UNIVERSIDADE DE CAXIAS DO SUL
CENTRO DE COMPUTAÇÃO E TECNOLOGIA DA INFORMAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

FERNANDO CASTILHOS MELO

Kinected Blender: Uma solução para criação de animações 3D utilizando Kinect e *Blender*

André Luis Martinotto
Orientador

Gelson Cardoso Reinaldo
Coorientador

Caxias do Sul, Julho de 2014

Kinected Blender: Uma solução para criação de animações 3D utilizando Kinect e *Blender*

por

Fernando Castilhos Melo

Projeto de Diplomação submetido ao curso de Bacharelado em Ciência da Computação do Centro de Computação e Tecnologia da Informação da Universidade de Caxias do Sul, como requisito obrigatório para graduação.

Projeto de Diplomação

Orientador: André Luis Martinotto
Coorientador: Gelson Cardoso Reinaldo
Banca examinadora:

André Gustavo Adami
CCTI/UCS
Carlos Eduardo Nery
CCTI/UCS

Projeto de Diplomação apresentado em
1 de Julho de 2014

Daniel Luís Notari
Coordenador

SUMÁRIO

LISTA DE ACRÔNIMOS 4

LISTA DE FIGURAS 5

LISTA DE TABELAS 7

RESUMO 8

ABSTRACT 9

1 INTRODUÇÃO 10
1.1 Objetivos do Trabalho 11
1.2 Estrutura do Trabalho 11

2 ANIMAÇÃO 12
2.1 Tipos de Animação 14

3 BLENDER 16
3.1 *Armature* 16
3.1.1 *Bones* 17
3.1.2 *Bone Constraints* 18
3.2 Animação no *Blender* 19
3.2.1 Interpolação e Extrapolação 20
3.3 *Addons* 21

4 KINECT 23
4.1 Imagem de Profundidade 25
4.2 Dados de Esqueleto 27

5 DESENVOLVIMENTO PARA KINECT 28
5.1 OpenNI 28

5.1.1 NiTE . . . 30
5.1.2 Python Bindings . . . 32
5.2 Microsoft Kinect *SDK* (*Software Development Kit*) . . . 32
5.2.1 PyKinect . . . 33
5.3 Testes dos *SDK*s . . . 34
5.3.1 Comparativo entre os *SDK*s . . . 36

6 APLICAÇÕES DE INTEGRAÇÃO . . . 38
6.1 Delicode Ni Mate . . . 38
6.2 Bloop . . . 41
6.3 Comparativo entre as Ferramentas . . . 42

7 IMPLEMENTAÇÃO E TESTES . . . 43
7.1 Arquitetura da Aplicação . . . 43
7.2 Comunicação . . . 45
7.3 Addon Kinected Blender . . . 46
7.3.1 Mapeamento para *Armature* / *Bones* . . . 47
7.4 Kinected Blender Receiver . . . 48
7.5 Testes Realizados . . . 49
7.6 Animação . . . 51

8 CONCLUSÕES . . . 52
8.1 Trabalhos Futuros . . . 53

REFERÊNCIAS . . . 54

LISTA DE ACRÔNIMOS

3D	3 Dimensões
API	*Application Programming Interface*
Bloop	*Blender Loop Station*
DLL	*Dynamic-Link Library*
GNU GPL	GNU *General Public License*
Ni Mate	*Natural Interaction Mate*
NiTE	*Natural Interface Technology for End-User*
NUI	*Natural User Interface*
OpenNI	*Open Natural Interaction*
OSC	*Open Sound Control*
RGB	*Red, Green, Blue*
SDK	*Software Development Kit*
TCP	*Transmission Control Protocol*
UDP	*User Datagram Protocol*
USB	*Universal Serial Bus*

LISTA DE FIGURAS

Figura 2.1: Taumatrópio que demonstra a união de duas imagens distintas. . 12
Figura 2.2: Exemplo de funcionamento de um fenacistoscópio. 13
Figura 2.3: Modelo de um zootrópio. 13
Figura 2.4: Início e fim dos quadros de uma animação com técnica em linha reta. 14
Figura 2.5: Início e fim dos quadros de uma animação com técnica pose a pose. 14

Figura 3.1: Exemplo de *armature* no *Blender*. 17
Figura 3.2: Função dos ossos em um *armature*. 18
Figura 3.3: *F-Curve* referente à movimentação da câmera nos eixos y e z. . . 19
Figura 3.4: Visualização de uma *timeline* no *Blender* durante a animação de um cubo. 20
Figura 3.5: Exemplo de informações de um *addon* no *Blender*. 22

Figura 4.1: Kinect sem o revestimento de plástico, observando-se o projetor infravermelho (1), a câmera RGB (2) e a câmera infravermelha (3) 24
Figura 4.2: Sombra gerada pela luz infravermelha. 25
Figura 4.3: Imagem de profundidade em escalas de amarelo. 25
Figura 4.4: Geração da imagem de profundidade 26
Figura 4.5: Esqueleto no modo normal e sentado, respectivamente. 27

Figura 5.1: Arquitetura do *SDK* OpenNI 29
Figura 5.2: Pose "Psi" para calibragem e detecção do usuário. 30
Figura 5.3: Pontos gerados através da detecção do usuário realizada pelo NiTE. 31
Figura 5.4: Esqueleto gerado através do Microsoft Kinect *SDK* no modo em pé. 33
Figura 5.5: Ambiente em que os testes dos *SDK*s foram feitos. 34
Figura 5.6: Tratamento de área inalcançável no NiTE (a) e no Microsoft Kinect *SDK* (b). 35
Figura 5.7: O Microsoft Kinect *SDK* detecta o usuário mas não consegue gerar o esqueleto à uma curta distância. 35

Figura 5.8: O Microsoft Kinect *SDK* não consegue capturar corretamente a posição das mãos em alguns casos. 36

Figura 6.1: Aba de configuração do esqueleto do Ni Mate. 39
Figura 6.2: Aba de configuração do controlador *OSC* (*Open Sound Control*) do Ni Mate. 39
Figura 6.3: *Addon* do *Blender* para a recepção de mensagens enviadas do Ni Mate. 40
Figura 6.4: *Armature* com *bones* nomeados, assim como o esqueleto gerado pelo Ni Mate. 40
Figura 6.5: Modelagem de um canguru no *Blender*, com os *control bones* utilizados para a animação com o Bloop. 41

Figura 7.1: Arquitetura inicial da aplicação Kinected Blender. 44
Figura 7.2: Nova arquitetura da aplicação Kinected Blender. 45
Figura 7.3: *Addon* Kinected Blender ativado. 46
Figura 7.4: A função *Auto keyframe insertion* deve estar ativada para gravação da animação. 47
Figura 7.5: *Armature* cinza mapeado no personagem e *armature* rosa mapeada do usuário. 48
Figura 7.6: Modelo 3D utilizado na animação. 51

LISTA DE TABELAS

Tabela 5.1: Tabela de comparação de recursos dos *SDK*s. 37

Tabela 6.1: Tabela de comparação das ferramentas de integração entre Kinect e *Blender*. 42

Tabela 7.1: Tabela para mapeamento de objetos vinculados às partes do corpo. 47

Tabela 7.2: Tabela de gestos para reconhecimento. 50

RESUMO

A geração da animação de um personagem 3D é complexa e custosa, especialmente para animadores que não possuem uma equipe que auxilie na produção. Isso deve-se ao fato do animador necessitar mudar a pose do personagem a cada quadro gravado, gastando-se muito tempo e gerando muito trabalho para a produção da animação.

Com a utilização do *Blender* como *software* de modelagem e animação 3D, e do Kinect como sensor de movimentos, pode-se desenvolver uma aplicação que integre esses dois recursos com o objetivo de facilitar o processo de criação de animações 3D. De fato, essa aplicação pode utilizar os movimentos capturados pelo Kinect e transmiti-los para o *Blender*, gerando assim uma animação baseada nos movimentos do usuário.

O *Blender* é um *software* de modelagem e animação 3D, sendo assim, é possível realizar a animação de um personagem através do uso desse *software*. Mesmo em pequenas animações, nota-se uma complexidade particular de cada cena e percebe-se o quão trabalhoso é criar uma animação que tente manter a fidelidade da linguagem corporal do ser humano. Já o Kinect é um sensor capaz de capturar movimentos, e através dessa captura pode-se obter a linguagem corporal das pessoas. Sendo assim, essa linguagem corporal pode ser utilizada para mover um personagem em uma animação 3D, preservando-se a naturalidade do movimento.

Atualmente, algumas aplicações já realizam essa integração entre Kinect e *Blender* para auxiliar o processo de animação 3D. Entretanto, essas aplicações não são gratuitas ou estão desatualizadas, não permitindo que todos os animadores tenham acesso a uma ferramenta atual, gratuita e multiplataforma.

Desta forma, neste trabalho foi desenvolvida uma aplicação que permite a integração entre Kinect e *Blender*, de forma gratuita, multiplataforma e atual. Para tanto, foi utilizado o *framework* OpenNI e o *middleware* NiTE. Além disso, foi desenvolvida uma animação 3D a partir da aplicação desenvolvida.

Palavras-chave: Kinect, Blender, Animação 3D.

Kinected Blender: A solution to create 3D animations using Kinect and Blender

ABSTRACT

Generating a 3D character animation is complex and costly, particularly for animators who do not have a team to assist in its production. This is due to the fact the animator need to change the pose of the character for each recorded frame, spending a lot of time and generating a lot of work to produce the animation.

Using Blender as a modeling and 3D animation software, and Kinect as motion sensor, can be build an application that integrates these two resources in order to facilitate the process of creating 3D animations. This application can use the movements captured by Kinect and transmit them for Blender, creating an animation based on the user's movements.

Blender is a 3D modeling and animation software, so you can make the animation of a character using this software. Even in small animations, there is a particular complexity of each scene and it is possible to realize how much work is needed to create an animation that represents the fidelity of human body language. Kinect is a sensor capable of capturing movements, and through this capture you can obtain the people's body language. So this body language can be used to move a character in 3D animation, preserving a natural movement.

There are some applications already perform this integration between Kinect and Blender to assist the process of 3D animation. However, these applications are non-free or outdated, not allowing all animators have access to a current, free, cross-platform tool.

Therefore, this paper was developed an application that enables integration between Kinect and Blender, in a free, cross-platform and current way. For this purpose the OpenNI framework and NITE middleware was used. Beyond that a 3D animation was produced using the developed solution.

Keywords: Kinect, Blender, 3D Animation.

1 INTRODUÇÃO

O Kinect é um sensor de captura de movimentos que foi desenvolvido pela Microsoft e que é utilizado em alguns jogos do console Xbox 360. Esse é um sensor que responde aos movimentos do usuário sendo desnecessário o uso de *joysticks*, ou seja, esse permite que o usuário interaja com o console através dos movimentos do próprio corpo (BORENSTEIN, 2012).

A partir do desenvolvimento do Kinect, essa tecnologia que é fruto de altos investimentos e décadas de pesquisa e que até então só estava ao alcance de militares, tornou-se acessível às pessoas comuns (BORENSTEIN, 2012). Pela primeira vez os computadores passaram a ter a capacidade de reconhecer a linguagem corporal e distinguir objetos. De fato, através do Kinect, essa tecnologia que inicialmente era utilizada, por exemplo, para a detecção de terroristas em um espaço público, passou a ser usada para a captura de movimentos em diferentes aplicações. Inclusive os movimentos capturados podem ser utilizados para a criação de animações para personagens 3D e construir interfaces gestuais para *softwares* (BORENSTEIN, 2012).

Existem diversas formas para a criação de uma animação de um personagem 3D, sendo que a forma mais comum é a utilização da gravação por quadros (MASHALKAR, 2012). Porém, essa forma de animação é muito trabalhosa, principalmente para animadores que trabalham sozinhos ou com equipes pequenas. Entretanto, a utilização da captura de movimentos para a criação de uma animação permite que o movimento do personagem seja mais real e natural. Além disso, esse processo leva menos tempo para ser realizado se comparado à utilização de quadros.

O *Blender* (Blender Foundation, 2013) é uma ferramenta gratuita e livre para modelagem e animação 3D, que suporta tanto a animação por quadros quanto por captura de movimentos. Entretanto, não existe nenhuma integração nativa entre o *Blender* e o Kinect. Desta forma, neste trabalho foi desenvolvida uma solução que permite a integração do sensor Kinect com o *software Blender*, ou seja, a solução desenvolvida possibilita que os movimentos capturados pelo Kinect sejam mapeados em um ambiente do *Blender* e sejam utilizados para a criação de uma animação.

1.1 Objetivos do Trabalho

O objetivo principal deste trabalho consistiu no desenvolvimento de uma solução que permitisse a integração entre o *Blender* e o sensor Kinect. A partir dessa integração facilitou-se o processo de criação de animações 3D no *Blender*.

De forma a atingir o objetivo principal desse trabalho, os seguintes objetivos específicos foram realizados:

- Implementação de uma solução multiplataforma que permite a integração entre o sensor Kinect e o *software Blender*;
- Geração de uma animação 3D a partir da solução desenvolvida.

1.2 Estrutura do Trabalho

O presente trabalho está estruturado da seguinte forma: no Capítulo 2 será apresentada a história da animação, seu surgimento e conceitos. No Capítulo 3 são apresentadas informações sobre o *software Blender*, assim como os métodos disponíveis nesse para a criação de animações 3D. No Capítulo 4 será apresentado o sensor Kinect, assim como seu funcionamento e alguns conceitos sobre as imagens capturadas. No Capítulo 5 serão apresentadas as bibliotecas e ferramentas para o desenvolvimento de aplicações utilizando Kinect. Além disso, é realizado um comparativo, onde são apresentadas as vantagens e desvantagens de cada uma dessas ferramentas. No Capítulo 6 serão apresentados alguns *softwares* que já fazem a integração entre o sensor e a ferramenta, assim como o funcionamento desses. No Capítulo 7 serão apresentados os detalhes da solução desenvolvida e da animação gerada, bem como são apresentados os testes realizados e os resultados obtidos. Por fim, no Capítulo 8 são apresentadas as considerações finais e sugestões de trabalhos futuros.

2 ANIMAÇÃO

Uma animação consiste na criação de uma série de imagens que, quando vistas rapidamente em sequência, geram uma ilusão de ótica, causando a sensação de movimento (MULLEN, 2011). Esta técnica é utilizada desde os primeiros desenhos animados, sendo aprimorada ao longo dos anos.

As primeiras ideias de animação originaram-se com a criação do taumatrópio em 1780 (BECKERMAN, 2012). O taumatrópio era um brinquedo composto por um disco de papelão, com duas figuras distintas em cada um dos lados, e por dois elásticos presos nas laterais do disco. Quando estes elásticos eram torcidos, o disco sofria uma rotação e causava uma ilusão de ótica, dando a impressão que a imagem havia se tornado uma só, conforme pode-se observar na Figura 2.1.

