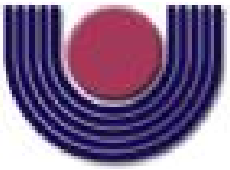

Unioeste - Universidade Estadual do Oeste do Paraná
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
Colegiado de Ciência da Computação
Curso de Bacharelado em Ciência da Computação

Implementação do módulo simulador de robôs baseado em Unity3D para o SimBot - Simulador de Robôs para Lego NXT

Remi Pietsch Junior

CASCAVEL
2014

REMI PIETSCH JUNIOR

IMPLEMENTAÇÃO DO MÓDULO SIMULADOR DE ROBÔS BASEADO EM UNITY3D PARA O SIMBOT - SIMULADOR DE ROBÔS PARA LEGO NXT

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação, do Centro de Ciências Exatas e Tecnológicas da Universidade Estadual do Oeste do Paraná - Campus de Cascavel

Orientadora: Profª. M.Eng. Adriana Postal

CASCAVEL
2014

REMI PIETSCH JUNIOR

IMPLEMENTAÇÃO DO MÓDULO SIMULADOR DE ROBÔS BASEADO EM UNITY3D PARA O SIMBOT - SIMULADOR DE ROBÔS PARA LEGO NXT

Monografia apresentada como requisito parcial para obtenção do Título de Bacharel em Ciência da Computação, pela Universidade Estadual do Oeste do Paraná, Campus de Cascavel, aprovada pela Comissão formada pelos professores:

Profª. M.Eng. Adriana Postal (Orientadora)
Colegiado de Ciência da Computação,
UNIOESTE

Prof. M.Eng. Josué Pereira de Castro
Colegiado de Ciência da Computação,
UNIOESTE

Prof. Dr. Pedro Luiz de Paula Filho
Departamento de Ciência da Computação,
UTFPR-Medianeira

Cascavel, 5 de dezembro de 2014

DEDICATÓRIA

Gostaria de dedicar esse trabalho a minha familia e aos professores que me auxiliaram durante esse ano, principalmente a minha orientadora, que teve que me aguentar o ano todo reclamando sobre o trabalho, faculdade e vida pessoal. Sem essas pessoas esse trabalho não teria saido.

AGRADECIMENTOS

Agradecer ao Programa de Educação Tutorial do Ministério da Educação pelo apoio durante os anos de faculdade e pelo suporte durante o projeto desenvolvido.

Agradecer aos professores do curso por terem tirado dúvidas sobre os assuntos que foram vistos durante o trabalho, em especial aos professores Adriana Postal, Josué Pereira de Castro e Jeferson José Baqueta.

Agradecer a minha familia e amigos por terem insistido para que eu continuasse a fazer o trabalho e não desistir no meio do caminho.

Lista de Figuras

Lista de Abreviaturas e Siglas

CAD	*Computer-Aided Design*
DLL	*Dinamic-Link Library*
GUI	*Graphical User Interface*
IDE	*Integrated Development Environment*
ODE	*Open Dynamics Engine*
SARGE	*Search and Rescue Game Environment*

Sumário

Resumo

Este trabalho apresenta a construção de um simulador de robôs baseados nas peças do kit LEGO Mindstorms que permite a criação de modelos e a simulação desses modelos em um ambiente virtual, de aparência semelhante a um ambiente real controlado. Apresenta-se também uma análise dos programas utilizados no desenvolvimento, assim como as ferramentas pertencentes a cada um deles.

Palavras-chave: LEGO, Simulação, Unity

Capítulo 1

Introdução

A disciplina optativa Introdução à Robótica Inteligente é ofertada todo o ano no Curso de Ciência da Computação da Universidade Estadual do Oeste do Paraná, Campus de Cascavel. A disciplina é dividida em aulas teóricas e práticas, onde as aulas práticas são realizadas utilizando kits LEGO Mindstorms. O kit LEGO Mindstorms é composto, inicialmente, por 76 peças diferentes (há a possibilidade de adicionar peças externas ao kit para que seja atendida as necessidades do usuário), dentre elas peças elétricas, não-elétricas, cabos e esteiras. Suas peças, assim como a quantidade de cada peça por kit, podem ser vistas na Figura 1.1 (LEGO Group, 2009).

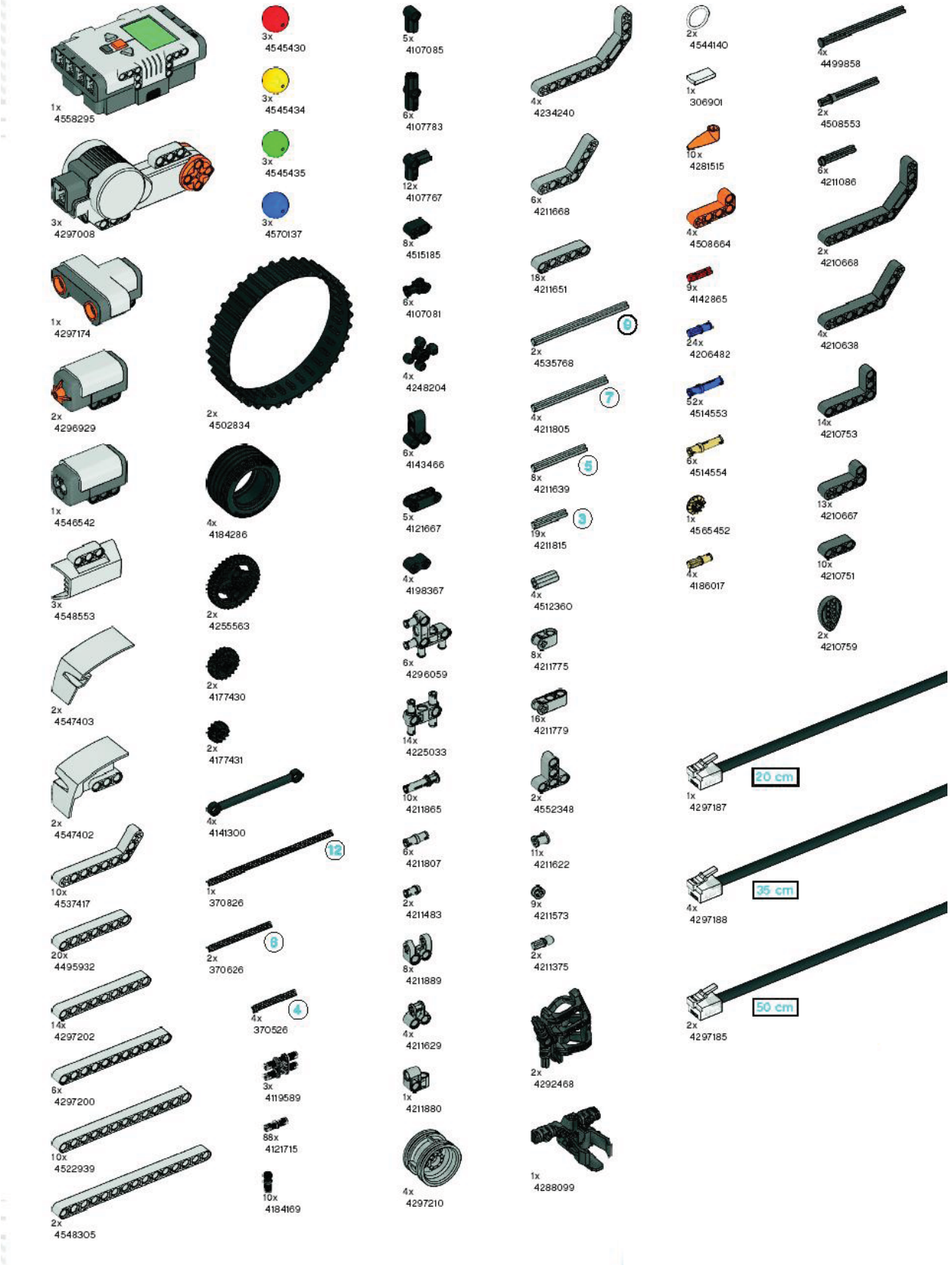

Figura 1.1: Lista de peças que compõem o kit LEGO Mindstorms.

1.1 Descrição do Problema

Devido ao número limitado de kits para a realização das aulas práticas foi discutido com os professores da disciplina sobre a possibilidade de desenvolver um programa que permitisse a criação e a simulação de robôs Lego virtualmente a fim de melhorar o aprendizado da disciplina. Com base nas discussões feitas sobre os requisitos do programa, o mesmo foi dividido em três módulos, que podem ser vistos na Figura 1.2:

- Editor/Tradutor de código NXC;
- Modelador de Robôs;
- Módulo Simulador.

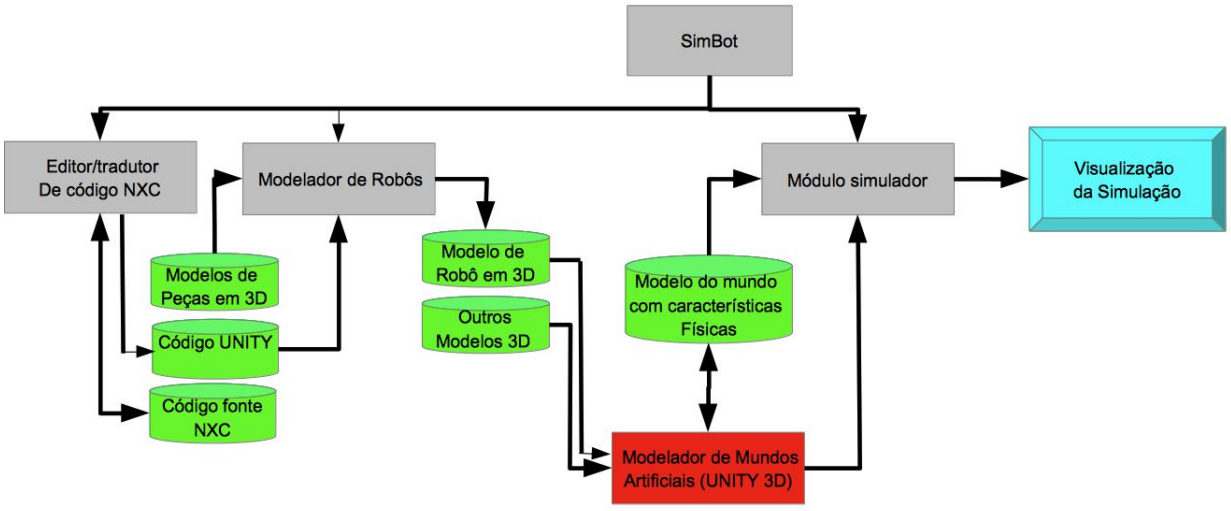

Figura 1.2: Divisão dos três módulos do SimBot.

1.2 Objetivos

O módulo desenvolvido nesse trabalho corresponde ao Modelador de Robôs mostrado na Figura 1.3, o qual permitirá a criação de modelos de robôs baseados no kit LEGO Mindstorms virtualmente, utilizando uma biblioteca virtual de peças. Os objetivos específicos deste trabalho são:

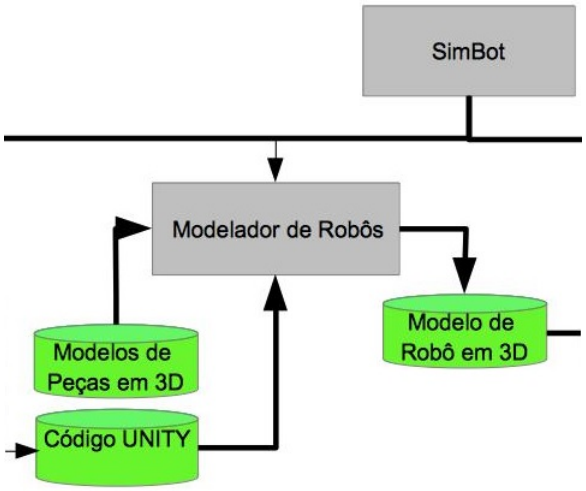

Figura 1.3: Modelador de Robôs.

- Criar uma biblioteca com as peças já modeladas do kit LEGO Mindstorms utilizado para a montagem dos robôs;
- Avaliar e definir as ferramentas que serão utilizadas no desenvolvimento do módulo;
- Avaliar e definir os movimentos que as peças podem realizar decorrentes do movimento dos eixos do motor;
- Aplicar aos motores a movimentação do seu eixo;
- Desenvolver *scripts* das peças como sensores, motores e *brick* para que possam desempenhar suas funções corretamente;
- Aplicar às peças comportamento físicos, tais como atrito e resistência mecânica, para que se mantenham juntas quando encaixarem e para que simulem de modo mais fiel uma peça real.

1.3 Justificativa

Embora existam outros programas, como por exemplo o RobotC (Robomatter Inc, 2014), e trabalhos que estejam focados na realização de simulações virtuais com modelos de robôs, como será visto na próxima seção, nenhum deles possibilita a criação e a simulação de robôs baseados no kit LEGO Mindstorms.

1.4 Trabalhos Correlatos

A fim de entender melhor como programas de simulação são desenvolvidos, bem como as *engines* e outros programas que são utilizados para a sua implementação os artigos analisados para esse trabalho foram selecionados dando preferência para os que utilizassem ou falassem sobre os programas que foram usados neste trabalho. As discussões encontradas nesses artigos foram de grande ajuda para melhor compreender as capacidades de cada programa escolhido para o trabalho e ter uma melhor visão de como um simulador funciona e como é criado.

