 3, p.231-268, mar. 2001. Elsevier BV. http://dx.doi.org/10.1006/cviu.2000.0897.

MOTTA, E. S.. **Desenvolvimento de um método para a captura de movimentos humanos usando uma câmera RGB-D.** 2016. 101 f. Dissertação (Mestrado) - Curso de Pós-graduação em Ciência da Computação, Universidade Estadual Paulista Júlio de Mesquita Filho, São José do Rio Preto, 2016.

MURPHY-CHUTORIAN, E.; DOSHI, A.; TRIVEDI, M. M.. Head Pose Estimation for Driver Assistance Systems: A Robust Algorithm and Experimental Evaluation. **2007 IEEE Intelligent Transportation Systems Conference,** [s.l.], p.1-6, set. 2007. IEEE. http://dx.doi.org/10.1109/itsc.2007.4357803.

MURPHY-CHUTORIAN, E.; TRIVEDI, M. M.. Head Pose Estimation in Computer Vision: A Survey. **IEEE Transactions On Pattern Analysis And Machine Intelligence,** [s.l.], v. 31, n. 4, p.607-626, abr. 2009. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/tpami.2008.106.

NAPLES, A.; NGUYEN-PHUC A.; COFFMAN, M.; KRESSE, A.; FAJA, S.; BERNIER, R.; MCPARTLAND, J. C.. A computer-generated animated face stimulus set for psychophysiological research. **Behavior Research Methods,** [s.l.], v. 47, n. 2, p.562-570, 16 jul. 2014. Springer Nature. http://dx.doi.org/10.3758/s13428-014-0491-x.

OSADCHY, M.; CUN, Y. L.; MILLER, M. L.. Synergistic Face Detection and Pose Estimation with Energy-Based Models. **Journal Of Machine Learning Research.** Estados Unidos, p. 1197-1215. 2007.

PERAKIS, P. ; PASSALIS, G.; THEOHARIS, T.; KAKADIARIS, I. A.. **3D Facial Landmark Detection & Face Registration:** A 3D Facial Landmark Model & 3D Local Shape Descriptors Approach. Atenas: University Of Athens, 2010.

ROWEIS, S.. **Levenberg-Marquardt Optimization.** 2005. Disponível em: <https://pdfs.semanticscholar.org/f755/72f88df05beed9c6dddabf131967517caa76.pdf>. Acesso em: 25 nov. 2017.

SIFAKIS, E.; NEVEROV, I.; FEDKIW, R. **Automatic Determination of Facial Muscle Activations from Sparse Motion Capture Marker Data**. Los Angeles: ACM Transactions On Graphics, 2005. 9 p.

SILVA, C. E. R. C.. **O desenvolvimento de um sistema de animação facial baseado em performance e no uso de câmera RGB-D.** 2017. 101 f. Dissertação (Mestrado) - Curso de Pós-graduação em Ciência da Computação, Universidade Estadual Paulista Júlio de Mesquita Filho, São José do Rio Preto, 2017.

SZELISKI, R.. **Computer Vision:** Algorithms and Applications. Springer, 2010. 812p.

TERZOPOULOS, D.; WATERS, K.. Physically-Based Facial Modeling, Analysis, and Animation. **Journal Of Visualization And Computer Animation.** Austin, p. 73-80. 1990.

TRIER, O. D.; JAIN, A. K.; TAXT, T.. Feature extraction methods for character recognition - a survey. **Pattern Recognition.** East Lansing, p. 641-662. 1996.

WITTIG, S.; KLOOS, U.; RÄTSCH, M.. Animation of Parameterized Facial Expressions for Collaborative Robots. In: INFORMATICS INSIDE, 1., 2015, Reutlingen. **Anais... .** Reutlingen: Steffen Wittig, 2015. 2p.

YANG, R.; ZHANG, Z.. Model-based Head Pose Tracking With Stereovision. In: IEEE INTERNATIONAL CONFERENCE ON AUTOMATIC FACE AND GESTURE RECOGNITION, 5., 2002, Washington. **Proceedings... .** Washington: Ieee, 2002. 6p.

ZHANG, Z.; LUO, P.; LOY, C. C.; TANG, X.. **Facial Landmark Detection by Deep Multi-task Learning.** 2014. Disponível em: <http://mmlab.ie.cuhk.edu.hk/projects/TCDCN.html>. Acesso em: 25 nov. 2017.

ZHU, Z.; LUO, P.; WANG, X.; TANG, X.. **Deep Learning Multi-View Representation for Face Recognition.** 2014. Disponível em: <https://arxiv.org/abs/1406.6947>. Acesso em: 25 nov. 2017.

ZUCCONI, A.. **How to Write Native Plugins for Unity.** 2015. Disponível em: <https://www.alanzucconi.com/2015/10/11/how-to-write-native-plugins-for-unity/>. Acesso em: 25 nov. 2017.

