Comunicação entre dispositivos e aplicações Framework de interface

Luiz Oliveira, Carla Rocha, Cristiano Miosso
Universidade de Brasília - Faculdade do Gama
LART – Laboratório de Pesquisa em Arte e TecnoCiência
Qd 06 IND 1440 SN - Setor Industrial - CEP: 72445-060 - Gama – DF

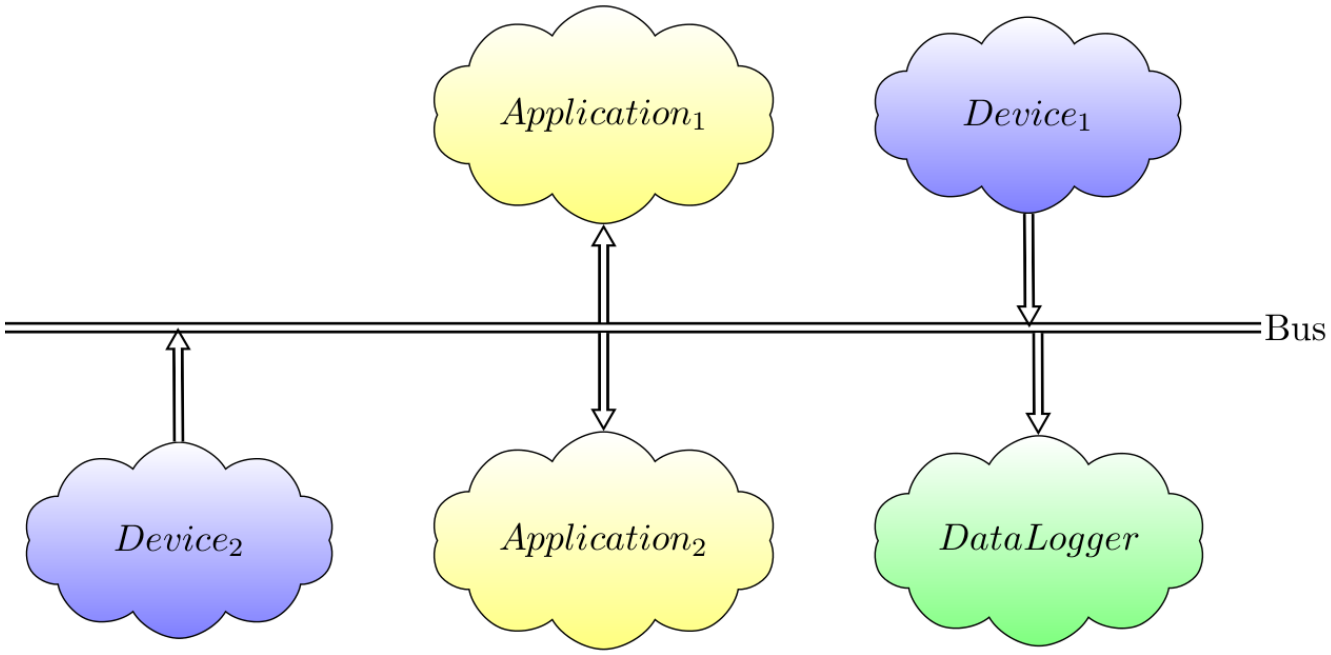

Figura 1: *Visão Geral - Compartilhamento de informações entre vários dispositivos, uma aplicação principal e um datalogger*

