Instituto de Matemática e Estatística - Universidade de São Paulo

MAC 499 – Trabalho de Formatura Supervisionado

Sistema de Animação Facial Tridimensional e Síntese de Voz

Aluna: Andréa Britto Mattos
Orientador: Roberto Marcondes Cesar Jr.

1 de Dezembro de 2008

Conteúdo

Lista de Figuras

1 Introdução

1.1 Animação Facial

O estudo sobre animação facial pelo computador teve início na década de 70, através do trabalho de Frederic Parke[1], que criou, em 1972, a primeira animação em uma face tridimensional. Nessa época, o modelo utilizado por Parke era bastante simplificado. Atualmente, graças aos avanços em computação gráfica e equipamento de hardware, é possível obter modelos com um nível de complexidade e realismo muito superior.

Figura 1: O modelo de face pioneiro utilizado por Parke

Hoje, a animação facial possui aplicações em diversas áreas, como, por exemplo, em filmes, em jogos de computador, na medicina e na criação de agentes sociais e avatares[2]. Esta última inspirou, em 2006, o ínicio do projeto *Avator*.

1.2 O Avator

O projeto começou a ser desenvolvido pelo aluno Marcos Paulo Moreti[13] e consistia de um programa no qual o usuário poderia interagir com uma face tridimensional, fornecendo perguntas através do teclado e recebendo as respostas respectivas, que eram pronunciadas pela face. O objetivo do trabalho, portanto, era a criação de um avatar que funcionasse como uma espécie de ator, simulando, da forma mais fiel possível, um ser humano real.

Mantendo este mesmo objetivo, a aluna Flávia Ost[14] deu continuidade ao projeto em 2007, permitindo que a face pudesse exibir expressões faciais e obter informações do ambiente através de uma webcam. Neste caso último caso, o avatar apenas respondia as perguntas do usuário quando detectava algum movimento em frente a câmera, caso contrário, ele perguntava onde o usuário estava. O algoritmo, embora fosse simples, representava os primeiros passos do projeto na área de Visão Computacional.

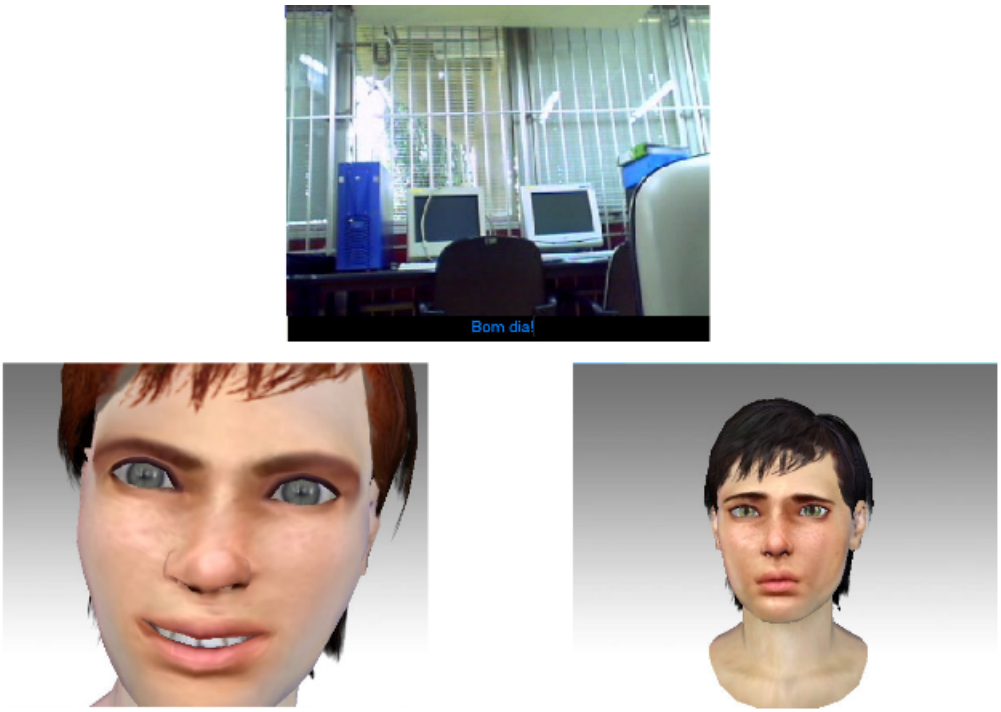


Figura 2: Screenshot da versão de 2007 do Avator

1.2.1 Funcionamento interno

O Avator foi implementado em C++ e, para realizar as tarefas mencionadas, contava com os seguintes programas funcionando em conjunto:

- **Haptek Player**[15]: responsável pelo carregamento e animação do modelo, síntese de voz e sincronização labial.
- **J-Alice**[16]: encarregado de tratar a Inteligência Artificial do avatar, fornecendo as respostas adequadas à cada pergunta. O J-Alice é uma implementação do dialeto AIML[17] (*Artificial Intelligence Markup Language*), que será descrito mais adiante.

Por fim, para implementar o processamento de vídeo, foi utilizada a biblioteca **OpenCV**[18], desenvolvida pela Intel.

Inicialmente, o plano era usar um sistema de animação próprio, implementado pelo Marcos. No entanto, como este projeto não pôde ser terminado, o Haptek Player ficou encarregado de cumprir a tarefa.

1.2.2 Motivação para as mudanças

O programa funcionava bem e cumpria com o seu objetivo, dado que a face possuía animação e fala bastante realistas. Na busca por tornar o Avator cada vez mais próximo de uma pessoa real, haviam muitas funcionalidades que poderiam ser adicionadas ao projeto, em áreas como, por exemplo, Visão Computacional, Inteligência Artificial e Reconhecimento de Áudio.

No entanto, o sistema em si também necessitava de grandes contribuições, em termos de modularização, documentação, e sobretudo, eliminando algumas dependências que tornavam o funcionamento do programa dependente de um ambiente muito restrito. Este último fator era causado pela utilização do Haptek, que tornava o sistema dependende da plataforma Windows. O problema mais grave era ainda com relação a IDE que deveria ser usada: por depender do conjunto de bibliotecas do MFC (*Microsoft Foundation Classes*), o Haptek requeria que o sistema fosse desenvolvido no Visual Studio, no entanto, o MFC não está disponível na versão gratuita da IDE.

Portanto, o software não atendia a duas exigências importantes: além de não ser multi-plataforma, dependia de uma IDE paga para o seu desenvolvimento. Além disso, a maior parte do código tratava a integração do Haptek com uma aplicação do MFC, tendo sido gerada automaticamente e sendo bastante ilegível. Temendo que esses problemas desestimulassem as pessoas a desenvolverem e utilizarem o programa, foi decidido que este trabalho iria priorizar a melhoria do sistema, antes de implementar novas funcionalidades.

1.3 Proposta

Para eliminar as restrições para rodar o Avator, seria necessário substituir o Haptek por um software multi-plataforma e gratuito, que desempenhasse o mesmo papel. Além disso, o programa deveria permitir a síntese de voz em português, idioma falado pelo avatar. Como essa era uma tarefa bastante específica, não foi encontrado nenhum programa que tivesse todos os requisitos necessários.

A solução proposta por esse projeto, então, é a implementação de um módulo próprio de animação facial e síntese de voz, como no plano inicial do Marcos. A idéia é que esse módulo possa ser usado não somente pelo Avator, mas também por outros programas que envolvam Computação Gráfica e Visão Computacional.

O objetivo do módulo é cuidar das mesmas tarefas que o Haptek, permitindo uma animação facial realista e cuidando para que o áudio esteja sincronizado com o movimento dos lábios da face. Além disso, o sistema propõe também um controle maior dos olhos e da face e a geração de mais expressões faciais.

2 Conceitos

2.1 Técnicas de Animação Facial

Existem diversas abordagens para criar uma animação bidimensional ou tridimensional através do computador. No caso particular da animação facial, estes métodos podem ser divididos em dois grupos[3].

Uma animação facial de *baixo-nível* utiliza métodos para modificar a aparência ou a estrutura geométrica da face ao longo do tempo, baseando-se em um conjunto de paramêtros.

Já a animação de *alto-nível* utiliza os parâmetros fornecidos por alguma das técnicas de baixo nível para produzir uma sequência de animação. Esta sequência pode ser obtida, por exemplo, por meio de scripts, comandos diretos (edição manual), ou copiando movimentos de atores reais.

2.2 Animação de Baixo-Nível

Existem três abordagens mais comuns para a animação de baixo nível[4]. Cada uma destas técnicas será brevemente explicada nos próximos itens.

2.2.1 Modelo Baseado em Músculos

Essa técnica procura simular a geometria dos ossos, músculos e tecidos da face[5]. Neste método, os vértices do modelo não são alterados diretamente, e sim, devem acompanhar esta estrutura interna, à medida em que ela é modificada.

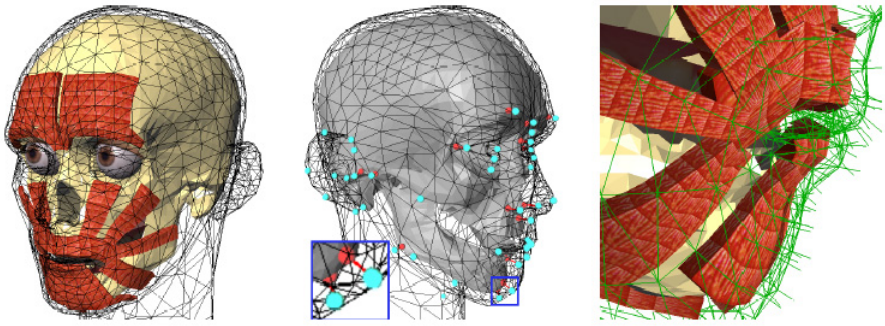


Figura 3: Esquema de animação facial pelo modelo baseado em músculos

2.2.2 Modelo de Face Paramétrica

Nessa técnica, são definidos pontos característicos na face e esse pontos são deslocados conforme valores fornecidos manualmente.

O primeiro padrão internacional para a animação de faces paramétricas é o do MPEG-4[26]. Embora seja conhecido principalmente pela sua relação com a compressão de vídeo, o MPEG-4 é um padrão para a codificação de objetos audiovisuais 2D e 3D em uma cena.

