Interação por Rastreamento de Mão em ambiente de Realidade Virtual
Interaction by Hand-Tracking in Virtual Reality

Mateus C. L. de Castro
Departamento de Engenharia de Computação
Instituto Militar de Engenharia
Rio de Janeiro, Brasil
samuraiexx@gmail.com

João P. A. Xavier
Departamento de Engenharia de Computação
Instituto Militar de Engenharia
Rio de Janeiro, Brasil
joaopedroaxavier@gmail.com

Paulo F. F. Rosa
Departamento de Engenharia de Computação
Instituto Militar de Engenharia
Rio de Janeiro, Brasil
rpaulo@ime.eb.br

Jauvane C. de Oliveira
Coordenação de Métodos Matemáticos e Computacionais
Laboratório Nacional de Computação Científica
Petrópolis, Brasil
jauvane@acm.org

***Abstract*—This article describes the ellaboration of a low cost system that allows user interaction with VR environments modeled in Unity and viewed through Google Card Board. The proposed system divides the processing in a client-server architecture in which the VR experience is executed on client, while it sends pictures captured from the mobile device inside the Card Board to the server. This server processes the hand pose information and sends it to the client.**

***Resumo*—Este artigo descreve a criação de um sistema de baixo custo que permite a interação de um usuário com ambientes virtuais modelados em *Unity* e visualizados através do *Google Card Board* - a interação testada se baseia no uso das mãos para mover objetos em uma tarefa de *pick and place*. O sistema proposto divide o processamento em uma arquitetura cliente-servidor na qual o ambiente virtual é executado no cliente, ao mesmo tempo que este envia fotos do celular do *Card Board* para o servidor. Este processa e retorna a pose da mão presente na imagem e envia de volta para o cliente, o qual renderiza e processa interações.**

***Palavras-chave*—Redes Neurais, Aprendizado Profundo, Realidade Virtual, Sistema de Tempo Real, Análise de Imagens, Detecção de Posição**

I. Introdução

O principal objetivo almejado durante a elaboração de um ambiente de Realidade Virtual (RV) é um alto grau de imersão, que pode ser definido como o quanto o usuário é capaz de sentir-se mentalmente imerso ou presente na simulação. Um exemplo simples é no caso da visão, em que pode ser usado um óculos que bloqueia a visão do mundo exterior, enquanto que as telas posicionadas em frente aos olhos do usuário apresentam os elementos visuais do ambiente. Outra maneira de aumentar o fator de imersão é melhorando a interatividade do usuário. Outros conceitos já foram relacionados com a imersão em ambientes virtuais, como o senso de presença e singularidade do ambiente [1].

Aproximando-se do propósito de tornar essa tecnologia mais acessível, espera-se que este trabalho aumente a disponibilidade de fatores de imersão, ao disponibilizar o código de uma ferramenta *open-source* que permite *hand-tracking* em ambiente virtual, usando apenas um celular e um servidor. Com o rastreamento das mãos do usuário e a possibilidade de interação com objetos em realidade virtual, torna-se desnecessário o uso de aparelhos auxiliares como sensores e controladores, ficando a critério do desenvolvedor de analisar o *tradeoff* entre custo e precisão aumentada por esses aparelhos.

O objetivo final deste trabalho é desenvolver uma solução de baixo custo que viabilize a interação através das mãos em tempo real com ambientes de Realidade Virtual. Para isto, será verificada a capacidade de realizar uma atividade de *pick and place*, que consistirá em mover um cubo para uma área específica.

II. Solução Proposta

O atual problema é permitir a interação das mãos em um ambiente virtual em tempo real, usando um celular comum tanto para capturar quadros quanto para reproduzir o ambiente virtual.

A solução proposta para este problema consiste em dividir a arquitetura em cliente-servidor, mantendo o processamento mais custoso do lado do servidor, a fim de diminuir o tempo total de processamento e atingir processamento em tempo real.

Tendo em vista esta arquitetura, o cliente (celular) irá:

1) Capturar a imagem da Câmera;
2) Comprimir a imagem e enviar para o servidor;
3) Receber de volta os pontos-chave da mão em três dimensões;
4) Renderizar um modelo da mão no ambiente virtual;

5) Fazer processamentos menores referentes a interação com o ambiente virtual.

