USP

Campus de São Carlos

PROTOTIPAÇÃO VIRTUAL DE PLATAFORMA AGRÍCOLA MÓVEL

HELENO MURILO CAMPEÃO VALE

Prof. Dr. Arthur José Vieira Porto

UNIVERSIDADE DE SÃO PAULO

ESCOLA DE ENGENHARIA DE SÃO CARLOS

Heleno Murilo Campeão Vale

Serviço de Pós-Graduação EESC/USP

EXEMPLAR REVISADO

Data de entrada no Serviço 30/11/05

Ass.: Cesar

PROTOTIPAÇÃO VIRTUAL DE PLATAFORMA AGRÍCOLA MÓVEL

Dissertação apresentada à Escola de Engenharia de São Carlos da Universidade de São Paulo, como parte dos requisitos para obtenção do título de Mestre em Engenharia Mecânica.

Orientador: Prof. Dr. Arthur José Vieira Porto

DEDALUS - Acervo - EESC

31100053614

São Carlos

2005

EESC-USP Biblioteca CAMPUS DE S. CARLOS

Class. TESE-EESC
Cutt. 6137
Tombo T309/05
Sysno 1488726

Ficha catalográfica preparada pela Seção de Tratamento
da Informação do Serviço de Biblioteca – EESC/USP

V149p

Vale, Heleno Murilo Campeão
Prototipação virtual de plataforma agrícola móvel / Heleno Murilo Campeão Vale. -- São Carlos, 2005.

Dissertação (Mestrado) -- Escola de Engenharia de São Carlos-Universidade de São Paulo, 2005.
Área: Engenharia Mecânica.
Orientador: Prof. Dr. Arthur José Vieira Porto.

1. Prototipação virtual. 2. Realidade virtual. 3. Ferramentas de desenvolvimento virtual. I. Título.

FOLHA DE JULGAMENTO

Candidato: Bacharel **HELENO MURILO CAMPEÃO VALE**

Dissertação defendida e julgada em 25-08-2005 perante a Comissão Julgadora:

Prof. Titular **ARTHUR JOSÉ VIEIRA PORTO (Orientador)** — Aprovado
(Escola de Engenharia de São Carlos/USP)

Prof. Dr. **RICARDO YASSUSHI INAMASU** — Aprovado
(Empresa Brasileira de Pesquisa Agropecuária/EMBRAPA)

Profa. Dra. **REGINA BORGES DE OLIVEIRA** — Aprovado
(Universidade Federal de São Carlos/UFSCar)

Prof. Associado **JONAS DE CARVALHO**
Coordenador do Programa de Pós-Graduação em Engenharia Mecânica

Profa. Titular **MARIA DO CARMO CALIJURI**
Presidente da Comissão de Pós-Graduação

AGRADECIMENTOS

Ao professor Arthur José Vieira Porto por toda a atenção, paciência e orientações concedidas à mim.

À minha família por toda a confiança depositada em meu trabalho.

À Professora Dr^{a}. Jandira Guenka Palma.

Ao Professor Dr. Ricardo Yassushi Inamasu.

Aos colegas do Laboratório de Simulação: Andrea, Beth, Hilano, Perfetto, Mamoru, Rafael de Souza e Santana, pelas sugestões e apoio.

Aos funcionários da Secretaria de Pós-Graduação: Ana Paula, Denise e Ricardo, agradeço pelo apoio burocrático.

RESUMO

VALE, H.M.C. (2005). Prototipação Virtual de Plataforma Agrícola Móvel. Dissertação (Mestrado) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2005.

A Realidade Virtual (RV) tem como principal área de aplicação o entretenimento. Porém, nos últimos anos, devido ao desenvolvimento acelerado do *hardware* de computadores, a RV se tornou mais acessível, sendo amplamente utilizada, também, nas áreas acadêmica e industrial. Uma parte da RV que, a cada dia, se torna mais importante para as áreas da engenharia é a Prototipação Virtual (PV). Normalmente, a prototipação física de um produto pode ser dispendiosa e inviável para determinados projetos. Nesses casos, a PV pode ser utilizada para evitar gastos desnecessários, alcançando resultados satisfatórios. Neste trabalho são descritos os passos referentes à implementação de um protótipo virtual, baseado no anteprojeto em desenvolvimento do projeto de pesquisa do Laboratório de Simulação e Controle de Sistemas Discretos do SEM-EESC, com auxílio pesquisa FAPESP processo: 2003/06582-0 "Veículo Agrícola Autônomo (VAA): Uma plataforma para desenvolvimento de tecnologias de navegação autônoma e para aquisição de dados em agricultura de precisão". Além da descrição da implementação do protótipo virtual, neste trabalho são analisadas cinco ferramentas de desenvolvimento virtual disponíveis para testes no Laboratório de Simulação: 3Dcanvas *Freeware*, WorldUp, EON Studio 4.0, WEBOTS e EON Professional. Os critérios de avaliação das ferramentas foram baseados nas necessidades principais do projeto em questão, como implementação de centros de massa distribuídos, criação de terrenos não-estruturados (constituídos por morros, buracos etc.), controles de tração, torque, além de outras características físicas essenciais. A fase inicial da implementação foi desenvolvida em EON Studio 4.0. Devido à falta de recursos disponíveis nessa ferramenta, a fase final da implementação foi desenvolvida no *software* EON Professional. A análise do comportamento da plataforma em terrenos não-estruturados, a verificação dos ângulos de inclinação suportados e a análise da reconfiguração das laterais da plataforma são alguns dos resultados obtidos.

Palavras-chave: Prototipação Virtual, Realidade Virtual, Ferramentas de Desenvolvimento Virtual.

ABSTRACT

VALE, H.M.C. (2005). Virtual Prototyping of Mobile Agricultural Platform. Dissertação (Mestrado) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2005.

Virtual Reality (VR) has entertainment as its main area of application. However, over the last few years, due to a fast-paced development of computers hardware, VR has become more accessible, being widely used both in the academic and industrial areas. A part of VR that is becoming more important for the engineering areas is Virtual Prototyping (VP). Normally, the physical prototyping of a product can be expensive and impracticable for some projects. In such cases, VP can be used to prevent unnecessary operating costs, reaching satisfactory results. In this paper, the steps of the implementation of a virtual prototype are described, based on the first draft in development of the research project of the Laboratory of Simulation and Control of Discrete Systems of SEM-EESC, with FAPESP support process: 2003/06582-0 "Autonomous Agricultural Vehicle (AAV): A platform for development of technologies for independent navigation and for data acquisition in precision agriculture ". Besides the description of the virtual prototype implementation, in this paper five tools for virtual development, available for tests in the Laboratory of Simulation, are analyzed: 3Dcanvas Freeware, WorldUp, EON Studio 4,0, WEBOTS and EON Professional. The criteria of evaluation of the tools were based on the main necessities of the project, such as the implementation of distributed centers of mass, not-structuralized land creation (made up of hills, holes etc.), traction controls, torque, besides other essential physical characteristics. The initial stage of the implementation was developed at EON Studio 4.0. Due to the lack of available resources in this tool, the final stage of the implementation was developed in EON Professional software. The behavior analysis of the platform at non-structuralized lands, the verification of the supported inclination angles and the analysis of the platform laterals reconfiguration are some of the achieved results.

Keywords: Virtual Prototyping, Virtual Reality, Virtual Development Tools.

LISTA DE FIGURAS

FIGURA 1 – ESTRUTURA DE UM DISPOSITIVO CAVE 8
FIGURA 2 – VISBOX 8
FIGURA 3 – LUVA PARA ENTRADA DE DADOS 9
FIGURA 4 – MOUSE 3D LOGITEC 9
FIGURA 5 – VARIÁVEIS QUE CARACTERIZAM SISTEMAS DE RV 11
FIGURA 6 – APLICAÇÃO DE RENDERIZAÇÃO SOBRE UMA IMAGEM 19
FIGURA 7 – TAREFAS DA PROTOTIPAÇÃO VIRTUAL 22
FIGURA 8 – CARACTERÍSTICAS FREQUENTEMENTE SIMULADAS EM PROTÓTIPOS VIRTUAIS 24
FIGURA 9 – CICLO DE DESENVOLVIMENTO DE UM PRODUTO 28
FIGURA 10 – PROCEDIMENTO DE DESENVOLVIMENTO DE UM PRODUTO 28
FIGURA 11 – EXEMPLOS DE PROTÓTIPOS VIRTUAIS 30
FIGURA 12 – PRINCIPAIS COMPONENTES DE UM SISTEMA PARA PROTOTIPAÇÃO VIRTUAL 30
FIGURA 13 – AMBIENTE DE TRABALHO DO EON STUDIO 40
FIGURA 14 – PLATAFORMA AGRÍCOLA MÓVEL 51
FIGURA 15 – PLATAFORMA SIMPLES CRIADA POR PROTÓTIPOS 53
FIGURA 16 – ÁRVORE FORMADA POR *FRAMES, MESHS* E *MATERIALS* 54
FIGURA 17 – ÁRVORE RESULTANTE DA IMPORTAÇÃO DE UM CUBO E UM CILINDRO 55
FIGURA 18 – JANELA *ROUTES* DO MOVIMENTOS DA SETA DIREITA 60
FIGURA 19 – APLICAÇÃO DE UM *LAYER* 63
FIGURA 20 – *SCRIPT* DE MOVIMENTO DE UMA DAS RODAS 64
FIGURA 21 – SOLO CRIADO POR *MESH* E TEXTURA 66
FIGURA 22 – PLATAFORMA ATRAVESSA O CHÃO E PÁRA EM Z=0 67
FIGURA 23 – PLATAFORMA COLIDINDO COM ELASTICIDADE 72
FIGURA 24 – *SCRIPT* DO MOVIMENTO SETA DIREITA 73
FIGURA 25 – VALORES DE RESISTÊNCIA ALTERADOS 74
FIGURA 26 – PROPRIEDADES DO *TIMESENSOR* 76
FIGURA 27 – VELOCIDADE ZERADA POR ROTAS 76
FIGURA 28 – ROTAS DA IMPLEMENTAÇÃO POR *SCRIPT* 77
FIGURA 29 – TERRENO COM DESNÍVEIS 79
FIGURA 30 – UM OBJETO E SEUS EIXOS DE COORDENADAS 81
FIGURA 31 – POSSÍVEIS VALORES DE *HEADING* 83
FIGURA 32 – SINAIS DOS EIXOS NOS QUADRANTES 84
FIGURA 33 – VALORES DE ORIENTAÇÃO POSITIVOS E NEGATIVOS 88
FIGURA 34 – ESFERA NORMAL (ESQUERDA), ESFERA DEFORMADA (DIREITA) 91
FIGURA 35 – QUANTIDADE DE SEGMENTOS DE UM CUBO 92
FIGURA 36 – OBJETO DIVIDIDO EM VÉRTICES 92
FIGURA 37 – GEOMETRIAS PRODUZIDAS COM O USO DA FUNÇÃO *COLLAPSE* 93
FIGURA 38 – VÉRTICES DESLOCADOS PARA CRIAÇÃO DE BURACOS NO PLANO 93

FIGURA 39 – SUB-ÁRVORE DA PLATAFORMA AGRÍCOLA 94
FIGURA 40 – ÁRVORE DE CENA DO WEBOTS 98
FIGURA 41 – PLATAFORMA CRIADA EM WEBOTS 100
FIGURA 42 – TERRENO IRREGULAR CRIADO EM WEBOTS 101
FIGURA 43 – EXEMPLO DE HDRMATERIAL 103
FIGURA 44 – EXEMPLO DE HATCHMATERIAL 103
FIGURA 45 – SOLO COM MULTILAYERMATERIAL 104
FIGURA 46 – ÁRVORE DE CENA DO SOLO, COM OS RESPECTIVOS MAPS 104
FIGURA 47 – EXEMPLO DE PROGRAMA CG WATER 105
FIGURA 48 – EXEMPLO DE PROGRAMA CG GLASS COM BUMPMAPPING 105
FIGURA 49 – BUMP MAPPING 106
FIGURA 50 – CUBEMAP E ENVIRONMENT MAPPING 106
FIGURA 51 – DARKMAPPING 107
FIGURA 52 – EXEMPLO DE CUBO DUPLO 110
FIGURA 53 – ÁRVORE DE SIMULAÇÃO FINAL DO PROJETO 118
FIGURA 54 – *FRAMES* DO OBJETO PLATAFORMA 119
FIGURA 55 – COMPONENTES DAS RODAS 119
FIGURA 56 – MILHARAL VISTO DE CIMA 123
FIGURA 57 – PROPRIEDADES DE UM NÓ PLACE 124
FIGURA 58 – PLACE E PLACE1 PARA UM PÉ DE MILHO 125
FIGURA 59 – GEOMETRIA DE COLISÃO DOS PÉS DE MILHO E DA PLATAFORMA 127
FIGURA 60 – ROTAS DE COLISÃO COM O MILHARAL 129
FIGURA 61 – ROTAS DE SIMULAÇÃO DO SOM DE COLISÃO COM O MILHARAL 130
FIGURA 62 – TERRENO NÃO ESTRUTURADO 131
FIGURA 63 – TERRENO NÃO ESTRUTURADO DAS LINHAS DE PLANTIO 132
FIGURA 64 – COMPONENTES DO *FRAME* TERRENOS 132
FIGURA 65 – OBSTÁCULOS DO AMBIENTE: DEGRAUS E PEDRAS 133
FIGURA 66 – VIEWPORTS DO PROJETO 134
FIGURA 67 – ROTAS REFERENTES AOS PONTOS DE VISTA DO AMBIENTE 134
FIGURA 68 – TODAS AS POSSÍVEIS VISTAS DO AMBIENTE 135
FIGURA 69 – VISÃO DO MILHARAL COM AS GEOMETRIAS DE COLISÃO E SEM AS MESMAS 136
FIGURA 70 – ROTAS CRIADAS PARA MOSTRAR MILHOS COLIDIDOS 136
FIGURA 71 – COMPONENTES DO NÓ CÂMERA 137
FIGURA 72 – ROTA DE SIMULAÇÃO DO 2DTEXT 137
FIGURA 73 – VALOR DA PROPRIEDADE *FRICTION* MOSTRADO NA TELA 138
FIGURA 74 – PROPRIEDADE DO NÓ *TIMESENSOR* 139
FIGURA 75 – COMPONENTES DO OBJETO ÁGUA 140
FIGURA 76 – ROTAS REFERENTES À ÁGUA 140
FIGURA 77 – PROPRIEDADES DO NÓ CGMATERIAL 141
FIGURA 78 – ÁGUA 142

FIGURA 79 – MANAGERS DO AMBIENTE 142
FIGURA 80 – ROTAS FINAIS DO PROJETO 144
FIGURA 81 – PLATAFORMA COM AS LATERAIS REPOSICIONADAS 145

LISTA DE TABELAS

TABELA 1 – CARACTERÍSTICAS DOS TIPOS DE RV 12
TABELA 2 – TIPOS DE ARQUIVOS USADOS PELO 3DCANVAS 36
TABELA 3 – COMPARAÇÃO DE FERRAMENTAS 49
TABELA 4 – CÁLCULOS DE FORÇA PARA O PRIMEIRO QUADRANTE 85
TABELA 5 – CÁLCULOS DE FORÇA PARA O SEGUNDO QUADRANTE 86
TABELA 6 – VALORES DE FORÇA NO EIXO X EM FUNÇÃO DA ORIENTAÇÃO 87
TABELA 7 – VALORES DE DENSIDADE 109
TABELA 8 – NECESSIDADES PARA RODAR O EON PROFESSIONAL 116
TABELA 9 – NECESSIDADES PARA O AMBIENTE DE CRIAÇÃO DO EON PROFESSIONAL 117
TABELA 10 – PROPRIEDADE GENDER PARA AS JUNTAS DO CARRO 120

LISTA DE ABREVIATURAS E SIGLAS

2D	Bidimensional
3D	Tridimensional
CAD	*Computer Aided Design*
CAVE	*Cave Automatic Virtual Environment*
DOF	*Degree Of Freedom*
EESC	Escola de Engenharia de São Carlos
HMD	*Head-Mounted-Display*
LOD	*Level Of Detail*
PV	Prototipação Virtual
RV	Realidade Virtual
VAA	Veículo Agrícola Autônomo
VRML	*Virtual Reality Modeling Language*
WIMP	*Windows, Icons, Menus and Pointing*

SUMÁRIO

RESUMO ..... **IV**

ABSTRACT ..... **V**

CAPÍTULO I – INTRODUÇÃO **1**

1.1 CONSIDERAÇÕES GERAIS 1
1.2 OBJETIVOS 2
1.3 JUSTIFICATIVA 3
1.4 ORGANIZAÇÃO DO TRABALHO 4

CAPÍTULO II - REVISÃO BIBLIOGRÁFICA **6**

2.1 REALIDADE VIRTUAL (RV) 6
2.1.1 DISPOSITIVOS *7*
2.1.2 GRAUS DE LIBERDADE *9*
2.1.3 CONCEITOS *10*
2.1.4 APLICAÇÕES DE RV *16*
2.1.5 PROBLEMAS *17*
2.2. PROTOTIPAÇÃO VIRTUAL (PV) 19
2.2.1 MODELAGEM *19*
2.2.2 SIMULAÇÃO *20*
2.2.3 A PROTOTIPAÇÃO VIRTUAL *22*

CAPÍTULO III -FERRAMENTAS DE DESENVOLVIMENTO VIRTUAL **32**

3.1. 3D CANVAS 33
3.1.1 DESCRIÇÃO DO SOFTWARE *33*
3.1.2 MODELAGEM GEOMÉTRICA *34*
3.1.3 OUTRAS CARACTERÍSTICAS DO 3DCANVAS *34*
3.1.4 PROGRAMANDO A CENA (SCRIPTS) *36*
3.2 WORLDUP 37
3.2.1 VANTAGENS *38*
3.2.2 MODELAGEM GRÁFICA *38*
3.2.3 ESTRUTURA DO SOFTWARE *39*
3.3 EON STUDIO 40
3.3.1 MODELAGEM GRÁFICA *42*
3.3.2 ADIÇÃO DE GEOMETRIAS *43*
3.3.3 ESTRUTURA DO SOFTWARE *43*
3.3.4 SCRIPTS *44*
3.3.5 DETECÇÃO DE COLISÃO *45*
3.3.6 DINÂMICA DE CORPOS RÍGIDOS *45*

3.3.7 PROTÓTIPOS *48*
3.4 COMPARAÇÃO DE FERRAMENTAS 49
CAPÍTULO IV -PROTOTIPAÇÃO VIRTUAL DE UMA PLATAFORMA AGRÍCOLA MÓVEL 51
4.1 IMPLEMENTAÇÃO EM EON STUDIO 4.0 52
4.1.1 MONTANDO UMA PLATAFORMA SIMPLES *52*
4.1.2 MOVIMENTANDO A PLATAFORMA *57*
4.1.3 ADICIONANDO GRAVIDADE E COLISÃO *65*
4.1.4 IMPLEMENTAÇÃO DAS FORÇAS FÍSICAS *69*
4.1.5 CRIANDO E IMPORTANDO A GEOMETRIA DA PLATAFORMA *89*
4.2 OUTRAS FERRAMENTAS DE DESENVOLVIMENTO VIRTUAL 95
4.2.1 WEBOTS *95*
4.2.2 EON PROFESSIONAL *102*
4.3 DESENVOLVIMENTO DA PLATAFORMA VIRTUAL AGRÍCOLA MÓVEL EM EON PROFESSIONAL 118
4.3.1 PLATAFORMA *119*
4.3.2 MILHARAL *122*
4.3.3 COLISÃO COM O MILHO *126*
4.3.4 TERRENOS *130*
4.3.5 VIEWPORTS E CÂMERAS *133*
4.3.6 CÂMERA *137*
4.3.7 ÁGUA *139*
4.3.8 OUTROS NÓS *142*
4.4 RESULTADOS OBTIDOS 145
CAPÍTULO V -CONCLUSÃO 147
5.1 SUGESTÕES PARA TRABALHOS FUTUROS 148
REFERÊNCIAS 149
Apêndices 155
Anexos 161

CAPÍTULO I – INTRODUÇÃO

1.1 Considerações Gerais

A Realidade Virtual (RV) evoluiu de simulações voltadas, principalmente, ao entretenimento, para uma ferramenta de apoio na resolução de problemas em diversas áreas, como medicina, arquitetura, engenharia etc. Especificamente, as atividades de concepção, projeto preliminar e de testes de novos produtos podem ser amplamente beneficiadas pela utilização de sistemas de RV, de diversas maneiras (Palma, 2001), (Porto, 2000), (Kirner, 1999). Geralmente empresas utilizam protótipos físicos, e mais atualmente protótipos digitais, para a realização de testes em novos produtos, porém muitas vezes o custo desses protótipos é inviável, ou no caso dos digitais não permite a realização dos testes (Zorriassatine, 2002). Com o avanço da tecnologia de computação, da queda de preços dos dispositivos de entrada&saida de dados e dos aplicativos de realidade virtual, várias empresas estão trocando protótipos físicos e digitais por protótipos virtuais (Choi, 2004), (Daí, 1994), (Ovtcharova, 1994).

Como a prototipação virtual de um produto não é uma tarefa simples, a escolha da ferramenta mais adequada ao trabalho é extremamente importante. Com isso se pode economizar tempo de desenvolvimento e conseguir um protótipo virtual que simule corretamente todas as características de um produto, permitindo a realização dos testes necessários (Stribersky, 2002).

Neste trabalho foi realizada uma análise de ferramentas de criação de ambientes virtuais com o propósito de prototipar virtualmente uma Plataforma Agrícola Móvel para atuar sobre um ambiente virtual agrícola criado, preferencialmente, na mesma ferramenta. Concomitante com a análise das ferramentas, foi escolhida a mais adequada para esse tipo de projeto. Em seguida, o projeto foi implementado sobre a ferramenta escolhida.

Os dados referentes à plataforma virtual agrícola móvel implementada tomaram como base os dados do anteprojeto em desenvolvimento no projeto de pesquisa do Laboratório de Simulação e Controle de Sistemas Discretos do SEM-EESC, com auxílio pesquisa FAPESP processo: 2003/06582-0 "Veículo Agrícola Autônomo (VAA): Uma plataforma para desenvolvimento de tecnologias de navegação autônoma e para aquisição de dados em agricultura de precisão".

1.2 Objetivos

O principal objetivo deste trabalho é a Prototipação Virtual de uma plataforma agrícola móvel, e do seu ambiente de operação. Para alcançar esse objetivo, duas etapas foram estipuladas: a escolha da ferramenta de desenvolvimento virtual mais adequada à criação de uma plataforma virtual agrícola móvel e a criação da plataforma, na ferramenta escolhida, para a realização de alguns tipos de testes. Cada uma dessas etapas será explicada abaixo.

A prototipação virtual de um produto não é uma tarefa simples, pois utiliza softwares de desenvolvimento virtual que possuem diversas funcionalidades, interfaces e linguagens de programação diferentes. Dessa forma, a primeira fase de um projeto de prototipação virtual, deve ser uma análise das ferramentas de desenvolvimento virtual existentes no mercado, para que sejam evitados gastos com ferramentas inadequadas e desperdício de tempo (Williams, 2003), (Bullinger, 2003), (Stribersky, 2002).

A análise de ferramentas virtuais para o desenvolvimento de uma plataforma virtual agrícola móvel é um dos principais objetivos deste trabalho. Essa análise deve ser executada, com base nas necessidades principais do projeto. No caso da prototipação virtual da plataforma agrícola, algumas das necessidades principais são: a implementação de um centro de massa distribuído para o veículo, criação de terrenos não-estruturados, adição de forças físicas como gravidade, torque, colisões etc.

A prototipação virtual de um produto visa, entre outras coisas, prever o funcionamento do mesmo, antes deste ser criado fisicamente. Além da análise de ferramentas de desenvolvimento virtual, outro objetivo deste trabalho é a prototipação virtual da plataforma agrícola móvel, na ferramenta escolhida, e a realização de alguns tipos de testes sobre a mesma, para verificação de seu comportamento.

Os valores de medidas utilizados nesse trabalho se assemelham aos dados reais do projeto original, pois estes foram adquiridos das primeiras etapas do projeto VAA. Portanto, a realização de testes sobre essa plataforma virtual pode prever, de forma aceitável, o comportamento da plataforma agrícola real. Os resultados obtidos nesse trabalho, como ângulos de inclinação tolerados pelo veículo, averiguação da sua dirigibilidade e as mudanças no centro de massa causadas pela re-configuração das laterais da plataforma, poderão ser averiguados somente em um trabalho futuro.

Inicialmente, os testes propostos a serem realizados na plataforma virtual foram:

- Análise do comportamento da plataforma em terrenos não-estruturados, ou seja, compostos por desníveis, buracos, degraus, rampas, pedras etc.;
- Verificação da inclinação necessária para tombar a plataforma;
- Verificação da capacidade da plataforma ultrapassar diferentes tipos de obstáculos, como pedras de grande, médio e pequeno porte;
- Análise do comportamento da plataforma em diferentes condições de terrenos, como por exemplo, terrenos secos e molhados, para a verificação do grau de derrapagem da plataforma;
- Comparação de dirigibilidade e estabilidade da plataforma, para diferentes posicionamentos das laterais.

Todos os testes propostos puderam ser executados sobre o protótipo virtual implementado. Não fez parte do escopo deste trabalho estudar os problemas relacionados com os sistemas de controle e de potência da plataforma agrícola.

1.3 Justificativa

Visto o rápido desenvolvimento do hardware de computadores na última década, pode-se prever um grande aprimoramento dos hardwares da Realidade Virtual nos próximos anos. Técnicas de modelagem universais também devem ser desenvolvidas para facilitar as atividades de modelagem, dentro da própria ferramenta de RV (Ma, 2004).

Uma parte da RV que, a cada dia, se torna mais importante para as áreas da engenharia é a Prototipação Virtual (PV). Normalmente, a prototipação física de um produto pode ser dispendiosa e/ou inviável para projetos novos e/ou de alta complexidade, sendo que nesses casos, a PV pode ser utilizada para evitar gastos desnecessários e alcançando resultados satisfatórios.

Segundo (Kerttula, 1997), (Williams, 2003), (Porto, 2000) os principais objetivos da Prototipação Virtual são:

- Fornecer um método totalmente digital para o desenvolvimento do produto;
- Aumentar a inovação dos conceitos dos produtos;
- Obter ciclos de desenvolvimentos mais curtos;
- Reduzir custos de desenvolvimento;
- Melhorar a exatidão e qualidade do desenvolvimento para atender as necessidades dos clientes no mercado.

Com este trabalho será possível avaliar as capacidades atuais da PV, bem como suas limitações. Pretende-se que o comportamento da plataforma agrícola do projeto VAA possa ser previsto com a utilização do ambiente criado. Além disso, será realizada uma pesquisa sobre algumas ferramentas de desenvolvimento virtual existentes no mercado.

1.4 Organização do Trabalho

O primeiro capítulo do trabalho mostra os objetivos principais e os motivos que justificam o desenvolvimento do trabalho.

No capítulo seguinte é realizada uma revisão bibliográfica para o embasamento dos assuntos: Realidade Virtual e Prototipação Virtual.

No terceiro capítulo é apresentada uma análise de algumas ferramentas de desenvolvimento de ambientes virtuais existentes no mercado e disponíveis para testes no laboratório da Escola de Engenharia de São Carlos EESC, sendo que ao final deste capítulo é escolhida a ferramenta de desenvolvimento virtual mais adequada à criação de uma plataforma agrícola móvel.

No quarto capítulo é implementado um ambiente virtual que possibilite a realização de testes sobre a plataforma agrícola móvel, com o uso da ferramenta escolhida no terceiro capítulo. Porém, devido ao fato de que mesmo a ferramenta escolhida não pôde suprir todas as necessidades para a implementação total do projeto, duas novas ferramentas de desenvolvimento virtual foram analisadas. A prototipação virtual da plataforma agrícola móvel teve continuidade com o uso dessa nova ferramenta escolhida.

No quinto capítulo são mostradas as conclusões alcançadas com a pesquisa, seguido de referências.

CAPÍTULO II - REVISÃO BIBLIOGRÁFICA

Este capítulo faz uma breve descrição dos principais conceitos envolvidos em Realidade Virtual e Prototipação Virtual. Isto é necessário devido à alta complexidade dos sistemas de RV, que muitas vezes envolvem tecnologias avançadas de *hardwares* e *softwares* do meio computacional.

2.1 Realidade Virtual (RV)

A necessidade atual de reduzir custos e tempo envolvidos no processo de levar um produto de seu conceito até sua produção vem forçando empresas a adotarem novas tecnologias. Uma dessas tecnologias é a Realidade Virtual Mujber (2004).

A RV é uma tecnologia que visa, entre outras coisas, a criação e percepção da realidade, usando dispositivos que estimulem mais que apenas um órgão dos sentidos (Her, 2003). Entendida como uma forma natural e poderosa de interação homem-máquina que permite ao usuário navegar, interagir e "imergir" em um ambiente 3D sintético gerado pelo computador (Araújo, 2005), a RV utiliza um conjunto de tecnologias para fornecer uma interface homem-máquina intuitiva, que possibilite ao usuário o sentimento de imersão num mundo virtual. O usuário pode visualizar e interagir com o ambiente virtual criado, geralmente chamado de Ambiente Virtual 3D (*Virtual Environment*) e pode manipular diretamente objetos dentro desse ambiente (Ma, 2004).

Na indústria, o uso das técnicas de RV é seguido por muita expectativa, visto que a RV se tornou uma tecnologia comum de interface homem-máquina, como uma alternativa, ou extensão, ao uso de *softwares* CAD para modelagem de objetos tridimensionais (Jezernik, 2003).

Os sistemas CAD (*Computer-Aided Design*) são, atualmente, reconhecidos como sistemas básicos para projetos. Modelos CAD 3D proporcionam um meio natural

para troca de informações entre desenvolvedores e outros participantes do processo de produção. Para que essa troca de informação possa ocorrer, todos os participantes envolvidos devem ter acesso a uma estação de trabalho CAD. Isto é aceitável, se os participantes envolvidos trabalham em uma mesma empresa, preferencialmente no mesmo departamento. Além disso, todos devem saber utilizar um sistema CAD.

Devido ao contínuo desenvolvimento dos sistemas CAD, eles têm se tornado ferramentas sofisticadas para modelagem de objetos 3D; altamente parametrizados e extremamente úteis para apresentações, análises e simulações. (Jezernik, 2003).

2.1.1 DISPOSITIVOS

Os *softwares* de RV requerem grande poder computacional para fornecer alguns requisitos básicos, como visualização e interação em tempo real (Jezernik, 2003). Porém, além desse alto poder computacional, a RV não alcançaria algumas de suas metas, como imersão do usuário, sem o uso de dispositivos avançados de entrada e saída.

O participante da experiência de Realidade Virtual pode "entrar" no mundo virtual através dos dispositivos de saída de dados. Os dispositivos de entrada de dados, por outro lado, permitem a movimentação e interação do usuário com o mundo virtual. Sem o dispositivo de entrada de dados o usuário participa da experiência em RV apenas de forma passiva.

Os dispositivos de saída de dados podem ser divididos em

- Dispositivos visuais: como mesas imersivas, capacetes ou *Head-Mounted Displays* (HMDs), óculos estereoscópicos, CAVEs etc;
- Dispositivos auditivos: fornecem maior realismo ao sistema de RV quando utilizados de maneira tridimensional;
- Dispositivos físicos: procuram estimular as sensações físicas, como o tato, tensão muscular e temperatura etc.

Tougaw (2003) diz que o dispositivo Cave Automatic Virtual Environment (CAVE): seis paredes que circundam totalmente o participante com imagens é,

atualmente, o que fornece maiores benefícios aos usuários da RV, porém, seu alto custo o torna inviável a projetos de pequeno e médio porte.

A figura abaixo mostra a estrutura de uma CAVE:

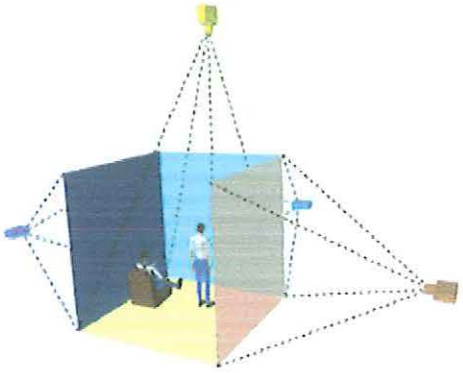

Figura 1. Estrutura de um dispositivo CAVE (Tougaw, 2003).

Como opção ao uso de CAVEs, Tougaw sugere a utilização de um VisBox, visor capaz de mostrar imagens de acordo com os movimentos da cabeça de um participante. Além disso, um VisBox é controlado por um joystick que tem seus movimentos identificados em seis graus de liberdade. Sendo assim, o usuário pode interagir com objetos apenas movimentando o joystick pelo espaço. O valor de um VisBox é significativamente menor que o de uma CAVE (menos de um porcento do valor). A foto de um VisBox é mostrada na figura 2:

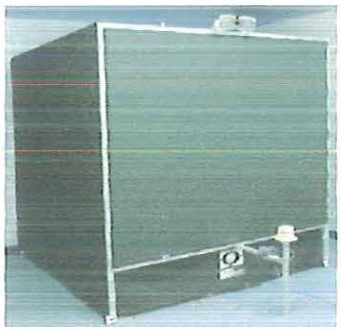

Figura 2 – VisBox (Tougaw, 2003).

Por sua vez, os dispositivos de entrada de dados mais comuns para interfaces normais (WIMP) são: mouses e teclados. Para a RV de baixo custo, esses dispositivos também são muito utilizados. Porém, para um ambiente virtual mais complexo, outros tipos de dispositivos de entrada de dados devem ser utilizados.

Segundo He (1993) os dispositivos de entrada da RV podem ser classificados em quatro categorias: Posição 2D, Posição 3D, Orientação 3D e Seleção.

Figueroa (2004) apresenta o exemplo de uma luva especial para entrada de dados, que é mostrada na figura 3:

Figura 3 - Luva para entrada de dados (Figueroa, 2004)

Chedmail (2003) mostra a figura de um mouse especial para o controle de sistemas tridimensionais:

Figura 4 - Mouse 3D Logitec (Chedmail, 2003).

2.1.2 GRAUS DE LIBERDADE

O termo Grau de Liberdade, do inglês: *Degree Of Freedom* (DOF), representa o número de posições independentes para rotação e translação que uma estrutura articulada possui no espaço. Os objetos podem mover-se para cima ou para baixo, no eixo Y, mover-se para esquerda ou direita no eixo Z, ou mover-se para frente ou para trás, acompanhando o eixo X. Os objetos também podem ser rotacionados ao redor desses três eixos seguindo os movimentos a seguir:

- Se rotacionados ao redor de X, esse movimento é chamado de *Roll*;
- Se rotacionados ao redor de Y, o movimento é conhecido como *Yaw*;

- Se rotacionados ao redor de Z, o movimento é chamado de *Pitch*.

Dependendo do sistema de Realidade Virtual, pode-se ter um grau de liberdade de até 6DOF (seis graus de liberdade), o que proporcionará maior sensação de imersão no mundo virtual.

2.1.3 CONCEITOS

São vários os conceitos envolvidos num sistema de Realidade Virtual. Segundo Kirner (1999) uma aplicação de Realidade Virtual pode ser realizada de três maneiras diferentes: Passiva, Exploratória ou Interativa.

- **Passiva**: Quando o usuário não pode realizar qualquer tipo de ação ou escolha sobre o ambiente virtual. Funciona praticamente como uma animação assistida pelo usuário.

- **Exploratória**: Na forma exploratória, como o próprio nome diz, o usuário pode explorar o ambiente, escolhendo as rotas e as visões que o sistema deve mostrar. Não se pode, no entanto, interagir com qualquer objeto do ambiente virtual, que funcionam apenas como cenário.

- **Interativa**: Nessa forma de RV, o usuário além de poder escolher rotas e vistas, pode também interagir com objetos do ambiente virtual. Por exemplo, ao encontrar uma porta, o usuário pode ou não abri-la, fazendo com que o sistema ative o procedimento de animação de abertura da porta devido a alguma ação realizada.

Sánchez-Segura (2003) mostra as variáveis que descrevem um sistema de RV (Interação, Imersão e Envolvimento), mostradas na figura 5:

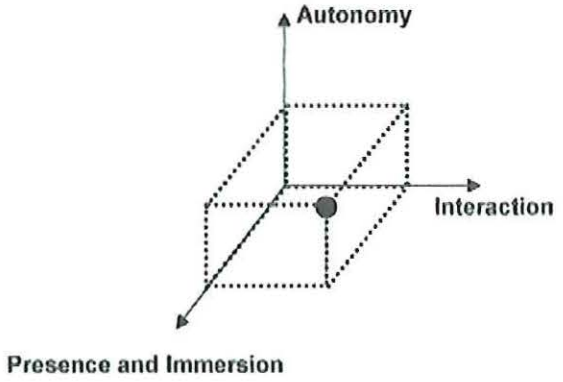

Figura 5 - Variáveis que caracterizam sistemas de RV (Sánchez-Segura, 2003).

Assim como Sánchez-Segura, Kirner (1999) considera a Realidade Virtual como a junção de três idéias básicas: imersão, interação e envolvimento.

2.1.3.1 Imersão

Quando uma pessoa utiliza dispositivos que permitem a sensação de estar realmente dentro do ambiente virtual, dizemos que esta pessoa está imersa no ambiente. Esses dispositivos podem ser desde capacetes de visualização até salas de projeções.

Qual é realmente o valor da imersão? O maior efeito causado por ela é a capacidade de bloquear distrações de eventos exteriores ao ambiente, e focalizar toda a atenção do usuário com o seu objeto de trabalho.

Apesar da diminuição considerável de preço que a Realidade Virtual vem sofrendo nos dias de hoje, ela ainda é muito cara. Para conseguir um grau de imersão aceitável em um ambiente virtual, é necessário o uso de alguns dispositivos, que são os maiores responsáveis pelo alto custo da RV.

Steuer (1992 apud. Bhatt 2004) divide a Imersão em duas categorias:

- Largura: está relacionada à quantidade de sentidos que o ambiente estimula no usuário;

- Profundidade:está relacionada à quantidade de informação (resolução) disponível ao usuário.

A RV oferece a possibilidade de interações imersivas ou semi-imersivas que pode ser uma ferramenta importante para simulações visuais do processo de produção (Okulicz, 2004). A respeito da Imersão, a RV pode ser dividida em três categorias: Não-Imersiva, Semi-Imersiva e Totalmente-Imersiva. Mujber (2004) mostra uma tabela com as características de cada tipo de RV:

Tabela 1 - Características dos tipos de RV

Sistema de RV	**RV Não-Imersiva**	**RV Semi-Imersiva**	**RV Totalmente-Imersiva**
Dispositivos de entrada	Mouses, teclados, joysticks etc.	Joysticks, luvas etc.	Luvas e comandos por voz.
Dispositivos de Saída	Monitores padrões de alta resolução	Monitores com tela grande, sistemas de projeção	HMDs, CAVE.
Resolução	Alta	Alta	Baixa-Média
Percepção da Imersão	Nenhuma-Baixa	Média-Alta	Alta
Interação	Baixa	Razoável	Alta
Preço	Sistema mais barato de RV	Alto	Muito alto

Mujber (2004) mostra na tabela acima que a RV totalmente imersiva é geralmente utilizada com níveis de resolução não muito altos, devido à grande necessidade de velocidade e respostas em tempo real. Porém, um ambiente ideal de RV deveria possuir respostas em tempo real juntamente com altos níveis de resolução.

2.1.3.2 Interação

A RV é um amplo campo que define um novo caminho para interação com a informação, diferente das tradicionais interfaces gráficas baseadas em Windows, Icons, Menus e Pointers (WIMP). A RV permite aos usuários visualizarem informações de projetos geográficos, moleculares ou industriais, da mesma forma que esses são entendidos no mundo real (Figueroa, 2004). A palavra interação diz respeito à

influência que a pessoa pode ter no ambiente virtual, devido a alguma ação realizada. Um exemplo simples de interação, é o clique do mouse sobre um interruptor que apague ou acenda uma luz. Outros métodos mais avançados de interação podem ser conseguidos através da utilização de dispositivos de entrada de dados mais sofisticados, como por exemplo, uma luva.

