# Adaptando e desenvolvendo jogos para uso com o Microsoft Kinect

Bruno Campagnolo de Paula

Instituto de Tecnologia do Paraná / (TECPAR) / Centro de Engenharia de Sistemas Inteligentes
Pontifícia Universidade Católica do Paraná

**Figura 1**: Exemplo de usuário interagindo com um jogo usando seu corpo.

## Abstract

We introduce in this tutorial a rapid prototyping tool that facilitates the full-body control of games and software. FAAST [FAAST 2011] (*Flexible Action and Articulated Skeleton Toolkit*) is this middleware that allows the integration with Microsoft Kinect. We also present some programming examples of Kinect control using the Processing programming language.

## Resumo

Este tutorial tem como objetivo principal apresentar uma ferramenta de prototipação rápida que facilita o controle de jogos e programas através do uso de gestos feitos pelo corpo e capturados por sensores de profundidade. O FAAST [FAAST 2011] (*Flexible Action and Articulated Skeleton Toolkit*) é este middleware que permite a integração, atualmente, com o Microsoft Kinect e similares. Também apresentaremos alguns exemplos de programação do Kinect em linguagem Processing.

**Palavras-chave**: Kinect, FAAST, Interface Natural

**Contato**:
campagnolo@tecpar.br

## 1. Introdução

Uma das principais características da última geração de videogames foi a revolução proporcionada por novas formas de interação com os jogos. O uso do controle do Wii (Wiimote) trouxe para a sala de estar uma maneira mais natural do jogador interagir com o jogo. Substitui-se o apertar de botões de um joystick pelo movimento de um controle simplificado associado a movimentos do corpo. Uma evolução natural desta tendência foi a criação do periférico Kinect, construído pela Microsoft para uso com o console XBOX 360. Este dispositivo elimina, inclusive, a necessidade de utilizar um controle físico. Constituído de um conjunto de câmeras e sistema de projeção, o Kinect retorna uma informação bastante exata de cor e da profundidade associada a cada ponto. O aparelho destaca-se também por devolver a posição (x, y e z) de um conjunto de pontos associados às juntas principais do corpo humano (cabeça, mãos, cotovelos, pernas, etc). É desejável que possamos aplicar o Kinect em nossos jogos e aplicativos.

Desenvolvido na University of Southern California com o objetivo de ser um framework extensível para aplicações de realidade virtual [Suma et al. 2011], na prática, o FAAST é um mapeador de movimentos do corpo em ações de dispositivos tradicionais de interação via software. Mas também possibilita o streaming da posição das articulações via rede através de um servidor VRPN (*Virtual Reality Peripheral Network*) [Taylor et al. 2001], sendo, portanto, adequado para aplicações em realidade virtual mais elaboradas que conversem com dispositivos compatíveis com este protocolo. Destaca-se que o usuário do FAAST não precisa conhecer nenhum tipo de linguagem de programação complexa para poder utilizar este programa. A idéia principal do software está no mapeamento de poses

específicas realizadas em frente à câmera em comandos de teclado e mouse. Desta forma, softwares e jogos não desenvolvidos para dar suporte à interação natural passam a ter suporte imediato, por exemplo, ao uso do Kinect. Mais importante do que este suporte, o qual também poderia ser feito com o Kit de Desenvolvimento Oficial da Microsoft (SDK do Kinect) ou com outras alternativas livres, está o fato de não haver necessidade de uso de programação. Abre, portanto, a possibilidade de uso para professores, projetistas e mesmo artistas.

Além disso, o FAAST também permite a configuração de parâmetros da interação sem necessariamente parar o programa e sua equipe promete o suporte a mais ações relacionadas ao corpo e a possibilidade de gravação e treino de gestos personalizados. Aconselhamos, no contexto deste tutorial, a aplicação do FAAST como uma ferramenta de prototipação inicial de aplicações para o Microsoft Kinect. Neste tutorial estaremos explorando a versão 0.08 do FAAST, lançada em 25 de Abril de 2011.

Iniciaremos este tutorial, porém, comentando sucintamente sobre o que são interfaces naturais e quais são as características do Microsoft Kinect enquanto dispositivo de interface natural. Mostraremos também quais são as principais alternativas de plataformas de programação para o Kinect. A maior parte do tutorial, entretanto, será dedicado ao FAAST e não exigirá conceitos de programação. Por fim, apresentamos uma evolução de tópico possível aos programadores e demais interessados, analisando códigos desenvolvidos na linguagem Processing e que agilizam a utilização de recursos mais avançados do Kinect.

Os exemplos deste tutorial, erratas, e mais material relacionado aos tópicos que serão discutidos estão disponíveis em **http://www.brunocampagnolo.com/tutorialkinect2011**.

## 2. Interfaces naturais

Uma interface natural para o usuário (NUI - *Natural User Interface*) é o próximo passo de evolução na maneira como um usuário interage com o computador. Nos primeiros sistemas computadorizados, o usuário limitava-se a interagir com os sistemas através da combinação de comandos limitados (CLI - *Command Line Interface*). Neste mecanismo de interação, caracterizado, por exemplo, pelas interfaces de *shell* dos sistemas operacionais, o sistema espera que o usuário forneça um comando, recebe o comando, o executa e devolve o resultado da ação executada. Este conceito, embora aparente estar desatualizado e inaplicável à exigência dos usuários atuais ainda está presente e disponível na maior parte dos sistemas.

Com o advento das interfaces gráficas (GUI - *Graphical User Interface*), o usuário passou a interagir com imagens ao invés de comandos textuais. As ações, neste caso, são concretizadas a partir da manipulação direta dos elementos gráficos. Por exemplo, o clique em uma miniatura específica de um arquivo permite a sua abertura em uma interface de edição. A popularização da linguagem visual da GUI foi um dos fatores predominantes na divulgação dos computadores fora do meio científico [Reimer 2005]. O estilo de interação WIMP (*Window, Icon, Menu e Pointing Device*) apresenta metáforas rapidamente compreensíveis por pessoas com pouca habilidade com computadores.

As interfaces naturais ao usuário (NUI - *Natural User Interface*) se referem a um estilo de interface caracterizado pela invisibilidade do controle ao usuário. Segundo Bill Buxton, uma interface é natural se explora as habilidades que o usuário adquiriu durante a vida ao interagir normalmente com o mundo [Buxton 2010]. Diferentemente dos paradigmas citados anteriormente, portanto, uma interface natural deve ser aprendida e utilizada rapidamente, beneficiando-se e adaptando-se a partir da atuação do corpo humano.

A vantagem do uso de interfaces naturais está na aplicação de habilidades simples e inatas ao ser humano e que podem ser adaptadas a diferentes tarefas sem muito esforço. O processo de aprendizagem é rápido pois pode ser alcançado, muitas vezes, apenas através da observação de outra pessoa demonstrando a habilidade uma vez ou duas [Blake 2011].

