UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

MODELAGEM TRIDIMENSIONAL DE AMBIENTE UTILIZANDO KINECT

DANIEL URIO MENDES

BLUMENAU
2012

2012/2-08

DANIEL URIO MENDES

MODELAGEM TRIDIMENSIONAL DE AMBIENTE UTILIZANDO KINECT

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciência da Computação — Bacharelado.

Prof. Aurélio Faustino Hoppe - Orientador

BLUMENAU
2012

2012/2-08

MODELAGEM TRIDIMENSIONAL DE AMBIENTE

UTILIZANDO KINECT

Por

DANIEL URIO MENDES

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Aurélio Faustino Hoppe, Mestre – Orientador, FURB
Membro:	Prof. Dalton Solano Dos Reis, Mestre – FURB
Membro:	Prof. Mauro Marcelo Mattos, Mestre – FURB

Blumenau, 11 de dezembro de 2012

Dedico este trabalho à minha família e todos os meus amigos, especialmente aqueles que me ajudaram diretamente na realização deste.

AGRADECIMENTOS

À minha família, e em especial aos meus pais, Daniel Mendes e Leonilda Fatima Mendes, pelo amor, compreensão, confiança e apoio que me incentivaram para o término de mais esta etapa.

Aos meus colegas e amigos, pelos empurrões, cobranças, apoio, companheirismo e principalmente pela compreensão durante essa etapa.

À todas as outras pessoas que de uma forma ou de outra contribuíram para o meu desenvolvimento intelectual e pessoal, e em especial ao professor Aurélio Faustino Hoppe, pela sabedoria e competência na orientação deste trabalho.

Em grupo, se sobrepor aos outros é burrice, procure buscar a junção de potenciais.

RESUMO

Este trabalho descreve a implementação de um algoritmo para criação de um modelo tridimensional do ambiente detectado através do sensor Kinect. Para desenvolver este trabalho foi utilizado um Kinect, o Kinect *for* Windows SDK, WPF e OpenTK. A partir dos dados coletados do Kinect através de seu SDK é criado uma malha de triângulos, formando a superfície, e apresentado usando funções de desenho tridimensionais do OpenTK. O trabalho gera a superfície em tempo real, os dados capturados do Kinect não são armazenados, assim ela gera a superfície a cada *frame* detectado pelo Kinect e elimina os dados do *frame* anterior.

Palavras-chave: Reconstrução de superfícies. Kinect. Kinect *for* Windows SDK.

ABSTRACT

This work describes the implementation of an algorithm for creating a three-dimensional model of the environment detected by the Kinect sensor. To develop this work we used a Kinect, Kinect for Windows SDK and OpenTK. From the data collected through Kinect SDK is created a mesh of triangles forming the surface, and presented using three-dimensional drawing functions of OpenTK. The work generates the surface in real time, the captured data from Kinect are not stored, so it generates the surface every frame detected by the Kinect and deletes the data from the previous frame.

Key-words: Surface reconstruction. Kinect. Kinect for Windows SDK.

LISTA DE ILUSTRAÇÕES

Figura 1 - Visão interna do Kinect 14
Figura 2 - Ângulo de visão do Kinect 15
Figura 3 - Pontos projetados pelo projetor infravermelho em uma folha de papel 16
Figura 4 - Distância de coleta de dados de profundidade 17
Figura 5 - Luz estruturada projetada em superfície irregular 18
Figura 6 - Limite de detecção do Kinect 19
Figura 7 – *Shift* 20
Quadro 1 - Fórmula para calibração 20
Figura 8 - Volta parcial 22
Figura 9 - Volta completa 22
Figura 10 - Estimação dos planos tangentes 24
Figura 11 - Abordagem por grafos para orientação dos planos tangentes 24
Figura 12 - Vetores normais orientados 25
Figura 13 - Construção da superfície através de vértices do cubo 26
Figura 14 - Algumas maneiras que a superfície intercepta os cubos lógicos 27
Figura 15 - A triangulação dos 15 casos representativos 28
Figura 16 - Qualidade da superfície para diferentes tamanhos de cubo 28
Figura 17 – KinectFusion 30
Figura 18 - Simulação de física 30
Figura 19 – KinectCAD 31
Figura 20 - Kinect SDK *dynamic time warping gesture recognition* 32
Quadro 2 - Características dos trabalhos relacionados 33
Figura 21 - Diagrama de casos de uso 35
Quadro 3 - Caso de uso 01 35
Quadro 4 - Caso de uso 02 36
Quadro 5 - Caso de uso 03 36
Figura 22 - Diagrama de classes 37
Quadro 6 - Localizar Kinect conectado ao computador 39
Quadro 7 - Ativação de *stream* e criação de eventos 40
Quadro 8 - Criação da *thread* 40
Quadro 9 - Obter coordenadas tridimensionais 41

Figura 23 - Método de reconstrução de superfície 42
Quadro 10 - Gerando superfície 42
Figura 24 - Reconstrução com e sem buracos 43
Quadro 11 - Limpar tela e criar matriz de visualização 43
Figura 25 - comparação de ambiente virtual com e sem luz 44
Quadro 12 - Adicionar luz ao ambiente e definir angulo de visão da câmera 45
Quadro 13 - Desenhando superfície 45
Figura 26 - Janela de apresentação de s*keleton* e vídeo 46
Figura 27 - Janela de apresentação da reconstrução da superfície 47
Figura 28 - Etapas do processo de reconstrução de superfície 48
Figura 29 - imperfeições na superfície reconstruída 48
Figura 30 - Exemplo de reconstrução de superfície 49
Figura 31 - Reconstrução de superfície incorreta 49
Quadro 14 - Comparação da aplicação desenvolvida com os trabalhos correlatos 50

LISTA DE SIGLAS

API – *Application Programming Interface*

CATIA - *Computer Aided Three-dimensional Interactive Application*

DSP - *Digital Signal Processing*

FPS - *Frames Per Second*

ICP - *Iterative Closest Point*

NUI - *Natural User Interface*

PLY - *PoLYgon file format*

RF - Requisito Funcional

RNF - Requisito Não-Funcional

SDK - *Software Development Kit*

SRU – Sistema de Refêrencia do Universo

WPF - Windows *Presentation Foundation*

UML - *Unified Modeling Language*

XAML - *Extensible Application Markup Language*

XML - *eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO 12
1.1 OBJETIVOS DO TRABALHO 12
1.2 ESTRUTURA DO TRABALHO 13
2 FUNDAMENTAÇÃO TEÓRICA 14
2.1 KINECT 14
2.1.1 Áudio, vídeo e motor de inclinação 15
2.1.2 Profundidade 16
2.1.2.1 Luz estruturada 17
2.1.3 Kinect *for* Windows SDK 18
2.2 MÉTODOS DE RECONSTRUÇÃO DE SUPERFÍCIES 21
2.2.1 Reconstrução incremental de superfície 21
2.2.2 Reconstrução a partir de pontos desorganizados 23
2.2.2.1 Marching Cubes 26
2.3 TECNOLOGIAS PARA DESENVOLVIMENTO DE INTERFACES GRÁFICAS 28
2.4 TRABALHOS CORRELATOS 29
3 DESENVOLVIMENTO 34
3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO 34
3.2 ESPECIFICAÇÃO 34
3.2.1 Diagrama de casos de uso 34
3.2.2 Diagrama de classes 36
3.3 IMPLEMENTAÇÃO 38
3.3.1 Técnicas e ferramentas utilizadas 38
3.3.2 Implementação da aplicação 39
3.3.2.1 Inicialização do Kinect 39
3.3.2.2 Reconstrução de superfícies 40
3.3.3 Operacionalidade da implementação 45
3.4 RESULTADOS E DISCUSSÃO 47
4 CONCLUSÕES 51
4.1 EXTENSÕES 51
REFERÊNCIAS BIBLIOGRÁFICAS 53

1 INTRODUÇÃO

Desde o surgimento dos *vídeo games*, também conhecidos como *consoles*, esses dispositivos estimulam o avanço e a inovação tecnológica (AGUIAR, 2010a). Atualmente os *consoles* estão alterando a forma de interagir com o usuário; ao invés de apertar apenas o botão no controle, eles estão capturando movimentos do usuário, como por exemplo, o levantar da mão do usuário. Para o *console* XBOX 360 da empresa Microsoft foi desenvolvido o acessório Kinect que permite captar o ambiente e movimentos a sua frente e transmitir para o XBOX 360 fazendo o usuário interagir com o *console* sem necessidade de estar segurando ou ter algum equipamento conectado a seu corpo.

O Kinect é uma tecnologia recente, em junho de 2011, a Microsoft disponibilizou o Kinect *for* Windows S*oftware Development Kit* (SDK), em versão beta, e a primeira versão para criação de produtos comerciais foi lançado em fevereiro de 2012. Desde o lançamento da primeira versão beta, já gerou uma grande quantidade de novos software e aplicações, criando novas possibilidades tanto para criação de novos programas ou melhorar a usabilidade de programas já existentes.

A modelagem tridimensional tem o objetivo de recuperar as informações geométricas e topológicas de um objeto ou ambiente, essa reconstrução acontece a partir de um processamento sobre um conjunto de amostras, que contém uma quantidade finita de informações do objeto.

Assim, neste trabalho desenvolveu-se uma aplicação utilizando o Kinect *for* Windows SDK para capturar um ambiente real e fazer uma modelagem tridimensional do ambiente para demonstrar a utilização do Kinect *for* Windows SDK.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é utilizar o Kinect para desenvolver uma aplicação que extraía uma nuvem de pontos do ambiente real para gerar a modelagem tridimensional virtual dessa ambiente.

O objetivo específico do trabalho é fazer a reconstrução do ambiente a partir da nuvem de pontos capturados pelo Kinect.

1.2 ESTRUTURA DO TRABALHO

Este trabalho está subdividido em capítulos que serão abordados a seguir.

O primeiro capítulo apresenta a justificativa para o desenvolvimento do objetivo do trabalho e alguns dados acerca do tema escolhido.

O segundo capítulo trata o funcionamento do Kinect, métodos para criação de modelos tridimensionais e métodos de reconstrução de superfícies.

O terceiro capítulo exibe a metodologia adotada durante o desenvolvimento do trabalho, assim como os resultados obtidos pela aplicação.

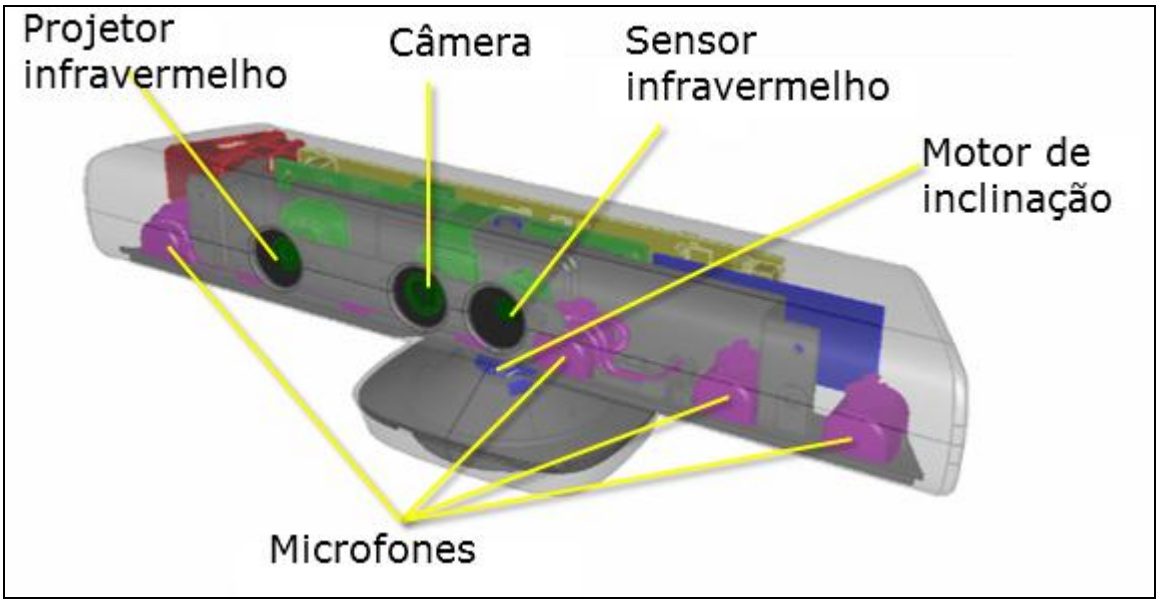
As conclusões são expostas no quarto capítulo, onde sugestões para trabalho futuros também podem ser encontradas.

2 FUNDAMENTAÇÃO TEÓRICA

Nas próximas seções são detalhados o funcionamento do Kinect, métodos para criação de modelos tridimensionais e métodos de reconstrução de superfícies. Na seção 2.1 é detalhado o funcionamento do Kinect e o SDK, os métodos para criação de modelos tridimensionais serão tratados na seção 2.2. Métodos de reconstrução de superfícies serão tratados na seção 2.3. Por fim, a seção 2.4 apresenta alguns trabalhos relacionados.

2.1 KINECT

Conforme Microsoft (2012f), o Kinect é um dispositivo que retorna informações do ambiente real utilizando: um projetor infravermelho, uma câmera infravermelha, uma câmera de vídeo e quatro microfones, para auxiliar a captura de dados ele também tem um motor de inclinação e através do Kinect *for* Windows SDK pode acessar as informações coletas.



Fonte: adaptado de Microsoft (2012f).

Figura 1 - Visão interna do Kinect

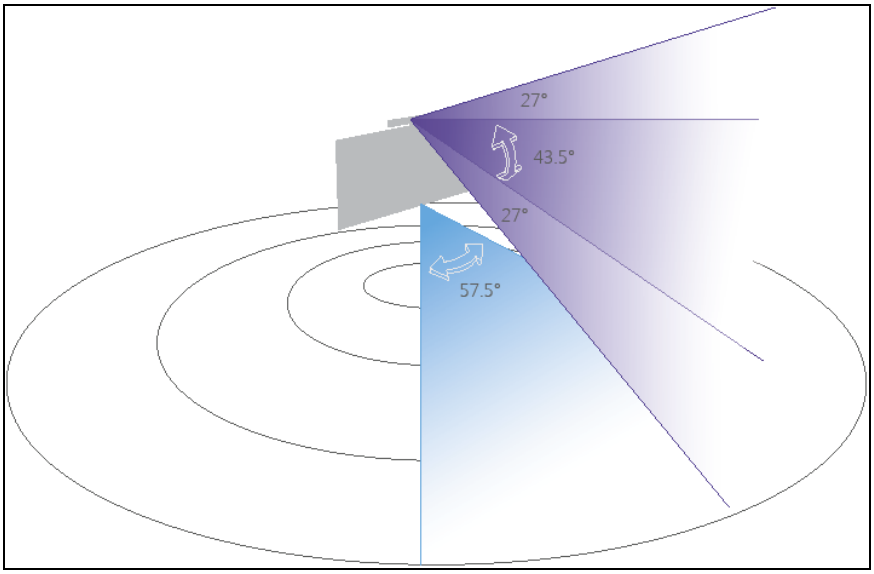
A seguir é detalhado o funcionamento dos componentes do Kinect: áudio, vídeo, motor de inclinação, profundidade e Kinect *for* Windows SDK.

2.1.1 Áudio, vídeo e motor de inclinação

O Kinect possui quatro microfones e *Digital Signal Processing* (DSP). Isso permite que ele distingua a direção da fonte sonora para saber qual fonte sonora está relacionada a um objeto ou pessoa, permitindo também ignorar sons indesejáveis e ecos (MILES, 2012, p. 6).

O Kinect prioriza por padrão sons mais altos e que venham da frente dele num limite de 100º, e a cada 10º ele pode determinar a direção da origem do som (MICROSOFT, 2012e).