Hounsell e Redel (2014) mostram um passo a passo de como implementar um simulador de robôs manipuladores em *Virtual Reality Modeling Language* (Association for Computing Machinery, 2014). Eles explicam, primeiramente, os conceitos básicos que são utilizados em simulações virtuais como robô, realidade virtual, robótica e tipos de programação. Em seguida, são mostrados os passos necessários para a criação de um simulador através de um exemplo desenvolvido pelos autores do artigo. Para que um simulador passe a funcionar, é necessário que exista descrição da geometria, especificação da estrutura e programação da interação.

Em GONÇALVES et al. (2009) é feita uma comparação de como um modelo de robô desenvolvido pelos autores se comporta em um ambiente real e em um ambiente virtual utilizando o programa SimTwo (COSTA, 2012) para a simulação do robô virtualmente. São feitas explicações sobre como são tratadas as peças dentro do programa e como o software reconhece e entende o que deve ser feito. Também trata-se sobre como a ODE (*Open Dynamics Engine*) (SMITH, 2008) é utilizada para produzir a movimentação e reconhecer os comportamentos físicos necessárias.

Yamamoto et al. (2003) mostram a criação de um simulador para um torneio de futebol de robôs, onde existe uma câmera para cada time que tira fotos do campo em um intervalo de tempo

determinado, um grupo de rôbos que funcionam como os jogadores, uma bola e um campo montado em cima de uma mesa ou bancada. Os autores dizem que devem ser observados três pontos importantes na hora de desenvolver um simulador: a modelagem do robô, a modelagem de colisões e o tempo de simulação. Nesse artigo são descritos também os passos para a criação do simulador para o torneio de futebol de robôs, desde a modelagem do robô até o tratamento de colisões e o sistema de simulação.

Craighead, Burke e Murphy (2008) trazem uma discussão sobre as vantagens do uso do Unity para a criação do SARGE (*Search and Rescue Game Environment*), um simulador desenvolvido para testar e aprimorar a capacidade de usuários de controlar um ou vários robôs utilizados em buscas, resgates e aplicações das leis (governamentais, estaduais). São comentadas as vantagens referentes a documentação do Unity disponibilizada, sobre a sua comunidade sempre ativa em relação a resolver problemas ou tirar dúvidas, sobre a organização da sua interface e a facilidade de uso comparada com outras *engines* estudadas pelos autores, distribuição multiplataforma, renderização e física empregadas pelo programa e o baixo custo para o seu uso.

Silva (2012) propõe a criação de uma subestação de energia elétrica virtual para o treinamento de operadores. A escolha da criação desse programa para treinamento considerou a possibilidade de submeter os operadores a diversas situações, inclusive aquelas com risco de morte. Outra vantagem avaliada pelo autor é a possibilidade de realizar os treinos a distância, mas que ainda seja possível realizá-los com a supervisão de alguém mais instruído. A ferramenta escolhida para o desenvolvimento desse projeto foi o Unity 3D, pois o mesmo se enquadrava em todas as exigências feitas pelo autor.

1.5 *Softwares* Correlatos

Foram analisados e testados alguns programas que realizam simulações virtuais com o objetivo de entender como funciona um programa de simulação, quais são as características que são necessárias para realizar as simulações, como as peças interagem com as outras e com o ambiente. Um dos *softwares*, o LDraw, foi de grande ajuda na produção do conjunto de peças que são utilizadas para a criação dos modelos físicos Lego.

O AForge é um *framework* de código aberto desenvolvido na linguagem C# que é voltado

para as áreas de Computação Visual e Inteligência Artificial como processamento de imagens, redes neurais, algoritmos genéticos, lógica *Fuzzy*, aprendizado de máquina, robótica, entre outras. O AForge disponibiliza diversas bibliotecas que auxiliam os usuários a desenvolver seus projetos voltados às áreas citadas anteriormente (AForge.NET, 2008).

O LDraw é um software de código aberto para programas LEGO CAD (*Computer-Aided Design*) que possibilita ao usuário a criação de modelos e cenas virtuais com LEGO. A animação dos modelos gerados pelo LDraw deve ser feita através de outros programas, visto que ele não possui as ferramentas necessárias para criar e simular animações. Pode ser utilizado para documentar modelos que foram fisicamente construídos, criar instruções para a montagem de modelos LEGO e criar ou recriar fotos reais em 3D a partir dos modelos desenvolvidos virtualmente. O LDraw possui uma grande biblioteca de peças onde o usuário não se limita a cores ou números de peças na hora da criação dos modelos (LDRAW, 2003).

O Microsoft Robotics Developer Studio é um programa implementado para Windows voltado para o desenvolvimento de aplicações robóticas para uma grande variedade de plataformas de hardware. Esse programa possui vários tutoriais e documentação que auxiliam na hora da criação dos modelos robóticos. Também conta com ferramentas de simulação, linguagem de programação visual, monitoramento em tempo real do robô, controle múltiplo de robôs a partir de códigos (MICROSOFT, 2014a).

O Simbad é um programa desenvolvido em Java que permite a simulação de robôs 3D para fins educacionais e científicos. Ele não tenta recriar uma simulação em mundo real e tem foco para estudos voltados à área de Inteligência Artificial, Aprendizado de Máquina, e geralmente algoritmos para Robôs e Agentes Autônomos. Dentro desse programa é possível implementar algoritmos próprios para controlar os modelos robóticos criados a fim de testá-los no ambiente virtual gerado. O Simbad possibilita uma visualização 3D do ambiente, simulações com um ou mais robôs, sensores de visão, distância e contato, e uma interface mais organizada para o controle do robô (HUGUES; BREDECHE, 2011).

O Webots é um software livre que é utilizado para desenvolver, programar e simular robôs. Dentro desse programa é possível desenvolver robôs simples ou complexos, tendo um ou mais robôs sendo controlados, sejam eles iguais ou diferentes. As características de cada robô, como cor, textura, massa, forma, são definidos pelo usuário. O robô criado pode ser controlado a partir

de algoritmos desenvolvidos na IDE (*Integrated Development Environment*) do programa, onde é atribuído o código ao modelo, que também pode ser testado em um ambiente virtual que possui as características físicas do mundo real. A análise do comportamento do modelo pode ser feita através da variedade de sensores disponíveis no programa (Cyberbotics Ltd., 2014).

1.6 Organização do Trabalho

O trabalho está organizado em quatro capítulos:

- Introdução;
- Modelagem 3D;
- Desenvolvimento;
- Considerações Finais.

Em Modelagem 3D será brevemente descrito o que é modelagem e algumas técnicas utilizadas para a modelagem de objetos tridimensionais. No Desenvolvimento será abordada a criação do módulo implementado, desde a criação da biblioteca até a finalização dos objetivos propostos no começo do trabalho. Também será visto nesse capítulo os testes realizados, bem como os seus resultados. Nas Considerações Finais será descrito o que foi realizado no trabalho, quais foram os problemas encontrados durante o processo de desenvolvimento, como foram resolvidos ou contornados, quais foram os resultados obtidos após a finalização e quais são os próximos passos a serem realizados.

Capítulo 2

Modelagem 3D

Modelagem Tridimensional, ou Modelagem 3D, é uma área da Computação Gráfica que tem como objetivo a geração de entidades em três dimensões, cenas estáticas e imagens em movimento, com ou sem interatividade. Ela consiste no processo de representação de um fato ou fenômeno através da abstração, onde podem ser físicos ou matemáticos. Os modelos gerados com a Computação Gráfica são os modelos matemáticos, implementados sob a forma de algoritmos computacionais. Em outras palavras é a criação de formas, objetos, personagens e cenários através da utilização de ferramentas computacionais avançadas e direcionadas para este tipo de tarefa, onde podem ser encarada sobre dois aspectos, modelagem geométrica e modelagem procedural (FONSECA, 2007).

A modelagem de um objeto tridimensional pode ser feita tomando como base um objeto primitivo contido no programa utilizado e modificá-lo, seja estendendo, mudando sua forma ou acrescentando outros objetos a ele. Um objeto primitivo pode ser um ponto, uma linha, uma curva ou um objeto tridimensional. A malha gerada pelo programa é chamada de *Mesh* (Tonka 3D, 2013).

O conhecimento sobre as técnicas que serão apresentadas a seguir foram obtidas através do trabalho de Silva (2000). O objetivo desse estudo foi entender algumas das técnicas que são utilizadas para a modelagem tridimensional de objetos para que auxiliasse na correção ou adaptação futura das peças, caso fosse necessário.

2.1 *Spline*

A *spline* é uma ferramenta para modelagem utilizada para a criação de objetos sólidos como construções, mobílias, acessórios, entre outros, pois usa uma combinação de curvas e ângulos (Figura 2.1).

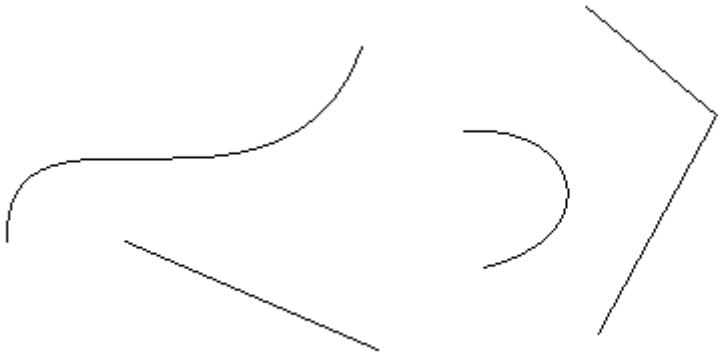

Figura 2.1: Exemplo de *Splines*.

A criação de modelos 3D a partir desse método de modelagem se faz com a criação de curvas (*splines*) definidas por pelo menos dois pontos de controle, onde esse conjunto de curvas é utilizado para montar o esqueleto do objeto. Dentre as curvas utilizadas nesse método de modelagem as mais comuns são as curvas de Bézier e NURBS.

São utilizados vários sub-objetos para a criação de uma *spline*, onde cada sub-objeto possui suas características próprias. Existem dois tipos de sub-objetos: os vértices e os segmentos.

- Vértices: Na perspectiva da geometria, o vértice é o nome que recebe o ponto de intersecção entre os segmentos que originam um ângulo;
- Segmento: Na geometria, um segmento é definido por uma linha que liga dois pontos chamados de extremos.

Cada *spline* é composta por um conjunto de vértices e segmentos. Existe um vértice especial que é chamado de primeiro vértice, onde esse vértice é importante para as formas fechadas. Para que uma *spline* seja chamada de forma fechada, o último vértice dela deve estar ligado com o primeiro (Figura 2.2), pois, por definição, uma forma fechada não possui cortes em seu

perímetro. Quanto mais segmentos uma *spline* possuir, mais detalhada será a forma criada, consequentemente mais memória será utilizada para guardar a *spline*.

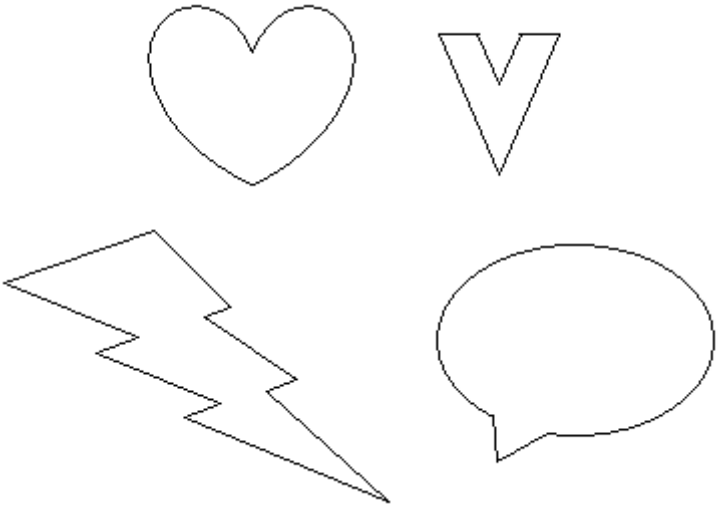

Figura 2.2: Exemplo de formas fechadas.

2.1.1 Curva de Bézier

Uma curva de Bézier é criada a partir de dois pontos extremos unidos por uma linha e dois outros pontos para controlar a tangente nos extremos. Através do ponto de controle é possível definir a curva que a linha irá fazer ao se aproximar do seu respectivo extremo (Figura 2.3 (DATUOPINION, 2011)).

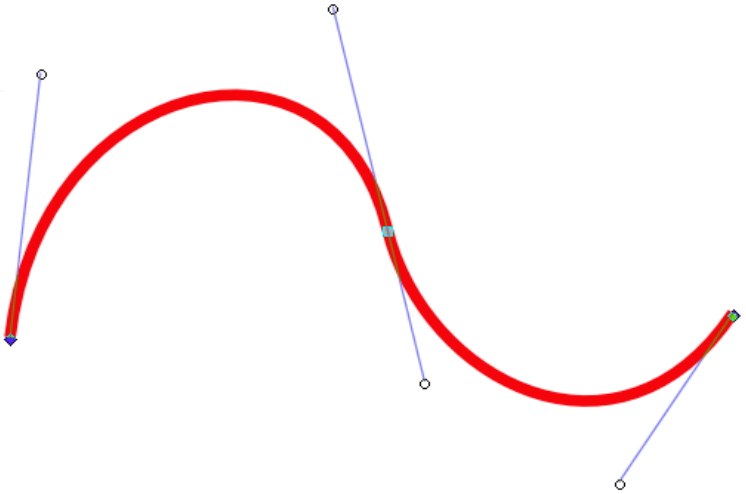

Figura 2.3: Exemplo de curva de Bézier.