No modelo proposto pelo MPEG-4, são definidos 84 pontos chaves na face, mostrados na imagem a seguir.

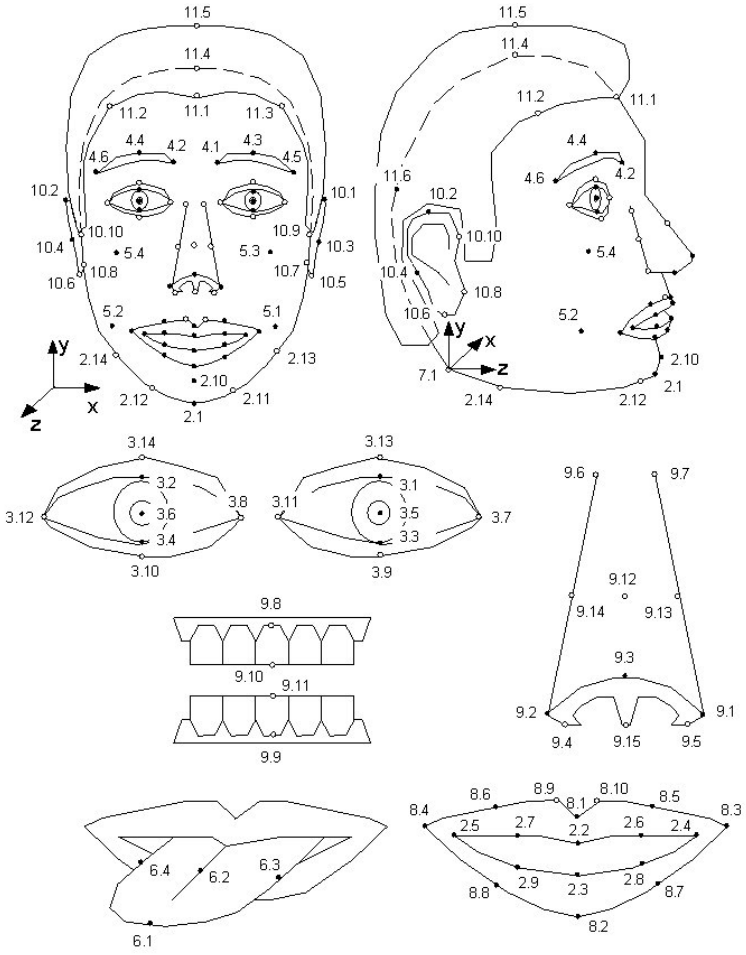


Figura 4: Pontos característicos no padrão MPEG-4

2.2.3 Morph Target

Dentre as técnicas de baixo nível, esta é a abordagem mais simples, no entanto é comumente usada. Nesta abordagem - também conhecida como animação *por vértices* - os vértices da face são modificados de acordo com poses pré-definidas em quadros chaves.

Para obter uma transição suave entre os quadros, utiliza-se a técnica de **interpolação**: dados dois quadros, o computador encarrega-se de encontrar posições intermediárias - denominadas *in-betweens* - calculando a distância entre dois pontos correspondentes, de forma linear ou não linear[6].

Devido à sua complexidade com relação às demais técnicas de baixo nível, essa foi a técnica de animação utlizada neste projeto.

2.3 Animação de Alto-Nível

Dentre os diversos métodos para realizar animações de alto nível, tem sido estudadas muitas técnicas para a recuperação de movimentos faciais a partir de vídeos, como é descrito a seguir.

2.3.1 Motion Capture

Na animação por captura de movimento (*Motion Capture*), a movimentação do modelo acompanha a ação de humanos, o que pode ser feito em tempo real.

Figura 5: Exemplos de Motion Capture aplicados à animação facial, utilizando modelos altamente realistas[19][20]

Um dos desafios do processo é mapear, de maneira adequada, a face real para a face sintética quando as proporções entre as duas é muito distinta.

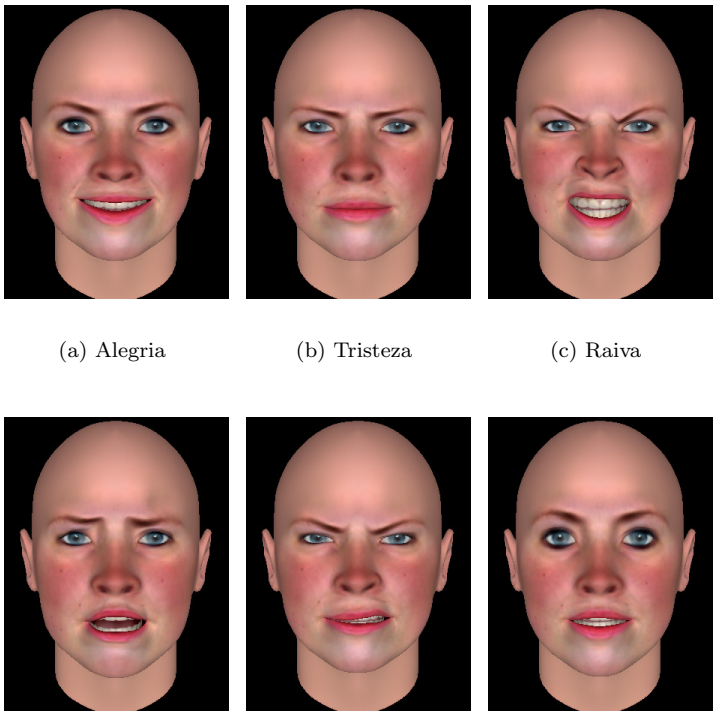
Normalmente, é necessário utilizar marcadores ou equipamentos especiais para efetuar o mapeamento. Atualmente, no entanto, existem sistemas que dispensam o uso destes instrumentos, aliando as idéias do Motion Capture com as do monitoramento baseado em Visão Computacional[7].

2.4 Poses chaves

Como foi dito, este projeto utilizou a técnica de Morph Target, na qual existe um conjunto fixo de poses chaves e as posições intermediárias são interpoladas pelo computador. Essas poses são representadas por dois grupos: as **expressões faciais** e os **visemas**.

2.4.1 Expressões faciais

Não existe um modelo único que defina o número de expressões faciais que uma face consegue exibir. Segundo o psicólogo Paul Ekman, existem seis expressões universais, que, quando combinadas, podem gerar todas as outras[8]. São elas: *alegria*, *tristeza*, *raiva*, *medo*, *nojo* e *surpresa*.



(a) Alegria (b) Tristeza (c) Raiva

(d) Medo (e) Nojo (f) Supresa

Figura 6: As seis expressões faciais propostas por Ekman

Ekman foi o pioneiro no estudo de expressões faciais, no entanto, existem outros modelos. Para Plutchik, por exemplo, as expressões de *antecipação* e

confiança também podem ser consideradas universais, além das seis já citadas[9]. Para este projeto, entretanto, foi utilizado o modelo de Ekman.

2.4.2 Visemas

Um visema é definido como o correspondente visual de um fonema, isto é, o formato da boca quando a face reproduz algum fonema[10]. Um visema pode corresponder a mais de um fonema.

Figura 7: Os fonemas /f/ e /v/ - como em **f**aca e **v**aca - correspondem ao mesmo visema (à esquerda). Os fonemas /j/ e /ʃ/ - como em **j**ato e **ch**ato - também estão representados pelo mesmo visema (à direita)

Não existe um consenso com relação ao número de visemas necessários para uma animação realista. Na animação dos movimentos da boca, durante a pronúncia de uma frase, dois aspectos devem ser levados em consideração: o visema exibido no momento é influenciado pelo visema anterior e pela expressão facial corrente.

Figura 8: O visema para o fonema /o/ na face neutra (à esquerda) e alterado pelas expressões de alegria, nojo e surpresa

3 Ferramentas

3.1 Requisitos

Uma animação facial realista, com síntese de voz, é uma tarefa que envolve três desafios:

1. Uma representação fiel da face, possuindo traços suaves e uma textura de pele natural, com manchas e imperfeições. Uma alternativa para esta tarefa são os métodos de geração de modelos a partir de fotos[11].

2. A modelagem da animação em si. Para alcançar um resultado fiel, é preciso levar em conta a complexidade da interação entre os músculos do rosto, o comportamento emocional do indivíduo e os movimentos involuntários da face.

3. Para a síntese de voz, é preciso que o áudio e os movimentos dos lábios estejam sincronizados. Os visemas exibidos na fala devem estar de acordo com os fonemas pronunciados no momento.

Portanto, para atender à primeira exigência, o projeto necessitava de um programa para gerar faces tridimensionais realistas, com as poses chaves mencionadas na seção anterior.

Para atender ao segundo requisito, era necessário uma ferramenta para interpolar estas poses, gerando uma animação suave.

Por fim, era preciso um programa para executar a síntese de voz. Estas três ferramentas serão descritas, respectivamente, a seguir.

3.2 FaceGen

Como este projeto não é focado na modelagem de faces tridimensionais, era necessário um programa específico para desempenhar esta tarefa.

Na versão anterior do Avator, as faces foram geradas pelo programa PeoplePutty, da Haptek. No entanto, estas faces poderiam ser exportadas apenas para o formato compatível com o Haptek Player.

Neste projeto, o FaceGen[21] foi o programa de modelagem escolhido, pelo fato de atender a uma série de requisitos fundamentais:

1. Facilidade para gerar modelos bastante realistas - podendo ser criados, também, a partir de fotos.

2. Geração das seis expressões faciais propostas por Ekman.

3. Geração de visemas.

4. Geração de movimentos como piscar, levantar as sobrancelhas, etc.

5. Possibilidade de exportar o resultado para formatos conhecidos (wavefront e 3ds).

6. Possibilidade de exportar os arquivos de texturas (em formato jpeg).

A face exibida anteriormente nas Figuras 6, 7 e 8, com diferentes expressões faciais e visemas, foi gerada pelo FaceGen.