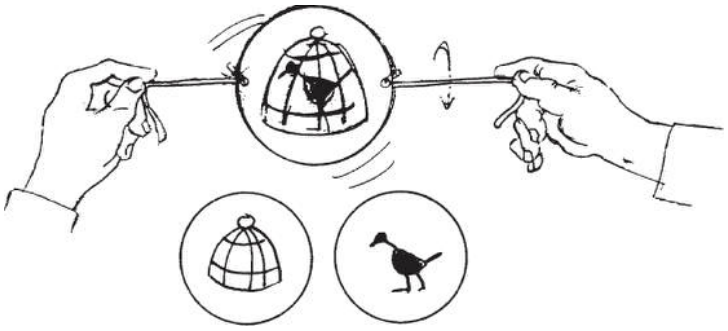


Figura 2.1: Taumatrópio que demonstra a união de duas imagens distintas.
Fonte: BECKERMAN (2012)

Em 1832 foi criado um disco chamado fenacistoscópio, dando origem às primeiras animações (BECKERMAN, 2012). Este disco possuía diversas imagens similares em sequência ao longo de sua circunferência e fendas no disco separando as imagens. Quando este disco era girado em torno de seu eixo em frente a um espelho, causava a sensação de movimento das imagens (Figura 2.2).

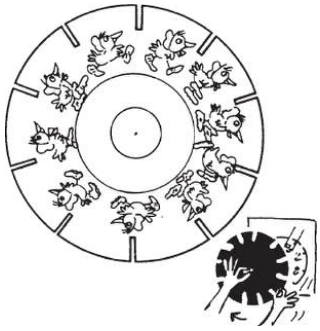


Figura 2.2: Exemplo de funcionamento de um fenacistoscópio.
Fonte: BECKERMAN (2012)

O zootrópio (Figura 2.3), também conhecido como roda-da-vida, foi outro instrumento criado na mesma época, mas ao contrário do fenacistoscópio, no zootrópio não se necessitava de um espelho para visualizar a animação (BECKERMAN, 2012). O zootrópio era uma espécie de copo com fendas nas bordas. A sequência dos desenhos era disposta dentro do copo, e, quando rotacionado, era possível visualizar a animação através das fendas.

Figura 2.3: Modelo de um zootrópio.
Fonte: BECKERMAN (2012)

Houveram diversas evoluções tecnológicas na época, como por exemplo o surgimento da fotografia e, a partir desta, surgiram os filmes que não passavam de várias fotos em sequência (BECKERMAN, 2012). O surgimento da indústria cinematográfica fez com que não somente filmes, mas também desenhos animados ficassem mais populares (BECKERMAN, 2012).

Devido a demanda de oportunidades, era muito trabalhoso somente um artista desenhar todos os quadros de uma animação, processo este conhecido como animação manual (MULLEN, 2011). Sendo assim, a animação manual sofreu uma evolução com o passar do tempo. Os animadores passaram a desenhar o personagem somente em alguns pontos cruciais, enquanto assistentes desenhavam os quadros intermediários entre cada um desses pontos, completando o processo de animação. Estes pontos cruciais são chamados de *keyframes* (MULLEN, 2011).

2.1 Tipos de Animação

Em uma animação tradicional existem dois métodos para a criação da mesma, que são: a animação em linha reta ou a animação pose a pose. A animação em linha reta (Figura 2.4) é basicamente ir do início ao fim da animação, gerando manualmente cada movimento, contendo assim uma grande quantidade de detalhes e criando um movimento mais natural e expressivo, desde que seja gerado um número de quadros adequado (VASCONCELOS, 2011).

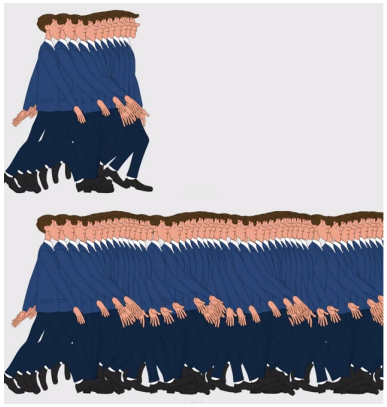


Figura 2.4: Início e fim dos quadros de uma animação com técnica em linha reta.
Fonte: Gilles Charbonneau (2012)

Já a animação pose a pose (Figura 2.5) consiste em gravar a posição dos personagens e objetos da cena em diferentes momentos (*keyframes*), gerando quadros intermediários posteriormente (MULLEN, 2011). Enquanto a animação em linha reta provê maior naturalidade na movimentação, a animação pose a pose dispõe de uma flexibilidade maior no tempo e posicionamento dos objetos da cena, além de uma chance maior de manter a proporção dos personagens (MULLEN, 2011).

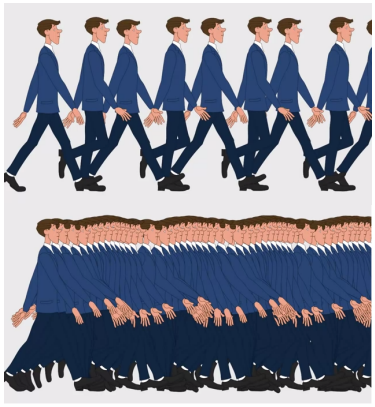


Figura 2.5: Início e fim dos quadros de uma animação com técnica pose a pose.
Fonte: Gilles Charbonneau (2012)

Observa-se que o melhor método de se produzir animação é um método híbrido entre estes dois tipos, preservando a naturalidade dos movimentos obtida pela

animação em linha reta e obtendo a flexibilidade e a proporção obtidas pela animação pose a pose.

Nos dias atuais, as técnicas de pose a pose e linha reta ainda são utilizadas na produção de animações, entretanto, com a evolução tecnológica, nem sempre o papel e lápis são utilizados como base do desenho. Muitos artistas utilizam o computador tanto para desenhar quanto para animar seus personagens e cenas, embora alguns ainda prefiram o método tradicional de desenho (UNIVESP, 2012). Os esboços dos personagens ainda são feitos no papel e só depois transferidos para o computador, principalmente, em filmes de animação 3D (Jornal da Globo, 2012). Estes filmes, além de serem trabalhosos para a equipe, necessitam de muito tempo, equipamentos e horas de processamento, dependendo da complexidade da animação. Como exemplo, pode-se citar o filme "Rio" que, devido a sua complexidade, levou 3 anos para ser produzido (Jornal da Globo, 2012).

3 BLENDER

O *Blender* é um *software* de código aberto e gratuito, que possui uma licença GNU *GPL* (GNU *General Public License*), e que é utilizado para criação de modelos, animações e jogos 2D ou 3D (Blender Foundation, 2013). Este *software* é utilizado por estudantes, artitas que trabalham com modelagem e animação, ou até mesmo por equipes que produzem filmes animados como, por exemplo, os filmes *Sintel* e *Big Buck Bunny* (Blender Wiki, 2013).

Assim como outros *softwares* utilizados para modelagem e animação, como por exemplo os *softwares 3ds Max* (Autodesk, 2013a) e *Maya* (Autodesk, 2013b), o *Blender* oferece uma grande quantidade de ferramentas e recursos que possibilita. a criação de cenas estáticas ou dinâmicas (MULLEN, 2011).

O *Blender* foi escrito na linguagem de programação Python e, através da utilização da mesma, possibilita a implementação de *addons*, que permitem a adição de novas funcionalidades sem a necessidade de alteração no código fonte. Neste capítulo serão apresentadas alguns recursos utilizados para a criação de animações nesta ferramenta.

3.1 *Armature*

O *Blender* possibilita que seja feita a animação de personagens 3D através do uso de um recurso chamado *armature*. No *Blender*, um personagem é formado por uma malha de vértices, arestas e faces, formando a aparência do mesmo. Este personagem é uma figura estática, ou seja, qualquer alteração na malha do personagem resulta em uma alteração no modelo. De forma a alterar a pose do personagem, sem modificar a malha de modelagem, utiliza-se o conceito de *armature*.

Um *armature* é uma espécie de esqueleto do personagem ou objeto, composto por *bones* e seus relacionamentos, conexões e restrições, que permite com que uma parte do modelo 3D seja movimentado sem que haja uma distorção ou deformação na imagem original (MULLEN, 2011). Na Figura 3.1 pode-se visualizar um exemplo de um *armature* do *Blender*.

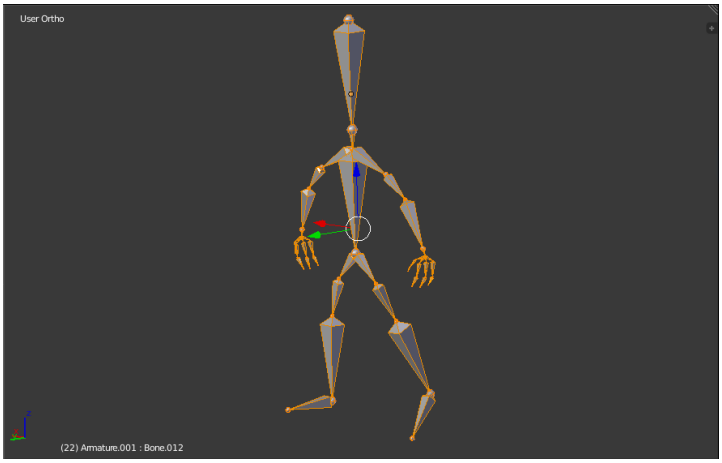


Figura 3.1: Exemplo de *armature* no *Blender*.

3.1.1 *Bones*

Os *bones* de um *armature* são os ossos que compõe o esqueleto de um modelo 3D. O número de ossos presentes em um *armature* varia de acordo com a precisão e a naturalidade do movimento desejado pelo animador (MULLEN, 2011). Em geral, dentro de um *armature*, todos os ossos são conectados uns aos outros, gerando um relacionamento hierárquico entre eles. Embora todos os *bones* possuam a mesma aparência e propriedades, cada um possui pelo menos um papel a desempenhar na animação do personagem. Na Figura 3.2 tem-se um exemplo de um *armature* com o papel de cada um dos *bones*. Já abaixo, tem-se uma descrição das funcionalidades de cada um deles:

- *Control Bones*: são ossos que controlam o posicionamento do esqueleto na cena. Em geral, cada personagem possui pelo menos um osso deste tipo, que é usado como referência dentro da cena, ou seja, através desse osso pode-se mover o personagem, assim como alterar sua pose (MULLEN, 2011).
- *Deform Bones*: são ossos que aplicam mudanças na posição da malha de modelagem, ou seja, fazem com que seja possível movimentar e rotacionar partes específicas de um personagem (MULLEN, 2011). Embora o nome sugira que estes ossos deformem o modelo 3D, estes apenas mudam a posição das faces da malha, mantendo a proporção da modelagem e fazendo com que a "deformação" seja referente apenas à mudança da pose original do personagem.
- *Function Bones*: são ossos utilizados tanto para adicionar restrições na movimentação do personagem, bem como para determinar uma reação em cadeia a partir de um movimento de um determinado osso (MULLEN, 2011). Como exemplo, pode-se citar o movimento de uma perna. A movimentação do osso da canela do personagem pode causar uma reação no osso da coxa, fazendo com que ambos se movimentem. Também é possível adicionar um osso para

restringir para que o movimento do osso da canela não ultrapasse o limite do joelho, evitando a sensação de "perna quebrada" no personagem.

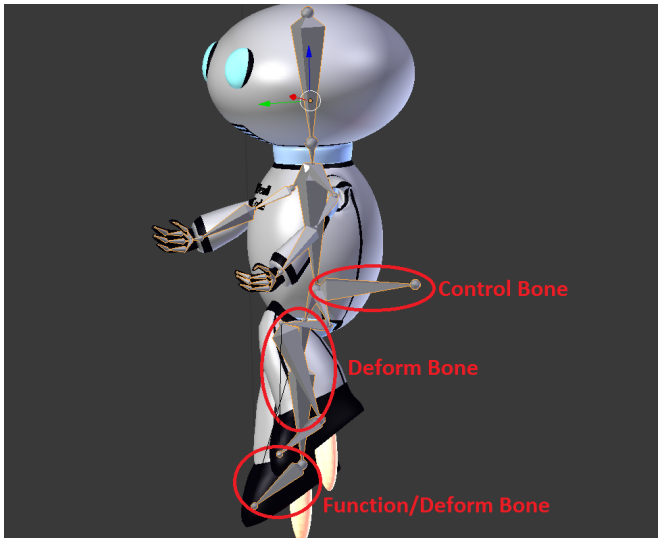


Figura 3.2: Função dos ossos em um *armature*.

3.1.2 *Bone Constraints*

Os *bones constraints* são restrições e controles que fazem com que o movimento de um osso dependa de outro (MULLEN, 2011). Esta propriedade é a principal característica de um *Function Bone* e é de extrema importância para que o movimento do personagem pareça real, além de auxiliar o animador na criação da cena. Por exemplo, o andar de um personagem pode ser algo complexo sem a utilização de uma *bone constraint*. Supondo que cada perna do *armature* de um personagem possua apenas 3 ossos (coxa, canela e pé), o animador deverá posicionar estes 3 ossos manualmente de forma a criar uma sensação de um movimento natural da perna, além de ter que se preocupar com a questão do personagem não "flutuar" ou "afundar" no chão do cenário. Com a utilização das *constraints* pode-se movimentar apenas o osso do pé, sendo que os demais ossos são movimentados por uma reação em cadeia, além de não ter a necessidade de preocupar-se com os limites do chão na cena. Dentre as diversas *bone constraints* existentes, pode-se citar:

- *Inverse Kinematics*: é uma técnica que facilita o processo de animação, fazendo com que uma cadeia de ossos seja movimentada de acordo com o posicionamento de um determinado osso da ponta da cadeia (Blender Wiki, 2013). Por exemplo, pode-se citar a movimentação da perna do personagem apenas com a movimentação do pé. Neste caso, o *Inverse Kinematics* gera uma reação nos ossos envolvidos pela *constraint*, produzindo a ação de dobrar o joelho do personagem.

- *Copy Location/Rotation/Scale*: são restrições relativas à localização, rotação e escala de um osso comparado a outro (Blender Wiki, 2013). Como exemplo, pode-se criar uma *constraint* que impeça que osso de um braço se aproxime do osso da espinha do personagem, mantendo uma distância mínima, e assim, evitando que o braço do modelo entre no corpo do mesmo.
- *Floor Constraint*: é uma restrição que pode ser aplicada tanto em objetos quanto em ossos com o objetivo de que um objeto não sobreponha o outro. Como exemplo, pode-se criar uma *constraint* que impeça que o pé do personagem ultrapasse do limite do chão (Blender Wiki, 2013).