Conforme Microsoft (2012b), Kinect possui limitações de ângulo de visão que ele consegue coletar informações. O ângulo de visão é de 57,5º na horizontal e 43,5º na vertical. O motor de inclinação pode movimentar o ângulo de visão vertical em até 27º para cima ou para baixo. A Figura 2 representa os ângulos de visão, sendo o feixe em azul o ângulo de visão horizontal e o feixe em roxo representa o ângulo de visão vertical, demonstrando também a possibilidade de alterar a direção de visão em 27º para cima ou para baixo.



Fonte: Microsoft (2012e).

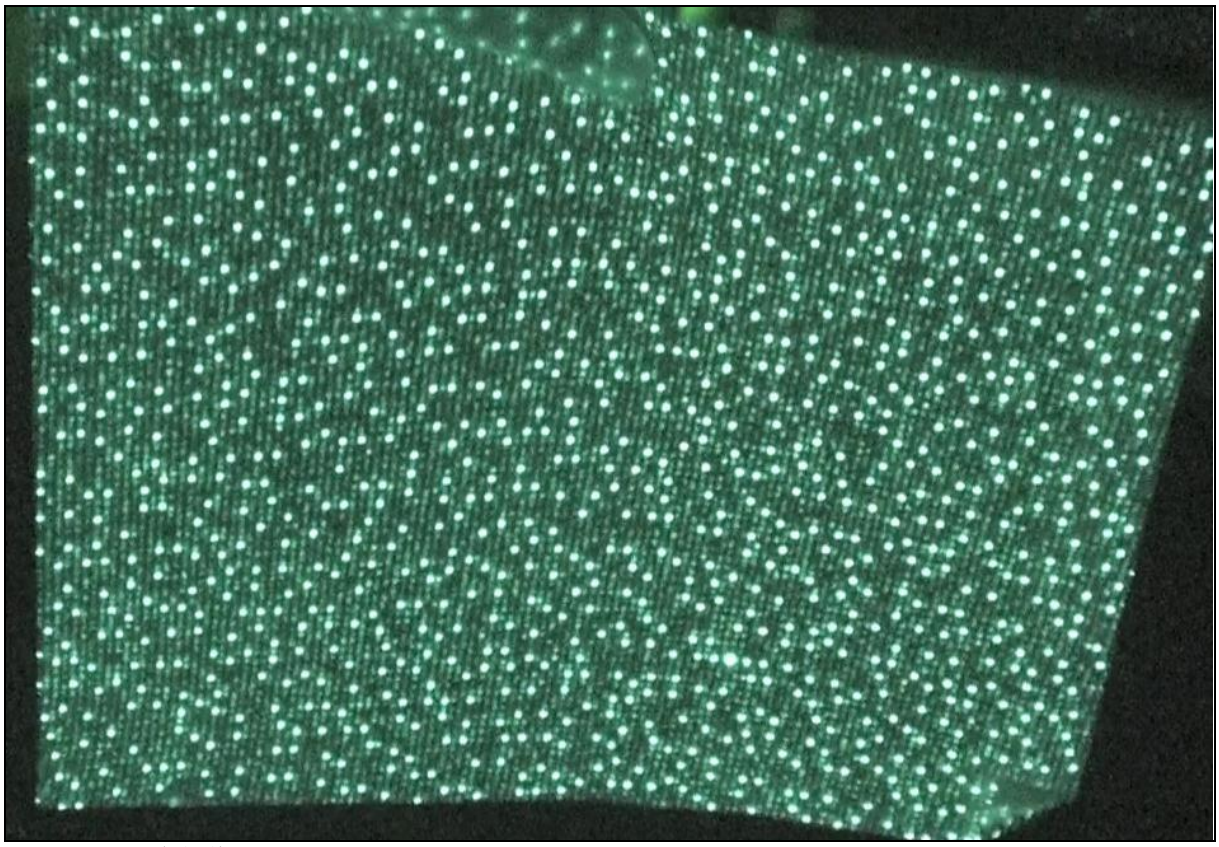
Figura 2 - Ângulo de visão do Kinect

O Kinect possui uma câmera de vídeo para capturar imagens. Essa é comparada a uma *webcam* comum já que ela não possui nenhum recurso especial ou uma resolução superior a *webcams*. A resolução máxima é de 1280x1024, mas normalmente é utilizada em 640x480 devido a quantidade de *Frames Per Second* (FPS). Na resolução de 1280x1024 são 12 FPS e em 640x480 são 30 FPS (MILES, 2012, p. 5-6).

2.1.2 Profundidade

O Kinect foi desenvolvido pela Microsoft, numa parceria com uma empresa israelense que produz câmeras que retornam mapas de profundidade, a PrimeSense que usa método para detectar a profundidade na cena, baseado em luz estruturada (QUEIROZ, 2010a).

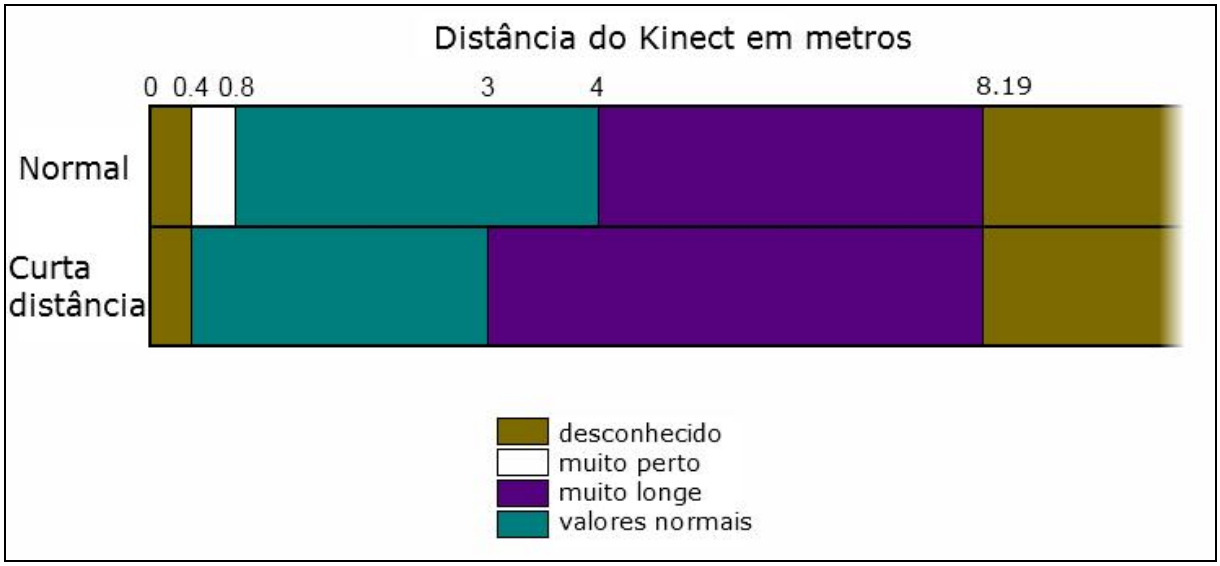
Para usar o método de luz estruturada o Kinect possui dois elementos. O primeiro é um projetor infravermelho, o qual projeta vários pontos infravermelhos no ambiente a sua frente sendo esses pontos a luz estruturada (Figura 3). O segundo é uma câmera infravermelha, a qual detecta estes pontos e conforme a distância entre estes pontos é calculada a distância desta área até o sensor (MILES, 2012, p. 7).



Fonte: Lowry (2010).

Figura 3 - Pontos projetados pelo projetor infravermelho em uma folha de papel

A distância que o Kinect retorna informações de profundidade é de acordo com a aplicação, podendo essa escolher entre dois modos, distância normal e o modo curta distância (Figura 4). O Kinect vai retornar dados de profundidade que estiverem além desses limites, mas serão de baixa qualidade e precisão (MICROSOFT, 2012a).



Fonte: adaptado de Microsoft (2012a).

Figura 4 - Distância de coleta de dados de profundidade

A Figura 4 apresenta os limites em metros que o Kinect retorna informações de profundidade, sendo a primeira barra horizontal do modo normal e a segunda do modo curta distância. As cores possuem a seguinte notação:

a) marrom: indica que a distância do objeto até o Kinect é desconhecida, os valores de distância estará incorreto;
b) branca: indica que o objeto está muito perto do Kinect, os valores de distância estará incorreto;
c) roxo: indica que o objeto está muito longe do Kinect, os valores de distância serão de baixa precisão;
d) ciano: indica que os valores da distância terão uma boa precisão.

A seguir é detalhado o método de luz estruturada para obtenção do mapa de profundidade utilizado pelo Kinect.

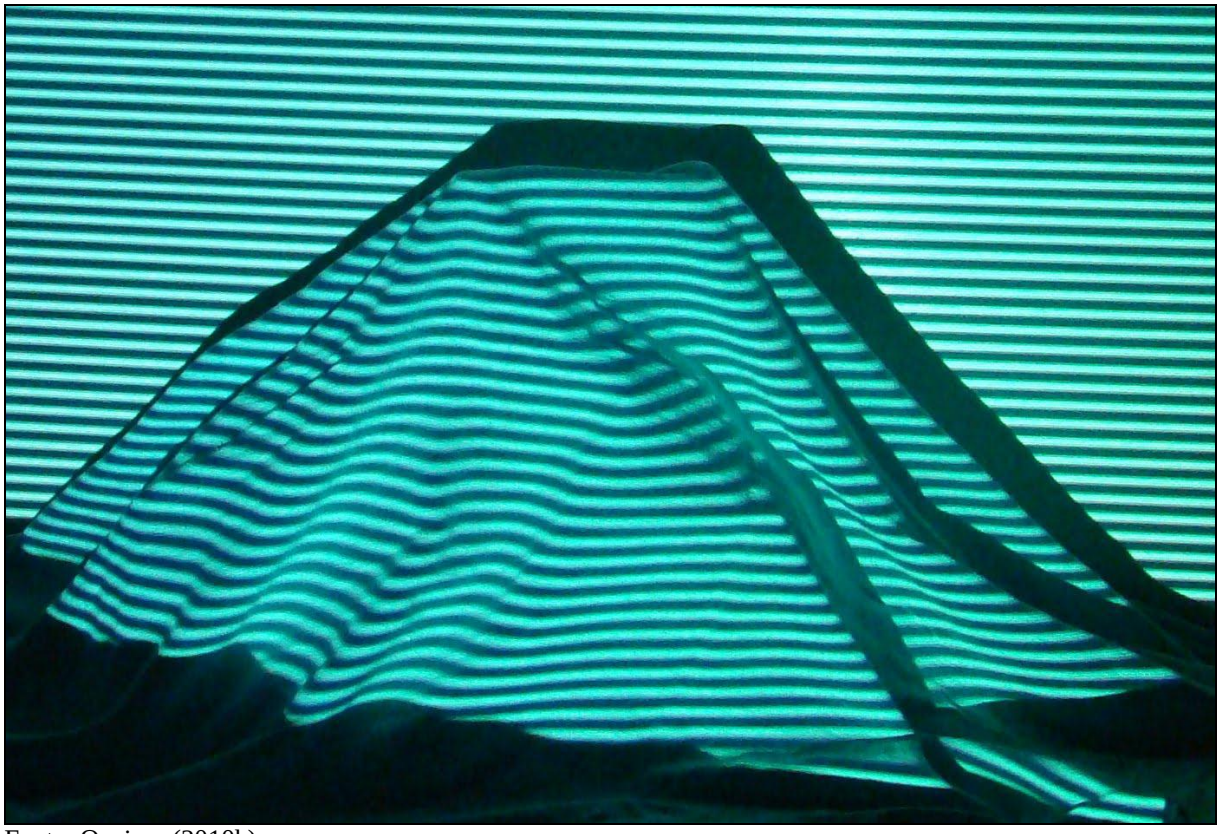
2.1.2.1 Luz estruturada

A técnica de reconstrução por luz estruturada envolve o uso de um projetor e uma câmara, na qual o sistema de projeção é usado para projetar o padrão sobre o objeto na cena (Figura 5) e uma câmara captura a imagem dos padrões que são distorcidos pela superfície de projeção. O projetor é equivalente à segunda câmara usada em fotogrametria[1] e qualquer

[1] Técnica e a arte de extrair de fotografias métricas, a forma, as dimensões e a posição dos objetos nelas contidos

ponto projetado na cena com direção e identificação conhecidas pode ser reconstruído usando os princípios de intersecção dos raios de luz da fotogrametria analítica (relação câmara-projetor) (CAMARGO, 2008).

A determinação das relações geométricas existentes entre o sensor e o padrão projetado é feita por um processo de calibração. O sistema de luz estruturada captura uma superfície em particular para cada ponto de vista (CAMARGO, 2008).



Fonte: Queiroz (2010b).

Figura 5 - Luz estruturada projetada em superfície irregular

A partir da Figura 5 pode-se observar um exemplo de luz estruturada, que são linhas horizontais com uma espessura fixa e com uma distância fixa entre cada linha. Ao ser projetada sobre uma superfície irregular a espessura e o espaço das linhas é alterado. Calculando-se a diferença gerada na luz estruturada e possível determinar a distância até superfície.

2.1.3 Kinect *for* Windows SDK

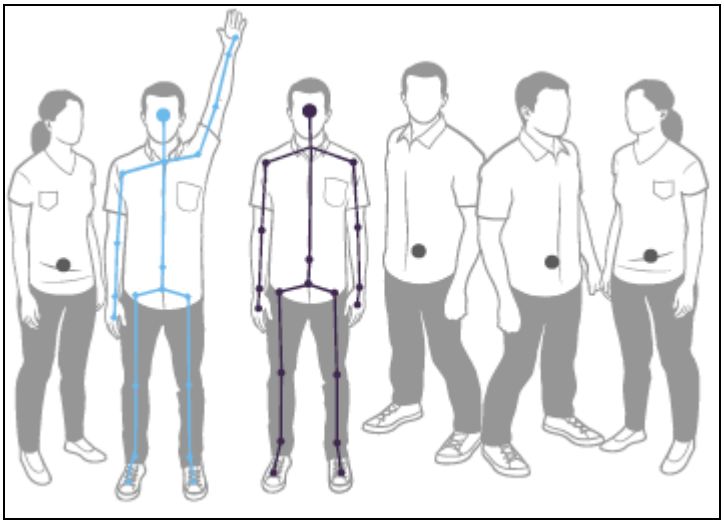
O Kinect *for* Windows SDK foi lançada em junho de 2011 pela Microsoft como versão beta. Em fevereiro de 2012 foi lançada a primeira versão final do SDK, permitindo a

utilização de mais de um Kinect em uma aplicação. Também foi melhorado a detecção de esqueleto e adicionado o modo de curta distância (EISLER, 2012).

A aplicação que vai utilizar o Kinect *for* Windows SDK pode ser desenvolvida nas linguagens de programação C#, VB.NET e C++ (MICROSOFT, 2012g).

O Kinect *for* Windows SDK contém duas *Application Programming Interface* (API): a *Natural User Interface* [2](NUI) API e a áudio API. Estas API permitem acesso aos *streams* de áudio, imagem, e profundidade do Kinect (MICROSOFT, 2012d).

Além de acesso aos *streams*, as API permitem acessar o motor de inclinação e ter acesso a dados já processados. *Skeletal tracking* é um desses dados já processados pelo SDK que detecta até seis pessoas na frente do Kinect e retorna o esqueleto que são as coordenadas de 20 ou 10 *joints* de até duas pessoas dentre as seis detectadas (Figura 6). Com os *joints* podem ser usados para detecção de movimentos dos usuários, fazendo a ligação destes *joints* forma-se um esqueleto do corpo da pessoa (MICROSOFT, 2012h).



Fonte: Microsoft (2012h).

