DIEGO ADDAN GONÇALVES

AVATAR 3D PARA SÍNTESE AUTOMÁTICA DE SINAIS DA LÍNGUA DE SINAIS BRASILEIRA

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Eduardo Todt

CURITIBA

2012

DIEGO ADDAN GONÇALVES

AVATAR 3D PARA SÍNTESE AUTOMÁTICA DE SINAIS DA LÍNGUA DE SINAIS BRASILEIRA

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Eduardo Todt

CURITIBA

2012

SUMÁRIO

RESUMO **iii**

ABSTRACT **iv**

1 INTRODUÇÃO **1**

1.1 Objetivos 3

1.2 Organização do Trabalho 5

2 TRABALHOS RELACIONADOS **6**

2.1 Representação Formal de Línguas de Sinais 6

2.2 Avatares 3D e e Ferramentas que Utilizam Agentes Virtuais 8

3 DESENVOLVIMENTO DO AVATAR 3D E INTERPRETAÇÃO DO MODELO FORMAL **13**

3.1 Modelo Descritivo da Língua de Sinais Brasileira 13

3.2 Modelagem e animação de um Avatar 3D 16

3.3 Mapeamento e Texturização do Avatar 3D 24

3.4 Interpretação do modelo formal descrito em XML e transcrição para o agente virtual 27

4 SÍNTESE AUTOMÁTICA DA LÍNGUA DE SINAIS BRASILEIRA COM O AVATAR DESENVOLVIDO **37**

4.1 VPC - Vetor de Parâmetros de Configuração 37

4.2 Síntese Automática Através do Avatar 3D 45

5 RESULTADOS E DISCUSSÃO **54**

5.1 Testes e Validações 54

5.2 Análise e Discussão 57

6 CONCLUSAO **59**

REFERÊNCIAS BIBLIOGRÁFICAS **62**

ANEXO 1 **67**

RESUMO

A Libras, Língua de Sinais Brasileira, utilizada por pessoas com deficiência auditiva, tem impulsionado estudos nos mais diversos campos. Na área de tecnologia de informação não é diferente. Para pessoas com esta deficiência, sistemas com representação virtual podem ter seu uso em vários aspectos do cotidiano, servindo de ferramenta para acesso a informações. Os softwares que utilizam um ambiente virtual voltado para línguas de sinais existentes, contudo, são baseados em sinais indivisíveis e não em parâmetros de um modelo computacional, ou mesmo soletração. Assim, os sistemas ficam limitados a uma lista de palavras, ou sinais, prontos. Além disso, muitos destes softwares são comerciais, e possuem suas bases limitadas, não oferecendo a possibilidade e incremento de elementos na base de conhecimento.

Este trabalho apresenta uma nova abordagem na construção de um sistema que utiliza um Avatar 3D para representação da Libras, utilizando entrada baseada em um modelo formal e gerando sinais automaticamente através de um motor gráfico, baseado em parâmetros gerais como as posições e articulações utilizadas e não em sinais completos, possuindo ainda a vantagem de permitir a inclusão de novos sinais e parâmetros. São apresentados conceitos e métodos para criação de um Avatar 3D, o estudo de modelos computacionais para Línguas de Sinais e por fim foi desenvolvido um sistema de síntese que obtem, a partir de uma entrada externa de dados, um vetor de parâmetros de configuração referentes a elementos de sinais da Libras que são representados pelo Avatar 3D automaticamente.

Com isto, este trabalho contribui para o desenvolvimento de um sistema para síntese de sinais em um ambiente virtual sem as limitações apresentadas nos sistemas atuais, contribuindo assim para o desenvolvimento de novas tecnologias para a comunidade surda abordando os conceitos corretos. Os experimentos descritos no trabalho validam o modelo proposto, o qual se apresenta como alternativa promissora na área de síntese de sinais para uso computacional de linguagens como a Libras.

ABSTRACT

The Libras, Brazilian Signal Language, used by deaf people, has driven studies in various fields. In the field of technology of Information is not diferente. For people with this disability, systems with virtual representation may have its use in various aspects daily, serving to tool for accessing information. The software using a virtual environment, oriented for languages existing signals, however, are based on indivisible signals and not in parameters of a computational model. Thus, the systems are dependent on a list of words or signs, ready, and depend on the interaction of a user to select a complete sentence which should represent for the avatar in the virtual environment. Moreover many of these software are commercial, and have limited bases, not offering the possibility of increase the knowledge base.

This work presents a new approach to building a system that uses a 3D Avatar for Libras representation using input based on a formal model and automatically generating signals through a graphics engine, general parameters based on the positions and joints used and not on complete signals having the additional advantage of allowing the inclusion of new signals and parameters. Presents concepts and methods for creating a 3D Avatar, the study of computational models for Sign Languages and finally presents a synthesis system that obtains, from an external input data, an array of configuration parameters related to elements which are signs of Libras represented by the 3DAvatar automatically.

With this, the work contributes to the development of a system for synthesizing signals in a virtual environment without the limitations presented in current systems, contributing to the development of new technologies for the deaf community covering the correct concepts. The experiments described in the study validate the proposed model, which is presented as a promising alternative in the area of synthesis of signals for computational use of languages like a Libras.

CAPÍTULO 1

INTRODUÇÃO

A Língua de Sinais Brasileira, ou Libras [32], oficializada no Brasil com o decreto de Lei n° 10.436/2002, possui níveis e regras formais (morfológicos, fonológicos, sintáticos, semânticos e pragmáticos), sendo uma ferramenta imprescindível para a inclusão social de pessoas com deficiência auditiva ou surdos, que constituem uma comunidade com aspectos culturais e linguísticos próprios de importância inegável [4]. O que é denominado de palavra ou item lexical nas línguas oral-auditivas é denominado sinal nas línguas de sinais, sendo que o que diferencia as Línguas de Sinais das demais línguas é a sua modalidade visual-espacial [37]. Embora seja legalmente oficial, a Libras não tem seu uso tão frequente como poderia nos diversos âmbitos da sociedade, como a educação e no meio profissional, pois o número de pessoas sem deficiência que dominam a linguagem de sinais é pequeno.

Segundo estatísticas do IBGE [16], no Brasil existem 5,7 milhões de pessoas portadoras de deficiência auditiva, sendo que estas têm direito de acesso ao ensino da Libras desde seu ingresso a instituições de ensino. Contudo, o que se observa é que muitas instituições ainda não estão preparadas para receber este tipo de aluno, e as que tem o ensino de libras incluído na grade ainda oferecem o conteúdo apenas para pessoas surdas ou deficientes auditivos em qualquer grau. O resultado disto é que o ensino da Libras fica restrito aos que realmente precisam aprender esta linguagem, deficientes auditivos ou quem trabalha com eles, e a grande maioria dos alunos não são motivados a aprender a língua de sinais.

Ainda, o uso da Libras em outros setores da sociedade, como na área corporativa, apresenta uma carência ainda mais evidente, visto que surdos e deficientes auditivos utilizam frequentemente locais como bancos e hospitais e contam com ferramentas específicas para sua classe em poucas ocasiões, não encontrando muitas vezes nem mesmo um intérprete humano para lhe auxiliar.

Nesse sentido, sistemas computacionais podem ser de grande importância apresentando

suportes específicos para diversos tipos de deficiência. Estudos nessa área apresentaram resultados interessantes como o *software MATRACA* [8] ou o *DosVox* [17], ambos para auxiliar deficientes visuais. No caso de pessoas surdas um grande problema é que as ferramentas de *software* disponíveis especificamente para esta classe ainda são poucas, e em alguns casos utilizam em sua interface a linguagem escrita tradicional, inserindo a língua de sinais somente em momentos de interação com o usuário. Estudos na área, no sentido de ampliar as ferramentas para acesso de pessoas surdas ainda mostram um grande desafio para pesquisadores. A tendência é que os estudos e pesquisas para estas ferramentas aumentem, já que a gama de possibilidades para inserção destas é enorme.

A construção de um sistema interpretador entre a língua portuguesa tradicional e a língua de sinais brasileira depende da definição de entradas, como reconhecimento em vídeo de gestos ou de símbolos escritos, neste caso o *SignWriting* (modelo de representação escrita da Libras) [36], e das saídas, como uma representação através de um avatar 3D. No caso das saídas é necessário ter uma base, ou modelo descritivo, dos aspectos fonológicos da Libras e posteriormente um sistema capaz de interpretar estes dados.

Como representado na Figura 1.1, caso a entrada de dados seja um vídeo, o sistema pode fazer o reconhecimento dos sinais e fornecer os dados ao modelo descritivo que gera a interpretação em *SignWriting*, português escrito tradicional ou qualquer forma de representação escrita. Se a entrada for em dados de escrita, como o *SignWriting*, ou por representação descritiva, o sistema pode fazer a síntese para gerar a interpretação através de um avatar 3D.

Estudos para pesquisa e desenvolvimento de sistemas que utilizam agentes virtuais para interpretação gestual ainda são escassos, faltam investimento e motivação para impulsionar pesquisas neste sentido. Um exemplo disso é no âmbito social, onde recursos que facilitem o uso da Libras são fundamentais, por exemplo, em ambientes específicos como hospitais, que poderiam oferecer instruções importantes, tais como avisos de emergência através de um agente virtual, oferecendo aos surdos a mesma agilidade na transmissão da informação. Ou em setores privados, como em bancos, onde o uso de um agente virtual interpretador da Libras poderia ser utilizado também para transmissão de informações

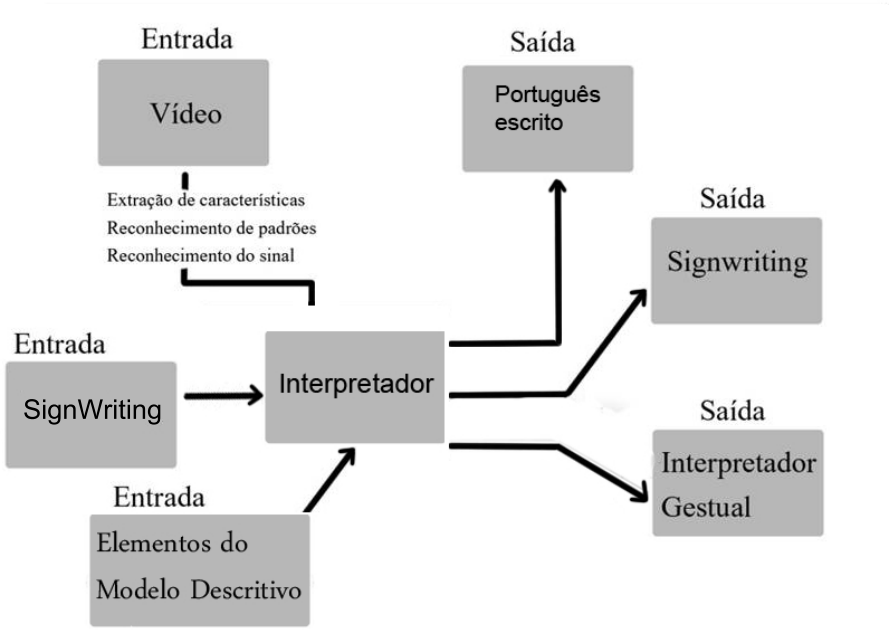


Figura 1.1: Esquema para interpretador de Lingua de Sinais - Figura do Autor

para surdos ou deficientes auditivos. Mais exemplos de uso destes sistemas podem ser encontrados na próxima seção.

1.1 Objetivos

Com este cenário, observa-se que as ferramentas de auxílio da representação visual da Libras, através de um Avatar 3D, existentes [43] [48] [41] [19], descritos no próximo capítulo, contam com recursos não automáticos e limitados e não geram movimentos automaticamente, tendo o usuário que arrastar uma lista de movimentos até o agente virtual, para então a animação ser gerada. Levando em conta os modelos formais apresentados na literatura, este trabalho busca:

i - estudar o desenvolvimento de um serviço que seja de representação gestual automática;

ii - contribuir para o desenvolvimento de um software para o usuário, com múltiplos usos, como um dicionário virtual da Libras, que poderia ser utilizado não só no âmbito acadêmico, mas no cotidiano tanto por pessoas que precisem lidar com deficientes auditivos como para quem quer se informar a respeito de determinado sinal ou frase;

Este trabalho tem como objetivos específicos:

a) estudar um modelo de agente virtual funcional e otimizado para representação da

Libras;

b) definir a representação animada de um sinal em Libras por m Avatar 3D, através de um modelo descritivo paramétrico;

c) estudar as integrações das ferramentas necessárias para o desenvolvimento de um protótipo que implemente o item anterior;

d) contribuir no cenário acadêmico ao desenvolvimento de um serviço para síntese automática de sinais da Língua de Sinais Brasileira por meio de um avatar 3D.

A Figura 1.2 mostra os blocos de processos estudados neste trabalho, que iniciam a partir de um modelo formal descritivo dos sinais, passando pela implementação, que deve fazer a tradução das entradas baseadas no modelo descritivo, produzindo uma saída através do avatar 3D.

Figura 1.2: Serviço de Síntese Automática de Sinais da Libras - Figura do Autor

O cenário que mais se beneficia com ferramentas deste tipo, contudo, ainda são os ambientes educacionais. O uso de um recurso visual no ensino de uma língua de sinais é indispensável, já que trata-se de uma linguagem puramente visual.

Um exemplo de uso da ferramenta nesta área seria em livros interativos. Com a popularização de dispositivos móveis como tablets, a tendência é que estes aparelhos substituam cada vez mais o livro tradicional ou caderno, já que a tendência é a digitalização de materiais impressos e criação de livros diretamente nesse formato. Assim, não só aplicativos em português, mas recursos visuais em geral ganham mais espaço no meio educacional e acadêmico. É facil imaginar então, um livro interativo, reproduzido em Libras gestual.

Outro exemplo de aplicação relevante é para uso de *Closed Caption*. Hoje um dos

recursos mais utilizados por deficientes auditivos são as legendas ocultas, chamadas de *Closed Caption*, no qual os aparelhos, tipicamente televisores, exibem uma legenda especial com representação descritiva de sons, falas e músicas. Com o uso do avatar 3D os surdos poderiam se beneficiar em dinamismo, proporcionando um recurso a mais para sua classe.

Os estudos para o desenvolvimento deste trabalho têm base a partir da arquitetura IHC-SL [2], modelo formal descritivo da língua de sinais brasileira.

Todos os conceitos, métodos e resultados obtidos apresentados neste trabalho intencionam a criação de um *software* livre, com repositório para contribuição colaborativa. Espera-se que seja possível incluir novas funcionalidades e novos parâmetros.

A premissa deste trabalho é que o ideal para um *software* com este propósito é mapear as configurações possíveis e gerar movimentos por classes independentes. Além de proporcionar maior facilidade na inclusão de novos movimentos ou sinais, desta maneira é possível gerar a animação sem que usuário precise ficar controlando cada parâmetro manualmente.

1.2 Organização do Trabalho

Organização do trabalho a partir deste ponto: no capítulo 2 é apresentada uma revisão de literatura sobre descrição formal da linguagem de sinais gestuais e algumas das propostas mais relevantes de trabalhos neste sentido. Ainda são apresentados trabalhos e técnicas referentes a construção de um avatar 3D, bem como modelagem e animação em 3D, além do uso de sistemas computacionais no ensino da Libras. No capítulo 3 é proposto um estudo para serviço voltado à síntese automática de sinais da Língua de Sinais Brasileira. No capítulo 4 é apresentada a criação de um sistema de síntese para representação da Língua de Sinais Brasileira através de um Avatar 3D. Os capítulos seguintes apresentam os resultados obtidos, validações, e conclusões, bem como as referências bibliográficas.

CAPÍTULO 2

TRABALHOS RELACIONADOS

Este capítulo tem por finalidade expor trabalhos relacionados e conceitos utilizados no desenvolvimento desta pesquisa. São abordados conceitos referentes a línguas de sinais, modelos de descrição computacional dos aspectos da fonologia dos sinais de línguas de sinais, ferramentas de interpretação textual, bem como trabalhos relacionados à modelagem e animação 3D e definição de um avatar 3D, além de sistemas computacionais para representação de línguas de sinais.

2.1 Representação Formal de Línguas de Sinais

Como todas as formas de linguagem formais, a Libras possui características e propriedades que a tornam única [2]. Também é correto dizer que a Libras não tem uma quantidade definida de sinais e movimentos, já que, assim como toda linguagem, está em constante transformação, se adaptando a novos termos, incluindo palavras novas ou excluindo palavras obsoletas.

A primeira arquitetura visual de representação esquemática dos sinais, baseada na ASL *American Sign Language*, foi apresentada em 1983, e já apresentava conceitos de animação a partir em um esqueleto com movimentos baseados na modelagem formal [26].

Stokoe [39] defendeu que as línguas de sinais são naturais, dispondo de todos os critérios lingüísticos de uma língua em sua sintaxe, no aspecto léxico e na capacidade de gerar um número infinito de sentenças. Ainda, afirma que os sinais são símbolos complexos e abstratos constituídos por diversas partes [44].

Analisando os sinais da língua de sinais americana (ASL), Stokoe comprovou que os sinais são compostos de três parâmetros independentes, com números limitados de combinações, que são o movimento, a configuração de mãos e a locação. Movimento é o deslocamento das mãos no espaço, como por exemplo no sinal peixe, onde o movimento

não é em linha reta mas sim em curvas. Configuração de mãos, é a posição que a mão e os dedos assumem no sinal, e locação, é a posição dos membros em relação ao corpo.[6].

Outros autores como Kilma e Bellugi [20] descreveram um outro parâmetro, que é a orientação da palma da mão. Em seus estudos são apresentados exemplos de pares de sinais compostos pelos três parâmetros originais de Stokoe que se diferenciavam a partir da orientação da mão.

Quadros e Karnopp [33] citam que o parâmetro de movimento é definido como a ação das mãos no espaço em torno do enunciador, sendo que as direções podem ser significativas na gramática da língua de sinais. O parâmetro de movimento é caracterizado quanto ao tipo, sentido da direção, frequência e maneira. Tipo é a forma do movimento, por exemplo um movimento de contato com outro membro do corpo, ou de contorno, como um movimento circular. Sentido de direção, por exemplo, pode ser para cima ou para baixo. A maneira pode representar a tensão, por exemplo a força do movimento e a frequência pode representar uma repetição.

Durante a realização do sinal a orientação é a direção para que a palma da mão aponta, que pode ser para cima, para baixo, para o corpo, para frente, para os lados (direita e esquerda) e para o receptor (dorso da mão para frente) [2].

Ainda em [2], os tipos de movimentos na Libras são descritos em tipo, direcionalidade, maneira e frequência, que na prática funcionam de maneira similar ao apresentado em [33]. Os autores dividem o elemento sinal em classes organizadas e dependentes, de forma hierárquica,apresentando um modelo formal descritivo da Libras para uso computacional.

Amaral [1] propõe uma notação XML para a língua de sinais brasileira, baseada em um modelo descritivo próprio. Nele a autora sugere que o elemento sinal é a raiz de qualquer modelo descritivo de língua de sinais, assumindo que um sinal pode ter movimentos globais e locais. Sugere ainda uma cadeia de posições e classes organizadas hierarquicamente para cobrir os principais movimentos possíveis na Libras. Este modelo formal segue a mesma idéia apresentada em [2] de cobrir os movimentos utilizados nos sinais formalizando por meio de classes, representando de maneira formal a língua de sinais para uso computacional.

Sagawa e Takeuchi utilizam um modelo descritivo semelhante aos citados anteriormente, com representação XML. Neste modelo as partes de interesse como boca, olhos e mãos são representados separadamente e independentes [35].

2.2 Avatares 3D e e Ferramentas que Utilizam Agentes Virtuais

A partir destes modelos formais ou outras formas de representação das línguas de sinais, alguns *softwares* de simulação de sinais das línguas de sinais foram desenvolvidos, para fins acadêmicos ou comerciais. A maior parte destes *softwares* dependem de um agente virtual, que é um avatar 3D, para simular os sinais. Como exemplos destes *softwares* podem ser citados o *Gesture Builder* da *VCom* [41], o *Max* da *Einfach Teilhaben* [40] e o *Sign 4 Me* [42].

Gibet [12] foi um dos primeiros desenvolvedores a integrar o uso do 3D para animação de uma língua de sinais. Seu trabalho apresentava um braço 3D com duas juntas. Dois anos depois Geitz [11] propôs um modelo de configuração de dedos que descrevia algumas palavras, letra por letra, conceituando a idéia de uma mão animada que poderia ser acessada através da internet utilizando VRML.

Um exemplo de trabalho relacionado à área da saúde que utiliza animação 3D e agente virtual pode ser visto em [28], utilizado na reabilitação de membros superiores, onde a simulação virtual ajuda o paciente a estimular os movimentos dos braços. Neste caso, assim como na simulação de línguas de sinais, o avatar 3D tem o foco no movimento do braço. Foram utilizados sensores eletromagnéticos acoplados a uma luva, assim que o paciente movimenta o braço, os sensores capturam este movimento e reproduzem no avatar 3D apresentado no ambiente virtual. Os autores utilizaram as ferramentas do Blender, ferramenta de código aberto utilizada para modelagem e animação e criação de aplicações interativas em 3D [3], para criação e animação do avatar 3D, e a *Blender Game Engine* (BGE), motor de jogo próprio do *Blender* utilizado para aplicações interativas em 3D, [3] para as simulações físicas, além da *Python API* integrada ao *Blender*.

Chadwick et al. [6] propõem um modelo para criação de um corpo animável, humano ou não, baseado em três camadas: esqueleto, músculos e pele. As principais ferramentas

do mercado de computação gráfica trabalham com um sistema parecido com este, uma vez que apenas a camada de músculos não se mostra indispensável para se obter um modelo animável completo.

Kitaguchi e Saito [22] apresentam um método para modelagem *Low Poly* (um objeto tridimensional formado por poucos polígonos) utilizando um escaner que captura a imagem de um determinado objeto e retorna uma extração de vértices. Neste caso é construída uma malha aproximando o contorno e as cores do objeto com base na lista de vértices que o dispositivo retorna. Esta técnica de modelagem, no entanto, é inviável de ser amplamente utilizada por depender de um *hardware* específico.

Segundo Sagawa e Takeuchi [35] *softwares* para fins didáticos que utilizam animação 3DCG (ou 3D no contexto de Computação Gráfica) são excelentes para o ensino de língua gestual. Os autores ainda sugerem um sistema para uso no ensino de língua de sinais japonesa, que utiliza um dispositivo de entrada acoplado a uma luva. O usuário reproduz o sinal e o sistema faz o processamento e reconhecimento deste sinal, gerando a saída em vídeo. Como base para aprendizado o sistema utiliza um modelo formal representado por XML. A ideia é que o professor e o aluno possam observar a reprodução do sinal por meio do agente virtual e identificarem onde o movimento estava errado.