2.1.2 Curva NURBS

A curva NURBS utiliza vértices de controle ou pontos. Quando são utilizados pontos para criar a curva, normalmente essa curva passa pelos pontos escolhidos (Figura 2.4 (Wikimedia Foundation, 2014b)). Quando são utilizados vértices de controle, raramente a curva passará por esses vértices (Figura 2.5 (CRUZ, 2007)). A utilização de vértices de controle gera vantagens sobre o uso apenas de pontos, pois com os vértices é possível ter um maior número de possibilidades para gerar uma curva com o mesmo número de vértices, isto é, um maior número de formas que a curva pode obter.

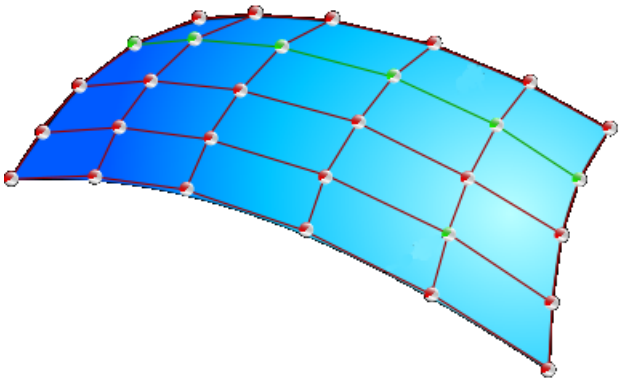

Figura 2.4: Exemplo de curva NURBS por pontos.

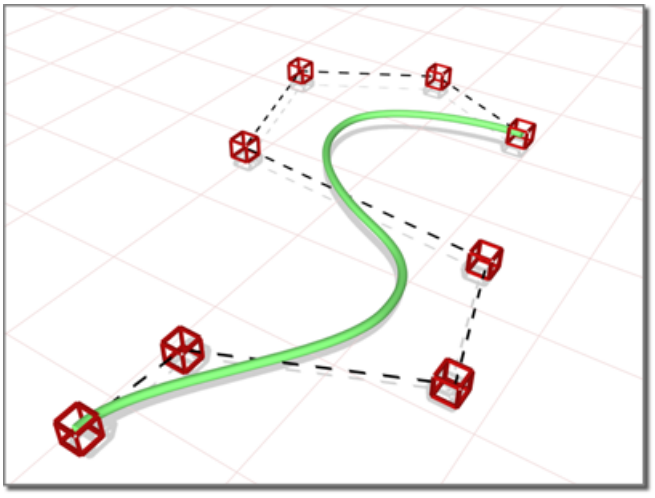

Figura 2.5: Exemplo de curva NURBS por vértices de controle.

2.1.3 Criando Objetos

O primeiro passo para a criação de um objeto pela técnica de *spline* é gerar os modelos que serão utilizados na modelagem. Esses modelos podem ser uma ou mais *splines* juntas. Após isso, esses modelos podem ser transformados em um objeto tridimensional através de técnicas de extrusão. A extrusão consiste em pegar o formato 2D criado e projetá-lo na direção de um vetor definido (a geratriz) (Figura 2.6 (PINHO, 2001)) ou seguindo um movimento escolhido (Figura 2.7 (CONCI, 2014)).

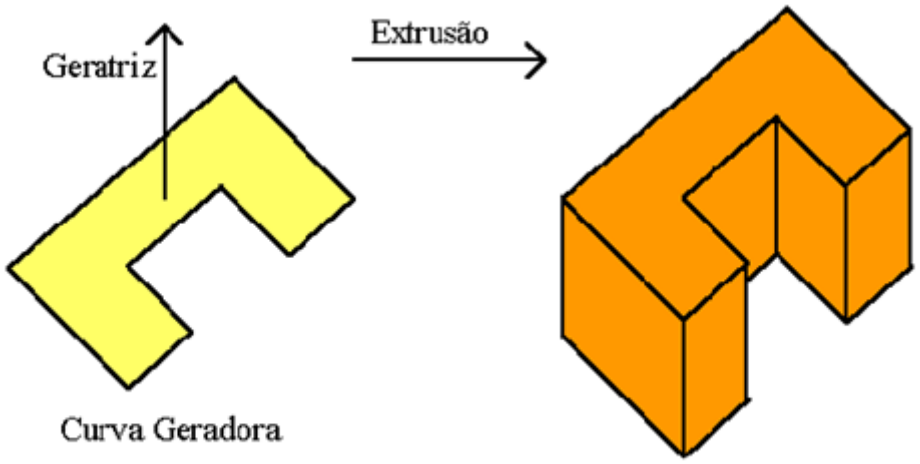

Figura 2.6: Exemplo de extrusão por vetor direcionado

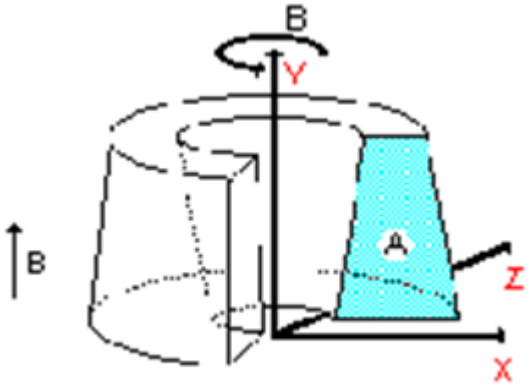

Figura 2.7: Exemplo de extrusão através de uma direção escolhida

É possível gerar uma forma através da combinação de mais de um modelo (Figura 2.8). Esse

conjunto pode ser utilizado para gerar objetos como paredes com janelas, molduras de quadros com várias partes juntas ou peças utilizadas em robôs Lego, como peças estruturais (Figura 2.9).

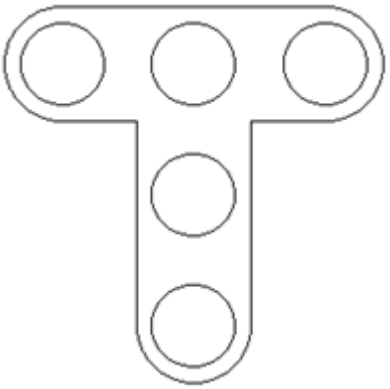

Figura 2.8: Exemplo de conjunto de modelos

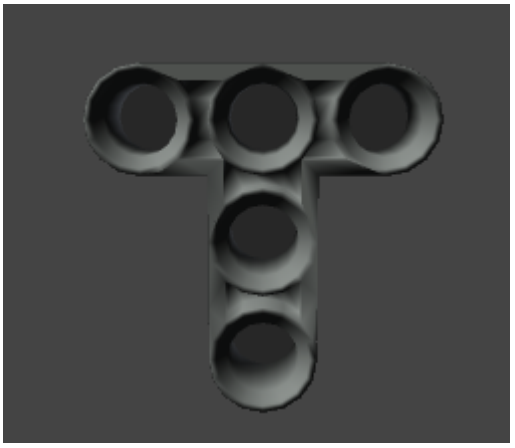

Figura 2.9: Resultado do uso de conjunto de modelos

Pode-se utilizar também uma *spline* como caminho a ser seguido, aplicando assim o modelo criado através dessa *spline* usada para definir o caminho (Figura 2.10 (Autodesk Inc, 2011)). Do mesmo modo que é extrudado um modelo através de um vetor, a extrusão de um modelo por um caminho cria várias cópias desse modelo até que seja completado o caminho.

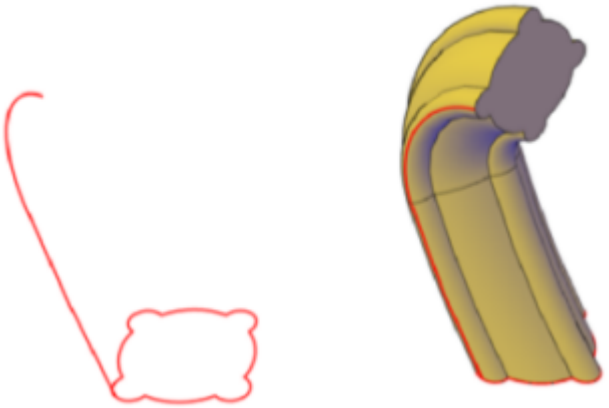

Figura 2.10: Exemplo do uso de uma *Spline* como caminho

Um outro método de usar a extrusão para a criação de objetos tridimensionais é utilizar uma *spline* criada e rotacioná-la sobre um eixo escolhido (Figura 2.11).

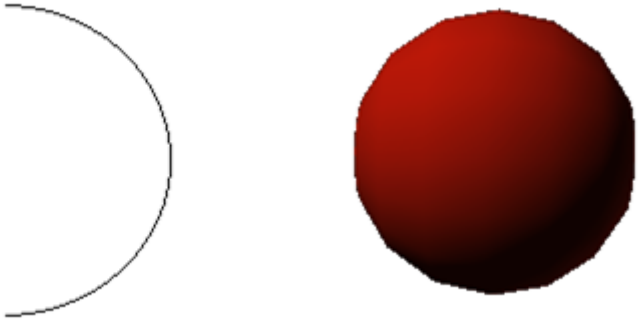

Figura 2.11: Exemplo do uso da rotação de uma *Spline* para criar uma esfera 3D

2.2 *Box Modeling*

A *Box Modeling* é considerada a técnica mais popular quando se trata de modelagem 3D. Pode-se assemelhar essa técnica à própria escultura tradicional, onde o objeto é iniciado por um primitivo, isto é, um bloco, uma esfera ou algum outro objeto, e a partir desse objeto fatia-se ou adiciona-se pedaços até que seja obtido o objeto modelado pretendido (Figura 2.12 (ANGHELESCU, 2008)).

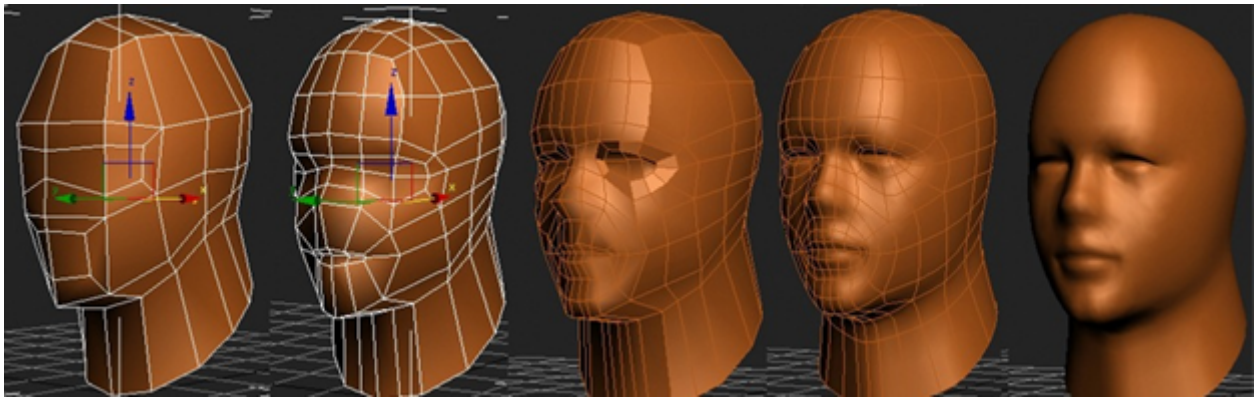

Figura 2.12: Exemplo de *Box Modeling*

Para acrescentar mais detalhes aos objetos modelados com essa técnica, é utilizado em conjunto com a *Box Modeling* a modelagem por subdivisão, onde cada face ou área do objeto é dividido em áreas menores a fim de melhorar o detalhamento do modelo (Figura 2.13 (Wikimedia Foundation, 2014a)).

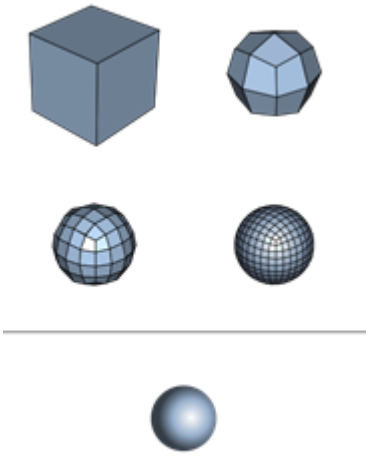

Figura 2.13: Exemplo do aumento de faces para deixar a imagem mais detalhada

A utilização dessa técnica pode-se tornar difícil quando o objetivo é criar um modelo com muitos detalhes, pois além de ser necessário que sejam feitos vários ajustes ao longo da modelagem, o resultado pode fugir do esperado. Contudo, a técnica de *Box Modeling* é uma técnica de fácil aprendizado e que é bem utilizada quando há a necessidade de criar modelos orgânicos como personagens e na criação de objetos rígidos.

2.3 Modelagem Poligonal

A criação de objetos tridimensionais através da modelagem poligonal se dá pela criação de uma malha de polígonos com três ou quatro lados. As malhas são compostas por sub-objetos chamados vértice, aresta e face (Figura 2.14). A três setas encontradas na figura representam apenas as direções que o objeto pode ser transladado pela função de translação.