Há dois aspectos de interatividade em um ambiente virtual. A navegação e as dinâmicas do ambiente (resposta do ambiente às ações do usuário). A navegação nada mais é que a capacidade do usuário movimentar-se livremente pelo ambiente. O desenvolvedor pode definir os tipos de movimentos do usuário para torná-los mais adequados à certas aplicações, podendo assim criar as sensações de voar, andar, nadar etc. Lécuyer (2004) diz em seu trabalho que a Navegação pode ser uma das interações chave da RV Imersiva.

Nos ambientes virtuais, a navegação é quase sempre estimulada somente pela visão. A maioria dos outros sentidos, que estão sempre presentes na navegação no mundo real, estão geralmente ausentes na RV. A falta dessas informações acarreta em problemas de conflito entre a visão e os outros sentidos. Atrasos na renderização das imagens geralmente causam náuseas nos usuários (problema muito encontrado em aplicações de RV).

Segundo Bhatt (2004), três fatores que contribuem para a interação são:

1) Velocidade de Processamento: refere-se à taxa a qual as entradas de dados são assimiladas pelo ambiente. Respostas em tempo real é o máximo valor de interatividade que o sistema pode possuir;
2) Alcance: o alcance de interatividade refere-se ao número de atributos do ambiente que podem ser manipulados e a quantidade de variações de cada atributo. O alcance de interatividade pode variar da manipulação das dimensões dos objetos, até a manipulação de sons, cores e freqüências;
3) Significância: refere-se ao modo que o ambiente responde às ações dos usuários. Um sistema que responda às ações dos usuários de uma maneira realista é considerado maior na significância do que um sistema que responda ao usuário de uma maneira inferior à esperada.

Berenguer (1997 apud. Sánchez-Segura 2003) diz que existem algumas características que diferenciam os sistemas de interação antigos dos novos:

1) A quantidade de controle possuído pelo usuário, ou seja, o grau de autonomia que permite o usuário decidir o que fazer, para onde navegar etc;
2) A quantidade de interação permitida que depende das possibilidades existentes para que o usuário interaja com o ambiente;

2.1.3.3 Envolvimento

A idéia de envolvimento, por sua vez, está ligada com o grau de motivação para o engajamento de uma pessoa com determinada atividade. O envolvimento pode ser passivo, como ler um livro ou assistir televisão, ou ativo, ao participar de um jogo com algum parceiro. A realidade virtual tem potencial para os dois tipos de envolvimento ao permitir a exploração de um ambiente virtual e ao propiciar a interação do usuário com um ambiente virtual dinâmico.

Segundo Wloka (1994), para um melhor aproveitamento da Imersão, Interação e Envolvimento, os sistemas de RV devem possuir:

- Entradas Múltiplas: trackers (para mãos e cabeça), luvas, reconhecimento de fala etc. Integrar esses dispositivos não é uma tarefa simples, pois além de terem de funcionar simultaneamente, eles têm de operar em harmonia;
- Saídas Múltiplas: além de dispositivos de saída visuais, a RV requer dispositivos táteis e auditivos. Os dispositivos de áudio que permitem a utilização de som 3D estão se tornando comuns. Porém, os dispositivos táteis ainda são uma área de pesquisa. Integrar múltiplos dispositivos de saída em uma única aplicação de RV é uma tarefa complicada devido a diversas razões. Todos os dispositivos têm de funcionar em uma sincronia extremamente precisa, pois a tolerância dos seres humanos a erros de sincronia é relativamente pequena;
- Múltiplos Participantes: num futuro próximo será normal o uso de ambientes virtuais distribuídos entre, não somente dois ou três usuários,

mas milhares de pessoas simultaneamente. Controlar milhares de participantes implica no uso de multi-processamento e trabalho via rede. Alguns problemas encontrados podem ser causados pelos tempos de latência indeterminados de uma rede de dados;

- Mundos Virtuais Dinâmicos: para que os mundos virtuais se tornem semelhantes ao mundo real, eles têm de ser dinâmicos. Ao invés de simular comportamentos dinâmicos pela reedição de uma base de dados estática, a base de dados da RV deve ser baseada fundamentalmente no comportamento dinâmico;
- Paradigmas de Interface: Menus flutuantes, movimentos, objetos 3D etc.;
- Performance: é medida pelo número de quadros por segundo que são dispostos ao usuário. Uma cena realista requer 80 milhões de polígonos X 8 quadros por segundo, num total de 640 milhões de polígonos.

2.1.3.4 Presença

Um ambiente de imersão completo é aquele onde o usuário está completamente ligado ao mundo virtual e sua percepção sobre o mundo real está completamente bloqueada. A navegação no mundo virtual permite que o usuário se mova para qualquer região do mesmo. Finalmente, a interação é a capacidade do usuário de interagir com os dados. Todos esses fatores contribuem para o principal conceito da RV: Presença. A presença nos mundos virtuais é a ilusão convincente de estar presente em um mundo artificial sintético.

Quanto maior o nível de detalhes dos componentes, maior a Presença. O grande problema é: quanto maior o nível de detalhamento do sistema, mais poder de hardware é requerido. Se o hardware não suportar as requisições do ambiente, a Presença pode não ser alcançada. Portanto, os dados e componentes do ambiente devem ser representados com o mínimo de detalhes possíveis para as necessidades da aplicação (Papson, 2004).

A avaliação da qualidade de um ambiente virtual está ligada à qualidade da Presença, ou seja, o quanto real ou natural é a experiência de um usuário quando este se encontra imerso no ambiente. A presença tem sido avaliada por questionários feitos a usuários, que visam medir a qualidade do sentimento de "estar lá" (Sutcliffe, 2004).

2.1.4 APLICAÇÕES DE RV

Hoje em dia, o computador está firmemente integrado na sociedade moderna, envolvendo as mais diversas áreas do conhecimento. A RV, sendo uma interface, acaba tendo a mesma abrangência. Pode-se perceber claramente uma significante evolução da RV, tanto na área de facilidade de uso quanto em variedade de aplicações (Papson, 2004).

Segundo Wojciechowski (2004), a RV já alcançou o nível de maturidade que permite seu uso em aplicações da vida real, como por exemplo, medicina e educação.

Embora o número de aplicações de realidade virtual seja imenso, serão relacionadas a seguir algumas áreas de aplicação identificadas por Kirner (1999).

- **Aplicações Industriais:** Visualização de protótipos, Treinamento, Simulação de montagens, Simulação da dinâmica de estruturas articuladas, Análise de tensões, Simulação do processo produtivo etc;
- **Aplicações Médicas e em Saúde:** Ensino de anatomia, Planejamento cirúrgico, Simulação cirúrgica, Terapia virtual, Tratamento de deficientes;
- **Aplicações em Arquitetura e Projeto:** Planejamento da obra, Inspeção tridimensional em tempo real, Interação em tempo real, Decoração de ambientes, Avaliação acústica;
- **Aplicações Científicas:** Visualização de superfícies planetárias, Síntese molecular, Visualização de elementos matemáticos, Análise de comportamento de estruturas atômicas e moleculares, Análise de fenômenos físico-químicos;
- **Aplicações em Artes:** Pinturas em relevo, esculturas, museus virtuais com detalhes nas paredes e teto além das próprias obras de arte, música com instrumentos virtuais etc.
- **Aplicações em Educação:** laboratórios virtuais, encontros remotos de alunos e professores para terem uma aula ou alguma atividade coletiva, participação em eventos virtuais, consulta a bibliotecas virtuais, educação de excepcionais etc.
- **Aplicações em Controle da Informação:** Cada vez mais é necessário o acesso rápido e adequado a conjuntos complexos de informações para as mais variadas aplicações de tomada de decisão. O espaço bidimensional é muito limitado para isto, de forma que a representação e o posicionamento de informações no mundo virtual tridimensional vem agregar um grande potencial para aplicações como: visualização

financeira, visualização de informações em geral, informação virtual, visualização de simulação de sistemas complexos etc.

- **Aplicações de Entretenimento:** Aplicações em entretenimento têm a vantagem de atingir escalas de consumo bastante altas, viabilizando o lançamento de uma série de produtos. É o caso dos videojogos que viabilizaram os chips de microprocessadores e alguns periféricos de realidade virtual de baixo custo. Dentre as diversas aplicações, além dos videojogos tridimensionais com interação em tempo real, tem-se turismo virtual, passeio ciclístico virtual, esportes virtuais, cinema virtual etc.
- **Outras Aplicações:** Há muitas outras aplicações, Machado (1995) mostra aplicações nas áreas de treinamento, cidades virtuais, comércio eletrônico, modelagem, simuladores, estúdios virtuais, robôs teleguiados, engenharia civil, telepresença etc. Além disso, novas aplicações surgem a cada dia, dependendo da necessidade e da imaginação de cada um. A realidade virtual vem propiciando uma nova maneira de ver coisas conhecidas ou o desenvolvimento de novas aplicações. Segundo Ma (2004), a RV tem sido aplicada com sucesso nas áreas de visualização científica, simulação de vôo, telepresença, treinamento, educação, medicina, entretenimento e manufatura. Uma das aplicações da RV na manufatura é chamada de Prototipação Virtual, que será explicada no próximo tópico da Revisão Bibliográfica. A RV tem aplicações tipicamente nas áreas industriais, projeto, prototipação rápida e visualização científica (Papson, 2004). McLeod (2001 apud Lister 2004) mostra algumas empresas que possuem aplicações bem sucedidas de RV, como General Motors, British Aerospace, Marconi, Scottish Nuclear Power e Nokia.

2.1.5 PROBLEMAS

Sutcliffe (2004) mostra alguns desafios que podem estar presentes no projeto de sistemas de RV:

- Dificuldades de Percepção: profundidade 3D ou distorção da perspectiva, baixa resolução das imagens;
- Dificuldades de Navegação e manipulação: mover e manipular a presença do usuário. Problemas com hardwares de RV;

- Dificuldades de interação com objetos;
- Características anormais: partes do ambiente que criam efeitos inesperados, como movimento através de paredes ou objetos flutuando;
- Interação com outros tipos de controles: como menus, por exemplo;
- Outros problemas de hardware: com HMDs ou óculos estereoscópicos.

Willis (2004) enfatiza em seu trabalho os problemas relacionados à implementação de forças físicas em um ambiente virtual. Willis diz que a modelagem de características físicas não é um problema trivial para a computação. Esta possui alguns efeitos prejudiciais de performance em tempo real. Além disso, as características físicas do dia a dia, como pegar, mover ou empilhar objetos envolvem pouca computação. Forças de tração e colisões também seguem o mesmo conceito. Interações elásticas são mais complexas, mas as necessidades de cálculos elásticos exatos são raras no nosso dia a dia. Dinâmicas complexas são simplificadas, quando apenas objetos rígidos são envolvidos. Isto significa que é necessária uma representação consistente para as forças físicas no mundo.

Outro problema muito freqüente das aplicações de RV diz respeito à navegação. Nos ambientes virtuais, a navegação é quase sempre estimulada somente pela visão. A maioria dos outros sentidos – que estão sempre presentes na navegação no mundo real – está geralmente ausente na RV. A falta dessas informações acarreta em problemas de conflito entre a visão e os outros sentidos, o que geralmente causa náuseas nos usuários.

2.2. Prototipação Virtual (PV)

Antes de entrar em detalhes sobre a Prototipação Virtual, alguns conceitos sobre Modelagem e Simulação de dados devem ser definidos.

2.2.1 MODELAGEM

Computer-Aided-Design (CAD), que significa: "projeto com a ajuda do computador", é uma área onde a computação contribuiu imensamente com a produtividade. Produtos como o AutoCad da AutoDesk™ proporcionam aos desenvolvedores criar e editar rapidamente projetos complexos de qualquer tipo de produto, em computadores pessoais.

Os computadores podem animar e simular como os produtos irão atuar quando forem construídos, como a luz irá refletir em uma superfície, e como o som irá soar em uma sala.

A construção de um ambiente virtual passa por três fases principais: modelagem, renderização e interação em tempo real.

- **Modelagem**: É o processo de construção das geometrias e atributos físicos dos objetos que constituem o ambiente virtual. Usam-se geralmente os *softwares* CAD. Esse é um processo complicado, pois depende de bases de dados e da configuração de *hardware* do computador;
- **Renderização**: É um processo que inclui aplicação de texturas e definição de parâmetros de luz, exemplificado na figura 6. O tempo de renderização depende de sua qualidade, da complexidade do modelo, e da velocidade do *hardware* do computador. Os fatores influentes para conseguir-se o ideal em texturas eram qualidade, velocidade, versatilidade e o tamanho do arquivo.

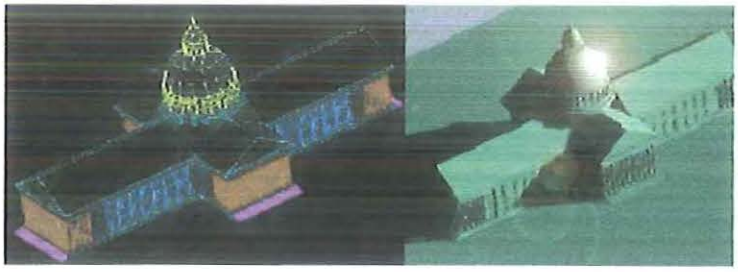

Figura 6 – Aplicação de renderização sobre uma imagem.

Projetar, dar forma a novos objetos ou ambientes, é uma questão de antecipar os trabalhos de ambientes materiais e de espaço, que podem virar realidade, bastando para isso, a sua construção (Breen, 2003). Figueiredo (1994) diz em seu trabalho que a representação da geometria de objetos 3D é um passo importante para a simulação gráfica da realidade, pois ela descreve objetos reais e fornece os dados necessários para a simulação realista dos objetos e seus comportamentos.

Alguns aspectos da modelagem são essenciais para alcançar simulações eficientes. Os componentes dinâmicos e geométricos do sistema devem ser integrados com grande cuidado. As funções das geometrias e das dinâmicas, na análise de contato, devem ser bem definidas. Por exemplo, no deslizamento de um bloco sobre uma mesa inclinada, há duas maneiras de quebrar o contato: uma força puxando o bloco para cima (evento dinâmico) ou o término do movimento do bloco, alcançando outra geometria (evento geométrico).

A modelagem por computador é, em alguns aspectos, semelhante à modelagem física, mas outros aspectos a tornam significativamente diferente. O computador pode ser usado, é claro, para desenhar, mas seu verdadeiro potencial está em criar ambientes virtuais tridimensionais. Trabalhar no computador é, essencialmente, trabalhar em "escala total (100%)". Um aspecto negativo disso é a falta de overview (visão geral). Para conseguir ver detalhes, é preciso aumentar o zoom, para ter-se uma visão completa é preciso diminuir o zoom (o que acarreta uma séria perda de informação visual). Para compensar isso, há muitos aspectos positivos, como a facilidade de implementar mudanças, o uso de Layers e a possibilidade de visualizar de diferentes maneiras, como modo simples, modo com sombreamento e modo renderizado (com texturas).

Na Realidade Virtual, o modelo não é somente em escala total (100%), mas também visto dinamicamente em tempo real e interativamente.

O melhoramento contínuo das tecnologias computacionais e da modelagem perfeccionista elevará o grau de ilusão até ambientes que proporcionarão algo muito semelhante à "realidade".

2.2.2 SIMULAÇÃO

O mundo já passou por várias fases: agrícola, mercantil, industrial, e hoje o que prevalece é a informação. A utilização e manipulação das informações têm proporcionado muitos avanços, mas muitos desses poderiam ser maximizados por meio

da inclusão da tecnologia de simulação existente. Entretanto, essas oportunidades estão sendo perdidas ou a tecnologia é reinventada pela falta de comunicação, marketing e convergência de interesses (Porto, 2000).

Diferente de antigamente onde muitas vezes as dificuldades e gastos para a implementação da simulação superavam os resultados obtidos, atualmente as novas tecnologias e a alta capacitação dos recursos humanos tornaram a simulação uma ferramenta eficiente (Palma, 2001).

O objetivo principal do estudo da simulação é melhorar a qualidade das decisões administrativas, tornando desejável a animação gráfica. Tal animação permite um excelente meio de estabelecer a credibilidade para o modelo simulado. A realidade virtual é apontada como a principal ferramenta gráfica visual a ser utilizada, pois além da animação ela pode disponibilizar a imersão e a interação, ou seja, o usuário teria sensação de estar dentro do modelo e poderia atuar no ambiente simulado em tempo real, permitindo uma exploração mais rica do mesmo.

As desvantagens na simulação estão sendo superadas pelas novas tecnologias e ampliação da capacitação dos recursos humanos. O avanço constante da capacidade de processamento computacional está se refletindo no potencial dos softwares de simulação que empregarão técnicas de simulação paralela e distribuída para o processamento de dados, cálculos e processamentos gráficos intensivos.

Burger (1994) mostra duas grandes vantagens do uso da simulação:

- Simulações permitem testes prolongados em quaisquer tipos de ambientes, sujeitos a mudanças de condições do ambiente e em situações extremas que podem ser raramente encontradas na realidade, mas que podem ser cruciais para a operação do sistema;
- As simulações podem rodar sem supervisão, a qualquer hora do dia, sob quaisquer condições climáticas ou com vários computadores em paralelo. Com isso, o grande aumento de situações e testes explorados deveria aumentar a confiabilidade do sistema drasticamente.

Porém, um detalhe importante ressaltado por Zhang (2003) é que, a princípio, todos os resultados das simulações estão errados, a menos que alguém possa provar que estes estão certos, de uma forma ou de outra.

As mudanças mais dramáticas na área de simulação se darão na capacidade de visualização. Esta área vem evoluindo rapidamente, e isto já pode ser observado nos avanços das capacidades e nas variações dos softwares comerciais de simulação de manufatura com visualização 3D à Realidade Virtual imersiva em institutos de pesquisas. O uso de um ambiente de RV imersiva se tornará comum no futuro. O usuário será capaz de entrar dentro do modelo e experimentar o sistema de simulação completo e com sons.

2.2.3 A PROTOTIPAÇÃO VIRTUAL

A Prototipação Virtual é um avanço tecnológico promissor e emergente para produtos, processos de planejamentos e qualificações (Fan, 2004). É a combinação da Realidade Virtual com técnicas avançadas de modelagem, simulação e interface de usuário. A tecnologia une modelos de desenvolvimento de diferentes disciplinas da engenharia em um modelo de simulação executável de um produto. Um projetista ou um cliente podem ver, tocar, ouvir e operar um futuro produto, antes da sua implementação física, com baixo custo e esforço (Kerttula, 1997).

Gausemeier (1994) mostra na figura 7 abaixo as tarefas da prototipação virtual:

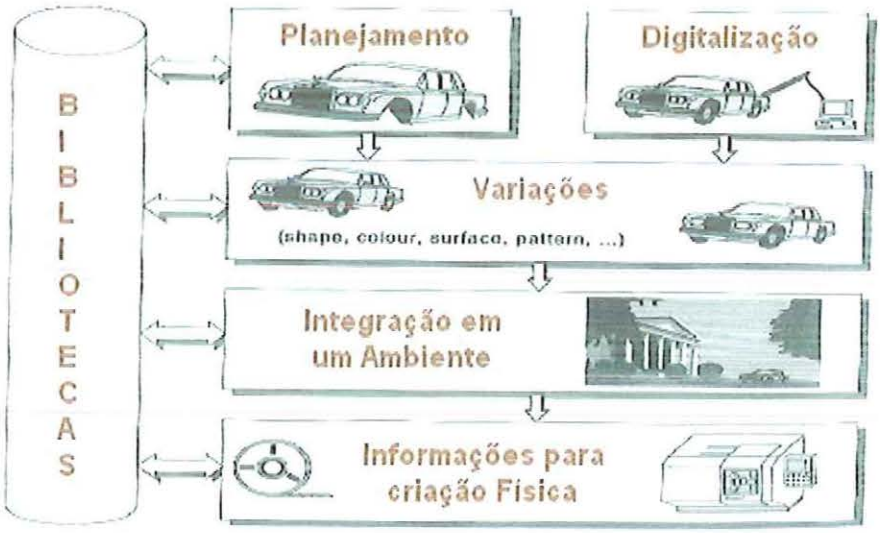

Figura 7 - Tarefas da Prototipação Virtual (Gausemeier 1994).

Não há padrão ou classificação universal para as funções da PV. Existem opiniões diferentes sobre o uso do termo Prototipação Virtual (Zorriassatine, 2002).

Implementar um novo produto ou uma nova característica não é um problema, pois o avanço da tecnologia, geralmente, oferece novas idéias e soluções. Na verdade, o maior problema é como criar um produto ideal para diferentes segmentos de clientes no mercado internacional. As tendências e preferências entre clientes são diferentes e podem mudar rapidamente. No futuro, as companhias encontrarão equilíbrio entre o planejamento, a utilização e as funções de produtos, visando satisfazer diferentes grupos de clientes. As necessidades e requisições dos usuários devem ser estudadas cuidadosamente e efetivamente, antes de qualquer tipo de desenvolvimento, pois a correção de produtos, durante as últimas fases de sua implementação, são muito caras (Kerttula, 1997).

2.2.3.1 Objetivos da PV

Segundo Kerttula (1997), os principais objetivos da Prototipação Virtual são:

- Fornecer um método totalmente digital para o desenvolvimento do produto;
- Aumentar a inovação dos conceitos dos produtos;
- Ciclos de desenvolvimentos mais curtos;
- Reduzir custos de desenvolvimento;
- Melhorar a exatidão e qualidade do desenvolvimento para atender as necessidades dos clientes no mercado.

Na PV um produto, ou o conceito de um produto, seu comportamento e situações de uso são simuladas da maneira mais realista possível usando modelos computacionais e técnicas de Realidade Virtual. O modelo virtual do produto é chamado Protótipo Virtual.

O protótipo virtual é uma simulação baseada em computador de um sistema ou sub-sistema de um protótipo, com um grau de realismo que pode ser comparado com o protótipo físico.

2.2.3.2 Características da PV

Na PV, as características típicas da simulação de um produto são:

- Provê a visualização do futuro produto;
- Propriedades de áudio;
- Funcionalidades da interface para o usuário;
- Provê Funcionalidades e comportamentos do produto;
- Provê Sensações táteis presentes nas interações do usuário com o produto.

A figura 8 ilustra tais características:

Figura 8 – Características freqüentemente simuladas em protótipos virtuais.

A PV está apoiada na aplicação integrada de:

- Modelagem avançada;
- Simulação de várias disciplinas;
- Técnicas de RV.

Dai (1994) diz que protótipos virtuais são alternativas para protótipos físicos, podendo, também, ser usados em conjunto com os mesmos. Um protótipo virtual não tem necessariamente todas as características do produto final, mas tem as características principais suficientes para permitir testes do projeto do produto referentes às necessidades principais (Yin, 2003).

2.2.3.3 Vantagens e Desvantagens da PV

A PV está em seu estágio inicial e promete proporcionar grandes benefícios num futuro próximo. Zorriassatine (2002), diz que o rápido crescimento do poder computacional, a sofisticação dos métodos e modelos computacionais de fenômenos

físicos e a capacidade de transportar resultados entre diversos modelos estão aumentando as áreas de aplicações, exatidão, realismo e o custo benefício da tecnologia de PV, de uma maneira extremamente rápida.

As principais vantagens da Prototipação Virtual, segundo (Porto, 2000) são:

- Diminuição de tempo e custos: os protótipos virtuais reduzem a necessidade de se fazer um grande número de protótipos físicos. Os resultados do protótipo virtual são obtidos rapidamente, possibilitando *feedback* para o projeto, antes dos custos da produção estarem fixados;
- Melhoria da Qualidade: a aplicação de diferentes alternativas para um projeto pode ser realizada rapidamente, permitindo a validação de soluções apropriadas.

Jasnoch (1994) mostra algumas outras vantagens da PV, mas que possuem alguma relação com as vantagens descritas por Porto:

- Usar os objetos criados em computador para comunicação, como enviar um arquivo por e-mail, sem a necessidade de cópias impressas;
- Superar diferenças de tempo e distâncias entre locais: transferência de arquivos, e-mail, compartilhamento de objetos, conferência multimídia integrada com o usuário final;
- Permitir cooperação direta dentro de uma aplicação como edição cooperativa ou planejamento colaborativo. Permitiria também visualizações em grupo junta à equipe de marketing, manufatura, gerência ou clientes.

Pratt (1994) também enfatizou em seu trabalho as vantagens da PV na diminuição do ciclo de vida de um produto:

> Prototipação Virtual é normalmente entendida como a construção de modelos computacionais de produtos, com o objetivo de criar simulações gráficas realistas, geralmente num ambiente de Realidade Virtual. Isto possibilita a realização de testes parciais em um produto, antes que este seja realizado. Essa é uma das muitas estratégias utilizadas para reduzir o ciclo de vida do planejamento de um produto.

Os protótipos virtuais facilitam a validação no estágio inicial do desenvolvimento de um produto. O desenvolvedor tem uma representação clara do produto para analisar suas características estruturais e sua aparência. Se algum problema for identificado, o desenvolvedor pode melhorar o produto antes deste avançar no ciclo de desenvolvimento. Isto é particularmente importante para aumentar a competitividade da indústria, que enfrenta uma pressão constante para atender demandas de produtos (Choi, 2003). Liu (2003) diz que com a utilização da tecnologia PV, o planejamento de idéias e detalhes de um novo instrumento pode ser facilmente verificado, a eficiência do trabalho aumentada e os custos de pesquisas podem ser reduzidos de maneira considerável.

Apesar das grandes vantagens do uso da PV, algumas questões ainda são objetos de discussão:

- Os ambientes virtuais existentes atualmente são visualmente ricos, porém pobres em questões de comportamento (Cremer, 1994).
- Um ponto que deve ser ressaltado sobre a PV é que, a princípio, todos os resultados das simulações estão errados, a menos que alguém possa provar que estes estão certos, de uma forma ou de outra. Portanto, os resultados da prototipação virtual também devem ser verificados com o uso de experimentos ou testes com opções óbvias e diretas (Zhang, 2003).
- Não é simples a obtenção do resultado ótimo para uma simulação. Zhang (2003) mostra em seu trabalho que um fator importante para o sucesso da PV é o princípio do co-planejamento: conjunto de conhecimentos e soluções de diversas disciplinas (mecânica, física, química etc.) e processos (montagem, empacotamento, testes etc.). O planejamento e a qualificação baseada em apenas uma disciplina ou processo levará a resultados que diferem do resultado ótimo (Zhang, 2003).

Ma (2004) cita em seu trabalho, três fatores que limitam o uso da PV:

1. Modelos CAD devem ser sempre criados com dimensões exatas, no entanto, essas dimensões não são especificadas com precisão no nível conceitual do projeto;

2. Relacionamentos e restrições entre entidades são perdidas quando um modelo CAD é exportado da ferramenta especial de criação para um sistema de RV;
3. Para modificar modelos num ambiente virtual, o desenvolvedor necessita retornar ao ambiente de criação de objetos CAD para realizar as mudanças necessárias e, somente então, re-importar os modelos modificados para os sistemas de RV.

2.2.3.4 Protótipos Virtuais Vs. Protótipos Físicos

O uso de protótipos virtuais é uma tendência universal. Seu objetivo é a redução do tempo de desenvolvimento do produto para que, não somente, as necessidades por baixos tempos de ciclo de vida sejam atendidas, mas também, a necessidade de redução dos altos custos causados pela utilização de protótipos físicos. Estudos experimentais sobre protótipos físicos são caros e requerem uma grande quantidade de tempo. Prototipação Virtual é uma nova forma de definir uma simulação dinâmica. Se escolhas do projeto mecânico e eletrônico podem ser testadas antes da montagem de um protótipo físico, o tempo de mercado é reduzido e o conhecimento do sistema é aumentado, obviamente quando a simulação retrata de forma confiável a realidade e a ferramenta de simulação é relativamente simples de utilizar para pessoas de diferentes áreas (Ferreti, 2004).

No geral, os protótipos são usados para três propósitos principais:

- Comunicação;
- Desenvolvimento planejado;
- Testes e verificações.

Baxter (1995 apud Zorriassatine, 2002) classificou os protótipos em três categorias:

- Protótipos estruturais, usados geralmente para validar a aparência e forma;
- Protótipos funcionais, para validar o comportamento;
- Protótipos funcionais e estruturais criados para examinar os problemas de produção e pré-produção.

Por quê considerar o uso de protótipos não-físicos?

Segundo Zorriassatine (2002) existem várias respostas para essa pergunta. Pode ser caro em termos de custos e tempo, a criação de protótipos físicos. Uma vez realizados, protótipos físicos são difíceis ou impossíveis de serem alterados. Por exemplo, Böhm (1994) mostra o gráfico do ciclo de desenvolvimento de um produto, figura 9:

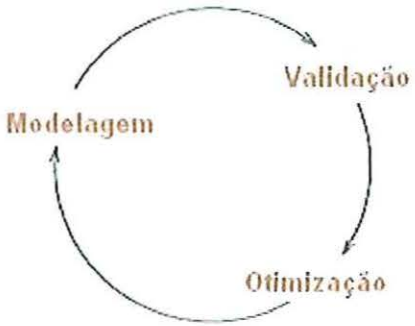

Figura 9 - Ciclo de desenvolvimento de um produto (Böhm 1994).

De uma maneira mais detalhada, Li (2003) mostra na figura 10 o procedimento normal de desenvolvimento de um produto:

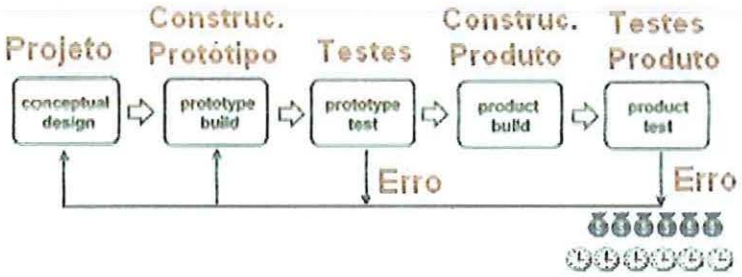

Figura 10 - Procedimento de desenvolvimento de um produto (Li 2003).

Pode-se perceber pela figura acima que muito tempo e dinheiro são desperdiçados quando erros são encontrados em testes sobre o produto. Isso endossa o uso de protótipos na fase de desenvolvimento. Porém, algum tempo e dinheiro também são desperdiçados na criação e testes de protótipos físicos, o que sugere fortemente o uso da PV. Segundo Choi (2004), o propósito da PV é exatamente esse, ou seja, reduzir as repetições do ciclo de testes do desenvolvimento de protótipos.

Choi (2004) define a PV como o processo de utilizar um protótipo digital, no lugar de um protótipo físico, para testes e validação de características específicas de um

produto ou processo. É freqüentemente realizado num ambiente de RV que forneça simulação estereoscópica 3D, com o uso de dispositivos de entrada e saída inovadores.

Além dos problemas mostrados acima acarretados pela utilização de protótipos físicos, Ovtcharova (1994) mostra as tradicionais limitações desse tipo de protótipo, que podem ser evitadas com o uso da PV:

- Peso;
- Medidas e
- Tratamento ou preparação dos materiais a serem utilizados.

No trabalho de Vale (2003) é citado um exemplo da diminuição de gastos proporcionada pela utilização de protótipos virtuais ao invés de protótipos físicos: num sistema de trânsito virtual para testes que utilizariam um protótipo físico repetidamente, a economia com combustível e desgastes do protótipo (pneus, junções, etc) seriam significativas. Para testes de colisão de veículos, também, os Protótipos Virtuais se mostrariam muito efetivos e econômicos.

No exemplo citado acima, a contenção de gastos não é a característica mais importante dos Protótipos Virtuais em relação aos reais. A segurança dos usuários na utilização de um Protótipo Virtual é um fator que pode justificar o uso dessa técnica.

Usando métodos corretos, testes virtuais podem ser implementados, para que o reconhecimento do maior número de falhas no desenvolvimento do produto e, para essas serem eliminadas o mais cedo possível do ciclo de vida de produção. Isso é claro, ajuda a minimizar os gastos causados que uma falha no planejamento pode ocasionar e que pode piorar caso essa falha seja reconhecida apenas em estágios avançados do desenvolvimento. A pior situação conhecida é quando o produto tem de ser totalmente retirado do mercado.

Bullinger (2003) exemplifica na figura abaixo, alguns tipos de protótipos virtuais. Seu trabalho foi direcionado para a prototipação de veículos.

A figura 11 mostra alguns diferentes tipos de protótipos. Os mais comuns são protótipos virtuais que podem ser manipulados com um monitor, um mouse ou um teclado. Esses não suportam qualidades táteis. Para que as sensações dos usuários, nas interações com o ambiente, sejam mais realistas, é necessário utilizar dispositivos de controle "reais", que proporcionem sensações táteis.

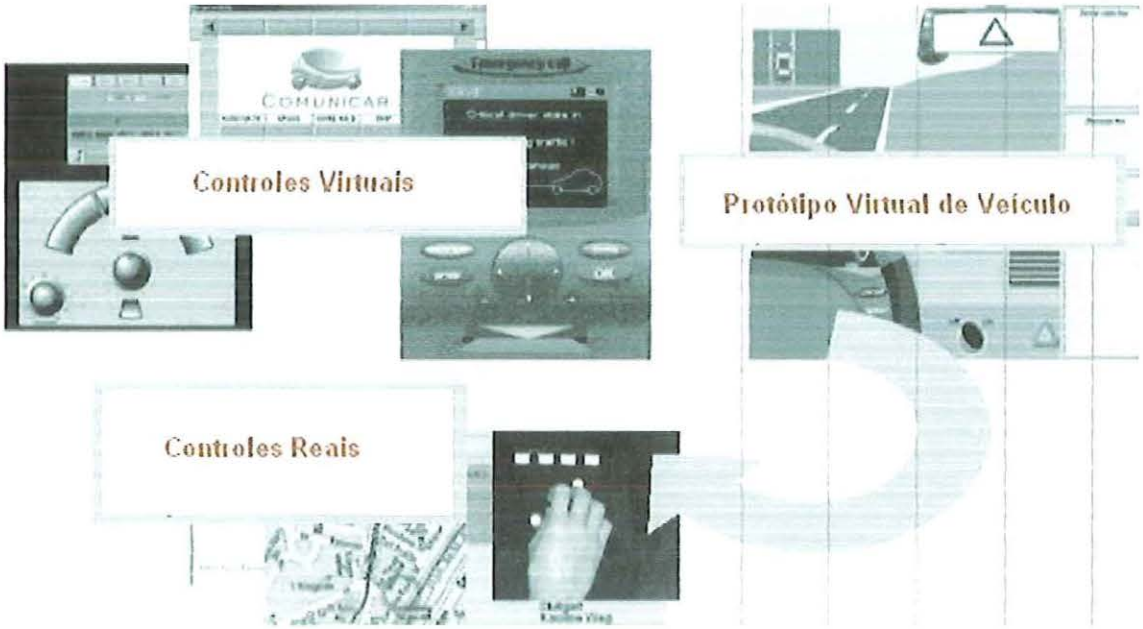

Figura 11 – Exemplos de Protótipos Virtuais (Bullinger 2003).

A figura 12 mostra os principais componentes de um sistema para Prototipação Virtual e Testes:

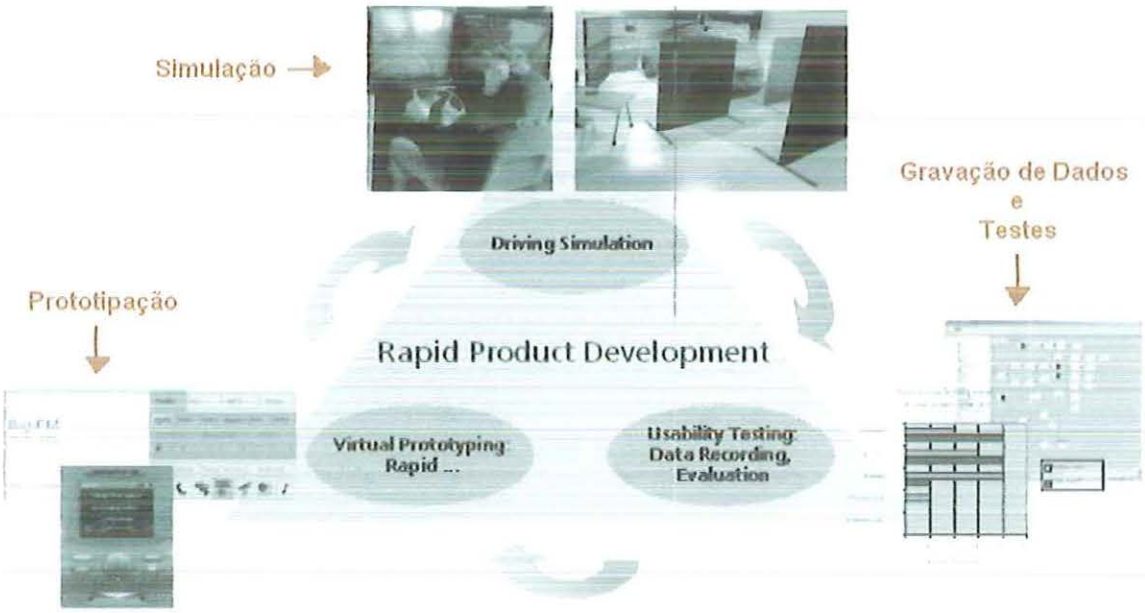

Figura 12 – Principais componentes de um sistema para Prototipação Virtual (Bullinger 2003).

O rápido aumento do poder computacional criou condições para grandes cálculos numéricos, permitindo que protótipos virtuais substituíssem a fase de geração de protótipos físicos no processo de desenvolvimento do produto (Raneda, 2003).

O aumento constante da complexidade e do tamanho dos sistemas digitais reduziu a efetividade das antigas metodologias de desenvolvimento baseadas em protótipos físicos. A prototipação via simulação deve ser usada para reduzir os custos de desenvolvimento e diminuir o tempo gasto para o produto chegar ao mercado, principalmente para o desenvolvimento de grandes sistemas digitais (Williams, 2003).

2.2.3.5 Prototipação Virtual de Veículos e Robôs Móveis

No campo do desenvolvimento de veículos, a competição gera alguns objetivos, como: diminuir o tempo levado para o produto chegar ao mercado, criar projetos inovadores e diminuir os preços dos veículos. Esses objetivos forçam os desenvolvedores a usarem novas metodologias de desenvolvimento. As ferramentas de computação para a Prototipação Virtual têm progredido muito nos últimos anos. Elas são amplamente utilizadas para modelagem e simulação do movimento dinâmico de sistemas de veículos complexos. O trabalho com a tecnologia de protótipos virtuais tem mostrado muito potencial para melhorar o processo de desenvolvimento do produto (Stribersky, 2002). Durante a última década, houve muitas pesquisas sobre PV, especialmente em grandes indústrias de automóveis e aeronaves (Ma, 2004). Na maioria dessas aplicações os modelos CAD geralmente são criados em ferramentas especializadas, como por exemplo, o AutoCad.

No caso da prototipação de robôs móveis, mesmo que o principal objetivo seja a robótica real, geralmente é muito adequado realizar simulações antes de executar tarefas em robôs reais (Oliver Michel, 2004). Isso se deve ao fato de simulações serem mais fáceis de configurar, mais baratas, mais rápidas e mais convenientes para uso. A construção de novos modelos de robôs e sua configuração torna possível o uso de grandes algoritmos de computação que demorariam anos para rodar em micro-controladores de robôs reais, como algoritmos genéticos.

No próximo capítulo serão descritas as características e funcionalidades de três ferramentas de desenvolvimento virtual e será criada uma tabela comparativa sobre estas ferramentas.

CAPÍTULO III -FERRAMENTAS DE DESENVOLVIMENTO VIRTUAL

A prototipação virtual tem uma série de ferramentas ao seu dispor. Basicamente, elas podem ser divididas em ferramentas genéricas e ferramentas dedicadas. Linguagens de programação como Java, JavaScript e C++ são ferramentas genéricas. Apesar de possuir a vantagem de serem universais, suas principais desvantagens são: a necessidade de um conhecimento satisfatório em programação, pelo desenvolvedor e o tempo gasto com a implementação, quando não existem protótipos implementados. Ferramentas específicas funcionam com o auxílio da orientação a objetos e tornam possível a criação de interfaces sem conhecimento de programação. Essas são, no entanto, menos universais.

Outro aspecto da PV que começa a ser estudado é mostrado no trabalho de Belanović (2004) que diz respeito à uma ferramenta de geração automática de protótipos virtuais.