Diversos exemplos de interface podem ser referenciadas como naturais, por exemplo, o controle do Nintendo Wii (WiiMote), a Playstation Eye Toy Camera, o Genesis Activator e tapetes de dança (ver Figura 2). Citando, é claro, apenas dispositivos focados na área de jogos. Fora deste campo, destaca-se o trabalho dos pioneiros na área de Realidade Virtual como Myron Krueger nos anos 70 e 80 e seu sistema VideoPlace [Krueger et al. 1985]. Este sistema é uma instalação interativa na qual a posição e movimento das mãos, vistas por uma câmera de vídeo, determinam o comportamento de objetos na tela, incluindo criaturas animadas.

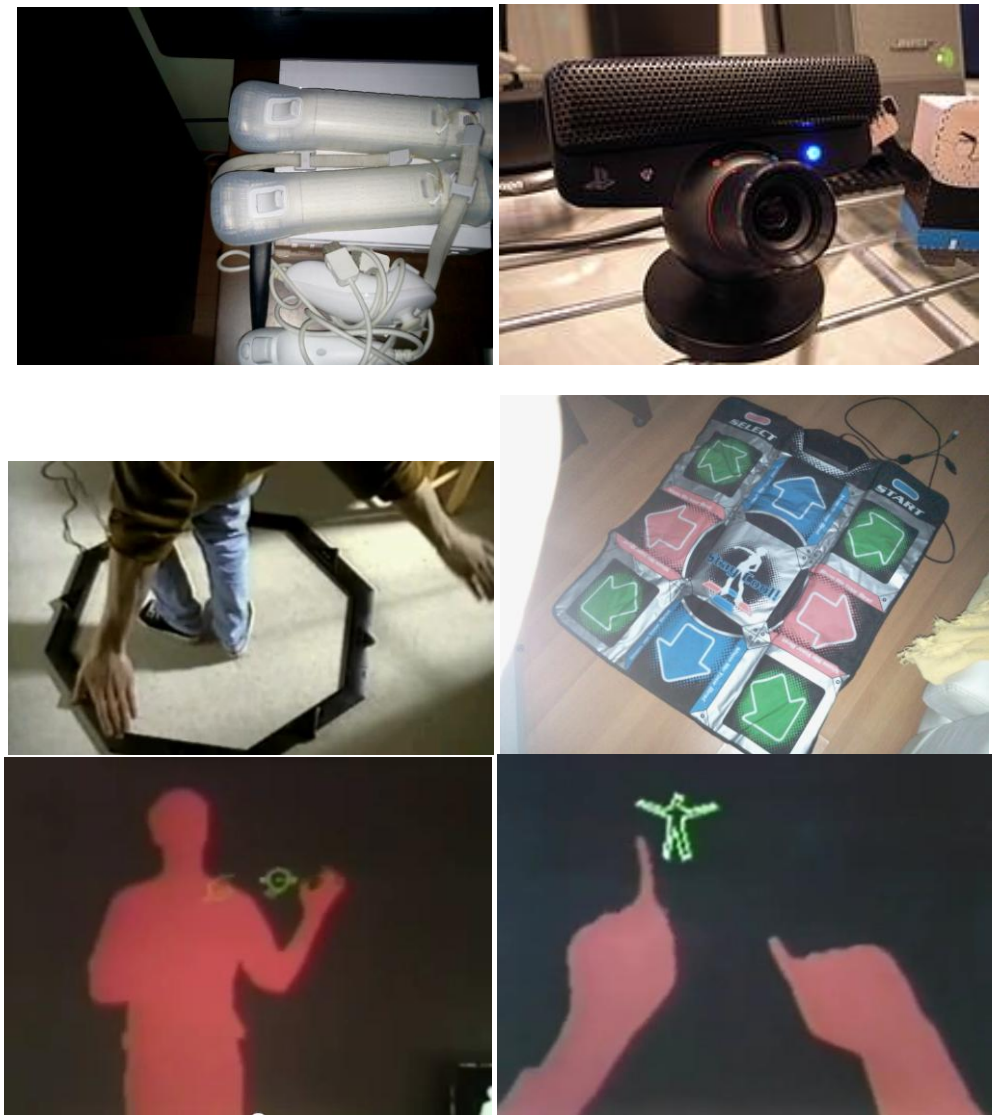


**Figura 2**: Exemplos de dispositivos de interface natural: WiiMote, Playstation Eye Toy, Sega Activator e Tapete de Dança. Abaixo, trabalhos pioneiros de Myron Krueger [Krueger 1988].

Neste tutorial, utilizaremos o dispositivo de interface natural Microsoft Kinect [Microsoft 2010]. Embora este aparelho tenha sido criado originalmente para o Microsoft XBOX, ele é facilmente integrável a um computador pessoal com sistemas Windows / Mac OS X ou Linux. É, sem dúvida, o aparelho de interface natural mais vendido e popular no mundo. Até março de 2011 cerca de 10 milhões de unidades foram vendidas [Microsoft 2011]. O Kinect destaca-se, principalmente, por sua performance em reconhecimento de gestos. Na próxima seção, discorreremos brevemente sobre os seus recursos e funcionamento básico.

## 3. Arquitetura do Kinect

A arquitetura básica do Kinect é formada por um projetor de luz infravermelha (invisível ao olho humano), uma câmera infravermelha, uma câmera RGB comum, um conjunto de microfones e um motor, conforme indica a Figura 3. Existem duas versões do dispositivo. A versão vendida junto com o console XBOX necessita de um cabo especial de alimentação quando usado no computador tradicional. A outra já vem com este cabo integrado. Sua interface de dados é via USB. Destaca-se, principalmente, que esta interface não está criptografada [Fried 2011], fato que facilitou desde o início sua utilização fora do XBOX.


**Figura 3**: Arquitetura básica do Kinect.

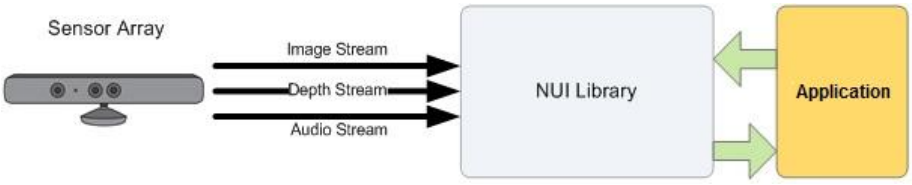


**Figura 4**: Resumo da arquitetura básica [Microsoft Research 2011].

O Kinect devolve, portanto, ao programador do dispositivo (ver esquema na Figura 4):

- *Image Stream* (imagens): cada pixel representando uma cor, resolução de 640x480 pixels em 30 frames por segundo (fps) ou 1280x1024 em um máximo de 15 fps. É possível obter tanto a imagem da câmera RGB quanto a da câmera infravermelha;
- *Depth Stream* (informação de profundidade): cada pixel indicando a distância do objeto em relação ao aparelho. O aparelho detecta cerca de 2000 níveis de sensibilidade e percebe objetos presentes de 1.2 a 3.5 metros à frente do aparelho. Além da informação de profundidade também é possível retornar com exatidão se o pixel faz parte do corpo de um ser humano. O Kinect consegue diferenciar até 6 corpos humanos em sua visada.
- *Audio Stream* (fluxo de audio): com um conjunto de 4 microfones e a anulação de ruído e eco, o Kinect permite a gravação de áudio e o reconhecimento da fala em inglês.