E do lado do servidor irá:

1) Aguardar o cliente enviar cada quadro;
2) Receber e descomprimir o quadro;
3) Detectar os pontos-chave em duas dimensões usando redes neurais conhecidas;
4) Realizar cinemática inversa para obter os pontos-chave em três dimensões;
5) Responder o cliente com os pontos-chave em três dimensões.

III. Implementação

As próximas sessões irão explicitar todo o processo pelo qual cada quadro passa. Vale observar que sempre que o cliente recebe os pontos em três dimensões um novo quadro é capturado e enviado ao servidor, de tal forma que qualquer mudança de pose na mão irá ser refletida no ambiente virtual.

Essa abordagem permite que o celular utilizado não fique sobrecarregado, utilizando-o apenas para interface e renderização do VR, que são essencialmente tarefas atribuídas a este aparelho nesta aplicação. Ao diminuir a carga de processamento do celular, diminui-se o consumo de bateria causado por essa aplicação, bem como aumenta a velocidade de processamento por estar utilizando o processador e a paca de vídeo do computador, que são mais rápidos.

A. Cliente

O cliente foi implementado usando a ferramenta de desenvolvimento de jogos *Unity* [2], com o intuito de reduzir a complexidade do projeto na parte de visão computacional, visto que esta ferramenta fornece abstrações para objetos, câmeras e luzes, além de outros elementos. Outro motivo é o fato desta ferramenta ser bem documentada e popular atualmente segundo Mishra e Shrawankar[3], desta forma desenvolvedores podem integrar esse processamento na criação de novas aplicações.

1) Captura de quadros, compressão de quadros e envio para o servidor: O inicio do processamento de cada quadro corre na captura deste usando uma das câmeras do celular. A API do *Unity* foi usada para a captura de quadros de 640x480 pixels no formato RGB.

As dimensões dos quadros foram escolhidas de tal forma que as dimensões da região da mão à distância máxima da câmera, a qual será recortada pelo servidor, tenha dimensões próximas à entrada da rede neural utilizada para processar o recorte no servidor. Desta forma não há perda de informação e o tempo para a transferência do quadro - além outros processos cuja complexidade depende das dimensões do quadro - é minimizado.

Após a captura da imagem, adicionou-se um passo de compressão usando o algoritmo *VP8* criado pela *Google* [4] - outros formatos como *AV1* [5] não foram testados. O passo de compressão foi escrito em C++ usando as bibliotecas fornecidas pela *Google*, compilado para a arquitetura *Android* e adicionado ao *Unity* como código nativo.

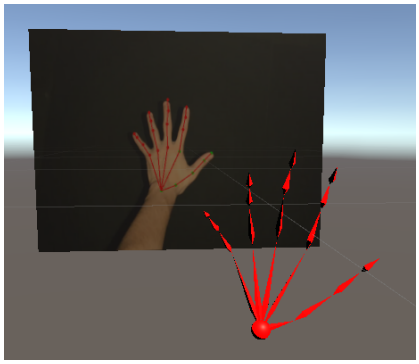

Fig. 1. Cena simples no *Unity* mostrando o anteparo com o quadro capturado junto dos pontos-chave e o esqueleto tridimensional após processamento no servidor.

Finalmente o quadro comprimido é enviado ao servidor para processamento.

2) Recebimento de dados e processamento: Após o processamento, o servidor envia de volta para o cliente os ângulos das juntas, a posição do pulso relativo à câmera, a rotação do pulso e adicionalmente todos os pontos-chave da mão em três e duas dimensões.

Um objeto criado para manter estes dados é atualizado e chama todos os métodos registrados para alertar a atualização dos dados. A implementação atual chama três métodos: o primeiro método atualiza o anteparo mostrado na imagem 1; o segundo cria um objeto 3d para representar a mão; e o terceiro analisa a distância entre a ponta do indicador e a ponta do polegar para definir se está ou não tentando agarrar um objeto.