Existem no mercado diversas ferramentas disponíveis para o desenvolvimento de ambientes virtuais. Como os diferentes tipos de aplicações de RV diferem bastante em suas necessidades, é importante a realização de um estudo detalhado sobre as características dessas ferramentas, antes da implementação de um projeto.

A princípio foram estipuladas três ferramentas a serem avaliadas neste trabalho:

- 3DCANVAS 6.5.1.0 da Amabilis Software (www.amabilis.com);
- WORLDUP 3 da Sense8 (http://sense8.sierraweb.net/index.html);
- EON STUDIO 4.0 da EON Reality Inc.

Essas ferramentas foram escolhidas dentre outras devido à sua disponibilidade. A ferramenta 3DCanvas foi escolhida para ser analisada devido à disponibilidade de

uma versão freeware. O software WorldUp e EON Studio foram analisados devido à sua disponibilidade no laboratório da Escola de Engenharia de São Carlos (EESC).

3.1. 3D Canvas

O 3DCanvas é uma ferramenta de desenvolvimento 3D que possibilita aos usuários criar, animar e interagir com objetos 3D. A Amabilis possui uma versão *freeware* do *software* 3DCanvas disponível para *download*. Esse trabalho limitou-se a testar apenas essa versão, por questão de disponibilidade do *software*.

A versão *freeware* do 3DCanvas permite apenas a criação de animações 3D simples, sem capacidade de importar gráficos de outros modeladores de geometria 3D. Existem mais duas opções do *software* 3DCanvas disponíveis, mas ambas requerem um custo para sua aquisição. A versão PLUS permite importar arquivos do AutoCad, 3DStudio entre outros, e também, exportar arquivos 3DStudio entre outros. A versão PRO fornece algumas opções a mais que a versão PLUS, como importar e exportar arquivos VRML e *WorldToolkit*. Somente a versão PRO permite a utilização de scripts, portanto na versão *freeware* não se pode interagir com a cena.

Os requisitos de sistema para a utilização do 3DCanvas são:

- Processador Pentium II 300 MHz;
- Windows 95, 98, 2000, ME, XP;
- 25 Mb de HD;
- 64 Mb de RAM;
- DirectX versão 8.0 ou superior;
- Placa gráfica totalmente compatível com directX com no mínimo 16 MB;

3.1.1 DESCRIÇÃO DO SOFTWARE

É composto por um painel de componentes, uma cena, uma barra de *status*, um controle de navegação, um editor de objetos, um painel de materiais, uma barra de ferramentas de edição, um menu, uma árvore de hierarquia e uma ferramenta para animação.

Para adicionar geometrias à cena basta arrastá-las até a mesma. Clicando-se em cima das geometrias pode-se editá-las em um menu aberto automaticamente

(redimensionar, mover ou rotacionar). Há possibilidade de adicionar texturas às geometrias pela biblioteca de texturas (*texture library)*.

A barra de *status* mostra a posição do objeto na cena e seu número de polígonos.

Pelo menu *view* pode-se mostrar a barra de animação. Nessa ferramenta existe um botão para criação de vídeos em formato AVI de forma simples. Essa ferramenta funciona baseada numa barra de *slide* onde o usuário define as posições dos objetos na cena, de acordo com o tempo da animação. Por exemplo, numa animação de dez segundos, o centro da barra de slide mostrará a cena no tempo 5 segundos da animação, assim como o início da barra mostrará a cena inicial da animação.

3.1.2 MODELAGEM GEOMÉTRICA

O 3DCanvas é um *software* que oferece diversas funções especiais para modelagem de geometrias 3D, podendo assim minimizar, ou mesmo dispensar, o uso de uma ferramenta especial para a modelagem geométrica, como o 3DStudio ou o AutoCad. Algumas dessas funções são: Deformação de objetos, Suavização, *Bend* (curvar um objeto), Aplicação de materiais, Dimensionamento (*scale*), Redução do número de pontos, Intersecção, União, Clonagem etc. Para utilizar a deformação de objetos, por exemplo, basta mostrar a barra *Operations* do menu *Tools* e, feito isso, clicar no botão *Warp*. Feito isso, são exibidas duas telas com a imagem do mesmo objeto. Uma delas será a imagem do objeto deformado e a outra será a imagem do objeto normal. Essas funções não serão explicadas de maneira detalhada, pois esse não é o foco deste trabalho.

Além de funções para a modelagem geométrica, o 3DCanvas possui funções úteis para a animação de uma cena. Um exemplo é a função para "emissão de partículas", ou *Particle Emission*. Com essa ferramenta é possível controlar a freqüência de geração de partículas e sua duração (somente para as versões PLUS e PRO).

3.1.3 OUTRAS CARACTERÍSTICAS DO 3DCANVAS

- Possui aproximadamente 30 figuras geométricas tridimensionais, básicas, prontas, entre elas: esfera, cubo, cilindro, cone e *torus*;
- Diversos tipos de luzes, entre elas: ambiente, direcional, pontual, *área* e *spot*;
- Cinemática inversa;
- Fontes de texto 3D;

- Ferramentas de pintura;
- Editor de faces e pontos;
- Renderização *smooth, flat* e *wireframe*;
- *Background* de cena;
- *Undo* e *Redo* de todos os passos da modelagem;
- Cena hierárquica;
- Sombras;
- Operações restritas por eixo;
- Aceleração de *hadware* usando *Microsoft Direct3d;*
- Renderização *Real-Time;*

Características disponíveis apenas na versão PRO:

- Importa 13 tipos de arquivos (ver tabela abaixo);
- Exporta 17 tipos de arquivos (ver tabela abaixo);
- Estruturas de animação: padrão, cilíndrica, esférica e *rounded*;
- Manipulação dos *key-frames* da animação;
- Acesso ao fórum *Moderated Support;*
- *Plug-ins;*
- Operações Booleanas;
- Mais opções de modelagem;
- *Help On-line* completo;
- Operações de animação;
- Mais figuras geométricas básicas;
- Suporta *scripts*;
- Ferramenta para construção de terrenos;
- Exporta animações como uma série de imagens (para, por exemplo, ser usada em programas de animações GIF);
- Refração.

Na tabela 2 são listados os tipos de arquivos usados no 3DCanvas:

Tabela 2 - Tipos de arquivos usados pelo 3DCanvas

Arquivo	Extensão	Importa	Exporta
AutoCad	DXF	SIM	SIM
3DStudio	3DS	SIM	SIM
DirectX	X	SIM	SIM
BioVision Motion Capture	BVH	SIM	*NÃO*
LightWave	LWO	SIM	SIM
Neutral Object Format	NOF	SIM	SIM
MD2	MD2	SIM	SIM
Raw	RAW	SIM	SIM
RenderWare	RWX	SIM	SIM
TrueSpace	COB	SIM	SIM
VRML	WRL	SIM	SIM
Alias WaveFront	OBJ	SIM	SIM
World Toolkit	NFF	SIM	SIM
POV Ray	POV	*NÃO*	SIM
RenderMan	RIB	*NÃO*	SIM
MS Train Simulator	-	*NÃO*	SIM
MS Flight Simulator	-	*NÃO*	SIM
ShockWave 3D	W3D	*NÃO*	SIM

3.1.4 PROGRAMANDO A CENA (SCRIPTS)

Os *scripts* em 3DCanvas utilizam a linguagem *VBScript* ou *JScript*. Não é possível criar objetos dinamicamente. Também não é possível aplicar alguns tipos de operações sobre objetos, como por exemplo, *smooth* (suavização).

Há dois tipos de *scripts* em 3DCanvas: o de propósito geral e o de animação. Um exemplo de *script* de propósito geral é a criação de uma mensagem em uma janela:

```
Language = VBScript

Sub Main (CanvasApplication)
MsgBox "Hello World!"

End Sub
```

Os *scripts* de animação servem para animar Grupos, Objetos, Luzes e as Câmeras de animação. Um exemplo desse tipo de *script* é a animação de um Grupo ao longo do seu eixo Y:

```
Language = VBScript

Sub AnimateGroup (Time, CanvasGroup)
   CanvasGroup.SetPosition Nothing, Time, 5, Time, 5
End Sub
```

Os arquivos texto dos *scripts* das aplicações de 3DCanvas são salvos na pasta de *scripts* do *software*. A primeira linha de cada *script* deve indicar a linguagem usada.

No 3DCanvas PRO, além de *scripts*, *plug-ins* podem ser utilizados para aumentar a funcionalidade da aplicação.

3.2 WorldUp

O WorldUp faz parte de uma geração de ferramentas 3D para simulação gráfica em tempo real. Ele combina várias tecnologias dentro de um ambiente integrado que permite ao usuário desenvolver e construir simulações complexas. Com o WorldUp o usuário pode desenvolver um protótipo em um tempo menor do que utilizando bibliotecas tradicionais de desenvolvimento. Simulações que levavam semanas para serem construídas, podem ser realizadas em questão de horas.

Apesar de não ser uma ferramenta *freeware*, o WorldUp possui várias características que motivam o seu uso no desenvolvimento de Ambientes Virtuais. Uma de suas principais vantagens é a sua interface de aplicação extremamente fácil e produtiva, principalmente pela organização da sua barra de classes, orientada a objetos.

Todos os tipos de objetos possíveis de serem criados no WorldUp estão relacionados nessa barra de classes, de forma hierárquica, o que facilita a visualização do conteúdo, e o entendimento da cena. Para a criação de uma sub-classe ou de um objeto, basta o clique de um botão (*new_object or new_sub-class*). Outra característica interessante da orientação a objetos, e que auxilia no desenvolvimento de um sistema virtual, é a herança. A herança permite que uma sub-classe herde todas as características de uma outra, simplificando, por exemplo, a criação de tipos de objetos diferentes, mas que possuam características semelhantes.

Outra vantagem do WorldUp, é a facilidade de acoplamento de *scripts* à objetos da cena. *Scripts* são arquivos que contém o código (sintaxe *basic* no caso do WorldUp) referente ao comportamento dos objetos dinâmicos da cena. Cada objeto no WorldUp

possui uma tabela de propriedades. Uma linha dessa tabela é referente ao *script* relacionado ao objeto. Basta entrar com o caminho do arquivo texto do *script* nessa linha, para que o objeto em questão seja controlado pelo código escrito no arquivo. O *script* é lido e executado a cada *frame* pelo sistema. A sintaxe Basic é muito simples, o que facilita bastante a implementação e entendimento de scripts do WorldUp.

3.2.1 VANTAGENS

Abaixo são mostradas algumas características da ferramenta WorldUp, que justificam seu uso:

- Fornece suporte a vários tipos de dispositivos (som, visão, navegação etc.);
- Importa arquivos de vários formatos, como: VRML, 3DStudio, Autocad etc;
- Capacidade de disponibilizar o Ambiente Virtual desenvolvido na Internet;
- Software orientado a objetos;
- Modelador de geometrias próprio.

3.2.2 MODELAGEM GRÁFICA

O software WorldUp possui um modelador geométrico (*WorldUp Modeler*) disponível junto com a ferramenta de desenvolvimento virtual. Além disso, na própria barra de tipos da ferramenta, existem tipos geométricos, como a esfera, o cilindro, o retângulo e o texto 3D, que permitem a rápida criação desses tipos de objetos, bastando apenas o clique de um botão e a configuração de suas propriedades, para disponibilizá-los na cena virtual.

No WorldUp é possível ligar e desligar a renderização. Se o ambiente possuir um nível muito alto de detalhes gráficos, pode ser útil desligar a renderização para aumentar o desempenho do sistema, tanto para aplicações exteriores ao WorldUp, quanto para dentro do mesmo.

O software fornece alguns meios diferentes de renderização:

- **WireFrame**: As geometrias são renderizadas como pontos e linhas, sem possuir texturas ou sombras. Esse é o tipo de renderização mais rápida.
- **Shaded**: Coloca sombras em cada polígono das geometrias, de acordo com a intensidade de luz incidente e com o material aplicado.

- **Fully Textured**: Renderiza geometrias com suas texturas e sombras.
- **Texture Perspective**: Mostra as geometrias com texturas e dimensiona as texturas de acordo com a posição do *viewpoint*.

O WorldUp não permite um controle completo de renderização do ambiente, dificultando, assim, o desenvolvimento de um sistema virtual que possua um bom nível de detalhes e, ao mesmo tempo, um desempenho satisfatório. Existem algumas formas de minimizar esse problema. O WorldUp fornece, em sua barra de classes, um objeto chamado *switcher*, e outro chamado *LOD (Level Of Detail)*, que podem ser utilizados para o controle de renderização do ambiente.

O *switcher* é um objeto que deriva da classe de objetos *Group*, que permite a integração de vários objetos para, por exemplo, a movimentação de todos em conjunto. O *switcher* permite com que o desenvolvedor decida qual dos objetos, integrantes do *switcher* em questão, será renderizado, em um determinado momento.

O *LOD* permite que o desenvolvedor decida qual dos objetos, integrantes do *LOD* em questão, será renderizado, baseado na distancia do *LOD* ao *viewpoint*. Isso possibilita que um objeto tenha uma aparência mais simples quando estiver longe do *viewpoint* e tenha uma aparência mais detalhada quando estiver perto. Isso torna os objetos menos pesados ao sistema, não prejudicando o desempenho da máquina com renderizações de pouca importância.

Ainda sobre o desempenho do sistema, o WorldUp fornece uma série de estatísticas referentes à execução da aplicação, para que o desenvolvedor possa manter um certo "controle à distância" do que acontece com o desempenho de sua aplicação.

3.2.3 ESTRUTURA DO SOFTWARE

É formado por quatro janelas principais:

1. *Properties Browser*: contém as propriedades das classes e objetos.
2. *Scene Graph Browser*: ou árvore de cena, mostra a hierarquia dos objetos que compõem a cena, onde grupos de objetos podem ser movimentados entre *frames*.
3. *ViewPoint's*: ou janela de simulação, mostra a renderização do ambiente em tempo real. Nela podemos fazer alterações nos objetos em tempo de execução.

4. *Type Browser*: mostra a hierarquia das classes que compõem o universo virtual. Há possibilidade de criação de novas classes.

3.3 EON Studio

O *EON Studio* é uma ferramenta de desenvolvimento 3D que possibilita a usuários, de diferentes níveis de experiência, construir de forma fácil e rápida ambientes virtuais interativos, sem a necessidade de conhecimento em programação. Para aplicações na área de arquitetura, *marketing* ou treinamento virtual, o *EON Studio* traz benefícios únicos (EON REALITY, 2005).

A *EON Reality* tem disponível em seu *site*, uma versão *freeware* do *software EON Studio 4.0*. Essa versão possui todas as funcionalidades da versão registrada, mas é válida apenas por duas semanas. Mesmo a versão *freeware* requer um registro que deve ser feito pelo próprio *software*.

Assim como o *software WorldUp*, o *EON Studio* tem uma interface que facilita muito o trabalho do programador de ambientes virtuais. Abaixo é mostrado na figura 13, com destaque, os componentes principais de sua estrutura:

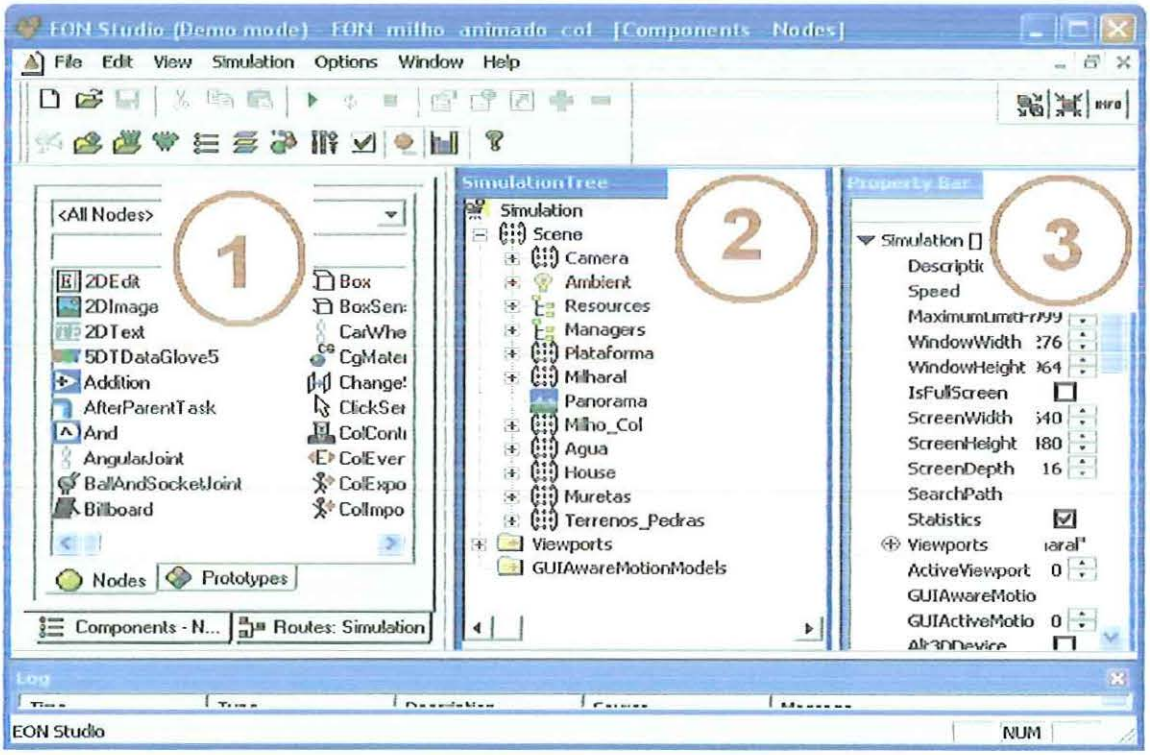

Figura 13 – Ambiente de trabalho do EON Studio.

Na figura acima, o número um indica a barra de nós e protótipos do software, o número dois mostra a árvore de simulação e o número três indica a barra de propriedades dos objetos. Essas três estruturas serão comentadas abaixo.

O EON Studio 4.0 é capaz de importar os seguintes tipos de arquivos: VRML 2.0 (incluindo *KeyFrames*), 3D Studio (.3ds), Lightwave (.lw), OpenFlight (.flt), Alias Triangle (.tri), DXF ASCII/Binary (.dxf), IGES v5.3 (.igs, .iges), Pro/Engineer .SLP (.slp), SoftImage (.hrc, .dsc), Stereo Lithography .STL (.stl), TrueSpace (.cob, .scn), USGS DEM (.dem), WaveFront (.obj).

Esse *software* possui uma janela com mais de cem nós de funcionalidades (mostradas na figura acima como número um), entre elas: modelos para movimento, sensores, operações etc. Esses nós são divididos em *Default Nós, Agent Nós, Assembly Nós, Base Nós, Collision Detection Nós, Flow Nós, ForceFeedback Nós, MotionModel Nós, Sensor Nós, Showroom Nós, SpecialFX Nós.* Além dos nós especiais para imersão, como: *Peripherals Nós* e *ClusterVisualization Nós.*

Outras características importantes do EON Studio 4.0 são:

- Facilidade de criação de códigos de programação, pois o *software* trabalha com dois tipos simples de linguagem, o *Java-Script* e o *VB-script*;
- Os efeitos visuais do EON Studio são de alto nível, pois o *software* possui algoritmos prontos para transparências, sombras, relexos etc;
- É possível integrar seu ambiente virtual com alguns *softwares*, como Internet Explorer, Netscape, Macromedia Director, Shockwave, Flash, PowerPoint, Word e Visual Basic;
- Para ambientes virtuais que necessitem simular efeitos físicos reais, o *software* possui *nós* preparados para implementar detecção de colisão, gravidade, dinâmicas de corpo rígido etc;
- É possível distribuir os ambientes virtuais criados na Internet ou rodá-los em computadores independentemente do *software* EON Studio. Existem *plug-ins* disponíveis para esses tipos de tarefas;
- Para aliviar o problema dos tamanhos de arquivos da Realidade Virtual existem no *software* algoritmos de compressão e subdivisão.

3.3.1 MODELAGEM GRÁFICA

Diferente do *software* WorldUp, o EON Studio não possui um modelador geométrico disponível junto com a ferramenta de desenvolvimento virtual. Mas, em sua barra de *prototypes* existem tipos geométricos, como esfera, cilindro, retângulo etc, que permitem a rápida criação desses tipos de objetos, bastando apenas o clique de um botão e a configuração de suas propriedades, para disponibilizá-los na cena virtual. Porém, não é viável utilizar esse tipo de geometria, pois elas não se adaptam a certas funcionalidades do EON Studio. O mais adequado é importar geometrias de *softwares* para modelagem 3D ou utilizar as geometrias prontas (*meshes* de extensão ".X") do diretório *MediaLibrary* que acompanha o *software*.

O EON Studio permite alterar a qualidade de renderização da cena e configurar a utilização de placas de aceleração 3D pelo menu *global rendering properties*. Além da configuração global, podem-se definir qualidades de renderização separadamente para cada objeto, alterando configurações de preenchimento, sombras e luzes.

Para auxiliar na velocidade e na qualidade de renderização do ambiente, o EON Studio fornece em sua barra de classes um objeto chamado *switcher*, outro chamado *PowerSwitcher* e outro chamado LevelOfDetail que podem ser utilizados para o controle de renderização do ambiente.

Tanto o *Switcher quanto o PowerSwitcher* são objetos que permitem a integração de vários objetos, para que esses possam ser mostrados na cena, somente em momentos necessários.

O *LevelOfDetail* permite que o desenvolvedor decida qual dos objetos, integrantes do *LevelOfDetail* em questão, seja renderizado, baseado na distancia do *LevelOfDetail* ao *viewpoint*. Isso possibilita que um objeto tenha uma aparência mais simples quando estiver longe do *viewpoint* e tenha uma aparência mais detalhada quando estiver perto. Isso torna os objetos menos pesados ao sistema, não prejudicando o desempenho da máquina com renderizações de pouca importância.

O EON Studio mostra a velocidade da simulação em *frames* por segundo, na barra de título da cena. Assim o programador pode verificar a complexidade de seu ambiente para que este não se torne inviável. Uma velocidade adequada para uma simulação seria de trinta *frames* por segundo.

3.3.2 ADIÇÃO DE GEOMETRIAS

O EON Studio disponibiliza dois métodos para adição de geometria à cena:

1- Protótipos;

2- Importação de geometrias.

Os protótipos de geometrias do EON Studio são: circunferências, cubos, cilindros, cones, esferas, planos, *anéis (torus)* e *chaleiras (teapot)*. Para utilizá-los basta arrastá-los da barra de protótipos à árvore de simulação.

O segundo método para adição de geometrias à cena (importação de geometrias) pode ser usado tanto para adicionar geometrias existentes na pasta *MediaLibrary* (biblioteca de sons e figuras) do próprio *software*, como para adicionar geometrias criadas pelo próprio programador em algum *software* de edição 3D. Os *softwares* de edição 3D usados nesse projeto foram o AutoCad (extensão .DXF) e o 3DStudio MAX (extensões .3DS e .MAX).

Uma outra maneira de importar arquivos de geometria para o *software* EON Studio é utilizando o *nó Mesh* (malha). Esse nó permite a adição de arquivos com extensão ".X" à árvore de simulação. A pasta *MediaLibrary*, fornecida com o próprio *software*, possui um diretório chamado "X", onde estão localizadas várias geometrias com essa extensão.

Quando uma figura é importada ao EON Studio pelo menu "Import", sua geometria é armazenada no *nó Mesh2*, que possui propriedades diferentes do *nó Mesh* descrito acima.

Para visualizar os objetos adicionados deve-se rodar a aplicação, clicando no botão Play. Dessa forma podemos fazer alterações em suas propriedades e visualizar os resultados de forma imediata.

A adição de texturas aos objetos criados pode ser realizada de maneira simples, bastando para isso a utilização do nó *Texture* ou *Texture2*.

3.3.3 ESTRUTURA DO SOFTWARE

Existem quatro tipos de janelas disponíveis para criação de aplicações em EON Studio:

1- *Janelas de Construção (Construction Windows):*

- *Árvore de Simulação (Simulation Tree):* árvore hierárquica de nós pertencentes à aplicação. Possui alguns nós padrões que sempre são

adicionados automaticamente ao início de cada projeto, como nós para iluminação, movimentação (*walk*), visualização etc.;

- *Janela de Rotas (Routes Window):* é um editor gráfico para criação de relacionamento (interação) entre nós. Sua função é conectar *nós*, por meio de linhas de tarefas criadas pelo programador, para definir seus comportamentos. Na janela de rotas, cada nó recebe uma variável de entrada e uma de saída, para que suas linhas de tarefas possam ser ligadas.

2- *Biblioteca (Library Window):*

- *Janela de Componentes (Components Window):* contém todos os nós e *protótipos* do EON Studio.

3- *Janelas de Apoio (Help Windows):*

- *Butterfly Window:* mostra as rotas (*routes*), de maneira mais clara, para um *nó* selecionado;
- *Find Window:* localiza *nós* na *simulation tree*;
- *Log Window:* mostra as operações internas do EON Studio.

4- *Display Window*

- *Simulation Window:* usada para rodar as aplicações.

3.3.4 SCRIPTS

Para a implementação de interatividade dos *nós* de uma aplicação, o EON Studio possui, além da janela *routes*, um nó chamado *Script*. Esse *nó* permite a adição de códigos de programação em linguagem *VBScript* ou *JScript*. Para utilizá-lo basta arrastá-lo da janela *Components Window* para a *Simulation Tree*. Suas propriedades podem ser acessadas pelo clique do botão direito do *mouse* e em seguida a escolha da opção *properties*. Nessa janela de propriedades é possível adicionar variáveis de entrada, saída, ou mesmo entrada e saída. Seus tipos podem ser *boolean, integer, float*, vetores, imagens, cores, *string* etc.

3.3.5 DETECÇÃO DE COLISÃO

Existem dois métodos para a implementação de detecção de colisão em EON Studio 4.0:

Primeiro Método: Utiliza apenas o nó Collision. Detecta colisão entre grupos de meshs, sem restrição de como estes são compostos.

Segundo Método: Mais rápida que a primeira alternativa, essa opção usa dois *nós* para implementar colisão: *ConvexCollisionManager* e *ConvexCollision*. Os *nós ConvexCollision* devem ser colocados embaixo dos *frames* referentes aos objetos que participarão da colisão. Quaisquer dimensionamentos de frames relacionados com objetos da colisão, serão ignorados por esse método. Os dimensionamentos devem ser feitos diretamente nos *meshs* das geometrias. Outro problema desse método ocorre quando vários vértices de um objeto sobrepõem outro objeto (isso ocorre, geralmente, quando são executados movimentos muito rápidos). Uma solução para o problema pode ser o aumento da propriedade *CollisionTolerance*, ou o aumento da geometria física e diminuição da *ConvexGeometry*.

3.3.6 DINÂMICA DE CORPOS RÍGIDOS

Uma importante característica do EON Studio é sua capacidade de trabalhar com dinâmicas de corpos rígidos (*Rigid Body*). Os *nós* especiais para esse tipo de aplicação são: *RigidBody, RigidBodyManager, Force, Torque, MouseDrag, Spring.*

O *nó RigidBody* deve ser adicionado embaixo de *frames* que possuam geometrias, para que estas possam assumir propriedades físicas dentro da simulação, como peso, densidade etc. Esse *nó* permite a especificação da densidade da geometria, assim como seu coeficiente de elasticidade numa colisão. O tamanho da geometria é também um fator importante.

Para a implementação de colisão entre corpos rígidos, o EON Studio permite apenas o uso da colisão Convexa ou *ConvexCollision.*

O *nó Force* é usado para aplicações de forças lineares entre corpos rígidos. Ele permite a especificação da intensidade da força para cada eixo (X, Y, Z) e do ponto de aplicação.

O *nó Torque* funciona de maneira semelhante ao *nó Force*, porém aplicado a forças angulares.

O *nó Spring* funciona como uma mola entre dois corpos rígidos. Ele permite a especificação da constante de elasticidade K entre outras características.

Esse tópico detalhará o uso de cada um desses *nós* e sua aplicação ao projeto da plataforma virtual agrícola móvel.

A detecção de colisão da dinâmica de corpos rígidos do EON Studio é implementada através dos *nós: ConvexCollision* e *ConvexCollisionManager*.

Os objetos rígidos, não são afetados por *nós* normais, como por exemplo: *Motion* e *Rotate*.

Os dois principais *nós* que gerenciam uma aplicação de corpos rígidos são os *nós RigidBody* e *RigidBodyManager*. Suas propriedades são descritas abaixo:

RigidBodyManager: Gerencia um conjunto de corpos rígidos (*RigidBody nós*). Calcula e atualiza novos estados (como posição e orientação) dos corpos rígidos. Esse *nó* pode ser colocado em qualquer lugar da árvore de simulação. Uma aplicação pode conter diversos *nós RigidBodyManager*, sendo que cada um deles controlará seus próprios *RigidBodies*. Os corpos rígidos controlados por diferentes gerenciadores não terão interação por colisões. Suas propriedades são descritas abaixo:

- *Gravitation*: O *nó Gravitation* descrito nos tópicos anteriores não tem função nas aplicações de corpos rígidos. Nesse caso, a gravidade aplicada sobre os objetos é controlada pelo próprio *RigidBodyManager* por esta propriedade. Seu valor padrão é o valor real da gravidade da Terra ("-9.82"). Essa força será aplicada a todos os *nós* gerenciados por este *RigidBodyManager*;
- *Linear Damping*: Constante de resistência a movimentos lineares. Seu valor padrão é "0.002". Quanto maior esse valor, mais os corpos rígidos controlados por este *RigidBodyManager* resistirão a aplicação de forças lineares. Seu propósito é manter a estabilidade do sistema. A força *Gravitation* descrita acima também encontrará resistência devido a essa propriedade.;
- *Angular Damping*: Constante de resistência a movimentos angulares. Seu valor padrão é "0.001". Quanto maior esse valor, mais os corpos rígidos controlados por este *RigidBodyManager* resistirão a aplicação de forças angulares. Seu propósito é manter a estabilidade do sistema;
- *Eneabled*: liga e desliga o gerenciador de corpos rígidos;

- *Numerical Integrator:* Método numérico para calcular os movimentos dos corpos rígidos. Nessa versão do *software* (versão estudante 4.0) não há como mudar essa opção;
- *MaximumTimeStep:* Tempo máximo tomado pelo método numérico para computar os estados seguintes. Valores muito altos resultam em instabilidade numérica e comportamento físico incorreto dos corpos rígidos. Valores muito baixos deixam a aplicação muito lenta.

O *nó RigidBodyManager* possui três pastas como filhos: *RigidBodies, Forces* e *CollisionManager*. Embaixo da basta *RigidBodies* devem ser colocadas referências aos *nós RigidBody* por quais esse gerenciador será responsável na aplicação. Na pasta *Forces* devem ser colocadas referências às forças que o gerenciador deve levar em consideração durante a simulação. E, finalmente, na pasta *CollisionManager* deve ser colocada apenas uma referência à um *nó ConvexCollisionManager*, que será responsável por gerenciar as colisões dos corpos rígidos.

RigidBody: Sua principal tarefa é computar propriedades físicas, gerenciar seu estado atual, fornecer informações para o *nó RigidBodyManager* e atualizar a posição e orientação do seu objeto. Suas propriedades são: Velocidade Linear Inicial; Velocidade Angular Inicial; Densidade: Coeficiente de restituição (elasticidade).

O volume, a massa e o centro de massa, também são propriedades do *nó RigidBody*, porém, estas são calculadas automaticamente (devido as características físicas da geometria física ou *PhysicalGeometry*) e não podem ser alteradas.

Esse *nó* possui duas pastas como filhos: *Forces* e *PhysicalGeometry*. Na pasta *Forces* devem ser adicionados os *nós* que aplicarão forças nesse corpo rígido, como por exemplo, um *nó Torque* ou um *nó Force*. Na pasta *PhysicalGeometry* devem ser adicionadas referências a geometrias das quais serão calculadas as propriedades físicas desse corpo rígido. Se essa pasta for deixada vazia, as propriedades físicas desse corpo serão calculadas pela mesma geometria visual do objeto. Essa pasta é muito importante para o desempenho da aplicação, pois um objeto muito complexo (com vários polígonos) pode exigir muito da máquina no momento do cálculo de suas propriedades físicas. Por exemplo, em uma aplicação que possua o corpo rígido de um carro muito detalhado, pode ser interessante que a pasta *PhysicalGeometry* desse corpo contenha uma referência para um objeto mais simples, como um cubo. Esse cubo teria que

possuir o mesmo tamanho do objeto carro, para que as propriedades físicas computadas fossem semelhantes às propriedades físicas calculadas sobre a própria geometria do carro.

Explicando melhor, todo o *nó RigidBody* deve ser colocado na árvore de simulação embaixo de um *Frame* que contenha uma geometria. Feito isso, esse *Frame* se torna um corpo rígido e começam a ser computadas propriedades físicas para seu objeto (geometria). Se esse fato diminuir demasiadamente a velocidade da simulação, uma solução é criar uma geometria física para esse objeto. Essa geometria pode ser adicionada em qualquer lugar da árvore de simulação (é aconselhado que esta se localize próximo ao corpo rígido, por questões de organização). Ela deve possuir um formato semelhante ao objeto original, mas com poucos detalhes. Essa geometria não poderá ser vista na cena, ela servirá apenas para o cálculo das propriedades físicas. O objeto mostrado continuará sendo aquele referente ao *Frame* onde foi adicionado o *RigidBody*. Para que isso aconteça, a propriedade *Hidden* (ou oculto) do *Frame* da geometria física deve ser ligada. Portanto, o objeto visualizado no ambiente não é o mesmo do qual são computadas suas características físicas, mas seu comportamento na cena não será comprometido. A simulação tende a ganhar desempenho quando esse método é utilizado.

3.3.7 PROTÓTIPOS

Semelhantes a sub-rotinas de linguagens de programação ou sub-árvores encapsuladas. Os protótipos devem ser utilizados para facilitar o gerenciamento da simulação. Podem ser incluídos num protótipo objetos e suas respectivas *routes*. Portanto, é possível reutilizar não somente os objetos de uma simulação, mas também, seus comportamentos em uma outra aplicação. Um protótipo pode conter vários outros protótipos filhos.

3.4 Comparação de Ferramentas

Tabela 3 - Comparação de ferramentas

	3D CANVAS FREEWARE	WORLDUP	EON STUDIO 4.0
IMPORTA	DirectX, BioVision, Neutral Obj. Format, MD2, RAW, Wavefront, Worldtoolkit(nff)	VRML, 3DStudio, AutoCad, entre outros	VRML 2.0 (incluindo *KeyFrames*) 3D Studio (.3ds) Lightwave (.lw) Alias Triangle (.tri) DXF ASCII/Binary (.dxf) WaveFront (.obj), entre outros
EXPORTA	POVRay (pov)		
INTERAÇÕES	Não suporta na versão *freeware*	VBScript	VBScript, JScript e janela de Rotas
REQUISITOS DE SISTEMA	Processador Pentium II 300 MHz; Windows 95, 98, 2000, ME, XP; 25 Mb de HD; 64 Mb de RAM; DirectX versão 8.0 ou superior; Placa gráfica totalmente compatível com directX com no mínimo 16MB	Processador Pentium, 200MB de HD, monitor 1024x768, 32 MB RAM, monitor com 16 bits de cores, placa aceleradora OpenGL	Pentium III 1GHz, 256MB RAM, Vídeo NVIDIA GForce3
MODELAGEM	Poucos recursos na versão freeware, porém ótimos	Possui modelador geométrico próprio, podendo	Não possui modelador geométrico próprio.

	recursos nas versões PLUS e PRO.	dispensar o uso de uma ferramenta especial para modelagem 3D.	
OUTRAS FUNCIONALIDADES	Permite somente modelagem e animação	Extensa árvore de funcionalidade, porém com poucos recursos físicos.	Possui recursos para gravidade, massa, colisão etc.
PREÇO	Gratuito	Aprox. $3.500,00	Aprox. $5.000,00

Nesse capítulo foi realizada uma avaliação de três ferramentas de desenvolvimento virtual existentes no mercado e disponíveis para uso no laboratório da EESC. Essa avaliação teve como objetivo selecionar a ferramenta mais adequada para o desenvolvimento de uma Plataforma Virtual Agrícola Móvel (PVAM).

Pela tabela 3 acima é possível perceber que o software EON Studio 4.0 fornece mais rescursos para a prototipação virtual de um veículo, embora seu custo seja elevado e este não possua uma ferramenta própria para modelagem de objetos. Isso se deve ao fato de que a implementação de propriedades físicas a objetos em um ambiente virtual é uma tarefa extremamente complexa, se feita sem o auxílio de nós especiais para esse tipo de funcionalidade. O projeto de um veículo virtual exige que a ferramenta de desenvolvimento possua algumas características, como capacidade de implementação de forças físicas (massa, gravidade, acelerações etc), entre outras. Esse foi o principal motivo que levou à escolha do software EON STUDIO 4.0 para a implementação do projeto. Dessa forma, esse *software* foi escolhido para a implementação descrita no próximo capitulo.

CAPÍTULO IV -PROTOTIPAÇÃO VIRTUAL DE UMA PLATAFORMA AGRÍCOLA MÓVEL

Depois de escolhida a ferramenta de desenvolvimento virtual mais adequada ao projeto, pôde-se iniciar a implementação da plataforma virtual agrícola móvel.

Os dados utilizados para essa implementação foram baseados no ante-projeto VAA e serão mostrados abaixo, juntamente com uma figura, criada no *software* Solid Edge da UGS:

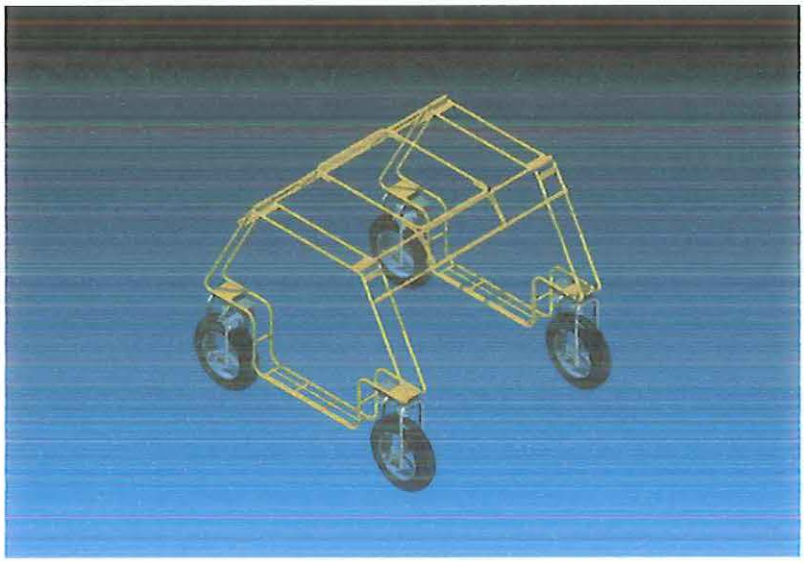

Figura 14. Plataforma Agrícola Móvel

- O Diâmetro das rodas seria de aproximadamente 660 milímetros;
- A distância entre os centros das rodas de uma mesma lateral da plataforma seria de aproximadamente 2000 milímetros;
- A altura da plataforma montada seria de aproximadamente 2040 milímetros;
- A distância entre as laterais da plataforma é um valor re-configurável que varia de 1680 a 2320 milímetros.

Inicialmente, a estrutura de implementação foi dividida em quatro etapas:

1. Montando uma plataforma simples;
2. Movimentando a plataforma;
3. Adicionando gravidade e colisão;
4. Implementação das forças físicas.

Porém, as dificuldades encontradas durante o desenvolvimento do projeto forçaram algumas mudanças nessa estrutura. Essas alterações serão explicadas no decorrer deste capítulo.

4.1 Implementação em EON Studio 4.0

Neste capítulo serão descritas as etapas da fase inicial da implementação da plataforma agrícola móvel. Em seguida, é feito um estudo sobre duas novas ferramentas: WEBOTS e EON Professional. Sendo que esta última foi utilizada para a conclusão da implementação do projeto.

4.1.1 MONTANDO UMA PLATAFORMA SIMPLES

A construção da plataforma virtual simples foi implementada utilizando os três métodos de adição de geometrias, descritos no capítulo de Análise de Ferramentas, para que estes pudessem ser comparados.

Os objetos componentes da plataforma simples foram um cubo e quatro cilindros.