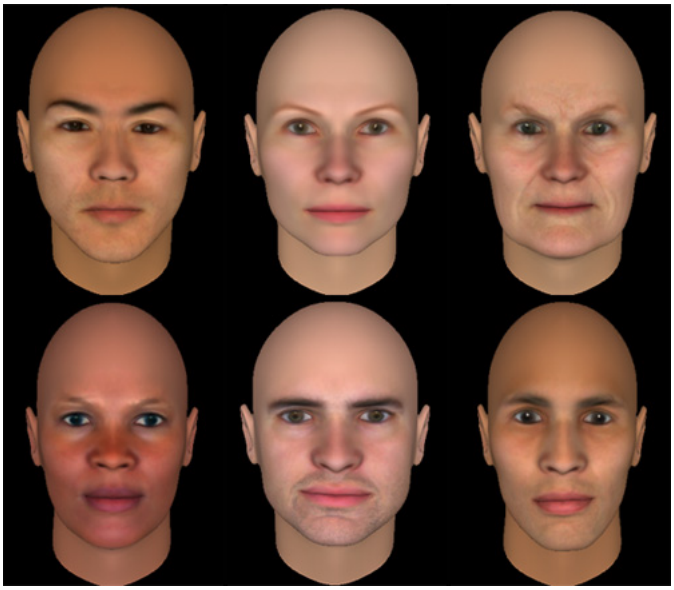


Figura 9: Exemplos de faces criadas pelo FaceGen, de diferentes idades e raças

Embora seja um software pago, assim como o PeoplePutty, o papel do FaceGen foi apenas criar as faces e exportá-las para que pudessem ser usadas pelo sistema de animação (respeitando as restrições de redistribuição destes modelos). Ou seja, não é necessário que o usuário possua o FaceGen para rodar o módulo de animação, apenas os modelos.

Algumas opções de programas gratuitos também foram consideradas, como o MakeHuman[22] e o Daz 3D[23], mas essas opções tiveram que ser descartadas por não atenderem à maioria dos requisitos citados anteriormente.

3.3 Ogre 3D

Uma vez tendo o modelo das faces para as expressões faciais e os visemas, seria preciso interpolar essas poses para obter uma animação natural (como mostra a Figura 10). Para cuidar deste papel, foi utilizada a engine gráfica Ogre3D[24].

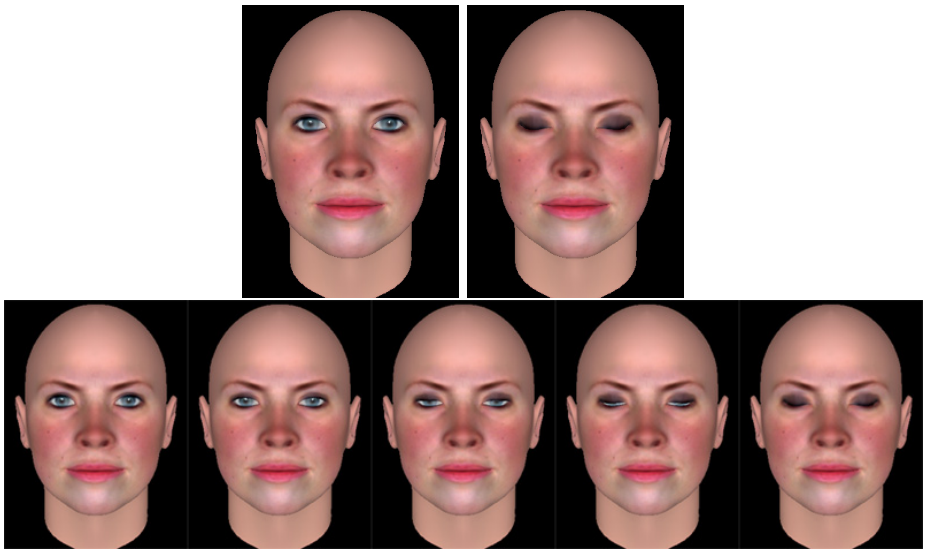


Figura 10: Dados os quadros-chaves de origem e destino (face neutra e face de olhos fechados) o Ogre é capaz de fazer a interpolação destes quadros, encontrando os in-betweens que tornam esta transição suave

Além de ser gratuito e multi-plataforma, o motor foi escolhido por possuir suporte específico para animação facial. O Ogre suporta um tipo de animação por vértices denominada de *Pose Animation*, que é capaz de combinar múltiplas **poses**, com diferentes **influências**.

Uma *pose* constitui uma série de valores que definem, para cada vértice da malha tridimensional[1], como ela será deformada. É importante enfatizar, portanto, que uma pose *não* contém a posição espacial de cada vértice da face, e sim, valores que representam o deslocamento de cada ponto com relação a malha da face neutra[2].

Por exemplo, na pose que representa a face de olhos fechados, apenas os vértices das pálbebras são deformados. Os demais vértices da face possuem

[1] Uma malha é uma estrutura geométrica composta por um conjunto de vértices, arestas e faces que compõe um modelo.

[2] A face neutra é a face que não é caracterizada por nenhuma expressão facial ou visema.

deslocamento de valor zero para esta pose e, portanto, não são modificados.

Essa abordagem é especialmente útil para o processo de combinar poses: basta aplicar uma deformação sobre a face neutra e, em seguida, aplicar uma nova deformação sobre a face resultante.

A *influência* de uma pose é um valor entre 0 e 1, que diz respeito à quantidade de deformação da pose. Por exemplo, se a pose que representa a expressão de raiva possui influência 0, tem-se apenas a face neutra. Aumentando o valor da influência, é possivel, por exemplo, obter uma face 50% raivosa, ou 100% raivosa (nestes casos, a influência valeria $\frac{1}{2}$ e 1, respectivamente).

(a) 0% (b) 25% (c) 50% (d) 75% (e) 100%

Figura 11: Expressão de raiva com diferentes influências

A possibilidade de combinar poses é especialmente útil em duas situações. Primeiro, para gerar emoções derivadas a partir de emoções puras:

(a) Tristeza (b) Surpresa (c) Preocupação

Figura 12: Combinação de expressões puras gerando a derivada de *Preocupação*

Segundo, pois é necessário que a face possa executar algumas ações ao mesmo tempo, como por exemplo, falar e movimentar os olhos, ou piscar e mudar de expressão:

(a) Ação de olhar para baixo

(b) Ação de falar

(c) Ações combinadas

Figura 13: Combinação de ações que podem ocorrer ao mesmo tempo

A engine tem código aberto e é escrita em C++, mesma linguaguem de desenvolvimento da versão anterior do Avator. O Ogre foi utilizado, também, para a criação de widgets, como a caixa de texto na qual o usuário digita as perguntas para o avatar. Esta tarefa, anteriormente, estava a cargo do MFC.

3.4 eSpeak

Na versão anterior do Avator, o Haptek utilizava a própria API de síntese de voz do Windows, a SAPI (*Speech Application Programming Interface*). Para sintetizar o texto em português, era necessário instalar uma voz que era disponibilizada junto com o Avator.

Nesta versão, essa tarefa ficou a cargo do eSpeak[25]. O eSpeak é um programa de TTS (*Text to Speech*) de código aberto capaz de sintetizar um dado texto. O programa foi escolhido por possuir suporte para diversas linguagens, dentre elas, o português.

3.4.1 Fonemas em ASCII

O **IPA** (*International Phonetic Alphabet*) é uma notação que procura descrever o conjunto de fonemas existentes. Os fonemas do IPA são universais, o que não quer dizer que cada linguagem deve utilizar, necessariamente, todos os fonemas especificados no alfabeto. Em outras palavras, embora a representação dos fonemas seja a mesma, o conjunto de fonemas usado varia de uma linguagem para outra.

Existem alguns sistemas para traduzir os elementos do IPA para ASCII, sendo o SAMPA e o Kirshenbaum[27] os mais conhecidos. O eSpeak utiliza o Kirshenbaum para realizar esta tradução. Para deixar este conceito mais claro, a tabela seguinte mostra exemplos de como ocorre este mapeamento para fonemas do português[12].

Mapeamento de Fonemas no Kirshenbaum		
Fonema no IPA	Fonema em ASCII	Exemplo do som
/a/	a	s**a**co
/e/	e	s**e**co
/ɛ/	E	b**e**la
/i/	i	s**i**co
/o/	o	s**o**co
/ɔ/	O	s**o**zinho
/u/	u	s**u**co
/p/	p	**p**ata
/b/	b	**b**ata
/t/	t	**t**ata
/d/	d	**d**ata
/k/	k	**c**ata
/g/	g	**g**ata
/f/	f	**f**aca
/v/	v	**v**aca
/s/	s	**s**aca
/z/	z	**z**aca
/ʃ/	S	**ch**aga
/ʒ/	Z	**j**aca
/m/	m	**m**ata
/n/	n	**n**ata
/ŋ/	N	ga**nh**o
/l/	l	ga**l**o
/ʎ/	L	ga**lh**o
/ɾ/	R	ca**r**o
/ɣ/	G	ca**rr**o

Tabela 1: Mapeamento de IPA para ASCII de fonemas do português, segundo o modelo Kirshenbaum

O eSpeak utiliza, para o português brasileiro, alguns fonemas a mais, além dos que estão representados na tabela acima. Como foi dito, não existe uma correspondência oficial entre fonemas e visemas para produzir uma animação natural. Os fonemas utilizados no eSpeak tiveram que ser mapeados para os visemas produzidos no FaceGen buscando a melhor aproximação possível. A seção 4.5.1 descreve mais detalhes deste mapeamento.

3.4.2 Definição da pronúncia de um texto

O eSpeak possui, para cada uma das linguagens que suporta, arquivos do tipo **list** e **rule**. O primeiro contém a forma explícita de pronunciar certas palavras da linguagem que não acompanham as regras gerais, como, por exemplo, algumas expressões e nomes estrangeiros comuns. O arquivo define, também, a pronúncia correta para os números, além de especificar artigos, preposições, pronomes, etc.

O arquivo rule define as regras gerais para a pronúncia de todas as outras palavras que não estejam particularmente especificadas no arquivo list. O arquivo determina os marcadores de pausa e acento tônico e, para todas as letras do alfabeto, classifica o fonema correspondente. Esse fonema irá variar de acordo com a posição em que essa letra aparece no texto, e as letras vizinhas.

Por exemplo, a letra *e*, quando precedida de *d* e no final da frase, possui som de /i/. Quando sucedida de *la*, no início da frase, possui som de /ɛ/. Pode ainda possuir som de /e/, quando estiver no início da frase e acompanhada por uma letra *u* e uma ou mais consoantes. Além dessas, uma série de outras regras para a letra *e* são listadas.