Hruz et al. [15] sugere que há duas formas de se utilizar um avatar sinalizador, que se trata de um modelo tridimensional utilizado para representar sinais específicos. A primeira é com geração de movimentos em tempo real para os usuários surdos e a segunda maneira é a exibição de vídeos curtos predefinidos inseridos em uma interface gráfica, em geral cada vídeo representando uma palavra ou frase simples.

A maior parte das ferramentas ou protótipos existentes utiliza uma base de conhecimento com sinais previamente montados, formado por palavras completas que são chamadas pelo usuário. Destas, as mais relevantes atualmente são apresentadas a seguir.

O sistema proposto em Yi et al. [48] possui uma interface que oferece a opção para o usuário com uma lista de palavras e, dependendo da escolha, reproduz o movimento através de uma mão virtual.

Kaneko et al. [19] propõem uma animação da língua de sinais japonesa utilizando a

TVML [30], Ferramenta para representação em computação gráfica de ambientes simulados que incorpora conceitos tradicionais da produção de programas de TV, exibindo uma representação em computação gráfica, com um cenário virtual, bem como avatares 3D controláveis através de um *script* próprio, onde o usuário pode determinar as palavras, movimentos, ações, expressões faciais e manuais das personagens 3D incluídas na cena.

Ainda em [15], os autores desenvolveram um ambiente para gravação dos movimentos corpóreos e faciais baseado em de três câmeras, sendo uma especialmente para gravação das expressões faciais. O usuário (devidamente posicionado) faz determinados gestos, então um algoritmo retorna a posição do rosto e mão. Tendo esta posição, a ideia dos autores é transpor isso para um avatar 3D. Este trabalho, contudo, não apresenta uma implementação para esta etapa, apresentando apenas conceitos.

Krnoul [24] apresenta um esquema de aplicação web para síntese de uma língua de sinais. O autor divide este modelo em duas classes principais, a do cliente e do servidor. A primeira é composta de um navegador *web* comum, com suporte a tecnologias como WebGL (API para HTML5 que permite a renderização de gráficos 3D em navegadores web) ou O3D (API de código aberto para criação de aplicações 3D em navegadores *web*) e *Java Script* (Linguagem de *Script* para navegadores web). O lado servidor recebe as requisições do lado cliente e faz o processamento de tradução e animação, seguido da conversão necessária para repassar os resultados ao navegador no lado cliente.

Kipp et al [21] realizaram um questionário com a finalidade de identificar quais aspectos, segundo os usuários, são mais relevantes em um avatar 3D para aplicações em línguas de sinais. Nos resultados, os autores obtiveram uma nota maior para expressões faciais e movimentos naturais, visto que avatares 3D em geral possuem movimentos mais robóticos. Fatores como postura, movimento corporal e representação de emoções, ainda, receberam um valor alto mostrando ser fatores importantes e relevantes para o usuário. Entre aspectos considerados menos importantes temos sincronia do movimento corporal com a boca, ou pequenas variações entre os dedos (problema reconhecido mesmo em intérpretes humanos).

Existem opções de *softwares* comerciais para interpretação gestual de línguas de sinais,

como o *Communicator Gesture Builder* [43], e o *Sign Smith Studio* [41], ambos da Vcom3D . A primeira ferramenta trabalha apenas com a ASL, a língua de sinais americana. O problema nestes casos é que, além de se tratar de uma ferramenta paga, o usuário ainda deve criar a animação através de botões e depois assisti-la. Por exemplo, o usuário seleciona a mão direita no avatar 3D, em seguida deve escolher entre uma lista de posições pré-definidas e repetir o processo para cada parte do corpo, inclusive os dedos, individualmente.

Além destes problemas, o tempo exigido para implementar cada sinal é grande e não há formalismo associado que permita a verificação da correção, além de impossibilitar o reaproveitamento dos sinais implementados. Ainda, um usuário não pode incluir novas posições na base de conhecimento do *software*, sendo que somente a empresa proprietária tem direitos para modificações e atualizações do *software*.

Antunes et al. [2] afirmam que um estudo das ferramentas computacionais existentes no contexto de representação de línguas de sinais revela que as mesmas não oferecem recursos para uso em ambientes reais por dependerem de tecnologias específicas como luvas ou sensores, que excluem o contexto de sinalização em ambiente natural pelos surdos. Ainda segundo os autores, muitos estudos apresentam base de dados incompleta, como um número pequeno de representações de sinais, ou poucos casos de teste.

Yi et al.[48] sugere ainda que um dos principais problemas nestes sistemas é que os usuários tem de saber a língua nativa do software. Os autores ainda reforçam a ideia de que estes softwares não tem a interface intuitiva ou simplificada o bastante, dificultando a interação de iniciantes. Um ponto importante no qual os autores focam é que estas ferramentas não podem ter seu conteúdo estendido, estando limitados a um conjunto de frases ou palavras pré-definidas. Em outras palavras, estes sistemas dependem de conteúdo controlado, não sendo *softwares* de animação automática.

Considerando os trabalhos relacionados neste capítulo, é possível perceber que os trabalhos relacionados a formalizar e representar linguagens de sinais vêm sendo desenvolvidos há décadas. A Tabela 2.1 apresenta um resumo dos principais trabalhos apresentados neste capítulo e sua relação com o trabalho proposto.

Modelo Formal Descritivo para Linguas de Sinais			
Autor	Ano	Referência	Descrição
Loomis et al.	1983	26	Primeira arquitetura visual da ASL
Stokoe	1978	39	Línguas de sinais tem base de conhecimento infinita e cada sinal é dividido em partes e compostos de três parâmetros: movimento, locação e config. de mãos
Kilma & Bellugi	1979	20	Sugerem novo parâmetro: orientação da palma da mão
Antunes et al.	2011	02	Classifica os parâmetros das línguas de sinais quanto ao: tipo, direção, frequência e maneira.
Quadros & Karnopp	2004	33	Sugerem um modelo formal descritivo para língua de sinais com posições independentes.
Antunes et al.	2011	02	Apresentam um modelo formal descritivo da Libras para uso computacional
Amaral	2008	01	Apresentam uma notação XMLpara representação da Libras.

Aplicações de um Avatar 3D para Linguas de Sinais			
Autor	Ano	Referência	Descrição
Chadwick et al.	1989	06	Propõe um modelo para Avatar 3D baseado em 3 camadas: esqueleto, músculos e pele
Kitaguchi & Saito	2005	22	Apresenta um método para modelagem tridimensional baseada em Low Poly através de um scanner
Kilma & Bellugi	1979	20	Apresentava conceitos para animação de um agente virtual para uso em ASL
Gibet	1994	12	Desenvolveu um braço 3D para representação de línguas de sinais
Geitz et al.	1996	11	Apresentou um modelo de configuração de dedos em 3D
Moya et al.	2011	28	Testou a simulação 3D na reabilitação de membros superiores
Segawa & Takeguchi	2002	35	Desenvolveu um sistema para ensino da língua de sinais japonesa baseado em uma luva com sensores para reconhecimento de movimentos
Hrúz et al.	2011	15	Sugere duas formas para uso de Avatar sinalizador: com movimentos em tempo real ou vídeos com animações pré-definidas
Yi et al.	2005	48	Apresentam um sistema parecido com um dicionário onde um avatar 3D interpreta o sinal escolhido.
Kaneko et al.	2010	19	Apresentam um sistema para representação de linguas de sinais baseado no software TVML.
Krnou	2011	24	Propões conceitos para representação de aplicação web para sintese de sinais baseada em Web.

Figura 2.1: Trabalhos Relacionados

CAPÍTULO 3

DESENVOLVIMENTO DO AVATAR 3D E INTERPRETAÇÃO DO MODELO FORMAL

Este capítulo objetiva apresentar métodos e técnicas utilizadas neste trabalho, que envolvem a modelagem do avatar 3D e sua estrutura de animação, com base nos parâmetros necessários para representação do modelo formal, além dos testes iniciais com a Engine Gráfica adotada para os processos do próximo capítulo. A ordem dos estudos segue como apresentado na Figura 3.1.

3.1 Modelo Descritivo da Língua de Sinais Brasileira

O primeiro passo quanto aos métodos de desenvolvimento está no estudo do modelo descritivo da língua de sinais, já que os processos restantes dependem das informações desta etapa. A base utilizada no trabalho é o modelo estruturado de representação computacional da Língua de Sinais Brasileira, apresentado em [2].

Após a análise do modelo, foi possível perceber que a estrutura de um sinal está dividida em quatro classes, que são suspensão, expressão não manual, movimento e sinal composto, conforme descrito em 2.1.

Cada classe é subdividida formando uma cadeia hierárquica dependente. A Figura 3.2 mostra um exemplo do modelo formal citado. Neste modelo, os parâmetros necessários para formar um sinal seguem uma cadeia hierárquica desde a posição dos dedos até o tronco ou ombros. Dependendo do sinal, pode envolver ainda expressões faciais.

Quanto à suspensão, o modelo divide-se em duas sub-classes: mão dominante e mão não dominante. Cada uma delas possuem as 61 configurações de mão da Libras, além dos valores possíveis para cada um dos cinco dedos [2].

Ainda segundo o modelo cada mão pode receber valores de locação, orientação e movimento local (movimentos sem deslocamento resultante no espaço).

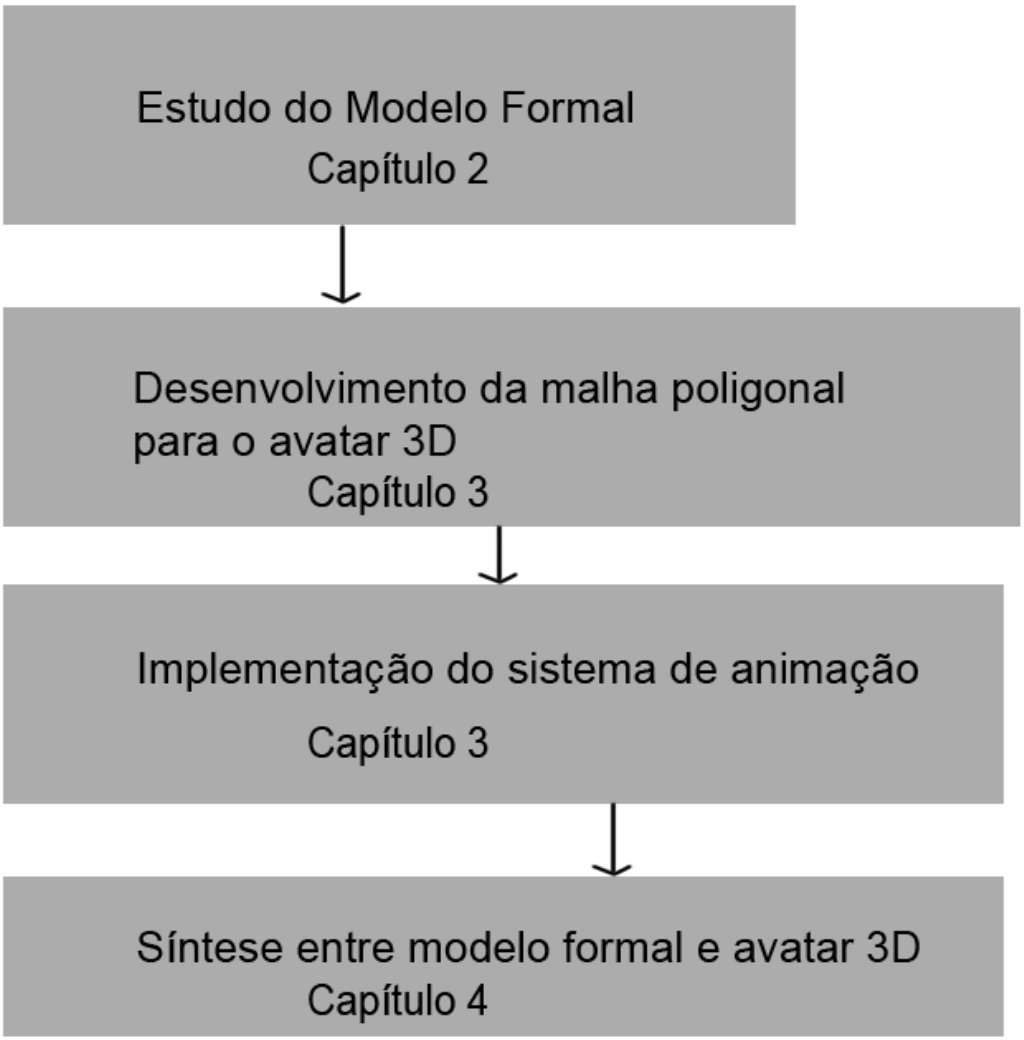


Figura 3.1: Etapas para implementação do trabalho - Figura do Autor

Quanto as expressões não manuais, são englobados valores referentes às expressões faciais, movimentos da cabeça e do tronco.

Com relação aos valores de movimento, o modelo adotado também divide entre a mão dominante e a não dominante, seguindo por seus próprios valores.

Por fim, a classe de sinais compostos cobre a repetição de determinado elemento na configuração do sinal.

Nenhuma classe descrita no modelo está completa, visto que a Libras sofre transformação constante, bem como, algumas classes não foram abordadas ou aprofundadas, como as expressões não manuais, que envolvem expressões faciais. Neste caso o modelo apresentado em [2] ainda não possui uma hierarquia definida para esta classe.

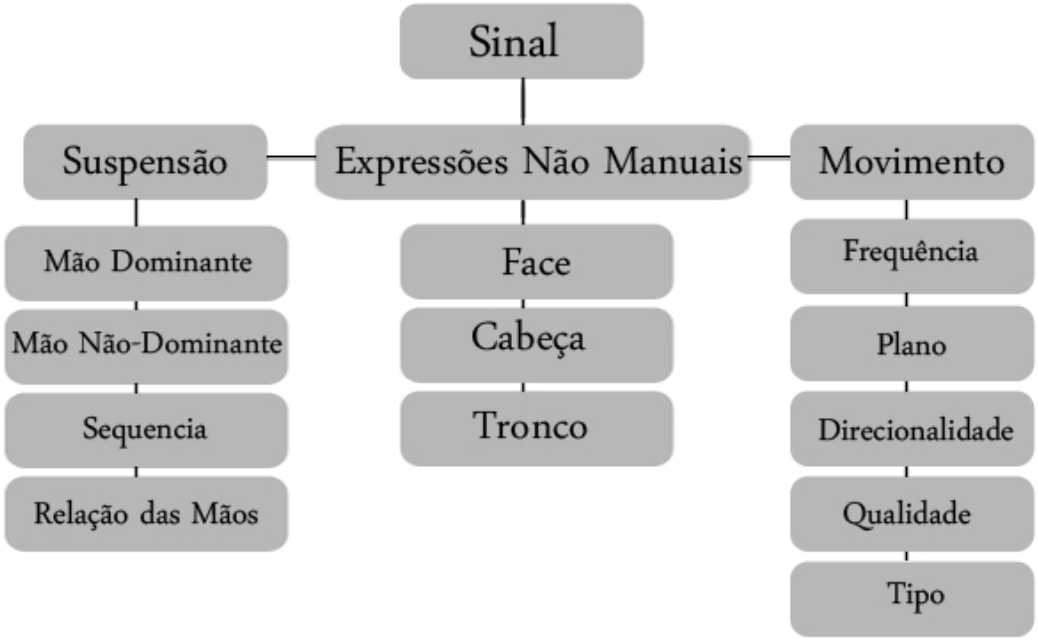


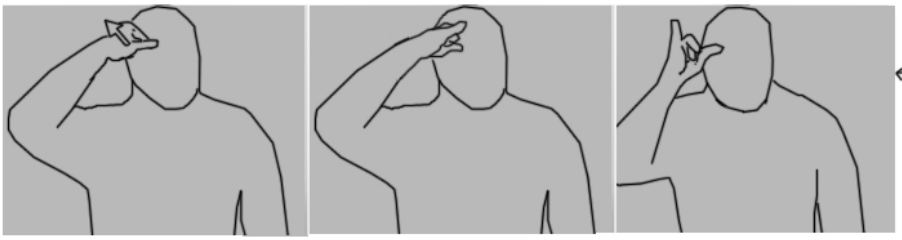
Figura 3.2: Modelo Formal apresentado por Antunes et al. - Figura do Autor

A intenção do uso do modelo formal é utlizar seus parâmetros de configuração para definir a animação do Avatar 3D. Esta idéia está ilustrada na Figura 3.3 onde temos um exemplo de VPC em XML representando o sinal boi. É possível perceber que esta entrada é dividida em blocos, no caso os blocos Sinal, Suspensão, Mão Dominante, Configuração de Mão, e outros como posição da cabeça ou palma da mão sendo que cada bloco está inserido dentro de um bloco pai e todos recebem determinados valores.

A partir deste XML são extraídos os VPCs e então o Avatar 3D é animado. Ainda, na Figura 3.4 pode ser observado outro exemplo onde temos o elemento contato-conf e seus respectivos valores, sendo que cada um deles irá gerar a animação correspondente no Avatar 3D.

Então, o modelo formal adotado possui os elementos para descrever as operações elementares de movimentos para gerar sinais da Libras. Maiores detalhes sobre o modelo adotado serão apresentados no próximo capítulo.

As próximas seções apresentam conceitos e técnicas usadas neste trabalho para a construção de um avatar 3D, além de estudos preliminares de interpretação do modelo formal em XML.




```
<sinal nome="boi">
  <suspensao>
    <mao-dominante>
      <configuracao-mao>4</configuracao-mao>
      <locacao ladocorpo="mão dominante">
        <cabeca-loc contato="polegar (ponta)">testa (lado)</cabeca-loc>
      </locacao>
      <orientacao mao="horizontal">
        <palma>para baixo</palma>
      </orientacao>
      <movimento-local sentido="horário">
        <pulso-mov>
          <torcedura>girar para direita</torcedura>
        </pulso-mov>
      </movimento-local>
    </mao-dominante>
  </suspensao>
</sinal>
```

Figura 3.3: Exemplo do uso dos VPC's - Figura do Autor

3.2 Modelagem e animação de um Avatar 3D

Existem muitas ferramentas completas de criação 3D no mercado, alguns exemplos como o *software* 3D *Studio Max* da *Autodesk*, o *Blender* [34], ou o *Zbrush* da *Pixologic* [50] e são utilizados frequentemente nas mais diversas áreas como cinema, jogos eletrônicos ou arquitetura. A maioria, no entanto, são comerciais, e por este motivo foi escolhido para o trabalho o Blender por ter sua distribuição livre, além de ter um grande suporte por meio de fóruns de usuários como o da comunidade *Blender*.

A construção do Avatar 3D segue as seguintes etapas: desenho da malha do avatar, escolha da técnica de modelagem do avatar, definição dos pontos de articulação, construção do *rigging*, que é a estrutura utilizada no processo de animação e sua ligação com a malha do objeto e as reações da malha ao movimento da estrutura controladora ligada a ela. Todas as etapas são detalhadas a seguir.

Uma malha 3D é a representação de um modelo de superfície poligonal, conceito muito usado na representação de objetos geométricos [29]. Para a criação de uma malha 3D são utilizadas técnicas de modelagem, diferentes para cada fim, como por exemplo modelagens

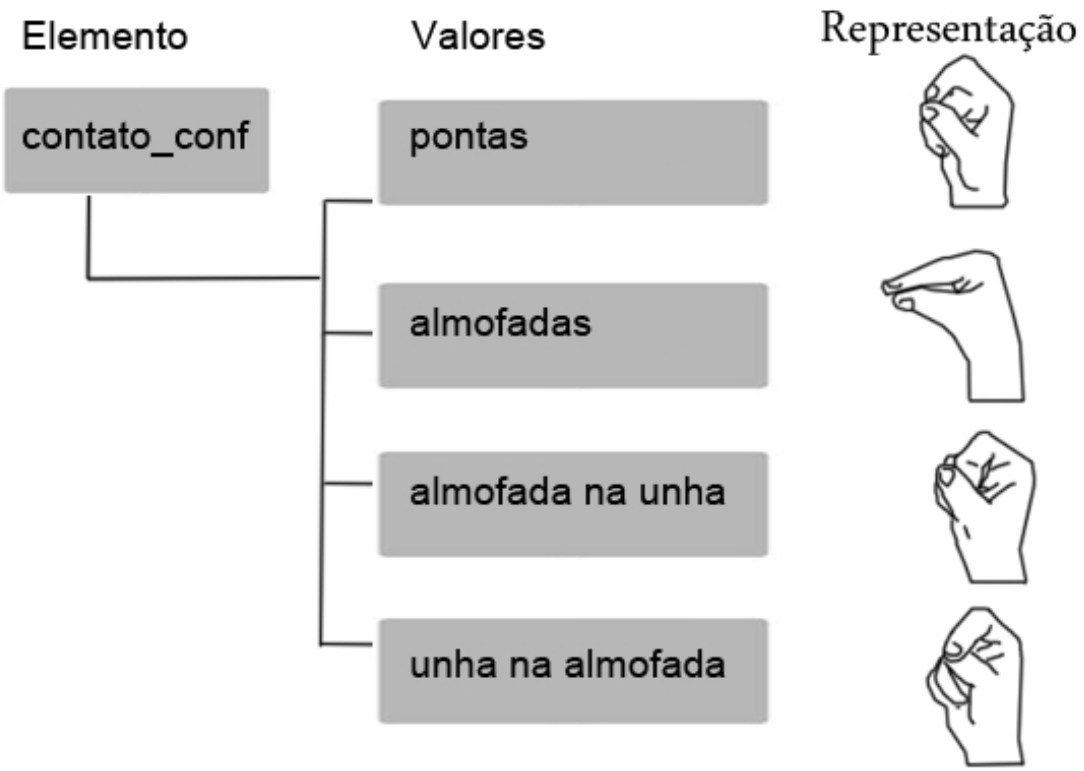


Figura 3.4: Exemplo de valores representados no modelo formal e sua ação no Avatar 3D. - Figura do Autor

mecânicas (objetos como automóveis ou peças industriais) ou orgânicas (objetos como animais, plantas etc.). Para esta tarefa existem algumas técnicas utilizadas onde destacam-se a modelagem *box-modeling* e a *poly-by-poly* [34].