4.1.1.1 A Construção por Protótipos

O primeiro passo executado foi arrastar os objetos necessários (um cubo e quatro cilindros) da barra de protótipos à árvore de simulação, mais especificamente ao nó *Scene* (cena). Dessa forma, os protótipos adicionados se tornaram filhos do nó *Scene* (*frame* raiz de toda aplicação).

Os nós ou protótipos adicionados à árvore de simulação foram nomeados de maneira adequada (que possuísse relação com suas funções), para que a aplicação não se tornasse difícil de entender ou ser modificada, quando esta assumisse um tamanho considerável.

Nesse momento da execução, todos os objetos adicionados à cena estavam sobrepostos, pois estes, quando adicionados, se localizavam na posição "X=0", "Y=0" e "Z=0". Ao alterar a propriedade Posição, foi possível visualizar os cinco objetos na cena.

Tanto o retângulo quanto os cilindros possuíam, nesse momento, tamanhos e orientações inadequadas para a construção da plataforma. Portanto, os próximos passos executados foram o dimensionamento dos objetos, usando a propriedade *Scale*, para que esses assumíssem os formatos desejados e a utilização da propriedade *Orientation* para colocar os objetos nas posições corretas.

Depois de algumas mudanças de posições dos objetos, alcançou-se o resultado final, mostrado na figura 15:

Figura 15 - Plataforma simples criada por protótipos.

O ambiente criado, mostrado na figura acima, não possuía qualquer tipo de interação, além da navegação.

4.1.1.2 A Construção por Malhas (*Meshs*)

A construção da plataforma por Malhas se diferencia da anterior pela necessidade de utilização de nós *Frame*. Isso se deve ao fato que os nós do tipo *Mesh* não possuem *Position* e *Orientation* como propriedades. Sendo assim, todo o *Mesh* deve ser adicionado embaixo de um *nó Frame*, para que este possa ser posicionado corretamente na cena. As cores das geometrias dos *Meshs* devem ser manipuladas por nós *Material* adicionados embaixo de cada nó *Mesh*. A figura16 abaixo mostra a árvore composta por nós *Frame, Mesh e Material*.

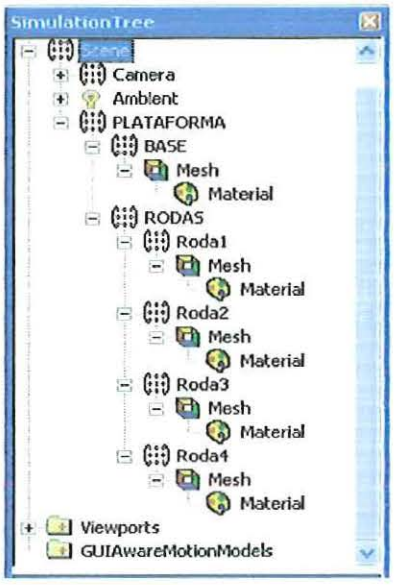

Figura 16 - Árvore formada por *frames, meshs* e *materials*.

Como visto no capítulo de análise de ferramentas, a árvore de simulação é hierárquica, ou seja, todos os nós filhos herdam todas as características do seu nó pai. Desse modo, podemos, por exemplo, mover todas as rodas de uma só vez, alterando apenas as propriedades do *frame* "RODAS". Para aplicações muito grandes, a característica hierárquica da árvore de simulação é de extrema importância, tanto para o entendimento, quanto para o funcionamento da aplicação.

O resultado final da construção por nós *Meshs* não será mostrado, pois a figura é idêntica à figura da construção por protótipos.

4.1.1.3 A Construção por Importação

A importação de um objeto para o EON Studio deve ser realizada pelo menu *Import*. Depois de escolhido o formato do arquivo a ser importado e de configuradas as suas propriedades (no caso desse exemplo todas as propriedades foram padrões), o objeto é criado na árvore de simulação como um *Frame*. A figura 17 abaixo mostra um cubo e um cilindro importados do *software* 3DStudio MAX.

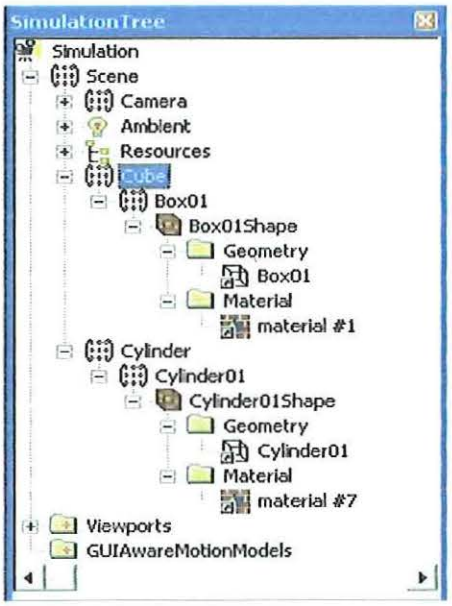

Figura 17 - Árvore resultante da importação de um cubo e um cilindro.

Nesse caso, todos os *nós* filhos de *Cube* e filhos de *Cylinder* foram adicionados à árvore automaticamente. O nó *Resources* (Recursos) também foi criado de forma automática. Esse nó contém todas as geometrias (nós *Mesh2*) e os materiais (nós *Material2*) importados. Não é possível utilizar o nó *Material* para alterar as configurações de cores dos objetos importados. Para alterar essa configuração, as propriedades dos nós *Material2* localizados dentro do *nó Resources*, devem ser alteradas.

Os métodos de posicionamento dos objetos são os mesmos das seções anteriores, ou seja, alterando as propriedades de seus *frames* pais. O resultado final também se assemelha muito ao resultado obtido pelos dois métodos anteriores, portanto a figura não será mostrada.

4.1.1.4 Diferenças dos Tipos de Construções

Nessa fase de adição de geometrias, pôde-se concluir que os nós *Frame* são bastante úteis às aplicações de EON Studio, por causa da característica hierárquica da árvore de simulação. Em aplicações muito grandes, isso pode se tornar um fator decisivo.

O uso de protótipos de geometrias foi descartado, pois foram encontrados alguns problemas quanto ao posicionamento dos objetos na cena. Quando alterados em tempo de execução (com a cena ou janela de simulação aberta), os protótipos mantinham suas

novas posições somente durante a seção atual da aplicação. Quando reiniciada a aplicação, os protótipos voltavam à suas posições originais. Para manter a posição dos objetos na cena, essas características deveriam ser alteradas com a janela de simulação (ou cena) fechada.

Outro detalhe que colaborou para a decisão de descartar o uso de protótipos de geometrias foi a escassez dos mesmos.

Uma observação realizada foi sobre a complexidade das árvores de simulação. A árvore do ambiente criado por importação de objetos se tornou significativamente mais complexa que a árvore do ambiente criado por *Meshs*. Esse fator influenciou na decisão de continuar as próximas fases do projeto utilizando apenas os nós *Mesh* para a adição de geometrias. Porém, era sabido que os arquivos de extensão ".X" aceitos pelo nó *Mesh* são limitados e que a aplicação final, necessitará de objetos mais complexos.

Uma parte desse capítulo explicará a criação de uma plataforma mais complexa em 3DStudio MAX e sua importação ao EON Studio. Os próximos tópicos desse capítulo continuarão a utilizar apenas os nós *Mesh*, por questão de facilidade de uso e peso da aplicação.

4.1.1.5 Aplicação de texturas

Concluída as três etapas de adição de geometrias, na construção da plataforma simples, foi implementada a adição de texturas às rodas da plataforma. Como dito anteriormente, as geometrias da plataforma estão usando os nós *Meshs*, portanto a aplicação de texturas teve de ser implementada com a utilização do nó *Texture*.

MESH → MATERIAL → TEXTURE

MESH2 → MATERIAL2 → TEXTURE2

Os nós *Material* e *Texture* podem ser usados em conjunto, mas nesse caso foi optado por deletar os nós *Material* para a aplicação do nó *Texture*.

Depois de colocados os nós *Texture* embaixo dos nós *Meshs*, as propriedades *Filename* (nome do arquivo) de cada nó *Texture* teve de ser associada a um arquivo de imagem. As texturas utilizadas foram retiradas da pasta *MediaLibrary* do próprio *software*.

4.1.1.6 Criação do Cenário

Um fator que auxiliou muito na parte de modelagem geométrica do projeto foi o *download* (aquisição via Internet) de objetos virtuais 3D, gratuitos. Isso facilitou a criação de cenários, diminuiu o tempo gasto com a modelagem de objetos e aumentou o realismo do ambiente.

Com a utilização do nó *Panorama* descrito na Análise de Ferramentas, pôde-se conseguir um nível de realismo maior rapidamente. Todos os arquivos de textura usados até o momento pertencem à pasta *MediaLibrary* do EON Studio. Essa pasta possui arquivos de textura padrões para o céu, solo e horizonte, além de texturas de ferro, pedras, madeiras etc.

Mais adiante nesse capítulo serão adicionados mais objetos à cena, como uma casa, uma plantação de milho, entre outros.

Por enquanto a plataforma virtual não possui interações com o usuário. No próximo tópico serão implementados movimentos simples via teclado.

4.1.2 MOVIMENTANDO A PLATAFORMA

Nesse tópico será mostrado o funcionamento da janela de rotas (*Routes Window*). Essa é uma ferramenta importante na construção de interações do usuário com a cena. Se usada de maneira correta, ela torna simples a adição de interações aos objetos. Por esse motivo o *software* EON Studio é mais adequado para esse tipo de trabalho. Outros *softwares* disponibilizam apenas a adição de *scripts* para implementação de interações.

Quando é possível trabalhar com rotas e *scripts*, a visualização e o controle da aplicação são mais efetivos. Deve-se tomar cuidado, porém, quando a aplicação ganha uma dimensão muito grande, possuindo diversas rotas. Nesse caso, a utilização correta desse tipo de ferramenta é essencial. Portanto, nesse capítulo será comentada, também, a utilização de *Layers* (camadas) na janela de rotas.

Primeiramente, o nó *Motion* foi adicionado à árvore de simulação como filho do nó Plataforma. Isso se deve ao fato de o nó *Motion* aplicar suas funcionalidades ao seu *Frame* pai.

A idéia inicial deste tópico é movimentar a plataforma de maneira simples, ou seja, pelas setas direcionais do teclado, sem a utilização de dispositivos de realidade

virtual. As rodas da plataforma também serão movimentadas, porém, estas não terão influência no movimento da plataforma.

As propriedades do *nó Motion* são Velocidade Linear, Aceleração Linear, Velocidade Angular e Aceleração Angular. No caso da movimentação da plataforma virtual desse tópico foram utilizadas apenas as propriedades "Velocidade Linear" e "Aceleração Linear".

4.1.2.1 A Movimentação Através da Velocidade Linear pela Janela de Rotas

A movimentação da plataforma foi implementada pelos seguintes comandos de teclado:

- Seta para direita: aumenta em 0.1 a velocidade da plataforma no eixo X;
- Seta para esquerda: diminui em 0.1 a velocidade da plataforma no eixo X;
- Seta para cima: aumenta em 0.1 a velocidade da plataforma no eixo Y;
- Seta para baixo: diminui em 0.1 a velocidade da plataforma no eixo Y;

Nesse trabalho será descrita apenas a implementação do movimento referente à Seta Direita (movimento linear para a direita ao longo do eixo X), pois os outros movimentos são implementados de maneira semelhante.

Para a implementação desse movimento foram usados os seguintes nós: 1 nó *Motion;* 1 *nó KeyboardSensor;* 1 nó *Constant;* 1 nó *Addition* e 2 nós *Converter*.

Todos os *nós* foram inicialmente colocados na árvore de simulação, embaixo do *nó* Plataforma. Em seguida, esses mesmos *nós* foram arrastados da árvore de simulação para a janela *Routes*.

As propriedades dos *nós* foram configuradas da seguinte maneira:

- *KeyboardSensor*: na sua propriedade *VirtualKeyName* foi colocado o valor "VK_RIGHT". Essa é a única propriedade desse *nó*;
- *Constant*: na sua propriedade *Value* foi associado o valor 0.1. Essa é a única propriedade desse *nó*;
- *Converter*: as propriedades *AutoTrigger* dos dois *nós Converter* usados foram setadas.

Os *nós Motion* e *Addition* não tiveram suas propriedades padrões modificadas.

Como visto na Análise de Ferramentas, cada nó arrastado à janela de rotas possui um ícone de entrada (bola azul) e um de saída (triângulo verde) de dados.

As primeiras rotas (ligação) criadas foram as rotas do *nó KeyboardSensor:*

KeyboardSensor_OnKeyDown → Constant_Trigger
KeyboardSensor_OnKeyDown → Addition_Trigger

Essas ligações são referentes aos *Triggers* (disparos) dos nós *Constant* e *Addition*. A opção *On_KeyDown* do *nó KeyboardSensor* é disparada sempre que sua tecla é pressionada. Portanto, os nós *Addition* e *Constant* só serão ativados quando a Seta Direita for pressionada.

A próxima ligação criada foi:

Constant_Value → Addition_InValue1

Quando o nó *Constant* for ativado (pelo pressionamento da Seta Direita), seu valor será colocado na variável *InValue1* do nó *Addition*.

O nó *Addition* possui duas variáveis de entrada: *InValue1* e *InValue2*. Quando ativado, esse nó soma os valores das duas variáveis e coloca o resultado numa variável de saída: *OutValue*. Como visto anteriormente, esse nó também é ativado pelo pressionamento da Seta Direita. Veremos adiante que o valor da variável *InValue2* será referente ao valor da velocidade atual do nó *Motion*.

Sua ligação de saída coloca o seu resultado no primeiro *Converter*:

Addition_OutValue → Converter_XIn

Por sua vez, o nó *Converter* transforma o valor *Float* que está na sua entrada para um vetor3D. O valor X do vetor conterá o valor *Float* da entrada e os outros valores (Y e Z) permanecerão com o valor zero. Esse passo tem que ser executado, pois a velocidade do nó *Motion* exige como entrada um Vetor3D. Como a propriedade *AutoTrigger* do *Converter* está ligada, ele é disparado automaticamente quando recebe um valor na sua entrada. Sua ligação de saída é:

Converter_3DVectorOut → Motion_Velocity

Como apenas o valor X do vetor3D é diferente de zero, o *nó Motion* aplicará uma velocidade apenas no eixo X de seu *Frame* pai.

O nó *Motion* possui a seguinte ligação de saída:

Motion_Velocity → Converter1_3DVectorIn

Essa ligação mostra que o nó *Motion* envia sua velocidade atual para o nó *Converter1*. Esse passo é necessário porque a velocidade de saída do nó *Motion* é enviada em forma de um Vetor3D, mas esta será usada como entrada do nó *Addition* que requer valores *float*.

A ligação de saída do nó *Converter1* envia um valor *float* à variável *InValue2* do nó *Addition*.

Converter1_XOut → Addition_InValue2

Portanto, com essas ligações, a velocidade do *Frame* Plataforma será incrementada em 0.1 a cada pressionamento da Seta Direita. Isso se deve ao fato de o nó *Addition* sempre somar 0.1 à velocidade de *Motion* a cada pressionamento dessa Seta.

A figura 18 abaixo mostra o estado final da janela *Routes* para o movimento da Seta Direita:

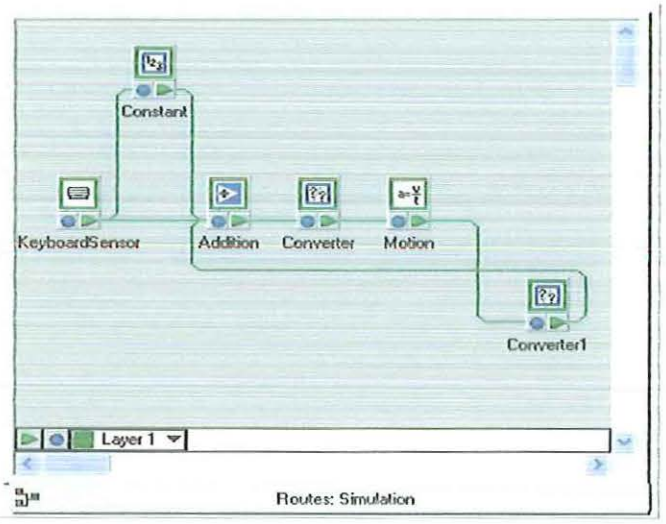

Figura 18 - Janela *Routes* do movimento da Seta Direita.

Esse exemplo mostra claramente a criação de uma interação sem a necessidade de criação de *scripts*. Essa é uma grande vantagem da ferramenta EON Studio. Para

criar essa interação em outras ferramentas de desenvolvimento de ambientes virtuais seria necessário declarar variáveis, escrever procedimentos etc.

A criação de interações pela janela de rotas é, muitas vezes, intuitiva e mais fácil de ser compreendida. Não há necessidade de conhecer uma linguagem de programação.

Ao ler esse exemplo pela primeira vez, sem ter contato com o *software* EON Studio, pode parecer complicada a criação de interações pela janela de rotas. Mas, após uma pequena familiarização com os nós e métodos do *software*, pode-se perceber que essa janela é de grande utilidade.

De maneira análoga, é possível criar os movimentos das Setas Esquerda, Cima e Baixo. Para a Seta Esquerda basta trocar o nó *Addition* por um nó *Subtraction*. Para as Setas Cima e Baixo deve-se trocar os eixos X por eixos Y (*Converters*).

4.1.2.2 A Movimentação Através de *Scripts*

Para verificar as diferenças entre a movimentação da plataforma criada pela janela *Routes* e a movimentação criada por *scripts*, esse tópico recriou o movimento da Seta Direita, mas agora pela utilização de *scripts*.

O primeiro passo foi adicionar um nó *Script* à árvore de simulação, embaixo do nó Plataforma (a posição desse nó não influencia em sua execução, mas deve ser administrada de forma correta, por questões de organização).

Nas propriedades do *Script*, pode-se escolher entre a linguagem *JScript* (padrão) e *VBScript*. No caso do exemplo foi usada a linguagem *VBScript*. Também, nas propriedades do nó, é possível criar variáveis, de entrada ou saída, dos tipos: *integer*, *boolean*, *float* etc.

No caso desse exemplo, a janela de rotas teve de ser usada, mas de maneira simplificada. Os nós usados foram os nós *Motion*, *KeyboardSensor* e *Script*.

O segundo passo foi arrastar os *nós KeyboardSensor* e *Script* à janela *Routes*. Feito isso, foi criada uma variável de entrada (eventIn), com o nome "SetaDir", do tipo *SFBool*, com o valor inicial "zero".

Essa variável terá a função de avisar o *Script* quando a Seta Direita for pressionada. Como é uma variável do tipo *Boolean*, ela pode assumir somente os valores *True* ou *False* (Verdadeiro ou Falso). O valor *False* é seu valor inicial. Abaixo é mostrada a ligação que ativa a variável SetaDir ao pressionamento da Seta Direita:

KeyboardSensor_OnKeyDown → Script_SetaDir

Para cada variável de entrada criada nas propriedades do *Script*, deve ser escrito um procedimento, de mesmo nome, na tela de edição de texto do *Script*. Esse procedimento será executado toda vez que essa variável mudar de valor.

No exemplo da variável SetaDir, o seu procedimento teve a seguinte forma:

```
Sub On_SetaDir
        'código executado quando a variável for ativada
end sub
```

Para abrir a tela de edição de texto do *Script* é necessário abrir o menu *Window*, clicar em *Script* e selecionar a opção *Open Editor*.

O código final do movimento para a direita da plataforma é mostrado abaixo:

```
set MotionNode=EON.FindNode("Motion")
set Vel=MotionNode. GetFieldByName ("Velocity")
Sub On_SetaDir
        v=Vel.Value
        v(0)=v(0)+0.1
        Vel.Value=v
end sub
```

As duas primeiras linhas se referem a declarações. A primeira linha associa o *nó Motion* a uma variável de nome "*MotionNode*". A segunda linha associa uma das propriedades do *nó Motion* a uma variável de nome "Vel". Dentro do procedimento "On_SetaDir", a primeira linha associa o valor da variável "Vel" à variável "v". O valor de "Vel" é um vetor3D, por isso essa associação é necessária. Na segunda linha é somado o valor "0.1" ao valor referente ao eixo X de "v". Nos *Scripts*, os eixos são referenciados pelos números 0 (X), 1 (Y) e 2 (Z). Na terceira linha, o valor incrementado é repassado à variável "Vel". Dessa forma, toda vez que a Seta Direita for pressionada, a velocidade do eixo X de Motion será incrementada em "0.1".

Nesse exemplo simples, o Script criado conteve poucas linhas e foi de fácil entendimento. Em aplicações mais complexas, os Scripts podem se tornar um problema, se esses não forem administrados de maneira correta.

Como dito anteriormente, as janelas *Routes* podem, também, assumir tamanhos inviáveis, em aplicações complexas. Para amenizar esse tipo de problema, existe um componente na janela *Routes* chamado: *Layer*.

4.1.2.3 Layers

São usados para organizar as rotas da janela Routes e torná-las mais fáceis de discernir umas das outras. Um *Layer* é um grupo de rotas e *nós* da janela *Routes*, que pode ser mostrado separadamente.

Abaixo, na figura 19, é mostrado o exemplo de um *Layer* para as rotas de movimentação da plataforma (Seta Direita). Esse *Layer* foi criado apenas por questão de ilustração, pois o exemplo ainda era muito simples e não necessitava de um *Layer*.

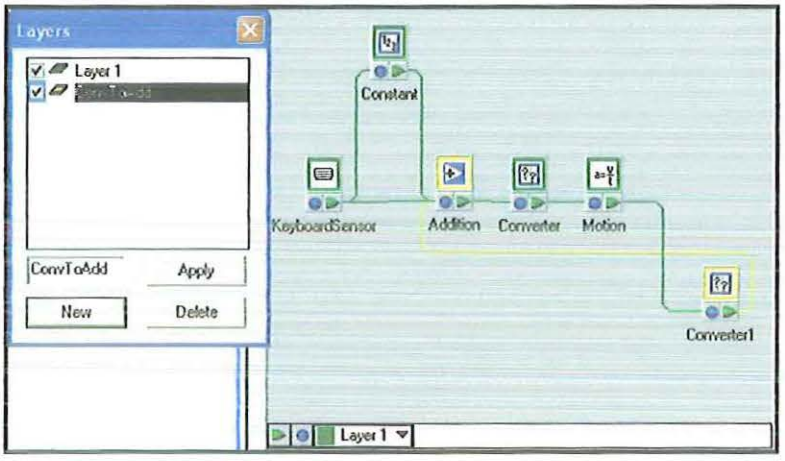

Figura 19 - Aplicação de um *Layer*.

Para exibir a janela para edição de *Layers*, basta clicar com o botão direito do *mouse* sobre a janela *Routes* e selecionar a opção *Layer Editor*. Para criar um novo *Layer*, é necessário apenas digitar um novo nome na caixa de texto e clicar no botão *New*. Para mudar a cor de um *Layer*, basta clicar no ícone localizado ao lado do nome referente ao *Layer* desejado. Para associar componentes da janela *Routes* com o novo *Layer* criado, deve-se clicar com o botão direito do *mouse* sobre o componente desejado e selecionar a opção *MoveToLayer*. Dentro dessa opção estarão as opções de *Layers* disponíveis para a associação.

4.1.2.4 Rotacionando as Rodas (usando a velocidade angular)

Até esse ponto do projeto, o movimento da plataforma era simples e pouco natural. Para tornar o movimento um pouco mais real, foi criada a movimentação angular das rodas, para que o usuário do ambiente tivesse uma sensação mais real do movimento.

Como ainda não havia sido implementada a detecção de colisão (das rodas com o solo), o movimento angular das rodas não influenciou o movimento linear do carro.

Foi escolhida a utilização de *scripts* ao invés de rotas, pois mais adiante nesse trabalho o esse tipo de implementação se tornará comum e é preciso estar bem familiarizado com novas funções.

Nesse tópico será descrita apenas a movimentação de uma das rodas, pois o movimento das outras rodas é análogo.

O único *nó* a ser adicionado à árvore de simulação é o *nó Rotate*. Ele deve ser colocado embaixo do *Frame* de uma das rodas, no caso a "Roda1". Suas propriedades são: Eixos de Rotação, *Lap Time, Active*.

A propriedade *Active* deve começar desligada, para que a rotação da roda comece apenas quando o veículo se mover. O *Lap Time* deve ser iniciado com o valor 30, no caso desse exemplo. Apenas o eixo X de rotação deve ser iniciado com o valor "1", os outros dois eixos devem conter o valor zero, para que a roda tenha movimento apenas no eixo X.

O *script* criado é bem simples, e é mostrado na figura 20:

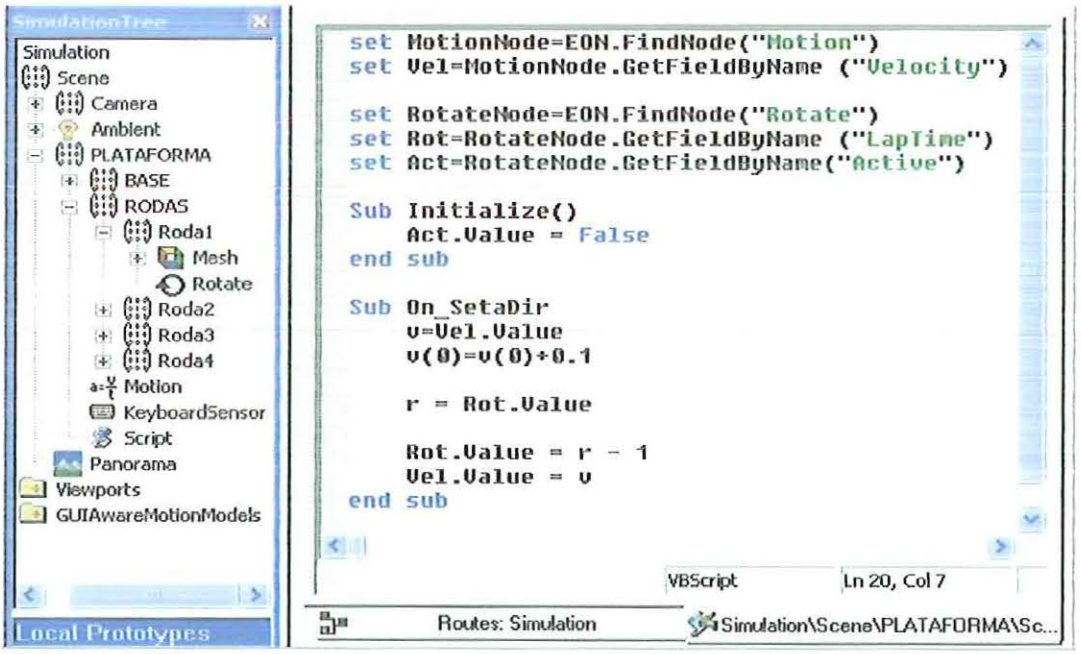

Figura 20 - Script de movimento de uma das Rodas.

Na parte superior do *script* foram adicionadas três declarações: *RotateNode, Rot e Act*, referentes, respectivamente ao *nó Rotate* e as propriedades *LapTime* e *Active* do mesmo.

Foi adicionado ao *script*, também, um procedimento chamado *Initialize*. Esse procedimento é executado apenas uma vez, logo no início da aplicação. Nele, o valor da variável *Act* é atualizado para *False*, para que haja possibilidade de a roda estar se movendo logo na entrada do ambiente.

As linhas adicionadas ao procedimento SetaDir se referem ao decremento de uma unidade do valor de *LapTime* a cada pressionamento da Seta Direita. O motivo disso é diminuir o tempo de rotação da roda, conforme a velocidade do veículo aumenta.

Os valores usados no exemplo (*LapTime*= 30 e decremento de 1) foram obtidos por tentativas. Esses foram os valores que mais se adequaram ao valor "0.1" de incremento da velocidade.

A função desse exemplo é completamente distinta do objetivo final do trabalho. O exemplo foi criado para familiarização com os novos *nós* e com o *script* do EON Studio. Mais adiante nesse trabalho serão estudados os *nós* especiais para dinâmica de corpos rígidos, gravidade, detecção de colisão etc.

4.1.3 ADICIONANDO GRAVIDADE E COLISÃO

Navegando pelo ambiente com o *mouse* pôde-se perceber que o *nó Panorama* adicionado nos tópicos anteriores funciona apenas como cenário. Isso porque por mais que se navegue pelo ambiente, em direção ao solo, por exemplo, a câmera nunca chegará a tocá-lo. Portanto o solo adicionado pelo *Panorama* não funciona como um objeto do ambiente.

Para testar o fato afirmado acima, foi adicionada uma força de gravidade à plataforma, para analisar se esta chegaria a tocar o solo.

O EON Studio possui um *nó* especial para adição de gravidade. Este *nó* deve ser colocado na árvore de simulação embaixo do *frame* que sofrerá a ação da gravidade. No caso do projeto, o *nó Gravitation* foi arrastado para baixo do *nó* Plataforma.

As propriedades do *nó Gravitation* são: *Active e Força Gravitacional*.

A propriedade Força Gravitacional tem seu valor padrão igual a -9.81 (gravidade da Terra).

Depois de rodada a aplicação e implementadas algumas mudanças na posição do *nó* Plataforma, foi verificado que o *nó Gravitation* aplica uma força ao objeto até que este alcance a posição "Z=0". Alcançada essa posição, o objeto para de se mover. Dessa forma, foi impossível testar a detecção de colisão da plataforma com o solo. Para resolver o problema, foi adicionada mais uma geometria a cena: um cubo. Esse cubo foi alongado nos seus eixos X e Y para poder representar um solo.

A adição do cubo a cena foi feita via *nó Mesh*. Portanto, foi arrastado um *nó Frame* para baixo do *nó* Scene, um *nó Mesh* para baixo do *nó Frame* e um *nó Texture* para baixo de *Mesh*. A propriedade *scale* do *Mesh* foi alterada em X, Y e Z. Também, foi escolhida uma textura de "terra" para o chão. O novo *Frame* foi renomeado para "Solo" e a posição da plataforma no eixo Z foi modificada para "3".

O resultado é mostrado na figura 21 abaixo:

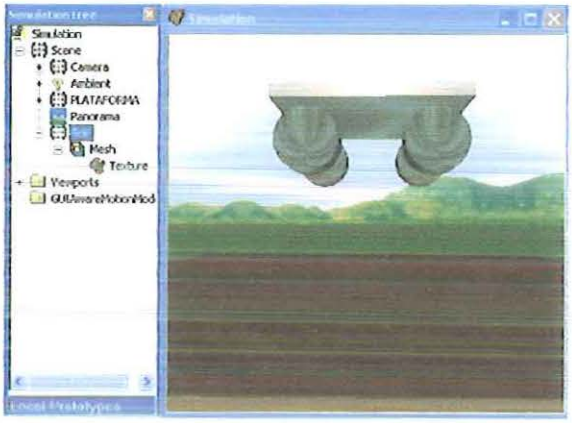

Figura 21 - Solo criado por *Mesh* e Textura.

Porém, isso tudo não é o suficiente para que a plataforma possa colidir com o solo. Se ativarmos a gravidade e rodarmos a simulação, a plataforma irá cair até a sua posição "Z=0" e atravessará o chão, como mostrado na figura 22:

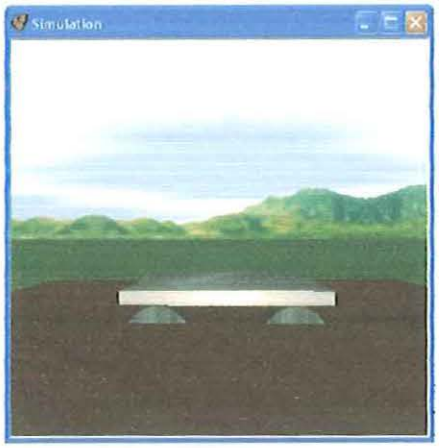

Figura 22 - Plataforma atravessa o chão e pára em Z=0.

Para fazer a plataforma colidir com o chão foi necessário o estudo de mais alguns *nós*: *Collision*, *ConvexCollision* e *ConvexCollisionManager*.

Como visto na Revisão Bibliográfica, existem duas possibilidades de implementar detecção de colisão em EON Studio: pelo *nó Collision* e pelo *nó ConvexCollision*. Neste trabalho foram testados os dois tipos de colisão.

4.1.3.1 Detecção de Colisão via Nó *Collision*

Primeiramente o *nó Collision* foi adicionado embaixo do *nó* Plataforma. Suas propriedades são:

- *Move Back Collided Objects*: faz objetos não atravessarem um ao outro na cena quando colididos (quando desmarcada, essa opção apenas registra que houve uma colisão, mas um objeto atravessa o outro);
- *Collision Detection With Children Frames*: faz todos os filhos de um *nó* participarem da colisão;
- *Active*: liga ou desliga a colisão.

Todas as propriedades do *nó Collsion* foram marcadas como ativas.

Esse *nó* possui como filhos duas pastas, que devem ser usadas de maneira correta para o funcionamento da colisão. São elas as pastas: *CollisionObjects* e *IgnoredObjects*.

Na pasta CollisionObjects devem ser colocadas referências a objetos que participarão da colisão com o *Frame*, onde está localizado o *nó Collision*. No caso desse exemplo, foi adicionada apenas a referência ao *nó* Solo. Para adicionar uma referência de um *nó* à pasta *CollisionObjects*, basta clicar com o botão direito do *mouse* em cima

do *nó* que deseja copiar e selecionar a opção *"Copy As a Link"*. Feito isso, clica-se com o botão direito do *mouse* em cima da pasta *CollisionObjects* e seleciona-se a opção *"Paste"*.

Na pasta *IgnoredObjects* teriam de ser colocados objetos que não participariam da colisão, mas nesse caso a pasta foi deixada vazia.

Como resultado, a plataforma não atravessa mais o objeto solo.

4.1.3.2 Detecção de Colisão via Nó *ConvexCollision*

Primeiramente foi adicionado um *nó ConvexCollisionManager* embaixo do *nó Scene* da árvore de simulação. Esse *nó* pode ser adicionado em qualquer lugar da árvore que seu funcionamento será o mesmo. O segundo passo foi adicionar um *nó ConvexCollision* embaixo do *Frame* Plataforma e um outro embaixo do *Frame* Solo.

As propriedades do *nó ConvexCollisionManager* são: *CollisionTolerance (tolerância) e Eneabled (habilitado).*

As propriedades do *nó ConvexCollision* são: *MoveBack Option, Static e Eneabled.*

O *nó ConvexCollision* localizado embaixo do *Frame* Plataforma foi renomeado para "ConvexCollisionPlataforma". O *nó ConvexCollision* adicionado embaixo do *Frame* Solo foi renomeado para *ConvexCollisionSolo*.

A propriedade *"MoveBack Option"* do *nó ConvexCollisionPlataforma* foi modificada para *"Slide"*. Dessa maneira, a plataforma pôde ter movimentos quando colidida sobre o chão. A propriedade *Static* do *nó ConvexCollisionSolo* foi marcada como ativa, para que o solo não possuísse movimentos durante a colisão. Os *nós ConvexCollision* possuem como filho uma pasta chamada *ConvexGeometry* que será explicada nos próximos tópicos desse trabalho.

O *nó ConvexCollisionManager* teve sua propriedade *CollisionTolerance* alterada para "1". Quando essa propriedade possui um valor muito baixo (ou até negativo), um objeto pode atravessar outro. Porém, valores muito baixo de *CollisionTolerance* podem não funcionar corretamente para objetos muito complexos. Quanto maior esse valor, mais distantes ficarão os objetos colididos, comprometendo a fidelidade da aplicação com a realidade.

O *nó ConvexCollisionManager* possui uma pasta como filho: *CollisionNó*. Nessa pasta devem ser colocadas referências a *nós ConvexCollision* que participarão da

colisão. No caso desse exemplo, foram colocadas referências aos *nós* *ConvexCollisionPlataforma* e *ConvexCollisionSolo*.

O resultado final da colisão por *ConvexCollision* foi, a princípio, semelhante ao resultado anterior. Porém no próximo tópico desse capítulo serão discutidas as dificuldades encontradas nos dois métodos de implementação de colisão.

4.1.3.3 Dificuldades Encontradas na Implementação de Gravidade e Colisão

Na implementação de detecção de colisão pelo *nó Collision* (primeiro tipo), não foi possível movimentar os objetos quando esses estão marcados como "colididos" (geometria do veículo encostada na geometria referente ao solo). Portanto esse tipo de implementação se tornou inútil para o projeto.

No segundo tipo de implementação (por *ConvexCollision*) pôde-se mover a plataforma normalmente, mesmo quando esta estava marcada como "colidida".

Esse tipo de implementação funcionou de maneira correta para o exemplo acima. Porém, em determinados testes realizados para o aprendizado dessa técnica de detecção de colisão, foram encontrados alguns problemas. Mesmo no exemplo citado, quando a propriedade *CollisionTolerance* foi alterada para "0.1", o movimento da plataforma começou a travar ao longo do percurso.

Em alguns testes realizados, os objetos colidiam normalmente por um determinado tempo, mas logo em seguida, os objetos ignoravam a colisão e atravessavam um ao outro.

Esses tipos de problemas juntamente com a necessidade do projeto de utilizar dinâmicas de corpos rígidos, forçaram uma mudança no foco de estudo do trabalho.

A detecção de colisão foi deixada de lado por um momento, para o estudo das dinâmicas de corpos rígidos (ou propriedades físicas dos objetos). Nesse novo estudo foi verificado que o sistema de utilização de objetos rígidos difere totalmente do sistema usado até o momento. Esse sistema possui seus próprios *nós* que não interagem com vários dos *nós* usados até agora. A uma mudança considerável, também, no sistema de detecção de colisão.

4.1.4 IMPLEMENTAÇÃO DAS FORÇAS FÍSICAS

Nesse tópico será descrito o uso da dinâmica de corpos rígidos (introduzido no capítulo anterior) em aplicações de EON Studio, bem como seus benefícios e

dificuldades. Será descrita, também, a montagem da Plataforma Virtual Agrícola Móvel e a criação e importação das geometrias finais da plataforma e do cenário.

4.1.4.1 Gravidade e colisão dos Corpos Rígidos

Depois de construída a plataforma simples do tópico anterior, nessa fase foi implementada a sua transformação em um corpo rígido. Para isso foram adicionados à árvore de simulação um *nó RigidBodyManager*, um *nó RigidBody* embaixo do *Frame* Plataforma. Esse *RigidBody* foi renomeado para *RigidBodyPlataforma.*

Algumas modificações foram feitas na árvore de simulação, antes da implementação da detecção de colisão de um corpo rígido:

- Uma referência do *nó RigidBodyPlataforma* foi colocada embaixo da pasta *RigidBodies* do *nó RigidBodyManager*;
- Os antigos *nós Gravitation, Motion e Rotate* foram excluídos da árvore, pois esses não têm funções em sistemas de corpos rígidos;
- Os códigos dos procedimentos do *script* foram excluídos (ou comentados). É possível comentar uma linha de código, colocando-se uma apóstrofe como seu primeiro caractere);

Com essas modificações, a simulação já começava a funcionar como dinâmica de corpos rígidos. As propriedades físicas começaram a ser calculadas e as forças começaram a ser aplicadas.

Após rodar a simulação pela primeira vez (depois de adicionados os *nós* referentes à dinâmica de corpos rígidos), as barras de propriedades dos *nós RigidBody* começam a informar as características físicas calculadas para os objetos. No caso do exemplo, o *RigidBody* referente a plataforma possuía as seguintes características (valores definidos no sistema de unidades SI: Metros, Kilogramas, Newton e Segundos):

:

- Densidade = 1;
- Volume = 26.93;
- Massa = 26.93;
- Coeficiente de Restituição = 0.5;
- Centro de Massa = X(-709895e-009), Y(-6.26627e-016) e Z(0.695122).

As propriedades Massa, Volume e Centro de Massa são calculadas automaticamente e não podem ser definidas pelo programador. Como o valor da massa da plataforma era "26.9" e o valor da gravidade era "-9.82", seu peso valia aproximadamente 264 quilos. O valor do peso não é mostrado na barra de propriedades.

A única força que existia até o momento era a força da gravidade (-9.82), implementada pelo *nó RigidBodyManager*. Portanto, ao rodar a aplicação, essa força era imediatamente aplicada ao *nó RigidBodyPlataforma* que caía indefinidamente.