O arquivo define uma sintaxe própria para fazer esse mapeamento entre letras e fonemas, utilizando alguns símbolos para definir vogais e consoantes arbitrárias. Por exemplo, na sintaxe do arquivo, a regra

```
o (rCA_    O
```

define que a letra o, quando sucedida de r, uma consoante arbitrária e uma vogal qualquer, possui som de /ɔ/.

4 Atividades realizadas

O trabalho foi dividido em duas partes: a implementação do módulo de animação e síntese de voz, e a integração deste módulo com o Avator, o que demandava uma refatoração total do sistema.

A ênfase do projeto foi dada na primeira parte, que consistia de duas tarefas:

1. Modelar a animação da face para que pudesse exibir diferentes poses
2. Adicionar a síntese de voz, sincronizada com o movimento dos lábios

Na segunda parte, as tarefas foram priorizadas da seguinte maneira:

1. Implementar o módulo de Visão Computacional
2. Implementar a Inteligência Artificial do avatar
3. Permitir a conversa com mais de um avatar ao mesmo tempo

4.1 Pesquisa inicial

Os primeiros dois meses do trabalho foram dedicados ao estudo sobre o Avator e à pesquisa das ferramentas que seriam necessárias para implementar um novo módulo de animação facial para o sistema, bem como alguns testes para verificar a viabilidade dessa tarefa.

A ferramenta mais estudada nesse período foi o Ogre 3D, para entender como funcionavam os programas da engine e como objetos modelados em outros softwares poderiam ser importados para ela.

4.2 Importação dos modelos para o Ogre

O Ogre suporta apenas arquivos binários no formato **mesh**. Existem diversas formas para converter um modelo para mesh, por exemplo, através de plugins para alguns softwares de modelagem (por exemplo, Blender, Wings ou Google SketchUp) ou de programas que podem ser baixados no site da engine. Essa conversão pode ser feita em dois passos:

- O modelo é convertido para o formato **mesh.xml**, um tipo particular de XML
- O modelo de extensão .mesh.xml é convertido para .mesh através do programa *OgreXMLConverter* (que é instalado junto com a engine).

Neste segundo passo, é criado um arquivo *material*, contendo as informações de textura e iluminação do modelo.

4.3 Importação das poses para o Ogre

O primeiro desafio do projeto foi exportar os modelos do FaceGen para o Ogre de forma que eles obedecessem as restrições de animação da engine.

Ao exportar as faces do FaceGen, vários arquivos diferentes eram obtidos, um para cada pose. No entanto, conforme mencionado anteriormente, uma pose no Ogre é representada por deslocamentos na malha original. Em outras palavras, não é possível animar diversas malhas no Ogre como um único modelo.

A primeira solução buscada foi tentar unir esses modelos como um só através de algum software de modelagem e exportar a animação final. No entanto, a maioria desses programas também espera que animações sejam feitas a partir da malha base, não permitindo a fusão vários arquivos diferentes.

A solução para este problema, então, foi implementar um script para modificar diretamente os arquivos mesh.xml.

4.3.1 O formato mesh.xml

Um modelo animado na especificação mesh.xml possui o seguinte formato:

```
<mesh>
   <submeshes>
      <submesh material="nome_do_material">
         <faces count="número_de_faces">
            ...
         </faces>
         <geometry vertexcount="número_de_vértices">
            <vertex>
               ...
            </vertex>
         </geometry>
      </submesh>
         ...
   </submeshes>
   <submeshnames>
      <submeshname name="nome_da_submalha" index="índice_da_submalha" />
         ...
   </submeshnames>
   <poses>
      <pose target="submesh" index="índice_da_submalha" name="nome_da_pose">
         ...
      </pose>
         ...
   </poses>
```

```
   <animations>
      ...
   </animations>
</mesh>
```

Primeiramente, são definidas as submalhas do modelo. Na malha das faces exportadas pelo FaceGen existem oito submalhas: as dos olhos, dos dentes, da língua, da parte interna da boca, da pele e do cabelo. Para cada submalha, são definidas suas faces e vértices. Em seguida, as submalhas são nomeadas e indexadas.

Se existirem poses, é dado um nome para cada uma e definida a submalha que ela irá deformar. No Ogre existem animações automáticas - definidas no final do arquivo - ou manuais. Na animação manual, o programa especifica o momento em que uma dada pose será exibida com uma certa influência.

4.3.2 O script de conversão

A definição de um vértice especifica sua posição nos eixos x, y e z, normais e coordenadas de textura. Por exemplo:

```
<vertex>
   <position x="30.242" y="70.9967" z="29.62" />
   <normal x="0.839101" y="-0.19281" z="0.508659" />
   <texcoord u="0.444124" v="0.673521" />
</vertex>
```

Cada vértice possui um índice, que é dado de acordo com a ordem em que o vértice é definido.

A definição das poses especifica, para cada vértice, o deslocamento que deve ser feito em x, y e z. Por exemplo:

```
<poseoffset index="índice_do_vértice" x="0.138" y="-3.268" z="-2.259" />
```

O script foi feito em Perl e funciona da seguinte forma: primeiro, recebe o arquivo mesh.xml da face neutra e de todas as poses que serão processadas. Os vértices da face neutra são lidos e armazenados. À medida em que os demais arquivos - das poses - são lidos, vai sendo calculada a diferença entre cada vértice do modelo que está sendo processado com o vértice correspondente da face neutra. O valor resultante é o deslocamento que o vértice da face neutra deve sofrer na pose.

Enquanto os arquivos são processados, os resultados vão sendo gravados no arquivo de saída, gerando um documento com o mesmo formato descrito anteriormente, ou seja:

```
<mesh>
   <submeshes>
     ...
   </submeshes>
     ...
   <poses>
     ...
   </poses>
</mesh>
```

onde o conjunto de submalhas correspondem à face neutra, e o conjunto de poses corresponde ao resultado da subtração ponto a ponto.

Um inconveniente para o processo seria o trabalho em exportar todas as poses do FaceGen e convertê-las para o formato mesh.xml, sempre que se desejasse adicionar um novo modelo ao programa.

Antes de descrever a solução para o problema, é preciso destacar um ponto fundamental: todos os modelos do FaceGen possuem o mesmo número de vértices, e portanto, após a conversão de dois modelos, apenas a primeira parte do arquivo resultante (o conjunto de submalhas) irá diferir entre um modelo e outro. O conjunto de poses será exatamente o mesmo, agindo sobre malhas neutras distintas.

Desta forma, foram exportadas, do FaceGen, a face neutra e as faces para todas as poses de um mesmo modelo arbitrário, denominado modelo *base*. Em seguida, todos estes modelos foram convertidos para o formato mesh.xml, servindo como entrada para o script. Assim, para a conversão de um novo modelo, basta exportar a face neutra deste modelo. A primeira parte do arquivo de saída - o conjunto de submalhas - é dado conforme este novo modelo, e a segunda parte - cálculo das poses - é calculada pela subtração entre os arquivos da base.

Por fim, programa encarrega-se também de separar as submalhas do arquivo original em arquivos distintos, para que o sistema de animação possa ter maior controle de cada parte da face - em especial dos olhos - e chamar automaticamente o *OgreXMLConverter* para converter esses arquivos para mesh.

4.3.3 Restrição dos modelos

O método de interpolação não pode ser aplicado se o problema não for bem definido, isto é, deve haver uma correspondência entre os dois modelos que se deseja interpolar. No caso dos modelos do FaceGen, todas as faces possuem o mesmo número de vértices, existindo uma correspondência entre eles, o que torna viável o método descrito, de aplicar a subtração ponto a ponto.

4.4 Implementação do módulo de animação

4.4.1 Obtendo uma transição suave entre as poses

Ao carregar um modelo animado no Ogre, é possível atualizar a influência de uma pose através do método

```
void updatePoseReference (ushort índice_da_pose, Real influência)
```

Para obter uma animação suave, portanto, é preciso chamar esse método repetidamente, incrementando o parâmetro da influência. A velocidade de transição entre uma pose e outra depende do valor do incremento.

Nas aplicações do Ogre o fluxo do programa é definido através de *Frame Listeners*. Um Frame Listener é uma classe que possui métodos para capturar eventos de entrada e definir as ações que ocorrem no início e final de cada frame. Se uma animação for iniciada, é preciso, em cada frame, incrementar o valor da influência e atualizar a pose, até o encerramento da animação.

4.4.2 Rotações

A animação de rotação da face, em um dado eixo, ocorre de maneira análoga: para obter uma animação suave, é preciso definir que a face, em cada frame, rotacione com uma angulação pequena, até alcançar o ângulo final.

A transformação de rotação no Ogre é aplicada sobre objetos do tipo *SceneNode*. No programa, existe um nó para a face, que reúne todos os modelos que a compõe (dentes, olhos, língua, etc). Quando a transformação é aplicada sobre o nó da face, ela é automaticamente transmitida para cada um dos modelos.

4.4.3 Comandos de animação da face

O programa disponibiliza comandos para alterar o estado do avatar. A face é capaz de exibir as seis expressões universais[3], um conjunto de visemas, e algumas ações que compõem a movimentação não-verbal da face.

Alguns comandos são mais complexos, isto é, em certas poses, não há a necessidade de deformar todos os componentes da face. Por exemplo, na expressão de sorrir, dado que na face neutra os dentes estão encostados e a língua está no centro da boca, basta deformar a região da pele. Na expressão de medo, a face deve abrir a boca, e portanto, os dentes de baixo, a língua, e a parte interna da boca são também deformados.

[3]No FaceGen, a expressão de *alegria* é dividida em *Sorriso aberto* e *Sorriso fechado*

4.4.4 Animações involuntárias da face

Uma animação facial realista deve cuidar para que a face não permaneça estática, apresentando alguns movimentos não-verbais. No programa, constantemente, a face exibe as ações de piscar, levantar as sobrancelhas, apertar os olhos e movimentar os cantos da boca[4].

Essas animações são disparadas, e ao final, desfeitas - decrementando o valor da influência. Estes movimentos acontecem em tempos aleatórios: quando a face termina o movimento de piscar, por exemplo, o programa sorteia o próximo momento em que irá iniciar, novamente, a animação de piscar.

Para a face parecer natural, é preciso também que a sua posição varie. Ao longo de todo o programa, a face rotaciona, aleatoriamente, ao redor dos três eixos. Analogamente, os movimentos são feitos e desfeitos, para que a face não vire de costas ou de cabeça para baixo.