Na *box-modeling* o desenvolvedor inicia seu trabalho a partir de uma primitiva, que pode ser uma *box* (primitiva tradicional em *softwares* 3D representada por um cubo simples com seis faces), como o nome da técnica sugere. Esta primitiva tem suas faces poligonais subdivididas de acordo com o contorno desejado e suas arestas arrastadas fazendo com que a estrutura tome a forma desejada.

No processo de modelagem utilizando a técnica de *poly-by-poly*, o modelo inicia com um polígono solto, sendo que uma ou mais arestas deste polígono são duplicadas e re-posicionadas formando a malha através da criação de faces novas.

As duas técnicas de modelagem citadas podem utilizar uma arte conceitual como referência para a modelagem (arte conceitual é uma imagem comum em 2D chamada

também de *blueprint*), e pode ser iniciada de qualquer ponto do objeto, com uma primitiva *plane* (polígono de uma face, para o caso na *poly-by-poly*) ou uma primitiva *box* (na *box-modeling*).

A Figura 3.5 mostra como funciona a estrutura básica dos dois métodos de modelagem citados.

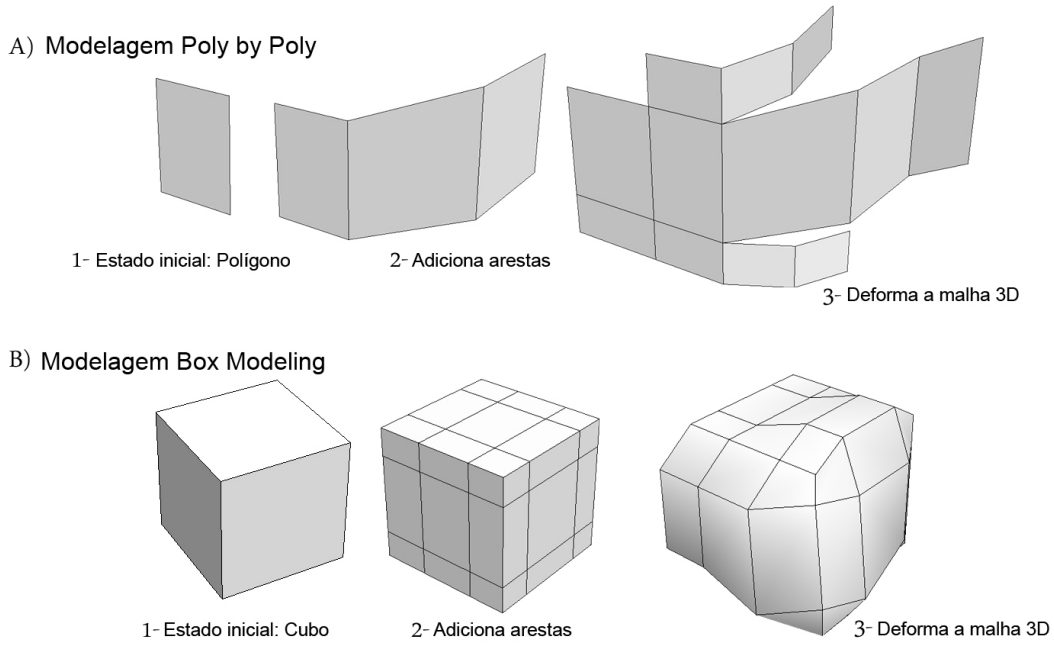


Figura 3.5: Técnicas de modelagem 3D - Figura do Autor

Para este trabalho foi utilizada a técnica de modelagem *poly-by-poly*, pois oferece maior liberdade já que não depende do formato atual do objeto para iniciar a modelagem de outras partes. Quando utilizados outros métodos de modelagem neste processo, contudo, os resultados tendem a ser equivalentes quanto à eficiência e à qualidade ja que o diferencial nesta etapa é o planejamento da malha. A técnica de *box-modeling* utiliza em todo o seu processo um corpo geométrico fechado. Existem desenvolvedores que em algum momento do processo removem uma face e fazem a extrusão de novas faces pelas arestas do buraco, o que transforma o processo em uma mistura das duas técnicas citadas. Em geral tenta-se conseguir um modelo com o menor número possível de polígonos, o que facilita os processos posteriores referentes à animação em si, além de tornar a malha reduzida, o que representa menor custo computacional.

Outra técnica utilizada na etapa de modelagem é chamada de *Edge loop* [31], técnica que sugere que as arestas do modelo devem simular as fibras dos músculos, ajudando a criar mais realismo no movimento quando o modelo é animado. Existem dois tipos de tensão (tensão é a contração em uma parte da malha), a tensão poligonal (descrita na construção da malha) e a tensão de movimento onde o *Edge loop* torna a distorção dos polígonos algo mais natural e real. A Figura 3.6 exemplifica o conceito das *Edge loops*.

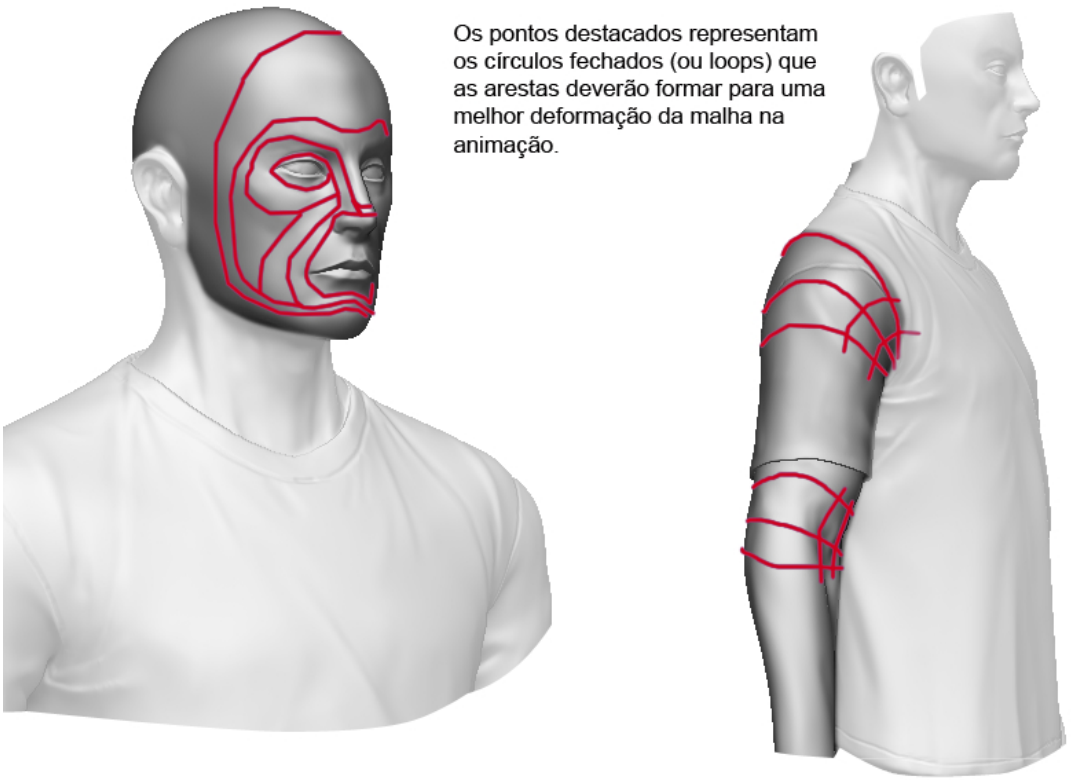


Figura 3.6: Edge Loops - Figura do Autor

A tensão poligonal é a deformação que a malha pode sofrer no caso de uma duplicação poligonal. É um recurso utilizado para deixar o modelo menos facetado e visualmente melhor, mais suave. Por exemplo, caso algum polígono esteja com mais de quatro arestas é possível que, no momento da animação, ele se apresente quebrado, ja que todo polígono é formado por dois triangulos, ou seja, possui quatro arestas. No caso de modelos mais complexos como um avatar 3D, é comum ocorrerem faces que não são formadas por dois triângulos, como normalmente acontece, o que faz com que a tensão poligonal fique

errada. Já na tensão de movimento o que pode acontecer é quebra de polígono durante a animação. Nos dois casos a utilização de um planejamento na construção da malha e o uso de *edge loops* deve tornar a malha mais eficiente e evitar estes problemas [29].

Para o processo de modelagem do Avatar 3D utilizado neste trabalho, foi desenvolvida uma arte 2D para referência na modelagem. Esta arte representa uma estrutura humana básica, com foco nas mãos, devido às muitas configurações manuais possíveis na Libras, bem como pelo nível elevado de detalhamento de movimentos nesta área.

Seguindo o princípio do modelo formal [2], a área do rosto não foi modelada, já que não é foco deste trabalho. A Figura 3.7 representa a arte conceitual utilizada para o desenvolvimento do modelo em 3D. Esta arte foi desenvolvida utilizando técnicas de desenho à mão, depois digitalizada e utilizada como referência para a modelagem. Ainda nesta imagem, é apresentado o modelo do avatar 3D inicial gerado a partir da modelagem usando a arte como referência. Este modelo possui 2.276 polígonos e pode ser considerado um modelo com poucos polígonos. O objetivo foi criar uma malha limpa, com poucos detalhes, para facilitar o processo de testes e animações.

A malha 3D é um objeto estático, para movimentar qualquer uma de suas partes são necessários controladores e um sistema de animação. Este processo se chama *rigging* e antecede os procedimentos de animação, objetivando criar uma estrutura controladora da malha [29]. O *Blender* possui um sistema de *rigging* completo, baseado em uma estrutura de controladores chamada *Armature*, constituída de *bones* (objetos controladores utilizados pela maior parte de *softwares* para animação 3D, onde cada objeto, ou *bone*, na estrutura é ligado a determinada área da malha).

Com base no modelo foi realizado um estudo para identificar em quais pontos de articulação o agente virtual sinalizador será animado. Foram definidos os pontos necessários para a representação de qualquer sinal da Libras, que são: pescoço, tronco, ombro, cotovelo, pulso e duas articulações para cada dedo. Com exceção das expressões faciais, todos os sinais da Libras podem ser representados com estas articulações. A Figura 3.8 apresenta as articulações definidas com base no modelo formal, bem como um exemplo de parâmetros apresentados no modelo formal para representação da Libras.

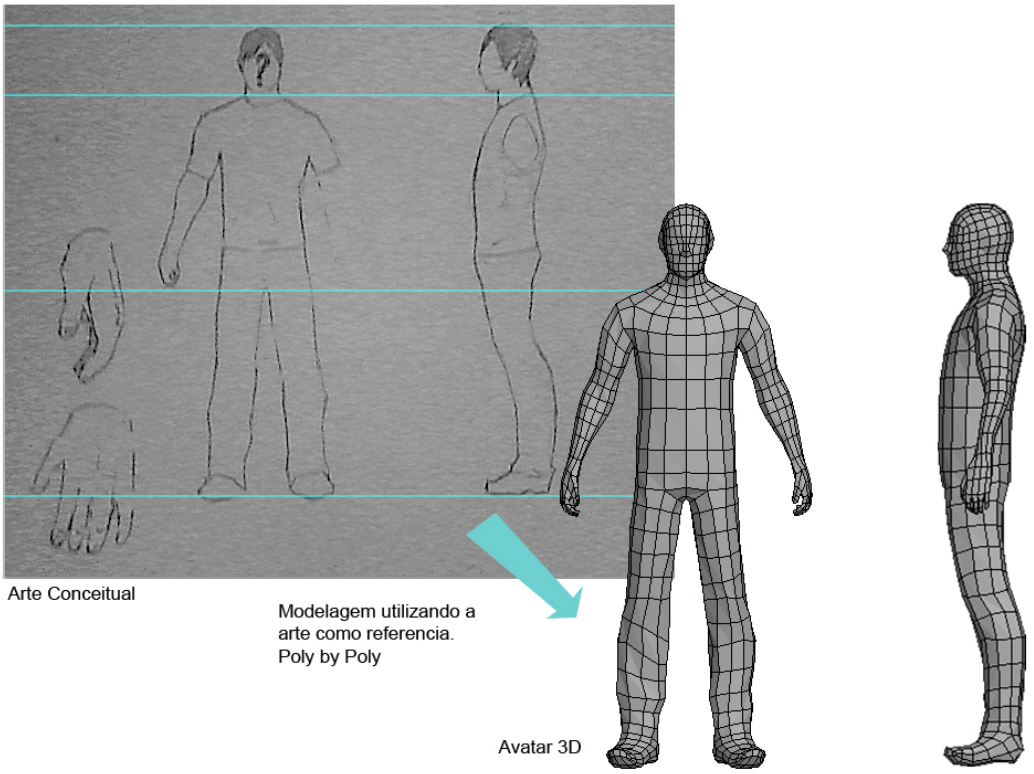


Figura 3.7: Arte Conceitual e Avatar 3D modelado - Figura do Autor

.

Em seguida foi iniciado o processo de construção do *rigging* do avatar, utilizando o recurso de simetria, que copia os objetos selecionados, para que toda a sequência do ombro, braço e mão sejam criadas igualmente, além de *bones* para os demais pontos de articulação. Foram adicionados *bones* para os braços, para os antebraços e um para cada mão. Do *bone* do tronco foi feita uma extrusão para o pescoço e em seguida um *bone* para a cabeça.

A estrutura das mãos é a parte mais complexa devido à quantidade de pontos de articulação e a vasta gama de configurações de mão que são utilizadas nas línguas de sinais. A partir do *bone* da mão foram adicionados dois bones para o dedo anelar e dois para o dedo médio. Para o indicador e o mínimo além dos dois *bones* para cada um foi construída uma ligação entre eles e o *bone* da mão. O polegar também recebeu dois *bones*, bem como uma ligação com o *bone* da mão. A estrutura de *rigging* da mão pode ser observada na 3.9.

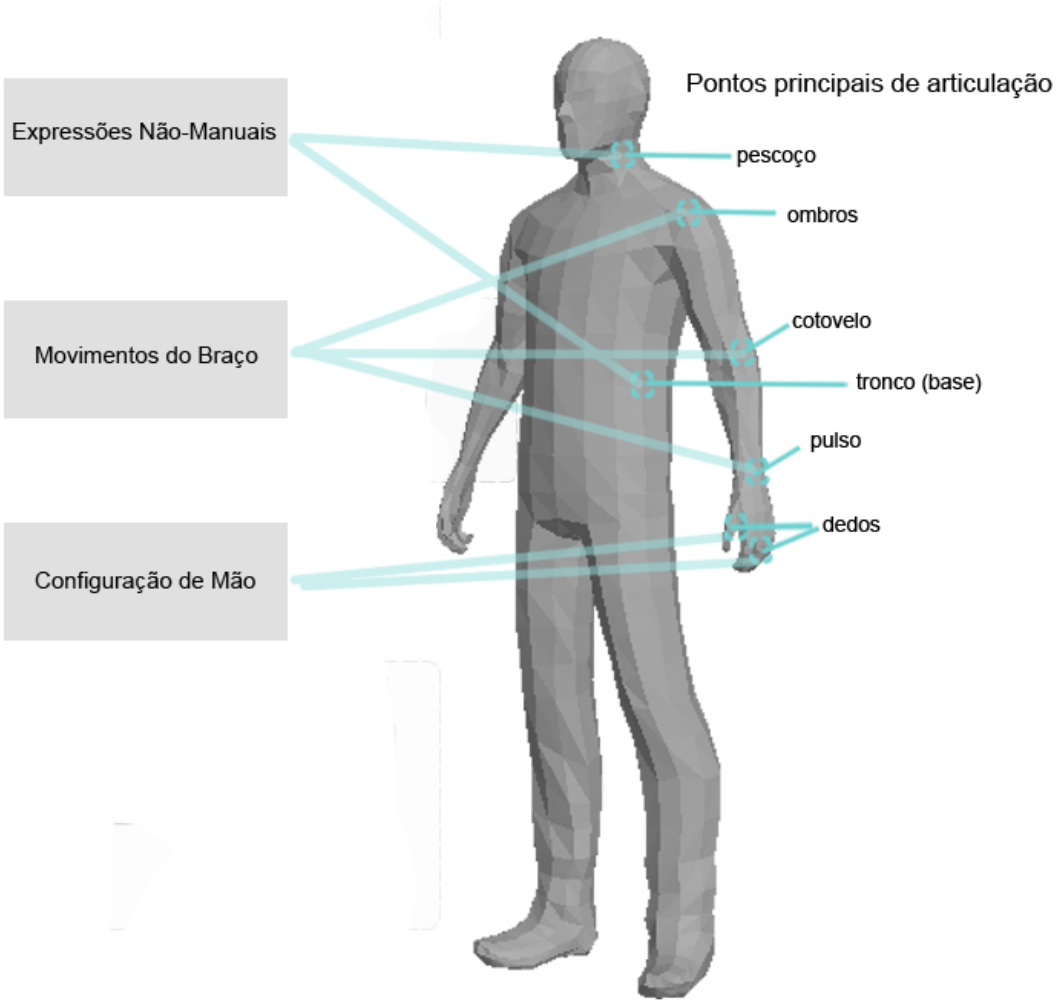


Figura 3.8: Articulações do Avatar 3D - Figura do Autor

A estrutura exibida na Figura 3.9, desenvolvida para este trabalho, possibilita reproduzir os movimentos dos sinais existentes da Libras, com exeção das expressões não manuais. Ainda, a estrutura pode ser usada para reproduzir movimentos de sinais que ainda vão ser incluidos na Libras, já que em geral todos utilizam as mesmas configurações de mãos e posições dos braços e ombros. Esta estrutura de controle para animação segue o padrão de *rigging* para avatares humanos, não sendo uma estrutura específica para língua de sinais, mas que comporta todos os parâmetros necessários para as posteriores animações.

O processo seguinte foi integrar a malha do avatar ao sistema de *rigging*. Para isto foi utilizada a configuração por peso dos vértices em relação aos *bones*, que consiste em selecionar a cadeia de vértices que responderá ao movimento de determinado *bone*. Se um vértice está ligado a um *bone* ou mais, a deformação da malha pode sofrer deformação

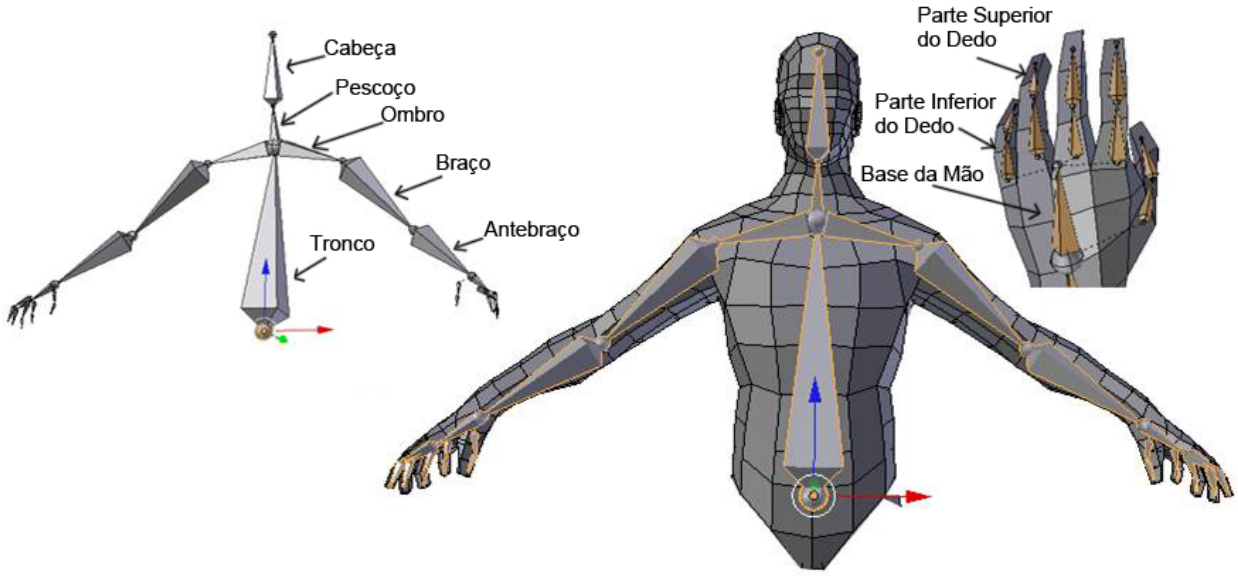


Figura 3.9: Sistema de Rig para animação - Figura do Autor

errada, ou ainda se algum vértice não tiver seu peso ligado a um bone, o mesmo não será animado ficando estático no momento da animação. Os pesos são representados por cores, quanto mais quente (tons de vermelho e laranja) maior é a interação do vértice com o *bone*, e cores frias (tons de azul) representa que o *bone* não afetará o vértice.

Segundo [2] as soluções encontradas para animação de agentes virtuais possuem a limitação de que muitas informações importantes para a reprodução não aparecem nas notações existentes. Segundo estes autores, algumas destas informações podem ser deduzidas por um ser humano, mas um avatar 3D depende de todas as informações necessárias para se obter um sinal com precisão.

Embora esta questão da qualidade de pequenos detalhes seja um ponto importante, essa afirmação traz à tona o fato de que algumas configurações como a posição da ponta de um dedo (algo relevante em teoria), na prática podem ser subentendidos pelo contexto. Até mesmo em um ambiente real, com intérprete humano na maioria das vezes o surdo identifica o sinal pelo conjunto de movimentos, impedindo que um detalhe pequeno interfira na identificação do sinal completo.

3.3 Mapeamento e Texturização do Avatar 3D

Após construir a malha 3D, é possível incluir uma textura e um material para o modelo. A malha poligonal não possui nenhum tipo de material definido, consiste apenas dos polígonos organizados a fim de simular uma estrutura desejada. A maioria de motores gráficos insere uma textura padrão no objeto que consiste, em geral, em uma cor chapada envolvendo todo o objeto. Então, para que a malha tenha uma visualização de cores, texturas, efeitos (rugosidade, aspecto metálico, etc.) é necessário incluir um material e uma textura no modelo.

O material representa a forma como o objeto irá reagir à luz, em função de vários fatores canais como brilho, opacidade, relevo, refração. O controle de cada canal pode ser feito por cálculos matemáticos como um mapa de ruídos para gerar uma superfície irregular no objeto (no caso de refração da luz, por exemplo) ou através de mapas de textura.

Ainda, o material é responsável por reproduzir no processo de renderização o aspecto da superfície do objeto. Se é orgânico, em geral apresenta uma aparência irregular utilizando canais como o tipo de material, como material mais rugoso, ou liso. Se é mecânico, utiliza-se canais próprios para se chegar a uma representação fiel de superfícies metálicas.

