UNIVERSIDADE DE SÃO PAULO
Escola Politécnica

# PROJETO E IMPLEMENTAÇÃO DE UM TORNO CNC EM AMBIENTE DE REALIDADE VIRTUAL

Caio Augusto Pereira Rolim
Raphael Mendes de Assumpção

São Paulo
2013

CAIO AUGUSTO PEREIRA ROLIM
RAPHAEL MENDES DE ASSUMPÇÃO

# PROJETO E IMPLEMENTAÇÃO DE UM TORNO CNC EM AMBIENTE DE REALIDADE VIRTUAL

Monografia apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Engenheiro Mecatrônico.

Orientador (a):
Prof. Dr. Fabrício Junqueira

São Paulo
2013

## FICHA CATALOGRÁFICA

**Assumpção, Raphael Mendes de**

**Projeto e implementação de um torno CNC em ambiente de realidade virtual/ R.M. de Assumpção, C. Rolim – São Paulo, 2013.**

**111p.**

**Trabalho de Formatura – Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos.**

**1. Tornos (Projeto; Implementação) 2. Indústria de manufatura**

**I.Rolim, Caio II. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Mecatrônica e Sistemas Mecânicos III.t.**

# RESUMO

Com o avanço do poder de processamento dos computadores concomitante ao seu barateamento, as simulações computacionais tornam-se cada vez mais factíveis não apenas para grandes empresas, mas também para as pequenas e médias. Simulações permitem reduzir custos e aumentar eficiência, e são essenciais para empresas que estão ligadas a temas como robótica, manufatura, automação, entre outros. Tratando-se em especial do processo de manufatura, temos com uma das principais ferramentas as máquinas de comando numérico computadorizado (CNC). Com o intuito de melhorar o uso em especial desse tipo de máquina, este trabalho tem como proposta o projeto e implementação de um ambiente virtual de manufatura, onde se escolheu como exemplo de aplicação um torno CNC. O *software* a ser construído deve ser de fácil manuseio para que possa ser utilizado sem qualquer treinamento anterior, exceto o conhecimento de programação em linguagem G. Antes da utilização de um torno CNC real o operador pode inserir o código G no *software*, que primeiro faz uma leitura para identificação de erros de sintaxe. Em seguida, caso não seja encontrado algum erro, o programa irá simular o torneamento de uma peça genérica, com animação gráfica. Por fim, pode-se observar a peça virtualmente torneada e compará-la ao resultado real ao qual se deseja chegar. O Blender, um programa de computador de código aberto para modelagem e animação 3D, é utilizado para a modelagem gráfica dos componentes do torno CNC e do ambiente onde está inserido. Em seguida os modelos são exportados para o Microsoft XNA, plataforma de desenvolvimento de jogos, onde toda a animação é programada em linguagem C#.



**Palavras-chave:** Simulação, Manufatura Virtual, Realidade Virtual, Torno CNC.

## ABSTRACT

With the advance of computer processing power accompanying their cheapness, computer simulations become increasingly feasible not only for big companies but also for small and medium. Simulations will reduce costs and increase efficiency, and are essential for companies that are linked to topics such as robotics, manufacturing, automation, among others. As regards in particular the manufacturing process, one of the major tools is the computer numerical control (CNC). In order to improve the use of this particular type of machine, this paper aims at the design and implementation of a virtual manufacturing environment, where a CNC lathe was chosen as an example application. The software should be easy to handle so it can be used without any prior training, except the knowledge of programming G language. Prior to the use of a real CNC lathe operator can enter the G-code in the software that makes a first read to identify syntax errors. Then, if no mistake is found, the program will simulate turning a generic piece, with the proper graphic animation. Finally, it will be possible to observe the virtually turned piece and compare it to the actual result which is wanted to reach. Blender, an open source computer program for 3D modeling and animation, is used for graphical modeling of the components of a CNC lathe and the environment in which it operates. Afterwards the models are exported to Microsoft XNA platform game development, where all the animation is programmed in C# language.



**Keywords:** Simulation, Virtual Manufacturing, Virtual Reality, CNC lathe.

# LISTA DE FIGURAS

# SUMÁRIO

# 1. INTRODUÇÃO

A introdução das ferramentas de controle de comando numérico (CNC) na indústria mudou radicalmente as formas que se dão os processos industriais. Curvas são facilmente cortadas, complexas estruturas com três dimensões tornam-se relativamente fáceis de serem produzidas e o número de passos no processo com intervenção de operadores humanos é drasticamente reduzido (Nishizawa, 2010). Ou seja, obtém-se facilidade de produção, rapidez na execução e redução na mão de obra envolvida no operacional. A contrapartida que se observa é um forte aumento na complexidade dos sistemas mecatrônicos constituintes do maquinário, que afetam diretamente os custos envolvidos. E num segundo momento, implicam menores margens a erros que danifiquem tais equipamentos, já que o preço dessas máquinas é geralmente elevado. Dessa maneira, o desenvolvimento de tecnologias que diminuam a referida pressão sobre custo e margem exerce papel de grande importância para o aperfeiçoamento de sistemas produtivos inteiros e, portanto, fica evidenciada a relevância do presente trabalho como será exposto.

Dessa forma, gera bons frutos a combinação da tecnologia de realidade virtual aliada à simulação da dinâmica de funcionamento dos equipamentos mencionados acima. Torna se cada vez mais frequente a predileção por etapas de simulação nesses sistemas de manufatura virtual aos testes propriamente ditos. Impacta diretamente na diminuição da ocorrência de erros de projeto e, em última instância, aumenta a frequência da melhor disposição possível dos equipamentos a que se tenha acesso. Ambos os fatores implicam, no mínimo, redução de custos e aumento na eficiência de toda a cadeia produtiva em questão. Antes de a manufatura virtual ser utilizada era necessário entender formas 2D e imaginá-las no mundo real, o que era suscetível a erros. Considerável experiência e alto custo eram necessários para resultados precisos (Kim; Yang; Han, 2006).

Apesar do uso cada vez mais difundido de programas dessa natureza, a maior parte desses *softwares* é relativamente cara. Assim, sobretudo em plantas fabris menores, muitas vezes esses gastos não superam o custo de oportunidade de projetistas experientes de confiarem em seu conhecimento prévio em detrimento da aquisição dos mesmos. Então, é onde se insere o seguinte trabalho que propõe a criação de um simulador de um torno CNC virtual baseado apenas em outros *softwares* livres. Sua relevância está em pelo menos possibilitar um amplo acesso a um mínimo de tecnologia que possa impactar positivamente os resultados de alguém que não disponha de acesso

a programas similares mais elaborados. Posteriormente, com o presente trabalho também se espera que sirva de suporte para desenvolvimento de uma variedade maior de simuladores de equipamentos e, em última instância, possibilite a simulação não apenas de uma célula produtiva, mas também de toda uma fábrica virtual.

A estrutura desse trabalho abrange na Seção 2 a conceituação, análise e vantagens relacionadas à exploração de tecnologias relativas à Realidade Virtual e Manufatura Virtual. Em seguida, uma breve revisão de alguns tópicos relativos à máquina-ferramenta que será alvo do projeto final do *software,* i.e., o torno de comando numérico. Após, apresentam-se as ferramentas computacionais que servirão de base para o desenvolvimento do projeto, seguidos por um estudo do estado da arte de trabalhos similares já existentes. Na Seção 3 é apresentada a definição dos requisitos do projeto, bem como o projeto em si. Na Seção 4 é apresentada a implementação do mesmo. Na Seção 5 são apresentadas as conclusões.

# 2. REVISÃO BIBLIOGRÁFICA

## 2.1. Realidade Virtual

Existe certa difusão acerca da definição de Realidade Virtual (RV), i.e., uma rápida busca sobre o tema retorna conceitos diferentes e muitas vezes até contraditórios. Existe uma notável discrepância acerca da definição de RV, uma vez que as definições propostas variam quanto ao que consideram como elementos necessários para se constituir a dita experiência de RV (Burdea; Coiffet, 2004). Em uma das principais referências a cerca do tema, propõe-se a definição de RV como: o uso de um ambiente 3D gerado computacionalmente – o ambiente virtual (AV) – no qual, o usuário pode navegar e interagir havendo uma simulação em tempo real de um efeito que estimule ao menos um dos cinco sentidos do usuário (Gutierrez *et al.,* 2008).

- Navegar: refere-se à habilidade de ocorrer uma movimentação e consequente exploração do AV.
- Interagir: refere-se à habilidade de selecionar e movimentar alguns dos objetos representados nesse AV.

Ainda segundo Gutiérrez, resultados satisfatórios (ou não) da experiência em RV dependem da capacidade em se atingir a imersão física e a presença psicológica do participante. Enquanto a imersão se refere ao quanto o usuário está isolado do ambiente externo ao participar do AV; a presença psicológica é algo intangível, sendo que um possível sinal dela é o quão similar a uma situação real uma pessoa se comporta dentro do AV. A imersão pode se dar em vários níveis, que impactam diretamente os resultados sobre a presença psicológica. Em um "sistema completamente imersivo", o usuário é completamente envolvido pelo AV, sem qualquer tipo de contato com o ambiente real; enquanto que em ambientes "semi-imersivos" e "não-imersivos", algum sentido do usuário ainda é influenciado por elementos fora do domínio do AV (Gutiérrez *et al,* 2008). Observa-se alguns exemplos nas Figuras 1 e 2.

A partir de 1990, o desenvolvimento das tecnologias relacionadas à exploração da RV foi intensificado com o avanço de tecnologias computacionais (Shi; Ding; Zhou, 2008), a saber: computação gráfica, conexão homem-máquina, tecnologia de sensores e inteligência artificial. Assim, o constante desenvolvimento de equipamentos mais modernos vem garantindo cada vez mais su-

cesso na experiência de RV que, por consequência, amplia e muito os horizontes de aplicação da tecnologia.

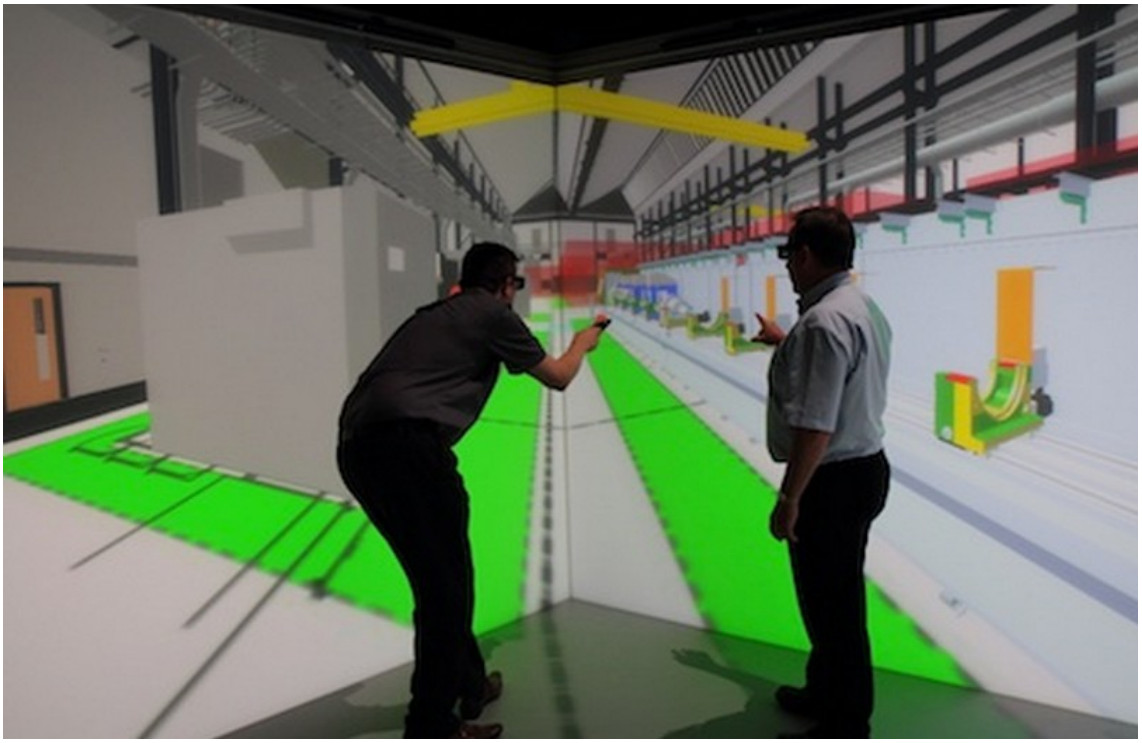

Figura 1. Exemplo de sistema completamente imersivo em RV (Kirkley, 2012).

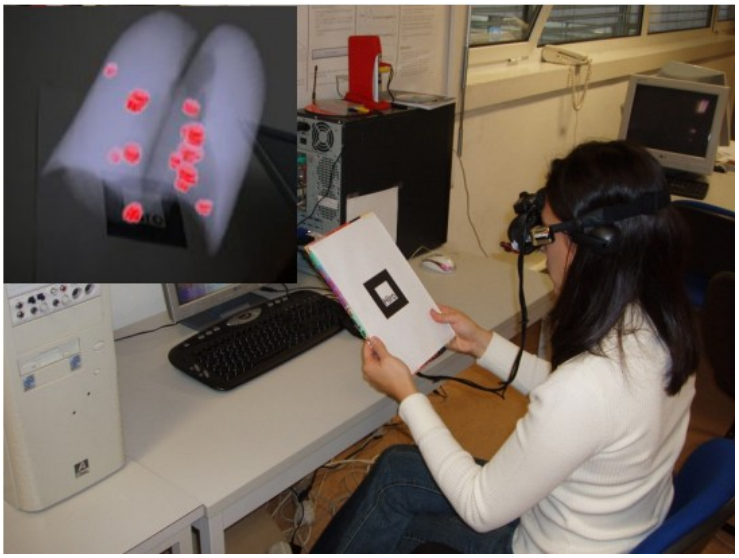

Figura 2. Exemplo de sistema semi-imersivo em RV (Santos, 2012).

Esse desenvolvimento tecnológico está diretamente relacionado às interfaces do ser humano com o meio externo, i.e., cada um dos cinco sentidos. Dessa forma, o surgimento de novos equipamen-

tos geralmente ocorre como forma de estímulo para um ou outro sentido. Na sequência, para os que são atualmente mais explorados, alguns comentários relevantes encontrados na literatura e exemplos dos equipamentos utilizados.

- Visão: um dos sentidos mais explorados pela RV e caracteriza-se por ser uma fonte primária de informação, que muitas vezes vem para confirmar outros sentidos (Gutiérrez *et al.,* 2008). Diversos tipos de monitores e projetores até mesmo combinações integradas de vários são muito frequentes, o uso de óculos especiais também vem sendo cada vez mais difundido (Figura 3).
- Audição: além da simulação direta dos ruídos de simulados no AV pura e simplesmente, é muito relevante para dar verossimilhança à reprodução do ambiente em si. Sobretudo, o som espacial que pode ser notado como proveniente de diferentes localizações (Gutiérrez *et al,* 2008). Conjuntos inteligentes de caixa de som são os equipamentos que conseguem transmitir o maior grau de imersão até mesmo para RV compartilhada entre vários usuários, mas geralmente são muito caros. Alternativas mais baratas são caixas de som sem integração ou até mesmo fones de ouvido.

- Tato: com a tecnologia atual, sistema com a exploração desse sentido são muito especializados (Gutierrez *et al.*, 2008). Parte da literatura considera que a exploração desse sentido só se dá por meio de equipamentos que estimulem nervos e ou tendões; no entanto, outros consideram que também há exploração tátil em equipamentos cujo foco esteja numa simulação mais realística de manipulação de objetos dentro do AV (Figura 4).

A RV como ferramenta complementar de ensino vem sendo amplamente usada com o avanço da capacidade computacional dos equipamentos modernos e cada vez mais aplicada a diferentes ramos do conhecimento. Essa aplicação está pautada por uma linha de teóricos chamados "Construtivistas". Como em Wang e Cheng (2010), o construtivismo afirma que a experiência é um fator crítico no processo de aprendizagem, i.e., o uso de atividades *hands on* no ensino se mostra mais eficiente do que assistir passivamente a longas aulas meramente expositivas. Ainda, ressaltam a relevância do conceito do "Espaço Virtual de Aprendizagem" (EVA) – uma representação 3D de um ambiente virtual que permite a interação entre usuários. Segundo Wang e Cheng, essa seria a ferramenta de maior eficácia dentro do tema de RV aplicada ao ensino.

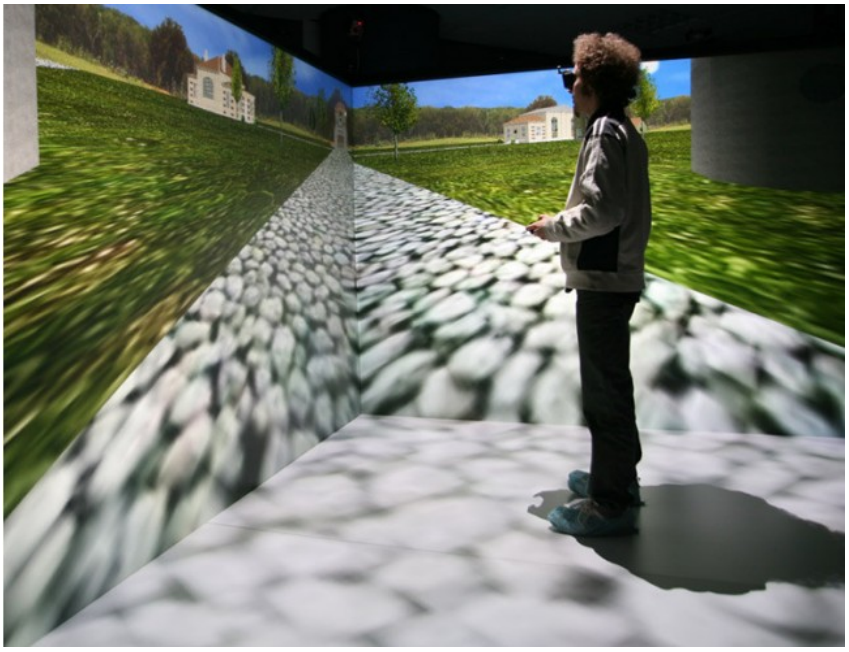

Figura 3. Exemplo de uso combinado de estimuladores visuais (Santos, 2004).

Figura 4. Diferentes tipos de luvas de estímulo tátil (Santos, 2012).

O comportamento humano está sujeito a influências do ambiente a que o indivíduo está exposto (Wang; *et al.,* 2010). Dessa forma, as pessoas tendem a tomar decisões acertadas mais facilmente num ambiente que lhes seja familiar. E, contrariamente, um ambiente desconhecido implica maior dificuldade no processo decisório das mesmas, o que pode gerar resultados completamente inesperados (Wang; *et al.,* 2010). Dessa maneira, o treinamento prévio num ambiente virtual seguro e fidedigno ao real pode gerar resultados positivos sobre a confiança dos estudantes que implica em última instância um comportamento mais adequado em situações reais posteriores.

Dentre as aplicações comuns, uma que vem sendo bastante desenvolvida é o treinamento de estudantes de medicina voltado para a prática cirúrgica. Relatos sobre acontecimentos indesejados adversos causados por erros técnicos durante a prática cirúrgica e mesmo pela melhor compreensão do ferramental disponível evidenciam a importância e necessidade de um ambiente de aprendizagem seguro e pedagógico (Grantcharov, 2008) – como ilustrado na Figura 5. Estudantes de medicina que tiveram acesso ao treinamento pelo simulador apresentaram desempenho muito superior aos que se limitaram às técnicas de ensino tradicionais (Seymour, 2002).


Figura 5. Exemplo de simulação de técnica cirúrgica (Santos, 2012).

Essa aplicação na medicina é apenas uma dentre inúmeras existentes da realidade virtual como ferramenta de ensino. Dessa maneira, é razoável pensar que os inúmeros benefícios mencionados possam ser aproveitados das mais diversas formas com o treinamento de pessoal para manuseio adequado de equipamentos de engenharia. Sobretudo se levar em conta o conhecido histórico atraso do Brasil na formação de equipe técnica suficiente que suporte um desenvolvimento do setor industrial mais robusto do país. Nesse sentido, dentro dos estudos de RV, existe uma área de pesquisa denominada Manufatura Virtual, que, como será explorado, deverá estar ligada diretamente com o mencionado sucesso (ou não) de indústrias que empreguem essa técnica à sua linha de produção.

## 2.2. Manufatura Virtual

A indústria manufatureira vem experimentando enormes transformações ao longo das últimas décadas. Essas mudanças estão diretamente relacionadas aos crescentes requisitos de preço e qualidade do produto, rapidez na fabricação etc. (Cheng; Bateman, 2008). A produção pautada por custo, eficiência e qualidade torna-se um problema numa realidade em que o ciclo de vida dos produtos se reduz continuamente (Zuehlke, 2004). Dessa forma, a manufatura virtual (MV) vem como uma ferramenta que permite aos produtores encararem melhor as diversas dificuldades nesse cenário. A MV é uma técnica promissora que viabiliza o atendimento aos requisitos impostos por essa realidade dinâmica, fazendo com que se torne essencial e inevitável para os futuros fabricantes (Cheng; Bateman, 2008).

O escopo do projeto de MV, envolve três paradigmas básicos (Porto et al., 2002), que resumidamente, são (ver Figura 6):

- Sistemas orientados para o projeto: uso de protótipos para definir o processo de manufatura mais adequado para o caso.
- Sistemas orientados para o processo: avaliação do planejamento dos parâmetros do processo de usinagem que foi definido
- Sistemas orientados para o controle: avaliação da simulação do plano de ação antes de sua implementação no sistema real.

Como a manufatura pode ser compreendida como a dinâmica da cadeia como um todo - ou seja, muito além do processo de fabricação em si – encontra-se na bibliografia uma conceituação muito ampla de MV. Conforme proposto em Cheng *et al.* (2008), envolveria todos os aspectos que se relacionassem a, por exemplo: projeto do processo de manufatura e plano de negócios, esforços de venda e marketing, operações de chão de fábrica, integração com cadeia de suprimentos etc (veja Figura 7).

Sem dúvidas é uma abordagem válida e lógica, mas o presente trabalho pretende explorar o enfoque da MV no âmbito do processo de fabricação do produto – o qual já é por si só um ramo com enorme espaço para desenvolvimentos – e não em temas de sistema de venda e logística de escoamento da produção. Particularmente nesse escopo, a MV faz uso de modelos digitais integrados criados a partir da modelagem física e lógica dos componentes do sistema de manufatura, e então

utiliza computadores para a simulação do comportamento previsto (Lee; Han; Yang, 2010). Nesse sentido, é frequente o uso da denominação como fábrica virtual (FV) ou em algumas situações *smart factory*. A FV abrange modelagem, visualização, simulação, avaliação de resultados, gerenciamento de dados e comunicação (Yang; *et al*., 2011), observe exemplo na Figura 8.

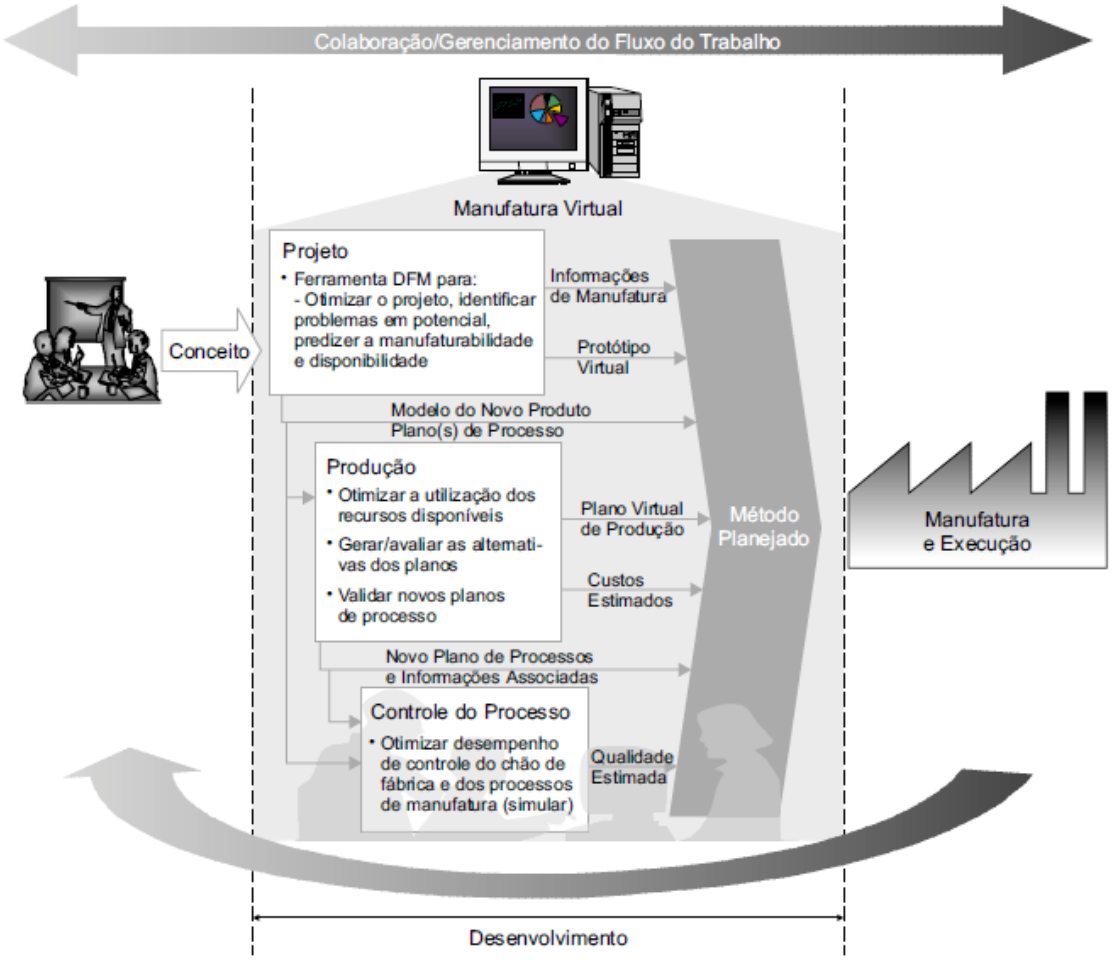


Figura 6. Ambiente da Manufatura Virtual. (Porto e Palma, 2000).

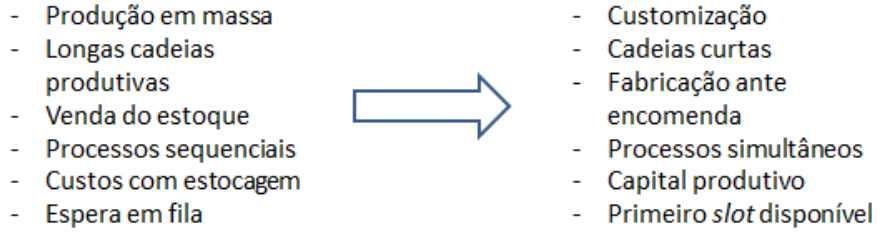


Figura 7. Transformação derivada da manufatura virtual (traduzido de Cheng; Bateman, 2008).

Figura 8. Ambiente de fábrica virtual (Lee; Han; Yang, 2010).

Na literatura, até mesmo as funcionalidades da FV foram abstraídas em diferentes segmentos. Cecil e Kanchanapiboon (2007) propõem uma decomposição nas seguintes sub-áreas: (i) prototipagem a nível fábrica, (ii) ambientes virtuais de montagem e (iii) prototipagem virtual de atividades básicas. À nível de fábrica, a FV proporciona simulações para possíveis modificações do projeto de *layouts* (Kesavadas *et al*., 1999), sendo que otimizações nesse sentido impactam diretamente na redução de tempos de espera e gargalos do sistema fabril em questão. No que tange à montagem, a FV é útil na análise individual de células de montagem que componham uma determinada estação de trabalho (Chryssolouris *et al*., 2000). Finalmente, para atividades de mais baixo nível, existem poucos *softwares* que possibilitem uma simulação fidedigna num ambiente virtual (Cecil *et al*., 2007).

Uma possível abordagem para a estruturação das fases envolvidas foi resumida em Yang *et al*. (2011), e consiste:

1) Modelagem: etapa em que se define as informações básicas necessárias para as etapas posteriores. Inclui, por exemplo, a modelagem e originação de um primeiro modelo geométrico em RV (Figura 9).

2) Aplicação: em essência é a fase de simulação – centro de todo o fluxo de trabalho. Ocorre a construção virtual do processo de manufatura, ou seja, proporciona a visualização do processo. E,

portanto, dá uma percepção mais realista do que é esperado que ocorra na realidade ainda na via virtual (Figura 10).

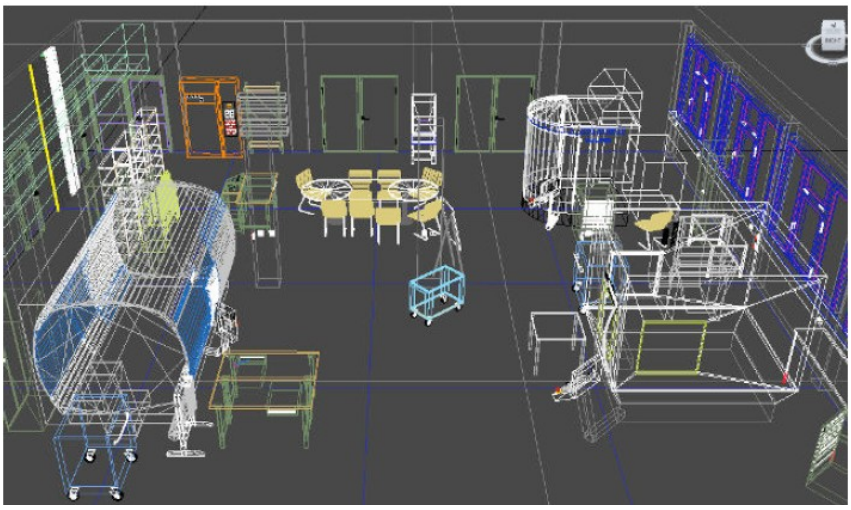

Figura 9. Exemplo de modelagem geométrica de um ambiente em RV (Yang; et al., 2011).

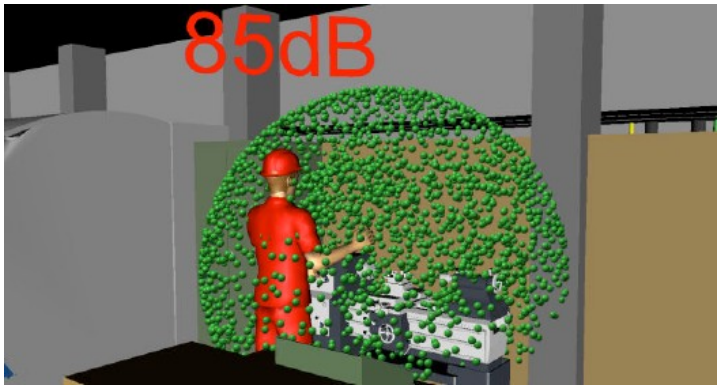


Figura 10. Simulação da propagação de som num torno virtual (Yang; Malak, et al, 2011).

3) Adaptação: de acordo com os resultados obtidos da fase anterior, deve-se realizar as devidas adaptações que visem a otimização do processo.