3.2 Animação no *Blender*

O conceito de *keyframe* é utilizado do *Blender* de forma similar ao método de animação manual. Nesse, o animador posiciona os personagens e objetos da cena em um determinado tempo da animação e logo após, modifica este posicionamento, gravando-o em uma nova posição de tempo. Com os *keyframes* gravados, o *Blender* gera automaticamente as imagens intermediárias, através da interpolação dos pontos que compõe a imagem, sendo que essa interpolação recebe o nome de *F-Curve* (Blender Wiki, 2013). Os quadros intermediários são gerados a partir da interpolação de valores como, por exemplo, ângulos, coordenadas, escalas, cores e até mesmo valores de influência para as *bone constraints* (MULLEN, 2011). Na Figura 3.3 pode-se visualizar as *F-Curves* geradas através da movimentação da câmera nos eixos y e z.

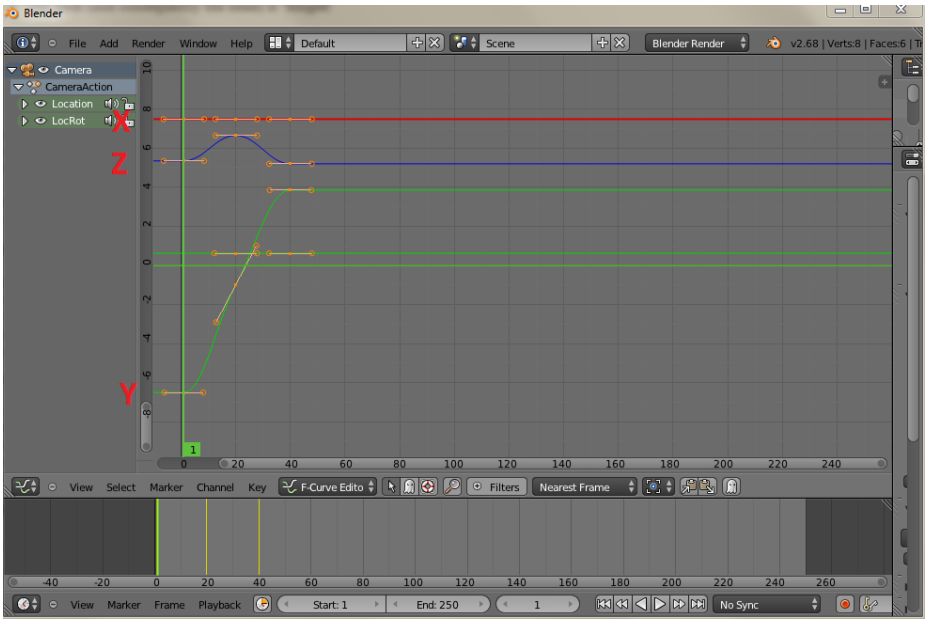


Figura 3.3: *F-Curve* referente à movimentação da câmera nos eixos y e z.

Outro artefato importante para realizar uma animação no *Blender* é a utilização da *timeline*, ou seja, a linha do tempo da animação. É nesta linha em que os *keyframes* serão gravados. Na Figura 3.4 pode-se visualizar a *timeline* (destacada em vermelho) contendo alguns *keyframes* (linhas amarelas) e tempo corrente da animação (linha verde).

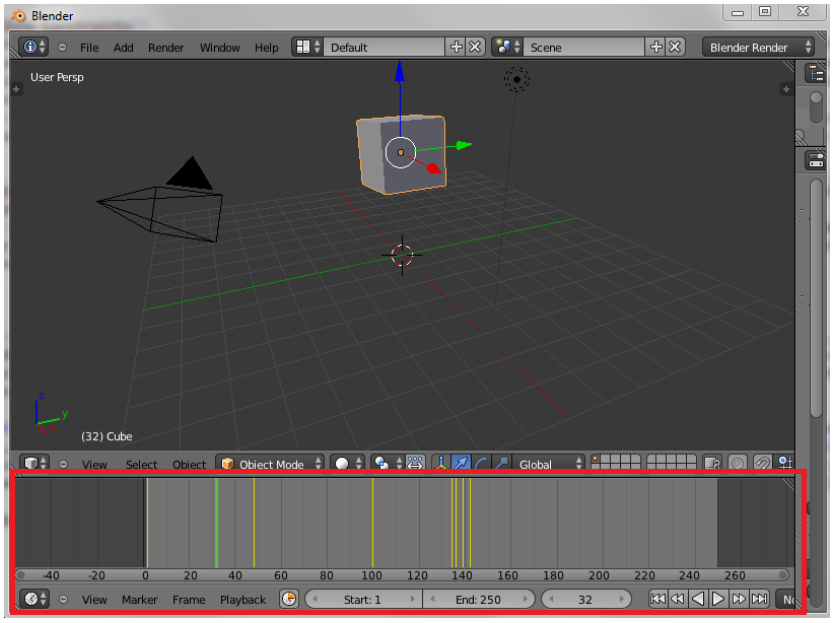


Figura 3.4: Visualização de uma *timeline* no *Blender* durante a animação de um cubo.

Dentre as opções disponíveis para a geração dos *keyframes*, pode-se mencionar a rotação, escala e localização e suas combinações como base para criação de uma animação. Como exemplo, pode-se citar a animação de um cubo. No momento em que um *keyframe* é gravado, é salva a posição inicial dos objetos na cena. Para mover o cubo de lugar, pode-se gravar dois *keyframes* de localização em posições distintas na *timeline*, sendo que no momento da gravação do *keyframe* 1 o cubo encontra-se em uma posição diferente da posição gravada no *keyframe* 2. Desta forma, a partir da interpolação dessas localizações são gerados quadros intermediários, criando uma animação de movimentação do cubo. O mesmo exemplo pode ser utilizado para a rotação e a escala. Além disso, a *timeline* possibilita a configuração da quantidade de quadros por segundo e tempo total da animação.

3.2.1 Interpolação e Extrapolação

O *Blender* gera as *F-Curves* entre os pontos referentes aos *keyframes* gravados, sendo que a forma de geração dessas curvas pode ser alterada de acordo com o tipo de interpolação e extrapolação selecionada. A interpolação refere-se ao método utilizado pelo *Blender* para gerar a curva entre dois *keyframes*, enquanto a extrapolação refere-se ao método utilizado para gerar o formato que curva possui antes e depois

dos *keyframes* (MULLEN, 2011). Existem três tipos de interpolação e dois tipos de extrapolação disponíveis no *Blender*, que são:

- Interpolação de Bézier: é o tipo de interpolação padrão do *Blender*. Esse tipo de interpolação gera curvas arredondas e suaves entre os dois *keyframes*, tornando assim a animação mais suave e com movimentação natural (MULLEN, 2011).
- Interpolação Linear: cria uma reta entre os dois *keyframes*. Este tipo de interpolação é útil para animações que desejam mostrar um movimento mecânico ou robótico (MULLEN, 2011).
- Interpolação Constante: gera uma mudança súbita de estado, fazendo com que a movimentação não seja vista, ou seja, apenas altera a posição de um ponto para outro sem gerar modificações nos quadros intermediários aos *keyframes* (MULLEN, 2011).
- Extrapolação Linear: mantém os parâmetros obtidos pela interpolação, antes e depois dos *keyframes*. Isso faz com que a curva mantenha a mesma forma que foi gerada pela interpolação (Blender Wiki, 2013).
- Extrapolação Constante: é o tipo de extrapolação padrão do *Blender*. Mantém o valor dos pontos-chave, mantendo o objeto animado fixo em uma determinada posição antes e depois da mudança gerada pela interpolação (Blender Wiki, 2013).

3.3 *Addons*

Os *addons* do *Blender* são extensões que agregam novas funcionalidades à ferramenta (Blender Wiki, 2013). Além disso, pode-se implementar extensões que permitam a integração do *Blender* com outras aplicações, exportação de dados ou até mesmo, a adição de novos renderizadores à ferramenta (Blender Wiki, 2013).

O desenvolvimento de *addons* no *Blender* é realizado através da criação de *scripts* na linguagem de programação Python. Para tanto, esses *scripts* necessitam de certos requerimentos e devem seguir um determinado modelo (Blender Wiki, 2013). Por exemplo, o modelo exige que inicialmente seja criado um dicionário de informações do *script*, contendo o nome do *script*, descrição, autor, versão do *script*, versão do *Blender* compatível, dentre outras informações. Na Figura 3.5 tem-se um exemplo de um dicionário de *addons* para o *Blender*.

```
bl_info = {
    "name": "My Script",
    "description": "Single line explaining what this script exactly does.",
    "author": "John Doe, Jane Doe",
    "version": (1, 0),
    "blender": (2, 65, 0),
    "location": "View3D > Add > Mesh",
    "warning": "", # used for warning icon and text in addons panel
    "wiki_url": "http://wiki.blender.org/index.php/Extensions:2.5/Py/"
                "Scripts/My_Script",
    "tracker_url": "http://projects.blender.org/tracker/index.php?"
                   "func=detail&aid=<number>",
    "category": "Add Mesh"}
```

Figura 3.5: Exemplo de informações de um *addon* no *Blender*.
Fonte: Blender Wiki (2013)

4 KINECT

Criado pela Microsoft em parceria com a PrimeSense (GIORIO; FASCINARI, 2013), o Kinect é um sensor capaz de detectar os movimentos de uma pessoa e determinar a distância entre o sensor e essa pessoa ou de outros objetos.

Diferente das câmeras comuns, o Kinect funciona como uma câmera de profundidade. Segundo BORENSTEIN (2012), as câmeras *RGB* (*Red, Green, Blue*) capturam a luz refletida em objetos e transformam esta luz em uma imagem, sendo que essa imagem é exibida na tela do computador de forma que cada *pixel* receba uma determinada cor, ou seja, a imagem capturada é exibida de forma similar ao que o olho humano consegue visualizar. Este tipo de imagem não fornece dados sobre a profundidade dos objetos na cena, pois a imagem apresenta somente *pixels* de diferentes cores, entretanto fornece alguns dados importantes, como por exemplo a aparência dos objetos.

O Kinect (Figura 4.1) é formado por uma câmera *RGB* que possibilita determinar a forma dos objetos em uma determinada cena. No entanto, as informações obtidas por essa câmera não são suficientes para determinar a profundidade dos objetos. Para isto, o Kinect possui um projetor de luz infravermelha e uma câmera infravermelha, também chamada de câmera de profundidade (GIORIO; FASCINARI, 2013).

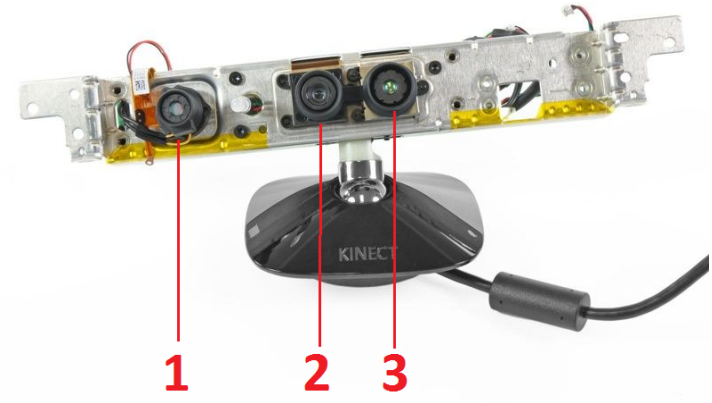


Figura 4.1: Kinect sem o revestimento de plástico, observando-se o projetor infravermelho (1), a câmera RGB (2) e a câmera infravermelha (3)
Fonte: BORENSTEIN (2012)

Através do projetor de luz infravermelha, o Kinect emite uma grade de pontos infravermelhos sobre os objetos. Estes pontos são invisíveis ao olho nu, entretanto a câmera infravermelha consegue capturar estes pontos de luz. Desta forma, o Kinect consegue determinar a distância entre o sensor e o objeto através da distorção dos pontos projetados. Isso é possível graças ao fato do Kinect armazenar, a partir de um processo de calibragem realizado em fábrica, a posição original da projeção dos pontos infravermelhos. Ou seja, quando calibrado na fábrica, o Kinect emite essa grade de pontos sobre uma parede que encontra-se posicionada a uma distância x do sensor (BORENSTEIN, 2012), sendo que essa grade de pontos é armazenada no Kinect em correlação a essa distância x, ou seja, quando esses pontos atingem um objeto que está a uma distância diferente da distância armazenada, estes pontos apresentam um padrão distorcido se comparado ao original. Sendo assim, é possível determinar a distância do objeto através da distorção dos pontos em relação à grade de pontos original (GIORIO; FASCINARI, 2013).

O Kinect possui algumas limitações com relação à sua câmera de profundidade, sendo que dentre estas pode-se citar a distância mínima entre o objeto e o sensor. A câmera de profundidade não consegue capturar objetos que estejam a uma distância inferior a 40 centímetros. Sendo assim, objetos que estão muito próximos ao sensor são tratados da mesma forma que objetos que estão a uma distância muito grande (BORENSTEIN, 2012). O sensor também não funciona corretamente em ambientes afetados por luz solar intensa, superfícies refletivas ou sob interferência de luzes que estão em uma faixa de onda similar à luz infravermelha (GIORIO; FASCINARI, 2013). Além disso, a luz infravermelha também pode reproduzir sombras na imagem, não sendo possível determinar a profundidade da área atingida por esta sombra. Na Figura 4.2 pode-se observar a sombra (área circulada na figura) gerada pela luz infravermelha.

Figura 4.2: Sombra gerada pela luz infravermelha.

4.1 Imagem de Profundidade

A manipulação da profundidade da imagem pode ser realizada com o funcionamento em conjunto da câmera de profundidade, da câmera *RGB* e do projetor infravermelho. Com a união destes três equipamentos é possível, por exemplo, modificar as cores de uma imagem de acordo com sua posição na cena, ignorar objetos que estejam a uma determinada distância ou até mesmo determinar a proximidade entre os objetos e o sensor (BORENSTEIN, 2012). Na Figura 4.3a tem-se a imagem manipulada em tons de amarelo, enquanto na Figura 4.3b tem-se a imagem original correspondente à mesma cena. Na imagem manipulada, todo objeto que está próximo ao sensor recebe tons mais claros de amarelo, enquanto tudo que está no fundo da cena recebe tons mais escuros de amarelo. Além disso, objetos que estejam fora da faixa de alcance do sensor são ignorados na cena e recebem a cor preta.

Figura 4.3: Imagem de profundidade em escalas de amarelo.

Na Figura 4.4 tem-se uma ilustração do processo de geração de uma imagem de profundidade no Kinect. O *chip* da PrimeSense envia um sinal de ativação para o emissor de luz infravermelha e para a câmera de profundidade. Após a ativação,

o emissor de luz infravermelha projeta uma grade de pontos de luz nos objetos da cena e a câmera de profundidade lê os dados coletados. Em seguida, estes dados são enviados para o *chip* e processados, disponibilizando a imagem de profundidade solicitada.

