Capítulo 2

Interfaces para Aplicações de Interação Natural Baseadas na API OpenNI e na Plataforma Kinect

Almerindo N. Rehem[1,2], Celso A. S. Santos[1] e Marcus V. R. Andrade[2]

[1] Universidade Federal da Bahia {almerindo.rehem@gmail.com , saibel@dcc.ufba.br}
[2] Universidade Tiradentes {vinny@touchingtheair.org}

Abstract

This course intent to introduce developers to concepts of the framework OpenNI (Open Natural Interaction) and the development of Natural Interaction applications using the Kinect hardware being compliant with the pattern proposed by the framework, so the same code can be used with different hardware that is certified by the OpenNI framework project. Besides the framework, the middleware library NITE will also be used, that is shipped and licensed separately, but can be used for non-commercial proposes.

Resumo

Este curso tem intenção de introduzir desenvolvedores aos conceitos do framework OpenNI (Open Natural Interaction) e o desenvolvimento de aplicações de interação natural usando o hardware Kinect de forma aderente aos padrões propostos pelo framewok. Sendo assim, o mesmo código pode ser usado com outro hardware que também seja certificado pelo projeto OpenNI. Além do framework também será usada a biblioteca de middleware NITE, que é fornecida e licenciada separadamente, mas pode ser utilizada para fins não comerciais.

2.1. Introdução

A Interação Usuário Computador (IUC) é uma área multidisciplinar que envolve áreas da ciência da computação, psicologia, linguística, artes, dentre outras. O conhecimento sobre as limitações da capacidade humana, restrições e evoluções (em termos de

dispositivos, interfaces e poder de processamento) das tecnologias existentes devem ser levados em conta para oferecer aos usuários uma forma adequada para interagir com plataformas computacionais. A evolução recente das tecnologias para captura de interações, muitas vezes de forma ubíqua, dos usuários criou novos paradigmas para a concepção de interfaces para aplicações interativas. O *Wiimote* [Wii Remote 2009] e o *Kinect* [Kinect (2010)], [Projeto OpenKinect 2011], [PrimeSense 2011]são os dois principais ícones comerciais que caracterizam essa nova forma de se pensar em interfaces, facilitando a comunicação entre usuário e computador por meio de interações naturais. Neste capítulo, considera-se como interação natural, uma comunicação entre o usuário e uma aplicação computacional realizada por gestos e/ou voz, em contrapartida à forma convencional baseada em janelas, ícones, *mouse* e apontadores (paradigma WIMP). O material apresentado neste texto é baseado na documentação oferecida aos desenvolvedores pela organização OpenNI disponível em [OpenNI User Guide 2011].

Lançado em 2006, o *Wiimote* (ou *Wii Remote*), baseado em acelerômetros, é o dispositivo principal de interação do console Nintendo Wii. O *Kinect*, fabricado pela *PrimeSence*, é o dispositivo principal de interação com o console *Xbox* 360, lançado em 2010 pela *Microsoft*, que traz como grande inovação uma interface baseada no reconhecimento de gestos. Apesar do foco inicial na área de jogos interativos, pesquisadores e desenvolvedores têm utilizado estas plataformas como base para a construção de interfaces naturais para aplicações interativas [Shiratori et all 2008],[Nakra et all 2009]. No caso do *Kinect*, em particular, menos de uma semana após seu lançamento oficial, a empresa *Adafruit* lançou o desafio "*Open KinectChallenge*", que oferecia U$3.000 para o primeiro desenvolvedor de um *driver* de código aberto para o novo dispositivo[Adafruit Industries 2011]. Com o desafio e a recompensa lançados, não demorou para que a primeira implementação *opensource* de *drivers* para o *Kinect* (a biblioteca *libfreeneck* [Projeto OpenKinect 2011]) fosse apresentada, abrindo o caminho para o desenvolvimento de aplicações fora da área de jogos.

Reconhecendo a iniciativa e os esforços dos desenvolvedores espalhados pelo mundo, a *PrimeSense* fundou a *OpenNI Organization* com o objetivo de acelerar a introdução de aplicações de interação natural no mercado e, com isso, liberou o código fonte do seu *framework* para os desenvolvedores e pesquisadores. Esta sequência de fatos aliada à disseminação de vídeos com resultados experimentais na Web (principalmente através do *YouTube*) desencadeou uma corrida para a criação e desenvolvimento de aplicações para a plataforma *Kinect* e para o paradigma de Interação Natural baseado no reconhecimento de gestos.

Um dos grandes desafios para a área de IUC é a concepção de interfaces genéricas e de utilização simples dentro de um contexto que é muito influenciado por fatores tecnológicos e culturais. Um bom exemplo é a maneira como as janelas, ícones e menus direcionam um paradigma. Atualmente, as pessoas estão habituadas a acessar informações e serviços por meio de interfaces desenhadas baseadas em paradigmas WIMP ou em abstrações, como a Web. Um exemplo de metáfora desatualizada e ainda em uso é a utilização da figura de um disquete como ícone para salvar um arquivo. Outro exemplo é a interface típica de programas de reprodução de vídeos digitais com ícones baseados nas interfaces dos antigos videocassetes.

2.2. Interação Natural

A Interação Natural procura formas de tornar uma IUC o mais natural que possível, com objetivo de fazer com que as pessoas manipulem uma máquina sem perceber qual(is) artefato(s) é (são) utilizado(s) e sem necessariamente aprender um novo vocabulário correspondente ao conjunto de instruções desta interface.

Para [Heckel 1993], a interface deve procurar reproduzir a linguagem natural do usuário de computador ou da máquina, tentando prover o uso de palavras e expressões conhecidas, respeitando o vocabulário do usuário. Esta tentativa terá seus reflexos na redução do uso da memória e na diminuição do esforço cognitivo do usuário e, consequentemente, influirá numa interação mais "agradável" e natural.

[Valli 2007] define a interação natural em termos experimentais: as pessoas naturalmente se comunicam através de gestos, expressões, movimentos, além de explorarem o mundo real através da observação e manipulação de objetos físicos. E como uma tendência cada vez mais forte, as pessoas querem interagir com a tecnologia da mesma forma como lidam com o mundo real no cotidiano. A criação de novos paradigmas de interação e padrões alternativos de mídia que explorem as novas capacidades dos sensores presentes nas máquinas, e ainda, respeitem a espontaneidade inerente ao modo como o ser humano descobre e interage com o mundo é o principal desafio da construção de novas interfaces para interação. A tendência de quebra do paradigma tradicional das interfaces é evidenciada através de diversos experimentos usando novas propostas de interação surgindo cotidianamente na Web [Kinecthacks 2011] e do interesse de empresas como a *Microsoft*, que começa a demonstrar interesse na exploração dessas novas formas de interação [Toyama 1998]. Dessa forma, os usuários têm sido cada vez mais direcionados a um cenário de computação ubíqua, isto é, um cenário no qual os serviços digitais (ou computacionais) estarão disponíveis às pessoas em qualquer lugar e acessíveis para que interajam de forma natural com esses serviços [Weiser and Brown 1995], [Cordeiro Jr. 2009], [Abowd and Mynatt 2000].

[Saffer 2009] diz que um gesto na interação natural usuário computador é qualquer movimento físico que um sistema digital possa reconhecer e ao qual se possa responder. Um som, um inclinar de cabeça ou até mesmo uma piscada podem ser considerados gestos. Sendo o reconhecimento dos gestos um dos problemas atuais no desenvolvimento de aplicações dependentes de contexto, o poder de se contextualizar para poder reconhecer os gestos de forma precisa e responder da mesma forma é o que possibilita a interação natural usuário computador.

[Gallud *et all* 2010] define a comunicação não verbal como aquela que utiliza pistas e sinais sem estrutura sintática e separa essa comunicação em três tipos:

- Paralinguagem**:** Refere-se aos elementos que acompanham expressão linguística, consistindo de pistas e sinais que sugerem uma interpretação diferente do que está sendo dito, percebe-se pela intensidade e/ou altura da voz do emissor bem como chorar, rir, controle respiratório, etc.
- Linguagem corporal: Caracteriza-se pela utilização de gestos, porém seguida de características distinguíveis utilizando o tronco, extremidade e cabeça também pela proximidade entre emissor e receptor, expressões de estado como felicidade, honestidade, cumplicidade, etc.

- Linguagem sonora: Acompanha os gestos e caracteriza-se pela interpretação qualquer som que expresse emoção, contextualiza uma cena e, até mesmo o silêncio pode ter significado em uma mensagem.

A detecção e interpretação de tantos sinais para uma resposta adequada dependem de análise e contextualização extremamente rápidas dos sinais recebidos. Assim, inúmeros fatores podem influenciar o entendimento correta de uma mensagem na comunicação usuário computador.

Se for possível utilizar linguagem corporal e falada como intervenções de usuário, abre-se a possibilidade de criar interfaces onde o usuário e a máquina poderão se comunicar de forma mais humana e natural. Por outro lado, a criação de interfaces baseadas na comunicação usuário computador por meio de gestos e voz traz consigo um novo tipo de problema: quais gestos de interação e comandos associados devem ser utilizados para realizar uma determinada tarefa?