No entanto, essa é uma área do conhecimento em que ainda há muito espaço para desenvolvimento, já que muito dos recursos disponíveis não atendem às necessidades do usuário de maneira simples e eficiente. Atualmente, o que se dispõe para tanto ainda está muito distante do ideal; uma vez que, numa busca por diferenciação do produto, estes acabam apresentando um excesso de funcionalidades, que se reflete na incapacidade do cliente de dominar completamente seu uso (Zuehlke, 2010). Além disso, há necessidade de melhoras na forma como a tecnologia é implementada, não apenas em termos da precisão da simulação, mas também no nível de imersão dos

usuários (Lee; Han; Yang, 2010). Ainda segundo estes autores, o custo envolvido em sistemas cuja precisão seja melhor, também é uma barreira que deve ser superada.

Finalmente, outro ponto importante no desenvolvimento das técnicas de FV, é a modelagem de máquina-ferramenta. Para se obter resultados mais eficientes de um modelo de FV, este deve abranger realisticamente e de forma satisfatória aspectos como, por exemplo: o *layout* do chão de fábrica, equipamentos instalados e suas condições no que tange a capacidade e necessidade de manutenção (Kjellberg et al., 2009). Além disso, ainda segundo os pesquisadores, como os atuais sistemas de manufatura envolvem agentes nas mais variadas localidades, é de grande importância que as informações contidas nos modelos garantam a possibilidade e facilidade de comunicação e interoperabilidade.

Com tudo isso, fica evidente a importância não só da adoção dessa tecnologia por parte das industrias manufatureiras, mas também da continuidade no desenvolvimento e melhoramento das técnicas por parte dos pesquisadores. Nesse sentido, a importância do seguinte trabalho está justamente na tentativa de se contribuir de alguma forma para esse avanço.

### 2.3. Máquinas ferramentas

Máquina-ferramenta, ou máquina operatriz, é um equipamento utilizado na fabricação de peças de revolução por meio da movimentação de um conjunto de ferramentas (Santos, 2007). Apesar de muitas vezes essa denominação ser associada ao estereótipo de máquina de grande complexidade estrutural, os primórdios desse tipo de equipamento estavam presentes desde nos desenhos da Renascença (Figura 11) e, muito antes, também no Egito Antigo (veja Figura 12). Ou seja, apresentavam-se como uma solução engenhosa e simples para entraves de projetos de épocas antigas sem necessariamente apresentarem a inicialmente imaginada complexidade.

Com a Revolução Industrial, surgiram as primeiras máquinas-ferramenta segundo os princípios modernos, e até a década de 70, os progressos obtidos nesses equipamentos eram evoluções no mecanismo do equipamento (Stoeterau, 2004). A partir de então os avanços posteriores estiveram relacionados à introdução de outras tecnologias que não a mecânica pura, por exemplo, o Comando Numérico (explorado no item seguinte). O torno é um equipamento de enorme versatilidade, empregado na fabricação e acabamento de peças dos mais diversos formatos. As peças a serem

esculpidas são fixadas entre as pontas de eixos revolventes a fim de que possam ser trabalhadas por uma ponta de ferramenta.

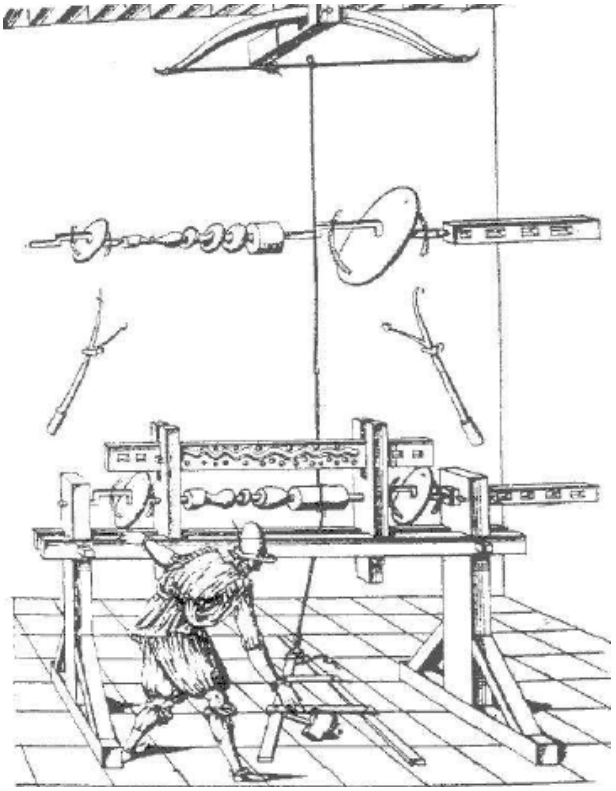

Figura 11. Torno acionado por arco de 1565 (Spur, 1979).

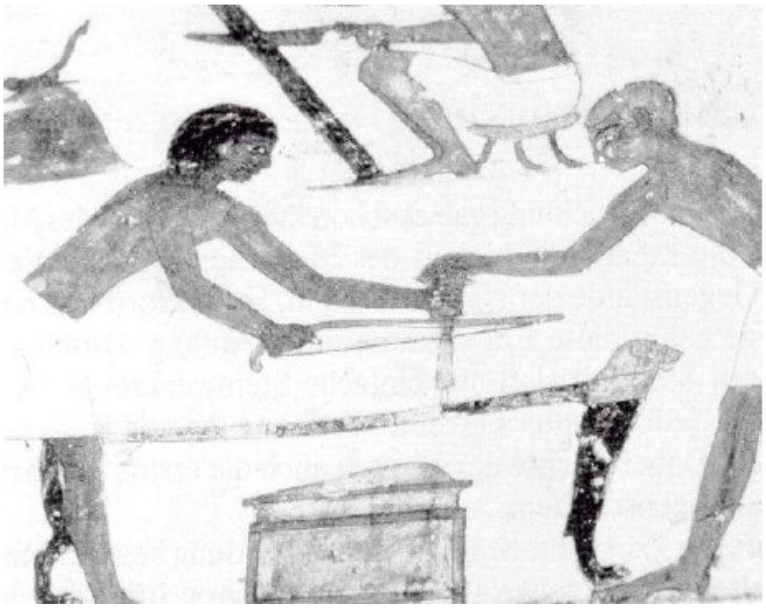

Figura 12. Furadeira de arco egípcia 1500 a.C. (Spur, 1979).

Dessa forma, explica-se a predileção por esse tipo de equipamento para o desenvolvimento do presente trabalho em detrimento de quaisquer outros tipos de máquinas-ferramenta (p.ex. furadeira, fresadora, aplainadora, etc). Entende-se que dada essa grande flexibilidade em um único equipamento, coloca-se como um bom início para o desenvolvimento do projeto proposto em linha com o que foi explorado no item de MV.

### 2.4. Comando Numérico e Código G

Como já explorado anteriormente, é crescente o número de requisitos impostos aos atuais processos de manufatura, o que implica a necessidade de constantes avanços nas ferramentas produtivas. A partir da década de 70, o Comando Numérico (CN) despontou como um dos principais alicerces no desenvolvimento das máquinas-ferramenta (Stoeterau, 2004). Com a tecnologia de CN, o fator tempo é reduzido substancialmente pela eliminação de intervalos improdutivos, consequência, por exemplo, da alta versatilidade destas máquinas que englobam várias operações em um único equipamento o que diminui ainda mais os tempos mortos intermediários (Mastelari, 1996).

Na prática, uma máquina-ferramenta de CN é equipada com um controlador computadorizado capaz de comandar a movimentação da ferramenta durante a operação de corte, substituindo total ou parcialmente o papel que era desempenhado pelo usuário. Para efetuar essas ações, o equipamento necessita de um programa que traduza as diversas tarefas a serem executadas numa lista de códigos padronizados que, genericamente, associa o movimento relativo entre a ferramenta de corte e a peça a ser usinada (Arthaya *at al.,* 2010).

Um programa CN genérico pode ser compreendido como um conjunto de blocos (Falck, 2008), que, resumidamente, consiste de:

- **Números de Sequência**: discriminam a ordem de execução dos blocos;

- **Funções preparatórias**: palavras padronizadas e reservadas para rotinas pré-definidas;

- **Informações das Coordenadas**: são números combinados ao sistema eixo em que se deve dar o movimento;

- **Função *feed***: código para se determinar a velocidade de avanço;

- **Função Velocidade**: código para se determinar a velocidade de corte;

- **Função Ferramenta**: código que determina a posição no castelo da ferramenta de corte a ser utilizada na usinagem;

- **Miscelânea**: código para controlar funcionalidades específicas do equipamento como, por exemplo: liberação de fluído refrigerante, sentido de giro do *spindle,* troca de ferramentas etc.

Na prática, a elaboração de um programa CN é uma tarefa minuciosa, na medida em que envolve a determinação de uma série de informações referentes à, por exemplo: geometria da peça, tipo de máquina-ferramenta, ferramentas disponíveis e fundamentos da usinagem a ser executada (Marcicano, 2001). Existem quatro métodos de programação CN que são, segundo Marcicano, resumidamente:

1) Programação Manual: elaboração de um programa na linguagem que o CNC compreenda. Caracteriza-se por ser uma sintaxe de baixo nível;

2) Programação APT: conjunto de quatro tipos básicos de declarações (de geometria, de movimento, de pós-processador e auxiliares). Caracteriza-se por ser uma linguagem de nível mais alto que a da programação manual, e foi criada para superar as dificuldades e complexidades de programação enfrentadas no método anterior.

3) Sistemas Gráficos-Interativos: são sistemas computacionais destinados à programação CN que fazem uso da interação homem-máquina de maneira a determinar as condições desejadas para sua elaboração. Existem dois tipos básicos, que são: com linguagem e sem linguagem. Enquanto o primeiro faz uso de linguagens simbólicas por meio de declarações que expressam desde a escolha da ferramenta até definição da geometria; o outro utiliza ferramentas como teclas funcionais e *mouse* para facilitar a manipulação dos dados.

4) Sistemas CAD-CAM: são as ferramentas mais modernas para a elaboração de programas CN, que permitem a interpretação da geometria da peça armazenada em um arquivo CAD e a posterior geração do programa de acordo com entradas do usuário no módulo CAM. As etapas dos sistemas CAD-CAM englobam: (i) preparação das superfícies a serem usinadas, (ii) seleção dos parâmetros de usinagem, (iii) geração da trajetória da ferramenta e (iv) pós-processamento.

O escopo do projeto em questão está inserido no método de Programação Manual, e fará uso do código G para a simulação de movimentação do Torno Virtual. Existe uma ampla gama de funcionalidades abrangidas no código G, dentre essas Marcicano (2001) ressalta alguns comandos empregados para o comando de equipamentos reais – ilustrado na Tabela 1:

Tabela 1: Listagem de Funções obrigatórias para o comando de um torno real (Marcicano, 2001).

| **Código** | **Função** |
|---|---|
| G00 | Posicionamento Rápido |
| G01 | Interpolação Linear com avanço programável |
| G02 | Interpolação Circular Horário |
| G03 | Interpolação Circular Anti-horário |
| G04 | Tempo de Permanência |
| G20 | Programação em Diâmetro |
| G21 | Programação em Raio |
| G30 | Cancela Imagem Espelho |
| G31 | Ativa imagem espelho em Ox |
| G32 | Ativa imagem espelho em Oz |
| G33 | Ciclo de Rosqueamento Básico |
| G37 | Rosqueamento Automático |
| G40 | Cancela compensação de raio da ponta da ferramenta |
| G41 | Compensação de raio da ponta da ferramenta (Esquerda) |
| G42 | Compensação de raio da ponta da ferramenta (Direita) |
| G46 | Desativa a velocidade de corte constante |
| G47 | Ativa a velocidade de corte constante |
| G53 | Cancela todos os deslocamentos de pontos zeros (DPZ) |
| G54 | Ativa o primeiro DPZ da peça |
| G55 | Ativa o segundo DPZ da peça |
| G60 | Cancela área de segurança |
| G61 | Ativa área de segurança |
| G66 | Ciclo Automático de Desbaste Longitudinal |
| G67 | Ciclo Automático de Desbaste Transversal |
| G68 | Ciclo Automático de Desbaste paralelo ao perfil final |
| G70 | Admite programa em polegada |
| G71 | Admite programa em milímetro |
| G73 | Interpolação linear ponto-a-ponto |
| G74 | Ciclo de Furação Com Descarga de Cavacos |
| G75 | Ciclo de Canais |
| G76 | Ciclo automático de roscamento (profundidade) |
| G80 | Cancela ciclo automático de furação |
| G83 | Ciclo automático de furação com quebra de cavacos |

Tabela 1: Listagem de Funções obrigatórias para o comando de um torno real (Marcicano, 2001) (cont.).

| Código | Função |
|---|---|
| G90 | Programação em Coordenadas Absolutas |
| G91 | Programação em Coordenadas Incrementais |
| G92 | Origem do Sistema de Coordenadas e Limite de Rotação (rpm) |
| G94 | Estabelece Programa de Avanço (pol/min ou mm/min) |
| G95 | Estabelece Programa de Avanço (pol/rotação ou mm/rotação) |
| G96 | Programação em velocidade de corte Constante (pés/minuto ou metros/minuto) |
| G97 | Programação em rpm direta |
| G99 | Cancela G92 e define a programação em função do zero máquina |
| M00 | Parada do Programa |
| M01 | Parada opcional do programa |
| M02 | Fim de Programa |
| M03 | Sentido Horário de Rotação do Eixo Árvore |
| M04 | Sentido Anti-horário de Rotação do Eixo Árvore |
| M05 | Desliga o eixo-árvore |
| M06 | Libera o giro da torre |
| M08 | Liga o Refrigerante de Corte |
| M09 | Desliga o Refrigerante de Corte |
| M10-14 | Troca de Faixa de Rotação |
| M24 | Abrir placa |
| M25 | Fechar placa |
| M26 | Recuar o mangote do contra-ponto |
| M27 | Acionar o mangote do contra-ponto |
| M30 | Fim de Programa |

A linha de comando apresentada foi empregada em modelos reais de Tornos CNC da Romi como o Centur 20 e Centur 30. Dessa forma, infere-se que a implementação das funcionalidades da Tabela 1 atende a requisitos práticos de uma indústria de manufatura.

Ainda no que tange ao Código G, resultados satisfatórios passam por duas etapas fundamentais, que são: a verificação do código e simulação do corte. Para essas etapas, Arthaya *et al.* (2010) propõe as seguintes estruturas para os algoritmos – Figuras 13 e 14.

A estrutura de verificação sintática apresentada (Figura 13) é uma abordagem viável como sugerem os resultados do trabalho de Arthaya *et al.* No entanto, essa é apenas uma dentre inúmeras alternativas para a solução do problema do desenvolvimento de um interpretador de código. Essa questão está inserida no amplo campo do Processamento de Linguagens (PL), uma das áreas de

intenso desenvolvimento na Ciência da Computação. Nas últimas décadas, o PL tem sido alvo de estudos que visam aumentar a eficiência das técnicas utilizadas com destaque para o trabalho desenvolvido por Valiant (1975) em "*General context-free recognition in less than cubic time"*.

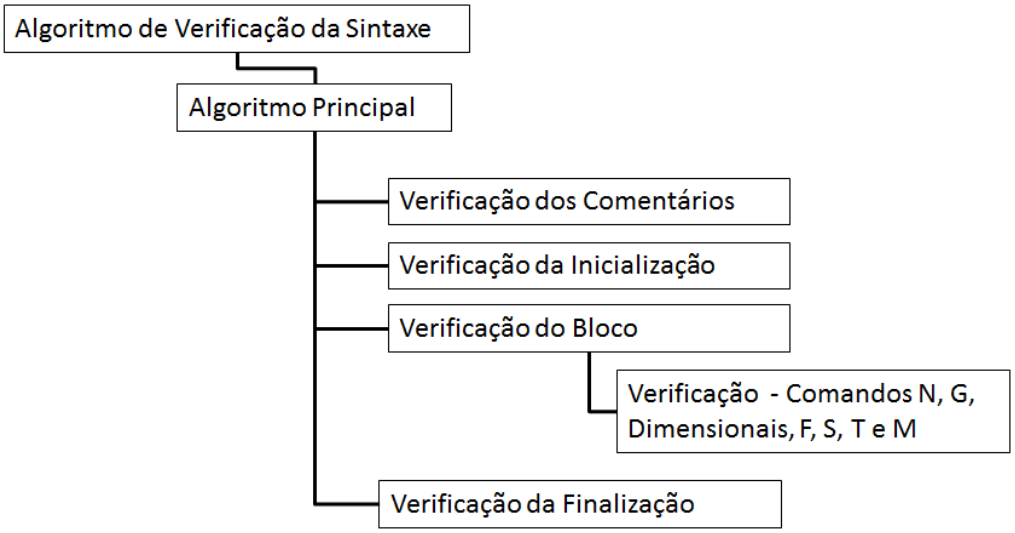


Figura 13. Estrutura de algoritmo de verificação de sintaxe (traduzido de Arthaya *et al.,*2010).

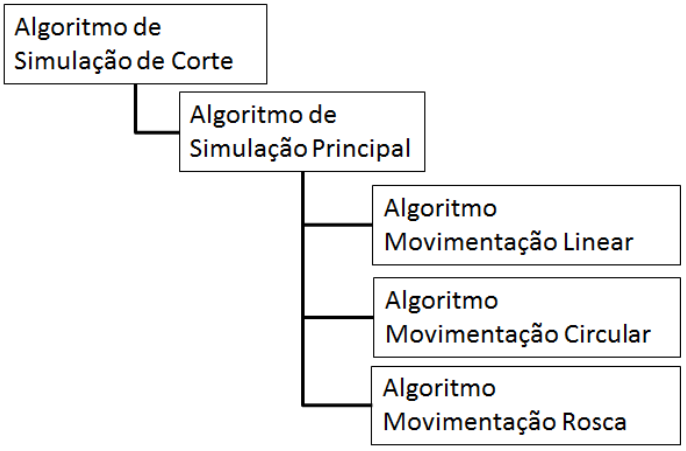


Figura 14. Estrutura de algoritmo da simulação (Arthaya *et al.,* 2010).

Segundo Vivancos (2011), um processador de linguagem (PdL) é um programa que recebe como entrada um texto escrito em algum formato, analisa-o e, opcionalmente, produz algum tipo de resultado. De acordo com o estudioso, o desenvolvimento de um PdL passa necessariamente por três etapas, que são:

- Análise Léxica: analisa uma sequência de caracteres e produz símbolos léxicos (*tokens*).

- Análise Sintática: analisa uma sequência de símbolos léxicos e os agrupa de modo a formar sentenças (frases).

- Análise Semântica: comprova a corretude dos significados da sentença.

Existem diversas ferramentas que auxiliam no desenvolvimento do PdL para cada uma dessas etapas. Alguns dos mais comuns são: Lex/Flex (Análise Léxica), Yacc/Bison (Análise Sintática), ANTLR (Análise Léxica e Sintática), etc.

Como o objetivo do presente trabalho não está relacionado com o aperfeiçoamento de PdL, mas seu sucesso final depende do uso adequado do mesmo, é útil fazer uso das ferramentas existentes para tanto. Dessa maneira, o desenvolvimento do interpretador e verificador de código G será pautado pela ferramenta do ANTLR por atender as necessidades do projeto e ser de grande aceitação na comunidade científica. Segundo Parr *et al.* (2011), o uso difundido do ANTLR (mais de 70.000 *downloads*/ano) evidencia que é eficiente para uma grande variedade de aplicações.

## 2.5. Blender

Quando se desenha objetos mais complexos, são utilizados modelos 3D. Basicamente, um modelo 3D é uma coleção de informações necessárias para desenhar os triângulos que formam um modelo 3D mais complexo. Modelos 3D são, geralmente, construídos em programas específicos para esse propósito. Entre os programas mais populares para esse fim estão o 3ds Max (Autodesk, 2013), o Blender (Blender, 2013), o Rhinoceros (Rhinoceros, 2013), entre outros. O Blender é *open-source* e gratuito.

As pessoas, em geral, associam programas gratuitos com termos como ruins ou com características limitadas, mas não é que acontece com o Blender, pois este é completamente funcional. O Blender funciona como um programa de código aberto, no qual o desenvolvimento do programa é compartilhado por várias pessoas ao redor do mundo (Chronister, 2011).

O Blender é desenvolvido pela Blender Foundation, para modelagem, animação, texturização, composição, renderização, edição de vídeo e criação de aplicações interativas em 3D. O programa foi escolhido exatamente por ser de código aberto, não sendo necessário arcar com custos de licença. Além disso, possui ferramenta própria para exportação de modelos em Autodesk FBX (Autodesk, 2013), os quais são reconhecidos no ambiente do Microsoft XNA Game Studio (Mi-

crosoft, 2013), a plataforma escolhida para animação dos modelos no presente trabalho. No momento, está sendo utilizada a versão 2.66 do Blender. Na Figura 15 é apresentado o ambiente do Blender com um modelo 3D de uma motocicleta em diferentes vistas possíveis de se trabalhar no programa.


Figura 15. Exemplo de uma modelagem em Blender e diferentes vistas possíveis (Chronister, 2011).

O Blender possibilita o desenvolvimento de modelos tão complexos como mostra a Figura 16, a qual possui a aparência de um torno real.

### 2.6. Microsoft XNA Game Studio

O Microsoft XNA Game Studio pode ser definido como um ambiente integrado de desenvolvimento (IDE, da sigla em inglês *Integrated Development Environment*) para desenvolvimento de jogos em XNA, o qual é uma extensão do Microsoft Visual Studio (Reed, 2011).

Os elementos básicos de um jogo digital podem ser divididos em (traduzido de Innocent, 2008):

1 – *Feedback loop*: o jogador tem controle direto do jogo e recebe respostas imediatas. Essa interação resulta em uma comunicação entre o jogador e o jogo com um fluxo contínuo de eventos.

2 – Modo de percepção: há uma sensação dos jogadores de realmente estarem dentro do jogo, através da habilidade que eles possuem de manipular o ambiente, entre outros fatores, como simulação física, perspectiva 3D, etc.

3 – Espaço transmutacional: apesar de os jogos dependerem da simulação de um mundo real, eles são altamente mutáveis, as representações podem mudar com mudança de parâmetros da simulação, através da ação do jogador ou até mesmo em alteração direta no software.

4 – O mundo como um sistema de signos: a natureza da computação requer que tudo seja formalmente definido. Sendo feita as definições, elas podem ser analisadas como uma relação entre signos, transformando o mundo virtual na incorporação de um sistema de signos.

Figura 16. Modelo de um torno feito em Blender 2.5 (TurboSquid, 2013).

Um modelo que resume as ideias de um jogo deve conter as seguintes características (Innocent, 2008):

1: ser um mundo simbólico definido em espaço virtual;

2: ser mediado por simulação;

3: a interação permitir controle e respostas; e

4: a jogabilidade definir a lógica do mundo simbólico.

Como pode ser inferido acima, o ambiente de um jogo, proposto para o desenvolvimento do presente trabalho, possui certas macroestruturas e definições que devem ser obedecidas para que uma boa sequência de eventos garanta a "fluidez" de um jogo. Portanto, a sugestão é utilizar o Microsoft XNA, um programa que foi desenvolvido com essa ideia, facilitar o desenvolvimento de jogos através de uma estrutura pronta que garanta um bom fluxo de jogo.

O XNA é baseado no .NET Framework (iniciativa da Microsoft que visa uma plataforma única para desenvolvimento e execução de sistemas e aplicações), com versões que rodam no Windows, Windows Phone e Xbox. Pode ser baixado de forma gratuita e para seu funcionamento é necessário instalar primeiro o Visual Studio 2010 Standard Edition ou o Visual C# 2010 Express Edition e para rodar jogos no Windows é necessário o DirectX 10 ou posterior (Reed, 2011). No caso do presente trabalho, estão sendo utilizados o Visual C# 2010 Express e o DirectX 10.

Além disso, será utilizada a versão mais atual do XNA, o XNA Game Studio 4.0. O fluxograma do funcionamento do XNA é apresentado na Figura 17.

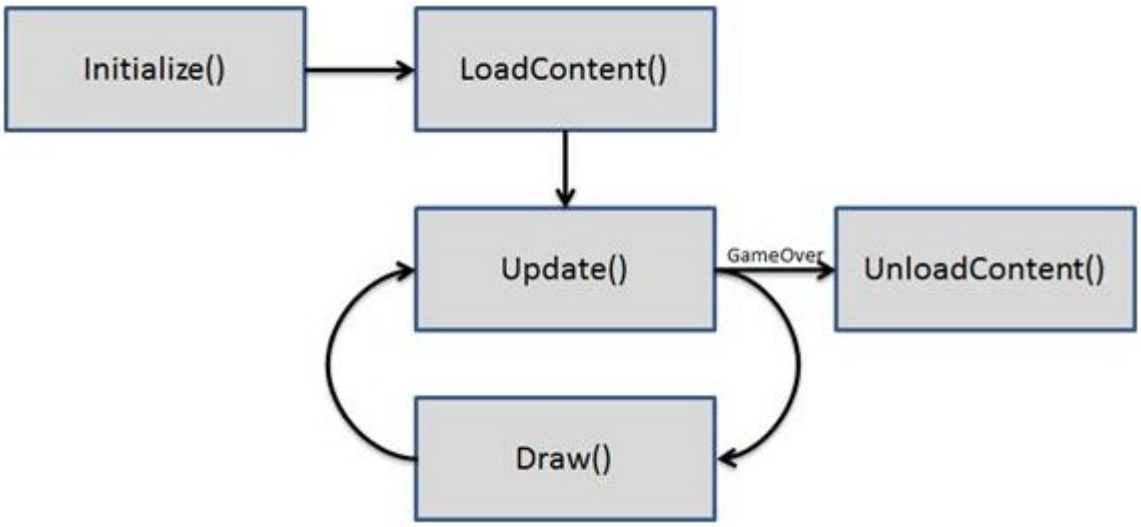


Figura 17. Fluxo básico de um jogo em Microsoft XNA Game Studio (Microsoft, 2013).

O método "Initialize( )" é o lugar onde se inicia variáveis e outros objetos.

O método “LoadContent( )” é chamado após a execução do método “Initialize( )” ou em qualquer momento onde seja aconselhável recarregar objetos gráficos do jogo (quando ocorrer mudanças nas configurações do dispositivo gráfico, por exemplo). Nesse método deverá ocorrer a carga de imagens, modelos 3D, sons, entre outros recursos.

Os métodos “Update( )” e “Draw( )” estão dentro de uma rotina que é conhecida como “Game Loop”, a qual é iniciada logo que tenha sido concluída a carga do jogo, nos métodos “Initialize( )” e “LoadContent( )”. Um “Game Loop” consiste de uma série de métodos que são chamados repetidamente até que o jogo encerre. Nesse caso toda a lógica do programa deverá ser acionada através dos métodos “Update( )” e “Draw( )”.

O método “Draw( )” é usado para desenhar os elementos gráficos na tela.

O método “Update( )” é o lugar para “processar o estado atual do jogo”. Na prática, pode-se dizer que todo processamento que não envolva desenho deve ser chamado a partir desse método, como, por exemplo, acionar o cálculo de atualização da posição de objetos, checar colisões, atualizar um placar, verificar se o jogo foi concluído, entre outros. Dessa forma um jogo sempre está executando alguma ação, independente de qualquer entrada do usuário.

O método “UnloadContent( )” é chamado sempre que o “Game Loop” é interrompido. Esse método é usado para “destruir” quaisquer conteúdos que tenham sido carregados, durante o método “LoadContent( )”, e precisem de algum tratamento especial (Microsoft, 2013).

## 2.7. Simuladores virtuais de tornos CNC

Existem soluções disponíveis no mercado para aplicação de manufatura virtual. Em especial para o tema do presente trabalho, há programas bem completos com várias funcionalidades para simulação de tornos CNC, com inserção de código G, e respectiva animação de torneamento em ambientes 3D.

No entanto, em sua maioria são programas que oferecem versões gratuitas, com várias funcionalidades restritas, caso contrário deve-se pagar pelo programa completo, que em geral são caros. Dentro os programas disponíveis destacam-se três por serem programas bem completos, com várias funções, muitas vezes oferecendo não apenas soluções para torno CNC, mas também para outras máquinas ferramentas CNC.

### *2.7.1. VR CNC Turning*

O VR CNC Turning é um programa oferecido pela Denford, empresa da Inglaterra provedora de soluções CAD / CAM e projetos para educação (REA Foundation, 2013). O *software* está disponível para compra por 550 dólares. No mesmo *site* é oferecido *softwares* de simulação virtual para fresamento CNC pelo mesmo preço.

As características incluem barras de ferramentas personalizáveis, gerenciamento abrangente de ferramentas, e opções de modificação do código NC. Utiliza código G, como pode ser observado na Figura 18.

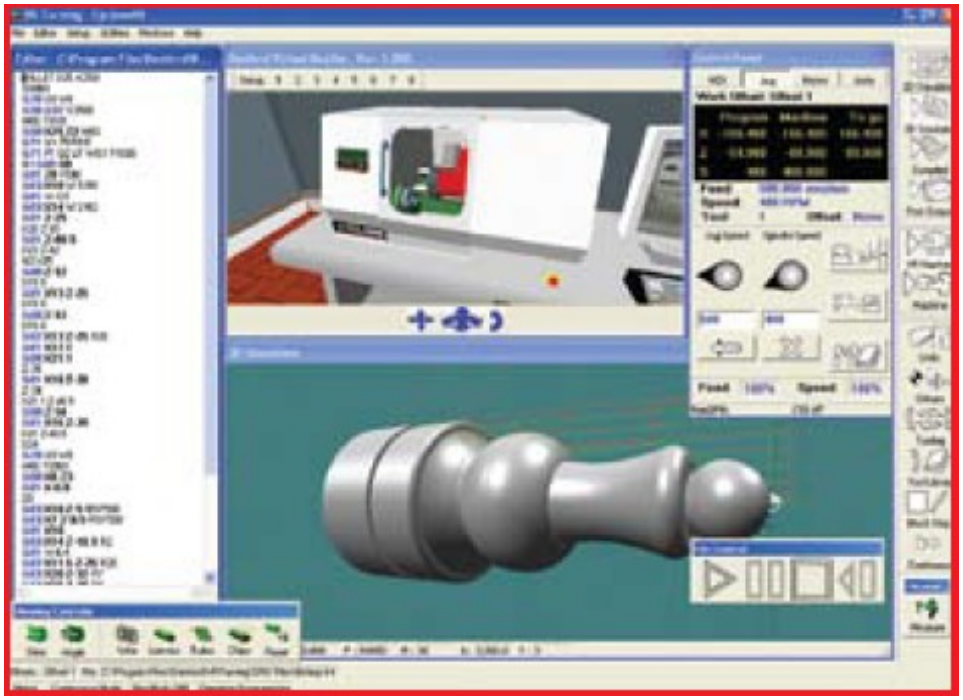

Figura 18. VR CNC Turning, *software* disponível para compra (REA Foundation, 2013).

### *2.7.2. Swansoft CNC Simulator*

Simulador 3D em tempo real de máquina CNC com *software* de verificação de código G (SSCNC, 2013).

O pacote de *software* inclui:

1 - Simulador realista de máquina: programação e operação de várias máquinas CNC em um ambiente virtual seguro. Todas as operações de *set-up*, como a determinação das dimensões da peça e suas fixações, definição, seleção de ferramentas de medição.

2 - Analisador e depurador de código G com recursos avançados.