O principal destaque do dispositivo está na possibilidade de uso de seu sensor de profundidade em jogos e aplicativos, obtendo com bastante exatidão a distância de cada pixel de uma imagem em relação ao sensor. O funcionamento deste sensor não se baseia em princípio estereoscópico como nas câmeras de profundidade mais tradicionais. Na verdade, um padrão infravermelho é projetado e a deformação neste padrão é medida, permitindo a inferência da distância. A Figura 5 mostra exemplo do que é capturado pela câmera infra-vermelha do Kinect e os padrões que permitem o cálculo da profundidade.

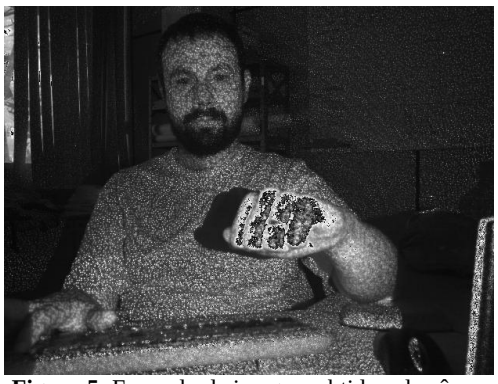

**Figura 5**: Exemplo da imagem obtida pela câmera infra-vermelha do Kinect.

Outro fator de sucesso do Kinect é seu aspecto híbrido, não se limitando a devolver a informação de profundidade sem nenhum tipo de tratamento. A cada pixel, portanto, também está associado um valor que indica se pertence ao corpo do ser humano. Assim, o aparelho permite a

diferenciação de até 6 pessoas em sua frente. E, dessas 6 pessoas, o Kinect também tem a capacidade de obter a informação de esqueleto de até 2 jogadores. A seguir, demonstra-se acima as articulações de um esqueleto reconhecido pelo Kinect.

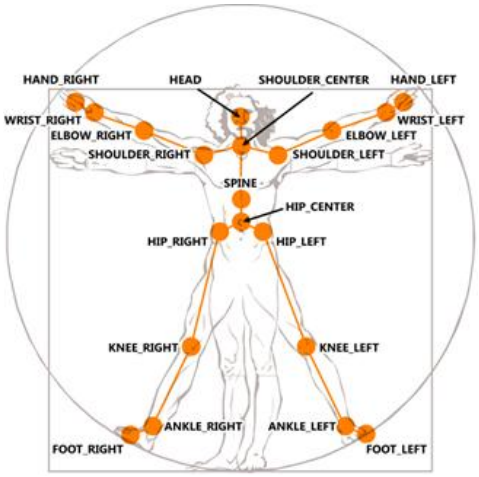


**Figura 6**: Articulações do esqueleto relativas ao corpo humano [Microsoft Research 2011].

Esta habilidade de reconhecimento de articulações é, inclusive, a principal contribuição do Kinect ao campo de Aprendizagem de Máquina, digna inclusive do prêmio MacRobert na área [Business Weekly 2011]. Assim sendo, também é possível obter a qualidade da captura e o dispositivo infere, por exemplo, quando uma junta está oclusa ou indica quando está com má qualidade. Para mais detalhes sobre o funcionamento, história e tecnologias envolvidas na criação do Kinect remete-se aos artigos [Queiroz 2010] e [Rowan 2010].

Todas estas informações, originalmente estavam disponíveis apenas para o XBOX. Na próxima seção comentaremos sobre as diferentes alternativas para desenvolvimento com o Kinect que surgiram imediatamente ao seu lançamento.

## 4. Desenvolvendo para o Kinect

Alguns dias depois do lançamento do Kinect em novembro já era possível desenvolver para o dispositivo devido à proposição de um concurso aberto à comunidade hacker. Uma iniciativa do produtor de kits eletrônicos Adafruit Industries foi lançada no dia de lançamento fornecendo US$ 1.000,00 à primeira pessoa que conseguisse executar um programa que interagisse com o Kinect em um computador. Após a resposta da Microsoft em não apoiar modificações não autorizadas do aparelho, em resposta, o prêmio aumentou para US$ 2.000,00 e posteriormente para US$ 3.000,00. Poucos dias depois, com o uso de um analisador de USB, o Kinect foi finalmente "hackeado" [Giles 2010]. Como resultado, surgiu a comunidade OpenKinect [OpenKinect 2011], que lançou um conjunto de drivers livres para o Kinect.

No mês seguinte, a PrimeSense [PrimeSense 2011], fabricante do sensor que serviu de base à Microsoft para criar o Kinect decidiu abrir o código de seu próprio driver e framework e disponibilizar também o módulo de rastreamento de esqueleto. Aliado a esta abertura, a PrimeSense decidiu criar a iniciativa OpenNI [OpenNI 2011], que é uma organização sem fins lucrativos para promover a interoperabilidade entre dispositivos de interação natural.

Tanto a OpenKinect e a OpenNI são abordagens abertas e gratuitas, sem nenhum tipo de restrição de uso ou licença demasiado restritiva e possuem versões para Windows, MacOS X e Linux. Existem wrappers que permitem a programação em linguagens como C, C++, Java, C# e Processing. Por fim, na metade de 2011, a própria Microsoft lançou sua própria plataforma de desenvolvimento [Microsoft Research 2011], voltada a computadores rodando o Windows 7. O SDK oficial possibilita que os desenvolvedores C++, C# ou Visual Basic acessem as informações das câmeras do sensor e o rastreamento do esqueleto, como as plataformas abertas, mas também dá o poder ao programador de criar aplicativos que usem o reconhecimento de fala. O SDK é limitado a aplicações não-comerciais, mas espera-se uma licença comercial nos próximos meses.