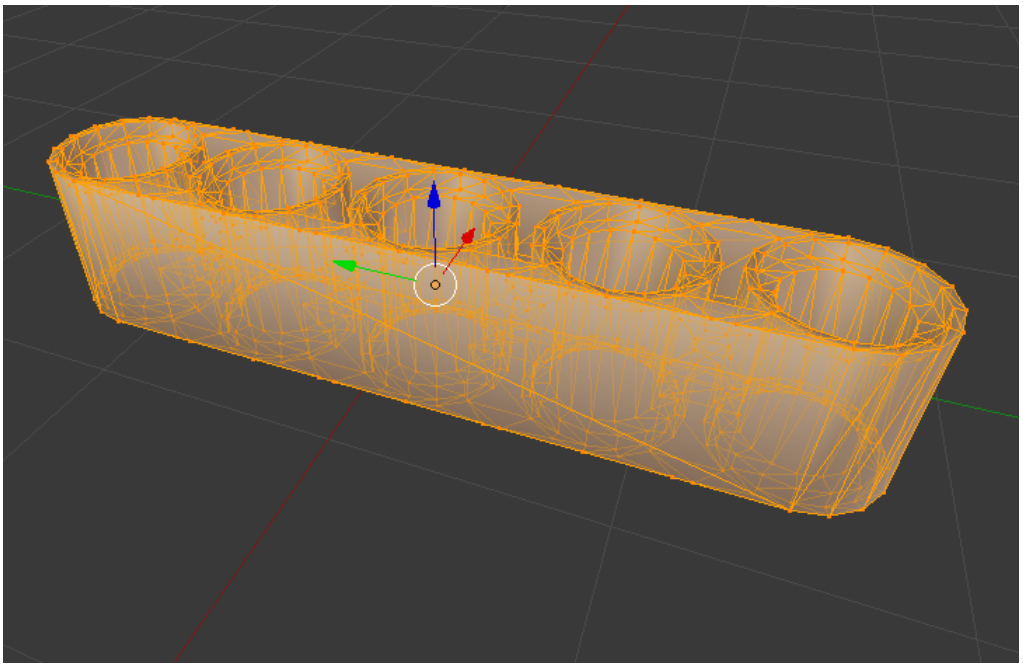

Figura 2.14: Exemplo de malha de um objeto

Os vértices, na modelagem poligonal, são pontos que, diferente da *spline*, não possuem controladores. Esses pontos são utilizados para ter um melhor controle do objeto modelado. Arestas são segmentos que ligam dois pontos em um objeto, onde, quanto mais segmentos uma malha conter, mais detalhes serão atribuídos para ela.

Faces são compostas por vértices e arestas. Quando o primeiro vértice está ligado ao último, tem-se um objeto fechado, ou seja, uma face. Quanto maior o número de faces que um objeto tiver, mais detalhado ele será. As faces contém um vetor normal que é projetado do seu centro, onde esse vetor é o responsável por dizer se essa face será visualizada ou não (Figura 2.15).

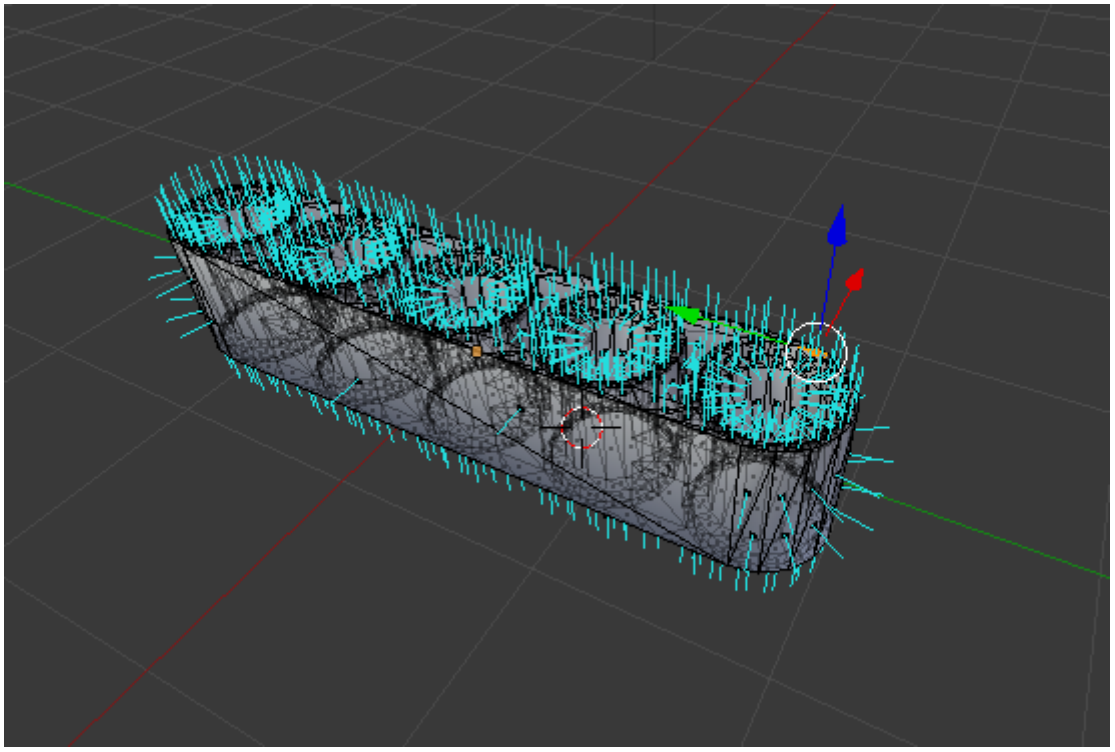

Figura 2.15: Vetores normais das faces de um objeto

A criação de um objeto pela modelagem poligonal inicia-se por um objeto primitivo, como um cubo ou cilindro, ou por objetos já modelados anteriormente. Normalmente é utilizado mais de um objeto para realizar a Modelagem Poligonal, por exemplo, caso queira modelar um haltere, basta juntar duas esferas e um cilindro (Figura 2.16). Um haltere é um peso utilizado durante treinos físicos para braços.

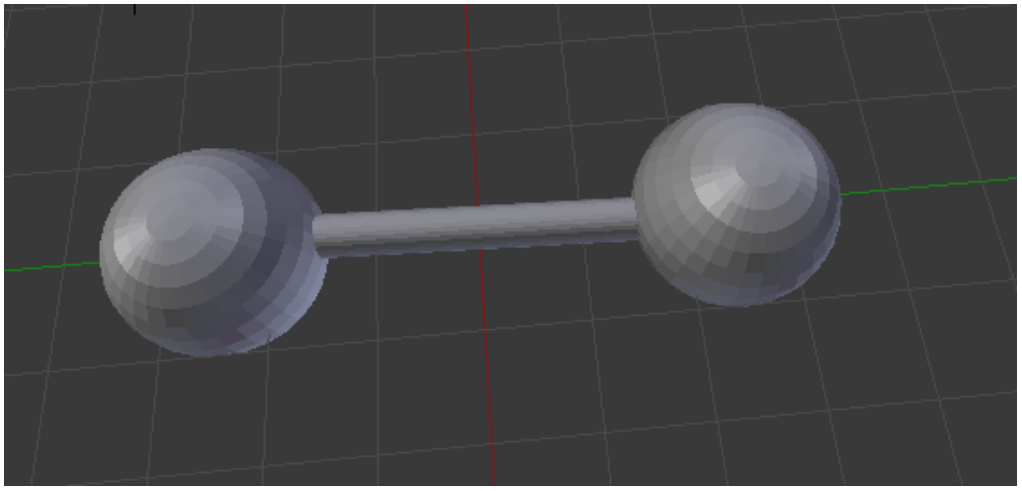

Figura 2.16: Exemplo de um objeto modelado pela técnica poligonal

A utilização dessa técnica para modelagem é fácil de usar, pois na sua maioria é a adição de um objeto a outro. Contudo, na criação principalmente de modelos orgânicos pode ser mais complicado de aplicá-la devido ao nível de detalhamento do objeto a ser modelado.

2.4 Modelagem por *Path*

A modelagem por *Path* utiliza uma superfície que será modelada e uma grade onde será definida a deformação aplicada na superfície a ser modelada. A deformação na grade é definida através da movimentação dos vértices da mesma. Quanto mais vértices e arestas uma grade possui, mais detalhado será o objeto modelado ao final, contudo, com um maior número de pontos na grade, maior será o custo computacional dessa modelagem (Figura 2.17 (QUACO, 2013)).

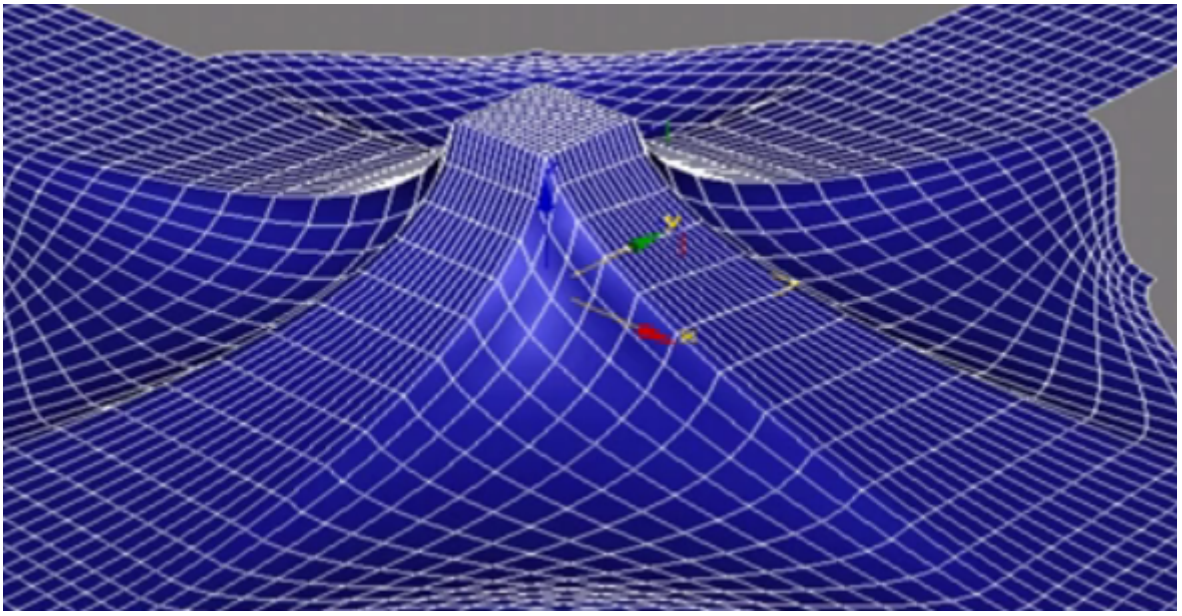

Figura 2.17: Exemplo de uma superfície modelada por *path*

A modelagem por *Path* não permite criar superfícies através da combinação de duas grades e também se torna difícil alinhar os pontos de mais de uma grade a fim de criar uma grade maior.

Capítulo 3

Desenvolvimento

Neste capítulo apresenta-se o processo do desenvolvimento do programa. Para que o desenvolvimento desse módulo fosse facilitado, o mesmo foi dividido em partes:

- Biblioteca de Peças;
- Movimentação das Peças;
- Uso de *Scripts*;
- Colisões;
- Criação de um modelo simples.

Foi necessário o estudo das ferramentas de alguns programas que foram avaliados anteriormente (JUNIOR et al., 2012). Os programas que foram utilizados nesse trabalho são o Blender (Blender Foundation, 2014), o LDraw e o Unity 3D (Unity Technologies, 2014a).

Unity

É um *framework* para o desenvolvimento de jogos que possui várias ferramentas para auxiliar os seus desenvolvedores. Dispõe de uma plataforma simples que possibilita o manuseio 2D e 3D de objetos. Também permite a criação de *scripts* (Apêndice A, seção A.1.8) para a realização de ações e controle dos objetos em cena e a importação de modelos criados em outros programas de modelagem.

LDraw

É um programa que consiste de duas partes principais, uma biblioteca desenvolvida pela própria comunidade do LDraw que contém uma grande variedade de peças Lego de vários kits disponíveis (essa biblioteca possui mais de 4800 peças e partes Lego) e um editor que permite a livre manipulação das peças contidas na biblioteca existente dentro do programa. Através de programas auxiliares, como o LDView (Travis Cobbs, 2009) e o POV-Ray (Persistence of Vision Raytracer Pty. Ltd., 2003), também é possível criar objetos e cenas 3D no LDraw.

Blender

É um programa que permite a modelagem de objetos em 3D, assim como a texturização, iluminação e outras características desses objetos. Também permite a criação de animações com objetos já existentes no programa ou modelados pelo usuário a fim de criar movimentação para esses objetos. Como o Blender é um programa de arquitetura aberta, isso permite que ele possua características como extensibilidade e interoperacionalidade. Esse programa é bastante utilizado na criação de jogos devido a sua grande capacidade de modelagem (Blender.org, 2014a).

O Blender foi utilizado para corrigir os erros das normais, suavização das bordas, criação de animações e geração das *meshs* dos objetos. No LDraw foi encontrado as peças que seriam necessárias para o programa. O Unity será o programa principal do *backend* do software desenvolvido. No Apêndice A pode ser visto um pouco mais sobre os programas utilizados.

3.1 Biblioteca de Peças

Para que fosse possível o desenvolvimento de um software que pudesse criar modelos virtuais de robôs, houve a necessidade de criar uma biblioteca de peças, compatível com o Unity, onde estariam modeladas as peças necessárias para qualquer projeto que utilize o Kit LEGO Mindstorm. A fim de encontrar exemplos de como eram modeladas as peças desse Kit e de interfaces de simuladores, foram feitas pesquisas sobre alguns simuladores disponíveis para download.

As peças encontradas possuíam algumas falhas quando exportadas para o Unity. Essas

falhas consistiam na inversão dos vetores normais de algumas faces. A fim de corrigir esse erro foi utilizado o 3D Studio Max para inverter as normais erradas. No Apêndice A, seção A.2.1, pode ser visto um pouco mais sobre essa opção no programa. Também foi utilizado o Blender para arrumar uma normal ou outra que estavam invertidas e que foram encontradas posteriormente.