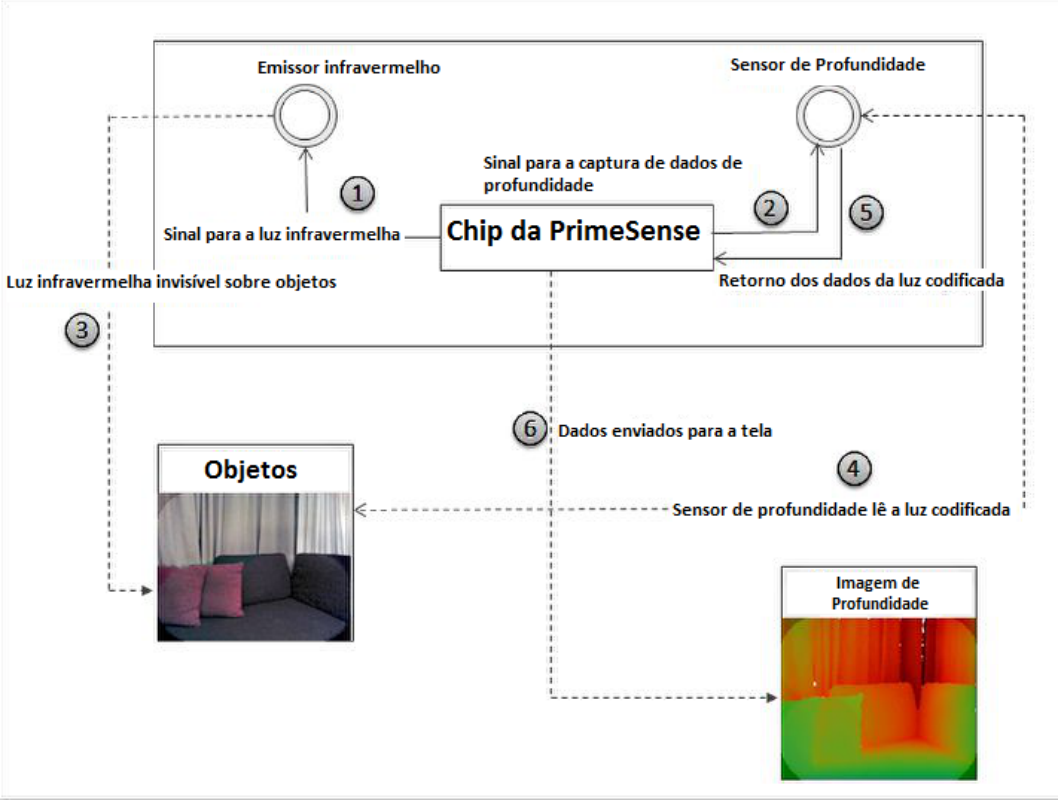


Figura 4.4: Geração da imagem de profundidade
Fonte: JANA (2012)

A imagem de profundidade gerada pelo processamento da luz infravermelha contém dados relativos à distância dos objetos na cena, sendo que a imagem pode ser exibida em diversas cores na tela. Entretanto, é comum serem utilizadas as escalas de cinza para determinar a posição dos objetos (BORENSTEIN, 2012), ou seja, é gerada uma imagem monocromática, onde cada pixel na tela possui um valor entre 0 e 255, sendo 0 representando a cor preta e 255 representando a cor branca. Qualquer valor intermediário entre valores será uma tonalidade da cor cinza, sendo que os objetos que estão no fundo da cena recebem uma tonalidade mais escura, enquanto os objetos próximos ao sensor recebem uma cor mais clara. Desta forma, a partir do valor de cada pixel é possível determinar a posição de um objeto no cenário.

4.2 Dados de Esqueleto

O Kinect permite identificar a presença de pessoas e informar a posição dessas. Para tanto, esse utiliza de algumas bibliotecas adicionais. Através do uso dessas bibliotecas, é possível detectar a posição de diversas partes do corpo, como, por exemplo, a cabeça, os braços e as mãos sem a necessidade do desenvolvimento de algoritmos específicos para o reconhecimento do corpo (BORENSTEIN, 2012). A partir do uso dessas bibliotecas torna-se possível a criação de um conjunto de nós interligados e devidamente posicionados, chamado de esqueleto. Um esqueleto detalhado contém 20 pontos no modo normal, que é o modo em que a pessoa está em pé, e 10 pontos no modo sentado, sendo que este modo é determinado através da visibilidade do usuário em relação ao sensor (GIORIO; FASCINARI, 2013). Na Figura 4.5 tem-se um exemplo de esqueleto em ambos os modos citados.

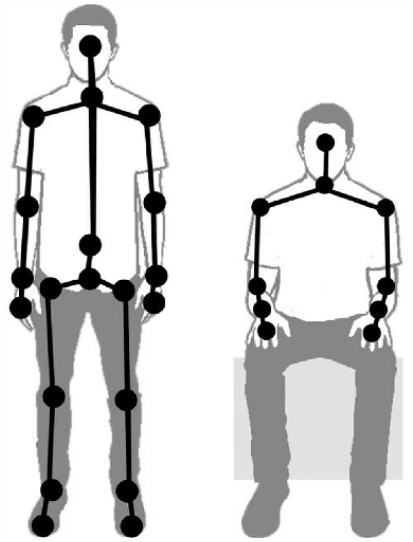


Figura 4.5: Esqueleto no modo normal e sentado, respectivamente.
Fonte: GIORIO; FASCINARI (2013)

As arestas determinadas pela junção desses pontos representam as partes do corpo, enquanto os pontos funcionam como "juntas" que delimitam onde uma parte do corpo termina e outra inicia. O Kinect consegue reconhecer até seis pessoas na cena, entretanto somente duas podem ser mapeadas detalhadamente (GIORIO; FASCINARI, 2013). A detecção e geração do esqueleto depende de qual biblioteca será utilizada para o desenvolvimento da aplicação, sendo que algumas bibliotecas necessitam de calibragem para a detecção do esqueleto. Como exemplo de biblioteca que necessita de calibragem pode-se citar a OpenNI (*Open Natural Interaction*) (FALAHATI, 2013).

5 DESENVOLVIMENTO PARA KINECT

Atualmente, os sensores detectores de movimento são utilizados em diversas aplicações, que envolvem desde simples sistemas de alarmes domésticos até radares militares (FALAHATI, 2013). O surgimento do Kinect fez com que novas funcionalidades surgissem com relação à utilização desse tipo de sensor, sendo possível a aplicação de *NUI* (*Natural User Interface*) de uma forma mais eficaz, como, por exemplo, a possibilidade de controlar um computador através de gestos e a criação de aplicações que utilizem o movimento do usuário (CATUHE, 2012). A utilização de *NUI* implica não somente na "mímica" de movimentos do usuário, mas também no controle natural que a pessoa possui sobre uma determinada aplicação (WIGDOR; WIXON, 2011). Porém, para que fosse possível a utilização de movimentos naturais para controlar aplicações ou realizar tarefas específicas através do uso do Kinect, alguns *SDK*s (*Software Development Kit*s) foram desenvolvidos.

5.1 OpenNI

O OpenNI é um *framework* e *SDK* de código aberto utilizado para o desenvolvimento de aplicações e bibliotecas para sensores como o Kinect e similares (OpenNI, 2013). Este *framework* é popular entre os desenvolvedores pois apresenta suporte para os sensores que utilizam o *chip* da empresa PrimeSense, sendo assim, é possível desenvolver aplicações que utilizem sensores Kinect, Asus Xtion e sensores fabricados pela PrimeSense (FALAHATI, 2013).

A OpenNI é a organização responsável pelo *framework* de mesmo nome, sendo que o fundador do projeto inicial era a própria PrimeSense (FALAHATI, 2013). O projeto foi muito difundido uma vez que esse foi o primeiro *framework* com suporte não-oficial ao Kinect, já que na época não existiam outros *frameworks* para o desenvolvimento de aplicações com este sensor (FALAHATI, 2013).

O *SDK* do OpenNI é composto por uma *API* (*Application Programming Interface*) que permite o desenvolvimento de aplicações utilizando sensores 3D (OpenNI, 2013), uma camada de abstração de *hardware* que permite que a aplicação seja uti-

lizada em diferentes arquiteturas (ECKER; MüLLER; DöMER, 2009), e um núcleo que realiza a comunicação com os sensores através do *driver* do dispositivo (OpenNI, 2013). Na Figura 5.1 tem-se a arquitetura do OpenNI.

Figura 5.1: Arquitetura do *SDK* OpenNI
Fonte: OpenNI (2013)

A linguagem de programação padrão para o desenvolvimento de aplicações que utilizam o OpenNI é a linguagem C, entretanto é possível utilizar outras linguagens, tais como C++, C#, Python e Java (OpenNI, 2013). Destaca-se que o OpenNI não gera informações sobre detecção de pessoas ou mapeamento de corpo de forma automática, sendo necessário o desenvolvimento de algoritmos para a detecção do esqueleto ou utilizar-se de outras bibliotecas e *middlewares* para a obtenção dessas funcionalidades.

O OpenNI possui algumas limitações quando utilizado em conjunto com o sensor Kinect. Uma das limitações que pode-se citar é que o *framework* não possui acesso ao motor de rotação do sensor (OpenNI, 2013). O Kinect possui um motor em sua base capaz de inclinar o sensor para cima, para baixo e para os lados (BORENSTEIN, 2012), sendo assim possível ajustar o melhor ângulo da cena sem a necessidade de mudança de localização do sensor no ambiente. Como o *framework* não provê acesso ao motor, esta funcionalidade não está disponível com o uso do OpenNI. Um outro problema que pode-se citar é com relação à detecção de uma pessoa. A detecção,

além de não ocorrer de forma automática, necessita que o usuário fique em uma posição de calibragem conhecida como "pose Psi" (BORENSTEIN, 2012). Esta pose consiste em fazer com que o usuário fique com os braços na altura do ombro e mãos na altura da cabeça, conforme pode-se observar na Figura 5.2.

Figura 5.2: Pose "Psi" para calibragem e detecção do usuário.
Fonte: BORENSTEIN (2012)

Algumas vantagens que pode-se citar sobre o uso deste *framework*, é que o mesmo consegue detectar múltiplas pessoas em uma única cena, sendo que cada pessoa deve posicionar-se na pose de calibragem para que seja mapeada corretamente (BORENSTEIN, 2012). Assim, cada usuário ganha um número identificador único para que haja a distinção dos usuários no ambiente, ou seja, pode-se realizar a manipulação de cada pessoa individualmente (BORENSTEIN, 2012). Outra vantagem do uso do OpenNI é o fato do *framework* ser multiplataforma, suportando os sistemas operacionais Windows, Linux e OS X (OpenNI, 2013). Entretanto, não existem *drivers* oficiais do Kinect para Linux e OS X, sendo necessário o uso de outros *drivers* e aplicações, como, por exemplo, o *libfreenect* (OpenKinect, 2013).

A versão atual do OpenNI provê suporte oficial ao sensor Kinect no sistema operacional Windows, desde que esse seja utilizado juntamente com o *Kinect for Windows SDK* da própria Microsoft (FALAHATI, 2013). Isso se dá pelo fato de que durante a instalação do *SDK* da Microsoft, os *drivers* oficiais do Kinect também são instalados, garantindo assim a compatibilidade com o *framework* OpenNI.

5.1.1 NiTE

O NiTE (*Natural Interface Technology for End-User*) é um *middleware* livre (licença Apache 2.0) desenvolvido pela PrimeSense baseado no *framework* OpenNI (FALAHATI, 2013). Este *middleware* disponibiliza informações sobre a cena e também possibilita a geração do esqueleto através da detecção do usuário (PrimeSense, 2013).

O NiTE recebe os dados de profundidade, cor, sinal infravermelho e áudio, fornecidos pelo OpenNI, e através da utilização de algoritmos específicos, efetua o mapeamento da pessoa (PrimeSense, 2013). Na Figura 5.3 pode-se verificar durante o mapeamento do usuário, a geração de alguns pontos chamados "juntas", que estão posicionados em alguns locais específicos do corpo. A união destas juntas gera uma espécie de esqueleto do usuário, sendo que cada junta pode ser acessada de forma individual, e destas, pode-se extrair algumas informações, como, por exemplo, a sua posição na cena (BORENSTEIN, 2012).

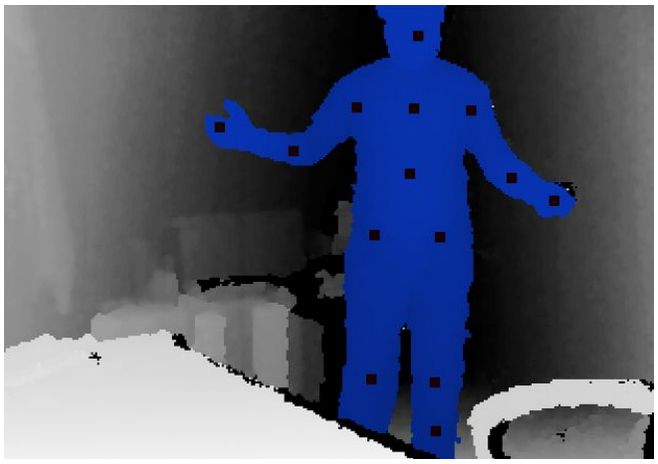


Figura 5.3: Pontos gerados através da detecção do usuário realizada pelo NiTE.
Fonte: FALAHATI (2013)

Um esqueleto gerado pelo NiTE possui 15 juntas que se encontram posicionadas nas mãos, pés, cabeça, pescoço, ombros, tórax, cotovelos, joelhos e quadris (PrimeSense, 2013). Por padrão, não há juntas nos punhos e nos tornozelos, sendo assim, não é possível detectar a movimentação dos pés e das mãos de forma individual. Além disso, este *middleware* só consegue mapear as pessoas no modo em pé, devido a necessidade de calibragem através da "pose Psi". Por fim, o Kinect consegue detectar até 20 pontos em uma pessoa em pé, não sendo possível detectar os pontos restantes com o uso do NiTE (PrimeSense, 2013).

O uso do NiTE em conjunto com o OpenNI é de extrema importância quando necessita-se do mapeamento do corpo do usuário, já que os algoritmos utilizados para realizar este mapeamento possuem alta complexidade (BORENSTEIN, 2012). Além disso, este *middleware* é composto por algoritmos que interpretam gestos, movimentos corporais e até mesmo a voz do usuário, fornecendo ao desenvolvedor diversas ferramentas para auxiliar na implementação de uma solução que utilize *NUI* (PrimeSense, 2013). Além disso, assim como o OpenNI, o NiTE também é multiplataforma e suporta as mesmas linguagens de programação do *SDK* (PrimeSense, 2013).

5.1.2 Python Bindings

Em Julho de 2013 a PrimeSense disponibilizou um pacote chamado *Python Bindings for OpenNI*. Este pacote torna possível a utilização da linguagem Python para o desenvolvimento de aplicações que utilizam OpenNI e NiTE (PrimeSense, 2013).

O pacote possui apenas as ligações com a *API* original do OpenNI na linguagem C, não possuindo nenhuma funcionalidade nova ou diferente, preservando inclusive os nomes das chamadas de função originais (PrimeSense, 2013).