Resumo

O mercado de jogos vem evoluindo no intuito da imersão do jogador com a realidade virtual proposta nas aplicações. A cada dia novos tipos de entradas são considerados para um aprofundamento nesta imersão, abrindo espaço para sensores fisiológicos, câmeras e afins. Há uma necessidade crescente do uso de diversos dispositivos de entrada em um mesmo jogo, ou da possibilidade de reconfiguração e seleção simples do dispositivo de entrada desejado. Essa necessidade é mais evidente em jogos e sistemas de realidade virtual pervasivos. Este trabalho apresenta um framework que possibilita a integração entre uma grande quantidade de input devices para jogos e sistemas de realidade virtual, possibilitando que haja uma comunicação direta entre os devices a fim de oferecer uma interação mais natural e intuitiva do usuário com a plataforma, apresentando-se assim como uma possível solução para sistemas de realidade virtual pervasivos. Para validar o conceito, são apresentados alguns resultados obtidos com o uso do framework para em um cenário de sistema enativo no qual a interação ocorre dentro de uma CAVE, e sinais fisiológico do usuário controlam o estado do sistema.

Author's Contact:

{ziuloliveira, rocha.carla, cristiano.jacques}@gmail.com

1 Introdução

Diversas ferramentas build-in são utilizadas atualmente para a produção de aplicações. Um bom exemplo é o uso da comum biblioteca GLUT[Kilgard 1996] para o gerenciamento de entradas e controle de janelas. Situações no qual há diversos dispositivos de entrada, como, por exemplo, interface de toque e sensores de posição, em que ambos são gatilhos para uma ação no jogo, normalmente são pontos críticos para a aplicação. A criticalidade é devido tanto ao acesso paralelo/concorrente ao hardware de interface quanto quanto ao processamento concorrente dessas entradas. Problemas semelhantes são encontrados em aplicações com múltiplos-usuários e em realidades virtuais pervasivas, onde o módulo da aplicação de decisões sofre uma grande carga de dados oriundos dos dispositivos de entrada. Em jogos pervasivos, no qual o mundo físico é parte integrante do jogo [Luis Valente 2013], o sensoriamento do ambiente, do estado do usuário, e extração de eventos a partir desses dados são requisitos indispensáveis para gerar imersão no jogo.

O framework apresentado neste trabalho tem como objetivo permitir a modularização da processo de tratamento dos sinais de dispositivos de entrada. O framework usa conceitos de computação ubiquá no projeto de uma rede para a aquisição de dados dos sensores de entrada e leitura dos sinais pelo laço principal do jogo. Assim, a adição/remoção de dispositivos de entrada em um dado jogo pode ser realizada de forma transparente e semi-automatica, sem necessariamente ter necessidade de fazer alterações no laço principal do jogo. O framework é inspirado no protocolo de comunicação CAN[Bosch 2012], padrão de integração de sensores na indústria automobilística e aeronáutica. Neste protocolo, todos os sensores podem interagir com qualquer nó da rede e com o bloco principal, normalmente a CPU de um veiculo ou equipamento.

Um dos motivos desse protocolo ter se tornado padrão é a simplicidade em se adicionar ou retirar um novo sensor na rede. Apresenta uma interface de conexão padrão para cada nó adicionado a rede, mesmo que um sensor deixe de funcionar durante algum processo de comunicação, a rede não é comprometida, permitindo assim o acesso aos demais sensores na rede.

O modelo adotado faz uso de uma rede interna de sokckets para compartilhamento de informações. Por ser uma ferramenta nativa

dos principais sistemas operacionais, a alternativa não se limita a linguagem utilizada para implementação. A limitação da quantidade de entradas esta ligada a quantidade de memoria local e ao desempenho do processador com múltipos processos em segundo plano.

A rede possui um servidor que ecoa qualquer mensagem recebida a todos os nós conectados a ele, independente da mensagem ter chegado completa ou não. Cada nó conectado ao servidor tem a acessibilidade de ler ou escrever para o servidor. Os nós que escrevem são naturalmente os dispositivos de entrada (sensores, *joysticks*, etc), já os nós que fazem apenas as leituras são aplicações locais como jogos, *loggers*, monitores de estados, entre outros.

Os dispositivos dedicados à leitura processam os dados de acordo com os requisitos específicos da aplicação, fazendo validações, mantendo ou não um histórico da entradas. Devido a padronização das mensagens enviadas, as aplicações podem selecionar se a informação recebida é ou não válida para seu bloco lógico, descartando as informações não pertinentes e se mantendo sensível apenas aos novos dados relevantes.