A proposta deste capítulo é apresentar uma forma padronizada de desenvolver de forma modular os componentes, que somados viabilizam a concepção de aplicações de interação natural, podendo ser compartilhados e reutilizados para aplicações de diferentes finalidades e possibilitando a criação de um ecossistema de bibliotecas e aplicações para auxiliar os desenvolvedores da área em questão.

2.3. Evolução dos Dispositivos de Interação Usuário Computador

Esta seção do texto irá abordar alguns dos principais dispositivos que mudaram a forma de se pensar a interação usuário computador, finalizando com a apresentação da arquitetura do *Kinect* e os seus recursos associados.

2.3.1. Os dispositivos Wiimote e Playstation Move

O *Wiimote* e o *Playstation Move* se parecem muito em termos de funcionamento. Eles são baseados essencialmente em emissão de luz infravermelha e uma câmera de RGB tradicional. A luz infravermelha é detectada pela câmera RGB e assim, é possível detectar a posição do ponto de referência: no caso do *Wiimote* esse ponto é o próprio controle e no caso do *Playstation Move*, é a peça com uma extremidade luminosa que melhora a precisão que é dada pela câmera RGB. Este processo pode ser feito de forma 2D resultando numa posição na tela, exatamente como um ponteiro de mouse, ou de forma 3D, onde para detectar a distância do controle até a tela é utilizada uma triangulação. Além disso, quando há um acelerômetro disponível ao conjunto de sensores, além de posição e distância, as inclinações em qualquer eixo também podem ser detectadas. Essa inclinação tem o nome de *DOF* (D*egrees Of Freedom*). Estas técnicas, já eram conhecidas desde a década de 80 quando a Nintendo lançou o controle *Power Glove* (Figura 1.1) que funcionava de maneira muito similar aos dispositivos *Wiimote* e *Playstation Move*, com a diferença que, em conjunto com a câmera RGB, ao invés de infravermelho, utilizavam-se sensores de ultrassom para medir o tempo entre a reflexão das ondas emitidas por um aparelho posicionado na TV e o material da luva.

Além das coordenadas (*x,y,z*) do controle em relação a cena, para uma boa experiência também é necessário que os seus ângulos (*roll*, *pitch* e *yaw*) possam ser detectados possibilitando calcular a inclinação (ver Figura 1.2). Tanto o *Playstation Move* quanto o *Wiimote* junto com o *Motion Plus* já trabalham com esses 6 graus de liberdade. A *Power Glove*, ainda na década de 80, já retornava além das coordenadas

(*x,y,z*), também fornecia o *yaw* e o *roll***.** Apesar da quantidade enorme de informações sobre posicionamento e inclinação obtidos com esses dispositivos, ainda assim são poucas as informações para imitar o mundo real, simplesmente porque as informações estão relacionadas apenas ao controle e por isso, não levam em conta a cena ou o jogador/ator em si, nem permite a detecção de gestos.

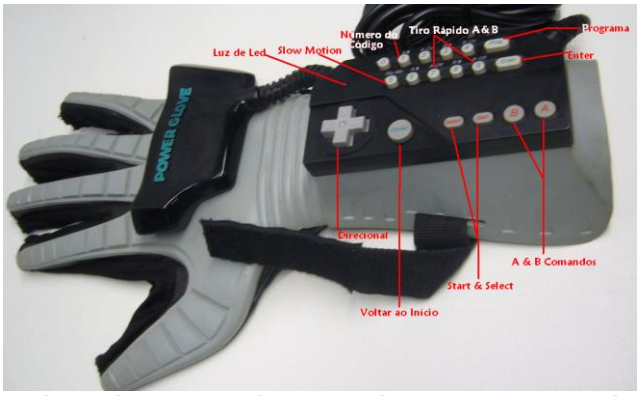

Figura 1.1. Nintendo *Power Glove*, um dos antecessores do *Wiimote*.
Fonte: [Power Glove 2008]

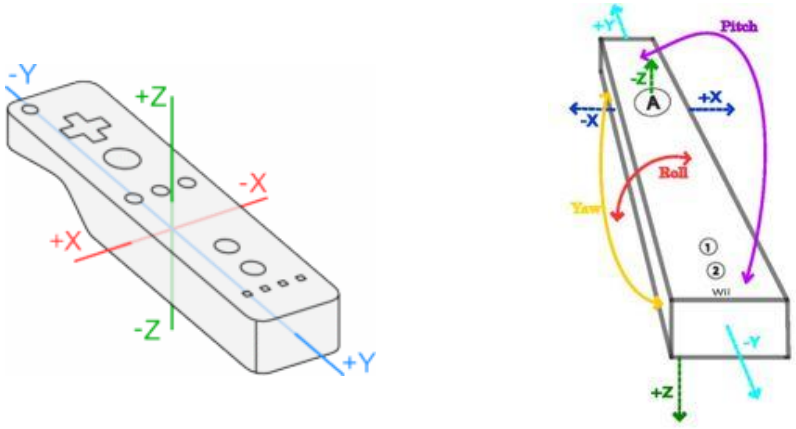

Figura 1.2. (a) Eixos de referência do *Wiimote* e (b) os seis graus de liberdade.
Fontes: [Kyma 2010] e [Wiimote 2010]

No caso do *Kinect*, a geração de um mapa de profundidade (cada pixel da imagem está relacionado a uma distância do pixel até a câmera) a qual é a principal inovação, possibilita ao dispositivo entregar informações 3D da cena completa, incluindo o jogador/ator e não somente do controle do usuário. Para se conseguir este resultado, a tecnologia por trás do *Kinect* utiliza o conceito de luz estruturada: uma luz com um padrão de projeção bem definido que permite o cálculo das informações da cena partindo da combinação entre os padrões do que é projetado e da distorção da projeção em função dos obstáculos encontrados pela luz (ver Figura 1.3).

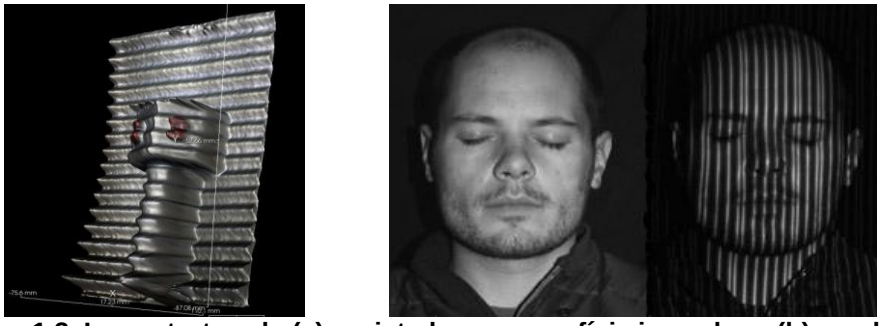

Figura 1.3. Luz estruturada (a) projetada em superfície irregular e (b) usada no mapeamento da face.
Fonte: [Shape Quest 2010]

2.3.2. O Kinect

O *Kinect* possui um sofisticado algoritmo de processamento paralelo (embarcado no chip SoC) necessário para extrair o mapa de profundidade a partir da luz estruturada recebida (Figura 1.4 e Figura 1.5). Para possuir mais precisão nas informações dos sensores, as imagens são alinhadas pixel a pixel, ou seja, cada pixel de imagem colorida é alinhado a um pixel da imagem de profundidade. Além disso, o *Kinect* sincroniza (no tempo) todas as informações dos sensores (profundidade, cores e áudio) e as entrega através do protocolo USB 2.0 [Crawford 2010].

A Figura 1.4 e a Figura 1.5 apresentam, respectivamente, o posicionamento dos sensores e o diagrama esquemático do *Kinect*. Nas figuras, é possível identificar como os sensores estão conectados e quais são os fluxos de dados gerados pelo dispositivo.

Figura 1.4. (a) Sensores do *Kinect* no *XBOX 360*.
Fonte: [Crawford 2010]

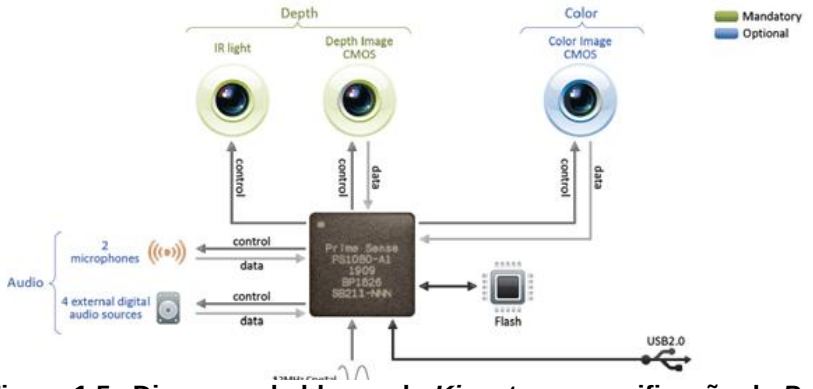

Figura 1.5. Diagrama de blocos do *Kinect* na especificação da *PrimeSense*. Fonte: [PrimeSense 2011]

O *Kinect* dispõe de vários recursos (som, imagem, profundidade, infravermelho, motor de movimentação) com um alto índice de precisão e sincronismo em um único dispositivo. Estes recursos oferecem uma série de possibilidades de interação inovadoras entre usuários e serviços e aplicações computacionais. O *framework OpenNI* e o módulo *NITE*, tratados nas próximas seções do texto, permitem construir interfaces de aplicações sofisticadas, em um nível mais alto de abstração, tendo como base a combinação de recursos da plataforma *Kinect*. As seções seguintes do texto são fortemente baseadas na recente documentação [OpenNI User Guide 2011], um guia para usuários desenvolvedores interessados em utilizar o *framework OpenNI*.