5.2 Microsoft Kinect *SDK*

Logo após o lançamento do Kinect no mercado, o OpenNI era o único *SDK* disponível para o desenvolvimento de aplicações utilizando o sensor. Seis meses após o lançamento do OpenNI, a Microsoft apresentou a primeira versão do Kinect *SDK* beta, composto de diversas bibliotecas e *API*s que permitiam o desenvolvimento de aplicações que utilizavam o sensor *Kinect for Xbox 360* para o sistema operacional Windows (GIORIO; FASCINARI, 2013). Recentemente a Microsoft lançou o sensor *Kinect for Windows*, que é possui apenas uma saída USB (*Universal Serial Bus*), ou seja, não é compatível com o *console Xbox 360*, e possui alguns aprimoramentos quando comparado ao Kinect do Xbox, como, por exemplo, uma maior resolução de câmera e uma limitação menor com relação à distância mínima que os objetos devem estar do sensor (Microsoft, 2013a).

O *SDK* atual suporta oficialmente as linguagens C++, C# e Visual Basic para o desenvolvimento das aplicações, entretanto o *SDK* é voltado para o desenvolvimento utilizando-se o *Kinect for Windows* (Microsoft, 2013b), ou seja, como esse sensor é uma versão aprimorada do *Kinect for Xbox 360*, nem todas as funcionalidades do *SDK* serão compatíveis com o *Kinect for Xbox 360* (CATUHE, 2012). Além disso, caso um usuário final (que não possua o *SDK* instalado no computador) tente executar a aplicação utilizando o *Kinect for Xbox 360*, um erro irá ocorrer na inicialização do programa, informando ao usuário que o dispositivo não é suportado (CATUHE, 2012).

O sistema de identificação de usuários funciona de forma similar ao OpenNI, sendo que cada usuário recebe um número identificador para sua distinção na cena. Entretanto, só é possível detectar até seis pessoas na cena e produzir um esqueleto detalhado para apenas duas delas (GIORIO; FASCINARI, 2013).

Ao contrário do OpenNI, o *SDK* da Microsoft não necessita de um *middleware* para geração do esqueleto durante o mapeamento do usuário, ou seja, o próprio *SDK* engloba as funcionalidades do OpenNI e do NiTE (Microsoft, 2013b). O esqueleto gerado mapeia os todos 20 pontos suportados pelo Kinect no modo em pé (Figura 5.4) e 10 pontos no modo sentado. Além disso, o *SDK* não necessita de um pose

para a calibragem, detectando o usuário de forma mais rápida, além de prover acesso ao motor de inclinação do sensor (Microsoft, 2013b).

Figura 5.4: Esqueleto gerado através do Microsoft Kinect *SDK* no modo em pé.

O *SDK* não possui código aberto e possui algumas restrições para sua utilização. Uma das restrições que pode-se citar é o uso limitado em conjunto com o sensor *Kinect for Xbox 360*. Após a fase beta, o *SDK* oficial foi lançado juntamente com o sensor *Kinect for Windows*, sendo que o desenvolvimento de aplicações ficou restrito somente à utilização desse mesmo sensor. Dessa forma, só é possível desenvolver aplicações que utilizam o sensor *Kinect for Xbox 360* com o *SDK* beta, ou então, utilizar este sensor somente para o desenvolvimento e para os testes de aplicações para o *Kinect for Windows* (Microsoft, 2013a). Conforme os termos de utilização do *SDK*, no caso da utilização do sensor *Kinect for Xbox 360* para o desenvolvimento de aplicações, o desenvolvedor ou distribuidor da aplicação não poderá comercializar ou incentivar o uso da mesma com este sensor (Microsoft, 2013a).

5.2.1 PyKinect

O PyKinect é uma ferramenta livre (licença Apache 2.0) para o *software* Microsoft Visual Studio 2013 que habilita o uso da linguagem Python para o desenvolvimento de aplicações que utilizam o sensor Kinect (Python Tools, 2013). A ferramenta funciona de forma similar ao *Python Bindings for OpenNI*, entretanto, o PyKinect utiliza o próprio *SDK* da Microsoft para o processamento das imagens de profundidade e geração do esqueleto durante o mapeamento do usuário. Atualmente, a ferramenta está em uma versão estável, entretanto é compatível somente com a instalação do Python 32-*bits* e necessita que o corpo inteiro esteja visível ao sensor para a geração do esqueleto do usuário (Python Tools, 2013).

5.3 Testes dos *SDK*s

Os *SDK*s citados nas seções 5.1 e 5.2 foram testados em conjunto com as aplicações de exemplo, que são disponibilizadas pela OpenNI/PrimeSense e pela Microsoft. Os testes realizados baseiam-se apenas com relação à detecção do usuário e à geração do esqueleto e suas particularidades. O ambiente em que os testes foram realizados foi um ambiente fechado, iluminado por lâmpadas e com influência indireta da luz do sol, conforme pode-se observar na Figura 5.5. A pessoa a ser mapeada estava a uma distância de aproximadamente três metros do sensor e realizou movimentos com braços, pernas e cabeça neste local.

Figura 5.5: Ambiente em que os testes dos *SDK*s foram feitos.

Em relação aos testes realizados verificou-se que o esqueleto gerado pelo Microsoft Kinect *SDK* é mais estável do que o gerado pelo NiTE, já que os ruídos da imagem de profundidade fazem com que o esqueleto do NiTE fique mais trêmulo, principalmente quando o usuário está próximo ao sensor. Entretanto, o esqueleto do NiTE aparenta ter movimentação mais natural. Isso deve-se ao fato do Microsoft Kinect *SDK* utilizar um algoritmo de suavização (para evitar que o esqueleto fique trêmulo), deixando o movimento desse com um leve atraso e não acompanhando o usuário de forma tão dinâmica quanto o NiTE.

Tanto o NiTE quanto o Microsoft Kinect *SDK* apresentaram o mesmo comportamento quando uma área do corpo não pudesse mais ser detectada, como, por exemplo, um braço nas costas. Nesse caso, o esqueleto permaneceu na última posição antes do braço atingir a na área inalcançável, e as juntas perdidas foram destacadas utilizando uma cor diferente. Na Figura 5.6a pode-se visualizar na área circulada o tratamento da área inalcançável pelo NiTE, com a geração de "juntas fantasmas". Já na na Figura 5.6b tem-se o mesmo tratamento feito pelo Microsoft Kinect *SDK*.

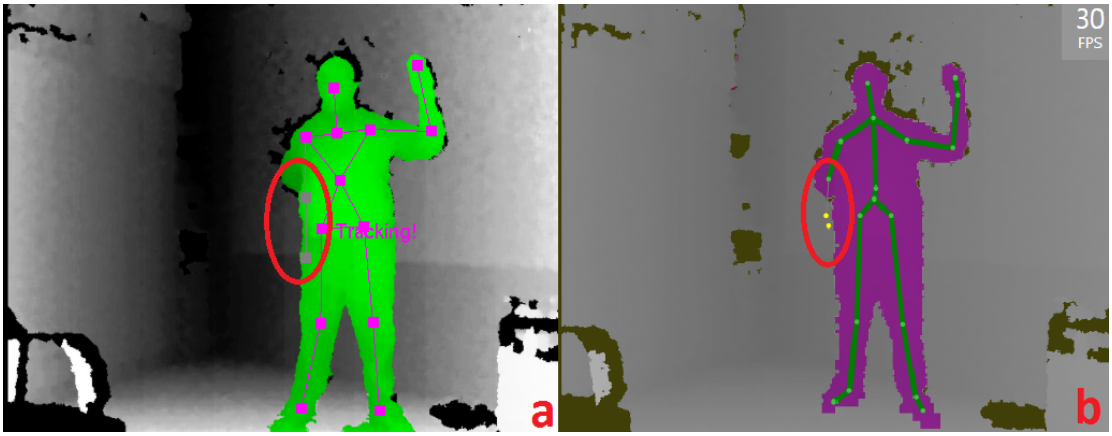


Figura 5.6: Tratamento de área inalcançável no NiTE (a) e no Microsoft Kinect *SDK* (b).

A utilização de ambos *SDK*s apresentou problemas com relação à proximidade do usuário com o sensor. Embora o usuário seja detectado, o esqueleto só é gerado caso a pessoa esteja a uma distância de aproximadamente um metro do sensor. De fato, o Microsoft Kinect *SDK* não gerou o esqueleto a uma distância inferior a esta, mesmo no modo sentado, como pode-se observar na Figura 5.7. Já o NiTE gerou o esqueleto, mas com uma quantidade muito grande de ruídos na imagem, deixando o esqueleto muito trêmulo e inviabilizando o uso.

Figura 5.7: O Microsoft Kinect *SDK* detecta o usuário mas não consegue gerar o esqueleto à uma curta distância.

O esqueleto do NiTE possui um número menor de juntas no esqueleto, não possuindo as juntas dos tornozelos e dos punhos, incapacitando que o movimento das mão e pés sejam detectados. Entretanto, mesmo com estas opções disponibilizadas pelo Microsoft Kinect *SDK*, o resultado obtido não foi satisfatório, uma vez que o esqueleto não foi capaz de acompanhar as mãos corretamente, como pode-se visualizar nas áreas circuladas da Figura 5.8.

Figura 5.8: O Microsoft Kinect *SDK* não consegue capturar corretamente a posição das mãos em alguns casos.

Portanto, pode-se concluir que ambos os *SDK*s possuem limitações com relação ao processamento da imagem de profundidade e da geração do esqueleto do usuário. Isso também deve-se ao fato das limitações de *hardware* do próprio sensor com relação ao ruído na imagem e influência da luz do sol, mesmo que de forma indireta. Todavia, o esqueleto gerado pelo Microsoft Kinect *SDK* possui uma estabilidade maior na cena quando comparado ao esqueleto gerado pelo NiTE. Para os demais casos, o desempenho das ferramentas foi equivalente.

5.3.1 Comparativo entre os *SDK*s

Após a análise do funcionamento, restrições e licenciamento dos *SDK*s disponíveis para o desenvolvimento de aplicações que utilizam o sensor Kinect, foi realizado um comparativo entre ambos, conforme é exibido na Tabela 5.1. A partir dessa análise, optou-se pela utilização do OpenNI e do NiTE para o desenvolvimento da solução. Verificou-se que ambos os *SDK*s são equivalentes com relação ao processamento das informações recebidas a partir do Kinect, entretanto, o OpenNI oferece compatibilidade com outros sensores e outros sistemas operacionais. Além disso, as vantagens oferecidas pelo Microsoft Kinect *SDK* não foram satisfatórias, uma vez que os pontos extras detectados pelo esqueleto não foram mapeados de forma correta e a calibragem não é um fator crucial para a aplicação a ser desenvolvida neste trabalho.

Tabela 5.1: Tabela de comparação de recursos dos *SDK*s.

	OpenNI + NiTE	**Microsoft Kinect *SDK***
Linguagens Suportadas (Padrão)	C	C++, C#, VB
Expansão para Python	Sim	Sim
Plataformas	Windows, Linux, Mac	Windows
Controle do Motor do Sensor	Não	Sim
Pontos por Esqueleto (Normal)	15	20
Pontos por Esqueleto (Sentado)	Não suporta	10
Limite de Pessoas Detectadas	Não se aplica	6
Limite de Esqueletos na Cena	Não se aplica	2
Requer Calibragem	Sim	Não
Compatível com Outros Sensores	Sim	Não
Licença	Livre	Restrita

6 APLICAÇÕES DE INTEGRAÇÃO

A integração do Kinect com o *Blender* possibilita uma melhora no método utilizado para a geração de uma animação 3D, tornando o processo mais ágil e menos custoso (MULLEN, 2011). Esta integração não é algo inovador, sendo que já existem algumas aplicações que realizam essa integração, como, por exemplo, o Ni Mate (Delicode, 2013) e o Bloop (*Blender Loop Station*) (TZi, 2011).

6.1 Delicode Ni Mate

O Ni Mate (*Natural Interaction Mate*) é um *software* produzido pela Delicode, que processa as informações coletadas de sensores como o Kinect através do uso do OpenNI (Delicode, 2013). O *software* recebe as informações sobre a detecção do usuário e possibilita a utilização dessas informações em outros programas, tais como o *Blender* e o Maya (Delicode, 2013). O *software* é multiplataforma e não é gratuito, entretanto é possível solicitar uma licença de avaliação que é vinculada ao número de série do sensor utilizado (Delicode, 2013).

O Ni Mate possui diversas abas de configuração da ferramenta, sendo possível configurar tanto opções de processamento da imagem e de profundidade, quanto da geração do esqueleto do usuário (Delicode, 2013). Para utilização do Ni Mate em conjunto com o *Blender*, duas destas abas de configurações são importantes, que são as abas *Full Skeleton* e *OSC (Open Sound Control) Controller*.

A aba *Full Skeleton* permite que um nome seja atribuído a cada vértice do esqueleto, conforme pode-se observar na Figura 6.1. Esse esqueleto, com seus vértices nomeados e suas coordenadas, é enviado para o *Blender* através do protocolo *OSC* (OSC, 2011).

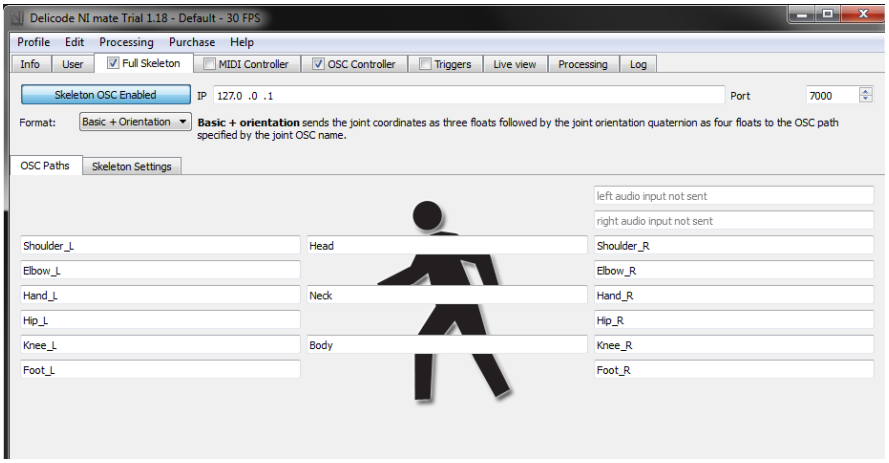


Figura 6.1: Aba de configuração do esqueleto do Ni Mate.

A aba *OSC Controller* permite que sejam configuradas as opções de envio de mensagens para o *Blender*. Os dados de movimentação do usuário são convertidos em mensagens, e essas são transmitidas através do protocolo *OSC* para o *Blender* (Delicode, 2013). Além disso, também é possível configurar os limites da captura do sinal e o número máximo de pessoas na cena (Figura 6.2). Este controlador possui um algoritmo de detecção de usuário diferente do *Full Skeleton*. Neste, o esqueleto do usuário não é detectado, mas sim, as partes do seu corpo através do processamento da imagem de profundidade (Delicode, 2013).