O *software* permite a simulação de programas CN seja na visualização 2D das seções transversais de peças ou na simulação 3D da área de usinagem virtual e simulação 3D de remoção de material. No decurso de uma simulação, os cálculos de detecção de colisão são executados levando em conta todos os componentes da área de operação da máquina.

3 - Módulo de servidor para instruir e gerenciar remotamente estudantes e professores.

As funcionalidades apresentadas podem ser observadas na interface do programa nas Figuras 19 e 20.

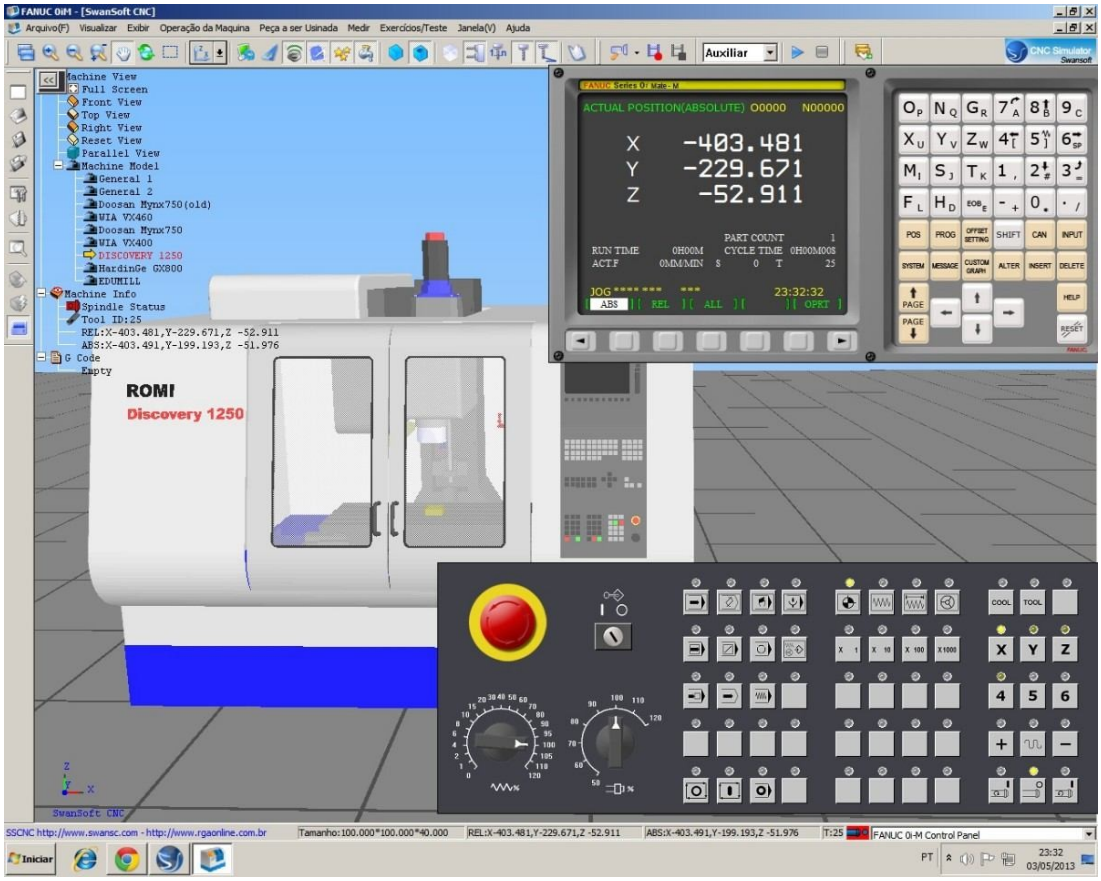


Figura 19. Swansoft CNC Simulator (SSCNC, 2013) [1].

O programa pode ser baixado gratuitamente para testar todas os recursos por 7 dias. Caso contrário pode-se pagar por versões que custam 198 ou 459 dólares, dependendo das funcionalidades desejadas.

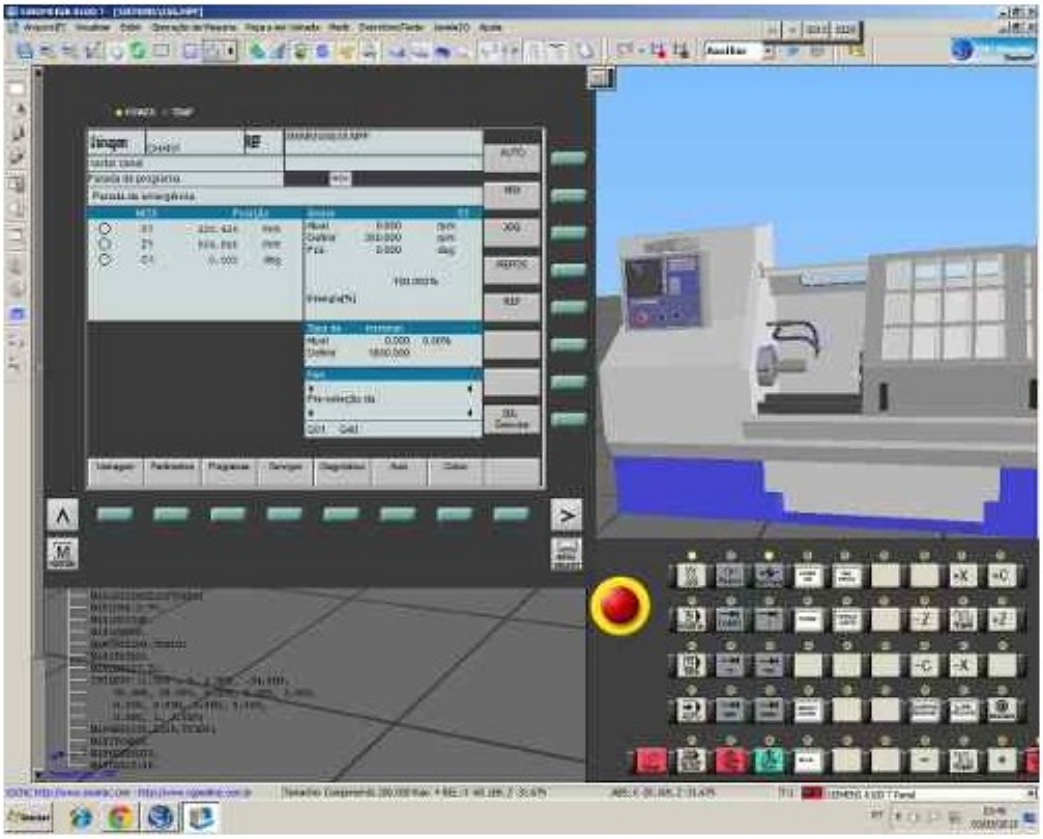

Figura 20. Swansoft CNC Simulator (SSCNC, 2013) [2].

### *2.7.3. CNC Simulator Pro*

A ideia do CNC Simulator Pro é proporcionar um simulador CNC 3D versátil e completo, com capacidades CAM (CNCSimulator.com, 2013). Usa a própria linguagem da fabricante, a "G-code Fanuc" (lateral direita da Figura 21). O programa é livre para qualquer um usar, mas com limitações para usuários não-pagantes.


Figura 21. CNC Simulator Pro Turning Center, código G na lateral direita (CNCSimulator.com, 2013).

Para adquirir a versão Platinum há três opções:

- Edição Platinum 1 por 99 dólares por ano, incluindo todas as atualizações e novas funcionalidades.
- Edição Platinum 2 por 245 dólares e acesso livre para sempre, não inclui atualizações gratuitas.
- Edição Platinum 3 por 155 dólares por três ano, não inclui atualizações gratuitas.

A Figura 22 mostra o programa em pleno funcionamento. À medida que a peça é torneada com simulação virtual, simultaneamente são percorridas as linhas do código G que estão sendo executadas no momento. A Figura 22 também proporciona uma ideia de colocar a máquina ferramenta em movimento na parede lateral do torno, ao invés de um carrinho percorrendo a base do torno horizontalmente, como foi modelado pensando inicialmente e será visto no capítulo 4.

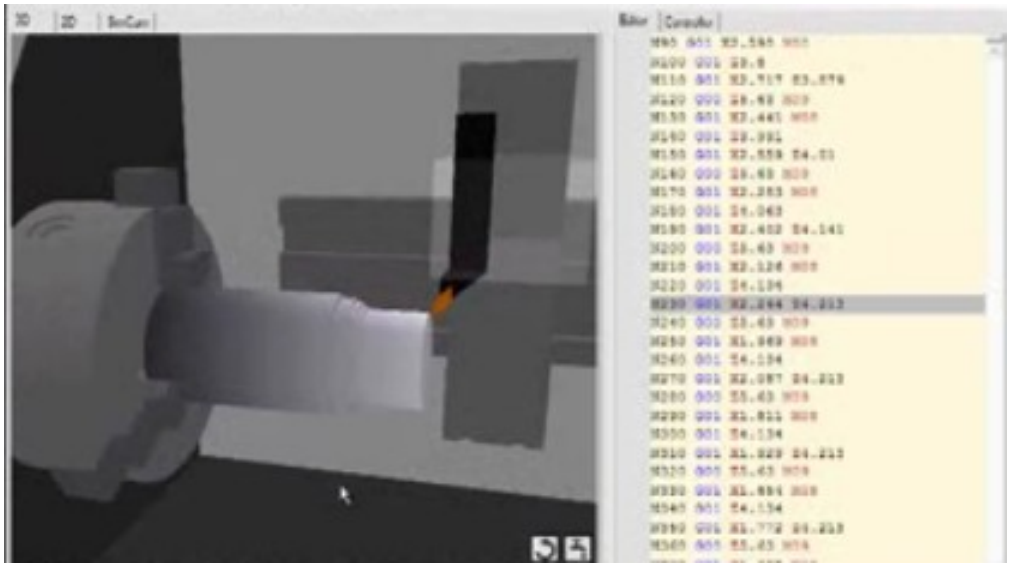

Figura 22. CNC Simulator Pro, ferramenta de corte em destaque (CNCSimulator.com, 2013).

# 3. DEFINIÇÃO DO PROJETO

O projeto consiste no desenvolvimento de um *software* que explora as funcionalidades de um Torno CNC num ambiente de Manufatura Virtual. A utilidade do programa é relevante conforme discutido anteriormente, já que pode servir de auxílio para trabalhos futuros de maior complexidade que venham a desenvolver o tema da RV. Em resumo, as vantagens do projeto incluem aspectos como os citados abaixo:

- Modelagem de um equipamento extremamente versátil e que, portanto, é de enorme utilidade na indústria manufatureira;
- Redução da necessidade de testes em equipamentos reais, diminuindo custos e riscos atrelados a isso;
- Verificação da usinabilidade de peças;
- Utilidade como ferramenta de ensino, facilitando o treinamento e capacitação de pessoas que venham a usar equipamentos reais similares;
- O modelo pode dar base para projetos mais robustos, por exemplo, servindo como elemento para uma cadeia produtiva que envolva diversos tipos de equipamentos ou em maior quantidade;
- Baixo custo do programa, uma vez que é baseado em ferramentas gratuitas como o Microsoft XNA e Blender.

Os requisitos do *software* visam atender as principais funcionalidades básicas de equipamentos atualmente difundidos. Assim, chega-se aos seguintes tópicos:

- **Selecionar Modo:** trata-se da opção que o usuário dispõe entre movimentar a ferramenta com avanço manual, ou segundo instruções do código G a ser importado;
- **Alterar parâmetros de usinagem:** o processo de usinagem no torno é composto por variáveis que caracterizam a movimentação durante o desbaste. É desejável que se tenha acesso a alterar tais parâmetros para adequar a simulação às necessidades reais do processo de manufatura em questão;
- **Carregar código:** ao solicitar essa opção o usuário tem acesso a uma lista de códigos definidos que caracterizem o processo de fabricação de uma dada peça;

- **Verificar sintaxe:** a verificação da sintaxe se refere à necessidade de se garantir que o código presente no arquivo carregado esteja de acordo com o que é esperado segundo as definições do Código G. Garante que não haverá simulação de comandos que eventualmente poderiam comprometer o resultado da mesma;
- **Executar arquivo:** após a verificação de sintaxe do arquivo carregado, deve-se poder executar o arquivo e, portanto, gerar a simulação do processo. Ou seja, é o resultado proposto para esse *software*.
- **Movimentar câmera:** o usuário pode movimentar o ponto de observação dentro do ambiente virtual de forma a satisfazer necessidades relacionadas à visualização do processo de usinagem da peça.

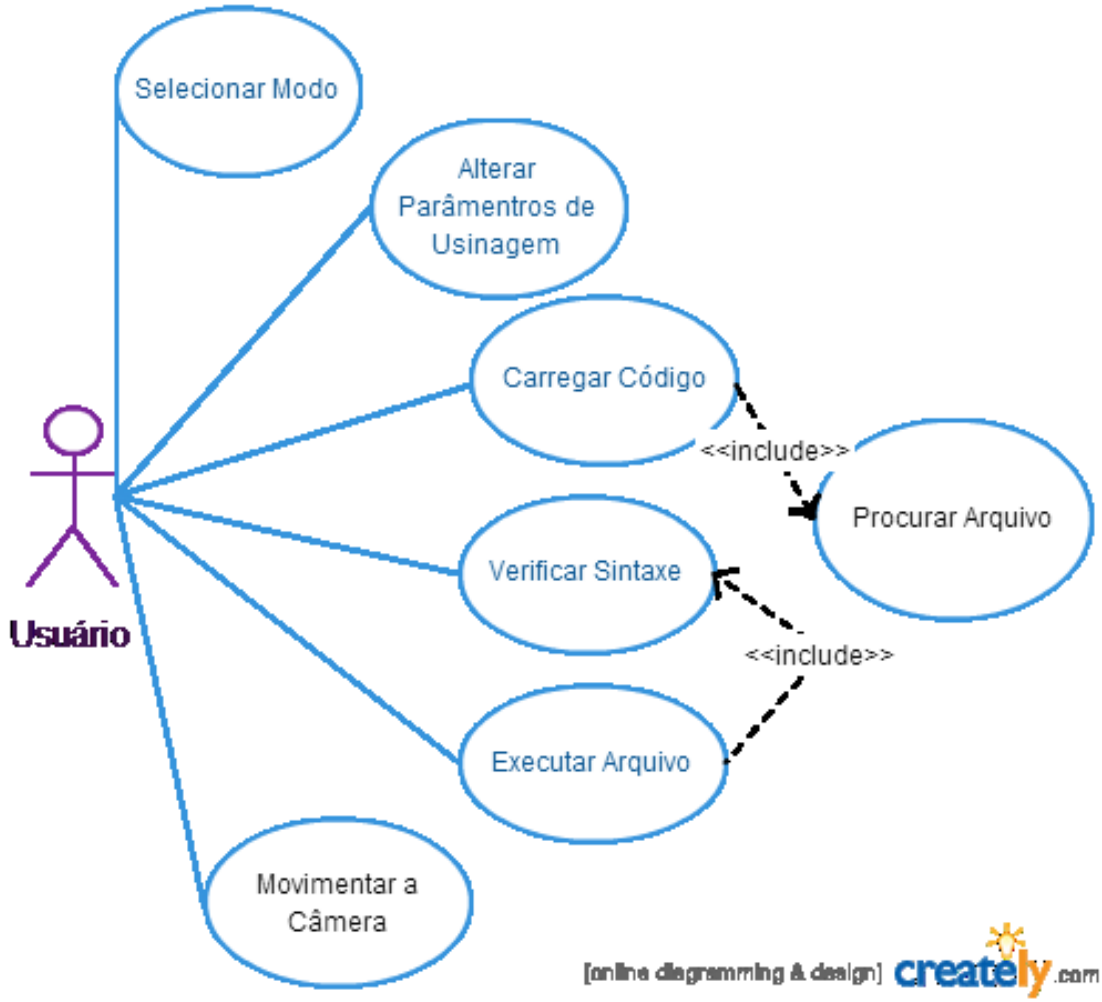


Figura 23. Diagrama de casos de uso.

Na Figura 24, é ilustrado o detalhamento do caso de uso “Selecionar Modo”. Com ela, o usuário pode optar entre movimentar a ferramenta via botões a serem exibidos na interface, ou segundo a execução de um código que contenha as instruções de deslocamento do processo – respectivamente modo manual e modo automático.

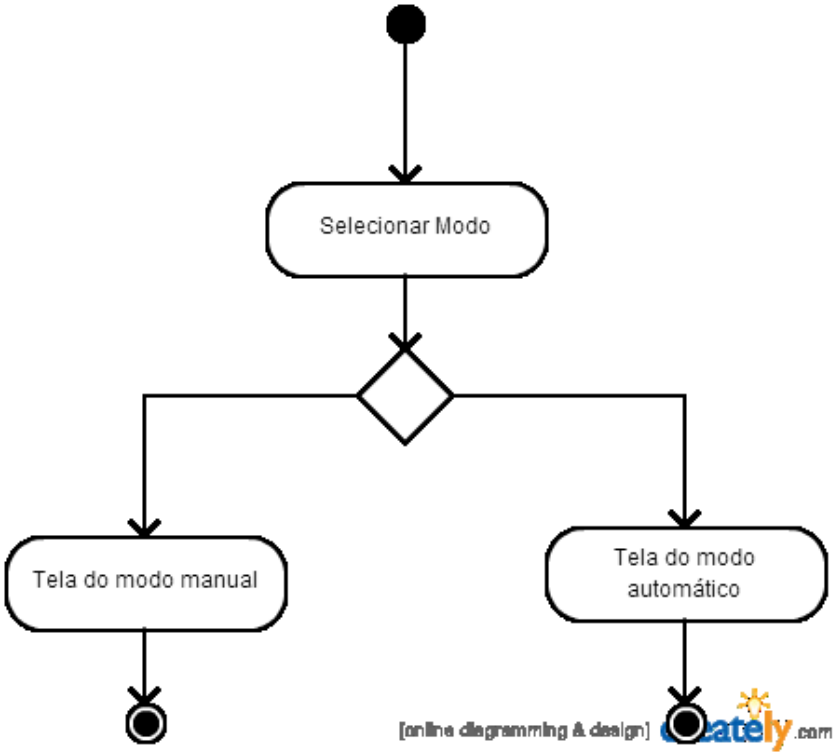


Figura 24. Diagrama de atividades: detalhando o caso de uso Selecionar modo.

Na Figura 25, o detalhamento do caso de uso “Alterar parâmetros de usinagem” ilustra a funcionalidade de o usuário poder alterar os parâmetros do processo de usinagem a ser simulado.

Na Figura 26, evidencia-se o detalhamento do caso de uso “Carregar Código”, i.e., o usuário poderá fazer uso de códigos pré-existentes para simulação.

Na Figura 27, é ilustrado o detalhamento do caso de uso “Executar Código” é ilustrado. Ao selecionar a opção de execução, o *software* deve primeiramente verificar a coerência sintática do mesmo, devendo prosseguir a simulação apenas se essa condição for satisfeita. Para o não atendimento do requisito de coerência sintática, deve-se retornar ao usuário a presença de erro no código.


Figura 25. Diagrama de atividades: detalhando o caso de uso Alterar parâmetros de usinagem.


Figura 26. Diagrama de atividades: detalhando o caso de uso Carregar código.

Na Figura 28, o detalhamento do caso de uso “Verificar Código” pode ser acessado diretamente pelo usuário, que queira verificar a validade do código inserido e/ou modificado antes mesmo de desejar realizar uma execução.

Na Figura 29, o detalhamento do caso de uso “Movimentar Câmera” se refere à possibilidade do usuário navegar no ambiente virtual; dessa forma, podendo ajustar a visualização da cena conforme sua necessidade.

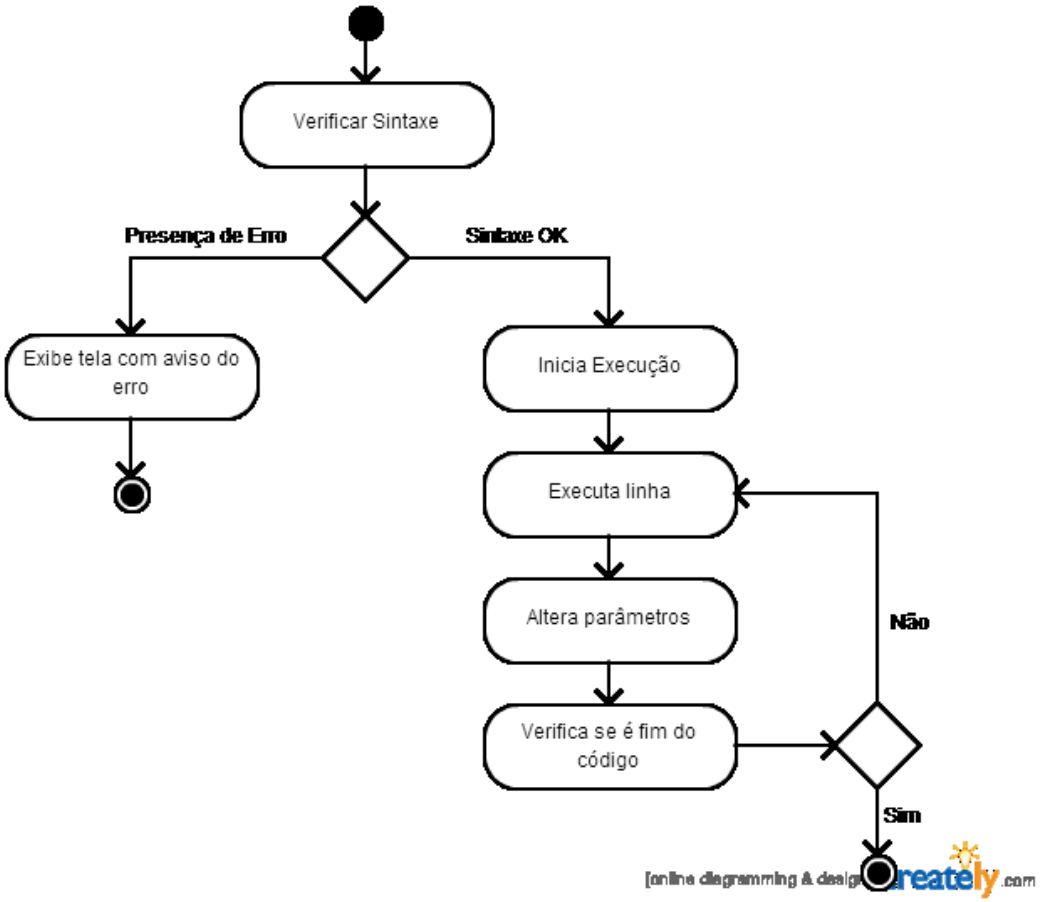


Figura 27. Diagrama de atividades: detalhando o caso de uso Executar código.

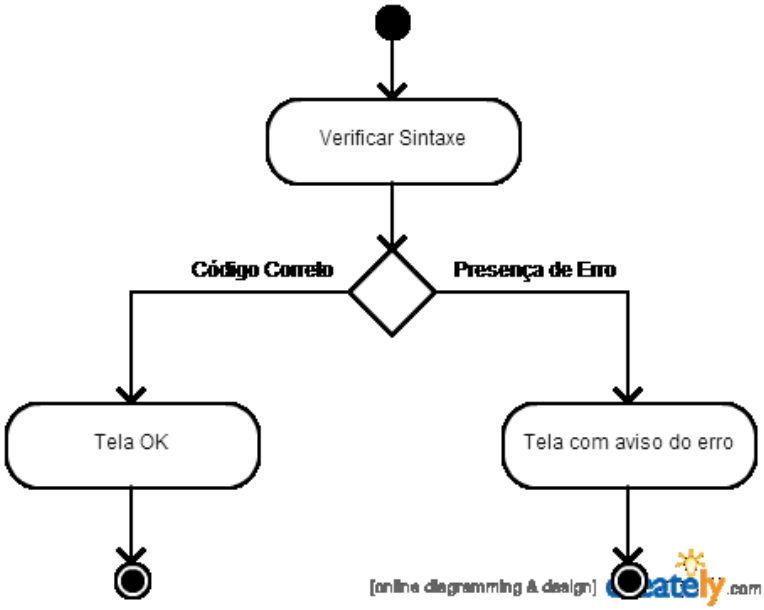


Figura 28. Diagrama de atividades: detalhando o caso de uso Verificar Sintaxe


Figura 29. Diagrama de atividades: detalhamento do caso de uso Movimentar câmera.

Tendo em vista o Diagrama de Casos de Uso da Figura 23, bem como a escrutinização das etapas nos Diagramas de Atividade, elaborou-se um Diagrama de Classes do projeto (Figura 30).

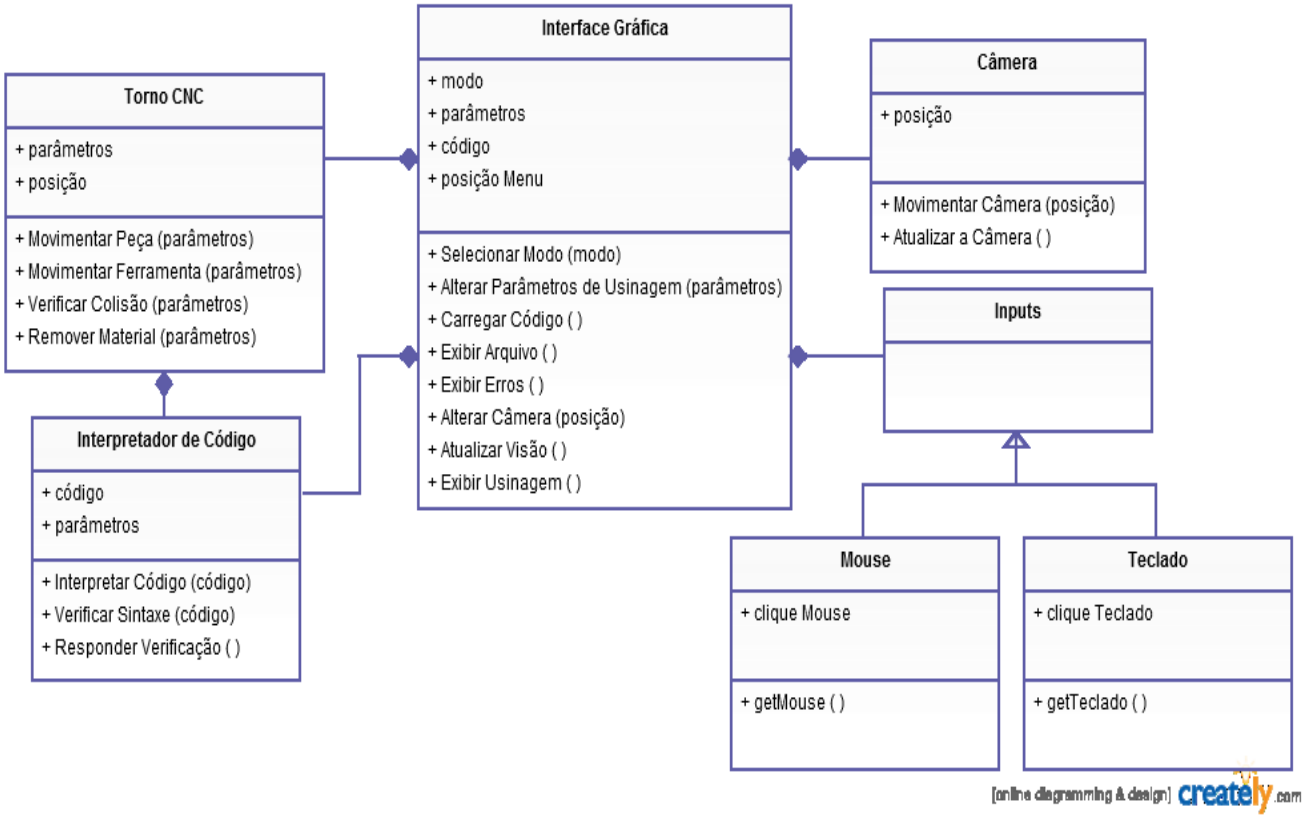


Figura 30. Diagrama de Classes.

Com a conclusão do Diagrama de Classes, criaram se os Diagramas de Sequência, como segue:

Na Figura 31, na interface gráfica (IG), o usuário pode ter acesso à funcionalidade “Selecionar Modo”, que retorna as funcionalidades disponíveis para cada modo.

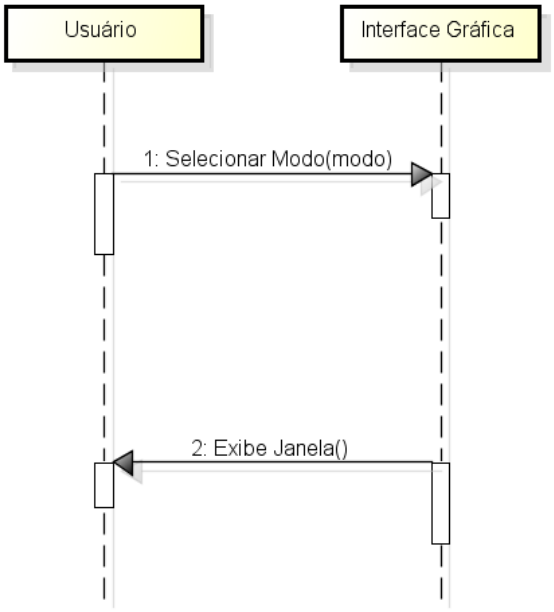


Figura 31. Diagrama de Sequência: “Selecionar Modo”

Na Figura 32, via IG, o usuário seleciona o novo posicionamento da câmera, e a IG manda uma mensagem para Câmera, que atualiza a nova posição.

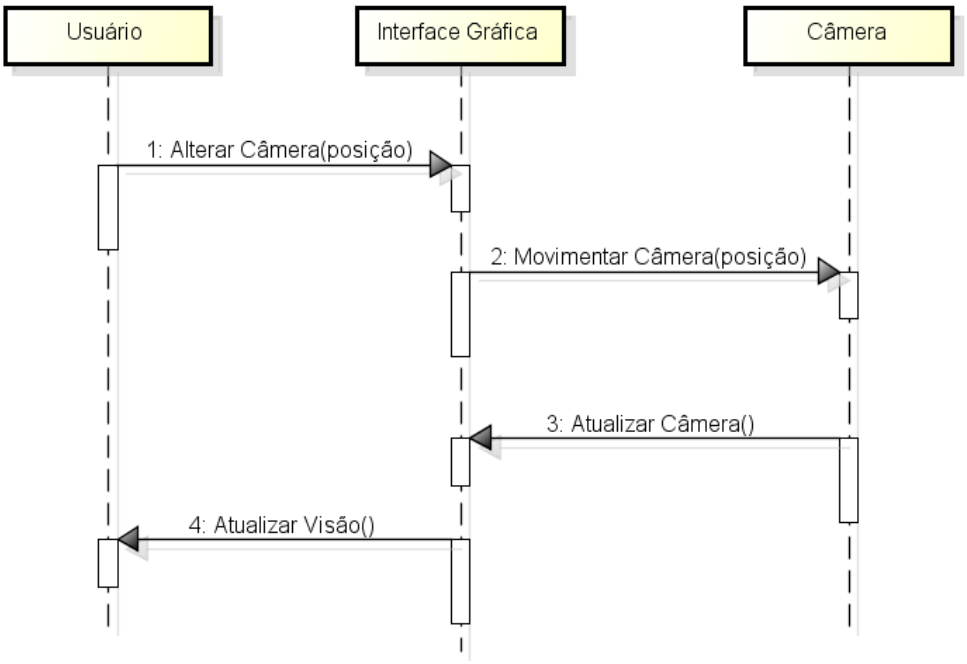


Figura 32. Diagrama de Sequência: “Alterar Câmera”

Na Figura 33 está ilustrado o diagrama de sequência “Carregar Código” e “Verificar Código”. Primeiramente, o usuário carrega o arquivo desejado na IG, ao selecionar a opção de verificação a IG envia a mensagem ao Interpretador de Código, responsável pelo resultado final a ser exibido para o usuário.