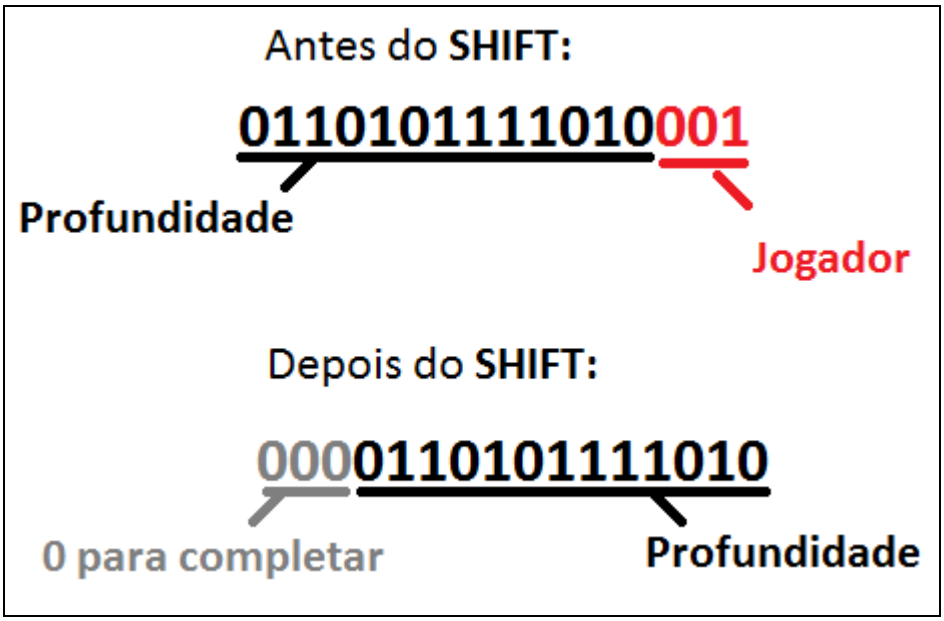
Figura 6 - Limite de detecção do Kinect

Conforme Queiroz (2010a), para a detecção de pessoas a Microsoft desenvolveu o Kinect usando técnicas de aprendizagem de máquina, utilizando uma gigantesca quantidade de dados de captura de movimentos usados em filmes e jogos antigos. Esses dados foram processados para inferir regras para a detecção de pessoas. Contando com uma enorme variação entre as pessoas: altos, baixos, magros, gordos, crianças, adultos, entre outros.

Conforme Castro (2012) o *stream* de profundidade do Kinect retorna um *array* de *short* que contém o valor da distância nos treze bits à esquerda e a identificação do jogador nos três bits à direita, para eliminar a informação do jogador pode usar o comando *shift* para

[2] Interface de usuário projetada para utilizar comportamentos para interagir diretamente com o conteúdo digital.

direita com valor três. Na Figura 7 é demonstrada a eliminação da informação do jogador utilizando o comando *shift*.



Fonte: Castro (2012).

Figura 7 – *Shift*

O valor da profundidade informado pelo Kinect pode chegar até 8191 milímetros e a resolução captada para profundidade é de no máximo 640x480, conforme Burrus (2011) para obter as coordenadas (x,y,z) dos pixels é necessário fazer a calibração dos valores x e y da resolução, com o valor da profundidade, utilizando a fórmula exibida no Quadro 1 para cada pixel, onde é obtido os valores para fazer o cálculo da seguinte maneira (Burrus, 2011):

a) o valor de cx_d é 3.3930780975300314e+02;
b) o valor de fx_d é 5.9421434211923247e+02;
c) o valor de cy_d é 2.4273913761751615e+02;
d) o valor de fy_d é 5.9104053696870778e+02;
e) resoluçãoX é o valor da coordenada x da resolução;
f) resoluçãoY é o valor da coordenada y da resolução;
g) profundidade é o valor da profundidade da coordenada (x,y) da resolução.

```
x = (resoluçãoX - cx_d) * profundidade / fx_d
y = (resoluçãoY - cy_d) * profundidade / fy_d
z = profundidade
```

Fonte: adaptada de Burrus (2011).

Quadro 1 - Fórmula para calibração

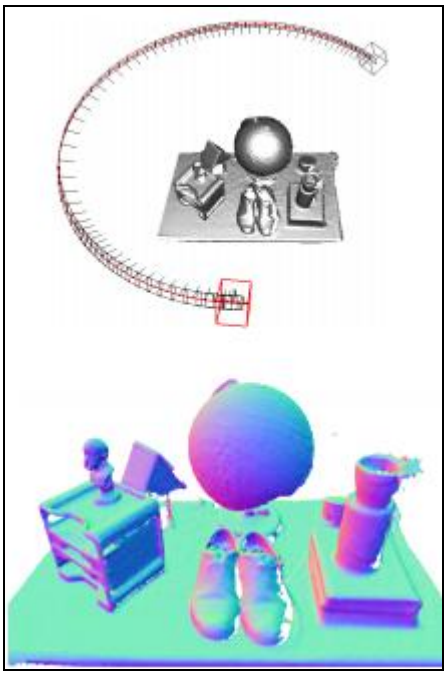
2.2 MÉTODOS DE RECONSTRUÇÃO DE SUPERFÍCIES

Existem vários métodos de reconstrução de superfícies, sendo que eles são definidos pela natureza das amostras e das funções que definem a superfície. A seguir são apresentados os métodos de reconstrução incremental de superfícies e reconstrução a partir de pontos desorganizados.

2.2.1 Reconstrução incremental de superfície

A reconstrução incremental de superfície é feita utilizando o algoritmo *Iterative Closest Point* (ICP). Esse algoritmo interage fazendo associação de todos os pontos entre um *frame* e os pontos do *frame* anterior. Com a associação é encontrado uma quantidade de pontos que tiveram uma distorção mínima. Esses pontos são usados para identificar em qual direção e ângulo a câmera ou o objeto foram movimentados. Com essa informação são definidas as coordenadas dos novos pontos adicionando-os na nuvem de pontos. A cada frame a nuvem de pontos vai sendo incrementada (HENRY, 2012, p. 4).

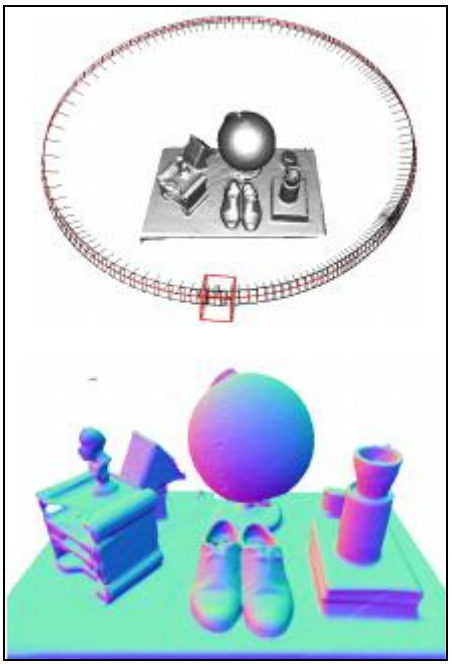
Para fazer a reconstrução incremental de superfícies com o Kinect, o objeto pode ser rotacionado na frente do Kinect ou se pode manter o objeto parado e percorrer o Kinect ao redor. Na Figura 8 é apresentada na parte inferior uma reconstrução incremental de superfície de um objeto que foi percorrido pelo Kinect, na parte superior é apresentado por uma linha vermelha como o Kinect o percorreu. O objeto apresentado na Figura 8 foi percorrido parcialmente, sendo perceptível falhas na reconstrução da superfície (IZADI et al., 2011b, p. 8).



Fonte: Izadi et al. (2011b, p. 8).

Figura 8 - Volta parcial

Na Figura 9 foi percorrida uma volta completa ao redor do mesmo objeto apresentado na Figura 8, com a maior quantidade de ângulos obtidos com a volta completa, a reconstrução da superfície do objeto apresenta melhor qualidade (IZADI et al., 2011b, p. 8).



Fonte: Izadi et al. (2011b, p. 8).

Figura 9 - Volta completa

2.2.2 Reconstrução a partir de pontos desorganizados

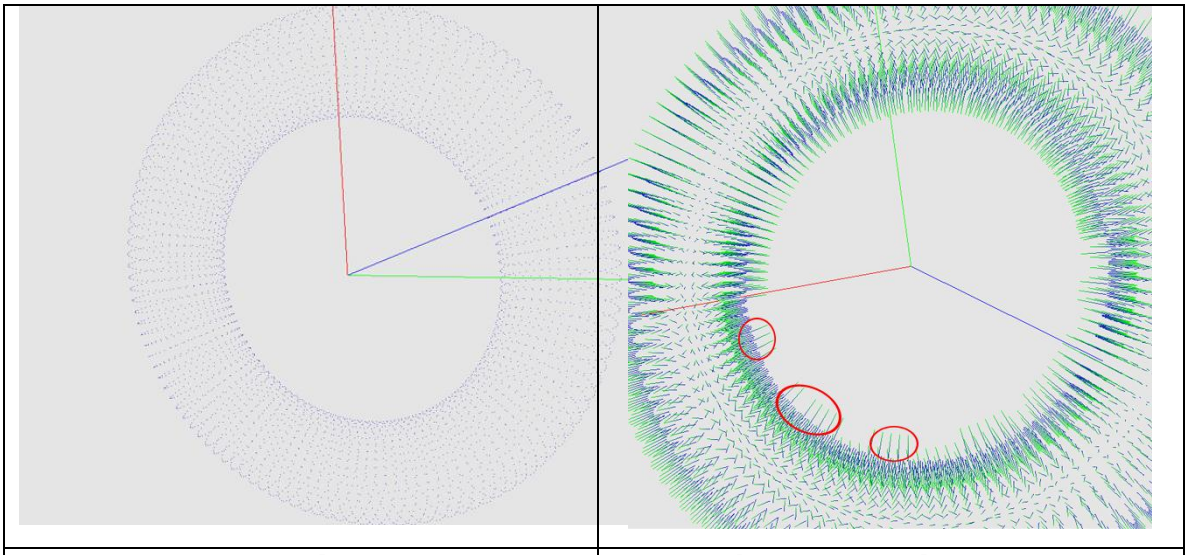
Este método foi apresentado em 1992 por Hugues Hoppe e se tornou referência, pois trouxe inovações no tratamento de nuvem de pontos, Este método pode ser dividido em duas etapas (MOTA, 2007, p. 22-25): calcular a distância geométrica com sinal e o algoritmo *Marching Cubes*.

O cálculo da distância geométrica com sinal é feita através de três etapas: estimação dos planos tangentes para cada ponto da nuvem, orientação dos planos tangentes e cálculo da distância com sinal (MOTA, 2007, p. 23).

Na etapa de estimação dos planos tangentes é definido um plano para cada ponto, esse plano forma um ângulo de 90º entre o ponto e o seu vetor normal, o vetor normal possui o valor que forma um ângulo de 90º com a superfície, cada ponto pode ter até dois vetores normais, um que é direcionado para dentro e outro para fora da superfície. O valor do vetor normal de cada ponto pode ser obtido através do cálculo de produto vetorial entre o ponto analisado e seus vizinhos (MOTA, 2007, p. 23).

Nas Figura 10, Figura 11 e Figura 12 no centro das imagens são exibidas três linhas nas cores verde, vermelho e azul, que representam o Sistema de Referência do Universo (SRU).

Na Figura 10 item (a) é ilustrado uma nuvem de pontos, e na Figura 10 item (b) é apresentado a nuvem de pontos com a direção da normal para cada ponto, sendo representada por linhas verdes, os pontos são representados com a cor azul. Na Figura 10 item (b) há três elipses vermelhas para dar ênfase em algumas normais, mostrando que nem todas estão orientadas.



(a) Nuvem de pontos | (b) Estimação de normais

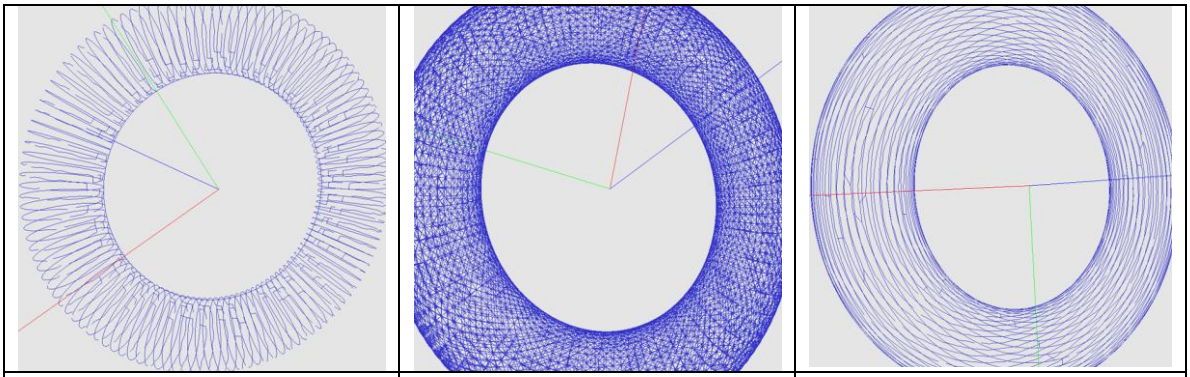
Fonte: Mota (2007, p. 31-32).

Figura 10 - Estimação dos planos tangentes

Na etapa de orientação dos planos tangentes são definidos os vetores normais que estão direcionados para fora da superfície. Para essa tarefa, usa-se abordagem por grafos, sendo necessária a utilização de três desses na seguinte ordem (MOTA, 2007, p. 24):

a) grafo da árvore geradora mínima euclidiana (Figura 11 item a): utilizado para gerar o caminho de menor custo;
b) grafo riemanniano (Figura 11 item b): esse grafo é obtido com adição de varias arestas ao grafo da árvore geradora mínima euclidiana, ligando cada pontos com seus vizinhos;
c) grafo da árvore geradora mínima não euclidiana (Figura 11 item c): utilizado para gerar o caminho com menor custo a partir do grafo riemanniano.

Na Figura 11 é apresentado o processo de abordagem por grafos para a nuvem de pontos da Figura 10 item (a).



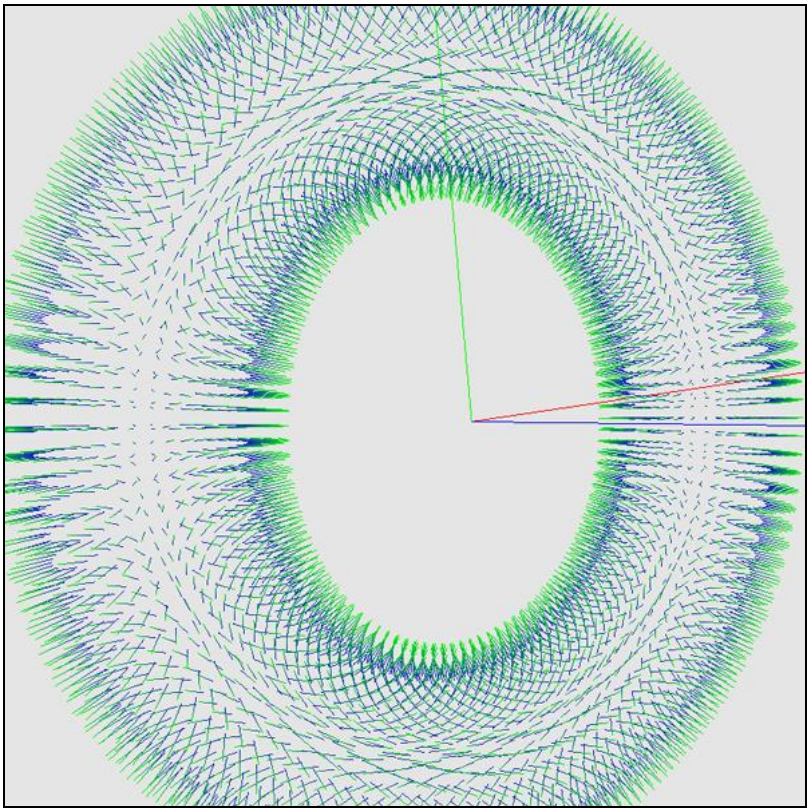
(a) árvore geradora mínima euclidiana | (b) grafo riemanniano | (c) árvore geradora mínima

Fonte: Mota (2007, p. 32-33).