2.4. Introdução ao OpenNI

OpenNI (ou *Open Natural Interaction*) designa uma instituição sem fins lucrativos e também uma marca registrada da *PrimeSense* [PrimeSense 2011]. A *OpenNI* busca promover a interoperabilidade e compatibilidade entre aplicações, *middleware* e dispositivos de interação natural, atuando na verificação e certificação da conformidade das soluções de diferentes fabricantes e desenvolvedores aos padrões definidos pelo *framework OpenNI*.

O principal propósito da instituição *OpenNI* é especificar uma API padrão para a comunicação tanto para dispositivos (sensores) quanto para aplicações e *middlewares.* Com isso, busca-se quebrar a dependência entre os sensores e os *middlewares*, além de permitir o desenvolvimento e portabilidade de aplicações para diferentes módulos de *middlewares* com menor esforço adicional (conceito "*write once, deploy everywhere"*). Mais ainda, a API fornece: (i) o acesso direto aos dados brutos dos sensores aos desenvolvedores e (ii) a possibilidade de se construir dispositivos que funcionem em qualquer aplicação compatível com o padrão proposto aos fabricantes.

O *framework OpenNI* habilita aos desenvolvedores acompanharem as cenas do mundo real (cenas em 3D), utilizando tipos de dados processados capturados por um sensor de entrada. Por exemplo, a representação do contorno de um corpo humano, a localização de uma mão na tela, etc. Essas aplicações, graças ao padrão *OpenNI*, são construídas com um alto nível de desacoplamento, ou seja, podem ser construídas de forma independente dos fabricantes de sensores ou de um determinado *middleware*.

A Figura 1.6 oferece uma visão abrangente, em camadas, da arquitetura *OpenNI* e dos conceitos associados a esse padrão. Essas camadas são divididas em *Top*, *Middle*, *Botton*, e descritas da seguinte forma:

- ***Top***: Representa os softwares que implementam aplicações de Interação Natural seguindo o padrão *OpenNI*;
- ***Middle***: Representa o próprio *framework OpenNI* e oferece interfaces de comunicação para interagir tanto com os middlewares quanto com os dispositivos (sensores);
- ***Bottom***: Ilustra a camada de mais baixo nível, onde se situam os dispositivos de hardware responsáveis por capturar elementos visuais e auditivos da cena.

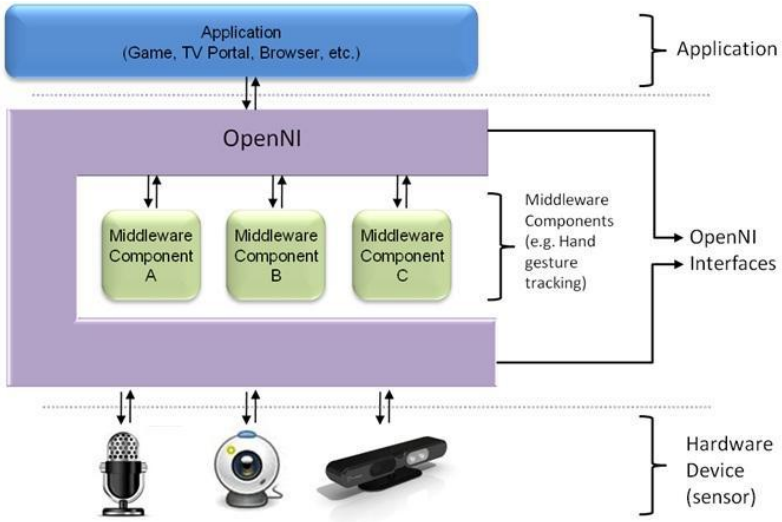

Figura 1.6. Abstração e componentização do *OpenNI*.
Fonte: [PrimeSense 2011]

2.4.1. O Framework OpenNI

Pelo exposto até aqui, pode-se afirmar que o *framework* é uma camada de abstração que fornece interfaces para os componentes de softwares (*middlewares*) e para os dispositivos físicos através de uma API. Esta API habilita registros de múltiplos componentes (módulos) no *framework* que são responsáveis por produzir ou processar dados dos sensores físicos de forma flexível. Atualmente, os componentes suportados pela arquitetura são agrupados em dois tipos, os módulos de sensores e *middlewares*, descritos como se segue:

- **Módulos de sensores**:
  - **Câmera RGB**: dispositivo responsável por gerar imagens coloridas;
  - **Sensor 3D**: dispositivo capaz de gerar o mapa de profundidade;
  - **IR câmera**: dispositivo de infravermelho;
  - **Dispositivo de áudio**: um ou mais dispositivos de microfones
- ***Middlewares***:
  - ***Middleware* de análise de corpo**: componente de software capaz de processar os dados de entrada dos sensores e retornar informações relacionadas ao corpo humano. Como por exemplo, uma estrutura de

dados que descreva as articulações, orientação e o centro de massa de uma pessoa na cena.

- ***Middleware* de análise de mão**: componente capaz de processar dados dos sensores, reconhecer e retornar uma coordenada de localização do centro de uma mão em cena.
- ***Middleware* de detecção de gestos**: componente capaz de identificar gestos pré-definidos e alertar a aplicação todas as vezes que houver ocorrência desses gestos.
- ***Middleware* de análise de cena**: componente capaz de analisar os dados da cena e produzir informações como: (i) a separação dos planos *foreground* e do *background*; (ii) as coordenadas do chão e (iii) a identificação de atores em cena.

A Figura 1.7 ilustra um exemplo de aplicação dos componentes da arquitetura. No exemplo, é exibido um cenário em que cinco módulos são registrados para trabalhar em uma instância do *framework OpenNI*. Dois dos módulos registrados são sensores 3D conectados fisicamente ao host. Os outros três, são componentes de *middleware*, sendo que os dois primeiros módulos retornam dados relacionados ao corpo humano e o último, dados relacionado à mão (*hand point*).

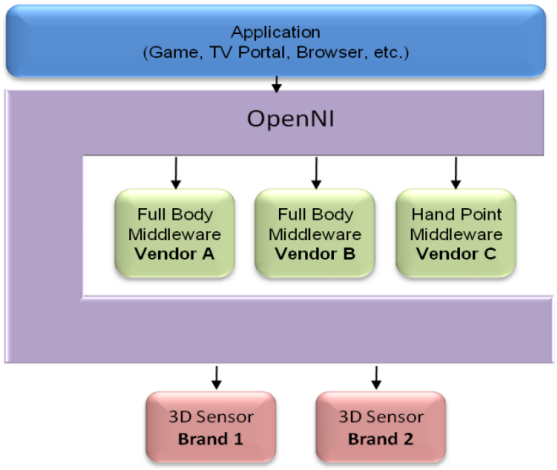

Figura 1.7. Exemplo de uso da arquitetura onde são registrado cinco módulos simultâneos em uma única instância do *OpenNI*.
Fonte: [PrimeSense 2011]

2.4.2. Nós de produção

Para um melhor entendimento da arquitetura *OpenNI*, é necessário compreender o conceito de base desta arquitetura, denominado nó de produção (*production node*). Todo componente que produza dados necessários para as aplicações baseadas em interação natural é chamado de nó de produção. Um nó de produção pode utilizar outros nós, ser usado por nós de mais alto nível ou diretamente pela própria aplicação.

Os nós de produção são divididos em duas categorias: (i) nós relacionados ao *middleware* e (ii) nós relacionados aos sensores.

- **Nós relacionados aos sensores**:
  - ***Device***: nó de produção que representa um dispositivo físico (sensor de profundidade, ou uma câmera RGB) e tem como principal objetivo permitir configurações dos dispositivos;

- ***Depth Generator***: nó responsável por gerar um mapa de profundidade;
- ***Image Generator***: nó responsável por gerar um mapa de pixels RGB (imagem colorida);
- ***IR Generator***: nó que gera o mapa de imagem infravermelha;
- ***Audio Generator***: nó responsável pelo tratamento da *stream* de áudio.