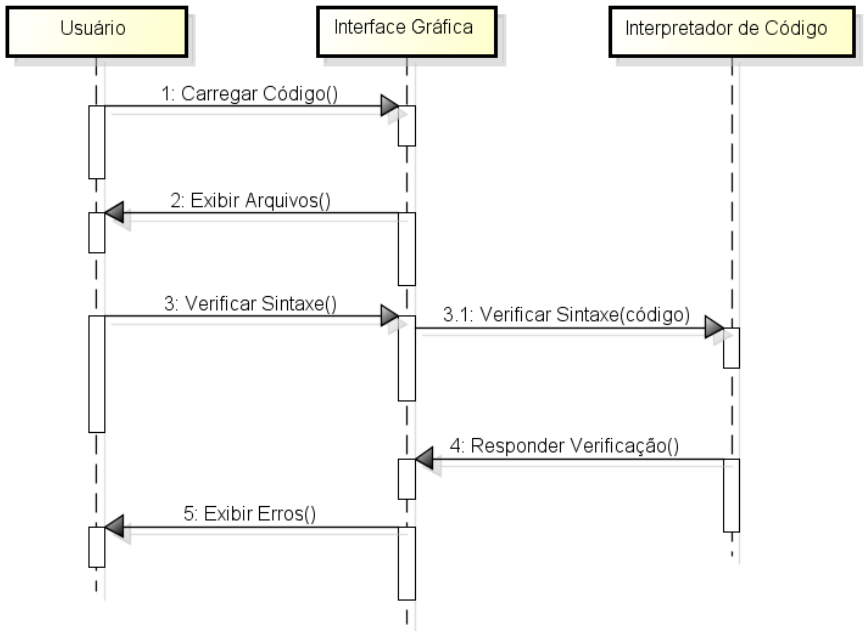


Figura 33. Diagrama de Sequência: “Carregar Código” e “Verificar Sintaxe”

Na Figura 34, está ilustrado o diagrama de sequência referente a “Executar Arquivo”. Por questão de clareza na figura, assume-se que o código desejado já foi selecionado via “Carregar Código”. Então, ao selecionar “Executar Arquivo”, a IG automaticamente envia ao Interpretador a necessidade de se verificar o código em questão. Na presença de alguma incoerência, procede-se com o aviso de erro ao usuário, caso contrário prossegue a execução. O Interpretador passa para Torno CNC as características do processo a ser simulado, por sua vez o Torno CNC envia paralelamente à IG os resultados visuais da simulação do código.

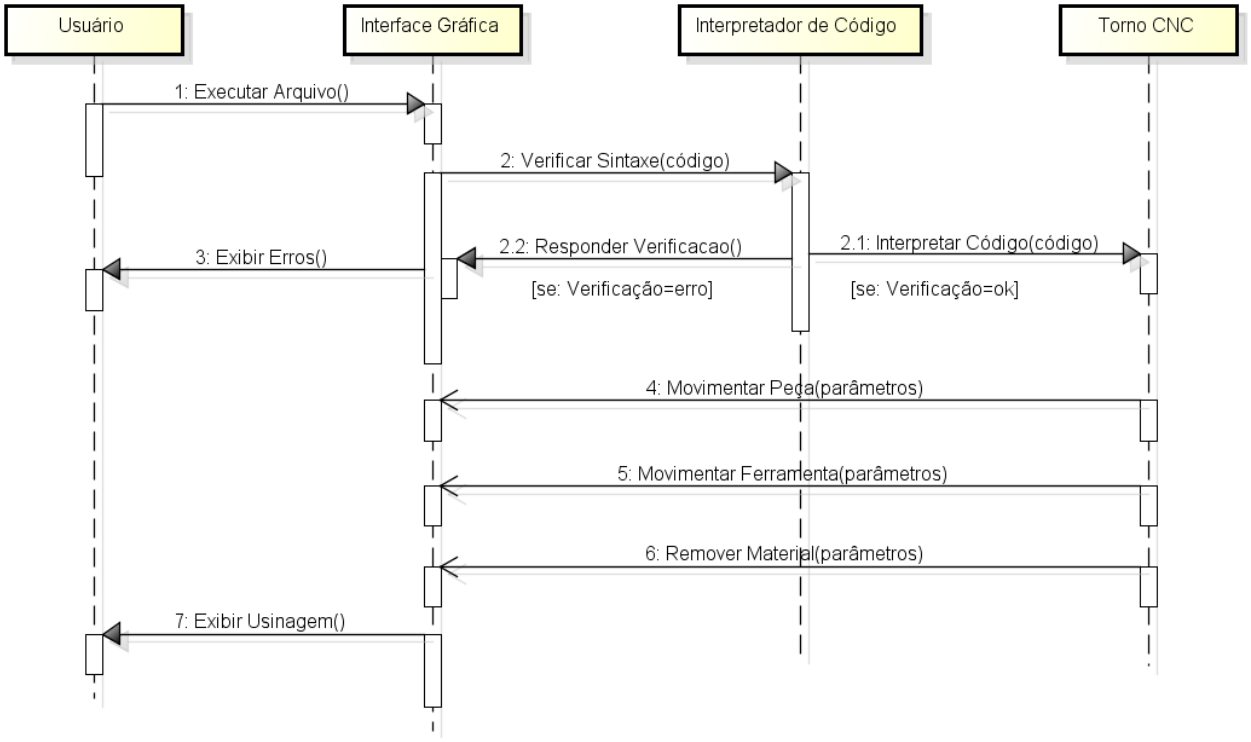


Figura 34. Diagrama de Sequência: “Executar Arquivo”

# 4. IMPLEMENTAÇÃO

## 4.1. Modelagem 3D no Blender

Inicialmente foram modeladas as partes que compõem a estrutura do torno CNC. As partes foram feitas através do desenho do seu perfil em um plano 2D, que depois sofreram extrusão e chegaram ao formato desejado no ambiente 3D. Partes ainda mais simples foram feitas através da inserção de modelos geométricos depois alterados. Foram desenhados a estrutura mais geral que suporta o torno, a placa de três castanhas, a porta deslizante, a ferramenta de corte, o trilho no qual ela se movimenta, bem como seu suporte. (Figura 35).

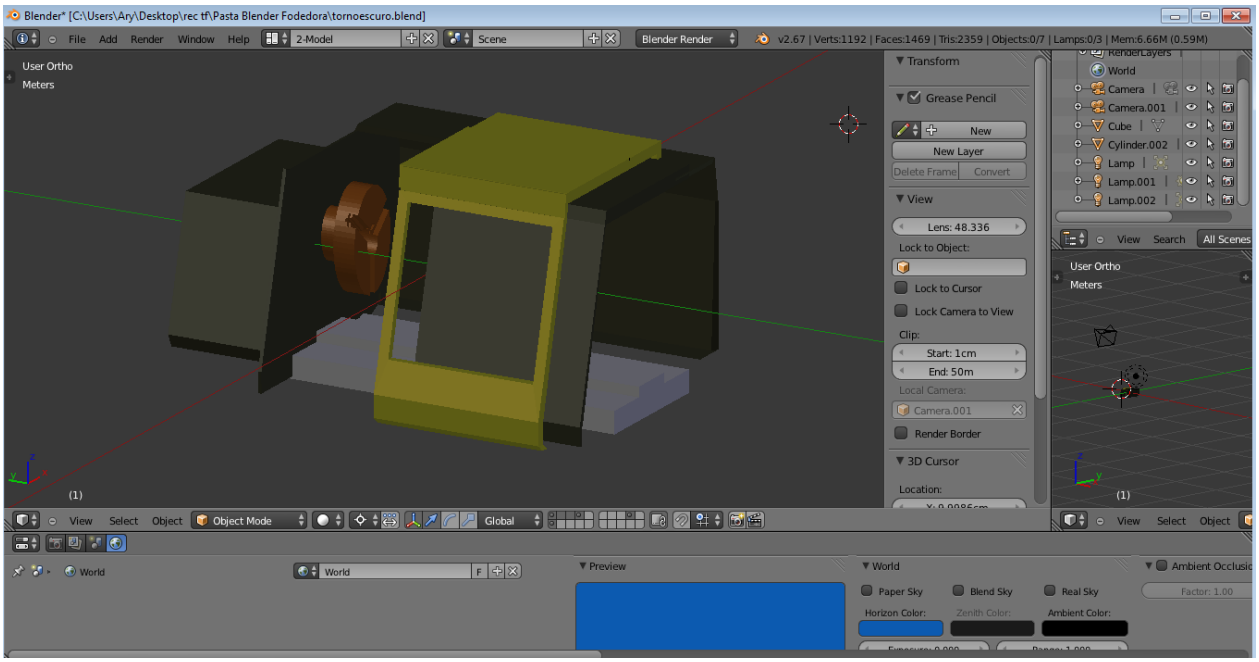


Figura 35. Modelagem 3D de um torno CNC no software Blender.

A seguir, modelos gerados no *software* Blender. Na Figura 36 é apresentada a placa de três castanhas, e na Figura 37 é apresentado o torno CNC.

## 4.2. Importação do modelo 3D para o XNA

Para a importação dos modelos para o XNA, há algumas restrições que devem ser obedecidas: deve-se bloquear a localização e tamanho dos modelos. Para isso, no "Object Mode" do Blender todas as partes modeladas devem ter as variáveis (X,Y,Z) do parâmetro "Location" iguais a 0, do

parâmetro “Rotation” iguais a 0º e do parâmetro “Scale” iguais a 1 (Smith, 2013). Com essas configurações os modelos podem então ser apenas alterados no “Edit Mode” (Figura 37).

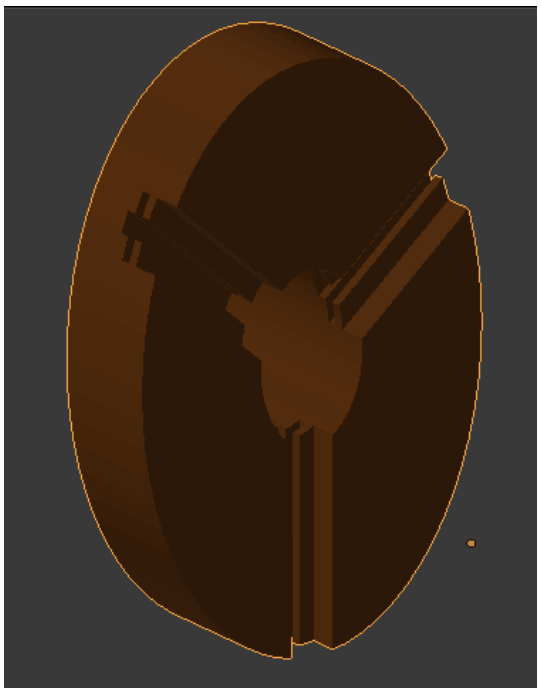

Figura 36. Modelagem de placa de três castanhas.

Essas restrições são necessárias para que ocorra um transporte correto do modelo do Blender para o XNA (os dois programas utilizam sistemas de coordenadas diferentes (Smith, 2013)), utilizando a ferramenta própria do Blender para exportação de modelos em Autodesk FBX, os quais são reconhecidos no ambiente do Microsoft XNA Game Studio, como anteriormente citado na seção 2.4.

## 4.3. Simulação no XNA

**Ambiente Virtual e Malha deformável da peça.**

A Figura 38 ilustra o ambiente do XNA Game Studio. Após aplicações das técnicas apresentadas na seção anterior, o torno foi transportado para um ambiente 3D no XNA (Figura 39), feito através da técnica de criação de uma *skybox* (método de criação de um plano de fundo para fazer um ambiente em um jogo parecer maior do que realmente é). O objeto é inserido dentro de um cubo e imagens são inseridas nas suas faces internas. O objeto pode inclusive estar inserido dentro de uma meia esfera, o que é chamado de “cúpula céu” (Schuld, 2008).

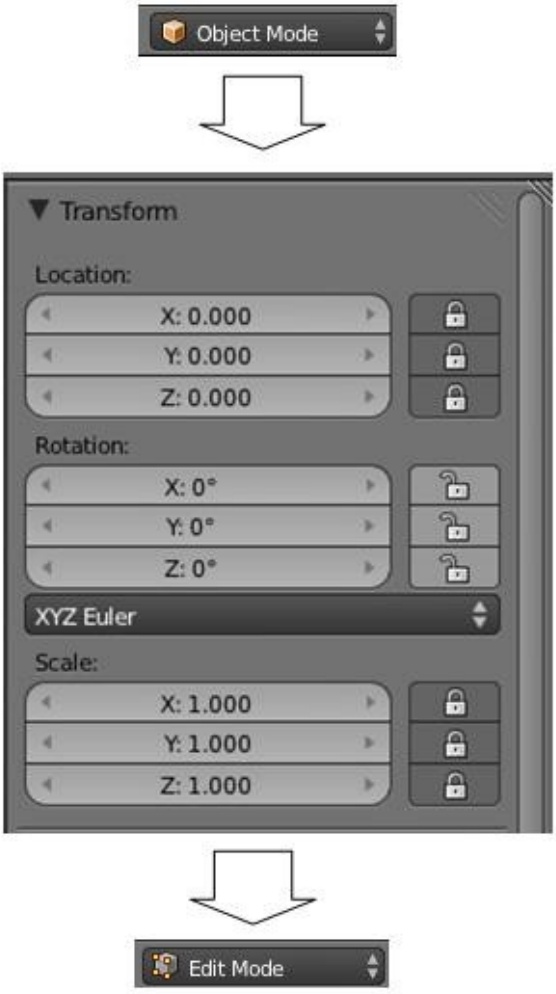


Figura 37. Restrições necessárias para transporte de um modelo Blender para XNA.

Foi utilizada a técnica de *armature* para a movimentação de partes no XNA. Essa técnica consiste na fixação de um elemento na malha de um objeto para deformá-lo e posicioná-lo no ambiente. Uma *armature* foi fixada no modelo do torno para que pudesse ser exportado para o XNA. Outra *armature* foi fixada na placa de três castanhas, para que ela possa rotacionar independentemente do torno.

Para a deformação da peça a ser torneada na simulação, foi utilizado o ceondeito de *mesh's* no XNA e a realização de operações com os mesmos (Microsoft Developer Network, 2013). No entanto, diante da dificuldade na operação com *mesh´s*, optou-se por outra solução. A solução adotada consiste em tratar a peça a ser usinada como um conjunto de pontos independentes no espaço, desenhados de tal forma que constituem o formato da peça. Por simplificação, a peça inicial a ser torneada será apenas cilíndrica.

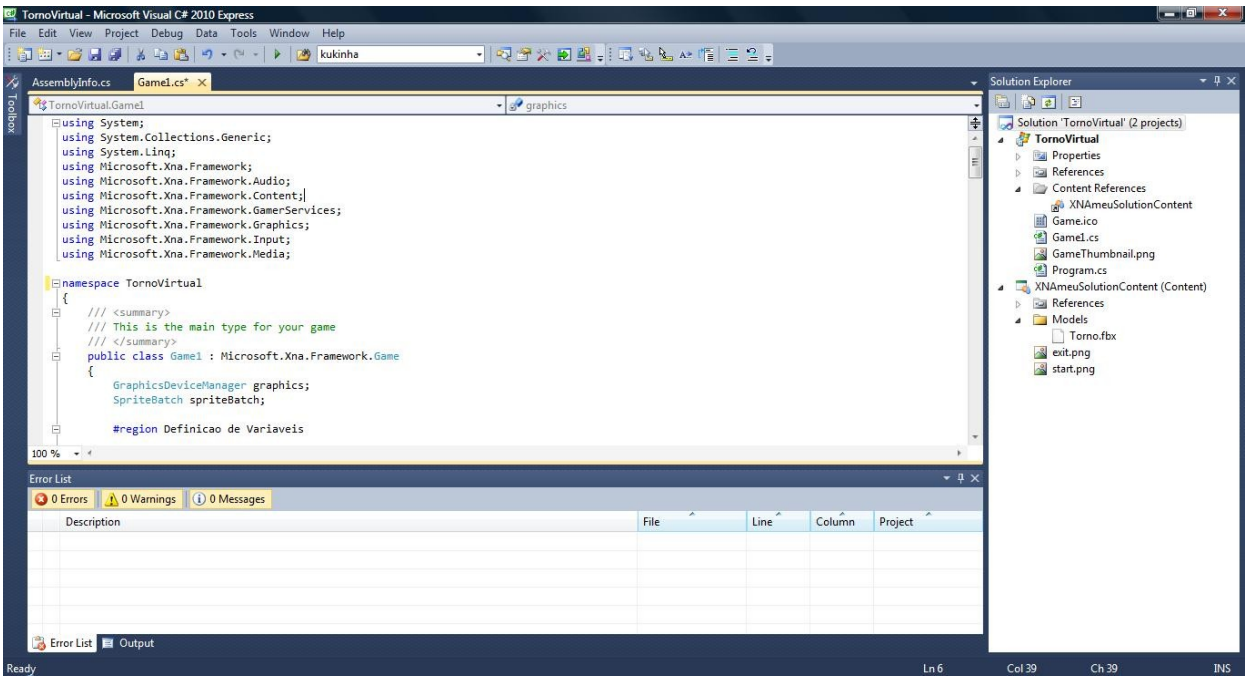


Figura 38. Ambiente do Visual Studio C# Express com o XNA Game Studio 4.0.

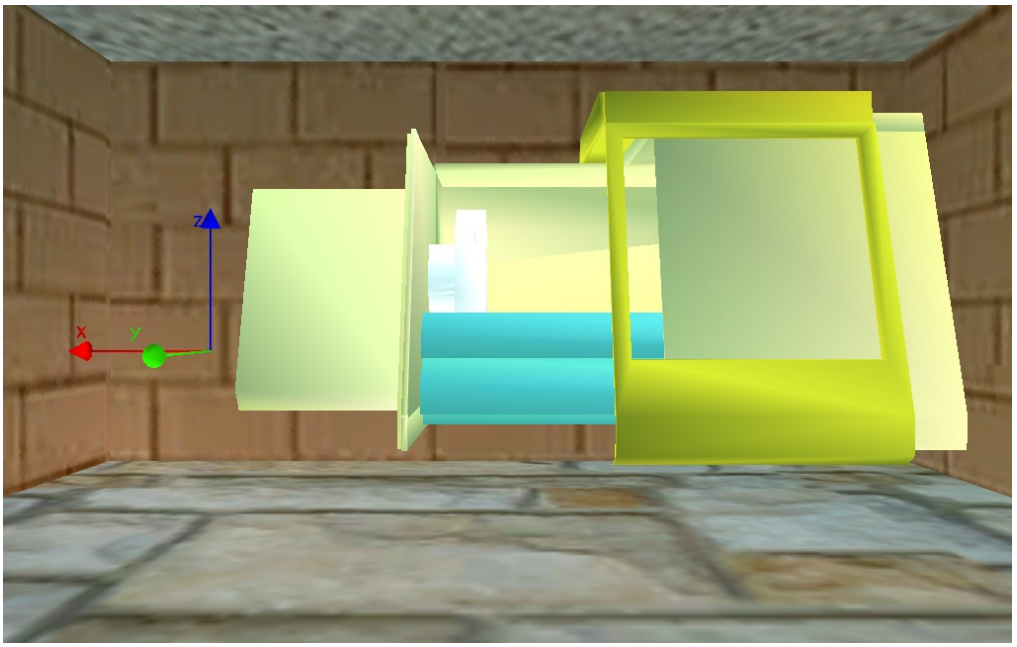


Figura 39. Simulação em XNA de um torno CNC em ambiente 3D.

Após os pontos serem inicializados na forma desejada, o conjunto todo é posto em rotação, através da atualização dos pontos em posições sucessivas várias vezes por segundo, de acordo com a rotação desejada, calculada em rpm.

Nas Figuras 40 e 41 a primeira representação do conjunto de pontos ligados por linhas é apresentado no espaço 3D onde poderá ser visualizado.

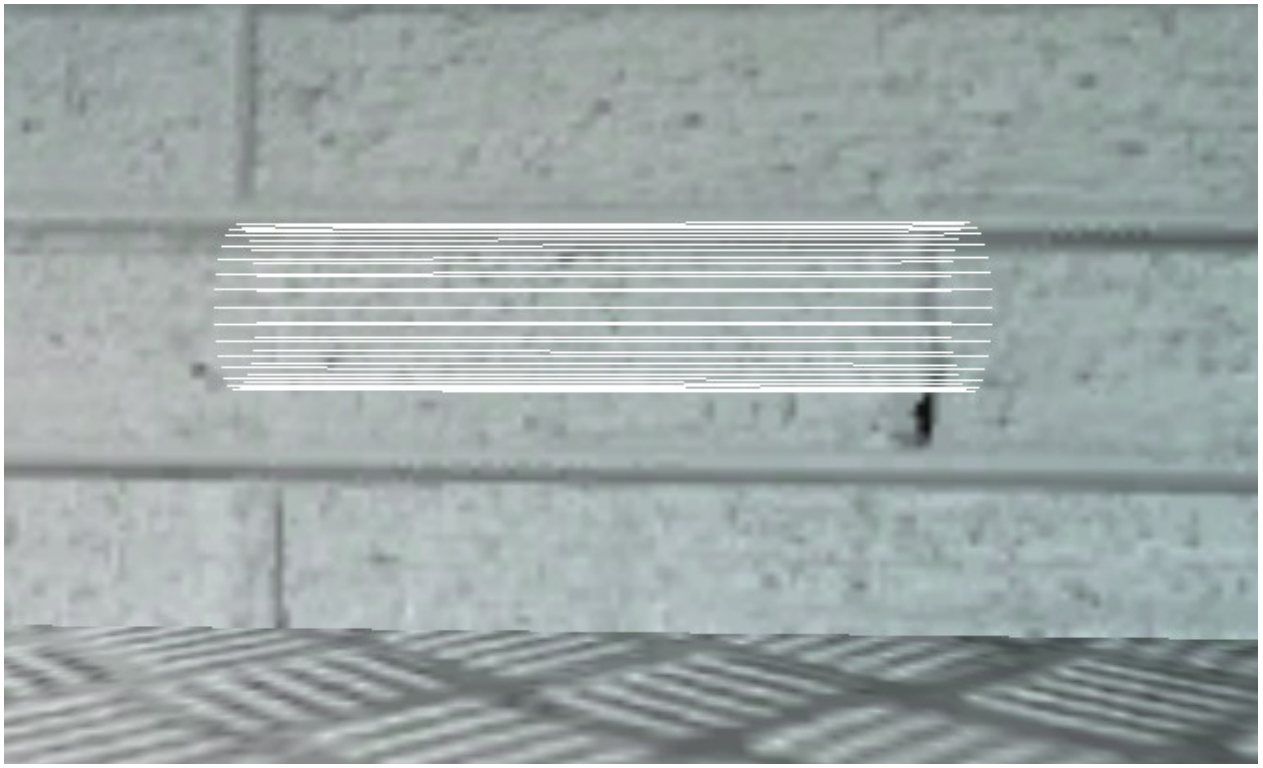

Figura 40. Projeção no espaço 3D dos pontos ligados por linhas que formam a malha da peça a ser usinada

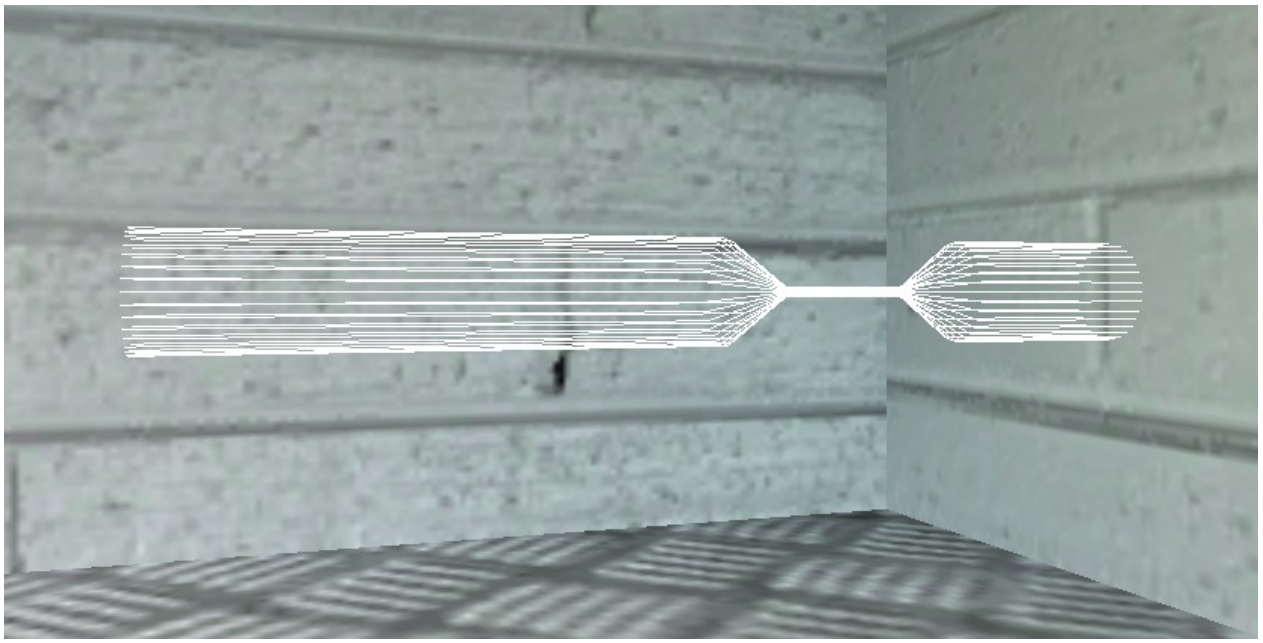

Figura 41. Projeção no espaço 3D dos pontos ligados por linhas que formam a malha da peça a ser usinada (2)

Na Figura 42 é apresentado um desenho esquemático da seção transversal da peça.

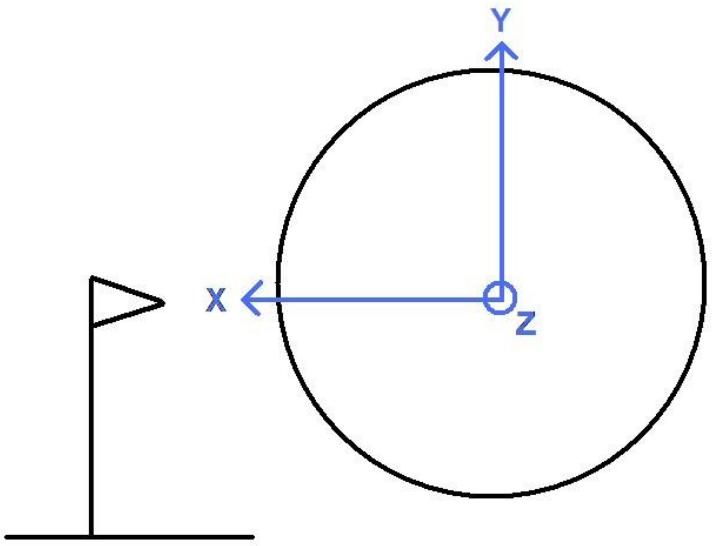


Figura 42. Desenho esquemático da seção transversal da peça, com a representação do sistema de coordenadas e a ferramenta da torno.

Para a deformação da peça, no modo Update() é utilizada uma função de colisão, que sinaliza caso ocorra uma colisão entre a ponta da ferramenta e algum ponto da peça. Havendo colisão, cada ponto é tratado individualmente, e o "raio" em que cada ponto se encontra é atualizado de acordo com a posição da ponta da ferramenta. Para simplificar o método, esse tratamento de "recuo do ponto" é tratado apenas no plano X-Y (onde é atualizado o "raio" do ponto, através de cálculos de geometria, ou seja, a distância do ponto até o centro é atualizada de acordo com a distância da ferramenta), pois a posição Z do conjunto de pontos a ser tratado é "guardada" no momento da colisão (ou seja, caso a ferramenta se aproxime de uma distância crítica *dz*, uma função detecta qual conjunto de pontos possui coordenada Z mais próxima da ferramenta (Figura 43). Em seguida é verificado no plano X-Y se de fato a ponta da ferramenta está em uma posição dentro da circunferência da peça naquele plano. Quando finalmente é declarada uma colisão, inicia-se o algoritmo de atualização do ponto em colisão com a ponta da ferramenta: os pontos são considerados em colisão caso estejam dentro de uma distância crítica *dy* da ponta da ferramenta, Figura 44).

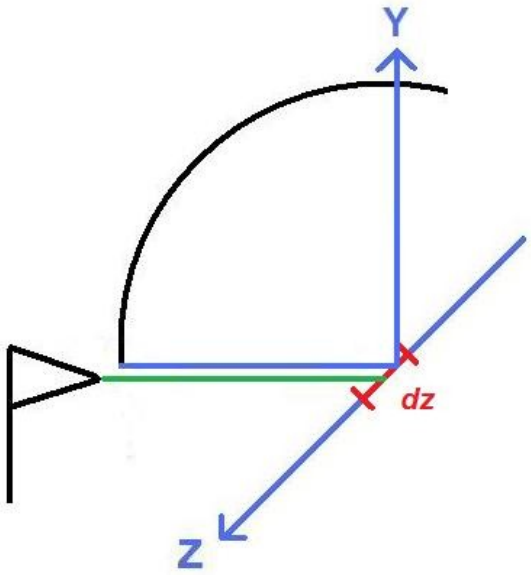


Figura 43. Em vermelho a distância crítica *dz*. a função de colisão guarda o valor da coordenada Z mais próxima que represente uma seção transversal da peça, e passa a atuar apenas no conjunto de pontos dessa seção.

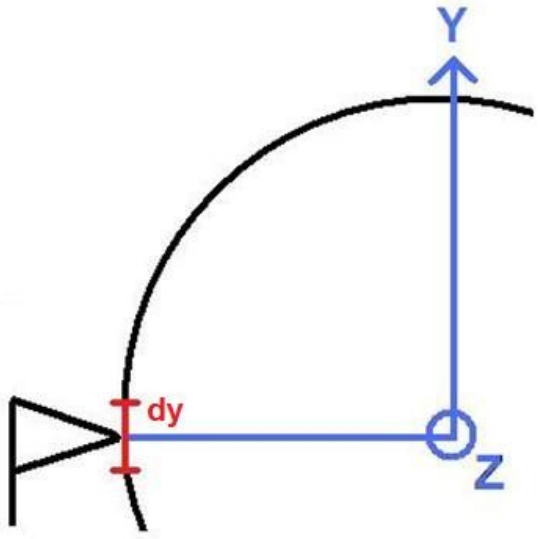


Figura 44. Em vermelho a distância crítica *dy*. Caso um ponto tenha coordenada Y dentro desse intervalo, este ponto é recuado, o que representa uma deformação na peça.

A Figura 45 é uma representação esquemática da deformação da peça.