Tanto o OpenKinect, quanto o OpenNI e o SDK oficial exigem algum grau de programação para criar jogos para o Kinect. Neste tutorial, porém, apresentaremos na próxima seção uma alternativa: o FAAST, que permite o mapeamento de ações corporais em eventos de mouse e teclado. Esta ferramenta dá poder ao desenvolvedor para prototipar e adaptar jogos ao uso do Kinect. O FAAST é baseado na interface do OpenNI, sendo necessária, primeiramente, sua instalação.

## 5. Instalação do FAAST

Para usar o FAAST com o Kinect será necessária a instalação de diversos pré-requisitos. Deve-se destacar que cada um destes requisitos está em suas versões preliminares e instáveis. Desta forma, ocorrendo alguma instabilidade, tente instalar a versão anterior. Por segurança, no site deste tutorial, colocamos as versões exatas de cada um dos pré-requisitos.

Segue a lista de pré-requisitos, que devem ser instalados na ordem indicada. Importante: **instale a versão 32 bits** de cada um dos itens a seguir, independente de sua plataforma.

1) Instalação do **OpenNI Unstable Build**:
   a. Download provável em: http://www.openni.org/downloadfiles/opennimodules/openni-binaries/20-latest-unstable/)
   b. Última versão testada: OpenNI-Win32-1.3.2.3-Redist.msi;
2) Instalação do **PrimeSense NITE Unstable Build**: d
   a. Download provável em: http://www.openni.org/downloadfiles/opennimodules/openni-compliant-middleware-binaries/33-latest-unstable
   b. Última versão testada: NITE-Win32-1.4.1.2-Redist.msi;
3) Instalação dos **drivers para o sensor do Kinect**:
   a. Download provável em: https://github.com/avin2/SensorKinect/tree/unstable/Bin
   b. Última versão testada: SensorKinect-Win-OpenSource32-5.0.3.4.msi;
4) Fazer o download da **pasta de drivers do Kinect**
   a. download provável em: https://github.com/avin2/SensorKinect/tree/unstable/Platform/Win32/Driver.
   b. Última versão testada: 13967
5) Reiniciar o computador.
6) Conectar o Kinect na porta USB.
7) Se necessário pedido, selecionar os arquivos da pasta de drivers indicada no passo 4.
8) Realizar o download do FAAST (http://projects.ict.usc.edu/mxr/faast) e descompactá-lo em uma pasta qualquer.

Ao executar o programa FAAST.exe a tela a seguir será exibida. Esta tela apresenta todas as possibilidades de configuração possíveis, sem precisar alterar com nenhum tipo de configuração externa. Ao detectar o usuário, o FAAST fica aguardando a pose de calibração.

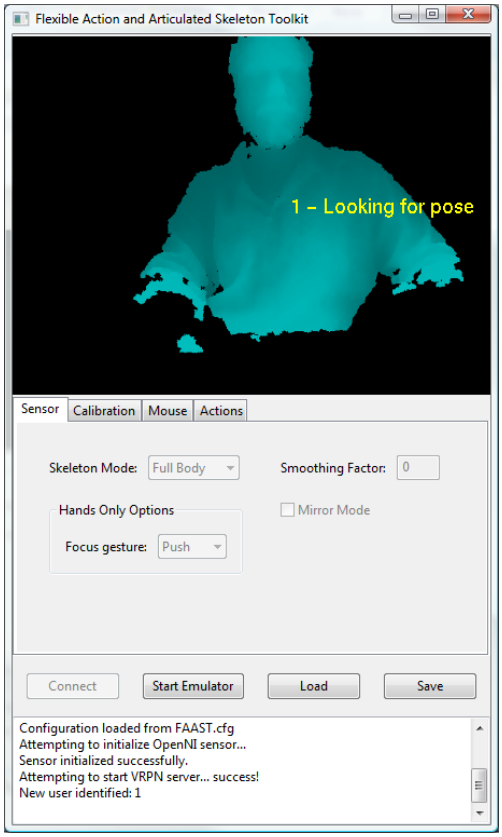


**Figura 7**: Tela inicial do FAAST e o programa aguardando a pose de calibração.

Ao abrir a ferramenta, é carregada a configuração padrão da ferramenta, armazenada no arquivo FAAST.cfg. Os botões *Load* e *Save* permitem o gerenciamento de configurações diferentes. Para testar a configuração padrão, clique em *Connect* e mantenha uma distância adequada do Kinect e faça a pose de *psi*, conforme indica a figura a seguir. O esqueleto será reconhecido (Figura 8). Na próxima seção, comentaremos sobre as principais opções de configuração do FAAST.

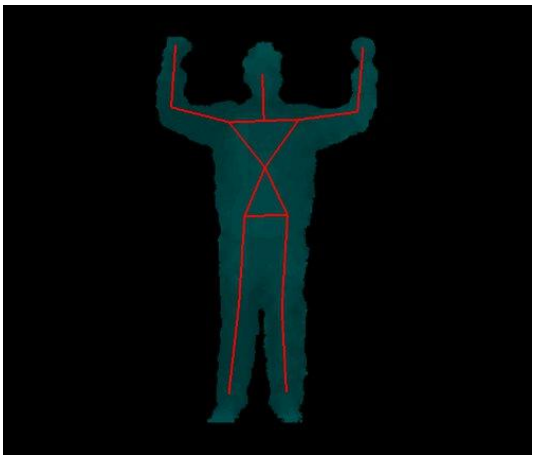

**Figura 8**: Pose Psi, necessária para calibração [FAAST 2011].

## 6. Configuração básica do FAAST

Cada software / jogo que se deseja controlar pode estar associado a uma configuração diferente. Existem quatro grupos de configuração: **Sensores**, **Calibração**, **Mouse** e **Ações**.

### 6.1 Configuração do Sensor

Em *Skeleton Mode*, pode-se configurar o quanto do esqueleto será considerado durante o reconhecimento. As opções disponíveis são:

- ***Full Body***: o corpo todo é considerado;
- ***Upper Body***: apenas a parte superior do corpo;
- ***Lower Body***: apenas a parte inferior;
- ***Hands Only***: apenas as mãos.

No caso da seleção *Hands Only*, é necessário também escolher um gesto de focalização (*Focus Gesture*). Este gesto habilitará o reconhecimento das ações. Também é possível inverter as articulações direita e esquerda através da caixa *Mirror Mode*. Outro fator importante a ser considerado, devido à precisão do Kinect é o fator de suavização (*Smoothing Factor*). Para diminuir a perturbação natural devido ao movimento do corpo, pode-se aplicar um valor entre 0 e 1. Cada situação deve ser testada e experimentada. Por exemplo, a equipe do FAAST sugere um valor entre 0.07 e 0.08 para implementar um controle preciso do mouse aliado ao parâmetro *Movement Threshold* com valor 2, que será explicado posteriormente.