- **Nós relacionados ao *middleware***:
  - ***Gestures Alert Generator***: nó de produção responsável por alertar a aplicação quando houver um reconhecimento de um gesto;
  - ***Scene Analyzer***: nó responsável por gerar os dados de análise de cena, possibilitando a separação das camadas *foreground, backgroung, floor plane* e identificação do ator;
  - ***Hand Point Generator***: nó responsável pelo suporte à detecção e rastreamento de uma mão (esse nó produz um sinal quando uma mão é detectada ou quando sua localização é alterada);
  - ***User Generator***: nó responsável por produzir uma representação parcial ou total do corpo humano em uma cena.
  - ***Recorder***: nó que implementa a funcionalidade de gravação de dados;
  - ***Player***: nó responsável por ler os dados gravados e exibi-los;
  - ***Codec***: nó responsável pela compressão e descompressão dos dados.

Para facilitar o entendimento da utilização desses nós de produção e seus relacionamentos, pode-se usar um exemplo inicial de aplicação na qual é necessário acompanhar o movimento de uma figura humana na cena. Esta aplicação requer nós de produção que forneçam dados referentes ao corpo humano ou, em outras palavras, um nó *UserGenerator.*Por sua vez, o nó de produção *UserGenerator* necessita obter dados do nó *DepthGenerator* (outro nó de produção de um nível mais baixo na hierarquia), que é implementado para capturar os dados brutos de um sensor de profundidade (capturar *N* frames por segundo e retornar o mapa de profundidade). Essa seqüência de relacionamentos e dependências entre os nós de produção é conhecida pelo *OpenNI* como cadeia de produção (*Production Chain*) e será mais detalhada na próxima seção.

Outros exemplos comuns de aplicações com saída de dados com um alto nível de abstração são os casos de localização de uma mão, identificação de pessoa em cena e a identificação de gestos, descritos a seguir:

- Localização da mão de um usuário (Figura 1.8.a): o nó de produção pode retornar o centro da palma da mão através de um ponto (comumente referenciada pelo *hand point),* ou a localização das pontas dos dedos;
- Identificação de um ator na cena (Figura 1.8.b). o nó de produção retorna a localização momentânea e orientação das articulações do ator (comumente referenciada como *body data*);
- Identificação de gestos (como um aceno, na Figura 1.8.c): o nó de produção se responsabiliza por identificar e alertar a aplicação, através do *framework*, quando um determinado gesto ocorrer.

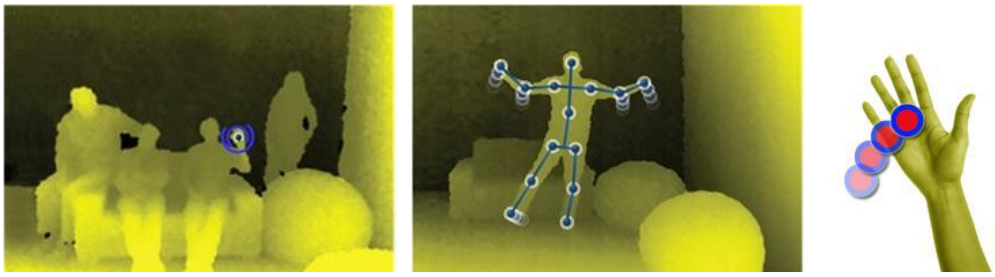

Figura 1.8. (a) Detecção da Mão; (b) Detecção do Corpo; (c) Detecção de Gestos. Fonte: [OpenNI User Guide 2011]

2.4.3. As cadeias de produção

Na seção anterior foi apresentado um exemplo de aplicação que usa o nó *User Generator* para produzir a representação do corpo humano (*body data*). Este nó de produção *UserGenerator*, por sua vez, necessita de informações geradas por outros nós dos níveis inferiores da arquitetura. Os dados, relativos ao corpo humano, gerados pelo nó de produção *UserGenerator* são utilizados pela aplicação em si ou por outros nós de produção, definindo assim, a cadeia de produção da informação.

Uma aplicação comumente irá necessitar dos nós de produção de mais alto nível, tais como o *UserGenerator* e o *HandPointGenerator*. Essa necessidade é contemplada pela arquitetura, pois o *framework* possibilita a utilização dos nós de produção de mais alto nível sem que o desenvolvedor se preocupe com a cadeia de produção. Entretanto, o *OpenNI* é flexível o suficiente para permitir o acesso direto à configuração dos nós pertencentes às cadeias de produção.

2.4.4. As funcionalidades

As funcionalidades (tradução livre para *capabilities*) são mecanismos para permitir o registro de nós de produção de um mesmo tipo, mas produzidos por fabricantes diferentes. Com isso, uma aplicação pode "perguntar" ao *framework* quais dos nós do conjunto possuem a funcionalidade que ela necessita e, assim, incluir os nós necessários para a geração da cadeia de produção. Atualmente, as funcionalidades suportadas pelo *framework* são:

- ***Alternative view***: habilita um sensor do tipo *Map Generator* a transformar seus dados para que pareça que o sensor tenha sido colocado em um local diferente do qual ele realmente se encontra;
- ***Cropping***: possibilita que um sensor do tipo *Map Generator* gere dados apenas de uma área do frame ao invés de todo o *frame*;
- ***Frame Sync*:** habilita dois sensores de qualquer tipo a sincronizarem suas informações de modo que os dados cheguem aos seus nós consumidores ao mesmo tempo;
- ***Mirror***: possibilita que sensores "visuais" informem seus dados de forma espelhada, de forma semelhante à formação da imagem num espelho;
- ***Pose Detection***: permite que um nó do tipo *User Generator* reconheça quando um usuário está em uma posição específica;

- ***Skeleton***: habilita que o *User Generator* forneça como dados de saída o esqueleto do usuário; permite também a localização das articulações, a identificação da posição do esqueleto e a calibração do usuário;
- ***User Position***: habilita um sensor do tipo *Depht Generator* a otimizar a saída do mapa de profundidade gerado para uma área específica da cena;
- ***Error State***: permite que um nó de qualquer tipo informe que ele está em um estado de erro ou que não está em perfeito funcionamento;
- ***Lock Aware***: permite que um nó qualquer seja bloqueado para uso fora de um determinado contexto; em outras palavras, não permite que ele seja acessado fora de uma determinada cadeia de produção ou por um nó diferente daquele que efetuou o bloqueio.

2.4.5. Porque utilizar o *famework OpenNI*?

A organização e implementação de uma camada de abstração do *hardware* e a flexibilidade obtida com registros de *middlewares* tornam as soluções desenvolvidas dentro do padrão *OpenNI* mais abrangentes e compatíveis com um número maior de hardwares e de bibliotecas que possam vir a ser distribuídas.

No caso da disponibilização de códigos fonte ou de bibliotecas compiladas não aderentes ao *framework* existe a possibilidade de concepção de *Wrapper* para garantir a aderência e a utilização dentro do *framework*. Em último caso, é possível ainda acessar diretamente a biblioteca e fazer com que o nó construído seja o ponto gerador de informação proveniente da biblioteca.

Com o objetivo de posicionar o projeto *OpenNI* e apresentar as vantagens desse projeto com relação ao *Kinect SDK*[1] recentemente liberado pela *Microsoft* [Kinect SDK (2010)], três tabelas comparativas são apresentadas na sequência do texto. Os critérios levados em conta na comparação das duas opções de programação foram: (i) o licenciamento das bibliotecas e ferramentas; (ii) o suporte a hardware e (iii) as funcionalidades inclusas. Note que a vantagem incontestável do *OpenNI* com relação à portabilidade (suporte ao *Windows 7/Vista/XP*, *Linux* e *MacOS X*) sobre o *Microsoft Kinect SDK* (que funciona apenas para *Windows* 7) não foi considerada na comparação.

Tabela 1.1. Comparação quanto ao licenciamento.

	OpenNI	***Microsoft Kinect SDK***
Ferramentas para desenvolvimento	Uso comercial	Uso comercial
Bibliotecas e *framework*	Uso comercial	Uso não comercial

Tabela 1.2. Comparação quanto ao suporte a dispositivos de hardware.

	OpenNI	***Microsoft Kinect SDK***
Suporte de Áudio	Não Suporta	Suporta
Suporte ao motor de passos	Suporte Parcial[2]	Suporta
Suporte ao sensor de profundidade	Suporta	Suporta
Suporte ao sensor RGB	Suporta	Suporta
Suporte a múltiplos sensores	Suporta	Suporta
Suporte a sensores de outros fabricantes	Suporta[3]	Não suporta

[1] *Kinect SDK* (ou *Kinect Software Development Kit*) é o *kit* de desenvolvimento de software para o *Kinect* oferecido pela *Microsoft*.
[2] O suporte pode ser obtido usando um *driver* de outro equipamento, que é parcialmente compatível.
[3] Qualquer hardware certificado *OpenNI*, aderente ao *framework*, é suportado.

Tabela 1.3. Comparação quanto às funcionalidades inclusas.