3) Analise de pose da garra: Para analisar se a mão está segurando um objeto ou não utiliza-se uma maquina de estados tendo os estados “Garra Fechada” e “Garra Aberta”. A cada atualização nos dados, o estado é atualizado baseado nas seguinte condições:

- Se o estado atual é “Garra Aberta” e a distância entre a ponta do indicador e a ponta do polegar é maior que um determinado limite, o próximo estado é “Garra Aberta”.
- Se o estado Atual é “Garra Aberta” e a distância é menor que o limite o próximo estado é “Garra Fechada” e o objeto segurável mais próximo dentro de outro limite passa a ter o estado “Sendo Segurado”.
- Se estado atual é “Garra Fechada” e a distância é menor que o limite, o próximo estado é “Garra Fechada”, nada muda no estado de outros objetos.
- Se o estado atual é “Garra Fechada” e a distância é maior que o limite, o objeto que estava no estado “Sendo Segurado” passam para o estado “Solto”.

Enquanto um objeto está no estado “Sendo Segurado”, a cada atualização a variação na posição do ponto médio entre o indicador e o polegar é somado à posição do objeto. A imagem 2 demonstra o estado “Garra Fechada” enquanto o cubo verde é um objeto no estado “Sendo Segurado”.

B. Servidor

A maior parte do processamento está no servidor. Este recebe cada quadro, descomprime, recorta, encontra os pontos-

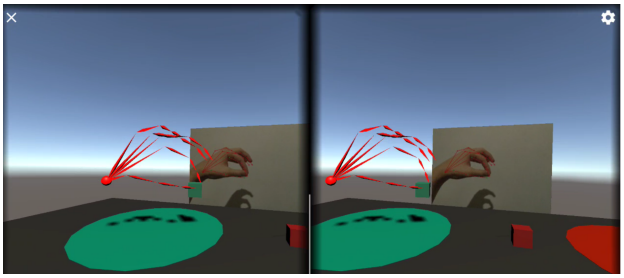

Fig. 2. Cena simples no *Unity* mostrando a tela do celular, com a visão de cada olho separada. Nesta cena pode-se notar um objeto sendo segurado.

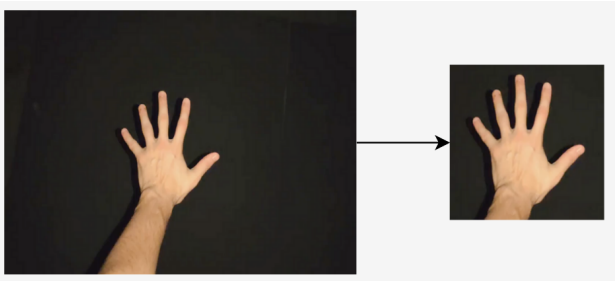

Fig. 3. Na esquerda imagem recebida do cliente e na direita imagem cortada na região de interesse - em torno da mão.

chave, faz a cinemática inversa baseado nestes pontos e retorna a pose da mão em três dimensões para o cliente.

A implementação foi feita em C++ tendo em vista otimizações no desempenho e a disponibilidade de bibliotecas de alto desempenho de visão computacional, álgebra linear e de regressão não-linear.

1) Recebimento e descompressão de quadros: O servidor fica ouvindo uma determinada porta para receber as solicitações de processamento de um quadro. Este processamento não é feito de forma funcional, uma vez que a descompressão depende dos quadros recebidos anteriormente e a pose é inicializada com a posição do quadro anterior.

Quando uma nova solicitação chega, os quadros comprimidos são recebidos e decodificados com *VP8*, a mesma biblioteca (*VPX*) utilizada no cliente. Usando a biblioteca *OpenCV* os quadros são então convertidos de *YV12* para *GBR*.

2) Corte da imagem na região de interesse: A tarefa de detectar os pontos-chave da mão é complexa e computacionalmente custosa. O tempo de execução da rede tem uma relação direta com o tamanho da imagem e, em contrapartida, imagens de tamanhos menores contém menos informação e diminuem a precisão do resultado. Como o objetivo é apenas detectar os pontos-chave da mão, a parte da imagem que contém a mão pode ser detectado e recortado do restante primeiro. Na implementação deste trabalho há apenas o recorte de um quadrado cujos lados são paralelos aos eixos da imagem como ilustrado na imagem 3.