### 6.2 Calibração

Para calibrar, o usuário deve segurar a pose de psi por alguns segundos. Após detectada, é exibida uma imagem similar a Figura 8. Sem esta etapa de calibração, não é possível interagir com o FAAST. Diferente do SDK Oficial do Kinect, sem a calibração o esqueleto não é detectado.

Nesta interface também é possível salvar na memória e em arquivos os dados de calibração de um certo usuário.

### 6.3 Configurações de mouse

O FAAST permite a associação da posição da mão esquerda ou mão direita à posição do mouse. Há dois tipos de controle de mouse: absoluto e relativo. No caso do controle absoluto a posição da mouse é determinada a partir da posição da mão dentro de um retângulo (*bounding rectangle*) delimitado pelos parâmetros *Left Bound*, *Right Bound*, *Bottom Bound* e *Top Bound*. Cada um destes parâmetros indica a distância entre a origem e, respectivamente, os lados esquerdo, direito, inferior e superior do retângulo que serve para movimentação do mouse. Quando a mão está fora do retângulo, o mouse fica nas extremidades da tela. O controle de mouse do tipo relativo, por sua vez, calcula a velocidade do mouse a partir da distância da mão escolhida até o centro do *bounding rectangle*.

O usuário pode também definir qual é a origem. Duas opções: centro de massa do corpo e articulação do ombro. Por fim, dois limiares importantes implementam o ajuste fino da velocidade e precisão do mouse. O *Forward Threshold* indica a distância relativa à origem que a mão deve se mover à frente para ativar o controle do mouse. O ajuste do *Movement Threshold*, por sua vez, representa o valor mínimo em pixels necessário para alterar a posição do mouse. Observa-se que alterando o *Movement Threshold* o movimento fica mais controlável.

No caso do movimento relativo, também é possível a determinação da velocidade máxima (*Speed*) que será alcançada ao se chegar nos limites do *bounding rectangle*. Cada jogo ou aplicativo a ser adaptado merece uma calibração de seus parâmetros. Por exemplo, para implementar um retângulo similar à Figura 9, foram aplicados os seguintes parâmetros:

- *Left Bound*: 5;
- *Right Bound*: 5;
- *Bottom Bound*: 5;
- *Top Bound*: 5.

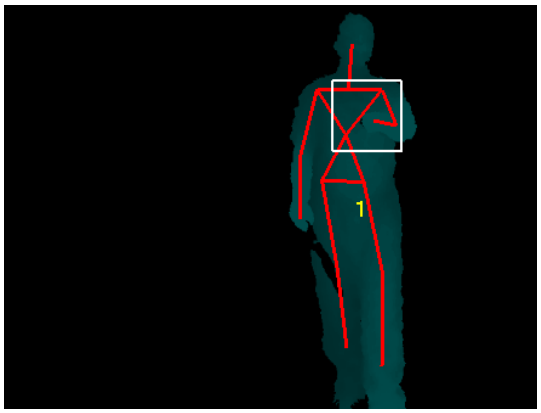


**Figura 9**: Em branco, o *bounding rectangle*, dentro do qual a posição da mão altera o movimento do mouse.

### 6.4. Mapeamento das ações em eventos

A aba Actions serve para mapeamento de uma ação do corpo em evento de teclado e mouse. A sintaxe para cada entrada é:

```
nome_ação limiar tipo_evento nome_evento
```

- **nome_ação**: O nome da pose ou gesto realizado pelo usuário;
- **limiar**: O limiar mínimo para ativação do evento. Este limiar varia para cada ação.
- **tipo_evento**: O tipo de evento virtual que será gerado;
- **nome_evento**: O evento específico que será gerado.

Por exemplo, para criar um jogo no qual a inclinação do corpo à esquerda cerca de 20 graus segurasse a tecla à esquerda e a inclinação do corpo à direita na mesma quantidade segurasse a tecla à direita e um pulo de cerca de 2 polegadas disparasse a tecla de espaço, o mapeamento necessário seria:

```
lean_left 20 key_hold left_arrow
lean_right 20 key_hold right_arrow
jump 2 key_press space
```

A tabela seguinte lista os valores possíveis para o nome da ação e seu valor de limiar, a partir do qual o evento indicado é disparado. Nesta tabela temos diversos tipos de ação, que podemos agrupar em:

- **Inclinação e rotação do corpo**:
  - lean_left,lean_right, lean_forwards, lean_backwards, turn_left, turn_right;
- **Posição dos braços**:
  - left_arm_forwards, left_arm_down, left_arm_up, left_arm_out, left_arm_across, right_arm_forwards, right_arm_down, right_arm_up, right_arm_out, right_arm_across.
- **Posição dos pés**:
  - left_foot_forwards, left_foot_sideways, left_foot_backwards, left_foot_up, right_foot_forwards, right_foot_sideways, right_foot_backwards, right_foot_up;
- **Movimentos específicos**:
  - jump, crouch, walk.

**Tabela 1**: Ações disponíveis no FAAST e seu significado.

| Nome da Ação **nome_da_ação** | Significado | Limiar **limiar_ação** |
|---|---|---|
| lean_left | Inclinação à esquerda | Ângulo em graus de inclinação à esquerda |
| lean_right | Inclinação à direita | Ângulo em graus de inclinação à direita |
| lean_forwards | Inclinação para frente | Ângulo em graus de inclinação para frente |
| lean_backwards | Inclinação para trás | Ângulo em graus de inclinação para trás |
| turn_left | Girar à esquerda | Ângulo em graus de rotação |
| turn_right | Girar à direita | Ângulo em graus de rotação à direita |
| left_arm_forwards | Braço esquerdo à frente | Distância em polegadas da mão esquerda até o ombro |
| left_arm_down | Braço esquerdo para baixo | Distância em polegadas da mão esquerda até o ombro |
| left_arm_up | Braço esquerdo para cima | Distância em polegadas da mão esquerda até o ombro |
| left_arm_out | Braço esquerdo para o lado | Distância em polegadas da mão esquerda até o ombro |
| left_arm_across | Braço esquerdo através do corpo | Distância em polegadas da mão esquerda até o ombro |
| right_arm_forwards | Braço direito à frente | Distância em polegadas da mão direita até o ombro |
| right_arm_down | Braço direito para baixo | Distância em polegadas da mão direita até o ombro |
| right_arm_up | Braço direito para cima | Distância em polegadas da mão direita até o ombro |
| right_arm_out | Braço direito para o lado | Distância em polegadas da mão direita até o ombro |
| right_arm_across | Braço direito através do corpo | Distância em polegadas da mão direita até o ombro |
| left_foot_forwards | Pé esquerdo para frente | Distância em polegadas do quadril esquerdo até o pé |
| left_foot_sideways | Pé esquerdo ao lado | Distância de lado em polegadas do quadril esquerdo até o pé |
| left_foot_backwards | Pé esquerdo para trás | Distância para trás em polegadas do quadril esquerdo até o pé |
| left_foot_up | Pé esquerdo para | Altura em |

| | | |
|---|---|---|
| | cima | polegadas do pé esquerdo em relação ao pé que está no chão |
| right_foot_forwards | Pé direito para frente | Distância em polegadas do quadril direito até o pé |
| right_foot_sideways | Pé direito ao lado | Distância de lado em polegadas do quadril direito até o pé |
| right_foot_backwards | Pé direito para trás | Distância para trás em polegadas do quadril direito até o pé |
| right_foot_up | Pé direito para cima | Altura em polegadas do pé direito em relação ao pé que está no chão |
| jump | Pulo | Altura em polegadas dos pés acima do chão |
| crouch | Agachamento | Diferença entre a altura do usuário e o quanto ele agachou em polegadas |
| walk | Andando | Altura em polegadas de cada passo acima do chão quando andando na mesma posição. |