Após consertadas, as peças foram exportadas para o Unity (Figuras 3.1 e 3.2), criando assim a biblioteca de peças que será utilizada no programa. As peças que, a princípio, estavam faltando no LDraw foram localizadas em outras pastas do mesmo programa e sofreram o mesmo processo de reparação para serem incorporadas à biblioteca.

Figura 3.1: Peças não elétricas que compõem a biblioteca de peças.

Figura 3.2: Peças elétricas que compõem a biblioteca de peças.

3.2 Movimentação das Peças

A movimentação das peças no simulador tem como objetivo tornar mais realista as simulações realizadas no programa. Inicialmente, para que fosse possível essa movimentação foram

criadas algumas animações básicas com as peças como a rotação de eixos, engrenagens e rodas. Essas animações foram geradas com o uso do programa Blender. No Apêndice A, seção A.4, é explicado como essas animações são criadas.

Após serem geradas, foram importadas para o Unity junto com suas respectivas peças para que fossem associadas. A visualização das animações nas peças é feita através das características de cada peça modelada, na aba *Animations* (Figura 3.3).

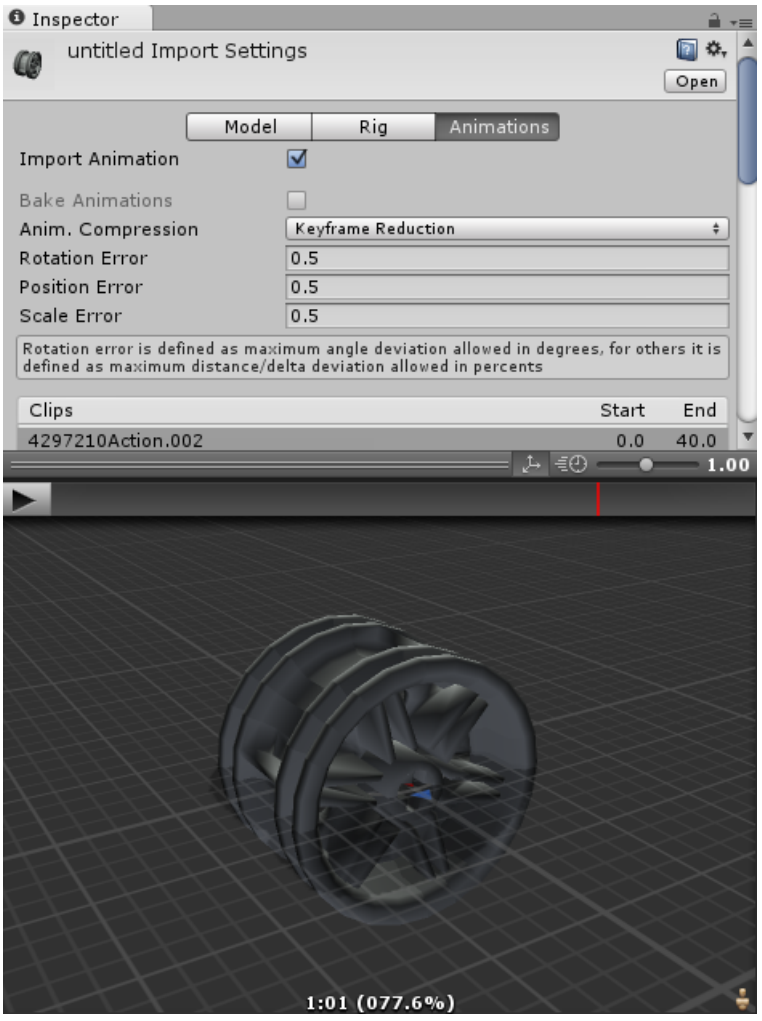

Figura 3.3: Visualização da aba *Animations* de uma peça importada.

A utilização da ferramenta de rotação do próprio Unity para simular a movimentação das peças foi uma outra opção testada. Juntando as peças e transformando-as em um único bloco, é possível rotacionar todas as peças que estão juntas ao mesmo tempo. Com isso é possível que elas se movimentem sem que seja necessário o uso das animações criadas.

3.3 Uso de *Scripts*

Para realizar o controle da movimentação de objetos, o Unity conta com o uso de *scripts*. Após discutir com o grupo qual seria a melhor linguagem para ser utilizada na criação desses *scripts* foi escolhido o C# (MICROSOFT, 2014b), uma linguagem de programação orientada a objetos desenvolvida pela Microsoft. Ela é parte da plataforma .NET e sua sintaxe foi baseada no C++, contudo possui influências de outras linguagens como o Java. Essa escolha foi feita devido ao maior conhecimento do grupo sobre essa linguagem em comparação com as outras compatíveis com o Unity.

Foi utilizado um *script* para testar o controle de uma peça dentro de um ambiente virtual gerado pelo Unity. Dentro desse arquivo foram colocados dois comandos básicos, o de rodar para a direita e o rodar para a esquerda, onde a rotação foi feita sobre o eixo X do bloco de objetos. O controle da rotação foi feito pela detecção do uso das teclas “d” e “a” do teclado: enquanto a tecla “d” estivesse pressionada a peça rotacionaria para a direita e enquanto a tecla “a” estivesse pressionada a peça rotacionaria para a esquerda. O uso das duas teclas ao mesmo tempo não movimentava a peça. O código desse *script* pode ser visto no Algoritmo 1.

Algoritmo 1 : Exemplo de *script* em C#

```
using UnityEngine;
using System.Collections;

public class teste1 : MonoBehaviour {
    private Vector3 speed;

    void Start () {
    // Comandos aqui colocados serão executados ao iniciar o ambiente
    }

    void Update () {

        if (Input.GetKey ("a")) {
            // define a direção e a velocidade com que o objeto irá
            // rotacionar
            transform.Rotate(new Vector3(-1, 0, 0),
                Time.deltaTime * 135);
            // define a direção que o objeto se movimentará
            speed = new Vector3(-1, 0, 0);
            // movimento do objeto
            rigidbody.MovePosition(rigidbody.position + (speed/5)
                 * Time.deltaTime*30);
        }
        if (Input.GetKey ("d")) {
            // define a direção e a velocidade com que o objeto irá
            // rotacionar
            transform.Rotate(new Vector3(1, 0, 0),
                Time.deltaTime * 135);
            // define a direção que o objeto se movimentará
            speed = new Vector3(1, 0, 0);
            // movimento do objeto
            rigidbody.MovePosition(rigidbody.position + (speed/5)
                 * Time.deltaTime*30);
        }
    }
}
```

3.4 Colisões

Collider é um dos componentes fornecidos pelo Unity que permite definir quais serão os limites físicos do objeto selecionado, isto é, caso esse objeto venha a entrar em contato com

outro objeto em cena, onde este segundo objeto tenha a componente também definida, ocorrerá uma colisão entre os objetos e os mesmos não se atravessarão.

Há vários tipos de *colliders* disponíveis no Unity, que são definidos como padrões, como por exemplo *box collider* e *capsule collider*, que seguem os formatos de objetos primitivos pré disponibilizados pelo programa, e o *mesh collider*, que se utiliza do formato do objeto selecionado em cena para criar um *collider* que atenda a *mesh* do objeto.

Foi utilizado os *colliders* já existentes no Unity para realizar a colisão entre as peças do robô criado. Cada peça contém uma *mesh* armazenada em seus dados que é utilizada no componente *Mesh Collider* (Figura 3.4). Esse componente seleciona a malha da peça utilizada e atribui como limites a própria peça, onde, caso essa *mesh* entre em contato com algum outro objeto do ambiente que possua um *collider*, essas peças não se atravessarão.

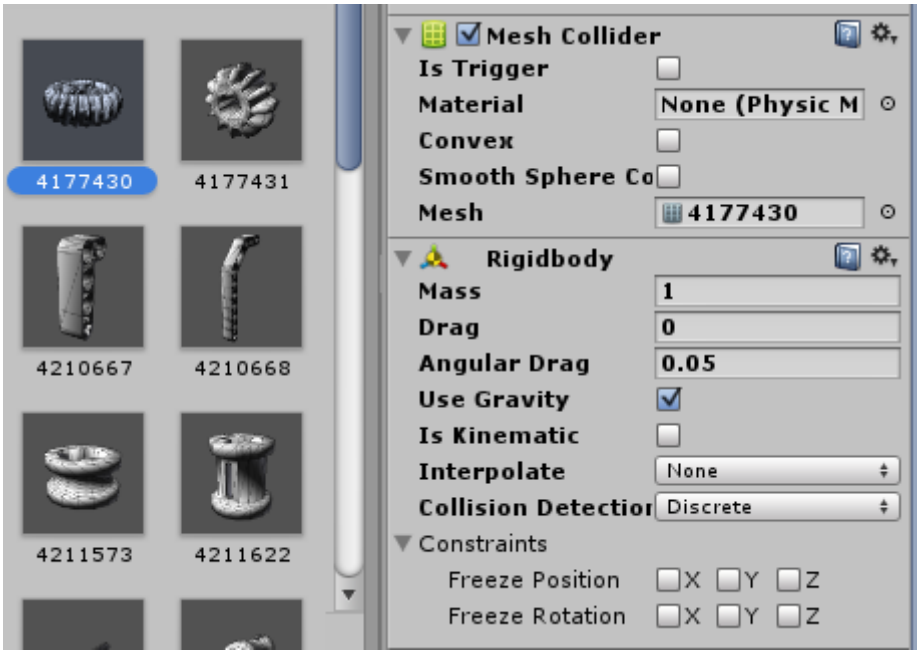

Figura 3.4: Componentes utilizados para gerar colisões no Unity.

Contudo, o *Mesh Collider* possui limitações quanto ao número de polígonos que podem compor a *mesh* que será utilizada.

Os *colliders* são atribuídos quando uma nova peça é criada no ambiente. Para que funcionem, o componente *RigidBody* também deve ser adicionado a peça. Esse componente é o responsável por atribuir os efeitos da gravidade a uma peça ou a um conjunto de peças. No *RigidBody* é escolhido se a peça sofrerá a ação da gravidade, qual será seu peso, como as colisões

serão detectadas, entre outras características. O Unity não permite a colisão entre duas *Mesh Colliders* e para tratar isso, as peças do modelo são organizadas em uma hierarquia de controle (Figura 3.5), onde quando encaixados e adicionados uns aos outros, eles se comportam apenas como um bloco, isto é, não se separam, mas cada peça mantém seu *collider* intacto (Figura 3.6).

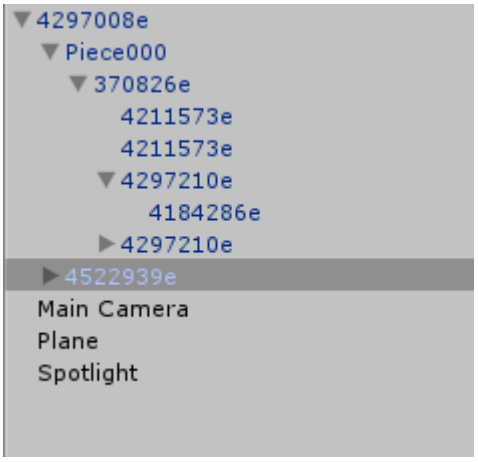

Figura 3.5: Exemplo de hierarquia de peças.

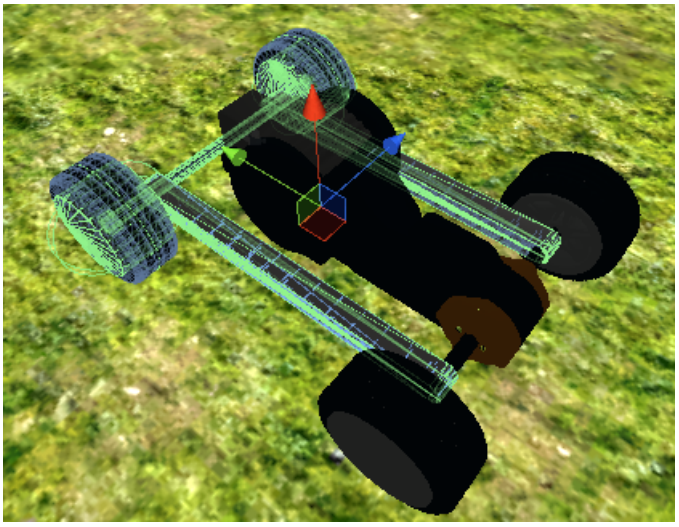

Figura 3.6: Exemplo de um bloco de peças com as *meshs* de cada peça.

3.5 Criação de um Modelo Simples

Para testar como um modelo simples de robô se comportaria após ter suas peças interligadas, foi desenvolvido um protótipo para simular um carro simples que anda para frente e para trás. O modelo a ser utilizado para realizar o teste foi obtido no site *NXT Programs* (PARKER, 2014) e

pode ser identificado como "*Mini Rover with 3-Button Remote*", onde este modelo consiste em um carrinho com dois motores e três rodas (Figura 3.7 (PARKER, 2011)).

O passo a passo para a montagem do robô foi encontrado no mesmo site e a construção do modelo se assemelha à técnica de modelagem poligonal, pois é feita a criação de um modelo com base em objetos já modelados anteriormente. Através da junção das peças já modeladas, foi possível criar um modelo semelhante ao exibido no site *NXT Programs*. Com isso foi possível a realização dos testes referentes a movimentação e colisão.