Para resolver esse problema e, finalmente, implementar a detecção de colisão de um corpo rígido, foram feitas m ais algumas alterações na árvore de simulação:

- O *nó ConvexCollisionPlataforma* foi reposicionado. Este passou a ser filho de seu respectivo *nó RigidBody;*
- Foi adicionada uma referência do *nó ConvexCollisionManager* à pasta *CollisionManager* do *nó RigidBodyManager*;
- A propriedade *Static* do *nó ConvexCollisionSolo* foi ligada;
- A propriedade *MoveBackOption* do *nó ConvexCollisionPlataforma* foi alterada para "none";

Nesse momento da aplicação, a plataforma cai de sua posição "Z=3" afetada pela gravidade do *nó RigidBodyManager*. O objeto Solo não é afetado pela gravidade, pois este não é um corpo rígido. Mesmo não sendo um corpo rígido, este agora colide com a plataforma pois ambos os objetos estão referenciados na pasta *CollisionNós* de *ConvexCollisionManager*. Na colisão, o objeto Solo não tem sua posição alterada devido a sua propriedade "*Static=True*".

Diferente da colisão dos tópicos anteriores, a colisão dos corpos rígidos é semelhante à colisão no mundo real, pois estas possuem agora um coeficiente de elasticidade e seus objetos possuem peso, figura 23. Esse fator contribui muito para o nível de realismo da aplicação.

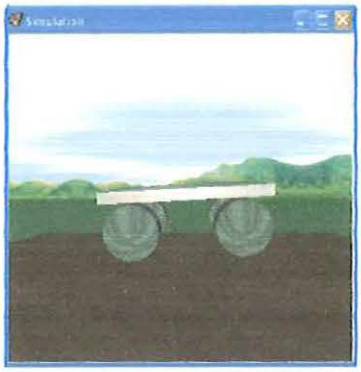

Figura 23 - Plataforma colidindo com elasticidade.

4.1.4.2 Movimentando um corpo rígido

Depois de criada a detecção de colisão da plataforma com o solo, foi implementada uma movimentação simples da mesma. Essa movimentação simples foi criada por comandos de teclado (as setas direcionais) que tinham como função aplicar uma força básica simples à plataforma.

Os *nós* usados para aplicação de forças à um corpo rígido são: *Force* e *Torque*. Esses *nós* funcionam de maneira semelhante, porém, *Force* aplica uma força básica sobre um corpo rígido e *Torque* aplica uma força de torque básica sobre um corpo rígido. Para serem utilizados, esses *nós* devem ser colocados embaixo da pasta *Forces* de um *nó RigidBody*.

A princípio, foi utilizado somente o *nó Force* para a movimentação da plataforma. Portanto este foi adicionado à árvore de simulação embaixo da pasta *Forces* do *nó RigidBodyPlataforma*. Suas propriedades são:

- Vetor de Força: direção e magnitude do vetor de força;
- Ponto de aplicação: o ponto do corpo rígido onde a força será aplicada;
- *Body Frame*: Determina se o vetor de força deve atuar somente no *Frame* do corpo rígido ou em todo o ambiente;
- Uniforme: determina se a força deve ser aplicada de maneira uniforme sobre o corpo rígido (em seu centro de massa);
- *Eneabled*: liga ou desliga a força.

Para testar a aplicação da força, a propriedade "Vetor de Força" do *nó* foi alterada para "X=1, Y=0, Z=0". Quando rodada a aplicação, a plataforma iniciou um movimento para a direita ao longo do eixo X.

Voltando X ao seu valor inicial (zero), pôde-se começar a implementar a movimentação da plataforma por comandos de teclado. Como nas seções anteriores, será descrito aqui apenas o movimento referente à seta direcional direita, pois os outros três movimentos são análogos.

Foi criada nas seções anteriores a rota:

KeyboardSensor_OnKeyDown → Script_SetaDir

Essa rota permaneceu a mesma, para que o procedimento *On_SetaDir* fosse chamado a cada pressionamento da tecla Seta Direita.

O código do *script*, porém, foi totalmente alterado, figura 24:

```
set ForceNode=EON.FindNode("Force")
set Force=ForceNode.GetFieldByName ("ForceVector")

Sub Initialize()
    F=Force
    F(0)=0
    F(1)=0
    F(2)=0
    Force.Value = F
end sub

Sub On_SetaDir
    F=Force
    F(0)=F(0)+1
    Force.Value = F
end sub
```

Figura 24 - *Script* do movimento Seta Direita.

Na primeira linha uma variável foi associada ao *nó Force*. A segunda linha associou a propriedade *ForceVector* do *nó Force* à variável "*Force*". Dentro do procedimento de inicio da aplicação (*Initialize*) foi criado um código responsável por zerar os valores da variável *ForceVector*.

No procedimento *On_SetaDir* (acionado a cada pressionamento da Seta Direita) o valor X da variável *ForceVector* é incrementado em "1". Dessa forma, a plataforma aumenta sua velocidade (empurrada pela força do *nó Force* associada a ela) a cada pressionamento da Seta Direita.

Nessa fase do projeto, dois problemas foram encontrados:

- Zerar a velocidade: Como ainda não havia sido criado um método para zerar velocidade da plataforma, seu movimento era infinito (depois de movimentada, a plataforma não voltava mais ao seu estado inicial, ou seja, não parava de se mover em nenhum momento da aplicação). Mesmo quando implementadas as

funções de todas as setas direcionais, era impossível controlar a plataforma, pois seus movimentos se tornavam muito instáveis;

- Resistência: Uma outra causa da instabilidade dos movimentos da plataforma foram os valores muito baixos das suas propriedades: *LinearDamping* e *AngularDamping* (responsáveis pela resistência da plataforma à forças).

Para resolver esses problemas, inicialmente a velocidade da plataforma foi zerada "manualmente" (alterada pelo programador diretamente em suas propriedades) durante a execução da aplicação. Mesmo depois de zerada sua velocidade, a plataforma continuava a se mover, pois os seus valores de resistência estavam muito baixos. Por esse motivo, os valores *LinearDamping* e *Angular Damping*, referentes à resistência a movimentos da plataforma, foram alterados respectivamente para 20 e 10, ver figura 25:

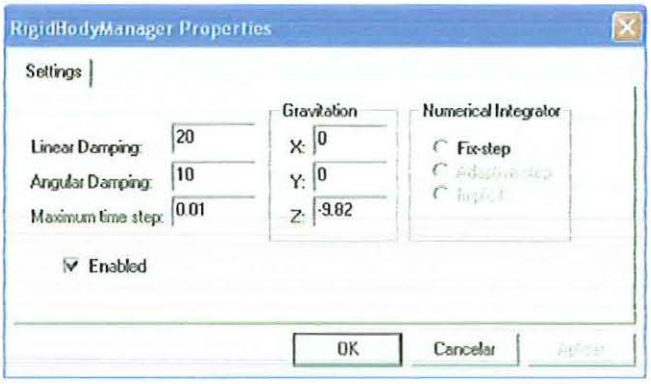

Figura 25 - Valores de resistência alterados.

Desse modo, quando zerada a velocidade da plataforma (manualmente), seu movimento perdia força até parar.

Foram testados vários valores para as propriedades *LinearDamping* e *AngularDamping*. Os valores que tornavam o movimento da plataforma mais fiel à realidade foram "20 e 10" respectivamente.

Depois de encontrados os valores ideais para as variáveis de resistência, foi criado um método para que a velocidade da plataforma fosse zerada quando as Setas Direcionais fossem soltadas.

Até o momento, a velocidade da plataforma era alterada apenas pelo pressionamento das Setas Direcionais. Quando estas teclas eram mantidas pressionadas, a velocidade não era alterada. Para que a velocidade fosse alterada constantemente pela manutenção do pressionamento das Setas Direcionais, uma única alteração foi feita na

janela *Routes*. A variável de saída do *KeyboardSensor* foi alterada de *OnKeyDown* para *Down:*

KeyboardSensor_Down → Script_SetaDir

Essa alteração foi implementada para aumentar o controle sobre a plataforma. Feito isso, pôde-se concluir o método para zerar a velocidade da plataforma.

Esse método foi baseado num sensor de tempo (*TimeSensor*) colocado abaixo do *Frame* Plataforma. Esse *nó* gera pulsos em intervalos regulares. As suas propriedades são: Tempo do ciclo, Tempo do início, Tempo do término, Enviar pulsos de início e término, *Loop, A*tivo no início da simulação.

Todos os valores das suas propriedades devem ser especificados em segundos.

Esse sensor começa a gerar pulsos no momento definido em sua propriedade "Tempo do início". Depois desse momento, pulsos são gerados em intervalos regulares especificados em "Tempo de ciclo". Se a propriedade *Loop* for desativada, os pulsos pararão de ser gerados no momento definido em "Tempo do término". Os tempos de Início e Término são medidos a partir do momento que o *nó* é ativado (propriedade *SetRun*).

Dois métodos foram utilizados para zerar a velocidade da plataforma: Rotas e *Scripts*. Na implementação por Rotas foram adicionados mais dois *nós* à árvore de simulação: *Constant* e *Converter*. Esses foram arrastados, logo em seguida, para a janela *Routes*, juntamente com o *nó Force*.

As ligações criadas foram as seguintes:

KeyboardSensor_OnKeyUp → TimeSensor_SetRun
TimeSensor_OnStopPulse → Constant_Trigger
Constant_Value → Converter_XIn
Converter_3DVectorOut → Force_ForceVector

A propriedade *AutoTrigger* do *nó Converter* foi ativada. As configurações do *nó TimeSensor* são mostradas na figura 26 abaixo:

Figura 26 - Propriedades do *TimeSensor*.

Quando a Seta Direcional é liberada (*OnKeyUp*) a primeira rota faz com que o *nó TimeSensor* seja ativado. Esse *nó* aguarda um intervalo definido (no caso 2 segundos) e dispara o *nó Constant* pela segunda rota. A terceira rota coloca o valor de *Constant* na variável de entrada "XIn" do *nó Converter*. Como o valor armazenado em *Constant* é zero, a variável XIn assume o valor zero. Então, o *nó Converter* cria um vetor 3D com o valor zero na posição "X". Como os valores de "Y" e "Z" não foram especificados nas variáveis de entrada de *Converter*, estes também assumem o valor zero. Portanto, a quarta rota associa um vetor de zeros à velocidade de *Force*. Desse modo, a velocidade da plataforma é zerada, ver figura 27.

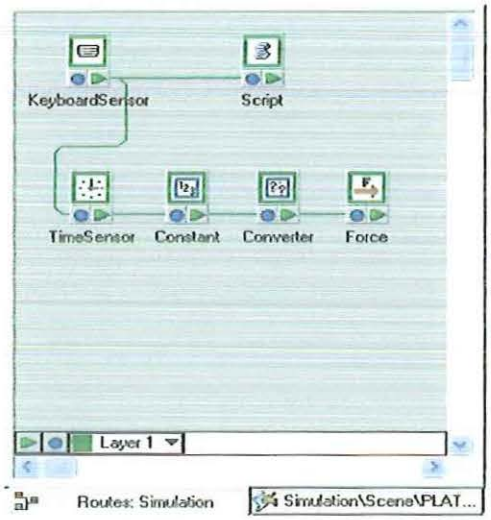

Figura 27 - Velocidade zerada por rotas.

A implementação por *Scripts* não utiliza os *nós Constant e Converter*. Sua janela de rotas é mais simples. Houve a necessidade de criar uma variável de entrada nas propriedades do *nó Script*. Essa variável foi chamada de "Zerar_Vel". O tipo de dados selecionado foi *Boolean* de valor inicial zero.

As rotas desse tipo de implementação são mostradas abaixo, na figura 28:

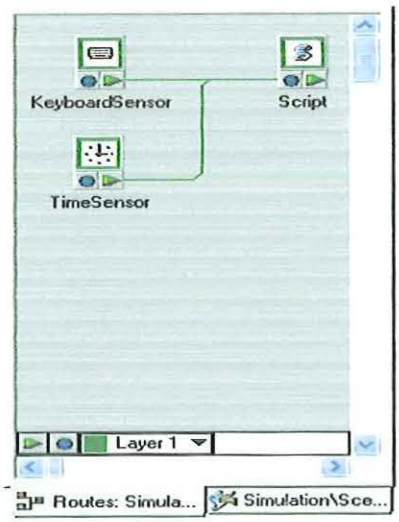

Figura 28 - Rotas da implementação por *Script*.

A única rota criada foi:

TimeSensor_OnStopPulse → Script_Zerar_Vel

Como explicado anteriormente, cada variável de entrada criada nas propriedades do *nó Script* são procedimentos desse *nó*. Portanto, essa rota executa o código criado dentro do procedimento "Zerar_Vel" quando o *nó TimeSensor* emitir um pulso de término.

O código dessa implementação é mostrado abaixo:

```
set ForceNó=EON.FindNó("Force")
set Force=ForceNó.GetFieldByName ("ForceVector")
set TimerSensorNó=EON.FindNó("TimeSensor")
set TimeSensor=TimerSensorNó.GetFieldByName ("SetRun")
Sub Initialize()
        F=Force
        F(0)=0
        F(1)=0
        F(2)=0
        Force.Value = F
```

```
        TimeSensor.Value = FALSE
end sub
Sub On_SetaDir
        F=Force
        F(0)=F(0)+1
        Force.Value = F
        TimeSensor.Value = TRUE
end sub
Sub On_Zerar_Vel
        F=Force
        F(0)=0
        Force.Value = F
end sub
```

Na parte superior do *Script* estão declarados os *nós Force e TimeSensor* e suas propriedades *ForceVector* e *SetRun*. No procedimento de inicialização (*Initialize*) essas duas propriedades estão sendo zeradas. A linha "*TimeSensor.Value = TRUE*" foi adicionada ao procedimento "On_SetaDir". Essa linha ativa o *nó TimeSensor* que por sua vez executa o código de "On_Zerar_Vel" dois segundos depois. O código desse procedimento zera o valor de *ForceVector* parando a plataforma.

4.1.4.3 Terrenos irregulares

Como nesse momento a plataforma possuía movimentos mais próximos aos reais e, também, controle realizado por teclado, foi testado seu comportamento em um tipo de terreno irregular simples.

O terreno foi criado apenas por cubos adicionados à árvore de simulação via *nós Mesh*. Para cada *Mesh* adicionado foi criado um *Frame* de mesmo nome responsável por posicionar os objetos na cena. As propriedades *Roll* dos *Frames* foram usadas para rotacionar os objetos (para criação de desníveis). O terreno final está mostrado na figura 29 abaixo:

Figura 29 - Terreno com desníveis.

A princípio, a plataforma não teve força suficiente para se movimentar nos desníveis, portanto a linha do *Script* referente à adição de força à plataforma teve de ser modificada:

De:	F(0)=F(0)+1
Para :	F(0)=F(0)+20

Aumentando a força aplicada na plataforma em vinte vezes, ela conseguiu completar o percurso de forma natural. A força da gravidade fazia com que a plataforma tivesse mais facilidade de movimento nas descidas do que nas subidas.

4.1.4.4 O movimento completo

Até este momento do trabalho, todos os exemplos de movimento da plataforma foram descritos apenas parcialmente. Esses exemplos mostravam apenas a utilização da Seta Direita do teclado. Portanto a plataforma realizava somente movimentos de aceleração para a direita (sua frente).

A idéia original do projeto era implementar um controle completo da plataforma móvel via teclado. Esse controle incluiria: Aceleração (Seta Direcional para cima); Frenagem e retrocesso (Seta Direcional para baixo); Curva para direita (Seta Direcional para direita) e Curva para esquerda (Seta Direcional para esquerda).

A plataforma possuiria, também, outros controles via teclado, que serão explicados nos próximos tópicos desse trabalho.

Antes de descrever o controle completo da plataforma, será descrito o uso do *nó Torque* para realizar os novos movimentos das Setas Direcionais Direita e Esquerda. Esses movimentos não influenciam a aceleração do veículo.

Para a implementação da força de torque foram adicionados ao exemplo anterior os seguintes *nós*: 2 *KeyboardSensor*; 1 Torque e 1 *TimeSensor*.

Os *nós KeyboardSensor e TimeSensor* foram colocados embaixo do *Frame* Plataforma, enquanto o *nó Torque* foi posicionado embaixo da pasta *Forces* do *nó RigidBody*. Foram criadas, também, mais três variáveis de entrada nas propriedades do *Script*: SetaEsq; SetaCima e Zerar_Tor.

O antigo *nó KeyboardSensor* foi renomeado para *keyboardSensorDir* e os outros dois para *KeyboardSensorEsq e KeyboardSensorCima*. As propriedades dos dois últimos foram alteradas respectivamente para "VK_Left" e "VK_Up". O antigo *nó TimeSensor* foi renomeado para *TimeSensorForce* e o novo *nó TimeSensor* foi renomeado para *TimeSensorTorque*. A propriedade *CycleTime* do *nó TimeSensorTorque* foi alterada para "1".

Os novos *nós KeyboardSensor* e *TimeSensor* foram arrastados para a janela de rotas, onde foram criadas as seguintes ligações:

KeyboardSensorEsq_OnKeyDown → Script_SetaEsq
KeyboardSensorCima_OnKeyDown → Script_SetaCima
TimeSensorTorque_OnStopPulse → Script_Zerar_Tor

Os funcionamentos das novas ligações são análogos aos explicados nos tópicos anteriores.

Várias modificações foram realizadas no *Script*, sendo que todas elas são análogas as explicações dos tópicos anteriores. O detalhe mais importante do *Script* é o eixo de rotação onde é modificado o valor de *Torque*. Enquanto o *nó Force* era alterado em seu eixo "0" (para velocidade), o *nó Torque* teve de ser alterado em seu eixo "2", para que a plataforma executasse o movimento de rotação no eixo correto.

O valor "30", usado na incrementação e decrementação doTorque, foi obtido por testes.

A força de Torque também teve de ser zerada. Essa implementação foi análoga à do *nó Force*.

Nesse momento, ao rodar a aplicação, era possível rotacionar a plataforma em torno de seu próprio eixo, pelo pressionamento das Setas Direcionais Direita e Esquerda. Porém, o movimento de rotação continuava indefinidamente, devido ao valor muito baixo da propriedade Resistência Angular do *nó RigidBodyManager*. O valor dessa propriedade foi trocado para "100". Com essa mudança, o movimento de rotação da plataforma diminuía gradativamente até que fosse zerado.

O maior problema encontrado na implementação dos movimentos da plataforma foi a utilização dos *nós Force e Torque* para uma movimentação real. Até aqui, se movêssemos a plataforma para frente, pela utilização da Seta Direcional para Cima, e, em seguida, tentássemos rotacionar a mesma, o movimento de aceleração não mudava sua direção, apenas a geometria da plataforma era rotacionada. Ou seja, o movimento mantinha sua direção, apesar da plataforma mudar de ângulo.

Para resolver esse problema foi criado um sistema de movimentação da plataforma onde a aceleração, alterada pelas Setas Cima e Baixo, dependia do ângulo de rotação alterado pelas Setas Direita e Esquerda. O código do *Script* nessa parte do projeto se tornou mais extenso e complexo.

Na cena de uma aplicação de EON Studio, um objeto possui três eixos de coordenadas, figura 30:

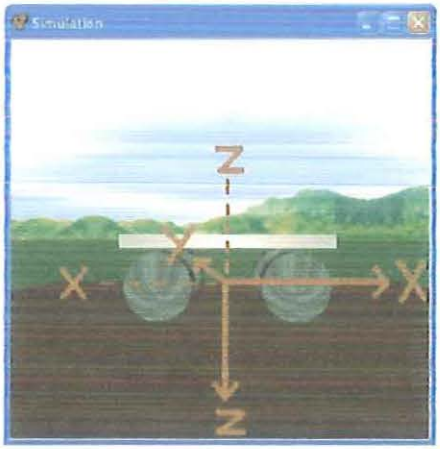

Figura 30 - Um objeto e seus eixos de coordenadas.

No caso da plataforma, a gravidade atua em seu eixo Z e os seus movimentos devem ser executados nos eixos X e Y.

Até o momento, foram descritos somente os movimentos da plataforma ao longo do seu eixo X e seu movimento de rotação em torno do seu eixo Z. Não havia qualquer ligação entre esses dois movimentos.

O "Movimento Completo" descrito neste tópico foi implementado para possibilitar que as alterações da força Torque, que rotacionam a plataforma, alterassem, também, a direção de sua velocidade linear.

O *nó Force* possibilita adição de uma força linear em qualquer um dos eixos X, Y e Z. É possível, também, fazer composições de forças entre os eixos para que o objeto se mova em mais de um eixo ao mesmo tempo. A figura acima mostra claramente que, para o movimento da plataforma, será utilizada a composição de forças entre os eixos X e Y (0 e 1).

Se uma força de valor "100" for adicionada no eixo X do *nó Force*, seu objeto se movimentará com velocidade proporcional a essa força, na direção X (para a direita). Se o valor dessa força fosse "-100", o objeto se movimentaria com a mesma velocidade e no mesmo eixo, mas na direção oposta (para a esquerda). O mesmo acontece para o eixo Y. Se uma força de valor "100" for adicionada no eixo Y do *nó Force*, seu objeto se movimentará com velocidade proporcional a essa força, na direção Y (no caso da figura acima, para longe da câmera). Se o valor dessa força fosse "-100", o objeto se movimentaria com a mesma velocidade e no mesmo eixo, mas na direção oposta (no caso da figura acima, para perto da câmera). Mas, se uma força de valor "100" for aplicada em ambos os eixos X e Y, o veículo executará um movimento em um ângulo de 45 graus em relação à posição desses eixos.

O valor da propriedade *Orientation* ("0" ou *Heading* no eixo do exemplo) do *nó* Plataforma pode assumir os seguintes valores quando alterado pelo *nó Torque*, figura 31:

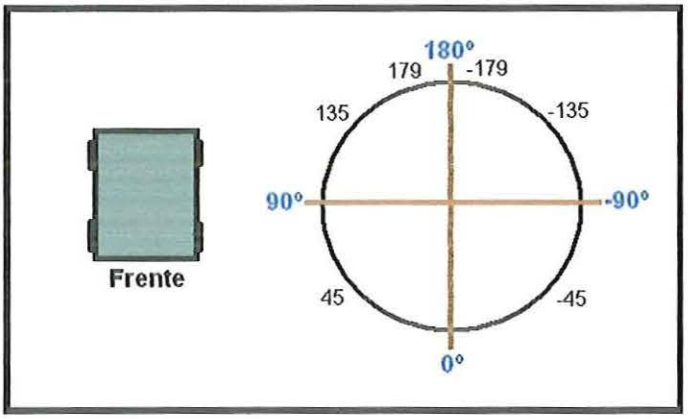

Figura 31 - Possíveis valores de *Heading*.

A implementação do movimento que será descrita abaixo teve como base o círculo de valores da figura acima.

Quando o *nó* Plataforma estivesse na posição "0 graus" ou "180 graus" e recebesse uma força de aceleração (pelo pressionamento da Seta Direcional para Cima), seu movimento deveria ser retilíneo ao longo do eixo X, não possuindo qualquer força em Y. Portanto, nesse caso, toda a força da aceleração deveria ser aplicada em X (propriedade X ou "0" do *nó Force*). O mesmo aconteceria para o eixo Y. Se a plataforma estivesse na posição "90 graus" ou "-90 graus", toda a aceleração deveria ser aplicada ao eixo Y (propriedade Y ou "1" do *nó Force*). Em qualquer outro caso, a aceleração deveria ser aplicada em ambos os eixos X e Y. Por esse motivo, o código referente à aplicação da força à plataforma executa testes (comandos "*if*") para saber em que ângulo a plataforma está posicionada. O posicionamento da plataforma foi dividido em quadrantes. O código para aceleração necessita apenas da informação do quadrante atual da plataforma, pois são nesses quadrantes que os valores de X e Y possuem combinações diferentes de sinais, figura 32:

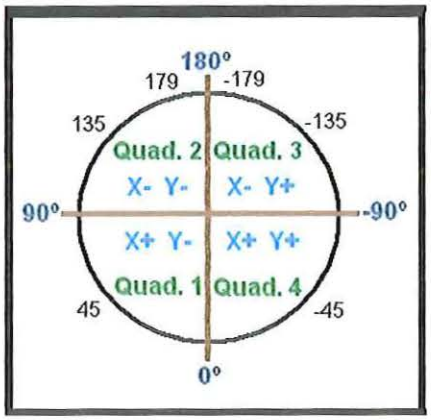

Figura 32 - Sinais dos eixos nos quadrantes.

Os valores de força de X e Y seguem uma proporção para cada quadrante. Essa proporção toma como base o valor da força máxima que um eixo pode assumir (posições 0°, 90°, 180° e -90°). Supondo que o valor absoluto máximo que a força possa assumir seja "1000", então:

- Na posição "0°" a força será de "1000" em X;
- Na posição "90°" a força será de "-1000" em Y;
- Na posição "180°" a força será de "-1000" em X;
- Na posição "-90°" a força será de "1000" em Y;

Abaixo serão explicados, primeiramente, os cálculos das forças no eixo Y e em seguida os referentes ao eixo X.

No primeiro quadrante as posições de orientação da plataforma variam de valores maiores que 0° até valores menores que 90°, portanto a linha de teste desse quadrante no *Script* é:

```
If ( ( Orient < 90 ) and ( Orient > = 0 ) ) then
```

Nesse quadrante, a força aplicada no eixo Y da plataforma (*F(1)*) possui o seguinte código:

```
F(1) = -( Orient * X )
```

O valor da força *F(1)* depende apenas do valor da orientação da plataforma e da constante X. Essa constante é definida em função da velocidade máxima (definida no próprio código) restringida para a aceleração do objeto. Seu valor é:

X = Velocidade / 90

Portanto, se a velocidade máxima definida tiver o valor "1000" (como no exemplo acima), o cálculo da força de *F(1)* ficaria da seguinte maneira:

*F(1) = - (Orient * 1000 / 90)*

A tabela 4 abaixo mostra alguns valores calculados para *F(1)* em função da orientação da plataforma:

Tabela 4 - Cálculos de força para o primeiro quadrante.

Orientação	**Força em Y ou *F(1)***
0º	0
15º	-166,666
45º	-500
60º	-666,666
89.9º	-999,99

Pela tabela é fácil notar que não existe força em Y quando a plataforma está em sua posição "0 graus". Essa força cresce conforme o ângulo da plataforma aumenta. Quando a plataforma atinge a posição 45 graus, ela se encontra exatamente entre os eixos X e Y, portanto sua força em Y assume o valor 500 (metade da força máxima). Nesse momento, a força em X também estará valendo 500, como será mostrado abaixo.

A soma das forças de X e Y sempre resultará no valor máximo definido para a força, não importando o ângulo de orientação.

Ainda pela tabela, é possível perceber que conforme o ângulo de orientação se aproxima de 90 graus, a força em Y se aproxima da força máxima definida. Quando esta chegar exatamente a 90 graus, não existirá mais força em X, pois toda a força estará sendo aplicada em Y.

O segundo quadrante funciona de maneira semelhante. Seu código de teste é:

```
if ( Orient >= 90 ) then
```

Os comandos desse teste serão executados quando o valor de *Heading* da plataforma estiver entre 90 graus e 180 graus (não existe valor maior que 180 graus para a orientação de um objeto).

O cálculo da força em Y para esse quadrante é realizado pelo seguinte código:

```
F(1) = - ( 2 * Y - Orient * X )
```

A constante Y guarda o valor máximo permitido para a força (no caso do exemplo Y = 1000).

A tabela 5 abaixo mostra alguns cálculos para os valores de *F(1)* no segundo quadrante:

Tabela 5 - Cálculos de força para o segundo quadrante.

Orientação	**Força em Y ou *F(1)***
90º → - (2*1000 – 90 * 11,111)	-1000
135º → - (2*1000 – 135 * 11,111)	-500
180º → - (2*1000 – 180 * 11,111)	0

Em todos os quadrantes, a o valor absoluto das forças em Y ou em X podem variar de "0" a "1000", mudando apenas seu sinal. Até o momento foram explicadas apenas as forças de Y nos quadrantes 1 e 2. Essas variam seu valor entre "0" e "-1000".

Para os quadrantes três e quatro, as linhas de teste e os cálculos da força em Y são feitos de maneira semelhante às descritas acima:

Teste para quadrante 3:

```
if ( Orient <= - 90 ) then
```

Cálculo da força para quadrante 3:

```
F(1)= (2*Y + Orient * X)
```

Teste para quadrante 4:

```
if ((Orient>-90) and (Orient<0)) then
```

Cálculo da força para quadrante 4:

```
F(1) = -(Orient * X)
```

Foram descritos acima os cálculos referentes às forças do eixo Y quando a Seta Direcional para Cima fosse pressionada. Abaixo serão explicadas as linhas de cálculos para a força no eixo X.

As forças em X não precisam ser calculadas para todos os quadrantes. Foi criada apenas uma linha de código que calcula essa força. Antes de utilizar esse código, é preciso transformar o valor da orientação para positivo (caso essa possua um valor negativo). Esse passo é executado pelo teste abaixo:

```
if ( Orient < 0 ) then
    Orient = Orient * ( -1 )
end if
```

O cálculo da força em X é realizado pelo código abaixo:

```
F(0) = ( Y - Orient * X )
```

A tabela abaixo mostra alguns valores calculados para a força em X (com Y = 1000 e X = 1000 / 90):

Tabela 6 - Valores de força no eixo X em função da orientação.

Orientação	**Força em X ou *F(0)***
0	1000
45	500
90	0
135	-500

180	-1000
-135 → convertido para 135 positivo	-500
-90 → convertido para 90 positivo	0
-45 → convertido para 45 positivo	500

O cálculo da força em X é simples, pois seu valor deve ser o mesmo para orientações positivas ou negativas. A figura 33 abaixo mostra um exemplo disso. Para o valor "45 graus" e "-45 graus", o valor da força no eixo X será o mesmo:

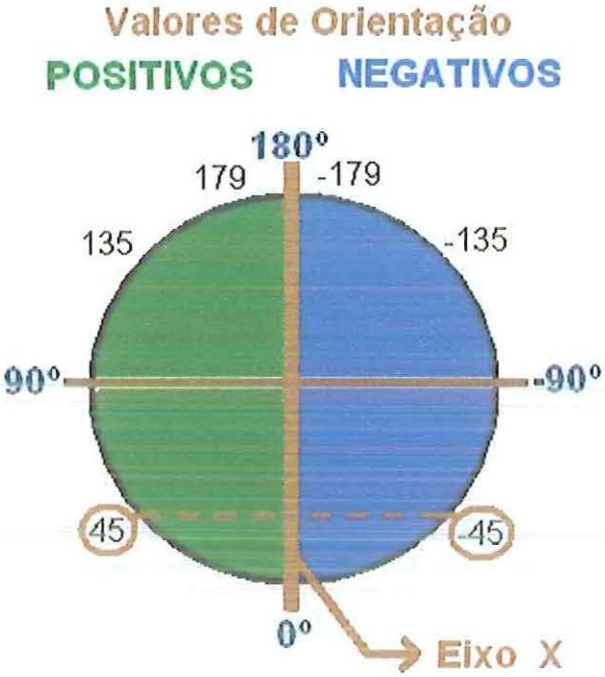

Figura 33 - Valores de orientação positivos e negativos.

O código da movimentação completa da plataforma é mostrado no apêndice A e será comentado abaixo:

- A variável "AceRet" informa se o veículo está em aceleração ou retrocesso (1= aceleração e 0 = retrocesso);
- Os procedimentos SetaDir e SetaEsq, referentes aos movimentos angulares do veículo, verificam se esse está em aceleração ou retrocesso antes de aplicar a força de torque. O sinal da força de torque no movimento de aceleração será o oposto do sinal no retrocesso;

- Os procedimentos SetaBaixo e SetaCima são chamados logo após a adição da força de torque em SetaDir e SetaEsq. Dessa forma, quando o veículo está em movimento e executa um movimento de rotação, a direção de sua força linear é corrigida. Se esses procedimentos não fossem chamados, o veículo executaria o movimento de rotação, mas seu movimento linear continuaria na direção antiga;
- Depois de executado um movimento de rotação, os procedimentos SetaDir e SetaEsq chamam o procedimento Zerar_Tor para que o movimento de rotação seja limitado;
- Os *nós* Zerar_Tor e Zerar_Vel são associados aos seus respectivos *nós TimeSensor* (essa associação foi explicada em tópicos anteriores);
- Os procedimentos SetaBaixo e SetaCima têm funcionamento semelhante. Apenas alguns sinais diferem um do outro;
- As constantes X (valor alterado na força com a mudança de um grau ou Y/90) e Y (valor da força máxima) dos procedimentos SetaDir e SetaEsq foram declaradas para facilitar a mudança de valores na fase de teste do projeto.

4.1.5 CRIANDO E IMPORTANDO A GEOMETRIA DA PLATAFORMA

Até essa fase do projeto, apesar de possuir movimentos semelhantes à realidade, a geometria da plataforma e suas texturas eram muito simples.

Nesse tópico do trabalho será comentado o procedimento de criação da geometria final da plataforma. O *software* utilizado para essa criação foi o 3DStudio MAX. Abaixo será feita uma breve introdução das funções básicas desse *software*.

4.1.5.1 3D STUDIO MAX

As figuras 3D do ambiente virtual foram modeladas no software 3D *Studio* Max 4.0. Neste tópico serão descritos os métodos usados na criação e importação das geometrias.

Os objetos que compõem o ambiente são: Plataforma agrícola móvel; Milharal; Terreno regular; Terreno montanhoso; Solo da plantação com linhas de plantio; Terreno com buracos; Pedras e muretas; Água; Casa.

Nem todos os objetos foram criados em 3D *Studio*. Os objetos mais simples, como os terrenos regulares e as muretas, foram criados dentro do próprio EON *Studio*.

Como visto na revisão bibliográfica, o EON *Studio* 4.0 fornece alguns protótipos prontos, que auxiliam na criação de geometrias. Porém, esses protótipos possuem poucas funcionalidades, não possuindo métodos para aplicação de texturas e re-dimensionamento. A versão mais atual do EON *Studio* (versão 5.1) possui mais recursos para a produção interna de geometrias. Além dos mesmos protótipos da versão anterior, é possível utilizar dois novos *nós*, uma esfera (*sphere*) e um retângulo (*Box*), para a produção de geometrias. Estes aceitam a aplicação de texturas e a operação de re-dimensionamento.

Os terrenos regulares e as muretas foram implementados com a utilização do *nó Box*, pois estes são apenas retângulos re-dimensionados.

O restante dos objetos foi criado no software 3D *Studio* Max 4.0.

4.1.5.1.1 O Software

O software 3D *Studio* MAX 4.0. foi escolhido para a implementação de geometrias do projeto devido a sua interface simples e a sua capacidade de exportar arquivos no formato 3DS. Outro fator que influenciou na decisão foi a exatidão da exportação. Apesar de não exportar cores e texturas, as geometrias dos objetos no formato 3DS sempre eram fieis quando passadas ao EON *Studio*.

Os tipos de objetos base disponíveis no 3D *Studio* são: *Box*; *Sphere*; *Cylinder*; *Torus*; *Teapot*; *Cone*; *Geosphere*; *Tube*; *Pyramid*; *Plane*.

Os objetos utilizados no projeto foram:

- Box, *Sphere*, *Cylinder*: para a construção da plataforma;
- Plane: para a construção dos terrenos irregulares;
- *Sphere*: para construção de pedras irregulares (com a utilização da função *noise* que será explicada a seguir).

Para exportar um arquivo em formato 3DS deve-se escolher a opção *Export* ou *Export Selected* no menu Arquivo.

A opção *Export* exporta todo o ambiente como um arquivo 3DS e a opção *Export Selected* exporta apenas o objeto selecionado.

Abaixo serão descritas algumas funções do 3D *Studio* MAX 4.0 que foram utilizadas no projeto.

4.1.5.1.2 A função Bend

A função *Bend* faz parte de uma série de ferramentas especiais para deformações de objetos encontradas no menu *Modifiers* do 3D *Studio* MAX. Sua função é curvar objetos ao longo de um de seus eixos.

Essa função foi usada no projeto para a criação do solo da linha de plantio.

Para utilizar essa função basta selecioná-la no Menu *Modifiers* dentro de *Parametric Deformers*.

Feito isso, é exibido o menu de opções da função *Bend*. Nesse menu pode-se escolher o eixo onde será aplicada a função, a direção e o valor do ângulo.

4.1.5.1.3 A função Noise

Para a criação de pedras no ambiente foram utilizadas esferas alteradas pela função *Noise*. Essa função deforma o objeto baseado em seus eixos.

Para utilizar a função *Noise* basta selecionar a opção *Noise* no Menu *Modifiers/Parametric Deformers*.

A figura 34 abaixo mostra o resultado final de uma deformação aplicada nos eixos X Y e Z de uma esfera. Esse tipo de objeto foi usado como pedra no ambiente da plataforma agrícola móvel, para testes de colisão e testes de superação de obstáculos.

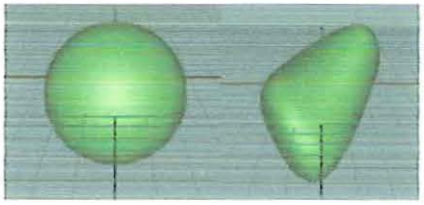

Figura 34 - Esfera normal (esquerda), esfera deformada (direita).

4.1.5.1.4 A função Collapse

Um objeto é composto de vértices, planos e polígonos. No 3D *Studio* pode-se modificar um objeto transformando-o em um conjunto de vértices, conjunto de planos ou conjunto de polígonos. Para isso existem algumas funções que devem ser utilizadas.

Inicialmente deve ser explicitado o número de segmentos de cada eixo do objeto. Por exemplo, a figura 35 abaixo mostra um cubo dividido em 10 segmentos em cada eixo:

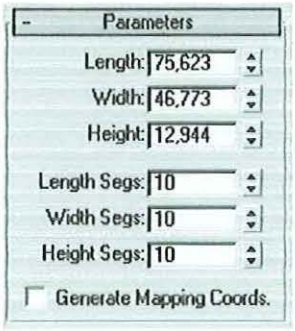

Figura 35 - Quantidade de segmentos de um cubo.

Essa especificação é importante, pois ela altera o número de vértices, planos e polígonos de um objeto.

Depois de criado o objeto, ainda não é possível selecionar, por exemplo, apenas um vértice do mesmo. Para isso a função *Collapse* deve ser utilizada.

Depois de usar a função *Collapse Selected*, o objeto pode ser dividido em vértices, como mostra a figura 36:

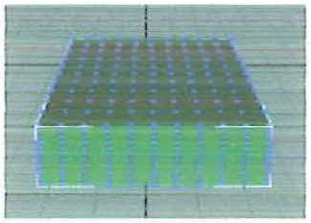

Figura 36 - Objeto dividido em vértices.

Quando um objeto é dividido em vértices, é possível selecionar apenas alguns de seus objetos, movê-los, apagá-los etc. Dessa maneira foram criados os objetos Antena e GPS da plataforma agrícola. Em ambos os casos foram criadas esferas e apagados alguns de seus pontos para produzir uma meia-esfera, figura 37.

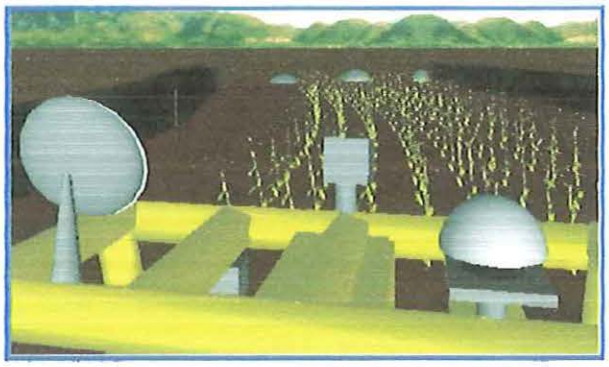

Figura 37 - Geometrias produzidas com o uso da função *Collapse*.

Além das geometrias da Antena e do GPS, foram criados, também, buracos num plano. Para isso foi criado um plano com 42 X 15 segmentos e foram movidos alguns de seus vértices com relação ao eixo Z (altura) , figura 38.