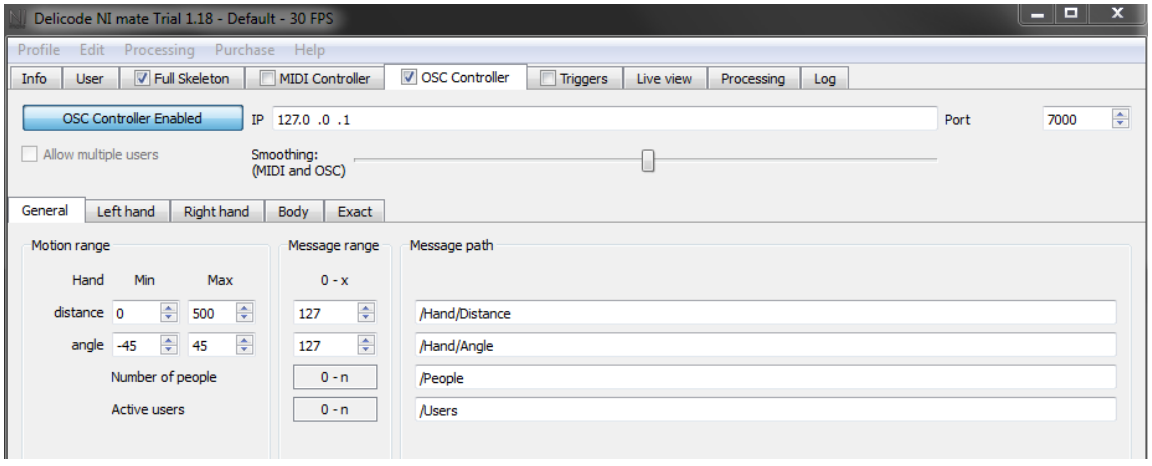


Figura 6.2: Aba de configuração do controlador *OSC* do Ni Mate.

Após os dados serem enviados via *OSC*, o *Blender* efetua o processamento das mensagens recebidas. Entretanto, o *software* não possui tal funcionalidade implementada, sendo necessária a instalação de um *addon* oficial da Delicode (Delicode, 2013). Esse *addon*, que é gratuito e apresenta uma licença GNU *GPL*, possibilita que o usuário defina a porta em que as mensagens serão recebidas e inicie ou pare o processo de recepção das mensagens (Figura 6.3).

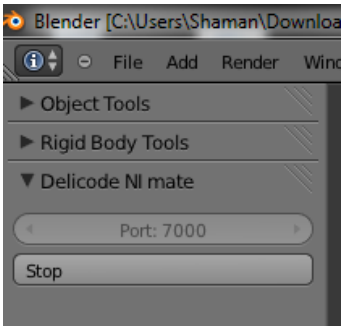


Figura 6.3: *Addon* do *Blender* para a recepção de mensagens enviadas do Ni Mate.

As mensagens recebidas pelo *Blender* são processadas e a ligação dos vértices recebidos com o *armature* é feita considerando o nome do vértice e o nome do *bone*, ou seja, vértices e *bones* que possuem o mesmo nome são associados na cena de forma automática. Também é possível associar um grupo de *bones* à uma parte do corpo específica. Nesse caso, o grupo de *bones* e a parte do corpo também devem possuir o mesmo nome. Na Figura 6.4 pode-se observar um *armature* com *bones* nomeados no *Blender*, e um esqueleto gerado pelo Ni Mate.

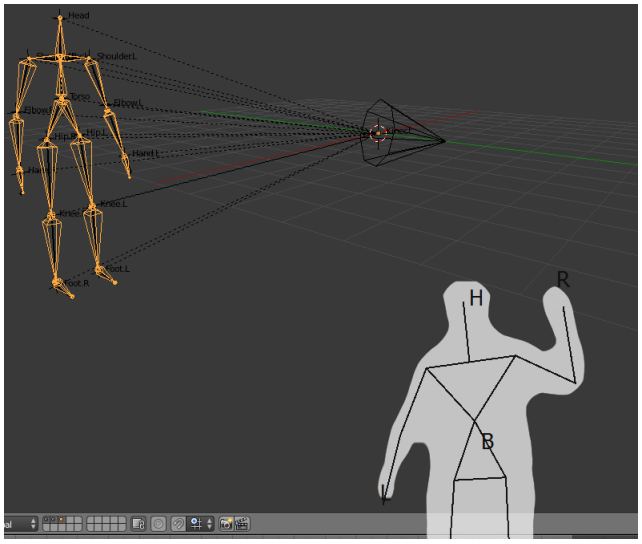


Figura 6.4: *Armature* com *bones* nomeados, assim como o esqueleto gerado pelo Ni Mate.

O Ni Mate possui algumas limitações com relação ao uso do *Kinect for Xbox 360*. Existem duas versões do Ni Mate para Windows, sendo que uma delas suporta o novo sensor *Kinect for Windows* e a outra suporta o sensor *Kinect for Xbox 360*. A versão que suporta o sensor *Kinect for Windows* funciona com a nova versão do OpenNI e os *drivers* oficiais da Microsoft. Entretanto, se utilizado o sensor do Xbox com essa versão, o aplicativo não inicia, informando incompatibilidade com o sensor. Já a versão que suporta o sensor *Kinect for Xbox 360* funciona somente com uma versão antiga do OpenNI e requer a instalação de *drivers* não-oficiais do

sensor. Devido a incompatibilidade entre a versão que suporta o sensor *Kinect for Xbox 360* e o OpenNI 2, não foi possível realizar testes com essa ferramenta.

6.2 Bloop

O Bloop, também conhecido como *Blender Loop Station* é uma ferramenta similar ao Ni Mate, que possibilita que as informações coletadas pelo Kinect sejam processados pelo *Blender* para a geração de uma animação de um personagem 3D (TZi, 2011). O projeto foi criado por dois estudantes da Universidade de Bremen, na Alemanha, e utiliza o Microsoft Kinect *SDK* para o processamento do sinal e do esqueleto do usuário (TZi, 2011).

O Bloop funciona por comando de voz, ou seja, o usuário utiliza a fala para enviar um comando para o Kinect, que executa a ação e envia os dados para o *Blender* (TZi, 2011). O usuário pode, por exemplo, falar o comando "*Start*" para que o Kinect envie um sinal para o *Blender* começar a gravação da animação. Além disso, a ferramenta possibilita ainda o mapeamento do personagem via gestos e permite que mais que um usuário controle o mesmo personagem (TZi, 2011).

Assim como no Ni Mate, a comunicação entre o Kinect e o Bloop é feita através do protocolo *OSC* (TZi, 2011). Entretanto, o Bloop recebe os dados via *OSC* diretamente, ou seja, com exceção do programa que transmite o sinal do Kinect via *OSC*, toda a aplicação foi escrita como um *addon* do *Blender* (TZi, 2011).

O princípio do Bloop é utilizar a movimentação dos *control bones* do *Blender* para realizar uma animação, não abrindo mão da utilização de *constraints*. Na Figura 6.5 pode-se visualizar a modelagem de um canguru com os *control bones* que o Bloop utiliza para a animação.

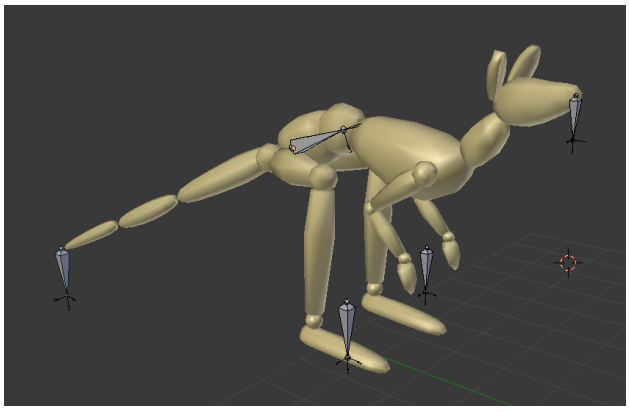


Figura 6.5: Modelagem de um canguru no *Blender*, com os *control bones* utilizados para a animação com o Bloop.

O Bloop é um projeto que aparentemente foi abandonado, pois desde 2011 não recebe atualizações (TZi, 2011). Desta forma, essa ferramenta foi desenvolvida para uma versão antiga do *Blender*, além de utilizar a versão beta do Microsoft Kinect *SDK*, já que a versão 1.0 do *SDK* foi lançada somente em 2012 (Microsoft, 2013b). Além disso, como a aplicação utiliza as informações processadas pelo Microsoft Kinect *SDK*, a ferramenta só funciona no sistema operacional Windows. Devido a ferramenta estar desatualizada, não foi possível realizar testes com a mesma.

6.3 Comparativo entre as Ferramentas

Pode-se concluir que a utilização do Bloop é inviável devido à falta de atualizações da ferramenta. De fato, através desse só é possível realizar a integração somente com versões mais antigas do *Blender* e do próprio Microsoft Kinect *SDK*. Já a utilização no Ni Mate é viável, entretanto o *software* não é gratuito e utiliza uma versão antiga do *framework* OpenNI para integração com o *Kinect for Xbox 360*. Desta forma, através desse não é possível utilizar os novos recursos oferecidos pelo OpenNI. Na Tabela 6.1 tem-se um comparativo entre as soluções de integração já existentes.

Tabela 6.1: Tabela de comparação das ferramentas de integração entre Kinect e *Blender*.

	Ni Mate	**Bloop**
Gratuito	Não	Sim
Código Aberto	Não	Sim
Plataformas	Windows, Linux, Mac	Windows
Comunicação	*OSC*	*OSC*
Framework	OpenNI	Microsoft Kinect *SDK*
Atualizado	Sim	Não
Kinect for Xbox 360	Suportado[1]	Suportado
Kinect for Windows	Suportado[2]	Não Suportado
Outros Sensores	Suportado	Não Suportado

Assim sendo, as opções disponíveis atualmente não satisfazem a necessidade de todos os usuários. Isso deve-se principalmente à inexistência de uma ferramenta gratuita, multiplataforma e atualizada que realize a integração entre Kinect e *Blender*. Desta forma, optou-se pelo desenvolvimento de uma aplicação que atenda as necessidades desses usuários.

[1]Somente com a versão 1.x do OpenNI e drivers não-oficiais.
[2]Somente com a versão 2.x do OpenNI e somente no Windows.

7 IMPLEMENTAÇÃO E TESTES

Devido as limitações das aplicações já existentes para a integração entre Kinect e Blender, optou-se pelo desenvolvimento de uma ferramenta que permitisse essa integração. Desta forma, nesse Capítulo é feita uma descrição da aplicação desenvolvida. Mais especificamente, na Seção 7.1 tem-se a arquitetura da aplicação desenvolvida. Na Seção 7.2 descreve-se como a comunicação interna da aplicação é realizada. Na Seção 7.3 tem-se as características do *Addon Kinected Blender*, além do seu funcionamento com os *armatures* do Blender. Na Seção 7.4 tem-se as características do *Kinected Blender Receiver*, que é responsável pela comunicação com o sensor Kinect. Por fim, na Seção 7.5 tem-se o resultado dos testes realizados e na Seção 7.6 tem-se a descrição da animação gerada através da aplicação desenvolvida.

7.1 Arquitetura da Aplicação

Para o desenvolvimento da aplicação optou-se pela utilização do OpenNI pelo fato do *framework* ser multiplataforma e possuir código aberto, assim como o Blender. O NiTE será utilizado como *middleware* de apoio, devido suas facilidades para o processamento do esqueleto do usuário.

Em um primeiro momento a aplicação foi inteiramente planejada dentro do Blender, em formato de *addon*, através da utilização do *Python Bindings*, devido a possibilidade de utilizar-se as funcionalidades do OpenNI e do NiTE na linguagem Python. Conforme pode-se observar na Figura 7.1, o *addon* Kinected Blender funcionaria juntamente com o Blender e se comunicaria com o OpenNI e o NiTE através do *Python Bindings*. Já o OpenNI faria a comunicação com o sensor Kinect através do *driver* do dispositivo, sendo que esse último não está embutido no OpenNI. De fato, no sistema operacional Windows seria utilizado o *driver* oficial do Kinect, disponibilizado pela Microsoft, conforme especificação do OpenNI. Já em sistemas Linux seria utilizado o *libfreenect*, que é um *driver* não-oficial desenvolvido pela PrimeSense. A opção pelo *libfreenect* deve-se a ausência de um *driver* oficial para essa plataforma. Por fim, seria utilizado o sensor *Kinect for Xbox 360* para os testes

da aplicação, devido ao fato de ser o único sensor disponível para a realização dos testes.

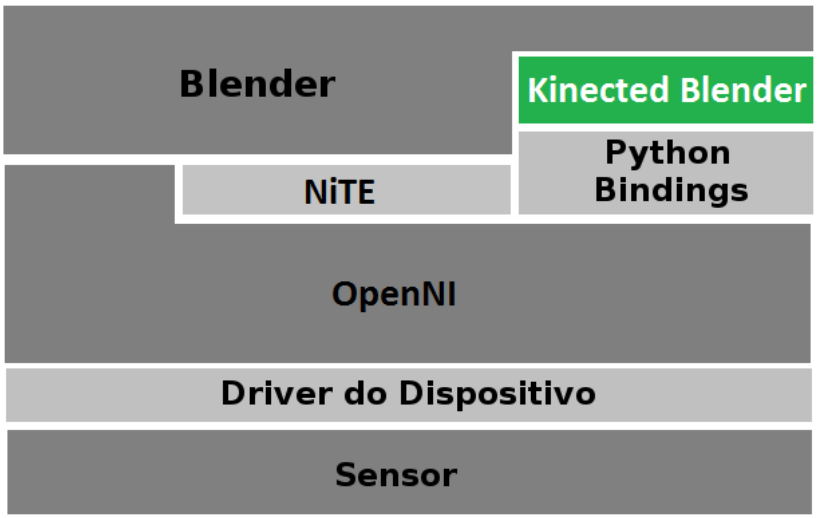


Figura 7.1: Arquitetura inicial da aplicação Kinected Blender.

Entretanto, não foi possível utilizar a arquitetura inicialmente planejada devido à existência de uma incompatibilidade entre o *Python Bindings* e o NiTE. As funções do *Python Bindings* são disponibilizadas através de uma DLL (*Dynamic-Link Library*) que transcreve as funções originais de C++ para Python. Entretanto, durante o desenvolvimento deste trabalho foi constatado que a última versão desta DLL era incompatível com a última versão do NiTE, impossibilitando a utilização de algumas funções cruciais para a aplicação, como por exemplo, a busca da posição de um determinado ponto do usuário mapeado.