A rede neural descrita em Redmon e Farhadi[6] foi criada para detectar a posição de objetos em imagens para qual foi treinada. No presente trabalho uma rede deste tipo, que foi treinada para detectar mãos, foi utilizada para determinar a região de interesse da imagem. Apesar de resolver o problema

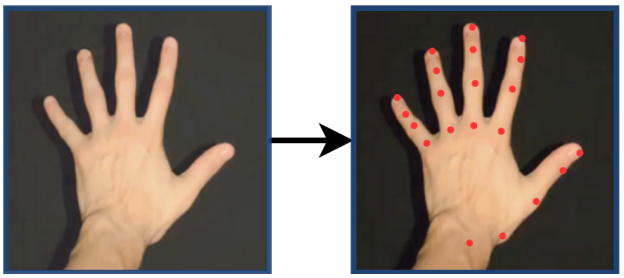

Fig. 4. Na esquerda imagem de entrada para rede neural e na direita imagem com pontos vermelhos em cima dos pontos detectados como de maior probabilidade para cada uma das 21 juntas.

inicial, usar esta rede em todos os quadros aumenta o tempo médio de execução, em vez disto, ela só é utilizada no primeiro quadro ou quando a detecção dos pontos-chave tem baixa confiança, o que pode indicar que o região recortada não contém mais os pontos. Nos demais quadros, é recortada uma região que é calculada baseada na posição dos pontos detectados no quadro anterior.

Vale notar que movimentos bruscos podem fazer o tempo de processamento de cada quadro aumentar uma vez que será necessário utilizar mais vezes a rede neural para detecção da região de interesse. Baixa quantidade de quadros processados por segundo também pode ter o mesmo efeito.

3) Detecção dos pontos chave da mão: Para fazer a detecção dos pontos chave basta redimensionar a região de interesse para o tamanho esperado pela entrada pelo rede neural utilizada para detectar os pontos-chave. A rede neural desenvolvida em Cao et al.[7] foi a escolhida - apesar da rede desenvolvida em Mueller et al.[8] ser mais rápida, esta apresenta imprecisões que impossibilitam a tarefa de *pick and place*. A rede escolhida usa uma arquitetura chamada de *Convolutional Pose Machines*.

A saída da redes neural são 21 matrizes bidimensionais cujo o valor em cada posição na i-ésima matriz é a probabilidade da i-ésima junta estar nesta posição, após redimensionamento, na imagem original. Apenas a informação do ponto com probabilidade máxima e o valor desta probabilidade, a confiança, são utilizados nos próximos passos. A figura 4 ilustra uma possibilidade para os pontos de maior probabilidade detectados pela rede neural.

4) Aplicação de filtro nos pontos detectados: Os pontos detectados pela rede neural em duas imagens com a mão na mesma posição mas com pequenas variações, por exemplo da posição relativa da mão na região de interesse, faz com que haja pequenas variações na predição da posição pela rede neural, fazendo com que a posição em função do tempo tenha ruído. Por conta deste ruido aplicações simples deste sistema como identificação da posição de garra, descrito na sessão do cliente, não eram possíveis, visto que o erro, causado pelo ruído, na posição da ponta do indicador e a do polegar eram maiores que a distância determinada como limite do estado de "Garra Fechada".

Para melhorar a qualidade do sinal, foi-se usado o mesmo filtro utilizado em Mueller et al.[8] e definido em Casiez,

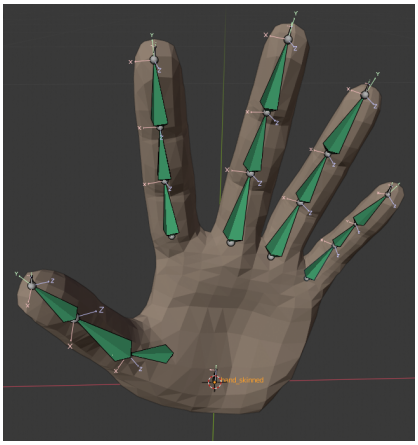

Fig. 5. Modelo da mão utilizada neste projeto com eixos de rotação visíveis. Este modelo foi obtido do projeto de Panteleris, Oikonomidis e Argyros[11].