Figura 11 - Abordagem por grafos para orientação dos planos tangentes

A última etapa para orientar as normais é fazer com que o ponto que tenha o maior valor na coordenada z, aponte seu vetor normal para o eixo z. Assim um ponto é garantido que tem seu vetor normal direcionado para fora da superfície, restando apenas propagar essa informação a todos os outros pontos. Isso é feito percorrendo a árvore geradora mínima (MOTA, 2007, p. 24).

Na Figura 12 é apresentada a nuvem de pontos da Figura 10 item (a) com os valores dos vetores normais orientados para cada ponto, os vetores normais são representados com linhas verdes, e os pontos em cor azul.



Fonte: Mota (2007, p. 33).

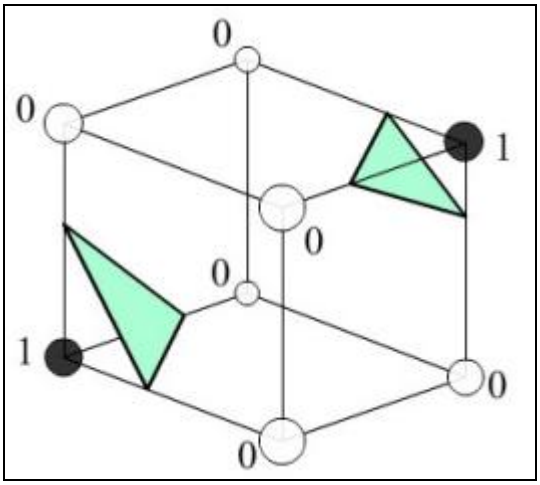
Figura 12 - Vetores normais orientados

Após a orientação dos planos tangentes é feito o cálculo da distância com sinal para obter o conjunto zero $Z(f)$, os pontos dentro do objeto serão definidos com valor 1 e os que estiverem fora terão valor 0 (MOTA, 2007, p. 25).

A etapa da extração do contorno é feito com o algoritmo *Marching Cubes*, utilizado $Z(f)$ para gerar superfície simples (MOTA, 2007, p. 25). A seguir é descrito o algoritmo *Marching Cubes*.

2.2.2.1 Marching Cubes

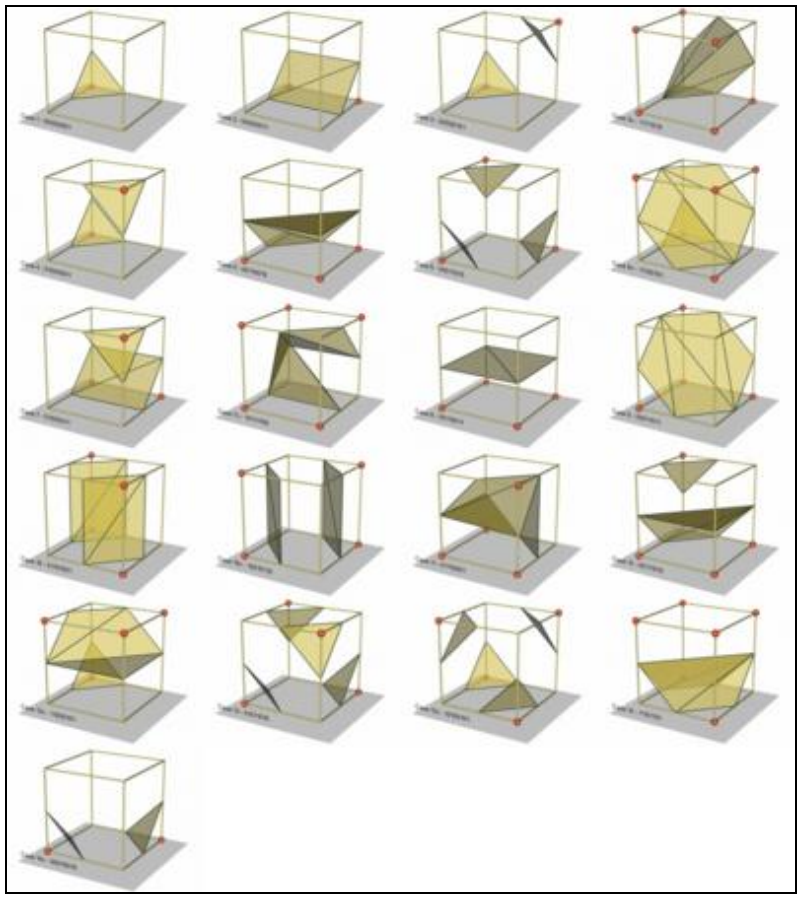
O método aproxima a superfície dividindo o espaço de domínio em uma série de cubos lógicos. O algoritmo realiza a verificação da interseção da superfície com todos os cubos do domínio. Para encontrar esta interseção é dado o valor 1 para um vértice caso seu valor seja maior ou igual ao valor da função da superfície. Estes vértices são considerados dentro da superfície. Vértices com valores menores que o valor da função da superfície recebem valor 0 e são considerados fora da superfície. A reconstrução da superfície é feita percorrendo todos os cubos lógicos do domínio. A cada cubo a superfície intercepta as arestas que possuem um vértice com valor 1 e outro com valor 0, como é demonstrado na Figura 13 onde a superfície é representada pela cor verde (AGUIAR, 2010b, p. 34).



Fonte: Aguiar (2010b, p. 35).

Figura 13 - Construção da superfície através de vértices do cubo

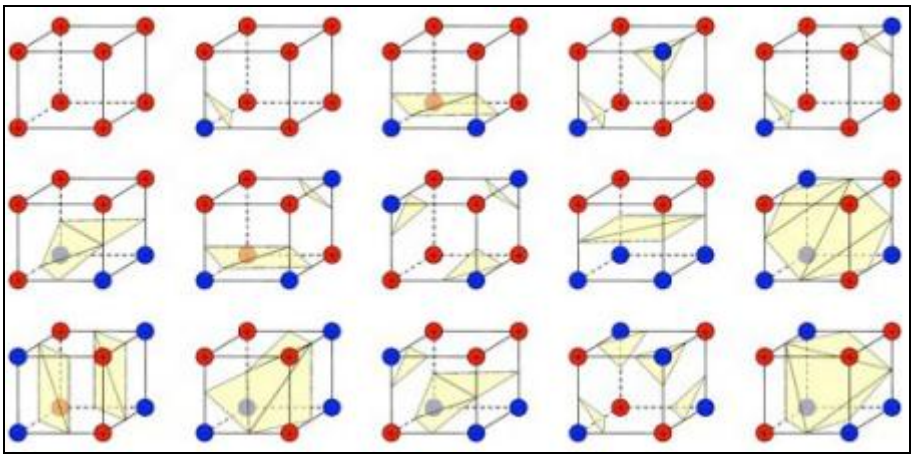
Há oito vértices em cada cubo, sendo que cada vértice pode ter valor 1 ou 0. Sendo assim existe 256 maneiras de uma superfície interceptar as arestas deste cubo. Na Figura 14 são representadas algumas possibilidades da superfície interceptar o cubo. Os vértices com valor 1 são representados por pontos vermelhos e a superfície gerada é representada em tons de amarelo (AGUIAR, 2010b, p. 35).



Fonte: Lorensen (2007).

Figura 14 - Algumas maneiras que a superfície intercepta os cubos lógicos

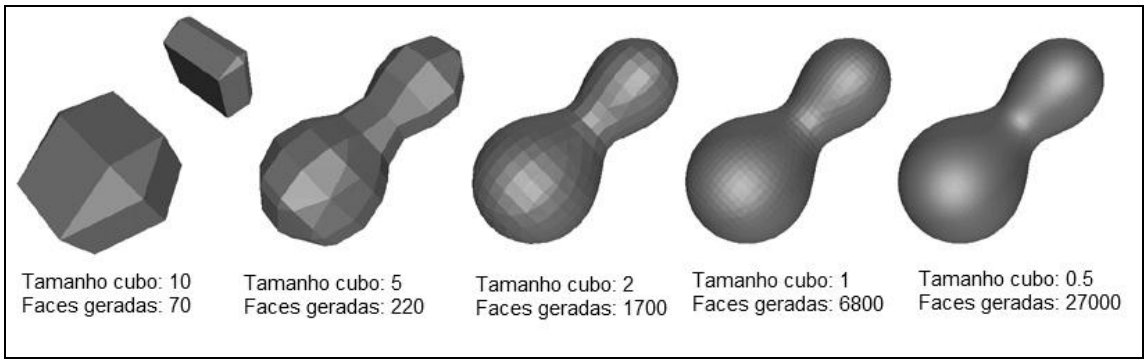
Conforme Dietrich (2008, p. 44) os autores Lorensen e Cline observaram que alguns dos grupos de polaridades eram simétricos. Estas operações mantém a topologia invariante entre os grupos simétricos, e permitem a redução das 256 para 15 casos representativos (DIETRICH, 2008, p. 44). Os casos representativos do *Marching Cubes* são ilustrados na Figura 15, onde os vértices com valores 1 estão representados em azul e a superfície gerada em amarelo.



Fonte: Dietrich (2008, p. 44).

Figura 15 - A triangulação dos 15 casos representativos

A qualidade da superfície gerada pelo algoritmo depende do tamanho que são definidos os cubos. Cubos menores geram uma superfície de melhor qualidade. Na Figura 16 é apresentada a reconstrução de superfície do mesmo objeto utilizando o método de *Marching Cubes*, com cinco tamanhos de cubo (BOURKE, 1994).



Fonte: adaptada de Bourke (1994).

Figura 16 - Qualidade da superfície para diferentes tamanhos de cubo

A redução do tamanho do cubo melhora a qualidade superfície gerada, mas quanto menor o tamanho do cubo, mais processamento e tempo vai ser necessário para gerar a superfície (BOURKE, 1994).

2.3 TECNOLOGIAS PARA DESENVOLVIMENTO DE INTERFACES GRÁFICAS

A seguir são apresentados duas tecnologias para o desenvolvimento de interfaces gráficas *Windows Presentation Foundation* (WPF) e a biblioteca OpenTK.

Conforme Microsoft (2012c) WPF é um *framework* para desenvolvimento de

interfaces gráficas ricas. Oferece um modelo de desenvolvimento que separa a regra de negócio do design da interface, sendo esta última orientada por uma linguagem de programação semelhante ao *eXtensible Markup Language* (XML) o *Extensible Application Markup Language* (XAML).

O núcleo do WPF é um mecanismo de resolução independente e renderização vetorial que é criado para tirar proveito dos modernos hardwares gráficos. WPF contém um conjunto abrangente de recursos de desenvolvimento de aplicativo que incluem,controles, associação de dados, layout, 2-D e 3-D Elementos gráficos, animação, estilos, modelos, documentos, mídias, texto e tipografia (MICROSOFT, 2012c).

OpenTK é uma biblioteca em C#, que permite em programas .Net a utilização das bibliotecas OpenGL para rotinas gráficas, OpenAL para rotinas de áudio e OpenCL para desenvolvimento de programas que utilizam plataformas com vários tipos de processamento (OPENTK, 2008).

Conforme Manssour (2003) OpenGL é uma biblioteca de rotinas gráficas e de modelagem, bidimensional e tridimensional, extremamente portável e rápida. A maior vantagem na sua utilização é a rapidez, uma vez que usa algoritmos cuidadosamente desenvolvidos e otimizados pela Silicon Graphics, Inc., líder mundial em Computação Gráfica e Animação. OpenGL dá suporte a iluminação, colorização, mapeamento de textura, transparência, animação, entre outros efeitos.

2.4 TRABALHOS CORRELATOS

A seguir são apresentados projetos que utilizam o Kinect: KinectFusion (IZADI et al., 2011a), KinectCAD (LÜBKE , 2012), “Kinect SDK Dynamic Time Warping Gesture Recognition” (RHEMYST; RYMIX, 2011) e Técnica de navegação em documentos utilizando Microsoft Kinect (SILVEIRA, 2011).

KinectFusion (IZADI et al., 2011a) é uma aplicação que permite a reconstrução de objetos e ambientes através da captura de dados da câmera de profundidade do Kinect. Essa aplicação consegue gerar modelos tridimensionais e aperfeiçoa-los em tempo real. Durante a execução da captura do modelo, ao movimentar-se o Kinect novos dados sobre o modelo são obtidos, tornando assim possível fazer a reconstrução de todos os lados do objeto e não somente de um lado. Na Figura 17 é apresentado a reconstrução de um objeto que foi

rotacionado na frente do Kinect enquanto ele fazia a reconstrução da superfície, fazendo uma reconstrução incremental desse objeto.

Fonte: Izadi et al. (2011a, p. 3).

Figura 17 – KinectFusion

Como se pode observar na Figura 17 é perceptível a semelhança do objeto real com o objeto reconstruído, mas a reconstrução não é perfeita. É possível notar que os detalhes internos do objeto contém imperfeições na reconstrução das superfícies.

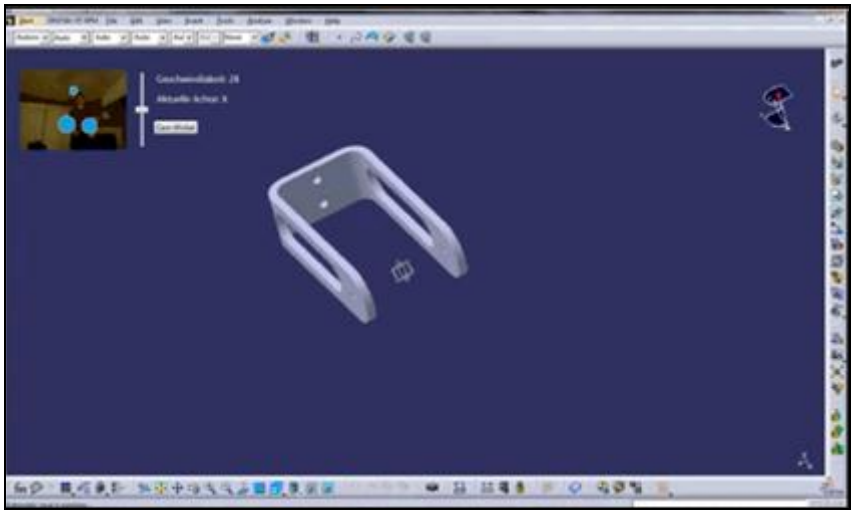
KinectFusion (IZADI et al., 2011a) também simula física no ambiente captado. Na Figura 18 é demonstrado através de bolinhas amarelas a colisão com objetos na cena, como sofás, mesas, chão entre outros.

Fonte: Izadi et al. (2011a, p. 4).

Figura 18 - Simulação de física

O KinectFusion pode também detectar o toque do usuário sobre superfícies detectadas, diferenciando cada dedo. A aplicação também consegue segmentar objetos da cena e fazer o acompanhamento desse objeto.