As ferramentas de desenvolvimento em 3D oferecem opções de criação de materiais, que podem unir diversas características, a fim de chegar na aparência de cada superfície proposta. Cada elemento, como cor ou rugosidade, é controlado por canais individuais. Nestes, por sua vez, são utilizados mapas de textura.

Os mapas de texturas em geral são imagens específicas que utilizam coordenadas UVW, utilizadas em um plano 2D para mapeamento de malhas 3D, para representar um determinado aspecto ou propriedade. São utilizadas imagens comuns (arquivos com extensão jpeg, png, tif, bmp entre varias outras) como controladores, respeitando as características de cada canal. Por exemplo, no canal de opacidade, áreas mais escuras recebem transparência na malha, e áreas mais claras são mais visíveis. Cada canal recebe uma imagem própria.

Como principais canais, em geral, temos:

i) *Diffuse*, ou canal de cor, onde é inserido um mapa representando apenas as cores da superfície, sem efeitos de iluminação.

ii) *Specular* que representa a reação da cor da superfície em contato com a luz.

iii) *Bump* que controla profundidade e relevos na malha (como rugosidade).

iv) *Opacity*, que controla os níveis de opacidade, os níveis de transparencia em cada ponto da malha 3D.

Para os testes deste trabalho, é utilizado o canal *difuse*, para efeito visual, excluindo os outros mapas para reduzir processamento, já que nesta etapa não são necessárias representações de um modelo final renderizado.

A primeira etapa, antes mesmo da criação da textura e do canal de material, é recortar a malha em um plano 2D. Este processo é chamado de mapeamento UVW [29]. O recorte da malha é feito manualmente, para uma melhor distribuição da estrutura no ambiente, sendo que este plano 2D tem um tamanho limitado.

Blender, *3DStudio Max*, e outras ferramentas trabalham com propriedades parecidas de espaçamento e edição no plano da textura. Para isso e inserido um modificador chamado *Unwrap Map* no objeto, que pode ser um único modificador ou um modificador *Unwrap* para cada parte isolada do objeto. Objetos podem ter partes isoladas, representadas por grupos de polígonos diferenciados ou assimilados por identificadores. Neste trabalho foi utilizado apenas um identificador para a malha inteira e então aplicado um único modificador *Unwrap* na estrutura.

Foi realizado manualmente o recorte da malha em diferentes partes e os polígonos foram abertos no plano 2D. A estrutura foi dividida nas seguintes partes: mão direita e mão esquerda, pés direito e esquerdo, cabeça, braço direito e esquerdo, parte de trás do corpo e parte da frente do corpo (estas últimas incluindo o tronco e pernas). Ainda, foi deixado um espaço na parte superior do plano 2D, caso seja necessária a inclusão de algum outro elemento poligonal, tal como planos para representação do cabelo, técnica muito utilizada em objetos humanos *Low-Poly*.

Com a malha aberta, foi possível criar a textura para o canal *diffuse*, onde foram

incluídos elementos básicos apenas para facilitar a visualização das partes do Avatar. Esta imagem foi inserida em um arquivo *Bitmap* com as mesmas medidas e resolução do plano 2D criado no modificador *Unwrap* de coordenadas UVW. A malha distribuída no plano 2D resultante, bem como a textura utilizada no Avatar está representada na Figura 3.10.

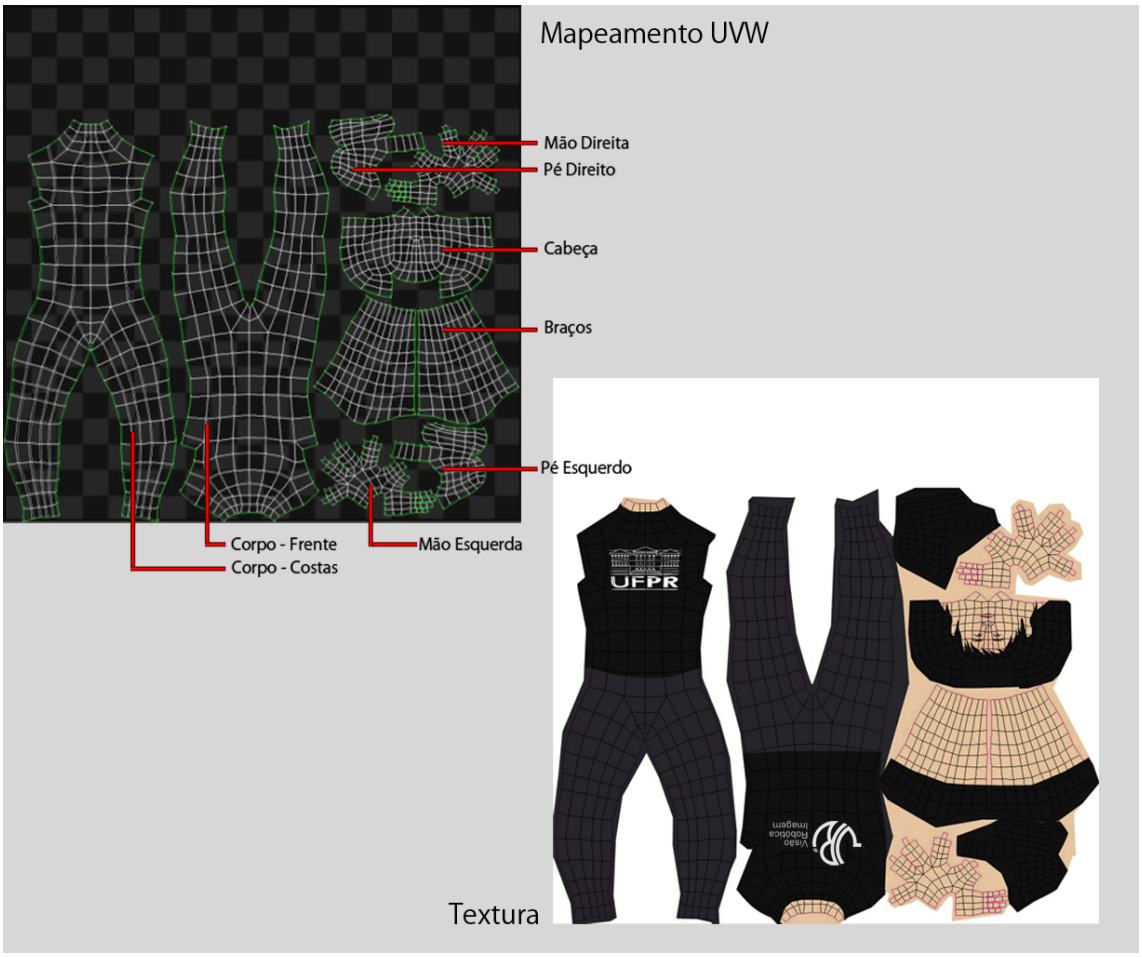


Figura 3.10: Mapeamento UVW e textura Diffuse - Figura do Autor

A seguir a textura foi aplicada ao canal *difuse* do material padrão do *Blender* e inserida no Avatar. É importante ressaltar a importância do recorte e abertura corretos da malha, já que um pequeno conjunto de polígonos achatados ou mal diagramados neste momento resulta em deformações na representação final do modelo renderizado. Este processo todo inclui cuidados na resolução e criação de cada textura, mas especialmente na distribuição inteligente da malha no plano 2D.

Para os testes no motor gráfico não serão incluídos neste momento efeitos de iluminação

no objeto e materiais com vários canais, sendo que o canal *difuse* já cumpre o papel de facilitar a representação do modelo animado.

O processo seguinte foi incluir a textura no modelo através do código C++, para que seja carregada no motor gráfico. Para isso foi observado que a textura, por convenção, deve possuir o mesmo nome do objeto 3D, além de ser incluída na mesma pasta do arquivo com a malha 3D a imagem referente a textura. Para inserção da malha no objeto foram usados métodos da *Irrlicht* próprios para este fim. Como não existe objetos de luz, nem canais mais complexos no material aplicado, o custo computacional permanece o mesmo, já que quando não é incluída uma textura ou material no modelo, a *Irrlicht* insere um material padrão, representado por uma cor chapada envolvendo toda a malha, ao passo que o modelo, embora texturizado pelo canal *diffuse*, permanece com as cores da textura chapada, sem sombras ou efeitos diversos.

O resultado do material e textura difuse aplicados ao Avatar 3D podem ser observados na Figura 3.11, onde temos uma imagem renderizada do modelo texturizado.

O avatar 3D desenvolvido tem estrutura, articulações, malha, textura e sistema de animação, sendo então apto a ser controlado para síntese de sinais da Libras.

3.4 Interpretação do modelo formal descrito em XML e transcrição para o agente virtual

O modelo descritivo tem sua aplicação para modelos XML. Para se gerar movimentos automaticamente através de entradas baseadas no modelo formal, é necessário identificar e interpretar os parâmetros do XML e conseguir transformar esta entrada em informações reconhecíveis pela aplicação 3D, de maneira automática e de fácil acesso ao repositório de dados, para que a base do sistema possa ser ampliada ao longo de seu uso.

Para estas próximas etapas algumas ferramentas e técnicas devem ser estudadas para verificação se tais aplicações suportam os objetivos do trabalho. A princípio, o *Blender* possui boa integração com a linguagem de programação *Python*, oferecendo muitos recursos e ferramentas para esta linguagem em sua interface.

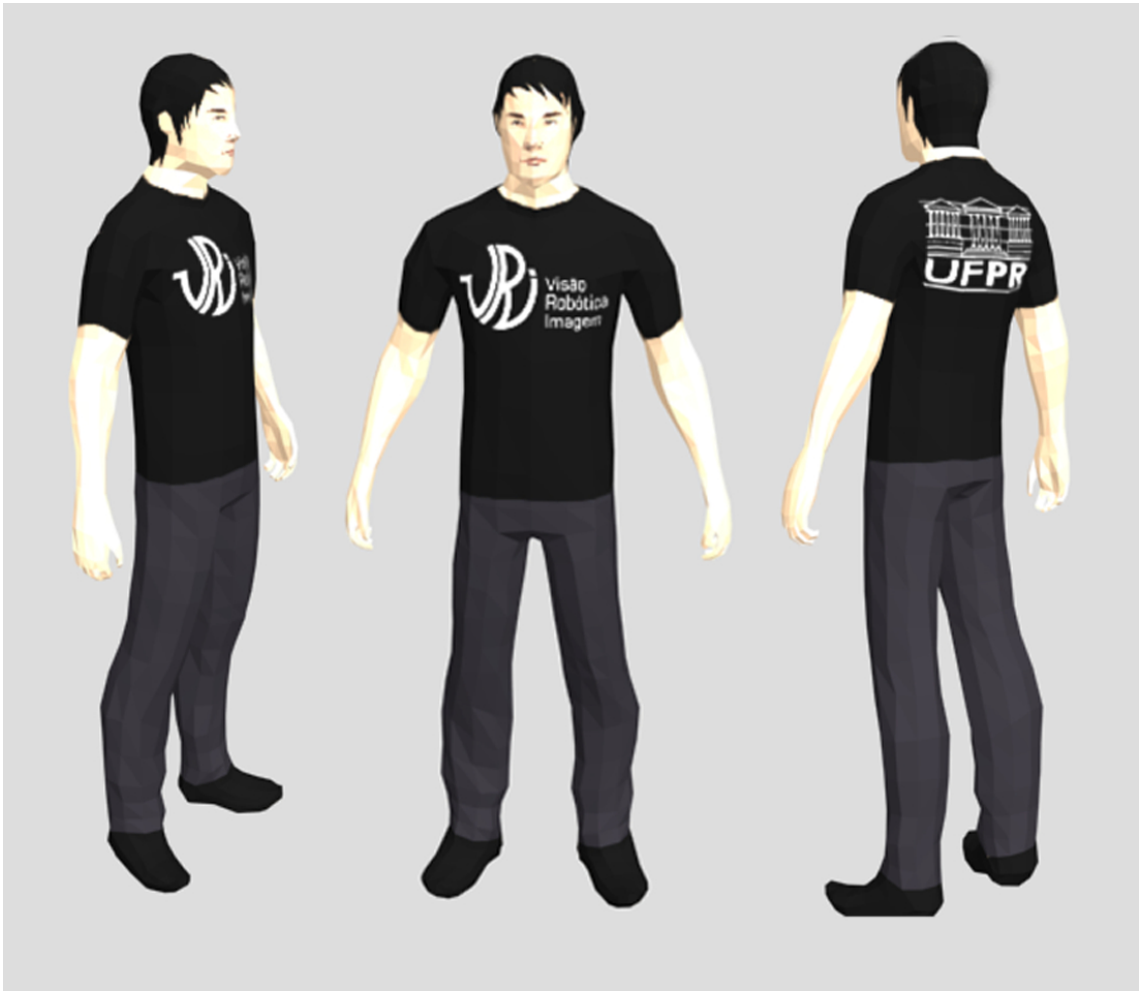


Figura 3.11: Textura aplicada. - Figura do Autor

Um exemplo disto é o console *Python* integrado a interface do *Blender* [3], onde é possivel criar ou executar *scripts python* completos. Afirmar se estes recursos podem ser utilizados para os objetivos deste trabalho, no entanto, requer um estudo mais profundo.

Em uma pesquisa exploratória foram identificadas outras ferramentas que podem ter seu uso empregado neste trabalho, como a BGE (*Blender Game Engine*) [3] que oferece recursos para o desenvolvimento de aplicações interativas, associado ao próprio *Blender* ou a *Irrlicht Engine* [47], ferramenta com objetivo similar a BGE, que utiliza a linguagem C++. Estas ferramentas são conhecidas como Game Engines, ferramentas muito utilizadas na criação de jogos, e que também podem ser empregadas no desenvolvimento de sistemas interativos independentes.

No mercado existem diversas *Engines* disponíveis, algumas de uso livre, mas em sua maioria *softwares* comerciais. Alguns exemplos conhecidos são a *Ogre 3D* [17] que utiliza

C++, Panda 3D [36], *Infinite Engine* [3], *Cry Engine* [30] (já na terceira versão), *Unreal Engine* [13], *Scimitar* [49], utilizada pela empresa de jogos eletrônicos Ubisoft, entre várias outras.

A principal diferença entre as *Engines* está no seu foco. Algumas são próprias para desenvolvimento de aplicações 2D, outras para 3D, algumas para o sistema operacional do *Android* como é o caso da *Ignifuga Game Engine* [27], outras com suporte a Java, que possibilita desenvolver aplicações para serem executadas em navegadores, como a *Jmonkey* [45].

Yijun et al [25] apresentam um gráfico demonstrativo representando a arquitetura de uma Engine gráfica onde temos na camada superior a aplicação em si, partindo para o núcleo da *Game Engine*, a seguir as camadas são divididas no sistema de navegação e nas bibliotecas utilizadas, por fim temos o sistema de animação e física.

Estes autores ainda classificam os principais módulos de função das *Engines* como módulo de controle do uso da memória, módulo matemático, módulo de controle de entrada módulo de renderização, e por fim o módulo para controle da cena.

Outras ferramentas de simulação de ambientes virtuais também poderiam ser utilizadas para o desenvolvimento de um software para auxílio de pessoas surdas, como *Open Dynamics Engine* (ODE) [10] e a *Newton Physics Engine* [9] para simulações físicas que utiliza a linguagem VRML para criar o ambiente 3D.

Zhenfeng e Zhao [14] utilizaram a tecnologia da ODE e a *Engine Ogre 3D* (*Open Gestures Recognition Engine*) para representar um ambiente virtual para simulação de robôs móveis. Segundo os autores, a *engine Ogre* exporta suas cenas para muitas ferramentas de modelagem, como o *Blender* e o *3D Studio Max*, apresentando as vantagens de ser de código aberto e ter suporte a orientação a objetos em sua programação.

Em [46], os autores estudaram a *Irrlicht* para criação de um ambiente virtual. Neste trabalho foi desenvolvido um cenário experimental, onde testavam o controle de de um aparato. O mecanismo poderia girar segundo comando do usuário, tendo seu movimento animado através do ambiente virtual. Para exportação das malhas os autores utilizaram o *plugin Panda Engine* [38] do *software* de edição *3D Studio Max*, formato suportado até

hoje pela *Engine Irrlicht*. Com os experimentos os autores concluiram que a *Engine* tem bom desempenho na renderização em tempo real do ambiente virtual, alem de ressaltar seus resultados de importação de malhas e outros objetos.

Segundo Kot et al. [23] *Game Engines* e suas implementações têm sido utilizadas para vários fins, para os quais não foram originalmente concebidos para atender, como simulações militares, planejamento de ambientes, e projetos de visualização arquitetônicas, por oferecerem ambientes interativos. Neste sentido, as *Game Engines* também oferecem recursos adequados para simulação ou representação de linguas de sinais.

Além de suportar modelos e cenários em 3D, as *engines* gráficas devem proporcionar controles de visão (normalmente por uma câmera pré-definida, interativa) além de gerar os aspectos físicos necessários no ambiente, como efeitos de colisão [5], importantes, por exemplo, para que determinada área da malha (como o braço) não entre em outra parte da estrutura (como o tronco) durante a animação.

Para o caso de representação de línguas de sinais não há grande necessidade no uso dos recursos de física e colisão de objetos oferecidos pelas *engines* gráficas, uma vez que o avatar não terá de interagir com objetos ou percorrer ambientes. Então, um *rigging* apropriado já simula satisfatoriamente os movimentos humanos, sem a necessidade de recursos adicionais.

As *engines* gráficas também podem ter uso didático como em [18], onde o autor propôs utilizar a *Engine Ogre 3D* como ferramenta para ensinar a linguagem de programação C++. Em muitos casos o propósito da *Engine* serve como motivador para que se aprenda a programar em determinada linguagem. Neste sentido, a BGE poderia ser utilizada para se ensinar *Python*, *Jmonkey* para JAVA, entre vários outros exemplos.

Na intenção de definir uma *Engine* para utilização neste trabalho foi desenvolvido um teste simples, objetivando analisar o funcionamento geral dos sistemas. O teste consistiu em exportar um modelo simples simulando uma articulação em movimento, com animação de 20 *frames* onde um braço fazia uma torção de 90 graus. O objeto em si consistia de uma primitiva *box* com extrusão na parte superior, como mostra a Figura 3.12. Em seguida o objeto animado deveria ser importado na Engine, sem perder a animação ou

qualquer de suas características como material e *rig*. O parâmetro de avaliação entre as *Engines* foi o nível de dificuldade de se obter o resultado esperado, seja na configuração da *engine* com as IDE's ou a implementação necessária para chamada e rederização da cena.

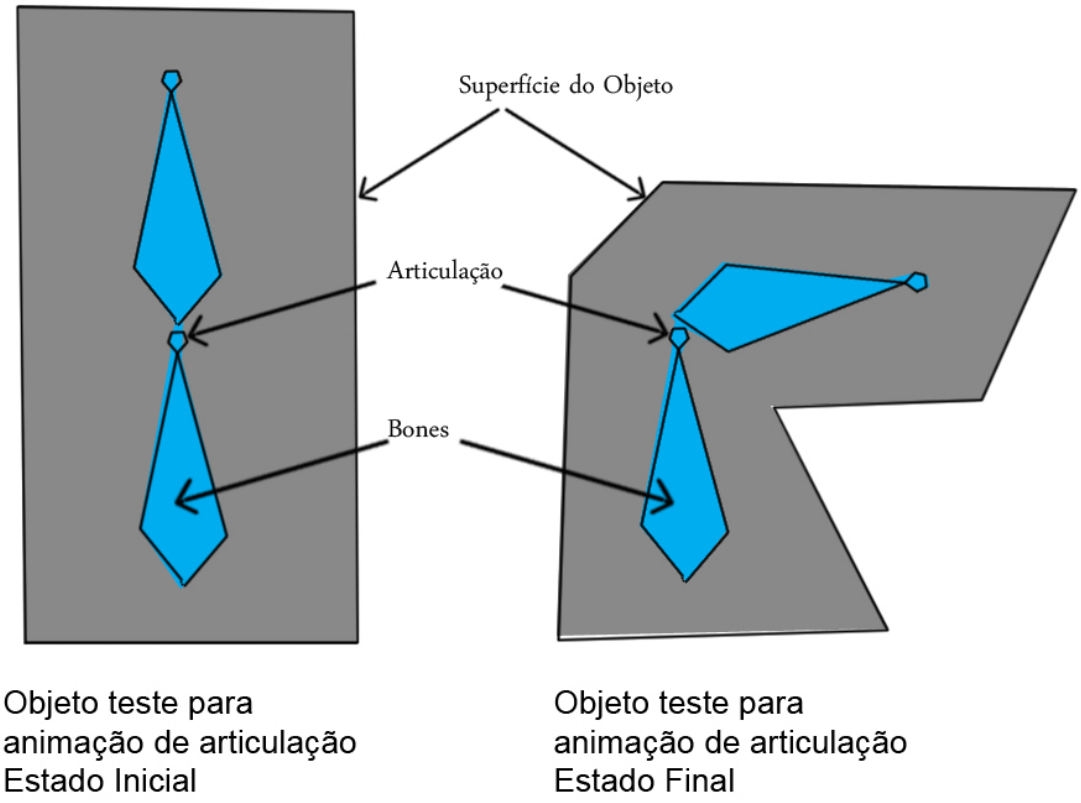


Figura 3.12: Objeto para teste nas Engines - Figura do Autor

Foram escolhidas quatro *Engines* para avaliação pelos respectivos motivos:

i) *Blender Game Engine*, que ja está integrada ao *software* de edição *Blender* utilizado neste trabalho, e portanto deveria ser testada.

ii) *Irrlicht*, por ser uma alternativa que utiliza a linguagem C++.

iii) *Ogre 3D*, pois se mostraram presentes em muitos dos artigos e trabalhos citados anteriormente.

iv) *Jmonkey*, por utilizar linguagem diferente das anteriores (Java, sendo que as outras utilizam *Python* e C++) e permitir exportação para *web*.

A princípio foi realizado o teste na *Irrlicht*, esta *Engine* utiliza o C++ em sua programação, portanto foi utilizado o ambiente de desenvolvimento integrado (IDE) *Code*

Blocks. Este programa já oferece uma opção para iniciar um projeto da *Irrlicht*, então foram instaladas as dependências e bibliotecas necessárias para utilizar a *Engine*, além de instalar a mesma.

Um problema encontrado é que na página na internet da *Irrlicht* não é possivel encontrar um arquivo necessário para exportar malhas e cenas do *Blender* no formato b3d, extensão utilizada na *Engine Irrlicht*. Este arquivo foi encontrado em uma fonte externa.