A primeira contribuição deste trabalho é a proposta de uma arquitetura de software, no qual se apresenta um framework para a integração de dispositivos de entrada e aplicativos. São apresentados os detalhes técnicos dos diversos módulos funcionais do framework proposto, assim como especificidades da implementação do mesmo. A segunda contribuição é a validação do framework aplicado à um sistema de realidade virtual imersivo em uma CAVE. O experimento mostra a vantagem do uso do framework para a reconfiguração dos dispositivos de entrada (adição/remoção de dispositivos de entrada) e também para o cenário de varias aplicações usarem os mesmos dispositivos de entradas simultaneamente.

Este artigo é organizado em cinco seções. A segunda seção apresenta uma visão geral do framework, enquanto a terceira seção expõe as especificações do framework. A implementação é detalhada na seção quatro, assim como o caso de uso desenvolvido. A seção cinco apresenta as principais conclusões do trabalho e algumas melhorias planejadas.

2 Proposta

O Framework proposto atua como uma camada de interface entre os possíveis canais de entradas e o processo de tomada de decisões na aplicação principal. Um jogo, no padrão arquitetural clássico, normalmente possuiu uma camada responsável por coletar todas as entradas recebidas e coloca-las em uma fila de espera[Kilgard 1996]. Porém, essa mesma aplicação faz a filtragem de toda a informação que é utilizada no contexto do jogo/aplicação para que em sequência o estado do jogo seja atualizada. Quando o software passa a ser evolutivo e novos dispositivos de entrada são adicionados, a entrada de dados fica sujeita a duplicidade de entradas ou falso-positivos. Nesse cenário, o desenhista de arquitetura deve reestruturar o sistema de software de forma a manter a interface de comunicação entre aplicação e dispositivos de entrada em camadas claramente distintas.

O uso de Toolkits como o GLUT é uma alternativa interessante quando se usa entradas padrões como teclados, mouse e até mesmo joysticks, mas passa a ser trabalhoso quando a aplicação tenta fazer de uso de um dispositivo de entrada que ainda não esta mapeado com o toolkit. O novo mapeamento, proposto no framework, tende a ficar fora da camada do toolkit, tornando-se um risco para a estrutura da arquitetura.

O framework proposto possui uma arquitetura que permite uma camada de disponibilização de dados não apenas entre o dispositivo e a aplicação, mas abre possibilidade para a comunicação direta entre dispositivos, possibilitando a tomada de decisões de saída antes mesmo da aplicação ser notificada da variação de estado. Sem o framework, é o aplicativo que gerencia os dispositivos de entrada, junto com os dispositivos de saída e a tomada de decisões, dificultando não somente a manutenção do sistema, como a adição novos dispositivos de entrada. Já um jogo que utilize o framework proposto, é criado um módulo a parte para o gerenciamento das entradas.

3 Adaptação técnica

Para a implementação do framework foi definido a criação de uma rede UDP, inicialmente local, onde os dados são enviados, sem a espera de algum retorno ou confirmação. A arquitetura do framework é composta por um servidor rodando em *background* para o redirecionamento das informações, e um conjunto de clientes, formado pelos os dispositivos de entrada, como agentes de escrita e leitura, enquanto a aplicação principal - seja ela um jogo ou um simulador - atue na rede apenas com o intuito de leitura dos dados. Por ser o agente interessado nos dados, espera-se que o controle do servidor seja efetuado pela aplicação principal. O diagrama simplificado do sistema pode ser observado na Figura 2.

Criado o servidor, todos os clientes que se conectam a este servidor possuem tarefas especificas. Os dispositivos de entrada tem como objetivo apenas escrever no servidor o seu valor atual junto com seu identificador único, ignorando qualquer mensagem enviada do servidor para o dispositivo. Os jogos que utilizam dados dos dispositivos de entrada fazem então a leitura dos valores no servidor, evitando assim o congestionamento de dados na rede.