Uma solução seria realizar *downgrade* da versão 2.2 para a versão 2.0, tanto na DLL quanto no NiTE. No entanto, a PrimeSense, empresa que desenvolvia o NiTE, o *libfreenect* e o *Python Bindings*, foi comprada pela Apple no final de novembro de 2013 (BBC, 2013), sendo que em 23 de abril de 2014 a Apple tirou do ar tanto o site da PrimeSense quanto o site da OpenNI, impossibilitando o *download* de qualquer uma dessas ferramentas.

Com isso, a utilização de *Python Bindings* tornou-se inviável, impossibilitando que toda a aplicação fosse escrita em Python e ocasionando uma mudança na arquitetura da aplicação. Na Figura 7.2 tem-se a nova arquitetura da aplicação desenvolvida. Observa-se que nesta arquitetura não foi utilizado o *Python Bindings*, sendo necessária o desenvolvimento uma solução cliente-servidor para a comunicação entre o Kinect e o *addon*. Nesta arquitetura, a aplicação cliente, chamada *Kinected Blender Receiver*, processa os dados do Kinect, e após este processamento, envia os dados para a aplicação servidor, chamada *Addon Kinected Blender*, através de uma comunicação por *socket*. Além disso, como o *libfreenect* também era disponibilizado

pela PrimeSense e foi retirado do site, tornou-se inviável desenvolver a aplicação para Linux, uma vez que não existem outros *drivers* disponíveis no momento.

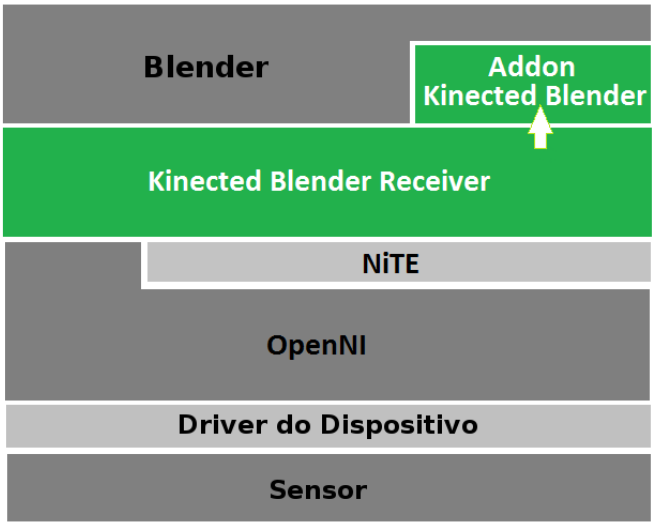


Figura 7.2: Nova arquitetura da aplicação Kinected Blender.

Apesar dessas limitações, optou-se por continuar utilizando-se o OpenNI e o NiTE a fim de evitar mudanças significativas na arquitetura. De fato, grande parte do código já havia sido desenvolvido, e uma mudança muito grande na arquitetura impossibilitaria a conclusão deste trabalho em tempo hábil. Além disso, como o fato é muito recente, torna-se possível que a comunidade continue os projetos OpenNI e NiTE e que outras empresas auxiliem no desenvolvimento dos mesmos.

7.2 Comunicação

Devido ao fato da solução desenvolvida ser uma aplicação cliente-servidor, necessitou-se a criação de um canal de comunicação. Desta forma, a comunicação entre o *Addon Kinected Blender* (servidor) e o *Kinected Blender Receiver* (cliente) ocorre através de uma conexão UDP (*User Datagram Protocol*) local, através da porta 5775. Utilizou-se essa porta uma vez que a mesma não corresponde a nenhum serviço registrado (IANA, 2014).

Além disso, optou-se pelo protocolo UDP devido ao fato do protocolo TCP (*Transmission Control Protocol*) ocasionar atrasos durante o processamento da animação em meio aos testes realizado. Este atraso deve-se ao fato do TCP requisitar a confirmação de entrega dos pacotes, o que inviabilizou o uso deste protocolo para a criação de animações em tempo real. Por fim, utilizou-se um *socket* não-bloqueante com o objetivo das aplicações não permanecerem bloqueadas até recepção de uma mensagem.

7.3 Addon Kinected Blender

O *addon* foi desenvolvido na linguagem Python e sua interface gráfica foi desenvolvida utilizando-se a API do *Blender*. Este *addon* inicializa um componente de tela no *Blender*, chamado de *Kinected Blender*, e possui um botão de "Start", conforme pode ser observado na Figura 7.3.

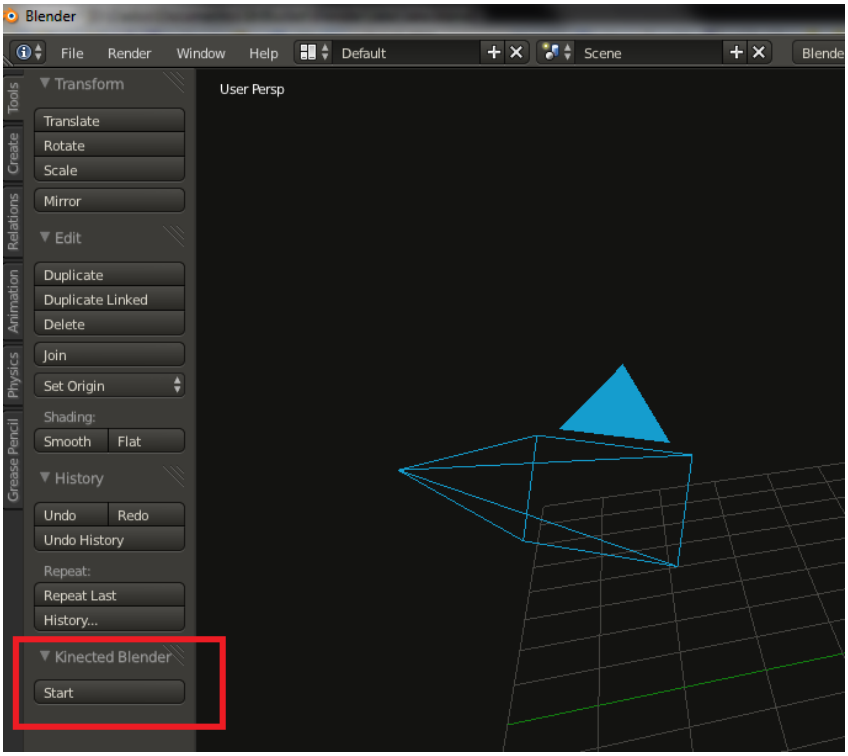


Figura 7.3: *Addon* Kinected Blender ativado.

No momento em que o botão de "Start" é pressionado, o *Blender* inicializa o servidor UDP e fica no aguardo de dados do Kinect. Ao receber as mensagens, o *addon* executa o processamento dessas através de uma função modal. Esse tipo de função no *Blender* apresenta como objetivo executar processos em um intervalo de tempo determinado, o que impede que a aplicação "trave" durante a execução de um *script* Python. O intervalo de tempo ideal para atualização da tela nesta aplicação foi estimada em 60 quadros por segundo. Com um número menor de quadros obteve-se pouca flexibilidade de manipulação da animação, enquanto que um número maior de quadros não foi possível de ser processado pelo *Blender*.

A função modal foi implementada através do uso de expressões regulares, e é responsável pela identificação dos dados pertencentes a cada ponto mapeado pelo Kinect. Os dados identificados pelas expressões regulares são transmitidos à objetos específicos do *Blender*, alterando sua posição em relação as coordenadas x, y e z. Os únicos objetos que não podem ter sua posição alteradas de forma direta são os objetos do tipo *bone*, sendo que para movê-los deve-se adicionar uma restrição no

bone de forma copiar o movimento de outro objeto, como por exemplo, um cubo. Ou seja, as posições desses objetos são alteradas e esses movimentos são copiados para os *bones* correspondentes. Estes objetos devem apresentar os nomes da Tabela 7.1 para poderem ser mapeados aos *bones*.

Tabela 7.1: Tabela para mapeamento de objetos vinculados às partes do corpo.

Objeto	Função
E.Head	Objeto a ser controlado pela cabeça
E.Neck	Objeto a ser controlado pelo pescoço
E.Shoulder.L	Objeto a ser controlado pelo ombro esquerdo
E.Shoulder.R	Objeto a ser controlado pelo ombro direito
E.Elbow.L	Objeto a ser controlado pelo cotovelo esquerdo
E.Elbow.R	Objeto a ser controlado pelo cotovelo direito
E.Hand.L	Objeto a ser controlado pelo punho esquerdo
E.Hand.R	Objeto a ser controlado pelo punho direito
E.Torso	Objeto a ser controlado pelo tronco
E.Hip.L	Objeto a ser controlado pelo quadril esquerdo
E.Hip.R	Objeto a ser controlado pelo quadril direito
E.Knee.L	Objeto a ser controlado pelo joelho esquerdo
E.Knee.R	Objeto a ser controlado pelo joelho direito
E.Foot.L	Objeto a ser controlado pelo tornozelo esquerdo
E.Foot.R	Objeto a ser controlado pelo tornozelo direito

Caso algum objeto não seja encontrado, os dados referentes a esse serão descartados, ou seja, o *addon* só irá transmitir os dados para os objetos existentes. Esta situação pode ser útil quando pretende-se animar somente uma parte do corpo de um personagem ou um personagem não-humanóide. Destaca-se ainda que, caso a opção *Automatic keyframe insertion* esteja ativada (Figura 7.4), o *addon* irá gravar automaticamente os *keyframes* da animação.

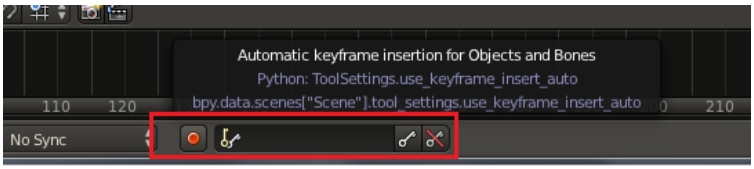


Figura 7.4: A função *Auto keyframe insertion* deve estar ativada para gravação da animação.

7.3.1 Mapeamento para *Armature* / *Bones*

Conforme descrito anteriormente, o *addon* transmite as informações processadas pelo Kinect somente para os objetos presentes na cena, não sendo possível a transmissão direta para *bones* de um *armature*. Isso se deve à necessidade de manter a proporção do personagem, ou seja, nem sempre o usuário será proporcionalmente do mesmo tamanho que o personagem modelado.

Para manter esta proporção, foi utilizada uma técnica de mapeamento de *bones* que encontra-se disponível no *Blender*. Os pontos mapeados do Kinect são transmitidos para objetos do tipo "*Empty*" que possuem os nomes citados na Tabela 7.1, sendo que objetos do tipo "*Empty*" são utilizados somente para manipulação de restrições, não aparecendo em uma animação. Após isso, um *armature* com tamanho proporcional do usuário é mapeado em cada *bone* correspondente no objeto "*Empty*". Após isso, um segundo *armature*, com o tamanho do personagem, deverá ser criado, sendo que cada *bone* desse *armature* deverá possuir um *bone constraint* de "*Copy Rotation*" de cada *bone* do *armature* mapeado anteriormente, fazendo com que os movimentos feitos com um *armature* sejam copiados para outro (Figura 7.5).

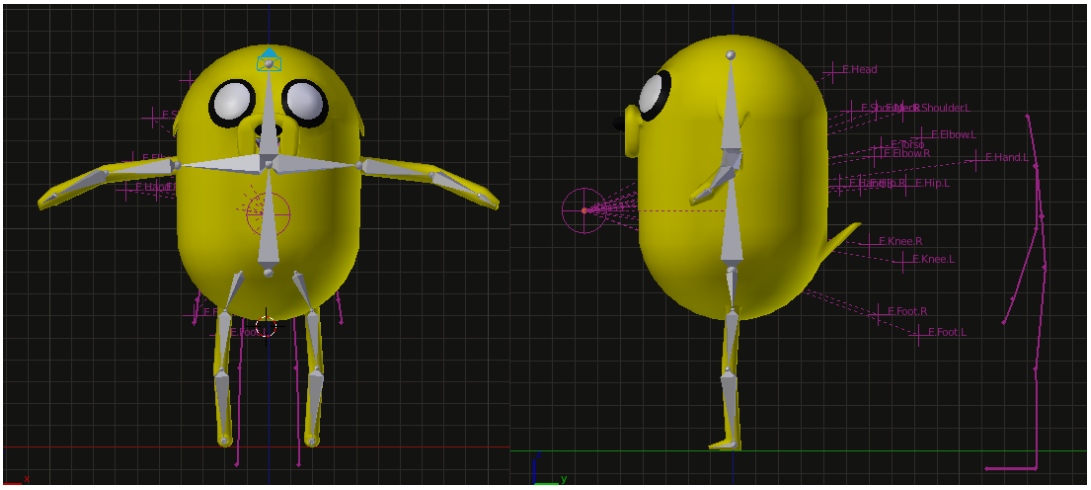


Figura 7.5: *Armature* cinza mapeado no personagem e *armature* rosa mapeada do usuário.

7.4 Kinected Blender Receiver

A aplicação foi escrita em C++, linguagem primária suportada pelo OpenNI e pelo NiTE e não possui interface gráfica. O *socket* deste programa é responsável por enviar mensagens para o *Addon Kinected Blender*. Os dados enviados na mensagem são obtidos através do uso do NiTE, sendo que somente um usuário pode ser detectado por vez. Todos os 15 pontos do corpo suportados pelo NiTE são capturados, após a calibragem ser feita através a "pose Psi".

Cada junta do corpo é processada individualmente, entretanto, nem sempre o valor atual da junta é transmitido, evitando-se que ruídos sejam repassados para o *Blender*. O ruído da imagem é causado por interferência de luz do sol ou da própria luz infravermelha ao tentar alcançar pontos cegos gerados pelas sombras. Essas interferências fazem com que os pontos capturados pelo NiTE mudem de posição mesmo com o usuário parado.

Para evitar que estes ruídos sejam passados para animação, é verificado o percen-

tual de deslocamento de cada ponto e o seu grau de confiança. O grau de confiança é obtido através do algoritmo *getPositionConfidence* do NiTE, que informa a probabilidade de um determinado ponto estar realmente no local indicado, sendo que a documentação do NiTE não informa maiores detalhes sobre o algoritmo. Já o percentual de deslocamento informa o quanto o ponto atual deslocou-se em relação à sua posição anterior, e é descrito pela equação

$$desloc_p = \left| \left(\frac{p_{n-1}}{p_n} - 1 \right) \times 100 \right|,$$

onde p representa a posição do ponto mapeado em um dos planos (x, y ou z) no tempo n.

Testes foram realizados com o grau de confiança, sendo que os melhores resultados foram obtidos utilizando-se um grau de confiança de 40%. Observa-se que valores superiores a 40% obteve-se uma perda de naturalidade nos movimentos. Isso deve-se ao fato do algoritmo não conseguir capturar movimentos rápidos. Já valores inferiores a 40% geravam uma grande quantidade de ruídos.