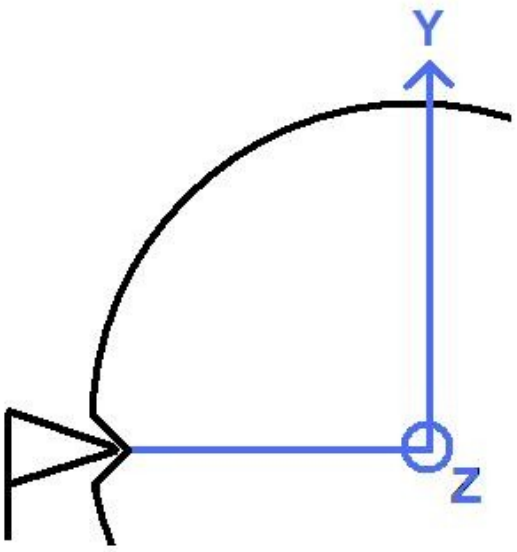


Figura 45. Esquema de deformação da peça.

O planejamento inicial foi feito com uma simplificação na deformação da peça, inicialmente tratando não os pontos individualmente, mas diminuindo todo o raio da peça no plano X onde ocorreu a colisão. Essa solução encontra limitações à medida que não permite usinar peças como roscas.

A solução final foi feita a partir de uma extensão da solução inicial, definindo não um raio para cada seção transversal da peça, mas sim um "raio" para cada ponto, ou seja, a distância de cada ponto do centro da seção transversal é tratada individualmente. Dessa forma, durante a colisão, os pontos são recuados de acordo com a aproximação da ponta da ferramenta. Um exemplo pode ser visto na Figura 46, uma peça arbitrária obtida com a utilização do programa.

Com essa solução, é possível desenhar os mais variados tipos de peças, inclusive roscas, como pode ser observada na peça em formato de rosca que foi obtida utilizando o programa, Figura 47.

**Textura da Peça**

Para o desenho da textura da peça, foi utilizada uma função do XNA para desenho de texturas através da declaração dos seus vértices (*VertexPositionTexture.VertexDeclaration*). Um arquivo com os *pixels* da textura é escolhido, com dimensões iguais e número de *pixels* em potências de dois. Por exemplo, neste trabalho foi utilizada uma figura em formato JPEG com dimensões de

64x64 pixels. A imagem foi aplicada em cada quadrante formado por quatro pontos adjacentes da peça, Figura 48.

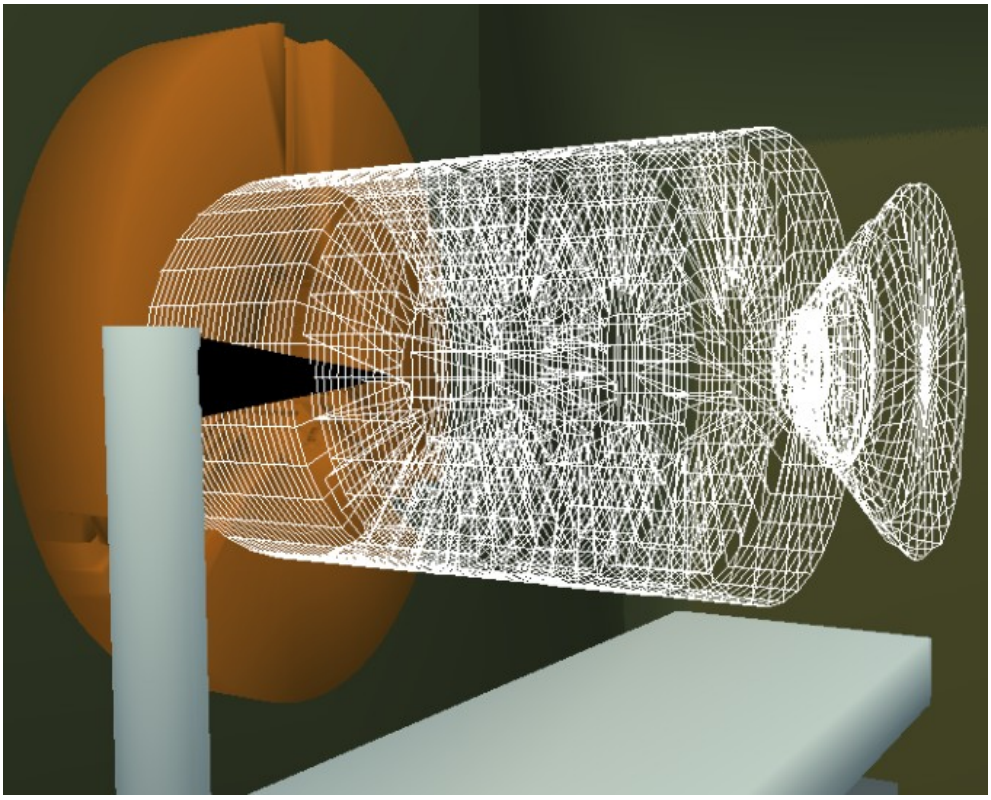

Figura 46. Peça arbitrária obtida no ambiente de simulação do XNA.

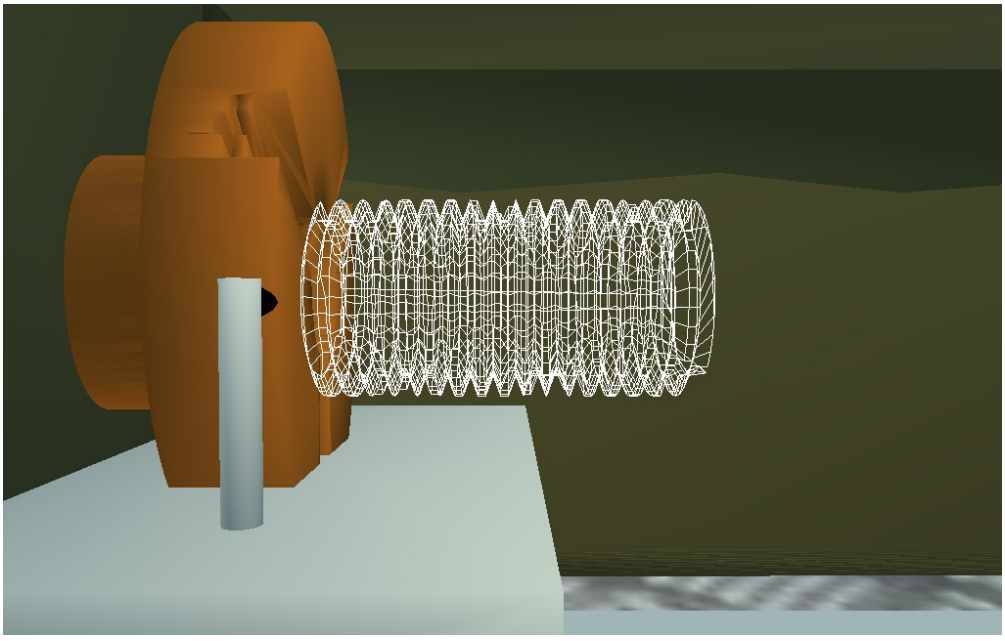

Figura 47. Peça deformada para obtenção de uma peça rosqueada.

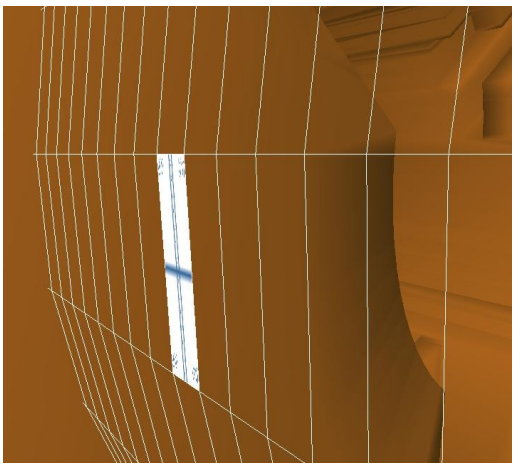

Figura 48. Exemplo de textura aplicada em um quadrante da peça.

Os vértices da textura são guardados em uma variável própria chamada *VertexPositionTexture*. Para aplicação da textura em um quadrante, é necessária a declaração de seis vértices, pois a função aplica a textura em triângulos (*PrimitiveType.TriangleList*), sendo então necessário dois triângulos para a aplicação da textura em cada quadrante. O desenho, no entanto, não é alterado, pois a função divide o desenho em dois, assim cada triângulo fica com metade dos pixels.

Os vértices das texturas são calculados de acordo com a posição dos pontos a cada atualização do programa, de forma que a textura então é corretamente desenhada mesmo com a peça em rotação.

No topo da peça, entretanto, a textura foi aplicada em triângulos, formada pelo centro da peça e dois pontos adjacentes na circunferência externa, dessa forma apenas metade dos pixels da imagem da textura foi aplicada, Figura 49.

Mesmo após a aplicação da textura, o modo de visualização em *wireframe* contínua sendo uma opção do programa. A Figura 50 exemplifica uma peça no formato “sólido”.

**Interface com o Usuário**

O usuário pode acessar as diversas funcionalidades do programa por duas vias: o *mouse* e o teclado. Enquanto o primeiro é mais limitado sendo possível apenas interagir com os recursos da interface dispostos na tela, o teclado permite a interação completa com o Ambiente Virtual. A seguir, exemplos da interface criada.

Figura 49. Textura aplicada no topo da peça.

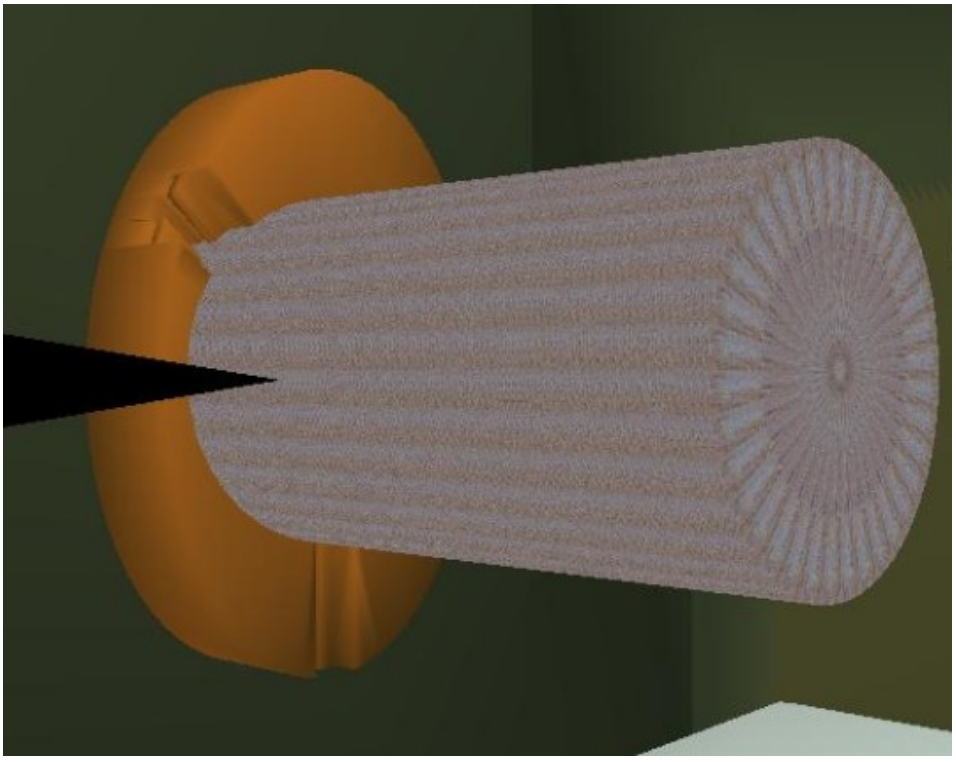

Figura 50. Peça em visualização no formato sólido: textura aplicada em toda a área externa.

A tela inicial do programa, Figura 51, onde é informado em destaque que se trata de um simulador gratuito, o que é um objetivo do projeto.


Figura 51. Aparência da interface inicial com o usuário.

Após a seleção da opção "ENTRAR" a simulação é iniciada, com a apresentação do ambiente virtual. O programa inicializa no modo manual, com uma interface do modo manual no canto superior direito com as opções de movimento da ferramenta (setas), mudança de velocidade da peça e da ferramenta, a opção de ligar e desligar o torno, a opção de alterar a visualização entre *wireframe* e "sólido", e mudança entre modo manual e automático, que são acessíveis pelo clique do *mouse*. A ferramenta possui a possibilidade de movimentação (translação) apenas em dois eixos, que no programa correspondem aos eixos Z e X, ou seja, ao longo do eixo da peça e perpendicularmente a esse eixo. A imagem do início da simulação é apresentada na Figura 52, com todos os componentes do ambiente virtual. Na Figura 53, é apresentado o menu no modo automático com o código G previamente inserido no programa. As coordenadas utilizadas no código G são em milímetros, de forma que a peça inicial possui comprimento de 500 mm e raio de 130 mm. O centro do sistema de coordenadas é a extremidade da peça em contato com a placa de três castanhas, que também corresponde ao centro dessa placa.

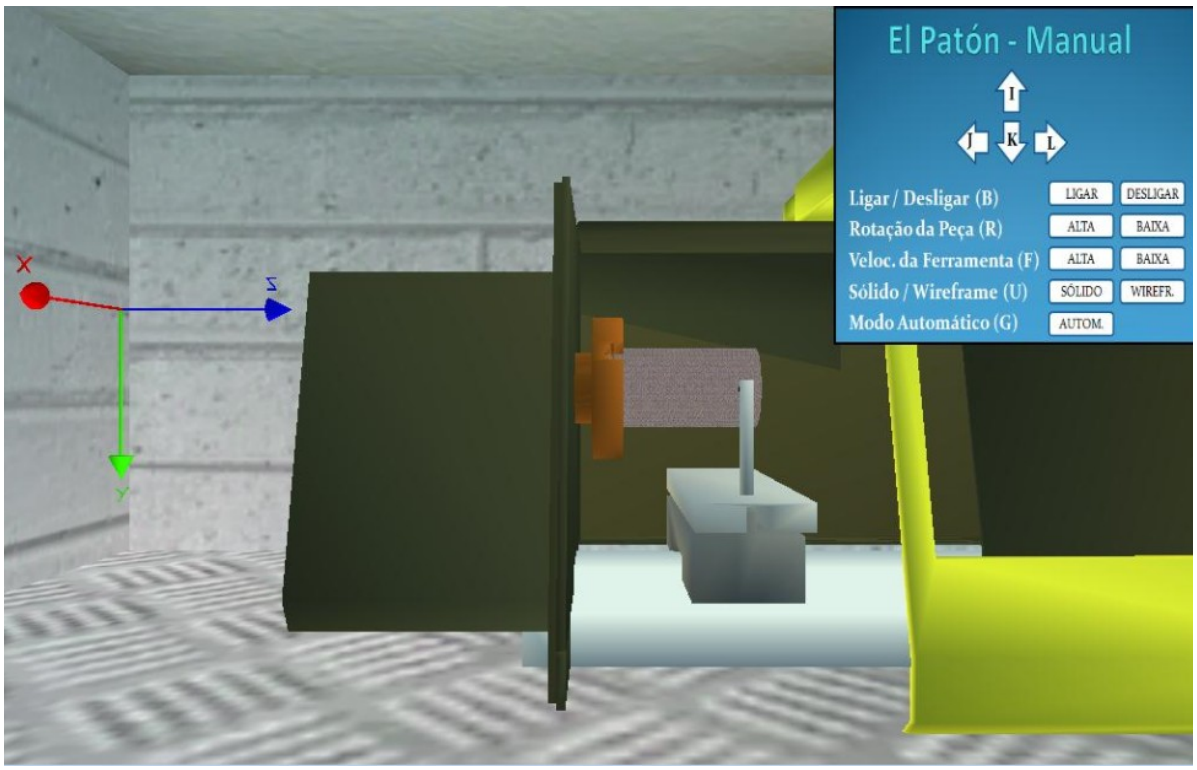


Figura 52. Aspecto inicial de uma simulação com a peça ainda sem nenhuma deformação.

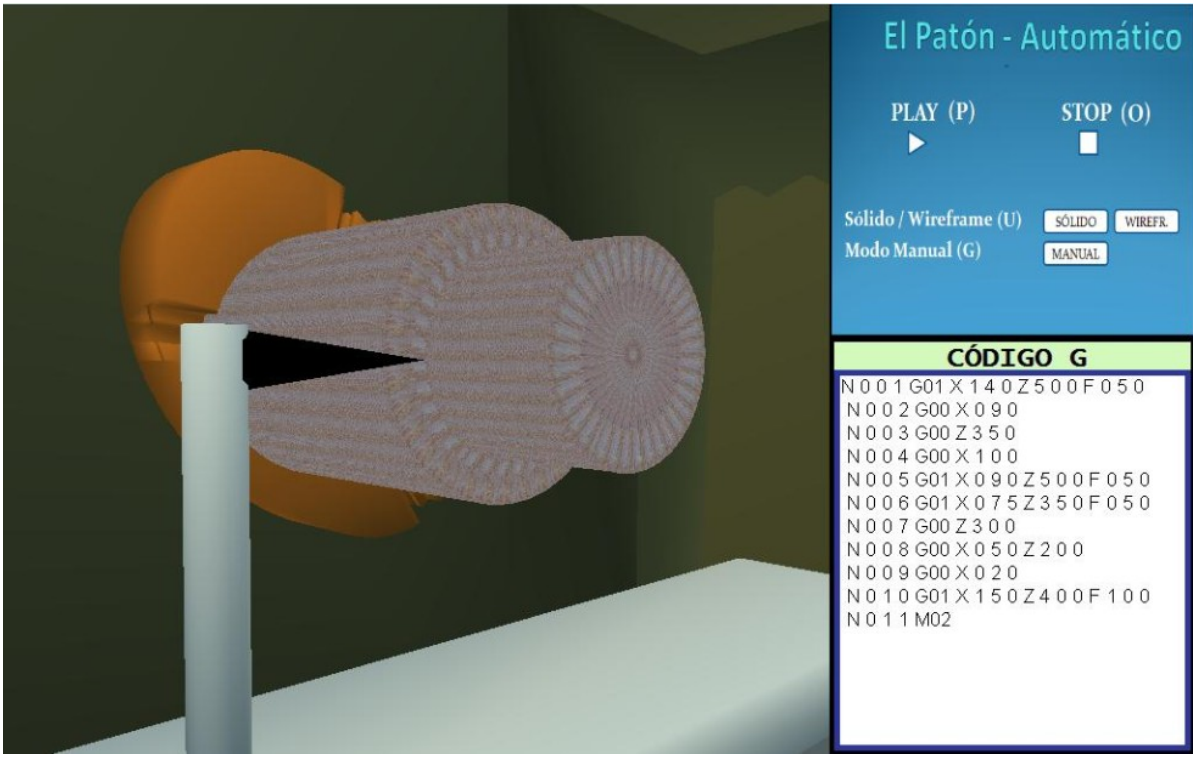


Figura 53. Interface do interpretador de linguagem G, coordenadas em milímetros. Detalhe da peça deformada e com textura.

Os demais recursos podem ser acessados pelo teclado e são, em resumo:

- **W/A/S/D**: Teclas responsáveis pela navegação da câmera no ambiente virtual.
- **Setas**: Ajuste da orientação da visão da câmera.
- **U**: Troca de visualização da peça entre *wireframe* ou peça sólida.
- **P**: Habilita a rotação da peça.
- **R**: Troca a velocidade de rotação da peça.
- **I/J/K/L:** Movimentação manual da ferramenta.
- **F**: Troca a velocidade de avanço da ferramenta.
- **T:** Reconstitui a peça no formato inicial, caso tenha sido deformada.
- **G:** Aciona o modo automático.
- **ESC:** Sai da simulação.

**Interpretador de Linguagem**

O ANTLR é um *software* de uso livre, estando disponível na plataforma virtual da instituição. Para os fins desse trabalho, empregou-se a versão ANTLRWorks 1-43. Dele foram geradas duas classes em C# - o Parser e o Lexer – a importância dessas já foi explorada em Item 2.4. Além dessas, foi criada manualmente uma terceira classe (Manager) para a criação da árvore sintática do código, de forma a permitir a leitura de um arquivo externo em *.txt* que contém os comandos em G a serem simulados.

O código G possui inúmeras opções de comando para as mais diversas funcionalidades presentes num equipamento real. Nesse projeto, foram implementadas apenas funções básicas de movimentação de um Torno CNC. A dinâmica de alguma dessas funções será discutida brevemente a seguir (maior detalhamento pode ser encontrado no próprio programa disponível ao final desse relatório no Anexo).

A) **Dinâmica geral do Interpretador.** Cada comando em G presente no arquivo *.txt,* é compreendida como um nó da árvore sintática (as palavras definidas no Lexer). Assim, a simulação consiste em percorrer sequencialmente esses nós. Para tanto, foi criado um enca-

deamento de *switch/case,* responsável por armazenar as regras de movimentação para cada palavra do código – ao final dessa execução incrementa-se o nó sucessivamente até o último ou comando de parada.

B) **Movimentação da ferramenta – "Posicionamento Rápido" e "Interpolação Linear".** Consiste nas movimentações básicas da ferramenta do Torno nos seus dois graus de liberdade. Para movimentos em apenas um dos eixos, a interpretação é relativamente simples. Como é sabida a posição dos objetos na cena, basta armazenar as coordenadas do ponto final desejado e incrementar a posição atual de um determinado valor periodicamente até que estas sejam idênticas.

   A movimentação concomitante nos dois eixos é ligeiramente mais trabalhosa. Antes da movimentação do objeto, deve-se reconhecer em qual dos eixos haverá o maior deslocamento. Então, calcula-se a razão entre as distâncias, que permite a atribuição de um incremento menor para a posição que sofrerá menor variação.

C) **Rosqueamento.** A possibilidade de construção de roscas é de grande importância para os equipamentos de torneamento. Para sua execução, além das coordenadas do ponto final desejado, deve ser armazenado o passo da rosca. Com essas informações, pode-se calcular a velocidade de movimentação da ferramenta para a qual, dada uma velocidade angular da peça, a rosca é executada. Finalmente, deve-se observar: em equipamentos reais, nos casos em que a altura do filete da rosca é relevante, o processo de rosqueamento deve ser efetuado em mais de uma passagem da ferramenta ao longo da peça. Isso ocorre, pois a retirada muito profunda de cavaco de uma só vez pode prejudicar a qualidade da peça. No entanto, esse requisito não é abrangido pelo projeto implementado.

# 5. CONCLUSÕES

A manufatura virtual é uma tecnologia relativamente nova, mas será determinante em processos fabris no futuro. O seu custo-benefício é cada vez maior, haja vista o crescente poder de processamento dos computadores e seu menor custo. Dessa forma, conquanto um programa inteiramente livre, o projeto desenvolvido tem grande potencial de atender as necessidades de uma planta fabril de menor porte e/ou requisitos para treinamento de pessoal.

O ambiente virtual criado simula de maneira abrangente o funcionamento de um Torno CNC, além de dispor de uma interface bastante simples para sua utilização. Isso permite, por exemplo, o melhor entendimento de um dado processo de manufatura sem a necessidade de se interromper as atividades de chão de fábrica, o que implica economia de tempo e diminuição de riscos.

O projeto tem como principais vantagens, além de ser gratuito, a possibilidade de execução de um Código G (amplamente usado em sistemas reais) e a simulação da deformação da peça de forma eficiente e simples. Finalmente, ressalta-se a possibilidade de seu emprego futuro como um módulo de projetos mais complexos e abrangentes – dentro do escopo da Fábrica Virtual – ou até mesmo de maiores aperfeiçoamentos do próprio sistema.

## 5.1 Futuros Desenvolvimentos

Apesar de os resultados finais obtidos estarem em linha com o inicialmente proposto, existem diversos espaços para melhorias. Dentre elas, enxergam-se duas oportunidades para essas melhorias, a saber: (i) refinar os resultados do projeto, e (ii) recursos para integrá-lo a sistemas mais robustos. Abaixo apenas será discutido temas referentes ao primeiro escopo; contudo, vale ressaltar que a eventual integração do presente projeto a sistemas mais completos que possam, por exemplo, simular o funcionamento de toda uma planta fabril, podem apresentar enorme potencial.

Assim, algumas das necessidades mais imediatas para refinar o funcionamento desse projeto são:

- Incorporar um número mais abrangente de funcionalidades de código G como, por exemplo, a interpolação circular;
- Desenvolvimento de um cronômetro para mensuração do tempo que é gasto no processo;

- Criação da funcionalidade “Verificar Código” independente da sua simulação;
- Para casos de dentes de rosca muito altos, a programação de mais passos da ferramenta ao longo da peça;
- Desenvolvimento de modelos em CAD mais realísticos;
- Inclusão de efeitos sonoros mais realísticos, o que permitiria uma maior imersão psicológica ao usuário;
- Incluir sombra na simulação, o que deve proporcionar uma melhor visualização da peça no modo “sólido”.

# 6. REFERÊNCIAS BIBLIOGRÁFICAS


ARTHAYA, B.; SETIAWAN, A.; SUNARDI, S. The Design and Development of G-Code Checker and Cutting Simulator for CNC Turning. *Journal of Advanced Manufacturing Systems*, Vol. 10, No. 2 (2010) 261–276.

AUTODESK. *3ds Max*, 2013. Disponível em <http://www.autodesk.com/products/autodesk-3ds-max/overview>, último acesso em 2 ago 2013.

BLENDER. *Art Gallery*, 2013. Disponível em <http://www.blender.org/features-gallery/gallery/art-gallery/>, último acesso em 25 jul 2013.

BURDEA, G.; COIFFET, P. *Virtual Reality techonology*. 2ed. Hoboken. 2003.

CECIL, J.; KANCHNAPIBOON, A.; *Virtual engineering approaches in product and process design",* International Journal of Advanced Manufacturing Technology, 2007.

CHENG, K; BATEMAN, R; *e-Manufacturing: Characteristics, applications and potentials.* Brunnel University, 2008.

CHENG, Y.; WANG, S; *Applying a 3D virtual learning environment to facilitate sutdent's application ability – The cas of marketing.* Journal of Computers in Human Behavior, 2010.

CHRONISTER, J. *Blender Basics*, Classroom Tutorial Book, 4th Edition, 2011.

CHRYSSOLOURIS, G.; MAVRIKIOS, D.; FRAGOS, D.; KARABATSOU, V. *A virtual reality based experimentation for the verification of human related factors in assembly processes.* Robotics and Computer Integrated Manufacturing, 2000.

CNCSimulator.com. *CNCSimulator Pro*. Disponível em <http://cncsimulator.info/>, último acesso em 10 jul 2013.

DÉPINCÉ, P.; CHABLAT D.; WOELK, P. *VIRTUAL MANUFACTURING Tools for improving Design and Production*. Thematic network on Manufacturing Technologies (MANTYS).

FALCK, D.; *G-Code Programming Basics,* 2008.

GUTIERREZ, M; VEXO, F; THALMANN, D.; *Stepping into Virtual Reality.* Londres, 2008.

INNOCENT, T. Blending Realities in Game Space. *ACM Computers in Entertainment*, Vol. 6, No. 3, Article 35, October 2008.

KESAVADAS, T.; ERNZER, M.; *Design of virtual factory using cell formation methodologies.* ASME, Material Handling Division, 1999.

KHAN, W. A.; CHENG, A. R. K. *Virtual Manufacturing*, Springer-Verlag London Limited 2011.

KIM, Y.; YANG, J.; HAN, S.; *A multichannel visualization module for virtual manufacturing.* 2006.

KIRKLEY, J; *Using Virtual Reality to model a nuclear components factory in four dimmensions*, Digital Manufacturing Report, 2012.

KJELLBERG, T.; EULER, A; HEDLIN, M.; LUNDGREN, M.; SIVARD, G.; CHEN, D. *The machine tool model – A core part of the digital factory.* CIRP Annals – Manufacturing Technology, 2009.

LEE, J; HAN, S; YANG, J. *Construction of a computer-simulated mixed reality environment for virtual factory layout planning.* Journal of Computers in Industry, 2010.

LISBOA, H. B.; SANTOS, L. A. R. O. *Movimentação de robô virtual por meio de Sensores de som e Movimento*. Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos, 2012.

MASTELARI, N. *Desenvolvimento de um Editor/Simulador para Centro de Torneamento CN.* Universidade Estadual de Campinas, 1996.

MARCICANO, J. *Introdução ao Controle Numérico.* (Apostila). Universidade de São Paulo, 2001.

MICROSOFT DEVELOPER NETWORK. *Mesh Builder Class*, 2013. Disponível em <http://msdn.microsoft.com/en-us/library/microsoft.xna.framework.content.pipeline.graphics.meshbuilder%28v=xnagamestudio.10%29.aspx>, último acesso em 30 jul 2013.

MICROSOFT. *Tutorial sobre XNA – Parte 1 – Primeiros Passos*, 2013. Disponível em <http://msdn.microsoft.com/pt-br/library/hh416748.aspx>, último acesso em 17 jul 2013.

MILOJEVIC, Z.; NAVALUSIC, S.; ZELJKOVIC, M. *Virtual Design and Manufacturing*, Faculty of Technical Sciences, Novi Sad, May 18th 2008.

MIYASHIRO, E. R; SUGAWARA, K. J. *Simulação de um robô virtual utilizando manufatura virtual*. Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos, 2011.

NETTO, A. V.; DE OLIVEIRA, M. C. F. *Procedimento para Prototipação Virtual de um torno CNC*, Instituto de Ciências Matemáticas e de Computação – ICMC, Universidade de São Paulo – USP.

NISHIZAWA, G. Y.; *Impressora de circuitos usando Visão Computacinal*. Universidade de São Paulo, 2010.

PARR, T.; FISHER, K.; *LL: The Foundation of the ANTLR Parser Generator*. University of San Francisco, 2011.

PORTO, A.; PALMA, J.; *Fábrica do Futuro: Entenda hoje como sua indústria vai ser amanhã.* Ed. Banas, 2000.

REA Foundation. *VR CNC Turning / Operating Software*, 2013. Disponível em <http://rea.org.au/shop/vr-cnc-turning/>, último acesso em 10 jul 2013.

REED, A. *Learning XNA 4.0*. O'Reilly, 2011.

RHINOCEROS. *Rhino 5*, 2012. Disponível em <http://www.rhino3d.com/>, último acesso em 2 ago 2013.

SANTOS, B; *An introduction to Virtual (and other) Realities;* Universidade de Aveiro, 2012.

SANTOS, M; CARVALHO, E; MACHADO, D; PICCININI, M. *A indústria brasileira de máquinas-ferramenta*. BNDES, 2007.

SCHULD, M. *A tutorial series written in C# using the Microsoft XNA Framework,* 2008.

SHI, Y.; DING, C.; ZHOU, L.; *On the Application of Virtual Reality Technology.* University of Techonology Wuhan 2008.

SMITH, L. D. Blender for XNA, *Codestock 2013*. Knoxville, TN, USA, July 2013.

SPUR, G. *Th. Handbuch de Fertigungstechnik.* Carl Hanser Verlag - Viena , 1979.

SSCNC. *Swansoft CNC Simulator*. Disponível em <http://swansoftcncsimulator.com/about>, último acesso em 10 jul 2013.

STOETERAU, R. L. *Introdução ao Projeto de Máquinas-Ferramentas Modernas*. Universidade Federal de Santa Catarina, 2004.

TURBOSQUID. *Lathe*. Disponível em <http://www.turbosquid.com/3d-models/3ds-max-lathe-factories/732299>, último acesso em 31 jul 2013.