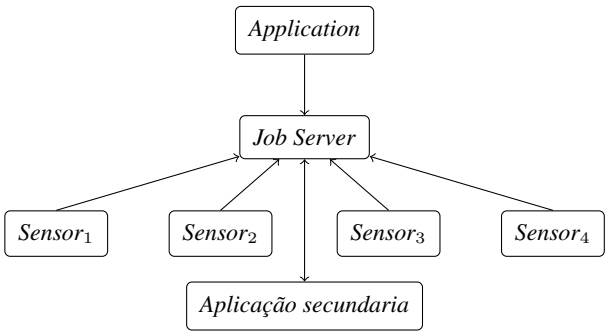

Figura 2: *Diagrama simplificado do sistema*

Após o processo de tratamento de erros, cada aplicação processa os dados de entrada de acordo com os requisitos da mesma. A taxa de aquisição de dados pode ser definida pela própria aplicação. Aplicações que não há restrição quanto à taxa de aquisição, aceitando rates de até 30 Hz de amostragem, podem configurar o framework para manter os valores em um vetor, utilizando-se dos identificadores de cada servidor para substituir os valores antigos lidos do servidor. Caso este processo esteja em uma tarefa da aplicação, sempre que requisitado um valor, será fornecido apenas o último valor recebido do sensor, salvo seja implementada uma fila de mensagem no driver do sensor.

Recomenda-se manter uma lista de valores recebidos no servidor ocorridos no intervalo entre cada momento de procura por novas entradas na aplicação. A frequência de taxa de amostragem para este modelo é bastante volátil, onde o hardware que mantém a aplicação e a quantidade de clientes conectados na rede irão influenciar a taxa máxima de amostragem possível. A implementação de uma fila de mensagem é uma opção fortemente recomendada para evitar a possibilidade de perca de valores recebidos pelo sensor no servidor durante o período de latência do servidor, minimizando assim a perca de dados.

4 Implementação

4.1 Aplicação vista no contexto tradicional

Como explicado na sessão 3, inicialmente deve ser criado um servidor com suporte a broadcast, para que cada cliente possa ter disponível os dados enviados ao servidor pelos dispositivos de entrada:

Listing 1: *Simple Broadcast Server*

```
import socket

def broadcast(sock, message):
    for client in connections_lists:
        if client != server_socket and client != sock:
            client.send(message)
```

```
TIMEOUT_TIME = 3
server_socket = socket.socket(socket.AF_INET, socket.↩
    SOCK_STREAM)
server_socket.setsockopt(socket.SOL_SOCKET,socket.↩
    SO_REUSEADDR,1)
server_socket.bind(ip, port)
server_socket.listen(TIMEOUT_TIME)
connections_lists.append(server_socket)

while(True):
    read_sockets, write_sockets, error_sockets = select↩
        .select(
                        connections_lists, [], [], 0.5)
    for sock in read_sockets:
        # New connection
        if sock == server_socket:
            sockfd, addr = server_socket.accept()
            connections_lists.append(sockfd)
        # Data recieved from client, process it
        else:
            data = sock.recv(1024)
            broadcast(sock, data)
```

Neste exemplo de servidor apresentado acima, ao receber uma mensagem de algum cliente, ela é redirecionada para todos os clientes já conectados ao servidor. Essa ação permite a um cliente receber qualquer mensagem que chegue ao servidor. Porém, do ponto de vista dos clientes deste servidor, não existe restrição para que estes façam leitura de tudo o que o servidor ecoa, permitindo assim que os sensores apenas "bombardeiem" o servidor com suas próprias informações.