	OpenNI	***Microsoft Kinect SDK***
Acompanhamento de corpo inteiro	Incluso	Incluso
Acompanhamento apenas de mão	Incluso	Não incluso
Alinhamento automático das informações do sensor de profundidade e RGB	Incluso	Não incluso
Cálculo da posição das juntas	Incluso	Incluso
Cálculo da rotação das juntas	Incluso	Não Incluso
Sistema de reconhecimento de gestos	Incluso	Não Incluso
Eventos para reconhecimento de entrada e saída de novos usuários na cena	Incluso	Não Incluso
Eventos para cada frame disponível na *stream* dos sensores RGB e de profundidade	Não Incluso	Incluso
Suporte para gravação para o disco e reprodução a partir dele, das informações dos sensores	Incluso	Não Incluso

Concluindo, pode-se prever a continuidade de desenvolvimento e a incorporação de melhorias no curto prazo no *SDK* da *Microsoft*, uma vez que o *Kinect* é um projeto lançado pela própria empresa. Por outro lado, no momento da escrita deste documento, o *OpenNI* é um projeto que se encontra com um nível alto de maturidade e com um número importante de aplicações e desenvolvedores já consolidados. Além disso, o *OpenNI*, diferentemente do *SDK* da *Microsoft*, não se limita a fornecer aos desenvolvedores um conjunto de métodos, mas uma arquitetura padrão de desenvolvimento para aplicações na área de interação natural, abrangendo fabricantes de sensores, desenvolvedores de *middlewares* e de aplicações dessa área.

2.5. Programando com o *OpenNI*

As seções seguintes deste capítulo terão como foco principal os aspectos práticos para programação utilizando o *framework OpenNI*, fornecendo o conhecimento básico e essencial para qualquer desenvolvedor iniciar suas aplicações relacionadas à interação natural dentro desta plataforma. O texto está estruturado de forma a apresentar: (i) os principais objetos usados na programação; (ii) a integração entre o *OpenNI* e o *Eclipse* CDT[4] [Eclipse CDT Project 2011] e (iii) alguns códigos comentados de exemplos de uso do *framework*. Por outro lado, não serão abordadas informações relacionadas à instalação do sistema operacional *Linux*, do *OpenNI* e do *middleware* NITE e suas bibliotecas requeridas. Estas informações podem ser encontradas em detalhes nas referências [PrimeSense 2011], [Projeto OpenKinect 2011], [OpenNI User Guide 2011].

2.5.1. Principais Objetos do Framework OpenNI

Os objetos mais comuns para o desenvolvimento de aplicações que utilizem o *framework OpenNI* são o *Context Object*, o *MetaData Object* e os objetos geradores de dados ou *Data Generators* (em especial, *Map Generator, Detph Generator, Image Generator, IR Generator, Scene Generator, Audio Generator, Gesture Generator, Hand Point Generator* e *User Generator*).

2.5.1.1. Objetos Context

A classe mais importante no *OpenNI* é a *Context* (Figura 1.9). O objeto pertencente a esta classe é a própria instância do *OpenNI*, sendo responsável por manter o estado

[4] *CDT* (*C/C++ Development Tooling*) é um ambiente de desenvolvimento integrado para C e C++ baseado na plataforma Eclipse resultado do projeto CDT [Eclipse CDT Project 2011].

completo da aplicação, inclusive de todas as cadeias de produção utilizadas. Mais de um *Context* pode ser instanciado, entretanto as informações entre eles não podem ser compartilhadas. Por exemplo, um nó de produção do tipo *middleware* não consegue acessar informações de um nó do tipo sensor que esteja em um contexto diferente.

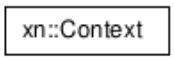

Figura 1.9. A Classe *Context* está no nível mais alto da hierarquia.

Para se iniciar o *framework,* o objeto *Context* deve ser instanciado em memória e todos os módulos necessários para aplicação que forem conectados ao *Context* serão carregados e analisados. O *Context* será então o ponto central de acesso a todos os módulos da aplicação e responsável também por liberar toda a memória utilizada pelo *OpenNI* através da função *shutdown*(), chamada pela aplicação.

2.5.1.2.Objetos Metadata

Os objetos *Metadata* (ver Figura 1.10) estão presentes em todos os nós de produção que gerem dados brutos (*Detph Generator, Image Generator, IR Generator, Scene Generator, Audio Generator*). Este objeto oferece um encapsulamento das propriedades relacionadas com os dados de quem ele pertence. Como por exemplo, a resolução (número de pixels nos eixos *x* e *y*) de um mapa de profundidade (*Depth Map*) obtido através do *DepthGenerator*.

Esse tipo de objeto seria necessário no caso de uma aplicação que tivesse que ler, a todo instante, os dados de um mapa de profundidade (*Depth Map*) que foi inicialmente configurado com resolução QVGA (320 x 240 *pixels*). Se, em algum momento, a resolução de saída fosse alterada para VGA (640 x 480 *pixels*), a aplicação teria uma inconsistência de informação até que novos *frames* fossem gerados. Na verdade, seria um erro assumir que a resolução encontra-se em VGA e tentar acessar *pixels* que não estivessem presentes na matriz (mapa de profundidade).

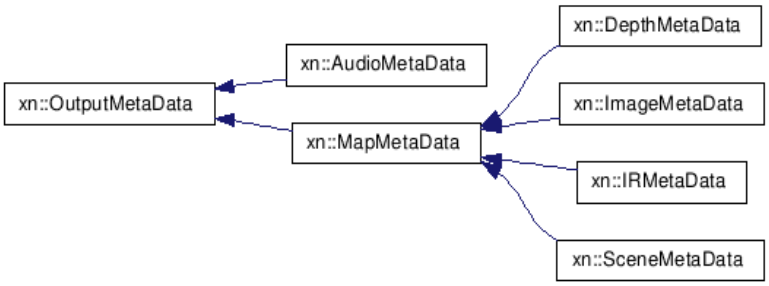

Figura 1.10. Hierarquia de Classes Metadata.

Como todos os *nós de produção geradores* possuem seu próprio *Metadata Object* responsável por registrar a propriedade dos dados. A solução do exemplo anterior seria obter esse *Metadata Object*, e a partir dele, ler os dados reais e sua resolução em tempo de execução (nesse caso, um mapa de profundidade QVGA).

2.5.1.3.Objetos *Data Generators*

Generator é um nó de produção que fornece dados de retorno para a aplicação. Se o nó retorna algum *map* (matriz de pixels) como resultado, então esse nó de produção implementa a interface básica *MapGenerator.* O diagrama da hierarquia dos objetos do tipo *Generator* é ilustrado pela Figura 1.11.

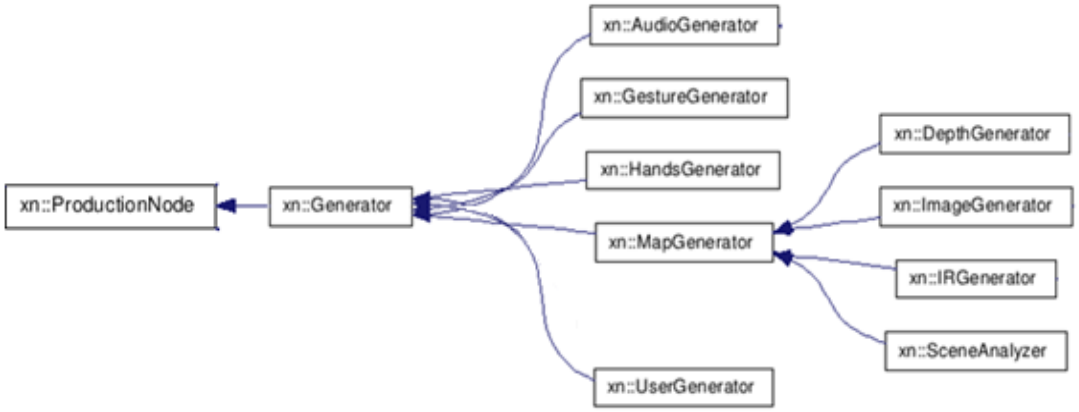

Figura 1.11. Hierarquia de classes dos *Generators*.

Os principais objetos e funções dos objetos do tipo *Generator* são os seguintes:

- ***DepthGenerator*:** Nó *Generator* que tem como foco retornar um mapa de profundidade (matriz de pixels de profundidade). Suas principais funcionalidades são:
  - ***Get Depth Map*:** retorna o ***map*** de profundidade;
  - ***Get Device Max Depth*:** retorna a distância máxima disponível pelo dispositivo;
  - ***Field of View property*:** configura o campo de visão (valores dos ângulos horizontais e verticais) do sensor;
  - ***User Position capability*:** retorna a posição do usuário.