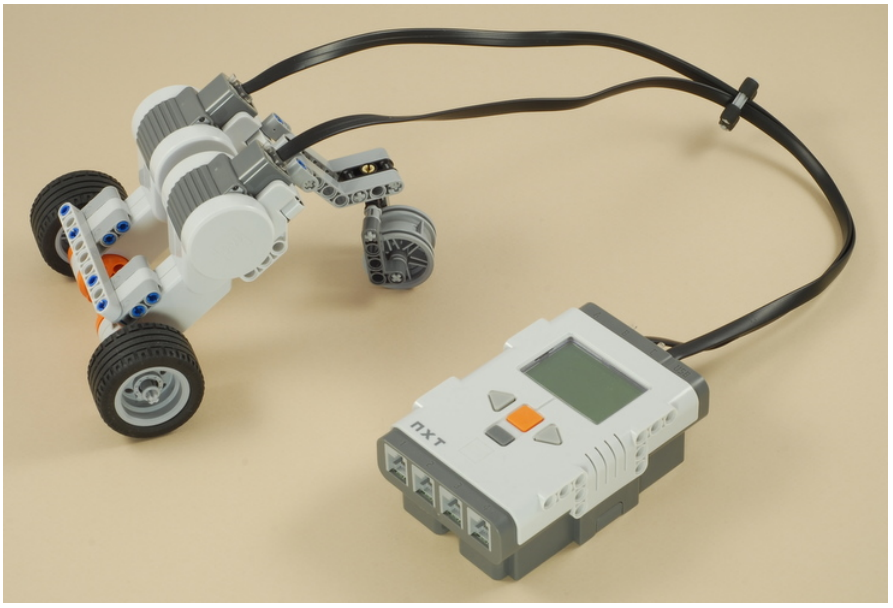

Figura 3.7: Exemplo de modelo de robô para testes em ambiente virtual.

Após gerar o modelo no Unity, foi tratada a questão de hierarquia das peças e criados os *scripts* que seriam responsáveis pelas animações das peças e da movimentação do robô no ambiente virtual. O ambiente de testes inicial consiste em um plano simples, sem nenhum relevo ou obstáculo, onde, para que o robô não atravessasse o chão foi utilizado um *Box Collider* (Figura 3.8).

Figura 3.8: Modelo final construído no Unity.

Um segundo teste foi realizado, colocando no caminho do modelo um cubo de massa significante para que não se mexesse. Esse teste foi realizado para verificar se o modelo pararia no obstáculo ou o atravessaria (Figura 3.9). Após movimentar o robô em direção do cubo, o modelo não passava por ele, apenas empurrava ou ficava apoiado no cubo.

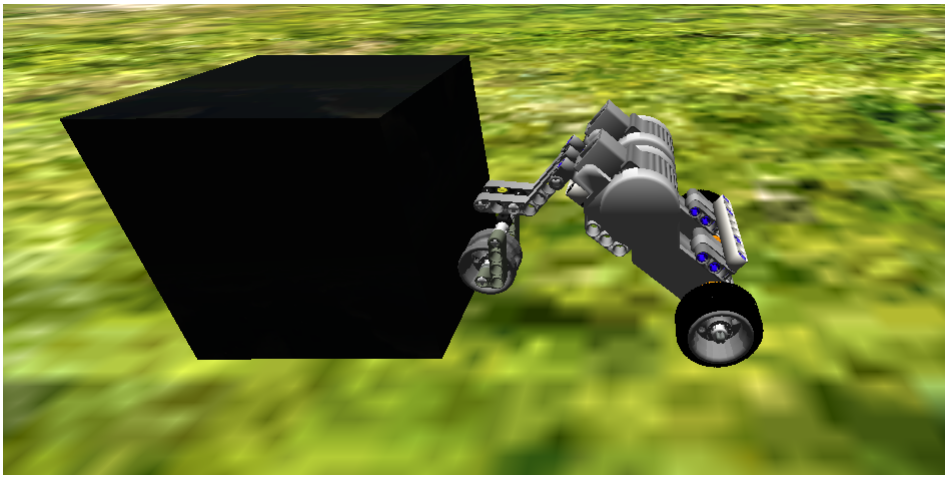

Figura 3.9: Teste de colisão entre o modelo e um objeto.

Os *scripts* utilizados para realizar a movimentação e gerar a animação do modelo criado no Unity estão descritos nos Algoritmos 2 e 3.

Algoritmo 2 : *Script* de movimentação

```
using UnityEngine;
using System.Collections;

public class teste2 : MonoBehaviour {
    private Vector3 speed;

    void Start () {
    // Comandos aqui colocados serão executados ao iniciar o ambiente
    }

    void Update () {
        if (Input.GetKey ("a")) {
            // Definição da direção que o objeto será movido
            speed = new Vector3(-1, 0, 0);

            // Movimentação do objeto da posição atual até que
            // a tecla seja liberada
            rigidbody.MovePosition(rigidbody.position + (speed/5)
                 * Time.deltaTime*30);
        }
        if (Input.GetKey ("d")) {
            // Definição da direção que o objeto será movido
            speed = new Vector3(1, 0, 0);

            // Movimentação do objeto da posição atual até que
            // a tecla sejaliberada
            rigidbody.MovePosition(rigidbody.position + (speed/5)
                * Time.deltaTime*30);
        }
    }
}
```

Algoritmo 3 : *Script* de animação

```
using UnityEngine;
using System.Collections;

public class teste1 : MonoBehaviour {
    void Start () {
    // Comandos aqui colocados serão executados ao iniciar o ambiente
    }

    void Update () {

        if (Input.GetKey ("a")) {
            // Rotação do objeto para a direção da movimentação
            transform.Rotate(new Vector3(-1, 0, 0),
                Time.deltaTime * 135);
        }
        if (Input.GetKey ("d")) {
            // Rotação do objeto para a direção da movimentação
            transform.Rotate(new Vector3(1, 0, 0),
                Time.deltaTime * 135);
        }
    }
}
```

Capítulo 4

Considerações Finais

O trabalho inicialmente pretendia desenvolver um módulo que possibilitasse a criação de um modelo de robô baseado no kit LEGO Mindstorms que pudesse ser testado em um ambiente virtual que assemelha-se a um ambiente real.

Ao final do cronograma, foi desenvolvido o *backend* do programa, que consiste na biblioteca com as peças que serão utilizadas para a criação dos modelos, os testes feitos com as animações e as movimentações das peças, tanto individuais como em blocos, a utilização de *scripts* para o controle de objetos dentro do ambiente virtual, colisões e ação gravitacional nos componentes do ambiente virtual criado e a geração de um protótipo simples de carro para testes.

Após os testes com o modelo de robô gerado, notou-se que os *Colliders* estavam agindo da forma esperada dentro do ambiente, isto é, o modelo não estava atravessando as peças colocadas como obstáculos ou afundando no terreno colocado.

4.1 Problemas e Soluções

Houve problemas com as animações exportadas do Blender e com os *Mesh Colliders* de cada uma das peças. As animações que foram exportadas do Blender não eram ativadas quando o ambiente era inicializado. Após algumas pesquisas na internet sobre o problema e em fóruns da área de computação, foi descoberto que era necessário modificar uma das características quanto ao tipo da animação, isto é, era necessário alterar o tipo da animação para a opção *Legacy*. Após realizar essa troca dentro da peça, a animação atribuída a ela era executada.

Quanto aos *Mesh Colliders* das peças do Unity, o próprio programa não permite que seja realizada a colisão entre duas *mesh collider*, com isso, um meio de evitar que as peças se se-

parassem durante a sua montagem foi a de organizar as peças em hierarquias de pais e filhos, onde o critério para decidir quem era pai e quem era filho foi a movimentação de cada peça, isto é, peças que deveriam realizar o mesmo movimento eram agrupadas juntas. Com isso, a componente *RigidBody* foi adicionada apenas na peça raiz, aplicando os efeitos de gravidade em todo o modelo, como se fosse apenas um bloco. A interface será a responsável por atribuir, para cada peça que entrar em cena, o seu respectivo *collider* e, ao final da montagem, adicionar o componente *RigidBody* a peça raiz.

Devido a esses problemas, a criação de uma interface para a criação dos modelos de robôs não foi desenvolvida, pois levou-se muito tempo para a localização e correção dos erros encontrados. As funcionalidades do *backend* do programa foram realizadas, mas o cronograma acabou em atraso.

4.2 Trabalhos Futuros

Para os próximos passos do projeto, será desenvolvida uma interface que possibilite o encaixe das peças escolhidas. Essa interface exibirá uma lista contendo todas as peças pertencentes ao kit LEGO Mindstorms. A medida que as peças são adicionadas ao ambiente virtual, também será papel da interface atribuir os componentes de *collider* e *RigidBody*, que serão responsáveis por possibilitar a aplicação da gravidade e das colisões para os testes. Após a finalização do Modelador de Robôs, ele será integrado com os outros dois módulos do programa SimBot para então ser realizados os testes finais.

Apêndice A

Materiais e Métodos

Para o desenvolvimento deste trabalho, foi necessário o estudo das ferramentas de alguns programas que foram avaliados anteriormente (JUNIOR et al., 2012). Os programas que foram utilizados nesse trabalho são o Blender (Blender Foundation, 2014), o 3D Studio Max (Autodesk Inc, 2014a), o Ldraw (LDRAW, 2003) e o Unity 3D (Unity Technologies, 2014a). Nas próximas seções serão abordadas as ferramentas que foram utilizadas para a criação da biblioteca de peças e as ferramentas foram necessárias para a criação do módulo. Mesmo com o estudo de todas essas ferramentas, o foco do trabalho consistiu no desenvolvimento do módulo para criação de robôs.

A.1 Unity 3D

Após realizar estudos sobre diversos programas que possibilitavam a criação de um ambiente onde era possível a montagem de modelos 3D, foi escolhido o Unity, já que o mesmo disponibiliza vários tipos de ferramentas que facilitam desde a criação de bibliotecas com as peças necessárias, controle de movimentos por código (através de *scripts*) até aplicação de efeitos físicos. Dentre todas as ferramentas que estão disponíveis, as utilizadas no trabalho foram estudadas e descritas nas próximas subseções para melhor entendimento das mesmas.

A.1.1 *Assets*

Assets (Unity Technologies, 2014b) é o nome dado aos arquivos que são carregados no Unity. Eles são organizados em pastas que estão localizadas na *Project View* da interface do Unity a fim de separar, por categorias, os itens lá encontrados. Também é possível criar no-

vos *Assets* no Unity, com objetos que existam no programa ou algum objeto externo que seja compatível com os tipos que o Unity aceita. Um *Asset* pode ser apenas um objeto ou um grupo de objetos, dependendo para qual finalidade ele foi criado, além de ser possível a inserção de trechos de códigos.

Existem no Unity três tipos de *Assets*: arquivos de som, arquivos de textura e arquivos de malha e animações.

A.1.2 Arquivo de Malha e Animações

O Unity possibilita a importação das malhas, um arquivo que contém as características do objeto importado (Figura A.1), e das animações de um arquivo externo (Unity Technologies, 2014c), desde que a aplicação onde foi criado o arquivo seja suportada pelo Unity. Contudo, não é necessariamente obrigatório que as malhas sejam importadas com as animações, elas podem ser importadas todas juntas em um arquivo, ou separadas, onde cada animação será guardada em um arquivo separado.

Figura A.1: Exemplo de arquivo de malha.

Uma vez importadas, é possível arrastar os arquivos para a *Scene* (área de visualização). Também é possível adicionar componentes ao *Asset* importado. As malhas importadas terão um material padrão atribuído a ela, onde as características do material já estão especificadas no programa.

A.1.3 Arquivo de Textura

Todos os tipos de formatos de imagens são suportados pelo Unity, isso facilita o trabalho com arquivos simples de textura (Figura A.2) (Unity Technologies, 2014c), possibilitando um melhor manuseio das texturas importadas e uma melhor experiência com o programa.

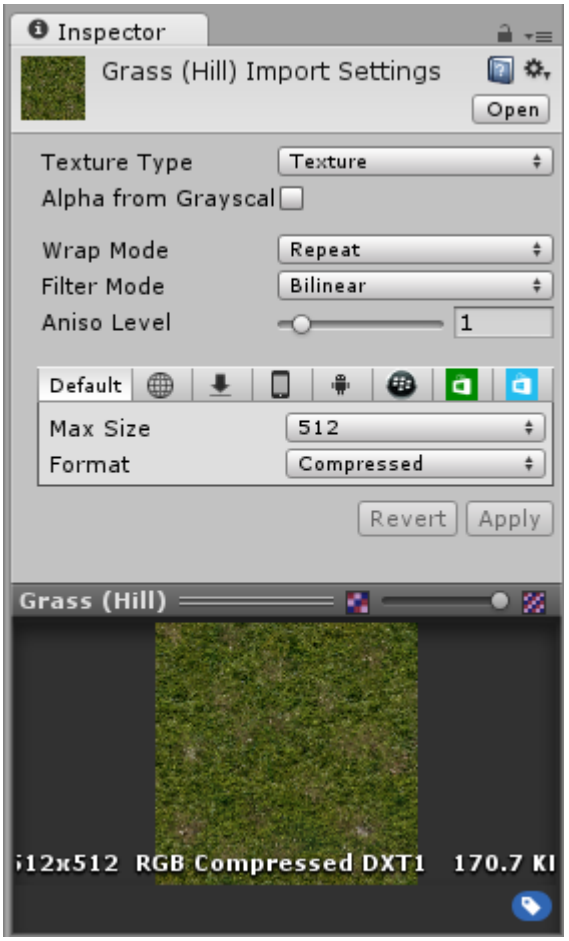

Figura A.2: Exemplo de arquivo de textura.

As dimensões dos arquivos de textura criados devem ter suas dimensões na ordem de dois, ou seja, a dimensão do arquivo deve ser quadrada e ser um número inteiro elevado a dois. Para que elas apareçam é necessário apenas arrastá-las para a pasta dos *Assets*, com isso automaticamente a textura aparecerá na *Project View*.