Roussel e Vogel[9], chamado de *Filtro de 1 Euro*. Este filtro foi utilizado por ser mais simples de implementar e definir os parâmetros que o Filtro de Kalman[10], além de ter resolvido o problema inicial de detecção de garra.

5) Obtendo pontos da mão em 3d: Neste ponto a aplicação tem, em relação à região de interesse, os pontos em duas dimensões das juntas. A partir destes e a posição da região de interesse em relação à imagem recebida é possível calcular a posição dos pontos em relação à imagem inteira.

Para obter os pontos em três dimensões em relação à câmera, foi-se utilizado o método descrito em Panteleris, Oikonomidis e Argyros[11], o qual define este problema como um problema de cinemática inversa.

6) Modelo da mão: O modelo da mão utilizado possui 26 graus de liberdade, sendo: 3 da posição do pulso; 3 da rotação do pulso, no qual cada grau de liberdade tem valores entre -180 e 180 graus; 2 para cada junta proximal, a qual pode girar entre 0 e 90 graus em torno do eixo x e entre -30 e 30 graus em torno do eixo z; 1 para cada uma das 10 juntas restantes, as quais podem girar entre 0 e 90 graus em torno do eixo x - os eixos de rotação são ilustrados na figura 5.

Assim como em Panteleris, Oikonomidis e Argyros[11], os 26 graus de liberdade são representados por 27 variáveis, uma vez que o pulso é representado por um quatérnio, em vez de 3 ângulos, a fim de evitar *Gimbal Lock*.

Para fazer a cinemática direta descrita na próxima subsessão, as informações que foram extraídas do modelo são:

- Os vetores unitários que representam a direção para todas as juntas ligadas, de tal forma que se forme uma arvore com o pulso como raiz e as pontas do dedo como folhas (o modulo dos vetores que ligam cada uma das juntas é calculado nos primeiros 30 quadros, considerando que a mão está estendida, como em Mueller et al.[8]).
- Os vetores unitários que representam os eixos de rotação de cada junta.

7) Cinemática Direta e Projeção dos Pontos: Para resolver o problema de cinemática inversa com o método proposto, primeiro é necessário resolver o problema de cinemática direta [12], ou seja, dado a posição do pulso e os ângulos de rotação quais são os pontos-chave em três dimensões e, indo além da cinemática direta, qual a projeção destes pontos na imagem.

A posição do pulso é dada na entrada. A posição das demais juntas pode ser calculada para cada junta J_i de acordo com a seguintes equações, considerando que o resultado do produto entre um quatérnio e um vetor é o vetor rotacionado nos ângulos representados pelo quatérnio:

$$Rot_{J_i}(X) = Rot_{P_i}(X) * RotLocal_{J_i}$$

$$Pos_{J_i}(X) = Pos_{P_i}(X) + Rot_{P_i}(X) * (Pos_{0_{P_i}} - Pos_{0_{J_i}})$$

Sendo:

- P_i é a junta mais próxima da raiz (pulso) na arvore que está diretamente ligada com J_i;
- $Pos_{0_{J_i}}$ é o vetor que representa a posição da junta i quando nenhuma rotação é aplicada às juntas.
- $RotLocal_{J_i}$ é um quatérnio que representa as rotações locais da junta i;
- X é uma variável de 27 dimensões que representa a posição do pulso e as rotações do pulso e das juntas, conforme descrito anteriormente;
- $Rot_{J_i}(X)$ é a um quatérnio que representa as rotações globais da junta i, isto é, considerando a rotação das demais juntas;
- $Pos_{J_i}(X)$ é a posição final da junta i;

Com isto, obtém-se os pontos-chave em três dimensões. Para obter os pontos projetados na imagem é necessário fazer a projeção destes, usando as equações de projeção da câmera [13].

$$v_1 = (-p.x/p.z, -p.y/p.z)$$

$$r = |v_1|$$

$$k = 1 + k_1 * r^2 + k_2 * r^4 + k_3 * r^6$$

$$v_2.x = v_1.x * k + 2 * p_1 * v_1.x * v_1.y + p_2 * (r^2 + 2 * v_1.x^2)$$

$$v_2.y = v_1.y * k + 2 * p_2 * v_1.x * v_1.y + p_1 * (r^2 + 2 * v_1.y^2))$$

$$PP = (fx * v_2.x, fy * v_2.y)$$

Sendo k_1, k_2 e k_3 os coeficientes de distorção radial, p_1 e p_2 os componentes de distorção tangencial e PP o ponto projetado.