Foi então implementado um código para chamada do braço de teste animado na *Irrlicht*, com configuração simples de velocidade e *frames* utilizados. A Figura 3.13 é uma impressão de tela com o ambiente de desenvolvimento e a saída onde o objeto é animado a partir da *Engine*.

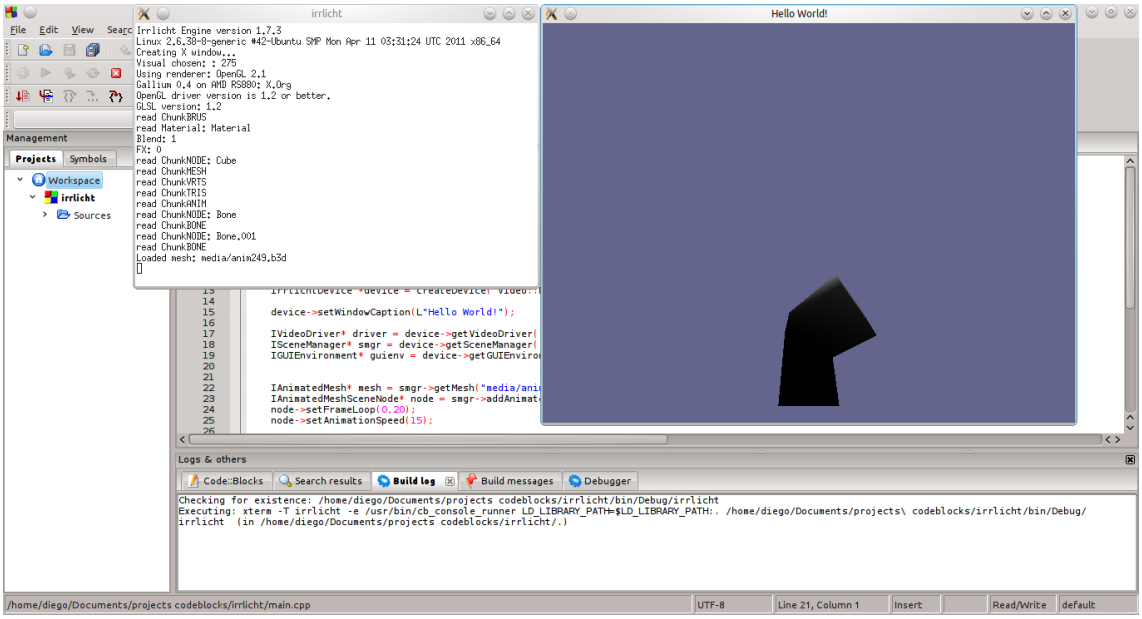


Figura 3.13: teste de importação na Engine Irrlicht - Figura do Autor

A IDE *Code Blocks* também possui suporte a *Ogre 3D*. Foi habilitada a extensão do *Blender* para exportação para *Ogre*, nativa do *Blender*, onde é possivel determinar que configurações da cena serão incluidas no arquivo .xml (formato reconhecido na *Ogre*). Para incluir as bibliotecas da *Ogre* ocorreram alguns problemas descritos a seguir. A IDE não reconheceu alguns arquivos de importação e configuração da *Engine* acusando incompatibilidade de versões, embora estivessem conforme sugestão do próprio site da ferramenta. Ainda assim, foi possivel desenvolver uma implementação para chamada

do objeto animado, como mostra a Figura 3.14, sendo que os prolemas citados foram contornados apenas repetindo o teste.

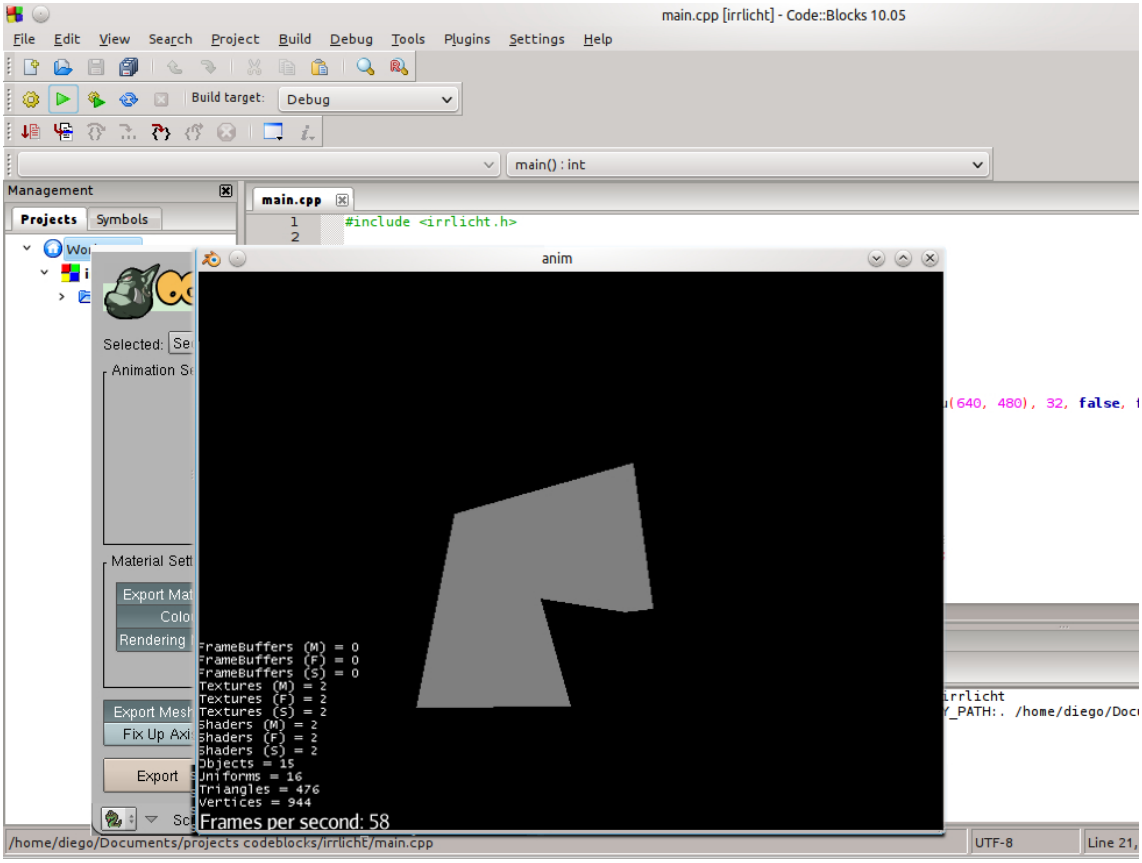


Figura 3.14: teste de importação na Engine Ogre 3D - Figura do Autor

A terceira *Engine* utilizada para o teste foi a própria BGE (*Blender Game Engine*). Neste caso foi necessário ativar uma extensão já existente na versão 2.6 do *Blender* para exportação em formato executável. Esta opção gera um arquivo independente com a animação definida na cena. O resultado porém se mostrou limitado pois não apresentava opção de movimento de câmera na janela resultante, seguindo a visão da câmera imposta na cena apenas. O resultado pode ser conferido na Figura 3.15, que apresenta o ambiente de desenvolvimento com a saída do objeto animado.

Por fim o teste foi realizado com a *Engine Jmonkey*, escolhida pelo fato de possuir exportações que podem ser executadas em um navegador. Esta ferramenta instala uma versão adaptada da IDE *NetBeans* baseada na 5.0, sendo que a versão mais atual desta IDE é a 7.1. A linguagem utilizada em sua programação é Java, e sua extensão própria é a .j3o. Foram constatados alguns problemas, por exemplo o fato da Engine não reconhecer

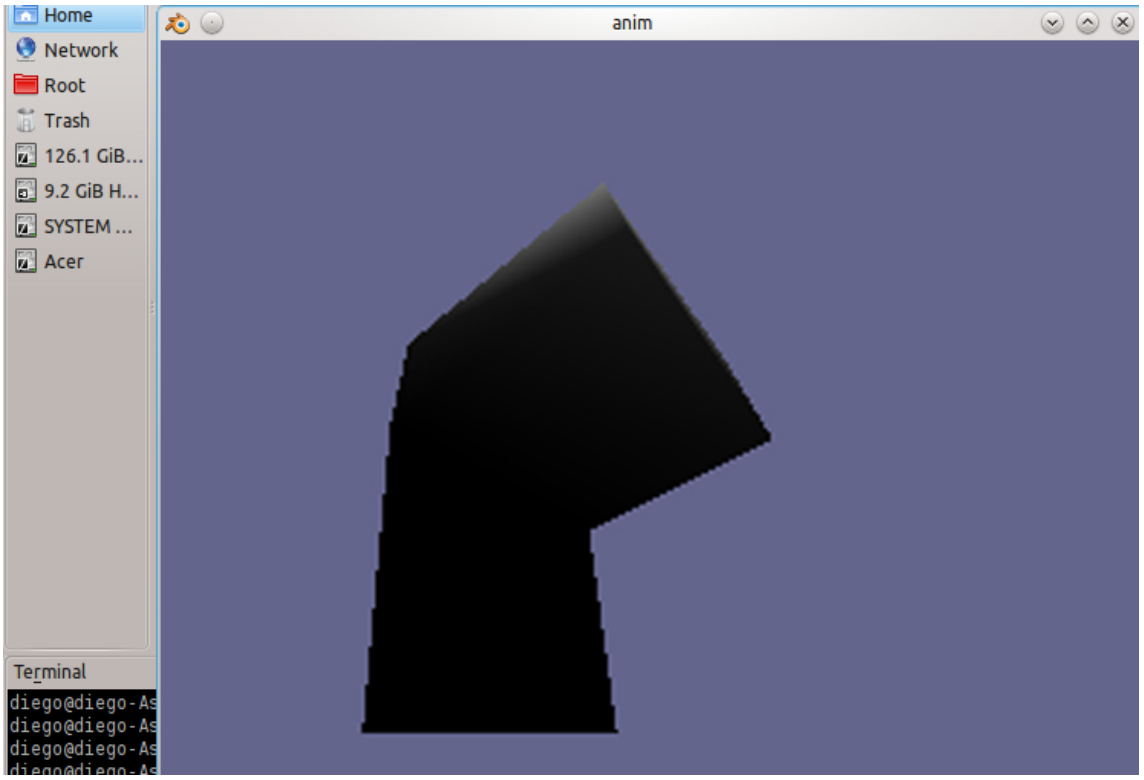


Figura 3.15: teste de importação na Engine BGE - Figura do Autor

exportações feitas em sua própria extensão, além das poucas opções de formatos que ela reconhece, sendo necessário utilizar a extensão própria do *Blender* como única opção funcional. Foi implementado um código para chamada do braço de teste animado, gerando o resultado visto na Figura 3.16.

Por fim, conclui-se que em todas foi possivel importar um objeto com animação própria, porém, todas apresentaram problemas em maior ou menor escala, tais como comandos que não funcionam como deveriam, travamento inesperado das IDE's ou de recursos específicos. A *Jmonkey* se mostrava promissora por possuir exportação para *Web*, contudo, foi a que mais apresentou inconvenientes, como seu *player* para renderização que travou diversas vezes, ou o fato de não reconhecer componentes e extensões. A *Ogre* por sua vez demonstrou problemas na fase de compilação e configuração da Engine na IDE *Code Blocks*.

Com relação a tempo de renderização todos foram satisfatórios, sendo que desde a execução da tarefa até a renderização final da animação o tempo de espera foi menor que 2 segundos em todos os casos.

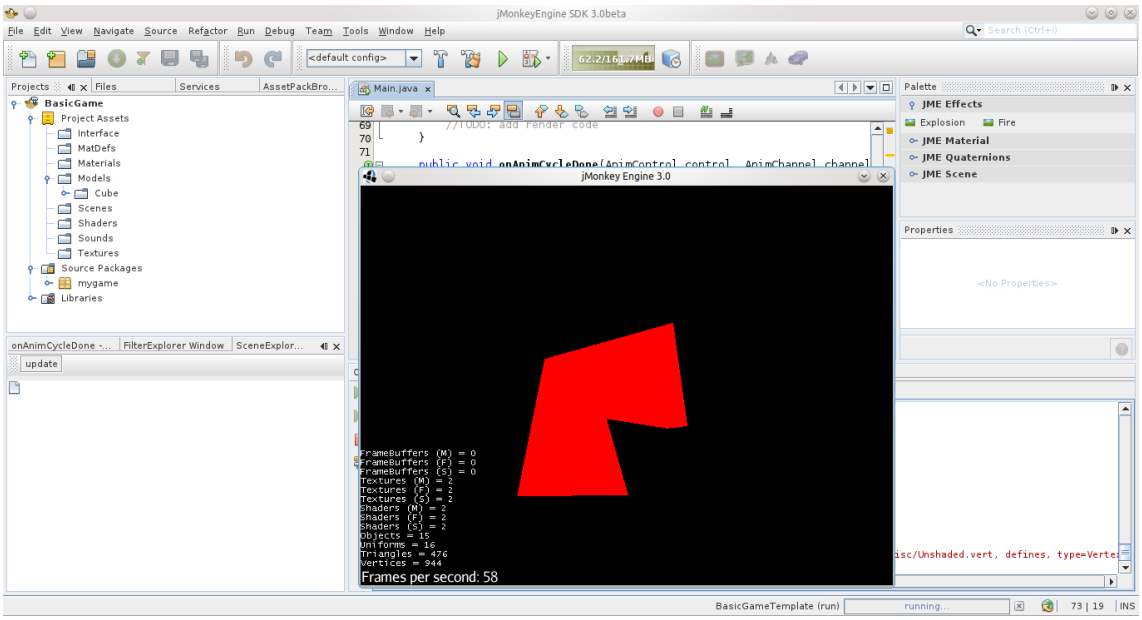


Figura 3.16: teste de importação na Engine Jmonkey - Figura do Autor

Para o trabalho foi utilizada a *Irrlicht*, pois se mostrou a mais simples na fase de configuração, como observado nos testes.

Foi realizado então o teste seguinte utilizando o Avatar 3D desenvolvido neste trabalho. Foi feita uma animação onde o braço esquerdo é movimentado. Como a *irrlicht* trabalha com o formato .b3d, e o exportador deste formato só é encontrado para versões antigas do *Blender* o modelo do avatar e o sistema de *rigging* tiveram que sofrer pequenos ajustes e alterações, onde a malha sofreu uma distorção ficando deslocada em relação ao esqueleto do *rig*, após alguns ajustes no próprio *Blender* realocando o objeto no espaço de trabalho do *Blender*, foi possível obter os resultados esperados, sendo feita a chamada da animação através do Avatar, na própria engine como apresentado na Figura 3.17.

No teste o Avatar aparece representado com uma textura preta pois ainda não havia sido inserido nenhum material na malha, que no caso de não haver cores ou texturas, a *engine* carrega um material padrão, geralmente uma cor chapada aleatória. Na imagem é possível observar dois Avatares, o primeiro com a posição inicial e o segundo com a posição final.

Concluindo, neste capítulo foi desenvolvida a malha referente ao Avatar 3D, e sua estrutura de animação. Ainda foram realizados os primeiros testes com o Motor Gráfico, ferramenta que será utilizada para o processo de síntese.

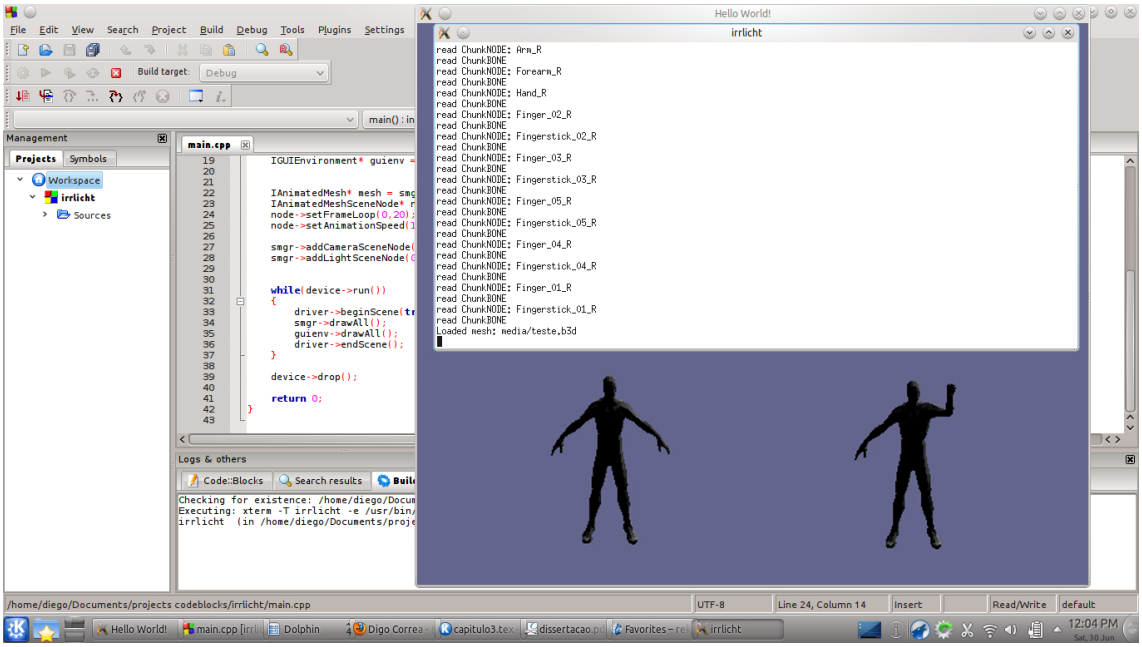


Figura 3.17: teste de importação na Engine Irrlicht utilizando o Avatar 3D - Figura do Autor

CAPÍTULO 4

SÍNTESE AUTOMÁTICA DA LÍNGUA DE SINAIS BRASILEIRA COM O AVATAR DESENVOLVIDO

Até este ponto, tem-se a criação de um avatar 3D funcional e otimizado, gerando uma malha 3D com poucos polígonos, com estrutura de animação organizada e que atende ao modelo formal descrito e seus parâmetros, além de testes iniciais e de configuração no uso de motores gráficos.

O objetivo das próximas etapas deste trabalho é o estudo e adaptação do modelo formal descritivo da língua de sinais brasileira para o modelo 3D ao qual será aplicado, construindo uma estrutura que, a partir de entradas baseadas no modelo formal, o avatar 3D possa representar funcionalmente os movimentos referentes a sinais ou elementos individuais que formam um sinal, além de ser atualizável para novos cenários.

Então, para gerar um sinal, devem ser encontrados meios para que, dada uma entrada referente a descrição formal, o avatar 3D possa interpretar e gerar uma saída equivalente, como mostrado na Figura 4.1, onde, por meio do XML, o agente virtual altera a configuração para gerar m movimento, de forma automática.

Então, os próximos passos se concentram em duas áreas: a transcrição do modelo formal representado pelo formato XML para pontos de referencia para o avatar 3D e a geração de uma saída dos movimentos através do agente virtual.

4.1 VPC - Vetor de Parâmetros de Configuração

O modelo descritivo para uso computacional é dividido em elementos e sub-elementos (a mão, por exemplo, e tratada como elemento no modelo formal, a configuração da mão, consequentemente, é um exemplo de sub-elemento para este caso), e seus respectivos valores. A fim de identificar os pontos referentes ao modelo, foi construída uma árvore hierárquica em grafo baseado na descrição XML adaptada do modelo IHC-SL [2].

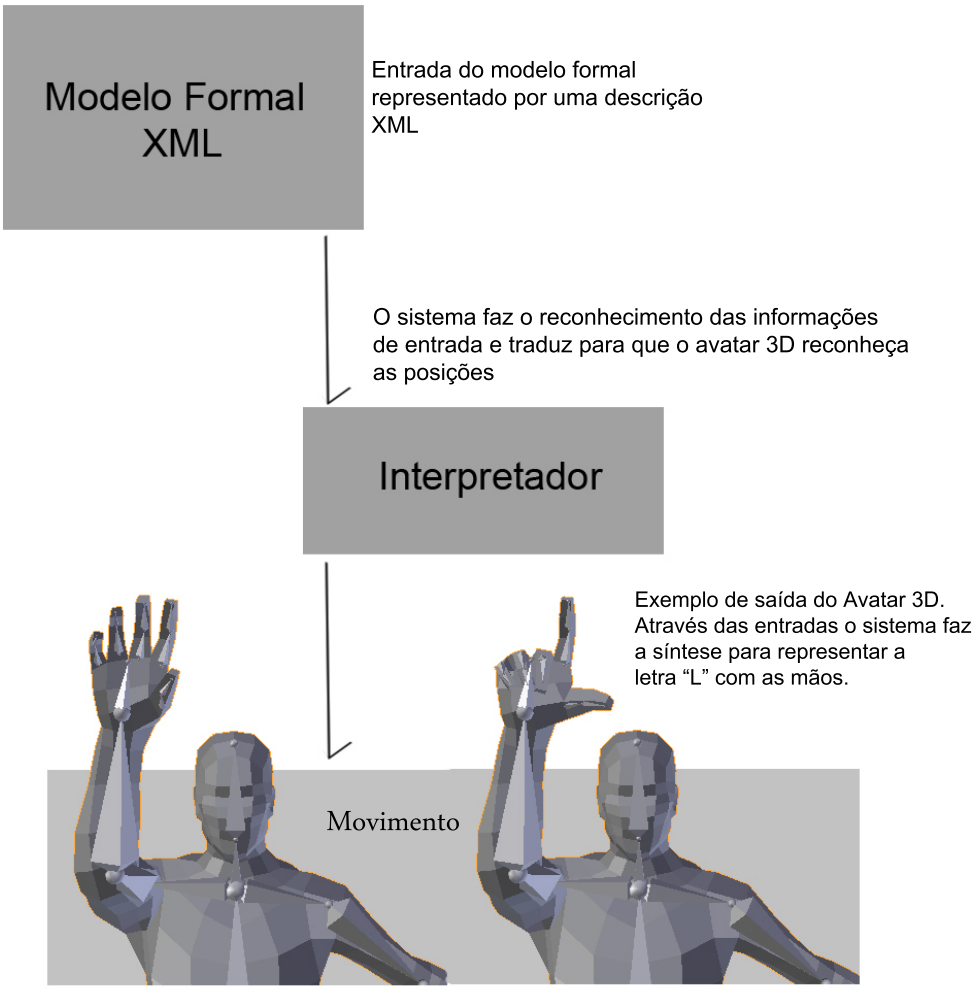


Figura 4.1: Síntese por meio do Avatar 3D - Figura do Autor

Esta árvore facilita a visualização e entendimento do modelo formal. Sua relevância, ao trabalhar com elementos espaciais e temporais, e não com sinais, é que suas entradas são independentes de sinais específicos. Ainda, suas entradas não trabalham com soletração, sendo um modelo desenvolvido especificamente para uso da comunidade surda, sendo que o usuário não precisará saber a língua portuguesa.