- ***ImageGenerator*:** Nó ***Generator*** que retorna um mapa de imagem colorida (matriz de pixels coloridos) e tem como principais funcionalidades:
  - ***Get Image Map*:** retorna o *map* da imagem;
  - ***Pixel Format Property***

- ***IRGenerator*:** Nó ***Generator*** que fornece um mapa infravermelho e que tem como principal funcionalidade:
  - ***Get IR Map*:** retorna o mapa de infravermelho atual

- ***SceneAnalyzer*:** Um ***Generator*** que obtem dados brutos do sensor e retorna um ***map*** rotulando as camadas da cena. Suas princpais funcionalidades são:
  - ***Get Label Map:*** Fornece um *map* em que cada pixel é rotulado de acordo coma cena. Por exemplo, Ator1, Ator2, chão, e assim por diante.
  - ***Get Floor:*** retorna as coordenadas do ***floor plane*** (chão da cena).

- ***AudioGenerator*:** Objeto que retorna dados de áudio e possui como principal funcionalidade:

- ***Get Audio Buffer;***
- ***Wave Output Modes Properties:*** Configura a saída do áudio, incluindo sua taxa de amostragem e bits por amostra.

- ***GestureGenerator*****:** Objeto que possibilita o rastreamento de um corpo humano ou de uma mão em específico com o intuito de identificar um gesto. Suas principais funcionalidades são:
  - ***Add/Remove Gesture:*** Habilita ou desabilita um determinado gesto. Uma vez habilitado o nó irá observar toda vez que o gesto acontece.
  - ***Get Active Gesture:***Retorna todos os gestos disponíveis
  - ***Register/Unregister Gesture Callbacks:*** Habilita ou desabilita funções de retorno. Estas funções são chamadas quando o gesto ocorrer.

- ***HandsGenerator*****:** Objeto que permite o ratreamento de uma mão
  - ***Start/Stop Tracking:*** Inicia ou pára o rastreamento de uma mão;
  - ***Register/Unregister Hand Callbacks:*** Permite que funções de retorno sejam registradas ou desregistradas para monitorar o rastreamento da mão. Essas funções de retorno serão chamadas quando:
    - Uma nova mão for criada;
    - Uma mão existente trocas sua posição;
    - O rastreamento da mão for perdido (se a mão sumir)

- ***UserGenerator*****:** Objeto que retorna dados relacionados ao ator em cena. Suas principais funcionalidades são:
  - ***Get Number of Users:*** fornece o número de usuários detectados em cena
  - ***Get Users:*** Fornece o usuário atual;
  - ***Get User CoM:*** Fornece a localização do centro de massa do usuário
  - ***Get User Pixels:*** Fornece os pixels que representam o usuário. A informação de saída é uma matriz de pixels da cena (*map of pixels*), onde os pixels que representam o usuário são rotulados com sua identificação (*User ID*)
  - ***Register/Unregister user callbacks:*** Habilita ou desabilita o registro das funções de retorno. Estas funções são chamadas quando um novo usuário for identificado ou quando este desaparecer da cena.

2.5.2.Criando um Projeto Vazio usando o OpenNI

Esta parte do trabalho descreve como configurar o ambiente de desenvolvimento para se construir aplicações usando o framework *OpenNI*, utilizando a IDE Eclipse CDT e o sistema operacional *Linux Ubuntu*.

Para se iniciar o desenvolvimento de uma aplicação com o *framework OpenNI* iremos criar um projeto novo clicando na aba `File`, depois em `New`, e logo após em `C++ Project`**,** como ilustra a Figura 1.12.

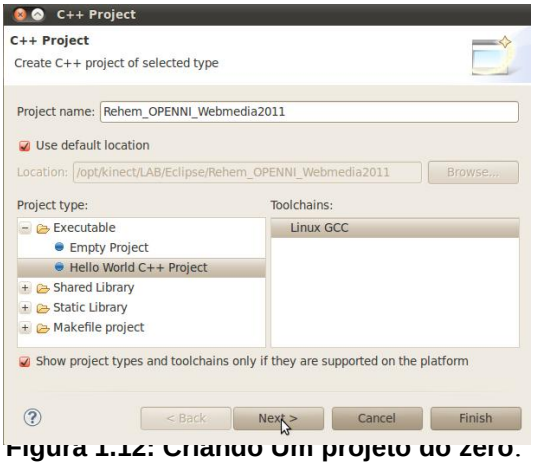

Figura 1.12: Criando um projeto do zero.

Em seguida, é necessário configurar as propriedades do projeto para que seja encontrado o caminho dos includes (arquivos `.h`) e das bibliotecas do *framework OpenNI* (arquivos `.so`), conforme ilustram as Figuras 1.Figura 1.13 e 1.Figura 1.14, respectivamente.

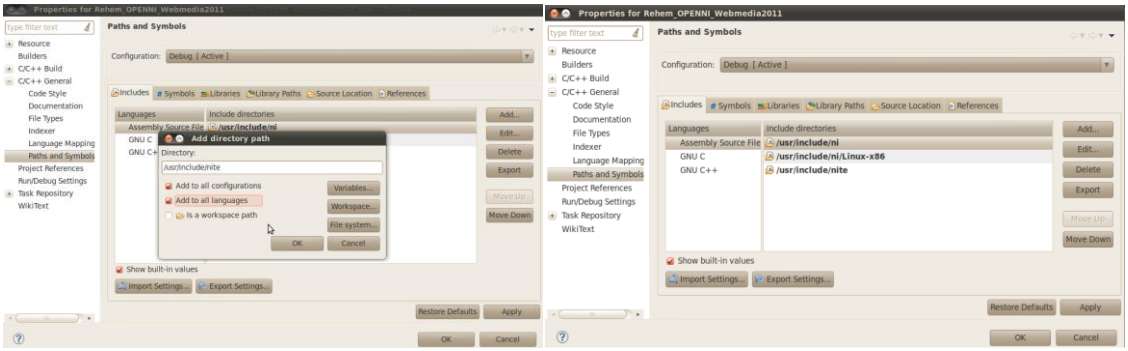

Figura 1.13. Configuração dos *includes* necessários para o *OpenNI*.

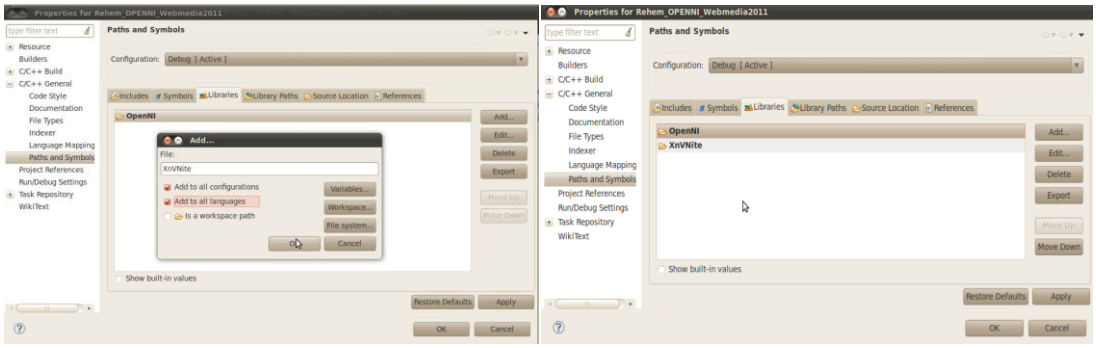

Figura 1.14. Configuração das bibliotecas necessárias o *OpenNI.*

É importante se certificar que os *Includes* e *Library directories* foram adicionados para as configurações de *Release* e de *Debug*.

Para iniciar a programação é necessário, que sejam incluídas as bibliotecas `XnOpenNI.h`, se for utilizar C, ou `XnCppWrapper.h`, se for utilizar C++ como linguagem de programação. A Figura a seguir ilustra um arquivo pronto para executar aplicações com objetos definidos pelo *framework OpenNI*.

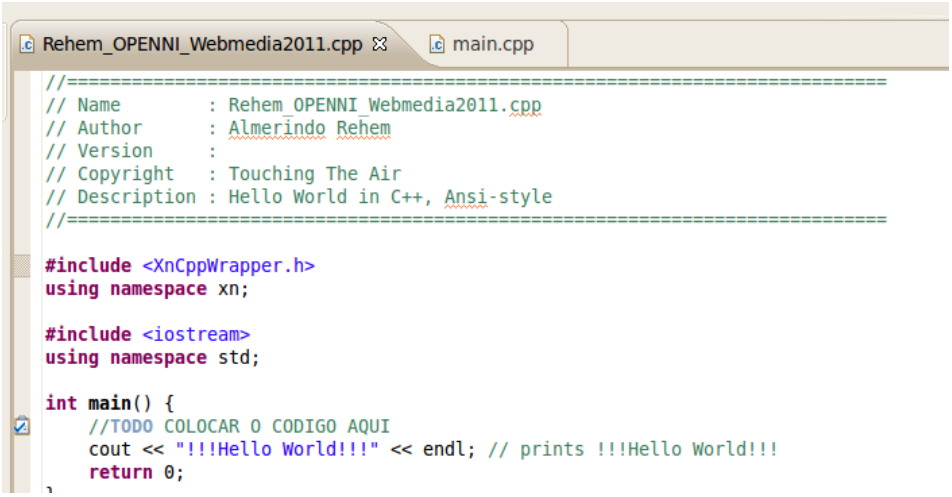

Figura 1.15: Código com include para o C++ (XnCppWrapper.h).