4.4.5 Animações comandadas

A face deve poder combinar os movimentos involuntários descritos com expressões faciais. As frases que o avatar deve falar podem possuir marcações, no início ou no final da frase, para exibir ou desfazer uma pose. Se a frase possui a marcação

```
<pose>
```

a face irá exibir - se já não estiver exibindo - esta pose. Já se houver a marcação

```
</pose>
```

a pose - se estiver sendo exibida - será desfeita. É possível haver mais de uma marcação, permitindo a combinação de expressões.

4.5 Implementação do módulo de síntese de voz

A tarefa de sintetizar a voz do avatar ficou a cargo da biblioteca disponibilizada pelo eSpeak. O módulo de áudio é responsável por inicializar o eSpeak, produzir o som e retornar os fonemas que estão sendo enunciados em cada momento.

Além disso, é possível definir o idioma e a velocidade da fala, bem como a voz do avatar, neste caso, como variantes da voz padrão do eSpeak[5].

[4] Nesse caso, é exibida a expressão de *Sorriso fechado*, com uma inlfuência bem pequena

[5] Existem quatro variantes femininas e seis masculinas para a voz padrão.

4.5.1 Mapeamento de fonemas para visemas

Para obter uma animação bem feita é fundamental garantir que, enquanto o áudio esteja sendo produzido, a face esteja executando movimentos de abrir e fechar a boca. Além disso, para conferir maior realismo, os fonemas enunciados devem estar de acordo com os visemas exibidos.

Não existe um mapeamento padrão de fonemas para visemas. Como foi dito, também não há um número correto de visemas para uma animação fiel. O mapeamento adotado foi baseado no mapeamento proposto pelo padrão MPEG-4, para a animação de faces paramétricas.

No entanto, como o MPEG-4 não define a representação de visemas em português, alguns ajustes tiveram que ser feitos. O mapeamento final adotado está exibido na tabela a seguir[6], e os visemas utilizados estão mostrados na Figura 14.

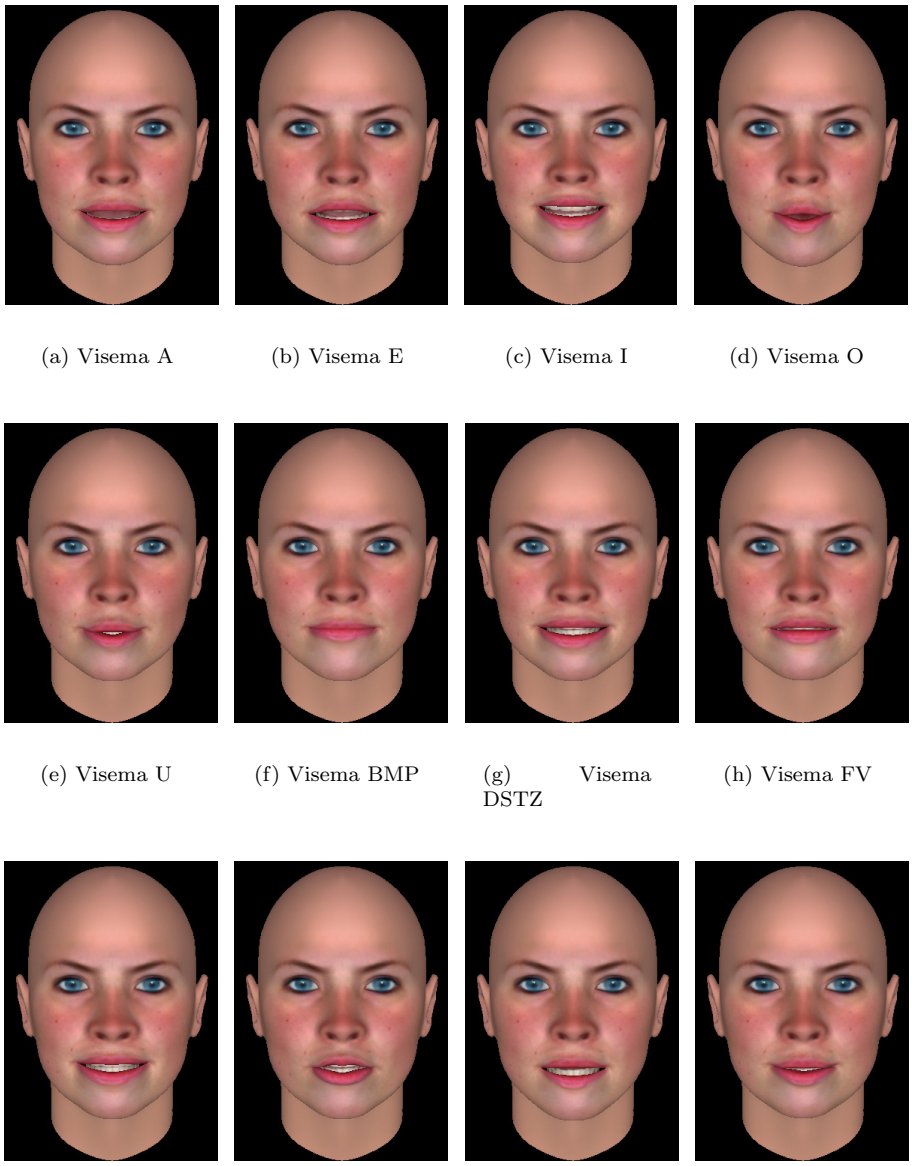
Mapeamento de Fonemas para Visemas	
Fonemas	Visema
/a/, /æ/	Visema A
/e/, /ɛ/	Visema E
/i/,	Visema I
/o/, /ɔ/	Visema O
/u/	Visema U
/b/, /m/, /p/	Visema BMP
/d/, /s/, /t/, /z/	Visema DSTZ
/f/, /v/	Visema FV
/k/, /g/	Visema GK
/ʒ/, /ʃ/	Visema JX
/l/, /n/	Visema LN
/ɣ/, /ɾ/	Visema R

Tabela 2: A tabela mostra como os fonemas foram agrupados e relacionados com visemas

[6]O fonema /æ/, cuja representação em ASCII é &, usualmente não é considerado como parte do conjunto de fonemas do português.

O eSpeak considera também, os fonemas aI, aU, eI, EI, eU, EU, iU, oI, OI, oU, OU, uI - com a sílaba tônica na primeira vogal - que não fazem parte do IPA, sendo considerados como dois fonemas distintos. Nestes casos, a animação foi disparada para os dois fonemas que constam no IPA.

Marcadores de pausa e nasalidade presentes nos fonemas do eSpeak não influenciaram no mapeamento.



(a) Visema A (b) Visema E (c) Visema I (d) Visema O

(e) Visema U (f) Visema BMP (g) Visema DSTZ (h) Visema FV

(i) Visema GK (j) Visema JX (k) Visema LN (l) Visema R

Figura 14: Visemas utilizados no programa

4.5.2 Sincronização Labial

Quando o programa requisita a pronúncia de alguma frase pelo avatar, esta frase é primeiramente tratada, verificando a existência de marcadores para exibir uma expressão facial. Em seguida, é disparada uma *thread* no programa, reproduzindo o áudio da frase que deve ser falada. A utilização de threads é necessária para que o áudio possa ser tocado ao longo de vários frames. Caso contrário, sua reprodução, se ocorresse em um único frame, congelaria a animação e a recepção de eventos até o final da síntese.

Para poder sincronizar os movimentos dos lábios com o áudio é preciso saber a hora exata em que cada fonema é pronunciado. Para isso, foi implementada e registrada uma callback no programa que é chamada automaticamente pelo eSpeak, sempre que um buffer de áudio é produzido.

Essa callback recebe uma lista de eventos, incluíndo eventos de fonema. Esses fonemas são então convertidos para uma *String*, e, conforme o mapeamento da tabela anterior, o visema respectivo é exibido.

Embora as poses sejam exibidas e, ao final, desfeitas, as animações são disparadas de forma independente, conforme os fonemas são pronunciados. Isto é, num dado momento da fala, pode haver a junção de mais de uma pose, de modo a respeitar que o visema atual seja influenciado pelo anterior.

4.6 Refatoração do Avator

Tendo encerrado o módulo de animação, partiu-se para a segunda tarefa do projeto: a refatoração do Avator. Como o programa possuía muitas dependências do MFC, o código não pôde ser aproveitado, tendo que ser refeito a partir do zero.

A primeira tarefa foi adicionar um módulo mais complexo de Visão Computacional ao programa, para então, implementar a Inteligência Artificial através do AIML. Essa ordem de tarefas foi dada para utilizar o módulo de visão para testar, no novo módulo, uma operação que não era trivial no Haptek: o controle do movimento dos olhos da face, como será descrito a seguir.

4.7 Visão Computacional

4.7.1 Primeiros passos

Na versão anterior do Avator, foi adicionado um algoritmo para que o avatar apenas respondesse as perguntas do usuário quando detectava algum tipo de movimento em frente a câmera. A idéia era fazer com que o avatar exigisse a presença de um interlocutor para iniciar a conversa.

O algoritmo usado era simples: inicialmente, era capturada uma imagem do ambiente, que seria a imagem de fundo. Então, a cada momento, a câmera captava novas imagens, que eram comparadas com a imagem de fundo. A comparação era dada pela subtração pixel a pixel entre as duas imagens. Se o valor resultante ultrapassasse um limiar pré-estabelecido, considerava-se a presença de movimento.

Como pode ser observado, o algoritmo não exigia, necessariamente, a presença de um interlocutor humano em frente a câmera. Além disso, caso o usuário ajustasse a posição da câmera, mesmo não estando em frente ao computador, o programa considerava que houve movimento, dado que o fundo foi alterado.

4.7.2 Versão atual

Para resolver os problemas citados, foi utilizada a implementação do próprio OpenCV de reconhecimento facial, estabelecendo que o avatar, de fato, apenas dialogasse com uma pessoa. Além disso, como a face do usuário poderia ser localizada na tela, foi possível que outra funcionalidade fosse adicionada: mover os olhos do avatar de acordo com a posição do interlocutor.

4.7.3 Detecção de Faces

O reconhecimento de objetos no OpenCV utiliza uma série de classificadores denominados *Haar-like features*[7], que consideram regiões retangulares da imagem.