A intenção do modelo é atender todos os sinais existentes em seus parâmetros, embora, como citado anteriormente, este modelo não é completo nem definitivo, uma vez que determinados elementos ainda não foram mapeados e, assim como toda língua, a Libras sofre alterações e incrementos com o tempo refletindo no modelo IHC-SL, consequentemente.

As figuras a seguir apresentam o grafo referente ao modelo. Ele foi construído utilizando o sistema apresentado em [49] baseado no *software* livre *GraphViz* [13] próprio para criação de representações gráficas como diagramas UML. Foi criada uma entrada XML baseada nos elementos e parâmetros do modelo formal, e então gerado os grafos das figuras a seguir.

Na figura 4.2, o modelo formal tem como base pai o elemento principal Sinal, e como filhos hierárquicos respectivamente: o elemento suspensão, as expressões não manuais, os movimentos e sinais compostos. Estes elementos serão detalhados nas próximas imagens, contudo, ja é possível analisar no grafo o valor sequencia para o elemento sinal composto. Este elemento pode guardar informações como simetria dos braços na execução de algum movimento, ou repetição de sub-elementos.

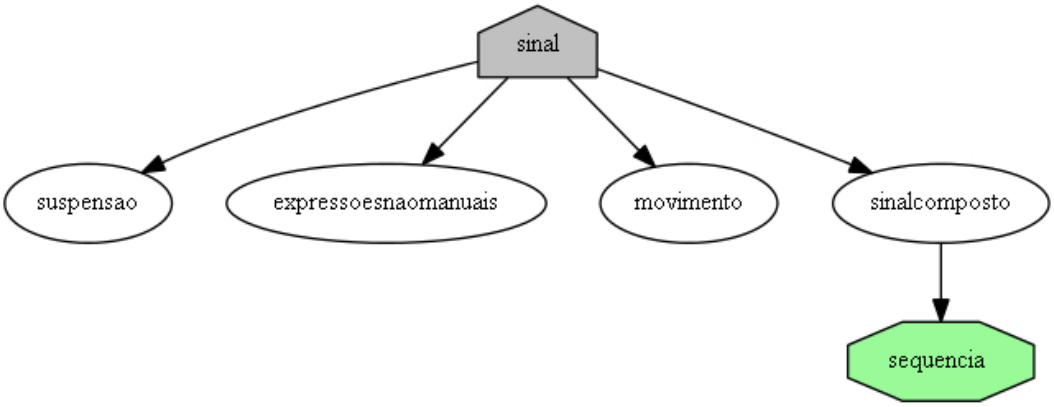


Figura 4.2: Árvore representativa do Modelo Formal - Elementos Principais

As figuras 4.3, 4.4, 4.5 mostram os sub-elementos contidos em cada um dos nós, mais detalhadamente. Sua estrutura segue o modelo IHC-SL apresentado em [2]. Na figura 4.3, temos a diferenciação entre mão dominante e mão não dominante, sendo que a segunda tem os mesmos sub-elementos da primeira, servindo apenas para diferenciar a mão do locutor na realização dos sinais. As mãos podem ter um movimento local, que pode ser no pulso, mão e antebraço, além da orientação da palma da mão (onde a palma está direcionada, por exemplo, pra cima, para baixo). Temos também a locação que indica onde a mão está posicionada com relação a um sub-elemento como a cabeça ou o tronco, por exemplo. Ainda temos a configuração de mãos, que define quais das 61 configurações

da Libras a mão vai assumir.

Figura 4.3: Árvore representativa do Modelo Formal - Elemento Suspensão

A figura 4.4 mostra as expressões não manuais, que são os elementos externos aos braços e mãos como expressões faciais e movimentos de tronco e cabeça.

E por fim, a figura 4.5 apresenta os sub-elementos do elemento movimento. Assim como na suspensão, temos a mão dominante e a não-dominante, com os mesmos sub-elementos. Dentre eles podem ser destacados o tipo que define, por exemplo, se o movimento possui

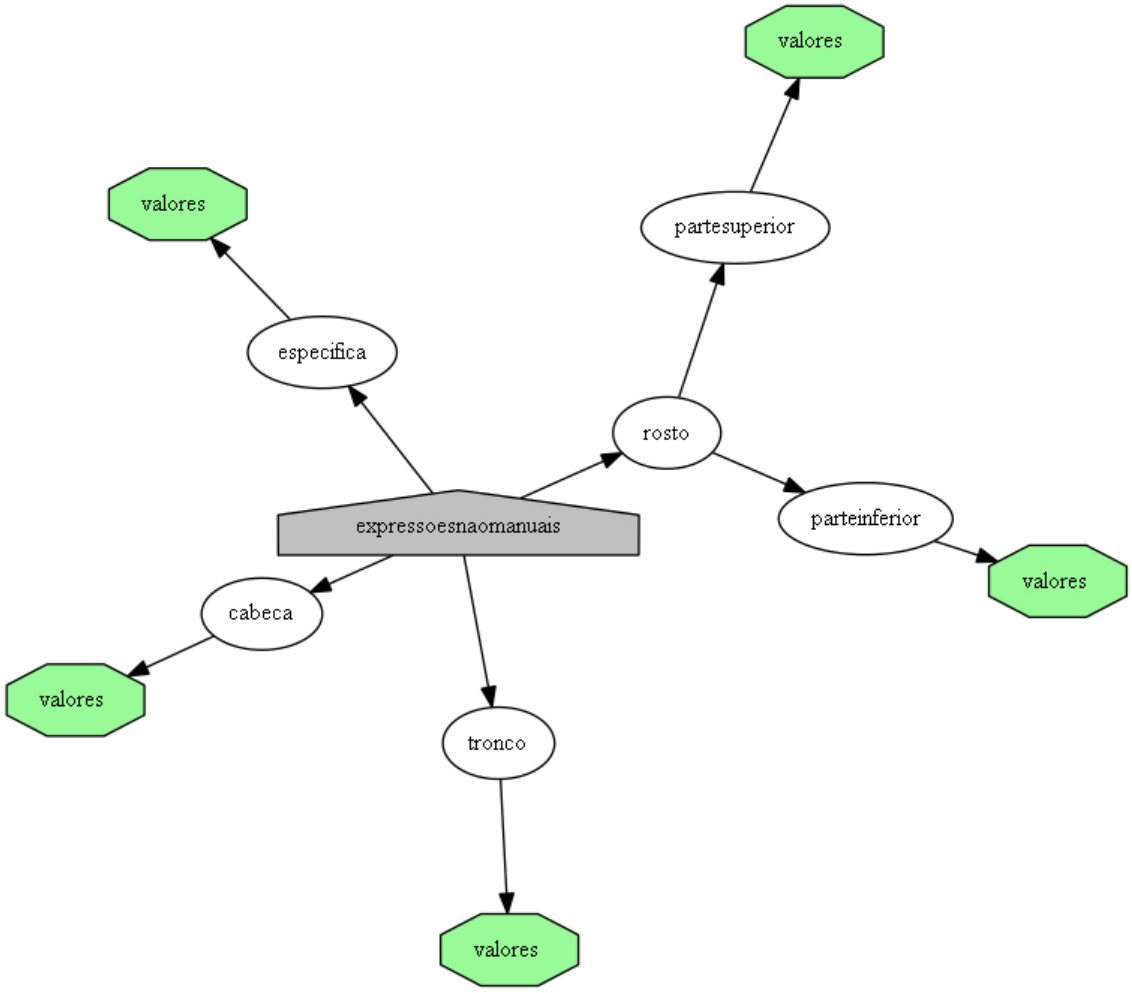


Figura 4.4: Árvore representativa do Modelo Formal - Elemento Expressões Não Manuais

algum contato com outra parte do corpo. Temos a qualidade que define, por exemplo, a velocidade do movimento.

A entrada utilizada para gerar este grafo, em formato XML, pode ser encontrada a seguir. Como pode ser observado, a entrada foi construída de maneira otimizada e com o objetivo de simplificar o máximo possível sua formatação. Em geral cada elemento é formado por *tags* que definem seu nome e valor, e faz parte de um bloco (que pode estar incluído em um bloco pai).

O Vetor de Parâmetros de Configuração, como visto anteriormente, é um vetor de parämetros de configurações extraído de uma entrada que segue o modelo descritivo IHC-SL, incluída em um arquivo XML. A *engine* gráfica Irrlicht trabalha com a linguagem C++, logo, o sistema desenvolvido neste trabalho utiliza esta linguagem em seus códigos.

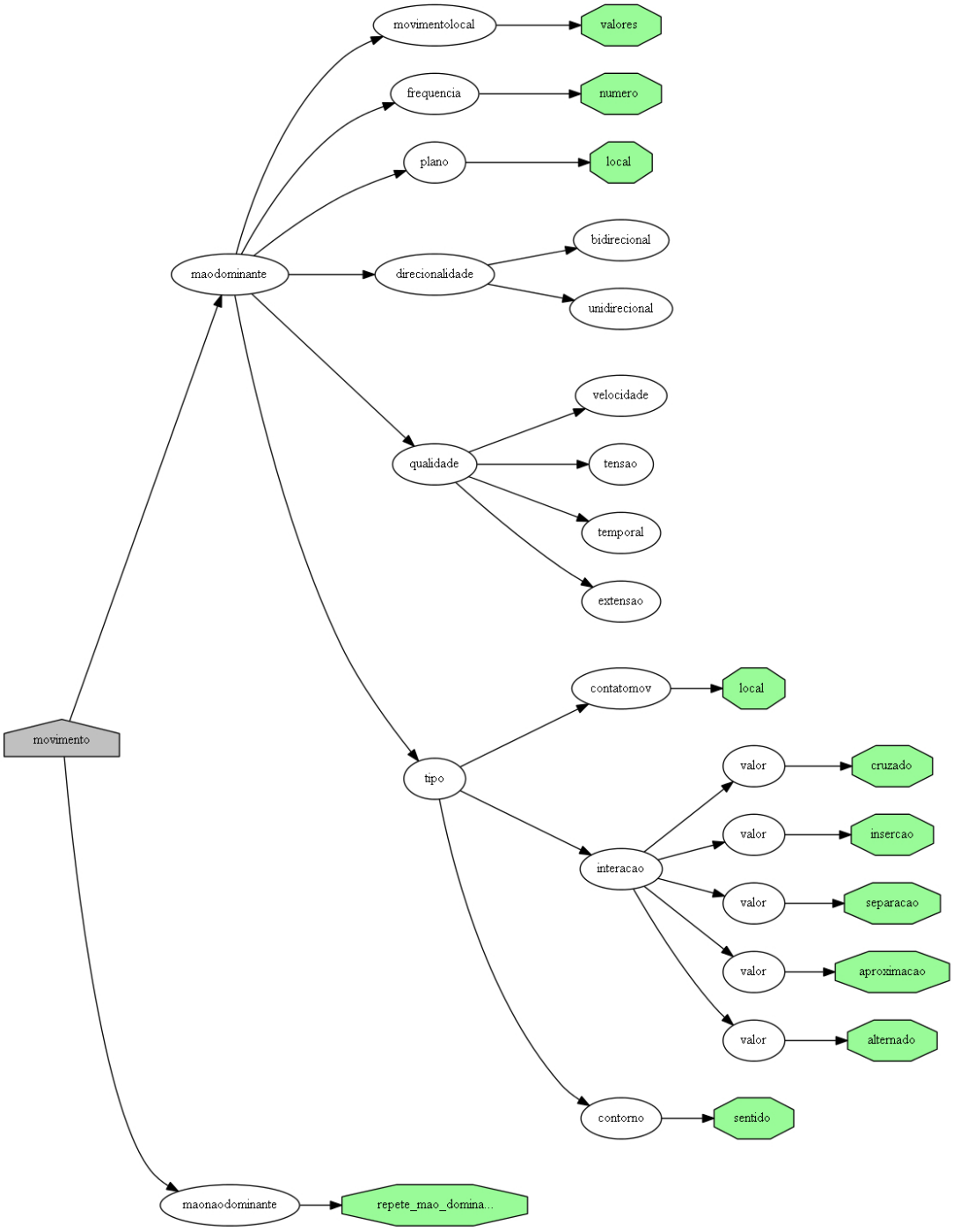


Figura 4.5: Árvore representativa do Modelo Formal - Elemento Movimento

Partindo deste princípio foi necessária a integração de um *parser* baseado em XML para leitura das entradas, implementado especificamente com a linguagem C++.

Na parte inicial do funcionamento do sistema proposto, obtida uma entrada referente ao VPC, com o formato XML, o sistema deve ler o arquivo de entrada e reconhecer sua

estrutura, elementos e valores. A estrutura é a ordem hierárquica em que os elementos estão dispostos. Os elementos são os nós que compõe a cadeia e seus valores são as posições que cada elemento do VPC deverá assumir (como exemplo, uma das 61 configurações de mãos especificadas na Libras). A figura 4.6 mostra como é o funcionamento desta etapa inicial.

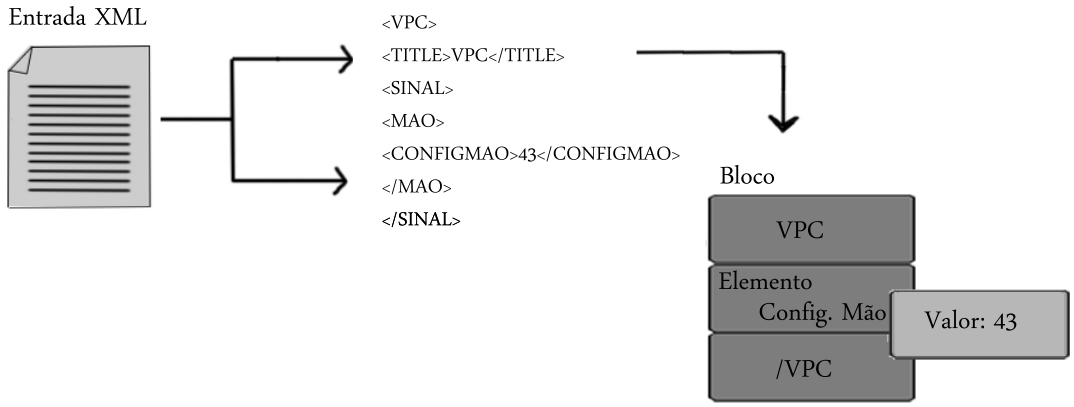


Figura 4.6: Leitura do Arquivo de Entrada - Figura do Autor

Um *parser*, ou analizador sintático, é um processo que interpreta e reconhece índices de uma estrutura gramatical segundo uma determinada gramática formal [27].

De forma geral, este processo consiste em dividir uma entrada em uma estrutura de entrada de dados, onde será identificada a hierarquia sintática em que o texto se baseia, dividida por blocos. No caso da entrada XML construída para este trabalho, baseada no modelo apresentado anteriormente, os blocos são as *tags*, ou nós, referentes ao modelo descritivo.

Ainda segundo [27] os analisadores sintáticos podem trabalhar de duas maneiras, de cima pra baixo, analisando a estrutura a partir de nós maiores até seus subsequentes filhos, ou pelo caminho inverso, procurando elementos básicos e identificando posteriormente elementos maiores com escopo mais abrangente que utilizam este elemento básico. Baseado no modelo formal, o analisador poderia identificar elementos pai, como "movimento", ou "expressões não manuais"e posteriormente subdividir e identificar os elementos seguintes da estrutura, até chegar nos elementos mais baixos, no caso, os valores finais.

Existem vários analisadores sintáticos implementados, específicos para trabalhar com arquivos XML, além dos citados anteriormente, como o encontrado na biblioteca *TinyXML* [47] (ou sua variante *TinyXML++*), que utilizam uma estrutura bem simplificada e prática, a biblioteca *Xerces* [45], também implementada em C++, a biblioteca *CMarkup* [7], que trabalha de maneira semelhante as citadas anteriormente, entre outras.

O desafio nesta etapa, foi encontrar uma analisador sintático que trabalhasse em conjunto com o motor gráfico, no caso a *Irrlicht*.

Foram realizados então testes de integração, a princípio independentes do uso do motor gráfico, entre a aplicação principal em C++ e o arquivo de entrada referente ao VPC em XML. O teste consistiu em criar uma função na aplicação principal que consiga ler o arquivo de entrada XML, extrair e reconhecer seus dados, e passar as informações em variáveis da aplicação para que possam ser utilizadas no processo ou em momento de execução. As informações do arquivo teste eram divididas em 2 blocos separados por *Tags*, o primeiro foi chamado de "teste1"com o valor "1", e um segundo bloco chamado de "teste2", possuia o valor "2". Basicamente o sistema deveria ler este arquivo através da aplicação, separar corretamente os blocos e identificar seus valores.

Inicialmente foi testada a integração do *TinyXML* em conjunto com a *Irrlicht*. Foram observados alguns problemas ao se trabalhar com a *engine* gráfica, onde algumas funções ou métodos entravam em conflito e geravam erros normalmente inexistentes de fato. A aplicação deixava de reconhecer o arquivo de entrada, ou eventualmente perdia a sequência de blocos trazendo erros nos valores encontrados (quando os mesmos estavam corretos).

Com o mesmo teste, a biblioteca *Cmarkup* se apresentou estável, funcionando sem problemas em conjunto com o motor gráfico, sendo escolhida então para uso neste trabalho.

Foram integrados os arquivos necessários encontrados no pacote da biblioteca *Cmarkup*, para ambiente do sistema principal. Em seguida foi criado um objeto do tipo *Cmarkup* na aplicação, configurado o arquivo de entrada (nomeado de vpc.xml) e utilizado métodos para navegação dentro deste arquivo, no caso os métodos *FindElem*, utilizado para tentar encontrar os blocos de dados, *IntoElem* utilizado para encontrar valores especificados dentro do bloco em que o ponteiro aponta, e *FindChildElem*, utilizado para identificar

blocos dependentes do atual, no caso, os nós filhos da cadeia hierárquica do VPC. Com o uso desta biblioteca e estes métodos, é possivel o controle, leitura e reconhecimento da entrada de VPCs.

Concluída esta fase, foi possível iniciar a implementação da síntese dos dados de entrada para a saída através do Avatar 3D.

4.2 Síntese Automática Através do Avatar 3D

Uma vez que a classificação dos elementos contidos na entrada de VPCs é realizada pela aplicação, o próximo passo foi fazer com que o Avatar3D reconheçesse as informações contidas no arquivo XML, e retornasse a saída referente ao movimento descrito, sendo este o último passo no sistema proposto neste trabalho.

Os arquivos de entrada criados para testes na seção anterior, foram utilizados para esta etapa. Os elementos e seus valores são descritos de forma textual na entrada de VPCs, onde temos tags representando cada nó filho, e subsequentemente sua posição, de forma descritiva e não em coordenadas espaciais. Na sequencia, o processo de conversão da entrada para animação ocorre dentro da aplicação, de maneira independente e automática.

Para se fazer a animação e chamada dos parâmetros, dois caminhos podem ser assumidos.

O primeiro método é fazer com que o código gere os movimentos da estrutura de animação através da *Irrlicht*, fazendo chamada de cada ponto de articulação e sua alteração de posição. Neste sentido, cada bone deve ser movimentado seguindo a árvore hierárquica apresentada anteriormente, do elemento principal, até o último sub-elemento da cadeia.

O código indica qual *bone* será movimentado naquele momento e sua posição em coordenada X, Y, Z. Após movimentar o primeiro elemento da cadeia do VPC ele executa o mesmo procedimento na cadeia seguinte (elemento filho), e repete o procedimento até o ultimo elemento da cadeia do VPC.

Neste processo o código só precisará do Avatar texturizado e ligado à estrutura de animação (*armature* e *bones*), todos os cálculos e procedimentos serão executados a partir do sistema principal através da *Irrlicht*, utilizando como entrada o arquivo XML referente ao VPC.

A principio, então, é construída a entrada no formato XML, referente ao sinal que será representado pelo Avatar 3D. A entrada segue o modelo apresentado na seção anterior adaptada da estrutura IHC-SL. Com a entrada, o sistema reconhece qual o valor da posição descrita no XML, sendo que o elemento atual é o mais alto da cadeia hierárquica ainda não processado. Esta posição possui sua descrição, no código, em coordenadas espaciais em x, y e z, e qual estrutura (no caso, o *bone*, *bones* ou *dummies*, que são objetos controladores) correspondente aquele elemento.

A figura 4.7 mostra o funcionamento geral do processo de síntese realizado desta maneira. Como pode ser observado, os cálculos de posições referentes à entrada VPC acontecem no momento do processamento principal, através do motor gráfico.

A segunda maneira de se chegar nos resultados propostos, segue em estrutura muito parecida com o caso anterior, funcionando de maneira diferente na execução da síntese das animações.

O processo inicial, que consiste na entrada XML, com os elementos referentes ao VPC, funciona da mesma maneira que no caso anterior, bem como sua leitura através do *parser*. O funcionamento da síntese em si é semelhante à maneira utilizada na industria de jogos, onde, em resumo, cada elemento (ou parte independente de um bloco do modelo formal) é implementada no editor 3D (no caso deste trabalho, o *Blender*), e armazenada em um banco de sequências de *frames* referentes a elementos do modelo.

Existem alguns motivos que justificam este método ser mais vantajoso que o citado anteriormente, entre eles no desempenho e menor possibilidade de erros de renderização ou imprevistos. As vantagens e desvantagens serão citadas mais detalhadamente nos testes apresentados a seguir.

A figura 4.8 apresenta, de maneira geral, o funcionamento deste método.

Para este trabalho foi desenvolvido um teste baseado nos dois métodos apresentados nas Figuras 4.7 e 4.8, que consiste em fazer a leitura do VPC e síntese através do Avatar 3D.