Algumas outras ações não necessitam de calibração ou de captura do esqueleto e são baseadas em gestos comuns. Antes, porém, do reconhecimento destas ações ser realizado, o usuário necessita realizar gesto de foco para "ligar" o reconhecimento destes gestos. Nesta versão do FAAST, o gesto de foco está fixo como uma onda (*wave*).

**Tabela 2**: Ações gestuais.

| Nome da Ação **nome_da_ação** | Significado | Limiar **limiar_ação** |
|---|---|---|
| push | Empurrar | Velocidade em polegadas por segundo |
| swipe_up | Rolagem para cima | Velocidade em polegadas por segundo |
| swipe_down | Rolagem para cima | Velocidade em polegadas por segundo |
| swipe_left | Rolagem para esquerda | Velocidade em polegadas por segundo |
| swipe_right | Rolagem para direita | Velocidade em polegadas por segundo |
| circle | Círculo | Raio do círculo feito em polegadas |
| wave | Onda | Deixar como zero, liga o reconhecimento |

Ao reconhecer alguma das ações programadas, um evento de teclado ou mouse é disparado. A tabela seguinte sumariza os possíveis tipos de evento e valores possíveis para cada tipo. Há um tipo de evento especial que habilita e desabilita o controle a partir do FAAST.

**Tabela 3**: Eventos de teclado e mouse.

| Tipo de evento virtual **tipo_evento_virtual** | Significado | Valores possíveis **nome_evento_virtual** |
|---|---|---|
| key_press | Tecla pressionada | Tecla que será pressionada uma vez quando a ação for detectada. Pode ser um caractere (a, b, c, etc), ou uma tecla especial. |
| key_hold | Tecla sendo pressionada | Tecla que será mantida pressionada enquanto a ação estiver sendo executada. Pode ser um caractere (a, b, c, etc), ou uma tecla especial. |
| mouse_click | Botão do mouse pressionado | • left_button: botão esquerdo do mouse;<br>• right_button: botão direito do mouse;<br>• middle_button: botão do meio do mouse. |
| mouse_double_click | Botão do mouse pressionado duas vezes | • left_button: botão esquerdo do mouse;<br>• right_button: botão direito do mouse;<br>• middle_button: botão do meio do mouse. |
| mouse_hold | Botão do mouse sendo pressionado | O botão permanecerá pressionado enquanto a ação estiver sendo executada.<br>• left_button: botão esquerdo do mouse;<br>• right_button: botão direito do mouse;<br>• middle_button: botão do meio do mouse. |
| faast | Controla o início da entrada de dados a partir do FAAST | • pause: interrompe temporariamente a execução do emulador;<br>• resume: continua a execução, se em modo de pausa.<br>• stop: sai do emulador. |

Os eventos key_hold e key_press permitem o mapeamento de teclas especiais:

*backspace, tab, clear, enter, shift, control, alt, pause, caps_lock, escape, space, page_up, page_down, end, home, left_arrow, up_arrow, right_arrow, down_arrow, select, print, execute, print_screen, insert, delete, help, left_windows, right_windows, applications, sleep, numpad_0,*

*numpad_1, numpad_2, numpad_3, numpad_4, numpad_5, numpad_6, numpad_7, numpad_8, numpad_9, multiply, add, separator, subtract, decimal, divide, f1, f2, f3, f4, f5, f6, f7, f8, f9, f10, f11, f12, f13, f14, f15, f16, f17, f18, f19, f20, f21, f22, f23, f24, num_lock, scroll_lock, left_shift, right_shift, left_control, right_control, left_alt, right_alt, browser_back, browser_forward, browser_refresh, browser_stop, browser_search, browser_favorites, browser_home, volume_mute, volume_down, volume_up, media_next_track, media_previous_track, media_stop, media_play_pouse, start_mail, media_select, start_application_1, start_application_2*.

Apresentada a ferramenta FAAST, que era nosso objetivo principal, fechamos este tutorial com alguns exemplos básicos de código que podem servir de esqueleto que o desenvolvedor mais experiente consiga desenvolver desde o início aplicativos que utilizem o Kinect. Desta forma, na próxima seção apresentaremos alguns recursos da biblioteca SimpleOpenNI do Processing como exemplo de códigos para trabalhar com o Kinect. Optou-se pelo Processing devido à simplicidade e aspecto multiplataforma, dando suporte a Windows XP, Vista, 7, Linux e Mac OS X. Destaca-se, porém, que esta não é a ferramenta oficial de desenvolvimento sugerida pela Microsoft. Mesmo assim, exemplos com o SDK oficial seriam similares e, a nível de comparação, estão presentes para comparação no site associado a este tutorial.

## 7. Exemplos com o Processing

Processing é um ambiente e uma linguagem de programação voltado para pessoas que desejam trabalhar com imagens, animação e interação [Processing 2011]. É uma opção bastante adequada para designers criarem protótipos de jogos digitais sem a necessidade de conhecer a fundo ferramentas de programação [Nakamura 2009]. Sugere-se o Processing como uma evolução natural para o desenvolvedor iniciar um protótipo com o Kinect após lidar com o FAAST.

Vamos usar a biblioteca *SimpleOpenNI* [Simple-OpenNi 2011], desenvolvida no Departamento de Design de Interação de Zurich. Esta biblioteca possui código-fonte aberto. É um *wrapper* para as funcionalidades do OpenNI e do NITE e seu objetivo não é ser completa em relação a todas as funcionalidades do OpenNI, mas sim entregar um acesso simples às suas principais.