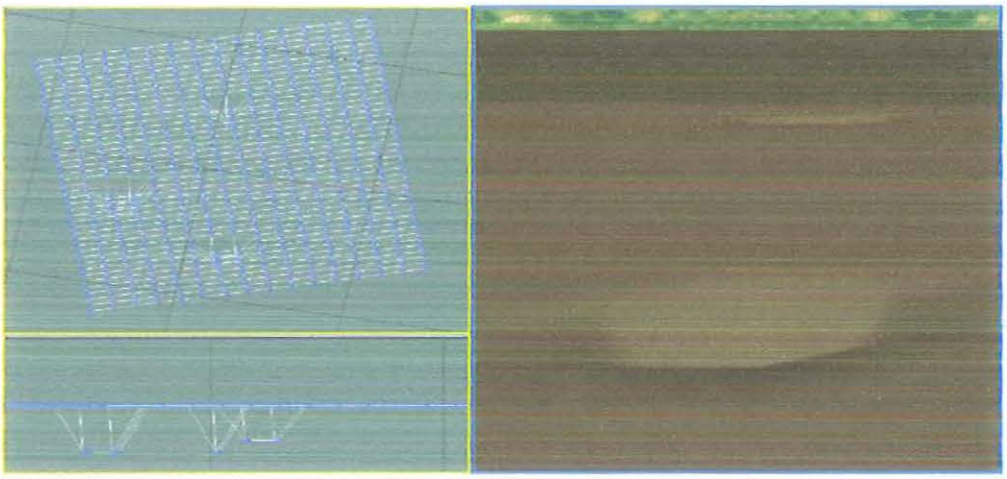

Figura 38 Vértices deslocados para a criação de buracos no plano.

A figura acima mostra o resultado final da criação de buracos em 3D *Studio* depois de o arquivo ser exportado para o EON *Studio*.

4.1.5.1.5 A função Group

Quando um projeto se torna muito grande, é aconselhável que os seus componentes sejam nomeados da maneira correta, com alguma lógica. Um exemplo disso é a árvore de cena do EON *Studio*. Ao final do projeto, a árvore de cena do ambiente da plataforma agrícola móvel contava com centenas de objetos. Seria

praticamente impossível trabalhar num ambiente com tantos objetos, sem que esses fossem nomeados de uma maneira adequada.

Quando um objeto é importado do 3D *Studio* para o EON *Studio*, ele é disponibilizado embaixo de um frame que possuirá o nome do arquivo 3DS. Cada geometria componente desse arquivo possuirá, também, um frame com seu nome.

No caso de um objeto que possua diversas geometrias, como é o caso da plataforma agrícola, é aconselhável utilizar a função *Group* do 3D *studio*. Essa função agrupa um conjunto de objetos para que estes possam ser movidos ou dimensionados em conjunto. Objetos agrupados serão importados para o EON *Studio* embaixo de um frame com o nome do grupo, tornando mais simples a árvore de cena.

Para agrupar dois ou mais objetos basta selecioná-los e usar a função *Group* do menu.

A sub-árvore da plataforma agrícola importada ficou da seguinte maneira, figura 39:

Figura 39 - Sub-árvore da Plataforma Agrícola.

4.1.5.1.6 A função Scale

Um objeto 3DS importado para o EON *Studio* só pode ser re-dimensionado no momento de sua importação. Mesmo assim, esse re-dimensionamento tem que ser proporcional entre os três eixos X, Y e Z. Portanto não é permitido alongar um objeto, isso deve ser feito na ferramenta de modelagem, no caso, o 3D *Studio*.

Há a possibilidade de importar o objeto para dentro de um frame e então re-dimensionar esse frame, dessa forma todos os objetos dentro desse frame serão

alongados da maneira que o usuário definir. Porém, dessa forma os objetos não se comportarão da maneira correta numa colisão. A melhor maneira trabalhar com objetos importados é dimensioná-los da maneira correta já na ferramenta de modelagem e somente depois importá-los para o EON *Studio*.

Para dimensionar um objeto há no 3D *Studio* uma ferramenta *Scale* (localizada ao lado das ferramentas Move e *Rotate*). Essa ferramenta permite alongar um objeto em qualquer um de seus eixos.

4.2 Outras Ferramentas de Desenvolvimento Virtual

No início do projeto foram propostas três ferramentas a serem estudadas, para avaliar qual seria a mais adequada, para o desenvolvimento do protótipo virtual da plataforma agrícola móvel. No entanto, durante o desenvolvimento do projeto, essas três ferramentas mostraram algumas limitações (falta de recursos para a implementação de veículos, colisões, centros de massa distribuídos, entre outras coisas) que forçaram o estudo de novas ferramentas.

Foram analisadas duas novas ferramentas: Webots e EON Professional.

Neste tópico serão descritos os estudos dessas novas ferramentas, suas limitações e a implementação final do projeto em EON Professional.

4.2.1 WEBOTS

Abaixo são descritas algumas características do software Webots retiradas do manual do usuário.

Webots é um simulador 3D de robôs móveis que permite aos seus usuários simular diferentes tipos de robôs, como por exemplo, robôs movidos por rodas, pernas ou mesmo robôs aéreos. Os seus usuários podem criar mundos virtuais complexos e simular seus robôs dentro desse ambiente. Junto ao *software* é fornecida uma biblioteca completa de programação que permite aos usuários programarem seus robôs (geralmente usando a linguagem C, C++ ou Java). Dos programas de controle é possível ler valores de sensores e enviar comandos aos motores dos robôs. O resultado dos controladores dos robôs pode ser transferido para robôs reais (robô Khepera com controladores C e robô Hemisson com controladores BotStudio).

O Webots possui um número de características essenciais que tornam a simulação fácil de usar e poderosa:

- Modela e simula qualquer tipo de robô móvel, incluindo rodas, pernas ou mesmo robôs aéreos;
- Possui uma biblioteca completa de sensores e atuadores;
- Permite a programação de robôs em C, C++ e Java, ou através de outros softwares (MatLab®, LabView®, Lisp® etc.) via TCP/IP;
- Transfere controladores para robôs móveis reais, incluindo Aibo®, Lego Mindstorms®, Khepera®, Koala® e Hemisson®;
- Usa a biblioteca ODE (Open Dynamics Engine) para aprimorar as simulações físicas. Pode-se especificar uma matriz para distribuição de massa, coeficientes de atrito, coeficientes de elasticidade etc.;
- Cria vídeos AVI ou MPEG para apresentações ou distribuição pela Internet;
- Inclui muitos exemplos com códigos de controladores e modelos de robôs comerciais;
- Permite a simulação de sistemas multi-agente, com facilidades para a comunicação global e local;
- É fornecida uma biblioteca de sensores;
- É fornecida uma biblioteca de atuadores;
- Pode-se criar ambientes complexos utilizando as capacidades da tecnologia OpenGL;
- Permite a importação de arquivos VRML97;
- Podem-se criar ambientes tão grandes quanto forem necessários, pois o Webots os otimizará, para que a simulação execute mais rapidamente;
- Possui métodos para facilitar a detecção de colisão;
- Pode controlar Torque, Posição ou Velocidade de servo-mecanismos

O sistema de simulação do Webots utiliza o tempo virtual, tornando possível executar simulações de forma muito mais rápida do que num robô real. Dependendo da complexidade do ambiente e do poder computacional da máquina, as simulações podem rodar até 300 vezes mais rapidamente do que num robô real, quando o modo Fast Simulation é usado. O passo de tempo da simulação pode ser ajustado de acordo com a necessidade (precisão versus velocidade). Um modo passo a passo ajuda a estudar em detalhes o comportamento dos robôs. (Oliver Michel, 2004).

Uma característica importante do software Webots para o desenvolvimento do projeto é a capacidade de utilização da biblioteca ODE. Algumas características dessa biblioteca foram retiradas do manual do usuário da ODE e são mostradas abaixo.

4.2.1.1 Open Dynamics Engine (ODE)

ODE é uma biblioteca gratuita para a simulação de dinâmicas de corpos rígidos articulados. É rápida, flexível e robusta e proporciona métodos para detecção de colisão. ODE está sendo desenvolvida por Russel Smith com a ajuda de vários contribuintes. Uma estrutura articulada é criada quando corpos rígidos de vários formatos são conectados por juntas de vários tipos. Os exemplos mais comuns são veículos terrestres (onde as rodas são conectadas por chassis) e criaturas que andam sobre pernas (onde as pernas estão conectadas ao corpo).

ODE foi desenvolvida para ser usada em simulações interativas ou em tempo real. Devido à sua velocidade e estabilidade, é particularmente ideal para a simulação de objetos móveis em ambientes dinâmicos de realidade virtual. O usuário tem total liberdade para mudar a estrutura do sistema, mesmo com a simulação em execução.

ODE utiliza um integrador altamente estável, para que os erros das simulações não saiam de controle.

ODE possui contatos rígidos. Isto significa que uma restrição de não-penetração de objetos é usada na colisão de dois corpos. Outra alternativa usada em muitos outros simuladores, é a utilização de molas virtuais para a representação de contatos (propícia a erros).

Principais Características:

- Corpos rígidos com distribuição de massa arbitrária;
- Tipos de juntas: ball-and-socket, hinge, slider (prismatic), hinge-2, fixed, angular motor, universal;
- Primitivas para colisão: Esfera, Cubo, Cilindro, Plano, Raio e malha triangular;
- Um integrador de primeira ordem está sendo utilizado. Rápido, mas não preciso suficiente. Integradores de ordens mais altas estão sendo produzidos;

No Webots pode-se utilizar a biblioteca ODE por meio de uma interface pré-implementada, não necessitando de programação. Porém, dessa forma, as funcionalidades da biblioteca se tornam muito restritas.

4.2.1.2 Uso do software

O software Webots possui uma interface simples, composta apenas por uma barra de ferramentas, um menu e uma janela de simulação.

Pelo menu Editar pode-se acessar a árvore de cena ou Scene Tree, onde são criados os objetos e propriedades da cena, figura 40.

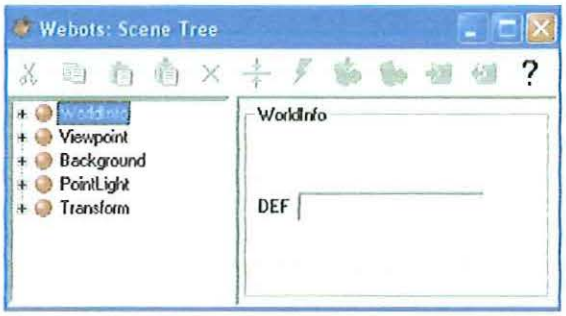

Figura 40 - Árvore de cena do Webots.

Os objetos iniciais da árvore de cena do Webots são:

- WorldInfo: contém o valor da gravidade da simulação (padrão = 0, -9.81, 0);
- Viewpoint: contém os valores que controlam o ponto de vista da simulação, como Position, Orientation, FieldOfView etc;
- Background: seu único objeto filho é SkyColor, que define a cor do plano de fundo da cena;
- PointLight: Fornece iluminação à cena. Define propriedades como intensidade, raio, posição etc;
- Transform: É responsável pela adição de objetos à cena. No caso do ambiente inicial do Webots, mostrado acima, o objeto adicionado foi o plano quadriculado.

Os nós WorldInfo, Viewpoint e Background são únicos para cada cena, não podendo ser duplicados. No entando, os nós PointLight e Transform podem ser duplicados, criando, respectivamente, novas luzes e objetos no ambiente.

Como dito anteriormente, para criar um objeto na cena do Webots podem ser utilizados os nós Transform ou Solid. Depois de adicionado um desses nós à cena, deve-se utilizar o seu campo "Children" para a criação de um objeto. Nesse campo, deve ser criado um nó Shape, responsável pelas geometrias e aparência de um objeto.

O nó Shape pode ser usado para a criação dos seguintes tipos de objetos:

- Sphere: cria uma esfera no ambiente;
- IndexedLineSet: representa uma geometria 3D pela criação de várias linhas;
- IndexedFaceSet: representa uma geometria 3D pela criação de várias faces;
- Extrusion: especifica figuras geométricas pela aplicação de extrusão em formas de duas dimensões;
- Cylinder: cria um cilindro no ambiente;
- Box: cria um retângulo no ambiente;

A detecção de colisão dos sólidos do ambiente é realizada pela adição de BoundingObjects aos mesmos. No nó Solid há um campo chamado BoundingObject, que permite a adição de esferas, cilindros e retângulos para a criação da geometria de colisão do objeto.

4.2.1.3 A implementação da Plataforma em Webots

Visto que as características do software Webots assemelhavam-se com as necessidades do projeto, tentou-se criar a Plataforma Virtual Agrícola Móvel no mesmo.

Um dos pontos negativos desse software é a capacidade de importação apenas de arquivos VRML97. Por esse motivo, algumas figuras do ambiente, como as figuras dos pés de milho, tiveram de ser criadas em 3DStudio Max, exportadas para VRML e somente depois importadas para o Webots. A exportação de uma figura para o formato VRML97 acarreta em algumas perdas de dados, porém esse não foi fator agravante para esse projeto.

Diferente do software EON Studio, o Webots não fornece estruturas prontas para a criação de texturas de solo, céu e horizonte. Seus recursos gráficos são limitados, pois seu foco está no teste de controladores de robôs.

Depois de importada a figura VRML97 referente ao milharal, tentou-se importar a figura da plataforma do 3D Studio Max. Os problemas com o VRML começaram aqui, pois a figura foi devidamente importada, porém o software não suportava a quantidade de objetos e logo disponibilizava uma mensagem de erro, fechando o programa. Devido a isso, não foi possível importar a geometria da plataforma para o Webots.

Resolveu-se, então, criar a geometria da plataforma novamente, apenas com os recursos disponíveis no Webots (esferas, cilindros, retângulos). O resultado da geometria da plataforma, solo, juntamente com a geometria do milharal é mostrado na figura 41 abaixo:

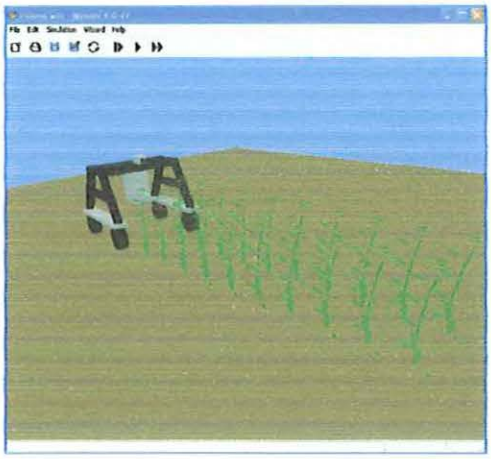

Figura 41 - Plataforma criada em Webots.

Apesar dos poucos recursos visuais fornecidos pelo software, o projeto foi levado adiante para verificação das propriedades físicas para a plataforma.

As geometrias da plataforma foram passadas para dentro de um nó DifferentialWheels, para que esta pudesse ser movimentada. Foi utilizado, inicialmente, um controlador fornecido com o software para a verificação do comportamento da plataforma. Esse controlador movimenta a plataforma com uma determinada velocidade e faz o controle da colisão pelos sensores de distância. Quando o objeto controlado chega a uma determinada distância de outro objeto, ele muda sua direção para que não haja colisão.

Devido à estrutura pronta para a criação de um robô móvel de apenas um eixo, primeiramente, a plataforma foi tratada como se esta não possuísse quatro rodas, mas sim duas.

O script utilizado foi criado em linguagem C e é mostrado no Anexo A.

Os níveis de tração (CoulombFriction), deslizamento (ForceDependentSlip), massa e densidade foram ajustados. Além disso, para cada roda da plataforma (neste caso a plataforma estava sendo tratada com duas rodas) foram criadas juntas com o uso do nó Joint.

EESC-USP
Biblioteca
CAMPUS DE S. CARLOS

No campo BoundingObject da plataforma foi criado um retângulo, de tamanho proporcional ao tamanho da plataforma, como geometria de colisão.

Para o teste das características físicas foi criado um terreno irregular, com os objetos da própria ferramenta, figura 42.

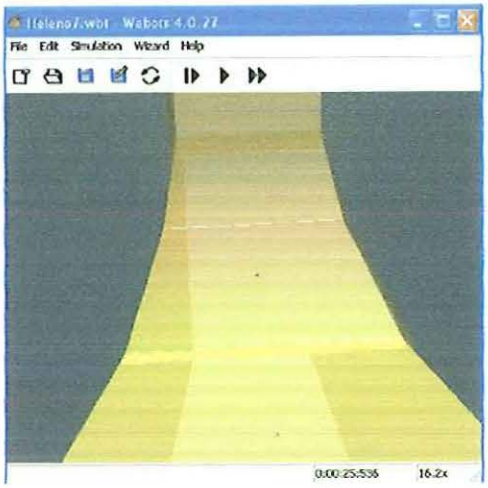

Figura 42 - Terreno irregular criado em Webots.

Os resultados obtidos não foram satisfatórios. A plataforma executava movimentos perturbados, que não puderam ser melhorados pela alteração de suas características.

4.2.1.4 Problemas encontrados

Além da baixa capacidade de importação de arquivos, dos problemas encontrados com tais importações e com a baixa capacidade gráfica do software, o principal fator que influenciou na decisão de abandonar o uso do Webots foi a dificuldade de implementação de veículos com dois eixos.

O software funciona muito bem para simulações de controladores de robôs que possuam apenas um eixo (apenas duas rodas). Na fase de avaliação desse software foi criado um veículo desse tipo e testadas algumas características da biblioteca ODE sobre o mesmo. Porém, a implementação de dois eixos num mesmo veículo não foi possível. Os tutoriais e exemplos fornecidos junto ao software não fornecem qualquer informação sobre esse tipo de implementação. Foi feito um contato com a empresa fabricante do software que afirmou ser possível tal tipo de implementação, porém não muito comum. A empresa também afirmou a não existência de exemplos gratuitos sobre veículos com dois eixos. O custo de um exemplo referente a esse assunto ultrapassa U$ 2.000,00.

4.2.2 EON PROFESSIONAL

As novidades do EON Professional com relação ao EON Studio 4.0 são:

- Novo sistema de colisão baseado na engine física da CMLabs;
- Novo nó IPlane que permite a configuração arbitrária de vetores complexos para a projeção de planos;
- O nó Clicksensor mostra o nó que está sendo clicado, para que possa-se implementar uma estratégia de detecção de cliques;
- Mapeamentos de ambientes podem ser gerados utilizando o RenderCubeEnvMap atualizado;
- Nova e atualizada documentação com melhor organização e estrutura;
- Nova ferramenta Mesh Merging que permite melhorar a performance da renderização tornando os objetos estáticos como apenas um objeto e removendo frames desnecessários;
- Importa formato SolidWorks;
- Suporte a novos dispositivos de visualização;

Existem quatro módulos principais no EON Professional, são eles:

- Módulo de Efeitos Visuais;
- Módulo de Propriedades Físicas;
- Módulo de Humanos RPC;
- Módulo de Sombras.

Neste tópico serão descritos os módulos de efeitos visuais e de propriedades físicas e serão apresentados os módulos de humanos e sombras.

4.2.2.1 Módulo de Efeitos Visuais

Este módulo habilita o usuário a utilizar shaders em sua aplicação. Shader é um material programável. Sua aparência é definida por um pequeno programa (Cg-programs), que são programados em uma linguagem chamada Cg. Esta linguagem é especialmente desenvolvida para que os usuários possam controlar os vários aspectos da aparência em uma programação de alto nível.

Os seguintes nós referentes aos materiais Cg são disponíveis:

- LeatherMaterial: cria um material de couro realista usando bumpmapping (mapeamento de irregularidades).
- HDRMaterial: iluminação baseada em imagem usando imagens HDR (high-dynamic range) como cubemaps (algumas texturas que mapeiam o ambiente como um todo) , figura 43.

Figura 43 - Exemplo de HDRMaterial.

- HatchMaterial: non-photorealistic shading (sombreamento) no estilo hatch. Similar a uma ilustração técnica. Requer o uso de seis texturas inseridas automaticamente junto ao nó Material, figura 44.

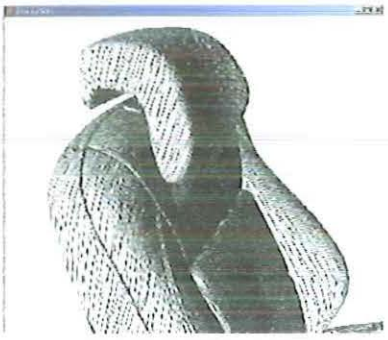

Figura 44 - Exemplo de HatchMaterial.

- MetalFlakesMaterial: um shader metálico completo. Simula superfícies envernizadas, geralmente de carros ou motos.
- MultiLayerMaterial: um nó de material com layers que permitem a combinação de um mapeamento de cor, mapeamento de irregularidades, mapeamento de sombras e mapeamento do ambiente, além das propriedades normais dos materiais, figura 45.

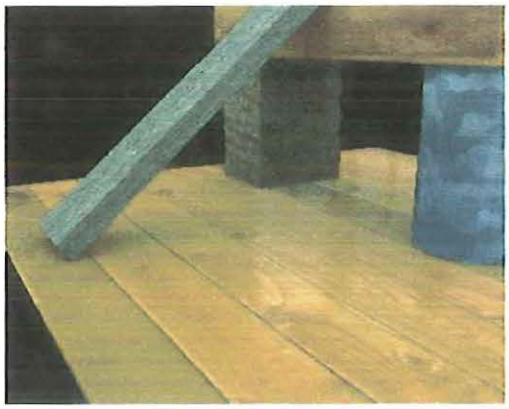

Figura 45 - Solo com MultiLayerMaterial.

Não há luzes nessa cena, apenas Maps aplicados aos objetos, como mostra a figura 46 abaixo:

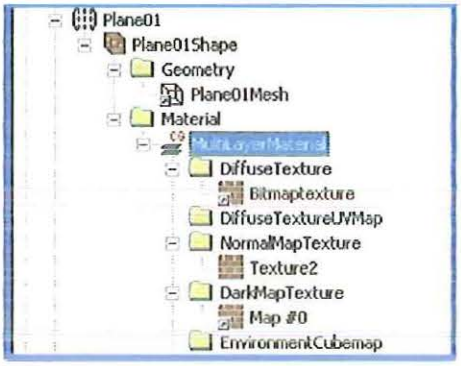

Figura 46 - Árvore de cena do solo, com os respectivos Maps.

4.2.2.2 Programas Cg

O nó CgMaterial é o nó de material Cg mais genérico e permite a modificação dos programas Cg pelo próprio usuário.

Além do nó CgMaterial e dos nós descritos acima, são incluídos no EON Professional os seguintes programas Cg:

- Water: um shader de água que perturba um plano para criar ondas. É possível mudar a altura das ondas, o comportamento, a reflexão e as cores, figura 47.

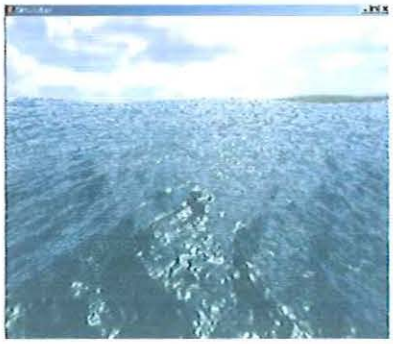

Figura 47 - Exemplo de Programa Cg Water.

- Wood: cria texturas de madeira 3D em um objeto. A textura é criada de maneira procedimental, portanto não há necessidade de especificar as coordenadas da textura para a geometria.
- Granite: semelhante ao Wood, mas cria uma textura de granito/mármore.
- Satin: um shader de tecido como veludo ou seda. Reflete de maneira diferente, dependendo da direção do visualizador.
- Glass: cria vidros que aplicam reflexão e refração nas luzes. Simula materiais transparentes.

Há a opção de criar efeitos de irregularidades nos vidros, como mostra a figura 48 abaixo:

Figura 48 - Exemplo de Programa Cg Glass com BumpMapping (irregularidades).

Para utilizar esses shaders usa-se o novo comando Import -> CgMaterial para adicioná-los à árvore de simulação. Eles dependem do nó CgMaterial genérico para criar o material para o objeto.

4.2.2.3 Técnicas avançadas de efeitos de luzes

Este tópico detalha as técnicas de efeitos visuais usadas para a criação dos efeitos citados acima.

4.2.2.3.1 Bump Mapping

Bump Mapping é uma técnica de renderização para simular efeitos de iluminação causados por irregularidades de superfícies. Codificando as características da superfície em mapeamentos de textura, bump mapping simula a iluminação de uma superfície irregular sem a complexidade e os gastos de processamento da modelagem de tais características como verdadeiras perturbações geométricas, figura 49.

Figura 49 - Bump Mapping.

4.2.2.3.2 Mapeamento de Ambiente

Environment mapping ou mapeamento de ambiente é uma técnica usada para renderizar um objeto que reflita inteiramente ou parcialmente seu ambiente. De uma maneira mais simples, o mapeamento de ambiente dá aos objetos renderizados uma aparência cromada, figura 50.

Figura 50 - CubeMap e Environment Mapping.

Os shaders Cg no EON Professional suportam um novo tipo de textura conhecido como cube map que consiste de não apenas uma, mas seis texturas. Juntas, essas imagens formam algo semelhante às faces de um cubo, ou seja, uma imagem ominidirecional.

Para usar um EnvironmentMapping basta selecionar a textura cube (formato .dds) map apropriada no campo EnvironmentCubeMap do shader. Para criar um CubeMap pode-se utilizar uma ferramenta como o DirectX Texture Tool incluído no Microsoft DirectX SDK (gratuito em MSDN.microsoft.com).

4.2.2.3.3 Dark Mapping

Dark Mapping ou mapeamento de sombras simula o efeito de uma luz estática em uma superfície. Uma DarkMap é uma textura que representa como a luz se comporta no contato com a superfície. Embora o DarkMapping seja usado para efeitos de luzes estáticas, ele também pode ser usado para criar efeitos de sombras estáticas, figura 51.

Figura 51 – DarkMapping

Para usar a técnica DarkMapping basta selecionar a textura apropriada no campo DarkMapTexture. Geralmente, um DarkMap é criado usando um pacote 3D de modelagem e animação como o Maya ou o 3Ds Max. Pode-se também utilizar ferramentas de pintura para criar DarkMaps próprios.

4.2.2.4 Propriedades principais de um Cg Shader

4.2.2.4.1 Culling

Da mesma maneira que um material normal, um material Cg também pode ser configurado para renderizar somente polígonos que estejam visíveis ou renderizar todos os polígonos. Os campos Culling e TwoSidedLightining são usados para esse controle.

4.2.2.4.2 Material Preset File

Como a maioria dos materiais Cg são muito versáteis, pode-se criar diversas variações de materiais num único nó de material Cg, variando os valores dos parâmetros do material. O usuário pode criar um arquivo de configuração, onde esses valores são descritos. O formato desse arquivo é muito simples:

```
[category]
parameter1=value
parameter2=value2
...
```

Por exemplo, para definer um arquivo de configuração para dois materiais: Cobre e Ouro, para o nó HDRMaterial, o usuário pode escrever:

```
[Cobre]
BaseColor = 0.514 0.31 0.004
EnvironmentLevel = 0.5
GlossLevel = 3
DiffuseLevel = 1
Reflectivity = 0.1

[Ouro]
BaseColor = 0.992 0.761 0.067
EnvironmentLevel = 10
GlossLevel = 4
DiffuseLevel = 1
Reflectivity = 0.5
```

4.2.2.5 Módulo de Propriedades Físicas

Utilizando a tecnologia Vortex da CMLabs, o módulo de dinâmica possibilita uma simulação de corpos rígidos consistente em EON Studio. Corpos Rígidos são objetos com propriedades físicas, como massa e volume. Eles não mudam de forma

durante a simulação (não sofrem deformações). A dinâmica de corpos rígidos no EON Professional torna possível o movimento realista de objetos, influenciado por forças, juntas e interação com outros corpos rígidos.

Os dois principais nós do módulo de Propriedades Físicas são o nó RigidBody e o nó DynamicsManager. O nó RigidBody possui as propriedades específicas de cada corpo rígido, como massa e velocidade, enquanto o nó DynamicsManager representa as propriedades globais da cena, como gravidade.

Os corpos rígidos são colocados embaixo de nós Frame mostrando que estes representam corpos rígidos.

O nó DynamicsManager gerencia um conjunto de corpos rígidos e computa de forma contínua suas posições, orientações e velocidades. Diferentemente da versão 4.0 do EON Studio, nesse módulo do EON Professional os corpos rígidos são incorporados ao nó DynamicsManager de forma automática, assim que colocados na árvore de cena.

4.2.2.5.1 Unidades de medidas

O usuário pode escolher entre qualquer sistema de unidades: metro-kilograma-segundos, centímetro-grama-segundos ou pé-libra-segundos. No entanto, é importante ser consistente em relação à escolha do sistema de unidades quando alterar os valores das propriedades. Por exemplo, mudando a gravidade para -9.82 m/s^2 implica em usar o sistema de unidades SI (Metros, kilogramas, segundos e Newton) e isso significa que a densidade da água será 1000 kg/m^3. Na tabela abaixo são mostrados alguns valores de densidade:

Tabela 7 – Valores de densidade.

Material	Density
Água	1000 kg/m^3
Alumínio	2,700 kg/m^3
Chumbo	11,300 kg/m^3
Ferro	7,860 kg/m^3

Os valores padrões são definidos no sistema de unidades SI: Metros, Kilogramas, Newton e Segundos.

4.2.2.5.2 Propriedades de Massa

O objetivo das propriedades de massa são ajustar a distribuição de massa de um corpo rígido para que o centro de massa possa ser ajustado.

O EON Professional tenta balancear de maneira adequada a massa dos corpos rígidos baseado nas propriedades de massa descritas nos corpos rígidos e nos nós MassProperties.

No exemplo abaixo, a massa do Cubo Duplo (Double Cube) foi ajustada para 10kg nas propriedades do nó RigidBody. Mas, no nó MassProperties localizado embaixo do Frame LeftCube (heavy) está declarado que essa parte do corpo total pesa 8kg. O EON balanceará automaticamente o peso do Frame RightCube (light) de forma que este pese 2kg (para completar o valor total do corpo que foi declarado como 10kg).

O resultado mostra o centro de massa do corpo rígido deslocado para a esquerda, como mostra a figura 52 abaixo;

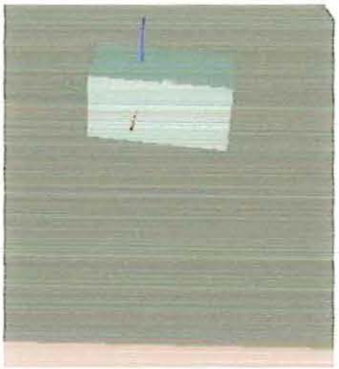

Figura 52 - Exemplo Cubo Duplo

4.2.2.5.3 Propriedades de Contato

As propriedades de contato controlam o comportamento da dinâmica quando dois corpos rígidos estão em contato. As propriedades normais de contato no EON Professional são *Friction* (tração) e *Restituition* (elasticidade). Como no mundo real, as propriedades de contato dependem das características dos dois corpos envolvidos no contato. No EON as características dos corpos são representadas nos seus materiais e as propriedades de contato são definidas para cada par de materiais. O conjunto de propriedades de contato para pares de materiais é chamado de definição de contato.

As propriedades de contato são definidas num arquivo texto chamado arquivo de definição de contato (*contact definition file*) que é referenciado pelo nó DynamicsManager. O formato do arquivo é bem simples:

```
[material1 - material2]
contact property 1=value 1
contact property 2=value 2
...
```

Por exemplo, para criar um arquivo de definição de contato envolvendo dois materiais chamados Madeira e Ferro, o usuário deveria escrever:

```
[Madeira - Madeira]
Restitution = 0.2
Softness = 0.0
AdhesiveForce = 0.0
FrictionType = Friction2D
FrictionModel = ScaledBox
FrictionSlip = 0.0 0.0
FrictionSlide = 0.0 0.0
BoxFriction = 1000
ScaledBoxFriction = 0.5
[Madeira - Ferro]
Restitution = 0.2
Softness = 0.0
AdhesiveForce = 0.0
FrictionType = Friction2D
FrictionModel = ScaledBox
FrictionSlip = 0.0 0.0
FrictionSlide = 0.0 0.0
BoxFriction = 1000
ScaledBoxFriction = 0.5
```

```
[Ferro - Ferro]
Restitution = 0.2
Softness = 0.0
AdhesiveForce = 0.0
FrictionType = Friction2D
FrictionModel = ScaledBox
FrictionSlip = 0.0 0.0
FrictionSlide = 0.0 0.0
BoxFriction = 1000
ScaledBoxFriction = 0.5
```

4.2.2.5.4 Tipos de Geometrias de Colisão

Para melhorar a performance e a estabilidade do ambiente é altamente recomendado usar geometrias mais simples que a geometria visual, para o sistema de dinâmicas. Essa é uma regra geral das simulações de dinâmicas e geralmente alcança resultados satisfatórios. O princípio básico é envolver a geometria em um volume mais simples. Os três tipos de volumes disponíveis para esta função em EON Professional são:

1. Bounding Box (Caixa);
2. Bounding Sphere (Esfera);
3. Convex Hull (Casca Convexa).

Esses volumes podem envolver geometrias individuais ou mesmo um conjunto de geometrias.

Os dois primeiros tipos, caixa e esfera, gastam menos processamento e portanto são mais aconselhados, porém o tipo de volume a ser usado dependerá do formato da geometria. Há situações onde a geometria não pode ser simplificada, como um terreno onde as características não-convexas são importantes.

A escolha do tipo de geometria para colisão pode ser feito tanto nas propriedades do nó RigidBody como em um nó chamado CollisionGeometryType. Esse nó pode ser colocado em qualquer Frame referente a geometrias de um corpo rígido. Ele atua como uma exceção à regra geral definida nas propriedades do nó RigidBody.

As opções de geometria de colisão disponíveis no EON Professional são:

- None: exclui geometrias da colisão;
- Bounding spheres: envolve cada objeto por uma esfera;
- Bounding sphere: envolve o objeto completo por uma só esfera;
- Bounding boxes: envolve cada objeto por uma caixa;
- Bounding box: envolve o objeto complete por um só caixa;
- Convex hulls: envolve cada objeto por uma casca convexa;
- Convex hull: envolve o objeto complete por uma só casca convexa;
- Geometry as is: envolve cada detalhe do objeto.

4.2.2.5.5 Juntas

Uma junta é uma ligação entre dois corpos ou entre um corpo e o ambiente. Serve para a criação de corpos articulados. Exemplos de corpos articulados são: braços de seres humanos, suspensão e direção de um carro etc.

Uma junta em EON Professional é um ponto de ligação que pode ser conectado a outro ponto equivalente.

Tipos de Juntas

Há diferentes tipos de ligações, mas somente tipos compatíveis podem ser ligados para criar uma junta.

Há duas características que definem o tipo de ligação:

Joint Node Type

AngularJoint, BallAndSocketJoint, CarWheelJoint, HingeJoint, LineJoint, PlaneJoint, PrismaticJoint, RigidJoint, SpringJoint, UniversalJoint

Joint User Type

Esse tipo de junta é definido pelo usuário em um arquivo texto com a seguinte estrutura:

```
[User type name]
Joint property 1=value 1
Joint property 2=value 2
...
```

Por exemplo, os tipos padrões para a junta Hinge são:

```
[Default]
DisableCollision = 1
AutoConnect = 0
AutoConnectTranslation = 0 0 0
AutoConnectRotation = 0 0 0
ForceLimit = INFINITY
TorqueLimit = INFINITY
LimitPosition = INFINITY -INFINITY
LimitRestitution = 0 0
LimitStiffness = INFINITY INFINITY
LimitDamping = 0 0
LockForceLimit = INFINITY
LockStiffness = INFINITY
LockDamping = 0
LockAtLimit = 1
```

Juntas Masculinas e Femininas

Uma junta pode ser Masculina, Feminina ou Não-Especificada. As possíveis combinações de ligação são:

- Fêmea – Macho;
- Fêmea – Não-especificada;
- Macho – Não-especificada;
- Não-especificada – Não especificada.

Por exemplo, o ponto de ligação para um carro para a junta CarWheelJoint deve ser sempre fêmea e o ponto de ligação para as rodas devem sempre ser Machos ou Não-especificados.

Outra propriedade a ser declarada para juntas é a opção Dual ou Single. O valor padrão é Dual. Esse valor exige que essa junta seja conectada a uma outra junta. O valor Single permite que a junta possa ser conectada diretamente a outro corpo.

Nós das Juntas

- Nó Joint Manager: Gerencia um conjunto de juntas e seus estados. Como nos corpos rígidos, as juntas serão automaticamente conectadas ao gerenciador;
- Nó Angular Joint: Remove todos os graus de liberdades angulares das ligações. Isto significa que somente a translação é permitida. Há a opção de ativar somente a rotação relativa ao eixo Z;
- Nó BallAndSocket Joint: Remove todos os graus de liberdades lineares das ligações. Isto significa que somente a rotação é permitida. Esta junta possui um cone limitador;
- Nó Hinge Joint: Remove todos exceto um grau de liberdade angular de uma ligação. Isto significa que somente a rotação no eixo Z é permitida. É útil, por exemplo, na modelagem de portas;
- Nó CarWheel Joint: É uma combinação de juntas que modelam a junta de um carro com direção e suspensão;
- Nó Line Joint: Cria dependências da posição relativa ao longo dos eixos Y e Z. Resulta numa dependência de um corpo com relação a uma linha de um outro corpo;
- Nó Plane Joint: Cria dependências da posição relativa ao longo do eixo Z. Resulta numa dependência de um corpo com relação a um plano de um outro corpo;
- Nó Prismatic Joint: Remove todos exceto um grau de liberdade linear entre ligações. Isto significa que somente a translação relativa ao eixo Z é possível. É útil, por exemplo, na criação de elevadores.
- Nó Rigid Joint: Remove todos os graus de liberdade. Não é possível realizar movimentos. No entanto, há uma opção "soften" que permite a especificação de relaxamentos;

- Nó Spring Joint: Modela uma mola baseada nas leis de Hook (F = -k (d -l)), onde F é a força da mola, l é a constante de descanso, k é a tensão e d é a distância entre as ligações.
- Nó Universal Joint: Remove todos os graus de liberdade lineares e um grau de liberdade angular entre as ligações. É semelhante ao BallAndSocket.

4.2.2.5.6 Outros Nós Físicos

- Force: Aplica um vetor de força simples em um corpo rígido;
- Torque: Aplica um vetor de torque simples em um corpo rígido;
- Impulse: Aplica um vetor de impulso simples em um corpo rígido. Ao contrário dos nós Force e Torque, um impulso é uma mudança instantânea de velocidade. O mesmo acontece quando dois corpos colidem.

4.2.2.6 Módulo de Humanos RPC

Com este módulo é possível importar e usar personagens RPC (formato .RPC) animados da ArchVision dentro do EON. Esta tecnologia fornece um método fácil para a visualização de vários humanos em um ambiente que não exija muito da capacidade de renderização, pela utilização da técnica Billboard;

4.2.2.7 Módulo de Sombras

Permite adicionar um realismo extra para a simulação pela capacidade de implementar sombras realistas que podem ser dispostas em qualquer face da cena, incluindo faces não planas;

4.2.2.8 Sistema necessário para rodar o EON Professional

Rodar o ambiente (Run-Time):

Tabela 8 – Necessidades para rodar EON Professional.

Sistema Operacional	Detalhes
Windows 95/98/ME	Requer DirectX 6.1 ou superior.

	Para visualizar os materiais Cg é necessário o DirectX 9.0b ou superior. Materiais Cg só funcionam com OpenGL.
Windows NT 4.0	SP3 ou superior deve estar instalado. Materiais Cg não podem ser visualizados nesse sistema.
Windows 2000/XP	Para ver materiais Cg, o DirectX 9.0 ou superior deve ser instalado. Materiais Cg só funcionam com OpenGL.
Placas de Vídeo	**Versão**
NVidia	ForceWare 53.03 ou superior. FX 5200 ou superior é necessária par aver simulações com materiais Cg.
ATI	Catalysator 4.10 ou superior. Radeon 9500 é necessária para ver simulações com materiais Cg.
Outras	Driver mais atual.

Criar o ambiente (Authoring-Time):

Tabela 9 – Necessidades para o ambiente de criação do EON Professional.

Sistema Operacional	**Comentários**
Windows 95	Não suportado
Windows NT 4.0	Não suportado
Windows 98/ME/2000/XP	DirectX 9.0b ou superior é necessário
512 MB or more system memory	

4.3 Desenvolvimento da Plataforma Virtual Agrícola Móvel em EON Professional

No capítulo anterior foram discutidos os motivos da utilização do software EON Professional para a implementação final do protótipo virtual da plataforma agrícola móvel.