Listing 2: *Simple Client Example*

```
import socket

TIMEOUT_TIME = 3
client_socket = socket.socket(socket.AF_INET, socket.↩
    SOCK_STREAM)
client_socket.connect(ip, port)
while(True):
    message = new_data()
    client_socket.send(message)
```

No exemplo acima é criado uma simples interface de um dispositivo, onde as informações do dispositivos são adquiridas no método *new_data* e publicado no servidor.

4.2 Aplicação vista no contexto de múltiplos processos

O framework pode ser implementado também com a vista de múltiplos processos e threads ainda para o controle de comunicação entre os dispositivos e aplicações.

Listing 3: *Multiprocess Client Example*

```
    from multiprocessing import Pool
    from Broadcast.server import Server
    from Broadcast.client import Client
    from Sensor.base import Sensor

def run_sensor(sensor):
    """ Generic action to each sensor """

    sensor_stream = Client()
    sleep(0.1)
    sensor.enable()
    while sensor.live:
        # write on server
        sensor_stream.write(sensor.read()) )
        sleep(0.1)

def main():
    """ Main method """
    server = Server()
    server.start()

    # Define the addresses of the sensors
    # sensors= {}

    _pool = Pool(processes=len(sensors))

    for i in sensors_address:
        sensors[i] = Sensor(port=0, address=↩
            sensors_address[i], mode=00)

    result = _pool.map(run_sensor, sensors.items())

    sleep(0.5)
    # stop server
    server.stop()
    # wait it stop
    server.join()
```

No exemplo acima, a aplicação é criada de forma que seja criado um processo dedicado para cada dispositivo. Uma thread é iniciada para ficar dedicada como servidor que servirá para a comunicação entre os dispositivos. É criado então um processo filho para cada sensor que se conectará com o framework. Cada processo filho possui uma thread dedicada para a comunicação com o servidor, enquanto o processo filho fica vinculado no ciclo de leitura e escrita no servidor, com um tempo de espera objetivado para a liberação do core e evitar a sobrecarga do servidor.

A configuração do dispositivo de entrada deve ser implementada de forma a ter uma interface de comunicação padrão, facilitando assim o modelo de divisão de tarefas entre processos. No exemplo apresentado acima para o uso com diversos dispositivos de comunicação I^2C, uma lista de sensores é criada contendo o endereço e o modo de funcionamento, sendo que os sensores possuem como método comum as chamada de habilitar e ler dado.

Recomenda-se manter uma lista de valores recebidos no servidor ocorridos no intervalo entre cada momento de procura por novas entradas na aplicação. A frequência de taxa de amostragem para este modelo é bastante volátil, onde o hardware que mantém a aplicação e a quantidade de clientes conectados na rede irão influenciar a taxa máxima de amostragem possível. A implementação de uma fila de mensagem é uma opção fortemente recomendada para evitar a possibilidade de perca de valores recebidos pelo sensor no servidor durante o período de latência do servidor, minimizando assim a perca de dados.

Esta visualização da aplicação permite que diversas interfaces sejam monitoradas em períodos bem definidos, além de oferecer um possível ganho de performance ao estar trabalhando com múltiplos processos sobre distintos sistemas operacionais[Nelson Filho et al.] ou até mesmo sobre GPUs[Zamith et al. 2013], visto que podem ser tratados como apenas mais um dispositivo a receber ou fornecer dados.

Em situações na qual um jogo possui inúmeras UI na tela além da recepção de dados via botões ou sensores de movimentos e câmeras eventualmente cria a situação de concorrência na aplicação, visto que uma determinada ação ofertada na UI pode ser ativada simultaneamente com um determinado valor de um sensor. Com o uso do framework, a UI não ficaria sensível apenas a entrada do usuário, mas ao valor do sensor também, já que o modulo projetado para a leitura da UI pode ter acesso aos atuais valores do sensor pelo framework. Sem o uso do framework, a concorrência à qual a UI esta submetida poderia acarretar em valores imprecisos ou falsos verdadeiros.