Após o download e instalação da biblioteca através da cópia de seus arquivos para a pasta Libraries do Processing, a utilização do Kinect se dá a partir da criação de um objeto da classe *SimpleOpenNI* quando o programa carrega. Depois, o programador deve escolher quais recursos do Kinect vai capturar, a partir da chamada a métodos deste objeto. Os principais são:

- **enableDepth**: habilita a captura do mapa de profundidades;
- **enableGesture**: captura os gestos;
- **enableHands**: captura a posição das mãos;
- **enableIR**: caso a câmera RGB não esteja habilitada no momento, obtém a imagem da câmera infravermelha;
- **enableRGB**: obtém a imagem da câmera RGB;
- **enableScene**: habilita o recebimento de informações sobre a cena, permitindo o reconhecimento de cada pixel associado que está associado aos usuários presentes na cena;
- **enableUser**: captura o esqueleto do usuário.

A cada frame, explicitamente, o programa deve chamar um método de *update*, o qual atualiza as informações do Kinect. Métodos auxiliares como, por exemplo, *depthImage* e *rgbImage* devolvem as imagens provenientes do aparelho.

A Listagem **1** dá como resultado a exibição da imagem de profundidade, da imagem RGB e do reconhecimento de cada pixel associado ao usuário (Figura 10). Em um jogo, a imagem RGB pode ser utilizada para armazenar a foto do jogador e a imagem de profundidade indica através de tonalidades de cinza a profundidade de cada ponto.

Com a profundidade de um ponto poderíamos, por exemplo, efetuar alguma ação quando o jogador aproximasse seu corpo ou mesmo suas mãos de uma certa distância mínima. A informação da cena, por sua vez, permite que se tenha exatidão na captura do jogador desejado mesmo que existam outros jogadores envolvidos na cena.

**Figura 10**: Resultado da execução da Listagem 1.

**Listagem 1**: Exibição da imagem RGB, imagem de profundidade e reconhecimento do usuário.

```
import SimpleOpenNI.*;
SimpleOpenNI  context;

void setup()
{
  // Cria a instância do objeto associado
  // ao Kinect
  context = new SimpleOpenNI(this);
  // Habilita a geração do mapa de
  // profundidade
  context.enableDepth();
  // Habilita a geração da imagem RGB
  context.enableRGB();
  // Habilita a obtenção da informação do
  // usuário
  context.enableScene();
  // Faz com que o tamanho da tela
  // permita a exibição da imagem de
  // profundidade e RGB
  size(context.depthWidth()*3,
context.depthHeight());
}

void draw()
{
  // Atualiza o Kinect
  context.update();

  // Desenha a imagem de profundidade
  image(context.depthImage(),0,0);

  // Desenha a imagem RGB
  image(context.rgbImage(),
      context.depthWidth(),0);
  // Desenha a cena com o usuário
  image(context.sceneImage(),
          context.depthWidth()*2,0);
}
```

Outro recurso interessante da biblioteca está na captura da informação do esqueleto. Primeiro, o programador deve definir quais grupos de articulação vai capturar no método *enableUser*. Depois, deve tratar alguns eventos que percebem quando um usuário entra e sai no foco (*onNewUser*, *onLostUser*), quando começa e termina a calibração (*onStartCalibration*, *onEndCalibration*) ou para tratar a detecção da pose inicial (*onStartPose*, *onEndPose*). O método *getJointPositionSkeleton* obtém a informação de cada articulação e o método drawLimb permite a exibição do desenho de cada junta. No exemplo da listagem a seguir, a distância da mão em relação à cabeça faz com que se altere a cor de preenchimento do círculo.

**Listagem 2**: Controle do comportamento de um objeto a partir .

```
import SimpleOpenNI.*;
SimpleOpenNI  context;

void setup()
{
  context = new SimpleOpenNI(this);
  // Habilita a profundidade e a geração
  // do esqueleto para parte de cima do
  // corpo
  context.enableDepth();
  context.enableUser(
       SimpleOpenNI.SKEL_PROFILE_UPPER);
  // Cria a tela a partir do capturado pelo
  // Kinect
  size(context.depthWidth(),
       context.depthHeight());
}

void draw()
{
  stroke(255,0,0);
  strokeWeight(10);
  context.update();// Atualiza a câmera
  // Desenha a imagem de profundidade
  image(context.depthImage(),0,0);
  //Caso esteja disponível, desenha o esqueleto
  if(context.isTrackingSkeleton(1)){
      drawEllipse(1);
      drawSkeleton(1);
  }
}

// Desenha elipse
void drawEllipse(int userId)
{
  // Obtém a posição da articulação da cabeça e
  //a posição da articulação da mão esquerda
  PVector neckPos = new PVector();
  context.getJointPositionSkeleton(1,
          SimpleOpenNI.SKEL_HEAD,neckPos);
  PVector leftHandPos = new PVector();
  context.getJointPositionSkeleton(1,
    SimpleOpenNI.SKEL_LEFT_HAND,leftHandPos);
  // Distância entre as duas
  float d = dist(neckPos.x, neckPos.y,
   leftHandPos.x, leftHandPos.y);
  // Altera a cor a partir da distância
  fill(map(d, 0, 1000, 0, 255));
  ellipse(30,30, 60, 60);
}

// Desenha cada uma das partes do corpo
// (superior)
void drawSkeleton(int userId)
{
  context.drawLimb(userId,
           SimpleOpenNI.SKEL_HEAD,
           SimpleOpenNI.SKEL_NECK);
  context.drawLimb(userId,
           SimpleOpenNI.SKEL_NECK,
          SimpleOpenNI.SKEL_LEFT_SHOULDER);
  context.drawLimb(userId,
           SimpleOpenNI.SKEL_LEFT_SHOULDER,
          SimpleOpenNI.SKEL_LEFT_ELBOW);
  context.drawLimb(userId,
         SimpleOpenNI.SKEL_LEFT_ELBOW,
         SimpleOpenNI.SKEL_LEFT_HAND);
  context.drawLimb(userId,
```

```
          SimpleOpenNI.SKEL_NECK,
          SimpleOpenNI.SKEL_RIGHT_SHOULDER);
  context.drawLimb(userId,
          SimpleOpenNI.SKEL_RIGHT_SHOULDER,
          SimpleOpenNI.SKEL_RIGHT_ELBOW);
  context.drawLimb(userId,
          SimpleOpenNI.SKEL_RIGHT_ELBOW,
          SimpleOpenNI.SKEL_RIGHT_HAND);
  context.drawLimb(userId,
          SimpleOpenNI.SKEL_LEFT_SHOULDER,
          SimpleOpenNI.SKEL_TORSO);
  context.drawLimb(userId,
          SimpleOpenNI.SKEL_RIGHT_SHOULDER,
          SimpleOpenNI.SKEL_TORSO);
}

// ---------------
// Eventos SimpleOpenNI
void onNewUser(int userId)
{
  context.startPoseDetection("Psi",userId);
}

void   onEndCalibration(int   userId,   boolean
successfull)
{
  if (successfull)
  {
    context.startTrackingSkeleton(userId);
  }
  else
  {
    context.startPoseDetection("Psi",userId);
  }
}

void onStartPose(String pose,int userId)
{
  context.stopPoseDetection(userId);
  context.requestCalibrationSkeleton(userId,
          true);
}
```

**Figura 11**: Tela exibida após a execução da segunda Listagem.

## 8. Considerações finais

Este tutorial objetivou apresentar a ferramenta FAAST e sua possível aplicação na criação de jogos que utilizem como interface o Kinect.