Neste capítulo serão descritos todos os passos do desenvolvimento.

A figura abaixo mostra a árvore de simulação final do projeto. Nela podem ser vistos os nós principais do projeto. Esses nós serão usados para dividir a explicação do desenvolvimento em tópicos.

Os nós principais (e conseqüentemente os próximos tópicos) do projeto são, figura 53:

- Plataforma: descreverá como foram implementadas as características de movimentação, colisão e outras funcionalidades da plataforma;
- Milharal: descreverá o sistema usado para animar o milharal;
- Milho_Col ou Colisão com o Milho: descreverá o sistema de colisão da plataforma com o milharal;
- Terrenos: descreverá a implementação dos terrenos irregulares;
- ViewPorts e Câmera: descreverá as funcionalidades ligadas às diferentes visões;
- Água: descreverá o sistema usado para a animação da água e deslizamento da plataforma;
- Outros Nós: descreverá os detalhes de outros nós menos importantes do sistema.

Figura 53 - Árvore de Simulação Final do Projeto.

Muitos passos do desenvolvimento do projeto no software EON Studio 4.0 foram aproveitados no desenvolvimento final. Portanto, todo o tempo gasto na implementação anterior foi fundamental para que a implementação final alcançasse os resultados esperados de uma maneira rápida e eficiente.

4.3.1 PLATAFORMA

O objeto Plataforma, criado em 3DStudio Max 4.0, não sofreu grandes alterações da implementação em EON 4.0 para a implementação final. Foi importado o mesmo objeto (extensão .3DS), porém, tiveram de ser realizadas algumas mudanças com relação à disponibilização dos frames internos à Plataforma, figura 54:

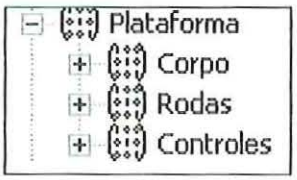

Figura 54 - Frames do objeto Plataforma.

Como mostra a figura acima, o objeto Plataforma foi dividido em Corpo, Rodas e Controles. A mudança mais importante para o projeto foi a disponibilização do frame Rodas no mesmo nível de hierarquia do frame Corpo e não como um nó filho de Corpo. Isso foi necessário devido ao fato de que nesse momento do projeto, as rodas deveriam funcionar como corpos rígidos, para que estas pudessem movimentar a plataforma. O frame Controles foi criado apenas para melhorar a organização do projeto.

A figura 55 abaixo mostra o conteúdo de uma das rodas da plataforma, sendo que as outras rodas funcionam de maneira semelhante.

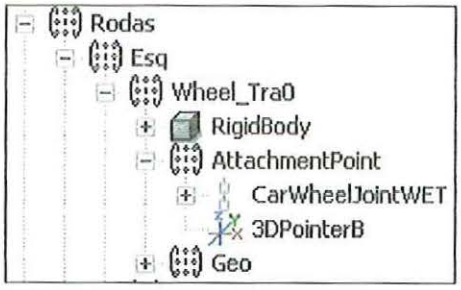

Figura 55 - Componentes das rodas.

Como dito anteriormente, para que as rodas fossem responsáveis pelo movimento da plataforma, foi necessário torná-las corpos rígidos. As propriedades importantes dos nós Rigid Body das rodas são:

Massa: 10.000.
CollisionGeometry Type: Bounding Sphere.

Foi importante definir o CollisionGeometry Type como Bounding Sphere para que o objeto referente à colisão da roda fosse uma esfera. Dessa maneira, a colisão da roda com o solo é realizada de maneira suave, possibilitando a movimentação da plataforma de maneira mais realista.

Alguns outros fatores são necessários para a implementação do movimento da Plataforma em função das rodas. Os dois fatores mais importantes são: a definição das juntas e a definição do atrito.

Cada uma das rodas da plataforma possui um objeto do tipo CarWheelJoint. Esse objeto é próprio para a definição de juntas de carros. As propriedades mais importantes que foram definidas nesses nós foram:

Gender: Male
Duality: Dual
SelectedPeer

Um veículo em EON Studio deve possui um nó CarWheelJoint em cada roda e mais um nó desse tipo localizado no corpo do carro, para cada roda. Por exemplo, para um carro com quatro rodas, deve existir um nó CarWheelJoint embaixo do Frame de cada roda e mais quatro nós CarWheelJoint embaixo do Frame referente ao corpo do carro.

A propriedade Gender das rodas deve ser Male e dos nós CarWheelJoint localizados no Corpo do carro deve ser Female:

Tabela 10 - Propriedade Gender para as juntas do carro.

CarWheelJoint no Frame Corpo	**CarWheelJoint Frames Rodas (1, 2, 3 e 4)**
Gender Female	Gender Male
Gender Female	Gender Male

Gender Female	Gender Male
Gender Female	Gender Male

Por sua vez, a propriedade Duality especifica que cada nó CarWheelJoint deve possuir, necessariamente, um par associado, cuja propriedade Gender seja compatível:

Male / Female
Unspecified / Male
Unspecified / Female
Unspecified / Unspecified

A propriedade SelectedPeer disponibiliza uma lista das juntas disponíveis (criadas) no ambiente, para que possam ser definidos, para cada junta, o seu respectivo par associado. Quando uma junta é associada à outra por essa propriedade, ela é removida da lista de juntas livres disponíveis.

Além dos nós CarWheelJoint, existe embaixo de cada Frame Roda um nó do tipo 3DPointerB. Esse nó mostra um eixo 3D (X, Y, Z) relativo ao Frame onde for adicionado. A sua utilização torna mais simples a visualização da posição e movimentação das rodas.

Definidas as quatro associações de juntas da plataforma, é necessário configurar de maneira correta as propriedades do nó DynamicsManager, para que o contato da plataforma com o solo seja realizado de maneira real. As propriedades do nó JointManager não precisaram sofrer alterações nesse projeto.

As propriedades do nó DynamicsManager mais importantes para o projeto foram:

Gravity:	0, 0, -9.81
Cp_Restitution:	0.1
Cp_Softness:	0.0001
Cp_AdhesiveForce:	0
Cp_FrictionType:	Friction2D
Cp_FrictionModel:	Scaled Box
Cp_FrictionSlip:	0
Cp_FrictionSlide:	0

Cp_BoxFriction:	0
Cp_ScaledBoxFriction:	1.0

O valor padrão da gravidade definida no EON Studio é "-9.81" no eixo Z (valor da gravidade da Terra).

O coeficiente de restituição define a elasticidade da colisão. O valor "0" indica uma colisão totalmente não-elástica, enquanto o valor "1" indica uma colisão totalmente elástica. Valores entre zero e um devem ser testados para verificar qual valor simula de maneira mais adequada o movimento real.

A propriedade Softness define a quantidade de penetração dos objetos numa colisão. É utilizada para definir colisões de objetos mais maleáveis ou macios. Um valor padrão para uma colisão normal é "0.0001".

AdhesiveForce atua como uma "cola" entre os corpos.

FrictionType define se a tração será aplicada em dois ou apenas um sentido (eixo). No caso de um veículo como a plataforma agrícola, a tração deve possuir o valor Friction2D.

O valor da propriedade FrictionModel é usado para definir se a tração levará em conta a massa do objeto ou não, ScaledBox ou Box respectivamente. O segundo tipo é mais leve ao sistema, mas também bem menos realista.

A propriedade FrictionSlip adiciona efeitos de deslizamento à tração.

FrictionSlide faz com que um dos objetos participantes da colisão atue como uma esteira, carregando o outro corpo em sua superfície.

A propriedade BoxFriction é utilizada apenas quando o FrictionModel é ajustado para Box. Quando este é alterado para ScaledBox deve-se deixar a propriedade BoxFriction com o valor zero e ajustar somente a propriedade ScaledBoxFriction. Essa propriedade regula a força de tração entre os corpos. Os valores vão de zero a um. No caso da plataforma, o valor inicial de tração foi definido como máximo, ou seja, como "1". Durante a execução do ambiente, alguns fatores podem alterar o valor da tração. Isso será explicado durante este capítulo.

4.3.2 MILHARAL

O milharal criado no ambiente foi dividido em linhas horizontais com oito pés de milho cada. Foi criado um total de sento e sessenta pés de milho divididos em vinte

linhas. Os pés de milho foram divididos em linhas horizontais para facilitar seu posicionamento dentro do ambiente, figura 56.

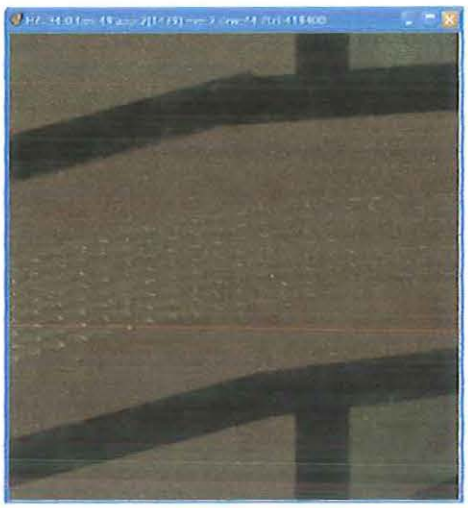

Figura 56 - Milharal visto de cima.

De maneira diferente da implementação anterior em EON Studio 4.0, nessa última fase do projeto, as linhas de plantio foram criadas com certo grau de curvatura e em solo não estruturado. Portanto, a tarefa de posicionar corretamente o milharal foi mais trabalhosa. No entanto, esse trabalho foi facilitado com a ajuda da divisão dos pés de milho em linhas horizontais.

As características principais do milharal nessa fase do projeto são mostradas abaixo:

- Linhas de plantio com curvatura;
- Solo não estruturado;
- Animação criada de forma individual para cada pé de milho;
- Sistema de colisão criado de forma individual para cada pé de milho;
- Câmera especial para monitorar o milharal visto de cima.

Nesse tópico será descrita apenas a forma de animação do milharal. Os próximos tópicos descreverão o restante de suas características.

O milharal foi animado para aumentar o nível de realismo do ambiente. Essa animação foi implementada com o objetivo de simular o movimento dos ventos em um milharal. Como visto na revisão bibliográfica, o nível de realismo de um ambiente virtual não é somente um detalhe, mas sim uma necessidade principal para que a Realidade Virtual alcance seus objetivos.

Apenas um tipo de nó foi utilizado para a implementação da animação: o nó Place. Sua programação não utilizou scripts, sendo totalmente realizada por rotas.

Um nó Place translada ou rotaciona um objeto com relação à sua posição no ambiente ou sua posição no próprio frame. Nas propriedades dos nós Place é possível estipular um tempo para que o movimento complete sua operação, figura 57.

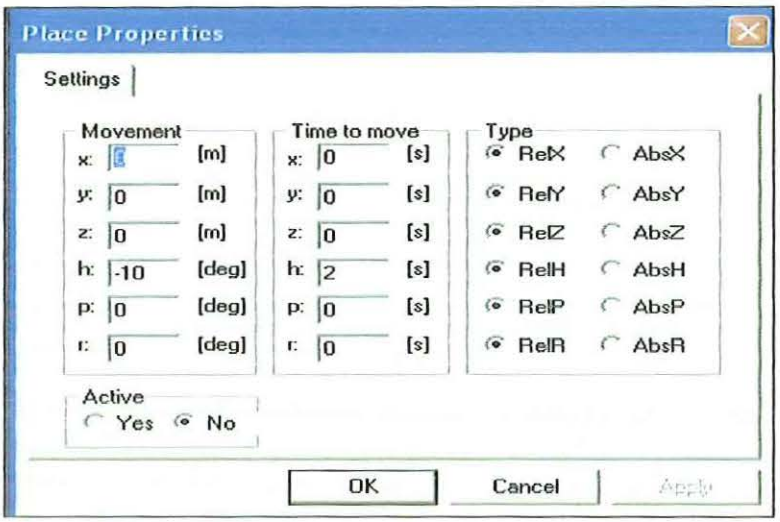

Figura 57 - Propriedades de um nó Place.

A figura acima mostra as propriedades de um nó Place do ambiente. A função Movement permite definir a quantidade de metros que o Frame será transladado ou a quantidade de graus que o Frame será rotacionado. A função Time To Move permite definir um tempo, em segundos, para que o Frame complete a operação Movement. A opção Type deve ser utilizada para definir se o movimento será executado de acordo com a posição global do Frame ou de acordo com sua posição relativa ao próprio Frame. A opção Active é usada para ativar o movimento no início da simulação.

Foram utilizados dois nós Place para cada pé de milho do ambiente. Esses foram nomeados de Place e Place1.

Apenas o movimento de rotação foi criado, os valores da translação foram deixados em zero.

A lógica criada foi a seguinte: primeiramente o nó nomeado Place aplicaria um movimento de rotação de alguns graus em seu respectivo pé de milho. Assim que esse movimento fosse concluído, o nó Place1 seria ativado, aplicando o movimento de rotação inverso ao aplicado pelo nó Place. Dessa forma o pé de milho retornaria à sua posição original.

Foi estipulado um tempo, em média de dois segundos, para a execução do movimento de cada nó Place.

A animação dos pés de milhos é um movimento constante, executado durante todo o tempo de simulação do ambiente. Isso se deve ao fato de que o fim da execução do nó Place ativa a execução do nó Place1 e vice-versa.

Abaixo é mostrado um exemplo, na figura 58, para um dos pés de milho do ambiente.

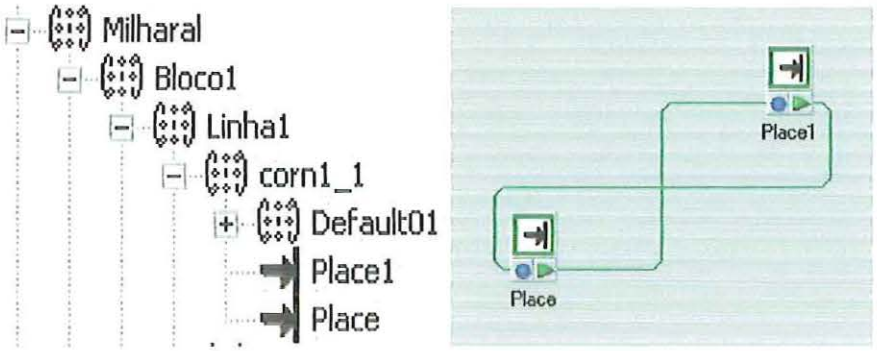

Figura 58 - Place e Place1 para um pé de milho.

A rota de simulação da figura acima possui as seguintes variáveis de entrada e saída:

Place_OnRunFalse → Place1_SetRun
Place1_OnRunFalse → Place_SetRun

Os valores da rotação e do tempo para a execução do movimento foram apenas semelhantes para cada pé de milho, mas não idênticos. Isso auxiliou para aumentar o grau de realismo do ambiente.

O projeto possui um total de cento e sessenta pés de milho. Apesar das geometrias usadas para os pés de milho serem geometrias simples, essa grande quantidade de pés de milho pode acarretar alguns problemas ao ambiente. Além da perda de velocidade na execução, foram encontrados alguns problemas no processo de criação do ambiente.

Como existem cento e sessenta pés de milho e dois nós Places de animação para cada um, existe um total de trezentos e vinte nós Places.

Foi verificado que o EON Studio não opera de maneira satisfatória quando utilizados tantos nós Places num mesmo ambiente. Portanto, para a implementação final

da animação dos pés de milho, foram utilizados apenas 144 nós Places, com a capacidade de animar apenas 72 pés de milho. O restante dos pés de milho foram deixados sem animação. Os pés de milho animados foram distribuídos de maneira aleatória por toda a plantação, fato que proporcionou uma sensação de realismo interessante e menos custosa ao processamento.

Depois de criada uma rota de animação entre os nós Place e Place1, não é necessário repetir a operação para cada pé de milho. Copiando e colando os nós Place e Place1 para cada pé de milho animado, as rotas de programação são automaticamente criadas na janela de rotas.

Conforme os pés de milho eram animados, a janela de rotas tornava-se cada vez mais pesada. Isso chegou até mesmo a travar o sistema e foi o fato decisivo para a diminuição da quantidade de pés de milho animados. A janela de rotas chegou a assumir um formato de difícil compreensão.

Mesmo depois da diminuição da quantidade de pés de milho animados, as rotas da animação não se tornaram totalmente legíveis, mas o funcionamento do software voltou ao seu estado normal.

4.3.3 COLISÃO COM O MILHO

Neste tópico será descrito o método utilizado para a colisão da plataforma com o milharal. Este método difere da colisão do solo com as rodas, pois são utilizados outros tipos de nós.

A Realidade Virtual tem como objetivo simular o mundo real da forma mais precisa possível. Portanto, os objetos componentes de um Ambiente Virtual, geralmente, são ricos em detalhes e pesados computacionalmente. Devido a essa grande necessidade de poder de processamento, existem alguns métodos de simplificação da colisão de objetos. É recomendado, por exemplo, não utilizar a geometria visual de um objeto, normalmente muito detalhada (rica em polígonos), para a implementação da colisão.

4.3.2.1 Geometrias de Colisão

As figuras usadas para a representação do milho e da plataforma não possuem excesso de detalhes, pois o objetivo principal da prototipação virtual desse trabalho não

incluía buscar uma representação gráfica perfeita, mas realizar testes físicos sobre a plataforma virtual.

Embora os objetos do ambiente não fossem ricos em detalhes, a colisão da plataforma com o milharal, quando realizada com as próprias geometrias visuais dos objetos, deixou o sistema lento. Portanto, foram criadas novas geometrias para a implementação da colisão. Para cada pé de milho foi criado um retângulo com espessura proporcional ao caule. Para a plataforma agrícola foram criados dois retângulos, um para cada lateral do veículo, figura 59.

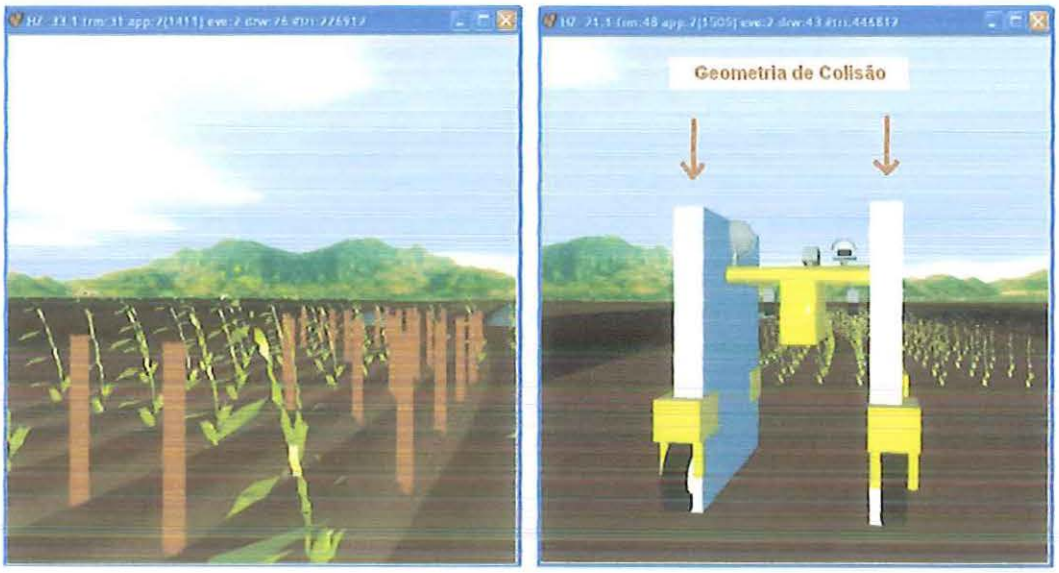

Figura 59 - Geometria de Colisão dos pés de milho e da plataforma.

A geometria de colisão do milho foi criada dessa maneira (um retângulo com pequena espessura) para que o contato com as folhas não acusasse colisão. Pois, nesse projeto, foi definido que o efeito da colisão da plataforma com um pé de milho seria a eliminação de sua geometria visual da cena. A geometria de colisão da plataforma foi criada dessa maneira para que dois pés de milho pudessem passar por entre as laterais da plataforma sem acusar colisão.

É possível perceber facilmente que essas novas geometrias criadas são muito menos custosas ao ambiente do que as geometrias visuais dos objetos, pois essas são compostas por uma quantidade menor de polígonos.

Para facilitar o entendimento em ambientes de grande porte, essas geometrias de colisão são colocadas embaixo de um frame, geralmente, nomeado de CollisionGeometry, ou algum outro nome adequado. A propriedade Hidden (escondido)

desses frames deve ser selecionada, para que as geometrias de colisão não sejam mostradas na cena.

4.3.2.2 Nós *Collision*

Depois de criadas as geometrias de colisão para o milharal e para a plataforma, foram adicionados 20 nós CollisionObject à árvore de simulação, um para cada pé de milho. Outro nó Collision foi adicionado embaixo do frame Col_Geo referente à plataforma. Portanto, nesse momento, todos os frames de cada pé de milho e o frame Plataforma, possuíam um frame filho (normalmente chamado de Col_Geo) que continha a geometria de colisão e um nó Collision.

No EON Professional o nó CollisionManager é criado de forma automática embaixo do Frame Managers. Esse nó gerencia o estado de todos os nós CollisionObject da árvore de simulação.

De todas as propriedades dos nós CollisionObject, a mais importante para o projeto foi a propriedade CollisionGeometryType. Ela define como os objetos da geometria de colisão serão tratados. Suas opções são: None; Bounding Spheres; Bounding Sphere; Bounding Boxes; Bounding Box; Convex Hull; Convex Hulls; Geometry as is.

Tomando como exemplo a roda de um veículo, nesse caso deve ser utilizada a opção Bounding Sphere para a geometria de colisão, pois a roda de um carro é semelhante a uma esfera. Se nesse caso fosse utilizada a opção Bounding Box, a roda do carro seria envolvida por uma caixa, invisível ao usuário e, o contato da roda do veículo com o solo, não aconteceria de forma adequada.

A opção "Geometry as is" é mais pesada ao sistema, mas envolve de forma correta qualquer tipo de objeto. Essa opção deve ser usada para objetos complexos, como um terreno não-estruturado, por exemplo.

Nesse momento do projeto, a colisão da plataforma com o milho já era acusada pelo sistema. Porém, as propriedades dos nós CollisionObject tiveram de ser modificadas, para que a colisão entre os dois objetos não afetasse o movimento da plataforma. Afinal, uma colisão com um pé de milho é diferente da colisão com uma parede de concreto.

A propriedade CollisionResponse do nó CollisionObject localizado embaixo do nó Plataforma foi desativada. Dessa forma a plataforma apenas acusava a colisão com os pés de milho, porém, isso não afetava qualquer propriedade física da mesma.

A colisão é acusada pela propriedade Collision de cada nó CollisionObject. Essa propriedade foi utilizada para a programação dos efeitos de colisão da plataforma com o milharal. Algumas rotas de simulação, juntamente com alguns scripts, foram criados para que a geometria visual do pé de milho sumisse da cena quando colidida com a plataforma, figura 60.

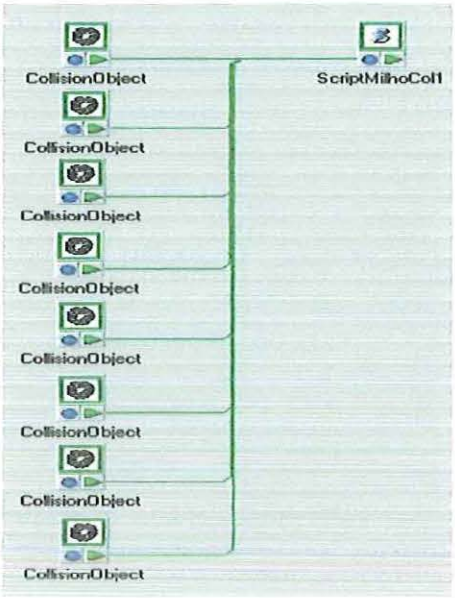

Figura 60 - Rotas de Colisão com o milharal.

Como o número de nós Collision era muito grande (mesmo número de pés de milho, ou seja cento e sessenta), a implementação dos efeitos da colisão foi dividida pelas linhas horizontais de pés de milho (oito pés), para simplificação do seu entendimento. Dessa forma apenas um script foi utilizado para controlar cada colisão de oito pés de milho.

A variável de saída de todos os nós Collision é a propriedade Collision.

As variáveis de entrada criadas nos scripts foram: In1, In2, In3, In4, In5, In6, In7, In8.

A figura acima mostra apenas uma das vinte rotas de Colisão criadas para o milharal.

A estrutura do script criado para as vinte rotas de colisão foi semelhante à mostrada no Apêndice B.

4.3.2.3 Som de Colisão

Antes da criação das animações do milharal, de seu terreno não-estruturado e de suas propriedades de colisão, o milharal possuía apenas uma detecção de colisão simples, que disparava um arquivo WAVE (som) para proporcionar uma colisão mais realista.

Nessa primeira implementação, o objeto de colisão da plataforma era um cubo de dimensões proporcionais às da plataforma. Os pés de milho não possuíam geometrias de colisão individuais. Foi criado um retângulo de dimensões proporcionais à dimensão total do milharal.

Não existia script para controlar a colisão, apenas uma rota simples de simulação, figura 61.

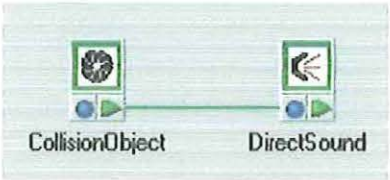

Figura 61 - Rotas de simulação do som de colisão com o milharal.

A variável de saída do nó Collision dessa rota era a propriedade Collision. A variável de entrada do objeto DirectSound era SetRun.

Nos próximos tópicos serão comentados mais alguns detalhes sobre milharal e suas propriedades de colisão, como por exemplo, a implementação da visão aérea do milharal, de seu terreno não estruturado e da visualização dos milhos colididos.

4.3.4 TERRENOS

Os terrenos do ambiente virtual criado podem ser divididos em dois tipos: os regulares e os não-estruturados.

Os terrenos regulares são aqueles que não possuem desníveis. Estes geralmente não causam grandes problemas aos desenvolvedores de ambientes virtuais. Todos os terrenos regulares do ambiente são, simplesmente, retângulos criados com recursos do próprio EON, onde são aplicadas texturas.

Os terrenos não-estruturados são aqueles compostos por morros, buracos, rampas e desníveis de qualquer tipo. Não possuem qualquer restrição de forma.

Portanto, esse tipo de terreno sempre acarreta em alguns problemas para o desenvolvedor.

O EON Professional possui diversas funcionalidades que minimizam tais problemas, como por exemplo, controle de tração, deslizamento, gravidade etc. Uma outra propriedade interessante do software, que foi muito utilizada para os terrenos criados no ambiente, foi a propriedade Frozen dos nós RigidBody. Quando selecionada, ela anula a gravidade para seu corpo rígido. Dessa forma os terrenos criados não eram afetados pela gravidade e nem pelas colisões sofridas.

Existem dois terrenos não-estruturados no ambiente criado. Um deles é composto por diversos tipos de morros e foi usado para testes iniciais dos movimentos da plataforma. Esse terreno foi importado do software 3DStudio Max (extensão .3DS) , figura 62.

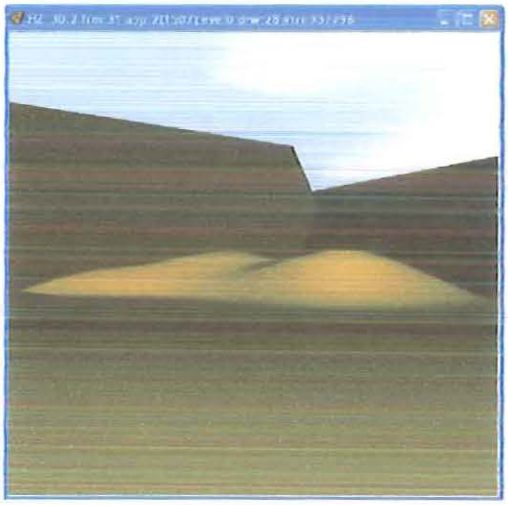

Figura 62 - Terreno não-estruturado.

O outro terreno não-estruturado do ambiente é o solo do milharal. Esse terreno também foi criado em 3DStudio Max e importado para o EON Studio. É formado por diversas concavidades referentes às linhas de plantio, figura 63.

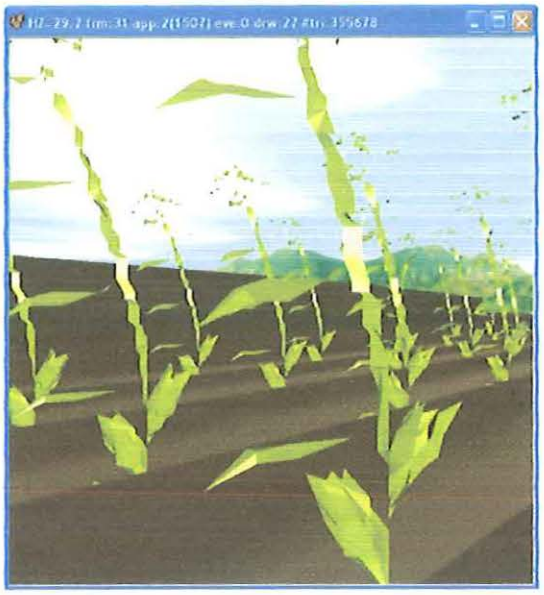

Figura 63 - Terreno não estruturado das linhas de plantio.

A figura 64 abaixo mostra todos os componentes do Frame Terrenos:

Figura 64 - Componentes do Frame Terrenos.

Além do Frame Terrenos existe também na árvore de simulação um Frame chamado Muretas que possui outros objetos relacionados aos terrenos do ambiente. Estes não serão explicados devido à sua simplicidade.

Os terrenos não-estruturados foram importantes para o ambiente virtual da plataforma agrícola, para que pudessem ser realizados testes referentes ao centro de massa da plataforma, entre outras coisas. Um exemplo disso foi a verificação da inclinação máxima suportada pela plataforma, sem que a mesma tombasse. Para isso foram criadas no ambiente algumas rampas com graus pré-definidos:

- 40° - a plataforma se movimentou normalmente;
- 50° - a plataforma ainda conseguia se movimentar sem tombar;

- 60° - a plataforma tomba com certa facilidade.

Essa verificação de inclinações da plataforma incluiu a descida de degraus e a capacidade de ultrapassar pedras e buracos no ambiente, figura 65.

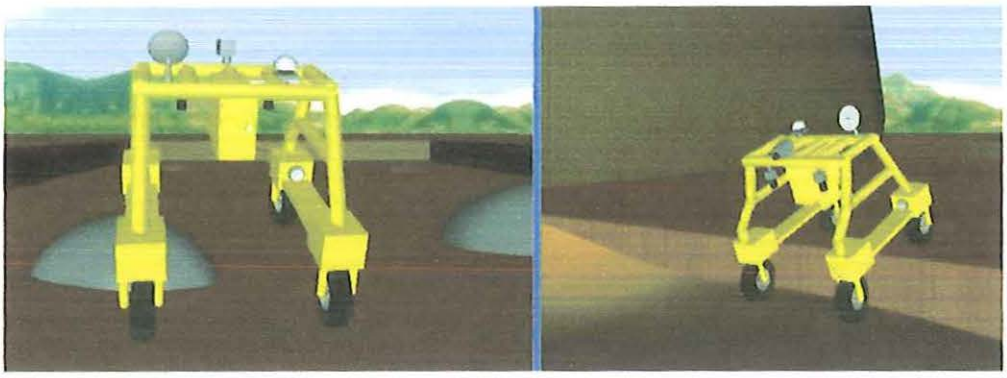

Figura 65 - Obstáculos do ambiente: degraus e pedras.

Foi criado um terreno especial para esse tipo de teste, composto por diversas pedras e buracos. Esse terreno também foi criado em 3DStudio Max.

4.3.5 VIEWPORTS E CÂMERAS

Um ambiente virtual em EON Studio pode conter diversos pontos de vista. Qualquer ambiente iniciado no software possui um ponto de vista básico ou padrão.

Um ponto de vista é criado pela adição de um nó viewport embaixo da pasta Viewports na árvore de simulação. Além disso, é necessário adicionar um atalho de um nó Frame à pasta câmera do novo viewport. Esse nó Frame referenciado deverá conter a posição do novo ponto de vista.

Nas propriedades de cada nó Viewport é possível definir o tamanho da janela onde será mostrado o novo ponto de vista do ambiente. Se este tamanho for definido com os valores "1" na horizontal e "1" na vertical, o ponto de vista será mostrado com o tamanho total da janela de simulação

Os pontos de vista criados no ambiente podem ser vistos na figura 66 abaixo:.

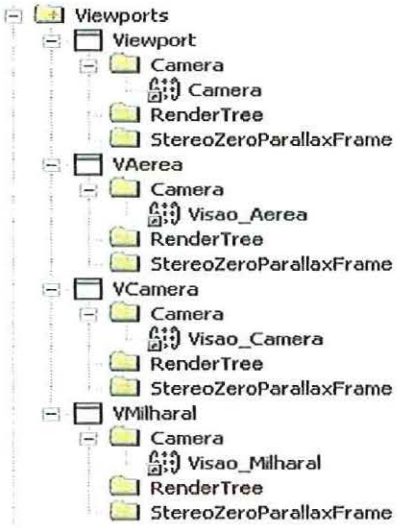

Figura 66 - Viewports do projeto.

Há diversos pontos de vista diferentes criados no ambiente para facilitar a análise e navegação. Todos eles são ativados ou desativados por teclado (teclas 1, 2, 3, 4, 5, 6 e 7). Foram criadas rotas de simulações e scripts para o controle das visões, figura 67.

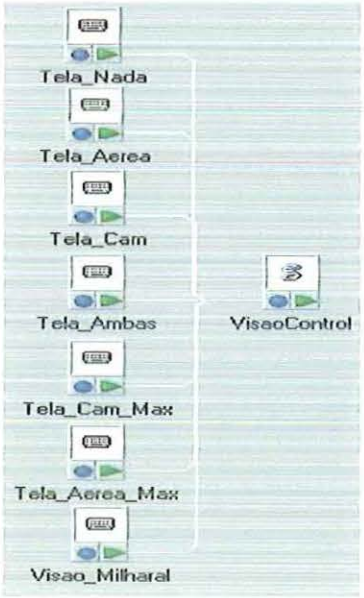

Figura 67 – Rotas referentes aos pontos de vista do ambiente.

Além da câmera normal, reposicionada constantemente com o uso do mouse, os outros pontos de vista disponíveis no ambiente são seis, figura 68:

1) Visão aérea da plataforma
2) Visão aérea da plataforma maximizada

3) Visão da câmera da plataforma
4) Visão da câmera da plataforma maximizada
5) Ambas as visões da câmera da plataforma e aérea
6) Visão aérea do milharal maximizada

As variáveis de saída dos nós KeyboardSensor são On_KeyDown. Foram criadas sete variáveis de entrada para o script controlador dos pontos de vista: InCam, InAerea, InNada, InAmbas, InCamMax, InAereaMax, InVisaoMilharal.

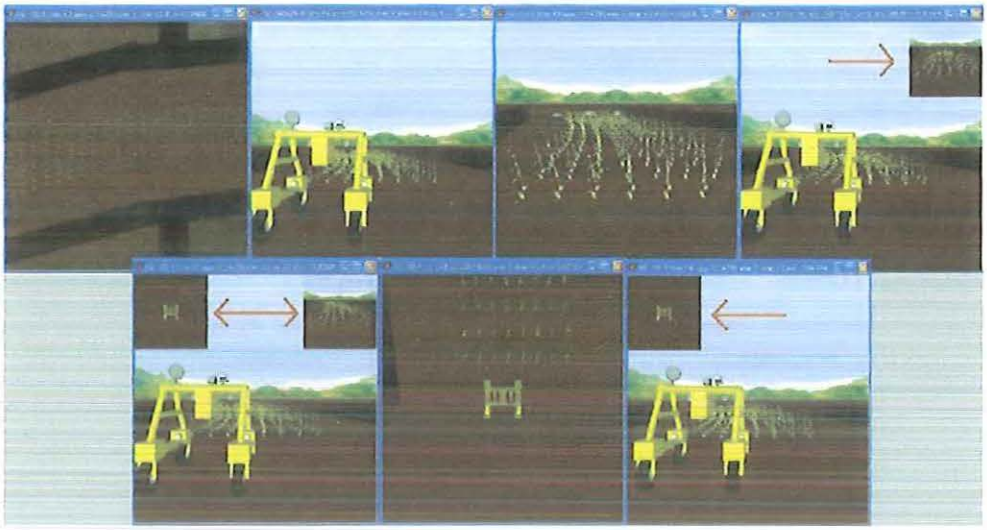

Figura 68 - Todas as possíveis vistas do ambiente.

4.3.4.1 Visão aérea do milharal

A visão aérea do milharal foi criada especialmente para a análise de quais milhos foram colididos com a plataforma. Essa visão engloba todos os pés de milho do milharal e só pode ser ativada em sua forma maximizada. Os outros pontos de vista (câmera da plataforma e aérea) podem ser disponibilizados nos cantos superiores da tela junto à visão padrão.

Há duas possibilidades de visualizar o milharal para a análise dos milhos colididos. Uma delas é ver espaços vazios onde a plataforma colidiu com o milho. A outra é ver, ao invés de espaços vazios, a geometria de colisão de cada pé de milho. Como essa geometria de colisão é um retângulo vermelho, o caminho onde a plataforma colidiu com o milharal se torna mais evidente. A figura 69 abaixo exemplifica os dois tipos de visões:

Figura 69 - Visão do milharal com as geometrias de colisão e sem as mesmas.

A figura 70 abaixo mostra as rotas criadas para mostrar as geometrias de colisão dos milhos colididos. A tecla de liga/desliga das geometrias de colisão do milharal é a tecla "C".

Figura 70 - Rotas criadas para mostrar milhos colididos.

As rotas possuem as seguintes variáveis de entrada e saída:

MostraColisao_OnKeyDown → Latch_Toggle
Latch_OnSet → MilhoCol_SetRun
Latch_OnClear → MilhoCol_SetRun_

O nó Latch funciona como um interruptor acionado pelas suas variáveis Ser e Clear. As variáveis SetRun e SetRun_ de um Frame se referem à propriedade Hidden, isto é, SetRun mostra o conteúdo do Frame e SetRun_ remove da cena os seus componentes.

Essa funcionalidade do ambiente torna mais simples a verificação da colisão da plataforma com o milharal. Ela pode ser bastante útil quando o ambiente virtual possuir um milharal mais extenso e semelhante ao real.

4.3.6 CÂMERA

O Frame Câmera além de ser responsável pelo ponto de vista padrão da cena, possui alguns componentes especiais, figura 71.

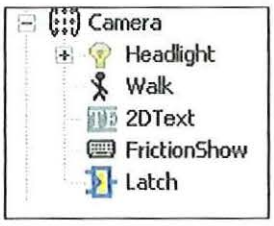

Figura 71 - Componentes do Nó Câmera.

O componente HeadLight aplica uma luz direcional nos objetos visualizados por esse ponto de vista. O componente Walk permite que o usuário movimente o ponto de visão com o mouse. Esses dois primeiros nós são adicionados automaticamente à árvore de simulação pelo EON Studio.

Os outros três componentes estão relacionados com a disponibilização do valor da propriedade Friction na tela.

Essa função foi implementada, pois o valor da propriedade Friction muda de acordo com o solo do ambiente. Por exemplo, na água, o valor da tração cai de "1" para "0.7".

Como dito anteriormente, o nó Latch atua como um interruptor, portanto, da forma como foi implementada essa rota de simulação, a tecla "F" referente ao nó keyboardSensor ativará ou desativará o texto do nó 2DText. Esse texto contém o valor das propriedades Friction e Slide. Quando as rodas da plataforma entram em contato com a água, os valores dessas duas propriedades são modificados e, conseqüentemente, o valor do 2DText também é modificado. O valor inicial desse nó é "Friction 1 Slide 0/0" , figura 72.

Figura 72 - Rota de simulação do 2DText.

As propriedades no nó 2DText são, figura 73:

- Text → "Friction = 1 Slide = 0/0": A propriedade Text define o texto a ser mostrado na tela;
- TextPosition → 0,0: Define em que posição da caixa o texto começará a ser mostrado;
- BoxPosition → 10,10: define a posição da caixa de texto;
- BoxArea → 220, 20: define o tamanho da caixa de texto;
- TextAlignment → Left: define o alinhamento do texto.