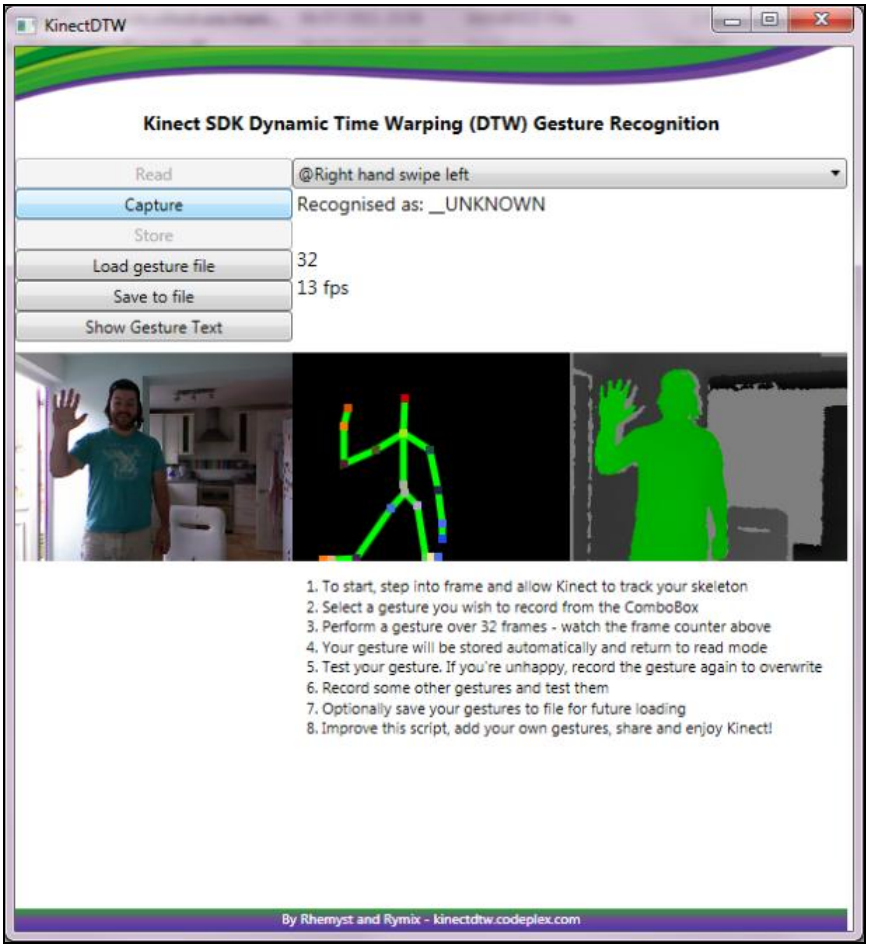
KinectCAD (LÜBKE, 2012) é uma aplicação que permite a manipulação de objetos no sistema *Computer Aided Three-dimensional Interactive Application* (CATIA) (Figura 19) através do esqueleto gerado pelo SDK. Se o usuário fizer algum movimento que esteja predefinido na aplicação, está executará a função correspondente, tal como ampliar, reduzir, rotacionar e movimentar o objeto. Também utiliza a função de reconhecimento de comandos por voz do SDK para alterar o eixo de rotação do objeto.



Fonte: Lübke (2012).

Figura 19 – KinectCAD

"Kinect SDK Dynamic Time Warping Gesture Recognition" (RHEMYST; RYMIX, 2011) é uma aplicação que grava movimentos de uma pessoa para seu posterior reconhecimento (Figura 20). Para gravar os gestos a aplicação utiliza os *joints* que formam o esqueleto e grava as coordenadas x e y desses. Para o posterior reconhecimento do gesto ele faz a comparação do novo movimento gerado com as coordenadas gravadas anteriormente, através de um algoritmo de verificação de distância entre pontos. O usuário, se preferir, pode gravar as coordenadas em um arquivo texto e importar o arquivo posteriormente. Como a aplicação só grava as coordenadas x e y, ela não reconhece movimentos tridimensionais, por exemplo, o aproximar da mão direita em direção ao Kinect.



Fonte: Rhemyst e Rymix (2011).

Figura 20 - Kinect SDK *dynamic time warping gesture recognition*

"Kinect SDK Dynamic Time Warping Gesture Recognition" (RHEMYST; RYMIX, 2011) não apresenta testes que demonstrem a eficiência da aplicação, mas utilizando a aplicação percebe-se que ela cumpre com o que foi proposto.

A técnica de navegação em documentos utilizando Microsoft Kinect (SILVEIRA, 2011) é uma técnica para calcular o ângulo formado pelo segmento do antebraço com seu respectivo braço e com este valor executa o comando próximo *slide* ou anterior desde que o antebraço esteja na região predefinida para cada comando. Para evitar execuções indesejadas, foi definido um tempo de 0,75 segundos, no qual o antebraço deve permanecer dentro da região definida para executar a ação. Para executar a ação repetidamente, deve-se manter o antebraço dentro da região por 1,5 segundos e a ação será repetida a cada intervalo de 1,5 segundos. Para o usuário poder gesticular mais livremente, foi calculada uma altura máxima do cotovelo com base nos *joints* da cabeça e do centro do quadril, reconhecendo os comandos apenas quando o cotovelo estiver a menos de 20% desta altura.

Conforme Silveira (2011), a técnica de navegação em documentos utilizando Microsoft Kinect foi avaliada com nove usuários, que após a utilização responderam a um

questionário, obtendo os seguintes resultados:

a) em média os usuários dominavam os comandos em dez minutos;
b) ocorria perda de desempenho no reconhecimento de comandos quando tinha mais de uma pessoa na frente do Kinect;
c) todos os usuários acreditam que o método como este deveria substituir o atualmente utilizado;
d) a maioria dos usuários considerou que o sistema ainda possui algumas inconsistências;
e) a maioria dos usuários informou que usaria este sistema com frequência.

O Quadro 2 mostra de forma resumida as principais características dessas aplicações tendo como base critérios considerados importantes, extraídos a partir dos conceitos descritos no decorrer desta seção.

características / trabalhos relacionados	IZADI et al. (2011)	LÜBKE (2012)	RHEMYST; RYMIX (2011)	SILVEIRA (2011)
utiliza *stream* de vídeo	X	X	X	
utiliza *stream* de profundidade	X			
utiliza *stream* de áudio		X		
utiliza *skeletal tracking*		X	X	X
faz reconstrução de superfícies	X			
faz reconstrução incremental	X			
simula física no ambiente tridimensional gerado	X			

Quadro 2 - Características dos trabalhos relacionados

As informações foram dispostas em colunas, representando na vertical os trabalhos analisados e as linhas apresentam as características de cada sistema, indicando semelhanças e/ou diferenças.

A partir do Quadro 2, pode-se observar que apenas a aplicação KinectFusion (IZADI et al., 2011a) faz a reconstrução de superfícies, os outros trabalhos utilizam o Kinect para outras funcionalidades, utilizando principalmente o *skeletal tracking*.

3 DESENVOLVIMENTO

Nesta seção são apresentados os requisitos do trabalho proposto na seção 3.1, a especificação na seção 3.2, o desenvolvimento do algoritmo na seção 3.3 e os resultados e discussões na seção 3.4.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

A aplicação de modelagem tridimensional utilizando Kinect deverá:

a) fazer o mapeamento tridimensional de ambientes ou objetos estáticos dentro do limite de visibilidade do Kinect (Requisito Funcional – RF);
a) disponibilizar uma interface para apresentar o mapeamento tridimensional (RF);
b) ser implementado utilizando a linguagem C# (Requisito Não-Funcional – RNF);
c) utilizar o ambiente Visual Studio 2010 para o desenvolvimento (RNF);
d) utilizar o equipamento Kinect (RNF);
e) utilizar o Kinect *for* Windows SDK (RNF);
f) executar em Windows 7 64 bits (RNF).

3.2 ESPECIFICAÇÃO

Na sequência é apresentada a especificação da aplicação de modelagem tridimensional, que foi modelada na ferramenta Enterprise Architect. A aplicação foi desenvolvida seguindo a análise orientada a objetos, utilizando a notação *Unified Modeling Language* (UML) para a criação dos diagramas de casos de uso e classes.

3.2.1 Diagrama de casos de uso

A Figura 21 exibe o diagrama de caso de uso da aplicação.

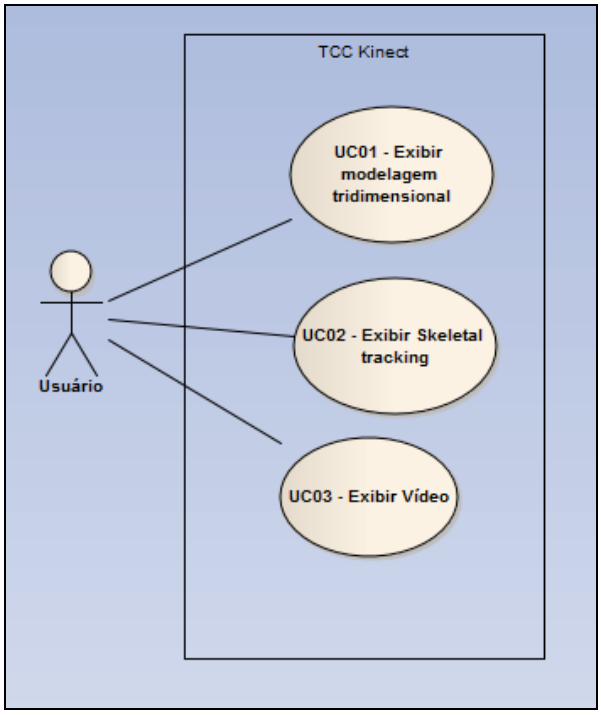


Figura 21 - Diagrama de casos de uso

Os casos de uso da aplicação são descritos a seguir:

a) `UC01` – exibir modelagem tridimensional: captura dados de profundidade do Kinect, cria uma modelagem tridimensional com os dados capturados e exibe para o usuário, durante a exibição o usuário pode navegar no ambiente tridimensional gerado;

b) `UC02` – exibir *skeletal tracking*: através dos *joints* detectados pelo Kinect e gerado e apresentado para o usuário o *skeletal tracking*;

c) `UC03` – exibir vídeo: exibe o vídeo detectado pelo Kinect para o usuário.

No Quadro 3 é detalhado o caso de uso `UC01`.

UC01 – Exibir modelagem tridimensional	
Pré-condições	Kinect deve estar conectado ao computador.
Cenário principal	01) O usuário inicia o aplicativo. 02) A aplicação procura um Kinect conectado. 03) A aplicação inicia o Kinect. 04) A aplicação inicia uma *thread* de `OpenTKThread`. 05) A aplicação inicia o *stream* de profundidade do Kinect. 06) A aplicação captura os dados do *stream* de profundidade e gera um conjunto de triângulos. 07) A aplicação exibe a malha de triângulos. 08) O usuário pode navegar no ambiente tridimensional através das teclas A, S, D, W, R e F
Exceção 01	No passo 02, a aplicação pode não encontrar um Kinect: 02.1) A aplicação exibe uma mensagem indicando que ocorreu um erro e encerra a aplicação.

Quadro 3 - Caso de uso 01

No Quadro 4 é detalhado o caso de uso UC02.

UC02 – Exibir *skeletal tracking*	
Pré-condições	Kinect deve estar conectado ao computador.
Cenário principal	01) O usuário inicia o aplicativo. 02) A aplicação procura um Kinect conectado. 03) A aplicação inicia o *stream* de *skeletal tracking*. 04) A aplicação inicia o Kinect. 05) A aplicação captura os *joints* do *stream* de *skeletal tracking*. 06) A aplicação exibe para cada dois *joint* vizinhos uma reta.
Exceção 01	No passo 02, a aplicação pode não encontrar um Kinect: 02.1) A aplicação exibe uma mensagem indicando que ocorreu um erro e encerra a aplicação.

Quadro 4 - Caso de uso 02

No Quadro 5 é detalhado o caso de uso UC03.

UC03 – Exibir modelagem tridimensional	
Pré-condições	Kinect deve estar conectado ao computador.
Cenário principal	01) O usuário inicia o aplicativo. 02) A aplicação procura um Kinect conectado. 03) A aplicação inicia o *stream* de cores do Kinect. 04) A aplicação inicia o Kinect. 05) A aplicação exibe os dados do *stream* de cores.
Exceção 01	No passo 02, a aplicação pode não encontrar um Kinect: 02.1) A aplicação cliente exibe uma mensagem indicando que ocorreu um erro e encerra a aplicação.

Quadro 5 - Caso de uso 03

3.2.2 Diagrama de classes

A Figura 22 apresenta as quatro classes utilizadas no desenvolvimento da aplicação.

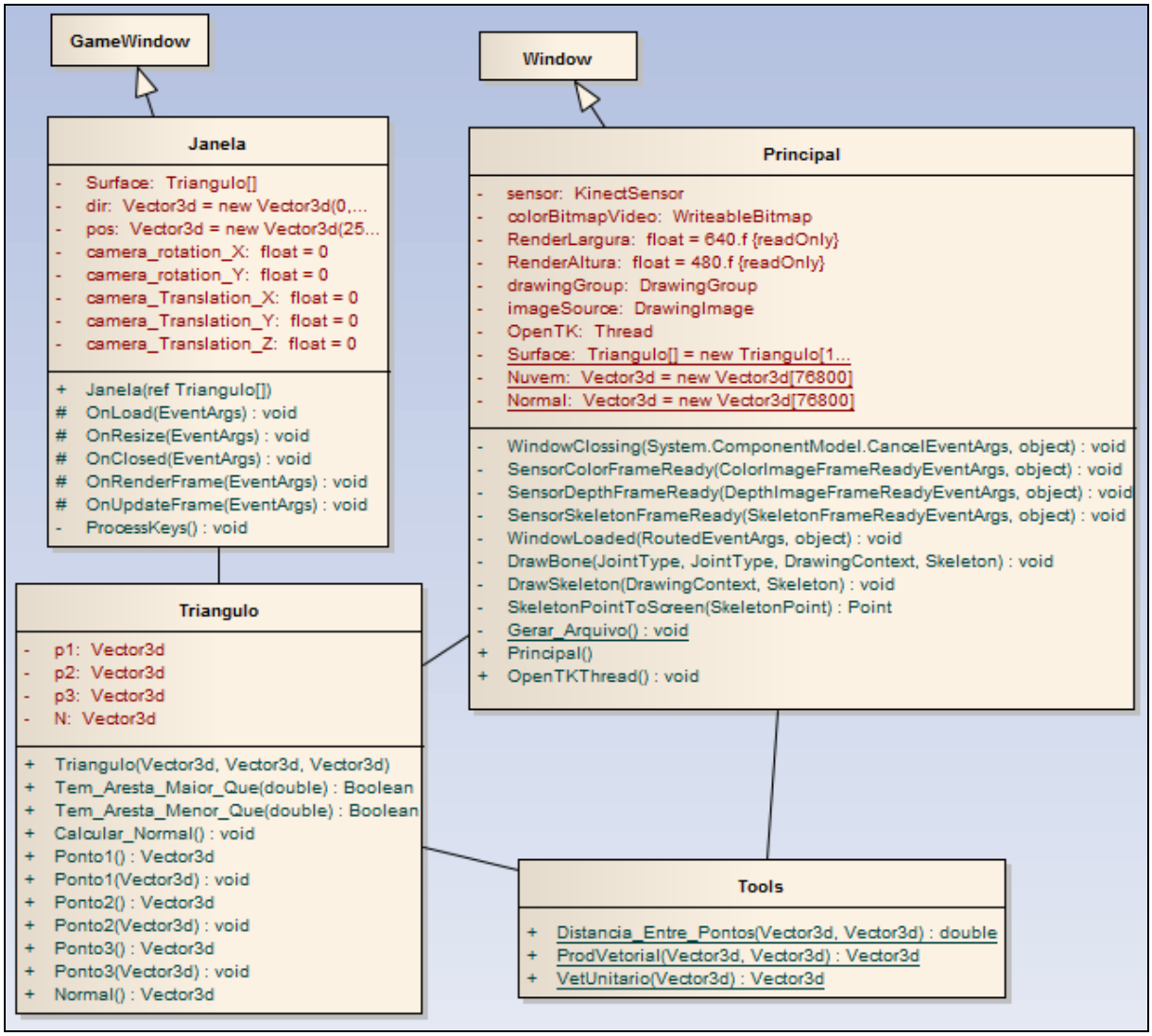


Figura 22 - Diagrama de classes

A seguir são detalhadas as quatro classes apresentadas no diagrama de classes (Figura 22).

A classe `Principal` é uma janela *Windows* WPF que é executada pelo ponto de inicialização de aplicações WPF o `App.xaml`. Quando executada a classe `Principal` ela executa o evento `WindowLoaded` que primeiramente vai procurar um aparelho Kinect conectado ao computador e vai carregar a variável `sensor` com este Kinect. Caso não encontrado, a aplicação apresentará uma mensagem de erro e será fechada. Caso encontrado um Kinect ele ativa nesse sensor os *stream* de vídeo e *skeletal tracking* e logo após é iniciado o `sensor` e criado uma *thread* do método `OpenTKThread`. No método `OpenTKThread` a variável `sensor` ativa o *stream* de profundidade e cria uma instância da classe `Janela` passando por parâmetro a `Surface` por referência.