VIVANCOS RUBIO, E. *Procesadores de Lenguaje – Introducción.* Universitat Politécnica de Valencia, 2011.

YANG, X.; MALAK, R. C.; LAUER, C.; WEIDIG, C.; HAGEN, H.; HAMANN, B.; AURICH, J. C. *Virtual Reality Enhanced Manufacturing System Design*. 7th International Conference on Digital Enterprise Technology. Athens, Greece, 28-30 September 2011.

WANG, S. CHENG, Y.; *Applying a 3D virtual learning environment to facilitate student's application ability.* Computers in Human Behavior, Elsevier, 2010.

ZUEHLKE, D. *Pervasive computing technologies in industrial applications.* 9th IFAC Symposium on analysis, design and evaluation of human-machine systems, 2004.

ZUEHLKE, D. *Smart-Factory – Towards a factory of things.* Annual Reviews in Control, 2010.

# ANEXO

## Arquivo: Program.cs

```
using System;

using System.Windows.Forms;
using System.Drawing;

namespace TornoSimulation
{
#if WINDOWS || XBOX
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        static void Main(string[] args)
        {
            using (Game1 game = new Game1())
            {
                game.Run();
            }
        }
    }
#endif
}
```

## Arquivo: Game1.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using XELibrary;

namespace TornoSimulation
{
    /// <summary>
    /// Classe principal do programa
    /// </summary>
    public class Game1 : Microsoft.Xna.Framework.Game
    {
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;
        GraphicsDevice device;
        Effect effect;
```

```
#region Definicao de Variaveis

Vector3[,] linhas;
Vector3[,] raio;
Vector3[] teste;
Vector3[] vetorAux;
int time;
int velRotacao;
int velRotBaixa;
int velRotAlta;
int numPontos;
int numLinhas;
float raioInicial;
float posXFerr;
float posYFerr;
float posZFerr;

float posXCarrinho;
float posYCarrinho;
float posZCarrinho;

float velFerr;
float velBaixaFerr;
float velAltaFerr;
float velUsin;
float auxX;
float auxZ;
float auxPasso;
float ratioVelocidades;
float distanciaX;
float distanciaZ;
float fatorCorrecaoVelocidades = 1.214285714f;
bool xpronto = false;
bool bateu = false;
float limiteFimdeCurso = 0.007f;

// Criar Timer
int counter = 1;
int limit = 1000;
float countDuration = 1f;
float currentTime = 0f;

// Variaveis auxiliares utilizadas no acionamento de comandos com o teclado
int t;
int u;
int o;
int q;
int r;
int b;

float velRotArvore;
float rotacaoArvore;
float rotArvoreBaixa;
float rotArvoreAlta;

int extremidadeAtual;
float extremidadeAtualCentro;
```

```
        bool pause; // Variavel para pausar a rotacao da peca
        bool wireframe; // Variavel para alternar entre o modo de visualizacao wireframe e
solido

        Texture2D texture;

        VertexPositionTexture[] vertices;
        VertexPositionTexture[] verticesExtrem;

        // Variavel para desenhar as linhas na extremidade da peca
        VertexPositionColor[] pointListExtrem;

        // Variaveis para desenhar no modo wireframe as linhas paralelas ao eixo da peca
        VertexPositionColor[] pointListAux1;
        VertexPositionColor[] pointListAux2;
        VertexPositionColor[] pointListAux3;
        VertexPositionColor[] pointListAux4;
        VertexPositionColor[] pointListAux5;
        VertexPositionColor[] pointListAux6;
        VertexPositionColor[] pointListAux7;
        VertexPositionColor[] pointListAux8;
        VertexPositionColor[] pointListAux9;
        VertexPositionColor[] pointListAux10;
        VertexPositionColor[] pointListAux11;
        VertexPositionColor[] pointListAux12;
        VertexPositionColor[] pointListAux13;
        VertexPositionColor[] pointListAux14;
        VertexPositionColor[] pointListAux15;
        VertexPositionColor[] pointListAux16;
        VertexPositionColor[] pointListAux17;
        VertexPositionColor[] pointListAux18;
        VertexPositionColor[] pointListAux19;
        VertexPositionColor[] pointListAux20;
        VertexPositionColor[] pointListAux21;
        VertexPositionColor[] pointListAux22;
        VertexPositionColor[] pointListAux23;
        VertexPositionColor[] pointListAux24;
        VertexPositionColor[] pointListAux25;
        VertexPositionColor[] pointListAux26;
        VertexPositionColor[] pointListAux27;
        VertexPositionColor[] pointListAux28;
        VertexPositionColor[] pointListAux29;
        VertexPositionColor[] pointListAux30;
        VertexPositionColor[] pointListAux31;
        VertexPositionColor[] pointListAux32;
        VertexPositionColor[] pointListAux33;
        VertexPositionColor[] pointListAux34;
        VertexPositionColor[] pointListAux35;
        VertexPositionColor[] pointListAux36;
        VertexPositionColor[] pointListAux37;
        VertexPositionColor[] pointListAux38;
        VertexPositionColor[] pointListAux39;
        VertexPositionColor[] pointListAux40;
        VertexPositionColor[] pointListAux41;
        VertexPositionColor[] pointListAux42;
        VertexPositionColor[] pointListAux43;
        VertexPositionColor[] pointListAux44;
        VertexPositionColor[] pointListAux45;
```

```
VertexPositionColor[] pointListAux46;
VertexPositionColor[] pointListAux47;
VertexPositionColor[] pointListAux48;
VertexPositionColor[] pointListAux49;
VertexPositionColor[] pointListAux50;

VertexPositionColor[] pointListAux51;
VertexPositionColor[] pointListAux52;
VertexPositionColor[] pointListAux53;
VertexPositionColor[] pointListAux54;
VertexPositionColor[] pointListAux55;
VertexPositionColor[] pointListAux56;
VertexPositionColor[] pointListAux57;
VertexPositionColor[] pointListAux58;
VertexPositionColor[] pointListAux59;
VertexPositionColor[] pointListAux60;
VertexPositionColor[] pointListAux61;
VertexPositionColor[] pointListAux62;
VertexPositionColor[] pointListAux63;
VertexPositionColor[] pointListAux64;
VertexPositionColor[] pointListAux65;
VertexPositionColor[] pointListAux66;
VertexPositionColor[] pointListAux67;
VertexPositionColor[] pointListAux68;
VertexPositionColor[] pointListAux69;
VertexPositionColor[] pointListAux70;
VertexPositionColor[] pointListAux71;
VertexPositionColor[] pointListAux72;
VertexPositionColor[] pointListAux73;
VertexPositionColor[] pointListAux74;
VertexPositionColor[] pointListAux75;
VertexPositionColor[] pointListAux76;
VertexPositionColor[] pointListAux77;
VertexPositionColor[] pointListAux78;
VertexPositionColor[] pointListAux79;
VertexPositionColor[] pointListAux80;
VertexPositionColor[] pointListAux81;
VertexPositionColor[] pointListAux82;
VertexPositionColor[] pointListAux83;
VertexPositionColor[] pointListAux84;
VertexPositionColor[] pointListAux85;
VertexPositionColor[] pointListAux86;
VertexPositionColor[] pointListAux87;
VertexPositionColor[] pointListAux88;
VertexPositionColor[] pointListAux89;
VertexPositionColor[] pointListAux90;
VertexPositionColor[] pointListAux91;
VertexPositionColor[] pointListAux92;
VertexPositionColor[] pointListAux93;
VertexPositionColor[] pointListAux94;
VertexPositionColor[] pointListAux95;
VertexPositionColor[] pointListAux96;
VertexPositionColor[] pointListAux97;
VertexPositionColor[] pointListAux98;
VertexPositionColor[] pointListAux99;
VertexPositionColor[] pointListAux100;
```

```
        // Variaveis para desenhar no modo wireframe as linhas perpendiculares ao eixo da
peca
        VertexPositionColor[] pointList1;
        VertexPositionColor[] pointList2;
        VertexPositionColor[] pointList3;
        VertexPositionColor[] pointList4;
        VertexPositionColor[] pointList5;
        VertexPositionColor[] pointList6;
        VertexPositionColor[] pointList7;
        VertexPositionColor[] pointList8;
        VertexPositionColor[] pointList9;
        VertexPositionColor[] pointList10;
        VertexPositionColor[] pointList11;
        VertexPositionColor[] pointList12;
        VertexPositionColor[] pointList13;
        VertexPositionColor[] pointList14;
        VertexPositionColor[] pointList15;
        VertexPositionColor[] pointList16;
        VertexPositionColor[] pointList17;
        VertexPositionColor[] pointList18;
        VertexPositionColor[] pointList19;
        VertexPositionColor[] pointList20;
        VertexPositionColor[] pointList21;
        VertexPositionColor[] pointList22;
        VertexPositionColor[] pointList23;
        VertexPositionColor[] pointList24;
        VertexPositionColor[] pointList25;
        VertexPositionColor[] pointList26;
        VertexPositionColor[] pointList27;
        VertexPositionColor[] pointList28;
        VertexPositionColor[] pointList29;
        VertexPositionColor[] pointList30;
        VertexPositionColor[] pointList31;
        VertexPositionColor[] pointList32;
        VertexPositionColor[] pointList33;
        VertexPositionColor[] pointList34;
        VertexPositionColor[] pointList35;
        VertexPositionColor[] pointList36;

        // Variavel que guardam lista de linhas que ligam os pontos para desenhar no modo
wireframe
        short[] lineListIndices1;
        short[] lineListIndices2;
        short[] lineListIndicesExtrem;

        private BasicEffect basicEffect;

        // Uso de objetos das classes da biblioteca XELibrary
        private FPS fps; // Calculo de FPS
        private FirstPersonCamera camera; // Movimentacao da camera
        private InputHandler input; // Interpretacao de comandos

        // Objetos que contem os modelos
        private Model modelSkybox;
        private Model modelEixos;
        private Model modelTornoXNA;
        private Model modelArvore;
        private Model modelCarrinho;
```

```
private Model modelFerramenta;

// Matrizes contendo as transformacoes dos modelos
Matrix modelTransformSkybox;
Matrix modelTransformTornoXNA;
Matrix modelTransformArvore;
Matrix modelTransformCarrinho;
Matrix modelTransformFerramenta;
Matrix modelTransformEixos;

Matrix[] transformSkybox; // Transformacao total da skybox
Matrix[] boneOriginalSkybox; // Posicao original da skybox
Matrix[] transformTornoXNA;
Matrix[] boneOriginalTornoXNA;
Matrix[] transformArvore;
Matrix[] boneOriginalArvore;
Matrix[] transformCarrinho;
Matrix[] boneOriginalCarrinho;
Matrix[] transformFerramenta;
Matrix[] boneOriginalFerramenta;
Matrix[] transformEixos;
Matrix[] boneOriginalEixos;

Matrix position;

// Indices contendo o ID dos ossos
int boneIDSkybox;
int boneIDEixos;
int boneIDTornoXNA;
int boneIDArvore;
int boneIDFerramenta;
int boneIDCarrinho;

// Fonte do texto
SpriteFont font;

// Variaveis para Interface do PATON!
bool StartPressed = false;
bool ExitPressed = false;
bool FerrTras = false;
bool FerrFrente = false;
bool FerrDireita = false;
bool FerrEsquerda = false;

// Para a tela inicial
private Texture2D Imanual;
private Texture2D Iautomatico;
private Texture2D telainicial;

private Texture2D startButton; // Botao de iniciar
private Texture2D exitButton; // Botao de sair
private Texture2D logo; // Logo inicial
private Texture2D barraCodigoG; // Barra lateral direita modo Automatico

// Posicoes na tela
private Vector2 startButtonPosition;
private Vector2 exitButtonPosition;
```

```
// Status do mouse
MouseState mouseState;
MouseState previousMouseState;
// Status do jogo (Playing/ Menu inicial)
GameState gameState;
GameState previousGameState;
// Modo manual automatico
GameMode gameMode;

// Variaveis para efeitos sonoros
SoundEffect soundEngine; // Motores
SoundEffectInstance soundEngineInstance;
SoundEffect soundBang; // Motores
SoundEffectInstance soundBangInstance;

Song metal;
Song giro;

// Areas para deteccao de clique
Rectangle StartRect = new Rectangle(650, 300, 335, 70);
Rectangle ExitRect = new Rectangle(720, 400, 135, 70);
Rectangle FerrTrasRect = new Rectangle(150, 75, 20, 20);
Rectangle FerrFrenteRect = new Rectangle(150, 50, 20, 20);
Rectangle FerrDireitaRect = new Rectangle(185, 75, 20, 20);
Rectangle FerrEsquerdaRect = new Rectangle(105, 75, 20, 20);

Rectangle loadRect = new Rectangle(889, 292, 60, 18);

// Objeto para interpretar krl
KRLManager krl = new KRLManager();

// Variaveis auxiliares
bool interpret;
int numNode = 0;

// Alternar entre a tela inicial e a tela de movimentacao da peca
enum GameState
{
    StartMenu,
    Playing
}

// Alternar entre modo manual e automatico
enum GameMode
{
    Manual,
    Automatic
}

#endregion

public Game1()
{
    graphics = new GraphicsDeviceManager(this);
    Content = new ContentManager(Services);
    Content.RootDirectory = "Content";

    // Adicao dos componentes da biblioteca XELibrary
```

```
            input = new InputHandler(this);
            Components.Add(input);
            camera = new FirstPersonCamera(this);
            Components.Add(camera);
#if DEBUG
            fps = new FPS(this, true, false); // Desenha na tela com o max de FPS possivel
para verificar o desempenho
#else
            fps = new FPS(this, true, false); // Desenha com 60 FPS
#endif
            Components.Add(fps);
            this.Window.Title = "Torno Virtual"; // titulo na barra superior
            // tamanho da tela
            graphics.PreferredBackBufferWidth = 1024;
            graphics.PreferredBackBufferHeight = 648;
            graphics.ApplyChanges();
        }

        protected override void Initialize()
        {
            Window.Title = "Torno Virtual";

            time = 0;
            velRotBaixa = 3;
            velRotAlta = 1;
            velRotacao = velRotAlta;
            numLinhas = 36; // A peca possui 36 pontos em sua secao transversal
            numPontos = 100; // A peca possui 100 pontos na direcao paralela ao eixo da
peca
            raioInicial = 0.13f; // A peca possui 130 mm de raio
            posXFerr = 0.4f;
            posYFerr = 0;
            posZFerr = 0.3f;

            posXCarrinho = 0.4f;
            posYCarrinho = -0.52f;
            posZCarrinho = 0;

            velBaixaFerr = 0.00008f;
            velAltaFerr = 0.001f;
            velFerr = velAltaFerr;

            t = 15;
            u = 15;
            o = 15;
            b = 15;
            q = 15;

            velRotArvore = 0.0f;
            rotArvoreBaixa = 0.05f;
            rotArvoreAlta = 0.25f;
            rotacaoArvore = rotArvoreBaixa;

            extremidadeAtual = 99;
            extremidadeAtualCentro = 0.5f;

            pause = true;
            wireframe = true;
```

```
teste = new Vector3[1];
vetorAux = new Vector3[1];

pointListExtrem = new VertexPositionColor[2*numLinhas];

pointListAux1 = new VertexPositionColor[numLinhas];
pointListAux2 = new VertexPositionColor[numLinhas];
pointListAux3 = new VertexPositionColor[numLinhas];
pointListAux4 = new VertexPositionColor[numLinhas];
pointListAux5 = new VertexPositionColor[numLinhas];
pointListAux6 = new VertexPositionColor[numLinhas];
pointListAux7 = new VertexPositionColor[numLinhas];
pointListAux8 = new VertexPositionColor[numLinhas];
pointListAux9 = new VertexPositionColor[numLinhas];
pointListAux10 = new VertexPositionColor[numLinhas];
pointListAux11 = new VertexPositionColor[numLinhas];
pointListAux12 = new VertexPositionColor[numLinhas];
pointListAux13 = new VertexPositionColor[numLinhas];
pointListAux14 = new VertexPositionColor[numLinhas];
pointListAux15 = new VertexPositionColor[numLinhas];
pointListAux16 = new VertexPositionColor[numLinhas];
pointListAux17 = new VertexPositionColor[numLinhas];
pointListAux18 = new VertexPositionColor[numLinhas];
pointListAux19 = new VertexPositionColor[numLinhas];
pointListAux20 = new VertexPositionColor[numLinhas];
pointListAux21 = new VertexPositionColor[numLinhas];
pointListAux22 = new VertexPositionColor[numLinhas];
pointListAux23 = new VertexPositionColor[numLinhas];
pointListAux24 = new VertexPositionColor[numLinhas];
pointListAux25 = new VertexPositionColor[numLinhas];
pointListAux26 = new VertexPositionColor[numLinhas];
pointListAux27 = new VertexPositionColor[numLinhas];
pointListAux28 = new VertexPositionColor[numLinhas];
pointListAux29 = new VertexPositionColor[numLinhas];
pointListAux30 = new VertexPositionColor[numLinhas];
pointListAux31 = new VertexPositionColor[numLinhas];
pointListAux32 = new VertexPositionColor[numLinhas];
pointListAux33 = new VertexPositionColor[numLinhas];
pointListAux34 = new VertexPositionColor[numLinhas];
pointListAux35 = new VertexPositionColor[numLinhas];
pointListAux36 = new VertexPositionColor[numLinhas];
pointListAux37 = new VertexPositionColor[numLinhas];
pointListAux38 = new VertexPositionColor[numLinhas];
pointListAux39 = new VertexPositionColor[numLinhas];
pointListAux40 = new VertexPositionColor[numLinhas];
pointListAux41 = new VertexPositionColor[numLinhas];
pointListAux42 = new VertexPositionColor[numLinhas];
pointListAux43 = new VertexPositionColor[numLinhas];
pointListAux44 = new VertexPositionColor[numLinhas];
pointListAux45 = new VertexPositionColor[numLinhas];
pointListAux46 = new VertexPositionColor[numLinhas];
pointListAux47 = new VertexPositionColor[numLinhas];
pointListAux48 = new VertexPositionColor[numLinhas];
pointListAux49 = new VertexPositionColor[numLinhas];
pointListAux50 = new VertexPositionColor[numLinhas];

pointListAux51 = new VertexPositionColor[numLinhas];
```

```
pointListAux52 = new VertexPositionColor[numLinhas];
pointListAux53 = new VertexPositionColor[numLinhas];
pointListAux54 = new VertexPositionColor[numLinhas];
pointListAux55 = new VertexPositionColor[numLinhas];
pointListAux56 = new VertexPositionColor[numLinhas];
pointListAux57 = new VertexPositionColor[numLinhas];
pointListAux58 = new VertexPositionColor[numLinhas];
pointListAux59 = new VertexPositionColor[numLinhas];
pointListAux60 = new VertexPositionColor[numLinhas];
pointListAux61 = new VertexPositionColor[numLinhas];
pointListAux62 = new VertexPositionColor[numLinhas];
pointListAux63 = new VertexPositionColor[numLinhas];
pointListAux64 = new VertexPositionColor[numLinhas];
pointListAux65 = new VertexPositionColor[numLinhas];
pointListAux66 = new VertexPositionColor[numLinhas];
pointListAux67 = new VertexPositionColor[numLinhas];
pointListAux68 = new VertexPositionColor[numLinhas];
pointListAux69 = new VertexPositionColor[numLinhas];
pointListAux70 = new VertexPositionColor[numLinhas];
pointListAux71 = new VertexPositionColor[numLinhas];
pointListAux72 = new VertexPositionColor[numLinhas];
pointListAux73 = new VertexPositionColor[numLinhas];
pointListAux74 = new VertexPositionColor[numLinhas];
pointListAux75 = new VertexPositionColor[numLinhas];
pointListAux76 = new VertexPositionColor[numLinhas];
pointListAux77 = new VertexPositionColor[numLinhas];
pointListAux78 = new VertexPositionColor[numLinhas];
pointListAux79 = new VertexPositionColor[numLinhas];
pointListAux80 = new VertexPositionColor[numLinhas];
pointListAux81 = new VertexPositionColor[numLinhas];
pointListAux82 = new VertexPositionColor[numLinhas];
pointListAux83 = new VertexPositionColor[numLinhas];
pointListAux84 = new VertexPositionColor[numLinhas];
pointListAux85 = new VertexPositionColor[numLinhas];
pointListAux86 = new VertexPositionColor[numLinhas];
pointListAux87 = new VertexPositionColor[numLinhas];
pointListAux88 = new VertexPositionColor[numLinhas];
pointListAux89 = new VertexPositionColor[numLinhas];
pointListAux90 = new VertexPositionColor[numLinhas];
pointListAux91 = new VertexPositionColor[numLinhas];
pointListAux92 = new VertexPositionColor[numLinhas];
pointListAux93 = new VertexPositionColor[numLinhas];
pointListAux94 = new VertexPositionColor[numLinhas];
pointListAux95 = new VertexPositionColor[numLinhas];
pointListAux96 = new VertexPositionColor[numLinhas];
pointListAux97 = new VertexPositionColor[numLinhas];
pointListAux98 = new VertexPositionColor[numLinhas];
pointListAux99 = new VertexPositionColor[numLinhas];
pointListAux100 = new VertexPositionColor[numLinhas];

pointList1 = new VertexPositionColor[numPontos];
pointList2 = new VertexPositionColor[numPontos];
pointList3 = new VertexPositionColor[numPontos];
pointList4 = new VertexPositionColor[numPontos];
pointList5 = new VertexPositionColor[numPontos];
pointList6 = new VertexPositionColor[numPontos];
pointList7 = new VertexPositionColor[numPontos];
pointList8 = new VertexPositionColor[numPontos];
```

```
            pointList9 = new VertexPositionColor[numPontos];
            pointList10 = new VertexPositionColor[numPontos];
            pointList11 = new VertexPositionColor[numPontos];
            pointList12 = new VertexPositionColor[numPontos];
            pointList13 = new VertexPositionColor[numPontos];
            pointList14 = new VertexPositionColor[numPontos];
            pointList15 = new VertexPositionColor[numPontos];
            pointList16 = new VertexPositionColor[numPontos];
            pointList17 = new VertexPositionColor[numPontos];
            pointList18 = new VertexPositionColor[numPontos];
            pointList19 = new VertexPositionColor[numPontos];
            pointList20 = new VertexPositionColor[numPontos];
            pointList21 = new VertexPositionColor[numPontos];
            pointList22 = new VertexPositionColor[numPontos];
            pointList23 = new VertexPositionColor[numPontos];
            pointList24 = new VertexPositionColor[numPontos];
            pointList25 = new VertexPositionColor[numPontos];
            pointList26 = new VertexPositionColor[numPontos];
            pointList27 = new VertexPositionColor[numPontos];
            pointList28 = new VertexPositionColor[numPontos];
            pointList29 = new VertexPositionColor[numPontos];
            pointList30 = new VertexPositionColor[numPontos];
            pointList31 = new VertexPositionColor[numPontos];
            pointList32 = new VertexPositionColor[numPontos];
            pointList33 = new VertexPositionColor[numPontos];
            pointList34 = new VertexPositionColor[numPontos];
            pointList35 = new VertexPositionColor[numPontos];
            pointList36 = new VertexPositionColor[numPontos];

            raio = new Vector3[numLinhas, numPontos]; // Distancia ate o centro de cada
ponto que forma o cilindro

            for (int i = 0; i < numLinhas; i++)
            {
                for (int j = 0; j < numPontos; j++)
                {
                    raio[i, j].X = raioInicial;
                }
            }

            linhas = new Vector3[numLinhas, numPontos];

            Vector3 point = new Vector3();
            for (int i = 0; i < numPontos; i++)
            {
                point.X = (float) (i+1) / (numPontos*2);
                point.Y = raioInicial;

                linhas[0, i] = point;
            }

            Vector3 point2 = new Vector3();
            float angle = 90;
            for (int i = 0; i < numLinhas; i++)
            {
                for (int j = 0; j < numPontos; j++)
                {
                    point2.X = linhas[0, j].X;
```

```
                    point2.Y = linhas[0, j].Y * (float)Math.Sin(angle * Math.PI / 180);
                    point2.Z = linhas[0, j].Y * (float)Math.Cos(angle * Math.PI / 180);
                    linhas[i, j] = point2;
                }
                angle -= (float) 10;
            }

            for (int i = 0; i < 2 * numLinhas; i++)
            {
                if (i % 2 == 0)
                {
                    vetorAux[0].X = extremidadeAtualCentro;
                    vetorAux[0].Y = 0;
                    vetorAux[0].Z = 0;
                    pointListExtrem[i] = new VertexPositionColor(vetorAux[0], Color.Blue);
                }
                else
                {
                    pointListExtrem[i] = new VertexPositionColor(linhas[(i - 1) / 2, extre-
midadeAtual], Color.Blue);
                }
            }

            for (int i = 0; i < numLinhas; i++)
            {
                pointListAux1[i] = new VertexPositionColor(linhas[i, 0], Color.Blue);
                pointListAux2[i] = new VertexPositionColor(linhas[i, 1], Color.Blue);
                pointListAux3[i] = new VertexPositionColor(linhas[i, 2], Color.Blue);
                pointListAux4[i] = new VertexPositionColor(linhas[i, 3], Color.Blue);
                pointListAux5[i] = new VertexPositionColor(linhas[i, 4], Color.Blue);
                pointListAux6[i] = new VertexPositionColor(linhas[i, 5], Color.Blue);
                pointListAux7[i] = new VertexPositionColor(linhas[i, 6], Color.Blue);
                pointListAux8[i] = new VertexPositionColor(linhas[i, 7], Color.Blue);
                pointListAux9[i] = new VertexPositionColor(linhas[i, 8], Color.Blue);
                pointListAux10[i] = new VertexPositionColor(linhas[i, 9], Color.Blue);
                pointListAux11[i] = new VertexPositionColor(linhas[i, 10], Color.Blue);
                pointListAux12[i] = new VertexPositionColor(linhas[i, 11], Color.Blue);
                pointListAux13[i] = new VertexPositionColor(linhas[i, 12], Color.Blue);
                pointListAux14[i] = new VertexPositionColor(linhas[i, 13], Color.Blue);
                pointListAux15[i] = new VertexPositionColor(linhas[i, 14], Color.Blue);
                pointListAux16[i] = new VertexPositionColor(linhas[i, 15], Color.Blue);
                pointListAux17[i] = new VertexPositionColor(linhas[i, 16], Color.Blue);
                pointListAux18[i] = new VertexPositionColor(linhas[i, 17], Color.Blue);
                pointListAux19[i] = new VertexPositionColor(linhas[i, 18], Color.Blue);
                pointListAux20[i] = new VertexPositionColor(linhas[i, 19], Color.Blue);
                pointListAux21[i] = new VertexPositionColor(linhas[i, 20], Color.Blue);
                pointListAux22[i] = new VertexPositionColor(linhas[i, 21], Color.Blue);
                pointListAux23[i] = new VertexPositionColor(linhas[i, 22], Color.Blue);
                pointListAux24[i] = new VertexPositionColor(linhas[i, 23], Color.Blue);
                pointListAux25[i] = new VertexPositionColor(linhas[i, 24], Color.Blue);
                pointListAux26[i] = new VertexPositionColor(linhas[i, 25], Color.Blue);
                pointListAux27[i] = new VertexPositionColor(linhas[i, 26], Color.Blue);
                pointListAux28[i] = new VertexPositionColor(linhas[i, 27], Color.Blue);
                pointListAux29[i] = new VertexPositionColor(linhas[i, 28], Color.Blue);
                pointListAux30[i] = new VertexPositionColor(linhas[i, 29], Color.Blue);
                pointListAux31[i] = new VertexPositionColor(linhas[i, 30], Color.Blue);
                pointListAux32[i] = new VertexPositionColor(linhas[i, 31], Color.Blue);
                pointListAux33[i] = new VertexPositionColor(linhas[i, 32], Color.Blue);
```

```
pointListAux34[i] = new VertexPositionColor(linhas[i, 33], Color.Blue);
pointListAux35[i] = new VertexPositionColor(linhas[i, 34], Color.Blue);
pointListAux36[i] = new VertexPositionColor(linhas[i, 35], Color.Blue);
pointListAux37[i] = new VertexPositionColor(linhas[i, 36], Color.Blue);
pointListAux38[i] = new VertexPositionColor(linhas[i, 37], Color.Blue);
pointListAux39[i] = new VertexPositionColor(linhas[i, 38], Color.Blue);
pointListAux40[i] = new VertexPositionColor(linhas[i, 39], Color.Blue);
pointListAux41[i] = new VertexPositionColor(linhas[i, 40], Color.Blue);
pointListAux42[i] = new VertexPositionColor(linhas[i, 41], Color.Blue);
pointListAux43[i] = new VertexPositionColor(linhas[i, 42], Color.Blue);
pointListAux44[i] = new VertexPositionColor(linhas[i, 43], Color.Blue);
pointListAux45[i] = new VertexPositionColor(linhas[i, 44], Color.Blue);
pointListAux46[i] = new VertexPositionColor(linhas[i, 45], Color.Blue);
pointListAux47[i] = new VertexPositionColor(linhas[i, 46], Color.Blue);
pointListAux48[i] = new VertexPositionColor(linhas[i, 47], Color.Blue);
pointListAux49[i] = new VertexPositionColor(linhas[i, 48], Color.Blue);
pointListAux50[i] = new VertexPositionColor(linhas[i, 49], Color.Blue);

pointListAux51[i] = new VertexPositionColor(linhas[i, 50], Color.Blue);
pointListAux52[i] = new VertexPositionColor(linhas[i, 51], Color.Blue);
pointListAux53[i] = new VertexPositionColor(linhas[i, 52], Color.Blue);
pointListAux54[i] = new VertexPositionColor(linhas[i, 53], Color.Blue);
pointListAux55[i] = new VertexPositionColor(linhas[i, 54], Color.Blue);
pointListAux56[i] = new VertexPositionColor(linhas[i, 55], Color.Blue);
pointListAux57[i] = new VertexPositionColor(linhas[i, 56], Color.Blue);
pointListAux58[i] = new VertexPositionColor(linhas[i, 57], Color.Blue);
pointListAux59[i] = new VertexPositionColor(linhas[i, 58], Color.Blue);
pointListAux60[i] = new VertexPositionColor(linhas[i, 59], Color.Blue);
pointListAux61[i] = new VertexPositionColor(linhas[i, 60], Color.Blue);
pointListAux62[i] = new VertexPositionColor(linhas[i, 61], Color.Blue);
pointListAux63[i] = new VertexPositionColor(linhas[i, 62], Color.Blue);
pointListAux64[i] = new VertexPositionColor(linhas[i, 63], Color.Blue);
pointListAux65[i] = new VertexPositionColor(linhas[i, 64], Color.Blue);
pointListAux66[i] = new VertexPositionColor(linhas[i, 65], Color.Blue);
pointListAux67[i] = new VertexPositionColor(linhas[i, 66], Color.Blue);
pointListAux68[i] = new VertexPositionColor(linhas[i, 67], Color.Blue);
pointListAux69[i] = new VertexPositionColor(linhas[i, 68], Color.Blue);
pointListAux70[i] = new VertexPositionColor(linhas[i, 69], Color.Blue);
pointListAux71[i] = new VertexPositionColor(linhas[i, 70], Color.Blue);
pointListAux72[i] = new VertexPositionColor(linhas[i, 71], Color.Blue);
pointListAux73[i] = new VertexPositionColor(linhas[i, 72], Color.Blue);
pointListAux74[i] = new VertexPositionColor(linhas[i, 73], Color.Blue);
pointListAux75[i] = new VertexPositionColor(linhas[i, 74], Color.Blue);
pointListAux76[i] = new VertexPositionColor(linhas[i, 75], Color.Blue);
pointListAux77[i] = new VertexPositionColor(linhas[i, 76], Color.Blue);
pointListAux78[i] = new VertexPositionColor(linhas[i, 77], Color.Blue);
pointListAux79[i] = new VertexPositionColor(linhas[i, 78], Color.Blue);
pointListAux80[i] = new VertexPositionColor(linhas[i, 79], Color.Blue);
pointListAux81[i] = new VertexPositionColor(linhas[i, 80], Color.Blue);
pointListAux82[i] = new VertexPositionColor(linhas[i, 81], Color.Blue);
pointListAux83[i] = new VertexPositionColor(linhas[i, 82], Color.Blue);
pointListAux84[i] = new VertexPositionColor(linhas[i, 83], Color.Blue);
pointListAux85[i] = new VertexPositionColor(linhas[i, 84], Color.Blue);
pointListAux86[i] = new VertexPositionColor(linhas[i, 85], Color.Blue);
pointListAux87[i] = new VertexPositionColor(linhas[i, 86], Color.Blue);
pointListAux88[i] = new VertexPositionColor(linhas[i, 87], Color.Blue);
pointListAux89[i] = new VertexPositionColor(linhas[i, 88], Color.Blue);
pointListAux90[i] = new VertexPositionColor(linhas[i, 89], Color.Blue);
```