2.5.2.1. Funções Básicas: Iniciando, Criando Nós de Produção e Lendo Dados

Como visto na seção 2.5.1. Principais Objetos do Framework, a classe *Context* é o objeto principal de uma aplicação que usa o *OpenNI*. É a partir dela que são acessados todos os outros objetos, tais como os nós de produção do tipo profundidade, RGB, infravermelho, áudio, dentre outros. Os trechos de códigos que se segue mostram como usar o *framework OpenNI* através da instância da classe *Context,* além de criar e ler dados de um nó de produção (*DepthGenerator)*.

```
1  XnStatus nRetVal = XN_STATUS_OK;
2
3  xn::Context context;
4
5  // Initialize context object
6  nRetVal = context.Init();
7  // TODO: check error code
```

Trecho de Código 1.1. Instanciando a classe *Context*.

No código acima não existem nós de produção, ou seja, o contexto está vazio. Para se adicionar nós de produção à classe *Context*, é necessário criar esses nós dentro desta classe. O Trecho de Código 1.2. Criando um *Depth Generator* dentro de um *Context* mostra como criar um nó de produção do tipo *DepthGenerator* dentro de um *Context* já instanciado.

```
9  // Create a DepthGenerator node
10 xn::DepthGenerator depth;
11 nRetVal = depth.Create(context);
12 // TODO: check error code
13
```

Trecho de Código 1.2. Criando um *Depth Generator* dentro de um *Context*.

A partir do código acima, os nós de produção já estão plugados em um contexto, entretanto ainda não estão apitos a gerar conteúdo. Uma forma simples de fazer com que os *Generators* pertencentes a um *Context* gerem conteúdo é solicitar a este último

que inicie a produção de dados em todos os seus nós filhos. Este exemplo é descrito no Trecho de Código 1.3.

```
14  // Make it start generating data
15  nRetVal = context.StartGeneratingAll();
16  // TODO: check error code
17
```

Trecho de Código 1.3. Iniciando a produção de dados em todos os *Generators* filhos.

Com o objeto *Context* e seus nós de produção (neste caso, o *DepthGenerator*) instanciados e gerando conteúdos, se faz necessário capturar os dados todas as vezes que houver uma atualização. O método `context.WaitOneUpdateAll(depth)`, faz com que a classe *Context* espere por dados de um nó (*DepthGenerator*) e atualize todos os outros nós filhos. Com isso, basta capturar os dados (usando o `depth.GetDepthMap()`) e processá-los em seguida, conforme ilustrado no Trecho de Código 1.4.

```
18  // Main loop
19  while (bShouldRun)
20  {
21      // Wait for new data to be available
22      nRetVal = context.WaitOneUpdateAll(depth);
23      if (nRetVal != XN_STATUS_OK)
24      {
25          printf("Failed updating data: %s\n",
26  xnGetStatusString(nRetVal));
27          continue;
28      }
29
30      // Take current depth map
31      const XnDepthPixel* pDepthMap = depth.GetDepthMap();
32
33      // TODO: process depth map
34      }
```

Trecho de Código 1.4. Esperando por atualização e capturando dados.

Para liberar toda a memória utilizada nos trechos de códigos anteriores, é necessário chamar o método `Shutdown()` da classe ***Context*** conforme o código abaixo.

```
35
36  // Clean-up
37  context.Shutdown();
```

Trecho de Código 1.5. Liberando a memória utilizada.

2.5.2.2. A extração de esqueletos

O exemplo ilustrado nesta subseção mostra como detectar um usuário, monitora uma determinada pose do usuário, executa uma calibração quando este estiver fazendo uma pose específica e após calibração, segue a trajetória do usuário. Como saída, o sistema imprime em tela a localização da cabeça do usuário na cena. A Figura 1.16 ilustra as fases mais relevantes deste exemplo e permite um maior entendimento dos códigos descritos a seguir.

Figura 1.16. Funcionamento do exemplo de extração de esqueleto.

Para que seja possível executar as fases ilustradas na figura acima, é necessário se utilizar os recursos oferecidos pelo *framework* para monitorar os usuários, as poses e a calibração. A seguir, estes recursos serão discutidos, assim como sua utilização, através de alguns exemplos de códigos.

A classe *UserGenerator* oferece a possibilidade se registrar funções que são chamadas pelo próprio *framework* quando: (i) um novo usuário for detectado; (ii) o usuário corrente for perdido; (iii) uma pose for detectada; (iv) quando a calibração for iniciada e (v) quando a calibração for finalizada. O Trecho de Código 1.6 ilustra como criar as funções de retorno que alertam à aplicação quando um usuário for detectado e quando este sair de cena.

```
1  #define POSE_TO_USE "Psi"
2  xn::UserGenerator g_UserGenerator;
3
4  void XN_CALLBACK_TYPE
5  User_NewUser(xn::UserGenerator& generator, XnUserID nId, void* pCookie){
6    printf("New User: %d\n", nId);
7    g_UserGenerator.GetPoseDetectionCap().StartPoseDetection(POSE_TO_USE,nId);
8  }
9
10 void XN_CALLBACK_TYPE
11 User_LostUser(xn::UserGenerator& generator, XnUserID nId,void* pCookie){
12 }
13
```

Trecho de Código 1.6. Criando funções de retorno para detecção de usuários.

No Trecho de Código 1.7, a função `Pose_Detected()` será chamada quando uma pose for detectada. Ela irá imprimir na tela, o nome da pose e o identificador do usuário. Porém, é necessário desligar a detecção de pose (linha 17) e solicitar que seja feita a calibração do usuário (linha 18) para que seja possível extrair o seu esqueleto.

```
14  void XN_CALLBACK_TYPE
15  Pose_Detected(xn::PoseDetectionCapability& pose, const XnChar* strPose,XnUserID nId, void* pCookie){
16    printf("Pose %s for user %d\n", strPose, nId);
17    g_UserGenerator.GetPoseDetectionCap().StopPoseDetection(nId);
18    g_UserGenerator.GetSkeletonCap().RequestCalibration(nId, TRUE);
19  }
```

Trecho de Código 1.7. Criando função de retorno para detecção de pose.

A linha 18 do de código acima solicita a calibração do usuário. Entretanto é necessário se saber quando a calibração foi iniciada, quando ela foi finalizada e se houve sucesso após seu término. Estas informações são obtidas por meio da implementação das funções `Calibration_Start()` e `Calibration_End()`, ilustradas no Trecho de Código 1.8 a seguir.

```
21  void XN_CALLBACK_TYPE
22  Calibration_Start(xn::SkeletonCapability& capability, XnUserID nId,void* pCookie)
23  {
24    printf("Starting calibration for user %d\n", nId);
25  }
26
27  void XN_CALLBACK_TYPE
28  Calibration_End(xn::SkeletonCapability& capability, XnUserID nId, XnBool bSuccess, void* pCookie){
29    if (bSuccess) {
30      printf("User calibrated\n");
31      g_UserGenerator.GetSkeletonCap().StartTracking(nId);
32    } else {
33      printf("Failed to calibrate user %d\n", nId);
34      g_UserGenerator.GetPoseDetectionCap().StartPoseDetection( POSE_TO_USE, nId);
35      }
36  }
37
```

Trecho de Código 1.8. Criando funções de retorno para calibração.

A função `Calibration_End()` (linha 28), verifica se a calibração do usuário obteve êxito, exibe em tela uma mensagem indicando que a calibração foi feita e chama o método `GetSkeletonCap().StartTracking()` (linhas 29-31), responsável por atualizar o esqueleto do usuário conforme sua posição em cena. Por outro lado, se não houver sucesso na calibração, será exibida uma mensagem de erro e retornará a detectar uma pose para possibilitar uma nova calibração (linhas 32-34).

Com as funções de retorno implementadas, é necessário que o objeto *UserGenerator* as reconheça no seu sistema de alerta. Para isso, as funções de retorno devem ser registradas através dos métodos `RegisterUserCallbacks()`, `RegisterToPoseCallbacks()`, `RegisterCalibrationCallbacks()`, conforme o código abaixo (linhas 50-53).

```
38  void main()
39  {
40    XnStatus nRetVal = XN_STATUS_OK;
41
42    xn::Context context;
43    nRetVal = context.Init();
44    // TODO: check error code
45
46    // Create the user generator
47    nRetVal = g_UserGenerator.Create(context);
48    // TODO: check error code
49
50    XnCallbackHandle h1, h2, h3;
51    g_UserGenerator.RegisterUserCallbacks(User_NewUser, User_LostUser,NULL, h1);
52    g_UserGenerator.GetPoseDetectionCap().RegisterToPoseCallbacks( Pose_Detected, NULL, NULL, h2);
53    g_UserGenerator.GetSkeletonCap().RegisterCalibrationCallbacks( Calibration_Start, Calibration_End, NULL, h3);
54
55    // Set the profile
56     g_UserGenerator.GetSkeletonCap().SetSkeletonProfile( XN_SKEL_PROFILE_ALL);
57    // Start generating
58    nRetVal = context.StartGeneratingAll();
59    // TODO: check error code
```

Trecho de Código 1.9. Registrando as funções de retorno no *UserGenerator*.