`Janela` é uma `GameWindow` uma classe da biblioteca OpenTK que contém cinco eventos:

a) `OnLoad`: inicializa os componentes da `GameWindow`;
b) `OnResize`: evento que ocorre quando a `GameWindow` é redimensionada, utilizado para redimensionar o espaço tridimensional dentro da `GameWindow`;
c) `OnClosed`: encerra os componentes da `GameWindow`;
d) `OnRenderFrame`: principal evento da classe Janela, onde é utilizado a `Surface` que é um *array* de `Trigangulo`. Para desenhar esses triângulos na janela através de comandos do OpenTK, este evento ocorre cada vez que tem uma atualização de *frame*;
e) `OnUpdateFrame`: evento que chama o método `ProcessKeys` utilizado para controlar a alteração do ponto de visão dentro do ambiente tridimensional na `GameWindow`.

A classe `Triangulo` tem as coordenadas dos três pontos que formam um triângulo num espaço tridimensional e o valor do vetor normal desse triângulo.

A classe `Tools` contém alguns métodos para auxiliar a manipulação de dados em ambientes tridimensionais.

3.3 IMPLEMENTAÇÃO

Esta seção descreve as técnicas e ferramentas utilizadas durante a implementação da aplicação, bem como a implementação de cada etapa e sua operacionalidade.

3.3.1 Técnicas e ferramentas utilizadas

A aplicação foi desenvolvida utilizando a linguagem de programação C# na IDE Microsoft Visual Studio 2010, utilizando as seguintes tecnologias:

a) um aparelho Kinect para obtenção dos dados do ambiente real;
b) Kinect *for* Windows SDK: *Kit* para desenvolvimento de aplicações utilizando Kinect;
c) OpenTK: biblioteca gratuita para utilização de OpenGL, OpenAL e OpenCL na linguagem de programação C#;

d) WPF: framework para desenvolvimento de interfaces gráficas ricas. Oferece um modelo de desenvolvimento que separa claramente a regra de negócio do design da interface, sendo esta última orientada por uma linguagem de programação semelhante ao XML.

3.3.2 Implementação da aplicação

Nesta seção são apresentadas as etapas de inicialização do Kinect e reconstrução de superfícies, para a implementação das principais classes e funções desenvolvidas neste trabalho.

3.3.2.1 Inicialização do Kinect

O primeiro passo é detectar um Kinect conectado ao computador. No Quadro 6, na linha 01 é apresentado o código para percorre a coleção `KinectSensor.KinectSensors` carregando o sensor em potencial na variável `potentialSensor`. Em seguida é verificado se o `potentialSensor` está conectado (Quadro 6 linha 03). Caso o `potentialSensor` esteja conectado ele é carregado na variável `sensor` (Quadro 6 linha 05) e encerrada a procura por sensores (Quadro 6 linha 06).

```
01 foreach (var potentialSensor in KinectSensor.KinectSensors)
02 {
03         if (potentialSensor.Status == KinectStatus.Connected)
04         {
05             this.sensor = potentialSensor;
06             break;
07         }
08 }
```

Quadro 6 - Localizar Kinect conectado ao computador

Com o Kinect detectado são ativados os *stream* de *skeleton* (`SkeletonStream`) e vídeo (`ColorStream`), e associados os eventos de cada *stream* aos métodos que trataram cada evento (Quadro 7 linha 01 até 07). Em seguida é iniciado o Kinect (Quadro 7 linha 14).

```
01 // Skeleton
02 this.sensor.SkeletonStream.Enable();
03 this.sensor.SkeletonFrameReady += this.SensorSkeletonFrameReady;
04
05 // Video
06 this.sensor.ColorStream.Enable(
                              ColorImageFormat.RgbResolution640x480Fps30);
07 this.sensor.ColorFrameReady += this.SensorColorFrameReady;
08 ...
09 // Depth
10 this.sensor.DepthStream.Enable(
                              DepthImageFormat.Resolution320x240Fps30);
11 this.sensor.DepthFrameReady += this.SensorDepthFrameReady;
12
13 ...
14 this.sensor.Start();
```

Quadro 7 - Ativação de *stream* e criação de eventos

Em seguida é criada uma *thread* do método `OpenTKThread` (Quadro 8), onde é criada uma instância de `Janela`, passando `Surface` que é um *array* de `Triangulo` por referência. Também será ativado o *stream* de profundidade e associado o evento gerado por esse (`DepthStream`) ao método `SensorDepthFrameReady` (Quadro 7 linhas 10 e 11). A criação da *thread* é necessária para criar uma instância da classe Janela em um processo diferente, para permitir a execução simultânea das janelas.

```
01 OpenTK = new Thread(OpenTKThread);
02
03 OpenTK.Start();
```

Quadro 8 - Criação da *thread*

No momento que são ativados os *streams* de profundidade e vídeo, são escolhidas as resoluções e a quantidade de FPS para cada *stream* (Quadro 7). Neste trabalho, foi utilizada a resolução de 640x480 e 30 FPS e para o *stream* de profundidade a resolução de 320x240 e 30 FPS.

3.3.2.2 Reconstrução de superfícies

Para criar a reconstrução de superfícies, primeiro é necessário obter as coordenadas do ambiente real. Foi utilizado o projetor infravermelho e a câmera infravermelha (seção 2.1.2) para obter essas informações.

A cada *frame* capturado pelo Kinect no *stream* de profundidade (seção 3.3.2.1) é acionado o método `SensorDepthFrameReady` (Quadro 9), onde é capturado um `DepthImageFrame` (Quadro 9 linha 03) que é copiado para um *array* de `short` (Quadro 9 linha 07). Posteriormente esse *array* é percorrido obtendo-se os valores das coordenadas

tridimensionais e carregando-as no *array* `Nuvem` (Quadro 9 linha 10 até 24). Enquanto está sendo percorrido *array*, deve-se retirar os três bits que contém a informação de qual pessoa pertence esse *pixel* do valor capturado do `DepthImageFrame`, para isso é feito um *shift* (seção 2.1.3) para direita (Quadro 9 linha 13) e fazer a calibração (ver seção 2.1.3) dos valores das coordenadas (Quadro 9 linha 15 até 21).

```
01 private void SensorDepthFrameReady(object sender,
                        DepthImageFrameReadyEventArgs e)
02  {
03      DepthImageFrame imageFrame = e.OpenDepthImageFrame();
04      if (imageFrame != null)
05      {
06           short[] pixelData = new
                                 short[imageFrame.PixelDataLength];
07           imageFrame.CopyPixelDataTo(pixelData);
08           double temp = 0;
09           int i = 0;
10           for (int y = 0; y < 240; y++)
11                for (int x = 0; x < 320; x++)
12                {
13                     temp = ((ushort)pixelData[x + y * 320]) >> 3;
14
15                     double fx_d = 1.0 / 5.9421434211923247e+02;
16                     double fy_d = 1.0 / 5.9104053696870778e+02;
17                     double cx_d = 3.3930780975300314e+02;
18                     double cy_d = 2.4273913761751615e+02;
19
20                     Nuvem[i].X = (float)((x - cx_d) * temp / fx_d);
21                     Nuvem[i].Y = (float)((y - cy_d) * temp / fy_d);
22                     Nuvem[i].Z = (float)temp;
23
24                     i++;
25                }
```

Quadro 9 - Obter coordenadas tridimensionais

Depois de carregado o *array* `Nuvem` é iniciado o processo para criação de uma coleção de triângulos (`Surface`) que vai formar a superfície tridimensional.

Em ambientes tridimensionais triângulos são os objetos que sempre vão formar uma superfície plana ao ligar suas três coordenadas. Ligando vários triângulos é possível formar outras formas complexas. Assim neste trabalho foi utilizada uma malha de triângulos para formar a superfície dos objetos.

Para fazer a reconstrução da superfície foi desenvolvido um método baseado na premissa que a resolução utilizada na aplicação para captura de profundidade é de 320x240. A partir disso é gerado um triângulo retângulo a cada três pixels mais próximos (Quadro 10). Na Figura 23 é exemplificado o método de reconstrução de superfície ampliando uma área da resolução de 320x240, onde os pixels são representados por esferas azuis e os triângulos são representados na cor vermelha. Um dos triângulos da Figura 23 não foi preenchido sua área com a cor vermelha, para representar a ligação dos três pixels mais próximos. Para cada

vértice dos triângulos é atribuída uma coordenada tridimensional que está no *array* `Nuvem`. Os triângulos então são armazenados no *array* `Surface` como é apresentado na linha 08 do Quadro 10.

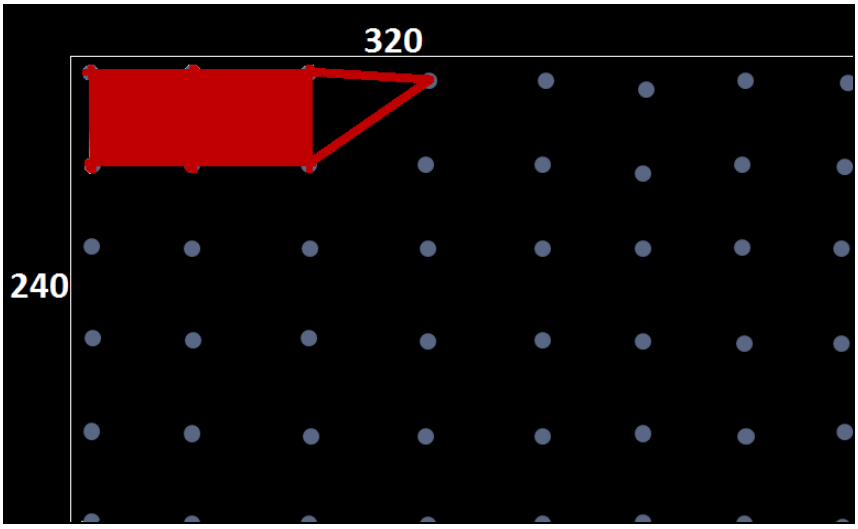


Figura 23 - Método de reconstrução de superfície

Para não gerar erros são eliminados triângulos que tenham um vértice com a coordenada de distância maior ou igual a 8191 e igual a zero. Estas situações caracterizam uma situação onde a distância é desconhecida (Quadro 10 linha 06).

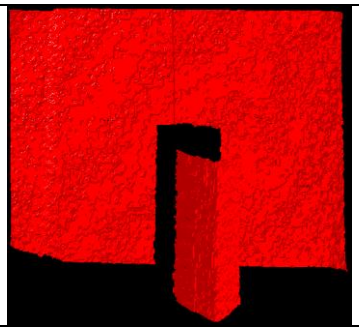
Para poder gerar superfícies com buracos, são eliminados triângulos que possuírem uma de suas arestas maiores que 250 (Quadro 10 linhas 09 e 10).

```
01 int qtdTriangulos = 0;
02 for (int x = 0; x < Nuvem.Length - 321; x++)
03 {
04    if (x % 320 != 0)
05       {
06         if (Nuvem[x].Z > 0 & Nuvem[x+1].Z > 0 & Nuvem[x+320].Z > 0 &
              Nuvem[x].Z < 8191 & Nuvem[x+1].Z < 8191 & Nuvem[x+320].Z < 8191)
07         {
08             Surface[qtdTriangulos]= new Triangulo(Nuvem[x], Nuvem[x+320],
                       Nuvem[x+1]);
09             if (Surface[qtdTriangulos].Tem_Aresta_Maior_Que(250))
10                Surface[qtdTriangulos] = null;
11          }
12          else
13             Surface[qtdTriangulos] = null;
14          qtdTriangulos++;
```

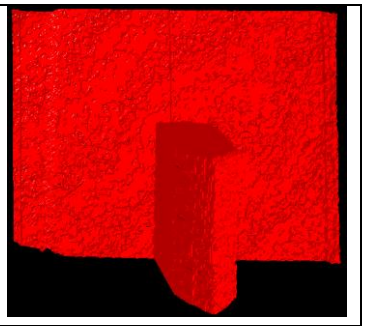
Quadro 10 - Gerando superfície

Gerar buracos na superfície é necessário para que toda a superfície não fique totalmente conectada. Na Figura 24 é apresentada a reconstrução com buracos (Figura 24 item b) e sem buracos (Figura 24 item c) de uma caixa próxima a uma parede (Figura 24 item a).

(a) Imagem real



(b) Reconstrução com buracos



(c) Reconstrução sem buracos

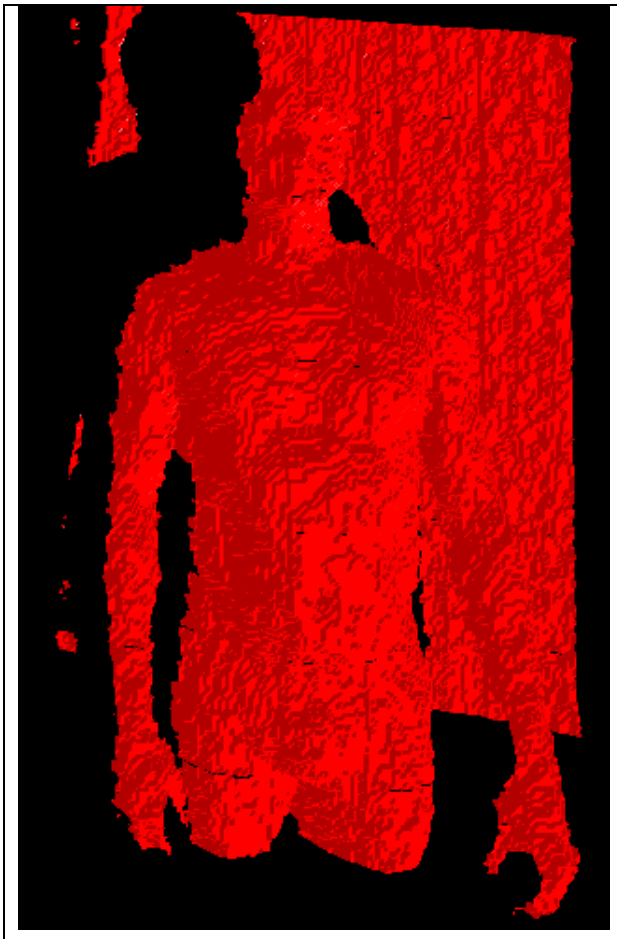
Figura 24 - Reconstrução com e sem buracos

A superfície carregada em `Surface` é apresentada na janela `Janela` através do evento `OnRenderFrame`, para isso, primeiramente é limpado o conteúdo que está sendo apresentado (Quadro 11 linhas 01 e 02) e carregada a matriz de visualização do ambiente tridimensional que contém a posição da câmera (Quadro 11 linhas de 04 até 07).

```
01 GL.Clear(ClearBufferMask.ColorBufferBit | ClearBufferMask.DepthBufferBit);
02 GL.ClearColor(0.0f, 0.0f, 0.0f, 1.0f);
03
04 Matrix4d modelview = Matrix4d.LookAt(pos.X, pos.Y, pos.Z, pos.X + dir.X, pos.Y +
                        dir.Y, pos.Z + dir.Z + 1000, 0, 1.0, 0.0);
05
06 GL.MatrixMode(MatrixMode.Modelview);
07 GL.LoadMatrix(ref modelview);
```

Quadro 11 - Limpar tela e criar matriz de visualização

Na aplicação a superfície é gerada no ambiente tridimensional com a cor vermelha. Para que a imagem exibida para o usuário não seja manchas vermelhas como apresentada na Figura 25 item (b), é adicionada uma fonte de luz ao ambiente tridimensional. Essa fonte de luz serve para alterar o tom da cor da superfície apresentada. Assim como uma luz em um ambiente real, ela projeta a sombra dos objetos, e altera o tom desses de acordo com a luminosidade que esse recebe, criando uma percepção tridimensional para o usuário. Na Figura 25 é apresentado um ambiente tridimensional com luz (Figura 25 item a), e esse ambiente sem luz (Figura 25 item b).