Pode-se perceber, portanto, que o FAAST é uma ferramenta de uso bastante prático na fase inicial da criação de um jogo. Pode-se vislumbrar, até mesmo, sua aplicabilidade em ambientes relacionados à prototipação rápida de jogos e aplicativos como Game Jams e Hackdays. Todavia, para aplicações mais complexas, sugere-se o uso de uma ferramenta mais avançada como o Processing ou as APIs disponibilizadas pelo SDK oficial.

Os conceitos deste tutorial também podem também ser úteis para educadores na adaptação de jogos e aplicativos já existentes em um contexto educacional. A aprendizagem baseada em gestos abre um leque de possibilidades no qual os jogos assumem uma interface muito mais acessível e de imediata utilização pelos alunos. Além disso, os alunos ao interagirem com o Kinect também se obrigam a socializar com os demais, principalmente devido à obrigatoriedade do uso de gestos corporais. Troca-se um controle individualizante como o de um computador tradicional, por um controle muito mais social e integrador, o qual engaja os jogadores estimulando o uso de seus corpos na interação.

Finalmente, relembra-se que o material de apoio a este tutorial, suas listagens, exemplos complementares e possíveis erratas estão disponíveis na página**:**
**http://www.brunocampagnolo.com/tutorialkinect2011**.

## Agradecimentos

O autor gostaria de agradecer ao Instituto de Tecnologia do Paraná (TECPAR) e à Pontifícia Universidade Católica do Paraná (PUCPR) pelo apoio a esta pesquisa. Também agradeço aos meus alunos da Pós-Graduação de Jogos da PUCPR pela revisão inicial deste material.

## Referências

BLAKE, J., 2011. Natural User Interfaces in .Net. Capítulo 1. p. 6. Disponível em: http://www.manning.com/blake/MEAP_Blake_ch01.pdf [Acesso em: 14 Agosto 2011]

BUSINESS WEEKLY, 2011. Microsoft Research Cambridge Wins MacRobert Award. Disponível em: http://www.businessweekly.co.uk/hi-tech/12019- microsoft-research-cambridge-wins-macrobert- award [Acesso em: 14 Agosto 2011]

BUXTON B., 2010. Entrevista CES 2010: NUI with Bill Buxton. Disponível em: http://channel9.msdn.com/posts/LarryLarsen/CES-2010-NUI-with-Bill-Buxton [Acesso em 27 Agosto 2011].

FAAST 0.08, 2011. Disponível em: http://projects.ict.usc.edu/mxr/faast/ [Acesso em 29 Agosto 2011].

FRIED L., 2011. DIY Kinect Hacking. Disponível em: http://www.ladyada.net/learn/diykinect/ [Acesso 29 Agosto 2011].

GILES, J., 2010. Inside the race to hack the Kinect. Disponível em: http://www.newscientist.com/article/dn19762-inside-the- race-to-hack-the-kinect.html [Acesso em 29 Agosto 2011].

KRUEGER, M., GIOFRIDDO, T., AND HINRICHSEN, K., 1985. VIDEOPLACE – An Artificial Reality. *In: Proceedings of CHI' 85*, San Francisco 14-18 April 1985. p. 35-40.

KRUEGER, M., HINRICHSEN, K., GIOFRIDDO, T. AND SONNANBURG, J., 1988. VIDEOPLACE '88. Studio in The Museum of Natural History. Disponível em: http://www.youtube.com/watch?v=dmmxVA5xhuo [Acesso em 29 Agosto 2011].

MICROSOFT, 2010. Introducing Kinect for XBOX 360. Disponível em: http://www.xbox.com/en-US/Kinect [Acesso em 10 Julho 2011].

MICROSOFT, 2011. Kinect Sales Surpass Ten Million. Disponível em: http://www.xbox.com/en-us/press/archive/2011/0308-ten-million-kinects [Acesso em 10 Julho 2011].

MICROSOFT RESEARCH, 2011. Programming Guide: Getting Started with the Kinect for Windows SDK Beta. Disponível em:http://research.microsoft.com/en-us/um/redmond/projects/kinectsdk/docs/ProgrammingGuide_KinectSDK.docx [Acesso em: 13 Setembro 2011].

NAKAMURA, R., TORI, R., 2009. Processing como Ferramenta para Game Design. In: Proc. VIII Brazilian Symposium on Games and Digital Entertainment, 08-10 October 2009 Rio de Janeiro. 1-22.

OPENKINECT, 2011. Disponível em: http://openkinect.org/ [Acesso em: 13 Setembro 2011].

OPENNI, 2011. Disponível em: http://openni.org/ [Acesso em: 13 Setembro 2011].

PRIMESENSE, 2011. Disponível em: http://www.primesense.com/ [Acesso em: 13 Setembro 2011].

PROCESSING 1.5, 2011. Disponível em: http://www.processing.org/ [Acesso em: 10 Setembro 2011].

QUEIROZ, M., 2010. Um cientista explica o Microsoft Kinect. Disponível em: http://webholic.com.br/2010/11/09/um-cientista-explica-o-microsoft-kinect/ [Acesso em: 13 Setembro 2011].

SUMA, E.A., LANGE, B., RIZZO, A. KRUM, D.M. AND BOLAS, M., 2011. FAAST: The Flexible Action and Articulated Skeleton Toolkit. *In: Proceedings of Virtual Reality Conference IEEE, 19-23 March 2011 Singapore*. 247-248.

REIMER, J., 2005. A History of the GUI. Ars Technica. Disponível em: http://arstechnica.com/old/content/2005/05/gui.ars [Acesso em 28 Agosto 2011].

ROWAN, D., 2010. Kinect for Xbox 360: The inside story of Microsoft's secret 'Project Natal'. Disponível em: http://www.wired.co.uk/magazine/archive/2010/11/features/the-game-changer?page=all [Acesso em 20 Agosto 2011].

SIMPLE-OPENNI, 2011. Disponível em: http://code.google.com/p/simple-openni/ [Acesso em: 10 Setembro 2011].

TAYLOR, R., HUDSON, T., SEEGER, A., WEBER, H., JULIANO, J. AND HELSER, A., 2001. VRPN: a device-independent, network-transparent VR peripheral. *In: Proceedings of ACM Symposium on Virtual Reality Software and Technology*, 2001. 55-61.