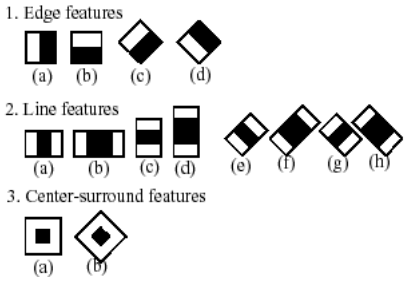


Figura 15: Classificadores usados pelo OpenCV

Primeiro, existe um processo de treinamento dos classificadores, no qual são consideradas imagens que possuem o objeto a ser detectado - chamadas de amostras positivas - e imagens arbitrárias de mesma dimensão, que não possuem o objeto em questão - chamadas de amostras negativas.

[7]Este nome é dado graças à semelhança com a transformada de Haar

Durante o processo de reconhecimento, os pixels das regiões de interesse são somados e o valor resultante permite que a imagem seja classificada, levando em conta os resultados do processo de treinamento.

Os classificadores em questão estão dispostos em forma de cascata, no sentido de que são formados por uma série de classificadores mais simples. Esta cascata constitui uma árvore de decisão, ou seja, para um objeto ser aceito, ele deverá ser testado por todos estes classificadores mais simples. Os classificadores verificam cada parte da imagem e podem ser redimensionados para encontrarem objetos de diferentes tamanhos.

Na implementação do OpenCV, a cascata de classificadores usada permite apenas a detecção de faces que estejam na posição frontal. Durante o processo de detecção, o ponto central da face é localizado, e uma circunferência vermelha é desenhada ao seu redor, exibindo a face encontrada. Este código foi integrado ao Avator, sofrendo algumas modificações para adequá-lo ao resto do programa.

4.7.4 Movimentação dos olhos

Para que o avatar pudesse seguir o usuário com os olhos, dois requisitos eram necessários:

1. Saber a localização exata da face com relação à tela, cujas dimensões eram conhecidas.
   Isto era possível uma vez que o algoritmo de visão encontrava o ponto central da face.

2. Ter o controle total dos olhos do modelo, podendo movê-los com diferentes intensidades nos eixos vertical e horizontal.
   Isto também era viável no novo módulo de animação.

Com estes requisitos cumpridos, foi possível implementar um algoritmo bastante simples.

Com relação a movimentação dos olhos no eixo horizontal, imagina-se um limiar vertical separando a tela em duas porções. Se o centro da face encontrada estiver na porção esquerda da tela, o avatar deve mover os olhos para esta direção. Caso contrário, o movimento dos olhos é dado para a direita.

A influência destas poses deve ser proporcional à distância do usuário com relação ao limiar central, sendo este valor normalizado para pertencer ao intervalo $[0, 1]$.

No eixo vertical, o raciocínio é análogo, traçando um limiar horizontal e movendo os olhos para cima ou para baixo.

Segue outra explicação sobre a movimentação, mais detalhada:

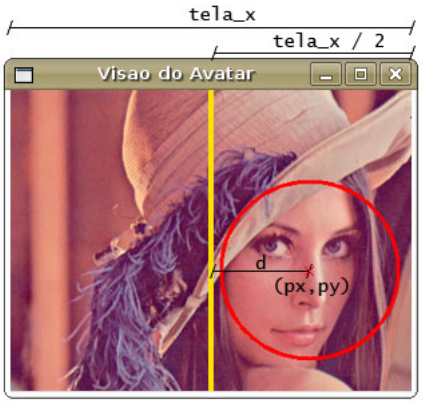


Figura 16: Esquema apontando as informações necessárias para o movimento dos olhos

Na figura acima, sejam:

$tela\_x$: a dimensão da tela no eixo x
$tela\_x/2$: a posição do limiar central
px: a coordenada em x do ponto central da face encontrada
d: a distância da face até o limiar central, isto é: $|px - tela\_x/2|$.

Então, a cada momento, os olhos devem mover-se para direita, ou para esquerda, com influência $d/tela\_x$.

Como a detecção acontece em todos os frames, não é necessário interpolar manualmente estes movimentos.

Evidentemente, o avatar consegue acompanhar apenas uma face por vez. O algoritmo de detecção, portanto, foi alterado para que encontrasse apenas uma face.

Caso o usuário não disponha de uma webcam, o programa pode receber, por parâmetro, um comando para não utilizar nenhuma câmera. Neste caso, nenhuma face é detectada, e o movimento dos olhos é dado aleatoriamente, como nos movimentos não-verbais da face.

4.8 Inteligência Artificial

4.8.1 AIML

O AIML é uma linguagem que define o comportamento de *chatterbots*, agentes computacionais que simulam a conversação com humanos. Um documento na linguagem contém dois elementos principais: *padrões* e *templates*. Os padrões consistem no conjunto de entradas possíveis que podem ser fornecidas pelo usuário. Um template corresponde à resposta para cada padrão. Por exemplo:

```
<pattern>QUAL O SEU NOME?</pattern>
<template>Meu nome é Ana.</template>
```

Além destes elementos, a linguagem pode possuir expressões regulares, elementos condicionais, retornar aleatoriamente um template dentro de um conjunto, dentre outros refinamentos[28].

4.8.2 Pesquisa

Na versão anterior do Avator, como foi mencionado, a IA do avatar foi tratada pelo programa J-Alice, uma implementação de AIML em C++. No entanto, este código estava diretamente relacionado com o Haptek, possuíndo influência do MFC, de modo que não poderia ser facilmente reutilizado neste projeto.

Como o J-Alice é um programa antigo, que não é mais mantido desde 2006, outras alternativas foram buscadas. No entanto, a pesquisa não retornou nenhum resultado satisfatório. A maioria dos programas implementados em C++ eram muito antigos, outros não rodavam em Linux. Alguns possuíam instalação complexa ou demorada, requisitando que outras bibliotecas fossem instaladas. Além disso, todos possuíam pouca, ou nenhuma, documentação. Desta forma, não foi possível completar o módulo de IA durante este projeto.

Para uma próxima versão do Avator, uma pesquisa mais detalhada pode ser feita, e outras idéias poderiam ser consideradas, como, por exemplo, utilizar outros meios para a construção de chatterbots ou implementar um tradutor próprio para AIML.

4.9 Arquitetura do Sistema

Como foi mencionado no início deste documento, a versão anterior do Avator era pouco modularizada. Neste projeto, portanto, buscou-se criar classes constituíndo módulos bem definidos, tratando tarefas específicas.

A seguir, é exibido um esquema simplificado da arquitetura do sistema e uma breve descrição dos módulos principais.

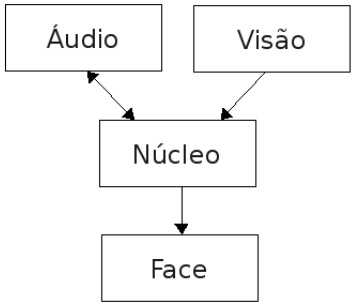


Figura 17: Esquema simplificado da arquitetura do sistema

O **núcleo** é responsável por carregar os recursos do programa, manipular o frame listener, criar a cena e os widgets. Além disso, gerencia a animação da face, sincronizando o áudio com os lábios do modelo e a visão da câmera com os olhos.

O módulo de **Áudio** recebe e sintetiza a frase a ser falada, e retorna o fonema atual para que o programa possa mapeá-lo para o visema correspondente.

O módulo de **Visão** retorna a informação se o usuário está diante da câmera e sua posição na tela.

O módulo da **Face** encapsula o modelo tridmensional, formado por submalhas animadas, e fornece comandos para exibir uma pose, alterando os vértices destes componentes.

Estes módulos utilizam programas específicos, mas podem ser substituídos, e desde que desempenhem os mesmos papéis, podem ser reintegrados ao sistema de maneira simples, como foi proposto.

5 Resultados

Como foi mencionado na última seção, não houve tempo para implementar um módulo de IA para o programa, de modo que a face apenas repete as frases e exibe as expressões faciais requisitadas pelo usuário.

Apesar disso, o projeto cumpriu com quase todas as tarefas propostas. O sistema de animação foi terminado, possuíndo um modelo realista e tendo êxito em seu maior desafio, a sincronização do movimentos dos lábios. Além disso, houve tempo para implementar o movimento dos olhos de acordo com um novo algoritmo, mais robusto, de Visão Computacional.

No entanto, também houve um ponto negativo: o projeto não pôde rodar no Windows, devido a algumas restrições na biblioteca do eSpeak nesta plataforma. Embora o eSpeak seja multi-plataforma, sua API não permite a reprodução do áudio, devido à implementação de threads no Windows, sendo possível apenas extrair os fonemas de que devem ser falados.

Os resultados serão exibidos na forma de *screenshots* do programa e de uma série de vídeos que estão disponíveis na página to trabalho.[29]

5.1 Resultados em Modelos e Animação

5.1.1 Expressões Faciais

Além das seis expressões puras propostas por Ekman, é possível gerar um conjunto de expressões derivadas.

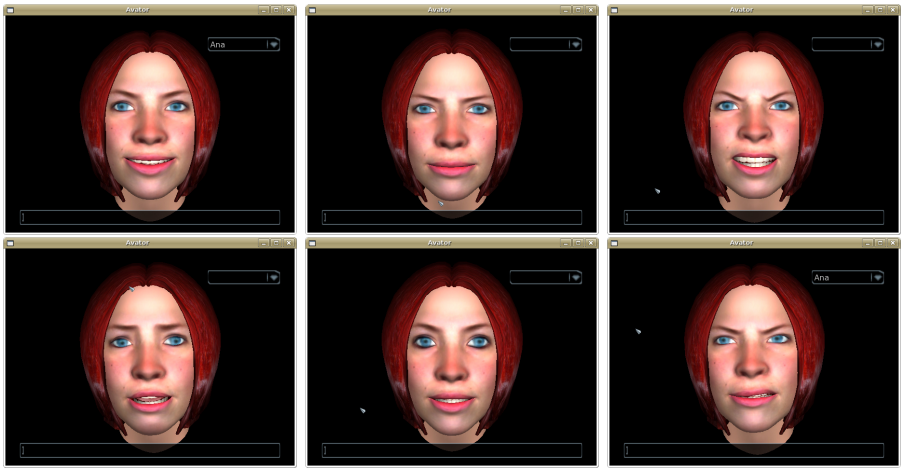


Figura 18: Expressões puras que podem ser exibidas

(a) Frustração: dada pela combinação das expressões de nojo e surpresa

(b) Sarcasmo: dada pela combinação das expressões de nojo e alegria

(c) Preocupação: dada pela combinação das expressões de tristeza e surpresa

Figura 19: Algumas das expressões derivadas que podem ser exibidas. Outras combinações são também possíveis

5.1.2 Modelos

Três modelos são disponibilizados no programa. O modelo exibido anteriormente, com diferentes expressões, é o da Ana, um dos avatares. Além dela, existe um modelo masculino (Caio) e outro modelo feminino (Helena).