APÊNDICE A — CÓDIGO-FONTE DO PLUGIN NATIVO: LANDMARKSDLL.CPP

```
#include <dlib/image_processing/frontal_face_detector.h>
#include <dlib/image_processing/render_face_detections.h>
#include <dlib/image_processing.h>
#include <dlib/image_io.h>
#include <opencv2/opencv.hpp>
#include "LandmarksDll.h"

using namespace dlib;
using namespace cv;
using namespace std;

extern "C" {
	bool RastrearPontosFaciais(char* dir_base_predicao, char* dir_img, int
total_pontos,	int * id_pontos, double** array_landmarks) {
		frontal_face_detector detector = get_frontal_face_detector();
		shape_predictor sp;
		deserialize(dir_base_predicao) >> sp;
		array2d<rgb_pixel> img;
		load_image(img, dir_img);
		std::vector<dlib::rectangle> dets = detector(img);
		if (dets.size() > 1)
			return false;
		full_object_detection shape = sp(img, dets[0]);

		double w = img.nc();
		double h = img.nr();

		memcpy(&(*array_landmarks)[0], &w, sizeof(double));
		memcpy(&(*array_landmarks)[1], &h, sizeof(double));

		for (int i = 0; i < total_pontos; i++)
		{
			double x = (double)(shape.part(id_pontos[i]).x());
			double y = (double)(shape.part(id_pontos[i]).y());
			memcpy(&(*array_landmarks)[2 * (i + 1)], &x,
sizeof(double));
			memcpy(&(*array_landmarks)[2 * (i + 1) + 1], &y,
sizeof(double));
		}

		return true;
	}

	bool CalcularRotacao(char* dir_img, int* coords, double**
array_rotacao) {
		cv::Mat im = cv::imread(dir_img);

		if (im.data == NULL)
			return false;

		std::vector<cv::Point2d> pontos_img_2d;
		pontos_img_2d.push_back(cv::Point2d(coords[0], coords[1]));
		// Nariz
```

```
            pontos_img_2d.push_back(cv::Point2d(coords[2], coords[3]));
            // Queixo
            pontos_img_2d.push_back(cv::Point2d(coords[4], coords[5]));
            // Olho esquerdo
            pontos_img_2d.push_back(cv::Point2d(coords[6], coords[7]));
            // Olho direito
            pontos_img_2d.push_back(cv::Point2d(coords[8], coords[9]));
            // Boca (esquerdo)
            pontos_img_2d.push_back(cv::Point2d(coords[10], coords[11]));
      // Boca (direito)

            std::vector<cv::Point3d> pontos_modelo_3d;
            pontos_modelo_3d.push_back(cv::Point3d(0.0f, 0.0f, 0.0f));
// Nariz
            pontos_modelo_3d.push_back(cv::Point3d(0.0f, -330.0f, -65.0f));
// Queixo
            pontos_modelo_3d.push_back(cv::Point3d(-225.0f, 170.0f,
-135.0f));  // Olho esquerdo
            pontos_modelo_3d.push_back(cv::Point3d(225.0f, 170.0f, -135.0f));
// Olho direito
            pontos_modelo_3d.push_back(cv::Point3d(-150.0f, -150.0f,
-125.0f)); // Boca (esquerdo)
            pontos_modelo_3d.push_back(cv::Point3d(150.0f, -150.0f,
-125.0f));  // Boca (direito)

            double dist_focal = im.cols;
            Point2d centro = cv::Point2d(im.cols / 2, im.rows / 2);
            cv::Mat matriz_camera = (cv::Mat_<double>(3, 3) << dist_focal, 0,
centro.x, 0, dist_focal, centro.y, 0, 0, 1);
            cv::Mat coef_dist = cv::Mat::zeros(4, 1,
cv::DataType<double>::type);
            cv::Mat vetor_rotacao;
            cv::Mat vetor_translacao;
            cv::Mat rotacao3x3;

            cv::solvePnP(pontos_modelo_3d, pontos_img_2d, matriz_camera,
coef_dist, vetor_rotacao, vetor_translacao);
            cv::Rodrigues(vetor_rotacao, rotacao3x3);

            Vec3d axis{ 0, 0, -1 };
            cv::Mat direcao = rotacao3x3 * cv::Mat(axis, false);
            double dirY = -direcao.at<double>(1);

            double dirX = direcao.at<double>(0);
            double dirZ = direcao.at<double>(2);
            double len = sqrt(dirX*dirX + dirY*dirY + dirZ*dirZ);
            dirX /= len;
            dirY /= len;
            dirZ /= len;

            memcpy(&(*array_rotacao)[0], &(dirX), sizeof(double));
            memcpy(&(*array_rotacao)[1], &(dirY), sizeof(double));
            memcpy(&(*array_rotacao)[2], &(dirZ), sizeof(double));
            memcpy(&(*array_rotacao)[3], &(len), sizeof(double));

            return true;
      }
}
```

APÊNDICE B — CABEÇALHO DO PLUGIN NATIVO: LANDMARKSDLL.H

```
#pragma once
#define DLLEXPORT_API __declspec(dllexport)

extern "C" {
	DLLEXPORT_API bool RastrearPontosFaciais(char* path_predictor, char*
path_img, int num_points, int * point_list, double** landmarks_array);
	DLLEXPORT_API bool CalcularRotacao(char* img_path, int* coords,
double** rotation_array);
}
```