```
        pointListAux91[i] = new VertexPositionColor(linhas[i, 90], Color.Blue);
        pointListAux92[i] = new VertexPositionColor(linhas[i, 91], Color.Blue);
        pointListAux93[i] = new VertexPositionColor(linhas[i, 92], Color.Blue);
        pointListAux94[i] = new VertexPositionColor(linhas[i, 93], Color.Blue);
        pointListAux95[i] = new VertexPositionColor(linhas[i, 94], Color.Blue);
        pointListAux96[i] = new VertexPositionColor(linhas[i, 95], Color.Blue);
        pointListAux97[i] = new VertexPositionColor(linhas[i, 96], Color.Blue);
        pointListAux98[i] = new VertexPositionColor(linhas[i, 97], Color.Blue);
        pointListAux99[i] = new VertexPositionColor(linhas[i, 98], Color.Blue);
        pointListAux100[i] = new VertexPositionColor(linhas[i, 99], Color.Blue);
    }

    for (int i = 0; i < numPontos; i++)
    {
        pointList1[i] = new VertexPositionColor(linhas[0, i], Color.Blue);
        pointList2[i] = new VertexPositionColor(linhas[1, i], Color.Blue);
        pointList3[i] = new VertexPositionColor(linhas[2, i], Color.Blue);
        pointList4[i] = new VertexPositionColor(linhas[3, i], Color.Blue);
        pointList5[i] = new VertexPositionColor(linhas[4, i], Color.Blue);
        pointList6[i] = new VertexPositionColor(linhas[5, i], Color.Blue);
        pointList7[i] = new VertexPositionColor(linhas[6, i], Color.Blue);
        pointList8[i] = new VertexPositionColor(linhas[7, i], Color.Blue);

        pointList9[i] = new VertexPositionColor(linhas[8, i], Color.Blue);
        pointList10[i] = new VertexPositionColor(linhas[9, i], Color.Blue);
        pointList11[i] = new VertexPositionColor(linhas[10, i], Color.Blue);
        pointList12[i] = new VertexPositionColor(linhas[11, i], Color.Blue);
        pointList13[i] = new VertexPositionColor(linhas[12, i], Color.Blue);
        pointList14[i] = new VertexPositionColor(linhas[13, i], Color.Blue);
        pointList15[i] = new VertexPositionColor(linhas[14, i], Color.Blue);
        pointList16[i] = new VertexPositionColor(linhas[15, i], Color.Blue);

        pointList17[i] = new VertexPositionColor(linhas[16, i], Color.Blue);
        pointList18[i] = new VertexPositionColor(linhas[17, i], Color.Blue);
        pointList19[i] = new VertexPositionColor(linhas[18, i], Color.Blue);
        pointList20[i] = new VertexPositionColor(linhas[19, i], Color.Blue);
        pointList21[i] = new VertexPositionColor(linhas[20, i], Color.Blue);
        pointList22[i] = new VertexPositionColor(linhas[21, i], Color.Blue);
        pointList23[i] = new VertexPositionColor(linhas[22, i], Color.Blue);
        pointList24[i] = new VertexPositionColor(linhas[23, i], Color.Blue);

        pointList25[i] = new VertexPositionColor(linhas[24, i], Color.Blue);
        pointList26[i] = new VertexPositionColor(linhas[25, i], Color.Blue);
        pointList27[i] = new VertexPositionColor(linhas[26, i], Color.Blue);
        pointList28[i] = new VertexPositionColor(linhas[27, i], Color.Blue);
        pointList29[i] = new VertexPositionColor(linhas[28, i], Color.Blue);
        pointList30[i] = new VertexPositionColor(linhas[29, i], Color.Blue);
        pointList31[i] = new VertexPositionColor(linhas[30, i], Color.Blue);
        pointList32[i] = new VertexPositionColor(linhas[31, i], Color.Blue);

        pointList33[i] = new VertexPositionColor(linhas[32, i], Color.Blue);
        pointList34[i] = new VertexPositionColor(linhas[33, i], Color.Blue);
        pointList35[i] = new VertexPositionColor(linhas[34, i], Color.Blue);
        pointList36[i] = new VertexPositionColor(linhas[35, i], Color.Blue);
    }

    basicEffect = new BasicEffect(graphics.GraphicsDevice);
    basicEffect.VertexColorEnabled = true;
```

```
            // Inicializa um arranjo de indices do tipo short
            lineListIndices1 = new short[numPontos];
            lineListIndices2 = new short[numLinhas];
            lineListIndicesExtrem = new short[2*numLinhas];

            // Popula o arranjo com referencia aos indices dos vertices
            for (int i = 0; i < numPontos; i++)
            {
                lineListIndices1[i] = (short)(i);
            }
            for (int i = 0; i < numLinhas; i++)
            {
                lineListIndices2[i] = (short)(i);
            }

            lineListIndices2[35] = (short)(0); // Liga o primeiro com o ultimo ponto da
sequencia

            for (int i = 0; i < 2*numLinhas; i++)
            {
                lineListIndicesExtrem[i] = (short)(i);
            }

            // Defininido a posicao dos objetos 2D da tela
            startButtonPosition = new Vector2((GraphicsDevice.Viewport.Width / 2) - 100,
380);
            exitButtonPosition = new Vector2((GraphicsDevice.Viewport.Width / 2) - 70 ,
470);

            // Define que o estado inicial é o da tela inicial
            gameState = GameState.StartMenu;
            previousGameState = GameState.StartMenu;
            gameMode = GameMode.Manual;

            // Obtem o status do mouse
            mouseState = Mouse.GetState();
            previousMouseState = mouseState;

            base.Initialize();
        }

        protected override void LoadContent()
        {
            spriteBatch = new SpriteBatch(GraphicsDevice);
            basicEffect = new BasicEffect(GraphicsDevice);
            device = graphics.GraphicsDevice;

            // Carrega os arquivos necessarios de imagem / som / fonte
            font = Content.Load<SpriteFont>("myFont");
            Imanual = Content.Load<Texture2D>(@"Imanual");
            Iautomatico = Content.Load<Texture2D>(@"Iautomatico");
            telainicial = Content.Load<Texture2D>(@"telainicial");
            barraCodigoG = Content.Load<Texture2D>(@"barraCodigoG");
            modelTornoXNA = Content.Load<Model>(@"Models\tornoXNA");
            modelArvore = Content.Load<Model>(@"Models\arvoreTorno");
            modelCarrinho = Content.Load<Model>(@"Models\carrinhoTorno");
            modelFerramenta = Content.Load<Model>(@"Models\ferramenta");
```

```
        modelSkybox = Content.Load<Model>(@"Models\skybox");
        modelEixos = Content.Load<Model>(@"Models\eixosXYZ");

        effect = Content.Load<Effect>("effects");
        texture = Content.Load<Texture2D>("textura");

        metal = Content.Load<Song>(@"Sound\metal");
        giro = Content.Load<Song>("giro");
        soundEngine = Content.Load<SoundEffect>("machine_1");
        soundEngineInstance = soundEngine.CreateInstance();

        soundBang = Content.Load<SoundEffect>("bang_1");
        soundBangInstance = soundBang.CreateInstance();
    }

    protected void LoadGame()
    {
        // Cria um SpriteBatch utilizado para o desenho das imagens 2D
        spriteBatch = new SpriteBatch(GraphicsDevice);

        // Verificacao dos IDs dos ossos
        boneIDSkybox = modelSkybox.Bones["Bone"].Index;
        modelTransformSkybox = modelSkybox.Bones[boneIDSkybox].Transform;
        boneIDEixos = modelEixos.Bones["Bone"].Index;
        modelTransformEixos = modelEixos.Bones[boneIDEixos].Transform;
        boneIDTornoXNA = modelTornoXNA.Bones["Bone"].Index;
        modelTransformTornoXNA = modelTornoXNA.Bones[boneIDTornoXNA].Transform;
        boneIDArvore = modelArvore.Bones["Bone"].Index;
        modelTransformArvore = modelArvore.Bones[boneIDArvore].Transform;
        boneIDCarrinho = modelCarrinho.Bones["Bone"].Index;
        modelTransformCarrinho = modelCarrinho.Bones[boneIDCarrinho].Transform;
        boneIDFerramenta = modelFerramenta.Bones["Bone"].Index;
        modelTransformFerramenta = modelFerramenta.Bones[boneIDFerramenta].Transform;

        // Criacao das matrizes de transformacao dos modelos
        transformSkybox = new Matrix[modelSkybox.Bones.Count];
        boneOriginalSkybox = new Matrix[modelSkybox.Bones.Count];
        transformTornoXNA = new Matrix[modelTornoXNA.Bones.Count];
        boneOriginalTornoXNA = new Matrix[modelTornoXNA.Bones.Count];
        transformArvore = new Matrix[modelArvore.Bones.Count];
        boneOriginalArvore = new Matrix[modelArvore.Bones.Count];
        transformCarrinho = new Matrix[modelCarrinho.Bones.Count];
        boneOriginalCarrinho = new Matrix[modelCarrinho.Bones.Count];
        transformFerramenta = new Matrix[modelFerramenta.Bones.Count];
        boneOriginalFerramenta = new Matrix[modelFerramenta.Bones.Count];
        transformEixos = new Matrix[modelEixos.Bones.Count];
        boneOriginalEixos = new Matrix[modelEixos.Bones.Count];

        modelTornoXNA.CopyAbsoluteBoneTransformsTo(transformTornoXNA);

        // Transformacoes iniciais para que os modelos fiquem nas posicoes corretas
        // Obs.: Blender e XNA possuem sistemas de coordenadas diferentes e por isso
        //       sao necessarias rotacoes para os modelos ficarem nas posicoes corretas

        modelSkybox.Root.Transform = Matrix.CreateTranslation(0, 0, 1) *
            Matrix.CreateRotationX(-MathHelper.PiOver2) *
            Matrix.CreateRotationY(-MathHelper.PiOver2);
        modelSkybox.CopyAbsoluteBoneTransformsTo(boneOriginalSkybox);
```

```
            modelTornoXNA.Root.Transform = Matrix.CreateTranslation((float)0.8, (float)0,
(float)0.15);
            modelTornoXNA.CopyAbsoluteBoneTransformsTo(boneOriginalTornoXNA);

            modelArvore.Root.Transform = Matrix.CreateTranslation((float)0, (float)0,
(float)0);
            modelArvore.CopyAbsoluteBoneTransformsTo(boneOriginalArvore);

            modelCarrinho.Root.Transform = Matrix.CreateTranslation(posXCarrinho, posYCar-
rinho, posZCarrinho);
            modelCarrinho.CopyAbsoluteBoneTransformsTo(boneOriginalCarrinho);

            modelFerramenta.Root.Transform = Matrix.CreateTranslation(posXFerr, posYFerr,
posZFerr);
            modelFerramenta.CopyAbsoluteBoneTransformsTo(boneOriginalFerramenta);

            modelEixos.Root.Transform = Matrix.CreateTranslation((float)-1.5, (float)0,
(float)0);
            modelEixos.CopyAbsoluteBoneTransformsTo(boneOriginalEixos);

            modelTornoXNA.CopyAbsoluteBoneTransformsTo(transformTornoXNA);
            position = transformTornoXNA[boneIDTornoXNA] * Matrix.CreateTranslation(0, 1,
0);
        }

        protected override void UnloadContent()
        {
            // TODO: Unload any non ContentManager content here
        }

        protected override void Update(GameTime gameTime)
        {
            // Para o jogo sair caso seja pressionado ESC
            if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
                this.Exit();

            // Aguarda clique do mouse
            mouseState = Mouse.GetState();
            if (previousMouseState.LeftButton == ButtonState.Pressed )
            {
                MouseClicked(mouseState.X, mouseState.Y);
            }

            previousMouseState = mouseState;

            // Se o jogo esta na tela de movimentacao da peca
            if (gameState == GameState.Playing)
            {
                currentTime += (float)gameTime.ElapsedGameTime.TotalSeconds; //Time passed
since last Update()

                if (currentTime >= countDuration)
                {
                    counter++;
                    currentTime -= countDuration;
                }
                if (counter >= limit)
```

```
{
    counter = 0;
}

if (posXFerr < 0.008f)
{
    soundBangInstance.Play();
}

u = u - 1;
if (u <= 0)
{
    u = 0;
}
if (input.KeyboardState.IsKeyDown(Keys.R) && gameMode == GameMode.Manual)
{
    if (velRotacao == velRotBaixa)
    {
        if (u == 0)
        {
            velRotacao = velRotAlta;
            rotacaoArvore = rotArvoreAlta;
            time = velRotAlta;
            u = 15;
        }
    }
    else
    {
        if (u == 0)
        {
            velRotacao = velRotBaixa;
            rotacaoArvore = rotArvoreBaixa;
            time = velRotBaixa;
            u = 15;
        }
    }
}

o = o - 1;
if (o <= 0)
{
    o = 0;
}
if (input.KeyboardState.IsKeyDown(Keys.F) && gameMode == GameMode.Manual)
{
    if (velFerr == velBaixaFerr)
    {
        if (o == 0)
        {
            velFerr = velAltaFerr;
            o = 15;
        }
    }
    else
    {
        if (o == 0)
        {
            velFerr = velBaixaFerr;
```

```
                o = 15;
            }
        }
    }

    b = b - 1;
    if (b <= 0)
    {
        b = 0;
    }
    if (input.KeyboardState.IsKeyDown(Keys.B) && gameMode == GameMode.Manual)
    {
        if (pause == true)
        {
            if (b == 0)
            {
                if (velRotacao == velRotAlta)
                {
                    velRotacao = velRotBaixa;
                    rotacaoArvore = rotArvoreBaixa;
                }
                pause = false;
                b = 15;
                if(currentTime<10.0f)
                {
                soundEngineInstance.Play();
                MediaPlayer.IsRepeating = true;
                }
            }
        }
        else
        {
            if (b == 0)
            {
                if (velRotacao == velRotBaixa)
                {
                    pause = true;
                    MediaPlayer.IsRepeating = false;
                    soundEngineInstance.Pause();
                }
                if (velRotacao == velRotAlta)
                {
                    velRotacao = velRotBaixa;
                    rotacaoArvore = rotArvoreBaixa;
                    r = 50; // Variavel para adicionar um pouco de inercia caso
esteja em rotacao alta
                }
                b = 15;
            }
        }
    }

    r = r - 1;
    if (r <= 0)
    {
        r = 0;
    }
    // Adiciona um pouco de inercia caso esteja em rotacao alta
```

```
                if (r == 1)
                {
                    pause = true;
                    MediaPlayer.IsRepeating = false;
                    soundEngineInstance.Pause();
                }

                q = q - 1;
                if (q <= 0)
                {
                    q = 0;
                }
                if (input.KeyboardState.IsKeyDown(Keys.U))
                {
                    if (wireframe == true)
                    {
                        if (q == 0)
                        {
                            wireframe = false;
                            q = 15;
                        }
                    }
                    else
                    {
                        if (q == 0)
                        {
                            wireframe = true;
                            q = 15;
                        }
                    }
                }

                if (input.KeyboardState.IsKeyDown(Keys.T) && gameMode == GameMode.Manual)
                {
                    for (int i = 0; i < numLinhas; i++)
                    {
                        for (int j = 0; j < numPontos; j++)
                        {
                            raio[i, j].X = raioInicial;
                        }
                    }
                }

                if (input.KeyboardState.IsKeyDown(Keys.P) && gameMode ==
GameMode.Automatic)
                {
                    interpret = true;
                }

                if (input.KeyboardState.IsKeyDown(Keys.O) && gameMode ==
GameMode.Automatic)
                {
                    interpret = false;
                }

                // Faz a translacao da ferramenta
                modelFerramenta.Root.Transform = Matrix.CreateTranslation(posXFerr,
posYFerr, posZFerr);
```

```
                // Faz a translacao do carrinho
                modelCarrinho.Root.Transform = Matrix.CreateTranslation(posXCarrinho, posY-
Carrinho, posZCarrinho);

                if (pause == false)
                {
                    // Faz a rotacao da arvore
                    modelArvore.Root.Transform = Matrix.CreateRotationX(velRotArvore);
                    velRotArvore = velRotArvore + rotacaoArvore;

                    // Condicao para evitar que o valor de velRotArvore fique infinitamente
grande
                    if (velRotArvore == 18000.0f || velRotArvore == 36000.0f || velRotArvo-
re == 72000.0f)
                    {
                        velRotArvore = 0.0f;
                    }
                }

                if ((input.KeyboardState.IsKeyDown(Keys.J) || (FerrEsquerda)) && gameMode
== GameMode.Manual)
                {
                    posXFerr = posXFerr - velFerr;
                    posXCarrinho = posXCarrinho - velFerr;

                    // Limite para movimentacao da ferramenta a esquerda
                    if (posXFerr <= 0.08f)
                    {
                        posXFerr = 0.083f;
                        posXCarrinho = 0.083f;
                        soundBangInstance.Play();
                    }
                    FerrEsquerda = false;
                }
                if ( (input.KeyboardState.IsKeyDown(Keys.L) || (FerrDireita)) && gameMode
== GameMode.Manual)
                {
                    posXFerr = posXFerr + velFerr;
                    posXCarrinho = posXCarrinho + velFerr;
                    // Limite para movimentacao da ferramenta a direita
                    if (posXFerr >= 0.501)
                    {
                        posXFerr = 0.501f;
                        posXCarrinho = 0.501f;
                    }
                    FerrDireita = false;
                }

                if ( (input.KeyboardState.IsKeyDown(Keys.I) || (FerrFrente)) && gameMode ==
GameMode.Manual)
                {
                    posZFerr = posZFerr - velFerr;
                    // Limite para movimentacao da ferramenta para frente, ao mesmo tempo
que "corta" a peca caso a ferramenta atinga o centro
                    if (pause == true)
                    {
                        if (posZFerr <= 0.13f)
                        {
```

```
                posZFerr = 0.13f;
            }
        }
        else if (posZFerr <= 0.001f)
        {
            posZFerr = 0.001f;
            int g;
            int h;
            int c = RaioMaisProximo(teste[0]);
            for(g = 0 ; g < numLinhas; g++)
            {
                for (h = c; h < numPontos; h++ )
                {
                    raio[g,h].X = 0.0f;
                }
            }
            extremidadeAtual = c-1;
            extremidadeAtualCentro = teste[0].X;
        }
        FerrFrente = false;
    }

    if ( (posZFerr <= 0.001f) && gameMode == GameMode.Automatic )
    {
        //posZFerr = 0.01f;
        int g;
        int h;
        int c = RaioMaisProximo(teste[0]);
        for (g = 0; g < numLinhas; g++)
        {
            for (h = c; h < numPontos; h++)
            {
                raio[g, h].X = 0.0f;
            }
        }
        extremidadeAtual = c-1;
        extremidadeAtualCentro = teste[0].X;
    }

    if (input.KeyboardState.IsKeyDown(Keys.Escape))
    {
        this.Exit();
    }


    if ( (input.KeyboardState.IsKeyDown(Keys.K) || (FerrTras)) && gameMode ==
GameMode.Manual)
    {
        posZFerr = posZFerr + velFerr;
        // Limite para movimentacao da ferramenta para tras
        if (posZFerr >= 0.5)
        {
            posZFerr = 0.5f;
        }
        FerrTras = false;
    }

    t = t - 1;
```

```
            if (t <= 0)
            {
                t = 0;
            }

            if (input.KeyboardState.IsKeyDown(Keys.G))
            {
                if (gameMode == GameMode.Manual)
                {
                    if (t == 0)
                    {
                        gameMode = GameMode.Automatic;
                        pause = false;
                        t = 15;
                    }
                }
                else
                {
                    if (t == 0)
                    {
                        gameMode = GameMode.Manual;
                        t = 15;
                    }
                }
            }

            teste[0].X = posXFerr;
            teste[0].Y = posYFerr;
            teste[0].Z = posZFerr;

            DiminuirRaio(teste[0]);

            if (!(previousGameState == GameState.Playing))
            {
                LoadGame();
                previousGameState = GameState.Playing;
            }

            if (gameMode == GameMode.Manual)
            {

            }
            // Se o jogo esta em modo Automatico
            if (gameMode == GameMode.Automatic)
            {
                if (interpret == true)
                    KRLInterpretor();
            }
        }
        base.Update(gameTime);
    }

    protected override void Draw(GameTime gameTime)
    {
        GraphicsDevice.Clear(Color.Gray);
        spriteBatch.Begin();

        // Se o jogo esta na tela de movimentacao da peca
```

```
            if (gameState == GameState.Playing)
            {
                Matrix world = Matrix.CreateTranslation(new Vector3(0, 0, 0));
                GraphicsDevice.DepthStencilState = new DepthStencilState() {
DepthBufferEnable = true };
                GraphicsDevice.BlendState = BlendState.Opaque;
                GraphicsDevice.DepthStencilState = DepthStencilState.Default;
                GraphicsDevice.SamplerStates[0] = SamplerState.LinearWrap;
                DrawModel(ref modelSkybox, ref world); // Desenha a skybox na tela
                DrawModel(ref modelTornoXNA, ref world); // Desenha o torno na tela
                DrawModel(ref modelArvore, ref world); // Desenha a arvore na tela
                DrawModel(ref modelEixos, ref world); // Desenha os eixos na tela
                DrawModel(ref modelCarrinho, ref world); // Desenha o carrinho na tela
                DrawModel(ref modelFerramenta, ref world); // Desenha a ferramenta na tela

                basicEffect.World = world;

                basicEffect.View = camera.View;
                basicEffect.Projection = camera.Projection;

                effect.CurrentTechnique = effect.Techniques["TexturedNoShading"];
                effect.Parameters["xWorld"].SetValue(world);
                effect.Parameters["xView"].SetValue(camera.View);
                effect.Parameters["xProjection"].SetValue(camera.Projection);
                effect.Parameters["xTexture"].SetValue(texture);

                if (posXFerr < 0.07f)
                {
                    velFerr = 0.0f;
                }

                if (pause == true)
                {
                    time = velRotacao;
                }
                else
                {
                    time = time + 1;
                }

                if (time >= velRotacao)
                {
                    if (pause == true)
                    {
                        time = velRotacao;
                    }
                    else
                    {
                        time = 0;

                        Vector3 point2;
                        Vector3 point2zero;
                        for (int i = 0; i < numLinhas; i++)
                        {
                            for (int j = 0; j < numPontos; j++)
                            {
                                point2 = new Vector3();
                                point2zero = new Vector3();
```

```
                                point2zero.X = linhas[0, j].X;
                                point2zero.Y = linhas[0, j].Y;
                                point2zero.Z = linhas[0, j].Z;

                                if (i == 35)
                                {
                                    point2.X = linhas[i, j].X;
                                    float h;

                                    point2.Y = (linhas[i, j].Y - linhas[0, j].Y) / 2 + li-
nhas[0, j].Y;
                                    point2.Z = (linhas[0, j].Z - linhas[i, j].Z) / 2 + li-
nhas[i, j].Z;
                                    h = (float)Math.Sqrt(point2.Z * point2.Z + point2.Y *
point2.Y);


                                    point2.Y = point2.Y * raio[i,j].X / h;
                                    point2.Z = point2.Z * raio[i,j].X / h;

                                    linhas[i, j] = point2;
                                }
                                else
                                {
                                    point2.X = linhas[i, j].X;
                                    float h;

                                    point2.Y = (linhas[i, j].Y - linhas[i + 1, j].Y) / 2 +
linhas[i + 1, j].Y;
                                    point2.Z = (linhas[i + 1, j].Z - linhas[i, j].Z) / 2 +
linhas[i, j].Z;
                                    h = (float)Math.Sqrt(point2.Z * point2.Z + point2.Y *
point2.Y);


                                    point2.Y = point2.Y * raio[i,j].X / h;
                                    point2.Z = point2.Z * raio[i,j].X / h;

                                    linhas[i, j] = point2;
                                }
                            }
                        }

                        for (int i = 0; i < 2*numLinhas; i++)
                        {
                            if (i % 2 == 0)
                            {
                                vetorAux[0].X = extremidadeAtualCentro;
                                vetorAux[0].Y = 0;
                                vetorAux[0].Z = 0;
                                pointListExtrem[i] = new VertexPositionColor(vetorAux[0],
Color.Blue);
                            }
                            else
                            {
                                pointListExtrem[i] = new VertexPositionColor(linhas[(i-
1)/2, extremidadeAtual], Color.Blue);
                            }
                        }
```

```
                    for (int i = 0; i < numPontos; i++)
                    {
                        pointList1[i] = new VertexPositionColor(linhas[0, i], Col-
or.Blue);
                        pointList2[i] = new VertexPositionColor(linhas[1, i], Col-
or.Blue);
                        pointList3[i] = new VertexPositionColor(linhas[2, i], Col-
or.Blue);
                        pointList4[i] = new VertexPositionColor(linhas[3, i], Col-
or.Blue);
                        pointList5[i] = new VertexPositionColor(linhas[4, i], Col-
or.Blue);
                        pointList6[i] = new VertexPositionColor(linhas[5, i], Col-
or.Blue);
                        pointList7[i] = new VertexPositionColor(linhas[6, i], Col-
or.Blue);
                        pointList8[i] = new VertexPositionColor(linhas[7, i], Col-
or.Blue);

                        pointList9[i] = new VertexPositionColor(linhas[8, i], Col-
or.Blue);
                        pointList10[i] = new VertexPositionColor(linhas[9, i], Col-
or.Blue);
                        pointList11[i] = new VertexPositionColor(linhas[10, i], Col-
or.Blue);
                        pointList12[i] = new VertexPositionColor(linhas[11, i], Col-
or.Blue);
                        pointList13[i] = new VertexPositionColor(linhas[12, i], Col-
or.Blue);
                        pointList14[i] = new VertexPositionColor(linhas[13, i], Col-
or.Blue);
                        pointList15[i] = new VertexPositionColor(linhas[14, i], Col-
or.Blue);
                        pointList16[i] = new VertexPositionColor(linhas[15, i], Col-
or.Blue);

                        pointList17[i] = new VertexPositionColor(linhas[16, i], Col-
or.Blue);
                        pointList18[i] = new VertexPositionColor(linhas[17, i], Col-
or.Blue);
                        pointList19[i] = new VertexPositionColor(linhas[18, i], Col-
or.Blue);
                        pointList20[i] = new VertexPositionColor(linhas[19, i], Col-
or.Blue);
                        pointList21[i] = new VertexPositionColor(linhas[20, i], Col-
or.Blue);
                        pointList22[i] = new VertexPositionColor(linhas[21, i], Col-
or.Blue);
                        pointList23[i] = new VertexPositionColor(linhas[22, i], Col-
or.Blue);
                        pointList24[i] = new VertexPositionColor(linhas[23, i], Col-
or.Blue);

                        pointList25[i] = new VertexPositionColor(linhas[24, i], Col-
or.Blue);
                        pointList26[i] = new VertexPositionColor(linhas[25, i], Col-
or.Blue);
```

```
                    pointList27[i] = new VertexPositionColor(linhas[26, i], Col-
or.Blue);
                    pointList28[i] = new VertexPositionColor(linhas[27, i], Col-
or.Blue);
                    pointList29[i] = new VertexPositionColor(linhas[28, i], Col-
or.Blue);
                    pointList30[i] = new VertexPositionColor(linhas[29, i], Col-
or.Blue);
                    pointList31[i] = new VertexPositionColor(linhas[30, i], Col-
or.Blue);
                    pointList32[i] = new VertexPositionColor(linhas[31, i], Col-
or.Blue);

                    pointList33[i] = new VertexPositionColor(linhas[32, i], Col-
or.Blue);
                    pointList34[i] = new VertexPositionColor(linhas[33, i], Col-
or.Blue);
                    pointList35[i] = new VertexPositionColor(linhas[34, i], Col-
or.Blue);
                    pointList36[i] = new VertexPositionColor(linhas[35, i], Col-
or.Blue);
                }

                for (int i = 0; i < numLinhas; i++)
                {
                    pointListAux1[i] = new VertexPositionColor(linhas[i, 0], Col-
or.Blue);
                    pointListAux2[i] = new VertexPositionColor(linhas[i, 1], Col-
or.Blue);
                    pointListAux3[i] = new VertexPositionColor(linhas[i, 2], Col-
or.Blue);
                    pointListAux4[i] = new VertexPositionColor(linhas[i, 3], Col-
or.Blue);
                    pointListAux5[i] = new VertexPositionColor(linhas[i, 4], Col-
or.Blue);
                    pointListAux6[i] = new VertexPositionColor(linhas[i, 5], Col-
or.Blue);
                    pointListAux7[i] = new VertexPositionColor(linhas[i, 6], Col-
or.Blue);
                    pointListAux8[i] = new VertexPositionColor(linhas[i, 7], Col-
or.Blue);
                    pointListAux9[i] = new VertexPositionColor(linhas[i, 8], Col-
or.Blue);
                    pointListAux10[i] = new VertexPositionColor(linhas[i, 9], Col-
or.Blue);
                    pointListAux11[i] = new VertexPositionColor(linhas[i, 10], Col-
or.Blue);
                    pointListAux12[i] = new VertexPositionColor(linhas[i, 11], Col-
or.Blue);
                    pointListAux13[i] = new VertexPositionColor(linhas[i, 12], Col-
or.Blue);
                    pointListAux14[i] = new VertexPositionColor(linhas[i, 13], Col-
or.Blue);
                    pointListAux15[i] = new VertexPositionColor(linhas[i, 14], Col-
or.Blue);
                    pointListAux16[i] = new VertexPositionColor(linhas[i, 15], Col-
or.Blue);
```