4.3 Experimentos realizados com o auxilio do framework

Para validar o framework, foram testados algumas séries de sensores em aplicações já funcionais. Dentre as aplicações experimentadas, a mais visual dentre elas foi a integração com uma CAVE composta por três telas para projeção com uso de diversos sensores fisiológicos usados como dispositivos de entrada, tando para definir o estado do simulador, quanto para controlar variáveis da computação gráfica. A configuração completa da CAVE e as especificações técnicas dos sensores utilizados é descrita em [Domingues et al. 2014].

O experimento funciona de forma que três aplicações rodam em simultâneo, sendo uma tratada como master e responsável pela leitura dos inputs e gerenciamento dos feedback gráfico baseado nas entradas fisiológicas adquiridas com os sensores. As outras duas aplicações são tratadas como slave e atuam na renderização das demais telas laterais.

Configurado a CAVE como a aplicação principal do servidor, foram agregados alguns sensores biologicos, tais como pressão plantar[do Carmo dos Reis et al. 2010] e um sensor EMC (Electromyography sensor) para capturar a intensidade da respiração[Domingues et al. 2011]. Cada sensor foi adicionado como um nó (cliente) ao servidor e foi designado apenas envio de dados.

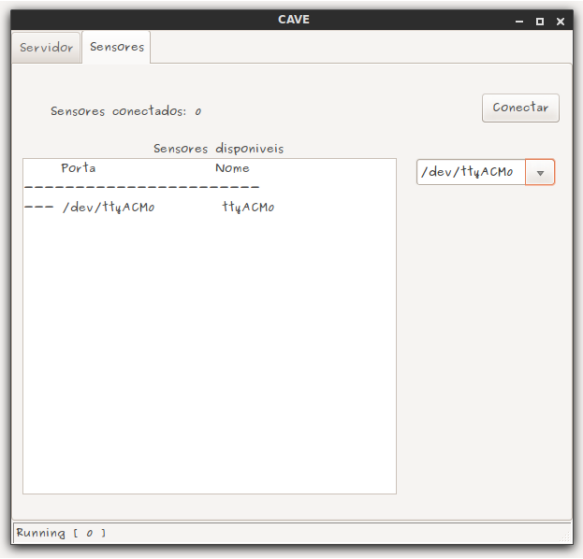

Figura 3: *Interface de controle do framework - Sensores*

Por ter uma quantidade de dados amostrados distintos, cada sensor foi identificado pela quantidade de amostras enviadas por mensagens. Na aplicação, foi implementado uma *thread* com a função específica de coletar os dados do servidor e tomar as ações devidas de acordo com os valores recebidos. Assim, ao pisar com força no chão a aplicação reagia emitindo sons de pisar, enquanto objetos flutuavam na tela acompanhando a respiração do usuário.

4.4 Resultados e discussões

Na ocasião do experimento com uma CAVE a aplicação respondia aos valores obtidos dos sensores, classificando assim o objetivo do framework como alcançado. Todavia algumas características precisam ser revisadas e otimizadas.

Por possuir uma thread dedicada para a aquisição dos dados, a integração do framework com as aplicações tem se mostrado incrivelmente efetiva e simples de se adicionar um novo nó, ou seja, um novo dispositivo. A maior dificuldade encontrada foi na criação de um socket UDP dentro da aplicação para comunicação com o servidor do framework. Feito este servidor, inserir ou retirar um novo dispositivo não aresenta nenhuma dificuldade, bastando apenas inceri-lo como um novo cliente.

As taxas de amostragens se mostravam bem elevadas, permitindo uma frequência de ate 62.5 Hz quando submetido a sistemas embarcados[1]. Todavia o uso da aplicação requer que todos os possíveis valores de entrada sejam inicializados na aplicação antes da verificação, para garantir a captura de valores reais significativos, visto que o framework não possui uma camada de validação dos dados recebidos, uma vez que determinada atividade descaracterizaria a objetividade do framework como uma interface que se porta de forma semelhante, independente do tipo de *device* conectado a ela.