Iniciando dos pontos comuns entre os dois métodos, foi criada a entrada em XML, para se extrair o VPC. Esta entrada possui 3 níveis hierárquicos, iniciando do nó principal

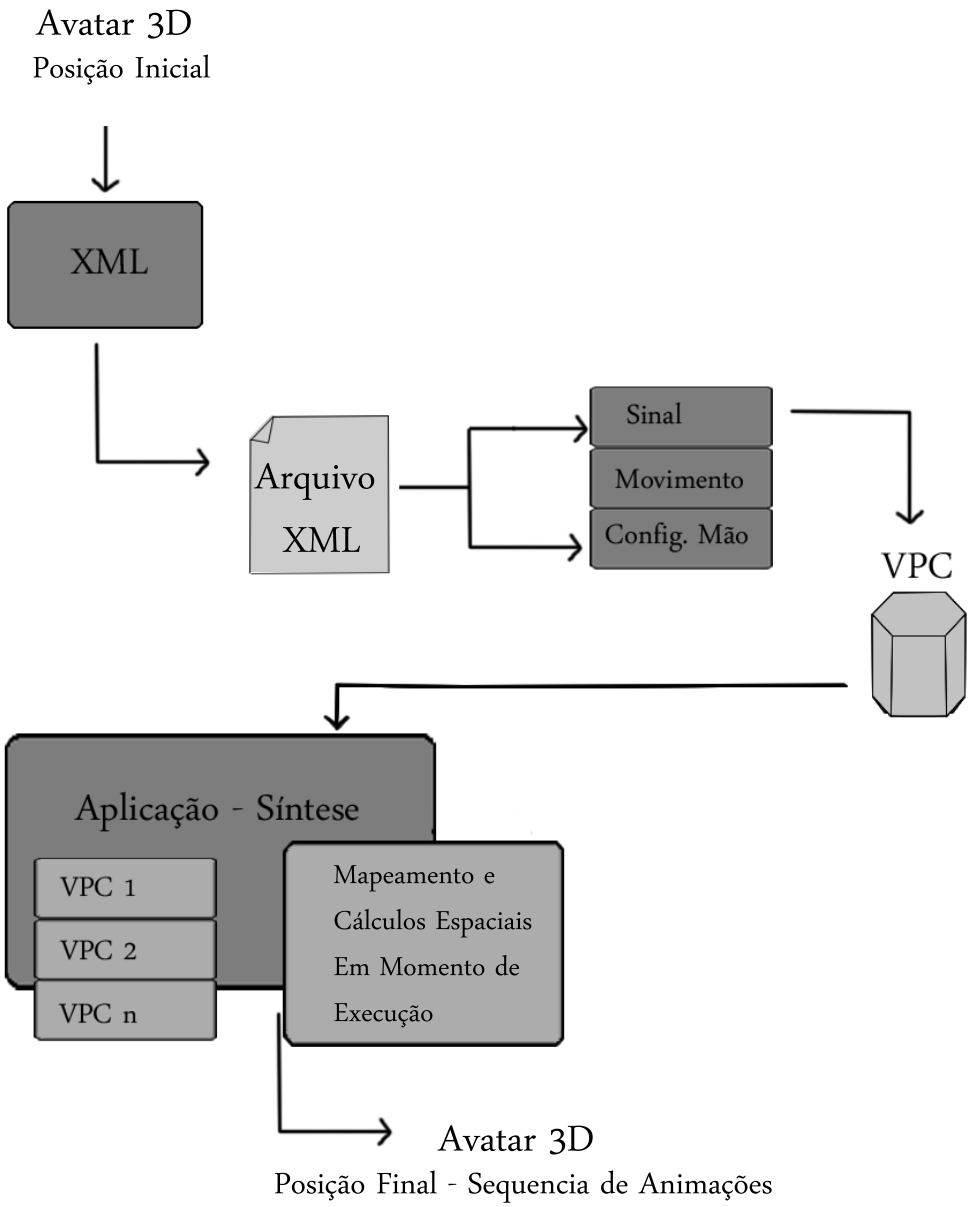


Figura 4.7: Síntese Com Execução Interna - Figura do Autor

chamado de Sinal, logo após um movimento simples do braço direito, no bloco Movimento (com a intenção de testar a movimentação de um elemento acima da mão, apenas), e, enfim, um ultimo nível para representação de uma configuração de de mão teste, com o bloco Configmao (referente a configuração de mão), implicando um movimento simples para síntese que trabalha com mais de um elemento diferente de parâmetros de configuração. Para efeitos de testes apenas, o bloco Movimento recebe o valor Movimentobraco e o bloco Configmao recebeu o valor 41, referencia a configuração de mão número 41 das

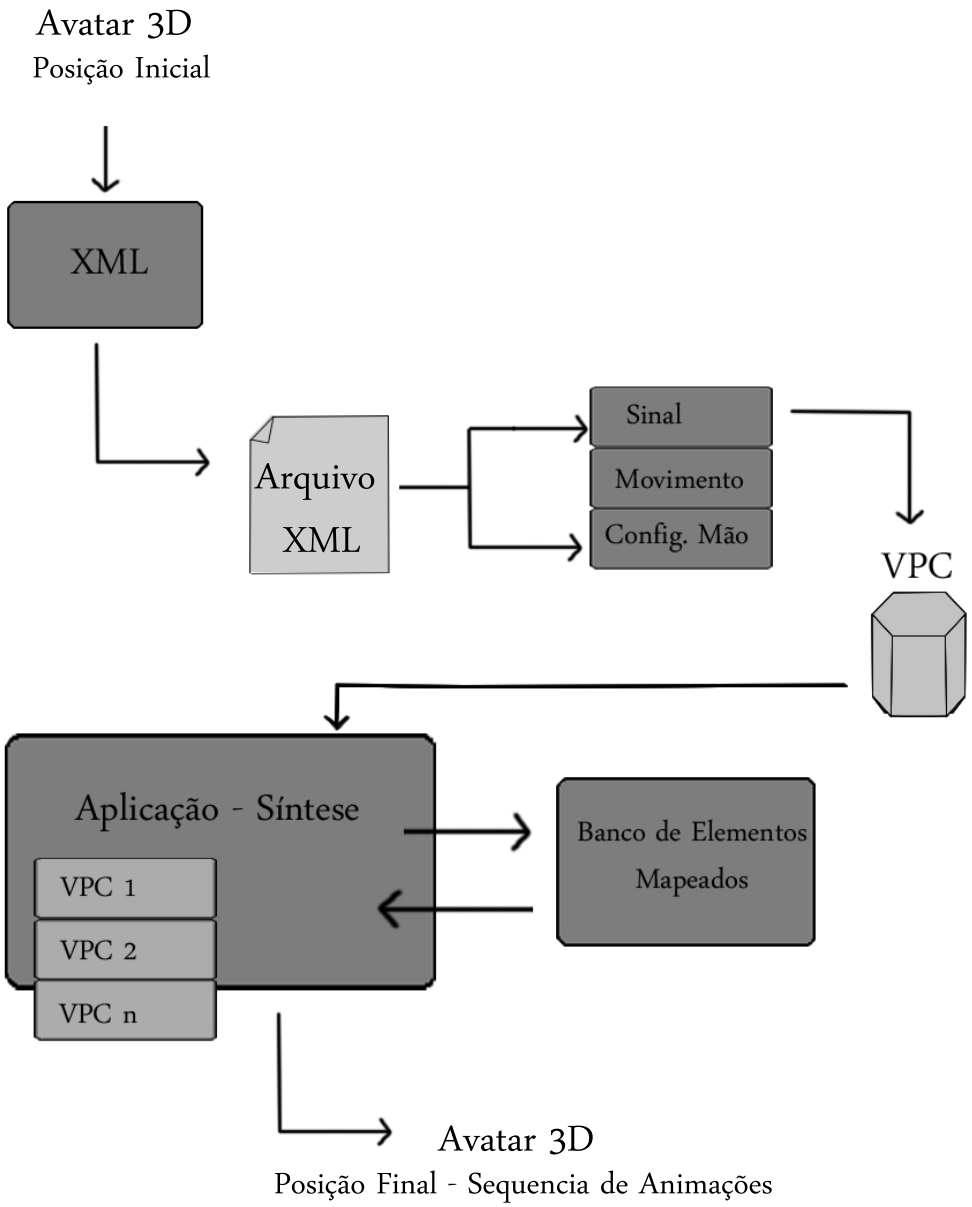


Figura 4.8: Síntese Com Execução Externa - Figura do Autor

61 configurações de mãos da Libras, como mostrado na Figura 4.9.

Com esta entrada ja foi possível realizar os testes principais da síntese entre a entrada de VPCs e o Avatar 3D.

Conseguindo gerar a animação através desta entrada, é possível testar basicamente o funcionamento geral do sistema, uma vez que, independente de quantos elementos serão utilizados em um VPC, da maneira como estes elementos sejam representados, bem como seus valores, o sistema para síntese será sempre o mesmo.

Figura 4.9: Configurações de Mãos da Libras [2]

A entrada representada pelo arquivo XML utilizou a estrutura apresentada na seção anterior. Em seguida foi indicado ao sistema qual o VPC a ser utilizada, sendo papel do *parser*, identificar os elementos contidos na entrada. A intenção dos testes descritos a seguir foram realizar o proposto na Figura 4.10, onde temos duas animações de elementos diferentes, sendo geradas automaticamente baseadas em uma entrada externa, integradas em uma cadeia hierárquica.

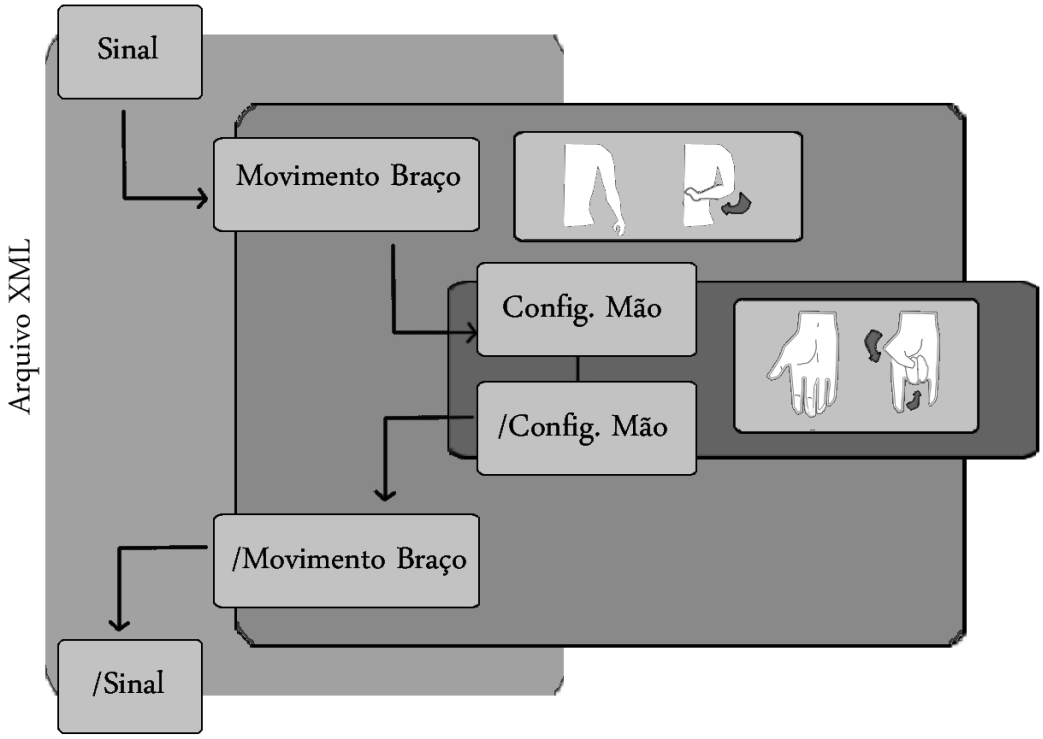


Figura 4.10: Teste Realizado - Figura do Autor

Para este primeiro caso, foi realizado um teste completo, partindo do princípio de integrar o parser ao ambiente do projeto final e utilizar os métodos explicados na seção anterior para conseguir identificar este arquivo de entrada bem como as informações contidas nele.

O arquivo de entrada utilizado é o mesmo explicado anteriormente. Foi realizado um teste simples onde os dois valores de elementos contidos no arquivo de entrada foram identificados e passados para variáveis da aplicação para que pudessem então ser utilizados pelo sistema.

Com os valores Movimentobraco e Configmao encontrados, o próximo passo seria gerar as animações de saída baseadas nestes valores. Como o elemento de nível maior (seguindo a árvore apresentada anteriormente, referente ao modelo formal) não possui dependencia de elementos anteriores, a animação do movimento do braço deveria ser calculado, e executado, anteriormente. Para isso foram utilizados os métodos do motor gráfico *Irrlicht*

encontrados em *IBoneSceneNode.h*, que oferece suporte a Avatares 3D com estrutura de animação mas sem mapeamento de animações ou elementos.

Então, com os valores identificados do arquivos de entrada, foi desenvolvido o cálculo primeiramente do elemento de movimento do braço, já que esta cadeia não depende da outra para ser executada, e em seguida foram realizados os movimentos dos *bones* do braço e antebraço direito utilizando os métodos do motor gráfico *positionHint* e *rotationHint*, responsáveis por animações de posicionamento e rotação respectivamente. Em seguida estes métodos foram utilizados nos bones dos dedos e mão para que a mão asumisse a posição 41 citada anteriormente.

O segundo bloco, referente à configuração de mão, localiza-se dentro de uma estrutura de repetição controlada por um ponteiro, que indica se a estrutura hierárquica encontrada no arquivo de entrada já foi totalmente percorrida. Assim, a animação de saída é gerada de forma sequencial, respeitando a ordem de execução do modelo formal.

Para o segundo teste, foi realizado um procedimento bastante conhecido na indústria de jogos eletrônicos, que utiliza um mapeamento dos elementos em um banco de *frames* referentes aos nós de VPCs. Na *Irrlicht* os métodos para se trabalhar desta maneira estão mapeados em *IAnimatedMeshSceneNode.h*, que oferece suporte a Avatares que já possuam elementos mapeados anteriormente.

Este mapeamento pode ser feito no próprio editor 3D, no caso deste trabalho, o *Blender*. Para isso é identificado o elemento que deverá ser representado na animação de saída, para cada elemento ou sub-elemento, independente do nó prévio e posterior do modelo formal, já que cada elemento deve trabalhar de forma individual e independete.

São processados os cálculos de animação dos *bones* em uma sequencia de *frames* referentes a cada elemento da cadeia de VPCs. Desta forma é possivel mapear todos os elementos do modelo formal, e utilizá-los na aplicação quantas vezes forem necessárias, independente da cadeia de entrada de VPCs tornando assim a aplicação flexivel e ágil, uma vez que o mapeamento de cada elemento é implementado nesta etapa, uma única vez.

Para se testar este procedimemento foram mapeados os elementos de teste referentes às

entradas do arquivo XML. Cabe citar que a entrada utilizada para este teste foi a mesma utilizada anteriormente, bem como a leitura e identificação dos elementos do arquivo XML através do *parser*.

Mais especificamente, foram mapeados dois elementos para testar a chamada e execução (a síntese proposta) dos elementos de VPCs. No caso, tratam-se dos blocos de Movimento e ConfigMao citados anteriormente, já que o nó pai do arquivo de entrada utilizado nos testes não possui valor para ser mapeado. Como no processo anterior, foi definido que o bloco de movimento consiste de uma simples movimentação do braço direito, para testar o trabalho com mais de um bloco de elementos, e o bloco de configuração de mão consiste na mão direita do avatar assumir a configuração de mão número 41 da Libras. Então foi definido uma cadeia de *frames* sequenciais referentes a cada bloco e feito seu mapeamento, através dos cálculos e alterações espaciais de cada ponto da estrutura controladora.

Primeiro foi mapeado o elemento de nível maior, já que este não é depende de animações de elementos anteriores, no caso deste teste, o movimento do braço. Foi então efetuado um pequeno giro no braço direito, e mapeado em 40 *frames*.

Para este momento não foi levado em conta nenhum cálculo anterior. Em seguida foi mapeada a configuração de mão, sendo que, neste momento, os cálculos eram os valores do movimento do braço, somados as alterações nos bones controladores dos dedos e da mão.

O fluxo do processo de síntese segue o apresentado na Figura 4.10. É feita a identificação do arquivo XML de entrada, que deverá conter os dados para gerar o VPC. O sistema fará a leitura deste arquivo, e deve extrair os elementos do modelo formal e seus valores, divididos em blocos, guardando e convertendo esses valores para que possam ser utilizados na aplicação. Em seguida, são feitas as operações para identificar quais blocos deverão ser representado pelo avatar, e estruturar a sequencia de animação dos proximos blocos. Em seguida o sistema buscará os elementos do modelo formal no banco de elementos mapeados e executará a sequencia da entrada, através do motor gráfico, sem a necessidade de realizar nenhum cálculo de posição interno. Por fim o Avatar 3D fará a síntese da entrada, automaticamente, até que não exista no VPC extraído do XML

nenhum nó filho a ser animado.

Após reconhecer os blocos de elementos referentes aos VPCs, cada valor foi passado para uma variável na aplicação principal, para que pudessem ser utilizados e trabalhados no processo da aplicação.

É realizada então uma comparação com os valores extraídos do VPC de entrada, e, assim que a aplicação reconhece qual é o bloco de entrada, busca no banco de elementos mapeados e executa. Estas comparações seguem a sequencia hierárquica do modelo formal. Desta forma, a saída gerada pelo avatar 3D segue exatamente a ordem descrita no XML de entrada, e na árvore do modelo computacional, apresentado na Seção anterior.

Assim como no caso anterior, foram executados os testes necessários para que a saída fosse executada corretamente, sendo que o processo, nos dois casos, abordou todos os passos propostos neste trabalho, que inicia da leitura de uma entrada externa em XML, contendo o VPC em seu conteúdo, identificação e análise destes dados e a saída através de um avatar 3D como mostra a Figura 4.11.

Figura 4.11: Saída representada através do Avatar 3D - Figura do Autor

Neste capítulo foi visto mais sobre o modelo formal e foi desenvolvido o sistema de síntese do VPC extraído do arquivo de entrada. Com isso, foi possível alcançar o proposto para síntese, sendo que maiores detalhes como desempenho em cada um dos testes, resultados e discussões, são tratados no próximo capítulo.

CAPÍTULO 5

RESULTADOS E DISCUSSÃO

5.1 Testes e Validações

Os testes do capítulo anterior exercitam o processo proposto de síntese. A fim de testar o sistema em outros cenários para efeito de validação, foi desenvolvido um novo teste, baseado no segundo método apresentado no capítulo anterior.

O protocolo de validação consistiu em:

a) avaliar a segurança dos dados extraídos do arquivo de entradas externo, onde é definido o VPC.

b) avaliar a reação das alterações na malha, quando utilizados outros parâmetros de configuração, baseados nos dados do VPC.

c) avaliar a consistência do processo todo, incluindo a saída através do motor gráfico.

O método de mapeamento pelo editor 3D, se mostrou mais consistente, como apresentado no capítulo anterior, não obstante, sendo adaptado da forma com que as indústrias de jogos trabalha onde o mapeamento de cada elemento do modelo formal deve ser feito no próprio editor 3D, no caso deste trabalho o *Blender*, sendo que o modelo é exportado para o motor gráfico já com uma cadeia de *frames* de animação devidamente separados por elementos baseados no modelo.

Os novos testes executam o processo completo, sendo escolhidos dois sinais em Libras para se aplicar o processo de síntese: o sinal referente à palavra "Moto"e o sinal referente a palavra "Bicicleta". Como no teste realizado no capítulo anterior, o processo inicial foi construir a entrada em XML referente aos blocos do modelo formal.

Em seguida foi realizado o processo de mapeamento dos elementos, que define qual será a representação de determinado *bone*, ou *bones*, referente a uma entrada específica.

Para o sinal "Moto", foram definidos os seguintes elementos para o VPC: Simetria entre os bones do antebraço e mão, rotação do bone referente ao antebraço em 90 graus,

configuração de mão 8 (sendo qe todos os *bones* dos dedos recebem movimento para assumir a posição 8), rotação do pulso da mão direita em 90 graus, repetição do ultimo elemento 3 vezes.

Então, foi realizado o processo de síntese onde foi feita a leitura do arquivo XML a fim de encontrar os elementos do VPC, descritos anteriormente, sendo que o sistema conseguiu encontrar todos os blocos de elementos em sequência hierárquica. Então, tendo a ordem de chamada e os elementos mapeados, o sistema gerou a saída correspondente, através do Avatar 3D como mostra a Figura 5.1

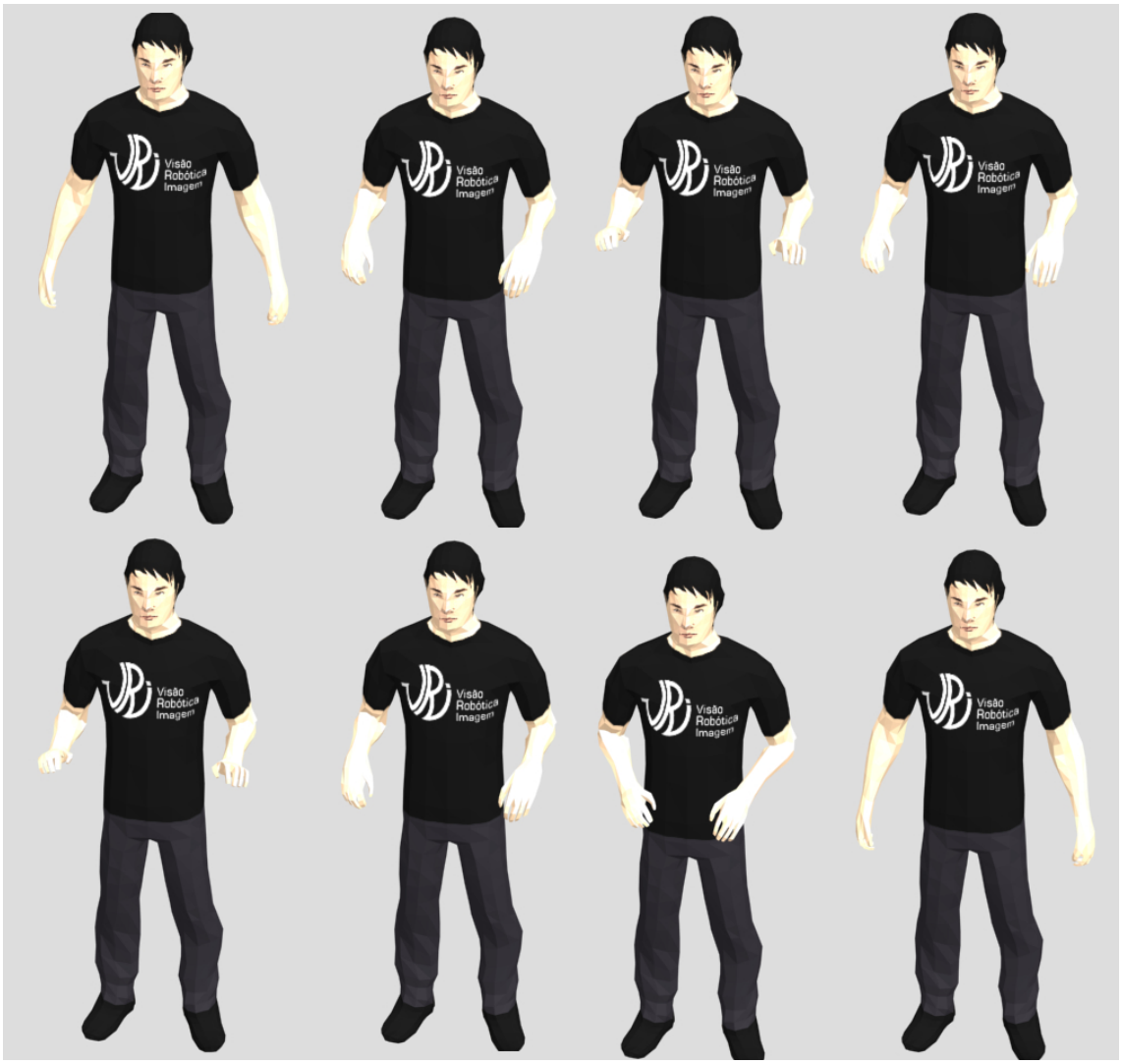


Figura 5.1: Representação do Sinal Moto utilizando o sistema de Síntese - Figura do Autor

A principal diferença nos elementos para o VPC do sinal "Bicicleta", em comparação ao anterior, foi a inclusão do bloco referente aos *bones* do braço, que neste caso receberam

movimentos de rotação de 45 graus tanto para o braço esquerdo quanto para o direito. Ainda, embora realizem os mesmos movimentos, os elementos, em geral, não receberam simetria, pois executam em tempos contrários para os dois braços e mãos.