```
                    pointListAux17[i] = new VertexPositionColor(linhas[i, 16], Col-
or.Blue);
                    pointListAux18[i] = new VertexPositionColor(linhas[i, 17], Col-
or.Blue);
                    pointListAux19[i] = new VertexPositionColor(linhas[i, 18], Col-
or.Blue);
                    pointListAux20[i] = new VertexPositionColor(linhas[i, 19], Col-
or.Blue);
                    pointListAux21[i] = new VertexPositionColor(linhas[i, 20], Col-
or.Blue);
                    pointListAux22[i] = new VertexPositionColor(linhas[i, 21], Col-
or.Blue);
                    pointListAux23[i] = new VertexPositionColor(linhas[i, 22], Col-
or.Blue);
                    pointListAux24[i] = new VertexPositionColor(linhas[i, 23], Col-
or.Blue);
                    pointListAux25[i] = new VertexPositionColor(linhas[i, 24], Col-
or.Blue);
                    pointListAux26[i] = new VertexPositionColor(linhas[i, 25], Col-
or.Blue);
                    pointListAux27[i] = new VertexPositionColor(linhas[i, 26], Col-
or.Blue);
                    pointListAux28[i] = new VertexPositionColor(linhas[i, 27], Col-
or.Blue);
                    pointListAux29[i] = new VertexPositionColor(linhas[i, 28], Col-
or.Blue);
                    pointListAux30[i] = new VertexPositionColor(linhas[i, 29], Col-
or.Blue);
                    pointListAux31[i] = new VertexPositionColor(linhas[i, 30], Col-
or.Blue);
                    pointListAux32[i] = new VertexPositionColor(linhas[i, 31], Col-
or.Blue);
                    pointListAux33[i] = new VertexPositionColor(linhas[i, 32], Col-
or.Blue);
                    pointListAux34[i] = new VertexPositionColor(linhas[i, 33], Col-
or.Blue);
                    pointListAux35[i] = new VertexPositionColor(linhas[i, 34], Col-
or.Blue);
                    pointListAux36[i] = new VertexPositionColor(linhas[i, 35], Col-
or.Blue);
                    pointListAux37[i] = new VertexPositionColor(linhas[i, 36], Col-
or.Blue);
                    pointListAux38[i] = new VertexPositionColor(linhas[i, 37], Col-
or.Blue);
                    pointListAux39[i] = new VertexPositionColor(linhas[i, 38], Col-
or.Blue);
                    pointListAux40[i] = new VertexPositionColor(linhas[i, 39], Col-
or.Blue);
                    pointListAux41[i] = new VertexPositionColor(linhas[i, 40], Col-
or.Blue);
                    pointListAux42[i] = new VertexPositionColor(linhas[i, 41], Col-
or.Blue);
                    pointListAux43[i] = new VertexPositionColor(linhas[i, 42], Col-
or.Blue);
                    pointListAux44[i] = new VertexPositionColor(linhas[i, 43], Col-
or.Blue);
                    pointListAux45[i] = new VertexPositionColor(linhas[i, 44], Col-
or.Blue);
```

```
                    pointListAux46[i] = new VertexPositionColor(linhas[i, 45], Col-
or.Blue);
                    pointListAux47[i] = new VertexPositionColor(linhas[i, 46], Col-
or.Blue);
                    pointListAux48[i] = new VertexPositionColor(linhas[i, 47], Col-
or.Blue);
                    pointListAux49[i] = new VertexPositionColor(linhas[i, 48], Col-
or.Blue);
                    pointListAux50[i] = new VertexPositionColor(linhas[i, 49], Col-
or.Blue);

                    pointListAux51[i] = new VertexPositionColor(linhas[i, 50], Col-
or.Blue);
                    pointListAux52[i] = new VertexPositionColor(linhas[i, 51], Col-
or.Blue);
                    pointListAux53[i] = new VertexPositionColor(linhas[i, 52], Col-
or.Blue);
                    pointListAux54[i] = new VertexPositionColor(linhas[i, 53], Col-
or.Blue);
                    pointListAux55[i] = new VertexPositionColor(linhas[i, 54], Col-
or.Blue);
                    pointListAux56[i] = new VertexPositionColor(linhas[i, 55], Col-
or.Blue);
                    pointListAux57[i] = new VertexPositionColor(linhas[i, 56], Col-
or.Blue);
                    pointListAux58[i] = new VertexPositionColor(linhas[i, 57], Col-
or.Blue);
                    pointListAux59[i] = new VertexPositionColor(linhas[i, 58], Col-
or.Blue);
                    pointListAux60[i] = new VertexPositionColor(linhas[i, 59], Col-
or.Blue);
                    pointListAux61[i] = new VertexPositionColor(linhas[i, 60], Col-
or.Blue);
                    pointListAux62[i] = new VertexPositionColor(linhas[i, 61], Col-
or.Blue);
                    pointListAux63[i] = new VertexPositionColor(linhas[i, 62], Col-
or.Blue);
                    pointListAux64[i] = new VertexPositionColor(linhas[i, 63], Col-
or.Blue);
                    pointListAux65[i] = new VertexPositionColor(linhas[i, 64], Col-
or.Blue);
                    pointListAux66[i] = new VertexPositionColor(linhas[i, 65], Col-
or.Blue);
                    pointListAux67[i] = new VertexPositionColor(linhas[i, 66], Col-
or.Blue);
                    pointListAux68[i] = new VertexPositionColor(linhas[i, 67], Col-
or.Blue);
                    pointListAux69[i] = new VertexPositionColor(linhas[i, 68], Col-
or.Blue);
                    pointListAux70[i] = new VertexPositionColor(linhas[i, 69], Col-
or.Blue);
                    pointListAux71[i] = new VertexPositionColor(linhas[i, 70], Col-
or.Blue);
                    pointListAux72[i] = new VertexPositionColor(linhas[i, 71], Col-
or.Blue);
                    pointListAux73[i] = new VertexPositionColor(linhas[i, 72], Col-
or.Blue);
```

```
                    pointListAux74[i] = new VertexPositionColor(linhas[i, 73], Col-
or.Blue);
                    pointListAux75[i] = new VertexPositionColor(linhas[i, 74], Col-
or.Blue);
                    pointListAux76[i] = new VertexPositionColor(linhas[i, 75], Col-
or.Blue);
                    pointListAux77[i] = new VertexPositionColor(linhas[i, 76], Col-
or.Blue);
                    pointListAux78[i] = new VertexPositionColor(linhas[i, 77], Col-
or.Blue);
                    pointListAux79[i] = new VertexPositionColor(linhas[i, 78], Col-
or.Blue);
                    pointListAux80[i] = new VertexPositionColor(linhas[i, 79], Col-
or.Blue);
                    pointListAux81[i] = new VertexPositionColor(linhas[i, 80], Col-
or.Blue);
                    pointListAux82[i] = new VertexPositionColor(linhas[i, 81], Col-
or.Blue);
                    pointListAux83[i] = new VertexPositionColor(linhas[i, 82], Col-
or.Blue);
                    pointListAux84[i] = new VertexPositionColor(linhas[i, 83], Col-
or.Blue);
                    pointListAux85[i] = new VertexPositionColor(linhas[i, 84], Col-
or.Blue);
                    pointListAux86[i] = new VertexPositionColor(linhas[i, 85], Col-
or.Blue);
                    pointListAux87[i] = new VertexPositionColor(linhas[i, 86], Col-
or.Blue);
                    pointListAux88[i] = new VertexPositionColor(linhas[i, 87], Col-
or.Blue);
                    pointListAux89[i] = new VertexPositionColor(linhas[i, 88], Col-
or.Blue);
                    pointListAux90[i] = new VertexPositionColor(linhas[i, 89], Col-
or.Blue);
                    pointListAux91[i] = new VertexPositionColor(linhas[i, 90], Col-
or.Blue);
                    pointListAux92[i] = new VertexPositionColor(linhas[i, 91], Col-
or.Blue);
                    pointListAux93[i] = new VertexPositionColor(linhas[i, 92], Col-
or.Blue);
                    pointListAux94[i] = new VertexPositionColor(linhas[i, 93], Col-
or.Blue);
                    pointListAux95[i] = new VertexPositionColor(linhas[i, 94], Col-
or.Blue);
                    pointListAux96[i] = new VertexPositionColor(linhas[i, 95], Col-
or.Blue);
                    pointListAux97[i] = new VertexPositionColor(linhas[i, 96], Col-
or.Blue);
                    pointListAux98[i] = new VertexPositionColor(linhas[i, 97], Col-
or.Blue);
                    pointListAux99[i] = new VertexPositionColor(linhas[i, 98], Col-
or.Blue);
                    pointListAux100[i] = new VertexPositionColor(linhas[i, 99],
Color.Blue);
                }
            }
        }
```

```
                if (wireframe == false)
                {
                    foreach (EffectPass pass in effect.CurrentTechnique.Passes)
                    {
                        pass.Apply();

                        vertices = new VertexPositionTexture[6];

                        for (int m = 0; m < numLinhas; m++)
                        {
                            for (int n = 0; n < numPontos - 1; n++)
                            {
                                if (m == (numLinhas - 1))
                                {
                                    vertices[0].Position = linhas[m, n];
                                    vertices[1].Position = linhas[0, n + 1];
                                    vertices[2].Position = linhas[m, n + 1];
                                    vertices[5].Position = linhas[0, n];

                                    vertices[3].Position = vertices[1].Position;
                                    vertices[4].Position = vertices[0].Position;
                                    vertices[0].TextureCoordinate.X = 0;
                                    vertices[0].TextureCoordinate.Y = 0;
                                    vertices[1].TextureCoordinate.X = 1;
                                    vertices[1].TextureCoordinate.Y = 1;
                                    vertices[2].TextureCoordinate.X = 0;
                                    vertices[2].TextureCoordinate.Y = 1;
                                    vertices[3].TextureCoordinate.X = 1;
                                    vertices[3].TextureCoordinate.Y = 1;
                                    vertices[4].TextureCoordinate.X = 0;
                                    vertices[4].TextureCoordinate.Y = 0;
                                    vertices[5].TextureCoordinate.X = 1;
                                    vertices[5].TextureCoordinate.Y = 0;
                                    device.DrawUserPrimitives(PrimitiveType.TriangleList,
vertices, 0, 2, VertexPositionTexture.VertexDeclaration);
                                }
                                else
                                {
                                    vertices[0].Position = linhas[m, n];
                                    vertices[1].Position = linhas[m + 1, n + 1];
                                    vertices[2].Position = linhas[m, n + 1];
                                    vertices[5].Position = linhas[m + 1, n];

                                    vertices[3].Position = vertices[1].Position;
                                    vertices[4].Position = vertices[0].Position;
                                    vertices[0].TextureCoordinate.X = 0;
                                    vertices[0].TextureCoordinate.Y = 0;
                                    vertices[1].TextureCoordinate.X = 1;
                                    vertices[1].TextureCoordinate.Y = 1;
                                    vertices[2].TextureCoordinate.X = 0;
                                    vertices[2].TextureCoordinate.Y = 1;
                                    vertices[3].TextureCoordinate.X = 1;
                                    vertices[3].TextureCoordinate.Y = 1;
                                    vertices[4].TextureCoordinate.X = 0;
                                    vertices[4].TextureCoordinate.Y = 0;
                                    vertices[5].TextureCoordinate.X = 1;
                                    vertices[5].TextureCoordinate.Y = 0;
```

```
                                device.DrawUserPrimitives(PrimitiveType.TriangleList,
vertices, 0, 2, VertexPositionTexture.VertexDeclaration);
                            }
                        }
                    }

                    verticesExtrem = new VertexPositionTexture[3];
                    // Desenho da textura no topo da peça
                    for (int i = 0; i < numLinhas; i++)
                    {
                        if (i == numLinhas - 1)
                        {
                            verticesExtrem[0].Position.X = extremidadeAtualCentro;
                            verticesExtrem[0].Position.Y = 0;
                            verticesExtrem[0].Position.Z = 0;
                            verticesExtrem[1].Position = linhas[0, extremidadeAtual];
                            verticesExtrem[2].Position = linhas[i, extremidadeAtual];

                            verticesExtrem[0].TextureCoordinate.X = 0;
                            verticesExtrem[0].TextureCoordinate.Y = 0;
                            verticesExtrem[1].TextureCoordinate.X = 1;
                            verticesExtrem[1].TextureCoordinate.Y = 1;
                            verticesExtrem[2].TextureCoordinate.X = 0;
                            verticesExtrem[2].TextureCoordinate.Y = 1;

                            device.DrawUserPrimitives(PrimitiveType.TriangleList, ver-
ticesExtrem, 0, 1, VertexPositionTexture.VertexDeclaration);
                        }
                        else
                        {
                            verticesExtrem[0].Position.X = extremidadeAtualCentro;
                            verticesExtrem[0].Position.Y = 0;
                            verticesExtrem[0].Position.Z = 0;
                            verticesExtrem[1].Position = linhas[i + 1, extremidadeAtu-
al];
                            verticesExtrem[2].Position = linhas[i, extremidadeAtual];

                            verticesExtrem[0].TextureCoordinate.X = 0;
                            verticesExtrem[0].TextureCoordinate.Y = 0;
                            verticesExtrem[1].TextureCoordinate.X = 1;
                            verticesExtrem[1].TextureCoordinate.Y = 1;
                            verticesExtrem[2].TextureCoordinate.X = 0;
                            verticesExtrem[2].TextureCoordinate.Y = 1;

                            device.DrawUserPrimitives(PrimitiveType.TriangleList, ver-
ticesExtrem, 0, 1, VertexPositionTexture.VertexDeclaration);
                        }
                    }
                }
            }

            if(wireframe == true)
            {
                foreach (EffectPass pass in basicEffect.CurrentTechnique.Passes)
                {
                    pass.Apply();

                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
```

```
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListExtrem, 0, pointListExtrem.Length, lineListIndicesExtrem, 0,
pointListExtrem.Length - 1);

                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointList1, 0, pointList1.Length, lineListIndices1, 0, pointList1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointList2, 0, pointList1.Length, lineListIndices1, 0, pointList1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointList3, 0, pointList1.Length, lineListIndices1, 0, pointList1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointList4, 0, pointList1.Length, lineListIndices1, 0, pointList1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointList5, 0, pointList1.Length, lineListIndices1, 0, pointList1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointList6, 0, pointList1.Length, lineListIndices1, 0, pointList1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointList7, 0, pointList1.Length, lineListIndices1, 0, pointList1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointList8, 0, pointList1.Length, lineListIndices1, 0, pointList1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointList9, 0, pointList1.Length, lineListIndices1, 0, pointList1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointList10, 0, pointList1.Length, lineListIndices1, 0, pointList1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointList11, 0, pointList1.Length, lineListIndices1, 0, pointList1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointList12, 0, pointList1.Length, lineListIndices1, 0, pointList1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointList13, 0, pointList1.Length, lineListIndices1, 0, pointList1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointList14, 0, pointList1.Length, lineListIndices1, 0, pointList1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointList15, 0, pointList1.Length, lineListIndices1, 0, pointList1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointList16, 0, pointList1.Length, lineListIndices1, 0, pointList1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointList17, 0, pointList1.Length, lineListIndices1, 0, pointList1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointList18, 0, pointList1.Length, lineListIndices1, 0, pointList1.Length - 1);
```

```
                        GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                            (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointList19, 0, pointList1.Length, lineListIndices1, 0, pointList1.Length - 1);
                        GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                            (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointList20, 0, pointList1.Length, lineListIndices1, 0, pointList1.Length - 1);
                        GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                            (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointList21, 0, pointList1.Length, lineListIndices1, 0, pointList1.Length - 1);
                        GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                            (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointList22, 0, pointList1.Length, lineListIndices1, 0, pointList1.Length - 1);
                        GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                            (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointList23, 0, pointList1.Length, lineListIndices1, 0, pointList1.Length - 1);
                        GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                            (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointList24, 0, pointList1.Length, lineListIndices1, 0, pointList1.Length - 1);
                        GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                            (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointList25, 0, pointList1.Length, lineListIndices1, 0, pointList1.Length - 1);
                        GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                            (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointList26, 0, pointList1.Length, lineListIndices1, 0, pointList1.Length - 1);
                        GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                            (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointList27, 0, pointList1.Length, lineListIndices1, 0, pointList1.Length - 1);
                        GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                            (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointList28, 0, pointList1.Length, lineListIndices1, 0, pointList1.Length - 1);
                        GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                            (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointList29, 0, pointList1.Length, lineListIndices1, 0, pointList1.Length - 1);
                        GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                            (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointList30, 0, pointList1.Length, lineListIndices1, 0, pointList1.Length - 1);
                        GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                            (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointList31, 0, pointList1.Length, lineListIndices1, 0, pointList1.Length - 1);
                        GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                            (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointList32, 0, pointList1.Length, lineListIndices1, 0, pointList1.Length - 1);
                        GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                            (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointList33, 0, pointList1.Length, lineListIndices1, 0, pointList1.Length - 1);
                        GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                            (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointList34, 0, pointList1.Length, lineListIndices1, 0, pointList1.Length - 1);
                        GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                            (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointList35, 0, pointList1.Length, lineListIndices1, 0, pointList1.Length - 1);
                        GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                            (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointList36, 0, pointList1.Length, lineListIndices1, 0, pointList1.Length - 1);

                        GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                            (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux1, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
```

```
                        GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                            (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux2, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                        GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                            (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux3, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                        GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                            (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux4, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                        GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                            (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux5, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                        GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                            (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux6, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                        GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                            (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux7, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                        GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                            (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux8, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                        GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                            (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux9, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                        GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                            (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux10, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                        GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                            (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux11, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                        GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                            (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux12, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                        GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                            (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux13, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                        GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                            (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux14, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                        GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                            (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux15, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                        GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                            (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux16, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                        GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                            (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux17, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                        GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                            (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux18, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                        GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                            (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux19, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                        GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                            (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux20, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                        GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
```

```
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux21, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux22, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux23, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux24, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux25, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux26, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux27, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux28, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux29, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux30, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux31, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux32, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux33, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux34, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux35, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux36, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux37, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux38, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux39, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
```

```
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux40, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux41, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux42, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux43, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux44, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux45, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux46, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux47, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux48, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux49, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux50, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);

                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux51, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux52, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux53, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux54, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux55, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux56, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux57, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux58, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
```

```
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux59, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux60, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux61, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux62, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux63, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux64, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux65, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux66, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux67, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux68, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux69, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux70, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux71, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux72, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux73, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux74, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux75, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux76, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux77, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
```

```
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux78, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux79, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux80, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux81, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux82, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux83, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux84, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux85, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux86, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux87, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux88, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux89, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux90, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux91, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux92, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux93, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux94, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux95, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                        (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux96, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                    GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
```

```
                            (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux97, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                        GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                            (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux98, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                        GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                            (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux99, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                        GraphicsDevice.DrawUserIndexedPrimitives<VertexPositionColor>
                            (Microsoft.Xna.Framework.Graphics.PrimitiveType.LineStrip,
pointListAux100, 0, pointListAux1.Length, lineListIndices2, 0, pointListAux1.Length - 1);
                        }
                        GraphicsDevice.DepthStencilState = new DepthStencilState() {
DepthBufferEnable = true };
                    }
                }

                // Se em modo automatico
                if (gameMode == GameMode.Automatic)
                {
                    // Desenho das imagens 2D
                    spriteBatch.Draw(barraCodigoG, new Vector2(710, 0), Color.White);
                    spriteBatch.Draw(Iautomatico, new Rectangle(710, 0, 314, 288), Col-
or.White);
                    krl = new KRLManager();
                    // Texto do codigo na tela
                    spriteBatch.DrawString(font, krl.treeString, new Vector2(720, 320), Col-
or.Black);
                }
                if (gameMode == GameMode.Manual && gameState == GameState.Playing)
                {
                    // Desenho das imagens 2D
                    spriteBatch.Draw(Imanual, new Rectangle(710, 0, 314, 288), Color.White);
                }

                // Se o jogo esta na tela inicial, desenha as imagens necessarias
                if (gameState == GameState.StartMenu)
                {
                    spriteBatch.Draw(telainicial, new Vector2(90,63), Color.White);
                }

                spriteBatch.End();
                RasterizerState state = new RasterizerState();
                state.CullMode = CullMode.None;
                graphics.GraphicsDevice.RasterizerState = state;
                base.Draw(gameTime);
            }

            // Função que lida com o clique do mouse
            void MouseClicked(int x, int y)
            {
                // Cria um retangulo ficticio na area onde o mouse foi clicado
                Rectangle mouseClickRect = new Rectangle(x, y, 10, 10);

                // Na tela inicial
                if (gameState == GameState.StartMenu)
                {
                    Rectangle startButtonRect = new Rectangle((int)startButtonPosition.X,
```

```
                (int)startButtonPosition.Y, 200, 70);
            Rectangle exitButtonRect = new Rectangle((int)exitButtonPosition.X,
                (int)exitButtonPosition.Y, 150, 65);

            if (mouseClickRect.Intersects(StartRect))
            {
                gameState = GameState.Playing;
            }
            else if (mouseClickRect.Intersects(ExitRect))
            {
               Exit();
            }
        }
        // Na tela de movimentacao
        if (gameState == GameState.Playing)
        {
            if (mouseClickRect.Intersects(FerrTrasRect))
            {
               FerrTras=true ;
            }

            if (mouseClickRect.Intersects(FerrFrenteRect))
            {
                FerrFrente = true;
            }

            if (mouseClickRect.Intersects(FerrDireitaRect))
            {
                FerrDireita = true;
            }

            if (mouseClickRect.Intersects(FerrEsquerdaRect))
            {
                FerrEsquerda = true;
            }
        }
        // Se em modo automatico
        if (gameMode == GameMode.Automatic)
        {
            if (mouseClickRect.Intersects(loadRect))
            {
                interpret = true;
            }
        }
    }

    // Funcao para desenhar na tela os objetos tridimensionais
    private void DrawModel(ref Model m, ref Matrix world)
    {
        m.CopyAbsoluteBoneTransformsTo(transformTornoXNA);
        foreach (ModelMesh mesh in m.Meshes)
        {
            foreach (BasicEffect e in mesh.Effects)
            {
                e.EnableDefaultLighting();
                e.World = transformTornoXNA[mesh.ParentBone.Index] * world;
                e.View = camera.View;
                e.Projection = camera.Projection;
```

```
                    e.AmbientLightColor = Color.LightBlue.ToVector3();
                }
                GraphicsDevice.SamplerStates[0] = SamplerState.LinearClamp;

                mesh.Draw();
            }
        }

        private int RaioMaisProximo(Vector3 position)
        {
            for (int i = 0; i < numPontos; i++)
            {
                if (Math.Abs(position.X - linhas[0, i].X) < 0.005f)
                {
                    return i;
                }
            }
            return -1;
        }

        public void DiminuirRaio(Vector3 position)
        {
            int index = this.RaioMaisProximo(position);

            if (index > -1)
            {
                for (int i = 0; i < numLinhas; i++)
                {
                    float distanciaFerr = (float)Math.Sqrt(Math.Pow((position.Y), 2) +
Math.Pow((position.Z), 2));
                    if (distanciaFerr < raio[i,index].X) // Indica colisao
                    {
                        if (linhas[i, index].Z >= position.Z - 0.005f && linhas[i, index].Z
<= position.Z + 0.005f && linhas[i, index].Y > 0)
                        {
                            raio[i, index].X = distanciaFerr;
                        }
                    }
                }
            }
        }

        // Interpretacao do Código G
        private void KRLInterpretor()
        {
            krl = new KRLManager();
            if (krl.tree.GetChild(numNode) != null)
            {
                // A velocidade de usinagem que permite que a peca fique continua é
0.00004f
                velRotacao = velRotAlta;
                rotacaoArvore = rotArvoreAlta;

                switch (krl.tree.GetChild(numNode).Text)
                {
                    case "G00":
                        velUsin = 0.00004f;
                        //Se o movimento for composto i.e. em X e Z ao mesmo tempo
```

```
                        if (krl.tree.GetChild(numNode + 1).Text == "X" &&
krl.tree.GetChild(numNode + 5).Text == "Z")
                        {
                            // Reconhece as coordenadas finais de X e Z
                            auxX = float.Parse(krl.tree.GetChild(numNode + 2).Text) * 100 +
float.Parse(krl.tree.GetChild(numNode + 3).Text) * 10 + flo-
at.Parse(krl.tree.GetChild(numNode + 4).Text);
                            auxX = auxX / 1000;

                            auxZ = float.Parse(krl.tree.GetChild(numNode + 6).Text) * 100 +
float.Parse(krl.tree.GetChild(numNode + 7).Text) * 10 + flo-
at.Parse(krl.tree.GetChild(numNode + 8).Text);
                            auxZ = auxZ / 1000;

                            distanciaX = Math.Abs(posZFerr - auxX);
                            distanciaZ = Math.Abs(posXFerr - auxZ);

                            if (distanciaX < distanciaZ && posXFerr >= 0.08f)
                            {
                                ratioVelocidades = distanciaX / distanciaZ;

                                if (posZFerr > auxX && posXFerr > auxZ)
                                {

                                    posZFerr = posZFerr - velUsin * ratioVelocidades;
                                    posXFerr = posXFerr - velUsin;
                                    posXCarrinho = posXCarrinho - velUsin;

                                    if (posXFerr <= auxZ || posZFerr <= auxX) numNode++;
                                }

                                if (posZFerr < auxX && posXFerr < auxZ)
                                {
                                    posZFerr = posZFerr + velUsin * ratioVelocidades;
                                    posXFerr = posXFerr + velUsin;
                                    posXCarrinho = posXCarrinho + velUsin;

                                    if (posXFerr >= auxZ || posZFerr >= auxX) numNode++;
                                }

                                if (posZFerr < auxX && posXFerr > auxZ)
                                {
                                    posZFerr = posZFerr + velUsin * ratioVelocidades;
                                    posXFerr = posXFerr - velUsin;
                                    posXCarrinho = posXCarrinho - velUsin;

                                    if (posXFerr <= auxZ || posZFerr >= auxX) numNode++;
                                }

                                if (posZFerr > auxX && posXFerr < auxZ)
                                {
                                    posZFerr = posZFerr - velUsin * ratioVelocidades;
                                    posXFerr = posXFerr + velUsin;
                                    posXCarrinho = posXCarrinho + velUsin;

                                    if (posXFerr >= auxZ || posZFerr <= auxX) numNode++;
                                }
                            }
```

```
                    else
                    {
                        if(posXFerr>=0.08f)
                        {

                        ratioVelocidades = distanciaZ / distanciaX;

                        if (posZFerr > auxX && posXFerr > auxZ )
                        {
                            posZFerr = posZFerr - velUsin * ratioVelocidades;
                            posXFerr = posXFerr - velUsin;
                            posXCarrinho = posXCarrinho - velUsin;

                            if (posXFerr <= auxZ || posZFerr <= auxX) numNode++;
                        }

                        if (posZFerr < auxX && posXFerr < auxZ )
                        {
                            posZFerr = posZFerr + velUsin * ratioVelocidades;
                            posXFerr = posXFerr + velUsin;
                            posXCarrinho = posXCarrinho + velUsin;

                            if (posXFerr >= auxZ || posZFerr >= auxX) numNode++;
                        }

                        if (posZFerr < auxX && posXFerr > auxZ)
                        {
                            posZFerr = posZFerr + velUsin * ratioVelocidades;
                            posXFerr = posXFerr - velUsin;
                            posXCarrinho = posXCarrinho - velUsin;

                            if (posXFerr <= auxZ || posZFerr >= auxX) numNode++;
                        }

                        if (posZFerr > auxX && posXFerr < auxZ)
                        {
                            posZFerr = posZFerr - velUsin * ratioVelocidades;
                            posXFerr = posXFerr + velUsin;
                            posXCarrinho = posXCarrinho + velUsin;

                            if (posXFerr >= auxZ || posZFerr <= auxX) numNode++;
                        }
                    }
                }
                }
                //Se o movimento for simples, um eixo por vez
                else
                {

                    if (krl.tree.GetChild(numNode + 1).Text == "X")
                    {
                        auxX = float.Parse(krl.tree.GetChild(numNode + 2).Text) *
100 + float.Parse(krl.tree.GetChild(numNode + 3).Text) * 10 +
float.Parse(krl.tree.GetChild(numNode + 4).Text);
                        auxX = auxX / 1000;

                        if (posZFerr < auxX )
```

```
                                {
                                    posZFerr = posZFerr + velUsin;
                                    if (posZFerr >= auxX) numNode++;
                                }
                                if (posZFerr > auxX)
                                {

                                    posZFerr = posZFerr - velUsin;
                                    if (posZFerr <= auxX) numNode++;
                                }
                            }

                            if (krl.tree.GetChild(numNode + 1).Text == "Z" &&
posXFerr>=0.08f)
                            {
                                auxZ = float.Parse(krl.tree.GetChild(numNode + 2).Text) *
100 + float.Parse(krl.tree.GetChild(numNode + 3).Text) * 10 +
float.Parse(krl.tree.GetChild(numNode + 4).Text);
                                auxZ = auxZ / 1000;

                                if (posXFerr < auxZ )
                                {
                                        posXFerr = posXFerr + velUsin;
                                        posXCarrinho = posXCarrinho + velUsin;


                                    if (posXFerr >= auxZ) numNode++;
                                }
                                if (posXFerr > auxZ)
                                {

                                        posXFerr = posXFerr - velUsin;
                                        posXCarrinho = posXCarrinho - velUsin;
                                        if (posXFerr <= auxZ) numNode++;
                                }
                            }
                        }
                        break;

                    case "G01":

                        auxX = float.Parse(krl.tree.GetChild(numNode + 2).Text) * 100 +
float.Parse(krl.tree.GetChild(numNode + 3).Text) * 10 +
float.Parse(krl.tree.GetChild(numNode + 4).Text);
                        auxX = auxX / 1000;

                        auxZ = float.Parse(krl.tree.GetChild(numNode + 6).Text) * 100 +
float.Parse(krl.tree.GetChild(numNode + 7).Text) * 10 +
float.Parse(krl.tree.GetChild(numNode + 8).Text);
                        auxZ = auxZ / 1000;

                        auxPasso = float.Parse(krl.tree.GetChild(numNode + 10).Text) * 100
+ float.Parse(krl.tree.GetChild(numNode + 11).Text) * 10 +
float.Parse(krl.tree.GetChild(numNode + 12).Text);
                        auxPasso = auxPasso / 1000;

                        velUsin = auxPasso / fatorCorrecaoVelocidades * 0.016f;
```

```
                        if (posZFerr >= auxX)
                        {
                            posZFerr = posZFerr - velUsin;
                            if (posZFerr <= auxX) xpronto = true;
                        }
                        if (posZFerr < auxX)
                        {
                            posZFerr = posZFerr + velUsin;
                            if (posZFerr >= auxX) xpronto = true;
                        }

                        if (posXFerr < auxZ && xpronto == true)
                        {
                            posXFerr = posXFerr + velUsin;
                            posXCarrinho = posXCarrinho + velUsin;
                            if (posXFerr >= auxZ)
                            {
                                numNode++;
                                xpronto = false;
                                velUsin = 0.00007f;
                            }
                        }
                        if (posXFerr > auxZ && xpronto == true)
                        {
                            posXFerr = posXFerr - velUsin;
                            posXCarrinho = posXCarrinho - velUsin;
                            if (posXFerr <= auxZ)
                            {
                                numNode++;
                                xpronto = false;
                                velUsin = 0.00007f;
                            }
                        }

                        break;

                    default:
                        numNode++;

                        break;
                }
            }
            else
            {
                interpret = false;
            }
        }
    }
}
```