A.1.4 Arquivo de Som

São suportados dois tipos de arquivos de áudio pelo Unity (Unity Technologies, 2014c): *Uncompressed Audio* e *Ogg Vorbis*. Ao importar um arquivo de áudio para o Unity, automaticamente ele será convertido para um dos seguintes formatos: .AIFF e .WAV (são os melhores tipos de formatos para sons curtos de efeitos), .MP3 e .OGG (são os melhores tipos de formatos para músicas longas). Caso o arquivo importado não esteja compactado como *Ogg Vorbis*, existem várias opções de importação no menu do *Audio Clip*. É possível comprimir o arquivo,

forçando-o a ser Mono ou Stereo, dependendo da necessidade.

A.1.5 MonoDevelop

O MonoDevelop (Unity Technologies, 2014d) é uma IDE suportada pelo Unity. Essa IDE possibilita a criação e modificação de códigos, realização de uma execução detalhada, entre outras tarefas de administração de projetos.

Por padrão, esse programa é instalado junto com o Unity. Para que seja possível a utilização da execução detalhada, é necessário que seja ativada, primeiramente, no menu do programa e depois sincronizar os projetos do Unity com os do MonoDevelop. Isso possibilita carregar os *scripts* (seção A.1.8) do Unity dentro do MonoDevelop, na área de exibição de projetos. Durante a execução detalhada, é possível colocar marcadores onde o programa deve parar, e a partir deles, fazer passo a passo, olhando o que cada variável guarda e como estão se comportando. Essas ações, além de outras como entrar em métodos para ver como funcionam, estão disponíveis em um menu que é habilitado ao iniciar esse tipo de execução.

A.1.6 Formatos 3D

O Unity aceita dois tipos principais de arquivos, os arquivos 3D exportados (como os formatos .FBX e .OBJ) e arquivos 3D de aplicações (como os formatos .MAX ou .BLENDER, pertencentes, respectivamente, aos programas 3D Studio Max e Blender)(Unity Technologies, 2014e).

O Unity consegue ler arquivos com os formatos .FBX, .DAE, .3DS, .DXF e .OBJ. Por utilizar esses tipos de formatos, o Unity consegue algumas vantagens como:

- Exportar apenas os arquivos que serão necessários;
- O tamanho dos arquivos geralmente é pequeno;
- Pode suportar outros pacotes de arquivos 3D de outros programas que não são diretamente compatíveis com o Unity.

Contudo, a sequência de passos feitos para a prototipação e para a interação pode se tornar lenta. Também há chances de perder o controle das versões entre a origem e o arquivo ex-

portado, isto é, as versões entre o arquivo original e o arquivo que foi exportado, podem ficar desordenadas ou serem perdidas.

Além do Blender e do 3D Studio Max, o Unity também suporta arquivos exportados dos programas Maya, Cinema4D, Modo, Lightwave e Cheetah3D . Isso traz grandes vantagens como a compatibilidade entre os arquivos gerados por esses programas o que deixa, inicialmente, mais simples a integração dos mesmos.

A.1.7 Materiais e *Shaders*

Existe uma grande relação entre os materiais e os *shaders* no Unity (Unity Technologies, 2014f). Nos *Shaders* estão contidos os códigos que definem os tipos de propriedades e os *Assets* que serão utilizados, ou seja, os *shaders* são responsáveis pelos métodos de renderização de um objeto, isso inclui o uso de diferentes métodos dependendo da placa gráfica do usuário final, algumas propriedades de textura, opções de cores e números podem ser atribuídas dentro do material. Já os materiais permitem que o usuário escolha quais cores e texturas serão atribuídas ao objeto e quais os *Assets* que serão necessários para a criação do objeto.

Na criação de um material novo, o *shader* que for escolhido será responsável por ditar as propriedades que poderão ser alteradas. Essas propriedades podem ser cor, textura, vetores, números. No caso da textura, é possível aplica-la de dois modos: distribuir a textura ao longo dos eixos do objeto ou aplicar a textura em um ponto e ir espalhando-a a partir do ponto inicial.

Por padrão, o Unity traz consigo, em sua instalação, um conjunto de *shaders* com mais de oitenta tipos diferentes, divididos em quatro principais categorias:

- Normal: abrange o grupo de objetos opacos;
- Transparente: grupo de objetos parcialmente transparentes. Cabe a textura definir o Nível de transparência;
- Auto Iluminado: grupo de objetos que possuem fontes de luz;
- Reflexiva: objetos opacos que refletem a luz em um ambiente.

Além das categorias principais, os *shaders* também estão divididos em algumas categorias mais específicas para modelagem como:

- FX: efeitos de luz e água;
- GUI (*Graphical User Interface*): display de interface gráfica do usuário;
- Natureza: terrenos e árvores;
- Partículas: sistema de efeitos de partículas;
- Renderização FX;
- Toon: renderização animada.

A.1.8 Scripting

Objetos no Unity são controlados pelos componentes neles inseridos, contudo, o usuário não está limitado apenas aos componentes nativos do programa. O Unity possibilita a criação de *scripts* (Unity Technologies, 2014g) que possibilitam um controle personalizado dos objetos em que eles estão colocados.

São aceitos, dentro do Unity, três linguagens de programação: o C#; o UnityScript, uma linguagem desenvolvida especialmente para o Unity, baseada no JavaScript; o Boo, uma linguagem .NET com uma sintaxe similar ao Python. O Unity também suporta outras linguagens .NET, necessitando apenas que exista uma DLL (*Dinamic-Link Library*), no caso do Windows, ou uma *Shared Library*, no caso do Unix, compatível para que a compilação possa ser feita.

O Unity facilita a criação de *Script*, necessitando apenas estar na pasta desejada e clicando em Criar Script. É aconselhável que seja dado o nome do arquivo na hora de sua criação, pois esse nome será utilizado no texto dentro do arquivo do *script*. Para editar este arquivo, basta clicar duas vezes nele. Por padrão, o Unity irá utilizar o MonoDevelop (seção A.1.5), mas caso o usuário queira usar outra ferramenta externa para edição e criação de códigos basta mudar nas preferências do Unity para o editor escolhido.

Scripts fazem a conexão com as ferramentas internas do Unity através da implementação de classes que derivam da `MonoBehaviour`. A `MonoBehaviour` é uma base para a criação de outras classes que interagirão com objetos no Unity, onde toda a vez que é atribuído um *script*

a um objeto, uma nova instância dessa classe base é criada. Para que a classe criada funcione, é necessário que o nome da classe seja o mesmo nome do arquivo.

Ao criar o *script*, são gerados duas funções dentro dele, uma função *Update* e uma *Start*. Na função *Update* são colocados códigos referentes a movimentação, gatilhos e respostas a ações, basicamente as funções que o objeto deve realizar durante a execução do programa. Na função *Start* são colocados os comandos que o objeto deve fazer ao iniciar a execução. Não é necessária a criação de um construtor para o objeto, pois o próprio editor se encarrega da criação do objeto. *Scripts* só terão seus códigos ativados a partir do momento que for inserido como componente em um objeto.

Variáveis criadas em um *script* podem ser alteradas tanto dentro do *script* quanto fora, na aba do próprio componente no objeto. Uma vez alterada pela aba, a variável receberá sempre esse valor, até que seja mudada para outro. Em C# e Boo, para que as variáveis possam ser visualizadas na aba do componente elas devem ser, necessariamente, públicas. Já no UnityScript, variáveis são, por padrão, públicas. O Unity permite que os valores das variáveis sejam mudadas durante a execução do programa, onde as mudanças são aplicadas sem que seja necessário reiniciar a execução. Ao final da execução, o valor das variáveis voltará ao valor que estava antes do início da execução, isso permite que sejam feitas mudanças nas variáveis sem que haja a possibilidade de causar problemas permanentes.

A.1.9 Controlando Objetos

Como citado antes, é possível fazer o controle de componentes (Unity Technologies, 2014h) através de suas abas, contudo também é possível chamá-los e modificar dentro de *scripts*. O Unity possibilita o acesso a componentes existentes de um modo facilitado. Como os componentes são classes instanciadas dentro do objeto utilizado, para que sejam modificadas dentro do *script*, basta declarar uma variável do tipo do componente que se deseja modificar e atribuir a ela o componente que está no objeto. Isso fará com que os dados que estão especificados naquele componente sejam passados para a variável. Caso algum componente não esteja presente no objeto chamado, será retornado um valor nulo, informando que o componente não está no objeto que se está trabalhando.

Também é possível acessar outros objetos que estejam instanciados. Isso é possível de-

clarando um tipo *GameObject* com o nome do objeto que se deseja acessar. Após feito isso é possível modificar os valores desse objeto e utilizar as funções que estão descritas dentro dele. Caso seja declarada uma variável de um componente pertencente a esse objeto, é possível acessar esse componente diretamente.

No caso de vários objetos do mesmo tipo instanciados na cena, é possível localizá-los através de funções já existentes no Unity. Isso faz com que seja mais fácil manipular um conjunto de objetos iguais, mas sem que cada um perca sua individualidade.

É possível realizar a busca por um objeto na cena através de seu nome ou sua *tag*. Isso é feito selecionando a variável em que o objeto foi salvo e passando como parâmetro para a função de busca o nome ou a *tag* do objeto necessário. Caso queira procurar um objeto específico dentro do conjunto de objetos, basta utilizar funções que buscam objetos da classe instanciada através do nome designado a ele.

A.1.10 Sistema de Animação

O Unity possui um sistema de animação sofisticado integrado chamado Mecanim (Unity Technologies, 2014i), que possibilita:

- Uma configuração e um fluxo mais fácil de animação para personagens humanóides;
- A utilização de animações de um personagem em outro (herança);
- Um método simplificado para alinhar clipes de animações;
- A pré-visualização da sequência, transições e interações entre as animações, o que permite que os animadores possam trabalhar de forma independente dos programadores e protótipos;
- Manipulação de interações complexas entre animações a partir de ferramentas visuais;
- Realização de diferentes animações para cada parte do corpo de um objeto.

O fluxo de trabalho no Mecanim pode ser dividido em três grandes partes:

- Preparação e importação do *Assets*: fase onde é feita a criação das animações com o uso de uma ferramenta 3D externa como o 3D Studio Max ou o Maya;

- Configuração da característica do personagem, que pode ser dividida em Humanóide e Genérico:
  - Humanóide: para atribuir as características humanóides a objetos, o Mecanim possui um fluxo específico com um GUI estendido e possibilidade de redirecionamento de animação, que envolve a criação e configuração do avatar do personagem e do comportamento e características dos músculos;
  - Genérico: é uma configuração padrão utilizada para qualquer objeto, mas, ao contrário da configuração de humanóides, não é possível realizar um redirecionamento dessa configuração para outro objeto.
- Trazer os personagens a vida: realização da configuração das animações que serão realizadas pelo objeto, bem como as interações entre as animações. Isso envolve máquinas de estados e árvores, atribuindo os parâmetros de animação e possibilitando o controle delas através de códigos.

O Unity suporta alguns formatos quando se fala em importação de animação (Unity Technologies, 2014j) como .MB ou .MA (Maya), .C4D (Cinema 4D) ou .FBX genéricos. Para realizar a exportação do arquivo de animação, basta arrastá-lo para a pasta de *Assets*. Após isso o usuário pode manipular as características da animação através de uma aba que aparecerá quando a animação for selecionada no *Project View*.

Através de *scripts*, é possível fazer o controle de animações (Unity Technologies, 2014k) de um objeto, definindo o tempo que a animação irá ocorrer, qual animação usar, como se comportará, entre outras características. O controle das animações pode ser feito na função *Start* ou *Update*, onde serão selecionadas animações já existentes no Unity ou alguma animação importada pelo usuário, onde, depois de instanciada, poderá ser definida a ação. O uso de uma animação também pode ser feito através de uma função do Unity chamada *CrossFade*, onde é escrita a ação que o objeto deve realizar dentro, como parâmetro, e caso a animação exista, ela será executada (*walk*, por exemplo).

Também é permitido criar um conjunto de animações, onde cada animação aja em cima de apenas uma parte do objeto, possibilitando que várias animações sejam executadas juntas: basta

adicionar a ação através da função *AddMixingTransform*. O mesmo se aplica para a realização de movimentos aditivos, que diminuem o número de animações e a quantidade de trabalho fazendo com que o processo de criação de uma animação seja mais simples e fácil.

A.2 3D Studio Max

Devido a necessidade de modelar as peças que foram utilizadas no programa ou consertar as peças com problemas (faces do objeto se tornavam invisíveis), foram pesquisados alguns programas que disponibilizassem ferramentas para manipulação e modelagem. Ao final da pesquisa, foi escolhido o 3D Studio Max (Autodesk Inc, 2014a). A escolha desse programa foi feita devido a interface mais organizada e a facilidade na utilização de seus recursos. O programa foi utilizado para a reparação das faces invisíveis das peças importadas (Autodesk Inc, 2014b).

A.2.1 Ajustando Normais

As normais devem estar ajustadas para que o objeto modelado possa ser renderizado corretamente. Uma normal é um vetor que indica a direção que a face ou objeto será observado, para onde a luz irá refletir e tornar o lado visível. Caso as normais de um objeto estejam invertidas no momento da renderização do mesmo, essas faces não existirão, apresentando assim apenas um lugar invisível em vez de uma face sólida.

Quando um objeto é criado, as normais de cada uma das faces são geradas automaticamente, mas, às vezes, em sua criação ou em uma exportação, o programa que recebe ou cria o objeto pode inverter algumas normais.