Estes modelos podem ser alternados através da *combo box* no canto direito da tela. Os três modelos executam as mesmas ações e possuem vozes únicas.

(a) Caio (b) Helena

Figura 20: Modelos alternativos disponibilizados no programa

5.1.3 Movimentos Não-Verbais

Ao longo do programa, a face não permanece estática, o que confere maior realismo à animação.

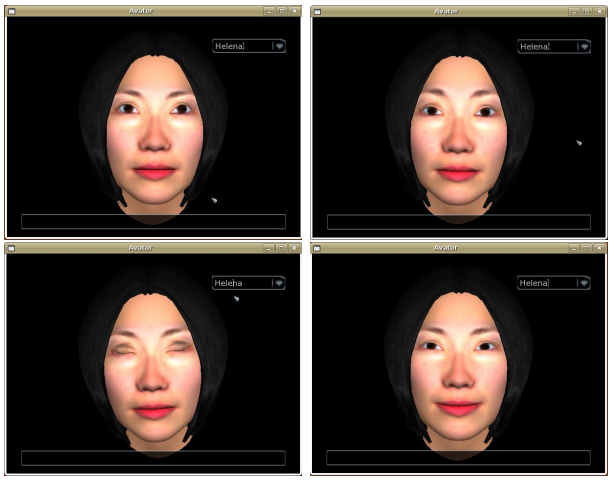
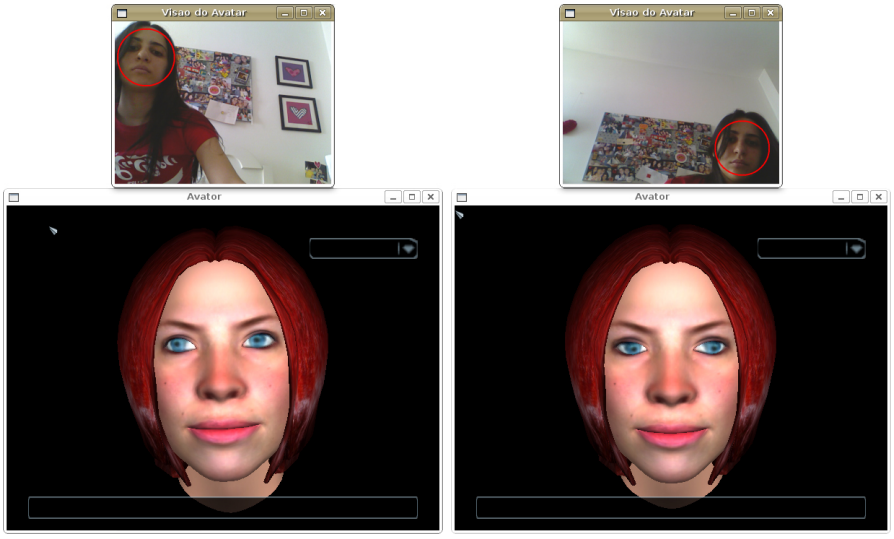


Figura 21: Movimentos não verbais da face ao longo do programa

5.2 Resultados em Visão Computacional

As telas seguintes mostram o avatar acompanhando, com os olhos, a face do usuário.



(a) Olhos movendo-se para o canto esquerdo superior da tela

(b) Olhos movendo-se para o canto direito inferior da tela

Figura 22: Visão Computacional

Assim como na versão anterior do Avator, a face exige a presença de um interlocutor na conversa. Neste caso, ela apenas pronuncia as palavras comandadas quando a face do usuário é detectada pela câmera. Caso contrário, ela pergunta onde ele está.

6 Conclusão

Uma desvantagem deste projeto com relação à versão anterior é o fato da qualidade de som do eSpeak ser um pouco inferior à da SAPI. No entanto, em termos gerais, o projeto pôde trazer boas mudanças.

6.1 Colaboração do projeto

Abaixo, segue um resumo de como o projeto contribuiu para o Avator, ressaltando as diferenças entre a versão anterior e a versão deste ano.

6.1.1 Em termos de sistema

- **IDE:**

  2007: Dependente do Visual Studio e MFC.

  2008: Não depende de nenhuma IDE específica.

- **Instalação:**

  2007: Além de instalar os programas e a voz, era preciso configurar manualmente o projeto do Visual Studio para encontrar as bibliotecas usadas.

  2008: Ainda é preciso instalar programas adicionais, mas o projeto pode ser compilado de maneira simples, usando o comando *make*.

- **Tamanho:**

  2007: O arquivo de voz usado era muito grande, tendo cerca de 40MB.

  2008: O pacote com o projeto completo, incluíndo todos os modelos, tem menos de 7MB.

- **Engenharia de software:**

  2007: O código não possuía comentários, era pouco modularizado e pouco legível, devido à integração com o Haptek Player.

  2008: Durante o desenvolvimento do projeto, documentação, modularização e legibilidade foram priorizados.

- **Sistema Operacional:**

  2007: Windows.

  2008: Linux.

  Todas as bibliotecas usadas são compatíveis também com o MAC OS, mas o programa não foi testado nesta plataforma, não havendo garantia sobre o seu funcionamento.

6.1.2 Em termos de funcionalidades extras

- **Expressões Facias:**

  2007: A face poderia exibir as expressões facias de raiva e tristeza, além da face neutra.

  2008: Existe a possibilidade de gerar um número muito maior de expressões faciais (as seis propostas por Ekman, e suas combinações).

- **Visão Computacional:**

  2007: O avatar apenas falava quando houvesse a detecção de movimento em frente à câmera.

  2008: O avatar só fala quando há a detecção de uma face. Além disso foi adicionada a funcionalidade de movimentar os olhos do avatar de acordo com a posição do usuário em frente a câmera.

- **Modelos:**

  2007: Uma única face estava disponível, com a voz feminina que era instalada.

  2008: São disponibilizados mais modelos de faces, com diferentes vozes, de ambos os genêros.

6.2 Futuro do Avator

Além das duas tarefas que faltaram ser implementadas, existe ainda uma série de funcionalidades que podem ser adicionadas ao projeto:

- Implementar a Inteligência Artifical do avatar.
- Permitir o diálogo com não apenas dois, mas vários avatares ao mesmo tempo.
- Adicionar mais idiomas. A síntese pode ser feita facilmente com o eSpeak, restaria mapear para visemas os novos fonemas que não estejam no português.
- Tratar mais problemas em Visão Computacional:
  - Extrair mais informações sobre o usuário.
  - Reconhecer e mapear expressões faciais.
  - Acompanhar o movimento de partes do rosto.
- Adicionar um módulo de Reconhecimento de Áudio, dispensando o uso do teclado, adicionando assim uma espécie de sentido auditivo para o avatar.

- Fazer o programa rodar em máquinas diferentes, permitindo, por exemplo, um comportamento de chat.
- Com relação ao módulo de animação, usar outros modelos além dos do FaceGen:
  - Usar modelos obtidos por scanner 3D ou a partir de fotos, que não possuem uma correspondência entre os vértices das diferentes poses, como ocorre no FaceGen.
  - Usar modelos genéricos que não possuam as poses especificadas, gerando-as de alguma forma a partir de suas malhas neutras, baseando-se nas poses do FaceGen.
- Tornar o avator compatível com o Windows:
  - Estudar outras alternativas para a síntese de voz, ou permitir a reprodução de áudio no eSpeak.
  - Incluir o tratamento de threads.

Além das tarefas sugeridas acima, muitas outras podem ser pensadas. O que se espera é que este projeto tenha cumprido o seu objetivo de estimular futuros desenvolvedores para o projeto.

7 Parte Subjetiva

Esta parte do documento possui meu relato pessoal de como foi desenvolvido este Trabalho de Conclusão, listando as discliplinas que serviram como base para a sua realização e de como pretendo continuá-lo.

Gostaria de aproveitar este espaço para agradecer ao Roberto por ter se empolgado com este projeto junto comigo, e principalmente, por toda a atenção, dedicação e ajuda durante o trabalho.

7.1 Desafios e Frustrações

Durante este projeto, houveram muitos desafios e incertezas, e felizmente, poucas frustrações.

Escolher a área na qual iria desenvolver o trabalho foi uma tarefa que não precisou de nenhuma reflexão: Computação Gráfica e Visão Computacional sempre foram os tópicos que mais me interessaram em computação, sobretudo o primeiro. No entanto, escolher o tema do projeto foi uma tarefa bem mais difícil, justamente por haverem muitos projetos interessantes nas duas áreas.

Como não tinha nenhum trabalho em mente, precisei conversar com alguns profesores do departamento. Ao falar com o Roberto, porém, tive a certeza de que o Avator era o projeto que mais me interessava dado que ele unia as duas áreas de que gostava tanto.

Fiquei bastante animada com o trabalho e comecei a estudar sobre as versões anteriores do programa. Porém, logo cedo, uma série de problemas passaram a me incomodar bastante. A instalação do Avator era maçante, e até então, o programa só rodava na versão 6.0 do Visual Studio, bastante antiga. Tentei, por algum tempo, compilar o programa no Visual Studio 8.0, até descobrir que esta tarefa era inviável na versão gratuita que estava utilizando.

Além disso, o código que envolvia o Haptek era ilegível e confuso, sendo bastante difícil aterar esta parte do código. Embora não houvesse documentação, a Flávia e o Marcos sempre se mostraram bastante solícitos as vezes que os procurei. Ainda assim, comecei a pensar que não valia a pena continuar o projeto do jeito que estava, não pelo trabalho dos dois, que considero muito bom, mas sim por todas as restrições impostas pelo Haptek.

Pesquisei, por algum tempo, alguma alternativa para a substituição do Haptek, não tendo encontrado nada adequado. Neste ponto, surgiu minha maior dúvida e meu grande dilema neste projeto: continuar o Avator como estava ou implementar um sistema próprio.