No exemplo a seguir, a quantidade máxima de usuários foi definida como sendo igual a quinze (linha 68). A partir deste ponto, todos os usuários em cena serão capturados por meio da função `GetUsers()` (linha 69). Em seguida, para cada um dos usuários que já estão sendo seguidos em cena (linhas 70 e 71), será extraída a posição do osso da cabeça deste usuário (linhas 72 e 73), sendo exibidas a sua identificação e a posição de sua cabeça (linhas 74-76).

```
60
61    while (TRUE)
62    {
63      // Update to next frame
64      nRetVal = context.WaitAndUpdateAll();
65      // TODO: check error code
66      // Extract head position of each tracked user
67      XnUserID aUsers[15];
68      XnUInt16 nUsers = 15;
69      g_UserGenerator.GetUsers(aUsers, nUsers);
70      for (int i = 0; i < nUsers; ++i){
71        if (g_UserGenerator.GetSkeletonCap().IsTracking(aUsers[i])){
72          XnSkeletonJointPosition Head;
73          g_UserGenerator.GetSkeletonCap().GetSkeletonJointPosition( aUsers[i], XN_SKEL_HEAD, Head);
74          printf("%d: (%f,%f,%f) [%f]\n", aUsers[i],
75                  Head.position.X, Head.position.Y,
76                  Head.position.Z, Head.fConfidence);
77        }
78      }
79    }
80
```

Trecho de Código 1.10. Capturando a cabeça do usuário e exibindo sua posição.

Por fim, é necessário liberar toda a memória utilizada pelo programa com o uso da função `Shudown()`, conforme o Trecho de Código 1.11.

```
80
81    // Clean up
82    context.Shutdown();
83  }
```

Trecho de Código 1.11. Liberação da memória antes de sair do programa.

2.6. Conclusão

Este capítulo tratou do desenvolvimento de aplicações voltadas para interação natural utilizando o *framework OpenNI e o middleware NITE*. A intenção não foi esgotar o assunto, mas levantar questões importantes e divulgar as tecnologias associadas à Interação Natural. Foram apresentados conceitos de Interação Natural, comunicação verbal e não verbal, dispositivos envolvidos e sua evolução, assim como uma explicação mais detalhada sobre o *Kinect* e toda a motivação atual para as pesquisas relacionadas a este dispositivo. As seções finais do capítulo foram dedicadas ao *framework OpenNI* e à sua aplicação em um estudo de caso que mostrou o desenvolvimento, passo a passo, de aplicações em C++ para interação natural.

Como foi observado no capítulo, o desenvolvimento de aplicações para interação natural traz consigo vários desafios e oportunidades para os desenvolvedores de software e produtores de dispositivos de hardwares. Atualmente, nota-se uma tendência de utilização da plataforma *Kinect* não apenas na sua função original de controle de jogos, mas também como um novo periférico de entrada para comunicação usuário computador. Por outro lado, a área de IUC passa por um momento em que as metáforas de usabilidade estão se atualizando, e a interação natural usuário computador recebe um reforço importante na busca da ubiquidade computacional, simplificando a forma de fornecer entradas de dados para software e utilizar dispositivos tecnológicos em geral.

Desta maneira, o *OpenNI* está se tornando a principal aposta dos desenvolvedores de aplicações e comunidades científicas de interação natural usuário computador. Não só por ser um excelente *framework*, mas por ter se tornado uma plataforma padrão (*de facto*) de desenvolvimento para aplicações de interação natural. O projeto *OpenNI* abordado nesse capítulo oferece uma padronização de desenvolvimento, uma abstração maior dos elementos de hardware e uma facilidade de integração de outros componentes de softwares para interpretação dessas informações. Isto que permite tanto reuso das implementações feitas, quanto a leitura conjunta de sensores de mesma natureza aumentando a acuidade da informação.

2.7. Agradecimentos

Os autores agradecem à CAPES pelo suporte financeiro ao primeiro autor através da bolsa do Programa de Demanda Social para o Programa Multiinstitucional de Pós-graduação em Ciência da Computação UFBA/UNIFACS/UEFS (28001010061P1).

2.8. Referências

Abowd G.D, Mynatt E. D. (2000) “Charting past, present, and future research in ubiquitous computing.”ACM Transactions on Computer-Human Interaction. v.7, n.1, Mar. 2000.

Adafruit Industries (2011), “Open Kinect Challenge”. Disponível em *URL*: http://www.adafruit.com/blog/2010/11/04/the-open-Kinect-project-the-ok-prize-get-1000-bounty-for-*Kinect*-for-xbox-360-open-source-drivers/

Cordeiro Jr. A. A. A. (2009) “Modelos e Métodos para Interação Homem-Computador Usando Gestos Manuais”. *Tese de Doutorado*. LNCC, Nov. 2009. *URL*: http://www.lncc.br/tdmc/tde_busca/arquivo.php?codArquivo=195

Eclipse CDT Project (2011) “Eclipse CDT (C/C++ Development Tooling)”. *URL*: http://www.eclipse.org/cdt/

Gallud, J.A.; Villanueva, P. G.; Tesoriero, R.; Sebastián, G.; Molina S.; A. Navarrete (2010). “Gesture-based Interaction. Concept Map and Application Scenarios”. *DOI: 10.1109/CENTRIC.2010.10*

Heckel, P. (1993) “Software amigável. Técnicas de projeto de software para uma melhor interface com o usuário”. Ed. Campus Ltda.

Crawford, S. (2010) “How Microsoft Kinect Works”. 13/07/10. HowStuffWorks.com. *URL*: http://electronics.howstuffworks.com/microsoft-kinect.htm

Kinect (2010) “Kinect Web Site”. Microsoft. Disponível em *URL*: http://www.xbox.com/pt-br/Kinect

Kinect SDK (2010) “Kinect for Windows SDK”. Microsoft Research. Disponível em URL: http://research.Microsoft.com/en-us/um/redmond/projects/Kinectsdk/default.aspx

Kinecthacks (2011). “Hacks para o Kinect*”*. Disponível em *URL*: http://Kinecthacks.net/.

Kyma (2010). “Using Nitendo Wiimote With Kyma”. Kima Twiky. Disponível em *URL*: http://www.symbolicsound.com/cgi-bin/bin/view/Learn/UsingTheNintendo WiimoteWithKyma

Nakra T., Ivanov Y., Smaragdis P., Ault C. (2009) “The USB Virtual Maestro: an Interactive Conducting System”, pp.250-255, *In:* NIME2009. Disponível em *URL*: www.cs.illinois.edu/~paris/pubs/nakra-nime09.pdf

OpenNI (2011) “Introducing OpenNI”. OpenNI Organization. Disponível em URL:http://www.OpenNI.org

OpenNI User Guide (2011). Disponível em *URL*: www.openni.org/images/stories/pdf/OpenNI_UserGuide.pdf

Power Glove (2008) “Games: Manual de Instruções Power Glove”, Nintendo. Disponível em *URL*: http://blablagames.net/?p=1006

PrimeSense (2011). “PrimeSense Natural Interaction”. PrimeSense. Disponível em *URL*: http://www.primesense.com/

Projeto OpenKinect (2011) “Open Kinect Roadmap”. OpenKinect Org. Disponível em *URL*: http://openKinect.org/wiki/Roadmap.

Saffer (2009) “*Designing Gestual Interfaces*”. D. Saffer, Books, O’Reilly. 2009.

Shape Quest in (2010) “Point Cloud Data from Structured Light Scanning”. Disponível em *URL*: http://www.shapecapture.com/gallery3.htm

Shiratori T.; Hodgins J. K. (2008) “Accelerometer-based user interfaces for the control of a physically simulated character”. *ACM Trans. Graph*. 27, 5, Article 123, Disponível em URL: http://doi.acm.org/10.1145/1409060.1409076.

Toyama K. (1998) “Look, Ma – No Hands! Hands-Free Cursor Control with Real-Time 3D Face Tracking”. *URL*: http://www.cs.ucsb.edu/conferences/PUI/PUIWorkshop98/Papers/Toyama.pdf

Valli A. (2007) “Natural Interaction White Paper”, Disponível em *URL*: http://www.naturalinteraction.org/

Weiser M.; Brown J. S. (1995) “Designing calm technology”. Xerox PARC.

Wii Remote (2009) “Wii Console Controllers”. Nintendo. Disponível em *URL*: http://www.nintendo.com/wii/console/controllers

Wiimote (2010) “Wiiremote Commander”. Disponível em *URL*: http://wiimotecommande.sourceforge.net/wiimote/index.html