O processo de síntese foi aplicado da mesma maneira dos testes anteriores, obtendo como saída a representação do sinal, como apresentado na Figura 5.2.

Figura 5.2: Representação do Sinal Bicicleta utilizando o sistema de Síntese - Figura do Autor

Para os dois testes não foram identificados erros ou imprevistos, e o processo de síntese ocorreu como esperado em todas as suas etapas, que são a leitura do arquivo de entrada e identificação do VPC, processamento e representação do sinal através do Avatar 3D. Todos os resultados obtidos serão discutidos na próxima seção.

5.2 Análise e Discussão

Todos os testes apresentados neste trabalho foram realizados em uma máquina com as seguintes configurações: processador AMD Turion II X2, placa de video ATI Radeon HD 5470, 4 GB DDR3 de moemória. Foram utilizados os sistemas operacionais Ubuntu 11.4 e Windows 7

Quanto a desempenho e tempo de execução, todos os testes obtiveram resposta de processamento inferior a 1 segundo, ja que a técnica, embora envolva gráficos 3D (normalmente processos pesados), não utiliza sistema de iluminação complexo, ou efeitos físicos como objetos secundários (cenários, ou malhas que não fazem parte do avatar) que carregam o processo de renderização.

Em casos futuros, renderizações mais complexas, com efeitos de iluminação que utilizem mais recursos, malhas mais complexas, entre outros fatores (efeitos de renderização pura como *ambient occlusion*, efeito conhecido nos principais renderizadores que trabalha a reação do objeto com sombras e iluminação ambiente), o desempenho pode ser prejudicado.

Com relação aos resultados, o segundo método do processo de síntese apresentado no capítulo anterior, que utiliza um banco de elementos mapeados no próprio *Blender*, se mostrou mais estável, como apresentado no capítulo anterior.

Nos testes do primeiro método o avatar 3D, em alguns casos, iniciava em uma posição alterada, o que refletia em toda a animação de saída. Além disso, como neste caso o processo de mapeamento e cálculos de posições acontece em tempo de execução, a taxa de quadros por segundo caiu em alguns testes, cortando aproximadamente um *frame* para uma sequência de quarenta.

Ja no segundo caso, a maior parte do processamento é feita durante o mapeamento dos elementos no *Blender*, sendo que o processamento de cada bloco, no processo de síntese, é feito por chamadas de blocos de frames pré-renderizados, o que garante, além de uma confiabilidade maior na saída, uma carga de processamento muito menor.

Esta diferença nos dois métodos se mostra valiosa no futuro, onde o sistema tem que calcular não só um sinal, ou parte dele, mas uma conversação inteira.

Concluindo, quanto ao tempo de execução e desempenho, os testes necessários para

validar todo o processo não exigiram alta capacidade de processamento nem renderizações complexas, portanto testes mais específicos podem ser interessantes no futuro. Quanto à validação do processo, o sistema faz o que se propõe, atuando desde a extração do VPC a partir dos dados incluídos na entrada, interpretação dos dados e síntese através de uma avatar 3D desenvolvido para este projeto. Para todos os testes foram desenvolvidas soluções para que se pudesse ter uma análise abrangente do sistema resultante.

O próximo capítulo apresenta as conclusões finais, contibuições, restrições e sugestões para trabalhos futuros.

CAPÍTULO 6

CONCLUSAO

Este capítulo apresenta as restrições e contribuições deste trabalho e mostra propostas de trabalhos futuros.

A proposta deste trabalho é a construção de um avatar 3D animado para o uso na representação da Língua de Sinais Brasileira, que utilize os parâmetros de um modelo formal descritivo (VPC, definido no capítulo 1) para gerar movimentos automaticamente através de chamadas externas (seguindo os parâmetros do modelo formal).

Como resultados podem ser destacados a escolha, estudo e adaptação de um modelo formal de representação computacional da língua de sinais brasileira, baseado em parâmetros descritivos em lugar de sinais indivisíveis, que serviu de base para o desenvolvimento da estrutura de animação aplicada no avatar 3D, no caso, o modelo descritivo IHC-SL.

Além disso foi apresentado o estudo de técnicas para criação de um avatar 3D, gerando uma malha 3D com poucos polígonos (2.276 ao total), com estrutura de animação organizada e que atende ao modelo formal descrito e seus parâmetros (uma vez que todos os sinais descritos no modelo podem ser representados pela estrutura de animação do modelo, levando-se em consideração que alguns parâmetros como espressões faciais e sinais compostos não são detalhados em [2]).

A contribuição principal deste trabalho foi o desenvolvimento de um sistema de síntese através de um avatar 3D para o uso em Libras, de forma automática, baseado em um modelo formal computacional paramétrico. Com trabalhos futuros, este sistema pode ser utilizado pela comunidade surda de fato, já que trabalha com elementos de VPC, e não com sinais prontos ou soletração.

Como restrição neste momento pode ser citado especialmente o processo de mapeamento das entradas para frames de animação, sendo que para este trabalho foram mapeados 3 elementos para a fase de testes. O avatar deverá ter, no futuro, a estrutura de animação

facial construida, além de melhorias no *rigging* de animação.

Como sugestão de trabalhos futuros, deve ser citado principalmente o mapeamento para o Avatar 3D de todos os elementos e sub-elementos do modelo formal. Com o *rigging* construído para este trabalho ja é possível mapear a maioria dos elementos, dependendo apenas no tempo e esforço repetitivo, mas sem alterar as técnicas utilizadas.

As animações referentes às expressões não-manuais, dependem de um *rigging* facial para o avatar, sendo que existem técnicas específicas para este tipo de animação como o modificador *morph*, que possibilita guardar uma sequencia de alterações faciais, ou o uso de *dummies* (controladores que podem ser utilizados para afetar a malha, causando deformações) específicos para este fim.

Além disso interessante tornar mais natural os movimentos, e mais precisos nas áreas em que os *bones* afetam a malha.

No primeiro caso, a solução é aumentar a taxa de *frames* para cada nó do modelo formal, possibilitando assim criar animações mais completas e realistas, não obstante, são necessários ajustes nas deformações causadas na malha pelos *bones* utilizando técnicas como o *vertice painting* (presente no *Blender* e em outros *softwares* de edição 3D como o *3D Studio Max*) onde é possível definir quanto cada vértice do modelo 3D será afetado por determinado *bone*.

Ainda, é interessante considerar a utilização de malhas mais complexas, para sistemas onde a capacidade computacional seja maior e, portanto, modelos de avatares 3D mais realistas e detalhados possam ser usados.

Vale apontar que é fundamental, no futuro, testar o sistema com entradas mais complexas, que abracem uma cadeia completa de VPC, percorrendo a árvore de entradas completa referente a um sinal, e posteriormente, frases, a fim de aplicar o sistema de síntese em um cenário de conversação real, onde o avatar 3D poderá simular, ou representar, textos completos.

Por fim, uma sugestão é aplicar o sistema de síntese apresentada neste trabalho para executar em navegadores e aplicativos móveis, tornando assim, seu uso cada vez mais acessível e abrangente.

Pode-se concluir que os objetivos deste trabalho foram alcançados. Foi construído o processo completo de síntese e animação de um avatar 3D baseado em uma entrada descritiva que utiliza a estrutura IHC-SL, desde a criação, modelagem e estruturação do Avatar 3D até a saída final referente as animações dos VPCs. Após o desenvolvimento foi validado o funcionamento desde a identificação das informações da entrada XML com os VPCs, até a saída com a animação pelo Avatar 3D, sendo que os resultados foram os esperados satisfazendo assim as condições propostas no trabalho.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] W. M. Amaral. Modelo de transcrição da língua de sinais brasileira voltado a implementação de agentes virtuais sinalizadores. 2008.

[2] Antunes, D.R., Guimaraes, C., Garcia, L.S., Oliveira, L.E.S., e Fernandes, S. A framework to support development of sign language human-computer interaction: Building tools for effective information access and inclusion of the deaf. *Research Challenges in Information Science (RCIS), 2011 Fifth International Conference on*, páginas 1 –12, may de 2011.

[3] Blender. *Blender*. blender.org, 2011.

[4] de 24 de abril de 2002 Brasil, Lei n. 10436. Dispõe sobre a libras - língua brasileira de sinais e de outras providências. *Diário Oficial da República Federativa do Brasil*, 2002.

[5] Catanese, S. A., Ferrara, E., Fiumara, G., e Pagano, F. Rendering of 3d dynamic virtual environments. *DISIO 2011, 21, Barcelona, Spain*, março de 2011.

[6] Chadwick, J. E., Haumann, D. R., e Parent, R. E. Layered construction for deformable animated characters. *Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '89, páginas 243–252, New York, NY, USA, 1989. ACM.

[7] CMarkup, 2012. Disponivel em firstobject.com. Acessado em Agosto de 2012.

[8] de Oliveira Neto, F., Fechine, J., e Pereira, R. Matraca: a tool to provide support for people with impaired vision when using the computer for simple tasks. *Proceedings of the 2009 ACM symposium on Applied Computing*, SAC '09, páginas 158–159, New York, NY, USA, 2009. ACM.

[9] Newton Game Dynamics, 2012. Disponivel em newtondynamics.com. Acessado em Julho de 2012.

[10] Open Dynamics Engine, 2012. Disponivel em ode.org. Acessado em Julho de 2012.

[11] Geitz, S., Hanson, T., e Maher, S. Computer generated 3-dimensional models of manual alphabet handshapes for the world wide web. *Proceedings of the second annual ACM conference on Assistive technologies*, Assets '96, páginas 27–31, New York, NY, USA, 1996. ACM.

[12] S. Gibet. Synthesis of sign language gestures. *Conference companion on Human factors in computing systems*, CHI '94, páginas 311–312, New York, NY, USA, 1994. ACM.

[13] GraphViz, 2011. Disponivel em GraphViz.org. Acessado em agosto de 2011.

[14] Hrúz, M., Campr, P., Krňoul, Z., Železný, M., Aran, O., e Santemiz, P. Multi-modal dialogue system with sign language capabilities. *The proceedings of the 13th international ACM SIGACCESS conference on Computers and accessibility*, ASSETS '11, páginas 265–266, New York, NY, USA, 2011. ACM.

[15] IBGE. *IBGE*. ibge.gov.br, 2012.

[16] Intervox. *DosVox*. intervox.nce.ufrj.br/dosvox, 2011.

[17] B. Johnson. Using ogre as a means of teaching c++ programming. *49th ACM Southeast Conference .*, março de 2011.

[18] Kaneko, H., Hamaguchi, N., Doke, M., e Inoue, S. Sign language animation using tvml. *Proceedings of the 9th ACM SIGGRAPH Conference on Virtual-Reality Continuum and its Applications in Industry*, VRCAI '10, páginas 289–292, New York, NY, USA, 2010. ACM.

[19] Kilma, E. e Bellugi, U. The signs of language. *Cambridge, MA: Harvard University*, 1979.

[20] Kipp, M., Nguyen, Q., Heloir, A., e Matthes, S. Assessing the deaf user perspective on sign language avatars. *The proceedings of the 13th international ACM SIGACCESS conference on Computers and accessibility*, ASSETS '11, páginas 107–114, New York, NY, USA, 2011. ACM.

[21] Kitaguchi, K. e Saito, M. A new method for low polygonal modeling of colored object. *Computer Aided Design and Computer Graphics, 2005. Ninth International Conference on*, páginas 6 pp., dec. de 2005.

[22] Kot, B., Wuensche, B., Grundy, J., e Hosking, J. Information visualisation utilising 3d computer game engines case study: A source code comprehension tool. *Department of Computer Science The University of Auckland.*, Julho de 2005.

[23] Zdeněk Krňoul. Web-based sign language synthesis and animation for on-line assistive technologies. *The proceedings of the 13th international ACM SIGACCESS conference on Computers and accessibility*, ASSETS '11, páginas 307–308, New York, NY, USA, 2011. ACM.

[24] Loomis, J., Poizner, H., Bellugi, U., Blakemore, A., e Hollerbach, J. Computer graphic modeling of american sign language. *SIGGRAPH Comput. Graph.*, 17:105–114, July de 1983.

[25] P. Mitchell. Theory of syntactic recognition for natural languages. *Cambridge: MIT Press, 335*, 1980.

[26] Moya, S., Grau, S., Tost, D., Campeny, R., e Ruiz, M. Animation of 3d avatars for rehabilitation of the upper limbs. *Games and Virtual Worlds for Serious Applications (VS-GAMES), 2011 Third International Conference on*, páginas 168 –171, may de 2011.

[27] K. L. Murdock. *3ds Max 2012 Bible*. Wiley; 1 edition, 2011.

[28] NHK, 2011. Disponivel em nhk.or.jp/strl/tvml. Acessado em setembro de 2011.

[29] A. Oliveira. *Estudo Dirigido De 3ds Max 2011*. Erica, 2011.

[30] Peres, S. M., Flores,F.C, Veronez, D., e Olguin, C.J.M. Libras signals recognition: a study with learning vector quantization and bit signature. *Proc. Ninth Brazilian Symp. Neural Networks SBRN 06*, páginas 119–124, 2006.

[31] Quadros, R. M. e Karnopp, L. B. Língua de sinais brasileira: Estudos linguísticos. *Artmed: Porto Alegre*, 2004.

[32] J. F. Reinicke. *Modelando Personagens com o Blender 3D*. NOVATEC, 2008.

[33] Sagawa, H. e Takeuchi, M. A teaching system of japanese sign language using sign language recognition and generation. *Proceedings of the tenth ACM international conference on Multimedia*, MULTIMEDIA '02, páginas 137–145, New York, NY, USA, 2002. ACM.

[34] signwriting. *signwriting*. signwriting.org, 2011.

[35] C. Skliar. *A Surdez: Um Olhar Sobre a Diferença*. Editora Mediação, 1996.

[36] Panda Software, 2012. Disponivel em andytather.co.uk. Acessado em Julho de 2012.

[37] W. C. Stokoe. Sign language structure. *Silver Spring: Linstock Press.Silver Spring*, 1960/1978.

[38] Einfach Teilhaben. *Max*. einfach-teilhaben.de, 2012.

[39] VCom3D. *Communicator Gesture Builder*. vcom3d.com, 2012.

[40] VCom3D. *Sign 4 Me*. signingapp.com, 2012.

[41] VCom3D. *Sign Smith Studio*. vcom3d.com, 2012.

[42] A. N. Xavier. Descricao fonetico-fonologica dos sinais da lingua de sinais brasileira (libras). *FFLCH: USP*, 2006.

[43] Xerces, 2012. Disponivel em xerces.apache.org. Acessado em Junho de 2012.

[44] Xie,Z., Xu,M., Zhenxiang, C., e Qiao,N. The design and implementation of three-dimensional virtual experiment system based on graphics engine. *201O 2nd International Conference on Education Technology and Computer (ICETC)*, 2010.

[45] Tiny XML, 2012. Disponivel em grinninglizard.com/tinyxml. Acessado em maio de 2012.

[46] Yi, B., Harris, F., e Dascalu, S. From creating virtual gestures to "writing"in sign languages. *CHI '05 extended abstracts on Human factors in computing systems*, CHI EA '05, páginas 1885–1888, New York, NY, USA, 2005. ACM.

[47] Yijun, L., Wenbin, C., e Xiaoman, H. 3d graphics engine technology research and implementation. *Computer Science College,Southwest Petroleum University*, 2010.

[48] Zanella, M. K. e SACCOL, D. B. Representação grafica de documentos xml. *ERBD - Anais da Escola Regional de Banco de Dados*, 2012.

[49] ZBrush, 2012. Disponivel em pixologic.com. Acessado em Setembro de 2012.

[50] Zhenfeng, H. e Zhao, J. Development of a low-cost high-fidelity simulator for search and rescue robot. *World Congress on Software Engineering*, Harbin Institute of Technology - Harbin, Heilongjiang Province, China de 2003.

ANEXO 1

Este anexo tem o objetivo de apresentar os parâmetros utilizados e mapeados nos testes e mostrar como funciona o mapeamento de novos parâmetros para inclusão de novos sinais.

Antes, é importante reforçar que a estrutura de animação apresentada no Capítulo 3 suporta todos os parâmetros do modelo formal apresentados no capítulo 4, com exceção das expressões não-manuais, sendo que, para as expressões não manuais, é necessário desenvolver o Rigging Facial do modelo.

Os parâmetros de configuração mapeados para os testes da segunda seção do capítulo 4 e as validações do capítulo 5 foram:

Sinal- Parâmetro inicial para todos os VPCs, não possui valor próprio indicando apenas o início da sequência;

Movimento - Parâmetro onde foi mapeado uma rotação de 45 graus no bone referente ao antebraço;

ConfigMao - Parâmetro onde foi mapeado uma rotação simples dos bones dos dedos para que a mão assumisse o formato da configuração de mão 41;

Simetria - Parâmetro que define se um parâmetro ou um conjunto de parâmetro do braço ou mão dominantes vão ser repetidos pelo braço e mão não-dominates, em execução simétrica;

RotacaoAntebraco - Parâmetro que define a rotação do bone do antebraço em 90 graus, paralelo ao tronco;

ConfigMao - Parâmetro semelhante à configuração de mão do capítulo anterior, sendo que neste mapeamento os bones da mão foram rotacionados para assumir a configuração de mão 08;

RotacaoPulso - Parâmetro onde foi mapeado uma rotação de 90 graus do pulso;

Repeticao - Parâmetro que define a existência de repetição de um parâmetro ou cadeia de parâmetros, e quantas vezes a repetição será executada;

RotacaoBraco - Parâmetro que define uma rotação de 45 graus no bone referente ao

braço;

É importante ressaltar que neste momento os parâmetros mapeados não são o ponto mais importantes, destacando-se o processo de síntese em si.

Para se mapear um novo parâmetro, baseado no método escolhido para os testes de validação, é necessário identificar quais bones serão utilizados no parâmetro, por exemplo, se a intenção é mapear um movimento onde o antebraço se posiciona na horizontal e encostado no tronco, os bones afetados serão os do braço e antebraço.

Então, o primeiro passo do mapeamento é feito no próprio Blender, onde as modificações nas posições dos bones são indicadas em posições X, Y e Z. Para exemplificar melhor todo o processo, assumindo que o parâmetro que se deseja mapear seja a configuração de mão 55. Como a mão do Avatar 3D tem a posição inicial com a palma e os dedos abertos, é necessário alterar os dois bones referentes ao dedão da mão. Então é definido um número de frames para que o movimento fique realista, neste caso 10 frames, e no processo são alterados manualmente através da viewport do Blender, que é a área de trabalho onde se rotaciona os bones livremente, ou por coordenadas X, Y e Z inserindo um valor de rotação para cada bone até que eles assumam a posição definida.

Este movimento dos bones pode ser registrado como uma sequência definida, no caso deste exemplo, a cadeia "Configuração de Mão 55". Em arquivos de objetos 3D com extensão como .b3D ou .md2 esses mapeamentos são exportados junto a malha, com as cadeias nomeadas e mapeadas eliminando o custo computacional de controle de bones em tempo real.

Sendo assim, a próxima e última etapa é o mapeamento no sistema para identificar como o processo vai reagir ao encontrar o valor equivalente a entrada, no caso, o que acontece quando a expressão "Configuração de Mão 55"é encontrada no arquivo XML de entrada.

Para isso são utilizados métodos da própria engine, como segue:

```
// Definição do arquivo de entrada XML
xml.Load(MCD_T("exemplo_vpc.xml"));
```

```
// Inicia o reconhecimento do bloco hierárquico do VPC
xml.FindElemMCD(MCD_T("SINAL"));

// Busca o bloco interno ao elemento pai
xml.IntoElem("Config_Mao");

// Busca o valor relacionado ao parâmetro
while ( xml.FindElem() = nextParam )
{
        xml.FindChildElem( "Configuracao_Mao" );
if ( xml.GetChildData() == std::string("55") )
        {
        if(node->getStartFrame() != 0)
        {
           node->setFrameLoop(Configuracao_mao_55());
        }
// Identifica o valor e executa a
// cadeia de frames mapeada no objeto
}
// Incrementa o contador para o
// processo executar o próximo parâmetro
        nextParam = next +1;
}
```

Como o parâmetro já foi mapeado no Blender, e exportado em conjunto a malha do objeto, o sistema poderá executar a cadeia de frames utilizando o método de chamada descrito acima. Este trecho de código faz ainda a validação do bloco hierárquico do VPC, fazendo com que o parâmetro de configuração seja executado no ordem correta.

Este sistema serve para o mapeamento de qualquer novo parãmetro no sistema, e

mesmo as expressões faciais, quando feito o rigging, utilizam os mesmos conceitos. Os objetos 3D, arquivos de textura, e outros utilizados neste trabalho encontram-se na página: web.inf.ufpr.br/vri.

DIEGO ADDAN GONÇALVES

AVATAR 3D PARA SÍNTESE AUTOMÁTICA DE SINAIS DA LÍNGUA DE SINAIS BRASILEIRA

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Eduardo Todt

CURITIBA

2012