8) Cinemática Inversa: Considere a seguinte função $F(X)$, similar à definida em Panteleris, Oikonomidis e Argyros[11]:

$$F(X) = \sum(|P_i - PP_i(X)| * c_i^3 * peso_i)^2$$

No qual:

- P_i é o ponto da junta i retornado pela rede neural;
- c_i é a confiança da previsão da junta i pela rede neural;
- $peso_i$ é um peso definido para a i-ésima junta;
- X é uma variável de 27 dimensões que representa a posição do pulso e as rotações do pulso e das juntas, conforme descrito anteriormente;
- $PP_i(X)$ é o ponto resultante da cinemática direta e projeção da junta i - este ponto é dado em função de X.

Desta forma, a pose X que gerou a mão da imagem enviada pelo cliente minimiza F(X), uma vez que esta função é miníma quando $|P_i - PP_i(X)|$ é zero para cada i, ou seja, a projeção de cada junta está na mesma posição identificada pela rede neural na imagem.

Dada a forma da função, uma aproximação para o mínimo pode ser encontrada usando o método de *Levenberg-Marquardt* [14], o qual é implementado pela biblioteca utilizada neste trabalho, *ceres-solver*.

Com isto, o servidor pode retornar para o cliente a posição do pulso todas as rotações que minimizam a função e, adicionalmente, os valores dos pontos-chave das juntas em três dimensões, obtidos após realizar a cinemática direta em X.

IV. Resultados e Discussões

Os testes realizados no desenvolvimento desse trabalho alcançaram localmente 5.37 quadros por segundo e 4.20 quadros por segundo no celular. A cena continua atualizando a 30 quadros por segundo, apesar da pose da mão não atualizar sempre. Desta forma, a imersão não é severamente afetada. O teste de *pick and place* no ambiente, proposto no inicio deste trabalho, foi realizado com sucesso.

O tempo usado para fazer a codificação e decodificação com o algoritmo *VP8* que é somado ao tempo total mas, conforme é notado na tabela I, este tempo é compensado durante a transferência em conexões não locais, principalmente no caso do uso da conexão *Wi-Fi*.

Na tabela I também pode ser notado que o tempo de transferência é responsável por no mínimo 68.62 ms, em testes não locais, o qual por si só faz com que a quantidade de quadros por segundo seja no máximo 14.57. Ainda nesta tabela, pode-se notar que o tempo de processamento do servidor diminui em testes não locais, possivelmente por não fazer os processamentos do cliente também.

	Duração (ms)		
Conexão	**Cliente**	**Servidor**	**Transferência**
Local	196.29	109.68	28.52
Local - S/ *VP8*	187.34	108.03	41.54
USB 3.0	241.75	92.97	68.62
USB 3.0 - S/ *VP8*	269.06	97.41	110.52
WiFi	238.35	91.73	80.29
WiFi - S/ *VP8*	316.42	97.03	158.74

Tabela I
TEMPO DE PROCESSAMENTO MÉDIO PARA CADA QUADRO POR TIPO DE CONEXÃO E COM E SEM O USO DE *VP8*. DADOS OBTIDOS USANDO COMO CLIENTE UM CELULAR ONE PLUS 7, O QUAL POSSUI UM *Chipset Snapdragon 855 Qualcomm SDM855*, UM PROCESSADOR *1x 2.84 GHz Kryo 485 + 3x 2.42 GHz Kryo 485 + 4x 1.8 GHz Kryo 485* E UMA *GPU Adreno 640*. E UM NOTEBOOK *HP OMEN - 15-dc0030np* COM UM PROCESSADOR *Intel Core i7-8750H* E UMA PLACA DE VÍDEO *NVIDIA GeForce GTX 1050 Ti*.