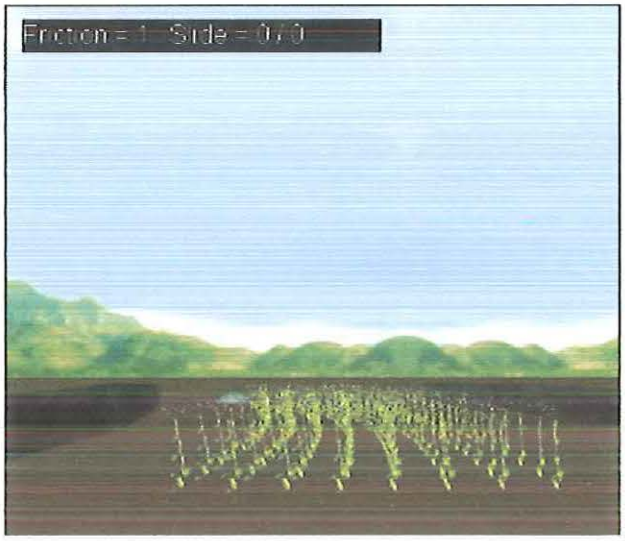

Figura 73 - Valor da propriedade Friction mostrado na tela.

Quando as rodas da plataforma entram em contato com a água, ou seja, quando o objeto CollisionObject referente à água acusa colisão, um script muda os valores das propriedades Friction e Slide do nó DynamicsManger, assim como o texto do nó 2DText. O script é mostrado no Apêndice C.

Para esse script foram criadas suas variáveis de entrada e uma variável de saída. As variáveis de entrada são: InTimer e InColidiu. A variável de saída é: OutTimer. No momento da colisão das rodas com a água, a variável InColidiu é ativada, modificando os valores de Friction, Slide e 2DText. Além disso, um nó TimeSensor é disparado. Esse nó conta com as seguintes propriedades, figura 74:

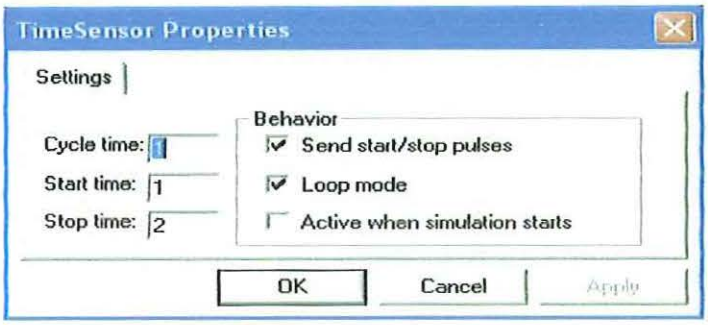

Figura 74 - Propriedades do nó TimeSensor.

Dentro do script, quando o nó TimeSensor dispara a variável InTimer (tempo de ciclo de um segundo), o respectivo procedimento verifica se o nó CollisionObject referente à água ainda está acusando colisão. Em caso negativo, esse procedimento volta os valores de Friction, Slide e 2DText para os valores originais e ativa a variável de saída OutTimer, que desativa o nó TimeSensor (contador). Em caso positivo, o nó TimeSensor não será desativado e continuará a acionar a variável InTimer, até que as rodas da plataforma parem de colidir com a água.

O próximo tópico detalha o método de criação do objeto água, suas texturas e animações.

4.3.7 ÁGUA

Uma nova funcionalidade do EON Professional é a capacidade de simular ambientes que possuam água de uma maneira mais realista.

Para a criação do objeto água foram utilizados alguns nós e texturas especiais. Os nós utilizados foram o CgMaterial (para mais detalhes ver revisão bibliográfica do EON Professional), dois nós Texture2 com arquivos de texturas especiais (extensão .DDS), aplicados num nó Shape. Nesse nó Shape também foi criada a geometria de um plano, onde foram aplicadas as texturas da água. A figura 75 abaixo mostra a árvore de simulação para o Frame Água:

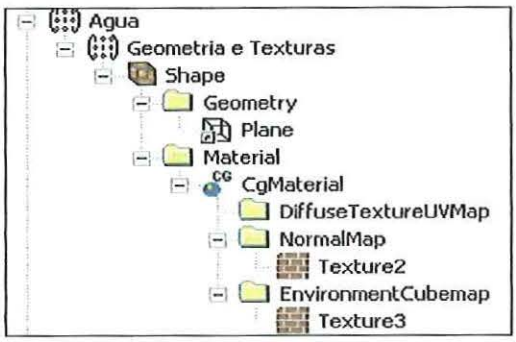

Figura 75 - Componentes do objeto Água.

As texturas escolhidas para a água foram os arquivos "waves_normalMap.DDS" para Texture2 e "sky_envmap_256_a8.DDS" para o nó Texture3. Para maiores detalhes sobre os arquivos DDS ver o capítulo de Revisão Bibliográfica.

Para adicionar efeitos de movimentação à água foi utilizada uma rota de simulação, figura 76. Os componentes dessa rota foram um nó TimeSensor, um nó Multiplication e o nó CgMaterial referente ao material da água.

Figura 76 - Rotas referentes à água.

As variáveis de entrada e saída dessas rotas foram ligadas da seguinte maneira:

TimerSensor_CurrentSimTime → Multiplication_InValue1
Multiplication_OutValue → CgMaterial_Time

A variável de saída CurrentSimTime do nó TimeSensor envia constantemente o tempo decorrido, desde o início da simulação, para o nó Multiplication. Esse nó armazena o valor de CurrentSimTime na variável InValue1. A variável InValue2 tem o valor padrão de "1.0". Essa variável funciona como uma constante de multiplicação para o valor de CurrentSim Time. Nesse projeto, essa variável pode ser alterada somente no ambiente de criação, mas não em tempo real. Os valores InValue1 e InValue2 são automaticamente multiplicados (devido à propriedade AutoTrigger do nó Multiplication) e o resultado é enviado para o nó CgMaterial pela variável de saída

OutValue. Esse valor é responsável pelo efeito de movimento das águas do ambiente. Esse movimento é aleatório devido à constante mudança de valor de CurrentSimTime.

As propriedades do nó CgMaterial são, figura 77:

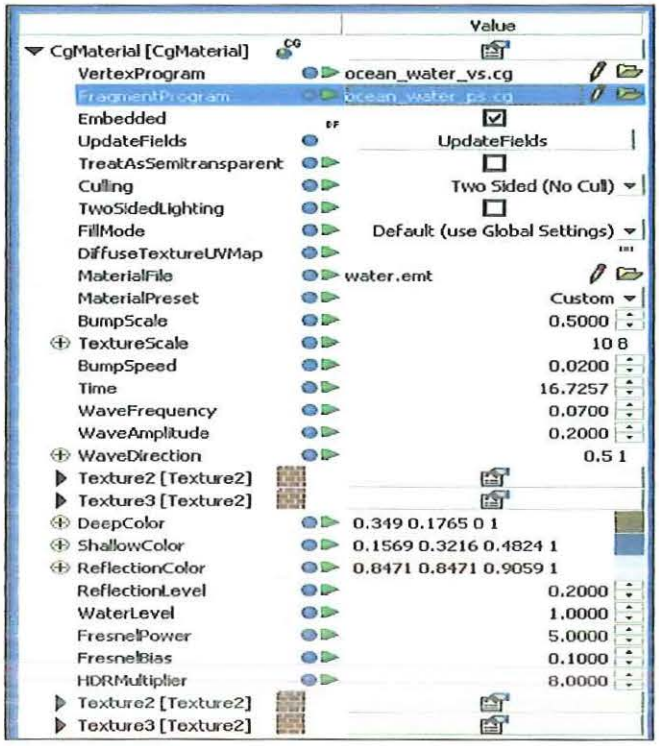

Figura 77 - Propriedades do nó CgMaterial.

Todos os arquivos DDS, CG e EMT usados para a criação da água são parte do pacote EON Professional e são detalhados na Revisão Bibliográfica.

Nesse projeto o efeito visual das águas não alcançou um nível de realismo satisfatório, pois tal tarefa é complexa, consome muito tempo e foge do foco do projeto. A figura 78 abaixo mostra o aspecto da água no projeto:

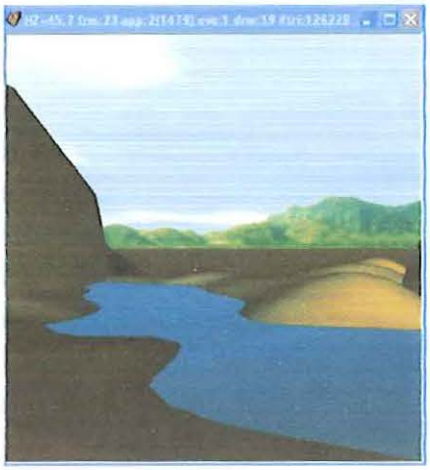

Figura 78 - Água.

Como dito no tópico anterior, mais do que aumentar a sensação de realidade dentro do ambiente, a água possui uma funcionalidade especial. Quando as rodas da plataforma entram em contato com a geometria da água, a variável de tração do nó DynamicsManager é decrementada de "1.0" para "0.7" e a propriedade Slide (derrapagem) é aumentada de "0/0" para "0.5/0.5" (eixos X e Y) . Isso dificulta o controle da plataforma tentando simular um movimento real de derrapagem.

4.3.8 OUTROS NÓS

4.3.7.1 Managers

Os nós Managers são criados para controlar as interações da simulação. Na versão Professional do EON os nós managers são criados automaticamente na árvore de simulação.

Os nós managers criados no ambiente do projeto são mostrados na figura 79 abaixo:

Figura 79 - Managers do ambiente.

O nó DynamicsManager é criado no momento em que algum nó RigidBody é adicionado à árvore de simulação.

O nó JointManager é criado no momento em que qualquer um dos tipos de nós Joints é adicionado à árvore.

O nó CollisionManager é adicionado ao ambiente no momento em que algum nó Collision é colocado na árvore de simulação.

Quando uma simulação necessita de que objetos sejam controlados por diferentes gerenciadores, é possível adicionar novos nós de controle à árvore de simulação. Dessa forma, objetos podem ser controlados por gerenciadores com características diferentes.

As propriedades mais importantes do nó DynamicsManager são:

- Gravity: seu valor padrão é "0,0,-9.81" (o valor da gravidade na Terra);
- ContactDefinitions: um arquivo texto ECD, que mostra as propriedades de contato dos objetos. O arquivo padrão "DefaultContactDefinitions" possui a definição dos contatos de dois objetos padrões (Default/Default), um objeto padrão e um macio (Default/Soft), ou dois objetos macios (Soft/Soft);
- ShowContactDefinitions: nesta propriedade deve ser escolhida uma das definições de contato do item anterior;
- CP_Restitution: define o grau de elasticidade da colisão;
- CP_Softness: Define a penetração na colisão de dois objetos;
- CP_AdhesiveForce: Funciona como uma "cola" entre dois objetos;
- CP_FrictionType: Define se a tração será executada em um ou dois eixos;
- CP_FrictionModel: Define se a tração levará em conta a massa do objeto;
- CP_FrictionSlip: Define o nível de deslizamento;
- CP_FrictionSlide: Funciona como uma esteira, onde um objeto carrega o outro em sua superfície;
- CP_BoxFriction: Usado juntamente com a propriedade Box de CP_FrictionModel;
- CP_ScaledBoxFriction: usado juntamente com a propriedade ScaledBox de CP_FrictionModel.

As outras propriedades do nó DynamicsManager não foram citadas, pois essas não foram estudadas, nem necessárias nesse projeto.

A função CP_FrictionSlide, descrita acima, funcionou no projeto como um fator de deslizamento para a plataforma.

Os nós JointManager e CollisionManager não possuem propriedades possíveis de serem alteradas diretamente. Essas propriedades são alteradas indiretamente pelos nós Joint (de qualquer tipo) e Collision, respectivamente.

Os nós componentes do ambiente que não foram citados nessa parte do trabalho (Desenvolvimento da plataforma virtual agrícola móvel em EON Professional) foram explicados anteriormente no tópico referente ao desenvolvimento da plataforma no EON Studio 4.0.

Todas as rotas de simulação utilizadas no projeto, com exceção das rotas de colisão e animação do milharal (muito extensas), são mostradas na figura 80 abaixo:

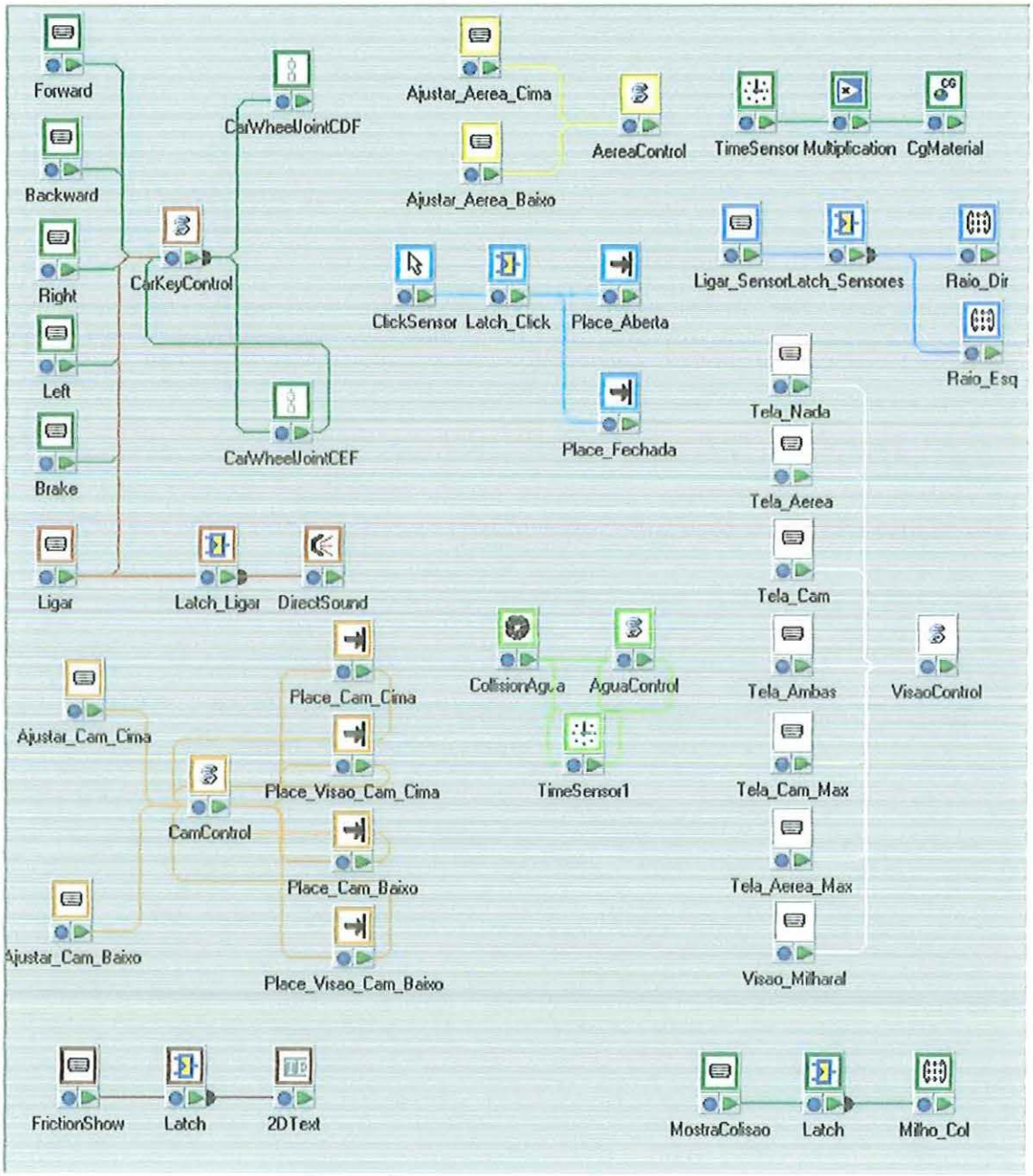

Figura 80 - Rotas finais do projeto.

4.4 Resultados Obtidos

Embora neste trabalho se tenha utilizado apenas dados relativos ao anteprojeto do VAA, alguns testes puderam ser executados e algumas conclusões puderam ser obtidas quanto ao comportamento da plataforma virtual.

Uma das características da plataforma testada neste trabalho foi a capacidade de re-configuração do posicionamento de suas laterais (para adaptação à outros tipos de culturas). Para isso, foi criado outro ambiente virtual, idêntico ao original, porém com as laterais da plataforma re-posicionadas, como mostra a figura 81 abaixo:

Figura 81 - Plataforma com as laterais reposicionadas.

Controlando a nova plataforma pelos terrenos irregulares do ambiente, puderam-se avaliar as dificuldades encontradas quanto à estabilidade da plataforma. De maneira diferente da plataforma original, quando movimentada sobre as rampas criadas no sistema, a nova plataforma logo perdia a estabilidade e tombava. Esses efeitos causados pela re-configuração das laterais da plataforma devem-se aos novos valores do centro de massa do veículo.

Ainda para testes de estabilidade, três rampas foram criadas no sistema. Os valores de seus ângulos foram: 40, 50 e 60 graus. A velocidade de movimentação da plataforma sobre as rampas criadas não respeitou valores exatos, pois nessa fase do projeto original, esses valores ainda não estavam disponíveis. Quando a plataforma original (sem as laterais re-posicionadas) foi movimentada sobre esses ângulos de inclinação, os resultados foram os seguintes:

- 40 graus: a plataforma nunca tombava;
- 50 graus: a plataforma tombava dependendo da velocidade de entrada;

- 60 graus: a plataforma tombava com alguma facilidade, mesmo em baixa velocidade.

Como mostrado no capítulo anterior, quando as rodas da plataforma entravam em contato com a geometria da água, a variável de tração do nó Dynamics Manager era decrementada de "1.0" para "0.7" e a propriedade Slide (derrapagem) é aumentada de "0/0" para "0.5/0.5" (eixos X e Y).

Essas alterações de valores foram implementadas para ter-se uma idéia do movimento e estabilidade da plataforma em terrenos comumente encontrados em ambientes agrícolas, como lama, por exemplo. Os valores utilizados aqui requerem algum estudo mais aprofundado para que esses possam ser utilizados como uma predição dos movimentos reais da plataforma agrícola móvel.

Com o ambiente virtual criado, puderam ser verificados, também, quais obstáculos (como pedras, buracos e degraus) que inviabilizam a passagem da plataforma, derrubando-a ou não permitindo seu progresso.

Uma limitação encontrada durante o processo de desenvolvimento do ambiente virtual foi a utilização de sombras. Apesar de aumentar muito a sensação de realidade do ambiente, a técnica de utilização de sombras tornava o sistema extremamente lento, dificultando o controle da plataforma.

Durante o desenvolvimento do projeto, foram encontradas algumas deficiências (*bugs*) do *software*. Quando utilizado por um longo período, o ambiente run-time do software EON Professional (assim como o do EON Studio 4.0) se tornava inconstante, principalmente em colisões de corpos rígidos, forçando a sua reinicialização para normalizar seu comportamento.

CAPÍTULO V -CONCLUSÃO

Ma (2004) mostra em seu trabalho que um dos fatores que limita a Prototipação Virtual é o fato de modelos CAD (Computer-Aided Design - objetos geométricos 3D criados em ferramentas especiais) terem de ser sempre criados com dimensões exatas, no entanto, essas dimensões não são especificadas com precisão no nível conceitual do projeto.

O fator observado por Ma (2004) foi também verificado neste trabalho, pois os dados referentes à plataforma virtual agrícola móvel implementada tomaram como base os dados do projeto com auxílio pesquisa FAPESP processo: 2003/06582-0 "Veículo Agrícola Autônomo (VAA): Uma plataforma para desenvolvimento de tecnologias de navegação autônoma e para aquisição de dados em agricultura de precisão". E, durante o desenvolvimento deste trabalho, o projeto VAA ainda não havia sido concluído, portanto os dados utilizados para a prototipação virtual da plataforma agrícola móvel não possuem precisão confiável e os resultados obtidos não puderam ter sua veracidade testada.

Embora o software EON Professional tenha se mostrado extremamente poderoso para tarefas de PV, dois fatores podem inviabilizar seu uso. O principal deles é o alto custo da ferramenta. A versão para faculdades custa em torno de trinta mil reais. O outro fator pôde ser verificado durante o desenvolvimento do projeto e será descrito a seguir. Um dos maiores desafios da RV é: realismo X visualização em tempo real. Algumas operações, como, por exemplo, a aplicação de sombras, aumentam consideravelmente o número de objetos da cena, tornando o sistema extremamente lento. Apesar do *software* EON Professional possuir alguns recursos para diminuição de polígonos, estes não foram suficientes para proporcionar o uso de imagens de alta resolução, mesmo num computador de alta capacidade: XEON de 3.06 GHz com 3,5 GB de RAM, placa de vídeo NVIDIA Quadro FX 1000. Portanto, para que fosse possível utilizar todos os recursos visuais do *software* EON Professional seria

necessário uma máquina com capacidade de processamento ainda maior, o que poderia inviabilizar um projeto, devido ao seu alto custo.

5.1 Sugestões Para Trabalhos Futuros

Neste trabalho foi realizada uma pesquisa das ferramentas de desenvolvimento virtuais mais adequadas ao projeto de uma plataforma virtual agrícola móvel. Durante essa pesquisa, foi verificado que não existe uma ferramenta que seja melhor que outra em todos os aspectos. Por exemplo, o software Webots fornece melhores métodos para testar controladores de robôs do que o software EON Professional, porém, suas capacidades gráficas são limitadas. Dessa forma, uma pesquisa interessante de ser realizada seria o teste de controladores sobre a plataforma virtual, no software Webots.

Como os dados da plataforma virtual foram baseados em um projeto em andamento, seria interessante utilizar dados reais nesse ambiente, depois de concluído o projeto VAA.

O ambiente virtual desenvolvido nesse trabalho foi baseado na cultura de milho. Porém, o projeto real dessa plataforma agrícola não é restrito a esse tipo de cultura. Uma extensão do ambiente virtual criado poderia ser implementada, para diferentes tipos de culturas.

Como o software utilizado possui um custo muito alto e exige uma grande capacidade de processamento de máquina, um trabalho interessante seria a adaptação do modelo criado para outra ferramenta de desenvolvimento virtual, preferencialmente, uma ferramenta *freeware* como, por exemplo, o BLENDER.

REFERÊNCIAS

3DCANVAS Documentation. Amabilis Software. Disponível em: <http://www.3d-canvas.info/3DCanvasDocumentation/tableofcontents.htm>. Acesso em: nov. 2004.

Araujo, R., B., Home-page do LVRNet – Laboratório de Realidade Virtual em Rede. Disponível em: http://www.dc.ufscar.br/~regina/ Acesso em: Ago. 2005.

Belanović, P., et al., Automatic Generation of Virtual Prototypes. Proceedings of the 15th IEEE International Workshop on Rapid Systems Prototyping, 2004.

Bhatt, G., Bringing Virtual Reality for Commercial Web Sites. International Journal of Human-Computer Studies, 60, 1-15, 2004.

Böhm, K., Wirth, H., John, W., Prototype Validation in Virtual Environments – Vision and first Implementation. In: Rix, J., Haas, S., Teixeira, J., Virtual Prototyping – Virtual Environments and the product design process, Londres: Chapman & Hall, p. 313-325, 1994.

Breen, J., Nottrot, R., Stellingwerff, M., Tangible Virtuality – perceptions of computer-aided and physical modeling. Automation in Construction 12, 649-653, 2003.

Bullinger, H.J., Dangelmaier M., Virtual Prototyping and Testing of in-vehicle Interfaces. Ergonomics, vol. 46, Nº 1-3, 41-51, 2003.

Burger, W., Barth, M., Virtual Reality for Enhanced Computer Vision. In: Rix, J., Haas, S., Teixeira, J., Virtual Prototyping – Virtual Environments and the product design process, Londres: Chapman & Hall, 1994. p. 247-257.

Carey R., Bell G., 1997. The Annotated VRML97 Reference Manual. Disponível em: <http://www.cs.vu.nl/~eliens/documents/vrml/reference/BOOK.HTM>. Acesso em: fev. 2005.

Carr, K., England, R., Simulated and Virtual Realities – Elements of Perception, Sowerby Research Centre, British Aerospace Operations Ltd., Bristol e Interface Technology ResearchLtd., Bristol, 1995.

Chedmail, P., Chablat, D., Roy, C.L., A Distributed Approach for Access and Visibility Task With a Manikin and a Robot in a Virtual Reality Environment. IEEE Transactions on Industrial Eletronics, vol. 50, Nº 4, agosto 2003.

Choi, S.H., Chan, A.M.M.,A Layer-Based Virtual Prototyping system for product development cycle. Computers In Industry 51 237-256, 2003.

Choi, S.H., Cheung, H.H., A multi-material Virtual Prototyping system. Computer-Aided Design 37 123-136, 2004.

Cremer, J.F., Vaněček, G., Isaac: Building Simulations for Virtual Environments. In: Rix, J., Haas, S., Teixeira, J., Virtual Prototyping – Virtual Environments and the product design process, Londres: Chapman & Hall, p. 213-227, 1994.

EON Reality, EON Studio, disponível em: www.eonreality.com. Acesso em: jan. 2005.

Fan, X.J., Zhou, J., Zang, G.Q., Multi-physics modeling in virtual prototyping of eletronics packages – combined thermal, thermo-mechanical and vapor pressure modeling. Microeletronics Reliability 44, 1967-1976, 2004.

Ferreti G., Magnani, G., Rocco, P., Virtual Prototyping of Mechatronic Systems. Annual Reviews in Control 28, 193-206, 2004.

Figueiredo, M., Teixeira, J., Solid Modeling as a Framework in Virtual Environments. In: Rix, J., Haas, S., Teixeira, J., Virtual Prototyping – Virtual Environments and the product design process, Londres: Chapman & Hall, 1994. p. 99-112.

Figueroa, P., et al., Efficient Comparison of Platform Alternatives in Interactive Virtual Reality Applications. Int. J. Human-Computer Studies 62, 73-103, 2004.

Gausemeier, J., Sabin, A., Virtual Prototyping – Design and realistic presentation of industrial products. In: Rix, J., Haas, S., Teixeira, J., Virtual Prototyping – Virtual Environments and the product design process, Londres: Chapman & Hall, p. 280-295, 1994.

He, T., Kaufman, A.E., Virtual Input Devices for 3D Systems. IEEE, 1993.

Her, M.G., Hsu, K.S., Lan, T.S., Virtual Reality Application for Direct-Drive Robot With Force Feedback. Int. J. Adv. Manuf. Technol. 21, 66-71, 2003.

Jasnoch, U., Kress, H., Rix, J., Towards a Virtual Prototyping Environment. In: Rix, J., Haas, S., Teixeira, J., Virtual Prototyping – Virtual Environments and the product design process, Londres: Chapman & Hall, p. 173-183, 1994.

Jezernik, A., Hren, G., A Solution to Integrate Computer-Aided Design (CAD) and Virtual Reality (VR) Databases in Design and Manufacturing Processes. Int. J. Adv. Manuf. Technol. 22, 768-774, 2003.

Kerttula, M., Salmela, M., Heikkinen, M., Virtual Reality Prototyping – a framework for the development of electronics and telecommunication products. IEEE, 1997.

Kirner, C., Introdução a Realidade Virtual, 1999. Disponível em: <http://www.realidadevirtual.com.br>. Acesso em: jan. 2003.

Lécuyer, A., et al., Can Haptic Feedback Improve the Perception of Self-Motion in Virtual Reality? Proceedings of the 12th International Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems, 2004.

Li, Q.M., Design Verification and Testing of Power Supply System by Using Virtual Prototyping. IEEE Transactions On Power Eletronics, Vol. 18, Nº 3, Maio 2003.

Lister, P.F., et al., Eletronic Simulation for Virtual Reality: Virtual Prototyping. Proceedings of the Theory and Practice of Computer Graphics, 2004.

Liu, D., et al., Research on the Theory and the Virtual Prototype of 3-DOF Parallel-Link Coordinate-Measuring Machine. IEEE Transactions on instrumentation and measurement, vol. 52, Nº 1, fev. 2003.

Ma, W., et al., A hierarchically structured and constraint-based data model for intuitive and precise solid modeling in a virtual reality environment. Computer-Aided Design 36, 903-928, 2004.

Machado, L.S., Conceitos Básicos da Realidade Virtual. São José dos Campos. 1995.

Michel, O.,Webots: Professional Mobile Robot Simulation,
Journal of Advanced Robotics Systems, 2004, vol. 1, pags. 39—42. Disponível em: <http://www.ars-journal.com/ars/SubscriberArea/Volume1/39-42.pdf>. Acesso em: jan. 2005.

Mujber, T.S., Szecsi, T., Hashmi, M.S.J., Virtual Reality Applications in manufacturing process simulation. Journal of materials processing technology 155-156, 2004.

Okulicz, K., Virtual Reality based approach to manufacturing process planning. Int. J. Prod. Res., vol. 42, Nº 17, 3493-3504, set. 2004.

Ovtcharova, J., Vieira, A.S., Virtual Prototyping Through Feature Processing. In: Rix, J., Haas, S., Teixeira, J., Virtual Prototyping – Virtual Environments and the product design process, Londres: Chapman & Hall, p. 78-90, 1994.

Palma, J.G., Metamodelo para a modelagem e simulação de sistemas a eventos discretos, baseado em redes de petri e realidade virtual. Uma aplicação em sistemas de manufatura. 2001. 133 f. Tese (Doutorado em Engenharia Mecânica). Escola de Engenharia de São Carlos, Universidade de São Paulo – USP.

Papson, S., et al., A Virtual Reality Environment For Multi-Sensor Data Integration. Sensors for Industry Conference, jan. 2004.

Porto, A.J.V., Palma, J.G., Manufatura Virtual, Revista Fábrica do Futuro, cap. 10, 2000.

Pratt, M.J., Virtual Prototypes and product models in mechanical engineering. In: Rix, J., Haas, S., Teixeira, J., Virtual Prototyping – Virtual Environments and the product design process, Londres: Chapman & Hall,.p. 113-128, 1994.

Raneda, A., et al., Utilization of Virtual Prototyping in development of CMM. Fusion Engineering and Design, 69, 183-186, 2003.

Sánchez-Segura, M.I., et al., Virtual Reality systems estimations vs. Traditional systems estimation. The journal of systems and software 72, 187-194, 2003.

Sousa, R.V., et al., Robô agrícola Móvel (RAM): Uma revisão das pesquisas recentes sobre sistemas de navegação autônoma de robôs e veículos agrícolas, IV Congresso Brasileiro da Sociedade Brasileira de Informática Aplicada à Agropecuária e à Agroindústria. Anais. 2003.

Sousa, R.V., et al., Uma plataforma autônoma para aquisição de variabilidade espacial, Congresso Brasileiro de Agricultura de Precisão – Piracicaba, SP – ESALQ/USP, 2004.

Stribersky, A., Moser, F., Rulka, W., Structural Dynamics and ride comfort of a rail vehicle system. Advances In Engineering software 33, 541-552, 2002.

Sutcliffe, A., Gault, B., Heuristic Evaluation of Virtual Reality Applications. Interacting With Computers 16, 831-849, 2004.

Tougaw, D., Will, J., Visualizing the Future of Virtual Reality. Cumputing in Science & Engineering. 2003.

Vale, H.M.C., Proposta de desenvolvimento de um ambiente de criação de sistemas virtuais de trânsito, voltado para estudos da psicologia, Trabalho de conclusão de curso, Universidade Estadual de Londrina – UEL, 2003.

Williams, R.D., Klenke, R.H., Aylor, J.H., Teaching Computer Design using Virtual Prototyping. IEEE Transactions on Education, vol. 46, Nº 2, maio 2003.

Willis, P., Virtual Physics for Virtual Reality. IEEE, Proceedings of the theory and Practice of Computer Graphics, 2004.

Wloka, M.M., Interacting with Virtual Reality. In: Rix, J., Haas, S., Teixeira, J., Virtual Prototyping – Virtual Environments and the product design process, Londres: Chapman & Hall, p. 199-212, 1994.

Wojciechowski, R., et al., Building Virtual and Augmented Reality Museum Exhibitions. Association for computer machinery, 2004.

WORLDUP Programmer's Guide. Sense8. Disponível em: <http://sense8.sierraweb.net/products/tech_pass.htm>. Acesso em: out. 2004.

WORLDUP User's Guide. Sense8. Disponível em: <http://sense8.sierraweb.net/products/tech_pass.htm>. Acesso em: out. 2004.

Yin, Z.P., Ding, H., Xiong, Y.L., A virtual prototyping approach to generation and evaluation of mechanical assembly sequences. Proc. Insto. Mech Engrs Vol. 218 Part. B: J. Engineering Manufacture, 2003.

Zhang, G.Q., The Challenges of Virtual Prototyping and qualification for future microeletronics. Microeletronics Reliability 43, 1777-1783, 2003.

Zorriassatine, F., et al., A Survey of Virtual Prototyping Techniques for mechanical product development. Proc. Insto. Mech Engrs Vol. 217 Part. B: J. Engineering Manufacture, 2002.

APÊNDICES

APÊNDICE A – Script referente ao movimento completo

```
set PlatNode=EON.FindNode("Plataforma")
set OriPlat=PlatNode.GetFieldByName ("Orientation")
set ForceNode=EON.FindNode("Force")
set Force=ForceNode.GetFieldByName ("ForceVector")
set TorqueNode=EON.FindNode("Torque")
set Torque=TorqueNode.GetFieldByName ("TorqueVector")
set TimerSensorForceNode=EON.FindNode("TimeSensorForce")
set TimeSensorForce=TimerSensorForceNode.GetFieldByName ("SetRun")
set TimerSensorTorqueNode=EON.FindNode("TimeSensorTorque")
set TimeSensorTorque=TimerSensorTorqueNode.GetFieldByName ("SetRun")
Dim AceRet 'informa aceleração ou retrocesso
'*-------------------------------------------*
'INICIALIZAÇÃO
'*-------------------------------------------*
Sub Initialize()
        F=Force
        F(0)=0
        F(1)=0
        F(2)=0
        Force.Value = F
        T=Torque
        T(0)=0
        T(1)=0
        T(2)=0
        Torque.Value = T
        TimeSensorForce.Value = FALSE
        TimeSensorTorque.Value = FALSE
        AceRet = 1 'começa em aceleração
end sub

'*-------------------------------------------*
'VIRAR DIREITA
'*-------------------------------------------*
```

```
Sub On_SetaDir
        T=Torque
        if (AceRet = 0) then
                T(2) = 1000
                Torque.Value = T
                On_SetaBaixo()
        else
                T(2) = -1000
                Torque.Value = T
                On_SetaCima()
        end if
        TimeSensorTorque.Value = TRUE
end sub
'*-------------------------------------------*
'VIRAR ESQUERDA
'*-------------------------------------------*
Sub On_SetaEsq
        T=Torque
        if (AceRet = 0) then
                T(2) = -1000
                Torque.Value = T
                On_SetaBaixo()
        else
                T(2) = 1000
                Torque.Value = T
                On_SetaCima()
        end if
        TimeSensorTorque.Value = TRUE
end sub
'*-------------------------------------------*
'ACELERAR
'*-------------------------------------------*
Sub On_SetaCima
        AceRet = 1 'informa que acelerou
        F = Force
        ori = OriPlat.Value
        Orient = ori(0)
```

```
        Y=1000 'constante 90º vel máxima
        X=Y/90 'constante passo
        If ((Orient<90) and (Orient>=0)) then 'Quad1
                F(1) = -(Orient * X)
        elseif (Orient>=90) then 'Quad2
                F(1) = -(2*Y - Orient * X)
        elseif ((Orient>-90) and (Orient<0)) then 'Quad4
                F(1) = -(Orient * X)
        elseif (Orient<=-90) then 'Quad3
                F(1)= (2*Y + Orient * X)
        end if
        if (Orient < 0) then
                Orient = Orient * (-1)
        end if
        F(0) = (Y - Orient * X) 'vel no X
        Force.Value = F
        TimeSensorForce.Value = TRUE
end sub
'*-----------------------------------------*
'RETROCEDER
'*-----------------------------------------*
Sub On_SetaBaixo
        AceRet = 0 'informa que retrocedeu
        F = Force
        ori = OriPlat.Value
        Orient = ori(0)
        Y=1000 'constante 90º vel máxima
        X=Y/90 'constante passo
        If ((Orient<90) and (Orient>=0)) then 'Quad1
                F(1) = (Orient * X)
        elseif (Orient>=90) then 'Quad2
                F(1) = (2*Y - Orient * X)
        elseif ((Orient>-90) and (Orient<0)) then 'Quad4
                F(1) = (Orient * X)
        elseif (Orient<=-90) then 'Quad3
                F(1)= -(2*Y + Orient * X)
        end if
```

```
        if (Orient < 0) then
                Orient = Orient * (-1)
        end if
        F(0) = -(Y - Orient * X) 'vel no X
        Force.Value = F
        TimeSensorForce.Value = TRUE
end sub
'*-------------------------------------------*
'ZERAR NODE FORCE
'*-------------------------------------------*
Sub On_Zerar_Vel
        F=Force
        F(0)=0
        F(1)=0
        Force.Value = F
end sub
'*-------------------------------------------*
'ZERAR NODE TORQUE
'*-------------------------------------------*
Sub On_Zerar_Tor
        T=Torque
        T(2)=0
        Torque.Value = T
end sub
```

APÊNDICE B – Estrutura do script para as vinte rotas de colisão

```
'Declarações
set Corn1_1Node=EON.FindNode("corn1_1")
set Corn1_1=Corn1_1Node.GetFieldByName("Hidden")

as declarações continuam de maneira semelhante à mostrada acima

'*****************************************************

set C151Node=EON.FindNode("c151")
set C151=C151Node.GetFieldByName("Hidden")

as declarações continuam de maneira semelhante à mostrada acima

'-------------------------------------------------

sub On_In1()
Corn1_1.Value = True
c151.Value = False
end sub

a implementação continua de maneira semelhante à mostrada acima
```

APÊNDICE C – Script controlador de derrapagem

```
set DMNode=EON.FindNode("DynamicsManager")
set DMF=DMNode.GetFieldByName("CP_ScaledBoxFriction")
set DMS=DMNode.GetFieldByName("CP_FrictionSlide")
Slide = DMS.Value
set ColAguaNode=EON.FindNode("CollisionAgua")
set ColAgua=ColAguaNode.GetFieldByName("Collision")
set TextNode=EON.FindNode("2DText")
set Text=TextNode.GetFieldByName("Text")
'-----------------------------------------
sub initialize()
        OutTimer = 0
        ColAgua.Value = False
end sub
'-----------------------------------------
sub On_InColidiu()
Slide(0) = 0.5
Slide(1) = 0.5
DMS.Value = Slide
DMF.Value = 0.7
Text.Value = "Friction = 0.7   Slide = 0.5 / 0.5"
end sub
'-----------------------------------------
sub On_InTimer()
if (ColAgua.Value = False) then
        DMF.Value = 1
        Slide(0) = 0
        Slide(1) = 0
        DMS.Value = Slide
  Text.Value = "Friction = 1   Slide = 0 / 0"
        OutTimer = 1
end if
end sub
```

ANEXOS

ANEXO A – Script para ambiente em WEBOTS

```
#include <device/robot.h>
#include <device/differential_wheels.h>
#include <device/distance_sensor.h>
#define SPEED 100
static DeviceTag ir0,ir1;
static void reset(void) {
  ir0 = robot_get_device("ir0");
  ir1 = robot_get_device("ir1");
  /* printf("ir0=%d ir1=%d\n",ir0,ir1); */
  distance_sensor_enable(ir0,64);
  distance_sensor_enable(ir1,64);}
static int run(int ms) {
  short left_speed,right_speed;
  unsigned short ir0_value,ir1_value;
    ir0_value = distance_sensor_get_value(ir0);
  ir1_value = distance_sensor_get_value(ir1);
  if (ir1_value>200) {
    left_speed  = -20;
    right_speed =  20;
  } else if (ir0_value>200) {
    left_speed  =  20;     right_speed = -20;
  } else {
    left_speed =SPEED;  right_speed=SPEED;  }
  /* Set the motor speeds */
  differential_wheels_set_speed(left_speed,right_speed);
  return 64; /* next step in 64 milliseconds */}
int main() {
  robot_live(reset);
  robot_run(run); /* this function never returns */
  return 0;}
```