A segunda opção me parecia ser a melhor para o projeto, no entanto, tinha medo de que essa tarefa não pudesse ser terminada, ou ainda, que não houvesse tempo para integrar o módulo com o Avator. Conversando com o meu orientador, decidimos que eu iria desevolver o sistema de animação ainda assim, e focar nesta tarefa. A integração, caso não houvesse tempo, ficaria para uma próxima versão.

Neste período, precisei pesquisar bastante sobre como poderia desenvolver um sistema de animação e síntese de voz. Tive que baixar e estudar uma série de programas de modelagem, buscando alguma alternativa gratuita. Pesquisei também sobre as alternativas de engine gráfica disponíveis. A tarefa de integrar um sistema de TTS no programa e sincronizar o texto falado com o movimento dos lábios parecia bastante complicada, e levei bastante tempo até decidir como seria feita.

Tendo optado pelas ferramentas que seriam usadas, houveram uma série de desafios. Aprender a manipular os programas do Ogre, entendendo o carregamento de recursos e fluxo do programa através de Frame Listeners foi uma tarefa complicada.

No entanto, o primeiro maior desafio foi a importação dos modelos do FaceGen para o Ogre. Quando a primeira alternativa buscada - a de fundir os modelos em softwares de modelagem - não funcionou, tive medo de que o projeto não poderia seguir em frente. Felizmente, tive a idéia de modificar diretamente os modelos mesh.xml. Apesar das dificuldades, que funcionou como o esperadodevido à complexidade dos arquivos, a idéia funcionou como o esperado.

O segundo maior desafio do projeto foi entender a API do eSpeak e usá-la para a sincronização labial. Como disse, não sabia como a sincronização seria feita, e teria ficado satisfeita em conseguir um resultado razoável, isto é, que não fosse muito grosseiro. No entanto, acabei ficando feliz com o produto obtido. Embora existam restrições devido ao fato de haver um conjunto fixo de visemas, acredito que a reprodução de algumas palavras ficou bastante fiel.

Fiquei contente em poder terminar o módulo de animação e ainda ter tido tempo para cuidar da Visão Computacional. Não foi muito difícil implementá-la, e acredito que o resultado ficou interessante.

Infelizmente, a parte da IA não pode ser incluída. No entanto, minha grande frustração foi não poder rodar o programa no Windows, tornando-o multiplataforma, como gostaria tanto. De qualquer forma, estou bastante satisfeita com o trabalho. Acredito que tomei uma boa decisão, que o módulo de animação pode trazer boas vantagens para o Avator, além disso, desenvolvê-lo foi uma ótima experiência.

7.2 Disciplinas relevantes para o trabalho

Muitas disciplinas, ao longo do curso, contribuíram bastante para a minha formação, aprimorando também o meu raciocínio lógico. Para este trabalho, especificamente, algumas puderam ser mais aplicadas. Estas disciplinas estão listadas a seguir.

- **MAC 110 - Introdução à Computação,**
  MAC 122 - Princípios de Desenvolvimento de Algoritmos
  Nestas matérias, tive meu primeiro contato com programação, tendo servido como uma ótima base para todos os projetos que desenvolvi ao longo do curso.

- **MAC 323 - Estruturas de Dados**
  Assim como nas disciplinas anteriores, acredito que esta matéria também fornece conceitos básicos que são indispensáveis. Estudar sobre estruturas de dados é fundamental para saber a melhor forma de armazenar e manipular as informações dentro de um programa.

- **MAC 211 - Laboratório de Programação I**
  Foi a primeira matéria na qual desenvolvemos um projeto grande, utilizando interface gráfica. Nesta disciplina, aprendi a importância em documentar todas as etapas de um projeto, utilizando, pela primeira vez, o LaTeX, que considero uma ferramenta excelente. Além disso, aprendi como é fundamental modularizar um código extenso.

- **MAC 242 - Laboratório de Programação II**
  Assim como na disciplina anterior, foi implementado um projeto grande, desta vez utilizando Java e Perl, e colocando em prática, novamente, os conceitos aprendidos. A familiaridade com Perl contribuiu neste projeto para o desenvolvimento do script de conversão dos modelos, implementado nesta linguagem.

- **MAC 332 - Engenharia de Software**
  Para mim, esta é uma das matérias mais importantes do curso. Acredito que os fundamentos ensinados, com respeito às boas práticas de programação e metodologias de projeto devem ser seguidos em qualquer trabalho. Pensando no desenvolvimento futuro deste projeto, que pode ser dado também por outras pessoas, procurei aplicar os conceitos aprendidos na matéria ao longo de todo o processo.

- **MAC 441 - Programação Orientada a Objetos**
  Como o sistema foi implementado em C++, conceitos básicos da matéria foram obrigatoriamente usados, como por exemplo, herança, encapsulamento, *design patterns*, dentre outros.

- **MAC 438 - Programação Concorrente**
  Embora, a princípio, não estivesse planejando utilizar concorrência no programa, algumas threads tiveram que ser usadas para o tratamento do áudio e vídeo do sistema. Assim, esta matéria acabou sendo fundamental, devido aos conceitos ensinados e exercícios desenvolvidos utilizando *pthreads*.

- **MAC 420 - Introdução à Computação Gráfica**
  Os conceitos aprendidos sobre modelos geométricos tridimensionais e construções de cenas em 3D foram amplamente usados neste TCC. Conceitos mais específicos da matéria, no que diz respeito à manipulação de câmera, modelos de iluminação, mapeamento de texturas, etc, foram tratados em boa parte pelo Ogre, de forma que não precisei aplicá-los de maneira tão direta. No entanto, ainda assim, acredito que foi fundamental conhecê-los durante o curso.

- **MAC 417 - Visão e Processamento de Imagens**
  Embora não tenhamos desenvolvido nenhum projeto lidando diretamente com processamento de vídeo, a disciplina serviu para me motivar na área de Visão Computacional. Comecei a estudar alguns programas que usavam o OpenCV, podendo conhecer um pouco sobre a biblioteca, o que me ajudou no desenvolvimento do módulo de Visão do projeto.

7.3 Futuro

Este trabalho pôde desenvolver ainda mais o meu gosto por Computação Gráfica e Visão Computacional. Certamente, foi uma das causas que mais contribuiu para que eu tomasse a decisão de continuar estes estudos no mestrado. Espero ingressar no programa deste Instituto para poder dar continuidade ao sistema, ter a oportunidade de estudar mais sobre a área e participar de mais projetos que possam envolvê-la.

A princípio, não pretendo seguir a carreira acadêmica, mas certamente gostaria de trabalhar na área. Por isso, acredito que o mestrado seja uma forma fundamental de aprimorar meus conhecimentos sobre este tema.

Referências

[1] PARKE, F. I.. Animation of faces. Proc ACM Annual Conference, vol 1, 1972.

[2] PARKE, F. I.; WATERS, K.. Computer Facial Animation. A K Peters, Natick, MA, USA, 1996.

[3] PANDZIC, I. S.; FORCHHEIMER, R.. MPEG-4 Facial Animation: The Standard, Implementation and Applications, John Wiley and Sons, Inc., New York, NY, 2003.

[4] MAGNENAT-THALMANN, N.; THALMANN, D.. Handbook of Virtual Humans, John Wiley and Sons, 2004.

[5] KÄHLER, K.; HABER, J.; YAMAUCHI, H.; SEIDEL, H.. Head shop: Generating animated head models with anatomical structure, Proceedings ACM SIGGRAPH Symposium on Computer Animation (SCA), 2002.

[6] MAGNENAT-THALMANN, N.; THALMANN, D.. Computer Animation: Theory and Practice, Springer-Verlag Nova York, Inc., Secaucus, NJ, 1990.

[7] CHAI, J.; XIAO, J.; HODGINS, J. K.. Vision-based Control of 3D Facial Animation, ACM SIGGRAPH/Eurographics Symposium on Computer Animation, 2003.

[8] EKMAN, P.. All Emotions are Basic. Oxford University Press, Nova York, 1994.

[9] PLUTCHIK, R.. A general psychoevolutionary theory of emotion. New York: Academic, 1980.

[10] RODRIGUES, P. S. L.. Um Sistema de Geração de Expressões Faciais Dinâmicas em Animações Faciais 3D com Processamento de Fala, 2007.

[11] PIDHIN, F.; KECKER, J.; LISCHINSKI, D.; SZELISKI, R.; SALESIN, D. H.. Synthesizing Realistic Facial Expressions From Photographs, SIGGRAPH 98 Proceedings, Orlando, FL, 1998.

[12] DE MARTINO, J. M.. Animação facial sincronizada com a fala: visemas dependentes do contexto fonético para o português do Brasil, 2005.

[13] Página do aluno Marcos Moreti

`http://www.linux.ime.usp.br/~cef/mac499-06/monografias/mpmoreti/`

[14] Página da aluna Flávia Ost

`http://www.vision.ime.usp.br/~fost/`

[15] Haptek

http://www.haptek.com/

[16] J-Alice

http://j-alice.sourceforge.net/

[17] AIML - Artificial Intelligence Markup Language

http://www.alicebot.org/aiml.html

[18] OpenCV - Open Source Computer Vision Library

http://www.intel.com/technology/computing/opencv/

[19] Projeto AlterEgo do Pendulum Studios

http://www.studiopendulum.com/alterego/

[20] Image Metrics: Performance-Driven Facial Animation Solutions for the Digital World

http://www.image-metrics.com/

[21] FaceGen

http://www.facegen.com/

[22] MakeHuman

http://www.makehuman.org/blog/index.php

[23] DAZ 3D Poser Modeling Company

http://www.daz3d.com/

[24] OGRE 3D - Open source graphics engine

http://www.ogre3d.org/

[25] eSpeak: Speech Synthesizer

http://espeak.sourceforge.net/

[26] Especificação do padrão MPEG-4

http://www.chiariglione.org/mpeg/standards/mpeg-4/mpeg-4.htm

[27] Especificação do sistema Kirshenbaum

http://www.kirshenbaum.net/IPA/ascii-ipa.pdf

[28] Especificação da linguagem AIML

http://docs.aitools.org/aiml/spec/

[29] Página do projeto, contendo os vídeos de resultados e o código-fonte do programa

http://www.linux.ime.usp.br/~dedea/mac499/