V. Conclusão

O problema de *Hand-Tracking* por si só é um grande desafio tecnológico contemporâneo. Integrar essa tecnologia para ambientes de realidade virtual possibilita a abstração deste problema, o que facilita obter melhor imersão nestes ambientes.

Como resultado deste trabalho, foi possível criar uma ferramenta *open-source* capaz de abstrair o conhecimento necessário de visão computacional para realizar o *hand-tracking*, e assim apenas fornecer a informação das juntas da mão para a aplicação VR, tudo isto com uma solução de baixo custo.

É importante mencionar também que com maior capacidade de processamento é possível alcançar uma taxa maior de quadros por segundo. Além disto há a possibilidade melhora usando a mesma Rede Neural utilizada por Zhang et al.[15], adicionando, inclusive, a perspectiva de remover a necessidade de um servidor.

Referencias

1 SHERMAN, W. R.; CRAIG, A. B. *Understanding virtual reality: Interface, application, and design*. Cambridge, MA: Morgan Kaufmann, 2018. 582 p.

2 GOLDSTONE, W. *Unity game development essentials*. Birmingham, UK: Packt Publishing Ltd, 2009.

3 MISHRA, P.; SHRAWANKAR, U. Comparison between famous game engines and eminent games. *International Journal of Interactive Multimedia & Artificial Intelligence*, v. 4, n. 1, 2016.

4 Sharrab, Y. O.; Sarhan, N. J. Detailed comparative analysis of vp8 and h.264. In: IEEE. *2012 IEEE International Symposium on Multimedia*. Irvine, CA, 2012. p. 133–140.

5 Chen, Y.; Murherjee, D.; Han, J.; Grange, A.; Xu, Y.; Liu, Z.; Parker, S.; Chen, C.; Su, H.; Joshi, U.; Chiang, C.; Wang, Y.; Wilkins, P.; Bankoski, J.; Trudeau, L.; Egge, N.; Valin, J.; Davies, T.; Midtskogen, S.; Norkin, A.; de Rivaz, P. An overview of core coding tools in the av1 video codec. In: IEEE. *2018 Picture Coding Symposium (PCS)*. San Francisco, CA: arXiv, 2018. p. 41–45.

6 REDMON, J.; FARHADI, A. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

7 CAO, Z.; SIMON, T.; WEI, S.-E.; SHEIKH, Y. Realtime multi-person 2d pose estimation using part affinity fields. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. San Franscisco, CA: IEEE, 2017. p. 7291–7299.

8 MUELLER, F.; BERNARD, F.; SOTNYCHENKO, O.; MEHTA, D.; SRIDHAR, S.; CASAS, D.; THEOBALT, C. Ganerated hands for real-time 3d hand tracking from monocular rgb. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. San Francisco, CA: IEEE, 2018. p. 49–59.

9 CASIEZ, G.; ROUSSEL, N.; VOGEL, D. 1€ filter: a simple speed-based low-pass filter for noisy input in interactive systems. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. New York, NY: ACM, 2012. p. 2527–2530.

10 KALMAN, R. E. A new approach to linear filtering and prediction problems. 1960.

11 PANTELERIS, P.; OIKONOMIDIS, I.; ARGYROS, A. Using a single rgb frame for real time 3d hand pose estimation in the wild. In: IEEE. *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. Lake Tahoe, NV/CA, 2018. p. 436–445.

12 KUCUK, S.; BINGUL, Z. *Robot kinematics: Forward and inverse kinematics*. London, UK: INTECH Open Access Publisher, 2006.

13 CALIBRATION, C. 3d reconstruction, opencv 2.4. 13.6 documentation. *Opencv dev team*, 2018.

14 MORÉ, J. J. The levenberg-marquardt algorithm: implementation and theory. In: *Numerical analysis*. Berlin, Heidelberg, DE: Springer, 1978. p. 105–116.

15 ZHANG, F.; BAZAREVSKY, V.; VAKUNOV, A.; TKACHENKA, A.; SUNG, G.; CHANG, C.-L.; GRUNDMANN, M. Mediapipe hands: On-device real-time hand tracking. *arXiv preprint arXiv:2006.10214*, 2020.