(a) Ambiente tridimensional com luz

(b) Ambiente tridimensional sem luz

Figura 25 - comparação de ambiente virtual com e sem luz

A luz é criada no ambiente tridimensional como apresentado no Quadro 12 nas linhas 02 até 16.

Também é definido o ângulo de visão da câmera no ambiente tridimensional (Quadro 12 linhas 19 até 21), esse ângulo de visão é o ângulo que o usuário vai ter do ambiente tridimensional.

```
01 // Luz
02 float[] mat_specular = { 1.0f, 1.0f, 1.0f, 1.0f };
03 float[] mat_shininess = { 20.0f };
04 float[] light_position = { 1.0f, 1.0f, 1.0f};
05 float[] light_ambient = { 0.5f, 0.5f, 0.5f, 1.0f };
06
07 GL.Material(MaterialFace.Front, MaterialParameter.Specular, mat_specular);
08 GL.Material(MaterialFace.Front, MaterialParameter.Shininess, mat_shininess);
09
10 GL.Light(LightName.Light0, LightParameter.Position, light_position);
11 GL.Light(LightName.Light0, LightParameter.Ambient, light_ambient);
12 GL.Light(LightName.Light0, LightParameter.Diffuse, mat_specular);
13
14 GL.Enable(EnableCap.Lighting);
15 GL.Enable(EnableCap.Light0);
16 GL.Enable(EnableCap.ColorMaterial);
17
18 // Câmera
19 GL.Translate(camera_Translation_X, camera_Translation_Y, camera_Translation_Z);
20 GL.Rotate(camera_rotation_X, 0, 1, 0);
21 GL.Rotate(camera_rotation_Y, 1, 0, 0);
```

Quadro 12 - Adicionar luz ao ambiente e definir angulo de visão da câmera

Depois é percorrido o *array* `Surface` e desenhado cada triângulo para sua visualização tridimensional (Quadro 13).

```
01 GL.Begin(BeginMode.Triangles);
02 foreach (Triangulo T in Surface)
03 {
04    if (T != null)
05    {
06       GL.Normal3(T.Normal);
07       GL.Color4(Color.Red); GL.Vertex3(T.Ponto1);
08       GL.Color4(Color.Red); GL.Vertex3(T.Ponto2);
09       GL.Color4(Color.Red); GL.Vertex3(T.Ponto3);
10    }
11 }
12 GL.End();
```

Quadro 13 - Desenhando superfície

Para a correta iluminação dos triângulos desenhados, no momento que é criado uma instância de `Triangulo` é calculado seu vetor normal e informado no momento que o triângulo é desenhado (Quadro 13 linha 06).

3.3.3 Operacionalidade da implementação

Para utilizar a aplicação o usuário deve iniciá-la com o Kinect devidamente conectado ao computador. A aplicação vai detectar o Kinect e iniciar a captura de dados e apresentar a reconstrução da superfície, *skeletal tracking* e o vídeo obtido pela câmera do Kinect, caso nenhum Kinect for detectado a aplicação será encerrada.

A aplicação possui duas janelas, uma onde é apresentado o *skeletal tracking* e o vídeo obtido pela câmera do Kinect (Figura 26), e outra onde é apresentada a reconstrução de superfície (Figura 27).

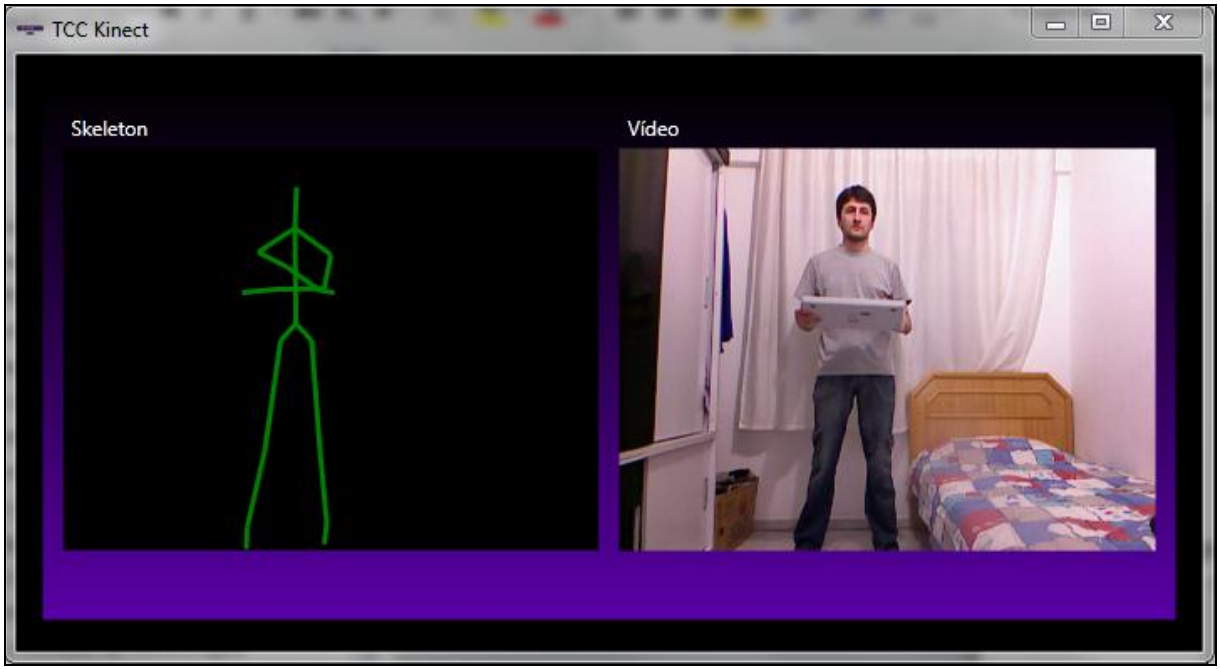


Figura 26 - Janela de apresentação de s*keleton* e vídeo

As duas janelas trabalham em paralelo, mas independentes uma da outra, sendo as duas iniciadas simultaneamente ao iniciar a aplicação.

A aplicação possui essas duas janelas, porque são utilizadas duas técnicas diferentes para apresentação para o usuário. Na janela onde é apresentado o *skeletal tracking* e o vídeo (Figura 26) trabalha-se com WPF e a janela onde é apresentada a reconstrução de superfície (Figura 27) trabalha-se com OpenTK (seção 2.3).

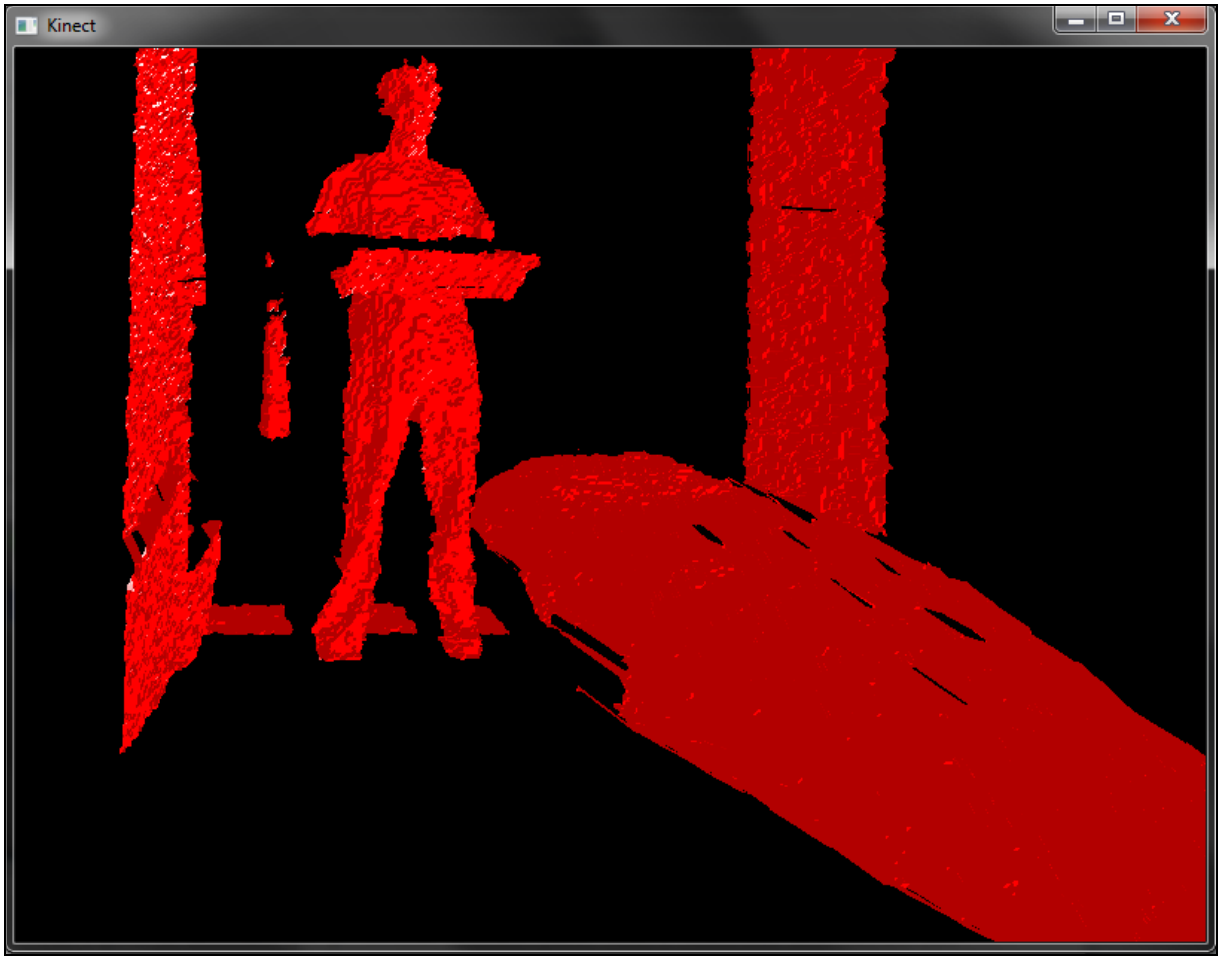


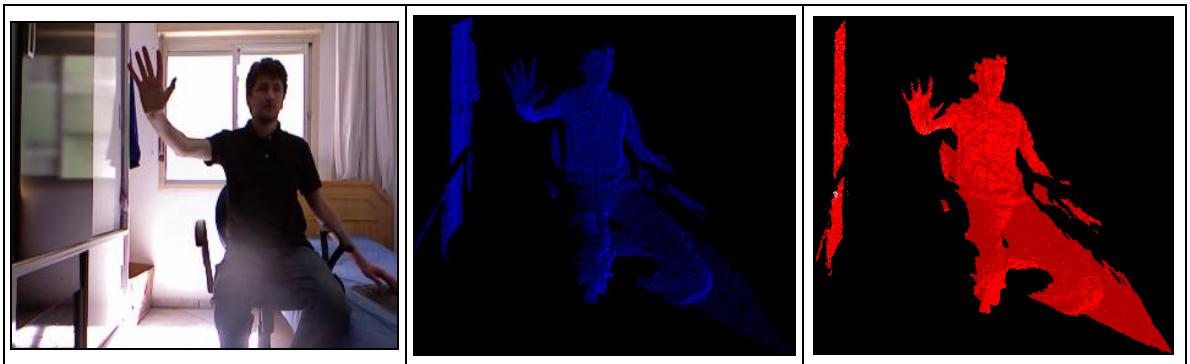
Figura 27 - Janela de apresentação da reconstrução da superfície

Na janela onde é apresentada a reconstrução de superfície é possível navegar no ambiente tridimensional, através das teclas: “a” navega no eixo x para a esquerda, “d” navega no eixo x para a direita, “f” navega no eixo y para baixo, “r” navega no eixo y para cima, “s” navega no eixo z para trás, “w” navega no eixo z para frente.

3.4 RESULTADOS E DISCUSSÃO

O objetivo dos experimentos é verificar se a reconstrução de superfície gerada pela aplicação tem semelhança com sua respectiva imagem real e nuvem de pontos.

Na Figura 28 são apresentadas as etapas do processo de reconstrução de superfícies: imagem real (Figura 28 item a), a nuvem de pontos (Figura 28 item b) obtida através do Kinect e a superfície reconstruída (Figura 28 item c) de uma cena.

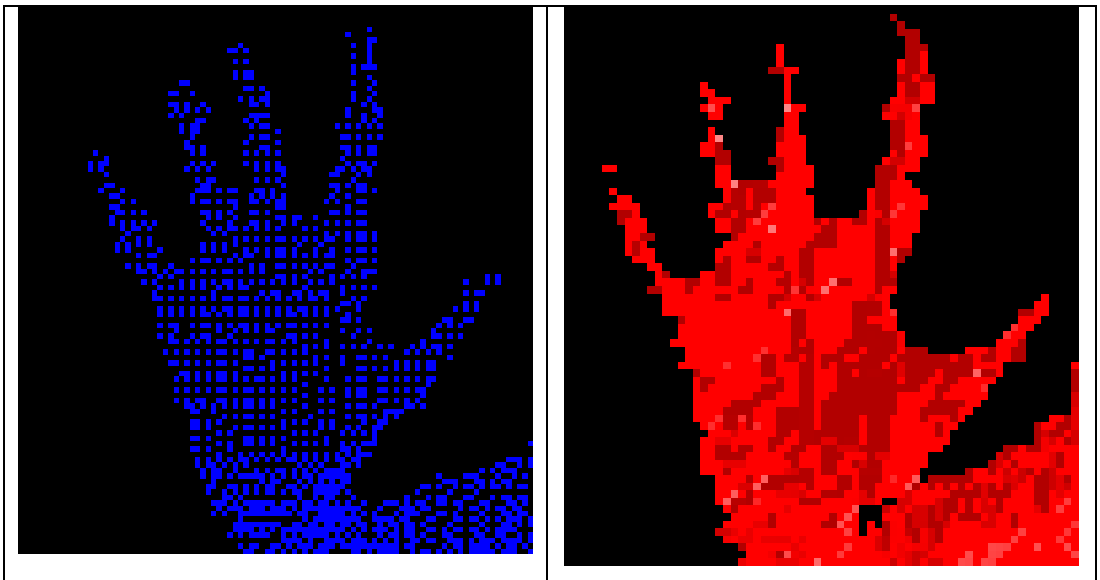


(a) Imagem real	(b) Nuvem de pontos	(c) Superfície reconstruída

Figura 28 - Etapas do processo de reconstrução de superfície

Analisando as etapas apresentadas na Figura 28 percebe-se que a superfície reconstruída (Figura 28 item c) é semelhante a nuvem de pontos (Figura 28 item b). Também é possível notar imperfeições na superfície gerada. Na Figura 29 é ampliada a imagem apresentada na Figura 28 na região da mão, apresentando imperfeições na superfície gerada nos dedos, na nuvem de pontos (Figura 29 item a) há uma quantidade de pontos suficiente para gerar uma reconstrução de superfícies melhor do que aquela apresentada na Figura 29 item (b).

A falta de cores na superfície reconstruída dificulta o entendimento de onde os objetos se delimitam, como pode ser percebido comparando-se a Figura 28 item (c) com a Figura 28 item (a) onde a superfície reconstruída da cama se mistura com a perna do usuário.