[1]Raspbery Pi com Debian Wheezy embarcado

É importante observar que a solução é voltada para aplicações onde o utilizador mantenha o habito de verificar periodicamente os valores de entrada. Situação onde o utilizador entre em descanso para ser acordado por um sensor externo necessitariam de uma estrutura sensível a entrada de mensagens distintas no *buffer* de leitura, no caso, o cliente da aplicação. Já aplicações onde o dispositivo espera uma iniciativa do utilizador para enviar algum dado seria mais interessante para estruturas onde o utilizador planeje eventuais momentos de descanso, já que o servidor permite *full-duplex* por padrão, possibilitando que o utilizador envie a mensagem de requisição de dados ao sensor e espere a resposta dele pelo mesmo canal.

5 Conclusões e trabalhos futuros

A integração de múltiplos dispositivos de entrada em aplicações e jogos é uma tendência para aumentar a imersão de jogadores. Porém, a arquitetura de software padrão, no qual é acoplado à aplicação o processo de acesso e leitura de dados dos dispositivos, dificulta a manutenção e a evolução de aplicativos para a integração com novos dispositivos de entrada. O framework proposto modulariza o processo de acesso e tratamento dos sinais dos dispositivos de entrada e a aplicação. A arquitetura proposta usa uma rede UDP para acesso aos dispositivos e comunicação com a aplicação. A validação do framework foi realizada em um sistema de realidade virtual no qual os estados de visualização do sistema era controlado por cerca de oito dispositivos de entrada. Os testes realizados mostraram estabilidade do framework e também possibilidade para a adição de mais dispositivos de entrada.

A interface do framwork desenvolvida encontra-se atualmente em seu estado beta e já permite a identificação e leitura de sensores para o framework, sendo funcional para sistemas operacionais Linux e Mac OS.

Agradecimentos

Nós agradecemos pelo suporte financeiro da CAPES, CNPq, bem como da Universidade de Brasília – Faculdade Gama.

Referências

BOSCH, R. 2012. *CAN with Flexible Data-Rate*, April.

DO CARMO DOS REIS, M., DE S. RODRIGUES FLEURY ROSA, S., AND ROCHA, A. F. 2010. Desenvolvimento de uma palmilha para pé diabético com controle de pressão.

DOMINGUES, D., MIOSSO, C., PAREDES, A., AND ROCHA, A. 2011. Bioengineering and technoscience art: Electromyography and signal processing for synaesthesia—in-activity muscles and data visualization. In *Health Care Exchanges (PAHCE), 2011 Pan American*, IEEE, 245–245.

DOMINGUES, D., MIOSSO, C. J., RODRIGUES, S. F., AGUIAR, C. S. R., LUCENA, T. F., MIRANDA, M., ROCHA, A. F., AND RASKAR, R. 2014. Embodiments, visualizations, and immersion with enactive affective systems. In *IS&T/SPIE Electronic Imaging*, International Society for Optics and Photonics, 90120J–90120J.

KILGARD, M. J. 1996. *The OpenGL Utility Toolkit (GLUT) Programming Interface*. Silicon Graphics, Inc.

LUIS VALENTE, BRUNO FEIJÓ, J. C. L. 2013. Features and checklists to assist in pervasive mobile game development.

NELSON FILHO, D., BOTELHO, S. C., CARVALHO, J. T., MAFFEI, R. D. Q., OLIVEIRA, R. R., MORAES, F. C., RODRIGUES, T. F., SANTOS, E., DOS SANTOS, R. A., AND IAHNKE, S. L. H-env: An architecture to create intelligent hyper-environments using ubiquitous computing and virtual reality.

ZAMITH, M., VALENTE, L., JOSELLI, M., JUNIOR, J. S., CLUA, E., AND FEIJÓ, B. 2013. A game architecture based on multiple gpus with energy management.