A visualisação das normais de um objeto é feita através de uma opção localizada no menu lateral onde estão contidas as propriedades do objeto selecionado. Após ativar essa opção, é possível escolher o modo de visualização de cada uma das faces ou vértices. Com as normais já habilitadas o usuário pode realizar algumas ações sobre elas:

- Unificação: ao selecionar as faces e vértices de um objeto, caso elas estejam com as normais desorganizadas, isto é, algumas para o lado de fora e outras para o lado de dentro do objeto, é possível unificá-las, ou seja, o programa selecionará as normais marcadas e as colocará na mesma direção que as outras normais próximas, essa ação faz com que as

normais próximas sejam passadas para uma mesma direção, contudo unificar as normais não garante que todas sejam invertidas ou consertadas.

- Inversão: consiste em selecionar uma face ou um vértice ou um conjunto deles que necessitam de alteração e invertê-las. Essa função apenas inverte a normal que está selecionada, deixando a cargo do usuário verificar se a inversão foi feita certa ou não. A opção de inversão pode ser utilizada após a unificação, a fim de corrigir aquelas normais que não foram invertidas pelo processo.

A.3 LDraw

LDraw é um programa que consiste de duas partes principais, uma biblioteca desenvolvida pela própria comunidade do LDraw que contém uma grande variedade de peças Lego de vários kits disponíveis (essa biblioteca possui mais de 4800 peças e partes Lego) e um editor que permite a livre manipulação das peças contidas na biblioteca existente dentro do programa. Através de programas auxiliares como o LDView (Travis Cobbs, 2009) e o POV-Ray (Persistence of Vision Raytracer Pty. Ltd., 2003) também é possível criar objetos e cenas 3D no LDraw.

O LDView é um programa desenvolvido pela comunidade LDraw que faz a conversão dos arquivos salvos no LDraw em arquivos que são aceitos pelo POV-Ray, um programa externo que possibilita a criação de imagens 3D de qualquer objeto ou cena importada para ele.

Utilizando esse programa, é possível criar construções baseadas em peças Lego para fins educativos, empresariais ou por diversão. Também pode ser usado para recriação ou criação de cenários que possam ser usados como base para o desenvolvimento de ambientes de jogos, apresentações acadêmicas ou para alguma atividade cinematográfica.

As peças contidas nesse programa foram utilizadas como modelos para as peças do programa proposto neste trabalho, pois as peças são as mesmas encontradas no kit LEGO Mindstorms.

A.4 Blender

Blender é um programa que permite a modelagem de objetos em 3D, assim como a texturização, iluminação e outras características desses objetos. Também permite a criação de

animações com objetos já existentes no programa ou modelados pelo usuário a fim de criar movimentação para esses objetos. Como o Blender é um programa de arquitetura aberta, isso permite que ele possua características como extensibilidade e interoperacionalidade. Esse programa é bastante utilizado na criação de jogos devido a sua grande capacidade de modelagem (Blender.org, 2014a).

A animação de objetos no Blender é feita a partir do menu *Animation* contido em sua interface. São gravados estados específicos da peça, que são usados como marcos para que o programa saiba qual a movimentação que o objeto deve seguir. Com isso o Blender faz a movimentação do objeto entre os marcos definidos, criando a animação para ele. É possível criar várias animações para um mesmo objeto para que elas sejam salvas junto dele ou separadas (Blender.org, 2014b).

Ao salvar um objeto modelado no Blender, automaticamente ele suaviza as bordas desse objeto, caso ele tenha sido exportado para o programa, e gera uma *mesh*, que pode ser utiliza para delimitar o objeto em um espaço virtual. Esse programa também conta com uma opção para a inversão da normal das faces do objeto, necessitando apenas selecionar uma das faces e dar um *flip* no vetor. A inversão de normais é feita face a face ou com um conjunto de faces.

Referências Bibliográficas

AForge.NET. *AForge.NET*. 2008. Acessado em 02 jun. 2014. Disponível em: <http://www.aforgenet.com/>.

ANGHELESCU, M. *The base head mesh*. 2008. Acessado em 21 nov. 2014. Disponível em: <http://www.anghelescu.net/tutorials/Russell_Crowe_3d/Russell_Crowe_page3.html>.

Association for Computing Machinery. *Virtual Reality Modeling Language (VRML)*. 2014. Acessado em 08 out. 2014. Disponível em: <http://www.acm.org/tsc/vrml.html>.

Autodesk Inc. *Criar um sólido ou superfície ao efetuar a extrusão*. 2011. Acessado em 21 nov. 2014. Disponível em: <http://exchange.autodesk.com/autocadmechanical/ptb/online-help/AMECH_PP/2012/PTB/pages/WS1a9193826455f5ffa23ce210c4a30acaf-6877.htm>.

Autodesk Inc. *3D Studio Max Overview*. 2014. Acessado em 02 jun. 2014. Disponível em: <http://www.autodesk.com/products/3ds-max/overview>.

Autodesk Inc. *Getting Started*: Adjusting normals and smoothing. 2014. Acessado em 02 jun. 2014. Disponível em: <http://knowledge.autodesk.com/support/3ds-max/getting-started/caas/documentation/3DSMAX/14/ENU/Autodesk-3ds-Max-and-3ds-Max-Design-2012-Help/files/GUID-FF29D920-7338-4614-8A72-B5C94D47889-155-htm.html>.

Blender Foundation. *Blender*. 2014. Acessado em 02 jun. 2014. Disponível em: <http://www.blender.org/>.

Blender.org. *Blender Animation*. 2014. Acessado em 23 jun. 2014. Disponível em: <http://wiki.blender.org/index.php/Doc:2.6/Manual/Animation>.

Blender.org. *Blender Introduction*. 2014. Acessado em 23 jun. 2014. Disponível em: <http://wiki.blender.org/index.php/Doc:2.6/Manual/Introduction>.

CONCI, A. *Modelagem Geométrica e Sweeping*. 2014. Acessado em 21 nov. 2014. Disponível em: <http://www2.ic.uff.br/˜ aconci/sweeping.html>.

COSTA, P. J. C. G. da. *SimTwo*. 2012. Acessado em 27 jun. 2014. Disponível em: <http://paginas.fe.up.pt/ paco/wiki/index.php?n=Main.SimTwo>.

CRAIGHEAD, J.; BURKE, J.; MURPHY, R. *Using the Unity Game Engine to Develop SARGE: A Case Study*: International conference on intelligent robots and systems. 2008. Acessado em 08 out. 2014. Disponível em: <http://www.robot.uji.es/research/events/iros08/contributions/craighead.pdf>.

CRUZ, A. J. G. *Implementacion de Curvas y Superficies con NURBS*. 2007. Acessado em 21 nov. 2014. Disponível em: <http://aide-comp-graf.blogspot.com.br/2007/04/implementacion-de-curvas-y-superficies.html>.

Cyberbotics Ltd. *Webots Overview*. 2014. Acessado em 02 jun. 2014. Disponível em: <http://www.cyberbotics.com/overview>.

DATUOPINION. *Opiniones de Curva de Bézier*. 2011. Acessado em 21 nov. 2014. Disponível em: <http://www.datuopinion.com/curva-de-bezier>.

FONSECA, G. L. *Modelagem Tridimensional do Campus Pampulha da UFMG - Uma Proposta Exploratória Utilizando a Ferramenta Google Sketchup*: UFMG - Instituto de Geociências Departamento de Cartografia. 2007. Acessado em 20 nov. 2014. Disponível em: <http://www.csr.ufmg.br/geoprocessamento/publicacoes/GizelleLira.pdf>.

GONÇALVES, J. et al. Realistic simulation of a Lego Mindstorms NXT based robot. In: *18th IEEE International Conference on Control Applications*. [S.l.: s.n.], 2009. p. 1242 to 1247.

HOUNSELL, M. da S.; REDEL, R. Implementação de simuladores de robôs com o uso da tecnologia de realidade virtual. In: *IV Congresso Brasileiro de Computação – CBComp*. [S.l.: s.n.], 2014. p. 396 – 401.

HUGUES, L.; BREDECHE, N. *Simbad*. 2011. Acessado em 02 jun. 2014. Disponível em: <http://simbad.sourceforge.net/>.

JUNIOR, R. P. et al. Estudo sobre simuladores de robótica. In: *ECA - Encontro em Computação Aplicada*. Cascavel: [s.n.], 2012. p. 11 – 12.

LDRAW. *Getting Started*. 2003. Acessado em 02 jun. 2014. Disponível em: <http://www.ldraw.org/article/104.html>.

LEGO Group. *Lego Mindstorms User Guide*. 2009. Acessado em 21 nov. 2014.

MICROSOFT. *Microsoft Robotics Developer Studio Overview*. 2014. Acessado em 02 jun. 2014. Disponível em: <http://msdn.microsoft.com/en-us/library/bb483024.aspx>.

MICROSOFT. *Visual C# Language*. 2014. Acessado em 08 out. 2014. Disponível em: <http://msdn.microsoft.com/en-us/library/aa287558(v=vs.71).aspx>.

PARKER, D. *Mini Rover with 3-Button Remote*. 2011. Acessado em 21 nov. 2014. Disponível em: <http://www.nxtprograms.com/NXT2/mini_rover/index.html>.

PARKER, D. *nxtprograms.com*. 2014. Acessado em 08 out. 2014. Disponível em: <http://www.nxtprograms.com/index1.html>.

Persistence of Vision Raytracer Pty. Ltd. *POV-Ray*. 2003. Acessado em 02 jun. 2014. Disponível em: <http://www.povray.org/>.

PINHO, M. S. *Modelagem de Sólidos*. 2001. Acessado em 21 nov. 2014. Disponível em: <http://www.inf.pucrs.br/ pinho/CG/Aulas/Modelagem/Modelagem3D.htm>.

QUACO, A. *20 Modelagem Path*. 2013. Acessado em 21 nov. 2014. Disponível em: <https://www.youtube.com/watch?v=tYszTsXDXlI>.

Robomatter Inc. *ROBOTC - a C Programming Language for Robotics*. 2014. Acessado em 20 nov. 2014. Disponível em: <http://www.robotc.net/>.

SILVA, C. M. da. *Técnicas em Modelagem de Objetos 3D*. Trabalho de Conclusão de Curso — Universidade Estadual do Oeste do Paraná, Cascavel, 2000.

SILVA, R. C. *VIRTUAL SUBSTATION - um sistema de realidade virtual para treinamento de operadores de subestações elétricas*. Dissertação (Mestrado) — Universidade Federal de Uberlândia, Uberlândia, 2012.

SMITH, R. *Open Dynamics Engine*. 2008. Acessado em 20 nov. 2014. Disponível em: <http://www.ode.org/>.

Tonka 3D. *Introdução a Modelagem 3D*. 2013. Acessado em 08 out. 2014. Disponível em: <http://tonka3d.com.br/blog/introducao-a-modelagem-3d>.

Travis Cobbs. *LDView*. 2009. Acessado em 02 jun. 2014. Disponível em: <http://ldview.sourceforge.net>.

Unity Technologies. *Unity - Game Engine, Tools and Multiplatform*. 2014. Acessado em 02 jun. 2014. Disponível em: <http://unity3d.com>.

Unity Technologies. *Unity Manual*: Asset import and creation. 2014. Acessado em 02 jun. 2014. Disponível em: <http://docs.unity3d.com/Manual/AssetImportandCreation.html>.

Unity Technologies. *Unity Manual*: Importing assets. 2014. Acessado em 02 jun. 2014. Disponível em: <http://docs.unity3d.com/Manual/ImportingAssets.html>.

Unity Technologies. *Unity Manual*: Monodevelop. 2014. Acessado em 02 jun. 2014. Disponível em: <http://docs.unity3d.com/Manual/MonoDevelop.html>.

Unity Technologies. *Unity Manual*: 3d formats. 2014. Acessado em 02 jun. 2014. Disponível em: <http://docs.unity3d.com/Manual/3D-formats.html>.

Unity Technologies. *Unity Manual*: Materials and shaders. 2014. Acessado em 02 jun. 2014. Disponível em: <http://docs.unity3d.com/Manual/Materials.html>.

Unity Technologies. *Unity Manual*: Creating and using scripts. 2014. Acessado em 02 jun. 2014. Disponível em: <http://docs.unity3d.com/Manual/CreatingAndUsingScripts.html>.

Unity Technologies. *Unity Manual*: Controlling gameobjects using components. 2014. Acessado em 02 jun. 2014. Disponível em: <http://docs.unity3d.com/Manual/ControllingGameObjectsComponents.html>.

Unity Technologies. *Unity Manual*: Mecanim animation system. 2014. Acessado em 02 jun. 2014. Disponível em: <http://docs.unity3d.com/Manual/MecanimAnimationSystem.html>.

Unity Technologies. *Unity Manual*: Importing animations. 2014. Acessado em 02 jun. 2014. Disponível em: <http://docs.unity3d.com/Manual/AnimationsImport.html>.

Unity Technologies. *Unity Manual*: Animation scripting (legacy). 2014. Acessado em 02 jun. 2014. Disponível em: <http://docs.unity3d.com/Manual/AnimationScripting.html>.

Wikimedia Foundation. *Catmull Clark subdivision surface*. 2014. Acessado em 21 nov. 2014. Disponível em: <http://en.wikipedia.org/wiki/Catmull%E2%80%93Clark_subdivision_surface>.

Wikimedia Foundation. *NURBS*. 2014. Acessado em 21 nov. 2014. Disponível em: <http://es.wikipedia.org/wiki/NURBS>.

YAMAMOTO, M. M. et al. Um simulador dinâmico para mini-robôs móveis com modelagem de colisões. In: *VI Simpósio Brasileiro de Automação Inteligente*. [S.l.: s.n.], 2003.