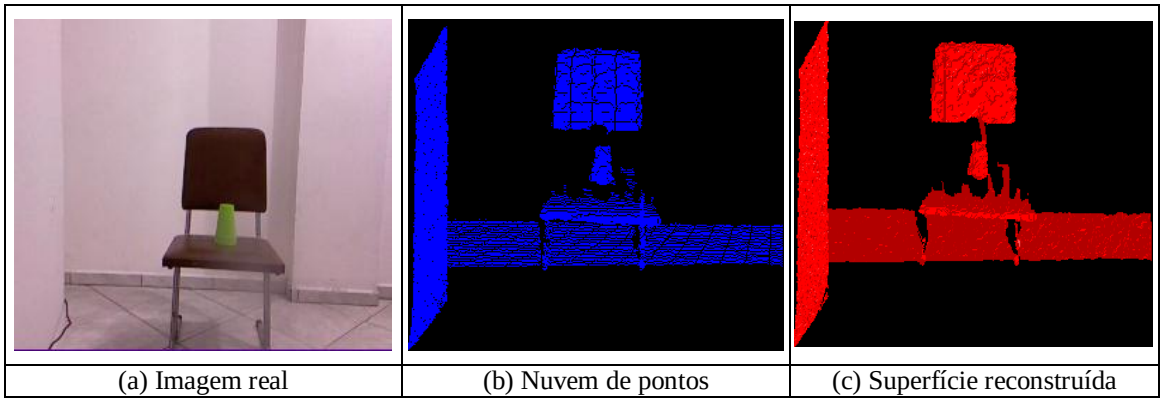


(a) Nuvem de pontos	(b) Superfície reconstruída

Figura 29 - imperfeições na superfície reconstruída

Na Figura 30 é apresentada a reconstrução de uma cadeira com um copo. Analisando essa figura percebe-se que a aplicação não conseguiu captar todos os pontos do assento (Figura 30 item b) da cadeira, causando assim uma reconstrução parcial do assento (Figura 30

item c).

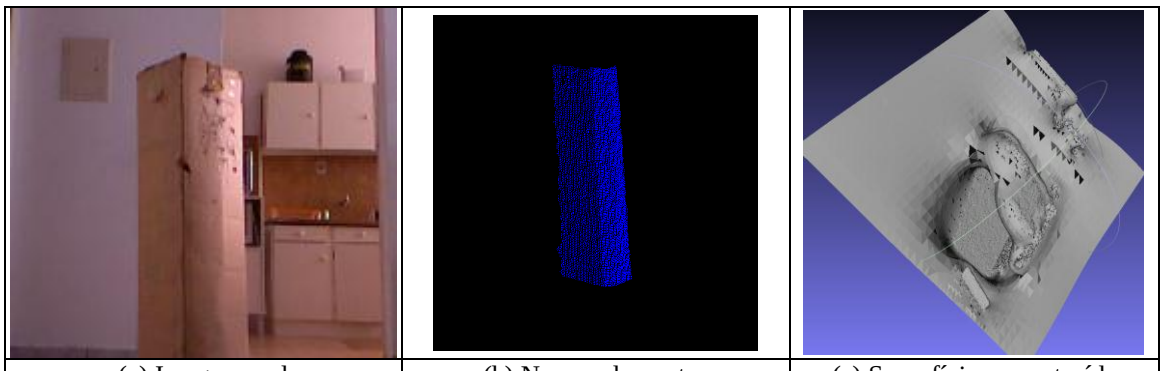


(a) Imagem real | (b) Nuvem de pontos | (c) Superfície reconstruída

Figura 30 - Exemplo de reconstrução de superfície

Durante o desenvolvimento da aplicação avaliou-se a solução de utilizar o executável "*Screened Poisson Surface Reconstruction*" que utiliza o método apresentado na seção 2.2.2.1 a partir dos pontos coletados pela aplicação desenvolvida. Foi fornecido um arquivo contendo as coordenadas tridimensionais e gerado um arquivo *PoLYgon file format* (PLY). Esse método apresentou problemas, porque ele necessita das coordenadas de todos os lados do objeto para gerar a superfície do mesmo, e a solução desenvolvida não possuía todas as informações necessárias.

A Figura 31 exemplifica o resultado dos testes utilizando o executável "*Screened Poisson Surface Reconstruction*". Analisando a superfície reconstruída na Figura 31 item (c), percebe-se que não é gerada uma superfície parecida com o seu objeto original (Figura 31 item a) e nem com a nuvem de pontos (Figura 31 item b).



(a) Imagem real | (b) Nuvem de pontos | (c) Superfície reconstruída

Figura 31 - Reconstrução de superfície incorreta

A superfície gerada pela aplicação não teve sua qualidade analisada ou comparada com outras aplicações. Foi percebido durante os testes da aplicação que objetos com superfícies uniformes, geram uma reconstrução de superfície melhor, do que objetos com superfície disforme. A aplicação não consegue fazer a reconstrução de superfícies de objetos que refletem luz, visto que o Kinect projeta luz infravermelha nos objetos para obter a distância deles.

Na aplicação pequenos objetos que estão perto de outra superfície muitas vezes se tornam difícil de distinguir, já que a aplicação mostra a reconstrução de superfície em tons de vermelho.

O Quadro 14 mostra uma comparação da aplicação desenvolvida com as principais características dos trabalhos correlatos.

características / trabalhos	Aplicação desenvolvida	IZADI et al. (2011)	LÜBKE (2012)	RHEMYST; RYMIX (2011)	SILVEIRA (2011)
utiliza Kinect *for* Windows SDK	X	X	X	X	X
utiliza *stream* de vídeo	X	X	X	X	
utiliza *stream* de profundidade	X	X			
utiliza *stream* de áudio			X		
utiliza *skeletal tracking*	X		X	X	X
faz reconstrução de superfícies	X	X			
faz reconstrução incremental		X			
simula física no ambiente tridimensional gerado		X			
principal funcionalidade utilizada do Kinect *for* Windows SDK	*stream* de profundidade	*stream* de profundidade	*skeletal tracking*	*skeletal tracking*	*skeletal tracking*

Quadro 14 - Comparação da aplicação desenvolvida com os trabalhos correlatos

A partir do Quadro 14, pode-se observar que aplicação desenvolvida comparada com o KinectFusion (IZADI et al., 2011a) que é o único trabalho correlato que faz reconstrução de superfícies, mostra que a aplicação desenvolvida não faz reconstrução incremental e simulação de física no ambiente tridimensional gerado. Os outros trabalhos correlatos não têm o objetivo de gerar uma reconstrução de superfície, mas utilizam o Kinect *for* Windows SDK, sendo que a principal funcionalidade utilizada desse é o *skeletal tracking*, onde essas aplicações detectam movimentos dos usuários para definir funções, enquanto que a aplicação desenvolvida e o KinectFusion utilizam principalmente o *stream* de profundidade, com objetivo de capturar dados de profundidade, para gerar a reconstrução de superfícies.

4 CONCLUSÕES

Este trabalho descreve o desenvolvimento de uma aplicação que permite gerar a reconstrução de superfície de um ambiente real para um ambiente tridimensional virtual, utilizando o Kinect.

A partir dos experimentos acima, pode-se concluir que o trabalho cumpriu o objetivo proposto de fazer a reconstrução de superfície de um ambiente real obtendo informações do Kinect. Através deste trabalho, é possível realizar a reconstrução de superfície em tempo real, obtendo informações do *stream* de profundidade do Kinect *for* Windows SDK, e exibido para o usuário através de funções da biblioteca OpenTK. A superfície gerada pela aplicação não teve sua qualidade analisada ou comparada com outras aplicações.

Como limitação, pode-se citar que a aplicação não gera uma reconstrução incremental de superfície, nem armazena os dados capturados do Kinect em alguma estrutura, sendo assim ela gera a superfície a cada *frame* detectado pelo Kinect eliminando os dados do *frame* anterior. Outra limitação da aplicação é que ela não apresenta a cor real do objeto reconstruído, mas somente em tons de vermelho.

A aplicação também tem limitações causadas pelo próprio Kinect, como:

a) ângulo de visão do Kinect;
b) distância que o Kinect consegue determinar a posição de uma superfície que atualmente é de 8,19 metros;
c) objetos que o Kinect não consegue obter informações de profundidade (como espelhos, superfícies anti-reflexivas).

Além do que foi proposto, na aplicação também foi desenvolvida a visualização do *skeleton view* e da imagem captada pela câmera do Kinect para demonstrar algumas funcionalidades do Kinect e seu SDK.

Contudo, este trabalho poderá ser utilizado como referência para novos estudos na reconstrução de superfícies e para o desenvolvimento de aplicações utilizando Kinect.

4.1 EXTENSÕES

Como proposta de continuidade sugere-se implementar a reconstrução incremental do

modelo tridimensional fazendo com que a cada *frame* que o Kinect tiver de profundidade, seja realizada uma comparação com o frame anterior, detectando pontos já obtidos e adicionando os novos pontos detectados à nuvem de pontos.

Realizar a reconstrução de superfície com a cor real do objeto através da câmera de vídeo do Kinect, considerando que a câmera que detecta a profundidade e a câmera de vídeo não são a mesma, o que introduz uma pequena diferença do ângulo de visão, outro cuidado a ser tratado é a resolução utilizada para capturar a profundidade e o vídeo, que podem ser diferentes.

Fazer mais testes com situações controladas como: ambientes com variações de iluminação, objetos com diferentes tamanhos, reconstrução de objetos nos limites de captura de profundidade do Kinect, entre outros.

REFERÊNCIAS BIBLIOGRÁFICAS

AGUIAR, Leonardo C. **A evolução dos video games.** [S.l.], 2010a. Disponível em: <http://awakefromnib.blogspot.com.br/2010/08/evolucao-dos-video-games.html>. Acesso em: 16 out. 2012.

AGUIAR, Thiago V. **Visualização da fronteira entre fluidos utilizando o método sph e o algoritmo de marching cubes.** Rio de Janeiro, 2010b. Disponível em: <http://www.maxwell.lambda.ele.puc-rio.br/Busca_etds.php?strSecao=resultado&nrSeq=15358>. Acesso em: 13 out. 2012.

BOURKE, Paul. **Polygonising a scalar field.** [S.l.], 1994. Disponível em: <http://paulbourke.net/geometry/polygonise/>. Acesso em: 03 nov. 2012.

BURRUS, Nicolas. **Kinect calibration.** [S.l.], 2011. Disponível em: <http://nicolas.burrus.name/index.php/Research/KinectCalibration>. Acesso em: 17 out. 2012.

CAMARGO, Marcelo A. R. **Geração de objetos virtuais a partir de imagens.** Piracicaba, 2008. Disponível em: <http://www.unimep.br/phpg/bibdig/pdfs/2006/KDJPMVVBHDVJ.pdf>. Acesso em: 07 abr. 2012.

CASTRO, André. **Kinect SDK 1.5 parte 2:** câmera de profundidade (depth)**.** [S.l.], 2012. Disponível em: <http://www.100loop.com/destaque/kinect-sdk-1-5-parte-2-camera-de-profundidade-depth/>. Acesso em: 15 out. 2012.

DIETRICH, Carlos A. **Grupos de arestas:** uma nova abordagem para entender a qualidade da malha gerada pelo marching cubes e suas variantes. Porto Alegre, 2008. Disponível em: <www.inf.ufrgs.br/~comba/papers/thesis/tese-dietrich.pdf>. Acesso em: 27 ago. 2012.

EISLER, Craig. **Kinect for Windows is now available!** [S.l.], 2012. Disponível em: <http://blogs.msdn.com/b/kinectforwindows/archive/2012/01/31/kinect-for-windows-is-now-available.aspx>. Acesso em: 05 abr. 2012.

HENRY, Peter et al. **RGB-D mapping:** using Kinect-style depth cameras for dense 3D modeling of indoor environments. [S.l.], 2012. Disponível em: <http://homes.cs.washington.edu/~xren/publication/henry-ijrr12-rgbd-mapping.pdf>. Acesso em: 05 nov. 2012.

IZADI, Shahram et al. **KinectFusion:** real-time 3D reconstruction and interaction using a moving depth camera. United Kingdom, 2011a. Disponível em: <http://research.microsoft.com/pubs/155416/kinectfusion-uist-comp.pdf>. Acesso em: 05 ago. 2012.

IZADI, Shahram et al. **KinectFusion:** real-time dense surface mapping and tracking. [S.l.], 2011b. Disponível em: <http://research.microsoft.com/pubs/155378/ismar2011.pdf>. Acesso em: 05 ago. 2012.

LORENSEN, Bill. **Marching cubes.** [S.l.], 2007. Disponível em: <http://www.marchingcubes.org/index.php/Marching_Cubes>. Acesso em: 07 out. 2012.

LOWRY, Nate. **How the Kinect senses depth.** [S.l.], 2010. Disponível em: <http://nongenre.blogspot.com.br/2010/12/how-kinect-senses-depth.html>. Acesso em: 18 out. 2012.

LÜBKE, Florian. **KinectCAD**. [S.l.], 2012. Disponível em: <http://sourceforge.net/projects/kinectcad/>. Acesso em: 03 abr. 2012.

MANSSOUR, Isabel H. **Introdução à opengl**. [S.l.], 2003. Disponível em: <http://www.inf.pucrs.br/~manssour/OpenGL/Introducao.html>. Acesso em: 25 jun. 2012.

MICROSOFT. **Coordinate spaces.** [S.l.], 2012a. Disponível em: <http://msdn.microsoft.com/en-us/library/hh973078.aspx>. Acesso em: 04 ago. 2012.

_____. **Interaction space.** [S.l.], 2012b. Disponível em: <http://msdn.microsoft.com/en-us/library/hh973071.aspx>. Acesso em: 04 ago. 2012.

_____. **Introduction to WPF.** [S.l.], 2012c. Disponível em: <http://msdn.microsoft.com/en-us/library/aa970268.aspx>. Acesso em: 01 out. 2012.

_____. **Kinect for Windows architecture.** [S.l.], 2012d. Disponível em: <http://msdn.microsoft.com/en-us/library/jj131023.aspx>. Acesso em: 04 ago. 2012.

_____. **Kinect for Windows human interface guidelines v1.5.0.** [S.l.], 2012e. Disponível em: <http://msdn.microsoft.com/en-us/library/jj663791.aspx>. Acesso em: 04 ago. 2012.

_____. **Kinect for Windows sensor components and specifications.** [S.l.], 2012f. Disponível em: <http://msdn.microsoft.com/en-us/library/jj131033.aspx>. Acesso em: 04 ago. 2012.

_____. **Kinect Natural User Interface (NUI) overview.** [S.l.], 2012g. Disponível em: <http://msdn.microsoft.com/en-us/library/hh855348.aspx>. Acesso em: 05 abr. 2012.

_____. **Skeletal tracking.** [S.l.], 2012h. Disponível em: <http://msdn.microsoft.com/en-us/library/hh973074.aspx>. Acesso em: 04 ago. 2012.

MILES, Rob. **Using Kinect for Windows with XNA.** Kinect for Windows SDK. [S.l.], 2012. Disponível em: <http://channel9.msdn.com/coding4fun/kinect/The-Purple-Book-Using-Kinect-for-Windows-with-XNA>. Acesso em: 15 mar. 2012.

MOTA, Tiago de S. **Métodos de reconstrução de superfície.** Juiz de Fora, 2007. Disponível em: <http://www.gcg.ufjf.br/pub/doc46.pdf>. Acesso em: 18 abr. 2012.

OPENTK. **The open toolkit library.** [S.l.], 2008. Disponível em: <http://www.opentk.com/project/opentk>. Acesso em: 03 ago. 2012.

QUEIROZ, Murilo. **Um cientista explica o Microsoft Kinect.** [S.l.], 2010a. Disponível em: <http://blog.vettalabs.com/2010/10/29/um-cientista-explica-o-microsoft-kinetic/>. Acesso em: 23 ago. 2012.

_____. **Um cientista explica o Microsoft Kinect, parte II.** [S.l.], 2010b. Disponível em: <http://blog.vettalabs.com/2010/11/02/um-cientista-explica-o-microsoft-kinect-parte-ii/>. Acesso em: 23 ago. 2012.

RHEMYST, John; RYMIX, Kaly. **Kinect SDK dynamic time warping (DTW) gesture recognition.** [S.l.], 2011. Disponível em: <http://kinectdtw.codeplex.com/>. Acesso em: 03 abr. 2012.

SILVEIRA, Marcus A. **Técnica de navegação em documentos utilizando Microsoft Kinect**. Porto Alegre, 2011. Disponível em: <http://www.lume.ufrgs.br/bitstream/handle/10183/36884/000819161.pdf?sequence=1>. Acesso em: 20 abr. 2012.