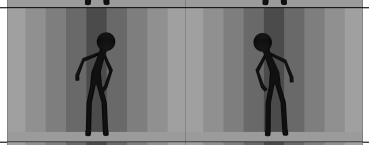
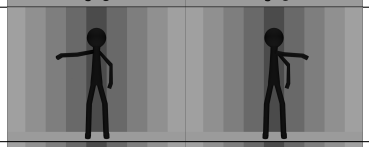
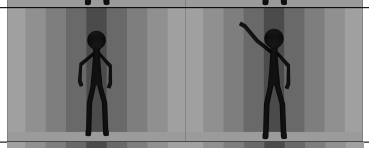
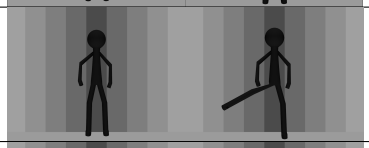
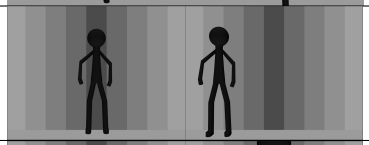
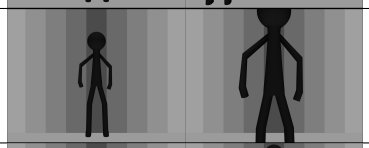
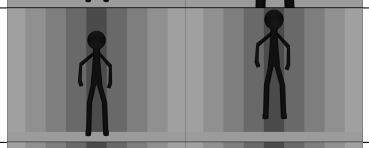
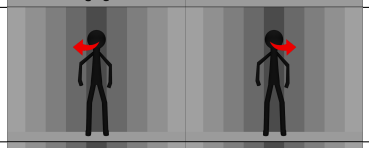
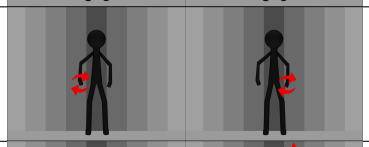
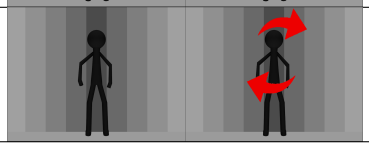
Nos testes realizados com o percentual de deslocamento, os melhores resultados foram obtidos através da utilização do percentual de 2%, sendo que com este percentual pequenos ruídos ainda são transmitidos. Já um valor superior a 2%, apesar de gerar menos ruído, provocavaram uma perda de pequenos movimentos voluntários do usuário, ocasionando limitações na reprodução da animação.

7.5 Testes Realizados

Para a realização dos testes dos movimentos capturados do Kinect foi utilizada uma adaptação do trabalho de Sá (2011). No trabalho citado foram realizados testes para o reconhecimento de gestos em jogos, enquanto neste trabalho os gestos serão adaptados de acordo com o movimento humano esperado.

Através dos testes realizados pode-se perceber certas limitações com o uso do sensor Kinect, sendo que alguns tipos de movimentos devem ser evitados para a animação do personagem. Estes gestos podem ser feitos manualmente no *Blender* ou ajustados posteriormente, ou seja, após os *keyframes* forem gravados. Além disso, para movimentos que geraram ruídos pode-se ter que realizar a edição da animação para a remoção dos mesmos. Na Tabela 7.2 tem-se os gestos realizados, bem como a reação obtida através dos testes.

Tabela 7.2: Tabela de gestos para reconhecimento.

Gesto	Representação	Reação Obtida
Inclinar a cabeça		Personagem inclinou a cabeça de forma proporcional ao movimento realizado
Inclinar o corpo		Personagem inclinou o corpo de forma proporcional ao movimento realizado. Foi detectada uma pequena elevação da perna oposta ao movimento
Mover o braço horizontalmente		Personagem moveu o braço horizontalmente. Pequenos ruídos foram detectados com os braços na frente do corpo
Mover o braço verticalmente		Personagem moveu o braço verticalmente
Mover a perna horizontalmente		Personagem moveu a perna com um movimento mais sutil devido ao tipo de modelagem realizada. Ao passar uma perna sobre a outra ruídos foram detectados
Mover a perna verticalmente		Personagem moveu a perna de forma proporcional ao movimento. Ao mover a perna para trás obteve-se ruído
Mover-se para os lados		Personagem moveu-se de forma proporcional
Mover-se para frente e para trás		Personagem moveu-se para trás até os limites do Kinect, e para frente até uma distância de aproximadamente 2,5m do sensor
Pular		Personagem pulou de forma proporcional ao movimento
Girar a cabeça		Movimento não-suportado
Girar os punhos		Movimento não-suportado
Girar o corpo em 360 graus		Personagem girou o corpo, porém com muitos ruídos

7.6 Animação

Após o desenvolvimento da aplicação, foi desenvolvida uma animação simples de um personagem 3D no *Blender*, através dos movimentos realizados pelo usuário. A animação gerada faz com que um cachorro de desenho animado dance num palco. O modelo e cenário utilizados foram modelados manualmente, assim como seu *armature*, sendo utilizada a técnica de cópia de movimentos descrita anteriormente. A dança é feita de forma simples, com o objetivo de repassar a naturalidade do movimento humano para o personagem. Na Figura 7.6 tem-se a modelagem do cachorro utilizada na animação. Já no CD em anexo tem-se a animação gerada.

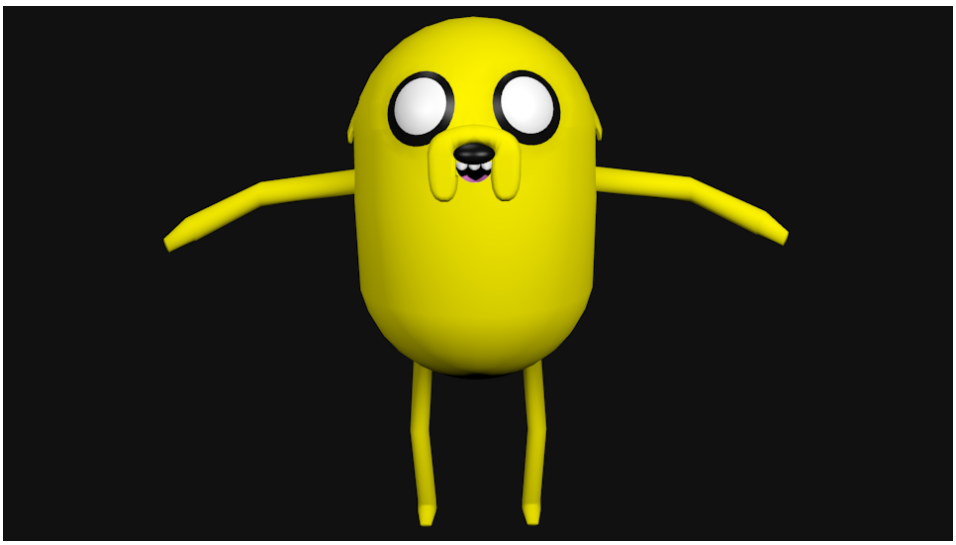


Figura 7.6: Modelo 3D utilizado na animação.

8 CONCLUSÕES

Ao longo do desenvolvimento deste trabalho, realizou-se um estudo sobre a integração entre o sensor Kinect e o *software* Blender com o objetivo de buscar uma simplificação no processo de criação de animações 3D. O Kinect é um sensor capaz de detectar movimentos, sendo que através dessa detecção pode-se capturar os movimentos de um ser humano. Já o Blender é um *software* utilizado para modelagem e animação 3D. Sendo assim, através do Blender torna-se possível a criação de personagens, movimentá-los e gravar esses movimentos para a criação de uma animação.

Entretanto, a animação de um personagem 3D é um processo trabalhoso devido ao fato do animador necessitar alterar a pose do personagem a cada quadro, dificultando o processo de animação. Sendo assim, com o objetivo de facilitar o processo de animação de um personagem, foi proposta uma integração do sensor Kinect com o Blender, utilizando-se a captura dos movimentos pelo Kinect para a geração de uma animação para um personagem 3D.

Atualmente, alguns *softwares* já realizam essa integração, entretanto, o desenvolvimento dessa aplicação torna-se interessante devido ao fato desses *softwares* não serem multiplataforma, gratuitos e atuais. Por exemplo, o Ni Mate é um aplicativo multiplataforma, entretanto não e gratuito. Já o Bloop é gratuito, todavia não é um aplicativo multiplataforma e está desatualizado desde 2011.

Desta forma, neste trabalho foi desenvolvido um *addon* para o Blender que permite a integração desse com o sensor Kinect através do uso do *framework* OpenNI e do *middleware* NiTE. Optou-se por essas ferramentas devido ao fato dessas serem gratuitas e multiplataforma. A utilização do pacote *Python Bindings*, que permitia a utilização da linguagem Python em conjunto com o OpenNI e o NiTE, não foi possível devido a uma incompatibilidade presente em sua última versão, além do fato do *downgrade* tornar-se inviável devido à aquisição da PrimeSense pela empresa Apple. Sendo assim, os objetivos propostos foram obtidos de forma parcial, sendo que foi possível realizar a integração entre o sensor e o *software*, entretanto, a solução desenvolvida não é multiplataforma devido a ausência de *drivers* para o

sistema operacional Linux. Por fim, mesmo com as limitações atuais do sensor e da biblioteca utilizada, foi possível obter a naturalidade do movimento do corpo humano e repassar a mesma para a animação do personagem através da aplicação desenvolvida.

8.1 Trabalhos Futuros

Devido ao fato do fechamento da PrimeSense ser algo recente, ainda não é possível determinar o futuro das bibliotecas OpenNI e NiTE. Ao passar do tempo deve-se observar o destino das bibliotecas, com o objetivo de não permitir que a aplicação fique obsoleta. Caso os projetos não sejam seguidos, deve-se alterar a arquitetura da aplicação de forma a utilizar outra biblioteca, como por exemplo a *Microsoft Kinect SDK*. Essa alteração na arquitetura permitiria a utilização de novos sensores, bem como manteria a biblioteca atualizada. Por fim, caso os projetos sejam continuados, deve-se verificar a possibilidade de expandir a solução para sistema operacional Linux através de um *driver* do sensor.

REFERÊNCIAS

Autodesk. **3ds Max - Software de Modelagem e Renderização 3D**. <Disponível em: http://www.autodesk.com.br/products/autodesk-3ds-max/overview>. Acesso em 17 de novembro de 2013.

Autodesk. **Maya - Software de animação 3D / Animação computadorizada**. <Disponível em: http://www.microsoft.com/en-us/kinectforwindows/>. Acesso em 17 de novembro de 2013.

BBC. **BBC News**. <Disponível em: http://www.bbc.com/news/technology-25083914>. Acesso em 02 de maio de 2014.

BECKERMAN, H. **Animation - The Whole Story**. New York, Estados Unidos: Allworth Press, 2012.

Blender Foundation. **Blender**. <Disponível em: http://www.blender.org/>. Acesso em 01 de setembro de 2013.

Blender Wiki. **Blender Wiki - Blender's Official Documentation**. <Disponível em: http://wiki.blender.org/>. Acesso em 27 de outubro de 2013.

BORENSTEIN, G. **Making Things See**. [S.l.]: Make, 2012.

CATUHE, D. **Programming with the Kinect for Windows Software Development Kit**. Washington, Estados Unidos: Microsoft Press, 2012.

Delicode. **Ni Mate**. <Disponível em: http://www.ni-mate.com/>. Acesso em 15 de novembro de 2013.

ECKER, W.; MüLLER, W.; DöMER, R. **Hardware-dependent Software**: principles and practice. Holanda: Springer, 2009.

FALAHATI, S. **OpenNI Cookbook**. Birmingham, Inglaterra: Packt, 2013.

Gilles Charbonneau. **YouTube - Animation Principle -**. <Disponível em: http://www.youtube.com/watch?v=sZHdfHJtNWc>. Acesso em 20 de junho de 2014.

GIORIO, C.; FASCINARI, M. **Kinect in Motion - Audio and Visual Tracking by Example**. Birmingham, Inglaterra: Packt, 2013.

IANA. **IANA - Internet Assigned Numbers Authority**. <Disponível em: https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.txt>. Acesso em 07 de junho de 2014.

JANA, A. **Kinect for Windows SDK Programming Guide**. Birmingham, Inglaterra: Packt, 2012.

Jornal da Globo. **YouTube - Saiba como é feita a animação no Brasil e nos EUA**. <Disponível em: http://www.youtube.com/watch?v=oxb_M8w_Bn4>. Acesso em 20 de junho de 2014.

MASHALKAR, J. **Motion Capture and Rigging using Microsoft Kinect**. <Disponível em: http://www.cse.iitb.ac.in/~jaaim/seminar/mocapKinectReport.pdf>. Acesso em 01 de setembro de 2013.

Microsoft. **Kinect for Windows — Voice, Movement and Gesture Recognition Technology**. <Disponível em: http://www.microsoft.com/en-us/kinectforwindows/>. Acesso em 10 de novembro de 2013.

Microsoft. **Kinect for Windows Dev Center**. <Disponível em: http://www.microsoft.com/en-us/kinectforwindowsdev/default.aspx>. Acesso em 10 de novembro de 2013.

MULLEN, T. **Introducing Character Animation With Blender**. 2.ed. Indianopolis, Estados Unidos: Sybex, 2011.

OpenKinect. **OpenKinect**. <Disponível em: http://www.primesense.com/>. Acesso em 09 de novembro de 2013.

OpenNI. **Open-source SDK for 3D sensors - OpenNI**. <Disponível em: http://www.openni.org/>. Acesso em 08 de novembro de 2013.

OSC. **Open Sound Control**. <Disponível em: http://opensoundcontrol.org/>. Acesso em 16 de novembro de 2013.

PrimeSense. **3D Sensing Technology Solutions - PrimeSense**. <Disponível em: http://www.primesense.com/>. Acesso em 09 de novembro de 2013.

Python Tools. **Python Tools for Visual Studio**. <Disponível em: http://pytools.codeplex.com/wikipage?title=PyKinect>. Acesso em 10 de novembro de 2013.

Sá, J. G. P. **Construindo uma DSL para reconhecimento de gestos utilizando Kinect**. <Disponível em: http://www.cin.ufpe.br/~tg/2011-2/jgps.pdf>. Acesso em 02 de junho de 2014.

TZi. **Arbeitsgruppe Digitale Medien**: bloop. <Disponível em: http://dm.tzi.de/en/bloop/>. Acesso em 15 de novembro de 2013.

UNIVESP. **YouTube - Lições de Animação**: animação no brasil. <Disponível em: http://www.youtube.com/watch?v=WqklB6oln7k>. Acesso em 20 de junho de 2014.

VASCONCELOS, V. **Blender 2.5 Character Animation Cookbook**. Birmingham, Inglaterra: Packt, 2011.

WIGDOR, D.; WIXON, D. **Brave NUI World**: designing natural user interfaces for touch and gesture. Burlington, Estados Unidos: Morgan Kaufmann, 2011.